An 8-bit Computer using FPGA

by

Aron G.
M013704

Under the guidance of

Prof. Karthik Subbu
Mithibai College, Mumbai

Nurturing talent...promoting excellence

मौलिक विज्ञान प्रकर्ष केन्द्र

Centre for Excellence in Basic Sciences
University of Mumbai
Kalina, Mumbai, India – 400098

Summer Semester 2018

Acknowledgement

I would like to express my profound sense of gratitude to Prof. Kathik Subbu for providing me this opportunity to learn under his invaluable guidance. I am highly indebted to Prof. Nagarajan for his constant comments and suggestions. I would specially thank Prof. Balwant Singh for the arrangements he made for the project. I would also like to thank my classmates and parents for their prolonged support. I also acknowledge the support provided by Centre for Excellence in Basic Sciences.

Abstract

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 project describes the realization of an 8-bit FPGA based simple processor. Our system was implemented on the Xilinx Spartan 6 xc6SLX9 using ISE foundation and VHDL.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Objectives . . . 1
1.3 Overview of FPGA . . . 1
1.3.1 What is FPGA? . . . 1
1.4 Why FPGA? . . . 2

2 Background **4**
2.1 Boolean Algebra . . . 4
2.2 Combinational Logic . . . 5
2.3 Sequential Logic . . . 6
2.3.1 D flip-flops . . . 6
2.3.2 T flip-flops . . . 7
2.3.3 Shift registers using D flip-flops . . . 8

3 Fundamentals of a computer **9**
3.1 Program Memory . . . 10
3.2 Data Memory . . . 10
3.3 Input/Output Ports . . . 10
3.4 Central Processing Unit . . . 11
3.4.1 Control Unit . . . 11
3.4.2 Data Path: Registers . . . 11
3.4.3 Data Path: Arithmetic Logic Unit . . . 12
3.5 A Memory Mapped System . . . 12

4 Conclusion **15**
4.0.1 VHDL code for 3 bit counter . . . 17
4.0.2 VHDL code for 4-bit shift register . . . 18

List of Figures

1.1 Architecture of a typical FPGA 2
1.2 Comparing a CPU, GPU and an FPGA 3

2.1 Basic Gates. 5
2.2 A clocked D flip-flop. 6
2.3 How a T flip-flop doubles the time period. 7
2.4 3 bit counter using a cascade of T flip flops. 7
2.5 A 4 bit shift register using D flip-flops. 8

3.1 Basic Hardware components of a computer. 9
3.2 Memory Mapping Configuration. 13
3.3 Memory Map of an 8-bit computer. 14

Chapter 1

Introduction

1.1 Motivation

Being in a world full of computers and Internet of Things(IoT), we are often ignorant about the mechanism of these technologies. To develop something futuristic, the core methodologies and working principles of these machines are to explored and understood.

1.2 Objectives

The main goals of this project are the following:

- To study the working principles of an FPGA.
- Emulate the mechanism of gates and flip-flops using VHDL.
- To understand the working principles and internals of computer.
- With the help of above knowledge, assemble an 8-bit computer on an FPGA.

1.3 Overview of FPGA

1.3.1 What is FPGA?

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Although one-time programmable (OTP) FPGAs are available, the dominant types are SRAM based which can be reprogrammed as the design evolves.

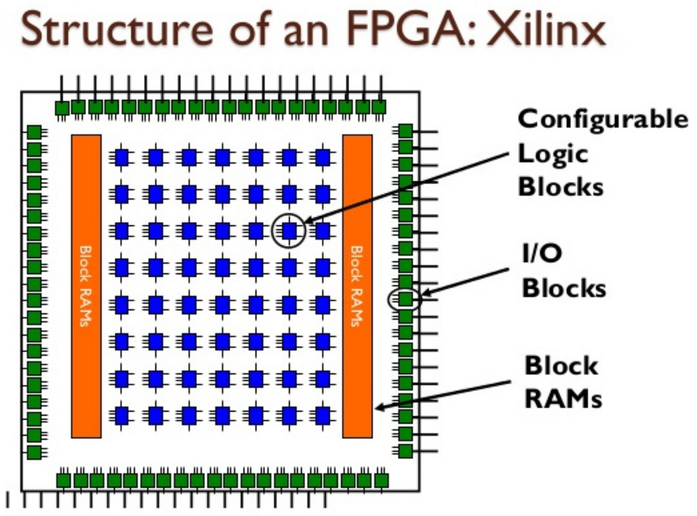


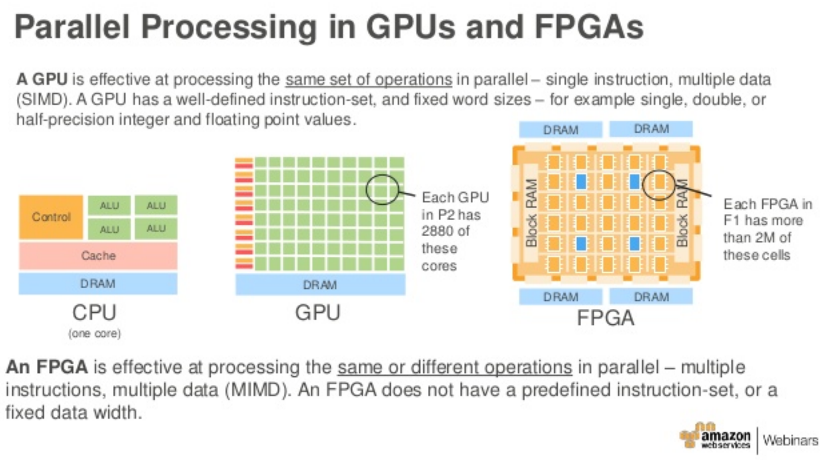
Figure 1.1: Architecture of a typical FPGA .

1.4 Why FPGA?

We do our daily computations on a CPU or Microprocessors which are Application Specified Integration Circuits (ASICs), which are a lot powerful than an FPGA. Then there seems to be no need for a device like an FPGA to be programmed on, but what really gives FPGAs an edge over the conventional ICs are that they are highly customizable. The following are few of the reasons why an FPGA has an edge over a conventional processor.

- FPGAs can be programmed at logic level. Hence it can implement faster and parallel processing of signals. This is difficult to be executed by processor.

- Unlike ASIC which are fixed once programmed, FPGAs are programmable at software level at any time. Hence FPGA IC can be re-programmed or reused any number of times. FPGA can also be programmed from remote locations.
- FPGA ICs are readily available which can be programmed using HDL code in no time. Hence the solution is available faster to the market.
- Unlike ASIC which requires huge NRE (Non Recurring Expenses) and costly tools, FPGA development is cheaper due to less costly tools and no NRE.
- In FPGA design, software takes care of routing, placement and timing. This makes lesser manual intervention. The design flow eliminates complex and time consuming place and router, floor planning and timing analysis.



Parallel Processing in GPUs and FPGAs

A GPU is effective at processing the same set of operations in parallel – single instruction, multiple data (SIMD). A GPU has a well-defined instruction-set, and fixed word sizes – for example single, double, or half-precision integer and floating point values.

An FPGA is effective at processing the same or different operations in parallel – multiple instructions, multiple data (MIMD). An FPGA does not have a predefined instruction-set, or a fixed data width.

Figure 1.2: Comparing a CPU, GPU and an FPGA .

Chapter 2

Background

We now look at the basic logic design fundamentals required for carrying out operations in a computer. Since the computer understands only '0's and '1's, which are off and on states respectively, we need to device a method to make meaning out of a series of those states. This is where Boolean Algebra, Combinational logic and Sequential logic comes to picture.

2.1 Boolean Algebra

Operations with 0 and 1:

$$X + 0 = X, \tag{2.1a}$$
$$X + 1 = 1, \tag{2.1b}$$
$$X.0 = 0, \tag{2.1c}$$
$$X.1 = X \tag{2.1d}$$

Idempotent laws:

$$X + X = X, \tag{2.2a}$$
$$X.X = X \tag{2.2b}$$

Involution law:

$$(X')' = X \tag{2.3a}$$

Laws of complementarity:

$$X + X' = 1, \tag{2.4a}$$
$$X.X' = 0 \tag{2.4b}$$

DeMorgan's Laws:

$$(X + Y)' = X'.Y', \tag{2.5a}$$
$$(X.Y)' = X' + Y' \tag{2.5b}$$

2.2 Combinational Logic

Now, we review combinational logic and then sequential logic. Combinational logic has no memory, so the present output depends only on the present input. Sequential logic, which we will encounter later, has memory, so the present output depends not only on the present input but also on the past sequence of inputs.

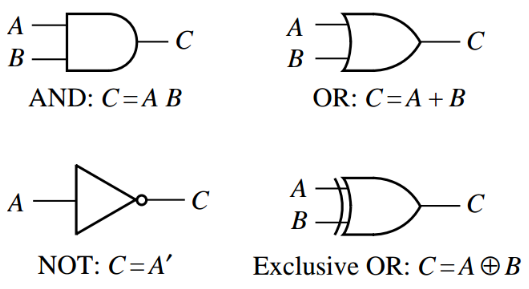


Figure 2.1: Basic Gates.

Some of the basic gates used in logic circuits are shown in Figure 2.1. Unless otherwise specified, all the variables that we use to represent logic signals will be two-valued, and the two values will be designated 0 and 1.We will normally use positive logic, for which a low voltage corresponds to a logic 0 and a high voltage corresponds to a logic 1.When negative logic is used, a low voltage corresponds to a logic 1 and a high voltage corresponds to a logic 0. For the AND gate of Figure 2.1, the output $C = 1$ if and only if the input $A = 1$ and the input $B = 1$. We will use a raised dot or simply write

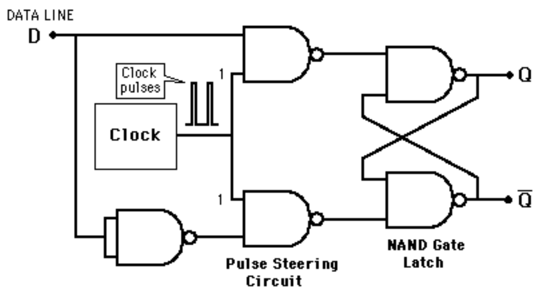
the variables side by side to indicate the AND operation; thus $C = A$ AND $B = A.B = AB$.
For the OR gate, the output C=1 if and only if the input A=1 or the input B=1 (inclusive OR). We will use + to indicate the OR operation; thus $C = A$ OR $B = A + B$. The NOT gate, or inverter, forms the complement of the input; that is, if $A = 1$, $C = 0$, and if $A = 0$, $C = 1$. We will use a prime (') to indicate the complement (NOT) operation, so $C =$NOT $A = A'$.

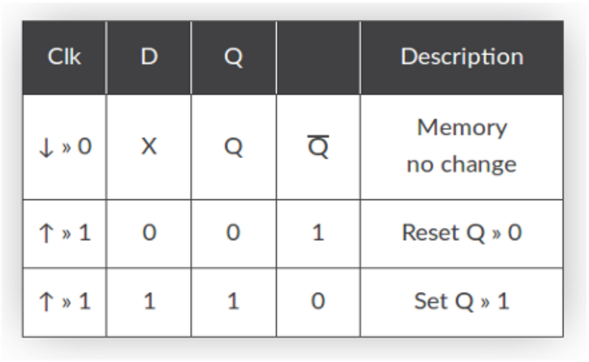
2.3 Sequential Logic

Sequential circuits commonly use flip-flops as storage devices. There are several types of flip-flops, such as Delay (D) flip-flops, J-K flip-flops, Toggle (T) flip-flops, and so on.

2.3.1 D flip-flops

Figure 2.2 shows a clocked D flip-flop. This flip-flop can change state in response to the rising edge of the clock input.The next state of the flip-flop after the rising edge of the clock is equal to the D input before the rising edge. The characteristic equation of the flip-flop is therefore $Q^+ = D$, where Q^+ represents the next state of the Q output after the active edge of the clock and D is the input before the active edge.





Clk	D	Q		Description
↓ » 0	X	Q	$\overline{Q}$	Memory no change
↑ » 1	0	0	1	Reset Q » 0
↑ » 1	1	1	0	Set Q » 1

Figure 2.2: A clocked D flip-flop.

2.3.2 T flip-flops

A T flip-flop is just a D flip-flop with the $\bar{Q}$ connected back to D. This makes the clock being the only input of a T flip-flop. This also brings down the frequency of the incoming clock as shown in figure. A cascade of T flip-flops connected to each other can make a counter.

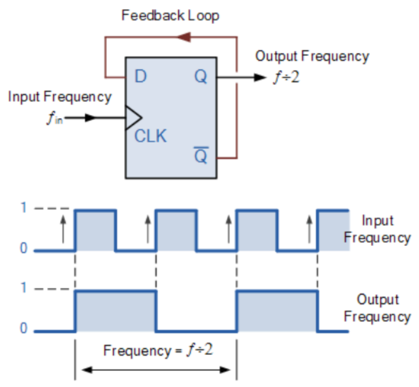
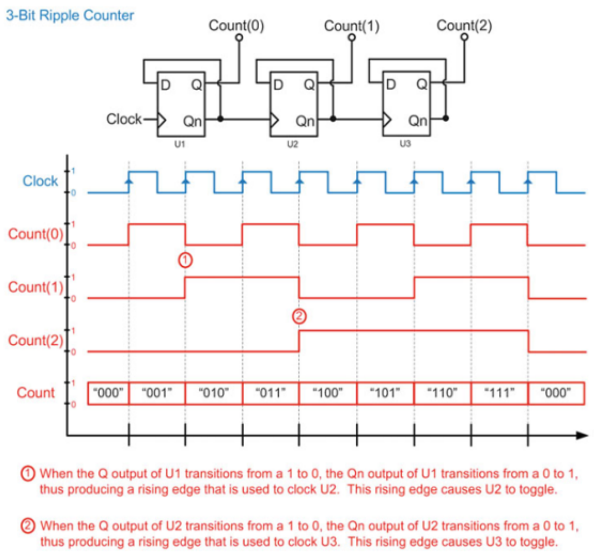


Figure 2.3: How a T flip-flop doubles the time period.



① When the Q output of U1 transitions from a 1 to 0, the Qn output of U1 transitions from a 0 to 1, thus producing a rising edge that is used to clock U2. This rising edge causes U2 to toggle.

② When the Q output of U2 transitions from a 1 to 0, the Qn output of U2 transitions from a 0 to 1, thus producing a rising edge that is used to clock U3. This rising edge causes U3 to toggle.

Figure 2.4: 3 bit counter using a cascade of T flip flops.

2.3.3 Shift registers using D flip-flops

Shift registers are the fundamental blocks of a computer, allowing it to transfer data(shift) across various pipelines. Just like we made counters using T flip-flops, shift registers can be made using a cascade of D flip-flops in the following clever manner.

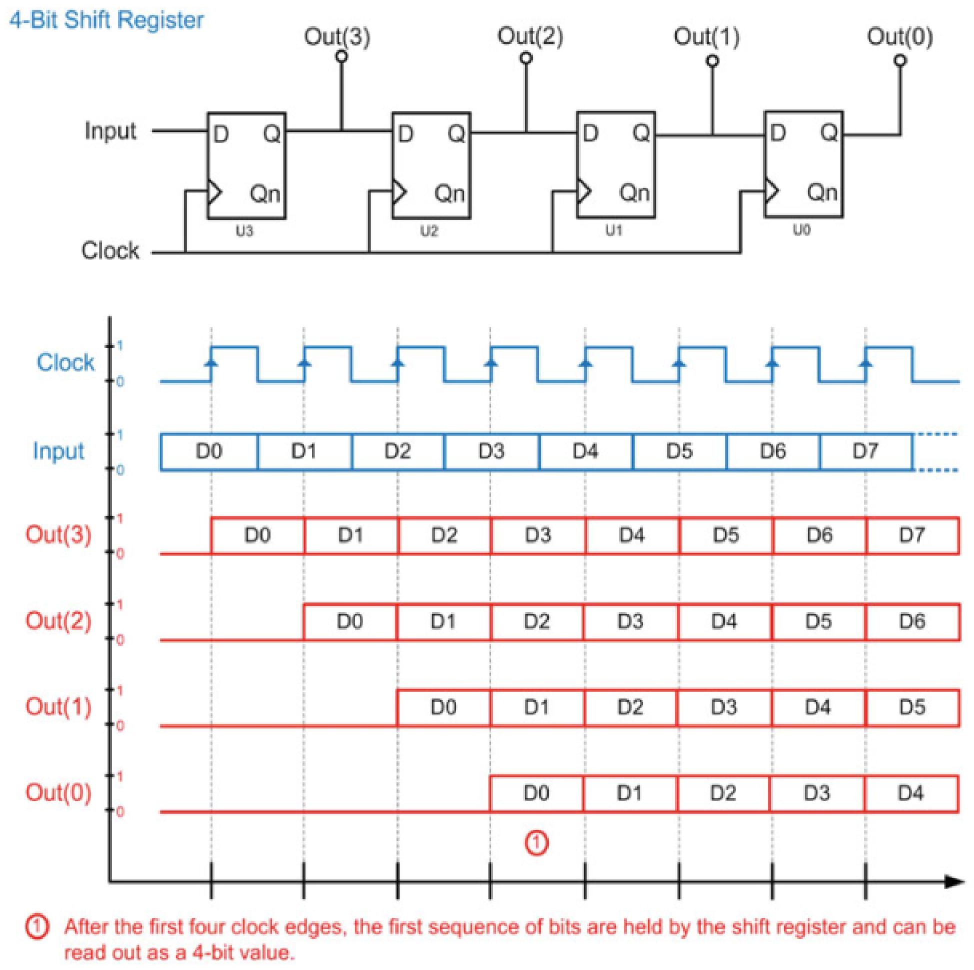

4-Bit Shift Register
Out(3)
Out(2)
Out(1)
Out(0)
Input
Clock
D Q Qn
U3
U2
U1
U0
Clock
Input
D0 D1 D2 D3 D4 D5 D6 D7
Out(3)
Out(2)
Out(1)
Out(0)
① After the first four clock edges, the first sequence of bits are held by the shift register and can be read out as a 4-bit value.

Figure 2.5: A 4 bit shift register using D flip-flops.

Chapter 3

Fundamentals of a computer

Computer Hardware

Computer hardware refers to all of the physical components within the system. This hardware includes all circuit components in a computer such as the memory devices, registers, and finite-state machines. Figure 3.1 shows a block diagram of the basic hardware components in a computer.

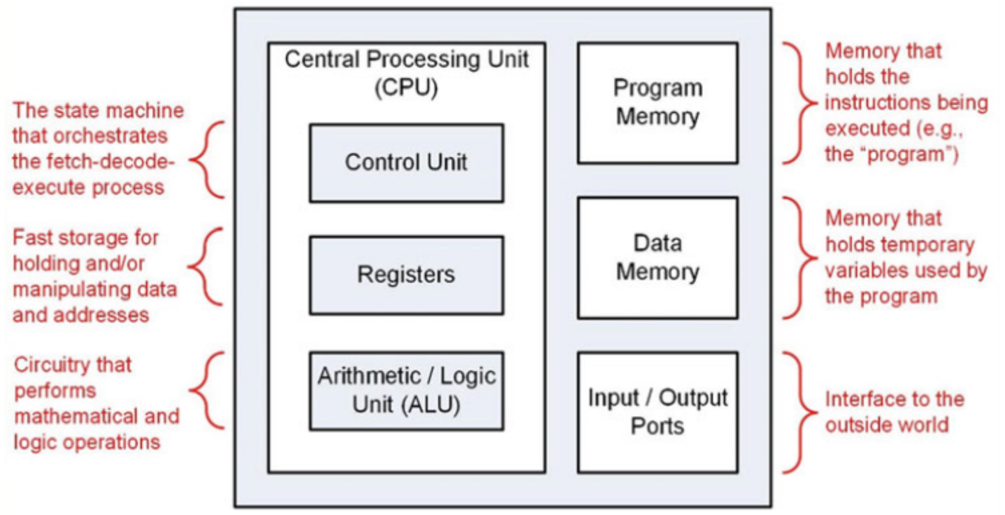

Central Processing Unit (CPU)
Control Unit
Registers
Arithmetic / Logic Unit (ALU)
Program Memory
Data Memory
Input / Output Ports
The state machine that orchestrates the fetch-decode-execute process
Fast storage for holding and/or manipulating data and addresses
Circuitry that performs mathematical and logic operations
Memory that holds the instructions being executed (e.g., the "program")
Memory that holds temporary variables used by the program
Interface to the outside world

Figure 3.1: Basic Hardware components of a computer.

3.1 Program Memory

The instructions that are executed by a computer are held in program memory. Program memory is treated as read-only memory during execution in order to prevent the instructions from being overwritten by the computer. Some computer systems will implement the program memory on a true ROM device (MROM or PROM), while others will use an EEPROM that can be read from during normal operation but can only be written to using a dedicated write procedure. Programs are typically held in nonvolatile memory so that the computer system does not lose its program when power is removed. Modern computers will often copy a program from nonvolatile memory (e.g., a hard disk drive) to volatile memory after start-up in order to speed up instruction execution. In this case, care must be taken that the program does not overwrite itself.

3.2 Data Memory

Computers also require data memory, which can be written to and read from during normal operation. This memory is used to hold temporary variables that are created by the software program. This memory expands the capability of the computer system by allowing large amounts of information to be created and stored by the program. Additionally, computations can be performed that are larger than the width of the computer system by holding interim portions of the calculation (e.g., performing a 128-bit addition on a 32-bit computer). Data memory is implemented with R/W memory, most often SRAM or DRAM.

3.3 Input/Output Ports

The term port is used to describe the mechanism to get information from the output world into or out of the computer. Ports can be input, output, or bidirectional. I/O ports can be designed to pass information in a serial or parallel format.

3.4 Central Processing Unit

The central processing unit (CPU) is considered the brains of the computer. The CPU handles reading instructions from memory, decoding them to understand which instruction is being performed, and executing the necessary steps to complete the instruction. The CPU also contains a set of registers that are used for general-purpose data storage, operational information, and system status. Finally, the CPU contains circuitry to perform arithmetic and logic operations on data.

3.4.1 Control Unit

The control unit is a finite-state machine that controls the operation of the computer. This FSM has states that perform fetching the instruction (i.e., reading it from program memory), decoding the instruction, and executing the appropriate steps to accomplish the instruction. This process is known as fetch, decode, and execute and is repeated each time an instruction is performed by the CPU. As the control unit state machine traverses through its states, it asserts control signals that move and manipulate data in order to achieve the desired functionality of the instruction.

3.4.2 Data Path: Registers

The CPU groups its registers and ALU into a subsystem called the data path. The data path refers to the fast storage and data manipulations within the CPU. All of these operations are initiated and managed by the control unit state machine. The CPU contains a variety of registers that are necessary to execute instructions and hold status information about the system. Basic computers have the following registers in their CPU:

1. Instruction Register (IR)—The instruction register holds the current binary code of the instruction being executed. This code is read from program memory as the first part of instruction execution. The IR is used by the control unit to decide which states in its FSM to traverse in order to execute the instruction.

2. Memory Address Register (MAR)—The MAR is used to hold the current address being used to access memory. The MAR can be loaded

with addresses in order to fetch instructions from program memory or with addresses to access data memory and/or I/O ports.

3. Program Counter (PC)—The program counter holds the address of the current instruction being executed in program memory. The program counter will increment sequentially through the program memory reading instructions until a dedicated instruction is used to set it to a new location.

4. General-Purpose Registers—These registers are available for temporary storage by the program. Instructions exist to move information from memory into these registers and to move information from these registers into memory. Instructions also exist to perform arithmetic and logic operations on the information held in these registers.

5. Condition Code Register (CCR)—The CCR holds status flags that provide information about the arithmetic and logic operations performed in the CPU. The most common flags are negative (N), zero (Z), two's complement overflow (V), and carry (C). This register can also contain flags that indicate the status of the computer, such as if an interrupt has occurred or if the computer has been put into a low-power mode.

3.4.3 Data Path: Arithmetic Logic Unit

The arithmetic logic unit (ALU) is the system that performs all mathematical (i.e., addition, subtraction, multiplication, and division) and logic operations (i.e., and, or, not, shifts). This system operates on data being held in CPU registers. The ALU has a unique symbol associated with it to distinguish it from other functional units in the CPU. Figure 3.3 shows the typical organization of a CPU. The registers and ALU are grouped into the data path. In this example, the computer system has two general-purpose registers called A and B. This CPU organization will be used throughout this chapter to illustrate the detailed execution of instructions.

3.5 A Memory Mapped System

A common way to simplify moving data in or out of the CPU is to assign a unique address to all hardware components in the memory system. Each

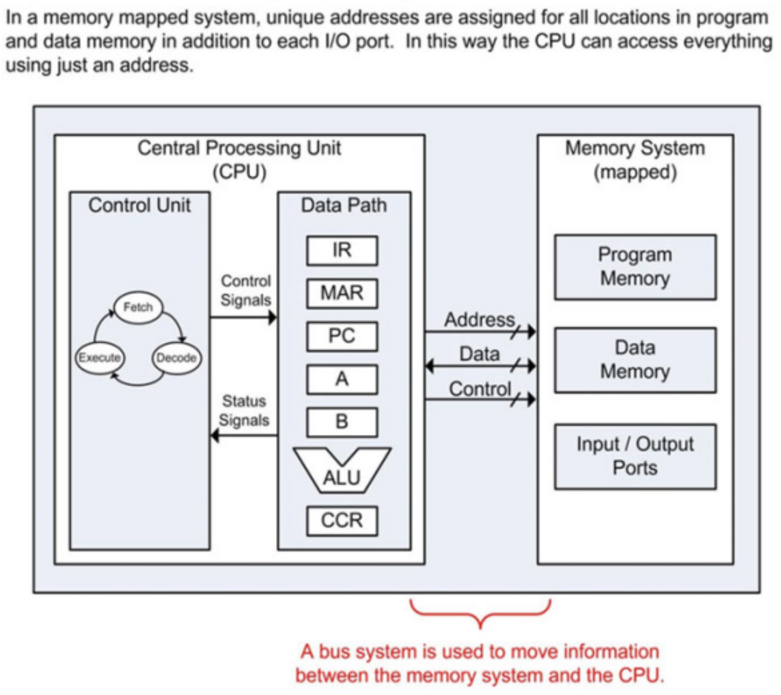

In a memory mapped system, unique addresses are assigned for all locations in program and data memory in addition to each I/O port. In this way the CPU can access everything using just an address.
Central Processing Unit (CPU)
Control Unit
Fetch
Decode
Execute
Control Signals
Status Signals
Data Path
IR
MAR
PC
A
B
ALU
CCR
Address
Data
Control
Memory System (mapped)
Program Memory
Data Memory
Input / Output Ports
A bus system is used to move information between the memory system and the CPU.

Figure 3.2: Memory Mapping Configuration.

input/output port and each location in both program and data memory are assigned a unique address. This allows the CPU to access everything in the memory system with a dedicated address. This reduces the number of lines that must pass into the CPU. A bus system facilitates transferring information within the computer system. An address bus is driven by the CPU to identify which location in the memory system is being accessed. A data bus is used to transfer information to/from the CPU and the memory system. Finally, a control bus is used to provide other required information about the transactions such as read or write lines. Figure 3.3 shows the computer hardware in a memory mapped architecture.

Memory map of an 8-bit computer

To help visualize how the memory addresses are assigned, a memory map is used. This is a graphical depiction of the memory system. In the memory map, the ranges of addresses are provided for each of the main subsections of memory. This gives the programmer a quick overview of the available resources in the computer system. Example 13.1 shows a representative memory map for a computer system with an address bus with a width of 8 bits. This address bus can provide 256 unique locations. For this example, the memory system is also 8 bits wide; thus the entire memory system is 256x8 in size. In this example 128 bytes are allocated for program memory; 96 bytes are allocated for data memory; 16 bytes are allocated for output ports; and 16 bytes are allocated for input ports.

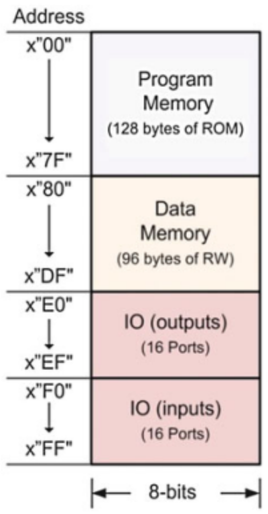


Figure 3.3: Memory Map of an 8-bit computer.

Chapter 4

Conclusion

We accomplished an 8-bit processor on an FPGA with a clockspeed of 25MHz, which is a pretty good number accordnig to FPGA standards. Nothing can give us a better understanding of a computer than building one from scratch. FPGA makes it possible without much effort because of its flexible architecture and a hassle free interface. What makes this project result in something more than just a calculator is the ability of FPGA to interact with almost every input and output devices that we daily use.

Bibliography

[1] Brock J. Lameres *Introduction to Logic Circuits* Springer 2017.

[2] Charles H Roth Jr and Lizy K John. *Digital Systems Design using VHDL.* Thomson Learning 2008.

[3] Peter J. Ashenden *Digital Design, An embedded Systems Approach using VHDL.* Morgan Kaufmann 2008

[4] David Romano *Make: FPGAs, Turning Software into Hardware with Eight Fun and Easy DIY Projects*. Make Media 2016

[5] Pong P. Chu *FPGA prototyping by VHDL Examples.* Wiley, 2008.

[6] Volnei A. Pedroni. *Circuit Design with VHDL.* MIT Press 2008

Appendix

Code listing for the model's architecture.

4.0.1 VHDL code for 3 bit counter

```
--------------First Counter--------------------------
Clock<=led;
process (Clock)
begin
if Clock'event and Clock='1' then
if T='0' then
tmp <= tmp;
elsif T='1' then
tmp <= not (tmp);
end if;
end if;
end process;
Q1 <= tmp;

-------------------------Second Counter----------------------
Clock2<= not Q1;
process (Clock2)
begin
if Clock2'event and Clock2='1' then
if T='0' then
tmp2 <= tmp2;
elsif T='1' then
tmp2 <= not (tmp2);
end if;
end if;
end process;
Q2 <= tmp2;
--------------------------Third Counter----------------------

Clock3<= not Q2;
process (Clock3)
begin
if Clock3'event and Clock3='1' then
if T='0' then
tmp3 <= tmp3;
elsif T='1' then
tmp3 <= not (tmp3);
end if;
end if;
end process;
Q3 <= tmp3;
```

4.0.2 VHDL code for 4-bit shift register

```
------------first bit-------------
Clock<=CLK;
D_flipflop1 : process (Clock)
begin
if (Clock'event and Clock='1') then
Q1<=D;
end if;
end process;

---------------second bit-------------
Clock<=led;
D_flipflop2 : process (Clock)
begin
if (Clock'event and Clock='1') then
Q2<=Q1;
end if;
end process;
```

```
----------third bit-------------

Clock<=led;
D_flipflop3 : process (Clock)
begin
if (Clock'event and Clock='1') then
Q3<=Q2;
end if;
end process;

----------fourth bit-------------

Clock<=led;
D_flipflop4 : process (Clock)
begin
if (Clock'event and Clock='1') then
Q4<=Q3;
end if;
end process;
```