

Faculty of Science and Technology

Middlesex University



Group 1 Project Report

Felix Mabale Coral
M00758209

Adam Sparkes
M00759855

Andrei-Paul Tocu
M00797763

Supervisor: **Dr. Purav Shah**

PDE2102 Digital System Design

BEng Computer System Engineering

March 2022

Table of Contents

1. Abstract.....	3
2. Introduction.....	3
3. Creating Project in Xilinx Vivado Suite	3
4. Task 1.....	4
4.1. The task.....	4
4.2. Data representation	4
4.3. Implementation	5
4.4. Simulation, testing and improvements.....	6
4.5. NEXYS4 DDR – Demo.....	9
5. Task 2.....	9
5.1. The task.....	9
5.2. Initial design	9
5.3. Implementation	13
5.4. Simulation and testing	14
5.5. NEXYS4 DDR – Demo.....	15
6. Challenges faced.....	15
6.1. Using splashtop.....	15
6.2. Vivado.....	15
7. Group Project Meeting and Progress Form	16

1. Abstract

In this group project of PDE2102 Digital Systems Design we are answering the two questions our group was asked, providing a brief introduction to the tasks, a detailed explanation of our design, truth tables, k-map figures, implementation steps, snapshots, testing and a demo on DIGILENT Nexys4 DDR.

2. Introduction

Prior to starting to work on the tasks we need to specify that previous knowledge from laboratories 1 – 5 was required in order to accomplish the tasks.

Labs 1 and 2 introduces us to the basics of VHDL and how to use the code and declare variables. It goes from the libraries we used in our code, both main and testbench to the different types of *std_logic* and *std_logic_vector*. This is important to understand what either of them do as both these are included in our code.

Lab 4 is the skeleton code on which our code is based on mainly due to having a clock, signals dependant on the clock and a counter which lets us understand the high state of the clock on which the work is based on. It gives an insight of the different signal declarations to use, one being used in our code is the counter, that makes the clock count upwards.

3. Creating Project in Xilinx Vivado Suite

In order to develop and implement the code, testing it, performing simulations and demo the code on a physical board for any of the tasks, we need to have a project created in Vivado:

1. Install and open Xilinx Vivado Suite 2020.2 / 2020.1
2. Create a New Project
 - 2.1. Project name: firstquestion ; select Create project subdirectory ; click next
 - 2.2. Project type: RTL Project ; click next
 - 2.3. Add Sources:
 - 2.3.1.click Create File ; File type: VHDL ; File name: firstquestion ;
 - 2.3.2.click Create File ; File type: VHDL ; File name: firstquestion_tb ;
 - 2.3.3.Target Language: VHDL ; Simulator Language: VHDL ; click next
 - 2.4. Add Constraints: click + sign and add file firstquestion.xdc ; click next

The XDC constraint file assigns the physical IO locations on FPGA to the switches and LEDs located on the board.
 - 2.5. Default Part:

2.5.1. Select "Boards" and search for "Nexys4 DRR" and select it ; click next

Or

2.5.2. Under "Parts" make the following configuration:

2.5.2.1. Product Category select General Purpose

2.5.2.2. Family select Artix-7

2.5.2.3. Package select "csg324"

2.5.2.4. Speed select "-1"

2.5.2.5. From the list of entries left, select "xc7a100tcsg324-1"

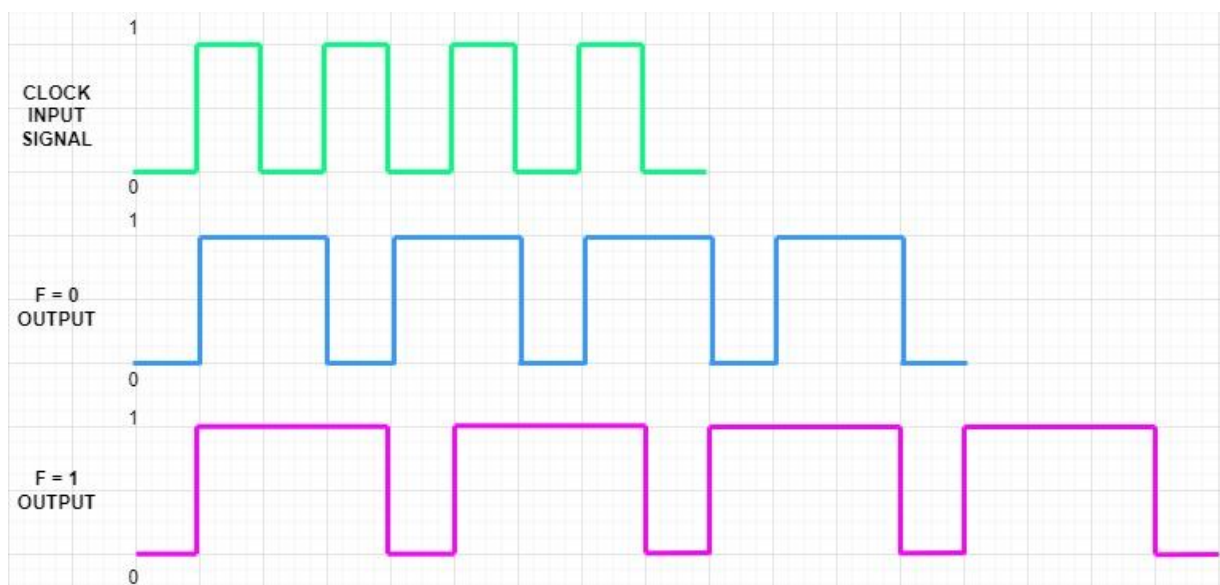
3. Click Finish

4. Task 1

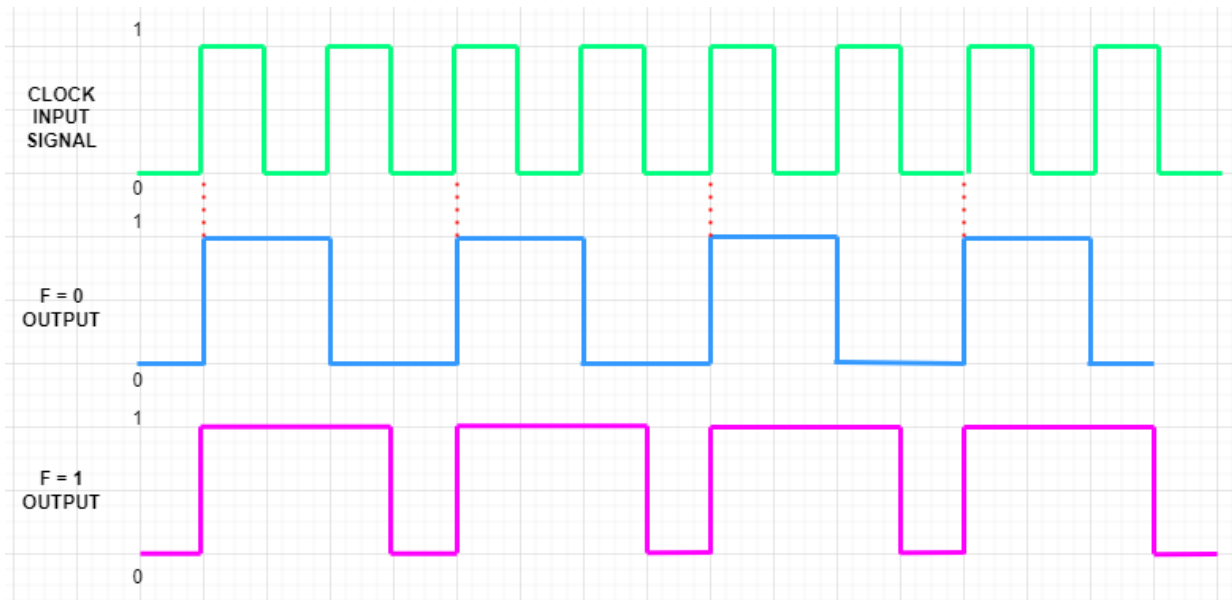
4.1. The task

Write a VHDL module to implement a circuit that can generate a clock signal whose time period is a multiple of the input clock. A control signal F determines the multiplying factor. If $F = 0$, the output signal has a time period twice that of the input clock. If $F = 1$, the output signal has a time period three times that of the input clock. The portion of the clock cycle when the clock is 1 may be longer than the portion when it is 0, or vice versa. Use a counter with an active-high synchronous clear input. Show the design along with your implementation. Present the implementation with timing simulations as well as Nexys4 implementation. Show how will you control the inputs and how will you represent the outputs.

4.2. Data representation



If the task was understood correctly, the above drawing should represent an actual input and output scenario, dependant on F 's value. But it doesn't respect the "use a counter with an active-high synchronous clear input", meaning that the state of the output should be in sync with the state of the input. In the image below the requirement is outlined by the red dotted line, and more obvious for $F=0$, when the output is double the input whilst the rising edges of both input and output are in sync.



4.3. Implementation

The file *firstquestion.vhd* – will be referring to it as main file – contains the main logic and mechanics on which the task is based on, whilst *firstquestion_tb.vhd* – referring to it as testbench (tb) file – serves as a testbench, being the source of all the signals that will be used in the *firstquestion.vhd*. Input signals such as *clk_in* and F are all sent from the testbench file, received, and put in action by the main file.

Constraint's file, *firstquestion.xdc*, is what helps mapping the input and outputs from our code and simulations to the pins, switches, LEDs etc. on the physical board.

Testbench contents:

```
clock : PROCESS
BEGIN
clk_in <= NOT clk_in;
wait for clk_period;
END PROCESS;

Fval : PROCESS
BEGIN
F <= '0';
wait for clk_period*50;
F <= '1';
wait for clk_period*50;
END PROCESS;
```

For a simulation of 1000ns the clk_in will change 100 times between 0 and 1, and F will change just once from 0 to 1 (500ns each state).

Their values are sent over to the main file using the “Unit Under Test” block:

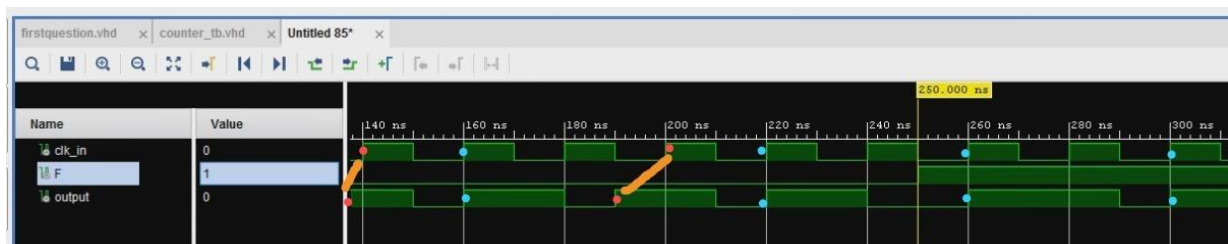
```
 uut: firstquestion PORT MAP(
 clk_in => clk_in,
 F => F,
 output => output,
 count_out => count_out
 );
```

The main file receives these values by having the following variable as IN or OUT ports:

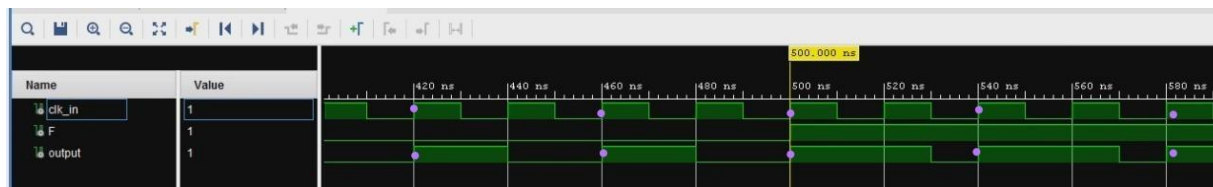
```
entity firstquestion is
port ( clk_in      : in  STD_LOGIC;
       F          : in  STD_LOGIC;
       count_out  : out STD_LOGIC_VECTOR (3 downto 0);
       output     : out STD_LOGIC);
end firstquestion;
```

We all worked on the code for this, having different variations of variables but overall Andrei in the group came up with the finalised code which will be displayed in the screenshots further into the report. Below are the steps we went through to get the answer for task 1 and shows how we did it.

4.4. Simulation, testing and improvements



Here can be seen one of the previous iterations of the code in which it is fairly clear that when $F=0$, a clk_in of 10ns is out of sync with the output. The only reason it is not out of sync for $F=1$ is just a coincidence of timing. If we were to change the timing of clk_in , making it random will definitely make output out of sync. Dots of the same colour should be below each other, like this:



This is the simulation of our final code. The output is in sync with the rising edge of input.

Final code is made so that it will count up to 2("10") for $F=0$ and up to 3("11") for $F=1$. When it reaches said values, both the counter and output will be set to "00" and 0 respectively. The code will only start counting if clk_in is on a rising edge and if counter's value is "00". Once counter is not "00" anymore, it will just get incremented whenever there is an event on clk_in (rising/falling).

```

q1_output : process (clk_in)
begin
  if (F = '0' AND countertemp = 2) OR (F='1' AND countertemp = 3) OR
F'event then
    output <= '0';
    countertemp <= "0000";
  else
    if countertemp = 0 then
      if clk_in='1' then
        output <= '1';
        countertemp <= countertemp + 1;
      end if;
    else --elsif countertemp > 0 then
      countertemp <= countertemp + 1;
    end if;
  end if;
  count_out <= countertemp;
end process;

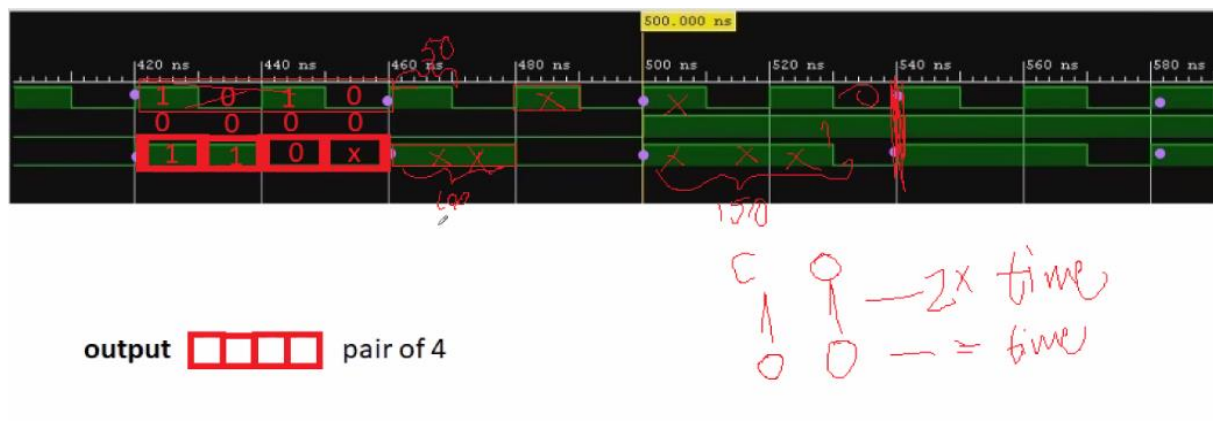
```

It can also be seen that the output and counter are set to 0 whenever the statement $F'event$ is true. This is because we don't want any current or ongoing counting to overlap into different values of F . For example, when $F=1$, $clk_in=0$ and counter "01", then F becomes 0, $clk_in=1$ and counter is incremented to "10" and then is reset. Maybe this display will help visualise the problem better:

clk_in	0	1	0	1	0	1	0
F	1	1	1	0	0	0	0
output	0	1	1	1	0	1	1
counter	00	00	01	10	11	00	01

With red are marked the values that overlap in the new state of F .

Since the task consists of keeping the same input but only changing its duration it is difficult generating a truth table for this problem. Input equals output pretty much. We tried a few times thinking about the output as going into pairs of 4, represented in this screenshot, with our doodling from our meeting:



But of course, this would involve recreating a whole new code, that would respect this kind of output. Yet, after a couple more meetings we agreed that we should continue working with what we already had. This being the closest thing we came up with for representing our data in form of a truth table:

INPUT		aux variable		OUTPUT
		COUNT[X,Y]		
clk_in	F	X	Y	
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	N/A	N/A	N/A
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 1 Truth Table for Task 1

N/A – the algorithm will never get to count to “11” when $F=0$.

4.5. NEXYS4 DDR – Demo

In order to implement the main code onto the Nexys4 DDR board, we need to map the variables used as input and output so the switches and LEDs present on the board. Nexys4 DDR is a digital circuit development platform for FPGA (Field Programmable Gate Array), property of Digilent.

For this task, the variable F takes as input values from the left most switch, V10, and the output will use the LEDs H17 (LSB) and K15 (MSB). The output represents the number to which the counting got up to.

Switch V10 will input a HIGH value for F when it is pushed up, and LOW when it's pushed down.

```
##LEDs
set_property -dict { PACKAGE_PIN J15    IOSTANDARD LVCMOS33 } [get_ports { count_out[0]
}]; #LSB input - RIGHT MOST SWITCH, 15
set_property -dict { PACKAGE_PIN L16    IOSTANDARD LVCMOS33 } [get_ports { count_out[0]
}]; #MSB input
##Switches
set_property -dict { PACKAGE_PIN V10    IOSTANDARD LVCMOS33 } [get_ports { F }]; #F
input
```

5. Task 2

5.1. The task

A circuit has four inputs A, B, C and D. The four inputs are applied a circuit that converts these into an excess-3 form. However, when the excess-3 form input is applied to the 7-segment decoder, the 7-segment LED displays the original BCD inputs (ABCD) as output on it. Show the design and implementation steps. Use the Nexys4 on-board clock and slow it down, so that all combinations are visible at a slow rate. Also, perform a timing simulation using testbench for all the BCD/excess-3 combinations. 3

Hint: To convert any decimal number into its excess- 3 form, add '3' to each decimal digit and then convert the sum to a BCD number – For example, decimal number '3' has excess-3 form of '6', i.e., it is '0110' instead of '0011'. This will be used as input to the 7-segment decoder. However, '3' will be displayed on the 7-segment display, as original BCD number.

5.2. Initial design

In the truth table it shows the inputs of A, B, C, D the Excess-3 form and the display it shows on the 7-segment decoder

	A	B	C	D	E	F	G	H	I	J	K	L
1	Excess 3											
2	A	B	C	D	A	B	C	D	Decimal	Display on 7-segment decoder		
3	0	0	0	0	0	0	1	1	3	0		
4	0	0	0	1	0	1	0	0	4	1		
5	0	0	1	0	0	1	0	1	5	2		
6	0	0	1	1	0	1	1	0	6	3		
7	0	1	0	0	0	1	1	1	7	4		
8	0	1	0	1	1	0	0	0	8	5		
9	0	1	1	0	1	0	0	1	9	6		
10	0	1	1	1	1	0	1	0	10	7		
11	1	0	0	0	1	0	1	1	11	8		
12	1	0	0	1	1	1	0	0	12	9		
13												

This meaning if the input is 0011 (3) the excess 3 digit would be 6 as it adds 3 to the initial binary digit, but the display on the 7-segment decoder should be the original input 0011(3).

Following on from the truth table are K-maps, I have 7 outputs as it's a 7-segment decoder it has 7 ways to show the digit.

A.

WX/YZ	00	01	11	10
00	1			1
01	1	1	X	1
11	X	X	X	X
10	X	X	X	X

$$A = \text{NOT}(Z) + \text{NOT}(W)X$$

B.

WX/YZ	00	01	11	10
00	1	1		
01	1	1	X	1
11	X	X	X	X
10	X	X	X	X

$$B = X + \text{NOT}(Y)$$

C.

WX/YZ	00	01	11	10
00	1	1	1	1
01	1	1	X	1
11	X	X	X	X
10	X	X	X	X

$$C = \text{NOT}(W)$$

D.

WX/YZ	00	01	11	10
00	1		1	1
01		1	X	1
11	X	X	X	X
10	X	X	X	X

$$D = Y + \text{NOT}(X, Y, Z) + XZ$$

E.

WX/YZ	00	01	11	10
00			1	
01		1	X	
11	X	X	X	X
10	X	X	X	X

$$E = XZ + YZ$$

F.

WX/YZ	00	01	11	10
00		1	1	1
01		1	X	1
11	X	X	X	X
10	X	X	X	X

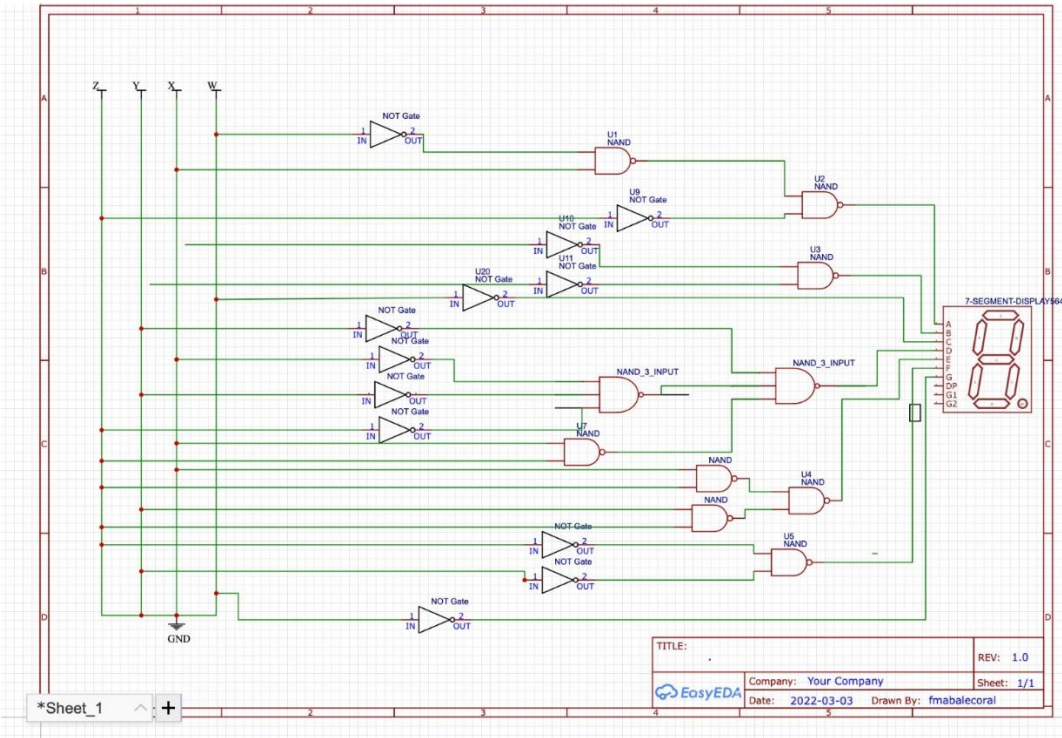
$F = Z + Y$

G.

WX/YZ	00	01	11	10
00	1	1	1	1
01	1	1	X	1
11	X	X	X	X
10	X	X	X	X

$G = \text{NOT}(W)$

We also created the circuit design from what we have from our truth table and k-maps



5.3. Implementation

The file secondquestion.vhd will be referred as the main code. In the main code it contains the main logic, variables and overall concept of what task 2 requires us to achieve. Secondquestion_tb is the testbench for the main code. The reason for a test bench file is to test the behaviour of our main code, testing its logic and different parameters we have set. In the test bench file, it has been set to the amount of time clk_period runs for which is 10000ps. Below is the main code, as mentioned it has the main logic case EX3 is the mapping of what it shows on the LED display for example for the display to output the digit 0 it has to light up only the edges of the 7-segment LED's this is done with the binary value 0111101.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity secondquestion is
PORT(
SIGNAL CLK,A,B,C,D : IN STD_LOGIC;
SIGNAL BIN: IN STD_LOGIC_VECTOR(3 downto 0);
SIGNAL EX3 : IN unsigned(3 downto 0);
SIGNAL BCD : IN integer;

SIGNAL LEDs : OUT STD_LOGIC_VECTOR(6 downto 0) -- [x,x,x,x,x,x,x] =
[a,b,c,d,e,f,g]
);
end secondquestion;

architecture Behavioral of secondquestion is
begin

valLED : PROCESS (CLK)
BEGIN
CASE EX3 IS
WHEN "0000" => LEDs <= "0111101"; -- EX3 = 0, LED shows d
WHEN "0001" => LEDs <= "1001111"; -- EX3 = 1, LED shows E
WHEN "0010" => LEDs <= "1000111"; -- EX3 = 2, LED shows F

WHEN "0011" => LEDs <= "1111110"; -- EX3 = 3, LED shows 0
WHEN "0100" => LEDs <= "0110000"; -- EX3 = 4, LED shows 1
WHEN "0101" => LEDs <= "1101101"; -- EX3 = 5, LED shows 2
WHEN "0110" => LEDs <= "1111001"; -- EX3 = 6, LED shows 3
WHEN "0111" => LEDs <= "0110011"; -- EX3 = 7, LED shows 4
WHEN "1000" => LEDs <= "1011011"; -- EX3 = 8, LED shows 5
WHEN "1001" => LEDs <= "1011111"; -- EX3 = 9, LED shows 6
WHEN "1010" => LEDs <= "1110000"; -- EX3 = 10(A), LED shows 7
WHEN "1011" => LEDs <= "1111111"; -- EX3 = 11(b), LED shows 8
WHEN "1100" => LEDs <= "1111011"; -- EX3 = 12(C), LED shows 9
WHEN "1101" => LEDs <= "1110111"; -- EX3 = 13(d), LED shows A
WHEN "1110" => LEDs <= "0011111"; -- EX3 = 14(E), LED shows b
WHEN "1111" => LEDs <= "1001110"; -- EX3 = 15(F), LED shows C
WHEN others => LEDs <= "1001001"; -- LED shows ? (IDENTICAL) for ERROR
END CASE;
END PROCESS;

end Behavioral;
```

The testbench tests this by using the clk_period as a clock count. Declaring the component secondquestion which is the main code to simulate further down.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity secondquestion_tb is
-- Port ( );
end secondquestion_tb;

architecture Behavioral of secondquestion_tb is
COMPONENT secondquestion
PORT(
SIGNAL CLK,A,B,C,D : IN STD_LOGIC;
SIGNAL LEDs : OUT STD_LOGIC_VECTOR(6 downto 0);
SIGNAL BIN : IN STD_LOGIC_VECTOR(3 downto 0);
SIGNAL EX3 : IN unsigned(3 downto 0);
SIGNAL BCD : IN integer
);
END COMPONENT;

SIGNAL CLK,A,B,C,D : STD_LOGIC := '0';
SIGNAL LEDs : STD_LOGIC_VECTOR(6 downto 0) := (others=>'0');
SIGNAL BIN : STD_LOGIC_VECTOR(3 downto 0) := (others=>'0');
SIGNAL EX3 : unsigned(3 downto 0);
SIGNAL BCD : integer := 0;
constant clk_period : time := 10 ns;

begin
uut: secondquestion PORT MAP(
A => A,
B => B,
C => C,
D => D,
LEDs => LEDs,
BIN => BIN,
BCD => BCD,
EX3 => EX3,
CLK => CLK
);

clock : PROCESS
BEGIN
CLK <= NOT CLK;
wait for clk_period;
END PROCESS;

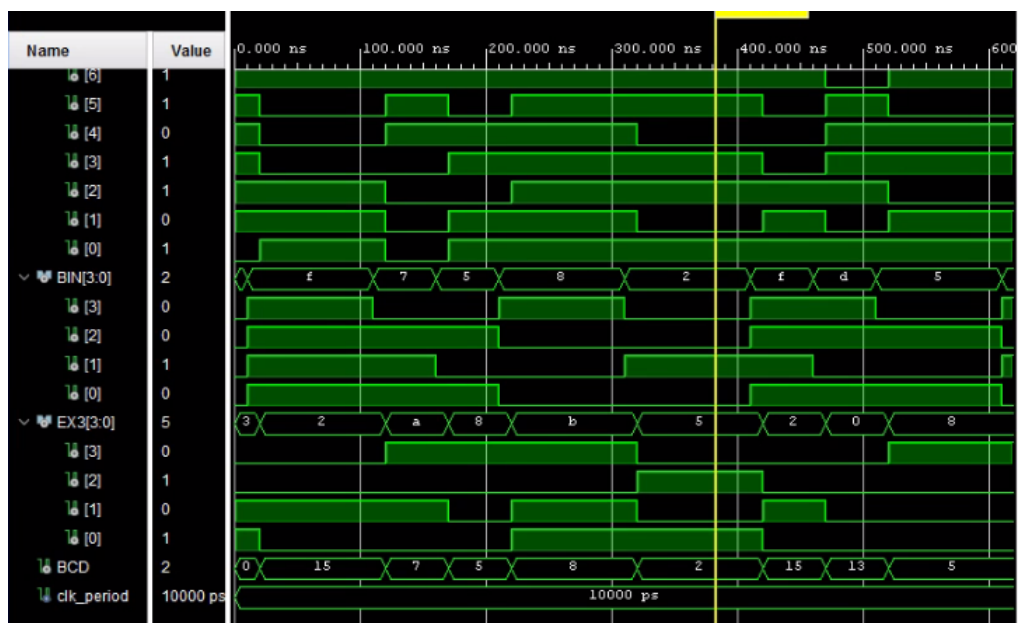
valA : PROCESS
BEGIN
A <= NOT A;
wait for clk_period*10;
END PROCESS;

valB : PROCESS
BEGIN
B <= NOT B;
wait for clk_period*20;
END PROCESS;

valC : PROCESS
BEGIN
C <= NOT C;
wait for clk_period*15;

```

5.4. Simulation and testing



This screenshot is from when we run simulation from our main code and testbench file, the binary number 0010 (2) going up to 0101 (5) with the excess 3 signal, but in the 7-segment part it still shows the output as 2 with the values when it is 1 lighting up the leds to display the number 2 on the LED display. This runs for 10000ps, shown from clk_period.

5.5. NEXYS4 DDR – Demo

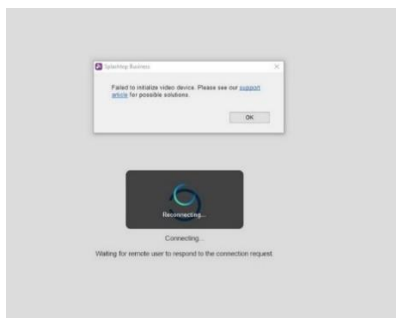
To implement the code onto the Nexys4 DDR board, we had to declare the variables in the constraints file and map the pin to the right switch of which it is on the board. Doing this allowed up to run bitstream in Vivado and program the device.

Once programming the device we found out we had issues with the code as it turned the LED on when the output is a '0' but we had coded it the opposite way around thinking it turned the LED on in the 7-segment decoder when the output is a '1'.

6. Challenges faced

6.1. Using splashtop

When going through the tasks using Splashtop we encountered many issues which affected our progress, issues ranging from connection problems to planned restarts of the computers we are connecting to. Alongside the issues mentioned we also faced many computers not having the Vivado software on them which made it confusing to which computer to use. On many occasions it shown this message below resulting us having to sign into Splashtop again and losing our progress on the code.



6.2. Vivado

Using Vivado also proved to be quite difficult. Not only that it sometimes takes a long time to simulate the code but generating the bitstream takes way much longer. And all of us being beginners in this domain, it is hard to visualise most of the things, so we rely on constantly simulating. And not only that it takes a long time, but also it is very rare that the errors are detailed or useful, most of the time saying it failed and not much else, so we resort to trial on error, commenting and uncommenting pieces of code.

7. Group Project Meeting and Progress Form

Group Members: 3

Names and IDs:

Adam Sparkes (M00759855)

Andrei Tocu (M00797763)

Felix Mabale Coral (M00758209)

Planned Meetings for Group Coursework Component (extend this section as needed):

(Copy paste emails, WhatsApp Group conversations or any other evidence of planned meetings – you can include screenshots)

For each meeting provide the agenda, the input (if any) of each member, and the output/conclusions of the meeting. Provide information also if the meeting was scheduled and some members did not show up or postponed the meeting.

19/02/22

Adam Sparkes asked for a group meeting to be done that day or the next one

Andrei Tocu and Felix Mabale both agreed to have meeting on 20/02/22 at 5pm

20/02/22

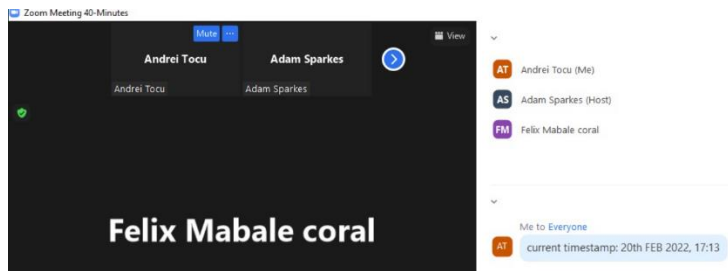
All three of us met on Zoom and discussed the project

Adam shared his screen and started looking for the previous labs comparing what labs are relevant towards our questions and appropriate for us to all look

Adam managed to find **week 7 lab 4** appropriate for task 1 and **week 10 lab 5** for task 2

Group 1 Project Report. F.M. Coral, A. Sparkes, A.P. Tocu, 2nd year

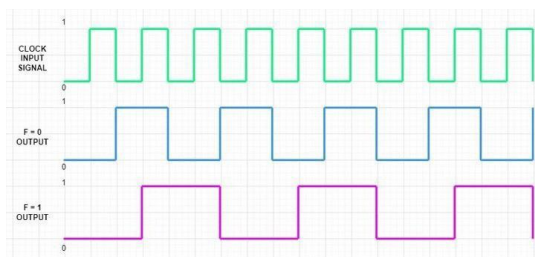
We finished meeting just before 18:00



Adam working through splash top to access Vivado programme to test out week 10 lab 5 to understand the 7-segment decoder for Q2

24/02/22

Had a meeting to go through a lab to understand what was going on. Running Testbenches to try and get the result. Understood Q1 for what it should look like:



Meeting 01/03/22

We had a meeting to go over the report and the progress on task 2. Finalising task 1. Getting all the screen shots together and putting them in the report. Also shared the report with everyone




Meeting 02/03/22



Adam and Andrei had video calls to work on the report and finalise the code for task 2.

Other multiple, unannounced, and short meetings, that were not documented here, took place just to show each other bits and pieces that could help the group work and make constant progress.

Final Tasks (who did the actual work)

Member Name	Task Done	Signature
Adam Sparkes	Wrote code snippets for task 2 and did the truth table and k-maps. Kept track of progress and developed the group meeting form. Wrote the report introduction and answer for task 2 including all the screenshots. Kept track of the meeting form and arranged some meetings.	
Andrei Tocu	Designed the final code for task 1 and developed its truth table, formatted the task 1 report, and wrote the documentation for task 1 including all the screenshots and code snippets. Kept track of meeting form, participated, and provided more information.	
Felix Coral	Helped write the report for task 2, provided code ideas and examples, did research on lab 2, 4 and 5. Developed the circuit for task 2. Participated and initiated meetings.	

Date:03/03/22