```c

#include "Command_Handler.h"
#include "nrf24.h"
#include "crc.h"



const CommandType commandList[] = {
    { .handlerFunction = &UPDATE_ALL_DEVICES_VALUE_H},
    { .handlerFunction = &UPDATE_DEVICE_VALUE_H},
    { .handlerFunction = &GET_ALL_DEVICES_VALUE_H},
    { .handlerFunction = &GET_DEVICE_VALUE_H},
    { .handlerFunction = &MESSAGE_STATUS_H}
};
#define commandListLength (uint8_t)(sizeof commandList/sizeof commandList[0])

bool initliazeMemory(){
    if(memoryInitialized) return false;
    parameter[0].startingPointer = (void*)calloc(23,1);
    parameter[1].startingPointer = (void*)calloc(2,1);
    parameter[2].startingPointer = (void*)calloc(2,1);
    for (uint8_t x = 3; x<12; x++) parameter[x].startingPointer = (void*)calloc(1,1);
    command_buffer = (uint8_t*)calloc(32,1);
    if(command_buffer==NULL) return false;
    for (uint8_t x = 0; x<12; x++) { if(parameter[x].startingPointer==NULL) return false; }
    memoryInitialized = true;
    return true;
}

CommandStatus DecomposeMessageFromBuffer(){
    // Search for header
    uint8_t* headerStart = command_buffer;
    uint8_t* footerEnd = command_buffer+31;

    for(;headerStart!=(command_buffer+22);headerStart++){
        if (*headerStart==SOH&&(*(headerStart+4)==STX)){
            for(;footerEnd!=(command_buffer+6);footerEnd--){
                if (*footerEnd==ETB&&(*(footerEnd-2)==ETX)){
                    uint8_t netMessageLength = ((footerEnd-2)-headerStart);
                    crc_t crc;
                    crc = crc_init();
                    crc = crc_update(crc, headerStart, netMessageLength);
                    crc = crc_finalize(crc);
                    if (*(footerEnd-1)!=crc) return WRONG_CHECKSUM_CONSISTENCY;
                    if (*(headerStart+2)!=currentModuleID&&*(headerStart+2)!=0xFF&&currentModuleID!=0x01) return WRONG_MODULE_ID;
                    lastTargetModuleID = *(headerStart+2);
                    lastTransmitterModuleID = *(headerStart+3);
                    if (*(headerStart+5)>commandListLength-1) return UNDEFINED_COMMAND_CODE;
```

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

```
 98      *(parameterStart+2) = ETB;
 99
100      return SUCCESFUL_COMPOSITION;
101  }
102
103  void HandleAvailableCommand(){
104      lastMessageCommandType.handlerFunction();
105  }
106
107  RF_TransmissionStatus RetransmissionToModule(){
108      nrf24_initRF_SAFE(lastTargetModuleID, TRANSMIT);    // CONNECTION TO MODULE:
           GENERAL RF CHANNEL 112
109      nrf24_send(command_buffer);
110      while(nrf24_isSending());
111
112      uint8_t messageStatus = nrf24_lastMessageStatus();
113      if(messageStatus == NRF24_TRANSMISSON_OK) { return
           RF_SUCCESFUL_TRANSMISSION; }
114      else if(messageStatus == NRF24_MESSAGE_LOST) { return
           RF_UNREACHEABLE_MODULE;}
115      return RF_UNREACHEABLE_MODULE;
116  }
117
118  void writeParameterValue(uint8_t parameterIndex, void* parameterData, uint8_t
       parameterByteLength){
119      parameter[parameterIndex].startingPointer = (uint8_t*) realloc(parameter
           [parameterIndex].startingPointer, parameterByteLength);
120      memcpy(parameter[parameterIndex].startingPointer, parameterData,
           parameterByteLength);
121      parameter[parameterIndex].byteLength = parameterByteLength;
122  }
123
124  void UPDATE_ALL_DEVICES_VALUE_H() {
125      for (uint8_t x = 0; x < AVAILABLE_DEVICES;x++)
126      {
127          deviceStoredValue[x] = *((uint8_t*)parameter[x].startingPointer);
128          switch (x) {
129              case 0x00:
130                  bit_write(deviceStoredValue[x], PORTD, BIT(3));
131                  break;
132              case 0x01:
133                  bit_write(deviceStoredValue[x], PORTD, BIT(2));
134                  break;
135              case 0x02:
136                  bit_write(deviceStoredValue[x], PORTD, BIT(6));
137                  break;
138              case 0x03:
139                  bit_write(deviceStoredValue[x], PORTD, BIT(5));
140                  break;
141          }
142      }
143
```

```
144
145
146 }
147
148 void UPDATE_DEVICE_VALUE_H() {
149     uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
150     uint8_t deviceValue = *((uint8_t*)parameter[1].startingPointer);
151
152     switch (deviceIndex) {
153         case 0:
154             bit_write(deviceValue, PORTD, BIT(3));
155             break;
156         case 1:
157             bit_write(deviceValue, PORTD, BIT(2));
158             break;
159         case 2:
160             bit_write(deviceValue, PORTD, BIT(6));
161             break;
162         case 3:
163             bit_write(deviceValue, PORTD, BIT(5));
164             break;
165     }
166
167     deviceStoredValue[deviceIndex] = deviceValue;
168
169 }
170 void GET_ALL_DEVICES_VALUE_H() {
171     _delay_ms(50);
172
173     for (uint8_t x = 0; x < AVAILABLE_DEVICES;x++)
174     {
175         writeParameterValue(x, &deviceStoredValue[x], 2);
176     }
177
178     ComposeMessageToBuffer(UPDATE_ALL_DEVICES_VALUE_ID, AVAILABLE_DEVICES,
        PHONE_MODULE);  // PHONE_MODULE deberia ser lastTransmitterModuleID
179
180     nrf24_initRF_SAFE(MAIN_BOARD, TRANSMIT);
181     nrf24_send(command_buffer);
182     while(nrf24_isSending());
183     uint8_t messageStatus = nrf24_lastMessageStatus();
184 }
185 void GET_DEVICE_VALUE_H() {
186     _delay_ms(50);
187     uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
188     writeParameterValue(0, &deviceIndex, 1);
189     writeParameterValue(1, &deviceStoredValue[deviceIndex], 2);
190     ComposeMessageToBuffer(UPDATE_DEVICE_VALUE_ID, 2, PHONE_MODULE);  //
        PHONE_MODULE deberia ser lastTransmitterModuleID
191
192     nrf24_initRF_SAFE(MAIN_BOARD, TRANSMIT);
193     nrf24_send(command_buffer);
```

```
194     while(nrf24_isSending());
195     uint8_t messageStatus = nrf24_lastMessageStatus();
196 }
197
198
199 void MESSAGE_STATUS_H() {}
```