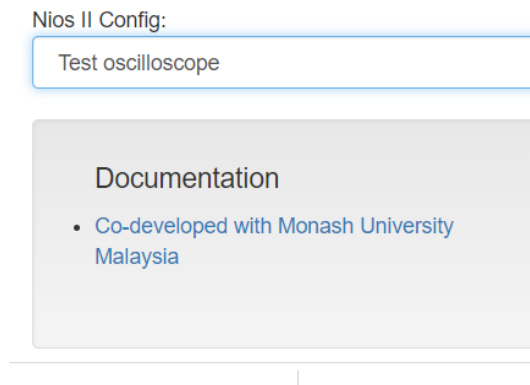
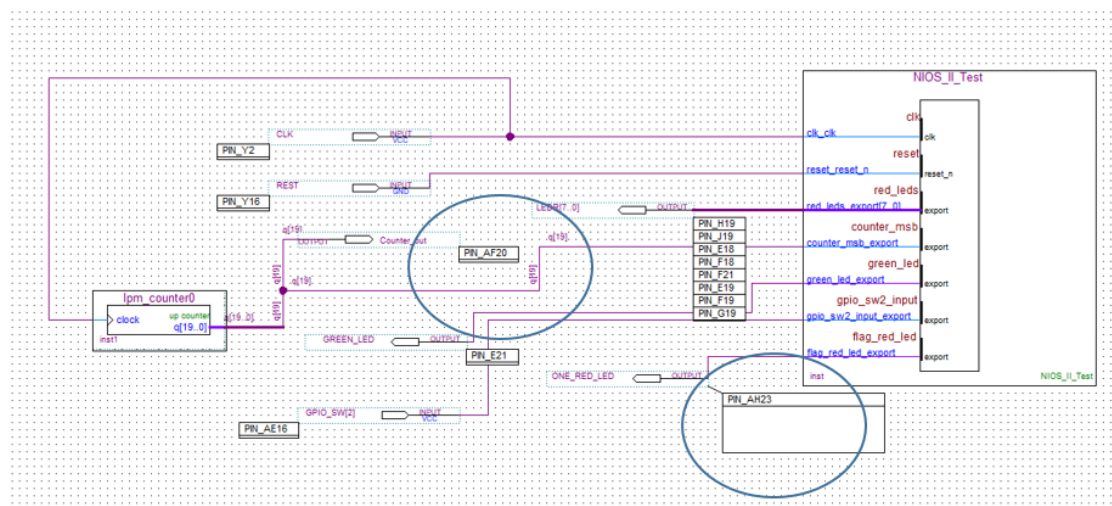


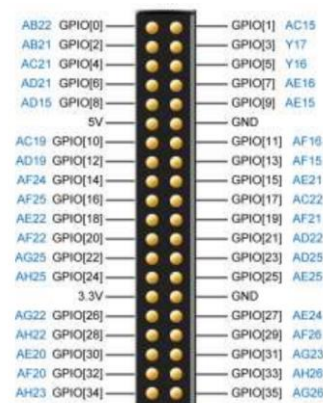
In this exercise we extend our previous Lab 3B work to further understand the latency caused by NIOS instructions through solving following challenging problems. Before getting into the problem let me describe what's been set in the labs land. You will need to choose "Test Oscilloscope" settings for this Lab – 3C









Following is the schematic of the FPGA (Qsys)



IMPORTANT to : I have connected PIN_AF20 and PIN_AH23 of DE2-115 boards to Channel 1 and Channel 2 of the oscilloscope. If you refer to the DE2-115 manual these pins are GPIO pins. Refer screenshot from manual attached below.



Following is the address- memory map of NIOS computer system

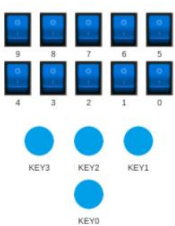
<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> red_leds	clk_0 [clk] [clk]	<div>8-RED LED Base Address (8 bit)</div>  0x3010	0x301f	
<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk1] [clk1]	 0x0000	0x1fff	
<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> counter_msb	clk_0 [clk] [clk]	<div>Counter MSB _GPIO_Base Address (1 bit)</div>  0x3040	0x304f	
<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> green_led	clk_0 [clk] [clk]	<div>Green LED – 1 bit Base Address</div>  0x3030	0x303f	
<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> gpio_sw2_input	clk_0 [clk] [clk]	<div>Labs_Land_Switch2_Base Address (Input)</div>  0x3020	0x302f	
<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> gpio_channel_2oscillosc...	clk_0 [clk] [clk]	<div>GPIO_PIN_Channel_2_Oscilloscope</div>  0x3000	0x300f	

How to Locate Oscilloscope in LabsLand?

Click “Edit” in user interface you will see below


User interface

☐ Standard



Default user interface: camera, switches and buttons

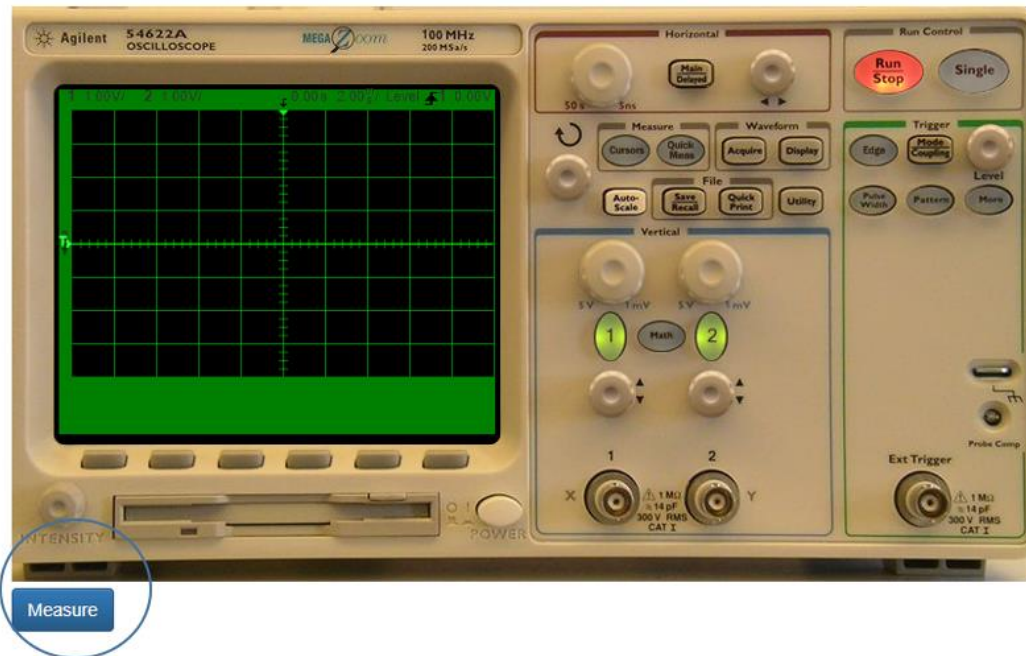
☒ Oscilloscope



Everything in standard plus an oscilloscope

Choose oscilloscope like above and close. Remember you will need to click the “Measure” button to do any oscilloscope measurement

Oscilloscope



Now solve the following challenging exercises

Exercise 1 : (2 Marks)

Continue lighting the 8 LEDs to display the count of every rising edge as you were doing in Lab 3b and in addition now Screen shot LabsLand Oscilloscope Channel _1 output that shows 50 Hz pulse from the counter MSB below, ensure your 8 LEDs are displaying the count ☺

C Code Compiled

```

1 // This code is for the Nios II C IDE. It is a simple program that counts the number of rising edges of a signal.
2 // The signal is connected to the 'fpga_io' block in the Nios II block design.
3 // The code is written in C and is compiled using the Nios II C compiler.
4 // The code is executed on the Nios II processor.
5 // The code is compiled and executed successfully.
6 // The code is compiled and executed successfully.
7 // The code is compiled and executed successfully.
8 // The code is compiled and executed successfully.
9 // The code is compiled and executed successfully.
10 // The code is compiled and executed successfully.
11 // The code is compiled and executed successfully.
12 // The code is compiled and executed successfully.
13 // The code is compiled and executed successfully.
14 // The code is compiled and executed successfully.
15 // The code is compiled and executed successfully.
16 // The code is compiled and executed successfully.
17 // The code is compiled and executed successfully.
18 // The code is compiled and executed successfully.
19 // The code is compiled and executed successfully.
20 // The code is compiled and executed successfully.
21 // The code is compiled and executed successfully.
22 // The code is compiled and executed successfully.
23 // The code is compiled and executed successfully.
24 // The code is compiled and executed successfully.
25 // The code is compiled and executed successfully.
26 // The code is compiled and executed successfully.
27 // The code is compiled and executed successfully.
28 // The code is compiled and executed successfully.
29 // The code is compiled and executed successfully.
30 // The code is compiled and executed successfully.
31 // The code is compiled and executed successfully.
32 // The code is compiled and executed successfully.
33 // The code is compiled and executed successfully.
34 // The code is compiled and executed successfully.
35 // The code is compiled and executed successfully.
36 // The code is compiled and executed successfully.
37 // The code is compiled and executed successfully.
38 // The code is compiled and executed successfully.
39 // The code is compiled and executed successfully.
40 // The code is compiled and executed successfully.
41 // The code is compiled and executed successfully.
42 // The code is compiled and executed successfully.
43 // The code is compiled and executed successfully.
44 // The code is compiled and executed successfully.
45 // The code is compiled and executed successfully.
46 // The code is compiled and executed successfully.
47 // The code is compiled and executed successfully.
48 // The code is compiled and executed successfully.
49 // The code is compiled and executed successfully.
50 // The code is compiled and executed successfully.
51 // The code is compiled and executed successfully.
52 // The code is compiled and executed successfully.
53 // The code is compiled and executed successfully.
54 // The code is compiled and executed successfully.
55 // The code is compiled and executed successfully.
56 // The code is compiled and executed successfully.
57 // The code is compiled and executed successfully.
58 // The code is compiled and executed successfully.
59 // The code is compiled and executed successfully.
60 // The code is compiled and executed successfully.
61 // The code is compiled and executed successfully.
62 // The code is compiled and executed successfully.
63 // The code is compiled and executed successfully.
64 // The code is compiled and executed successfully.
65 // The code is compiled and executed successfully.
66 // The code is compiled and executed successfully.
67 // The code is compiled and executed successfully.
68 // The code is compiled and executed successfully.
69 // The code is compiled and executed successfully.
70 // The code is compiled and executed successfully.
71 // The code is compiled and executed successfully.
72 // The code is compiled and executed successfully.
73 // The code is compiled and executed successfully.
74 // The code is compiled and executed successfully.
75 // The code is compiled and executed successfully.
76 // The code is compiled and executed successfully.
77 // The code is compiled and executed successfully.
78 // The code is compiled and executed successfully.
79 // The code is compiled and executed successfully.
80 // The code is compiled and executed successfully.
81 // The code is compiled and executed successfully.
82 // The code is compiled and executed successfully.
83 // The code is compiled and executed successfully.
84 // The code is compiled and executed successfully.
85 // The code is compiled and executed successfully.
86 // The code is compiled and executed successfully.
87 // The code is compiled and executed successfully.
88 // The code is compiled and executed successfully.
89 // The code is compiled and executed successfully.
90 // The code is compiled and executed successfully.
91 // The code is compiled and executed successfully.
92 // The code is compiled and executed successfully.
93 // The code is compiled and executed successfully.
94 // The code is compiled and executed successfully.
95 // The code is compiled and executed successfully.
96 // The code is compiled and executed successfully.
97 // The code is compiled and executed successfully.
98 // The code is compiled and executed successfully.
99 // The code is compiled and executed successfully.
100 // The code is compiled and executed successfully.

```

[C Code](#)

```
/* Declare volatile pointers to I/O registers. This will ensure that the resulting code will bypass the cache*/
// Declaring the volatile pointers for the input ports
volatile int * InPort_Key2 = (int *) 0x00003020;    // check port address
volatile int * InPort_MSB = (int *) 0x00003040;    // check port address

// Declaring the volatile pointers for the output ports
volatile int * OutPort_REDLEDs = (int *) 0x00003010; // check port address
volatile int * OutPort_GREENLED = (int *) 0x00003030; // check port address
volatile int * OutPort_FLAGREDLED = (int *) 0x00003000; // check port address

// Declaring the main function here
int main (void)
{
    // Declaring and initialising the variables
    int Prev_MSB;           // temporary variable to save previous MSB input value
    int Prev_SW;           // temporary variable to save previous switch input value
    int Curr_MSB = *(InPort_MSB); // temporary variable to save current MSB input value
    (Initialized to current value)
    int Curr_SW = *(InPort_Key2); // temporary variable to save current switch input value
    (Initialized to current value)

    // Using a while loop here
    while (1)
    {
        // Reinitialising the variables that we will be using
        // Transfer the processed MSB value to the Prev_MSB variable
        Prev_MSB = Curr_MSB;

        // Initialising the new MSB value
        Curr_MSB = *(InPort_MSB);

        // Checking THE rising edge condition
        if (!Prev_MSB && Curr_MSB)
        {
            // Increment by 0x01 for Rising Edge Count
            // If there is an overflow at 0xFF, it will loop back to the beginning and return to 0
            *(OutPort_REDLEDs) = *(OutPort_REDLEDs) + 0x01;

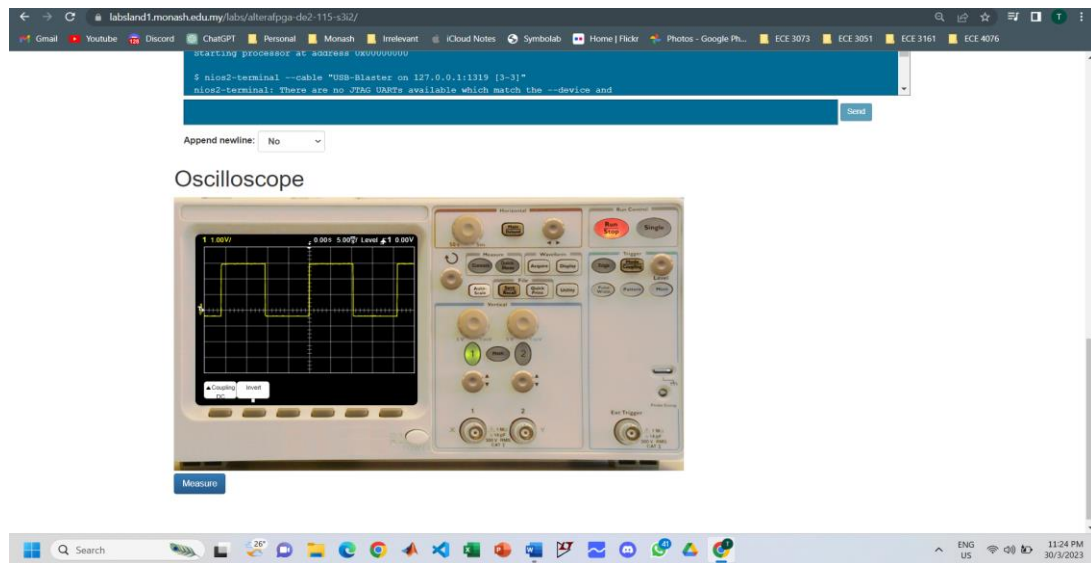
            // Invert the bit for flag red LED
            *(OutPort_FLAGREDLED) = !(*(OutPort_FLAGREDLED));
        }

        // Reinitialising the previous switch value with the current switch value
        Prev_SW = Curr_SW;

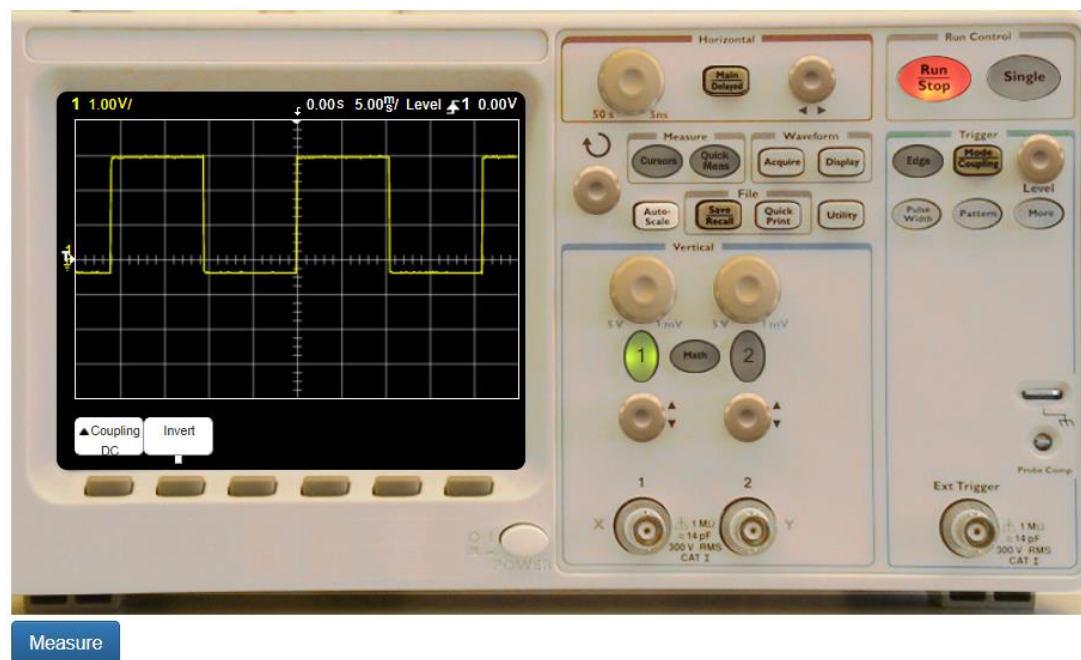
        // Checking the new value of the switch
        Curr_SW = *(InPort_Key2);
    }
}
```

```
// Checking the rising edge conditions
if(!Prev_SW && Curr_SW)
{
    // Invert the bit for the green LED
    *(OutPort_GREENLED) = !(*(OutPort_GREENLED));
}
}
```

The Oscilloscope



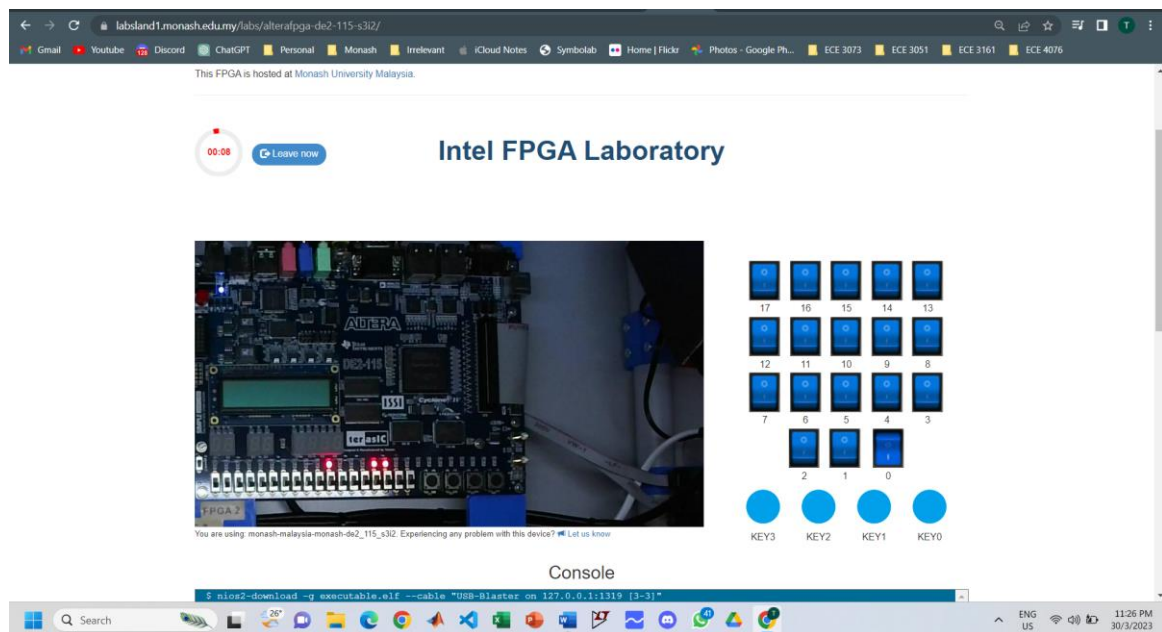
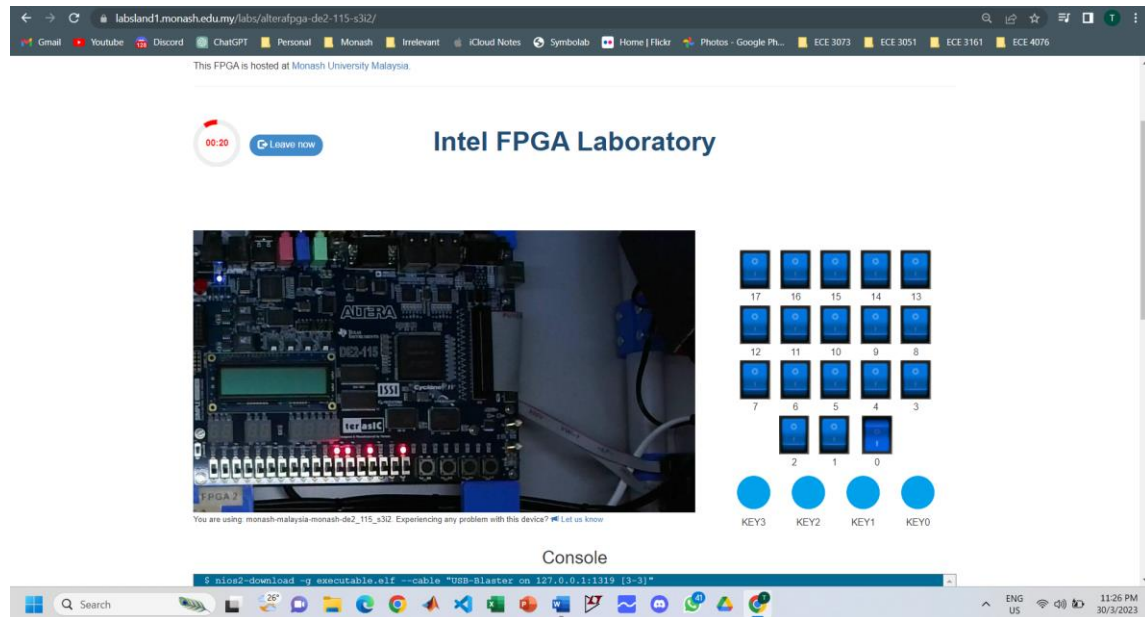
Oscilloscope



As we can see, each division is 5ms (shown in the oscilloscope above), each period of the square wave occupies approximately 4 divisions. So, each square wave will have a period of 20ms. The frequency can be calculated by using the following formula which is $f = 1/s$ and we will obtain a value of 50Hz which is what we want.

ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

The screenshot of the FPGA



2023 – Sem 1.

That must be easy lets go to the next level, Now introduce suitable C codes such that Channel 2 output (that is from GPIO connected to NIOS- The flag) follows counter MSB pulse. You can show the result by choosing Channel 1 or 2 , or using the Y-axis movement knobs as I have described in my oscilloscope demo video. An example result should like this (you can see two pulses exactly following)

The image shows an Agilent 54622A digital oscilloscope. The screen displays a green square wave on a black grid. The top of the screen shows settings: 1.00 V, 1.00 V, 0.00 V, 2.00 V, Level, 0.00 V. The right side of the image shows the physical controls of the oscilloscope, including knobs for horizontal and vertical scaling, buttons for measurements and waveforms, and a trigger section. The bottom left of the image has a blue button labeled 'Measure'.

[illegible]

[C Code](#)

```
/* Declare volatile pointers to I/O registers. This will ensure that the resulting code will bypass the cache*/
// Declaring the volatile pointers for the input ports
volatile int * InPort_Key2 = (int *) 0x00003020;    // check port address
volatile int * InPort_MSB = (int *) 0x00003040;    // check port address

// Declaring the volatile pointers for the output ports
volatile int * OutPort_REDLED = (int *) 0x00003010; // check port address
volatile int * OutPort_GREENLED = (int *) 0x00003030; // check port address
volatile int * OutPort_FLAGREDLED = (int *) 0x00003000; // check port address

// The main function
// Declaring the main function here
int main (void)
{
    // Declaring and initialising the variables
    int Prev_MSB;           // temporary variable to save previous MSB input value
    int Prev_SW;           // temporary variable to save previous switch input value
    int Curr_MSB = *(InPort_MSB); // temporary variable to save current MSB input value
    (Initialized to current value)
    int Curr_SW = *(InPort_Key2); // temporary variable to save current switch input value
    (Initialized to current value)

    // Using a while loop here
    while (1)
    {
        // Reinitialising the variables that we will be using
        // Transfer the processed MSB value to the Prev_MSB variable
        Prev_MSB = Curr_MSB;

        // Initialising the new MSB value
        Curr_MSB = *(InPort_MSB);

        // Checking THE rising edge condition
        if (!Prev_MSB && Curr_MSB)
        {
            // Increment by 0x01 for Rising Edge Count
            // If there is an overflow at 0xFF, it will loop back to the beginning and return to 0
            *(OutPort_REDLED) = *(OutPort_REDLED) + 0x01;
        }

        // Assigning the Flag red LED with MSB value
        *(OutPort_FLAGREDLED) = Curr_MSB;

        // Reinitialising the previous switch value with the current switch value
        Prev_SW = Curr_SW;

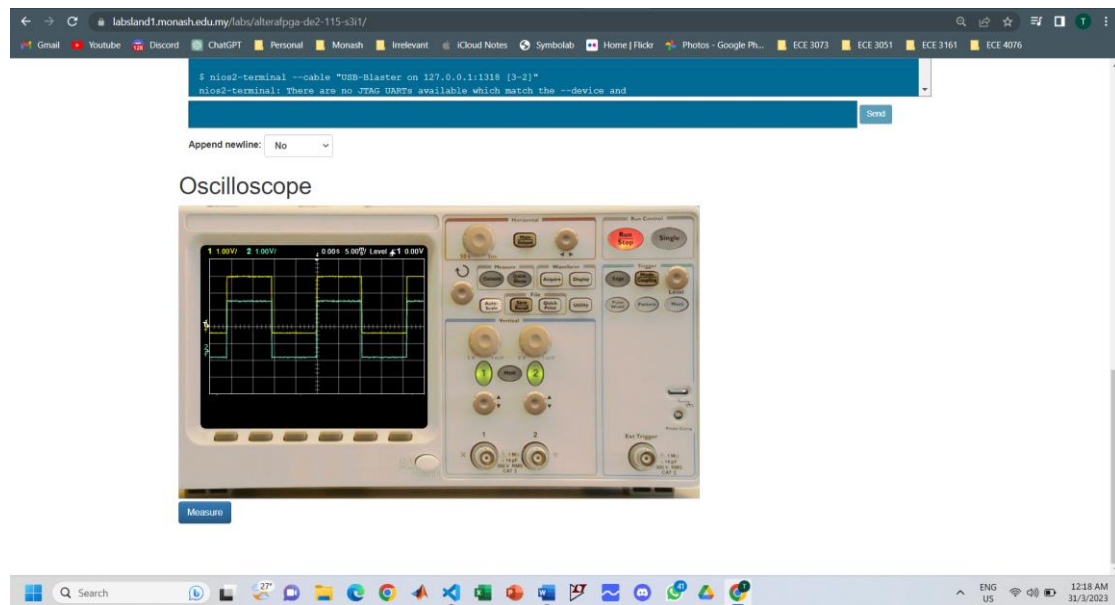
        // Checking the new value of the switch
    }
}
```

```
Curr_SW = *(InPort_Key2);

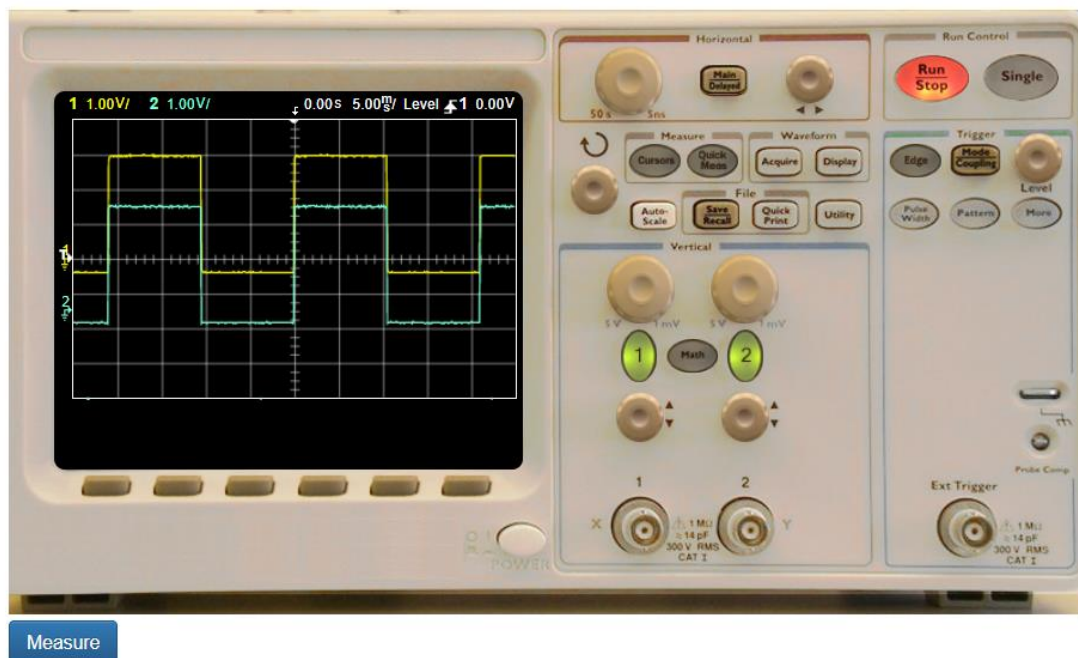
// Checking the rising edge conditions
if(!Prev_SW && Curr_SW)
{
    // Invert the bit for the green LED
    *(OutPort_GREENLED) = !(*(OutPort_GREENLED));
}
}
```

ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

Oscilloscope



Oscilloscope



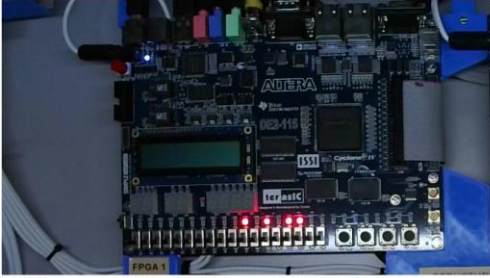
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

FPGA Screenshot

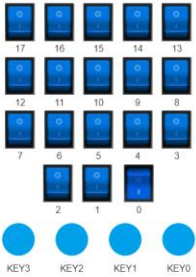
labsland1.monash.edu.my/labs/alterafpga-de2-115-s31/

00:50 Leave now

Intel FPGA Laboratory



You are using: monash-malaysia-monash-de2_115_s31. Experiencing any problem with this device? Let us know



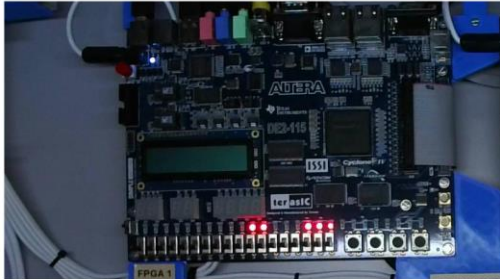
Console

```
$ nios2-download -g executable.elf --cable "USB-Blaster on 127.0.0.1:1312 [3-2]"  
Using cable "USB-Blaster on 127.0.0.1:1312 [3-2]" device 1, instance 0x00
```

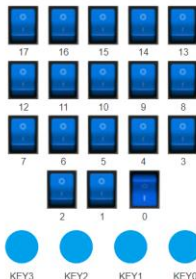
labsland1.monash.edu.my/labs/alterafpga-de2-115-s31/

00:28 Leave now

Intel FPGA Laboratory



You are using: monash-malaysia-monash-de2_115_s31. Experiencing any problem with this device? Let us know



Console

```
$ nios2-download -g executable.elf --cable "USB-Blaster on 127.0.0.1:1312 [3-2]"  
Using cable "USB-Blaster on 127.0.0.1:1312 [3-2]" device 1, instance 0x00
```

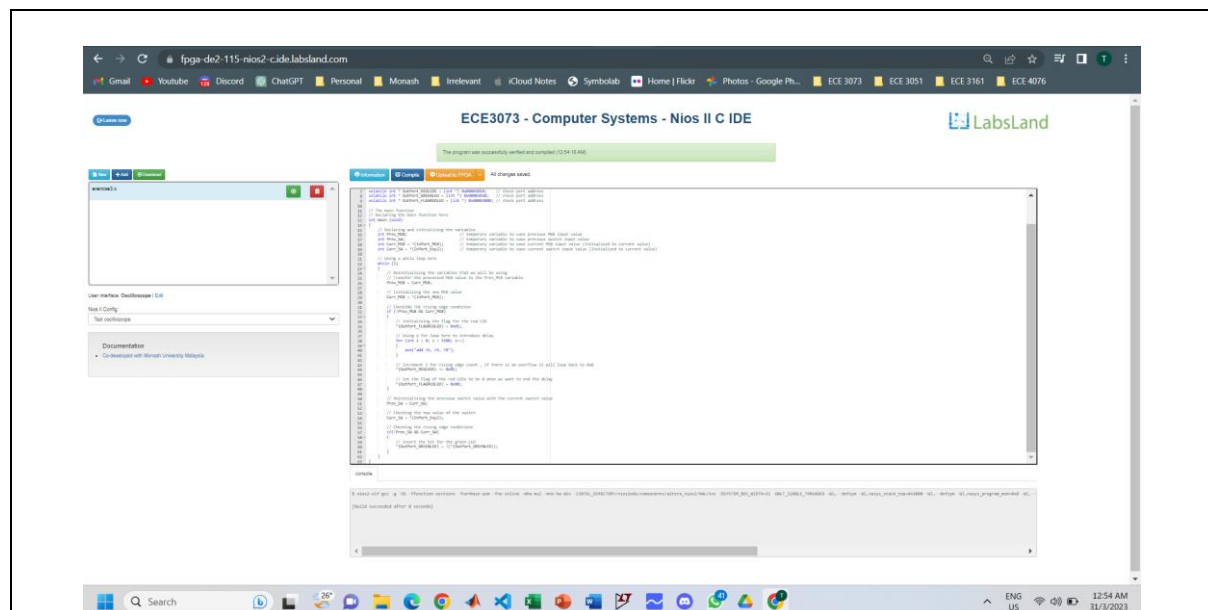
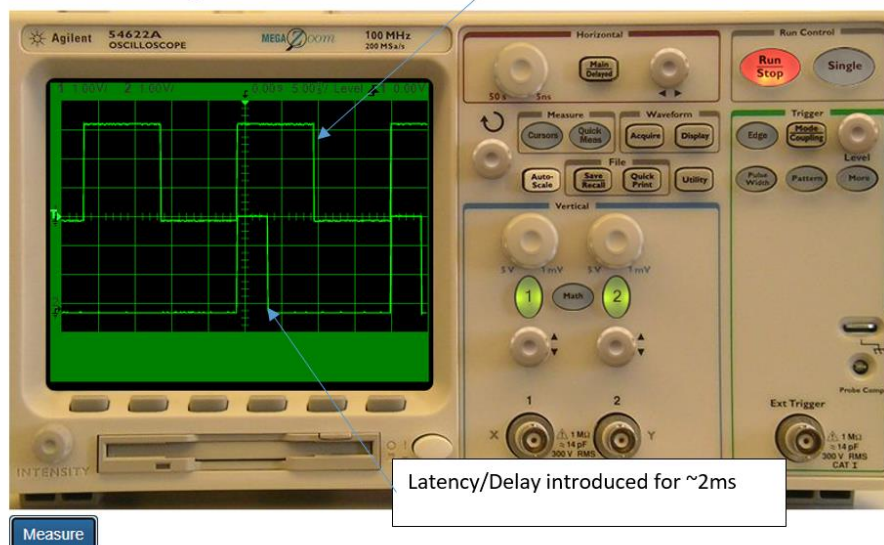
Exercise 3 (10 marks)

Ok that's getting bit moderately hard ☺ , let me test you further hard

Lets assume each NIOS instruction including the FPGA and our labsland delays takes 200 ns.
Example an add instruction takes 200 ns. Using this as clue introduce a latency(delay) of approximately 2 ms to your 8 – LED flash. Which means once the rising edge is detected it should take approximately 2 ms to show the count value /change in count value in 8 – RED LEDs. Show that you introduced 2 ms delay once the rising edge is detected using oscilloscope measurement. One hint I can give here as follows, we can execute assembly code in C program using asm function. Example `asm("ldw r6, 100(r5)");` tells the compiler to execute single NIOS instruction `ldw` as machine code. Use this idea to create the latency. Screen shot your code and the oscilloscope output , I am attaching a sample how it can look like . You can see that channel 2 is now a short pulse of 2 ms ON time. How to use the oscilloscope knobs pls refer to the demo video.

how it can look like

Oscilloscope



[C Code](#)

```
/* Declare volatile pointers to I/O registers. This will ensure that the resulting code will bypass the
cache*/
// Declaring the volatile pointers for the input ports
volatile int * InPort_Key2 = (int *) 0x00003020;    // check port address
volatile int * InPort_MSB = (int *) 0x00003040;    // check port address

// Declaring the volatile pointers for the output ports
volatile int * OutPort_REDLED = (int *) 0x00003010; // check port address
volatile int * OutPort_GREENLED = (int *) 0x00003030; // check port address
volatile int * OutPort_FLAGREDLED = (int *) 0x00003000; // check port address

// The main function
// Declaring the main function here
int main (void)
{
    // Declaring and initialising the variables
    int Prev_MSB;           // temporary variable to save previous MSB input value
    int Prev_SW;           // temporary variable to save previous switch input value
    int Curr_MSB = *(InPort_MSB); // temporary variable to save current MSB input value
    (Initialized to current value)
    int Curr_SW = *(InPort_Key2); // temporary variable to save current switch input value
    (Initialized to current value)

    // Using a while loop here
    while (1)
    {
        // Reinitialising the variables that we will be using
        // Transfer the processed MSB value to the Prev_MSB variable
        Prev_MSB = Curr_MSB;

        // Initialising the new MSB value
        Curr_MSB = *(InPort_MSB);

        // Checking THE rising edge condition
        if (!Prev_MSB && Curr_MSB)
        {
            // Initialising the flag for the red LED
            *(OutPort_FLAGREDLED) = 0x01;

            // Using a for loop here to introduce delay
            for (int i = 0; i < 5500; i++)
            {
                asm("add r6, r6, r0");
            }

            // Increment 1 for rising edge count , if there is an overflow it will loop back to 0x0
            *(OutPort_REDLED) += 0x01;
        }
    }
}
```



```
    // Set the flag of the red LEDs to be 0 when we want to end the delay
    *(OutPort_FLAGREDLED) = 0x00;
}

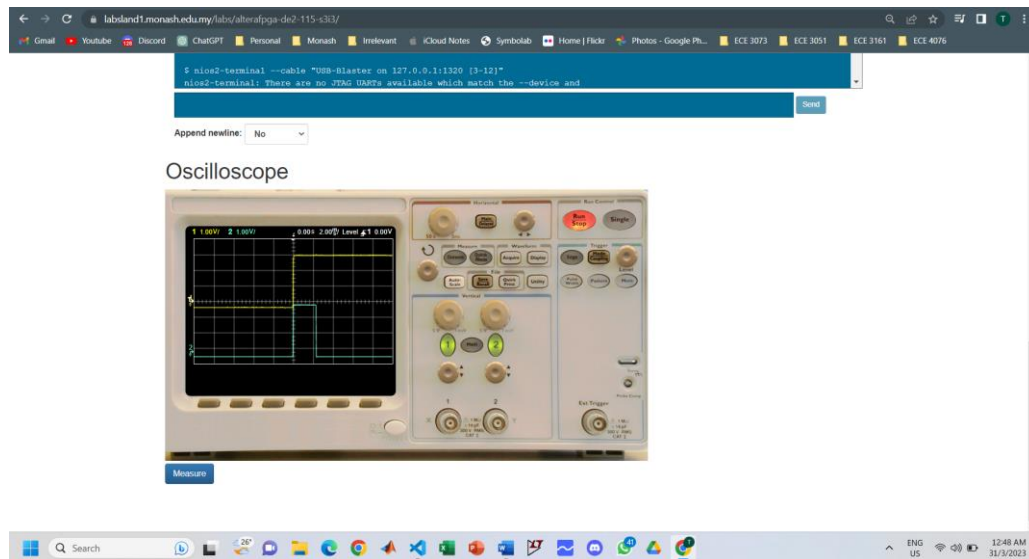
// Reinitialising the previous switch value with the current switch value
Prev_SW = Curr_SW;

// Checking the new value of the switch
Curr_SW = *(InPort_Key2);

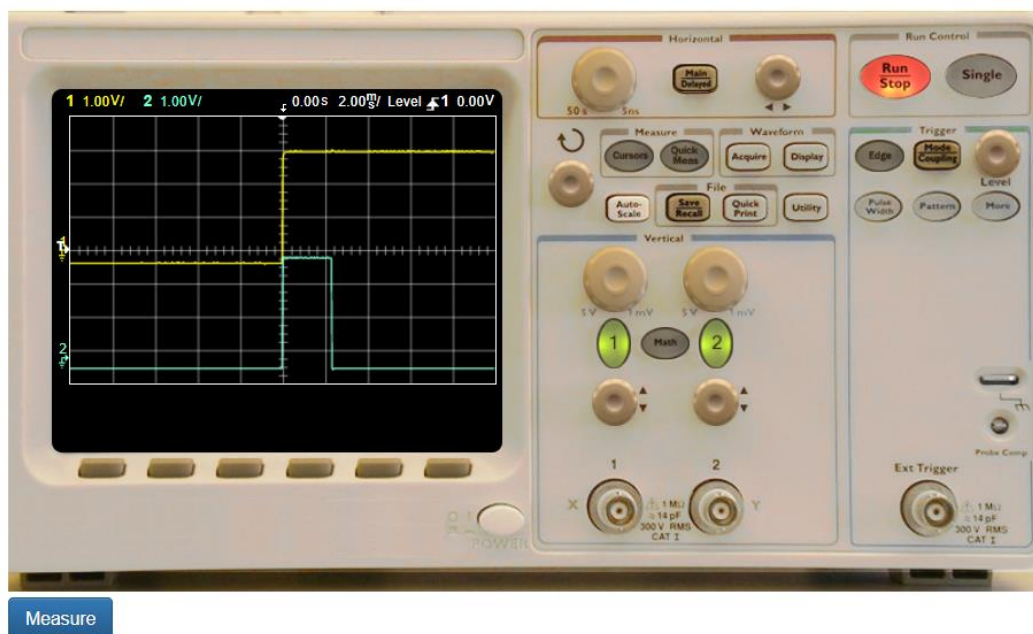
// Checking the rising edge conditions
if(!Prev_SW && Curr_SW)
{
    // Invert the bit for the green LED
    *(OutPort_GREENLED) = !*(OutPort_GREENLED);
}
}
```

Oscilloscope

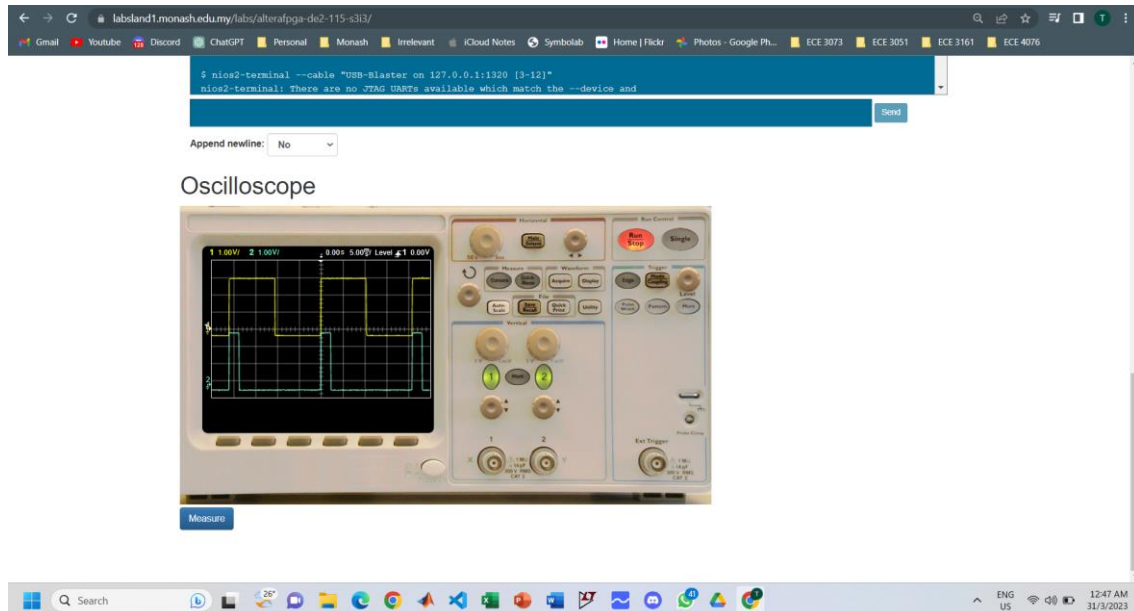
Scaling of 2ms/div



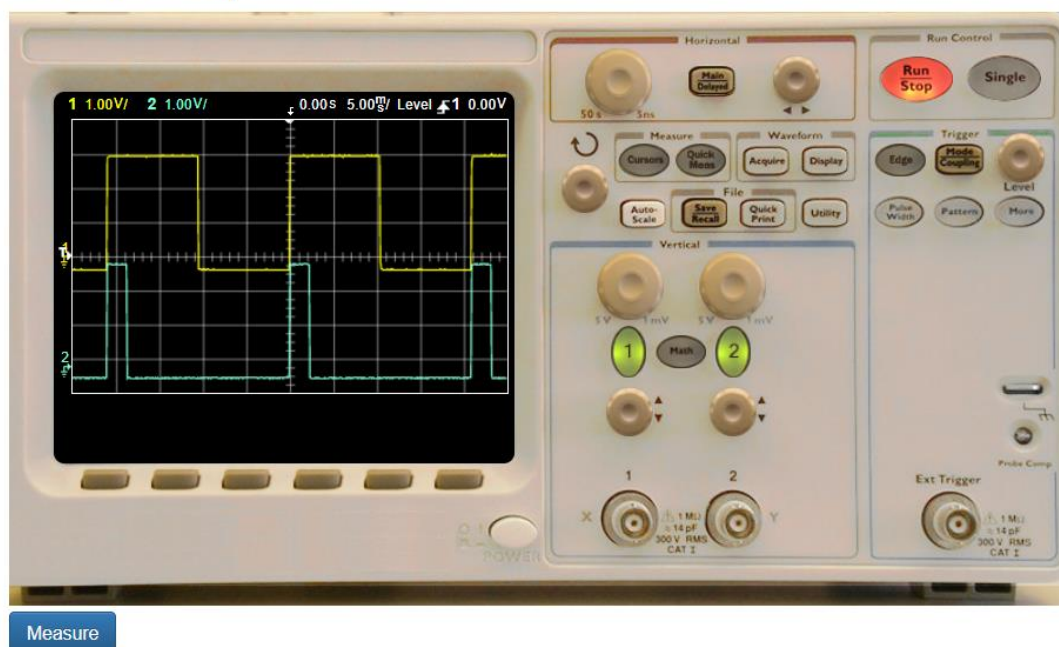
Oscilloscope



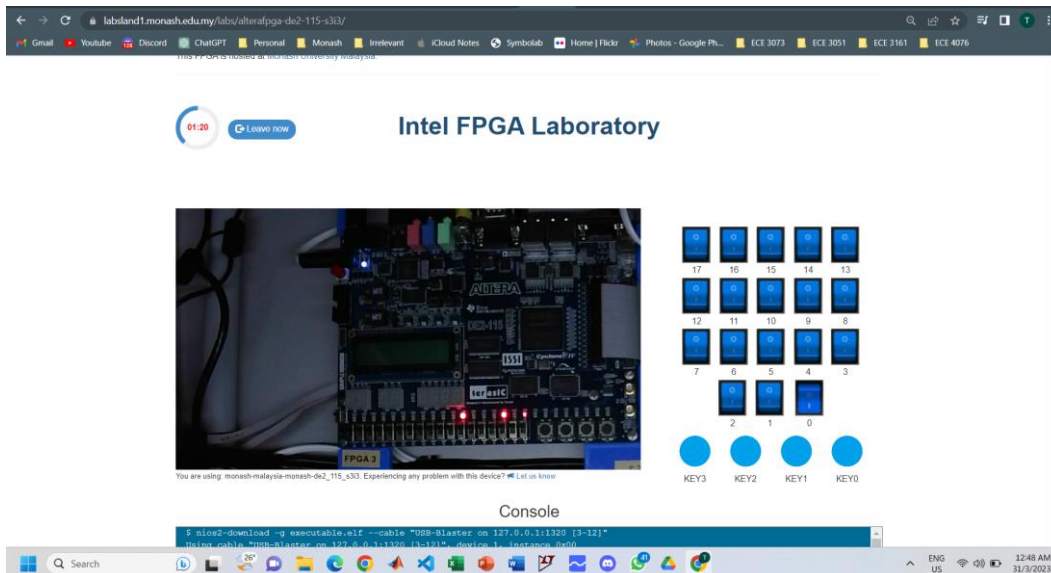
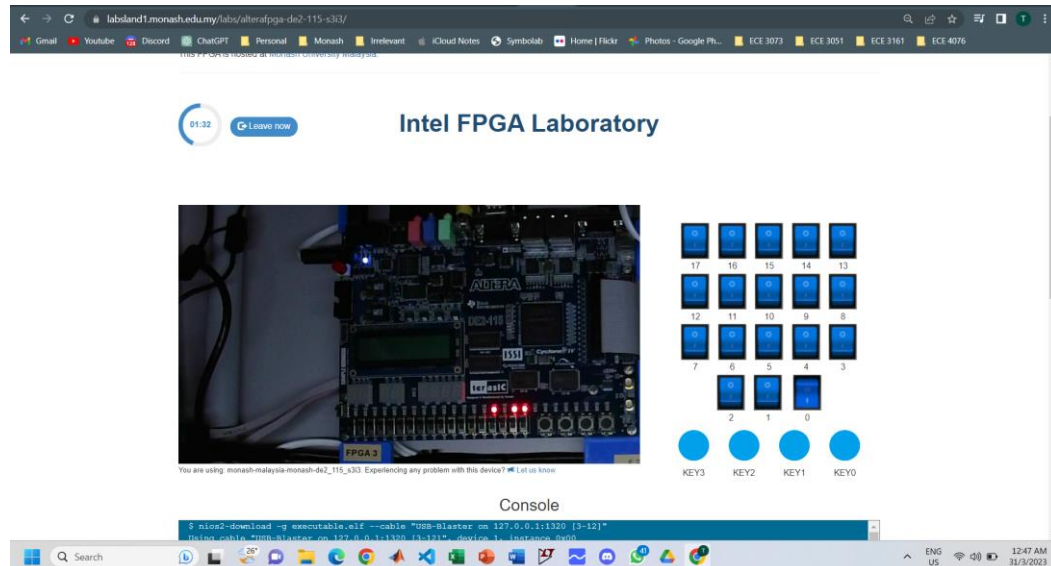
Scaling of 5ms/div



Oscilloscope



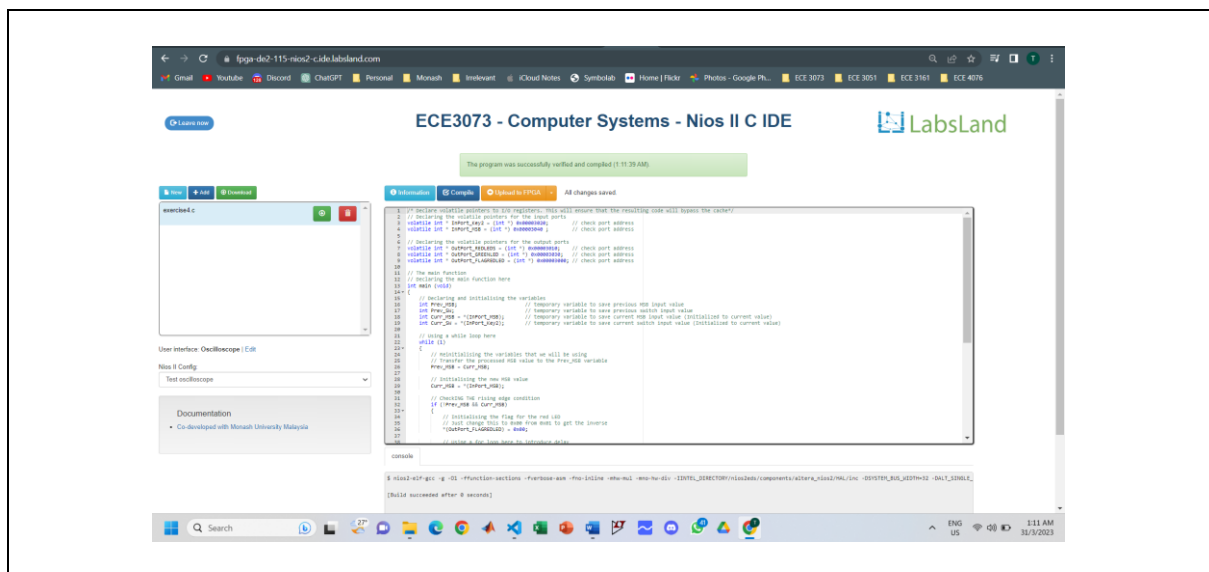
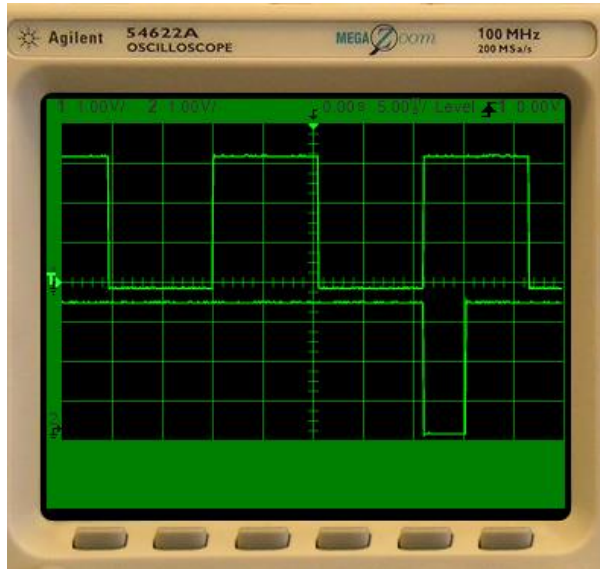
FPGA Screenshot



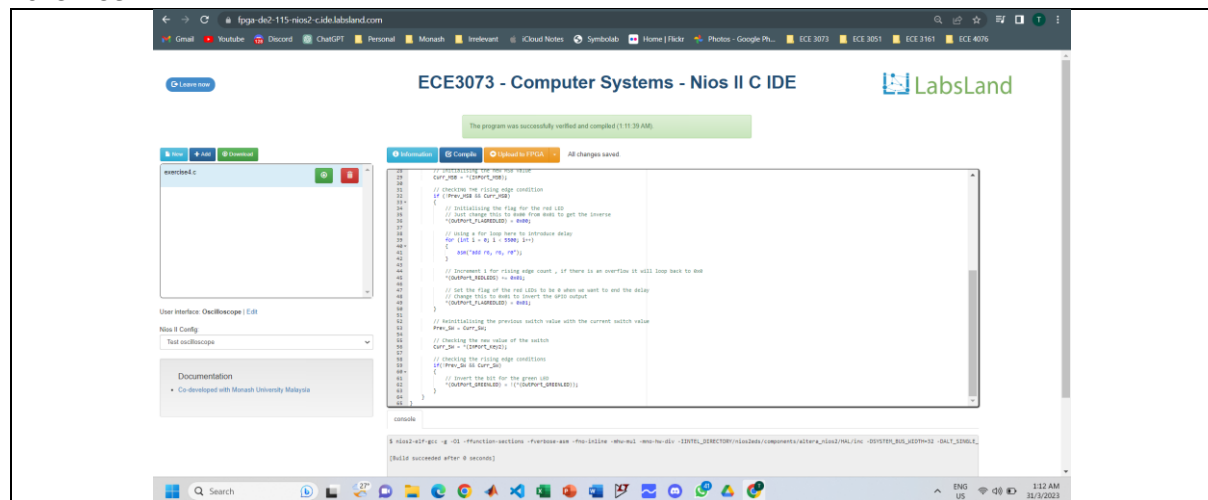
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

Exercise – 4 (5 marks)

Ok now more harder, invert your channel 2 GPIO output using suitable logic and code. Example my previous exercise output is now inverted as below



ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.



C Code

```
/* Declare volatile pointers to I/O registers. This will ensure that the resulting code will bypass the cache*/
```

```
// Declaring the volatile pointers for the input ports
```

```
volatile int * InPort_Key2 = (int *) 0x00003020;    // check port address
```

```
volatile int * InPort_MSB = (int *) 0x00003040;    // check port address
```

```
// Declaring the volatile pointers for the output ports
```

```
volatile int * OutPort_REDLEDs = (int *) 0x00003010; // check port address
```

```
volatile int * OutPort_GREENLED = (int *) 0x00003030; // check port address
```

```
volatile int * OutPort_FLAGREDLED = (int *) 0x00003000; // check port address
```

```
// The main function
```

```
// Declaring the main function here
```

```
int main (void)
```

```
{
```

```
    // Declaring and initialising the variables
```

```
    int Prev_MSB;           // temporary variable to save previous MSB input value
```

```
    int Prev_SW;           // temporary variable to save previous switch input value
```

```
    int Curr_MSB = *(InPort_MSB);    // temporary variable to save current MSB input value
```

```
(Initialized to current value)
```

```
    int Curr_SW = *(InPort_Key2);    // temporary variable to save current switch input value
```

```
(Initialized to current value)
```

```
    // Using a while loop here
```

```
    while (1)
```

```
    {
```

```
        // Reinitialising the variables that we will be using
```

```
        // Transfer the processed MSB value to the Prev_MSB variable
```

```
        Prev_MSB = Curr_MSB;
```

```
        // Initialising the new MSB value
```

```
        Curr_MSB = *(InPort_MSB);
```

```
        // Checking THE rising edge condition
```



```
if (!Prev_MSB && Curr_MSB)
{
    // Initialising the flag for the red LED
    // Just change this to 0x00 from 0x01 to get the inverse
    *(OutPort_FLAGREDLED) = 0x00;

    // Using a for loop here to introduce delay
    for (int i = 0; i < 5500; i++)
    {
        asm("add r6, r6, r0");
    }

    // Increment 1 for rising edge count , if there is an overflow it will loop back to 0x0
    *(OutPort_REDLEDS) += 0x01;

    // Set the flag of the red LEDs to be 0 when we want to end the delay
    // Change this to 0x01 to invert the GPIO output
    *(OutPort_FLAGREDLED) = 0x01;
}

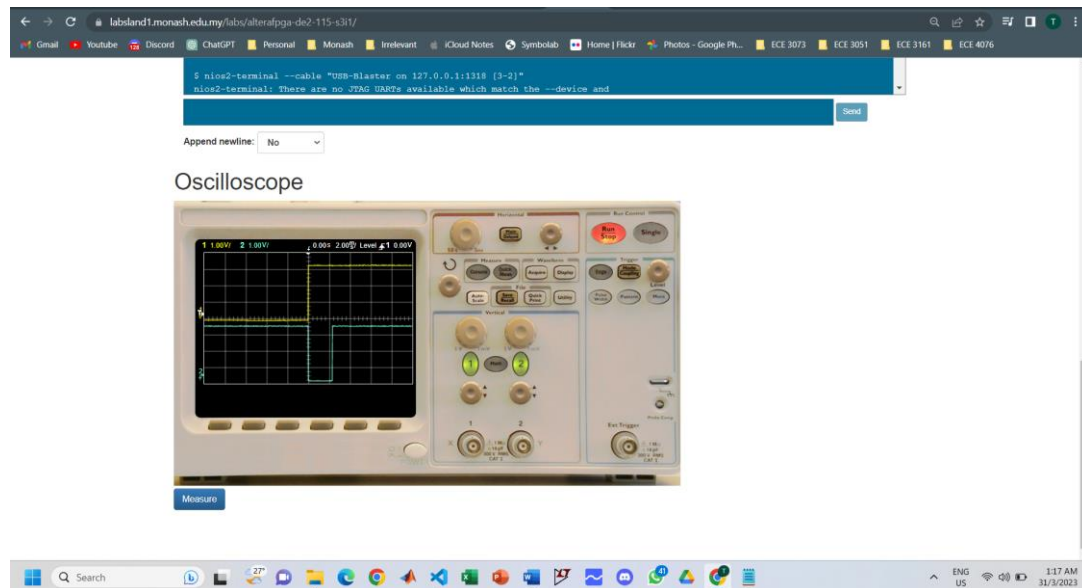
// Reinitialising the previous switch value with the current switch value
Prev_SW = Curr_SW;

// Checking the new value of the switch
Curr_SW = *(InPort_Key2);

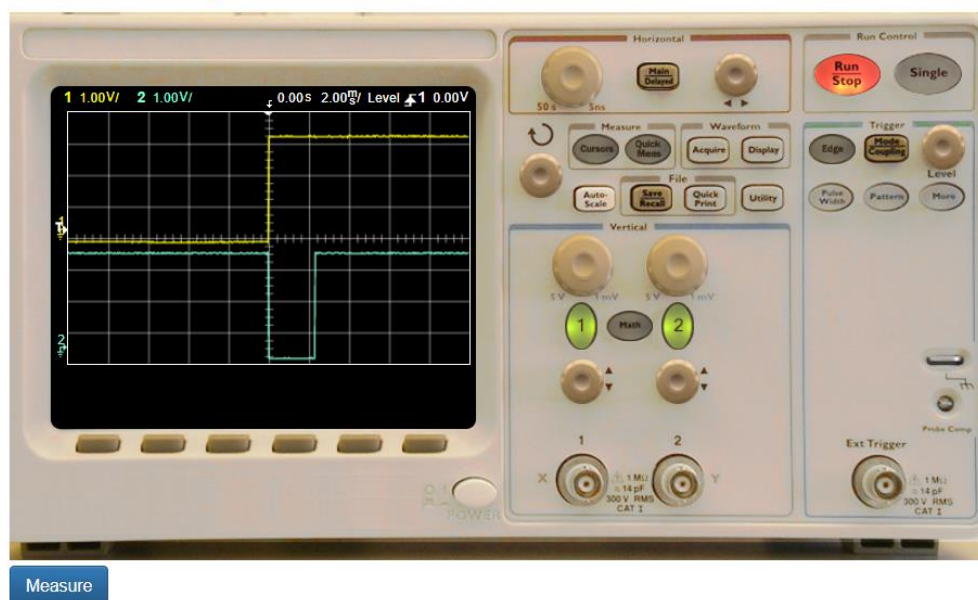
// Checking the rising edge conditions
if (!Prev_SW && Curr_SW)
{
    // Invert the bit for the green LED
    *(OutPort_GREENLED) = !(*(OutPort_GREENLED));
}
}
```

Oscilloscope

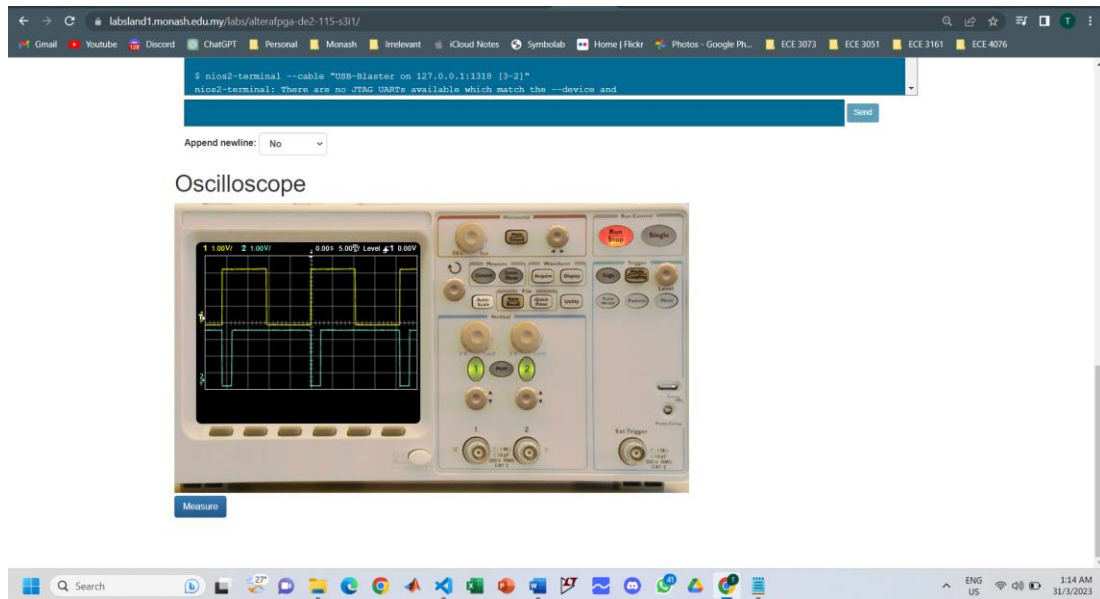
Scaling of 2ms/div



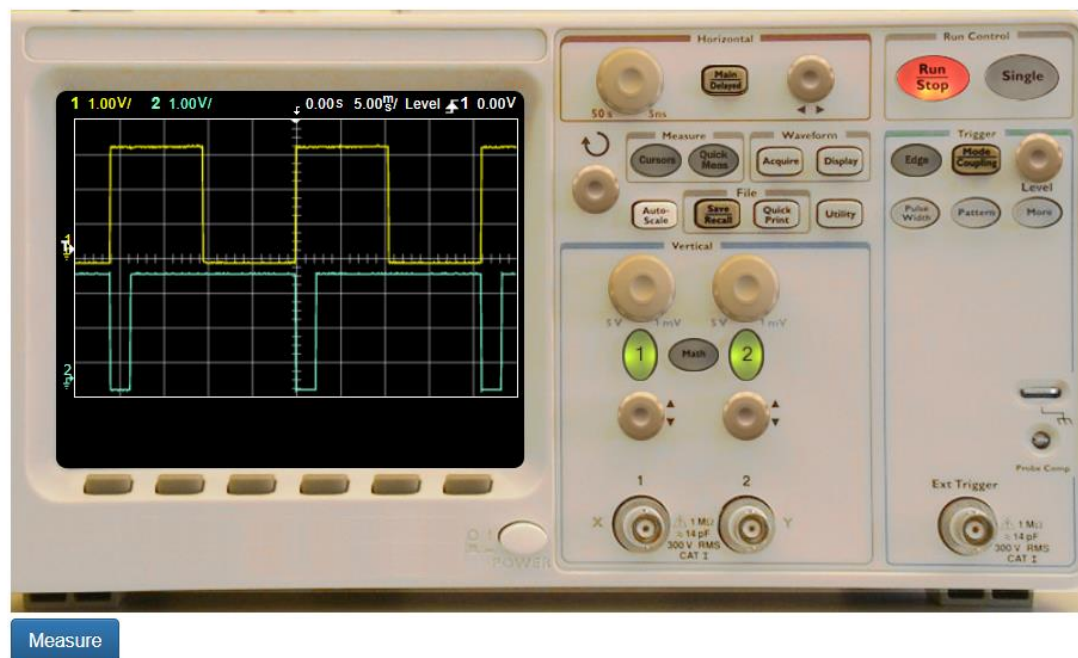
Oscilloscope



Scaling of 5ms/div

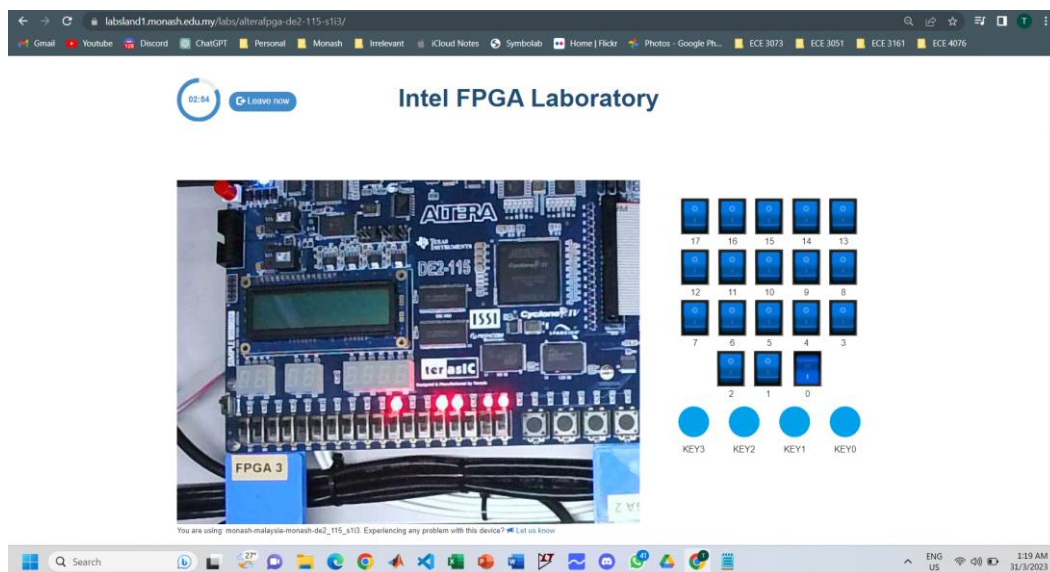
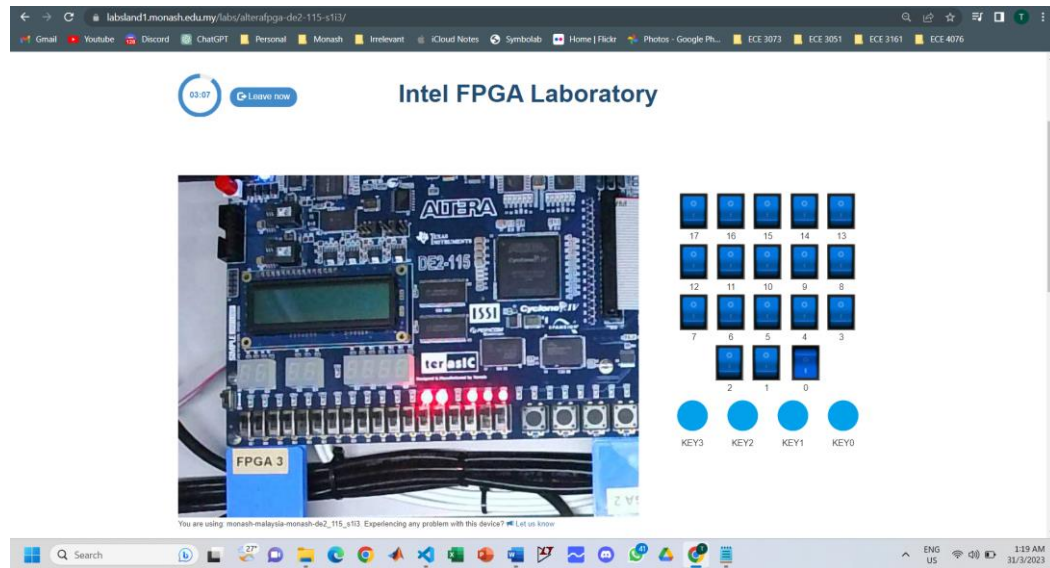


Oscilloscope

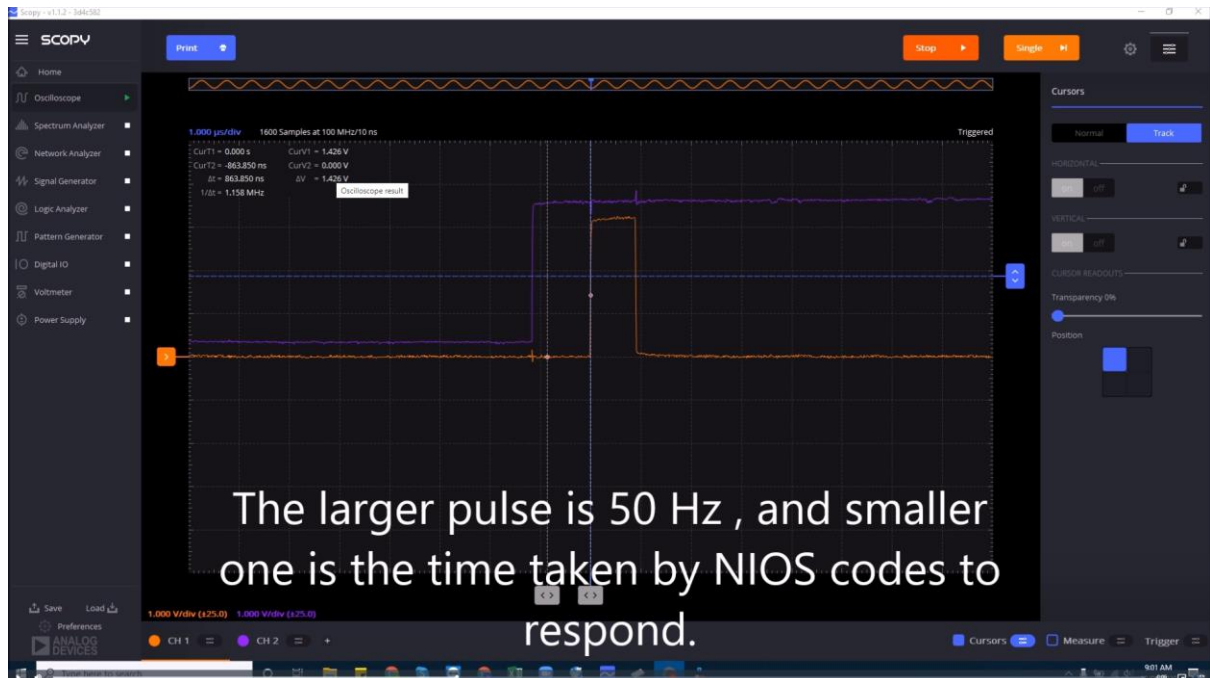


ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

FPGA Screenshot



Exercise 5 (5 marks)



The above is the screenshot from the oscilloscope response measured for Lab3B latency between the MSB and RED LED Response. Estimate from the screenshot here what is the latency. Hint: We have to subtract the time when the GPIO went high (orange pulse) once detecting when the rising edge occurred (Purple pulse)

Type your answer here:

We can see from the above screenshot of the oscilloscope response measured for Lab3B that that 1 division will be approximately 1 microsecond (Scaling of 1 microsecond per division). We can estimate that there are 1.15 division between the orange pulse (the time when GPIO went high) and the purple pulse (the time when the rising edge occurred) based on the above graph. Therefore, the latency is approximately $1.15 \text{ division} \times 1 \text{ microsecond per division}$ which equates to about **1.15 microseconds**.

The answer is **1.15 μs** .