A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics
May 23 – 25, 2002, Cluj-Napoca, Romania

Speeding up Design Verification Using Co-Operation of Simulation and Emulation

Ghazanfar Asadi **Seyed Ghasem Miremadi**
Siavash Bayat Sarmadi **Ali Reza Ejlali**

{asadi, bayat, ejlali}@ce.sharif.edu, miremadi@sharif.edu
Sharif University of Technology
Department of Computer Engineering
Azadi Ave., Tehran, IRAN

Abstract
A method for simulation-emulation co-operation of digital circuits is presented. In this approach, parts of a circuit are simulated, while emulating the rest of the circuit. The way of establishing communication between the simulator and the emulator is based on Verilog PLI. The results show that this method can significantly reduces the simulation time.

Keywords
Simulation-Emulation Co-Operation, Verilog PLI, Emulation, Design Verification

1. INTRODUCTION
Simulation and logic emulation [12], [13] are two important approaches to design verification. One advantage of the logic emulation is its speed, which is about 10^3 to 10^6 faster than simulation [11]. Another advantage of the logic emulation as compared with the simulation is the possibility of connecting an emulator to the real application environment. However, the time overhead of emulation is high, because both logic synthesis and technology mapping are very time-consuming steps [3].

While hardware emulation provides very high performance, simulation benefits some features not available in emulators, such as:

- The possibility of observing and controlling the signals and values of a circuit during simulation [1].
- Both synthesizable [4] and non-synthesizable models can be simulated.

As discussed above, both simulation and emulation have some advantages and disadvantages. Some attempts to incorporate the advantages of both methods have been described in [2] and [3]. The idea presented in [2], uses an emulator to accelerate event-driven simulation of some non-synthesizable behavioral models. In that work, logic is synthesized into FPGA chips, while simulator manages events. This removes the event-processing overhead from behavioral simulation.

In [3], parts of a circuit are simulated, while emulating the rest of the circuit. The way of establishing communication between the simulator and the emulator in [3] is based on using the TEXTIO package [8], which is available in the standard VHDL library.

It should be noted that in the simulation-emulation co-operation method in [3], no physical emulator is used; another simulator is also employed instead of the emulator in order to verify the method. This method has some notable disadvantages. For example, the use of text files could be very time-consuming because of the disk access time and also, the approach is only applicable to VHDL models.

In this paper, a Simulation-Emulation Co-OPeration environment, called SECOP, is described. This is based on the Verilog Program Language Interface (PLI) [9], [10] to establish communication between a simulator and an emulator. This method combines most of the advantages of simulation and emulation.

In essential, PLI is a mechanism to invoke a C function from a Verilog code [10]. Verilog does not have any predefined construct for sending data to or receiving data from a port. Furthermore, Verilog does not have any construct for writing data to or reading data from a specific memory location. Here, it is possible to use PLI to write routines for the purpose of doing the above tasks. In this way, the C routines can be invoked to access computer ports as well as memory locations.

Some simulators can simulate both VHDL and Verilog codes. Also, some simulators can simulate a model in which parts of the model are coded by VHDL, while the rest of the model is coded by Verilog. Using such simulators, Verilog PLI can be employed even in the VHDL codes. Figure 1 shows a schematic diagram of the method.

The main advantages of the SECOP method as compared with simulation and emulation are as follows:

- A prototype of a model, which is not completely synthesizable, can be developed. In this case, the synthesizable part of the model is emulated, while the non-synthesizable part is simulated.
- The time overhead caused by the logic synthesis and technology mapping during the iterations of modification can be removed [3]. In this case, those parts of the circuit that require redesigns are simulated and the other parts are emulated for which do not need the logic synthesis and technology mapping steps in each iteration.
- As mentioned above, an advantage of emulators as compared with simulators is the possibility of connecting an emulator as a prototype to the application environment. With the use of the SECOP method, simulators can also be connected to the application environment through emulators.
- Some circuits may be too large and need a great deal of FPGA resources, which can be emulated with some emulators, such as CoBALT [7]. But synthesizing and technology mapping of these large designs are very time-consuming as well as pure simulation of them. In such models, the SECOP method can be used to accelerate the simulation of the circuit model.

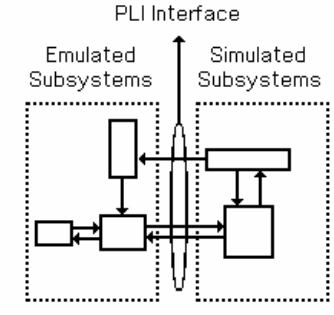

Figure 1. SECOP method based on PLI

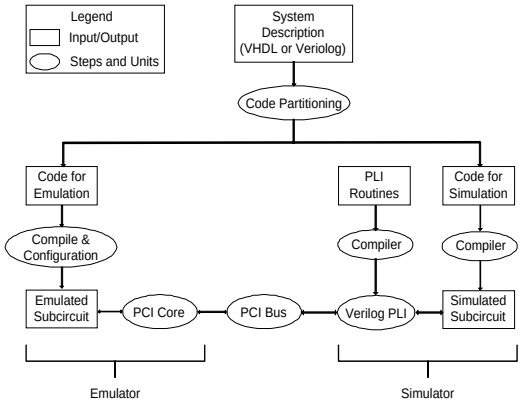

Figure 2. Overview of the SECOP environment

In this paper, a PCI-based PLDA board [6] is used for emulation part and the ModelSim simulator (Version 5.5a)[5] is used for simulation part of the SECOP environment. To evaluate the method presented in this paper, two example circuits have been used: a simple RISC processor and a 256-point FFT processor. Furthermore, TEXTIO method, which is proposed in [3], is also experimentally evaluated. The results and speedups obtained from the experiments are reported.

The rest of this paper is organized as follows. In Section 2, the hardware, and the software of the SECOP environment, as well as the communication strategy to interconnect the simulator and the emulator, is described. In Section 3, the experimental results are presented. Finally, the conclusion is given in Section 4.

2. SIMULATION-EMULATION CO-OPERATION ENVIRONMENT

This section describes the SECOP environment, which consists of two parts: a PCI-based PLDA board, and a ModelSim simulator (Version 5.5a). In the experiment, a Pentium III system is used. (933 MHz, RAM=256 MB, OS=Windows Me).

The PLDA board is connected to the computer via a PCI expansion slot. A Flex 10K200SFC484-1 [6] is mounted on the board. The logic, which is synthesized to the FPGA, consists of two parts: a user-defined logic, and a PCI core. The PCI core automatically handles all bus events. The user-defined logic can communicate with PCI bus only through the PCI core.

The PLDA board can be accessed through memory-mapped I/O technique. It is possible to implement a PLI routine, which can access memory locations. The simulator can communicate with the FPGA chip by means of this PLI routine.

To evaluate a design by the SECOP method, three file groups must be prepared for compilation, which are codes for emulation, codes for simulation, and PLI routines. A software tool prepares these files automatically. Figure 2 gives an overview of the SECOP environment.

The ModelSim simulator can simulate a model coded by VHDL and Verilog. Here, the model is a mixture of two parts; one part is coded by Verilog and the other part is coded by VHDL. In this way, the Verilog PLI can be used even in VHDL codes.

In the SECOP environment, the simulator controls and coordinates the activities of the emulator by the PLI routines. When, the simulator needs information from the emulator, a PLI routine is called.

The communication strategy in Verilog is schematically presented in figure 3. As shown in this figure, when the result of a subcircuit in the emulator is needed, the PLI routine, i.e., the $WriteToEmulator, is called. This routine activates the subcircuit in the emulator. To receive the results of the subcircuit from the emulator, another PLI routine, i.e., the $ReadFromEmulator, is called.

The operation of the emulated subcircuit is concurrent with the execution of the statements which are inserted between $WriteToEmulator() and $ReadFromEmulator().

```
module SECOP(inputs,outputs);
  //declaration statements

  always @(inputs)
  begin
    //behavioral statements
    $WriteToEmulator(SubcircuitId, InputV); //PLI routine
    //behavioral statements
    $ReadFromEmulator(SubcircuitId, OutputV); //PLI routine
    // behavioral statements
  end
Endmodule
```

Figure 3. Communication Strategy

Figure 4 shows how the PLI routines are used. Figure 4.a shows the code of a half-adder, which is a subcircuit of a design, and is inserted in the emulator. As shown in figure 4.b, the inputs of the HA (a,b) are sent to the emulator by the $WriteToEmulator PLI routine. The results of the HA (s,c) is received from the emulator by the call of the $ReadFromEmulator.

```
module half_adder(a,b,s,c);
xor(s,a,b);
and(c,a,b);
Endmodule
```

(a)

```
module half_adder(a,b,s,c);
always @(a,b)
begin
$WriteToEmulator(HalfAdderId, a,b); //PLI routine
$ReadFromEmulator(HalfAdderId, s,c); //PLI routine
end
Endmodule
```

(b)

Figure 4. a) A synthesizable gate level model of a half adder b) The substituted code for the emulated half adder module

3. EXPERIMENTAL EVALUATION

Two digital circuits are used to evaluate the SECOP approach: a simple RISC processor, and a 256-point FFT processor. These circuits are general purpose and special purpose, respectively. General purpose digital system can be programmed. So, the resulted speedups of the system will depend on the program executed by it. The speedup is computed according to the following relation:

$$Speedup = \frac{Simulation\ time}{Simulation - emulation\ \ co - operation\ \ time} \qquad (1)$$

3.1 A Simple Arithmetic RISC Processor

The arithmetic RISC processor (ARP) is coded by Verilog with 530 lines of code. The instruction set of this processor is shown in Table 1.

Booth algorithm, restoring division algorithm, and Newton-Raphson algorithm have been used to implement multiplication, division, and square root operations, respectively.

This processor has sixteen 16-bit user registers, stored in a register file. A 16-bit fixed-point number system is used. Figure 5 shows the architecture of this processor.

Arithmetic instructions	Add, Sub, Mul, Div Sqrt	Rd, Rs1, Rs2 Rd, Rs
Logical instructions	And, Or, Xor Not	Rd, Rs1, Rs2 Rd, Rs
Jump and Branch instructions	Jmp, Jz, Jv	Addr

Table 1. The instruction set of the simple arithmetic RISC processor

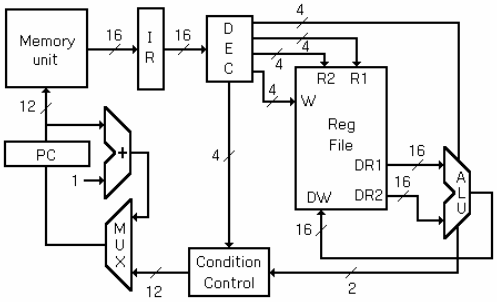

Figure 5. The architecture of the simple arithmetic RISC processor

The ALU and register file of the ARP have been inserted in the emulator, which uses 85 percent of the FPGA resources and all other parts are simulated.

The instructions are divided into three groups:

1- Group1: Simulated instructions = {Jmp, Jz, Jv}, these instructions are executed in simulator.

2- Group2: Emulated time-consuming instructions = {Div, Sqrt}, these instructions are executed in emulator and they are complex.

3- Group3: Emulated fast instructions = {Add, Sub, Mul, And, Or, Xor, Not}, these instructions are executed in emulator and they are simple.

Several programs with different number of instructions from Group1, Group2 and Group3 have been used. Table 2.a shows the resulted speedups when the Verilog model is used. For calculating the speedup, each program was run several times. As shown in Table 2.a, when a program only consists of simulated instructions, even though the program never uses emulator, the resulted speedup is the highest. The reason is that the code, which is loaded into the simulator in the SECOP approach, is smaller than what is loaded into the simulator in a pure simulation method. Therefore, even when the emulator is not used, a smaller code in the simulator results in a high speedup. Also, a program, which only consists of simulated instructions, need not communicate with the emulator. Therefore, it does not consume communication time. On the other hand, a program with emulated instructions consumes communication time, which results in a reduction of the speedup in comparison with the speedup resulted from running a program, which only consists of simulated instructions.

N1	N2	N3	Simulation Time (ms)	SECOP Time (ms)	Speedup
3	2	10	22137	4450	4.97
1	4	0	22080	1540	14.34
2	0	8	9413	3023	3.11
3	0	0	974	42.8	22.76
1	12	42	102900	20033	5.14
5	2	7	19720	3383	5.83
4	2	1	19533	3383	5.77

(a) For 10000 iteration

N1	N2	N3	Simulation Time (ms) for 10000 iteration	SECOP Time(ms) for 10 iteration	Speedup
3	2	10	22137	6043	0.00366
1	4	0	22080	1956	0.01129
2	0	8	9413	4025	0.02340
3	0	0	974	488.7	1.990
1	12	42	102900	26385	0.00390
5	2	7	19720	4597	0.00429
4	2	1	19533	4560	0.00428

(b)

N1, N2, and N3: Number of Group1, Group2, and Group3 instructions in the program

Table 2. The resulted speedups for a) PLI-based SECOP, b) TEXTIO-based SECOP method vs. pure simulation for ARP

In order to compare the method which is presented in this paper, and the method, which is presented in [3], a similar set of experiments have been done using TEXTIO package. In these experiments, a method which is proposed in [3] is used; one text file have been used to transfer data from the simulator to the emulator, while another text file has been used to transfer data from the emulator to the simulator. Figure 6 shows an overview of this method.

Table 2.b shows the resulted speedups from the TEXTIO method vs. pure simulation. As shown in Table 2.b, the TEXTIO method cannot reduce the simulation time. The reason is that reading from or writing to text files is very time-consuming. The only exception is when the program only consists of Group1 instructions. In this case, simulation time is reduced because the simulator needs not communicate with the emulator. Therefore, there are no time-consuming file accesses.

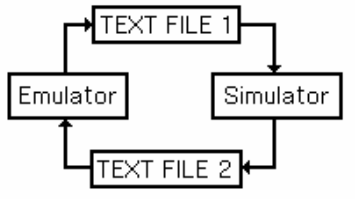

Figure 6. Overview of the experimented TEXTIO method

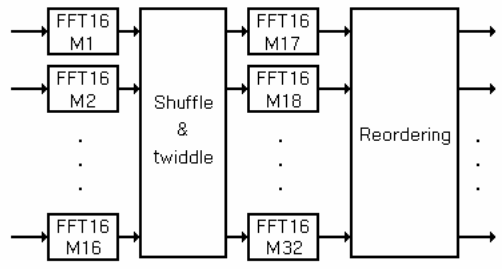

Figure 7. A 256-point FFT processor

3.2 A 256-Point FFT Processor

Another benchmark is a 256-point FFT processor. The code size of Verilog for 256-point FFT processor is 580 lines. The processor uses decimation-in-time (DIT) FFT algorithm.

The architecture of this FFT processor is shown in Figure 7. The 256-point FFT processor is made up of a shuffle and twiddle module, a reordering module and 32 16-point FFT modules.

The code, which is described the 256-point FFT consists of non-synthesizable modules of shuffle and twiddle unit and reordering unit as well as synthesizable module of 16-point FFT unit.

A 256-point FFT processor is too big to be synthesized into a Flex 10K200SFC484-1 FPGA. Also, a 16-point FFT unit uses 75 percent of the FPGA resources (This amount of resources is used by 16-point FFT unit as well as PCI core). Therefore, only a 16-point FFT unit has been synthesized into the FPGA, and all the other parts have been simulated. To compute a 256-point FFT, the simulated part uses the emulated 16-point FFT unit 32 times.

The resulted speedup is 12.75.

4. CONCLUTION

Simulation and emulation are two approaches to design verification. Another approach is the SECOP, which incorporates the advantages of both simulation and emulation. In this method some parts of a circuit are simulated and the other parts are emulated.

In this paper, the SECOP method is presented, which is based on Verilog PLI. This work shows that the SECOP method can be used in order to reduce the simulation time significantly. The proposed approach has been verified by two sample codes. As the results show, depending on the sample code and the inputs, the speedup of this method varies from 3 to 22 times, as compared with the pure simulation method.

Furthermore, this method has been compared with a method, which is based on VHDL TEXTIO package. The results show that the method presented in this paper, is much faster than the method based on VHDL TEXTIO.

5. REFERENCES

[1] Abramovici, M., Breuer, M. A., and Friedman, A. D., [1995], Digital Systems Testing and Testable Design, IEEE Press, Revised edition.

[2] Bauer, J., Bershteyn, M., Kaplan, I., and Vyedin, P., [1998], A Reconfigurable Logic Machine for Fast Event-Driven Simulation, *in Proceedings of ACM/IEEE Design Automation Conf. (DAC)*, pp. 668-671.

[3] Canellas, N., Moreno, J. M., [2000], Speeding up hardware prototyping by incremental Simulation/Emulation, *in Proceedings of 11th International Workshop on Rapid System Prototyping*.

[4] De Micheli, G., [1994], Synthesis and Optimization of Digital Circuits, McGraw-Hill.

[5] HTTP://www.model.com.

[6] HTTP://www.plda.com.

[7] HTTP://www.quickturn.com/w_new/CoBALTUltra.htm.

[8] IEEE Std 1076-1993: IEEE Standard VHDL Language Reference Manual.

[9] IEEE Std 1364-1995: IEEE Standard Verilog HDL Language Reference Manual.

[10] Mittra, S., [1999], Principles of Verilog PLI, Kluwer Academic Publishers.

[11] Rowson, J. A., [1996], Hardware/Software Co-Simulation, *in Proceedings of 31st ACM/IEEE Design Automation Conference*, pp. 439-440.

[12] Varghese, J., Butts, M., and Batcheller, J., [1993], An efficient logic emulation system," *IEEE Trans. on VLSI Syst.*, vol. 1, pp. 171-174.

[13] Walters, S., [1991], Computer-aided prototyping for ASIC-based system, *IEEE Design Test Comput.*, pp. 4-10