ACCELERATING THE TEST AND VERIFICATION OF OPEN CORE MICROCONTROLLERS AND OTHER IPS WITH AN ETHERNET INTERFACE

Sibilla B. L. França, Christophe F. L. Bricout, Ricardo P. Jasinski, Volnei A. Pedroni

UTFPR, Dept. of Electronics Engineering
Curitiba- PR, Brazil

ABSTRACT

Test and verification are a crucial stage in the development of an IP (intellectual property) core. Using industry standard techniques, an extensive series of tests are performed, looking for possible failures in the design. This paper describes an approach that can be used to accelerate the test and verification of IP cores such as microcontrollers, coprocessors, communication controllers and others, using a standard Ethernet interface. Instead of running an extensive set of testbenches on a simulator, a test environment is set up in which a large part of the functional verification is replaced by actual execution in hardware. A case study is presented in which the presented method is applied in the verification of an open source 8-bit microcontroller IP core, drastically reducing total verification time. To a large extent, the proposed method is technology and operating system independent. The proposed method is supplementary to the conventional methods based on testbenches, enabling the designer to choose between both approaches for each test case.

1. INTRODUCTION

Functional verification is a fundamental phase of circuit design, in which all functionalities should be exercised in order to guarantee operation according to the original specifications. It is usually a complex and time demanding task, frequently consuming more than half of the computer and human resources dedicated to this kind of project [1]-[2]. If performed late in the design process and the resulting implementation does not match the specified features, enormous commercial losses can occur [3].

The verification process usually includes several distinct techniques. Commonly, typical use cases and corner case tests are manually chosen, implemented and applied. Functional and timing simulations are performed, usually as part of an automated test suite. After the hardware is considered ready, extensive high-level (i.e., application) tests are executed. Additionally, when suitable, pseudo-random stimuli can also be used [3]. Since most of these tests are usually performed via a simulator, when the device under test (DUT) contains a high number of circuit nodes, this operation can be highly time consuming.

Two traditional approaches to circuit verification are the "golden design" (or reference model) and self-checking testbenches. In the golden design (Fig. 1) approach (also called golden chip, of gold vectors), a reference model is instantiated side by side with the DUT, and the same set of input stimuli is applied to both units. The resulting outputs are then compared, and should be equal within the expected tolerance.

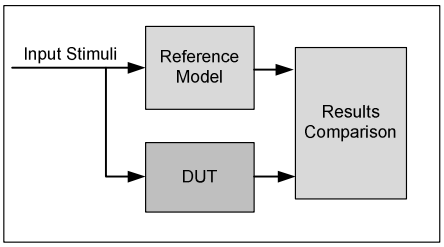

Figure 1. Golden design approach.

If self-checking testbenches are also used (Fig. 2), as in [4]-[7], the DUT is usually instantiated inside the testbench top-level entity. The testbench itself will then generate the appropriate stimuli, and compare the DUT outputs with known results, or even with a different implementation of the same algorithm.

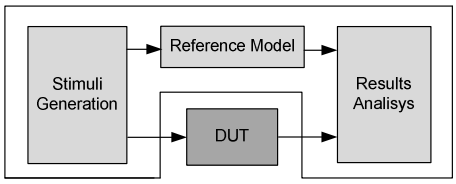

Figure 2. A self-checking testbench.

In both cases, if a comprehensive test coverage must be achieved, the corresponding simulations will be very time consuming.

The approach proposed in this paper allows the designer to replace part (or most) of these simulations with actual execution of the test cases in the DUT, implemented in an FPGA. A case study is included in the paper for the verification of an 8-bit open source microcontroller (an Atmel AVR clone), where simulation runs requiring hours to be executed on a high-end PC were replaced successfully with the actual execution of test code in the DUT, reducing the average time to run a test case to less than a second. This time gain allowed the development of more extensive test sets, improving the effectiveness of the verification process.

2. PROPOSED METHOD

Many IP cores operate on data available in RAM, either on-chip or off-chip. Examples include microprocessors, which execute instructions stored in a program memory; video controllers, in which pixel data is stored in a dedicated memory section; or many other DMA (direct memory access) capable peripherals.

Traditionally, the initial contents of such memories are specified in simulations via memory initialization files. These memories are then connected to the DUT by instantiation inside a top-level entity, which will then coordinate the process of feeding data to the IP core being tested. However, in a complex architecture, the number of clock cycles needed to execute even a simple application can be prohibitively high, and a test case may take several hours to complete.

The proposed approach (Fig. 3) consists in the following steps:

1) Synthesize the DUT to an FPGA, along with:
   a) an Ethernet interface (more specifically, the MAC – media access controller – stage);
   b) an on-chip memory;
   c) an interface block, used to initialize the Ethernet MAC and control the test sequence.
2) Generate memory initialization files containing the test programs to be executed on the DUT.
3) Use a PC tool to coordinate the process of transmitting the data to the on-chip memory, starting the DUT, and reading back the outputs of the process.
4) The PC is then used to check the results against the golden vectors, and restarts the process until all tests are run.

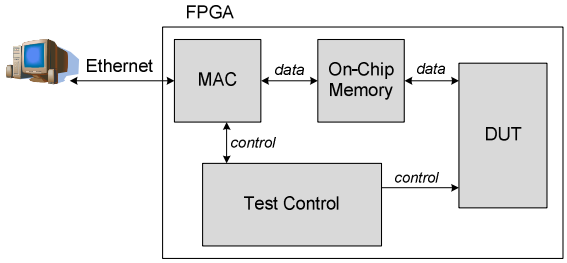

Figure 3. Test setup for the proposed method.

As can be seen, the only requirements are a set of software tools (which are all freely available, either from open sources or provided by FPGA vendors) and a board containing an FPGA chip and a physical Ethernet interface. This setup can be run in practically any operating system supported by the FPGA vendor, and integrates easily with the standard tools composing the original FPGA design flow.

It is worth mentioning that most current FPGA development suites support real-time updating and read back of on-chip memories, which can aid in the initial setup and debugging. This eliminates the need to update the entire FPGA configuration whenever new test data must be sent for evaluation.

3. STUDY CASE: MICROCONTROLLER

The proposed technique was used in the verification of an open core microcontroller IP (an Atmel AVR clone), which was being considered for utilization in a real industrial application. The selected microcontroller IP core was available via the OpenCores Project website (opencores.org), as a synthesizable description in VHDL language. However, like most non-commercial cores, it had never undergone formal verification or rigorous performance tests.

3.1. Microcontroller Selection

The work described in this paper was part of a larger, industry sponsored project, which also included the selection of an adequate 8-bit microcontroller under the following requirements:

i) The project should be an implementation of an existing, commercial product.
ii) Availability of documentation, describing the implemented functions and design limitations.
iii) Availability of testbenches.
iv) Support tools for software development.
v) Preferably described in VHDL.
vi) It should be a design with recent updates.

A total of thirty three microcontroller designs were surveyed and evaluated. However, due to the ample requirements, no one could be found that satisfied all of the original goals. Especially hard to find were designs with a proper documentation, which reinforced the need for a commercial counterpart with documentation available from other sources.

Another important requirement is the availability of testbenches, which help understand the core's internal working, operations sequencing and the values of control signal and buses. Most of the surveyed projects provided some sort of test data demonstrating basic design functions; however, few of them were actual testbenches implemented in an HDL. The larger part included only software tests in Assembly language, performing tasks such as printing messages on a serial port, arithmetic calculations, or reading and writing to a memory section.

Among all surveyed candidates, four processor models were selected, with one or more implementations available for each one. The selected microcontrollers included the Intel 8051, Zilog Z80, Motorola 6805, and Atmel AVR.

A more detailed examination of these preselected cores was then performed, more specifically, studying the available documentation and code, and running brief operational tests. Eventually, the project named AVR Core was chosen. This project is recent, described in VHDL, consumes few logic resources in the FPGA (less than 2,000 LUTs and 800 registers), can use freely available C and Assembly compilers, and also featured a software tool to convert the generated binary code into

VHDL, which is useful for debugging both the microcontroller and the initial test setup. On the downside, no testbenches were available.

3.2. Initial Tests of the AVR Core

The selected microcontroller is a RISC CPU, code-compatible with the Atmel ATmega103. Its features include 32 × 8-bit general purpose registers, up to 128 kB of program and 64 kB of data memory, a UART, two 8-bit timer/counters, and two parallel ports. The core also supports the AVR port of the uC/OS-II real-time operating system kernel.

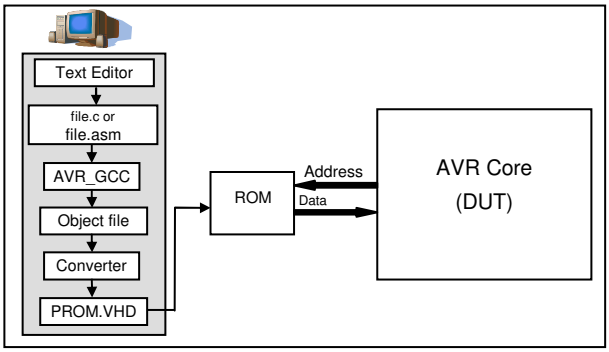

Figure 4. Initial test setup (without the proposed method).

After correctly synthesizing the core to an FPGA development board (containing an Altera Cyclone II EP2C35F672C6N device), an initial test setup was prepared (Fig. 4), in order to confirm its basic working. At this time, the Ethernet interface was not used; instead, the generated binary file was converted into a VHDL file with a proper tool, and synthesized along with the processor code.

Obviously, this approach in not adequate for the execution of a large number of test cases, since the FPGA programming bitstream would have to be regenerated and reprogrammed for each test case. Nevertheless, the core was tested with several ASM and C programs, until its basic operation was found to be satisfactory.

3.3. Speeding Up Tests with the Proposed Method

The initial test setup presented in Section III.B clearly has many disadvantages. First, the entire FPGA design must be recompiled; depending on the vendor tools suite, this can require all source files in the design to be reanalyzed and resynthesized. Second, it involves the generation of VHDL code from a software object file, unnecessarily adding complexity to what should be a software testing process. Third, the FPGA must be reprogrammed, and all on-chip peripherals and other circuits must be reinitialized. Fourth, it may be extremely hard to automate in practice, due to the difficulties of integrating a large number of software tools from different vendors.

The tests using the proposed method were executed in a custom FPGA board, which included a Cyclone II FPGA EP2C8F256C8 and an Ethernet PHY 78Q2123 chip. The Ethernet-accelerated test setup is shown in Fig. 5. The process begins with the initial configuration of the Ethernet MAC IP core. Since the DUT in this case was a microcontroller, its programmability has been exploited in order to ease this initialization process; the startup code is stored in its ROM, which is automatically executed when the processor is run. After that, the processor simply awaits the arrival of new test data.

When a new test program is received via the Ethernet interface, the Test Control Unit (TCU) senses it and starts the execution. Fig. 6 shows an Ethernet frame being sent to the FPGA and captured on the wire using a packet sniffing tool.

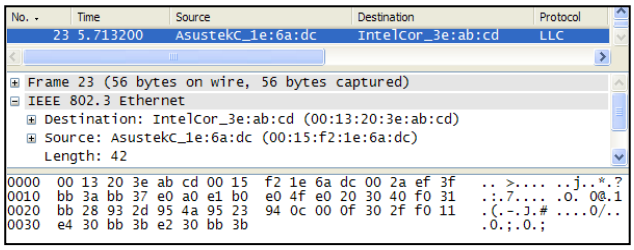

Figure 6. Frame sent to a test circuit.

In this particular implementation, the maximum test program length is equal to the maximum frame length enabled by the Ethernet controller, i.e., approximately 1,500 bytes. Naturally, this could be increased with the implementation of additional control logic, but this amount was considered enough to run all intended test cases.

At the end of every test program, selected data (typically, operation results) are stored back in data memory, and the processor indicates the end of a test run through a data port. The TCU then transmits the resulting test data to the PC, where it is evaluated. These results are checked and logged to a test output file, and a new test run is then started.

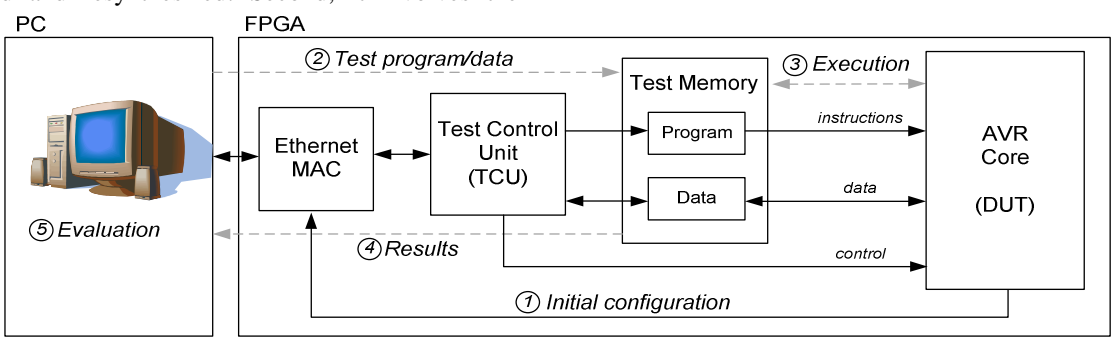

Figure 5. Proposed method applied to AVR Core.

Since we were interested in proving the correctness of the processor implementation, test programs were created that exercised most instructions from its instruction set.

4. RESULTS

Currently, more than 80% of the documented instructions were already tested, and all of the available addressing modes. All of the arithmetic, logic, and branch instructions were exercised, and matched the expected results in all cases.

The proposed method and the presented test setup enabled the execution of all tests sequentially, which would be impossible via simulation due to memory and processing limitations. When unexpected results were found, the test could be quickly re-run in real time, which was unthinkable with the simulation approach. All tests were performed at a fixed speed of 50 MHz.

In order to evaluate the performance gain provided by the proposed method, a test run with a length of 5×10^6 clock cycles was executed in Mentor Graphics Modelsim simulator, which required 2,262 seconds (37.7 minutes). When executed in real-time in the FPGA, using the proposed method, this same test is run in only 100 ms (for a clock frequency of 50 MHz). These results indicate that, in this case, the proposed method is 22,620 times faster than a simulation of the same test code.

Finally, some faults were inserted in the microcontroller to demonstrate that the proposed method is capable of identifying errors in design. By examining test sequence outputs, the failing tests were identified and provided enough information to locate the fault in the original circuit.

Figs. 7 and 8 demonstrate a test sequence in which a register is cleared and then incremented 15 times; in the end, its value is compared with a hexadecimal value of $0F. At the same time, the count value is replicated at one of the microcontroller IO ports (Port B, in this case) and stored in memory, in order to be evaluated by the PC coordinating the tests. Fig. 8 shows the test signals captured with a logic analyzer; the count value can be seen in port B[3..0]. The test code presented in Fig. 7 is the same that was shown in Fig. 6 on the Ethernet wire.

```
      .org $0008
000008      ef3f        ser TEMP
000009      bb3a        out DDRA, TEMP
00000a      bb37        out DDRB, TEMP
00000b      e0a0        ldi r26, $00
00000c      e1b0        ldi r27, $10
00000d      e04f        ldi r20, $0F
00000e      e020        ldi r18, $00
00000f      3040        loop: cpi r20, $00
000010      f031              breq check
000011      bb28              out PORTB, r18
000012      932d              st X+, r18
000013      954a              dec r20
000014      9523              inc r18
000015      940c              jmp loop
000016      000f
000017      302f        check: cpi r18, $0F
000018      f011              breq ok
000019      e430              ldi r19, $40
00001A      bb3b              out PORTA, r19
00001B      e230        ok:   ldi r19,$20
00001C      bb3b              out PORTA, r19
```

Figure 7. Assembly code for the sample test sequence.

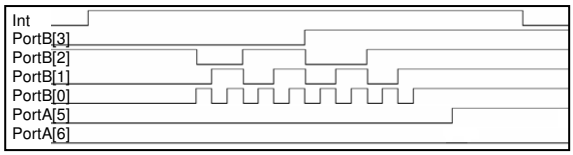

Figure 8. Test results captured with a logic analyzer.

6. CONCLUSIONS

The approach presented in this paper allows a system designer to choose between the traditional simulation approach and actual hardware execution for each test case, in a larger test sequence. A case study was presented, in which the proposed method was applied to the verification of an 8-bit open core microcontroller. The new test setup provided speed gains of up to 22,620 times compared to the simulation-only approach. This time savings allowed the development of more extensive test sets, improving the effectiveness of the verification process.

Even though the DUT programmability has been exploited to ease the initialization process, this is not a strict requirement, and the same approach can be applied to simpler IP cores like peripherals and coprocessors. To a large extent, the proposed method is technology and operating system independent.

The simple requirements of the proposed approach (an FPGA with modest on-chip memory, and an Ethernet interface) enable its adoption in virtually any development system.

10. REFERENCES

[1] A. Meyer, *Principles of Functional Verification*, Newnes, 2003.

[2] J. Bergeron, *Functional Verification of HDL models*, Springer, 2nd ed., 2002.

[3] L. Fournier, Y. Arbetman, M. Levinger, "Functional verification methodology for microprocessors using the genesys test-program generator," Proc. Design, Automation and Test in Europe Conference, Mar. 1999, pp. 434-441.

[4] W. S. Encinas Jr, C. A. Dueñas, "Functional verification in 8-bit microcontrollers: a case study," Symp. Microeletronics Tecnology and Devices, 2001.

[5] I. Rancea, V. Sgarciu, "Functional verification of digital circuits using a software system," IEEE Int. Conf. on Automation, Quality and Testing, Robotics, May 2008, pp. 152-157.

[6] Myoung-Keun,You, Yong-Jin Oh, Gi-Yong Song, "Implementation of a hardware functional verification system using SystemC infrastructure," IEEE Region 10 Conference, Jan. 2009, pp. 1-5.

[7] K. R. G. Silva, E. U. K. Melcher, G. C. S. Araújo, V. A. Pimenta, "An automatic testbench generation tool for a SystemC functional verification methodology," 17th Symp. Integrated Circuits and Systems Design, Apr. 2010, pp. 66-70.