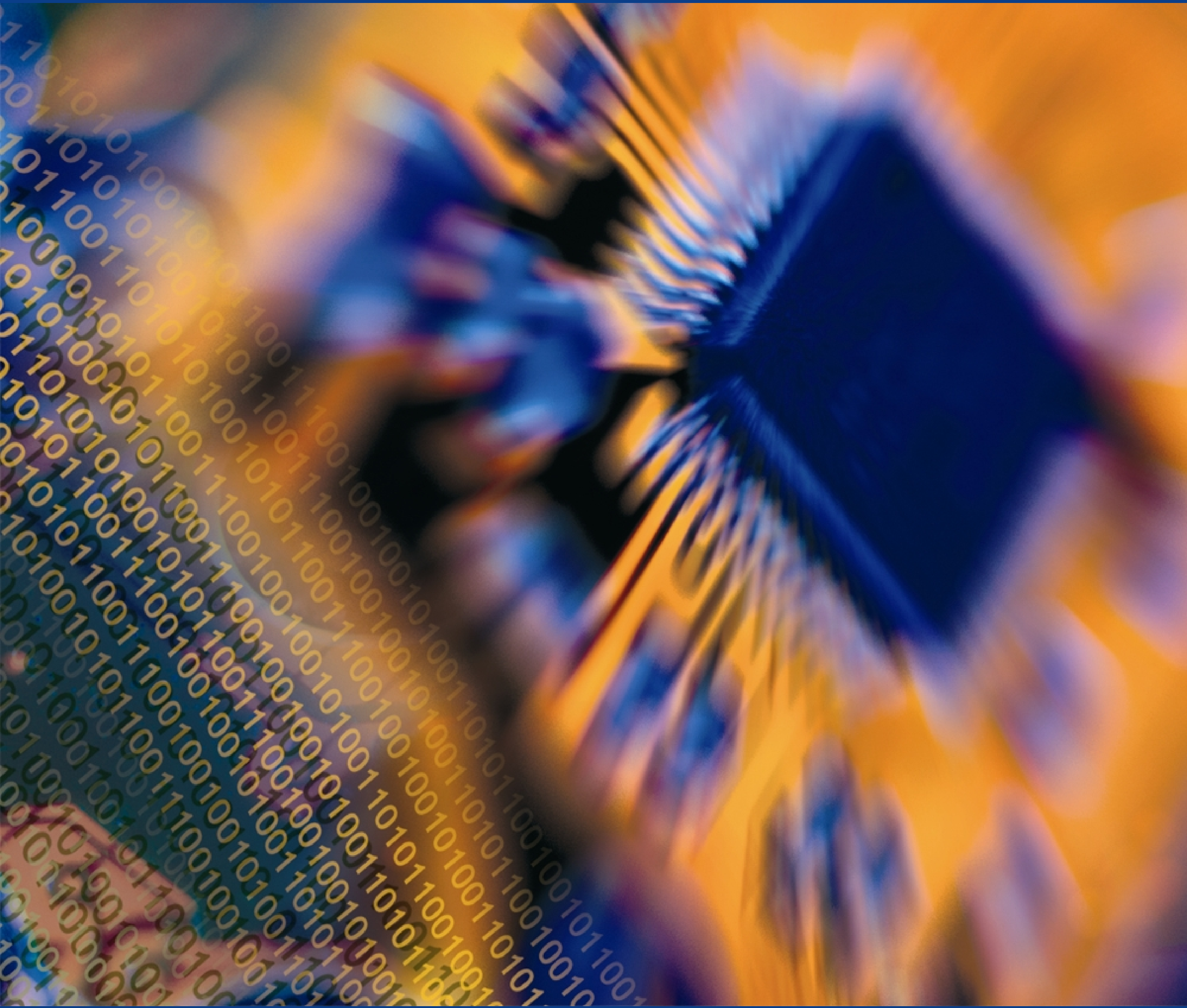
EURASIP Journal on Embedded Systems

Field-Programmable Gate Arrays in Embedded Systems

Guest Editors: Miriam Leeser, Scott Hauck, and Russell Tessier



EURASIP Journal on Embedded Systems

Field-Programmable Gate Arrays in Embedded Systems

EURASIP Journal on Embedded Systems

Field-Programmable Gate Arrays in Embedded Systems

Guest Editors: Miriam Leeser, Scott Hauck,
and Russell Tessier

Editor-in-Chief

Associate Editors

Contents

Field-Programmable Gate Arrays in Embedded Systems, Miriam Leeser, Scott Hauck, and Russell Tessier
Volume 2006 (2006), Article ID 51312, 2 pages

An Overview of Reconfigurable Hardware in Embedded Systems, Philip Garcia, Katherine Compton, Michael Schulte, Emily Blem, and Wenyin Fu
Volume 2006 (2006), Article ID 56320, 19 pages

Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC, Ari Kulmala, Olli Lehtoranta, Timo D. Hämäläinen, and Marko Hännikäinen
Volume 2006 (2006), Article ID 38494, 15 pages

A Real-Time Wavelet-Domain Video Denoising Implementation in FPGA, Mihajlo Katona, Aleksandra Pižurica, Nikola Teslić, Vladimir Kovačević, and Wilfried Philips
Volume 2006 (2006), Article ID 16035, 12 pages

A Dynamic Reconfigurable Hardware/Software Architecture for Object Tracking in Video Streams, Felix Mühlbauer and Christophe Bobda
Volume 2006 (2006), Article ID 82564, 8 pages

Speech Silicon: An FPGA Architecture for Real-Time Hidden Markov-Model-Based Speech Recognition, Jeffrey Schuster, Kshitij Gupta, Raymond Hoare, and Alex K. Jones
Volume 2006 (2006), Article ID 48085, 19 pages

A Visual Environment for Real-Time Image Processing in Hardware (VERTIPH), C. T. Johnston, D. G. Bailey, and P. Lyons
Volume 2006 (2006), Article ID 72962, 8 pages

FPGA-Based Communications Receivers for Smart Antenna Array Embedded Systems, Constantin Siriteanu, Steven D. Blostein, and James Millar
Volume 2006 (2006), Article ID 81309, 13 pages

Modeling and Design of Fault-Tolerant and Self-Adaptive Reconfigurable Networked Embedded Systems, Thilo Streichert, Dirk Koch, Christian Haubelt, and Jürgen Teich
Volume 2006 (2006), Article ID 42168, 15 pages

MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution, Riad Ben Mouhoub and Omar Hammami
Volume 2006 (2006), Article ID 54074, 14 pages

Rapid Energy Estimation for Hardware-Software Codesign Using FPGAs, Jingzhao Ou and Viktor K. Prasanna
Volume 2006 (2006), Article ID 98045, 11 pages

FPGA Dynamic Power Minimization through Placement and Routing Constraints, Li Wang, Matthew French, Azadeh Davoodi, and Deepak Agarwal
Volume 2006 (2006), Article ID 31605, 10 pages

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 51312, Pages 1–2
DOI 10.1155/ES/2006/51312

Editorial
Field-Programmable Gate Arrays in Embedded Systems

Miriam Leeser,[1] Scott Hauck,[2] and Russell Tessier[3]

[1] *Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA*
[2] *Department of Electrical Engineering, University of Washington, Seattle, WA 98195-2500, USA*
[3] *Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA*

Received 13 July 2006; Accepted 13 July 2006

Welcome to the special issue on field programmable gate arrays (FPGAs). FPGAs are becoming an increasingly important part of embedded systems, as the collection of papers in this issue illustrates.

"An overview of reconfigurable hardware in embedded systems" provides a comprehensive overview of the state-of-the-art use of reconfigurable hardware in embedded systems. A detailed discussion of the use of FPGAs for application areas such as encryption, software-defined radio, and robotics is provided. Additionally, a concise assessment of design issues and current design tools is included. A sizable collection of citations provides a handy reference for newcomers to the field.

The remaining papers address applications and tools for embedded systems design. The applications presented here are typical of the spectrum of FPGA applications. They fall into the categories of multimedia processing, including video, image and speech processing, as well as communications applications.

The implementation of an MPEG-4 image encoder using a scalable number of Altera NIOS soft processors is presented in "Scalable MPEG-4 encoder of FPGA multiprocessor SOC." An image is partitioned so that each processor receives a horizontal slice of the image. The author's own on-chip interconnection network is used to connect the soft processors. The authors demonstrate a significant application speedup as additional soft processors are added to the FPGA platform.

In "A real-time wavelet domain video denoising implementation in FPGA," the authors present a two-FPGA solution for performing video denoising via a 3D (two spatial and one temporal dimension) wavelet filter. By careful consideration of the algorithm, data movement, and pipelining, a complete and complex image processing pipeline is produced.

In "A dynamic reconfigurable hardware/software architecture for object tracking in video streams," the authors present a feature tracker that has been implemented on an FPGA. The authors focus on choosing an algorithm that is well matched to reconfigurable hardware, hardware/software partitioning, and efficient use of memory structures. Their implementation, which runs faster than a software-only solution, has applications for mobile autonomous platforms.

The paper "Speech silicon: an FPGA architecture for real-time hidden Markov model-based speech recognition" details the implementation of an FPGA SoC that can perform real-time speech recognition of medium-sized speech vocabularies. This pipelined approach maximizes the throughput by minimizing the amount of required control circuitry. The FPGA implementation of each part of the pipeline is carefully documented to demonstrate the benefits of FPGA specialization. FPGA floorplanning plays an important role in achieving real-time performance.

A common application for FPGAs is image processing algorithms. In "A visual environment for real-time image processing in hardware (VERTIPH)," the authors propose a new tool for designing image processing implementations on FPGAs. The proposed tool aims to improve the productivity of designers targeting FPGAs for their image processing algorithms, and provides visual information for the timing and structure of the implementation.

In "FPGA-based communications receivers for smart antenna array embedded systems," the authors consider the design of adaptive receivers on FPGAs. The receivers can support an array of antennas, and make use of adaptive algorithms to change parameters depending on the environment. This approach is particularly good at reducing the power required to receive signals.

An interesting aspect of embedded systems using FPGAs is the use of both CPUs and reconfigurable logic in the same system; and two papers in this issue address tools for supporting hardware/software codesign.

The first paper is “Modeling and design of fault-tolerant and self-adaptive reconfigurable networked embedded systems.” In traditional mixed hardware/software systems, the designer picks which resources will support which tasks. In this paper, the authors explore a different approach—dynamic movement between these resources. This paper describes a framework for implementing a fault-tolerant system containing FPGAs and processors. Tasks can be dynamically bound to hardware or software, and support for checkpointing and rollback is provided.

“MOCDEX: multiprocessor on-chip multiobjective design space exploration with direct execution” supports the design of multiprocessor systems on a chip. The processors here are MicroBlaze soft-cores on a Xilinx Virtex chip. Four are used to implement an image filtering application. The contribution of this paper is in the use of a multiobjective evolutionary algorithm to optimize the design of each processor. The optimization criteria chosen are number of logic slices, amount of block RAM and number of cycles. Real FPGA implementations are used in the evaluation phase of the algorithm.

Another important aspect of tools for embedded systems is energy estimation and power minimization. Two papers in this issue address this problem. “Rapid energy estimation for hardware-software codesign using FPGAs” outlines the design and implementation of a high-level energy estimation approach for combined hardware/software designs mapped to FPGAs. The FPGA design includes both a soft processor and custom application hardware. Cosimulation of the hardware and software is performed to determine software instruction usage and hardware switching activities. This information is then used by low-level instruction-level and hardware models to estimate energy consumption. A 6000x speedup in energy estimation time is achieved versus synthesis-based approaches with a loss of about 10% energy estimation accuracy.

Power consumption in an embedded system is often the crucial design constraint. Although research efforts have developed CAD algorithms to replace vendor tools to perform power optimization, real designers are still reliant on the vendor’s tools. The paper “FPGA dynamic power minimization through placement and routing constraints” takes a different track, showing that by carefully devising placement constraints, power reductions in a Xilinx FPGA are possible within the vendor’s tool suite. A number of schemes for devising these placement constraints are considered, and overall achieve approximately a 10% power savings.

This collection of papers represents a good overview of active research in the field of reconfigurable hardware in embedded systems. We hope you enjoy this special issue.

Miriam Leeser
Scott Hauck
Russell Tessier

Miriam Leeser is a Professor at Northeastern University, Department of Electrical and Computer Engineering. She received her B.S. degree in electrical engineering from Cornell University, and Diploma and Ph.D. degrees in computer science from Cambridge University in England. After completion of her Ph.D., she joined the faculty of Cornell University, Department of Electrical Engineering, as an Assistant Professor. In January, 1996 she joined the faculty of Northeastern University, where she is the Head of the Reconfigurable Computing Laboratory and a Member of the Computer Engineering Research Group and the Center for Communications and Digital Signal Processing. In 1992 she received an NSF Young Investigator Award to conduct research into floating-point arithmetic. Her research interests include hardware description languages, high-level synthesis, computer arithmetic, and reconfigurable computing for signal and image processing applications. She is a Senior Member of the IEEE, and a Member of the ACM.

Scott Hauck received the B.S. degree in computer science from the University of California, Berkeley, in 1990, and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Washington, Seattle, in 1992 and 1995, respectively. He is an Associate Professor of electrical engineering at the University of Washington. From 1995 to 1999, he was an Assistant Professor at Northwestern University. His research concentrates on FPGAs, including architectures, applications, and CAD tools, reconfigurable computing, and FPGA-based encryption and image compression. He has received a National Science Foundation (NSF) Career Award, a Sloan Fellowship, and a TVLSI Best Paper Award. He is a Senior Member of the IEEE.

Russell Tessier is an Associate Professor of electrical and computer engineering at the University of Massachusetts, Amherst, Mass. He received the B.S. degree in computer engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1989 and S.M. and Ph.D. degrees in electrical engineering from MIT, Cambridge, Mass, in 1992 and 1999, respectively. He is a Founder of Virtual Machine Works, a logic emulation company, and has also worked at BBN, Ikos Systems, and Altera. Professor Tessier currently leads the Reconfigurable Computing Group at UMass. His research interests include computer architecture, field-programmable gate arrays, and system verification.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 56320, Pages 1–19
DOI 10.1155/ES/2006/56320

An Overview of Reconfigurable Hardware in Embedded Systems

Philip Garcia, Katherine Compton, Michael Schulte, Emily Blem, and Wenyin Fu

Department of Electrical and Computer Engineering, University of Wisconsin-Madison, WI 53706-1691, USA

Received 5 January 2006; Revised 7 June 2006; Accepted 19 June 2006

Over the past few years, the realm of embedded systems has expanded to include a wide variety of products, ranging from digital cameras, to sensor networks, to medical imaging systems. Consequently, engineers strive to create ever smaller and faster products, many of which have stringent power requirements. Coupled with increasing pressure to decrease costs and time-to-market, the design constraints of embedded systems pose a serious challenge to embedded systems designers. Reconfigurable hardware can provide a flexible and efficient platform for satisfying the area, performance, cost, and power requirements of many embedded systems. This article presents an overview of reconfigurable computing in embedded systems, in terms of benefits it can provide, how it has already been used, design issues, and hurdles that have slowed its adoption.

1. WHY USE RECONFIGURABLE HARDWARE IN EMBEDDED SYSTEMS?

Reconfigurable hardware (RH) provides a flexible medium to implement hardware circuits. The RH resources are configurable (and generally reconfigurable) post-fabrication, allowing a single-base hardware design to implement a variety of circuits. The hardware itself is composed of a set of logic and routing resources controlled by configuration memory. This memory is frequently implemented as SRAM cells, though flash RAM and other technologies are also possible. (Some FPGAs employ anti-fuses as a configuration medium [1, 2]. However, because these devices are essentially one-time programmable, they are not reconfigurable, and are thus not the focus of this article.) These memory cells (and their stored values in particular) affect the functionality of both routing and logic. In the routing architecture, a cell may control whether or not two wires are electrically connected, or provide a multiplexer select input. In logic, the cell may control the function of an ALU, or implement logic equations in the form of a lookup table (LUT), which is the most common logic resource in field-programmable gate arrays (FPGAs).

Essentially, circuits are decomposed into small subfunctions implemented in LUTs or other logic resources in the RH, and the routing resources are configured to electrically connect the logic resources to match the structure of the target circuit. Writing a new set of values into the configuration, memory reconfigures the hardware to implement a different circuit. Complex RH designs may also contain communication structures and processor cores that may or may not be reconfigurable.

Embedded systems often have stringent performance and power requirements, leading designers to incorporate special-purpose hardware into their designs. Hardware-based implementations avoid the instruction fetch/decode/execute overhead of traditional software execution, and use resources spatially to increase parallelism. In many embedded applications, such as multimedia, encryption, wireless communication, and others, highly repetitive parallel computations well-suited to hardware implementation represent a significant fraction of the overall computation required by the system [3, 4].

Unfortunately, application-specific integrated circuit (ASIC) implementation is not feasible or desirable for all circuits. One key problem is that the non-recurring engineering costs (NREs) of ASICs have been increasing dramatically. A mask set for an ASIC in the 90 nm process cost about \$1M [5]. Previously, using FPGAs as ASIC substitutes was only cost-effective in low-volume applications. FPGAs have high per-unit costs, which are essentially an amortization of the FPGA NREs themselves over all customers for those chips. However, as ASIC NREs rise and FPGAs sell in higher volumes, the ASIC NREs begin to outweigh the per-unit cost of FPGAs for higher-volume applications, shifting the balance towards FPGAs [6]. Especially considering the flexibility

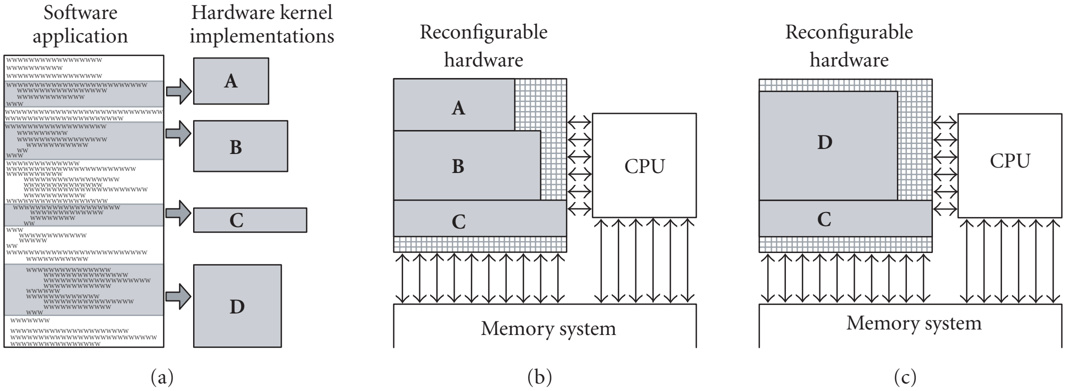


Figure 1: Reconfigurable computing implements compute-intensive application kernels (a) as hardware in RH and the remaining code in software on a CPU (b). Run-time reconfiguration allows RH to implement circuits that would otherwise not fit simultaneously (c).

of RH to accommodate new circuitry for bugfixes, protocol updates, or new advances, expensive and fixed-design ASIC technology becomes less appealing.

Furthermore, devices traditionally categorized as embedded systems, such as PDAs (personal digital assistants) and cellular phones, are becoming increasingly multipurpose. These systems may implement a very diverse set of applications that require the performance and power benefits of hardware implementation, such as wireless communications, cryptography, and digital audio/video. Including a fixed custom hardware accelerator for each possible application type is generally infeasible, particularly if one or more of the applications is not known at designtime. RH can act as a "general" hardware accelerator, implementing a variety of different computations within or across applications. Compute-intensive sections of applications can be swapped into the hardware when needed, and later swapped out to make room for other computations, a process called reconfigurable computing. Figure 1 illustrates a case where, after computations **A** and **B** are complete in hardware, they can be replaced with computation **D**—potentially while computation **C** is still running. In effect, run-time reconfiguration allows RH to act as a virtual hardware accelerator, with capacities and capabilities beyond its actual physical structure.

Low-power operation is critical to many embedded systems to improve battery life, reduce costs of operation, and even improve reliability [7]. Computations implemented in RH often dissipate less power than equivalent software running on embedded processors, since they typically can be implemented at lower clock rates and avoid the overhead associated with fetching, decoding, issuing, and committing individual instructions [8–12]. However, they also often have higher power dissipation than fixed ASIC solutions [10, 13].

Finally, the flexibility of RH can also be used to increase the fault-tolerance of designs. RH can be reconfigured to avoid hardware faults [14], whether they result from fabrication or the environment. If the fault is from fabrication, this increases product yield, decreasing costs. If the fault develops after deployment, this allows a faulty device to potentially continue normal operation. The new configuration can even be deployed remotely [14, 15] to avoid inconveniencing the consumer or allow updates for a device that cannot be physically accessed (systems deployed in space, on the ocean floor, or at other remote or unsafe locations). Extra reconfigurable logic in a design can also allow a system to compensate if a fault occurs in a nonreconfigurable resource [16]. The fault-tolerance of RH can even extend to design faults, allowing bug fixes or even upgrades for emerging standards to increase device lifespan. Fault-tolerance advantages and techniques are discussed in greater depth in Section 4.2.

This article discusses the benefits and issues of employing RH in embedded systems designs. Section 2 lists a variety of applications implemented in embedded systems with RH. Section 3 discusses basic architectural aspects, and describes several example systems. Other design issues critical to many embedded systems are discussed in Section 4. Section 5 addresses configuration overhead, and Section 6 discusses design tools. Future issues in reconfigurable embedded computing are discussed in Section 7 For more specific technical information on RH and reconfigurable computing, as well as their use outside of embedded systems, please refer to one or more of the following surveys: [10, 17–22].

2. WHAT APPLICATIONS BENEFIT FROM RH?

Initially, smaller reconfigurable devices such as PLDs and PALs were used as board level glue logic. Similarly, RH can now be used as chip-level glue logic on systems-on-a-chip (SoCs) [23]. In particular, RH can act as a flexible communication fabric for different cores on the SoC [24–26]. This allows hardware design to proceed even if the intercomponent communication methods have not yet been finalized. This approach also improves time-to-market and design costs because the testing of a single reconfigurable communication fabric is faster and less costly than the testing of separate communications fabrics for many different SoC designs. Furthermore, the configurable communication fabric can potentially be reconfigured if necessary to circumvent design errors in other SoC components [23, 27].

RH can also perform computations in a capacity beyond simple ASIC replacement. By reconfiguring the hardware at runtime, one or more RH structure can be reused for many different computations over time (Figure 1) [10, 20–22]. Since many embedded systems must be both high-performance and low-power, yet may also have size or flexibility constraints preventing fixed-ASIC implementation, RH provides a valuable implementation method. Furthermore, computational cores used in many applications are available as predesigned intellectual property (IP), simplifying the design process.

Software-defined radio

Telecommunications industries employ constantly evolving wireless technologies. Companies under significant pressure to deliver products before their competitors sometimes even release products before standards are finalized. Software-defined radios (SDR) are programmable to implement a variety of wireless protocols, potentially even those not yet introduced [28–35]. Custom hardware allows many embedded systems to meet stringent power and performance requirements, particularly for small battery-powered mobile devices, but in this case the system must also be extremely flexible. A system with RH can implement parallel DSP operations with a higher degree of both performance and power efficiency than a software-only system, plus an RH system can be reconfigured for different protocols as needed.

Medical imaging

Recently, several RH-based systems and algorithms have been proposed for medical imaging [36, 37]. The ECAT HRRT PET scanner from CTI PET Systems, Inc. [36] detects abnormalities in organ systems, helping to find cancerous tumors and assisting in monitoring ongoing patient treatment. This system can dynamically reconfigure itself for setup, detection, and equipment self-diagnosis modes. One project implementing a parallel-beam backprojection for medical computer tomography on RH was able to accelerate the application 100x over a 1 GHz Pentium by implementing a custom design in RH and performing a thorough bit-precision analysis [37]. This system also scales well with additional hardware (4x more hardware leads to 4x better performance).

Networking

RH is commonly used in network processors [38–42] which have high performance demands and inherently parallel workloads. Furthermore, networks can use many different routing protocols, and different system administrators may have varying needs at different times. RH has been used in network devices to run tasks such as packet classification [38], dynamic routing protocols [39, 40], and intrusion detection systems [42] among others. RH can also accommodate emerging network protocols through reconfiguration.

Encryption

Many encryption algorithms are well-suited to hardware implementation. Operations are generally highly parallel and repetitive, with the same series of operations performed on each piece of data. Furthermore, these algorithms frequently use exclusive-or operations, which do not require the area and delay overhead of a complete ALU. As encryption research continues to evolve, RH can be reconfigured to implement new standards. For these reasons, encryption algorithms are a popular choice for RH implementation [9, 43, 44].

Scientific data acquisition and analysis

Scientific data-acquisition systems receive and preprocess vast quantities of data before archiving or sending the data off for further processing. These systems may be remote or inaccessible, operating on battery or solar power, yet requiring extremely high performance to handle the required volume of data. These systems are increasingly using RH to provide this performance in a flexible medium that can be changed as new approaches to data aggregation and preprocessing are researched. RH has been used in systems proposed or created for weather radar [45], seismic exploration [46], and adaptive cameras for solar study [47]. RH is also used to compress the massive volume of data prior to transmission [48].

Spacecraft

RH's low-volume costeffectiveness and hardware flexibility make it particularly applicable to space applications, where it has been used for several missions, including Mars Pathfinder and Surveyor [49, 50]. These devices can be reconfigured to add functionality for updated mission objectives or fix design errors without requiring a space mission for repair. Spacecraft require special radiation-hardened devices that are not produced in the same volume (due to higher cost and lower demand) as standard microchips, leading designers to incorporate the functionality of many different discrete components into one or a few radiation-hardened FPGAs. Fault-tolerance issues are discussed in more depth in Section 4.2. More experimental research examines the use of genetic algorithms to design evolvable RH that can automatically adapt to needed tasks [51].

Robotics

Robotic control systems often consist of a mix of hardware and software solutions to meet strict size and power demands. One military system prototype uses RH to control unmanned aerial vehicles [46]. These vehicles cannot support large payloads, and must execute heavy-duty image processing algorithms. Other research focuses more generally on developing algorithms and hardware cores for robotic control and vision [46, 52, 53]. An overview of RH in robotic applications appears in [53].

Automotive

The automotive industry has embraced RH because it can implement the functionality of many different parts, reducing repair inventories. Its programmable nature also simplifies product recalls. Furthermore, FPGAs are well-suited to the increasingly complex informational and entertainment systems in newer automobiles [54, 55]. IP companies such as Drivven provide cores for many engine control systems (such as fuel injection) required by modern automobiles [56], which can be implemented in one of several FPGAs rated for automotive use.

Image and video

Digital cameras often need to implement many different image-processing operations that must operate quickly without consuming much battery power. With RH, the hardware can be reconfigured to implement whichever operation is needed [57, 58]. For systems requiring secure image transmission, the RH can also be reconfigured to perform encryption and network interfaces [57]. Some systems can also be configured to accelerate image display [57, 58], video playback [35, 59], and 3D rendering [59–61].

3. WHAT DO THESE SYSTEMS LOOK LIKE?

This section discusses the RH design and system-level integration, examining different design aspects and how they relate to embedded systems design. These topics are covered more generally in several FPGA and reconfigurable computing survey articles [10, 17–22]. Finally, the end of this section presents several specific embedded systems with RH.

3.1. Reconfigurable logic

Although commercial RH tends to contain LUT-based or sum-of-products compute structures, these are not necessarily ideal for many embedded systems. Each configuration point in these structures contributes some level of area, delay, and power overhead, and significant flexibility of these structures may not be required if computations are limited to a particular domain. In these cases, a more specialized reconfigurable fabric can provide the necessary level of flexibility with lower overhead than a fine-grained bit-level logic structure [62–66]. However, some applications, including certain encryption algorithms, cyclic redundancy check, Reed-Solomon encoders/decoders, and convolution encoders, do require bit-level manipulations. A number of reconfigurable architectures combine fine- and coarse-grained compute structures to accommodate both computation styles [67–69]. Most frequently this involves embedding coarse-grained structures, such as multipliers and memory blocks, into a conventional fine-grained fabric [70], or designing the fine-grained fabric specifically to support coarse-grained computations [63, 71].

To implement a needed circuit in RH, a CAD flow transforms its descriptions into an RH configuration. First, the circuit is synthesized, converting the circuit schematic or hardware design language (HDL) description into a structural circuit netlist. Then a technology mapper further decomposes that netlist into components matching the capabilities of the RH's basic blocks (LUTs, ALUs, etc.). Next, the placer determines which netlist components should be assigned to which physical hardware blocks, and a router decides how to best use the RH's routing fabric to connect those blocks to form the needed circuit. Finally, the CAD flow determines the specific binary values to load into the configuration bits for the determined implementation. More details on generic CAD issues for RH can be found elsewhere [21, 72].

Like fixed hardware design, the CAD flow can target different area/delay/power tradeoffs through resource selection, resource sharing, pipelining, loop unrolling, wordlength optimization, precision estimation, and others [73–81]. CAD issues particularly applicable to embedded systems, however, include heterogenous CAD topics [82–84], CAD tools for nonsquare RH designs incorporated into SoCs [25], power-aware CAD [84–91] (discussed further in Section 4.1), and fast CAD algorithms [92–97]. Fast CAD algorithms can move configurations to new locations on RH at run-time or make small modifications to circuits based on run-time conditions to increase efficiency [98, 99], based on available resources [75], or potentially to provide fault-tolerance.

3.2. System-level integration

Embedded systems typically couple a traditional processor (the "host") with custom hardware specifically to handle compute-intensive highly-parallel sections of application code [100]. The processor controls the hardware, and executes the parts of applications not well-suited to hardware. Reconfigurable computing systems also frequently couple RH with a processor, for the same reasons as well as to control the configuration processor of the RH [10, 20–22, 101]. RH-processor coupling styles can be divided into three basic categories: RH as a functional unit on the processor data path, RH as a coprocessor, and RH as an attached processor in a heterogeneous multiprocessor system. The coupling methods are best differentiated by how and how often the RH and host processors(s) interact.

Reconfigurable functional units (RFUs) are very tightly coupled with a host processor. Input and output data are generally read from and written to the processor's register file [66, 71, 102–106]. These units essentially provide new instructions to an otherwise fixed instruction set architecture (ISA). In some cases, the processor itself may be implemented on reconfigurable logic, allowing significant processor customization [106, 107]. In Section 6.2 we will examine some of the design tools that help simplify the process of creating these custom-ISA processors.

If the circuits on the RH can operate for some time independently of the host processor, a coprocessor or even heterogeneous multiprocessor coupling may be more appropriate [3, 4, 108–112]. A coprocessor may or may not share the data cache of the host processor but generally shares the main memory. Figure 1 shows an example of a reconfigurable coprocessor that has its own path to a shared memory structure. A heterogeneous multiprocessor may contain one or more reconfigurable units, one or more embedded or general purpose processors, and possibly other special-purpose processing elements [33, 109, 113]. Like homogenous multiprocessor systems, heterogeneous multiprocessors may use shared memory for communication between compute nodes [24], a communication bus, or even a network architecture [113]. Synchronization and scheduling issues of these systems are similar to those of homogenous multiprocessors.

In some cases, using one or more separate FPGA chips (plus the other system circuitry) would violate the area, performance, or power constraints of the embedded system. However, FPGA capacities are always increasing, so to address this problem, designers can now use platform FPGAs or systems on programmable chips (SoPCs), which are large and complex enough to contain entire SoC designs, and frequently include fixed communication structures and other commonly-needed circuitry [67–69, 114]. Alternately, reconfigurable logic can be embedded within an SoC [62, 64, 115, 116] to implement one or more computations. This provides for domain-specific SoCs that can be customized to the actual application(s) needed by programming the reconfigurable logic appropriately. Domain-specific SoCs therefore provide higher performance and lower power consumption than a traditional FPGA structure, with some parts of the hardware implemented as standard cells or even full custom. The RH itself can even be customized to the applications needed [117]. Domain-specific SoCs facilitate highly efficient embedded systems, but with NREs that are amortized over all applications within the domain [118].

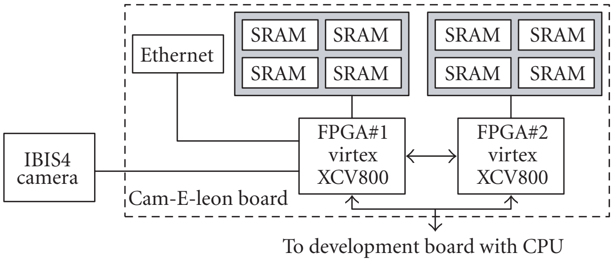


Figure 2: Cam-E-leon is a dynamically reconfigurable web camera platform from IMEC [57].

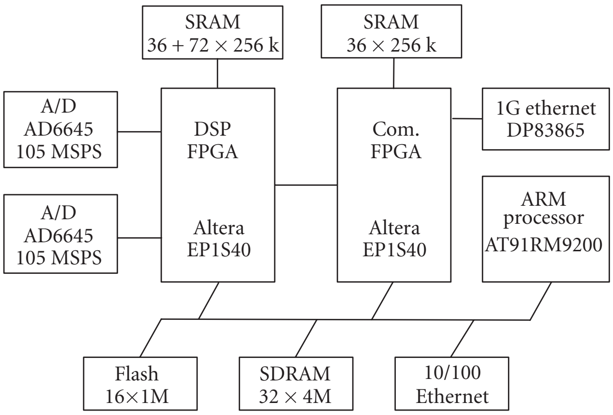


Figure 3: Block diagram of CASA: an embedded radar-based hazardous weather detection system using RH [45].

3.3. Example systems

Embedded systems with RH span a range of sizes and complexities, some using many discrete RH components, with others primarily contained in an SoPC. Many of these systems use Linux or a modified lighter-weight Linux as an operating system because the source code is freely available for recompilation to the custom platform. This section presents the high-level design details of a number of systems to provide a flavor of the range of systems using RH. However, this list is by no means exhaustive, as there are a great many interesting RH-based embedded systems.

One large system was designed for 3D vision [60]. This system contains an image acquisition board connected to a matrix of 36 Xilinx XC4005 FPGAs used for low-level image processing (such as edge detection and edge tracking). Images preprocessed by the FPGAs are then sent to a board containing 16 DSPs for high-level image processing. This board also contains four more FPGAs used to create a reconfigurable interconnection network between the DSP chips.

Cam-E-leon (Figure 2) is another image-related embedded system, designed in particular as a dynamic web camera [57]. This system is capable of downloading new image processing algorithms from a networked server and incorporating them into the system, implemented in RH. However, it is significantly smaller than the 3D vision system, using a custom FPGA board with two Xilinx Virtex XCV800 FPGAs. The FPGA board is responsible for the image processing computations. A processor board running a Linux variant is responsible for network communication and reconfiguring the FPGAs. The camera itself is a 1.3 megapixel image sensor, directly connected to the FPGA containing the camera interface. This FPGA is also responsible for image processing, while the other FPGA encrypts the image for secure transmission. All circuitry would normally have fit in one of the two FPGAs, but bandwidth concerns necessitated design partitioning between two chips.

CASA is a weather radar data acquisition and processing system used to detect hazardous conditions [45]. A block diagram is given in Figure 3. Like Cam-E-leon [57], one of the two FPGAs in CASA is dedicated to signal processing (the left FPGA in both figures), and can be updated with new functionality remotely by a networked server. In CASA, the other FPGA is responsible for communication of result data, but may also process data depending on the configuration. An ARM-based microcontroller running Linux manages the FPGA resources. CASA also contains multibanked memory, multiple Ethernet interfaces, and analog-to-digital (A/D) converters to digitize incoming radar data. CASA can process data at sustained rates of 88.3 Mb/s.

The Linux-based SDR application described in [35] uses a single Xilinx Virtex-4 FX FPGA, in conjunction with an analog RF card, memory, and an output device (frame buffer and audio). The FPGA contains two hard embedded

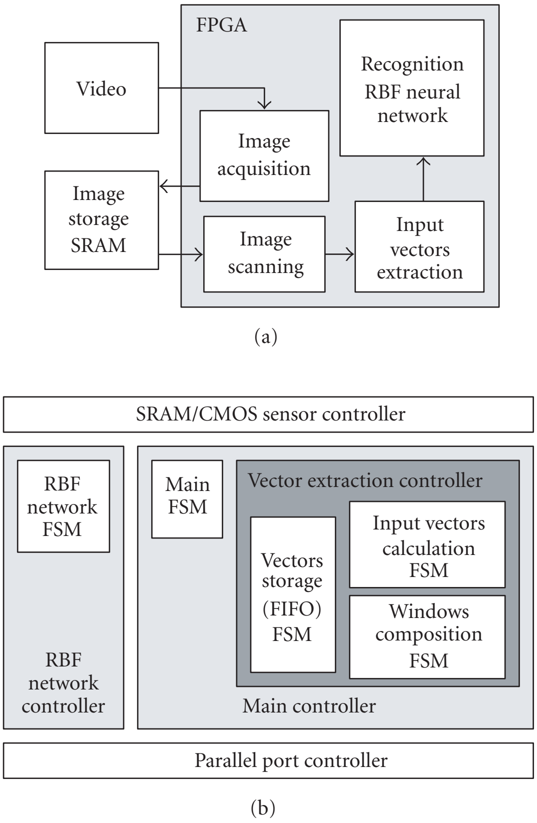


Figure 4: Block-level diagrams of the system-level design (a) and the FPGA design details (b) of a facial-recognition system [119].

PowerPC cores, and several soft-core components: a demodulation core, a memory controller, and an IDCT. The analog board receives the data over a wireless network and sends it to the first processor. The first processor, coupled with the demodulation core, processes the data and writes it to main memory. The second CPU then decodes the data from memory using the IDCT core, and the resulting video and audio stream is then written to the output device. A Linux-based reconfigurable encryption processor system also uses embedded PowerPC devices, but instead in a Virtex-II Pro [44]. In this system, the RH contains a memory controller, a bus bridge to communicate with the on-chip peripheral bus (OPB), which in turn connects to an Ethernet controller, a UART, the cryptographic engine itself, and control logic to manage the reconfiguration of the cryptographic engine. The on-chip PowerPC core communicates with these structures using the built-in processor local bus (PLB). This system can be reconfigured to implement different encryption algorithms.

One project compared several systems implementing a face tracking algorithm, including a Xilinx Spartan-II 300 FPGA-based system, a custom ASIC-based hardware system, and a software-based DSP implementation [119]. The FPGA implementation is shown in Figure 4, including a system-level block diagram (a) and details of the FPGA design (b). The FPGA contains multiple interfacing controllers for the sensors, the parallel port, and the network, and also implements a 15-node radial basis function (RBF) neural network to detect faces and recognize facial expressions. The custom hardware system also used an FPGA, but as glue logic, not a compute engine. As typically expected when comparing ASIC, FPGA, and software implementations, the software implementation had the lowest throughput (one-fifth of the ASIC), and the custom hardware had the highest. The FPGA implementation had half the throughput of the ASIC version. However, the recognition rates were higher for the more flexible solutions, with the programmable DSP achieving the highest, demonstrating a throughput/accuracy trade-off. Both the FPGA and DSP implementations also have the benefit that they can be modified post-deployment to implement new algorithms.

Several embedded systems use RH as custom functional units on a processor's data path. One example of this system type is a 3D facial recognition program [120] using a Stretch S5 processor [66]. This system beams an invisible light pattern on a user's face, which is then detected by cameras interfaced with the processor. By examining differences in the projected and detected light patterns, the system reconstructs a 3D model of the target face in real time. The system also contains an Ethernet link to allow the data to be sent over a network. The embedded design implemented on a 300 MHz S5 processor matched the performance of a 3 GHz PC by using RH as an application accelerator. However, this application was designed entirely in software and compiled by the Stretch compiler to a mix of software and hardware—a process completed in five person-months. Design tools for this development style are discussed further in Section 6.2.

4. WHAT ARE OTHER IMPORTANT DESIGN ISSUES?

Beside the basic choices of RH logic design and RH integration, low power, fault-tolerance, and real-time issues are also critical to embedded systems designers. Understanding the interaction between these topics and RH is important whether the designer is choosing off-the-shelf components to include in a system, choosing between completed systems, or designing a new RH fabric specifically for a particular embedded system.

4.1. Low power

Many embedded devices are battery powered, increasing the importance of power efficiency. Computations on FPGAs typically consume less power than equivalent software running on embedded processors, but more power than ASICs [10]. Studies examining the data-per-watt efficiency of FPGA-based implementations have found that they can process just under 20*x* more data-per-watt than a RISC-style processor for both the IDEA encryption algorithm [9] and an FIR filter operation [8]. Yet another study shows the use of RH yielding performance increases of 4.3*x* to 13.5*x*, while simultaneously reducing power consumption by up to 93% over a very-long-instruction-word-style (VLIW-style) processor [11]. To further improve RH power-efficiency,

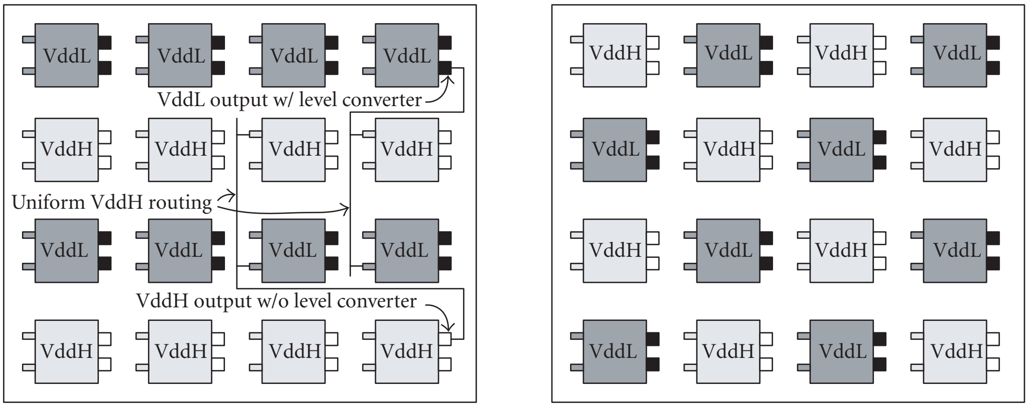

VddL
VddL output w/ level converter
VddH
Uniform VddH routing
VddH output w/o level converter

Figure 5: Two different layout patterns for fixed-distribution dual-Vdd FPGA fabrics [88].

researchers have investigated energy-efficient architectures, the use of multiple supply voltages or threshold voltages, and energy-efficient mapping techniques to implement algorithms on RH.

Several energy-efficient reconfigurable architectures have been specifically developed to reduce power dissipation. The FPGA interconnect and clock networks are responsible for most of the power dissipation in traditional FPGA architectures [121]. One proposed fine-grained FPGA structure improves energy efficiency through a hybrid interconnect structure using nearest-neighbor connections, a symmetric mesh architecture, and hierarchical connectivity to shorten and reduce the number of necessary wires [121]. This FPGA architecture also uses low-voltage circuit swing techniques and dual edge-triggered flip-flops to reduce the power dissipation from clock distribution. MONTIUM is an energy-efficient coarse-grained reconfigurable architecture designed for 16-bit DSP applications [122]. It improves power efficiency by reducing interconnect and configuration overhead, providing access to small, local memories, and optimizing the RH for word-level DSP applications. The MONTIUM reconfigurable processor can implement an adaptive Viterbi algorithm using 200 times less energy than an ARM9 processor [12].

Multiple supply voltages (Vdd) or threshold voltages (Vt) can also improve energy-efficiency in RH. Reducing Vdd decreases dynamic power, while increasing Vt decreases leakage power. Since changes to Vdd and Vt also affect noise margins and circuit speed, appropriate values for Vdd and Vt must be carefully selected. Proposed fabrics with predefined dual-Vdd and dual-Vt fabrics use low-leakage SRAM cells and dual-Vt lookup tables that do not penalize performance, but reduce total power dissipation by 13.6% and 14.1% on average for combinational and sequential circuits, respectively [88]. An example fixed dual-Vdd FPGA layout is given in Figure 5. In dual-Vdd architectures, timing-critical circuit paths are assigned to high-Vdd logic and routing, while the remaining parts of the circuit are assigned to low-Vdd resources. Level converters preserve a signal's value when transitioning between Vdd levels. Programmable dual-Vdd architectures can provide an average power savings of 61% across various Microelectronics Center of North Carolina (MCNC) benchmarks [87]. Multiple-Vt architectures, combined with low-leakage multiplexer and routing structures, gate biasing, and redundant SRAM cells can reduce leakage current by roughly $2X$ to $4X$ over FPGA implementations without any leakage reduction techniques [89]. Finally, many commercial FPGAs contain multiple clock domains to allow designers to clock critical circuit sections at fast rates, and noncritical sections at slower rates, lowering overall power consumption of the design [67–69].

Dual-Vdd and dual-Vt architectures require a CAD flow to choose between fast but power-hungry resources or slower but lower-power resources for circuit components [87–89]. However, CAD algorithms can also affect circuit power-efficiency in existing RH designs. For example, resource selection, module disabling, parallel processing, pipelining, and algorithmic selection together improved energy efficiency of FFT and matrix multiplication algorithms [85]. A dynamic programming-based approach to map beamforming applications on a Xilinx Virtex-II Pro reduces energy dissipation by 52% on average over a greedy algorithm [86]. Considering power implications of embedded memory blocks can reduce embedded memory dynamic power by an average of 21% and overall core dynamic power by an average of 7% [84]. Power information can also be incorporated into cost functions used for existing CAD processes. Adding an FPGA power model [91] and using power-aware algorithms throughout the CAD flow can provide 26.5% power-delay product savings [90].

4.2. *Fault tolerance*

Faults can be divided into two categories: permanent and transient. Fabrication faults and design faults are among the permanent faults. Transient faults, commonly called single event upsets (SEUs), are brief incorrect values resulting from external forces (terrestrial radiation, particles from solar flares, cosmic rays, and radiation from other space phenomena) altering the balance or locations of electrons,

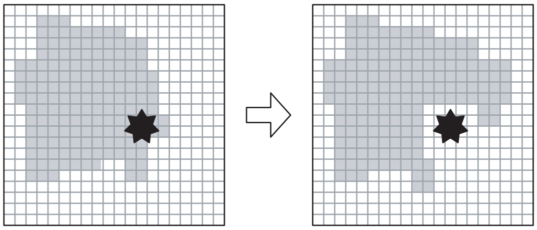


Figure 6: Faults (black) can be overcome by remapping affected configurations (gray) to nonfaulty areas of reconfigurable hardware.

usually in a small area of the system. We discuss both categories of faults as they relate to RH in this section.

Tolerating permanent faults is critical to maximizing device and system yields to decrease costs, and to increasing the lifespan of deployed devices. Lifespan is of particular concern when a system has been deployed to a location difficult, dangerous, or impossible to reach for repair or replacement. Space-deployed unmanned systems, for example, must be extremely fault-tolerant, as replacement/repair would be expensive, and at worst, impossible. RH can increase tolerance of permanent physical faults because the hardware is modifiable to potentially compensate for these faults (from fabrication or other sources) within the RH (Figure 6) [14, 123] or even elsewhere in the system [16]. Yields of "static" FPGA devices (chips used for a single, nonchanging configuration) can be increased by using application-specific test vectors to determine if a particular faulty chip is capable of implementing a particular configuration, allowing designers to successfully use otherwise faulty chips [124, 125]. Finally, design faults are among the easiest to fix in RH, as these devices can be reprogrammed with corrected versions of the faulty circuits.

Unfortunately, although RH's value is in its flexibility, and that flexibility can increase RH's tolerance to permanent faults, it can also increase its underlying susceptibility to faults. The flexibility of RH results from the ability to control its resources based on configuration bit values, frequently stored in SRAM. These SRAM bits, along with any other hardware used to provide flexibility, such as multiplexers, tri-state buffers, and pass transistors, are additional failure points not present in ASIC-equivalent circuit implementations, and increase the chip area to present a larger target to radiation particles. Furthermore, unless the underlying RH design prevents multiple drivers to a wire (instead of relying on the design tools to prevent it), a fault in configuration memory could cause a short-circuit, damaging the device.

Using properly-shielded radiation-hardened devices can minimize SEU errors. Unfortunately, these devices are expensive, difficult to find, and generally use less advanced technologies than their unshielded counterparts [14, 123]. Triple modular redundancy (TMR) can detect and correct faults in circuits implemented in FPGAs [126]. In TMR three copies of all routing and logic resources perform the same computation, and the three "vote" on the correct result. The downsides of this technique include area, power, and performance overheads that are generally unacceptably high for embedded devices, and the fact that TMR cannot accommodate simultaneous errors in multiple copies [14, 127]. Other fault-tolerance techniques focus only on the configuration structure. Scrubbing reads back all of the configuration bits, compares them to the correct values, and re-writes the correct values if a discrepancy is found [127, 128]. Checksums can also be used to detect errors in subsets of configuration information (such as a single logic block), but requires additional resources to store the checksum values in the hardware [127]. Los Alamos has researched methods to decrease SEU-susceptibility of RH destined for spacecraft use [129], with the goal of tolerating and recovering from SEUs without a full system restart. Continuous configuration bit polling, combined with circuit mapping techniques to make SEUs more easily visible allow easier detection of errors in configuration data [129]. Similar work uses an SEU watchdog to reset RH after SEUs in high-radiation environment [130].

Self-testing can also be applied to RH, with the hardware split into multiple self-testing areas (STARs). Periodically, each STAR is isolated from the rest of the system for testing, while the remainder of the system continues operation. Detected faults cause the system to reconfigure the application to avoid the fault without interrupting system function, and partial or entire STAR blocks can be marked as unusable [131]. This approach requires partitioning the hardware to match the STAR structure and ensuring each block is sufficiently computationally independent. Besides testing itself, RH can act as a built-in reconfigurable tester for other parts of the system, particularly for SoC devices [132].

Any fault-tolerance technique will impose additional overhead in terms of area, delay, power, or some combination of the three. One way to reduce this overhead is to apply fault-tolerance techniques selectively within the system. Hardware where faults could cause catastrophic failure (improper levels of anesthesia to be delivered, improper nitrogen/oxygen mix in a pressurized vehicle, etc.) receive the most protection, while hardware where faults cause less critical errors (momentary glitch in an LCD display) receive less. The COFTA project uses an automatic approach to determine where duplicate-and-compare hardware and assertions should be added to provide the same level of fault tolerance as TMR but with 60% less area overhead [133].

4.3. *Real-time support*

Many embedded systems require real-time operation. Generally, there are two types of real-time deadlines: deadlines that must always be met (hard deadlines), and deadlines that must be met the majority of the time (soft deadlines) [134]. Hard deadlines represent tasks critical to system operation, causing system failure if missed. Soft deadlines are used for tasks such as video playback, where as long as the video processing generally keeps up, a few dropped frames are not critical. These requirements shift the focus of the real-time operating system (RTOS) to consider both deadline times and types, and concentrate on optimizing worst-case task execution times instead of average-case times.

In dynamically reconfigurable systems, the RTOS must take into account not only task types, deadlines, and deadline types, but also RH/task resources and task configuration time [135–137]. If multiple tasks reside on the RH simultaneously, the RTOS must also consider their locations in the hardware. Generally, a configuration is tied to specific resources at specific locations on RH. However, to facilitate run-time reconfiguration, partially reconfigurable architectures with relocation allow the locations of the tasks to be moved to accommodate other tasks [137]. Issues related to configuration architectures and reconfiguration management are discussed in Section 5.

An RTOS may use preemptive scheduling of tasks onto RH [138]. For example, a soft-deadline task present on the RH may be removed to make room for a hard-deadline task. These scheduling algorithms offer tradeoffs in terms of overall system utilization and the total number of tasks that can be effectively scheduled. The OVERSOC project [135] investigates the interaction between embedded RTOSs and reconfigurable SoC platforms, and proposes a variety of methods to model reconfigurable fabrics and techniques for scheduling real-time tasks on reconfigurable SoC platforms.

Although using RH to create a real-time system with customized hardware instructions can improve task completion ratios, most tools used to design these instructions [139, 140] focus on reducing *average* application execution time, when in fact worst-case time is generally more important for real-time operation. One custom instruction generator tool designed specifically for real-time systems instead selects subgraphs for custom instruction implementation to minimize worst-case task execution time [141]. Topics related to custom instruction generation for non-real-time systems are discussed in more depth in Section 6.2.

4.4. Design security

High-quality hardware cores for embedded systems are extremely useful to embedded designers, speeding the development process. However, these cores are also time-consuming and expensive to develop and verify. Furthermore, since the hardware designs frequently reside in a configuration bitstream loaded at startup or at runtime into the RH, designs can be intercepted and reverse-engineered. Therefore, design security of this intellectual property (IP) is critical to core developers, leading to encryption of configuration bitstreams [142, 143]. Both Altera and Xilinx have implemented configuration encryption in their commercial products [144, 145].

5. WHAT ABOUT CONFIGURATION OVERHEAD?

Reconfiguring hardware at runtime allows a greater number of computations to be accelerated in hardware than could be otherwise, but introduces configuration overhead as the configuration SRAM must be loaded with new values for each reconfiguration. For separate FPGA chips, this process can take on the order of milliseconds [136], possibly overshadowing the benefits of hardware computation. This section briefly presents both hardware- and software-related aspects of managing the configuration overhead.

A straightforward strategy to reduce configuration overhead is to reduce the amount of data transferred. The structure of the logic/routing itself has an effect: fine-grained devices provide great flexibility through a very large number of configuration points. Coarse-grained architectures by nature require fewer configuration bits because fewer choices are available. The Stretch S5 embedded processor [66], for example, is composed of 4-bit ALU structures. This architecture can be configured in less than 100 microseconds if the configuration data is located in the on-chip cache.

Partially-reconfigurable RH can be selectively programmed [68, 71, 110, 111, 114, 146] instead of forcing the entire device to be reconfigured for any change (a common requirement). However, to be truly effective for run-time reconfigurable computing, the devices must also relocate and defragment configurations to avoid positioning conflicts within the hardware and fragmentation of usable resources [137, 147–149], maintaining intraconfiguration communication and connections to the outside of the RH. A page-based architecture is an alternate form of partially reconfigurable architecture that simplifies communication problems. In a page-based design, identical tiles of reconfigurable resources are connected by a communication bus, and configurations occupy some number of complete pages [150–152]. Pipeline reconfigurable architectures have a similar quality, as each configuration stage may be assigned to any physical pipeline unit [111]. These types of organizations can also be imposed on existing FPGA architectures by dedicating part of the hardware to the required communication infrastructure [150, 153] that simplifies cross-configuration communication. Furthermore, page- or tile-based architectures would be especially useful in a system also requiring fault-tolerance, as the same division used for scheduling could be used for the STARS fault-detection approach discussed in Section 4.2, and faulty pages could be avoided.

Configuration data can also be compressed [154], particularly useful when the RH and the configuration memory are on separate chips. When possible, on-chip configuration memory or a configuration cache can dramatically decrease configuration times [66, 155] due to shorter connections and wider communication paths. Finally, multiple configurations can be stored within the RH at the configuration points in a multicontexted device [156, 157]. These devices have several multiplexed planes of configuration information. Swapping between the loaded configurations involves simply changing which configuration plane is addressed. A key benefit of this approach is background-loading of a configuration while another is active.

Software techniques such as prefetching [158] or scheduling can also reduce configuration overhead by predicting needed configurations and loading them in advance, as well as retaining configurations (in a partially reconfigurable device) that may be needed again in the near future. If the system operation is well-defined and known in advance, temporal partitioning and static scheduling may be sufficient [159, 160]. For other systems, the simplest approach is

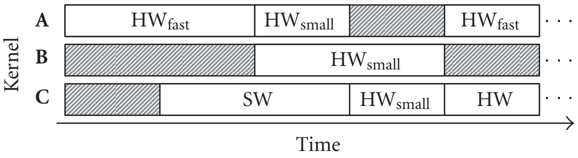


FIGURE 7: Different implementations (fast but large, small but slower, or software) for three kernels (**A**, **B**, and **C**) are shown over time. Shaded areas show when kernels are not needed. In this example, one fast or two small kernels can fit in RH simultaneously.

to load configurations as they are needed, removing one or more configurations from the RH if necessary to free sufficient resources [66, 155, 161, 162].

In more complex systems, compiler- or user-inserted directives can be used to preload the configurations in order to minimize configuration overhead [155], or the configuration schedule can be determined during application compilation [163], dynamically at runtime [137, 153, 164–171], or a combination of the two [152]. Although dynamic scheduling requires some overhead to compute the schedule, this is essential if a variety of applications will execute concurrently on the hardware, breaking the static predictability of the next-needed configuration. Dynamic scheduling also raises the possibility of runtime binding of resources to either the reconfigurable logic or the host processor [168–170], and of choosing between different versions of the computation created in advance or dynamically [75, 99] based on area/speed/power tradeoffs [153, 165, 170, 172] as shown in Figure 7. This could allow an embedded device to run much faster when plugged in, and save power when operating on batteries. To facilitate this scheduling, the RH could be context-switched, saving the current state before loading a new one [66, 173, 174], possibly allowing preemptive scheduling of the resources [137].

6. WHAT TOOLS AID THE RECONFIGURABLE EMBEDDED DESIGNER?

The design of reconfigurable embedded systems, or applications for them, is frequently a complex process. Fortunately, tools can assist the designer in this process, as described in this section.

6.1. Hardware/software codesign

The reconfigurable computing hardware/software (HW/SW) codesign problem is similar to general HW/SW codesign, and in many cases FPGAs are used to demonstrate techniques even if they do not leverage run-time reconfiguration [24, 175, 176]. Design patterns [77] in many cases can apply equally well to general hardware design and hardware design for reconfigurable computing. This section primarily focuses on areas of codesign specific to embedded reconfigurable computing. More information on general HW/SW codesign can be found elsewhere [177–180].

Designers can manually HW/SW partition applications using a combination of profiling and intuition, and develop the components separately for each resource [171]. Alternately, applications can be specified in a more unified form, generally using a high-level language (HLL) such as C or Java [66, 175, 181–183], but in many cases these compilers require code annotations to specify hardware-specific information (custom bitwidths, parallelism, etc.) or only operate on a restricted subset of the language. Some compilers permit parallelism to be specified at the task level using threads [184, 185]. However, compiling hardware from a software-style description can be difficult or inefficient due to the sequential nature of software, and the spatial nature of hardware [186–188]. Some efforts have therefore focused on new ways to express computations that are more agnostic to final implementation in hardware or software, expressing instead the dataflow of the application [151, 189–191]. One aspect of HW/SW codesign unique to RH is temporal partitioning [160, 171, 192, 193], the process of breaking up a single circuit or a series of computations into a set of configurations swapped in and out of the RH over time. Some systems also allow these configurations to be dynamically placed and connected to the other components on RH [162, 194].

Finally, designing an application for an embedded system with RH has the advantage that verification tools can use the RH in conjunction with software simulation and debugging to accelerate the verification process [66, 195–198]. If design errors are found, the RH can be reconfigured with a fixed design because configuration is not a permanent process.

6.2. Processor ISA customization

Backwards-compatibility is generally far less critical to embedded systems than to general-purpose computers. This allows embedded systems designers the freedom to adapt processors' ISAs to changing needs and technologies, and makes custom compilers for such ISAs less of a burden as embedded applications are frequently developed by the same company that develops the hardware (or one of its partners). RH allows the designers to use a single chip design to implement dramatically different ISAs by reprogramming the RH with different functionalities. Multiple design tools are available to automate this process [66, 139, 140, 199, 200]. These tools generally examine precompiled binary instruction streams and generate data flow graphs as candidates for custom instructions. Another approach is to create a compile-time list of potential configurations and their associated binary instruction graph, and at run time detect those graphs in the instruction stream, replacing them with the appropriate RH operations [140].

The SPREE tool [200] is a manual-assist tool that allows a designer to explore processor tradeoffs such as pipeline depth, software versus hardware implementation of components such as multiplication and division, and other design features. The tool also removes unused instructions to save area. Tool chains from Altera and Xilinx focus on SoPC platform design, with parameterizable soft-core processors manually tuned to the respective FPGA architectures, and core generators to create other common computational structures needed on SoPC designs. Developers using Stretch processors write applications in C, profile them, and choose candidate functions for RH to implement in a C variant designed to specify hardware [66, 120]. Finally, for designers wanting to create a fixed-silicon custom processor with a reconfigurable functional unit (instead of a soft-core processor implemented on an FPGA), customizable processors such as Xtensa [201] provide a base processor design and a tool-set for customization. Xtensa is the base of Stretch, Inc. commercially available reconfigurable embedded processors [66].

6.3. Automated RH design

Finally, automatic design tools can aid in the creation of the RH itself [202–204]. The Totem project focuses on the creation of automatic design tools to create coarse-grained domain-specific RH for SoCs based on the intended applications [203]. Other work investigates the use of synthesizable FPGA structures either specifically for embedding in SoCs [23, 202] or tile-based FPGA layout generators usable either in SoCs or as stand-alone architectures [204]. This latter work created architectures in 34 person-weeks instead of 50 person-years, with only a 36% area penalty.

7. WHAT DOES THE FUTURE HOLD?

Reconfigurable hardware faces a number of challenges if it is to become commonplace in embedded systems. First, there is a Catch-22 in that because reconfigurable computing is not a common technique in commercial hardware, it is not yet something that many embedded designers will know to consider. This problem is gradually being overcome with the introduction of reconfigurable computing in certain embedded areas, such as network routers, high-definition video servers, automobiles, wireless base stations, and medical imaging systems. Furthermore, a greater number of people are exposed to reconfigurable hardware as more universities include courses and laboratories using FPGAs. Second, the strict power limitations of many embedded systems highlights the power inefficiency of LUT-based reconfigurable hardware compared to ASIC designs. Because power concerns are intensifying in all areas of computing, research will increasingly focus on power efficiency. Efforts are already underway, with researchers studying a variety of architectural and CAD techniques to improve power dissipation in reconfigurable hardware and computing. Third, the flexibility of reconfigurable hardware that permits the fault tolerance benefits discussed in this article also increases the hardware's susceptibility to faults due to the extra area introduced to support reconfigurability and the use of SRAM-based configuration bits. Innovative reconfigurable architectures, circuit-level design methodologies, and techniques for detecting and avoiding faults are needed to further improve the fault tolerance of reconfigurable hardware.

There are also a number of software-related issues to consider. Compiler support, while improving, is not yet at the level required for widespread adoption of embedded reconfigurable computing. In most cases the computations to be implemented in software and the computations to be implemented in hardware must be specified separately in different languages, and compiled with different toolsets. While some systems and tool suites do offer a more unified flow, these are currently less common. Continued research in effective hardware-software codesign is essential to improve the ease of application design for embedded reconfigurable systems. Furthermore, even though the concept of OS support of reconfigurable hardware was proposed nearly a decade ago, this area remains open.

These challenges are worth addressing, as reconfigurable hardware has many advantages for embedded systems. Implementing compute-intensive applications partially or completely in hardware can dramatically improve system performance and/or decrease system power consumption. The flexibility of the hardware allows a single structure to act as an accelerator for a variety of calculations, saving the area that discrete specialized structures would otherwise require, and allowing new computations to be implemented on the hardware after fabrication. That flexibility can also be used to reduce the design and production cost of embedded system components, as one physical design can be reused for multiple different tasks, amortizing NREs. Finally, reconfigurability provides new opportunities for fault-tolerance, since a design implemented in the reconfigurable hardware can be configured to avoid faulty areas of that hardware. In some cases, the reconfigurable hardware can even be configured to implement the functionality of a faulty component elsewhere in the system. For all of these reasons, reconfigurable hardware is a compelling component for embedded system design.

REFERENCES

[1] J. Greene, E. Hamdy, and S. Beal, "Antifuse field programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1042–1056, 1993.

[2] Actel Corporation, "Programming Antifuse Devices Application Note," Actel, Mountain View, Calif, USA, 2005, http://www.actel.com.

[3] G. Lu, H. Singh, M. Lee, N. Bagherzadeh, F. J. Kurdahi, and E. M. C. Filho, "The morphoSys parallel reconfigurable system," in *Proceedings of 5th International Euro-Par Conference on Parallel Processing (Euro-Par '99)*, pp. 727–734, Toulouse, France, August-September 1999.

[4] G. Kuzmanov, G. Gaydadjiev, and S. Vassiliadis, "The MOLEN processor prototype," in *Proceedings of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 296–299, Napa Valley, Calif, USA, April 2004.

[5] D. Pramanik, H. Kamberian, C. Progler, M. Sanie, and D. Pinto, "Cost effective strategies for ASIC masks," in *Cost and Performance in Integrated Circuit Creation*, vol. 5043 of *Proceedings of SPIE*, pp. 142–152, Santa Clara, Calif, USA, February 2003.

[6] Actel Corporation, "Flash FPGAs in the value-based market white paper," Tech. Rep. 55900021-0, Actel, Mountain View, Calif, USA, 2005, http://www.actel.com.

[7] B. Moyer, "Low-power design for embedded processors," *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1576–1587, 2001.

[8] A. Abnous, K. Seno, Y. Ichikawa, M. Wan, and J. Rabaey, "Evaluation of a low-power reconfigurable DSP architecture," in *Proceedings of the 5th Reconfigurable Architectures Workshop (RAW '98)*, pp. 55–60, Orlando, Fla, USA, March 1998.

[9] O. Mencer, M. Morf, and M. J. Flynn, "Hardware software tri-design of encryption for mobile communication units," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '98)*, vol. 5, pp. 3045–3048, Seattler, Wash, USA, May 1998.

[10] R. Tessier and W. Burleson, "Reconfigurable computing and digital signal processing: a survey," *Journal of VLSI Signal Processing*, vol. 28, no. 1-2, pp. 7–27, 2001.

[11] A. Lodi, M. Toma, and F. Campi, "A pipelined configurable gate array for embedded processors," in *Proceedings of ACM/SIGDA 11th International Symposium on Field-Programmable Gate Arrays (FPGA '03)*, pp. 21–29, Monterey, Calif, USA, February 2003.

[12] G. K. Rauwerda, G. J. M. Smit, and P. M. Heysters, "Implementation of multi-standard wireless communication receivers in a heterogeneous reconfigurable system-on-chip," in *Proceedings of the 16th ProRISC Workshop*, pp. 421–427, Veldhoven, The Netherlands, November 2005.

[13] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proceedings of the ACM/SIGDA 14th International Symposium on Field-Programmable Gate Arrays (FPGA '06)*, pp. 21–30, Monterey, Calif, USA, February 2006.

[14] P. A. Laplante, "Computing requirements for self-repairing space systems," *Journal of Aerospace Computing, Information and Communication*, vol. 2, no. 3, pp. 154–169, 2005.

[15] T. Branca, "How to Add Features and Fix Bugs - Remotely. Here's What You Need to Consider When Designing a Xilinx Online Application," Xilinx, 2001.

[16] C. F. Da Silva and A. M. Tokarnia, "RECASTER: synthesis of fault-tolerant embedded systems based on dynamically reconfigurable FPGAs," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 2003–2008, Santa Fe, NM, USA, April 2004.

[17] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–1029, 1993.

[18] W. H. Mangione-Smith, B. Hutchings, D. Andrews, et al., "Seeking solutions in configurable computing," *IEEE Computer*, vol. 30, no. 12, pp. 38–43, 1997.

[19] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[20] R. Hartenstein, "Trends in reconfigurable logic and reconfigurable computing," in *Proceedings of the 9th IEEE International Conference on Electronics, Circuits, and Systems (ICECS '02)*, pp. 801–808, Dubrovnik, Croatia, September 2002.

[21] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[22] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[23] N. Kafafi, K. Bozman, and S. J. E. Wilton, "Architectures and algorithms for synthesizable embedded programmable logic cores," in *Proceedings of ACM/SIGDA 11th International Symposium on Field-Programmable Gate Arrays (FPGA '03)*, pp. 3–11, Monterey, Calif, USA, February 2003.

[24] M. Luthra, S. Gupta, N. Dutt, R. Gupta, and A. Nicolau, "Interface synthesis using memory mapping for an FPGA platform," in *Proceedings of IEEE 21st International Conference on Computer Design: VLSI in Computers and Processors (ICCD '03)*, pp. 140–145, San Jose, Calif, USA, October 2003.

[25] T. Wong and S. J. E. Wilton, "Placement and routing for non-rectangular embedded programmable logic cores in SoC design," in *IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 65–72, Brisbane, Australia, December 2004.

[26] L. Shannon and P. Chow, "Simplifying the integration of processing elements in computing systems using a programmable controller," in *Proceedings of 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 63–72, Napa Valley, Calif, USA, April 2005.

[27] B. R. Quinton and S. J. E. Wilton, "Post-silicon debug using programmable logic cores," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '05)*, pp. 241–248, Singapore, Republic of Singapore, December 2005.

[28] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk, "Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems," in *Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pp. 205–214, Napa Valley, Calif, USA, April 2000.

[29] C. Dick and F. Harris, "FPGA implementation of an OFDM PHY," in *Proceedings of the 37th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 905–909, Pacific Grove, Calif, USA, November 2003.

[30] B. Mohebbi, E. C. Filho, R. Maestre, M. Davies, and F. J. Kurdahi, "A case study of mapping a software-defined radio (SDR) application on a reconfigurable DSP core," in *Proceedings of 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 103–108, Newport Beach, Calif, USA, October 2003.

[31] K. Sarrigeorgidis and J. M. Rabaey, "Massively parallel wireless reconfigurable processor architecture and programming," in *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS '03)*, pp. 170–177, Nice, France, April 2003.

[32] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, "Implementing an OFDM receiver on the RaPiD reconfigurable architecture," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1436–1448, 2004.

[33] G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit, "Mapping wireless communication algorithms onto a reconfigurable architecture," *Journal of Supercomputing*, vol. 30, no. 3, pp. 263–282, 2004.

[34] A. Rudra, "FPGA-based applications for software radio," *RF Design Magazine*, pp. 24–35, 2004.

[35] P. Ryser, "Software define radio with reconfigurable hardware and software: a framework for a TV broadcast receiver," in *Embedded Systems Conference*, San Francisco, Calif, USA, March 2005, http://www.xilinx.com/products/design_resources/proc_central/resource/proc_central_resources.htm.

[36] Altera Inc., "Altera Devices on the Cutting Edge of Medical Technology," 2000, http://www.altera.com/corporate/cust_successes/customer/cst-CTI_PET.html.

[37] S. Coric, M. Leeser, E. Miller, and M. Trepanier, "Parallel-beam backprojection: an FPGA implementation optimized

for medical imaging," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '02)*, pp. 217–226, Monterey, Calif, USA, February 2002.

[38] A. Johnson and K. Mackenzie, "Pattern matching in reconfigurable logic for packet classification," in *Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '01)*, pp. 126–130, Atlanta, Ga, USA, November 2001.

[39] F. Braun, J. Lockwood, and M. Waldvogel, "Protocol wrappers for layered network packet processing in reconfigurable hardware," *IEEE Micro*, vol. 22, no. 1, pp. 66–74, 2002.

[40] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *Proceedings of the 39th Design Automation Conference*, pp. 343–348, New Orleans, La, USA, June 2002.

[41] Lattice Semiconductor Corporation, "Lattice Orca ORLI10G Datasheet," 2002.

[42] Z. K. Baker and V. K. Prasanna, "A methodology for synthesis of efficient intrusion detection systems on FPGAs," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 135–144, Napa Valley, Calif, USA, April 2004.

[43] F. Crowe, A. Daly, T. Kerins, and W. Marnane, "Single-chip FPGA implementation of a cryptographic co-processor," in *Proceedings of the IEEE International Conference on Field-Programmable Technology*, pp. 279–285, Brisbane, Australia, December 2004.

[44] T. T.-O. Kwok and Y.-K. Kwok, "On the design of a self-reconfigurable SoPC based cryptographic engine," in *Proceedings of 24th International Conference on Distributed Computing Systems Workshops (ICDCS '04)*, pp. 876–881, Tokyo, Japan, March 2004.

[45] R. Khasgiwale, L. Krnan, A. Perinkulam, and R. Tessier, "Reconfigurable data acquisition system for weather radar applications," in *Proceedings of 48th Midwest Symposium on Circuits and Systems (MWSCAS '05)*, pp. 822–825, Cincinnati, Ohio, USA, August 2005.

[46] C. Sanderson and D. Shand, "FPGAs supplant processors and ASICs in advanced imaging applications," *FPGA and Structured ASIC Journal*, 2005, http://www.fpgajournal.com/articles_2005/20050104_nallatech.htm.

[47] T. R. Rimmele, "Recent advances in solar adaptive optics," in *Advancements in Adaptive Optics*, vol. 5490 of *Proceedings of SPIE*, pp. 34–46, Glasgow, Scotland, UK, June 2004.

[48] T. Fry and S. Hauck, "SPIHT image compression on FPGAs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 9, pp. 1138–1147, 2005.

[49] R. O. Reynolds, P. H. Smith, L. S. Bell, and H. U. Keller, "Design of Mars lander cameras for Mars Pathfinder, Mars Surveyor '98 and Mars Surveyor '01," *IEEE Transactions on Instrumentation and Measurement*, vol. 50, no. 1, pp. 63–71, 2001.

[50] M. Kifle, M. Andro, Q. K. Tran, G. Fujikawa, and P. P. Chu, "Toward a dynamically reconfigurable computing and communication system for small spacecraft," in *Proceedings of the 21st International Communication Satellite System Conference & Exhibit (ICSSC '03)*, Yokohama, Japan, April 2003.

[51] A. Stoica, D. Keymeulen, C.-S. Lazaro, W.-T. Li, K. Hayworth, and R. Tawel, "Toward on-board synthesis and adaptation of electronic functions: an evolvable hardware approach," in *Proceedings of IEEE Aerospace Applications Conference*, vol. 2, pp. 351–357, Aspen, Colo, USA, March 1999.

[52] J. W. Weingarten, G. Gruener, and R. Siegwart, "A state-of-the-art 3D sensor for robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '04)*, vol. 3, pp. 2155–2160, Sendai, Japan, September-October 2004.

[53] W. J. MacLean, "An evaluation of the suitability of FPGAs for embedded vision systems," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 3, pp. 131–131, San Diego, Calif, USA, June 2005.

[54] K. Parnell, "You can take it with you: on the road with Xilinx," *Xcell Journal*, no. 43, 2002.

[55] K. Parnell, "The changing face of automotive ECU design," *Xcell Journal*, no. 53, 2005.

[56] Drivven, "Programmable Logic IP Cores for FPGA and CPLD," http://www.drivven.com/ProgrammableLogic-IPCores.htm, 2006.

[57] D. Desmet, P. Avasare, P. Coene, et al., "Design of Cam-E-leon: a run-time reconfigurable web camera," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation (SAMOS '02)*, vol. 2268 of *LNCS*, pp. 274–290, Springer, Berlin, Germany, 2002.

[58] M. Leaser, S. Miller, and H. Yu, "Smart camera based on reconfigurable hardware enables diverse real-time applications," in *Proceedings of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 147–155, Napa Valley, Calif, USA, April 2004.

[59] J.-Y. Mignolet, S. Vernalde, D. Verkest, and R. Lauwereins, "Enabling hardware-software multitasking on a reconfigurable computing platform for networked portable multimedia appliances," in *Proceedings of the International Conference on Engineering Reconfigurable Systems and Algorithms*, pp. 116–122, Las Vegas, Nev, USA, June 2002.

[60] K. M. Hou, E. Yao, X. W. Tu, et al., "A reconfigurable and flexible parallel 3D vision system for a mobile robot," in *Proceedings of Computer Architectures for Machine Perception*, pp. 215–221, New Orleans, La, USA, December 1993.

[61] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, P. F. Curt, and D. W. Prather, "FPGA-based acceleration of the 3D finite-difference time-domain method," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 156–163, Napa Valley, Calif, USA, April 2004.

[62] Elixent, *DFA1000 RISC Accelerator*, Elixent, Bristol, England, 2002.

[63] K. Leijten-Nowak and J. L. Van Meerbergen, "An FPGA architecture with enhanced datapath functionality," in *Proceedings of ACM/SIGDA 11th International Symposium on Field-Programmable Gate Arrays (FPGA '03)*, pp. 195–204, Monterey, Calif, USA, February 2003.

[64] Silicon Hive, "Silicon Hive Technology Primer," Phillips Electronics NV, The Netherlands. 2003.

[65] A. G. Ye and J. Rose, "Using multi-bit logic blocks and automated packing to improve field-programmable gate array density for implementing datapath circuits," in *IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 129–136, Brisbane, Australia, December 2004.

[66] J. M. Arnold, "S5: the architecture and development flow of a software configurable processor," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '05)*, pp. 121–128, Singapore, Republic of Singapore, December 2005.

[67] Altera Inc., *Stratix II Device Handbook, Volume 1*, Altera, San Jose, Calif, USA, 2005.

[68] Xilinx Inc., *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*, Xilinx, San Jose, Calif, USA, 2005.

[69] Xilinx Inc., *Virtex-4 Family Overview*, Xilinx, San Jose, Calif, USA, 2004.

[70] S. Haynes, A. Ferrari, and P. Cheung, "Flexible reconfigurable multiplier blocks suitable for enhancing the architecture of FPGAs," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 191–194, San Diego, Calif, USA, May 1999.

[71] S. Hauck, T. Fry, M. Hosler, and J. Kao, "The Chimaera reconfigurable functional unit," in *Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97)*, pp. 87–96, Napa Valley, Calif, USA, April 1997.

[72] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic, Boston, Mass, USA, 1999.

[73] K.-I. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, 2001.

[74] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "The multiple wordlength paradigm," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 51–60, Rohnert Park, Calif, USA, April-May 2001.

[75] U. Malik, K. So, and O. Diessel, "Resource-aware run-time elaboration of behavioural FPGA specifications," in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT '02)*, pp. 68–75, Hong Kong, December 2002.

[76] Z. Zhao and M. Leeser, "Precision modeling of floating-point applications for variable bitwidth computing," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '03)*, pp. 208–214, Las Vegas, Nev, USA, June 2003.

[77] A. DeHon, J. Adams, M. DeLorimier, et al., "Design patterns for reconfigurable computing," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 13–23, Napa Valley, Calif, USA, April 2004.

[78] K. Han, B. L. Evans, and E. E. Swartzlander Jr., "Data wordlength reduction for low-power signal processing software," in *IEEE Workshop on Signal Processing Systems (SIPS '04)*, pp. 343–348, Austin, Tex, USA, October 2004.

[79] J. Park, P. C. Diniz, and K. R. Shesha Shayee, "Performance and area modeling of complete FPGA designs in the presence of loop transformations," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1420–1435, 2004.

[80] M. L. Chang and S. Hauck, "Précis: a usercentric wordlength optimization tool," *IEEE Design and Test of Computers*, vol. 22, no. 4, pp. 349–361, 2005.

[81] C. Morra, J. Becker, M. Ayala-Rincon, and R. Hartenstein, "FELIX: using rewriting-logic for generating functionally equivalent implementations," in *Proceedings of International Conference on Field-Programmable Logic and Applications*, pp. 25–30, Tampere, Finland, August 2005.

[82] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '98)*, pp. 179–188, Monterey, Calif, USA, February 1998.

[83] S. J. E. Wilton, "Implementing logic in FPGA memory arrays: heterogeneous memory architectures," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '02)*, pp. 142–147, Napa Valley, Calif, USA, April 2002.

[84] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware RAM mapping for FPGA embedded memory blocks," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '06)*, pp. 189–198, Monterey, Calif, USA, February 2006.

[85] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy-efficient signal processing using FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '03)*, pp. 225–234, Monterey, Calif, USA, February 2003.

[86] J. Ou, S. Choi, and V. K. Prasanna, "Performance modeling of reconfigurable SoC architectures and energy-efficient mapping of a class of application," in *Proceedings of 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '03)*, pp. 241–250, Napa Valley, Calif, USA, April 2003.

[87] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "A dual-vdd low power FPGA architecture," in *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications (FPL '04)*, pp. 145–157, Leuven, Belgium, August-September 2004.

[88] F. Li, Y. Lin, L. He, and J. Cong, "Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics," in *Proceedings of ACM/SIGDA 12th International Symposium on Field-Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 42–50, Monterey, Calif, USA, February 2004.

[89] A. Rahman and V. Polavarapuv, "Evaluation of low-leakage design techniques for field programmable gate arrays," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 23–30, Monterey, Calif, USA, February 2004.

[90] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware computer-aided design algorithms for field-programmable gate arrays," *Journal of Low Power Electronics*, vol. 1, no. 2, pp. 119–132, 2005.

[91] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279–302, 2005.

[92] A. DeHon, R. Huang, and J. Wawrzynek, "Hardware-assisted fast routing," in *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '02)*, pp. 205–215, Napa Valley, Calif, USA, April 2002.

[93] P. Maidee, C. Ababei, and K. Bazargan, "Fast timing-driven partitioning-based placement for island style FPGAs," in *Proceedings of the 40th Design Automation Conference (DAC '03)*, pp. 598–603, Anaheim, Calif, USA, June 2003.

[94] M. G. Wrighton and A. M. DeHon, "Hardware-assisted simulated annealing with application for fast FPGA placement," in *ACM/SIGDA 11th International Symposium on Field-Programmable Gate Arrays (FPGA '03)*, pp. 33–42, Monterey, Calif, USA, February 2003.

[95] M. Handa and R. Vemuri, "Hardware assisted two dimensional ultra fast placement," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '04)*, vol. 18, pp. 1915–1922, Santa Fe, NM, USA, April 2004.

[96] S. Li and C. Ebeling, "QuickRoute: a fast routing algorithm for pipelined architectures," in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 73–80, Brisbane, Australia, December 2004.

[97] R. Lysecky, F. Vahid, and S. X.-D. Tan, "A study of the scalability of on-chip routing for just-in-time FPGA compilation," in *Proceedings of 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 57–62, Napa Valley, Calif, USA, April 2005.

[98] M. Chu, N. Weaver, K. Sulimma, A. DeHon, and J. Wawrzynek, "Object oriented circuit-generators in Java," in *Proceedings of the 6th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '98)*, pp. 158–166, Napa Valley, Calif, USA, April 1998.

[99] A. Derbyshire and W. Luk, "Compiling run-time parametrisable designs," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '02)*, pp. 44–51, Hong Kong, December 2002.

[100] W. Wolf, *Computers as Components: Principles of Embedded Computer Systems Design*, Morgan Kaufmann, San Francisco, Calif, USA, 2000.

[101] F. Barat, R. Lauwereins, and G. Deconinck, "Reconfigurable instruction set processors from a hardware/software perspective," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 847–862, 2002.

[102] F. Razdan and M. Smith, "A high-performance microarchitecture with hardware-programmable functional units," in *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO '94)*, pp. 172–180, San Jose, Calif, USA, November-December 1994.

[103] R. D. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 126–135, Napa Valley, Calif, USA, April 1996.

[104] J. E. Carrillo and P. Chow, "The effect of reconfigurable units in superscalar processors," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '01)*, pp. 141–150, Monterrey, Calif, USA, February 2001.

[105] B. Mei, S. Vernalde, D. Verkest, and R. Lauwereins, "Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: a case study," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, vol. 2, pp. 1224–1229, Paris, France, February 2004.

[106] Altera Inc., *Nios II Processor Reference Handbook*, Altera, San Jose, Calif, USA, 2005.

[107] Xilinx Inc., *MicroBlaze Processor Reference Guide*, Xilinx, San Jose, Calif, USA, 2003.

[108] A. Lawrence, A. Kay, W. Luk, T. Nomura, and I. Page, "Using reconfigurable hardware to speed up product development and performance," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications (FPL '95)*, pp. 111–118, Oxford, UK, August-September 1995.

[109] J. M. Rabaey, A. Abnous, Y. Ichikawa, K. Seno, and M. Wan, "Heterogeneous reconfigurable systems," in *IEEE Workshop on Signal Processing Systems, Design and Implementation (SiPS '97)*, pp. 24–34, Leicester, UK, November 1997.

[110] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97)*, pp. 12–21, Napa Valley, Calif, USA, April 1997.

[111] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, "PipeRench: a virtualized programmable datapath in 0.18 Micron technology," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 63–66, Orlando, Fla, USA, May 2002.

[112] M. Bocchi, C. De Bartolomeis, C. Mucci, et al., "A XiRisc-based SoC for embedded DSP applications," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 595–598, Orlando, Fla, USA, October 2004.

[113] R. B. Kujoth, C.-W. Wang, D. B. Gottlieb, J. J. Cook, and N. P. Carter, "A reconfigurable unit for a clustered programmable-reconfigurable processor," in *Proceedings of ACM/SIGDA 12th International Symposium on Field-Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 200–209, Monterey, Calif, USA, February 2004.

[114] Xilinx Inc., *Virtex-II Platform FPGAs: Complete Data Sheet*, Xilinx, San Jose, Calif, USA, 2004.

[115] Actel Corporation, "VariCore^{TM} Embedded Programmable Gate Array Core (EPGA^{TM}) 0.18μm Family," Actel, Mountain View, Calif, USA, 2001.

[116] M2000, *Press Release—May 15, 2002.* M2000, Bièvres, France, 2002.

[117] K. Compton and S. Hauck, "Totem: custom reconfigurable array generation," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 111–119, Rohnert Park, Calif, USA, April-May 2001.

[118] STMicroelectronics, "STMicroelectronics Introduces New Member of SPEArTM Family of Configurable System-on-Chip ICs," Press Release, 2005, http://us.st.com/stonline/press/news/year2005/p1711p.htm.

[119] F. Yang and M. Paindavoine, "Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1162–1175, 2003.

[120] P. Weaver and F. Palma, "Using software-configurable processors in biometric applications," *Industrial Embedded Systems Resource Guide*, pp. 84–86, 2005, http://www.industrial-embedded.com.

[121] V. George, Z. Hui, and J. Rabaey, "The design of a low energy FPGA," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 188–193, San Diego, Calif, USA, August 1999.

[122] P. Heysters, G. J. M. Smit, and E. Molenkamp, "Energy-efficiency of the MONTIUM reconfigurable tile processor," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '04)*, pp. 38–44, Las Vegas, Nev, USA, June 2004.

[123] G. Asadi and M. B. Tahoori, "Soft error rate estimation and mitigation for SRAM-based FPGAs," in *Proceedings of the ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays (FPGA '05)*, pp. 149–160, Monterey, Calif, USA, February 2005.

[124] Xilinx Inc., *EasyPath Devices Datasheet*, Xilinx, San Jose, Calif, USA, 2005.

[125] N. Campregher, P. Y. K. Cheung, G. A. Constantindes, and M. Vasilko, "Yield enhancements of design-specific FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '06)*, pp. 93–100, Monterey, Calif, USA, February 2006.

[126] L. Sterpone and M. Violante, "Analysis of the robustness of the TMR architecture in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 5, pp. 1545–1549, 2005.

[127] P. Bernardi, M. Sonza Reorda, L. Sterpone, and M. Violante, "On the evaluation of SEU sensitiveness in SRAM-based FPGAs," in *Proceedings of the 10th IEEE International On-Line Testing Symposium (IOLTS &04)*, pp. 115–120, Madeira Island, Portugal, July 2004.

[128] A. Tiwari and K. A. Tomko, "Enhanced reliability of finite-state machines in FPGA through efficient fault detection and correction," *IEEE Transactions on Reliability*, vol. 54, no. 3, pp. 459–467, 2005.

[129] P. Graham, M. Caffrey, M. Wirthlin, D. E. Johnson, and N. Rollins, "Reconfigurable computing in space: from current technology to reconfigurable systems-on-a-chip," in *Proceedings of the IEEE Aerospace Conference*, vol. 5, pp. 2399–2410, Big Sky, Mont, USA, March 2003.

[130] K. Hasuko, C. Fukunaga, R. Ichimiya, et al., "A remote control system for FPGA-embedded modules in radiation enviornments," *IEEE Transactions on Nuclear Science*, vol. 49, no. 2, part 1, pp. 501–506, 2002.

[131] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Efficiently supporting fault-tolerance in FPGAs," in *Proceedings of the ACM/SIGDA 6th International Symposium on Field-Programmable Gate Arrays (FPGA '98)*, pp. 105–115, Monterey, Calif, USA, February 1998.

[132] N. Mokhoff, "'Infrastructure IP' Seen Aiding SoC Yields," *EE Times*, July 2002.

[133] B. P. Dave and N. K. Jha, "COFTA: hardware-software co-synthesis of heterogeneous distributed embedded systems for low overhead fault tolerance," *IEEE Transactions on Computers*, vol. 48, no. 4, pp. 417–441, 1999.

[134] J. W. S. Liu, *Real-Time Systems*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2000.

[135] F. Verdier, J. Prevotet, A. Benkhelifa, D. Chillet, and S. Pillement, "Exploring RTOS issues with a high-level model of a reconfigurable SoC platform," in *Proceedings of the European Workshop on Reconfigurable Communication Centric (ReCoSoC '05)*, Montpellier, France, June 2005.

[136] B. Griese, E. Vonnahme, M. Porrmann, and U. Ruckert, "Hardware support for dynamic reconfiguration in reconfigurable SoC architectures," in *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications (FPL '04)*, pp. 842–846, Leuven, Belgium, August-September 2004.

[137] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[138] K. Danne and M. Platzner, "Periodic real-time scheduling for FPGA computers," in *Proceedings of the 3rd Workshop on Intelligent Solutions in Embedded Systems (WISES '05)*, pp. 117–127, Hamburg, Germany, May 2005.

[139] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh, "Instruction generation and regularity extraction for reconfigurable processors," in *Proceedings of the International Conferences on Compilers Architectures and Synthesis of Embeded Systems (CASES '02)*, pp. 262–269, Grenoble, France, October 2002.

[140] S. Yehia, N. Clark, S. Mahlke, and K. Flautner, "Exploring the design space of LUT-based transparent accelerators," in *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '05)*, pp. 11–21, San Francisco, Calif, USA, September 2005.

[141] P. Yu and T. Mitra, "Satisfying real-time constraints with custom instructions," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (CODES+ISSS '05)*, pp. 166–171, New Jersey, NJ, USA, September 2005.

[142] T. Kean, "Secure configuration of field programmable gate arrays," in *Proceedings of 11th International Conference on Field-Programmable Logic and Applications (FPL '01)*, pp. 142–151, Belfast, Northern Ireland, UK, August 2001.

[143] L. Bossuet, G. Gogniat, and W. Burleson, "Dynamically configurable security for SRAM FPGA bitstreams," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 1995–2002, Santa Fe, NM, USA, April 2004.

[144] Xilinx Inc. and A. Telikepalli, *Is Your FPGA Design Secure?*, Xilinx, San Jose, Calif, USA, 2003.

[145] Altera Inc., *FPGA Design Security Solution Using Max II Devices*, Altera, San Jose, Calif, USA, 2004.

[146] C. R. Rupp, M. Landguth, T. Garverick, et al., "The NAPA adaptive processing architecture," in *Proceedings of 6th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '98)*, pp. 28–37, Napa Valley, Calif, USA, April 1998.

[147] K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[148] K. Compton, Z. Li, J. Cooley, S. Knol, and S. Hauck, "Configuration relocation and defragmentation for run-time reconfigurable computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 209–220, 2002.

[149] U. Malik and O. Diessel, "On the placement and granularity of FPGA configurations," in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 161–168, Brisbane, Australia, December 2004.

[150] G. Brebner, "Swappable logic unit: a paradigm for virtual hardware," in *Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97)*, pp. 77–86, Napa Valley, Calif, USA, April 1997.

[151] E. Caspi, R. Huang, Y. Markovskiy, J. Yeh, J. Wawrzynek, and A. DeHon, "A streaming multi-threaded model," in *Proceedings of the 3rd Workshop on Media and Stream Processors (MSP '01)*, pp. 21–28, Austin, Tex, USA, December 2001.

[152] Y. Markovskiy, E. Caspi, R. Huang, et al., "Analysis of quasi-static scheduling techniques in a virtualized reconfigurable machine," in *Proceedings of 10th ACM International Symposium on Field-Programmable Gate Arrays (FPGA '02)*, pp. 196–205, Monterey, Calif, USA, February 2002.

[153] V. Nollet, J.-Y. Mignolet, T. A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, "Hierarchical run-time reconfiguration managed by an operating system for reconfigurable systems," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, pp. 81–87, Las Vegas, Nev, USA, June 2003.

[154] Z. Li and S. Hauck, "Configuration compression for virtex FPGAs," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 147–159, Rohnert Park, Calif, USA, April-May 2001.

[155] Z. Li, K. Compton, and S. Hauck, "Configuration caching techniques for FPGA," in *Proceedings of 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, Napa Valley, Calif, USA, April 2000.

[156] A. DeHon, "DPGA utilization and application," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '96)*, pp. 115–121, Monterey, Calif, USA, February 1996.

[157] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, “A time-multiplexed FPGA,” in *Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 22–28, Napa Valley, Calif, USA, April 1997.

[158] Z. Li and S. Hauck, “Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation,” in *Proceedings of 10th ACM International Symposium on Field-Programmable Gate Arrays (FPGA ’02)*, pp. 187–195, Monterey, Calif, USA, February 2002.

[159] R. Maestre, F. J. Kurdahi, N. Bagherzadeh, H. Singh, R. Hermida, and M. Fernandez, “Kernel scheduling in reconfigurable computing,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pp. 90–96, Munich, Germany, March 1999.

[160] K. M. Gajjala Purna and D. Bhatia, “Temporal partitioning and scheduling data flow graphs for reconfigurable computers,” *IEEE Transactions on Computers*, vol. 48, no. 6, pp. 579–590, 1999.

[161] G. Brebner, “A virtual hardware operating system for the Xilinx XC6200,” in *Proceedings of the 6th International Workshop on Field-Programmable Logic and Applications (FPL ’96)*, pp. 327–336, Dermstadt, Germany, September 1996.

[162] J. Resano, D. Mozos, D. Verkest, and F. Catthoor, “A reconfiguration manager for dynamically reconfigurable hardware,” *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 452–460, 2005.

[163] A. Sudarsanam, M. Srinivasan, and S. Panchanathan, “Resource estimation and task scheduling for multithreaded reconfigurable architectures,” in *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS ’04)*, pp. 323–330, Newport Beach, Calif, USA, July 2004.

[164] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt, “Dynamic scheduling of tasks on partially reconfigurable FPGAs,” *IEE Proceedings: Computers and Digital Techniques*, vol. 147, no. 3, pp. 181–188, 2000.

[165] H. Quinn, L. A. S. King, M. Leeser, and W. Meleis, “Runtime assignment of reconfigurable hardware components for image processing pipelines,” in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’03)*, pp. 173–182, Napa Valley, Calif, USA, April 2003.

[166] G. Stitt, R. Lysecky, and F. Vahid, “Dynamic hardware/software partitioning: a first approach,” in *Proceedings of the 40th Design Automation Conference (DAC ’03)*, pp. 250–255, Anaheim, Calif, USA, June 2003.

[167] J. Noguera and R. Badia, “Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling,” *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 385–406, 2004.

[168] A. Ahmadinia, C. Bobda, D. Koch, M. Majer, and J. Teich, “Task scheduling for heterogeneous reconfigurable computers,” in *Proceedings of the 17th Symposium on Integrated Cicuits and Systems Design*, pp. 22–27, Pernambuco, Brazil, September 2004.

[169] R. Lysecky and F. Vahid, “A configurable logic architecture for dynamic hardware/software partitioning,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, pp. 480–485, Paris, France, February 2004.

[170] W. Fu and K. Compton, “An execution environment for reconfigurable computing,” in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’05)*, pp. 149–158, Napa Valley, Calif, USA, April 2005.

[171] T. Wiangtong, P. Y. K. Cheung, and W. Luk, “Hardware/software codesign: a systematic approach targeting data-intensive applications,” *IEEE Signal Processing Magazine*, vol. 22, no. 3, pp. 14–22, 2005.

[172] P. Benoit, L. Torres, G. Sassatelli, M. Robert, and G. Cambon, “Automatic task scheduling / loop unrolling using dedicated RTR controllers in coarse grain reconfigurable architectures,” in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS ’05)*, p. 148a, Denver, Colo, USA, April 2005.

[173] H. Simmler, L Levison, and R. Manner, “Multitasking on FPGA coprocessors,” in *The International Conference on Field-Programmable Logic, Reconfigurable Computing, and Applications (FPL ’00)*, pp. 121–130, Villach, Austria, August 2000.

[174] H. Kalte and M. Porrmann, “Context saving and restoring for multitasking in reconfigurable systems,” in *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL ’05)*, pp. 223–228, Tampere, Finland, August 2005.

[175] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood, “Hardware-software co-design of embedded reconfigurable architectures,” in *Proceedings of 37th Design Automation Conference (DAC ’00)*, pp. 507–512, Los Angeles, Calif, USA, June 2000.

[176] M. J. W. Savage, Z. Salcic, G. Coghill, and G. Covic, “Extended genetic algorithm for codesign optimization of DSP systems in FPGAs,” in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT ’04)*, pp. 291–294, Brisbane, Australia, December 2004.

[177] S. Kumar, J. H. Aylor, B. W. Johnson, and W. A. Wulf, *The Codesign of Embedded Systems: A Unified Hardware/Software Representation*, Springer, New York, NY, USA, 1995.

[178] M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, “Hardware-software codesign of embedded systems,” *IEEE Micro*, vol. 14, no. 4, pp. 26–36, 1994.

[179] R. Ernst, “Codesign of embedded systems: status and trends,” *IEEE Design and Test of Computers*, vol. 15, no. 2, pp. 45–54, 1998.

[180] W. Wolf, “A decade of hardware/software codesign,” *IEEE Computer*, vol. 36, no. 4, pp. 38–43, 2003.

[181] M. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, “Stream-oriented FPGA computing in the Streams-C high level language,” in *Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’00)*, Napa Valley, Calif, USA, April 2000.

[182] Synopsys Inc., “CoCentric System C Compiler,” Synopsys, Mountain View, Calif, USA, 2000.

[183] M. Weinhardt and W. Luk, “Pipeline vectorization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 234–248, 2001.

[184] D. Niehaus and D. Andrews, “Using the multi-threaded computation model as a unifying framework for hardware-software co-design and implementation,” in *Proceedings of the 9th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS ’03)*, p. 317, Capri, Italy, October 2003.

[185] B. Swahn and S. Hassoun, “Hardware scheduling for dynamic adaptability using external profiling and hardware threading,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD ’03)*, pp. 58–64, San Jose, Calif, USA, November 2003.

[186] G. De Micheli, “Hardware synthesis from C/C++ models,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pp. 382–383, Munich, Germany, March 1999.

[187] A. DeHon, “Very large scale spatial computing,” in *Proceedings of the 3rd International Conference on Unconventional Models of Computation (UMC ’02)*, pp. 27–37, Kobe, Japan, October 2002.

[188] D. Andrews, D. Niehaus, and P. Ashenden, “Programming models for hybrid CPU/FPGA chips,” *IEEE Computer*, vol. 37, no. 1, pp. 118–120, 2004.

[189] J.-P. David and J.-D. Legat, “A data-flow oriented co-design for reconfigurable systems,” in *Proceedings of the 9th International Workshop on Rapid System Prototyping*, pp. 207–211, Leuven, Belgium, June 1998.

[190] R. Rinker, M. Carter, A. Patel, et al., “An automated process for compiling dataflow graphics into reconfigurable hardware,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 130–139, 2001.

[191] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, “DRESC: a retargetable compiler for coarse-grained reconfigurable architectures,” in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT ’02)*, pp. 166–173, Hong Kong, December 2002.

[192] J. M. P. Cardoso, “On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures,” *IEEE Transactions on Computers*, vol. 52, no. 10, pp. 1362–1375, 2003.

[193] S. Banerjee, E. Bozorgzadeh, and N. Dutt, “Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration,” in *Proceedings of the 42nd Design Automation Conference (DAC ’05)*, pp. 335–340, Anaheim, Calif, USA, June 2005.

[194] C. Bobda and A. Ahmadinia, “Dynamic interconnection of reconfigurable modules on reconfigurable devices,” *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 443–451, 2005.

[195] B. Hutchings and B. Nelson, “Developing and debugging FPGA applications in hardware with JHDL,” in *Proceedings of 33rd Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 554–558, Pacific Grove, Calif, USA, October 1999.

[196] K. A. Tomko and A. Tiwari, “Hardware/software codebugging for reconfigurable computing,” in *Proceedings of the 5th IEEE International High-Level Design, Validation, and Test Workshop (HLDVT ’00)*, pp. 59–63, Berkeley, Calif, USA, November 2000.

[197] T. Rissa, W. Luk, and P. Y. K. Cheung, “Automated combination of simulation and hardware prototyping,” in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA ’04)*, pp. 184–193, Las Vegas, Nev, USA, June 2004.

[198] G. Talavera, V. Nollet, J.-Y. Mignolet, et al., “Hardware-software debugging techniques for reconfigurable systems-on-chip,” in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT ’04)*, vol. 3, pp. 1402–1407, Hammamet, Tunisia, December 2004.

[199] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, “An automated exploration framework for FPGA-based soft multiprocessor systems,” in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS ’05)*, pp. 273–278, Jersey City, NJ, USA, September 2005.

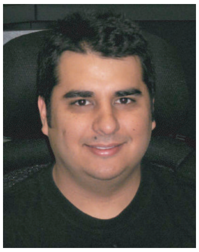
[200] P. Yiannacouras, J. G. Steffan, and J. Rose, “Application-specific customization of soft processor microarchitecture,” in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA ’06)*, pp. 201–210, Monterey, Calif, USA, February 2006.

[201] R. E. Gonzalez, “Xtensa: a configurable and extensible processor,” *IEEE Micro*, vol. 20, no. 2, pp. 60–70, 2000.

[202] A. Yan and S. J. E. Wilton, “Sequential synthesizable embedded programmable logic cores for system-on-chip,” in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC ’04)*, pp. 435–438, Orlando, Fla, USA, October 2004.

[203] S. Hauck, K. Compton, K. Eguro, M. Holland, S. Philips, and A. Sharma, “Totem: domain-specific reconfigurable logic,” to appear in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[204] I. Kuon, A. Egier, and J. Rose, “Design, layout and verification of an FPGA using automated tools,” in *Proceedings of the ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays (FPGA ’05)*, pp. 215–226, Monterey, Calif, USA, February 2005.



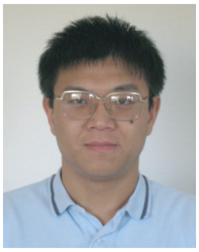
Philip Garcia received a B.S. degree in computer engineering from Lehigh University. He also received his M.S. degree at Lehigh University, concentrating on architecture-aware database algorithms. He currently is an Electrical Engineering Ph.D. Student at the University of Wisconsin-Madison studying under the advisement of Dr. Katherine Compton. His current research is in the design of interfaces between reconfigurable hardware and general processor systems.

Katherine Compton received her B.S., M.S., and Ph.D. degrees from Northwestern University in 1998, 2000, and 2003, respectively. Since January of 2004, she has been an Assistant Professor at the University of Wisconsin-Madison in the Department of Electrical and Computer Engineering. She and her graduate students are investigating new architectures, logic structures, integration techniques, and systems software techniques for reconfigurable computing. She serves on a number of program committees for FPGA and reconfigurable computing conferences and symposia. She is also a Member of both ACM and IEEE.

Michael Schulte received a B.S. degree in electrical engineering from the University of Wisconsin-Madison, and M.S. and Ph.D. degrees in electrical engineering from the University of Texas at Austin. He is currently an Associate Professor at the University of Wisconsin-Madison, where he leads the Madison Embedded Systems and Architectures Group. His research interests include high-performance embedded processors, computer architecture, domain-specific systems, computer arithmetic, and reconfigurable computing. He is a Senior Member of the IEEE and the IEEE Computer Society, and an Associate Editor for the IEEE Transactions on Computers and the Journal of VLSI Signal Processing.

Emily Blem received a B.S. degree in Engineering and a B.A. degree in Mathematics from Swarthmore College. She is currently pursuing her Ph.D. degree at the University of Wisconsin-Madison. Her research interests include computer architecture, performance analysis and modeling, and reconfigurable computing. She is a Member of the IEEE and the IEEE Computer Society.

Wenyin Fu received the B.S. degree from Shanghai Jiaotong University in 1999 and the M.S. degree in both electrical engineering and computer science from the University of Wisconsin at Madison, in 2003 and 2004, respectively. His research interests center on computer architecture, embedded systems, and reconfigurable computing. He is currently working toward a Ph.D. degree at the same university, studying with Dr. Katherine Compton.



Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 38494, Pages 1–15
DOI 10.1155/ES/2006/38494

Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC

Ari Kulmala, Olli Lehtoranta, Timo D. Hämäläinen, and Marko Hännikäinen

Department of Information Technology, Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, Korkeakoulunkatu 1, 33101 Tampere, Finland

Received 15 December 2005; Revised 16 May 2006; Accepted 27 June 2006

High computational requirements combined with rapidly evolving video coding algorithms and standards are a great challenge for contemporary encoder implementations. Rapid specification changes prefer full programmability and configurability both for software and hardware. This paper presents a novel scalable MPEG-4 video encoder on an FPGA-based multiprocessor system-on-chip (MPSOC). The MPSOC architecture is truly scalable and is based on a vendor-independent intellectual property (IP) block interconnection network. The scalability in video encoding is achieved by spatial parallelization where images are divided to horizontal slices. A case design is presented with up to four synthesized processors on an Altera Stratix 1S40 device. A truly portable ANSI-C implementation that supports an arbitrary number of processors gives 11 QCIF frames/s at 50 MHz without processor specific optimizations. The parallelization efficiency is 97% for two processors and 93% with three. The FPGA utilization is 70%, requiring 28 797 logic elements. The implementation effort is significantly lower compared to traditional multiprocessor implementations.

1. INTRODUCTION

Video is becoming an essential part of embedded multimedia terminals. There are, however, many contradicting constraints in video codec implementations. One challenge is the rapid evolution of compression standards with several different algorithms. This requires programmability that is easy to achieve with processor-based platforms. However, achieving the best power, energy, and silicon area efficiency requires custom hardware implementations. On the other hand, hardware (HW) design is more demanding than software (SW) development, and modifications are very expensive and time consuming. For example, nonrecurring engineering (NRE) costs, especially photo mask fabrication costs, increase rapidly with each technology generation making frequent HW upgrades less favorable. Software only implementation solves the flexibility and upgradeability problem but is not an optimal solution from a performance versus silicon area point of view.

The work in this paper solves the HW video codec design flexibility and upgradeability problem with a fully programmable, scalable MPSOC approach [1, 2]. The key idea is to use synthesizable soft-core processors and a synthesizable system-on-chip (SOC) interconnection network, which allows prototyping and implementation on any FPGA platform, or in an ASIC technology with a rapid design cycle. In addition, our implementation framework enables a seamless trade-off between performance and area without creating an extra burden in system design by scaling the number of identical processors. Furthermore, the architecture is designed to be easily reusable for any kind of application.

A data parallel MPEG-4 simple profile (SP) software encoder is implemented on the MPSOC to demonstrate the effectiveness and scalability of the presented solution. The video images are divided into independent horizontal slices which are mapped to different processors for encoding. A master-slave configuration is used where the master processor is responsible for overall control and the slaves perform the video encoding.

In this paper, we use an Altera Stratix FPGA as the target platform [3], Altera Nios processors, and our heterogeneous IP block interconnection v.2 (HIBI) [4] as the communication network. No MPEG-4 specific HW accelerators, for example, for motion estimation, are currently used, but HIBI provides a very convenient plug-and-play method to add intellectual property (IP) blocks independent of the vendor.

Topics of interest in this paper include practical implementation issues, such as utilized FPGA resources and achieved performance, design cycle improvement, scalability, and encoder specific issues like memory optimization due to scarce on-chip memories. The implementation works in practice with an FPGA board attached to a PC that sends source video streams and receives compressed data.

This paper is organized as follows. Related work is reviewed in Section 2. The MPSOC architecture is described in Section 3. The video encoding software and our encoder parallelization approach are presented in Section 4. Section 5 explains the integration of the HW architecture and software. In Section 6 the results are presented. Finally, Section 7 summarizes the paper and discusses future work.

2. RELATED WORK

In this section we consider the related work in two categories, parallel video encoding and FPGA-based MPSOC architectures.

2.1. Parallel encoder implementations

Due to high computational complexity of video encoding [5], several parallel solutions have been developed in contrast to traditional sequential program flow [6]. There are at least four general parallelization methods used: functional, temporal, data, and video-object parallelism. For functional parallelism [7] different functions, such as DCT and motion estimation, are connected in a functional pipeline to be executed in parallel by different processing units. However, scaling a functional parallel application requires a lot of work (high scaling effort). When each processor executes a specific function, adding or removing processors requires a whole system redesign to balance the computational load in the pipeline. For temporal parallelism (i.e., parallel in time) a full frame is assigned to every CPU. The scalability of this style is high. However, as the number of parallel encoders increase, it introduces a significant latency in encoding, since one frame is buffered for each encoding CPU. Works in this category include [8–10]. In data parallelism, the image is divided into slices that are assigned to different CPUs. The slices are encoded in parallel frame-by-frame. This approach is used in [11–13]. For video-object parallelism, which is specific to MPEG-4, arbitrary sized shapes referred to as video-objects in the image are assigned to different CPUs. The objects can be considerably unequal in size, which may lead to unbalanced execution time between different CPUs if care is not taken. Such work is presented, for example, in [14].

We are mainly interested in real-time encoding. Functional, data, and video-object parallelism are all eligible for real-time, low-latency video encoding, because they do not require frames to be buffered. We chose data parallelism because functional parallelism has a high scaling effort and video-object parallelism is strictly MPEG-4 specific. Scalability is the most feasible criterion used to compare different architectures and parallel implementations, because reported results typically vary in accuracy. The scalability of different parallelization methods is compared in Section 6.

Contemporary FPGA designs tend to use single encoder cores with HW accelerators arranged in a functional pipeline. Our implementation is one of the first utilizing multiple parallel encoders on an FPGA in a data parallel configuration. In [15], an FPGA-based H.263 encoder is demonstrated requiring 400 kgates for HW accelerators while providing 30 QCIF frames/s at 12 MHz. Reference [16] presents FPGA implementations of hardware accelerators for an H.264 video codec. In [17], an H.264 coder is designed and verified with an FPGA emulator platform. An interface between a host PC and an FPGA-based MPEG-4 encoder is built in [18] enabling fast prototyping and debugging.

2.2. FPGA multiprocessor architectures

Although multiprocessor systems have been researched for a while, most of the work has concentrated on ASIC implementations. FPGAs have only recently grown large enough to hold such implementations, which is one reason for a low number of reported FPGA-based MPSOCs. However, two trends of research can be identified.

First, FPGA multiprocessor systems are used to develop parallel applications. In these works, the main emphasis is usually on the application and its parallelization. The hardware architectures are briefly summarized. Typical implementations rely on vendor-dependant solutions, because they are usually easy to use. The hardware restrictions to scalability or flexibility and the ease of adding or removing components are often not addressed.

In [19], Martina et al. have developed a shared memory FPGA multiprocessor system of digital signal processor (DSP) cores of their own design that run basic signal processing algorithms. The implemented bus-interconnection is not described. There are no synthesis results for the whole system, but the system runs at 89 MHz on a Xilinx XCV1000 FPGA. Wang and Ziavras presented a system of six soft-core Nios processors [20]. Processors are interconnected with a multimaster Avalon bus [21]. No figures of the area required for the whole system are presented. However, the maximum clock frequency is 40 MHz using an Altera EP20K200EFC484-2x FPGA board.

Second, a hardware-oriented point of view for future multiprocessor requirements is presented. Reconfigurability is often emphasized [22]. Also, IP-block-based systems are stressed and a need for a scalable, standard interface interconnection network is anticipated. Kalte et al. [22] have presented a multilayer advanced microcontroller bus architecture (AMBA) interconnection architecture used in an FPGA. In AMBA, access to slaves is multiplexed between masters and different masters can use different peripherals simultaneously. A conceptual view of a system is depicted although not implemented. The interconnection architecture is synthesized separately, as is the processor. No application is described.

This work combines both of the above categories, since an application and a working prototype is implemented on the proposed architecture. The architecture itself is constructed of IP blocks and a general purpose interconnection architecture with support for an OCP-IP interface [23], which is a standard interconnection interface. A standardized IP block interface along with high scalability ensures the future use of the architecture.

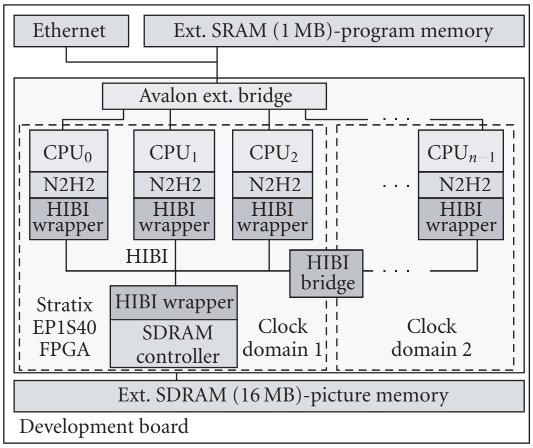


Figure 1: High-level view of the architecture on an FPGA development board.

Table 1: Summary of HIBI characteristics.

Property	HIBI implementation
Topology	Hierarchical bus with wrappers and bridges between bus segments
Interface	FIFO, OCP
Clocking	Multiple clock domains
Arbitration	Distributed, pipelined
Arbitration algorithm	Priority, round-robin, time division multiple access (TDMA), or combination
Synthesis-time configurable parameters	FIFO sizes, data width, addresses, initial configuration, number of configuration pages and their type (RAM or ROM), included properties
Run-time configurable parameters	All arbitration parameters and algorithm, cycle counters, power mode
Quality-of-service (QoS)	TDMA, send limit + priority/round-robin, multiple priorities for data, fast reconfiguration
Bus resolution	OR-network
Addressing	Multiple addresses per IP, multiplexed in the bus with data, allows multicast

3. MPSOC ARCHITECTURE

In this section we present our novel architecture for multiprocessor systems. The key elements of the architecture are processor programmability and scalability and flexibility that are obtained with the HIBI on-chip network [4]. A high-level view of the architecture is depicted in Figure 1. It contains a parameterizable number of soft-core processors, an HIBI network, direct memory access (DMA) blocks (N2H2), and two external memories located on the development board. In this case, external SRAM memory is used for instructions for all processors. HIBI is used to access the external SDRAM memory, which is used for shared data, and to let processors to send messages directly to each other.

3.1. *Heterogeneous IP block interconnection v.2*

HIBI is a hierarchical interconnection for SOCs, which was originally developed for ASICs. The objective of HIBI is to provide a topology-independent, scalable, yet high-performance on-chip network. It is highly configurable and supports multiple clock domains. To provide maximum efficiency, there are no empty cycles during bus transfers under a high load. Split transactions are used in read operations. Bus capacity is fully exercised by starting new transfers right after the preceding one has finished. The HIBI network can be reconfigured at run-time, using a configuration RAM. This should not be confused with FPGA reconfiguration.

The basic building block of the HIBI network is an HIBI wrapper. As HIBI utilizes distributed arbitration, each HIBI wrapper is responsible for seizing the bus at right time. The main characteristics of HIBI are presented in Table 1 [1]. In Figure 1, three processors and an SDRAM controller form one segment, Clock domain 1, and the rest of the processors form another segment, Clock domain 2. There is no master/slave configuration in the interconnection and, thus, every IP block can freely access any other IP block. An HIBI bridge is used to connect the segments together to allow transfers between segments. Every wrapper is assigned an address space, which can vary depending on the number of wrappers and the need for different addresses per wrapper. When data is sent via HIBI, only the destination address is delivered. In order to distinguish between different sources, each source must use a separate destination address within the recipients address space.

There are several different HIBI wrapper interfaces for IP blocks. It is left to the designer to decide which interface is the most appropriate, but there may be a mix of different kinds of interfaces. For example, FIFO and OCP interfaces can be utilized in the same architecture. Also, HIBI supports multiple priorities for data. Optionally, there can be different FIFOs for every priority. In that case, the highest priority FIFOs are always treated first. Thus, they can interrupt lower priority data transfers to get service immediately.

3.2. *Soft-core processors*

Currently, we have used soft-core processors Nios [24] and Nios II [25, 26] in our architecture. The master processor is Nios with a configuration shown in Figure 2. The peripherals are timers, LEDs button parallel I/Os (PIOs), and an UART. Nios can be used as a 16-bit or 32-bit processor, which affects the data bus width. Nios always uses 16-bit instruction words, which restricts the immediate values to five bits. With prefix-instructions this is increased to 16 bits. A large number of prefix instructions reduces code efficiency.

Nios II is a 32-bit CPU with 32-bit instructions. Nios II has more limited configurability. We use the fastest version,

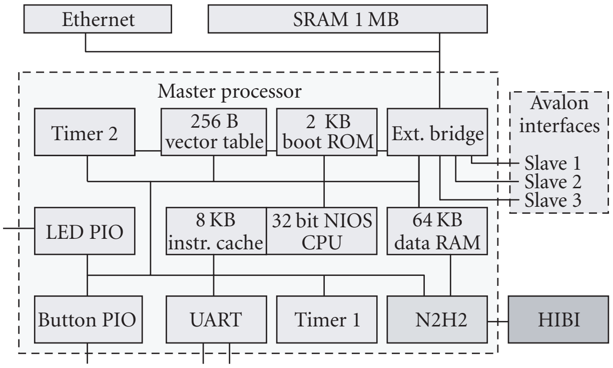


Figure 2: Master processor configuration.

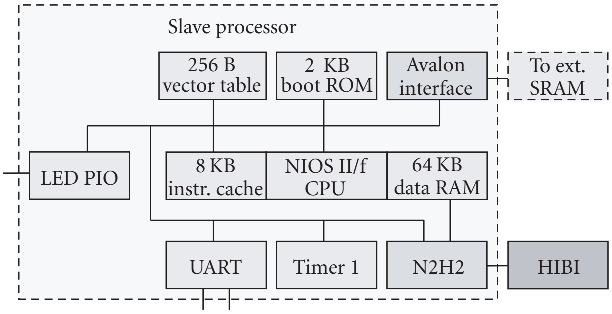


Figure 3: Slave processor configuration.

Nios II/f (fast) that provides 1.16 DMIPS/MHz and an area consumption of around 2000 LEs. Nios II is used for video encoding slaves with the configuration depicted in Figure 3. Unlike Nios, Nios II uses *tightly-coupled memory* between the processor and data RAM. Data does use the Avalon [21] bus that significantly reduces the memory access time.

Nios natively use the Avalon bus to connect to memories and peripherals. Avalon bus masters can command slaves, but slaves can only get the attention of a master by raising an interrupt. A typical master is a processor. A drawback is that masters cannot communicate directly with each other.

Both Nios processors have separate instruction and data buses. In Figures 2 and 3 only the data bus is drawn for simplicity. The instruction bus connects to the boot ROM, instruction memory (ext. SRAM), and vector table. With the Avalon bus, there are 20 bus master *address* lines, 32 *data* lines, and *waitrequest* and *read* signals. Data bus master has 20 *address* lines, two 32-bit *data* buses for read and write, and signals *waitrequest*, *read*, *write*, *irq* and *irq number*. This makes 92-signal lines in total. There are possibly other signal lines as well, but even with this practical minimum, there are 146-signal lines in the buses. As a comparison, 32-bit HIBI bus consists of only 38-signal lines (32-bit *data*, 3-bit *command*, *address_valid*, *lock*, and *full*). In addition, HIBI supports interprocessor communication without restrictions. The features of Avalon are not sufficient for data intensive multiprocessor communication, motivating the use of HIBI.

3.3. Nios-to-HIBI v.2 DMA

Nios processors do not have a native support for the HIBI on-chip network. Therefore, a DMA block, Nios-to-HIBI v.2 (N2H2), was implemented to attach the processors to HIBI. DMA minimizes CPU intervention in transfers. This allows the CPU to execute the application while DMA transfers data on the background.

N2H2 includes three Avalon interfaces. The slave interface is used by a CPU to control and query N2H2. These configuration and status registers include the state of the DMA block, DMA transfer instructions, and DMA receiving instructions. Two master interfaces are used separately for receiving and transmitting. In order to increase the reusability, these interfaces have been isolated from the other as much as possible. Thus, with minimal modifications, the same block can be applied to different processors.

The transmitter side is fairly straightforward. First, the CPU writes the memory address (e.g., a pointer), the amount of data, priority, and destination address to the configuration register. Following this, the transmitter sends the address to the HIBI. The transmitter then reads the data from memory and instantly transfers the data.

Receiving is more complicated. Data sent through HIBI may get fragmented. To circumvent this, we have implemented multiple receiving channels that wait for a given amount of data before interrupting the CPU. Each channel can be configured to receive data from any source and save it to memory locations defined by the CPU. There can be several data transfers going on simultaneously, so N2H2 uses the HIBI address to distinguish between them. For example, if two sources are sending data simultaneously, two channels are used. When the expected number of data has arrived on a channel, the CPU is notified by an interrupt or via a poll register.

Figure 4 depicts a data transfer over HIBI. CPU1 sends four words to CPU2. On cycle 0, CPU1 gives a start command to the N2H2 DMA. IRQ is acknowledged in clock cycle 1 and the transfer is started immediately. The address is sent first and then the data. Clock cycles 3–8 are consumed by the HIBI wrapper and arbitration latency. During this delay, another transmission can be proceeding in HIBI, so the latency is hidden. When the access to the HIBI is gained, the address and the data are sent forward in clock cycles 9–13. The data propagates through the receiving unit and buffers until at clock cycle 15 N2H2 sees the address and determines the right channel. Clock cycles 16–19 are used to store the data in the memory of the CPU2. After all the data expected has been received, an IRQ is given to the CPU2 at clock cycle 21.

4. MPEG-4 SOFTWARE IMPLEMENTATION

One of the key advantages of data parallel encoding methods is that they enable scalability by using macroblock row, macroblock, or block-level image subdivision. Moreover, spatial data parallelization can be performed with vertical, horizontal, rectangular, or arbitrary shaped slices. The problem of

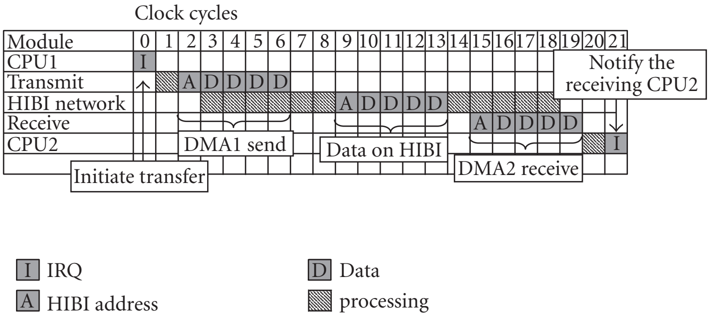
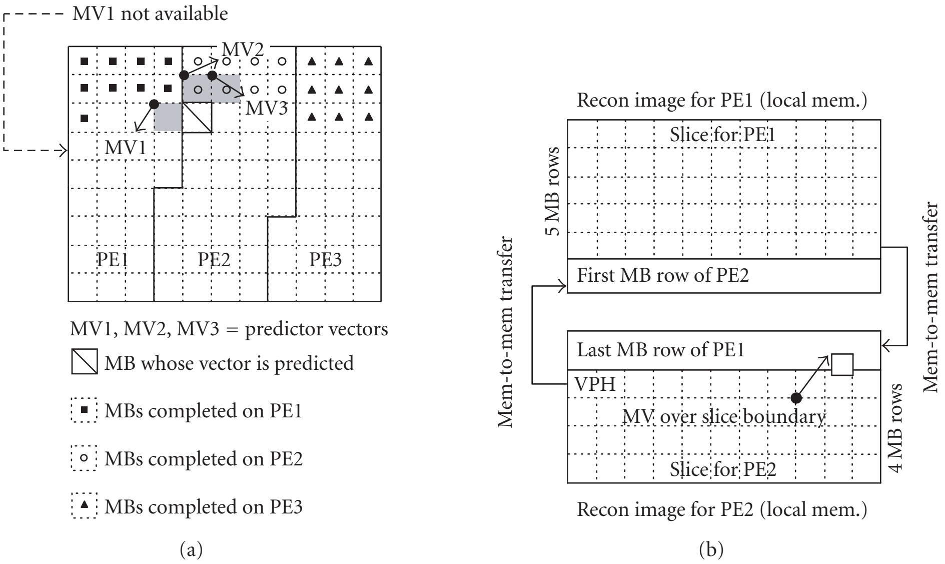


Figure 4: An arbitrary four-word data transfer over HIBI between two CPUs.



(a)

(b)

Figure 5: (a) Motion vector dependency problem in vertical parallelization and (b) horizontal parallelization for distributed memory machines.

vertical parallelization, shown on the left side of Figure 5, is that predictive coding is not considered, leading to motion vector (MV) and DQUANT (denoting changes in quantization parameter (QP)) dependency problems [27]. For example, H.263/MPEG-4 vector prediction is performed by computing the median of three neighboring vectors referred to as MV1, MV2, and MV3 in Figure 5. Due to data-dependent computations and the different shape of slices, computations do not proceed in a synchronized manner in different slices. For this reason, a data dependency problem arises in the slice boundaries where one of the predictor vectors may not be available.

Horizontal spatial partitioning, however, is natural to raster scan macroblock (MB) coding. The right side of Figure 5 depicts our previous implementation on a distributed memory DSP using MB row granularity [27]. The reconstructed images are made slightly overlap to allow motion vectors to point over slice boundaries. The overlapping areas are also exchanged between processors after local image reconstruction. Prediction dependencies are eliminated by inserting slice headers such as H.263 group-of-block (GOB) or MPEG-4 video packet headers (VPH) in the beginning of a slice. Clearly, this results in some overhead in the bit stream but prediction dependencies are avoided. In addition, interprocessor communication and extra memory is needed to implement the overlapping.

However, a drawback of [27] is a somewhat coarse granularity leading to unbalanced computational loads due to the unequal size of slices. For this reason, the original approach is improved by subdividing images using macroblock granularity as in Figure 6. Interprocessor communication and overlapping are further avoided by exploiting a shared memory in an MPSOC type of platform. The new method is highly scalable since the whole image is assignable to a single processor while the largest configuration dedicates a processor for each MB. No interprocessor communication is needed since data can be read directly from the global memory buffer. The shared memories, however, are potential bottlenecks, and thus efficient techniques for hiding transfer latencies are needed.

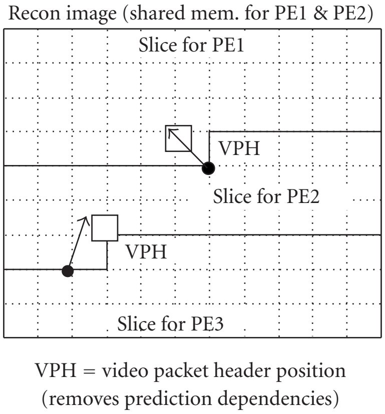

Recon image (shared mem. for PE1 & PE2)
Slice for PE1
VPH
Slice for PE2
VPH
Slice for PE3
VPH = video packet header position
(removes prediction dependencies)

Figure 6: Horizontal data parallelization for shared memory.

The data parallelization in Figure 6 was implemented with a master control program and slave MPEG-4 encoders. In addition, a host PC program has been implemented as a user interface. It should be noted that the master and the slave refer to the encoding processors, not, for example, to the Avalon bus master or slave used for Avalon communication. The flow graphs and the synchronization of SW are depicted in Figure 7 while the implementation and the integration details of the master and the slave processor are discussed in Section 5. The master processor controls and synchronizes the encoding. The tasks of the host PC, the master, and the slaves are presented in the following.

4.1. *Software implementation*

The host PC implements a user interface for inputting encoding parameters to the master. The user interface enables the selection of a video format (resolution and frame rate), bit rate control mode (constant or variable), quantization parameter (QP), as well as the number of slaves used in the encoding. The host PC and the master can communicate via a custom UDP/IP-based messaging protocol, which supports its own flow control, retransmissions, packet structures, fragmentation, and assembly. Our messaging protocol allows real-time modification of the frame rate, QP and bit rate parameters during the encoding.

The tasks of the host PC also include capturing and loading a raw video image, sending the raw data to the master and decoding the output. Received bits are stored to the local disk for debugging. In addition, the host PC measures statistics such as the average encoding frame rate and bit rate. At any time, the host PC can issue a reinitialization command to stop the encoding, release dynamically allocated SW resources, for example, memory, and return to the initial state. For example, this feature enables changes in the video resolution and the number of slaves without rebooting the platform. Also, prototyping and testability are improved since several video formats can be successively tested by changing the parameters.

The tasks of the master are illustrated in the middle of Figure 7. To encode a frame, the master first waits for the parameters from the host PC. Next, the PC sends the raw image (one frame at a time). The master slices the received image, configures the slaves, and signals them to start the encoding. As the slaves complete, each informs the master that it has finished. After all the slaves have completed encoding, the master finds out the sizes of the bit streams of the slaves, merges the bit streams, and sends the merged bit stream (encoded image) to the PC.

Slave tasks are illustrated on the right of Figure 7. First, the slave waits for the parameters from the master. Then, the slave downloads a local motion estimation (ME) window and pixels of the corresponding image macroblock. Then, it encodes the macroblock. This continues as long as there are macroblocks in the slice left to encode. If the local bit buffer goes full, the bits are uploaded to the external image memory. After all the macroblocks have been encoded, the slave uploads the bit buffer to the external memory and begins to wait for the next slice.

The video encoder can run in two different modes: first, it can run in real-time, so one frame at a time is transferred forth and back. Second, it can run in buffered mode, where the PC downloads a video sequence to the master. It is encoded as a whole and sent back to the PC. The video sequence length is parameterizable. Buffered mode mimics a situation where, for example, a video camera is attached to the system and feeding the encoder.

5. INTEGRATION

We have now presented a highly scalable hardware platform and a data parallel video encoder. Their integration is presented in the following. The main properties of the Stratix FPGA chip [28] used for this project are given in Table 2. A logic element (LE) contains a four input look-up table and a flip-flop. A digital signal processing (DSP) block contains multiply-and-accumulate (MAC) blocks. These blocks can also be used as fast embedded multipliers, which are utilized by the processors. Phase-locked loops (PLL) are used to generate different clock frequencies. Embedded memory provides on-chip storage.

Apart from the Stratix 1S40 FPGA, the development board offers two UARTs and an Ethernet connection. It has 8 MB of external Flash memory, 1 MB of external SRAM memory, and 16 MB of external SDRAM memory. Downloading and debugging is done via a JTAG connection.

5.1. *Initial constraints for the architecture*

The application requires 64 KB local data memories. As this memory is used for stack and local variables, it needs to be fast on-chip memory. Also, some memory is used for instruction caches. Thus, the limited amount of on-chip RAM in the FPGA bounds the maximum number of processors to four. Optionally, external memories could be used, but the development board does not contain any free, suitable

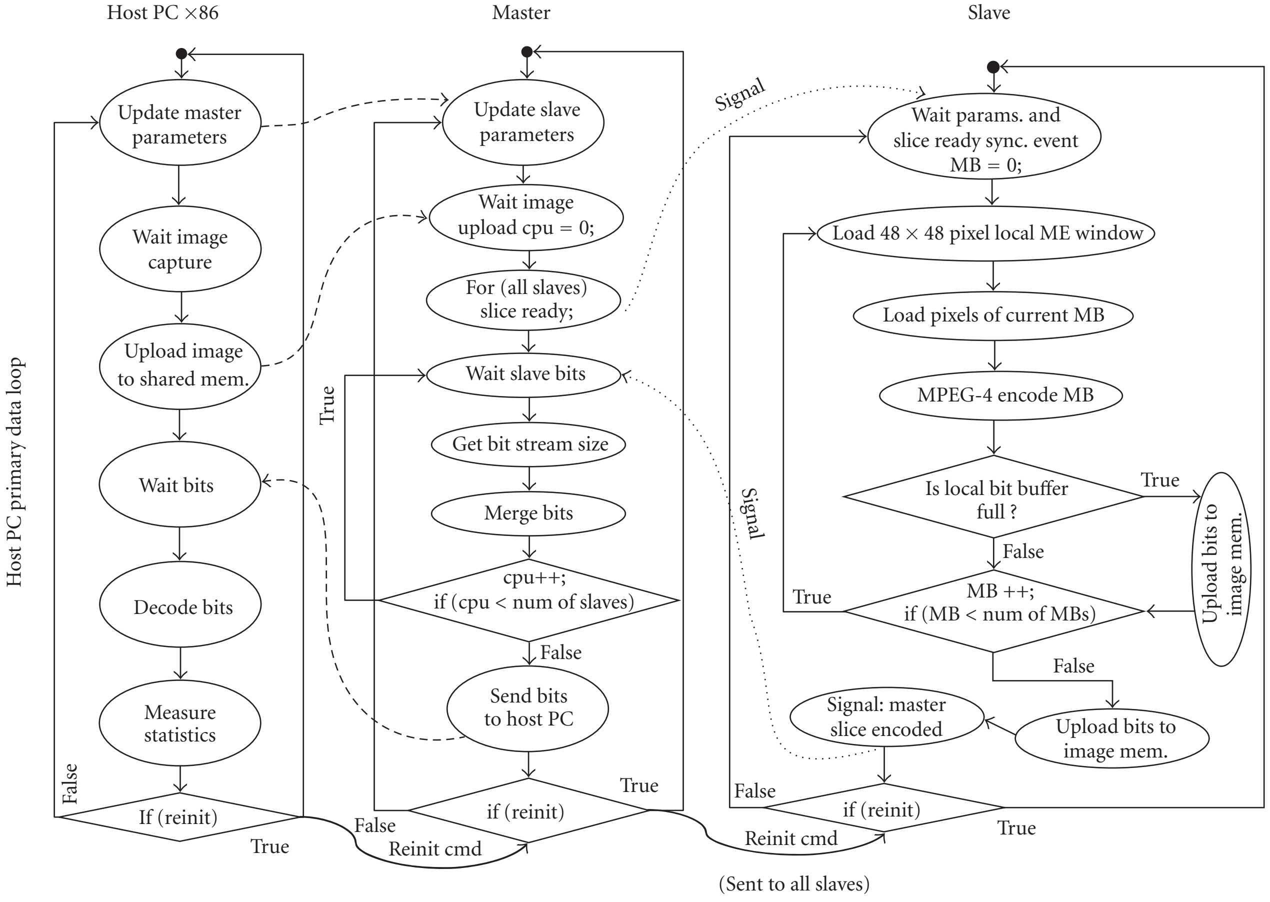

Host PC ×86
Master
Slave
Update master parameters
Wait image capture
Upload image to shared mem.
Wait bits
Decode bits
Measure statistics
If (reinit)
False
True
Host PC primary data loop
Update slave parameters
Wait image upload cpu = 0;
For (all slaves) slice ready;
Wait slave bits
Get bit stream size
Merge bits
cpu++; if (cpu < num of slaves)
True
False
Send bits to host PC
if (reinit)
True
False
Reinit cmd
Signal
Signal
Wait params. and slice ready sync. event MB = 0;
Load 48 × 48 pixel local ME window
Load pixels of current MB
MPEG-4 encode MB
Is local bit buffer full ?
True
False
Upload bits to image mem.
MB ++; if (MB < num of MBs)
True
False
Upload bits to image mem.
Signal: master slice encoded
if (reinit)
False
True
Reinit cmd
(Sent to all slaves)

Figure 7: Software flow graphs and task synchronization.

Table 2: Stratix 1S40 FPGA properties.

Feature	Stratix 1S40 contains
Logic elements (LEs)	41 250
Embedded memory, RAM (bits)	3 423 744
DSP blocks	14
PLLs	12

memories as the external SRAM and external SDRAM are already utilized. Therefore, at maximum one master processor and three slaves are used. The amount of the LEs in the FPGA is more than sufficient to allow for scalability.

Three different memories are used for the video encoding: the on-chip embedded memory, the external SRAM, and the external SDRAM. The flash memory is only used to configure the FPGA upon power-up. The same encoding software can be used for all slaves, which provides an opportunity to use the same instruction memory for all of them. As the application fits to 512 KB, the 1 MB SRAM memory was divided evenly between the master and the slave processors. To reduce the shared memory contention, instruction memory caches were used. Each cache utilizes 8 KB of on-chip memory.

The video encoder was configured in such a way that a 64 KB of local data memory is sufficient for each processor. Small buffers and memories like 2 KB boot program ROMs were also assigned to the on-chip memory. The external 16 MB SDRAM was allocated as the frame memory. A custom SDRAM controller was implemented with special DMA functions. General block transfer commands are implemented with an option to support application-specific commands. For example, we have currently implemented a command for an automatic square block retrieval, for instance an 8 × 8 block, for the video encoder application. In this way we need to commit only a single transfer instead of eight separate transfers. However, the SDRAM controller is fully reusable and can also be used without the application-specific features. The control interface also supports sequential block writes and reads, increasing the efficiency comparing to single write/read operations. The SDRAM DMA is configured with the highest priority messages. However, in practice, using higher priority does not have a notable effect on performance in this application.

5.2. Configurations of the components

The exact configurations of the processors are shown in Figures 2 and 3. The bus to the external SRAM goes through the master processor in practice, as opposed to the architecture shown in Figure 1. Slaves have no access to the address space of the master. Master, however, can access the slave portion of the memory. That gives an opportunity to reconfigure the slaves at run-time, changing the program on the fly. This feature is not utilized in the presented encoder. The data RAM in all CPUs is dual-port, the alternate port is used by the N2H2. Slave UARTs are multiplexed so that the PC can monitor any of the slaves. Timers are used for benchmarking and profiling the program. The master needs an extra timer to be used with the Ethernet.

HIBI is configured as a 32-bit wide single bus and used with the single-FIFO interface. HIBI uses priority-based arbitration, in which the master processor has the highest priority. Each of the HIBI wrappers has a five words deep low priority data FIFO and a three words deep high priority data FIFO. The HIBI bus single burst transfer length is limited to 25 words. Each N2H2 has eight Rx channels, supports 256 separate addresses for channels and has a maximum packet size of 65536 32-bit words. HIBI bridges are not used, because there is no need for multiple clock domains. The MPEG-4 encoding SW exploits the MPSOC features as explained in the following sections.

5.3. Implementation of master's tasks

The tasks of the master are discussed in Section 4 and illustrated in the middle of Figure 7. All parameterization is performed via the external shared SDRAM. However, since N2H2 DMA is used to access the shared data memory, SDRAM, one cannot refer to SDRAM via pointers. For example, the C language statement *sdramVariable* = *pSdramAddr* [0] is not possible. Instead, data between external and local memories is moved with help of dedicated software library calls, for example, *sdramRead ( )* and *sdramWrite ( )*.

The current 64 KB limitation of the local data memory, however, presents a more demanding challenge considering that a raw QCIF image takes 37.1 KB. Our solution is to allocate large memory buffers, such as the currently encoded image, the reconstructed images, and the output bit buffers, on the external SDRAM. Two additional 1KB buffers are allocated on the on-chip RAM, which are used for processing data from the large buffers a small segment at a time.

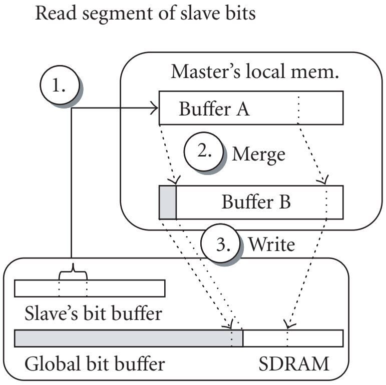


Figure 8: Master's bit stream merge task.

For example, bit stream merging is implemented as a three-step process, illustrated in Figure 8, which is repeated in a loop. As long as there are slave bits remaining, the master first reads a small portion of the slave's bit stream into the input buffer A. Second, the master concatenates and shifts the data after the tail of the master's global bit buffer in buffer B. Third, the master writes the result to the SDRAM and resynchronizes the buffer B with the updated tail of the global bit buffer. The tail contains the bits that did not form a full 32-bit word. The tail is stored to the SDRAM to keep the buffers synchronized, but the master uses the local copy to avoid unnecessary memory traffic. This allows merging of large, arbitrary sized bit streams, and realizes a general merging procedure for variable length coded (VLC) streams that are not aligned to byte boundaries. A similar buffering scheme is also used in the "send bits" phase of the master, except no bit shifting is needed since the bit stream is ensured to be byte aligned by the implementation.

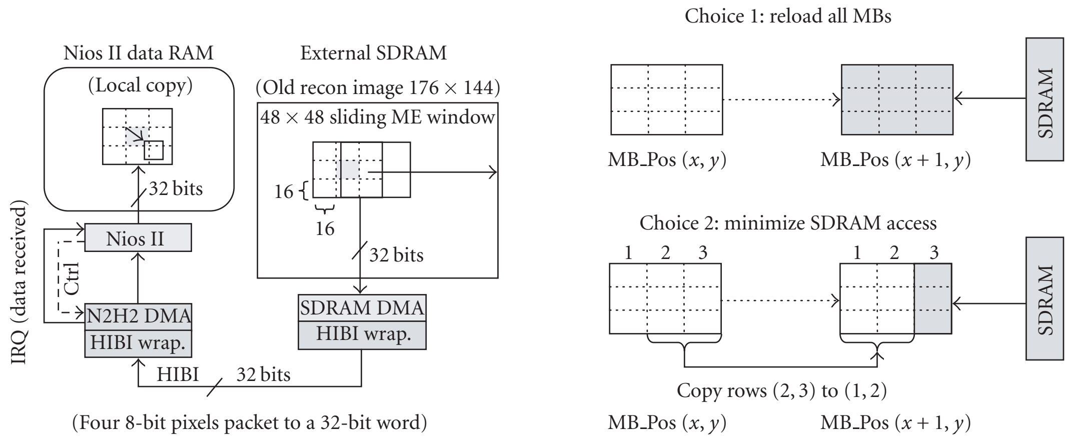
5.4. Implementation of slaves' tasks

Our MPEG-4 encoder software is identical to [29] except that all DSP specific optimizations have been omitted. A platform-independent portable ANSI-C implementation has been used for Nios II. A single program multiple data (SPMD)-approach has been used, so the slave programs are identical. The program execution is, however, greatly dependant on the data, so the execution flows differ from one CPU to another. Slave tasks are discussed in Section 4 and illustrated in the right side of Figure 7.

The slaves have to operate under low memory conditions. To circumvent this issue, the ME process is carried out in a 48×48 sliding window under the programmers control as illustrated in Figure 9(a). The ME window moves in a raster scan pattern centered on the current MB position. An exception is an image boundary, where the window is clipped inside the image. The ME window loading is optimized by packing four consecutive pixels into a 32-bit word. Furthermore, the overlapping of subsequent ME window positions could be used to minimize accesses to the external memory.

Figure 9(b) shows two approaches for updating the ME window. First, one can load a whole window from SDRAM every time the MB position changes, for example, with DMA. In the second approach, only the rightmost column is loaded from SDRAM while the remaining pixels are copied inside the on-chip memory. The data transmissions are executed beforehand in the background in order to minimize the time consumed by waiting for data to be processed.

The drawback of the first approach is high SDRAM bandwidth requirement. For example, the ME of a 4CIF video (704×576) at 30 frames/s demands 104 MB/s. In comparison, the second approach requires 34 MB/s but the drawback is that 136 million operations per second (MOPS) are consumed by load/store operations needed for copying. Considering that current SDRAM is fast, for example, 133 MHz



(a) Sliding ME window

(b) Window update approaches

Figure 9: (a) 48 × 48 sliding window, (b) ME window updating.

32-bit SDRAM yields maximum of 4 bytes ∗ 133 M = 507 MB/s, the first approach is still a viable option due to DMA [27]. Hence, the second approach is well suited for systems with slow SDRAM, while fast SDRAM and DMA can be used to reduce CPU utilization. In this work, we support the first approach because we have an efficient DMA that can be used to minimize the CPU load. When ME is carried out, the rest of the encoding is performed similar to a nonparallel encoder except that each slave works on a different slice. The slave bit output uses a local/global buffering scheme comparable to Figure 8.

6. RESULTS

The performance of our system is evaluated by measuring the FPGA utilization, the encoding frame rate as a function of the number of slaves, and the complexities of encoding tasks. All timing and profiling procedures for measurements are implemented with HW timers running at the CPU clock frequency, 50 MHz. The system was benchmarked in buffered mode, since the main concern is the pure video encoding speed with no Ethernet activity.

The measurements were carried out with two standard QCIF sequences *carphone* and *news*. The encoder was configured to use *IPPP...* frame structure, where only the first frame is Intra (I) coded while the rest are motion compensated Inter (P) frames. All tests were done in variable bit rate (VBR) mode where different bit rates are realized by changing QP. Video sequence relative input/output frame rates were fixed to 30 frames/s in all test cases.

During a benchmarking run, 120 frames of the selected test sequence were encoded. The average encoding frame rate (FPS_{avg}) is computed as

$$\text{FPS}_{\text{avg}}(n) = \frac{f_{\text{cpu}}}{c_{\text{frame}}(n)}, \tag{1}$$

where f_{cpu} is the CPU frequency and $C_{\text{frame}}(n)$ denotes average encoding cycles per QCIF frame with n encoding slaves. Due to the presence of data/instruction caches, IRQ processing, and the underlying on-chip network causing deviations in the results, three benchmarking runs were made and their average is reported as the result. Scalability is evaluated by computing the speed-up ($S(n)$) as

$$S(n) = \frac{\text{FPS}_{\text{avg}}(n)}{\text{FPS}_{\text{avg}}(1)}, \tag{2}$$

where $\text{FPS}_{\text{avg}}(n)$ is the average frame rate of a multislave configuration and $\text{FPS}_{\text{avg}}(1)$ is the average frame rate with a single slave. In addition, parallelization efficiency ($E(n)$) is computed as

$$E(n) = \left(\frac{\text{FPS}_{\text{avg}}(n)}{(n * \text{FPS}_{\text{avg}}(1))}\right) * 100\%. \tag{3}$$

6.1. FPGA utilization

Table 3 shows the FPGA utilization of the MPSOC HW modules. The area consumption is reported in terms of Logic Elements (LE) and mem usage is the utilization of the on-chip RAM. The statistics have been obtained by synthesizing MPSOC with Quartus II 5.0 into a Stratix 1S40. Currently, the maximum frequency (50 MHz) is dictated by the connection to external SRAM. Total memory and area refer to the maximum capacity of the FPGA chip. Memory figures are determined from the theoretical maximum number of the available memory bits. However, if we also count the bits that cannot be used due to the memory block architecture, the memory usage rises to 87% of the available memory resources. Therefore, the memory utilization restricts the system and not the logic utilization. The FIFO buffers in the system have been implemented with on-chip RAM. We have also implemented an FIFO using LE flip-flops as data storage. Thus, we can optionally save the on-chip RAM and use the spare LEs instead.

LE resources are abundant and have been exploited in the architecture. For example, in the SDRAM controller, there

Table 3: FPGA utilization statistics.

HW module	Module count	Total mem [KB]	% of total mem	Module area [LE]	Total area [LE]	% of LE
Master Nios I	1	80.8	19.3	2 720	2 720	6.6
Slave Nios II	3	225.1	53.9	2 324	6 972	16.9
N2H2 DMA	4	0	0	1 894	7 576	18.4
HIBI network	1	0.4	0.1	8 506	8 506	20.6
SDRAM DMA	1	0.3	0.1	3 205	3 205	7.8
Utilization		306.6	73.3		28 979	70.2

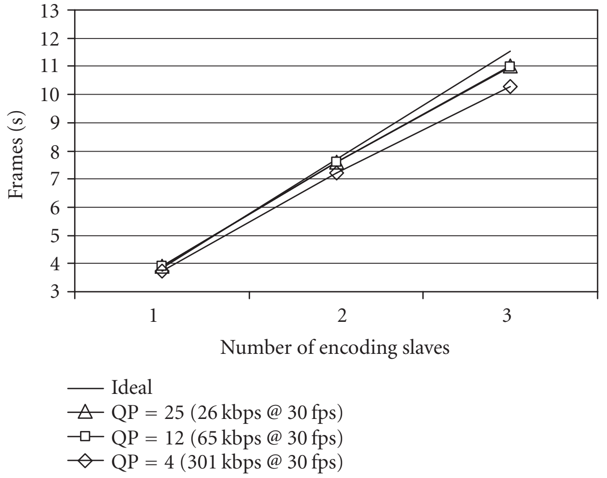


Figure 10: Frame rates for sequence carphone.qcif (176 × 144).

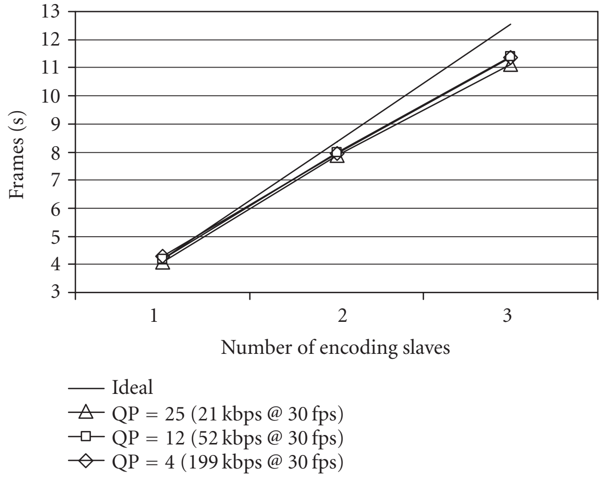


Figure 11: Frame rates for sequence news.qcif (176 × 144).

are four read and four write ports, ensuring that no CPU has to unnecessarily wait. N2H2 has eight channels to provide flexibility for the software programmer. There are spare LEs on the FPGA, since only 70% have been utilized.

6.2. Encoding speed and software scalability

Figures 10 and 11 present average encoding frame rates as a function of QP and the number of slaves for the *carphone* and *news* QCIF sequences. The bit rates are reported relative to the fixed 30 frames/s sequence output rate. The straight lines depict an ideal speed-up, which is obtained by multiplying the frame rate of a single slave with the number of slaves. The frame rates are measured from the master's main encoding loop.

As scalability was one of our main objectives, the results indicate very good success. The parallelization efficiency of *carphone* using two slaves is within 97% of the ideal result. If we further increase the number of slaves to three, the parallelization efficiency is 93%. As the current FPGA restricts the number of processors to four (one master and three slaves), we estimate the performance of larger configurations in the following.

6.2.1. Performance estimation for larger configurations

The complexity of image encoding task depends on slice encoding times as well as the overhead of the master. This information yields

$$C_{\text{img}}(x, y, n, C_{\text{mb}}) = C_{\text{slice}}(x, y, n, C_{\text{mb}}) + C_{\text{master}}(n), \tag{4}$$

where C_{img} is the clock cycles required to encode a frame, x and y are the width and height of the luma image in pixels, n is the number of encoding processors, and C_{mb} is the average clock cycles required to encode a macroblock. The term C_{master} denotes the master's overhead resulting from the sequentially computed parts. C_{slice} represents parallelized computations and is the number of clock cycles required to encode the largest slice in the system. Mathematically C_{slice} is computed as

$$C_{\text{slice}} = \left\lceil \frac{\lceil x/16 \rceil * \lceil y/16 \rceil}{n} \right\rceil * C_{\text{mb}}, \tag{5}$$

where the rounding for x and y takes care that the image is always divisible to macroblocks, for example, x and y do not need to be divisible by 16. The overall rounding finds the size of the largest slice in case the number of macroblocks is not evenly divisible by the number of processors.

The master's overhead results from four subfunctions which can be combined as

$$\begin{aligned} C_{\text{master}}(n) = {} & C_{\text{config}}(n) + C_{\text{getBitStreamSize}}(n) \\ & + C_{\text{merge}}(n) + C_{\text{oth}}(n), \end{aligned} \tag{6}$$

where C_{config} is due to the configuration of encoding parameters for the slaves, $C_{\text{getBitStreamSize}}$ results from reading the

Table 4: Measured clock cycles for Master's subfunctions.

f (subfunction)	1 slave CPU	2 slave CPUs	3 slave CPUs
Merge	32876.4150	38009.8750	43160.7267
Config	2821.6367	5313.8333	7878.8450
GetBitStreamSize	1264.2967	2517.6617	3772.6450
Oth	117016.7367	117465.5550	117982.0483

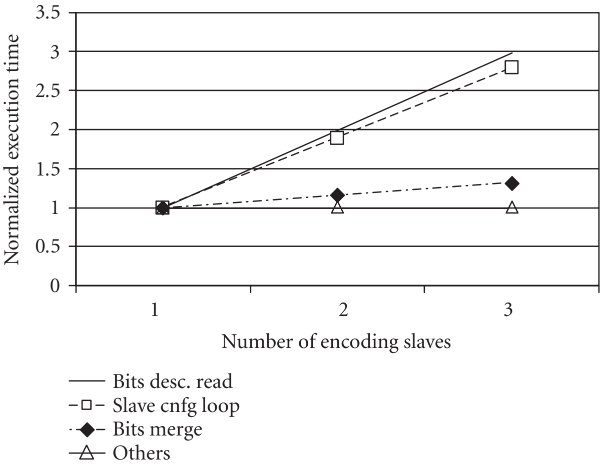


Figure 12: Growth rate of complexity of master's subtasks.

sizes of slave bit streams from SDRAM, C_{merge} is the number of clock cycles due to merging of slave bits streams, and others are related to the IRQs of the master and an internal state management. It is pointed out that for an optimized system, all Ethernet related tasks are omitted. The measured average clock cycles for the aforementioned subfunctions are presented in Table 4 as a function of the number of encoding processors. In Table 4, f identifies the subfunction.

For the mathematical model, it is necessary to model the growth of the master task complexity as a function of n. The complexity change is illustrated in Figure 12, which is plotted using the values in Table 4. For each subfunction, a curve was obtained by plotting the clock cycles with n encoding processors divided by the clock cycles required for one encoding CPU.

The results in Figure 12 show a linear increase in complexity for all subfunctions of the master. Therefore, the complexities of each subfunction as a function of n are approximated with

$$\begin{aligned} C_{f\in\{\text{merge, config, getBitStreamSize, oth}\}}(n) \\ = (a(f) * n + b(f)) * c(f), \end{aligned} \tag{7}$$

where a is the *slope* of the line (the gradient) and b is the *intercept* on the vertical axis in Figure 12, and c is the number of clock cycles of a subfunction with one encoding processor. In practice, the clock cycles of one encoding processor are scaled with the linear model to obtain a prediction for an n CPU system. The subfunction specific parameters for (7) are presented in Table 5.

Table 5: Parameterization of linear equations for complexity modeling.

f (subfunction)	$a(f)$	$b(f)$	$c(f)$ (CPU cycles)
Merge	0.1564	0.8436	32876.4150
Config	0.8961	0.1039	2821.6367
GetBitStreamSize	0.9920	0.0080	1264.2967
Oth	0.0041	0.9959	117016.7367

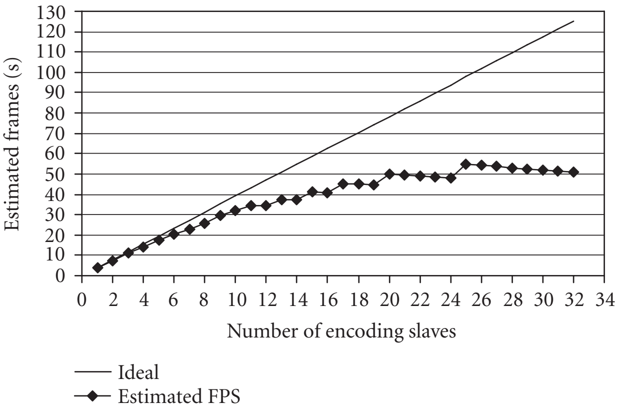


Figure 13: Estimated frame rate for n-processor MPSOC system.

Due to the simultaneous access to the shared data memory at the beginning of each frame encoding, the slave's start-up latency, that is, the time to get enough data to start processing, also increases as the number of slaves increase. This time is not included in the estimate. Each slave requires one motion estimation window, 2560 bytes (640 words), to start processing. It can be assumed that this amount can be transferred from SDRAM to CPU in around 1000 cycles. Thus, since the frame encoding time is millions of clock cycles, the impact is quite insignificant.

Finally, the encoding frame rate estimation on the MPEG-4 MPSOC system is computed with

$$\text{FPS}_{\text{MPSOC}}(x, y, n, C_{\text{mb}}, f_{\text{cpu}}) = \frac{f_{\text{cpu}}}{C_{\text{imge}}(x, y, n, C_{\text{mb}})}, \tag{8}$$

where f_{cpu} is the clock frequency of 50 MHz. With benchmarking it was found that C_{mb} is on the average of 133394.8 clock cycles per macroblock for *carphone* if a QP value of 12 is used.

Figure 13 presents the predicted encoding frame rate for the optimized MPSOC as a function of n for the QCIF video format. The values are obtained with (8) using the parameters in Table 5. The system scales nearly linearly when n is smaller than 12. After 12 encoding processors, the complexity of the master's sequential overhead starts to increase faster than is the benefit of data parallelization and the frame rate saturates after 24 slaves. The small variation at large n is due to the unbalanced sizes of slices.

One solution to smooth out the variations would be to use a finer subdivision granularly, for example, 8 × 8 or 4 × 4 blocks, but this is impractical from an implementation

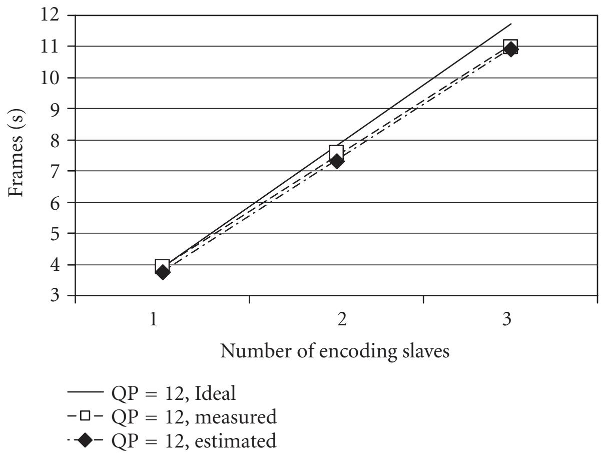


Figure 14: Ideal, measured, and estimated frame rates for carphone.qcif (176 × 144).

point of view since a macroblock would span two processors. In practice, advanced horizontal parallelization scales better than the row-wise approach used in [27] due to finer granularity.

Figure 14 shows that the model applies well to real, measured performance. It is slightly lower than the measured frame rates with the maximum error being about 4%. The model is applicable to QCIF video streams and also for larger formats by changing the measured execution times appropriately. It takes into account the number of macroblocks and as the number of processors increases, the sizes of slices may not be equal in terms of number of macroblocks resulting in computational unbalance. However, if we use a larger video size, for example CIF, the number of macroblocks also increases. Therefore, with larger video sizes, we can benefit from having more processors than is currently practical for QCIF.

6.3. Relative complexity of encoding tasks

The complexities of different CPU tasks show how computing power is shared within one processor. For the master processor, up to 96% of the time is spent waiting the slaves to complete. In buffered mode, frame transmissions over the Ethernet are not executed until the whole video sequence is encoded and thus, this time is not included in the utilization figures. In this case, the master operates as a state machine. However, the master processor is designed to handle all the external tasks that are not directly related to the video encoding. This can include I/O handling (e.g., Ethernet protocol stack), audio coding, and user interfaces, like changing quantization parameters at run-time. The greatest requirement is fast response time for the user interface processor. Double buffering could also be used.

Figure 15 illustrates how the execution time is divided for one slave processor to encode one macroblock. *Motion estimation* (ME) is by far the most computationally challenging task. Other time consuming tasks are *interpolation, quantization* (Q), *inverse quantization* (IQ), *DCT*, and *IDCT*. The time for *MasterWait* + *Poll* is consumed by waiting for the master to collect and merge the bit stream, deliver a new raw slice, and provide the MPEG-4 coding parameters for next frame.

6.4. Hardware scalability

HIBI offers good scalability. For this architecture, adding or removing processors is simple—it only takes minutes to parameterize and prepare for synthesis. HIBI also offers a convenient way to add new IP components, for example new processors or hardware accelerators. It is possible to add new features to the system without altering the video encoding. A simple example is the addition of a hardware utilization monitor. The addition does not require any changes to the encoding and it is easy to plug into the system by just adding one HIBI wrapper to the interconnection. Encouraging preliminary results have been obtained from attaching a hardware accelerator (DCT, IDCT, quantizer, and inverse quantizer) to the system (around a couple of FPS increase).

The utilization of the HIBI bus is measured to be only 3%. Therefore, the interconnection is not likely to become a bottleneck even with larger configurations. From a performance point of view, the interconnection architecture should be as invisible as possible in hiding data transmission time from the CPU.

The speed-up gained by adding processors (e.g., Figure 10) shows that the interconnection architecture performs well. Figure 15 shows that only 4% (SDRAM config + comm wait) of the CPU encoding time is spent on data transmissions and waiting for data.

As the HIBI utilization is low, it is expected that the shared data memory (SDRAM) will not become the bottleneck in the future. We have determined that the interconnection is capable of transmitting the required data for a CIF image, which is four times larger at 25 frames/s using a clock frequency of 28 MHz without forming a significant performance bottleneck. The application can perform data fetches from memory in parallel with computation. Therefore, memory latencies can be tolerated.

Shared instruction memory utilization, via the Avalon bus, is at most 20% of available bandwidth. We have investigated the effect of shared instruction memory on performance and preliminary results indicate that it is currently negligible with respect to total frame encoding time. However, four processors could share one instruction memory port. The absolute maximum for sufficient performance [30] is ten processors per port.

6.5. Comparison of scalability to related work

In Figure 16, the scalability of different video encoders is presented. The optimal situation would be a gain of 100% parallel efficiency, that is, a speed-up of 2.0 for 2 CPUs and 3.0 for 3 CPUs. Of the four general categories of parallelization, results from three are presented. No publications were found that describe a functional parallel encoder that scales with a varying number of processors.

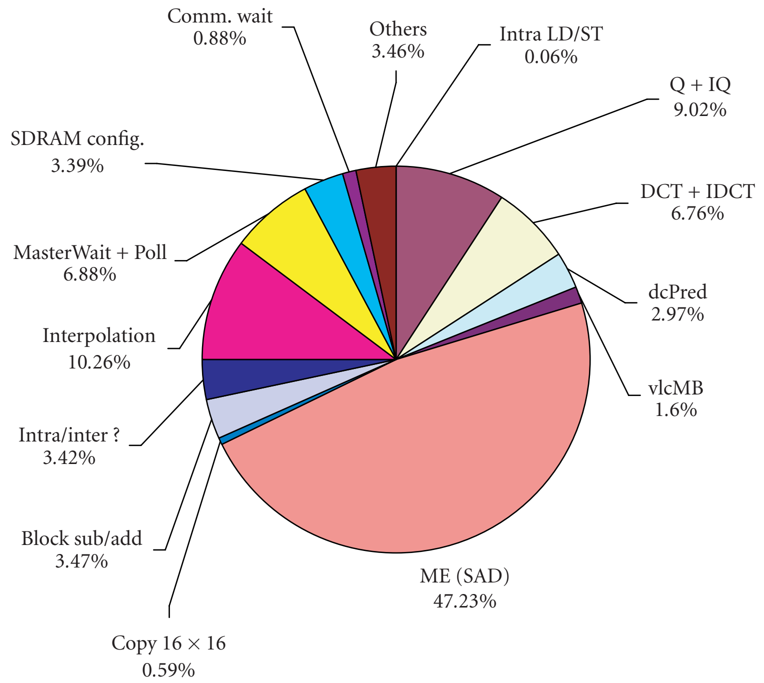


Figure 15: Relative complexities of encoding tasks on one slave CPU.

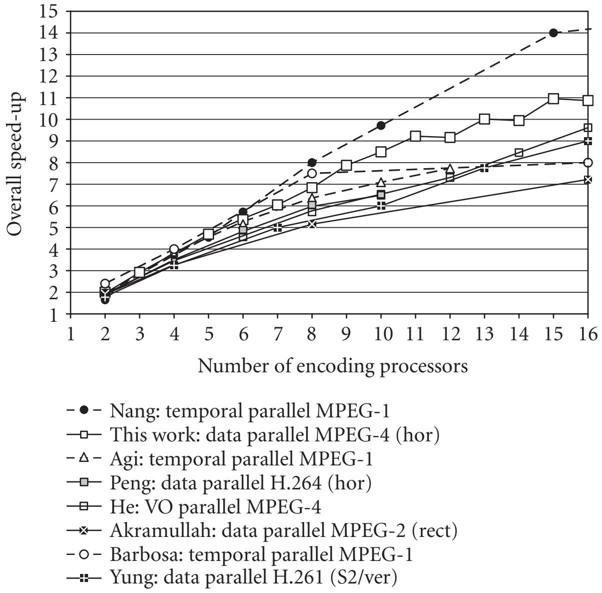


Figure 16: Speed-ups of different video encoder implementations.

Nang and Kim [8], Agi and Jagannathan [9], and Barbosa et al. [10] use temporal parallelism. Nang et al. have received great parallelization efficiency. However, as temporal parallelism needs frames to be buffered (increasing the latency) it is not considered to be low-latency real-time video encoding. In our work we present a data parallel application. Similar works are Peng and Zhao [11], Akramullah et al. [12], and Yung and Leung [13]. One work based on video-object parallelism, which is closely related to data parallelism, is presented in He et al. [14].

The results from related research are frequently presented as speed-up or frame rate curves, so the exact numerical values cannot be obtained. We have reproduced the curves in Figure 16 to be as accurate as possible. Results from Nang and Kim, Agi and Jagannathan, He et al., and Yung and Leung are plotted. Barbosa et al. have given results for three cases. The result for 16 CPUs is estimated from the sketched figure. Our results are from the model described earlier. Results for Peng and Zhao and Akramullah et al. are plotted from the given numerical figures.

Figure 16 shows that the scalability of our implementation is the highest of all data parallel implementations. That implies both good parallelization efficiency achieved with software and an efficient hardware architecture.

7. CONCLUSIONS

A highly scalable MPSOC architecture with an MPEG-4 encoder has been presented. The parallelization efficiency of the application, when the number of encoding processors is increased from one to two, is 97% and to three, 93%. No real-time video encoder was found that has such a high scalability for real-time video encoding. Our benefit is due to both a well-designed architecture and application. The architecture efficiently hides the data transmissions from the processors. The software takes full advantage of the parallelism by elegantly sharing the encoding load. The software does not need any changes when the number of processors is altered, thus the scaling effort is very low.

The scalability and flexibility of the MPSOC architecture is gained by using an HIBI on-chip network and softcore processors in a plug-and-play fashion. The performance and area usage can be flexibly compromised by changing the number of processors. It takes only minutes to change the number of processors and then the new system is ready to be synthesized. Since the architecture is implemented in an FPGA, the amount of on-chip memory becomes a limiting factor.

In the future, platform flexibility will be demonstrated with the use of different soft-core processors and hardware accelerators. The connection to external instruction memory will be replaced with an HIBI interface to achieve better clock frequencies. Furthermore, scalability will be further evaluated with larger FPGA chips and by connecting several ones together to form a larger system.

REFERENCES

[1] E. Salminen, A. Kulmala, and T. D. Hämäläinen, "HIBI-based multiprocessor SoC on FPGA," in *IEEE International Symposium on Circuits and Systems (ISCAS '05)*, pp. 3351–3354, Kobe, Japan, May 2005.

[2] O. Lehtoranta, E. Salminen, A. Kulmala, M. Hännikäinen, and T. D. Hämäläinen, "A parallel MPEG-4 encoder for FPGA based multiprocessor SoC," in *15th International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 380–385, Tampere, Finland, August 2005.

[3] Altera Corporation, *Nios Development Board: Reference Manual, Stratix Professional Edition*, Ver. 1.1, July 2003.

[4] E. Salminen, T. Kangas, J. Riihimäki, V. Lahtinen, K. Kuusilinna, and T. D. Hämäläinen, "HIBI v.2 communication network for system-on-chip," in *Computer Systems: Architectures, Modeling, and Simulation*, A. D. Pimentel and S. Vassiliadis, Eds., vol. 3133 of *Lecture Notes in Computer Science*, pp. 412–422, Springer, Berlin, Germany, July 2004.

[5] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*, Kluwer Academic, Dordrecht, The Netherlands, 1999.

[6] I. Ahmad, Y. He, and M. L. Liou, "Video compression with parallel processing," *Parallel Computing*, vol. 28, no. 7-8, pp. 1039–1078, 2002.

[7] O. Cantineau and J.-D. Legat, "Efficient parallelisation of an MPEG-2 codec on a TMS320C80 video processor," in *IEEE International Conference on Image Processing (ICIP '98)*, vol. 3, pp. 977–980, Chicago, Ill, USA, October 1998.

[8] J. Nang and J. Kim, "An Effective parallelizing scheme of MPEG-1 video encoding on ethernet-connected workstations," in *Proceedings of the Conference on Advances in Parallel and Distributed Computing (APDC '97)*, pp. 4–11, Shanghai, China, March 1997.

[9] I. Agi and R. Jagannathan, "A portable fault-tolerant parallel software MPEG-1 encoder," *Multimedia Tools and Applications*, vol. 2, no. 3, pp. 183–197, 1996.

[10] D. M. Barbosa, J. P. Kitajima, and W. Weira Jr., "Parallelizing MPEG video encoding using multiprocessors," in *Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI '99)*, pp. 215–222, Sao Paulo, Brazil, October 1999.

[11] Q. Peng and Y. Zhao, "Study on parallel approach in H.26L video encoder," in *4th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '03)*, pp. 834–837, Chengdu, China, August 2003.

[12] S. M. Akramullah, I. Ahmad, and M. L. Liou, "Performance of software-based MPEG-2 video encoder on parallel and distributed systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 4, pp. 687–695, 1997, Transaction Briefs.

[13] N. H. C. Yung and K.-K. Leung, "Spatial and temporal data parallelization of the H.261 video coding algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 1, pp. 91–104, 2001.

[14] Y. He, I. Ahmad, and M. L. Liou, "MPEG-4 based interactive video using parallel processing," in *International Conference on Parallel Processing (ICPP '98)*, pp. 329–336, Minneapolis, Minn, USA, August 1998.

[15] M. J. Garrido, C. Sanz, M. Jiménez, and J. M. Menasses, "An FPGA implementation of a flexible architecture for H.263 video coding," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 4, pp. 1056–1066, 2002.

[16] R. Kordasiewicz and S. Shirani, "Hardware implementation of the optimized transform and quantization blocks of H.264," in *Canadian Conference on Electrical and Computer Engineering (CCECE '04)*, vol. 2, pp. 943–946, Niagara Falls, Ontario, Canada, May 2004.

[17] Q. Peng and J. Jing, "H.264 codec system-on-chip design and verification," in *5th International Conference on ASIC (ASICON '03)*, vol. 2, pp. 922–925, Beijing, China, October 2003.

[18] Y.-L. Lin, C.-P. Young, Y.-J. Chang, Y.-H. Chung, and A. W. Y. Su, "Versatile PC/FPGA based verification/fast prototyping platform with multimedia applications," in *Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference (IMTC '04)*, vol. 2, pp. 1490–1495, Como, Italy, May 2004.

[19] M. Martina, A. Molino, and F. Vacca, "FPGA system-on-chip soft IP design: a reconfigurable DSP," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems (MWSCAS '02)*, vol. 3, pp. 196–199, Tulsa, Okla, USA, August 2002.

[20] X. Wang and S. G. Ziavras, "Parallel direct solution of linear equations of FPGA-based machines," in *IEEE International Parallel & Distributed Processing Symposium (IPDPS '03)*, pp. 113–120, Nice, France, April 2003.

[21] Altera Corporation, "Avalon Interface Specification," Ver. 2.4, January 2004.

[22] H. Kalte, D. Langen, E. Vonnahme, A. Brinkmann, and U. Rückert, "Dynamically reconfigurable system-on-programmable-chip," in *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP '02)*, pp. 235–242, Canary Islands, Spain, January 2002.

[23] OCP-IP Alliance, "Open Core Protocol specification," Release 2.0, 2003.

[24] Altera Corporation, "Nios 3.0 CPU datasheet," October 2004.

[25] Altera Corporation, "Nios II Processor Reference Handbook," May 2005.

[26] Altera Corporation, Nios II, Site visited 28.11.2005, http://www.altera.com/products/ip/processors/nios2/ni2-index.html.

[27] O. Lehtoranta, T. D. Hämäläinen, V. Lappalainen, and J. Mustonen, "Parallel implementation of video encoder on quad DSP system," *Microprocessors and Microsystems*, vol. 26, no. 1, pp. 1–15, 2002.

[28] Altera Corporation, "Stratix Device Handbook," January 2005.

[29] O. Lehtoranta and T. D. Hämäläinen, "Feasibility study of a real-time operating system for a multichannel MPEG-4 encoder," in *Multimedia on Mobile Devices*, vol. 5684 of *Proceedings of SPIE*, pp. 292–299, San Jose, Calif, USA, January 2005.

[30] A. Kulmala, E. Salminen, O. Lehtoranta, T. D. Hämäläinen, and M. Hännikäinen, "Impact of shared instruction memory on performance of FPGA-based MP-SoC video encoder," in *The 9th IEEE workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS '06)*, pp. 59–64, Prague, Czech Republic, April 2006.

Ari Kulmala received the M.S. degree in computer science from the Tampere University of Technology (TUT), Finland, in August 2005. He is currently pursuing Ph.D. degree and working as a research scientist in the DACI Research Group in the Institute of Digital and Computer Systems at TUT. His research interests include system-on-chip architectures, interconnections, and low power design.

Olli Lehtoranta was born in Vantaa, Finland, on 2nd of June 1976. He received his M.S. degree "with distinction" in computer science from Tampere University of Technology, in November 2001. In 1999, he joined the Institute of Digital and Computer Systems Laboratory where he currently works towards his Ph.D. degree. He has authored two journals and nine conference papers for refereed scientific publications. His current research interests include parallel processing for video encoding, error resilient video encoding methods, computational complexity analysis of video encoding algorithms, and low-level assembly optimization of video codecs for media instruction set architectures (ISA) as well as DSP.

Timo D. Hämäläinen received the M.S. degree in 1993 and the Ph.D. degree in 1997, both from Tampere University of Technology (TUT). He acted as a Senior Research Scientist and Project Manager at TUT during 1997–2001. He was nominated to be Full Professor at TUT, Institute of Digital and Computer Systems, in 2001. He heads the DACI Research Group that focuses on three main lines: wireless local area networking and wireless sensor networks, high-performance DSP/HW-based video encoding, and interconnection networks with design flow tools for heterogeneous SoC platforms.

Marko Hännikäinen received the M.S. degree in 1998 and the Ph.D. degree in 2002, both from Tampere University of Technology (TUT). Currently he acts as a Senior Research Scientist in the Institute of Digital and Computer Systems at TUT, and a Project Manager in the DACI Research Group. His research interests include wireless local and personal area networking, wireless sensor and ad hoc networks, and novel web services.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 16035, Pages 1–12
DOI 10.1155/ES/2006/16035

A Real-Time Wavelet-Domain Video Denoising Implementation in FPGA

Mihajlo Katona,[1] Aleksandra Pižurica,[2] Nikola Teslić,[1] Vladimir Kovačević,[1] and Wilfried Philips[2]

[1] *Chair for Computer Engineering, University of Novi Sad, Fruškogorska 11, 21000 Novi Sad, Serbia and Montenegro*
[2] *Department of Telecommunications and Information Processing, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium*

Received 15 December 2005; Accepted 13 April 2006

The use of field-programmable gate arrays (FPGAs) for digital signal processing (DSP) has increased with the introduction of dedicated multipliers, which allow the implementation of complex algorithms. This architecture is especially effective for data-intensive applications with extremes in data throughput. Recent studies prove that the FPGAs offer better solutions for real-time multiresolution video processing than any available processor, DSP or general-purpose. FPGA design of critically sampled discrete wavelet transforms has been thoroughly studied in literature over recent years. Much less research was done towards FPGA design of overcomplete wavelet transforms and advanced wavelet-domain video processing algorithms. This paper describes the parallel implementation of an advanced wavelet-domain noise filtering algorithm, which uses a nondecimated wavelet transform and spatially adaptive Bayesian wavelet shrinkage. The implemented arithmetic is decentralized and distributed over two FPGAs. The standard composite television video stream is digitalized and used as a source for real-time video sequences. The results demonstrate the effectiveness of the developed scheme for real-time video processing.

1. INTRODUCTION

Video denoising is important in numerous applications, such as television broadcasting systems, teleconferencing, video surveillance, and restoration of old movies. Usually, noise reduction can significantly improve visual quality of a video as well as the effectiveness of subsequent processing tasks, like video coding.

Noise filters that aim at a high visual quality make use of both spatial and temporal redundancy of video. Such filters are known as spatio-temporal or three-dimensional (3D) filters. Often 2D spatial filter and 1D temporal filter are applied separately, and usually sequentially (because spatial denoising facilitates motion detection and estimation). Temporal filtering part is often realized in a recursive fashion in order to minimize the memory requirements. Numerous existing approaches range from lower complexity solutions, like 3D rational [1] and 3D order-statistic [2, 3] algorithms to sophisticated Bayesian methods based on 3D Markov models [4, 5].

Multiresolution video denoising is one of the increasingly popular research topics over recent years. Roosmalen et al. [6] proposed video denoising by thresholding the coefficients of a specific 3D multiresolution representation, which combines 2D steerable pyramid decomposition (of the spatial content) and a 1D wavelet decomposition (in time). Related to this, Selesnick and Li [7] investigated wavelet thresholding in a nonseparable 3D dual-tree complex wavelet representation. Rusanovskyy and Egiazarian [8] developed an efficient video denoising method using a 3D sliding window in the discrete cosine transform domain. Other recent multiresolution schemes employ separable spatial/temporal filters, where the temporal filter is motion adaptive recursive filter. Such schemes were proposed, for example, by Pižurica et al. [9] where a motion selective temporal filter follows the spatial one, and by Zlokolica et al. [10] where a motion-compensated temporal filter precedes the spatial one. Less research was done so far towards hardware design of these multiresolution video denoising schemes.

The use of the FPGAs for digital signal processing has increased with the introduction of dedicated multipliers, which facilitate the implementation of complex DSP algorithms. Such architectures are especially effective for data-intensive applications with extremes in data throughput. With examples for video processing applications Draper et al. [11] present performance comparison of FPGA and general-purpose processors. Similarly, Haj [12] illustrates two different wavelet implementations in the FPGAs and compares these with general-purpose and DSP processors. Both studies come to the conclusion that the FPGAs are far more suitable for real-time video processing in the wavelet domain than any available processor, DSP or general-purpose.

The hardware implementation of the wavelet transform is related to the finite-impulse-response (FIR) filter design. Recently, the implementation of FIR filters has become quite common in the FPGAs. A detailed guide for the FPGA filter design is in [13] and techniques for area optimized implementation of FIR filters are presented, for example, in [14]. A number of different techniques for implementing the critically sampled discrete wavelet transform (DWT) in the FPGAs exist [15–21] including the implementation of MPEG-4 wavelet-based visual texture compression system [22]. Recently, the lifting scheme [23–25] is introduced for real-time DWT [20, 26] as well as the very-large-scale-integration (VLSI) implementation of the DWT using embedded instruction codes for symmetric filters [27]. The lifting scheme is attractive for hardware implementations because it replaces multipliers with shift operations. The FPGA implementations of overcomplete wavelet transforms are much less studied in literature.

Our initial techniques and results in FPGA implementation of wavelet-domain video denoising are in [28, 29]. These two studies were focusing on different aspects of the developed system: implementation of the wavelet transform and distributed computing over the FPGA modules in [28] and customization of a wavelet shrinkage function by look-up tables for implementation in read-only-memories (ROMs) [29]. The description was on a more abstract level focusing on the main concepts and not on the details of the architectural design.

In this paper, we report a full architectural design of a real-time FPGA implementation of a video denoising algorithm based on an overcomplete (nondecimated) wavelet transform and employing sophisticated locally adaptive wavelet shrinkage. We propose a novel FIR filter design for the nondecimated wavelet transform based on the algorithm *à trous* [30]. The implemented spatial/temporal filter is separable, where a motion-adaptive recursive temporal filter follows the spatial filter as was proposed in [9]. We present an efficient customization of the locally adaptive spatial wavelet filter using a combination of read-only-memories (ROMs) and a dedicated address generation network. We design an efficient implementation of a local window for wavelet processing using an array of delay elements. Our design of the complete denoising scheme distributes computing over two FPGA modules, which switch their functionality in time: while one module performs the direct wavelet transform of the current frame, the other module is busy with the inverse wavelet transform of the previous frame. After each two frames, the functioning of the two modules is reversed. We present a detailed data flow of the proposed scheme. For low-to-moderate noise levels, the designed FPGA implementation yields a minor performance loss compared to the software version of the algorithm. This proves the potentials of the FPGAs for real-time implementations of highly sophisticated and complex video processing algorithms.

The paper is organized as follows. Section 2 presents an overview of the proposed FPGA design, including the memory organization (Section 2.1) and data flow (Section 2.2). Section 3 details the FPGA design of the different building blocks in our video denoising scheme. We start with some preliminaries for the hardware design of the nondecimated wavelet transform (Section 3.1) and present the proposed pipelined FPGA implementation (Section 3.2). Next, we present the FPGA design of the locally adaptive wavelet shrinkage (Section 3.3) and finally the FPGA implementation of the motion-adaptive recursive temporal filter (Section 3.4). Section 4 presents the real-time environment used in this study. The conclusions are in Section 5.

2. REAL-TIME IMPLEMENTATION WITH FPGA

An overview of our FPGA implementation is illustrated in Figure 1. We use two independent modules working in parallel. Each module is implemented in a separate FPGA. While one module performs the wavelet decomposition of an input TV frame, the other module performs the inverse wavelet transform of the previous TV frame. The two modules switch their functionality in time. The wavelet-domain denoising block is located in front of the inverse wavelet transform.

The proposed distributed algorithm implementation over the two modules allows effective logic decentralization with respect to input and output data streams. Namely, while one FPGA module is handling the input video stream performing the wavelet decomposition, the other FPGA module is reading the wavelet coefficients for denoising, sending them to the wavelet reconstruction, and building up the visually improved output video stream.

2.1. Memory organization

The nondecimated wavelet transform demands significant memory resources. For example, in our implementation with three decomposition levels we need to store nine frames of wavelet coefficients for every input frame. In addition, we need an input memory buffer and an output buffer for isolating data accesses from different clock domains.

The input data stream is synchronized with a 13.5 MHz clock. For three decomposition levels the complete wavelet decomposition and reconstruction has to be completed with the clock of at least $3 \times 13.5 = 40.5$ MHz. The set-up of our hardware platform requires the output data stream at 27 MHz. Table 1 lists the required interfaces of the buffers that are used in the system.

The most critical timing issue is at the memory buffer for storing the wavelet coefficients. It has to provide simultaneous *read* and *write* options at 40.5 MHz. Due to lack of the SDRAM controller that supports this timing issue, the whole processing is split in two independent parallel modules. The idea is to distribute the direct and the inverse wavelet processing between these modules. While one module is performing the wavelet decomposition of the current frame, the other module is performing the inverse wavelet transform of the

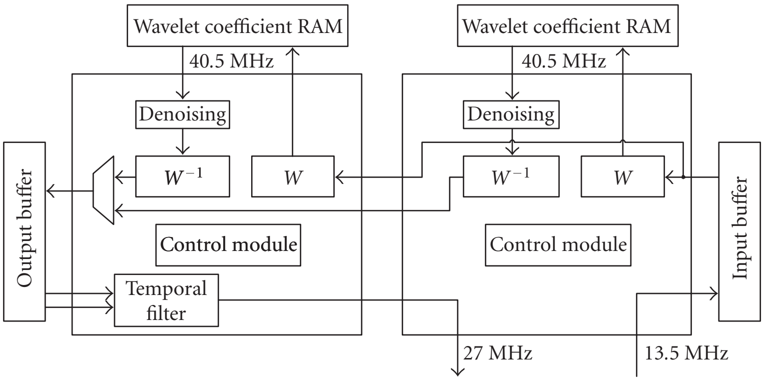


Figure 1: A detail of the FPGA implementation of the proposed wavelet-domain video denoising algorithm.

Table 1: Memory interfaces.

Buffers	Write port (MHz)	Read port (MHz)
Input buffer	13.5	40.5
Wavelet coefficients buffer	40.5	40.5
Output buffer	40.5	27

previous frame. With such organization, one module reads and the other module writes the coefficients. The approximation subband (LL band) during the wavelet decomposition and composition is stored in the onboard SRAM memory. This allows us to use only *read* or *write* memory port during one frame.

2.2. Data flow

The data flow through all the memory buffers and both FPGA's in our scheme is shown in Figure 2. The total delay is 4 frames. During the first 20 milliseconds, the input frame A_0 is stored in the input buffer at a clock rate of 13.5 MHz. During the next 20 milliseconds, this frame is read from the input buffer and is wavelet transformed in a 40.5 MHz clock domain, with 3 decomposition scales $W_1(A_0)$, $W_2(A_0)$, and $W_3(A_0)$. In parallel to this process, the next frame A_1 is written in the input buffer. The following time slot of 20 milliseconds is currently not used for processing A_0, but is reserved for future additional processing in the wavelet domain. Within this period the frame A_1 is read from the input buffer and is decomposed in its wavelet coefficients. The frames A_0 and A_1 are processed by FPGA1. The next input frame, A_2, is written in the input buffer, and is wavelet transformed in the next time frame by FPGA2.

The denoising and the inverse wavelet transform of the frame A_0 are performed afterwards. During this period the wavelet coefficients of the frame A_0 are read from the memory, denoised and the output frame is reconstructed with the inverse wavelet transform $W^{-1}(A_0)$. During the last reconstruction stage (the reconstruction at the finest wavelet scale), the denoised output frame is written to the output memory buffer. Parallel to this process, FPGA2 performs the wavelet decomposition of the frame A_2 and the input frame A_3 is stored in the input buffer.

Finally, 4×20 milliseconds $= 80$ milliseconds after the frame A_0 appeared at the system input (4 frames later), it is read from the output buffer in a 27 MHz clock domain and is sent to the selective recursive temporal filter and to the system output afterwards. The output data stream is aligned with a 100 Hz refresh rate, which means that the same frame is sent twice to the output within one time frame of 20 milliseconds. Additionally, FPGA2 performs the wavelet decomposition of the frame A_3. Further on, A_4 frame is written to the input buffer and is decomposed in the following time frame under the control of FPGA1.

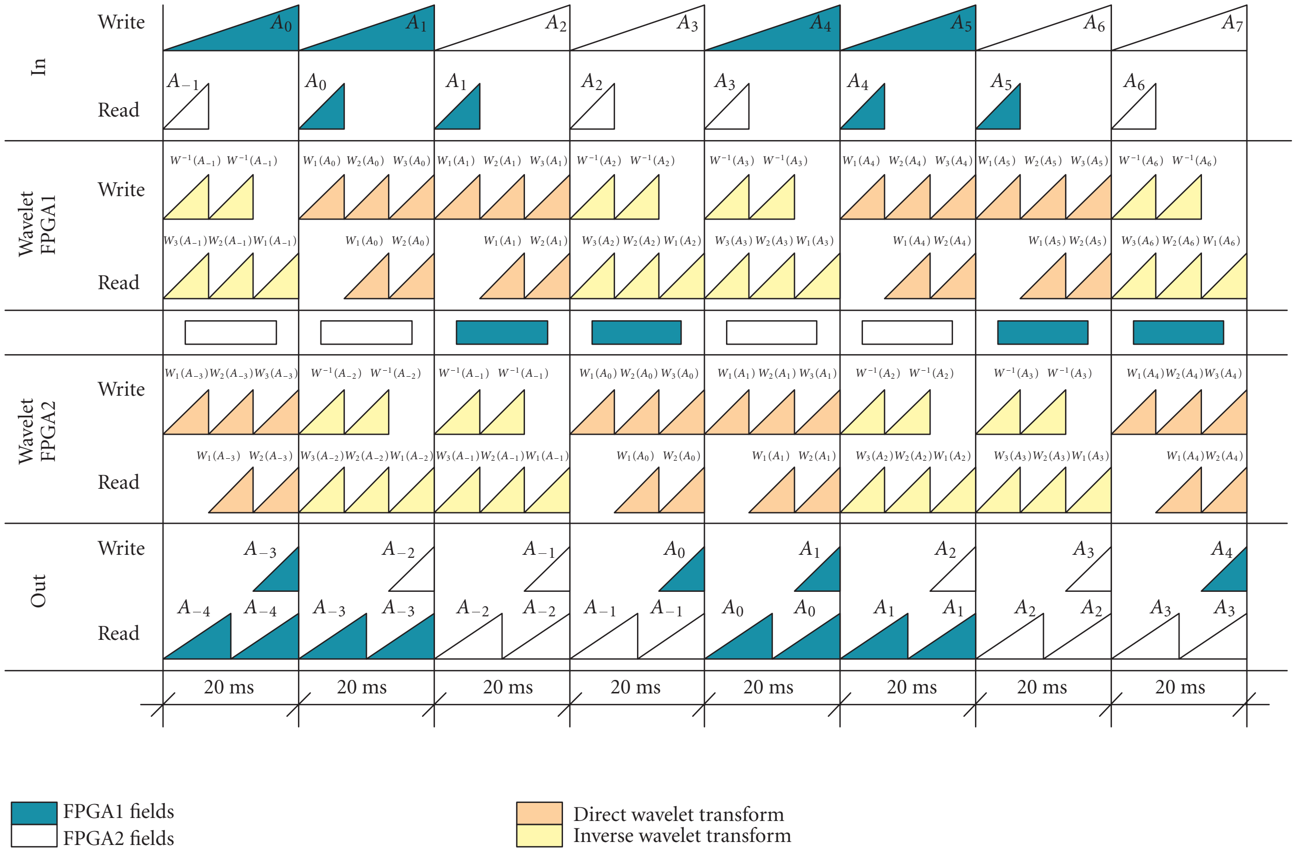
In this scheme, the two FPGAs actually switch their functionality after each two frames. The FPGA1 performs the wavelet decomposition for two frames, while the FPGA2 performs the inverse wavelet transform of the previous two frames. After two frames, this is reversed.

3. ALGORITHM CUSTOMIZATION FOR REAL-TIME PROCESSING

We design an FPGA implementation of a sequantial spatial/temporal video denoising scheme from [9], which is depicted in Figure 3. Note that we use an overcomplete (nondecimated) wavelet transform to guarantee a high-quality spatial denoising. In this representation, with three decomposition levels the number of the wavelet coefficients is 9 times the input image size. Therefore we choose to perform the temporal filtering in the image domain (after the inverse wavelet transform) in order to minimize the memory requirements.

3.1. The customization of the wavelet transform

While hardware implementations of the orthogonal wavelet transform have been extensively studied in literature [16–21, 26, 27], much less research has been done towards implementations of the nondecimated wavelet transform. We develop a hardware implementation of the non-decimated wavelet transform based on the algorithm *à trous* [30] and with the classical three orientation subbands per scale. This


In
Write
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7
Read
A_{-1} A_0 A_1 A_2 A_3 A_4 A_5 A_6
Wavelet FPGA1
Write
$W^{-1}(A_{-1})$ $W^{-1}(A_{-1})$
$W_1(A_0)$ $W_2(A_0)$ $W_3(A_0)$
$W_1(A_1)$ $W_2(A_1)$ $W_3(A_1)$
$W^{-1}(A_2)$ $W^{-1}(A_2)$
$W^{-1}(A_3)$ $W^{-1}(A_3)$
$W_1(A_4)$ $W_2(A_4)$ $W_3(A_4)$
$W_1(A_5)$ $W_2(A_5)$ $W_3(A_5)$
$W^{-1}(A_6)$ $W^{-1}(A_6)$
Read
$W_3(A_{-1})W_2(A_{-1})W_1(A_{-1})$
$W_1(A_0)$ $W_2(A_0)$
$W_1(A_1)$ $W_2(A_1)$
$W_3(A_2)W_2(A_2)W_1(A_2)$
$W_3(A_3)W_2(A_3)W_1(A_3)$
$W_1(A_4)$ $W_2(A_4)$
$W_1(A_5)$ $W_2(A_5)$
$W_3(A_6)W_2(A_6)W_1(A_6)$
Wavelet FPGA2
Write
$W_1(A_{-3})W_2(A_{-3})W_3(A_{-3})$
$W^{-1}(A_{-2})$ $W^{-1}(A_{-2})$
$W^{-1}(A_{-1})$ $W^{-1}(A_{-1})$
$W_1(A_0)W_2(A_0)W_3(A_0)$
$W_1(A_1)W_2(A_1)W_3(A_1)$
$W^{-1}(A_2)$ $W^{-1}(A_2)$
$W^{-1}(A_3)$ $W^{-1}(A_3)$
$W_1(A_4)W_2(A_4)W_3(A_4)$
Read
$W_1(A_{-3})$ $W_2(A_{-3})$
$W_3(A_{-2})W_2(A_{-2})W_1(A_{-2})$
$W_3(A_{-1})W_2(A_{-1})W_1(A_{-1})$
$W_1(A_0)$ $W_2(A_0)$
$W_1(A_1)$ $W_2(A_1)$
$W_3(A_2)W_2(A_2)W_1(A_2)$
$W_3(A_3)W_2(A_3)W_1(A_3)$
$W_1(A_4)W_2(A_4)$
Out
Write
A_{-3} A_{-2} A_{-1} A_0 A_1 A_2 A_3 A_4
Read
A_{-4} A_{-4} A_{-3} A_{-3} A_{-2} A_{-2} A_{-1} A_{-1} A_0 A_0 A_1 A_1 A_2 A_2 A_3 A_3
20 ms
20 ms
20 ms
20 ms
20 ms
20 ms
20 ms
20 ms
FPGA1 fields
FPGA2 fields
Direct wavelet transform
Inverse wavelet transform

$W_j(A_i)$-wavelet decompositions at scale j
$W^{-1}(A_i)$-wavelet reconstruction of the frame A_i
A_j-processing frame with index i

Figure 2: The data flow of wavelet processing.

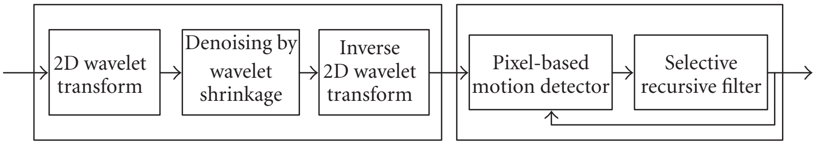

2D wavelet transform
Denoising by wavelet shrinkage
Inverse 2D wavelet transform
Pixel-based motion detector
Selective recursive filter

Figure 3: The implemented denoising scheme.

algorithm upsamples the wavelet filters at each decomposition level. In particular, $2^j - 1$ zeros ("holes," in French, *trous*) are inserted between the filter coefficients at the decomposition level j, as it is shown in Figure 4.

We use the SystemC library [31] and a previously developed simulation environment [32, 33] to develop a real-time model of the wavelet decomposition and reconstruction. Figure 5 shows the simulation model. After a number of simulations and tests we have concluded that the real-time wavelet implementation with 16 bit arithmetic gives practically the same results as a referent MATLAB code of the algorithm *à trous* [30]. At a number of input frames there were more than 97.13% errorless pixels with mean error of 0.0287. Analyzing those figures at the level of bit representation, we can conclude that maximally 1 bit out of 16 was wrong. The wrong bit may occur on the bit position 0 shown in Figure 6. Taking into account that input pixels are 8 bit integers we can ignore this error.

3.2. The pipelined FPGA implementation of the nondecimated wavelet transform

Here we develop an FPGA implementation of a nondecimated wavelet transform with three orientation subbands per scale. We design FIR filters for the algorithm *à trous* [30] with the Daubechies' minimum phase wavelet of length four [34] and we implement the designed FIR filters with dedicated multipliers in the Xilinx Virtex2 FPGAs [35].

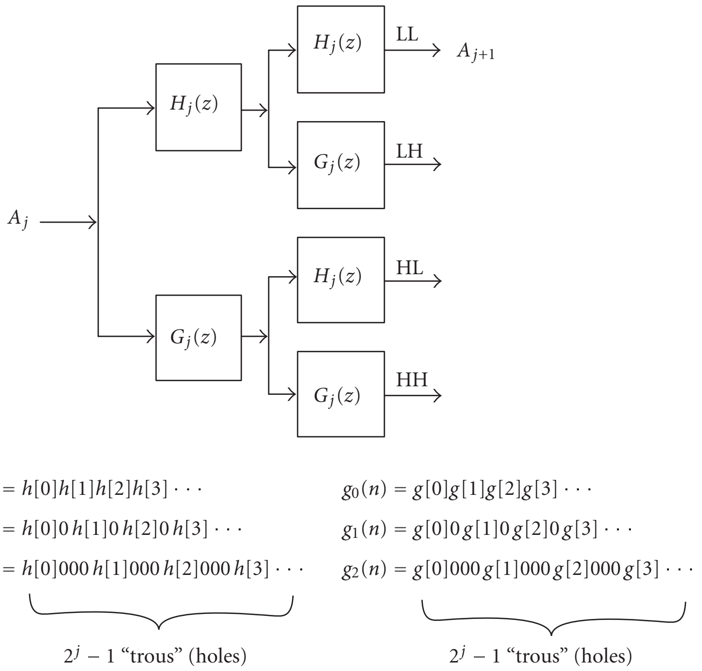

A_j
$H_j(z)$
$G_j(z)$
$H_j(z)$
LL
A_{j+1}
$G_j(z)$
LH
$H_j(z)$
HL
$G_j(z)$
HH
$h_0(n) = h[0]h[1]h[2]h[3]\cdots$
$h_1(n) = h[0]0h[1]0h[2]0h[3]\cdots$
$h_2(n) = h[0]000h[1]000h[2]000h[3]\cdots$
$2^j - 1$ "trous" (holes)
$g_0(n) = g[0]g[1]g[2]g[3]\cdots$
$g_1(n) = g[0]0g[1]0g[2]0g[3]\cdots$
$g_2(n) = g[0]000g[1]000g[2]000g[3]\cdots$
$2^j - 1$ "trous" (holes)

FIGURE 4: The nondecimated 2D discrete wavelet transform.

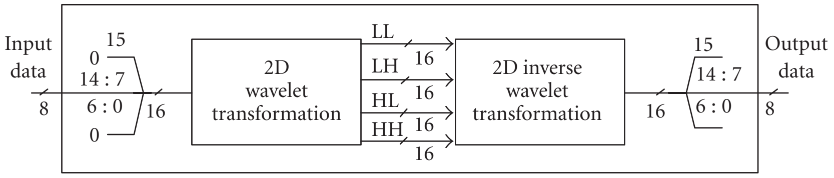


FIGURE 5: The developed simulation model for the implementation of the wavelet transform.

Our implementation of the 2D wavelet transform is line-based as shown in Figure 7. We choose the line alignment in order to preserve the video sequence input format and to *pipeline* the whole processing in our system. The horizontal and the vertical filtering is performed within one pass of the input video stream. We avoid using independent horizontal and vertical processing which requires two cycles and an internal memory for storing the output of the horizontal filtering. Instead, we use the line-based vertical filtering with as many internal line buffers as there are taps in the used FIR filter.

The horizontal and vertical FIR filters differ only in the filter delay path implementation. The data path of the horizontal filter is a register pipeline as shown in Figure 8. The data path of the vertical filter is the output of the line buffers. Hence, the vertical FIR filter does not include any delay elements, but only the pipelined filtering arithmetics (multipliers and an adder). Pipelining the filtering arithmetics ensures the requested timing for data processing and we use this approach both for the horizontal and vertical filters.

The algorithm *à trous* [30] upsamples the wavelet filters by inserting $2^j - 1$ zeros between the filter coefficients at the decomposition level j (see Figure 4). We implement this filter up-sampling by using a longer filter delay path and the appropriate data selection logic. The required number of the registers depends on the length of the mother wavelet function and on the number of the decomposition levels used. We use a wavelet of length four and three decomposition levels, and hence our horizontal filter in Figure 8 contains $3\times4 = 12$ registers. Four registers are dedicated to the 4-tap filter and 3 times as many are needed to implement the required up-sampling up to the third decomposition level. Analogously, on the vertical filtering side, each line buffer for vertical filtering is able to store up to 4 lines.

For the calculation of the first decomposition level of the wavelet transform, only the first 4 registers d0, d1, d2, and d3 in Figure 8 are used in the FIR filter register pipeline. At the second decomposition level, the wavelet filters have to be up-sampled with 1 zero between the filter coefficients. In our implementation, this means that registers d0, d2, d4, and d6 are used for filtering. Figure 8 illustrates the FIR filter configuration during the calculation of the wavelet coefficients from the third decomposition level. During this period, the d0, d4, d8, and d12 registers are involved in the filtering process.

F	E	D	C	B	A	9	8	7	F	5	4	3	2	1	0
0	Input								0	0	0	0	0	0	0
0	Output								X	X	X	X	X	X	X

Figure 6: Input and output data format.

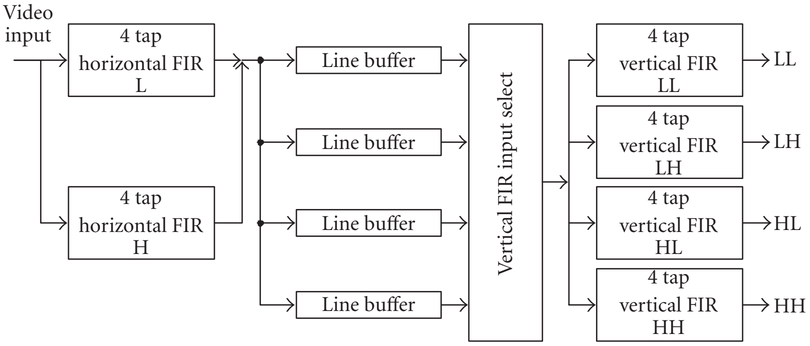


Figure 7: A block schematic of the developed hardware implementation of the wavelet transform.

We implement the inverse wavelet transform accordingly. The processing is mirrored when compared to the wavelet decomposition: the vertical filtering is done first and the horizontal processing afterwards. The FIR filter design is the same as for the direct wavelet transform, only the filter coefficients a(0), a(1), a(2), and a(3) in Figure 8 are mirrored.

3.3. The wavelet shrinkage customization

Our video denoising scheme employs a spatially adaptive wavelet shrinkage approach of [36]. A brief description of this denoising method follows.

Let y_l denote the noise-free wavelet coefficient and w_l its observed noisy version at the spatial position l in a given wavelet subband. For compactness, we suppressed here the indices that denote the scale and the orientation. The method of [36] shrinks each wavelet coefficient by a factor which equals the probability that this coefficient presents a signal of interest. The signal of interest is defined as a noise-free signal component that exceeds in magnitude the standard deviation of noise σ. The probability of the presence of a signal of interest at position l is estimated based on the coefficient magnitude $|w_l|$ and based on a local spatial activity indicator $z_l = \sum_{k\in\partial_l}|w_k|$, where ∂_l is the neighborhood of the pixel l (within a squared window) and N_l is the number of the neighboring coefficients. For example, for a 3×3 window ∂_l consists of the 8 nearest neighbors of the pixel l ($N_l = 8$).

Let H_1 denote the hypothesis "*the signal of interest is present*:" $|y_l| > \sigma$ and let H_0 denote the opposite hypothesis: "$|y_l| \le \sigma$." The shrinkage estimator of [9] is

$$\hat{y}_l = P(H_1 \mid w_l, z_l)w_l = \frac{\rho\xi_l\eta_l}{1+\rho\xi_l\eta_l}w_l, \tag{1}$$

where

$$\rho = \frac{P(H_1)}{P(H_0)}, \qquad \xi_l = \frac{p(w_l \mid H_1)}{p(w_l \mid H_0)}, \qquad \eta_l = \frac{p(z_l \mid H_1)}{p(z_l \mid H_0)}. \tag{2}$$

$p(w_l \mid H_0)$ and $p(w_l \mid H_1)$ denote the conditional probability density functions of the noisy coefficients given the absence and given the presence of a signal of interest. Similarly, $p(z_l \mid H_0)$ and $p(z_l \mid H_1)$ denote the corresponding conditional probability density functions of the local spatial activity indicator. The input-output characteristic of this wavelet denoiser is illustrated in Figure 9. This figure shows that the coefficients that are small in magnitude are strongly shrinked towards zero, while the largest ones tend to be left unchanged. The displayed family of the shrinkage characteristics corresponds to the different values of the local spatial activity indicator. For the same coefficient magnitude $|w_l|$ the input coefficient will be shrunk less if LSAI z_l is bigger and vice versa.

We now address the implementation of this shrinkage function. Under the Laplacian prior for noise-free data $p(y) = (\lambda/2)\exp(-\lambda|y|)$ we have [9] $\rho = \exp(-\lambda T)/(1 - \exp(-\lambda T))$. The analytical expressions for ξ_l and η_l seem too complex for the FPGA implementation. We efficiently implement the two likelihood ratios ξ_l and η_l as appropriate *look-up tables*, stored in two "read-only" memories (ROM). The generation of the particular look-up-tables is based on an extensive experimental study, as we explain later in this section. The developed architecture is presented in Figure 10. One ROM memory, containing the look-up table ξ_l, is addressed by the coefficient magnitude $|w_l|$, and the other ROM memory, containing the look-up table $\rho\eta_l$ is addressed by LSAI z_l. For calculating LSAI, we average the coefficient values from the current line and from the previous two lines within

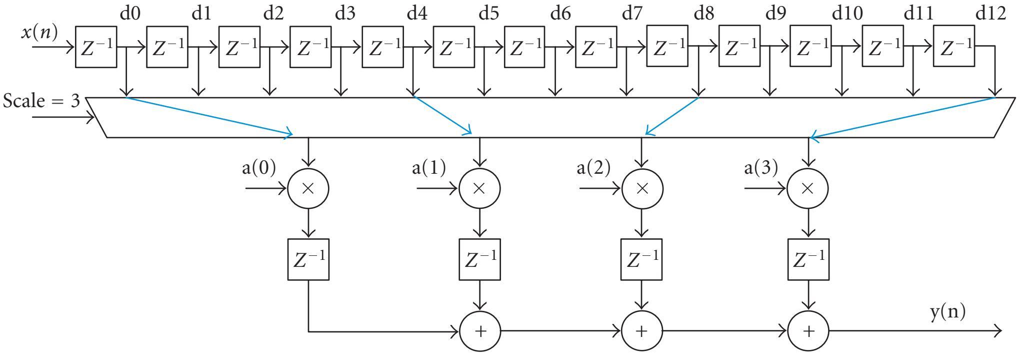


Figure 8: The proposed FIR filter implementation of the algorithm *à trous* for a mother wavelet of length 4 and supporting up to 3 decomposition levels. The particular arithmetic network using the registers d0, d4, d8, and d12 corresponds to the calculation of the wavelet coefficients at the third decomposition level.

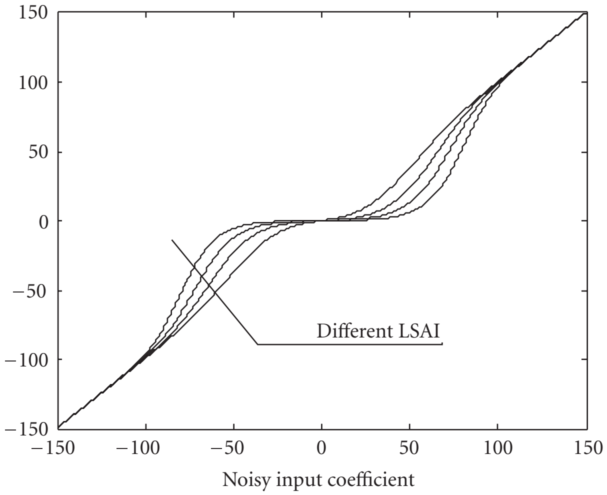


Figure 9: An illustration of the employed wavelet shrinkage family.

a 3 × 3 window. The read values from ROM's are multiplied to produce the generalized likelihood ratio $r = \rho\xi_l\eta_l$. We found it more efficient to realize the shrinkage factor $r/(1+r)$ using another ROM (look-up-table) instead of using the arithmetic operations. The output of this look-up-table denoted here as "shrinkage ROM" is the desired wavelet shrinkage factor. Finally, the output of the shrinkage ROM multiplies the input coefficient to yield the denoised coefficient.

We denoise in parallel three wavelet bands LH, HL, and HH at each scale. Different resolution levels (we use three) are processed sequentially as illustrated in Figure 2. The lowpass (LL) band is only delayed for the number of clock periods that are needed for denoising. This delay, which is in our implementation 6 clock cycles, ensures the synchronization of the inputs at the inverse wavelet transform block (see the timing in Figure 2).

The generation of the appropriate look-up tables for the two likelihood ratios resulted from our extensive experiments on different test images and different noise-levels as it is described in [29]. Figure 11 illustrates the likelihood ratio ξ_l calculated from one test image at different noise levels. These diagrams show another interpretation of the well-known threshold selection principle in wavelet denoising: a well-chosen threshold value for the wavelet coefficients increases with the increase of the noise level. The maximum likelihood estimate of the threshold T (i.e., the value for which $p(T \mid H_0) = p(T \mid H_1)$) is the abscissa of the point $\xi_l = 1$. Figure 12 displays the likelihood ratio ξ_l, in the diagonal subband HH at third decomposition level, for 10 different frames with fixed noise standard deviations ($\sigma = 10$ and $\sigma = 30$). We showed in [29] that from a practical point of view, the difference between the calculated likelihood ratios for different frames is minor, especially for lower noise levels (up to $\sigma = 20$). Therefore we average the likelihood ratios over different frames and store these values as the corresponding look-up tables for several different noise levels ($\sigma =$ 5, 10, 15, and 20). In the denoising procedure, the user selects the input noise level, which enables addressing the correct set of the look-up tables. The performance loss of the algorithm due to simplifications with the generated look-up tables is for different input noise levels shown in Figure 13. These results

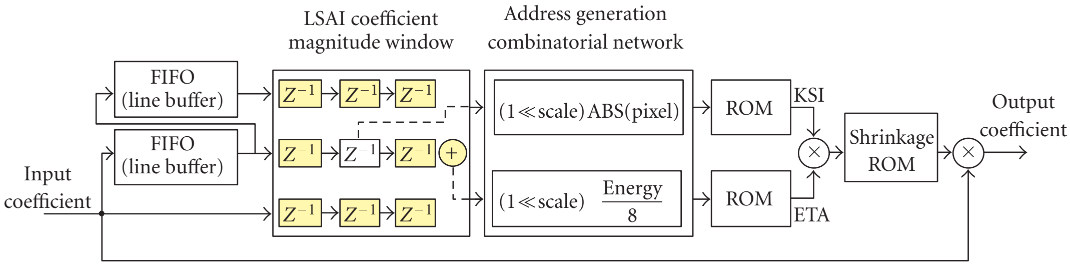


Figure 10: Block schematic of implemented denoising architecture.

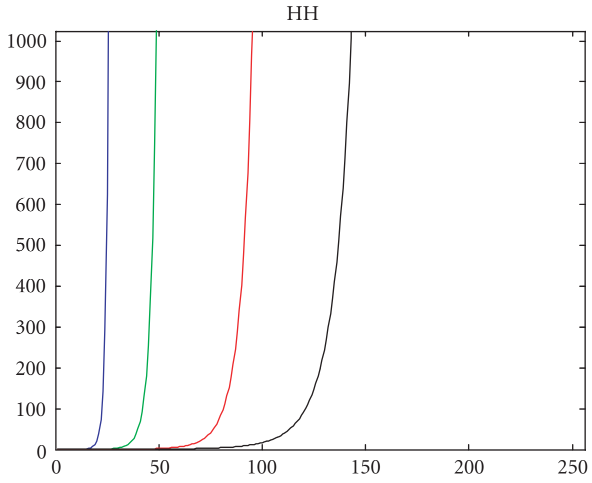


Figure 11: Likelihood ratio ξ_l for one test frame and 4 different noise levels ($\sigma = 5, 10, 20, 30$).

represent peak signal-to-noise ratio (PSNR) values averaged over frames of several different video sequences. For $\sigma = 10$ the average performance loss was only 0.13 dB (and visually, the differences are difficult to notice) while for $\sigma = 20$ the performance loss is 0.55 dB and is on most frames becoming visually noticeable, but not highly disturbing. For higher noise levels, the performance loss increases.

In the current implementation, the user has to select one of the available noise levels. With such approach, it is possible that the user will not choose the best possible noise reduction. If the selected noise level is smaller from the real noise level in the input signal, some of the noise will remain in the output signal. On the other hand, if the noise level is over-estimated, the output signal will be blurred without satisfying visual effect.

This user intervention can be avoided by implementing a noise level estimator. The output of this block could be used for the look-up table selection, which further enables adjustable noise reduction according to the noise level in input signal. For example, a robust wavelet-domain noise estimator based on the median absolute deviation [37] can be used for this purpose or other related wavelet-domain noise estimators like [38].

The likelihood ratios ξ_l and η_l are monotonic increasing functions. We are currently investigating the approximation of these functions by a family of piece-wise linear functions parameterized by the noise standard deviation and by the parameter of the marginal statistical distribution of the noise-free coefficients in a given subband.

3.4. Temporal filtering

A pixel-based motion detector with selective recursive temporal filtering is quite simple for hardware implementation. Since we first apply a high quality spatial filtering the noise is already significantly suppressed and thus a pixel-based motion detection is efficient. In case the motion is detected the recursive filtering is switched off.

Two pixels are involved for temporal filtering at a time: one pixel from the current field and another from the same spatial position in the previous field. We store the two fields in the output buffer and read the both required pixel values in the same cycle. If the absolute difference between these two pixel values is smaller than the predefined threshold value, *no motion* case is assumed and the two pixel values are subject to a weighted averaging, with the weighting factors defined in [9]. In the other case, when motion is detected, the current pixel is passed to the output. The block schematic in Figure 14 depicts the developed FPGA architecture of the selective recursive temporal filter described above. We use the 8 bit arithmetic because the filter is located in the time domain where all the pixels are represented as 8 bit integers.

4. REAL-TIME ENVIRONMENT

In our implementation we use the standard television broadcasting signal as a source of video signal. A common feature of all standard TV broadcasting technologies is that the video sequence is transmitted in analog domain (this excludes the latest DVB and HDTV transmission standards). Thus, before digital processing of television video sequence the digitalization is needed. Also, after digital processing the sequence has to be converted back to the analogue domain in order to be shown on a standard tube display. This pair of A/D and D/A converters is well known as a codec. The 8 bit codec, with 256 levels of quantization per pixel, is considered sufficient from the visual quality point of view. Figure 15 shows a block schematic of digital processing for television broadcasting systems.

We use the PAL-B broadcasting standard and 8 bit YUV 4 : 2 : 2 codec. The hardware platform set-up consists of

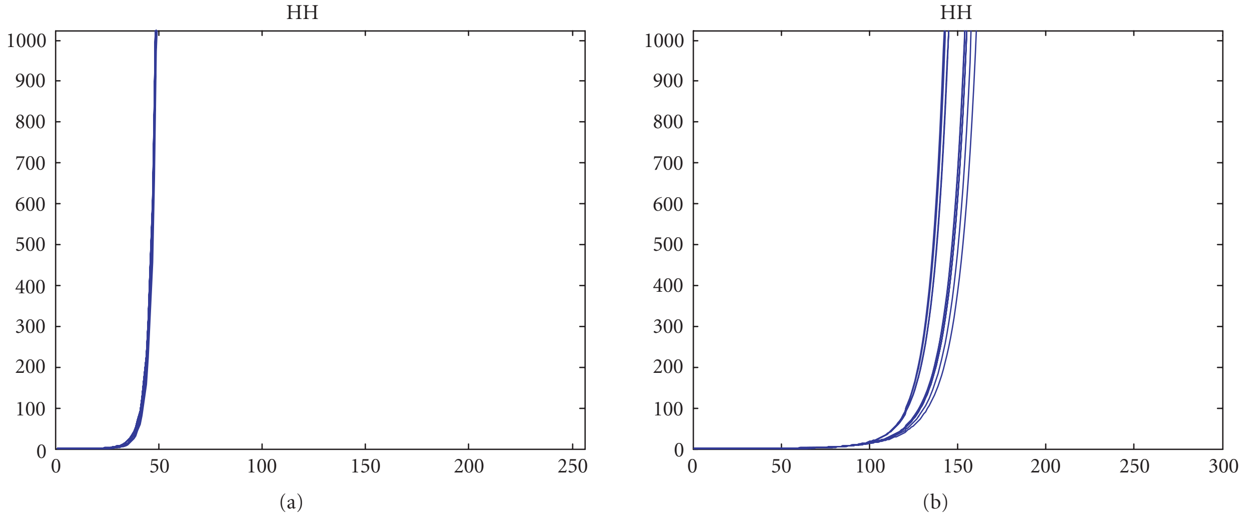


Figure 12: Likelihood ratio ξ_l displayed for 10 frames with fixed-noise levels: $\sigma = 10$ (a) and $\sigma = 30$ (b).

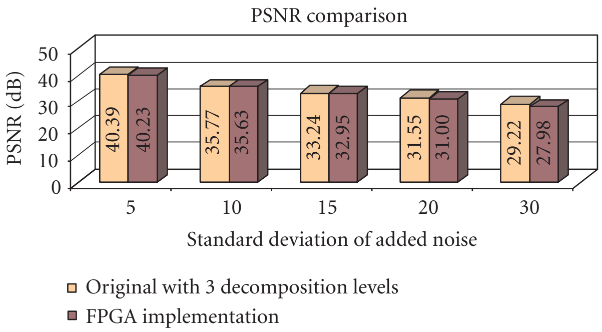


Figure 13: Performance of the designed FPGA implementation in comparison with the original software version of the algorithm, which employs exact analytical calculation of the involved shrinkage expression.

three separate boards. Each board corresponds to one of the blocks presented in Figure 15:

(i) Micronas IMAS-VPC 1.1 (A/D—analog front-end) [39];
(ii) CHIPit Professional Gold Edition (processing block) [40];
(iii) Micronas IMAS-DDPB 1.0 (D/A—analog back-end) [41].

We made all the connections among the previously mentioned boards with a separate *interconnection* board designed for this purpose. This interconnection board consists of the interconnection channels and the voltage adjustments between the CHIPit board (3.3 V level) and the Micronas IMAS boards (5 V level).

The *processing* board consists of two Xilinx Virtex II FPGAs (XC2V6000-5) [35] and is equipped with plenty of SDRAM memory (6 banks with 32 bit access made with 256 Mbit ICs).

All boards of the used hardware platform are configured with the I2C interface. The user is able to set up the needed noise level in input signal. This is fulfilled with writing appropriate value to the corresponding register in the FPGA accessible via the I2C interface. Appropriate look-up table with the averaged likelihood ratio is selected according to the value in this register.

5. CONCLUSION

We designed a real-time FPGA implementation of an advanced wavelet-domain video denoising algorithm. The developed hardware architecture is based on innovative technical solutions that allow an implementation of sophisticated adaptive wavelet denoising in hardware. We believe that the results reported in this paper can be interesting for a number of industrial applications, including TV broadcasting systems. Our current implementation has limitations in practical use due to the required user-intervention for noise level estimation. Our future work will integrate the noise level estimation to avoid these limitations and to allow automatic adaptation of the denoiser to the noise level changes in the input signal.

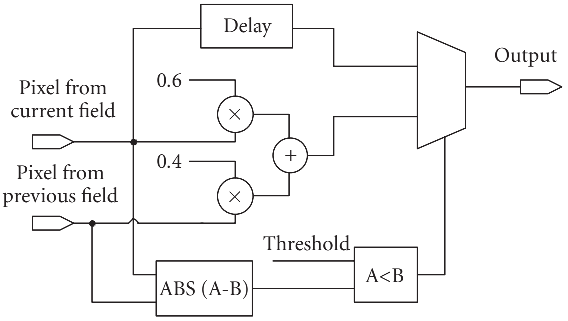


Figure 14: Block schematic of implemented temporal filter.

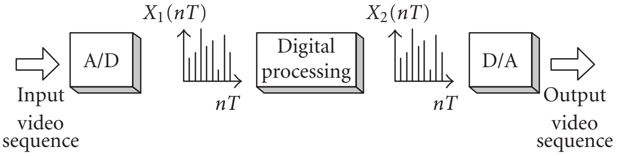


Figure 15: A digital processing system for television broadcasting video sequences.

ACKNOWLEDGMENT

The second author is a Postdoctoral Researcher of the Fund for the Scientific Research in Flanders (FWO), Belgium.

REFERENCES

[1] F. Cocchia, S. Carrato, and G. Ramponi, "Design and real-time implementation of a 3-D rational filter for edge preserving smoothing," *IEEE Transactions on Consumer Electronics*, vol. 43, no. 4, pp. 1291–1300, 1997.

[2] G. Arce, "Multistage order statistic filters for image sequence processing," *IEEE Transactions on Signal Processing*, vol. 39, no. 5, pp. 1146–1163, 1991.

[3] V. Zlokolica and W. Philips, "Motion and detail adaptive denoising of video," in *Image Processing: Algorithms and Systems III*, vol. 5298 of *Proceedings of SPIE*, pp. 403–412, San Jose, Calif, USA, January 2004.

[4] L. Hong and D. Brzakovic, "Bayesian restoration of image sequences using 3-D Markov random fields," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '89)*, vol. 3, pp. 1413–1416, Glasgow, UK, May 1989.

[5] J. Brailean and A. Katsaggelos, "Simultaneous recursive displacement estimation and restoration of noisy-blurred image sequences," *IEEE Transactions on Image Processing*, vol. 4, no. 9, pp. 1236–1251, 1995.

[6] P. van Roosmalen, S. Westen, R. Lagendijk, and J. Biemond, "Noise reduction for image sequences using an oriented pyramid thresholding technique," in *IEEE International Conference on Image Processing*, vol. 1, pp. 375–378, Lausanne, Switzerland, September 1996.

[7] I. Selesnick and K. Li, "Video denoising using 2D and 3D dual-tree complex wavelet transforms," in *Wavelets: Applications in Signal and Image Processing X*, vol. 5207 of *Proceedings of SPIE*, pp. 607–618, San Diego, Calif, USA, August 2003.

[8] D. Rusanovskyy and K. Egiazarian, "Video denoising algorithm in sliding 3d dct domain," in *Proceedings of the 7th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS '05)*, J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, Eds., vol. 3708 of *Lecture Notes on Computer Science*, pp. 618–625, Antwerp, Belgium, September 2005.

[9] A. Pižurica, V. Zlokolica, and W. Philips, "Noise reduction in video sequences using wavelet-domain and temporal filtering," in *Wavelet Applications in Industrial Processing*, vol. 5266 of *Proceedings of SPIE*, pp. 48–59, Providence, RI, USA, October 2003.

[10] V. Zlokolica, A. Pižurica, and W. Philips, "Video denoising using multiple class averaging with multiresolution," in *The International Workshop on Very Low Bitrate Video Coding (VLBV '03)*, pp. 172–179, Madrid, Spain, September 2003.

[11] B. A. Draper, J. R. Beveridge, A. P. W. Bohm, C. Ross, and M. Chawathe, "Accelerated image processing on FPGAs," *IEEE Transactions on Image Processing*, vol. 12, no. 12, pp. 1543–1551, 2003.

[12] A. M. Al-Haj, "Fast discrete wavelet transformation using FPGAs and distributed arithmetic," *International Journal of Applied Science and Engineering*, vol. 1, no. 2, pp. 160–171, 2003.

[13] G. Goslin, "A guide to using field programmable gate arrays (FPGAs) for application-specific digital signal processing performance," XILINX Inc., 1995.

[14] C. Dick, "Implementing area optimized narrow-band FIR filters using Xilinx FPGAs," in *Configurable Computing: Technology and Applications*, vol. 3526 of *Proceedings of SPIE*, pp. 227–238, Boston, Mass, USA, November 1998.

[15] R. D. Turney, C. Dick, and A. Reza, "Multirate filters and wavelets: from theory to implementation," XILINX Inc.

[16] J. Ritter and P. Molitor, "A pipelined architecture for partitioned DWT based lossy image compression using FPGA's," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '01)*, pp. 201–206, Monterey, Calif, USA, February 2001.

[17] M. Nibouche, A. Bouridane, F. Murtagh, and O. Nibouche, *FPGA-Based Discrete Wavelet Transforms System*, School of Computer Science, The Queen's University of Belfast, Belfast, UK, 2001.

[18] M. A. Trenas, J. Lopez, and E. L. Zapata, "FPGA implementation of wavelet packet transform with reconfigurable tree structure," in *Proceedings of the 26th Euromicro Conference (EUROMICRO '00)*, pp. 1244–1251, Maastricht, The Netherlands, September 2000.

[19] K. Wiatr and P. Russek, "Embedded zero wavelet coefficient coding method for FPGA implementation of video codec in real-time systems," in *The International Conference on Information Technology: Coding and Computing (ITCC '00)*, pp. 146–151, Las Vegas, Nev, USA, March 2000.

[20] S. G. Mathen, "Wavelet transform based adaptive image compression on FPGA," M.S. thesis, University of Kansas, Manhattan, Kan, USA, 2000.

[21] J. B. Ballagh, "An FPGA-based run-time reconfigurable 2-D discrete wavelet transform core," M.S. thesis, Virginia Polytechnic Institute, Blacksburg, Va, USA, 2001.

[22] L. Nachtergaele, B. Vanhoof, M. Peón, G. Lafruit, J. Bormans, and I. Bolsens, "Implementation of a scalable MPEG-4 wavelet-based visual texture compression system," in *Proceedings of the 36th Design Automation Conference (DAC '99)*, pp. 333–336, New Orleans, La, USA, June 1999.

[23] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, 1998.

[24] W. Sweldens, "Lifting scheme: a new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III*, vol. 2569 of *Proceedings of SPIE*, pp. 68–79, San Diego, Calif, USA, July 1995.

[25] W. Sweldens, "Wavelets and the lifting scheme: a 5 minute tour," *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 76, no. 2, pp. 41–44, 1996.

[26] G. Dillen, B. Georis, J.-D. Legat, and O. Cantineau, "Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 944–950, 2003.

[27] B.-F. Wu and Y.-Q. Hu, "An efficient VLSI implementation of the discrete wavelet transform using embedded instruction codes for symmetric filters," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 936–943, 2003.

[28] M. Katona, A. Pižurica, V. Zlokolica, N. Teslić, and W. Philips, "Real-time wavelet domain video denoising implemented in FPGA," in *Wavelet Applications in Industrial Processing II*, vol. 5607 of *Proceedings of SPIE*, pp. 63–70, Philadelphia, Pa, USA, October 2004.

[29] M. Katona, A. Pižurica, N. Teslić, V. Kovacevic, and W. Philips, "FPGA design and implementation of a wavelet-domain video denoising system," in *Proceedings of the 7th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS '05)*, J. Blanc-Talon, D. Popescu, W. Philips, and P. Scheunders, Eds., vol. 3708 of *Lecture Notes on Computer Science*, pp. 650–657, Antwerp, Belgium, September 2005.

[30] S. Mallat and S. Zhong, "Characterization of signals from multiscale edges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 7, pp. 710–732, 1992.

[31] *SystemC Version 2.0 Users Guide*, SystemC Inc., 2002, http://www.systemc.org.

[32] M. Katona, N. Teslić, V. Kovacevic, and M. Temerinac, "Test environment for bluetooth baseband inegrated circuit development," in *Proceedings of the 5th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS '01)*, B. D. Milovanovic, Ed., vol. 2, pp. 405–408, Nis, Yoguslavia, Septmeber 2001.

[33] M. Katona, N. Teslić, and Z. Krajacevic, "FPGA design with SystemC," in *The 10th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES '03)*, A. Napieralski, Ed., vol. 1, pp. 220–223, Lodz, Poland, June 2003.

[34] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, Pa, USA, 1992.

[35] *Virtex II Platform FPGA: Complete Data Sheet*, XILINX Inc., 2004, http://www.xilinx.com.

[36] A. Pižurica and W. Philips, "Estimating the probability of the presence of a signal of interest in multiresolution single- and multiband image denoising," *IEEE Transactions on Image Processing*, vol. 15, no. 3, pp. 654–665, 2006.

[37] D. L. Donoho and I. M. Johnstone, "Adapting to unknown smoothness via wavelet shrinkage," *Journal of the American Statistical Association*, vol. 90, no. 432, pp. 1200–1224, 1995.

[38] V. Zlokolica, A. Pižurica, and W. Philips, "Noise estimation for video processing based on spatial-temporal gradient histograms," to appear in *IEEE Signal Processing Letters*.

[39] *VPC 3205C Video Processor Family*, ITT Semiconductors: ITT Intermetall, 1997, http://www.micronas.com.

[40] *CHIPit Gold Edition Handbook*, ProDesign Electronic & CAD Layout, 2003, http://www.prodesigncad.de.

[41] *DDPB 3310B Display and Deflection Processor*, Micronas Intermetal, 1998, http://www.micronas.com.

Mihajlo Katona was born in 1974, in Vrbas, Yugoslavia. In 1999, he received the Diploma degree in computer engineering and in 2001, M. S. degree in computer science both from the University of Novi Sad (Serbia and Montenegro). In 1999, he joined the Chair for Computer Engineering at the University of Novi Sad, where he is currently working as a Teaching Assistant in the "design of complex digital systems." He is currently pursuing his Ph.D. thesis. His research interests include digital signal processing, DSP algorithm customization for hardware implementation, system-on-chip architectures, and FPGA prototyping.

Aleksandra Pižurica was born in Novi Sad, Yugoslavia, on September 18, 1969. In 1994, she received the Diploma degree in electrical engineering from the University of Novi Sad, Yugoslavia, in 1997 the M.S. degree in telecommunications from the University of Belgrade, Yugoslavia, and in 2002 the Ph.D. degree from the Ghent University, Belgium. Since 1994 until 1997, she was working at the Department of Telecommunications of the University of Novi Sad, and in 1997 she joined the Department of Telecommunications and Information Processing of the Ghent University. She is the author of 15 papers in international journals and more than 50 papers at international scientific conferences. Her research interests include image restoration, multiresolution representations, Markov random field models, signal detection and estimation, multimedia applications, and remote sensing.

Nikola Teslić is a Professor at the Chair for Computer Engineering, Faculty of Engineering, University of Novi Sad, Serbia and Montenegro. In 1995, he received the Diploma degree in electrical engineering from the University of Novi Sad, Yugoslavia, in 1997 the M.S. degree in computer engineering, and in 1999 the Ph.D. degree from the University of Novi Sad, Serbia and Montenegro. Currently he lectures in the "design of complex digital systems" and "software for TV sets and image processing" and "DSP architectures and algorithms." His scientific interests are in the area of computer engineering, especially in the area of real-time systems, electronic computer-based systems, digital system for audio-video processing. He is the author of 6 papers in international journals and more than 50 papers at international scientific conferences.

Vladimir Kovačević is a Professor and he leads the Chair for Computer Engineering, Faculty of Engineering, University of Novi Sad, Yugoslavia. Currently he lectures in the "design of complex digital systems" and "computer systems design." He received his Ph.D. degree at the University of Belgrade. His scientific interests are in the areas of computer engineering, especially in the area of real-time systems, electronic computer-based systems, large-scale digital system design, computer systems design, communication networks, and systems programming.

Wilfried Philips was born in Aalst, Belgium on October 19, 1966. In 1989, he received the Diploma degree in electrical engineering and in 1993 the Ph.D. degree in applied sciences, both from the University of Ghent, Belgium. Since November 1997, he is a Lecturer at the Department of Telecommunications and Information Processing of the University of Ghent. His main research interests are image and video restoration and analysis and data compression. He is the author of more than 50 papers in international journals and 200 papers in the proceedings of international scientific conferences, the Editor of 8 conference proceedings and 1 special issue of a journal. He has received 10 national and internal awards for his research. He coorganizes 2 international conferences in the area of image and video processing and computer vision and is a member of the program committee of several national and international workshops.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 82564, Pages 1–8
DOI 10.1155/ES/2006/82564

A Dynamic Reconfigurable Hardware/Software Architecture for Object Tracking in Video Streams

Felix Mühlbauer and Christophe Bobda

Department of Computer Sciences, University of Kaiserslautern, Gottlieb-Daimler Street 48, 67653 Kaiserslautern, Germany

Received 15 December 2005; Revised 8 June 2006; Accepted 11 June 2006

This paper presents the design and implementation of a feature tracker on an embedded reconfigurable hardware system. Contrary to other works, the focus here is on the efficient hardware/software partitioning of the feature tracker algorithm, a viable data flow management, as well as an efficient use of memory and processor features. The implementation is done on a Xilinx Spartan 3 evaluation board and the results provided show the superiority of our implementation compared to the other works.

1. INTRODUCTION/MOTIVATION

Pervasive and ubiquitous computing is gaining more and more in popularity. Boosted by advances in broadband communication and processing systems, *computer anytime and anywhere* is slowly but surely becoming a reality. Ubiquitous and pervasive computing usually involve a set of distributed sensing and computing nodes geographically located at different sites. Each node collects a given amount of raw data that is exchanged with other nodes in the system. One of the main requirements here is that raw data collected by sensors at a given geographic location by a given system or part of it should be processed by a corresponding module at that location. Therefore, the communication between different nodes is reduced. Only the results of computations at different sites—which are mostly sensor data interpretation with a reduced amount compare to the raw data—have to be sent to other nodes.

The constraints imposed on pervasive and ubiquitous computing systems—which are mostly untethered—lead to a very challenging design process. Large amount of data must be computed in real time whilst at the same time maintaining a very low power consumption for the whole system. Furthermore, the system must be able to adapt to changing environmental and operational conditions. None of the processors commonly used in embedded systems like DSPs, ASICs or general purpose processors can provide the features alone (performance, low power, and adaptivity) that are required in ubiquitous and pervasive systems.

The last decade has experienced an increasing interest in deployment of FPGAs in embedded systems. With the progress in manufacturing technology, FPGAs have become 40 times faster and consume 50 times less power with an increase of 200 fold in their capacity (number of available logic cells) whilst at the same time becoming 500 times cheaper in less than 15 years. According to several studies, this trend is going to be maintained at least in medium term. It is increasingly possible to implement a complete system on-chip solution using the lowest cost FPGA device. Furthermore, the partial reconfiguration capability of FPGAs allows for the realization of adaptivity, thus making FPGAs more and more attractive for pervasive and ubiquitous systems [1].

A main advantage of these programmable logic devices is the ability to realize parallel processing hardware. Especially image processing algorithms are inherently parallel and thus FPGAs can be used to develope highly efficient solutions. In many systems, for example, in surveillance systems, video data is captured by modules and sent to other modules for further processing. The processing task can be, for example, the detection of movement or the detection and tracking of suspect objects in a given environment that is monitored by a camera.

A system on chip is usually made up of a processor connected to a set of peripherals and dedicated hardware modules via a bus system. The bus system is mastered by the processor to access peripherals and collect data to be processed. In video streaming applications in which large amount of data must be computed in real time while streaming through different computational blocks, a traditional system on chip in which all the data transfer between different modules is done on the bus is no longer viable. The exclusive use of the bus at a given time by one master hinders simultaneous access to data by different modules.

In this work, we present a modular implementation of a feature tracker for video streams in an FPGA. The architecture is made up of a system on chip in which a processor and dedicated hardware accelerator modules cohabit. Contrary to traditional system on chip, we do not rely only on a bus for communication. A set of dedicated line and protocols allow for a real-time computation of data while they are streamed.

The implementation is done on a Xilinx evaluation board featuring a Spartan 3 FPGA and on a ML310 board with a Virtex2 Pro.

The remainder of this paper is organized as follows. Section 2 introduces the feature tracking and presents algorithms used. In Section 3, we present an overview of the related work. Section 4 presents the design of the feature tracking system on an FPGA. There we will discuss the main design decision. The adaptivity of the system is also discussed in this section. In Section 5, we present the implementation results for two platforms. Finally, Section 6 concludes the paper and gives some indication of future work.

2. OBJECT TRACKING IN VIDEO STREAMS

For object tracking purposes often feature trackers are used, which analyze image sequences and detect motion. For this purpose small windows, called features, with certain attributes are selected and then attempts are made to find them in the next frame. Such attributes can, for example, be some measure of texturedness or cornerness, like a high standard deviation in the spatial intensity profile, the presence of zero crossings of the Laplacian of the image, or a simple corner. Yet, apparently promising features can be useless or even harmful for tracking, if they do not correspond to a point in the real world. This happens with hotspots (a reflection of a highlight on a glossy surface), mirroring, or in the case of straddling a depth discontinuity. Conversely, useful features can be lost if they leave the field of vision by obstruction or by moving out of the image. The well-known KLT-tracker[1] is often used as a base for further development.

The same case is with the following algorithm which is the approach of Shi and Tomasi [4]. According to them, the pure translation model is not an adequate model for image motion when measuring dissimilarity between the features. They provide experimental evidence for this and introduce an extended model considering affine image changes.

Image motion can be regarded as a change in image intensity I of a given point (x, y) at time $t + \tau$:

$$I(x, y, t + \tau) = I(x - \xi(x, y, t, \tau), y - \eta(x, y, t, \tau), t). \quad (1)$$

The time-dependent functions ξ and η provide the displacement in x and y directions. So, $\delta = (\xi, \eta)$ defines the amount of motion and, respectively, the displacement of the point at $\mathbf{x} = (x, y)$. Even within the small windows used for feature tracking, δ varies and a certain displacement vector does not exist. A more efficient way is to consider the affine motion model:

$$\delta = D\mathbf{x} + \mathbf{d}, \quad (2)$$

where

$$D = \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix} \quad (3)$$

is a deformation matrix and $\mathbf{d}$ is the translation of the feature window's center. Applying this to the intensity relation leads to

$$J(A\mathbf{x} + \mathbf{d}) = I(\mathbf{x}), \quad \text{where } A = \mathbf{Id} + D. \quad (4)$$

This means that for any two given images I and J six parameters must be calculated. The quality of the results depends on the size of the feature window, the texture of the image within it, and the amount of motion (camera or object) between frames. Smaller features result in less reliable values for D, because only few image changes are considered, but are generally preferable because they are less likely to straddle a depth discontinuity.

Because of image noise and because the affine motion model is not perfect, (4) is in general not satisfyingly exact enough. To solve this problem the following equation is used to reduce the minimal error to a sensible value for A and $\mathbf{d}$:

$$\epsilon = \iint_W [J(A\mathbf{x} + \mathbf{d}) - I(\mathbf{x})]^2 \, w(\mathbf{x}) d\mathbf{x}, \quad (5)$$

where W is the feature window and $w(\mathbf{x})$ a weighting function, which comes to 1 in the simplest case. Alternatively, a Gaussian-like function can be used to emphasize the center area of the window.

The problem can be converted to a linear 6 × 6 system and D and $\mathbf{d}$ can be found with an iterative Newton-Raphson style minimization (see [5]).

Shi and Tomasi use the pure translation model for tracking and affine motion for comparing features between the first and the current frames in order to monitor quality.

This algorithm has its advantages and drawbacks. First, it works at subpixel precision. Feature windows in frames will never be identical because of image noise, intensity changes, and other interfering factors. Thus translation estimation cannot be absolutely accurate, the errors accumulate and feature windows drift from their actual positions. In [6] Zinßer et al. take care of this problem and also deal with illumination compensation. Another advantage of their concept is the detection of distorted and rotated features, which is achieved by the affine motion model.

[1] Kanade, Lucas, and Tomasi [2, 3].

The algorithm excessively uses floating point operations causing high resource costs. Also, only small translations can be estimated which requires slow moving objects in the observed scene or high frame rates of the incoming video stream, which results in high resource consumption, too. We want to introduce another procedure for tracking features which is much more suitable for an implementation on FPGAs.

The following algorithm refers to [7]. In contrast to the KLT-tracker, features (in this case: Harris corners[2]) are detected in *each* frame and only comparisons between *features* are permitted. For that purpose each position in the current frame, or to be more precise a feature window with this position (feature point) in the middle, is assessed: the derivatives I_x and I_y are computed by horizontal and vertical filters in the form $[-1 \ \ 0 \ \ 1]$. Next, the products I_xI_x, I_xI_y and I_yI_y are separately convolved with the binomial filter $[1 \ \ 4 \ \ 6 \ \ 4 \ \ 1]$, again horizontally and vertically, to produce the values G_{xx}, G_{xy}, and G_{yy}. Now determinant $d = G_{xx}G_{yy} - {G_{xy}}^2$, trace $t = G_{xx} + G_{yy}$, and finally the strength $s = d - kt^2$ with $k = 0.06$ of the corner response are calculated (see Figure 1(a), white areas represent high values of s).

To define the actual feature points, nonmax suppression is used: each pixel for which the corner response is strongest, considering a 5×5 neighborhood, is declared a feature point. This method is an alternative to using a global threshold for the strength of the corner response (see Figure 1(b), where features are marked).

For matching, each feature is compared with all features of the next frame which reside within a certain distance of the original window. To achieve this, normalized correlation is used. The distance can be adjusted to performance requirements. Because Harris corners happen to be in the corner of their feature window, which impedes correct matching, a bigger window of 11×11 pixels ($n = 121$) is used instead. Many comparisons have to be made but fortunately some values can be precomputed:

$$\begin{aligned} A &= \sum I, \\ B &= \sum I^2, \\ C &= \frac{1}{\sqrt{nB - A^2}}. \end{aligned} \tag{6}$$

With the scalar product

$$D = \sum I_1 I_2 \tag{7}$$

of the two features to be compared, the normalized correlation is

$$\epsilon = (nD - A_1A_2)C_1C_2. \tag{8}$$

To decide which matches to accept, a mutual consistency check is used: all features are compared under several different aspects. For each feature, the preferred counter part, which produces the highest value of ϵ, is saved. Finally only features which mutually fit each other are valid matches.

This algorithm is not equipped with a drift detection. In addition, because of the irregular input data, as mentioned above, the features, which are detected for every frame, will vary and matching is partly impeded. On the other hand, translations over larger distances can be estimated while only low frame rates are needed and calculations are simple, compared to the first algorithm which excessively uses floating point operations. Further, the complete feature detection and selection process is highly parallelizable and additionally can be computed using integer operations only. These are very good preconditions for hardware/software co-design implementation on an FPGA.

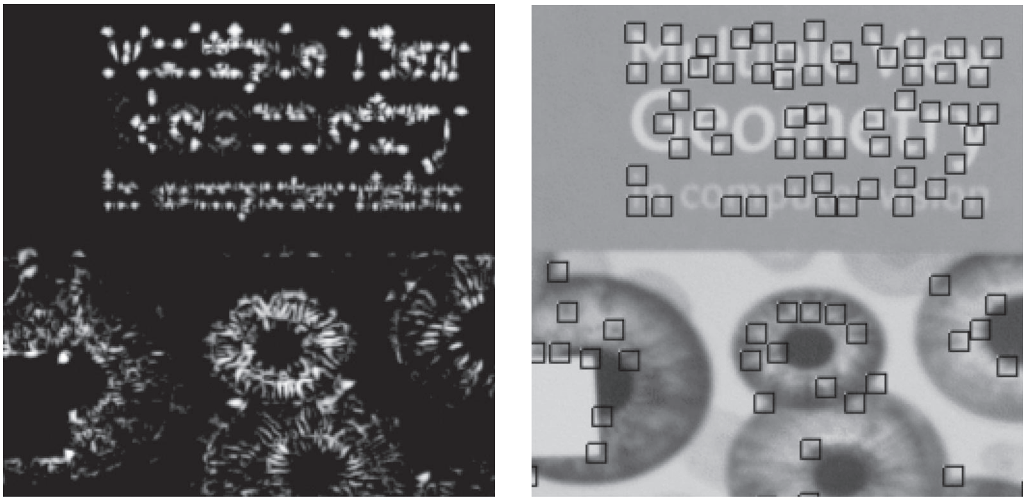
3. RELATED WORK

Feature tracking is usually implemented in the context of autonomous navigation where objects have been detected and tracked by a given entity. Most of the available systems are implemented as a pure software solution [4–7]. Usually a personal computer is mounted on a robot to perform the required computation. Acceleration of feature tracking on parallel computers is considered in [8–10]. The MIMD is considered in [9] while [10] implements the SIMD paradigm. The target platform for the MIMD implementation is a MasPar MP-1 with a 128×128 mesh of processing elements while the SIMD targets the Intel Paragon and the IBM SP2 platforms. The implementation of [8] is used more often for simulation purposes and is done on an adaptive grid machine called GrACE.

While such solutions can be useful for experimental purposes and for proof of concept, it is not applicable to real autonomous systems. Parallel machines, for example, cannot be used in an embedded environment because the power consumption of workstations mounted on a robot will allow the robot only to drive a few meters. Moreover, the size of the robot must be sufficiently large to carry the PC, thus leading to a very large system.

Some effort to tackle the aforementioned problem has been done in [11–13]. In [12] the goal is to have a real-time implementation of the feature tracking using a hardware platform. The target system is a C4x board featuring eight Texas instrument processors C4x running at 50 MHz. Each processor is assigned a specific task. One processor grabs the frame, two processors perform feature selection, and one processor performs motion estimation. Feature tracking is done by three processors and the rendering is done by one processor. The system is able to process 0.8 frames per second for feature detection (100 features) and 4 frames per second for feature tracking, leading to an overall performance of 0.8 frames per second. Although the size of this system as well as its power consumption remains low compared to a software solution, it is still far from being suitable to be used in a mobile system.

[2] The Harris corner detector computes the locally averaged moment matrix computed from the image gradients, and then combines the eigenvalues of the moment matrix to compute a corner *strength*, of which maximum values indicate the corner positions.



(a)

(b)

Figure 1: Feature detection and selection.

Some recent works [11, 13] have targeted FPGAs for an efficient implementation of feature tracking. In [11] features are selected in an FPGA mounted on a PCI-Board, which is embedded in a workstation. Not only cannot this solution be used in small mobile environments, it also presents the drawback that the processor, a 3 GHz Pentium, must be used all the time for data transfer between the FPGA and the processor.

The system in [13] is a more compact hardware/software system. The software part is implemented on a PowerPC processor that is attached to an FPGA in which the hardware part is implemented. The FPGA is in charge of the image enhancement that is done using a high pass filtering process. The implementation uses a sliding window that is used to capture the neighborhood of an incoming pixel. The latter is then used to compute the enhanced value of the pixel that is stored in a memory shared by the PowerPC and the FPGA. The adaptive process is done via the modification of the filter parameters as well as the threshold parameter for the number of features to be selected. Because the single available memory can be accessed only by one module (processor or FPGA) at a time, the streaming computation process will be delayed leading to a decrease in the number of frames that may be processed in a second. The FPGA is used in this system as an ASIC, since no reconfiguration is done. Because the structure of filters varies according to the algorithms used, a simple change in the filter coefficient is not sufficient to replace a filter in the FPGA. The Sobel operator, for example, is two dimensional while the Laplace is only one dimensional. Therefore, a Laplace filter cannot become a Sobel just by replacing the coefficients. The configuration of the FPGA can be used to replace the complete filter structure.

This work presents a better use of a single chip to implement an embedded adaptive hardware/software solution to feature tracking. The system is optimized to perform computations on all the streamed frames without delay. We also exploit the possibilities to dynamically extend the instruction set of the embedded processor by binding an accelerator directly to the processor. Adaptivity is done by means of configuration rather than by parameter modifications as is the case in [13].

4. DESIGN OF A HARDWARE/SOFTWARE SOLUTION FOR FPGA

This section describes our implementation of the chosen feature tracking algorithm. The data flow from the incoming images to the positions of image movements looks like this: video in → feature detection → feature selection → feature tracking → further processing.

The feature detection is highly parallelizable and can be implemented completely in hardware (see Figure 2). A compilation of five convolve filters (for I_x, I_y, G_{xx}, G_{xy}, and G_{yy}) and simple arithmetic operations is used. Usually convolving is realized using a sliding window, whose size can be 3×3, 5×5, and so on. In case of a 3×3 window, incoming pixel data must fill up two line buffers before the first calculation is possible. The latency results in $2\ \textit{lines} + 2\ \textit{pixels} + 1$ clock cycles. The corner strength can be computed by add, subtract, and multiply units. Down shifting provisional results prevent arithmetical overflows. The factor $k = 0.06$ in $s = d - kt^2$ can be approximated by shifting by 4 ($\widehat{=} k = 0.0625$). The FPGA used is equipped with single clock cycle multipliers. Thus, the corner strength calculation needs only three further clock cycles to be completely computed while using only integer numbers.

For feature selection nonmax suppression is used, which is realized by a sliding window, too. Each value within the

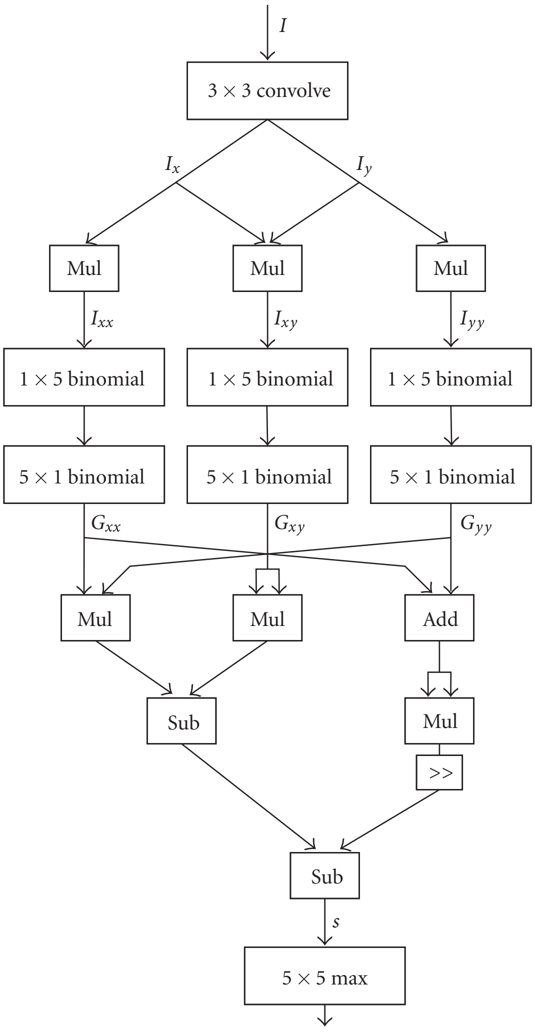


Figure 2: Logic for feature detection.

window is compared with the center value. If it is the maximum the window position is declared a feature point.

For a system on-chip layout, these preliminary ideas allow a hardware/software partitioning as shown in Figure 3(a). The feature detection and selection is completely implemented in a dedicated hardware module (FT) and feature tracking is done in software by the Xilinx MicroBlaze processor. The video frames are captured by the videoin module and stored in the SRAM. The RS232 module is used for debugging purposes.

Considering the data transfer, there is communication between video input module and feature module, between feature module and memory in order to access the current frame and store the selected features, and between processor and memory in order to continue with tracking these features. All transactions simultaneously utilize the bus, which is in general only designed for low peripheral load. The amount of data produced by video streams is very high and clutters up the bus. That means that this solution is not fit to process data in real time.

4.1. *Efficient hardware/software partitioning*

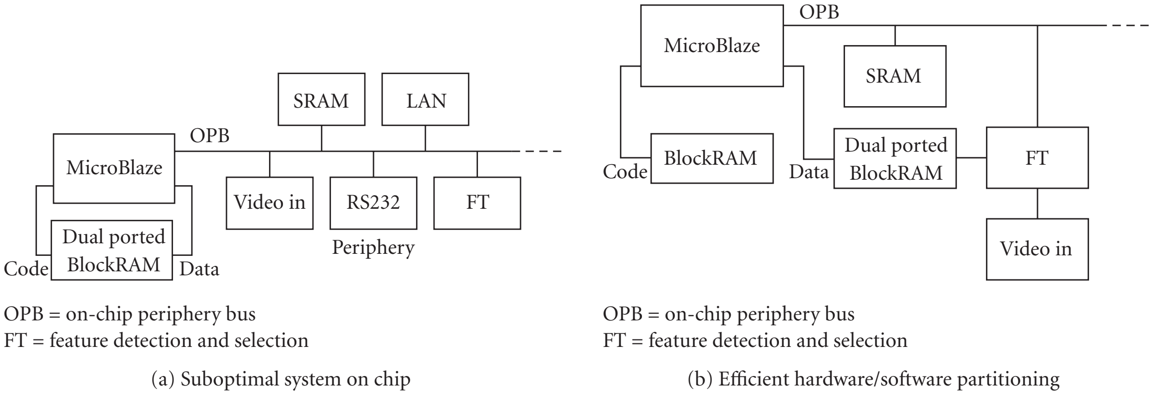
By rearranging the components while accounting for the data flow, performance can be improved. Our architecture is shown in Figure 3(b). The videoin module sends image sequences to the feature module, which stores image data and selected features in the memory. A dual ported memory is used that can be accessed from two different clients and clock domains. The feature module is, furthermore, connected to the bus to exchange information with the processor. This information consists of controlling instructions like "start" and "stop," but also of parameters like the base address of image data in the memory or parameters which influence the processing. The BlockRAM is the main memory of the processor and holds data like variables and heap of the application running on it. A certain area of the memory is reserved for image and feature data by the application. To notify the feature module about the location of this area the base address is transmitted by the processor to the module over the bus.

4.2. *Dynamic reconfiguration increases flexibility*

The Xilinx FPGAs allow partial reconfiguration of each individual column. The Erlangen slot machine (ESM) [14] is an architecture that exploits this feature. Its new concept allows an unrestricted relocation of modules on the device. The FPGA is logically divided into *slots* which can be reprogrammed independently. Via a programmable crossbar each module can, regardless of placement, communicate with its peripherals and also with other modules. Memory banks are vertically attached to each slot, providing enough memory space to store temporary data. In streaming applications, this memory can also be used for shared memory communication between neighboring modules. Smaller data chunks can be transferred either by placing (dual ported) BlockRAM between them or via a reconfigurable multiple bus (RMB).

Figure 4 shows the possible placement of our feature tracker on the ESM platform. The data flow was already described above. Using multiple memory banks, a technique called double buffering can further increase performance: image frames are filled alternately into two banks by the videoin module. The feature module reads out data but always from the respectively other bank. Hence, in contrast to a single memory architecture, no bottleneck will occur while accessing the memory.

Reconfiguration highly increases flexibility of the feature tracker. This means that, for example, the source of incoming image stream can simply be an analog video input as well as a firewire or LAN connection. By reconfiguration the input module can be replaced by an appropriate one. Image prefiltering, like illumination compensation or the Gaussian function to smooth image noise, increases the quality of the tracking results. In addition, feature detection and selection processes can be altered or exchanged to adapt to the system environment. The tracker unit can use dedicated helper hardware, fore example, to speed up the comparison of features (see next subsection). In contrast to works like [13] we



(a) Suboptimal system on chip

(b) Efficient hardware/software partitioning

Figure 3: System architecture.

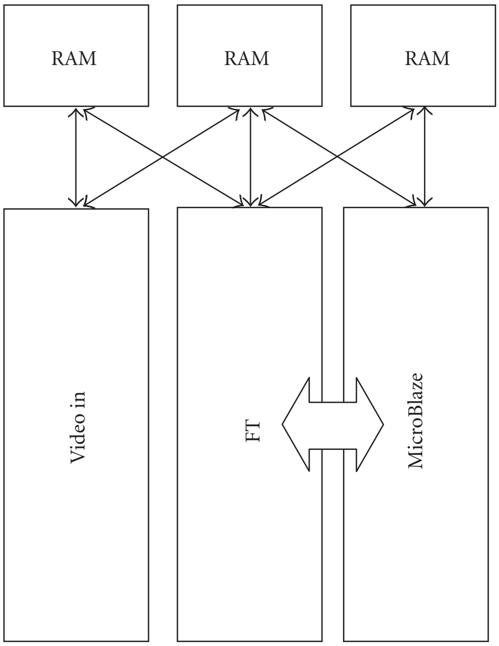


Figure 4: Mapping of the feature tracker on the column-based reconfigarable device.

rely on reconfiguration rather than on just exchanging module parameters to increase flexibility.

4.3. Instruction set metamorphosis

By analyzing the remaining part of the algorithm, namely, feature tracking, which is done in software, some more improvements are possible. The features of the MicroBlaze processor can be upgraded. Eight fast communication channels, called fast simplex links (FSL), are available to connect dedicated hardware accelerators, which are linked through FIFO buffers. The instruction set offers special put and gets commands to access these pipelines by software (shown in Figure 5).

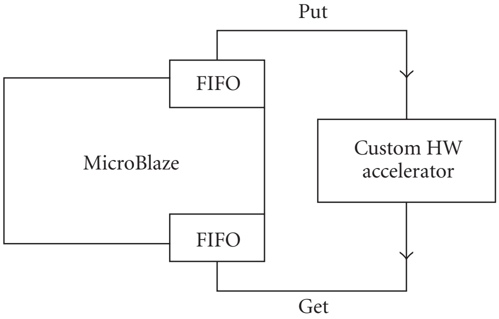


Figure 5: Instruction set extension through fast simplex link.

The tracking code reads each image point from the memory to calculate the parameters for the normalized correlation (8). A software implementation only allows a sequential computation of each individual pixel. We take advantage of the FSL and the dedicated hardware attached to it to increase the throughput as well as the speed of feature comparison. Since one pixel is represented by 8 bits and the FSL and memory bus width are both 32 bits, we are able to transfer and process four pixels at once. The hardware accelerator simultaneously calculates the sum A and the sum of squares B while the processor only pushs data into the FIFO. A similar procedure is used to compute scalar products D while feature comparison phase.

5. RESULTS

Our design was implemented on a Spartan 3 (xc3s400) FPGA. This FPGA is to small to host all hardware accelerators together with the microBlaze processor. Thus our first implementation is a software only version which runs completely on the microBlaze.

Table 1: Performance of software only solution.

	Spartan 3 board	ML310 board (Virtex2 Pro)
Feature detection	5.01 s	153 ms
Feature selection (30 features)	0.89 s	78 ms
Tracking	1.6 s	28 ms

Table 2: Pipeline stages for hardware feature detection and selection and their latencies. Examples with different image formats and system clocks.

Stage	Logic	Purpose	Latency
1	3 × 3 convolve	I_x and I_y	$2l+2+1$
2	3 multipliers	I_* products	1
3	1 × 5 convolve	Binomial horiz.	$4+1$
4	5 × 1 convolve	Binomial vert.	$4l+1+1$
5	Arithmetic	d, t, s	3
6	5 × 5 convolve	Feature selection	$4l+4+1$
Total			$10l+19$
QCIF (176 × 144 pixels), 50 MHz			35.58 μs
QCIF (176 × 144 pixels), 100 MHz			17.79 μs
VGA (640 × 480 pixels), 100 MHz			64.19 μs

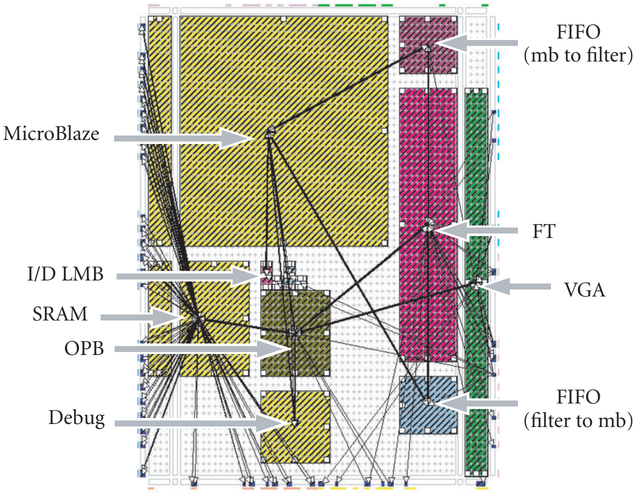


Figure 6: Floorplan of the feature tracker on a Xilinx Spartan 3 FPGA.

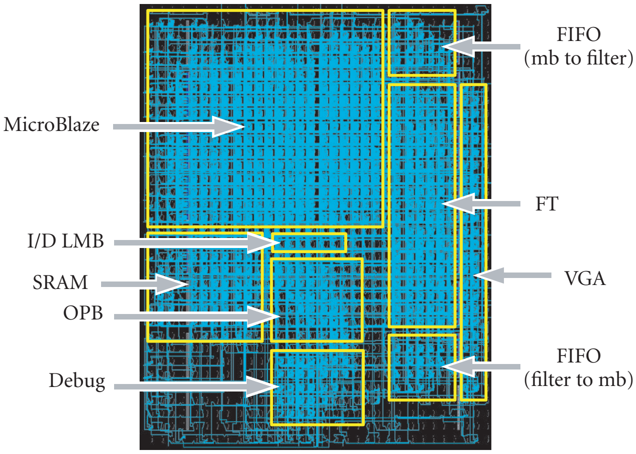


Figure 7: Placed and routed design of the feature tracker.

Figure 6 shows the placement of the modules in the floorplanner tool. The resulting design in its placed and routed form can be seen in Figure 7.

Considering the timings of a software only solution (which takes no advantage of hardware accelerators) feature detection takes 5.01 s executed on the introduced design. The latency for feature selection is proportional to the amount of features found, for example, the latency for 80 features is 2.39 s. As we will see in the following this is much slower than the hardware solution. The tracking part takes 1.6 s per frame.

Porting to a Xilinx ML310 board equipped with a Virtex2 Pro FPGA and using a newer version of the development tools further increased the performance and allowed timings in the range of milliseconds (see Table 1).

Table 2 summarizes the pipeline stages of the hardware feature detection and selection module (independent from the complete design) and their latencies. The underlying algorithm was already described in Section 2 so only some remarks follow: Stage 2 computes the values I_xI_x, I_xI_y, and I_yI_y and stage 3 and 4 produce the values G_{xx}, G_{xy}, and G_{yy}. Stage 5 uses arithmetic units to calculate the corner response strength s. Finally stage 6 selects features using a modified convolve filter. The total latency results in $10l+19$ clock cycles, where l is the image width. The table also shows examples for different image formats and system clocks.

Because this design processes one pixel per clock, very high frame rates are achieved. The frame rate can be calculated as system clock divided by image size, for example, 1972 fps for a QCIF format or 162 fps for a VGA (640 × 480 pixels) resolution. Of course the tracking part that runs in software on the MicroBlaze cannot achieve this high performance and thus is the bottleneck in this case. The tracking delay of 28 ms allows about 3-4 frames per second with a video stream in QCIF format.

6. CONCLUSIONS

In this paper, we have designed and implemented an efficient and flexible feature tracker on a reconfigurable device. The efficiency is obtained by using a viable hardware/software partitioning, by communication between modules, as well as by using an efficient memory access. Furthermore, the exploitation of the MicroBlaze features, like the fast simplex link, improves the performance further. Contrary to other works that modify the parameters of some filter to increase flexibility, we use reconfiguration to exchange hardware modules with different structures. The progress made in the last decade have affected the power consumption and the size of FPGAs, their costs have dropped rapidly while their capacity continues to increase. This trend is expected to continue, allowing the use of FPGAs in mobile autonomous environments. Our future work is to further improve the tracking part of our solution and the deployment in a system of cooperative intelligent robots using FPGAs as computing platform.

REFERENCES

[1] P. Lysaght, "FPGAs in the decade after the von Neumann century," in *Friday workshop at Design, Automation and Test European (DATE '06)*, Munich, Germany, March 2006.

[2] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pp. 674–679, Vancouver, BC, Canada, August 1981.

[3] C. Tomasi and T. Kanade, "Detection and tracking of point features," Tech. Rep. CMUCS-91-131, Carnegie Mellon University, Pittsburgh, Pa, USA, 1991.

[4] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pp. 593–600, Seattle, Wash, USA, June 1994.

[5] J. Shi and C. Tomasi, "Good features to track," Tech. Rep. TR-93-1399, Department of Computer Science, Cornell University, Ithaca, NY, USA, 1993.

[6] T. Zinßer, C. Gräßl, and H. Niemann, "Efficient feature tracking for long video sequences," in *Proceedings of the 26th DAGM Symposium*, pp. 326–333, Tübingen, Germany, August-September 2004.

[7] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," Tech. Rep. CN5300, Sarnoff Corporation, Princeton, NJ, USA, 2004.

[8] J. Chen, D. Silver, and M. Parashar, "Real time feature extraction and tracking in a computational steering environment," in *Proceedings of the High Performance Computing Symposium (HPC '03)*, pp. 155–160, Society for Modeling and Simulation International, San Diego, Calif, USA, March-April 2003.

[9] M. B. Kulaczewski and H. J. Siegel, "Implementations of a feature-based visual tracking algorithm on two MIMD machines," in *Proceedings of the International Conference on Parallel Processing*, pp. 422–430, Bloomington, Ill, USA, August 1997.

[10] M. B. Kulaczewski and H. J. Siegel, "SIMD and mixed-mode implementations of a visual tracking algorithm," in *Proceedings of the International Parallel Processing Symposium (IPPS/SPDP '98)*, pp. 716–720, Orlando, Fla, USA, March-April 1998.

[11] A. Bissacco, J. Meltzer, S. Ghiasi, S. Soatto, and M. Sarrafzadeh, "Fast visual feature selection and tracking in a hybrid reconfigurable architecture," Tech. Rep., UCLA, Los Angeles, Calif, USA, 2004. http://www.cs.ucla.edu/~bissacco/hybridfeatrack.html.

[12] X. Feng and P. Perona, "Real time motion detection system and scene segmentation," Tech. Rep. CIT-CDS-98-004, California Institute of Technology, Pasadena, Calif, USA, 1998.

[13] S. Ghiasi, A. Nahapetian, H. J. Moon, and M. Sarrafzadeh, "Reconfiguration in network of embedded systems: challenges and adaptive tracking case study," *Journal of Embedded Computing*, vol. 1, no. 1, pp. 147–166, 2005.

[14] C. Bobda, M. Majer, A. Ahmadinia, et al., "The erlangen slot machine: a highly flexible fpga-based reconfigurable platform," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 319–320, Napa, Calif, USA, April 2005.

Felix Mühlbauer completed his degree in the diploma course of studies in computer science at the University of Erlangen-Nuremberg, in 2005. Since November 2005 he works as a Scientific Assistant in the Self-Organizing Embedded Systems group in the Department of Computer Sciences at the University of Kaiserslautern .

Christophe Bobda is the leader of the new created working group "Self-Organizing Embedded Systems" in the Department of Computer Sciences at the University of Kaiserslautern. He received the Licence in Mathematics from the University of Yaounde, Cameroon, in 1992, the diploma of computer science and the Ph.D. degree (with honors) in computer science from the University of Paderborn in Germany, in 1999 and 2003, respectively. In June 2003, he joined the Department of Computer Science at the University of Erlangen-Nuremberg in Germany as a Postdoctoral. He received the Best Dissertation Award 2003 from the University of Paderborn for his work on synthesis of reconfigurable systems using temporal partitioning and temporal placement. He is a Member of the IEEE Computer Society, the ACM, and the GI. He is also in the program committee of several conferences (FPT, RAW, RSP, ERSA, DRS), the DATE executive committee as proceedings chair (2004, 2005, 2006). He served as a Reviewer of several journals (IEEE TC; IEEE TVLSI; Elsevier Journal of Microprocessor and Microsystems; Integration, the VLSI Journal) and conferences (DAC, DATE, FPL, FPT, SBCCI, RAW, RSP, ERSA).

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 48085, Pages 1–19
DOI 10.1155/ES/2006/48085

Speech Silicon: An FPGA Architecture for Real-Time Hidden Markov-Model-Based Speech Recognition

Jeffrey Schuster, Kshitij Gupta, Raymond Hoare, and Alex K. Jones

University of Pittsburgh, Pittsburgh, PA 15261, USA

Received 21 December 2005; Revised 8 June 2006; Accepted 27 June 2006

This paper examines the design of an FPGA-based system-on-a-chip capable of performing continuous speech recognition on medium-sized vocabularies in real time. Through the creation of three dedicated pipelines, one for each of the major operations in the system, we were able to maximize the throughput of the system while simultaneously minimizing the number of pipeline stalls in the system. Further, by implementing a token-passing scheme between the later stages of the system, the complexity of the control was greatly reduced and the amount of active data present in the system at any time was minimized. Additionally, through in-depth analysis of the SPHINX 3 large vocabulary continuous speech recognition engine, we were able to design models that could be efficiently benchmarked against a known software platform. These results, combined with the ability to reprogram the system for different recognition tasks, serve to create a system capable of performing real-time speech recognition in a vast array of environments.

1. INTRODUCTION

Many of today's state-of-the-art software systems rely on the use of hidden Markov model (HMM) evaluations to calculate the probability that a particular audio sample is representative of a particular sound within a particular word [1, 2]. Such systems have been observed to achieve accuracy rates upwards of 95% on dictionaries greater than 1000 words; however, this accuracy comes at the expense of needing to evaluate hundreds of thousands of Gaussian probabilities resulting in execution times of up to ten times the real-time requirement [3]. While these systems are able to provide a great deal of assistance in data transcription and other offline collection tasks, they do not prove themselves as effective in tasks requiring real-time recognition of conversational speech. These issues combined with the desire to implement speech recognition on small, portable devices have created a strong market for hardware-based solutions to this problem. Figure 1 gives a conceptual overview of the speech recognition process using HMMs. Words are broken down into their phonetic components called phonemes. Each of the grey ovals represents one phoneme, which is calculated through the evaluation of a single three state HMM. The HMM represents the likelihood that a given sequence of inputs, senones, is being traversed at any point in time. Each senone in an HMM represents a subphonetic sound unit, defined by the particular speech corpus of interest. These senones are generally composed of a collection of multivariant Gaussian distributions found through extensive offline training on a known test set. In essence, each HMM operates as a three-state finite-state machine that has fixed probabilities associated with the arcs and a dynamic "current state" probability associated with each of the states, while each word in the dictionary represents a particular branch of a large, predefined tree style search space.

The set of senones used during the recognition process is commonly referred to as the acoustic model and is calculated using a set of "features" derived from the audio input. For our research we chose to use the RM1 speech corpus which contains 1000 words, and uses an acoustic model comprised of 2000 senones [4]. The RM1 corpus represents the most common words used in "command-and-control" type tasks and can be applied to a large number of tasks from navigation assistance to inventory ordering systems. This particular dictionary also represents a medium-sized task (100–10 000 words) and presents a reasonable memory requirement for a system looking to be implemented as a single-chip solution. This corpus requires that every 10 milliseconds, 300 000 operations must be performed to determine the probability that a particular feature set belongs to a given multivariant

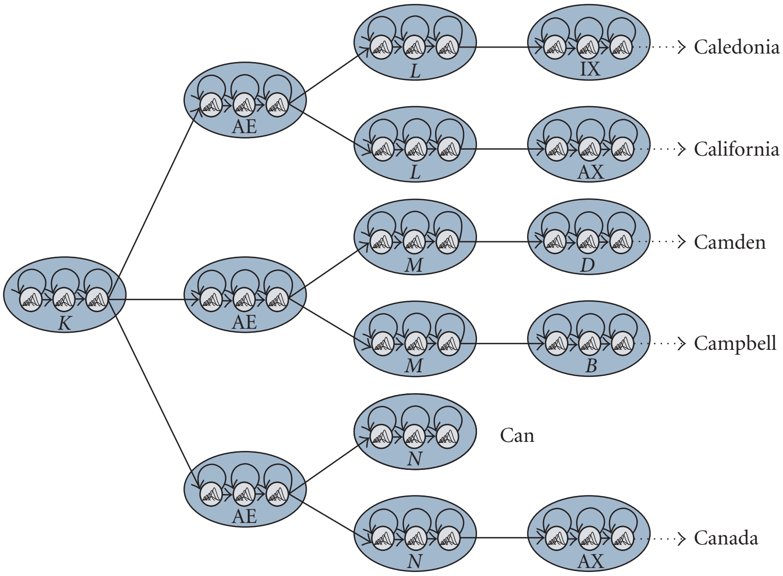


Figure 1: Conceptual overview of speech recognition using hidden Markov models.

Gaussian distribution, resulting in over 60 million calculations per second, just to calculate the senones.

1.1. Background

Although several years of research has gone into the development of speech recognition, the progress has been rather slow. This is a result of several limiting factors, amongst which recognition accuracy is the most important. The ability of machines to mimic the human auditory perceptory organs and the decoding process taking place in the brain has been a challenge, especially when it comes to the recognition of natural, irregular speech [5].

To date, however, state-of-the-art recognition systems overcome some of these issues for systems with regular speech structures, such as command- and control-based applications. These systems provide accuracies in excess of 90% for speaker independent systems with medium sized dictionaries [6]. Despite the satisfactory accuracy rate achieved for such applications, speech recognition has yet to penetrate our day-to-day lives in a meaningful way.

The majority of this problem stems from the computationally intensive nature of the speech recognition process, which generally requires several million floating-point operations per second. Unfortunately using general purpose processors (GPPs) with traditional architectures is inefficient due to limited numbers of arithmetic logic units (ALUs) and insufficient caching resources. Cache sizes in most processors available today, especially those catering towards embedded applications, are very limited: only on the order of tens of kBs [7]. Therefore, accessing tens of MBs of speech data using tens of kBs of on-chip cache results in a high cache miss rate thereby leading to pipeline stalls and significant reduction in performance.

Further, since several peripherals and applications running on a device need access to a common processor, bus-based communication is required. Thus, all elements connected to the bus are synchronized by making use of bus transaction protocols thereby incurring several cycles of additional overhead. Because of these inefficiencies, speech recognition systems execute less than one instruction per cycle (IPC) [1, 2] on GPPs. As a result, the process of recognizing speech by such machines is slower than real time [3].

To counter these effects, implementers have two options. They could either use processors with higher clock-rates to account for processor idle time caused by pipeline stalls and bus arbitration overheads, or they could redesign the processor that caters to the specific requirements of the application. Since software-based systems are dependent on the underlying processor architecture, they tend to take the first approach. This results in the need for devices with multi-GHz processors [1, 2] or the need to reduce the model complexity. However, machines with multi-GHz processors are not always practical, especially in embedded applications. The alternative is to reduce bit-precision or use a more coarse-grain speech model to decrease the data size. While this helps in making the system practically deployable, the loss in computational precision in most cases, leads to degraded performance (in terms of accuracy) and decreases the robustness of the system. For example, a speaker-independent system becomes a speaker-dependent system or continuous speech recognition moves to discrete speech recognition.

The second option involves designing a dedicated architecture that optimizes the available resources required for processing speech and allows for the creation of dedicated data-paths that eliminate significant bus transaction overhead.

Projects at the University of California at Berkeley, Carnegie Mellon University, and the University of Birmingham in the United Kingdom have made some progress with hardware-based speech recognition devices in recent years [8, 9]. These previous attempts either had to sacrifice model complexity for the sake of memory requirements or simply encountered the limit of the amount of logic able to be placed on a single chip. For example, the solution in [8] is to create a hardware coprocessor to accelerate a portion of speech recognition, beam search. The solution in [9] requires device training. In contrast, our work presents a novel architecture capable of solving the entire speech recognition problem in a single device with a model that does not require training through the use of task specific pipelines connected via shared, multiport memories. Thus, our implementation is capable of processing a 1000 word command and control-based application in real time with a clock speed of approximately 100 MHz.

The remainder of this paper describes the speech silicon project, providing an in-depth analysis of each of the pipelines derived for the system-on-a-chip (SoC). Specifically, we introduce a novel architecture that enables real-time speech recognition on an FPGA utilizing the 90 nm ASIC multiply-accumulate and block RAM features of the Xilinx Virtex 4 series devices. Final conclusions as well as a summary of synthesis and post place-and-route results will be given at the end of the paper.

2. THE SPEECH SILICON PROJECT

The hardware speech processing architecture is based on the SPHINX 3 speech recognition engine from Carnegie Mellon University [10]. Through analysis of this algorithm, a model of the system was created in MATLAB. As a result, complex statistical analysis could be performed to find which portions of the code could be optimized. Further, the data was able to be rearranged into large vectors and matrices leading to the ability to parallelize calculations observed to be independent of one another. Preliminary work on this topic has been discussed in [11, 12].

The majority of automatic speech recognition engines on the market today consist of four major components: the feature extractor (FE), the acoustic modeler (AM), the phoneme evaluator (PE), and the word modeler (WM), each presenting its own unique challenge. Figure 2 shows a block diagram for the interaction between the components in a traditional software system, with inputs from a DSP being shown on the left of the diagram.

The FE transforms the incoming speech into its frequency components via the fast fourier transform, and subsequently generates mel-scaled Cepstral coefficients through mel-frequency warping and the discrete cosine transform. These operations can be performed on most currently available DSP devices with very high precision and speed and will therefore not be considered for optimization within the scope of this paper.

The AM is responsible for evaluating the inputs received from the DSP unit with respect to a database of known Gaussian probabilities. It produces a normalized set of scores, or senones, that represent the individual sound units in the database. These sound units represent subphonetic components of speech and are traditionally used to model the beginning, middle, and end of a particular phonetic unit. Each of the senones in a database is comprised of a mixture of multivariant Gaussian probability distribution functions (PDFs) each requiring a large number of complex operations. It has been shown that this phase of the speech recognition process is the most computationally intensive, requiring up to 95% of the execution time [2, 13], and therefore requires a pipeline with very high bandwidth to accommodate the calculations.

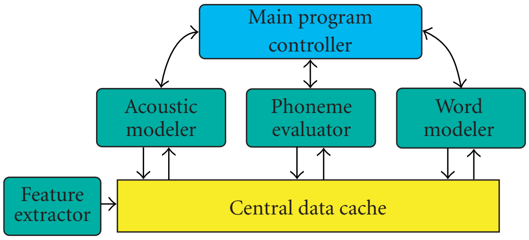


FIGURE 2: Block diagram of software-based automatic speech recognition system.

The PE associates groups of senones into HMMs representing the phonetic units, phonemes, allowable in the systems dictionary. The basic calculations necessary to process a single HMM are not extremely complex and can be broken down into a simple ADD-COMPARE-ADD pipeline, described in detail in Section 4. The difficulty in this phase is in managing the data effectively so as to minimize unnecessary calculations. When the system is operational not all of the phonemes in the dictionary are active all the time, and it is the PE that is responsible for the management of the active/inactive lists for each frame. By creating a pipeline dedicated to calculating HMMs and combining it with a second piece of logic that acts as a pruner for the active list, a two step approach was conceived for implementing the PE, allowing for maximal efficiency.

The WM uses a tree-based structure to string phonemes together into words based on the sequences defined in the system dictionary. This block serves as the linker between the phonemes in a word as well as the words in a phrase. When the transition from one word to another is detected, a variable penalty is applied to the exiting word's score depending on what word it attempts to enter next. In this way, basic syntax rules can be implemented in addition to pruning based on a predefined threshold for all words. WM is also responsible for resetting tokens found inactive by the PE. The pruning stage of the PE passes two lists to the WM, one for active tokens and the other for newly inactive tokens. Much like the PE, the WM takes a two stage approach, first resetting the inactive tokens and then processing the active tokens. By doing the operations in this order we ensure that while processing the active tokens, all possible successor tokens are available if and when they are needed.

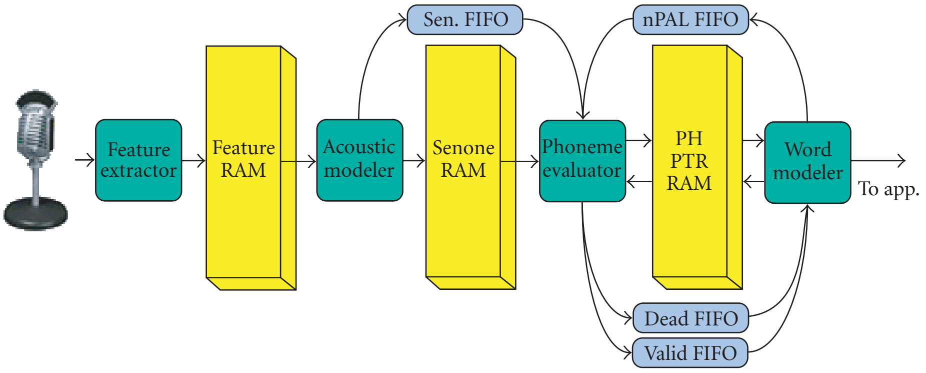


Figure 3: Block diagram of the *speech silicon* hardware-based ASR system.

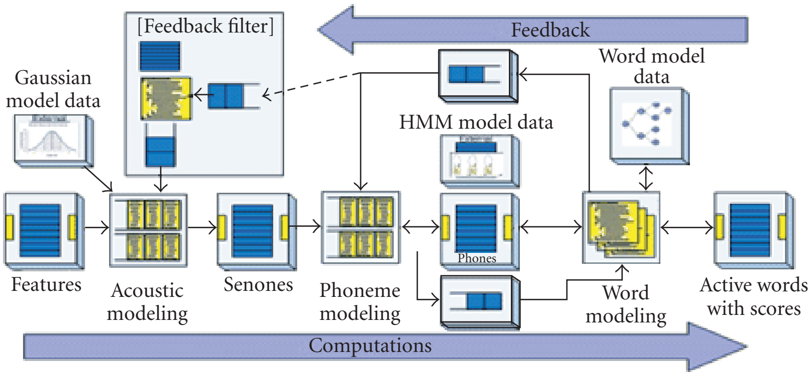


Figure 4: Conceptual diagram of high-level architecture.

When considering such systems for implementation on embedded platforms the specific constraints imposed by each of the components must be considered. Additionally, the data-dependencies between all components must be considered to ensure that each component has the data it requires as soon as it needs it. To complicate matters, the overall size of the design and its power consumption must also be factored into the design if the resultant technology is to be applicable to small, hand-held devices. The most effective manner for accommodating these constraints was determined to be the derivation of three separate cells, one for each of the major components considered, with shared memories creating interface between cells. To minimize the control logic and communication between cells, a token-passing scheme was implemented using FIFOs to buffer the active tokens across cell boundaries. A block diagram of the component interaction within the system is shown in Figure 3.

By constructing the system in this fashion and keeping the databases necessary for the recognition separate from the core components, this system is not bound to a single dictionary with a specific set of senones and phonemes. These databases can in fact be reprogrammed with multiple dictionaries in multiple languages, and then given to the system for use with no required changes to the architecture. This flexibility also allows for the use of different model complexity in any of the components, allowing for a wide range of input models to be used, and further aiding in the customizability of the system. Figure 4 shows a detailed diagram of the high-level architecture of the speech recognition engine.

2.1. Preliminary analysis

During the conceptual phase of the project, one major requirement was set: the system must be able to process all data in real time. It was observed that speech recognition for a 64 000 word task was 1.8 times slower than real time on a 1.7 GHz AMD Athalon processor [14]. Additionally, the models for such a task are 3 times larger than the models used for the 1000-word command and control task on which our project is focused. Therefore, extending this linearly in terms of the number of compute cycles required, it can be said that a 1000-word task would take 1.6 times real time, or 160% longer than real time, to process at 1.7 GHz. Thus, a multi-GHz processor cannot handle a 1000-word task in real time, and custom hardware must be considered to help expedite the process. This certainly eliminates real-time speech

Table 1: Number of compute cycles for three different speech corpuses.

Speech corpus	No. of words	No. of gaussians	No. of evaluations per frame
TI digits	12	4816	192 600
RM1	1000	15 480	619 200
HUB-4	64 000	49 152	1 966 080

Table 2: Timing requirements for frame evaluation.

	AM	PE	WM	Total
No. of cyles [per 10 ms frame]	603 720	8192	102 400	714 312
Memory bandwidth [MB/sec]	495	—	5	—

processing from mobile phones and PDAs due to the far more limited capabilities of embedded processors.

In modern speech processing, incoming speech is sampled every 10 milliseconds. By assuming a frame latency of one for DSP processing, it can be said that a real-time hardware implementation must execute all operations within 10 milliseconds. To find our total budget a series of experiments were conducted on open-source SPHINX models [15, 16] to observe the cycle counts for different recognition tasks. Table 1 summarizes the results of these tests for three different sized tasks: digit recognition [TI Digits], command and control [RM1], and continuous speech [HUB-4].

The table shows the number of "compute cycles" required for the computation of all Gaussians for different tasks assuming a fully pipelined design. It can be seen that assuming one-cycle latency for memory accesses, the RM1 task would require 620 000 compute cycles, while HUB4 would require 2 million cycles. Knowing that we need to process all of the data within a 10- milliseconds window we observe that the minimum operating speeds for systems performing these tasks would be 62 MHz and 200 MHz, respectively.

Since the computation of Gaussian probabilities in AM constitutes the majority of the processing time, keeping some cushion for computations in the PHN and WRD blocks, it was determined that 1 million cycles would be sufficient to process data for every frame for RM1 task. Therefore a minimum operating speed of 100 MHz was set for our design. Having set the target frequency, a detailed analysis of the number of compute cycles was performed and is summarized in Table 2.

The number of cycles presented in this table is based on the assumption that all computations are completely pipelined. While a completely pipelined design is possible in the case of AM and PHN, computations in the WRD block do not share such luxury. This is a direct result of the variable branching characteristic of the word tree structure. Hence, to account for the loss in parallelism, the computation latency (estimated at a worst case of 10 cycles) has been accounted into the projected cycles required by the WRD block.

Further, the number of cycles required by the PE and WM blocks is completely dependent on the number of phones/words active at any given instant. Therefore, an analysis of the software was performed to obtain the maximum number of phones active at any given time instant. It was observed from SPHINX 3.3 for an RM1 dictionary, a maximum of 4000 phones were simultaneously active. Based on this analysis a worst case estimate of the number of cycles required for the computation is presented in the table.

3. ACOUSTIC MODELER

Acoustic modeling is the process of relating the data received from the FE, traditionally Cepstral coefficients and their derivatives, to statistical models found in the system database, which can account for 70% to 95% of the computational effort in modern HMM-based ASR systems [2, 13]. Each of the i senones, in the database are made up of c components, each one representing a d-dimensional multivariant Gaussian probability distribution. The components of a senone are log-added [17] to one another to obtain the probability of having observed the given senone. The equations necessary to derive a single senone score are shown in (1)–(6).

$$P(\overline{X}) = \frac{1}{\sqrt{(2\pi)^D \,|\, \overline{V}^* \,|}} e^{-\sum_{d=1}^{D} ((X_d - \mu_d)^2 / 2 * \sigma_d^2)} \qquad (1)$$

$$\ln\left(P(\overline{X})\right) = -0.5 \ln\left[(2\pi)^D \,|\, \overline{V}^* \,|\,\right] - \sum_{d=1}^{D} \frac{\left(X_d - \mu_d\right)^2}{2 * \sigma_d^2}. \qquad (2)$$

Consider the first term on the left-hand side of (2). If the variance matrix V is constant, then the V^* term will also be constant, making the entire term a predefined constant K. Additionally, the denominator of the second term can be factored out and replaced with a new variable Ω_d that can be used to create a simplified version of the term $\text{Dist}(X)$. $\text{Dist}(X)$ becomes solely dependent on the d-dimensional input vector X. These simplifications are summarized in the three axioms below with a simplified version of (2) given as (3)

$$\begin{aligned} &\text{let: } K = -0.5 \ln\left[(2\pi)^D \,|\, \overline{V}^* \,|\,\right], \\ &\text{let: } \text{Dist}(\overline{X}) = \sum_{d=1}^{D} \frac{\left(X_d - \mu_d\right)^2}{2 * \sigma_d^2} = \sum_{d=1}^{D} \left(X_d - \mu_d\right)^2 * \Omega_d, \\ &\text{let: } \Omega_d = \left(\frac{0.5}{\sigma_d^2}\right), \\ &\ln\left(P(\overline{X})\right) = K - \text{Dist}(\overline{X}). \end{aligned} \qquad (3)$$

Equation (3) serves to represent the calculations necessary to find a single multidimensional Gaussian distribution, or component. From here we must combine multiple components with an associated weighting factor to create senones as summarized in (4):

$$S_i(\overline{X}) = \sum_{c=1}^{C} \left[W_{i,c} * P_{i,c}(\overline{X})\right]. \qquad (4)$$

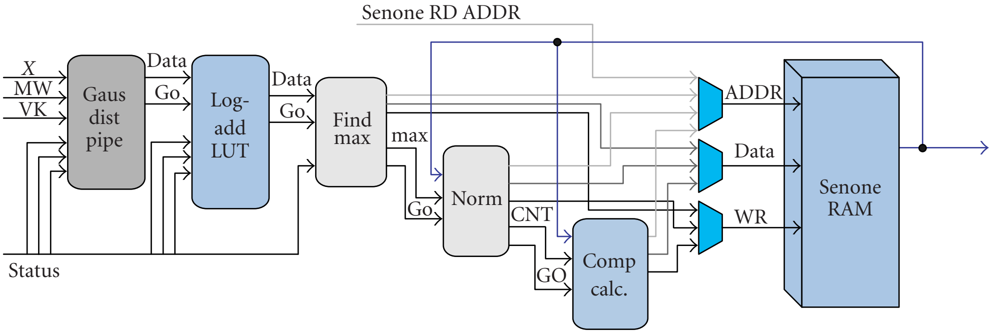


Figure 5: Block diagram of acoustic modeling pipeline.

At this point in our models it is necessary to define a log-base conversion factor, ψ, in order to stay in line with the SPHINX models used as our baseline. The use of a conversion factor in these equations is useful in transforming the $P_{i,c}(X)$ term of (4) into the required $\ln(P_{i,c}(X))$ term required for insertion of (3), but the use of the specific value is unique to the SPHINX system. By moving into the log-domain, the multiplication of (4) can also be transformed into an addition helping to further simplify the equations. The following axioms define the conversion factor with the result of its insertion shown in (5)–(6):

$$\text{let: } \psi = 1.0003,$$

$$\text{let: } f = \frac{1}{\ln(\psi)},$$

$$f * \ln\left[S_i(\overline{X})\right] = \log_\psi\left[S_i(\overline{X})\right], \tag{5}$$

$$\log_\psi\left[S_i(\overline{X})\right] = \log\sum_{c=1}^{C}\left[\log_\psi\left(W_{i,c}\right) + \log_\psi\left(P_{i,c}(\overline{X})\right)\right],$$

$$\text{let: } W'_{i,c} = \log_\psi\left(W_{i,c}\right),$$

$$\log_\psi\left[S_i(\overline{X})\right] = \log\sum_{c=1}^{C}\left[W'_{i,c} + \log_\psi\left(P_{i,c}(\overline{X})\right)\right]. \tag{6}$$

The values μ, σ, V, K, and W relate to specific speech corpus being used and represent the mean, standard deviation, covariance matrix, scaling constant, and mixture weight, respectively. These values are stored in ROMs that are otherwise unassociated with the system and can be replaced or reprogrammed if a new speech corpus is desired. The f & Ψ values are log-base conversion factors ported directly out of the SPHINX 3 algorithm and the X vector contains the Cepstral coefficient input values provided by the FE block.

For our system we chose to use the 1000-word RM1 dictionary provided by the Linguistic Data Consortium [16], which utilizes 1935 senones, requiring over 2.5 million floating-point operations to calculate scores for every senone. For any practical system these calculations become the critical path and need to be done as efficiently as possible. By performing an in-depth analysis of these calculations, it was found that the computationally intensive floating-point Gaussian probability calculations could be replaced with fixed-point calculations while only introducing errors on the order of 10^{-4}. The ability to use fixed-point instead of floating-piont calculations allowed for the implementation of a pipelined acoustic modeling core running at over 100 MHz post place-and-route on a Virtex-4 SX35-10. Figure 5 illustrates the main components of the AM pipe.

Each of the stages in the pipeline sends a "go" signal to the following stage along with any data to be processed, allowing for the system to be stalled anywhere in the pipe without breaking. The first three stages also receive data from a status bus regarding the particular nature of the calculation being performed (i.e., is this the first, middle, or last element of a summation), which removes the need for any local FSM to control the pipeline.

3.1. *Gaussian distance pipe*

The Gaussian distance pipe is the heart of AM block and is responsible for calculating (1)–(3) for each senone in the database. This pipe must execute (1) over 620 000 times for each new frame of data and therefore must have the highest throughput of any component in the system. To accommodate this requirement while still trying to minimize the resources consumed by pipeline, the inputs to crucial arithmetic operations are multiplexed, allowing the inputs to the operation to be selected based on the bits of the status bus. The bits of the status bus, the *calc-bits*, provide information as to which element of the summation is being processed so that the output of the given stage can be routed properly to the next stage. Figure 6 shows a data-flow graph (DFG) for the order of operations inside the Gaussian distance pipe.

In order to help with low-power applications, the Gaussian distance pipe has a "pipe freeze" feature included, which is not shown in the DFG. If the last bit of the calculation is seen at the end of the pipe before a new first bit to be calculated has arrived, the pipe will completely shut down

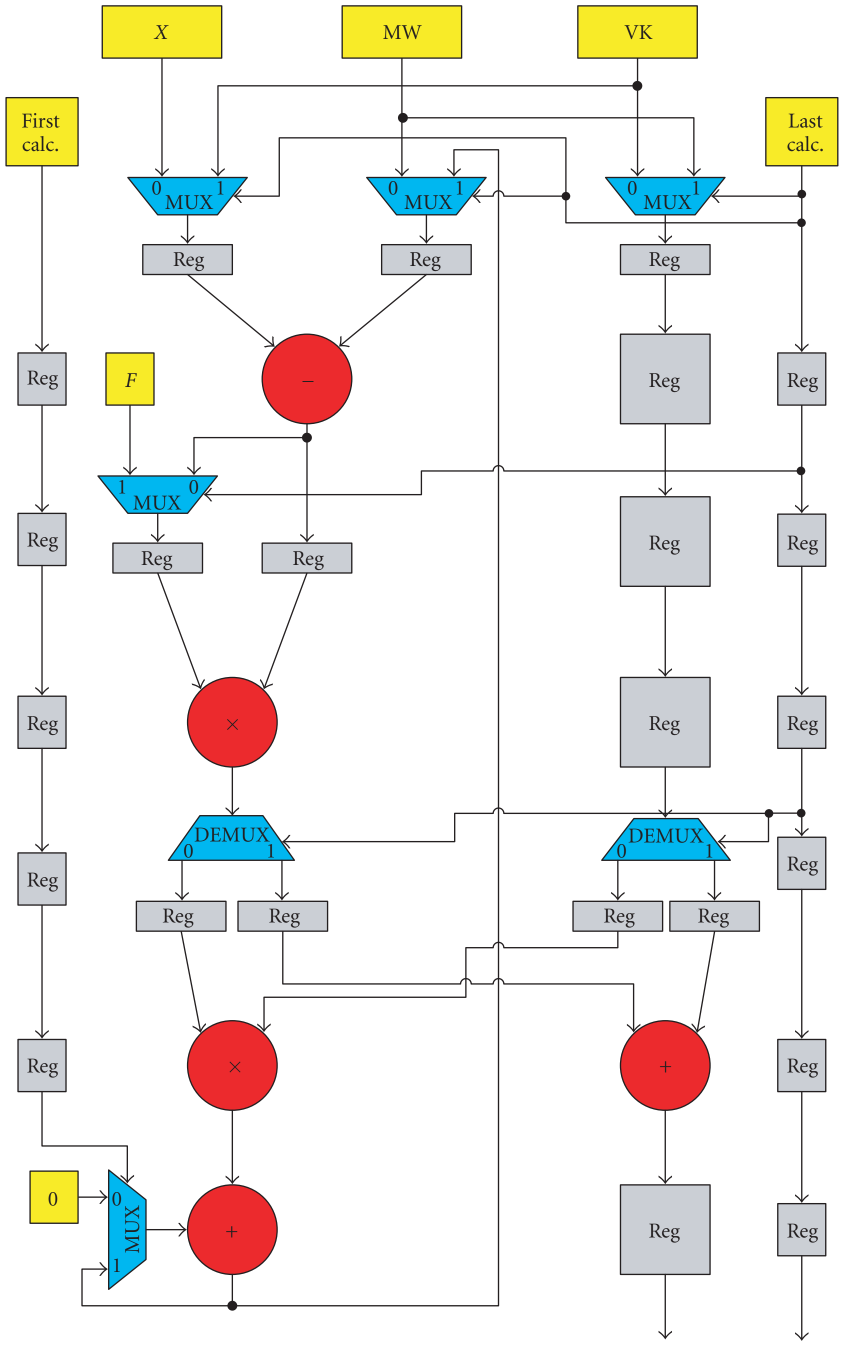


Figure 6: Data-flow graph for Gaussian distance pipe.

and wait for the presence of a new data. Internal to the pipe, each stage passes a valid bit to the successive stage that serves as a local stall, which will freeze the pipe until the values of the predecessor stage have become valid again.

Examining (2)–(4) reveals that to calculate a single component based on a d-dimensional Gaussian PDF actually requires $d + 1$ cycles, since the result of the summation across the d-dimensions must be subtracted from a constant and then scaled. As shown in Figure 6, the data necessary for the subtraction and scaling (K & W) can be interleaved into the data for the means and variances (M & V), leading to the need to read $d+1$ values from the ROM for each component in the system. This creates a constraint for feeding data into the pipe such that once the $d+1$ values have been read in, the system must wait for 1 clock cycle before feeding the data for the next component of the pipe. This necessity comes from the need to wait for the output of the final addition shown at the bottom of Figure 6. At the beginning of clock cycle $d + 1$, the K & W values are input into the pipe, but these values cannot be used until the summation of $DIST\ (X)$ is complete. This does not occur until clock cycle $d + 2$, resulting in the need to hold the input values to the pipe for one extra cycle.

Figure 6 further indicates that it takes seven clock cycles to traverse from one end of the pipe to the next. However, the next stage of the design, the log-add lookup table (LUT), described in Section 3.2, takes ten cycles to traverse. Therefore we must add three extra cycles to the Gaussian distance pipe to keep both stages in sync. To ensure that the additional cycles are not detrimental to the system, a series of experiments were conducted examining the effects of additional pipeline stages on the achieved $f_{\max}$ of the system. The results of these experiments, as well as the synthesis and post place-and-route results for this block are summarized in Section 4.

3.2. Log-add lookup

After completing the scoring for one component, that component is sent to the log-add LUT for evaluation of (4)–(6). This block is responsible for accumulating the partial senone scores and outputting them when the summation is complete. Equations (7)–(10) show the calculations necessary to perform the log-add of two components $P_{1,1}$ and $P_{1,2}$,

$$D = |P_{1,1} - P_{1,2}|, \tag{7}$$

$$R = P_{1,1} \longrightarrow \text{if} : P_{1,1} > P_{1,2}, \tag{8}$$

$$\text{else: } R = P_{1,2},$$

$$\begin{gathered}\text{let: } \Psi = 1.0003, \\ \text{let: } f = \frac{1}{\log(\psi)},\end{gathered} \tag{9}$$

$$\text{RES} = R + 0.5 + f^*\left(\log\left(1 + \psi^{-D}\right)\right). \tag{10}$$

Due to the complexity of (10), it has been replaced by a LUT, where D serves as the address into the table. By using this table, (10) can be simplified to the result seen in (11),

$$\text{RES} = R + \text{LUT}(D). \tag{11}$$

While the use of a lookup to perform the bulk of the computation is a more efficient means of obtaining the desired result, it creates the need for a table with greater than 20 000 entries. In an effort to maximize the speed of the LUT, it was divided into smaller blocks and the process was pipelined over 2 clock cycles. The address is demultiplexed in the first cycle and the data is fetched and multiplexed onto the output bus during the second.

Equations (7)-(8) illustrate the operations necessary to find the address to this LUT. We chose to implement these operations as a three stage pipeline. The first stage of operation performs a subtraction of the two raw inputs and strips the sign bit from the output. In the second cycle the sign bit is used as a select signal to a series of multiplexers that assign the larger of the two inputs to the first input of the subtraction and the smaller to the second input of the summation. The third cycle of the pipe registers the larger value for use after the lookup and simultaneously subtracts the two values to obtain the address for the table. Similarly to the Gaussian distance pipe, the log-add LUT also has a pipe-freeze function built in. Figure 7 shows a detailed data-flow graph of the operations being performed inside the log-add lookup.

As mentioned in Section 3.1, the entire log-add calculation takes a minimum of 10 clock cycles to process a single input and return the partial summation for use by the next input. When this block is combined with the Gaussian distance pipe to form the main pipeline structure for the AM block the result is a 20 stage pipeline capable of operating at over 140 MHz, and requiring no local FSM for managing the traffic through the pipe, or possible stalls within the pipe.

3.3. Find Max/normalizer

Once a senone has been calculated, it must first pass through the find Max block before being written to the senone RAM. This block is a 2-cycle pipeline that compares the incoming data to the current best score and overwrites the current best when the incoming data is larger. Once the larger of the two values has been determined, the raw senone is output to the senone RAM. This is accompanied by a registered write signal ordinarily supplied by the log-add LUT. A data-flow graph for the find Max block is shown in Figure 8.

As mentioned in Section 3.2, the find Max unit only needs to operate once every 10 cycles, or whenever a new senone is available, therefore the values being fed to the compare are only updated when the senone valid bit is high. Aside from this local stall, the find Max unit has a similar pipe freeze function to conserve power.

When the last raw senone is put into the senone RAM, the "*MAX done*" signal in Figure 8 will be set high, signaling to the normalizer block that it can begin. During the process of normalization the raw senones are read sequentially out of the senone RAM and subtracted from the value seen at the "*Best Score*" output of the find Max block. The normalizer block consists of a simple 4-stage pipeline that first registers the input, then reads from the RAM, performs the normalization, and finally writes the value back to the RAM. The normalizer block also has pipe-freeze and local stall capabilities.

3.4. Composite senone calculation

In the RM1 speech corpus there are two different types of senones. The first type is "normal" or "base" senones, which are calculated via the processes described in Sections 3–3.3. The second type is a subset of the normal senones called composite senones. Composite senones are used to represent more difficult or easily confusable sounds, as well as nonverbal anomalies such as silence or coughing. Each composite senone is pointer to a group of normal senones, and for a given frame the composite senone takes the value of the best scoring normal senone in its group.

In terms of computation this equates to the evaluation of a series of short linked lists, where the elements of the list

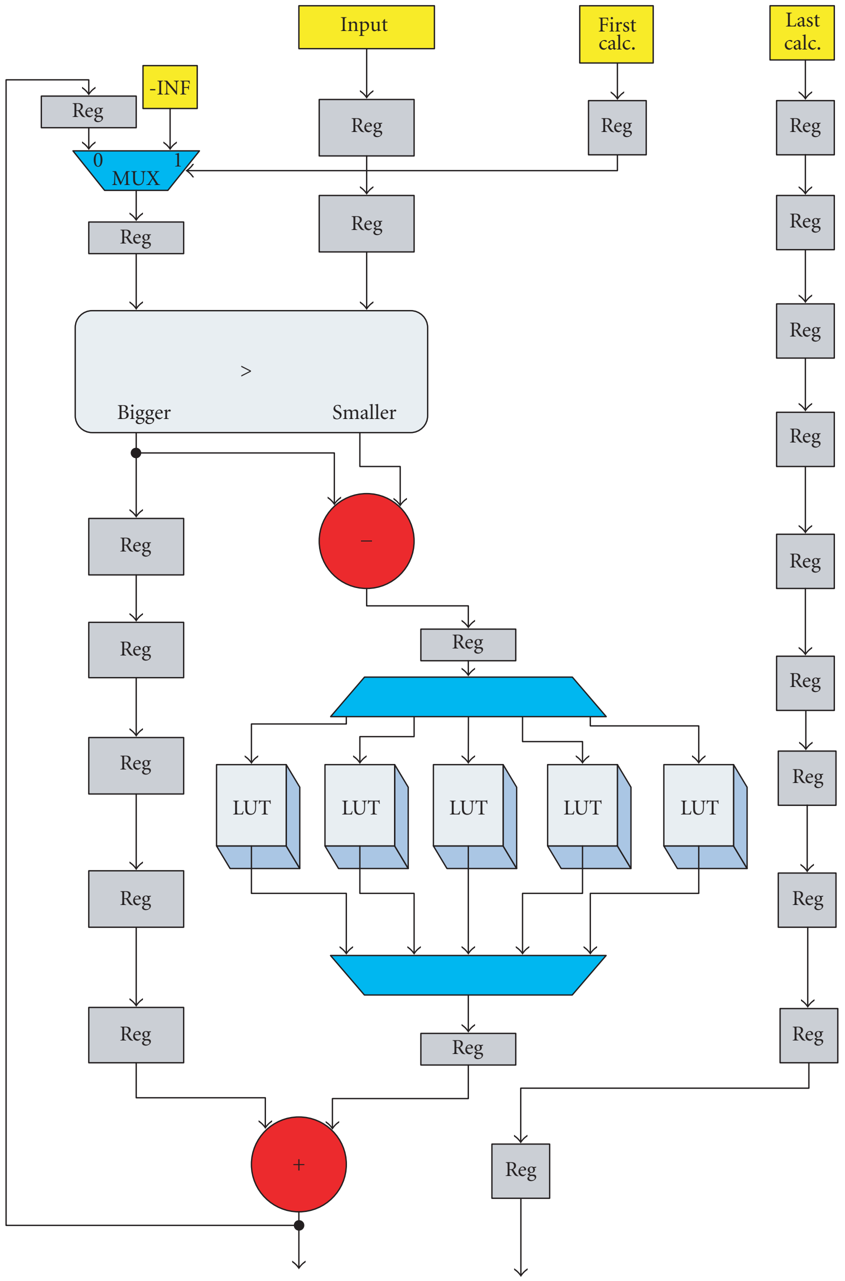


Figure 7: Data-flow graph for log-add LUT.

must be compared to find the greatest value. Once this greatest value is found it is written to a unique location in the senone RAM at some address above the address of the last normal senone. By writing this entry into its own location in the senone RAM instead of creating a pointer to its original location, the phoneme evaluation block is able to treat all senones equally, thus simplifying the control for that portion of the design.

The composite calculation works through the use of two separate internal ROMs to store the information needed for processing the linked-lists. The first ROM (*COUNT ROM*) contains the same number of entries as the number of composite senones in the system, and holds information about the number of elements in each composite's linked list. When a count is obtained from this ROM, it is added to a base address and used to address a second ROM (*ADDR ROM*) that contains the specific address in the senone RAM, where the normal senone resides.

Once the normal senone has been obtained from the senone RAM, it is passed through a short pipeline similar to

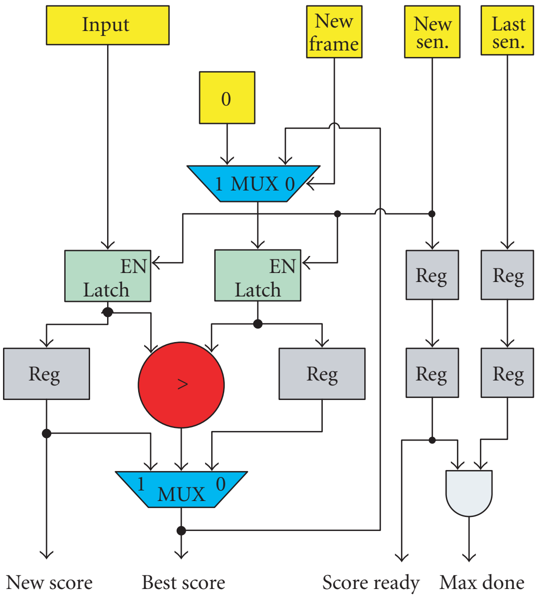


Figure 8: Data-flow graph for find Max unit.

the find MAX block except that only the best score is written back to the senone RAM. The count is then decremented and the process repeated until the count equals zero. At this point the next element of the count ROM is read and the process is repeated for the next composite senone. Once all elements of the count ROM have been read and processed, the block will assert a done signal indicating that all the senone scores for a given frame have been calculated. A DFG for the composite senone calculation is shown in Figure 9.

Like the other blocks of the AM calculation, the composite senone calculation has the built-in ability for locally stalling during execution and freezing completely when no new data is present at the input. This feature is more significant because composite senone calculations can only be performed after all of the normal senones have been completely processed. This results in a significant portion of the runtime where this block can be completely shut down leading to notable power savings. Specifically, it takes approximately 650 000 clock cycles to calculate all of the normal senones, during which the composite senone calculation block is active for only 2200 cycles.

In order to minimize the data access latency of later stages in the design, the senone RAM is replicated three times. When processing AM, the address and data lines of each of the RAMs are tied together so that one write command from the pipeline will place the output value in each of the RAMs during the same clock cycle. When the control of these RAMs is traded off to the phoneme evaluator (PE), the address lines are decoupled and driven independently by the three senone ID outputs from the PE. While this design choice does create a nominal area increase, the 3x improvement in latency is critical for achieving real-time performance.

4. PHONEME EVALUATOR

During phoneme evaluation, the senone scores calculated in the AM are used as state probabilities within a set of HMMs. Each HMM in the database represents one context-dependent phone or phoneme. In most English speech corpuses, a set of 40–50 base phones is used to represent the phonetic units of speech. These base phones are then used to create context-dependent phones called mono-, bi-, or triphones based on the number of neighbors that have influence on the original base phone. In order to stay close to the SPHINX 3 system, we chose to use a triphone set from the RM1 speech corpus represented by 3-state Bakis-topology HMMs. Figure 10 shows an example Bakis HMM with all states and transitions labeled for later discussion.

The state shown at the end of the HMM represents a null state called the exit state. While this exit state has no probability associated with it, it does have a probability for entering it. It is this probability that defines the cost of transitioning from one HMM to another. One of the main advantages of HMMs for speech recognition is the ability to model time-varying phenomena. Since each state has a self transition as well as a forward transition, it is possible to remain inside an HMM for a very large amount of time or conversely, to exit an HMM in as little as four cycles, visiting each state only once. To illustrate this principle, Figure 11 maps a hypothetical path through an HMM on a two-dimensional trellis.

By orienting the HMM along the Y-axis and placing time on the X-axis, Figure 11 shows all possible paths through an HMM with the hypothetical best path shown as the darkened line through the trellis. In our HMM decoder we chose to use the Viterbi algorithm to help minimize the amount of data needed to be recorded during calculation. The Viterbi algorithm states that if, at any point in the trellis, two paths converge, only the best path need be kept and the other discarded. This optimization also is widely used in speech recognition systems, including SPHINX 3 [18].

For each new set of senones, all possible states of an active HMM must be evaluated to determine the actual probability of the HMM for the given inputs. The operations necessary to calculate these values are described in (12)–(15),

$$\begin{array}{l}\left.\begin{array}{l}\mathrm{H}_3(t-1)+\mathrm{T}_{22}\rightarrow\\ \mathrm{H}_2(t-1)\mathrm{T}_{12}\rightarrow\end{array}\right.\textcircled{$>$}+S_2(t)=\mathrm{H}_3(t),\\ \\ \left.\begin{array}{l}\mathrm{H}_2(t-1)+\mathrm{T}_{11}\rightarrow\\ \mathrm{H}_1(t-1)+\mathrm{T}_{01}\rightarrow\end{array}\right.\textcircled{$>$}+S_1(t)=\mathrm{H}_2(t),\end{array}\tag{12}$$

$$\left.\begin{array}{l}\mathrm{H}_1(t-1)+\mathrm{T}_{00}\rightarrow\\ \mathrm{H}_0\rightarrow\end{array}\right.\textcircled{$>$}+S_0(t)=\mathrm{H}_1(t),\tag{13}$$

$$\mathrm{H}_{\mathrm{BEST}}(t)=\mathrm{MAX}\left\{\mathrm{H}_1(t),\mathrm{H}_2(t),\mathrm{H}_3(t)\right\}\tag{14}$$

$$\mathrm{H}_{\mathrm{EXIT}}(t)=\mathrm{H}_2+\mathrm{T}_{2e}.\tag{15}$$

Equations (12)-(13) show that the probability of an HMM being in a given state at a particular time is dependent not only on that state's previous score and associated transition penalties, but also on the current score of its associated senone. This relationship helps to enhance the accuracy of the model when detecting time-varying input patterns.

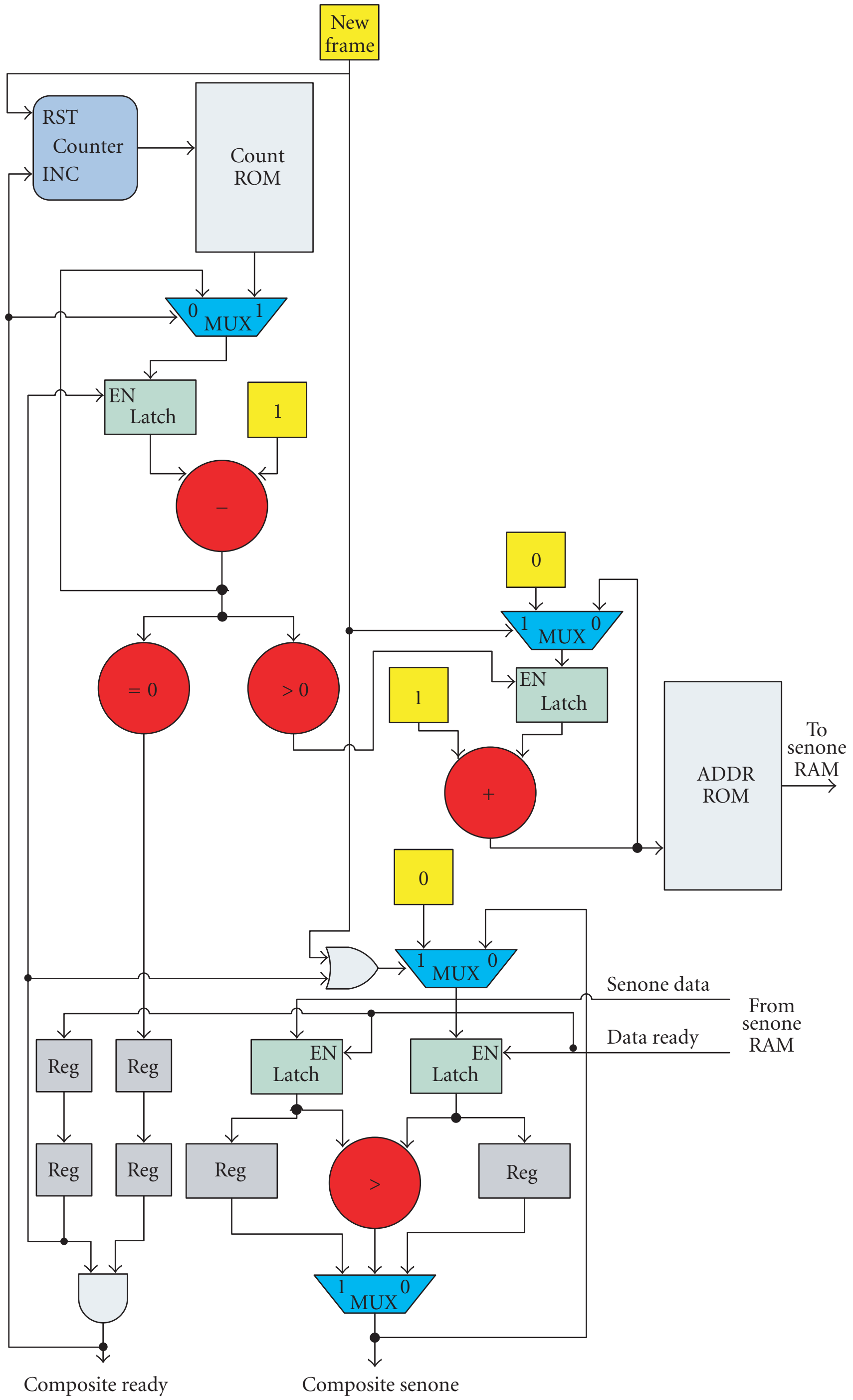


Figure 9: Data-flow graph for composite senone calculation.

Looking specifically at (13) it can be seen that the H_0 input is not time-dependent. While H_0 is not completly constant, it does not change with every new time tick, and therefore is not considered strictly time-dependent. The specific reasons for this functionality relate to the way HMMs are activated and deactivated in the system and are described in more detail in Section 5. However, it should be noted that this value only changes on the transition from inactive to active. Equations (12)-(13) show the insertion of the Viterbi algorithm in that only the best of the possible transitions will be held in the value use in the next time.

It is also relevant to note that, while we may have a very large number of bi- or tri-phones in the database, we only have as many unique sets of transition matrices as we have

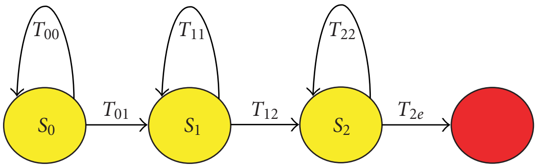


Figure 10: Sample HMM topology.

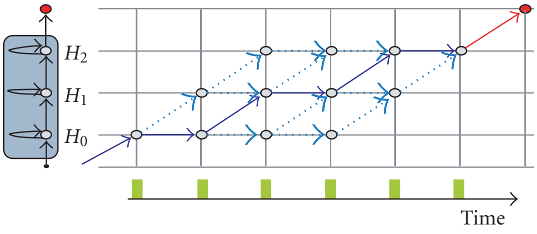


Figure 11: Sample HMM trellis.

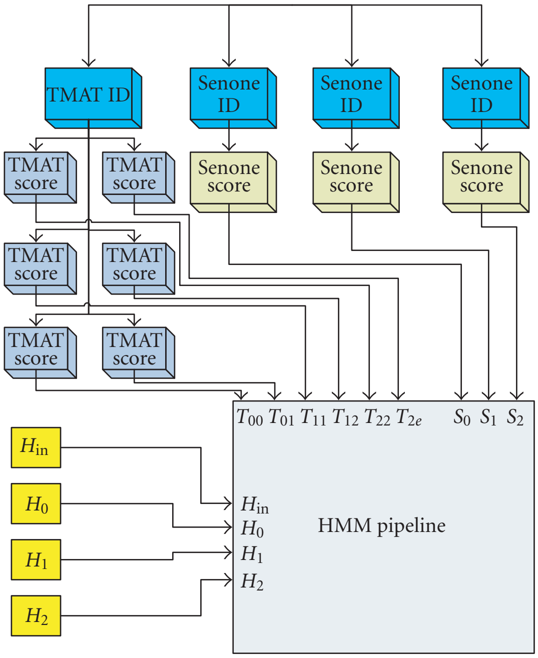


Figure 12: Data hierarchy for HMM calculation.

base phones. When a new HMM is to be processed, its ID is put into a decoder that outputs the ID of the transition matrix to be used for the evaluation. This transition score ID, or TMAT ID, is then used to address the ROMs that contain the actual scores associated with the arcs of Figure 10. For each evaluation of an HMM the best of all state scores as well as the HMM exit score must be calculated to facilitate the pruning of the HMM in the next stage of the PE.

Once the H_{BEST} and H_{EXIT} values have been found for all the active HMMs, a beam pruning algorithm is applied to the set to help mitigate the amount of active data in the system. During this process a constant offset or beam is added to the best H_{BEST} and H_{EXIT} values. Only values with scores above this beam remain active for the next stage. As the HMMs are being pruned, a token with the HMMs unique ID is written to a status FIFO based on the result of pruning. During the word modeling portion of the SR process, these FIFOs will be read and the data in the shared RAM will be processed accordingly.

When implementing the PE in hardware, the majority of the logic necessary resides in the large amount of constant data that must be stored and retrieved in order to process an HMM. Figure 12 illustrates the data structure that must be traversed to acquire all of the necessary data to process a single HMM.

In the structure shown in Figure 12 the HMM ID input is used to address 4 separate LUTs, one for the transition score ROMs, and one for each of the senones needed for the HMM. As mentioned previously, the TMAT ID ROM serves as a decoder to map one of a large set of HMM IDs to one of the relatively small set of TMAT IDs. This single TMAT ID is then used to address six TMAT score ROMs in parallel in order to decrease the latency of the data access. The other three LUTs receive the same HMM ID, but are used to decode the appropriate senone IDs needed for the HMM. Again in an effort to increase parallelism and decrease latency, these senone IDs are found in parallel and output from the block back to the shared senone RAM described in Section 3. Once the TMAT scores, current senone scores, and previous state scores have been retrieved, the actual processing of the HMM begins. As previously stated, the remainder of the calculation can be implemented as a high throughput pipeline and is described in detail in the following sections.

4.1. HMM control logic

While developing our architecture, it was necessary to consider the fact that unlike AM, the amount of work that needs to be done at any one time is variable and therefore some control must be included to monitor the amount of active data in the system. As illustrated in Figures 3 and 4 the data to be processed by the PE is most efficiently managed by a series of FIFOs containing lists of active HMM IDs. Specifically, data entering PE is provided via either the new phoneme active list (nPAL) FIFO, and data exiting PE is written to either the exit (VALID) FIFO, the inactive (DEAD) FIFO, or the phoneme active list (PAL) FIFO. Figure 13 illustrates the relationships between these FIFOs.

The first observation made when looking at Figure 13 is that the PAL FIFO is actually completely internal to the PE block and can be loaded by either the HMM pipeline or the pruner. In order for this to work properly a special end of phase (EOP) token was created to serve as a place marker in the FIFO. To create the EOP token the PAL FIFO was designed to be 1 bit wider than the other FIFOs so that the extra

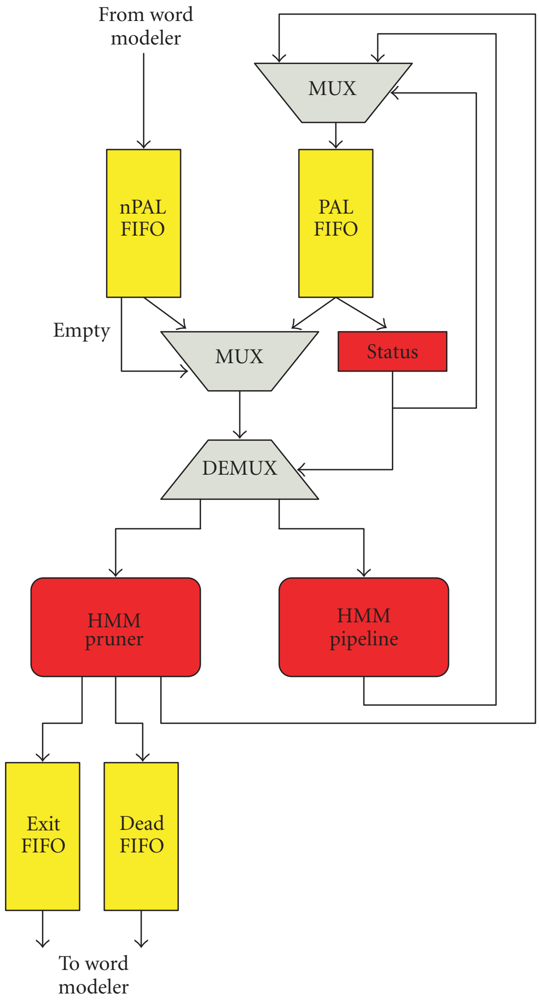


Figure 13: Control for the phoneme evaluator.

bit could be used to hold information about the EOP token. All standard tokens in the queue are appended with a "0" for this extra bit, while the EOP token is appended with a "1," making it very easy to detect the presence of the EOP token. At the beginning of each new frame PE will start pulling tokens from the nPAL FIFO until the FIFO is completely empty. When the FIFO is emptied, or in the case there was nothing there to start, the PAL FIFO is then read until and EOP token is detected by the STATUS block. When the EOP token is seen it is then known that all HMMs requiring processing have been completed and pruning may commence.

At the beginning of pruning the EOP token is written back to the PAL FIFO. Pruning continues until the EOP token is popped back out of the FIFO. Upon this second observation, all data has been processed for that frame and the word modeler may begin processing the tokens in the DEAD and EXIT FIFOs.

4.2. HMM pipeline

The execution of (12)–(15) constitutes the majority of the calculations necessary to perform phoneme evaluation and therefore a significant amount of time was put into examining the optimal way to perform these operations. After establishing the control for this pipeline the calculations were examined and it was found that to find all H values for a given HMM a simple ADD-COMPARE-ADD-COMPARE pipeline can be constructed as shown in the DFG in Figure 14.

The DFG in Figure 14 highlights the regularity of the structure for the pipeline and leads directly to a high-throughput low-latency design for calculation of the HMM scores in the system. Further, the complexity of the pipeline is actually quite small and requires a noticeably smaller amount of logic than even the ROMs required to drive it. As each of the active HMMs are evaluated, the five output values are written to a shared RAM called the phone-pointer RAM (PH RAM) and the HMM ID token is written into the pruner queue. Results on synthesis for this block are summarized in Section 6 of this document.

4.3. HMM pruner

After having calculated all HMM scores for a given frame the scores are then read back out of the RAM and compared to the beams. In our system we use two different beams to prune the HMMs based on both their exiting score and their best score. If an HMM has a valid exit score it will be passed to the word modeler as well as remain in the active queue. If the HMM score is not above the exit beam, it will be checked against a second beam to see if the HMM should remain in the active queue. This two-step approach helps to minimize the number of HMMs mistakenly pruned from the system and significantly increases the recognition accuracy of the system. It also helps to maintain a time-varying system, in that an HMM can exit and remain active so that in successive frames the HMM could exit again, but with a higher probability.

A third beam is also calculated by the pruner and is passed forward to the word modeler for later use. This beam is calculated based off of the exit score for any active HMM in the cue that represents the end of a word in a dictionary. Just as transitioning from one HMM to another incurs a penalty so does transitioning from one word to another, and the word beam helps to prune out unlikely sequences of words. While the HMM pruner does not actually process the data in the PH RAM based on the result of pruning, it does establish the work order for the word modeler and helps to greatly simplify that stage of processing.

5. WORD MODELER

Word modeling can be broken down into two major steps: resetting of newly inactive tokens and updating of currently active tokens. Given that the functionality of these two components is distinctly different, two separate components were designed, one for each task. The token deactivator reads data from the DEAD FIFO and resets the scores in the PH RAM. Simultaneously, the token activator reads from the EXIT FIFO and processes the word tree to determine which new tokens need to be placed in the nPAL FIFO. The creation of

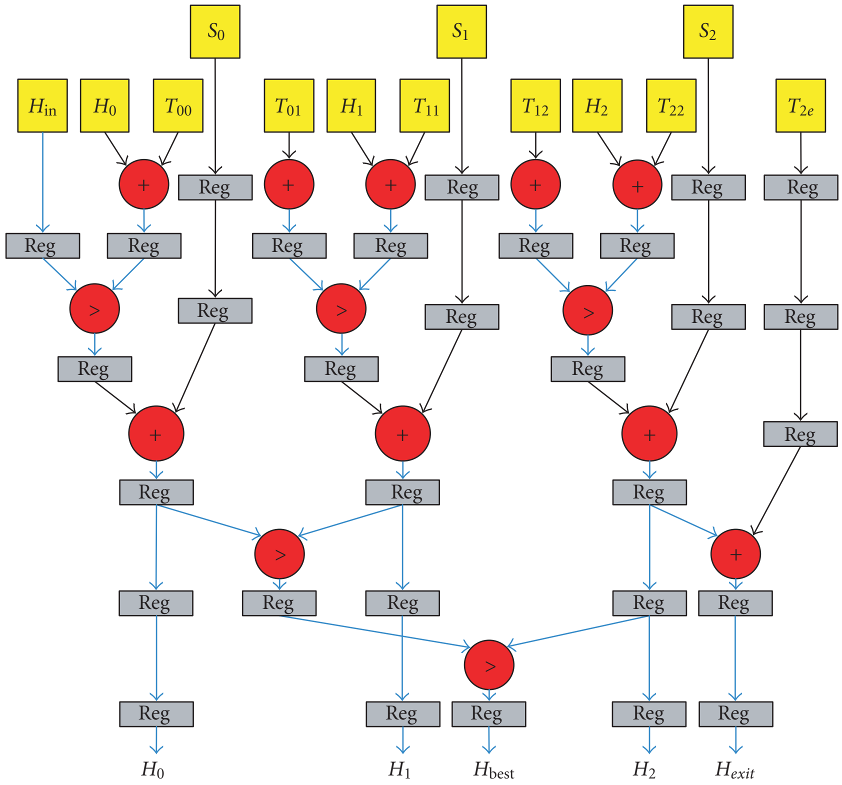


Figure 14: Data-flow graph for HMM pipeline.

two separate blocks also minimizes the amount of logic active at any one point in time leading to power savings crucial to ensuring sustainability on a mobile platform. The operations of these blocks are described in detail in Sections 5.1 and 5.2, respectively.

5.1. Token deactivator

After the PE block has pruned the active HMMs and placed the appropriate tokens in the DEAD FIFO the word modeler can begin the task of resetting the PH RAM entries corresponding to these tokens. This process is simplified by the fact that all PH RAM values need to be reset to the exact same value. This means that to deactivate a token the token must be popped from the DEAD FIFO and used to address the PH RAM. When this address is applied to the RAM the reset constant is then written to the RAM and the process is repeated until the FIFO is empty. This process can be performed in a simple-two stage, POP-WRITE pipeline, and is capable of running at close to the f_{MAX} of the target device.

5.2. Token activator

The token activator portion of the word modeler is noticeably more complicated than deactivator portion. When an HMM is found to have a valid exit score, the word modeler must determine the location of that HMM in the word tree and which HMMs are tied to its exit state. As shown in Figure 1, a word tree can have a large number of branches stemming from one root. Mapping these types of structures into hardware is not obvious. Another unique problem in token activation is that while a given HMM may be used multiple times in multiple different word trees such as the *"CH"* sound at the end of *pouch* and *couch*, these two sounds must be represented by completely unique events. This means that while a given dictionary may not need all possible phonemes in a language, it will most likely need multiple instances of some of the phones.

Thus, it was necessary to determine a way of indexing specific nodes in the search space such that their information could remain in a unique location in the PH RAM. To do this, the entire word search space was mapped and each of the nodes was given a unique ID. An example of this process is shown in Figure 15.

Based on this mapping scheme, even though nodes 12–14 in Figure 15 all relate to the *AE* phoneme, they all have unique IDs and will be treated separately in search algorithm. The mappings determined in the process relate directly to the HMM IDs stored in the tokens passed between the PE and WM blocks, and define the core of the token passing algorithm as implemented in our system.

Having established our mapping scheme it was necessary to implement a tree structure in hardware. Unlike the previous section of the design, this portion is less arithmetically

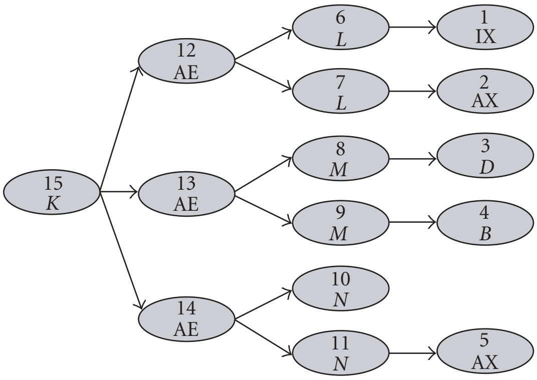


Figure 15: Sample search space node mapping.

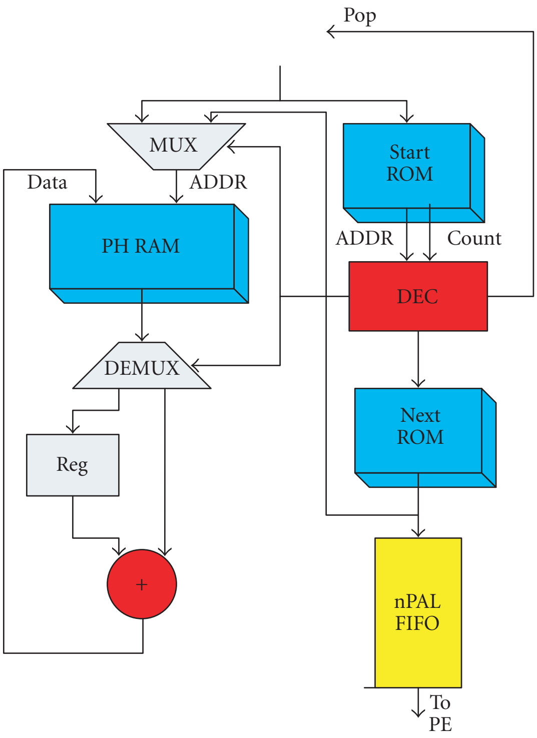


Figure 16: Data-flow graph for token activator.

intensive and involves searching instead of computational overheads. One of our immediate observations when looking at the data structures was that each node in the search space can be thought of as a short linked list. Evaluating linked lists in hardware is a well-defined process, so to create our tree we created a linked list of linked lists.

During each evaluation a token must be read from the EXIT FIFO and its linked list retrieved. Further the H_{EXIT} score from the exiting HMM must be added to a word penalty and propagated to the H_0 score of the HMMs in the linked list. Since the word penalty will be different for each HMM in the linked list, it becomes necessary to process the HMMs one at a time until the end of the list is reached. To efficiently keep track of the linked list, two ROMs were designated: the START ROM and the NEXT ROM. The START ROM is directly addressed by the token in the EXIT FIFO and contains both a starting address in the NEXT ROM for the linked list and a count of how many values are in the list. The NEXT ROM holds all of the HMM IDs necessary to process the linked lists. Figure 16 shows the DFG for the token activator.

When a token is read from the EXIT FIFO a multiplex control bit is reset to determine which token is in control of the PH RAM address. While the count for the linked list is nonzero, the bit will remain set but once the final decrement has been completed the bit control bit will switch to allow a new exiting HMM to be read from the PH RAM. This process is repeated for each element in the EXIT FIFO until the FIFO is emptied at which time the system goes idle and awaits the next frame of input data.

6. SYNTHESIS AND PLACE-AND-ROUTE

While performing the synthesis and place-and-route operations on our hardware architecture implemented in VHDL, two distinctly different approaches were taken in an effort to observe the effect of design tools on overall system design. The extremely computationally heavy AM block was written using VDHL syntax specific to the Synplify synthesis engine in an effort to focus the bulk of the work load on the tools as opposed to the designers. This approach allows for the generation greatly simplified VHDL which in turn makes the processes of debugging and optimization significantly easier. Additionally, the Synplify synthesis tool provides built-in features to perform pipelining and retiming of the target VHDL.

As a second design strategy we implemented the PE and the WM blocks in traditional VHDL and synthesized using precision synthesis from Mentor Graphics. Precision synthesis tends to leave more optimizations up to the designers, and knowing that the implementation of the PE and the WM are significantly more intuitive than the implementation for AM, we chose to leave the bulk of the design optimization to ourselves. To compare the overall quality of the design as well as the time for development, limited portions of PE were also written for Synplify so a one to one comparison of results could be obtained.

6.1. Acoustic modeling results

To examine the effects of pipelining and retiming in Synplify, a series of experiments was conducted on the Gaussian distance pipe to find the optimal number of pipeline stages. To do this, extra registers were put into the design and the pipelining and retiming options were enabled in the synthesis tool. When the synthesis was executed, the tool was able to move these registers to what it determined were the optimal locations in the design, minimizing the amount of analysis done by the designer. Our primary target in these

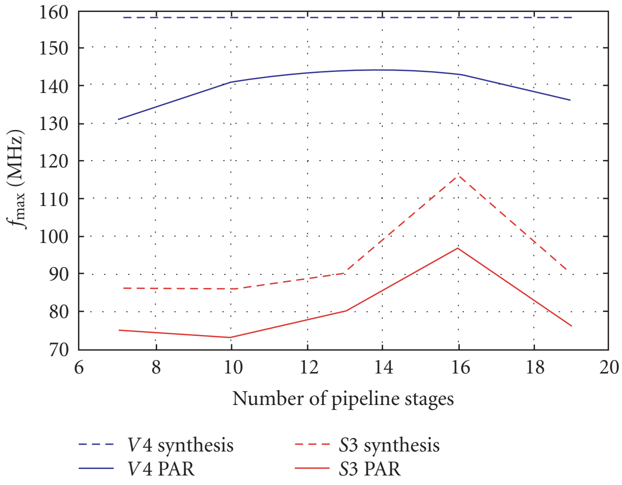


FIGURE 17: Analysis of $f_{\max}$ versus pipeline stages of the gaussian distance pipe for virtex-4 SX and spartan-3 FPGAs.

experiments was the Xilinx Virtex-4 SX35 FPGA due to its high performance and large number of embedded DSP (DSP48) and RAM (BRAM) cells. For sake of comparison we also targeted a smaller device, the Xilinx Spartan-3, a 90 nm FPGA with embedded 18 × 18 multipliers. The graph in Figure 17 shows the results of these experiments for a pipeline between 7 and 19 stages deep, with the dotted lines representing the projected $f_{\max}$ of the system from the synthesis engine and the solid lines representing the post place-and-route $f_{\max}$.

While the $f_{\max}$ obtained for the Virtex-4 device is noticeably higher than the $f_{\max}$ of the Spartan-3 devices, it is also observed that increasing the number of pipeline stages improves the speed of the Spartan-3 device more significantly than the speed of the Virtex-4 device. It is also observed that for both devices there are an optimum number of stages beyond which the performance of the device actually degrades due to the increased amount of area consumed by the design.

Another interesting result of these experiments was that regardless of the number of pipeline stages the projected synthesis speed for the Virtex-4 did not change. This implies that even when the pipeline is configured with the minimum number of allowable stages, the results of the pipelining and retiming processes are the same. The post place-and-route timing results for the Virtex-4, however, do change with the number of pipeline stages implemented. Since we know we are utilizing the embedded DSP slices on the chip and we can trace the critical path of the circuit we can conclude that the physical distance between two individual DSP cells is great enough that adding extra registers along the path will in fact increase the speed of the design. Figure 16 further shows that when targeting the Virtex-4, a 10-stage pipe will provide an acceptable operating frequency for the system with only minor improvements being gained with each additional pipe stage beyond 10. This is a promising result because we know that the depth of the Log-Add LUT is 10 cycles as well, allowing us to match the depths of the two pipelines without having to sacrifice a considerable amount of speed.

TABLE 3: Summary of synthesis and place-and-route results for Virtex-4 SX35.

Component	Synthesis (MHz)	PAR (MHz)	Area
Gaussian dist. pipe	157	145	6 DSP48s 423 slices
Log-add LUT	164	150	13 BRAMs 307 slices
Find max	181	160	90 slices
Normalizer	197	172	155 slices
Composite senone calc.	197	140	2 BRAMs 147 slices
AM block (total)	164	125	6 DSP48s, 30 BRAMs 1527 slices

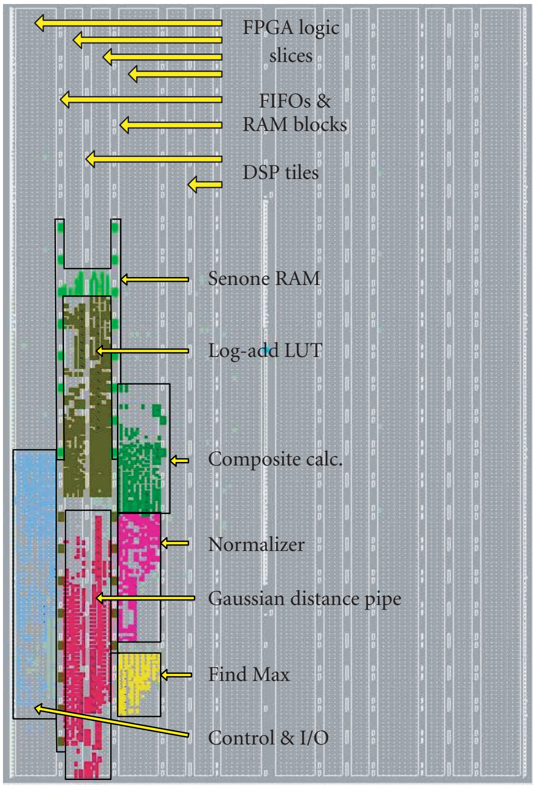


FIGURE 18: Floor plan for AM pipeline.

In addition to the experiment described above all individual components of the AM block were synthesized and routed on the chip to fully characterize their performance. Table 3 summarizes the results of these tests and makes note of any special cells used by each stage of the design. Figure 18 shows the FPGA layout for the AM block.

Table 4: Power analysis for multiple target FPGA devices.

Power (mW)	Virtex II Pro XC2VP100	Virtex 4 XCE4VSX35	Spartan 3 XC3S1500
Dynamic	163.46	35.46	41.75
Static	571.88	395.50	178.00
Total	735.34	430.97	219.76

Table 5: Summary of post-place-and-route results for the phoneme evaluator.

Component	f_{MAX} (MHz)	Slices	Flip-flops	LUTs	MISC
HMM pipeline	189	1475	2393	593	24 BRAMs
Pruner	115	48	24	77	—
PE core	115	1713	2645	861	24 BRAMs
PE control	213	8	14	13	—
PE block (total)	117	1941	2983	1050	25 BRAMs

Further, since AM consumes over 90% of the run-time, it was relevant to analyze the power consumption of this block to get a feel for the overall consumption of the device. To do this, post-place-and-route simulations were performed in ModelSim using data generated in Xilinx Xpower and the results of our experiments are shown in Table 4 for Virtex-4, Virtex-2pro, and Spartan-3 devices all running at $f_{\max}$ for the specific device. The stimulus for these experiments was based on randomly generated inputs to the system.

6.2. Phoneme evaluator results

Given that the design for the PE was noticeably less complex than the design for the AM, we chose to implement all optimizations by hand and derive our own custom models for all necessary components as opposed to allowing the tools to do this. We used FPGAdvantage GUI to derive the code and precision synthesis to do our placement analysis. Our post-place-and-route results are summarized in Table 5.

6.3. Word modeler results

Observing that word modeling took less than 5% of the overall execution effort, little time was spent analyzing the synthesis results for this block. As with the PE block we opted for custom derivation of VHDL and synthesis via precision synthesis. The results of the place-and-route operations are summarized in Table 6.

Table 6: Summary of the word modeler synthesis results.

Component	f_{MAX} (MHz)	Slices	Flip- flops	LUTs	MISC
Token deactivate	319	51	48	88	—
Token activate	145	174	66	320	2 BRAMs
WM block (total)	142	357	131	638	2 BRAMs

6.4. Hardware development summary

The hardware development presented in this work presents a novel processing architecture capable of executing the CSR algorithm at over 100 MHz. As discussed in Section 3, 100 MHz has been determined to be the minimum operating speed for a device to process speech in real time. This requirement comes from the known input frame rate of 10 milliseconds and the proposed maximum cycle count of one million. Preliminary results on the entire operational system have shown a device running at 105 MHz on a Virtex-4 SX35 ff668-10 and requiring less than 800 000 cycles to complete all necessary operations. These results clearly show a system able to run at sufficient speeds for real-time speech recognition as well as maintain an average of 20% down-time during which the engine is inactive.

Aside from being able to recognize human speech in real time, special attention was paid to ensure that the throughput of the design was maximized via the creation of custom pipelines for each stage of the algorithm. The first major portion of the algorithm, acoustic modeling, has been shown to be the most computational intensive part of the problem and significant effort was taken to design this block as efficiently as possible. The result is a custom hardware pipeline capable of operating at 125 MHz post-place-and-route on a Virtex-4 SX35. This pipeline is completely data-driven and involves no internal state machines to guide the process. By giving the design this flexibility the complexity of the inputs can be varied without needing to reconfigure the design.

During the design of the phoneme evaluation stage the large data access problem encountered was effectively reduced through the use of multiple small parallel ROMs and pointer arrays. When processing an HMM a large amount of data must first be retrieved to perform the calculations. While the need for moving such large quantities of data within the design adversely effects the performance of the pipeline, speeds of 111 MHz after post-place-and-route are still possible, with the core of the processing unit able to operate as fast as 140 MHz post-place-and-route.

During the final portion of the design, word modeling, a tree-search algorithm was designed in hardware. The hardware was designed as a large linked list evaluation unit capable of propagating information throughout the tree while also deactivating nodes in the tree and connecting multiple

Table 7: Summary of hardware performance results.

Component	f_{MAX} synthesis/PAR (MHz)	Area	Cycle count
Gaussian distance pipeline	157/145	6 DSP tiles, 411 slices	10 cycle/Gauss
Log-add lookup	164/150	13 BRAMs, 307 slices	10 cycle/comp
AM block total	164/125	6 DSP tiles, 30 BRAMS, 1328 slices	162 cycle/senone 640 K cycle total
Hidden Markov model pipeline	261/140	775 slices	8 cycle/load 5 cycle/calc
Pruner	277/177	112 slices	4 cycle/HMM
PE block total	115/111	84 BRAMs, 1866 slices	22 cycle/HMM
Token deactivator	377/170	54 Slices	2 cycle/dead HMM
Token activator	184/120	3 BRAMs, 160 slices	10*branch cycle/active HMM
WM block total	166/129	3 BRAMs, 414 slices	22 cycle/dead HMM + 10*branch cycle/active HMM

trees for the creation of word strings. The deactivation portion of the hardware is capable of running at 170 MHz post-place-and-route but the activation logic can only operate at 120 MHz, limiting the overall performance of the word modeler but not impacting the overall performance of the system.

The majority of the verification for the design was done through post-place-and-route simulations models and comparing their results to the results obtained in the MATLAB environment for the SPHINX 3 models. Knowing that our MATLAB model provided a one-to-one representation of the SPHINX system, we were confident that if the MATLAB results were identical to the hardware results, we had correctly implemented the algorithms. Table 7 summarizes the performance results for the major portions of the hardware development.

7. CONCLUSIONS

In this work we have shown the ability to implement high-performance acoustic modeling pipeline on an FPGA device. Further we designed a unique architecture for a design capable of performing critical operations in the speech recognition process in real time with minimized power consumption and maximum processing bandwidth. Our system also highlights an architecture built to be driven completely by the data leading to a system that can be reprogrammed for multiple applications and dictionaries by altering the input to the system. This research has proven the effectiveness of the proposed design methodology and helped further the development of portable low-power speech recognition systems.

REFERENCES

[1] K. K. Agaram, S. W. Keckler, and D. Burger, "Characterizing the SPHINX speech recognition system," Tech. Rep. TR2001-18, Department of Computer Sciences, University of Texas at Austin, Austin, Tex, USA, January 2001.

[2] C. Lai, S.-L. Lu, and Q. Zhao, "Performance analysis of speech recognition software," in *Proceedings of the 5th Workshop on Computer Architecture Evaluation Using Commercial Workloads*, Cambridge, Mass, USA, February 2002.

[3] M. Ravishankar, R. Singh, B. Raj, and R. Stern, "The 1999 CMU 10x real time broadcast news transcription system," in *Proceedings of DARPA Workshop on Automatic Transcription of Broadcast News*, Washington, DC, USA, May 2000.

[4] L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall Signal Processing Series, Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[5] X. Huang, A. Acero, and H. Hon, *Spoken Language Processing*, Prentice Hall, Englewood Cliffs, NJ, USA, 2001.

[6] Results or a medium vocabulary test, CMU Sphinx, http://cmusphinx.sourceforge.net/MediumVocabResults.html.

[7] ARM922T (Rev 0) Technical Reference Manual, ARM.

[8] T. S. Anantharaman and R. Bisiani, "A hardware accelerator for speech recognition algorithms," in *Proceedings of the 13th Annual International Symposium on Computer Architecture (ISCA '86)*, pp. 216–223, Tokyo, Japan, June 1986.

[9] S. Nedevschi, R. K. Patra, and E. A. Brewer, "Hardware speech recognition for user interfaces in low cost, low power devices," in *Proceedings of Design Automation Conference (DAC '05)*, pp. 684–689, Anaheim, Calif, USA, June 2005.

[10] P. Placeway, S. Chen, M. Eskenazi, et al., "The 1996 Hub-4 Sphinx-3 system," in *Proceedings of the DARPA Speech Recognition Workshop*, pp. 85–89, Chantilly, Va, USA, February 1997.

[11] R. Hoare, K. Gupta, and J. Schuster, "Speech silicon: a data-driven SoC for performing hidden Markov model based speech recognition," in *Proceedings of High Performance Embedded Computing Workshop (HPEC '05)*, MIT, Lexington, Mass, USA, September 2005.

[12] R. Hoare, et al., "A hardware based acoustic modeling pipeline for hidden Markov model based speech recognition," in *Proceedings of 13th Reconfigurable Architectures Workshop (RAW '06)*, Rhodes Island, Greece, April 2006.

[13] J. Nouza, "Feature selection methods for hidden Markov model-based speech recognition," in *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 2, pp. 186–190, Vienna, Austria, August 1996.

[14] B. Mathew, A. Davis, and Z. Fang, "A low-power accelerator for the SPHINX 3 speech recognition system," in *Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '03)*, pp. 210–219, San Jose, Calif, USA, November 2003.

[15] CMU Sphinx, http://cmusphinx.sourceforge.net/html/cmusphinx.php.

[16] Linguistic Data Consortium, http://www.ldc.upenn.edu/.

[17] X. Li and J. Bilmes, "Feature pruning in likelihood evaluation of HMM-based speech recognition," in *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU '03)*, pp. 303–308, St. Thomas, Virgin Islands, USA, November-December 2003.

[18] M. Ravishankar, *Efficient algorithms for speech recognition*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pa, USA, May 1996, CMU-CS-96-143.

Jeffrey Schuster received his B.S.E.E from the University of Pittsburgh's Department of Electrical Engineering in 2004 focusing his work on signal processing, specifically for audio and speech implementations. During his studies he began work for Drs. Raymond Hoare and Amro El-Jaroudi designing parallel speech processing systems based on the SPHINX 3 engine from Carnegie Mellon University. Through this work funding was obtained from the Tech Collaborative, formerly the Pittsburgh Digital Greenhouse, to complete his M.S.E.E. developing his work into hardware based solutions to the speech recognition problem. The work completed at the University of Pittsburgh let to the derivation of numerous papers presented at conferences such as the HPEC 2005 event at MIT and the RAW 2006 event in Rhodes, Greece. After completion of his M.S.E.E. he accepted a position with the hardware engineering division of Exegy Inc. located in St. Louis, MO working to design the next generation of high speed data search appliances. He currently resides in the St. Louis area where his free time is spent applying his love of engineering and computers to his passion for music.

Kshitij Gupta received his M.S. degree in electrical engineering from the University of Pittsburgh, Pittsburgh (USA) in 2005. With primary focus on FPGA/system-on-chip design, he has over three years of research experience in analyzing, optimizing, and implementing algorithms employed in speech recognition systems onto hardware platforms, towards speech-on-silicon. Prior to this, he received his B.E. degree in electronics & communication engineering from Osmania University, Hyderabad (India) in 2002. His current interests include design, implementation & verification of custom hardware, and multiprocessor SoC-based designs using FPGAs, ASICs, and configurable processors for high-performance computing and embedded consumer applications.

Raymond Hoare received his B.E. degree from Steven's Institute of Technology in 1991. He received his M.S. degree from the University of Maryland in 1994. He received his Ph.D. degree from Purdue University in 1999. He is currently the President and Founder of Concurrent Design Automation in Pittsburgh, Pennsylvania. Previously, he was an Assistant Professor of electrical and computer engineering at the University of Pittsburgh in Pittsburgh, Pennsylvania. His research interests include high-performance parallel architectures, communication networks, systems-on-a-chip, and design automation.

Alex K. Jones received his B.S. degree in 1998 in physics from the College of William and Mary in Williamsburg, Virginia. He received his M.S. and Ph.D. degrees in 2000 and 2002, respectively, in electrical and computer engineering from Northwestern University. He is currently an Assistant Professor of Electrical and Computer Engineering and Computer Science at the University of Pittsburgh, Pennsylvania. He was formerly a Research Associate in the Center for Parallel and Distributed Computing and Instructor of electrical and computer engineering at Northwestern University. He is a Walter P. Murphy Fellow of Northwestern University, a distinction he was awarded twice. His research interests include compilation techniques for behavioral and low-power synthesis, embedded systems, radio frequency identification (RFID), and high-performance computing. He is the author of over 40 publications in these areas.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 72962, Pages 1–8
DOI 10.1155/ES/2006/72962

A Visual Environment for Real-Time Image Processing in Hardware (VERTIPH)

C. T. Johnston, D. G. Bailey, and P. Lyons

Institute of Information Sciences and Technology, Massey University, Private Bag 11222, Palmerston North 4442, New Zealand

Received 14 December 2005; Revised 4 May 2006; Accepted 28 May 2006

Real-time video processing is an image-processing application that is ideally suited to implementation on FPGAs. We discuss the strengths and weaknesses of a number of existing languages and hardware compilers that have been developed for specifying image processing algorithms on FPGAs. We propose VERTIPH, a new multiple-view visual language that avoids the weaknesses we identify. A VERTIPH design incorporates three different views, each tailored to a different aspect of the image processing system under development; an overall architectural view, a computational view, and a resource and scheduling view.

1. INTRODUCTION

FPGAs (field programmable gate arrays) are ideal in many embedded systems applications because they have several desirable attributes: small size, low-power consumption, a large number of I/O ports, and a large number of computational logic blocks. As they have grown in size and functionality, there has been increasing interest in using them as implementation platforms for image processing applications, particularly real-time video processing [1]. Images have a high degree of spatial parallelism, and thus image processing applications are ideally suited to implementation on FPGAs, which contain large arrays of parallel logic and registers and can support pipelined algorithms.

However, there is a significant cost in obtaining the increased performance of FPGAs because their architecture differs significantly from the fixed architecture of standard processors. As Offen [2] has stated, the classical serial architecture is so central to modern computing that the architecture-algorithm duality is firmly skewed towards this type of architecture. Consequently, most image processing practitioners are not familiar with parallel programming issues such as concurrency, pipelining, priming, and bandwidth issues.

Programming FPGAs differs significantly from writing software for conventional single-processor, large-memory systems in another respect. With FPGA-based designs one designs not only the algorithm, but also the architecture on which it is implemented. FPGA-based designs generally comprise a large number of simple processors which all work in parallel and may compete for memory access or other resources. In designing an appropriate algorithm for the FPGA it is therefore necessary to take into account the limited bandwidth, particularly when accessing memory.

The three main processing models used for image processing algorithms on FPGAs—stream, offline, and hybrid processing—have differing characteristics.

In *stream processing*, data is presented as a one-dimensional pixel stream by means of a suitable access pattern [3], typically raster order (in which pixels are presented left to right for each image row beginning with the top row). This converts the spatial distribution to a temporal stream and is often used for processing video data in real time as the data is streamed through the system. This type of processing is well suited to stand-alone configurations—for example, a system in which an FPGA fed directly by a continuous stream of data from a video source is acting as the "front end" of a smart camera, processing the image from a sensor before storing the result into memory.

The strict time constraints involved with stream processing depend on the video capture rate and image size (e.g., each of the 25 frames that PAL produces per second contains a 768 by 576 colour image). Stream processing constrains the design into performing all of the required calculations for each pixel at the pixel clock rate. If this is not possible, then some pixels in the stream will be missed and so will not be processed.

In some nontrivial applications, such as lens distortion correction [4, 5] or object tracking [6, 7] it is difficult to achieve these high data rates, because each pixel requires complex calculations that may easily exceed a single clock cycle. In such situations it is common to break the calculation down into several phases, and to implement the hardware algorithm as a pipeline, with one clock cycle allocated to each stage. At any instant, successive stages of the pipeline will contain pixels at successive stages of processing. The overall rate of output will be one pixel per clock cycle, but there may be a latency of several clock cycles between inputting a raw pixel and outputting the processed result. Pipelining is an important technique for exploiting the temporal parallelism inherent in stream data.

In stream processing, memory bandwidth constraints dictate that as much processing as possible is performed on the data as it arrives. For some operations, the order in which pixels are required for processing does not directly correspond to their arrival order from the raster, so the image must be partly or wholly buffered. However, memory is limited on an FPGA, and applications that require full-frame buffering, such as image warping, must typically use off-chip memory, which introduces additional latency. In applications where multiple accesses are required such as bilinear interpolation [8], the limited bandwidth and serial access make it difficult to retrieve desired pixel values.

Offline processing is commonly used in hosted system configurations. In such a configuration, the FPGA is a coprocessor in the embedded system, whose role is to complement the host computer by accelerating certain tasks. In this mode, there is no longer the strict timing constraint on the processing; random access to shared memory is possible and desired pixel values can be obtained over a number of clock cycles. This allows the bandwidth constraints to be relaxed at the expense of processing time.

Hybrid processing combines stream and offline processing. For example, stream processing can be used for image capture and display while offline processing can be used to provide random access to a region of interest in the captured image.

VERTIPH is a visual programming language that has been designed to capture algorithms for real-time video processing on FPGAs. This application area has a number of specialised requirements, and VERTIPH provides three views of a design. Each view is tailored to the characteristics of a particular level of abstraction. Before describing these views in detail, we shall characterise some existing languages designed for capturing image processing applications.

2. PRESENT LANGUAGES

Schematic entry is too low-level as a design tool for image processing as it does not capture the algorithmic nature of image processing functions adequately. HDLs (hardware description languages) were developed to allow designers to capture the high-level temporal behaviour of complex digital designs as well as their circuit structure. Verilog [9] and VHDL [10] are industry standard HDLs. Such languages can be thought of as the assemblers of hardware programming providing great flexibility from gate level up to the behavioural level. As most of them offer similar functionality, we will concentrate on two, VHDL and JHDL. The low-level constructs supported by VHDL make it a poor choice for implementing complex image processing algorithms. As a general purpose language, VHDL offers no specific support for image processing operations. While HDLs offer a great deal of flexibility in terms of the control logic it is up to the designer to construct any state machines required to control the system. This can be advantageous, thus allowing very efficient control over the execution path. However this burdens the designer with designing both algorithm and the control logic.

JHDL [11–13] is a structural HDL developed specifically for custom computing machine design on FPGA devices. This has led to a language which is more intuitive and easy to learn than existing FPGA design tools. JHDL incorporates the ability to design a circuit and simulate this circuit in an integrated package. This includes visualisation tools for the design including: schematics, waveform diagrams, memory views, and hierarchical design viewers. The biggest advantage of JHDL over other low-level HDLs is the integration of the development and debug environments.

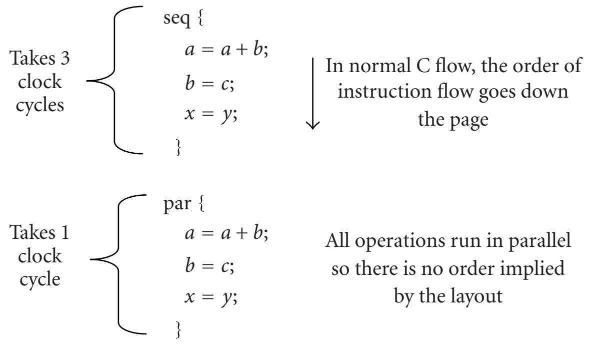
The power and flexibility of HDLs imposes an exacting low-level programming style that can obscure the broad sweep of a high-level algorithm. There have been a number of approaches to producing high-level design tools that circumvent this problem.

One is to modify an existing software programming language to add in the constructs required for building hardware. In most conventional programming languages, statements are executed sequentially following the order of assignment statements, and branches are specified by flow-of-control (*while*-, *if*-, etc.) statements. In general, conventional programming languages do not offer the ability to run processes in parallel, although some support process threads. The lengths of data types are defined by either the fixed architecture of the processor (ANSI-C) or by the language (Java). These languages are not designed to be compiled into hardware, so they lack hardware-oriented constructs such as ways to define communication between different processes, to create RAMs, and to assign I/O pins.

There are five main areas in which conventional programming languages need to be extended in order to support hardware design. It should be possible

(i) to build architectural components such as RAMs, ROMs, WOMs, channels,
(ii) to specify that operations occur concurrently and to specify the timing or clock speed of processes,
(iii) to define communication between processes running at different speeds,
(iv) to create low-level structures such as wires along with bit-level operations such as bit concatenation,
(v) to define data types in terms of their bit length.

Handel-C compiles algorithms written in a high-level C-like language directly into gate-level netlists. It is based on a subset of ANSI-C with hardware-oriented syntax extensions such as variable data widths, parallel processing, and channel communication between parallel processing blocks. The language is designed to allow software engineers to express an



```
seq {
    a = a + b;
    b = c;
    x = y;
}

par {
    a = a + b;
    b = c;
    x = y;
}
```

Takes 3 clock cycles
In normal C flow, the order of instruction flow goes down the page
Takes 1 clock cycle
All operations run in parallel so there is no order implied by the layout

Figure 1: Logical flow of instructions.

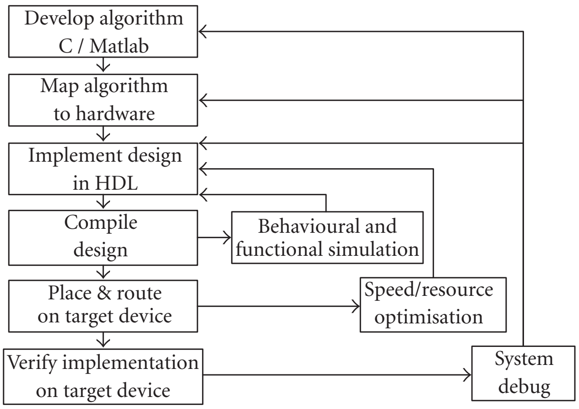

Develop algorithm C / Matlab
Map algorithm to hardware
Implement design in HDL
Compile design
Behavioural and functional simulation
Place & route on target device
Speed/resource optimisation
Verify implementation on target device
System debug

Figure 2: Image processing on FPGA design flow.

algorithm without requiring any knowledge of the underlying hardware [14]. Apart from the introduction of architectural constructs and bit-level operations, the only significant difference between ANSI-C and Handel-C is the introduction of the *par* construct. All statements within a *par* block run in parallel.

Handel-C provides a good level of abstraction from hardware design. However, its textual nature makes the data flow in a parallel design difficult to understand. Figure 1 shows that there is almost no visual difference between sequential and parallel codes. This is common to all text-based HDLs.

The increased ability to concentrate on algorithm development comes at a cost: loss of control over details such as control flow; Handel-C builds an implied state machine to control the data processors.

Another approach is the hardware compiler which takes all the hardware design decisions except data-type lengths away from the designer. This approach has been taken by SA-C [15, 16] and MATCH [17].

SA-C incorporates common image processing functions such as array summing for histograms and window loops. It exploits parallelism primarily through loop unrolling and low-level pipelining.

These systems take all control away from the designer. They can achieve real-time operation using an offline design model. However they can only optimise an algorithm through pipelining the sequential algorithm.

While many image processing algorithms are inherently parallel, they are commonly expressed serially, for implementation on a serial processor. For example a filter is parallel in its specification, but is normally implemented as loops. Most image processing applications involve several steps which can each run concurrently as pipelined processes. It is therefore desirable to have a development tool which allows this parallelism to be captured at an appropriate level of abstraction.

3. CURRENT APPROACH

When implementing algorithms on an FPGA we have used the design flow shown in Figure 2.

Most of the effort in following this path is in the first step: mapping an algorithm into a form suitable for FPGA implementation, generally using a stream processing model. The aim is to make the implementation as efficient as possible, which we accomplish by coarse-gain pipelining (between operations), fine-grain pipelining (breaking up operations), combining operations into one, utilising look up tables, CORDIC functions, and redesigning a standard algorithm for a single-pass implementation.

This high-level design is then implemented onto hardware using a hardware language, such as Handel-C. There is a large semantic gap between our design mapping and the hardware languages used to implement the design. A high-level language for expressing image processing algorithms in hardware should make this gap easier to bridge. It should

(i) allow a mixture of parallel and sequential design;
(ii) make it clear to the designer what runs in parallel and what forms part of a pipeline;
(iii) be able to detect when concurrent processes may access a shared resource such as a RAM, and manage this accordingly by informing the designer and giving some suggestions as to how to resolve the issue;
(iv) be able to handle stream, offline, and hybrid processing models;
(v) include some of the common image processing functions and data types as primitives. Examples include row and pixel buffering, window filters, and look up tables (LUT);
(vi) be intuitive and easy to use;
(vii) provide multiple views onto the design.

Currently no system incorporates all of these features, and this paper describes a system which meets these requirements.

Visual design tools can aid in the specification and development of image processing algorithms. There have been a number of different visual image processing languages for use on a serial computer including Khoros [18] and OpShop [19]. There are also several general purpose visual languages which can be used for image processing, including LabView [20] and Simulink [21]. Khoros, LabView, and Simulink now have extensions that allow them to be used for FPGA design,

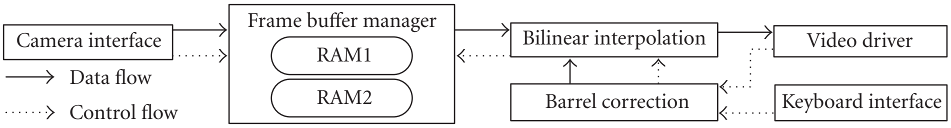


Figure 3: Architecture view of a barrel distortion correction system showing components, control, and data flows.

although this was not their original purpose. Khoros offers a high-level view for algorithm development, but it does not include lower-level design capabilities, as it was not designed to support the implementation of novel image processing operations. Recently other IP-based systems such as Celoxica's PixelStreams [22] and Xilinx's DSP block sets [23] have been developed to provide faster development time for projects and provide similar functionality to Khoros.

These languages all follow a form of the dataflow paradigm where streams of data flow through a network of nodes, each of which performs a computation on the tokens within the stream before passing the output data to the next node [24]. It has been noted [25] that dataflow graphs (the natural visual representation of this programming paradigm) are an effective representation for problems in digital signal processing (DSP), both because they are a natural representation for many DSP researchers and because they expose parallelism in the algorithm with limited constraints on evaluation order.

4. VERTIPH

As discussed in Section 3, textual languages represent concurrency and complex scheduling poorly. We have developed VERTIPH, which incorporates a visual representation for representing the parallel design of image processing algorithms. As image processing algorithms often involve a number of largely independent processing blocks, a suitable high-level view allows the designer to specify the data flow through a sequence of modules. This is then augmented with lower-level views that support the definition of parallel computations that make up the higher-level modules. Finally a resource and scheduling view is provided, so that the designer can specify the timing between the operations, and access to resources. These three are the defined views of the VERTIPH system: the top-level *architecture view*, a *computational view*, and the *scheduling and resource view*. A comparison of VERTIPH with other HDLs and its required features was presented in [26]. This work expands on VERTIPH's features including data types and operators.

4.1. Architecture view

The *architecture view* (Figure 3) aims to provide the designer with a perspective on the overall system. As image processing algorithms are broken up into blocks which perform very specific processing tasks, they can be developed independently and validated using test image data. This view allows the designer to construct an image processing algorithm as several blocks that operate sequentially on the image data. Khoros and OpShop are other systems that act at a similar level.

The use of component blocks allows resources such as frame buffers to be encapsulated, and related computational processes to be logically grouped. For example, a frame buffer component will have both an input stream and an output stream, and it will contain two RAM banks. Other components which communicate with this only see address and data lines and the switching between memory banks can be done within the component.

Processors which are logically related to each other are also encapsulated. For example, a colour segmentation and tracking algorithm detailed in [6, 7], represents each uniquely coloured detected object as a bounding box. It stores the bounding boxes for each colour class in a data structure, and it incorporates processors for tracking-related bounding boxes between frames and for calculating the position of all the bounding boxes that have been detected. The data structure and the processors are logically related and should therefore be kept together. This idea of encapsulation borrows from object-orientated software engineering.

Encapsulation simplifies the sharing of data and resources and it becomes clear which processor can access them and for what purpose. It can in turn make it easier to schedule these processors, as the developer does not need to remember all the parts of the system which are related to the resource or data structure being used.

Hierarchical encapsulation can allow for very complex IP blocks to be built, with one block and interfaces representing a complex system of data structures, resources, processes, and their scheduled operations or response to events. It also allows for a hierarchy of state machines to be used, with each component within a component having its own state machine which may or may not then be controlled by a higher level of the design.

The aim of the architectural view is to allow logical separation of image processing operators, to show the data flow through the operators, and to encapsulate data and processors related to each operation.

Data types

Data types commonly encountered in image processing include 16-, 24-, and 32-bit colour, 8- and 16-bit grey scale, and signed and unsigned integers, and fixed point numbers of arbitrary size. The user therefore needs to be able to specify the type of a data stream; that is, its size and format. And, as communication in FPGAs may be by channel, by register, or by wire (no storage), it is appropriate to include path-type information in the type specification along with the more traditional size and format information.

The data flow between high-level blocks is shown in VERTIPH's architecture view, so the architecture view editor incorporates type checking to give the user feedback about whether the data being output from one block is acceptable as input to another. This happens as soon as a connection is established between two high-level operators. The data types of the output that drives the connection and the input that it feeds into are immediately compared to ensure that they are of the same type, and the user is informed if they do not match.

Floating point numbers have not been included within the system for several reasons: 32- or 64-bit IEEE standard 754 floating point numbers are expensive to implement in terms of memory, circuit size, and power consumption. Image processing operations generally do not require the dynamic range which floating point offers. Fixed point numbers offer better overall noise performance when the probability density function of the signals is uniform [27]. As long as appropriate fixed point word lengths are chosen, almost all standard image processing operations can be implemented (with some degree of rounding error). Fixed point operations have a small footprint in hardware and lead to lower-power consumption, thus making them the best choice for embedded applications [27].

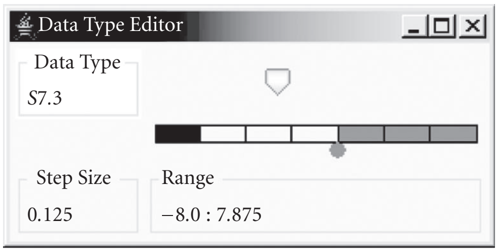
Figure 4 is the dialogue for specifying the size and range of fixed point numbers in VERTIPH. The dialogue allows the number of bits before and after the binary point to be altered, using either a slider interface or a text box.

Another advantage to capturing type information is that this information can be used to automatically align values for arithmetic manipulation, and to generate a register of the correct width to store the result. Figure 5 shows the intermediate registers required to implement a multiphase calculation involving a multiplication, an addition, and a subtraction. It shows that if the order of operations is changed, the registers for the intermediate results must be altered. This is an exacting task, and—if it is performed by the designer—a fruitful source of errors, but the availability of type information in VERTIPH makes it possible to eliminate the errors by calculating the register sizes automatically.

Specialised operators

Window filters are a very common low-level image processing operator. There are several forms that a window operator can take in hardware [28], and they need to be tailored to the application. Therefore a design wizard for constructing operators of this type has been developed for VERTIPH.

As in other fields, certain patterns are found repeatedly in the design of image processing systems. Each application shares some properties with other applications, and each application has some unique parameters. We have therefore designed VERTIPH to allow new patterns to be incorporated into the language as wizards. The window operator is simply the first of these.



Data Type Editor

Data Type: S7.3

Step Size: 0.125

Range: −8.0 : 7.875

Figure 4: Fixed point data type editor.

$$\underbrace{\underbrace{S4 * U5}_{S9} + \underbrace{U6 - U4}_{U6}}_{S10} \qquad \underbrace{\underbrace{\underbrace{S4 * U5}_{S9} + U6}_{S10} - U4}_{S10}$$

S	Signed
U	Unsigned
9	9 bits

Figure 5: Different temporary register sizes depending on arithmetic order.

4.2. Computational view

Developers who never design their own algorithms can use the architecture view's editor to assemble predefined library modules into a high-level overview like the one shown in Figure 3. This is similar to the way that other IP-based systems such as Celoxica's PixelStreams and Xilinx's DSP block sets operate.

However, to allow developers to design their own operations and to help with buffering, pipeline priming, and synchronisation, a lower level timing view is needed. The *computational view* aims to improve the visualisation of the concurrent aspects of the low-level computations. To accomplish this we have modified the Gantt chart notation [29] which was designed as a visual tool to highlight the temporal relationships and dependencies between phases in large construction projects, and thus make it easy to schedule time-critical activities. In this notation, time flows from left to right, so Figure 6(a) shows a sequential set of operations; operation **A** is followed by operation **B**, which is followed by operation **C**. In Figure 6(b), the operations occur concurrently, and in Figure 6(c) they are pipelined. This representation is an abbreviation of Figure 6(d) which explicitly shows the parallel repeating processes, the passage of data from one to the other, and that each process is active in succeeding phases.

Of course, these basic types can be used together as shown in Figure 7, which is the pipeline for row processing used by the barrel distortion algorithm [5]. This figure also shows the *if*- and *while*- control structures provided by VERTIPH, which are based on the control structures used in Nassi-Shneiderman diagrams [30]. The top bar displays the

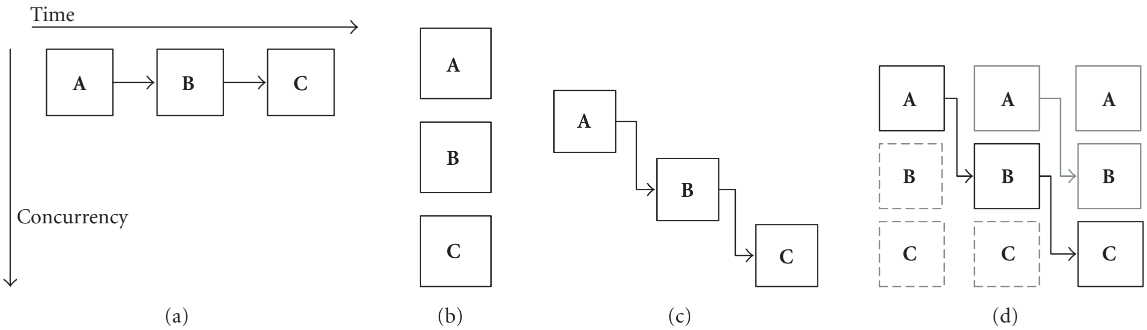


Figure 6: Process representations: (a) sequential, (b) parallel, (c) pipelined, (d) actual pipeline structure.

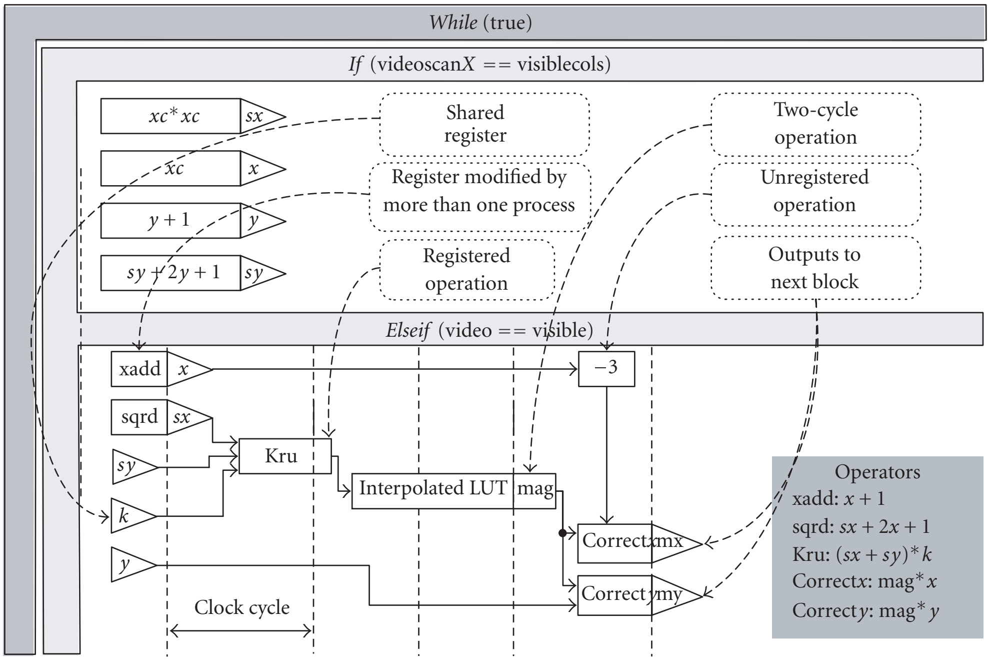


Figure 7: Low-level view of Barrel distortion block showing control functions, timing, and operation representation. Note that text in dashed boxes are comments added to the figure for clarification; they do not form part of the language.

control expression for the structure, with the vertical bar enclosing the processors controlled. This pipeline view graphically conveys to the developer the time required to prime and flush the pipeline.

Operations can be registered or unregistered with unregistered operations having to be fed into a register before a clock cycle can finish. To save space on the screen only the operation or register name is shown, an operations key has the instructions for the block in a C-type syntax. This view shows the same information as a textual language but the layout makes the structure of the algorithm easier to visualise. For example, it is easy to see that the x value must be offset by 3 before it is used in the calculation of the undistorted x value. Additional visual linkage between the operations and the key can be provided by highlighting the operation in the key when the mouse is moved over the corresponding box, and vice versa.

The language should, where possible, automatically generate structures to handle pipeline priming, stalling, and flushing and it should prompt developers when their design might be using values from a different stage of the pipeline.

4.3. Resource and scheduling view

In an embedded image processing system using FPGAs there are a large number of processors competing for access to a limited number of resources. There are also processors which can only run after certain events have occurred, such as an external trigger or another processor finishing. These competing and cooperative processors need to be managed and

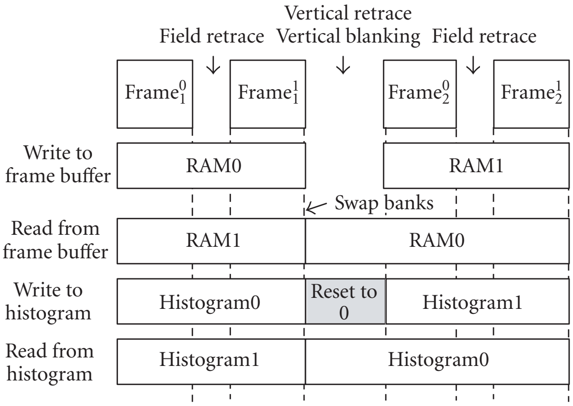


Figure 8: Timing of processes and resources used for a streamed histogram function.

scheduled. VERTIPH facilitates resource sharing by encapsulating resources and the processes that act on them, so that the processes can be scheduled. The *resource and scheduling view* also allows for both global scheduling for processors and for local scheduling within components.

To help the designer avoid resource conflicts such as two parallel processes accessing an external RAM at the same time, a resource usage view is incorporated. This view works like standard Gantt software packages and identifies when resources are used more than once in a time period. The view can then suggest changes in the ordering of events. In the case of a multiprocess design, this would involve either modifying start conditions for processes (to ensure they do not run together) or using semaphores or other similar mechanisms to arbitrate access to the resource. For a time-critical design such as stream processing from a video camera, the blocking approach is not desirable as it can cause data to be lost, such as when writing from a pixel stream to a frame buffer. Fortunately, blanking periods or pixel buffering can often be used to allow changes in the scheduling of competing processes. This view can also help in the scheduling of processes which run only at specific times—for example, when a new frame is received—or for identifying where caching of pixels would be more appropriate than memory access, such as when a RAM access occurs when another process is using the RAM and no rescheduling is possible.

One example of this type of resource conflict can occur when a histogram is being constructed and displayed. It is desirable to construct the histogram while the video stream is buffered into one RAM. At the same time, in a different clock domain, both the last full image and its histogram are being processed or displayed. Keeping one of these processes from trying to write to one RAM while the other is reading can be accomplished with a simple condition test. The problem occurs due to the need to reset the histogram values in each bin before the histogram construction algorithm is run, as shown in Figure 8. While this requires a more complex passing of control of resources from process to process, it can also lead to error.

5. DISCUSSION

This work has identified several existing languages which are used for image processing on FPGAs, and commented on both their benefits and limitations.

A new visual language, VERTIPH, has been presented. VERTIPH makes sequential, concurrent, and pipelined operations clear to the developer. It also breaks the design process into three parts to aid in its implementation. VERTIPH includes a block-level architecture view similar to many other DSP block set systems; a computational view based on Nassi-Shneiderman diagrams that expresses the operations required in each block; and a resource and scheduling view to aid in the development of the complex state machines that are required to respond to events and to avoid resource contention between processors. At present the block-level design view and data-type implementation are nearing completion, with the computational and scheduling views still to be implemented.

VERTIPH is only one of several approaches that can be taken when developing image processing systems on FPGAs; it is a step towards better tools and methodologies that will make FPGAs more usable and useful for embedded image processing applications.

REFERENCES

[1] J. Villasenor and B. Hutchings, "The flexibility of configurable computing," *IEEE Signal Processing Magazine*, vol. 15, no. 5, pp. 67–84, 1998.

[2] R. J. Offen, *VLSI Image Processing*, Collins, London, UK, 1st edition, 1985.

[3] V. M. Bove Jr., M. M. Lee, Y.-M. Liu, C. M. McEniry, T. M. Nwodoh, and J. M. Watlington, "Media processing with field-programmable gate arrays on a microprocessor's local bus," in *Media Processors 1999*, vol. 3655 of *Proceedings of SPIE - The International Society for Optical Engineering*, pp. 14–20, San Jose, Calif, USA, January 1999.

[4] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "Implementing image processing algorithms on FPGAs," in *Proceedings of the 11th Electronics New Zealand Conference (ENZCon '04)*, pp. 118–123, Palmerston North, New Zealand, November 2004.

[5] K. T. Gribbon, C. T. Johnston, and D. G. Bailey, "A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation," in *Proceedings of Image and Vision Computing New Zealand (IVCNZ '03)*, pp. 408–413, Massey University, Palmerston North, New Zealand, November 2003.

[6] C. T. Johnston, D. G. Bailey, and K. T. Gribbon, "Optimisation of a colour segmentation and tracking algorithm for real-time FPGA implementation," in *Proceedings of Image and Vision Computing Conference New Zealand (IVCNZ '05)*, pp. 422–427, Dunedin, New Zealand, November 2005.

[7] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "FPGA based remote object tracking for real-time control," in *Proceedings of International Conference on Sensing Technology (ICST '05)*, pp. 66–72, Palmerston North, New Zealand, November 2005.

[8] K. T. Gribbon and D. G. Bailey, "A novel approach to real-time bilinear interpolation," in *Proceedings of 2nd IEEE International Workshop on Electronic Design, Test and Applications (DELTA '04)*, pp. 126–131, Perth, Australia, January 2004.

[9] IEEE Standard Verilog Hardware Description Language, visited on August 2004, http://www.verilog.com/IEEEVerilog.html

[10] J. Bhasker, *A VHDL Primer*, Prentice-Hall, Englewood Cliffs, NJ, USA, 3rd edition, 1999.

[11] Brigham Young University, JHDL, visited on 21 February 2005, www.jhdl.org.

[12] P. Bellows and B. Hutchings, "JHDL-an HDL for reconfigurable systems," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '98)*, pp. 175–184, Napa Valley, Calif, USA, April 1998.

[13] P. Bellows and B. Hutchings, "Designing run-time reconfigurable systems with JHDL," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 28, no. 1-2, pp. 29–45, 2001.

[14] I. Alston and B. Madahar, "From C to netlists: hardware engineering for software engineers?" *Electronics and Communication Engineering Journal*, vol. 14, no. 4, pp. 165–173, 2002.

[15] R. Rinker, J. Hammes, W. A. Najjar, W. Bohm, and B. Draper, "Compiling image processing applications to reconfigurable hardware," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 56–65, Boston, Mass, USA, July 2000.

[16] J. Hammes, B. Rinker, W. Bohm, W. Najjar, B. Draper, and R. Beveridge, "Cameron: high level language compilation for reconfigurable systems," in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT '99)*, pp. 236–244, Newport Beach, Calif, USA, October 1999.

[17] P. Banerjee, N. Shenoy, A. Choudhary, et al., "A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems," in *Proceedings of 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pp. 39–48, Napa Valley, Calif, USA, April 2000.

[18] K. Konstantinides and J. R. Rasure, "The Khoros software development environment for image and signal processing," *IEEE Transactions on Image Processing*, vol. 3, no. 3, pp. 243–252, 1994.

[19] P. M. Ngan, *The development of a visual language for image processing applications*, Ph.D. thesis, Computer Science, Massey University, Palmerston North, New Zealand, 1992.

[20] National Instruments LabVIEW, visited on 16 February 2005, www.ni.com/labview.

[21] The MathWorks, Simulink 6.1, visited on 16 February 2005, www.mathworks.com/products/simulink/.

[22] Celoxica, *PixelStreams Manual*, 1st ed: Celoxica, 2005.

[23] Xilinx System Generator for DSP Blockset, visited on November 2005, http://www.xilinx.com/products/software/sysgen/.blockset.htm

[24] J. T. Buck, *Scheduling dynamic dataflow graphs with bounded memory using the token flow model*, Ph.D. thesis, Electrical Engineering and Computer Sciences, University of California, Berkeley, Calif, USA, 1993.

[25] J. T. Buck and E. A. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '93)*, vol. 1, pp. 429–432, Minneapolis, Minn, USA, April 1993.

[26] C. T. Johnston, D. G. Bailey, P. Lyons, and K. T. Gribbon, "Formalisation of a visual environment for real time image processing in hardware (VERTIPH)," in *Proceedings of Image and Vision Computing New Zealand (IVCNZ '04)*, pp. 291–296, Akaroa, New Zealand, November 2004.

[27] A. S. L. Bainbridge-Smith, "Real number representation for image processing on FPGAs," in *Proceedings of Image and Vision Computing New Zealand (IVCNZ '05)*, pp. 471–475, Dunedin, New Zealand, November 2005.

[28] K. T. Gribbon, C. T. Johnston, and D. G. Bailey, "Formalizing design patterns for image processing algorithm development on FPGAs," in *Proceedings of IEEE Tencon Conference*, pp. 21–24, Melbourne, Australia, November 2005.

[29] J. R. Schermerhorn, *Management*, John Wiley & Sons, New York, NY, USA, 6th edition, 2001.

[30] I. Nassi and B. Shneiderman, "Flowchart techniques for structured programming," *ACM SIGPLAN Notices*, vol. 8, no. 8, pp. 12–26, 1973.

C. T. Johnston is a Ph.D. candidate in information engineering at the Institute of Information Sciences and Technology, Massey University, New Zealand. He received a Bachelor of Engineering degree with first class honours in information and telecommunications engineering from Massey University. His research focus has been in the area of implementing image processing algorithms on FPGA hardware, concentrating on designing a visual programming language to aid in the implementation of image processing algorithms.

D. G. Bailey has B.E. (Hons) and Ph.D. degrees in electrical and electronic engineering from the University of Canterbury, New Zealand. After spending two years applying image analysis techniques to the wool and paper industries within New Zealand, he spent two and half years as a Visiting Researcher in the Electrical and Computer Engineering Department, University of California, Santa Barbara. In 1989, he returned to New Zealand as a Director of the Image Analysis Unit at Massey University. In 1998 he moved to the Institute of Information Sciences and Technology where he is currently a Senior Lecturer and Leader of the Image and Signal Processing Research Group. His primary research interests are in the application of signal processing, image analysis, and image processing techniques. One research topic of particular interest is the implementation of imaging algorithms on FPGAs.

P. Lyons is a Senior Lecturer in the Institute of Information Sciences and Technology at Massey University, where he teaches computer architecture and human-computer interaction, amongst other things. He has been involved in research into visual programming languages for a number of years. Although his main focus in this area has been on general purpose programming languages, he has previously been involved in the design of two VPLs related to electronics design, PICSIL, a pictorial language for silicon compilation, and VISTA, a specialised language for specifying control systems for induction motors, which he designed under contract, and is now in use worldwide.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 81309, Pages 1–13
DOI 10.1155/ES/2006/81309

FPGA-Based Communications Receivers for Smart Antenna Array Embedded Systems

Constantin Siriteanu,[1, 2] Steven D. Blostein,[1] and James Millar[3]

[1] *Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada K7L 3N6*
[2] *Communications Signal Processing Laboratory, Department of Electrical and Computer Engineering, Hanyang University, Seoul, Korea*
[3] *CMC Microsystems, Kingston, ON, Canada K7L 3N6*

Received 15 December 2005; Revised 7 May 2006; Accepted 2 June 2006

Field-programmable gate arrays (FPGAs) are drawing ever increasing interest from designers of embedded wireless communications systems. They outpace digital signal processors (DSPs), through hardware execution of a wide range of parallelizable communications transceiver algorithms, at a fraction of the design and implementation effort and cost required for application-specific integrated circuits (ASICs). In our study, we employ an Altera Stratix FPGA development board, along with the DSP Builder software tool which acts as a high-level interface to the powerful Quartus II environment. We compare single- and multibranch FPGA-based receiver designs in terms of error rate performance and power consumption. We exploit FPGA operational flexibility and algorithm parallelism to design eigenmode-monitoring receivers that can adapt to variations in wireless channel statistics, for high-performing, inexpensive, smart antenna array embedded systems.

1. INTRODUCTION

Conventional wireless communications systems employ a single receiving antenna. Enhanced, antenna array receivers employing beamforming (BF) and maximal-ratio combining (MRC) can generate antenna and diversity gain, that is, increased average and instantaneous (with respect to channel fading) receiver signal-to-noise ratio (SNR) [1–4]. Although beneficial in terms of performance, these enhanced, multibranch algorithms can require much larger computational volumes than the conventional, single-branch receiver. Recent analytical and simulation studies [1–4] of a hybrid algorithm entitled maximal-ratio eigencombining (MREC) claimed efficient performance-complexity tradeoffs for smart antenna arrays.

Receiver algorithms have traditionally been deployed on general-purpose, sequential, digital signal processors (DSPs), or on application-specific integrated circuits (ASICs). Enhanced receiver algorithms, which are generally highly parallelizable, and higher data transmission rates can burden DSPs beyond their capacity for real-time processing. Time-critical, highly parallelizable applications are common in areas ranging from modern communications [5–7] to image [6] and speech [8] processing, and even bioinformatics [9]. ASICs are hardwired for specific tasks. Although fast (sometimes several orders of magnitude faster than DSPs, through hardware parallelism) and power-efficient, implemented designs are inflexible [7]. More importantly, ASIC design and production are time-consuming and extremely expensive for chips produced in small numbers, due to very high nonrecurring engineering cost.

Unlike ASICs, field-programmable gated arrays (FPGAs) are reconfigurable, that is, their internal structure is only partially fixed at fabrication, leaving to the application designer the wiring of the internal logic for the intended task. This can significantly shorten design and production, and thus time to market, for FPGA-based embedded systems. Although FPGAs tend to be slower and to consume more power than ASICs [7], FPGA reconfigurability can benefit platform longevity (which is extremely important in an era of fast-changing wireless communications standards) by allowing design changes/upgrades even in systems already in operation. This flexibility can be effectively exploited for rapid prototyping of advanced communications signal processing, such as Bell Labs Layered Space-Time (BLAST) multi-input multi-output (MIMO) architecture for third-generation Universal Mobile Telecommunications System (UMTS) [5]. Furthermore, an FPGA can, for example, implement MRC branches either sequentially, or in parallel, or anywhere in between, depending on required speed, available chip resources, and power constraints. FPGA-based implementations concurrently operating several hardware modules can outpace many times their processor-based counterparts [6, 9]. An insightful DSP, FPGA, and ASIC implementation comparison for a four-antenna orthogonal frequency-division multiplexing (OFDM) receiver can be found in [7].

FPGAs are especially well suited for embedded systems (e.g., cellular system base station line cards, or mobile stations) because, beside an area of reconfigurable logical elements, they can also incorporate large amounts of memory, high-speed DSP blocks, clock management circuitry, high-speed input/output (I/O), as well as support for external memory, and high-speed networking and communications bus standards. For a small share of the resources, processors can be included within the FPGA fabric as well [9].

Power consumed in embedded systems is, in general, strictly limited. Otherwise, line-powered designs would require special and/or expensive power sources and heat sinks or may not operate reliably, while portable devices would quickly deplete the battery [10, 11]. Although FPGA chips are judiciously manufactured for power efficiency, application designers also need to carefully consider this issue because a consistently underutilized design wastes static and dynamic powers [10–13].

The objective of this paper is to investigate FPGA suitability for efficient smart antenna array embedded receivers. In the process, we overview an Altera FPGA-based design environment, and implement conventional and enhanced (BF, MRC, MREC) receiver algorithms. It is demonstrated that FPGA implementations of eigenmode-based combining adapted to the slow variations in channel statistics can yield near-optimum bit error rate (BER) performance, for affordable power budgets.

The paper is organized as follows Section 2 presents the received signal model, and overviews BF, MRC, and MREC. Section 3 describes the Altera software and hardware employed to design, simulate, analyze, and implement these receiver algorithms. Comparative performance and cost results are provided in Section 4.

2. SIGNAL MODEL AND COMBINING METHODS

2.1. Received-signal model

Consider a source transmitting a BPSK signal through a frequency-flat Rayleigh fading channel, and an L-element receiving antenna array. After demodulation, matched filtering, and symbol-rate sampling, the complex-valued received signal vector is given by [4]

$$\widetilde{\mathbf{y}} = \sqrt{E_s}\, b\widetilde{\mathbf{h}} + \widetilde{\mathbf{n}}, \tag{1}$$

where dependence on the sampling time is not explicit, to simplify notation. The L elements $\widetilde{y}_i$, $i = 1 : L \triangleq 1, \dots, L$, of the received signal vector $\widetilde{\mathbf{y}} = [\widetilde{y}_1 \widetilde{y}_2 \cdots \widetilde{y}_L]^{\mathrm{T}}$ are called *branches*, and the elements $\widetilde{h}_i$, $i = 1 : L$, of the channel vector $\widetilde{\mathbf{h}} = [\widetilde{h}_1 \widetilde{h}_2 \cdots \widetilde{h}_L]^{\mathrm{T}}$, are called *channel gains*. In (1), E_s is the energy transmitted per symbol, and b is the transmitted BPSK symbol, with $|b|^2 = 1$ ($b = 1$ for transmitted bit 0, $b = -1$ for transmitted bit 1). We assume that the channel vector $\widetilde{\mathbf{h}}$ and the noise vector $\widetilde{\mathbf{n}}$ are complex-valued, mutually independent, zero-mean Gaussian, with $\widetilde{\mathbf{h}} \sim \mathcal{CN}(\mathbf{0}, \mathbf{R}_{\widetilde{\mathbf{h}}})$ and $\widetilde{\mathbf{n}} \sim \mathcal{CN}(\mathbf{0}, N_0 \mathbf{I}_L)$, respectively. Further assumptions are that channel fading [14] is frequency-flat with unit variance on each branch, the noise is temporally white, and the received signal is interference-free. This signal model is simple, yet sufficient for basic performance evaluations [15]. Current-standard wireless communications signaling is beyond the scope of this work.

2.2. Azimuth angle spread model

Due to radio-wave scattering, transmitted signals are received with azimuthal dispersion [14, 16]. Without loss of generality, numerical results presented herein assume truncated Laplacian power azimuth spectrum (p.a.s.) [4] because it accurately models empirical results [16]. The p.a.s. root second central moment is denoted as *azimuth spread* (AS) [16]. Analytical expressions for the elements of $\mathbf{R}_{\widetilde{\mathbf{h}}}$, obtained through straightforward calculations in [4] for a uniform linear array (ULA), indicate that antenna correlation (and thus receiver BER performance [1, 2]) is a function of p.a.s. type, azimuth spread, average angle of arrival (which is assumed to be zero with respect to the broadside, for all the results shown later), and normalized interelement distance d_n (i.e., the ratio between the physical interelement distance and half of the carrier wavelength).

The azimuth spread depends on the environment and antenna array location/height, and is variable [16]. Radio channel measurements for sub/urban scenarios [16] showed that base station azimuth spread is well modeled as a log-normal random variable [16, equation (9)]. For typical urban scenarios [16, Table I], these measurements found that base-station azimuth spread correlation decreases exponentially with the distance traveled by the mobile [16, equation (14)]. The azimuth spread *decorrelation distance*, that is, the distance over which the azimuth spread correlation decreases by a factor of two, was determined as $d_{\mathrm{AS}} = 50\,\mathrm{m}$ [16]. Comparing d_{AS} with the fading coherence distance [17, equation (4.40.b)] d_c computed for the typical system parameter values from Table 1, we conclude that the azimuth spread variation is much slower (by about 3 orders of magnitude) than the fading. Furthermore, for this typical urban scenario, it was found in [16] that $\Pr(1^\circ < \mathrm{AS} < 20^\circ) \approx 0.8$, that is, azimuth spread is small to moderate, producing significant (greater than 0.5) correlations between adjacent elements of a compact ULA, for example, $d_n = 1$ [1, 3].

2.3. MRC and BF

For perfectly known channel (p.k.c.), the optimum (maximum-likelihood) receiver linearly combines the received signal vector with the channel vector, that is, it computes

Table 1: Mobile, channel, and receiver (channel estimation) parameters.

Parameter	Value
Mobile speed	$v = 60\,\text{km/h}$
Transmitted BPSK symbol rate	$f_s = 10\,\text{ksps}$
Carrier frequency	$f_c = 1.8\,\text{GHz}$
Pilot symbol period [18, Section III.C]	$M_s = 7$
Maximum Doppler frequency	$f_D = 100\,\text{Hz}$
Normalized maximum Doppler frequency	$f_m = f_D/f_s = 0.01$
Channel coherence time [17, equation (4.40.b)]	$T_c \approx 1.8\,\text{ms}$
Channel coherence distance	$d_c = v,$ $T_c \approx 30\,\text{mm}$
Interpolator size [18, Section III.D]	$T = 11$

$\tilde{\mathbf{h}}^{\mathrm{H}}\tilde{\mathbf{y}}$, and then detects the BPSK symbol as

$$\hat{b} = \operatorname{sign}\left[\Re\left(\tilde{\mathbf{h}}^{\mathrm{H}}\tilde{\mathbf{y}}\right)\right]. \tag{2}$$

This approach is also known as maximal-ratio combining (MRC) [19] because it maximizes the SNR (instantaneous, i.e., conditioned on the channel gains) at the combiner's output. MRC with $L = 1$ reduces to the conventional, single-branch, receiver.

In actual systems, with imperfectly known channel (i.k.c.), knowledge of the channel gains is acquired through estimation [1, 18]. The received symbol can then be detected as $\hat{b} = \operatorname{sign}\{\Re[\tilde{\mathbf{g}}^{\mathrm{H}}\tilde{\mathbf{y}}]\}$, where $\tilde{\mathbf{g}} = [\tilde{g}_1\tilde{g}_2\cdots\tilde{g}_L]^{\mathrm{T}}$, and $\tilde{g}_i$, $i = 1 : L$, are the channel gain estimates. This combining approach has often been employed and studied [1, 3, 15, 19], although it is suboptimal (when the channel gains are not independent and identically distributed—non-i.i.d.)[3].

MRC is known to provide full diversity gain [19]—that is, the greatest performance improvement, averaging over fading and noise, compared to a single-branch system—for i.i.d. branches. This requires either widely spaced elements, which are unfeasible for pocketsize mobile stations, or rich scattering, which is unlikely at base stations [16].

For narrow azimuth spread, received signals are highly correlated [1, 2] and the received signal energy, proportional to $\operatorname{tr}(\mathbf{R}_{\tilde{\mathbf{h}}}) \triangleq \sum_{i=1}^{L}(\mathbf{R}_{\tilde{\mathbf{h}}})_{i,i} = \sum_{i=1}^{L}\lambda_i$, where λ_i, $i = 1 : L$, are the eigenvalues of $\mathbf{R}_{\tilde{\mathbf{h}}}$, is concentrated within the first few eigenmodes. Then, the channel is said to be spatially nonselective, and the available diversity gain is small [20–22]. Enhanced performance can then be obtained by taking advantage of antenna gain using maximum average SNR beamforming (BF), that is, by combining the received signal vector with the dominant eigenvector of $\mathbf{R}_{\tilde{\mathbf{h}}}$ [1–4]. Increasing azimuth spread decreases antenna correlation, that is, the channel becomes spatially more selective and higher diversity gain becomes available [1–4]. In subsequent sections, we show how to exploit available antenna and diversity gains within complexity and power constraints.

2.4. Eigencombining method

BF has traditionally been applied in scenarios with very small azimuth spread. Otherwise, MRC has been employed. However, it was recently claimed that a unifying approach, called maximal-ratio eigencombining (MREC), and described below, can adapt to channel correlation (i.e., azimuth spread) variation [1–4, 20]. Our analytical and simulation results have shown that MREC may thus outperform MRC and BF in terms of BER performance and complexity [1–4].

The channel correlation matrix $\mathbf{R}_{\tilde{\mathbf{h}}}$ has real nonnegative eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_L \geq 0$, orthonormal eigenvectors $\mathbf{e}_i$, $i = 1 : L$, and can be decomposed as $\mathbf{R}_{\tilde{\mathbf{h}}} = \mathbf{E}_L\mathbf{\Lambda}_L\mathbf{E}_L^{\mathrm{H}}$, where $\mathbf{\Lambda}_L \triangleq \operatorname{diag}\{\lambda_i\}_{i=1}^{L}$ is a diagonal matrix, and $\mathbf{E}_L \triangleq [\mathbf{e}_1\mathbf{e}_2\cdots\mathbf{e}_L]$ is a unitary matrix. Hereafter, $\mathbf{R}_{\tilde{\mathbf{h}}}$, $\mathbf{\Lambda}_L$, and $\mathbf{E}_L$ are assumed perfectly known because, in practice, enough independent channel samples would be available for an accurate estimation. Actual MREC could employ computationally insignificant low-rate eigenstructure updating [20].

MREC of order N consists of the following steps [1–4]:

(i) Karhunen-Loève transformation (KLT) [22] of the received signal vector from (1) with the full-column rank matrix $\mathbf{E}_N \triangleq [\mathbf{e}_1\mathbf{e}_2\cdots\mathbf{e}_N]$; the elements of the transformed signal vector, $\mathbf{y} = \mathbf{E}_N^{\mathrm{H}}\tilde{\mathbf{y}} = \sqrt{E_s}b\mathbf{E}_N^{\mathrm{H}}\tilde{\mathbf{h}} + \mathbf{E}_N^{\mathrm{H}}\tilde{\mathbf{n}} = \sqrt{E_s}b\mathbf{h} + \mathbf{n}$, are denoted as *eigenbranches*;

(ii) MRC of the N eigenbranches.

The components of the transformed channel gain vector $\mathbf{h} = \mathbf{E}_N^{\mathrm{H}}\tilde{\mathbf{h}}$ are further referred to as channel *eigengains*. They are mutually uncorrelated, with zero mean, and variances $\sigma_{h_i}^2 \triangleq E\{|h_i|^2\} = \lambda_i$, that is, $\mathbf{R}_{\mathbf{h}} \triangleq E\{\mathbf{h}\mathbf{h}^{\mathrm{H}}\} = \mathbf{\Lambda}_N = \operatorname{diag}\{\lambda_i\}_{i=1}^{N}$, for any channel gain distribution [21]. From the initial assumptions on fading and noise we obtain $\mathbf{h} \sim \mathcal{CN}(\mathbf{0}, \mathbf{\Lambda}_N)$, and $\mathbf{n} = \mathbf{E}_N^{\mathrm{H}}\tilde{\mathbf{n}} \sim \mathcal{CN}(\mathbf{0}, N_0\mathbf{I}_N)$, so that the eigengains are independent, which supports straightforward MREC analysis [1–4].

Of all possible transforms, the KLT packs the largest amount of energy from the original, L-dimensional signal vector $\tilde{\mathbf{y}}$ into the transformed, N-dimensional signal vector $\mathbf{y}$ [22], which is desirable for dimension (i.e., complexity) reduction. Note also that MREC of order $N = 1$ represents in fact BF, while it can be shown that full-MREC, that is, MREC of order $N = L$, is equivalent to MRC [1–4].

2.5. Order selection for MREC

A simple criterion for optimal MREC order selection is [21]

$$\min_{N=1:L}\left[E_s \cdot \sum_{i=N+1}^{L}\lambda_i + N_0 \cdot N\right], \tag{3}$$

better known as the *bias-variance tradeoff criterion* [3, 4] (BVTC) because (3) balances the loss incurred by removing the weakest $(L - N)$ intended-signal contributions (the first term) against the residual-noise contribution (the second term). Computer evaluations found the BVTC effective for MREC adaptation to channel conditions [3, 4]. Note

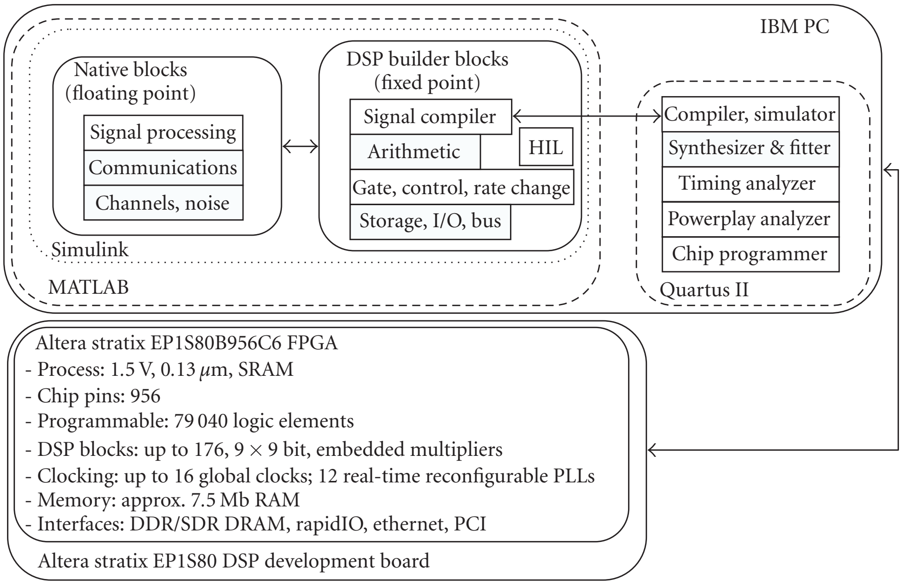

IBM PC
Native blocks
(floating point)
Signal processing
Communications
Channels, noise
DSP builder blocks
(fixed point)
Signal compiler
Arithmetic
HIL
Gate, control, rate change
Storage, I/O, bus
Simulink
MATLAB
Compiler, simulator
Synthesizer & fitter
Timing analyzer
Powerplay analyzer
Chip programmer
Quartus II
Altera stratix EP1S80B956C6 FPGA
- Process: 1.5 V, 0.13 μm, SRAM
- Chip pins: 956
- Programmable: 79 040 logic elements
- DSP blocks: up to 176, 9 × 9 bit, embedded multipliers
- Clocking: up to 16 global clocks; 12 real-time reconfigurable PLLs
- Memory: approx. 7.5 Mb RAM
- Interfaces: DDR/SDR DRAM, rapidIO, ethernet, PCI
Altera stratix EP1S80 DSP development board

Figure 1: FPGA development system hardware/software diagram.

however that since BVTC disregards the MREC complexity, it can overload limited resources.

A different MREC adaptation criterion is described next. Assume that signals received (independently) from N_u mobile stations require processing at a base station with only $N_e \ll N_u L$ available eigenbranch processing modules. Then, a control algorithm determines the largest (dominant) N_e eigenmodes among all transmitting mobiles, and allocates available resources accordingly. For instance, if a receiving antenna array system with $L = 4$ elements has only $N_e = 3$ available eigenbranch processing modules while $N_u = 2$, the available resources are allocated as follows: if the 3 largest eigenvalues (out of $N_u L = 8$) are such that two correspond to User 1, and one to User 2, then two eigenbranch processing modules are allocated to process the received signal vector from User 1, and the other available eigenbranch is allocated to User 2. This approach to selecting eigenbranches for MREC is hereafter denoted as the *eigenvalue-based tradeoff criterion* (EVTC), while MREC adapted based on EVTC is referred to as EVTC MREC.

2.6. Channel estimation using pilot-symbol-aided modulation (PSAM)

In PSAM, the transmitter periodically inserts known pilot symbols b_p of energy E_p ($= E_s$ for results shown herein), into the information-encoding symbol stream, and the receiver interpolates the pilot samples acquired across several slots to estimate the channel during data symbols [1–4, 18]. The notation (t, m) is used below to denote temporal indexing, where $t = -T_1 : T_2$ is the time slot index, and $m = 0 : M_s - 1$ is the symbol index within the slot of length M_s. Here $t = 0$ refers to the slot in which estimation takes place, $m = 0$ corresponds to pilot symbols, and $m = 1 : M_s - 1$ corresponds to data-encoding symbols; $T = T_1 + T_2 + 1$ slots (in general, $T_1 = T_2$) are used for interpolation.

The estimate of the ith eigengain at the mth data symbol position in the current slot can be written as

$$g_i(0, m) = \mathbf{v}_i^{\mathrm{H}}(m)\mathbf{r}_i, \tag{4}$$

where $\mathbf{v}_i(m)$ is the interpolation filter and

$$\mathbf{r}_i \triangleq \frac{1}{\sqrt{E_p} b_p}\left[y_i(-T_1, 0), \ldots, y_i(T_2, 0)\right]^{\mathrm{T}} \tag{5}$$

contains the samples taken during pilot symbols.

The interpolation filter chosen for the numerical results shown later is the filter with brick-wall-type frequency response, which is optimum in the absence of noise; we will refer to this filter, with impulse-response tapered by a raised-cosine window [1, 2], as the SINC filter, and the corresponding estimation approach as SINC PSAM. The interpolator coefficients, given by

$$\left[\mathbf{v}(m)\right]_{t+T_1+1} = \operatorname{sinc}\left(\frac{m}{M} - t\right)\frac{\cos[\pi\beta(m/M - t)]}{1 - [2\beta(m/M - t)]^2}, \tag{6}$$

enter the FPGA-based receiver designs from Section 4. Note that channel estimation is among the most demanding receiver functions resource-wise [5].

3. FPGA HARDWARE AND SOFTWARE

3.1. FPGA system description

CMC Microsystems provided the system shown in Figure 1. The Altera DSP Development Kit Stratix Professional Edition, which comprises the Stratix EP1S80 DSP development board, is built around the Stratix EP1S80B956C6 FPGA chip, and comes with the DSP Builder interface to the Quartus II design flow.

Quartus II provides a comprehensive design, synthesis, and analysis environment for system-on-a-programmable-chip (SoPC) applications. DSP Builder helps, create the hardware representation of the required digital signal processing functions using the MATLAB and Simulink user-friendly algorithm-development environments, for shorter design and implementation cycles. MATLAB functions and native Simulink blocks can be combined with Altera DSP Builder library blocks (see Figure 1) to create FPGA designs which can be simulated under Simulink. For automated design flow, the "signal compiler" block, which is at the core of DSP Builder, can generate hardware description language (HDL) code, and scripts for Quartus II-based synthesis and fitting from within Simulink. Furthermore, the DSP Builder "hardware in the loop" (HIL) block enables chip programming for hardware-software cosimulation.

3.2. Power usage considerations

Power loss in FPGA devices can be categorized as static and dynamic [10–13]. Static (standby) power is consumed by the chip when no input signals are exercised [10]. This loss occurs due to transistor leakage, which is frequency-independent, but highly dependent on junction temperature and transistor size. Static power has been increasing (exponentially, at processes below 0.25 μm [11]) with each finer semiconductor technology, to become the dominant loss component in current chips. This is a concern for designers of portable embedded systems which spend long intervals in standby mode [10]. Dynamic power is consumed in normal operation, due to the charging and discharging of the internal capacitive loads, and is proportional to gate output load, square of the supply voltage, clock frequency, and gate switching activity [10–13]. Although the supply voltage has decreased significantly in newer process technologies, high operating frequencies can still yield significant dynamic power losses [10]. A tight power budget may thus limit clock speed.

Line-powered embedded systems are more competitive when they require less expensive power supplies and cooling devices [10]. Designs for portable products should aim for the longest possible battery life. Moreover, devices operating at high temperatures can become unreliable, emphasizing the importance of minimizing power consumption in embedded systems. FPGA structure is judiciously designed to minimize power losses [10–12, 23]. Nonetheless, power-aware application design can also increase efficiency, for example, by using gated clock signals, and thus virtually turning off unnecessary chip sections [10, 12, 23]. Gating as close as possible to the clock source is a good practice since clock signal trees are important dynamic power consumers [12]. On the other hand, static power consumption can be reduced by adaptive distribution of available FPGA resources, as shown in Section 4.3.

For the designs described further below, we relied on Quartus II reports on resource usage, for example, the number of logic elements (LEs), chip pins, and dedicated 9×9-bit DSP blocks. Static and dynamic power losses were estimated using the Quartus II Powerplay analyzer (dynamic power was estimated for default toggle rates of 12.5%).

4. FPGA-BASED WIRELESS COMMUNICATIONS RECEIVERS

For the system shown in Figure 1, we focus on FPGA-based receiver algorithm implementation, assuming availability of digitized received signals. The transmitted signal and channel/receiver impairments, that is, noise and temporally and spatially correlated fadings, are generated in MATLAB and Simulink. Various receiver algorithms were simulated and run from the FPGA, through DSP Builder HIL. Computer simulations and the corresponding hardware/software HIL co-simulations were found to perform identically. Computations done in MATLAB or with native Simulink blocks are very precise, due to floating-point number representation. On the other hand, DSP Builder relies on fixed-point representation, which can limit the dynamic range and can introduce quantization noise.

As mentioned earlier in Table 1, we consider a scenario with Doppler spread $f_D = 100$ Hz and transmission rate $f_s = 10$ ksps, that is, normalized Doppler spread $f_m = 0.01$ Hz. PSAM with slot length $M_S = 7$ (1 pilot symbol followed by 6 information-encoding symbols) is combined with SINC interpolation over $T = 11$ slots ($T_1 = T_2 = 5$), for channel estimation as in (4)–(6). ULA with $d_n = 1$ is assumed to provide the received signals for the enhanced receivers.

4.1. Conventional, single-branch versus enhanced, multibranch MRC receivers

In this section, a conventional, single-branch receiver, and an enhanced MRC receiver, with $L = 2$ i.i.d. branches, are considered. We employ the well-established Jakes' model [14] for temporal channel fading correlation, with parameters given in Table 1. For BPSK, receiver BERs were computed for perfectly known channel (p.k.c.), as well as imperfectly known channel (i.k.c.) for SINC PSAM. We verified that BER expressions derived in [1] and the corresponding MATLAB simulation results agree closely for p.k.c. as well as for i.k.c. Then, for i.k.c., FPGA-based designs were simulated as well as hardware-software (HIL) cosimulated. For HIL cosimulation, the receiver design is compiled and then downloaded into the FPGA chip. Afterwards, received signals emulated using MATLAB are processed online by the programmed FPGA. In terms of numerical representation precision within the FPGA for the computer-generated received signal $\widetilde{\mathbf{y}}$, two cases are compared next: (1) 8 bits for the integer part and 8 bits for the fractional part (denoted further as 8.8); (2) the 4.4 case. Finally, the channel gain estimation root mean-square error (RMSE) is determined from theory [4], simulations, and HIL implementations.

The upper part of Figure 2 shows the Simulink/DSP Builder design involved in channel gain estimation for one branch, while the lower part details our "SINC interpolator"

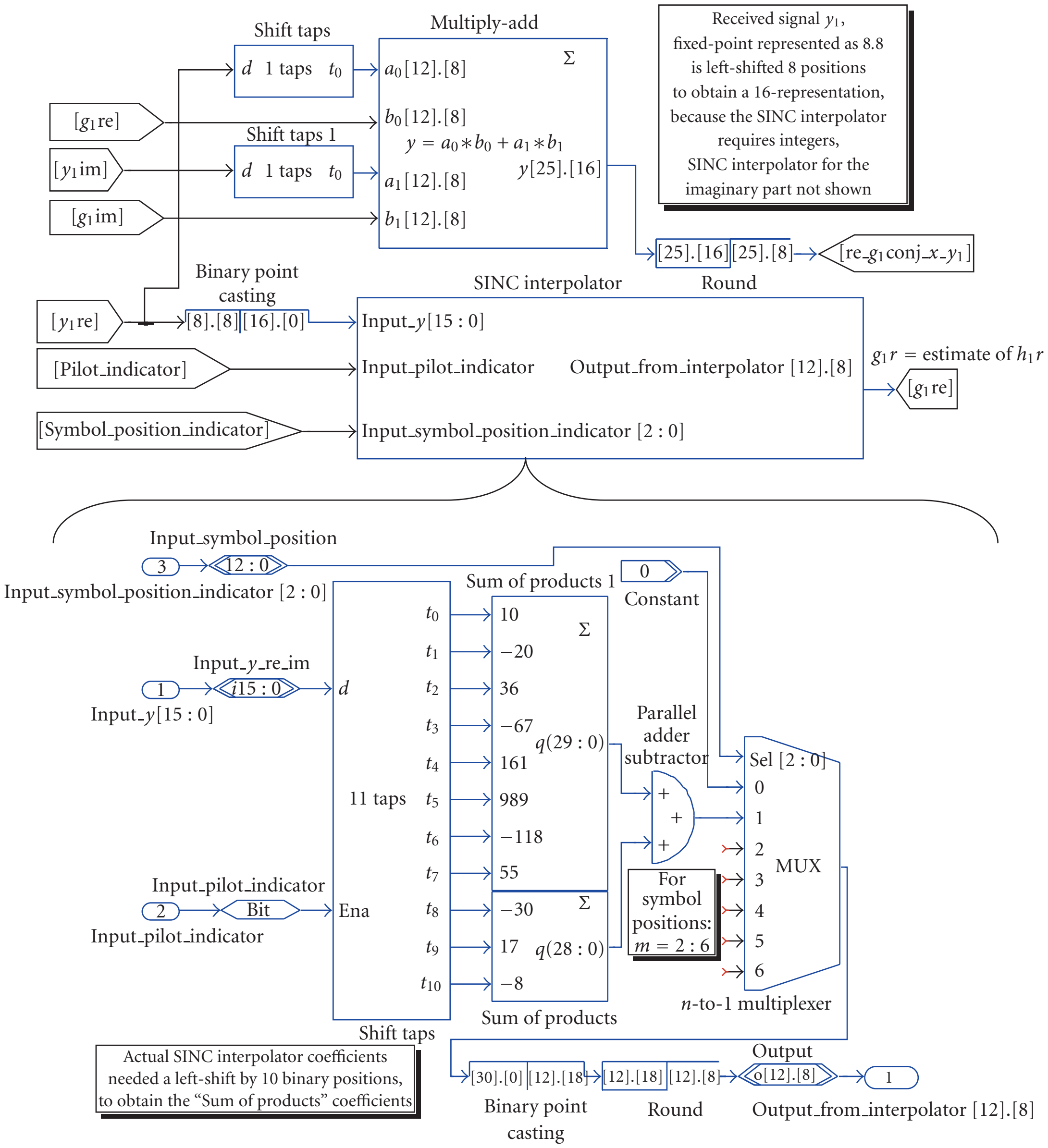
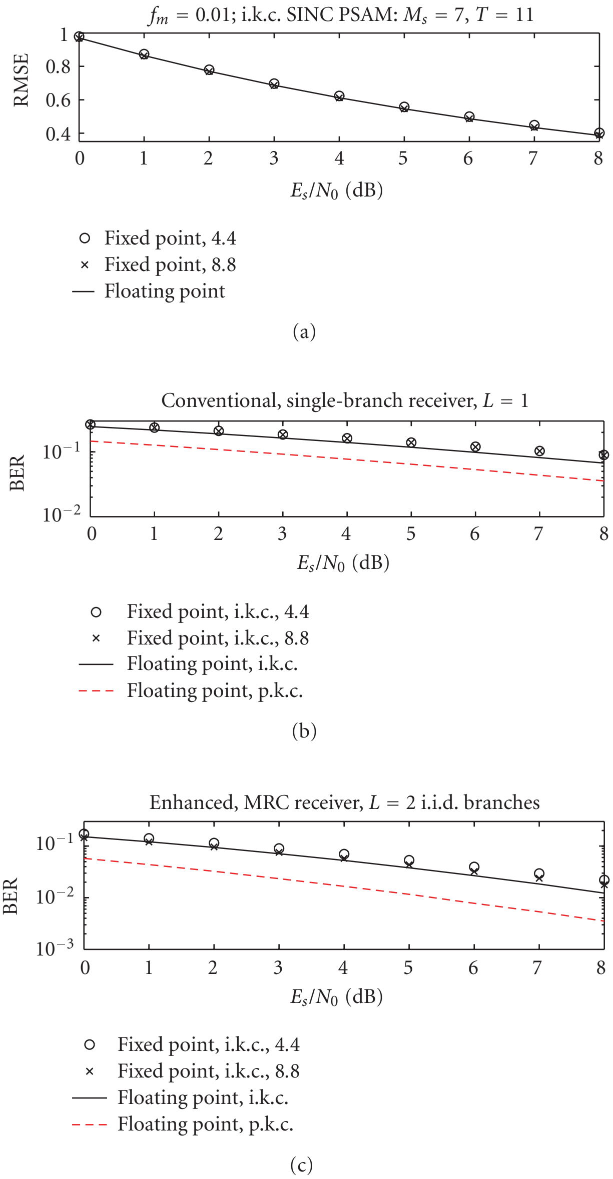


Figure 2: Simulink model detail with DSP Builder blocks implementing channel gain estimation (through SINC interpolation) for MRC.

design. (Symbols appear without the tilde due to Simulink editing limitations.) The upper "shift taps" DSP Builder blocks delay the received signal by $(T_1 + 1)M_s = 42$ samples, while the "multiply-add" block computes $\Re(\widetilde{g}_1^* \ \widetilde{y}_1)$, used as test variable for symbol detection. Since the DSP Builder blocks "sum of products" in the "SINC interpolator" design require integer input and coefficients, binary shifting of the received signal and interpolator coefficients (computed from [1, Table 1]) is required. The "SINC interpolator" "shift taps" block outputs $\Re(\widetilde{\mathbf{r}}_1)$, see (5), while the "parallel Adder/Subtractor" outputs $\Re(\widetilde{g}_1)$—see (4). The interpolator output is then used for combining. Notice that channel estimation can be very demanding resource-wise, especially for multibranch receivers.

The RMSE subplot in Figure 3 indicates that 4.4 and 8.8 fixed-points FPGA computation does not visibly degrade channel estimation accuracy compared to floating-point (computer) computation. Nevertheless, the lower subplots show that fixed-point computation with narrow word (i.e., poor precision, narrow dynamic range) can significantly degrade BER performance, an effect which cumulates with more branches.

Figure 3 also indicates that the performance degradation (i.e., about 3.4 dB) which occurs for a conventional receiver due to i.k.c. can be successfully compensated for an FPGA-based dual-branch MRC, due to its diversity gain. Confidence intervals for all these results are very tight, since 10 000 slots, that is, 60, 000 data symbols, were detected.



(a)

(b)

(c)

Figure 3: (a) RMSE for channel gain estimates. (b) and (c) Performance of the conventional, single-branch receiver, and of the dual-branch MRC receiver for various computer- and FPGA-based implementations. Fixed-point results correspond to both DSP Builder-based simulations and HIL implementations.

Table 2: Resource usage for 8.8 implementations of MRC, BF, and adaptive MREC, for up to $L = 4$ branches.

Method	LEs	Pins	DSP	CF	DP
	(79 040)	(692)	(176)	(MHz)	(mW)
MRC	13,227	43	16	41.06	69.35
$L = 1$	16.73%	6.21%	9.09%		
MRC	26,478	83	32	38.56	119.67
$L = 2$	33.49%	11.99%	18.18%		
MRC	39,731	123	48	38.35	169.78
$L = 3$	50.27%	17.77%	27.27%		
MRC	55,983	167	64	36.74	221.62
$L = 4$	70.83%	24.13%	36.36%		
BF	13,457	259	48	40.57	74.95
$L = 4$	17.02%	37.43%	27.27%		
BVTC MREC	13,458	262	48	41.15	74.95
$L = 4, N = 1$	17.02%	37.86%	27.27%		
BVTC MREC	26,940	358	96	39.73	130.89
$L = 4, N = 2$	34.08%	51.73%	54.54%		
BVTC MREC	40,423	454	144	39.09	186.64
$L = 4, N = 3$	51.14%	65.60%	81.81%		
BVTC MREC	55,847	550	176	38.82	244.64
$L = 4, N = 4$	70.66%	79.48%	100%		
EVTC MREC	13,561	424	48	41.09	75.67
$L = 4, N = 1$	17.16%	61.27%	27.27%		
EVTC MREC	27,372	524	96	39.14	132.95
$L = 4, N = 2$	34.63%	75.72%	54.54%		
EVTC MREC	40,983	624	144	35.43	189.23
$L = 4, N = 3$	51.85%	90.17%	81.81%		

For designs shown hereafter, we settled for an 8.8-representation, since it was found to offer a fair compromise between representation accuracy/dynamic range (i.e., receiver performance) and FPGA resource utilization. Furthermore, we instructed DSP Builder to allocate hard-wired DSP circuitry embedded into the reconfigurable FPGA fabric, which yields effective and efficient chip utilization [7]. Then, Quartus II reports on FPGA resource usage, maximum allowable clock frequency (CF), and dynamic power (DP) usage, as shown in Table 2. Estimated static power loss is 1.395 W. Note that for the BER advantage shown in Figure 3 over the conventional receiver, dual-branch MRC nearly doubles resource requirements and dynamic power loss. Since the MRC performance gradient diminishes with increasing number of branches [4], implementation/operational costs can be minimized either with tightly matched chips, or through clock gating of excess resources.

In the above MRC receiver design, channel gains on different branches were considered statistically independent, for simplicity. However, this is rarely the case in practice [16]. Although scattering is richer around the mobile than around the base station, mobile antenna array size limitations can still lead to large interbranch correlation, that is, scarce diversity gain availability. Then, adaptive MREC [3, 4] may provide more suitable tradeoffs between performance and resource/power utilization, as shown next.

4.2. *Enhanced MREC receiver designs: the case of a single user processed per FPGA chip*

We extended the previously discussed FPGA-based MRC receiver design to support $L = 4$ branches, and also designed the BF, and the BVTC adaptive MREC receivers. See Table 2

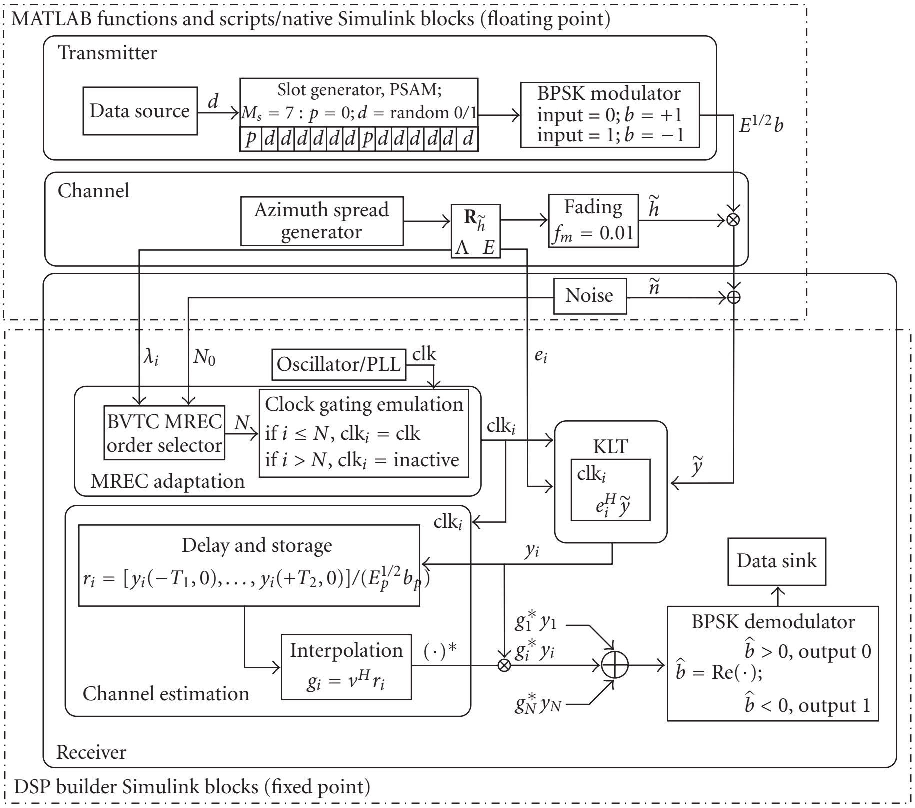


Figure 4: Transmitter, channel, and FPGA-based BVTC MREC receiver diagram.

for the resource and power usage report. Note that a standalone BF implementation takes about as many resources as order-1 MREC takes in the BVTC MREC implementation since these two designs are almost identical. Furthermore, MRC can be obtained from an MREC design by bypassing the KLT. Thus, an MREC design can easily be reconfigured (even during operation, on the fly) to implement BF or MRC instead. Implementation details are provided in Figure 4, for the case when the receiver implements BVTC adaptive MREC.

For resource/power usage and performance evaluation, we model a typical urban scenario for realistic channel conditions from the base station perspective [16], and apply the conventional and enhanced receiver combining algorithms (after estimating channel gains and eigengains as in Section 2.6) to detect the transmitted symbols. Using MATLAB and Simulink, the actual log-normal distributed, time-correlated azimuth spread is simulated and then employed to compute the spatial correlation matrix, for realistic Laplacian power azimuth spectrum (p.a.s.) [16]—see Figure 4. In an actual embedded receiver, the channel correlation matrix and its eigenvalue decomposition could be updated by a processor (e.g., Altera's soft-core FPGA-based Nios II). We selected a correlation update period of 0.14 second (denoted further as a *frame*, corresponding to a distance of roughly 2.3 m traveled by the mobile) since the azimuth spread remains relatively constant over this interval [16], providing the processor with sufficient time and uncorrelated samples for eigenstructure updating [3, 4]. The computed correlation matrix $\mathbf{R}_{\tilde{\mathbf{h}}}$ inputs a customized Simulink "multipath Rayleigh fading channel" block to simulate $L = 4$ correlated branches.

The top subplot in Figure 5 depicts an azimuth spread sequence generated using the model described in Section 2.2. The predominantly small-to-moderate azimuth spread values indicate that we should often expect significant spatial correlation [1, 3], that is, small available diversity gain. Performance enhancement can then arise from BF antenna gain. Occasionally however, the azimuth spread can also become fairly large, but then the available diversity gain cannot benefit BF performance. On the other hand, significant diversity gain may be available too infrequently to justify permanent use of an MRC receiver. As we will see, an FPGA-based MREC receiver can provide, for a channel with slowly varying statistics, flexibility that yields affordable performance.

The main benefit of an FPGA-based BVTC adaptive MREC receiver is that unnecessary eigenbranches can be virtually turned off using the clock gating technique [12] to reduce dynamic power loss, while necessary eigenbranches can be implemented to run in parallel, for high speed. Exempting weak eigenbranches can also benefit performance [1]. Furthermore, as mentioned earlier, an MREC implementation can easily be reduced to standalone BF or MRC implementations, if required, either at system setup or during operation.

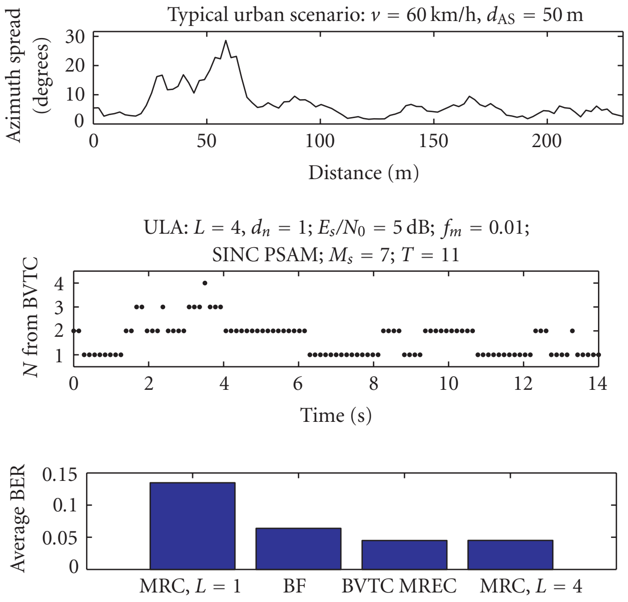


Figure 5: Azimuth spread, MREC order selected with the BVTC, and BER performance (averaging over trial) for BF, MRC, and BVTC MREC.

Altera documentation states that clock gating is available only through lower-level (Quartus II) design. Therefore, clock gating was only emulated in DSP Builder, for the BVTC MREC implementation shown in Figure 4. First, nonadaptive MREC designs with $N = 1 : 4$ eigenbranches were compiled to determine their resource usage (shown in Table 2). Then, after each eigenstructure update during the BVTC MREC simulation, we stored the selected MREC orders and disconnected unused eigenbranches from the active structure. Finally, average resource usage was computed. Figure 5 shows in the middle subplot the MREC order selected adaptively using the BVTC, and in the lower subplot the BER averaged over the trial. Notice that for $L = 4$, MRC and BVTC adaptive MREC slightly outperform BF, and greatly outperform the single-branch receiver.

For the same typical urban scenario and system parameters, Figure 6 shows resource usage, in percentage points of the total available, and dynamic power consumption, averaged over 8 trials. In each trial, the azimuth spread samples are correlated, as described in Section 2.2, but the azimuth spread sequences are independent between trials. Note that BF and BVTC MREC require a significantly smaller share of the FPGA programmable fabric, that is, LEs, compared to MRC (for $L = 4$), but more dedicated DSP blocks, due to KLT. The upper-right subplot appears to imply more chip pins demand for BF and MREC, because a MATLAB/Simulink-computed eigenvector matrix $\mathbf{E}_N$ inputs the FPGA. Nevertheless, eigenstructure updating is possible with a soft processor, from within the FPGA.

Figure 7 shows performance and total (dynamic + static) power used by a cellular operator's large network of base stations similar to the one described in [11]. The single-branch receiver consumes least but performs poorly. For performance similar to BF and BVTC MREC, MRC (with $L = 4$) doubles the dynamic power loss (see also Figure 6(d)). Thus, BF and BVTC MREC appear to provide a better tradeoff. Recall however that a compact ULA with $d_n = 1$ is considered. For larger interelement distances (feasible at base stations), MREC with more than one eigenbranch can significantly outperform BF [4].

Note that significant branch correlation can occur even at mobile stations, due to limited antenna spacing, so that an FPGA-based BVTC MREC implementation employing clock gating can efficiently achieve near-optimum performance.

Notice from Figure 5(b) that, frequently, only one or two (out of the four implemented) eigenbranches were actually employed for MREC for that particular azimuth spread sequence. Similar results were obtained in other trials for independent azimuth spread sequences. This suggests that adaptive FPGA chip resource allocation among several active users may significantly increase base station user processing capacity, or, equivalently, reduce the required number of FPGA chips per base station, lowering both hardware cost and static power losses. A possible path towards such implementations is described next.

4.3. Enhanced MREC receiver designs: the case of two users processed per FPGA chip

EVTC-based adaptive MREC, described in Section 2.5, can provide more consistent use of the FPGA chip, compared to BVTC MREC. We propose to efficiently exploit a total of 3 eigenbranch processing modules, which fit into our FPGA, to process concurrently the signals received with $L = 4$ branches from two mobiles (without interference). Rather than permanently allotting chip processing resources to a certain user (which may or may not need to use them, depending on channel conditions and required performance), herein we will adaptively deploy these resources to simultaneously detect the symbols transmitted from two mobiles.

Resource usage information for EVTC MREC when $N = 1 : 3$ eigenbranches are selected can be found in Table 2. Note that the BVTC and EVTC MREC implementations differ significantly only in the required number of chip pins. The larger number of pins required for EVTC MREC (to input the received signals from two mobiles) limits to 3 the possible number of implemented eigenbranches. Larger N_e leads to unsuccessful compilation. Mutually independent azimuth spread sequences for the signals arriving at the base station from the two mobile stations were simulated, as shown in the top subplots of Figure 8. The MREC orders selected with the EVTC for each of the users are shown in the middle subplots. The lower subplots indicate that EVTC MREC can perform remarkably close to the enhanced receivers discussed previously.

Figure 9(a) indicates that our FPGA would not fit concurrent four-branch MRC implementations for the two users. On the other hand, the successfully compiled two-user EVTC MREC implementation with $N_e = 3$ requires about half of the dynamic power consumed by MRC, for similar

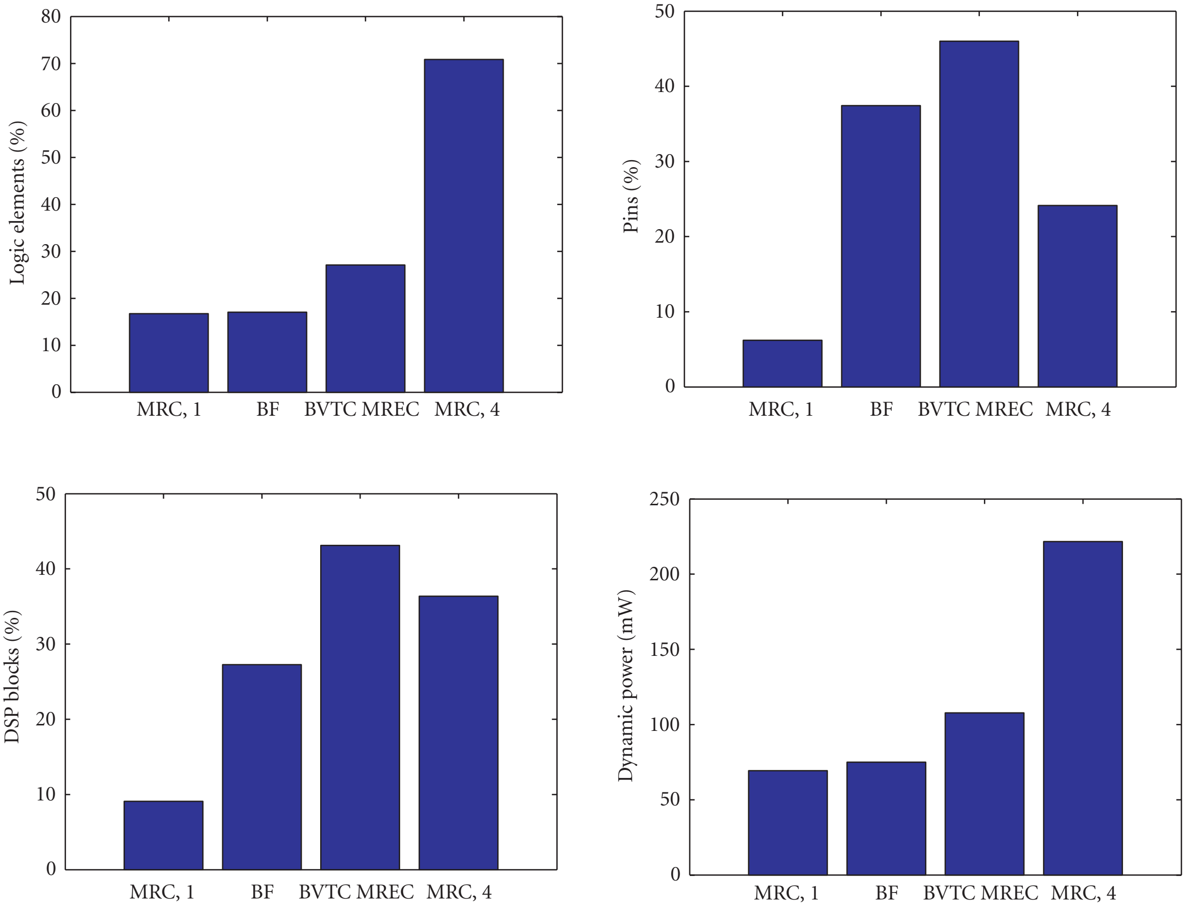


Figure 6: Average resource and dynamic power usage for BF, BVTC MREC, and MRC, over 8 trials with mutually independent azimuth spread sequences.

performance. Furthermore, since EVTC MREC allows for effective concurrent processing of two users on a single FPGA, it yields a twofold reduction in static power consumption or a doubling of the base station user processing capacity. Thus, both implementation and operational costs can be drastically reduced with EVTC MREC.

Ideally, an FPGA-based embedded base station receiver would comprise: (1) a number of FPGAs programmed for KLT, channel estimation, signal combining, and symbol detection; (2) an embedded processor monitoring each user's channel conditions (i.e., eigenmodes). At the beginning of each frame, the embedded processor browses a user hierarchy, and allocates the FPGA resources so as to achieve desired performance for minimum resource/power consumption [3, 4]. Thus, it is possible that for a certain period, several users whose respective received signals are highly correlated will share the resources of a single FPGA because none of them will demand a large number of eigenbranches. If the azimuth spread for one of these users later widens significantly (yielding more available diversity gain) or if its SNR degrades (while a certain steady performance level is imposed), a larger share of the FPGA resources can be allocated accordingly. An FPGA-based embedded system for a performance- and a power-aware antenna array receivers can thus be flexibly implemented.

5. CONCLUSIONS

We have described and implemented adaptive techniques that enhance the performance and reduce the power consumption for Altera-FPGA-based embedded wireless receivers. We found that smart antenna array receiver algorithms, for example, beamforming (BF) and maximal-ratio combining (MRC), outperform the conventional, single-branch receiver, but the performance gain may not always justify the additional implementation and operational costs. Tracking the slowly varying dominant channel eigenmodes, and using maximal-ratio eigencombining (MREC) is found to benefit more than BF and MRC from the parallelism and flexibility of FPGA-based implementation. For similar performance, a twofold increase in user processing capacity or decrease in power consumption is found possible over MRC, for a typical urban scenario and 4 receiving antennas. Adaptive MREC outperforms BF, for slightly

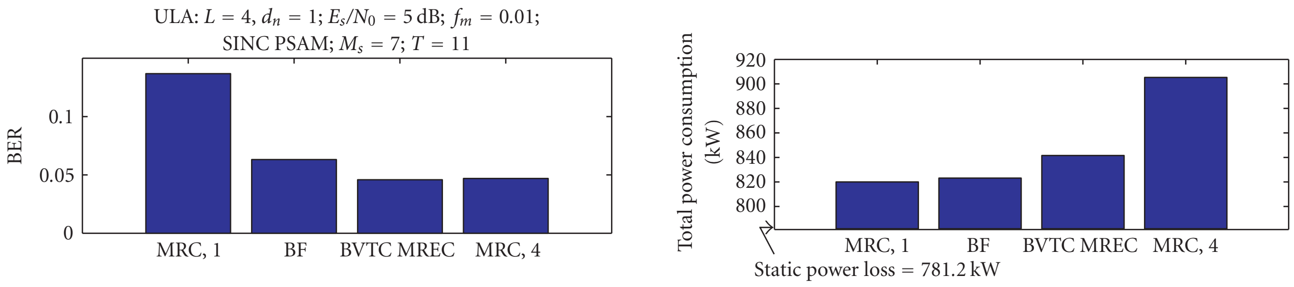


FIGURE 7: Average BER and total (static + dynamic) power consumption for BF, BVTC MREC, and MRC, over 8 independent azimuth spread trials.

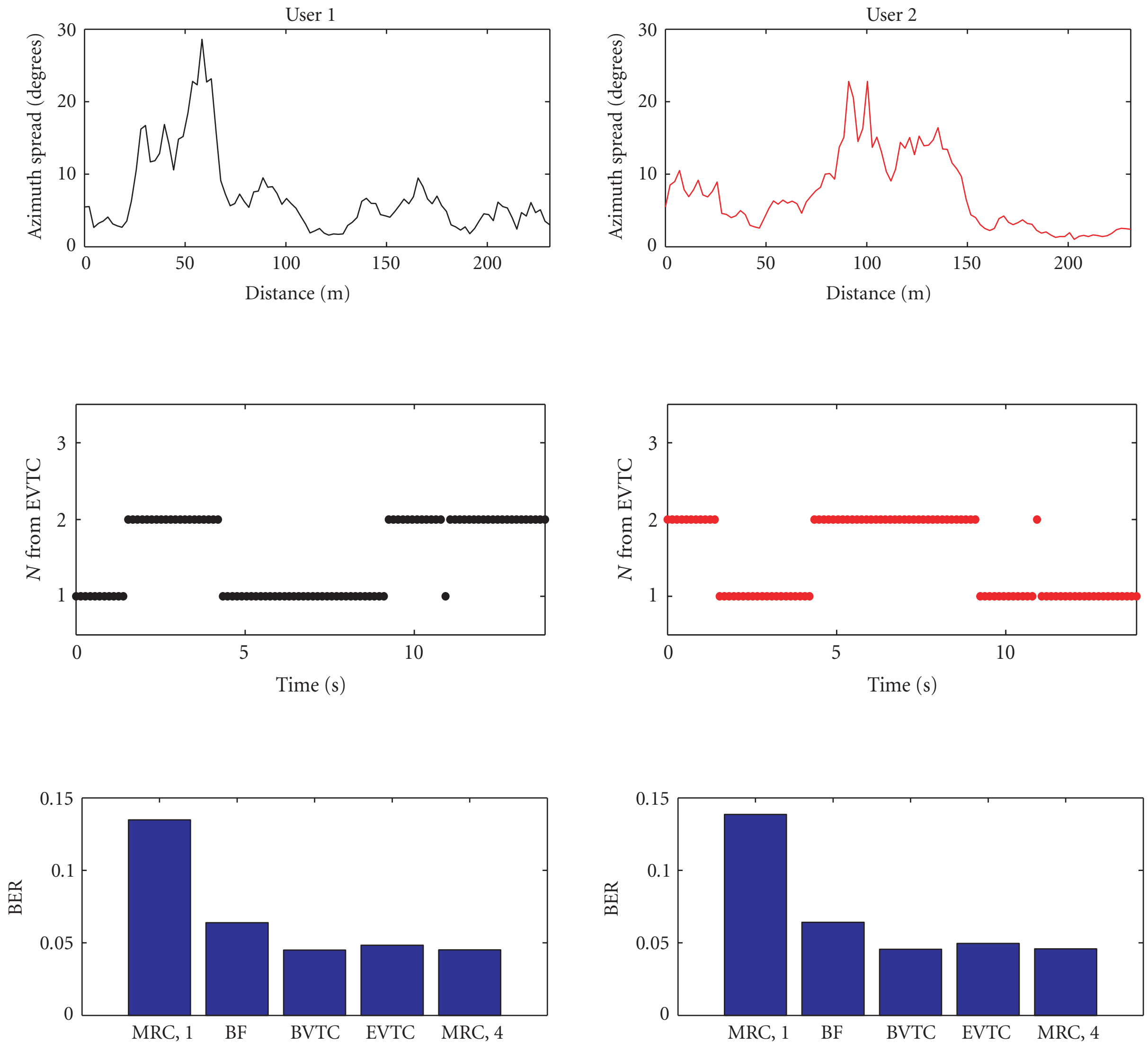


FIGURE 8: Azimuth spread, EVTC MREC order, and average BER performance, for two users.

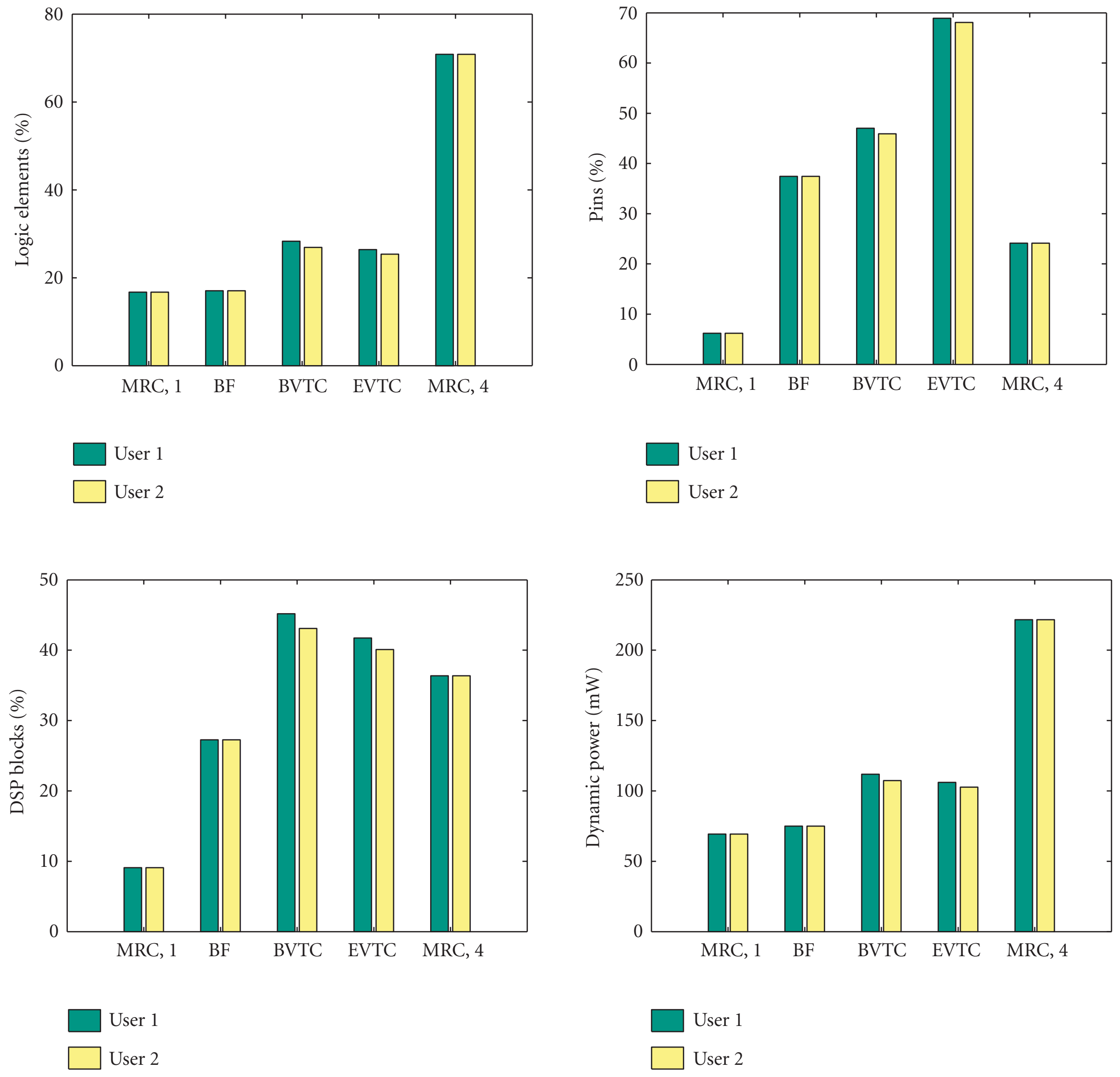


Figure 9: Resource usage (in percentage of total available) and dynamic power consumption for all discussed receiver algorithms, for two independent users.

higher resource consumption. FPGA flexibility and wide range of on-chip resources can thus yield very efficient embedded implementations of adaptive receivers for current and future generations of wireless communications systems.

ACKNOWLEDGMENT

The Altera Stratix FPGA development board, DSP Builder software tool, and Quartus II environment were provided by CMC Microsystems (www.cmc.ca) as part of the System-on-Chip Research Network infrastructure available to researchers at Canadian universities.

REFERENCES

[1] C. Siriteanu and S. D. Blostein, “Maximal-ratio eigencombining: a performance analysis,” *Canadian Journal of Electrical and Computer Engineering*, vol. 29, no. 1, pp. 15–22, 2004.

[2] C. Siriteanu and S. D. Blostein, “Smart antenna arrays for correlated and imperfectly-estimated Rayleigh fading channels,” in *Proceedings of IEEE International Conference on Communications (ICC ’04)*, vol. 5, pp. 2757–2761, Paris, France, June 2004.

[3] C. Siriteanu and S. D. Blostein, “Maximal-ratio eigencombining for smarter antenna arrays,” to appear in *IEEE Transactions on Wireless Communication*.

[4] C. Siriteanu, “Maximal-ratio eigen-combining for smarter antenna arrays,” Ph.D. dissertation, Queen’s University, Kingston, ON, Canada, September 2006.

[5] M. Guillaud, A. Burg, M. Rupp, E. Beck, and S. Das, “Rapid prototyping design of a 4×4 BLAST-over-UMTS system,” in *Proceedings of Conference Record of the 35th Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1256–1260, Pacific Grove, Calif, USA, November 2001.

[6] B. L. Hutchings and B. E. Nelson, “Gigaop DSP on FPGA,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP ’01)*, vol. 2, pp. 885–888, Salt Lake City, Utah, USA, May 2001.

[7] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, “Implementing an OFDM receiver on the RaPiD reconfigurable architecture,” *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1436–1448, 2004.

[8] F. L. Vargas, R. D. R. Fagundes, and D. Barros Jr., “A FPGA-based Viterbi algorithm implementation for speech recognition systems,” in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP ’01)*, vol. 2, pp. 1217–1220, Salt Lake City, Utah, USA, May 2001.

[9] T. S. T. Mak and K. P. Lam, “Embedded computation of maximum-likelihood phylogeny inference using platform FPGA,” in *Proceedings of IEEE Computational Systems Bioinformatics Conference (CSB ’04)*, pp. 512–514, Stanford, Calif, USA, August 2004.

[10] S. Sharp, “Conquering the three challenges of power consumption,” *XCell Journal*, no. 53, 2005.

[11] A. Telikepalli, “Performance vs. power: getting the best of both worlds,” *XCell Journal*, no. 54, 2005.

[12] L. Benini, G. De Micheli, and E. Macii, “Designing low-power circuits: practical recipes,” *IEEE Circuits and Systems Magazine*, vol. 1, no. 1, pp. 6–25, 2001.

[13] A. Yang, “Design techniques to reduce power consumption,” *XCell Journal*, no. 54, 2005.

[14] W. C. Jakes, Ed., *Microwave Mobile Communications*, John Wiley & Sons, New York, NY, USA, 1974.

[15] J. Proakis, *Digital Communications*, McGraw-Hill, Boston, Mass, USA, 4th edition, 2001.

[16] A. Algans, K. I. Pedersen, and P. E. Mogensen, “Experimental analysis of the joint statistical properties of azimuth spread, delay spread, and shadow fading,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 3, pp. 523–531, 2002.

[17] T. S. Rappaport, *Wireless Communications. Principles and Practice*, Prentice-Hall, Upper Saddle River, NJ, USA, 1996.

[18] J. K. Cavers, “An analysis of pilot symbol assisted modulation for Rayleigh fading channels,” *IEEE Transactions on Vehicular Technology*, vol. 40, no. 4, pp. 686–693, 1991.

[19] M. K. Simon and M.-S. Alouini, *Digital Communication over Fading Channels: A Unified Approach to Performance Analysis*, John Wiley & Sons, New York, NY, USA, 2000.

[20] C. Brunner, W. Utschick, and J. A. Nossek, “Exploiting the short-term and long-term channel properties in space and time: eigenbeamforming concepts for the BS in WCDMA,” *European Transactions on Telecommunications*, vol. 12, no. 5, pp. 365–378, 2001, special issue on Smart Antennas, http://www.chrisbrunner.org.

[21] J. Jelitto and G. Fettweis, “Reduced dimension space-time processing for multi-antenna wireless systems,” *IEEE Wireless Communications*, vol. 9, no. 6, pp. 18–25, 2002.

[22] F. Dietrich and W. Utschick, “On the effective spatio-temporal rank of wireless communication channels,” in *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 5, pp. 1982–1986, Lisbon, Portugal, September 2002.

[23] T. Simunic, L. Benini, and G. De Micheli, “Energy-efficient design of battery-powered embedded systems,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 15–28, 2001.

Constantin Siriteanu was born in Sibiu, Romania, in 1972. He received his B.S. and M.S. degrees in electrical engineering, from “Gheorghe Asachi” Technical University, Iasi, Romania, in 1995 and 1996, respectively. Between 1995 and 1997, he was a Part-Time Engineer with the Research Institute for Automation, Iasi, Romania, working on data transmission over power lines. Between 1996 and 1998, he was a Research Assistant with the Department of Automatic Control and Computer Science, “Gheorghe Asachi” Technical University, Iasi, Romania, working on digital control systems. Since 1998, he has been a Research Assistant with the Department of Electrical and Computer Engineering, Queen’s University, Kingston, Canada. His Ph.D. research has been in adaptive signal processing for smart antenna array receivers, with a focus on performance-complexity tradeoffs based on channel statistics. Between 2004 and 2006, he has also been a Course Instructor for the 4th year undergraduate Electrical/Computer Engineering Project Course at Queen’s University.

Steven D. Blostein received his B.S. degree in electrical engineering from Cornell University, Ithaca, NY, in 1983, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at Urbana-Champaign, in 1985 and 1988, respectively. He has been on the faculty at Queen’s University since 1988 and he currently holds the position of Professor and Head of the Department of Electrical and Computer Engineering. From 1999 to 2003, he was the Leader of the Multirate Wireless Data Access Major Project sponsored by the Canadian Institute for Telecommunications Research. He has also been a Consultant to industry and government in the areas of image compression and target tracking, and was a Visiting Associate Professor in the Department of Electrical Engineering at McGill University in 1995. His current interests lie in the application of signal processing to wireless communications systems, including smart antennas, MIMO systems, and space-time frequency processing for MIMO-OFDM systems. He served as Chair of IEEE Kingston Section in 1993–1994, Chair of the Biennial Symposium on Communications in 2000 and 2006, and as Associate Editor for IEEE Transactions on Image Processing from 1996 to 2000.

James Millar has B.S. degree in electrical engineering and computer science from the University of Western Ontario. He received his M.S. degree from the University of Manitoba. He previously worked with Nortel Networks providing carrier network design and planning services, and was an Instructor of electronics at St. Lawrence College in Kingston, Canada. Currently he is working with CMC Microsystems in Kingston, Canada, as a System-On-Chip Design Engineer with an interest in system-level design for microsystems integration.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 42168, Pages 1–15
DOI 10.1155/ES/2006/42168

Modeling and Design of Fault-Tolerant and Self-Adaptive Reconfigurable Networked Embedded Systems

Thilo Streichert, Dirk Koch, Christian Haubelt, and Jürgen Teich

Department of Computer Science 12, University of Erlangen-Nuremberg, Am Weichselgarten 3, 91058 Erlangen, Germany

Received 15 December 2005; Accepted 13 April 2006

Automotive, avionic, or body-area networks are systems that consist of several communicating control units specialized for certain purposes. Typically, different constraints regarding fault tolerance, availability and also flexibility are imposed on these systems. In this article, we will present a novel framework for increasing fault tolerance and flexibility by solving the problem of hardware/software codesign online. Based on field-programmable gate arrays (FPGAs) in combination with CPUs, we allow migrating tasks implemented in hardware or software from one node to another. Moreover, if not enough hardware/software resources are available, the migration of functionality from hardware to software or vice versa is provided. Supporting such flexibility through services integrated in a distributed operating system for networked embedded systems is a substantial step towards self-adaptive systems. Beside the formal definition of methods and concepts, we describe in detail a first implementation of a reconfigurable networked embedded system running automotive applications.

1. INTRODUCTION

Nowadays, networked embedded systems consist of several control units typically connected via a shared communication medium and each control unit is specialized to execute certain functionality. Since these control units typically consist of a CPU with certain peripherals, hardware accelerators, and so forth, it is necessary to integrate methods of fault-tolerance for tolerating node or link failures. With the help of reconfigurable devices such as field-programmable gate arrays (FPGA), novel strategies to improve fault tolerance and adaptability are investigated.

While different levels of granularity have to be considered in the design of fault-tolerant and self-adaptive reconfigurable networked embedded systems, we will put focus on the system level in this article. Different to architecture or register transfer level, where the methods for detecting and correcting transient faults such as bit flips are widely applied, static topology changes like node defects, integration of new nodes, or link defects are the topic of this contribution. A central issue of this contribution is *online hardware/software partitioning* which describes the procedure of binding functionality onto resources in the network at runtime. In order to allow for moving functionality from one node to another and execute it either on hardware or software resources, we will introduce the concepts of *task migration* and *task morphing*. Both task migration as well as task morphing require *hardware* and/or *software checkpointing* mechanisms and an extended design flow for providing an application engineer with common design methods.

All these topics will be covered in this article from a formal modeling perspective, the design methodology perspective, as well as the implementation perspective. As a result, we propose an operating system infrastructure for networked embedded systems, which makes use of dynamic hardware reconfiguration and is called ReCoNet.

The remainder of the article is structured as follows. Section 2 gives an overview of related work including dynamic hardware reconfiguration and checkpointing strategies. In Section 3, we introduce our idea of fault-tolerant and self-adaptive reconfigurable networked embedded systems by describing different scenarios and by introducing a formal model of such systems. Section 4 is devoted to the challenges when designing a ReCoNet-platform, that is, the architecture and the operating system infrastructure for a ReCoNet. Finally, in Section 5 we will present our implementation of a ReCoNet-platform.

2. RELATED WORK

Recent research focuses on operating systems for single FPGA solutions [1–3], where hardware tasks are dynamically assigned to FPGAs. In [1] the authors propose an online scheduling system that assigns tasks to block-partitioned devices and can be a part of an operating system for a reconfigurable device. For hardware modules with the shape of an arbitrary rectangle, placement methodologies are presented in [2, 3]. A first approach to dynamic hardware/software partitioning is presented by Lysecky and Vahid [4]. There, the authors present a *warp configurable logic architecture* (WCLA) which is dedicated for speeding up critical loops of embedded systems applications. Besides the WCLA, other architectures on different levels of granularity have been presented like PACT [5], Chameleon [6], HoneyComb [7], and dynamically reconfigurable networks on chips (DyNoCs) [8] which were investigated intensively too. Different to these reconfigurable hardware architectures, this article focuses too on platforms consisting of field-programmable gate arrays (FPGA) hosting a softcore CPU and free configurable hardware resources.

Some FPGA architectures themselves have been developed for fault tolerance targeting on two objectives. One direction is towards enhancing the chip yield during production phase [9] while the other direction focuses on fault tolerance during runtime. In [10] an architecture for the latter case that is capable of fault detection and recovery is presented. On FPGA architectures much work has been proposed to compensate faults due to the possibility of hardware reconfiguration. An extensive overview of fault models and fault detection techniques can be found in [11]. One approach suitable for FPGAs is to read back the configuration data from the device while comparing it with the original data. If the comparison was not successful the FPGA will be reconfigured [12]. The reconfiguration can further be used to move modules away from permanently faulty resources. Approaches in this field span from remote synthesis where the place and route tools are constrained to omit faulty parts from the synthesized module [13] to approaches, where design alternatives containing holes for overlying some faulty resources have been predetermined and stored in local databases [14, 15].

For tolerating defects, we additionally require checkpointing mechanisms in software as well as in hardware. An overview of existing approaches and definitions can be found in [16]. A *checkpoint* is the information necessary to recover a set of processes from a stored fault-free intermediate state. This implies that in the case of a fault the system can resume its operation not from the beginning but from a state close before the failure preventing a massive loss of computations. Upon a failure this information is used to *rollback* the system. Caution is needed if tasks communicate asynchronously among themselves as it is the case in our proposed approach. In order to deal with known issues like the *domino effect*, where a rollback of one node will require a rollback of nodes that have communicated with the faulty node since the last checkpoint, we utilize a *coordinated checkpointing* scheme [17] in our system. In [18] the impact of the checkpoint scheme on the time behavior of the system is analyzed. This includes the checkpoint overhead, that is, the time a task is stopped to store a checkpoint as well as the latencies for storing and restoring a checkpoint. In [19], it is examined how redundancy can be used in distributed systems to hold up functionality of faulty nodes under real-time requirements and resource constraints.

In the FPGA domain checkpointing has been seldomly investigated so far. Multicontext FPGAs [20–22] have been proposed, that allow to swap the complete register set (and therefore the state) among with the hardware circuit between a working set and one or more shadow sets in a single cycle. But due to the enormous amount of additional hardware overhead, they have not been used commercially. Another approach for hardware task preemption is presented in [23], where the register set of a preemptive hardware module is completely separated from the combinatorial part. This allows an efficient read and write access to the state by the cost of a low clock frequency due to routing overhead arising by the separation. Some work [24, 25] has been done to use the read back capability of Xilinx Virtex FPGAs in order to extract the state information in the case of a task preemption. The read back approach has the advantage that typically hardware design-flows are nearly not influenced. However, the long configuration data read back times will result in an unfavorable checkpoint overhead.

3. MODELS AND CONCEPTS

In this article, we consider networked embedded systems consisting of dynamically hardware reconfigurable nodes. The nodes are connected via point-to-point communication links. Moreover, each node in the network is able, but is not necessarily required, to store the current state of the entire network which is given by its current topology and the distribution of the tasks in the network.

3.1. ReCoNet modeling

For a precise explanation of scenarios and concepts an appropriate formal model is introduced in the following.

Definition 1 (ReCoNet). A ReCoNet $(g_t, g_a, \beta_t, \beta_c)$ is represented as follows.

(i) The *task graph* $g_t = (V_t, E_t)$ models the application implemented by the ReCoNet. This is done by communicating tasks $t \in V_t$. Communication is modeled by data dependencies $e \in E_t \subseteq V_t \times V_t$.

(ii) The *architecture graph* $g_a = (V_a, E_a)$ models the available resources, that is, nodes in the network $n \in V_a$ and bidirectional links $l \in E_a \subseteq V_a \times V_a$ connecting nodes.

(iii) The *task binding* $\beta_t : V_t \rightarrow V_a$ is an assignment of tasks $t \in V_t$ to nodes $n \in V_a$.

(iv) The *communication binding* $\beta_c : E_t \rightarrow E_a^i$ is an assignment of data dependencies $e \in E_t$ to paths of length i in the architecture graph g_a. A *path* p of length i is given by an i-tuple $p = (e_1, e_2, \ldots, e_i)$ with $e_1, \ldots, e_i \in E_a$ and $e_1 = \{n_0, n_1\}, e_2 = \{n_1, n_2\}, \ldots, e_i = \{n_{i-1}, n_i\}$.

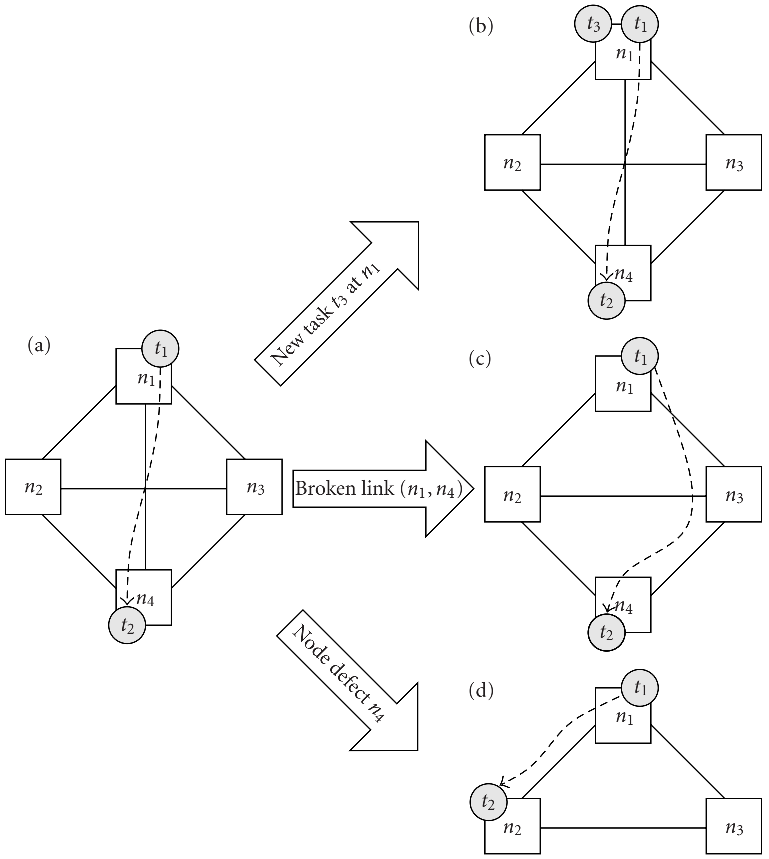


Figure 1: Different scenarios in a ReCoNet. (a) A ReCoNet consisting of four nodes and six links with two communicating tasks. (b) An additional task $t3$ was assigned to node $n1$. (c) The link $(n1, n4)$ is broken. Thus, a new communication binding is mandatory. (d) The defect of node $n4$ requires a new task and communication binding.

Example 1. In Figure 1(a), a ReCoNet is given. The task graph g_t is defined by $V_t = \{t1, t2\}$ and $E_t = \{(t1, t2)\}$. The architecture graph consists of four nodes and six links, that is, $V_a = \{n1, n2, n3, n4\}$ and $E_a = \{\{n1, n2\}, \{n1, n3\}, \{n1, n4\}, \{n2, n3\}, \{n2, n4\}, \{n3, n4\}\}$. The shown task binding is $\beta_t = \{(t1, n1), (t2, n4)\}$. Finally, the communication binding is $\beta_c = \{((t1, t2), (\{n1, n4\}))\}$.

Starting from this example, different scenarios can occur. In Figure 1(b) a new task $t3$ is assigned to node $n1$. As this assignment might violate given resource constraints (number of logic elements available in an FPGA or number of tasks assigned to a CPU), a new task binding β_t can be demanded. A similar scenario can be induced by deassigning a task from a node.

Figure 1(c) shows another important scenario where the link $(n1, n4)$ is broken. Due to this defect, it is necessary to calculate a new communication binding β_c for the data dependency $(t1, t2)$ which was previously routed over this link. In the example shown in Figure 1(c), the new communication binding is $\beta_c((t1, t2)) = (\{n1, n3\}, \{n3, n4\})$. Again a similar scenario results from reestablishing a previously broken link.

Finally, in Figure 1(d) a node defect is depicted. As node $n4$ is not available any longer, a new task binding β_t for task $t2$ is mandatory. Moreover, changing the task binding implies the recalculation of the communication binding β_c. The ReCoNet given in Figure 1(d) is given as follows: the task graph g_t with $V_t = \{t1, t2\}$ and $E_t = \{(t1, t2)\}$, the architecture graph consisting of $V_a = \{n1, n2, n3\}$ and $E_a = \{\{n1, n2\}, \{n1, n3\}, \{n2, n3\}\}$, the task binding $\beta_t = \{(t1, n1), (t2, n2)\}$ and communication binding $\beta_c = \{((t1, t2), (\{n1, n2\}))\}$.

From these scenarios we conclude that a ReCoNet given by a task graph g_t, an architecture graph g_a, the task binding β_t, and the communication binding β_c might change or might be changed over time, that is, $g_t = g_t(\tau)$, $g_a = g_a(\tau)$, $\beta_t = \beta_t(\tau)$, and $\beta_c = \beta_c(\tau)$, where $\tau \in R_0^+$ denotes the actual time. In the following, we assume that a change in the application given by the task graph as well as a change in the architecture graph is indicated by an event e. Appropriately reacting to these events e is a feature of *adaptive and fault tolerant systems*.

The basic factors of innovation of a ReCoNet stem from (i) *dynamic rerouting*, (ii) *hardware and software task migration*, (iii) *hardware/software morphing*, and (iv) *online partitioning*. These methods permit solving the problem of hardware/software codesign online, that is, at runtime. Note that this is only possible due to the availability of dynamic and partial hardware reconfiguration. In the following, we discuss the most important theoretical aspects of these methods. In Section 4, we will describe the basic methods in more detail.

3.2. Online partitioning

The goal of *online partitioning* is to equally distribute the computational workload in the network. To understand this particular problem, we have to take a closer look at the notion of *task binding* β_t and *communication binding* β_c. We therefore have to refine our model. In our model, we distinguish a finite number of the so-called *message types M*. Each message type $m \in M$ corresponds to a communication protocol in the ReCoNet.

Definition 2 (message type). M denotes a finite set of message types $m_i \in M$.

In a ReCoNet supporting different protocols and bandwidths, it is crucial to distinguish different demands. Assume a certain amount of data has to be transferred between two nodes in the ReCoNet. Between these nodes are two types of networks, one which is dedicated for data transfer and supports multicell packages and one which is dedicated for, for example, sensor values and therefore has a good payload/protocol ratio for one word messages. In such a case, the data which has to be transferred over two different networks would cause a different traffic in each network. Hence, we associate with each data dependency $e \in E_t$ the so-called *demand values* which represent the required bandwidth when using a given message type.

Definition 3 (demand). With each pair $(e_i, m_j) \in E_t \times M$, associate a real value $d_{i,j} \in \mathbb{R}_0^+$ (possibly ∞ if the message type cannot occur) indicating the *demand* for communication bandwidth by the two communicating tasks t_1, t_2 with $e_i = (t_1, t_2)$.

Example 2. Figure 2 shows a task graph consisting of three tasks with three demands. While the demand between $t1$ and $t2$ as well as the demand between $t1$ and $t3$ can be routed over all two message types ($|M| = 2$), the demand between $t2$ and $t3$ can be routed over the network that can transfer message type $m2$ only.

On the other hand, the supported bandwidth is modeled by the so-called *capacities* to each message type $m \in M$ associated with a link $l \in E_a$ in the architecture graph g_a.

Definition 4 (capacity). With each pair $(l_i, m_j) \in E_a \times M$, associate a real value $c_{i,j} \in \mathbb{R}_0^+$ (possibly 0, if the message type cannot be routed over l_i) indicating the *capacity* on a link l_i for message type m_j.

In the following, we assume that for each link $l_i \in E_a$ exactly one capacity c_i is greater than 0.

Example 3. Figure 2 shows a ReCoNet consisting of four nodes and four links. While $\{n1, n3\}$ and $\{n3, n4\}$ can transfer the message type $m1$, $\{n2, n3\}$ and $\{n2, n4\}$ can handle message type $m2$. As the data dependency $(t1, t3)$ is bound to path $(\{n1, n3\}, \{n3, n2\})$, node $n3$ acts as a gateway. The gateway converts a message of type $m1$ to a message of type $m2$. Note that only capacities with $c > 0$ and demands with $d < \infty$ are shown in this figure. In our model, we assign exactly one capacity with $c > 0$ to each communication link $l \in E_a$ in the architecture graph g_a and at least one demand with $d < \infty$ to the data dependencies $e \in E_t$ in the task graph g_t.

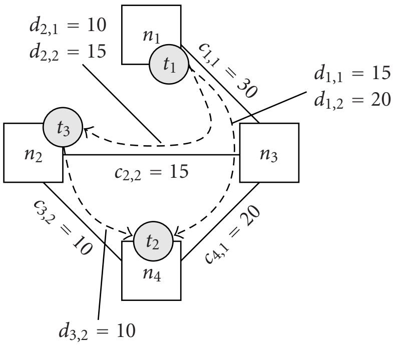


FIGURE 2: Demands are associated with pairs of data dependencies and message types while capacities are associated with pairs of links and message types.

Depending on the type of capacity, a demand of the corresponding type can be routed over such an architecture graph link. With this model refinement of a ReCoNet, it is possible to limit the routing possibilities, and moreover, to assign different demands to one problem graph edge.

Beside the communication, tasks have certain properties which are of most importance in embedded systems. These can be either soft or hard, either periodic or sporadic, have different arrival times, different workloads, and other constraints, see, for example, [26]. For *online partitioning* a precise definition of the workload is required which is known to be a complex topic. As we are facing dynamically and partially reconfigurable architectures, we have to consider two types of workload, *hardware workload* and *software workload*, which are defined as follows.

Definition 5 (software workload). The *software workload* $w^S(t, n)$ on node n produced by task t implemented in software is the fraction of execution time to its period.

This definition can be used for independent periodic and preemptable tasks. Buttazzo [26] proposed a load definition where the load is determined dynamically during runtime. The treatment of such definitions in our algorithm is a matter of future work.

Definition 6 (hardware workload). The *hardware workload* $w^H(t, n)$ on node n produced by task t is defined as a fraction of the required area and maximal available area, respectively, configurable logic elements in case of FPGA implementations.

As a task t bound to node n, that is, $(t, n) \in \beta_t$, can be implemented partially in hardware and partially in software, different implementations might exist.

Definition 7 (workload). The *workload* $w_i(t, n)$ on node n produced by the ith implementation of task t is a pair $w_i(t, n) = (w_i^H(t, n), w_i^S(t, n))$, where $w_i^H(t, n)$ $(w_i^S(t, n))$ denotes the hardware workload (software workload) on node n produced by the ith implementation of task t.

The overall hardware/software workload on a node n in the network is the sum of all workloads of the t_ith implementation of tasks bound to this node, that is, $w(n) = \sum_{(t,n)\in\beta_t} w_{t_i}(t, n)$. Here, we assume constant workload demands, that is, for all $t \in T, w_i(t, n) = w_i(t)$.

With these definitions we can define the task of online partitioning formally.

Definition 8 (online partitioning). The task of *online partitioning* solves the following multiobjective combinatorial optimization problem at runtime:

$$\min \begin{pmatrix} \max\left(\Delta_n(w^H(n)), \Delta_n(w^S(n))\right) \\ \left| \sum_n w^H(n) - \sum_n w^S(n) \right| \\ \sum_n w^H(n) + w^S(n) \end{pmatrix}, \tag{1}$$

such that

$$\begin{gathered} w^H(n), w^S(n) \leq 1, \\ \beta_t \text{ is a feasible task binding,} \\ \beta_c \text{ is a feasible communication binding.} \end{gathered} \tag{2}$$

The first objective describes the workload balance in the network. With this objective to be minimized, the load in the network is balanced between the nodes, where hardware and software loads are treated separately with $\Delta_n(w^H(n)) = \max_n(w^H(n)) - \min_n(w^H(n))$ and $\Delta_n(w^S(n)) = \max_n(w^S(n)) - \min_n(w^S(n))$.

The second objective balances the load between hardware and software. With this strategy, there will always be a good *load reserve* on each active node which is important for achieving fast repair times in case of unknown future node or link failures.

The third objective reduces the total load in the network. Finally, the constraints imposed on the solutions guarantee that not more than 100% workload can be assigned to a single node. The two feasibility requirements will be discussed in more detail next.

A feasible binding guarantees that communications demanded by the problem graph can be established in the allocated architecture. This is an important property in explicit modeling of communication.

Definition 9 (feasible task binding). Given a task graph g_t and an architecture graph g_a, a *feasible task binding* β_t is an assignment of tasks $t \in V_t$ to nodes $n \in V_a$ that satisfies the following requirements:

(i) each task $t \in V_t$ is assigned to exactly one node $n \in V_a$, that is, for all $t \in V_t$, $|\{(t, n) \in \beta_t \mid n \in V_a\}| = 1$;

(ii) for each data dependency $e \in (t_i, t_j) \in E_t$ with $(t_i, n_i), (t_j, n_j) \in \beta_t$ a path p from n_i to n_j exists.

This definition differs from the concepts of feasible binding presented in [27] in a way that communicating processes require a *path* in the architecture graph and not a direct link for establishing this communication. This way, we are able to consider *networked embedded systems*. However, considering multihop communication, we have to regard the capacity of connections and data demands of communication. This step will be named *communication binding* in the following.

Definition 10 (feasible communication binding). The task of *communication binding* can be expressed with the following ILP formulation. Define a binary variable with

$$x_{i,j} = \begin{cases} 1 & \text{data dependency } e_i \text{ is bound on link } l_j, \\ 0 & \text{else,} \end{cases} \tag{3}$$

and a mapping vector $\overrightarrow{m_i} = (m_{i,1}, \ldots, m_{i,|V_a|})$ for each data dependency $e_i = (t_k, t_j)$ with the elements

$$m_{i,l} = \begin{cases} 1 & \text{if } (t_k, n_l) \in \beta_t, \\ -1 & \text{if } (t_j, n_l) \in \beta_t, \\ 0 & \text{else.} \end{cases} \tag{4}$$

Then, the following two kinds of constraints exist.

(i) For all $i = 1, \ldots, |E_t|, C \cdot \overrightarrow{x_i} = \overrightarrow{m_i}$, with C being the incidence matrix of the architecture graph and $\overrightarrow{x_i} = (x_{i,j}, \ldots, x_{i,|E_a|})^T$.
This constraint literally means that all incoming and outgoing demands of a node have to be equal. If a demand producing or consuming process is mapped onto an architecture graph node, the sum of incoming demands differs from the sum of outgoing demands.

(ii) The second constraint restricts the sum of demands $d_{i,j}$ bound onto a link l_j to be less than or equal to the edge's capacity c_j, where $d_{i,j}$ is the demand of the data dependency e_i. For all $j = 1, \ldots, |E_a|, \sum_{i=1}^{|E_t|} d_{i,j} \cdot x_{i,j} \leq c_j$.

The objective of this ILP formulation is to minimize the total flow in the network: $\min(\sum_{i=1}^{|E_t|} \sum_{j=1}^{|E_a|} d_{i,j} \cdot x_{i,j})$. A solution to this ILP assigns data dependencies e in the task graph g_t to paths p in the architecture graph g_a. Such a solution is called a *feasible communication binding* β_c.

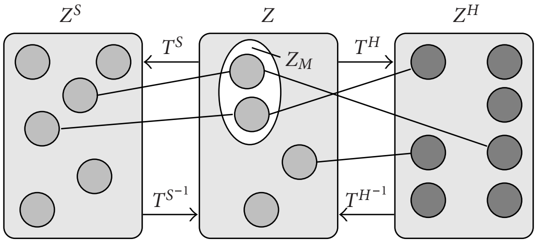


Figure 3: Hardware/software morphing is only possible in the morph states $\mathbb{Z}_{\mathrm{M}} \subseteq \mathbb{Z}$. These states permit a bijective mapping of refined states ($\mathbb{Z}^S$ and $\mathbb{Z}^H$) of task t to $\mathbb{Z}$.

3.3. Task migration, task morphing, and replica binding

In order to allow online partitioning, it is mandatory to support the *migration* and the *morphing* of tasks in a ReCoNet. Note that this is only possible by using dynamically and partially reconfigurable hardware.

A possible implementation to migrate a task $t \in V_t$ bound to node $n \in V_a$ to another node $n' \in V_a$ with $n \neq n'$ is by duplicating t on node n' and removing t from n, that is, $\beta_t \leftarrow \beta_t \backslash \{(t, n)\} \cup \{(t, n')\}$. The duplication of a task t requires two steps: first, the implementation of t has to be instantiated on node n' and, second, the current context $\mathcal{C}(t)$ of t has to be copied to the new location.

In hardware/software morphing an additional step, the transformation of the context $\mathcal{C}^H(t)$ for a hardware implementation of t to an appropriate context $\mathcal{C}^S(t)$ for the software implementation of t or vice versa, is needed. As a basis for hardware/software morphing, a task $t \in V_t$ is modeled by a deterministic finite state machine m.

Definition 11. A *finite state machine* (*FSM*) m is a 6-tuple $(I, O, S, \delta, \omega, s_0)$, where I denotes the finite set of inputs, O denotes the finite set of outputs, S denotes the finite set of states, $\delta : S \times I \rightarrow S$ is the state transition function, $\omega : S \times I \rightarrow O$ is the output function, and s_0 is the initial state.

The state space of the finite state machine m is described by the set $Z \subseteq E \times O \times S$. During the software build process and the hardware design phase, state representations, $\mathbb{Z}^S$ for software and $\mathbb{Z}^H$ for hardware, are generated by transformations T^S and T^H, see Figure 3, for instance. After the refinement of $\mathbb{Z}$ in $\mathbb{Z}^S$ or $\mathbb{Z}^H$ it might be that the states $z \in \mathbb{Z}$ do not exist in $\mathbb{Z}^S$ or $\mathbb{Z}^H$. Therefore, hardware/software morphing is only possible in equivalent states existing in both, $\mathbb{Z}^H$ and $\mathbb{Z}^S$. For these states, the inverse transformation $T^{H^{-1}}$, respectively, $T^{S^{-1}}$ must exist. The states will be called *morph states* $\mathbb{Z}_M \subseteq \mathbb{Z}$ in the following (see Figure 3). Note that a morph state is part of the context $\mathcal{C}(t)$ of a task t.

In summary, both task migration and hardware/software morphing are based on the idea of context saving or *checkpointing*, respectively. In order to reduce recovery times, we create *one replica* t' for each task $t \in V_t$ in the ReCoNet. In case of task migration, the context $\mathcal{C}(t)$ of task t can be transferred to the replica t' and t' can be activated, assuming that the replica is bound to the node n' the task t should be migrated to. Thus, our ReCoNet model is extended towards a so-called *replica task graph* g'_t.

Definition 12 (replica task graph). Given a task graph $g_t = (V_t, E_t)$, the corresponding *replica task graph* $g'_t = (V'_t, E'_t)$ is constructed by $V'_t = V_t \cup \widetilde{V}_t$ and $E'_t = E_t \cup \widetilde{E}_t$. $\widetilde{V}_t$ denotes the set of replica tasks, that is, for all $t \in V_t$ there exists a unique $t' \in \widetilde{V}_t$ and $|V_t| = |\widetilde{V}_t|$. $\widetilde{E}_t$ denotes the set of edges representing data dependencies (t, t') resulting from sending checkpoints from a task t to its corresponding replica t', that is, $\widetilde{E}_t \subset V_t \times \widetilde{V}_t$.

The replica task graph g'_t consists of the task graph g_t, the replica tasks $\widetilde{V}_t$, and additional data dependencies $\widetilde{E}_t$ which result from sending checkpoints from tasks to their replica. With the definition of the replica task graph g'_t, we have to rethink the concept of online partitioning. In particular, the definition of a *feasible task binding* β_t must be adapted.

Definition 13 (feasible (replica) task binding). Given a replica task graph g'_t and a function $r : V_t \mapsto \widetilde{V}_t$ that assigns a unique replica task $t' \in \widetilde{V}_t$ to its task $t \in V_t$. A *feasible replica task binding* is a feasible task binding β_t as defined in Definition 9 with the constraint that

$$\forall t \in V_t, \beta_t(t) \neq \beta_t(r(t)). \tag{5}$$

Hence, a task t and its corresponding replica $r(t)$ must not be bound onto the same node $n \in V_a$. In the following, we use the term *feasible task binding* in terms of *feasible replica task binding*.

3.4. Hardware checkpointing

Checkpointing mechanisms are integrated for task migration as well as morphing to save and periodically update the context of a task. In [16], checkpoints are defined to be consistent (fault-free) states of each task's data. In case of a fault or if the tasks' data are inconsistent, each task restarts its execution from the last consistent state (checkpoint). This procedure is called *rollback*. All results computed until this last checkpoint will not be lost and a distributed computation can be resumed. As mentioned above, several tasks have to go back to one checkpoint if they depend on each other. Therefore, we define checkpoint groups.

Definition 14 (checkpoint group). A checkpoint group is a set of tasks with data dependencies. Within such a group, one leader exists which controls the checkpointing.

For each checkpoint group the following premise holds: (1) each member of a checkpoint group knows the whole group, (2) the leader of a checkpoint group is not necessarily known to all the others in a group, and (3) overlapping checkpoint groups do not exist. As the developer knows the structure of the application, that is, the task graph g_t at design time, checkpoint groups can be built a priori. Thus, protocols for establishing checkpoint groups during runtime are not considered in this case.

Model of Consistency

Assume a task graph g_t with a set of tasks $V_t = \{t_0, t_1, t_2\}$ running on different nodes in a ReCoNet. The first task t_0 produces messages and sends them to the next task t_1 which performs some computation on the message's content and sends them further to task t_2. Our communication model is based on message queues for the intertask communication. Due to rerouting mechanisms, for example, in case of a link defect, it is possible that messages were sent over different links. Hence, the order of messages in general cannot be assured to stay the same.

But if messages arrive at a task t_j, we have to ensure that these were processed in the same order they have been created by task t_{j-1}. As a consequence, we assign a consecutive identifier i to every generated message. Let us assume that the last processed message by a task t_j was m_i produced at task t_{j-1}, then task t_j has to process message m_{i+1} next. If the message order arranged by task t_{j-1} has changed during communication this will be recognized at task t_j by an identifier larger than the one to be processed next. In this case all messages m_{i+k}, for all $k > 1$ will be temporarily stored in the so-called *local data set* of task t_j to be processed later in correct order.

If task t_j receives a message to store a checkpoint by the leader of a checkpoint group it will stop to process the next messages and consequently t_j will stop to produce new output messages for node t_{j+1}. In the following, all tasks of this checkpoint group will start to move incoming messages into their local data set. In addition, all tasks of the checkpoint group will store their internal states on the local data set. As a consequence, all tasks of the checkpoint group will reach a consistent state.

Hence, we define a checkpoint as follows.

Definition 15 (checkpoint). A checkpoint is a set of local data sets. It can be produced if all tasks inside a checkpoint group are in a consistent state. This is when (i) all message producing tasks are stopped and (ii) after all message queues are empty.

The checkpoint is stored in a distributed manner in the local data sets of all tasks belonging to their checkpoint group. All tasks $t \in V_t$ of the task graph g_t will have to copy their current local data set to their corresponding replica task $t' \in V_t'$ of the replica task graph g_t'. If a node hosting a task t fails, the corresponding replica task t_0 takes over the work of t and all tasks of the checkpoint group will perform a rollback by restoring their last checkpoint.

Hardware checkpointing

As we model tasks' behavior by finite state machines and we have seen how to handle input and output data to keep checkpoints consistent, we are now able to present a new model for hardware checkpointing. An FSM m that allows for saving and restoring a checkpoint can also be modeled by an FSM cm. Subsequently, we denote cm as *checkpoint FSM* or for short *CFSM*. In order to construct a corresponding CFSM cm for a given FSM m, we have to define a subset of states $S_c \subseteq S$ that will be used as a checkpoint. Using $S_c \subset S$ might be useful due to optimality reasons. First, we define a CFSM formally.

Definition 16. Given an FSM $m = (I, O, S, \delta, \omega, s_0)$ and a set of checkpoints $S_c \subseteq S$, the corresponding *checkpoint FSM* (*CFSM*) is an FSM $cm = (I', O', S', \delta', \omega', s_0')$, where

$$\begin{gathered} I' = I \times S_c \times I_{\text{save}} \times I_{\text{restore}} \quad \text{with } I_{\text{save}} = I_{\text{restore}} = \{0, 1\}, \\ O' = O \times S_c, \qquad S' = S \times S_c. \end{gathered} \tag{6}$$

In the following, it is assumed that the current state is given by $(s, s') \in S'$. The current input is denoted by i'. The state transition function $\delta' : S' \times I' \to S'$ is given as

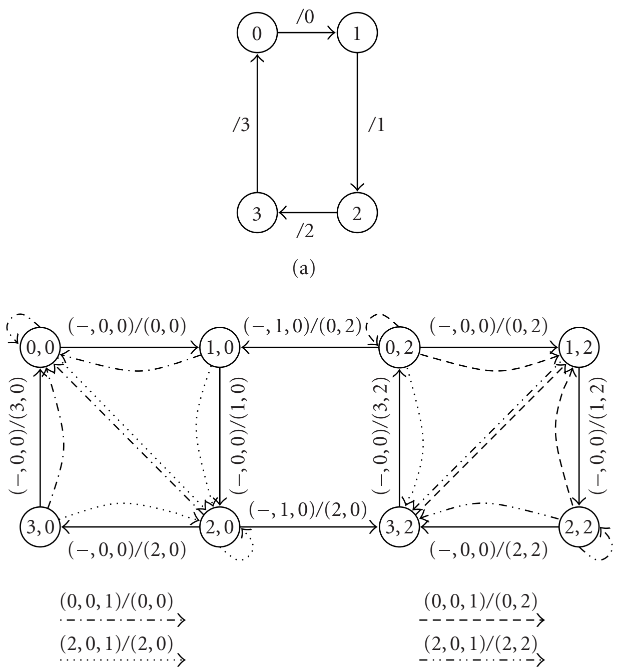
$$\delta' = \begin{cases} (\delta(s,i), s') & \text{if } i' = (i, -, 0, 0), \\ (\delta(s,i), s) & \text{if } i' = (i, -, 1, 0) \wedge s \in S_c, \\ (\delta(s,i), s') & \text{if } i' = (i, -, 1, 0) \wedge s \notin S_c, \\ (\delta(i_c,i), s') & \text{if } i' = (i, i_c, 0, 1), \\ (\delta(i_c,i), s) & \text{if } i' = (i, i_c, 1, 1) \wedge s \in S_c, \\ (\delta(s,i), s') & \text{if } i' = (i, i_c, 1, 1) \wedge s \notin S_c. \end{cases} \tag{7}$$

The output function ω' is defined as

$$\omega' = \begin{cases} (\omega(s,i), s') & \text{if } i' = (i, -, -, 0), \\ (\omega(i_c,i), s') & \text{if } i' = (i, i_c, -, 1). \end{cases} \tag{8}$$

Finally, $s_0' = (s_0, s_0)$.

Hence, a CFSM cm can be derived from a given FSM m and the set of checkpoints S_c. The new input to cm is the original input i and additionally an optional checkpoint to be restored as well as two control signals i_{save} and i_{restore}. These additional signals are used in the state transition function δ'. In case of $i_{\text{save}} = i_{\text{restore}} = 0$, cm acts like m. On the other hand, we can restore a checkpoint $s_c \in S_c$ if $i_{\text{restore}} = 1$ and using s_c as additional input, that is, $i_c = s_c$. In this case, i_c is treated as current state, and the next state is determined by $\delta(i_c, i)$. It is also possible to save a checkpoint by setting $i_{\text{save}} = 1$. In this case, the current state s is set to the latest saved checkpoint. Therefore, the state space of cm is given by the current state and the latest saved checkpoint ($S \times S_c$). Note that it is possible to swap two checkpoints by setting $i_{\text{save}} = i_{\text{restore}} = 1$. The output function is extended to output also the latest stored checkpoint s. The output function is given by the original output function ω and s' as long as no checkpoint should be restored. In case of a restore ($i_{\text{restore}} = 1$), the output depends on the restored checkpoint i_c and the input i. The initial state s_0' of cm is the initial state s_0 of m where s_0 is used as latest saved checkpoint, that is, $s_0' = (s_0, s_0)$.



(a)

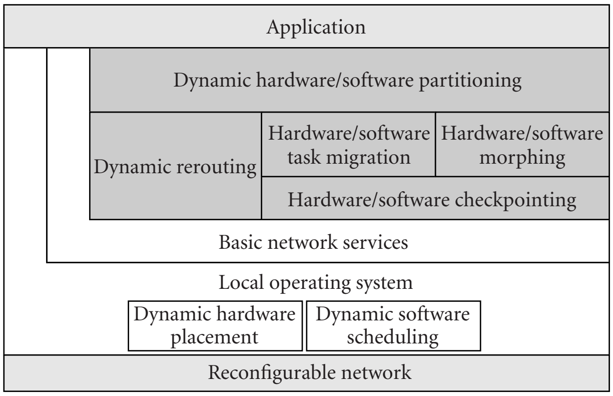
(b)

Figure 4: (a) FSM of a modulo-4-counter. (b) Corresponding CFSM for $S_c = \{0, 2\}$, that is, only in states 0 and 2 saving of the checkpoint is permitted. The state space is given by the actual state and the latest saved checkpoint.

Example 4. Figure 4(a) shows a modulo-4-counter. Its FSM m is given by $I = \emptyset, O = S = \{0, 1, 2, 3\}$, $\delta(s) = (s + 1)\%4$, $\omega(s) = s$, and $s_0 = 0$. The corresponding CFSM cm for $S_c = \{0, 2\}$ is shown in Figure 4. For readability reasons, we have omitted the swap state transitions. The state space has been doubled due to the two possible checkpoints. To be precise, there are two copies of m, one representing $s' = 0$ to be the latest stored checkpoint and one representing $s' = 2$ being the latest stored checkpoint. We can see that there exist two state transitions connecting these copied FSMs when saving a checkpoint, that is, $((2, 0), (3, 2))$ and $((0, 2), (1, 0))$. Of course it is possible to save the checkpoints in the states $(0, 0)$ and $(2, 2)$ as well. But the resulting state transitions do not differ from the normal mode transitions. The restoring of a checkpoint results in additional state transitions.

4. ARCHITECTURE AND OPERATING SYSTEM INFRASTRUCTURE

All previously mentioned mechanisms for establishing a fault-tolerant and self-adaptive reconfigurable network have to be integrated in an OS infrastructure which is shown in Figure 5. While the reconfigurable network forms the physical layer consisting of reconfigurable nodes and communication links, the top layer represents the application that will be dynamically bound on the physical layer. This binding of tasks to resources is determined by an online partitioning approach that requires three main mechanisms: (1) dynamic rerouting, (2) hardware/software task migration, and (3) hardware/software morphing. Note that the dynamic rerouting becomes more complex because messages will be sent between tasks that can be hosted by different nodes. The service provided by task migration mechanisms are required for moving tasks from one node to another while the hardware/software morphing allows for a dynamic binding of tasks to either reconfigurable hardware resources or a CPU. The task migration and morphing mechanisms require in turn an efficient hardware/software checkpointing such that states of tasks will not get lost. Basic network services for addressing nodes, detecting link failures, and sending/receiving messages are discussed in [28]. In connection with the local operating system the hardware reconfiguration management has to be considered. Recent publications [3, 29, 30] have presented algorithms for placing hardware functionality on a reconfigurable device.



Application
Dynamic hardware/software partitioning
Dynamic rerouting
Hardware/software task migration
Hardware/software morphing
Hardware/software checkpointing
Basic network services
Local operating system
Dynamic hardware placement
Dynamic software scheduling
Reconfigurable network

Figure 5: Layers of a fault-tolerant and self-adaptive network. In order to abstract from the hardware, a local operating system runs on each node. On top of this local OS, basic network tasks are defined and used by the application to establish the fault-tolerant and self-adaptive reconfigurable network.

4.1. *Online partitioning*

The binding of tasks to nodes is determined by a so-called online hardware/software partitioning algorithm which has to (1) run in a distributed manner for fault-tolerance reasons, (2) work with local information, and (3) improve the binding concerning objectives presented in the following. In order to determine a binding of processes to resources, we will introduce a two-step approach as shown in Figure 6. The first step performs a fast repair that reestablishes the functionality and the second step tries to optimize the binding of tasks to nodes such that the system can react upon a changed resource allocation and newly arriving tasks.

Fast repair

Two of the three scenarios presented in Figure 1 will be treated during this phase. In case of a newly arriving task the decision of task binding is very easy. Here, we use discrete

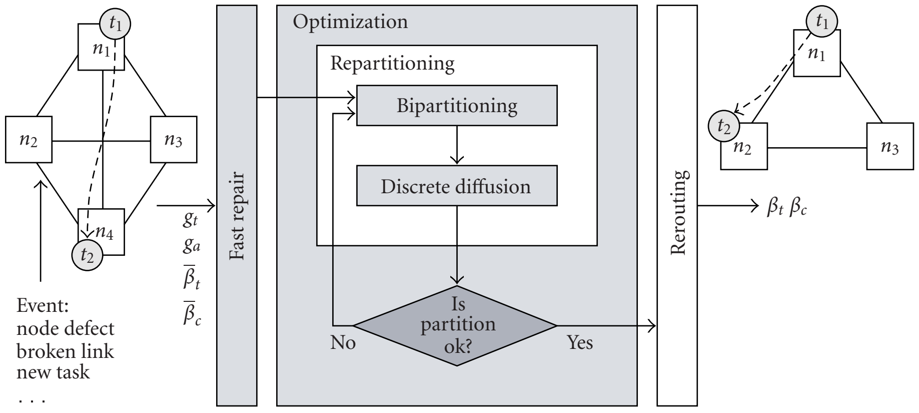


FIGURE 6: Phases of the two-step approach: while the *fast repair* step reestablishes functionality under timing constraints, the optimization phase aims on increasing fault tolerance.

diffusion techniques that will be explained later. Due to the behavior of these techniques, the load of all nodes is almost equally balanced. Hence, the new task can be bound on an arbitrary node.

In the third scenario a node defect occurs. So, tasks bound onto this node will be lost and replicas will take over the functionality. A replicated task t' will be hosted on a different node than its main task $t \in V_t$. Periodically, a replicated task receives a checkpoint by the main task and checks whether the main task is alive. If the main task is lost, the replicated task becomes a main task, restores the last checkpoint, and creates a replica on one node in its neighborhood. The main task checks either if its replicated task is still alive. If this is not the case, a replica will be created in the neighborhood again.

Bipartitioning

The applied heuristic for local bipartitioning first determines the load ratio between a hardware and a software implementation for each task $t_i \in V_t$, that is, $w^H(t_i)/w^S(t_i)$. According to this ratio, the algorithm selects one task and implements it either in hardware or software. If the hardware load is less than the software load, the algorithm selects a task which will be implemented in hardware, and the other way round. Due to the competing objectives that (a) the load on each node's hardware and software resources should be balanced and (b) the total load should be minimized, it is possible that tasks are assigned, for example, to software although they would be better assigned to hardware resources. These tasks which are suboptimally assigned to a resource on one node will be migrated to another node at first during the diffusion phase.

Discrete diffusion

While bipartitioning assigns tasks to either hardware or software resources on one node, a decentralized discrete diffusion algorithm migrates tasks between nodes, that is, changing the task binding β_t. Characteristic to the class of diffusion algorithms, first introduced by Cybenko [31] is that iteratively each node is allowed to move any size of load to each of its neighbors. The quality of such an algorithm is measured in terms of the number of iterations that are required in order to achieve a balanced state and in terms of amount of load moved over the edges of the graph.

Definition 17 (local iterative diffusion algorithm). A local iterative load balancing algorithm performs iterations on the nodes of g_a determining load exchanges between adjacent nodes. On each node $n_i \in V_a$, the following iteration is performed:

$$\begin{aligned} y_c^{k-1} &= \alpha\left(w_i^{k-1} - w_j^{k-1}\right) \quad \forall c = \{n_i, n_j\} \in E_a, \\ x_c^k &= x_c^{k-1} + y_c^{k-1} \quad \forall c = \{n_i, n_j\} \in E_a, \\ w_i^k &= w_i^{k-1} - \sum_{c=\{n_i,n_j\} \in E_a} y_c^{k-1}. \end{aligned} \tag{9}$$

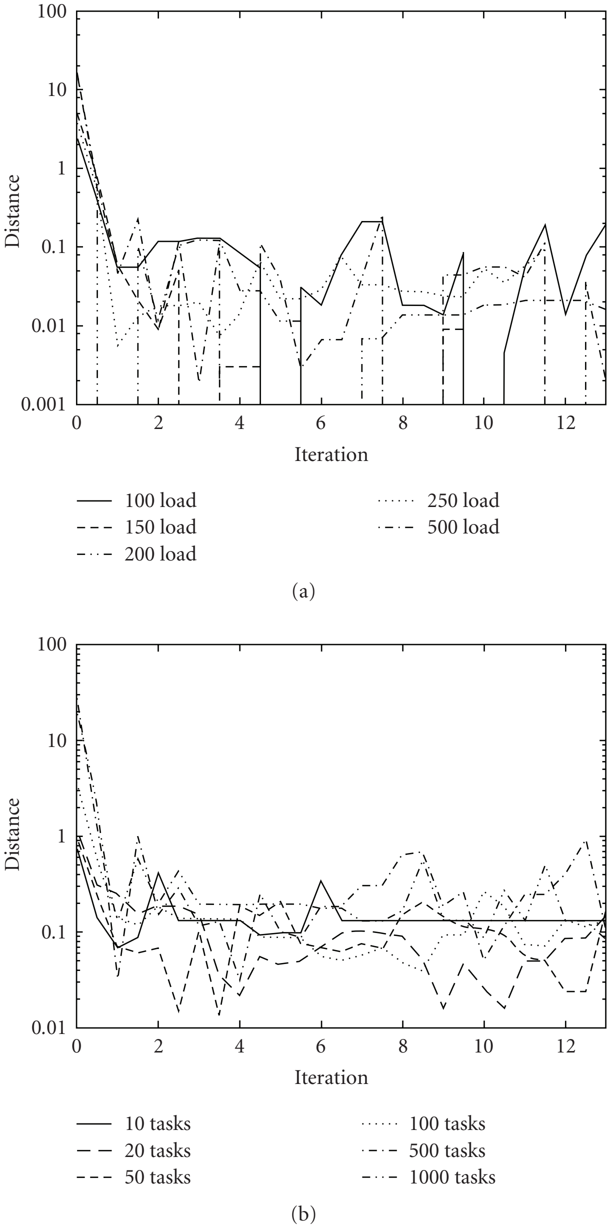
In (9), w_i denotes the total load on node n_i, y is the load to be transferred on a channel c, and x is the total transferred load during the optimization phase. Finally, k denotes the integer iteration index.

In order to apply this diffusion algorithm in applications where we cannot migrate a real-valued part of a task from one node to another, an extension is introduced. With this extension, we have to overcome two problems.

(1) First of all, it is advisable not to split one process and distribute it to multiple nodes.

(2) Since the diffusion algorithm is an alternating iterative balancing scheme, it could occur that negative loads are assigned to computational nodes.

In our approach [32], we first determine the real-valued continuous flow on all edges to the neighboring nodes. Then, the node tries to fulfill this real-valued continuous flow for each incident edge, by sending or receiving tasks, respectively. By applying this strategy, we have shown theoretically and by experiment [32, 33] that the discrete diffusion algorithm



(a)

(b)

Figure 7: Presented is the distance between the solutions of our distributed online hardware/software partitioning approach and an algorithm with global knowledge. In (a) tasks are bound to network nodes such that each node has a certain load. In (b) a certain number of tasks is bound to each node and each task is implemented in the optimal implementation style.

converges within provable error bounds and as fast as its continuous counterpart.

In Figure 7 the experimental results are shown. There, our distributed approach has been evaluated by comparing it with a centralized methodology that possesses global knowledge. The centralized methodology is based on evolutionary algorithms and determines a set R of reference solutions and calculates the shortest normalized distance $d(s)$ from the solution s found by the online algorithm to any reference solution $r \in R$:

$$d(s) = \min_{r \in R} \left\{ \left| \frac{s_1 - r_1}{r_1^{\max}} \right| + \left| \frac{s_2 - r_2}{r_2^{\max}} \right| \right\}. \qquad (10)$$

In the first experiment, we are starting from a network which is in an optimal state such that all tasks are implemented optimally according to all objectives. Now, we assume that new software tasks arrive on one node. Starting from this state, Figure 7(a) shows how the algorithm performs for different load values. In the second experiment, the initial binding of tasks and load sizes were determined randomly. For this case, which is comparable to an initialization phase of a network, we generated process sets with 10 to 1000 processes, see Figure 7(b). In this figure, we can clearly see that the algorithm improves the distribution of tasks already with the first iteration leading to the best improvement. We can see in Figure 7 that the failure of one node causes a high normalized error. Interestingly, the algorithm finds global optima but due to local information our online algorithm cannot decide when it finds a global optimum.

4.2. Hardware/software task migration

In case of software migration, two approaches can be considered. (1) Each node in the network contains all software binaries, but executes only the assigned tasks, or (2) the binaries are transferred over the network. Note that the second alternative requires that binaries are relocatable in the memory and only relative branches are allowed. With these constraints, an operating system infrastructure can be kept tiny. Besides software functionality, it is desired to migrate functionality implemented in hardware between nodes in the reconfigurable network. Similar to the two approaches for software migration, two concepts for hardware migration exist. (1) Each node in the network contains all hardware modules preloaded on the reconfigurable device, or (2) FPGAs supporting partial runtime reconfiguration are required. Comparable to location-independent software binaries, we demand that the configuration data is relocatable, too. In [34], this has been shown for Xilinx Virtex E devices and in [35], respectively, for Virtex 2 devices. Both approaches modify the address information inside the configuration data according to the desired resource location.

4.3. Hardware/software morphing

Hardware/software morphing is required to dynamically assign tasks either to hardware or software resources on a node. Naturally, not all tasks can be morphed from hardware to software or vice versa, for example, tasks which drive or read I/O-pins. But those tasks that are migratable need to fulfill some restrictions as presented in Section 3.3.

Basically, the morph process consists of three steps. At first, the state of a task has to be saved by taking a checkpoint in a morph state. Then, the state encoding has to be transformed such that the task can start in the transformed state with its new implementation style in the last step.

A requirement to morphable tasks is that they have to be equivalent such that the surrounding system does not recognize the implementation style of the morphable task. Also the transformation depends heavily on the implementation which especially leads to problems when transforming data types. While it is possible to represent numbers in hardware with almost arbitrary word width, current processors perform computations on 16 bit or 32 bit wide words. Thus, the numbers have to be extended or truncated. This modification causes again difficulties if numbers are presented in different representations. The representation which can either be one's complements, two's complement, fixed point, or floating point numbers needs to be transformed, too. Additional complexity arises if functionality requires a sequential computation in software and a parallel computation in hardware. Due to these implementation-dependent constraints, we currently support an automated morph-function generation only for bit vectors in the hardware that are interpreted as integers in the software. The designer needs to give information about the possible morph states and together with the help of the automated insertion of checkpoints into hardware/software tasks, the morphing becomes possible.

4.4. Hardware checkpointing

In Section 3, we have shown how to model checkpoints for tasks modeled by FSMs. Here, we are introducing and analyzing the overhead of three possibilities for extracting the state of a hardware module.

(i) *Scan chain*. As shown in Figure 8(a), an extra scan multiplexer in front of each flip-flop in the circuit switches between a *regular execution mode* and a *scan mode*. In the latter one, the registers are linked together to form a shift register chain. If the output of the register chain is connected to the input forming a ring shift, the module can continue regular execution immediately after the checkpoint has been read. In the case of a rollback, the last error-free state is shifted into the module.

(ii) *Scan chain with shadow registers*. Each flip-flop of the original circuit is duplicated and connected to a chain, see Figure 8(b). The multiplexer in front of the main flip flop can either propagate the value of the combinatorial circuit or the value of the corresponding shadow register. Hence, it is possible to store, restore, or swap a checkpoint within one single clock cycle.

(iii) *Memory mapping*. As shown in Figure 8(c), each flip-flop is directly accessible by the CPU via an address and a data bus. Depending on the data bus width several flip-flops can be grouped together to one word.

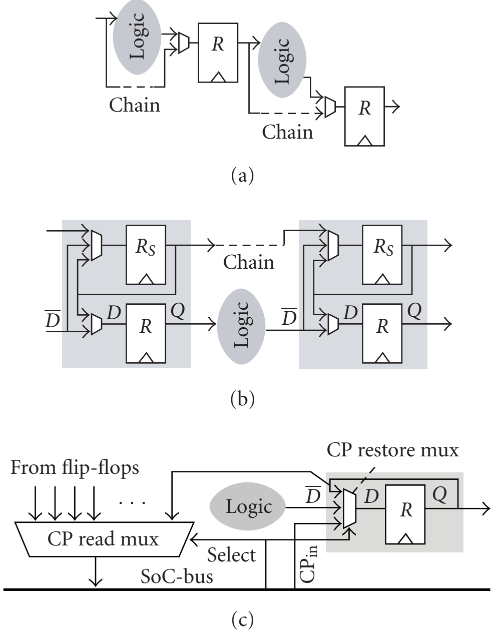


FIGURE 8: Hardware checkpointing methodologies: (a) the flip-flops are connected to form a scan chain, (b) each flip flop is replicated with a so-called shadow register which are connected to a scan chain again, (c) a set of flip-flops can be directly accessed via an address and a data port of a CPU.

All three state extraction architectures can be used for automatically modifying a given RTL design. But due to the optimization during the synthesis process from an RTL to a netlist description, it is advantageous to integrate the hardware checkpointing techniques on netlist level. Starting from the netlist, we can directly identify flip-flops by the instantiated primitives. These primitives are replaced with primitives for dedicated extended flip-flops that support saving and restoring a checkpoint, see Figure 9. Finally, the connections between the replaced flip-flops and the interface have to be determined and integrated.

In our experiments, we used the Synopsis design compiler to generate an EDIF netlist consisting of GTECH primitives. The identified GTECH flip-flops are replaced by our extended flip-flops allowing for hardware checkpointing. For our approach, we evaluated the different state extraction mechanisms discussed above according to the following properties.

(i) *Checkpoint hardware overhead*. The *checkpoint hardware overhead H* specifies the amount of additional resources required by a certain checkpointing mechanism. Here, we distinguish H_L and H_F being the checkpoint hardware overhead in terms of look up tables and flip-flops, respectively.

(ii) *Checkpoint performance reduction*. The *checkpoint performance reduction R* specifies the reduction of the maximal achievable clock frequency. As additional logic needs to be included into the original control and data paths, routing distances will slightly increase leading to a reduced clock frequency of the design.

(iii) *Checkpoint overhead*. The *checkpoint overhead C* specifies the amount of time a module is interrupted when storing a checkpoint which leads to an increase in the execution time.

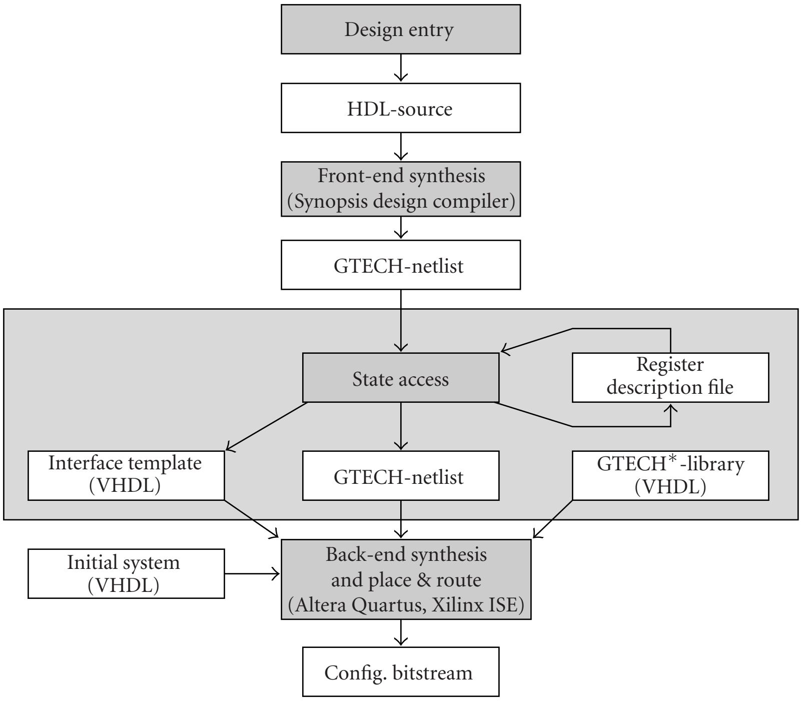

Design entry
HDL-source
Front-end synthesis (Synopsis design compiler)
GTECH-netlist
State access
Register description file
Interface template (VHDL)
GTECH-netlist
GTECH*-library (VHDL)
Initial system (VHDL)
Back-end synthesis and place & route (Altera Quartus, Xilinx ISE)
Config. bitstream

Figure 9: Design flow for integrating hardware checkpoints. Starting from the netlist, the StateAccess tool replaces flip-flops in the design by extended flip-flops that support saving and restoring of checkpoints.

(iv) *Checkpoint latency*. The *checkpoint latency L* specifies the amount of time required until the complete checkpoint data has arrived at the node hosting the specific replica task.

Table 1 presents measured values of a DES cryptographic hardware module from [36] that was automatically modified for checkpointing and tested on an Altera NIOS2 system. The table points out that each state extraction strategy is optimal in the sense of one of the defined properties. The shadow scan chain method leads in the case of high checkpoint rates to a higher throughput by the cost of almost doubling of the required logic resources. The simple scan chain approach demonstrates that it is possible to enhance a hardware module to be capable of checkpointing with an overhead of about 20% as compared to the original module.

Table 1: Results obtained by our approach by implementing the different state extraction mechanisms: scan chain, scan chain with shadow registers, and memory mapping.

	#LUTs/H_L	#Flip-Flops/H_F	
Original DES	2015/100%	984/100%	
Scan chain	2414/120%	1138/116%	
Shadow chain	3937/195%	2023/205%	
Memory mapped	2851/141%	1026/105%	
	F_{max} [MHz] / P	C	L
Original DES	116/100%	—	—
Scan chain	110/95%	10354	24979
Shadow chain	99/85%	0	16813
Memory mapped	107/92%	1306	16931

5. IMPLEMENTATION AND APPLICATION

The previously described methods have been implemented on the basis of a network consisting of four FPGA-based boards with a CPU and configurable logic resources. As an example, we implemented a driver assistant system that warns the driver in case of an unintended lane change and is implemented in a distributed manner in the network. As shown in Figure 10, a camera is connected to node $n4$. The camera's video stream is then processed in basically three steps: (a) preprocessing, (b) segmentation, and (c) lane detection. Each step is implemented as one task. The result of the lane detection is evaluated in a control task that gets in addition the present state of the drop arm switch. If the driver changes the lane without switching on the correct turn signal, an acoustic signal will warn the driver of an unintended lane change.

5.1. Architecture and local OS

As depicted in Figure 10, our prototype implementation of a ReCoNet consists of four fully connected FPGA boards. Each node is configured with a NIOS-II softcore CPU [37] running MicroC/OS-II [38] as a local operating system. The

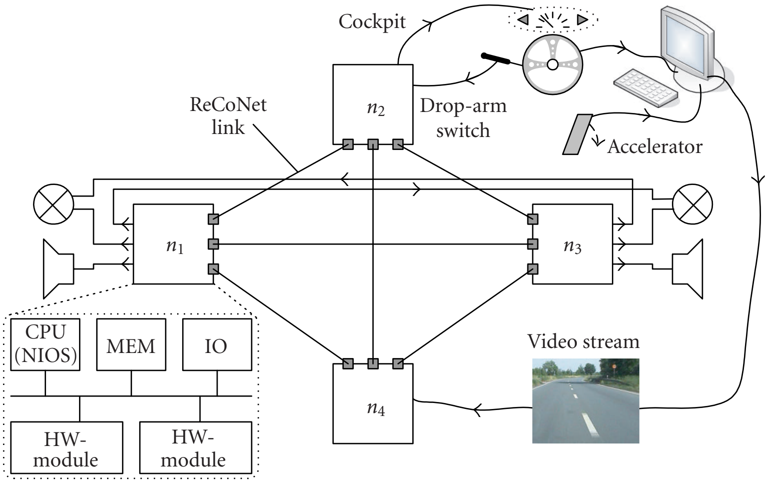


Figure 10: Schematic composition of a ReCoNet demonstrator: on the basis of four connected FPGA boards, we implemented a distributed operating system infrastructure which executes a lane detection algorithm. This application warns the driver acoustically in case of an unintended lane change.

local OS supports multitasking through preemptive scheduling and has been extended to a message passing system. On top of this extended MicroC/OS-II, we implemented the different layers as depicted in Figure 5. In detail, these are functions for checkpointing, task migration, task morphing and online hardware/software partitioning.

As Altera FPGAs do not support dynamic partial hardware reconfiguration, we configured each node with a set of hardware modules. This allows us to emulate the dynamic reconfiguration processes by selectively enabling hardware modules.

Although the MicroC/OS-II has no runtime system that permits dynamic task creation, we enabled the software task migration by transferring binaries to other nodes and linking OS functions to the same address such that tasks can access these functions on each node. This methodology reduces the amount of transferred binary data drastically compared to the alternative that the OS functions are transferred either. Also, this methodology avoids implementing a complex runtime system and the operating system keeps tiny.

5.2. *Communication*

For fault-tolerance reasons, the ReCoNet is based on a point-to-point (P2P) communication protocol [28]. As compared to a bus, we will produce some overhead by the routing on the one side while omitting the problem of bus arbitration.

The routing allows us to deal with link failures by changing the routing tables in such a way that data can be sent via alternative paths. Besides the fault tolerance, P2P networks have the advantage of an extremely high total bandwidth. In the present implementation, we set the physical data transfer rate of a single link to 12.5 Mbps and measured a maximum throughput of 700 kbps allowing even to transfer the video stream in our driver assistant application.

Each node stores a so-called *task resolution table* that allows a mapping from the task layer to the network layer, where the communication is performed with respect to the given node addresses. The task resolution is the key function for the task-2-task communication, allowing tasks to communicate among themselves regardless of there present hosting node. In the case of links, we have to distinguish between intermediate and long term failures. A single bit flip, for example, is an intermediate failure that will not demand additional care with respect to the routing, while a link down should be recognized as fast as possible in order to determine a new routes. As the link state is recognized in the transceiver ports of our implementation, we chose the advantageous variant to perform the line detection in hardware.

6. CONCLUSIONS

In this article, we presented concepts of self-adaptive networked embedded systems called ReCoNets. The particularities and novelties of such self-balancing and self-healing architectures stem from three central algorithmic innovations that have been proposed here for the first time and verified on a real platform for real applications, namely;

(i) fully decentralized online partitioning algorithms for hardware and software tasks;
(ii) techniques and overhead analysis for migration of hardware and software tasks between nodes in a network; and finally
(iii) morphing of the implementation style of a task from hardware to software and vice versa.

Although some of these techniques rely on existing principles of fault tolerance such as checkpoint mechanisms, we believe that their extension and the combination of the above three mechanisms is an important step towards self-adaptive and organic computing networks.

ACKNOWLEDGMENT

This work was supported in part by the German Science Foundation (DFG) under project Te/163-ReCoNets.

REFERENCES

[1] H. Walder and M. Platzner, “Online scheduling for block-partitioned reconfigurable devices,” in *Proceedings of Design, Automation and Test in Europe (DATE ’03)*, pp. 290–295, Munich, Germany, March 2003.

[2] A. Ahmadinia, C. Bobda, D. Koch, M. Majer, and J. Teich, “Task scheduling for heterogeneous reconfigurable computers,” in *Proceedings of the 17th Symposium on Integrated Cicuits and Systems Design (SBCCI ’04)*, pp. 22–27, Pernambuco, Brazil, September 2004.

[3] A. Ahmadinia, C. Bobda, and J. Teich, “On-line placement for dynamically reconfigurable devices,” *International Journal of Embedded Systems*, vol. 1, no. 3/4, pp. 165–178, 2006.

[4] R. Lysecky and F. Vahid, “A configurable logic architecture for dynamic hardware/software partitioning,” in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE ’04)*, vol. 1, pp. 480–485, Paris, France, February 2004.

[5] V. Baumgarte, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, “PACT XPP—a self-reconfigurable data processing architecture,” in *Proceedings of 1st International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA ’01)*, Las Vegas, Nev, USA, June 2001.

[6] Chameleon Systems, *CS2000 Reconfigurable Communications Processor, Family Product Brief*, 2000.

[7] A. Thomas and J. Becker, “Aufbau- und Strukturkonzepte einer adaptive multigranularen rekonfigurierbaren Hardwarearchitektur,” in *Proceedings of Organic and Pervasive Computing, Workshops (ARCS ’04)*, pp. 165–174, Augsburg, Germany, March 2004.

[8] C. Bobda, D. Koch, M. Majer, A. Ahmadinia, and J. Teich, “A dynamic NoC approach for communication in reconfigurable devices,” in *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL ’04)*, pp. 1032–1036, Antwerp, Belgium, August-September 2004.

[9] Altera, “FLEX 10K Devices,” November 2005, http://www.altera.com/products/devices/flex10k/f10-index.html.

[10] P. Zipf, *A fault tolerance technique for field- programmable logic arrays*, Ph.D. thesis, Siegen University, Siegen, Germany, November 2002.

[11] A. Doumar and H. Ito, “Detecting, diagnosing, and tolerating faults in SRAM-based field programmable gate arrays: a survey,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 11, no. 3, pp. 386–405, 2003.

[12] CERN, “FPGA Dynamic Reconfiguration in ALICE and beyond,” November 2005, http://alicedcs.web.cern.ch/alicedcs/.

[13] W. Xu, R. Ramanarayanan, and R. Tessier, “Adaptive fault recovery for networked reconfigurable systems,” in *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’03)*, p. 143, IEEE Computer Society, Los Alamitos, Calif, USA, April 2003.

[14] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, “Efficiently supporting fault-tolerance in FPGAs,” in *Proceedings of the ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays (FPGA ’98)*, pp. 105–115, ACM Press, Monterey, Calif, USA, February 1998.

[15] W.-J. Huang and E. J. McCluskey, “Column-based precompiled configuration techniques for FPGA,” in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’01)*, pp. 137–146, IEEE Computer Society, Rohnert Park, Calif, USA, April-May 2001.

[16] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, “A survey of rollback-recovery protocols in message-passing systems,” *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[17] K. M. Chandy and L. M. Lamport, “Distributed snapshots: determining global states of distributed systems,” *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, 1985.

[18] N. H. Vaidya, “Impact of checkpoint latency on overhead ratio of a checkpointing scheme,” *IEEE Transactions on Computers*, vol. 46, no. 8, pp. 942–947, 1997.

[19] S. Poledna, *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*, Kluwer Academic, Boston, Mass, USA, 1996.

[20] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, “A time-multiplexed FPGA,” in *Proceedings of 5th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM ’97)*, pp. 22–29, IEEE Computer Society, Napa Valley, Calif, USA, April 1997.

[21] S. M. Scalera and J. R. Vázquez, “The design and implementation of a context switching FPGA,” in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM ’98)*, p. 78, IEEE Computer Society, Napa, Calif, USA, April 1998.

[22] K. Puttegowda, D. I. Lehn, J. H. Park, P. Athanas, and M. Jones, “Context switching in a run-time reconfigurable system,” *Journal of Supercomputing*, vol. 26, no. 3, pp. 239–257, 2003.

[23] G. Brebner, “The swappable logic unit: a paradigm for virtual hardware,” in *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, Eds., pp. 77–86, IEEE Computer Press, Napa Valley, Calif, USA, April 1997.

[24] H. Simmler, L. Levinson, and R. Männer, “Multitasking on FPGA coprocessors,” in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications (FPL ’00)*, pp. 121–130, Villach, Austria, August 2000.

[25] H. Simmler, “Preemptive Multitasking auf FPGA Prozessoren,” Dissertation, University of Mannheim, Mannheim, Germany, 2001, page 279.

[26] G. C. Buttazzo, *Hard Real-Time Computing Systems*, Kluwer Academic, Boston, Mass, USA, 2002.

[27] T. Blickle, J. Teich, and L. Thiele, “System-level synthesis using evolutionary algorithms,” in *Design Automation for Embedded Systems*, R. Gupta, Ed., vol. 3, pp. 23–62, Kluwer Academic, Boston, Mass, USA, January 1998.

[28] D. Koch, T. Streichert, S. Dittrich, C. Strengert, C. D. Haubelt, and J. Teich, “An operating system infrastructure for fault-tolerant reconfigurable networks,” in *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS ’06)*, pp. 202–216, Frankfurt/Main, Germany, March 2006.

[29] K. Bazargan, R. Kastner, and M. Sarrafzadeh, “Fast template placement for reconfigurable computing systems,” *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 68–83, 2000.

[30] H. Walder and M. Platzner, “Fast online task placement on FPGAs: free space partitioning and 2D-hashing,” in *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS ’03) / Reconfigurable Architectures Workshop (RAW ’03)*, p. 178, Nice, France, April 2003.

[31] G. Cybenko, “Dynamic load balancing for distributed memory multiprocessors,” *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, pp. 279–301, 1989.

[32] T. Streichert, C. D. Haubelt, and J. Teich, "Distributed HW/SW-partitioning for embedded reconfigurable systems," in *Proceedings of Design, Automation and Test in Europe Conference and Exposition (DATE '05)*, pp. 894–895, Munich, Germany, March 2005.

[33] T. Streichert, C. D. Haubelt, and J. Teich, "Online hardware/software partitioning in networked embedded systems," in *Proceedings of Asia South Pacific Design Automation Conference (ASP-DAC '05)*, pp. 982–985, Shanghai, China, January 2005.

[34] E. L. Horta, J. W. Lockwood, and S. T. Kofuji, "Using parbit to implement partial run-time reconfigurable systems," in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, pp. 182–191, Springer, Montpellier, France, September 2002.

[35] H. Kalte, G. Lee, M. Porrmann, and U. Rückert, "REPLICA: a bitstream manipulation filter for module relocation in partial reconfigurable systems," in *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium—Reconfigurable Architectures Workshop*, p. 151, Denver, Colo, USA, April 2005.

[36] OpenCores, 2005, http://www.opencores.org.

[37] Altera, "Nios II Processor Reference Handbook," July 2005.

[38] J. Labrosse, *Micro-C/OS-II*, CMP Books, Gilroy, Calif, USA, 2nd edition, 2002.

Thilo Streichert received the Diploma degree in electrical engineering and computer science from the University of Hannover, Germany, in 2003. Beside his studies, he gained industrial research experience at the Multimedia Research Labs of NEC in Kawasaki (2002), Japan, and in the Semiconductor and ICs Advanced Engineering-Design Methodology Group of Bosch (2003), Germany. He is currently a Ph.D. degree candidate in the Department of Computer Science at the University or Erlangen-Nuremberg, Germany. His research interests include reconfigurable computing and networked embedded systems.

Dirk Koch received his Diploma degree in electrical engineering from the University of Paderborn, Germany, in 2002. During his studies, he worked on neural networks on coarse-grained reconfigurable architectures at Queensland University of Technology, Brisbane, Australia. In 2003, he joined the Department of Computer Science of the University of Erlangen-Nuremberg, Germany. His research interests are distributed reconfigurable embedded systems and reconfigurable hardware architectures.

Christian Haubelt received his Diploma degree in electrical engineering from the University of Paderborn, Germany, in 2001, and received his Ph.D. degree in computer science from the Friedrich-Alexander University of Erlangen-Nuremberg, Germany, in 2005. He leads the System-Level Design Automation Group in the Department of Hardware-Software Codesign at the University of Erlangen-Nuremberg. He serves as a Reviewer for several well-known international conferences and journals. His special research interests focus on system-level design, design space exploration, and multiobjective evolutionary algorithms.

Jürgen Teich received his Master's degree (Dipl. Ing.) in 1989 from the University of Kaiserslautern (with honors). From 1989 to 1993, he was a Ph.D. student at the University of Saarland, Saarbrücken, Germany, from where he received his Ph.D. degree (summa cum laude). His Ph.D. thesis entitled "A compiler for application-specific processor arrays" summarizes his work on extending techniques for mapping computation intensive algorithms onto dedicated VLSI processor arrays. In 1994, he joined the DSP Design Group of Prof. E. A. Lee and D. G. Messerschmitt in the Department of Electrical Engineering and Computer Sciences (EECS) at UC Berkeley, where he was working in the Ptolemy Project (postdoc). From 1995 to 1998, he held a position at the Institute of Computer Engineering and Communications Networks Laboratory (TIK) at ETH Zürich, Switzerland, finishing his habilitation entitled "Synthesis and optimization of digital hardware/software systems" in 1996. From 1998 to 2002, he was a Full Professor in the Electrical Engineering and Information Technology Department of the University of Paderborn, holding a Chair in computer engineering. Since 2003, he is appointed a Full Professor in the Computer Science Institute of the Friedrich-Alexander University Erlangen-Nuremberg holding the Chair of Hardware-Software Codesign. He has been a Member of multiple program committees of well-known conferences and workshops. He is a Member of the IEEE and the author of a textbook edited by Springer in 1997. His research interests are massive parallelism, embedded systems, codesign, and computer architecture. Since 2004, he also has been an elected reviewer for the German Science Foundation (DFG) for the area of computer architecture and embedded systems. He is involved in many interdisciplinary national basic research projects as well as industrial projects. He is supervising 19 Ph.D. students currently.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 54074, Pages 1–14
DOI 10.1155/ES/2006/54074

MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution

Riad Ben Mouhoub and Omar Hammami

UEI, ENSTA 32, Boulevard Victor, 75739 Paris, France

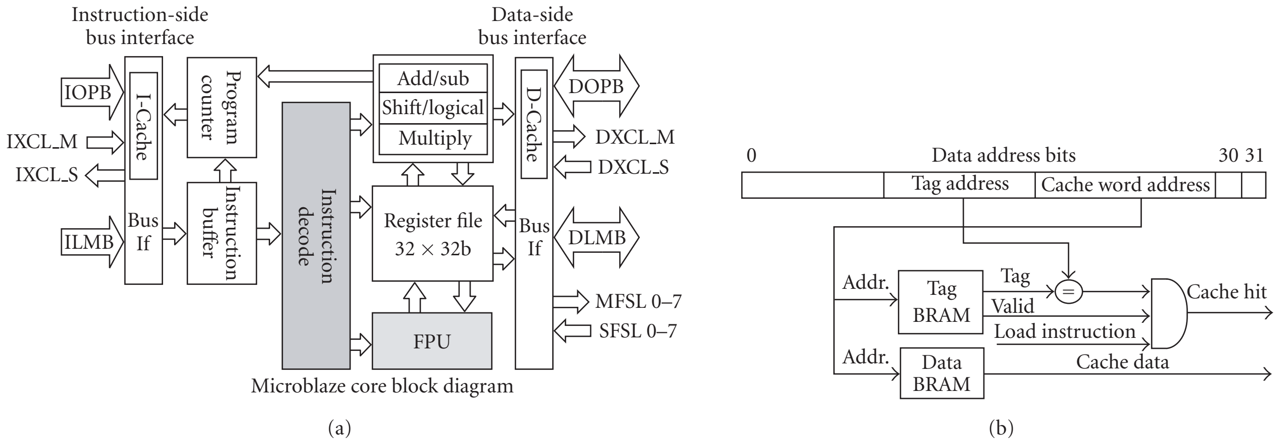
Received 15 December 2005; Revised 5 May 2006; Accepted 2 June 2006

Fully integrated system level design space exploration methodologies are essential to guarantee efficiency of future large scale system on programmable chip. Each design step in the design flow from system architecture to place and route represents an optimization problem. So far, different tools (computer architecture, design automation) are used to address each problem separately with at best estimation techniques from one level to another. This approach ignores the various and very diverse vertical relations between distinct levels parameters and provides at best local optimization solutions at each step. Due to the large scale of SoC, system level design methodologies need to tackle the system design process as a global optimization problem by fully integrating physical design in the design space exploration. We propose MOCDEX, a multiobjective design space exploration methodology, for multiprocessor on chip which closes the gap between these associated tools in a fully integrated approach and with hardware in the loop. A case study of a 4-way multiprocessor demonstrates the validity of our approach.

1. INTRODUCTION

System on chip are increasingly becoming complex to design, test, and fabricate. SoC design methodologies make intensive use of intellectual properties (IPs) [1] to reduce the design cycle time and meet stringent time to market constraints. However, associated tools still lag behind when addressing the huge associated design space exposed by the combination of soft IP. In addition, failure to meet an efficient distribution in terms of performance, area, and energy consumption makes the whole design inappropriate. Although this problem is already hard to solve in the ASIC domain, it is exacerbated in the system on programmable chip (SoPC) domain. SoPC are large scale devices offering abundant resources but in fixed amount and in fixed location on chip. Implementing embedded multiprocessors on these devices presents several advantages, the most important is to be able to quickly evaluate various configurations and tune them accordingly. Indeed, embedded multiprocessor design is highly application-driven and it is therefore highly advantageous to execute applications on real prototypes. However, due to the fact that specific resources are located at fixed positions on these large chips it is hard not to take into account the important impact of place and route results on the critical paths and therefore on the overall performance. In this paper, we address this multiobjective optimization problem [2] restricted to performance and area through the combination of an efficient design space exploration (DSE) technique coupled with direct execution on an FPGA board [3]. The direct execution removes the prohibitive simulation time associated with the evaluation of embedded multiprocessor systems. A side effect of this approach is that direct execution requires actual on chip implementation of the various multiprocessor configurations to be explored which provides actual post synthesis and place and route area information. The resulting flow is fully integrated from multiprocessor platform specification to execution.

The paper is organized as follows. In Section 2, we review previous work. Section 3 describes an example of soft IP-based multiprocessor and the breadth of the problem associated with the design of such multiprocessor on a particular instance of embedded memories optimization. Section 4 presents our approach, MOCDEX, based on multiobjective evolutionary algorithms (EA) and direct execution. In Section 5 we describe a case study and validation, while Section 6 provides exploration results. Section 7 provides statistical insight in the explored design space and demonstrates the diversity of multiprocessor configurations explored during the automatic process. Finally, we conclude in Section 8 with remarks and directions for future work.



(a)

(b)

Figure 1: (a) MicroBlaze soft IP processor. (b) MicroBlaze processor cache organization.

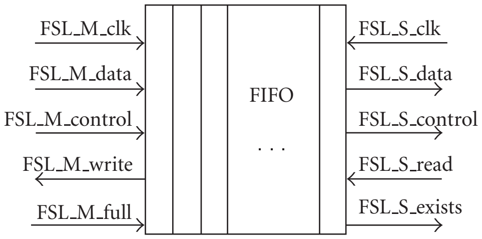


Figure 2: Fast simplex link.

2. PREVIOUS WORK

The recent emergence of multiprocessors on chip as strong potential candidates to address performance, energy, and area constraints for embedded applications has resulted in the following question: how do we design efficient multiprocessors on chip for a target application? Design automation tools fail to address this question, while traditional parallel computer architectures techniques [4] have not been exposed to the huge diversity brought by soft IP-based design methodologies and the strong constraints of embedded systems [5]. Therefore, the design of multiprocessor on chip is the convergence focus of previously unrelated techniques and as such represents a new problem on how to establish a close integration between those techniques. It is then not surprising that few works so far have been devoted to design methodologies for multiprocessors on chip. In [6] they present a design flow for the generation of application-specific multiprocessor architectures. In the flow, architectural parameters are first extracted from a high-level specification and are used to instantiate architectural components such as processors, memory modules, and communication networks. Cycle accurate cosimulations of the architectures are used for performance evaluation while all results in our case are obtained through actual execution and they do not use design space exploration algorithm. In [7], synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors is proposed based on an iterative improvement algorithm implemented in the context of a commercial design flow. The proposed algorithm is based on cycle count estimation and instruction-set simulations, and although synthesis results are used, both architecture and implementation flows are still decoupled. In [8] they propose an automated exploration framework for FPGA-based soft multiprocessor systems. Using as input the application graph that describes tasks and communication links, outputs of the exploration step are a microarchitecture configuration of processors and communication channels, a mapping of the application tasks and links onto the processors and channels of the micro-architecture. They formulate the exploration problem as an integer linear problem. The "best design" based on the ILP results is selected and synthesized to verify performance. This verification may fail because routing details are not taken into account during the exploration process. This approach still keeps decoupled design automation tools and exploration, while in our approach design space exploration fully integrates design automation tools since solutions are ranked on the area results obtained post-synthesis and place and route and performance results obtained from actual execution on board. Besides, the problem formulation ignores the arbitration overhead when computing the communication access time again due to the static nature of the design space exploration decoupled from actual execution. As pointed out by the authors, this can lead to a significant source of errors when there are a large number of masters on the bus. Finally, it should be clear that no single "best design" exists in any multiobjective optimization problem and only a Pareto set can be obtained. In [9] they present high-level scheduling and interconnect topology synthesis techniques for embedded multiprocessor system-on-chip that are streamlined for one or more digital signal processing applications. The proposed interconnect synthesis method utilizes a genetic algorithm (GA) operating in conjunction with a list scheduling algorithm which produces candidate topology graphs based on direct physical communication. The proposed algorithm is a single objective algorithm, while the algorithm used in our work is a multiobjective algorithm; and although we use direct link we optimize also buffering capacities by trading on-chip memory among embedded processor cache memories and connection link buffers. To the best of our knowledge our work is the first to fully integrate and therefore close the gap between design automation tools and architecture design space exploration technique in a multiobjective constraints paradigm with actual execution for all multiprocessor on chip configurations explored during the design space exploration process.

3. SOFT IP-BASED EMBEDDED MULTIPROCESSOR SYSTEMS

Soft IP-based embedded multiprocessor systems are SoC fully designed with soft IPs. This includes soft IP processors, interconnect infrastructure and memories. An example of such soft IP multiprocessor is described below based on Xilinx EDK IPs [10].

3.1. *MicroBlaze soft IP processor*

MicroBlaze soft IP [11] is a 32-bit 3-stage single issue pipelined Harvard style embedded processor architecture provided by Xilinx as part of their embedded design tool kit.

Both caches are direct mapped, with 4-word cache lines allowing configurable cache and tag size and user selectable cacheable memory area. Data cache uses a write-through policy. MicroBlaze core configurability extends to functional unit through user selectable barrel shifter (BS), hardware multiplier (HWM), hardware divider (HWD), and floating point unit (FPU). MicroBlaze has neither static nor dynamic branch prediction unit and supports branches with delay slots. For its communication purposes, MicroBlaze uses either a bus or a direct link. The on-chip peripheral bus (OPB) is part of IBM CoreConnect bus architecture and allows the design of complete single processor systems with peripherals and uses designed hardware accelerators [12, 13]. However, even for a simple embedded-processor-based multiprocessors designs such as MicroBlaze, the OPB bus is not suitable because of its lack of scalability. Another approach is provided by “Fast Simplex Link” [14] which allows direct connection between embedded processors through FIFO channels.

3.2. *MicroBlaze fast simplex link*

The fast simplex link (FSL) [14] is an IP developed by Xilinx to achieve a fast unidirectional point-to-point communication between any two components. The FSL link is implemented as a 32-bit wide FIFO with configurable depth and width option. The FSL can be either a master or a slave interface depending upon its use.

MicroBlaze soft embedded processor allows up to 8 master and slave FSL interfaces. Basic software drivers are provided to simplify the use of FSL connection. They consist of read/write routines and control functions. The read/write routines can be executed in two different ways: blocking and nonblocking mechanism.

3.3. *IBM interconnect*

The IBM interconnect [10] represents a set of IPs used to develop SoC devices. It includes the PLB and OPB bus, a PLB-OPB bridge, and various peripherals.

3.4. *MPSoC platform description*

Our FPGA multiprocessor platform consists of four MicroBlaze processors with instruction and data cache units. These processors are connected with each other through FSL channels.

Each MicroBlaze is connected, as shown in Figure 3, to an OPB bus to use a timer and an interrupt controller for threads and OS execution. MicroBlaze MB0 is connected to the OPB bus which is connected to the PCI interface of the host (WS). This allows the designer to send and receive data from the host to the multiprocessor system. We implemented a soft layer of communication in each MicroBlaze which performs send and receive functions of packets. The packets consist of headers representing the destination and source addresses and the number of flits in the payload. A wormhole routing algorithm was used since it uses less memory, making it suitable for network on chip communication. As it can be seen a 4-way multiprocessor has been built based on the previously described soft IPs.

The implementation of such a soft IP multiprocessor on FPGA platform requires a variable amount of resources as each soft IP composing the multiprocessor requires a variable amount of resources depending on the configuration options [10]. Table 1 provides an insight on such variability.

Such a soft IP multiprocessor can be easily adapted to the need of a specific application adapted to a particular application. However, these systems for best efficiency and low memory latency require the use of embedded on chip memories. Unfortunately, embedded memories are scarce resources for which processors instruction and data cache memories as well as bus and network on-chip FIFO-based interfaces will compete. This competition is dominated by the absolute requirement of efficiency in performance, area, and energy consumption [5]. If we focus on cache and FSL configurability, we have for each cache memory 7 possible configurations and for the FSL 11 possible configurations. The design space associated with those parameters ($7^4 \times 11^8$, thus 514 675 673 281 different configurations) requires 16 321 years of simulation for 1 minute simulation per configuration.

4. MOCDEX MULTIOBJECTIVE DESIGN SPACE EXPLORATION

4.1. *Problem formulation*

The design challenge represented by soft IP-based multiprocessor design is a multiobjective optimization problem [2].

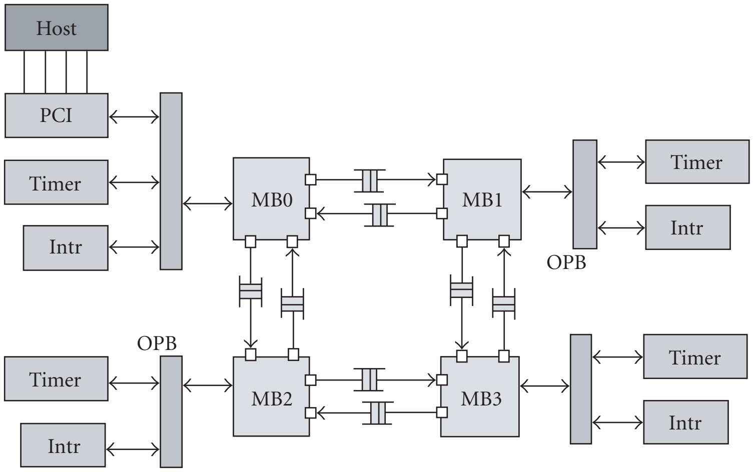


Figure 3: Mesh platform 2 × 2.

Table 1: Multiprocessor soft IP resources variation.

	Slices	FF	BRAM			Slices	FF	BRAM	
Soft IP	Min	Min	Min	Parameters	Soft IP	Min	Min	Min	Parameters
	Max	Max	Max			Max	Max	Max	
MicroBlaze	731 var	552 var	0 var	Cache sizes 1 K, 2 K, 4 K, 8 K, 16 K, 32 K, 64 K	OPB	46 410	5 121	N/A N/A	Data bus width, address bus width, arbiter
					OPB PCI	340 3025	445 2105	0 2+	Interface/DMA parameters
FSL width/depth	21 451	36 34	0 17	FIFO sizes 8, 16, 32, 64, 128, 256, 512, 1 K, 2 K, 4 K, 8 K	OPB timer	99 200	105 266	0	Timer counter widths
					OPB intr ctr	54 307	63 342	0	Number of interrupt inputs

The multiobjective optimization problem is the problem of simultaneously minimizing the n components (e.g., area, number of execution cycles, energy consumption), f_k, $k = 1, \dots, n$, of a possibly nonlinear function f of a general decision variable x in a universe U, where

$$f(x) = (f_1(x), f_2(x), \dots, f_n(x)). \tag{1}$$

The problem has usually no unique optimal solution but a set of nondominated alternative solutions known as the Pareto-optimal set. The dominance is defined as follows.

Definition 1 (Pareto dominance). A given vector $u = (u_1, u_2, \dots, u_n)$ is said to dominate $v = (v_1, \dots, v_n)$ if and only if u is partially less than v ($u_p < v$), that is,

$$\forall i \in \{1, \dots, n\}, \quad u_i \leq v_i, \quad \exists i \in \{1, \dots, n\} : u_i < v_i. \tag{2}$$

The Pareto optimality definition derives from the Pareto dominance.

Definition 2 (Pareto optimality). A solution $x_u \in U$ is said to be Pareto optimal if and only if there is no $x_v \in U$ for which $v = f(x_v) = (v_1, \dots, v_n)$ *dominates* $u = f(x_u) = (u_1, \dots, u_n)$.

Pareto-optimal solutions are also called efficient, nondominated, and noninferior solutions. The corresponding objective vectors are simply called nondominated. The set of all nondominated vectors is known as the nondominated set or the Pareto set (also Pareto-optimal set or Pareto-optimal front). This Pareto set can be seen as the tradeoff surface of the problem. The solution of a practical problem such as multiprocessor system on chip (MPSoC) design may be constrained by a number of restrictions imposed on a decision variable. Constraints may express the domain of definition of the objective function or alternatively impose further restrictions on the solution of the problem according to knowledge at a higher level. In the general case of system on programmable chip, the amount of on chip memory for example is fixed and represents a clear and stringent constraint. The constrained optimization problem is that of minimizing a multiobjective function $(f_1, \dots, f_k)$ of some generic decision variable x in a universe U subject to a positive number $n - k$ of conditions involving x and eventually expressed as a functional vector inequality of the type

$$(f_{k+1}(x), \ldots, f_n(x)) < (g_{k+1}, \ldots, g_n), \quad (3)$$

where the inequality applies component-wise. It is implicitly assumed that there is at least one point in U which satisfies all constraints although in practice that cannot always be guaranteed.

The case study of multiobjective optimization we will address in this paper is the minimization of area (BRAM $f1$ and slices resources $f2$) and execution time (number of cycles $f3$) representing a 3-objectives multiobjective problem.

4.2. Multiobjective optimization and multiobjective evolutionary algorithms (MOEA)

Multiobjective optimization have not been addressed properly by traditional optimization techniques (gradient based, simulated annealing, linear programing) since most of these techniques are mono-objective. Extending these techniques through approaches using aggregation functions does not represent true multiobjective optimization and does not produce multiple solutions. Multiobjective evolutionary algorithms (MOEA) are more appropriate to solve optimization problems with concurrent conflicting objectives and are particularly suited for producing Pareto-optimal solutions. Several Pareto-based evolutionary algorithms have been proposed during the last decade, SPEA-2, PESA, and NSGA-II, [2, 15] to solve multicriteria optimization problems. The NSGA-II [16] is an MOEA considered to outperform other MOEA [17] and is briefly presented below.

Individuals classification

Initially, before carrying out the selection, one assigns to each individual in the population a row *rank* (by using the Pareto set). All the nondominated individuals of the same row are classified in a category. To this category, we assign effectiveness, which is inversely proportional to the order of Pareto set. Figure 4 presents an example of classification in Pareto sets.

Main loop of algorithm NSGA-II [16]

Initially, a random parent population $P0$ is created. Each individual of this population is affected to an adequate Pareto rank. From the population $P0$, we apply the genetics operators (selection, mutation, and crossover) to generate the population child $Q0$ of size N. The elitism is ensured by the comparison between the current population P_t and the preceding population P_{t-1}. The NSGA-II procedure follows (see Algorithm 1).

The NSGA-II algorithm runs in time $O(GN \log^{M-1} N)$, where G is the number of generations, M is the number of objectives, and N is the population size [17]. In addition, our previous experience on multiobjective optimization of soft IP embedded processor [18, 19] emphasizes this choice.

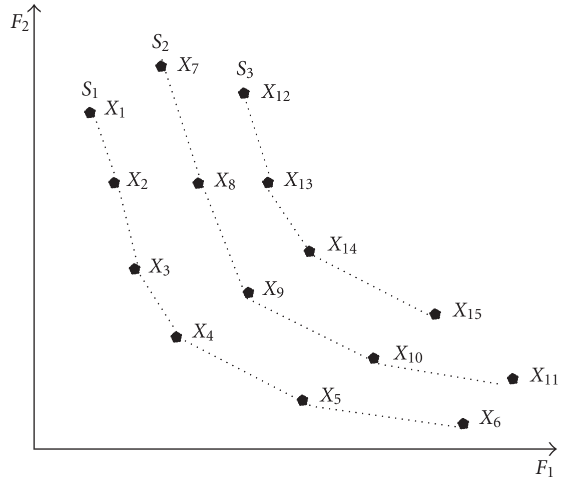


Figure 4: Classification of the individuals in several fronts according to the Pareto rank (list of Pareto sets).

```
$R_t = P_t U Q_t$ # combine parent and children
      population
$F$ = fast-nondominated-sort ($R_t$) # $F$ all
      nondominated fronts sets
$P_{t+1} = \varnothing$ and $i = 1$ # initialization
until $|P_{t+1}| + |F_i| \leq N$ # till parent pop is filled
  Crowding-distance-assignment ($F_i$) # compute
  distance in $Fi$
  $P_{t+1} = P_{t+1} U F_i$ # include $i$th nondominated
          front in the parent pop
  $i = i + 1$ # check the next front for inclusion
Sort ($F_i, <_n$) # Sort in descending order using $<_n$
$P_{t+1} = P_{t+1} U F[1 : (N - |P_{t+1}|)]$ # Choose the first
        ($N - |P_{t+1}|$) elements
$Q_{t+1}$ = make-new-pop ($P_{t+1}$) # apply genetic
        operators to create new pop $Q_{t+1}$
$T = t + 1$ # increment to next generation
```

Algorithm 1: NSGA-II.

4.3. MOCDEX

It is clear that MOEAs such as NSGA-II requires the evaluation of individuals (MPSoC configurations) with regard to the 3 objectives considered, BRAM, slices and number of cycles Although, BRAM and slices, could be estimated, we advocate the full use of design automation tools including place and route to access this information. Indeed, for complex systems on large platform FPGA place and route impact cannot be overlooked and can hardly be estimated with sufficient accuracy to be used in an automatic multiobjective design space exploration tool. The execution time of multiprocessor on chip can be obtained through simulation either at RTL level which would be prohibitive for large design space exploration without massive use of computing resources (compute farms) or at TLM level (SystemC) as often advocated [20, 21].

However although SystemC level simulation has been regularly proved to outperform RTL VHDL level simulation, it does not outperform actual execution on FPGA. We argue that for large scale MPSOC, FPGA platform represents an opportunity to both reduce simulation time through actual execution and increase the design space exploration through this reduction of the evaluation of each MPSOC configuration. Our proposal follows.

MOCDEX (general)

(1) Generate random population of MPSOC configurations within soft IP parameters constraints.
(2) For all configurations,
    (a) generate hardware/software platform specification files,
    (b) generate through system EDA and IPs HW/SW model of the MPSOC,
    (c) synthesize/place and route MPSOC configuration using EDA tools,
    (d) record place and route reports,
    (e) download configuration file on FPGA platform,
    (f) execute MPSOC configuration and record execution clock cycles,
    (g) rank the solution.
(3) Generate new population using MOEA algorithm.
(4) Is the Pareto front satisfactory or the number of generations reached if no goto 3?
(5) Final Pareto front MPSOC configurations are available for selection.

As shown in Figure 5, both the DSE and physical design are executed on a host PC while the execution is achieved on a PCI-based FPGA platform which communicates execution results to the host.

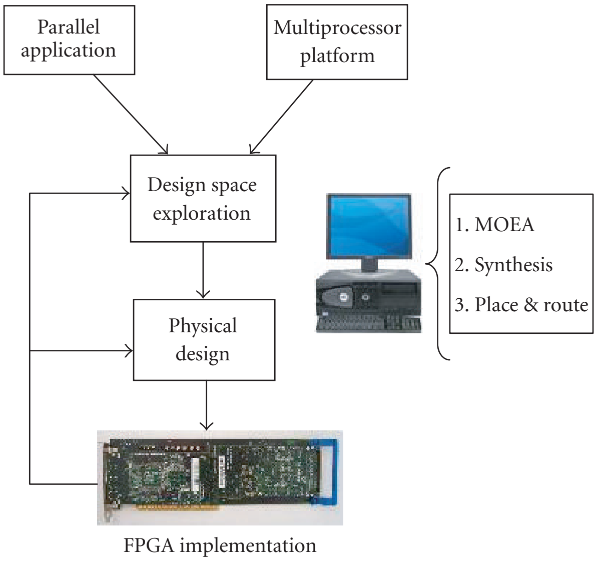


Figure 5: MOCDEX MPSOC exploration flow.

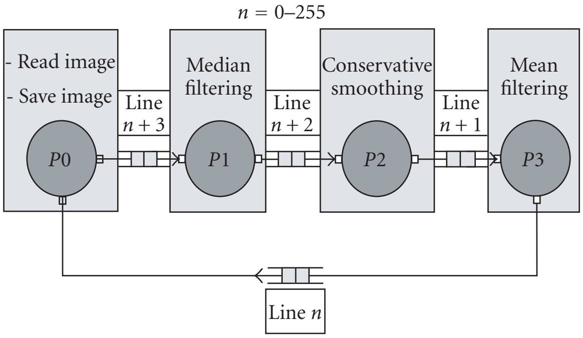


Figure 6: Image filtering application multiprocessor platform distribution.

5. CASE STUDY AND VALIDATION

The previously described design flow has been applied in the framework of Xilinx FPGA platforms.

5.1. *Image filtering application*

A design of four Xilinx MicroBlaze processors, communicating with eight FSL channels in a mesh topology and executing image filtering algorithms, was implemented at 100 MHz. This application was chosen because it requires extensive data processing and data communication among the filters for a good and fast testing of our exploration framework.

Figure 6 shows our filtering methodology. As we can see, the execution is achieved in a pipelined way where image lines are sent from a processor to another as soon as the previous processor has finished its work on it. Obviously, this type of execution makes us save a significant amount of time and memory which are often the major constraints for embedded systems in general and for our platform in particular. Indeed, performing this task in a pipelined way allows us to have a maximum of three image lines stored in the associated processor's memory rather than the whole image. The rest of the image lines will enter the FIFOs (FSLs) of their respective processors one by one. The processor $P0$ in Figure 6 receives image data from the host computer through the PCI bus. Once it receives the data it immediately sends it to the next processor which is $P1$. $P1$ performs a median filtering which results in noise reduction from the image. It is performed on a 3-by-3 pixel window where the center pixel value is replaced by the median of the neighboring pixel values. This value is obtained by sorting the pixels based on their numerical values and then replacing the pixel to be processed by the middle value. The processor $P2$ fetches the line coming from $P1$ and performs a conservative smoothing on it which is an operation that preserves the high spatial frequency details. Finally, the third processor $P3$ performs a mean filtering which consists of very simple method used for noise reduction where the pixel to be processed is replaced by the average

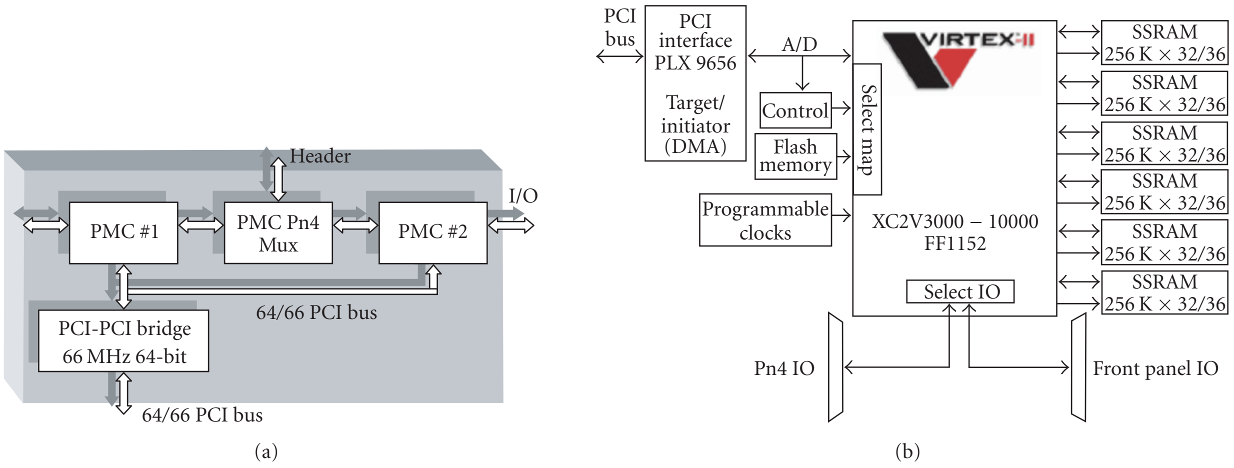


Figure 7: Alpha-data ADM-XRC-II and ADC-PMC boards.

Table 2: Multiprocessor on chip design space.

Procs	FSL1Out	FSL2Out	D-Cache	I-Cache
MB0	16 . . . 2048	16 . . . 2048	512 . . . 4096	512 . . . 4096
MB1	16 . . . 2048	16 . . . 2048	512 . . . 4096	512 . . . 4096
MB2	16 . . . 2048	16 . . . 2048	512 . . . 4096	512 . . . 4096
MB3	16 . . . 2048	16 . . . 2048	512 . . . 4096	512 . . . 4096

Table 3: Xilinx virtex-II XC2V 8000 resources.

XC2V8000	Values
Slices	46 952
BRAM (18 Kbits)	168
18 × 18 multipliers	168
DCM	12
Max. Dist RAM Kb	1456

value of its neighbors. Due to the different amount of computations required by each filter, it results in different workload for each processor. Thus the execution time for each algorithm differs and hence involves an unequal FIFOs occupancy. Therefore, the application used has to be naturally unbalanced to thoroughly analyze the problem. The problem at hand is to optimally distribute the limited on chip embedded memory among the embedded processors cache memories (instruction, data) and the communication FIFOs while optimizing execution time and area. The design space for this problem is specified in Table 2.

The possible number of different configurations is given by the product of the number of distinct configurations for each configurable architectural parameter. Each cache memory may have up to 4 different sizes and each FIFO up to 8 different sizes. The total design space represents $(4 \times 4 \times 8 \times 8)^4 = 2^{40}$ configurations. If each configuration evaluation would require 1 second, the total evaluation time would be 34 865 years of evaluation. Clearly an exhaustive evaluation technique is unfeasible and multiobjective optimization techniques are able to efficiently prune this design space while simulation is clearly outperformed by direct execution on large scale FPGA devices.

5.2. Alpha-data environment

For the implementation of MOCDEX we used the alpha-data hardware and software environment.

5.2.1. Alpha data hardware environment

The alpha-data hardware environment described in Figure 7 is composed by (1) the ADC-PMC and (2) the ADM-XRC-II. The ADC-PMC is a dual PMC adapter for PCI. It supports 64-bit 66 MHz primary and secondary PCI via an Intel 21154 PCI-PCI bridge device. The ADM-XRC-II is a high performance reconfigurable PMC (PCI mezzanine card) based on the Xilinx Virtex-II range of platform FPGAs. Features include high-speed PCI interface, external memory, high-density I/O, programmable clocks, temperature monitoring, battery backed encryption, and flash boot facilities.

On board clock generator provides a synchronous local bus clock for the PCI interface and the Xilinx Virtex-II FPGA. A second clock is provided to the Xilinx Virtex-II FPGA for user applications and can be free running or stepped under software control. Both clocks are programmable and can be used by the Virtex clock. The user clock has a maximum value of 100 MHz. The ADM-XRC-II uses a Xilinx XC2V8000-6 FF1152 device [22] whose characteristics are described Table 3.

5.2.2. Alpha-data software environment

The ADM-XRC SDK is a set of resources including an application-programing interface (API) intended to assist the user in creating an application using one of Alpha-data's ADM-XRC range of reconfigurable coprocessors. The API

Table 4: ADM XRC SDK API functions.

Group	Application
Initialization	ADMXRC2_CloseCard ADMXRC2_OpenCard ADMXRC2_OpenCardByIndex ADMXRC2_SetSpaceConfig
FPGA configuration through PCI	ADMXRC2_ConfigureFromBuffer ADMXRC2_ConfigureFromBufferDMA ADMXRC2_ConfigureFromFile ADMXRC2_ConfigureFromFileDMA ADMXRC2_LoadBitstream ADMXRC2_UnloadBitstream
Data transfer PC = FPGA board	ADMXRC2_BuildDMAModeWord ADMXRC2_DoDMA ADMXRC2_DoDMAImmediate ADMXRC2_MapDirectMaster ADMXRC2_Read ADMXRC2_ReadConfig ADMXRC2_SetupDMA ADMXRC2_SyncDirectMaster ADMXRC2_UnsetupDMA ADMXRC2_Write ADMXRC2_WriteConfig
Interrupt handling	ADMXRC2_RegisterInterruptEvent ADMXRC2_UnregisterInterruptEvent

makes use of a device driver that is normally not directly accessed by the user's application. The API library described in Table 4 takes care of open, close, and device I/O control calls to the driver. The ADM-XRC SDK is designed to be thread-safe. Table 4 describes the main API functions which allow initializing the board, configuring the FPGA though the PCI bus, and transfering data between the FPGA and the host computer and the interrupt handling.

Clearly since MOCDEX explore the design space by implementing on FPGA new multiprocessor configurations the FPGA is reconfigured through the PCI bus from the main program by executing the ADM-XRC SDK FPGA reconfiguration API using the bitfile generated from EDK synthesis and place and route. Resulting execution number of cycles are provided as well through the PCI bus to the host using ADM-XRC SDK data transfer API.

5.3. *Xilinx EDK tools*

The embedded development kit (EDK) bundle is an integrated software solution for designing embedded processing systems.

Table 5 and Figure 8 describe the use of each configuration file in the process of hardware platform generation, software platform generation, and software application and creation.

The MHS file defines the system architecture, peripherals, and embedded processors. It also defines the connectivity of the system, the address map of each peripheral in the system, and configurable options for each peripheral. The MHS file can be defined through XPS Gui wizards. However for the time being Xilinx wizards do not allow the design of multiprocessors platforms and therefore they should be defined directly in the MHS file. It is clear that in the purpose of design space exploration of multiprocessor architecture the MHS file is the prime target of modifications. Changing parameters value in the MHS file generates a new multiprocessor configuration and invoking the XPS tool in no window mode from a main program allows the generation of the multiprocessor netlist. Table 6 provides examples of MHS file parts.

5.4. *Exploration flow description*

The proposed automatic design flow described in Figure 5 can be applied in the framework of Xilinx EDA tools and the Alpha-data environment. The flow is mainly composed of 3 parts: (1) architecture design space exploration engine (DSE), (2) physical design, and (3) FPGA platform PCI board. The architecture design space exploration part controls the whole flow and runs on a host PC. First based on the user specified design space parameters and parameters range, the DSE specifies the architectural parameters of the multiprocessors configurations to be evaluated then translates those parameters into platform EDA design tool input file specifications. In our case,

(1) *MOCDEX for Xilinx FPGA platform*,
(2) generate random population of MPSoC configurations (caches and FSL variations),
(3) for all configurations,
    (a) generate hardware/software platform specification files (mhs, mpd, pao, mss, mld, mdd, files),
    (b) generate through Xilinx system XPS and Xilinx IPs HW/SW model of the MPSOC,
    (c) synthesize/place and route MPSOC configuration using Xilinx ISE 6.3,
    (d) record place and route reports generated from Xilinx ISE 6.3,
    (e) download configuration file on FPGA Alpha-data platform using ADM-XRC SDK API,
    (f) execute MPSOC configuration and record execution clock cycles using ADM-XRC SDK API,
    (g) rank the solution,
(4) generate new population using NSGA-II algorithm,
(5) is the Pareto front satisfactory or the number of generations reached if no goto 3?
(6) final Pareto front MPSOC configurations available for selection.

The Xilinx system EDA tools Xilinx platform studio (XPS) is ran in no window mode with all batch commands launched from a *C* main program. Those input file specifications are used to control the physical design part of the implementation by synthesizing, placing, and routing the multiprocessor configurations onto FPGA platform devices. The generated

Table 5: EDK specifications files.

Files	Description	Comments
MHS	Microprocessor hardware specification	The MHS defines the hardware component
MSS	Microprocessor software specification	The MSS contains directives for customizing libraries, drivers, and file systems
MDD	Microprocessor driver definition	An MDD file contains directives for customizing software drivers
MPD	Microprocessor peripheral definition	The MPD defines the interface of the peripheral
MLD	Microprocessor library definition	the MLD contains directives for customizing software libraries and operating systems
PAO	Peripheral analyze order	Contains a list of HDL files that are needed for synthesis, and defines the analyze order for compilation.

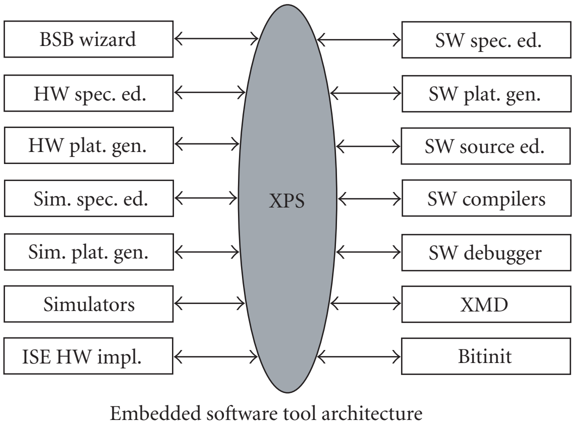


Figure 8: Xilinx EDK (XPS Xilinx platform studio).

FPGA configuration bitstream is downloaded on the FPGA device for execution and performance evaluation of the multiprocessor. The board hosting the FPGA device is an Alphadata PCI FPGA board [3]. The implementation area and resources of the multiprocessor configurations are provided by the design automation tools composing part (2) while performance results in number of clock cycles are obtained from the actual execution of the multiprocessor configurations. These informations are automatically fed back to the DSE engine which runs on the host through the PCI bus.

The number of cycles are obtained directly from the execution, thanks to a timer connected to the MicroBlaze (MB0) OPB bus, which counts the number of clock cycles. After that, the execution time results are communicated to the host PC using an IP which bridges the MicroBlaze OPB bus to the PCI host bus. These results (occupied slices, occupied BRAM, and the execution time) are then injected as feedback input to the evolutionary algorithm for the next generation run. For this work we initially executed two explorations where the first consisted of a population size of 22 individuals and 10 generations (242 implementations with the initialization generation).

6. EXPLORATION RESULTS

6.1. *Flow execution results*

Figures 10 and 11 describe the corresponding results of these implementations. Figure 10(b) represents Pareto solutions for the second exploration where we attempted to increase the population size to 30 individuals and the number of generations to 14 in order to observe the behavior of the evolutionary algorithm for bigger explorations. From the results of second exploration it is obvious that the algorithm is converging to optimal solutions showing that for larger population size and generation size, potential of convergence is increased in NSGA-II algorithm as was expected. From the two preceding exploration flow executions, it appears as expected since we focused on embedded memories that the number of occupied slices does not vary much across multiprocessor configurations. However the variations are much more significant concerning both the number of occupied BRAMs and the execution time. So we decided to continue the execution of the proposed exploration flow in order to see its evolution.

TABLE 6: MHS file parts: Microprocessor IP, FSL IP, BRAM controller IP.

MicroBlaze processor	FSL communication	BRAM controller
BEGIN MicroBlaze	BEGIN fsl_v20	BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = MicroBlaze_0	PARAMETER INSTANCE = fsl_v20_7	PARAMETER INSTANCE = ilmb_cntlr3
PARAMETER HW_VER = 3.00.a	PARAMETER C_FSL_DEPTH = 8	PARAMETER HW_VER = 1.00.b
PARAMETER C_FSL_LINKS = 2	PARAMETER HW_VER = 2.00.a	PARAMETER C_BASEADDR
BUS_INTERFACE MFSL0 = fsl_v20_2	PARAMETER C_EXT_RESET_HIGH = 0	= 0 × 00000000
BUS_INTERFACE SFSL0 = fsl_v20_1	PARAMETER C_IMPL_STYLE = 1	PARAMETER C_HIGHADDR
BUS_INTERFACE DLMB = dlmb0	PARAMETER C_USE_CONTROL = 0	= 0 × 00003fff
BUS_INTERFACE ILMB = ilmb0	PORT SYS_Rst = lreseto_l	BUS_INTERFACE SLMB = ilmb3
BUS_INTERFACE DOPB = mb_opb0	PORT FSL_Clk = lclk	BUS_INTERFACE BRAM_PORT
BUS_INTERFACE IOPB = mb_opb0	PORT FSL_M_Clk = lclk	= ilmb_port3
PORT INTERRUPT = Interrupt 0	PORT FSL_S_Clk = lclk	END
PORT CLK = lclk	END	
END		

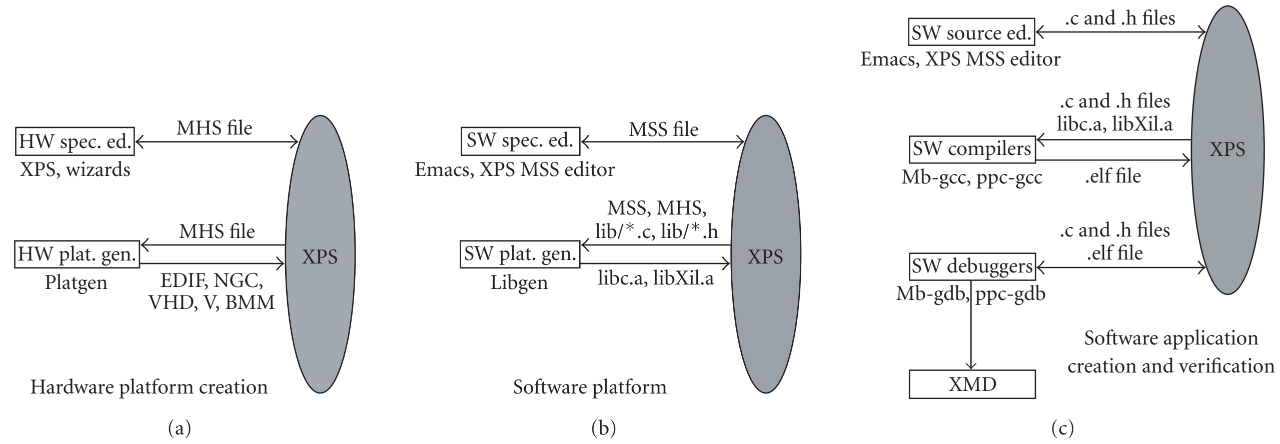


FIGURE 9: Xilinx EDK. (a) Hardware platform generation. (b) Software platform. (c) Simulation and verification.

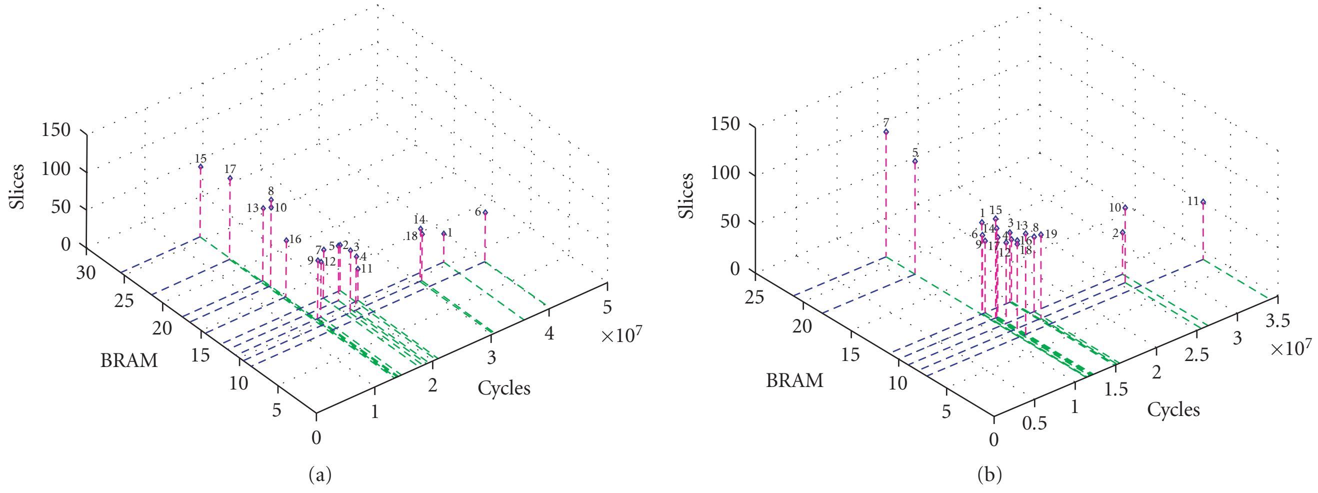
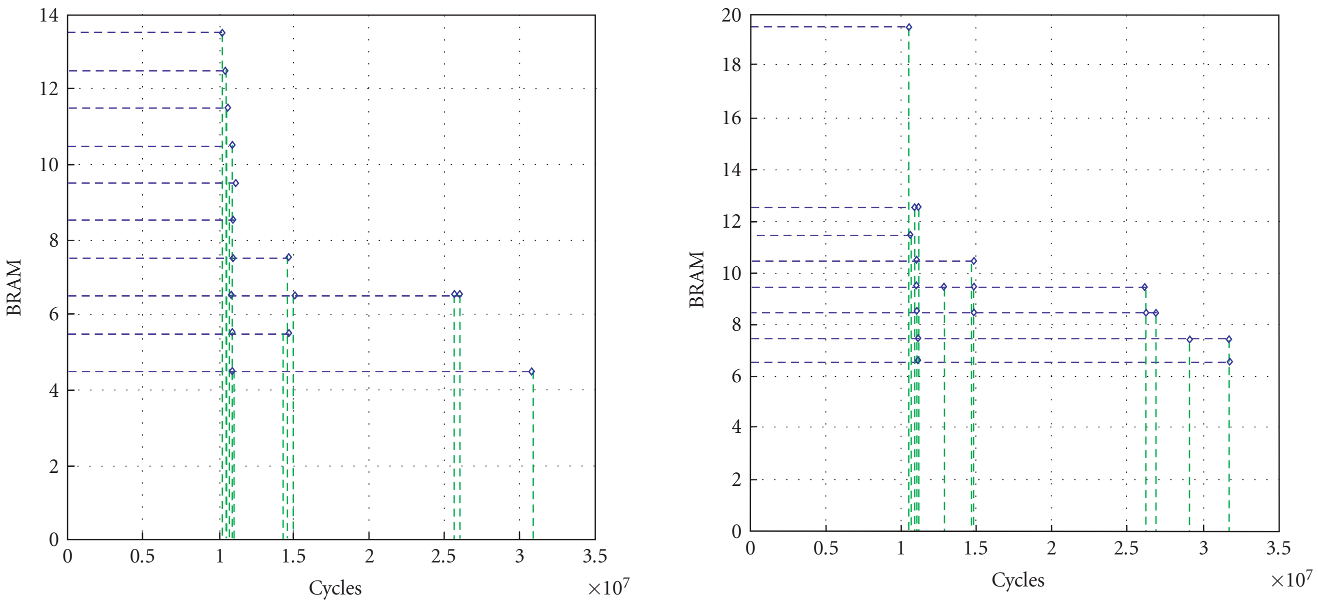


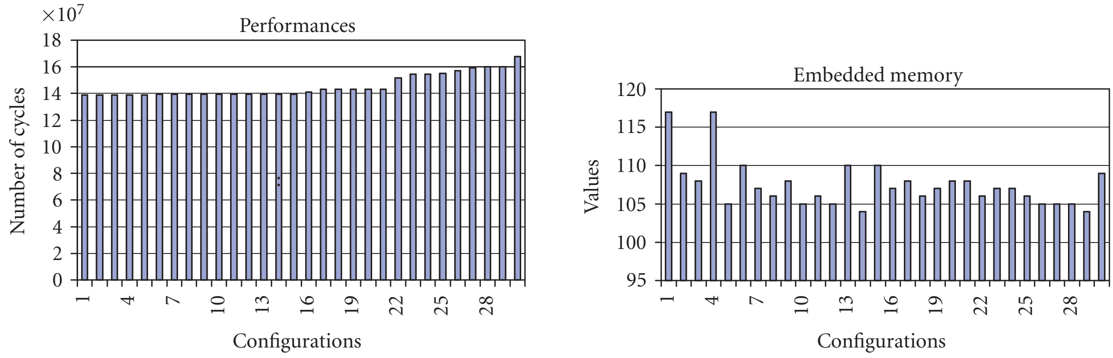
FIGURE 10: (a) For 10 generations-popsize = 22. (b) For 14 generations-popsize = 30.



(a)

(b)

Figure 11: (a) For 30 generations-popsize = 30. (b) For 60 generations-popsize = 30.



(a)

(b)

Figure 12: Pareto front. (a) Pareto front performance distribution. (b) Pareto front BRAM distribution.

For this second part of the exploration, we fixed the population size to 30 individuals and changed the number of generation to 30 and finally 60 generations. The results for each execution are, respectively, described in Figures 11(a) and 11(b). From these different figures we can clearly observe that the NSGA-II evolutionary algorithm tends to converge to the optimal Pareto solutions front which proves the correct implementation of the algorithm. The figures show different execution times for the same BRAM occupation meaning that using more BRAM will not systematically result in performance improvements.

However, to achieve better results BRAM resources need to be well distributed among the IPs where it would be used for getting optimal resource utilization.

Figure 12 shows the distribution of performance in the final Pareto front and clearly few configurations demonstrate superior performance while BRAM distribution for the same front demonstrates an uneven use of BRAM. This clearly shows the impact of BRAM careful distribution.

Examples of final Pareto front configurations are given in Table 7. The configurations chosen represent, respectively, 69.64%, 61.90%, and 64.88% of all BRAM resources. 11.11% BRAM reduction is obtained in the second configuration for a 0.004% increase in execution time while a 6.8% BRAM reduction is obtained in the third configuration for a 0.009% increase in the execution time.

6.2. *Flow execution time*

The results achieved in the previous section required the performance evaluation of 3120 different multiprocessor

Table 7: The design space associated with those parameters (74 × 118, thus 514 675 673 281 different configurations) requires 16 321 years of simulation for 1 minute simulation per configuration.

(a) Cycles: 138 974 816 BRAM: 109.

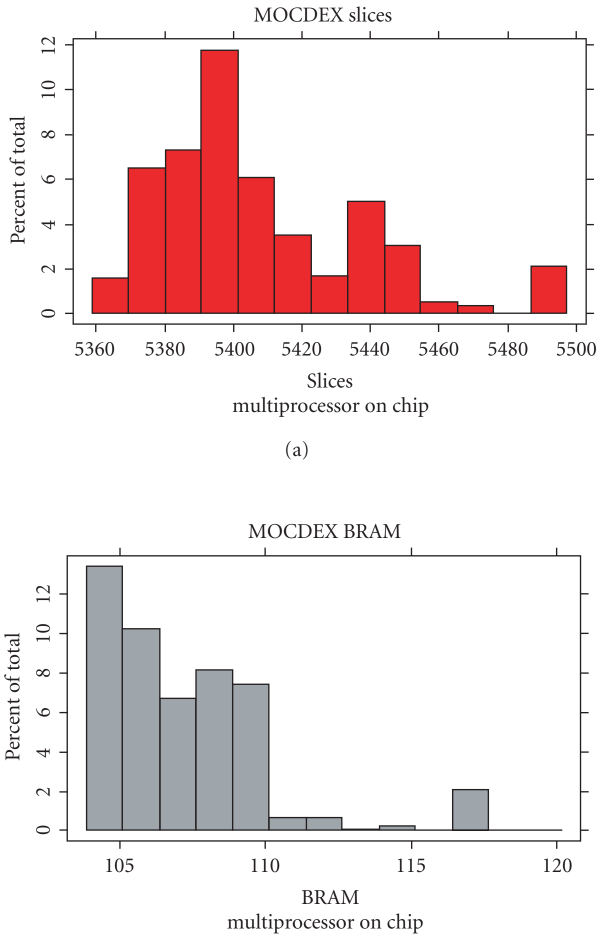
Procs	FSL1Out	FSL2Out	D-Cache	I-Cache
MB0	2048	2048	1024	4096
MB1	512	512	1024	1024
MB2	2048	512	2048	2048
MB3	1024	1024	4096	4096

(b) Cycles: 138 844 064 BRAM: 117.

Procs	FSL1Out	FSL2Out	D-Cache	I-Cache
MB0	2048	128	2048	2048
MB1	256	32	2048	512
MB2	512	16	4096	512
MB3	1024	32	512	2048

Table 8: Flow execution time direct execution versus simulation.

Flow main steps	Functions	Time
Multi-objective evolutionary algorithm (ms)	Indi. Gene.	190
	Obj functions eval.	293
	Selection	0.116
	Crossover	0.033
	Mutation	1.118
Synthesis (sec)	Synthesis	523.503
	P and *R*	655.174
	P/*R* & Bitgen	797.856
Evaluation	Exploration 60 × 30 Sim. 64 × 64 Direct exec. 256 × 256	2250 days 1.39 hour



(a)

(b)

Figure 13: Explored design space. (a) Slices histogram. (b) BRAM histogram.

on-chip configurations. These evaluations have been cycle-accurate after actual implementation on single-chip large scale FPGA devices. Contrary to traditional board-based multiprocessor, multiprocessors on chip are implemented on single chip; and due to the complexity of these architectures and the scale of the target devices, it is not possible to overlook the impact of place and route on the number of cycles required for various operations and on the cycle time.

It results from this fact that comparing different multiprocessors on chip configurations on the number of execution cycles is meaningless if one does not take into account the impact of place and route on each distinct configuration resulting from actual implementation. From this point mainly two alternatives exist: (1) post place and route simulation which will accurately represent the multiprocessor on chip behavior, and (2) emulation through direct execution. We conducted cycle accurate simulations using a powerful multi-language (SystemC, VHDL, Verilog-HDL) simulator ModelSim 6.0. Indeed, ModelSim 6.0 can handle large and complex designs and allow their simulation in a post-synthesis and post-place and route modes. Table 8 describes the very important time savings while using direct execution instead of simulation. Simulation would require 2250 days of simulation versus 1.39 hour for direct execution.

In order to reach the same evaluation speed at this level of accuracy it would require a compute farm (grid computing) of well over 25 000 workstations. The proposed flow execution time is obviously very competitive with regard to SystemC approaches [20, 21] for platform-based design. Similar observations have been drawn for embedded processors design space exploration [18, 19].

7. EXPLORED DESIGN SPACE STATISTICAL ANALYSIS

If we analyze in detail the complexity landscape of such a design space exploration we obtain the configurations distribution found in Figures 13 and 14. Clearly from these histograms we see that slices, BRAM, and performance (execution time) distributions in the explored design space are very different and demonstrate that the design space exploration was not confined in a limited subspace but explored a large diversity of multiprocessor configurations. The explored design landscape is given in Figure 15.

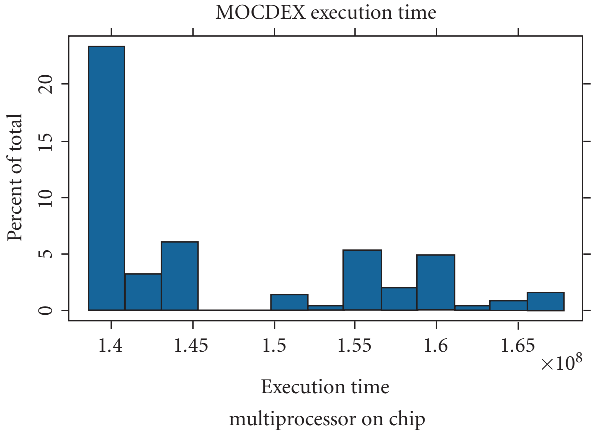


Figure 14: Explored design space execution time histogram.

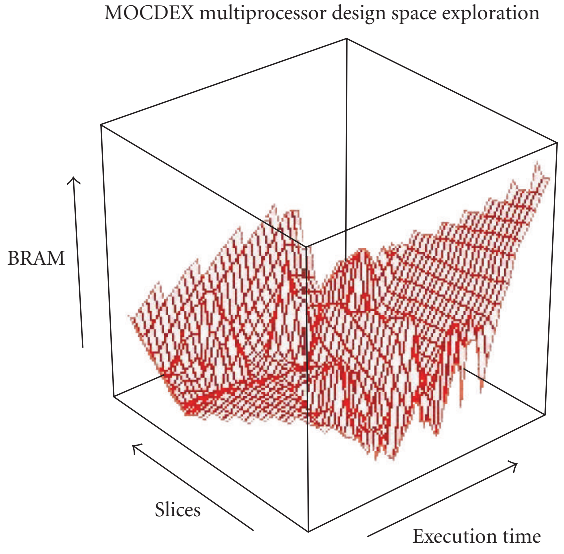


Figure 15: MOCDEX explored design space.

Figure 15 demonstrates the complexity of the design landscape and emphasizes the need to match this complexity with appropriate applied mathematics optimization techniques.

8. CONCLUSION

The design complexity of multiprocessors on chip requires efficient design methodologies. We propose in this paper a novel technique which fully integrates architectural design space exploration with design automation tools, where all area and performance results are obtained from actual post-synthesis place and route and actual execution on large scale FPGA platforms. To the best of our knowledge, our work is the first to fully integrate and therefore close the gap between design automation tools and architecture design space exploration technique in a multiobjective constraints paradigm with actual execution for all multiprocessor on chip configurations explored during the design space exploration process. It is important to note that actual execution reduces exploration time and can be exploited for either reducing design cycle time (i.e., TTM) and/or exploring even larger design space by including additional parameters. This work can be easily extended to include more parameters at various abstraction levels from architecture to circuit allowing interesting tradeoffs between usually uncorrelated various abstraction levels in the general design flow.

REFERENCES

[1] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*, Springer, New York, NY, USA, 2002.

[2] C. A. C. Coello, D. V. Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, vol. 5 of *Genetic Algorithms and Evolutionary Computation*, Kluwer Academic, Dordrecht, The Netherlands, 2002.

[3] Alpha-Data, ADM-XRC-II PCI mezzanine card, http://www.alpha-data.com.

[4] D. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann, San Francisco, Calif, USA, 1999.

[5] A. A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips*, Morgan Kaufman, San Francisco, Calif, USA, 2004.

[6] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 518–523, Las Vegas, Nev, USA, June 2001.

[7] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha, "Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors," in *Proceedings of the 18th IEEE International Conference on VLSI Design*, pp. 551–556, Kolkata, India, January 2005.

[8] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, "An automated exploration framework for FPGA-based soft multiprocessor systems," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES '05)*, pp. 273–278, New York, NY, USA, September 2005.

[9] N. K. Bambha and S. S. Bhattacharyya, "Joint application mapping/interconnect synthesis techniques for embedded chip-scale multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 99–112, 2005.

[10] Xilinx, Embedded system tools guide, http://www.xilinx.com/ise/embedded/edk_docs.htm.

[11] Xilinx microblaze soft core processor, http://www.xilinx.com/ise/embedded/mb_ref guide.

[12] I. Aouadi, R. B. Mouhoub, and O. Hammami, "System on a programmable chip oriented JPEG-2000 entropy coder implementation for multimedia embedded systems," in *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE '05)*, pp. 447–448, Las Vegas, Nev, USA, January 2005.

[13] R. B. Mouhoub, I. Aouadi, and O. Hammami, "System on programmable chip platform based design of JPEG- 2000 entropy coder," in *Proceedings of the 12th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI '04)*, pp. 103–106, Kanazawa, Japan, October 2004.

[14] Xilinx Fast Simplex Link IP, http://www.xilinx.com.

[15] C. A. C. Coello, "An updated survey of GA-based multiobjective optimization techniques," *ACM Computing Surveys*, vol. 32, no. 2, pp. 109–143, 2000.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] M. T. Jensen, "Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, 2003.

[18] K. Ghali and O. Hammami, "Embedded processor characteristics specification through multiobjective evolutionary algorithms," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '03)*, vol. 2, pp. 907–912, Rio de Janeiro, Brazil, June 2003.

[19] K. Ghali and O. Hammami, "Embedded processors optimization with hardware in the loop," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '04)*, vol. 1, pp. 561–564, Ajaccio, France, May 2004.

[20] F. Fummi, S. Martini, G. Perbellini, and M. Poncino, "Native ISS-SystemC integration for the co-simulation of multiprocessor SoC," in *Proceedings of the IEEE Conference and Exhibition on Design, Automation and Test in Europe (DATE '04)*, vol. 1, pp. 564–569, Paris, France, February 2004.

[21] F. Ghenassia, *Transaction-Level Modeling with SystemC TLM Concepts and Applications for Embedded Systems*, Springer, New York, NY, USA, 2005.

[22] Xilinx Virtex-II Platform FPGA, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_platform_fpgas/index.htm

Riad Ben Mouhoub received the Electronics Engineering degree in 2002 from the University of Algiers USTHB. He also received a Master degree in electronic systems and data processing from the University of Paris Sud Orsay in 2003. He currently holds a Doctorate position in electrical and computer engineering from the University of Paris Sud in the Department of Electronics and Computer Engineering at École Nationale Supérieure de Techniques Avancées in Paris (ENSTA). His research interests include design automation, design methodologies for multiprocessor system on programmable chips (MPSoPC), and NoC synthesis. He is a Student Member of the IEEE.

Omar Hammami is an Associate Professor at ENSTA/DGA since 2000. Prior to that he was Assistant Professor from 1991 to 1993 with ENSEEIHT, Toulouse, and Associate Professor with the University of Aizu, Japan, from 1993 to 2000. He received his Ph.D. degree in computer science and electrical engineering from Paul Sabatier University, Toulouse, in 1993 and has since worked in the field of circuits, system level design methodologies, embedded parallel architectures, and system on chip (SOC) for multimedia and wireless communications. He has been involved in numerous international and national research and industrial projects in those areas and have been funded by various government and funding agencies. He is a regular reviewer for various journals (IEEE, EURASIP, etc.) and conferences as Program Committee Member.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 98045, Pages 1–11
DOI 10.1155/ES/2006/98045

Rapid Energy Estimation for Hardware-Software Codesign Using FPGAs

Jingzhao Ou[1] and Viktor K. Prasanna[2]

[1] *DSP Design Tools and Methodologies Group, Xilinx, Inc., San Jose, CA 95124, USA*
[2] *Veterbi School of Engineering, University of Southern California, Los Angeles, CA 90089, USA*

Received 1 January 2006; Revised 25 May 2006; Accepted 19 June 2006

By allowing parts of the applications to be executed either on soft processors (as software programs) or on customized hardware peripherals attached to the processors, FPGAs have made traditional energy estimation techniques inefficient for evaluating various design tradeoffs. In this paper, we propose a high-level simulation-based two-step rapid energy estimation technique for hardware-software codesign using FPGAs. In the first step, a high-level hardware-software cosimulation technique is applied to simulate both the hardware and software components of the target application. High-level simulation results of both software programs running on the processors and the customized hardware peripherals are gathered during the cosimulation process. In the second step, the high-level simulation results of the customized hardware peripherals are used to estimate the switching activities of their corresponding register-transfer/gate level ("low-level") implementations. We use this information to employ an instruction-level energy estimation technique and a domain-specific energy performance modeling technique to estimate the energy dissipation of the complete application. A Matlab/Simulink-based implementation of our approach and two numerical computation applications show that the proposed energy estimation technique can achieve more than 6000x speedup over low-level simulation-based techniques while sacrificing less than 10% estimation accuracy. Compared with the measured results, our experimental results show that the proposed technique achieves an average estimation error of less than 12%.

1. INTRODUCTION

The integration of multimillion gate configurable logic and various heterogeneous hardware components, such as embedded multipliers and memory blocks, offers FPGAs exceptional computational capabilities. Soft processors, which are RISC processors realized using configurable resources available on FPGA devices, have become popular for embedded system development. Examples of such soft processors include Nios from Altera [1], a SPARC architecture-based LEON3 from Gaisler [2], an ARM7 architecture-based CoreMP7 from Actel [3], and MicroBlaze from Xilinx [4]. As shown in Figure 1, for the development of FPGA-based embedded systems, parts of the application can be executed either on soft processors as programs or on customized hardware peripherals attached to the processors. Customized hardware peripherals are efficient for executing many data intensive computations. On the other hand, processors are efficient for executing many control and management functions, and computations with tight data dependency between steps (e.g., recursive algorithms). The use of soft processors leads to more compact designs and thus requires a much smaller amount of hardware resources than that of customized hardware peripherals. Having a compact design that fits into a small FPGA device can effectively reduce static energy dissipation [5]. The ability to make hardware and software design tradeoffs has made FPGAs an attractive choice for implementing a wide range of embedded systems.

Energy efficiency is an important performance metric for many embedded systems, such as software-defined radio (SDR) systems. In SDR systems, dissimilar and complex wireless standards (e.g., GSM, IS-95) are processed in a single adaptive base station, where a large amount of data from the mobile terminals present high computational requirements. State-of-the-art RISC processors and DSPs are unable to meet the signal processing requirements of these base stations. Power consumption minimization has become a critical issue for base stations, due to the high computational requirement that leads to high energy dissipation in inaccessible and distributed base station locations. FPGAs stand out as an attractive choice for implementing various SDR functions due to their high performance, low power

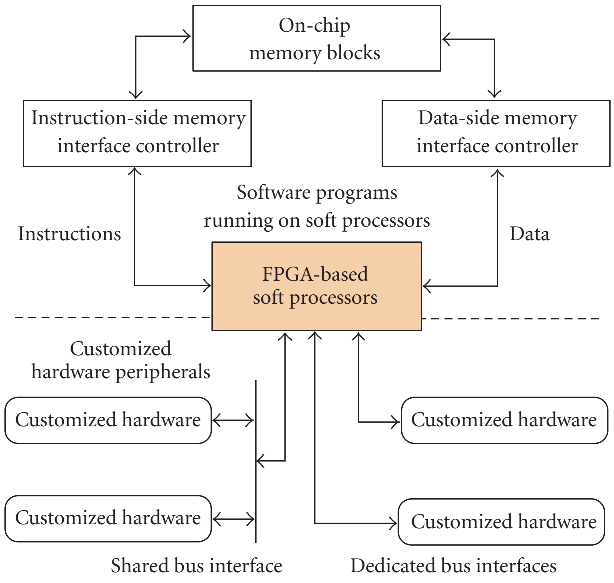

On-chip memory blocks
Instruction-side memory interface controller
Data-side memory interface controller
Software programs running on soft processors
Instructions
Data
FPGA-based soft processors
Customized hardware peripherals
Customized hardware
Customized hardware
Customized hardware
Customized hardware
Shared bus interface
Dedicated bus interfaces

Figure 1: FPGA-based hardware-software codesign.

dissipation per computation, and reconfigurability [6]. Many hardware-software mappings and application implementations are possible on modern FPGA devices. The various hardware-software mappings and implementations can result in a significant variation in energy dissipation. Therefore, being able to obtain the energy dissipation of these different mappings and to evaluate implementations of the applications rapidly is crucial to energy efficient application development using FPGAs.

In this paper, we consider an FPGA device configured with a soft processor and several customized hardware peripherals attached to it. The processor and the hardware peripherals communicate with each other through specific bus protocols. The target application is decomposed into a set of tasks. Each task can be mapped onto either a soft processor (i.e., software), or a specific customized hardware peripheral (i.e., hardware), for execution. A specific mapping and execution schedule of the tasks are given. For tasks executed on customized hardware peripherals, their implementations are described using high-level modeling environments (e.g., MILAN [7], Matlab/Simulink [8], and Ptolemy [9]). For tasks executed on the soft processor, the software implementations are described as C code and compiled using the appropriate C compiler. One or more sets of sample input data are also given. Under these assumptions, our objective is *to rapidly and accurately (within about 10%) obtain the energy dissipation of the complete application.*

There are two major challenges for rapid and accurate energy estimation for hardware-software codesigns using FPGAs. One challenge is that state-of-the-art energy estimation tools are based on low-level (register transfer level and gate level) simulation results. While these low-level energy estimation techniques can be accurate, they are time-consuming and would be intractable when used to evaluate the energy performance of the different FPGA implementations. This is especially true for software programs running on soft processors. Considering the designs described in Section 5, the simulation of ~ 2.78 milliseconds execution time of a matrix multiplication application using post place-and-route simulation models takes about 3 hours in ModelSim [10]. Using XPower [4] to analyze the simulation file that records the switching activities of low-level hardware components and to calculate the overall energy dissipation requires an additional hour. The other challenge is that high-level energy performance modeling, which is crucial for rapid energy estimation, is difficult for FPGA designs. Lookup tables connected through programmable interconnect, the basic elements of FPGAs, can realize a wide range of different hardware architectures. They lack a single high-level model found in general purpose processors, which can capture the energy dissipation behavior of the various possible architectures.

As discussed in Section 2, while instruction-level energy estimation techniques can provide rapid energy estimates of processor cores with satisfactory accuracy, they are unable to account for the energy dissipation of customized instructions and tightly coupled hardware peripherals. More detailed energy performance models are required to capture the energy behavior of the customized instructions and hardware peripherals.

We propose a high-level simulation-based two-step rapid energy estimation technique for hardware-software codesign using FPGAs. In the first step, a high-level modeling environment is created to combine the corresponding high-level abstractions that are suitable for describing the hardware and software execution platforms. Within this high-level modeling environment, hardware-software cosimulation is performed to evaluate a cycle-accurate high-level behavior of the complete system. Instruction profiling information of the software execution platform and high-level activity information of the customized hardware peripherals are gathered during the cycle-accurate cosimulation process. The switching activities of the corresponding low-level implementations of the customized hardware peripherals are then estimated. In the second step, by utilizing the instruction profiling information, an instruction-level energy estimation technique is employed to estimate the energy dissipation of software execution. Also, by utilizing the estimated low-level switching activity information, a domain-specific modeling technique is employed to estimate the energy dissipation of hardware execution. The energy dissipation of the complete system is obtained by summing the energy dissipation of hardware and software execution.

A Matlab/Simulink-based implementation of the proposed energy estimation technique and two widely used numerical computation applications are used to demonstrate the effectiveness of our approach. For various implementations of these two applications, our high-level cosimulation technique achieves more than a 6000x speedup versus techniques based on low-level simulations. Such speedups can directly lead to a significant speedup in energy estimation. Compared with low-level techniques, our high-level simulation approach achieves an average estimation error of less than 10%. Compared with experimentally measured results, our approach achieves an average estimation error of less than 12%.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes our two-step rapid energy estimation technique. An implementation of our technique based on a state-of-the-art high-level modeling environment is presented in Section 4. The design of two numerical computation applications is described in Section 5. We conclude in Section 6.

2. RELATED WORK

Energy estimation techniques for FPGA designs can roughly be divided into two categories. One category is based on low-level simulation, which is employed by tools such as Quartus II [1], XPower [4], and the tool developed by Poon et al. [11]. In low-level simulation-based energy estimation techniques, the user generates low-level implementations of the FPGA designs. Simulation is performed based on the low-level implementations to obtain the switching activity of the low-level hardware components used in the FPGA design (e.g., basic configurable units and programmable wires). Each of the low-level hardware components is associated with an energy function that captures its energy behavior with different switching activities. Using the low-level simulation results and the low-level energy functions, the user can estimate the energy dissipation of all low-level components. The energy dissipation of the complete application is calculated as the sum of the energy dissipation of the low-level hardware components. Low-level estimation techniques are inefficient for FPGA-based hardware-software codesign. The creation of a low-level implementation includes synthesis, placement, and routing. This sequence forms a lengthy process. Simulations based on low-level implementations are very time consuming. This is especially true for the simulation of software.

The other category of energy estimation techniques is based on high-level energy models. The FPGA design is represented as a few high-level models interacting with each other. The high-level models accept parameters that have a significant impact on energy dissipation. These parameters are predefined or provided by the application designer. This technique is used by tools such as the RHinO tool [12] and the web power analysis tools from Xilinx [13]. While energy estimation using this technique can be fast, as they avoid time-consuming low-level simulation, its estimation accuracy varies among applications and application designers. One reason is that different applications demonstrate different energy dissipation behaviors. We show in [14] that using predefined parameters for energy estimation results in energy estimation errors as high as 32% for input data with different statistical characteristics. The other reason is that requiring the application designer to provide these important parameters would demand a deep understanding of the energy behavior of the target devices and applications, which can prove to be very difficult in practice. This approach is not suitable for estimating the energy estimation of software execution as instructions with different energy dissipations are executed on soft processors.

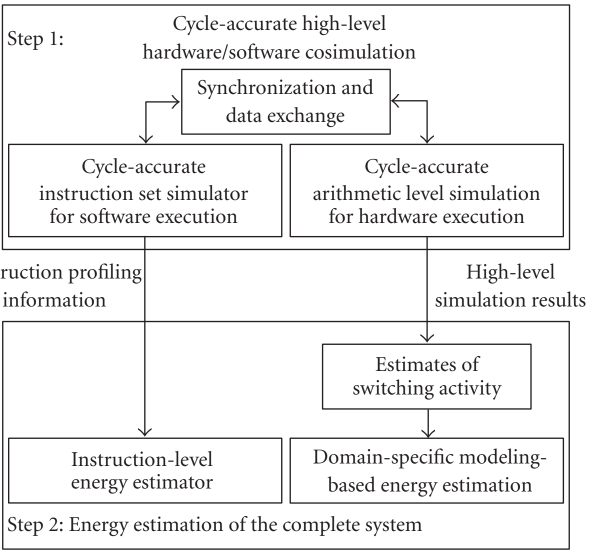

Step 1:
Cycle-accurate high-level hardware/software cosimulation
Synchronization and data exchange
Cycle-accurate instruction set simulator for software execution
Cycle-accurate arithmetic level simulation for hardware execution
Instruction profiling information
High-level simulation results
Estimates of switching activity
Instruction-level energy estimator
Domain-specific modeling-based energy estimation
Step 2: Energy estimation of the complete system

FIGURE 2: The two-step energy estimation approach.

For software execution on processors, instruction-level energy estimation is an effective technique for obtaining energy dissipation. This technique is used by several popular commercial and academic processors, such as *Wattch* [15], *JouleTrack* [16], and *SimplePower* [17]. *JouleTrack* estimates the energy dissipation of software programs on StrongARM SA-1100 and Hitachi SH-4 processors. *Wattch* and *SimplePower* estimate the energy dissipation of an academic *SimpleScalar* processor. We proposed an instruction-level energy estimation technique in [18], which can provide rapid and accurate energy estimation for FPGA-based soft processors. These energy estimation frameworks and tools target processors with fixed architectures. They do not account for the energy dissipated by customized hardware peripherals and communication interfaces. Thus, they are unable to provide energy estimation of combined hardware-software designs targeted to FPGA platforms. Low-level energy models are required for customized hardware peripherals.

3. OUR APPROACH

Our two-step approach for the rapid energy estimation of the hardware-software designs using FPGAs is illustrated in Figure 2. The two energy estimation steps are discussed in detail in the following sections.

3.1. *Step 1: high-level cosimulation*

In the first step, a high-level cosimulation is performed to simultaneously simulate hardware and software execution on a cycle-accurate basis. Note that we use "cycle-accurate" to denote that on both positive and negative edges of the simulation clock, the behavior of the high-level simulation models matches the corresponding low-level implementations. Other timing information between the clock edges (e.g., the glitches), as well as the logic and path delays between the

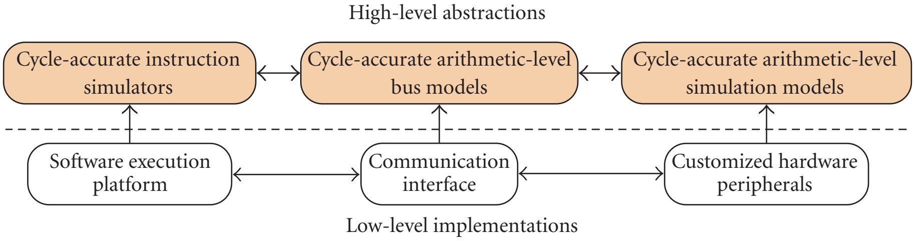

High-level abstractions
Cycle-accurate instruction simulators
Cycle-accurate arithmetic-level bus models
Cycle-accurate arithmetic-level simulation models
Software execution platform
Communication interface
Customized hardware peripherals
Low-level implementations

Figure 3: Architecture of the cycle-accurate high-level cosimulation environment.

hardware components, is not accounted for in the high-level simulation. There are two major advantages of maintaining cycle accuracy during cosimulation. One advantage is that by ignoring the low-level implementation and sacrificing some timing information, the high-level cosimulation framework can greatly speed up the simulation. This greatly speeds up the energy estimation process. Most importantly, the simulation results gathered during the high-level cosimulation process can be used to estimate the switching activities of the corresponding low-level implementations, and can be used in the second step of the energy estimation process to derive rapid and accurate energy estimates of the complete system.

It can be argued that urging cycle accuracy early, the design process prevents efficient design space exploration as cycle accuracy is usually not required in early hardware-software partitioning and in the development of software drivers. Our cosimulation framework only maintains cycle accuracy at the instruction level for software execution and arithmetic level for hardware execution. The cosimulation environment presents a view similar to the combination of the *architects view* and *programmers view* in transaction level modeling (TLM). Kogel et al. points out in [19] that *"there is usually no need for 100% timing accuracy since the impact of an architecture change is on a much bigger scope than a single clock cycle. Still an accuracy of 70–80% needs to be maintained to ensure the quality of the analysis results."* Many state-of-the-art high-level modeling environments for digital signal processing systems, control systems, and so forth, enforce such cycle accuracy in their modeling process. Examples include the concept of high-level simulation clocks within the Matlab/Simulink and Ptolemy modeling environments. Compared with System C implementations of the transaction-level models, our design and cosimulation framework is based on visual data-flow modeling environments and thus is more suitable for describing embedded systems.

The architecture of the cosimulation environment is illustrated in Figure 3. The low-level implementation of the FPGA execution platform consists of three major components: the *soft processor* (for executing programs), *customized hardware peripherals* (hardware accelerators for parallel execution of some specific computations), and *communication interfaces* (for exchanging data and control signals between the processor and the customized hardware components). High-level abstractions are created for each of the three major components. The high-level abstractions are simulated using their corresponding simulators. The hardware and software simulators are tightly integrated into our cosimulation environment and concurrently simulate the high-level behavior of the hardware-software execution platform. Most importantly, the simulation among the integrated simulators is synchronized at each clock cycle and provides cycle-accurate simulation results for the complete hardware-software execution platform. Once the high-level design process is completed, the application designer specifies the required low-level hardware bindings for the high-level operations (e.g., binding the embedded multipliers to multiplication arithmetic operations). Finally, register-transfer/gate level ("low-level") implementations of the complete platform with corresponding high-level behavior can be automatically generated based on the high-level abstraction of the hardware-software execution platforms.

3.1.1. Cycle-accurate instruction-level simulation of programs running on the processor

We employ cycle-accurate instruction-level simulation models to simulate the execution of the instructions on a soft processor. These simulation models provide cycle-accurate simulation information regarding the execution of the instructions of the target program. With MicroBlaze [4], for example, the cycle-accurate instruction-set simulator records the number of times that an instruction passes the multiple execution stages, as well as the status of the soft processor, on a cycle-accurate basis. Most importantly, as we show in Section 4.2.1, such cycle-accurate instruction-level information can be used to derive rapid and accurate energy estimation.

3.1.2. Cycle-accurate arithmetic level simulation of customized hardware peripherals

Arithmetic level simulation is performed to simulate the customized hardware peripherals attached to the processors. By "arithmetic level," we mean that only the arithmetic aspects of the hardware-software execution are captured by the coimulation environment. For example, low-level implementations of multiplication on Xilinx Virtex-II FPGAs can be realized using either slice-based multipliers or embedded multipliers.

3.1.3. Maintenance of cycle accuracy throughout the cosimulation process

For each simulation clock cycle, the high-level behavior of the complete FPGA hardware platform predicted by the cycle-accurate cosimulation environment should match with the behavior of the corresponding low-level implementation. When simulating the execution of a program on a soft processor, cycle-accurate cosimulation should take into account the number of clock cycles required for completing a specific instruction (e.g., the multiplication instruction of the MicroBlaze processor takes three clock cycles to finish) and the processing pipeline of the processor. Also, when simulating the execution of customized hardware peripherals, cycle-accurate simulation should take into account delays in the number of clock cycles caused by the processing pipelines within the customized hardware peripherals. Our high-level simulation environment ignores low-level implementation details, and only focuses on the arithmetic behavior of the designs. By doing so, the hardware-software cosimulation process can be greatly sped up. In addition, cycle accuracy is maintained between the hardware and software simulators during the cosimulation process. Thus, the instruction profiling information and the low-level switching activity information, which are used in the second step for energy estimation, can be accurately estimated from the high-level cosimulation process.

3.2. Step 2: rapid energy estimation

In the second step, the information gathered during the high-level cosimulation process is used for rapid energy estimation. The types and the numbers of instructions executed on soft processors are obtained from the cycle-accurate instruction simulation process. The instruction execution information is used to estimate the energy dissipation of the programs running on the soft processor. For customized hardware implementations, the switching activities of the low-level implementations are estimated by analyzing the switching activities of the arithmetic level simulation results. Then, with the estimated switching activity information, energy dissipation of the hardware peripherals is estimated by utilizing a domain-specific energy performance modeling technique proposed in [20]. Energy dissipation of the complete system is calculated as the sum of the energy dissipation of the software and hardware implementations.

3.2.1. Instruction-level energy estimation for software execution

An instruction-level energy estimation technique is employed to estimate the energy dissipation of the software execution on the soft processor. A per-instruction energy lookup table is created, which stores the energy dissipation of each type of instruction for the specific soft processor. The types and the number of instructions executed when the program is running on the soft processor are obtained during the high-level hardware-software cosimulation process. By querying the instruction energy lookup table, the energy dissipation of these instructions is obtained. The energy dissipation of the program is calculated as the sum of the energy dissipations of all of the instructions.

3.2.2. Domain-specific modeling-based energy estimation for hardware execution

The energy dissipation of the customized hardware peripherals is estimated through domain-specific energy performance modeling presented in [20]. Domain-specific modeling is proposed to address the challenge of high-level FPGA energy performance modeling. FPGAs allow for implementing designs using a variety of architectures and algorithms. These architectures and algorithms use a different amount of logic components and interconnect. While these tradeoffs offer a great design flexibility, they prevent energy performance modeling using a single high-level model. For example, matrix multiplication on an FPGA can employ a single processor or a systolic architecture. An FFT on an FPGA can adopt a radix-2-based or a radix-4-based algorithm. Each architecture and algorithm would have different energy dissipation.

Domain-specific modeling (DSM) is a hybrid (top-down followed by bottom-up) modeling approach. It starts with a top-down analysis of the algorithms and the architectures for implementing a kernel. Through top-down analysis, the various possible low-level implementations of the kernel are grouped into *domains*, depending on the architectures and algorithms used. This DSM technique enforce a high-level architecture for the implementations belonging to the same domain. With such enforcement, high-level modeling within the domain becomes possible. Analytical formulation of energy functions are derived within each domain to capture the energy behavior of the corresponding implementations. Then, a bottom-up approach is used to estimate the constants of these analytical energy functions for the identified domains through low-level sample implementations. This includes profiling individual system components through low-level simulations, hardware experiments, and so forth. These domain-specific energy functions are platform-specific. That is, the constants in the energy functions would have different values for different FPGA platforms. During the application development process, these energy functions are used for rapid energy estimation of hardware implementations belonging to a particular domain.

The domain-specific models can be hierarchical. The energy functions of a kernel can contain the energy functions of the subkernels that constitute the kernel. Characteristics of the input data (e.g., switching activities) can have considerable impact on energy dissipation and are also inputs to the energy functions. This characteristic information is obtained through low-level simulation, or through high-level cosimulation described in Section 4.1. See [20] for more details regarding the domain-specific modeling technique.

4. AN IMPLEMENTATION

To illustrate our approach, an implementation of our rapid energy estimation technique based on Matlab/Simulink is described in the following sections.

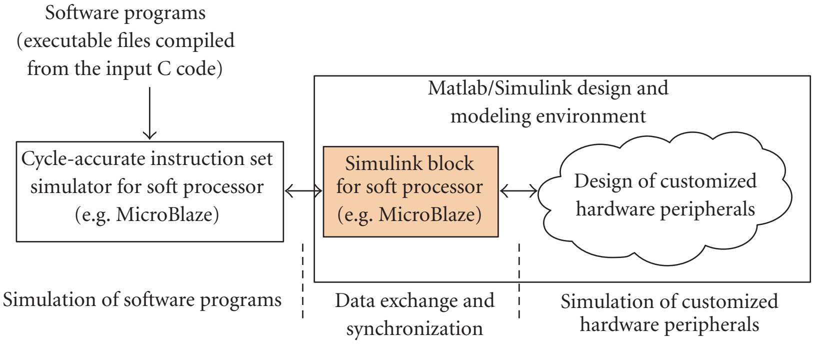


Figure 4: An implementation of the hardware-software cosimulation environment based on Matlab/Simulink.

4.1. Step 1: cycle-accurate high-level cosimulation

An implementation of the high-level cosimulation framework presented in Section 3.1 is shown in Figure 4. The four major functionalities of our Matlab/Simulink-based cosimulation environment are described as follows.

4.1.1. Cycle-accurate simulation of the programs

The input C programs are compiled using the compiler for the specific processor (e.g., the GNU C compiler *mb-gcc* for MicroBlaze) and translated into binary executable files (e.g., *.ELF* files for MicroBlaze). These binary executable files are then simulated using a cycle-accurate instruction set simulator for the specific processor. Taking the MicroBlaze processor as an example, the executable *.ELF* files are loaded into *mb-gdb*, the GNU C debugger for MicroBlaze. A cycle-accurate instruction set simulator for the MicroBlaze processor is provided by Xilinx. The *mb-gdb* debugger sends instructions of the loaded executable files to the Micro Blaze instruction set simulator and performs cycle-accurate simulation of the execution of the programs. *mb-gdb* also sends/receives commands and data to/from Matlab/Simulink through the Simulink block for the soft processor and interactively simulates the execution of the programs in concurrence with the simulation of the hardware designs within Matlab/Simulink.

4.1.2. Simulation of customized hardware peripherals

The customized hardware peripherals are described using the Matlab/Simulink-based FPGA design tools. For example, *System Generator* supplies a set of dedicated Simulink blocks for describing parallel hardware designs using FPGAs. These Simulink blocks provide arithmetic-level abstractions of the low-level hardware components. There are blocks that represent the basic hardware resources (e.g., flip-flop-based registers, multiplexers), control logic, mathematical functions, memory, and proprietary (intellectual property IP) cores (e.g., the IP cores for fast Fourier transform and finite impulse filters). For example, the *Mult* Simulink block for multiplication provided by *System Generator* captures the arithmetic behavior of multiplication by presenting at its output port the product of the values presented at its two input ports. The low-level design tradeoff of using either embedded or slice-based multipliers is not captured in its arithmetic level abstraction. The application designer assembles the customized hardware peripherals by dragging and dropping the blocks from the block set to his/her designs and connecting them via the Simulink graphic interface. Simulation of the customized hardware peripherals is performed within Matlab/Simulink. Matlab/Simulink maintains a simulation timer to keep track of the simulation process. Each unit of simulation time counted by the simulation timer equals one clock cycle experienced by the corresponding low-level implementations. Finally, once the design process in Matlab/Simulink completes, the low-level implementations of the customized hardware peripherals are automatically generated by the Matlab/Simulink-based design tools.

4.1.3. Data exchange and synchronization among the simulators

The soft processor Simulink block is responsible for exchanging simulation data between the software and hardware simulators during the cosimulation process. Matlab/Simulink provides *Gateway In* and *Gateway Out* Simulink blocks for separating the simulation of the hardware designs described by *System Generator* from the simulation of other Simulink blocks (including the MicroBlaze Simulink blocks). These *Gateway In* and *Gateway Out* blocks identify the input/output communication interfaces of the customized hardware peripherals. For the MicroBlaze processor, the Simulink MicroBlaze block sends the values of the processor registers stored in the MicroBlaze instruction set simulator to the *Gateway In* blocks as input data to the hardware peripherals. Vice versa, the Simulink MicroBlaze block collects the simulation output of the hardware peripherals from *Gateway Out* blocks and use the output data to update the values of the processor registers stored in the MicroBlaze instruction set simulator. The Simulink block for the soft processor also simulates the communication interfaces between the soft processor and the customized hardware peripherals described in Matlab/Simulink. For example, the Simulink MicroBlaze block simulates the communication protocol and the FIFO buffers for communication through Xilinx dedicated (fast simplex link FSL) interfaces [4].

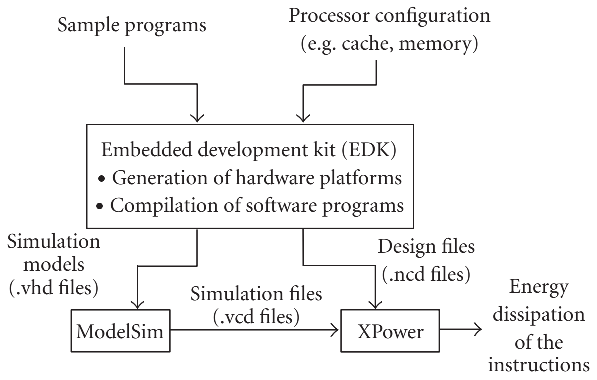

Sample programs
Processor configuration (e.g. cache, memory)
Embedded development kit (EDK)
• Generation of hardware platforms
• Compilation of software programs
Simulation models (.vhd files)
Design files (.ncd files)
Simulation files (.vcd files)
ModelSim
XPower
Energy dissipation of the instructions

Figure 5: Flow of generating the instruction energy lookup table.

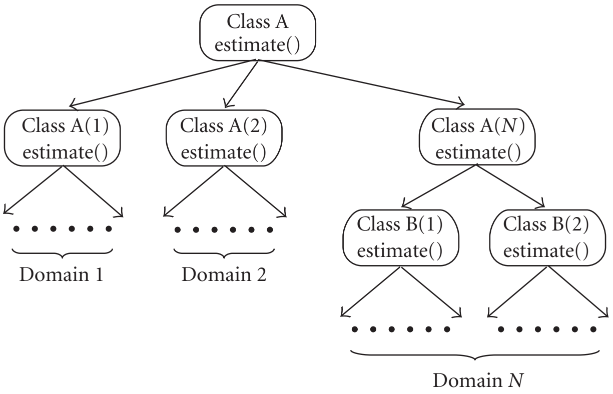


Figure 6: Python classes organized as domains.

The Simulink soft processor block maintains a global simulation timer which keeps track of the simulation time experienced by the hardware and software simulators. When exchanging the simulation data between the simulators, the Simulink soft processor block takes the number of clock cycles required by the processor and the customized hardware peripherals into account. This process considers both the input data and the delays caused by transmitting the data between them. Then, the Simulink block increases the global simulation timer accordingly. By doing so, the hardware and software simulations are synchronized on a cycle-accurate basis.

4.2. Step 2: rapid energy estimation

The energy dissipation of the complete system is obtained by summing up energy dissipation of the software and the hardware. These values are estimated separately by utilizing the activity information gathered during the high-level cosimulation process.

4.2.1. Instruction-level energy estimation for software execution

We use the MicroBlaze processor to illustrate the creation of the instruction energy lookup table. The overall flow for generating the lookup table is illustrated in Figure 5. We developed sample programs that target each instruction in the MicroBlaze processor instruction set by embedding assembly code into the sample C programs. In the embedded assembly code, we repeatedly execute the instruction of interest for a certain amount of time with more than 100 different sets of input data and under various execution contexts. ModelSim was used to perform low-level simulation for executing the sample programs. The gate-level switching activities of the device during the execution of the sample programs are recorded by ModelSim as simulation record files (.vcd files). Finally, a low-level energy estimator such as XPower was used to analyze these simulation record files and estimate energy dissipation of the instructions of interest. See [18] for more details on the construction of instruction-level energy estimators for FPGA configured soft processors.

4.2.2. Domain-specific modeling-based energy estimation for hardware execution

The energy dissipation of the customized hardware peripherals is estimated using the domain-specific energy modeling technique discussed in Section 3.2.2. In order to support this modeling technique, the application designer must be able to group different designs of the kernels into domains and associate the performance models identified through domain-specific modeling with the domains. Since the organization of the Matlab/Simulink block set is inflexible and is difficult to reorganize and extend, we map the blocks in the Simulink block set into classes in the object-oriented Python scripting language [21] by following some naming rules. For example, block *xbsBasic_r3/Mux*, which represents hardware multiplexers, is mapped to a Python class *CxlMul*. All the design parameters of this block, such as *inputs* (number of inputs) and *precision* (precision), are mapped to the data attributes of its corresponding class and are accessible as *CxlMul.inputs* and *CxlMul.precision*. Information on the input and output ports of the blocks is stored in data attributes *ips* and *ops*. By doing so, hardware implementations are described using Python language and are automatically translated into corresponding designs in Matlab/Simulink. For example, for two Python objects A and B, A.ips [0 : 2] = B.ops [2 : 4] has the same effect as connecting the third and fourth output ports of the Simulink block represented by B to the first two input ports of the Simulink block represented by A.

After mapping the block set to the flexible class library in Python, reorganization of the class hierarchy according to the architectures and algorithms represented by the classes becomes possible. Considering the example shown in Figure 6, Python class A represents various implementations of a kernel. It contains a number of subclasses $A(1), A(2), \dots, A(N)$. Each of the subclasses represents one implementation of the kernel that belongs to the same *domain*. Energy performance models identified through domain-specific modeling (i.e., energy functions shown in Figure 7) are associated with these classes. Input to these energy functions is determined by the attributes of Python classes when they are instantiated. When invoked, the *estimate*() method associated with the Python

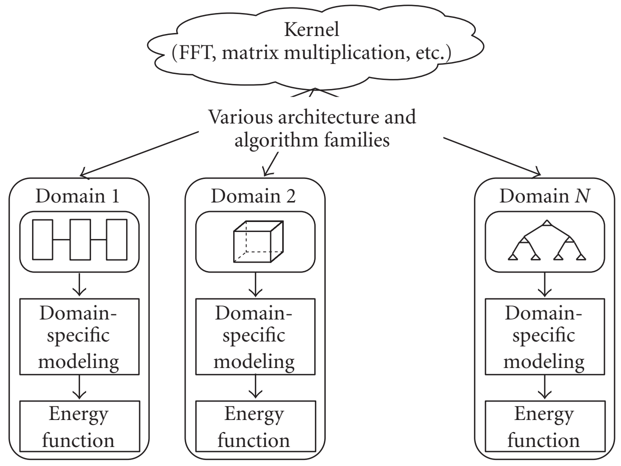


Figure 7: Domain-specific modeling.

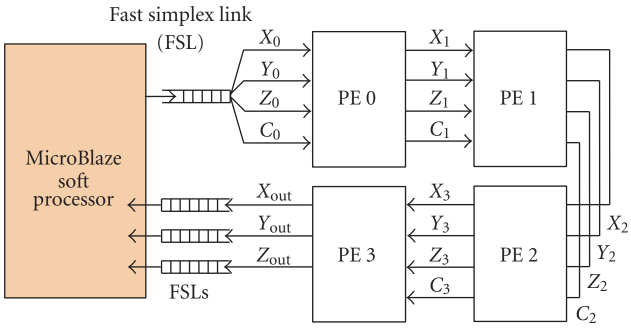


Figure 8: CORDIC processor for division ($P = 4$).

classes returns the energy dissipation of the Simulink blocks calculated using the energy functions.

As a key factor that affects energy dissipation, switching activity information is required before these energy functions can accurately estimate energy dissipation of a design. The switching activity of the low-level implementations is estimated using the information obtained from the high-level cosimulation described in Section 4.1. For example, the switching activity of the Simulink block for addition is estimated as the average switching activity of the two input data and the output data. The switching activity of the processing elements (PEs) of the (coordinate rotation digital computer CORDIC) design [22] shown in Figure 8 is calculated as the average switching activity of all the wires that connect the Simulink blocks contained by the PEs. As shown in Figure 9, high-level switching activities of the processing elements (PEs) shown in Figure 8 obtained within Matlab/Simulink coincide with their power consumption obtained through low-level simulation. Therefore, using such high-level switching activity estimates can greatly improve the accuracy of our energy estimates. Note that for some Simulink blocks, their high-level switching activities may not coincide with their power consumption under some circumstances. For example, Figure 10 illustrates the power consumption of slice-based multipliers for input data sets with different switching activities. These multipliers demonstrate "ceiling effects" when switching activities of the input data are larger than 0.23. Such "ceiling effects" are captured when deriving energy functions for these Simulink blocks in order to ensure the accuracy of our rapid energy estimates.

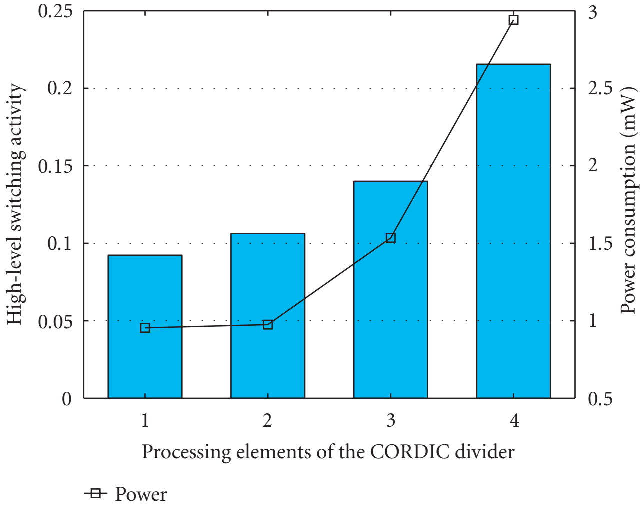


Figure 9: High-level switching activities and power consumption of the PEs shown in Figure 8.

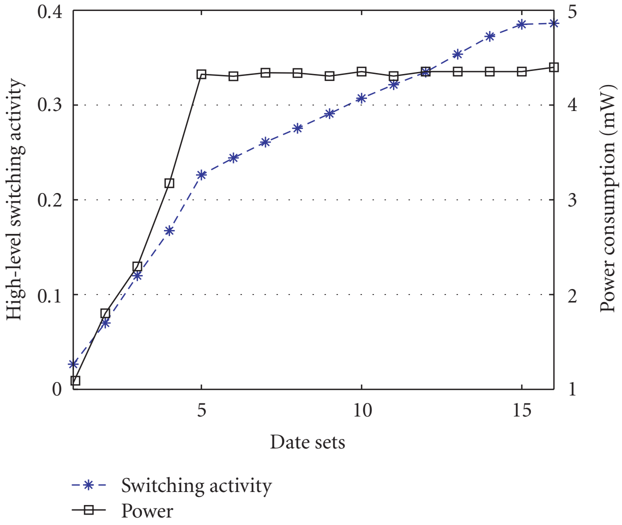


Figure 10: High-level switching activities and power consumption of slice-based multipliers.

5. ILLUSTRATIVE EXAMPLES

To demonstrate the effectiveness of our approach, we evaluate the design of a CORDIC processor for division and a block matrix multiplication algorithm. These designs are widely used in systems such as software-defined radio, where energy is an important performance metric [6]. We focus on MicroBlaze and *System Generator* in our illustrative examples

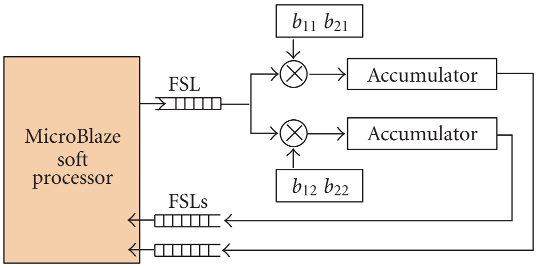


Figure 11: Matrix multiplication with customized hardware for multiplying 2 × 2 matrix blocks.

due to their easy availability. Our approach is also applicable to other soft processors and other design tools.

(i) CORDIC processor for division

The architecture of the CORDIC processor is shown in Figure 8. The customized hardware peripheral is implemented as a linear pipeline of P processing elements (PEs). Each of the PEs performs one CORDIC iteration. The software program controls the data flowing through the PEs and ensures that the data are processed repeatedly until the required number of iterations is completed. Communication between the processor and the hardware implementation is through the FSL interfaces. It is simulated using our MicroBlaze Simulink block. Our implementation uses 32-bit data precision.

(ii) Block matrix multiplication

Smaller matrix blocks of matrices A and B are multiplied using a customized hardware peripheral. As shown in Figure 11, data elements of a matrix block from matrix B (e.g., b_{11}, b_{21}, b_{12} and b_{22}) are fed into the hardware peripheral, followed by data elements of a matrix block from matrix A. The software program running on MicroBlaze controls the data to be sent to and retrieved from the attached customized hardware peripheral, performs part of the computation (e.g., accumulating the multiplication results from the hardware peripheral), and generates the result matrix.

In our experiments, the MicroBlaze processor is configured on a Xilinx Spartan-3 xc3s400 FPGA [4]. The processor, the two (local memory bus LMB) interface controllers and the customized hardware peripherals operate at 50 MHz. (embedded development kit EDK) 6.3.02 [4] is used to describe the software execution platform and for compiling software programs. *System Generator* 6.3 is used to describe the customized hardware peripherals. ISE (integrated software environment) 6.3.02 [4] is used for synthesizing and implementing (placing and routing) the complete applications.

Power measurement is performed using a Spartan-3 FPGA board from Nu Horizons [23] and a SourceMeter 2400 instrument (a programmable power source with the measurement functions of a digital multimeter) from Keithley [24]. Except for the Spartan-3 FPGA device, all the other components on the prototyping board (e.g., the power supply indicator, the SRAM chip) are kept in the same state during measurement. We assume that the changes in power consumption of the board are mainly caused by the FPGA device. We fix the input voltage and measure the changes in input current to the FPGA board. The dynamic power consumption of the designs is calculated based on the changes in input current. Note that *static power* (power consumption of the device when there is no switching activity) is ignored in our experimental results, since it is fixed in the experiments.

The simulation time and energy estimation for implementations of the two numerical computation applications are shown in Table 1. Our high-level cosimulation environment achieves simulation speedups between 5.6x and 88.5x compared with low-level timing simulation using ModelSim. The low-level timing simulation is required for low-level energy estimation using XPower. The speed of the cycle-accurate high-level cosimulation is the major factor that determines the estimation time and varies depending on the hardware-software mapping and scheduling of the tasks that constitute the application. This is due to two main reasons. One reason is the difference in simulation speeds of the hardware simulator and the software simulator. Table 2 shows the simulation speeds of the cycle-accurate MicroBlaze instruction set simulator, the Matlab/Simulink simulation environment for simulating the customized hardware peripherals, and ModelSim for timing-based low-level simulation. Cycle-accurate simulation of software executions is more than 4 times faster than cycle-accurate arithmetic level simulation of hardware execution using Matlab/Simulink. If more tasks are mapped to execute on the customized hardware peripherals, the overall simulation speed of the proposed high-level cosimulation approach is further slowed down. Compared with low-level simulation using ModelSim, our Matlab/Simulink-based implementation of the cosimulation approach can potentially achieve simulation speedups from 29x to more than 114x for the chosen applications. A reason for the variance is the frequency of data exchanges between the software program and the hardware peripherals. Every time the simulation data is exchanged between the hardware simulator and the software simulator, the simulation performed within the simulators is stalled and later resumed. This adds quite some extra overhead to the cosimulation process. There are close interactions between the hardware and software execution for the two numerical computation applications considered in the paper. Thus, the speedups achieved for the two applications are smaller than the maximum speedups that can be achieved in principal.

If we consider the implementation time (including synthesizing, placing-and-routing), the complete system, and generating the post place-and-route simulation models (required by the low-level energy estimation approaches) our high-level cosimulation approach would lead to even greater simulation speedups. For the two numerical applications, the time required to implement the complete system and generate the post place-and-route simulation models is about 3

Table 1: High-level/low-level simulation time and measured/estimated energy performance of the CORDIC-based division application and the block matrix multiplication application.

Designs	Simulation time		Energy estimation		
	High-level	Low-level*	High-level	Low-level	Measured
CORDIC with $N = 24$, $P = 2$	6.3 sec	35.5 sec	1.15 μJ (9.7%)	1.19 μJ (6.8%)	1.28 μJ
CORDIC with $N = 24$, $P = 4$	3.1 sec	34.0 sec	0.69 μJ (9.5%)	0.71 μJ (6.8%)	0.76 μJ
CORDIC with $N = 24$, $P = 6$	2.2 sec	33.5 sec	0.55 μJ (10.1%)	0.57 μJ (7.0%)	0.61 μJ
CORDIC with $N = 24$, $P = 8$	1.7 sec	33.0 sec	0.48 μJ (9.8%)	0.50 μJ (6.5%)	0.53 μJ
12×12 matrix mult. (2×2 blocks)	99.4 sec	8803 sec	595.9 μJ (18.2%)	675.3 μJ (7.3%)	728.5 μJ
12×12 matrix mult. (4×4 blocks)	51.0 sec	3603 sec	327.5 μJ (12.2%)	349.5 μJ (6.3%)	373.0 μJ

Note: * timing-based post place-and-route simulation. The times for placing-and-routing and generating simulation models are not included.

Table 2: Simulation speeds of the hardware-software simulators considered in this paper.

	Instruction set simulator	Simulink$^{(1)}$	ModelSim$^{(2)}$
Simulated clock cycles per second	>10000	254.0	8.7

Note: (1) only considers simulation of the customized hardware peripherals; (2) timing-based post place-and-route simulation. The time for generating the simulation models of the low-level implementations is not accounted for.

hours. Thus, our high-level simulation-based energy estimation technique can be about 200x to 6500x faster than those based on low-level simulation for these two numerical computation applications.

For the hardware peripheral used in the CORDIC division application, our energy estimation is based on the energy functions of the processing elements shown in Figure 8. For the hardware peripheral used in the matrix multiplication application, energy estimation is based on the energy functions of the multipliers and the accumulators. As one input to these energy functions, we calculate the average switching activity of all the input/output ports of the Simulink blocks during arithmetic level simulation. Table 1 shows the energy estimates obtained using our high-level simulation-based energy estimation technique. Energy estimation errors ranging from 9.5% to 18.2% and 11.6% on average are achieved for these two numerical computation applications compared with measured results. Low-level simulation-based energy estimation using XPower achieves an average estimation error of 6.8% compared with measured results.

6. CONCLUSIONS

A two-step rapid energy estimation technique for hardware-software codesign using FPGAs was proposed in this paper. An implementation of the proposed energy estimation technique based on Matlab/Simulink and the design of two numerical computation applications were provided to demonstrate its effectiveness. One major approximation that affects the energy estimation accuracy of the proposed technique is a failure to consider glitches in high-level simulation. This limitation creates two scenarios that causes our technique to fail to give energy estimates with satisfactory errors. One scenario occurs when an application runs close to its maximum operating frequency. The other scenario occurs when an application has long combinational circuit paths. In both scenarios, numerous glitches can occur in the circuits, causing high energy estimation errors for the proposed technique. The integration of high-level glitch power estimation techniques is an important extension of the proposed technique. Another important extension of our work is to provide confidence level information of the energy estimates. Providing such information is desired in the development of many practical systems.

ACKNOWLEDGMENTS

This work is supported by the United States National Science Foundation (NSF) under Award No. CCR-0311823. The authors would like to thank Brent Milne, Haibing Ma, Shay P. Seng, and Jim Hwang from Xilinx, Inc. for their help and discussions on creating the Matlab/Simulink-based high-level cosimulation environment.

REFERENCES

[1] Altera Inc., http://www.altera.com.

[2] Gaisler Research Inc., "LEON3 User Manual," http://www.gaisler.com.

[3] Actel Inc., http://www.actel.com.

[4] Xilinx Inc., http://www.xilinx.com.

[5] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '03)*, pp. 57–60, San Jose, Calif, USA, September 2003.

[6] C. Dick, "The platform FPGA: enabling the software radio," in *Proceedings of the Software Defined Radio Technical Conference and Product Exposition (SDR '02)*, San Diego, Calif, USA, November 2002.

[7] A. Bakshi, V. K. Prasanna, and Á. Lédeczi, "MILAN: a model based integrated simulation framework for design of embedded systems," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pp. 82–93, Snowbird, Utah, USA, June 2001.

[8] MathWorks Inc., http://www.mathworks.com.

[9] "The Ptolemy Project," http://ptolemy.eecs.berkeley.edu.

[10] Mentor Graphics Inc., http://www.mentor.com.

[11] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279–302, 2005.

[12] "Reconfigurable Hardware in Orbit (RHinO)," Information Sciences Institute, http://rhino.east.isi.edu.

[13] "Web Power Analysis Tools," Xilinx, http://www.xilinx.com/power.

[14] J. Ou and V. K. Prasanna, "PyGen: a MATLAB/Simulink based tool for synthesizing parameterized and energy efficient designs using FPGAs," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 47–56, Napa, Calif, USA, April 2004.

[15] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*, pp. 83–94, Vancouver, BC, Canada, June 2000.

[16] A. Sinha and A. Chandrakasan, "JouleTrack: a web based tool for software energy profiling," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 220–225, Las Vegas, Nev, USA, June 2001.

[17] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Proceedings of the 37th Design Automation Conference (DAC '00)*, pp. 340–345, Los Angeles, Calif, USA, June 2000.

[18] J. Ou and V. K. Prasanna, "Rapid energy estimation of computations on FPGA based soft processors," in *Proceedings of the IEEE International System-on-Chip Conference (SoCC '04)*, pp. 285–288, Santa Clara, Calif, USA, September 2004.

[19] T. Kogel, A. Haverinen, and J. Aldis, "OCP TLM for Architectural Modeling (white paper)," OCP-IP, 2005, http://www.ocpip.org.

[20] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-specific modeling for rapid energy estimation of reconfigurable architectures," *Journal of Supercomputing*, vol. 26, no. 3, pp. 259–281, 2003.

[21] Python, http://www.python.org.

[22] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 191–200, Monterey, Calif, USA, February 1998.

[23] Nu Horizons Electronics Inc., http://www.nuhorizons.com.

[24] Keithley Instruments Inc., http://www.keithley.com.

Jingzhao Ou received his B.S. and M.S. degrees in electrical engineering from the South China University of Technology, and his M.S. and Ph.D. degrees in computer engineering from the University of Southern California. He is now working for the DSP Design Tools and Methodologies Group at Xilinx, Inc. His main research interests include hardware-software codesign and energy efficient application development using reconfigurable hardware.

Viktor K. Prasanna received his B.S. degree in electronics engineering from the Bangalore University, his M.S. degree from the School of Automation, Indian Institute of Science, and his Ph.D. degree in computer science from the Pennsylvania State University. He is now a Professor of electrical engineering and Professor of computer science at the University of Southern California (USC). He is also a Member of the NSF Supported Integrated Media Systems Center (IMSC), an Associate Member of the Center for Applied Mathematical Sciences (CAMS), and a Member of USC-ChevronTexaco Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft) at USC. His research interests include high-performance computing, parallel and distributed systems, network computing, and embedded systems.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 31605, Pages 1–10
DOI 10.1155/ES/2006/31605

FPGA Dynamic Power Minimization through Placement and Routing Constraints

Li Wang, Matthew French, Azadeh Davoodi, and Deepak Agarwal

Information Sciences Institute, University of Southern California, Arlington, VA 22203, USA

Received 15 December 2005; Accepted 18 April 2006

Field-programmable gate arrays (FPGAs) are pervasive in embedded systems requiring low-power utilization. A novel power optimization methodology for reducing the dynamic power consumed by the routing of FPGA circuits by modifying the constraints applied to existing commercial tool sets is presented. The power optimization techniques influence commercial FPGA Place and Route (PAR) tools by translating power goals into standard throughput and placement-based constraints. The Low-Power Intelligent Tool Environment (LITE) is presented, which was developed to support the experimentation of power models and power optimization algorithms. The generated constraints seek to implement one of four power optimization approaches: slack minimization, clock tree paring, N-terminal net colocation, and area minimization. In an experimental study, we optimize dynamic power of circuits mapped into 0.12 μm Xilinx Virtex-II FPGAs. Results show that several optimization algorithms can be combined on a single design, and power is reduced by up to 19.4%, with an average power savings of 10.2%.

1. INTRODUCTION

Field-programmable gate arrays (FPGAs) now handle most digital signal processing functions in an embedded platform. However, many embedded platforms, such as handheld devices, distributed sensors, and satellites, demand low power in order to increase their functional lifetime. While SRAM-based FPGAs have a short design cycle, steadily decreasing cost, and growing performance, power consumption remains a concern [1]. The trend from one FPGA device family to another is the number of configurable logic blocks (CLBs) and maximum operating frequency scale exponentially, while corresponding decreases in operating voltage have been much slower to arrive, resulting in an exponentially increasing maximum power consumption per device [2]. Therefore, power must be considered at every level, from VLSI issues such as transistor layout and leakage current, to the software that determines how efficiently a user's design is implemented on an FPGA.

There have been many FPGA power reduction approaches addressing different design levels. Several techniques for low power FPGA design have appeared in literature addressing the VLSI design of an FPGA [2–4]. Research has also considered various synthesis-level power optimizations, such as technology mapping to LUT-based FPGAs techniques [5] or reducing glitching power through pipelining [6]. It has also been shown that power can be addressed in the suite of computer-aided design (CAD) algorithms that place and route an end user's circuit onto the FPGA fabric [7].

For our research, we are considering techniques that yield immediate results on today's devices and interoperate with commercial off-the-shelf (COTS) CAD tools. We further restrict our focus to techniques that do not modify the functional behavior of the circuit and guarantee that the user's original timing, or throughput, constraints are met. In this paper, we propose a novel power optimization methodology that converts power optimization goals into constraints compliant with throughput-based COTS PAR tools, minimizing the power consumption of a design's routing interconnect.

In today's FPGAs about 50–70% of total power is dissipated in the interconnection network [8]. The dynamic power of nets is characterized by

$$P_{\text{dynamic}} = \sum_{i} \left(C_i \times F_i \times V^2\right), \qquad (1)$$

where C_i and F_i are the capacitance and average toggle rate of the ith net, and V is the internal voltage. For a given net, the dynamic power can be reduced by diminishing its capacitance, or length. Nets with high toggle rates and/or high capacitance therefore are good potential targets for decreasing the overall power and serve as the motivation of the power optimization schemes presented.

In this work, we first introduce the Low-Power Intelligent Tool Environment (LITE) created for this research. This environment allows the development and experimentation of power models, tracking dynamic power consumption during simulation, and power estimation at the synthesis level, while providing an infrastructure to rapidly design and execute new power optimization algorithms. Using LITE, four power optimization approaches were created and implemented that generate constraints compliant with the COTS Xilinx PAR tools.

The rest of the paper is organized as follows. In Section 2, we introduce the relevant background on the Xilinx Virtex-II FPGA microarchitecture as it pertains to routing interconnects and power consumption. Section 3 addresses the software, first describing the Xilinx CAD tool flow and then the infrastructure of the Low-Power Intelligent Tool Environment (LITE). Section 4 introduces the power optimization algorithms and their experimental results. In Section 5, the results of combining the power optimization methods are presented. In Section 6, we extend our software results to a hardware testbed and validate our approach. Finally, Section 7 concludes the paper.

2. FPGA DEVICE POWER CHARACTERISTICS

In order to create efficient power optimization algorithms, the underlying FPGA architecture must be well understood. Though the techniques presented here work for a variety of FPGA microarchitectures, we will limit our focus in this paper to the Xilinx Virtex-II FPGA. The Virtex-II FPGA devices are comprised of input/output blocks (IOBs), located on the edges of FPGA chips, and configurable logic blocks (CLBs) organized as a two-dimensional array inside the ring of IOBs [9]. Each CLB includes four slices and an interconnect block. Slices provide functional elements for combinational and synchronous logic which can be configured as ROMs, LUTs, or SRLs, flip-flops, or other circuitry. The logic of a user's circuit will be considered static after synthesis and capacitance information of each microarchitecture feature can be found in literature [8] or in software by exporting information from Xilinx XPower power analysis tool.

In Virtex-II FPGAs, CLBs connect to the global routing matrix through the interconnect fabric. Global routing resources are comprised of 4 types of lines: long lines, hex lines, double lines, and direct connect lines, in the order of their length. Interconnect capacitance can also be found by exporting results from the Xilinx XPower tool. It is important to note that a net in a user's circuit may have any combination of routing, from carry-chains and internal CLB routing with minimal capacitance, to several vertical and horizontal hops along longer interconnect routes. A quick glance at the interconnect capacitance in Table 1 shows that a reduction by only one interconnect length can yield about a 30% reduction in capacitance.

The clocking infrastructure is also critical to consider when optimizing power. With 100% toggle rates and extremely high fanouts, these nets typically consume the most power in a design, even with dedicated clocking lines. The Virtex-II architecture supports 16 clocks, and 8 global clocks can be used in each quadrant of the device. In each quadrant, clocks are organized in clock regions. Figure 1 depicts the clock tree and clock regions in the XC2V6000 FPGA device.

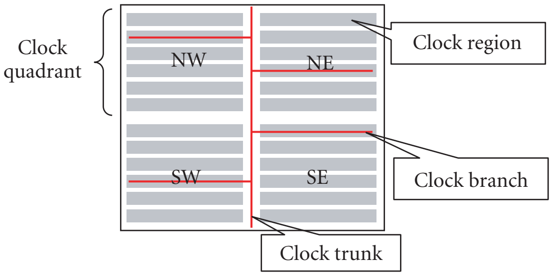


Figure 1: Clock tree and clock regions in XC2V6000 FPGA.

Table 1: Interconnect capacitance.

Interconnect line	Capacitance (pF)
Direct line	9.4
Double line	13.2
Hex line	18.4
Long line	26.1

Although we are focusing on the Virtex-II architecture, the algorithms presented here can be adapted to other architectures as well, as long as cost tables such as those in Table 1 are adjusted to account for minor architecture differences.

3. SOFTWARE INFRASTRUCTURE

This section discusses the software infrastructure developed to rapidly analyze FPGA power consumption and implement power optimization algorithms. As the developed tools interoperate with the COTS CAD tool flow, the Xilinx PAR tools will be discussed first with respect to power and the Low-Power Intelligent Tool Environment (LITE) is described afterwards. Finally, the experiment framework and validation methodology are presented.

3.1. *Xilinx tool flows*

The Xilinx tool flow of design implementation includes the following steps [10].

(i) Translate, which merges the incoming netlists and constraints into a Xilinx design file.

(ii) Map, which fits the design into the available resources on the target device.

(iii) Place and Route, which places and routes the design to the timing constraints.

After Place and Route, the resulting netlist can be input into the Xilinx XPower tool to create a detailed power consumption report. HDL models can be created after PAR for back-annotated simulation to increase the precision of

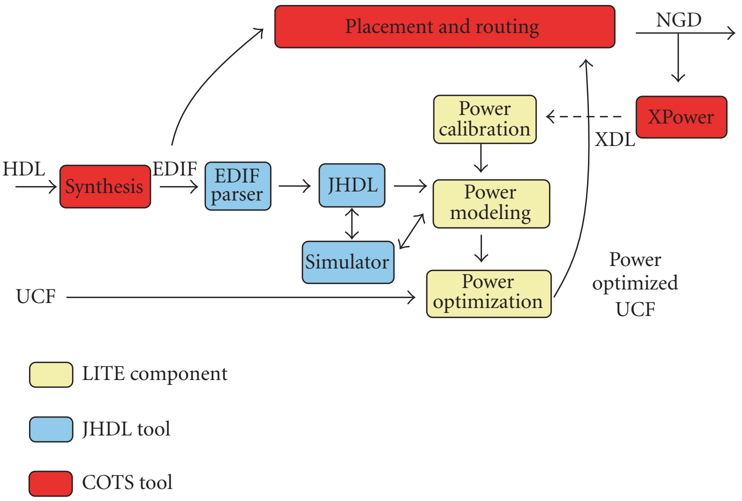


Figure 2: LITE tool flow.

XPower reports. All experiments were run using the Xilinx ISE 6.3 toolset.

3.2. LITE tool flow

The Low-Power Intelligent Tool Environment (LITE) was created to facilitate power research by elevating power to a first-order design parameter. It uses calibration, modeling, and estimation techniques to provide automated power estimation at the higher, logic-based EDIF level, where it is easier for a circuit designer to relate the analysis back to their HDL input. In this work, LITE is expanded to incorporate power optimization algorithms that generate UCF file constraints to be passed along to the Xilinx PAR tools as shown in Figure 2.

LITE consists of three components designed to expand the existing COTS power analysis capabilities and experiment with power optimization algorithms: power calibration, power modeling, and power constraint generation. The LITE tool infrastructure is an extension of the JHDL environment. As presented in [11], the JHDL environment provides a high-level tool suite for querying circuit components, running simulations, and tracking signal transitions. LITE builds upon these capabilities to add knowledge about circuit component and interconnect capacitance, monitor a circuit's power consumption during simulation, sort the most power intensive modules within a circuit, and plot various power consumption metrics of the design. A separate EDIF import tool was developed that enables FPGA designs generated by any 3rd party synthesis tool to be imported into LITE. Simulation results can be obtained by either importing a VCD file or writing a JHDL test bench.

The power calibration component interacts with the Xilinx CAD tools to extract the relevant parameters for power modeling: capacitance, toggle rates, fanout, and power. Xilinx XPower reports contain detailed analysis of placed and routed circuits' power characteristics, and this information can be imported to LITE to obtain the capacitance values of every microarchitectural component, logic element, and interconnect. LITE can then use this information to track and display dynamic power consumption during simulation, or use these values as device power libraries for post-synthesis power modeling and estimation.

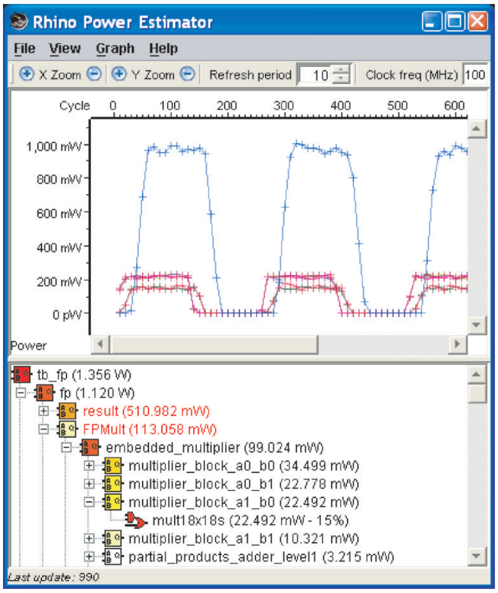
The power modeling component allows detailed power analysis of a user's circuit both at the post-synthesis level and the placed and routed level. Post-synthesis power modeling is achieved by combining known logic component capacitance values with routing interconnect length projection techniques developed in [11]. Exact routing capacitances cannot be known until PAR has been completed, however these estimation models are extremely useful in pinpointing power consumption hot spots early on in the design flow and prioritizing nets for power optimization during the PAR process.

By leveraging the JHDL/EDIF infrastructure, this tool suite also enables users to import their designs into the LITE environment, run simulations, track signal transition rates and power consumption over time, as in Figure 3, sort hierarchy modules by power consumption, and cross-probe power overlays with the schematic and waveform viewers inherent to JHDL. Simulations and power analysis can be performed at either the post-synthesis or placed and routed netlist level and allows the direct comparison of the synthesized circuit power against it's placed and routed netlist power.

The power optimization component utilizes the output of the power analysis component to apply the power optimization techniques discussed in Section 4. As mentioned earlier, the power optimization techniques in LITE do not modify design logic, but rather feed additional constraints to the PAR tools such that the existing PAR algorithms can still meet a user's throughput specifications while also reducing power. To support this, the power optimization component is capable of inspecting the area, resources, and size of the targeted FPGA device and the user's circuit, reads in any existing UCF file constraints, and prioritizes the original constraints.

Table 2: Benchmark circuits.

Design	Part number	Original timing (MHz)	Signal power (%)	Logic power (%)	Clock power (%)	Baseline power (mW)
CRC	XC2V80	16	28	42	30	31
FM	XC2V250	55	43	45	12	102
VGA	XC2V250	125 133.3	18	39	43	138
USBF	XC2V500	238 105	33	30	37	82
PCI	XC2V1000	100	10	33	57	39
Conv	XC2V1000	66	23	55	22	163
DES3	XC2V2000	100	43	21	36	139
Mem	XC2V6000	83	8	59	33	643
S1	XC2V6000	160 40 180 75 33	12	10	78	251
S2	XC2V6000	33 250 100	9	12	79	1020



tb_fp (1.356 W)
fp (1.120 W)
result (510.982 mW)
FPMult (113.058 mW)
embedded_multiplier (99.024 mW)
multiplier_block_a0_b0 (34.499 mW)
multiplier_block_a0_b1 (22.778 mW)
multiplier_block_a1_b0 (22.492 mW)
mult18x18s (22.492 mW - 15%)
multiplier_block_a1_b1 (10.321 mW)
partial_products_adder_level1 (3.215 mW)
Last update: 990

Figure 3: LITE simulation.

3.3. Experimental framework

The methodology for power optimization and power verification can also be seen in Figure 2. To perform power optimization, a user imports its design using the EDIF parser, generates a power simulation using the LITE power modeling component, and then generates a new UCF file using the LITE power optimization component. The original, unaltered EDIF file can then be fed through the Xilinx tools using the new constraints file. To measure the results, we use the Xilinx XPower tool with placed and routed netlists and the same value change dump (VCD) simulation data used as inputs in the LITE power simulation stage.

In order to verify the developed power optimization algorithms, a test suite of ten circuit benchmarks was utilized, listed in Table 2. This suite represents a fairly wide taxonomy of applications, from glue logic (Mem) to cores (CRC, FM, VGA, USBF, PCI, and DES3) to end-to-end applications (Conv, S1, and S2), spanning a wide range of device sizes. Each circuit is mapped into the smallest device possible, such that underutilization does not skew results. All designs also had UCF files specifying I/O pin locations and minimum clocking requirements, shown in the 3rd column. Multiple clocks are represented by multiple entries. Table 2 also shows the breakout of power consumed by signal, logic, and clock elements and reveals that there is a mix of clock dominant, signal dominant, and logic dominant designs. In the final column, the baseline power, the internal dynamic power of each circuit as reported by XPower is shown, that is, the sum of the dynamic power consumed by logic elements, clock nets, and signal nets. Figure 4 shows the slice/IOB utilizations of these designs. Slice occupation ranges from 14% to 86%, and IOB occupation from 11% to 90%, so there is a fair representation of I/O bound as well as compute resource bound circuits.

It should be noted that we have spot checked our results on hardware as well. Our power measurement testbed, shown in Figure 5, is comprised of a PCI-DAS1200 ADC which samples the current sensors connected to the isolated internal voltage supply lines on an Osiris board's XC2V6000 device and provides a resolution 2.7 mA. While actual power consumption was difficult to verify due to variables such as room temperature, device fabrication variances, and conservatism inherent in XPower's capacitance reporting, the

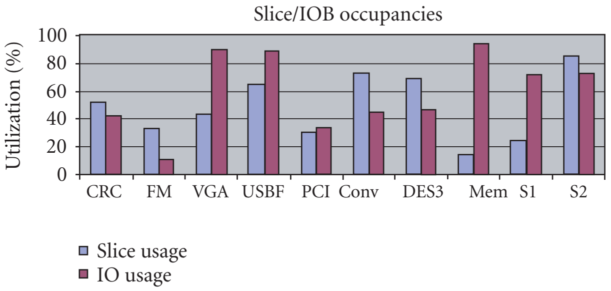


Figure 4: Benchmarks slice/IOB utilization.

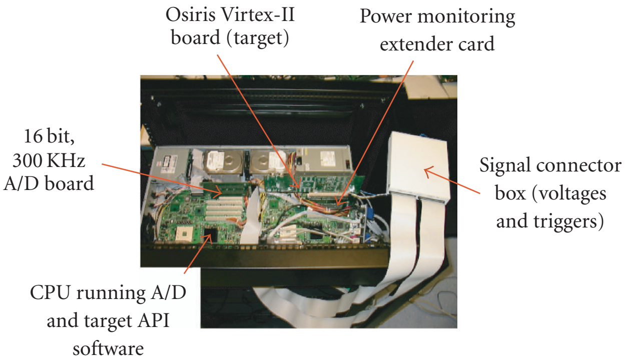

Osiris Virtex-II board (target)
Power monitoring extender card
16 bit, 300 KHz A/D board
Signal connector box (voltages and triggers)
CPU running A/D and target API software

Figure 5: Power measurement testbed.

percentage power reduction between the optimized and baseline versions remained constant between XPower software reports and hardware measurements in experimental testing.

4. POWER OPTIMIZATION TECHNIQUES

The power optimization techniques developed center around the theme of creating timing and placement constraints that interoperate with existing COTS PAR tools in order to preserve a user's throughput specifications while also reducing power consumption. The timing and placement constraints influence the COTS tools to use shorter, lower capacitance interconnects. In this paper we provide an overview of four power optimization techniques that each utilizes a different constraint type to enact power optimization. The following subsections explain each technique and present the experimental results achieved.

4.1. *Clock tree paring*

For our first technique, we will focus on trying to reduce the amount of power utilized by the clock nets. As Table 2 shows, even though these nets utilize dedicated, specialized circuitry within the FPGA, these few nets can contribute with 12% to 79% of the overall power consumption of a design. This is due to the inherent high toggle rate, high fanout to hundreds or thousands of synchronous logic elements, and long interconnects that span a data path from input to output often across the entire device.

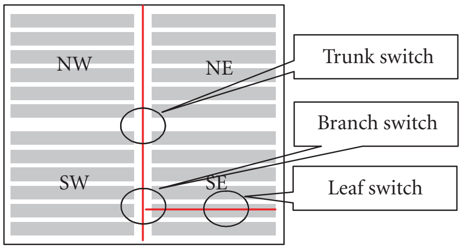


Figure 6: Clock net switch types.

The clock tree paring algorithm targets the clock power by utilizing placement constraints to minimize the size of the clock net tree utilized. As introduced in Section 2, in the Xilinx Virtex-II FPGAs, clock nets are distributed on dedicated routing resources. Through FPGA editor and experimentation, we observe that clock network is like a tree, with the main trunk traveling north to south in the middle of the chip, and branches extending west and east into clock regions. The number of clock regions varies depending on the size of the device. The clock tree is gated such that completely unused branches of the tree are effectively turned off. Therefore by placing logic closer together, clocking power can be reduced by gating more of the branches of the clock tree.

From our analysis, we found that there were three types of gating switches, shown in Figure 6, which we will call the trunk switch, branch switch, and leaf switch. The trunk switch is located at the center of the chip. This type of switch is used for turning on or off the upper- or lower-half of the main clock trunks. When a clock net comes into the chip from an input port or digital clock manager (DCM), it goes to the center of the switch-fabric to be routed to the north, or south, or both. Figure 7(a) shows two clock nets as the examples: the clock net on the left is switched to both the upper- and lower-half of the chip. The clock net on the right is switched to the upper-part of the chip only. Figure 7(b) depicts a branch switch. Each Virtex-II has multiple branch switches, and the number varies depending on the size of the device. The switches are located on the path of the main clock trunks. They are responsible for transmitting the clock signals to the clock regions. The clock wire shown in Figure 7(b) travels to both the left and right. The leaf switch is depicted in Figure 7(c). As shown in Figure 7(d), a clock net in the clock region includes a major branch and many subbranches that connect to slices. The leaf switch turns on/off these subbranches. By placing the flip-flops closer to each other, clocking power can be reduced by leaving more branch/subbranch turned off.

The clock tree paring algorithm analyzes a user's circuit, computes a minimum bound to contain all the logic associated with a clock net, and generates area constraints to specify where the associated clock logic may be placed. The area constraint is rectangular, stretching north to south around the clock main trunk. The size of the area is proportional to a clock's fanout. For multiple clock cases, the LITE power analysis component is used to prioritize clocks with higher-power consumption and place them closer to

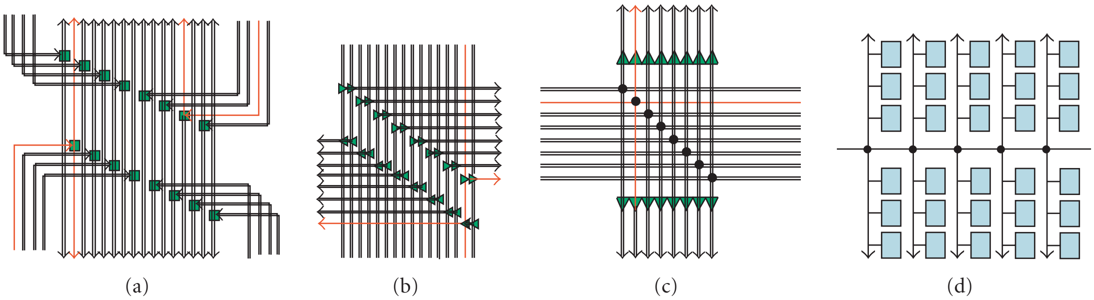


Figure 7: (a) Trunk switch; (b) branch switch; (c) leaf switch; (d) clock net connected with FFs within a clock region.

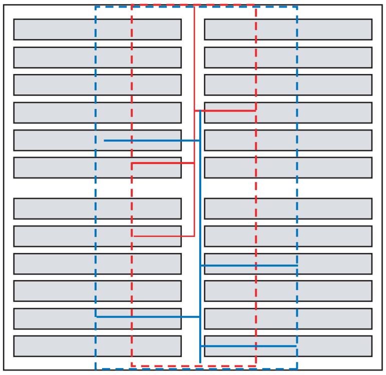


Figure 8: Clock area constraints.

Table 3: Clock tree paring results.

Design	Signal power reduction	Logic power reduction	Clock power reduction	Total power reduction
CRC	3.6%	0.0%	16.4%	5.9%
FM	−3.0%	0.0%	36.0%	2.9%
VGA	6.4%	0.0%	26.5%	12.5%
USBF	2.1%	0.0%	26.8%	10.7%
PCI	−5.1%	0.0%	34.2%	18.7%
Conv	4.0%	0.0%	18.2%	4.9%
DES3	−4.0%	0.0%	29.2%	8.6%
Mem	−11.2%	0.0%	5.1%	0.7%
S1	−59.9%	0.0%	11.1%	10.7%
S2	−13.8%	−0.1%	28.7%	19.4%

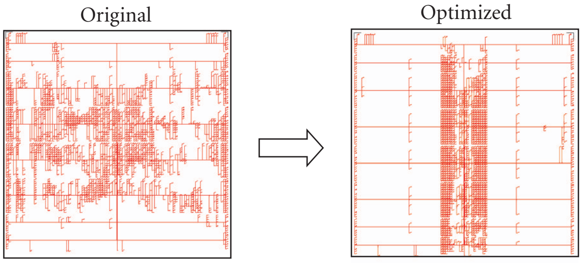


Figure 9: Clock area optimization in S1.

the clock trunk, as depicted in Figure 8. It should be noted that the clock groups do not have to be placed radially to the main trunk to save power. Clock power savings, especially in larger designs, come from clustering groups of flip-flops to minimize the number of leaf switches that are activated. In the cases that I/O timing is critical, flip-flop clusters can be placed between the I/O pins and a central flip-flop mass about the clock trunk, to pipeline and better preserve timing constraints while also minimizing power. Figure 9 shows an illustrative example of the distributions of one of the clock trees in S1 before and after the clock optimization.

Table 3 shows the results for clock tree paring power optimization. It is interesting to note that even though the signal power increases in several cases, the clock power savings are dominate and almost all benchmarks show significant overall power improvement by using this approach. As can be expected, the test circuits not responding as well to this approach (Mem, FM, Conv, and CRC) are considered logic power dominant designs according to Table 2. The clock power dominant designs (S2, PCI, VGA, S1, and USBF) are much more responsive. It should also be noted that though Figure 9 depicts a circuit with low device utilization for illustrative purposes, the efficacy of this technique is more a function of a circuit being clock power dominant than high- or low-logic utilization. For example, S2, a clock power dominant circuit, achieves the most significant power reduction with a more than 80% device utilization, while Mem, the lowest device utilization circuit in our test suite, yields the least significant results.

4.2. *N*-terminal net colocation

N-terminal net colocation power optimization is targeted to reduce the power consumed by signal nets. “Terminal” is defined as the sum of the fanin and fanout of a net. For a simplified case, a 2-terminal net is a net with a single fanout. *N*-terminal net colocation restricts net terminals to be placed in adjacent slices. As depicted in Figure 10, net terminals are grouped in pairs, and for each pair, a constraint is used to restrict the two terminals to be located close to each other, and thus reducing the signal net length and power. From our

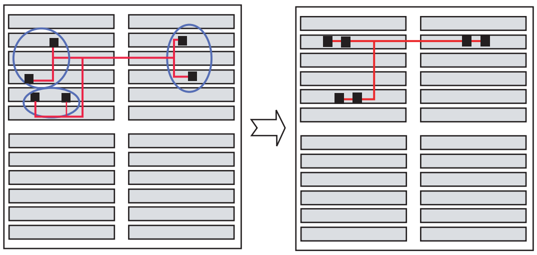


Figure 10: N-terminal placement.

LITE calibration and analysis studies, we found that the Xilinx Virtex-II architecture has an east-west bias, meaning that direct connection interconnected in the east-west direction has less capacitance than direct connections in the north-south direction, sometimes by a factor of up to 50%. So, this algorithm is further enhanced to take advantage of this particular microarchitecture design by prioritizing east-to-west relative placement constraints. This algorithm can be updated to reflect other FPGA architecture features as well. The nets are sorted and prioritized by power consumption based on simulations using the LITE power analysis environment to target high-capacitance and high toggle rate nets. In high fanout cases where nets may belong to multiple terminal groups, only the highest priority constraint is created.

Initial experimentation showed that this technique worked well on some nets, however some nets that would naturally be mapped by the COTS PAR tools to low capacitance lines such as carry chains and internal slice nets were now being routed on higher capacitance routing interconnect lines due to the constraints. To avoid this, the algorithm was enhanced to analyze the circuits and selectively avoid putting constraints on certain nets. Several rules were developed to avoid overconstraining the designs as follows.

(i) Avoid nets that are a part of shift registers as the Xilinx slice contains low capacitance, dedicated connection between shift registers that are naturally used by the PAR tools.

(ii) Avoid nets that are a part of carry-chains. The Virtex-II architecture uses dedicated low capacitance carry logic to cascade function generators and provide fast arithmetic addition and subtraction.

(iii) Avoid nets that are mapped internally to slices as these are also low capacitance routes. These nets can be identified as those between look-up tables (LUTs) and multiplexers, and between LUTs and inverters.

The results for the N-terminal net colocation algorithm are depicted in Table 4. Here, we see that the overall power savings is negligible and in a few cases actually becomes worse. The nonzero values in the logic power reduction column show that in some cases slices are being packed more efficiently as desired, however in some designs the N-terminal approach causes ripple effects in unconstrained nets, causing more slices to be utilized. While the constrained nets are reduced, other nets belonging to multiple terminal groups may be bumped out of internal slice mappings. Comparing the signal power, clock power, and total power columns is insightful as well. For a few circuits, CRC, USBF, and S1, there is a significant reduction in signal power. Closer inspection revealed that these circuits had relatively few high fanout nets. In all cases however, clock power is still dominating and is the main influence on total power.

Table 4: N-terminal placement results.

Design	Signal power reduction	Logic power reduction	Clock power reduction	Total power reduction
CRC	−9.8%	0.0%	0.0%	−2.9%
FM	−0.9%	−0.5%	1.6%	−0.4%
VGA	1.8%	0.2%	0.6%	0.7%
USBF	−9.2%	0.4%	−4.2%	−4.5%
PCI	2.6%	0.1%	−6.8%	−3.8%
Conv	1.6%	0.0%	−3.4%	−0.4%
DES3	1.2%	0.0%	−3.3%	−0.7%
Mem	9.1%	0.0%	−1.8%	0.4%
S1	−10.1%	0.3%	−0.5%	−0.6%
S2	−1.6%	−1.4%	1.6%	1.0%

4.3. *Area minimization*

Another approach to reducing signal power was area minimization. The area minimization power optimization technique is based on the observation that routing interconnect lengths highly depend on the placement of components. By prioritizing the location in favor of power, high capacitance signal lines with high fanout or high transition rates can be grouped together to minimize the power consumed on long interconnects. Constraining the area also has the added affect of trimming the clock tree; however in this case the total area is constrained and clock tree pruning is a residual affect.

This technique is expected to work well on circuits that underutilize the logic available on the chip due to I/O bound designs or poor device size selection. In these designs, the COTS PAR tools place the circuits loosely over the whole chip, doing the minimum to meet the user's timing requirements, as it was designed to do. This behavior however causes longer connection wires and hence increases the total net power. By using area minimization constraints, a design is compacted more tightly in a given area of a chip. Net lengths are shortened and thus power is saved. In an effort to balance the north-south bias of the clock trunk with the east-west bias of the direct connect signal wires, a rectangular area placed at the center of a chip, with sides proportional to the chip dimensions, is utilized. The size of the area is estimated by analyzing and computing the slice count that each design element needs.

Figure 11 shows an example of the results. On the left-hand side, the circuit is placed loosely over the chip. After using the area minimization power optimization, the circuit is tightly located in an area at the center. It is worth mentioning that eventhough area minimization may have the same effect on the placement of logic components as clock power

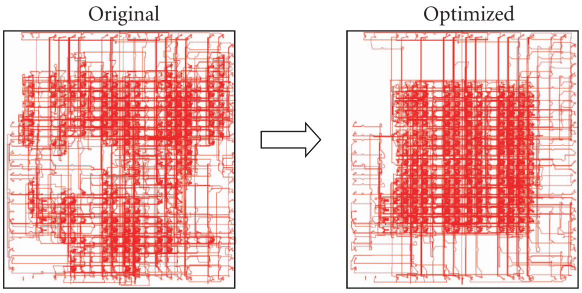


Figure 11: Area minimization in VGA.

optimization does, it utilizes different constraints. The clock tree paring technique constrains the clock routing area, influencing the placement of all the logic elements driven by the clock. The area minimization technique explicitly restricts the placement of all components, clocked or nonclocked.

The results of area minimization approach are shown in Table 5, with all circuits showing a positive power reduction. On closer examination the power savings mostly come due to clock power reductions, due to residual clock tree minimization effects similar to those developed in the clock tree paring technique. This technique was unable to be used on the S2 circuit, as this design occupies 87% of an XC2V6000 device and the area cannot be further minimized.

4.4. Slack minimization

Finally, the slack minimization technique seeks to optimize the power on signal nets by tightening timing constraints on power critical nets. The slack minimization algorithm assumes that the PAR tools will leave each net at or just under the user's specified timing requirements, in many cases leaving slack, or extra net length that could be further tightened to reduce capacitance. For this algorithm slack is defined as

$$\text{Slack} = T_{\text{Spec}} - T_{\text{Logic}} - T_{\min wr}, \qquad (2)$$

where T_{Spec} is the user's timing specification, T_{Logic} is the timing delay of any combinatorial logic in between flip-flops on the net, and $T_{\min wr}$ is the minimal wire timing delay. For example, in the left-hand side of Figure 12, a flip-flop to flip-flop path has two intermediate components, with 1 ns and 2 ns individual delay. The user's specified clock is running at 100 MHz, that is, 10 ns in period. Therefore, the slack of the path is 7 ns. Without additional constraints, the PAR tools will typically meet the maximum delay necessary to still meet the constraints as it should, creating a wire delay of up to 7 ns. If we allow 1 ns delay between each logic element, we can reduce the interconnect length to 3 ns and reduce the interconnect capacitance.

The slack minimization technique uses the LITE analysis component to prioritize high capacitance, high toggle rate nets, calculate the slack, and tighten the timing constraints on these nets allowing for only minimal wire length. In practice, nets with ample slack are typically those with two or less levels of combinational logic between flip-flops.

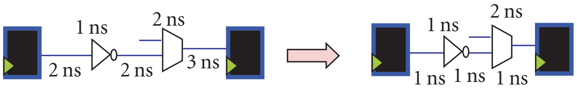


Figure 12: Slack minimization.

Table 5: Area minimization results.

Design	Signal power reduction	Logic power reduction	Clock power reduction	Total power reduction
CRC	−3.1%	0.0%	6.9%	1.2%
FM	−7.7%	0.0%	31.6%	0.4%
VGA	−0.7%	0.0%	1.3%	0.4%
USBF	−1.6%	0.0%	2.0%	0.2%
PCI	2.3%	0.0%	1.1%	0.6%
Conv	3.2%	0.0%	1.9%	1.2%
DES3	−7.7%	0.0%	28.8%	6.9%
Mem	13.3%	0.0%	1.4%	1.6%
S1	−38.0%	0.0%	3.0%	2.8%
S2	NA	NA	NA	NA

Table 6: Slack minimization results.

Design	Signal power reduction	Logic power reduction	Clock power reduction	Total power reduction
CRC	0.9%	0.0%	0.0%	0.3%
FM	NA	NA	NA	NA
VGA	−1.7%	0.0%	−1.0%	−0.7%
USBF	0.0%	0.0%	−2.0%	−0.7%
PCI	2.4%	0.0%	−0.4%	0.0%
Conv	−0.6%	0.0%	1.2%	0.1%
DES3	−0.7%	0.0%	−3.8%	−1.7%
Mem	14.4%	0.0%	2.8%	2.1%
S1	−15.3%	0.0%	−0.8%	−0.9%
S2	−1.6%	0.1%	5.4%	4.1%

The results of using the slack minimization approach on the circuit test suite are shown in Table 6. In the table the three columns in the middle provide the power reduction in signal, logic, and clock dynamic power in percentage. The right-most column presents the overall power savings. As can be seen, this technique presents mixed results, with a few circuits obtaining positive results, most with negligible difference, and a few circuits even increasing in power consumption. The FM core contained no nets with only 1 or 2 levels of combinational logic and so was not applicable to this test run.

Individually, this technique proved the least successful and most difficult to work with. The clock tree paring, *N*-terminal net colocation, and area minimization utilize placement constraints, effectively making the placement part of the PAR tools power savvy and balancing the work load of the PAR tools well between the placer and the router, and little to no growth in runtime operation of the PAR tools was observed. The slack minimization technique however utilizes timing constraints, effectively putting both the power optimization and original timing constraint work loads onto the router portion of the PAR tools. PAR runtime increased sharply using this technique and it was observed that even though slack was minimized on the specified nets, unspecified nets would often experience a corresponding increase in wire length. Tightening the slack on too many nets would also result in the original timing specifications to be unable to be met. While individually this technique did not yield good results, as we will see in Section 5, this technique did prove useful when combined with the other techniques.

5. COMBINED POWER OPTIMIZATION

In the previous experiments, the four power optimization approaches are considered individually in order to determine the effects of the algorithm and learn more about power consumption, the underlying FPGA architecture, and the behavior of the COTS PAR tools. As we have observed, the clock paring technique yields good results, while the rest of the techniques provide mixed results. A more detailed analysis of the test circuits and our results shows that on a per net perspective, the clocks are the most dominant power consumers for all circuits in our test bench. Moreover, all of the techniques presented are complimentary, utilizing different constraint types, and can be combined together. So for the last experiment in our paper, we will consider clock tree paring to be a first order optimization that needs to be performed before we can truly measure the results of the second-order optimizations, N-terminal net colocation, area minimization, and slack minimization. As all of the techniques are complimentary we will consider the case where all of the constraints are applied to simplify our discussion.

Table 7 shows the overall results for the combined optimization techniques, the additional power savings over the first-order optimization, and the total power saved for each circuit. As shown in the table, 5 out of 10 benchmark designs reach their maximum power reduction by using a combination of techniques. In the referencing of Table 2, the circuits which seemed to respond well to multiple optimizations, CRC, Conv, and Mem, are all logic power dominated circuits. Clock power dominated circuits saw little to no benefit from combining constraints. The final power reduction ranges from 2.9% to 19.4%, and the average improvement is 10.2%.

Table 7: Combined power optimization results.

Design	Combined power reduction	Increase over clock paring	Max power saved (mW)
CRC	6.7%	0.8%	2
FM	2.9%	0.0%	3
VGA	12.7%	0.2%	17
USBF	10.7%	0.0%	9
PCI	19.4%	0.7%	8
Conv	7.1%	2.2%	9
DES3	8.6%	0.0%	12
Mem	3.3%	2.6%	21
S1	10.7%	0.0%	27
S2	19.4%	0.0%	116

6. HARDWARE VALIDATION RESULTS

In this section we seek to validate that the results we have seen in the previous sections utilizing XPower and our LITE tools are realizable in the real world. However, the real world brings other constraints that further complicate matters. For starters, the Osiris FPGA hardware boards have a fixed FPGA device, the V2 6000. While S1, S2, and Mem from our test suite target this same device, S1 and S2 assume different bus and memory interfaces than our hardware, and the Mem kernel did not produce enough dynamic power to yield statistically stable data with the resolution of our A/D board and the current sensors in our testbed.

Table 8: Hardware power measurement results.

Design description	XPower estimation (mW)	Hardware result (mW)	XPower: measure ratio
Baseline	351	281	1.25
Clock paring	334	270	1.24
Slack minimization	352	286	1.23
N-terminal net colocation	354	280	1.26
Area minimization	342	270	1.26
Combined	333	268	1.24

So for the purposes of this paper, we created a variant of the Conv circuit to be tested on the hardware. In this version, the Conv circuit was instanced 5 times in order to fill the device and achieve large enough power for measurement in our testbed.

The measurement results as well as the XPower estimation are shown in Table 8. The table lists the power results of the unoptimized design (baseline), the power optimized designs that use a single power technique, and the combined technique power optimized design. The second column provides the dynamic power consumption estimated by the Xilinx XPower tool. The third column is the power number measured on hardware. The final column calculates the ratio of the software measured values to that of the hardware measured values. So, while XPower seems to report a consistently higher value than that measured on hardware, the ratio is nearly constant, approximately 1.24. Power optimizations measured in software carry over into hardware. Though the absolute power varies, the relative percentage of power decreased remains relatively constant between software and hardware.

7. CONCLUSIONS

In this paper, we present a variety of techniques that seek to reduce power by feeding power driven constraints into COTS

PAR tools. These constraints seek to influence the FPGA implementation tools to place and route a user's design in a more power efficient manner. Four power optimization approaches are introduced in detail and are evaluated in Xilinx Virtex-II FPGA devices. The results show that the clock tree is the dominant dynamic power contributor and the clock tree paring approach is the most effective method to save power. The techniques are not mutually exclusive and clock tree paring can be combined with the other techniques to further reduce power. The average overall dynamic power savings on our test suite is 10.2%. Though our experimentation has focused on the Xilinx Virtex-II architecture, these techniques are expected to have similar results on other FPGA devices as well.

ACKNOWLEDGMENTS

The authors thank Michael Wirthlin, Kevin Lundgreen, and Nathan Rollins of Brigham Young University for their assistance with JHDL/EDIF infrastructure. This research was performed under NASA AIST Grant NAG5-13516, Reconfigurable Hardware in Orbit (RHINO).

REFERENCES

[1] J. H. Anderson, F. N. Najm, and T. Tuan, "Active leakage power optimization for FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '04)*, vol. 12, pp. 33–41, Monterey, Calif, USA, February 2004.

[2] M. French, "A power efficient image convolution engine for field programmable gate arrays," in *7th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD '04)*, Washington, DC, USA, September 2004.

[3] J. H. Anderson and F. N. Najm, "A novel low-power FPGA routing switch," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '04)*, pp. 719–722, Orlando, Fla, USA, October 2004.

[4] E. Kusse and J. Rabaey, "Low-energy embedded FPGA structures," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 155–160, Monterey, Calif, USA, August 1998.

[5] J. H. Anderson and F. N. Najm, "Power-aware technology mapping for LUT-based FPGAs," in *IEEE International Conference on Field-Programmable Technology (FPT '02)*, pp. 211–218, Hong Kong, December 2002.

[6] N. Rollins and M. J. Wirthlin, "Reducing energy in FPGA multipliers through glitch reduction," in *7th Annual International Conference on Military Applications of Programmable Logic Devices (MAPLD '05)*, Washington, DC, USA, September 2005.

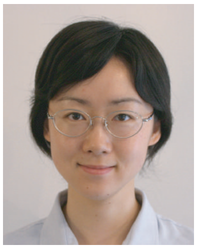
[7] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware FPGA CAD algorithms," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '03)*, pp. 701–708, San Jose, Calif, USA, November 2003.

[8] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in virtex-II FPGA family," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '02)*, pp. 157–164, Monterey, Calif, USA, February 2002.

[9] "Virtex-II Platform FPGAs: Complete Data Sheet," www.xilinx.com.

[10] *Xilinx ISE Software Manual*, www.xilinx.com.

[11] M. French, L. Wang, T. Anderson, and M. Wirthlin, "Post synthesis level power modeling of FPGAs," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 281–282, Napa, Calif, USA, April 2005.



Li Wang received the B.E. degree in electrical engineering from Tsinghua University, Beijing, China, in 1998, and the M.S. degree in electrical and computer engineering from the University of Maryland, College Park, in 2001, where she is currently pursuing the Ph.D. degree. She has been a Computer Systems Engineer since 2001 with the Information Sciences Institute, the University of Southern California working in Dynamic Systems Division. Her current research interests include low-power FPGA, low-power computing systems, analog VLSI, and biomedical engineering especially in heart models.

Matthew French is a Project Leader at the Information Sciences Institute, University of Southern California, and leads research in application mapping to embedded computing systems, incorporating novel computing architectures, ruggedized environment constraints, and tool development. He has over 10 years experience in the field of adaptive computing systems and holds 3 FPGA-related patents. Prior to USC/ISI, he worked at Lockheed Sanders on a variety of communications and SIGINT platforms. He received the Masters of Engineering and Bachelors of Science degrees from Cornell University, in 1996.

Azadeh Davoodi received the B.S. degree in electrical and computer engineering from the University of Tehran, Iran, in 1999, and the M.S. degree from University of Maryland, College Park, in 2002, where she is currently a Ph.D. candidate. Her research interests include design automation issues for ASICs and FPGAs in deep submicron fabrication technologies, such as power optimization and interconnect modeling.

Deepak Agarwal received the B.Tech. degree in electrical engineering from Indian Institute of Technology (IIT), Kanpur, in 1999 and joined Texas Instruments (TI) as an IC Design Engineer. At TI, he was part of the team that successfully designed the C28X DSP core. In 2001, he joined Proceler Inc. Atlanta as a Senior Systems Engineer where he worked on design problems related to reconfigurable computing. Following this, he enrolled at Virginia Polytechnic Institute and State University where he was a Graduate Research Assistant at the Configurable Computing Lab and received his M.S. degree in computer engineering in 2005. He is currently a Staff Hardware Engineer at National Instruments in the Distributed IO Group. His research interests include computer architecture, VLSI, reconfigurable computing, ASIC/FPGA design and testing.