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

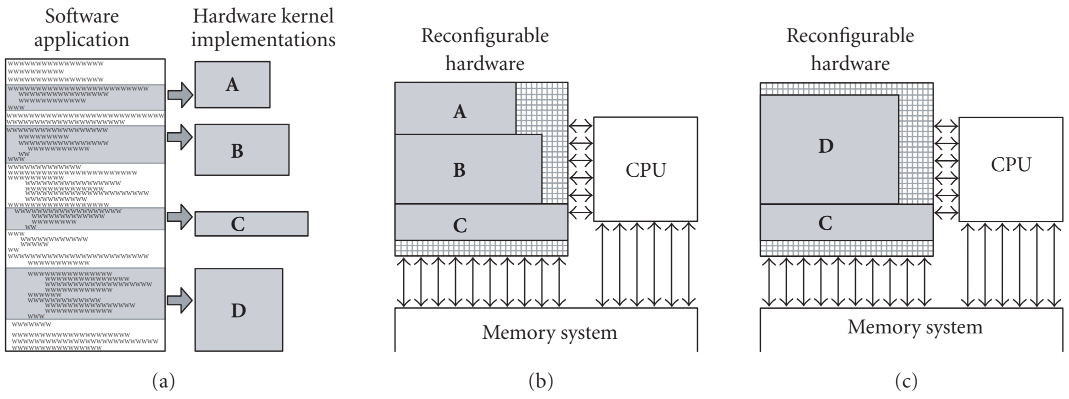


Figure 1: Reconfigurable computing implements compute-intensive application kernels (a) as hardware in RH and the remaining code in software on a CPU (b). Run-time reconfiguration allows RH to implement circuits that would otherwise not fit simultaneously (c).

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].

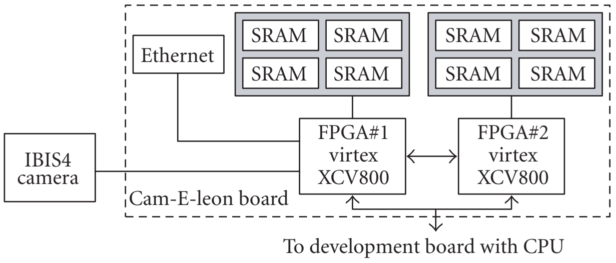


Figure 2: Cam-E-leon is a dynamically reconfigurable web camera platform from IMEC [57].

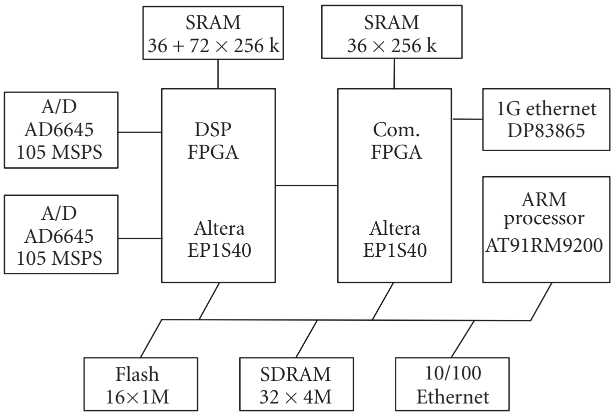


Figure 3: Block diagram of CASA: an embedded radar-based hazardous weather detection system using RH [45].

3.3. Example systems

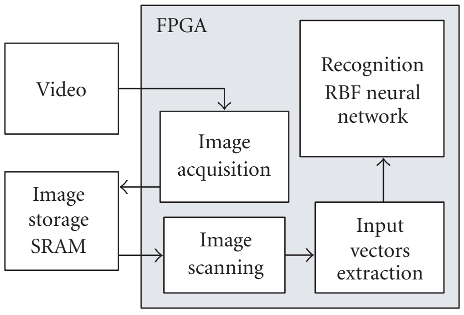
Embedded systems with RH span a range of sizes and complexities, some using many discrete RH components, with others primarily contained in an SoPC. Many of these systems use Linux or a modified lighter-weight Linux as an operating system because the source code is freely available for recompilation 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.

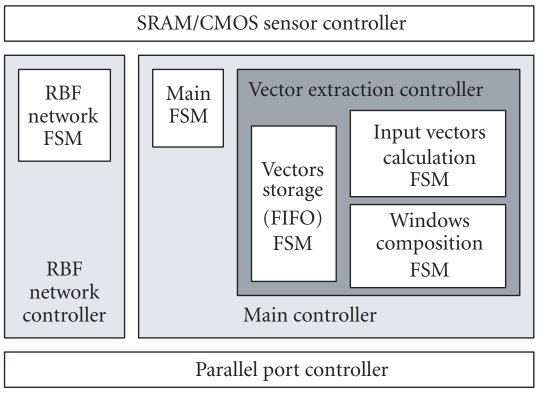
Cam-E-leon (Figure 2) is another image-related embedded system, designed in particular as a dynamic web camera [57]. This system is capable of downloading new image processing algorithms from a networked server and incorporating them into the system, implemented in RH. However, it is significantly smaller 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



(a)


SRAM/CMOS sensor controller
RBF network FSM
RBF network controller
Main FSM
Vector extraction controller
Vectors storage (FIFO) FSM
Input vectors calculation FSM
Windows composition FSM
Main controller
Parallel port controller

(b)

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,

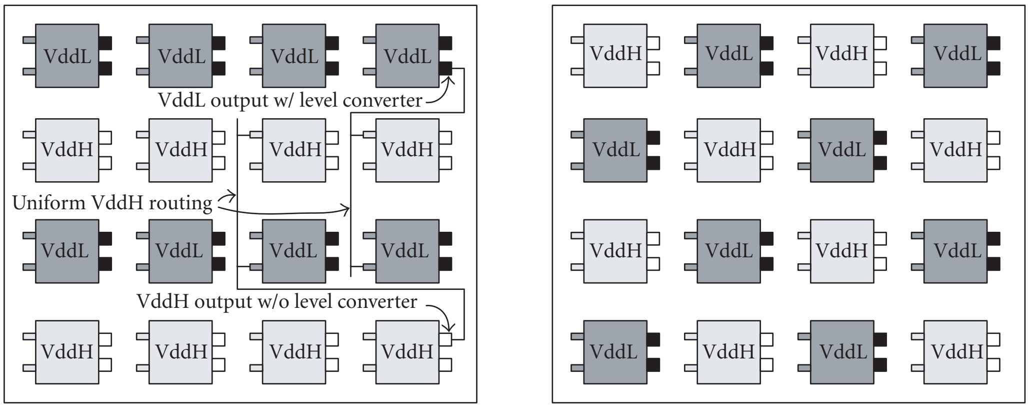

VddL
VddH
VddL output w/ level converter
Uniform VddH routing
VddH output w/o level converter

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,

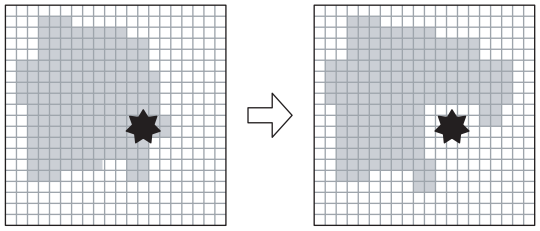


Figure 6: Faults (black) can be overcome by remapping affected configurations (gray) to nonfaulty areas of reconfigurable hardware.

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

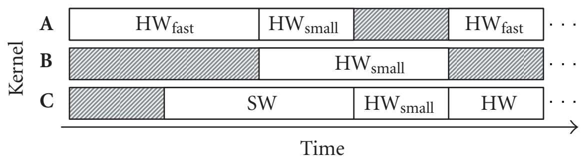


FIGURE 7: Different implementations (fast but large, small but slower, or software) for three kernels (**A**, **B**, and **C**) are shown over time. Shaded areas show when kernels are not needed. In this example, one fast or two small kernels can fit in RH simultaneously.

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 co-debugging 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.