Project 5: UART

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

Name: David Perez

Contents

1	Introduction	3
	Program Description 2.1 Program 1	
3	Conclusion	4
4	Appendix	5

 $2 \ {\rm of} \ 13$

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 configurable 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 program 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 received 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 until 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 arithmetic 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 **/
 2
      #include <stdint.h>
 3
      #include <avr/interrupt.h>
4
5
      /** Memory mapped register defines **/
6
      #define REG_DDRD (*((volatile uint8_t*)0x2A))
      #define REG_PORTD (*((volatile uint8_t*)0x2B))
7
9
      #define REG_UBRR0H (*((volatile uint8_t*)0xC5))
10
      #define REG_UBRR0L (*((volatile uint8_t*)0xC4))
11
      #define REG_UDR0 (*((volatile uint8_t*)0xC6))
12
      #define REG_UCSR0A (*((volatile uint8_t*)0xC0))
13
      #define REG_UCSR0B (*((volatile uint8_t*)0xC1))
14
      #define REG_UCSR0C (*((volatile uint8_t*)0xC2))
15
16
      /** Global interrupts register **/
17
      #define REG_SREG (*((volatile uint8_t*) 0x5F))
18
19
      /** Defines **/
20
      #define BIT0 0x01
21
      #define BIT1
                    0x02
22
      #define BIT2
                    0 x 0 4
23
      #define BIT3 0x08
      #define BIT4
24
                    0 x 1 0
25
      #define BIT5
                    0x20
26
      #define BIT6
                    0 x 4 0
27
      #define BIT7 0x80
28
      #define RX_BUFFER_MAX_LENGTH (100)
29
30
      /** Global Variables **/
31
      volatile uint8_t g_RXComplete;
32
      volatile uint8_t g_head = 0;
33
      volatile uint8_t g_tail = 0;
34
      volatile char g_buf[20];
35
36
37
38
      int main ()
39
40
        //initialize the uart peripheral with a 9600 baud rate
41
        uart_init_8N1(9600);
42
43
        uart_puts("David Perez \n");
44
        uart_puts("Project 5, Program 1 \n");
45
46
        while(1)
47
48
          char str[RX_BUFFER_MAX_LENGTH];
49
          uart_gets_until(str, RX_BUFFER_MAX_LENGTH, '\n');
50
          uart_puts(str);
51
        }
52
      }
53
54
      //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)
55
56
      void uart_init_8N1(uint32_t baud)
57
58
        //Enable global interrupts in the SREG register
59
        REG_SREG |= BIT7;
60
61
        uint32_t mcuClock = 16000000;
62
        uint32_t ubrr0 = round((mcuClock / (16 * baud)) - 1);
63
        //apply our values into baud rate registers
64
        REG_UBRR0H = ubrr0 >> 8;
65
        REG_UBRR0L = ubrr0 & 0x00FF;
```

```
66
 67
         //UART control and status register A
 68
         //don't want to use any of these bitfields so set all to 0 \,
 69
         REG_UCSR0A = 0x00;
 70
 71
         // Set up UCSR0C
 72
         // bits 7 and 6: UMSEL[1:0] = 0b00 for Asynchronous UART mode
 73
         // bits 5 and 4: UPM0[1:0] = 0b00 for disable parity bit
 74
            bit 3: USBS0 = 0b0 for 1 stop bit
 75
         // UCSZ0[2:0] = 0b011 for 8-bit data frames
 76
         // bits 2 and 1: UCSZ0[... 1:0] = 0b11
 77
         // Concatenation: 0b00000110 == 0x06
 78
         REG_UCSR0C = 0x06;
 79
 80
         // Set up UCSR0B (this register also enables the interrupt)
 81
         // bit 7: RXCIE0 = 0b1 to enable RX complete ISR
 82
         // bit 6: TXCIE0 = 0b0 to disable TX complete ISR
 83
         // bit 5: UDRIE0 = 0b0 to disable data register empty ISR
 84
            bit 4: RXEN0 = 0b1 to enable UART receiver
         // bit 3: TXEN0 = 0b1 to enable UART transmitter
 85
 86
         // bit 2: UCSZ0[2 ...] = 0b0 (other two bits in UCSR0C)
         // Concatenation: 0b10011000 == 0x98
 87
 88
         REG_UCSR0B = 0x98;
 89
 90
         //initalize flag indicating RX complete
 91
         g_RXComplete = 0;
 92
 93
         // Set pin TX (PD1) as an output
 94
         REG_DDRD |= BIT1;
 95
 96
         // Set pin RX (PD0) as an input
 97
         REG_DDRD &= ~BIT0;
 98
 99
100
       //waits for a UART data register to be empty then
101
       //transmits a single char to the cmoputer
102
       void uart_putc(char c)
103
104
         while((REG_UCSR0A & BIT5) == 0)
105
         {
106
           // Wait for data register to be empty
107
108
109
         // Re-transmit that same byte over the TX line of the UART (back to the computer)
110
         REG_UDR0 = c;
111
         g_RXComplete = 0;
112
       }
113
114
       void uart_puts(char* str)
115
       {
116
         char c; // variable to hold each character from our string
117
         uint8_t i = 0; //incrementing variable
118
         c = str[i]; //initializing our character to make sure it's not null
119
120
         while(c != '\0')
121
122
           c = str[i]: // put each value from the inputed string into a char
123
           uart_putc(c); // transmit each character
124
           i++;
125
         }
126
       }
127
128
       uint8_t uart_rx_available(void)
129
130
         //Return 1 if there in an unread char in the RX ring buffer
131
         // and returns 0 if not
132
         if(g_head == g_tail || (g_head + g_tail) == 0)
133
```

```
134
                            return 0;
135
                       }else
136
137
                            return 1:
138
                       }
139
                  }
140
141
                  char uart_getc(void)
142
143
144
                       volatile uint8_t charAvailable = 0;
145
                       while(charAvailable == 0)
146
147
                            //wait until there is a new char available in the RX ring buffer
148
                            charAvailable = uart_rx_available();
149
                       char c = g_buf[g_tail]; //get the character from the ring buffer
150
151
                       if(g_tail == 20)
152
153
                            g_tail=0;
154
                       }else
155
                       {
156
                            g_tail++;
157
158
                        return c;
159
                  }
160
161
                  void uart_gets_until(char* buf, uint8_t buf_size, char end_char)
162
163
                       uint8_t bufLen = 20;
164
                        //this coditional checks if there are any bytes available to read by the buffer
165
                       if((g_head \ge g_tail \&\& ((g_head - g_tail) != bufLen) || (g_head < g_tail \&\& bufLen != bufLen || (g_head < g_tail &\& bufLen )| (g_head < g_tail &\& bufLen )
                                     - (g_tail-g_head))))
166
167
                            uint8_t j = 0;
168
                            char c = uart_getc();
169
                            g_{-}tail--; //tail for our buffer gets iterated after well call getc for the initial check
170
                            while(c != end_char && (j < (buf_size - 1)))</pre>
171
                            {
172
                                 c = uart_getc();
173
                                 buf[j] = c;
174
                                 j++;
175
176
                            //add a null terminator to the end of the string and reset our increment variable
177
                            buf[j] = ' \setminus 0';
178
                            j = 0;
179
                       }
180
                  }
181
182
183
                  /** Interrupt **/
184
                  ISR(USART_RX_vect, ISR_BLOCK)
185
186
                        //check if the head flag has reached the end of the ring buffer
187
                       if(g_head == 20)
188
189
                            g_head = 0;
190
                            g_buf[g_head] = REG_UDR0;
191
                            g_head++;
192
                       }else
193
194
                            g_buf[g_head] = REG_UDR0;
195
                            g_head++;
196
                       }
197
198
                  }
```

Listing 1: Program 1

```
/** Includes **/
    #include <stdint.h>
    #include <avr/interrupt.h>
4
    #include <stdio.h>
5
    #include <string.h>
6
    #include <ctype.h>
    /** Memory mapped register defines **/
9
    #define REG_DDRD (*((volatile uint8_t*)0x2A))
10
    #define REG_PORTD (*((volatile uint8_t*)0x2B))
11
12
    #define REG_UBRR0H (*((volatile uint8_t*)0xC5))
13
    #define REG_UBRR0L (*((volatile uint8_t*)0xC4))
14
    #define REG_UDR0 (*((volatile uint8_t*)0xC6))
15
    #define REG_UCSR0A (*((volatile uint8_t*)0xC0))
16
    #define REG_UCSR0B
                        (*((volatile uint8_t*)0xC1))
17
    #define REG_UCSR0C (*((volatile uint8_t*)0xC2))
18
19
    /** Global interrupts register **/
20
    #define REG_SREG (*((volatile uint8_t*) 0x5F))
21
22
    /** Defines **/
23
    #define BIT0 0x01
24
    #define BIT1 0x02
25
    #define BIT2
                  0 x 0 4
26
    #define BIT3 0x08
27
    #define BIT4 0x10
28
    #define BIT5 0x20
29
    #define BIT6 0x40
30
    #define BIT7 0x80
31
    #define RX_BUFFER_MAX_LENGTH (100)
32
33
    /** Global Variables **/
34
    volatile uint8_t g_RXComplete;
35
    volatile uint8_t g_head = 0;
36
    volatile uint8_t g_tail = 0;
37
    volatile char g_buf[20];
38
    volatile char c;
39
40
41
    int main ()
42
43
      //initialize the uart peripheral
44
      uart_init_8N1(57600);
45
46
      while(1)
47
48
        //variables used in the parsing algorithm
49
        char msg[80]; //message sent back to uart
50
        uint8 t i = 0:
51
        uint8_t errorFlag = 0; //indicates invalid character
52
        uint8_t operation = 0; //indicates which operation to perform
53
54
        //variables for numbers recieved over uart
55
        uint8_t num1 = 0;
56
        uint8_t num1Index = 0;
57
58
        uint8_t num2 = 0;
59
        uint8_t num2Index = 0;
60
61
        //get the incoming string from the uart
62
        char str[RX_BUFFER_MAX_LENGTH];
63
        uart_gets_until(str, RX_BUFFER_MAX_LENGTH, '\n');
64
65
        while(str[i] != '\n')
66
67
```

```
68
           //check for white spacee
 69
           while(isspace(str[i]))
 70
           {
 71
             i++;
 72
           }
 73
 74
            //check if our first digit is negative
 75
           if(str[i] == '-')
 76
           {
 77
             i++;
 78
              if(isdigit(str[i]))
 79
              {
 80
               i--;
 81
                num1Index = i;
 82
               i++;
 83
                while(isdigit(str[i]))
 84
                {
 85
                  i++;
 86
               }
 87
              }
 88
              else if(str[i] == '-'){
 89
                errorFlag = 1;
 90
              }
 91
           }
 92
 93
 94
           //check if there if the first digit is not negative
 95
           if (isdigit(str[i]))
 96
           {
 97
             num1Index = i;
 98
             i++;
 99
100
             //case where its a positive first number then find the end of the number
101
              while(isdigit(str[i]))
102
              {
103
                i++;
104
             }
105
           }
106
107
           //check for white spacee
108
           while(isspace(str[i]))
109
           {
110
             i++;
111
           }
112
113
            /** check for the operation **/
           if(str[i] == '-')
114
115
           {
116
              operation = 1;
117
             i++;
118
           else if(str[i] == '+')
119
120
           {
121
             operation = 2;
122
             i++;
123
           }
124
           else if(str[i] == '*')
125
           {
126
             operation = 3;
127
             i++;
128
           }
129
           else if(str[i] == '/')
130
           {
131
             operation = 4;
132
133
           }
134
135
           //check for white spacee
```

```
136
                                             while(isspace(str[i]))
137
                                             {
138
                                                    i++:
139
                                             }
140
141
                                             /* Check for second Number **/
142
                                             //check if our first digit is negative
143
                                             if(str[i] == '-')
144
                                             {
145
                                                     i++:
146
                                                     if(isdigit(str[i]))
147
                                                     {
148
                                                            i--;
 149
                                                             num2Index = i;
150
                                                             i++;
151
                                                             while(isdigit(str[i]))
152
                                                             {
153
                                                                    i++;
 154
                                                             }
155
                                                     }
156
                                                    //double check that we don't get double negative valuese
157
                                                     else if(str[i] == '-'){
158
                                                             errorFlag = 1;
159
                                                     }
160
                                             }
161
162
                                             //if second number is not negative we find it's index and increment past the
163
                                             //last digit if it's a multiple digit number
164
                                             if (isdigit(str[i]))
165
                                             {
166
                                                    num2Index = i;
167
                                                    i++;
168
                                                     //case where its a positive first number then find the end of the number
169
                                                     while(isdigit(str[i]))
170
                                                     {
 171
                                                             i++;
172
                                                     }
 173
174
                                             //check if a character in the string is in the alphabet
175
                                             if(isalpha(str[i]))
176
                                             {
177
                                                     errorFlag = 1;
 178
                                             }
179
180
                                             /** Perform the operation and print to serial monitor**/
181
                                             //convert the string characters to integers
182
                                             int32_t a = atol(&str[num1Index]);
 183
                                             int32_t b = atol(&str[num2Index]);
184
                                             int32_t result = 0;
185
                                             char operationSymbol;
186
187
                                             // Based on the character we got from a string we peroform the desired operation % \left( 1\right) =\left( 1\right) +\left( 1\right) +\left
188
                                             if(operation == 1)
189
                                             {
190
                                                     operationSymbol = '-';
191
                                                     result = a - b;
192
                                             }
193
                                             else if(operation == 2)
194
                                             {
195
                                                     operationSymbol = '+';
196
                                                     result = a + b;
197
                                             }
198
                                             else if(operation == 3)
199
                                             {
 200
                                                     operationSymbol = '*';
 201
                                                     result = a * b;
 202
203
                                             else if(operation == 4)
```

```
204
205
             operationSymbol = '/';
206
             result = a / b;
207
           }
208
209
           if(errorFlag != 1)
210
           {
211
             //print out our equation to be performed and its results
212
             sprintf(msg,"%li %c %li = %li \n", a,operationSymbol, b, result);
213
             uart_puts(msg);
214
             errorFlag =0;
215
             char c = ' \ n';
216
             i = 0;
217
             str[i] = c;
218
           }
219
           else
220
           {
221
             //print out a question mark if there is an incorrect string
222
             //such as characters and double minus signs
223
             char str2[] = "?\n";
224
             uart_puts(str2);
225
             errorFlag = 0;
226
             i = 0;
227
             str[i] = '\n';
228
229
         }
230
       }
231
     }
232
233
     //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)
235
     void uart_init_8N1(uint32_t baud)
236
237
       //Enable global interrupts in the SREG register
238
       REG_SREG |= BIT7;
239
240
       uint32_t mcuClock = 16000000;
241
       uint32_t ubrr0 = round((mcuClock / (16 * baud)) - 1);
242
       //apply our values into baud rate registers
243
       REG_UBRR0H = ubrr0 >> 8;
244
       REG_UBRR0L = ubrr0 & 0x00FF;
245
246
       //UART control and status register A
247
       //don't want to use any of these bitfields so set all to 0 \,
248
       REG_UCSR0A = 0x00;
249
250
       // Set up UCSR0C
251
       // bits 7 and 6: UMSEL[1:0] = 0b00 for Asynchronous UART mode
252
       // bits 5 and 4: UPM0[1:0] = 0b00 for disable parity bit
253
       // bit 3: USBS0 = 0b0 for 1 stop bit
254
       // UCSZ0[2:0] = 0b011 for 8-bit data frames
255
       // bits 2 and 1: UCSZ0[... 1:0] = 0b11
256
       // Concatenation: 0b00000110 == 0x06
257
       REG_UCSR0C = 0x06;
258
259
       // Set up UCSR0B (this register also enables the interrupt)
260
          bit 7: RXCIE0 = 0b1 to enable RX complete ISR
       11
261
           bit 6: TXCIE0 = 0b0 to disable TX complete ISR
262
       // bit 5: UDRIE0 = 0b0 to disable data register empty ISR
263
       // bit 4: RXEN0 = 0b1 to enable UART receiver
264
       // bit 3: TXEN0 = 0b1 to enable UART transmitter
265
          bit 2: UCSZ0[2 ...] = 0b0 (other two bits in UCSR0C)
266
       // Concatenation: 0b10011000 == 0x98
267
       REG_UCSR0B = 0x98;
268
269
       //initalize flag indicating RX complete
270
       g_RXComplete = 0;
271
```

```
272
       // Set pin TX (PD1) as an output
273
       REG_DDRD |= BIT1;
274
275
       // Set pin RX (PD0) as an input
276
       REG_DDRD &= ~BIT0;
277
278
279
     //waits for a UART data register to be empty then
280
     //transmits a single char to the cmoputer
281
     void uart_putc(char c)
282
283
       while((REG_UCSR0A & BIT5) == 0)
284
285
         // Wait for data register to be empty
286
287
288
       // Re-transmit that same byte over the TX line of the UART (back to the computer)
289
       REG_UDR0 = c;
290
       g_RXComplete = 0;
291
292
293
     void uart_puts(char* str)
294
     {
295
       char c; // variable to hold each character from our string
296
       uint8_t i = 0; //incrementing variable
297
       c = str[i]; //initializing our character to make sure it's not null
298
299
       while(c != '\0')
300
       {
301
         c = str[i]; // put each value from the inputed string into a char
302
         uart_putc(c); // transmit each character
303
304
       }
305
306
307
     uint8_t uart_rx_available(void)
308
309
       //Return 1 if there in an unread char in the RX ring buffer
310
       // and returns 0 if not
311
       if(g_head == g_tail || (g_head + g_tail) == 0)
312
       {
313
         return 0;
314
       }else
315
316
         return 1;
317
       }
318
     }
319
320
     char uart_getc(void)
321
322
323
       volatile uint8_t charAvailable = 0;
324
       while(charAvailable == 0)
325
326
         //wait until there is a new char available in the RX ring buffer
327
         charAvailable = uart_rx_available();
328
329
       char c = g_buf[g_tail]; //get the character from the ring buffer
330
       if(g_tail == 20)
331
       {
332
         g_tail=0;
333
       }else
334
335
         g_tail++;
336
337
       return c;
338
     }
339
```

```
340
               void uart_gets_until(char* buf, uint8_t buf_size, char end_char)
341
342
                      uint8_t bufLen = 20;
343
                      //this coditional checks if there are any bytes available to read by the buffer
344
                       if((g_head \ge g_tail \&\& ((g_head - g_tail) != buflen) || (g_head < g_tail \&\& buflen != buflen - g_tail && buflen != buflen - g_tail && buflen != buflen || 
                                      (g_tail-g_head))))
345
346
                            uint8_t j = 0;
347
                            char c = uart_getc();
                             g_tail--; //tail for our buffer gets iterated after well call getc for the initial check
348
349
                            while(c != end_char && (j < (buf_size - 1)))</pre>
350
351
                                  c = uart_getc();
352
                                  buf[j] = c;
353
                                  j++;
354
                             //add a null terminator to the end of the string and reset our increment variable
355
356
                            buf[j] = ' \setminus 0';
357
                            j = 0;
358
                      }
359
               }
360
361
362
                /** Interrupt **/
363
               ISR(USART_RX_vect, ISR_BLOCK)
364
365
                      //check if the head flag has reached the end of the ring buffer
366
                      if(g_head == 20)
367
                      {
368
                             g_head = 0;
369
                            g_buf[g_head] = REG_UDR0;
370
                             g_head++;
371
                      }else
372
                             g_buf[g_head] = REG_UDR0;
373
                             g_head++;
374
375
                      }
376
377
```

Listing 2: Program 2