```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  void HandleAvailableCommand(){
71      lastMessageCommandType.handlerFunction();
72  }
73
74  CommandStatus ComposeMessageToBuffer(CommandTypeID targetTypeID, uint8_t
      parameterCount, uint8_t targetBoardID){
75      memset(command_buffer, 0, 32);
76      command_buffer[0] = SOH;
77      if (lastMessagePID==0xFF) { lastMessagePID++; } else { lastMessagePID = 0; }
78      command_buffer[1] = lastMessagePID;
79      command_buffer[2] = targetBoardID;
80      command_buffer[3] = currentModuleID;
81      command_buffer[4] = STX;
82      command_buffer[5] = targetTypeID;
83
84      if (parameterCount>12) return PARAMETER_COUNT_OVERSIZE;
85
86      uint8_t* parameterStart = &command_buffer[6];
87
88      for (uint8_t x = 0; x < parameterCount; x++){
89          *parameterStart = parameter[x].byteLength;
90          memcpy(parameterStart+1, parameter[x].startingPointer, parameter
              [x].byteLength);
91          parameterStart+=(parameter[x].byteLength)+1;
92      }
93
94      crc_t crc;
95      crc = crc_init();
96      uint8_t crc_length = ((parameterStart)-(&command_buffer[0]));
97      crc = crc_update(crc, &command_buffer[0], crc_length);
```

```
 98      crc = crc_finalize(crc);
 99
100      *parameterStart = ETX;
101      *(parameterStart+1) = crc;
102      *(parameterStart+2) = ETB;
103
104      return SUCCESFUL_COMPOSITION;
105  }
106
107  void writeParameterValue(uint8_t parameterIndex, void* parameterData, uint8_t
       parameterByteLength){
108      parameter[parameterIndex].startingPointer = (uint8_t*) realloc(parameter
           [parameterIndex].startingPointer, parameterByteLength);
109      memcpy(parameter[parameterIndex].startingPointer, parameterData,
           parameterByteLength);
110      parameter[parameterIndex].byteLength = parameterByteLength;
111  }
112
113  void UPDATE_ALL_DEVICES_VALUE_H() {
114      for (uint8_t x = 0; x < AVAILABLE_DEVICES;x++)
115      {
116          deviceStoredValue[x] = *((uint8_t*)parameter[x].startingPointer);
117
118          switch (x) {
119              case 0:
120                  STRETCHER_POS_CHANGE_HANDLE(deviceStoredValue[x]);
121              break;
122              case 1:
123                  CURTAIN_POS_CHANGE_HANDLE(deviceStoredValue[x]);
124              break;
125              case 2:
126                  if (deviceStoredValue[x]==0xFF){
127                      for (uint8_t x = 0; x < 6; x++)
128                      {
129                          bit_flip(PORTB, BIT(0));
130                          bit_flip(PORTB, BIT(1));
131                          bit_flip(PORTB, BIT(2));
132                          _delay_ms(200);
133                      }
134                      bit_clear(PORTB, BIT(0));
135                      bit_clear(PORTB, BIT(1));
136                      bit_clear(PORTB, BIT(2));
137                  }
138              break;
139          }
140      }
141
142  }
143
144  #define MOTOR_DELAY_MS 1
145  #define CURTAIN_CALIBRATION_CONSTANT 200
146  #define STRETCHER_CALIBRATION_CONSTANT 50
```

```
147
148 void UPDATE_DEVICE_VALUE_H() {
149     const uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
150     const uint8_t deviceValue = *((uint8_t*)parameter[1].startingPointer);
151
152     switch (deviceIndex) {
153         case 0:
154             STRETCHER_POS_CHANGE_HANDLE(deviceValue);
155         break;
156         case 1:
157             CURTAIN_POS_CHANGE_HANDLE(deviceValue);
158         break;
159         case 2:
160             for (uint8_t x = 0; x < 6; x++)
161             {
162                 bit_flip(PORTB, BIT(0));
163                 bit_flip(PORTB, BIT(1));
164                 bit_flip(PORTB, BIT(2));
165                 _delay_ms(200);
166             }
167             bit_clear(PORTB, BIT(0));
168             bit_clear(PORTB, BIT(1));
169             bit_clear(PORTB, BIT(2));
170         break;
171     }
172
173     deviceStoredValue[deviceIndex] = deviceValue;
174
175 }
176
177 void GET_ALL_DEVICES_VALUE_H() {}
178
179 void GET_DEVICE_VALUE_H() {
180     _delay_ms(100);
181     uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
182     writeParameterValue(0, &deviceIndex, 1);
183     writeParameterValue(1, &deviceStoredValue[deviceIndex], 2);
184     ComposeMessageToBuffer(UPDATE_DEVICE_VALUE_ID, 2, 0x7C);
185
186     nrf24_initRF_SAFE(MAIN_BOARD, TRANSMIT);
187     nrf24_send(command_buffer);
188     while(nrf24_isSending());
189     uint8_t messageStatus = nrf24_lastMessageStatus();
190 }
191 void MESSAGE_STATUS_H() {}
192
193
194 uint8_t previousCurtainPosition = 0;
195 uint8_t previousStretcherPosition = 0;
196
197
198 void CURTAIN_POS_CHANGE_HANDLE(uint8_t positionToMove){
```

```
199     bit_set(PORTB, BIT(1));
200     bit_set(PORTB, BIT(2));
201
202
203     if (positionToMove<8) {
204         uint16_t degreesToMove = abs(positionToMove-previousCurtainPosition)
              *CURTAIN_CALIBRATION_CONSTANT;
205
206         if((positionToMove-previousCurtainPosition)>0){
207             for (uint16_t x = 0; x < degreesToMove;x++){
208                 PORTD = 0b00000011;
209                 _delay_ms(MOTOR_DELAY_MS);
210                 PORTD = 0b00000110;
211                 _delay_ms(MOTOR_DELAY_MS);
212                 PORTD = 0b00001100;
213                 _delay_ms(MOTOR_DELAY_MS);
214                 PORTD = 0b00001001;
215                 _delay_ms(MOTOR_DELAY_MS);
216             }
217             }else{
218             for (uint16_t x = 0; x < degreesToMove;x++){
219                 PORTD = 0b00001100;
220                 _delay_ms(MOTOR_DELAY_MS);
221                 PORTD = 0b00000110;
222                 _delay_ms(MOTOR_DELAY_MS);
223                 PORTD = 0b00000011;
224                 _delay_ms(MOTOR_DELAY_MS);
225                 PORTD = 0b00001001;
226                 _delay_ms(MOTOR_DELAY_MS);
227             }
228         }
229
230         PORTD = 0b00000000;
231         previousCurtainPosition = positionToMove;
232     }
233     bit_clear(PORTB, BIT(1));
234     bit_clear(PORTB, BIT(2));
235 }
236
237 void STRETCHER_POS_CHANGE_HANDLE(uint8_t positionToMove){
238     bit_set(PORTB, BIT(1));
239     bit_set(PORTB, BIT(2));
240
241     if (positionToMove<4) {
242         uint16_t degreesToMove = abs(positionToMove-previousStretcherPosition)
              *STRETCHER_CALIBRATION_CONSTANT;
243
244         if((positionToMove-previousCurtainPosition)>0){
245             for (uint16_t x = 0; x < degreesToMove;x++){
246                 PORTD = 0b00110000;
247                 _delay_ms(MOTOR_DELAY_MS);
248                 PORTD = 0b01100000;
```

```
249                 _delay_ms(MOTOR_DELAY_MS);
250                 PORTD = 0b11000000;
251                 _delay_ms(MOTOR_DELAY_MS);
252                 PORTD = 0b10010000;
253                 _delay_ms(MOTOR_DELAY_MS);
254             }
255             }else{
256             for (uint16_t x = 0; x < degreesToMove;x++){
257                 PORTD = 0b11000000;
258                 _delay_ms(MOTOR_DELAY_MS);
259                 PORTD = 0b01100000;
260                 _delay_ms(MOTOR_DELAY_MS);
261                 PORTD = 0b00110000;
262                 _delay_ms(MOTOR_DELAY_MS);
263                 PORTD = 0b10010000;
264                 _delay_ms(MOTOR_DELAY_MS);
265             }
266         }
267 
268         PORTD = 0b00000000;
269         previousStretcherPosition = positionToMove;
270     }
271     bit_clear(PORTB, BIT(1));
272     bit_clear(PORTB, BIT(2));
273 }
```