Hardware-Assisted Simulation and Evaluation of IP Cores

M. Sangeetha, M. Kumaran, J. Raja Paul Perinbam and R. Mythili
B.S.A. Crescent Engineering College, Chennai, India
Jaya Engineering College, Chennai, India
Department of Media Sciences, College of Engineering, Chennai, India

Abstract: The study presents a methodology for achieving hardware-assisted simulation and evaluation based on the concept of Intellectual Property (IP) cores, providing standalone test vectors as inputs. It is also described that a prototype software/hardware implementation of the proposed approach and presented a case study for PIC Microcontrollers to demonstrate the feasibility of our approach.

Key words: IP cores, PIC, gpsiim, JTAG

INTRODUCTION

In recent days, the continuous increase in silicon capacity and the problem of the rapid growth of design productivity gap and time-to-market demands, leads to the reuse of Intellectual Property (IP) and rapid system prototyping using post-fabrication programmable logic devices such as Field-Programmable Gate Arrays (FPGAs) for System-on-a-Chip (SoC) and embedded system designs. It is true that the development time of a system or a chip can be significantly reduced if existing IP components are (re-)used and can be embedded earlier into the design for functional simulation and evaluation purposes. In some cases where IPCores are available as gate-level netslists or they are very complex, the software based hardware simulation process usually takes long time. Therefore, it would be beneficial if we could implement or emulate these IPCores in hardware using a low-cost FPGA prototyping board.

To conclude the efficiency of the methodology, a novel and practical approach that allows the user to seamlessly integrate hardware-implemented IPCores into a software simulation environment. In this approach, IPCores are implemented in an FPGA device. The designer can evaluate the IPCores either standalone or together with other design components by generating input vectors on the host computer (e.g., a PC or a workstation), applying these test vectors to the IPCores implemented in the FPGA device, retrieving the output vectors from the device and finally sending them back to the host computer for visualization and analysis. For demonstration purpose, the test vectors are given as standalone inputs.

Integrating a hardware verification platform into a software simulation environment is not a new concept. Earlier efforts are reported, for example, in Haufe et al. (1998) to accelerate hardware verification tasks, several commercial and non-commercial FPGA-based hardware emulation engines and rapid prototyping platforms have been introduced in Sarmadi et al. (2002). Although they offer a large amount of logic capacity and high performance, they are typically very expensive. Furthermore, for each platform the user needs a board-specific software layer and a proprietary protocol to communicate with the associated hardware, although there is an ongoing effort to define a standard co-emulation API for emulation hardware (e.g., The SCe-API). Siripokarpirom and Mayer Lindenburg (2000) discusses the hardware-assisted simulation.

Some earlier work also uses a JTAG-based interface for debugging purposes as discussed in de la Torre et al. (2000). It uses Boundary-Scan Cells (BSCs) to control or capture the states of internal signals inside the FPGA device. Synapti-CAD’s SimuTAG uses a similar approach, but it enables both a hardware prototyping board and an HDL simulator (Bellows and Hutching, 1998) to be linked together through the boundary scan. The main drawback of such an approach is that the number of BSCs available restricts the number of signals in the device. In order to capture internal signals, the designer must route the signal watch-points to the available BSCs. Since the number of available I/O pins of the device limits the length of the boundary-scan chain, the number of internal signals that can be monitored is also limited. In addition, if only a few BSCs are used, all bits of unused BSCs in the boundary-scan chain have to be shifted in or out. In contrast to this approach, we do not use BSCs in our

Corresponding Author: M. Sangeetha, B.S.A. Crescent Engineering College, Chennai, India

**IPCORES FOR PIC MICROCONTROLLERS-SYNTHPIC18**

For the case study, we are going to use the recent version of PIC Microcontrollers (i.e., PIC 18FXX2 (2002)). The IPCore version of PIC 18F series is still not available in the industry. The design of SynthPic18 is constrained by a synthesizable-pipelined design pattern, with each stage of the pipeline, capable of supporting internal parallelism. The four important building blocks of SynthPic18 are Processing Unit, ALU, Registers and Program Memory is implemented as shown in Fig. 1. The 18F MIPS instruction set is implemented in the processor level, with all internal concurrent processing, to reduce the simulation time. The above said modules are designed with synchronous blocks to reduce the data errors. Synthpic18 can be delivered with the following three most important advantages: (1) It accelerates the development of new products to meet today’s time-to-market challenges, (2) To reduce the possibility of failure, based on design and verification of a block for the first time and (3) Portable -that is, able to be easily inserted into any vendor technology or design methodology.

**Features:** In this, we are going to place the complete instruction set (in binary format) in the program memory in order to reduce the access time. In each instruction cycle, the instruction is read from the ROM and the execution done following the flowchart. During the RESET operation, all registers are given with the default values and the inputs to the program are given to the input ports. After completion of the execution of the complete instruction set, the result can be verified in the output ports. Here, the system oscillator clock can be used by internally dividing by 4. Concurrent instruction processing for each instruction cycle and both Processing and Storage in the same instruction cycle as shown in Fig. 2. The path which used in code implementation is shown in Fig. 3.

Some of the constraints we have taken for the core are:

- Initially, we have tried RAM with multiple reads and writes (using arrays). But it is not synthesizable. So, we gone for Block EEPROM with single read and write ports. We have planned an alternate structure (i.e.,) File structures.
- For Table Read Operations (2 cycle), the instruction at the location pointed by Table Pointer is mapped during the first cycle and the program byte is read in the second cycle.

**Clock divider:** Crystal Oscillator clock is divided into 4-phase clock in order to synchronize the instruction

---

**Fig. 1: IP cores-block diagram**
Fig. 2: IP cores-flowchart

Fig. 3: Data path implementation
Fig. 4: Instruction cycle-timing diagram

Table 1. Clock phase-operations

<table>
<thead>
<tr>
<th>Clock phase</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Instruction decode and read PC</td>
</tr>
<tr>
<td>Q2</td>
<td>Operand fetch</td>
</tr>
<tr>
<td>Q3</td>
<td>ALU processing or update table pointer</td>
</tr>
<tr>
<td>Q4</td>
<td>Update RAM with results and PC mapping</td>
</tr>
</tbody>
</table>

execution time. CLK is divided into Q1, Q2, Q3 and Q4 as shown in Fig. 4. Table 1 shows the implemented operation in each clock phase.

SIMULATION RESULTS

In present approach, the software partition is placed as IP Blocks. On the other side, the hardware partition is placed as FPGA CLBs. Usually, the access will be faster for this when compared to software partition. Since, in our methodology, the software partition is converted into IP Blocks, the access will be faster when compared to regular method. The simulation results (Fig. 5) represents the operations in four clock phase.

**Frequency of operation:** According to current simulations for each instruction, it takes 4 clocks to execute. So, if

Clock period = 100 n sec then
Instruction Cycle = 4 * clock period = 4 * 100 n sec
T = 400 n sec
Frequency F = 1/T = 1/100 n sec
F = 25 MHZ

This is within the range 4-40 MHZ (Default). We can improve by finding the feasible minimum and maximum frequency ranges.

Fig. 5: Simulation results

CASE STUDY

**SynthPic18 Core:** As a case study, we used SynthPic18 Core, which we developed for PIC 18F series of Microcontrollers for the partitioned software part. SynthPic18 Core has four input ports and one output port as shown in Fig. 6.

The input to this taken as the partitioned software code of the PIC 18F device and the four input pin values. The output of this is mapped to the FPGA output pin.

The partition is made based on the gpsim (PIC simulator) profile (i.e.,) the most repeatedly used
both hardware/software partitions as separate FPGA CLBs with our method of software partition as IP Blocks. It is sure that our method will show lesser simulation time than the regular method.

CONCLUSIONS AND FUTURE WORK

In this study we have proposed an approach that enables the user to integrate hardware-implemented IPCores into a software-based simulation environment using a simple FPGA-ROM based structure. We have demonstrated the functionality of a prototype implementation of our approach using a case study for which a SynthPic18 Core was mapped onto an FPGA. Currently, we are planning to extend our approach to a simulation/emulation environment to allow the user to use an FPGA prototyping board and also to get the serial inputs from JTAG.

REFERENCES


