**بسم الله الرحمن الرحيم**

**King Abdulaziz University – Faculty of Engineering - EE-460**

**Digital Design II**

**Design Assignment#2**

Design of a Simple 8-bits Processor

|  |  |
| --- | --- |
| **Name** | **ID** |
| Muhannad Saeed Alghamdi | **1846525** |

Contents

[Introduction 3](#_Toc83399878)

[Design of the 8-bit Processor 3](#_Toc83399879)

[Top-Level Design 4](#_Toc83399880)

[Verilog Code 5](#_Toc83399881)

[The Design of the Datapath 5](#_Toc83399882)

[A) The Register File………………………………..…………………………………………………………………………………..6](#_Toc83399883)

[B) The 4x1 multiplexer……………………………………………………………………………………………………………….8](#_Toc83399884)

[C) Arithmetic and Logic Unit (ALU)…………………………………………………………………………………………….9](#_Toc83399885)

[The design of the Control Unit 11](#_Toc83399886)

[Testing and Validation 12](#_Toc83399887)

[Conclusion 14](#_Toc83399888)

# Introduction

As a part of the digital design II course, and to introduce us to the digital design strategies, we were asked to design a simple 8-bit processor with eight registers to perform eight different operations. It is required to follow all the design steps starting from the top-level design, and the interaction between the Datapath and the Control Unit including the modules, and the test unit for validating our design's results. The processor must be able to perform eight different operations, loading an 8-bits value to a register, adding the values of two register or subtracting them, moving a value from one register to another, and the logical operations AND, OR and XOR, and to store a register value to the output of the processor.

# Design of the 8-bit Processor

As in this assignment we were asked to design a simple 8-bits processor, with a register file of 8 registers each 8-bit size. This processor has 8 operations load, move, adding, subtracting, and, or, xor and storing Table-1. Resulting an opcode of 3 bits Table-2. To address one of the eight available registers in the register file we need 3 bits for the source register and another 3 bits for the destination register. The 3 bits opcode, 3 bits address of the source register and the 3 bits for the address of the destination register is combined together in a 9 bits function format as in Fig.1.

Table 1 the description for the operations of the 8-bit processors

|  |  |
| --- | --- |
| **Operation** | **Description** |
| load Rx, dataIn | Rx ← dataIn |
| mov Rx, Ry | Rx ← [Ry] |
| add Rx, Ry | Rx ← [Rx] + [Ry] |
| sub Rx, Ry | Rx ← [Rx] − [Ry] |
| and Rx, Ry | Rx ← [Rx] & [Ry] |
| or Rx, Ry | Rx ← [Rx] | [Ry] |
| xor Rx, Ry | Rx ← [Rx] ⊕ [Ry] |
| store Rx, dataOut | Rx → dataOut |

Table 2 the opcode for each operation of the processor

|  |  |
| --- | --- |
| **operation** | **opcode** |
| **load** | 000 |
| **mov** | 001 |
| **add** | 010 |
| **sub** | 011 |
| **and** | 100 |
| **or** | 101 |
| **xor** | 110 |
| **store** | 111 |

A picture containing timeline

Description automatically generated

Figure 1. the format of the 9-bits function

## Top-Level Design

For an 8 bits processor to work appropriately there must be two 8 bits ports one for the input and another for the output. In our design we call the 8 bits input dataIn, and the 8 bits output dataOut. For the 9 bits function input which its format shown in Fig.1 will include the two registers used in the operation with the required operation code known as opcode, see Table-2 for the assigned opcode for each operation. The design will need a clock which is used only for writing, as for reading the design acts as a combinational design. Fig.2 shows the top-level design of the 8 processors with the input and output signals.

Diagram

Description automatically generated

Figure 2 the top-level design of the 8-bit processor

Getting a closer look to the design, Fig.3 shows the interaction between the Datapath and the control unit inside the processor. The dataIn input goes immediately to the Datapath, while the function is splits into two segments the first 3 bits go to the control unit to generate the required control signals based on the input opcode. The remaining 6 bits go to the Datapath to specify the addresses of the used registers 3 bits for each address. Then after executing the operation the output of the Datapath will be the dataOut of the processor.

Text

Description automatically generated

Figure 3 the processor block diagram with interaction between the Datapath and Control Unit

## Verilog Code

Until now we have discussed only the top-level design of the 8-bit register, Fig.4 shows the source code of the top-level design including the interaction between the Datapath and the Control Unit.

Figure 4 the source code for the top-level module

Text, letter

Description automatically generated

## The Design of the Datapath

So far we discussed the discussed the top-level design of the 8-bit processor, and the interaction between the Datapath and the Control Unit. Fig.4 shows the design of the Datapath including the three signals coming from the control unit. The inputs Ra and Rb represent the operands registers used in any operation of the processor. We see that the input Rb is connected also to the Rw which is the address of write destination register. The port dataW is connected to the output of the mux which is the data that will be written to Rw in case WE was one. The outputs of the register file dataA and dataB then will be connected to the ALU inputs to perform the selected operation through selOp port. The output of the ALU then connected to one of the inputs to the MUX with dataIn, dataA and dataB. The selection for the MUX will be based on the signal coming to the port distSrc. The output of the MUX is connected to the output port for the processor and back to the dataW port in the register file. For the purpose of getting a detailed view of the used components we will discuss each component individually and its Verilog module in details.

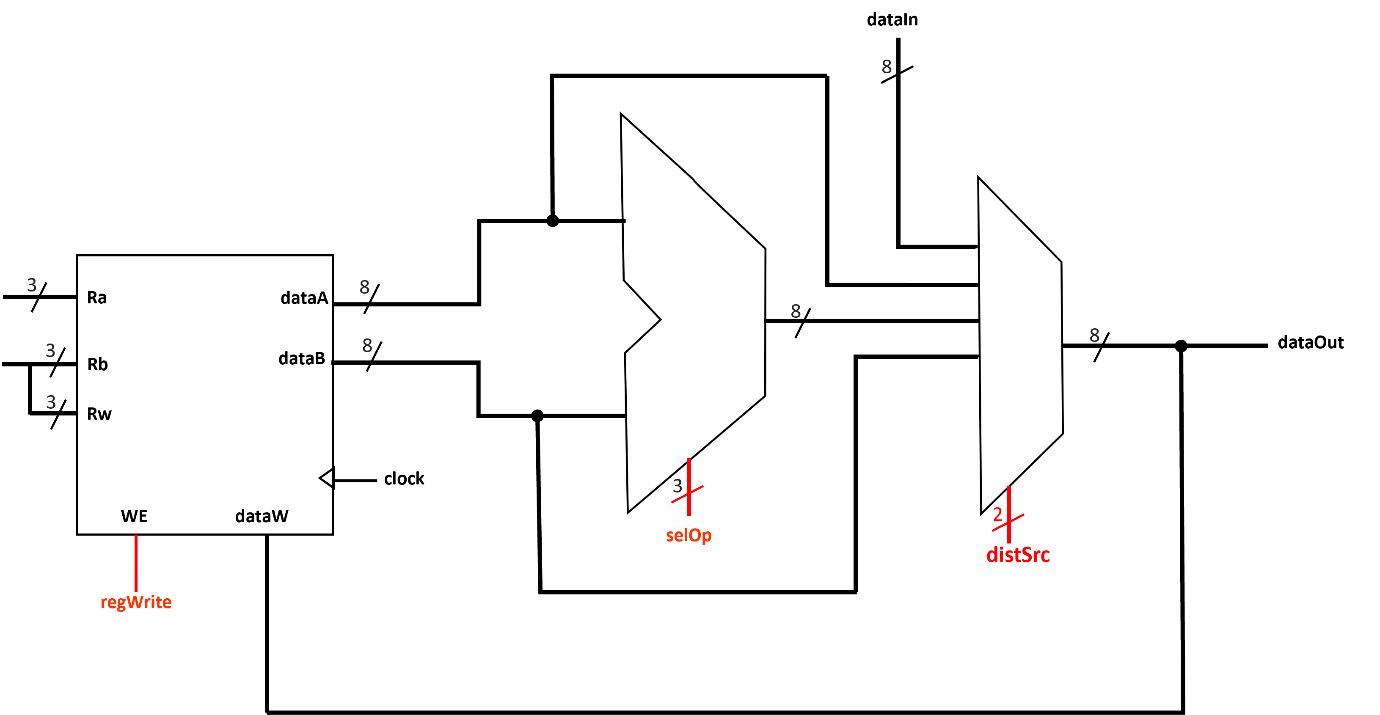


Figure 4 the design of the Datapath

### The Register File

The register file shown in Fig.5 consists of 8 registers each with 8 bits size. For this purpose, we need 3-bits to address a register, in this design we have two inputs Ra and Rb for the addresses of two registers. The data of the two addressed registers will be outputted to the output pins dataA and dataB. The Rw is the address of the destination register used to write the data coming from the dataW to it. In a single cycle two registers will be read and one be written to if the WE was 1.

Graphical user interface, application, Teams

Description automatically generated

Figure 5 register file of 8x8 bits registers

Figure 5 register file of 8x8 bits registers

Fig.6 shows the Verilog code used to create the register file, note that the chosen address size is 3 which implies that we have 8 registers in this register file, each with 8-bits. All registers start with initial value of eight bits zeros as shown in the initial block.

Figure 6 the source code for the register file

Text

Description automatically generated

### The 4x1 multiplexer

In this design the multiplexer used to select the source of the output data, as shown in the Datapath design on Fig.4, there are four possible sources the dataIn, dataA, ALUout and dataB. Each input is an 8-bits input and to choose one of the four inputs to the MUX we need 2-bits select port as shown in Fig.7 distSrc used as 2-bits select signal for the MUX.

A picture containing night sky

Description automatically generated

Figure 7 the 4x1 MUX

The MUX selections with respect to the different possible distSrc signal shown in the Table-3. The corresponding Verilog module used for the 4x1 MUX is shown in Fig.8

Table 3 the truth table of the 4x1 MUX

|  |  |
| --- | --- |
| **operation** | **sel** |
| dataIn | 00 |
| dataA | 01 |
| ALUout | 10 |
| dataB | 11 |

Figure 8 the 4x1 MUX module

Table

Description automatically generated

### Arithmetic and Logic Unit (ALU)

in this design, the processor performs eight possible operations, five of them are considered Logical or Arithmetic operations. Which are performed in the ALU. To select one of the possible operations we need an input port of 3-bits. Fig.9 shows the block diagram of the ALU with its input and outputs port. The selection of the operation performed follows the Table-4

A picture containing graphical user interface

Description automatically generated

Figure 9 the ALU block diagram with 3-bits select input

Table 4 the operations of the ALU

|  |  |
| --- | --- |
| **operation** | **selOp** |
| add | 000 |
| sub | 001 |
| and | 010 |
| or | 011 |
| xor | 100 |

the Verilog module used for the ALU is shown in Fig.10. from the source code we notice that we have five operations, and the default case is eight bits zeros.

Figure 10 the Verilog module for the ALU

Text

Description automatically generated

To conclude our discussion of the top-level design for the Datapath and the details of each component used, Fig.11 shows the overall module used for the Datapath.

Figure 11 the Verilog module for the Datapath

Text, letter

Description automatically generated

## The design of the Control Unit

A picture containing night sky

Description automatically generatedPreviously, in the Datapath design we have discussed the need of three control signals to control three different components in the Datapath. The three signals as in Fig.12 are regWrite, selOp and distSrc. The regWrite signal controls the operation of writing the output data to the register file one means enable writing to the register addressed using Rw port and zero means not to perform the writing operation. While the 3-bits control signal selOp controls the operation performed in ALU according to Table-5. The distSrc signal chooses which of the MUX inputs is the desired output based as shown also in Table-5.

Figure 12 the block diagram of the Control Unit

|  |  |  |  |
| --- | --- | --- | --- |
| **operation** | **regWrite** | **selOp** | **destSrc** |
| load | 1 | 000 | 00 |
| mov | 1 | 000 | 01 |
| add | 1 | 000 | 10 |
| sub | 1 | 001 | 10 |
| and | 1 | 010 | 10 |
| or | 1 | 011 | 10 |
| xor | 1 | 100 | 10 |
| store | 0 | 000 | 11 |

Table 5 the operation and the corresponding control signals

The module for the Control Unit is shown in Fig.13.

Figure 13 the Verilog module for the Control Unit

Text, table

Description automatically generated

# Testing and Validation

For testing the 8-bit processor, the testbench shown on Fig.14 is used to validate all the operations this processor can perform.

Figure 14 the Testbench used to test the 8-bit processor

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

the simulation waveform shown in Fig.15, the timeline from 0-50ns followed by the rest of the waveform. At 10ns the falling edge of the clock comes and from the function code we see it is a load operation of value to the register R1. At the next falling edge of the clock we see that the function input has a value of 000000000 and the input data is 5, corresponds to loading a value 5 to R0. The next active is at 30ns where the function is 010001000 which corresponds to adding the value stored in R1, to the value in R0 storing the results in R1. We see the output in decimal is 9 which is the result of adding the 4 and 5 stored in R1 and R0. Writing this value occurs on the raising edge of the clock cycle. Now the value store in the R1 is 9 (in decimal). At 40ns the function is 001011001 which is a mov operation, from R1 to R3. As the result stored in R1 is 9 (in decimal) we see that the output data is 9 which will be written to R3 in the raising edge at 45 ns. At 50 ns, the given function corresponds to a subtraction operation R0-R1 and storing the result in R0, the values stored in R1 and R0 before the falling edge are R0=5 and R1=9, the expected result is -4 in the 2's complement form which is (11111100) which is equivalent to the one in the wave form. Now, the R0 is (11111100) which will an operand to an AND with R1 which is (00001001). Doing so, the result is (00001000) which is shown in the waveform at 60ns, and the result is stored in R0. So, the next operation is done at 70ns R0 OR R1 results (00001001) stored in R0. At 80ns the operation is an XOR between the two registers R0, R1 (00001001) XOR (00001001) is (00000000) stored in R0. At 90ns we store the value of the R3 which we move the value (00001001) to it previously, to the dataOut port.

Figure 15 the waveform for testing the 8-bit processor

Chart

Description automatically generated with medium confidenceA picture containing graphical user interface

Description automatically generated

# Conclusion

To conclude, we have discussed the design of an 8-bit processor with eight possible operations. As the simulation shows, we have successfully fulfilled all the requirement by following all the steps for the digital design. In this assignment, we have learned some strategies used in the digital design and we followed those steps to successfully design the 8-bit processor. Furthermore, we have used Quartus to write the Verilog modules and the testbench. We also used Multisim as a waveform viewer to validate our results.