上海宏英智能科技股份有限公司
Shanghai Smart Control Co., Ltd.

10.4” Display
User Manual

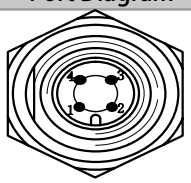
Contents

1 Technical Parameters

Kernel	Processor	Cortex-A9 Dual-Core	
	Frequency	1GHz	
	RAM	1GB	
	FLASH	4GB	
Software	Operating System	Linux	
	Development Platform	CODESYS V3.5	
Display	Size	10.4″	
	Resolution	1024*768	
	Brightness	600cd/m^2	
HMI	Function Keys	Front panel standard 8	
	Touch Screen	Resistance	
Environmental Parameter	Operating Temperature	-30℃ ~ +70℃	
	Storage Temperature	-40℃ ~ +80℃	
	Degree of Protection	IP65	
Electrical Parameter	Operating Voltage	12 ～ 32V DC	
	Power Consumption	530mA@24VDC	
Functional Parameters	CAN	CAN	2
	Ethernet	10/100 Base-T	1
	Video	720P AHD	4
	Audio	MIC Input, HP Output	1
	WIFI	AP/STA Mode	1
	Bluetooth	Voice Call, Music Play	1
EMC	Electrical transient conducted immunity along the power line	GB/T 21437.2-2021 (ISO 7637-2:2011)	
	Electrical transient immunity of non-power line coupling	GB/T 21437.3-2021 (ISO 7637-3:2016)	
	ESD	GB/T 19951-2019（ISO 10605：2008）	
	EFT	GB/T 17626.4-2018（IEC 61000-4-4:2012）	
Mechanical Stresses	Random Vibration	GB/T 2423.56-2018 (IEC 60068-2-64:2008)	
	Mechanical Shock	GB/T 2423.5-2019 (IEC 60068-2-27-2008)	
	Sine Vibration	GB/T 2423.10-2019 (IEC 60068-2-6:2007)	
Environmental Stresses	Triple Combination (vibration, temperature, humidity)	GB/T 2423.35-2019(IEC 60068-2-53-2010)	
	Alternating Humid Heat	GB/T 2423.4-2008 （IEC 60068-2-30：2005）	
	Temperature Shock	GB/T 2423.22-2012 (IEC 60068-2-14:2009)	
	Enclosure Protection Grade	GB/T 4208-2017（IEC 60529:2013）	
	Resistant to salt spray corrosion and penetration	GB/T 2423.17-2008 （IEC 60068-2-11:1981）	

2 Interface Definition

Power Supply

Port	Function	Note	Port Diagram
1	24VDC	Positive	
2	24VDC	Positive	
3	GND	Negative	
4	GND	Negative	

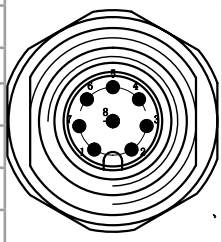
CAN1 Port - Audio

Port	Function	Note	Port Diagram
1	Audio-IN	Audio Input	
2	Audio-OUT	Audio Output	
3	Audio-GND	Audio Signal Ground	
4	CAN_H1	CAN1_H	
5	CAN_L1	CAN1_L	

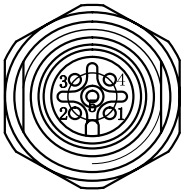
CAN2 Port

Port	Function	Note	Port Diagram
1	12V	12V Power Supply	
2	/	/	
3	/	/	
4	CAN_H2	CAN2_H	
5	CAN_L2	CAN2_L	

DOWNLOAD

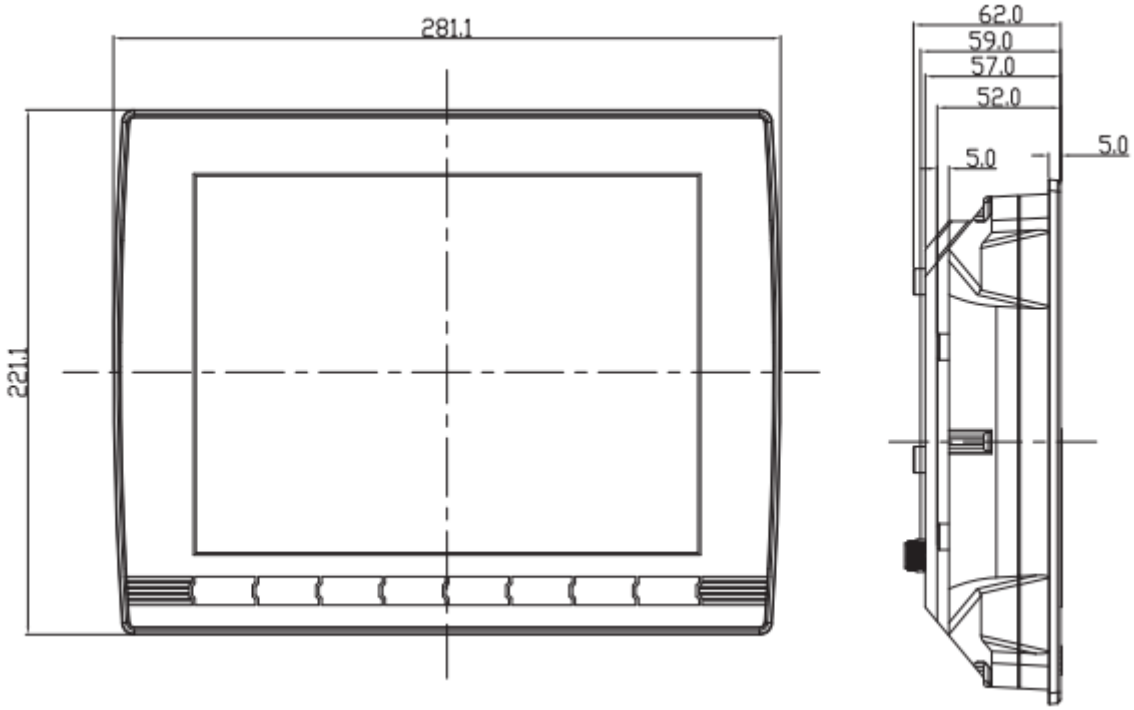
Port	Function	Note	Port Diagram
1	RX-	Ethernet	
2	TX-	Ethernet	
3	TX+	Ethernet	
4	RX+	Ethernet	
5	+5V	USB	
6	D-	USB	
7	D+	USB	
8	GND	USB	

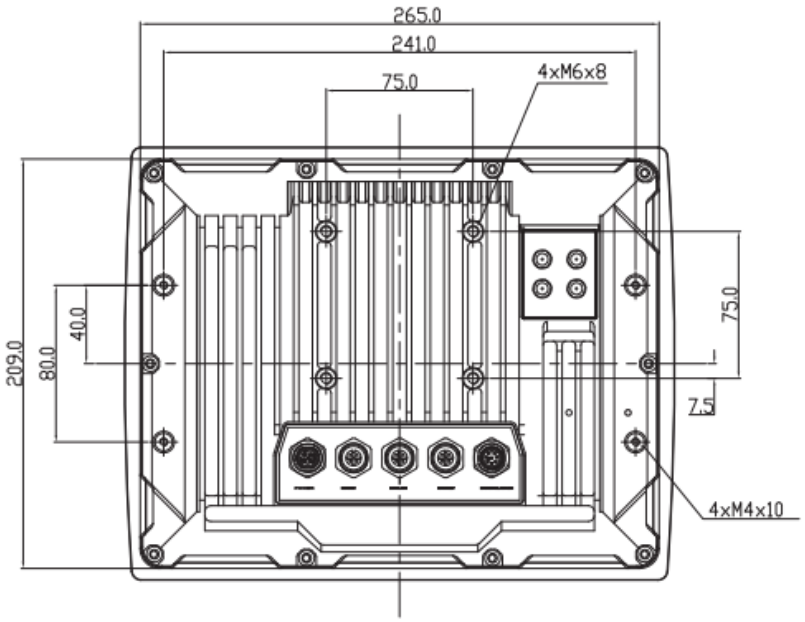
MULT1 Video

Port	Function	Note	Port Diagram
1	VIDEO 1	Video	
2	VIDEO 2	Video	
3	VIDEO 3	Video	
4	VIDEO 4	Video	
5	GND	Video Power Ground	

Notes:
Video power ground for camera power supply. Audio signal ground for audio signal use, not as audio power ground.

3 Installation Dimensions





6 Attentions

1）When the environment where the display is located requires soldering, all display connection cables need to be physically disconnected before such work is performed, otherwise the display will be burned.

2）The display should only be operated by trained and qualified personnel. Smart Control cannot guarantee the validity of its warranty if this product is damaged due to improper operation by non-professional personnel.

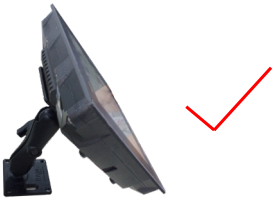
3）The display of the car display terminal will be easily scratched or seriously damaged if it is rubbed with sharp or hard objects. In order to ensure the service life of the display, this situation should be avoided.

4）The display of the monitor can be easily damaged in high temperatures, so please do not cover it with other objects, such as jackets or other clothing.

5）The display should preferably be installed under a cover to avoid exposure to liquids. For safety reasons, the vehicle-mounted display terminal should preferably be mounted on a stable object and should be prevented from moving, otherwise it is susceptible to traffic accidents that could damage the device or injure the operator.

6）Installation of the monitor: It is recommended to tilt a certain angle to install, using bracket installation or embedded mounting plate, as shown in the figure below.

Horizontal installation is not recommended, because horizontal installation will cause visual inconvenience, and once there is rain or liquid attached to the surface of the LCD screen, it can not flow away in time, affecting the touch. Please do not press the mounting plate in the monitor touch screen bezel, once pressed to the critical touch area, it will cause touch drift or failure.



7） In the process of programming with Codesys 3.5, note that the name of the image called must be English or numeric.

8）The display does not support mouse, keyboard and other peripherals, do not connect the mouse, keyboard and other peripherals when the power is on, there is a chance that the display will cause abnormalities.

9）If using the "plclogic" folder to update the application, please make sure that the copy of the file is complete and not damaged, and that the new version of the program file uses unofficial library files to ensure that the same version as the original display, otherwise the display may be stuck at the boot logo and cannot enter the system.

5 Connector Accessories Selection List

No.	Pic	Name	Model	Note	Qty
1		Binder Connector (4-core female)	9904301404	Power Connector	1
2		Binder Connector (5-core male)	9904371405	CAN Connector	2
3		Binder Connector (5-core male)	9904371405	Video Connector	1
4		Binder Connector (8-core female)	9904861208	Network/Audio connector	1

Appendix I Demo Description

（1） Bluetooth

Related Functions：

SysRestoreRetains(stFileName);

HY_BT.Sys_BT_Open(port,protocol);

HY_RADIO.Sys_Radio_Close(reserved);

HY_BT.Sys_BT_Close(reserved);

HY_BT.Sys_BT_GetCmdResponse(cmd,resp);

HY_BT.Sys_BT_ExecCommandByte(pcmdbyte,len);

Concat(STR1,STR2);

LEN(STR);

DELETE(STR,LEN,POS);

SysMemSet(dwDest,bCharacter,dwCount);

Related Libraries：HY_BT 1.0.0.4

HY_Radio 1.0.0.2

Standard 3.5.14.0

SysMem23 3.5.13.0

SysPlcCtrl23 3.5.13.0

Variable Declaration:

F_TRIG_Back	:F_TRIG;
F_TRIG_BackMain	:F_TRIG;
BackButton	:BOOL;
BackMainButton	:bool;
F_TRIG_OpenBT	:F_TRIG;
OpenBTButton	:BOOL;
FM_Result	:BOOL;
OpenBT	:BOOL;
BTOpenResult	:BOOL;
bt_uart_port	:BYTE := 4;
Reqcmd	:BYTE;
req_ret	:BOOL;
BT_Name	:STRING[40];
Response	: HY_BT.BTCmdResponse;
ShowBTID	:INT;
BTCloseResult	:BOOL;

```
BT_ConnectState          :BOOL;
bt_status                :INT;
BT_CallInFlag            :BOOL;
BT_CallOutFlag           :BOOL;
callin_tel               :HY_BT.BTCmdResponse;
callout_tel              :HY_BT.BTCmdResponse;
Telephone_Callin         :STRING[64];
F_TRIG_GetCall           :F_TRIG;
GetCallButton            :BOOL;
CallAct                  :BOOL;
cmd_buf                  : ARRAY [0..255] OF BYTE;
send_ret                 :INT;
DelayShow                :INT;
ShowPoint                :INT;
BLINK_TIME               :BLINK;
CallTimeSencond          :INT;
CallTimeMinte            :INT;
BLINK_RTIRG              :R_TRIG;
F_TRIG_NotCall           :F_TRIG;
NotCallButton            :BOOL;
TelephoneNum             :STRING[40];
F_TRIG_Num               :ARRAY [0..11] OF F_TRIG;
KeyPress                 :ARRAY [0..11] OF BOOL;
F_TRIG_CallOut           :F_TRIG;
F_TRIG_NotCallOut        :F_TRIG;
CallOutButton            :BOOL;
NotCallOutButton         :BOOL;
tel_number               :ARRAY [1..32] OF BYTE;
tel_len                  :INT;
Pos                      :INT;
I                        :INT;
F_TRIG_OutCall           :F_TRIG;
OutCallButton            :BOOL;
music_status             :INT;
music_a2dp_connect_flag: BOOL;
music_name_wstring       :WSTRING(64);
music_name               : HY_BT.BTCmdResponse;
music_lyric_wdata        :WSTRING(64);
music_name_string        :STRING[64];
```

```
music_name_len          :INT;
music_name_byte         :ARRAY [1..64] OF BYTE;
music_times_ms          :DWORD;
music_total_time        :HY_BT.BTCmdResponse;
music_step              :HY_BT.BTCmdResponse;
music_time_step_ms      :DWORD;
TimeSecond              :INT;
DisplayTotalTime        :STRING[20];
F_TRIG_StartMusic       :F_TRIG;
OpenMusicButton         :BOOL;
MusicOpen               :BOOL;
DisplayNowTime          :STRING[20];
MoveState               :REAL;
music_lyric             :HY_BT.BTCmdResponse;
music_lyric_string      :STRING;
music_lyric_len         :INT;
music_lyric_byte        :ARRAY [1..64] OF BYTE;
TelePhone_CallOut       :STRING[20];
BT_Volume_Old           :BYTE;
R_TRIG_InitBT           :R_TRIG;
Init_BT                 :BOOL;
Test                    :INT;
R_TRIG_Start            :R_TRIG;
BT_CallingFlag          :BOOL;
DisplayVolume           :BYTE;
DisplayCallTime         :STRING[20];
F_TRIG_NextSong         :F_TRIG;
F_TRIG_LastSong         :F_TRIG;
F_TRIG_VolumeUp         :F_TRIG;
F_TRIG_VolumeDown       :F_TRIG;
DelayShowCount          :INT;
ShowID                  :INT;
VAR_GLOBAL              RETAIN
        BT_Volume:BYTE;     // Bluetooth Sound
END_VAR
VAR_GLOBAL
        During_Day:BOOL;    // Bluetooth Interface
        NextSongButton:BOOL;
        LastSongButton:BOOL;
```

```
        BTVolumeUpButton:BOOL;
        BTVolumeDNButton:BOOL;
END_VAR
```

Code Implementation：

```
R_TRIG_InitBT(CLK:=VisuElems.CurrentVisu='BT_PAGE');          // Jump to the Bluetooth Interface
F_TRIG_Back(CLK:=BackButton);                                 // Jump to the ENTMAIN Interface
F_TRIG_BackMain(CLK:=BackMainButton);                         // Jump to the Mainpage Interface
F_TRIG_OpenBT(CLK:=OpenBTButton);                             // Turn on Bluetooth
F_TRIG_GetCall(CLK:=GetCallButton);                           // Answer the Phone Button
F_TRIG_CallOut(CLK:=CallOutButton);                           // Dialing out Calls
F_TRIG_NotCallOut(CLK:=NotCallOutButton);                     // Backspace to delete 1 entered number
F_TRIG_OutCall(CLK:=OutCallButton);                           // Hang up the Phone
F_TRIG_StartMusic(CLK:=OpenMusicButton);                      // Play, Pause Music
F_TRIG_NextSong(CLK:=NextSongButton);                         // Next Song
F_TRIG_LastSong(CLK:=LastSongButton);                         // Previous Song
F_TRIG_VolumeUp(CLK:=BTVolumeUpButton);                       // Volume+
F_TRIG_VolumeDown(CLK:=BTVolumeDNButton);                     // Volume-
(*Bluetooth Initialisation*)
IF      R_TRIG_InitBT.Q  THEN
        SysRestoreRetains(FileName);
        BT_Volume_Old:=BT_Volume;                             // Get the last set volume
        ShowBTID:=1;                                          // "Turn on Bluetooth" is displayed after initialisation
END_IF

IF      VisuElems.CurrentVisu='BT_PAGE'  THEN
        IF      F_TRIG_OpenBT.Q         THEN                  // Turn on Bluetooth
                BTOpenResult:=FALSE;
                BTCloseResult:=FALSE;
                IF      NOT     FM_Result       THEN
                        FM_Result:=HY_RADIO.Sys_Radio_Close(0);// Turn on Bluetooth and Turn off the Radio
                END_IF

                OpenBT:=NOT     OpenBT;

                IF      OpenBT THEN                           // Turn on Bluetooth
                        BTOpenResult:=HY_BT.Sys_BT_Open(bt_uart_port, 16#80);
                ELSE
                        BTCloseResult:= HY_BT.Sys_BT_Close(0);// Turn off Bluetooth
                END_IF
```

```
        END_IF

        IF      BTCloseResult    THEN
                BTOpenResult:=FALSE;                    // Turn off Bluetooth, clear open function return value
        END_IF

        IF      BTOpenResult     THEN
                ShowBTID:=0;                            // Turn on Bluetooth, display "Turn off Bluetooth"
        END_IF

        IF      BTCloseResult    THEN
                ShowBTID:=1;                            // Turn off Bluetooth, display "Turn on Bluetooth"
        END_IF

        IF      BTOpenResult     THEN                   // Bluetooth is switched on
                reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_BT_CONNECT_
STATUS);
                req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(response));// Get Bluetooth connection status
                IF req_ret THEN
                        BT_ConnectState := TO_BOOL(TO_INT(response.data));      // Connection status
                END_IF
                reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_BT_NAME);// Get
Bluetooth Name
                req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(response));
                IF req_ret THEN
                        BT_Name := response.data;       // Bluetooth Name
                END_IF
        ELSE
                BT_ConnectState:=FALSE;
        END_IF



IF      BT_ConnectState THEN                            // Bluetooth is connected
        IF      NOT     Init_BT  THEN
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_REQUEST_SET_SYSTEM_
VOLUME);// System Volume
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_REQUEST_SET_SYSTEM_
VOLUME);// Bluetooth Volume
```

```
            cmd_buf[2] := BT_Volume;          // Bluetooth Volume
            cmd_buf[3] := 16#D;
            cmd_buf[4] := 16#A;
            send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 5);
            Init_BT:=TRUE;
        END_IF

        // Get Bluetooth phone status
        reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_CALL_STATUS);
        req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(response));
        bt_status := TO_INT(response.data);
        BT_CallInFlag := bt_status.1 ;                    // Incoming calls
        BT_CallOutFlag := bt_status.2;                    // O utgoing calls
        BT_CallingFlag := bt_status.0;



        IF      BT_CallInFlag     THEN                    // Incoming calls
            reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_CALLIN_
TELEPHONE);
            req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(callin_tel));
            IF req_ret THEN
                Telephone_Callin := callin_tel.data;      // Get the incoming phone number
            END_IF


            IF      F_TRIG_GetCall.Q          THEN        // Answering the phone
                CallAct:=TRUE;
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_CALL);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_CALL_MODE.ENUM_BT_CALL_ANSWER);
                cmd_buf[2] := 16#D;
                cmd_buf[3] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
            END_IF

        IF      CallAct   THEN                            // Answer
            IF      F_TRIG_OutCall.Q          THEN      // Hang up
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_CALL);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_CALL_MODE.ENUM_BT_CALL_HANG_UP);
```

```
                cmd_buf[2] := 16#D;
                cmd_buf[3] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
                CallAct:=FALSE;
                OutCallButton:=FALSE;
        END_IF
    ELSE
        IF      F_TRIG_OutCall.Q        THEN            // Refusal to answer
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_CALL);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_CALL_MODE.ENUM_BT_CALL_REFUSE);
                cmd_buf[2] := 16#D;
                cmd_buf[3] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
                CallAct:=FALSE;
                OutCallButton:=FALSE;
        END_IF
    END_IF

    BLINK_TIME(ENABLE:=CallAct,TIMEHIGH:=T#500MS,TIMELOW:=T#500MS);// Call Duration
    BLINK_RTIRG(CLK:=BLINK_TIME.OUT);

    IF      BLINK_RTIRG.Q   THEN                            // Call Duration
            CallTimeSencond:=CallTimeSencond+1;             // Second
            IF      CallTimeSencond>59      THEN
            CallTimeMinte:=CallTimeMinte+1;                 // Minute
            CallTimeSencond:=0;
    END_IF
END_IF

    IF      CallAct   THEN                                  // Connected - call duration display
            DisplayCallTime:='';
    IF      CallTimeMinte<10        THEN
            DisplayCallTime:=concat('0',TO_STRING(CallTimeMinte));
            DisplayCallTime:=concat(DisplayCallTime,':');
            ELSE
            DisplayCallTime:=concat(TO_STRING(CallTimeMinte),':');
    END_IF

    IF      CallTimeSencond<10      THEN
```

```
                DisplayCallTime:=concat(DisplayCallTime,'0');
                DisplayCallTime:=concat(DisplayCallTime,TO_STRING(CallTimeSencond));
                ELSE
                DisplayCallTime:=concat(DisplayCallTime,TO_STRING(CallTimeSencond));
        END_IF
END_IF

        IF      NOT     CallAct   THEN                          // Call not answered - ringing dots
                CallTimeSencond:=0;
                CallTimeMinte:=0;
                DelayShow:=DelayShow+1;
                IF      DelayShow>50    THEN
                        ShowPoint:=ShowPoint+1;
                        DelayShow:=0;
                END_IF

                IF      ShowPoint>3     THEN
                        ShowPoint:=0;
                END_IF
        ELSE
                        ShowPoint:=0;
                        DelayShow:=0;
                        END_IF
        ELSE
                CallAct:=FALSE;
        END_IF


        IF      BT_CallOutFlag THEN                                     // Have a call out
                reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_CALLOUT_
TELEPHONE);
                req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(callout_tel));
        IF      req_ret THEN
                TelePhone_CallOut := callout_tel.data;          // Dial outgoing phone number display
        END_IF

        IF      F_TRIG_OutCall.Q        THEN                    // Hang up the phone
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_CALL);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_CALL_MODE.ENUM_BT_CALL_HANG_UP);
```

```
        cmd_buf[2] := 16#D;
        cmd_buf[3] := 16#A;
        send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
END_IF

IF      NOT     BT_CallingFlag    THEN                    // Dialing out and waiting for a call to be answered
        CallTimeSencond:=0;
        CallTimeMinte:=0;
        DelayShow:=DelayShow+1;
        IF      DelayShow>50    THEN
        ShowPoint:=ShowPoint+1;
        DelayShow:=0;
        END_IF

        IF      ShowPoint>3     THEN
                ShowPoint:=0;
        END_IF
ELSE
        ShowPoint:=0;
        DelayShow:=0;
        END_IF

BLINK_TIME(ENABLE:=BT_CallingFlag,TIMEHIGH:=T#500MS,TIMELOW:=T#500MS);
BLINK_RTIRG(CLK:=BLINK_TIME.OUT);

IF      BLINK_RTIRG.Q    THEN                             // Dial out connection time
        CallTimeSencond:=CallTimeSencond+1;
        IF      CallTimeSencond>59      THEN
                CallTimeMinte:=CallTimeMinte+1;
                CallTimeSencond:=0;
        END_IF
END_IF

        DisplayCallTime:='';
        IF      CallTimeMinte<10        THEN
                DisplayCallTime:=concat('0',TO_STRING(CallTimeMinte));
                DisplayCallTime:=concat(DisplayCallTime,':');
        ELSE
                DisplayCallTime:=concat(TO_STRING(CallTimeMinte),':');
```

```
            END_IF

            IF      CallTimeSencond<10      THEN
                    DisplayCallTime:=concat(DisplayCallTime,'0');
                    DisplayCallTime:=concat(DisplayCallTime,TO_STRING(CallTimeSencond));
            ELSE
                    DisplayCallTime:=concat(DisplayCallTime,TO_STRING(CallTimeSencond));
            END_IF
        END_IF


            IF      NOT     BT_CallInFlag   AND     NOT     BT_CallOutFlag  THEN            // Dialing
in the absence of a telephone
                    TelephoneNumString();                                   // Phone Number

            IF      F_TRIG_CallOut.Q        AND     TelephoneNum<>''THEN // Making a phone call
                    SysMemSet(ADR(tel_number), 0, 16);
                    tel_len := LEN(TelephoneNum);
                    pos := 0;
                    FOR i:=1 TO tel_len BY 1 DO
                            pos := pos + 1;
                            tel_number[i] := WORD_TO_BYTE(BYTE_TO_HEXinASCII(TO_
BYTE(MID(TelephoneNum, 1, pos))));
                    END_FOR

                    cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_CALL);
                    cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_CALL_MODE.ENUM_BT_CALL_ASK);
                    SysMemCpy(ADR(cmd_buf[2]), ADR(tel_number), tel_len);
                    cmd_buf[tel_len+2] := 16#D;
                    cmd_buf[tel_len+3] := 16#A;
                    send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), tel_len+4);
            END_IF

            IF      F_TRIG_NotCallOut.Q     AND     LEN(TelephoneNum)>0     THEN            // Delete
1 digit number
                    TelephoneNum:=DELETE(TelephoneNum,1,LEN(TelephoneNum));
            END_IF

        END_IF
```

```
// Get Bluetooth Music Status
reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_MUSIC_STATUS);
req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(response));
IF req_ret THEN
        music_status := TO_INT(response.data);
        music_a2dp_connect_flag :=  music_status.0;
        MusicOpen := music_status.1;                                    // Play Music
END_IF

R_TRIG_Start(CLK:=music_time_step_ms>0);

IF      music_a2dp_connect_flag         THEN

        IF      R_TRIG_Start.Q   THEN
                MusicOpen:=TRUE;
        END_IF

        IF      F_TRIG_StartMusic.Q     THEN
                MusicOpen:=NOT          MusicOpen;

                IF      MusicOpen       THEN
                        cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_MUSIC);
                        cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_MUSIC_MODE.ENUM_BT_MUSIC_
START);
                        cmd_buf[2] := 16#D;
                        cmd_buf[3] := 16#A;
                        send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
                ELSE
                        cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_MUSIC);
                        cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_MUSIC_MODE.ENUM_BT_MUSIC_
PAUSE);
                        cmd_buf[2] := 16#D;
                        cmd_buf[3] := 16#A;
                        send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
                END_IF

        END_IF
```

```
IF      MusicOpen       THEN
        DelayShowCount:=DelayShowCount+1;
        IF      DelayShowCount>40       THEN
                ShowID:=ShowID+1;
                IF      ShowID>2        THEN
                        ShowID:=0;
                END_IF
                DelayShowCount:=0;
        END_IF

ELSE
        DelayShowCount:=0;
        ShowID:=0;
END_IF


 SysMemSet(ADR(music_name_wstring), 0, 128);            // Name of the Song
 SysMemSet(ADR(music_lyric_wdata), 0, 128);             // Singer

 // Get Bluetooth Music Song Name
 reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_SONG_
NAME);
 req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(music_name));
 IF req_ret THEN
        music_name_string := music_name.data;
        //music_name_len := LEN(music_name_string);
        music_name_len := music_name.size;
        SysMemCpy(ADR(music_name_byte), ADR(music_name_string), music_name_len);
        SysMemCpy(ADR(music_name_wstring), ADR(music_name_string), music_name_
len);
 END_IF

 // Get the lyrics corresponding to a Bluetooth Music Song
 reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_SONG_LYRIC);
 req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(music_lyric));
 IF req_ret THEN
        music_lyric_string := music_lyric.data;
        music_lyric_len := music_lyric.size;
        SysMemCpy(ADR(music_lyric_byte), ADR(music_lyric_string), music_lyric_len);
```

```
                SysMemCpy(ADR(music_lyric_wdata), ADR(music_lyric_string), music_lyric_len);
                END_IF


        // Get the total time that a Bluetooth music song has been playing (ms)
        reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_SONG_
DURATION);
        req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(music_total_time));
        IF req_ret THEN
                music_times_ms := TO_DWORD(music_total_time.data);
                TimeSecond:=TO_INT(music_times_ms/1000);
                DisplayTotalTime:='';

                IF      (TimeSecond/60)<10      THEN
                        DisplayTotalTime:=concat('0',TO_STRING(TimeSecond/60));
                ELSE
                        DisplayTotalTime:=concat(DisplayTotalTime,TO_STRING(TimeSecond/60));
                END_IF

                DisplayTotalTime:=concat(DisplayTotalTime,':');

                IF      (TimeSecond MOD 60)<10THEN
                        DisplayTotalTime:=concat(DisplayTotalTime,'0');
                        DisplayTotalTime:=concat(DisplayTotalTime,TO_STRING(TimeSecond MOD
60));
                ELSE
                DisplayTotalTime:=concat(DisplayTotalTime,TO_STRING(TimeSecond MOD 60));
                END_IF
        END_IF

        // Get the progress of Bluetooth music song playing time
        reqcmd := INT_TO_BYTE(HY_BT.BT_REQUEST_CMD.ENUM_BT_REQUEST_GET_SONG_
PROGRESS);
        req_ret := HY_BT.Sys_BT_GetCmdResponse(reqcmd, ADR(music_step));
        IF req_ret THEN
                music_time_step_ms := TO_DWORD(music_step.data);
                TimeSecond:=TO_INT(music_time_step_ms/1000);
                IF      TimeSecond<0    THEN
                        TimeSecond:=0;
```

```
                END_IF
                DisplayNowTime:='';

                IF      (TimeSecond/60)<10      THEN
                        DisplayNowTime:=concat('0',TO_STRING(TimeSecond/60));
                ELSE
                        DisplayNowTime:=concat(DisplayNowTime,TO_STRING(TimeSecond/60));
                END_IF

                DisplayNowTime:=concat(DisplayNowTime,':');

                IF      (TimeSecond MOD 60)<10THEN
                        DisplayNowTime:=concat(DisplayNowTime,'0');
                        DisplayNowTime:=concat(DisplayNowTime,TO_STRING(TimeSecond MOD
60));
                ELSE
                        DisplayNowTime:=concat(DisplayNowTime,TO_STRING(TimeSecond MOD
60));
                END_IF

        END_IF

        // Song Progress Bar
        MoveState:=LIMIT(0,437.0*(TO_REAL(music_time_step_ms)/TO_REAL(music_times_
ms)),437);

        // Next Song
        IF      F_TRIG_NextSong.Q       THEN
                test:=test+1;
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_MUSIC);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_MUSIC_MODE.ENUM_BT_MUSIC_PLAY_
NEXT_SONG);
                cmd_buf[2] := 16#D;
                cmd_buf[3] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
        END_IF

        // Previous Song
        IF      F_TRIG_LastSong.Q       THEN
```

```
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_ACTION_MUSIC);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_MUSIC_MODE.ENUM_BT_MUSIC_PLAY_
LAST_SONG);
                cmd_buf[2] := 16#D;
                cmd_buf[3] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 4);
            END_IF

            // Volume Increase
            IF      F_TRIG_VolumeUp.Q       THEN
                BT_Volume:=LIMIT(0,BT_Volume+1,10);

                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_REQUEST_SET_SYSTEM_
VOLUME);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_REQUEST_SET_SYSTEM_
VOLUME);
                cmd_buf[2] := BT_Volume;
                cmd_buf[3] := 16#D;
                cmd_buf[4] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 5);

            END_IF

            // Volume Decrease
            IF      F_TRIG_VolumeDown.Q    THEN
                IF BT_Volume > 0 THEN
                    BT_Volume:=LIMIT(0,BT_Volume-1,10);
                END_IF
                cmd_buf[0] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_REQUEST_SET_SYSTEM_
VOLUME);
                cmd_buf[1] := INT_TO_BYTE(HY_BT.BT_ACTION.ENUM_BT_REQUEST_SET_SYSTEM_
VOLUME);
                cmd_buf[2] := bt_volume;
                cmd_buf[3] := 16#D;
                cmd_buf[4] := 16#A;
                send_ret := HY_BT.Sys_BT_ExecCommandByte(ADR(cmd_buf), 5);
            END_IF

            IF      BT_Volume_Old<>BT_Volume      THEN
```

```
                    SysSaveRetains(FileName);                    (* Saving variable data to a regular file *)
                    BT_Volume_Old:=BT_Volume;
                END_IF

        END_IF

ELSE
        CallAct:=FALSE;
        ShowPoint:=0;
        DelayShow:=0;
        TelephoneNum:='';
END_IF

DisplayVolume:=BT_Volume*10;

END_IF

 (* Number Entry *)
F_TRIG_Num[0](CLK:=KeyPress[0]);
F_TRIG_Num[1](CLK:=KeyPress[1]);
F_TRIG_Num[2](CLK:=KeyPress[2]);
F_TRIG_Num[3](CLK:=KeyPress[3]);
F_TRIG_Num[4](CLK:=KeyPress[4]);
F_TRIG_Num[5](CLK:=KeyPress[5]);
F_TRIG_Num[6](CLK:=KeyPress[6]);
F_TRIG_Num[7](CLK:=KeyPress[7]);
F_TRIG_Num[8](CLK:=KeyPress[8]);
F_TRIG_Num[9](CLK:=KeyPress[9]);
F_TRIG_Num[10](CLK:=KeyPress[10]);
F_TRIG_Num[11](CLK:=KeyPress[11]);

IF      F_TRIG_Num[0].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'0');
END_IF

IF      F_TRIG_Num[1].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'1');
END_IF
```

```
IF      F_TRIG_Num[2].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'2');
END_IF

IF      F_TRIG_Num[3].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'3');
END_IF

IF      F_TRIG_Num[4].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'4');
END_IF

IF      F_TRIG_Num[5].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'5');
END_IF

IF      F_TRIG_Num[6].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'6');
END_IF

IF      F_TRIG_Num[7].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'7');
END_IF

IF      F_TRIG_Num[8].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'8');
END_IF

IF      F_TRIG_Num[9].Q         THEN
        TelephoneNum:=concat(TelephoneNum,'9');
END_IF

IF      F_TRIG_Num[10].Q        THEN
        TelephoneNum:=concat(TelephoneNum,'*');
END_IF

IF      F_TRIG_Num[11].Q        THEN
        TelephoneNum:=concat(TelephoneNum,'#');
END_IF
```

（2） CAN Initialization

Related Functions：

HY_CAN.CANInit(wDrvNr,wBauteRate,wBufferSize);

Related Libraries：HY_CAN 1.0.0.5

Variable Declaration：

```
can1init  :BOOL;
can2init  :BOOL;
```

Code Implementation：

```
IF NOT can1init THEN
        can1init:=HY_CAN.CANInit(0,250,2000);          //CAN1 initialisation with a baud rate of 250K
END_IF
IF NOT can2init THEN
        can2init:=HY_CAN.CANInit(1,250,2000);          //CAN2 initialisation with a baud rate of 250K
END_IF
```

（3） CAN Reception

Related Functions：

HY_CAN.MgrFindRecMessage(dwCOBId,wDrvNr,pBuffer);

Related Libraries：HY_CAN 1.0.0.5

Variable Declaration:

```
CAN1Buffer              :HY_CAN.CAN_Message;
CAN1Buffer_00           :ARRAY [1..15] OF HY_CAN.CAN_Message;
CAN1Buffer_18FF0200     :HY_CAN.CAN_Message;
rec_flag1               :BOOL;
Count                   :INT;
CAN1_CONTROL            :BOOL;
X                       :ARRAY [0..1] OF DWORD;
Z                       :DWORD;
clear_can1R             :BOOL;
```

Code Implementation：

```
FOR count:=1 TO 100 BY 1 DO          // 100 cycles in one period to ensure that no frames are lost in reception
IF can1init THEN
        rec_flag1 := HY_CAN.MgrFindRecMessage(0,0,ADR(CAN1Buffer));    // Receive CAN1 Telegrams
        IF rec_flag1 THEN
                CASE  CAN1Buffer.dwID  OF
                16#001:
```

```
                CAN1Buffer_00[1]:= CAN1Buffer;          // Receive the telegram with ID 001H
                X[1]:=X[1]+1;
                16#38FF0200:                            // Receive a telegram with ID 18FF0200H, ID+20000000
                CAN1Buffer_18FF0200:= CAN1Buffer;       when extending the frame
                Z:=Z+1;
                END_CASE
        ELSE
                count:=120;                             // If not received then jump out of the loop
        END_IF
END_IF
END_FOR
```

(4) CAN Transmission

Related Functions：

```
can1_send();                    //can1_send : HY_CAN.CanOpenWriteMSG_FB;
```

Related Libraries：HY_CAN 1.0.0.5

Variable Declaration :

```
sendcnt             :DWORD;
Init                :BOOL;
can1_send           :HY_CAN.CanOpenWriteMSG_FB;
CAN1SEND            :BOOL;
CAN_1S_ting         :BOOL;
ASQW                :BYTE;
CAN_1S              :BOOL;
test_can            :ARRAY [1..20] OF DWORD;
send_Count1         :DWORD := 0;
RST_SENDCAN1   :BOOL;
```

Code Implementation：

```
IF CAN1SEND THEN                                //CAN1 port data transmission
        IF can1init THEN
                CASE send_id1 OF
                        0:                      // With ID=0, send 101,11 22 33 44 55 66 77 88 message
                                test_can[1] := test_can[1]+1;
                                can1_send.wDrvNr := 0;
                                can1_send.bRtrFrame := 0;
                                can1_send.pPointer8byte := ASQW;
```

```
                    can1_send.ucLen := 8;
                    can1_send.dwCanID := 101;
                    can1_send.ucByte1:= 16#11;
                    can1_send.ucByte2:= 16#22;
                    can1_send.ucByte3:= 16#33;
                    can1_send.ucByte4:= 16#44;
                    can1_send.ucByte5:= 16#55;
                    can1_send.ucByte6:= 16#66;
                    can1_send.ucByte7:= 16#77;
                    can1_send.ucByte8:= 16#88;
                    can1_send();        // Calling the send function
                1:                      // With ID=1, send 201,11 22 33 44 55 66 77 88 message
                    test_can[2]:=test_can[2]+1;
                    can1_send.wDrvNr := 0;
                    can1_send.bRtrFrame := 0;
                    can1_send.pPointer8byte := ASQW;
                    can1_send.ucLen := 8;
                    can1_send.dwCanID := 201;
                    can1_send.ucByte1:=16#11;
                    can1_send.ucByte2:=16#22;
                    can1_send.ucByte3:=16#33;
                    can1_send.ucByte4:=16#44;
                    can1_send.ucByte5:=16#55;
                    can1_send.ucByte6:=16#66;
                    can1_send.ucByte7:=16#77;
                    can1_send.ucByte8:=16#88;
                    can1_send();
            END_CASE
        END_IF
    END_IF
```

（5） Video

Related Functions：

HY_Video.Sys_Video_Init(video_channel,video_channel_count,video_input_format,video_input_interface);

HY_Video.Sys_Video_Query_Status(init_timeout_ms,reserved);

HY_Video.Sys_Video_Signal_Valid(channel,reserved);

HY_Video.Sys_Video_Play(channel,pos_x,pos_y,pos_w,pos_h,videoflag);

HY_Video.Sys_Video_Close(channel,flag);

HY_Video.Sys_Video_Set_Default_Format(fmt,reserved);

HY_Video.Sys_Video_Get_Default_Format(reserved);

Related Libraries：HY_Video 1.0.0.6

Variable Declaration：

```
cnt                 : DWORD;
video_need_init     :BOOL ;
video_play          : BOOL ;
video_close         : BOOL;
act_read_fmt        : BOOL;
init_result         : BOOL;
insert_result       : BOOL;
act_clear           : BOOL := FALSE;
video_valid         : BOOL;
chan : BYTE         := 3;
x : DWORD           := 420;
y : DWORD           := 100;
w : DWORD           := 400;
h : DWORD           := 300;
fullscr             : BYTE := 0;
play_flag           : BOOL;
close_flag          : BOOL;
video_fmt           : INT := 255;
act_video_set       : BOOL;
set_result          : INT;
set_color           : DWORD;
get_result          : INT;
video_status        : INT;
video_init_succeed  : BOOL := FALSE;
video_cnt           : DWORD := 0;
try_times           : DWORD;
```

Code Implementation：

```
IF video_need_init THEN                    // Initialisation Video
        init_result := HY_Video.Sys_Video_Init(0, 4, HY_Video.ENUM_FORMAT_720P, HY_Video.YUV_422);
        IF init_result THEN
                video_need_init := FALSE;
        END_IF
END_IF

IF init_result AND NOT video_init_succeed THEN   // Determine if video initialisation is complete
        video_status := HY_Video.Sys_Video_Query_Status(0, 0);
        IF (video_status = 1) THEN
                video_init_succeed := TRUE;
        ELSE
                video_cnt := video_cnt + 1;
        END_IF
END_IF

IF video_init_succeed THEN                 // Determining whether a video signal is connected
        IF NOT insert_result  THEN
                try_times := try_times + 1;
                insert_result := HY_Video.Sys_Video_Signal_Valid(chan, 0);
                IF insert_result THEN
                        try_times := 0;
                        video_valid := TRUE;
                ELSE
                        video_valid := FALSE;
                END_IF
        END_IF

        IF video_valid THEN                // Show/Switch Video
                IF video_play THEN
                        video_play := FALSE;
                        play_flag:=HY_Video.Sys_Video_Play(channel:=chan,pos_x:=x,pos_y:=y,pos_
w:=w,pos_h:=h, videoflag:=fullscr);
                END_IF

                IF video_close THEN                    // Close Video
                        close_flag := HY_Video.Sys_Video_Close(chan, 0);
                        video_close := NOT close_flag;
```

```
                END_IF
        END_IF
END_IF
(* Setting the video format (0: 720P, 2: 1080P)
 1: Setup successful
-1: Unsupported input formats
-2: Lower firmware version (kernel version greater than or equal to K11, SO library version greater than 1.0.0.6)
-3: Setup failed
The A8 7" and 10.1" displays are shipped with a default camera format of 720P.
(This function is only invoked to set the corresponding video mode when the external camera standard is changed, and must
 be restarted after successful setting*)

IF act_video_set THEN                                   // Video Format Setting
        set_result := HY_Video.Sys_Video_Set_Default_Format(video_fmt, 0);
        IF (set_result = 1) THEN
                set_color := 16#FF00FF00;
        ELSE
                set_color := 16#FFC0C0C0;
        END_IF
        act_video_set := NOT act_video_set;
END_IF
IF NOT act_read_fmt THEN                                // Reading Video Format
        get_result := HY_Video.Sys_Video_Get_Default_Format(0);
        IF (get_result >= 0) THEN
                video_fmt := get_result;
        END_IF
        act_read_fmt := TRUE;
END_IF
```

（6） WiFi

Related Functions：

HY_Wifi.Sys_Wifi_Open(mode,connected_ap,ip_sta);

HY_Wifi.Sys_Wifi_Close(reserved);

SysMemSet(dwDest,bCharacter,dwCount);

HY_Wifi.Sys_Wifi_Scan(result);

HY_Wifi.Sys_Wifi_Connect(ssid_name,passwd,ip_sta);

SysMemCpy(dwDest,dwSrc,dwCount);

HY_Wifi.Sys_Wifi_RemoveAP(ssid_name);

HY_Wifi.Sys_Wifi_AP_Get_AccountInfo(wifi_id,plogin_info);

HY_Wifi.Sys_Wifi_AP_Set_AccountInfo(wifi_id,login_info);

Related Libraries：SysMem23 3.5.13.0

HY_WIFI 1.0.0.5

Variable Declaration：

```
wifi_loop                : DWORD;
Workmode                 : BYTE := 1;
retry_times              : DWORD := 0;
wifi_open_act            : BOOL;
wifi_scan_act            : BOOL;
wifi_connect_act         : BOOL;
wifi_delete_act          : BOOL;
wifi_ap_first_read       : BOOL;
wifi_ap_read_info_act    : BOOL;
wifi_ap_change_info_act  : BOOL;
change_result            : BOOL;
open_result              : BOOL;
close_result             : BOOL;
connect_result           : BOOL;
delete_result            : BOOL;
ap_scan_number : INT;
Aplist                   : HY_Wifi.WifiAPList;
Apscanlist               : HY_Wifi.WifiScanResult;
Apinfo                   : HY_Wifi.WifiInfo;
Apuserinfo               : HY_Wifi.WifiAPAccountInfo;
open_idx                 : INT := 0;
con_idx                  : INT := -1;
conn_state               : INT := -1;
```

```
Ssid                    : STRING := '111';
Pwd                     : STRING := '123456789';
Flag                    : BOOL := 0;
requery_cnt             : INT := 0;
max_cnt                 : INT := 3;
Iparray                 : HY_Wifi.NetIP;
Ipaddr                  : ARRAY[0..3] OF BYTE;
Conncolor               : DWORD := 16#FF000000;
Scancolor               : DWORD := 16#FF000000;
```

Code Implementation：

```
IF wifi_open_act THEN
        IF (open_idx = 0) THEN
                open_result := HY_Wifi.Sys_Wifi_Open(workmode, ADR(aplist), ADR(iparray)); // Turn on wifi as STATION mode
        IF open_result THEN
                        open_idx := 1;
                END_IF
        ELSE
                close_result := HY_Wifi.Sys_Wifi_Close(0);              // Turn off wifi as AP mode

                IF close_result THEN
                        open_idx := 0;
                        con_idx := 2;
                        SysMemSet(ADR(iparray.addr), 0, 4);
                END_IF
        END_IF
        wifi_open_act := FALSE;
END_IF

IF wifi_scan_act THEN
        ap_scan_number := HY_Wifi.Sys_Wifi_Scan(result := ADR(apscanlist));  // Scanning hotspots number
        wifi_scan_act := FALSE;
        IF (ap_scan_number > 0) THEN
                scancolor := 16#FF0000CC;
        ELSE
                scancolor := 16#FFFF0000;
        END_IF
END_IF
```

```
IF wifi_connect_act THEN
        retry_times := retry_times + 1;
        SysMemSet(ADR(iparray.addr), 0, 4);
        con_idx := -1;
        connect_result := HY_Wifi.Sys_Wifi_Connect(ssid_name := ssid, passwd := pwd, ip_sta := ADR(iparray));
// wifi connection
        IF connect_result OR retry_times > 0 THEN
                wifi_connect_act:= FALSE;
                retry_times := 0;
                        IF connect_result THEN
                        con_idx := 0;
                        conncolor := 16#FFFF00CC;
                        SysMemCpy(ADR(ipaddr), ADR(iparray.addr), 4);       // Obtaining an IP address
                ELSE
                        con_idx := 1;
                        conncolor := 16#FFFF0000;
                END_IF
        END_IF
END_IF

IF wifi_delete_act THEN                                                     // Delete wifi hotspot
        delete_result := HY_Wifi.Sys_Wifi_RemoveAP(ssid_name := ssid);
        wifi_delete_act := FALSE;
END_IF

IF wifi_ap_read_info_act OR NOT wifi_ap_first_read  THEN                    // Read Wifi AP username and password
        change_result := HY_Wifi.Sys_Wifi_AP_Get_AccountInfo(0, ADR(apuserinfo));
        wifi_ap_first_read := TRUE;
END_IF

IF wifi_ap_change_info_act THEN                                             // Update Wifi AP username and password
        change_result := HY_Wifi.Sys_Wifi_AP_Set_AccountInfo(0, apuserinfo);
END_IF
```

Appendix II FCC Warning

This device complies with part 15 of the FCC rules. Operation is subject to the following two conditions:

(1) this device may not cause harmful interference, and

(2) this device must accept any inte rference received, including interference that may cause undesired operation.

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a Car installation. This equipment generates uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interferenceto radio or television reception,which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

-Reorient or relocate the receiving antenna.

• Increase the separation between the equipment and receiver.

• Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.

• Consult the dealer or an experienced radio/TV technician for help.

Radiation Exposure Statement

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment.This equipment should be installed and operated with minimum distance 20cm between the radiator and your body.

1. Professional installation must be justified in the filing, and grant condition must state " This device must be
professionally installed. "
Description: Device is Display screen and must need special trained professional in configurating and installing the product. More details please refer to user manual exhibits.
2. Professional installation does not permit use of any antenna with the transmitter; the permitted types of antenna must be specified.
Description: Below listed of Antennas has been compliance FCC Rules Part 15 requirement, more details please refer to test reports.Antenna Type: DipoleR-N Type Connector, Antenna Gain (dBi): 2400-2500MHz 3.0dBi
3. The applicant should address the following items when justifying professional installation.
i) To qualify for professional installation, the applicant must explain why the hardware
Description: Due to this product will not be sold directly to the general public through retail store therefore the hardware is not readily available to average customer.
ii) Marketing—Device cannot be sold via retail to the general public or by mail order; it must be sold to authorized dealers or installers only
Description: Due to this product will not be sold directly to the general public through retail store. It will be sold to authorized declarers or install only.
iii) Filing must show that intended use is not for consumers and general public; rather device is generally for industrial/commercial use.
Description: Device is for industrial/commercial use.
iv) Explain what is unique, sophisticated, complex, or specialized about the equipment that REQUIRES it to be installed by a professional installer?
Description: Please be advised that due to the unique Market and function targeted by this product. this product will need special trained professional in selecting the proper location, adjusting the antenna anele on the outside pole or on the wall to satisfy the relevant rule requirements, we hereby declare that the product will be distributed through controlled distribution channel which has special trained professional to install this product and will not be sold directly to the general public through retail store.
4. Other professional installation requirements
(1) Installation must be controlled.
Description: The product will be distributed through controlled distribution channel which has special trained professional to install this product.
(2) Installed by licensed professionals (*e.g.,* device sold to dealer who hire installers).
Description: Device sold to dealer who hires installers and need special trained professional in configuring and installing the product.
(3) Installation requires special training (*e.g.,* special programming, access to keypad, field strength measurements made).
Description: The product needs special programming. access to keypad. field strength measurements made, so must need special trained professional in configuring and installing the product.