EE2026: DIGITAL DESIGN

Academic Year 2020-2021, Semester 1

LAB 3: Sequential Circuits in Verilog - Part 1

OVERVIEW

A sequential circuit is one where the outputs depend on the current inputs and the sequence of past inputs. As a result, a sequential circuit has memory, also called states. In this lab, some basic sequential circuits will be designed to make an LED blink at various speeds.

The pre-requisites for this lab are:

- A very good understanding and application of dataflow modelling and structural modelling in designing modules.
- Knowing how to use the Vivado IDE well.
- Familiarity and knowledge on how to use "*Set as Top*", "*reg*" and "*wire*".

This lab will cover the following:

- Introduction to the implementation of a 1-bit two-to-one multiplexer.
- Using a signal that inverts itself periodically, which shall be called **CLOCK**.
- Making a physical LED blink by using the Basys 3 development board clock signal.

Tasks for this lab include:

- Understanding a 1-bit two-to-one multiplexer that can be used in the graded post-lab assignment.
- Creating a slower clock from a faster clock.
- Having a physical LED blink noticeably on the Basys 3 development board, by using the slower clock.
- Using a switch to make a physical LED blink at two different speeds on the Basys 3 development board.

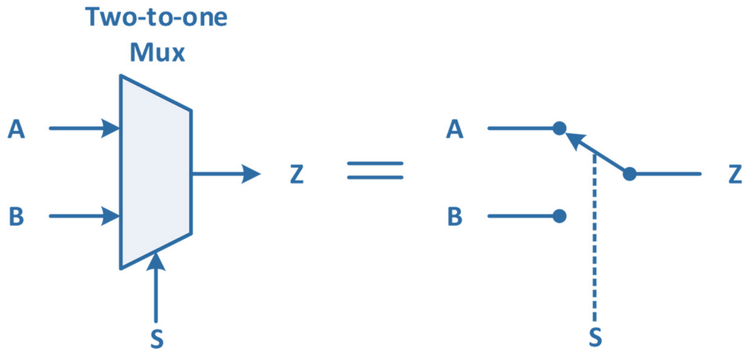
GRADED ASSIGNMENT [LUMINUS SUBMISSION: WEDNESDAY 23rd SEPTEMBER 2020, NOON]:

- Display periodically changing characters on the 7-segment displays of the Basys 3 development board

Further details are available at the end of this lab manual.

1-BIT TWO-TO-ONE MULTIPLEXER [To attempt before the lab session – Not covered in lab]

A multiplexer (MUX) is a combinational circuit that connects one of its input signals to the output, based on the control signal. A simple 1-bit two-to-one mux, with inputs **A**, **B**, control signal **S**, and output **Z**, is illustrated as a functional block diagram, together with its simplified truth table, in **Figure 3.1.**



S	Z
0	A
1	B

Figure 3.1: *Functional block diagram and truth table of a 1-bit two-to-one multiplexer*

UNDERSTANDING | TASK 1

A quick way to implement the Verilog code for a 1-bit two-to-one multiplexer is using the conditional syntax:

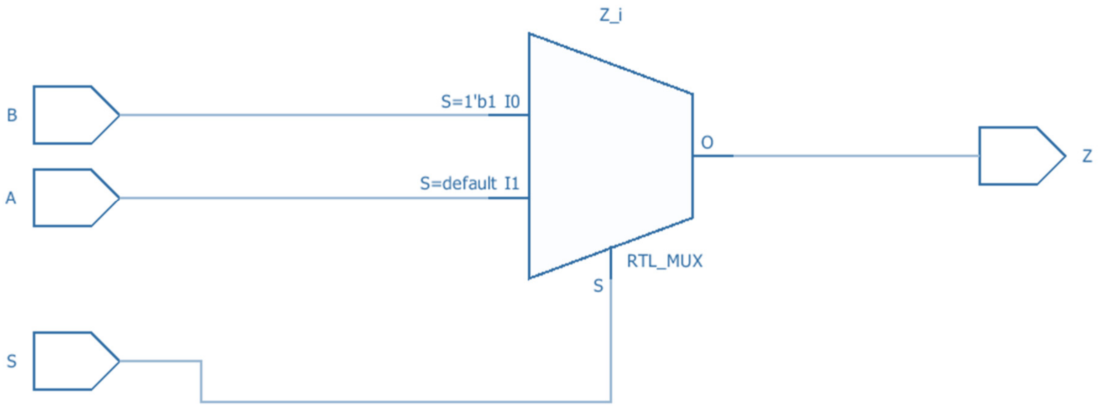
condition ? expression1 : expression2;

Notice in the schematic, how the code is automatically recognised as a MUX.

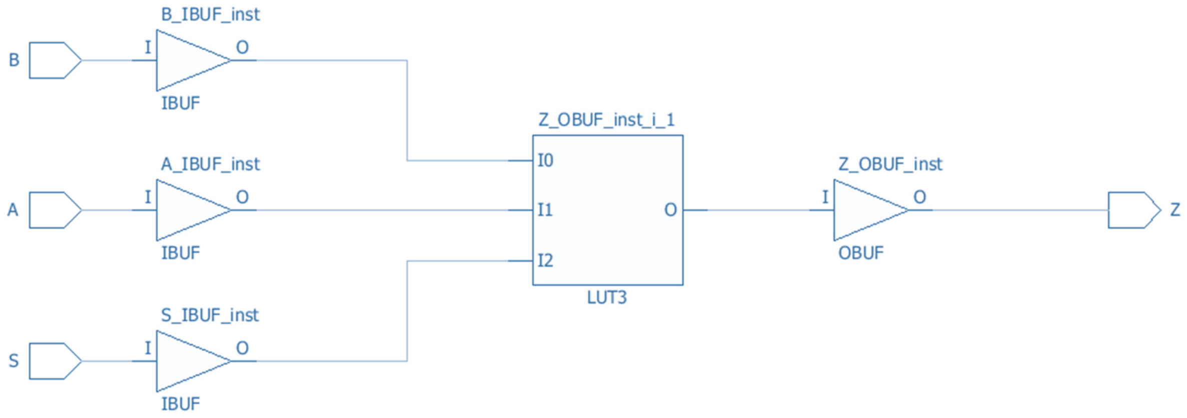
Verilog code for 1-bit two-to-one mux, using the dataflow method

```
module my_2_to_1_mux (input A, B, S, output Z);
    assign Z = S ? B : A; // assign B to Z if S = 1 or assign A to Z if S = 0;
endmodule
```

RTL schematic for the 1-bit two-to-one mux



Synthesised design schematic for the 1-bit two-to-one mux



Fill in the truth table for the **LUT3**, as extracted from the synthesised design schematic for the 1-bit two-to-one mux:

Explain how the truth table for the **LUT3** matches that of the truth table indicated in **Figure 3.1**:

THE BLINKING LED [To attempt before the lab session]

A simple blinking LED is required to be implemented on the FPGA. To do this, a new signal, **CLOCK**, will be introduced.

The **CLOCK** signal is an external input signal that resembles a square wave of 50% duty cycle. If this **CLOCK** signal is connected directly to a physical LED, the latter will light up when the signal is HIGH, and will switch off when the signal is LOW, as illustrated in **Figure 3.2**.

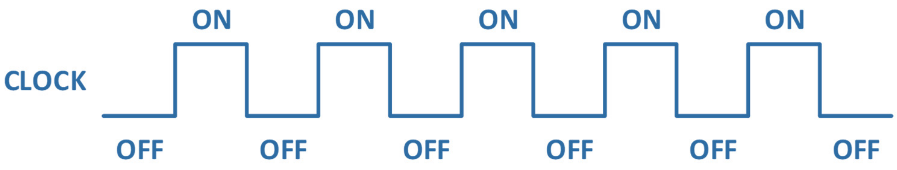


Figure 3.2: *A **CLOCK** signal with 50% duty cycle*

A simple dataflow description in Verilog for a blinky module is written first, followed by a simulation source to verify the design. To create the square wave, or **CLOCK** signal, in the simulation source, a new section of codes will now be introduced:

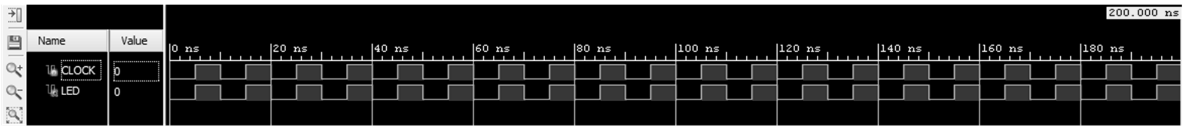
Verilog code for blinky, using the dataflow method

```
module blinky (input CLOCK, output LED);

    assign LED = CLOCK;

endmodule
```

Simulation source code to test the blinky design

```
`timescale 1ns / 1ps

module test_blinky( );

    reg CLOCK; wire LED;

    blinky dut (CLOCK, LED);

    initial begin
        CLOCK = 0;
    end

   always begin
       #5 CLOCK = ~CLOCK;
   end

endmodule
```

Expected simulation waveform for the blinky design



UNDERSTANDING | TASK 2

Based on the Verilog code and simulation results, check your understanding by answering the following questions:

1. What is the unit of time being used in the simulation source?

   ____

2. Every 5 units of time, the value of **CLOCK** is being inverted. What is the clock frequency being used in this simulation?

   ____

3. What would happen if the testbench code **CLOCK = 0** is removed?

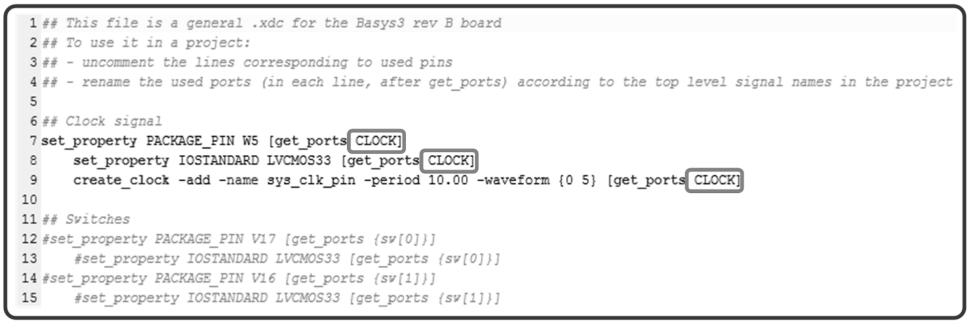
   ____

For the hardware implementation, instead of using an external signal generator for the **CLOCK** signal to the Artix-7 FPGA, the Basys 3 development board includes a single 100 MHz clock generator connected to pin W5 of the Artix-7 FPGA.

Using the original contents of the ***Basys3_Master.xdc*** in your constraint file, follow these steps:

1. Uncomment lines 7 to 9 to create a clock signal of 100 MHz with 50% duty cycle. If required, rename the signal to the name used in your **blinky** code. In our example, the name **CLOCK** was used, and the final changes may look similar to **Figure 3.3**.

2. Configure the output signal **LED** that is present in your **blinky** code (or the name chosen by you while writing the code) by linking it to any physical LED on the Basys3 development board.



```
 1 ## This file is a general .xdc for the Basys3 rev B board
 2 ## To use it in a project:
 3 ## - uncomment the lines corresponding to used pins
 4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
 5
 6 ## Clock signal
 7 set_property PACKAGE_PIN W5 [get_ports CLOCK]
 8     set_property IOSTANDARD LVCMOS33 [get_ports CLOCK]
 9     create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports CLOCK]
10
11 ## Switches
12 #set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14 #set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
```

Figure 3.1: *Modifying your constraint file, based on the contents of the Basys3_Master.xdc*

UNDERSTANDING | TASK 3

You may optionally generate the bitstream and upload your code to the Basys3 development board. What do you "*notice"* about the "*blinking"* LED?

____

____

THE NOTICEABLE BLINKING LED

To be able to observe a blinking LED at a frequency that is visible to the human eyes, modifications need to be done to the Verilog code. Let us introduce a temporary variable **COUNT** that is incremented by 1 at every rising edge (transition from low to high, and also called a positive edge) of the **CLOCK** signal, as shown in **Figure 3.4**. By making use of **COUNT**, a lower frequency signal can be obtained, while the Verilog code for **COUNT** can be created by using the behavioural method of modelling.

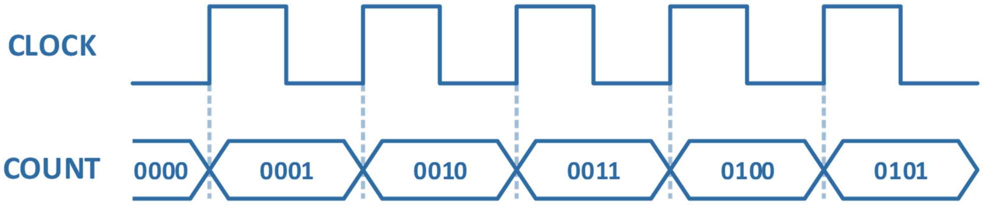


Figure 3.4:** Increasing **COUNT** at each rising edge of **CLOCK

Verilog code for a slower blinky, using behavioural modelling

```
module slow_blinky_module (input CLOCK);

    reg [3:0] COUNT = 4'b0000;

    always @ (posedge CLOCK) begin
        COUNT <= COUNT + 1;
    end

endmodule
```

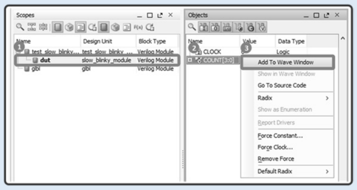
UNDERSTANDING | TASK 4

Create a simulation source for the **slow_blinky_module** design, and observe the waveform of signal **COUNT.**

[NOTE] Analysing a variable in the simulation waveform window

By default, the simulation window only shows the waveforms of input and output signals. To see the waveforms of variables during the simulation, such as the variable **COUNT**:

1. Select the **dut** under the simulation module being used
2. In the **Objects** window, right click on the **COUNT** variable
3. Choose **Add To Wave Window**

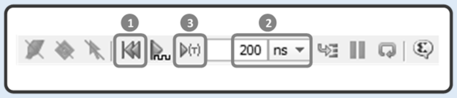


Name	Design Unit	Block Type
test_slow_blinky...	test_slow_blinky_...	Verilog Module
dut	slow_blinky_module	Verilog Module
glbl	glbl	Verilog Module

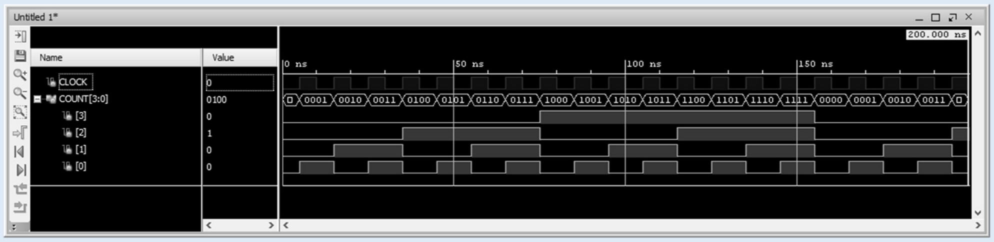
Name	Value	Data Type
CLOCK	0	Logic
COUNT[3:0]		

After adding the variable **COUNT** to the wave window, the current simulation needs to be re-run. Follow these steps:

1. Restart the simulation
2. Set the simulation time and units
3. Run the simulation for the amount of time set in step 2



The **COUNT** variable can then be expanded by clicking on the **+** symbol to the left of **COUNT**. This allows for every individual bit to be observed as independent waveforms:



Name	Value
CLOCK	0
COUNT[3:0]	0100
[3]	0
[2]	1
[1]	0
[0]	0

State the frequency of **COUNT[3]**, **COUNT[2]**, **COUNT[1]** and **COUNT[0]**: __________ , __________ , __________ , __________

UNDERSTANDING | TASK 5 * ***Warning:*** *Do not simulate the code. Why?* *

Using the knowledge and coding obtained from **UNDERSTANDING | TASK 3** and **UNDERSTANDING | TASK 4**, calculate the minimum size of **COUNT** and the bit number which will allow a clock output of around 0.75 Hz.

Minimum of bits required for **COUNT**: ___________ Bit number of **COUNT**: ___________

Name that waveform with the frequency of 0.75 Hz as **SLOWCLOCK_A**. Modify the **slow_blinky_module** design, and implement the LED blinking at a frequency of around 0.75 Hz on the Basys 3 development board.

UNDERSTANDING | TASK 6

In the **slow_blinky_module**, insert the following line of code in the always block:

```
always @ (posedge CLOCK) begin
    COUNT <= COUNT + 1;
    LED <= ( COUNT == 4'b0000 ) ? ~LED : LED ;
end
```

Do additional modifications in the **slow_blinky_module** and the testbench code, and simulate the **slow_blinky_module** design.

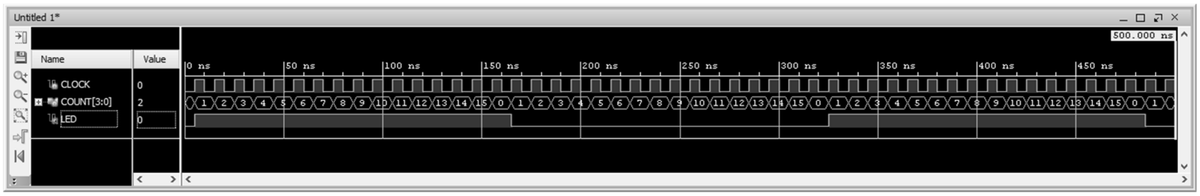
[NOTE] Hints on the modifications related to slow_blinky_module

- `LED` is an output signal of the design module

- A signal declared as **`reg`** can be reused within the design module. An example is an output signal that needs to be reused as input within the design module

- A design module signal declared as **`reg`** can be given an initial value, especially if toggling is involved

If the codes have been correctly written, a simulation waveform similar to what is shown below can be obtained:



Observe the waveform that you have obtained in your simulation window, and calculate the frequency of the signal **LED**:

f = ___________ MHz

From your understanding of the multiplexer, state what the following line of code means:

```
LED <= ( COUNT == 4'b0000 ) ? ~LED : LED ;
```

____

____

UNDERSTANDING | TASK 7 * ***Warning:*** *Do not simulate the code. Why?* *

Using the knowledge and coding obtained from **UNDERSTANDING | TASK 6**, calculate the number of bits for **COUNT** which will allow **LED** to have a frequency of around 0.75 Hz.

Number of bits required for **COUNT**: ___________

Name that waveform with the frequency of 0.75 Hz as **SLOWCLOCK_B**. Modify the **slow_blinky_module** design, and implement the LED blinking at a frequency of around 0.75 Hz on the Basys 3 development board.

Finally, modify and implement the Verilog code such that there are two blinking speeds for the LED based on the state of a switch:

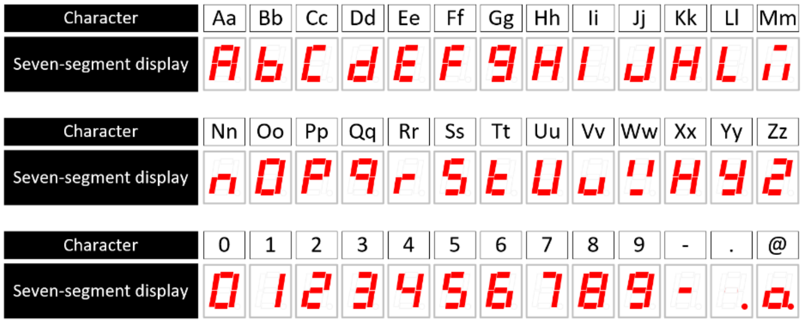
- If the switch is in the OFF position, the LED should blink at a frequency of around 0.75 Hz
- If the switch is in the ON position, the LED should blink at a frequency of around 1.50 Hz

One way to accomplish this task is to use multiplexers in Verilog, as learnt earlier in this lab manual. It may also be implemented using other methods that involve conditional statements.

GRADED POST-LAB ASSIGNMENT

SUB-TASK A [STARTING SEQUENCE]

For this post-lab graded assignment, a simplified system to fill or empty a storage container is required. In your program, all characters appearing on the seven-segment displays must follow *Table A*, as shown below:



Character	Aa	Bb	Cc	Dd	Ee	Ff	Gg	Hh	Ii	Jj	Kk	Ll	Mm

Character	Nn	Oo	Pp	Qq	Rr	Ss	Tt	Uu	Vv	Ww	Xx	Yy	Zz

Character	0	1	2	3	4	5	6	7	8	9	-	.	@

Seven-segment display
Seven-segment display
Seven-segment display

Table A

Throughout your program, the following seven-segment displays must be activated depending on the **1st (First)** rightmost numerical value of your student matriculation number, as indicated in *Table B*:

Seven-segment displays that are required to be always ON or OFF				
1st (First) rightmost numerical value of your student matriculation number	AN3 Display	AN2 Display	AN1 Display	AN0 Display
0	ON	ON	ON	ON
1	ON	ON	ON	OFF
2	ON	ON	OFF	ON
3	ON	ON	OFF	OFF
4	ON	OFF	ON	ON
5	ON	OFF	ON	OFF
6	ON	OFF	OFF	ON
7	ON	OFF	OFF	OFF
8	OFF	ON	ON	ON
9	OFF	ON	ON	OFF

Table B

At the start of your program, all switches must be in the OFF position, and all LEDs are OFF. When your program starts, there is a sequence of character that needs to be seen. Depending on your **3rd (Third)** rightmost numerical value of your student matriculation number, it is required that the character shown on the activated seven-segment displays follows the sequence indicated in *Table C* below, with each update in character occurring at the indicated frequency.

3rd (Third) rightmost numerical value of your student matriculation number	Sequence of characters that need to be seen and repeated non-stop, until SW15 is set to ON	Frequency of character update (See Note 1 and Note 2)
0	0 ▸ 1 ▸ 2 ▸ 3 ▸ 4 ▸ 5 ▸ 6 ▸ Repeat again from 0	6.00 Hz
1	1 ▸ 2 ▸ 3 ▸ 4 ▸ 5 ▸ Repeat again from 1	1.50 Hz
2	2 ▸ 3 ▸ 4 ▸ 5 ▸ 6 ▸ 7 ▸ 8 ▸ Repeat again from 2	6.00 Hz
3	3 ▸ 4 ▸ 5 ▸ 6 ▸ 7 ▸ 8 ▸ Repeat again from 3	6.00 Hz
4	4 ▸ 5 ▸ 6 ▸ 7 ▸ Repeat again from 4	1.50 Hz
5	5 ▸ 4 ▸ 3 ▸ 2 ▸ 1 ▸ 0 ▸ Repeat again from 5	6.00 Hz
6	6 ▸ 5 ▸ 4 ▸ 3 ▸ 2 ▸ 1 ▸ 0 ▸ Repeat again from 6	6.00 Hz
7	7 ▸ 6 ▸ 5 ▸ 4 ▸ 3 ▸ 2 ▸ Repeat again from 7	1.50 Hz
8	8 ▸ 7 ▸ 6 ▸ 5 ▸ Repeat again from 8	1.50 Hz
9	9 ▸ 8 ▸ 7 ▸ 6 ▸ 5 ▸ 4 ▸ 3 ▸ 2 ▸ Repeat again from 9	6.00 Hz

Table C

Note 1: *An error ± 0.25 Hz for the frequency is acceptable*
Note 2: *Use a stopwatch to test if you are using the correct frequency*

SUB-TASK B [SWITCH ACTIVATED FILLING AND EMPTYING]

When SW15 is set to ON, the system changes to a SAFE mode. Assume that the user will not turn OFF SW15 after it has been turned ON. In other words, SW15 will remain ON throughout the remaining parts of the program.

In SAFE mode, the seven segment displays stop showing the starting sequence of characters from sub-task A, and instead show a certain character based on the current amount of LEDs (From LD0 to LD15) being on, as indicated in *Table D* below:

Number of LEDs that are currently ON	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Character to show on the activated seven segment displays	E	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	F**.**

Table D

Anytime during SAFE mode, the user can make SW0 ON or OFF

When SW0 is ON, the amount of LEDs that are turned ON is increased by one for each clock cycle of frequency f_{ON}, in the direction from LD0 to LD15. The 16th LED (LD15) that is ON is required to blink at a frequency f_{BLINK}. When LD0 to LD14 are ON, and LD15 is blinking, nothing else happens to the 16 LEDs unless the user changes the state of SW0 to OFF. It is not required to make the seven-segment displays blink when 16 LEDs are ON (only LED LD15 must blink).

When SW0 is OFF, the amount of LEDs that are ON is decreased by one for each clock cycle of frequency f_{OFF}, in the direction from LD15 to LD0. When all 16 LEDs are OFF, nothing happens to the 16 LEDs, unless the user changes the state of SW0 to ON.

The frequencies used during SAFE mode are dependent on the **1st (First)** rightmost numerical value of your student matriculation number, as tabulated in *Table E* below:

1st (First) rightmost numerical value of your student matriculation number	Frequency, f_{ON} (See Note 3 and Note 4)	Frequency, f_{OFF} (See Note 3 and Note 4)	Frequency, f_{BLINK} (See Note 3 and Note 4)
0	0.75 Hz	3.00 Hz	6.00 Hz
1	3.00 Hz	0.75 Hz	6.00 Hz
2	0.75 Hz	3.00 Hz	6.00 Hz
3	3.00 Hz	0.75 Hz	6.00 Hz
4	0.75 Hz	3.00 Hz	6.00 Hz
5	6.00 Hz	0.75 Hz	3.00 Hz
6	1.50 Hz	6.00 Hz	3.00 Hz
7	6.00 Hz	0.75 Hz	3.00 Hz
8	1.50 Hz	6.00 Hz	3.00 Hz
9	6.00 Hz	0.75 Hz	3.00 Hz

Table E

Note 1: *An error ± 0.25 Hz for the frequency is acceptable*
Note 2: *Use a stopwatch to test if you are using the correct frequency*

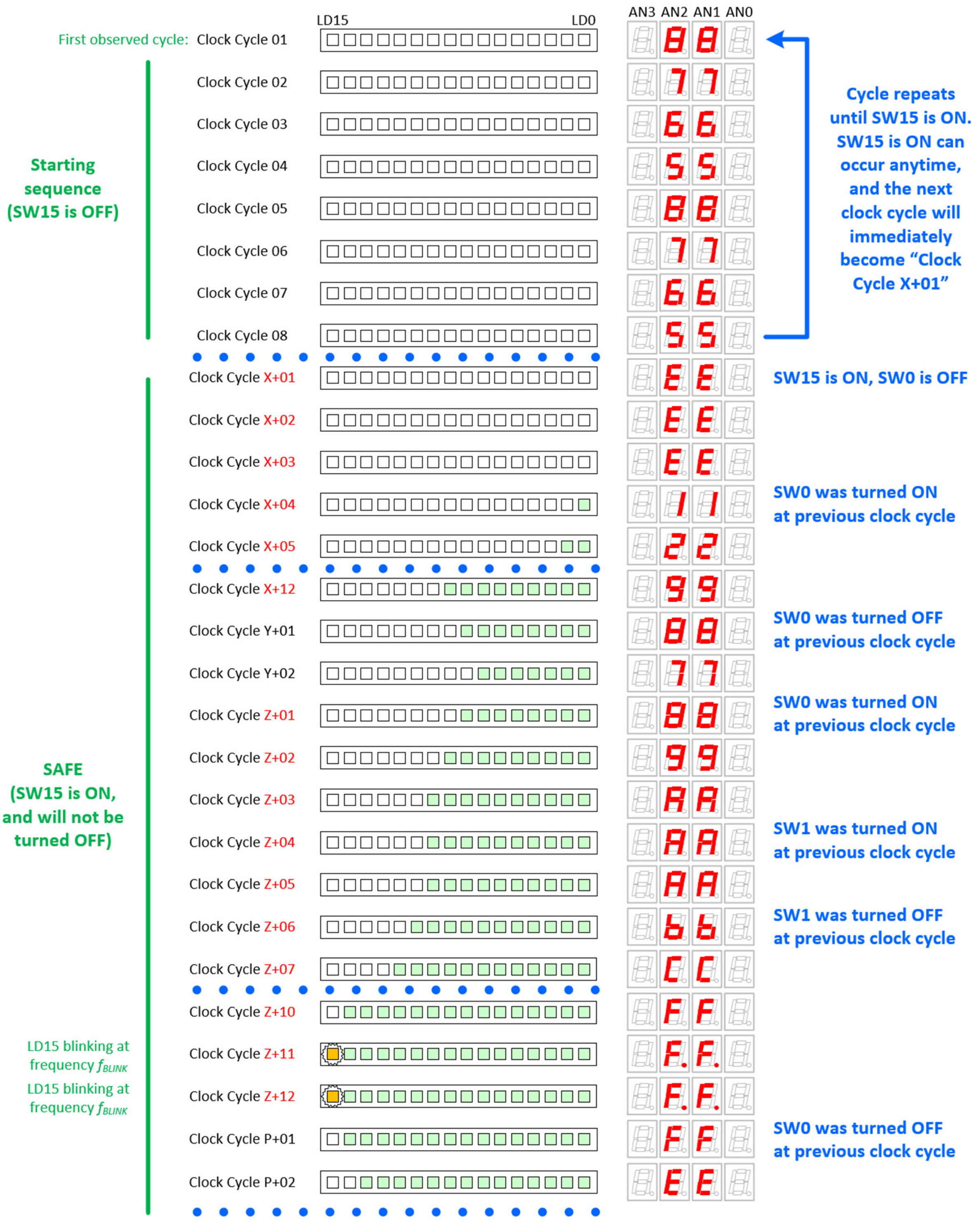
Furthermore, anytime during SAFE mode, when the user makes SW1 to be ON, no additional LEDs can turn ON or OFF, regardless of whether SW0 is ON or OFF. If SW1 is turned OFF, SAFE mode behaves as per normal.

EXAMPLE BASED ON YOUR STUDENT MATRICULATION CARD NUMBER

If a student has matriculation number “A0002809N” then:

- **1st right-most digit is 9:** Only the middle 2 seven-segment displays out of the 4 seven-segment displays will be ON throughout the running of the program
Frequency, f_{ON} = 6.00 Hz (One LEDs turn ON every 0.167 second)
Frequency, f_{OFF} = 0.75 Hz (One LEDs turn OFF every 1.333 seconds)
Frequency, f_{BLINK} = 3.00 Hz (LD15 will turn ON or OFF every 0.333 second)

- **3rd right-most digit is 8:** Starting sequence is 8 ▸ 7 ▸ 6 ▸ 5 ▸ 8 ▸ 7 ▸ 6 ▸ 5 ▸ 8 ▸ 7 ▸ 6 ▸ 5 ▸ 8 ▸ 7 ▸ 6 ▸ 5 ▸ 8 …
Character changes at a frequency of 1.50 Hz (Character changes every 0.667 seconds)



Cycle repeats until SW15 is ON. SW15 is ON can occur anytime, and the next clock cycle will immediately become “Clock Cycle X+01”

SW15 is ON, SW0 is OFF

SW0 was turned ON at previous clock cycle

SW0 was turned OFF at previous clock cycle

SW0 was turned ON at previous clock cycle

SW1 was turned ON at previous clock cycle

SW1 was turned OFF at previous clock cycle

SW0 was turned OFF at previous clock cycle

HINTS

- Complete and understand **UNDERSTANDING | TASK 7** clearly before working on this assignment
- Excellent mastery of structural modelling and instantiation is recommended, to properly send signals from one instance to other instances
- Creating different modules for different functions, and then simulating them before instantiating them in a main module will make complex systems easier to debug
- Use multiplexers to select between different clock signals coming from multiple clock dividers
- In Lab 1, single-bit was used for each signal, and information regarding the seven-segment displays were given as well. In Lab 2, multi-bits signals were taught. From this stage onwards, use one set of multi-bits signal to connect to the seven-segments and decimal point, and another set of multi-bit signals to connect to the four seven-segment displays
- A counter whose value can change at each clock cycle can be considered. For example, that counter can count 0, 1, 2, 3, 0, 1, 2, 3 etc ... Following that, if-else statements, multiplexers, or case statement that indicates what to do at each specific counter value can be created

  Case statements are recommended here, and the case statement template is given below.

  ```
     case (expression)
        case_item: statement or statement_group
        case_item: statement or statement_group
        default: statement or statement_group
     endcase
  ```

- An example in using case is given below. Note that case statements (if-else statements also) must lie within an always block:

  ```
  always @ (posedge clk_25_mhz)
  begin
     case (counter_value)
        2'd0:
           begin
                 my_value_a <= 20;
                 my_value_b <= 40;
           end
        2'd1:
           begin
                 my_value_a <= 100;
                 my_value_b <= 200;
           end
        2'd2: my_value_c <= 5;
        default: my_value_d <= 9;
     endcase
  end
  ```

- **Be careful of parallel execution of the always blocks. Multi-driven nets indicate that there are conflicting values being given to the same signal from different always blocks.** For example, one cannot tell a signal to increase at a time instant t, and at that same time instant t, telling it to decrease

- Refer to **http://tiny.cc/ee2026wiki** for more details on commonly encountered errors.

LUMINUS SUBMISSION INSTRUCTIONS

- Complete as much required functionalities **as possible within the given deadline**, and ensure that your bitstream has been successfully generated and tested on your Basys 3 development board **BEFORE** archiving your Vivado workspace for LumiNUS upload. No working bitstream is equivalent to no marks (It is best to have some working functionalities / requirements, instead of not having any bitstream at all while trying all requirements)

- It is compulsory to archive your project in a compressed form without any simulation waveforms. In the uploaded archive, the codes (.v files) are important, not the waveforms (.wdb files). **The archive size should not exceed 2 MB in size for lab 3.** Follow the instructions given in the pdf: “Archive Project in Vivado 2018.02”

- **After** following the instructions in “**Archive Project in Vivado 2018.02**”, rename your project archive as indicated in the appendix of this lab manual.

- Upload to LumiNUS EE2026 -> Files -> Lab and Project - Materials and Submissions -> Lab 3 Submission

- Download your LumiNUS archive after uploading. **Unzip it / Extract all, and check if you can run your bitstream correctly**. No project files and no working bitstream is equivalent to losing all marks

- The LumiNUS upload must be completed by **Wednesday 23rd September 2020, 12:00 P.M. (Noon)**. Do not plan to upload during the grace period of 2 hours

- A penalty of 25% applies for late submissions of up to 1 week.

- The late submission folder closes 1 week after the original deadline. Late submissions are not accepted if you have already submitted on time, or if grading has already started on an earlier submitted file. The late submission folder will be located at: LumiNUS EE2026 -> Files -> Lab and Project - Materials and Submissions -> Lab 3 Submission (Late Submission)

Plagiarism is penalised with a 100% penalty for all SOURCES and RECIPIENTS

All past and future submissions, and marks, will be reviewed in greater detail, for any person found to have plagiarised

ALL THE SUBMISSION INSTRUCTIONS LISTED ABOVE WILL AFFECT YOUR GRADES!

GRADING PROCESS

- During subsequent lab sessions, our graders will be providing you updates on the grading of your submission

- Submissions not following all the ***LUMINUS SUBMISSION INSTRUCTIONS*** (listed above) will not be graded immediately, and will instead be reviewed towards the end of the semester. **You will not be able to see your results during the labs in such situations**

APPENDIX (Renaming submissions just before LumiNUS upload):

It is **compulsory to rename your project archive and report**, just before LumiNUS upload, as indicated in the table below. Copy your respective "Archive Naming" for the archive, and then replace the "xxxxxxxxx" with your student ID number. Do not change any other part of the naming, except the "xxxxxxxxx".

Submission example for project archive: L3_Fri_AM_Alan Turing_Archive_A0131086Z

Name	Archive Naming
A AKIL AHAMED	L3_Fri_AM_A AKIL AHAMED_Archive_XXXXXXXXX
Abdul Hadi Bin Abdul Samad	L3_Fri_AM_Abdul Hadi Bin Abdul_Archive_XXXXXXXXX
Adabelle Lim Ru Leng	L3_Fri_AM_Adabelle Lim Ru Leng_Archive_XXXXXXXXX
Alfred Wrong Jia Qing	L3_Fri_AM_Alfred Wrong Jia Qin_Archive_XXXXXXXXX
Alvin Goh Jia Hao	L3_Fri_AM_Alvin Goh Jia Hao_Archive_XXXXXXXXX
Alvinci Merquita	L3_Wed_AM_Alvinci Merquita_Archive_XXXXXXXXX
ANG KENG SIANG	L3_Wed_AM_ANG KENG SIANG_Archive_XXXXXXXXX
Aryl Ng Shen Le	L3_Wed_AM_Aryl Ng Shen Le_Archive_XXXXXXXXX
Au Yuan Xian	L3_Fri_AM_Au Yuan Xian_Archive_XXXXXXXXX
Bai Xiaoru	L3_Fri_AM_Bai Xiaoru_Archive_XXXXXXXXX
Bryan Yu Cheng You	L3_Fri_AM_Bryan Yu Cheng You_Archive_XXXXXXXXX
Chai Wei Lynthia	L3_Fri_AM_Chai Wei Lynthia_Archive_XXXXXXXXX
Cheang Zhi Yi Jordan	L3_Fri_AM_Cheang Zhi Yi Jordan_Archive_XXXXXXXXX
Chee Poh Hock	L3_Wed_AM_Chee Poh Hock_Archive_XXXXXXXXX
Cheng Wei Qiao	L3_Fri_AM_Cheng Wei Qiao_Archive_XXXXXXXXX
Cheung Po Rui Bryan	L3_Fri_AM_Cheung Po Rui Bryan_Archive_XXXXXXXXX
CHONG LEE TENG VALENCIA	L3_Wed_AM_CHONG LEE TENG VALEN_Archive_XXXXXXXXX
Davian Chan Sze Peng	L3_Fri_AM_Davian Chan Sze Peng_Archive_XXXXXXXXX
David Michael Woodside	L3_Fri_AM_David Michael Woodsi_Archive_XXXXXXXXX
ELJER CHUA	L3_Fri_AM_ELJER CHUA_Archive_XXXXXXXXX
FANG XINJIA	L3_Fri_AM_FANG XINJIA_Archive_XXXXXXXXX
Fidel Tan Yan Sheng	L3_Fri_AM_Fidel Tan Yan Sheng_Archive_XXXXXXXXX
Foo Fang Kiang	L3_Fri_AM_Foo Fang Kiang_Archive_XXXXXXXXX
Gao Zhixuan	L3_Fri_AM_Gao Zhixuan_Archive_XXXXXXXXX
Giam Xiong Yao	L3_Fri_AM_Giam Xiong Yao_Archive_XXXXXXXXX
Gillian Ho Xin Ying	L3_Fri_AM_Gillian Ho Xin Ying_Archive_XXXXXXXXX
Goh Jia Hong Edwin	L3_Wed_AM_Goh Jia Hong Edwin_Archive_XXXXXXXXX
Guinne Teresa Sng Yu Lin	L3_Fri_AM_Guinne Teresa Sng Yu_Archive_XXXXXXXXX
Hariharan Hadrian S/O Subramaniam	L3_Fri_AM_Hariharan Hadrian S_Archive_XXXXXXXXX
HO MING JUN	L3_Wed_AM_HO MING JUN_Archive_XXXXXXXXX
Ho Yi Shu Keon	L3_Wed_AM_Ho Yi Shu Keon_Archive_XXXXXXXXX
Hou Yinjiayi	L3_Fri_AM_Hou Yinjiayi_Archive_XXXXXXXXX
Ian Isaiah Tan Jun Wei	L3_Fri_AM_Ian Isaiah Tan Jun W_Archive_XXXXXXXXX
Jacob Zhang Zhiqiang	L3_Wed_AM_Jacob Zhang Zhiqiang_Archive_XXXXXXXXX
JEROME TEO SZE YONG	L3_Wed_AM_JEROME TEO SZE YONG_Archive_XXXXXXXXX
Jonathan Ang Xu Wen	L3_Wed_AM_Jonathan Ang Xu Wen_Archive_XXXXXXXXX
JONATHAN KHOO TENG YANG	L3_Fri_AM_JONATHAN KHOO TENG Y_Archive_XXXXXXXXX
Kabeta Takuma	L3_Wed_AM_Kabeta Takuma_Archive_XXXXXXXXX
Khoo Wu Jian Samuel	L3_Wed_AM_Khoo Wu Jian Samuel_Archive_XXXXXXXXX
KIM JOOHWAN	L3_Fri_AM_KIM JOOHWAN_Archive_XXXXXXXXX
Lau Wai Kit	L3_Wed_AM_Lau Wai Kit_Archive_XXXXXXXXX
LEE KE HUI	L3_Wed_AM_LEE KE HUI_Archive_XXXXXXXXX
Lee Shao Yu	L3_Wed_AM_Lee Shao Yu_Archive_XXXXXXXXX
Lek Ju Ying	L3_Wed_AM_Lek Ju Ying_Archive_XXXXXXXXX
Leong Ka Weng, Rachelle	L3_Fri_AM_Leong Ka Weng Rache_Archive_XXXXXXXXX
LEW POH CHEN, DOUGLAS	L3_Fri_AM_LEW POH CHEN DOUGLA_Archive_XXXXXXXXX
Long Deng Jie	L3_Wed_AM_Long Deng Jie_Archive_XXXXXXXXX
Markus Lim Yi Qin	L3_Wed_AM_Markus Lim Yi Qin_Archive_XXXXXXXXX
Mohamad Adam Bin Mohamad Yazid	L3_Wed_AM_Mohamad Adam Bin Moh_Archive_XXXXXXXXX
Muhammad Irfan Bin Zakaria	L3_Wed_AM_Muhammad Irfan Bin Z_Archive_XXXXXXXXX
Myat Thwe Naing	L3_Wed_AM_Myat Thwe Naing_Archive_XXXXXXXXX
Ng Etek	L3_Wed_AM_Ng Etek_Archive_XXXXXXXXX
Noorhakim Bin Jasman	L3_Wed_AM_Noorhakim Bin Jasman_Archive_XXXXXXXXX
NUR SYADIYAH BTE LUTFI	L3_Fri_AM_NUR SYADIYAH BTE LUT_Archive_XXXXXXXXX
ONG WEI SHENG	L3_Fri_AM_ONG WEI SHENG_Archive_XXXXXXXXX
PANG JUN WEN, ADRIC	L3_Fri_AM_PANG JUN WEN ADRIC_Archive_XXXXXXXXX
PUN ZE YONG	L3_Wed_AM_PUN ZE YONG_Archive_XXXXXXXXX
Qiang Zhuang	L3_Wed_AM_Qiang Zhuang_Archive_XXXXXXXXX
QIU YI WEN	L3_Fri_AM_QIU YI WEN_Archive_XXXXXXXXX
R M RAAJAMANI	L3_Fri_AM_R M RAAJAMANI_Archive_XXXXXXXXX
RIZAVUR RAHMAN FASLUR RAHMAN	L3_Fri_AM_RIZAVUR RAHMAN FASLU_Archive_XXXXXXXXX
Ryan Tan Jun Hao	L3_Wed_AM_Ryan Tan Jun Hao_Archive_XXXXXXXXX
Saw Wee Kiat	L3_Wed_AM_Saw Wee Kiat_Archive_XXXXXXXXX
SIM BOWEN	L3_Fri_AM_SIM BOWEN_Archive_XXXXXXXXX
TAM LI NA	L3_Fri_AM_TAM LI NA_Archive_XXXXXXXXX
Tan Javen	L3_Wed_AM_Tan Javen_Archive_XXXXXXXXX
Tan Kai Hao Andrew	L3_Wed_AM_Tan Kai Hao Andrew_Archive_XXXXXXXXX
Tan Suet Ying	L3_Fri_AM_Tan Suet Ying_Archive_XXXXXXXXX
Tan Yeung Ming Sean Eugene	L3_Wed_AM_Tan Yeung Ming Sean _Archive_XXXXXXXXX
Teoh Yi Zheng	L3_Fri_AM_Teoh Yi Zheng_Archive_XXXXXXXXX
Tey Zi Le	L3_Wed_AM_Tey Zi Le_Archive_XXXXXXXXX
Thet Ke Min, Sonia	L3_Wed_AM_Thet Ke Min Sonia_Archive_XXXXXXXXX
Trina Wern Qin Rong	L3_Fri_AM_Trina Wern Qin Rong_Archive_XXXXXXXXX
Venessa Chee Li Lin	L3_Fri_AM_Venessa Chee Li Lin_Archive_XXXXXXXXX
WANG HUA CHEN	L3_Fri_AM_WANG HUA CHEN_Archive_XXXXXXXXX
Wee Cheng Yuan Andrew	L3_Wed_AM_Wee Cheng Yuan Andre_Archive_XXXXXXXXX
Wee Xin Ze	L3_Wed_AM_Wee Xin Ze_Archive_XXXXXXXXX
Yeo Zhong Kang Dennis	L3_Wed_AM_Yeo Zhong Kang Denni_Archive_XXXXXXXXX
ZHONG SHUHAO	L3_Fri_AM_ZHONG SHUHAO_Archive_XXXXXXXXX