Laboratory No. 9:
Simple Calculator (Final Lab)

By: Derek Hildreth

Instructor: Brother Fisher

BYU-Idaho CompE 340

July 16, 2009

I. INTRODUCTION

A. Purpose

Students will combine the sub-modules that were created in the previous lab exercises to complete the SIMPLE_CALC module. They will also use a Verilog generate statement to choose between the two versions of the COMP module. Students will develop a test bench that thoroughly verifies the structural module. They will also modify the ROM contents to hold the specific values that you will use in the verification process. Finally, they will implement the logic, targeting the Spartan-3 FPGA, and examine the contents of the report files.

The schematic for the calculator is shown in Figure 4 below. Students will write the RTL description for the top-level module SIMPLE_CALC.

B. Equipment

There is a minimal amount of equipment to be used in this lab. The few requirements are listed below:

- Xilinx ISE Navigator Software (v10.0.1)
- Spartan 3E Developer Board
- Computer capable of running the software mentioned

C. Procedure

1. Start the ISE Navigator.
2. Create a new project.
3. Import the .v source files for the MEM, COMP_BEH, COMP_RTL, ALU and CNTRL_FSM sub-modules
4. Modify the source code for the MEM module, updating the ROM contents by using the data that you recorded earlier for the ALU verification.
5. Write Verilog code and check syntax for the SIMPLE_CALC module.
6. Create generate statements that choose either the COMP_BEH or COMP_RTL sub-modules. Recall that the COMP_BEH module is written in a behavioral style – intended only for simulation – and that the COMP_RTL module is written in an RTL style for synthesis.
7. Perform a syntax check.
8. Create a test bench for SIMPLE_CALC and run a simulation to verify functionality.
9. Implement (Place & Route) the design and examine the contents of the MAP and PAR report files.

II. SCHEMATIC DIAGRAMS

This section consists of block diagrams which are useful for the laboratory procedure as well as simulation results.

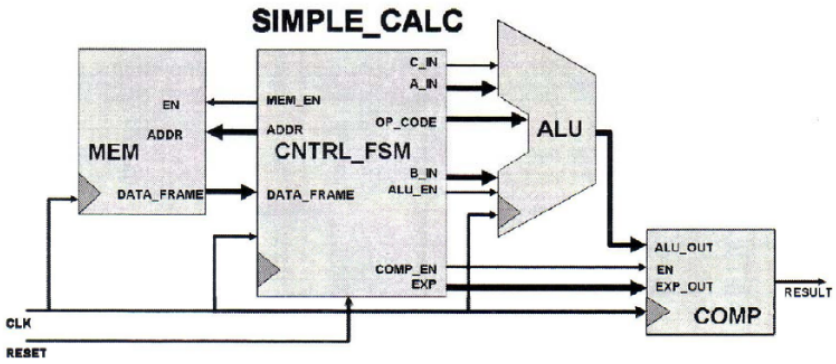

FIG. 1: Overall block diagram of the calculator project showing the relationship of all of the components.

FIG. 2: RTL view for the simple calculator.

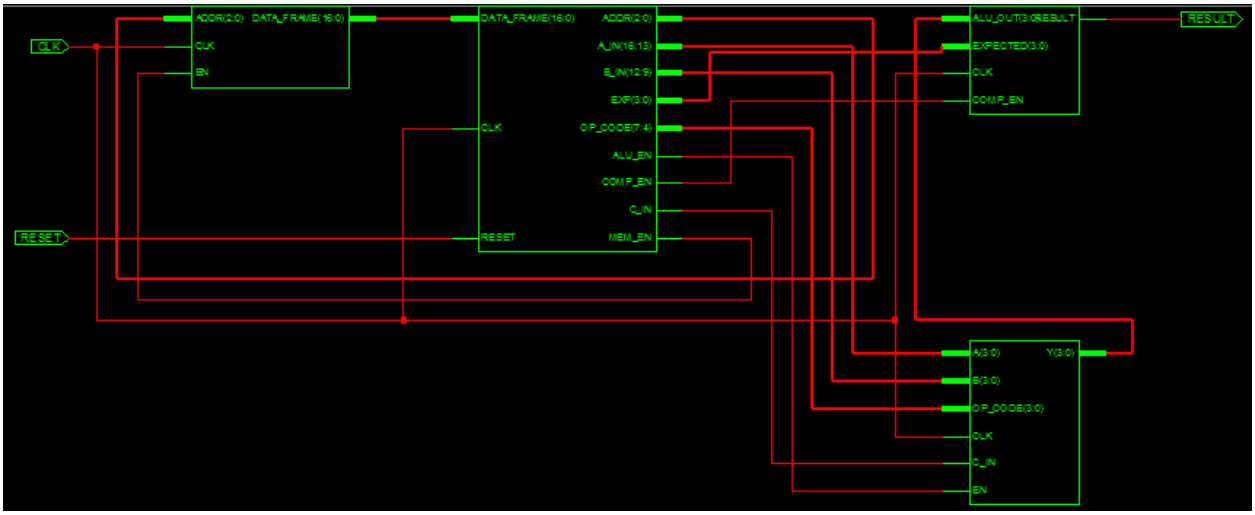

FIG. 3: Detailed RTL view for the simple calculator with all modules.

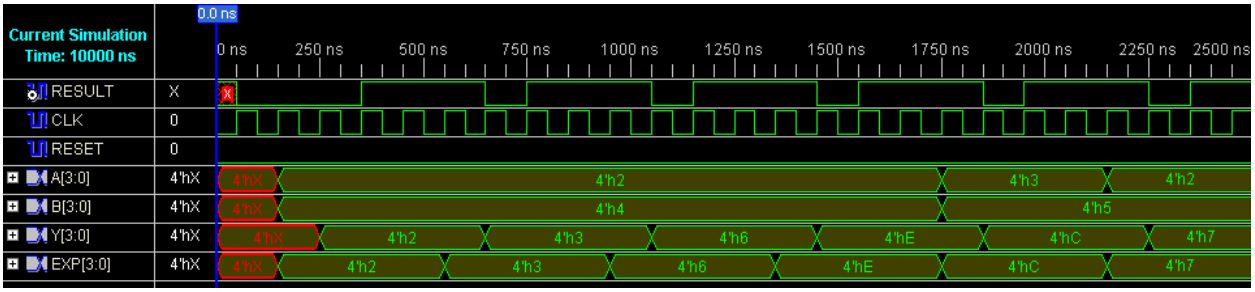

FIG. 4: Simulation results for the entire simple calculator.

III. EXPERIMENT DATA

This section will consist of the code blocks that created the counter block as seen in Figure 4.

A. Main Simple Calculator Module

This is the verilog simple calculator file called (`SIMPLE_CALC.v`).

```
`timescale 1ns / 1ps

`define SYNTH 1

module SIMPLE_CALC(
    input CLK,
    input RESET,
    output RESULT
    );

    wire[3:0] A_IN, B_IN, OP_CODE, EXP_OUT, ALU_OUT;
    wire C_IN, MEM_EN, COMP_EN, ALU_EN;
    wire [2:0] ADDR;
    wire [16:0] DATA_FRAME;

    ALU U0(.A(A_IN), .B(B_IN), .C_IN(C_IN), .OP_CODE(OP_CODE), .CLK(CLK), .EN(ALU_EN), .Y(ALU_OUT));

    CNTRL_FSM U2(.RESET(RESET), .CLK(CLK), .DATA_FRAME(DATA_FRAME), .A_IN(A_IN), .B_IN(B_IN), .C_IN(C_IN), .OP_CODE(
        OP_CODE),
        .EXP(EXP_OUT), .ALU_EN(ALU_EN), .MEM_EN(MEM_EN), .COMP_EN(COMP_EN), .ADDR(ADDR));

    MEM U3(.ADDR(ADDR), .DATA_FRAME(DATA_FRAME), .CLK(CLK), .EN(MEM_EN));


    generate
        if (`SYNTH)
            //COMP_RTL U4(COMP_EN, EXPECTED, ALU_OUT, CLK, RESULT);
            COMP_RTL U4(.COMP_EN(COMP_EN), .EXPECTED(EXP_OUT), .ALU_OUT(ALU_OUT), .CLK(CLK), .RESULT(RESULT));
        else
            //COMP_BEH U5(COMP_EN, EXPECTED, ALU_OUT, CLK, RESULT);
            COMP_BEH U5(.COMP_EN(COMP_EN), .EXPECTED(EXP_OUT), .ALU_OUT(ALU_OUT), .CLK(CLK), .RESULT(RESULT));
    endgenerate

endmodule
```

This is the text fixture for the simple calculator called (`SIMPLE_CALC_TB.v`).

```
`timescale 1ns / 1ps

module SIMPLE_CALC_TB;

    // Inputs
    reg CLK;
    reg RESET;

    // Outputs
    wire RESULT;

    // Instantiate the Unit Under Test (UUT)
    SIMPLE_CALC uut (
        .CLK(CLK),
        .RESET(RESET),
        .RESULT(RESULT)
    );

    initial begin
        // Initialize Inputs
        CLK = 0;
        RESET = 0;

        #1 RESET = 1;
        #1 RESET = 0;

        // Wait 100 ns for global reset to finish
        #100;

    end

    always
        #50 CLK = ~CLK;

endmodule
```

This is the pin out information for the hardware simple calculator called `Full.ucf`.

```
#PACE: Start of Constraints generated by PACE

#This ucf file has all the switches and LEDs that might be used in CompE 224
```

```
#Modify this file with your net names and comment out or delete the lines not needed.

#PACE: Start of PACE I/O Pin Assignments

#Main Switches
#NET "SW3"   LOC = "N17"   ;
#NET "SW2"   LOC = "H18"   ;
#NET "SW1"   LOC = "L14"   ;
#NET "SW0"   LOC = "L13"   ;

#Extra Switches
#NET "SW7"   LOC = "A6"   ;
#NET "SW6"   LOC = "B6"   ;
#NET "SW5"   LOC = "E7"   ;
#NET "SW4"   LOC = "F7"   ;

#Main LED's
NET "RESULT"   LOC = "F9"   ;
#NET "LED6"   LOC = "E9"   ;
#NET "LED5"   LOC = "D11"   ;
#NET "LED4"   LOC = "C11"   ;
#NET "LED3"   LOC = "F11"   ;
#NET "LED2"   LOC = "E11"   ;
#NET "LED1"   LOC = "E12"   ;
#NET "LED0"   LOC = "F12"   ;

#Extra LED's
#NET "LED11"   LOC = "B4"   ;
#NET "LED10"   LOC = "A4"   ;
#NET "LED9"    LOC = "D5"   ;
#NET "LED8"    LOC = "C5"   ;



#Pushbutton Switches
#NET "BTN_North"   LOC = "V4"   | PULLDOWN   ;
NET "RESET"    LOC = "H13" | PULLDOWN   ;  # East
NET "CLK"   LOC = "K17" | PULLDOWN   ;  #South
#NET "BTN_West"   LOC = "D18"   | PULLDOWN   ;

NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE;

#Debouncer C9_50MHz
#NET "CLK"    LOC = "C9"   ;

#Used for correction of error with refrence to none dedicated clock inputs
#Need to input the net name in place of switch inside the ""
#NET "switch" CLOCK_DEDICATED_ROUTE = FALSE;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

This is the map report for the simple calculator called `Map_Report.txt`.

```
Release 10.1.02 Map K.37 (nt)
Xilinx Mapping Report File for Design 'SIMPLE_CALC'

Design Information
------------------
Command Line   : map -ise C:/CompE340/Lab09/Lab09.ise -intstyle ise -p
xc3s500e-fg320-4 -cm area -pr off -k 4 -c 100 -o SIMPLE_CALC_map.ncd
SIMPLE_CALC.ngd SIMPLE_CALC.pcf
Target Device  : xc3s500e
Target Package : fg320
Target Speed   : -4
Mapper Version : spartan3e -- $Revision: 1.46.12.2 $
Mapped Date    : Wed Jul 15 20:20:56 2009

Design Summary
--------------
Number of errors:      0
Number of warnings:    1
Logic Utilization:
  Number of Slice Flip Flops:            20 out of   9,312    1%
  Number of 4 input LUTs:                40 out of   9,312    1%
Logic Distribution:
  Number of occupied Slices:             23 out of   4,656    1%
    Number of Slices containing only related logic:     23 out of      23 100%
    Number of Slices containing unrelated logic:         0 out of      23   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:          40 out of   9,312    1%
  Number of bonded IOBs:                  3 out of     232    1%
  Number of BUFGMUXs:                     1 out of      24    4%

Peak Memory Usage:  153 MB
Total REAL time to MAP completion:  6 secs
Total CPU time to MAP completion:   3 secs

!--- RESULTS SNIPPED FOR LAB REPORT ---!
```

B. Individual Modules

This is `alu.v`.

```
`timescale 1ns / 1ps

module ALU(
    input [3:0] A,
    input [3:0] B,
    input C_IN,
    input [3:0] OP_CODE,
    input CLK,
    input EN,
    output reg [3:0] Y
    );

always @ (posedge CLK)

   if(EN)
   begin
      case (OP_CODE) // Specify outputs for each address.
         4'b0000:
            Y = A + C_IN;
         4'b0001:
            Y = A + B + C_IN;
         4'b0010:
            Y = A + (~B) + C_IN;
         4'b0011:
            Y = A - 1 + C_IN;
         4'b0100:
            Y = A & B;
         4'b0101:
            Y = A | B;
         4'b0110:
            Y = A ^ B;
         4'b0111:
            Y = ~A;
         4'b1000:
            Y = 0;
         default:
            Y = 0;
      endcase
   end
endmodule
```

This is `CNTRL_FSM.v`.

```
`timescale 1ns / 1ps

module CNTRL_FSM(
    input RESET,
    input CLK,
    input [16:0] DATA_FRAME,

    output reg [16:13] A_IN,
    output reg [12:9] B_IN,
    output reg C_IN,
    output reg [7:4] OP_CODE,
    output reg [3:0] EXP,

    output reg ALU_EN,
    output reg MEM_EN,
    output reg COMP_EN,
    output reg [2:0] ADDR
    );

localparam[4:0]
   S0_INIT   =  5'b00001,
   S1_FETCH  =  5'b00010,
   S2_ALU    =  5'b00100,
   S3_COMP   =  5'b01000,
   S4_DONE   =  5'b10000;  // Explicite make sure condition to stay in state

reg [4:0] CURR_STATE;
reg [4:0] NEXT_STATE;
reg [2:0] ADDR_I;  // used to increment ADDR internally.

always @ (posedge CLK, posedge RESET) // current state proc block
begin
   //$display("Current State: %b @ ADDR_I: %d", CURR_STATE, ADDR_I);
   if (RESET == 1'b1)  // if reset asserted
      begin
         CURR_STATE <= S0_INIT; // Reset to state 0
         ADDR <= 3'b000;
      end
   else
      begin
         CURR_STATE <= NEXT_STATE; // goto next state
         ADDR <= ADDR_I;
      end
end

always @ (CURR_STATE)
begin

   A_IN = DATA_FRAME[16:13];
   B_IN = DATA_FRAME[12:9];
```

```
    C_IN = DATA_FRAME[8];
    OP_CODE = DATA_FRAME[7:4];
    EXP = DATA_FRAME[3:0];

    ADDR_I = ADDR; // assignment prevents latch inference.

    case (CURR_STATE)
        S0_INIT:
            begin
                ALU_EN  = 1'b0;
                MEM_EN  = 1'b0;
                COMP_EN = 1'b0;

                NEXT_STATE = S1_FETCH;
            end
        S1_FETCH:
            begin
                ALU_EN  = 1'b0;
                MEM_EN  = 1'b1;
                COMP_EN = 1'b0;

                NEXT_STATE = S2_ALU;
            end
        S2_ALU:
            begin
                ALU_EN  = 1'b1;
                MEM_EN  = 1'b0;
                COMP_EN = 1'b0;

                NEXT_STATE = S3_COMP;
            end
        S3_COMP:
            begin
                ALU_EN  = 1'b0;
                MEM_EN  = 1'b0;
                COMP_EN = 1'b1;

                NEXT_STATE = S4_DONE;
            end
        S4_DONE:
            begin
                ALU_EN  = 1'b0;
                MEM_EN  = 1'b0;
                COMP_EN = 1'b0;

                if (ADDR_I >= 3'b101)
                    begin
                        NEXT_STATE = S4_DONE;
                    end
                else
                    begin
                        NEXT_STATE = S1_FETCH;
                        ADDR_I = ADDR_I + 1;
                    end
            end
        default:
            begin
                ALU_EN  = 1'b0;
                MEM_EN  = 1'b0;
                COMP_EN = 1'b0;

                NEXT_STATE = S0_INIT;
            end
    endcase
end

endmodule
```

This is `comp_beh.v`.

```
`timescale 1ns / 1ps

module COMP_BEH(
    //INPUTS
    input wire COMP_EN,   // 1-bit - enable wire from FSM
    input wire [3:0] EXPECTED, // 4-bit - expected output, loaded from ROM
    input wire [3:0] ALU_OUT,  // 4-bit - actual output from ALU
    input wire CLK ,       // 1-bit - clock input

    //OUTPUTS
    output reg RESULT  // 1-bit - result of comparison
    );


always @ (posedge CLK)
begin
    assign RESULT = EXPECTED == ALU_OUT;

    if (!RESULT)
        $display("At time %t, EXP = %b, ACT = %b, RESULT = %b", $time, EXPECTED, ALU_OUT, RESULT);
end
endmodule
```

This is `comp_rtl.v`.

```
`timescale 1ns / 1ps
```

```
module COMP_RTL(
    //INPUTS
    input wire COMP_EN,   // 1-bit - enable wire from FSM
    input wire [3:0] EXPECTED, // 4-bit - expected output, loaded from ROM
    input wire [3:0] ALU_OUT,   // 4-bit - actual output from ALU
    input wire CLK ,       // 1-bit - clock input

    //OUTPUTS
    output reg RESULT  // 1-bit - result of comparison
    );

always @ (posedge CLK)
   if (ALU_OUT == EXPECTED)
      RESULT = 1'b1;
   else
      begin
         RESULT = 1'b0;
         //display describes simulation mismatch and specifies the simluation time step when mismatch occured
         //$display("At time %t, EXP = %b, ACT = %b, RESULT = %b", $time, EXPECTED, ALU_OUT, RESULT);
      end

endmodule
```

This is `mem.v`.

```
`timescale 1ns / 1ps
`include "MY_HEADER.txt"

module MEM(ADDR, DATA_FRAME, CLK, EN);
//inputs
input CLK, EN;
input [`ADDR-1:0] ADDR;

//Output
output reg [`WIDTH-1:0] DATA_FRAME;


always @ (posedge CLK)
  if (EN)
    case (ADDR)
       3'b000: DATA_FRAME = 17'b0010_0100_0_0000_0010;
       3'b001: DATA_FRAME = 17'b0010_0100_1_0000_0011;
       3'b010: DATA_FRAME = 17'b0010_0100_0_0001_0110;
       3'b011: DATA_FRAME = 17'b0010_0100_1_0010_1110;
       3'b100: DATA_FRAME = 17'b0011_0101_0_0111_1100;
       3'b101: DATA_FRAME = 17'b0010_0101_0_0101_0111;
       3'b110: DATA_FRAME = 17'b0000_0000_0_0000_0000;
       3'b111: DATA_FRAME = 17'b0000_0000_0_0000_0000;
       default: DATA_FRAME = 17'bx;
   endcase


endmodule
```

This is `MY_HEADER.txt`.

```
`define WIDTH 17
`define ADDR 3
```

IV. DISCUSSION & CONCLUSION

The purpose of this lab was to bring together all of the modules from the previous labs together and create a simple calculator. This calculator is pre-programmed with the aritmetic and logicical operations with an expected output which the user give. The actual results are compared and if the calculator worked properly, the output is a high 1. Students now know how to design, build, test, and synthesis a circuit by using verilog HDL language.

Personally, I was happy to have worked on this final project with my partner, Tim Price. Together, we were able to quickly bring all of the modules together, hook them up, simulate, and synthesis the project to the Spartan 3E board. We had some stumbling blocks along the way, but nothing too sever. The one module which was giving us most hassle was the `CNTRL_FSM` module. There were all sorts of strange things it was doing, and we had to modify it a bit from our original one that we did in the previous lab. One in particular that I can remember was the `RST` pin being set to a high asserted reset. When we changed it to be a low asserted, most everything started to work properly.

There were some questions posed in the lab handout that I would like to address here in the conclusion:

- Using the Map Report, answer the following about your design:
  - Number of registers?
    - 20
  - Number of 4-input LUTs?
    - 40
  - Number of I/O blocks?
    - 3
  - Number of global clocks?
    - 1

Other conclusions that I can make is that the overall experience writing in verilog was much nicer and actually a lot of fun, which is something I didn't get with Circuit Maker and Cadence OrCAD. This was just a simple calculator and it demonstrated quite a bit of verilog capability. There's still a lot to learn about verilog and it will be exciting to continue learning it to build custom circuits into FPGAs.