به نام خدا

درس سیستم‌های نهفته

پروژه

شایان صالحی

ایمان محمدی

آرمین ثقفیان

عنوان پروژه، پریز هوشمند (پروژه 20)

مقدمه

در عصر حاضر، جایی که تکنولوژی و اینترنت اشیاء (IoT) نقش بسزایی در بهبود کیفیت زندگی ما دارند، هوشمندسازی منزل و محیط کار به یک نیاز اساسی تبدیل شده است. پروژه‌ی «پریز هوشمند» با هدف ارتقاء امکانات خانه هوشمند و افزایش کارایی و انعطاف‌پذیری در کنترل دستگاه‌های الکتریکی طراحی و پیاده‌سازی شده است. این پروژه با استفاده از ماژول کوییکتل و یک رله، امکان کنترل از راه دور دستگاه‌های الکتریکی را از طریق وب یا اپلیکیشن موبایل فراهم می‌کند.

در این پروژه، ما به ساخت یک پریز هوشمند پرداخته‌ایم که قادر است به واسطه‌ی فناوری GSM و اتصال به اینترنت، دستورات کاربر را از هر جای دنیا دریافت کرده و عملیاتی نظیر روشن یا خاموش کردن و یا برنامه‌ریزی زمانی برای فعالیت دستگاه‌های متصل شده را اجرا کند. این امر با استفاده از قابلیت Open CPU موجود در ماژول کوییکتل محقق شده که اجازه می‌دهد بدون نیاز به میکروکنترلر یا پردازنده کمکی، برنامه‌های کاربردی مختلفی را بر روی خود ماژول اجرا کنیم.

قابلیت Open CPU این امکان را به ما می‌دهد که با نوشتن کد مستقیماً بر روی ماژول، ویژگی‌های پیچیده‌تری نظیر برقراری ارتباط با سرورهای MQTT برای دریافت و ارسال پیام، ذخیره‌سازی داده‌ها در حافظه خود قطعه برای ذخیره کاربران، و کنترل دقیق‌تر رله‌ها و سایر قطعات الکترونیکی را بدون واسطه و با دقت بالا انجام دهیم.

هدف

هدف از این پروژه، ایجاد یک پریز هوشمند قابل اعتماد و کاربردی است که کاربران بتوانند به صورت از راه دور و به آسانی دستگاه‌های برقی خود را مدیریت کنند. این شامل برنامه‌ریزی زمان‌های روشن و خاموش شدن پریز از طریق وب یا اپلیکیشن است.

قطعات استفاده شده

- ماژول *M*65 نصب شده بر روی بورد
- مبدل *USB* به *TTL*
- رله و سایر قطعات الکتریکی برای مکانیزم سوئیچینگ

توضیح فنی

این پروژه با استفاده از قابلیت GSM ماژول و ویژگی open CPU آن، برنامه‌ریزی شده است تا دستورات دریافتی از طریق پروتکل MQTT را پردازش کند و عملیاتی نظیر روشن یا خاموش کردن پریز را بر اساس برنامه‌های تعریف شده توسط کاربر اجرا کند..

پیاده‌سازی

در ابتدا برای پیاده‌سازی هرکد مورد نیاز روی این بخش، نیاز داریم که کدهای مورد نیاز را کامپایل کنیم. این کار توسط MS-DOS تعبیه شده در SDK مربوطه انجام می‌شود که مسیر target آن کامپایلر مربوط به آن است. پس از اجرای برنامه مدنظر ما در main.c در پوشه custom، یک فایل با فرمت APPGS5MDM32A01.lod ساخته شده که با لود کردن آن در نرم‌افزار QFlash با استفاده از پورت‌های دیباگ، قطعه را پروگرام می‌کنیم.

برای دریافت log های مربوط به قطعه از نرم‌افزار QNavigator بهره گرفتیم که در برنامه اصلی با دستور APP_DEBUG می‌توان پیام‌های مدنظر را از طریق UART و از طریق پورت‌های RXD و LXD انتقال دهیم.

بخش اول: تعریف و مقدمات

تعریف ماکروها و متغیرها
کد با تعریف ماکروها و متغیرها شروع می‌شود که شامل پورت‌های UART برای ارتباط و دیباگ، طول بافر برای ذخیره داده‌های دریافتی و ارسالی، و پین‌های مربوط به کنترل رله‌ها می‌شود. همچنین، ساختار داده‌ای برای نگهداری وضعیت خروجی‌ها (رله‌ها) تعریف می‌شود.

```
#define DEBUG_ENABLE 1
#if DEBUG_ENABLE > 0
#define DEBUG_PORT UART_PORT1
#define DBG_BUF_LEN 512
static char DBG_BUFFER[DBG_BUF_LEN];
#define APP_DEBUG(FORMAT, ...)
    {
        Ql_memset(DBG_BUFFER, 0, DBG_BUF_LEN);
        Ql_sprintf(DBG_BUFFER, FORMAT, ##__VA_ARGS__);
        if (UART_PORT2 == (DEBUG_PORT))
        {
            Ql_Debug_Trace(DBG_BUFFER);
        }
        else
        {
            Ql_UART_Write((Enum_SerialPort)(DEBUG_PORT), (u8 *)(DBG_BUFFER), Ql_strlen((const char *)(DBG_BUFFER)));
        }
    }
#else
#define APP_DEBUG(FORMAT, ...)
```

در این قسمت مانند تمامی example ها متغییرهای مربوط به لاگ انداختن، UART و دستور APP_DEBUG تعریف شده است که در طول پروگرم‌ما نقش اساسی برای نشان دادن استیت‌های مختلف قطعه دارد و همچنین این امکان را فراهم می‌کند تا پیام‌های دیباگ و داده‌های دریافتی از MQTT یا SMS را بخوانیم.

بخش دوم: تعریف متغیرهای اساسی

در این بخش تمامی متغیرهای اساسی همچون چراغ‌های قطعه، تایمرهای مورد نیاز برای پروتکل MQTT و چک کردن استیت آنها و یک متغیر TIMER_ID منحصر به فرد برای تمامی خروجی‌های که بتواند به صورت موازی و مستقل مکانیزم زمان‌دهی آن را هندل کند.

همچنین درنهایت در اینجا برای ایجاد پروتکل MQTT یک HOST NAME و یک پورت در نظر گرفته شده است که ما آنها را دامین test.mosquitto.org و پورت 1883 استفاده کرده‌ایم.

تصویر مربوط به این قسمت را در صفحه بعد می‌توانید مشاهده کنید.

```
#define CLOCK PINNAME_PCM_OUT
#define LATCH PINNAME_PCM_IN
#define DATA PINNAME_PCM_SYNC

#define OUT_1 PINNAME_DTR
#define OUT_2 PINNAME_CTS
#define OUT_3 PINNAME_RTS

#define LSBFIRST 0
#define MSBFIRST 1

#define MQTT_TIMER_ID         0x200
#define MQTT_TIMER_PERIOD     500

#define TIMER_ID_OUTPUT1 1001
#define TIMER_ID_OUTPUT2 1002
#define TIMER_ID_OUTPUT3 1003
#define TIMER_ID_OUTPUT4 1004
#define TIMER_ID_OUTPUT5 1005
#define TIMER_ID_OUTPUT6 1006

#define MQTT_TOPIC_CMD "module/outputs/cmd"
#define MQTT_TOPIC_STATUS "module/outputs/status"

#define HOST_NAME "test.mosquitto.org"      // Replace with your MQTT broker's address
#define HOST_PORT 1883              // Standard MQTT port, use 8883 for MQTT over TLS
```

بخش سوم: کار به صورت استیت برای پروتکل MQTT

برای پیاده‌سازی پروتکل MQTT یک تابع اصلی تحت عنوان Callback_Timer داریم که تمامی استیت های پروتکل MQTT را مدیریت می‌کند.

در ابتدا در استیت دریافت کوئری هستیم و پس از آن می‌بایست با توجه به تنظیمات و پارامترهای داده شده کانفیگ صورت گیرد. پس از آن یک MQTT با هاست و پورت داده شده باز شده و پس از آن متصل می‌شود.

بعد از انجام این‌کارها قطعه‌ما در تایپک از قبل تعیین شده "MIc Outputs" سابسکرایب کرده و پیام‌های رد و بدل شده تحت این موضوع را به عنوان دستور می‌پذیرد. پس از این استیت وارد استیت STATE_MQTT_PUB شده و یک پیام از قبل تعیین شده در این فضا publish می‌کند. در نهایت وارد استیت دریافت پیام و خواندن کامند از آن می‌شویم که با توجه به تایمر تعیین شده این تابع صدا زده شده که بررسی کند که آیا پیامی دریافت شده است یا نه.

این قسمت تعریف تمامی استیت‌ها بوده:

```
104  typedef enum{
105      STATE_NW_QUERY_STATE,
106      STATE_MQTT_CFG,
107      STATE_MQTT_OPEN,
108      STATE_MQTT_CONN,
109      STATE_MQTT_SUB,
110      STATE_MQTT_PUB,
111      STATE_MQTT_TUNS,
112      STATE_MQTT_CLOSE,
113      STATE_MQTT_DISC,
114      STATE_MQTT_TOTAL_NUM
115  }Enum_ONENETSTATE;
116  static u8 m_mqtt_state = STATE_NW_QUERY_STATE;
117
118  #define APN        "CMcom\0"
119  #define USERID     ""
120  #define PASSWD     ""
```

و این قسمت ساختار کلی تابع طراحی شده را نشان می‌دهد:

```
1309 static void Callback_Timer(u32 timerId, void* param)
1310 {
1311     s32 ret;
1312
1313     if(MQTT_TIMER_ID == timerId)
1314     {
1315         switch(m_mqtt_state)
1316         {
1317             case STATE_NW_QUERY_STATE:
1318             {…
1340             case STATE_MQTT_CFG:
1341             {…
1358             case STATE_MQTT_OPEN:
1359             {…
1374             case STATE_MQTT_CONN:
1375             {…
1391             case STATE_MQTT_SUB:
1392             {              …
1416             case STATE_MQTT_PUB:
1417             {
1418                 pub_message_id++;  //< The range is 0-65535. It will be 0 only when<qos>=0.
1419                 ret = RIL_MQTT_QMTPUB(connect_id,pub_message_id,QOS1_AT_LEASET_ONCE,0,test_topic,Ql_strlen(test_data
1420                 if(RIL_AT_SUCCESS == ret)…
1425                 else
1426                 {
1427                     APP_DEBUG("//<Publish a message to MQTT server failure,ret = %d\r\n",ret);
1428                 }
1429                 break;
```

که در اینجا شاهد این هستیم که خواندن این تابع به عنوان یک تایمر ست شده است:

```
1146    // mqtt_connect_and_subscribe();
1147    Ql_Timer_Register(MQTT_TIMER_ID, Callback_Timer, NULL);
1148    ret = Ql_Mqtt_Recv_Register(mqtt_message_received_callback);
```

بخش چهارم: تابع اصلی

در تابع اصلی که تحت عنوان proc_main_task می‌شناسیم در ابتدا سریال پورت‌ها initialize شده سپس تمام قسمت‌های اصلی دیگر همانند وضعیت سیم‌کارت، تایمرها، استک تایمر، چک‌کردن فایل برای بررسی کاربرهای دارای دسترسی و غیره انجام می‌شود.

پس از آن یک حلقه اصلی داریم که همانند زیر همواره پیام‌های دریافتی را بررسی می‌کند:

```
1170            switch (taskMsg.param1) {
1171
1172 >              case URC_SYS_INIT_STATE_IND: {…
1186
1187 >              case URC_SIM_CARD_STATE_IND: {…
1195
1196 >              case URC_GSM_NW_STATE_IND: {…
1200
1201 >              case URC_GPRS_NW_STATE_IND: {…
1205
1206 >              case URC_CFUN_STATE_IND: {…
1210
1211 >              case URC_COMING_CALL_IND: {…
1216
1217 >              case URC_NEW_SMS_IND: {…
1222
1223 >              case URC_MODULE_VOLTAGE_IND: {…
1227
1228 >              case URC_MQTT_OPEN: {…
1239
1240 >              case URC_MQTT_CONN: {…
1251
1252 >              case URC_MQTT_SUB: {…
1263
1264 >              case URC_MQTT_PUB: {…
1275
1276 >              case URC_MQTT_CLOSE: {…
```

در اینجا در ابتدا تمامی استیت‌های مختلف چک می‌شود که پیام دریافتی از طرف os قطعه چه حالتی دارد. در صورتی که مربوط به پیامک بوده وارد آن استیت‌های شده و در غیر این صورت قسمت‌های

MQTT را مدیریت می‌کند. این حلقه به ما اجازه می‌دهد که همزمان پیام‌های SMS و MQTT را دریافت و براساس دستورهای داده شده، action های مورد نظر را انجام دهد.

در صورتی که در حالت URC_NEW_SMS_IND باشیم دستورهای جدید از طریق پیامک دریافت شده و برای دریافت پیام‌های MQTT از تابع زیر استفاده کرده‌ایم:

```
1148        ret = Ql_Mqtt_Recv_Register(mqtt_message_received_callback);
```

بخش پنجم: خواندن فایل از روی حافظه تراشه

در اینجا فایل ذخیره شده برای شماره تمامی کاربرها خوانده شده و تمامی user های که برنامه حق دارد از آنها دستور بگیر را می‌توانیم در اینجا تعیین کنیم.

```
1075  void check_file() {
1076      handle = Ql_FS_Open(DATA_FILE_PATH, QL_FS_READ_ONLY);
1077      if (handle > 0) {
1078          Ql_FS_Seek(handle, 0, QL_FS_FILE_BEGIN);
1079          Ql_FS_Read(handle, file_content, LENGTH - 1, &readenLen);
1080          Ql_FS_Close(handle);
1081          Ql_strncpy(users[0], file_content, 13);
1082          Ql_strncpy(users[1], file_content + 13, 13);
1083          Ql_strncpy(users[2], file_content + 26, 13);
1084          Ql_strncpy(users[3], file_content + 39, 13);
1085          APP_DEBUG("Manager: %s\r\n First User: %s\r\n Second User : %s\r\n Third User: %s\r\n", users[0], users[1],
1086          APP_DEBUG("The content of the file:%s\r\n", file_content);
1087      } else {
1088          APP_DEBUG("Couldn't find the file \r\n");
1089      }
1090  }
```

همچنین با استفاده از دستور 0[ADD +98[Phone Number می‌توان کاربر جدید به این مجموعه اضافه نمود.

بخش ششم: تابع پارسر کامندها و انجام دستورات

در اینجا تابعی برای پارس کردن تمامی دستورات و انجام‌ها طراحی شده است که دستورها شامل این بخش‌ها می‌شود:

- *ALL ON -> Turn all on*
- *ALL OFF -> Turn all off*
- *ON OUT [number] -> On out [number] 1 < [number] < 6*
- *OFF OUT [number] -> OF out [number]*
- *Manager -> Make that phone number manager*

- *Get ALL Users -> All the users that have permission to command*
- *Add [Phone number] -> Add +989138094457*
- *Get Outputs -> Get the state of all outputs*
- *Current time -> Get the time of M65*
- *UNTIL [time] ON OUT [number] -> Until given time set output [number] state to on*
- *FOR [seconds] ON OUT [number] -> For amount of [seconds] keeps out [number] on*

که ۶ خروجی درنظر گرفته شده که ۳ تای از آنها مربوط به چراغ‌های LED خود قطعه و سه‌تای دیگر مربوط به خروجی‌های IO بوده است.

در اینجا می‌توانید نحوه بررسی تمامی کامندها را مشاهده کنید:

```
668     {
669         APP_DEBUG("sender = %s\r\n", pDeliverTextInfo->oa);
670         if (Ql_strncmp((pDeliverTextInfo->oa), users[0], 7) == 0 || Ql_strncmp((pDeliverTextInfo->oa), users[1], 7)
671             if (Ql_strncmp((pDeliverTextInfo->data), "Add user ", 9) == 0) {…
701
702             if (Ql_strcmp((pDeliverTextInfo->data), "Get All Users") == 0) {…
709
710             if (Ql_strcmp((pDeliverTextInfo->data), "Get Outputs") == 0) {…
717
718             if (Ql_strcmp((pDeliverTextInfo->data), "ALL OFF") == 0) {…
734
735             if (Ql_strcmp((pDeliverTextInfo->data), "ALL ON") == 0) {…
751
752             if (Ql_strcmp(pDeliverTextInfo->data, "Current Time") == 0) {…
763
764             if (Ql_strncmp((pDeliverTextInfo->data), "ON OUT ", 7) == 0) {…
799
800             if (Ql_strncmp(pDeliverTextInfo->data, "UNTIL ", 6) == 0) {…
863
864             if (Ql_strncmp(pDeliverTextInfo->data, "FOR ", 4) == 0) {…
922
923             if (Ql_strncmp((pDeliverTextInfo->data), "OFF OUT ", 8) == 0) {…
958         }
```

نکته حائز اهمیت در این قسمت این است که برای پابلیش کردن یک پیام در MQTT یا ارسال یک پیامک به همان شماره فرستنده از دو تابع زیر بهره می‌بریم:

```
257 void SMS_TextMode_Send(char *phone, char *msg) {
258     s32 iResult;
259     u32 nMsgRef;
260     ST_RIL_SMS_SendExt sExt;
261     Ql_memset(&sExt, 0x00, sizeof(sExt));
262     APP_DEBUG("< Send Normal Text SMS begin... >\r\n");
263     iResult = RIL_SMS_SendSMS_Text(phone, Ql_strlen(phone), LIB_SMS_CHARSET_GSM, msg, Ql_strlen(msg), &nMsgRef);
264     if (iResult != RIL_AT_SUCCESS) {
265         APP_DEBUG("< Fail to send Text SMS, iResult=%d, cause:%d >\r\n", iResult, Ql_RIL_AT_GetErrCode());
266         return;
267     }
268     APP_DEBUG("< Send Text SMS successfully, MsgRef:%u >\r\n", nMsgRef);
269 }
```

```
1057 void publish_mqtt_msg(const u8* mqtt_data) {
1058     s32 ret;
1059     pub_message_id += 50;  //< The range is 0-65535. It will be 0 only when<qos>=0.
1060     APP_DEBUG("Massage: %s\r\n", mqtt_data);
1061     ret = RIL_MQTT_QMTPUB(connect_id,pub_message_id,QOS1_AT_LEASET_ONCE,0,test_topic,Ql_strlen(mqtt_data),mqtt_data);
1062     APP_DEBUG("After Massage: %s\r\n", mqtt_data);
1063     if(RIL_AT_SUCCESS == ret) {
1064         APP_DEBUG("//<Start publish a message to MQTT server\r\n");
1065     } else {
1066         APP_DEBUG("//<Publish a message to MQTT server failure,ret = %d\r\n",ret);
1067     }
1068 }
```

و تابع زیر را پس از بررسی تمامی دستورها صدا می‌زنیم که استیت تمامی خروجی‌های را آپدیت کند:

```
1437 void update_IO() {
1438     Ql_GPIO_SetLevel(OUT_1, Data.set.out_1 ? PINLEVEL_HIGH : PINLEVEL_LOW);
1439     Ql_GPIO_SetLevel(OUT_2, Data.set.out_2 ? PINLEVEL_HIGH : PINLEVEL_LOW);
1440     Ql_GPIO_SetLevel(OUT_3, Data.set.out_3 ? PINLEVEL_HIGH : PINLEVEL_LOW);
1441     Ql_GPIO_SetLevel(PINNAME_PCM_CLK, Data.set.out_4 ? PINLEVEL_HIGH : PINLEVEL_LOW);
1442     Ql_GPIO_SetLevel(PINNAME_PCM_SYNC, Data.set.out_5 ? PINLEVEL_HIGH : PINLEVEL_LOW);
1443     Ql_GPIO_SetLevel(PINNAME_PCM_IN, Data.set.out_6 ? PINLEVEL_HIGH : PINLEVEL_LOW);
1444 }
```

بخش هفتم: بخش تایمر

در اینجا دو مکانیزم تایمر یکی تحت عنوان duration و دیگری تحت عنوان stack پیاده‌سازی شده است. برای روش زمانی از کامند FOR [second]s طراحی شده و برای حالت استک از دستور UNTIL [HH:mm:ss] ON استفاده کرده‌ایم.

بخش زیر برای دستور FOR طراحی شده است:

```
859             if (Ql_strncmp(pDeliverTextInfo->data, "FOR ", 4) == 0) {
860                 char timeStr[10] = {0};
861                 Ql_strncpy(timeStr, pDeliverTextInfo->data + 4, 3); // Extract time string
862                 timeStr[3] = '\0';
863                 APP_DEBUG("ON FOR = %d\r\n", pDeliverTextInfo->data[15]);
864                 if (pDeliverTextInfo->data[15] > 47 && pDeliverTextInfo->data[15] < 58) {
865                         switch (pDeliverTextInfo->data[15] - 48) {
866                         case 1:
867                             timerDuration1 = (u32)Ql_atoi(timeStr);
868                             APP_DEBUG("Time duration: %d\r\n", timerDuration1);
869                             Data.set.out_1 = 1;
870                             update_IO();
871                             Ql_Timer_Start(TIMER_ID_OUTPUT1, timerDuration1 * 1000, FALSE); // Start timer
872                             break;
```

که در آن زمان مدنظر (به صورت ۳ رقم) از کامند برداشته شده و یک تایمر بر اساس آن تنظیم می‌شود.

همچنین برای بخش UNTIL به این صورت داریم:

```
795             if (Ql_strncmp(pDeliverTextInfo->data, "UNTIL ", 6) == 0) {
796                 char timeStr[10] = {0};
797                 Ql_strncpy(timeStr, pDeliverTextInfo->data + 6, 8); // Extract time string
798                 targetTimestamp = TimeStringToTimestamp(timeStr);
799                 currentTimestamp = ConvertToTimestamp(Ql_GetLocalTime(&currentTime));
800                 APP_DEBUG("ON UNTIL = %d\r\n", pDeliverTextInfo->data[22]);
801                 if (pDeliverTextInfo->data[22] > 47 && pDeliverTextInfo->data[22] < 58) {
802                     APP_DEBUG("Target time: %d\r\n", targetTimestamp);
803                     APP_DEBUG("Current time: %d\r\n", currentTimestamp);
804                     if (targetTimestamp > currentTimestamp) {
805                         switch (pDeliverTextInfo->data[22] - 48) {
806                         case 1:
807                             timerDuration1 = targetTimestamp - currentTimestamp;
808                             APP_DEBUG("Time duration: %d\r\n", timerDuration1);
809                             Data.set.out_1 = 1;
810                             update_IO();
811                             Ql_Timer_Start(TIMER_ID_OUTPUT1, timerDuration1 * 1000, FALSE); // Start timer
812                             break;
```

که در آن زمان داده شده در دستور براساس فرمت HH:mm:ss دریافت کرده و براساس تابعی که در ادامه می‌توان دید به فرمتی درمی‌آید که به ثانیه بوده و با کم کردن از زمان فعلی قطعه می‌توان تعیین کنیم که یک خروجی تا چه زمانی روشن بماند.

```
u32 ConvertToTimestamp(const ST_Time *time) {
    static const int monthDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    long days = time->year * 365L + (time->year / 4) - (time->year / 100) + (time->year / 400);
    for (int i = 0; i < time->month - 1; i++) {
        days += monthDays[i];
    }
    if (time->month > 2 && (time->year % 4 == 0 && (time->year % 100 != 0 || time->year % 400 == 0))) {
        days += 1;
    }
    days += time->day - 1;
    long seconds = days * 86400L + time->hour * 3600 + time->minute * 60 + time->second;
    return (u32)seconds;
}

u32 TimeStringToTimestamp(const char* timeStr) {
    int hh, mm, ss;
    Ql_GetLocalTime(&currentTime);
    Ql_sscanf(timeStr, "%d:%d:%d", &hh, &mm, &ss);
    ST_Time targetTime = currentTime;
    targetTime.hour = hh;
    targetTime.minute = mm;
    targetTime.second = ss;
    return ConvertToTimestamp(&targetTime);
}
```

همچنین تابع زیر زمانی صدا زده می‌شود که تایمر به انتها می‌رسد:

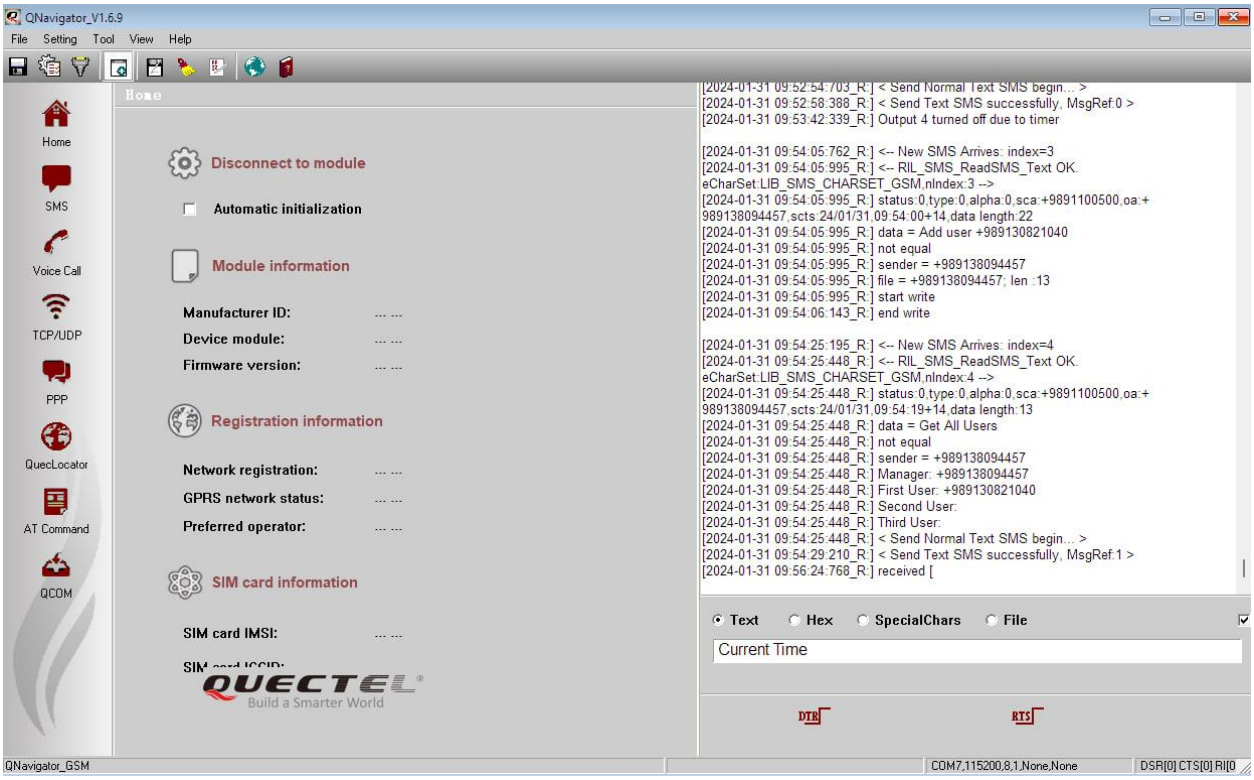
```
void TimerCallback2(u32 timerId, void* param) {
    if (timerId == TIMER_ID_OUTPUT2) {
        Data.set.out_2 = 0;  // Turn off output 2
        update_IO();
        APP_DEBUG("Output 2 turned off due to timer\r\n");
    }
}
```

که به ازای هر خروجی یک تابع منحصر به فرد تعریف شده است و پس از بررسی شرط اصلی آن خروجی را دوباره به حالت خاموش می‌برد.

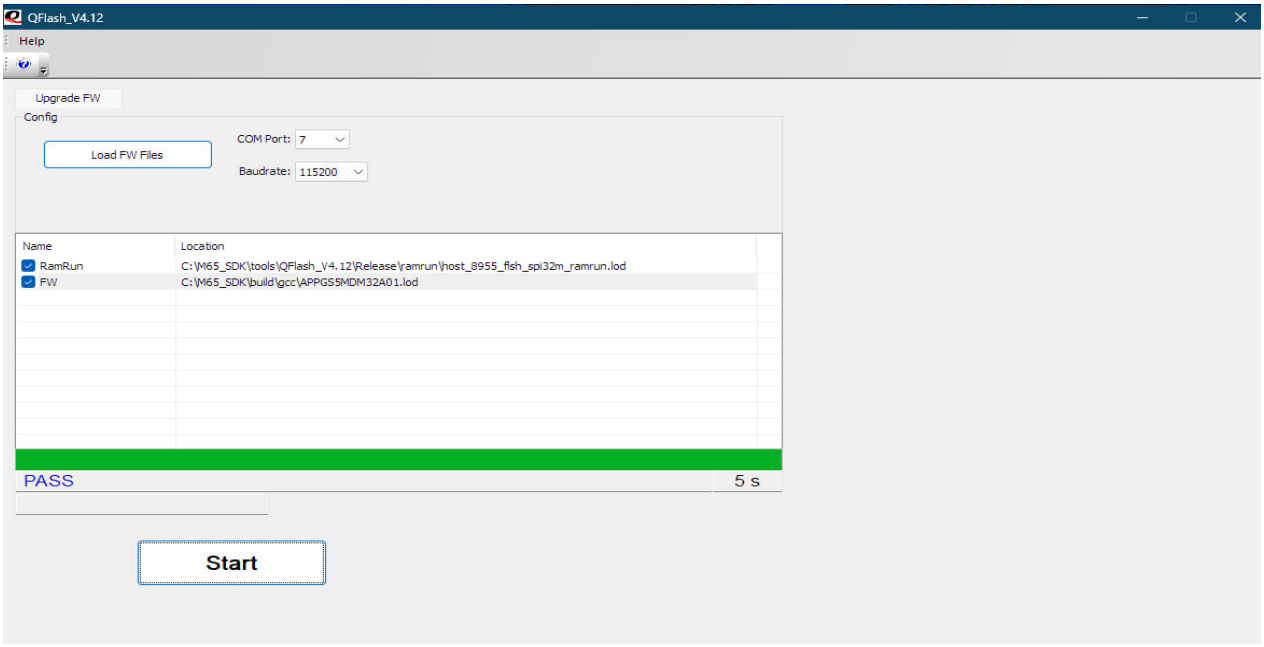
این دو دستور به ما این امکان را می‌دهد که به صورت موازی و مستقل مدت زمان روشن شدن تمامی خروجی‌ها را کنترل کنیم.

بخش هشتم: نرم‌افزارهای استفاده شده در طول انجام پروژه

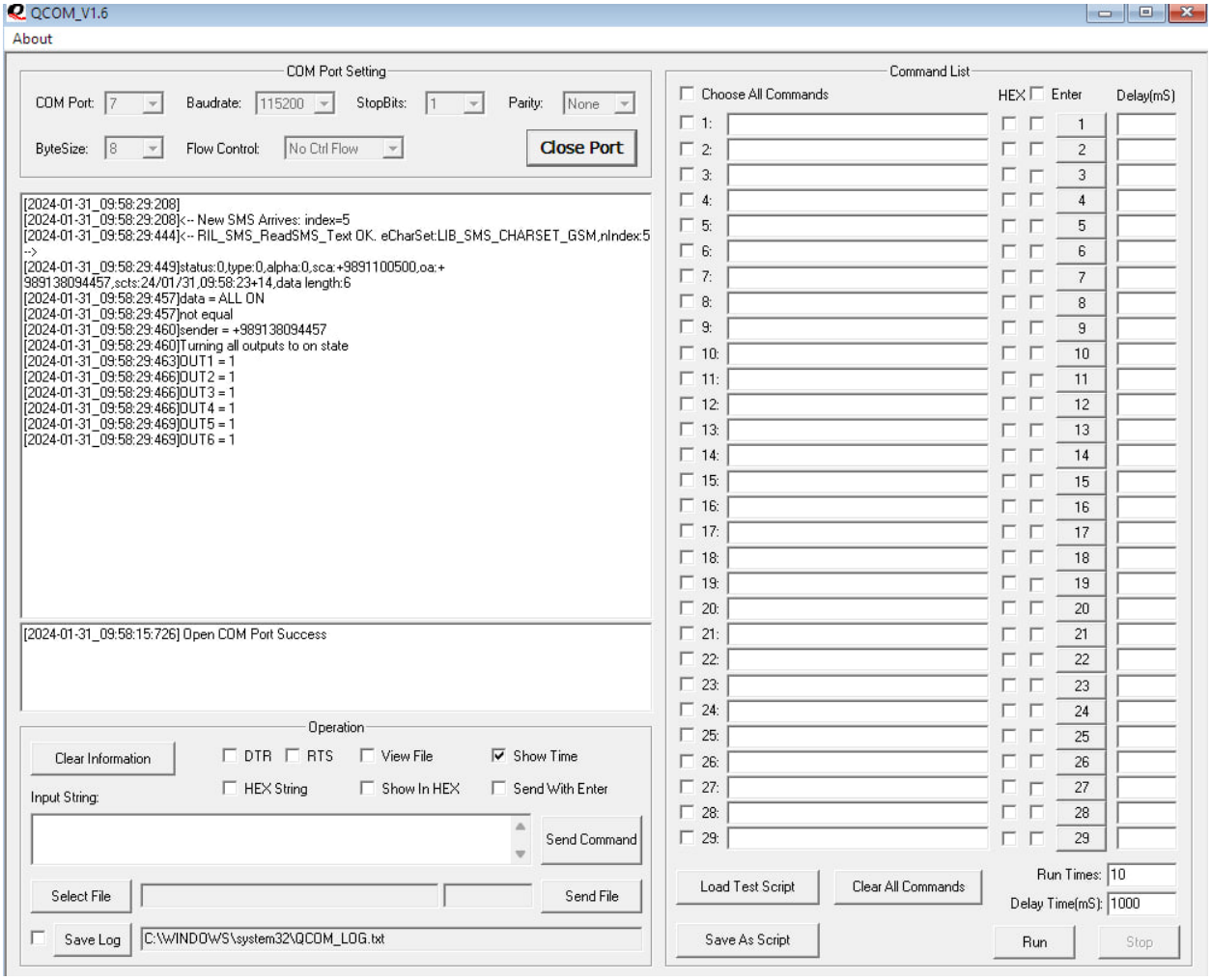
نرم‌افزار :QNavigator



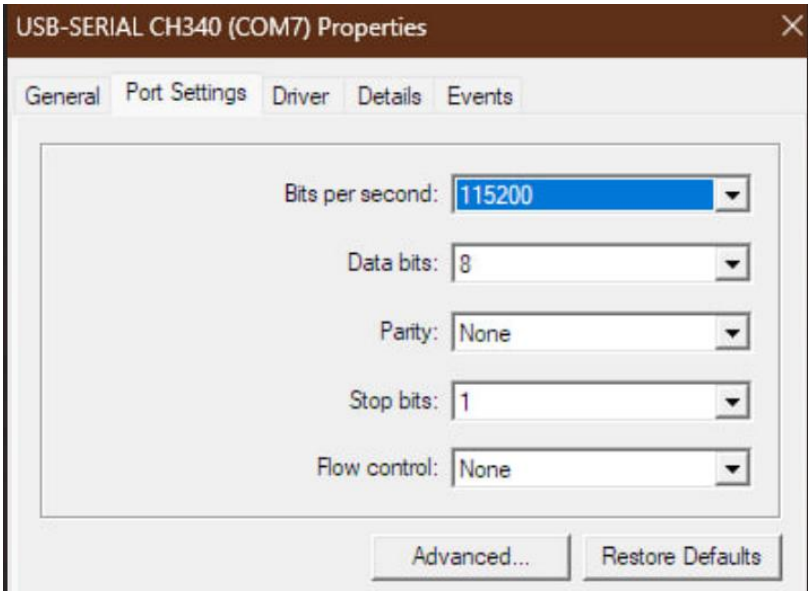
نرم‌افزار QFlash:



نرم‌افزار QCOM:



تنظیمات سریال مربوط به قسمت UART:



خلاصه:

در این پروژه که هدف اصلی آن کار با ماژول 65M شرکت کوییکتل بود که توانستیم با استفاده از SMS و پروتکل MQTT دستورها را به ماژول داده و از راه دور بتوانیم یک خروجی یک پریز را به همراه زمان‌بندی آن کنترل کنیم. چالش‌های اصلی این پروژه آشنایی و یادگیری با نحوه پروگرام کردن ماژول و استفاده از تمامی example های موجود در SDK مربوطه به قطعه، برای طراحی برنامه‌ای که بتواند به صورت همزمان از دستورهای پیامکی و MQTT پشتیبانی کنید و همچنین قابلیت تایمر و استفاده از حافظه خود ماژول را داشته باشد.

همچنین با بکارگیری نرم‌افزارهای همچون QCOM ،QNavigator توانستیم تمامی لاگ‌های مربوط به ماژول را بررسی کرده و در طی تکمیل کردن کد پروگرام آن، تمامی حالت‌های دارای اشکال را رفع کرده و برنامه‌ای بی‌نقص برای این ماژول ارائه دهیم.