

```
1
2
3 #include "UART_Bluetooth.h"
4 #include <avr/io.h>
5 #include <avr/interrupt.h>
6 #include "Command_Handler.h"
7 #include <stdlib.h>
8 #include <string.h>
9
10 uint8_t* uartBufferPos;
11 uint8_t* uartTxMessageEnd;
12 bool commandAvailable;
13
14 void initBluetoothUart(){
15     // UART Initialization : 8-bit : No parity bit : 1 stop bit
16     UBRR0H = (BRC >> 8); UBRR0L = BRC; // UART BAUDRATE
17     UCSR0A |= (1 << U2X0); // DOUBLE UART SPEED
18     UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // 8-BIT CHARACTER SIZE
19
20     // Setup UART buffer
21     initliazeMemory();
22     uartBufferPos = command_buffer;
23 }
24
25 void transmitMessage(uint8_t* message, uint8_t length){
26     while (!(UCSR0A & (1<<UDRE0)));
27     uartBufferPos = command_buffer;
28     uartTxMessageEnd = (command_buffer+length);
29     memcpy(command_buffer, message, length);
30     UCSR0A |= (1<<TXC0) | (1<<RXC0);
31     UCSR0B |= (1<<TXEN0) | (1<<TXCIE0);
32     UCSR0B &=~(1<<RXEN0) & ~(1<<RXCIE0);
33
34     uartBufferPos++;
35     UDR0 = *(command_buffer);
36 }
37
38 void transmitMessageSync(uint8_t* message, uint8_t length){
39     while (!(UCSR0A & (1<<UDRE0)));
40     uartBufferPos = command_buffer;
41     uartTxMessageEnd = (command_buffer+length);
42     memcpy(command_buffer, message, length);
43     UCSR0A |= (1<<TXC0) | (1<<RXC0);
44     UCSR0B |= (1<<TXEN0) | (1<<TXCIE0);
45     UCSR0B &=~(1<<RXEN0) & ~(1<<RXCIE0);
46
47     uartBufferPos++;
48     UDR0 = *(command_buffer);
49
50     while (transmissionState());
51
52 }
```

```
53
54 bool transmissionState(){
55     // True : Currently transmitting | False : Transmission finished
56     if (uartBufferPos!=uartTxMessageEnd)
57     {
58         return true;
59     }
60     else
61     {
62         return false;
63     }
64 }
65
66
67 void setupReceiveMode(){
68     while (!(UCSR0A & (1<<UDRE0)));
69     uartBufferPos = command_buffer;
70
71     UCSR0A |= (1<<RXC0) | (1<<TXC0);
72     UCSR0B &=~(1<<TXEN0) &~(1<<TXCIE0);
73     UCSR0B |= (1<<RXEN0) | (1<<RXCIE0);
74 }
75
76 void processReceivedLine(){
77     commandAvailable = false;
78
79     commandType currentCommand;
80     bool success = decomposeCommand(command_buffer, &currentCommand, parameter);
81     if(success) currentCommand.handlerFunction();
82 }
83
84 void disableUART(){
85     UCSR0B &=~(1<<TXEN0) &~(1<<TXCIE0);
86     UCSR0B &=~(1<<RXEN0) &~(1<<RXCIE0);
87 }
88
89 ISR(USART_TX_vect){
90     if (uartBufferPos!=uartTxMessageEnd){
91         UDR0 = *uartBufferPos;
92         uartBufferPos++;
93     }
94 }
95
96 ISR(USART_RX_vect){
97     if(uartBufferPos!=(command_buffer+uartBufferSize)) {
98         *uartBufferPos=UDR0;
99         if (*uartBufferPos!=uartEndMsgChar) {
100             if(*uartBufferPos!=uartCarriageReturnChar) {uartBufferPos++;} else
101                 { uartBufferPos = command_buffer; }
102             }
103         else { disableUART(); commandAvailable = true; }
104     } else {uartBufferPos = command_buffer;}
```

104 }