AN INTEGRATED-APPROACH FOR DESIGNING AND TESTING SPECIFIC PROCESSORS

[1,2,3] Cesar Giacomini Penteado, [4]Edward David Moreno, [1]Sérgio Takeo Kofuji

[1]University of Sao Paulo (USP), Sao Paulo, Brazil
[2]LSI-TEC of University of Sao Paulo, Sao Paulo, Brazil
[3] UNIVEM – Centro Universitário Euripides de Marilia, Brazil
[4]Federal University of Sergipe (UFS), Aracaju, Brazil

ABSTRACT

This paper proposes a validation method for the design of a CPU on which, in parallel with the development of the CPU, it is also manually described a testbench that performs automated testing on the instructions that are being described. The testbench consists of the original program memory of the CPU and it is also coupled to the internal registers, PORTS, stack and other components related to the project. The program memory sends the instructions requested by the processor and checks the results of its instructions, progressing or not with the tests. The proposed method resulted in a CPU compatible with the instruction set and the CPU registers present into the PIC16F628 microcontroller. In order to shows the usability and success of the depuration method employed, this work shows that the CPU developed is capable of running real programs generated by compilers existing on the market. The proposed CPU was mapped in FPGA, and using Cadence tools, was synthesized on silicon.

KEYWORDS

Validation, Testbench, FPGAs, Microcontroller, Cadence, ASICs

1. INTRODUCTION

The development of any processor includes an extra work to obtain a validation of each of the operations done for it, which are seen by programmers as instruction implemented. During the encoding of a processor, either in Verilog, VHDL, SystemC or other hardware description language, the description evolves until the first stable versions are obtained. During this process may occur a collapse of some instructions or blocks already previously validated, due to changes in the hardware encoding.

This paper proposes a simple validation and development method which automatically validates the computation of each instruction described. This method also keeps the CPU designed in an idle state, in case of error detection. The encoded program memory itself can be described with some structures of decision and it can be used as a testbench, simply connecting target registers as a feedback.

The proposed method repeats the tests from the first instructions to the new and current instruction being described. It easily shows possible errors during the progress of the CPU encoding. Figure 1 illustrates the concept of the proposed method. In figure 1, on the top, it can

DOI : 10.5121/vlsic.2013.4501

be seen a classic simplified Von Neumann CPU being developed, and it accesses its program memory. Several programs must be written in this memory to test the proper operation of its instructions. At the bottom, it is proposed that this memory is modified and strongly coupled to the CPU, in order to obtain a feedback of its own instructions that were requested by the CPU. Thus, automatic tests can be performed to validate the correct execution of the each instruction.

For each register of interest, in each CPU project, it is proposed that it is made a bus connecting the register and the program memory, making a testbench, which contains the instructions and data to be sent to the CPU. Thus, the testbench is stimulated by the results of its own instructions, progressing or not with the tests. To prove its efficiency, the proposed method was applied in the development of a VHDL version of a real CPU. It is compatible with the CPU present in the PIC16F628 microcontroller, which was successfully validated.

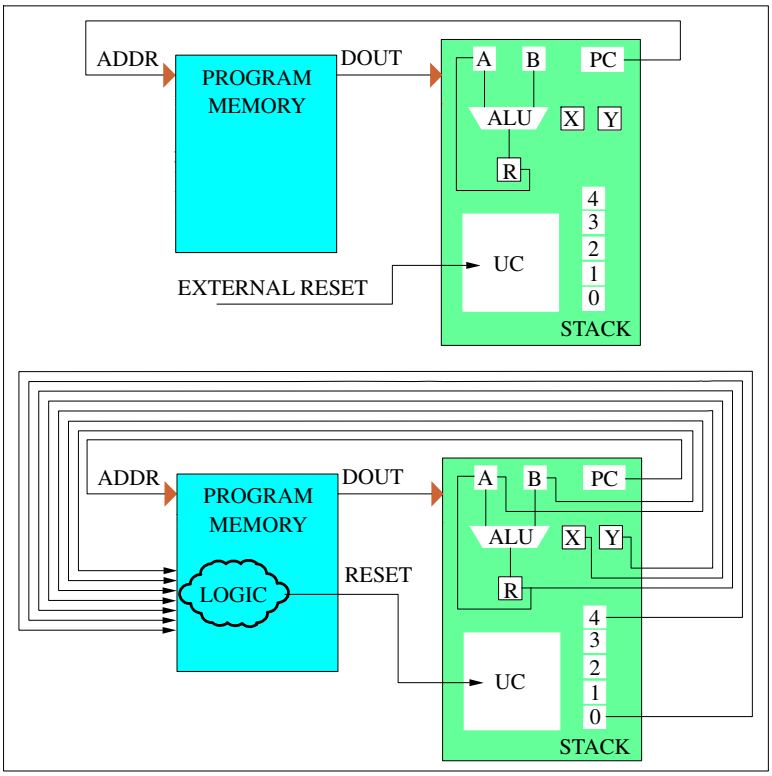

Figure 1. Validation method where feedback registers act in the Testbench

A previous processor was designed by the autors and it called as UPEM (Processing Unit Specific for Peripheral) [Penteado, 2009, Penteado, 2011]. The UPEM have a limited processing capability and it was designed for specific purposes of low power and minimal area. The presented method in this work was conceived to make a new UPEM processor, full compatible with real programs existing on the market. Thus, the proposed CPU is an evolution of UPEM and it is here referred to as UPEM2. It was validated in Cadence digital tools and physically validated in FPGA. The tests were carried out in the LSITEC Design House of the University of Sao Paulo, a laboratory specialized in microelectronics and project for industrial applications.

2. RELATED WORKS

The works listed here clearly show an evolution in the design methodology and verification of processors. It is also related the works that performed VHDL descriptions of parts of the PIC microcontroller.

In [Hu, 1994] the authors present an activity diagram with a technique of resources distribution for verification. The verification methodology presented was divided into: (i) Specifications of engineering requirements. The required information includes registers maps, descriptions of the internal buffers, comparators, counters, initial values of registers, a fast algorithm of how the project will be used and a list of external devices to be used; (ii) Verification plan, which is a baseline for the verification engineer; (iii) Final test chip and code test; (iv) Layout preparation; (v) Post-layout simulation and; (vi) Creation of operation vectors, that are useful for testing the chip in operation.

In [Zhenyu, 2002], the authors present a methodology for functional verification based on simulation to validate a 32 –bit RISC microprocessor, in which the pseudo-random generating method and a method focused on pipeline were used to generate testbenches. This paper proposed a bottom-up verification strategy, consisting of three stages: (i) in the first stage, the goal is to check the basic functions of each module. So, the authors showed that are not necessary to have many testbenches and the handwriting can be used. The authors showed that handwriting is a good direction; (ii) in the second stage, the main functions and the interfaces between the blocks need to be checked and, obviously, the number of needed testbenches increases considerably; (iii) in the final stage, the handwriting method is used to verify borderline cases – minimum and maximum values of registers, software parameters, addressing, stack use, and other parameters.

In [Schmitt, 2004], the Tricorel, an IP owner of a microcontroller, provided by the company Infineon, was mapped in FPGA and the performance of some algorithms was verified, among them the Euclidian algorithm that calculates the MDC of a number, the result of Fibonnacci for number 10,000 and Erastothenes that calculates the first 1000 prime numbers.

In [Castro, 2008], the authors proposed an efficient and functional methodology based on a framework of parameters domain. Additionally, this work has presented an automated tool for generating sources of stimuli in SystemC language, from a specification module of parameters domain. The authors implemented a test environment consisting of five modules around the RTL model under test. Even with the use of automated tools, the source of stimuli needs to be defined during the project and must be written manually.

In [Kwanghyun, 2008], the authors presented a reusable project platform for integration and verification of SOC based on the IP reuse and IP-XACT standard, which is a specification for SoC in Meta data XML, as a standard for describing the components of a SoC platform. This specification describes an IP under several aspects: hardware model, signs, bus interface, memory map, address space and model views.

In work [Penteado, 2009], the authors developed and physically validated a CPU, known as UPEM (UPEM means Processing Unit Specific for Peripheral). The UPEM was developed to perform peripheral functions of microcontrollers and consists of a reduced PIC CPU version. We have used the CQPIC version [Morioka, 1999]. The UPEM has a few PIC registers, a reduced address space and, thus, it is able to address only a single RAM block. The Morioka´s development is different from our work, because it was focused on another microcontroller from Microchip, the PIC16F84.

In [Wrona, 2011], the authors developed a design methodology of tests for the final digital circuit, composed by a software/hardware environment in with, the software parts is being executed at a Personal Computer and the hardware parts are mapped in a FPGA device. This methodology is different from our work, because the tests are performed in the final digital circuit, while our methodology is used during the development of digital circuit.

In [Amandeep, 2012] the authors proposed a system composed by a microcontoller chip and a FPGA device that performs standard tests in the digital circuit under development. The novel feature is that there no need of test patter generator and output analyser as microcontroller performs the function of both. There are some similar characteristics with our work: the proposed system is used while the digital circuit is under development and; the microcontroller is used to compares the outputs of digital circuit with a predefined set of expected values. In our work, the last task is performed by the testbench.

In other works, [Meng, 2010], [Becker, 2012], [Cha, 2012] and [Zhaohui, 2012] the authors discuss about modern techniques to verify the final digital circuit, in order to detects errors in the entire design. These techniques includes the use of SystemC and SystemVerilog languages, UVM - Universal Verification Methodology - and, Co-design approaches to test and verify modern and complexes System On a Chips. These researches has focus on development of systems composed by digital and analog parts, differing of our purpose. Our work is exclusively aimed to digital development.

3. AN INTEGRATED - APPROACH FOR DESIGNING AND TESTING SPECIFIC PROCESSORS

The need to continually check a processor or a specific digital system under development is an expensive task. Therefore, it is possible that the proposed method could assist in the development, by means of self verification of the project under development. For this, we simply describe a testbench with some feedback lines and continuously we made a comparison using a few structures with previously known values.

According to [Hu, 1994], the hardware engineer encodes and performs basic tests on the chip under development. These basic tests can be interpreted as being close tasks to the proposed method. In [Zhenyu, 2002], the work cited the formal verification methods and the simulation-based verification. The last one applies testbenches to stimulate a project and a reference model. Thus, the functionality of the project can be known by comparing the final results.

In [Schmitt, 2004], whereas TriCore1 IP is a complete microcontroller core and has already been validated by Infineon, the verification method differs from our proposal since it is focused on software and our method is implemented directed in hardware.

According to [Castro, 2008], one of the alternatives used to complete the design of a SoC, core or task verification of a module is the functional verification, in which a model in hardware description language in the RLT level is stimulated both random and direct stimuli. Even using automated tools, the source of stimuli needs to be defined during the project and must be written manually. In [Kwanghyun, 2008], the first step of the SoC platform design is an exploration and optimization stage of the architecture, accompanied by an intense architectural analysis.

3.1 THE PROPOSED METHOD

The validation method used in this work could be reused in new developments of new processors. Figure 2 illustrates an example of a developing processor and three simple programs to test the correct execution of the instructions. The detailed architecture of the developing CPU is the same as illustrated in Figure 1. In the examples in Figure 2, our convention is the following mnemonic: (i) MOV X, to move data from the program memory to the destination register, (ii) STO X to store the results of ALU´s operations and; (iii) ADD, SUB and MUL that perform addition, subtraction and multiplication between registers A and B.

In Figure 2, the verification of the program results and the CPU running is done manually; it is an expensive task and may contain typing errors, in case the CPU come to have complex instructions and actions in many registers. In this stage, the CPU is under development and deep changes in the RTL description may occur, leading to a new manual verification cycle of all instructions that have already been validated.

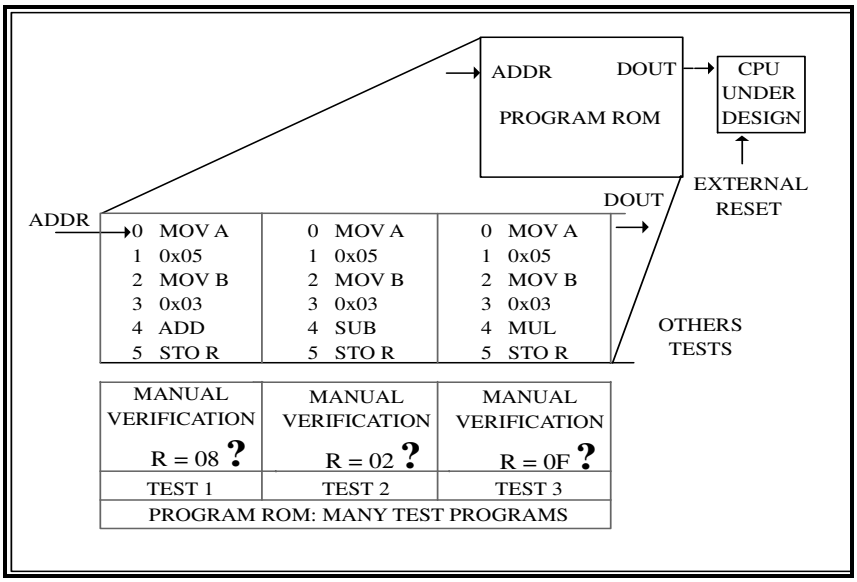

Figure 2. Manual verification of the instructions being developed

Figure 3 illustrates in detail our method, proposed and used in this work, which can help during the development of any CPU, making the verification of results an automated task. In figure 3, the same CPU illustrated in figure 1 is under test. The same programs used in figure 2 are represented by PROG 1, 2 and 3. The fundamental difference is that the program memory is no longer a simple memory and it becomes an efficient depuration mechanism for the CPU computations.

It has been necessary a simple coding of the verification logic of the results, represented by LOGIC 1, 2 and 3. This coding was done manually. With a simple feedback from the target registers, one comparison between the value received and the expected value is sufficient for automatically validating the results of the instructions and the architecture.

In figure 3, the sequence of the automated testing occurs, if the logic indicates success; if any value difference is detected, the CPU stops its activities.

At the end of each test successfully performed, a reset is sent to the CPU, making each test can be independently from each other. Thus, the CPU description can be changed several times and, to check it, just simulate the system again waiting for the successful passage through all instructions or phases which were previously written and were related to validated tests.

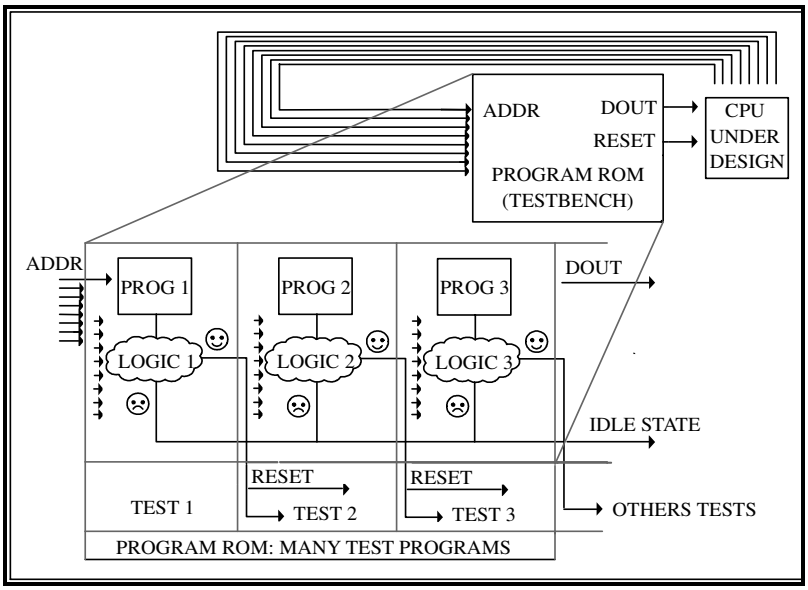

Figure 3. Detailing of the proposed method

3.2 APPLICATION OF THE PROPOSED METHOD

The validation method detailed in section 4 of this work was applied during the development of UPEM2. The datasheet of PIC16F62X, provided by Microchip [PIC16F62X] was used for the verification of operations. We used details of each instruction and we obtained the respective numbers of cycles used for each instruction. The final simplified architecture of UPEM2 is illustrated in figure 4. Both, PORTA and PORTB are bidirectional.

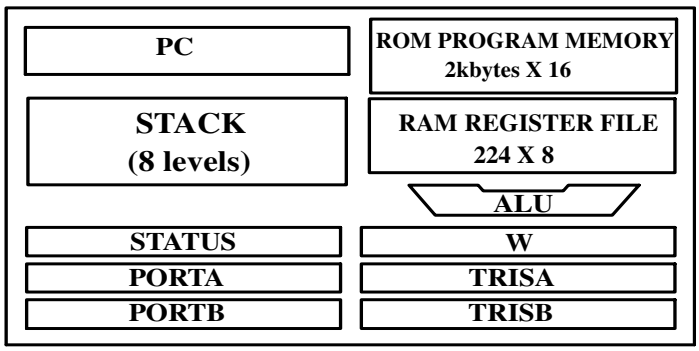

Figure 4. UPEM2 compatible with the PIC16F628 CPU

To apply the method proposed in this work, the first step was to provide the direct access to the internal registers through buses, as illustrated in figure 5.
In figure 5, only the registers considered important have been represented and we have shown them in this work. In this CPU, we monitoring: (i) the current values in the main registers

STATUS, W, PORTA, TRISA, PORTB and TRISB; (ii) lower and upper limits of the cell, (iii) at least one data present in the RAM memory, in this case RAM[0] and RAM[224]; (iv) the current value in PC, which differs from the current value in ADDR; (v) the value of ADDR, which refers to the access to different addressable internal registers and internal RAM memory; and (vi) two temporary registers created for debugging purposes, CYCLES counting machine cycles and CNT_CLK that stores the actual value of clocks spent in the instructions execution.

The second step in the application of the proposed method was the interconnection between the buses listed in figure 5 and the developed testbench, partially illustrated in figure 6.

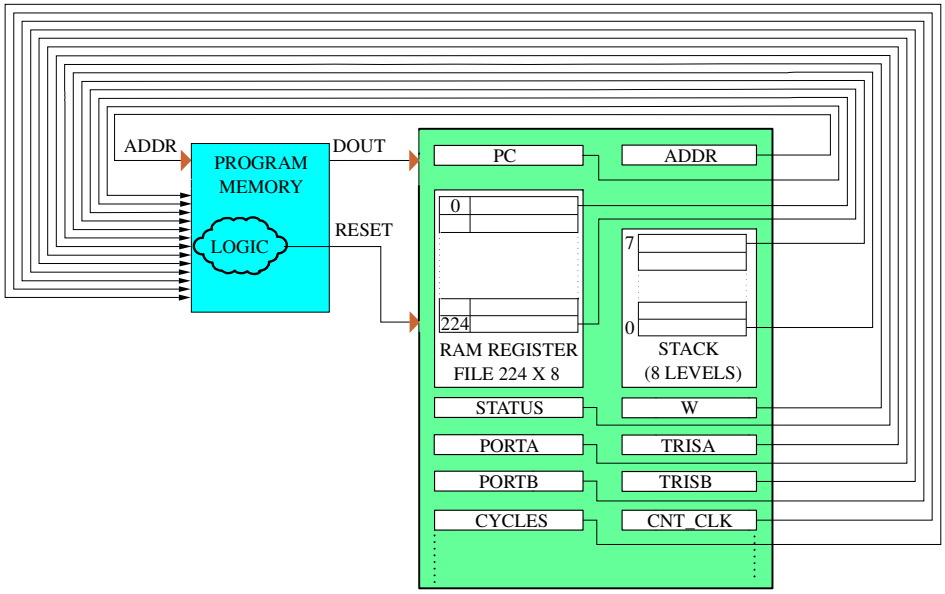

Figure 5. External access to the target registers of UPEM2

In figure 6, the values of the registers are considered as stimuli for each test performed. The tests consist of programs to test each instruction being developed. The results are verified using simple comparators, written manually in code, with respectively comparison values which should previously be known and encoded.

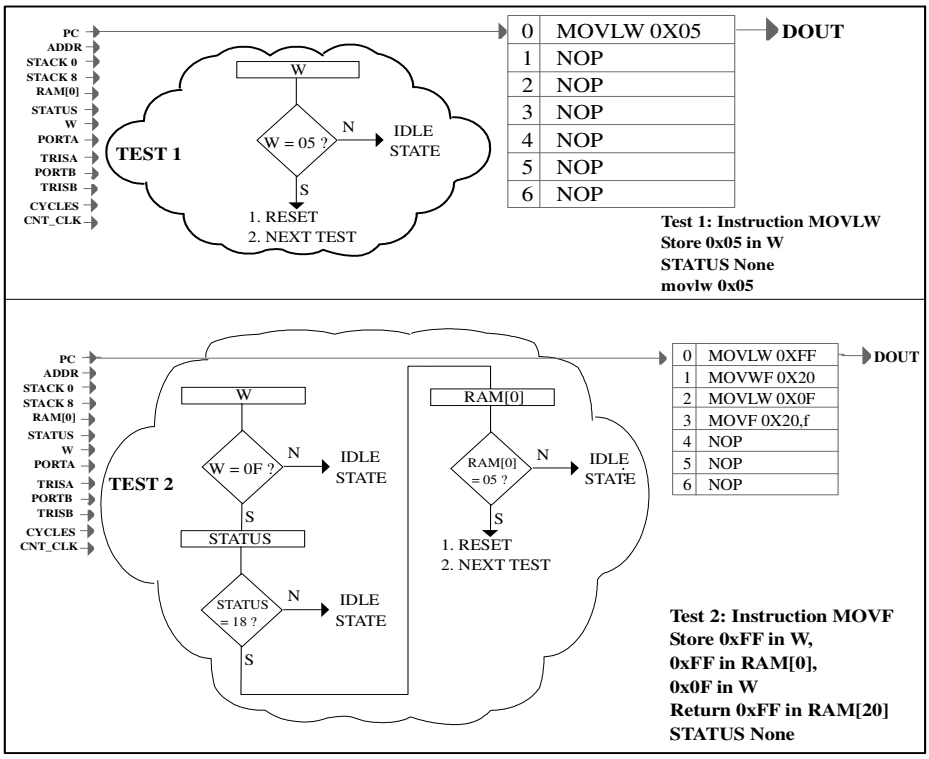

Figure 6. Partial detailing of the testbench

The testbench has been developed taking care that each instruction can be validated in its origin modes or data destiny. For this reason, our case study, the UPEM2´s processor has several addressable registers.

Another care for writing this testbench, we have used only instructions that have already been validated in previous tests which are used for writing new tests. Thus, it started with simple instructions for moving data between registers (movlw, movwf) and then new instructions such as arithmetic and logic (addwl, sublw, andlw, xorwl, etc.) were included.

Thus, the test advances to the next instruction if just the previous test has been successful. The writing of this detailed testbench has allowed us a validation of the UPEM2, and possible changes or enhancements have been revalidated by the testbench itself. This eliminates the visual verification of functionality of each modified or optimized instruction.

When each instruction is validated, a reset pulse is automatically sent to UPEM2, ensuring that all instructions under test will have all their registers “clean”. Thus, there will be no risk of previous instructions influencing the next ones.

Figure 7 illustrates a part of this testbench, where it can be observed that according to the received address in ADDR, the testbench returns instructions and data in “dout”. In the test shown in figure 7, based on the input values RAM [0], the registers W and STATUS, the testbench is able to validate or not the final values in these registers and the instruction set involved.

```
--#########################################################################
-- MOVF
--#########################################################################
elsif count_inst = 131 then
case ADDR is
-------------------------------------------------------------------------------
-- Test MOVF
-- Store 0xFF in W, 0xFF in RAM[0], 0x0F in W and return 0xFF in W
-- STATUS None
-------------------------------------------------------------------------------
  --movlw 0xFF
  --movwf 0x20
  --movlw 0x0F
  --movf  0x20,w
        when conv_std_logic_vector(0,13) => dout <= "110000111111111";
        when conv_std_logic_vector(1,13) => dout <= "000000100100000";
        when conv_std_logic_vector(2,13) => dout <= "110000000011111";
        when conv_std_logic_vector(3,13) => dout <= "001000001000000";

when others =>  dout   <= "000000000000000";
end case;
    if (W = "11111111") and (STATUS = "0011000") and (RAM = "11111111") then
       inst_ok <= '1';
       RST <= '0';
       count_rst <= 0;
    end if;
    if inst_ok = '1' then
      count_rst <= count_rst+1;
      if count_rst = 1 then
         RST <= '1';
         inst_ok <= '0';
         count_inst <= 132;
      end if;
    end if;

elsif count_inst = 132 then
case ADDR is
-------------------------------------------------------------------------------
        -- Test MOVF
.
.
.
```

Figure 7. Part of the testbench developed

If the final, in case of values match the expected values, the testbench generates a reset signal RST and it starts a new test for a new instruction. If the final figures are not as expected, the data throughout the system can be freeze, even with the clock active, leading the CPU to an idle state.

This testbench allowed the validation of about 150 tests in 34 instructions described. Possible modifications and optimizations were easily revalidated, simply going over all the automated tests. The instruction tests with the testbench were considered sufficient, and it was started the final validation stage of UPEM2, with real programs.

In our methodology, we always chosen the main registers to be routed into memory, such as A and B registers, ALU result register, Program Counter, Instruction Register, the top and bottom of the stack, etc. Another specific registers of interest can be easily routed.

Modern processors involve hundreds of possible instructions and each of them might involve multiple possible addressing modes. In this case, it is possible that several instructions have a similar behavior and addressing modes. Some tests can be reutilized, just changing the expected final values.

The method can be portable among processors with significantly different ISAs. The technique is the same, regardless of ISA used. In this case, a new study of the architecture must be performed and a new testbench must be written.

The proposed method includes an extra logic synthesized for validation purposes. This extra logic can be easily removed. This is composed by buses between the program memory and the CPU under development and a few comparators inside program memory.

4. A CASE STUDY - FINAL VALIDATION OF A PROCESSOR

For the final validation on physical level of a processor, in our case we have used the developing of UPEM2, it is a specialized processor for running some peripheral functions of microcontrollers [Penteado 2009, Penteado 2011]. In this paper, we have used two programs that make up games in the PIC16F628 microcontroller which were simulated in three ways: (i) in the ModelSim software [ModelSim, 2012], (ii) simulation Cadence tools and (iii) specific simulators for PIC16F628. After achieving full compatibility between the simulations, the VHDL description was mapped in FPGA Spartan XC2S200E with the software Xilinx ISE 9.1, both from Xilinx [Xilinx, 2012]. The FPGA prototype faithfully reproduced the same behaviour obtained previously by simulation way when we let the execution of the two programs tested. The Cadence simulator indicated success in all the UPEM2´s operations and, subsequently, the project in ASIC was started.

Aiming the full compatibility between UPEM2 and the existing compilers for the PIC16F628, the following steps were completed: (i) to keep the compatibility between the UPEM2 and existing programs, several simulators were studied and simulators with easy interaction have been chosen for step-by-step functions when running the code execution; (ii) The simulators chosen were Feersum miSim DE [Feersum, 2007] and Pic Simulator IDE developed by Vladimir Soso, from OshonsSoft (2010); (iii) with PIC Simulator IDE, the number of cycles and the execution of each isolated instruction were compared to the results obtained in the ModelSim and the UPEM2 testbench and; (iv) real programs that can use the greatest number of instructions from PIC and that run properly with minimal external hardware was obtained;

The actual programs chosen were two games, Tetris an Pong, both developed by Rickard Gunée [Rickard, 2003], which are complex programs that generate composite video signal NTSC.

It was initially conceived the use of the benchmark programs used in [Schmitt, 2004], however, it was chosen the use of games to facilitate the visualization of the physical tests in FPGA. The remainder of this section is divided into a short presentation of the Feersum miSim De software, a comparison of the Tetris game execution when it is running into the UPEM2, using ModelSim tool, and into the PIC Simulator IDE software and, we show physical tests of UPEM2 when it was prototyped in FPGAs.

4.1 DEPURATION OF THE TETRIS GAME WITH FEERSUM MISIM SOFTWARE

The “Feersum miSim De” software [Feersum, 2007] does not provide advanced depuration operation, however, it has been chosen because it provides a plug-in that allows doing simulations using a television set and it receives a composite video signal.

Figure 8 illustrates the main interface of the “Feersum miSim De” program and figure 9 illustrates the result of the Tetris game in the simulator using that plug-in. This plug-in was developed by the authors of the software specifically to emulate the behaviour of the two actual programs (Testis ad Pong) which were selected by us as target test of our UPEM2´s processor.

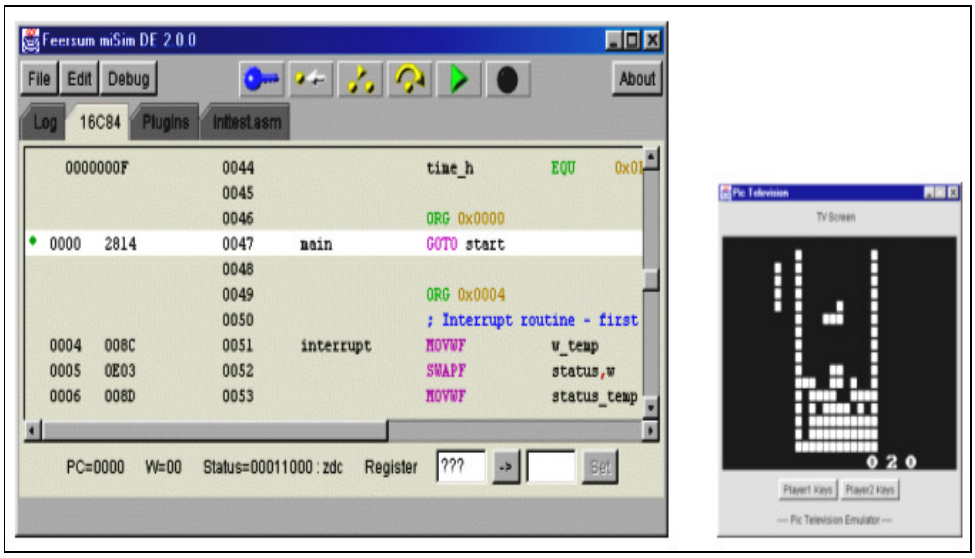

Figure 8. Feersum miSim De simulator and its Tetris game plug-in

With this software installed and properly displaying the images of Tetris game in its plug-in, it was possible to include breakpoints into the program memory to verify the exact moment when the several images of the program are created and displayed in the plug-in.

The UPEM2 was charged with the Tetris program, compiled for PIC16F628, no longer using the testbench. For this, the tetris.hex program available at [Rickard, 2003], has been translated directly into a VHDL description that represents the program memory of PIC, through hex2vhd program, this is a part of the CQPIC project [Morioka, 1999].

Some changes were necessary into the VHDL description after generated by the hex2vhd program to make it adequate to our UPEM2. Thus, it was possible to simulate the execution of the real Tetris program into the ModelSim environment.

4.2 THE PIC SIMULATOR IDE SOFTWARE

The PIC Simulator IDE software [Oshonsoft, 2010] is an excellent tool for debugging and depuration of the operation of programs developed for PIC Microcontrollers. The features considered most important to assist in the development of UPEM2 were: (*i*) its cycle count, the depuration of the next instruction and the last opcode executed; (*ii*) inclusion of breakpoints and; (*i*) the viewing of all the registers values, RAM, ROM, and I/O gates. The Tetris program was charged into the PIC Simulator IDE software and its execution was compared to data obtained when it executes the same program and we used the ModelSim software. The details of the PIC Simulator IDE relevant to this work are highlighted and illustrated in figure 9.

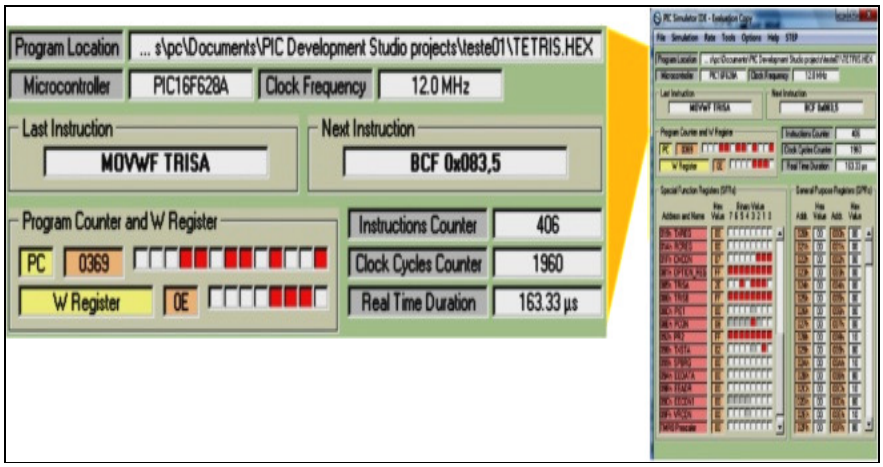

Figure 9. PIC Simulator IDE, by OshonsSoft [Oshonsoft, 2010]

The main purpose of this comparison was to verify the count of cycles spent to execute the programs and compare them to the cycle count of UPEM2 being simulated in the ModelSim software, besides verifying the final behaviour of the actual program tested. In figure 10, we have showed a simulation of UPEM2 in the ModelSim software with the same program that was charged into the PIC Simulator IDE software, as seen in figure 9.

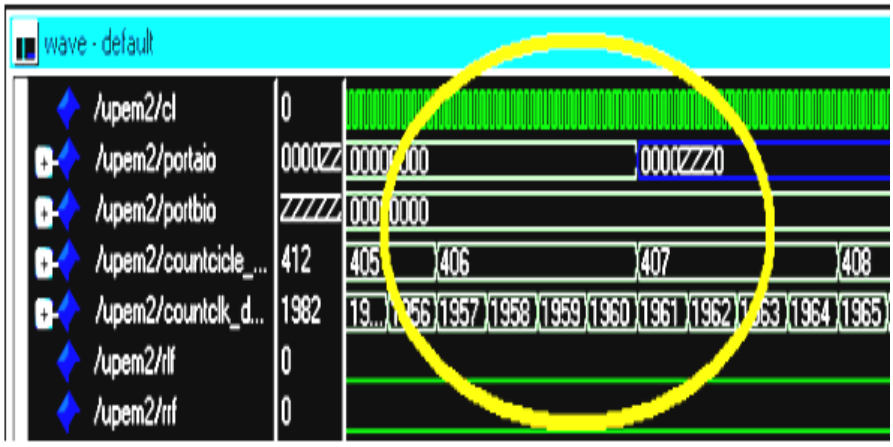

Figure 10. UPEM2 simulation in the ModelSim software

A very positive and encouraging point in the UPEM2 development was that in the highlighted area (see figure 10), the number of clocks and cycles were identical in both simulations, at the moment when the same change occurs in PORTAIO in the Pic Simulator IDE.

In figure 10, Instructions Counter and Clock Cycles Counter, have exactly the same value as “countcicle” and “countclk_d”, in figure 9, which were signals implemented for depuration and tests. In figure 10, the change is viewed in “portaio in synchrony” with the simulation in the PIC Simulator IDE software.

After this, both simulators, ModelSim and PIC Simulator IDE, were left in continuous operation and the results were compared in random points. At all points of our breakpoint the results were identical and then we decided to stop these tests after ModelSim had run for more than 7 hours,

on a Intel Core2Duo 4GB DRAM, reaching a bit more than 50 thousand instructions correctly executed in relation to PIC Simulator IDE.

4.3 PHYSICAL TESTS IN FPGA

In figure 11 a real physical test is illustrated, in which UPEM2 was mapped into the FPGA XC2S200 with the Tetris game, top right corner, and the Pong game, the lower right corner. The FPGA, mapped with the UPEM2, RAM memory and ROM with Tetris game, properly generated the composite video signal and it was possible to execute the game. The video signal being correctly generated allowed us the implementation of the game´s sounds, which are mapped into the PIC´s EEPROM. Following the same sequence presented in section 4.2 of this work, the EEPROM´s information was automatically extracted using the hex2vhd program and some modifications.

The final result was the Tetris game running in UPEM2, which properly generated the composite video signal and sound, making it possible to play. With this positive result, it was decided to repeat the sequence presented in section 4.2 and we execute the program that represents the Pong game, which has also been correctly executed in UPEM2. Other real programs will still be tested for further validation of UPEM2.

The UPEM2 was mapped in the FPGA XC2S200 and the utilization data in FPGA were 480 slices (20%) for the CPU and 2018 slices to CPU, four banks of RAM and ROM with program of Tetris game.

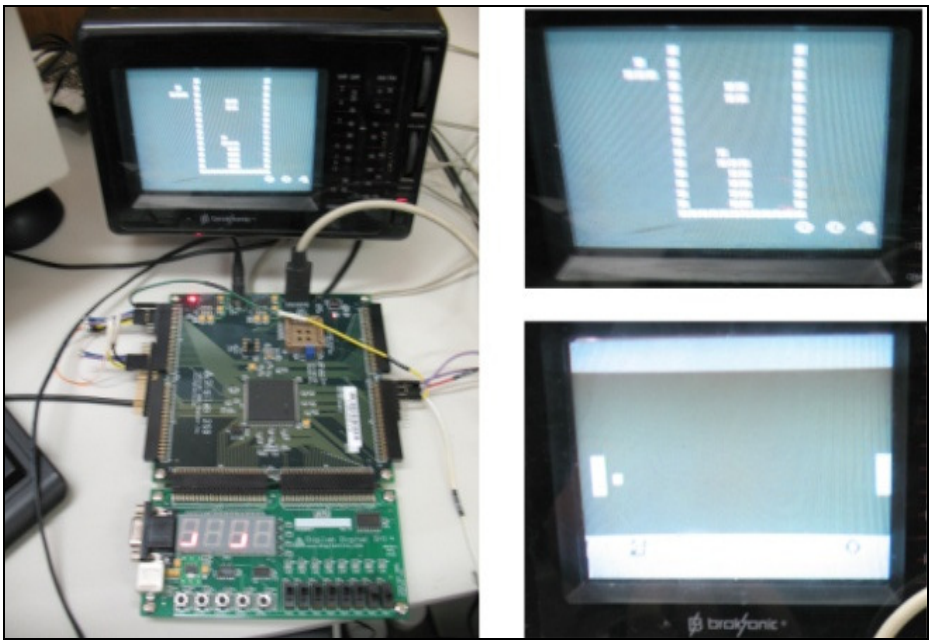

Figure 11. Real test of UPEM2: The Pong and Tetris game on Television

Table 1 shows the final ISA set of UPEM2. The ISA set of the CPU present in the PIC microcontroller has 35 instructions, while the proposed UPEM2 used 32 of these instructions. The interruption control hardware was not implemented, so, the instruction RETFIE (interruption return) can be discarded. There is not either any low-power mode, discarding the SLEEP instruction. Finally, the watchdogtimer peripheral was not implemented and, therefore, the

instruction CLRWDT was discarded. For the correct execution of the real programs tested, two obsolete instructions were implemented in the PIC, the TRISA and TRISB instructions, which assign values to the gate control registers in only one machine cycle. Thus it amounted to 34 instructions.

Table 1: The ISA set of the proposed CPU

1	addlw	10	clrf	19	iorwf	28	sublw
2	addwf	11	clrw	20	movf	29	subwf
3	andlw	12	comf	21	movlw	30	swapf
4	andwf	13	decf	22	movwf	31	xorlw
5	bcf	14	decfsz	23	nop	32	xorwf
6	bsf	15	goto	24	retlw	33	trisa
7	btfsc	16	incf	25	return	34	trisb
8	btfss	17	incfsz	26	rlf		
9	call	18	iorlw	27	rrf		

4.4 ASIC DEVELOPMENT

After the success of the description and implementation in FPGA, we begun the UPEM2 project using the Cadence software [Cadence, 2012], for obtaining an ASIC chip. With the First Encounter tool, the VHDL description of UPEM2 and the memories mapped with the Tetris program were compiled for a Verilog description with the cells from the Foundry adopted.

With UPEM2 in Cadence, the description was simulated in the Cadence SimVision software and the results were compared to the simulation in the ModelSim software.

The results were positive; therefore both simulations behaved the same way. Figures 12 and 13 illustrate both simulations, respectively in ModelSim and Cadence SimVision,

It can be seen at the bottom, highlighted in both figures that the sequence of values at the output gate PORTB is identical: intermittent values of 00h and change to FEh. It indicates that UPEM2 can function properly in silicon.

The physical design in silicon was designed in the 0.18 technology with 6 metals. Considering the highest number of metals, a compact layout was obtained without routing errors. The area used for the whole system, including CPU, RAM, and ROM was approximately 621µm x 517µm, in the 0.18 technology, disregarding the PADS.

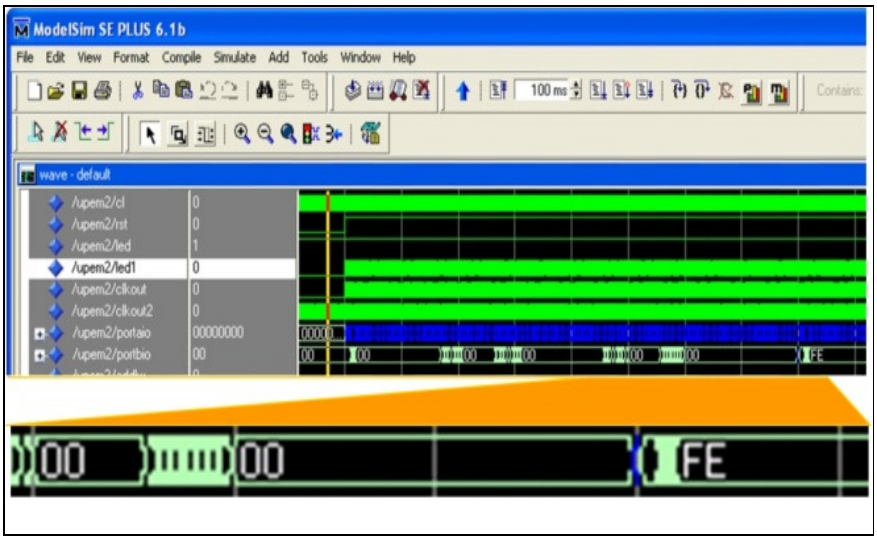

Figure 12. UPEM2 executing the Tetris program in ModelSim

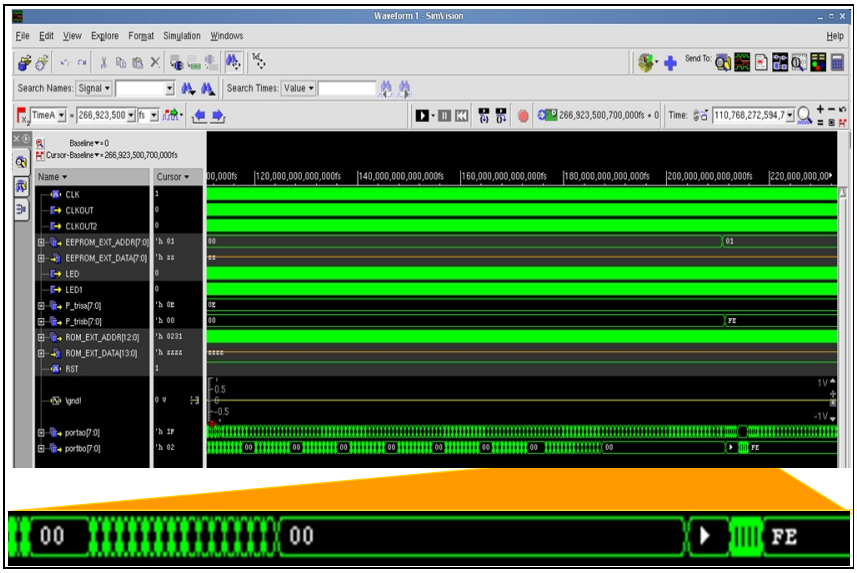

Figure 13. UPEM2 executing the Tetris program in Cadence SimVision

5. CONCLUSIONS

This work showed the proposed method conceived to aids the designer in the encoding and depuration fase of a CPU or dedicated logic. The method consists in some modifications in the program memory and allows an automatic debug of the encoded logic. Also, this work showed the applicability of the proposed method in the UPEM2 design, a CPU developed by authors. The UPEM2 is compatible with the instruction set of the CPU inside in the PIC16F628 microcontroller. All programs tested in the UPEM2 was running successfully in both Modelsim tool and Cadence tools and also in FPGA device.

The validation of our method proposed can be reused for the development of any other processor. For this, simply describe the program memory to be connected to the target processor. Following the same method, it may be possible to adapt any memory and any processor making it possible to get an automatic validation of the system.

The validation method presented and used in this work has provided a successful development of a complex CPU. After completing the tests with the testbench, the CPU executed correctly the programs tested with a few changes in the structure previously validated by the methodology.

According to [Castro, 2008], the manual coding of the stimuli sources can lead to redundant stimuli generation and invalid test vectors. This can be considered a negative point for the use of our methodology. Even so, it is considered that the methodology can be used in the development of any CPU, or even circuits that require stimuli and the results are known in advance.

In our case, the generated testbench has approximately 1000 lines of code manually written, discounting the comments, and the CPU around 800 lines, only the CPU. Theoretically, the same manual work, however, with considerable gains in facility of depuration and verification of results. The evolution from UPEM to UPEM2 has allowed the execution of two real complex problems developed for the PIC16F628. Other real programs must still be tested to ensure compatibility with the compilers and real programs.

According to [Zhenyu, 2002], to ensure the functional verification, a code coverage analysis is necessary to obtain code information not covered by the project verification. Therefore it is believed that UPEM2 still has some error in the execution of its instructions. These errors can be detected and corrected with other tools and verification methods, which still are under studied by us. After the validation in the Cadence software, with the positive results in the SimVision simulator, the digital layout in ASIC has been completed. The complete project is still in progress.

REFERENCES

[1] [Amandeep, 2012] "Microcontroller Based Testing of Digital IP-Core", Singh, Amandeep; Singh, Balwinder, International Journal of VLSI design & Communication Systems VLSIC 2012, DOI: 10.5121/vlsic.2012.3205

[2] [Becker, 2012] A1:A25"MOUSSE: Scaling modelling and verification to complex Heterogeneous Embedded Systems evolution," Becker, M.; Defo, G.B.G.; Fummi, F.; Mueller, W.; Pravadelli, G.; Vinco, S., Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012 , vol., no., pp.296,299, 12-16 March 2012 doi: 10.1109/DATE.2012.6176482

[3] [Cadence, 2012] Ferramentas Cadence para C.I. Available in: http://www.cadence.com/ Date accessed: 22 Oct. 2013.

[4] [Castro, 2008] Automatic generation of random stimuli sources based on Parameter Domains for functional verification, Castro, C.I., et al, in proceedings of the Circuits and Systems Workshop: System-on-Chip - Design, Applications, Integration, and Software, 2008 IEEE 2008 Page(s):1 - 4

[5] [Cha, 2012] "Verification of massive advanced node SoCs," DaeSeo Cha; HyunWoo Koh; NamPhil Jo; Kim, J.B.; Byeong Min; Kothandapani, K.; Oddone, R.; Sherer, A.,SoC Design Conference (ISOCC), 2012 International , vol., no., pp.395,397, 4-7 Nov. 2012 doi: 10.1109/ISOCC.2012.6407124

[6] [Feersum,2007] Feersum miSimDE, Feersum Technology. Available at: http://www.feertech.com/ Date accessed: 22 Oct. 2013.

[7] [Hu, 1994] A methodology for design verification, Hu, E.; Yeh, B.; Chan, T. in proceedings of the ASIC Conference and Exhibit,., Seventh Annual IEEE International 19-23 Sept. 1994 Page(s):236 – 239

[8] [Kwanghyun, 2008] Reusable platform design methodology for SoC integration and verification, Kwanghyun Cho, et al, in proceedings of the SoC Design Conference, 2008 IEEE. ISOCC '08. International Volume 01, Page(s):78 - 81

[9] [Meng, 2010] "Functional verification of external memory interface IP core based on restricted random testbench," Qingdong Meng; Zhaolin Li; Fang Wang, Computer Engineering and Technology (ICCET), 2010 2nd International Conference on , vol.7, no., pp.V7-253,V7-257, 16-18 April 2010 doi: 10.1109/ICCET.2010.5485235

[10] [ModelSim, 2012] ModelSim, Mentor Grafics, Available at: http://www.model.com/ Date accessed: 22 Oct. 2013.

[11] [Morioka, 1999] Morioka, Sumio, project CQPIC, 1999. Available at: http://www002.upp.so-net.ne.jp/morioka/papers.html. Date accessed: 22 Oct. 2013.

[12] [Oshon, 2010] Pic Simulator IDE, Oshon Software Available at: http://www.oshonsoft.com/ Date accessed: 22 Oct. 2013.

[13] [Penteado 2011] Penteado, C., Kofuji, S., Moreno, E.. A Flexible and Parameterized Architecture for Multicore Microcontroller. Journal of Computers, Vol 6, No 11. Available at: http://ojs.academypublisher.com/index.php/jcp/article/view/jcp061122772284. Date accessed: 22 Oct. 2013.

[14] [Penteado, 2009] Um Processador Específico para Periféricos de Microcontroladores, Penteado, C.G., Moreno, E.D. in IEEE Latin American Transactions, Vol. 7, Issue 2, June , Issue 2, June 2009, ISSN: 1548-0992

[15] [PIC16F62X] PIC16F62X FLASH-Based 8-Bit CMOS Microcontrollers Novembro /1999. Available at: http://www.microchip.com/ Date accessed: 03 Jul. 2012.

[16] [Rickard, 2003] Rickard Gunée, Rickard's electronic projects page Available at: http://www.rickard.gunee.com/projects/ Date accessed: 22 Oct. 2013.

[17] [Schmitt, 2004] Verification of a microcontroller IP core for system-on-a-chip designs using low-cost prototyping environments, Schmitt, S.; Rosenstiel, W. in proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2004. IEEE Volume 3, 16-20 Page(s):96 - 101

[18] [Wrona, 2011] "An example of DISPLAY-CTRL IP Component verification in SCE-MI based emulation platform," W. Wrona, et al., presented at the Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2011.

[19] [Xilinx, 2012] Xilinx ISE Foundation, Available at http://www.xilinx.com/ Date accessed: 22 Oct. 2013.

[20] [Zhaohui, 2012] "Practical and efficient SOC verification flow by reusing IP testcase and testbench," Hu Zhaohui; Pierres, A.; Hu Shiqing; Chen Fang; Royannez, P.; Eng Pek See; Yean Ling Hoon, SoC Design Conference (ISOCC), 2012 International , vol., no., pp.175,178, 4-7 Nov. 2012 doi: 10.1109/ISOCC.2012.6407068

[21] [Zhenyu, 2002] Functional verification methodology of a 32-bit RISC microprocessor, Zhenyu Gu, et. al., in proceedings of the Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on Volume 2, 29 June-1 July 2002 Page(s):1454 - 1457