```
 1
 2  #include "Command_Handler.h"
 3  #include "UART_Bluetooth.h"
 4  #include "nrf24.h"
 5  #include "crc.h"
 6
 7
 8
 9  const CommandType commandList[] = {
10      { .handlerFunction = &UPDATE_ALL_DEVICES_VALUE_H},
11      { .handlerFunction = &UPDATE_DEVICE_VALUE_H},
12      { .handlerFunction = &GET_ALL_DEVICES_VALUE_H},
13      { .handlerFunction = &GET_DEVICE_VALUE_H},
14      { .handlerFunction = &MESSAGE_STATUS_H}
15  };
16  #define commandListLength (uint8_t)(sizeof commandList/sizeof commandList[0])
17
18  bool initliazeMemory(){
19      if(memoryInitialized) return false;
20      parameter[0].startingPointer = (void*)calloc(23,1);
21      parameter[1].startingPointer = (void*)calloc(2,1);
22      parameter[2].startingPointer = (void*)calloc(2,1);
23      for (uint8_t x = 3; x<12; x++) parameter[x].startingPointer = (void*)calloc
          (1,1);
24      command_buffer = (uint8_t*)calloc(32,1);
25      if(command_buffer==NULL) return false;
26      for (uint8_t x = 0; x<12; x++) { if(parameter[x].startingPointer==NULL)
          return false; }
27      memoryInitialized = true;
28      return true;
29  }
30
31  CommandStatus DecomposeMessageFromBuffer(){
32      // Search for header
33      uint8_t* headerStart = command_buffer;
34      uint8_t* footerEnd = command_buffer+31;
35
36      for(;headerStart!=(command_buffer+22);headerStart++){
37          if (*headerStart==SOH&&(*(headerStart+4)==STX)){
38              for(;footerEnd!=(command_buffer+6);footerEnd--){
39                  if (*footerEnd==ETB&&(*(footerEnd-2)==ETX)){
40                      uint8_t netMessageLength = ((footerEnd-2)-headerStart);
41                      crc_t crc;
42                      crc = crc_init();
43                      crc = crc_update(crc, headerStart, netMessageLength);
44                      crc = crc_finalize(crc);
45                      if (*(footerEnd-1)!=crc) return WRONG_CHECKSUM_CONSISTENCY;
46                      if (*(headerStart+2)!=currentModuleID&&*(headerStart+2)!
                         =0xFF&&currentModuleID!=0x01) return WRONG_MODULE_ID;
47                      lastTargetModuleID = *(headerStart+2);
48                      lastTransmitterModuleID = *(headerStart+3);
49                      if (*(headerStart+5)>commandListLength-1) return
```

```
                    UNDEFINED_COMMAND_CODE;
50                  lastMessageCommandType = commandList[*(headerStart+5)];
51                  lastMessagePID = *(headerStart+1);
52
53                  uint8_t* parameterStart = headerStart+6;
54
55                  for (uint8_t x = 0; x < 12; x++) {
56                      realloc(parameter[x].startingPointer, *parameterStart);
57                      parameter[x].byteLength = *parameterStart;
58                      memcpy(parameter[x].startingPointer,parameterStart+1,
                     *parameterStart);
59                      parameterStart+=((*parameterStart)+1);
60                      if (parameterStart>=(footerEnd-2)) break;
61                  }
62
63                  return SUCCESFUL_DECOMPOSITION;
64              }
65          }
66      }
67  }
68  return WRONG_HEADER_SEGMENTATION;
69 }
70
71 CommandStatus ComposeMessageToBuffer(CommandTypeID targetTypeID, uint8_t
    parameterCount, uint8_t targetBoardID){
72
73      memset(command_buffer, 0, 32);
74
75      command_buffer[0] = SOH;
76      if (lastMessagePID==0xFF) { lastMessagePID++; } else { lastMessagePID = 0; }
77      command_buffer[1] = lastMessagePID;
78      command_buffer[2] = targetBoardID;
79      command_buffer[3] = currentModuleID;
80      command_buffer[4] = STX;
81      command_buffer[5] = targetTypeID;
82
83      if (parameterCount>12) return PARAMETER_COUNT_OVERSIZE;
84
85      uint8_t* parameterStart = &command_buffer[6];
86
87      for (uint8_t x = 0; x < parameterCount; x++){
88          *parameterStart = parameter[x].byteLength;
89          memcpy(parameterStart+1, parameter[x].startingPointer, parameter
              [x].byteLength);
90          parameterStart+=(parameter[x].byteLength)+1;
91      }
92
93      crc_t crc;
94      crc = crc_init();
95      uint8_t crc_length = ((parameterStart)-(&command_buffer[0]));
96      crc = crc_update(crc, &command_buffer[0], crc_length);
97      crc = crc_finalize(crc);
```

```
 98
 99      *parameterStart = ETX;
100      *(parameterStart+1) = crc;
101      *(parameterStart+2) = ETB;
102
103      return SUCCESFUL_COMPOSITION;
104  }
105
106  void HandleAvailableCommand(){
107      lastMessageCommandType.handlerFunction();
108  }
109
110  RF_TransmissionStatus RetransmissionToModule(){
111      nrf24_initRF_SAFE((lastTargetModuleID-1), TRANSMIT);    // CONNECTION TO
           MODULE: GENERAL RF CHANNEL 112, (lastTargetModuleID-1) offset 1
112      nrf24_send(command_buffer);
113      while(nrf24_isSending());
114
115      uint8_t messageStatus = nrf24_lastMessageStatus();
116      if(messageStatus == NRF24_TRANSMISSON_OK) { return
           RF_SUCCESFUL_TRANSMISSION; }
117      else if(messageStatus == NRF24_MESSAGE_LOST) { return
           RF_UNREACHEABLE_MODULE;}
118      return RF_UNREACHEABLE_MODULE;
119  }
120
121  void RetransmissionToPhone(){
122      transmitMessageSync(command_buffer, 32);
123  }
124
125
126
127  void writeParameterValue(uint8_t parameterIndex, uint8_t* parameterData, uint8_t
       parameterByteLength){
128      parameter[parameterIndex].startingPointer = (uint8_t*) realloc(parameter
           [parameterIndex].startingPointer, parameterByteLength);
129      memcpy(parameter[parameterIndex].startingPointer, parameterData,
           parameterByteLength);
130      parameter[parameterIndex].byteLength = parameterByteLength;
131  }
132
133  void UPDATE_ALL_DEVICES_VALUE_H() {}
134  void UPDATE_DEVICE_VALUE_H() {}
135  void GET_ALL_DEVICES_VALUE_H() {
136      _delay_ms(100);
137
138      uint8_t boardState[2];
139
140      ComposeMessageToBuffer(MESSAGE_STATUS_ID, 0, POWER_MODULE);
141      nrf24_initRF_SAFE(POWER_BOARD_RF, TRANSMIT);    // CONNECTION TO MODULE:
           GENERAL RF CHANNEL 112
142      nrf24_send(command_buffer);
```

```
143     while(nrf24_isSending());
144
145     uint8_t messageStatus = nrf24_lastMessageStatus();
146     if(messageStatus == NRF24_TRANSMISSON_OK) { boardState[0] = 0xFF; }
147     else if(messageStatus == NRF24_MESSAGE_LOST) { boardState[0]= 0x00; }
148
149     _delay_ms(50);
150
151     ComposeMessageToBuffer(MESSAGE_STATUS_ID, 0, MOTOR_MODULE);
152     nrf24_initRF_SAFE(MOTORIZED_BOARD_RF, TRANSMIT);    // CONNECTION TO MODULE:
          GENERAL RF CHANNEL 112
153     nrf24_send(command_buffer);
154     while(nrf24_isSending());
155
156     uint8_t messageStatusSecond = nrf24_lastMessageStatus();
157     if(messageStatusSecond == NRF24_TRANSMISSON_OK) { boardState[1] = 0xFF; }
158     else if(messageStatusSecond == NRF24_MESSAGE_LOST) { boardState[1]= 0x00; }
159
160
161     writeParameterValue(0, &boardState[0], 1);
162     writeParameterValue(1, &boardState[1], 1);
163     ComposeMessageToBuffer(UPDATE_ALL_DEVICES_VALUE_ID, 2, PHONE_MODULE);  //
          PHONE_MODULE should be lastTransmitterModuleID
164     transmitMessageSync(command_buffer, 32);
165 }
166
167 void GET_DEVICE_VALUE_H() {
168     _delay_ms(100);
169     uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
170     uint8_t deviceValue;
171
172     switch(deviceIndex){
173         case 0:
174             ComposeMessageToBuffer(MESSAGE_STATUS_ID, 0, POWER_MODULE);
175             nrf24_initRF_SAFE(POWER_BOARD_RF, TRANSMIT);    // CONNECTION TO
                  MODULE: GENERAL RF CHANNEL 112
176             nrf24_send(command_buffer);
177             while(nrf24_isSending());
178
179             uint8_t messageStatus = nrf24_lastMessageStatus();
180             if(messageStatus == NRF24_TRANSMISSON_OK) { deviceValue = 0xFF; }
181             else if(messageStatus == NRF24_MESSAGE_LOST) { deviceValue= 0x00; }
182             break;
183         case 1:
184             ComposeMessageToBuffer(MESSAGE_STATUS_ID, 0, MOTOR_MODULE);
185             nrf24_initRF_SAFE(MOTORIZED_BOARD_RF, TRANSMIT);    // CONNECTION TO
                  MODULE: GENERAL RF CHANNEL 112
186             nrf24_send(command_buffer);
187             while(nrf24_isSending());
188
189             uint8_t messageStatusSecond = nrf24_lastMessageStatus();
190             if(messageStatusSecond == NRF24_TRANSMISSON_OK) { deviceValue =
```

```
                0xFF; }
191             else if(messageStatusSecond == NRF24_MESSAGE_LOST) { deviceValue=
                0x00; }
192             break;
193     }
194
195     writeParameterValue(0, &deviceIndex, 1);
196     writeParameterValue(1, &deviceValue, 2);
197
198     ComposeMessageToBuffer(UPDATE_DEVICE_VALUE_ID, 2, PHONE_MODULE);  //
        PHONE_MODULE should be lastTransmitterModuleID
199
200     transmitMessageSync(command_buffer, 32);
201 }
202 void MESSAGE_STATUS_H() {}
```