A Simple 8-bit CPU Implementation on FPGA

1 MOTIVATION

In the computer architecture course, we learned about pipelines, branch prediction, multiple issue, reservation stations and etc. But those technologies are very complicated to implement. So we are wondering, aside from these advanced technologies, how does the simplest CPU work? What is a bus? What micro instructions are like? To answer these questions, we decide to learn a very simple CPU. We found a Youtube channel [1] explaining how various CPU components work, and make an 8-bit CPU using TTL logic components. Because we do not have so many kinds of TTL logic components and we also want to learn about FPGA, we decided to make a similar 8-bit CPU on the FPGA development board. All the code for this project is uploaded to github [2].

2 DESIGN

In this section, we introduce the overall design and each major components in this toy CPU.

2.1 Design Specification

To make the design simple enough so that we can fully understand, 8-bit processor seems to be a good choice. Because a byte has 8 bits, a 4-bit processor can be cumbersome when reading/writing memory. It needs to read/write the memory twice for one byte, and the 4-bit register cannot store the entire byte. After deciding the processor is 8-bit, next, we want to make a 8-bit, fixed length instruction set. It's much easier to decode and to fetch instructions compares to variable length instruction set as in x86 architecture.

The following is our design specification, based on the 8-bit setting.

- 8-bit instruction
- 8-bit general register
- 4-bit memory address
- Shared 4-bit address bus and 8-bit data bus (share 8 lines)
- Control signals (control bus)

The CPU has one 4-bit program counter (PC), one 8-bit instruction register (IR), two 8-bit general register, one arithmetic logic unit (ALU), one memory address register (MAR), one output register and 16-byte static memory (SRAM). These components are all connected to an 8-bit bus. It also has a control logic module that issues control signals to all the components. The overall view of the CPU's schematic is shown in Figure 1.

The thick line in the middle is the bus which has 8 lines. It transfers both address and data. All the component connects to the bus, that is, all components can be connected to each other through the bus. And this is achieved through the control logic module. We will explain it in more detail in the following sections.

[1]https://www.youtube.com/user/eaterbc
[2]https://github.com/whensungoesdown/8bitcpu

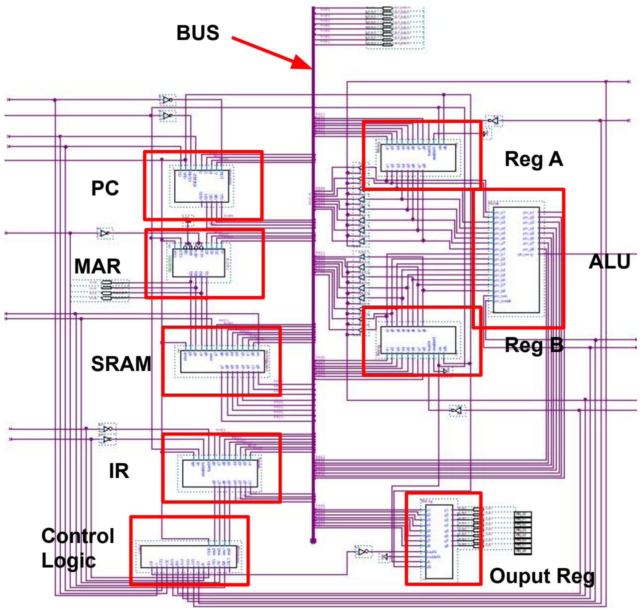

Figure 1: The overall view of the CPU

2.2 Instruction Set

Now we can create instruction set by ourselves. First we have to determine the format of the instruction. The upper 4-bit is opcode. All instructions have at most one operand, and this operand is a 4-bit memory address, as shown in Figure 2. There is no register operand, so some instructions have a default register. This design also leads us to only one memory addressing mode.

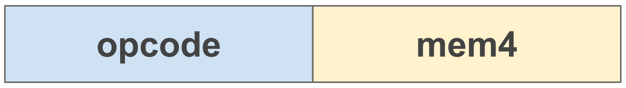

Figure 2: 8-bit instruction, 4-bit for opcode and 4-bit for memory operand.

Currently, we have only implemented 6 instructions, as shown below.

- NOP : No operation
- LDA mem4 : Load memory into register A
- ADD mem4 : Load memory then add it to register A
- DISPLAY_CFG : Send configuration data to 1602A LCD [2]
- OUT : Send data to 1602A LCD
- HLT : Halt

2.3 Register

The CPU has two general registers and one instruction register which are 8-bit. The MAR is 4-bit due to the memory space we have.

For 4-bit register, we use 74173 TTL logic module that provided by the FPGA software Quartus. 74713 is a 4-bit D-type register with 3-state outputs. For 8-bit register, we put two 74173 in parallel to form one 8-bit register, as shown in Figure 3.

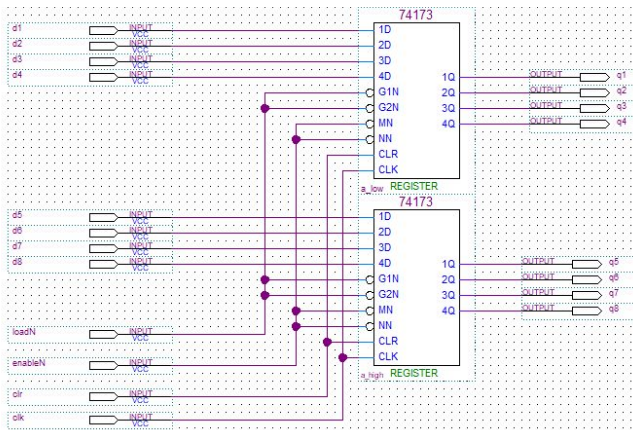

Figure 3: 8-bit register is composed of two 4-bit registers in parallel.

2.4 MAR and Memory

In this simple CPU, we do not use dynamic memory (DRAM) but static memory (SRAM). First, we are not sure whether FPGA can simulate capacitors and thus implement DRAM. Secondly, the CPU on modern computers runs much faster than memory, which requires a chipset to control it separately, or a CPU with a memory controller. We are not clear about these concepts yet, so we use SRAM to make memory and CPU work on the same frequency. This memory is quit similar to L1 instruction cache in modern computers, it only take one clock cycle to read/write. (Actually, because we use the memory module provided by Quartus, it takes 2 clock cycle to read/write. This is reflected in the control logic in our source code. But for simplicity, later in section 3, we assume that the memory only takes one clock cycle to read/write data)

The memory address register either stores the memory address from which data will be fetched from the memory module, or the address to which data will be sent to store. Basically, MAR holds the memory location of data that needs to be accessed. Since the CPU only have 16 bytes memory space, the MAR only needs 4-bit. As shown in Figure 4, the MAR's input (d) connects to bus and it's output (q) connects to the memory module. There is only one signal for MAR.

- MI (MAR IN)

When it is signaled, the MAR loads 4-bit data from bus. There is no MAR OUT signal, which means MAR always output current value to the memory module.

The memory module's input (d) and output (q) are all connected with the bus. There are two control signals.

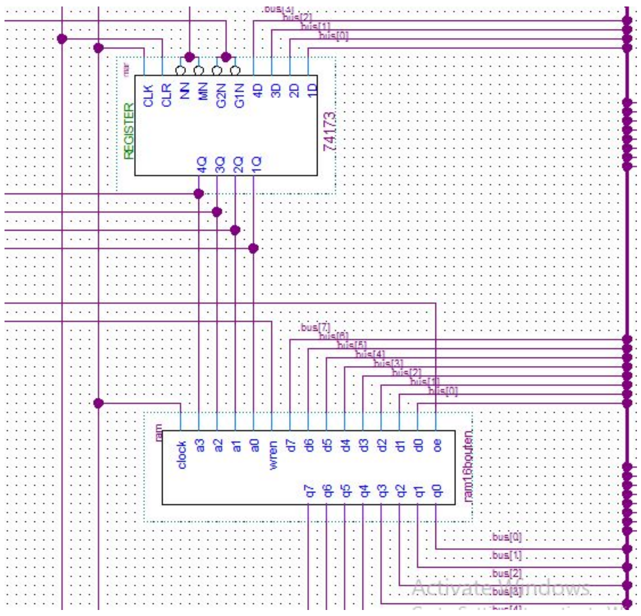

Figure 4: MAR's input connects to bus and output connects to memory module.

- RI (RAM IN)
- RO (RAM OUT)

When RI is signaled, data will be sent to the memory. When RO is signaled, memory outputs data to the bus.

2.5 IR

IR is the register that holds the instruction being executed or decoded. Decoding the opcode in the instruction register includes determining the instruction, determining where its operands in memory, retrieving the operands from memory, allocating processor resources to execute the command, etc. In this case, as mentioned earlier, the low 4-bit is the only memory operand, so it will be output to the bus. The high 4-bit output of the IR is available to control logic, which generate the timing control signal that controls various processing components involved in executing the instruction. Figure 5 shows that its input (d) connects to the bus, and its output (q) is divided into two parts, the low 4-bit part is sent to the bus, the high 4-bit part which is the opcode is sent to the control logic.

It has two control signals.

- II (IR IN)
- IO (IR OUT)

II controls when the IR gets the instruction from the bus. When IO is signaled, IR outputs to both bus and control logic.

2.6 General Registers

We have two general registers A and B. As shown in Figure 6, they are both connected to ALU module. Although they are called general-purpose registers, register A is special, it's also an accumulator register. Because our instruction set design is too simple, no

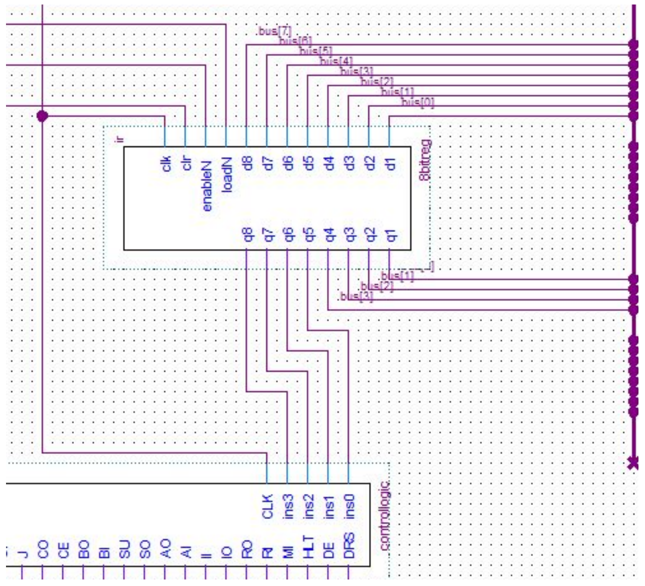

Figure 5: IR's input connects to bus and output to both bus and control logic.

operand is used to indicate the register, so the default register for some instructions such as "ADD mem4" is register A.

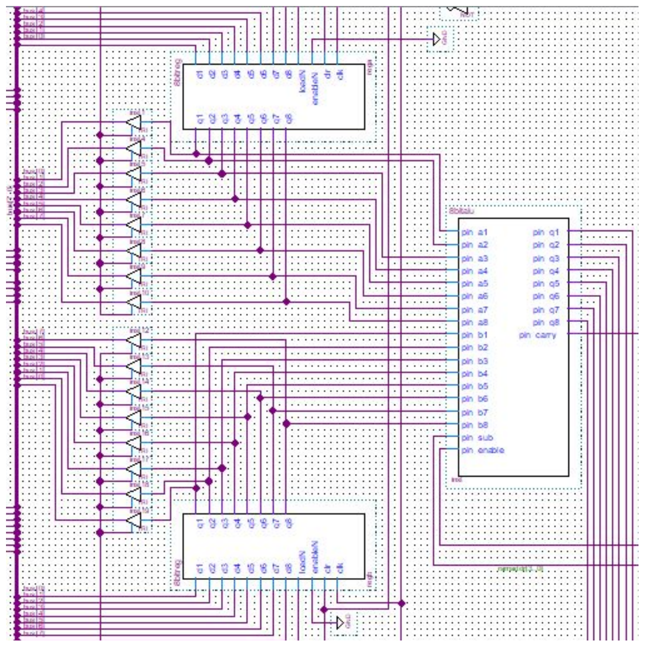

Figure 6: Both general register A and B are connected to ALU and bus.

Each general register has two control signals, corresponding to the read and write of the register, as listed below.

- AI (A IN)
- AO (A OUT)
- BI (B IN)
- BO (B OUT)

2.7 PC

The program counter (PC), commonly called the instruction pointer (IP) in Intel x86 microprocessors, and sometimes called the instruction address register (IAR) is a processor register that indicates where a CPU is in its program sequence.

Because our memory address space is only 16 bytes, the PC only needs 4 bits to cover the entire memory.

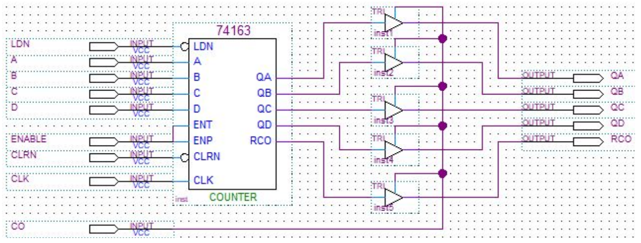

Figure 7: 4-bit Program Counter

We use 74163 4-bit counter as our program counter. Its 4-bit output will be sent to bus as a memory address to get one instruction from memory.

It also has two signals.

- CE (COUNTER ENABLE)
- CO (COUNTER OUT)

When CO is signaled, the program counter output its content to the bus. Without CE signal, program counter will not increase on each clock cycle.

2.8 Control Logic

Their are 18 control signals in our design. The combination of signals can control the transmission of data between corresponding components. For example, issuing AO (A OUT) and BI (B IN) at the same clock cycle can make the data in register A transferred to register B. These signals can be regarded as atomic operations, and the function of control logic is to form a complete instruction through these operations.

Each instruction requires many clock cycles to complete, and several control signals need to be issued in each clock cycle. To do this, we use ROM to encode the instruction logics. Some instructions are more complicated and required more steps, and some instructions are relatively simple. So we set that each instruction needs to be completed in 6 steps. We use an 74163 4-bit counter to count the steps, as shown in Figure 8.

The opcode of the instruction and each step number are encoded into an address. This address is used to index the data in the ROM. The data word width is 24 bits, which corresponds to 24 control signals. So far we have only used 18 of them.

3 CASE STUDY

In this section, we use an example to show how our CPU executes an instruction. We designed a load instruction

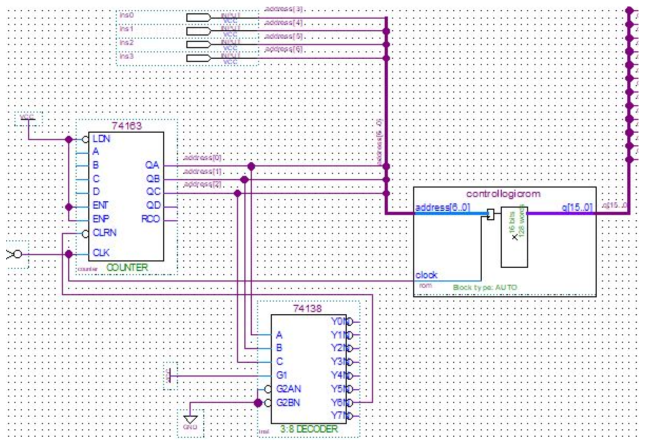

Figure 8: Control Logic Module

LDA mem4

This instruction indicates that the 4-bit operand is used as the memory address to read 1 byte of data into register A. For example, we want to load data from memory address 0xE, then the instruction is "LDA 0xE". Because the opcode of the LDA instruction is defined as 0x1, the binary form of this instruction is 0001 1110 (0x1E).

LDA instruction takes 4 clock cycle to finish.

(1) CO MI
(2) RO II CE
(3) IO MI
(4) RO AI

In these four clock cycles, the first two can be regarded as the instruction fetch phase. These two cycles are the same in almost all the instructions we implemented.

Suppose the initial state of the CPU is as follows.

- PC = 0x0
- Memory is initialized as in Table 1

0x0	0x1	0x2	0x3	...	0xB	0xC	0xD	0xE	0xF
0x1E	0	0	0	...	0	0	0	0xA	0

Table 1: Memory

The program counter points to the beginning of the memory, which stores the instruction binary 0x1E. At memory 0xE, which is the position to be read by this instruction, there is data 0xA.

3.1 Clock Cycle 1

We issue two control signals.

- CO (COUNTER OUT)
- MI (MAR IN)

By issuing these two control signals, the PC sends its 4-bit value to the bus, and the MAR get the 4-bit value from the bus, as shown in Figure 9.

As a result, the value in the PC is sent to the MAR and now MAR is 0x0.

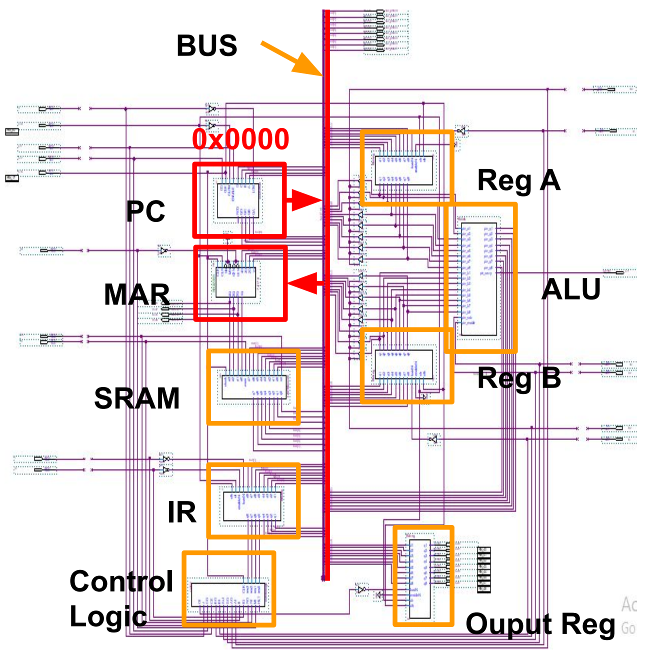

Figure 9: Clock Cycle 1

3.2 Clock Cycle 2

We issue three control signals.

- RO (RAM OUT)
- II (IR IN)
- CE (COUNTER ENALBE)

By issuing RO and II, the RAM sends the byte to the bus and the IR gets it, as shown in Figure 10. CE is also issued at this clock cycle, so the PC will increase one.

As a result, the instruction is successfully loaded into the IR. Now IR contains value 0x1E.

3.3 Clock Cycle 3

We issue two control signals.

- IO (IR OUT)
- MI (MAR IN)

This clock cycle can be viewed as the decoding phase. When IO is signaled, as mentioned earilier, the data in the IR is divided into two parts, the lower 4 bits are sent to the bus, and the upper 4 bits are send to the control logic, as shown in Figure 11.

As a result, the operand which is a memory address, now is loaded into MAR. MAR has 0xE and the control logic also gets the opcode.

3.4 Clock Cycle 4

We issue two control signals.

- RO (RAM OUT)
- AI (A IN)

Because in the last clock cycle, the MAR already loaded with the memory address 0xE. In this clock cycle, the RO causes the memory

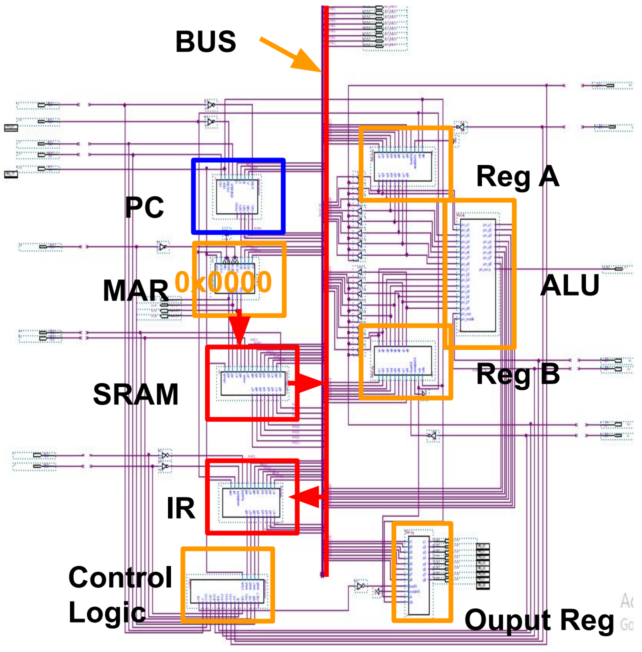

Figure 10: Clock Cycle 2

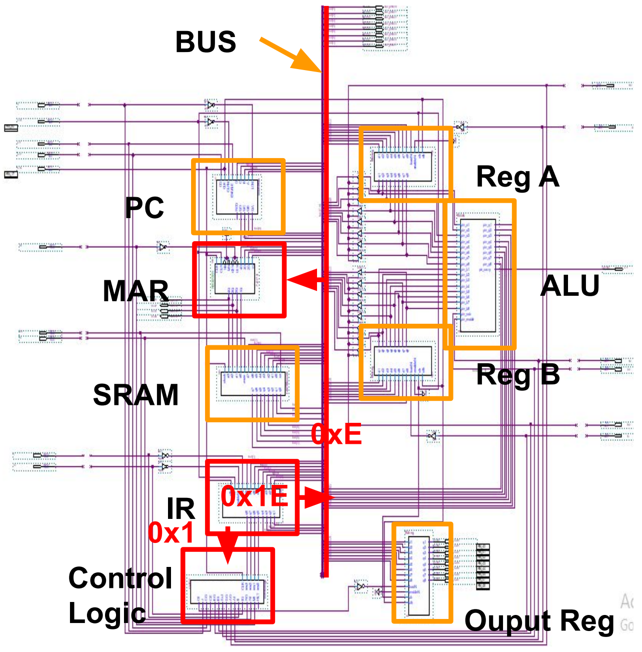

Figure 11: Clock Cycle 3

to send the corresponding data to the bus. Meanwhile, register A gets data from the bus, as shown in Figure 12.

So we see that when this clock cycle ends, the value in memory 0xE has been successfully loaded into register A. Now register A has the value 0xA.

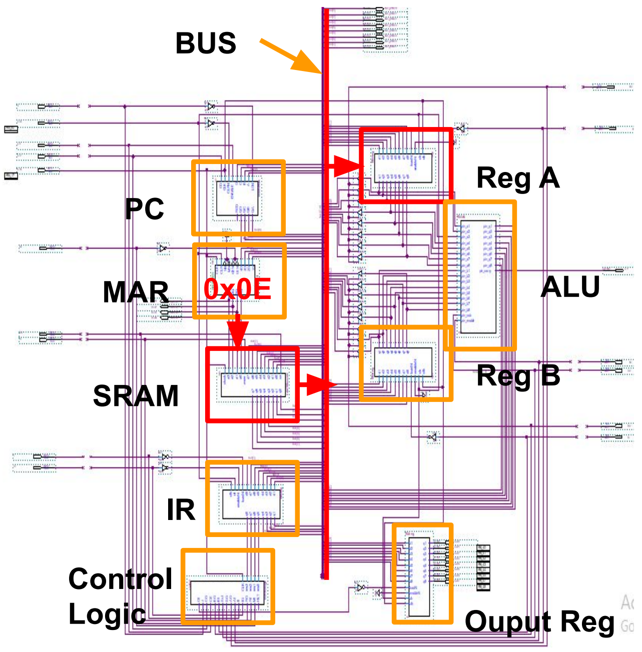

Figure 12: Clock Cycle 4

4 SYSTEM SETUP

4.1 Source Code

The source code of this 8-bit CPU is uploaded to Github [3].

It has the following files.

8-bit register:

- 8bitreg.bdf
- 8bitreg.bsf

8-bit ALU:

- 8bitalu.bdf
- 8bitalu.bsf

Control Logic:

- controllogic.bdf
- controllogic.bsf
- controllogicrom.bsf
- controllogicrom.cmp
- controllogicrom.mif
- controllogicrom.qip
- controllogicrom.vhd

4-bit counter:

- counter4b.bdf
- counter4b.bsf

Memory:

- ram16bouten.bdf
- ram16bouten.bsf
- ram16byte.bsf
- ram16byte.cmp
- ram16byte.qip

[3]https://github.com/whensungoesdown/8bitcpu

- ram16byte.vhd
- 16byteram.mif

Frequency divider:

- clk1MHz.bsf
- clk_div50.vhd

CPU assembly:

- testcpu3.bdf
- testcpu3.qpf
- testcpu3.qsf

Waveform files for debugging purpose:

- *.vwf

4.2 Setup

The FPGA developing board we use is Cyclone II EP2C5 Mini Dev Board [1]. And the software we use is Quartus II 64-bit Web Edition.

The display we use is 1602A LCD. The hardware setup is shown in Figure 13.

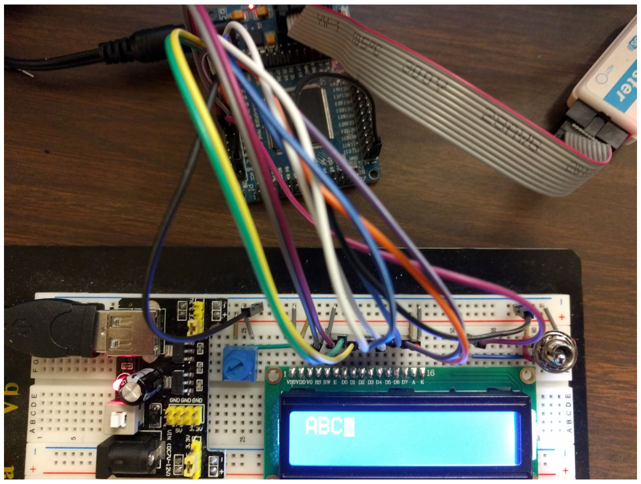

Figure 13: The CPU and display setup.

We also need a USB Blaster to download the compiled code to the development board.

Both the Cyclone II development board and the LCD need 5V power supply. We power them separately with common ground. The FPGA's GPIO operating voltage is 3.3V. Because the LCD data pins have a wide range of operating voltage which accept 3.3V signals, we can directly connect the GPIO on the FPGA to the LCD.

5 CONCLUSIONS

In this course project, we implemented a very simple 8-bit CPU on FPGA development board using simple TTL logic components such as the 74 series and very small amount of VHDL code. The CPU only has two general registers, and the ALU part can only do addition and subtraction. But it has some typical elements in CPU design such as IR, MAR, PC, memory and control logic. Through this course project, we understand better how a CPU executes an instruction. And have a more intuitive understanding of the bus.

REFERENCES

[1] Ltd Intel Co. 2008. Cyclone II Device Handbook. (2008). https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyc2/cyc2_cii5v1.pdf

[2] LTD SHENZHEN EONE ELECTRONICS CO. 2000. Specification for LCD Module 1602A-1. (2000). https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf