FPGA Implementation of a 64-Bit RISC Processor Using VHDL

Imran Mohammad¹, Ramananjaneyulu K²
¹²QIS College of Engineering, Ongole-523001, A.P., INDIA.

Abstract: In this paper, the Field Programmable Gate Array (FPGA) based 64-bit RISC processor with built-in-self test (BIST) feature implemented using VHDL and was, in turn, verified on Xilinx ISE simulator. The VHDL code supports FPGA, System-On-Chip (SoC), and Spartan 3E kit. This paper also presents the architecture, data path and instruction set (IS) of the RISC processor. The 64-bit processors, on the other hand, can address enormous amounts of memory up to 16 Exabyte’s. The proposed design can find its applications in high configured robotic work-stations such as, portable pong gaming kits, smart phones, ATMs.

Keywords: FPGA, RISC, BIST, VHDL, SoC, IS, Exabyte.

1 Introduction

In today’s technology, RISC Processors are playing a prominent and the RISC with BIST feature is one of the more dominant test pattern which can provides, in system testing of the Circuit-Under-Test (CUT). BIST design is becoming more complicated with the increase of IC size.

Though the RISC has less instruction set, as its the bit processing size increases then the test pattern becomes complicated and the structural faults are maintained high. And BIST is highly reliable, low cost. BIST is beneficial in many ways: First, it can reduce dependency on external Automatic Test Equipment (ATE). In addition, BIST can provide at speed, in system testing of the Circuit-Under-Test (CUT).

This is crucial to the quality component of testing. Also, BIST can overcome pin limitations due to packaging, make efficient use of available extra chip area, and provide more detailed information about the faults present. In our thesis, a 64 bit RISC processor with limited functionality is designed with an architecture that supports BIST.

The proposed design is done by implementing MICA (Minimal Instruction Set Computer Architecture) architecture. The design is implemented on Xilinx ISE 10.1 Simulator and programmed by using VHDL. The programmed code is supports FPGA Spartan-3E Kit. However, contemporary CAD tools allow the designer of hardwired control units almost as easy as micro programmed ones. This enables the single cycle rule to be enforced, while reducing transistor count.

In order to facilitate the implementation of most instruction as register-to-register operations, ALU is analyzed and an exhaustive set of test patterns is developed.

2 Architectural Design - Implementation

In this session, Architecture, Data path, and the instruction set are described. The FPGA based RISC Processor has its architecture with BIST, control and timing module is a Hardware module. The ALU is divided into two parts as: The Operational Architecture (OA) and the Testing Architecture (TA).

Operational Architecture (OA) does the actual operation of the ALU. It has five units, 4-bit Carry Look Ahead adder (CLA), and a 4-bit AND, OR, XOR and INVERTER gates. There is a PreCLA to prepare the inputs based on the arithmetic operation to be done. There is a MUX which uses the select pins to select one of the results from the above five units.

Testing Architecture (TA), which comes into play only during testing, has a ROM which has the discovered test patterns stored in. There is an address decoder to select which of the test patterns will be applied. There is a TestMUX, which depending on the value on the TestMode pin will present the test pattern or the actual inputs to be operated upon, to the Operation Architecture.
Table 2.1: 33 Instruction Set (IS) for 64 bit RISC Processor

| INSTRUCTIONS | DESCRIPTION |
|--------------|-------------|
| ADD          | ADD dest. Src: Adds “src” to “dest” and replacing the original contents of “destination”. Both operands are binary. |
| IAND         | ADD dest. Src: Performs a logical AND of the two operands replacing the destination with result. |
| SKIPZ        | Skipz, Skips one clock cycle when data entered is zero. |
| LTR          | LTR src; Loads the current task register with the value specified in “src”. |
| LSL          | LSL dest. Src: Loads the segment limit of a selector into the destination register if the selector is valid and visible at the current privilege level. If loading is successful the Zero Flag is set, otherwise it is cleared. |
| INOT         | NOT dest; Inverts the bits of the “dest” operand formatting the 1s complement. |
| NEG          | NEG dest; Subtracts the destination from 0 and saves the 2s complement of “dest” back into “dest”. |

Figure 2.1: 64-bit RISC Processor Architecture.

POP – Pop Word off Stack

POP dest; Transfers word at the current stack top (SS:SP) to the destination then increments SP by two point to the new stack top. CS is not a valid destination.

PUSH – Push Word onto Stack

PUSH src

PUSH immed (80188+only): Decrements SP by the size of the operand (two or four, byte values are sign extended) and transfers one word from source to the top (SS:SP).

SETS – Set

SETS dest; Sets the byte in the operand to 1 if the Sign Flag is set, otherwise sets the operand to 0.

ROL – Rotate Left

ROL dest, count ;Rotates the bits in the destination to the left count times with all data pushed out the left side re-entering on the right. The Carry Flag will contain the value of the last bit rotated out.

ROR – Rotate Right

ROR dest, count; Rotates the bits in the destination to the right “count” times with all data pushed out the right side re-entering on the left. The Carry Flag will contain the values of the last bit rotated out.

SAL / SHL – Shift Arithmetic Left / Shift

SAL dest, count

SHL dest, count; Shifts the destination left by “count” bits.
| Logical                  | Operation                                      |
|-------------------------|-----------------------------------------------|
| SAR – Shift Arithmetic Right | SAR dest, count; the destination right by “count” bits with the current sign bits replicated in the leftmost bit. The carry flag contains the last bit shifted out. |
| SETC – Set if Carry (386+) | SETC dest; Sets the byte in the operand to 1 if the carry flag is set, otherwise sets the operand to 0. |
| SETO – Set if Overflow | SETO dest; Sets the byte in the operand to 1 if the overflow flag is set, otherwise sets the operand to 0. |
| STC – Set Carry | STC; Sets the Carry Flag to 1. |
| STI – Set Interrupt Flag (Enable Interrupt) | STI; Sets the Interrupt Flag to 1, which enables recognition of all hardware, interrupts. If an interrupt is generated by a hardware device, an END of interrupt (EOI) must also be issued to enable other hardware interrupts of the same or lower priority. |
| SUB – Subtract | SUB dest.src; The source is subtracted from the destination and the result is stored in the destination. |
| VERR – Verify Read (286+protected) | VERR src; Verifies the specified segment selector is valid and is readable at the current privilege level. If the segment is readable, the Zero Flag is set, otherwise it is cleared. |
| CLC – Clear Carry | CLC; Clears the Carry Flag. |
| IXOR – Exclusive OR | XOR dest, src; Performs a bitwise exclusive OR of the operands and returns the results in the destination. |
| INAND – Logical NAND | Inand dest, src; Performs a bitwise logical NAND of the two operands replacing the destination with the result. |
| ADDI – Add Immediate | ADD dest, src; Adds “src” to “dest” and replacing the original contents of “dest”. Both operands are binary. It performs immediate addition i.e., takes half clock cycle than in add |

The architecture and data path for the proposed design are shown Fig. 2.1 and 2.2, respectively. Table 2.2 gives the salient technical features of the proposed processor. Table 2.1 provides detailed description of entire 33 instruction set.

Table 2.2: Salient Technical Features of RISC processor

| Features of RISC processor |   |
|----------------------------|---|
| Architecture               | MICA |
| Instructions               | 33bit |
| Instruction Register       | 32 bit |
| Address Counter            | 32 bit |
| Data memory                | 64 bit |
| Data bus                   | 64 bit |
| Address bus                | 32 bit |
3 Synthesis Report

| Logic Utilization     | Used | Available | Utilization | Note(s) |
|-----------------------|------|-----------|-------------|---------|
| Total Number Slice Registers | 209  | 9,112     | 2%          |         |
| Number used as Flip Flops | 55   |           |             |         |
| Number used as Latches | 154  |           |             |         |
| Number of Input LUTs  | 1,780| 9,112     | 12%         |         |

Logic Distribution:

- Number of occupied slices: 654, 4,956, 14%
- Number of slices containing only related logic: 654, 954, 100%
- Number of slices containing unrelated logic: 0, 954, 0%

| Total Number of 4 input LUTs | 1,205 | 9,112 | 12% |
|------------------------------|-------|-------|-----|
| Number used in logic         | 1,180 |       |     |
| Number used as a route thru  | 7     |       |     |
| Number of bonded FFs         | 66    | 122   | 25% |
| Number of DFGs               | 4     | 24    | 16% |

4 Simulation Results

Figure 3.1: Synthesis report.

Figure 3.2: Routing of RISC Processor.

Figure 3.3: Floor Planning for RISC Processor.

Figure 4.1: Simulation of top module with central processing unit inputs.

Figure 4.2: Simulation results of general purpose register.
The above results show the simulation of 64 bit RISC Processor. It has clock and reset signal as the input for the top module shown in 4.1.1. It consists of a 16 general purpose register of 64 bit size which is shown in 4.1.2. And the operation of arithmetic logic unit with program counter is shown in 4.1.3. The instruction set having 33 instructions and the memory module shown in figure 4.1.4 and the total processor result is obtained by combining all the results which is verified using Xilinx ISE simulator.

5 Applications

The proposed design can find its applications in automation, high configured robotic work-stations such as, portable pong gaming kits, smart phones, Vender Machines, ATMs, bottling plant, etc.

Bottles start filling from the right side and boxes start to move from the left side. Here four tracks of bottles are used simultaneously therefore packing is made of four bottles. When bottle reaches to the fourth position, box moves to the first position. After that, bottle is dropped in the box and hence, box moves one position ahead. In this way, when box is at the fifth position, signal ‘lb’ is set to ‘1’ indicating to lift the box.

5.1 Flow Chart for bottling Plant application:

**Figure 3: Flow chart for bottling plant**

5.2 Algorithm for bottling Plant application:

```plaintext
a=1, b=7, weight=0
loop a till a = 8
    a = a+1;
    wait for 15 secs
    If (a = 4) then
        drop bottle in box
        a = a-1;
    End If;
    If (a = 5) then
        report error in bottle machine
    End If;
End loop;
loop b till b = 5
    b = b+1; wait till weight = 1;
    If (b = 5) then
        given signal to left box;
        b = b-1;
    End If;
    If (b = 6) then
        report error in packing machine
    End if;
```

Figure 4.3: Simulation results for the ALU outputs.

Figure 4.4: Simulation results of 33 instructions and memory module.
End loop;

6 Conclusions

The 64-bit RISC Processor with 33 instructions set and MICA (Minimal Instruction Set Computer Architecture) architecture has been designed and it can be implemented on FPGA. The design is verified on Xilinx ISE 10.1i simulator and programmed by using VHDL. The programmed code can be implemented on FPGA Spartan-3E Kit. ALU is analyzed and an exhaustive set of test patterns is developed. Future work will be added by increasing the number of instructions and make a pipelined design with less clock cycles per instruction and more improvement can be added in the future work.

References

[1] Samuel O. Aletan, “An Overview of RISC Architecture”, Proc. Symposium on Applied Computing, 1992, pp.11-12.
[2] Design and Implementation of a 64-bit RISC Processor using VHDL, 2009 IEEE.
[3] Design and Implementation of a 64-bit RISC Processor using System On Chip (SOC), 2011, IJCSCN, 360-370.
[4] Dal Poz, Marco Antonio Simon, Cobo, Jose Edinson Aedo, Van Noije, Wilhelmus Adrianus Maria, Zuffo, Marcelo Knorich, “Simple Risc microprocessor Core designed For digital set top box applications”, Proceedings of the International Conference on Application Specific Systems, Architectures And Processors, 2000, p 3544.
[5] Brunelli Claudio, Cinelli Federico, Rossi Davide, Nurmi Jari, “A VHDL model And implementation of a coarse grain reconfigurable coprocessor for a RISC core”, 2nd Conference on Ph.D. Research in Microelectronics and Electronics Proceedings, PRIME, 2006, p 229232.
[6] Rainer Ohlendorf, Thomas Wild, Michael Meitinger, Hofm Rauchfuss, Andreas Herkersdorf, “Simulated and measured performance evaluation of RISC based SoC Platforms in network processing applications”, Journal of Systems Architecture 53 (2007) 703–718.
[7] Luker, Jarrod D., Prasad, Vinod B., “RISC system design in an FPGA”, MWSCAS 2001, v2, 2001, p532536.
[8] Jiang, Hongtu, “FPGA implementation of controller data path pair in custom Image Processor design”, IEEE International Symposium on Circuits and Systems Proceedings,2004, p V141V144.
[9] Lou Dongjun, Yuan Jingkun, Li Daguang, Jacobs Chris, “Datapath verification With SystemC reference model”, ASICON 2005, 6th International Conference on ASIC, 2005, Proceedings, v 2, p 906909.
[10] K. Vlachos, T. Orphanoudakis, Y. Papaeftathiou, N. Nikolaou, D. Pnevmatikatos, G. Konstantoulakis, J.A. Sanchez P., “Design and performance evaluation of a Programmable Packet Processing Engine (PPE) suitable for high speed network Processors units”, Microprocessors and Microsystems 31, 2007, p 188–199.
[11] John L. Hennessy, and David A. Patterson, “Computer Architecture A Quantitative Approach”, 4th Edition; 2006.
[12] Vincent P. Heuring, and Harry F. Jordan, “Computer Systems Design and Architecture”, 2nd Edition, 2003.
[13] Wayne Wolf, FPGA Based System Design, Prentice Hall, 2005.