Wilson Au-Yeung

CMPE 121

Anujan Varma

October 29, 2016

Lab Report 3

Introduction:

This lab is comprised of four sections, each of which were focused on teaching us the basics of using and getting familiar with the UART component on the PSoc-5LP Development kit. In order to familiarize ourselves with the UART component, we had to study the data sheet carefully to understand the configuration options. The four parts mostly focused on seeing how it was able to transfer data from sender to receiver using multiple different methods. The first section of the lab was to send and receive data using the UART without interrupts. The second part comprised of the same action using Rx and Tx interrupts. And the third part we saw how the UART behaved with and without hardware flow control. And lastly the last part required us to program a UART data transfer between two different Psoc Boards, while simultaneously sending and receiving signal.

Procedure:

Part 1: Transmit / Receive Based on Polling

This part did not require the use of UART interrupts, so we turned off all Rx and Tx interrupts. First part of this section was to transmit the byte value 0x45 into an output pin and identify all the start and stop bits, data bits, and parity. The rest was to transfer all data in SRAM memory from one array to another. So to do this we had to set up an array of size 4096 and have an increasing pattern from 0 – 255. The other array just held an empty array. I created a for loop so I could access the data stored in the array 4096 times.

```
for(i = 0; i < 4096; i++){
    sourceArray[i] = i%256;
}


for(i=0; i < 4096; i++){
    destinationArray[i] = 0;
}
```

Figure 1: Setting up the Array

Transfer & Receive:

Inside that for loop, I used a while loop to check when the FIFO was full so it would not overflow and transfer any data. It would continue to store the BitMask into a variable until it is not full. I created a while loop stating as long as the process is not complete, stall time. Then it should copy the data into the destination array as long as it's not empty. I started the timer before the for loop and stopped it immediately after so I could time how long the UART process is.

```
for(i = 0; i < 4096; i++){
    FULL = UART_ReadTxStatus();
    while(FULL == UART_TX_STS_FIFO_FULL){
        FULL = UART_ReadTxStatus();
        //stall time
    }
    if(UART_TX_STS_FIFO_NOT_FULL){
        UART_PutChar(sourceArray[i]);
    }

    while(~UART_ReadTxStatus() & UART_TX_STS_COMPLETE);
    {
        //Stall Time
    }

    destinationArray[i] = UART_GetChar();
}
```

Figure 2: When to transfer and receive

Part 2: Transmit / Receive Using Interrupts

To set up this part of the lab, I had to turn on the UART Rx and Tx interrupts. Setting up the array followed the exact same steps as Part 1. The only difference here in this part was that it

waited for built in conditions in the Rx and Tx Interrupts before transmitting data and receiving data. By making use of the interrupts, it allows less software overhead when using the UART.

Transfer & Receive:

Using the internal UART interrupt services, we were able to check (in real time) when each of the conditions we checked earlier occurs. For the Transmit (Tx), I was able to determine when start the transfer every time it was empty. Because I chose to use this particular interrupt, I never have to check for overflow conditions because it the max number of transfers in the FIFO is one. It would check every time there is a queue in the FIFO, and if there still is, it would store the value inside the destination array.

```
for(i = 0; i < 4096; i++)
{
    while(sourceISR == FALSE){
        //stall
    }
    if(sourceISR == TRUE){
        sourceISR = FALSE;
        UART_PutChar(sourceArray[i]);
    }

    while(destinationISR == FALSE){
        //stall
    }
    if(destinationISR == TRUE){
        destinationISR = FALSE;
        destinationArray[i] = UART_GetChar();
    }

}
```

```
sourceISR= TRUE;
NumISR++;
```

Figure 3: ISR transfers and receive

Part 3: Hardware Flow Control

This part of the lab was a little bit like part 1 and part 2 combined. We were allowed to use the Tx portion of the code from part 2 to know when it should continue to transfer and when not to but we had to turn off our Rx interrupt when receiving data. The method to know when to receive data was by implementing a timer onto our top design which was every 2 milliseconds. Running without flow control would give us a lot of errors because of the large amount of inputs transmitted and only having one receiving action per 2 milliseconds. We then switch the flow

control to be handled by hardware by enabling the flow control and connecting the RTS and CTS pins together.

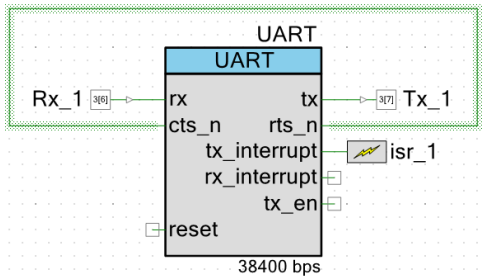

Figure 4: Hardware Flow Control

Part 4: Data Transfer with Remote System

This final part of the lab required us to connect our UART to a remote system. In this part of the lab we were told to simultaneously transfer and receive data from another system. I used everything I learned from parts 1 -3 on this part. I used an ISR to transmit data every time the FIFO was empty. And as long as there was a queue in the FIFO, it was receive data into the array. Since the first received data could be a random number, I checked what number I was receiving and immediately after I would start transmitting the number received + 1 back. So after the first error, it should automatically be in sync.

Results:

Part 1: I was able to check for any inconsistencies between the two arrays and they all seemed to match up. I forced in different values to check if the error count worked, and it did.

Part 2: The ISR counter that counted the number of times that each ISR ran for was 4096. This proves that there was an interrupt that was generated with each byte that was sent and received.

Part 3: I first ran the program without using hardware control and 4080 bytes were incorrectly transmitted. Because the UART kept overwriting over the data it had just transmitted without being received. So it would send those numbers and random numbers would be chosen into the received array. Using the hardware flow control allowed us to achieve 0 errors.

Part 4: I was able to put the two numbers in sync when being transferred and received after one error by incrementing the first received number by one.

Conclusions

The purpose of the lab was to understand how the UART works and how to use its configuration to manipulate memory and where to transfer it. I figured out how thoroughly you have to read the datasheets as it would leave a lot of hints for you to progress on. I also enjoyed that we are finding more creative and innovative ways to design our schematics. I also really appreciated that it gradually built on what you learned previously and you had to really grasp the previous section to really progress on.