

ECS2039: Digital Systems – Practical Part (2022/23): Final Project

Submission deadline: 23/04/2023 at 23.59pm

Assignment Rules:

1. This assessment is worth **30%** of your overall grade. The project includes 4 tasks and a total of 150 marks is available. Note that the assignment deadline is advertised as 23 April 2023 (23:59 pm) as originally scheduled at the beginning of the semester. **You are highly encouraged to start the assessment early and complete the assessment gradually.**
2. Submission and late submission: The solutions **must be submitted in pdf** and a **ZIP file containing all source codes and test benches** and VIVADO reports (utilization, timing and power) only via **Canvas**. All the source codes should be tested and compiled via the VIVADO tool **without error**. The codes with errors will not be accepted. You should take a screenshot (using snipping or other similar tools) of your **codes, simulation** waveforms and **schematics** inside the VIVADO as answers to the tasks as well as attach the source files in a ZIP file.
3. You may wish to complete your coursework electronically using Word, LaTeX, or other technology. If you complete a part of your coursework on paper you must create a digital copy of your work using a scanner. **Photographs of paper based coursework will not be accepted.**
4. Late submission: Late submissions will be deducted 5% marks per day of delay. No submission will be accepted more than 5 calendar days later than the deadline.
5. No plagiarism allowed: **This is an individual assignment.** No plagiarism is permitted: you should not copy your solutions from each other or any resource. If you need to refer to online sources/books, you must refer to them appropriately. If plagiarism is detected you may lose marks and/or face other action.
6. Collusion is not permitted: The submitted work should be **solely** of your own completion in accordance of Section 2.5 of the Academic Offences guidelines. You are not permitted to work with other students or third parties e.g. using online forums or pay-for-solution websites.
7. A guide to academic offences for students can be viewed: https://www.qub.ac.uk/directorates/AcademicStudentAffairs/AcademicAffairs/AppealsComplaintsandMisconduct/AcademicOffences/Student-Guide/Links_to_an_external_site.

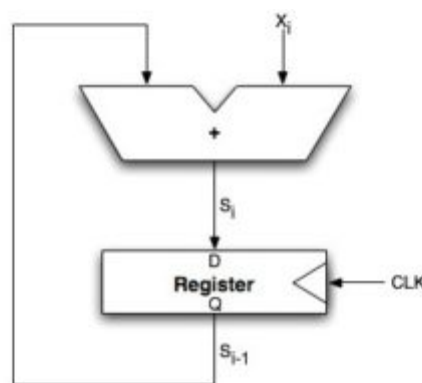
8. This is an open book and open resource assignment. You are allowed to access books and online resources. However, you must attribute sources (see point 6) and the solutions must be in your own words.
9. *Attribution:* If you need to cite any sources/books (standard definitions do not require a citation), have a separate **references** section at the end.
 - If you require clarity on questions, please email Dr Molahosseini (a.sabbaghmolahosseini@qub.ac.uk).

Project Title:**FPGA Design of a Simple Processing Unit for a Self-Checkout machine****Overall Target:**

The overall target of this project is to design a simple processing Unit for a self-checkout machine for a shop. Each task completes one part of this customized processor based on the previous min-lab tests and tutorials, and the last task integrates all of the parts in a unified top module of a customized processor for a self-checkout machine.

**Task 1. The Accumulator Unit (Total 42 marks)**

We need an accumulator unit to add the price of the new item to the current total amount. An adder followed by a register can be used as follows to address this requirement.



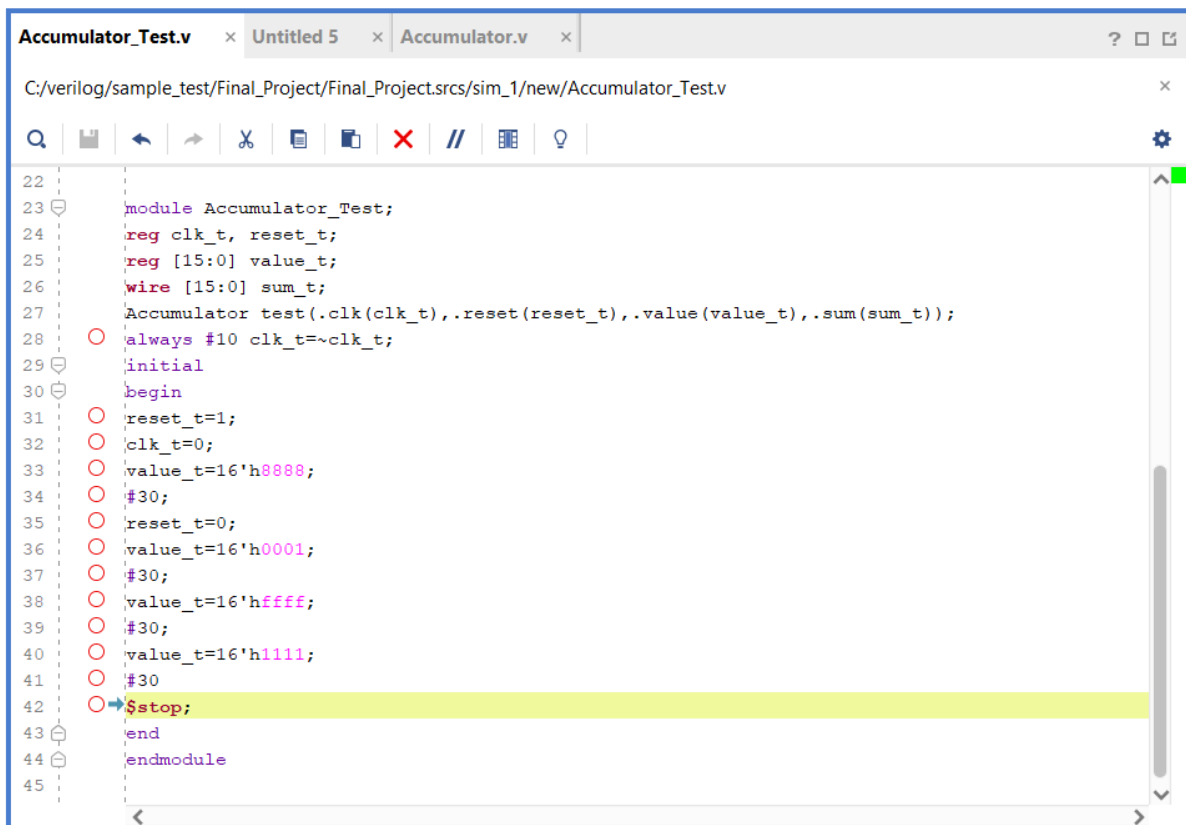
1.1. Write the Verilog code of a 16-bit Accumulator unit with the following input and outputs (12 marks):

module **Accumulator** (input **clk**, input **reset**, input [15:0] **xi**, output wire [15:0] **si**);

Hints:

- 1- According to the picture, the output is s_i (adder's result) which is calculated as $x_i + s_{i-1}$. An activated **reset** signal makes the accumulator output (i.e. s_i) to zero, otherwise, the output will be the addition of input and the previous value stored in the register ($x_i + s_{i-1}$).
- 2- We assume no overflow will happen. Therefore, you can ignore the carry-out of the addition.

1.2. . Then check your code using the following testbench, and insert a screenshot of the **behavioral simulation** result (12 marks).



```

22
23 module Accumulator_Test;
24     reg clk_t, reset_t;
25     reg [15:0] value_t;
26     wire [15:0] sum_t;
27     Accumulator test(.clk(clk_t),.reset(reset_t),.value(value_t),.sum(sum_t));
28     always #10 clk_t=~clk_t;
29     initial
30     begin
31         reset_t=1;
32         clk_t=0;
33         value_t=16'h8888;
34         #30;
35         reset_t=0;
36         value_t=16'h0001;
37         #30;
38         value_t=16'hffff;
39         #30;
40         value_t=16'h1111;
41         #30;
42         $stop;
43     end
44 endmodule
45
  
```

1.3. Run Synthesis and Implementation. Then, **change the previous testbench** (do not change value_t values) to have a **correct Post-Implementation Functional Simulation**, and insert a screenshot of the result (6 marks).

1.4. Task screenshots from **RTL Schematic** and **Implementation Schematics**, and then clearly **describe why** they are different (12 marks).

Task 2. The Read Only Memory (ROM) unit (Total 35 marks)

We need a simple ROM unit to store the code and price of items, and then retrieve the price of an item by sending its code as an address to the ROM. Suppose we have 16 items in the store; therefore, a ROM with 4 bits address is enough (since $16=2^4$). Moreover, the price of items is less than £65536. Therefore, 16 bits for the data field of the ROM is enough (since $65536=2^{16}$).

2.1. Write a **Verilog code** for the above-mentioned **ROM unit** based on the following code and price **table** (12 marks).

*module **ROM** (input **clk**, input [3:0] **address**, output reg [15:0] **data**);*

Code	Price
0	0
1	25
2	4
3	100
4	12
5	5
6	7
7	91
8	25
9	44
10	17
11	55
12	200
13	11
14	97
15	242

2.2. Write a **test bench** to test **all the possible cases**, and Show the **Behavioral Simulation** waveforms where output values showed on the data waveform (use zoom in to see numbers on the waveform, and then take multiple screenshots to show the corresponding outputs against all input values). It should be noted that price numbers in this table are in decimal form, but they will be represented in hexadecimal on your simulation waveforms (13 marks).

2.3. Run synthesis and implementation. Then, complete the following **Table** based on **VIVADO reports**, and show screenshots from these data inside reports (remember to also attach full report files to the project ZIP file) (10 marks).

<i>Parameter</i>	<i>Value</i>
Total Number of used Slices	
Total Number of used LUTs	
Total Number of used FLIPFLOPS	
Maximum Delay Path	
Total Power Consumption	

Task 3. A Customized Subtraction Unit (Total 30 marks)

We need a subtractor to calculate the change during the payment step. In other words, after adding items, in the payment stage, the user will enter some coins or banknotes into the machine, and then the change (the difference between the received money and the total price (balance input)) should be calculated. This process needs a subtractor.

Here, we are going to design a **customized subtraction unit**. First, your circuit needs to check **if the entered money is higher or equal to the total balance**, in this case, the output is the calculated change, and a signal **success** should be set as **1**. Otherwise, if the entered money is **less** than the balance, then the change output should be **0**, and the signal **success** should be **0**.

3.1. Write a Verilog code for the above-mentioned **customized subtraction unit** (12 marks).

```
module Subtraction (input [15:0] money, input [15:0] balance, output [15:0] change,  
output success);
```

3.2 Use the following test bench to show **Post-Synthesis Functional Simulation** and **Post-Implementation Timing Simulation**. Now, clearly describe **where** they are different, and **why** they are different (18 marks).

```

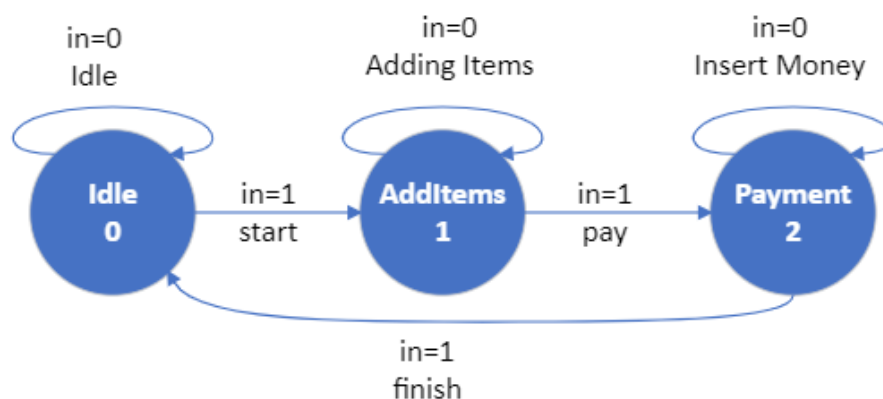
Subtraction_Test.v x Subtraction.v x Untitled 4 x
C:/verilog/sample_test/Final_Project/Final_Project.srcs/sim_1/new/Subtraction_Test.v

20 //////////////////////////////////////////////////
21
22
23 module Subtraction_Test;
24   reg [15:0] money_t, balance_t;
25   wire [15:0] change_t;
26   wire success_t;
27   Subtraction test(.money(money_t),.balance(balance_t),.change(change_t),.success(su
28   initial
29   begin
30     balance_t=16'd10;
31     money_t=16'd15;
32     #50;
33     balance_t=16'd10;
34     money_t=16'd7;
35     #50;
36     $stop;
37   end
38 endmodule
39

```

4. Finite State Machine (Total 43 marks)

Now, we are going to define the top module of the self-checkout machine based on the following finite-state machine (FSM). In each rising edge of the clock, the system state should be checked and updated according to the *in* signal. Moreover, the previous modules should be appropriately instantiated.

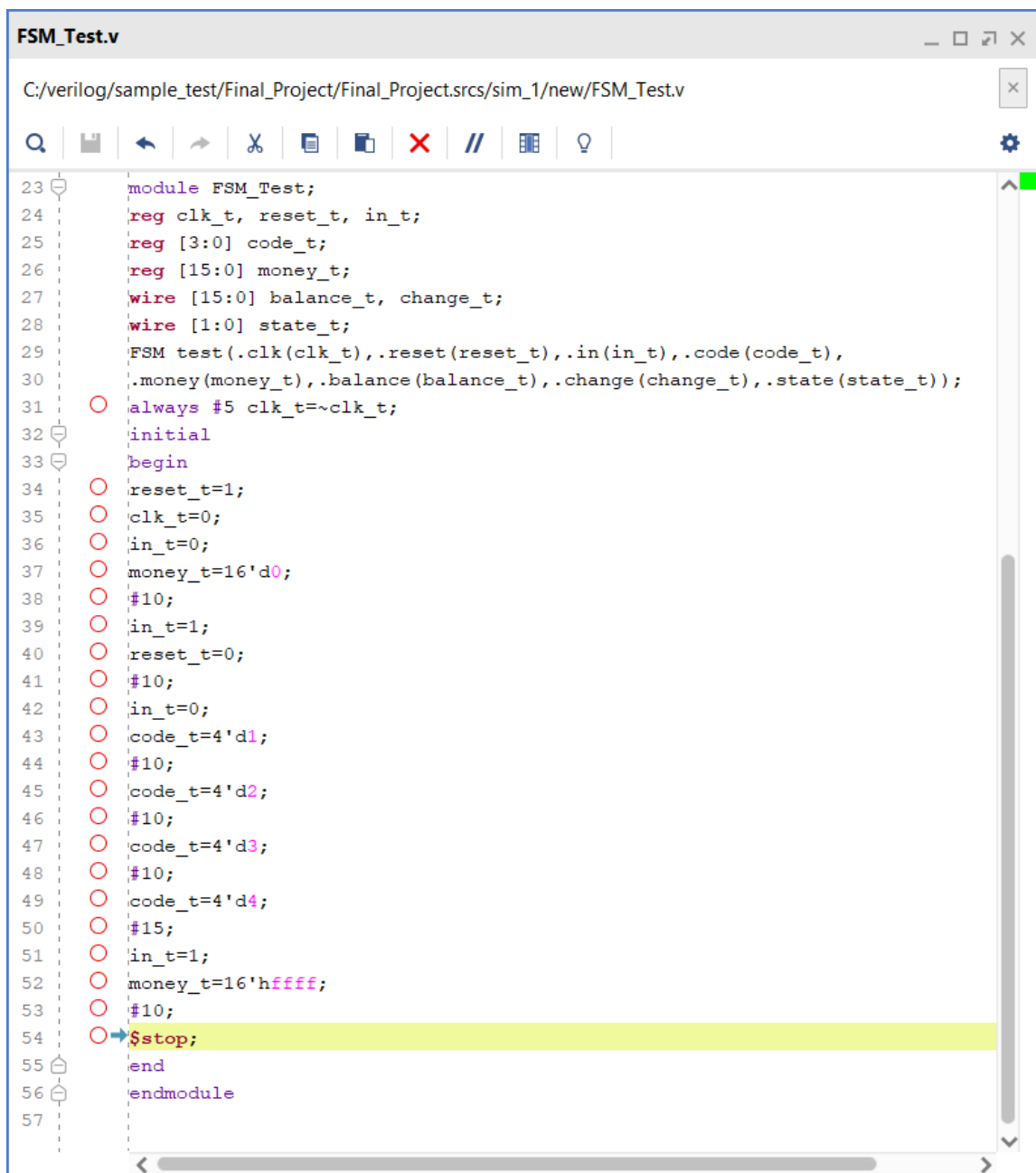


4.1. Write a **Verilog code** for the above-mentioned **Finite State Machine** (20 marks).

module FSM (clk, reset, in, code, money, balance, change, state);

```
    input clk, reset, in;
    input [3:0] code;
    input [15:0] money;
    output [15:0] balance, change;
    output reg [1:0] state;
    parameter idle=0, additems=1, payment=2;
```

4.2. use the following **test bench** to test your code, and insert a screenshot of your **Behavioral Simulation** (5 marks).



The screenshot shows a Verilog test bench named `FSM_Test.v` in a text editor. The code defines a test module that instantiates the `FSM` module and applies various test stimuli to its inputs. The test bench includes a clock signal, reset, and data inputs for code, money, balance, change, and state. The test sequence starts with a reset, followed by a series of input changes and delays, and ends with a `$stop` command.

```

23 module FSM_Test;
24     reg clk_t, reset_t, in_t;
25     reg [3:0] code_t;
26     reg [15:0] money_t;
27     wire [15:0] balance_t, change_t;
28     wire [1:0] state_t;
29     FSM test(.clk(clk_t), .reset(reset_t), .in(in_t), .code(code_t),
30             .money(money_t), .balance(balance_t), .change(change_t), .state(state_t));
31     always #5 clk_t=~clk_t;
32     initial
33     begin
34         reset_t=1;
35         clk_t=0;
36         in_t=0;
37         money_t=16'd0;
38         #10;
39         in_t=1;
40         reset_t=0;
41         #10;
42         in_t=0;
43         code_t=4'd1;
44         #10;
45         code_t=4'd2;
46         #10;
47         code_t=4'd3;
48         #10;
49         code_t=4'd4;
50         #15;
51         in_t=1;
52         money_t=16'hffff;
53         #10;
54         $stop;
55     end
56 endmodule
57

```


4.3. Show the **RTL Schematic** and describe why the full system design is correct, and the required parts are **connected correctly** (13 marks).

4.4. Run **synthesis and implementation**, and complete the following **Table** based on **VIVADO reports** (remember to attach text files of reports to the project ZIP file) (5 marks).

<i>Parameter</i>	<i>Value</i>
Total Number of used Slices	
Total Number of used LUTs	
Total Number of used FLIPFLOPS	
Maximum Delay Path	
Total Power Consumption	