```
 1
 2
 3  #include "UART_Bluetooth.h"
 4  #include <avr/io.h>
 5  #include <avr/interrupt.h>
 6  #include "Command_Handler.h"
 7  #include "nrf24.h"
 8  #include <stdlib.h>
 9  #include <string.h>
10
11  uint8_t* uartBufferPos;
12  uint8_t* uartTxMessageEnd;
13  bool commandAvailable;
14
15  void initBluetoothUart(){
16      // UART Initialization : 8-bit : No parity bit : 1 stop bit
17      UBRR0H = (BRC >> 8); UBRR0L =  BRC;             // UART BAUDRATE
18      UCSR0A |= (1 << U2X0);                          // DOUBLE UART SPEED
19      UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);        // 8-BIT CHARACTER SIZE
20
21      // Setup UART buffer
22      initliazeMemory();
23      uartBufferPos = command_buffer;
24  }
25
26  void transmitMessage(uint8_t* message, uint8_t length){
27      while (!(UCSR0A & (1<<UDRE0)));
28      uartBufferPos = command_buffer;
29      uartTxMessageEnd = (command_buffer+length);
30      memcpy(command_buffer, message, length);
31      UCSR0A |= (1<<TXC0) | (1<<RXC0);
32      UCSR0B |= (1<<TXEN0) | (1<<TXCIE0);
33      UCSR0B &=~(1<<RXEN0) &~(1<<RXCIE0);
34
35      uartBufferPos++;
36      UDR0 = *(command_buffer);
37  }
38
39  void transmitMessageSync(uint8_t* message, uint8_t length){
40      while (!(UCSR0A & (1<<UDRE0)));
41      uartBufferPos = command_buffer;
42      uartTxMessageEnd = (command_buffer+length);
43      memcpy(command_buffer, message, length);
44      UCSR0A |= (1<<TXC0) | (1<<RXC0);
45      UCSR0B |= (1<<TXEN0) | (1<<TXCIE0);
46      UCSR0B &=~(1<<RXEN0) &~(1<<RXCIE0);
47      sei();
48
49      uartBufferPos++;
50      UDR0 = *(command_buffer);
51
52      while (transmissionState());
```

```
53
54  }
55
56  bool transmissionState(){
57      // True : Currently transmitting | False : Transmission finished
58      if (uartBufferPos!=uartTxMessageEnd)
59      {
60          return true;
61      }
62      else
63      {
64          return false;
65      }
66  }
67
68
69  void setupReceiveMode(){
70      while (!(UCSR0A & (1<<UDRE0)));
71      uartBufferPos = command_buffer;
72
73      UCSR0A |= (1<<RXC0) | (1<<TXC0);
74      UCSR0B &=~(1<<TXEN0) &~(1<<TXCIE0);
75      UCSR0B |= (1<<RXEN0) | (1<<RXCIE0);
76      sei();
77  }
78
79  bool catchModuleReply(){
80      nrf24_initRF_SAFE((lastTargetModuleID-1), RECEIVE); // CONNECTION TO MODULE:
        GENERAL RF CHANNEL 112 (lastTargetModuleID-1) offset 1
81      uint8_t targetModuleID = lastTargetModuleID;
82      uint8_t RF_TIME_OUT;
83      while(RF_TIME_OUT!=0xFF)
84      {
85          if(nrf24_dataReady()){
86              nrf24_getData(command_buffer);
87              CommandStatus status = DecomposeMessageFromBuffer();
88              if
                  (status==SUCCESFUL_DECOMPOSITION&&lastTargetModuleID==targetModuleI
                  D) {
89                  transmitMessageSync(command_buffer, 32);
90                  return true;
91              }
92          }
93          RF_TIME_OUT++; _delay_ms(2);
94      }
95      return false;
96  }
97
98  void processReceivedLine(){
99      commandAvailable = false;
100
101     CommandStatus status = DecomposeMessageFromBuffer();
```

```
102     if(status==SUCCESFUL_DECOMPOSITION) {
103         if (lastTargetModuleID==MAIN_MODULE){
104             //Executed by main module
105             HandleAvailableCommand();
106         } else {
107             //Retransmitted to other module
108 
109             RF_TransmissionStatus RF_Status = RetransmissionToModule();
110 
111             //Catch module reply
112 
113             //bool didModuleRelpy = catchModuleReply();
114 
115             // Send RF STATUS
116             switch (RF_Status) {
117                 case RF_UNREACHEABLE_MODULE:
118                 writeParameterValue(0, &(uint8_t){RETRANSMISSION_FAILED}, 1);
119                 break;
120                 case RF_ACKNOWLEDGE_FAILED:
121                 writeParameterValue(0, &(uint8_t){RETRANSMISSION_FAILED}, 1);
122                 break;
123                 case RF_SUCCESFUL_TRANSMISSION:
124                 writeParameterValue(0, &(uint8_t){SUCCESFUL_RETRANSMISSION}, 1);
125                 break;
126             }
127             ComposeMessageToBuffer(MESSAGE_STATUS_ID, 1, PHONE_MODULE);
128             transmitMessageSync(command_buffer, 32);
129 
130 
131         }
132     }else {
133 }
134 
135 
136 }
137 
138 void disableUART(){
139     UCSR0B &=~(1<<TXEN0) &~(1<<TXCIE0);
140     UCSR0B &=~(1<<RXEN0) &~(1<<RXCIE0);
141 }
142 
143 ISR(USART_TX_vect){
144     if (uartBufferPos!=uartTxMessageEnd){
145         UDR0 = *uartBufferPos;
146         uartBufferPos++;
147     }
148 }
149 
150 ISR(USART_RX_vect){
151     if(uartBufferPos!=(command_buffer+uartBufferSize)) {
152         *uartBufferPos=UDR0;
153         if ((*uartBufferPos==ETB)&&(DecomposeMessageFromBuffer()
```

```
            ==SUCCESFUL_DECOMPOSITION)) {
154             disableUART(); commandAvailable = true;
155         }
156         else if(*uartBufferPos==uartCarriageReturnChar) {
157
158             bool hasToReturnCarriage = true;
159             uint8_t* savedUartBufferPos = uartBufferPos+1;
160
161             for (uint8_t x = 1; x < 4; x++) {
162                 if ((uartBufferPos-x)<command_buffer) uartBufferPos =
                      command_buffer+(uartBufferSize-1);
163                 if (*(uartBufferPos-x)!=uartCarriageReturnChar)
                      { hasToReturnCarriage = false; break; }
164             }
165             if (hasToReturnCarriage) {
166                  uartBufferPos = command_buffer;
167
168             } else {
169                 uartBufferPos = savedUartBufferPos;
170             }
171
172         } else {
173             uartBufferPos++;
174         }
175
176     } else {
177         uartBufferPos = command_buffer;
178         *uartBufferPos=UDR0;
179     }
180 }
```