A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device

Kyrre Glette and Jim Torresen

Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
E-mail: {kyrrehg,jimtoer}@ifi.uio.no

Abstract. There have been introduced a number of systems with evolvable hardware on a single chip. To overcome the lack of flexibility in these systems, we propose a single-chip evolutionary system with the evolutionary algorithm implemented in software on a built-in processor. This architecture is implemented in a Xilinx Virtex-II Pro FPGA with an embedded PowerPC processor. This allows for a rapid processing of the time consuming parts in hardware and leaving other parts to more easily modifiable software. This platform will be beneficial for future work regarding both cost and compactness. Experiments have been performed on the physical device with software running in parallel with fitness computation in digital logic. The results show that the system uses only twice as much time when compared to a PC running at 10 times faster clock speed.

1 Introduction

Evolution time is critical for online evolvable systems. Further, often the compactness and cost of the system would be important. Thus, integrating as much as possible of a system on a single chip would be important.

There have been undertaken some implementations earlier. Kajitani et al have introduced several LSI (Large-Scale Integrated Circuits) devices with evolution undertaken in hardware [2,3]. The benefit of such an approach is the evolution speed but the problem is lack of flexibility. This would be important since there are often many degrees of freedom when evolving hardware systems. On-chip evolution using a prototype of the VLSI (Very Large-Scale Integration) POEtic chip has also been reported [7]. A robot controller and logic functions (3-input multiplexer and full adder) were evolved. The architecture contains an on-chip custom 32-bit processor, and a bio-inspired array of building blocks. This chip is specialized for the implementation of bio-inspired mechanisms.

In this paper, we demonstrate how a commercial FPGA (Field Programmable Gate Array) can provide a platform for System-On-Chip evolution. This is by integrating the evolution running as software on a processor with the target evolvable hardware implemented in reconfigurable logic. This allows for fast fitness computation – normally the most time consuming part of evolution, by measuring fitness in hardware communicating with a processor within the same hardware chip.

Implementing complete evolution in an FPGA has been proposed by Tufte and Haddow in [12]. The evolving design is implemented in the same device as the evolutionary algorithm. A similar approach is proposed by Perkins et al in [6]. Significant speedup is achieved for non-linear filtering compared to conventional processing. Several custom accelerators in FPGA for solving a protein folding problem have been introduced by Shackleford et al [10].

Running complete evolution within a Virtex XC2V3000 FPGA has been reported by Sekanina [9]. In this work the evolution (mutation only) is implemented in reconfigurable logic. Correctly working 3×3 and 3×4 bit multipliers were evolved.

In the work presented in this paper, a XC2VP7 Virtex-II Pro FPGA has been applied. It consists of reconfigurable logic, a PowerPC 405 hard-core processor block, on-chip RAM and high speed serial links for external interfaces.

There has been developed one other system for the Virtex-II Pro device [8]. This work is based on designing a co-processor for an analog neural network ASIC. This is contrasted to our work, where we focus on evolution of digital circuits and an evolutionary system in a single device. Further, in our system, all parts of the evolution (except the fitness evaluation, which is implemented in digital logic) are undertaken in software, providing a flexible system for later modifications. This is slower than implementing the evolution in dedicated hardware, but it is expected that the fitness evaluation time will still be the most time consuming part. This balanced software-hardware approach will allow for a low implementation effort while still being able to have a single-chip design, suitable for embedded real-world applications.

Another motivation, for having on-chip evolution and fitness computation in a single unit, is that it allows for scalable systems. By connecting a number of such units into a grid, they can perform concurrent evolution. Thus, this will be a very scalable architecture as well as flexible. In addition to the flexibility provided by software, the hardware would also be relatively easy to modify.

To demonstrate the achievable performance, experiments will be based on evolving small multiplier circuits. This will be to document the speed of evolution rather than evolution of very large and complex circuits which time has not allowed us to do so far. A number of papers have earlier contained work on evolving multipliers off-chip and extrinsically [4,5,13,11].

The next section introduces the applied FPGA followed by our architecture in Section 3. Results from the implementation are given in Section 4. Finally, Section 5 concludes the paper.

2 The Virtex-II Pro FPGA

The design is synthesized for a Xilinx Virtex-II Pro (XC2VP7-FG456-7) – see Fig. 1. This device contains 11,088 logic cells, 792 Kbit dual-port SRAM - named Block SelectRAM (BRAM), and one PowerPC 405 (PPC) embedded processor. The maximum processor speed is 300MHz.

The FPGA is situated on a Memec Design Virtex-II Pro development board. The board also contains two Xilinx XC18V04 configuration EEPROMS, 32MB SDRAM, Rocket I/O ports, an RS-232 port, an LCD panel and other useful connectors.

3 Implementing Evolutionary Algorithm on FPGA

In this section, the implementation of the evolvable hardware system will be detailed.

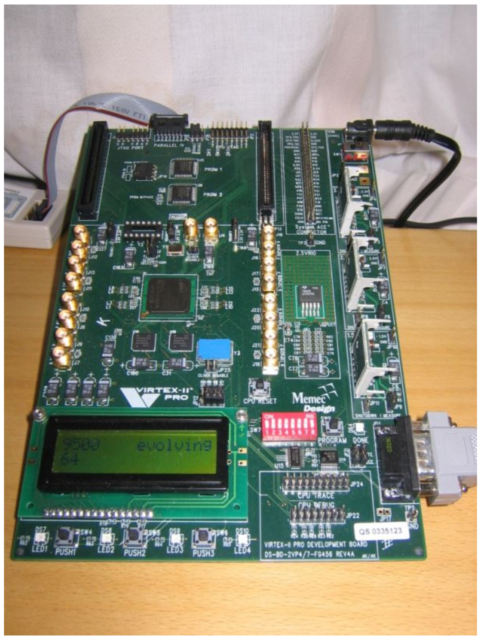
Fig. 1. The prototype board with the Virtex-II Pro FPGA.

3.1 System Overview

The on-chip system is built using the Xilinx Embedded Development Kit (EDK) [15]. EDK is a collection of Intellectual Property (IP) cores and tools for building embedded systems on FPGAs. The hardware and software parts of the system can be specified parametrically through various configuration files, and net lists and libraries are automatically generated.

The architecture consists of a set of modules interconnected with buses. The bus system is a part of IBM's CoreConnect architecture [16]. Two main buses are used to connect the on-chip peripherals – the Processor Local Bus (PLB) and the On-chip Peripheral Bus (OPB), as seen in Fig. 2. The PLB is a high-performance 64-bit datapath bus, while the OPB is a 32-bit wide bus designed for peripherals with lower requirements. These buses can be run in different clock domains, and they are interconnected with the PLB to OPB bridge.

The communication intensive modules are connected to the PLB: The PPC CPU and 64KB BRAM for program instructions. The PPC processor is connected as a bus master to the PLB. In addition to the PLB interface, there are two On-Chip Memory (OCM) interfaces in the PPC [14]. These are used as dedicated interfaces between the FPGA BRAM and the PPC core. One OCM is for the instruction-side memory space and the other is for the data-side memory space. These interfaces are usually used for accessing instruction and data caches, built from BRAM (there is no cache inside the PPC block). The advantage of these OCM interfaces over the PLB interface is that no bus arbitration

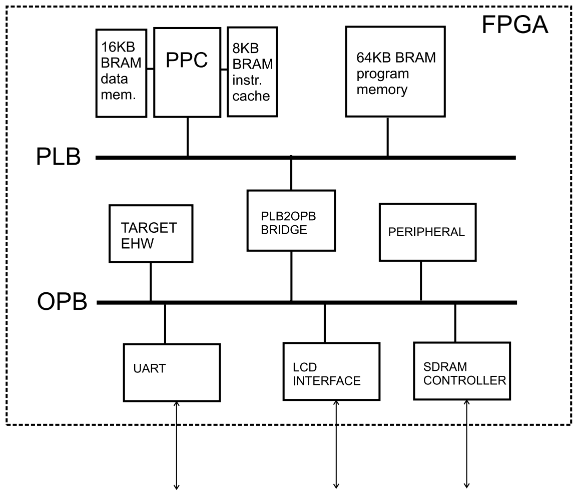

Fig. 2. Hardware architecture including our target EHW.

is necessary for memory access, and the instruction and data accesses do not have to share the same interfaces. In our design, 8KB of BRAM is connected to the instruction side OCM interface, used as an instruction cache. 16KB is connected to the data side OCM interface. This memory is used as storage for all program data, ie. no access to the PLB is needed at all. This increases program execution speed. In our case, a $2\times$ program execution speed increase was obtained by introducing instruction caching, and another $3\times$ increase was obtained by accessing all program data through the data side interface.

The target evolvable hardware is at the moment connected to the OPB. This will be detailed in section 3.3. Various on-chip peripherals are also connected to the OPB, including a UART for RS-232 serial communications, an LCD interface and a LED interface. An SDRAM controller can also be connected to the OPB should more memory be needed.

3.2 Implementing a Genetic Algorithm on the PPC

A Genetic Algorithm (GA) was implemented to run on the PPC. The program was written in C and compiled and linked using the PPC405 version of the GNU GCC compiler tools. Some system limitations had to be taken into consideration when implementing the GA on the embedded PPC system.

Firstly, program memory is limited. A maximum of 64KB of BRAM was allowed for the executable size. Although there exists 32MB of SDRAM on the development board, it was decided to only use BRAM internal to the FPGA. The BRAM is faster, and it allows for the program to be loaded directly from the bitstream that configures the FPGA. A program which is too large to fit into the BRAM would have to be loaded into SDRAM using a boot loader from external nonvolatile memory at startup (although during development it is possible to initialize SDRAM through the JTAG interface). SDRAM could still be used for data storage, but this would be a rather slow solution, at least if no data caching is used.

Secondly, there is no floating point support on the PPC405. Floating point operations used in C programs have to be emulated, unless a floating point co-processor is available. Emulating floating point is not speed efficient, and it increases the executable size.

The limitations on the executable size led us to use of C instead of C++, and to minimize the use of standard library functions. Speed considerations made us avoid using floating point in time-critical program parts. Ideally, only fixed point or integer solutions should be employed in order to reduce the executable size.

The combination of C-only programming, restricted use of library functions and floating point operations, makes the implementation slightly more challenging and time consuming than it would have been on a PC. However, the degree of program flexibility and the speed of algorithm implementation is still very high compared to assembly programming or custom hardware solutions. Having the above-mentioned limitations in mind, the program was developed mostly using Microsoft's Visual Studio, and it can be run on both the PC and the FPGA platform. Only a few code paths had to be written specifically for the PPC. The PC version of the program is equally fast as if it would have been developed for PC only.

The GA implemented for this experiment follows the Simple GA style, given by Goldberg [1]. Fitness scaling has been implemented, including linear scaling. A fitness-proportionate selection scheme is implemented through the use of a roulette wheel mechanism. The individuals are sorted with the qsort algorithm. For mutation, instead of having one probability of mutation for every bit in the genome, a quicker solution has been adopted. The number of mutations, *n*, for the whole genome is calculated by a random lookup in a 10-position array. Then, *n* random places are mutated (bit-flipped) in the genome. This is more efficient than performing a check for every bit if a mutation should occur or not.

3.3 Target Evolvable Hardware

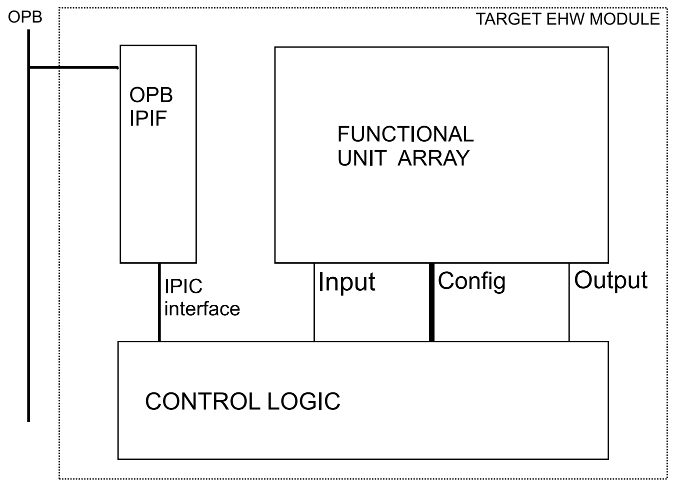

Fig. 3. The architecture of the target EHW system.

The target EHW is implemented as an OPB slave peripheral module – see Fig. 3. Interfacing with the OPB bus has been simplified by the use of a Xilinx IP Interface core (IPIF). This provides a simpler interface standard, the Xilinx IPIC, for the user module. IPIF cores exist for both OPB and PLB buses, so an adaptation of the target EHW to the PLB should be a feasible task.

Control and configuration of this module are undertaken through register write operations. Genome values are written to registers which are again connected to the configuration inputs of each functional unit. Registers are also provided for feeding the EHW with inputs and for storing the outputs.

As an example application, a configurable functional unit array has been implemented – see Fig. 4. Each subsystem evolved consists of a fixed-size array of functional units. The array consists of *n* unit layers from input to output.

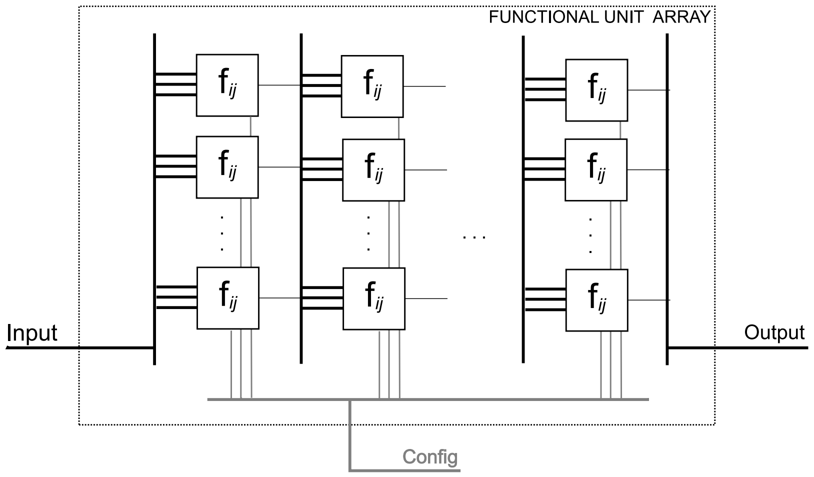

Fig. 4. The architecture of the functional unit array subsystem.

Each unit's three inputs in layer *l* are connected to the outputs of three units in layer $l-1$. Each of the input signals can be inverted. Each unit can have one of four functions: *BUF*, *MUX*, *AND*, or *XOR*. The function of each unit and its three inputs are configurable and determined by evolution. The encoding of each functional unit in the genome string is as follows, in the case of 4 different functions for each unit and 8 units in each layer:

Function (2 bit)	Input 1 (4 bit)	Input 2 (4 bit)	Input 3 (4 bit)

Of the 4 bits for each input, one bit is toggling an inversion of the input signal, while the 3 others code for which output from the previous layer to use. For our array consisting of 6 layers with 8 units, the genome string length becomes $(2+(3\times 4))\times 6\times 8 = 672$ bit long. The array is constructed in a pipelined fashion, that is, registers are connected to the outputs of each layer. Currently, this is not exploited for fitness evaluation. Only one training vector is evaluated at a time.

3.4 GA Parameters and Fitness Function

For the evolution, a population size of 20 is used. Elitism is used, thus, the best individuals from each generation are carried over to the next generation. The (single point) crossover rate is 0.5, thus the cloning rate is 0.5. A roulette wheel selection scheme is applied, and linear scaling is used. The mutation rate is expressed as a probability for a certain number, *n*, of mutations on each genome. The probabilities are as follows:

n	0	1	2	3
$p(n)$	$\frac{1}{10}$	$\frac{6}{10}$	$\frac{2}{10}$	$\frac{1}{10}$

The fitness function is computed in the following way:

$$F = \sum_{\text{vec}} \sum_{\text{outp}} x \qquad \text{where } x = \begin{cases} 0 \text{ if } y \neq d \\ 1 \text{ if } y = d \end{cases} \tag{1}$$

For each output the computed output y is compared to the target d. If these are equal then 1 is added to the fitness function. The function sum these values for the assigned outputs (outp) for the assigned truth table vectors (vec).

4 Results

This section presents and discusses the results of our implementation and experiments.

4.1 Device Utilization and Clock Speed

Resource	Used	Available	Percent
Slices	1025	4928	20
Slice Flip Flops	896	9856	9
4 input LUTs	1231	9856	12

Table 1. Device utilization for the EHW module

Table 1 shows the amount of logic used for our target EHW module containing an 8×6 functional unit array. A maximum of 20% of the FPGA's total resources are used. The total resource usage for the system, including bus structure and peripherals, is 43%. This indicates that there is more space for either more complex EHW modules, or a larger number of them. The FPGA used for these experiments is also relatively small compared to the larger Virtex-II Pro and Virtex-4 FX devices - devices with up to 142,000 logic cells are available at the time of writing.

The maximum clock frequencies currently attained are 200MHz for the PPC core, and 50MHz for the rest of the system, including both PLB and OPB modules. The maximum possible speed for the PPC is stated to be 300MHz, and 100MHz for the rest of the system. Since no analysis has currently been done to localize bottlenecks, it should be possible to increase these frequencies later.

4.2 Evolution Speed

Evolution runs were conducted on our on-chip evolution system and a Pentium 4 (P4) workstation for speed comparisons. The P4 workstation has a clock frequency of 2GHz. For the speed test, 10,000 generations of 20 individuals were evolved. The fitness evaluation was for a 2×2 bit multiplier, thus 16 input/output vectors were used.

Raw GA execution speed The execution time of only the GA operation without fitness evaluation was measured. The results are shown in the first row of table 2. As can be seen, the P4 system outperforms the on-chip system. This was expected, as the P4 processor is running at a much higher clock frequency and operates with a more efficient memory interface and caches. However, although the clock frequency of the P4 is 10 times greater than on the PPC, the evolution speed is only around 6.4 times greater. This may be explained by processor architecture factors, such as the high number of pipeline stages on the P4, in effect giving a lower instructions per clock cyle count.

Configuration	PPC	P4
GA without fitness evaluation	8.3s	1.3s
GA with fitness evaluation	20.0s	8.6s
Fitness time in % of total	59	85

Table 2. Evolution speeds on PPC and P4 systems.

GA with fitness evaluation speed The execution time of the GA operation including fitness evaluation was then measured. The phenotype is evaluated in hardware on the on-chip system and simulated in software on the P4 for comparison.

The results are shown in the second row of table 2. Here, the P4 system is still faster than the on-chip system, but this time only by a factor of 2.3. The hardware evaluation is advantageous to the on-chip system, but it is still limited by time-consuming fitness evaluation administration in software. An estimate of the time for a complete hardware fitness evaluation is around 9.7s, but this number would increase if the number of training vectors becomes higher.

The on-chip system will be little affected by increased complexity in the phenotype structure, whereas the simulated fitness evaluation will be more time-consuming.

4.3 Circuits Evolved

Correct 2×2 bit multiplier circuits were evolved after an average of 5702 generations over 10 evolution runs. The same experiment was conducted as a verification on the PC platform, where the average was 5649. The different numbers can be explained by the different programs using different random number generators. Still, the results are rather similar, which indicates that the FPGA implementation works correctly.

4.4 Discussion

The system architecture should be analyzed more thoroughly in order to obtain higher clock frequencies. A higher bus speed would be beneficial for the configuration phase of the target EHW, especially as the genomes get large. This could be combined with moving the target EHW module to the PLB bus, in order to get a wider datapath. A way of bursting data could also be explored. A direct connection to the other side of the PPC's data BRAM would also be possible, since the BRAM is dual-ported.

To speed up fitness evaluation, certain software operations can be moved into hardware. For digital circuits with defined training vectors, like the multiplier circuits, it would be advantageous to make a system that feeds the EHW circuit with one training vector per clock cycle. As the functional unit array is pipelined, the number of cycles needed for one complete fitness evaluation of an individual would be roughly equal to the number of training vectors. Another option would be to increase the number of EHW units on the same chip. However, an increased degree of hardware specialization may come at the prize of reduced flexibility and, naturally, a higher implementation effort.

We have presented the result of our first initial experiments on the evolutionary platform. Our motivation for implementing evolution on the PPC is mainly that we would like to apply the platform for evolving systems for real-world applications in the future. A large part of the total computation time is used for the fitness computation even for evolving the small (2 bit) multiplier in a relatively small circuit above. Thus, we expect that for more complex problems with much data to measure fitness on, the amount of time used for evolution compared to fitness computation will be small. On the other hand, to experiment with several evolutionary and bio-inspired methods, flexibility would be important. This is obtainable with our platform with the processor containing the parts not critical on computation time. This would also be important with our goal of designing a scalable system with incremental evolution.

5 Conclusions

The paper has presented an approach for evolving digital circuits in a new technology – System-On-Chip by FPGA. The work is focused on demonstrating the potential of running evolution on an embedded processor in an FPGA. The first experiments – by measuring performance of the real device, are promising. As this technology progresses, it will probably be an interesting platform for cost effective evolution in embedded systems.

Acknowledgments

The research is funded by the Research Council of Norway through the project *Biological-Inspired Design of Systems for Complex Real-World Applications* (project no 160308/V30).

References

1. D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning.* Addison–Wesley, 1989.

2. I. Kajitani et al. A myoelectric controlled prosthetic hand with an evolvable hardware lsi chip. *Technology and Disability*, 15(2):129–143, 2003.
3. I. Kajitani, T. Hoshino, N. Kajihara, M. Iwata, and T. Higuchi. An evolvable hardware chip and its application as a multi-function prosthetic hand controller. In *Proc. of 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 182–187, 1999.
4. T. Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In J. Lohn et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 65–74. IEEE Computer Society, Silicon Valley, USA, July 2000.
5. J.F. Miller, D. Job, and V.K. Vassilev. Principles in the evolutionary design of digital circuits – Part I. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.
6. S. Perkins, P. Porter, and N. Harvey. Self-contained spatially-structured genetic algorithm for signal processing. In J. Miller et al., editors, *Evolvable Systems: From Biology to Hardware. Third International Conference, ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pages 165–174. Springer-Verlag, 2000.
7. D. Roggen, Y. Thoma, and E. Sanchez. An evolving and developing cellular electronic circuit. In J. Pollack, M. Bedau, P. Husbands, T. Ikegami, and R. A. Watson, editors, *ALife9: Proceedings of the Ninth International Conference on Artificial Life*, pages 33–38, Boston, MA, 2004. MIT Press.
8. T. Schmitz et al. Speeding up hardware evolution: A coprocessor for evolutionary algorithms. In P. Hadddow A. Tyrrel and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth International Conference, ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 274–285. Springer-Verlag, 2003.
9. L. Sekanina and S. Friedl. On routine implementation of virtual evolvable devices using combo6. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 63–70. IEEE, 2004.
10. B. Shackleford et al. A high-performance, pipelined, FPGA-based genetic algorithm machine. *Journal of Genetic Programming and Evolvable Machines*, 2(1):33–60, 2001.
11. J. Torresen. Evolving multiplier circuits by training set and training vector partitioning. In P. Hadddow A. Tyrrel and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Fifth International Conference, ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 228–237. Springer-Verlag, 2003.
12. Gunnar Tufte and Pauline C. Haddow. Prototyping a ga pipeline for complete hardware evolution. In *1st NASA / DoD Workshop on Evolvable Hardware (EH '99)*, pages 18–25, 1999.
13. D. Job V. Vassilev and J. Miller. Towards the automatic design of more efficient digital circuits. In J. Lohn et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 151–160. IEEE Computer Society, Silicon Valley, USA, July 2000.
14. Xilinx Inc. *PowerPC 405 Processor Block Reference Guide*, August 2004.
15. Xilinx Inc. *Embedded System Tools Reference Manual*, February 2005.
16. Xilinx Inc. *Processor IP Reference Guide*, February 2005.