ECE 571- Winter 2020
Homework #1

Create a single .zip or .rar file containing your source code files and transcripts showing that your implementations simulates correctly. Name the file <yourname>_hw1.zip (ex: rkravitz_hw1.zip).

Submit your deliverables to your Homework #1 dropbox no later than 10:00 PM on Sunday 19-Jan-2020. We will only grade the submission with the latest date stamp.

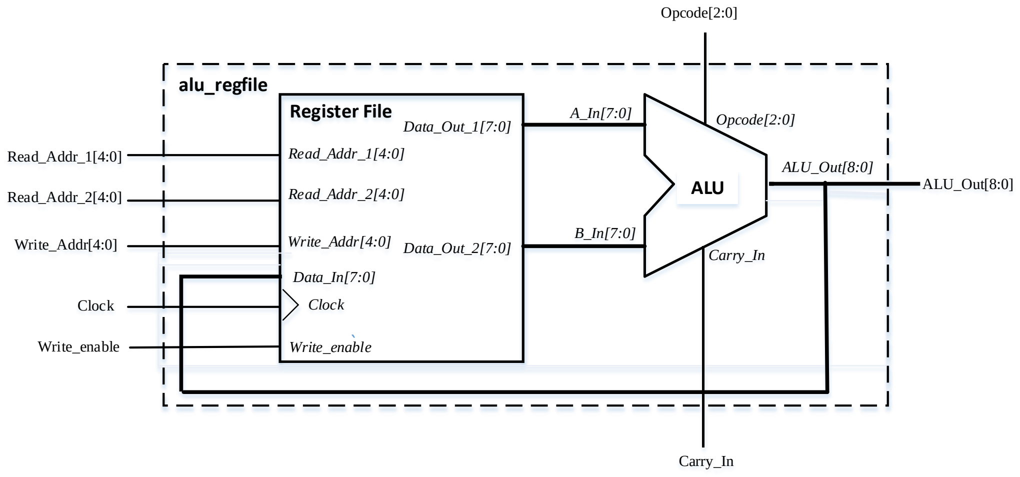
Source code should be structured and commented with meaningful variables. Include a header at the top of each file listing the author and a description. You may use the following template:

```
//////////////////////////////////////////////////////////////
// <filename>.sv - <one line description>
//
// Author:  <your name> (<your email address>)
// Date:    <date you created the code>
//
// Description:
// ------------
// <text description of what function the module performs>
////////////////////////////////////////////////////////////////
```

Acknowledgement: These problems are based on an exercises from *Advanced Digital Design with the Verilog HDL: 2e* by Michael D. Ciletti, Pearson India Education Services Pvt. Ltd, 2017. Modifications for SystemVerilog designed by Roy Kravitz, 2020.

Problem 1 (60 pts)

For this homework problem we are going to create a SystemVerilog model for a simple digital system the includes an ALU (Arithmetic Logic Unit) with inputs taken from a 32 entry x 16 bit wide Register File and the results written back to the Register file.   A block diagram of the system is shown below:



A. (25 pts) Write a SystemVerilog module that implements an 8-bit ALU. The ALU is a block of combinational logic that implements the following functionality (ex: Add Opcode = 3’b000,…Exnor Opcode = 3’b111):

Operand	Function
Add	$a + b + c\_in$
Subtract	$a + \sim b + c\_in$
Subtract_a	$b + \sim a + \sim c\_in$
Or_ab	{1'b0, a / b}
And_ab	{1'b0, a & b}
Not_ab	{1'b0, (~a) & b}
Exor	{1'b0, a ^ b}
Exnor	{1'b0, a ~^ b}

Make use of SystemVerilog constructs as appropriate. The signature for the module is:

```
import ALU_REGFILE_Defs::*;

module alu (
   input logic [ALU_INPUT_WIDTH-1:0]   A_In, B_In,  // A and B operands
   input logic                         Carry_In,    // Carry In
   input aluop_t                       Opcode,      // operation to perform
   output logic [ALU_OUTPUT_WIDTH-1:0] ALU_Out      // ALU result(extended by 1 bit
                                                    // to preserve Carry_Out from
                                                    // Sum/Diff)
);
```

Note that the ALU Output is one bit wider than the inputs. This is to preserve the Carry Out. Note also the use of a typedef enum for the ALU opcode (operation to perform)

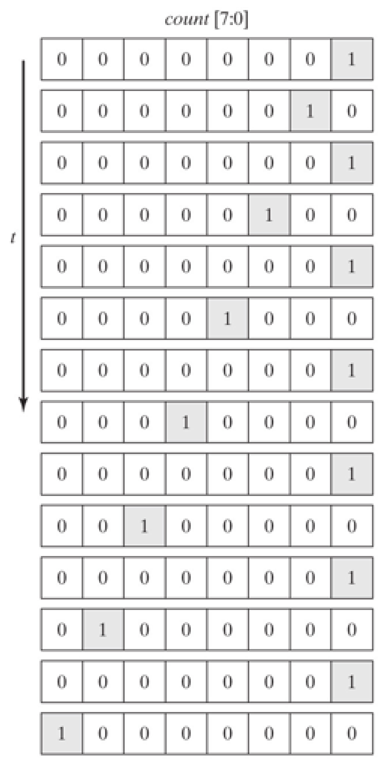
B. (15 pts) Create a model of the DUT called `alu_regfile` that instantiates instances of your ALU module and the Register File module provided in the release. The signature for the module is:

```
import ALU_REGFILE_Defs::*;

module alu_regfile (

   // register file interface
   input  logic [REGFILE_ADDR_WIDTH-1:0]      Read_Addr_1, // read port addresses
                                              Read_Addr_2,
   input  logic [REGFILE_ADDR_WIDTH-1:0]      Write_Addr,  // write port address
   input  logic                               Write_enable,// write enable (1 to
                                                           // write)
   input  logic [REGFILE_WIDTH-1:0]           Write_data,  // data to write into the
                                                           // register file

   // ALU interface.  Data to the ALU comes from the register file
   input  logic                               Carry_In,    // Carry In
   input  aluop_t                             Opcode,      // operation to perform
   output logic [ALU_OUTPUT_WIDTH-1:0]        ALU_Out,     // ALU result

   // system-wide signals
   input  logic                               Clock        // system clock
);
```

C. (20 pts) Simulate your source code with QuestaSim using the provided testbench to demonstrate that your design is correct. You will be graded on the correctness of your design.

Question 2 (40 pts)

Write and verify (i.e. write your own testbench) a SystemVerilog model for a “jerky” counter having the sequence shown below:



0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0

A. (15 pts) Write a SystemVerilog model of the counter. Use SystemVerilog constructs as appropriate. The signature for the module is:

```
module jerky_counter (
   input  logic        clock, reset,    // clock and reset
   output logic [7:0]  count            // counter output
);
```

B. (15 pts) Write a testbench for your counter. Testbenches for counters are pretty simple. Generate a clock, apply the clock to the counter, and wait a while until the sequence completes. Display the counter output and check your results manually.

C. (10 pts) Simulate your source code with QuestaSim with your testbench to demonstrate that your design is correct.