**Microprocessor Systems Lab 3**

Serial Communication

Shane Conaboy

Justin Yost

ECSE-4790 Microprocessor Systems

October 19, 2015

**Introduction**

The purpose of this lab was to investigate and implement the use of asynchronous serial communication ports and the synchronous serial peripheral interface. The asynchronous communication was done with two Universal Asynchronous Receiver/Transmitters (UARTs) and the synchronous part through use of the SPI communication protocol. Using the UARTs, two separate ports were monitored simultaneously, taking user input from a keyboard and echoing the typed characters back to both ports. For the synchronous part, keyboard input was sent across a SPI network to another microprocessor and the characters were displayed on both terminals.

**Procedure**

The first and second parts of the lab involved connecting to two different serial ports on a computer and using UART0 and UART1 on the 8051 microcontroller to communicate through the ports. Since the 8051 board only has 1 serial port connection, another one had to be wired using input and output ports. Using an RS232 DB-9 Male connector and it’s respective pinout form shown in **FIGURE X**, UART0’s TX (P0.0) and RX (P0.1) lines were wired to the respective TX and RX lines on the connector. Ultimately, UART1 was to be used with the DB-9 connector, however the hardware was first verified using a program with a known UART0 setup. After the hardware was verified, the next step was to set up UART1.



Figure : RS232 DB-9 Male Pinout

UART1 was set up using Timer1 as its baud rate generator. The equation for the UART1 baud rate (Eq1) is shown below. Using (Eq1), a desired UART1 baud rate of 9600, and a SYSCLK rate of 22.1184Mhz, the T1 preload value was calculated as 0xA0.

Equation 1: UART1 Baud Rate Calculation

Since UART1 was using Timer1, UART0 was set up using Timer2 as its baud rate generator. The equation for the UART0 baud rate (Eq2) is shown below. Using (Eq2), a desired UART0 baud rate of 115200, and a T2CLK (SYSCLK) rate of 22.1184MHz, the T2 reload value, RCAP2, was calculated as 0xF4.

Equation 2: UART0, Timer2 Baud Rate Calculation

Once both UARTs were connected and communicating with their respective terminals, the next step was to have them monitor their ports simultaneously and output and received input to both ports. The first strategy, implemented in part 1 of the lab, was to continuously poll the Receive Interrupt Flag of both UARTs, RI0 for UART0 and RI1 for UART1. If one of the flags was set to 1, there was input data ready to be read from the respective data buffer, SBUF0 or SBUF1. When data was read from this buffer, it was sent back out through the same buffer and the other UART’s buffer to display to both terminals.

The second strategy of monitoring both ports, which was implemented in part 2, was to use interrupts for both UARTs. When the RI0 or RI1 flag was set by hardware, the program on the 8051 would vector to the appropriate interrupt service routine (ISR). Within the ISR, the same strategy was implemented from part 1: any received data from a data buffer would be sent back out through both UART buffers. Since the transmit receive interrupt flag (TI0 and TI1), which is set to 1 upon transmission of data, would also cause the ISR to be called, code had to be added within each ISR to determine what caused the interrupt and whether or not to read from the data buffer. Another concern was that the priority of a UART0 interrupt is higher than that of a UART1 interrupt, and if this was left unchanged, the program would never detect data on the UART1 port. To remedy this, an arbitrary delay was added to main routine, and after each delay, the priority of the UART1 interrupt would be toggled between high and low. Essentially, the program would toggle between listening to the UART0 port and listening to the UART1 port.

Part 3 of the lab involved communicating with an external device, a 68HC11 EVB, through the 8051’s SPI. To initially get the SPI working, it the MOSI pin was wired to its own MISO pin so data sent out would be echoed back in. Both the output and input data were displayed on the ProComm Plus terminal and verified that they matched. Once confirmed that data could be sent and received, the 68HC11 was set up by downloading the given spislave.c program and wiring the appropriate pins on the 8051 to the indicated pins on the 68HC11. Similar to the beginning of this part, sent and received data would be displayed on the respective terminals.

**Results**

Ultimately, everything in the lab worked as intended, but not without overcoming many problems along the way. Initial configuration of the UARTs and their respective timers was tried using the Configuration Wizard, and while most of the setup was correct, the overflow value for the baud rates that the program calculated was incorrect. After almost a full day of trying to figure out why it was calculating incorrect values, it was decided to manually calculate the overflow values for each timer. After this, part 1 was easily implemented.

The major problem that was encountered in part 2 was enabling the UART interrupts correctly. Even though all the correct interrupts were enabled, while testing the code only on UART0, the UART with the highest interrupt priority, the program never vectored into the ISR. The problem was that timer interrupts were enabled because it was thought that in order for the UARTs to use the timers for the baud rates, the interrupts needed to be enabled. Timer interrupts have a higher priority than that UART interrupts, so the program would always choose to ignore the UARTs. Disabling the timer interrupts fixed the problem and enabled the UART ISRs to work correctly. Within the UART ISRs, detecting what caused the interrupts, either the RI or TI flag, and resetting the appropriate flag prevented the continues display of the input character.

The part 2 communication worked on one computer independently, however when connected with another team, the input characters from each computer would be output continuously. It was logically reasoned that any data coming into the 8051 would be echoed back to both ports, and since the UART1 port is connected to another 8051 (instead of a terminal), the echoed data would be seen by the other microcontroller as new data and be echoed back out again. This process would happen continuously, causing the infinite output of the input characters. Simply modifying the code so that data coming in on the UART1 port would not be sent back out on UART1 fixed the issue, and part 2 was successfully completed.

The main problem in part 3 was wiring the correct pins to the correct ports. Hooking the 8051’s SPI to itself was no issue once it was figured out what flags to monitor before reading the data from the input buffer. Adding the 68HC11 required careful attention to wiring, and after delays were added so the 8051 could wait for the slower 68HC11 to process the data and send it back, communication between both boards was established.

**Conclusion**

This lab investigated and implemented serial communication through UART and SPI methods. It heavily stressed how the microcontroller sets up the UART and SPI channels, namely how baud rates are generated, how buffers are utilized in data transfer, and the difference between synchronous and asynchronous communication. After many subtle problems, each part of the lab was implemented, establishing communication with the ProComm Plus terminal and the 68HC11 EVB.

**Appendix A: Baud Rate Calculations**

UART0 Timer2 Baud Rate Calculations:

65524 = 0xFFF4

Timer2 preload value low byte = **0xF4**

UART1 Timer1 Baud Rate Calculations:

Timer1 preload value = **0xA0**

**Appendix B: Circuit Schematic**

**Appendix C: Code**