C3PU: A 16-bit Soft-Core Processor

Simon Cossart [1], Filip Keri [1], Luka Kovač [1], Ivan Sović [2]
[1] Summer School of Science (S3++), Požega, Croatia
[2] Ruđer Bošković Institute/Centre for Informatics and Computing, Zagreb, Croatia
ivan.sovic@irb.hr

***Abstract* - CPU is a small chip that can be found in electronic devices whose job is to execute programs, process data and control input/output units. CPUs can be classified in several ways, but from implementation viewpoint they can be either hard- or soft-core. The main difference between them is that hard-core CPUs have fixed architecture which cannot be modified, while soft-core CPUs are designed for reconfigurable platforms such as Field Programmable Gate Arrays (FPGAs).**

The main goal of this project was to design and implement a 16-bit soft-core CPU. We designed all components using VHDL, and simulated and checked their operation separately. After that, we connected all components to the control unit (CU), defined the instruction set, designed the CU's state machine, and tested the operation of the entire CPU. We also defined our assembly language and implemented two assemblers that translate our assembly code to the CPUs machine code. In this paper we show an example program (and simulation results) for calculating the Fibonacci sequence using our CPU. We named our processor C3PU, and it can be found freely available on the following link: https://github.com/isovic/C3PU .

***Keywords* – Von Neumann, VHDL, CPU, soft-core, FPGA, S3++, C3PU**

I. INTRODUCTION

A processor is a small chip that can be found in electronic devices whose job is to receive input and provide the appropriate output [1]. Nowadays, processors have a great role in everyday life. They are used for industrial purposes, medicine, traffic control, entertainment, etc. The use of processors is increasing day by day, and as a result we have to make the most of them; we are making systems that are energy efficient, fast, and suitable for the task.

Today, processors can be divided in two types from the implementation point of view: hard-core processors and soft-core processors. Hard-core processors are made using very-large-scale of integration (VLSI) technology which means putting a lot of components on a single chip. These components are fixed in place and cannot be modified. Their clock speed ranges from ≈ 100 MHz to several GHz. There are many manufacturers of hard-core processors, the largest ones being Intel and AMD. Although hard-core processors are most often used, we cannot neglect soft-core processors. They are designed using hardware description languages and can easily be reconfigured. Soft-core processors are usually targeted for implementation on *Field Programmable Gate Array* (FPGA) devices. FPGAs are logic devices that contain configurable logic blocks and connections between them [2]. Each configurable logic block contains lookup tables – tables with predefined outputs for some inputs (1 or 0). At the beginning, none of these blocks has any information about what to do with incoming input – we program them to do what we want. Hardware description languages like VHDL or Verilog can be used for that. One big downside to the soft-core processors is the clock speed of the FPGAs which usually ranges from 50 MHz to 500 MHz. Soft-core processors can be commercial (like Microblaze [3]) or free (and/or open-source; like OpenRISC [4]).

In this paper we will describe the making of our 16-bit soft-core CPU using VHDL and Spartan-3AN FPGA board (we call it C3PU). Our CPU uses the Von Neumann architecture (where data and program use the same memory). Also, to write programs for our CPU, we made our own assembly language and two assemblers that were programmed in C++ and Python to convert our assembly code to machine code. Both the assemblers and C3PU are available for free at https://github.com/isovic/C3PU .

II. GENERAL HARDWARE ARCHITECTURE

Central Processing Units (CPUs) or processors can be described as the heart of all "smart" devices. They can run complex programs through execution of sequences of very low-level commands (instructions) which perform some basic functions such as reading and writing data, calculating basic arithmetic, etc. The complete list of commands that can be run by a CPU is called the *instruction set*.

In general, computer architecture can be divided into three main types: Von Neumann, Harvard and modified Harvard. The basic difference among these is the organization of the memory access during instruction and data retrieval and storage: in Von Neumann architecture the same memory is used both for instructions and data, while in Harvard architecture instructions and data are stored on separate memory devices and are accessed over physically distinct data and address buses. In Von Neumann architecture, a program can access and change instructions as if it were data, while this is not possible in Harvard architecture. The modified Harvard architecture differs from the original Harvard only in that it allows the program to access and change instructions (as in the Von Neumann architecture).

Additionally, CPUs can be classified from the application viewpoint (general-purpose or application-specific) or implementation viewpoint (hard-core or soft-core).

A. General Von Neumann CPU Architecture

The Von Neumann CPU is connected to the memory using only one data bus, one address bus and one control bus. The type of connection to the memory greatly influences the way that internal CPU components are organized.

Although the concrete components of the CPU architecture vary from processor to processor, there are several components that can commonly be found in almost all such devices: registers, a comparator, a shifter, an arithmetic-logic unit and a control unit.

1) Registers

Registers are memory locations contained within the CPU. They are designed for quick access and fast data retrieval and storage [5]. A CPU can consist of many registers, depending on the architecture. However, some of them have predefined purposes which can be found between most processor designs. These are:

- *Program Counter (PC)* – holds the memory address of the next instruction that is to be executed.
- *Instruction Register (IR)* – holds the current instruction in the processor after it is fetched from memory, and while it is being decoded.
- *Accumulator (Acc)* – holds the result of operations performed by the arithmetic-logic unit.
- *Address Register (Addr)* – used for storage of a memory address during data retrieval and storage. Usually holds the address specified by an instruction.
- *Status Register (SR)* – holds 1-bit status flags (such as the zero and the carry flags) which are changed as a result of performing arithmetic-logic operations.
- *General purpose register arrays* – these registers have no specific purpose, but are used for the quick storage of pieces of data required by the program being executed.

2) Comparator

Comparator is a combinational component used for comparing two values. Comparison operations that this component implements are: *equal*, *not equal*, *less than*, *less than or equal*, *greater than* and *greater than or equal*. The output of the comparator is a 1-bit Boolean result value.

3) Shifter

Shifters are also combinational components, and are used for moving bits in a binary word one position either to the left or to the right [6]. The operations that the shifter can perform are shifting and rotating. Shifting operation will move the bits either to the left or the right but it would then replace the first or the last bit with a zero. When a rotation operation is performed, the bit that would normally be "shifted out" is placed instead of the zero at the beginning or the end of the word.

4) Arithmetic and Logic Unit (ALU)

ALU is another combinational component of the CPU that is involved with executing operations such as addition, subtraction, multiplication, division and bitwise operations. It performs operations on the data obtained from the register array, while the results are usually stored in the Acc and SR registers.

5) Control Unit (CU)

The CU is the most complex component of a CPU. CU is a sequential component, which implements a finite state machine that coordinates all other components of the CPU. The CU fetches the code of all of the instructions in the program, decodes the instructions and executes them by providing timing and control signals to all components.

6) Program and Code Memory

The program and code memory is not commonly built inside a CPU, but is connected externally with the CPU using the data, the address and the control buses. The maximum available size of the memory is dependent on the width of the address bus and the implemented instruction set.

B. FPGA Architecture

FPGA devices generally consist of several components: CLBs (configurable logic blocks), IOBs (input/output blocks), block RAM, multipliers, DCMs (digital clock management) and interconnect network. CLB is the basic logic unit of FPGA and it can be configured to handle shift registers, RAM, combinational and sequential logic [7]. Every CLB consists of 4 or 6 inputs (depending on the concrete FPGA architecture), selection circuit and flip-flops. IOB are special blocks for external connections, they control data flow between I/O pins and internal logic. Block RAM is dedicated RAM with two port memory with capacity of several kilobits up to several megabits [8]. DCM is used to implement delay locked loop, digital frequency synthesizer, digital phase shifter, or a digital spread spectrum. Interconnects are wires which connects inputs and outputs to logic blocks [9].

FPGAs can be reconfigured using hardware description languages. VHDL is one of such hardware description languages. It is used in electronic design automation to describe digital and mixed signal systems.

III. C3PU Implementation

The final architecture of the CPU designed during this project is shown in Figure 1. The architecture consists of five special-purpose registers: instruction register (IR), accumulator register (Acc), program counter (PC), address register (Addr)

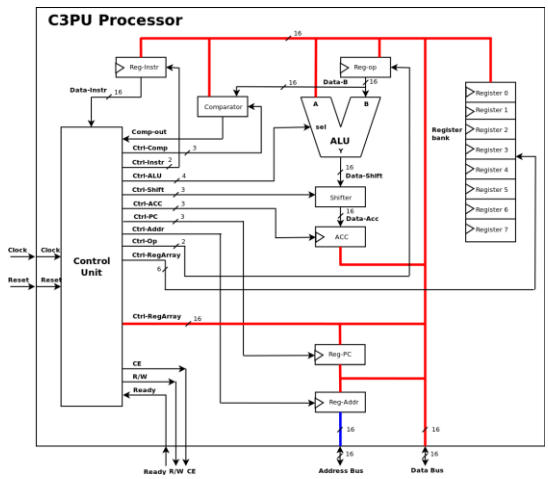

Figure 1. C3PU architecture.

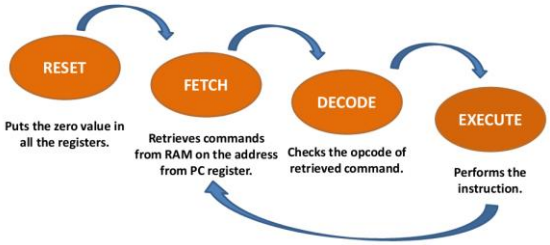

Figure 2. Phases of the execution pipeline implemented as a state machine in the control unit of C3PU.

and operand register (OpReg); one general-purpose register array of 8 registers, an arithmetic-logic unit (ALU), a comparator, a shifter and a control unit (CU). ALU, comparator and shifter are implemented as combinational components, while all registers and CU are implemented as sequential components. Register array is implemented as distributed RAM consisting of 8 data words (one word per register). C3PU is designed as a 16-bit CPU, which means that both the data and the address bus have 16 bits. Also, each register also consists of 16-bit data (the size of one word).

CU is the central part of any CPU. It controls the entire operation of the CPU, sends control signals, and directs the data flow. It is implemented as one big finite state machine, concretely consisting of 120 states. These states are generally divided into four phases: reset, fetch, decode and execute (as shown in Figure 2). Reset, fetch and decode phases do not depend on the instruction that is being executed, while the execute phase is implemented for every instruction separately.

When C3PU is started, the state machine of C3PU begins with the reset phase. During the reset phase, all registers are set to zero and the data bus is cleared. When all states of the reset phase are executed, C3PU goes into the fetch phase, where the next instruction (one data word) is retrieved from the RAM and stored in the instruction register. The address of the current instruction is always stored in the program counter, and is first transferred from the PC to the address register before the instruction if fetched. Following is the decode phase, where the operation code (opcode) of the instruction is checked in order to determine the correct execution state to jump to. This jump starts the execution phase. When an instruction is executed, the

Table 1. C3PU Instruction set.

Instruction Set		Instruction Set (continued)	
Name	Opcode	Name	Opcode
NOP	00000	OR	01010
LOAD	00001	XOR	01011
STORE	00010	NOT	01100
MOV	00011	ADD	01101
LOADI	00100	SUB	01110
STOREI	10111	ZERO	01111
JMP	00101	SHL	10000
JMPC	00110	SHR	10001
INC	00111	ROTR	10010
DEC	01000	ROTL	10011
AND	01001	END	10110

```
INC R1
INC R2
INC R4
MOV R2 R3
ADD R1 R2
MOV R3 R1
MOV R2 R4
JMP 3
```

Figure 3. Assembly code of the Fibonacci sequence generator.

```
0011100000000001
0011100000000010
0011100000000100
0001100000010011
0110100000001010
0001100000011001
0001100000010100
0010100000000011
```

Figure 4. Machine code of the Fibonacci sequence generator.

PC is increased by 1, and C3PU jumps back into the fetch phase.

C3PU instructions are 16-bit wide, where the most significant 5 bits represent the opcode of the instruction. The implemented instruction set consists of 22 instructions, as shown in Table 1. All instructions except LOADI and STOREI are single-word instructions. Single-word means that all memory addresses or register pointers are stored within the instructions. LOADI and STOREI are double-word instructions, and are implemented to allow addressing of the entire RAM (first word specifies the instruction and the target register, and the second word is the 16-bit address of the data in RAM).

We also created our own assembly language to simplify the program writing process. Assembly commands are as the ones specified in Table 1. We implemented two variations of our assembler (both providing similar functionality): one in Python and one in C++. These assemblers convert our assembly code into machine code, which can be transferred directly into C3PU RAM and executed.

The developed CPU was implemented in VHDL. For development purposes we used Xilinx ISE Webpack 14.6. To simulate the behavior of C3PU during development, we used ISim 14.6.

IV. Results

Using VHDL we have successfully designed and implemented the C3PU processor. For each of the components and for each of the instructions we have performed simulations to check if C3PU is working properly. For this purpose, test bench VHDL codes were written in which we controlled the inputs of the C3PU, and in the end we checked if the obtained results were the same as the predictions. Test benches have proven that each of the components and instructions worked correctly.

Table 2. C3PU logic utilization of the Spartan-3AN FPGA.

Logic Utilization	Used	Available	Utilization
Number of Slices	337	5888	5%
Number of Slice Flip-Flops	268	11776	2%
Number of 4 input LUTs	580	11776	4%
Number of bonded IOBs	2	372	0%
Number of GCLKs	1	24	4%

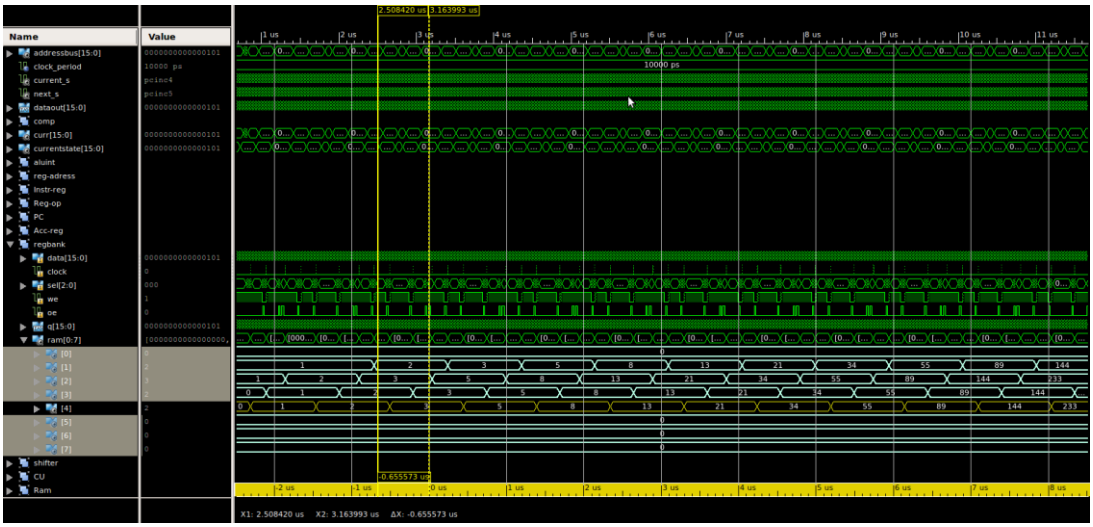

Figure 5. Results of C3PU simulation obtained after running the Fibonacci sequence generator code.

To further test C3PU, we have written a simple application for calculating the Fibonacci sequence. Fibonacci sequence is a sequence of numbers where the next number is the sum of two previous numbers in the sequence, where the first two numbers equal to 1. For example, first 7 numbers of the Fibonacci sequence would be: 1, 1, 2, 3, 5, 8 and 13.

Figure 3 shows the code for the Fibonacci sequence generator written in the developed C3PU assembly language. Figure 4 shows the corresponding machine code of the instructions from Figure 3.

After we run this program through the test bench, we should get a Fibonacci sequence written in the register 4 of the register array. Figure 5 shows the results of the simulation (only one part) obtained from the simulator output. Marked with the red square is the data stored in register 4. As can be seen, the results of the simulation match the Fibonacci sequence, which demonstrates the operation of C3PU.

Upon finishing the simulation procedures, we synthesized the design using Xilinx ISE Webpack 14.6, where the target device was Xilinx Spartan-3AN (XC3S700AN) on a Starter Kit development board. The design summary for device utilization is shown in Table 2. Unfortunately, the C3PU design did not at first successfully run in hardware, and will have to be further refined and optimized - for this we did not have sufficient time in the duration of this project.

V. CONCLUSION

The goal of this project was to design and implement a soft-core CPU. To achieve this, we implemented and thoroughly tested all components separately. After that, all components were connected together to form the complete architecture of the designed C3PU processor. Again, the operation of the entire processor was simulated to test every implemented instruction, following which sample programs were written to additionally check the operation of C3PU. In order to simplify the program development process, we defined our assembly language, and have written two versions of assemblers that translate our assembly code into C3PU's machine code. The final processor implementation has proven to be successful in simulation; however C3PU has not yet been successfully run in real hardware.

In future work we plan to implement the multiplication and division ALU operations; to expand the defined architecture with the status register that will contain flag bits (bits such as carry, zero, etc.); to allow C3PU to support input/output units, and finally, to test C3PU in real hardware. Also, it is our hope, that by publishing our design as open-source we will receive input from the user community which might help us achieve these goals and further improve C3PU.

ACKNOWLEDGMENT

Authors would like to thank the Society for Out-of-Frame Education, as well as the organizers and participants of the Summer School of Science 2013 in Požega for endorsing and supporting this project.

REFERENCES

[1] "Processor Definition", Accessed: 26. August 2013, URL: http://www.techterms.com/definition/processor

[2] P. P. Chu, "FPGA prototyping by VHDL examples Xilinx Spartan-3 Version", John Wiley & Sons, 2008.

[3] "Microblaze Soft Processor", Accessed: 26. August 2013, URL: http://www.xilinx.com/tools/microblaze.htm

[4] "OpenRisc", Accessed: 26. August 2013, URL: http://openrisc.net/

[5] "Microprocessor Tutorial", Accessed: 26. August 2013, URL: http://www.eastaughs.fsnet.co.uk/cpu/structure-index.htm

[6] E. O. Hwang, "Digital logic & microprocessor design with VHDL", La Sierra University, Riverside, 2005.

[7] "What is a FPGA", Accessed: 26. August 2013, URL: http://www.xilinx.com/fpga/

[8] "Block and distributed RAM's on Xilinx FPGA's", Accessed: 26. August 2013, URL: http://vhdlguru.blogspot.com/2011/01/block-and-distributed-rams-on-xilinx.html

[9] "FPGA", Accessed: 26. August 2013, URL: http://cs.washington.edu/education/courses/cse467/03wi/FPGA.pdf