CMPE 121L: Microprocessor System Design Lab
Spring 2019
Lab Exercise 3

Check-off Due in lab section, week of April 29, 2019
Report due: May 6

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 to pins and loop the data back through an external wire by connecting the TX output to the RX input.

You will also need to connect the LCD display as explained in Lab 1.

Part 1: Transmit/Receive Based on Polling

1. Disable the UART interrupts.
2. Write a program to transmit the byte value 0xa5 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, … That is, set the *i*th element of the array to *i* modulo 256.
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 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. Display the elapsed time on the LCD display. 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 close to the theoretical transfer time?

CHECKOFFS:

1. Show the waveform on the oscilloscope when transmitting 0xa5 continuously and identify the start, stop, data and parity bits.
2. Demonstrate the block transfer program of steps 3–8.
3. Show the estimated time and the actual time taken to transfer the block from step 8.

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 approximately at intervals of 0.5 millisecond.

6. Write the timer ISR, which checks the RX FIFO of the UART and reads any data present, and adds it 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. Use the LCD display to indicate any errors. Also check the error flag and if the UART reported any errors. Do you see any errors? Why?
9. Increase the timer period to 1.1 milliseconds and repeat the experiment. Do you see any errors now? Why is the behavior different for the two different timer settings? Estimate the maximum period the timer can be set to without seeing any errors (you don't need to test this experimentally).

10. 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 the blue LED, so that you can observe the activity on the RTS_N output.
11. Repeat the same experiment. Observe the activity of the LED. Show the total number of data mismatches on the LCD display. 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.
3. With hardware flow control, show that the number of errors is 0.

Part 4: Data Transfer between Two UARTs

In this part, you will form a link between two UARTs, continuously transmit and receive data across the link, and measure the throughput. You could connect two boards to achieve this, but we will take a simpler approach and use two UARTs on the same board to form the link. We will call one of them the primary UART and the other the secondary UART.

1. Set up both UARTs 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 the serial transmit and receive data from both UARTs to the GPIO pins. Connect the TX pin of one of the UARTs to the RX pin of the other board and vice-versa.
3. Start with the primary UART first. Write a program to simultaneously transmit and receive data continuously from the primary UART 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.

   - To test the primary UART alone, loop the data back from its TX output to the RX input and verify that the data is received with no errors.
   - Now add the secondary UART. Enable its RX interrupt, but keep the TX interrupt disabled. In the receive ISR, read data from the RX FIFO and add it to the TX FIFO. This will make the secondary UART echo the data from the primary UART to itself. Connect the two UARTs and test to make sure the primary UART is receiving the data back with no errors.
   - Calculate the receive throughput of the primary UART (in kbytes/second) approximately every second (by dividing the number of bytes received over the one-second window) and update them on the LCD display. The LCD display should refresh every second with the current throughput observed. 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 should be made saturating with an upper limit of 1,000. If you are seeing too many errors, there is clearly something wrong with your design.).
   - Remove the jumper connecting the TX output of the primary UART to the RX input of the secondary UART temporarily and reconnect it. Is your primary UART able to recover? You might see some errors when the jumper is connected back, but the errors should stop eventually.

4. Compare the measured throughput to the theoretical maximum value. Ideally, you should achieve close to the theoretical throughput and should not see any errors after the receiver has synced up with the transmitter.

CHECKOFFS:

1. Demonstrate the program transmitting and receiving data continuously between the two systems at close to the maximum theoretical throughput and no errors. On removing the jumper wire connecting the TX output of one UART to the RX input of the other and reconnecting, the receiver should sync up with the transmitter.

Part 5: Receiver Clock Tolerance

One of the reasons for the popularity of UART is that it can transfer data reliably even when the clocks on both sides are not very accurate. This is because the receiver over-samples the data and re-aligns the sample points at the start of each byte (using the start bit). In fact, you can even use a software-generated clock to transmit and receive data if the baud rate is not too high. In this part, we will attempt to determine the maximum clock tolerance for the receiver to work reliably.

1. Read the section on clock tolerance in the UART data sheet.

2. Set up the primary UART in Part 4 to transmit bytes continuously with increasing values 00, 01, …, 0xff, and repeating the sequence. Configure the UART to use the internal clock and set the baud rate to 115,200 with 8-bit data, 1 stop bit and odd parity. Leave the receive side unconnected. Do not enable hardware flow control.
3. Configure the secondary UART with the external clock option. Set it up for 8-bit data, 1 stop bit and odd parity. Set the receiver for 8x oversampling and 2/3 voting. Connect a clock component to its clock input. Configure the clock component to use PLL_OUT (24 MHz) as its source. You can then adjust the divider value to generate various frequencies at the UART clock input.
   For a baud rate of 115200 and with 8x oversampling, the corresponding clock frequency is 115200 x 8 = 921.6 kHz. We can't generate exactly 921.6 MHz from the 24 MHz clock, but we can get very close by setting the divider to 26, which generates a 923 MHz clock (an error of only 0.15 percent).
4. Connect the TX output of the primary UART to the RX input of the secondary UART and transmit data from the primary UART continuously. Poll the receive side of the secondary UART continuously and display any errors found.
5. Run experiments with different clock divider settings on the secondary UART to determine how far you can increase or decrease the clock frequency, and still able to receive the data reliably. You should run the test for at least a few seconds without errors before concluding that the link is reliable.
6. Compare your results with the analysis in the datasheet and discuss your findings.

CHECKOFFS:

You do not need to demonstrate this part, but you should show your results from the test to the TA. What is the minimum and maximum frequency for the receiver clock for which the UART was able to receive the data without errors?

What to submit?

- Schematics and code for all parts
- Descriptions of your designs
- Answers to the questions in the description above.
- Results

EXTRA CREDIT

You should attempt this part only after completing all the previous parts. You can receive an extra credit of up to 20% if you successfully complete this part. You are on your own for this part and we will not be able to help with debugging your design.

In Part 5, we observed that a UART link can operate reliably even with a large clock tolerance. However, clock tolerance can sometimes lead to data loss if the receiver clock is slower than the transmitter clock.

For this part, configure the primary UART as a data pattern generator/checker.

- Full-duplex, 115200 baud, 8-bit data, odd parity, 1 stop bit, no hardware flow control, internal clock, 16x oversampling with 2/3 voting.
- Configure the sizes of the TX and RX FIFOs as 4 bytes each.

Attach the LCD display. Write a program to transmit the repeating sequence 00, 01, …, 0xff and simultaneously receive data and check for any missing data. You may use polling or interrupts to send data to the transmit FIFO and check for data from the receive FIFO. Include code to display any errors found, along with the time at which the first error was detected after the primary UART started receiving good data. Test your program by looping data from the TX to the RX pin of the primary UART and verify that there are no errors.

Now configure the secondary UART as follows:

- Full-duplex, 115200 baud, 8-bit data, odd parity, 1 stop bit, no hardware flow control, internal clock, 8x oversampling with 2/3 voting.
- Configure the sizes of the TX and RX FIFOs as 4 bytes each.

Keep the TX interrupt of the secondary UART disabled. Enable its RX interrupt and write code for the receive ISR to read data from the RX FIFO and add to the TX FIFO. The code in the ISR should read a byte from the RX FIFO, then check if the TX FIFO is full. It should simply discard the byte when the TX FIFO is full, otherwise it should add the data to the TX FIFO and return.

Now connect the two UARTs together and transmit data from the primary UART for several minutes to make sure that the primary UART does not detect any losses at its receiver.

Change the secondary UART configuration to use an external clock and configure as in Part 5. Set the clock component to divide the 24 MHz clock by 27, which results in a slower clock than before. Is the resulting frequency within the range for reliable operation in Part 5?

Run the system for a long time. Did the primary UART detect any loss of data after a while? Can you identify where the missing data was lost?

Knowing the frequencies in this test, can you predict how long it would take for the first error to appear after starting the test? Does this agree with your observation?

Can you think of a simple way to avoid the data loss in this case, without changing any of the frequencies? Implement your idea in the system and check if the system can operate without data loss for a long time.