\

Spring 2009

Department of Computer Engineering  
San Jose State University

CMPE127 Microprocessor Design

Instructor: Dr. Harry Li

**Lab 1**Establishing RS485 Communications

Submitted By: Raja Kantamaneni

March 4, 2009

rajak87@pacbell.net

**CmpE 127 Lab1   
Establishing RS485 Communications**

Raja Kantamaneni, Student

*Computer Engineering Department, College of Engineering*

*San Jose State University, San Jose, CA 94303*

*E-mail: rajak87@pacbell.net*

**Abstract**

*The goal of this lab was to setup an RS485 interface for the ARM7TDMI CPU. This involves using one of the three serial communications ports of the CPU and interfacing it to a MAX485 to achieve logic level shifting.*

**1. Introduction**

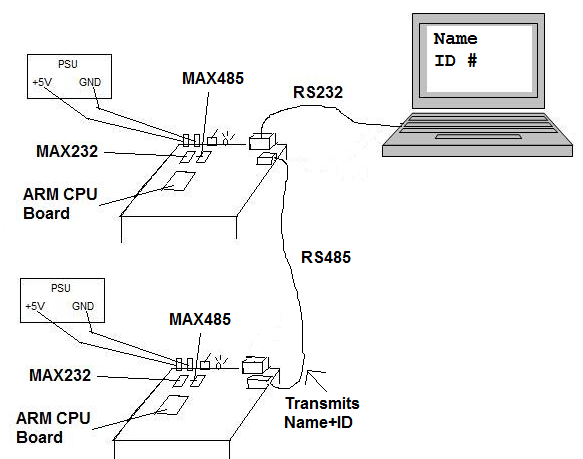
In this lab, we build an RS485 interface. The RS485 (EIA485) standard uses two lines to transmit data. The difference in potential between the two lines defines a logic signal. This would be difficult to do with just the ARM, thus in order to build a working RS485 interface, a MAX485 will be used to convert between the ARM CPU (3.3 VDC) and RS485 logic levels.

RS485 features many benefits over RS232 communication. It can support up to 32 devices on a multi-point network, supports longer communication distances, has better data integrity, and more.

One of the serial communication ports from the ARM7 CPU Board will be interfaced with a MAX485. The MAX485 provides one driver and receiver that share data lines. Since the MAX485 is half duplex, the driver enable (DE) and receiver enable (\RE) must be set properly to ensure we can either receive or transmit data. In order to test the RS485 communication, we will communicate between two ARM boards. One ARM board will transmit data (name and student ID) to another ARM board which will receive the data and output it over RS232 to a terminal on a laptop. A program such as HyperTerminal or Tera Term will be required on the PC side in order to view the transmitted data from the ARM.

**2. Methodology**

This section describes the design and implementation of the system. The different sections are RS485 interface setup, RS485 communication program (basic), RS485 transmit program (advanced). A system diagram of the final setup is shown in Figure 2.1.

  
Figure 2.1  
Overall System Diagram

**2.1. RS485 Interface Setup**

In order for the ARM7 CPU Board to communicate via RS485, we must first setup the interface. A 4-pin RJ-11 jack, one MAX-485 DIP IC, and a 100ohm resistor are needed.

  
Figure 2.2  
Required Parts for RS485 Interface

Several styles of RJ11 jacks or sockets are available; the one pictured above is merely an example of what can be used. It is recommended that a DIP IC wire wrapping socket is used on the prototyping board to make it easier to insert and wire the MAX485 device. Such sockets should also be used for all future components in order to facilitate the easy removal or replacement of parts.

**2.2. RS-485 Communication Program (Basic)**

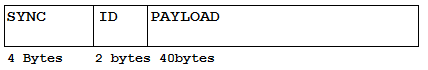
Communication via RS-485 can be very simple. We write a very basic program to send/receive data simply to test our newly built interface. This program will transmit “Raja Kantamaneni 004614949” (Name/SID) through the ARM’s serial communication port that is connected to the MAX485.

This program must be developed in the mbed compiler at http://mbed.co.uk . In order to design this program, it’s necessary to learn about the Serial API in the mbed library. This API was previously discussed in the HW2 Report; please refer to that report or the Mbed handbook if needed.

**2.3. RS-485 Communication Program**

**(Advanced)**

After completing the simple communication test described in section 2.2, we can implement a more advanced protocol for communicating. This advanced method will feature packets of data.

  
Figure 2.3

The entire packet is 46 bytes. The first 4 bytes are used for synchronization bytes (four 0xFFh bytes). The next two bytes are used for an ID field. If there are multiple devices/nodes attached to the RS485 interface, the ID bytes allow us to specify a particular target device for our packet. The remaining 40 bytes are the payload.

We’ll be using a master/slave setup. The master will initiate communication by sending a packet as described above. The receiving end will be waiting for 4 Sync Bytes, its ID, and then the payload. For our purposes, we’ve reserved the first 3 bytes of the payload for an instruction. So far, we’ve implemented and OUT and SWP instruction. An OUT instruction tells the target device to printout the remainder of the payload to a computer/laptop over RS232. A SWP instruction tells the target device to ignore the rest of the payload, switch modes (from receiving to output), send ONE packet containing a name and student ID, and then revert back to receiving mode. If the master sends a SWP instruction, it will also switch modes (from sending to receiving) and wait for the expected packet from the slave.

**3. Implementation**

Implementation of our design started with the gathering and purchasing of materials. Once all the necessary materials were obtained, they were laid out onto the board and wire wrapped or soldered. Next, the software to transmit a message over RS485 was created a loaded onto the ARM CPU board.

**3.1. Hardware Implementation**

ARM7TDMI

RJ11

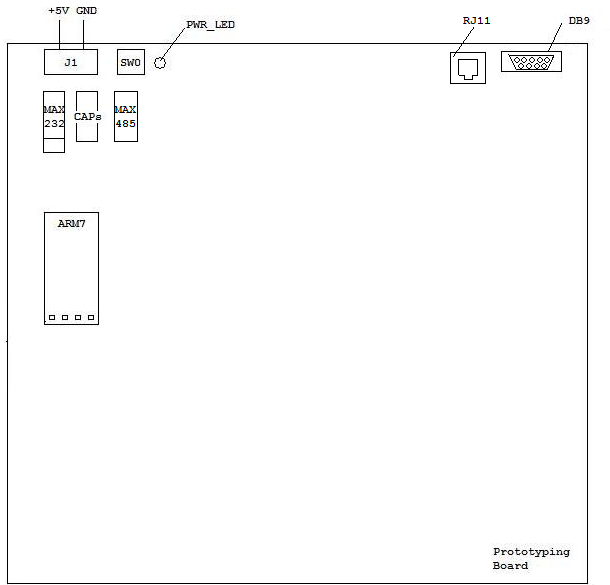
MAX485

DB9

MAX232

Figure 3.1 – Basic System Block Diagram

Our system has five major components – the ARM7TDMI CPU board, the MAX232, a DB9 Connector, a MAX 485, and an RJ11 jack. The MAX485 and RJ11 Connector are new additions to the prototyping board. Figure 3 shows the placement of each part. A larger, clearer version of the figure is shown in Appendix B.

Figure 3.2 - Board Schematics

The MAX485 is used as a level shifter to convert logic signals on the Rx and Tx lines between the ARM CPU board and the RS485 RJ11 connector. It’s important to note that RS485 using two lines for data; the potential difference between the two lines represents the logic signal. The MAX485 takes care of converting our ARM logic signals to RS485 signals.

The pin assignment between the components is shown in Figure 3.3 below.

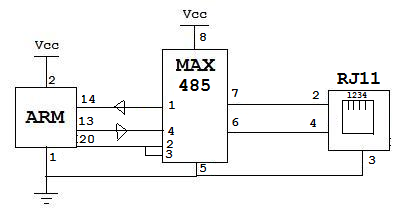


Figure 3.3

Pin Assignment

The MAX485 requires two ~100ohm resistors as part of its external circuitry. See the MAX485 datasheet or Appendix B for more details. Figure 3.4 shows the final MAX232 and DB9 Connector setup.

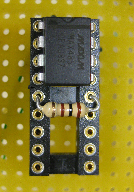
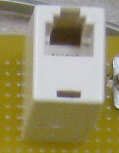
 

Figure 3.4  
MAX 485 and RJ-11 Connector

**3.2. Software Implementation**

The basic RS-485 communication program was fairly easy to implement.

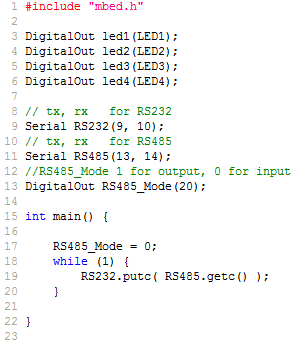
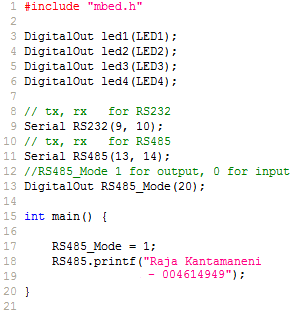


Figure 3.5 – Basic Communication Program  
Receiving End

  
Figure 3.6 – Basic Communication Program  
Sending End

These programs should be easy to understand. The sending side just sends a string of bytes across the RS485 port. The receiving end simply gets bytes on the RS485 connection and outputs them over RS232.

In order to do the advanced communication described in section 2.3, we require a bit more code. The code is long, so it has been attached in Appendix E.

This code is fairly easy to understand as well. RS485\_Mode controls whether our Max485 is in send or receive mode. To send our first packet, we call the send\_sync function. The send\_sync() function sends four SYNC bytes (0xFFh). We then send two bytes containing the ID of our target device (for now we don’t do anything with this since there are only two devices in our RS485 network). We then send our 40-byte payload, with the instruction OUT in the first 3 bytes.

The receiving ARM will see the SYNC bytes, the ID, and the OUT instruction, and will then output the remainder of the payload over RS232 to a PC.

We then send another packet with the SWP instruction, which instructs the receiving end to switch modes and send some data back to us. At this point, the receiver switches to sending mode, and we switch to receiving mode. The function receive\_sync() looks for the 4 sync bytes that start our receiving packet. We receive a single packet containing the other ARM user’s name and student ID and print it out to our RS232 port.

**4. Testing and Verification**

All circuits and components were tested as they were assembled to verify they worked properly. After the two ARM boards were connected over RS485, we used the simple communication program to quickly test that our wiring was correct. We then proceeded to establish advanced communications by developing the packet-based software. PC terminals on both sides received the correct information.

**5. Conclusion**

For this lab, an RS-485 interface was built for the ARM CPU board. We took our first step and tested it by communicating between two ARMs over an RJ-11 cable. The receiving arm outputted the message it received over RS485 to a PC local connected to it over RS 232. The PC properly received messages transmitted by the ARM CPUs, thus completing the main goals of this lab.

It was very interesting to see the MAX485 in action. We checked the voltage potentials on the two data lines to see how it worked, and realized the benefits of such an approach (data integrity, longer communication distances, etc).

With the RS485 interface, we could eventually create a network of ARM CPUs that communicate with each other. This can be useful for sensor networks, etc.

We faced a few problems in this lab. One problem was related to the online mbed compiler. The mbed compiler interface was recently updated and it seems to have a few bugs. We often found that if we deleted characters, and later highlighted over that space, the characters would suddenly re-appear. Sometimes, when alt-tabbing between windows, deleted characters would return as well. This problem was experienced by multiple students so it’s most likely a problem on the mbed website.

Another interesting problem we faced was that our RJ11 connects seemed to be flipped even though the pins were properly connected. Upon further inspection, we found that the RJ11 coupler that was being used as a connector was actually reversing the signals. Thus, we had to change some wiring to compensate.

**6. Acknowledgement**

Roger Duong, CmpE 127 Lab Partner

**7. References**

[1] H. Li, “Lab 1 Establishing RS485 Communication”, Computer Engineering Department, College of Engineering, San Jose State University, February 2009.

[2] mbed Handbook. 18 February, 2009

< http://mbed.co.uk/handbook/>

[3] RS-485 Specifications, 28 February, 2009

<http://www.lammertbies.nl/comm/info/RS-485.html>

**8. Appendix**

Appendices are attached at the end of this report.