Project 5: UART

ECEN 220: Introduction to Embedded Systems
University of Nebraska–Lincoln
April 26, 2021

Name: David Perez

Contents

1 Introduction

For this project we were tasked with understanding and using the UART serial transmission on our Arduino boards. UART is an ancronym for Universal Asynchronous Reciever-Transmitter. What this means is that because it is asynchronous it doesn't rely on the clock meaning the transmission speed is adjustable. Let me note that this is not a communication but rather a physical circuit in the microcontroller. Another thing to note is that it's said to be universal as there are countless different chircuit boards that use this type of transmission.

On our microcontroller the UART sends a single byte (8-bits) per transmission along with a start and a stop bit for data protection. This of course is also configuarable but for the purposes of this project we simply had those default settings. Another key thing to note about the UART is the baud rate. The baud rate is simply a unit to denote the rate or speed at which the transmission is performing at. For the program 1 and 2 we used a baud rate of 9600, and 57600 respectively.

2 Program Description

The programs that I had wrote for this project build off of one another. Both programs use the UART in a similar fashion so the set up for them and background functions didn't differ much. For the first of the two programs established the fundamental UART functions. One of the main functions in the progam was the uart_init_8N1(uint32_t baud). It's purpose was setting up the UART in a 8N1 configuration. This stands for 8 data bits, no parity bits, and 1 stop bit. This is a commonly used configuration for the UART so we used it here. Note that the parameter to this function is a variable called baud. This allows us to call this function with a specified baud rate that allows us to initialize the UART with whatever baud rate we desire.

For program 2 our objective was to make a calculator that we can communicate to with our computer. When using the serial plotter on the Arduino IDE we could send a string to our microcontroller that we would then parse and transmit back the desired operation of those number. For instance if we sent " 5 + 2" our program would read that string and send back another string containing both those numbers along with the summation of them. In order to achieve this we used some C libraries such as string.h which allowed use of the isdigit(), isspace(), and isalpha() functions. These functions are fairly straight forward and easy to use. The isdigit() function returns a 1 if a character is a digit and a 0 if it is not. The isspace() and isalpha() do the same thing but for white spaces and letters of the alphabet rather than digits.

2.1 Program 1

I'd like to now get into a little more depth about program 1 and explain a bit more about the functions that I had created and used. First I'm going to talk about the main function and what the program is doing at the high level then break down how each operation is done in their corresponding functions. In the main function we first initialize the UART and then we print two strings, one containing my name, and the other containing the project and program number. Following this we go into and infinite loop that reads the data recieved from in the RX register until we reach a new line character then print it back to the serial plotter. The function for retrieving data is called uart_gets_until() where we pass in a string, the strings size as well as the end character.

Before we send information to the serial plotter we first have to retrieve that data from the RX register. The UART has a interrupt that I used that would place each byte we got into a ring buffer. The interrupt would populate the ring butter unitl it reaches it's length where it would restart to the first index. While this is happening the uart_gets_until function is getting the new characters and placing them into a string. This function calls another important function called uart_getc(). This function checks if there is a new character in the ring buffer and if there is it returns that character. This continues until all new available chracters are read in the ring buffer by the uart_getc and until the new line character is read. The uart_gets_until function finishes populating the string which would then be sent over the tx line back to the serial plotter.

2.2 Program 2

For program 2 the underlying is entirely the same as in program 1. The same UART configuration was used along with the functions to recieve and transmit characters and strings over the UART. The difference lies in what we are doing with the strings that we are recieving and transmitting. As stated earlier our objective is to parse the incoming strings that are in the form of some sort of arithmatic equation, then transmit that equation back with its appropriate solution.

Inside the main loop I had two important loops, the first of which was an infinite loop and the other loop was as follows: while(str[i] != '\n'). The purpose of this was to increment over every character from the incoming string until we reach the newline character. Inside this while loop there were several other conditional statements in order to parse this string in the desired fashion. The first conditional was a while loop that would increment "i" for every white space in the string, this was done using the isspace() function. Following this was an if statement that checked for a minus sign. If there was a minus sign it would then check if there was a digit following it and if so, get that index and increment until we weren't indexed onto a digit. If there wasn't though, we would search for a digit, take note of its index, then loop until we werent indexed onto a digit. Then we would search for our operation using if states and taking not of which operation was in the string. After that we'd search again for the second number using the same method as before. Note that after each specific search we would increment over white spaces. Next, after we got all the data we needed we used the atol() function and the index of each number to convert them to a long int. Lastly, We placed all this information into a string and sent it back to the computer using the TX of the UART.

In the advent of an improper recieved sting I'd send a question marc back over the TX line. These scenarios would occur when the user sent double minus signs or had a letter included in their string.

3 Conclusion

During this project we dove into the UART hardware device that is on our Arduino nano. In program 1 I learned how to configure the UART in 8N1 configuration and create a function that allowed me to adjust it's baud rate. With this I was able to send and recieve strings to a from my computer to the Arduino. On top of this, I also created a calculator type algorithm that allowed me to perform arithematic operations by parsing a string. Because I was parsing strings I also learned about some of the different functions of string.h library that allowed me to easily decompose the string in the desired manner. All in all, I was able to better understand the UART and use it to send and recieve data to and from my computer and even parse those strings to maniputlate the data in any way that I desired.

4 Appendix

```
/** Includes **/
#include <stdint.h>
#include <avr/interrupt.h>

/** Memory mapped register defines **/
#define REG_DDRD  (*((volatile uint8_t*)0x2A))
#define REG_PORTD (*((volatile uint8_t*)0x2B))

#define REG_UBRR0H  (*((volatile uint8_t*)0xC5))
#define REG_UBRR0L  (*((volatile uint8_t*)0xC4))
#define REG_UDR0  (*((volatile uint8_t*)0xC6))
#define REG_UCSR0A  (*((volatile uint8_t*)0xC0))
#define REG_UCSR0B  (*((volatile uint8_t*)0xC1))
#define REG_UCSR0C  (*((volatile uint8_t*)0xC2))

/** Global interrupts register **/
#define REG_SREG   (*((volatile uint8_t*) 0x5F))

/** Defines **/
#define BIT0  0x01
#define BIT1  0x02
#define BIT2  0x04
#define BIT3  0x08
#define BIT4  0x10
#define BIT5  0x20
#define BIT6  0x40
#define BIT7  0x80
#define RX_BUFFER_MAX_LENGTH  (100)

/** Global Variables **/
volatile uint8_t g_RXComplete;
volatile uint8_t g_head = 0;
volatile uint8_t g_tail = 0;
volatile char g_buf[20];



int main ()
{
  //initialize the uart peripheral with a 9600 baud rate
  uart_init_8N1(9600);

  uart_puts("David Perez \n”);
  uart_puts("Project 5, Program 1 \n”);

  while(1)
  {
    char str[RX_BUFFER_MAX_LENGTH];
    uart_gets_until(str, RX_BUFFER_MAX_LENGTH, ’\n’);
    uart_puts(str);
  }
}

//Inializes the UART with 8 data bits, no parity bits, and 1 stop bit(8N1)
//argumnet allows for a customizable baud rate (in bits per second)
void uart_init_8N1(uint32_t baud)
{
  //Enable global interrupts in the SREG register
  REG_SREG |= BIT7;

  uint32_t mcuClock = 16000000;
  uint32_t ubrr0 = round((mcuClock / (16 * baud)) - 1);
  //apply our values into baud rate registers
  REG_UBRR0H = ubrr0 >> 8;
  REG_UBRR0L = ubrr0 & 0x00FF;

  //UART control and status register A
  //don’t want to use any of these bitfields so set all to 0
  REG_UCSR0A = 0x00;

  // Set up UCSR0C
  //  bits 7 and 6: UMSEL[1:0] = 0b00 for Asynchronous UART mode
  //  bits 5 and 4: UPM0[1:0] = 0b00 for disable parity bit
  //  bit 3: USBS0 = 0b0 for 1 stop bit
  // UCSZ0[2:0] = 0b011 for 8-bit data frames
  //  bits 2 and 1: UCSZ0[... 1:0] = 0b11
  // Concatenation: 0b00000110 == 0x06
  REG_UCSR0C = 0x06;

  // Set up UCSR0B (this register also enables the interrupt)
  //  bit 7: RXCIE0 = 0b1 to enable RX complete ISR
  //  bit 6: TXCIE0 = 0b0 to disable TX complete ISR
  //  bit 5: UDRIE0 = 0b0 to disable data register empty ISR
  //  bit 4: RXEN0 = 0b1 to enable UART receiver
  //  bit 3: TXEN0 = 0b1 to enable UART transmitter
  //  bit 2: UCSZ0[2 ...] = 0b0 (other two bits in UCSR0C)
  // Concatenation: 0b10011000 == 0x98
  REG_UCSR0B = 0x98;

  //initalize flag indicating RX complete
  g_RXComplete = 0;

  // Set pin TX (PD1) as an output
  REG_DDRD |= BIT1;

  // Set pin RX (PD0) as an input
  REG_DDRD &= ~BIT0;
}

//waits for a UART data register to be empty then
//transmits a single char to the cmoputer
void uart_putc(char c)
{
  while((REG_UCSR0A & BIT5) == 0)
  {
    // Wait for data register to be empty
  }

  // Re-transmit that same byte over the TX line of the UART (back to the computer)
  REG_UDR0 = c;
  g_RXComplete = 0;
}

void uart_puts(char* str)
{
  char c; // variable to hold each character from our string
  uint8_t i = 0;  //incrementing variable
  c = str[i]; //initializing our character to make sure it’s not null

  while(c != ’\0’)
  {
    c = str[i]; // put each value from the inputed string into a char
    uart_putc(c); // transmit each character
    i++;
  }
}

uint8_t uart_rx_available(void)
{
  //Return 1 if there in an unread char in the RX ring buffer
  // and returns 0 if not
  if(g_head == g_tail || (g_head + g_tail) == 0)
  {
    return 0;
  }else
  {
    return 1;
  }
}

char uart_getc(void)
{

  volatile uint8_t charAvailable = 0;
  while(charAvailable == 0)
  {
    //wait until there is a new char available in the RX ring buffer
    charAvailable = uart_rx_available();
  }
  char c = g_buf[g_tail]; //get the character from the ring buffer
  if(g_tail == 20)
  {
    g_tail=0;
  }else
  {
    g_tail++;
  }
  return c;
}

void uart_gets_until(char* buf, uint8_t buf_size, char end_char)
{
  uint8_t bufLen = 20;
  //this coditional checks if there are any bytes available to read by the buffer
  if((g_head >= g_tail && ((g_head - g_tail) != bufLen) || (g_head < g_tail && bufLen != bufLen
       - (g_tail-g_head))))
  {
    uint8_t j = 0;
    char c = uart_getc();
    g_tail--; //tail for our buffer gets iterated after well call getc for the initial check
    while(c != end_char && (j < (buf_size - 1)))
    {
      c = uart_getc();
      buf[j] = c;
      j++;
    }
    //add a null terminator to the end of the string and reset our increment variable
    buf[j] = ’\0’;
    j = 0;
  }
}


/** Interrupt **/
ISR(USART_RX_vect, ISR_BLOCK)
{
  //check if the head flag has reached the end of the ring buffer
  if(g_head == 20)
  {
    g_head = 0;
    g_buf[g_head] = REG_UDR0;
    g_head++;
  }else
  {
    g_buf[g_head] = REG_UDR0;
    g_head++;
  }

}
```

Listing 1: Program 1

```c
/** Includes **/
#include <stdint.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/** Memory mapped register defines **/
#define REG_DDRD  (*((volatile uint8_t*)0x2A))
#define REG_PORTD (*((volatile uint8_t*)0x2B))

#define REG_UBRR0H  (*((volatile uint8_t*)0xC5))
#define REG_UBRR0L  (*((volatile uint8_t*)0xC4))
#define REG_UDR0  (*((volatile uint8_t*)0xC6))
#define REG_UCSR0A  (*((volatile uint8_t*)0xC0))
#define REG_UCSR0B  (*((volatile uint8_t*)0xC1))
#define REG_UCSR0C  (*((volatile uint8_t*)0xC2))

/** Global interrupts register **/
#define REG_SREG   (*((volatile uint8_t*) 0x5F))

/** Defines **/
#define BIT0  0x01
#define BIT1  0x02
#define BIT2  0x04
#define BIT3  0x08
#define BIT4  0x10
#define BIT5  0x20
#define BIT6  0x40
#define BIT7  0x80
#define RX_BUFFER_MAX_LENGTH  (100)

/** Global Variables **/
volatile uint8_t g_RXComplete;
volatile uint8_t g_head = 0;
volatile uint8_t g_tail = 0;
volatile char g_buf[20];
volatile char c;


int main ()
{
  //initialize the uart peripheral
  uart_init_8N1(57600);

  while(1)
  {
    //variables used in the parsing algorithm
    char msg[80]; //message sent back to uart
    uint8_t i = 0;
    uint8_t errorFlag = 0;  //indicates invalid character
    uint8_t operation = 0;  //indicates which operation to perform

    //variables for numbers recieved over uart
    uint8_t num1 = 0;
    uint8_t num1Index = 0;

    uint8_t num2 = 0;
    uint8_t num2Index = 0;

    //get the incoming string from the uart
    char str[RX_BUFFER_MAX_LENGTH];
    uart_gets_until(str, RX_BUFFER_MAX_LENGTH, ’\n’);

    while(str[i] != ’\n’)
    {

      //check for white spacee
      while(isspace(str[i]))
      {
        i++;
      }

      //check if our first digit is negative
      if(str[i] == ’-’)
      {
        i++;
        if(isdigit(str[i]))
        {
          i--;
          num1Index = i;
          i++;
          while(isdigit(str[i]))
          {
            i++;
          }
        }
        else if(str[i] == ’-’){
          errorFlag = 1;
        }
      }


      //check if there if the first digit is not negative
      if (isdigit(str[i]))
      {
        num1Index = i;
        i++;

        //case where its  a positive first number then find the end of the number
        while(isdigit(str[i]))
        {
          i++;
        }
      }

      //check for white spacee
      while(isspace(str[i]))
      {
        i++;
      }

      /** check for the operation **/
      if(str[i] == ’-’)
      {
        operation = 1;
        i++;
      }
      else if(str[i] == ’+’)
      {
        operation = 2;
        i++;
      }
      else if(str[i] == ’*’)
      {
        operation = 3;
        i++;
      }
      else if(str[i] == ’/’)
      {
        operation = 4;
        i++;
      }

      //check for white spacee
      while(isspace(str[i]))
      {
        i++;
      }

      /* Check for second Number **/
      //check if our first digit is negative
      if(str[i] == ’-’)
      {
        i++;
        if(isdigit(str[i]))
        {
          i--;
          num2Index = i;
          i++;
          while(isdigit(str[i]))
          {
            i++;
          }
        }
        //double check that we don’t get double negative valuese
        else if(str[i] == ’-’){
          errorFlag = 1;
        }
      }

      //if second number is not negative we find it’s index and increment past the
      //last digit if it’s a multiple digit number
      if (isdigit(str[i]))
      {
        num2Index = i;
        i++;
        //case where its  a positive first number then find the end of the number
        while(isdigit(str[i]))
        {
          i++;
        }
      }
      //check if a character in the string is in the alphabet
      if(isalpha(str[i]))
      {
        errorFlag = 1;
      }

      /** Perform the operation and print to serial monitor**/
      //convert the string characters to integers
      int32_t a = atol(&str[num1Index]);
      int32_t b = atol(&str[num2Index]);
      int32_t result = 0;
      char operationSymbol;

      // Based on the character we got from a string we peroform the desired operation
      if(operation == 1)
      {
        operationSymbol = ’-’;
        result = a - b;
      }
      else if(operation == 2)
      {
        operationSymbol = ’+’;
        result = a + b;
      }
      else if(operation == 3)
      {
        operationSymbol = ’*’;
        result = a * b;
      }
      else if(operation == 4)
      {
        operationSymbol = ’/’;
        result = a / b;
      }

      if(errorFlag != 1)
      {
        //print out our equation to be performed and its results
        sprintf(msg,"%li %c %li = %li \n", a,operationSymbol , b, result);
        uart_puts(msg);
        errorFlag =0;
        char c = ’\n’;
        i = 0;
        str[i] = c ;
      }
      else
      {
        //print out a question mark if there is an incorrect string
        //such as characters and double minus signs
        char str2[] = "?\n";
        uart_puts(str2);
        errorFlag = 0;
        i = 0;
        str[i] = ’\n’;
      }
    }
  }
}

//Inializes the UART with 8 data bits, no parity bits, and 1 stop bit(8N1)
//argumnet allows for a customizable baud rate (in bits per second)
void uart_init_8N1(uint32_t baud)
{
  //Enable global interrupts in the SREG register
  REG_SREG |= BIT7;

  uint32_t mcuClock = 16000000;
  uint32_t ubrr0 = round((mcuClock / (16 * baud)) - 1);
  //apply our values into baud rate registers
  REG_UBRR0H = ubrr0 >> 8;
  REG_UBRR0L = ubrr0 & 0x00FF;

  //UART control and status register A
  //don’t want to use any of these bitfields so set all to 0
  REG_UCSR0A = 0x00;

  // Set up UCSR0C
  //  bits 7 and 6: UMSEL[1:0] = 0b00 for Asynchronous UART mode
  //  bits 5 and 4: UPM0[1:0] = 0b00 for disable parity bit
  //  bit 3: USBS0 = 0b0 for 1 stop bit
  // UCSZ0[2:0] = 0b011 for 8-bit data frames
  //  bits 2 and 1: UCSZ0[... 1:0] = 0b11
  // Concatenation: 0b00000110 == 0x06
  REG_UCSR0C = 0x06;

  // Set up UCSR0B (this register also enables the interrupt)
  //  bit 7: RXCIE0 = 0b1 to enable RX complete ISR
  //  bit 6: TXCIE0 = 0b0 to disable TX complete ISR
  //  bit 5: UDRIE0 = 0b0 to disable data register empty ISR
  //  bit 4: RXEN0 = 0b1 to enable UART receiver
  //  bit 3: TXEN0 = 0b1 to enable UART transmitter
  //  bit 2: UCSZ0[2 ...] = 0b0 (other two bits in UCSR0C)
  // Concatenation: 0b10011000 == 0x98
  REG_UCSR0B = 0x98;

  //initalize flag indicating RX complete
  g_RXComplete = 0;

  // Set pin TX (PD1) as an output
  REG_DDRD |= BIT1;

  // Set pin RX (PD0) as an input
  REG_DDRD &= ~BIT0;
}

//waits for a UART data register to be empty then
//transmits a single char to the cmoputer
void uart_putc(char c)
{
  while((REG_UCSR0A & BIT5) == 0)
  {
    // Wait for data register to be empty
  }

  // Re-transmit that same byte over the TX line of the UART (back to the computer)
  REG_UDR0 = c;
  g_RXComplete = 0;
}

void uart_puts(char* str)
{
  char c; // variable to hold each character from our string
  uint8_t i = 0;  //incrementing variable
  c = str[i]; //initializing our character to make sure it's not null

  while(c != '\0')
  {
    c = str[i]; // put each value from the inputed string into a char
    uart_putc(c); // transmit each character
    i++;
  }
}

uint8_t uart_rx_available(void)
{
  //Return 1 if there in an unread char in the RX ring buffer
  // and returns 0 if not
  if(g_head == g_tail || (g_head + g_tail) == 0)
  {
    return 0;
  }else
  {
    return 1;
  }
}

char uart_getc(void)
{

  volatile uint8_t charAvailable = 0;
  while(charAvailable == 0)
  {
    //wait until there is a new char available in the RX ring buffer
    charAvailable = uart_rx_available();
  }
  char c = g_buf[g_tail]; //get the character from the ring buffer
  if(g_tail == 20)
  {
    g_tail=0;
  }else
  {
    g_tail++;
  }
  return c;
}

```

```
void uart_gets_until(char* buf, uint8_t buf_size, char end_char)
{
  uint8_t bufLen = 20;
  //this coditional checks if there are any bytes available to read by the buffer
  if((g_head >= g_tail && ((g_head - g_tail) != bufLen) || (g_head < g_tail && bufLen != bufLen -
       (g_tail-g_head))))
  {
    uint8_t j = 0;
    char c = uart_getc();
    g_tail--; //tail for our buffer gets iterated after well call getc for the initial check
    while(c != end_char && (j < (buf_size - 1)))
    {
      c = uart_getc();
      buf[j] = c;
      j++;
    }
    //add a null terminator to the end of the string and reset our increment variable
    buf[j] = '\0';
    j = 0;
  }
}


/** Interrupt **/
ISR(USART_RX_vect, ISR_BLOCK)
{
  //check if the head flag has reached the end of the ring buffer
  if(g_head == 20)
  {
    g_head = 0;
    g_buf[g_head] = REG_UDR0;
    g_head++;
  }else
  {
    g_buf[g_head] = REG_UDR0;
    g_head++;
  }

}
```

Listing 2: Program 2