

Designing BEE: a Hardware Emulation Engine for Signal Processing in Low-Power Wireless Applications

Kimmo Kuusilinna^{1,2}, Chen Chang², M. Josephine Ammer², Brian Richards², Robert W. Brodersen²

¹Tampere University of Technology
Korkeakoulunkatu 1, FIN-33720, Tampere
Finland
kimmo.kuusilinna@tut.fi

²UC Berkeley / BWRC
2108 Allston Way, Berkeley, CA 94704, USA
{chenzh, mjammer, richards, rb}
@eecs.berkeley.edu

ABSTRACT

This paper describes the design of a large-scale emulation engine and an application example from the field of low-power wireless devices. The primary goal of the emulator is to support design space exploration of real-time algorithms. The emulator is customized for dataflow dominant architectures especially focusing on telecommunication related applications. Due to its novel routing architecture and application specific nature, the emulator is capable of real-time execution of a class of algorithms in its application space. Moreover, the dataflow structure facilitates the development of a highly abstracted design flow for the emulator. Simulations and practical measurements on commercial development boards are used to verify that real-time emulation of a low-power TDMA receiver is feasible at a clock speed of 25 MHz.

Keywords: Rapid Prototyping, FPGA, Hardware Emulation, Low-Power, Design Flow.

1. INTRODUCTION

Hardware emulation is one of the most promising approaches to address the problem of constantly growing simulation execution times while simultaneously retaining high confidence on the results. Accurate simulations of large systems can be extremely slow and the reduced reliability of faster, more abstract simulation models cannot always be tolerated [27]. Especially logic-level development and verification of systems-on-a-chip (SoC) designs requires advanced methods. The purpose of the *Berkeley Emulation Engine* (BEE) project is to be able to emulate in real-time the digital portion of low-power communication chips and systems, while providing a robust and flexible interface to analog radio front-ends.

Figure 1 depicts the BEE infrastructure. The actual emulation is done with the *BEE Processing Units* (BPUs). Multiple BPUs can be connected together to form a larger emulation platform; a four-BPU BEE is shown in Figure 1. Each of the BPUs has a separate

connection to the *Host Server* through the dedicated Ethernet. The main purpose of the server is to allow remote access from *Client Workstations* to BEE. The server is also responsible for controlling BEE configuration and data read-back from the BPUs. Moreover, each BPU has 2400 external I/O signals, which can connect to expansion modules like a processor or other computation acceleration cards. Figure 1 depicts a cable connection to the *Analog Front-end*. Analog radios cannot be emulated on BPUs and, therefore, the reusability of BEE is supported with the utilization of external analog components.

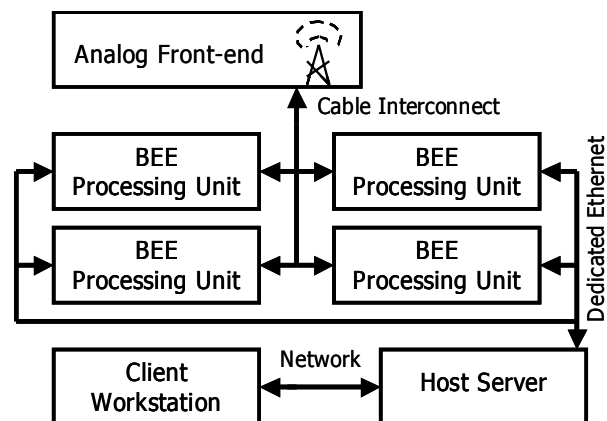


Figure 1. BEE hardware infrastructure.

Every BPU contains twenty FPGAs (Field Programmable Gate Arrays), each with a capacity of 518,400 ASIC (Application Specific Integrated Circuit) gate equivalents (2,541,952 FPGA gates) [37]. The primary reasons for such coarse grain granularity are maximizing application speed, I/O resource utilization between functional elements, and performance headroom within a unified configurable fabric that provides easy place and route for even large designs. The configurable logic itself, being look-up table and register based, is relatively fine-grain. Although approaches that time-multiplex area [10] or interconnections [1] could be utilized with BEE, the loss of performance associated with these methods is deemed too great. Especially when the real-time operation of the

algorithms is critical, slowing down the whole system under test may cause unexpected integration problems.

Specifically, BEE enables real-time emulation of a class of wireless communication applications [5, 18, 30], hence providing real-time feed back of algorithm optimizations, including bit-length and quantization selection. Once the algorithm has been defined, the same description can be used in an ASIC design flow, while maintaining cycle-to-cycle and bit-true equivalence.

This paper is organized as follows. Section 2 discusses the basics of hardware emulation and rapid prototyping focusing on the emulation methodology. Section 3 describes the BEE hardware architecture in detail focusing on the multi-FPGA board routing architecture, which is one of the primary distinguishing features in any multi-FPGA system. The BEE specific software flow, based on high-level Simulink [33] descriptions is explained in Section 4. Section 5 is an application example of a base-band TDMA receiver. Finally, Section 6 concludes the paper.

2. HARDWARE EMULATION AND RAPID PROTOTYPING

Rapid prototyping is here defined as developing a physical system that can implement the algorithm in a timely fashion. Rapid prototyping leverages high-level design entry and an automatic tool flow [9]. In the ideal case, the prototype operates at the same speed as the final product. *Hardware emulation* is a method for achieving this goal using synthesis-based design and configurable logic.

Hardware (HW) emulation addresses many of the most crucial problems in prototype development. Namely, it is possible to get working prototypes before the work is obsolete due to rapid technology changes or architectures becoming outdated. In addition, HW emulators typically aid with debugging, allowing more internal nodes to be observable. However, typical HW emulators are not tools for detailed gate-level timing analysis.

2.1. Design Flow for Hardware Emulation

Hardware emulators tend to follow the generic design flow depicted in Figure 2 [5, 9, 25, 29]. A typical design begins with behavioral and algorithmic descriptions in addition to other system specifications, which are converted to an RTL (Register Transfer-Level) representation. This conversion can be automatic, but more often than not this phase involves manual design work.

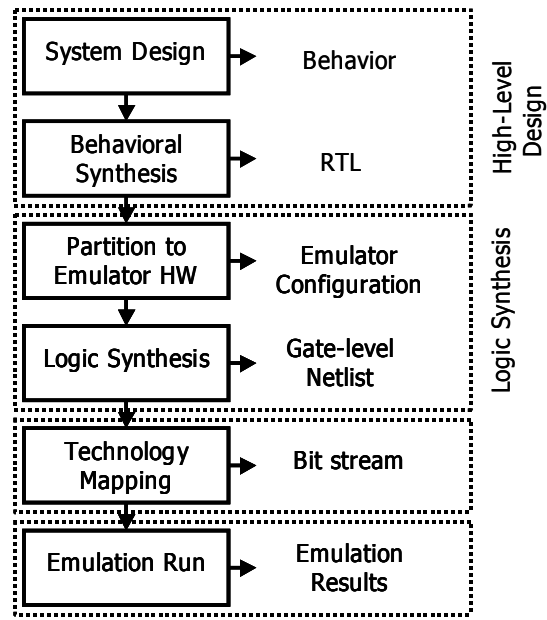


Figure 2. Generic HW emulator design flow.

Partitioning of heterogeneous resources is known to be a hard problem. FPGA-centric approaches for this problem are described, for example, in [19] and [21]. The system-level routing architecture has a heavy influence on this design phase as discussed in more detail in Section 3.

After the resource assignment, the design needs to be implemented with configurable logic, or if processors are available [22] with software, inside the actual emulation hardware components, which is a hard problem on its own. However, most FPGA vendors provide good tools for this purpose. The resulting configuration data is then downloaded to the emulator.

Finally, an *emulation run* produces the emulation results. These results can take a variety of forms as described in the following Section. Typically these results are the responses to test vectors or an ASIC replacement in a larger system under test.

One of the important questions within the emulation flow is what to do with critical paths. The basic approach is to slow down the emulation speed to meet the timing requirements. However, during the various levels of synthesis and HW mapping, it should be possible to optimize the back-annotated critical paths. For those signals with high fan-outs and especially for signals crossing physical chip and PCB boundaries, detailed and optional manual optimization can become necessary to obtain the best results.

2.2. Classification of Prototyping and Hardware Emulation

Using the abstraction level of the design as the distinguishing factor, three main classes of prototyping can be identified. *Concept-oriented* prototyping explores

the design space of requirements, specification, and feasibility. *Architecture-oriented* prototyping deals with system design, sub-system specifications, test development, and system-level verification. *Implementation-oriented* prototyping involves module design, in addition to RTL and even gate-level test and verification. [26]

Concept-oriented prototyping typically means hardware-accelerated simulation or computation. Many of the current signal processing algorithms are computationally expensive but parallel in nature. The speed-up from hardware acceleration can be crucial for the usability of these algorithms. An alternative to the simple dataflow paradigm for the implementation of these systems is described in [6], where FPGA local memory is exploited as caches. This allows an efficient construction of a DSP (Digital Signal Processor) like computing paradigm with multiple small processing units. Simulations can be accelerated with synthesizable test-benches and, of course, mapping the design itself onto the emulator hardware. Emulators usually have relatively large system memories, which can store a large number of test vectors and the corresponding responses. An alternative to this approach is to map the simulation algorithm itself on the hardware, as described in [24].

Rapid prototyping also facilitates the use of the prototype in the software development sense, enabling the demonstration of proposed product features and user interfaces. The prototype can already have correct peripherals, execution speed, and system memory.

One of the most important applications for architecture-oriented prototyping is to support hardware/software (HW/SW) partitioning and co-simulation. Co-development of software can usually begin earlier if a logic emulator is available. Simulations of complex composite HW/SW systems are often insufficient due to synchronization and performance issues. If only small time intervals can be simulated, correcting a detected error during the simulated period can cause other undetected errors. The new errors are only uncovered after the first error is corrected and the system run for a longer time. The goals of the simulation and the simulation models themselves are often not described with precise mathematical formulations, thus making simulations poorly suited for design validation. In addition, finding the actual partition between HW and SW is facilitated with the use of HW/SW co-emulation due to architecture exploration. Furthermore, developing real physical test vectors without the prototype may prove to be a problematic task. Real-time prototypes allow the designer to test and optimize more design parameters in shorter time and in an environment resembling very closely the target system. [3]

Many HW emulators also allow co-simulation with VHDL (VHSIC Hardware Description Language) or Verilog simulators running on workstations. This enables the use of C-language models within the

emulation, for example through the foreign language interfaces of the simulators. [32, 34, 35]

Implementation-oriented prototyping has several possible sub-tasks. Design changes during product development favors configurable logic and other programmable elements (processors) in the prototype. In addition, it is possible to mix traditional prototyping and emulation by adding physical components from the target design to the emulation system. This is sometimes called *in-circuit verification*. Real-time tests for the HW are important for integration purposes. For many multimedia applications, the operating frequency affects the perceived quality of the product, such as video encoders and decoders. Some prototyping platforms are built on top of *programmable interconnections* [5] with the idea that the actual HW utilized for the emulation run can be changed from prototype to prototype. Due to the additional components in the emulator, this strategy requires substantial I/O bandwidth and flexible clocking options.

In the hardware sense, a simpler option is to accelerate RTL or gate-level designs. This is the primary model for testing designs intended for ASIC implementation. Usually no exotic accessories are required and the strategy is fully supported by the emulator software. One of the advantages of this testing method over testing the ASIC implementation itself is the visibility of internal nodes.

A related activity is developing new IP (Intellectual Property) blocks and smaller library elements. Verifying the operation of these macros is typically a convenient operation for HW emulation platforms [11]. Particularly important is the need to verify the interfaces to other design components, all the more critical if the IP cores are offered in binary or encrypted form.

Some emulators offer special fault analysis capabilities enabling the emulation of stuck-at faults at the pin and net level [34]. Moreover, an emulator can act as a substitute system offering a temporary replacement for a unique system or allowing testing on an otherwise unavailable environment.

3. BEE HARDWARE ARCHITECTURE

Rephrasing the earlier discussion and to list the design requirements for the system, the primary goal of the BEE project is to provide a large, unified, real-time emulation platform for low-power dataflow designs. Plentiful resources are required in a unified fabric to facilitate design with parallel structures, ease the partitioning, and to avoid slow off-board connections that can also result in system integration problems. These requirements support coarse board-level granularity.

A driving requirement of the BEE project is the need to thoroughly test analog RF (Radio Frequency) front-ends in real-time. This is feasible for a number of target

applications because many low-power designs use only moderate clock speeds and have large I/O requirements. For example, an Ultra Wide Band (UWB) receiver, under development in a separate project, requires approximately 150 relatively high-speed (160 MHz) signals to connect to BEE. This suggests that the external I/O should allow modular expansion of BEE.

In addition, real-time operation means that the system needs to be able to optimize local routing on the board-level to control the delay in the data flow. This requirement excludes the use of purely hierarchical networks due to the inserted latency. To this end, the number of *hops*, chip-to-chip connections, to neighboring FPGAs must be minimized.

Low-power designs often encourage parallelism and contain multiple clock domains and, therefore, BEE needs to support this design style. Furthermore, to support the emulation of run-time reconfiguration, each of the system FPGAs is required to be individually configurable.

Each processing FPGA should have an external SRAM (Static Random Access Memory) attached to it to allow table-lookup and data buffering. In addition, this memory can function as system memory and cache for processors if needed. This memory should also provide a non-volatile storage during FPGA reconfigurations. In addition, HW emulator design goals are discussed in [20, 23, 28].

3.1. Routing Architecture Basics

One of the key aspects of any multi-FPGA system is its routing architecture and this is especially true for hardware emulators. The basic architectures analyzed in the literature are mesh [12, 17, 31], depicted in Figure 3a, and folded-Clos network, also called partial crossbar (Figure 3b) [2, 4, 29]. The performances of these networks are compared in [14].

In the folded-Clos network, all connections between FPGAs are made through crossbars. The FPGA device pins are divided into subsets. The number of subsets equals the number of crossbars on that hierarchy level. In a hierarchical system, the crossbars themselves are connected to other crossbars, which complete the interconnection. Traditionally, the connections in a Clos network are point-to-point with dedicated input and output switches. However, FPGA pads can be configured to be inputs, outputs, or bidirectional, which can be used to reduce the amount of required interconnection resources. A more detailed discussion on FPGA implementation of Clos networks is given in [4].

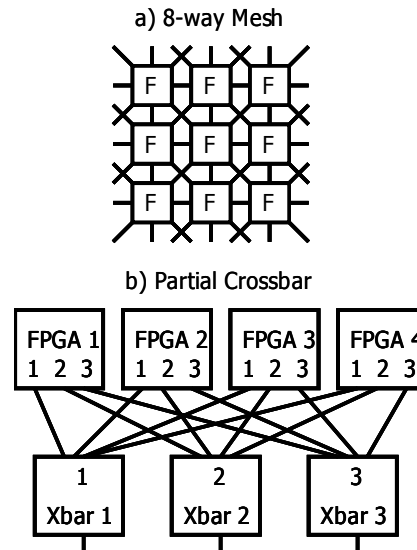


Figure 3. a) 6-FPGA 8-way Mesh b) 4-FPGA Partial Crossbar.

A combination of complete-graph network and partial crossbar, called Hybrid Complete-Graph and Partial-Crossbar (HCGP), was introduced in [15, 16] and corroborating performance results for this interconnection strategy were given in [13]. A hierarchical HCGP (HHCGP) is depicted in Figure 4.

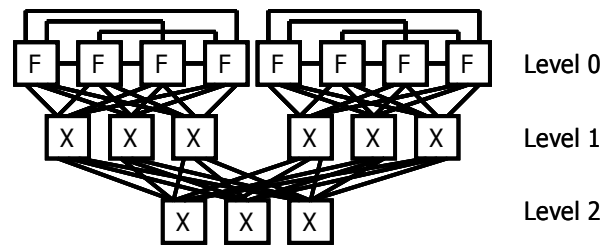


Figure 4. 8-FPGA Hierarchical Hybrid Complete Graph and Partial Crossbar.

Figure 5 depicts four-by-four matrices of FPGAs where each square represents a single FPGA. Each lattice also has an underlying communication architecture, namely an 8-way mesh, a folded-Clos network, or a hierarchical HCGP as indicated in Figure. The numbers indicate the number of hops necessary to reach other FPGAs in the lattice beginning from the FPGA marked with '0'. A number before a slash indicates the hop distance using the mesh interconnect or the complete graph. A number after the slash is the hop distance traversing the partial crossbar. A dash '-' indicates that the connection cannot be formed using that communication architecture. Analyzing Figure 5 and the discussion in [16] and [13] indicate that the HCGP has the superior routing performance from these networks.

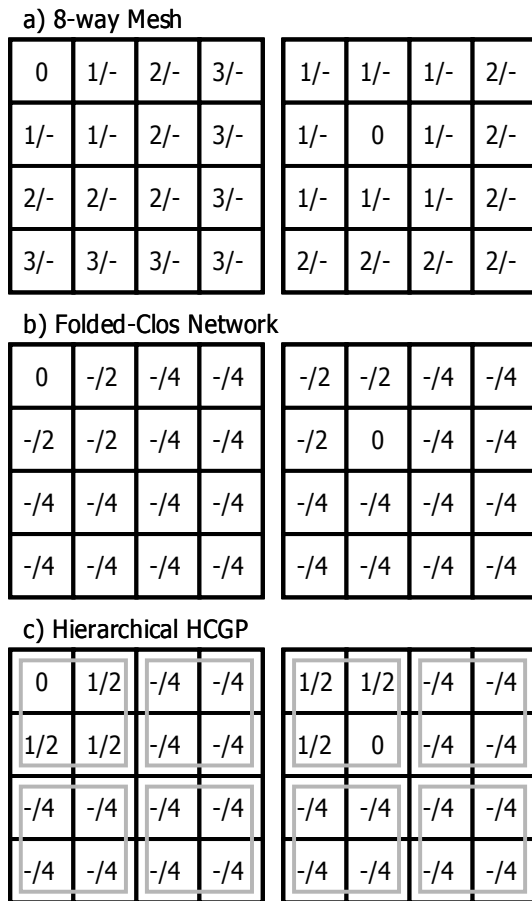


Figure 5. Hop-matrices for various networks. a) 8-way Mesh, b) Folded-Clos Network, c) Hierarchical HCGP.

3.2. Designing the BEE Routing Architecture

This analysis is based on the general requirements and design goals for the BEE from the beginning of this Section, the work reported in the references above, and the following practical assumptions. All the functional elements in the design are FPGAs, that is, there are no chips dedicated solely for interconnects. This is because we want to support designs that have computation as an integral part of their mapping to the hardware. Typical examples are the tree structures in certain neural network applications. In addition, it may be convenient to place control structures in the routing components.

All devices have 512 available I/Os, which is consistent with the Xilinx VirtexE 2000 -FG680 [37] chips that are available for this design. Moreover, the minimum useful connection width between elements is 36 signals since that breaks down to three 12-bit buses. Needless to say, this results in a large Printed Circuit Board (PCB), which must be routable in a reasonable number of layers. The practical size requirement on the PCB also places a hard constraint on the number of FPGAs on one board. Hence, we estimated that one PCB

could contain 16 processing FPGAs and their programmable interconnections.

Using the HHCGP as a starting point, several problems are immediately evident. HHCGP offers low-latency interconnects, that is, the complete graph, within a hierarchy segment, but all connections from one segment to another are through the partial crossbar network. Using non-hierarchical HCGP would solve this problem, but that is not very practical. The PCB could have routing problems and only a small number of signals is available for each interconnection due to the large number of arcs in the graph. Therefore, the 8-way mesh was selected as the local interconnection.

A large number of crossbar circuits is not desirable due to PCB component spacing constraints. However, the crossbars on each segment can be collapsed into one big crossbar, thus leaving four crossbars on hierarchy Level 1 and one crossbar on Level 2. Furthermore, the remaining crossbar on Level 2 is problematic. It adds a two-hop latency to the network and the logical place for this component is in the middle of the PCB where routing is expected to be congested anyway. Therefore, we decided to connect the Level 1 with a mesh structure effectively eliminating Level 2. The advantage of this arrangement is that the crossbar-to-crossbar latency is only one hop. However, some routing generality was sacrificed limiting the number of long-haul interconnections and direct routing expansion through the Level 2 crossbar. The resulting routing architecture is depicted in Figure 6 and called a *Two-Layer Mesh* (TLM) network. Since only half of the mesh and the FPGAs are shown, the numbers on the connection arcs indicate the number of links traversing off-network to the other half of the mesh or to BEE external I/Os. The structure is relatively regular and maps well to a PCB, as seen later in this Section.

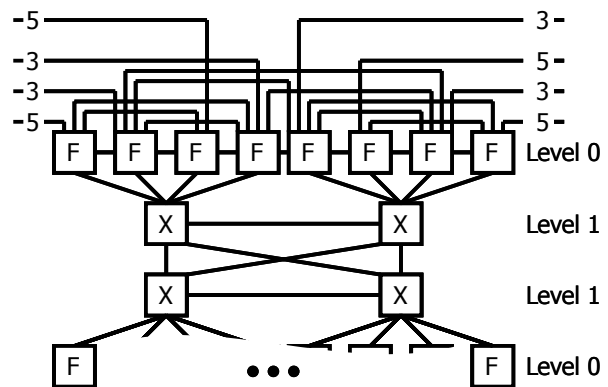


Figure 6. BEE Two-Layer Mesh routing architecture. Only half of the FPGAs and the 8-way mesh are shown.

The hop lattice for the TLM network is depicted in Figure 7. TLM has clearly an even better hop performance than HHCGP. However, it still remains to be seen if a reasonable number of signals can be assigned for each of the connection arcs.

0	1/2	2/3	3/3	1/2	1/2	1/3	2/3
1/2	1/2	2/3	3/3	1/2	0	1/3	2/3
2/3	2/3	2/3	3/3	1/3	1/3	1/3	2/3
3/3	3/3	3/3	3/3	2/3	2/3	2/3	2/3

Figure 7. BEE TLM network hop-matrices.

To keep the number of pins in the crossbars manageable, the minimum number of signals (36) is assigned to pass between each of the FPGAs and the crossbars. Therefore, the inter-crossbar links can be 96 bits wide.

For the FPGAs, 62 signal lines are required to connect the SRAMs to each FPGA and 13 pins are reserved for the configuration and read-back bus, leaving $512 - 36 - 62 - 13 = 401$ FPGA pins. Dividing 401 by 8 indicates that 50 signal lines for each connection are available to form the 8-way mesh. However, using 48 signals for this purpose (4×12 -bit bus) leaves a few signals for user I/O and hardware debugging. This arrangement fulfills the original requirement of no less than 36 signal lines per connection. In addition, it slightly favors the fast local interconnects, which is consistent with the dataflow paradigm.

Moreover, the emphasis on local interconnects supports our methodology for designing low-power circuits. Local wiring dominates when designing with direct-mapped components and highly parallel structures. These properties allow designs to be run at lower clock frequencies, and subdesigns to have long inactive periods, which can be exploited for power optimization.

3.3. System Overview

Most hardware emulators are designed to maximize reusability by using modular components that each individually possesses limited design capacity, while the integrated system cumulatively has the desired large design capacity. However, each BEE Processing Unit (BPU) is individually large enough to emulate systems with up to 10 million ASIC equivalent gates. This is sufficient to emulate many of the contemporary DSP and communication applications.

Depicted in Figure 8, the core component of a BPU is the *Main Processing Board* (MPB), which provides the basic emulation and computation capabilities for the system. Eight *Riser I/O Cards* (RICs) are vertically mounted to allow 2400 external connections off the BPU. A StrongARM-based *Single Board Computer* (SBC) is utilized to establish a connection between the BPU and the host server through a 10Base-T Ethernet link. A separate *Power Board* (PB), not shown in Figure,

along with two modular AC/DC converters, is capable of supplying the MPB and the SBC with up to 600 W.

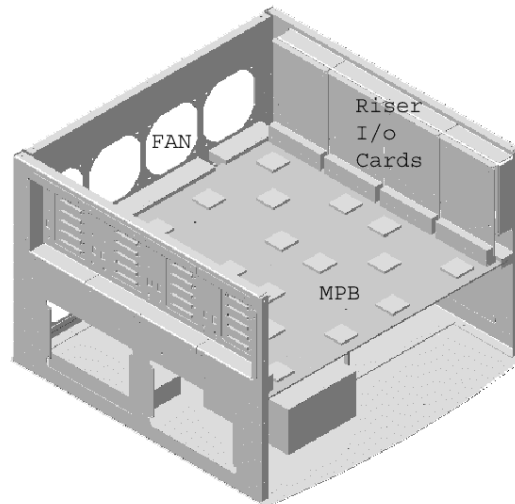


Figure 8. The BEE Processing Unit and its chassis.

3.4. Main Processing Board

Utilizing the described architecture, the Main Processing Board is a 26-layer PCB, with 20 Xilinx VirtexE 2000 chips, 16 ZBT (Zero Bus Turnaround) SRAMs, and 8 VHDM-HSD (Very-High-Density-Modular High-Speed-Differential) I/O connectors. The conceptual layout of this board is shown in Figure 9.

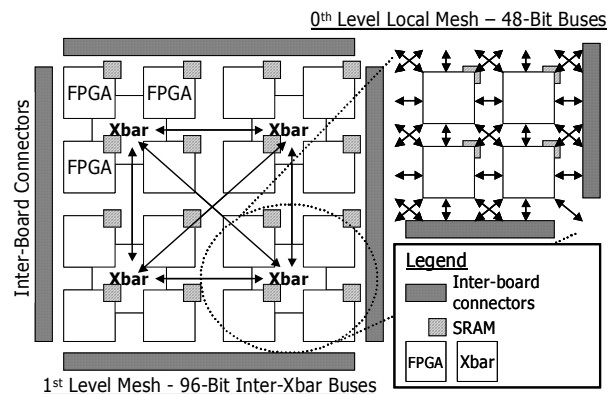


Figure 9. Main Processing Board interconnect structure.

The effectiveness of the external interconnects directly affects the system clock speed and the algorithm mapping capability of the emulator. The FPGAs on the periphery of the board have either 3 or 5 neighboring FPGAs, thus the rest of the links in the 8-way mesh are routed to the nearby off-board connectors (see also Figure 6).

Although using Low-Voltage TTL (LVTTTL) signaling standard is adequate for on-board interconnects, off-board connections have much higher speed and reliability requirements. The connection between the

analog front-end and the BPU can require high data rates over several meters of cable. Therefore, Low-Voltage-Differential-Signaling (LVDS) is used for BPU external links. The targeted external link speed is 160 Mbps per LVDS pair over 6 feet of twisted-pair cable. LVDS is not without its own problems. Mainly, LVDS signals are inherently asymmetrical. The driver has no termination, while the receiver requires a 100 Ω differential termination. This means the direction of a link is directly related to the physical characteristics of the signal traces. Both LVDS and LVTTTL signaling standards can be used on the riser cards, but the links are physically dedicated to implement one standard at a time.

Figure 10 depicts the MPB PCB layout image. The board dimensions are 58 cm by 53 cm. Table 1 tabulates statistics from the MPB PCB. Out of the total of 3400 components, approximately 2400 are bypass capacitors. Most of the signal traces are matched to 50 Ω and the minimum trace width is 5 Mils.

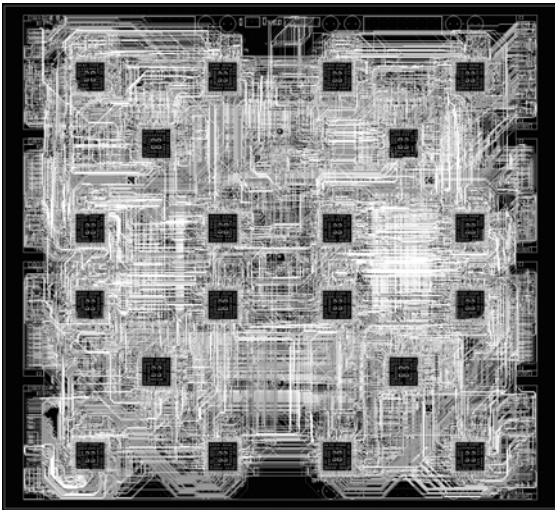


Figure 10. MPB PCB layout image, seen from the top.

Table 1. Main Processing Board PCB statistics.

Component Count	3,400
Total Pin Count	28,611
Layout Area (sq in)	427
Number of Nets	8,493
Number of Connections	19,877
Manhattan Distance (in)	45,950
Etch Length (in)	51,804
Number of Vias	32,334

Since the longest connection on board is below 40 cm, between Xbars, interconnect speed of 50 MHz can be practically achieved with LVTTTL. This was verified through simulations with trace parameters extracted from the PCB layout. External LVTTTL and LVDS signaling were tested using a commercial demonstration board.

3.5. Control Flow and Debugging

The use of an integrated Single Board Computer from Intrinsic in each BPU removes the requirement of an external controlling workstation. Control and configurations are provided through the SBC Ethernet interface. The SBC consists of a 206 MHz Intel StrongARM processor, 32 Mbytes of SDRAM, 16 Mbytes of Flash ROM, 10 Base-T Ethernet controller, and a compact flash slot for expansion. The SBC runs a Linux, kernel 2.4.9, operating system, an Apache web server, and other servicing programs. With the compact flash slot, additional storage, such as a micro drive or compact flash memory, can further expand the capabilities of the SBC.

Figure 11 depicts the information flow between the Host Server and the BEE system. First, the user generates the necessary design files using the BEE Design Flow. Second, the design files are sent to each BPU through the Ethernet and stored in memory or the hard disk of the SBC. Finally, the user issues commands to the SBC, instructing it to either configure the MPB or read back information. The Apache web server interface is used for common operations and status displays for additional convenience, while more complex operations are conducted through server daemons and shell commands.

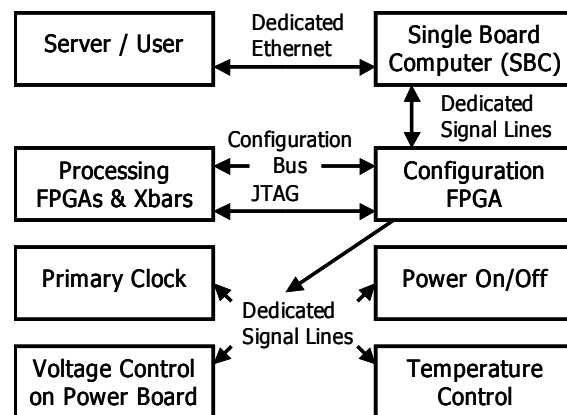


Figure 11. BEE Configuration Diagram.

The SBC connects to the 20 FPGAs on a MPB through a configuration FPGA, which mainly serves as a bi-directional signal multiplexer between the 16 general-purpose I/O lines from the SBC to over 100 control signals on a MPB. In addition, the off-board main power supply system is controllable through this link. All control functions on a BPU can be controlled from the SBC. The functions are divided into the following categories: programming of the processing FPGAs, data read-back from the processing FPGAs, clock domain control, power management, and thermal management.

Processing FPGAs can be programmed by the SBC using either the Xilinx SelectMAP mode or the JTAG (Joint Test Action Group) mode. The JTAG daisy chain

originates from the configuration FPGA, loops through all 20 processing FPGAs and back to the configuration FPGA. Although the JTAG interface is necessary for configuring the FPGAs and the read-back, the speed through this connection is very limited. Programming the entire board through the JTAG cable can take up to 10 minutes. Therefore, the SelectMAP mode is used for fast programming. The 12-bit configuration bus originates from the configuration FPGA and snakes through all 20 FPGAs. According to simulations, it can be driven by the SBC at 2 MHz. Hence, the programming time for the whole board is reduced to 20 seconds. Read-back of the FPGA signal states can be achieved using both the JTAG interface and the user bus mode.

A total of 13 different clock sources are available to each MPB. The primary clock provides a single synchronous clock domain throughout the entire board. The source of the primary clock is a digitally configurable PLL (Phase Locked Loop) clock driver. The output frequency can be digitally programmed between 1 MHz and 200 MHz by the SBC with an accuracy of under 5 ppm. The secondary clock consists of four independent clock domains throughout the MPB, one for each quadrant. A quadrant is a collection of four FPGAs and an Xbar. Each quadrant clock can be independently driven from an external SMA connector. The rest of the clock signals are provided through the external VHDM-HSD connectors to the quadrants. Half of these signals use the LVDS and the other half LVTTTL signaling. With this three-tier clock structure, a single global clock can be distributed to every FPGA with a maximum skew of 300 ps and the quadrant clocks can deliver additional clocks to the board facilitating designs with multiple clock domains. The internal Delay-Locked Loops (DLLs) in the FPGAs provide further clocking options.

At run time, each FPGA can consume up to 20 W and even unconfigured FPGAs draw some power. Therefore, to save energy, the main 600 W power supply is turned off by the SBC when the BPU is not in use. A separate 150 W power supply provides the always-on power for the SBC and the configuration FPGA as well as the chassis fan system. In addition, the I/O voltages delivered to the MPB can be configured by the SBC to be either 3.3 V for LVTTTL or 2.5 V for LVDS standard.

Although high quality passive heat-sinks and high air-flow chassis fans are used to cool the BPU, all FPGAs are continuously monitored by the configuration FPGA for thermal problems. A total of five remote sensor chips are utilized, each monitoring four FPGAs. The sensors are programmed at boot time to alert the configuration FPGA when junction temperature in any of the FPGAs exceeds 80 °C. If necessary, the configuration FPGA automatically shuts off the main power and alerts the user through the SBC.

On the MPB, less than 2% of the total signals on the board are directly accessible through probing headers. Using FPGA programming, internal signals could be

routed to these headers for direct probing with a logic analyzer. However, this is not enough for practical hardware debugging. Therefore, a software-based digital logic analyzer solution is used as the primary debugging tools for the MPBs.

Using Xilinx ChipScope Integrated Logic Analyzer (ILA) [37], small synchronous logic analyzer cores can be inserted into the design, acting as tiny logic analyzers at run-time. The ILA records the values of the monitored signals and transmits these through the JTAG interface back to the host workstation. The ChipScope software collects data from different ILA cores, which can reside on different FPGAs, and combines them onto a signal waveform display. However, due to the long JTAG chain on the MPB, the JTAG interface can only read-back signals with a data rate of less than 56 kbits per second.

In addition to the ILA cores, which use the on-chip BlockRAM as data storage, the external SRAM could be used for synchronous signal recording. Since the SRAM has a capacity of 1 MByte and it supports 32-bit wide accesses at 133 MHz, high bandwidth data can be stored in the SRAM. The read-back can occur at the end of the emulation run.

From the application perspective, the control flow allows the reprogramming of any FPGA at run-time. This property could be used, for example, to emulate multi-standard systems that time-multiplex hardware.

4. BEE SOFTWARE TOOL-FLOW

The primary requirement for the BEE design flow is to enhance designer productivity. It should not be necessary to re-enter designs when the designer proceeds from one abstraction level to another. This means a streamlined flow through high-level synthesis, logic synthesis, partitioning, technology mapping, and finally the emulation run. In addition, the model should expose the typical designer as little as possible to the actual hardware. Moreover, the flow should support the use of commercial software where adequate tools are available. However, the flow must be flexible enough to allow point-tools to be developed wherever the performance of commercial tools is deemed unacceptable.

Figure 12 depicts the rapid prototyping flow for BEE designs. Another very similar flow, based on Xilinx SystemGenerator software, is used for designs intended solely for emulation. This “design for emulation” flow attempts to achieve the maximum performance from BEE in terms of area and operation speed. However, the rapid prototyping flow described in this paper approximates the target ASIC architecture as closely as possible. Hence, the emulation performance may be compromised to favor a greater confidence in gate-level verification results. In accordance with the rapid prototyping paradigm, the flow supports high abstraction

level design entry from Simulink [33]. The flow facilitates early verification through cycle-accurate Simulink simulations, thus speeding up the normal prototype development process.

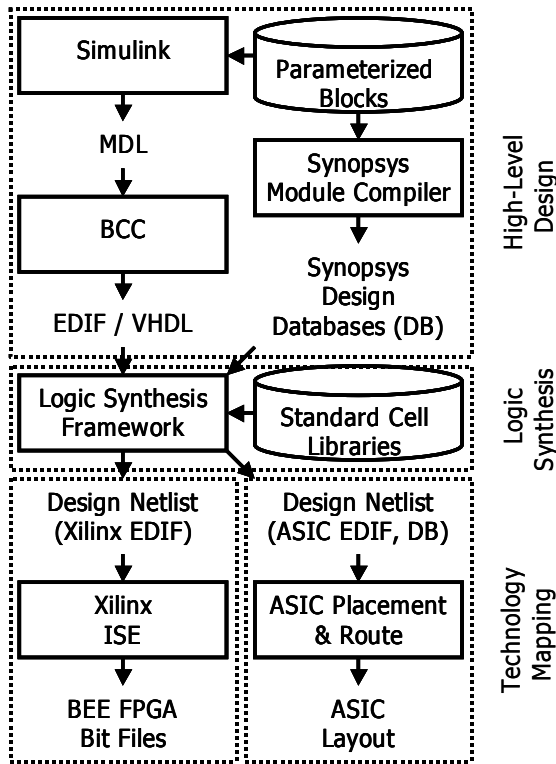


Figure 12. BEE rapid prototyping design flow.

The parts of this flow leading to ASIC implementations and especially a telecom-centric block library have been developed in a separate project called SSFAFT (Simulink-to-Silicon Hierarchical Automated Flow Tool) [8]. The blocks were first characterized and

implemented using Synopsys Module Compiler [36]. In addition, a corresponding block library was created for Simulink. Design entry uses these blocks and the design is stored in Simulink storage format (MDL). This phase may also contain user specified implementation hints like partition information. A program called BCC (Berkeley Cross Compiler) interprets the design and outputs EDIF (Electronic Design Interchange Format). Finite state machines are translated into VHDL descriptions.

For ASIC design, the logic synthesis framework (Synopsys synthesis tools) is primarily used for scan-chain insertion and synthesizing the control. For emulation, the tools allow the translation of gate-level ASIC designs to the FPGA technology. Moreover, synthesis can be used to optimize designs. Since the flow does not currently offer automatic partitioning to BEE, this design phase is carried out using scripting in Synopsys Design Compiler.

The resulting netlist is fed to a technology dependent backend tool-flow. The ASIC backend flow is typically heavily dependent on silicon vendor specific files and operations. However, the BEE side of the flow proceeds through the Xilinx ISE (Integrated Synthesis Environment) flow and produces separate bitstreams for each of the FPGAs actually in use. The BEE Configurator Program is responsible for programming the FPGAs on the BPU's.

There are several goals for the emulation system tool-flow. One goal is to minimize the number of required FPGAs, which also tends to minimize the inter-FPGA signals lines. This is important to avoid a possible design bottleneck. Another goal is to define the interfaces between FPGAs, a task that requires special care. In particular, the timing characteristics of these signals are important and good design practices like pipelining, registered outputs, and restricting the logic depth are typically necessary for off-chip signals. [5]

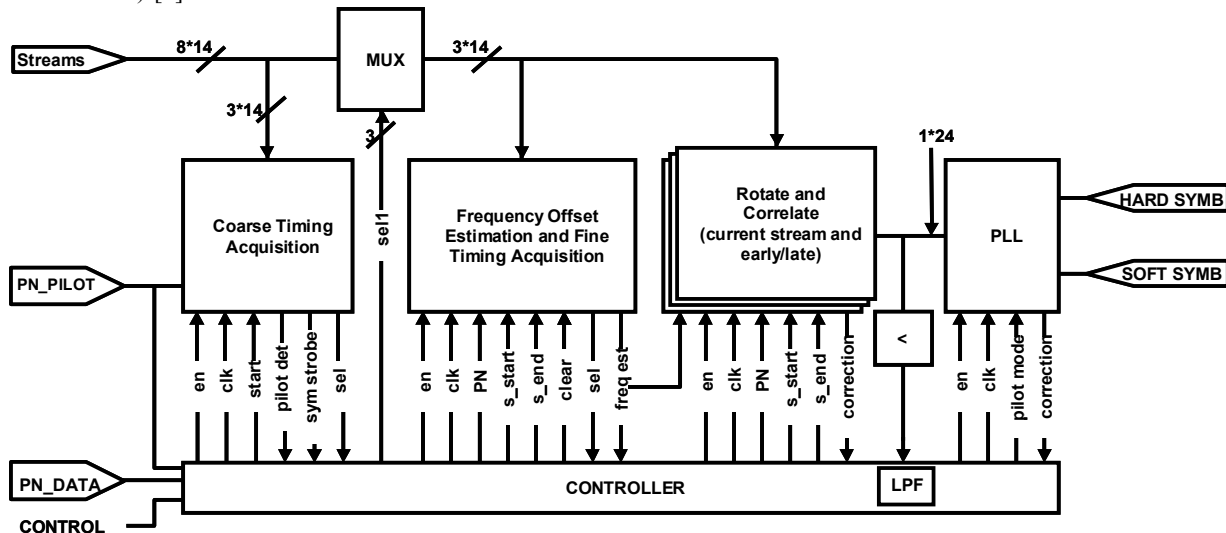


Figure 13. Low-power TDMA receiver.

5. BEE APPLICATION EXAMPLE

To illustrate hardware utilization and the details of the design flow, a baseband TDMA (Time Division Multiple Access) receiver is used as an application example. The receiver, documented in [7], was developed using the automated SSHAFT tool-flow. A block diagram of the low-power, spread-spectrum receiver is depicted in Figure 13. The design is intended to be a part of an extremely low-power wireless sensor network.

The receiver uses a direct-sequence code with a length of 31 to spread the spectrum and to provide resistance to narrow-band fading. The symbol rate is 806 kHz, which translates to 1.6 Mb/s data rate with QPSK (Quadrature Phase Shift Keying) modulation. In-phase and quadrature (I and Q) samples from an A/D Converter are fed into the design as 8-bit streams of 100 MHz (64 pins). They are interpolated on-chip to produce 7-bit streams at 200 MHz, corresponding to 8 samples per chip, each offset by one-eighth of a chip. The primary function of the receiver is to provide coherent timing recovery for the input stream.

The Coarse Timing Acquisition block uses a feed-forward non-data aided algorithm based on a matched filter to estimate timing within 3/8 of a chip. Three of the eight streams are correlated with the known spreading code until one of them passes a threshold. The Frequency Estimation and Fine-Timing block utilizes this stream and its two nearest neighbors to estimate the frequency and to further estimate the timing to the best of the three streams. The method utilized is a joint estimation feed-forward, data-aided algorithm synthesized directly from maximum likelihood equations.

The Rotate and Correlate block uses the frequency estimate to correct any frequency mismatch, the block correlates each stream with the spreading code and also tracks timing offset, which may necessitate changing the current stream. The PLL block gets the best stream and performs phase correction and tracks frequency and phase offsets. In addition, the PLL block produces the output symbols. Doing a coarse rotation and correlation outside the PLL allows a simple second-order PLL design operating at a low clock speed. The accuracy of the coarse rotation guarantees that this hybrid solution is less than 1 dB from the ideal where correlation is performed inside the PLL loop. The PLL initially operates in a data-aided acquisition mode for 19 symbols and then uses a decision-directed algorithm to track phase offset error throughout the remainder of the packet.

The ASIC implementation of this design has approximately 700,000 transistors covering an area of 2.4 mm². Altogether the design has 147 external I/O signals. The same design maps to several FPGAs in a BEE quadrant, as shown in Figure 14, which also depicts the external I/Os and the number of signals between the chips.

The Control logic is located in FPGA 1 since it shares pilot codes (64 bits) with the Coarse Timing function. In terms of routing, it would be expensive to separate these two blocks. The logical position for the Control would have been in the Xbar 1 because it needs to access and control all the blocks in the design. However, this partitioning does not affect the functionality or the real-time performance of the design. Aside from the controller, FPGA 1 is basically a large multiplexer. FPGA 1 feeds the selected 3 streams to Frequency Estimation (FPGA 2) and to Rotate and Correlate (FPGA 3). Moreover, FPGA 3 encompasses the digital PLL block and produces the outputs. To facilitate testing of the target ASIC, the symbol outputs in the ASIC design are routed off-chip along the test outputs. These test outputs happen to reside on FPGA 1 and, therefore, the symbol outputs are routed off-BEE from FPGA 1 as well.

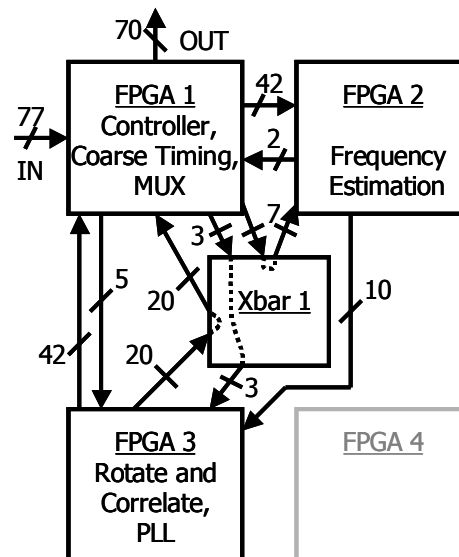


Figure 14. A TDMA receiver mapped to a BEE quadrant.

The design was manually partitioned in Synopsys Design Compiler using a synthesis script. The unmodified gate-level descriptions of the sub-systems from the ASIC design were utilized in this mapping. The high-level design in Simulink provided the necessary structural information to connect these sub-designs. In addition, this phase was used to automatically translate the design into a Xilinx FPGA specific netlist.

Each of the partitioned designs was then fed to the Xilinx ISE flow, which produced the bitstreams. As mentioned earlier, this design flow produces an emulation that is a very close match to the ASIC implementation and the design for emulation requires little effort. However, it does not fully exploit FPGA specific features, therefore making the emulation slow. A method for speeding up the emulation run through substituting critical blocks with FPGA specific designs

and discussion on the advantages and disadvantages of the respective methods are documented in [18].

Despite the apparent disadvantages of the FPGA and ASIC technology mismatch, the emulator architecture combined with the tool flow overcomes these limitations for the target applications. For example, the real-time operation requirement for the ASIC design is a clock frequency of 25 MHz. The synthesis tools, after placement and route to the FPGAs, indicate a maximum operating frequency of 25.0 MHz for the BEE implementation. Therefore, the real-time requirement is satisfied. The primary statistics for the FPGAs are tabulated in Table 2. The external I/Os are off-BEE connections and the internal I/Os connect to other components inside the BEE.

Table 2. Key statistics for the BEE implementation of the receiver.

	FPGA 1	FPGA 2	FPGA 3	Xbar 1
Ext. I/O	147	0	0	0
Int. I/O	121	61	80	60
Max. Clk (MHz)	28.8	25.0	26.1	10.3
Slices	9,466	2,961	5,395	0
BlockRAM	4	4	4	0

The Slices in Table 2 indicate the utilized area on the FPGA chip. Each of the FPGAs contains 19,200 Slices making FPGA 1 the most populated chip with 49% area utilization. The Slice counts include the overhead induced by the Xilinx ChipScope architecture and the ChipScope exclusively utilizes the BlockRAMs in this design.

The slowest chip is FPGA 2 with a maximum clock frequency of 25.0 MHz. FPGA 3 had an approximately 100 ns combinatorial path from its registered outputs to an internal Cordic block. However, this is a multi-cycle path that can afford to take 31 clock cycles to complete. Therefore, the maximum clock frequency for this chip was obtained with the multi-cycle path designated as a false path.

Since Xbar 1 does not contain any sequential logic, its operating speed is characterized as the maximum routing delay through the chip (10.3 ns). Based on simulations using trace models extracted from the PCB and IBIS driver models for the FPGAs, the maximum delay introduced by a hop is below 5 ns. Therefore, a worst-case estimate for 2-hop inter-chip critical routing is $2 \times 5 \text{ ns} + 10.3 \text{ ns} = 20.3 \text{ ns}$. This is well below the 40 ns cycle time imposed by the real-time requirement.

6. CONCLUSIONS

A novel architecture for the emulation and rapid prototyping of dataflow and low-power oriented hardware designs is introduced in this paper. The

architecture is shown to have good local routing properties. However, the global routing is somewhat penalized compared to known HW emulator architectures. This is acceptable due to the application specific nature of the BEE emulator. The architecture is also shown to be practically implementable on a PCB.

Mapping an existing design to BEE, in this case a low-power TDMA receiver, verifies in part that the architecture is suitable for emulating dataflow centric designs and that BEE emulations are compatible with the existing hierarchical and automated design flow. Simulations suggest that the design can be emulated in real-time using a primary clock frequency of 25 MHz.

Physically a BEE Processing Board is a 26-layer PCB containing 3400 components. The system was designed to handle signaling speeds of 50 MHz between any two chips on the board. Overall, a BPU has the emulation capability of approximately 10 million ASIC gate equivalents. Thus, we see BEE as an excellent specification and architecture exploration environment for full DSP systems. BEE offers real-time hardware, and secondarily software, prototyping and emulation, the possibility to prototype during all levels in the design trajectory, and a transparent software flow.

The future work in this project involves the finalization of the HW infrastructure and building additional analog front-ends for Ultra Wide Band (UWB) and Multi-Carrier Multi-Antenna (MCMA) applications. Moreover, we intend to streamline the software flow and include multi-chip support while primarily using the Xilinx SystemGenerator tool-flow. Especially the critical path backannotation problem, mentioned in Section 2.1 must be studied further.

7. ACKNOWLEDGEMENTS

Dr. Kuusilinna's work was supported by the Technology Development Center of Finland (Tekes), Jenny and Antti Wihuri Foundation, and the Finnish Cultural Foundation.

This work was funded by DARPA (the TDMA receiver under the PAC/C program, SIA GSRC), the U.S. Army Research Office, and the Berkeley Wireless Research Center supporting companies. In addition, we would like to thank Xilinx for donating the FPGA chips and the software tools.

8. REFERENCES

- [1] J. Babb, R. Tessier, and A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators," *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 142-151, Apr. 5-7, 1993.
- [2] M. Butts, J. Batcheller, and J. Varghese, "An Efficient Logic Emulation System," *Proc. 1992 IEEE Int'l Conf.*

- Computer Design: VLSI in Computers and Processors*, pp. 138-141, Oct. 11-14, 1992.
- [3] S. Cardelli, M. Chiodo, P. Giusto, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli, "Rapid-Prototyping of Embedded Systems via Reprogrammable Devices," *Proc. 7th IEEE Int'l Workshop on Rapid System Prototyping*, pp. 133-138, June 19-21, 1996.
- [4] P.K. Chan and M.D.F. Schlag, "Architectural Tradeoffs in Field-Programmable-Device-Based Computing Systems," *Proc. 1993 IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 152-161, Apr. 5-7, 1993.
- [5] M. Courtoy, "Rapid Prototyping for Communications Design Validation," *Conference Record Southcon/96*, pp. 49-54, June 25-27, 1996.
- [6] J.-P. David and J.-D. Legat, "A Data-Flow Oriented Co-Design for Reconfigurable Systems," *Proc. 1998 9th Int'l Workshop on Rapid System Prototyping*, pp. 207-211, June 3-5, 1998.
- [7] W.R. Davis, N. Zhang, K. Camera, et al., "An Automated Design Flow for Low-Power, High-Throughput, Dedicated Signal Processing Systems," *Proc. Asilomar Conf. for Signals, Systems, and Computers*, pp. 475-480, Nov. 4-7, 2001.
- [8] W.R. Davis, N. Zhang, K. Camera, et al., "A Design Environment for High-Throughput, Low-Power Dedicated Signal Processing Systems," *IEEE J. Solid-State Circuits*, Vol. 37, No. 3, pp. 420-431, Mar. 2002.
- [9] G. Doncev, M. Leeser, and S. Tarafdar, "Truly Rapid Prototyping Requires High Level Synthesis," *Proc. 1998 9th Int'l Workshop on Rapid System Prototyping*, pp. 101-106, June 3-5, 1998.
- [10] K.M. GajjalaPurna and D. Bhatia, "Emulating Large Designs on Small Reconfigurable Hardware," *Proc. 1998 9th Int'l Workshop on Rapid System Prototyping*, pp. 58-63, June 3-5, 1998.
- [11] H. Hakkarainen and J. Isoaho, "VHDL Macro Library Testing Using BOAR Emulation Tool," *Proc. 8th Ann. IEEE Int'l ASIC Conference and Exhibit*, pp. 105-108, Sep. 18-22, 1995.
- [12] S. Hauck, G. Borriello, and C. Ebeling, "Mesh Routing Topologies for Multi-FPGA Systems," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, Vol. 6, No. 3, pp. 400-408, Sept. 1998.
- [13] S.C. Jain, S. Kumar, and A. Kumar, "Evaluation of Various Routing Architectures for Multi-FPGA Boards," *Proc. 13th Int'l Conf. VLSI Design, Wireless and Digital Imaging in the Millennium*, pp. 262-267, Jan. 3-7, 2000.
- [14] M.A.S. Khalid and J. Rose, "Experimental Evaluation of Mesh and Partial Crossbar Routing Architectures for Multi-FPGA Systems," *Proc. 6th IFIP Int'l Workshop on logic and Architecture Synthesis*, pp. 45-54, Dec. 1997.
- [15] M.A.S. Khalid and J. Rose, "A Hybrid Complete-Graph Partial-Crossbar Routing Architecture for Multi-FPGA Systems," *ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays*, pp. 45-54, Feb. 22-25, 1998.
- [16] M.A.S. Khalid and J. Rose, "A Novel and Efficient Routing Architecture for Multi-FPGA Systems," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, Vol. 8, No. 1, pp. 30-39, Feb. 2000.
- [17] C. Kim and H. Shin, "A Performance-Driven Logic Emulation System: FPGA Network Design and Performance-Driven Partitioning," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 5, pp. 560-568, May 1996.
- [18] H. Krupnova, Dinh Duc Anh Vu, G. Saucier, and M. Boubal, "Real Time Prototyping Method and a Case Study," *Proc. 1998 9th Int'l Workshop on Rapid System Prototyping*, pp. 13-18, June 3-5, 1998.
- [19] H. Krupnova, C. Rabedaoro, and G. Saucier, "FPGA Partitioning for Rapid Prototyping: A 1 Million Gate Design Case Study," *Proc. 1999 IEEE Int'l Workshop on Rapid System Prototyping*, pp. 13-18, June 16-18, 1999.
- [20] D.M. Lewis, D.R. Galloway, M. Van Ierssel, J. Rose, and P. Chow, "The Transmogripher-2: A 1 Million Gate Rapid-Prototyping System," *IEEE Trans. VLSI Systems*, Vol. 6, No. 2, pp. 188-198, June 1998.
- [21] W.Y. Lo, C.S. Choy, and C.F. Chan, "Hardware Emulation Board Based On FPGAs And Programmable Interconnections," *Proc. 1994 5th Int'l Workshop on Rapid System Prototyping. Shortening the Path from Specification to Prototype*, pp. 126-130, June 21-23, 1994.
- [22] J. Madrenas, J.M. Moreno, E. Canto, J. Cabestany, J. Faura, I. Lacadena, and J.M. Insenser, "Rapid prototyping of electronic systems using FIPSOC," *Proc. 1999 7th IEEE Int'l Conf. Emerging Technologies and Factory Automation*, pp. 287-296, Vol. 1, Oct. 18-21, 1999.
- [23] S. Note, P. van Lierop, and J. van Ginderdeuren, "Rapid prototyping of DSP systems: requirements and solutions," *Proc. 1995 6th Int'l Workshop on Rapid System Prototyping*, pp. 88-96, June 7-9, 1995.
- [24] P. Sabet and L. Vuillemin, "An Approach to Mapping the Timing Behavior of VLSI Circuits on Emulators," *Proc. 12th Int'l Workshop on Rapid System Prototyping*, pp. 168-173, June 25-27, 2001.
- [25] F. Slomka, M. Dorfel, R. Munzenberger, and R. Hofmann, "Hardware/Software codesign and Rapid Prototyping of Embedded Systems," *IEEE Design & Test of Computers*, Vol. 17, No. 2, pp. 28-38, Apr.-June 2000.
- [26] B. Spitzer, M. Kuhl, and K.D. Muller-Glaser, "A Methodology for Architecture-Oriented Rapid Prototyping," *Proc. 12th Int'l Workshop on Rapid System Prototyping*, pp. 200-205, June 25-27, 2001.
- [27] R. Turner, "System-Level Verification-a Comparison of Approaches," *Proc. 1999 10th Int'l Workshop on Rapid System Prototyping*, pp. 154-159, June 16-18, 1999.
- [28] D.E. Van den Bout, J.N. Morris, D. Thomae, S. Labrozzi, S. Wingo, and D. Hallman, "AnyBoard: an FPGA-based, reconfigurable system," *IEEE Design & Test of Computers*, Vol. 9, No. 3, pp. 21-30, Sept. 1992.
- [29] J. Varghese, M. Butts, and J. Batcheller, "An Efficient Logic Emulation System," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, Vol. 1, No. 2, pp. 171-174, June 1993.

- [30] M. Vasilko, L. Machacek, M. Matej, P. Stepien, and S. Holloway, "A Rapid Prototyping Methodology and Platform for Seamless Communication Systems," *Proc. 2001 12th Int'l Workshop on Rapid System Prototyping*, pp. 70-76, June 25-27, 2001.
- [31] S. Walters, "Computer-aided prototyping for ASIC-based systems," *IEEE Design & Test of Computers*, Vol. 8, No. 2, pp. 4-10, June 1991.
- [32] www.ikos.com.
- [33] www.mathworks.com.
- [34] www.mentor.com.
- [35] www.quickturn.com.
- [36] www.synopsys.com.
- [37] www.xilinx.com.