CMPE 121L: Microprocessor System Design Lab

Fall 2016

Lab Exercise 3

Check-off Due in lab section, week of October 24, 2016

Report due: Within 3 days after checkoff due

In this exercise, you will learn how to use the capabilities of the UART in PSoC 5. Using PSoC Creator, add a UART to your design and study its data sheet. Configure the UART as follows:

- Full-duplex UART, 38400 baud, 8-bit data, odd parity, 1 stop bit, no hardware flow control, internal clock, 16x oversampling.
- Configure the sizes of the TX and RX FIFOs as 4 bytes each.
- Bring out serial transmit and receive data pins to one of the connectors and loop the data back through an external wire.

Part 1: Transmit/Receive Based on Polling

1. Disable the UART interrupts.
2. Write a program to transmit the byte value 0x45 continuously. Connect the serial data output to the oscilloscope and view the waveform. Copy the waveform into your notebook and submit it with the report. Identify the start and stop bits, data bits, and parity.
3. Define two byte arrays in the SRAM memory, each of size 4096 bytes, one for the transmit data and the other for the receive data.
4. Initialize the transmit array to an increasing pattern of bytes 00, 01, …
5. Clear the receive array to all zeroes.
6. Write a program in which a single software loop transmits data from the transmit array and also stores any data received by the UART into the receive array. The program should not use any interrupts. Instead it should poll the UART status registers to determine if the status of the transmitter and receiver FIFOs. The program should read a byte of data from the RX FIFO when it is found to be non-empty, and it should add a byte to the TX FIFO when it is not full.
7. After all the data has been received, compare the received data byte by byte with the data in the transmit array and display the number of errors found on the LCD display.
8. Add a hardware timer block to your design. Start the timer on receiving the first byte on the RX side of the UART and stop it on receiving the last byte. Compare the elapsed time from the timer to the estimated time based on the 38,400 baud rate. Do they agree? If not, what is the reason for the discrepancy and how will you fix the code to get to the theoretical transfer time?

CHECKOFFS:

1. Show the waveform on the oscilloscope when transmitting 0x5a continuously and identify the start, stop, data and parity bits.
2. Demonstrate the block transfer program of steps 3–7.
3. Show the estimated time and the actual time taken to transfer the block from step 9.

Part 2: Transmit/Receive Using Interrupts

This is similar to Part 1, except that you will make use of the interrupts to reduce the software overhead in using the UART.

1. Enable the TX and RX interrupts. You need to determine the interrupting conditions that are needed for your solution. The objective is to minimize the total number of interrupts generated during the execution of the program.
2. As before, define two byte arrays in the SRAM memory. Initialize the transmit array to an increasing pattern of bytes 00, 01, …
3. Clear the receiver array to all zeroes.
4. Write the ISRs to deal with the TX and RX interrupts.
5. The main program should set up the UART and start the transmission of bytes from the transmit array. It should then stay in an idle loop, waiting for all the data to be received (based on a status flag posted by the ISR for the RX interrupt). The idle loop should not access the UART directly.
6. After all the data has been received, compare the received data byte by byte with the data in the transmit array and display any errors on the LCD display.
7. Using counters in your program, determine the number of times the ISR was entered for both TX and RX.

CHECKOFFS:

1. Demonstrate the block transfer program of steps 1–6.
2. Show the counts from step 7.

Part 3: Hardware Flow Control

In this part, you will add hardware flow control to Part 2, thus preventing receiver FIFO overflows when the receiver software is not able to process that data at the rate it is being received by the UART.

1. Disable the RX interrupt and Enable only the TX interrupt.
2. As before, define two byte arrays in the SRAM memory. Initialize the transmit array to an increasing pattern of bytes 00, 01, …
3. Clear the receive array to all zeroes.
4. Write the ISRs to deal with the TX interrupts (on interrupt, transmit the next byte from the transmit array if the UART TX FIFO is not full).
5. Add a timer to the design and configure it to generate an interrupt every millisecond.
6. Write the timer ISR, which checks the RX FFIFO of the UART and reads any data present to the receive array. Also check the UART receiver status and record any errors found in an error flag.
7. The main program should set up the UART, enable the TX interrupt, and stay in a loop.
8. Run the program. After all the data is transmitted, compare the contents of the transmit buffer with the receive buffer. Do you see any errors? Why? Check the error flag and see if the UART reported any errors.

9. Enable hardware flow control in the UART and bring out the CTS_N input and RTS_N output to pins. Loop RTS_N to CTS_N through a wire. Connect this signal also to a pin driving an LED on the board, so that you can observe the activity on the RTS_N output.
10. Repeat the same experiment. Observe the activity of the LED. Have the errors gone away? Why?

CHECKOFFS:

1. Demonstrate the program with and without hardware flow control.
2. With hardware flow control, show the RTS_N output waveform on an oscilloscope.

Part 4: Data Transfer with Remote System

In this part, you will connect your UART to a remote system, continuously transmit and receive data, and measure the performance. You will develop the code on your own, but will need to work with a partner to demonstrate it.

1. Set up the UART with the following parameters: full-duplex, 38400 baud, 8-bit data, odd parity, 1 stop bit, no hardware flow control, internal clock, 16x oversampling. Configure the sizes of the TX and RX FIFOs as 4 bytes each. Enable TX and RX interrupts.
2. Bring out serial transmit and receive data pins to one of the connectors. For testing with a remote system, connect your TX pin to the RX pin of the remote system and vice-versa (you also need a common ground). Most of the testing, however, can be done without a remote system by looping the data back from TX to RX.
3. Your program should simultaneously transmit and receive data continuously at the speed of the serial link.
    - The transmit side should continuously send bytes with increasing values 00, 01, …, 0xff, and repeating the sequence.
    - The receive side should receive each byte, check against its expected value, and display any errors on the LCD display. Note that, when your program starts up, the first byte received can be of any value, so it should first wait for an error-free byte to be received and from this point, start checking subsequent bytes based on the increasing pattern.
    - The program should calculate the transmit and receive throughputs (in kbytes/second) approximately every second and update them on the LCD display. It should also show the total number of errors found. The errors include mismatches with the expected value, parity errors, framing errors and receive FIFO overruns (the error count can be made saturating with an upper limit of 1,000. If you are seeing too many errors, there is clearly something wrong with your hardware or software.).
4. Test your program in loopback first, then with a remote system. Ideally, you should achieve close to the theoretical throughput in both cases and should not see any errors.

CHECKOFFS:

1. Demonstrate the program transmitting and receiving data continuously with a remote system at close to the maximum theoretical throughput

What to submit?

- Schematics and code for all parts
- Descriptions of your designs
- Answers to the questions in the description above.
- Results
- For Part 2, state your result on the number of times the ISRs for TX and RX were entered.