ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Programming the Serial Interface

Week 9

Objectives

The laboratory exercise introduces asynchronous serial communications and will give you experience of writing C programs that can handle multiple sources of interrupts. You will also investigate the interaction between interrupt code and normal code. The Qsys is able to produce Verilog code (out of your schematics) for an RS232 asynchronous serial interface. Using Qsys you will design a Nios microprocessor system incorporating an RS232 interface. This system will then be programmed to send and receive serial data.

In this exercise you will:

• Develop a Nios processor system incorporating an RS232 serial interface

• Write C code to send and receive serial data using interrupts

Prelim (10 Marks)

(1) Write down the offset numbers of following UART registers(need to refer data sheet of NIOS UART)

Receiver register(Rx)	0
Transmitter(Tx)	1
Status	2
Control	3

(2) What value in control register need to be assigned if you want UART to trigger interrupt to NIOS processor whenever a character received in UART Rx register.

0b0000 0000 0000 0000 0000 0000 1000 0000 which equates to 0x80. This value is assigned as we set a value 1 to the bit “irrdy” (bit 7) in the control register which allows an interrupt to be triggered when the bit “rrdy” in the status register changes to 1.

(3) Where do you think you need to write the above setting code , main or exception ?

We need to write the above setting code in the main code. Before any interrupt can happen, the setup for the control register is always in the main body of the code.

(4) We need to clear interrupt bits of UART to ensure new interrupt can be trigger for new character at Rx register. Which register and what value we need to write for this setting

The UART status register. The value that we need to write for this setting is 0x00. The setting will reset the UART to the pre-interrupt state where all bits in the status register is zero.
A write operation to the status register clears the R, TOE, ROE, BRK, FE, DCTS and PE bits which allows the new character to be received at the Rx register normally.

(5) Which part of the code you will need to write the setting in Question 4 (Main or exception) can specify clearly

Exception Code. The interrupt is handled in the exception code and the clear setting can only be triggered in that section of code. After the character is read from the UART Rx register and before the UART is prepared to receive additional characters, the setting must be made in a specified location at the end of the exception code that handles UART character reception.

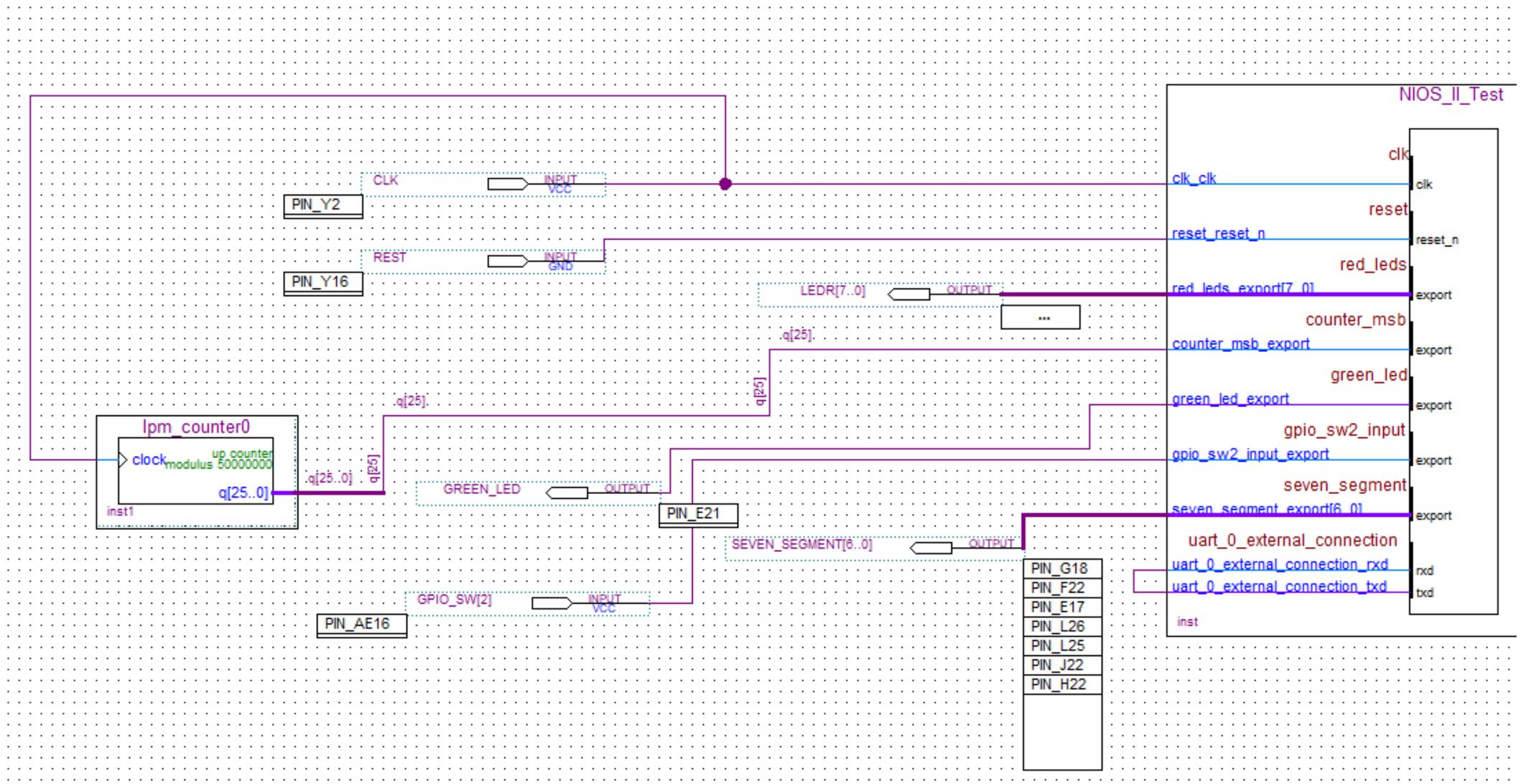
2. The computer system for this lab details

(a) Using Qsys to create a Nios system with the following components:

- a Nios II/e processor,
- 16384 bytes of RAM
- one 1-bit PIO input port (to read SW[2] of labsland – same as lab 4)
- one 1-bit PIO input port configured to produce an interrupt on an input rising edge ((MSB) of 1 Hz counter)
- 7 - bit PIO input port configured to first seven segment display (refer to pin numbers in screenshot)
- an RS232 UART configured for odd parity, 8 data bits, 1 stop bit and a fixed Baud rate of 9600
- a JTAG UART Note: you will find the UARTs in the left hand window of Qsys if you expand Interface protocols and then Serial.

Click on the bubbles in the IRQ column of Qsys to connect interrupts to counter PIO together with the RS232 and JTAG UARTs. You should make a note of the interrupt number assigned to each device as well as all of the I/O base addresses. (refer to the QYSYS screen shot)

(b) Following are the screen shot of the schematics



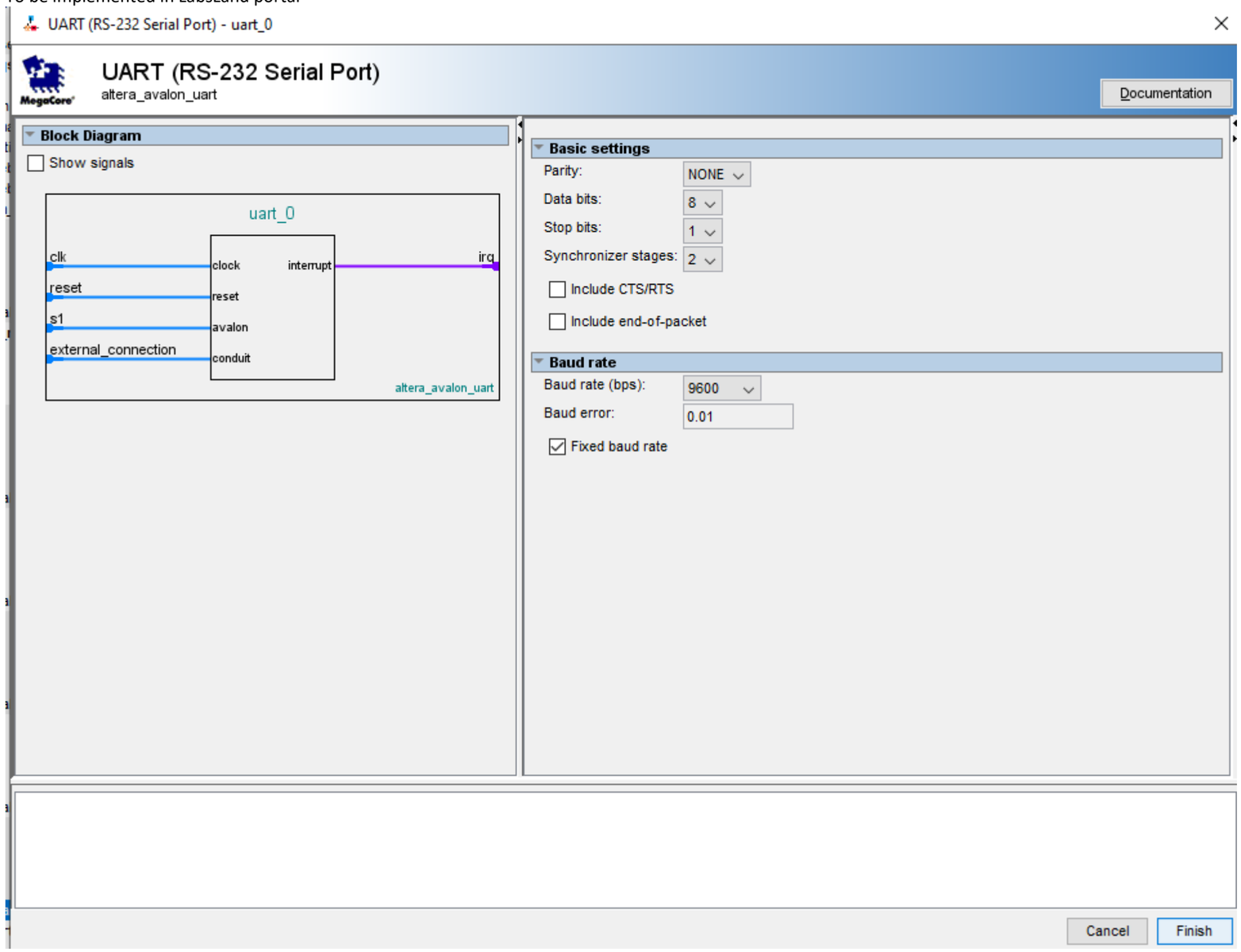
(Note the green Led just there but not used in LAB5)

ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Opc
☑		**clk_0**	Clock Source						
		clk_in	Clock Input	**clk**					
		clk_in_reset	Reset Input	**reset**					
		clk	Clock Output	*Double-click to export*	clk_0				
		clk_reset	Reset Output	*Double-click to export*					
☑		**nios2_qsys_0**	Nios II Processor						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset_n	Reset Input	*Double-click to export*	[clk]				
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			IRQ 31	
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
		jtag_debug_module_re...	Reset Output	*Double-click to export*	[clk]				
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x4800**	0x4fff		
		custom_instruction_m...	Custom Instruction Master	*Double-click to export*					
☑		**pio_0**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5030**	0x503f		
		external_connection	Conduit	**red_leds**					
☑		**onchip_memory2_0**	On-Chip Memory (RAM or ROM)						
		clk1	Clock Input	*Double-click to export*	**clk_0**				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0000	0x3fff		
		reset1	Reset Input	*Double-click to export*	[clk1]				
☑		**pio_1**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5060**	0x506f	0	
		external_connection	Conduit	**counter_msb**					
☑		**pio_2**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5050**	0x505f		
		external_connection	Conduit	**green_led**					
☑		**pio_3**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5040**	0x504f		
		external_connection	Conduit	**gpio_sw2_input**					
☑		**pio_4**	PIO (Parallel I/O)						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5020**	0x502f		
		external_connection	Conduit	**seven_segment**					
☑		**uart_0**	UART (RS-232 Serial Port)						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5000**	0x501f	1	
		external_connection	Conduit	**uart_0_external_connecti...**					
☑		**jtag_uart_0**	JTAG UART						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x5070**	0x5077	2	

(c) The QYSYS

ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal



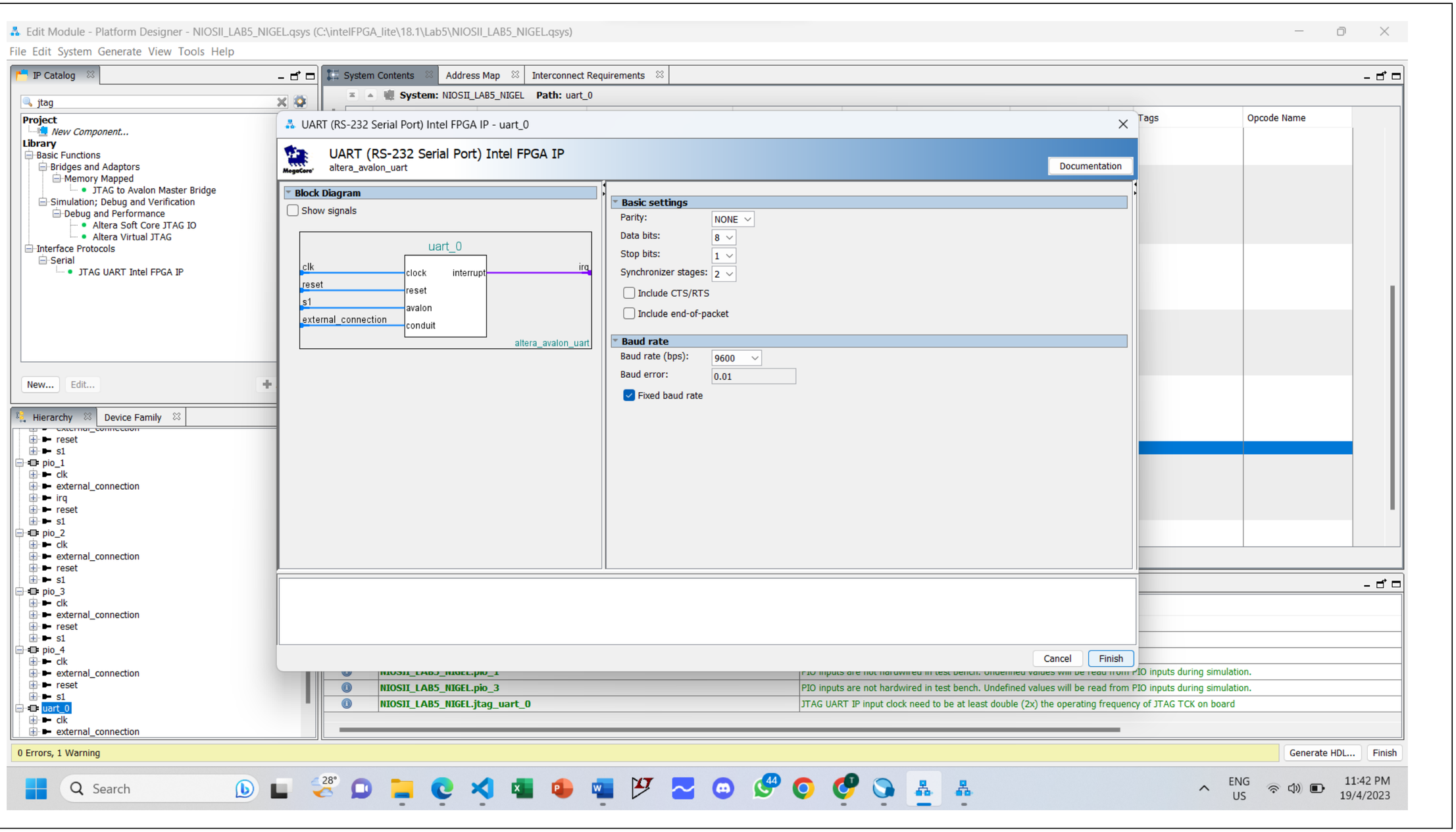
UART Settings as above.

NOTE: Even though we include JTAG UART here, we do not use . This can be used if you want to print statement in console window, unfortunately we could not do that yet in LabsLand. So just include JTAG UART and ignore while program, you will work only with the above UART-RS232 (screenshot)

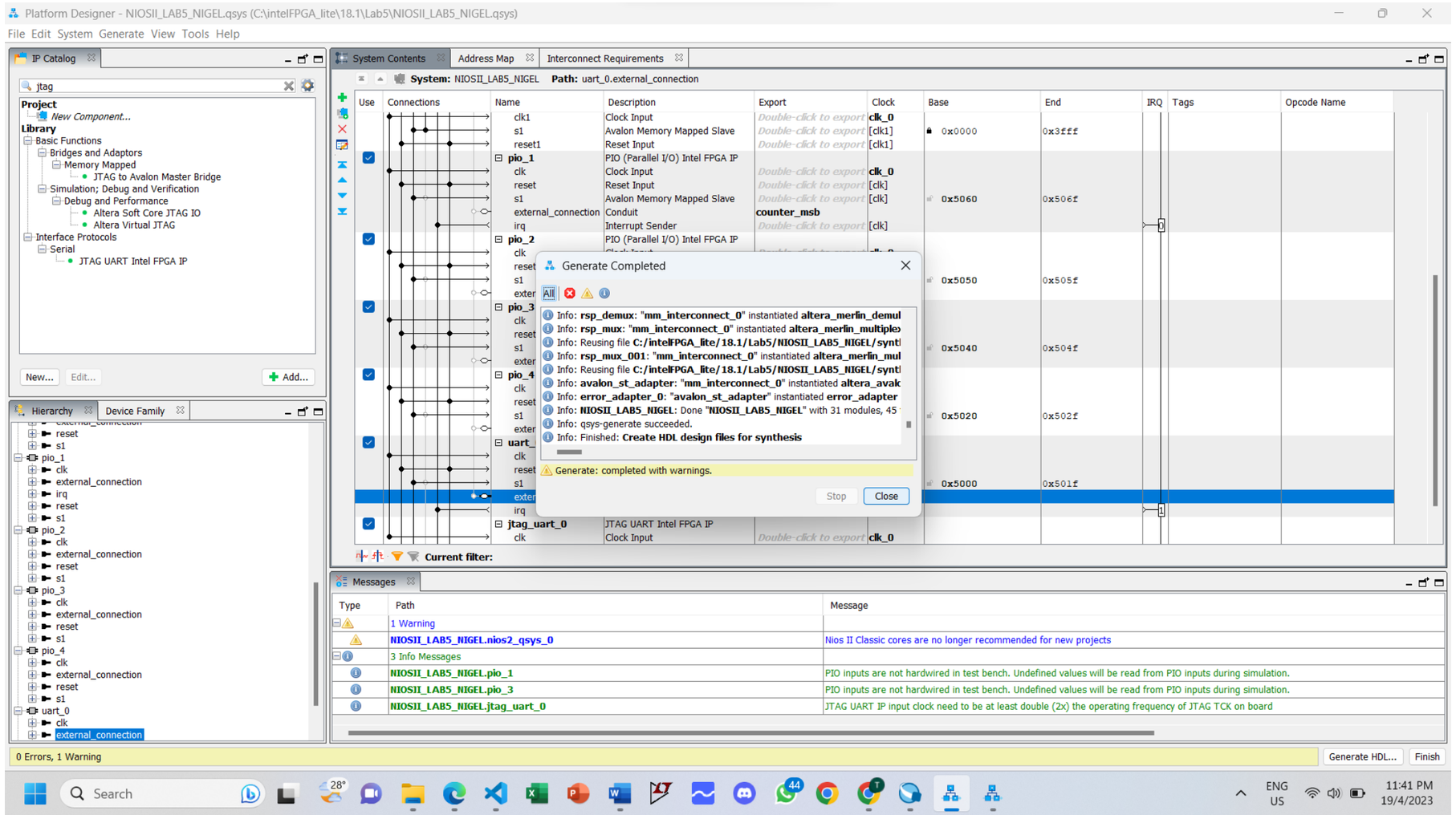
ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Exercise (1) (5 marks)

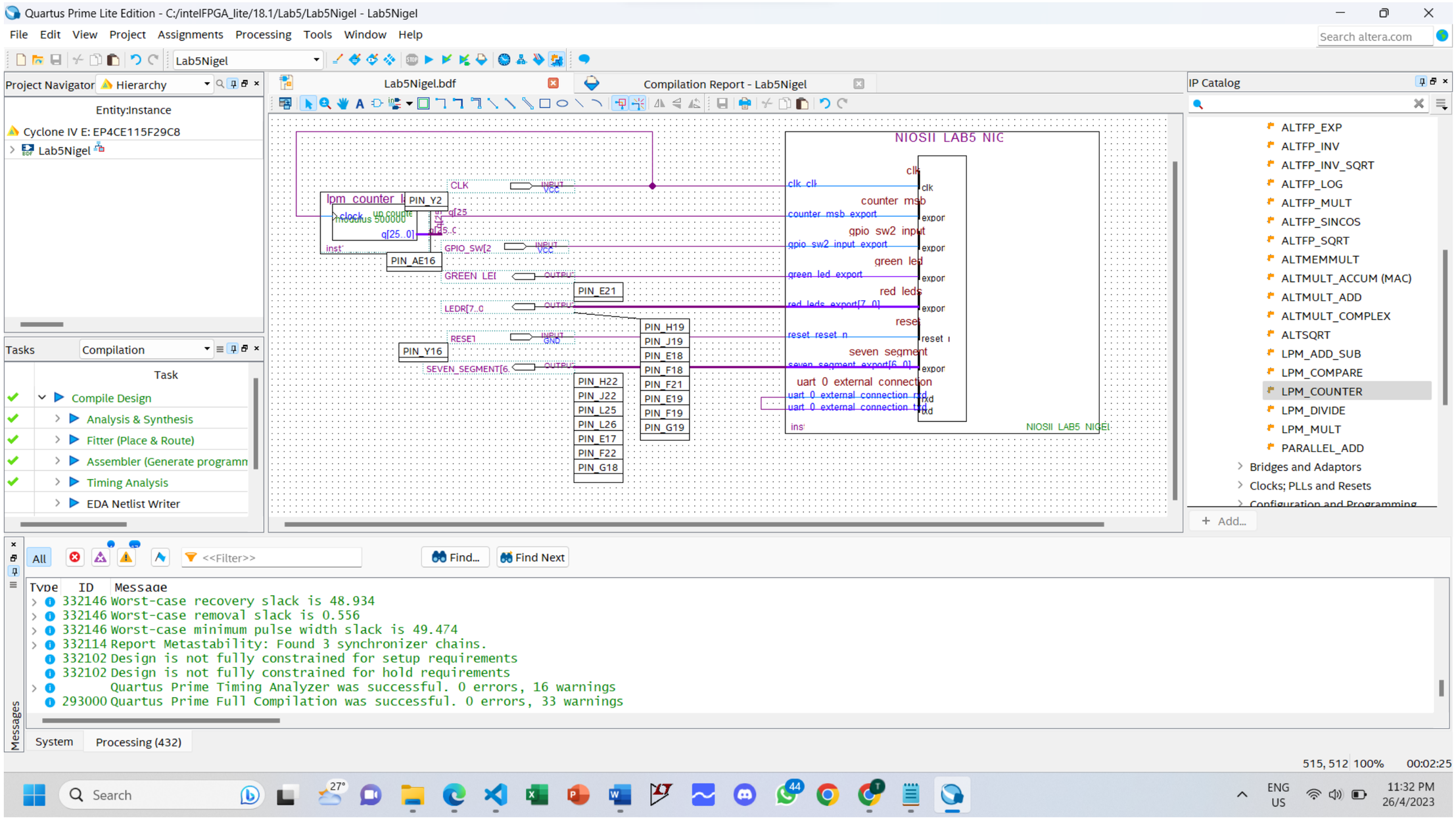
Complete the Lab5 computer system schematics and qysys without errors in your computers , and Paste screenshots here (its same as above- you could develop from Lab4)



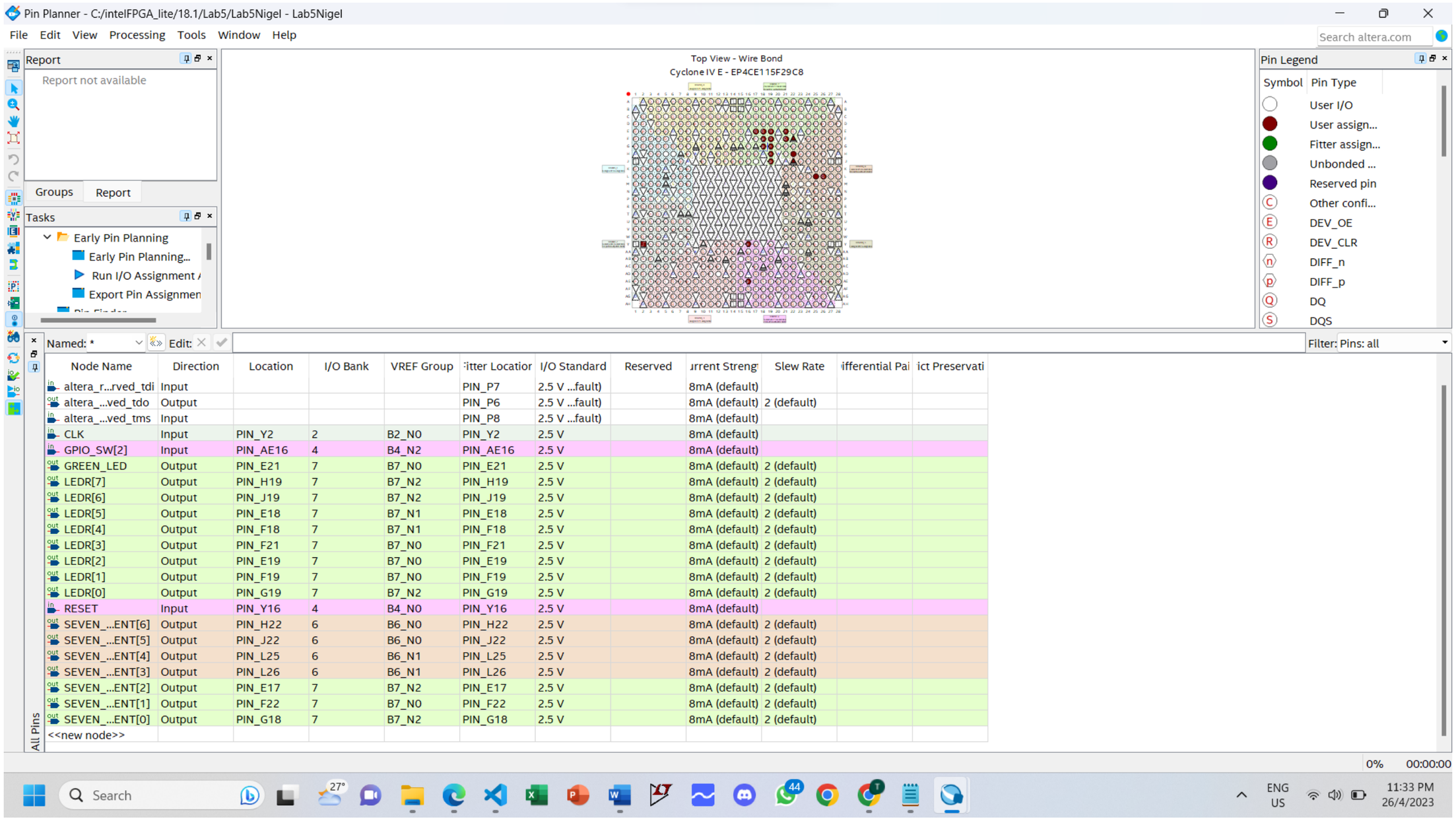
ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal



ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal



ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Bus Hold
altera_r...rved_tdi	Input				PIN_P7	2.5 V ...fault)		8mA (default)			
altera_...ved_tdo	Output				PIN_P6	2.5 V ...fault)		8mA (default)	2 (default)		
altera_...ved_tms	Input				PIN_P8	2.5 V ...fault)		8mA (default)			
CLK	Input	PIN_Y2	2	B2_N0	PIN_Y2	2.5 V		8mA (default)			
GPIO_SW[2]	Input	PIN_AE16	4	B4_N2	PIN_AE16	2.5 V		8mA (default)			
GREEN_LED	Output	PIN_E21	7	B7_N0	PIN_E21	2.5 V		8mA (default)	2 (default)		
LEDR[7]	Output	PIN_H19	7	B7_N2	PIN_H19	2.5 V		8mA (default)	2 (default)		
LEDR[6]	Output	PIN_J19	7	B7_N2	PIN_J19	2.5 V		8mA (default)	2 (default)		
LEDR[5]	Output	PIN_E18	7	B7_N1	PIN_E18	2.5 V		8mA (default)	2 (default)		
LEDR[4]	Output	PIN_F18	7	B7_N1	PIN_F18	2.5 V		8mA (default)	2 (default)		
LEDR[3]	Output	PIN_F21	7	B7_N0	PIN_F21	2.5 V		8mA (default)	2 (default)		
LEDR[2]	Output	PIN_E19	7	B7_N0	PIN_E19	2.5 V		8mA (default)	2 (default)		
LEDR[1]	Output	PIN_F19	7	B7_N0	PIN_F19	2.5 V		8mA (default)	2 (default)		
LEDR[0]	Output	PIN_G19	7	B7_N2	PIN_G19	2.5 V		8mA (default)	2 (default)		
RESET	Input	PIN_Y16	4	B4_N0	PIN_Y16	2.5 V		8mA (default)			
SEVEN_...ENT[6]	Output	PIN_H22	6	B6_N0	PIN_H22	2.5 V		8mA (default)	2 (default)		
SEVEN_...ENT[5]	Output	PIN_J22	6	B6_N0	PIN_J22	2.5 V		8mA (default)	2 (default)		
SEVEN_...ENT[4]	Output	PIN_L25	6	B6_N1	PIN_L25	2.5 V		8mA (default)	2 (default)		
SEVEN_...ENT[3]	Output	PIN_L26	6	B6_N1	PIN_L26	2.5 V		8mA (default)	2 (default)		
SEVEN_...ENT[2]	Output	PIN_E17	7	B7_N2	PIN_E17	2.5 V		8mA (default)	2 (default)		
SEVEN_...ENT[1]	Output	PIN_F22	7	B7_N0	PIN_F22	2.5 V		8mA (default)	2 (default)		
SEVEN_...ENT[0]	Output	PIN_G18	7	B7_N2	PIN_G18	2.5 V		8mA (default)	2 (default)		
<<new node>>											

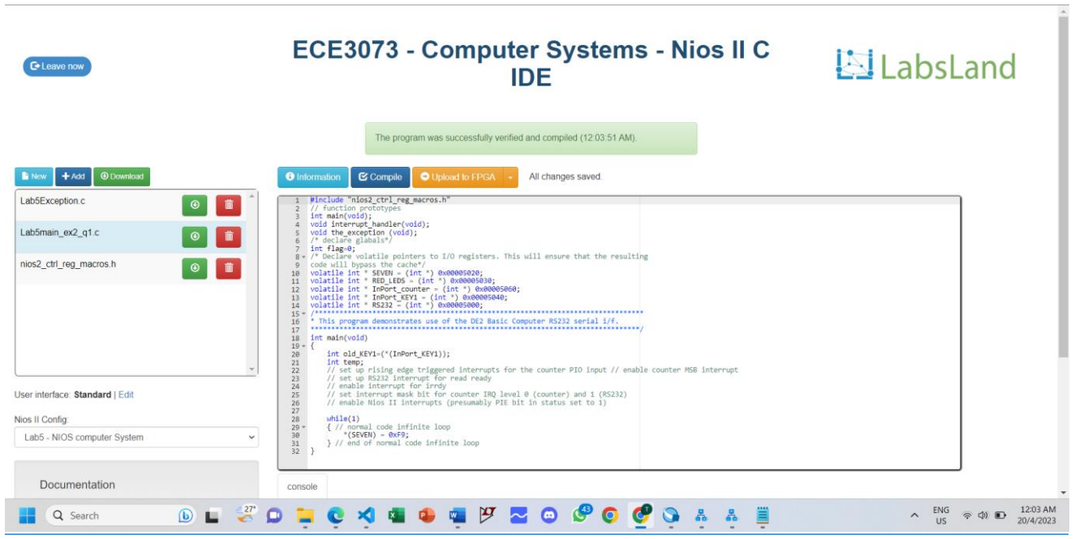
ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Exercise (2) (10 Marks)

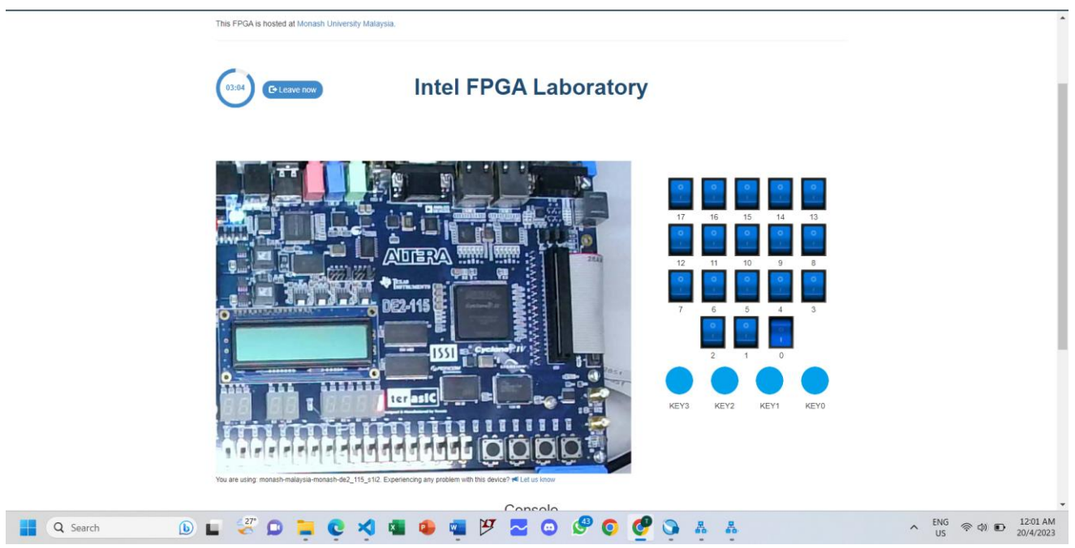
Write simple code to display “1” and then “2” in seven segment display. Paste your screen shot here (basically 2 screen shots)

Displaying “1”

[Compiled Code on Labsland]

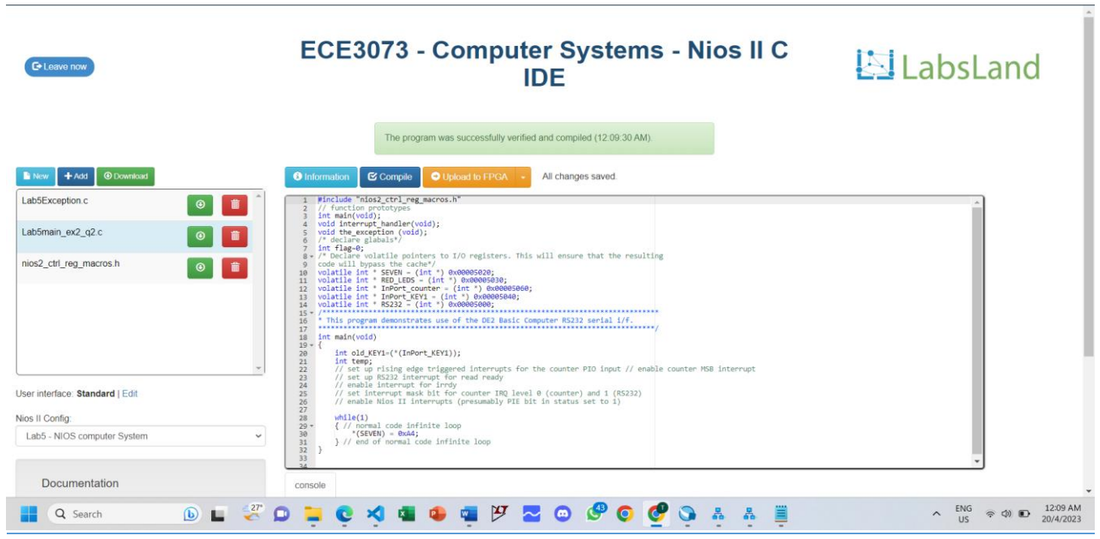


[FPGA Output]

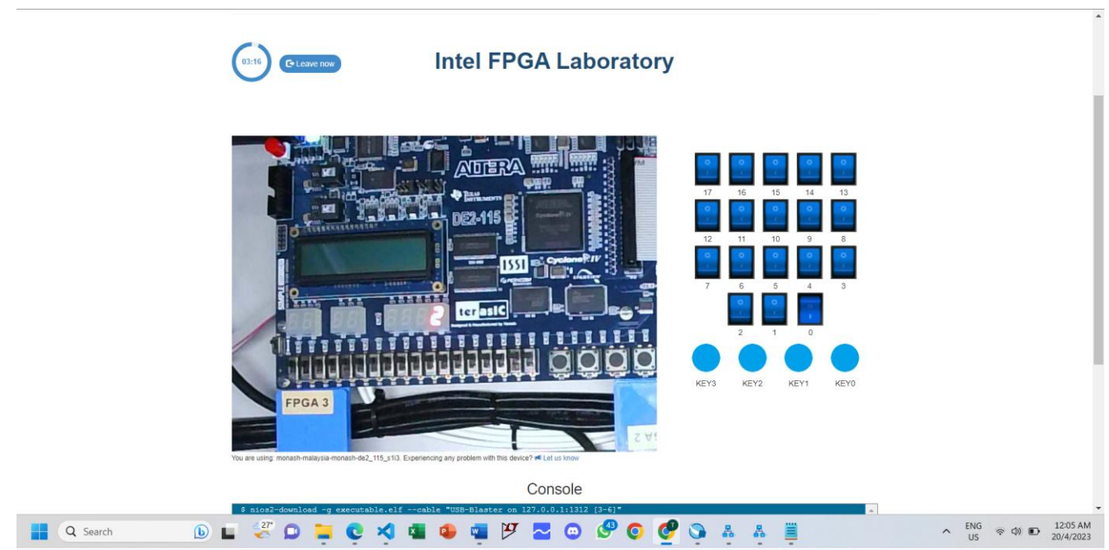


Displaying “2”

[Compiled Code on Labsland]



[FPGA Output]



Code

```
#include "nios2_ctrl_reg_macros.h"
// function prototypes
int main(void);
void interrupt_handler(void);
void the_exception (void);
/* declare glabals*/
int flag=0;
/* Declare volatile pointers to I/O registers. This will ensure that the resulting
code will bypass the cache*/
volatile int * SEVEN = (int *) 0x00005020;
volatile int * RED_LEDS = (int *) 0x00005030;
volatile int * InPort_counter = (int *) 0x00005060;
volatile int * InPort_KEY1 = (int *) 0x00005040;
volatile int * RS232 = (int *) 0x00005000;
/*****************************************************************************
* This program demonstrates use of the DE2 Basic Computer RS232 serial i/f.
*****************************************************************************/
int main(void)
{
   int old_KEY1=(*(InPort_KEY1));
   int temp;
   // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt
   // set up RS232 interrupt for read ready
   // enable interrupt for irrdy
   // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)
   // enable Nios II interrupts (presumably PIE bit in status set to 1)

   while(1)
   { // normal code infinite loop
      *(SEVEN) = 0xF9;
   } // end of normal code infinite loop
}
```

Code

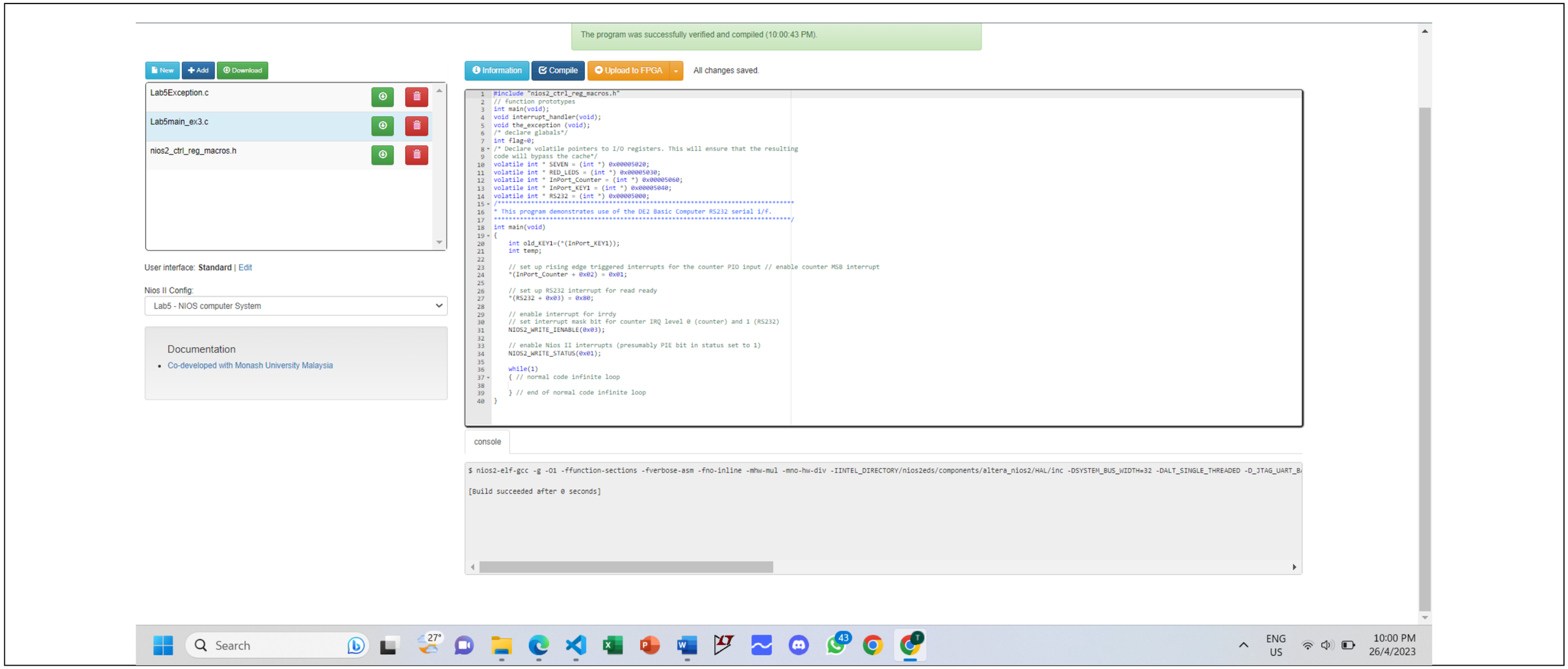
```
#include "nios2_ctrl_reg_macros.h"
// function prototypes
int main(void);
void interrupt_handler(void);
void the_exception (void);
/* declare glabals*/
int flag=0;
/* Declare volatile pointers to I/O registers. This will ensure that the resulting
code will bypass the cache*/
volatile int * SEVEN = (int *) 0x00005020;
volatile int * RED_LEDS = (int *) 0x00005030;
volatile int * InPort_counter = (int *) 0x00005060;
volatile int * InPort_KEY1 = (int *) 0x00005040;
volatile int * RS232 = (int *) 0x00005000;
/*****************************************************************************
* This program demonstrates use of the DE2 Basic Computer RS232 serial i/f.
*****************************************************************************/
int main(void)
{
   int old_KEY1=(*(InPort_KEY1));
   int temp;
   // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt
   // set up RS232 interrupt for read ready
   // enable interrupt for irrdy
   // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)
   // enable Nios II interrupts (presumably PIE bit in status set to 1)

   while(1)
   { // normal code infinite loop
      *(SEVEN) = 0xA4;
   } // end of normal code infinite loop
}
```

ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Exercise (3) (35 Marks)

Write suitable C – code to transmit your student ID (which are numbers but you will transmit them as characters, in the sample output video I have used alphabets for illustration purpose) serially one by one character to NIOS UART Transmitter register for every rising edge of 1 Hz MSB counter (trigger interrupts), as the transmitter and receiver are shorted , display the received characters (trigger interrupt) to 7 – segment display. Using LabsLand portal :Record your output using screen recording tool with your name and student ID typed in notepad are captured . You may refer the recorded sample video that I used to transmit a string :" ABCD". You need to post your screen capture video in the moodle submission link (20 marks) – Refer to **sample solution for Exercise 3** video in moodle.

Paste the screen shot of your main code from labsland (5 Marks)

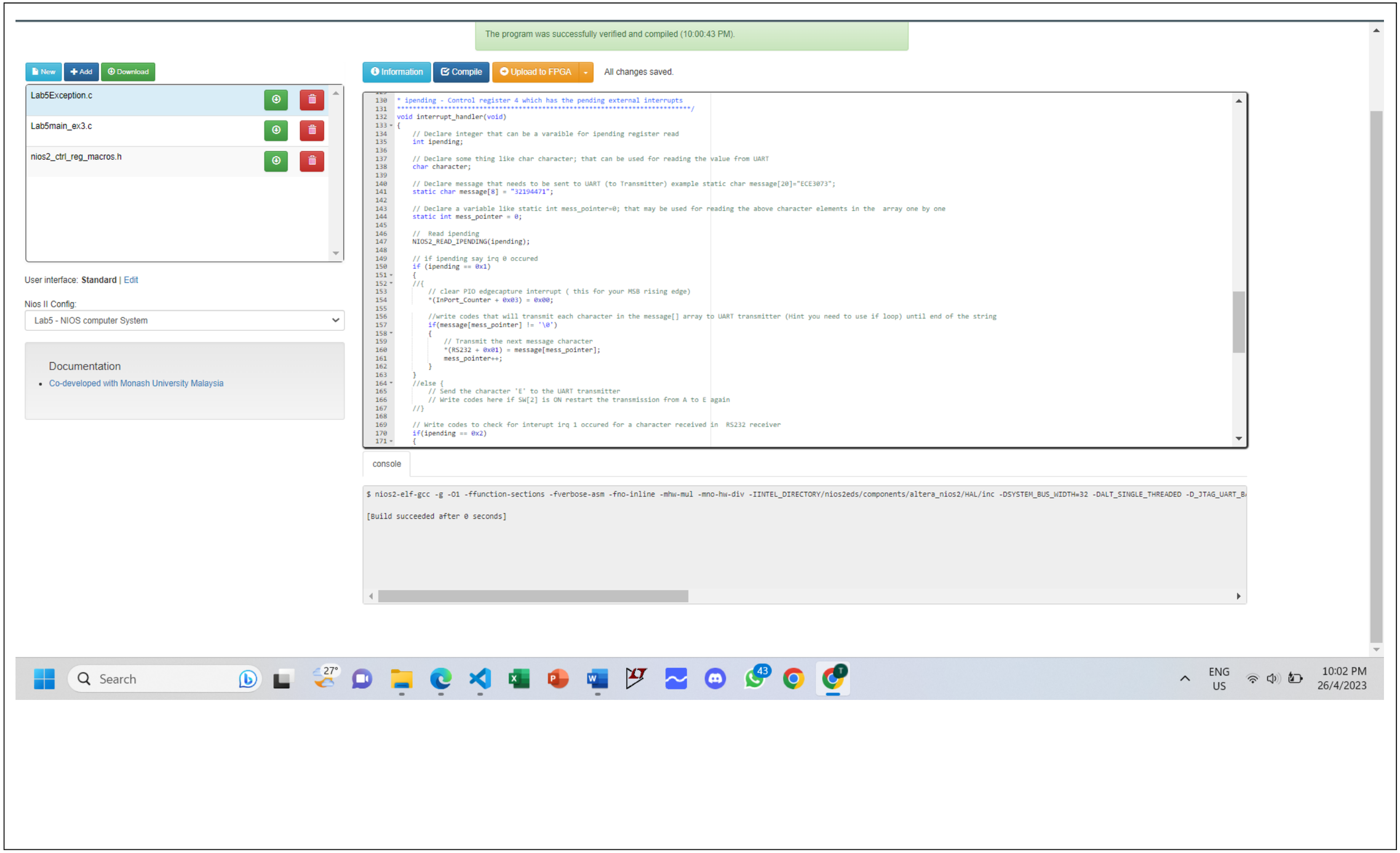


Actual Code

```
#include "nios2_ctrl_reg_macros.h"
// function prototypes
int main(void);
void interrupt_handler(void);
void the_exception (void);
/* declare glabals*/
int flag=0;
/* Declare volatile pointers to I/O registers. This will ensure that the resulting
code will bypass the cache*/
volatile int * SEVEN = (int *) 0x00005020;
volatile int * RED_LEDS = (int *) 0x00005030;
volatile int * InPort_Counter = (int *) 0x00005060;
volatile int * InPort_KEY1 = (int *) 0x00005040;
volatile int * RS232 = (int *) 0x00005000;
/*****************************************************************************
* This program demonstrates use of the DE2 Basic Computer RS232 serial i/f.
*****************************************************************************/
int main(void)
{
   int old_KEY1=(*(InPort_KEY1));
   int temp;

   // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt
   *(InPort_Counter + 0x02) = 0x01;

   // set up RS232 interrupt for read ready
   *(RS232 + 0x03) = 0x80;

   // enable interrupt for irrdy
   // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)
   NIOS2_WRITE_IENABLE(0x03);

   // enable Nios II interrupts (presumably PIE bit in status set to 1)
   NIOS2_WRITE_STATUS(0x01);

   while(1)
   { // normal code infinite loop

   } // end of normal code infinite loop
}
```

ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Paste the screen shot of interrupt handler code from labsland (10 marks) – Note that I need to see the codes only for the above function explained in the exercise (3)



ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

The program was successfully verified and compiled (10:00:43 PM).

New | Add | Download

Lab5Exception.c
Lab5main_ex3.c
nios2_ctrl_reg_macros.h

User interface: **Standard** | Edit

Nios II Config:
Lab5 - NIOS computer System

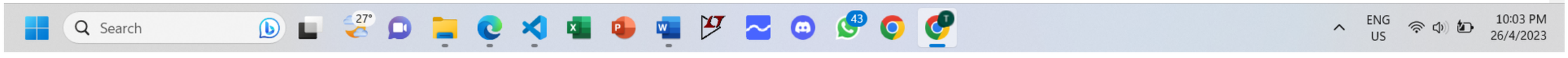
Documentation
- Co-developed with Monash University Malaysia

Information | Compile | Upload to FPGA | All changes saved.

```
167     //}
168
169     // Write codes to check for interupt irq 1 occured for a character received in  RS232 receiver
170     if(ipending == 0x2)
171     {
172         //  Write code to read the value to character variable
173         character = *(RS232);
174
175         // Write code to display received value in SEVEN SEGMENT DISPLAY , you need to know HEX display truthtable, ascii values of message chars.
176         switch (character)
177         {
178             case '0':
179                 *(SEVEN) = 0xC0;
180                 break;
181             case '1':
182                 *(SEVEN) = 0xF9;
183                 break;
184             case '2':
185                 *(SEVEN) = 0xA4;
186                 break;
187             case '3':
188                 *(SEVEN) = 0xB0;
189                 break;
190             case '4':
191                 *(SEVEN) = 0x99;
192                 break;
193             case '5':
194                 *(SEVEN) = 0x92;
195                 break;
196             case '6':
197                 *(SEVEN) = 0x82;
198                 break;
199             case '7':
200                 *(SEVEN) = 0xF8;
201                 break;
202             case '8':
203                 *(SEVEN) = 0x80;
204                 break;
205             case '9':
206                 *(SEVEN) = 0x90;
207                 break;
208             case 'E':
                *(SEVEN) = 0x86;
```

console

```
$ nios2-elf-gcc -g -O1 -ffunction-sections -fverbose-asm -fno-inline -mhw-mul -mno-hw-div -IINTEL_DIRECTORY/nios2eds/components/altera_nios2/HAL/inc -DSYSTEM_BUS_WIDTH=32 -DALT_SINGLE_THREADED -D_JTAG_UART_B

[Build succeeded after 0 seconds]
```



ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

The program was successfully verified and compiled (10:00:43 PM).

New | Add | Download

Lab5Exception.c
Lab5main_ex3.c
nios2_ctrl_reg_macros.h

User interface: **Standard** | Edit

Nios II Config:
Lab5 - NIOS computer System

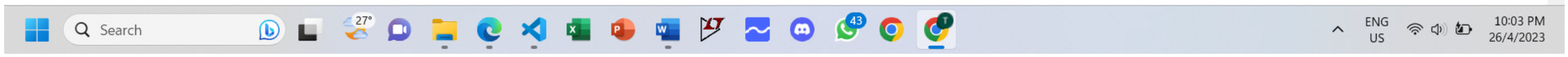
Documentation
- Co-developed with Monash University Malaysia

Information | Compile | Upload to FPGA | All changes saved.

```
            case '2':
                *(SEVEN) = 0xA4;
                break;
            case '3':
                *(SEVEN) = 0x80;
                break;
            case '4':
                *(SEVEN) = 0x99;
                break;
            case '5':
                *(SEVEN) = 0x92;
                break;
            case '6':
                *(SEVEN) = 0x82;
                break;
            case '7':
                *(SEVEN) = 0xF8;
                break;
            case '8':
                *(SEVEN) = 0x80;
                break;
            case '9':
                *(SEVEN) = 0x90;
                break;
            case 'E':
                *(SEVEN) = 0x86;
                break;
            default:
                *(SEVEN) = 0xFF;
                break;
        }

        // Simultaneously display the count of the character received at UART receiver
        LED_Count++;
        *(RED_LEDS) = LED_Count;

        // Write code for clear RS232 interrupt bits
        *(RS232 + 0x02) = 0x00;
    }

    // else, ignore the interrupt
    return;
}
```

console

```
$ nios2-elf-gcc -g -O1 -ffunction-sections -fverbose-asm -fno-inline -mhw-mul -mno-hw-div -IINTEL_DIRECTORY/nios2eds/components/altera_nios2/HAL/inc -DSYSTEM_BUS_WIDTH=32 -DALT_SINGLE_THREADED -D_JTAG_UART_B

[Build succeeded after 0 seconds]
```



Actual Code

```
#include "nios2_ctrl_reg_macros.h"
/* function prototypes */
void main(void);
void interrupt_handler(void);
void the_exception(void);
/* global variables */
/// Declare all extern int like lab 4
extern int flag;
extern int * SEVEN;
extern int * RED_LEDS;
extern int * InPort_Counter;
extern int * InPort_KEY1;
extern int * RS232;

//Declare any global value suchas count that can be used for turning 8-RED LEDs
int LED_Count = 0;

/* The assembly language code below handles CPU reset processing */
void the_reset (void) __attribute__ ((section (".reset")));
void the_reset (void)
/*****************************************************************************
* Reset code. By giving the code a section attribute with the name ".reset"*
* we allow the linker program to locate this code at the proper reset *
* vector address. This code just calls the main program. *
*****************************************************************************/
{
asm (".set noat"); // Magic, for the C compiler
asm (".set nobreak"); // Magic, for the C compiler
asm ("movia r2, main"); // Call the C language main program
asm ("jmp r2");
}
/* The assembly language code below handles CPU exception processing. This
* code should not be modified; instead, the C language code in the function
* interrupt_handler() can be modified as needed for a given application.
*/
void the_exception (void) __attribute__ ((section (".exceptions")));
void the_exception (void)
/*****************************************************************************
* Exceptions code. By giving the code a section attribute with the name *
* ".exceptions" we allow the linker program to locate this code at the *
* proper exceptions vector address. *
* This code calls the interrupt handler and later returns from the *
* exception. *
*****************************************************************************/
{
asm ( ".set noat" ); // Magic, for the C compiler
asm ( ".set nobreak" ); // Magic, for the C compiler
asm ( "subi sp, sp, 128" );
asm ( "stw et, 96(sp)" );
asm ( "rdctl et, ctl4" );
```

```
asm ( "beq et, r0, SKIP_EA_DEC" ); // Interrupt is not external
asm ( "subi ea, ea, 4" ); // Must decrement ea by one
 // instruction for external
 // interrupts, so that the
 // interrupted instruction will
 // be run
asm ( "SKIP_EA_DEC:" );
asm ( "stw r1, 4(sp)" ); // Save all registers
asm ( "stw r2, 8(sp)" );
asm ( "stw r3, 12(sp)" );
asm ( "stw r4, 16(sp)" );
asm ( "stw r5, 20(sp)" );
asm ( "stw r6, 24(sp)" );
asm ( "stw r7, 28(sp)" );
asm ( "stw r8, 32(sp)" );
asm ( "stw r9, 36(sp)" );
asm ( "stw r10, 40(sp)" );
asm ( "stw r11, 44(sp)" );
asm ( "stw r12, 48(sp)" );
asm ( "stw r13, 52(sp)" );
asm ( "stw r14, 56(sp)" );
asm ( "stw r15, 60(sp)" );
asm ( "stw r16, 64(sp)" );
asm ( "stw r17, 68(sp)" );
asm ( "stw r18, 72(sp)" );
asm ( "stw r19, 76(sp)" );
asm ( "stw r20, 80(sp)" );
asm ( "stw r21, 84(sp)" );
asm ( "stw r22, 88(sp)" );
asm ( "stw r23, 92(sp)" );
asm ( "stw r25, 100(sp)" ); // r25 = bt (skip r24 = et, because
// it issaved above)
asm ( "stw r26, 104(sp)" ); // r26 = gp
// skip r27 because it is sp, and there is no point in saving this
asm ( "stw r28, 112(sp)" ); // r28 = fp
asm ( "stw r29, 116(sp)" ); // r29 = ea
asm ( "stw r30, 120(sp)" ); // r30 = ba
asm ( "stw r31, 124(sp)" ); // r31 = ra
asm ( "addi fp, sp, 128" );
asm ( "call interrupt_handler" ); // Call the C language interrupt
// handler
asm ( "ldw r1, 4(sp)" ); // Restore all registers
asm ( "ldw r2, 8(sp)" );
asm ( "ldw r3, 12(sp)" );
asm ( "ldw r4, 16(sp)" );
asm ( "ldw r5, 20(sp)" );
asm ( "ldw r6, 24(sp)" );
asm ( "ldw r7, 28(sp)" );
asm ( "ldw r8, 32(sp)" );
asm ( "ldw r9, 36(sp)" );
asm ( "ldw r10, 40(sp)" );
asm ( "ldw r11, 44(sp)" );
asm ( "ldw r12, 48(sp)" );
asm ( "ldw r13, 52(sp)" );
```

```
 asm ( "ldw r14, 56(sp)" );
 asm ( "ldw r15, 60(sp)" );
 asm ( "ldw r16, 64(sp)" );
 asm ( "ldw r17, 68(sp)" );
 asm ( "ldw r18, 72(sp)" );
 asm ( "ldw r19, 76(sp)" );
 asm ( "ldw r20, 80(sp)" );
 asm ( "ldw r21, 84(sp)" );
 asm ( "ldw r22, 88(sp)" );
 asm ( "ldw r23, 92(sp)" );
 asm ( "ldw r24, 96(sp)" );
 asm ( "ldw r25, 100(sp)" ); // r25 = bt
 asm ( "ldw r26, 104(sp)" ); // r26 = gp
 // skip r27 because it is sp, and we did not save this on the stack
 asm ( "ldw r28, 112(sp)" ); // r28 = fp
 asm ( "ldw r29, 116(sp)" ); // r29 = ea
 asm ( "ldw r30, 120(sp)" ); // r30 = ba
 asm ( "ldw r31, 124(sp)" ); // r31 = ra
 asm ( "addi sp, sp, 128" );
 asm ( "eret" );
 }
 /*****************************************************************************
 * Interrupt Service Routine
 * Services the counter interrupt.
 *
 * ipending - Control register 4 which has the pending external interrupts
 ******************************************************************************/
 void interrupt_handler(void)
 {
    // Declare integer that can be a varaible for ipending register read
    int ipending;

    // Declare some thing like char character; that can be used for reading the value from UART
    char character;

    // Declare message that needs to be sent to UART (to Transmitter) example static char message[20]="ECE3073";
    static char message[8] = "32194471";

    // Declare a variable like static int mess_pointer=0; that may be used for reading the above character elements in the  array one by one
    static int mess_pointer = 0;

    //  Read ipending
    NIOS2_READ_IPENDING(ipending);

    // if ipending say irq 0 occured
    if (ipending == 0x1)
    {
    //{
       // clear PIO edgecapture interrupt ( this for your MSB rising edge)
       *(InPort_Counter + 0x03) = 0x00;

       //write codes that will transmit each character in the message[] array to UART transmitter (Hint you need to use if loop) until end of the string
       if(message[mess_pointer] != '\0')
       {
```

```
        // Transmit the next message character
        *(RS232 + 0x01) = message[mess_pointer];
        mess_pointer++;
    }
}
//else {
    // Send the character 'E' to the UART transmitter
    // Write codes here if SW[2] is ON restart the transmission from A to E again
//}

// Write codes to check for interupt irq 1 occured for a character received in  RS232 receiver
if(ipending == 0x2)
{
    //  Write code to read the value to character variable
    character = *(RS232);

    // Write code to display received value in SEVEN SEGMENT DISPLAY , you need to know HEX display truthtable, ascii values of message chars.
    switch (character)
    {
        case '0':
            *(SEVEN) = 0xC0;
            break;
        case '1':
            *(SEVEN) = 0xF9;
            break;
        case '2':
            *(SEVEN) = 0xA4;
            break;
        case '3':
            *(SEVEN) = 0xB0;
            break;
        case '4':
            *(SEVEN) = 0x99;
            break;
        case '5':
            *(SEVEN) = 0x92;
            break;
        case '6':
            *(SEVEN) = 0x82;
            break;
        case '7':
            *(SEVEN) = 0xF8;
            break;
        case '8':
            *(SEVEN) = 0x80;
            break;
        case '9':
            *(SEVEN) = 0x90;
            break;
        case 'E':
            *(SEVEN) = 0x86;
            break;
        default:
            *(SEVEN) = 0xFF;
```

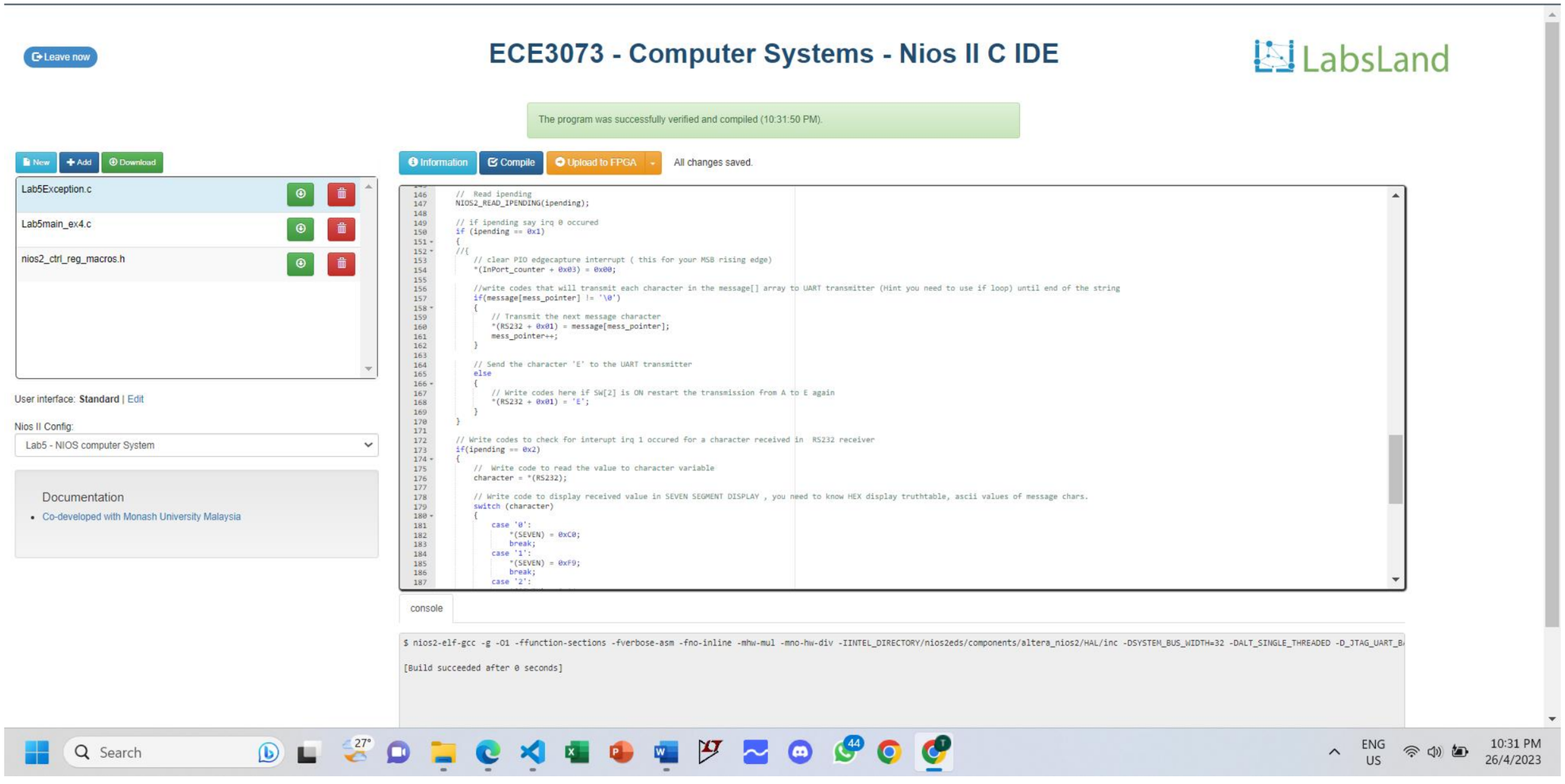
```
            break;
        }

        // Simultaneously display the count of the character received at UART receiver
        LED_Count++;
        *(RED_LEDS) = LED_Count;

        // Write code for clear RS232 interrupt bits
        *(RS232 + 0x02) = 0x00;
    }

    // else, ignore the interrupt
    return;
}
```

ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Exercise 4 (25 marks)

Now further write codes such that when all characters (your student numbers one by one as char) is transmitted and received, transmit character "E" continuously . That means I should see after your ID numbers, E will be always displayed (Refer to **Sample output of Exercise 4 video** in moodle). Ensure you keep the Notepad indicating your student ID and name aside in your recording Screen record submit to exercise 4 moodle link. (15 Marks)

Paste here the additional code from Exercise 3 to achieve the output (10 marks)



Actual Code (Interrupt Handler Code)

```
void interrupt_handler(void)
{
   // Declare integer that can be a varaible for ipending register read
   int ipending;

   // Declare some thing like char character; that can be used for reading the value from UART
   char character;

   // Declare message that needs to be sent to UART (to Transmitter) example static char message[20]="ECE3073";
   static char message[8] = "32194471";

   // Declare a variable like static int mess_pointer=0; that may be used for reading the above character elements in the  array one by one
   static int mess_pointer = 0;

   //  Read ipending
   NIOS2_READ_IPENDING(ipending);

   // if ipending say irq 0 occured
   if (ipending == 0x1)
   {
   //{
      // clear PIO edgecapture interrupt ( this for your MSB rising edge)
      *(InPort_counter + 0x03) = 0x00;

      //write codes that will transmit each character in the message[] array to UART transmitter (Hint you need to use if loop) until end of the string
      if(message[mess_pointer] != '\0')
      {
         // Transmit the next message character
         *(RS232 + 0x01) = message[mess_pointer];
         mess_pointer++;
      }

      // Send the character 'E' to the UART transmitter
      else
      {
         // Write codes here if SW[2] is ON restart the transmission from A to E again
         *(RS232 + 0x01) = 'E';
      }
}

// Write codes to check for interupt irq 1 occured for a character received in  RS232 receiver
if(ipending == 0x2)
{
   //  Write code to read the value to character variable
   character = *(RS232);

   // Write code to display received value in SEVEN SEGMENT DISPLAY , you need to know HEX display truthtable, ascii values of message chars.
   switch (character)
   {
      case '0':
```

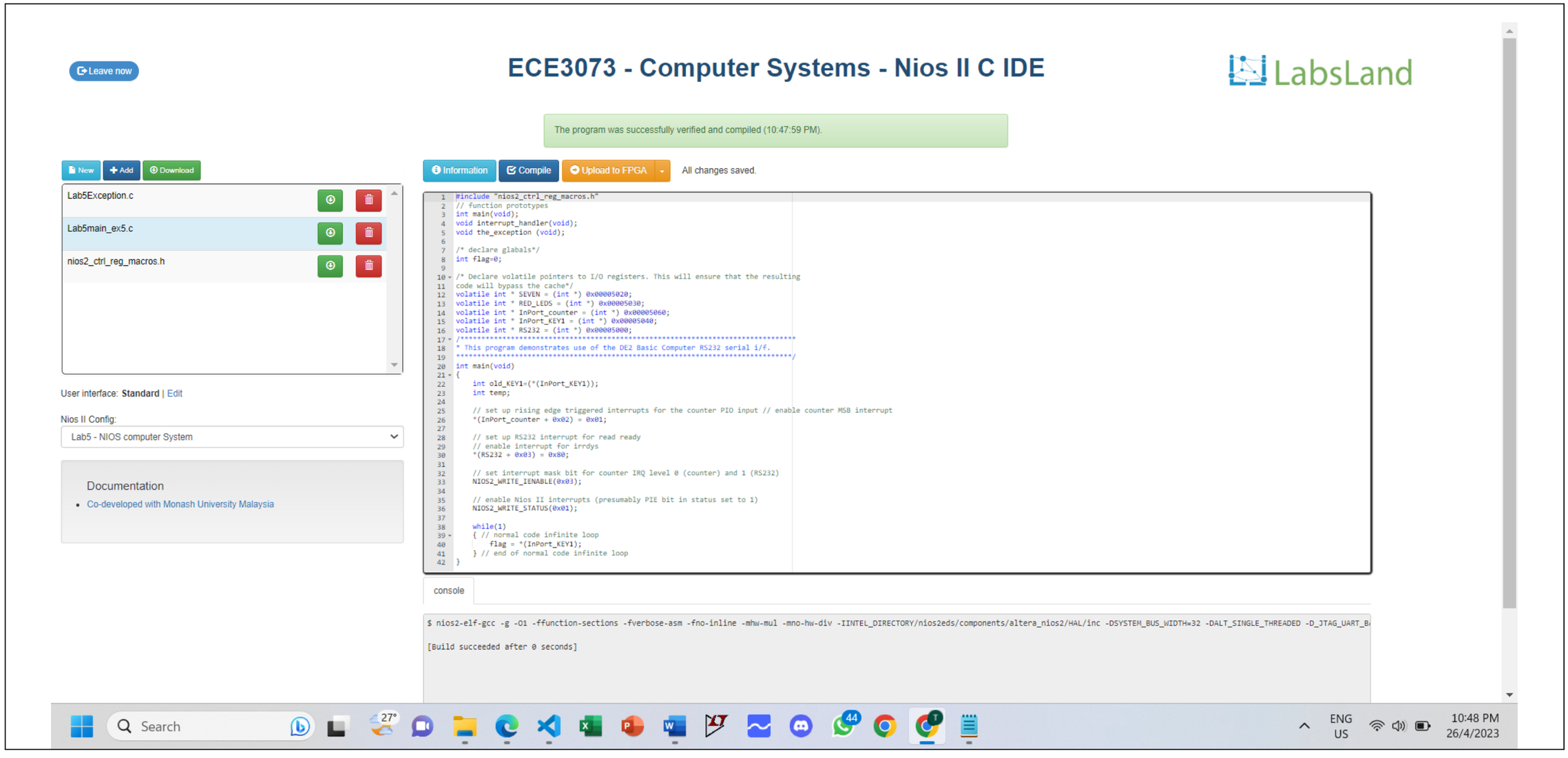
```
            *(SEVEN) = 0xC0;
            break;
        case '1':
            *(SEVEN) = 0xF9;
            break;
        case '2':
            *(SEVEN) = 0xA4;
            break;
        case '3':
            *(SEVEN) = 0xB0;
            break;
        case '4':
            *(SEVEN) = 0x99;
            break;
        case '5':
            *(SEVEN) = 0x92;
            break;
        case '6':
            *(SEVEN) = 0x82;
            break;
        case '7':
            *(SEVEN) = 0xF8;
            break;
        case '8':
            *(SEVEN) = 0x80;
            break;
        case '9':
            *(SEVEN) = 0x90;
            break;
        case 'E':
            *(SEVEN) = 0x86;
            break;
        default:
            *(SEVEN) = 0xFF;
            break;
    }

    // Simultaneously display the count of the character received at UART receiver
    LED_Count++;
    *(RED_LEDS) = LED_Count;

    // Write code for clear RS232 interrupt bits
    *(RS232 + 0x02) = 0x00;
  }

  // else, ignore the interrupt
  return;
}
```

ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Exercise 5 (15 Marks)

Now further include in your code such that when SW[2] of LabsLand is kept ON the cycle repeats . (Refer the **sample Exercise 5 output video**). Ensure you keep the Notepad indicating your student ID and name aside in your recording (10 marks)

Paste the additional code you wrote to achieve the output (5 Marks)



ECE3073 – Lab 5 on UART
Lab Sheet
Monash University Malaysia
To be implemented in LabsLand portal

Leave now

ECE3073 - Computer Systems - Nios II C IDE

LabsLand

The program was successfully verified and compiled (10:47:59 PM).

New | Add | Download

Lab5Exception.c
Lab5main_ex5.c
nios2_ctrl_reg_macros.h

User interface: **Standard** | Edit

Nios II Config:

Lab5 - NIOS computer System

Documentation

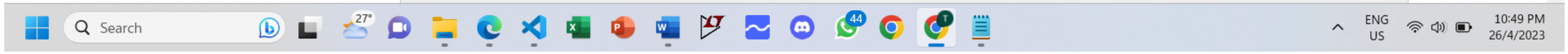
- Co-developed with Monash University Malaysia

Information | Compile | Upload to FPGA | All changes saved.

```
146     //  Read ipending
147     NIOS2_READ_IPENDING(ipending);
148
149     // if ipending say irq 0 occured
150     if (ipending == 0x1)
151     {
152     //{
153         // clear PIO edgecapture interrupt ( this for your MSB rising edge)
154         *(inPort_counter + 0x03) = 0x00;
155
156         //write codes that will transmit each character in the message[] array to UART transmitter (Hint you need to use if loop) until end of the string
157         if(message[mess_pointer] != '\0')
158         {
159             // Transmit the next message character
160             *(RS232 + 0x01) = message[mess_pointer];
161             mess_pointer++;
162         }
163
164         // Send the character 'E' to the UART transmitter
165         else
166         {
167             // Write codes here if SW[2] is ON restart the transmission from A to E again
168             *(RS232 + 0x01) = 'E';
169
170             // If the main code wants to loop back the student ID
171             if (flag)
172             {
173                 mess_pointer = 0x00;
174             }
175         }
176
177         // Write codes here if SW[2] is ON restart the transmission from A to E again
178         if(*(inPort_KEY1) == 0x1 && mess_pointer > sizeof(message)/sizeof(message[0]))
179         {
180             mess_pointer = 0;
181         }
182     }
183
184     // Write codes to check for interupt irq 1 occured for a character received in  RS232 receiver
185     if(ipending == 0x2)
186     {
187         //  Write code to read the value to character variable
```

console

```
$ nios2-elf-gcc -g -O1 -ffunction-sections -fverbose-asm -fno-inline -mhw-mul -mno-hw-div -IINTEL_DIRECTORY/nios2eds/components/altera_nios2/HAL/inc -DSYSTEM_BUS_WIDTH=32 -DALT_SINGLE_THREADED -D_JTAG_UART_BA

[Build succeeded after 0 seconds]
```



Actual Code (Main Code)

```
#include "nios2_ctrl_reg_macros.h"
// function prototypes
int main(void);
void interrupt_handler(void);
void the_exception (void);

/* declare glabals*/
int flag=0;

/* Declare volatile pointers to I/O registers. This will ensure that the resulting
code will bypass the cache*/
volatile int * SEVEN = (int *) 0x00005020;
volatile int * RED_LEDS = (int *) 0x00005030;
volatile int * InPort_counter = (int *) 0x00005060;
volatile int * InPort_KEY1 = (int *) 0x00005040;
volatile int * RS232 = (int *) 0x00005000;
/*****************************************************************************
* This program demonstrates use of the DE2 Basic Computer RS232 serial i/f.
*****************************************************************************/
int main(void)
{
   int old_KEY1=(*(InPort_KEY1));
   int temp;

   // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt
   *(InPort_counter + 0x02) = 0x01;

   // set up RS232 interrupt for read ready
   // enable interrupt for irrdys
   *(RS232 + 0x03) = 0x80;

   // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)
   NIOS2_WRITE_IENABLE(0x03);

   // enable Nios II interrupts (presumably PIE bit in status set to 1)
   NIOS2_WRITE_STATUS(0x01);

   while(1)
   { // normal code infinite loop
      flag = *(InPort_KEY1);
   } // end of normal code infinite loop
}
```

Actual Code (Interrupt Handler)

```
void interrupt_handler(void)
{
   // Declare integer that can be a varaible for ipending register read
   int ipending;

   // Declare some thing like char character; that can be used for reading the value from UART
   char character;

   // Declare message that needs to be sent to UART (to Transmitter) example static char message[20]="ECE3073";
   static char message[8] = "32194471";

   // Declare a variable like static int mess_pointer=0; that may be used for reading the above character elements in the  array one by one
   static int mess_pointer = 0;

   //  Read ipending
   NIOS2_READ_IPENDING(ipending);

   // if ipending say irq 0 occured
   if (ipending == 0x1)
   {
   //{
      // clear PIO edgecapture interrupt ( this for your MSB rising edge)
      *(InPort_counter + 0x03) = 0x00;

   //write codes that will transmit each character in the message[] array to UART transmitter (Hint you need to use if loop) until end of the string
   if(message[mess_pointer] != '\0')
   {
      // Transmit the next message character
      *(RS232 + 0x01) = message[mess_pointer];
      mess_pointer++;
   }

   // Send the character 'E' to the UART transmitter
   else
   {
      // Write codes here if SW[2] is ON restart the transmission from A to E again
      *(RS232 + 0x01) = 'E';

      // If the main code wants to loop back the student ID
      if (flag)
      {
         mess_pointer = 0x00;
      }
   }

   // Write codes here if SW[2] is ON restart the transmission from A to E again
   if(*(InPort_KEY1) == 0x1 && mess_pointer > sizeof(message)/sizeof(message[0]))
   {
      mess_pointer = 0;
   }
}
```

```
// Write codes to check for interupt irq 1 occured for a character received in  RS232 receiver
if(ipending == 0x2)
{
   //  Write code to read the value to character variable
   character = *(RS232);

   // Write code to display received value in SEVEN SEGMENT DISPLAY , you need to know HEX display truthtable, ascii values of message chars.
   switch (character)
   {
      case '0':
         *(SEVEN) = 0xC0;
         break;
      case '1':
         *(SEVEN) = 0xF9;
         break;
      case '2':
         *(SEVEN) = 0xA4;
         break;
      case '3':
         *(SEVEN) = 0xB0;
         break;
      case '4':
         *(SEVEN) = 0x99;
         break;
      case '5':
         *(SEVEN) = 0x92;
         break;
      case '6':
         *(SEVEN) = 0x82;
         break;
      case '7':
         *(SEVEN) = 0xF8;
         break;
      case '8':
         *(SEVEN) = 0x80;
         break;
      case '9':
         *(SEVEN) = 0x90;
         break;
      case 'E':
         *(SEVEN) = 0x86;
         break;
      default:
         *(SEVEN) = 0xFF;
         break;
   }

   // Simultaneously display the count of the character received at UART receiver
   LED_Count++;
   *(RED_LEDS) = LED_Count;

   // Write code for clear RS232 interrupt bits
   *(RS232 + 0x02) = 0x00;
```

```
    }

    // else, ignore the interrupt
    return;
}
```