TCES 330, Spring 2018
Laboratory Exercise B
Programmable Processor

General Comments:
This is a two week team exercise similar to Laboratory Exercise A that was explained recently in class. You will work in teams of two; individual submissions will not be accepted. The submission for this exercise will consist of your ModelSim files, a Quartus project, and a written report. Your written report should follow the format of the sample laboratory report posted earlier in the quarter.

Make sure that both team members share the workload for this project as equally as possible. The work you turn in must come from your group only. Do not share your work with other groups.

Problem.

- Implement the six-instruction programmable processor described in Processor2018.pptx using SystemVerilog.

- Your Quartus project should be in a folder called LabB and your top-level module should also be called LabB. Your top-level module will instantiate your processor module (called Processor) and interface to the DE2 board as follows:
  - KEY[2] acts as your system clock; (so we don't wear out KEY[0]
  - KEY[1] acts as a synchronous system reset;
    - Note that reset will not be able reload memory contents, but it should reinitialize your state machine and reset (clear) all your registers (but not your register file).
  - Several internal variables are brought to the top-level for debug and display purposes. These include the program counter, the instruction register, state machine current state, both inputs to the ALU, and the ALU output.
  - HEX3, 2, 1, and 0 always display the current contents of the IR;
  - SW[17:15] determines what HEX 7, 6, 5, and 4 display as follows:
    - 0 => HEX7, HEX6 = PC; HEX5, HEX4 = Current State; Note: for this to correspond to the values you set for your states, you should tell Quartus to use your state machine encoding.
    - 1 => HEX7, 6, 5, 4 = ALU_A (A-side input to ALU)
    - 2 => HEX7, 6, 5, 4 = ALU_B (B-side input to ALU)
    - 3 => HEX7, 6, 5, 4 = ALU_Out (ALU output)
    - 4 => Next State (FSM next state variable, if you use one)
    - 5 Unused
    - 6 Unused
    - 7 Unused
  - Your top level module (LabB) should instantiate a processor module, a multiplexer for selecting displays and a key conditioner (described below). Your top level RTL view should look very close to Figure 1, below.

- Turn in the following sample program compiled and loaded into Instruction Memory:

```
RF[0] = D[0B] - D[1B] + D[06] - D[8A];
D[CD] = RF[0];
HALT
```

Data memory should initially contain

```
D[6]  = 0x10AC
D[B]  = 0xCC05
D[1B] = 0x01B5
D[8A] = 0xA040
```

- You will be provided a high-level ModelSim testbench (in a file called testProcessor.sv – keep this in a separate file), so the Processor module you turn in must use this signature (if your processor does not conform to this signature, change your processor):

```
module Processor( Clk, Reset, IR_Out, PC_Out, State, NextState, ALU_A,
ALU_B, ALU_Out );

input Clk;              // processor clock
input Reset;            // system reset
output [15:0] IR_Out;   // Instruction register
output [4:0] PC_Out;    // Program counter
output [3:0] State;     // FSM current state
output [3:0] NextState; // FSM next state (or 0 if you don't use one)
output [15:0] ALU_A;    // ALU A-Side Input
output [15:0] ALU_B;    // ALU B-Side Input
output [15:0] ALU_Out;  // ALU current output
```

You will want to run testbenches for your various modules, but the runrtl.do and wave.do that you turn in must run my testbench as specified above.

- Make sure that the In System Memory Content Editor can display the contents of your memories instruction memory and your data memory; it cannot be used on your register file (dual ported) memory.

- Try to make sure Quartus recognizes your processor state machine (as a state machine); this may not be possible if you use enums for your state names, however. Note that there will be a second state machine in your Quartus design: the ButtonSync state machine. In Quartus State Machine View you can select which state machine to view in the upper left corner of the state machine window.

- Data memory should be a 256 X 16 Quartus RAM LPM with Memory Content Editor enabled.

- The Register File should be implemented with Verilog memory as discussed in Processor2018.pptx.

- The ALU is as discussed in Lec10.pdf and the Verilog is provided for you.

- The Instruction memory should be a 128 X 16 Quartus ROM LPM with Memory Content Editor enabled.

- The controller state machine should be similar to the one shown in Processor2018.pptx.

- Run TimeQuest and include the sdc file in your Quartus project. Note that there will be two clocks in this project (the 50 MHz clock for the key conditioning circuits and the processor clock derived from the KEY).

- You will be provided a ModelSim testbench. Run this testbench and include a screen shot of the ModelSim Transcript area where the results are printed in your report. Of course you can also include the waveforms in your report, if you wish along with other tests that you might have used.

- Write your report using the same outline as before. You know by now to be very explicit and detailed about your test procedure and test results; go back and review the sample lab report. Photographs are allowed!

When you are done with your project.

Zip the contents of the project folder (LabB) and your written report into a .zip file (the exact name of the zip file is not critical, but should contain the name of at least one of your team members). When your zip file got opened, we should see a top-level folder called LabB. Make sure that you turn in all files necessary for the project and that these files are in the proper locations for compilation. It's a good idea to test this by unzipping your files to a new location and recompile using Quartus. Make sure that the correct *.mif files get loaded and that the In System Memory Content Editor can find the data in your memories; this is often a source of difficulties. Submit the complete package on Canvas. There is a separate place on Canvas to turn in your report.

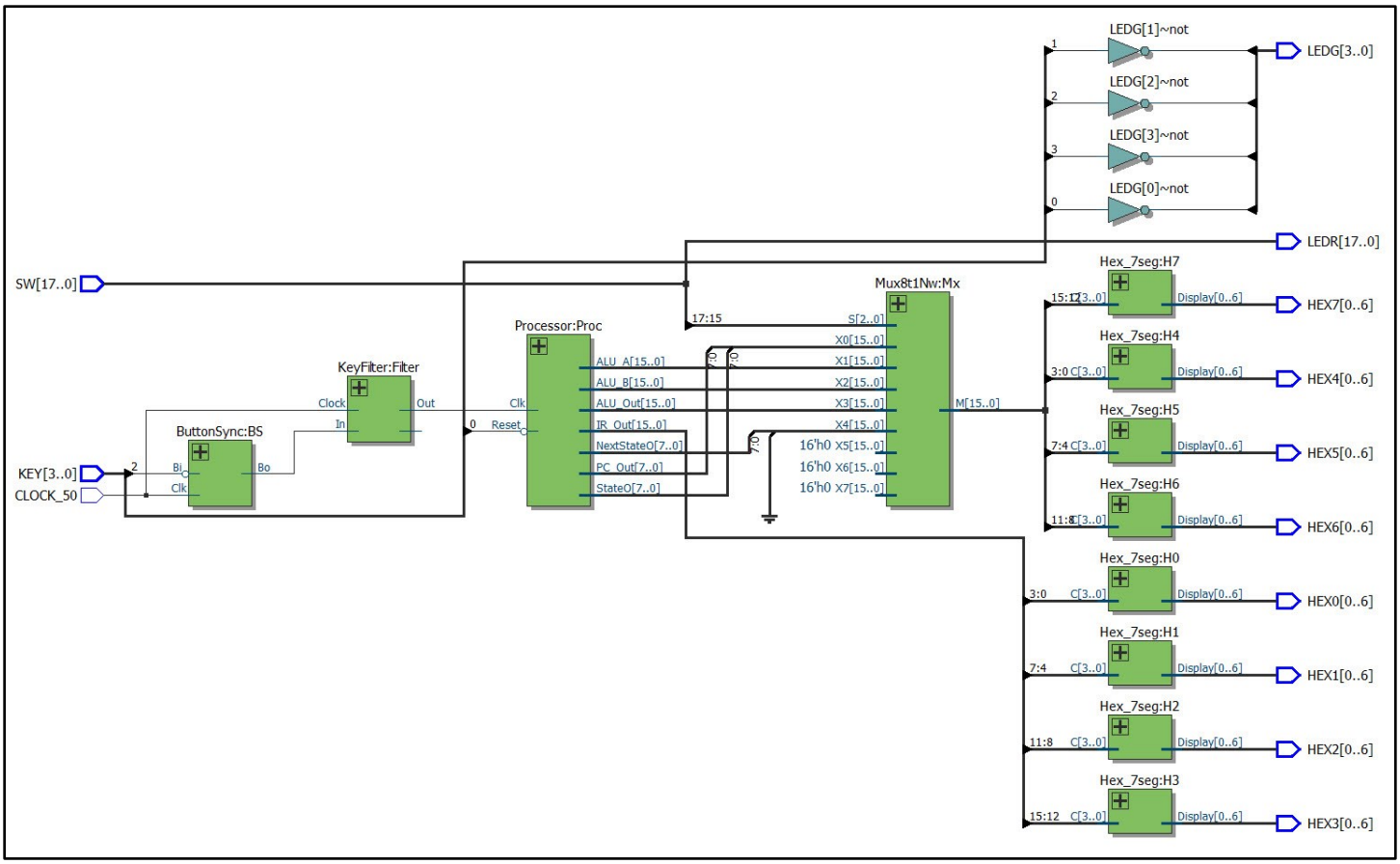

Figure 1. Top-level RTL View