Implementation of Low Power UART Module and Simulation for Serial Data Transmission with Arduino and PIC

May Zin Tun, Hla Myo Tun
1Dept of Electronic Engineering, Technological University, Mandalay, mayzintun954@gmail.com
2Dept of Electronic Engineering, Yangon Technological University

Abstract--In this project, Serial communication is the main method used for communication between external microcontrollers and computers, and can be a challenge to set up correctly. This application note will provide a walk-through example on how to set up a simple UART communication link using Proteus and the Arduino Atmel ATmega328 microcontroller. Peripheral Interface Controllers (PICs) are inexpensive microcontroller units with built-in serial communication functionality. Similarly, Proteus, widely used technical computing software, allows serial communication with external devices. This project exploits the serial communication capability of PIC microcontrollers and the Proteus software. Four examples are included to illustrate that the integration of low-cost PIC microcontrollers with the Arduino Atmel ATmega328 allows data transmission, and data receiving. Universal Asynchronous Receiver Transmitter (UART) implements serial communication between peripherals and remote embedded systems. The UART protocol is defined based on fixed frequencies with a sampling method to achieve robustness under reasonable frequency variations between systems. Such design specifications are natural for clocked domains. This work investigates whether this simple clocked hardware protocol can be advantageously implemented using asynchronous design techniques. A full duplex clocked and asynchronous UART are implemented and compared. The asynchronous design results in average power of about one fourth that of the clocked design under standard operating modes. A Universal Asynchronous Receiver/Transmitter, abbreviated UART, is a type of “asynchronous receiver/transmitter”, a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA RS-232, RS-422 or RS-485. A UART is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port. Modern ICs now come with a UART that can also communicate synchronously; these devices are called USARTs. The UART implemented with C language can be integrated into the Arduino Atmega328 and PIC16F887 to achieve compact, stable and reliable data transmission. The simulation results with Proteus are completely consistent with the UART protocol.

Keywords: Arduino; PIC; Proteus Software; UART.

I. INTRODUCTION

This project will provide instructions on how to obtain simple communications between a computer and a microcontroller using UART (Universal Asynchronous Receiver/Transmitter). UART is a commonly used piece of hardware which translates data between parallel and serial communication mediums. In this project, it will be taught how to set up a simple interface between a computer and a microcontroller by utilizing Proteus 8, and the Arduino Atmel ATmega328 microcontroller. The implementation described in this project will provide a walk-through to set up the Arduino Atmel ATmega328 to receive communication from the computer and then respond...
appropriately. In this case, the microcontroller will receive characters from the computer consisting of either ‘a’ or ‘b’ and ‘c’ or ‘d’. These characters will activate an LED by turning it on or off, by interpreting a ‘b’ as an “on” signal, and an ‘a’ as an “off” signal. The Universal Asynchronous Receiver Transmitter (UART) is the most widely used serial data communication circuit ever. UARTs allow full duplex communication over serial communication links as RS232. UARTs are available as inexpensive standard products from many semiconductor suppliers, making it unlikely that this specific design is useful by itself. The basic functions of a UART are a microprocessor interface, double buffering of transmitter data, frame generation, parity generation, parallel to serial conversion, double buffering of receiver data, parity checking, serial to parallel conversion. The data is transmitted asynchronously one bit at a time and there is no clock line. The frame format of used by UARTs is a low start bit, 5-8 data bits, optional parity bit, and 1 or 2 stop bits. Universal Asynchronous Receive/Transmit consist of baud rate generator, transmitter and receiver. The number of bits transmitted per second is called baud rate and the baud rate generator generates the transmitter and receiver clocks separately. Transmitter interfaces to the data bus with the transmitter data register empty (TDRE) and write signals. When transmitting, UART takes eight bits of parallel data and converts it into serial bit stream and transmit them serially. Receiver interfaces to the data bus with the receiver ready and the read signals. When UART detects the start bit, it receives the data serially and converts it into parallel form and when stop bit (logic high) is detected, data is recognized as a valid data. Asynchronous serial communication has advantages of less transmission line, high reliability, and long transmission distance, therefore is widely used in data exchange between computer and peripherals. Asynchronous serial communication is usually implemented by Universal Asynchronous Receiver Transmitter (UART). UART allows full duplex communication in serial link, thus has been widely used in the data communications and control system. In actual applications, usually only a few key features of UART are needed. Specific interface chip will cause waste of resources and increased cost. Particularly in the field of electronic design, SOC technology is recently becoming increasingly mature. This situation results in the requirement of realizing the whole system function in a single or a very few chips. Designers must integrate the similar function module into FPGA.

Peripheral Interface Controllers (PICs), developed and marketed by Microchip Technology, are inexpensive microcontroller units that include a central processing unit and peripherals such as memory, timers, and input/output (I/O) functions on an integrated circuit (IC). There are more than 100 varieties of PIC microcontrollers available, each providing functionality for different types of applications, making PICs one of the most popular microcontrollers for educational, hobby, and industrial applications. Similar to other microcontrollers, PICs are usually not designed to interface with human beings; instead they are directly embedded into automated products/processes. Thus, Proteus 8 software capabilities, which have become a mainstay of many personal computer (PC) applications, are nonexistent for PICs and many different Arduinos.

![Figure 1. Block Diagram of Serial Data Transmission](image-url)
II. IMPLEMENTATION OF UART

The UART serial communication module is divided into three sub-modules: the baud rate generator, receiver module and transmitter module, shown in Figure 2. Therefore, the implementation of the UART communication module is actually the realization of the three sub-modules. The baud rate generator is used to produce a local clock signal which is much higher than the baud rate to control the UART receive and transmit; The UART receiver module is used to receive the serial signals at RXD, and convert them into parallel data; The UART transmit module converts the bytes into serial bits according to the basic frame format and transmits those bits through TXD. The UART should perform:

1. Start bit detection/generation.
2. Stop bit detection/generation.
3. Data bits sampling/transmitting.

Baud Rate Generator is actually a frequency divider. The baud rate frequency factor can be calculated according to a given system clock frequency (oscillator clock) and the requested baud rate. The calculated baud rate frequency factor is used as the divider factor. In this design, the frequency clock produced by the baud rate generator is not the baud rate clock, but 16 times the baud rate clock. The purpose is to precisely sample the asynchronous serial data at the receiver.

Assume that the system clock is 32MHz, baud rate is 9600bps, and then the output clock frequency of baud rate generator should be 16 * 9600Hz. Therefore the frequency coefficient (M) of the baud rate generator is:

\[ M = \frac{32\text{MHz}}{16 \times 9600\text{Hz}} = 208 \]

When the UART receives serial data, it is very critical to determine where to sample the data information.

The ideal time for sampling is at the middle point of each serial data bit. In this design, the receive clock frequency is designed to be 16 times the baud rate, therefore, each data width received by UART is 16 times the receive clock cycle. The baud rate at which data will be transmitted will be one of four possible bit rates. The following table shows the clock frequencies should generate:

<table>
<thead>
<tr>
<th>Rate Select</th>
<th>Bit Rate</th>
<th>Clock Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(00)</td>
<td>1200 bits/s.</td>
<td>1200Hz</td>
</tr>
<tr>
<td>1(01)</td>
<td>2400 bits/s.</td>
<td>2400Hz</td>
</tr>
<tr>
<td>2(10)</td>
<td>4800 bits/s.</td>
<td>4800Hz</td>
</tr>
<tr>
<td>3(11)</td>
<td>9600 bits/s.</td>
<td>9600Hz</td>
</tr>
</tbody>
</table>

Basically, this circuit does the same exact thing as the 1 kHz clock generator from the previous homework assignment, with the added capability of a selectable frequency.

B. Receiver Module

During the UART reception, the serial data and the receiving clock are asynchronous, so it is very important to correctly determine the start bit of a frame data. The receiver module receives data from RXD pin. RXD jumps into logic 0 from logic 1 can be regarded as the beginning of a data frame. When the UART receiver module is reset, it has been waiting the RXD level to jump. The start bit is identified by detecting RXD level changes from high to low. In order to avoid the misjudgment of the start bit caused by noise, a start bit error detect function is added in this design, which requires the received low level in RXD at least over 50% of the baud rate to be able to determine the start bit arrives.
Since the receive clock frequency is 16 times the baud rate in the design, the RXD low level lasts at least 8 receiving clock cycles is considered start bit arrives. Once the start bit been identified, from the next bit, begin to count the rising edge of the baud clock, and sample RXD when counting. Each sampled value of the logic level is deposited in the register rbuf [7, 0] by order. When the count equals 8, all the data bits are surely received, also the 8 serial bits are converted into a byte parallel data. The serial receiver module includes receiving, serial and parallel transform, and receive caching, etc. In this project, it has been used finite state machine to design, shown in Figure 3. The state machine includes five states: R_START (waiting for the start bit), R_CENTER (find midpoint), R_WAIT (waiting for the sampling), R_SAMPLE (sampling), and R_STOP (receiving stop bit).

**C. Transmit Module**

The function of transmit module is to convert the sending 8-bit parallel data into serial data, adds start bit at the head of the data as well as the parity and stop bits at the end of the data. When the UART transmit module is reset by the reset signal, the transmit module immediately enters the ready state to send. In this state, the 8-bit parallel data is read into the register txdbuf [7: 0]. The transmitter only needs to output 1 bit every 16 bclkt (the transmitting clock frequency generated by the baud rate generator) cycles. The order follows 1 start bit, 8 data bits, 1 parity bit and 1 stop bit.

**III. DATA COMMUNICATION**

Data communication is one of the most challenging fields today as far as technology development is concerned. Data, essentially meaning information coded in digital form, that is, 0s and 1s, is needed to be sent from one point to the other either directly or through a network. And when many such systems need to share the same information or different information through the same medium, there arises a need for proper organization of the whole network of the systems, so that the whole system works in a cohesive fashion. Therefore, in order for a proper interaction between the data transmitter and the data receiver there has to be some set of rules or (“protocols”) which all the interested parties must obey. The requirement above finally paves the way for some data communication standards.

Depending on the requirement of applications, one has to choose the type of communication strategy. There are basically two major classifications, namely serial and
parallel, each with its variants. The discussion about serial communication will be undertaken in this lesson.

![Figure 5. Data Transmission]

In serial communication, it is transmitting the data/information bit after bit (only one bit goes through in a particular moment). In parallel communication, it is transmitting a number of bits at once from one computer to the second computer.

**A. Serial Data Transmission**

Serial transmission technology is increasingly used for the transmission of digital data. A large number of up-to-date communications networks apply serial transmission. The numerous applications include computer networks for office communications, field bus systems in process, building and manufacturing automation, Internet and, finally, ISDN. Serial data transmission implies that one bit is sent after another (bit-serial) on a single transmission line. Since the microprocessors in the devices process data in bit-parallel mode, the transmitter performs parallel-to-serial conversion, while the receiver performs serial-to-parallel conversion (Figure.6).

This is done by special transmitter and receiver modules which are commercially available for different types of networks. Extremely high data rates are possible today so that the increased time consumption required by this technology is accepted in most cases. The reductions in costs and installation effort as well as user-friendliness, on the other hand, are points not only for locally extended systems in favor of serial data transmission.

Serial communication is one of the simplest ways of communication between a microcontroller and PC or vice versa. It requires only single wire for transmission of a data and another for receiving.

**B. Serial Data Transmission for Arduino**

Serial Communication is a standard for communicating electronically which the process of sending data one bit at a time, sequentially, over a communication or computer bus. Arduino can send out bytes of serial data through a USB connection to the computer. Arduino can listen for bytes or serial data coming from the computer. Serial communication is a common method of communication between microcontrollers and PCs. For two devices to communicate serially are digital pulses are sent back and forth between devices. They must agree on a rate of communication and sync to that rate. Each device has a different clock. Duh. ‘Sender’ sends pulses that represent data, at the agreed on rate and ‘receiver’ listens for pulses at the same rate.

![Figure 7. TX and RX]

Three connections needed for Serial Comm. A common ground so both devices have a common reference point to measure voltage by a wire to send data and a wire to receive data. Compiling turns the program into binary data (ones and zeros). Uploading sends the bits through USB cable to the Arduino. The two LEDs near the USB
connector blink when data is transmitted. RX blinks when the Arduino is receiving data. TX blinks when the Arduino is transmitting data.

Both devices agree to transmit data at a speed of 9600 bits per second (9600 “baud”). Receiver will continually read the voltage sent by the Sender. Every 1/9600th of a second, the Receiver interprets this voltage as a new BIT of data. Arduino: +5V is a bit of 1, 0V is a bit of 0 (HIGH or LOW). This way, in one second, 1200 BYTES of data can be sent/second. a BIT is 0 or 1. There are 8 BITS in a BYTE. Each alphanumerical symbol takes up a byte. The ASCII code assigns a number from 0-255 to each byte.

C. Serial Data Transmission for PIC

This project is intended to show several aspects of how to communicate between the PC and a Pic microcontroller. The main focus is the communication itself. It will communicate via a RS232 link using the serial port of the PC and UART of the Microcontroller. The communication link has to pass through an interface to convert the serial information from TTL levels to RS232 levels. The inputs that are demonstrated are simple digital input through Port B and analog input through Ports A and E. The outputs provided are digital out through Port D and PWM output through bits 1 and 2 of Port C. The UART connections are also made through Port C on bits 6 and 7. This includes the eight DIP switches on Port B to provide the digital input, the LED board that consists of eight LEDs to show a change in digital output on Port D. It shows that one of the PWM outputs drives an H-bridge board that in turn drives a DC motor and a variable DC supply that provides an analog input on Port A.

IV. IMPLEMENTATION

Purpose of this project is to show the flowchart of serial data transmission for Arduino and PIC, flowchart of serial data reception for Arduino and PIC, simulation results for PIC16F887 (TX) and Arduino (RX). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer. Information passes between the computer and Arduino through the USB cable. Information is transmitted as zeros (‘0’ and ones (‘1’) also known as bits. At first, compiling turns the program into binary data (ones and zeros) and uploading sends the bits through USB cable to the Arduino. The two LEDs near the USB connector blink when data is transmitted RX blinks when the Arduino is receiving data and TX blinks when the Arduino is transmitting data.

To build a project it has to make a circuit using electrical and electronic components, connect them to the Arduino pins and then write the software on the PC: the sketch. But before writing the software, the IDE for Arduino type have been downloaded and then, on the editor, it can be written the program. The implementation described in this project to set up the Arduino ATmega328 to receive communication from the computer and then respond appropriately. In this case, the microcontroller will receive characters from the computer consisting of either ‘1’ or ‘0’. These characters will activate an LED by turning it on or off, by interpreting a ‘1’ as an “on” signal, and a ‘0’ as an “off” signal.

A. Arduino Program for Serial Data Transmission

```c
void setup()
{
    Serial.begin(9600);
    pinMode(2,INPUT);
    pinMode(3,INPUT);
}

void loop()
{
    if(digitalRead(2)==0&&digitalRead(3)==0)
        Serial.println("a");
    if(digitalRead(2)==0&&digitalRead(3)==1)
        Serial.println("b");
}```
if(digitalRead(2)==1&&digitalRead(3)==0) 
Serial.println("c");
if(digitalRead(2)==1&&digitalRead(3)==1) 
Serial.println("d");
delay (10); }

if(digitalRead(2)==0&&digitalRead(3)==0)Serial.println("a");
if(digitalRead(2)==0&&digitalRead(3)==1)Serial.println("b");
if(digitalRead(2)==1&&digitalRead(3)==0)Serial.println("c");
if(digitalRead(2)==1&&digitalRead(3)==1)Serial.println("d");

Figure 8. Flowchart of Serial Data Transmission for Arduino

B. PIC Program for Serial Data Reception

char uart_rd;
void main()
{UART1_Init(9600);
Delay_ms(100);
TRISD=0;
PORTD=0;
while(1)
{if (UART1_Data_Ready())
{uart_rd=UART1_Read();
switch(uart_rd)
{ case 'a':PORTD=0b00000000;break;
case 'b':PORTD=0b00000001;break;
case 'c':PORTD=0b00000010;break;
case 'd':PORTD=0b00000011;break;}}}}

Figure 9. Flowchart of Serial Data Reception for PIC

C. PIC Program Serial Data Transmission

void main()
{UART1_Init(9600);
Delay_ms(100);
TRISD=0b00000011;
PORTD=0;
while(1)
{if(!RD0_bit && !RD1_bit) UART1_Write_Text("a");
if(!RD0_bit && RD1_bit) UART1_Write_Text("b");
if(RD0_bit && !RD1_bit) UART1_Write_Text("c");
if(RD0_bit && RD1_bit) UART1_Write_Text("d");
Delay_ms(10); }}

D. Arduino Program for Serial Data Reception

void setup()
{ Serial.begin(9600);
pinMode(2,OUTPUT);
pinMode(3,OUTPUT); }
void loop()
V. SIMULATION RESULTS

For Arduino (TX) serial data transmission, first step run once in the program, when the sketch starts and sets the data rate in bits per second at 9600 bps for serial data transmission and then prints data to the serial port as human-readable ASCII text.

When the pushbutton on pin 2 and pin 3 as an input press, the value from pin 2 and pin 3 reads and check if both LOW, print out the character “a” with a line break. When the pushbutton on pin 2 and pin 3 as an input press, the value from pin 2 and pin 3 reads and check if pin 2 is LOW and pin 3 is HIGH, print out the character “b” with a line break.

When the pushbutton on pin 2 and pin 3 as an input press, the value from pin 2 and pin 3 reads and check if pin 2 is HIGH and pin 3 is LOW, print out the character “c” with a line break.

When the pushbutton on pin 2 and pin 3 as an input press, the value from pin 2 and pin 3 reads and check if both HIGH, print out the character “d” with a line break.

End function has a name, two parenthesis (void or with something inside) and then starts with a curly bracket and stops with a curly bracket.

For PIC16F887 (RX) serial data reception, declare variable and start the main program. This program uses the built in UART of the microcontroller and the UART is configured to operate with 9600 baud rate. The serial TX pin RC6 and the serial RX pin is RC7. Wait 100 million seconds. The outputs provided are digital out through Port D and Port D initialize to zero. If a character ready to read, it gets a character. If it receives the character “a”, both RD0 and RD1 will be LOW. If it receives the character “b”, RD0 will be HIGH and RD1 will be LOW. If it receives the character “c”, RD0 will be LOW and RD1 will be HIGH. If it receives the character “d”, RD0 will be HIGH and RD1 will be HIGH. End function has a name, two parenthesis (void or with something inside) and then starts with a curly bracket and stops with a curly bracket. Figure 10 to 22 shows simulation result for PIC16F887 (RX) and Arduino (TX).

For PIC16F887 (TX) serial data transmission, it starts the main program firstly. This program uses the built in UART of the microcontroller and the UART is configured to operate with 9600 baud rate. The serial TX pin RC6 and the serial RX pin is RC7. The UART connections are also made through Port C on bits 6 and 7. Chose the serial pin 25 (RC6) and wait 100 million seconds. The inputs provided are digital in through Port D (RD0 and RD1) and Port D initialize to zero. If RD0 and RD1 are not equal to 1, it will send the value “a”. If RD0 is not equal to 1 and RD1 is equal to 1, it will send the value “b”. If RD0 is
equal to 1 and RD1 is not equal to 1, it will send the value “c”. If RD0 and RD1 are equal to 1, it will send the value “d”. Wait 10 million seconds and stop with curly closing bracket. Figure 23 to 35 shows simulation result for PIC16F887 (TX) and Arduino (RX).

Figure 14. Simulation Result for Character “b” (PIC-RX and UNO-TX)

Figure 15. Output Waveform for Character “b” (PIC-RX and UNO-TX)

Figure 16. Output for Character “b” (PIC-RX and UNO-TX)

For Arduino (RX) serial data reception, the Arduino microcontroller needs to be set up to receive a character from the USB port and also needs to be programmed to respond appropriately based upon the received character. First step run once in the program, when the sketch starts and sets the data rate in bits per second at 9600 bps for serial data transmission and then prints data to the serial
port as human-readable ASCII text. At second step, the LED pin 2 and pin 3 configure the specified pins to behave either as the outputs. Declare the variable and read a character from serial interface. If it reads the character “a” from a specified digital pin, both pin 2 and pin 3 write a LOW. If it reads the character “b” from a specified digital pin, a, pin 2 writes a LOW and pin 3 writes a HIGH. If it reads the character “c” from a specified digital pin, a, pin 2 writes a HIGH and pin 3 writes a LOW. If it reads the character “d” from a specified digital pin, a, pin 2 writes a HIGH and pin 3 writes a HIGH.
Figure 23. Simulation Result for PIC16F887 (TX) and Arduino (RX)

Figure 24. Simulation Result for Character “a” (PIC-TX and UNO-RX)

Figure 25. Output Waveform for Character “a” (PIC-TX and UNO-RX)

Figure 26. Output for Character “a” (PIC-TX and UNO-RX)

Figure 27. Simulation Result for Character “b” (PIC-TX and UNO-RX)

Figure 28. Output Waveform for Character “b” (PIC-TX and UNO-RX)

Figure 29. Output for Character “b” (PIC-TX and UNO-RX)

Figure 30. Simulation Result for Character “c” (PIC-TX and UNO-RX)
VI. CONCLUSION

In this project, it has been covered the setup of the software, the design and programming of the user interface, as well as the programming of the Arduino ATmega328 firmware. By following these instructions, it should be a very straight-forward task to develop communication between the computer, PIC16F887 and the Arduino ATmega328 microcontroller. Once these techniques have been practiced, it can be applied to more complex situations, as well as to other microcontrollers. The Arduino ATmega328 can also be used to send data to the computer using the serial interface. The serial ports on the Arduino ATmega328 can be used to communicate to more complex devices, such as other microcontrollers. More complex Arduino Microcontrollers are capable of communication through a much wider array of communication mediums, such as WiFi, rather than just serial communication. The most common is ASCII (American Standard Code for Information Interchange), using 7 bits to encode 96 printable characters and 32 control characters. Data is transmitted asynchronously. UART provides a way to synchronize the receiver shift register with the transmitter shift register. This project uses Arduino ATmega328 as design language to achieve the modules of UART. Using Proteus 8 software, it has been implemented low power UART module and simulation for serial data transmission. It has to be showed simulation and test completely with Arduino and PIC16F887.
VII. REFERENCES


