

# 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 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.

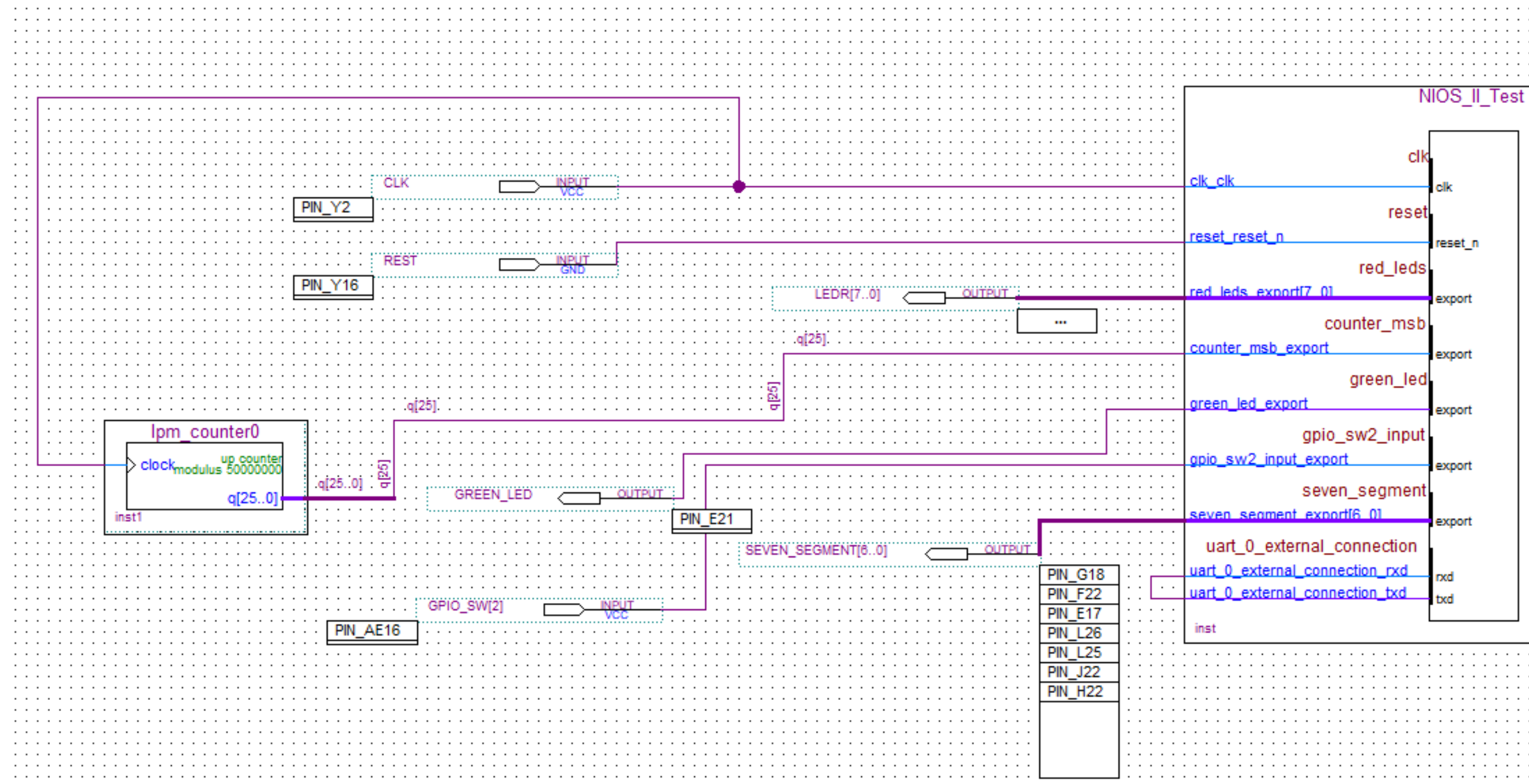
## 2. The computer system for this lab details

(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)

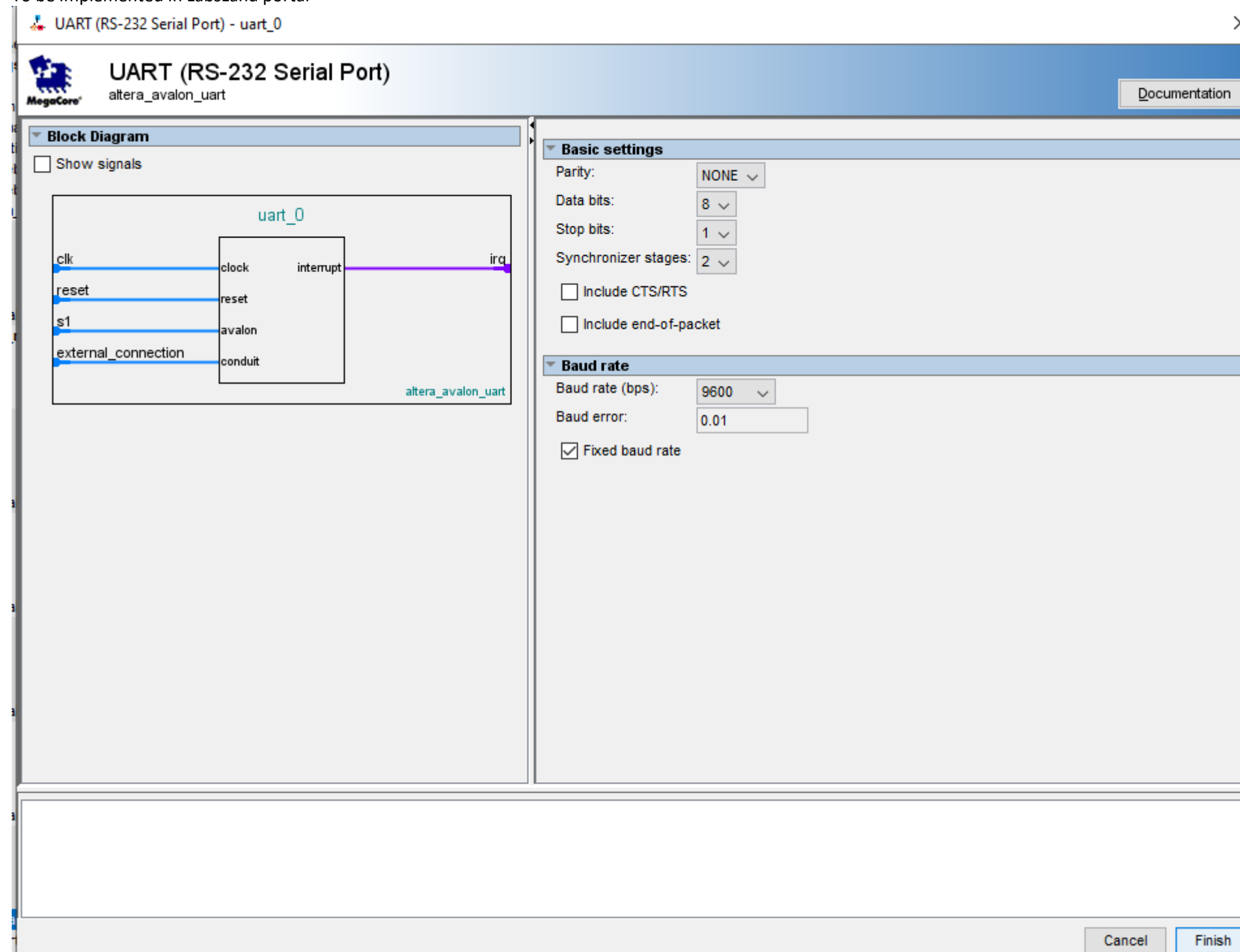
(b) Following are the screen shot of the schematics



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

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

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)

## Exercise (1) ( 5 marks)

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

The screenshot displays the Altera Platform Designer (Qsys) interface for a project named "NIOSII\_LAB5\_NIGEL.qsys". The main window shows the configuration for the "UART (RS-232 Serial Port) Intel FPGA IP" (altera\_avalon\_uart). The configuration is divided into two main sections: "Block Diagram" and "Basic settings".

**Block Diagram:** The diagram shows the UART IP block with its inputs and outputs. The inputs are "clk", "reset", "s1", and "external\_connection". The outputs are "clock", "reset", "avalon", and "conduit". The "interrupt" output is connected to the "irq" signal.

**Basic settings:** The settings are as follows:

- Parity: NONE
- Data bits: 8
- Stop bits: 1
- Synchronizer stages: 2
- Include CTS/RTS: ☐
- Include end-of-packet: ☐
- Baud rate (bps): 9600
- Baud error: 0.01
- Fixed baud rate: ☒

The "Hierarchy" pane on the left shows the project structure, including the "UART (RS-232 Serial Port) Intel FPGA IP" component. The "System Contents" pane on the right shows the system components, including the "UART (RS-232 Serial Port) Intel FPGA IP" component.

The bottom status bar indicates "0 Errors, 1 Warning". The "Generate HDL..." and "Finish" buttons are visible.



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

Platform Designer - NIOSII\_LAB5\_NIGEL.qsys (C:\intelFPGA\_lite\18.1\Lab5\NIOSII\_LAB5\_NIGEL.qsys)

File Edit System Generate View Tools Help

IP Catalog System Contents Address Map Interconnect Requirements

System: NIOSII\_LAB5\_NIGEL Path: uart\_0.external\_connection

Use Connections Name Description Export Clock Base End IRQ Tags Opcode Name

clk1	Clock Input	Double-click to export	clk_0	0x0000	0x3fff			
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]					
reset1	Reset Input	Double-click to export	[clk1]					
pio_1	PIO (Parallel I/O) Intel FPGA IP	Double-click to export	clk_0	0x5060	0x506f			
clk	Clock Input	Double-click to export	[clk]					
reset	Reset Input	Double-click to export	[clk]					
s1	Avalon Memory Mapped Slave	Double-click to export	counter_msb					
external_connection	Conduit	Double-click to export						
irq	Interrupt Sender							
pio_2	PIO (Parallel I/O) Intel FPGA IP			0x5050	0x505f			
clk	Clock Input							
reset	Reset Input							
s1	Avalon Memory Mapped Slave			0x5040	0x504f			
external_connection	Conduit							
pio_3	PIO (Parallel I/O) Intel FPGA IP			0x5020	0x502f			
clk	Clock Input							
reset	Reset Input							
s1	Avalon Memory Mapped Slave			0x5000	0x501f			
external_connection	Conduit							
pio_4	PIO (Parallel I/O) Intel FPGA IP							
clk	Clock Input							
reset	Reset Input							
s1	Avalon Memory Mapped Slave							
external_connection	Conduit							
uart_0	JTAG UART Intel FPGA IP							
clk	Clock Input	Double-click to export	clk_0					

Generate Completed

Info: rsp\_demux: "mm\_interconnect\_0" instantiated altera\_merlin\_demul

Info: rsp\_mux: "mm\_interconnect\_0" instantiated altera\_merlin\_multiplex

Info: Reusing file C:/intelFPGA\_lite/18.1/Lab5/NIOSII\_LAB5\_NIGEL/synti

Info: rsp\_mux\_001: "mm\_interconnect\_0" instantiated altera\_merlin\_mul

Info: Reusing file C:/intelFPGA\_lite/18.1/Lab5/NIOSII\_LAB5\_NIGEL/synti

Info: avalon\_st\_adapter: "mm\_interconnect\_0" instantiated altera\_avak

Info: error\_adapter\_0: "avalon\_st\_adapter" instantiated error\_adapter

Info: NIOSII\_LAB5\_NIGEL: Done "NIOSII\_LAB5\_NIGEL" with 31 modules, 45

Info: qsys-generate succeeded.

Info: Finished: Create HDL design files for synthesis

Generate: completed with warnings.

Stop Close

Current filter:

Messages

Type	Path	Message
Warning	1 Warning	
Warning	NIOSII_LAB5_NIGEL.nios2_qsys_0	Nios II Classic cores are no longer recommended for new projects
Info	3 Info Messages	
Info	NIOSII_LAB5_NIGEL.pio_1	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Info	NIOSII_LAB5_NIGEL.pio_3	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Info	NIOSII_LAB5_NIGEL.jtag_uart_0	JTAG UART IP input clock need to be at least double (2x) the operating frequency of JTAG TCK on board

0 Errors, 1 Warning

Generate HDL... Finish

ENG US 11:41 PM 19/4/2023





Pin Planner - C:/intelFPGA\_lite/18.1/Lab5/Lab5Nigel - Lab5Nigel

File Edit View Processing Tools Window Help

Search altera.com

Report

Report not available

Groups Report

Tasks

- Early Pin Planning
  - Early Pin Planning...
  - Run I/O Assignment /
  - Export Pin Assignment

Top View - Wire Bond  
 Cyclone IV E - EP4CE115F29C8

Pin Legend

Symbol	Pin Type
○	User I/O
●	User assign...
●	Fitter assign...
●	Unbonded ...
●	Reserved pin
○	Other confi...
○	DEV_OE
○	DEV_CLR
○	DIFF_n
○	DIFF_p
○	DQ
○	DQS

Named: \* Edit: X

Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Input Preservation
altera_reserved_tdi	Input				PIN_P7	2.5 V ...fault		8mA (default)			
altera_reserved_tdo	Output				PIN_P6	2.5 V ...fault		8mA (default)	2 (default)		
altera_reserved_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_SEGMENT[6]	Output	PIN_H22	6	B6_N0	PIN_H22	2.5 V		8mA (default)	2 (default)		
SEVEN_SEGMENT[5]	Output	PIN_J22	6	B6_N0	PIN_J22	2.5 V		8mA (default)	2 (default)		
SEVEN_SEGMENT[4]	Output	PIN_L25	6	B6_N1	PIN_L25	2.5 V		8mA (default)	2 (default)		
SEVEN_SEGMENT[3]	Output	PIN_L26	6	B6_N1	PIN_L26	2.5 V		8mA (default)	2 (default)		
SEVEN_SEGMENT[2]	Output	PIN_E17	7	B7_N2	PIN_E17	2.5 V		8mA (default)	2 (default)		
SEVEN_SEGMENT[1]	Output	PIN_F22	7	B7_N0	PIN_F22	2.5 V		8mA (default)	2 (default)		
SEVEN_SEGMENT[0]	Output	PIN_G18	7	B7_N2	PIN_G18	2.5 V		8mA (default)	2 (default)		
<<new node>>											

0% 00:00:00

ENG US 11:33 PM 26/4/2023

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

ECE3073 - Computer Systems - Nios II C IDE

The program was successfully verified and compiled (12:03:51 AM).

Lab5Exception.c

Lab5main\_ex2\_q1.c

nios2\_ctrl\_reg\_macros.h

1 #include "nios2\_ctrl\_reg\_macros.h"

2 // function prototypes

3 int main(void);

4 void interrupt\_handler(void);

5 void the\_exception(void);

6 // declare globals

7 int flag=0;

8 // Declare volatile pointers to I/O registers. This will ensure that the resulting

9 code will bypass the cache\*/

10 volatile int \* SEVEN = (int \*) 0x00005020;

11 volatile int \* RED\_LED5 = (int \*) 0x00005030;

12 volatile int \* InPort\_counter = (int \*) 0x00005060;

13 volatile int \* InPort\_KEY1 = (int \*) 0x00005040;

14 volatile int \* RS232 = (int \*) 0x00005000;

15 // =====

16 \* This program demonstrates use of the DE2 Basic Computer RS232 serial I/F.

17 // =====

18 int main(void)

19 {

20 int old\_KEY1=(\*InPort\_KEY1);

21 int temp;

22 // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt

23 // set up RS232 interrupt for read ready

24 // enable interrupt for irqdy

25 // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)

26 // enable Nios II interrupts (presumably PIE bit in status set to 1)

27 while(1)

28 { // normal code infinite loop

29 \*(SEVEN) = 0x49;

30 } // end of normal code infinite loop

31 }

32 }

User interface: Standard | Edit

Nios II Config:

Lab5 - NIOS computer System

Documentation

console

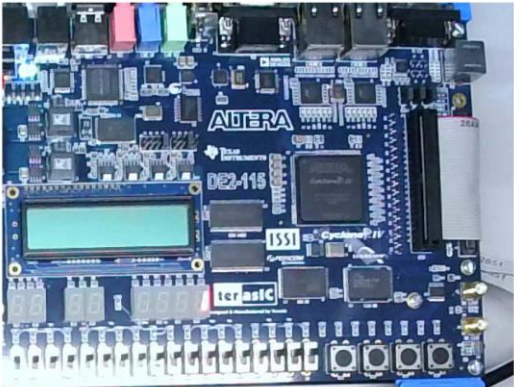
FPGA Output

This FPGA is hosted at Monash University Malaysia.

03:04

Leave now

Intel FPGA Laboratory



17 16 15 14 13

12 11 10 9 8

7 6 5 4 3

2 1 0

KEY3 KEY2 KEY1 KEY0

You are using: monash-malaysia-monash-de2\_115\_112. Experiencing any problem with this device? Let us know

Console

Displaying “2”

Compiled Code on Labsland

ECE3073 - Computer Systems - Nios II C IDE

The program was successfully verified and compiled (12:09:30 AM).

Lab5Exception.c

Lab5main\_ex2\_q2.c

nios2\_ctrl\_reg\_macros.h

1 #include "nios2\_ctrl\_reg\_macros.h"

2 // function prototypes

3 int main(void);

4 void interrupt\_handler(void);

5 void the\_exception(void);

6 // declare globals

7 int flag=0;

8 // Declare volatile pointers to I/O registers. This will ensure that the resulting

9 code will bypass the cache\*/

10 volatile int \* SEVEN = (int \*) 0x00005020;

11 volatile int \* RED\_LED5 = (int \*) 0x00005030;

12 volatile int \* InPort\_counter = (int \*) 0x00005060;

13 volatile int \* InPort\_KEY1 = (int \*) 0x00005040;

14 volatile int \* RS232 = (int \*) 0x00005000;

15 // =====

16 \* This program demonstrates use of the DE2 Basic Computer RS232 serial I/F.

17 // =====

18 int main(void)

19 {

20 int old\_KEY1=(\*InPort\_KEY1);

21 int temp;

22 // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt

23 // set up RS232 interrupt for read ready

24 // enable interrupt for irqdy

25 // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)

26 // enable Nios II interrupts (presumably PIE bit in status set to 1)

27 while(1)

28 { // normal code infinite loop

29 \*(SEVEN) = 0x44;

30 } // end of normal code infinite loop

31 }

32 }

User interface: Standard | Edit

Nios II Config:

Lab5 - NIOS computer System

Documentation

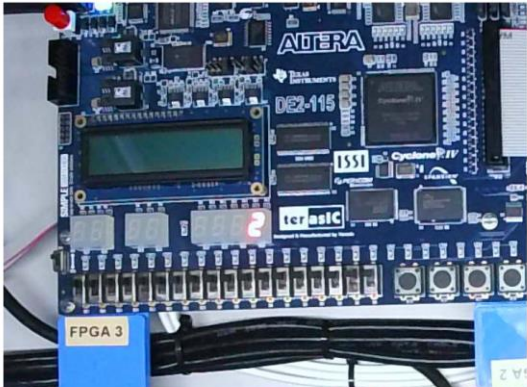
console

FPGA Output

03:16

Leave now

Intel FPGA Laboratory



17 16 15 14 13

12 11 10 9 8

7 6 5 4 3

2 1 0

KEY3 KEY2 KEY1 KEY0

You are using: monash-malaysia-monash-de2\_115\_113. Experiencing any problem with this device? Let us know

Console

<div><a href="#">Code</a><pre>#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         <b>*(SEVEN) = 0xF9;</b>     } // end of normal code infinite loop }</pre></div>	<div><a href="#">Code</a><pre>#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         <b>*(SEVEN) = 0xA4;</b>     } // end of normal code infinite loop }</pre></div>
---	---



## 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)

The screenshot displays the LabsLand portal interface. On the left, a file explorer shows 'Lab5Exception.c', 'Lab5main\_ex3.c', and 'nios2\_ctrl\_reg\_macros.h'. Below it, the 'User interface' is set to 'Standard' and 'Nios II Config' is set to 'Lab5 - NIOS computer System'. The main area shows the C code for 'main(void)' which includes headers, declares global variables, and implements a loop that transmits characters to the UART register. A green status bar at the top indicates 'The program was successfully verified and compiled (10:00:43 PM)'. The bottom console shows the compilation command and a successful build message.

```
1 #include "nios2_ctrl_reg_macros.h"
2 // function prototypes
3 int main(void);
4 void interrupt_handler(void);
5 void the_exception(void);
6 /* declare globals */
7 int flag=0;
8 /* Declare volatile pointers to I/O registers. This will ensure that the resulting
9 code will bypass the cache */
10 volatile int * SEVEN = (int *) 0x00005020;
11 volatile int * RED_LEDS = (int *) 0x00005030;
12 volatile int * InPort_Counter = (int *) 0x00005060;
13 volatile int * InPort_KEY1 = (int *) 0x00005040;
14 volatile int * RS232 = (int *) 0x00005000;
15 /* This program demonstrates use of the DE2 Basic Computer RS232 serial i/f.
16 *****
17 */
18 int main(void)
19 {
20     int old_KEY1=(*InPort_KEY1);
21     int temp;
22
23     // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt
24     *(InPort_Counter + 0x02) = 0x01;
25
26     // set up RS232 interrupt for read ready
27     *(RS232 + 0x03) = 0x00;
28
29     // enable interrupt for irrdy
30     // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)
31     NIOS2_WRITE_IENABLE(0x03);
32
33     // enable Nios II interrupts (presumably PIE bit in status set to 1)
34     NIOS2_WRITE_STATUS(0x01);
35
36     while(1)
37     { // normal code infinite loop
38         // end of normal code infinite loop
39     }
40 }
```

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
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
}
```

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)

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.

```
130 * ipending - Control register 4 which has the pending external interrupts
131 .....
132 void interrupt_handler(void)
133 {
134     // Declare integer that can be a variable for ipending register read
135     int ipending;
136
137     // Declare some thing like char character; that can be used for reading the value from UART
138     char character;
139
140     // Declare message that needs to be sent to UART (to Transmitter) example static char message[20]="ECE3073";
141     static char message[8] = "32194471";
142
143     // 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
144     static int mess_pointer = 0;
145
146     // Read ipending
147     NIOS2_READ_IPENDING(ipending);
148
149     // if ipending say irq 0 occurred
150     if (ipending == 0x1)
151     {
152         // clear PIO edgecapture interrupt ( this for your MSB rising edge)
153         *(InPort_Counter + 0x03) = 0x00;
154
155         //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
156         if(message[mess_pointer] != '\0')
157         {
158             // Transmit the next message character
159             *(RS232 + 0x01) = message[mess_pointer];
160             mess_pointer++;
161         }
162     }
163     //else {
164     // Send the character 'E' to the UART transmitter
165     // Write codes here if SW[2] is ON restart the transmission from A to E again
166     //}
167
168     // Write codes to check for interrupt irq 1 occurred for a character received in RS232 receiver
169     if(ipending == 0x2)
170     {
171
```

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]
```

Windows taskbar: Search, 27°C, 10:02 PM 26/4/2023



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

NewAddDownload

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

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

InformationCompileUpload to FPGAAll changes saved.

```
167 //)
168
169 // Write codes to check for interrupt irq 1 occurred 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':
209             *(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]

Windows Taskbar

10:03 PM 26/4/2023

NewAddDownload

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

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

InformationCompileUpload to FPGA

All changes saved.

```
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':
209          *(SEVEN) = 0x86;
210          break;
211      default:
212          *(SEVEN) = 0xFF;
213          break;
214      }
215      // Simultaneously display the count of the character received at UART receiver
216      LED_Count++;
217      *(RED_LEDS) = LED_Count;
218
219      // Write code for clear RS232 interrupt bits
220      *(RS232 + 0x02) = 0x00;
221      }
222
223      // else, ignore the interrupt
224      return;
225  }
226  }
```

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\_3TAG\_UART\_B  
[Build succeeded after 0 seconds]

Windows Taskbar

Search

Taskbar Icons

System Tray

10:03 PM  
26/4/2023

### 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 is saved 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 variable 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 occurred
    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 interrupt irq 1 occurred 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) = 0xCo;
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;
}
```

## 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)

The screenshot displays the ECE3073 - Computer Systems - Nios II C IDE interface. At the top, a green notification bar states: "The program was successfully verified and compiled (10:31:50 PM)." The main workspace is divided into three panes. The left pane shows a file explorer with files: Lab5Exception.c, Lab5main\_ex4.c, and nios2\_ctrl\_reg\_macros.h. The middle pane contains C code for UART communication. The code includes comments and logic for transmitting a message and receiving a character. The right pane shows the console output, which includes the command: `$ 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` and the output: `[Build succeeded after 0 seconds]`. The bottom of the image shows a Windows taskbar with various application icons and the system clock indicating 10:31 PM on 26/4/2023.

```
146 // Read ipending
147 NIOS2_READ_IPENDING(ipending);
148
149 // if ipending say irq 0 occurred
150 if (ipending == 0x1)
151 {
152     // clear PIO edgecapture interrupt ( this for your MSB rising edge)
153     *(InPort_counter + 0x03) = 0x00;
154
155     //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
156     if(message[mess_pointer] != '\0')
157     {
158         // Transmit the next message character
159         *(RS232 + 0x01) = message[mess_pointer];
160         mess_pointer++;
161     }
162
163     // Send the character 'E' to the UART transmitter
164     else
165     {
166         // Write codes here if SW[2] is ON restart the transmission from A to E again
167         *(RS232 + 0x01) = 'E';
168     }
169
170 }
171
172 // Write codes to check for interrupt irq 1 occurred for a character received in RS232 receiver
173 if(ipending == 0x2)
174 {
175     // Write code to read the value to character variable
176     character = *(RS232);
177
178     // Write code to display received value in SEVEN SEGMENT DISPLAY , you need to know HEX display truthtable, ascii values of message chars.
179     switch (character)
180     {
181     case '0':
182         *(SEVEN) = 0xC0;
183         break;
184     case '1':
185         *(SEVEN) = 0xF9;
186         break;
187     case '2':
```

**Actual Code (Interrupt Handler Code)**

```
void interrupt_handler(void)
{
    // Declare integer that can be a variable 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 occurred
    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 interrupt irq 1 occurred 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;
}
```

## 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 - Computer Systems - Nios II C IDE**

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

Information Compile Upload to FPGA All changes saved.

1 #include "nios2\_ctrl\_reg\_macros.h"  
2 // function prototypes  
3 int main(void);  
4 void interrupt\_handler(void);  
5 void the\_exception (void);  
6  
7 /\* declare globals \*/  
8 int flag=0;  
9  
10 /\* Declare volatile pointers to I/O registers. This will ensure that the resulting  
11 code will bypass the cache \*/  
12 volatile int \* SEVEN = (int \*) 0x00005020;  
13 volatile int \* RED\_LEDS = (int \*) 0x00005030;  
14 volatile int \* InPort\_counter = (int \*) 0x00005060;  
15 volatile int \* InPort\_KEY1 = (int \*) 0x00005040;  
16 volatile int \* RS232 = (int \*) 0x00005000;  
17  
18 /\* This program demonstrates use of the DE2 Basic Computer RS232 serial i/f.  
19  
20 int main(void)  
21 {  
22 int old\_KEY1=\*(InPort\_KEY1);  
23 int temp;  
24  
25 // set up rising edge triggered interrupts for the counter PIO input // enable counter MSB interrupt  
26 \*(InPort\_counter + 0x02) = 0x01;  
27  
28 // set up RS232 interrupt for read ready  
29 // enable interrupt for irrdys  
30 \*(RS232 + 0x03) = 0x00;  
31  
32 // set interrupt mask bit for counter IRQ level 0 (counter) and 1 (RS232)  
33 NIOS2\_WRITE\_IENABLE(0x03);  
34  
35 // enable Nios II interrupts (presumably PIE bit in status set to 1)  
36 NIOS2\_WRITE\_STATUS(0x01);  
37  
38 while(1)  
39 { // normal code infinite loop  
40 flag = \*(InPort\_KEY1);  
41 } // end of normal code infinite loop  
42 }

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]

Windows taskbar at the bottom shows the time as 10:48 PM on 26/4/2023.

Leave now

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

ECE3073 - Computer Systems - Nios II C IDE

LabsLand

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

Information

Compile

Upload to FPGA

All changes saved.

```
146 // Read ipending
147 NIOS2_READ_IPENDING(ipending);
148
149 // if ipending say irq 0 occurred
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 // Write codes here if SW[2] is ON restart the transmission from A to E again
177 if(*(InPort_KEY1) == 0x1 && mess_pointer > sizeof(message)/sizeof(message[0]))
178 {
179 mess_pointer = 0;
180 }
181 }
182
183 // Write codes to check for interrupt irq 1 occurred for a character received in RS232 receiver
184 if(ipending == 0x2)
185 {
186 // Write code to read the value to character variable
187
```

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]

Search

27°

ENG US

10:49 PM 26/4/2023



**Actual Code (Main Code)**

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

/* declare globals */
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 variable 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 occurred
    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 interrupt irq 1 occurred 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;
```

ECE3073 – Lab 5 on UART

Lab Sheet

Monash University Malaysia

To be implemented in LabsLand portal

```
}
```

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

```
}
```