

Rapid Design and Analysis of Communication Systems Using the BEE Hardware Emulation Environment

Chen Chang¹, Kimmo Kuusilinna², Brian Richards¹, Allen Chen¹, Nathan Chan¹, Robert W. Brodersen¹

¹University of California, Berkeley
Berkeley Wireless Research Center
2108 Allston Way, Berkeley, CA 94704
chenzh@eecs.berkeley.edu

²Tampere University of Technology
Institute of Digital and Computer Systems
P.O.BOX 553, FIN-33101 Tampere, Finland
kimmo.kuusilinna@tut.fi

Abstract

This paper describes the early analysis and estimation features currently implemented in the Berkeley Emulation Engine (BEE) system. BEE is an integrated rapid prototyping and design environment for communication and digital signal processing (DSP) systems, consisting of four multi-FPGA based processing units, each capable of emulating 10 million ASIC (Application Specific Integrated Circuits) equivalent gates at an overall system clock rate up to 60 MHz. This translates to over 600 billion 16-bit additions (operations) per second on one unit. An integrated software design flow enables the users to specify the design using a data-flow diagram, then automatically generates both the FPGA implementation for real-time rapid prototyping and a cycle-accurate, bit-true, and functionally equivalent ASIC implementation. For system-level design, the BEE hardware and software support rapid design turn-around and early performance analysis, without full synthesis or hardware mapping, from the high-level design entry. A case study detailing a turbo-decoder explains how the processing capability of the emulator can be utilized to verify a design using one billion input vectors with a speed-up factor exceeding 10^6 over equivalent software simulation methods.

1 Introduction

With the ever-increasing complexity and integration in communication systems, traditional software simulation of the system can no longer provide accurate modeling of the target hardware implementation in a timely manner. Therefore, FPGA-based hardware emulation and rapid prototyping have become an attractive solution, which can run up to 10^6 times faster than software simulation at the same level of modeling complexity. However, this speed-up at the run-time has been suffering from the time consuming FPGA design cycle, which typically starts from a RTL description of the system in either VHDL or Verilog. The translation from the original communication

algorithm to the RTL description is usually accomplished manually. Behavioral synthesis can be used to facilitate the process; nevertheless, it provides neither the predictability of the final implementation, nor the identical cycle and bit-level functionality between different hardware implementations. This gap between the algorithm and hardware description of the system not only increases the design cycle drastically, but also potentially introduces additional human errors into the design process. Given this labor intensive process, typically only the ASIC RTL description is produced, which is later translated on the gate-level into a FPGA netlist, and then emulated on a hardware emulator for functionality verification, at a system clock rate significantly slower than the target system speed. With this speed penalty, in most cases, it becomes very difficult to integrate the digital portion of the system with analog radio front-ends, which pose strict real-time requirements. Without a fully connected system, identifying potential integration problems early in the design cycle becomes an even harder problem.

The BEE system was designed to provide an integrated real-time hardware emulation environment for the digital portion of the system and enable algorithm designers to precisely control the outcome of multiple hardware implementations using only the high-level description of the system. In addition, the environment can integrate analog parts to form a fully-functional system replica. Instead of the RTL-level description of the design, the BEE design entry is a Simulink data-flow diagram using a set of parameterizable high-level block libraries that model the hardware functionality, cycle behavior, and bit-accuracy for a range of hardware implementations. By directly mapping the data-flow diagram to the target hardware implementation through the automatic design flow, corresponding functionality can be maintained between the FPGA and ASIC implementations with little performance penalty on either of the technologies. In addition, the design cycle between the fix-point block-level description of the design and the final implementation is drastically shortened through the usage of design flow automation, direct-mapped architecture, and high-level performance

estimation without synthesis. Therefore, the rapid turn-around time of design changes enables the users to explore a much wider range of the target design space enabling the analysis of multiple architectures and algorithms.

The rest of this paper discusses the hardware emulator in Section 2 and the integrated software design flow in Section 3. Section 4 presents a case study of a 3G cellular system like decoder design utilizing the BEE environment. Finally Section 5 concludes the paper. Due to the scope of this paper, the description of the BEE system concentrates on its features and applications on rapid system prototyping. More detailed description on the BEE system capabilities can be found in [1], [2].

2 The Emulation Engine

2.1 System Overview

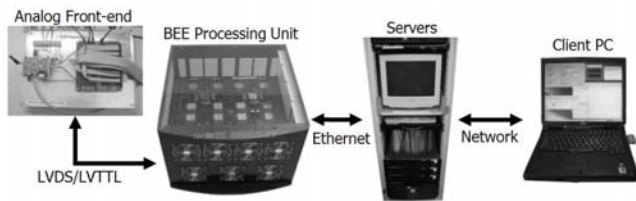


Figure 1: BEE emulation system overview

As shown in Figure 1, the BEE emulation system consists of four major components. The BEE processing unit (BPU) is the emulation engine with 20 Xilinx Virtex-E 2000 FPGAs arranged in a two-level mesh structure. A StrongARM based control module running Linux OS is attached to each of the BPUs for configuration, run-time management, and communication to external servers and clients through the built-in Ethernet interface. High bandwidth communication between the BPU and the analog front-end is provided through the eight riser I/O cards attached to each BPU.

Designs for emulation start as Simulink diagrams on a client PC and are compiled on the dedicated back-end servers into the proper implementation format. For emulation purposes, the final output is the FPGA bit configuration file and the emulation run.

2.2 Emulation Capacity and Performance

To properly emulate and prototype a communication system in an environment that models the final operation conditions as closely as possible, emulation capacity and overall system performance are the two critical hardware requirements for the emulation system.

Emulation capacity directly limits the size of the target design. Many modern communication algorithms have challenging requirements on the computational throughput, while maintaining low power consumption for mobile applications is important as an implementation target. Therefore, highly parallel low clock rate architectures can be used in such applications to achieve the throughput while minimizing the power [3].

The emulation capacity of the BEE emulator is verified using a reference design of 10240-tap FIR filter. The design utilizes all 20 FPGAs on a single BPU, with each FPGA implementing 512 taps of the FIR. Each tap consists of a 16-bit adder, 12-bit fix-coefficient multiplier, and a register. The design achieves a system clock rate of 28.5 MHz with continuous streaming input data. By counting the addition and multiplication as equivalent operations, the overall computation throughput exceeds 600 billion operations per second. The FIR design has an equivalent ASIC gate count of 8 million. By connecting all four BPUs through the external I/O cards, a total of over 30 million ASIC equivalent gates are available to the user. In addition, each of the 16 processing FPGAs has one external 1 Mbyte SRAM attached that is capable of operating at 110 MHz. The connection is 32-bit and therefore provides approximately 440 Mbytes per second off-chip memory bandwidth for high-speed data caching and other data storage-related operations.

In addition to the large design emulation capacity, a high overall system operation rate is not only crucial to meet the real-time requirements of the target application, but also essential in matching the emulation hardware to the final ASIC implementation of the design. In a multi-FPGA system, such as BEE, inter-FPGA connection speed and FPGA granularity are very important factors determining the overall system performance. In the BEE system, since the design goal is to provide a uniform high speed reconfigurable array on a single PCB, large capacity FPGAs are used to reduce the need for off-chip connections, while providing ample inter-chip direct parallel connections when needed.

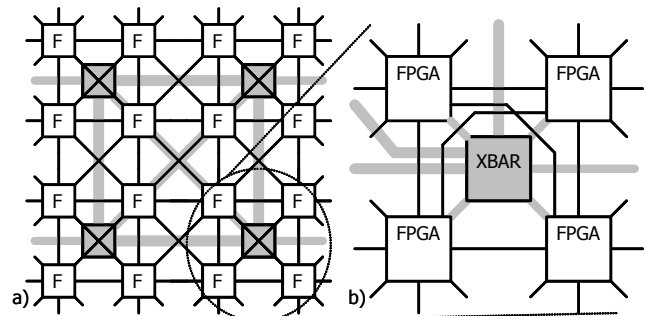


Figure 2: BEE interconnect structure, a) all twenty FPGAs and b) detailed routing in one

board quadrant, emphasizing the nature of crossbar FPGAs.

On each BPU, the 20 FPGAs are divided into two groups. The processing group consists of 16 FPGAs arranged in a 4-by-4 array, interconnected in an 8-way mesh structure with 48-bit point-to-point bidirectional connection between any two adjacent FPGAs in the group. The second group of 4 FPGAs forms a second layer mesh on top of the first layer mesh. These 4 FPGAs are called “XBARS” to distinguish them from the processing FPGAs. The processing array is logically divided into 4 quadrants. Each quadrant has one XBAR, which connects to all 4 FPGAs in the same quadrants through 36-bit connections. The four XBARS are additionally connected to each other through 96-bit connections. The resulting structure is called a two-layer mesh and depicted in Figure 2. All 20 FPGAs on the BPU can be used for both datapath components and control logic. For example, the 16 processing FPGAs can form a tightly connected datapath, while the XBARS are primarily used for centralized control logic and long range inter-FPGA communications. In addition, the XBARS can be configured to model non-ideal communication channels between multiple nodes implemented on different FPGAs.

Between the FPGAs on the same BPU, a 60 MHz connection speed has been verified using the LVTTTL signaling standard on all BPUs. Locally within the same quadrant, FPGAs can communicate at as high as 100 MHz; however, to simplify the programming model, the lower bound of 60 MHz is used as the guaranteed register-to-register speed between any directly connected FPGAs. Figure 3 illustrates the number of hops needed for inter-chip communications from a corner or center FPGA. The two integer numbers in each square denote the two routing scenarios, either the first or second layer mesh, respectively. Therefore, any FPGA can communicate to any other FPGA on the same BPU within three clock cycles of latency.

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 3: Inter-FPGA connection hop counts

2.3 Run-time Data I/O

In a rapid prototyping environment, the external data I/O interface directly affects the flexibility of the overall system

as well as the overall performance when connected with the analog portions of the target design. The BEE system is designed to support two run-time data I/O modes: *real-time emulation* mode and *hardware acceleration* mode.

In real-time emulation mode, the emulator is directly connected to the analog front-end through the riser I/O card. The goal is to prototype both the digital and analog parts of the design in one framework at the final operation speed of the target design. Therefore, the data I/O speed and bandwidth is crucial to meet the requirements set by the analog front-end. The riser I/O cards support either LVDS or LVTTTL signaling by substituting a different resistor network on the cards. Each riser card fans out 300 single-ended or 150 differential pair signals to six 68-pin external connectors. When configured to use the LVDS signal standard, a reliable signaling speed of 160 MHz has been verified on the links. This aggregates to approximately 192 Gbits per second raw bandwidth per BPU using all the eight riser I/O cards. Typically, the analog front-end is directly connected to the FPGA to reduce latency. If custom data buffering is needed for synchronization, the FPGA can be configured to meet the application requirements. Therefore, a wide range of analog front-ends can be readily connected to the emulator with minimal changes to the interface. For example, in an ultra wide band radio application prototyped using the BEE system, the ADC samples the analog signal at 1.2 GHz, thus creating a continuous data stream of 9.6 Gbps. After parallelizing the data stream on the analog board into eight 160 MHz streams, the data can be easily transferred through three one meter long cables to the BEE emulator, and then further parallelized on the FPGAs down to 40 MHz streams for digital processing.

As an alternative to hardware emulation, the BEE can also be used for hardware acceleration. In this mode, since the goal is to emulate and functionally verify a portion of the digital system with extensive test vectors, a high bandwidth I/O interface is not always necessary. Rather a flexible interface that can be easily integrated with the Simulink design entry and simulation environment would be desirable. Figure 4 illustrates the hardware-in-the-loop data I/O interface used in the BEE system. Using the Matlab control GUI, users can connect to the BEE emulator from anywhere on the network. All necessary operations, such as configuring the FPGAs, download or upload of data at run-time, and changing the main system clock rate can be accomplished remotely.

The Matlab control GUI sends data or control packets through the network to the Ethernet interface on the Strong-ARM Linux module where a software daemon controls the operations of the FPGAs through a 14-bit bus. On each running FPGA, a special control module is inserted into the user design, which consists of an embedded bus controller

and two dual port SRAM blocks. From the user design perspective, the control module operates as an ideal SRAM with two address ports, one for read and the other for write. Internally, the two dual port SRAM blocks operate independently to support simultaneous read and write from the user design. The embedded controller arbitrates the memory operations sent from the software daemon to the two SRAM blocks, and also controls the on-chip clock used by the emulated design. Therefore, design states can be maintained during run-time data operations.

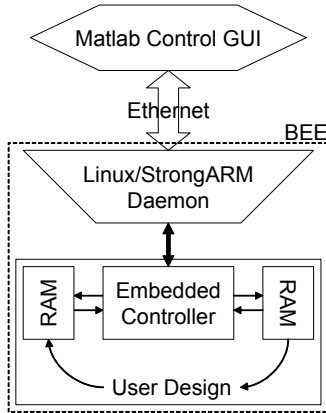


Figure 4: Hardware-in-the-loop Data I/O Interface

The overall data throughput is limited by the 10Base-T Ethernet interface on the Linux module maximally to 1 Mbytes per second, which is typically much slower than the data processing rate of the user design emulated on the FPGA hardware. Therefore, input data needs to be first stored in the SRAM before it can be batch processed by the hardware. The emulation results are sent back to the user to be viewed on a Matlab controlled GUI. Since this is not very efficient when dealing with a large amount of input data, the alternative is to generate the input data with the FPGA hardware. Illustrating a common approach in communication system designs, a small amount of known input data is first used to verify that the hardware emulation works identically to the Simulink simulation. Then a large amount of data is generated on-the-fly using hardware pseudo-random generators on the FPGA to further verify the correct behavior of the system as well as its performance. Typically millions or billions of input data vectors are used and the output is in the form of performance metrics calculated with the FPGA hardware, such as the bit error rate or packet error rate. Furthermore, communication channel characteristics and impairments can be modeled with the FPGA hardware as well, providing a controlled environment for the users to debug their designs. Section 4 will illustrate this process with a design example.

3 Integrated Design Flow

The basic approach has been to build this design flow using commercially available tools whenever possible and to use batch processing and scripting to allow the flow to be driven from the high-level description, a Simulink data-flow diagram, and automatically generate cycle-accurate bit-true hardware implementations for both FPGA emulation and ultimately an ASIC chip. Conventional implementation using a low-level hardware description language is thus avoided. Since the design flow is very complex with many detailed steps, this discussion concentrates on three things: rapid design iteration and optimization, abstraction model equivalency, and high-level performance estimation.

3.1 Rapid Design Iteration

The design and analysis time for a particular design instance can be very dependent on the quality and speed that feedback is available from lower abstraction levels. A crucial optimization goal is the turn-around time when a change occurs at the top-level. The BEE estimation and verification flow, depicted in Figure 5, has three feedback loops (the black arrows on right in Figure) to provide information swiftly to the user. Architecture exploration is possible only if these loops are tight enough to allow the user to evaluate several architectures in timely manner. In addition, Figure 5 depicts the verification flow, in which the Simulink simulation determines the test-vectors that are automatically applied to validate the design on multiple lower abstraction levels.

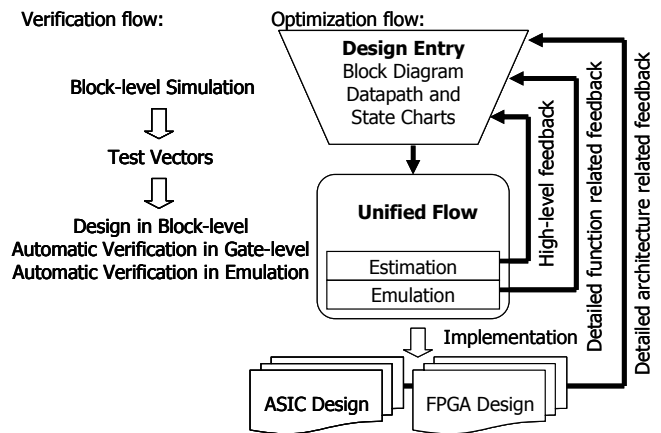


Figure 5: BEE design verification and optimization flows.

The emulation loop provides functional verification for the design, testing the algorithm and the architecture on hardware. The compilation run-time for the emulation depends on the design (and other factors), but a run-time of

less than an hour is typical for a FPGA populated up to 70% of the total area. The emulation run-time is heavily dependent on the nature of the test, the size of test-vectors, and the clock speed of the design. Finally, the implementations on an ASIC or a FPGA provide the final verification that the design operates properly and within the specified constraints, such as area and power.

With the emulator running at or near the intended clock speed of the final product, tests can be run on a design with real-world I/O if the rest of the system exists, test-vectors can be fed from mass storage devices, or test-benches can be compiled into the emulator to achieve comprehensive verification. Similarly, performance and functionality verification can be implemented on the emulator. In addition, these methods facilitate data collection over long periods of time, for example to validate signal to noise ratios or bit error rates, which may be so low that long tests have to be run to detect any error at all.

Design changes based on estimation can be done within minutes and emulation driven changes usually take a couple of hours. ASIC layout implementations can take a very long time, but it is the aim of this research to keep the tool flow run-time within one day. For example, a 1Mbit/s transmitter with an area of 0.28mm² in 0.13μm CMOS technology took 27 seconds to estimate the area. Correspondingly, the synthesis for emulation took 162 seconds and the ASIC layout 57 minutes. These times do not include simulations or emulation since those run-times are very dependent on what kind of information the designer is trying to gather. The exact values are relatively unimportant, but we find the speed-up factor of 127:6:1 from layout to synthesis to estimation in this example to be illustrative. The differences between these feedback abstractions tend to be even more pronounced with larger designs.

3.2 Implementation Flow and Equivalency

The top-level design implementation flow is shown in Figure 6, which begins with a block diagram description of the algorithm that explicitly defines the data dependencies in a cycle accurate simulation. In addition, control is described by a graphical state machine entry that co-simulates with the data-flow description. This simulation can be performed in non real-time on a standard computer or by mapping to a library of equivalent blocks, allowing efficient FPGA and ASIC implementations. The blocks used in the data-flow description have a direct correspondence to the FPGA and ASIC libraries, which are all bit true equivalents. The FPGA libraries are based on vendor optimized library components and macros and the ASIC libraries are based on manually optimized architectures using Synopsys Module Compiler [4] as the datapath optimization tool. These parametrizable libraries

provide a method to extend the descriptive power of the flow.

The correspondence between a FPGA and an ASIC implementation is functional. Both implementations are cycle and bit-accurate representations of the original description, but the internal hardware architecture may vary. Different low-level architectural optimizations for different target technologies are necessary to optimize the implementation so that the clock rates of both implementations can be approximately equivalent with typical clock speeds for applications on the order of 50 MHz on both platforms. The relatively low frequency clock rates are a result of the emphasis on low-power design practices and the use of a high level of parallelism to achieve the necessary throughputs.

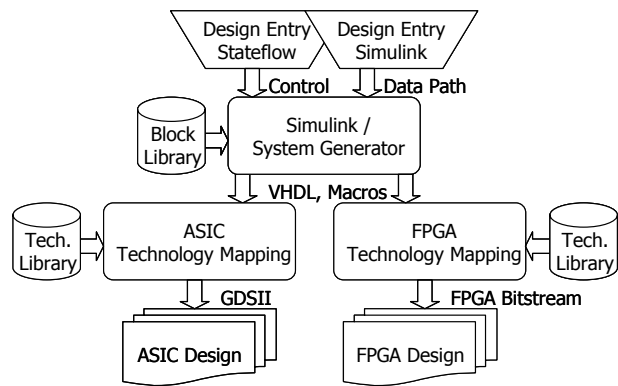


Figure 6: BEE Implementation design flow.

3.3 High-level Estimation

Another key feature of the BEE design flow is the ability to generate high-level performance estimations on the same abstraction level as the design entry without detailed hardware synthesis. The goal is not to calculate exactly the absolute numbers of the performance metrics, but rather to facilitate the evaluation of different architectures and tradeoffs. Therefore, area, speed, and power of the design can be estimated on each of the library blocks based upon its parameters, such as bit-width, pipeline stages, latency, micro-architecture, supply voltage, and target clock rate. For a simple block with a predictable structure, direct formula calculation, based on its parameters, is good enough. However, for complex blocks with the necessary formula too convoluted to be derived, a database of pre-characterized performance values is used in conjunction with linear extrapolation. FPGA area (or resource utilization) estimated in this way is typically within 20% of the actual result. On the other hand, ASIC area estimation can be 30% off in extreme cases, for example due to inter-module routing overhead.

However, system-level estimation can happen in a matter of minutes instead of hours or days that are typical for detailed simulations after hardware synthesis. An additional advantage of this approach is that since the high-level description contains all the information necessary to drive the design flow, estimates can be made of power, area, and speed of the final design. This allows the algorithm designer to trade-off algorithm and architecture issues to rapidly and efficiently optimize the design, while still maintaining full control over the final hardware implementation.

4 Case Study

To illustrate the usage of the design flow, a MAP decoder using the BCJR algorithm [5], targeting FPGA hardware acceleration, is used as an example. The BCJR decoder is used as part of a turbo decoder design, documented in [6]. Figure 7 depicts the top level representation of the design with hardware testbench described in Simulink.

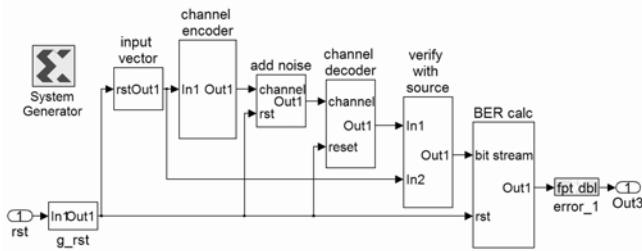


Figure 7: Turbo decoder design in Simulink

Each of the blocks represents a particular subsystem, which contain multiple hierarchies of children subsystems and leaf library components. The source data (or input vectors) are generated using a hardware pseudo-random number generator. The generator produces a uniform distribution of numbers and is implemented using a cellular-automata algorithm [7]. The channel encoder encodes the source data at 2/3 rate to 1/3 systematic parallel concatenated convolutional code produced from generator polynomials $1+D^2+D^3$ or $1+D+D^3$. The actual channel is modeled as all-white-Gaussian (AWG) with configurable signal-to-noise (SNR) values. The AWG noise is generated through the summation of eight independent uniform pseudo-random number generators, similar to the one used for source data. The SNR is controlled through the multiplication of the AWG noise with preset scaling factors. After the channel, the received data is decoded using the BCJR decoder, and then the decoded data is verified with the original source data for bit-error-rate (BER) calculation at the end. Finally, the total number of bit error values are recorded in a SRAM for each of the 16

preset SNR values, and for each SNR value, one billion input vectors are used for the BER calculation. The experiment result is captured in Figure 8.

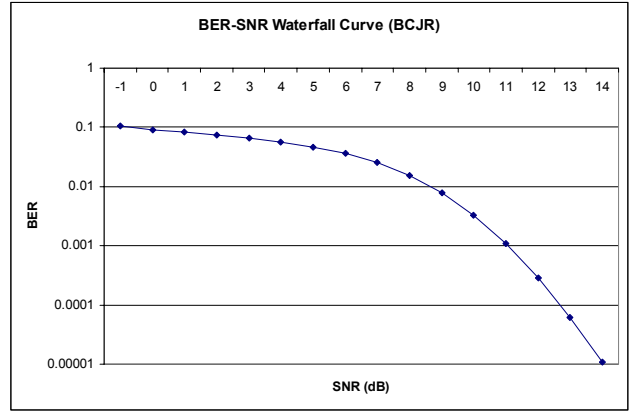


Figure 8: BCJR decoder waterfall curve

The entire design fits easily in a Xilinx Virtex-E 2000-6-FG680 FPGA with a maximum clock rate of 13.2 MHz. Running at the maximum frequency, the whole BER experiment took 20.2 minutes on the FPGA. The compilation process from the Simulink design input to final FPGA bit configuration file took 44 minutes, out of which, 17 minutes were used for VHDL netlist and function core generation and 27 minutes for FPGA backend synthesis, mapping, and placement & routing.

High-level area estimation was used to determine the size of the design early in the design cycle. The estimation time of the final design is a little over 9 minutes, which is less than a quarter of the whole compilation time. Table 1 lists the comparison between estimated and actual resource utilization of the design. Usually, dedicated FPGA components, such as the block RAM, are estimated with high accuracy, because the usage of such components corresponds directly to the library components used in the design. On the other hand, utilization of more flexible hardware components, such as the look-up-table (LUT), is often subject to logic optimization and timing driven synthesis, therefore 20% estimation error can be expected. Finally, somewhat complex resource unit counts, such as those based on slices in Xilinx FPGAs, can be greatly influenced by timing-driven placement and routing (PAR) algorithm as well as the overall resource utilization of the FPGA chip. Hence estimation error can be as high as 30%.

Table 1: BCJR decoder FPGA area estimation comparison

	Estimation	Actual	Difference
Slices	3,981	5,296	-24.8%
LUTs	7,550	8,905	-15.2%
Block RAM	82	82	0%

The same Simulink design description also provides the cycle-accurate and bit-true software simulation on a PC. Using a dual 1.8 GHz Intel Pentium 4 Xeon processor server, the software simulation runs at an average rate of 70 seconds for every 1000 input vectors. Extrapolating to 16 billion input vectors, the whole simulation would take 35.5 years, which is approximately 10^6 times slower than the FPGA emulation time. If the hardware cycle-accurate requirement is removed, software simulation can be improved by a factor 10 to 100, depending on the level of abstractions maintained. In general, from this experience we can estimate that the FPGA emulation provides 10^4 to 10^6 times speed-up over equivalent fix-point software simulations.

5 Conclusion

So far, four complete BEE emulation systems have been built, tested, and are operating as an integral part of the design infrastructure for a number of novel communication systems designs. These systems include designs for multi-channel-multi-antenna and ultra-wide band radios. A 2.4 GHz, single channel, radio system has been successfully demonstrated using the BEE emulator in real-time emulation mode, connected with its analog radio front-end, running at 32 MHz system clock rate. Currently a four antenna radio system is under development using the BEE. Other applications of BEE include complex iterative decoder design, a video encoder system, and other DSP related component designs.

Rapid design iteration, high-level performance estimation, and the real-time hardware emulation capability are the three most compelling features provided by the BEE emulation environment. Therefore, extensive design space exploration is not only feasible, but also effective. Drastically different algorithms and system architectures can be rapidly designed and implemented on the hardware for in-system verification. The process takes a matter of hours and days, rather than the weeks and months associated with traditional physical prototyping. Through direct-mapping, from the high-level description to the final hardware implementation, users can precisely control the outcome of the implementation while exploring trade-offs

on the top-level design abstraction. Furthermore, after the design has been verified using the BEE emulator, an ASIC implementation can be automatically generated with cycle and bit-level functional equivalency.

Currently a new Simulink-to-implementation compiler is under development to enable even higher abstraction level design entry as well as to improve the run-time performance and capability of the compilation process. Future research also includes improvements on the efficiency and accuracy of the high-level performance estimations, design scalability, and ASIC back-end automation for new fabrication technologies.

References

- [1] Chen Chang, Kimmo Kuusilinna, Brian Richards, Robert W. Brodersen, "Implementation of BEE: a Real-time Large-scale Hardware Emulation Engine," *Proc. FPGA 2003*, Feb. 2003
- [2] K. Kuusilinna, C. Chang, M.J. Ammer, B. Richards, R.W. Brodersen, "Designing BEE: a Hardware Emulation Engine for Signal Processing in Low-Power Wireless Applications," *EURASIP Journal on Applied Signal Processing, special issue on Rapid Prototyping of DSP Systems*, 2003
- [3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, *Low Power CMOS Digital Design*, Kluwer, 1995.
- [4] <http://www.synopsys.com/products/datapath/datapath.html>
- [5] L. Bahl, J. Cocke, F. Jelinek, and R. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Information Theory*, vol. 20, no. 2, pp. 284-287, Mar 1974.
- [6] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magnetics*, vol. 37, no. 2, pp. 748-755, Mar. 2001.
- [7] B Shackleford, M.Tanaka, R.J. Carter, G. Snide, "FPGA Implementation of Neighborhood-of-Four Cellular Automata Random Number Generators," *Proc. FPGA 2002*, pp. 106-112, Feb. 2002.