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

1. **Introduction** ........................................... 6  
   1.1 Document overview .................................. 6  
   1.2 List of abbreviations .................................. 6  

2. **Host interface introduction** .............................. 7  
   2.1 Hardware interface overview ............................ 7  
   2.2 Software interface overview ............................ 7  
   2.3 Modes of operation .................................... 8  
   2.4 ST8500 boot sequence and messaging ................. 9  

3. **Firmware Download mode** ................................. 11  
   3.1 Overview ............................................... 11  
   3.2 Description ............................................. 12  

4. **Host Interface mode** .................................... 14  
   4.1 Overview ............................................... 14  
   4.2 Driver API naming rules ................................ 14  
   4.3 Call back functions .................................... 14  
   4.4 List of Host Interface APIs ............................. 15  
   4.5 Driver implementation .................................. 18  
      4.5.1 Buffer management .................................. 18  
      4.5.2 Call back invocation ................................ 19  
      4.5.3 Driver API HIF_XXX_Name request ................. 20  
      4.5.4 Driver API HIF_XXX_Name confirm and indication 21  
   4.6 Secure Host Interface .................................. 21  
      4.6.1 Principle .......................................... 21  
      4.6.2 Performances ....................................... 21  
      4.6.3 Crypto lib activation ................................ 22  

5. **Porting to customer environment** ......................... 23  
   5.1 RTOS related functions ................................ 23  
   5.2 USART related functions ............................... 24  
   5.3 Flash related functions ................................ 24
5.4 Debug related functions ........................................... 25

6 Bibliography ............................................................. 26

7 Revision history .......................................................... 27
List of tables

Table 1. Request Message Format (default mode) ...................................................... 7
Table 2. Confirm/Indication Message Format (default mode). ...................................... 7
Table 3. Request Message Format (secure mode) ....................................................... 8
Table 4. Confirm/Indication Message Format (secure mode). ...................................... 8
Table 5. List of Host Interface APIs ........................................................................... 15
Table 6. Encryption/Decryption time on secure HI ...................................................... 22
Table 7. Revision history. .......................................................................................... 27
List of figures

Figure 1. Host interface principle .................................................. 7
Figure 2. Boot sequence exchange (unsecure case) ................................ 9
Figure 3. Boot sequence exchange (secure case) ................................ 10
Figure 4. RTE and PE images download sequence ............................. 11
Figure 5. RTE and PE images download state machine ....................... 13
Figure 6. RX buffer management .................................................. 19
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

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

![Host interface principle](image)

Messages have the following format:

**Table 1. Request message format (default mode)**

<table>
<thead>
<tr>
<th>Synchro</th>
<th>Control</th>
<th>State</th>
<th>Message</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x16</td>
<td>0x16</td>
<td>CMD</td>
<td>LEN</td>
<td>CMD</td>
</tr>
<tr>
<td>2 bytes</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Synchro</th>
<th>Control</th>
<th>State</th>
<th>Error</th>
<th>Message</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x16</td>
<td>0x16</td>
<td>CMD</td>
<td>LEN</td>
<td>EC</td>
<td>CMD</td>
</tr>
<tr>
<td>2 bytes</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>4 bytes</td>
<td>LEN-1 bytes</td>
</tr>
</tbody>
</table>
Table 3. Request message format (secure mode)

<table>
<thead>
<tr>
<th>Synchro</th>
<th>Control</th>
<th>Security</th>
<th>Message</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 x16</td>
<td>0 x16</td>
<td>CMD ID</td>
<td>LEN</td>
<td>SEC MODE</td>
</tr>
<tr>
<td>2 bytes</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

Header | Ciphertext | Tag

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

<table>
<thead>
<tr>
<th>Synchro</th>
<th>Control</th>
<th>Security</th>
<th>Error</th>
<th>Message</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 x16</td>
<td>0 x16</td>
<td>CMD ID</td>
<td>LEN</td>
<td>SEC MODE</td>
<td>COUNTER</td>
</tr>
<tr>
<td>2 bytes</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>4 bytes</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

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:
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 0000000 00000000 0965
```

In the case of secure Host Interface (secure mode 1), the sequence is described below:
Figure 3. Boot sequence exchange (secure case)

```
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 0002A29BFDF5E22694137F9C801F58F843200
        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
```
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

![Flowchart diagram](image)

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.

```c
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:

```c
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:

```c
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).
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)
- 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:
In the current implementation example, the registration is invoked in HI_StartTasks from hi_Tasks.c.

### 4.4 List of Host Interface APIs

<table>
<thead>
<tr>
<th>API name</th>
<th>Parameter structure name</th>
<th>CallBack Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIF_HI_HwResetRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_HI_HwResetConfirm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_HI_ModeSetRequest</td>
<td>uint8_t *</td>
<td>HIF_CNF_MODE</td>
</tr>
<tr>
<td>HIF_HI_ModeSetConfirm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_HI_ModeGetRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_HI_ModeGetConfirm</td>
<td>uint8_t *</td>
<td>HIF_CNF_MODEGET</td>
</tr>
<tr>
<td>HIF_HI_BaudrateSetRequest</td>
<td>uint32_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_BaudrateSetConfirm</td>
<td>uint32_t *</td>
<td>HI_CNF_BAUDRATE</td>
</tr>
<tr>
<td>HIF_HI_TestSetRequest</td>
<td>hi_testsetreq_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_TestGetRequest</td>
<td>hi_testgetreq_t *</td>
<td>HI_CNF_TESTGET</td>
</tr>
<tr>
<td>HIF_HI_SflashRequest</td>
<td>hi_sflashreq_t *</td>
<td>HI_CNF_SFLASH</td>
</tr>
<tr>
<td>HIF_HI_SflashConfirm</td>
<td>hi_sflashreq_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_NvmRequest</td>
<td>hi_nvm_req_t *</td>
<td>HI_CNF_NVRAM</td>
</tr>
<tr>
<td>HIF_HI_NvmConfirm</td>
<td>hi_nvm_req_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_TraceRequest</td>
<td>Hi_TraceRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_TraceConfirm</td>
<td>Hi_TraceConfirm_t *</td>
<td>HIF_CNF_TRACE</td>
</tr>
<tr>
<td>HIF_HI_RdbgRequest</td>
<td>hi_rdbg_req_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_RdbgConfirm</td>
<td>hi_rdbg_req_t *</td>
<td>HI_CNF_RDBG</td>
</tr>
<tr>
<td>HIF_HI_SecInitRequest</td>
<td>hi_SecInitRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_SecInitConfirm</td>
<td>hi_SecInitConfirm_t *</td>
<td>HIF_CNF_SECINIT</td>
</tr>
<tr>
<td>HIF_HI_OtpRequest</td>
<td>hi_otp_req_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_HI_OtpConfirm</td>
<td>hi_otp_conf_t *</td>
<td>HIF_CNF_OTP</td>
</tr>
<tr>
<td>HIF_G3LIB_GetRequest</td>
<td>G3_LIB_GetAttributeRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_G3LIB_GetConfirm</td>
<td>G3_LIB_GetAttributeConfirm_t *</td>
<td>HI_CNF_G3LIBGET</td>
</tr>
<tr>
<td>HIF_G3LIB_MultipleGetRequest</td>
<td>G3_LIB_GetAttributeRequest_t *</td>
<td></td>
</tr>
</tbody>
</table>

```c
HIF_RegisterCallBack(HI_indCallback, HI_cnfCallback, HI_errCallback);
```
### Table 5. List of Host Interface APIs (continued)

<table>
<thead>
<tr>
<th>API name</th>
<th>Parameter structure name</th>
<th>CallBack Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIF_G3LIB_MultipleGetConfirm</td>
<td>uint8_t *</td>
<td>HIF_CNF_G3LIBMULTIGET</td>
</tr>
<tr>
<td>HIF_G3LIB_SetRequest</td>
<td>G3_LIB_SetAttributeRequest_t *</td>
<td>HIF_CNF_G3LIBSET</td>
</tr>
<tr>
<td>HIF_G3LIB_SetConfirm</td>
<td>G3_LIB_SetAttributeConfirm_t *</td>
<td>HIF_CNF_G3LIBSET</td>
</tr>
<tr>
<td>HIF_G3LIB_SWResetRequest</td>
<td>G3_LIB_SWResetRequest_t *</td>
<td>HIF_CNF_G3LIBSWRESET</td>
</tr>
<tr>
<td>HIF_G3LIB_SWResetConfirm</td>
<td>G3_LIB_SWResetConfirm_t *</td>
<td>HIF_CNF_G3LIBSWRESET</td>
</tr>
<tr>
<td>HIF_G3LIB_TestModeRequest</td>
<td>G3_LIB_TestModeRequest_t</td>
<td>HIF_CNF_G3LIBTESTMODE</td>
</tr>
<tr>
<td>HIF_G3LIB_TestModeConfirm</td>
<td>G3_LIB_TestModeConfirm_t</td>
<td>HIF_CNF_G3LIBTESTMODE</td>
</tr>
<tr>
<td>HIF_G3LIB_EventIndication</td>
<td>G3_LIB_EventIndication_t</td>
<td>HIF_IND_G3LIB_EVT</td>
</tr>
<tr>
<td>HIF_PHY_PdDataRequest</td>
<td>PHY_PdDataRequest_t *</td>
<td>HIF_CNF_PDDATA</td>
</tr>
<tr>
<td>HIF_PHY_PdDataConfirm</td>
<td>PHY_PdDataConfirm_t *</td>
<td>HIF_CNF_PDDATA</td>
</tr>
<tr>
<td>HIF_PHY_PdTrxRequest</td>
<td>uint8_t *</td>
<td>HIF_CNF_PDTRX</td>
</tr>
<tr>
<td>HIF_PHY_PdTrxConfirm</td>
<td>uint8_t *</td>
<td>HIF_CNF_PDTRX</td>
</tr>
<tr>
<td>HIF_PHY_PdCsRequest</td>
<td>uint8_t *</td>
<td>HIF_CNF_PDCS</td>
</tr>
<tr>
<td>HIF_MAC_MCPSDataRequest</td>
<td>MAC_McpsDataReq_t *</td>
<td>HIF_CNF_MACDATA</td>
</tr>
<tr>
<td>HIF_MAC_MCPSDataConfirm</td>
<td>MAC_McpsDataConfirm_t *</td>
<td>HIF_CNF_MACDATA</td>
</tr>
<tr>
<td>HIF_MAC_MCPSDataIndication</td>
<td>MAC_McpsDataIndication_t *</td>
<td>HIF_IND_MACDATA</td>
</tr>
<tr>
<td>HIF_MAC_MLMEBeaconNotifIndication</td>
<td>MAC_MlmBeaconNotif_t *</td>
<td>HIF_IND_MACBEACON</td>
</tr>
<tr>
<td>HIF_MAC_MLMEResetRequest</td>
<td>MAC_MlmResetReq_t *</td>
<td>HIF_CNF_MACRESET</td>
</tr>
<tr>
<td>HIF_MAC_MLMEResetConfirm</td>
<td>MAC_MlmResetConfirm_t *</td>
<td>HIF_CNF_MACRESET</td>
</tr>
<tr>
<td>HIF_MAC_MLMEScanRequest</td>
<td>MAC_MlmScanReq_t *</td>
<td>HIF_CNF_MACSCAN</td>
</tr>
<tr>
<td>HIF_MAC_MLMEScanConfirm</td>
<td>MAC_MlmScanConfirm_t *</td>
<td>HIF_CNF_MACSCAN</td>
</tr>
<tr>
<td>HIF_MAC_MLMEStartRequest</td>
<td>MAC_MlmStartReq_t *</td>
<td>HIF_CNF_MACSTART</td>
</tr>
<tr>
<td>HIF_MAC_MLMEStartConfirm</td>
<td>MAC_MlmStartConfirm_t *</td>
<td>HIF_CNF_MACSTART</td>
</tr>
<tr>
<td>HIF_ADP_ADPDDataRequest</td>
<td>ADP_AdpdDataRequest_t *</td>
<td>HIF_CNF_ADPDATA</td>
</tr>
<tr>
<td>HIF_ADP_ADPDDataConfirm</td>
<td>ADP_AdpdDataConfirm_t *</td>
<td>HIF_CNF_ADPDATA</td>
</tr>
<tr>
<td>HIF_ADP_ADPDDataIndication</td>
<td>ADP_AdpdDataIndication_t *</td>
<td>HIF_IND_ADPDATA</td>
</tr>
<tr>
<td>HIF_ADP_ADPDDiscoveryRequest</td>
<td>ADP_AdpdDiscoveryRequest_t *</td>
<td>HIF_CNF_ADPDISCO</td>
</tr>
<tr>
<td>HIF_ADP_ADPDDiscoveryConfirm</td>
<td>ADP_AdpdDiscoveryConfirm_t *</td>
<td>HIF_CNF_ADPDISCO</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkStartRequest</td>
<td>ADP_AdpmNetworkStartRequest_t *</td>
<td>HIF_CNF_ADPDSTART</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkStartConfirm</td>
<td>ADP_AdpmNetworkStartConfirm_t *</td>
<td>HIF_CNF_ADPDSTART</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkJoinRequest</td>
<td>ADP_AdpmNetworkJoinRequest_t *</td>
<td>HIF_CNF_ADPDJOIN</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkJoinConfirm</td>
<td>ADP_AdpmNetworkJoinConfirm_t *</td>
<td>HIF_CNF_ADPDJOIN</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkLeaveRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>API name</td>
<td>Parameter structure name</td>
<td>CallBack Id</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkLeaveConfirm</td>
<td>ADP_AdpmNetworkLeaveConfirm_t *</td>
<td>HIF_CNF_ADPLEAVE</td>
</tr>
<tr>
<td>HIF_ADP_ADPMNetworkLeaveIndication</td>
<td></td>
<td>HIF_IND_ADPLEAVE</td>
</tr>
<tr>
<td>HIF_ADP_ADPMResetRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_ADP_ADPMResetConfirm</td>
<td>ADP_AdpmResetConfirm_t *</td>
<td>HIF_CNF_ADPRESET</td>
</tr>
<tr>
<td>HIF_ADP_ADPMRouteDiscoveryRequest</td>
<td>ADP_AdpmRouteDiscoveryRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_ADP_ADPMRouteDiscoveryConfirm</td>
<td>ADP_AdpmRouteDiscoveryConfirm_t *</td>
<td>HIF_CNF_ADPROUTE</td>
</tr>
<tr>
<td>HIF_ADP_ADPMPathDiscoveryRequest</td>
<td>ADP_AdpmPathDiscoveryRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_ADP_ADPMPathDiscoveryConfirm</td>
<td></td>
<td>HIF_CNF_ADPPATH</td>
</tr>
<tr>
<td>HIF_ADP_ADPMMLbpRequest</td>
<td>ADP_AdpmMLbpRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_ADP_ADPMMLbpConfirm</td>
<td>ADP_AdpmMLbpConfirm_t *</td>
<td>HIF_CNF_ADPLBP</td>
</tr>
<tr>
<td>HIF_ADP_ADPMLbpIndication</td>
<td>ADP_AdpmLbpIndication_t *</td>
<td>HIF_IND_ADPLBP</td>
</tr>
<tr>
<td>HIF_ADP_ADPMRouteDeleteRequest</td>
<td>ADP_AdpmRouteDeleteRequest_t</td>
<td></td>
</tr>
<tr>
<td>HIF_ADP_ADPMRouteDeleteConfirm</td>
<td>ADP_AdpmRouteDeleteConfirm_t</td>
<td>HIF_CNF_ROUTEDEL</td>
</tr>
<tr>
<td>HIF_BOOT_ServerStartRequest</td>
<td>BOOT_ServerStartRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_BOOT_ServerStartConfirm</td>
<td>BOOT_ServerStartConfirm_t *</td>
<td>HIF_CNFBOOT_SERVER_START</td>
</tr>
<tr>
<td>HIF_BOOT_ServerStopRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_BOOT_ServerStopConfirm</td>
<td>BOOT_ServerStopConfirm_t *</td>
<td>HIF_CNFBOOT_SERVER_STOP</td>
</tr>
<tr>
<td>HIF_BOOT_ServerLeaveIndication</td>
<td>BOOT_ServerLeaveIndication_t *</td>
<td>HIF_INDBOOT_SERVER_LEAVE</td>
</tr>
<tr>
<td>HIF_BOOT_ServerKickRequest</td>
<td>BOOT_ServerKickRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_BOOT_ServerKickConfirm</td>
<td>BOOT_ServerKickConfirm_t *</td>
<td>HIF_CNFBOOT_SERVER_KICK</td>
</tr>
<tr>
<td>HIF_BOOT_ServerJoinIndication</td>
<td>BOOT_ServerJoinIndication_t *</td>
<td>HIF_INDBOOT_SERVER_JOIN</td>
</tr>
<tr>
<td>HIF_BOOT_DeviceStartRequest</td>
<td>BOOT_DeviceStartRequest_t *</td>
<td></td>
</tr>
<tr>
<td>HIF_BOOT_DeviceStartConfirm</td>
<td>BOOT_DeviceStartConfirm_t *</td>
<td>HIF_CNFBOOT_DEVICE_START</td>
</tr>
<tr>
<td>HIF_BOOT_DeviceLeaveRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIF_BOOT_DeviceLeaveConfirm</td>
<td>BOOT_DeviceLeaveConfirm_t *</td>
<td>HIF_CNFBOOT_DEVICE_LEAVE</td>
</tr>
<tr>
<td>HIF_BOOT_DeviceLeaveIndication</td>
<td></td>
<td>HIF_INDBOOT_DEVICE_LEAVE</td>
</tr>
<tr>
<td>HIF_BOOT_DevicePANSortRequest</td>
<td>BOOT_DevicePANSortRequest_t</td>
<td>HIF_INDBOOT_DEVICE_PANSORT</td>
</tr>
<tr>
<td>HIF_BOOT_DevicePANSortRequest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5. List of Host Interface APIs (continued)

<table>
<thead>
<tr>
<th>API name</th>
<th>Parameter structure name</th>
<th>CallBack Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIF_BOOT_DevicePANSortConfirm</td>
<td>BOOT_DevicePANSortConfirm_t</td>
<td>HIF_CNF_BOOT_DEVICE_PANSORT</td>
</tr>
<tr>
<td>HIF_BOOT_ServerGetPSKIndication</td>
<td>BOOT_ServerGetPSKIndication_t</td>
<td>HIF_IND_BOOT_SERVER_GETPSK</td>
</tr>
<tr>
<td>HIF_BOOT_ServerSetPSKRequest</td>
<td>BOOT_ServerSetPSKRequest_t</td>
<td></td>
</tr>
<tr>
<td>HIF_BOOT_ServerSetPSKConfirm</td>
<td>BOOT_ServerSetPSKConfirm_t</td>
<td></td>
</tr>
<tr>
<td>HIF_IP_UdpDataRequest</td>
<td>IP_G3UdpDataRequest_t *</td>
<td>HIF_CNF_UDPDATA</td>
</tr>
<tr>
<td>HIF_IP_UdpDataConfirm</td>
<td>IP_DataConfirm_t *</td>
<td>HIF_CNF_UDPDATA</td>
</tr>
<tr>
<td>HIF_IP_UdpDataIndication</td>
<td>IP_DataIndication_t *</td>
<td>HIF_IND_UDPDATA</td>
</tr>
<tr>
<td>HIF_IP_UdpConnSetRequest</td>
<td>IP_UdpConnSetRequest_t *</td>
<td>HIF_CNF_UDPCONN_SET</td>
</tr>
<tr>
<td>HIF_IP_UdpConnSetConfirm</td>
<td>IP_UdpConnSetConfirm_t *</td>
<td>HIF_CNF_UDPCONN_SET</td>
</tr>
<tr>
<td>HIF_IP_UdpConnGetRequest</td>
<td>IP_UdpConnGetRequest_t *</td>
<td>HIF_CNF_UDPCONN_GET</td>
</tr>
<tr>
<td>HIF_IP_IcmpEchoRequest</td>
<td>IP_G3IcmpDataRequest_t *</td>
<td>HIF_CNF_ICMP_ECHO</td>
</tr>
<tr>
<td>HIF_IP_IcmpEchoConfirm</td>
<td>IP_DataConfirm_t *</td>
<td>HIF_CNF_ICMP_ECHO</td>
</tr>
<tr>
<td>HIF_IP_IcmpEchoIndication</td>
<td>IP_DataIndication_t *</td>
<td>HIF_IND_ICMP_ECHO</td>
</tr>
</tbody>
</table>

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 with no 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.
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.
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_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);
}

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 [7] 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

<table>
<thead>
<tr>
<th>Payload size</th>
<th>Message transmission duration on UART at 115kbps</th>
<th>Encryption time</th>
<th>Decryption time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>3.1 ms</td>
<td>0.26 ms</td>
<td>0.26 ms</td>
</tr>
<tr>
<td>1000 bytes</td>
<td>88.5 ms</td>
<td>2.68 ms</td>
<td>2.72 ms</td>
</tr>
</tbody>
</table>

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

```c
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

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

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

```c
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:

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

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

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

```c
u8 HI_UsartGetData(void)
```

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

```c
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

```c
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.

```c
#define HI_IMG_TYPE_PE 0
#define HI_IMG_TYPE_RTE 1
```
5.4 Debug related functions

```c
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.

```c
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

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-May-2019</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved