THE DESIGN AND IMPLEMENTATION OF A MICROCONTROLLER IN VHDL
PRESENTED TO DR. TONY WILCOX

BY ZEESHAN MUSTAFA LATIF ANSARI
BENG (HONS) ELECTRONICS ENGINEERING
Abstract

This report provides a brief description on how to use a hardware description language (VHDL) in Altera Cyclone III FPGA (EP3C10E144C7) using Quartus II software for the design, simulation and implementation of a simple microcontroller (CPU), how to add a peripheral and how to program the microcontroller with assembly code to fulfil the project requirements given in the assessment brief.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Section 1: Datapath Design and Test</td>
<td>4</td>
</tr>
<tr>
<td>Arithmetic Logic Unit (ALU)</td>
<td>5</td>
</tr>
<tr>
<td>ACCUMULATOR (ACC)</td>
<td>6</td>
</tr>
<tr>
<td>Program Counter (PC)</td>
<td>6</td>
</tr>
<tr>
<td>Instruction Register (IR)</td>
<td>7</td>
</tr>
<tr>
<td>Address Multiplexer (ADDMUX)</td>
<td>8</td>
</tr>
<tr>
<td>INPORT</td>
<td>9</td>
</tr>
<tr>
<td>OUTPORT</td>
<td>9</td>
</tr>
<tr>
<td>Immediate Multiplexer (IMMUX)</td>
<td>9</td>
</tr>
<tr>
<td>Memory (RAM)</td>
<td>10</td>
</tr>
<tr>
<td>Section 2: Controller Analysis and Integration with Datapath</td>
<td>11</td>
</tr>
<tr>
<td>TP24 Schematic</td>
<td>11</td>
</tr>
<tr>
<td>Fetch</td>
<td>13</td>
</tr>
<tr>
<td>Execute</td>
<td>14</td>
</tr>
<tr>
<td>TP24 Waveform</td>
<td>15</td>
</tr>
<tr>
<td>Section 3: Addition of Display Peripheral and Assembly Code to test it</td>
<td>15</td>
</tr>
<tr>
<td>Display State-machine Diagram</td>
<td>16</td>
</tr>
<tr>
<td>Display Schematic</td>
<td>17</td>
</tr>
<tr>
<td>Display Waveform</td>
<td>17</td>
</tr>
<tr>
<td>TP24V1DV1</td>
<td>17</td>
</tr>
<tr>
<td>Summary/Conclusion</td>
<td>18</td>
</tr>
<tr>
<td>References</td>
<td>18</td>
</tr>
</tbody>
</table>
Introduction

Keeping this project in mind, the CPU was defined to be nothing more than a finite state-machine that executes the machine code programs – a machine to fetch instructions from memory and then to execute them. Here, the finite state-machine consists of an architecture and a controller. The controller holds the current state of the system whereas the architecture computes the next state which is dependent on the current state and on any other inputs. Machine codes are numbers that represent instructions and data which are organised in some logical way (using Von Neumann architecture for this project) (Dr Wilcox, 2017).

Minimum requirements considered for this CPU instruction set was firstly data manipulation including arithmetic and logical operations, secondly data transfer operations with read and write memory and lastly program flow control using conditional and unconditional branches. The ALU, which provides the core functionality of the CPU, supported the following operations to fulfil the requirements of the instruction subset:

\[
\text{INC, DEC, SHR, SHL, NOT, ADD, SUB, AND, OR, XOR, F=A, F=B}
\]

These 12 ALU operations including 4-bit ALU mode select were all that were required in order to implement an accessible instruction set for “Datapath Architecture Design of a TP24 processor”.

The RTL (Register Transfer Level) is a synthesizable HDL model or dataflow design. It is a technique used to design the complex logic circuits such as a microprocessor. Three stages that consist of RTL design are given below:

1. Determining the sizes and number of registers required to hold the data used by the device.
2. Concluding the logic and arithmetic operations required to make use register contents.
3. Designing a state-machine whose outputs are used to determine that how these register contents are updated with the results of those operations in order to obtain the desired results (ELEC, 2017).

ISA (Instruction set architecture) basically makes a connection between the software and hardware where software sends instructions to hardware to perform operations.

The elements of the architecture included ALU, IR, PC, ACC, MUX and MEM. Simple design integrated an 8-bit opcode with a 16-bit operand in one 24-bit instruction word. For memory, locations used were \(2^{16} \rightarrow 64K = 65536\).

First the “Datapath” was designed and tested, then “Controller” was analysed and integrated with Datapath making it “TP24” and finally the “Display Peripheral” was made, tested and integrated with TP24.
Section 1: Datapath Design and Test

INSTRUCTION SET ARCHITECTURE (Bottom up Hierarchy)
Arithmetic Logic Unit (ALU)

ALU makes the main part of the CPU functionality as it consists of data manipulation of the CPU like logical and arithmetic operations.

ALU operates on two input vectors (being A(n..0) and B(n..0)) to give results based on the mode selected. One of these two inputs is registered while fetching the other input. Register hold the data that is being operated. Accumulator is used to hold one of the inputs above. Flags Include Carry, Sign, Zero, and Overflow. Generally it is a combinational logic unit. Instruction sets for ALU chosen for this project were tested and simulated individually.
**Accumulator (ACC)**

The accumulator is a 16-bit edge-triggered latch which is used to store and send back a 16-bit word when accumulator clock is raised. Operation results were registered by connecting the ALU output to the accumulator input. The register was used to execute the accumulator functionality. The accumulator was simulated and tested to check its functionality as shown below:

**Program Counter (PC)**

The program counter is used to store the address of the next instruction ready for execution (decoding). In order to determine the next instruction, the controller increments the instruction address by one, whereas, sometimes there is a need to use jump between addresses. Jump is used to load particular address and reset is also used to jump to the beginning of the program memory. Therefore, in order to make good use of PC, it should be designed to be able to store address, parallel loading and resetting. The PC design and code is compiled and simulated to check its functionality as can be seen below:
Instruction Register (IR)

IR is used to hold the instruction read from the memory which is ready for decoding. In **24-bit words**, both the opcode (8-bit) and operand (16-bit) are called by a single memory access. IR divides the 24-bit code to opcode (8-bit) and operand (16-bit). The IR code and design has been complied, simulated and finally vector waveform was created to check its functionality as shown below:

Pc_load has priority over pc_inc
If pc_load = 1 then addr out = addr_in
If pc_inc = 1 then addr_out = precious addr_out + 1

If reset happens then
addr_out = 0000
Reset has priority, therefore when reset is “high” pc_load and pc_inc do not affect the results
Address Multiplexer (ADDMUX)

In ADDMUX, the “addr_select” determines the source of address which can be either PC or the IR.

Rising edge of ir_clk

At rising edge Data is registered to two parts as shown above, 12 to opcode and 3456 to operand

When addr_select is 1 and value is 1234 so 1234 is moved to “f”.

When addr_select is 0 and value is 5678 so 5678 is moved to “f”.
**INPORT**

As shown by the code, when input “sel” is active low it outputs “b” and when its active high it outputs “a”. The inport was tested to prove its functionality.

```vhdl
op1: process (sel, a, b)
    begin
        if sel = '0' then
            f <= b;
        else
            f <= a;
        end if;
    end process op1;
```

**OUTPORT**

The port here can be defined as a memory location brought to the outside world. This “outport” is a 16-bit register.

```vhdl
begin
    op1: process
        begin
            wait until clk = '1';
            data_out <= data_in(15 downto 0);
        end process op1;
    end behav;
```

**Immediate Multiplexer (IMMUX)**

IMMUX is immediate data multiplexer which loads “alu” data to “f” when “im” is low and loads “opr” data to “f” when “im” is high.

At every rising edge “data_in” moves to “data_out”
Memory (RAM)
This memory can read and write. It is read by pulling the “wren” pin low and clocking the mem clock line. Then the memory location is read on the “address” port and the value of the memory location is sent on the “q” port (which is data out). It is written by taking the “wren” high and then clocking the mem clock line. Then the value is stored on the “data” (which is data in) at the memory location on the “address” port.

To achieve successful data transfer so that each control signal transition can reach the stability status, the datapath architectures need to be tested on each edge of the clock.
Section 2: Controller Analysis and Integration with Datapath

The controller is mainly used here to sequence the flow of data through the datapath architecture. Controller first fetches the instruction from the memory, then decodes it and finally executes it according to the 8-bit opcode. For microprocessors, it is known as Fetch-execute cycle. The datapath architecture is controlled by using the output signals of the controller. Its outputs act as the control signals for the multiplexers, gates and edge triggered flip flops that pass and store data.

Before the “Fetch” starts, the controller must sequence the control lines. As the “Program Counter” holds the address of the next instruction, on start-up (reset), the PC equals “0000H” and all the clocks are set to low. Thus for activation the clock is set to high whereas all other control signals are set to low.

The diagram above is the top-level FSM for the CPU.

The diagram above is the top-level FSM for the CPU.

The Fetch sequence refers to fetching the next instruction from the memory. The fetch cycle consists of four consecutive stages (F0, F1, F2 and F3). In the F0 stage, the current PC (which is reset to 000H at start-up) is applied to the RAM address input. In the F1 stage, it starts with reading 24-bit instruction from RAM and ends with writing them to the Instruction
Register. In the F2 stage, it increments the PC to point to the next instruction and in the final stage F3 the Instruction Register decodes the instruction register into opcode (8-bit) and operand (16-bit). F3 stage completes the fetch cycle and lastly architecture has the op-code latched into the IR for the controller to use, and the operand is on the internal bus.

**Step 1**
Setting addr_select = 1 (PC) passes the PC address to memory via the address multiplexer.

**Step 2**
Setting wr_en = 0 (Read) enables read from memory.

**Step 3**
Setting ram_clk = 1 latches the address on the rising edge.

**Step 4**

When ir_clk = 1 latch the opcode and operand.

Set ram_clk to ‘0’

Increment the PC ready for next FETCH

Enable read from memory

---

*Image of diagram with labeled steps:*
Fetch

Mif file:

<table>
<thead>
<tr>
<th>Addr</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AB2233</td>
<td>ED4455</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>1</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

2nd instruction: When mem_clk is triggered (high), ir_clk and pc_clk is also high, therefore the opcode (CD) is fetched from first mif file location. IR_CLK holds the current instruction (CD) until the next or 3rd ir_clk is high.

1st instruction: When mem_clk is triggered (high), ir_clk and pc_clk is also high, therefore the opcode (AB) is fetched from first mif file location. IR_CLK holds the current instruction (AB) until the next or 2nd ir_clk is high.

At reset, the fetched value in opcode is 00 as it should be.

Execute: LDA 0E, Load acc with contents of location 0Eh

Table used to create the machine code for the required instruction:

<table>
<thead>
<tr>
<th>Addr</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16000E</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>1</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

s09466807 Page 13
Execute: STA 0D, Store acc with contents of location 0Dh

Table used to create the machine code for the required instruction:

<table>
<thead>
<tr>
<th>Addr</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>00</td>
<td>0D</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>8</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

Opcode = 10h = LDA

Acc = 6789h

Opcode = 11h = STA
**TP24 Waveform**

**Execute: LDA 0D, Load acc with contents of location 0Dh**

Table used to create the machine code for the required instruction

<table>
<thead>
<tr>
<th>Addr</th>
<th>-0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100000</td>
<td>F00000</td>
<td>020000</td>
<td>1F0000</td>
<td>030000</td>
<td>1F0000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>8</td>
<td>000001</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td>00224</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

Section 3: Addition of Display Peripheral and Assembly Code to test it

The processor can be connected to other peripherals (i.e. switches, LCDs, 7Segment Display, RS232) by using their inputs and outputs so that it can be used as a microcontroller. The processors job is to process the data from inports and send them to the outports according to the machine codes (memory instructions).

The peripheral connected to the processor (TP24) is a 7 segment display. It allows the user to display debug information or feedback information which can be used while entering information into a program. The display unit is comprised of counter, address decoder, register file and segment decoder. The “sysclkdiv” is the first block in the display. It is used to divide the clock frequency to 200HZ – the system clock is 50mHz therefore sysclkdiv divides the frequency by $2^{18}$ (262144) to get the output frequency of approximately 200Hz. These blocks actually time multiplexes the display. The refresh rate is give as $50\text{MHz} / 2^{18} = 50\text{MHz} / 2^{18} \approx 190.7\text{Hz} \approx 5\text{ms}$
There are in total five states of the LCD controller state machine. The LCD controller enters the Power-up state when it is started but until the supply voltage stabilises it has to wait 50ms. After that, it enters the initialize state where an initialization sequence is started in which the controller cycles the LCD and sets the LCD’s display setting e.g. cursor on or off and the number of lines etc. After setting the LCD display, it jumps to its Ready state and then moves on to the Send state when the Icd_enable input is activated. In the Send state, the required information provided by the Icd_bus input is sent to the LCD. Now until further invoked, it returns to its Ready state after the time specified in the software which is 50us. The controller resets to the Power-up state and re-initializes at any time when logic high is applied to the reset input.

Out of the CPU output which is 16-bit the first 7 bits are connected to the display unit. These 7 bits have 3 sections as follow:

1. In section 1, the first 4 bits are connected to “data_in” which are the output value displayed on each 7 segments. The 7 segment (4-16 bit) decoder decodes these 4 bits and displays them. For example if the 4 bits are 0101, it will display 5.

2. In section 2, the bit 5 and 6 are connected to the “reg_addr” which gives value to one of the 7 segments displays. The address (2-4 bit) decoder decodes these 2 bits to 4 statuses such as 10 writes value to the third segment and 11 writes value to the fourth segment.

3. In section 3, the bit 7 of the processors output port is connected to “reg_wr” which is used to enable writing to the buffer or disable writing on the display.
For testing to take place, numbers of push buttons are connected to the CPU inputs. The 24-bit CPU input ports are connected to specified pins as shown in the schematic below:
Summary/Conclusion
The design, simulation and implementation of this simple microcontroller (CPU) has been performed correctly and it has fulfilled the basic requirements of the assessment brief. However due to the time and limitations of writing the assembly code to program the microprocessor for display peripheral was not fully achieved. Author has acquired basic skills and knowledge required to use a hardware description language (VHDL) in Altera Cyclone III FPGA (EP3C10E144C7) using Quartus II software for the DADIV method - method to define, analyse, design, implement and verify a basic microcontroller. Programming the microprocessor for adding different peripherals, using assembly code, can be learnt in future. It was not easy; in fact a challenging project based activity.

References