```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;
                    lastMessageCommandType = commandList[*(headerStart+5)];
                    lastMessagePID = *(headerStart+1);

                    uint8_t* parameterStart = headerStart+6;

                    for (uint8_t x = 0; x < 12; x++) {
                        realloc(parameter[x].startingPointer, *parameterStart);
                        parameter[x].byteLength = *parameterStart;
                        memcpy(parameter[x].startingPointer,parameterStart+1, *parameterStart);
                        parameterStart+=((*parameterStart)+1);
                        if (parameterStart>=(footerEnd-2)) break;
                    }

                    return SUCCESFUL_DECOMPOSITION;
                }
            }
        }
    }
    return WRONG_HEADER_SEGMENTATION;
}

void HandleAvailableCommand(){
    lastMessageCommandType.handlerFunction();
}

CommandStatus ComposeMessageToBuffer(CommandTypeID targetTypeID, uint8_t parameterCount, uint8_t targetBoardID){
    memset(command_buffer, 0, 32);
    command_buffer[0] = SOH;
    if (lastMessagePID==0xFF) { lastMessagePID++; } else { lastMessagePID = 0; }
    command_buffer[1] = lastMessagePID;
    command_buffer[2] = targetBoardID;
    command_buffer[3] = currentModuleID;
    command_buffer[4] = STX;
    command_buffer[5] = targetTypeID;

    if (parameterCount>12) return PARAMETER_COUNT_OVERSIZE;

    uint8_t* parameterStart = &command_buffer[6];

    for (uint8_t x = 0; x < parameterCount; x++){
        *parameterStart = parameter[x].byteLength;
        memcpy(parameterStart+1, parameter[x].startingPointer, parameter[x].byteLength);
        parameterStart+=(parameter[x].byteLength)+1;
    }

    crc_t crc;
    crc = crc_init();
    uint8_t crc_length = ((parameterStart)-(&command_buffer[0]));
    crc = crc_update(crc, &command_buffer[0], crc_length);
```

```c
    crc = crc_finalize(crc);

    *parameterStart = ETX;
    *(parameterStart+1) = crc;
    *(parameterStart+2) = ETB;

    return SUCCESFUL_COMPOSITION;
}

void writeParameterValue(uint8_t parameterIndex, void* parameterData, uint8_t parameterByteLength){
    parameter[parameterIndex].startingPointer = (uint8_t*) realloc(parameter[parameterIndex].startingPointer, parameterByteLength);
    memcpy(parameter[parameterIndex].startingPointer, parameterData, parameterByteLength);
    parameter[parameterIndex].byteLength = parameterByteLength;
}

void UPDATE_ALL_DEVICES_VALUE_H() {
    for (uint8_t x = 0; x < AVAILABLE_DEVICES;x++)
    {
        deviceStoredValue[x] = *((uint8_t*)parameter[x].startingPointer);

        switch (x) {
            case 0:
                STRETCHER_POS_CHANGE_HANDLE(deviceStoredValue[x]);
            break;
            case 1:
                CURTAIN_POS_CHANGE_HANDLE(deviceStoredValue[x]);
            break;
            case 2:
                if (deviceStoredValue[x]==0xFF){
                    for (uint8_t x = 0; x < 6; x++)
                    {
                        bit_flip(PORTB, BIT(0));
                        bit_flip(PORTB, BIT(1));
                        bit_flip(PORTB, BIT(2));
                        _delay_ms(200);
                    }
                    bit_clear(PORTB, BIT(0));
                    bit_clear(PORTB, BIT(1));
                    bit_clear(PORTB, BIT(2));
                }
            break;
        }
    }

}

#define MOTOR_DELAY_MS 1
#define CURTAIN_CALIBRATION_CONSTANT 200
#define STRETCHER_CALIBRATION_CONSTANT 50
```

```c

void UPDATE_DEVICE_VALUE_H() {
    const uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
    const uint8_t deviceValue = *((uint8_t*)parameter[1].startingPointer);

    switch (deviceIndex) {
        case 0:
            STRETCHER_POS_CHANGE_HANDLE(deviceValue);
        break;
        case 1:
            CURTAIN_POS_CHANGE_HANDLE(deviceValue);
        break;
        case 2:
            for (uint8_t x = 0; x < 6; x++)
            {
                bit_flip(PORTB, BIT(0));
                bit_flip(PORTB, BIT(1));
                bit_flip(PORTB, BIT(2));
                _delay_ms(200);
            }
            bit_clear(PORTB, BIT(0));
            bit_clear(PORTB, BIT(1));
            bit_clear(PORTB, BIT(2));
        break;
    }

    deviceStoredValue[deviceIndex] = deviceValue;

}

void GET_ALL_DEVICES_VALUE_H() {}

void GET_DEVICE_VALUE_H() {
    _delay_ms(100);
    uint8_t deviceIndex = *((uint8_t*)parameter[0].startingPointer);
    writeParameterValue(0, &deviceIndex, 1);
    writeParameterValue(1, &deviceStoredValue[deviceIndex], 2);
    ComposeMessageToBuffer(UPDATE_DEVICE_VALUE_ID, 2, 0x7C);

    nrf24_initRF_SAFE(MAIN_BOARD, TRANSMIT);
    nrf24_send(command_buffer);
    while(nrf24_isSending());
    uint8_t messageStatus = nrf24_lastMessageStatus();
}
void MESSAGE_STATUS_H() {}


uint8_t previousCurtainPosition = 0;
uint8_t previousStretcherPosition = 0;


void CURTAIN_POS_CHANGE_HANDLE(uint8_t positionToMove){
    bit_set(PORTB, BIT(1));
    bit_set(PORTB, BIT(2));


    if (positionToMove<8) {
        uint16_t degreesToMove = abs(positionToMove-previousCurtainPosition)*CURTAIN_CALIBRATION_CONSTANT;

        if((positionToMove-previousCurtainPosition)>0){
            for (uint16_t x = 0; x < degreesToMove;x++){
                PORTD = 0b00000011;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00000110;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00001100;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00001001;
                _delay_ms(MOTOR_DELAY_MS);
            }
            }else{
            for (uint16_t x = 0; x < degreesToMove;x++){
                PORTD = 0b00001100;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00000110;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00000011;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00001001;
                _delay_ms(MOTOR_DELAY_MS);
            }
        }

        PORTD = 0b00000000;
        previousCurtainPosition = positionToMove;
    }
    bit_clear(PORTB, BIT(1));
    bit_clear(PORTB, BIT(2));
}

void STRETCHER_POS_CHANGE_HANDLE(uint8_t positionToMove){
    bit_set(PORTB, BIT(1));
    bit_set(PORTB, BIT(2));

    if (positionToMove<4) {
        uint16_t degreesToMove = abs(positionToMove-previousStretcherPosition)*STRETCHER_CALIBRATION_CONSTANT;

        if((positionToMove-previousCurtainPosition)>0){
            for (uint16_t x = 0; x < degreesToMove;x++){
                PORTD = 0b00110000;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b01100000;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b11000000;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b10010000;
                _delay_ms(MOTOR_DELAY_MS);
            }
            }else{
            for (uint16_t x = 0; x < degreesToMove;x++){
                PORTD = 0b11000000;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b01100000;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b00110000;
                _delay_ms(MOTOR_DELAY_MS);
                PORTD = 0b10010000;
                _delay_ms(MOTOR_DELAY_MS);
            }
        }

        PORTD = 0b00000000;
        previousStretcherPosition = positionToMove;
    }
    bit_clear(PORTB, BIT(1));
    bit_clear(PORTB, BIT(2));
}
```