FPGA Implementation of an 8-bit Simple Processor

E. Ayeh, K. Agbedanu, Y. Morita, O. Adamo, and P. Guturu
[1]Department of Electrical Engineering
University of North Texas, Denton, TX. 76207
USA

Abstract*— Since its "birth" in 1971, embedded microprocessor has been widely used as a tool for technological innovations and cost reduction. Its speed and programmability are the main characteristics determining its performance. Therefore, for a design to be competitive, its processor has to fit the following characteristics: relatively inexpensive, flexible, adaptable, fast, and reconfigurable. A solution to this is the use of Field-programmable gate arrays (FPGA) as design tool. This paper describes the realization of an 8-bit FPGA based simple processor. Our system was implemented on the Xilinx Spartan 3 xc3s200FT256 using ISE foundation 8.1 and VHDL. 132 (6%) of the slices, 351 Bels were used. A maximum frequency of 95.364 MHz was reached with a minimum period of 10.486 ns.

I. INTRODUCTION

Nowadays, embedded microprocessors are used in a variety of electronic products such as personal computers, cell phones, and robots. They consist of thousands of electronic components and use a collection of machine instructions to perform not only mathematical operations but also to move information from one memory location to another.

The processor or Central Processing Unit (CPU) is the heart of the computer. It determines in a large part, how fast the computer will be and what capabilities the machine will have. As designers we have a choice between using digital signal processors (DSPs), FPGAs, or application specific integrated circuits (ASICs) in our designing process. In this paper, we implemented an FPGA-based processor.

ASICs customized for a particular use are very expensive even though they provide the highest performance. DSP based designs, on the other hand, are cost efficient and low in power consumption and heat-emission. However, they only provide a limited speed for data processing because using special memory architectures that are able to fetch multiple data and/or instructions at the same time, they are susceptible to arithmetic saturation.

FPGAs are usually slower than ASICs but have the advantage of shorter time to market, ability to be re-programmed in the field for errors correction and upgrades, flexibility, and low-cost. Therefore, they combine many advantages of ASICs and DSPs. The use of hardware description languages (HDLs) allows FPGAs to be more suitable for different types of designs where errors and components failures can be limited.

Our FPGA-based 8-bit processor designed using the VHDL language mainly consists of 4 4-bit registers, a 16-word memory with 8-bit words, a control unit, and an arithmetic logic unit (ALU).

Due to the exponential increase of technologies; designers are faced with problems that require the advent of systems that can be fast, flexible, and mainly re-programmable. FPGAs because of their advantage of real-time in-circuit reconfigurability make the FPGA based microprocessor flexible, programmable, and reliable. They also facilitate the prototyping of complex electronic designs.

II. RELATED WORK

Due to the cost of ASIC design and the speed of DSPs processors that is involved in the development of flexible devices, many engineers are now turning to FPGA. As a result, many significant works have been based on FPGA. However, not all of them are related to the actual design of a processor.

In [1], the implementation of a systolic array architecture in hardware using FPGAs for processing compressed binary images without decompressing them is proposed. In [2], the author talked about how a high performance of embedded systems and real time computing can be obtained through the use of FPGA technology. Similar to the previous writer the authors in [3] dealt with the reliability of FPGA platforms in the design of embedded systems. In fact, a soft processor which is a programmable instruction processor and a methodology for measuring their area, performance, and power was proposed as a solution. The authors in [4] described a system that can enable FPGAs to generate machine code for various CPUs. The system will perform a conversion of an intermediate to a CPU's native code for applications in virtual machines such as the Java Virtual Machine. In [5], the implementation of a floating point processor array for a high-precision dot product is described. The authors mentioned the cost and configurability benefits that can be obtained through the use of FPGA. The authors in [6] used FPGA as a practical experimentation and verification platform for emulation of hardware without the non-recurring cost engineering (NRE) costs of ASIC hardware. The use of FPGA as a design tool can provide not only cost but also performance benefits. In [7], the accuracy, rapidity, (re)configurability, transparency and the cost of FPGA as computer simulator were discussed. In this paper, we present the FPGA implementation of a simple

978-1-4244-2077-3/08/$25.00 ©2008 IEEE.

processor with limited resources. The processor was implemented on the Xilinx Spartan 3 xc3s200FT256 using ISE foundation 8.1 and VHDL.

III. INSTRUCTION SET

Our processor is designed to input an 8-bit set of instructions, which will be used to determine most of the operations needed as well as their addresses. The first (higher) four bits are generally used to determine an operation while the last (lower) four bits are used to determine an address. A memory unit is used to store the instructions and registers are used to store the addresses. The three different types of instructions used are: Register Instructions, Branch Instructions, and Halt & I/O Instructions. The register instructions are used to perform arithmetic operations and add them to and from the data registers. The instruction format for register instruction is shown in figure 1 and it consists of the OP, CC, SRC, and DST fields.

In Table 1, an input of 00 will extract information from a source register; add it to the carry bit (C_I), and place the result in the destination register (DST). In the same fashion, an input of 01 will insert the information from the source, CC, and destination addresses back into the destination address. Finally, the input of 10 will delete the information in the Source and Carry addresses from the destination address and add the result to the destination.

The value of Carry In can be determined by the user. However, predetermined values exist depending on CC. In fact, as shown by Table 2, if CC is 00, C_I will always be 0. Likewise, C_I will always be 1 with a CC value of 01. If the user inputs a value of 0 for a CC of 10, the outcome will be 0. If the input value is 0 or 1 for a CC of 11, the outcome will be 1 and 0 respectively.

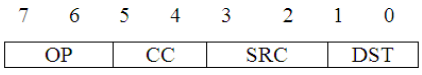

Figure 1: Register Instructions

OP	Function
00	(SRC) + C_I => (DST)
01	(SRC) + (DST) + C_I => (DST)
10	(DST) - (SRC) - C_I => (DST)

Table 1: Basic Register Instructions

CC	C_I
(R0) 00	0
(R1) 01	1
(R2) 10	C
(R3) 11	C'

Table 2: Carry Bit usage based on CC bits

Branch Instructions are used to jump to a particular memory address specified by the last four instruction bits. The operation will only be carried out whenever the first two bits are 11 and the carry is positive as illustrated in Figure 2. The following three conditions result in a jump of address: CC = 01, CC = 10 and C_I = 1, CC = 11 and C_I = 0. A CC address of 00 will never be reached, since the conditions will not always jump, leaving room for the last set of instructions to be carried out.

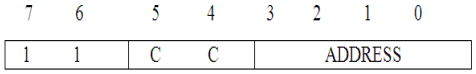

Figure 2: Branch Instructions

The Halt & I/O Instructions are carried out only with an input of 1100 in the first four bits. When the third bit is 1, the processor will enable loading from a fixed register to the data register specified. When the fourth bit is 1, the processor will display the information located in the destination register on the seven-segment display. When both L and H are, an error will occur and no changes will be made. (Figure 3)

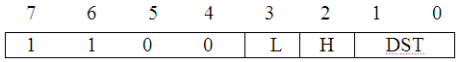

Figure 3: Halt & I/O Instructions

IV. MICROPROCESSOR DESIGN

Our FPGA-based processor consists of several units. These units can be classified in two different categories. We have the main components which are: a control unit, an Arithmetic Logic Unit (ALU), and the memory. The sub-components are: a program counter, an instruction register, a 4-bit data register, multiplexers (MUX), demultiplexers (DMUX), NAND gates, and D-latches. The figure below represents the block diagram of our processor.

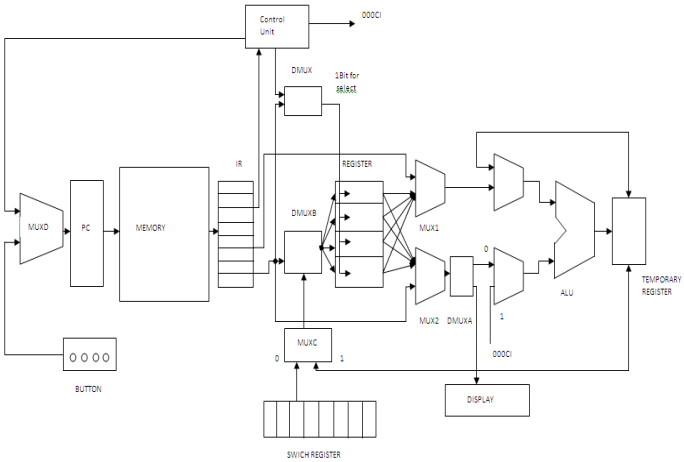

Figure 4: Complete processor map

Assuming that we have the instruction "00000100" we first used the switch registers to input the instruction, and then stored it in the first memory location. Once the start button is set on, the same instruction goes to the instruction register which is the temporary storage for the memory and it separates the instruction into four fields: OP code IR[7-6]"00", CC IR[5-4]"00", SRC IR[3-2]"01"and DST IR[0-1]"00". OP and CC goes to the control unit which coordinate the different units of the processor. For example, the program counter indicates where the computer is in its instruction sequence and is incremented by the control unit. The 2-bit value of the SRC segment which represents the address of the register containing the actual value to be calculated goes to the MUX 1 as shown

in figure 4. The MUX then chooses a register and pick up the value and then send it to MUXA and then to the ALU. The instruction $(\mathrm{SRC})+\mathrm{C}_{\mathrm{I}} \rightarrow (\mathrm{DST})$ sets C_{I} to"0000" which contains the value of CC, and send it to MUXB, and then to ALU. Finally, the ALU processes the addition and outputs the result to DMUXB. Furthermore, the 2-bit value of the DST goes to the DMUXB as "select", and the DMUXB transfers the ALU's output value to the destination register.

A. *Control Unit*

Our control unit is modeled as finite state machine (FSM) and has 6 states. In the initial state, nothing happen until the start button has been pushed. Once pushed, the control unit moves to the zero state. In that state, an instruction is fetched from the memory and put into the instruction register $(IR \leftarrow Memory[PC])$. The state is then upgraded from zero to one. The instruction is then checked by the control unit. If a Register Instruction is determined, the PC is incremented and the actual value in the source register along with the carry is sent to the ALU $(ALUout = (SRC) + C_I)$. The state is moved from state 1 to 2. If an Input instruction is detected, the PC is incremented and the input bits are loaded into the register pointed to by the DST bits $(\mathrm{Re}\, g[DST] \leftarrow SW\ \mathrm{Re}\, g)$ and then the state goes back to state 0. If the instruction is Halt & Display instruction, the PC is incremented; the machine halts, and displays the contents of DST register $(Display \leftarrow \mathrm{Re}\, g[DST])$. As a result, the state is once again reset. If the instruction is Unconditional Branch or Branch on Carry, the PC is incremented and the address bits in the instruction register goes to the pc $(PC \leftarrow IR[0-3])$. The state then goes to zero. The second state is only reached in the occurrence of a register instruction. Depending on the op-code, the control unit works differently. If the Op-code is "00", the output of the ALU goes to the destination register $(DST \leftarrow ALUout)$ and the state to zero. On the other hand, if the op code is "01" or "10", then the output of the ALU goes back to MUX A, and the value in the destination register also goes to the ALU which then adds them for "01" $(ALUout \leftarrow ALUout + DST)$ and subtracts for "10" $(ALUout \leftarrow DST - ALUout)$. The control unit then goes to the next state. The third state serves as delay for implementation purpose. Finally, in the fourth state the output of the ALU goes to the destination register $(DST \leftarrow ALUout)$ and the state is reset. The state diagram of our controller is shown in Figure 5.

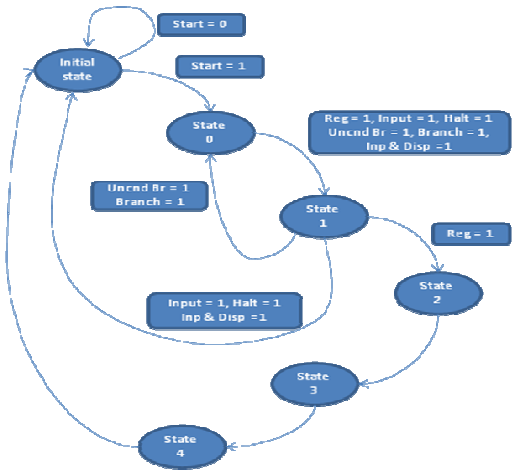

Figure 5: Control Unit state diagram

B. *Arithmetic Logic Unit (ALU)*

An ALU is a digital circuit that calculates arithmetic operations like: addition, subtraction, shifting and exclusive or. For our case, the ALU will only be taking care of addition and subtraction. It is composed of a 4-bit full adder and a unit used to obtain the 2'complement of numbers for simpler subtractions.

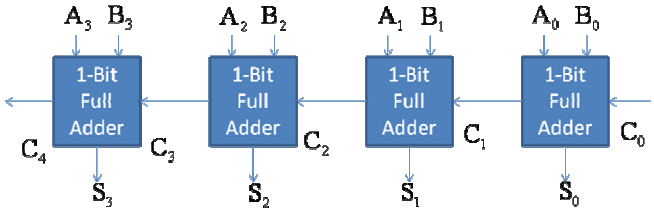

Figure 6: ALU structure

C. *Memory*

Our memory is a 16-word 8-bit. Each bit of the memory (SRAM cell) in a static RAM has an AND gate, a D-latch, and a tri-state buffer as shown in Figure 7. When a cell's SEL_L input is asserted, the stored data is placed on the cell's output, which is connected to a bit line. When both SEL_L and WR_L are asserted, the latch is open and a new data bit is stored. SRAM cells are combined in an array with additional control logic to form a complete statistic RAM for a 16X8 SRAM. As in a simple ROM, a decoder on the address lines selects a specific row of SRAM to be accessed at any time. Once the row is decided, 8 bits data is written in the 8 SRAM or road 8-bit data to next component. Figure 8 illustrates the whole RAM.

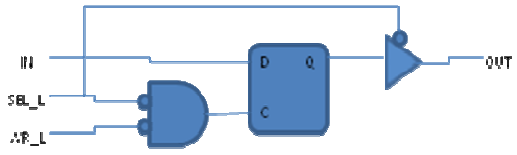

Figure 7: Structure of an SRAM cell

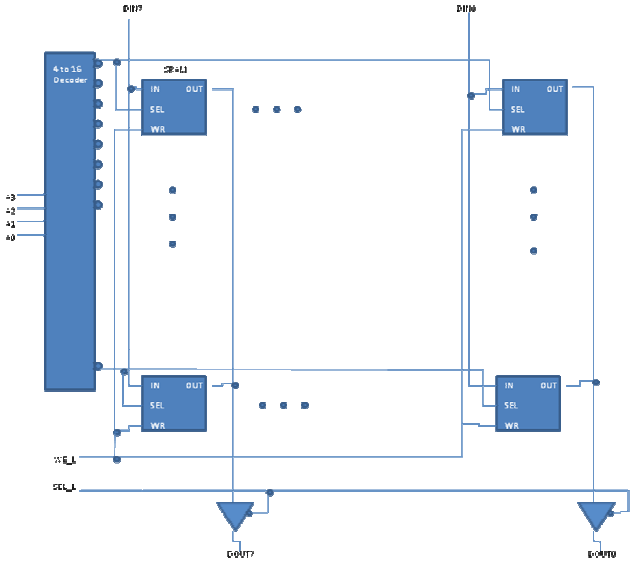

Figure 8: Memory structure

V. IMPLEMENTATION & RESULTS

To ensure the quality of our design, all the units in the processor were designed and tested separately. Once the functionalities of each unit were verified they were combined together as a block and once again tested. The following images (captures of the testing) are the results of the implementation of the main units.

Control Unit: The main function of the control unit is to direct the operations of the processor. The inputs of the control unit are taken from the instruction registers which are: **OP, CC**, **L** and **H.** Depending on these inputs, the output signal will be generated.

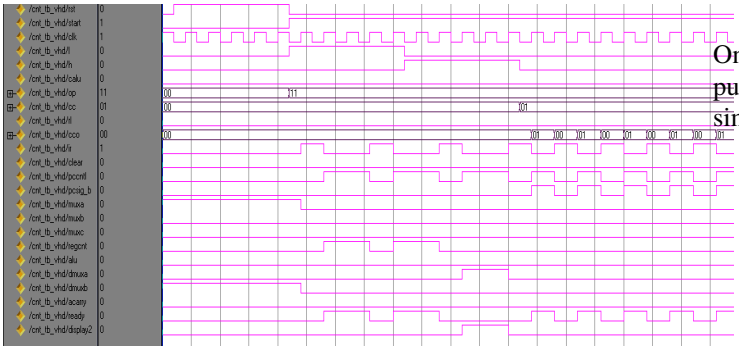

Figure 9: Control Unit simulation

ALU: Results from binary addition and subtraction tested on our ALU unit are as followed:

1010-0011=0111
(10-3=7)

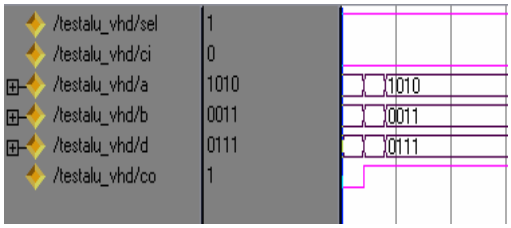

Figure 10: ALU subtraction operation

1011+0001=1100
(11 + 1= 12)

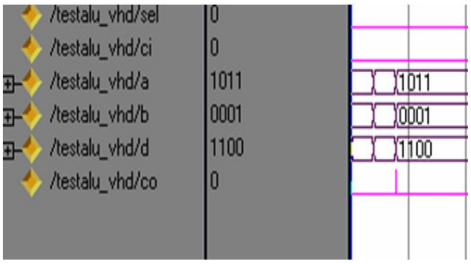

Figure 11: ALU addition operation

Memory: The 4 bits input of Decoder "0000", which is the first row of the memory (address A0) is selected. If the purpose of this instruction in the position of address A0 is to write and store, WR is also binary '0', so both SEL and WR are '0', then the input Data in the D-latch will be the same as the output of our D-latch. If Din (8 bits) is binary "10000000", then, '1' goes through the D-latch. Because of the MUX, only the values of first row of the memory will go to the tri-state buffer, and then the output can be obtained.

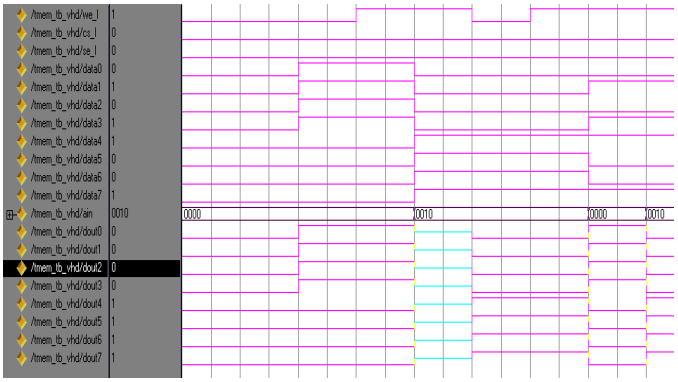

Figure 12: Memory simulation

Once these sub-elements were designed and tested, they were put together to constitute our FPGA-based processor. The simulation result of the whole processor is shown below:

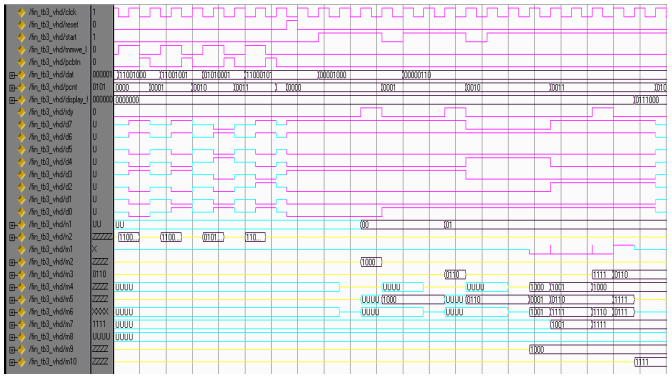

Figure 13: Processor simulation

Slices	132(6%)
Bells	351
Frequency	95.364MHz
Minimum period	10.486 ns

Table 3: Synthesis report

VI. CONCLUSION

We have successfully implemented an FPGA-based 8-bit processor on Xilinx Spartan 3 board using the VHDL language. Our processor is made of the control unit, and an arithmetic logic unit (ALU) and a memory unit. Our processor has a maximum frequency of 95.364 MHz and 132 slices were utilized.

VII. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support provided by National Science Foundation (NSF) grant number NSF – 0431818 “A Design- and Project- Oriented Innovative Electrical Engineering Program”.

VIII. REFERENCES

[1] V. S. Balakrishnan, H. Pottinger, F. Ercal, and M. Agarwal, Design and implementation of an FPGA based processor for compressed images (poster abstract). In *Proceedings of the 2000 ACM/SIGDA Eighth international Symposium on Field Programmable Gate Arrays* (Monterey, California, United States, February 10 - 11, 2000). FPGA '00. ACM, New York, NY, 218.

[2] R. Fryer, FPGA based CPU instrumentation for hard real-time embedded system testing. *SIGBED Rev.* 2, 2 (Apr. 2005), 39-42.

[3] P. Yiannacouras, Rose, J., and Steffan, J. G. 2005. The microarchitecture of FPGA-based soft processors. In Proceedings of *the 2005 international Conference on Compilers, Architectures and Synthesis For Embedded Systems* (San Francisco, California, USA, September 24 - 27, 2005). CASES '05. ACM, New York, NY, 202-212.

[4] G. Achery, C. Trinitis, R. Buchty, "CPU-independent assembler in an FPGA," *Field Programmable Logic and Applications, 2005. International Conference* , vol., no., pp. 519-522, 24-26 Aug. 2005

[5] F. Mayer-Lindenberg, V. Beller, “An FPGA-based floating-point processor array supporting a high-precision dot product” *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on Dec. 2006* Pages: 317 – 320.

[6] Y Nagaonkar and M. L. Manwaring. “An FPGA-based Experiment Platform for Hardware-Software Codesign and Hardware Emulation” In proceedings of *The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing* Pages: 169-174.

[7] D. Chiou, H. Sanjeliwala, D. Sunwoo, J. Z. Xu, and N. Patil, "FPGA-based Fast, Cycle-Accurate, Full-System Simulators," in *Proceedings of the second Workshop on Architecture Research using FPGA Platforms, held in conjunction with HPCA-12, Austin, TX*, Feb. 2006.