AN5336
Application note

ST8500 host interface driver

Description, purpose and scope

This application note provides a description of the Host Interface SW Driver. It can be integrated into the customer application running on an external microcontroller (Host) like the STM32 to interface with the ST8500 host interface in the case of G3 protocol.

This Host Interface Driver operates in two modes:

- Firmware download mode in which the STM32 application interacts with the commands implemented in the ST8500 boot ROM
- Host interface mode in which the STM32 application interacts with the commands implemented in the ST8500 G3 image.

The application note first provides an overview of the Host Interface Driver for both modes. It also provides some hints to help the porting to the customer environment.

Contents

List of tables

List of figures

1 Introduction

1.1 Document overview

This document aims at providing a detailed description of the Host Interface Driver. This Host Interface Driver (HI DRV) is running on a Host Controller and handles the communication between any application running on the Host Controller and the ST8500 device that runs the G3 PLC software.

- *Section 2* is an introduction about the Host Interface (from hardware and software point of view) and the modes of operation
- *Section 3* details the Firmware Download mode
- *Section 4* details the Host Interface mode
- *Section 5* gives some hints to integrate the HI Driver.

1.2 List of abbreviations

ADP	ADaPtation layer
HI	Host Interface
ISR	Interrupt Service Routine
MAC	Medium Access Control layer
NVM	Non-Volatile Memory
PE	Protocol Engine (ARM® Cortex™ in ST8500)
PHY	PHYsical layer
PLC	Power Line Communication
RTE	Real Time Engine (DSP in ST8500)
RTOS	Real Time Operating System

2 Host interface introduction

2.1 Hardware interface overview

Only the UART interface is supported to communicate between Host and ST8500 in HI DRV.

The Host interface is composed of 3 lines:

- **TX** : UART transmit line
- **RX** : UART receive line
- **Reset** : Active low input line to put the ST8500 in reset

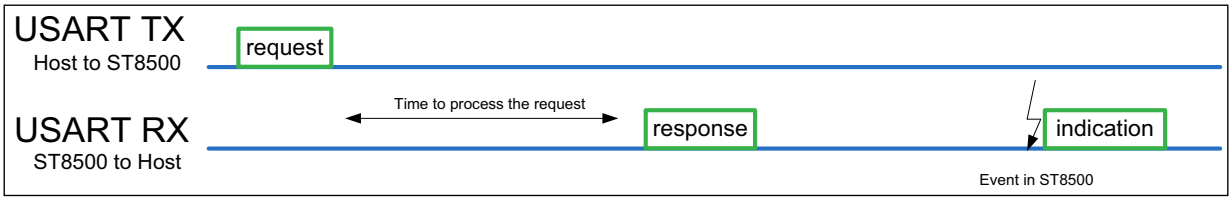
2.2 Software interface overview

The communication between the Host controller and the ST8500 is based on messages sent and received on the UART interface.

The communication is full duplex:

- The Host sends request messages to ST8500
- ST8500 sends confirm messages (corresponding to the previously issued request) or indication messages to the Host.

Figure 1. Host interface principle



Messages have the following format:

Table 1. Request message format (default mode)

Synchro		Control		State		Message			Check
0 x16	0 x 16	CMD ID	LEN	MODE	STATE	Payload [0]	...	Payload [LEN-1]	CRC
2 bytes		1 byte	2 bytes	1 byte	4 bytes	LEN bytes			2 bytes

Table 2. Confirm/indication message format (default mode)

Synchro		Control		State		Error	Message			Check
0 x16	0 x 16	CMD ID	LEN	MODE	STATE	EC	Payload [0]	...	Payload [LEN-2]	CRC
2 bytes		1 byte	2 bytes	1 byte	4 bytes	1 byte	LEN-1 bytes			2 bytes

Table 3. Request message format (secure mode)

Synchro		Control		Security		Message			Check	
0 x16	0 x 16	CMD ID	LEN	SEC MODE	COUNTER	Payload [0]	...	Payload [LEN-2]	TAG	CRC
2 bytes		1 byte	2 bytes	1 byte	4 bytes	LEN-1 byte			8 bytes	2 bytes
		Header				Ciphertext			Tag	

Table 4. Confirm/indication message format (secure mode)

Synchro		Control		Security		Error	Message			Check	
0 x16	0 x 16	CMD ID	LEN	SEC MODE	COUNTER	EC	Payload [0]	...	Payload [LEN-2]	TAG	CRC
2 bytes		1 byte	2 bytes	1 byte	4 bytes	1 byte	LEN-1 byte			8 bytes	2 bytes
		Header					Ciphertext			Tag	

Note that the EC field is not present in response messages sent by the ST8500 bootROM (case of FW download). Please refer to [2] for details.

For convention in this document, a received message with:

- No error code and good CRC is considered as **ACK** response and generate **HI_EVENT_RESP_ACK**
- Error code or bad CRC is considered as **NACK** response and generate **HI_EVENT_RESP_NACK**
- In case no answer is received in due time, system generates **HI_EVENT_TIMEOUT**.

2.3 Modes of operation

Depending on ST8500 boot pins, ST8500 boot code can download RTE/PE images from its attached serial Flash (“boot on Flash”) or from the host connected on the host interface (“boot on USART”).

Thus, the host interface can be used in 2 different modes:

- Just after any ST8500 reset and if boot pins are set to “boot on USART”
  - This is called “firmware download mode” in the current document
- During runtime operation to control PLC activity
  - This is called “host interface mode” in the current document.

The application has to indicate to the HI Driver the mode using the following API:

```
void HIF_SetAppMode(HiSetAppMode_t mode)
with mode =
- MODE_BOOTROM to select Firmware download mode
- MODE_G3_APP to select Host Interface mode (unsecure mode)
- MODE_G3_APP_SEC1 to select Host Interface mode (secure mode 1)
- MODE_G3_APP_SEC2 to select Host Interface mode (secure mode 2)
```

2.4 ST8500 boot sequence and messaging

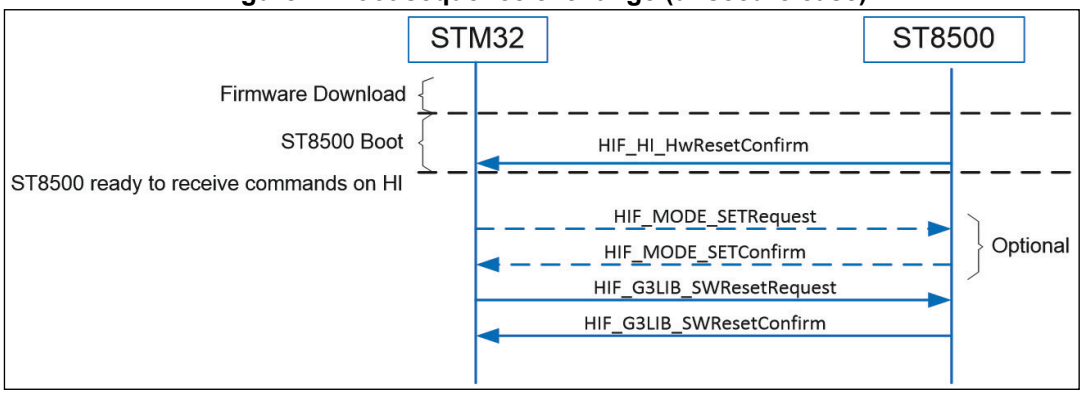
After download of the ST8500 image, during the boot of this image, the following sequence occurs:

- the ST8500 sends a message to confirm the reset using HIF_HI_RESETSTATE_CNF message
- If ST8500 runs in secure mode (indicated by MODE field from reset confirm message), the secure initialization is performed
- the host may send the G3 Lib mode
  - If not sent, the G3 Lib mode is set to ADP by default
- the host sends the band and device type with HIF_G3LIB_SWRESET_REQ message
- the ST8500 acknowledges the request by sending the HIF_G3LIB_SWRESET_CNF message.

The complete sequence of messages, in case of non-secure Host Interface, can be found below:

```
RESP-> 1616 01 0100 00 00000000 00 10EF
REQ -> 1616 02 0100 00 00000000 03 37F2
RESP-> 1616 03 0100 00 00000000 00 7729
REQ -> 1616 24 0200 00 00000000 0000 F8CA
RESP-> 1616 25 0400 00 00000000 00000000 0965
```

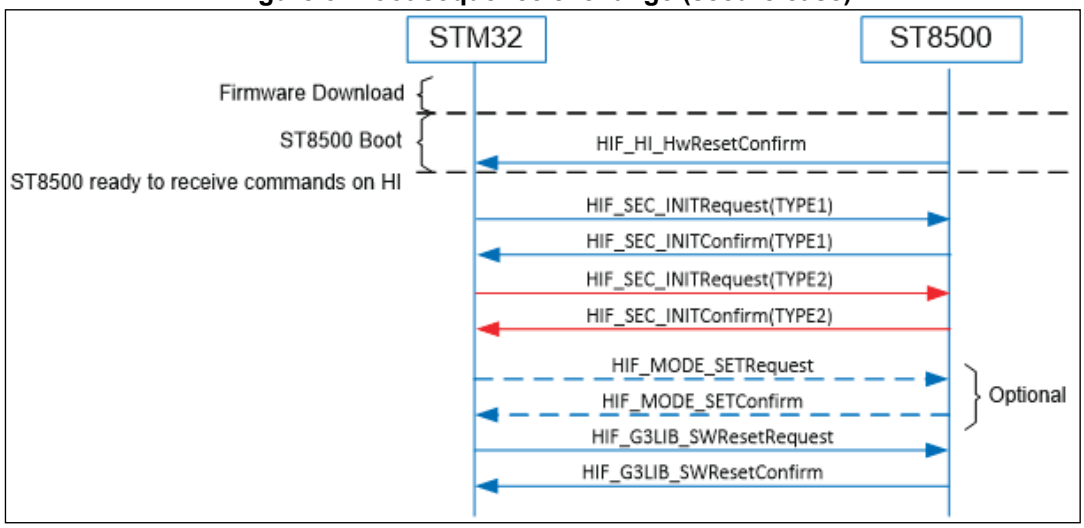
Figure 2. Boot sequence exchange (unsecure case)



In the case of secure Host Interface (secure mode 1), the sequence is described below:

```
RESP-> 1616 01 0200 01 00000000 0001  C599
REQ -> 1616 16 0100 01 00000000 01  C458
RESP-> 1616 17 2200 01 00000000
       0001BB8F8CA0AF2C0834F3EC4EC55EDA
       CB027B3E817EECCC2E835901888A05BD8C5A  FC65
REQ -> 1616 16 1100 01 00000000
       79A1BB092926EB3F6AF98173A730DEAB39
       4A8F0E77EEF175D6 98A3
RESP-> 1616 17 1200 01 00000000
       0002A29BFD5E22694137F9C801F58F843200
       AF4445BD16EF26BB 30C3
REQ -> 1616 02 0100 01 00000000 03  97B7
RESP-> 1616 03 0100 01 00000000 00  D76C
REQ -> 1616 24 0200 01 00000000 0000 XXXX
RESP-> 1616 25 0400 01 00000000 00000000 XXXX
```

Figure 3. Boot sequence exchange (secure case)



3 Firmware download mode

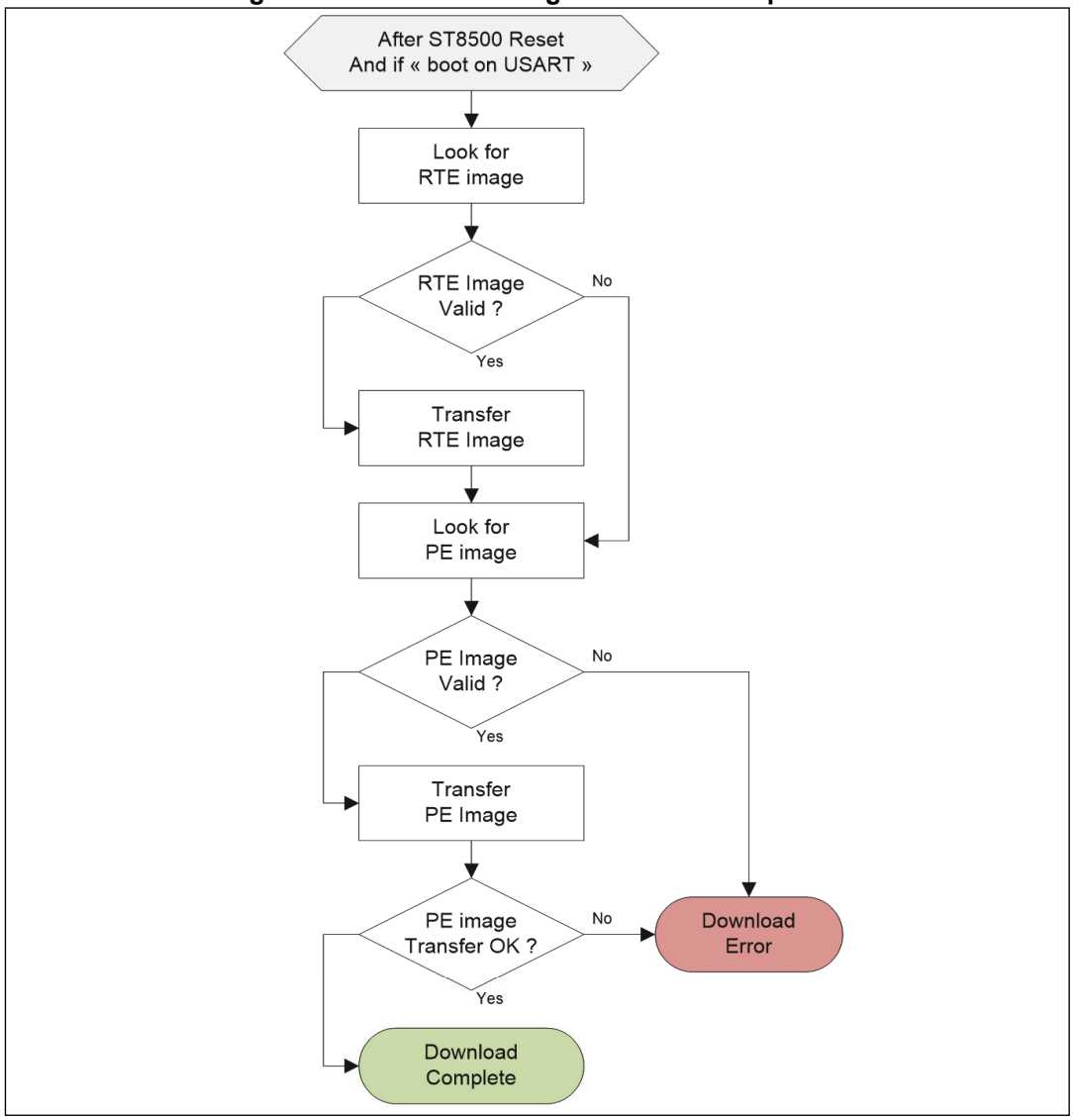
3.1 Overview

This mode only applies to the case “boot on USART”.

During initialization phase, it is possible to interact with the ST8500 using commands that are implemented in ST8500 Boot ROM. The details of these commands can be found in [2].

The main operation is the download of firmware images towards the ST8500 for both RTE and PE processors. It is necessary to download 2 images; first the image for RTE and then the image for PE. This operation is shown in the diagram below.

Figure 4. RTE and PE images download sequence



The images are assumed to be located in a non-volatile memory accessible from the host.

3.2 Description

When triggering a download from the application the following is executed:

- Initialization of Firmware Download state machine variables.

```
hiVars.status.step = 0;
 hiAppVars.imgType = 0xFF;
 hiAppVars.baudRateStep = HI_BAUDRATE_RESET;
```

- The Firmware Download state machine is invoked with the event HI_EVENT_IMG_DL.
  - This triggers the download of the RTE image first
- Once RTE image transfer is completed (or if no valid RTE image was found) the state machine is invoked again using the same event HI_EVENT_IMG_DL in order to download the PE image.

The Firmware download state machine is implemented in the following function.

```
void HIF_ImgDlStateMachine(uint32_t event)
```

The possible events to trigger the various transitions are:

- HI_EVENT_IMG_DL
  - This event triggers the start of an image download
- HI_EVENT_TIMEOUT
  - This event is used on expiration of the timer used for timeout
- HI_EVENT_RESP_ACK
  - This event is used after reception of a valid message (good CRC and no error code)
- HI_EVENT_RESP_NACK
  - This event is used after reception of an invalid message.

The FW download state machine uses the following API to read the PE and RTE images:

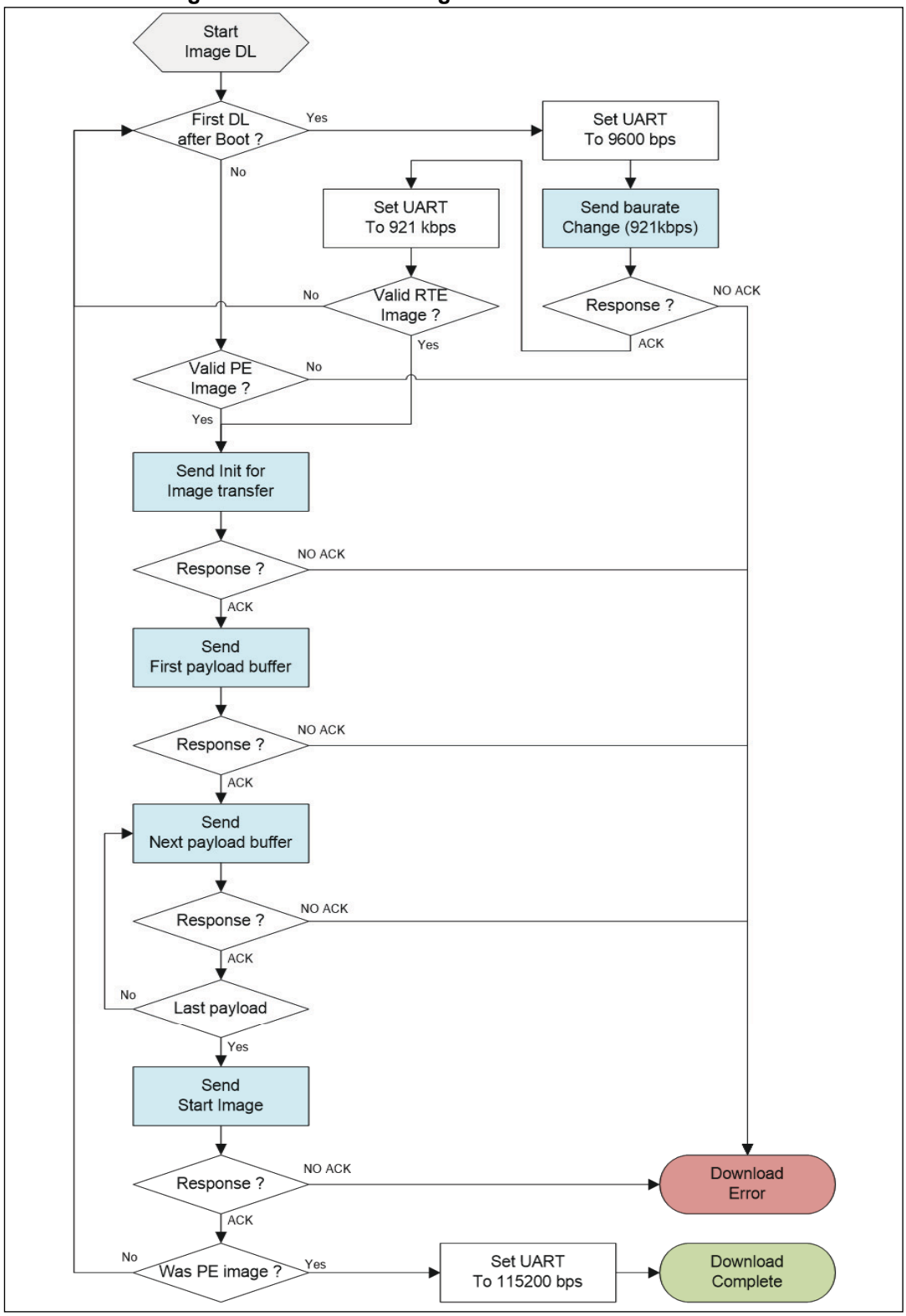
```
void HI_ImageRead(uint8_t *ptr, uint8_t imageType,
                  uint32_t offset, uint16_t size)
```

This function reads <size> bytes of the image, starting from <offset> bytes. It is part of the functions that must be ported for each application in hi_os_wrapper.c/.h.

When image download is ongoing, the transition between states is done on reception of HI_EVENT_RESP_ACK and HI_EVENT_RESP_NACK. Details on the various steps is shown in the diagram below.

In the diagram below, “NO ACK” means either message response with an error (NACK) or no message response (TIMEOUT).

Figure 5. RTE and PE images download state machine



In case of error, the download is just aborted and an error message is printed out. It is up to the application to retry download.

4 Host interface mode

4.1 Overview

Once the initialization phase of the ST8500 is completed and the ST8500 PE FW code is up and running, it is possible to interact with the ST8500 using commands that are implemented in the ST8500 G3 PLC software.

The operational mode of the ST8500 G3 PLC library may be selected from the host and may be chosen in the following list: PHY, MAC, ADP, BOOT, IPv6 + ADP or Ipv6 + BOOT. Depending of the selected operational mode, a subset of the complete list of available commands is usable.

Refer to [*1*] to get details about the way to use the Host Interface and also to have the complete list of commands with their availability in function of the operational mode.

In this mode of operation, the Host waits for confirm message before issuing a new request message.

4.2 Driver API naming rules

For each Host Interface API, the following functions are defined in the HI driver with the following naming rules:

```
HIF_XXX_Name (void *ptr)
HIF_XXX_Name (void *ptr)
XXXHI, G3, IP, BOOT, ADP, MAC, PHY
NameADPDDataRequest, ADPMNetworkJoinConfirm…
XXX_Name_t Structure used to convey payload
```

Example:

```
HIF_ADP_ADPDDataRequest
HIF_ADP_ADPMNetworkJoinConfirm
With structure: ADP_AdpmNetworkJoinConfirm_t
```

4.3 Callback functions

Feedback from the HI Driver to the application is handled thanks to 3 callback functions:

- Confirm callback function
  - Invoked when confirm data are available
- Indication callback function
  - Invoked when indication data are available
- Error callback function
  - Invoked in case of error.

These callback functions must be implemented in the application code and registered at initialization using the following HI DRV API:

```
HIF_RegisterCallBack(HI_indCallback, HI_cnfCallback,
HI_errCallback);
```

In the current implementation example, the registration is invoked in HI_StartTasks from hi_Tasks.c.

4.4 List of Host Interface APIs

Table 5. List of Host Interface APIs

API name	Parameter structure name	CallBack Id
HIF_HI_HwResetRequest		
HIF_HI_HwResetConfirm		HIF_CNF_RESET
HIF_HI_ModeSetRequest	uint8_t *	
HIF_HI_ModeSetConfirm		HIF_CNF_MODE
HIF_HI_ModeGetRequest		
HIF_HI_ModeGetConfirm	uint8_t *	HIF_CNF_MODEGET
HIF_HI_BaudrateSetRequest	uint32_t *	
HIF_HI_BaudrateSetConfirm	uint32_t *	HI_CNF_BAUDRATE
HIF_HI_TestSetRequest	hi_testsetreq_t *	
HIF_HI_TestSetConfirm		HI_CNF_TESTSET
HIF_HI_TestGetRequest	hi_testgetreq_t *	
HIF_HI_TestGetConfirm	hi_testgetreq_t *	HI_CNF_TESTGET
HIF_HI_SflashRequest	hi_sflashreq_t *	
HIF_HI_SflashConfirm	hi_sflashreq_t *	HI_CNF_SFLASH
HIF_HI_NvmRequest	hi_nvm_req_t *	
HIF_HI_NvmConfirm	hi_nvm_req_t *	HI_CNF_NVRAM
HIF_HI_TraceRequest	HI_TraceRequest_t *	
HIF_HI_TraceConfirm	HI_TraceConfirm_t *	HIF_CNF_TRACE
HIF_HI_RdbgRequest	hi_rdbg_req_t *	
HIF_HI_RdbgConfirm	hi_rdbg_req_t *	HI_CNF_RDBG
HIF_HI_SecInitRequest	hi_SecInitRequest_t *	
HIF_HI_SecInitConfirm	hi_SecInitConfirm_t *	HIF_CNF_SECINIT
HIF_HI_OtpRequest	hi_otp_req_t *	
HIF_HI_OtpConfirm	hi_otp_conf_t *	HIF_CNF_OTP
HIF_G3LIB_GetRequest	G3_LIB_GetAttributeRequest_t *	
HIF_G3LIB_GetConfirm	G3_LIB_GetAttributeConfirm_t *	HI_CNF_G3LIBGET
HIF_G3LIB_MultipleGetRequest	G3_LIB_GetAttributeRequest_t *	

Table 5. List of Host Interface APIs (continued)

API name	Parameter structure name	CallBack Id
HIF_G3LIB_MultipleGetConfirm	uint8_t *	HIF_CNF_G3LIBMULTIGET
HIF_G3LIB_SetRequest	G3_LIB_SetAttributeRequest_t *	
HIF_G3LIB_SetConfirm	G3_LIB_SetAttributeConfirm_t *	HI_CNF_G3LIBSET
HIF_G3LIB_SWResetRequest	G3_LIB_SWResetRequest_t *	
HIF_G3LIB_SWResetConfirm	G3_LIB_SWResetConfirm_t *	HIF_CNF_G3LIBSWRESET
HIF_G3LIB_TestModeRequest	G3_LIB_TestModeRequest_t	
HIF_G3LIB_TestModeConfirm	G3_LIB_TestModeConfirm_t	HIF_CNF_G3LIBTESTMODE
HIF_G3LIB_EventIndication	G3_LIB_EventIndication_t	HIF_IND_G3LIB_EVT
HIF_PHY_PdDataRequest	PHY_PdDataRequest_t *	
HIF_PHY_PdDataConfirm	PHY_PdDataConfirm_t *	HIF_CNF_PDDATA
HIF_PHY_PdDataIndication	PHY_PdDataIndication_t *	HIF_IND_PDDATA
HIF_PHY_PdTrxRequest	uint8_t *	
HIF_PHY_PdTrxConfirm	uint8_t *	HIF_CNF_PDTRX
HIF_PHY_PdCsRequest		
HIF_PHY_PdCsConfirm	uint8_t *	HIF_CNF_PDCS
HIF_MAC_MCPSDataRequest	MAC_McpsDataReq_t *	
HIF_MAC_MCPSDataConfirm	MAC_McpsDataConfirm_t *	HIF_CNF_MACDATA
HIF_MAC_MCPSDataIndication	MAC_McpsDataIndication_t *	HIF_IND_MACDATA
HIF_MAC_MLMEBeaconNotifIndication	MAC_MlmeBeaconNotify_t *	HIF_IND_MACBEACON
HIF_MAC_MLMEResetRequest	MAC_MlmeResetReq_t *	
HIF_MAC_MLMEResetConfirm	MAC_MlmeResetConfirm_t *	HIF_CNF_MACRESET
HIF_MAC_MLMEScanRequest	MAC_MlmeScanReq_t *	
HIF_MAC_MLMEScanConfirm	MAC_MlmeScanConfirm_t *	HIF_CNF_MACSCAN
HIF_MAC_MLMEStartRequest	MAC_MlmeStartReq_t *	
HIF_MAC_MLMEStartConfirm	MAC_MlmeStartConfirm_t *	HIF_CNF_MACSTART
HIF_ADP_ADPDDataRequest	ADP_AdpdDataRequest_t *	
HIF_ADP_ADPDDataConfirm	ADP_AdpdDataConfirm_t *	HI_CNF_ADPDATA
HIF_ADP_ADPDDataIndication	ADP_AdpdDataIndication_t *	HI_IND_ADPDATA
HIF_ADP_ADPMDiscoveryRequest	ADP_AdpmDiscoveryRequest_t *	
HIF_ADP_ADPMDiscoveryConfirm	ADP_AdpmDiscoveryConfirm_t *	HI_CNF_ADPDISCO
HIF_ADP_ADPMNetworkStartRequest	ADP_AdpmNetworkStartRequest_t *	
HIF_ADP_ADPMNetworkStartConfirm	ADP_AdpmNetworkStartConfirm_t *	HI_CNF_ADPSTART
HIF_ADP_ADPMNetworkJoinRequest	ADP_AdpmNetworkJoinRequest_t *	
HIF_ADP_ADPMNetworkJoinConfirm	ADP_AdpmNetworkJoinConfirm_t *	HI_CNF_ADPJOIN
HIF_ADP_ADPMNetworkLeaveRequest		

Table 5. List of Host Interface APIs (continued)

API name	Parameter structure name	CallBack Id
HIF_ADP_ADPMNetworkLeaveConfirm	ADP_AdpmNetworkLeaveConfirm_t *	HI_CNF_ADPLEAVE
HIF_ADP_ADPMNetworkLeaveIndication		HI_IND_ADPLEAVE
HIF_ADP_ADPMResetRequest		
HIF_ADP_ADPMResetConfirm	ADP_AdpmResetConfirm_t *	HI_CNF_ADPRESET
HIF_ADP_ADPMNetworkStatusIndication	ADP_AdpmNetworkStatusIndication_t *	HI_IND_ADPNTW_STATUS
HIF_ADP_ADPMRouteDiscoveryRequest	ADP_AdpmRouteDiscoveryRequest_t *	
HIF_ADP_ADPMRouteDiscoveryConfirm	ADP_AdpmRouteDiscoveryConfirm_t *	HI_CNF_ADPROUTE
HIF_ADP_ADPMPathDiscoveryRequest	ADP_AdpmPathDiscoveryRequest_t *	
HIF_ADP_ADPMPathDiscoveryConfirm		HI_CNF_ADPPATH
HIF_ADP_ADPMLbpRequest	ADP_AdpmLbpRequest_t *	
HIF_ADP_ADPMLbpConfirm	ADP_AdpmLbpConfirm_t *	HI_CNF_ADPLBP
HIF_ADP_ADPMLbpIndication	ADP_AdpmLbpIndication_t *	HI_IND_ADPLBP
HIF_ADP_ADPMRouteDeleteRequest	ADP_AdpmRouteDeleteRequest_t	
HIF_ADP_ADPMRouteDeleteConfirm	ADP_AdpmRouteDeleteConfirm_t	HIF_CNF_ROUTEDEL
HIF_BOOT_ServerStartRequest	BOOT_ServerStartRequest_t *	
HIF_BOOT_ServerStartConfirm	BOOT_ServerStartConfirm_t *	HI_CNF_BOOT_SERVER_START
HIF_BOOT_ServerStopRequest		
HIF_BOOT_ServerStopConfirm	BOOT_ServerStopConfirm_t *	HI_CNF_BOOT_SERVER_STOP
HIF_BOOT_ServerLeaveIndication	BOOT_ServerLeaveIndication_t *	HIF_IND_BOOT_SERVER_LEAVE
HIF_BOOT_ServerKickRequest	BOOT_ServerKickRequest_t *	
HIF_BOOT_ServerKickConfirm	BOOT_ServerKickConfirm_t *	HI_CNF_BOOT_SERVER_KICK
HIF_BOOT_ServerJoinIndication	BOOT_ServerJoinIndication_t *	HIF_IND_BOOT_SERVER_JOIN
HIF_BOOT_DeviceStartRequest	BOOT_DeviceStartRequest_t *	
HIF_BOOT_DeviceStartConfirm	BOOT_DeviceStartConfirm_t *	HI_CNF_BOOT_DEVICE_START
HIF_BOOT_DeviceLeaveRequest		
HIF_BOOT_DeviceLeaveConfirm	BOOT_DeviceLeaveConfirm_t *	HI_CNF_BOOT_DEVICE_LEAVE
HIF_BOOT_DeviceLeaveIndication		HIF_IND_BOOT_DEVICE_LEAVE
HIF_BOOT_DevicePANSortIndication	BOOT_DevicePANSortIndication_t	HIF_IND_BOOT_DEVICE_PANSORT
HIF_BOOT_DevicePANSortRequest	BOOT_DevicePANSortRequest_t	

Table 5. List of Host Interface APIs (continued)

API name	Parameter structure name	CallBack Id
HIF_BOOT_DevicePANSortConfirm	BOOT_DevicePANSortConfirm_t	HIF_CNF_BOOT_DEVICE_PANSORT
HIF_BOOT_ServerGetPSKIndication	BOOT_ServerGetPSKIndication_t	HIF_IND_BOOT_SERVER_GETPSK
HIF_BOOT_ServerSetPSKRequest	BOOT_ServerSetPSKRequest_t	
HIF_BOOT_ServerSetPSKConfirm	BOOT_ServerSetPSKConfirm_t	HIF_CNF_BOOT_SERVER_SETPSK
HIF_IP_UdpDataRequest	IP_G3UdpDataRequest_t *	
HIF_IP_UdpDataConfirm	IP_DataConfirm_t *	HIF_CNF_UDPDATA
HIF_IP_UdpDataIndication	IP_DataIndication_t *	HIF_IND_UDPDATA
HIF_IP_UdpConnSetRequest	IP_UdpConnSetRequest_t *	
HIF_IP_UdpConnSetConfirm	IP_UdpConnSetConfirm_t *	HIF_CNF_UDPCONN_SET
HIF_IP_UdpConnGetRequest	IP_UdpConnGetRequest_t *	
HIF_IP_UdpConnGetConfirm	IP_UdpConnGetConfirm_t *	HIF_CNF_UDPCONN_GET
HIF_IP_IcmpEchoRequest	IP_G3IcmpDataRequest_t *	
HIF_IP_IcmpEchoConfirm	IP_DataConfirm_t *	HIF_CNF_ICMP_ECHO
HIF_IP_IcmpEchoIndication	IP_DataIndication_t *	HIF_IND_ICMP_ECHO

Note: The table above gives the list of the currently available APIs.

4.5 Driver implementation

4.5.1 Buffer management

Request messages are handled the following way:

- Request messages are handled sequentially
- These messages are prepared in TX buffer (hiReqBuf[])
- Once correctly formatted and optionally ciphered/authenticated, CRC is computed
- Once complete, the message is sent on the USART TX line
- The TX buffer cannot be used until the current message is fully transmitted.

Confirm/Indication messages are handled the following way:

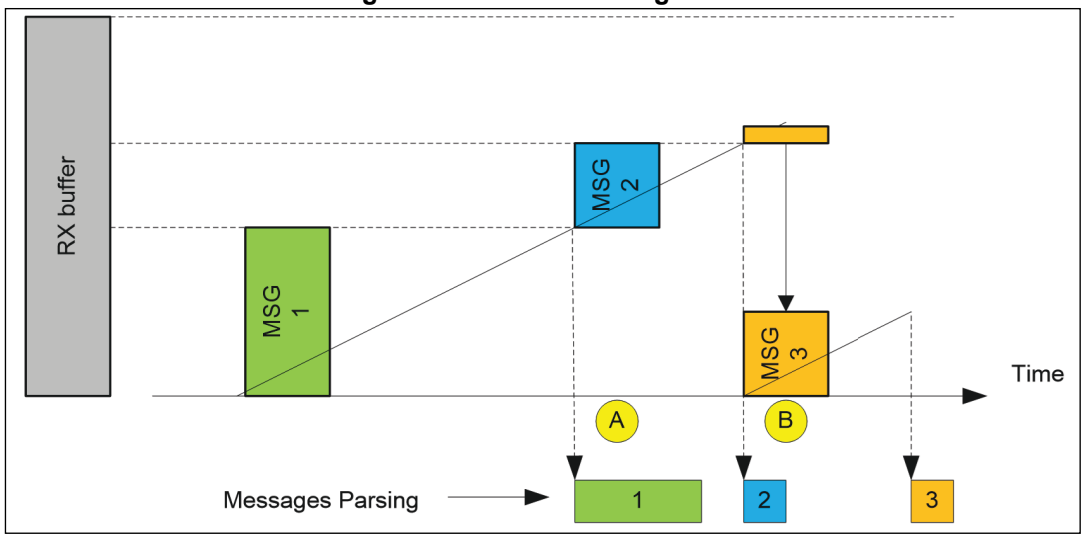
- These messages come from ST8500 asynchronously and can be sent consecutively without any delay
- Time to parse and optionally decrypt/authenticate the messages can be much more than a character time on the USART RX line
- This is the reason why the RX buffer (hiRespBuf[]) can contain more than a message as shown on the diagram below
  - When RX buffer is empty, the first message is naturally stored at the beginning of the buffer

– After the reception of the 5 first bytes of a new message (to compute the message size), it is checked that the previously parsed messages have freed enough space at the beginning of the RX buffer to copy this new message to the beginning of the RX buffer

–This is not the case at time A, as parsing of first message is not completed.

–This is the case at time B as third message is small enough to be stored at the place of first and second messages that have already been parsed.

Figure 6. RX buffer management



4.5.2 Callback invocation

The callback invocation for confirmation and indication is made when receiving a valid response with the Callback ID and the payload.

The validity of the message is checked in HIF_AppParseCnfIndMsg() function.

The callback invocation for error is made when the message received is not valid (CRC error, bad length, bad preamble…).

All the above is implemented in HIF_AppParseCnfIndMsg() function in hif_g3_common.c file.

```
case X_HI_NEW_RX_MSG:
 {
   uint8_t errorStatus = HIF_ERROR_NONE;

    errorStatus = HIF_ParseMsg(1, msg->payload[0]);
    if (errorStatus == HIF_ERROR_NONE)
     {
       hifControl_t *ptr;
       ptr = (hifControl_t *)(&hifControl[hiVars.status.cmdId]);
       if (ptr->cnfCallBackId != HIF_CNF_NONE)
       {
         hiVars.cnfCallback(ptr->cnfCallBackId);
       }
       if (ptr->indCallBackId != HIF_IND_NONE)
       {
         hiVars.indCallback(ptr->indCallBackId);
       }
       if ((ptr->cnfCallBackId == HIF_CNF_NONE) &&
            (ptr->indCallBackId == HIF_IND_NONE))
       {
         HIF_CloseRequest(1);
       }
     }

     else
     {
        HIF_HandleError(1, errorStatus);
     }
     HIF_ParseMsgComplete(msg->payload[0]);
     HIF_DebugPrint(0, hiVars.status.len, hiVars.status.tag);
   }
   break;
```

4.5.3 Driver API HIF_XXX_Name request

Each driver API HIF_XXX_NameRequest() fills the message payload, calculates the payload length and calls a generic function HIF_Send_Request() which:

- Calculates the CRC
- Builds the complete message including header, length, CRC
- Sends the complete message to the UART handler.

```
Void HIF_ADP_ADPMNetworkJoinRequest(ADP_AdpmNetworkJoinRequest_t *ptr)
{
  uint16_t index = HIF_MSG_PAYLOAD;

  index = HIF_Fill_16(index, ptr->PANId);
  index = HIF_Fill_16(index, ptr->LBAAddress);

  HIF_Send_Request(HIF_ADPM_NTWJOIN_REQ, index - HIF_MSG_PAYLOAD, 0xF0)
}
```

4.5.4 Driver API HIF_XXX_Name confirm and indication

Each driver API HIF_XXX_NameConfirm()/HIF_XXX_NameIndication() extracts the message payload and fills the structure associated to the API.

```
void HIF_ADP_ADPMNetworkJoinConfirm(ADP_AdpmNetworkJoinConfirm_t *ptr)
{
  uint16_t index = HIF_MSG_PAYLOAD_RX;

  index =  HIF_Extract_08(index, &ptr->Status);
  index =  HIF_Extract_16(index, &ptr->NetworkAddress);
  index =  HIF_Extract_16(index, &ptr->PANId);
}
```

4.6 Secure Host Interface

4.6.1 Principle

Depending on ST8500 OTP settings, the messages exchanged on the Host Interface may be ciphered.

Please refer to [*1*] for details.

For ciphered request messages, the following is performed:

- Encrypt payload
- Add authentication tag
- Compute CRC on header + payload + tag.

For ciphered confirm/indication messages, the following is performed:

- Check CRC
- Decrypt payload
- Check authentication tag.

Encryption/decryption is done in software using the STM32 crypto library.

4.6.2 Performances

The following measurements were done on the STM32F437:

- Core Cortex M4
- Frequency 100 MHz

- Crypto lib optimized for size.

Table 6. Encryption/Decryption time on secure HI

Payload size	Message transmission duration on UART at 115kbps	Encryption time	Decryption time
16 bytes	3.1 ms	0.26 ms	0.26 ms
1000 bytes	88.5 ms	2.68 ms	2.72 ms

4.6.3 Crypto lib activation

The STM32 crypto library is not delivered with the ST8500 STM32 firmware package. It must be requested on the ST website separately.

To add it to the ST8500 STM32 project the following must be done:

- Unzip the "STM32CubeExpansion_Crypto_V3.1.0.zip" lib in "stm32_cube" directory
- Include the crypto library in the project build:
  – Middlewares/ST/STM32_Cryptographic/Lib/STM32CryptographicV3.0.0_CM4_IAR.a
- Uncomment ASK_CRYPTO_TO_ST definition in hif_secure_drv.c.

5 Porting to customer environment

The functions below are used by HI DRV to get various services. They are all located in the same file hi_os_wrapper.c.

5.1 RTOS related functions

```
void HI_HandleNewMsg(void)
```

This function is used by the HI DRV to trigger from the USART interrupt service routine (potentially out of FreeRTOS scope) another interrupt that has access to FreeRTOS services (like sending message).

```
void HI_SendMessage(taskId_t task, msgId_t id, u32 data1,
u32 data2, u8 fromIsr)
```

This function is used by the HI DRV to send a new message to task "task".

```
void HI_ReceiveMessage(msgId_t *id, u32 *data1, u32 *data2)
```

This function is used by the HI DRV to receive a message that is to be stored in (*id, *data1, *data2).

```
void HI_StartTimer(void)
```

This function is used by the HI DRV to start a system timer.

```
void HI_StopTimer(void)
```

This function is used by the HI DRV to stop a system timer.

```
void HI_ChangeTimer(u32 time)
```

This function is used by the HI DRV to change a system timer.

“time” is the new value for the timer. The unit is 10 ms in our implementation example.

```
void HI_DownloadComplete(void)
```

This function is used by the HI DRV when the FW download operation is completed. This can be used by the application to switch to PLC operations.

5.2 USART related functions

```
void HI_UsartChangeBaudRate(u32 baudrate)
```

This function is used by the HI DRV to change USART speed.

```
bool HI_UsartStatusFlag(u32 flag)
```

This function is used by the HI DRV to know if USART RX register is full or if USART TX register is empty.

“flag” indicates which side to check (RX or TX) and can be one of the values below:

```
#define HI_USART_FLAG_RX          0x00000020
#define HI_USART_FLAG_TX          0x00000040
```

```
void HI_UsartClearFlag(u32 flag)
```

This function is used by the HI DRV to clear USART TX flag.

```
u8 HI_UsartGetData(void)
```

This function is used by the HI DRV to read last received byte on USART RX.

```
void HI_UsartPutData(u8 data)
```

This function is used by the HI DRV to send a byte on USART TX.

5.3 Flash related functions

```
void HI_ImageRead(uint8_t *ptr, uint8_t imageType, uint32_t offset,
uint16_t size)
```

This function is used by the HI DRV to read “size” bytes of an image at offset “offset”. The image can be read from an attached serial Flash or from the embedded Flash.

The read data is copied into STM32 ram at a location pointed by “*ptr”.

“imageType” is one of the definitions below to read data from either the RTE or PE image.

```
#define HI_IMG_TYPE_PE            0
#define HI_IMG_TYPE_RTE           1
```

5.4 Debug related functions

```
void HI_Printf(const char *str, u32 data, u32 filter)
```

This function is used by the HI DRV to print out debug information.

The debug information is composed of a string and a data.

“filter” helps selecting the debug prints. It can be any of the definitions below.

```
#define HI_TRC_DOWNLOAD          0x00000001
#define HI_TRC_DL_REQRSP         0x00000002
#define HI_TRC_GENERIC           0x00000004
#define HI_TRC_GEN_REQRSP        0x00000008
```

6 Bibliography

1. ST-G3-PLC_Solutions Application Note
2. STarCOM Boot User Manual

7 Revision history

Table 7. Revision history

Date	Revision	Changes
30-May-2019	1	Initial release.