ECE 274a Lab Assignment 6: RTL Design – **Verilog (Behavioral way)**
Objectives:

- Practice writing Verilog code (Behavioral way) to implement RTL-based digital system
- Use to keep signals after synthesis in Vivado

Start: Week 13 (Nov 13, 2023) – attend your scheduled lab session
Demo Due:
Demo Part A by the end of the lab session during the week of Nov 13-17, 2023.
Demo Part B during the week of Nov 20-22, 2023 (open lab week: attend as many lab sessions as you need to demo this part. Lab sessions: Mon 9 – 11.50 am, Tues 2 -4.50 pm and Wed 9 – 11.50 am)

Submission: Submit the HLSM diagram, the .v files that you/your group create or modify, .xdc file on D2L Assignment Dropbox by 11.59 pm on Friday Nov 24, 2023.

Given the C code (pseudo-code),
Inputs: **byte A[16], go (1 bit),**
Outputs: **sum (8 bits), done (1 bit)**

```
while(1){
        while(!go);
        done = 0;
        sum = 0;
        i = 0;
        while(i < 16){
                temp = A[i];
                if((temp > 47) && (temp <58)) {
                        A[i] = temp - 48;
                        sum = sum + (temp-48);
                }
                i = i + 1;
        }
        done = 1;
}
```

Part A
A.1) Convert the C code to **HLSM diagram**

A.2) **Write Verilog code (Behavioral way)**

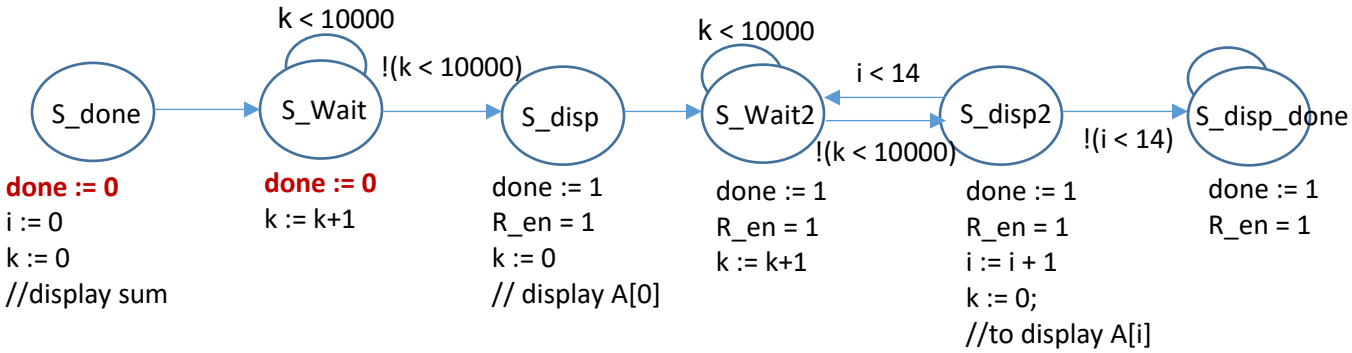
Note: Register file is a component used in this circuit to support the 16-element array A (byte A[16]). Even though, it shows as an input in the given C code. Verilog code for 16x8 Register file (RegFile16x8) is given on page 4. **Your system MUST use the given register file.**

A.3) **Write a testbench to perform behavioral simulation.** Use Clk of 200 ns period in your testbench. Bring content of Register File (RegFile[0] – RegFlie[15]) from RegFile16x8 to your simulation waveform. Use Add signal to Wave window to add the internal signals into the waveform (see the last page of lab 5 handout on how to). Observe the values in these registers. If working, the content of each register should be as shown in the waveform on page 5.

A.4) After the behavioral simulation waveform works correctly, you will
a) **Retain the signals** (modify the .v files). See page 6 for how to retain the signals.
b) Run synthesis and run **post-synthesis functional simulation**. See waveform on page 5.

Part B: Program to the FPGA board

B.1) **Modify** your ***HLSM diagram*** by **adding the following states** into it. These states are added so that during the time that done = 0, the circuit displays value of sum. When done = 1 (LED for done is lit up), the circuit displays the content of each register in the register file.



S_done state is the name used in my HLSM diagram of Part A where done is set to 1 based on the given C code. Your HLSM diagram of part A will likely use a different name for that state that you set done to 1. For this part B, S_done is now used to display the value of sum.
S_Wait is to be used as a timer for 1 second (used with 10kHz ClkOut). At S_Wait, value of sum is displayed on the FPGA board.
S_disp is to access the content of the register file in order to display RegFile[0] (since i is 0)
S_Wait2 – same idea as S_Wait
S_disp2 is to display RegFile[1] - RegFile[15], one at a time by incrementing i.

B.2) Use the following suggestions to modify your Verilog code from part A

- Add more Verilog statements into your code of Part A to implement your modified HLSM from B.1). you may need to ***change number of bits*** for your "state" and "nextstate". **k is a 14-bit local storage.**
- Add R_data (8-bit output from the register file) as the output of your .v file of lab 6 (from part A) since R_data will be used as the input to the two-digit display module as shown in Figure 1 (page 3).

B.3) Add ClkDiv module to the project and **Change DivVal** such that **ClkOut** has a **frequency of 10000 Hz.**

B.4) Create and Write ***Structural*** Verilog code (top-level code) to connect the modules together as shown in Figure 1 (page 3). Use the following as ***the code template for the top-level code***.

Note: Lab6_toplevel is the name that I use for my top-level code. You can use any name that you want BUT it CANNOT be the same name that you use for your .v file in part A.

```
module Lab6_toplevel (CLK100MHZ, BTNU, LED, CA, CB, CC, CD, CE, CF, CG, AN);
   input CLK100MHZ;
   input BTNU; //BTNU is Reset
   output [0:0] LED; //LED[0] is done
   output CA, CB, CC, CD, CE, CF, CG; //segment a, b, ... g
   output [7:0] AN; //enable each digit of the 8 digits
  //write your code to connect the modules as shown in Figure 1 (page 3)
endmodule
```

B.5) Add two-digit display code and sevensegment code to your project.

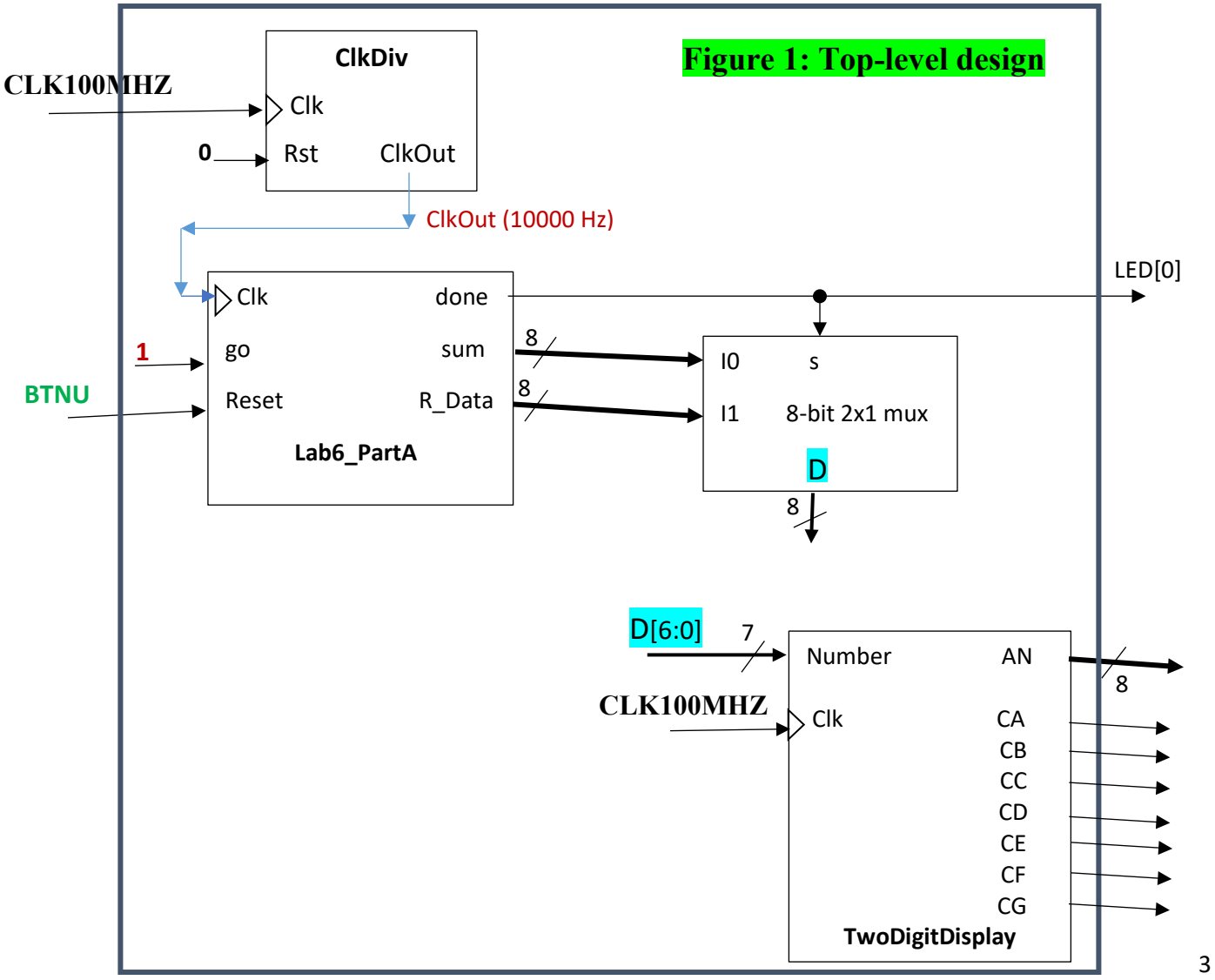
Note: You do not have to simulate the top-level code. If you want to, you can perform a simulation as follows: *Before* the simulation, first change DivVal value in ClkDiv to 1 (instead of value that you use for ClkOut of 10000 Hz. ***After*** the simulation, change DivVal back to the number that you use for 10000 Hz.

B.6) **Get Nexys4DDR_Master.xdc file** (from Lab4_code_used_inPart_B zipped folder on D2L Lab4 module) and Uncomment the pins that you will use for the inputs and outputs of this top-level code.

B.7) Generate Bitstream and program the FPGA board.

If working, you will see that the display starts with the final value of sum (which is 59 for the given values in the register file). Then the content of each register in the register file is shown from RegFile[0] to RegFile[15]. The display stops at 09 which is the value of RegFile[15]. Each value is displayed for 1 second since Clk used with lab6 is 10000 Hz and we check k < 10000 in S_Wait and S_Wait2 states. Note:

- The reset button can be pushed to reset your RTL-based circuit. When the reset is pushed, you will see value of sum which can be 59 or 00 depending on how you write the code for sum.
- You do not see the value of sum changes from 0, 8, 11, …, 59 since 10000-Hz clock is relatively fast and we cannot see it. If you want to see the value of sum changes, you have to 1) modify DivVal so that ClkOut is 100 Hz and 2) use k < 100 (instead of k < 10000) in states S_Wait and S_Wait2. You can then download/program FPGA again to see how your circuit works.

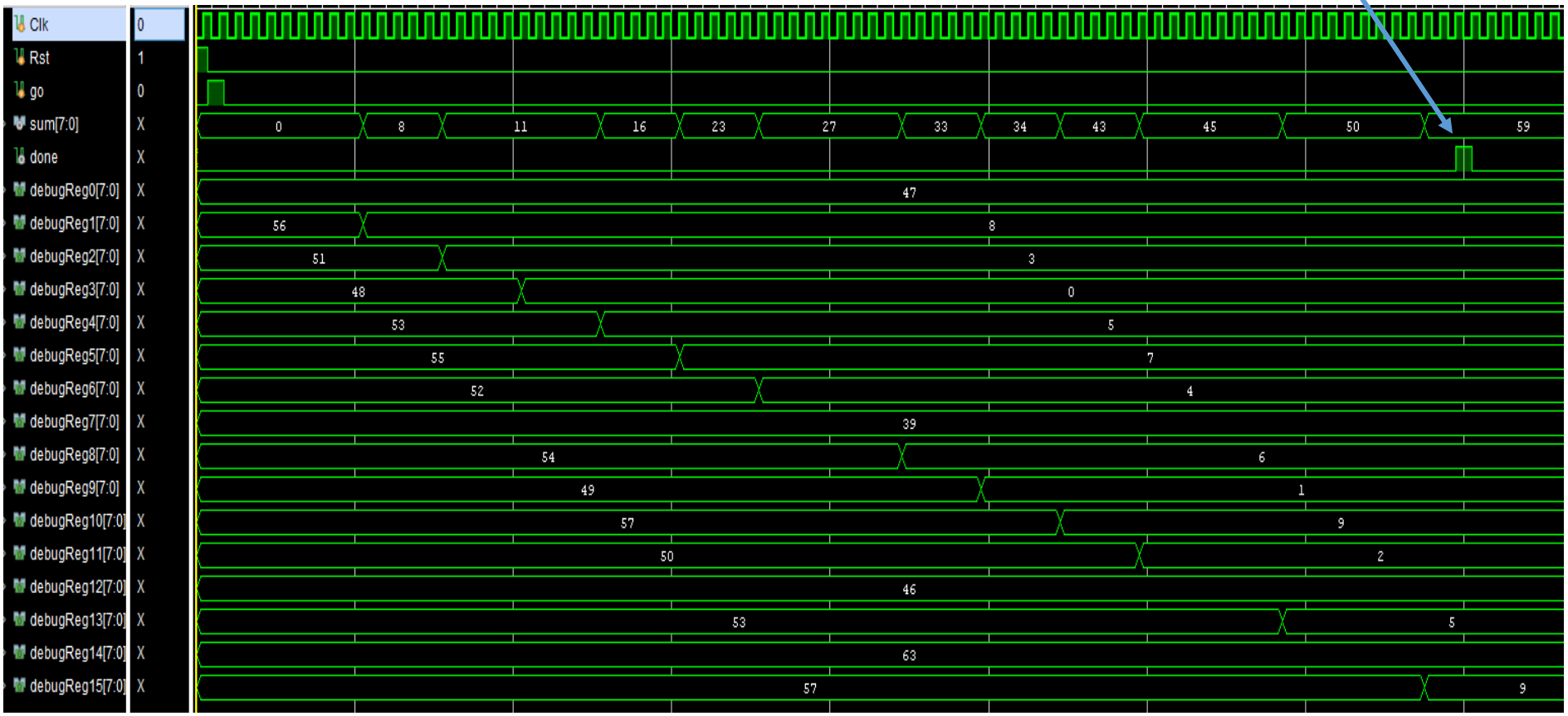


```
`timescale 1ns / 1ps

module RegFile16x8(R_Addr, W_Addr, R_en, W_en, R_Data, W_Data, Clk, Rst);
  input [3:0] R_Addr, W_Addr;
  input Clk, Rst, R_en, W_en;
  output reg [7:0] R_Data;
  input [7:0] W_Data;

  //simple memory declaration
  reg [7:0] RegFile [0:15];

  // Write procedure
  always @(posedge Clk) begin
    if (Rst == 1) begin
        RegFile[0] <= 8'd47;
        RegFile[1] <= 8'd56;
        RegFile[2] <= 8'd51;
        RegFile[3] <= 8'd48;
        RegFile[4] <= 8'd53;
        RegFile[5] <= 8'd55;
        RegFile[6] <= 8'd52;
        RegFile[7] <= 8'd39;
        RegFile[8] <= 8'd54;
        RegFile[9] <= 8'd49;
        RegFile[10] <= 8'd57;
        RegFile[11] <= 8'd50;
        RegFile[12] <= 8'd46;
        RegFile[13] <= 8'd53;
        RegFile[14] <= 8'd63;
        RegFile[15] <= 8'd57;
    end
    else if (W_en==1) begin
        RegFile[W_Addr] <= W_Data;
    end
  end

  // Read procedure
  always @(*) begin
    if (R_en==1)
      R_Data <= RegFile[R_Addr];
    else
      R_Data <= 8'bZZZZZZZZ;
  end

endmodule
```

The waveform below is from the ***post-synthesis functional simulation*** (therefore, the word “debug” in front of the register file content). In the behavioral simulation, you will see RegFile[0] – RegFile[15]

Below is what the register file and sum should be for the given register file (on page 4) when your circuit is working (done = 1).

When done = 1, sum is 59.
The register file contents are changed when its original value is between 47 and 58.



How to retain signals in post synthesis/implementation

After the synthesis, Vivado creates/synthesizes (also optimizes) the circuit from your Verilog code. Some signals may not stay as they are after the synthesis. This method gives you a way to preserve signal names after the synthesis.

To retain or preserve a few signals in post synthesis or implementation, use the followings:

(* mark_debug = "true" *) wire q0_r;

Vivado will retain signal "q0_r" after synthesis/implementation and you will be able to drag and drop (Add to Wave Window) the signal into the post synthesis/implementation simulation waveform.

NOTE: This should not be used with all internal signals as it will drastically increase compilation time and tool may run out of resources.

For this lab, we will **retain the content of register file (from all locations).** The *example* below is to show how to add the content from only *one location* (location 8). Using the similar idea, you can add the content from **ALL** locations of the register file.

In RegFile16x8.v,

1. Create an output port in addition to the existing input and output ports, for example, debug_reg8
module RegFile16x8(R_Addr, W_Addr, R_en, W_en, R_Data, W_Data, Clk, Rst, **debug_Reg8**);

output [7:0] debug_Reg8;

2. mark the entire RegFile with mark_debug
(* mark_debug = "true" *) reg [7:0] RegFile [0:15];

3. assign value of the specific register you want to observe in the post-routing simulation waveform

assign debug_Reg8 = RegFile [8];

In the .v file of your register file, write the above statement after the read procedure code and before endmodule.

In top.v (the one that you use to call/instantiate RegFile16x8),

If debug_reg8 is NOT an output of the top module, this signal again will be ignored. Therefore, in the top module define the followings:

(* mark_debug = "true" *) wire [7:0] debug_Reg8;

RegFile16x8 a1(R_Addr, W_Addr, R_en, W_en, R_Data, W_Data, Clk, Rst, **debug_Reg8**);

Grading rubrics:

Part A: a correct HLSM but INcomplete code	maximum of 10 out of 100 points
Part A: a correct HLSM, a complete code but the behavioral simulation is NOT working	maximum of 35 out of 100 points
Part A: a correct HLSM, a complete code, the behavioral simulation is working but the post-synthesis simulation is NOT working.	maximum of 50 out of 100 points
Part A: a correct HLSM, a complete code, the behavioral simulation is working but only sum shows up in the post-synthesis simulation. No retaining signals -> no content of register file can be displayed in the post-synthesis	maximum of 55 out of 100 points
Part A: a correct HLSM, a complete code, both behavioral and post-synthesis simulations are working. sum works correctly and Register file is written correctly (retaining signals is used to show register file content in post-synthesis	maximum of 65 out of 100 points
Part A work correctly. Part B: You have the top-level code and .xdc file to attempt to download to the board but NOT working.	maximum of 75 out of 100 points
Part A work correctly. Part B: You have the top-level code and .xdc files but only 'sum' is correctly displayed on the board.	maximum of 85 out of 100 points
Part A and B work correctly. Both A[i] for all i and sum are displayed.	100/100 points