ECE 480 Senior Design Team 1

Design and Implementation of a Universal Asynchronous Receiver Transmitter Using the RS485 Communications Bus

Application Note

Brandon M. Boozer

November 17, 2010

Table of Contents

1. Introduction

The Universal Asynchronous Receiver Transmitter is a highly flexible widely used digital communications protocol. This serial data protocol is implemented in systems throughout the world in devices as small as a penny to huge infrastructures like the power grid. The protocol is useful primarily due to its simplicity, flexibility, and robustness. Implementing this communications protocol over the RS 485 data line bus standard makes it even more useful by providing a significant improvement in viable transmission line length.

2. Objectives

This application note is intended to provide the reader with sufficient information to design, build, and develop code for a UART over RS 485 serial communications bus. The approach taken is indicative of one particular method and is not intended to be an exhaustive description of the material or possible configurations. Of particular interest will be the overview of RS 232 to RS 485 conversion using the Maxim Max 485 IC. Furthermore all coding examples used in this document have been developed using a derivative of the C programming language called “Wiring”. Wiring is sufficiently close to the C language that examples in this document should be easily convertible to native C code.

3. Background

a. The Universal Asynchronous Receiver Transmitter

UART’s are very simply hardware which has the ability translate digital communications between parallel and serial forms. These communications protocols are commonly used in conjunction with standard serial data cable communications such as the RS 232 and RS 485 standards. The universal designation refers to the ability to configure the Receiver / Transmitter to operate using different data formats and transmission speeds. Due to the Asynchronous nature of this communications bus it is imperative that a Baud rate be set to control the speed at which transmission occurs. Communication may be "full duplex" (both send and receive at the same time) or "half duplex" (devices take turns transmitting and receiving).

b. Synchronous Versus Asynchronous Transmission

Synchronous and asynchronous data transmissions are highly similar except for one key differentiator. In synchronous transmission the clock is independently transmitted from the data packet. This allows both the sending and receiving side of the bus to understand what is happening on the line and when to send or expect data. Conversely in asynchronous transmission there is no defined "clock" but rather a preset Baud Rate at which data is transmitted along with start and end bits. The Baud rate is independently set up on both the receiver and transmitter such that the two are identical or else data corruption will occur. The Baud Rate is a programmed per packet clock which denotes the frequency of data transmission within a packet but is not indicative of the frequency of packet transmission on a line. The start and stop bits which are embedded at the beginning and end of each packet serve as a wake up and sleep flag for the receiver. When the receiving end senses the start bit of a packet it turns on and accepts the data. When the stop bit of the packet is read the receiver turns off and stops taking in data. This means that the transmitter does not need to wait for a clock pulse to send its signal but can rather send information out as soon as it is ready. The obvious benefit of Asynchronous transmission is that it requires no clock, however this flexibility comes at a cost. Since there is no predefined transmit period to be obeyed any transmitter on the line can send out data at any time. This can manifest itself in overlap on a bus with multiple transmitters and results in data corruption. For this reason all UART's are restricted to only 2 devices per line unless a device hierarchy is present.

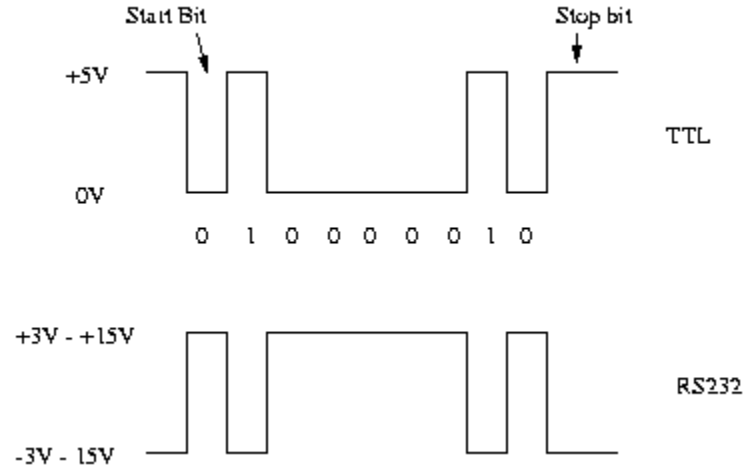

Fig. 1 – UART Data Packet Waveform with Start and Stop Bits

c. The RS 485 Standard

Much like the RS 232 standard RS 485 supports serial communication over a twisted pair configuration. The fundamental difference between the two standards stems from its use of a balanced line for transmission. Also known as differential this format sends the same signal on two separate lines with phase delay and then compares the signals at the end, subtracts any noise, and adds them to regain signal strength. This process allows the RS 485 standard to be viable over significantly longer distances than its short range RS 232 counterpart. The table below provides a more detailed specification breakdown of the benefits of RS 485.

RS-232	RS-422	RS-485
Single Ended	Differential	Differential
Unidirectional	Unidirectional	Bi-directional
Point-to-Point	Multiple Receiver	Multiple Transmitter and Receiver
20 Kbps max	10 Mbps max @ 40 ft	10 Mbps max @ 40 ft
50 ft	4000 ft max length @ 100 Kbps	4000 ft max length @ 100 Kbps

Fig. 2 – RS 232, RS 422, & RS 485 Comparison Table

4. Wiring the Maxim MAX 485 IC (PDIP)

The first step in creating a wired serial connection between two devices is to wire the transmitter and receiver circuits. The pin-out below should provide a visual guide on how to wire the Max 485 IC such that it is both transmit and receive enabled.

1. First the IC must be powered. Pins 5 and 8 represent Ground (0v) and Vcc (5v) respectively and should be wired to these corresponding signals.

2. In order to enable the Read and Write ports of the IC Pin 2, the Recieve Enable (RE bar) and Pin 3, the Driver Enable (DE) must be wired to Ground (0v) and Vcc (5v) respectively.

3. Now the signal lines must be added. These are the wires which will carry data to and from the IC. They are separate and correspond to distinct Input and Output ports. Pin 1 Reciever Out (RO) will be wired to your device's read port and Pin 4 Driver In (DI) will be wired to your devices write port.

4. Now that the IC has been properly powered and connected to a desired device the output section must be wired. To provide an acceptable line termination load Rt on at the output of the IC a 100 ohm resistor should be added between Pins 6 and 7.

5. Once the resistor is added the transmission lines can be connected. Any twisted pair wiring connection is sufficient to carry the signal however the better shielded wiring used the less susceptible the connection is to noise and data loss. A good choice is the use of certified RS 485 cabling such as CAT 3 or higher Ethernet cable.

6. The circuit is now completely wired and can be used.

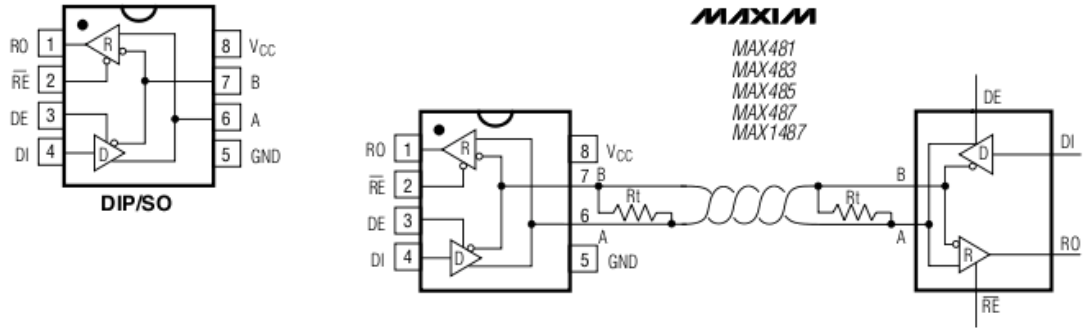

Fig. 3 – Maxim Max485 IC PDIP Pin-out and Wiring Diagram

The MAX485 has 8 pins:

RO – Receiver out

RE – Receiver enable (enabled when this pin is LOW)

DE – Driver enable (enabled when this pin is HIGH)

DI – Driver in (the transmitter pin)

GND – Ground (0V)

A – Connect to pin A of the other 485 IC

B – Connect to pin B of the other 485 IC

Vcc – Power, (+5v)

7. Developing "Wiring" Code for the UART Bus

While it is entirely possible to develop code to configure and run a UART bus from the ground up, it should be noted that numerous iterations of this same application (and their corresponding open source code) exist in the public domain. As such it is often wise to at least start your development from an example. This is beneficial in two ways. For one, it gives you a head start and offers a general framework which you can modify and add to. Secondly it allows you to have a working setup from the beginning which you can use to test your device and gain intuition about potential problems.

* Since all microcontrollers vary slightly in native programming environments, it should be noted tha the code provided below was developed using the Arduino IDE. This IDE is natively compatible with all Ardruino Microcontrollers and development boards. It includes its own compiler and encapsulates the "Wiring" language header files which are modify standard C code for use on Arduino MCU's.

The code below is a simple example which will put the UART in Transmit Tx mode and intermittently send out a command to turn on an LED provided an LED is connected to the correct port of the MCU on the receiving end.

* Note that the code below first sets the Baud rate of the UART to 57600. This is only one of the many Baud rate choices available but must always be matched by the receiving end.

```
#define LED_PIN 13;
int i;
boolean b;

void setup() {
Serial.begin(57600);

i = 0;
b = false;
pinMode(LED_PIN, OUTPUT);

Serial.println("Init() complete");
}

void loop() {
i ++;

Serial.println(i);

b = !b;
digitalWrite(LED_PIN, b);

delay(1000);
}
```

Fig. 4 – Arduino Wiring code block example for intermittent Tx over RS 485 UART

The code then executes through a simple counter loop to periodically turn on and off the LED roughly every 1 second(s).

The Receiver in this case must be read enabled the entire time if the signal is to be receiver. More dynamic configurations can be achieved through the use of packet detection implemented in software. Such configurations do not require single direction communication or a designated master slave relationship between the Transmitter and Receiver.

In case the above code block's function is not readily apparent the pseudo code flow diagram below is available for reference. The diagram describes the function of a single loop of the above code.

Step	Master	Slave
1	Prepare command 0xFF 0x05 0x01 0xFF	Waiting
2	Send command	Waiting
3	Change to RX mode Set pin 2 to LOW	Receive serial data 0xFF 0x05 0x01 0xFF
4	Waiting	Parse serial data Start byte ok, ID byte OK, command = send status, end byte ok
5	Waiting	Change to TX mode Set pin 2 to HIGH
6	Waiting	Send response (0x05 = ID, 0x01 = status ok) 0xFF 0x05 0x01 0xFF
7	Receive serial data	Change back to RX mode Set pin 2 to LOW
9	Change back to TX mode Set pin 2 to HIGH	Waiting
10	Parse data....	Waiting

Fig. 5 - Pseudo Code flow diagram for Fig. 4

8. Conclusion

Using the provided data it should be a simple process of wiring and a limited amount of code to obtain a working UART over RS485 communications bus. The depth of code development is intentionally high level due to the broad nature of applications for this configuration. It is also worth mentioning that many more data types other than standard UART serial communication can be transmitted over the same hardware interface detailed in this document. MODBUS, SPI, CAN, and many other can all be implemented using the RS 485 standard. Hopefully the user can appreciate the simplicity, flexibility, and universality of this configuration and will find many uses for it in current and future applications.