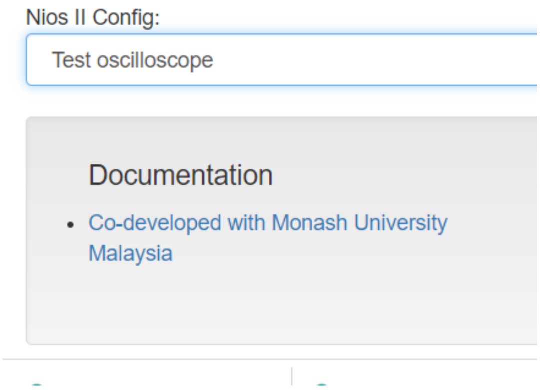
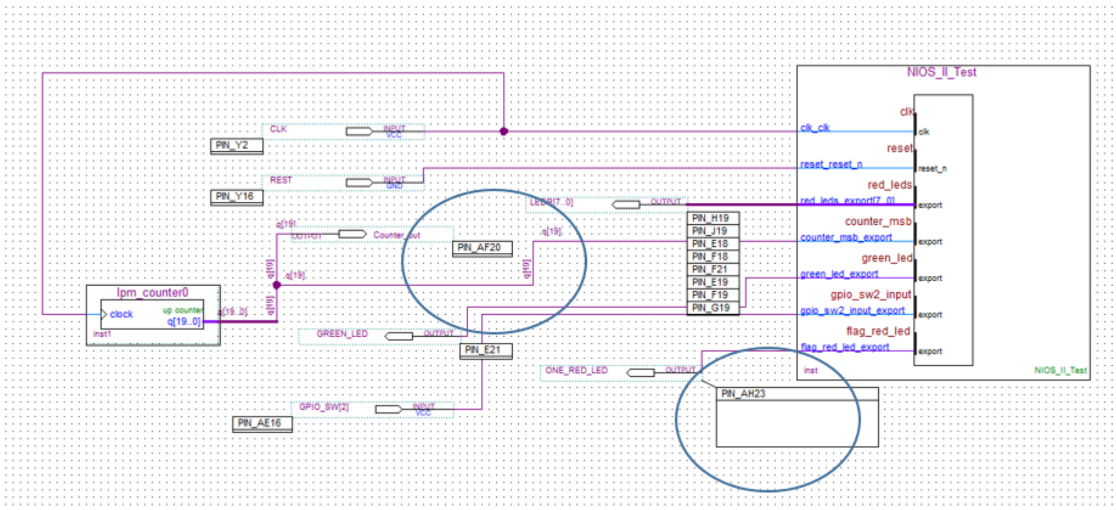
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

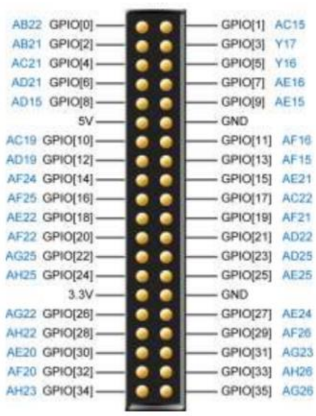
In this exercise we extend our previous Lab 3B work to further understand the latency caused by NIOS instructions through solving following challenging problems. Before getting into the problem let me describe what's been set in the labs land. You will need to choose "Test Oscilloscope" settings for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



Following is the schematic of the FPGA (Qysys)



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.



ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.
Following is the address- memory map of NIOS computer system

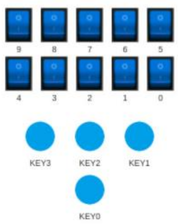
Name	Export	Clock	Base	End
red_leds	Double-click to export Double-click to export Double-click to export	clk_0 [clk] [clk]	8-RED LED Base Address (8 bit) 0x3010	0x301f
	Double-click to export Double-click to export Double-click to export	clk_0 [clk1] [clk1]	0x0000	0x1fff
counter_msb	Double-click to export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

How to Locate Oscilloscope in LabsLand?

Click "Edit" in user interface you will see below

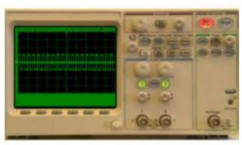
User interface

○ Standard



Default user interface: camera, switches and buttons

◉ Oscilloscope



Everything in standard plus an oscilloscope

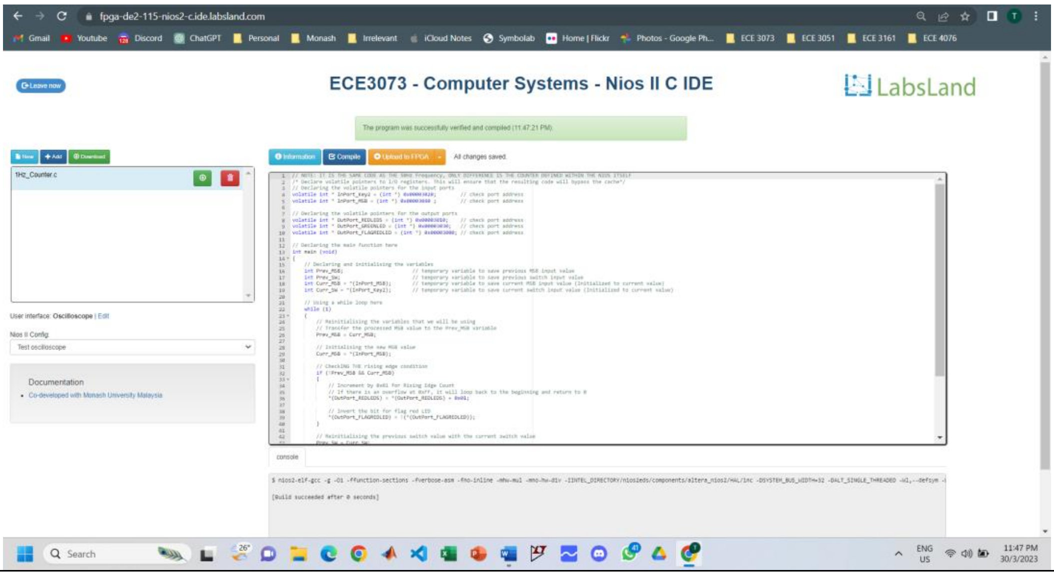
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.
Choose oscilloscope like above and close. Remember you will need to click the "Measure" button to do any oscilloscope measurement

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

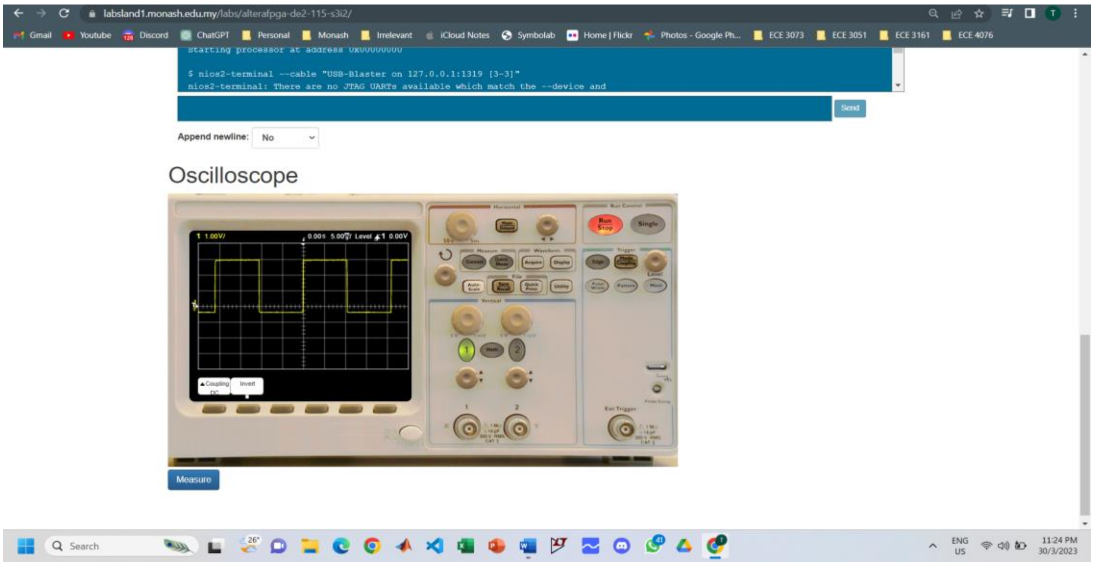


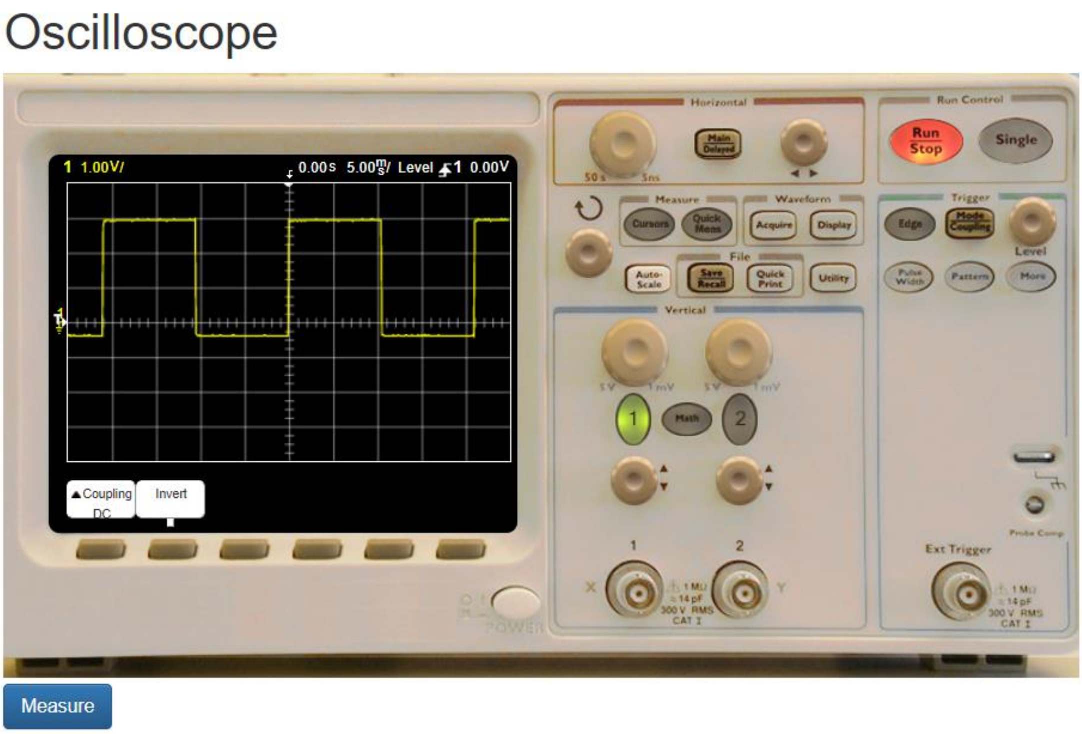
C Code

```
/* 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));
        }
    }
}
```

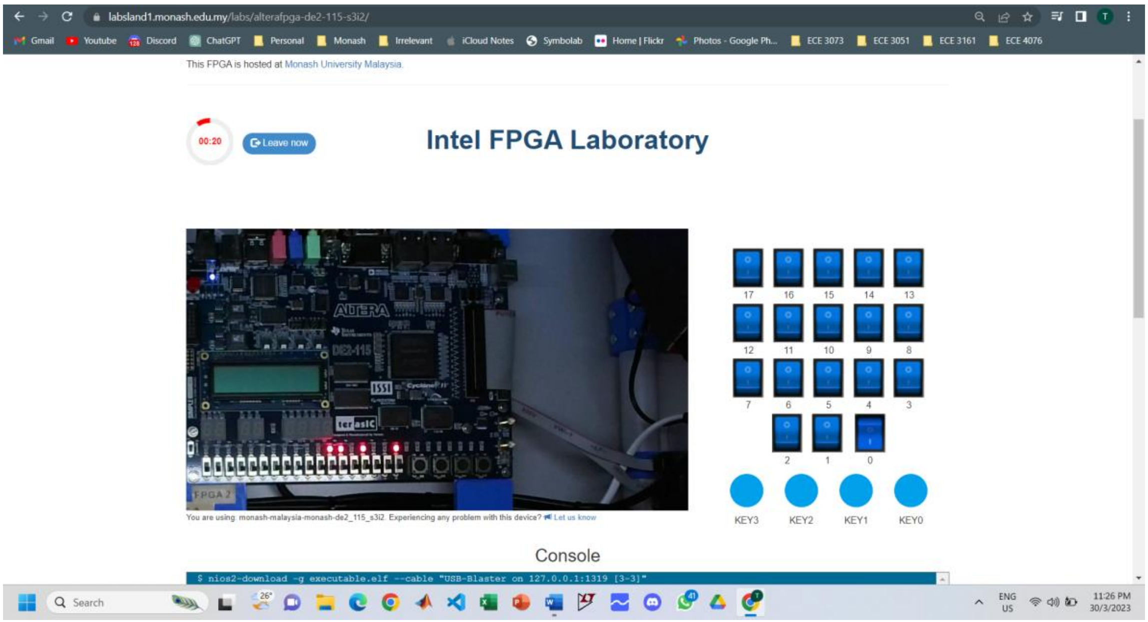
The Oscilloscope

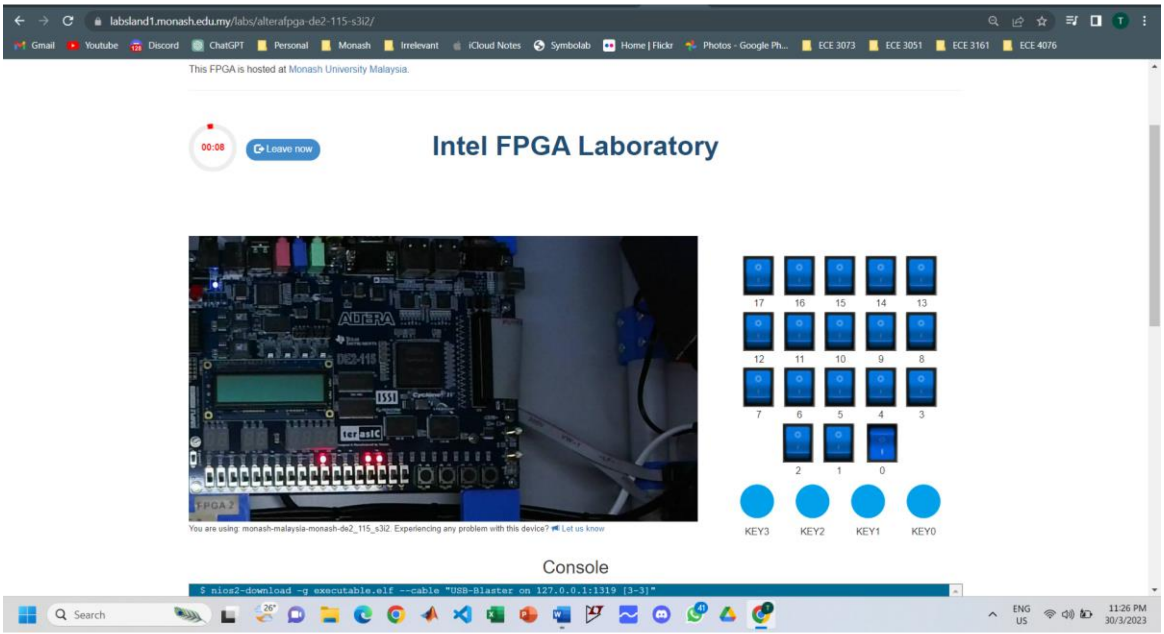




As we can see, each division is 5ms (shown in the oscilloscope above), each period of the square wave occupies approximately 4 divisions. So, each square wave will have a period of 20ms. The frequency can be calculated by using the following formula which is f = 1/s and we will obtain a value of 50Hz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

The screenshot of the FPGA

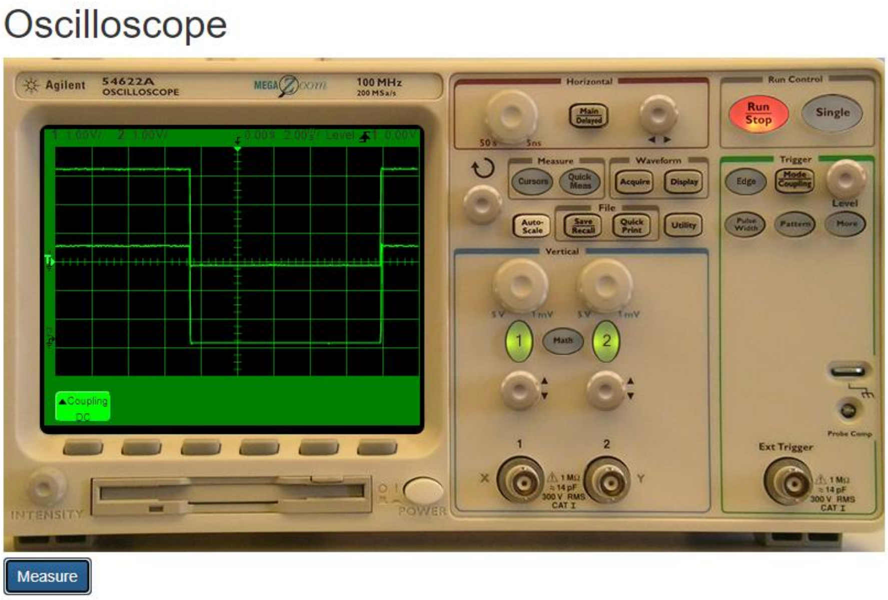




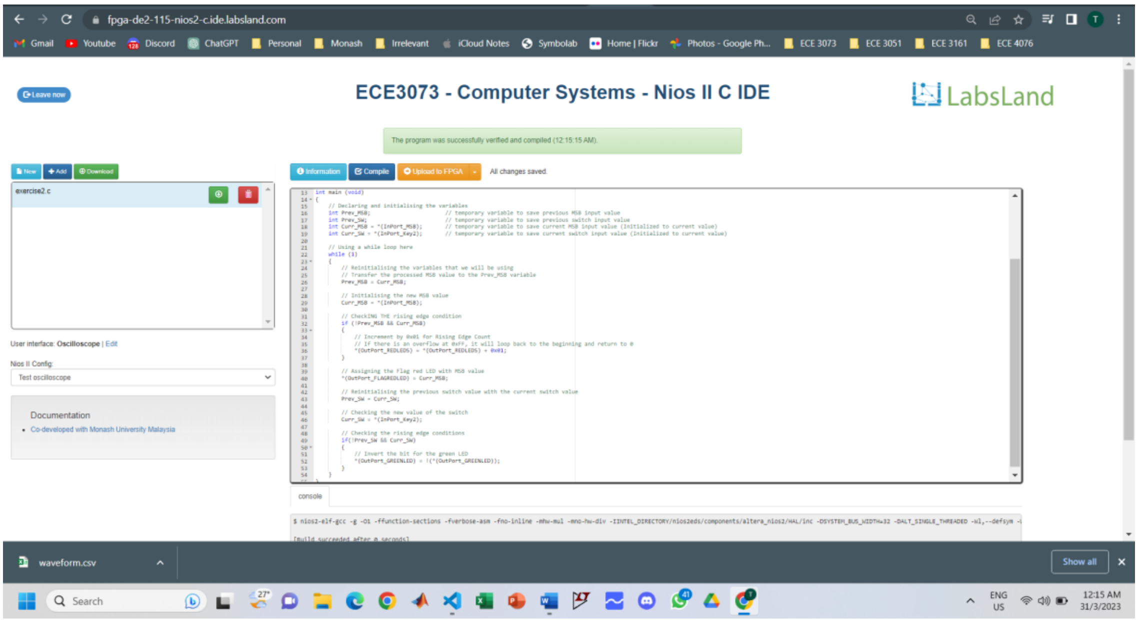
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

Exercise 2

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)



Now, screenshot your code what you wrote to achieve this, also screenshot your result.

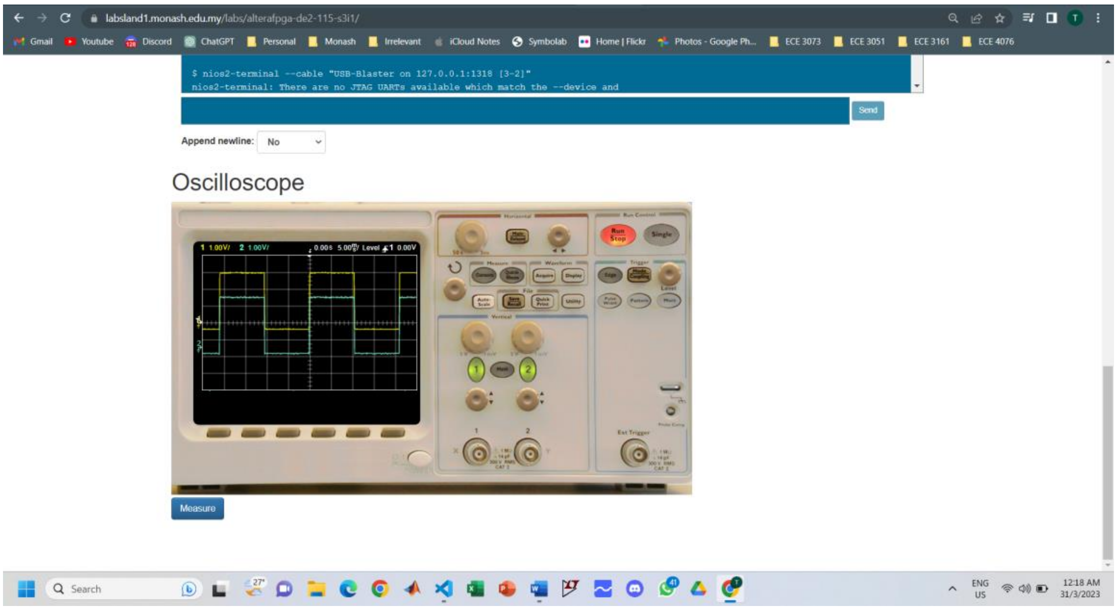


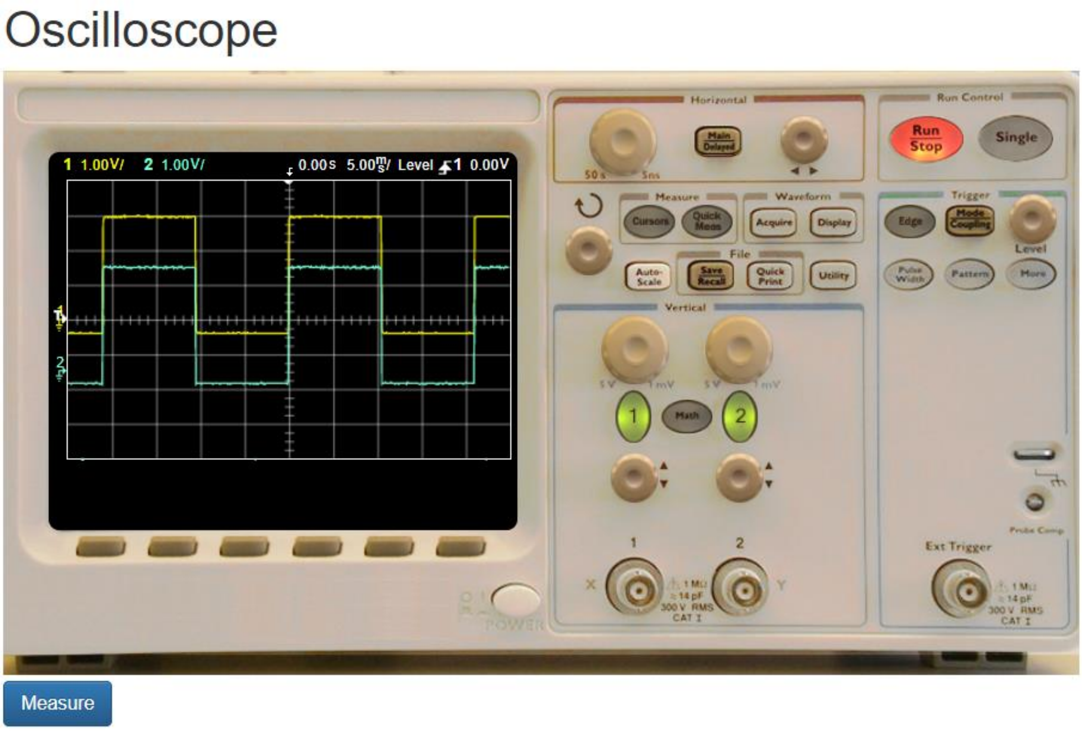
C Code

```
/* 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)
      {
         // 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;
      }

      // 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
```

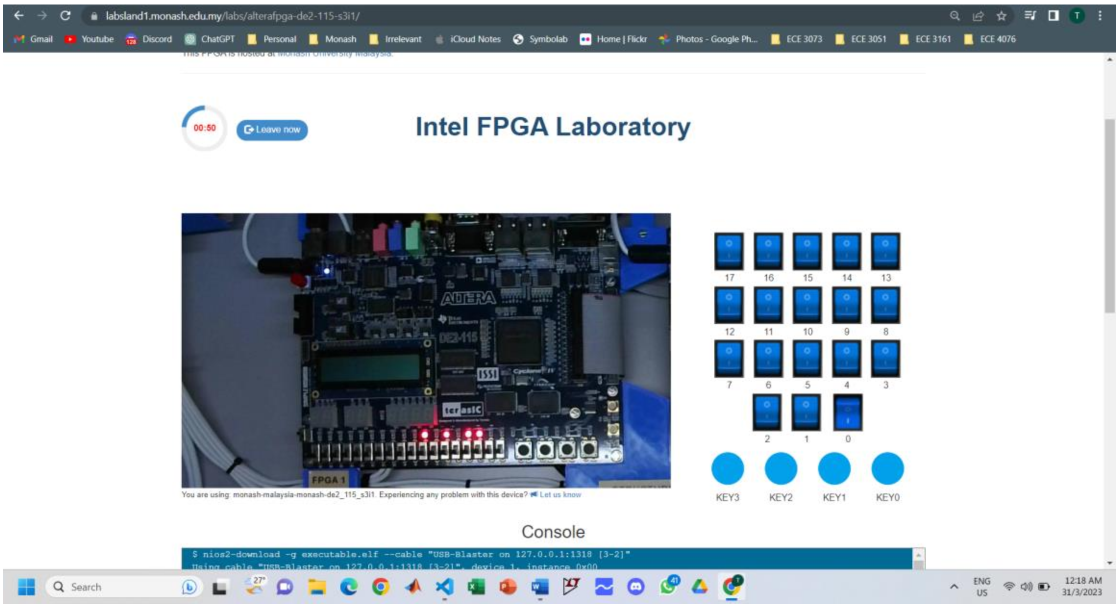
```
        Curr_SW = *(InPort_Key2);

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

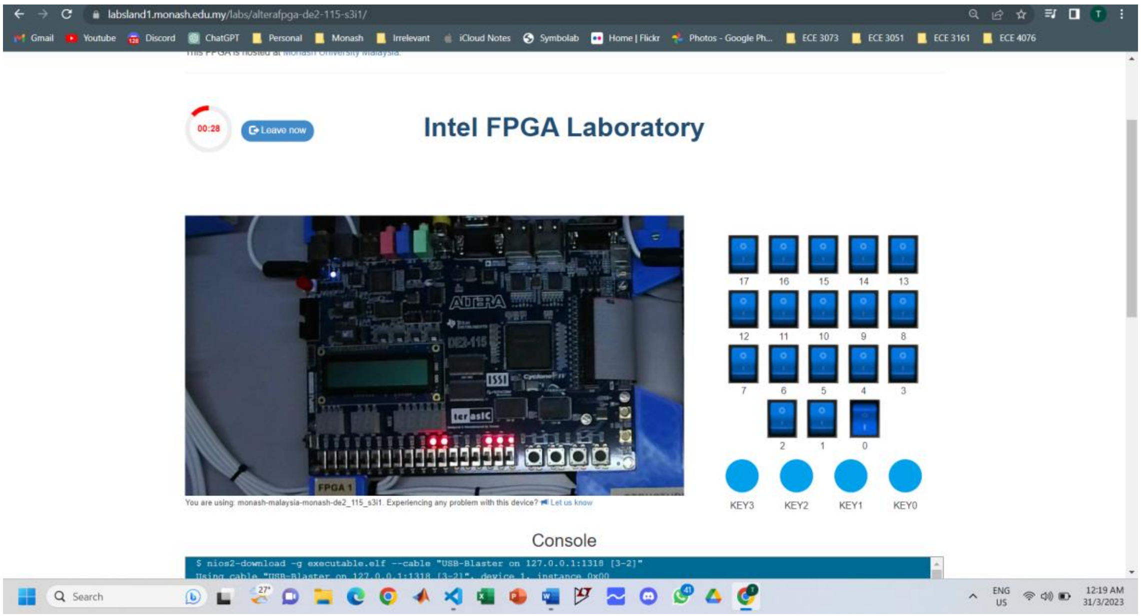
Oscilloscope





FPGA Screenshot



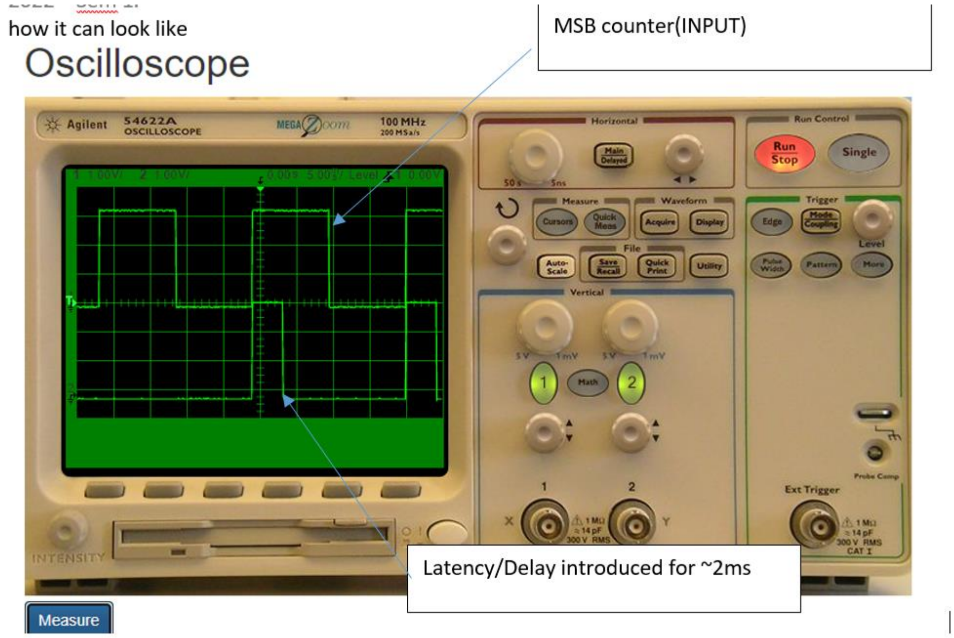


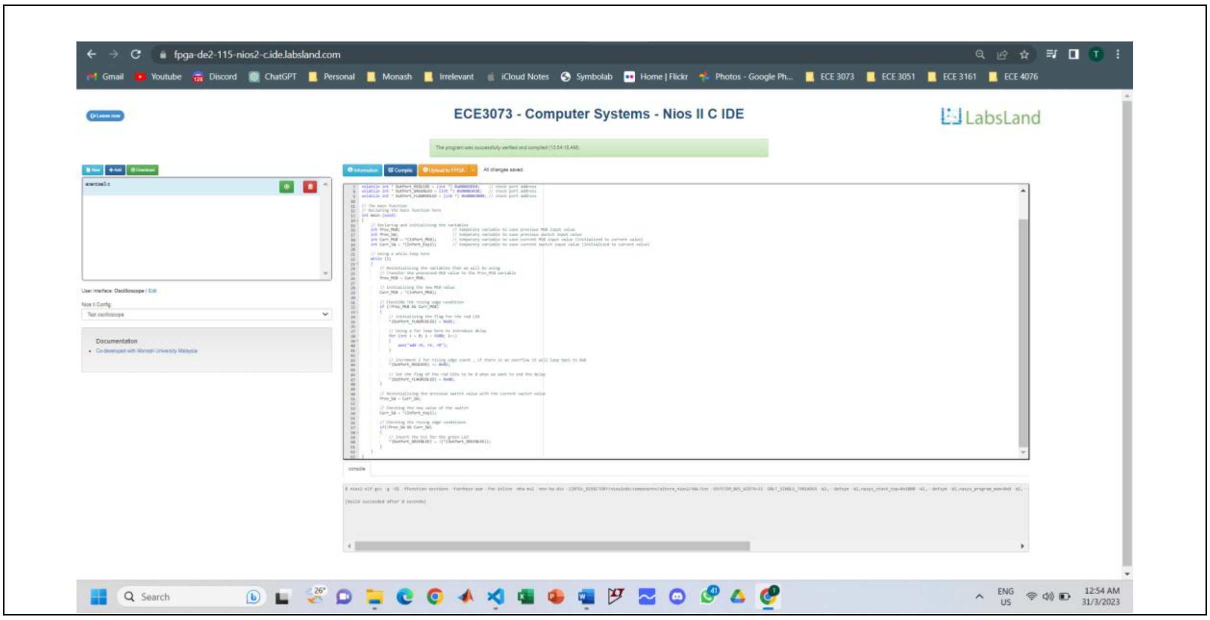
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

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.





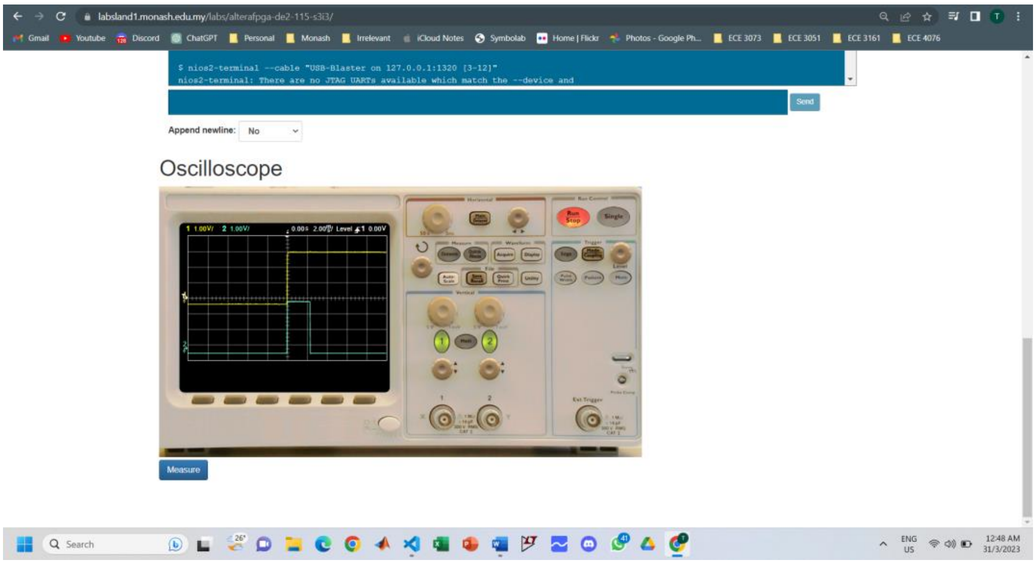
C Code

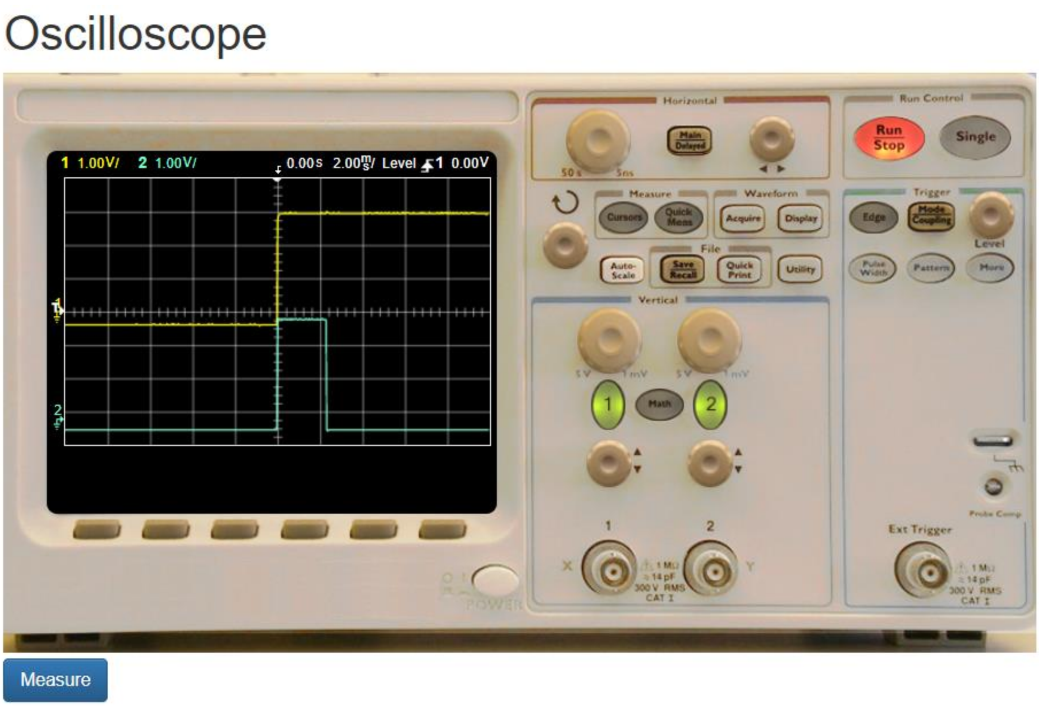
```
/* Declare volatile pointers to I/O registers. This will ensure that the resulting code will bypass the
cache*/
// Declaring the volatile pointers for the input ports
volatile int * InPort_Key2 = (int *) 0x00003020;      // check port address
volatile int * InPort_MSB = (int *) 0x00003040 ;     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

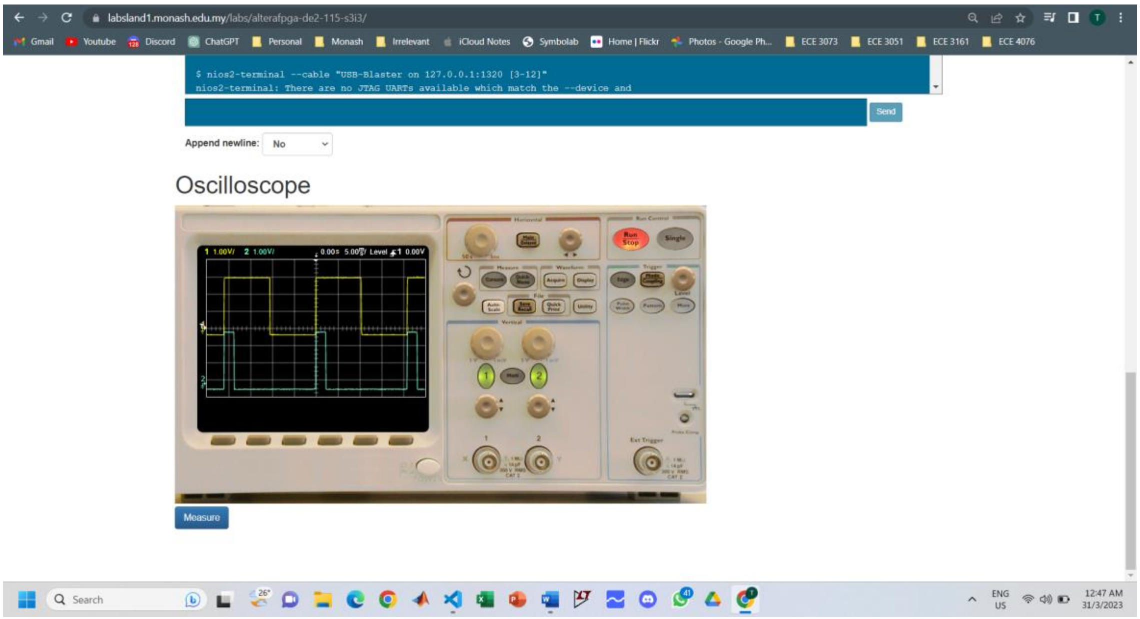
Oscilloscope

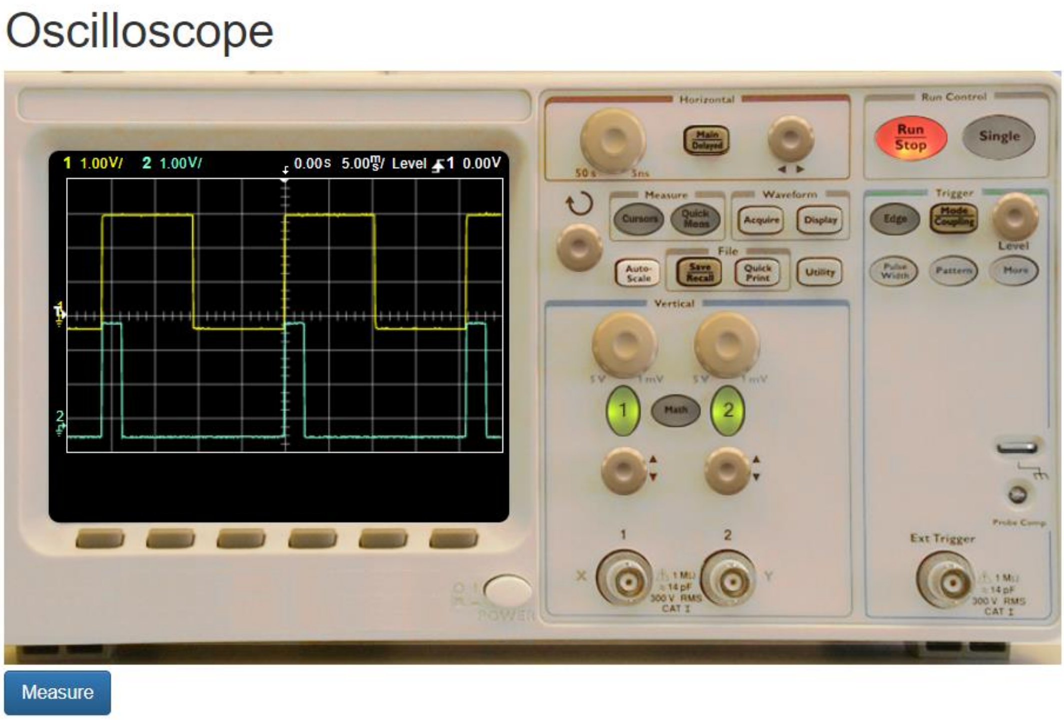
Scaling of 2ms/div



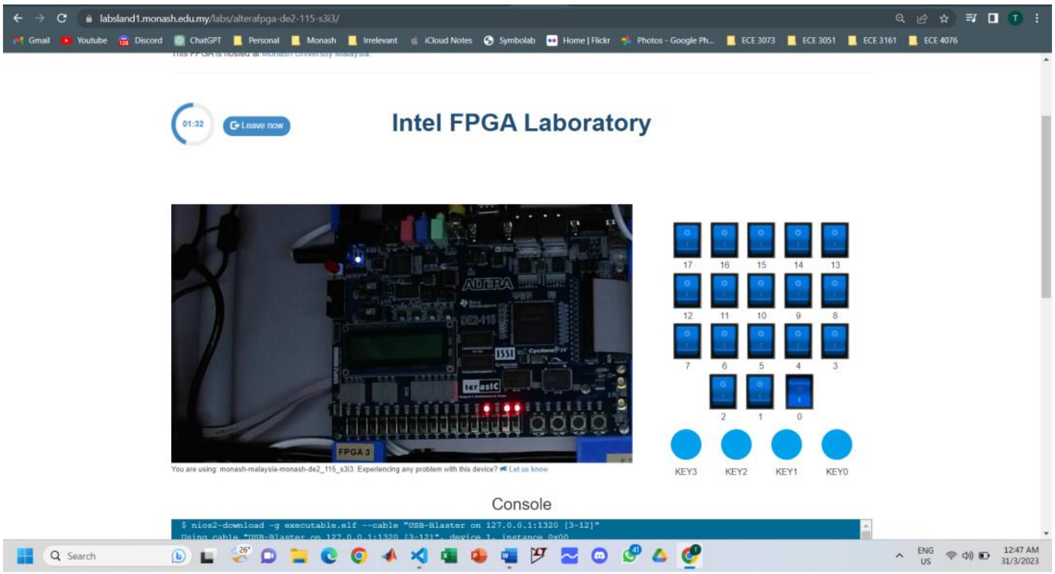


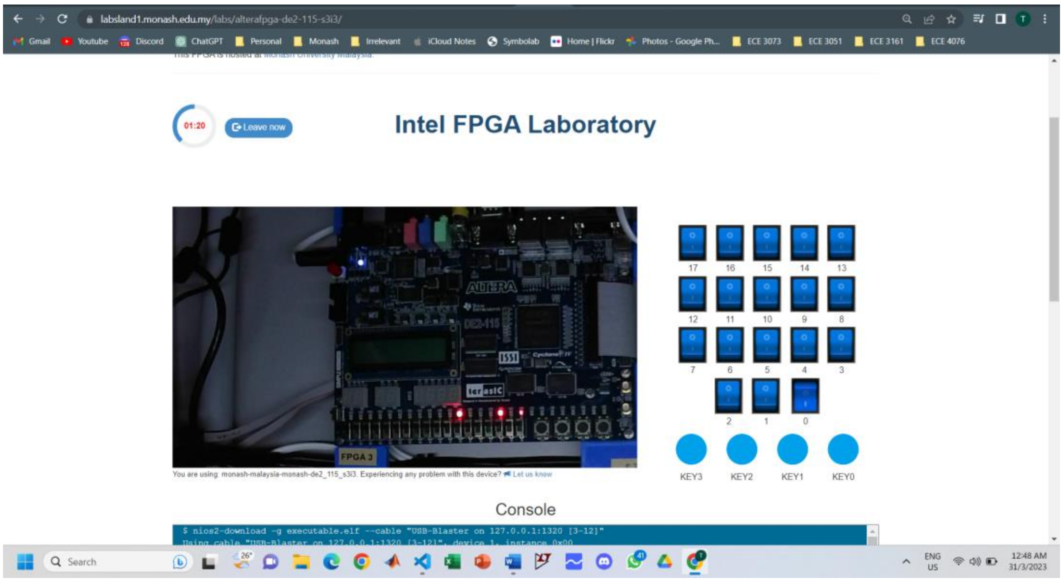
Scaling of 5ms/div





FPGA Screenshot

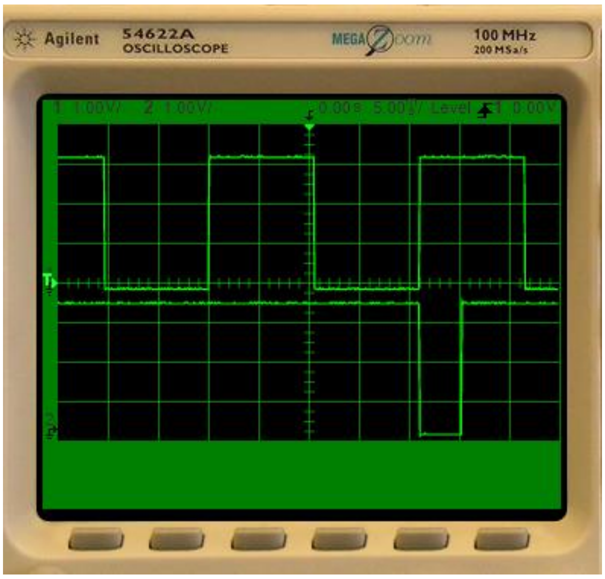


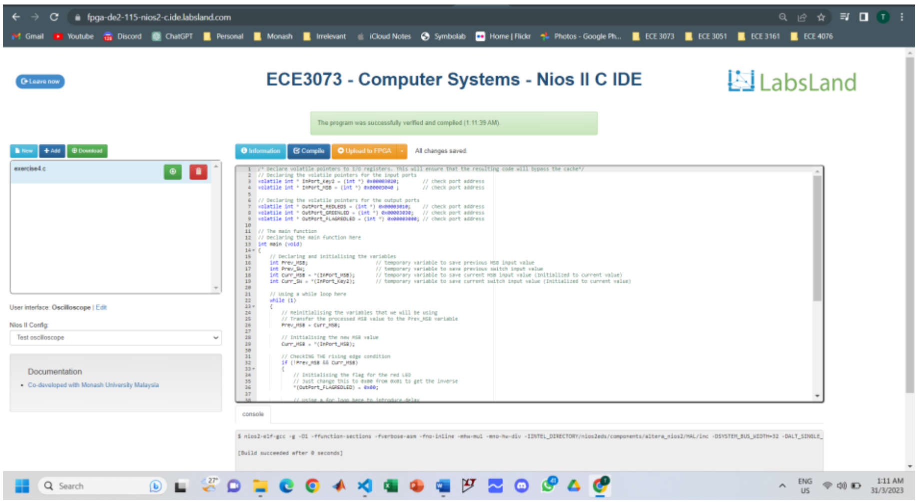


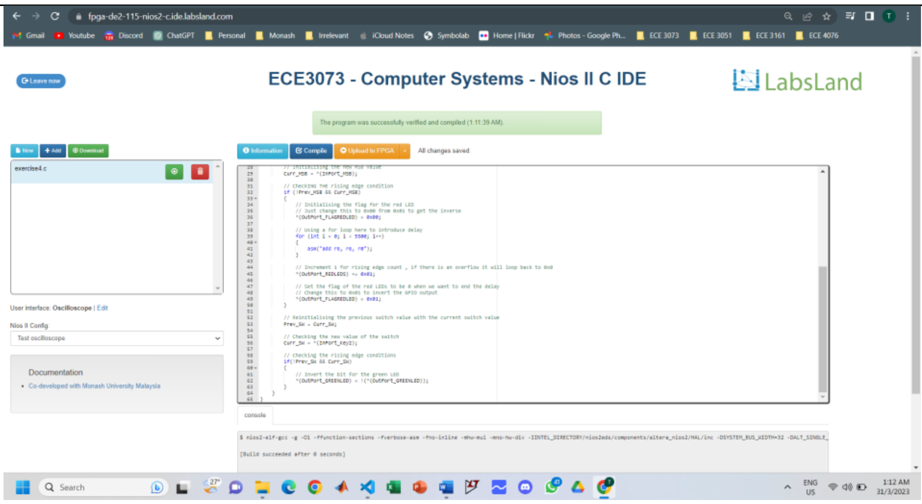
ECE3073 – Computer Systems
Lab 3C – Studying Latency and Instruction Execution Time
2023 – Sem 1.

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







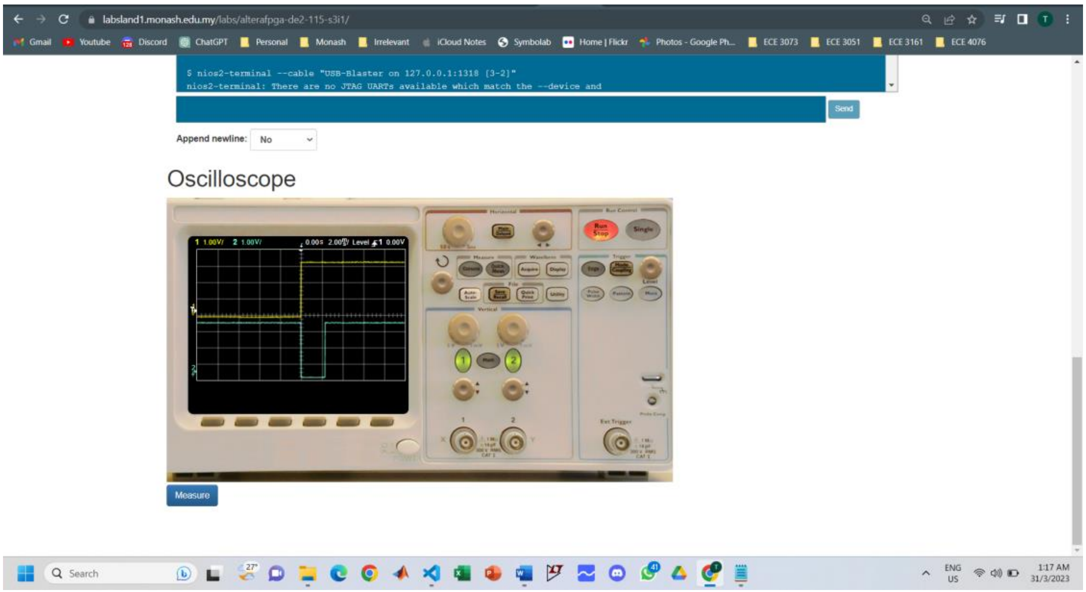
C Code

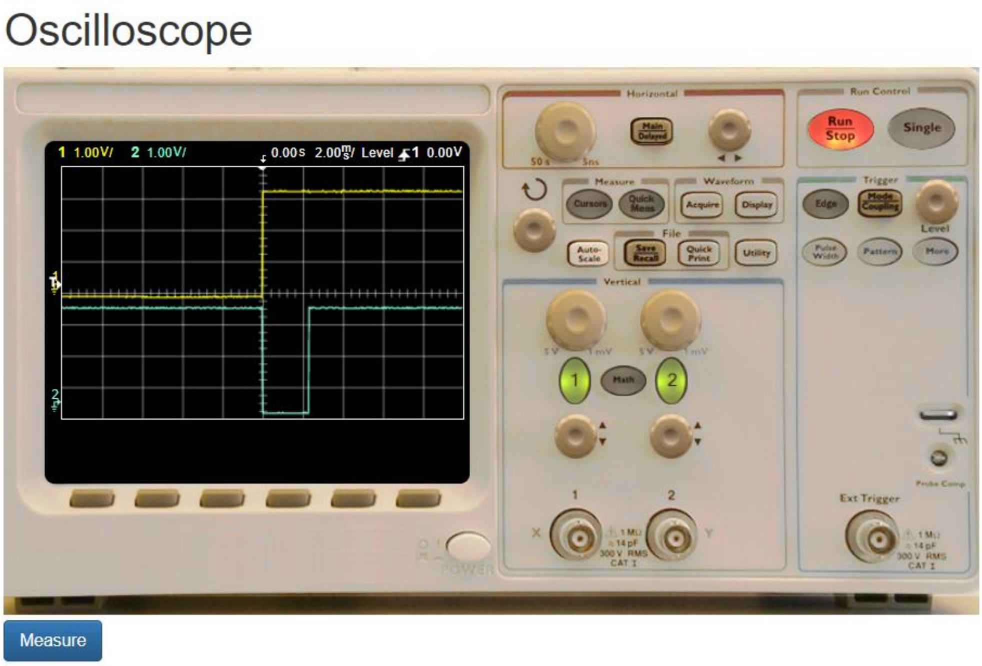
```
/* Declare volatile pointers to I/O registers. This will ensure that the resulting code will bypass the cache*/
// Declaring the volatile pointers for the input ports
volatile int * InPort_Key2 = (int *) 0x00003020;      // check port address
volatile int * InPort_MSB = (int *) 0x00003040 ;     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

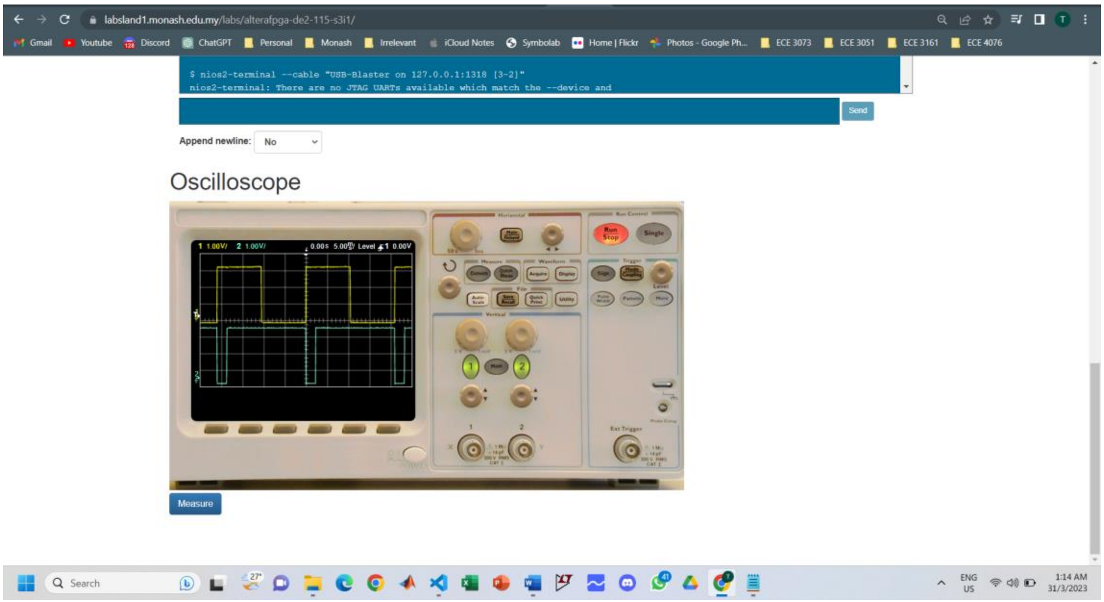
Oscilloscope

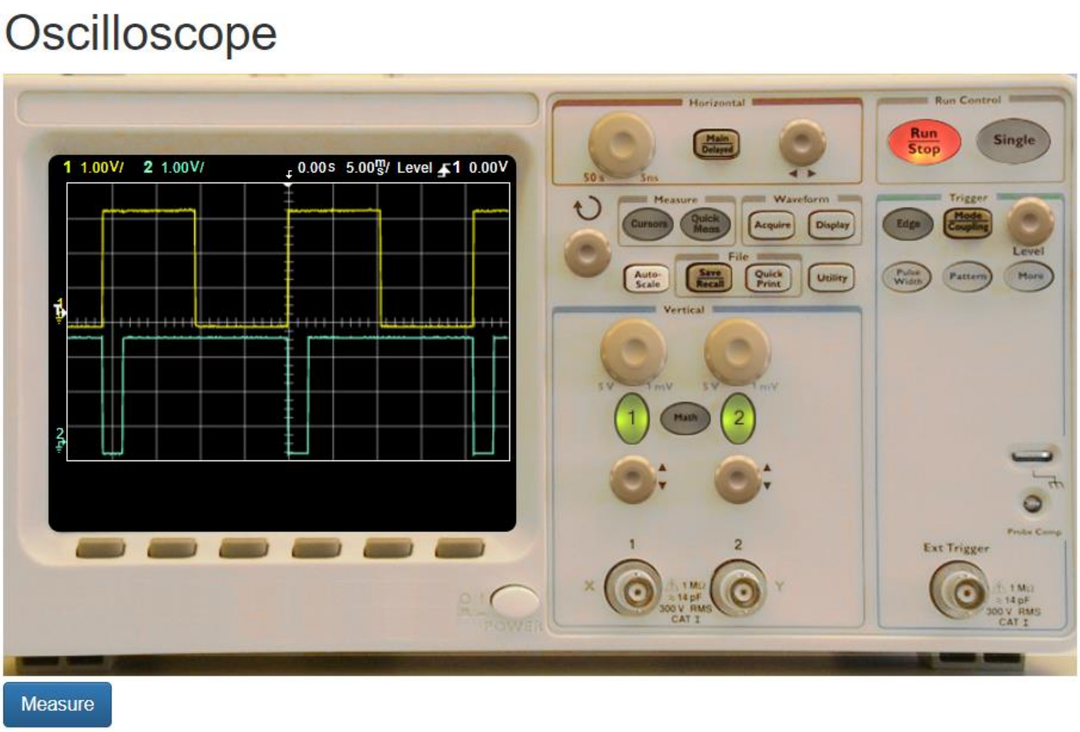
Scaling of 2ms/div



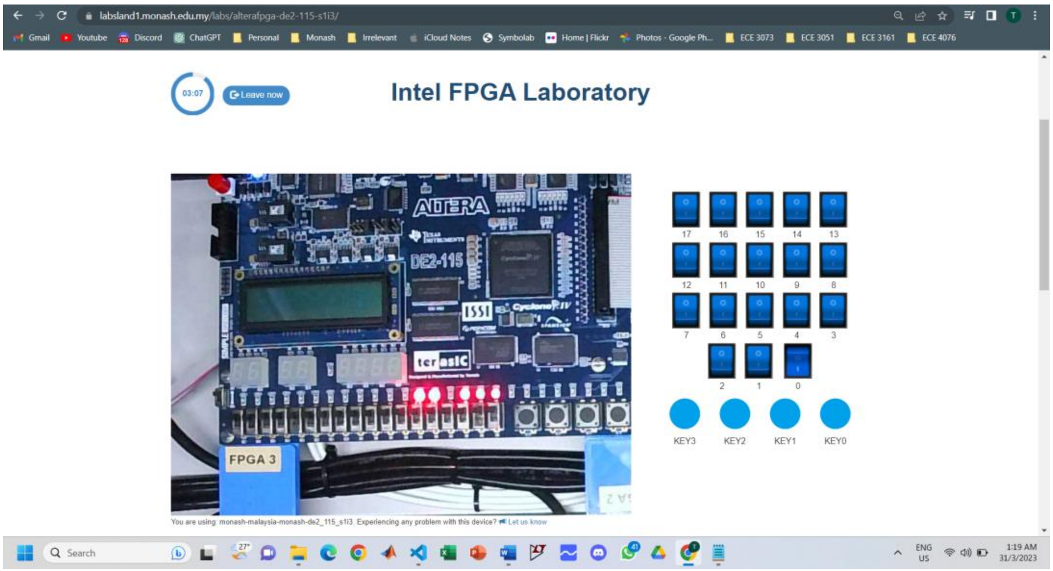


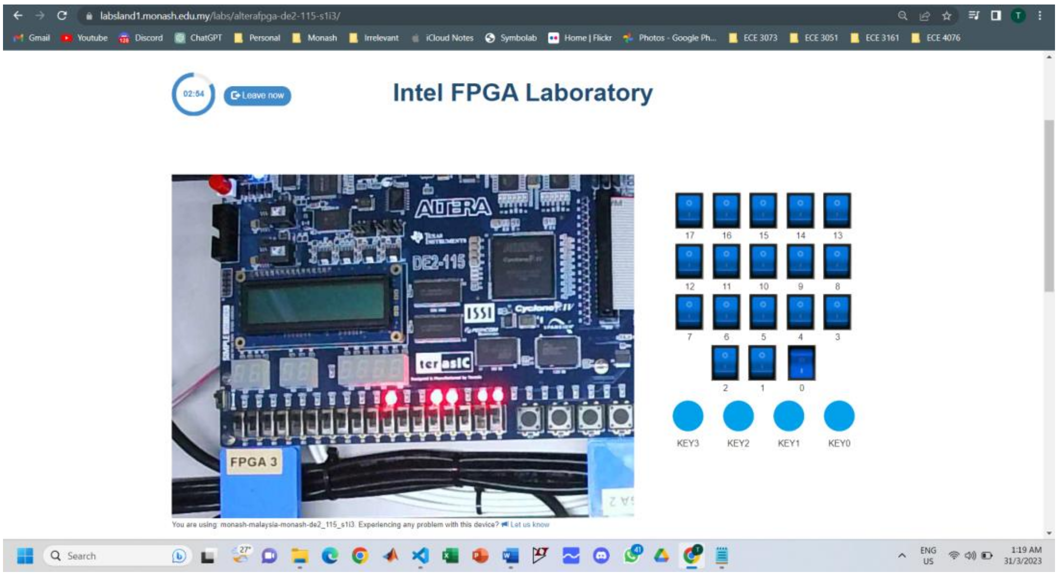
Scaling of 5ms/div





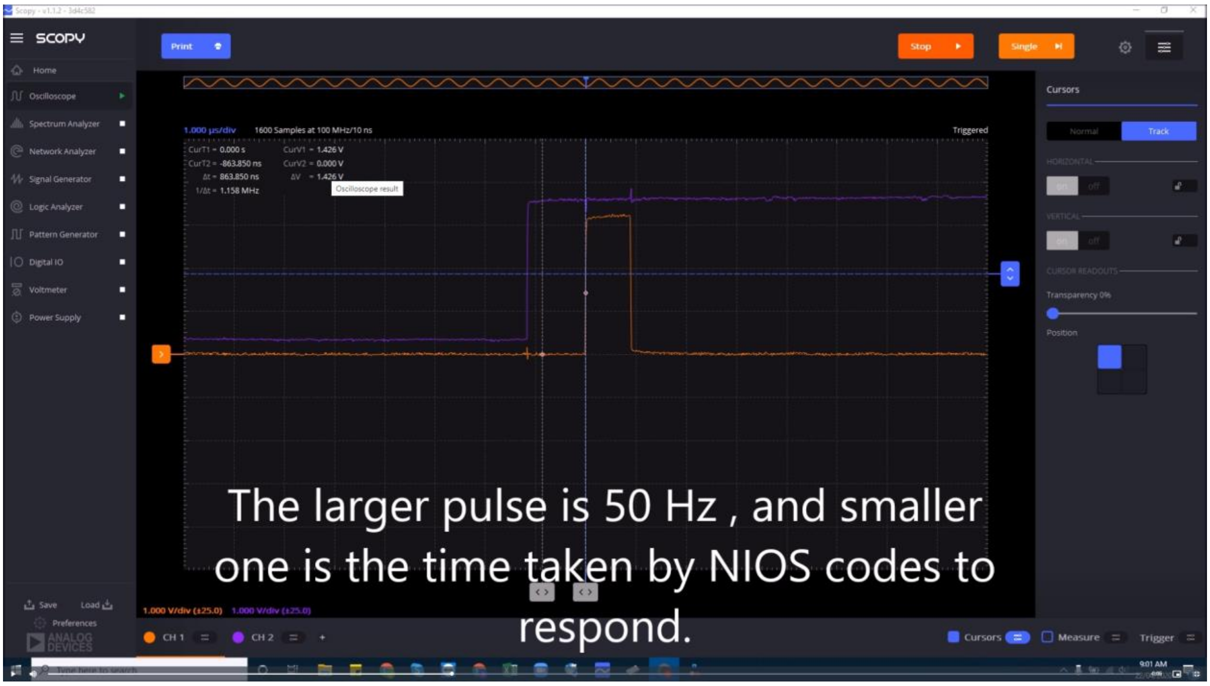
FPGA Screenshot





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

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 division * 1 microsecond per division which equates to about **1.15 microseconds**.

The answer is **1.15µs.**