

# MATLAB/Simulink Based Hardware/Software Co-Simulation for Designing Using FPGA Configured Soft Processors

Jingzhao Ou and Viktor K. Prasanna  
Department of Electrical Engineering, University of Southern California  
Los Angeles, California, 90089-2560 USA  
Email: {ouj, prasanna}@usc.edu

## Abstract

*FPGA configured soft processors are an attractive choice for implementing many embedded systems. For application development using these soft processors, the users can execute portions of the applications as software programs and the other portions as customized hardware implementations. Being able to rapidly simulate various partitions of the applications on hardware and software is crucial to efficiently execute them on soft processors because (a) there are many possible configurations of soft processors, and (b) low-level simulation techniques are too time consuming for evaluating these different partitioning and configuration possibilities. While state-of-the-art design tools rely on low-level simulation and are unable to deliver such a fast simulation speed, we propose a high-level cycle-accurate hardware/software co-simulation environment based on MATLAB/Simulink for application development using soft processors. By utilizing the high-level cycle-accurate abstractions of the low-level hardware implementations and the arithmetic simulation capability provided by MATLAB/Simulink, our tool considerably accelerates the time for cycle-accurate functional simulation of both hardware and software portions of a given application running on soft processors. To illustrate our approach, we develop a CORDIC division application and a matrix multiplication application on a commercial soft processor. Up to 19.4x improvement in simulation time is achieved using our co-simulation environment compared with that of low-level simulation for various partitions of these applications and for various configurations of the soft processor.*

## I. Introduction

Integrated with multi-million gate configurable logic and various heterogeneous hardware components (such as embedded multipliers and memory blocks), FPGAs have

This work is supported by United States National Science Foundation (NSF) under award No. CCR-0311823.

become an attractive choice for implementing many embedded systems. Soft processors, which are RISC processors realized using configurable resources available on FPGA devices, have become popular. Examples of such soft processors include Nios from Altera [2] and MicroBlaze and PicoBlaze from Xilinx [13].

Designing using soft processors offers several advantages. One advantage is that they provide new design trade-offs by time sharing the limited hardware resources available on FPGA devices. There are computations that are inherently more suitable for software implementations on processors than the corresponding (parallel) hardware implementations. For example, some applications have tightly coupled data dependency among computation steps and do not benefit from parallel execution. Many recursive algorithms (e.g. Levinson Durbin recursion) and communication protocols (e.g. UART and the GSM converter discussed in [7]) fall into this category. Their software implementations are more compact and require much smaller amount of resources than their customized parallel implementations. Having a compact design that can be fit into a smaller device can effectively reduce quiescent energy dissipation [12]. Most importantly, soft processors are “configurable” by allowing the customization of the instruction set and/or the attachment on-chip customized hardware implementations in order to speed up the computations that can benefit from parallel implementations (e.g. FFT and RGB conversion, etc.). The Nios processor allows users to customize up to five instructions. The MicroBlaze processor supports various dedicated interfaces and bus protocols for attaching user hardware designs to it.

Since soft processors are “configurable” and highly extensible, they can be customized by adding dedicated instructions and/or attaching hardware peripherals to them and be optimized for the applications running on them. For applications running on soft processors, portions of them are executed as software instructions while the other portions are executed using customized hardware peripherals. As is discussed in Section II, while state-of-the-art design tools follow the traditional design flow and

are based on register transfer/gate level simulation for functional verification, they are unsuitable for application development using soft processors. This is because two reasons. One reason is that low-level simulation based on register transfer/gate level implementations is too time consuming for evaluating the correctness and performance of configurations of the processors and various possible partitions of the target applications. Especially, low-level simulation is unsuitable for simulating the execution of software programs. For the design example shown in Section IV-B, low-level simulation using ModelSim [8] takes more than 25 minutes to simulate 1.5 msec execution time of a software program running on the MicroBlaze processor. Another reason is that soft processors provide a potentially large design space for implementing applications on them. In addition to the various hardware/software partitions of the applications, there are many possible realizations of the customized hardware peripherals based on the architectures, hardware bindings, etc. used. Also, customized hardware peripherals can communicate with soft processors through various dedicated interfaces and bus protocols. Low-level simulation can become overwhelming for exploring all the design trade-offs.

Therefore, a high-level cycle-accurate hardware/software co-simulation environment which can rapidly simulate the software programs running on the processor, the customized hardware peripherals and the interactions between them is crucial for efficient application development using these soft processors. By “high-level”, we denote that only the arithmetic aspects of the low-level implementations are captured by the simulation process. For example, low-level implementations of multiplication on Xilinx Virtex-II FPGAs can be realized using either slice-based multipliers and embedded multipliers while the high-level simulation only captures the multiplication arithmetic aspect of these multipliers. Also, the high-level simulation of the communication interface only captures the arithmetic aspects of the communication protocols regardless whether the data buffering on the bus interfaces is realized using registers, slices or embedded memory blocks. By “high-level cycle-accurate”, we denote that during each of the simulated clock cycles, the functional behavior of the system predicted by the high-level cycle-accurate simulation environment should match the functional behavior of the corresponding low-level implementations. For example, when simulating the soft processor, the high-level cycle-accurate simulation should respect the fact that the multiplication instruction requires three clock cycles to complete. Also, the simulation should also respect the delay in number of clock cycles caused by the customized hardware peripherals.

As the major contribution of this paper, we demonstrate that a *high-level cycle-accurate co-simulation environment which does not involve the low-level register transfer*

*level/gate level implementations is possible and can greatly accelerate the co-simulation process for soft processors.*

We have developed such a co-simulation environment by integrating a cycle-accurate instruction simulator for simulating software programs running on soft processor and a high-level FPGA design tool. Taking the development of two example applications shown in Section IV as an example, our high-level co-simulation environment is shown to be capable of speeding up the simulation time ranging from 5.6x to 19.4x and 11.0x on average compared with that of the low-level simulation tools.

The paper is organized as follows. Section II discusses related work. Section III describes our approach for building a MATLAB/Simulink based hardware/software co-simulation environment for application development using FPGA configured soft processors. Due to its wide availability, we focus on integrating *System Generator* with a cycle-accurate simulator for MicroBlaze. The development of two signal processing applications is provided in Section IV to demonstrate the effectiveness of our co-simulation approach. The designs identified using our design environment achieve up to 5.6 times performance improvements compared with other designs considered in our experiments. Finally, we conclude in Section V.

## II. Related Work

MATLAB/Simulink based design tools, such as *DSP Builder* [2] from Altera and *System Generator* [13] from Xilinx, are becoming popular for developing signal processing applications on FPGAs. In *System Generator*, a block set is provided through which the designer can get access to proprietary IP cores and HDL designs. Application designers assemble designs by dragging and dropping the blocks from the block set into their designs and connecting them via a GUI. One major advantage offered by these MATLAB/Simulink based design tools is that they provide high-level abstractions of the various low-level hardware implementations. Thus, the end users can make use of the powerful modeling environment offered by MATLAB/Simulink to perform arithmetic level simulation, which is much faster than the behavioral and architectural simulations in traditional FPGA design flows [5]. Besides, *System Generator* provides a resource estimator for rapid estimation of the hardware resources used by a design. However, there is no support for simulating the execution of software programs running on soft processor within these MATLAB/Simulink based design tools, which is desired by application development using these “configurable” soft processors.

Commercial integrated design environments (IDEs) are available for application development using hard/soft processors. This includes SOPC Builder from Altera [2] and Embedded Development Kit (EDK) from Xilinx [13].

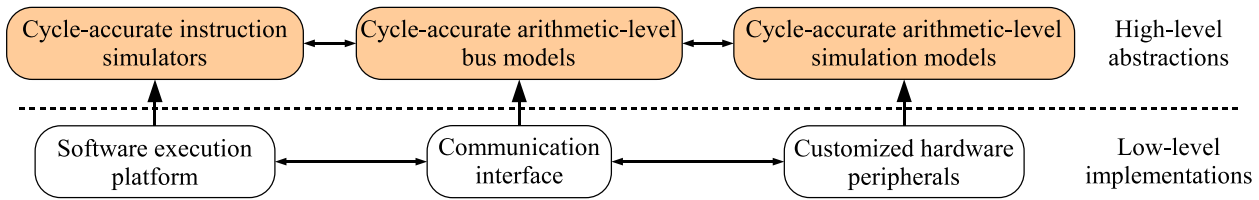


Fig. 1. Our approach for high-level hardware/software co-simulation

Taking EDK as an example, based on a simple configuration file, it instantiates the processor(s) as well as peripheral hardware components. Then, EDK generates the glue logic to connect these components. Software programs are automatically compiled and combined with the hardware configuration bitstream to initialize the on-chip memory blocks specified by the linker scripts. While these IDEs greatly simplify the designing tasks, end users still need to go through the time-consuming synthesis, place-and-route processes in order to simulate the designs and verify their correctness.

Xilinx provides a MicroBlaze stub block and an EDK export tool within *System Generator*. Users develop hardware designs that attach to the MicroBlaze stub block within MATLAB/Simulink. Then, using the EDK export tool, *System Generator* generates the corresponding low-level implementations that can be later imported into EDK as hardware peripherals of the processor. The MicroBlaze stub block helps to identify the I/O interface for communicating with the processor. However, it does not provide support for simulating the execution of software programs running on MicroBlaze as well as the bus transactions between the processor and its peripherals within MATLAB/Simulink. Simulation of the complete systems can only be performed after the low-level implementations are generated from EDK and using low-level simulation tools such as ModelSim [8].

*Seamless* is a tool from Mentor Graphics [8] for hardware/software co-verification. It contains processor models for simulating the execution of software programs on various processors. It also provides bus models for simulating the communication between hardware and software executions. However, *Seamless* only supports time-consuming low-level hardware simulation. The end users need to implement abstract C bus peripherals themselves using the bus access APIs before they can separate the software simulation and the hardware simulation.

The POLIS hardware/software co-design framework [1] can automatically generate low-level behavioral VHDL simulation models for simulating both hardware and software components based on the partitioning specifications provided by end users.

To summarize, the design tools discussed above are based on low-level simulation, which would prevent them

from efficiently exploring the various ways of executing applications on soft processors. Compared with these tools, we show in Section IV that our co-simulation environment provides a significantly faster simulation speed through high-level abstractions of the soft processors.

### III. Our Approach

Our approach for building a high-level MATLAB/Simulink based hardware/software co-simulation environment is illustrated in Figure 1. The low-level implementations of soft processors consist of three major components: *software execution platform* for executing software programs, *customized hardware peripherals* for parallel execution of some specific functionalities, and *communication interface* for exchanging data between the other two components. We create high-level abstractions for each of these three components within our co-simulation environments so as to speed up the simulation time for different configurations of the soft processors. Within our co-simulation environment, we have cycle-accurate instruction simulators for simulating the software execution platform, cycle-accurate arithmetic-level simulation models for simulating the status of the customized hardware peripherals, as well as cycle-accurate arithmetic-level bus models for simulating the communication interface. All these simulation models are integrated within MATLAB/Simulink modeling environment. By utilizing the arithmetic simulation capability provided by MATLAB/Simulink, we are able to perform high-level hardware/software co-simulation for applications running on soft processors.

In the following, Xilinx MicroBlaze is used to demonstrate the effectiveness of our co-simulation approach due to its wide availability. Our techniques can be applied to other soft processors as well. As inputs to our environment, software designs are developed as C programs while customized hardware peripherals are described within the MATLAB/Simulink using *System Generator*. A Simulink block for MicroBlaze is created to provide various I/O ports for the customized hardware peripherals to communicate with the processor. The end users can perform cycle-accurate arithmetic level simulation within our co-simulation environment to verify the correctness of their designs and to evaluate the various possibilities for exe-

cutting the applications on the processor. Once the final design is identified, the low-level implementation can be generated automatically using *System Generator* and EDK.

## A. Software Architecture of Co-simulation Environment

The software architecture of our MATLAB/Simulink based hardware/software co-simulation environment is illustrated in Figure 2, which is composed of three components. In correspondence to the configurations of the MicroBlaze processor shown in Figure 2, there is one component for simulating the software execution platform and one component for simulating the customized hardware peripherals. There is one additional component, the *MicroBlaze Simulink block*, which is responsible for simulating the communication interface between the processor and the customized hardware peripherals and for exchanging data between the other two components.

- *Simulation of software execution platform:* The input C programs are compiled using *mb-gcc*, the GNU C compiler for MicroBlaze, and translated into binary executable *.ELF* files. The *.ELF* files are then loaded by *mb-gdb*, the GNU debugger. Xilinx provides a cycle-accurate simulator for MicroBlaze. *mb-gdb* communicates with the simulator using TCP/IP protocol. In order for the MicroBlaze cycle accurate simulator to be able to simulate the execution of the software programs, the MicroBlaze processor and the two LMB (Local Memory Bus) interface controllers (for accessing instructions and data of the software programs stored at the BRAMs) are required to operate at the same frequency. Under this configuration, a fixed latency of one clock cycle is guaranteed for the soft processor to access instructions and data through these two controllers.

- *Simulation of customized hardware peripherals:* The customized hardware peripherals are simulated using *System Generator*. There are *Gateway In* and *Gateway Out* Simulink blocks for separating the simulation of the *System Generator* hardware designs with that of other Simulink designs and for specifying the input/output ports of the hardware designs. Whenever there is data coming from the processor, simulation of these hardware designs is carried out within the Simulink modeling environment. The output results is generated and is sent back to the processor for further processing.

- *Simulink block for MicroBlaze:* We create a MicroBlaze Simulink block which provides an interface for MATLAB/Simulink to get access to the status of the software programs running on the MicroBlaze processor. There are three major functions provided this block. First, this Simulink block communicate with *mb-gdb* to obtain the execution status of the software programs. *mb-gdb* is run within a bidirectional software pipe implemented using

TCL scripting language [11]. It accepts commands from the MicroBlaze Simulink block and interactively runs the software programs. It also changes the status of the registers of the MicroBlaze processor based on the results from the customized hardware designs. Second, the MicroBlaze Simulink block simulates the bus protocols for communications between MicroBlaze and the hardware designs described using *System Generator*. Various bus protocols, such as the IBM on-chip peripheral bus (OPB) and the Xilinx fast simplex link, are supported in our environment. Especially, we show in detail the simulation of communication through fast simplex links in Section III-B. The MicroBlaze Simulink block simulates the FSL FIFO and the hand-shaking protocol. Finally, the Simulink block for MicroBlaze sends the data from *mb-gdb* to the hardware designs and initiates the simulation within *System Generator*. The output data of the hardware designs is retrieved by the MicroBlaze Simulink block and is sent back to *mb-gdb* to update the status of the processor.

## B. Simulation of Communication Through Fast Simplex Links

Users can connect customized IP cores to the MicroBlaze processor through Fast Simplex Links (FSLs). FSLs are implemented as unidirectional FIFOs. The MicroBlaze processor includes a maximum of 16 FSLs, eight for data input and eight for data output. Both synchronous (blocking) and asynchronous (non-blocking) communication modes are supported by these FSLs. Blocking read or write will stall the MicroBlaze processor until the read or write can occur. In contrast, non-blocking read or write will not stall MicroBlaze even if the read or write is unable to complete.

The MicroBlaze Simulink block implements the FSL FIFO and the data input and output interfaces. Let “#” denote the number of the FSL input/output channel for accessing the MicroBlaze processor. “#” is an integer number between 0 and 7. Behavior of the following control signals of these FSLs are captured by the MicroBlaze processor. (When *In#\_write* is high and there is data to be written into the FSL, it stores the data presented at *In#\_write* into the internal data structure and raises the *Out#\_exists* flag bit to indicate the availability of the data. Likewise, when the FSL FIFO is full, it raises the *In#\_full* flag bit to prevent data coming in the FSL FIFO (e.g. in blocking communication mode).

Xilinx provides a set of C functions for communication through FSLs. After compilation, these C functions are translated to the corresponding assembly instructions. For example, C function *microblaze\_nbread\_datafsl(val,id)* for non-blocking reading of data from the *id*-th FSL channel is translated into assembly instruction *nget(#val#, rfsl#id)*.

When simulating the execution of the software programs, the Simulink block for MicroBlaze keeps track of

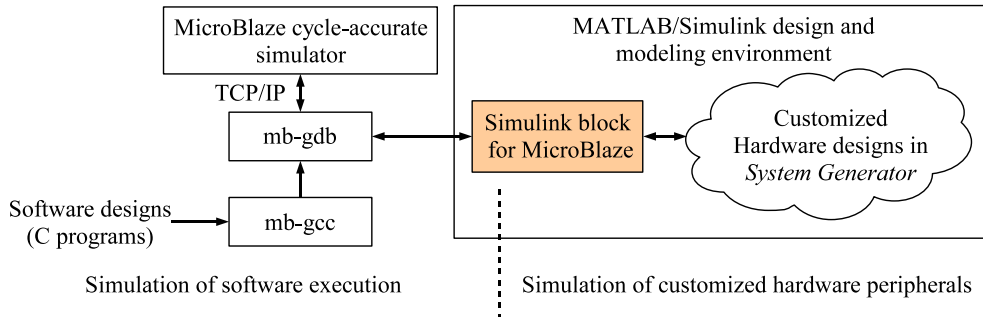


Fig. 2. Software architecture of our hardware/software co-simulation environment

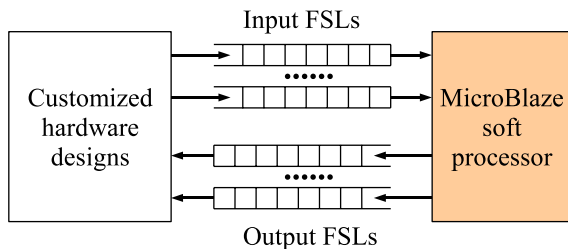


Fig. 3. Communication between MicroBlaze and customized hardware designs through Fast Simplex Links

the status of the processor by communicating with *mb-gdb*. When it encounters a request for writing data to the customized hardware designs through FSLs, it stalls the execution of software programs and tries to write the data to the FSL FIFO. In blocking communication mode, simulation of the software programs is stalled until the write operation succeeds. In this case, the processor gets stalled until *In#\_full* becomes low and the FSL FIFO is ready to accept more data. Otherwise, in non-blocking communication mode, the MicroBlaze Simulink block continues the execution of the software programs immediately regardless of the outcome of the write operation.

Similarly, the MicroBlaze Simulink block stalls the simulation of the software programs when it encounters a request for reading data from the customized hardware peripherals. In blocking communication mode, it waits until the *Out#\_exists* flag bit becomes low, which indicates that the results from the customized hardware peripherals are available at the FSL FIFO. Otherwise, in non-blocking communication mode, the MicroBlaze Simulink block resumes the execution of the software programs immediately regardless of the outcome of the read operation.

### C. Rapid Resource Estimation

Being able to rapidly obtain the hardware resource occupied by the soft processor under different configurations is important for identifying the most efficient partitioning

of the applications running on them. For Xilinx FPGAs, we focus on the number of slices, the number of BRAM memory blocks and embedded 18bit-by-18bit multipliers used by the processor.

There are four major sources of resource usage of the complete design: the MicroBlaze processor, the customized hardware peripherals, the communication interface, and the storage of the software programs. Resource usage of the MicroBlaze processor and the two LMB interface controllers is obtained from the Xilinx data sheet. Resource usage of the customized hardware designs is obtained using the resource estimator provided by *System Generator*. Since the software programs are stored in BRAMs, we obtain the size of the software program using the *mb-objdump* tool and then calculate the number of BRAMs required to store the software program based on its size.

## IV. Illustrative Examples

To demonstrate the effectiveness of our approach, we show in this section the designs of an adaptive CORDIC processor for division and a block matrix multiplication algorithm using MicroBlaze. These designs can be used in applications such as adaptive beamforming, where they are used to update the weight coefficients of the filters in accordance with the changes of the communication environment. Designing using soft processors have the advantages of easy deployment and the ability to handle different problem sizes. Besides, weight coefficient of the adaptive filters are updated from every tens to every millions data samples. Designs that provide different time and resource usage trade-offs are highly desired.

For the experiments discussed in the paper, the MicroBlaze processor is configured on Xilinx Virtex-II Pro devices. As is required by the MicroBlaze cycle-accurate simulator, the operating frequencies of the processor, the two LMB interface controllers and the customized hardware peripherals are set at 50 MHz. We use EDK 6.2.02 for describing the software execution platform and for compiling the software programs. *System Generator* 6.2 is

used for describing the customized hardware peripherals and generating low-level implementations. Finally, ISE 6.2.02 [13] is used for synthesis and implementation of the complete designs. In addition, we have verified the functional correctness of the designs on an ML300 Virtex-II Pro prototyping board [13].

### A. Adaptive CORDIC Processor for Division

• *Algorithm:* The CORDIC (COordinate Rotation DIgital Computer) iterative algorithm for dividing  $b$  by  $a$  [3] is described as follows. Initially, we set  $X_{-1} = a$ ,  $Y_{-1} = b$  and  $Z_{-1} = 0$ . During each iteration  $i$  ( $i = 0, 1, \dots$ ), the following computation is performed.

$$\begin{cases} X_{i+1} = X_i \\ Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i} \\ Z_{i+1} = Z_i - d_i \cdot 2^{-i} \end{cases} \quad (1)$$

where,  $d_i = +1$  if  $Y_i < 0$  and  $-1$  otherwise. After a certain amount of iterations, we have  $Z_{i+1} \approx -b/a$ . Many telecommunication systems have a wide dynamic data range, it is desired that the number of iterations can be dynamically adapted to the communication environment. Also, for some CORDIC algorithms, the effective precision of the output data cannot be computed analytically. For example, the effective bit precision of hyperbolic CORDIC algorithms depends on the angular value  $Z_i$  during iteration  $i$  and needs to be determined dynamically.

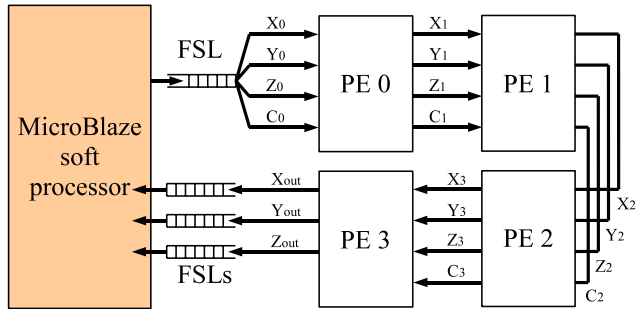


Fig. 4. CORDIC processor for division with  $P = 4$

• *Implementation:* The architecture of our CORDIC processor for division based on MicroBlaze is shown in Figure 4. We consider 16-bit data precision. The customized hardware peripheral is configured with  $P$  processor elements (PEs). Each PE performs one iteration of computation described in Equation 1. All the PEs form a linear pipeline and is fully pipelined between them. We consider 32-bit data precision in our designs. Since software programs are executed in a serial manner in the processor, only one FSL is used for sending the data from MicroBlaze to the customized hardware peripheral. The software program controls the number of iterations for each

set of data based on the specific application requirement. To support more than 4 iterations for the configuration shown in Figure 4, the software program sends  $X_{out}$ ,  $Y_{out}$  and  $Z_{out}$  generated by  $PE_4$  back to  $PE_1$  for further processing until the desired number of iterations is reached.

In order to allow for such repeatedly processing of the data through the linear pipeline of PEs, we rewrite the representation of  $Z_{i+1}$  as follows.

$$\begin{aligned} Y_{i+1} &= Y_i + d_i \cdot X_i \cdot C_i \\ Z_{i+1} &= Z_i - d_i \cdot C_i \\ C_{i+1} &= C_i \cdot 2^{-1} \end{aligned} \quad (2)$$

where,  $C_0 = 1$  initially. For the hardware design shown in Figure 4,  $C_0$  is decided by the software program based on the number of passes through the linear pipeline.  $C_0$  is sent out from the MicroBlaze processor to the FSL as a control word. That is, when there is data available in the corresponding FSL FIFO and  $Out\#\_control$  is high,  $PE_0$  updates its own copies of  $C_0$  and then populates it to the other PEs along the linear pipeline. For the other PEs,  $C_i$  is updated as  $C_i = C_{i-1} \cdot 2^{-1}$ ,  $i = 1, 2, \dots, P-1$ , and is obtained by right shifting  $C_{i-1}$  from the previous PE.

When performing division on a large set of data, the input data is divided into several sets and is processed set by set. Each set of input data is fed into the customized hardware peripheral continuously. The output data of the hardware peripheral is stored at the FIFOs of the data output FSLs and is then sent back to the processor. The size of each set of data is selected carefully so that the results generated by the set of data would not overflow the FIFOs of the data output FSLs.

The linear pipeline of PEs is described using *System Generator*.  $P$ , which is the number of PEs, is parameterized in our designs using the *PyGen* developed by us [10]. For larger  $P$ , the configuration of the MicroBlaze processor consumes more hardware resources. However, it speeds up the execution of the complete application.

• *Performance:* The time performance of various configurations of the CORDIC processor for division is shown in Figure 5 while its resource usage is shown in Table I. The resource usage estimated using our design tool is calculated as shown in Section III-C. The actual resource usage is obtained from the place-and-route reports (*.par* files) generated by ISE. For CORDIC algorithms with 24 iterations, attaching a customized linear pipeline of 4 PEs to the soft processor results in a 5.6 times improvement in time performance compared with “pure” software implementation while it requires 280 (30%) more slices.

• *Simulation speed-ups:* The simulation time of the CORDIC processor for division using our high-level co-simulation environment is shown Table I. For comparison purpose, we also show the simulation time of the low-level behavioral simulation using ModelSim. For the ModelSim

TABLE I. Resource usage of the CORDIC based division and the block matrix multiplication applications as well as the time for cycle-accurate functional simulation using different simulators

Designs	Estimated/actual resource usage			Simulation time	
	Slices	BRAMs	Multippliers	Our environment	ModelSim (Behavioral)
24 iteration CORDIC division with $P = 2$	729 / 721	1 / 1	3 / 3	6.3 sec	35.5 sec
24 iteration CORDIC division with $P = 4$	801 / 793	1 / 1	3 / 3	3.1 sec	34.0 sec
24 iteration CORDIC division with $P = 6$	873 / 865	1 / 1	3 / 3	2.2 sec	33.5 sec
24 iteration CORDIC division with $P = 8$	975 / 937	1 / 1	3 / 3	1.7 sec	33.0 sec
$12 \times 12$ matrix multiplication with $2 \times 2$ blocks	851 / 713	1 / 1	5 / 5	99.4 sec	1501 sec
$12 \times 12$ matrix multiplication with $4 \times 4$ blocks	1043 / 867	1 / 1	7 / 7	51.0 sec	678 sec

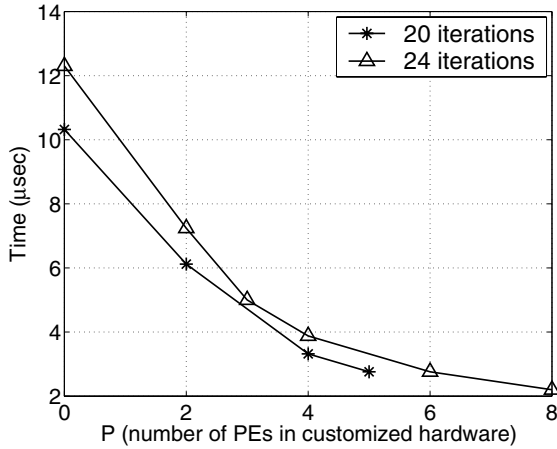


Fig. 5. Time performance of the CORDIC processor for division ( $P = 0$  denotes “pure” software implementations)

simulation, the time for generating the low-level implementations is not accounted for. We only consider the time for compiling the VHDL simulation models and the actual simulation. Compared with the low-level simulation in ModelSim, our simulation environment achieves speed-ups in simulation time ranging from 5.6x to 19.4x and 12.8x on average for the four designs shown in Table I.

- *Analysis of simulation performance:* The simulation speeds of the instruction simulator for simulating the software programs running MicroBlaze and MATLAB/Simulink for simulating the hardware peripherals. Compared with the low-level behavioral (functional) simulation using ModelSim, our co-simulation environment can potentially achieve simulation speed-ups from 5.5X to more than 1000X. There are two major factors that slow down the actual simulation of the CORDIC division application: a considerable portion of the computations are executed on the hardware peripherals and simulated within MATLAB/Simulink; there are frequent data exchanges between the software program and the hardware peripherals.

## B. Block Matrix Multiplication

- *Algorithm:* In our design of block matrix multiplication, we first decompose the original matrices into a number

TABLE II. Simulation speeds of the simulators for the CORDIC division application

	Instr. simulator	Simulink <sup>(1)</sup>	ModelSim <sup>(2)</sup>
Clk cycles / sec	>1000	254.0	46.2

Note: (1) Only simulate the hardware peripherals; (2) Low-level behavioral simulation. The time for generating the simulation models of the low-level implementations is not accounted for.

of smaller matrix blocks. Then, the multiplication of these smaller matrix blocks is performed within the customized hardware peripheral. The software program is responsible for controlling data to and from the customized hardware peripheral, combining the multiplication results of these matrix blocks, and generating the result matrix. As is shown in Equation 3, to multiply two  $4 \times 4$  matrices,  $A$  and  $B$ , we first decompose them into four  $2 \times 2$  matrix blocks respectively. The multiplication of these  $2 \times 2$  matrix blocks is performed within the customized hardware peripheral. To minimize the required data transmission between the processor and the hardware peripheral, the matrix blocks of matrix  $A$  are loaded into the hardware peripheral column by column so that each block of matrix  $B$  only needs to be loaded into the hardware peripheral once.

$$\begin{aligned}
 & A \cdot B \\
 &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \\
 &= \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & B_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & B_{21}B_{12} + A_{22}B_{22} \end{pmatrix}
 \end{aligned} \quad (3)$$

- *Implementation:* The architecture of our block matrix multiplication based on  $2 \times 2$  matrix blocks is shown in Figure 6. Similar to the design of the CORDIC processor, the data elements of matrix blocks from matrix  $B$  (e.g.  $b_{11}$ ,  $b_{21}$ ,  $b_{12}$  and  $b_{22}$  in Figure 6) are fed into the hardware peripheral as control words. That is, when data is available in the FSL FIFO and  $Out\#\_control$  is high, the hardware peripheral puts the input data into the corresponding registers. Thus, when the data elements of matrix blocks from matrix  $A$  come in as normal data words, the multiplication and accumulation are performed accordingly to generate the output results.

Similar to the previous section, the size of the matrix blocks  $N$  is parameterized in the designs of the customized

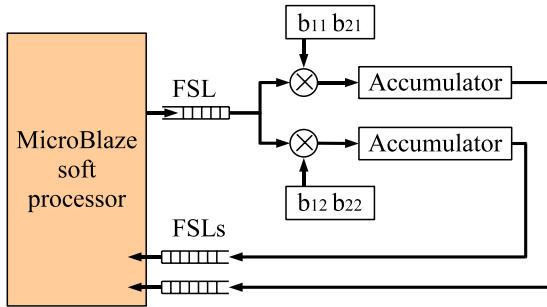


Fig. 6. Matrix multiplication with customized hardware peripheral for  $2 \times 2$  matrix block multiplication

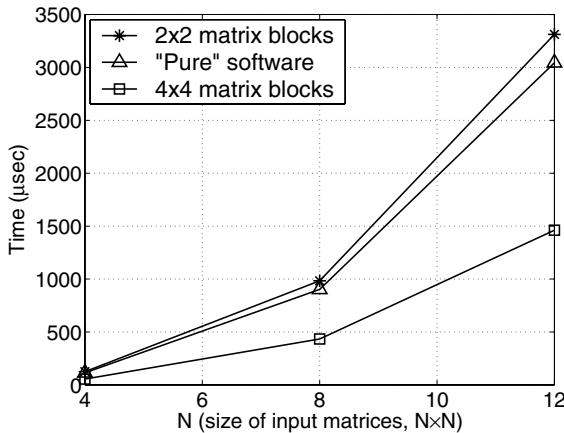


Fig. 7. Time performance of our design of block matrix multiplication

hardware peripheral. For larger  $N$ , the configuration of the MicroBlaze processor consumes more hardware resources while achieving a higher execution time.

- **Performance:** The time performance of various implementations of block matrix multiplication is shown in Figure 7 while their resource usage are shown in Table I. For multiplication of two  $12 \times 12$  matrices, attaching a customized hardware peripheral for performing  $4 \times 4$  matrix block multiplication to the soft processor results in a 2.2 times speed-up compared with “pure” software implementation while it requires 767 (17%) more slices.

Note that attaching a customized hardware peripheral for computing  $2 \times 2$  matrix blocks to the MicroBlaze processor results in performance penalties for the performance matrices considered (more specifically, 8.8% more execution time, 56 (8.6%) more slices and 2 (67%) more embedded multipliers) compared with the corresponding “pure” software implementations. This is due to the fact that the communication overhead for sending data to and back from the customized hardware peripheral is greater than the time saved by the parallel execution of multiplying the matrix blocks.

- **Simulation speed-up:** Similar to Section IV-A, we compare the time of cycle-accurate functional simulation in our co-simulation environment with that of low-level behavioral simulation in ModelSim. Speed-ups in simulation time of 13x and 15.1x are achieved for the two matrix multiplication designs shown in Table I.

## V. Conclusion

A MATLAB/Simulink based hardware/software co-simulation environment for application development using FPGA configured soft processors is presented in this paper. Design examples were provided to show the effectiveness of our co-simulation approach.

Energy performance is not addressed by our co-simulation environment while it has become a very important performance metric in the design of many embedded systems. One important extension of our work is to provide rapid energy estimation for application development using soft processors. We have developed an instruction-level energy estimation technique for computations on soft processors in [9]. We have also developed a domain-specific energy modeling technique for different parallel hardware designs using FPGAs in [10]. We are working on to integrate these two rapid energy estimation techniques into the co-simulation framework proposed in the paper.

## References

- [1] “A Framework for Hardware-Software Co-Design of Embedded Systems,” <http://www-cad.eecs.berkeley.edu/Ressep/Research/hsc/abstract.html>.
- [2] Altera, Inc., <http://www.altera.com>.
- [3] R. Andraka, “A Survey of CORDIC Algorithms for FPGAs,” ACM Inter. Symp. on Field Programmable Gate Arrays (FPGA), 1998.
- [4] S. Choi, J.-W. Jang, S. Mohanty, V. K. Prasanna, “Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures,” *Inter. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2002.
- [5] J. Hwang, B. Milne, N. Shirazi, J. Stroomer, “System Level Tools for DSP in FPGAs,” *Inter. Conf. on Field Programmable Logic and its applications (FPL)*, 2001.
- [6] MathWorks, Inc., <http://www.mathworks.com>.
- [7] “MicroBlaze Application Notes,” *Programmable World*, 2003.
- [8] Mentor Graphics, Inc., <http://www.mentor.com>.
- [9] J. Ou and V. K. Prasanna, “Rapid Energy Estimation of Computations on FPGA based Soft Processors,” *IEEE Inter. SoC Conference (SoCC)*, 2004.
- [10] J. Ou and V. K. Prasanna, “PyGen: A MATLAB/Simulink based Tool for Synthesizing Parameterized and Energy Efficient Designs Using FPGAs,” *IEEE Inter. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2004.
- [11] TCL Developer Exchange, <http://www.tcl.tk>.
- [12] T. Tuan and B. Lai, “Leakage Power Analysis of A 90nm FPGA,” *IEEE Custom Integrated Circuits Conference (CICC)*, 2003.
- [13] Xilinx Corporation, Inc., <http://www.xilinx.com>.