Introduction

This user manual describes ST8500 bootloader functionalities and operations to be done for a correct device boot and the firmware images download.

The following sections are included in this document

- **Section 2 on page 4** covers the bootloader and the firmware image format
- **Section 3 on page 7** covers the boot from an external SPI Flash
- **Section 4 on page 10** covers the boot from an external host processor and the boot communication protocol
- **Section 5 on page 20** explains the usage of the image generator tool

This document does not include the description of the ST8500 device itself that can be found on www.st.com. Detailed information on the various PLC protocols running on the ST8500 and the device tools can be found in specific software packages, separately delivered under Software license agreement by contacting your local ST sales office.
## Contents

1. **Document conventions** .................................................. 3

2. **Description and bootloader** ............................................. 4
   2.1 Boot modes ................................................................. 4
   2.2 Image format ............................................................. 5
   2.3 User security configuration ........................................... 6

3. **Boot from external SPI Flash** .......................................... 7
   3.1 Small configuration ..................................................... 8
   3.2 Large configuration .................................................... 9

4. **Boot from host interfaces** .............................................. 10
   4.1 SPI interface ............................................................ 10
   4.2 UART interface .......................................................... 11
   4.3 Command message flows .............................................. 12
       4.3.1 Setup of UART configuration and MIB request flow .......... 13
       4.3.2 Write image flow ................................................ 13
   4.4 Command message description ..................................... 15
       4.4.1 COM Init frame .................................................. 15
       4.4.2 MIB GET frame .................................................. 16
       4.4.3 IMG Init Frame ................................................ 17
       4.4.4 IMG write frame ............................................... 18
       4.4.5 IMG start frame .............................................. 18
       4.4.6 SYS reset frame ............................................... 19

5. **Image generator tool** .................................................. 20

6. **Revision history** ....................................................... 22
1 Document conventions

Table 1. List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex</td>
<td>Cortex®-M4</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>NVM</td>
<td>Non-volatile memory</td>
</tr>
<tr>
<td>OTP</td>
<td>One Time Programming</td>
</tr>
<tr>
<td>OTW</td>
<td>One time writable</td>
</tr>
<tr>
<td>PE</td>
<td>Protocol engine</td>
</tr>
<tr>
<td>RAM</td>
<td>Random access memory</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-only memory</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial peripheral interface</td>
</tr>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver/transmitter</td>
</tr>
</tbody>
</table>
2 Description and bootloader

The ST8500 is a power line communication modem device supporting several narrow-band PLC standards. It is based on two different computing subsystems: an ARM® Cortex-M4 (called also PE) dedicated to the non-real time firmware and a real time engine (RTE) dedicated to the real time firmware.

The ST8500 boot procedure is managed by the Cortex-M4 core. The bootloader code is stored in an embedded ROM during the silicon manufacturing process. The Cortex-M4 core starts executing the instructions present in the embedded ROM after the power on reset (POR) or after a system reset.

The basic function implemented by the boot process is to decrypt, validate, load images from various boot sources into ST8500 RAMs and to transfer control to the target cores running the images. Once the control is transferred to the loaded code, it is no more possible to run the bootloader code without performing a system reset.

2.1 Boot modes

The bootloader is able to support up to 8 different boot modes. The configuration is done through the 3 BOOTx pins. Only 4 boot modes are currently used as shown in Table 2.

<table>
<thead>
<tr>
<th>Boot2</th>
<th>Boot1</th>
<th>Boot0</th>
<th>Boot ID</th>
<th>Boot mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x0</td>
<td>Boot from UART host interface</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0x1</td>
<td>Boot from SPI host interface</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0x2</td>
<td>Boot from SPI external Flash (large configuration)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0x3</td>
<td>Boot from SPI external Flash (small configuration)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0x4</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0x5</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0x6</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0x7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

The bootloader mandates to download the RTE image first and then the Cortex-M4 image. If only the Cortex-M4 image is provided, the RTE is not booted. This means that the RTE is not enabled and that it is not possible to load any further RTE image without performing a complete reset to deal with a new boot procedure.

The ST8500 is able to load new or different images only after the POR or a complete reset.
2.2 Image format

The bootloader is able to load the firmware into the proper core only if it is provided in an image format. The image contains a clear-text header, one or more section headers and a payload that is the binary firmware. The format of the image is the same for all the boot modes.

The firmware can be divided into different sections of payload to be stored at different RAM addresses. The whole firmware payload can be either clear-text or encrypted.

The image payload and all the header data following the authentication tag are authenticated.

The format of the firmware image is detailed in Table 3.

<table>
<thead>
<tr>
<th>Clear-text Firmware Image Header</th>
<th>Data type</th>
<th>Values/meaning</th>
<th>Data Bytes offset</th>
<th>Data Bytes size</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGE_TYPE</td>
<td>FW_PE_IMAGE = 0x0000F1F1</td>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>FW_RTE_IMAGE = 0x0000A3A3</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VALIDITY STATUS</td>
<td>Validity of the image can change without impact on the authentication tag. Managed by the application during the FW upgrade on the external Flash (0x2 and 0x3 BOOT MODES only).</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>APPLICATION PARAMETERS</td>
<td>Application specific data (32 bytes), can change without impact on the authentication code.</td>
<td>8</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>FIRMWARE SIZE</td>
<td>The size of the firmware payload in bytes.</td>
<td>40</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>SECURITY MODE</td>
<td>Specifies the image encryption mode, the encryption B seed (PE only) or no encryption.</td>
<td>44</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>AES IV</td>
<td>The AES initialization vector</td>
<td>48</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>AUTHENTICATION TAG</td>
<td>The authentication tag, generated during AES authentication/encryption.</td>
<td>64</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>FIRMWARE ENTRY ADDRESS</td>
<td>Address where the execution of the code contained in this image shall start.</td>
<td>80</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF SECTIONS</td>
<td>The number of sections (n) of the firmware image. The number of sections has to be greater than zero and minor than 25.</td>
<td>84</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Clear-text Firmware Image n-th Section Header</td>
<td>FIRMWARE DESTINATION ADDRESS</td>
<td>The internal load address of the section image</td>
<td>88 + 8 * (n - 1)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>FIRMWARE n-th SECTION SIZE</td>
<td>The size of the n-th section in firmware payload</td>
<td>92 + 8 * (n - 1)</td>
<td>4</td>
</tr>
<tr>
<td>HEADER_PADDING</td>
<td>Optional</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>FIRMWARE PAYLOAD</td>
<td>Variable length firmware payload (one or more sections), either clear-text or AES encrypted</td>
<td>-</td>
<td>Variable length</td>
<td></td>
</tr>
<tr>
<td>PAYLOAD_PADDING</td>
<td>-</td>
<td></td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
The “number of sections (n)” valid range is 1 to 24.
The image must be padded (up to 8 bytes) in order to keep the authenticated header size multiple of an AES block (16 bytes).

2.3 **User security configuration**

The ST8500 allows the user to define security rules that the bootloader applies at the startup by a stored configuration into an OTP memory area. Refer to the ST8500 datasheet for details on the OTP data structure and functions. The OTP area includes the “PE key” (Cortex-M4F image decryption key) used for the (optional) decryption and authentication of the PE firmware image. The image is encrypted and authenticated using the AES-GCM algorithm.
3 Boot from external SPI Flash

Booting from the SPI Flash requires having at least one firmware image already present in the external Flash. The SPI1 peripheral is used to control the SPI Flash at a nominal frequency of 12 MHz.

Two kinds of Flash size are supported
- Small size Flash
- Large size Flash

The firmware images are searched at fixed offsets according to the memory organization as showed in Figure 1 and Figure 2.

The external SPI Flash can contain two versions of each image to support upgrading and rollback to the previous firmware version in case of firmware malfunctioning. The selection of the images to be loaded is based on the “Validity” field inside the image header. The meaning of the field “Validity” is detailed in Table 4.

<table>
<thead>
<tr>
<th>Validity binary value (4 bits)</th>
<th>Image validity status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110b0</td>
<td>Updated image, not yet functionally validated from the application code</td>
</tr>
<tr>
<td>1100b0</td>
<td>First choice image, validated from the application code</td>
</tr>
<tr>
<td>1000b0</td>
<td>Second choice image, validated but superseded by a newer one</td>
</tr>
<tr>
<td>0000b0</td>
<td>Functionally invalid image, invalidated from the application code</td>
</tr>
<tr>
<td>All other values</td>
<td>Invalid value, corrupted image - header check will fail</td>
</tr>
</tbody>
</table>

In particular, if two images satisfy the integrity checks based on the header and authentication code, the selection process is defined by following rules
- A new, not validated image needs validation (or invalidation), so it will get the priority
- If no new images exist, a first-choice image is selected
- If no new or first-choice image exists, a second-choice image is selected
- If no new, first-choice or second-choice image exists, a functionally invalid image is selected and an error condition is flagged
- If the validity flag value is the same for both images, then the image #1 (PE#1 and/or RTE#1) has the preference

According to the coding of Table 4, the image with the higher valid value of the validity flag is selected. If only one image satisfies the integrity checks based on the header and authentication code, this image is loaded regardless to the value of the field “Validity”.

If no image satisfies the integrity checks based on the authentication code, the boot does not load the firmware for the related computing subsystem regardless to the value of the field “Validity” and enters in the “Boot from UART HOST interface” mode. The image validity is managed by the application code and not by the bootloader code.
3.1 Small configuration

In the small configuration, the minimum SPI Flash size is 512 kB. The boot loader uses only the first 512 kB ignoring the other part of the SPI Flash if bigger. *Figure 1* shows the Flash organization for the small size SPI Flash.

*Figure 1. Small SPI Flash organization*
3.2 Large configuration

If the large configuration is selected, the minimum SPI Flash size must be 1 MB. The boot loader uses only the first 1 MB ignoring the other part of the SPI Flash if bigger that is left to application purposes. The large configuration is the preferred one. Figure 2 shows the Flash organization for the large size SPI Flash.

Figure 2. Large SPI Flash organization
4 Boot from host interfaces

Boot modes from the host interface require an external host controller connected to the ST8500 via UART or SPI. The host controller is able to download the firmware images according to the protocol specified in the following paragraphs. The host controller acts as a master, while the ST8500 is a slave.

The frame format and the protocol commands are common to both UART and SPI interfaces. The frame format specified in Table 5 is the same for both requests and responses. All fields are in the little endian format.

It is always the master (host controller) that initiates the communication by sending a request. The ST8500 answers with a response using the same command ID (CMD_ID).

<table>
<thead>
<tr>
<th>Synchro field</th>
<th>Control field</th>
<th>State field</th>
<th>Message payload field</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x16</td>
<td>0x16</td>
<td>CMD_ID</td>
<td>MSG_LEN</td>
<td>CRC</td>
</tr>
<tr>
<td>2 Bytes</td>
<td>1 Byte</td>
<td>2 Bytes</td>
<td>1 Byte</td>
<td>N Bytes</td>
</tr>
</tbody>
</table>

4.1 SPI interface

The SPI0 peripheral is used as the interface with the following configuration

- CPHA = 0 (the first clock transition is the first data capture edge)
- CPOL = 0 (clock to 0 when idle)
- Maximum frequency is 15 MHz

Since the SPI is a synchronous protocol and the ST8500 is the slave of this communication, the response management needs and extra signal to indicate to the host that a response is available. The ST8500 signals the availability of the response setting GPIORsp low that is mapped on GPIO0_2.

The master sends the request commands message on the MOSI line ignoring the received bytes on the MISO line. When the request has been processed, the ST8500 signals the availability of the response by setting the GPIORsp (GPIO0_2) line in a low state. At this point the master sends dummy bytes on the MOSI line in order to receive the response. When all the response bytes are transmitted, the ST8500 changes the GPIORsp (GPIO0_2) line status back to the original status. At this point, the master may issue a new command - a response transmission cycle may start again.
Figure 3 shows a complete request - response sequence.

**Figure 3. SPI messages sequence**

4.2 UART interface

The UART0 is used as an interface with the following initial configuration
- 9600 baud
- 1 start bit
- 1 stop bit
- No parity
- No flow control

Since the UART is an asynchronous interface, both the host controller and the ST8500 may send messages without any additional hardware handshake. The UART configuration can be modified by a specific command to be sent at the beginning.
4.3 **Command message flows**

The following paragraphs describe the host interface messages and sequence flows. In particular, the command IDs and the STATE values are described respectively in Table 6 and Table 7. The common fields like SYNCH, MODE and CRC are not reported.

### Table 6. Command ID description

<table>
<thead>
<tr>
<th>CMD_ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>COM_Init To change the UART configuration</td>
</tr>
<tr>
<td>0x02</td>
<td>SYS_Reset To perform a system reset</td>
</tr>
</tbody>
</table>

### Image download commands

<table>
<thead>
<tr>
<th>CMD_ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11</td>
<td>IMG_Init To signal the start of an image download</td>
</tr>
<tr>
<td>0x12</td>
<td>IMG_Write To download an image block</td>
</tr>
<tr>
<td>0x13</td>
<td>IMG_Start To signal the end of the image and the request of the start the related computing subsystem</td>
</tr>
</tbody>
</table>

### Management information base

<table>
<thead>
<tr>
<th>CMD_ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21</td>
<td>MIB_Get To request the value of a specific MIB</td>
</tr>
</tbody>
</table>

### Table 7. State description

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 0x00 0x00 0x00</td>
<td>ACK - request received and correctly performed</td>
</tr>
<tr>
<td>0x01 0x00 0x00 0x00</td>
<td>NACK - request received with errors, ask for retransmission</td>
</tr>
<tr>
<td>0x02 0x00 0x00 0x00</td>
<td>Abort - request correctly received, but the related operation can’t be performed</td>
</tr>
</tbody>
</table>
4.3.1 Setup of UART configuration and MIB request flow

The following flowchart in Figure 4 shows the starting communication from the host to the ST8500.

The first step shows the UART configuration setup (i.e. in order to increase the UART speed), then the flowchart shows the optional loop for multiple MIB get requests.

**Figure 4. Setup of UART configuration and MIB request flow**

4.3.2 Write image flow

The following flowchart in Figure 5 shows the write image procedure.

On the first step the host sends a request to the ST8500 to start the download. This message must include only the plaintext header of the binary image. After that, a loop is used to write the whole image payload. At the end, the host controller sends the start message asking the core to boot the loaded image.

The procedure is the same for both RTE and PE images, and it must be performed first for the RTE image download and then for the PE image download. If only the PE procedure is done, the boot process ends without loading the RTE image.
Figure 5. Write image flow

If a valid response is not received before MAX_TIMER or if response STATE is "Abort", then the procedure is failed.

```
loop: while (STATE == NACK)
    request: COM_ID=IMG_Init(payload=header)
    response: COM_ID=IMG_Init
        [STATE=ACK]
        [STATE=NACK]
        [STATE=Abort]

loop: for i=0 to n
    request: COM_ID=IMG_Write(payload=image_block(x))
    alt
        [STATE=ACK]
        response: COM_ID=IMG_Write
            [STATE=ACK]
        [STATE=NACK]
        request: COM_ID=IMG_Write
            [STATE=NACK]
    [STATE=NACK]
    request: COM_ID=IMG_Write
        [STATE=ACK]
        [STATE=NACK]
        [STATE=Abort]
```
4.4 Command message description

4.4.1 COM Init frame

This command is used to initialize the communication between the host and the ST8500 specifying a new UART configuration: baud rate, parity bit and stop bit. The possible responses are ACK, NACK and Abort.

Table 8. COM_Init.Request command frame

<table>
<thead>
<tr>
<th>Byte</th>
<th>Label</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CMD_ID</td>
<td>See Table 6</td>
<td>Command ID</td>
</tr>
<tr>
<td>3</td>
<td>MSG_LEN</td>
<td>6</td>
<td>Length of message payload</td>
</tr>
<tr>
<td>10</td>
<td>Baudrate</td>
<td>See Table 9</td>
<td>Baudrate</td>
</tr>
<tr>
<td>14</td>
<td>Parity</td>
<td>0…2</td>
<td>Parity: 0x00 = None, 0x01 = Odd, 0x02 = Even</td>
</tr>
<tr>
<td>15</td>
<td>Stop</td>
<td>0…2</td>
<td>Stop bits: 0x1= 1 bit, 0x2 = 2 bits</td>
</tr>
</tbody>
</table>

Table 9. Baudrate configuration

<table>
<thead>
<tr>
<th>COM_Init PAYLOAD - baudrate</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
</tr>
<tr>
<td>14400</td>
</tr>
<tr>
<td>19200</td>
</tr>
<tr>
<td>38400</td>
</tr>
<tr>
<td>57600</td>
</tr>
<tr>
<td>115200</td>
</tr>
<tr>
<td>230400</td>
</tr>
<tr>
<td>921600</td>
</tr>
</tbody>
</table>

Table 10. COM_Init.Response frame

<table>
<thead>
<tr>
<th>Byte</th>
<th>Label</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CMD_ID</td>
<td>See Table 6</td>
<td>Command ID</td>
</tr>
<tr>
<td>3</td>
<td>MSG_LEN</td>
<td>0</td>
<td>Length of message payload</td>
</tr>
<tr>
<td>6</td>
<td>STATE</td>
<td>See Table 9</td>
<td>STATE: 0x00 = ACK, 0x01 = NACK, 0x02 = Abort</td>
</tr>
</tbody>
</table>
4.4.2 MIB GET frame

This command is used to get the value of a specific ST8500 MIB (see Table 13). The possible responses are ACK with the asked value, NACK and Abort.

<table>
<thead>
<tr>
<th>Table 11. MIB_GET.Request command frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byte</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3 ... 4</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 12. MIB_GET.Response frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byte</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3 ... 4</td>
</tr>
<tr>
<td>6 ... 9</td>
</tr>
<tr>
<td>10 ... 10 + (MSG_LEN-1)</td>
</tr>
</tbody>
</table>

The available MIB are listed in Table 13.

<table>
<thead>
<tr>
<th>Table 13. MIB list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID</strong></td>
</tr>
<tr>
<td>0x00</td>
</tr>
<tr>
<td>0x01</td>
</tr>
<tr>
<td>0x02</td>
</tr>
<tr>
<td>0x03</td>
</tr>
<tr>
<td>0x04</td>
</tr>
<tr>
<td>0x05</td>
</tr>
<tr>
<td>0x06</td>
</tr>
<tr>
<td>0x07</td>
</tr>
<tr>
<td>0x08</td>
</tr>
<tr>
<td>0x09</td>
</tr>
<tr>
<td>0x0A</td>
</tr>
</tbody>
</table>
RTE/PE boot status bits are listed in Table 14.

Table 14. RTE/PE status bits description

<table>
<thead>
<tr>
<th>Bit</th>
<th>Status value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Image transfer initiated</td>
</tr>
<tr>
<td>1</td>
<td>Image transfer completed</td>
</tr>
<tr>
<td>2</td>
<td>Image transfer successful</td>
</tr>
<tr>
<td>3</td>
<td>RTE load error</td>
</tr>
<tr>
<td>4</td>
<td>PE load error</td>
</tr>
<tr>
<td>5</td>
<td>RTE IMG_Init error</td>
</tr>
<tr>
<td>6</td>
<td>PE IMG_Init error</td>
</tr>
<tr>
<td>7</td>
<td>RTE IMG_Write error</td>
</tr>
<tr>
<td>8</td>
<td>PE IMG_Write error</td>
</tr>
<tr>
<td>9</td>
<td>No valid PE image error</td>
</tr>
<tr>
<td>10</td>
<td>No valid RTE image error</td>
</tr>
<tr>
<td>11</td>
<td>OTP read error</td>
</tr>
<tr>
<td>12</td>
<td>FW image size error</td>
</tr>
<tr>
<td>13</td>
<td>Image start address error</td>
</tr>
<tr>
<td>14</td>
<td>RTE start timeout error</td>
</tr>
<tr>
<td>15</td>
<td>Functional invalid image error</td>
</tr>
<tr>
<td>16</td>
<td>Loaded image number (only for SPI Flash boot modes)</td>
</tr>
</tbody>
</table>

4.4.3 IMG init frame

This command is used to start the loading of an image on the ST8500. The message payload includes the plaintext header of the binary image. The possible responses are ACK, NACK and Abort.

Table 15. IMG_Init.Request command frame

<table>
<thead>
<tr>
<th>IMG_Init.Request command frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byte</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3 … 4</td>
</tr>
<tr>
<td>10 … 10 + MSG_LEN</td>
</tr>
</tbody>
</table>
4.4.4 IMG write frame

This command is used to load an image payload block on the ST8500. The block size must be a multiple of 16 bytes (a single AES block). The maximum allowed size is 1 kB. The possible responses are ACK, NACK and Abort.

<table>
<thead>
<tr>
<th>IMG_Init.Response Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byte</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3 … 4</td>
</tr>
<tr>
<td>6 … 9</td>
</tr>
</tbody>
</table>

4.4.5 IMG start frame

This command is used to send to the ST8500 the restart signal after the image loading. The possible responses are ACK, NACK and Abort.

<table>
<thead>
<tr>
<th>IMG_Write.Request command frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byte</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3 … 4</td>
</tr>
<tr>
<td>10 … 10 + MSG_LEN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMG_Write.Response frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Byte</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3 … 4</td>
</tr>
<tr>
<td>6 … 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMG_Start.Request frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Label</strong></td>
</tr>
<tr>
<td>CMD_ID</td>
</tr>
<tr>
<td>MSG_LEN</td>
</tr>
</tbody>
</table>
4.4.6 SYS reset frame

This command is used to send a reset request to the ST8500. Possible responses are ACK, NACK and Abort.

Table 20. IMG_Start.Response frame

<table>
<thead>
<tr>
<th>Label</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_ID</td>
<td>See Table 6</td>
<td>Command ID</td>
</tr>
<tr>
<td>MSG_LEN</td>
<td>0</td>
<td>Length of message payload</td>
</tr>
<tr>
<td>STATE</td>
<td>ACK, NACK, Abort</td>
<td>STATE: 0x00 = ACK, 0x01 = NACK, 0x02 = Abort</td>
</tr>
</tbody>
</table>

Table 21. SYS_Reset.Request command frame

<table>
<thead>
<tr>
<th>Byte</th>
<th>Label</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CMD_ID</td>
<td>See Table 6</td>
<td>Command ID</td>
</tr>
<tr>
<td>3 ... 4</td>
<td>MSG_LEN</td>
<td>0</td>
<td>Length of message payload</td>
</tr>
</tbody>
</table>

Table 22. SYS_Reset.Response frame

<table>
<thead>
<tr>
<th>Byte</th>
<th>Label</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CMD_ID</td>
<td>See Table 6</td>
<td>Command ID</td>
</tr>
<tr>
<td>3 ... 4</td>
<td>MSG_LEN</td>
<td>0</td>
<td>Length of message payload</td>
</tr>
<tr>
<td>6 ... 9</td>
<td>STATE</td>
<td>See Table 7</td>
<td>STATE: 0x00 = ACK, 0x01 = NACK, 0x02 = Abort</td>
</tr>
</tbody>
</table>
5 Image generator tool

The image generator is a tool that receives as inputs a configuration file and some sections binaries (at least one section) in order to produce as an output either a PE or an RTE firmware image composed by the specified sections, arranged and encrypted according to the configuration file. The sections sizes must be aligned to 128 bits, and the image generator tool provides for this alignment by adding pad bits if needed.

The configuration file is an *.xml file, formed by the following tags

```xml
<header image_type="processor_type" validity="validity_value" entry="entry_address" fw_encryption_status="encryption_type" fw_encryption_seed="seed_value">
  <application_data value0="app_data[0]" value1="app_data[1]" value2="app_data[2]" value3="app_data[3]" value4="app_data[4]" value5="app_data[5]" value6="app_data[6]" value7="app_data[7]"/>
  <aes_iv value0="iv[0]" value1="iv[1]" value2="iv[2]" value3="iv[3]">
    <aes_key value0="key[0]" value1="key[1]" value2="key[2]" value3="key[3]" value4="key[4]" value5="key[5]" value6="key[6]" value7="key[7]"/>
  </aes_iv>
  <section address="sect_0_address" filename="section_0_filename">
  </section>
  <section address="sect_1_address" filename="section_1_filename">
  </section>
  <section address="sect_2_address" filename="section_2_filename">
  </section>
  <section address="sect_3_address" filename="section_3_filename">
  </section>
  ...
  <section address="sect_n_address" filename="section_n_filename">
  </section>
</header>
```
Table 23 specifies the meaning and the allowed values for every tag fields.

### Table 23. Image generator fields description

<table>
<thead>
<tr>
<th>Configuration field</th>
<th>Meaning</th>
<th>Allowed values</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>image_type</strong></td>
<td>Specifies the image type to be generated</td>
<td>cpu, rte</td>
<td></td>
</tr>
<tr>
<td><strong>validity</strong></td>
<td>Validity value</td>
<td>Unsigned 32-bit integer greater than 0</td>
<td></td>
</tr>
<tr>
<td><strong>entry</strong></td>
<td>Entry address from where the execution of the fw image shall start</td>
<td>32-bit address</td>
<td>The entry address must be 64-bit aligned (see sect_n_address)</td>
</tr>
<tr>
<td><strong>fw_encryption_status</strong></td>
<td>Specifies if the payload is encrypted</td>
<td>cleartext, aes_256</td>
<td>An authentication tag is always generated for the whole image, also if cleartext.</td>
</tr>
<tr>
<td><strong>fw_encryption_seed</strong></td>
<td>Reserved</td>
<td>0</td>
<td>Must be set to “0”</td>
</tr>
<tr>
<td><strong>Aes_iv value i, 0≤i≤3</strong></td>
<td>AES-GCM Initial vector's i-th word, input for the encryption algorithm</td>
<td>32-bit integer</td>
<td></td>
</tr>
<tr>
<td><strong>Aes_key value i, 0≤i≤7</strong></td>
<td>AES-GCM key's i-th word, input for the encryption algorithm</td>
<td>32-bit integer</td>
<td></td>
</tr>
<tr>
<td><strong>Section address</strong></td>
<td>Memory address where the first section's word is written</td>
<td>32-bit address</td>
<td>The section address must be 64-bit aligned</td>
</tr>
<tr>
<td><strong>Section filename</strong></td>
<td>Filename containing the input hexadecimal bytes of the section</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
6 Revision history

Table 24. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>13-Dec-2017</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
