# 

# **Serial Communications**

***C8051 Microcontroller***

Jeffrey Pistacchio

Christopher Pybus

Microprocessor Systems

17 October 2016

# Introduction

The purpose of this lab was to use and demonstrate the purpose of synchronous and asynchronous communication. Universal Asynchronous Receive Transmit (UART) was used to demonstrate the asynchronous portion of the lab. Synchronous Serial Port Interface (SPI) was used to demonstrate the synchronous portion of the lab. The exercise was divided into three parts. Part one involved wiring a DB-9 connector on the protoboard so that the microprocessor was connected to the computer screen on two communication ports (the on chip connection and the protoboard connection). Each port had its own terminal on the computer screen and its own UART on which to communicate. Any key pressed on one screen would print on both screens. The second part of the lab required the same goal but needed to use the UART interrupts instead of manually tripping flags. Furthermore, the software must be used to communicate two microprocessors between the two screens and each other. Finally part three involved using SPI to have the C8051 send characters to itself and print the sent and received characters on the computer screen. Then this software needed to be configured to work with an HC11 EVB.

# Methods and Procedures

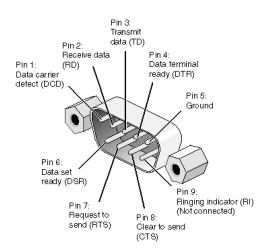
Each of the three parts for this lab were divided into more specific goals so the team could focus on specific tasks instead of the entire lab. Part one was divided into three goals: configure all registers properly for use with a second UART, wire a second DB9 connector to the protoboard, and write software that used the UARTs to communicate with each other. Part two had two goals as well: edit the software to use interrupts instead and use the interrupt software to communicate between two microprocessors. Part three had three goals: configure SPI communication so that the microprocessor could talk to itself using MOSI and MISO, format communication from SPI on the ANSI terminal, and attach a slave device to the SPI with commands downloaded using the ANSI terminal.

## Part 1 - Goal 1

Before the team could write software to use the second UART on the 8051, they first had to configure all the various software items they planned to use. They configured the timers first, as those set the rate at which the UARTs would communicate. Timer1 and Timer2 were used to set the baud rates for UART1 and UART0, respectively. The team chose to use 115200 baud rate for both UARTs instead of 115200 for one and 9600 for the other. They felt that using the same rate for both would simplify configuration and reduce possible error. The crossbar was configured next. As the only communication along the crossbar was UART1, that was all that was needed. They set the proper pins so that UART1’s transmit and receive buffers would communicate along Ports 0.2 and 0.3, which are pins 19 and 20 on the crossbar, respectively. The UARTs were configured next so that they would receive the signals output by timer1 and timer2 to set the baud rates. Both UARTs were set to 8 bits with a variable baud rate. They both ignored the logic level of the stop bit, and they both were enabled.

## Part 1 - Goal 2

After the software was configured, the team built the hardware that linked to the transmit and receive ports for the second UART on the crossbar. As said before, it simply came down to connecting a serial cable to the transmit and receive ports that were enabled through the software, as well as connecting the serial cable to a common ground. As transmissions were sent through UART1, they would output to the crossbar and relay along the serial cable to the ANSI terminal on the computer. The crossbar ports had to be connected to the transmit and receive ports on the serial cable, which are ports 0.3 and 0.2 respectively. Figure 1 shows the layout of the DB-9 serial cable below.



*Figure 1 - pinout of the DB-9 connector.*

## Part 1 - Goal 3

Goal 3 moved back to the software side. After configuring the registers, timers, crossbar and UARTs, and building the hardware, the team now had to write software that could send and receive messages on both UARTs. For example, if a character was typed on the ANSI terminal linked to UART0, it would show up on UART1’s terminal as well. This was one by constantly monitoring the transmit and receive flags for both UARTs. As soon as one of them was set to high, the software would read in the character from the buffer, and send it back to the other UART.

## Part 2 - Goal 1

In Goal 3 of the previous part, the UARTs communications were managed by simply constantly checking the flags of the UARTs. This works fine, but is rather inefficient. In this goal, the software was reconfigured to work using interrupts instead. Interrupts 04 and 20 are run whenever one of the transmit or receive flags for either UART are changed. This way, the code that prints characters to the screen is only run when the interrupt is triggered, and the program does not have to constantly check if the flags have been triggered. Even though the method in the code was changed, the overall function of the code remained the same. Whenever a character was typed on one UART, it was sent to the other UART via the code and displayed on the ANSI terminal.

## Part 2 - Goal 2

In this goal, the team had to bring in a separate 8051 microprocessor that had to be running working code from goal 1. Previously, both UART1’s were connected to serial cables that led into computers so the output could be seen on an ANSI terminal, but the team disconnected those cables, and instead connected the UART1’s together. The transmit of the UART1’s were connected to the receive of the other. This way, any message that was sent over UART1 in one microprocessor would appear on the UART1 in the other microprocessor. The two serial cables were connected to both UART0’s on each respective 8051 board. Now, if a character was sent typed into a UART0 of board A, it would be sent to board A’s UART1, which was connected to board B’s UART1. It would be received by board B’s UART1 and sent to board B’s UART0, appearing on the screen again because UART0 was connected to ANSI terminal.

One of the biggest problems the team had when they first started this part was an infinite loop that would repeat a character until the program was stopped. This was due to the fact that when a character was received by UART1 (from either UART0 of the same board, or UART1 of the other board), it would repeat it back to the source. After repeating it back, the other UART1 would again repeat it. This caused an infinite loop. The team solved this problem by distinguishing between where characters were coming from when they were received by UART1. If a character was received from the other UART1, it would simply forward it along to UART0, and if a character was received from UART0, it would forward it along to the other UART1. This selective forwarding stopped the infinite loop from occurring and made this goal successful.

## Part 3 - Goal 1

For this goal, the team had to first configure all the proper registers, timers, UARTs, and crossbar ports so the code would work properly. For the most part, the team decided to stay with the same configuration they had been using for the previous parts of the lab, but they also added the SPI configuration. This involved enabling SPI as a master device in 4 wire mode. The crossbar was also configured to accept MOSI and MISO input and output. A wire was placed between the MOSI and MISO ports on the crossbar so whenever a message was sent on MOSI, it would be immediately received by MISO and appear back in the SPI buffer. The team tested this by sending various characters. The team also set the connection to ground and attempted communication on the SPI, and as expected, the result was all zeros. When the team set the connection to high by connecting to +5V, the result was FF, signifying all bits were high.

## Part 3 - Goal 2

After getting characters to send and receive on the SPI bus, the team used the ANSI terminal and escape formatter sequences to format the ANSI terminal so that characters typed on the keyboard would appear on the top of the screen, and characters received by the SPI connection would appear on the bottom of the screen. This was done by formatting (splitting) the screen beforehand with escape sequences, and then simply printing to the proper slot upon receiving a character.

## Part 3 - Goal 3

Once the microprocessor demonstrated its ability to send and receive over SPI with itself, It was tested with an external device. In this case it was tested with the EVB HC11. Code was downloaded to the EVB through a specific set of commands on the Procomm terminal. These commands took a file given to the team and downloaded it to the HC11. In order for these programmed commands to be of value, the HC11 would need to be in communication with the C8051. This was done by wiring the 60 pin bus on the HC11 to the same on the C8051. Specifically, connecting the MISO, MOSI, NSS, and SCK lines to each other.

# Results and Analysis

Part 1 was tested by simply typing characters on one terminal and confirming that they appear on the other terminal. The team typed 50 characters on each terminal and confirmed they appeared on the other terminal without an issue.

Part 2 was tested very similarly, by typing on one terminal and confirming that it appeared on the other terminal. Again, 50 characters were typed on each terminal and each time the character appeared on the other terminal.

Part 3 was tested in many ways. First, as with the previous two parts, characters were typed into the terminal and it was confirmed that they appeared back from the SPI interface. The next test the team did was grounding the SPI connection, and verifying that any received characters were all zeros because all the bits were being held low by the physical ground. The same was done after but with +5V, and all the bits received were high which correlated with FF hex value.

# Conclusion

The team successfully accomplished the first two parts of Lab 3 by making two programs that accomplished all the subgoals. Dividing the project into separate goals made the exercise significantly more approachable due to implementing the divide and conquer mindset. Each goal was simple enough that each group member could complete a goal by the time they came to class, and spend lab time debugging code and building hardware.

Given more time the team would have liked to finish part 3 of the lab. They would also like to add enhancements, such as adding on two devices onto SPI bus and communicate with both. In addition to the issue of time, the team also recognizes that going to find help earlier would have remedied many issues. Furthermore, paying attention to details in the code and better commenting practices would have led to catching mistakes earlier and being better able to finish the lab on time.

# Appendix A - Schematic for Part 1

# Appendix B - Schematic for Part 2

