Sheffield Hallam University
Semester I of academic year 2021/22

"SoC architectures and FPGA prototyping"

Lab 2 -
Digital design using asynchronous procedural Verilog

Assignment 1: converting behavioural Verilog code into asynchronous procedural one

This is strictly an academic assignment introduced in order to highlight differences in handling behavioural Verilog code inside a procedural block. This assignment builds upon your completed code lab1_3, and all you need to do is to adjust the Lab1_3 code.

Copy your completed Lab_1_3 project folder and rename it to Lab_2_1. Launch Quartus and open project from this new folder.

Write two header lines for the Verilog ***asynchronous*** procedural block

```
always @(*) begin

end
```

Asynchronous procedural blocks operate on any changes in the signals, which are present in the right-hand side (RHS) of the behavioural code inside the block. This is shown by the asterisk ***** in the sensitivity list.

`begin` … **`end`** keywords work like curly brackets {} in C, allowing placing several statements in between that are all included within the same block.

Move your behavioural code (some students condensed this code to three lines, other keep five lines specifying the output of every gate involved) inside the procedural block. Compile it getting a multitude of errors and try to make sense out of the compiler messages. They are all about the syntax Verilog error – you try using **`wire`** in the left-hand side (LHS) of a procedural Verilog block. Additionally, **`assign`** keywords are not allowed either and must be deleted.

Delete **`assign`** keywords from the behavioural statements, and declare all the LHS signals as **`reg`**. If you drive some output ports from the assign statement directly, you will need to introduce additional register(s) for the appropriate LHS, and additionally assign the appropriate output port to this additional register. For example

```
wire G3out, tmp;
assign HEX[0] = G3out ^ tmp;  // equation driving an output port
assign G5out = ~tmp;  // equation driving an internal wire
```

Getting it inside an asynchronous procedural block gives

```
wire G3out, tmp;
always @(*) begin
     assign HEX[0] = G3out ^ tmp;
     assign G3out = ~tmp;
end
```

Correcting the code according to the procedural Verilog conventions

```
wire tmp;
reg G3out, tmp1;  // need an extra register for LHS
always @(*) begin
     tmp1 = G3out ^ tmp;
     G3out = ~tmp;
end
assign HEX[0] = tmp1;
// HEX[0] cannot be driven from inside the block
```

It is a common convention and good practice to use only blocking assignments (=) in synchronous procedural Verilog. Blocking means that these statements are to be implemented/executed/simulated in the stated order, from top to bottom. All our previous codes were not sensitive to the order of Verilog statements, but here it is the case. Re-arrange (if necessary) your statements in the procedural block in order to make sure all the signals used in the RHSes of any statements are already defined in the LHS of some preceding statement.

Compile the corrected design and verify it running on board. Take screenshots for the report as follows

- Verilog code
- resource utilisation and confirm that no extra resources are required comparing to the Lab_1_3.

Assignment 2: driving a seven segment LED using `case`

You will write a code to display a HEX value entered by four slide switches on a particular seven segment LEDs.

The most convenient way to complete the design is to use the `case` statement that allows multiplexing the output values, required to drive the segments, depending on the switches' position. Please be reminded that `case` keyword can only be used inside a procedural block therefore only `reg` are allowed in the LHS of corresponding behavioural statements.

Your particular design will depend on your seven-digit student ID number, here ABCDEFG, as follows:

- seven segment LEDs to drive (the top row only is required for the assignment 2, the bottom row will be needed for the assignment 3):

G	0	1	2	3	4	5	6	7	8	9
	HEX5 HEX4	HEX4 HEX3	HEX3 HEX2	HEX2 HEX1	HEX1 HEX0	HEX5 HEX3	HEX4 HEX2	HEX3 HEX1	HEX2 HEX0	HEX5 HEX0

- switches to enter the value to be displayed

F	0	1	2	3	4	5	6	7	8	9
	SW[3] .. SW[0]	SW[4] .. SW[1]	SW[5] .. SW[2]	SW[6] .. SW[3]	SW[7] .. SW[4]	SW[8] .. SW[5]	SW[9] .. SW[6]	SW[3] .. SW[0]	SW[5] .. SW[2]	SW[7] .. SW[4]

For example, the students with ID number 1234567 will use

- HEX3 to display the value when no KEY is pressed (and additionally HEX1 if any or both is (are) pressed for the assignment 3);
- SW[9] down to SW[6] to enter the value to display.

Open the Lab_2_2 folder and comment out the line that blanks the HEX you need to use.

Your code will have the following structure

```
reg [7:0] r_7LED; // for an LHS of the procedural block
always @(*) begin // procedural block to use case
// strictly speaking begin..end is not required for a single
// statement but it is a good practice to put it anyway
      case ( <here_you_need_to_insert_your_value> )
            // value - either bit slice or bit concatenation
            4'b0000: r_7LED  = 8'b_1_100_0000;
            // remember 0 for a segment ON

            // 15 more lines for all the cases, e.g.
            4'b1010: r_7LED = 8'b_0_000_1000; // see APPENDIX

            default: r_7LED = 8'b_1_111_1111;
            // strictly not required here as all the cases are
            // to be covered but again a good practice

      endcase
end
assign HEX0 = r_7LED; // to drive the output port
```

EXAMPLE of bit slicing and concatenation for two bits only (here SW[3] and SW[2])
out of the ten slide switches SW[9]..SW[0]

bit slicing – taking some part of a wide signal – **`SW[3:2]`**

bit concatenation – joining individual bits together in a list – **`{ SW[3], SW[2] }`**

Using the decimal dot (most significant bit – MSB - of the HEX):

please switch it ON if the value to be displayed is more then 9 (put 0 for this bit instead of 1). This will indicate that the displayed value is a hexadecimal, rather than decimal, digit.

Compile your design and verify it running on board.
Take screenshots for the report as follows

- Verilog code
- resource utilisation.

Set the required slide switches to the value of the last digit of your student ID number. Take a photograph of the board for the report.

Assignment 3: alternating the display when a push button is pressed; developing your first Verilog module

Assignment 3 builds upon the assignment 2 but you will use two seven segment displays. If any key is pressed (or both), the other display in the pair should be used (think of a local and panel display; the value is to be displayed on a control panel all the time but sometimes it needs to be displayed locally; using a local push button saves energy).

As you now have to drive the two displays, it makes a lot of sense to write a single module then instantiate it twice rather than copy and paste the relevant procedural block (especially because in the lab 3 you will need to drive all the six displays simultaneously).

Copy your folder Lab_2_2 and rename the copied folder to Lab_2_3.

Open a new file, make sure you select Verilog for it, and the checkbox "Add file to the project" is ticked. This new file is to be used for the **`SevSeg.v`** module you will develop. Save it with this new name. Start your code by providing an appropriate module declaration; there will be one four bit input port for the value to be displayed, and one eight bit output port to drive the display. Add the **`endmodule`** line and compile your code to make sure there are no errors so far.

Cut your code for driving the display from the top module and paste it here. Make sure your **`case`** is controlled by the input port, and the output port is assigned to the value of the temporary register set inside the **`case`** statement. Compile the code again and fix errors if any. Now the display is not driven anymore because the new module is not yet used in the design though.

Instantiate the new module in the top module by adding a line in Verilog-2001 like

```
SevSeg SevSeg_1 (
   .<input_port_in SevSeg> ( <appropriate_signal_in_the_top_module> ),
   .< output_port_in SevSeg> ( <appropriate_signal_in_the_top_module> )
);
```

Compile your design and run it on board; all should work like in the lab2_2.

Instantiate another copy of the **`SevSeg`**, namely **`SevSeg_2`** . Connect **`SevSeg_2`** input to the same input as the **`SevSeg_1`**, and its output to some temporary eight-bit wire, say, **`SevSeg_2_out`**. Comment out the line that blanks the second display (say, HEX0 used below).

Write behavioural code to drive the second display like

```
assign HEX0=( <all_or_some_KEY_pressed> ) ? SevSeg_2_out : 8'b_1111_1111;
```

Depending on the value of KEY, the code will either drive HEX0 with the entered value or blank it out to save on energy. For the logical condition required you can either use logical or of the individual KEY bits or, if you are comfortable with it, the reduction operator.

In fact, most Verilog designers prefer using **IF** statements to describe even 2 input multiplexers. That is because **IF** allows getting priority multiplexers with complex logic. Despite the objective of the assignment is now completed, we will additionally explore this possibility.

Comment out the line that drives the second display. You will now need to add code with the following structure

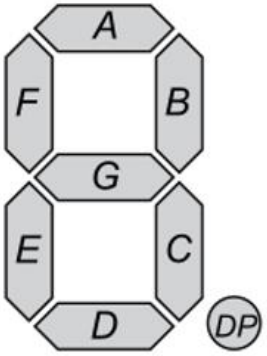
```
reg [7:0] r_7LED; // for LHS of the procedural block
always @(*) begin // block to use if
// strictly speaking begin..end is not required for a single // statement
but it is a good practice to put it anyway
      if ( <all_or_some_KEY_pressed> )
            r_7LED  = SevSeg_2_out;
      else
            r_7LED  = 8'b_1111_1111;
            // you could use begin..end or nested IF
            // for your TRUE and/or FALSE branches
            // ! make sure every IF has ELSE branch
            // ! not to infer any latches
end
assign HEX0 = r_7LED; // to drive the output port
```

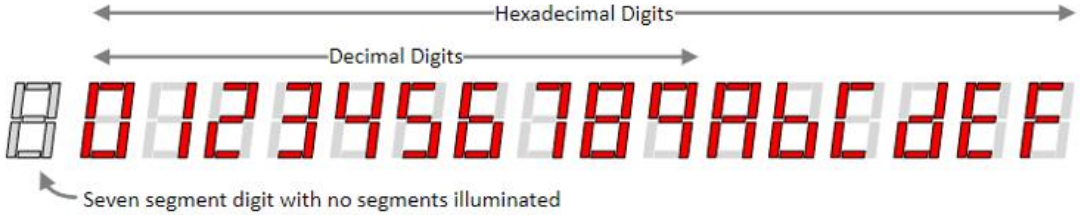
Compile the corrected design and verify it running on board. Take screenshots for the report as follows

- Verilog code
- resource utilisation.

Set the required slide switches to the value of the last digit of your student ID number and press any push button. Take a photograph of the board for the report.

Report for the lab 2: answer questions at the end of the report template for the lab 2.





DE10-Lite: 1 for segment OFF; 8'b_DP_G_F_E_D_C_B_A

Inputs				Segments							
A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	For display 0
0	0	0	1	0	1	1	0	0	0	0	For display 1
0	0	1	0	1	1	0	1	1	0	1	For display 2
0	0	1	1	1	1	1	1	0	0	1	For display 3
0	1	0	0	0	1	1	0	0	1	1	For display 4
0	1	0	1	1	0	1	1	0	1	1	For display 5
0	1	1	0	1	0	1	1	1	1	1	For display 6
0	1	1	1	1	1	1	0	0	0	0	For display 7
1	0	0	0	1	1	1	1	1	1	1	For display 8
1	0	0	1	1	1	1	1	0	1	1	For display 9
1	0	1	0	1	1	1	0	1	1	1	For display A
1	0	1	1	0	0	1	1	1	1	1	For display b
1	1	0	0	1	0	0	1	1	1	0	For display C
1	1	0	1	0	1	1	1	1	0	1	For display d
1	1	1	0	1	0	0	1	1	1	1	For display E
1	1	1	1	1	0	0	0	1	1	1	For display F

8'b_1_100_0000;

8'b_0_000_1000;