THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Engineering

Dr Peter Stepien (peter.stepien@newcastle.edu.au)

Assignment 1

Instructions

- Complete all questions.
- Work in groups of two.
- Assignment files due end of Week 6 on Friday 5 September 2025 by 11:59pm.
- Demonstration during laboratory session in Week 7.
- Total of 90 Marks possible.

Assessment

This assignment will be assessed using the criteria in Table 1 for each of the questions:

Component	*Unsatisfactory*	*Improving*	*Satisfactory*	*Excellent*
Question 1 (5 Marks)	Decoder not working	Simulating, but not on FPGA	All working with good comments	
Question 2 (5 Marks)	Multiplier not working	Simulating, but not on FPGA	All working with good comments	
Question 3 (5 Marks)	Multiplier not working	Simulating, but not on FPGA	All working with good comments	
Question 4 (10 Marks)	Logic unit not working	Simulating, but not on FPGA	All working with good comments	
Question 5 (10 Marks)	Adder/Subtractor not working	Simulating, but not on FPGA	All working with good comments	
Question 6 (10 Marks)	Classifier not working	Working, but large propagation delay	Optimised propagation delay	
Question 7 (15 Marks)	Shifter not working	Working, but not on FPGA	All working with good comments	
Question 8 (15 Marks)	Multiply/Divide not working	Multiply or divide working	All working with good comments	
Question 9 (15 Marks)	ALU not working	ALU partially working	All working with good comments	

Table 1: Marking criteria

Objectives

The main objective is to develop a library of building blocks that will be used in the next assignment to build a MIPS processor based system that will be able to run programs. This will give you experience in writing System Verilog designs, test benches, ModelSim and targeting the FPGA development board.

The test benches in ModelSim should include code that verifies the correct operation of your design. The test bench should have complete coverage of all input combinations where practical.

The connection of switches and LEDs on the FPGA development board is indicated for each question to give consistent output. In most cases, the size parameter, n, has been chosen to allow all input combinations to be tested on the hardware.

Submission

For each question, create a single file by joining together all of the System Verilog source code. Include both the design files and the test benches. Add comments to your files so that it is clear which question they refer to and who are the group members. Each group member needs to submit the code on Canvas.

Demonstration

Solutions to be demonstrated on the FPGA hardware only the week after the code is submitted. Since some questions build on previous questions, only the following questions need to be demonstrated:

- Question 3: 8-bit Multiplier
- Question 6: Instruction Classifier
- Question 8: Multiplier/Divider
- Question 9: Arithmetic Logic Unit

If the design does not work on the FPGA hardware for a particular question, the solution for previous questions can be demonstrated and/or the ModelSim simulation can be demonstrated. Otherwise, there is no need to demonstrate the ModelSim simulation.

1 7-Segment Display Driver (5 Marks)

A System Verilog description for a 7-segment display decoder is required. It has a four-bit input D, and a seven bit output $a..g$, which is used to display the character associated with the hexadecimal code represented. It is to also have a ripple blanking in/out, $\overline{RBI}/\overline{RBO}$. The functionality of the decoder is described in Figure 1, however, the logic for the segment outputs needs to be inverted for the hardware being used. Also, no latching function needs to be implemented, so $\overline{LE}$ can be considered as always low.

TRUTH TABLE

BINARY STATE	INPUTS						OUTPUTS								DISPLAY
	$\overline{LE}$	$\overline{RBI}$	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g	$\overline{RBO}$	
—	H	*	X	X	X	X	◄			STABLE			►	H	STABLE
0	L	L	L	L	L	L	L	L	L	L	L	L	L	L	BLANK
0	L	H	L	L	L	L	H	H	H	H	H	H	L	H	0
1	L	X	L	L	L	H	L	H	H	L	L	L	L	H	1
2	L	X	L	L	H	L	H	H	L	H	H	L	H	H	2
3	L	X	L	L	H	H	H	H	H	H	L	L	H	H	3
4	L	X	L	H	L	L	L	H	H	L	L	H	H	H	4
5	L	X	L	H	L	H	H	L	H	H	L	H	H	H	5
6	L	X	L	H	H	L	H	L	H	H	H	H	H	H	6
7	L	X	L	H	H	H	H	H	H	L	L	L	L	H	7
8	L	X	H	L	L	L	H	H	H	H	H	H	H	H	8
9	L	X	H	L	L	H	H	H	H	L	L	H	H	H	9
10	L	X	H	L	H	L	H	H	H	L	H	H	H	H	A
11	L	X	H	L	H	H	L	L	H	H	H	H	H	H	b
12	L	X	H	H	L	L	H	L	L	H	H	H	L	H	C
13	L	X	H	H	L	H	L	H	H	H	H	L	H	H	d
14	L	X	H	H	H	L	H	L	L	H	H	H	H	H	E
15	L	X	H	H	H	H	H	L	L	L	H	H	H	H	F
X	X	X	X	X	X	X	L	L	L	L	L	L	L	L**	BLANK

*The $\overline{RBI}$ will blank the display only if a binary zero is stored in the latches.
**The $\overline{RBO}$ used as an input overrides all other input conditions.
H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

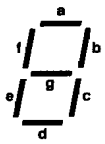

Figure 1: 9368 7-segment decoder

a) Write the System Verilog code for the decoder and test bench. Verify that it is operating correctly using ModelSim. Include comments that describe the function of each input and output and the operation of the decoder. (3 Marks)

b) Instantiate two 7-segment decoders on the FPGA development board, using two lots of four switches for the each of the digits as shown in Figure 2. Connect the ripple blanking signals so that the leading zeros are blanked. (*Hint: Use a structural description to instantiate two decoders*). (2 Marks)

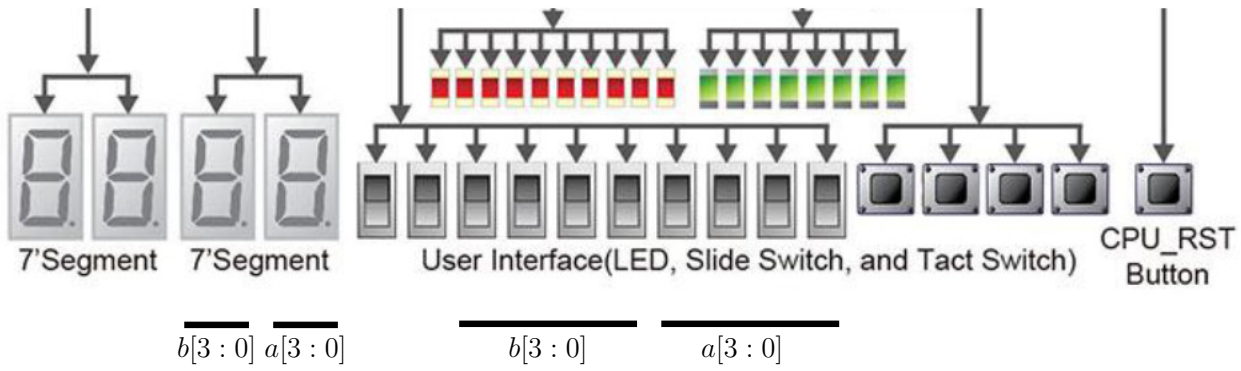

Figure 2: Mapping of indicators and switches

2 4-bit Multiplier (5 Marks)

A System Verilog description for a 4-bit multiplier is required. The product of two 4-bit numbers, a and b will give up to an 8-bit result c. Implement the multiplier using long multiplication as outlined in Table 2.

MSB							*LSB*	
0	0	0	0	$a[3]$	$a[2]$	$a[1]$	$a[0]$	$\times$
0	0	0	0	$b[3]$	$b[2]$	$b[1]$	$b[0]$	
0	0	0	0	$(a[3]\cdot b[0])$	$(a[2]\cdot b[0])$	$(a[1]\cdot b[0])$	$(a[0]\cdot b[0])$	
0	0	0	$(a[3]\cdot b[1])$	$(a[2]\cdot b[1])$	$(a[1]\cdot b[1])$	$(a[0]\cdot b[1])$	0	$+$
0	0	$(a[3]\cdot b[2])$	$(a[2]\cdot b[2])$	$(a[1]\cdot b[2])$	$(a[0]\cdot b[2])$	0	0	$+$
0	$(a[3]\cdot b[3])$	$(a[2]\cdot b[3])$	$(a[1]\cdot b[3])$	$(a[0]\cdot b[3])$	0	0	0	$+$
$c[7]$	$c[6]$	$c[5]$	$c[4]$	$c[3]$	$c[2]$	$c[1]$	$c[0]$	

Table 2: Long multiplication of two 4-bit numbers

a) Write the System Verilog code for the 4-bit multiplier and test bench. Verify that it is operating correctly using ModelSim. Include comments that describe the function of each input and output and the operation of the decoder. (3 Marks)

b) Instantiate the 4-bit multiplier on the FPGA development board using two lots of four switches for the each input as shown in Figure 3. (*Hint: Use a structural description to instantiate the multiplier and four 7-segment decoders, one each for the input and two for the output*). (2 Marks)

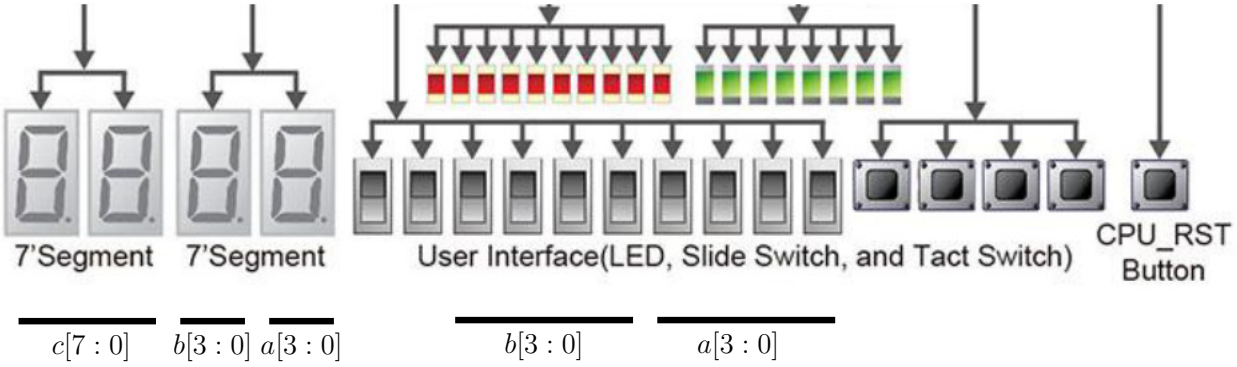

Figure 3: Mapping of indicators and switches

3 8-bit Multiplier (5 Marks)

A System Verilog description for an 8-bit multiplier is required. The product of two 8-bit numbers, a and b will give up to an 16-bit result c. Implement the multiplier using long multiplication similar to that done in the previous question.

a) Write the System Verilog code for the 8-bit multiplier and test bench. Verify that it is operating correctly using ModelSim. Include comments that describe the function of each input and output and the operation of the decoder. (3 Marks)

b) Modify the System Verilog code for the 8-bit multiplier and test bench to use a high level *behavioural* description of multiplication. Attempt to instantiate both versions of the 8-bit multiplier one at a time and comment on the FPGA resources required for each using the switches and LEDs as shown in Figure 4. Since there are insufficient inputs, assume $a[5:2]$ are zero. Write your comments in the System Verilog source code. (2 Marks)

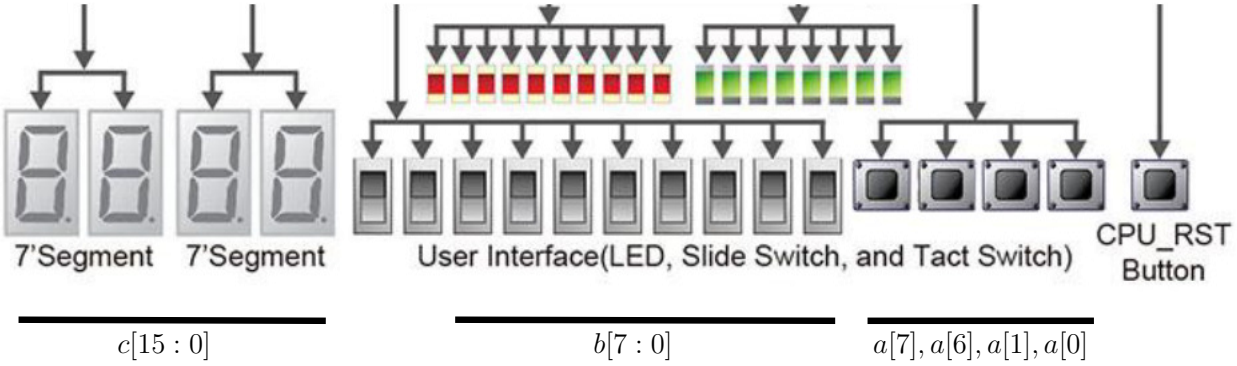

Figure 4: Mapping of indicators and switches

4 Logic Unit (10 Marks)

The design for the logic unit of a MIPS processor is required. This unit has one control input F and two data inputs A and B. F is 2 bit wide. Each of A and B are 2^n bit wide, where n is an integer. Typically $n > 2$, but that is not important for this assignment. The output Y is also 2^n bit wide. The relation between the inputs and outputs is given in Table 3.

$F[1:0]$	Y	*Operation*
00	$A \cdot B$	AND
01	$A + B$	OR
10	$A \oplus B$	XOR
11	$\overline{A + B}$	NOR

Table 3: Truth table for the MIPS compatible logic unit.

a) Design a parameterised System Verilog module to implement the logic unit, with the with of the data being n bits. Verify that it is operating correctly using ModelSim. (6 Marks)

b) Design the top level module and test the circuit on the development board using the switches and LEDs as shown in Figure 5 for $n = 2$. (4 Marks)

(*Hint: Use bit-wise operators to implement the logic operations. Then use a 4:1 MUX to select the output of the logic operation chosen by F to be routed to the output.*)

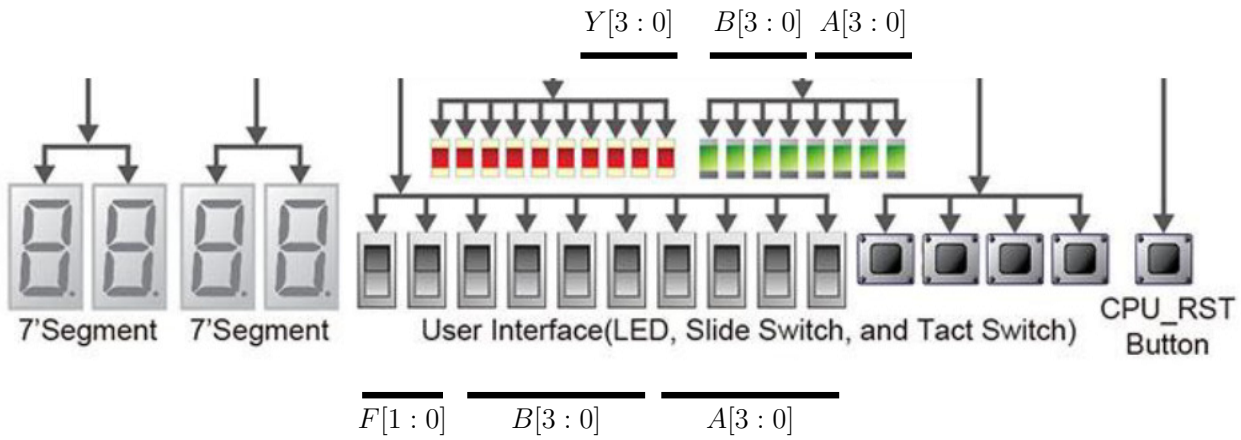

Figure 5: Mapping of indicators and switches

5 Add/Subtract Unit (10 Marks)

The design for the add/subtract unit of a MIPS processor is required. The circuit will either add or subtract the inputs A and B depending on a control input s to give the output Y. Each of A, B, and Y are 2^n bit wide. The functional specification of this circuit is given in Table 4.

s	Y	*Operation*
0	$A + B$	Add
1	$A - B$	Subtract

Table 4: Truth table for the add/subtract unit.

a) Use the $+$ operator in System Verilog to implement a parameterised 2^n bit adder to allow the size of the data to be set during instantiation. Confirm correct operation using a test bench. (4 Marks)

b) Design a System Verilog module to implement the add/subtract unit using adder designed in the previous question and a 2:1 multiplexer. Your design should not use more than one adders. Confirm correct operation using a test bench. (3 Marks)

c) Design a top-level module to test the circuit on the development board using the switches and LEDs as shown in Figure 6 for $n = 2$. (3 Marks)

Hint: Recall that in 2's complement arithmetic:

```
-B = ~B + 1
```

Hence we can subtract B from A with a standard adder as per the following formula:

```
A-B = A + ~B + 1
```

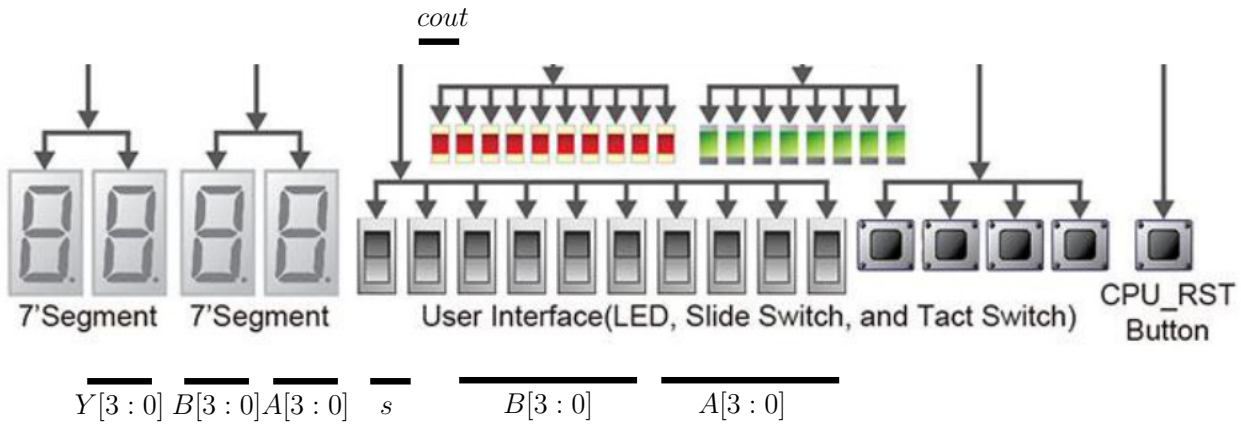

Figure 6: Mapping of indicators and switches

6 Instruction Classifier (10 Marks)

The design of the top level instruction classifier, which is central to the instruction decoder in a MIPS processor, is required. A MIPS instruction has a 6-bit *Opcode* input which determines the operation of the processor. This module has seven single-bit outputs R, B_1, J, B_2, I, F, and M. This is shown in Table 5, which is an extract from the MIPS instruction set and gives a summary of MIPS opcodes.

Opcode	R	B_1	J	B_2	I	F	M	*Type/Function*
000000	1	0	0	0	0	0	0	R-type instruction
000001	0	1	0	0	0	0	0	Branch less than zero
$00001x$	0	0	1	0	0	0	0	Jump instructions
$0001xx$	0	0	0	1	0	0	0	Branch instructions
$001xxx$	0	0	0	0	1	0	0	Immediate operands
$01xxxx$	0	0	0	0	0	1	0	F-type instructions
$1xxxxx$	0	0	0	0	0	0	1	Memory access

Table 5: MIPS instruction classification based on opcodes.

Note that the instruction classifier circuit can be used to simplify the instruction decoder, where the outputs of the instruction classifier are used to *activate* various sub-modules in the processor as necessary.

a) Design the instruction classifier, optimising the propagation delay through the circuit. Confirm correct operation using a test bench. (6 Marks)

b) Design a top-level module to test the circuit on the development board using the switches and LEDs as shown in Figure 7. (4 Marks)

The mark for this question will also depend on the propagation delay through the circuit. Refer to the topic slides for ways to reduce the propagation delay.

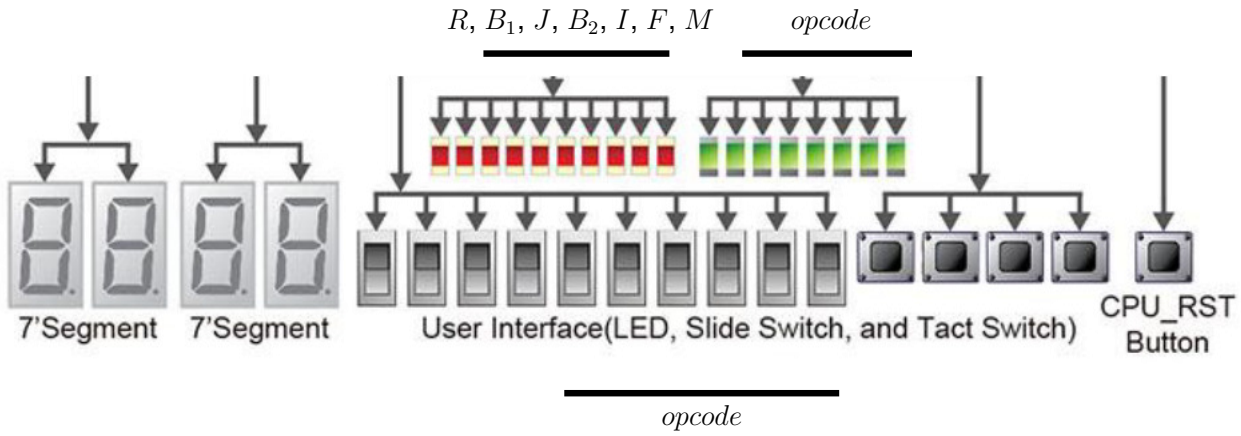

Figure 7: Mapping of indicators and switches

7 Shifter (15 Marks)

A shifter unit is required for the MIPS processor. In particular, an implementation for shift operations among the MIPS R-type instructions.

a) As a starting point, design the following shifter and confirm correct operation using a test bench. The shifter has three inputs:

1. A 2-bit control field, $F[1:0]$.
2. The value to be shifted, A.
3. The number of bits to be shifter, Sh.

The input F is the control input and determines the operation to be performed. The output is Y. The operation of the shifter is summarised in Table 6.

$F[1:0]$	Y
00	$A \ll Sh$ (left shift A by Sh)
01	$A \gg Sh$ (right shift A by Sh)
11	$A \ggg Sh$ (right arithmetic shift A by Sh)

Table 6: Truth table for MIPS shift hardware.

Note that if Sh is n bits wide then A and Y are 2^n bits wide. Make n a parameter with a default value of 4. Any shift operation implementation can be used, including behavioural with the System Verilog <<, >>, and >>> operators. With >>>, the `$signed directive` must be used, such as:

```
assign a = $signed(b) >>> 3;
```

A structural implementation can also be used. (6 Marks)

b) Using the module designed in the previous part, design hardware that readily supports MIPS R-type shift instructions. The there data inputs to this hardware are denoted as b, a, and c. For a 2^n bit processor b, a, and the output y are 2^n bit wide and c is n bit wide. The relation between the inputs and the output is controlled by the control input $F[2:0]$ as summarised in Table 7, for an assumed value of $n = 4$.

$F[2{:}0]$	y
000	$a \ll c$
001	$a \gg c$
011	$a \ggg c$
100	$a \ll b_{3:0}$
101	$a \gg b_{3:0}$
111	$a \ggg b_{3:0}$

Table 7: Truth table for MIPS compatible shifter.

Hint: Identify the role of F[2] and find out how a 2:1 MUX could be used along with the module designed in the previous part to implement a circuit that satisfied the requirements in Table 7. (5 Marks)

c) Design a top-level module to test the circuit on the development board using the switches and LEDs as shown in Figure 8 for $n = 2$. (4 Marks)

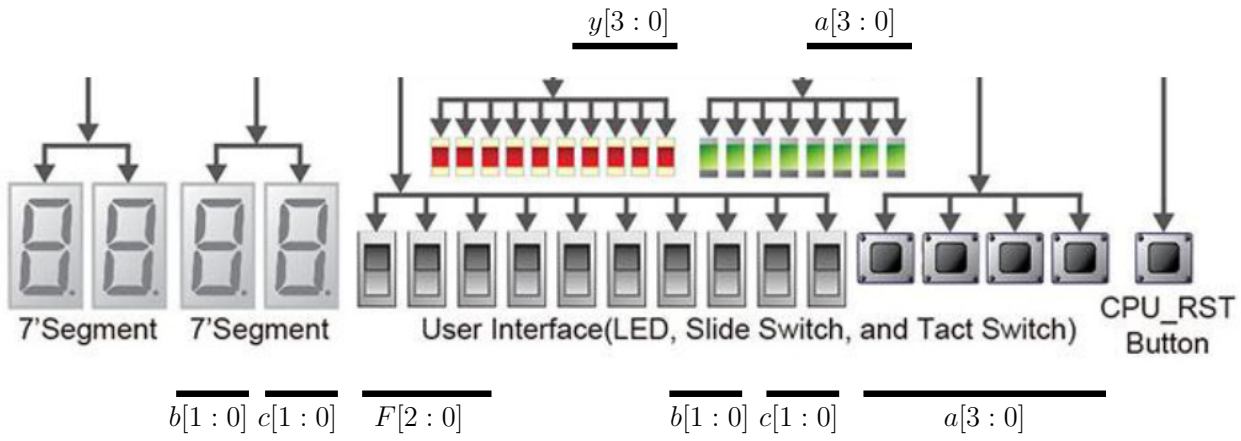

Figure 8: Mapping of indicators and switches

8 Multiplier/Divider (15 Marks)

A design for a multiply-divide hardware of the MIPS processor is required. In particular, to implement the MIPS R-type instructions.

An n bit MIPS compatible multiply-divide hardware can be seen as a simple state machine, which has two internal states $[hi]$ and $[lo]$, each n bit wide. When write-enabled, these state elements are updated at the leading edge of an input clock signal clk. In addition it has got two n bit data inputs a and b and an n bit output y. The next state and the output logic is controlled via $F[3:0]$ as described in Table 8, where x stands for a don't care condition, $[lo]_{next}$ denotes the content of the lo register after the next positive clock edge, and $[hi]_{next}$ denotes the content of the hi register after the next positive clock edge.

F[3:0]	y, $[hi]_{\text{next}}$, $[lo]_{\text{next}}$
0000	$y = [hi],\ [lo]_{\text{next}} = [lo],\ [hi]_{\text{next}} = [hi]$
0001	$y = \text{x},\ [lo]_{\text{next}} = [lo],\ [hi]_{\text{next}} = a$
0010	$y = [lo],\ [lo]_{\text{next}} = [lo],\ [hi]_{\text{next}} = [hi]$
0011	$y = \text{x},\ [lo]_{\text{next}} = a,\ [hi]_{\text{next}} = [hi]$
1000	$y = \text{x},\ \{[hi]_{\text{next}}, [lo]_{\text{next}}\} = a * b$
1010	$y = \text{x},\ [hi]_{\text{next}} = a\%b,\ [lo]_{\text{next}} = a/b$

Table 8: Subset of MIPS R-type instruction set involving multiply-divide operations. Here x stands for don't care conditions, $[lo]_{next}$ denotes the content of the lo register after the next positive clock edge, and $[hi]_{next}$ denotes the content of the hi register after the next positive clock edge.

a) Design a multiply-divide hardware as per Table 8 and confirm correct operation using a test bench. A simple way to implement this is given in Figure 9. You need to find how to derive the controls signals $s0$, $s1$, $s2$ from F. The multiplier and divider can be easily implemented in System Verilog using the high level operators $*$, $/$, and $\%$. (10 Marks)

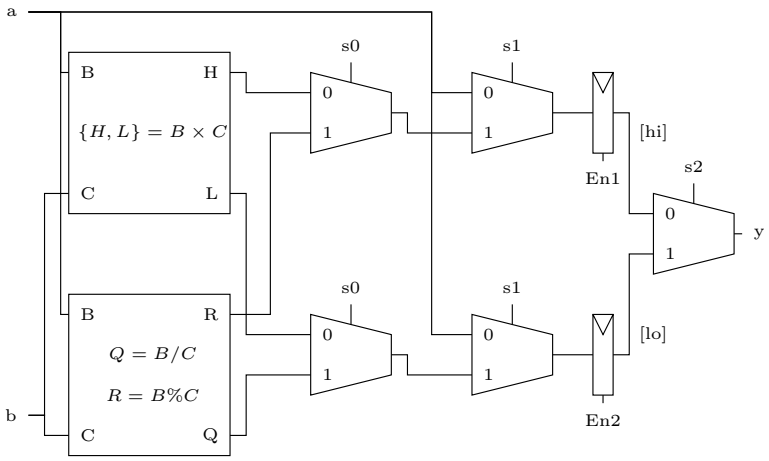

Figure 9: Multiply-divide circuit.

b) Design a top-level module to test the circuit on the development board using the switches and LEDs as shown in Figure 10 for $n = 3$. (5 Marks)

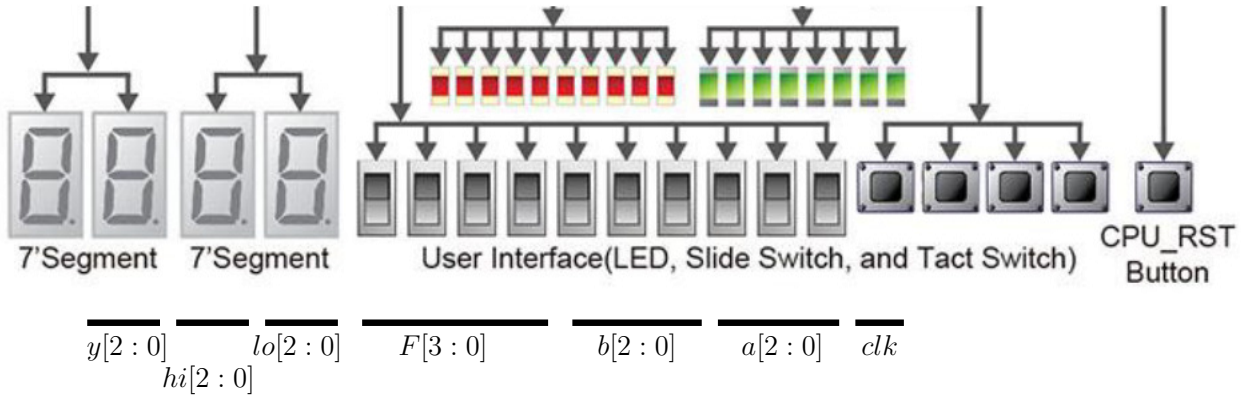

Figure 10: Mapping of indicators and switches

9 Arithmetic Logic Unit (15 Marks)

The design of the arithmetic logic unit (ALU) for the MIPS processor is required. In particular, the arithmetic logic instructions for the MIPS R-type instructions.

A MIPS compatible ALU takes $F[3:0]$ as a control input. This control input determines the relation between the data inputs A and B, and the output Y as per Table 9, which is a part of the set of MIPS R-type instructions. A step by step design process is described in the lecture slides. Note that the ALU hardware includes an adder. Implementation can use the System Verilog + operator.

F[3:0]	Y
0000	$A + B$ (signed addition)
0001	$A + B$ (unsigned addition)
0010	$A - B$ (signed subtraction)
0011	$A - B$ (unsigned subtraction)
0100	A & B (AND)
0101	$A\|B$ (OR)
0110	$A \oplus B$ (XOR)
0111	$\overline{(A\|B)}$ (NOR)
1010	if$(A < B)$ $Y = 1$; else $Y = 0$
1011	if$(A < B)$ $Y = 1$; else $Y = 0$ (unsigned comparison)

Table 9: Excerpt of MIPS R-type instruction set involving arithmetic-logic operations.

a) Design an n bit wide parameterized ALU module as per Table 9. The ALU should have two additional outputs C_{out} and OV to flag the carry out and arithmetic overflow in arithmetic operations. (10 Marks)

b) Design a top-level module to test the ALU circuit on the development board using the switches and LEDs as shown in Figure 11 for $n = 3$. (5 Marks)

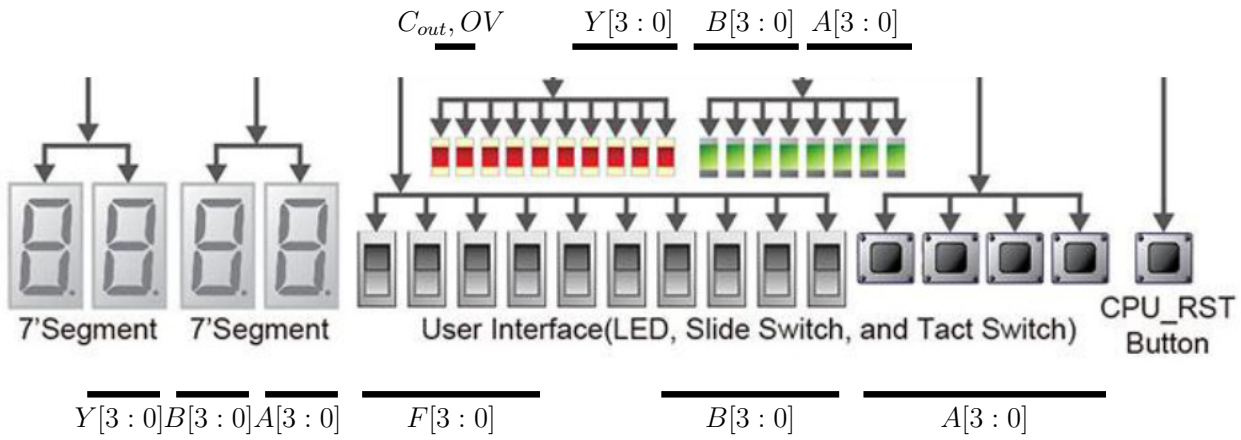

Figure 11: Mapping of indicators and switches