

EE2026: DIGITAL DESIGN

Academic Year 2021-2022, Semester 2

LAB 3: Sequential Circuits in Verilog

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:

- 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:

- 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.

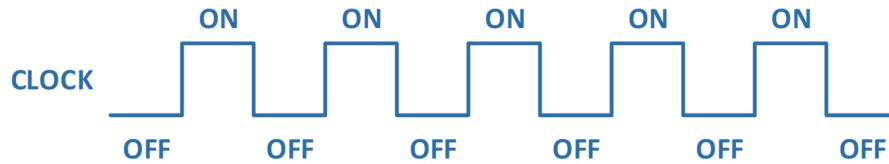
GRADED ASSIGNMENT [LUMINUS SUBMISSION: WEDNESDAY 23rd FEBRUARY 2022, NOON]:

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

THE BLINKING LED

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:

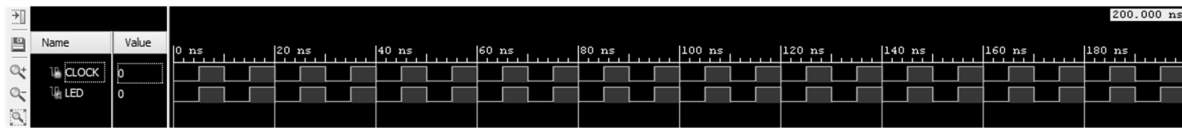
Verilog code for blinky, using the dataflow method

```
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 {sv[0]}]
13 set_property IOSTANDARD LVCMOS33 [get_ports {sv[0]}]
14 set_property PACKAGE_PIN V16 [get_ports {sv[1]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {sv[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?

COUNTER FOR A SLOWER CLOCK

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 to have a slower clock. Let us introduce a temporary variable **COUNT** that is incremented by 1 at every rising edge (transition from low to high; also called a positive edge) of the **CLOCK** signal, as shown in **Figure 3.4**.

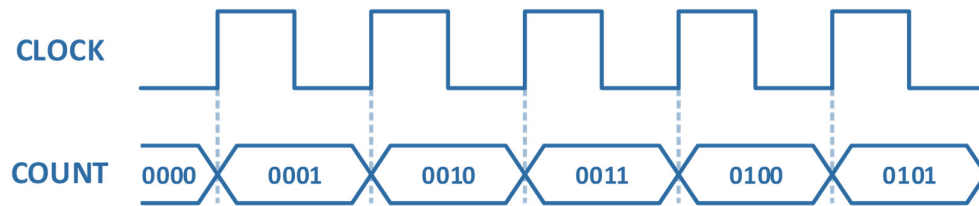


Figure 3.4: Increasing **COUNT** at each rising edge of **CLOCK**

Verilog code for an incrementing counter of 4-bit, using behavioural modelling

```
module slow_blinky_module (input CLOCK);  
    reg [3:0] COUNT = 4'b0000;  
  
    always @ (posedge CLOCK) begin  
        COUNT <= COUNT + 1;  
    end  
  
endmodule
```

Now, use a conditional statement that changes the state of an output signal, called **SLOW_CLOCK**, whenever the **COUNT** reaches zero. The initial state of **SLOW_CLOCK** can be zero or one. Here, since the 4-bits **COUNT** has 16 possible states, it would mean that **SLOW_CLOCK** changes state after 16 clock cycles of **CLOCK**. In other words, **SLOW_CLOCK** is 1/16th the clock speed of **CLOCK**.

Partial Verilog code for toggling **SLOW_CLOCK** signal, using behavioural modelling

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

UNDERSTANDING | TASK 4

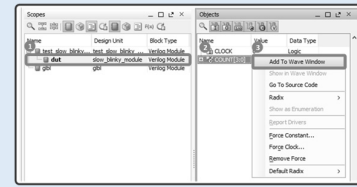
Simulate the Verilog codes, so that a waveform that toggles every 16th clock cycle can be observed

Finally, explore what happens if **SLOW_CLOCK** is forcefully toggled before the overflow to 0 occurs. To do so, modify the conditional statement to allow **SLOW_CLOCK** to toggle an earlier clock cycle instead of waiting for 16 clock cycles. Simulate the Verilog codes again and observe if the waveforms have toggled earlier.

FURTHER ANALYSIS 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**



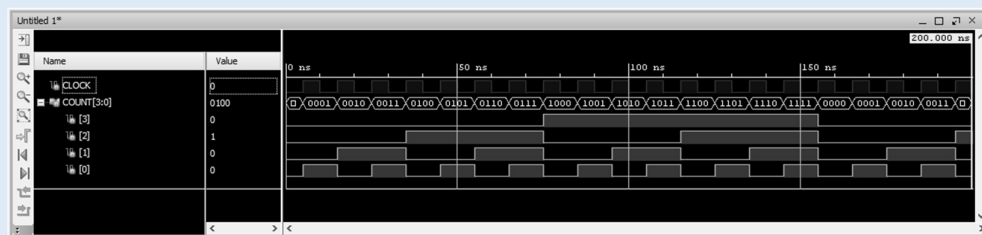
After adding the variable **COUNT** to the wave window, the current simulation needs to be re-run. Follow these steps:

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

This is an important step to remember if you want to simulate for more than the default 1000 ns of simulation!!!



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:



UNDERSTANDING | TASK 5

Create 4 different clock signals with approximately (A precise value is not required) the following clock frequencies:

- clock_A : 1000 Hz
- clock_B : 100 Hz
- clock_C : 10 Hz
- clock_D : 1 Hz

Simulate clock_A to confirm that you have a clock of approximately 1000 Hz.

Simulate clock_B to confirm that you have a clock of approximately 100 Hz.

Self-thinking: Why do you think it is not always feasible to simulate the very slow clocks, such as 10 Hz and 1 Hz?

THE NOTICEABLE BLINKING LED (POST-LAB NON-GRADED – NO SUBMISSION REQUIRED):

It is strongly encouraged to complete this practice task before working on the graded post-lab assignment!

Assume LD0 on the Basys 3 development is the LED to blink.

Consider the three switches SW0, SW1 and SW2, and their behaviours:

- When these four switches are OFF, LD0 must be OFF
- When SW0 is ON, LD0 must blink at 100 Hz
- When SW1 is ON, LD0 must blink at 10 Hz
- When SW2 is ON, LD0 must blink at 1 Hz

Implement the above requirement on the Basys 3 development board. You can use the multiplexers, if-else statements, or case statements to write the conditional codes. It is also up to you to decide which switch has higher importance.

Note: In the EE2026 labs, a blinking frequency of 1 Hz means that the LED should be ON for 0.5 seconds, and OFF for around 0.5 seconds, for each blinking cycle

GRADED POST-LAB ASSIGNMENT

Complete as much as possible, in **one working bitstream for this whole assignment**. It is much better to have a working program with some completed functionalities, instead of submitting a program without a working bitstream (No marks given).

IMPORTANT CHARACTERS

In this assignment, these are the important characters to note from your student matriculation number:

- The 1st rightmost numerical value of your student matriculation number (Subtask A)
- The 2nd rightmost numerical value of your student matriculation number (Subtask B)
- The five rightmost numerical values of your student matriculation number (Subtask D)
- The rightmost alphabet of your student matriculation number (Subtask D)

INITIALISATION

When the program starts, switches SW0 to SW15 must be in the OFF position. The seven segment displays are all be OFF.

SUBTASK A

At the start of the program, a set of LEDs starting from LD0 to MAX_LED (Including) are required to turn ON after every TIME_COUNT. Other LEDs, from MAX_LED (Excluding) to LD15, are OFF. MAX_LED is dependent on the **1st rightmost numerical value of your student matriculation number**, as indicated in the table below:

1 st Rightmost Numerical Value	MAX_LED	TIME_COUNT
0	LD14	0.20 seconds (Error of ± 0.05 seconds accepted)
1	LD13	0.36 seconds (Error of ± 0.05 seconds accepted)
2	LD12	0.54 seconds (Error of ± 0.05 seconds accepted)
3	LD11	0.75 seconds (Error of ± 0.05 seconds accepted)
4	LD10	1.00 seconds (Error of ± 0.05 seconds accepted)
5	LD9	1.20 seconds (Error of ± 0.05 seconds accepted)
6	LD8	1.11 seconds (Error of ± 0.10 seconds accepted)
7	LD7	1.00 seconds (Error of ± 0.10 seconds accepted)
8	LD6	0.86 seconds (Error of ± 0.10 seconds accepted)
9	LD5	0.67 seconds (Error of ± 0.10 seconds accepted)

When all the LEDs from LD0 to MAX_LED are ON (and not before), the user can turn ON certain switches. These switches should make certain LEDs blink, as indicated in the truth table below (X is the don't care condition):

INPUTS			OUTPUTS			
SW2	SW1	SW0	MAX_LED to LD2	LD2	LD1	LD0
0	0	0	ON	ON	ON	ON
0	0	1	ON	ON	ON	Blink at 1 Hz
0	1	X	ON	ON	Blink at 10 Hz	ON
1	X	X	ON	Blink at 100 Hz	ON	ON

(Error of $\pm 10\%$ accepted for all blinking frequencies)

SUBTASK B

When MAX_LED is ON at the end of SUBTASK A, the user will then be required to press certain pushbuttons based on the characters that appear on the seven segment displays. Recall that the seven segment displays do not show anything before MAX_LED is ON.

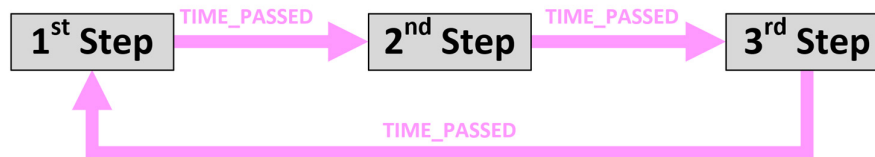
Characters to Appear on Seven Segment Displays in this Order							
2 nd Rightmost Numerical Value		1 st Step	2 nd Step	3 rd Step	4 th Step	5 th Step	6 th Step
	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						
	9						
	UP Pushbutton (BTNU) must be pressed before next step						
	RIGHT Pushbutton (BTNR) must be pressed before next step						
	DOWN Pushbutton (BTND) must be pressed before next step						
	LEFT Pushbutton (BTNL) must be pressed before next step						
	CENTER Pushbutton (BTNC) must be pressed before next step						

After the user has pressed the required pushbuttons in the correct order, the system goes into an “UNLOCKED” mode, which is indicated by LD15 being ON. When LD15 is ON, the seven segment displays must show the 1st step again before proceeding to SUBTASK C

SUBTASK C

While in “UNLOCKED” mode, it is required to cycle sequentially through the first three steps at a speed determined by switches SW2, SW1, and SW0. It takes TIME_PASSED amount of time to move to the next step. After the 3rd Step has passed, it is required to restart from the 1st Step, and the process of cycling through the three steps continues infinitely. The value for TIME_PASSED, as well as an illustration of the sequential cycle, is shown below:

SW2	SW1	SW0	TIME_PASSED
0	0	0	<i>Infinity</i> (Character paused / No change in characters)
0	0	1	0.500 seconds (Error of ± 0.100 seconds accepted)
0	1	X	0.050 seconds (Error of ± 0.010 seconds accepted)
1	X	X	0.005 seconds (Error of ± 0.001 seconds accepted)



Note: The blinking LEDs and other LEDs from SUBTASK A must still work while doing SUBTASK C

SUBTASK D

Anytime throughout the whole program (SUBTASK A, B and C), if the user turns ON SW15 for 3 seconds (Error of ± 0.3 seconds accepted), then the following two things must happen:

- All LEDs must turn OFF and stop blinking, except for the LEDs LD \mathbf{X} that must be ON. \mathbf{X} represents each one of the five rightmost numerical values of your student matriculation number. Example: If the five values are 59089, then LD0, LD5, LD8 and LD9 must be ON.
- The seven segment displays must show the last character of your student matriculation on all four anodes (AN3, AN2, AN1, AN0), with the characters as indicated below:

Rightmost Alphabet	A	B	E	H	J	L	M	N	R	U	W	X	Y
Required 7-Segments Character	A	b	E	H	J	L	n	n	r	U	!	H	4

If the user turns SW15 OFF again, immediately (Unlike the 3 seconds waiting time when SW15 is turned ON) the LEDs LD0 to LD15, as well as the seven segment displays should work as described in all the previous SUBTASKS.

SUGGESTIONS

- 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.

```
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
```

- Case statements and if-else statements must be inside an always block.
- Conditional statements using multiplexers can be within or outside always block.
- **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

- Ensure that your bitstream has been successfully generated and tested on your Basys 3 development board **BEFORE** archiving your Vivado workspace for LumiNUS upload
- It is compulsory to archive your project in a compressed form without any saved simulation waveforms. In the uploaded archive, the codes (.v files) are important, not the waveforms (.wdb files). **The archive size should not exceed 4 MB in size for any lab assignments.** 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. **Click and drag the single folder within that archive to desktop, and then open the Vivado project in that extracted folder to see if it can be opened. Check if you can also 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 February 2022, 12:00 P.M. (Noon)**. Avoid uploading 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 and not graded if a submission is found within the on-time folder, 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 (COMPULSORY renaming before just LumiNUS upload):

It is **compulsory to rename your project archive** just before the LumiNUS upload, as listed in the table below.

Do not change any other part of the naming. Simply **copy** the naming from the table below, and **paste** it while renaming your project archive.

Penalties will be incurred if your submission cannot be found according to the exact naming template below.

Name	Archive Naming
Aaron Chan Zhi	L3_Thurs_AM_Aaron Chan Zhi_476_Archive
Ajay Shanker	L3_Thurs_AM_Ajay Shanker_806_Archive
Alphonsus Teow Rui Jie	L3_Thurs_AM_Alphonsus Teow Rui Jie_502_Archive
Alvin Ben Abraham	L3_Thurs_AM_Alvin Ben Abraham_394_Archive
Amadeus Lim Ding Shin	L3_Thurs_AM_Amadeus Lim Ding Shin_412_Archive
Amit Rahman	L3_Thurs_AM_Amit Rahman_599_Archive
Ang Jia Le Marcus	L3_Thurs_AM_Ang Jia Le Marcus_025_Archive
Chan Ee Hong	L3_Thurs_AM_Chan Ee Hong_898_Archive
Chen Zi Han	L3_Thurs_AM_Chen Zi Han_549_Archive
Chien Jing Wei	L3_Thurs_AM_Chien Jing Wei_540_Archive
CHUA WEI XUAN	L3_Thurs_AM_CHUA WEI XUAN_716_Archive
Chua Wen Xin Kyrene	L3_Thurs_AM_Chua Wen Xin Kyrene_431_Archive
Darren Loh Rui Jie	L3_Thurs_AM_Darren Loh Rui Jie_289_Archive
Dennis Wong Guan Ming	L3_Thurs_AM_Dennis Wong Guan Ming_806_Archive
Huang Yu Chiao	L3_Thurs_AM_Huang Yu Chiao_102_Archive
Ivan Theng Wen Rong	L3_Thurs_AM_Ivan Theng Wen Rong_344_Archive
Jia Yixuan	L3_Thurs_AM_Jia Yixuan_150_Archive
Karthikeyan Vigneshram	L3_Thurs_AM_Karthikeyan Vigneshram_697_Archive
Leong Wei Lun, Alfred	L3_Thurs_AM_Leong Wei Lun Alfred_609_Archive
Liu Junhao	L3_Thurs_AM_Liu Junhao_523_Archive
Marvin Pranajaya	L3_Thurs_AM_Marvin Pranajaya_683_Archive
Ng Sihan, Ian	L3_Thurs_AM_Ng Sihan Ian_817_Archive
Ong Zhi Hong	L3_Thurs_AM_Ong Zhi Hong_922_Archive
Raymond Bala	L3_Thurs_AM_Raymond Bala_127_Archive
See Zhuo Rui Jorelle	L3_Thurs_AM_See Zhuo Rui Jorelle_490_Archive
Shanmugam Surya	L3_Thurs_AM_Shanmugam Surya_189_Archive
Shawn Tan Jinhui	L3_Thurs_AM_Shawn Tan Jinhui_247_Archive
Shin Donghun	L3_Thurs_AM_Shin Donghun_808_Archive
Stefan Choo Bin Hao	L3_Thurs_AM_Stefan Choo Bin Hao_098_Archive
Syed Muhamad Amali B Syed A A	L3_Thurs_AM_Syed Muhamad Amali B Syed_373_Archive
Teh Zi-Chun	L3_Thurs_AM_Teh ZiChun_328_Archive
Wen Chen Yu	L3_Thurs_AM_Wen Chen Yu_109_Archive
Wilson Ng Jing An	L3_Thurs_AM_Wilson Ng Jing An_686_Archive
Yong Chin Han	L3_Thurs_AM_Yong Chin Han_814_Archive
Yuan Xinrui	L3_Thurs_AM_Yuan Xinrui_211_Archive