2/2022

Simple 8-bit Processor

Section: CA

Instructor: Dr. Mohammad Awedh

Name	ID
Asaad Waleed Dadoush	1766585

Contents

Figures

List of tables

Introduction:

This design assignment is about to extend an 8-bit processor that can perform multiple operations, it extends to the old design that has load and addition, subtraction, so now it has more operations like or, xor, and, store, so the process has an 8-bit register file with 8-bit data input, so there are eight registers in the register file, so for the perform the operation it's okay to use two of the registers, so for the top-level design, there are 2 top modules the control unit and the datapath, starting with the datapath so for all the operations described above its all bullied in inside the datapath to make sure all the datapath flow through a protected system, for the control unit its very important to regulate the data flow in the datapath, also the control unit receive signals from the datapath to select the right decision before sending the data back to the datapath so it's very clear now the design is divided into 2 main units the datapath and control unit.

Top-level block diagram:

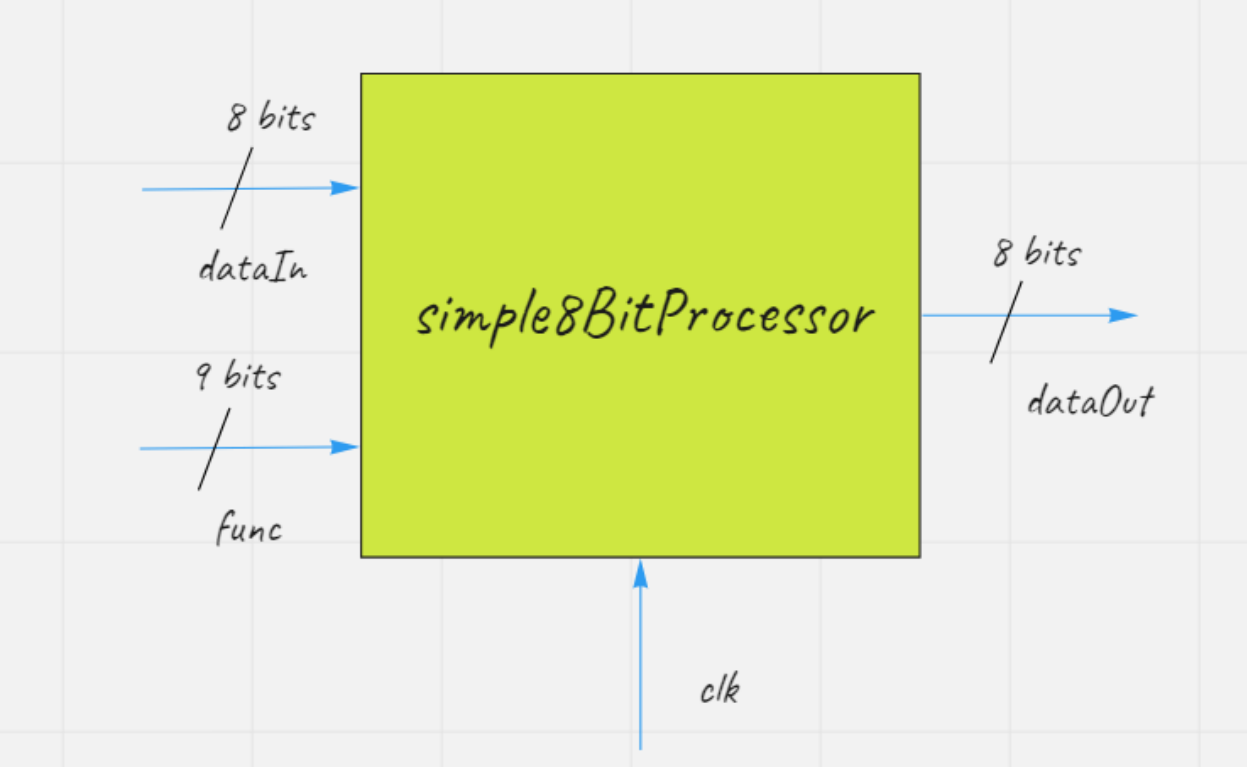

Figure 1: Top-level Design

The figure above is the top-level design for the simple 8-bit processor.

Simple 8-bit processor:

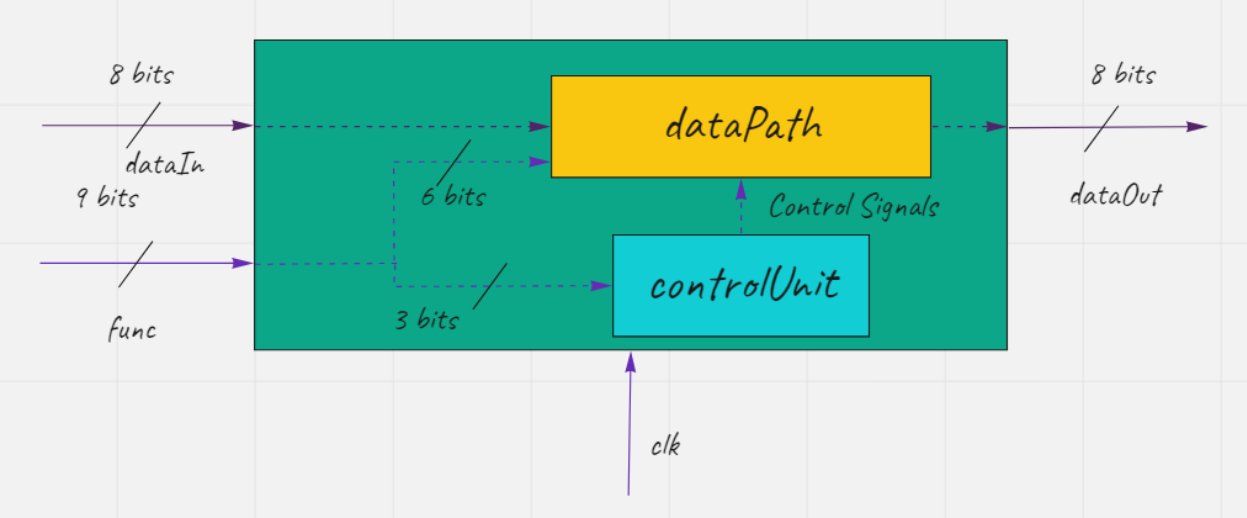

Figure 2: Simple 8-bit processor design

The figure above it goes more depth inside the simple 8-bit processor as shown there are two main components the datapath and control unit.

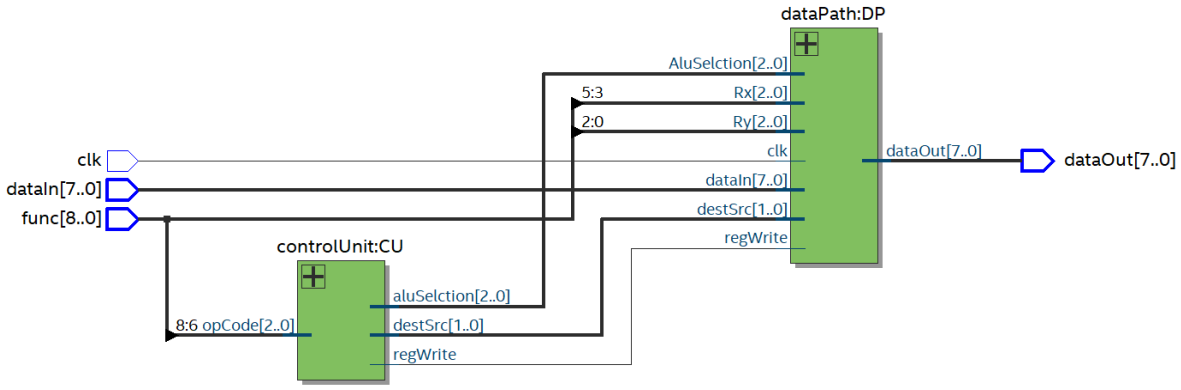

Figure 3: Simple 8-bit processor synthesis

Datapath:

This section will describe the datapath from the inside to see how it works and what are the main components that use to build the datapath.

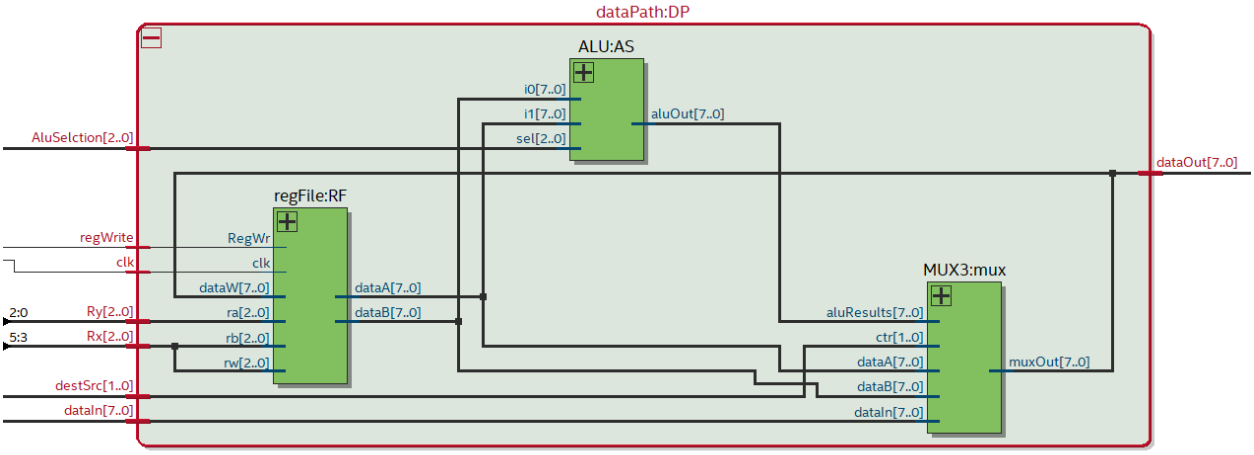

Figure 4: Datapath synthesis

The figure above shows inside the datapath, as shown above, there are three main components the multiplexer, ALU, register file each of those as shown has a control signal coming from the control unit as shown.

controlUnit:

the control unit is very important for the simple 8-bit processor without the design can't work or perform the task that meant to be so in this section will go inside the control unit and describe it.

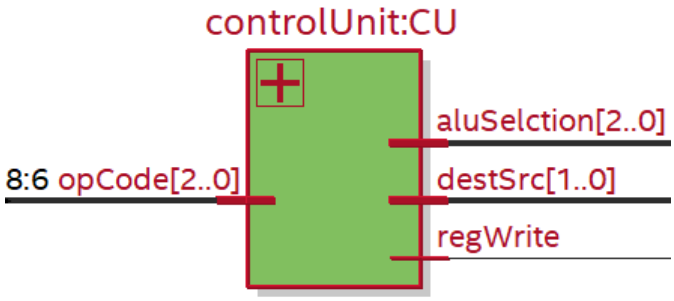

Figure 5: ControlUnit synthesis

The control unit has 6-bits of outputs and 3-bits for inputs, the input bit for function input as shown previously back in the top-level to choose the operation, for the output the 3-bits for the ALU to choose between the arithmetic and logical operations, 2 bits for the MUX to choose which output will choose based on the instruction type and the last bit for the write on the register in the register file the control unit have 8 different signals for the instructions.

Table 1: Simple 8-bit processor operation

opCode	Operation	regWrite	ALU	MUX
000	**Load**	**1**	**XXX**	**00**
001	**Move**	**1**	**XXX**	**01**
010	**Add**	**1**	**000**	**11**
011	**Sub**	**1**	**001**	**11**
100	**And**	**1**	**010**	**11**
101	**Or**	**1**	**011**	**11**
110	**Xor**	**1**	**100**	**11**
111	**Store**	**0**	**XXX**	**10**

Testbench:

```
`timescale1 ns / 1 ps
module testProc;
    // Inputs
    reg[7:0] dataIn;
    reg[8:0] func;
    reg clock;
    // Outputs
    wire[7:0] dataOut;

    // Instantiate Unit Under Test(UUT)
    simple8BitProcessor uut(.dataIn(dataIn),
        .dataOut(dataOut),
         .func(func),
         .clk(clock)
     );
    always
        # 5 clock = ~clock;
    initial begin
    func = 0;clock = 0;dataIn = 0;
    @(posedge clock);
        dataIn = 5;
        func = 9'b000_000_XXX; // Load 5 in R0    ===== Result = 5 ===== R0 = 5
        @(posedge clock);
        dataIn = 7;
        func = 9'b000_001_XXX; // Load 7 in R1    ===== Result = 7 ===== R1 = 7
        @(posedge clock);
        dataIn = 3;
        func = 9'b000_010_XXX; // Load 3 in R2    ===== Result = 3 ===== R2 = 3
        @(posedge clock);
        func = 9'b001_001_010; // Mov R2 to R1    ===== Result = 3 ===== R1 = 3
        @(posedge clock);
        func = 9'b010_001_000; // Add R1 to R0    ===== R1 + R0 = 8 ===== R1 = 8
        @(posedge clock);
        func = 9'b011_001_010; // Sub R2 from R1 ===== R1 - R2 = 5 ===== R1 = 5
        @(posedge clock);
        dataIn = 11;
        func = 9'b000_111_000; // Load 10 in R7   ===== Result = 11 ===== R7 = 11
        @(posedge clock);
        dataIn = 7;
        func = 9'b000_110_000; // Load 15 in R6   ===== Result = 7 ===== R6 = 7
        @(posedge clock);
        func = 9'b100_111_110; // And R7 with R6 ===== R7 & R6 = 3 ==== R7 = 3
        @(posedge clock);
        func = 9'b101_111_110; // or R7 with R6   ===== R7 | R6 = 7 ==== R7 = 7
        @(posedge clock);
        func = 9'b110_110_001; // Xor R6 with R1 ===== R6 ^ R1 = 2 ===== R6 = 2
        # 10
        @(posedge clock);
        func = 9'b111_111_XXX; // Store R7        ====== Results = 7 ==== R7 = 7
        # 10
        $end;
    end
endmodule
```

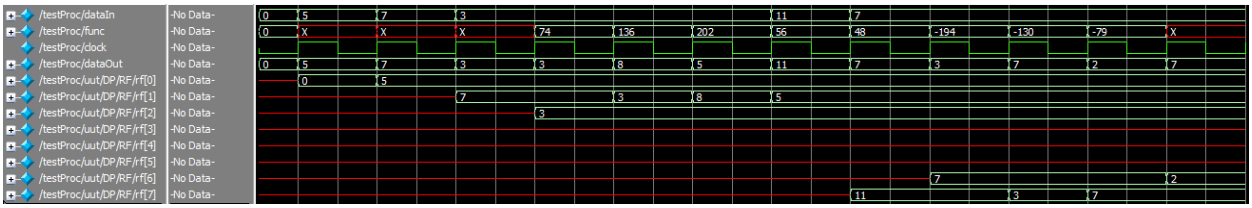

Figure 6: Testbench Waveform Results

As shown in figure 6 the results for the testbench code are matching so the design for the simple 8-bit processor working correctly.

Conclusion:

By the end of this assignment what was learned about its was helpful how to apply the design steps starting from the top level to the small details until to finish the whole design, and how to apply the basics of Verilog that was learned in EE-460 class, and become beyond that and apply that knowledge to make my design I think that design was just a preparation to what is coming next so that now I have a better knowledge about the designing process and how to apply I think that will help in the future projects and designs.