LAB 12 – NIOS II – SERIAL COMMUNICATION

Purpose

The purpose of this lab is to learn how serial communication works in computer systems.

Overview

Computer systems communicate in a variety of ways, but one of the most common ways is to send the bits of data in a serial fashion. A serial communication channel takes a byte of data and then sends one bit at a time on a single wire connected to another system. The receiving system takes the bit values from the single wire and re-assembles them into a byte of data which can then be handed off to the CPU. The hardware device that does the sending of the serial bits and/or receives the serial bits is called a UART (Universal Asynchronous Receiver-Transmitter.)

The port created by the Qsys Tool simplifies the UART interface. What is provided is a byte location where an ASCII character can be written so that the UART can begin transmitting the byte across the serial channel. The same location can be read to see if a byte has been received by the UART. The bits assigned to the DATA register are shown below. Bits 7..0 are used for writing DATA to the UART and reading DATA from the UART. The only other important bit is the 15th bit ("RVALID"). If this bit is set, the DATA read (bits 7..0) is a legitimate, new ASCII character that has come in. If the bit is not set, then what is in bits 7..0 is not valid and can be discarded. The byte found in bits 23..16 of the word will indicate how many ASCII characters are in the receive buffer (including the one found in bits 7..0) that can be read. Note that as soon as you read a character (or read the port register), it is no longer available for reading from the "DATA" bits (bits 7..0) and the RAVAIL will decrement until it reaches zero. The buffer is used to keep track of bytes/characters that have arrived faster than the receiver's CPU has been able to read them.

Table 2. RS232 UART Core register map

Offset in bytes	Register Name	R/W	Bit description										
			31...24	23...16	15	14...11	10	9	8	7	6...2	1	0
0	data	RW	*(1)*	RAVAIL	RVALID	*(1)*		PE	*(2)*	*(2)*	DATA		

Pre-Lab – Review the Use of Pointers in C

Review how pointers are used in a language like C. There are many tutorials available on-line. Here are a few:
https://www.tutorialspoint.com/cprogramming/c_pointers.htm
http://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/
https://www.programiz.com/c-programming/c-pointers
https://www.cprogramming.com/tutorial/c/lesson6.html
Remember that when you declare a variable, whether it is a normal variable or a pointer, the compiler allocates a spot in memory for that variable. That spot in memory must then be assigned a value. For a normal variable, the value is typically an integer or a character. For a pointer variable, that value is always the address of another variable (usually a normal variable). The reference operator (&) is used on normal variables to get the address of the variable. The Dereference operator (*) is used with pointer variables to get the value of the normal variable that the pointer is pointing at.

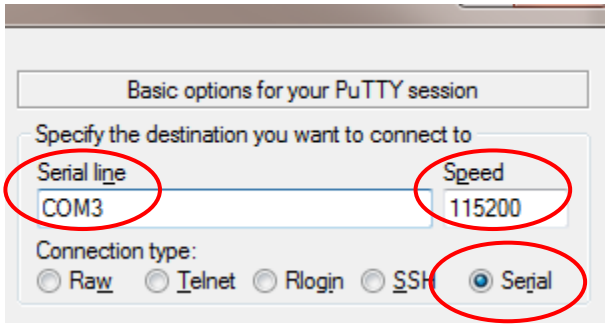
Part I – Setting Up the DE2 Board

STEP #1A

- Plug in serial/RS232, USB, and power to Altera DE2 board
- Setup a Lab 12 folder using the templates from Moodle
- Load DE2 board with Lab 12 NIOS Processor Image (Qsys system)

⊟	**RAM**	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	0x7fff
→	reset1	Reset Input	*Double-click to export*	[clk1]		
⊟	**LCD**	16x2 Character Display				
→	clock_reset	Clock Input	*Double-click to export*	**clk_0**		
→	clock_reset_reset	Reset Input	*Double-click to export*	[clock_reset]		
→	avalon_lcd_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clock_reset]	0x8000	0x8001
	external_interface	Conduit	**lcd**			
⊟	**UART**	RS232 UART				
→	clock_reset	Clock Input	*Double-click to export*	**clk_0**		
→	clock_reset_reset	Reset Input	*Double-click to export*	[clock_reset]		
→	avalon_rs232_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clock_reset]	0x8020	0x8027
	external_interface	Conduit	**uart**			
⊟	**timer_0**	Interval Timer				
→	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]	0x8040	0x805f

In this part of the lab, you will be setting up your system. The system includes the NIOS CPU with RAM, a port for communicating with the LCD, and a UART port for serial communication. The figure above shows the Qsys system contents, including memory locations and device names in the leftmost column (e.g. "/dev/UART" – note the capitalization of the module name). In addition to the NIOS pieces, the system requires a serial connection to your PC (cable with 9-pin D-style connectors). The cable should be connected to the "RS232" connector on the top, right side of the DE2 board and then connected to the serial port on the PC. On the PC, you should start the PUTTY program and set it up for serial communication using a baud rate of 115200. You may have to look at the Windows Device Manager to find out which of the COM ports should be used. The figure below shows COM3 as the selected com port. Yours might be COM0, COM1, or some other com port ID. After setting up the connection, click on the "Open" button at the bottom of the dialog box.



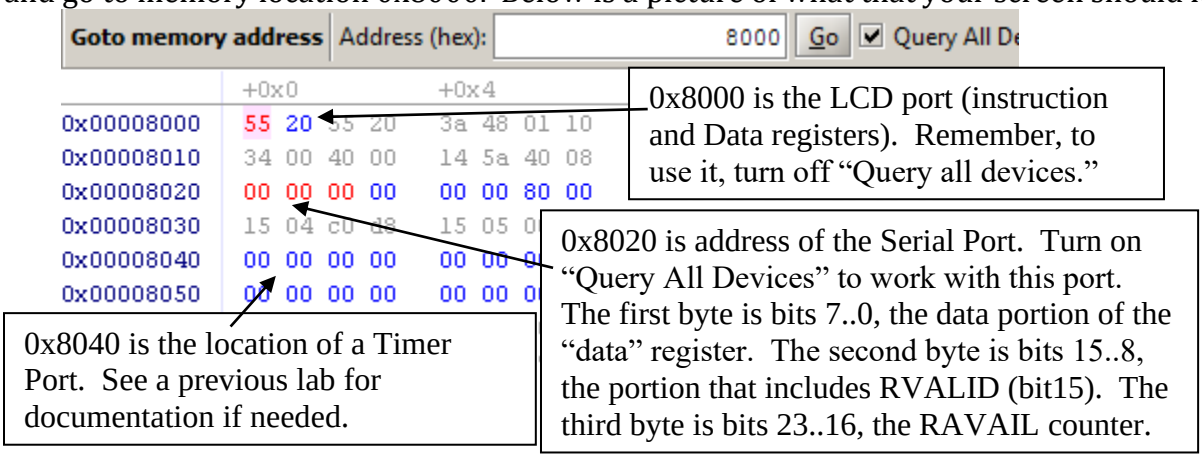
Finally, open up the Altera Debugger using the *.sof, *.qsys, and *.jdi files from the S: drive or Moodle. Using the Part I source code, compile/load the program on the DE2 board. At this point you won't be running the program, but you do need something that can be used to initialize the debugger, allowing you to use the memory tab.

STEP #1B

- Compile & Load & Run Part I's example C code onto the NIOS Processor
  - Use the Device Driver Support project type (BSP)
- Open Windows Device Manager, check for available COM Ports
- Connect Putty on COM port to DE2 board (baud 115200)

Part II – Using the UART from the memory tab

After loading your program onto the DE2 board with the Debugger, switch to the memory tab and experiment with the Serial Port (UART). Enable "Query All Devices," choose a byte size display, and go to memory location 0x8000. Below is a picture of what that your screen should look like.



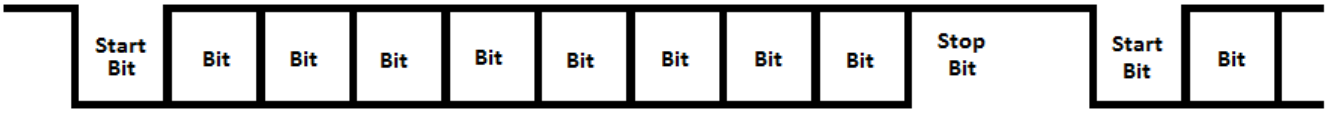
Position your Putty terminal on one screen and your Debugger on the other screen. To send a character from the DE2 board to the PC, enter the ASCII code in the spot for the data byte (address 0x8020) and it should show up on the Putty screen. To send a character from Putty (the PC) to the DE2 board, use your mouse to select the Putty program and then start typing keys on the PC keyboard. You should be able to see them in the data byte (address 0x8020) register if you "refresh" the memory tab. You should also see the RVALID flag set (the most significant bit of address 0x8021) and you should see RAVAIL (address 0x8022) indicate a count of how many characters have arrived and have yet to be read. Fill in the table at the end of this assignment sheet as you test the system.

STEP #2A

- Note: http://www.asciitable.com/
- With DE2 board loaded & running, Putty connected…
- Open Altera's memory tab to 0x8000
- Set view to bytes instead of words
- Check "Query All Devices"
- Pause the program, write an ASCII character into byte 0x8020, observe Putty
- Type several characters into Putty, on Altera rapidly "Refresh Memory", observe
- Fill in table on worksheet

Part III – Capture the Transmit and Receive signals in SignalTap

Screen capture the UART_RXD and UART_TXD signals using the SignalTap scope. To get the RXD signal to change, click on the Run Analysis button in SignalTap and then type a key into the Putty window. The waveform you would expect to see should look similar to the figure shown. The figure shows one full byte being transmitted and the beginning of a second byte. The Start Bit is always a low value and the Stop Bit is always a high value. The state of each data bit could be either high or low, depending on whether you are sending a one or a zero bit.



Next, capture the waveform on the TXD signal when a byte is being sent to the PC. To do this, start the analysis in Signal Tap and then type a character into the 0x8020 memory location. Finally, answer the question at the bottom of the answer sheet.

STEP #3A

- With DE2 board loaded & running, Putty connected…
- Open SignalTap, start acquiring
- Type an ASCII character into Putty, observe SignalTap
- Pause Altera debugger
- Restart SignalTap acquiring
- Write an ASCII character into Altera's memory byte 0x8020, observe SignalTap
- Answer question on worksheet

Part IV – Simple C Program

In this part of the lab, you will be writing a simple C program which will read characters entered into Putty, echo them back to the Putty screen, and display each character on the LCD screen. You should refer to the "Character_LCD.pdf" and "RS232.pdf" manual pages for information on the various software functions for interfacing to the two devices.

Your main() function should do the following:

1. Open up both devices (make sure you use the correct capitalized names – see the above Qsys screenshot for the names used). You do not need to print anything if there are errors opening the devices – use the debugger to check the return value from the function call which opens the device.
2. Initialize the LCD device.
3. Create an infinite loop that checks to see if a character has arrived. When one arrives, you should do the following:
    a. display the character on the LCD display exactly as it was received and
    b. echo the character back to the putty screen by writing it back to the UART device. If the character was in the ranges A-Z or a-z, switch the case (upper -> lower or lower -> upper) before sending it back to the putty screen. All other ASCII codes can be echoed exactly as they were received.

Some things to consider as you build the program:

a. The LCD program from Lab 11 is a good starting point for your new program.
b. You will need to include another *.h file as indicated in the RS232 pdf file.
c. The only functions listed in the serial port pdf file that you will need are: alt_up_rs232_write_data(), alt_up_rs232_read_data(), and alt_up_rs232_open_dev(). The other functions will not be needed for this application, although you may use them.
d. Don't print error messages if the device doesn't open properly – just use the debugger to see what the return value (register R2) is after the call to the open_dev function when you want to verify the device properly opened.
e. You will need to repeatedly call the alt_up_rs232_read_data() function (polling) to see if a new character has come from the PC. The only indication a new character has been received is if the return value of the function call is 0 (success = new character received or RVALID was 1). If no new character has come in, the function returns -1 (failure = nothing new has been received or RVALID was 0).
f. Your program will have to declare two variables (declared as the alt_u8 type) to hold the data and parity information when the alt_up_rs232_read_data() function is called. The function expects that the addresses of those two variables are passed as the second and third arguments (pointers are being passed). If the function is successful in finding a new character, those two variables are updated with that information. You don't have to do anything with the parity argument/value, but the function expects it to be passed as an argument.

STEP #4A

- See template "TODO" comments
- Requirements:
    - When a letter A-Z is typed in Putty…
        - Write letter to LCD
        - Echo lower case letter back to Putty
    - When a letter a-z is typed in Putty…
        - Write letter to LCD
        - Echo upper case letter back to Putty
    - When any other "non-special" ACII character is typed in Putty…
        - Write character to LCD
        - Echo character back to Putty
    - For "special" ASCII codes…
        - Behavior is not defined
        - Consider making "LF" and "CR" cause an LCD init instead of writing a character to the LCD (optional)

Test your application. Use SignalTap to capture the activity on the RXD, LCD, and TXD signals for a case when a letter A-Z or a-z was entered in the putty screen. The SignalTap printout should show 1) the receiving of the character on the RXD signal, 2) the sending of the case-switched character on the TXD signal, and 3) the sending of the character to the LCD display. Print out the screen capture and hand-write comments that clearly indicate where those 3 actions take place.

STEP #4B

- Testing:
    - Check if each requirement is satisfied (repeatably)
    - Capture SignalTap screenshots per Lab 12 instructions

Hand In:

1. Completed table as provided on the last page of this handout
2. Note that you do NOT have to turn in printouts of the Part III SignalTap captures.
3. Part IV screen capture printout of the sending and receiving of the characters (UART + LCD)
    a. …With hand-written comments about what was happening (e.g. what direction data is moving, what characters are being sent and/or displayed, etc) .
4. Nicely formatted printout of your C-code for Part IV (single-spaced, fixed-width font (e.g. Consolas), point size of 10, few long lines of code that have to wrap on the page, and as much code on each printed page as possible).

Lab 12 Worksheet **Name: ___________________________**

UART Examples

Part II	
Sending Characters to the PC	
Enter the ASCII codes for your first name as a sequence of HEX values	0x , 0x , 0x , 0x , 0x ,
Enter each HEX byte into memory location 0x8020, one at a time. What shows up on Putty?	
Receiving Characters from the PC	
(Note: be sure to clear the receive queue before starting this step by hitting the "refresh" button until the RVALID bit is cleared.) At the PC keyboard, type the following: "2468<enter>" where <enter> is the enter key on the keyboard (be sure to only press those 5 keys). Then, click on the "refresh memory" button on the debugger. What values do you see for RVALID, Data, and RAVAIL?	RVALID = Data = RAVAIL =
Explain what those values mean.	
Click on the "refresh memory" button on the debugger again. What values do you see for RVALID, Data, and RAVAIL?	RVALID = Data = RAVAIL =
Click on the "refresh memory" button on the debugger again. What values do you see for RVALID, Data, and RAVAIL?	RVALID = Data = RAVAIL =
Click on the "refresh memory" button on the debugger again. What values do you see for RVALID, Data, and RAVAIL?	RVALID = Data = RAVAIL =
Click on the "refresh memory" button on the debugger again. What values do you see for RVALID, Data, and RAVAIL?	RVALID = Data = RAVAIL =
Click on the "refresh memory" button on the debugger again. What values do you see for RVALID, Data, and RAVAIL?	RVALID = Data = RAVAIL =
Explain what those values mean.	
What is the ASCII code used for the <enter> key? (in hex)	0x
Part III	
In which order are the bits of a byte sent? Is the most significant bit (MSb) sent first (bit7) or the least significant bit (LSb) (bit0)? You may need to try different ASCII codes to figure it out.	Circle One: MSb first LSb first