Lab 5: Interrupts, Serial Communication, External Memory

OBJECTIVES

In this lab you will learn how interrupts function, how to utilize serial communication, and to how to interface external memory, specifically on the F28335. To learn interrupts on the DSP, you will learn how to use the Peripheral Interrupt Expansion (PIE). You will communicate with your computer with the serial communication interface (SCI) on the DSP. Lastly, you will add external memory onto your DSP board.

REQUIRED MATERIALS

- You WILL need the following documentation:
    - System Control and Interrupts Reference Guide (sprufb0c.pdf)
    - Serial Communications Interface (SCI) Reference Guide (sprufz5a.pdf)
    - CY62256 Datasheet
- Wire wrap tool, UF DSP Board, **two** USB cables
- Br@y++ Terminal (provided on website)
- 32K x 8 CY62256 SRAM (provided)
- QUARTUS II, USB Byte Blaster
- 0.01 µF ceramic capacitor

INTRODUCTION

Note: Do <u>**NOT**</u> wait until the night before to attempt this lab, you <u>**WILL**</u> fail miserably … and lose a night of sleep.

Read carefully the <u>ENTIRE</u> lab document before you get started!!

In the first part of the lab, you will create your first completely interrupt driven program for this course. To do this you will break this down into two steps. Approaching a bigger problem is easier to do by breaking it down into multiple smaller steps. First you will configure the Serial Communication Interface (SCI) and Br@y++ terminal program and use them to send characters from your computer to display on your LED's via polling. Note you can always use the LED's in this way for debugging purposes. In the next part of the lab, you will write a program to receive a character from your computer and then send it back to your computer for viewing, all via an interrupt service routine.

In the second part of the lab, you will add a 32k x 8 external SRAM into the DSP memory map. You will place the SRAM in **ZONE7** of the memory map, starting at address **0x2C8000.** We would like to have only one image, therefore you will need to do **full address decoding.** Once you have your SRAM interfaced, you will then write a simple test program to test its functionality.

Lastly, you will transfer an entire JPEG image ('tebow_322.jpg') from your computer into your SRAM. To do this, you will combine what you have learned from parts, A, B, and C. The data file that you will transfer to your SRAM will be provided for you. This part of the lab will most likely take you the longest, so do not take any shortcuts in understanding Parts A, B, or C.

PRELAB REQUIREMENTS

NOTE: Prelab requirements MUST be accomplished **PRIOR** coming to lab. Your TA will not allow you to enter lab without the prelab complete.

<u>PARTA: SERIAL COMMUNICATION VIA POLLING</u>

In this part of the lab, the goal is to transmit a character from your computer by a "key press" and to display this character on your LED's array. You will use a terminal called Br@y++ to transmit the characters. Read through the SCI documentation and determine all of the registers that you will need to use.

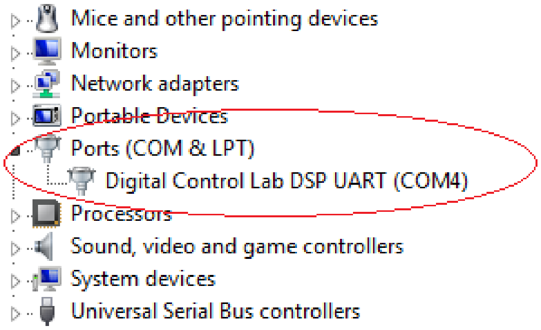
Figure 1 (on the last page) shows that the SCI on the DSP is connected to an FTDI USB controller. Since there are multiple SCI modules on the DSP, the only thing you should take away from this schematic is that the FTDI USB controller is hard wired to use <u>**SCIB**</u>**.**

Write a program that:

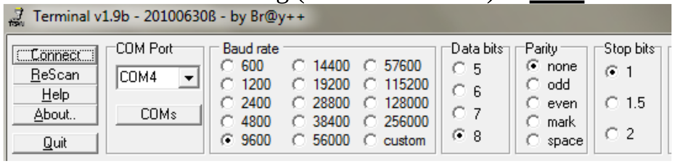
1. Configures the SCI module for the following:
    a. **9600 BUAD**
    b. **8 Data Bits**
    c. **No Parity**
    d. **1 Stop Bit**
2. Constantly check the receive buffer flag's status to see if a character has been received. (This is known as polling.)
3. Display the received character ASCII value on your LED's.
4. Repeat this forever.

After you write your program, you will need to configure the Br@y++ terminal as follows:

1. With your UF DSP Board powered, connect the additional USB cable up to the DSP UART port on your board. Once you plug it into your computer, Windows may ask you to install the drivers, you will need to verify that the FTDI drivers have installed correctly. To do this:
    a. Right click **My Computer**
    b. Select **Properties**
    c. Click **Device Manager**
    d. Check to see if there is a COM PORT associated with the UART:


Mice and other pointing devices
Monitors
Network adapters
Portable Devices
Ports (COM & LPT)
Digital Control Lab DSP UART (COM4)
Processors
Sound, video and game controllers
System devices
Universal Serial Bus controllers

2. Open up Br@y++ terminal and match the settings that your wrote in your program, and the COM Port. Note: Set handshaking (not shown below) to **none**.


Terminal v1.9b - 20100630ß - by Br@y++

COM Port: COM4

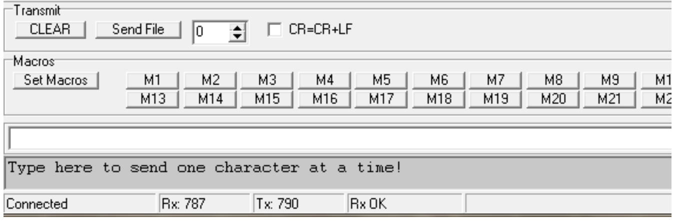
Baud rate: ○ 600 ○ 1200 ○ 2400 ○ 4800 ● 9600 ○ 14400 ○ 19200 ○ 28800 ○ 38400 ○ 56000 ○ 57600 ○ 115200 ○ 128000 ○ 256000 ○ custom

Data bits: ○ 5 ○ 6 ○ 7 ● 8

Parity: ● none ○ odd ○ even ○ mark ○ space

Stop bits: ● 1 ○ 1.5 ○ 2

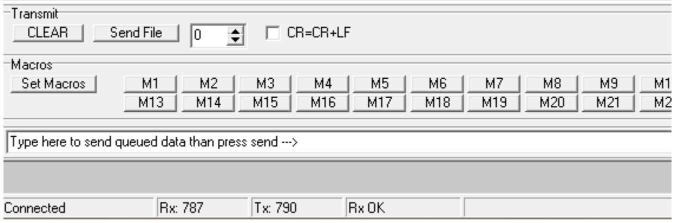
3. Select “Connect.” If everything is installed correctly, you should **NOT** get a warning, and be ready to transmit and receive data.
4. There are two ways you can transmit data to your DSP:
   a. **Single Key Press**: Place your cursor at the bottom of the Br@y++ terminal in the **Transmit** section and begin typing:


Type here to send one character at a time!

Connected | Rx: 787 | Tx: 790 | Rx OK

Transmit
CLEAR
Send File
0
CR=CR+LF
Macros
Set Macros
M1 M2 M3 M4 M5 M6 M7 M8 M9 M1
M13 M14 M15 M16 M17 M18 M19 M20 M21 M2

   b. **Send Queued Data:** Place your cursor in the white box and type a message; when done press “-> Send’ on the right side (not shown below) to send the entire message:


Type here to send queued data than press send --->

Connected | Rx: 787 | Tx: 790 | Rx OK

Transmit
CLEAR
Send File
0
CR=CR+LF
Macros
Set Macros
M1 M2 M3 M4 M5 M6 M7 M8 M9 M1
M13 M14 M15 M16 M17 M18 M19 M20 M21 M2

The characters are transmitted in ASCII through the SCI, so when you display your output on the LED, it should be the corresponding ASCII value of the key press. Example, you press a “7” on your computer, 0x37 should show up on your LED’s.

PARTB: SERIAL COMMUNICATION VIA INTERRUPT

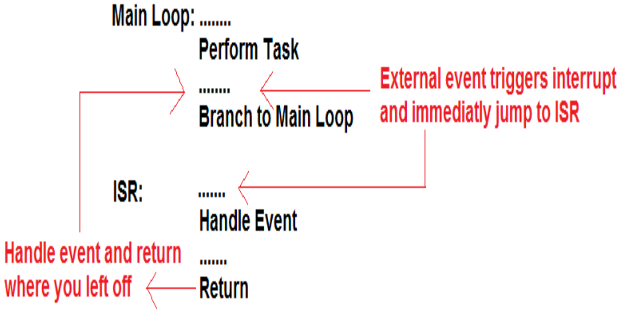
In this part of the lab, we would like to take Part A one step further. The goal is to transmit data from your computer, this time you will not display it on your LED’s, but echo it back to the computer for display in Br@y++ Terminal. Additionally, you will write this using an interrupt service routine, **NOT** polling.

Read over the PIE documentation and determine all of the registers that you will need to use to configure interrupts. Additionally, re-read the SCI documentation and determine any new registers that you did not use in Part A that you will need in Part B.

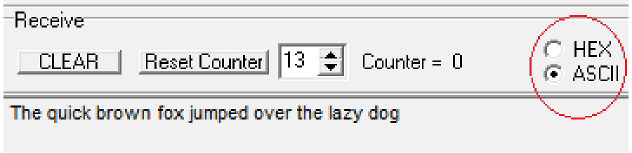
Write a program that:

1. Configures the SCI module for the following:
   a. **9600 BAUD**
   b. **8 Data Bits**
   c. **No Parity**
   d. **1 Stop Bit**
2. Generate an interrupt when a character has been received and branch to an interrupt service routine.
3. In the interrupt service routine, send back the character to the Br@y++ terminal for viewing.
4. To demonstrate the concept of an interrupt driven program, toggle an LED with a 1/4 second delay in the main loop, and nothing else!
5. Repeat forever.

A properly interrupt driven program should have the following format:


Main Loop: ....
Perform Task
....
Branch to Main Loop
External event triggers interrupt and immediatly jump to ISR
ISR:
....
Handle Event
....
Return
Handle event and return where you left off

When Br@y++ terminal receives data, it will be displayed in the receive section, the data can be viewed in **ASCII or HEX**:


The quick brown fox jumped over the lazy dog

Counter = 0

○ HEX ● ASCII

PARTC: ADDING EXTERNAL MEMORY

In this part of the lab, you are going to add external memory to the DSP memory map. The goal is to interface the SRAM, and then write a simple program that will fill up the contents of the SRAM to verify its functionality.

- **NOTE:** You SHOULD already have your SRAM soldered onto your board at this point.
- **IMPORTANT:** In the previous lab, you were not allowed to use the **XZCS0N** signal when adding your ports. What addresses, if any, in ZONE7, are affected by reads or writes to your I/O ports? Since

we will be using a ZONE7 in this lab for your SRAM, we do not want any read/writes in ZONE7 with your I/O ports. You will therefore **NEED** to add **XZCS0N** to your decode equation in Quartus II for your I/O ports. (If A21 and A20 were available on your board, this would **NOT** have been necessary.)

BEFORE you wire wrap, perform a continuity test with a multimeter to ensure that you have solid connections to your SRAM.

SOLDER a **0.01 uF** ceramic capacitor between VCC and GND of the SRAM.

Read through the datasheet of the 32k x 8 CY62256 SRAM and look at the pin out diagram.

1. Place the 32Kx8 SRAM in **ZONE7** starting at **0x2C8000.**
2. To have **ONE** image, you will need to do full address decoding. You will need to create the necessary decode circuit in QUARTUS II (schematic or VHDL) and place it into your CPLD.
3. You **MUST** use **XZCS7N** in your decode circuit.
4. Draw the wiring diagram on paper and find the functional block diagram of the SRAM in the datasheet and **VERIFY THAT YOU WILL NOT BLOW UP THE SRAM.** If you do blow up your SRAM, you will have to buy another one and you will lose significant lab points.
5. Create a timing diagram for the SRAM and the DSP, show a read and a write cycle.
6. Wire wrap the SRAM to your UF DSP board.
7. To test the functionality of the SRAM you will write a program that does the following:
    a. Create a subroutine that writes 0x55 to **EVERY** address in the SRAM.
    b. Once you have written this data, go back and read each value, one at a time, and check to see if the value is in fact 0x55. If you make it through the entire SRAM without any errors, place 0x55 on your LED's; otherwise place 0xFF on your LED's.
    c. Create a subroutine that writes 0xAA to every address in the SRAM.
    d. Once you have written this data, go back and read each value, one at a time, and check to see if the value is in fact 0xAA. If you make it through the entire SRAM without any errors, place 0xAA on your LED's; otherwise place 0xFF on your LED's.
    e. For the TA to check this, they will ask you to place a break point in your code after you fill up the SRAM with either 0x55 or 0xAA, they will pick an address at random and corrupt the value. They will then ask you to finish running your program to see if it catches the error check.

PARTD: LOADING AN IMAGE INTO MEMORY

In this final part of the lab, the goal is to transfer the "tebow_322.jpg" image file from your computer into your SRAM. To do this you will use the Br@y++ terminal and the "tebow_data.txt" file that contains the image data.

Many images used today use a 24 bit color map: that is 8 bits for red, 8 bits for blue, and 8 bits for green (also known as RGB888).

$R_7R_6R_5R_4R_3R_2R_1R_0$ | $G_7G_6G_5G_4G_3G_2G_1G_0$ | $B_7B_6B_5B_4B_3B_2B_1B_0$

For each pixel, this gives us $2^8 * 2^8 * 2^8 = 2^{24} = 16.7$ million color combinations! Since our external SRAM only has an 8 bit data length, it would be difficult to store a 24 bit image into memory without sacrificing resolution. To get around this, a MATLAB program was written to convert a RGB888 image into an 8-bit JPEG image, or RGB332. You will load the 8 bit image into your SRAM. Using the 24 bit map, we can take the most significant bits:

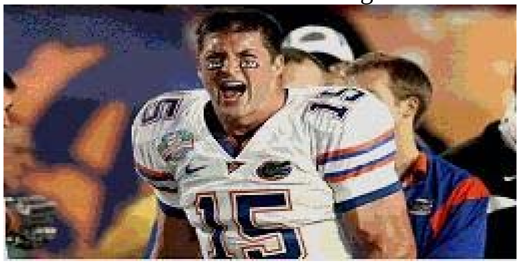
$R_7R_6R_5$XXXXX | $G_7G_6G_5$XXXXX | B_7B_6XXXXXX

This can be expressed in 8 bits as:

$R_2R_1R_0$ | $G_2G_1G_0$ | B_1B_0 or $R_7R_6R_5$ | $G_4G_3G_2$ | B_1B_0

So a RGB888 image such as:

Would look like this as a RGB332 image:



Not too bad! We would like to transfer the above "tebow_322.jpg" image into our SRAM. Since the resolution of the image is 256 x 128 pixels, you will fill up the entire contents of your SRAM! The file "tebow_data.txt" file contains the extracted JPG data information. The file contains **2048** rows of **16 unsigned 8-bit** numbers in hexadecimal. You are encouraged to open the text file and scroll through it. Each byte is

Lab 5: Interrupts, Serial Communication, External Memory

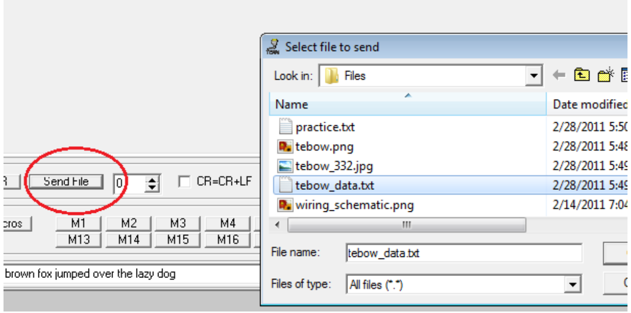
separated by a space, and at the end of each row there is a space and a newline character. **We are not interested in the white spaces, new lines, OR carriage return characters.**

Write a program that does the following:

1. Configures the SCI module for the following:
    a. **128000 BAUD**
    b. **8 Data Bits**
    c. **No Parity**
    d. **1 Stop Bit**
2. Use Br@y++ terminal to initiate the file transfer. The file will begin to send one byte at a time (corresponding to the ASCII for each hex nibble) at the appropriate speed until the file has been completely transferred.
3. Generate an interrupt when a character is received.
4. In the interrupt service routine, create a method for transferring the sampled data into the SRAM.
5. This program needs to be completely interrupt driven. Again to verify that this actually works, toggle an LED with every 1/4 second in the main loop. There should be nothing else in your main loop.
6. Once the file has transferred, verify that the contents of the SRAM looks EXACTLY like that of the **tebow_data.txt** file. **It should take approximately 35 seconds to transfer the entire file!**
7. Before you attempt to send the entire "tebow_data.txt" file, a smaller "practice.txt" file that only contains two lines of data has been provided to test your code. A great way to see how the file transfer works is to use Part B of the lab, and transfer the file so you can view the incoming data in both ASCII and hex.

Remember, the Br@y++ terminal transfers data in ASCII, and **NOT** hex.

To initialize a file transfer using Br@y++ Terminal:



Look in: Files

Name	Date modifie
practice.txt	2/28/2011 5:5
tebow.png	2/28/2011 5:4
tebow_332.jpg	2/28/2011 5:4
tebow_data.txt	2/28/2011 5:4
wiring_schematic.png	2/14/2011 7:0

File name: tebow_data.txt
Files of type: All files (*.*)

brown fox jumped over the lazy dog

The number that is directly to the right of the 'Send File' box is a millisecond delay, make sure that is a 0 or you will be waiting for a while to fill up your SRAM.

PRELAB QUESTIONS

1. What does the peripheral clock that is connected to SCI directly affect?
2. How does an interrupt service routine work?
3. What happens if the BAUD rate is different between two communicating devices?
4. Why did you place a 0.01 µF capacitor between VCC and GND of the SRAM?
5. What is the range of the SRAM that you added in the DSP memory map?
6. How many bits is an ASCII character?
7. Why do you need the FTDI bridge?
8. What is the total time that the file transfer should require. Hints: At the end of each line is a space, a carriage return and a line-feed. There are two ASCII values representing a single hex value. A single space is inserted between each set of ASCII values (representing a single hex value).

PRELAB PROCEDURE

1. Email the following to the class gmail account:
    a. Your list files to from parts A,B,C and D.
    b. Your Quartus archive file from part C.
2. Bring the following printed document to turn in to your TA:
    a. Answer the prelab questions.
    b. Pseudo-code/flowcharts for parts A,B,C and D.
    c. The timing diagrams from Part C

LAB PROCEDURE

1. Demonstrate Part A to the TA. If it does not fully work, show your TA the code that you have completed, and explain the possible errors.
2. Demonstrate Part B to the TA. Place a breakpoint in the interrupt service routine to show the TA that you used an ISR.
3. Demonstrate Part C to your TA. The TA will verify that the SRAM is functioning correctly.
4. Demonstrate Part D to your TA. The TA will verify that the SRAM contents contain the exact data of the **tebow_data.txt** file. If this portion of your code is not working, demonstrate what you have and explain the possible errors.

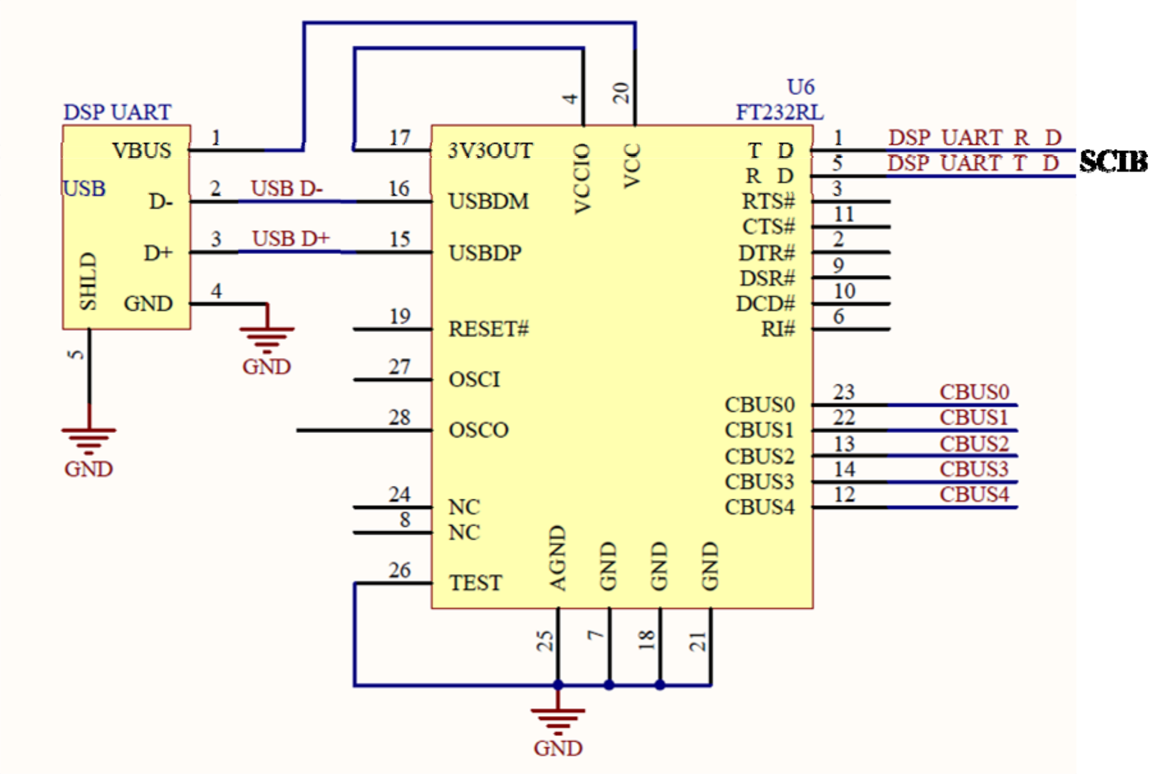


Figure 1: FTDI Schematic for UF DSP Board