IN602 Bootloader Software Guide

Department: Software

Prepared by: K.Yu

Project Name: Hummingbird

September 22, 22

Control Number: 123-4567

# Revision History

| **Revision History** | **Description** | **Prepared By** | **Date** |
| --- | --- | --- | --- |
| Ver1.0 | 1st draft | K. Yu | 2018-09-27 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Contents

[Revision History 2](#_Toc526685561)

[Contents 3](#_Toc526685562)

[1 Introduction 5](#_Toc526685563)

[2 Flow Chart 5](#_Toc526685564)

[3 Boot Rom 9](#_Toc526685565)

[3.1 Reset 9](#_Toc526685566)

[3.2 Rom CRC checking 9](#_Toc526685567)

[3.3 Efuse 9](#_Toc526685568)

[3.4 Clock configuration 11](#_Toc526685569)

[3.5 SPI flash 11](#_Toc526685570)

[3.6 Boot options 11](#_Toc526685571)

[3.7 Secure boot 11](#_Toc526685572)

[3.8 Ram secure boot 12](#_Toc526685573)

[4 Boot Ram 13](#_Toc526685574)

[4.1 Entry 13](#_Toc526685575)

[4.2 SPI flash interface functions 13](#_Toc526685576)

[4.3 Cold boot 13](#_Toc526685577)

[4.4 Warm boot 13](#_Toc526685578)

[4.5 Run time SPI flash program 13](#_Toc526685579)

[4.6 Boot ram configurations 14](#_Toc526685580)

[5 Host control 15](#_Toc526685581)

[5.1 Host control interface 15](#_Toc526685582)

[5.2 Host control interface initialization 15](#_Toc526685583)

[5.3 Host control secure programming 15](#_Toc526685584)

[6 Commands 17](#_Toc526685585)

[6.1 CMD\_READY 17](#_Toc526685586)

[6.2 CMD\_ACK 18](#_Toc526685587)

[6.3 CMD\_NAK 18](#_Toc526685588)

[6.4 CMD\_GO 19](#_Toc526685589)

[6.5 CMD\_READ\_REG 20](#_Toc526685590)

[6.6 CMD\_READ\_REG\_RSP 21](#_Toc526685591)

[6.7 CMD\_WRITE\_REG 22](#_Toc526685592)

[6.8 CMD\_EFUSE\_READ 23](#_Toc526685593)

[6.9 CMD\_EFUSE\_READ\_RSP 24](#_Toc526685594)

[6.10 CMD\_EFUSE\_WRITE 25](#_Toc526685595)

[6.11 CMD\_SECURE\_FLASH\_KEY\_XCHG 26](#_Toc526685596)

[6.12 CMD\_SECURE\_FLASH\_KEY\_XCHG\_RSP 27](#_Toc526685597)

[6.13 CMD\_SECURE\_FLASH\_HPB\_DNLD 28](#_Toc526685598)

[6.14 CMD\_SECURE\_FLASH\_NPB\_DNLD 30](#_Toc526685599)

[6.15 CMD\_DNLD 32](#_Toc526685600)

[6.16 CMD\_FLASH\_ERASE 33](#_Toc526685601)

[6.17 CMD\_BOOTRAM\_ENABLE 34](#_Toc526685602)

[6.18 CMD\_AUTH\_REQ 35](#_Toc526685603)

[6.19 CMD\_AUTH\_RSP 35](#_Toc526685604)

[6.20 CMD\_AUTH\_VERIFY 36](#_Toc526685605)

[6.21 CMD\_READ\_ROM\_VER 37](#_Toc526685606)

[6.22 CMD\_READ\_ROM\_VER\_RSP 37](#_Toc526685607)

[6.23 CMD\_CHANGE\_UART\_BAUD\_RATE 39](#_Toc526685608)

# Introduction

This document is to detail the IN602 bootloader software implementation. IN602 bootloader software consists of implementation in the ROM and RAM. The ROM part of the bootloader software controls the CPU boot, security, and communication with the host. The RAM part of the bootloader controls the external SPI flash, customer’s configuration data, and the branching to the final customer’s application. The RAM part of the bootloader software is open and can be modified by the customer to fit his or her needs.

# Flow Chart

Reset

A

Read customer’s configuration data from efuse

Host control

No

Yes

Ok

Efuse ready?

Not Ok

Cold boot

ROM CRC Check?

Warm boot

Branch to RAM

Cold/Warm boot ?

B

Load boot ram from SPI flash

Turn SPI flash power on

Set SPI flash pin mux

Host control

No

Yes

SPI flash present?

XO startup check

No

Copy it

Yes

Is XO calibration result exist?

No

Turn on RTC

Yes

Is RTC present?

A

D

No

Yes

C

Yes

Check GPIO

Boot option?

Time out?

No

No

No

Yes

Host control

Receive host command?

Yes

Wait for host

Boot option?

No

Host control

Retrieve SPI flash APIs from boot ram

Yes

Boot ram valid?

B

END

Branch to RAM

Decrypted to RAM

Read encrypted RAM code from SPI flash

Yes

No

Encrypted RAM code in SPI flash?

Yes

Host control

No

Signature match?

Yes

Calculate signature

D

No

Secure boot?

Yes

Host control

GPIO set?

C

No

# Boot Rom

IN602 bootloader rom:

Code size: 32K bytes;

Address: 0x00000000

Data size: 8K bytes;

Address: 0x20E000

(Note: Data ram (8K bytes) can be reused once boot finished.)

## Reset

Upon power on, the ARM CPU will start execution from the boot rom “reset” function. The “reset” function checks the bit 31 of the “always on domain” register 0x11A4 to determine if the current boot is due to system reset (cold) or resume(warm). If it is cold boot, the “reset” function will branch to the boot rom “main” function. If it is warm boot, the “reset” function will program the ARM CPU’s SP register from the “always on domain” register 0x11A0 and branch to the boot ram code to continue the resuming process.

## Rom CRC checking

IN602 has 256K bytes of rom that includes both the boot and BLE rom code. To check rom image’s integrity, the boot rom code will call HW’s rom crc32 checking function in every cold boot. If the check fails, the boot rom will stop booting and branch to command loop.

## Efuse

IN602 has 3 banks of efuse that are used for persistent information storage. Each bank has 16 of 32 bits word that can be programmed. Bank 0 is reserved by Inplay. Bank1 and 2 are reserved for customer.

Currently,

Bank 0 is assigned as:

1. word 0: miscellaneous configuration,
2. word 1 ~ 9: unique ID,
3. word 14 ~ 15: RF characterization

Bank 1 is assigned as:

1. word 0 ~ 15: ECC public key for security.

Bank 2 is assigned as:

1. word 0: customer’s miscellaneous configuration.

The following tables list the efuse bits that had been defined by Inplay and are used by the bootloader software. The usage of each bit will be explained in the following sections.

|  |  |  |
| --- | --- | --- |
| bit | field | Description |

|  |  |  |
| --- | --- | --- |
| 1 | external flash present | 0: flash not present; 1: flash present |
| 13 | qspi\_0\_pin\_mux [0] | QSPI 0 for pin mux |
| 14 | qspi\_0\_pin\_mux [1] |
| 15 | qspi\_0\_pin\_mux [2] |
| 16 | qspi\_2\_pin\_mux [0] | QSPI 2 for pin mux |
| 17 | qspi\_2\_pin\_mux [1] |
| 18 | qspi\_2\_pin\_mux [2] |
| 19 | qspi\_3\_pin\_mux [0] | QSPI 3 for pin mux |
| 20 | qspi\_3\_pin\_mux [1] |
| 21 | qspi\_3\_pin\_mux [2] |
| 22 | qspi\_4\_pin\_mux [0] | QSPI 4 for pin mux |
| 23 | qspi\_4\_pin\_mux [1] |
| 24 | qspi\_4\_pin\_mux [2] |
| 25 | qspi\_5\_pin\_mux [0] | QSPI 5 for pin mux |
| 26 | qspi\_5\_pin\_mux [1] |
| 27 | qspi\_5\_pin\_mux [2] |
| 29 | Tvsl [0] | Define the waiting time between power up of the QSPI flash and the first CS low. The waiting time (uint is us) has 8 choices which are 30 (corresponding tvsl[2:0]=0x0), 100 (0x1),300 (0x2), 1000 (0x3), 2000 (0x4), 4000 (0x5),6000 |
| 30 | Tvsl [1] |
| 31 | Tvsl [2] |

|  |  |  |
| --- | --- | --- |
| bit | Field | Description |
| 2 | Flash encryption enable | 0: not; 1: yes |
| 3 | Secure boot enables | 0: not; 1 yes |
| 5 | the presence of a preferred interface | 0: not present; 1: present |
| 6 | Host interface option [0] | 0: Uart0, 1: Uart1, 2: SPI, 3: Uart0 backup,  4: Uart1 backup |
| 7 | Host interface option [1] |
| 8 | Host interface option [2] |
| 9 | Ram secure boot enable | 0: not; 1: yes |
| 10 | Host authentication enable | 0: not; 1: yes |
| 11 | SPI flash read speed [0] | 0: 2M, 1: 4M, 2: 8M, 3: 16M |
| 12 | SPI flash read speed [1] |  |
| 16 | RTC is installed | 0: RTC not installed; 1: RTC installed |
| 17 | XO Calibration result present | 0: not present; 1: present |

## Clock configuration

During cold boot, IN602 HW will set the clock to 64Mhz RC clock. Since RC clock is not accurate, boot rom needs to calibrate clock by using external XO. If efuse bit 17 of bank 2 word 0 is set, boot rom will just copy the previous calibration result. If not set, boot rom will start calibration. Once the calibration finished, boot rom will switch the clock to run at 64Mhz XO clock.

## SPI flash

During cold boot, boot rom checks bit 1 of efuse bank 0 word 0 to see if external SPI flash exists or not. If no external SPI flash exists, boot rom will branch to host command loop to wait for host commands. If there is external SPI flash, boot rom will configure SPI’s IO pins by reading efuse bit 13 ~ 27 of bank 0 word 0. After configuration of the IO pins, boot rom will turn on external SPI flash power and wait for the time specified by the efuse bit 29 ~ 30 of bank 0 word 0.

After SPI flash power on finished, boot rom will issue SPI flash read command, with speed specified by the efuse bit 11~12 of band 2 word 0, to read 4K bytes from flash address 0x300000 to RAM address 0x200000. Boot rom then checks if boot ram’s “magic word (0xA72E0129)” is presented at the last 4 bytes of the 4K bytes. If “magic word” is verified, boot rom will continue boot process. Otherwise, it will branch to host command loop to wait for host’s command.

## Boot options

IN602 bootloader software provides two options for customer to stop the cold boot process and branch to the host command loop. The two options are:

1. Wait for host command:
   1. Boot rom will wait for a specific amount of time before continue booting. If boot rom detect host command during the wait, it will branch to host command loop and stop booting,
   2. The wait time can be configured in the boot ram,
   3. The wait time unit is 100 milliseconds.
2. GPIO:
   1. Boot rom will check a pre-configured GPIO pins. If the pin condition matched, boot rom will branch to host command loop and stop booting,
   2. The GPIO pins can be configured in the boot ram.

## Secure boot

This is to protect customer’s application from tampering. This option can be turn on by setting bit 3 of the efuse bank 2 word 0 to 1. The secure boot is based on ECDSA. During cold boot, boot rom implements the followings to support ECDSA;

1. Boot rom calculates a hash (SHA-256) value based on provided image’s start address and size (note: this implies the image must be contiguous.). The image started address and size can be configured in the boot ram,
2. Boot rom uses “ECC P-256” curve to calculate a signature from the step 1 hash value. The customer’s provided public key is read from efuse bank 1 word 0 ~ 15. (note: public key byte order should be little endian)
3. Boot rom verifies the calculated signature against the customer’s provided signature. It they are matched, boot rom will continue booting. Otherwise, it will branch to host command loop. The signature can be configured in the boot ram. (note: signature byte order should be little endian)

## Ram secure boot

This is to protect customer’s RAM application from being duplicated. This option can be turn on by setting bit 9 of the efuse bank 2 word 0 to 1. The encryption and decryption are based on the AES CBC algorithm with Inplay proprietary key (256 bits). During cold boot, boot rom implements the followings to support this option:

1. Read the encrypted image from the external SPI flash. The encrypted image SPI flash starting address and size can be configured in the boot ram,
2. Decrypted the encrypted image and write to RAM. The decrypted image RAM starting address can be configured in the boot ram.

# Boot Ram

IN602 bootloader ram:

Code size: 4K bytes;

Load address 0x300000

Execution address: 0x200000

(note: the above addresses are fixed which means customer should avoid these regions.)

## Entry

Boot ram entry address is at 0x200000 and has 4 options. They are:

1. Retrieve SPI flash interface functions, such as erase, program, etc.….
2. Cold boot
3. Warm boot
4. Run time SPI flash program

The “entry” function always runs from the caller’s stack. For 1 and 2, it runs from boot rom stack. For 3, it runs from saved stack. For 4, it runs from caller’s stack.

## SPI flash interface functions

The five interface functions are:

1. Open: this is for SPI flash software preparation,
2. Close: this is for SPI flash software clean up,
3. Erase: this is to erase SPI flash,
4. Prog: this is to program SPI flash,
5. Read: this is to read SPI flash.

## Cold boot

This is a continuation of cold boot from boot rom. Before branching to the final application address, the followings should enable:

1. If bit 2 of efuse bank 2 word 0 is set (flash code protection), then bit 0 of HW register 0x441070C0 should be set to enable HW decryption. Referred to “” for details,
2. SPI flash Quad mode. This depends on each SPI flash vendor,
3. XIP “execution in place” mode enable.

## Warm boot

This is a continuation of warm boot from boot rom. It should also preform cold boot’s step 1, 2 and 3 before branching to the final resume address.

## Run time SPI flash program

To program SPI flash during run time, the XIP must disable. After finished programming, XIP should be re-enabled before return to caller.

## Boot ram configurations

The last 128 bytes of boot ram image are reserved for some boot configurations. They are defined in the table below:

|  |  |  |
| --- | --- | --- |
| Address | Name | Description |
| 0x200F80 | Reserved |  |
| 0x200F84 | Slave SPI interrupt host enable | This is to inform boot rom to use interrupt pin before transmitting. |
| 0x200F88 | Slave SPI interrupt GPIO group number | SPI slave interrupt pin group number. |
| 0x200F8C | Slave SPI interrupt GPIO pin number | SPI slave interrupt pin within group. |
| 0x200F90 | Slave SPI interrupt GPIO level | SPI slave interrupt pin level for interrupt. |
| 0x200F94 | Ram secure boot image flash address | The starting flash address for the encrypted RAM image. |
| 0x200F98 | Ram secure boot image RAM address | The starting RAM address for the decrypted RAM image. |
| 0x200F9C | Ram secure boot image size | The secure RAM image size. |
| 0x200FA0 | Boot options | 0: no option,  1: wait for host,  2: GPIO. |
| 0x200FA4 | Boot option 1’s wait time | Time in 100 milliseconds unit |
| 0x200FA8 | Boot option 2’s GPIO group number |  |
| 0x200FAC | Boot option 2’s GPIO pin number |  |
| 0x200FB0 | Boot option 2’s GPIO level |  |
| 0x200FB4 | Secure boot image flash address | The starting flash address of secure boot image. |
| 0x200FB8 | Secure boot image size | The secure boot image size. |
| 0x200FBC | Secure boot signature starts | The secure boot’s signature first 4 bytes. |
| ………. | Signature | The secure boot’s signature. |
| 0x200FF8 | Secure boot signature end | The secure boot’s signature last 4 bytes. |
| 0x200FFC | Magic word | 0xA72E0129 |

# Host control

IN602 bootloader software implements a simple communication protocol to handle host’s requests. The protocol is always a command, followed by an “ACK” or a “NAK”, and optional command’s response data. The packet format is as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start | command | Data length | data | End |
| 1 byte | 1 byte | 2 bytes | Variable bytes | 1 byte |

The “NAK” response indicates either the command’s operation failed, or the command is illegal.

## Host control interface

The host control’s communication interface is either UART or SPI. Each description is as follows:

UART:

IN602 has two UART controllers and each controller have two sets of GPIO pins. Customer can specify UART interface by programming bits 6~8 of efuse bank 2 word 0. The “uart0”, “uart0 backup”, “uart1”, and “uart1\_backup” are refer to different set of GPIO pins.

The default UART baud rate is 115200. Customer can change UART baud rate during operation by issuing “CMD\_CHANGE\_UART\_BAUD\_RATE” command.

SPI:

IN602 is a SPI slave and Host is a SPI master. Host should provide SPI clock with speed less or equal to 4Mhz. Customer can specify SPI interface by programming bit 6~8 of efuse bank 2 word 0. To provide a better synchronization between SPI master and slave, customer can optionally use an interrupt pin to let SPI slave inform master that it has data to transmit. Customer can specify SPI slave interrupt pin in the boot ram’s configuration data section.

## Host control interface initialization

To establish communication with the host, boot rom will wait on each interface for around 20 milliseconds if bit 5 of efuse bank 2 word 0 is not set. Host should issue “CMD\_READY” every 20 milliseconds and wait for the “CMD\_ACK” from the device.

## Host control secure programming

To prevent customer’s software from being duplicated, IN602 boot rom implements the followings to protect customer’s software:

1. Host authentication: this is to make sure that boot rom is communicated with a legitimate host (prevent illegal download to customer’s device). This is based on ECDSA (curve P-256) and host should aware of the followings:
   1. Host should program its public key into the efuse bank 1 before enabling this feature,
   2. Host set bit 10 of efuse bank2 word 0 to 1 to enable this feature,
   3. Once feature enabled, boot rom will block any “write” commands until host has been authenticated,
   4. Host should have its private key ready for the signature generation,
   5. Host should sign the plain text from boot rom and send the signature to boot rom,
   6. Host should receive an “ACK” if signature match.

Please referred to section 6.18 for commands detail.

1. Host secure transfer: this is to prevent sniffing during downloading of customer’s application. This is based on ECDH (curve P-256) and host should aware of the followings:
   1. Host should use ECC (curve P-256) to generate a public and private key pair,
   2. Host should generate share secret thru ECC modular multiplication,
   3. Host should hash (SHA-256) the shared secret to generate an AES (256 bits) symmetrical key,
   4. Host should encrypt the download data by AES-CBC with the above symmetrical key,
   5. Host should set AES-CBC IV vector first byte to 1.

Please referred to section 6.11 for commands detail.

1. Host HPB (high protection block) download: this is to protect customer’s RAM based application from being duplicated from SPI flash. Boot rom decrypted downloaded data from the secure transfer, in turn, it will re-encrypt the downloaded data by using an IN602’s proprietary AES key before loading them into SPI flash.

Please referred to section 6.13 for command detail.

1. Host NPB (normal protection block) download: this is to protect customer’s flash-based application from being duplicated from SPI flash. The procedure is very similar to the “HPB” download, except the “re-encryption” step is different from the “HPB”. After “NPB” download, customer needs to set bit 2 of efuse bank 2 word 0 to 1 to indicate to the XIP engine that the flash code has been encrypted.

Please referred to section 6.14 for command detail.

# Commands

## CMD\_READY

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x01 | Command code. |
| 0x00 | There is no data, so length is 0. |
| 0x00 |  |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to sync up communication with the device. This command should be the first command issued by the host.

***Message chart:***

Host Device

CMD\_ACK

CMD\_READY

## CMD\_ACK

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x02 | Command code. |
| 0x02 | Length 2 for response data length. |
| 0x00 |  |
| 0xXX | 1st byte of response data length |
| 0xXX | 2nd byte of response data length |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to acknowledge issued command. The “ack” is issued after the issued command operation has completed and the result is good.

## CMD\_NAK

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x03 | Command code. |
| 0x02 | Length 2 for response data length. |
| 0x00 |  |
| 0x00 | 1st byte of response data length |
| 0x00 | 2nd byte of response data length |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to acknowledge issued command. The “nak” is issued after the issued command operation has completed and the result is no good.

## CMD\_GO

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x04 | Command code. |
| 0x00 | There is no data, so length is 0. |
| 0x00 |  |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to branch to boot ram. This command has no “ack” return.

## CMD\_READ\_REG

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x05 | Command code. |
| 0x04 | length is 4 for the register address. |
| 0x00 |  |
| 0xXX | 1st byte of register address. |
| 0xXX | 2nd byte of register address |
| 0xXX | 3rd byte of register address |
| 0xXX | 4th byte of register address |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to read the device’s HW register. If command successful, the device will issue first an “ack” and then the read register value. Otherwise, the device will issue a “nak”.

***Message chart:***

Host Device

CMD\_ACK

CMD\_READ\_REG

CMD\_READ\_REG\_RSP

## CMD\_READ\_REG\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x06 | Command code. |
| 0x04 | length is 4 for the register value. |
| 0x00 |  |
| 0xXX | 1st byte of register value. |
| 0xXX | 2nd byte of register value. |
| 0xXX | 3rd byte of register value. |
| 0xXX | 4th byte of register value. |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to return the value of HW register.

## CMD\_WRITE\_REG

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x07 | Command code. |
| 0x08 | length is 8 for the register address and value. |
| 0x00 |  |
| 0xXX | 1st byte of register address. |
| 0xXX | 2nd byte of register address. |
| 0xXX | 3rd byte of register address. |
| 0xXX | 4th byte of register address. |
| 0xXX | 1st byte of register value. |
| 0xXX | 2nd byte of register value. |
| 0xXX | 3rd byte of register value. |
| 0xXX | 4th byte of register value. |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to write value to the device’s HW register. If command is successful, the device will issue an “ack”. Otherwise, it will issue a “nak”.

***Message chart:***

Host Device

CMD\_ACK

CMD\_WRITE\_REG

## CMD\_EFUSE\_READ

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x08 | Command code. |
| 0x01 | length is 1 for the efuse word index. |
| 0x00 |  |
| 0xXX | Word index (0, 1, 2, …, 15). |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to read efuse data. If command is successful, the device will issue an “ack” and then follow by the efuse data. Otherwise, the device issues a “nak”.

***Message chart:***

Host Device

CMD\_ACK

CMD\_EFUSE\_READ

CMD\_EFUSE\_READ\_RSP

## CMD\_EFUSE\_READ\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x09 | Command code. |
| 0x04 | length is 4 for the efuse data. |
| 0x00 |  |
| 0xXX | 1st byte of efuse data. |
| 0xXX | 2nd byte of efuse data. |
| 0xXX | 3rd byte of efuse data. |
| 0xXX | 4th byte of efuse data. |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to return the efuse data.

## CMD\_EFUSE\_WRITE

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x0A | Command code. |
| 0x05 | length is 5 for the efuse bank, word index, and data. |
| 0x00 |  |
| 0xXX | Word index (0, 1, 2, …, 15). |
| 0xXX | 1st byte of efuse data. |
| 0xXX | 2nd byte of efuse data. |
| 0xXX | 3rd byte of efuse data. |
| 0xXX | 4th byte of efuse data. |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to write data to the device’s efuse. If command is successful, the device will issue an “ack”. Otherwise, it will issue a “nak”.

***Message chart:***

Host Device

CMD\_ACK

CMD\_EFUSE\_WRITE

## CMD\_SECURE\_FLASH\_KEY\_XCHG

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x0B | Command code. |
| 0x40 | length is 64 for the host’s ECDH public key |
| 0x00 |  |
| 0xXX | LSB of the public key. |
| ….. | Public keys… |
| 0xXX | MSB of the public key. |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to request the device to return its public key. This is based on the ECDH key agreement protocol to establish a shared secret. Upon receive this command, the boot rom code will;

1. Generate a 32 bytes random number as private key,
2. Based on curve25519, calculate a 64 bytes public key,
3. Based on curve25519, calculate a 32 bytes shared secret by modular multiplied our private key with host’s public key,
4. Hash (SHA-256) the 32 bytes shared secret to generate an AES-256 symmetric key,
5. Return our public key to the host.

***Message chart:***

Host Device

CMD\_ACK

CMD\_SECURE\_FLASH\_KEY\_XCHG

CMD\_SECURE\_FLASH\_KEY\_XCHG\_RSP

## CMD\_SECURE\_FLASH\_KEY\_XCHG\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x0C | Command code. |
| 0x40 | length is 64 for the device’s ECDH public key |
| 0x00 |  |
| 0xXX | LSB of the public key. |
| ….. | Public keys… |
| 0xXX | MSB of the public key. |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to return its public key.

## CMD\_SECURE\_FLASH\_HPB\_DNLD

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x0D | Command code. |
| 0xXX | length is (2 bytes of block length) + (2 bytes of data length) + (4 bytes of flash address) + (variable data length but must be less than 1K bytes). |
| 0xXX |  |
| 0xXX | 1st byte of Block length (note: this is 16 bytes aligned length) |
| 0xXX | 2nd byte of block length |
| 0xXX | 1st byte of true data length (note: this is length without 16 bytes alignment) |
| 0xXX | 2nd byte of true data length |
| 0xXX | 1st byte of flash address |
| 0xXX | 2nd byte of flash address |
| 0xXX | 3rd byte of flash address |
| 0xXX | 4th byte of flash address |
| 0xXX | 1st byte of encrypted download data |
| ….. | Encrypted download data |
| 0xXX | Last byte of encrypted download data |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to download the encrypted data to the SPI flash. The followings are the limitation:

1. Downloaded flash address must be above 0x301000 since the first 4K bytes is used by the boot ram,
2. Encrypted data will be loaded into RAM during cold boot and the RAM address must be above 0x201000 since, again, the first 4K bytes is used by boot ram,
3. Encryption algorithm is AES CBC with key (256 bits) derived from the “CMD\_SECURE\_FLASH\_KEY\_XCHG” procedure. Since AES algorithm handle 16 bytes of data at a time, for image with size not 16 bytes aligned, the host should pad with 0.
4. Set AES IV first byte to 0x01.

Upon receive this command, the boot rom will decrypt the encrypted data and then re-encrypted with IN602 proprietary AES key before loaded into SPI flash.

***Message chart:***

Host Device

CMD\_ACK

CMD\_SECURE\_FLASH\_HPB\_DNLD

:

:

:

:

CMD\_SECURE\_FLASH\_HPB\_DNLD

CMD\_ACK

## CMD\_SECURE\_FLASH\_NPB\_DNLD

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x0E | Command code. |
| 0xXX | length is (2 bytes of block length) + (2 bytes of data length) + (4 bytes of flash address) + (variable data length but must be less than 1K bytes). |
| 0xXX |  |
| 0xXX | 1st byte of Block length (note: this is 16 bytes aligned length) |
| 0xXX | 2nd byte of block length |
| 0xXX | 1st byte of true data length (note: this is length without 16 bytes alignment) |
| 0xXX | 2nd byte of true data length |
| 0xXX | 1st byte of flash address |
| 0xXX | 2nd byte of flash address |
| 0xXX | 3rd byte of flash address |
| 0xXX | 4th byte of flash address |
| 0xXX | 1st byte of encrypted download data |
| ….. | Encrypted download data |
| 0xXX | Last byte of encrypted download data |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to download the encrypted data to the SPI flash. The followings are the limitation:

1. Downloaded flash address must be above 0x301000 since the first 4K bytes is used by the boot ram,
2. Encryption algorithm is AES CBC with key (256 bits) derived from the “CMD\_SECURE\_FLASH\_KEY\_XCHG” procedure. Since AES algorithm handle 16 bytes of data at a time, for image with size not 16 bytes aligned, the host should pad with 0.
3. Set AES IV first byte to 0x01.

Upon receive this command, the boot rom will decrypt the encrypted data and then re-encrypted with IN602 proprietary algorithm before loaded into SPI flash. The encrypted data in the flash will not be loaded in to RAM and will be executed directly from the SPI flash.

***Message chart:***

Host Device

CMD\_ACK

CMD\_SECURE\_FLASH\_NPB\_DNLD

:

:

:

:

CMD\_SECURE\_FLASH\_NPB\_DNLD

CMD\_ACK

## CMD\_DNLD

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x0F | Command code. |
| 0xXX | length is (2 bytes of data length) + (4 bytes of flash address) + (variable data length but must be less than 1K bytes). |
| 0xXX |  |
| 0xXX | 1st byte of data length |
| 0xXX | 2nd byte of data length |
| 0xXX | 1st byte of flash address |
| 0xXX | 2nd byte of flash address |
| 0xXX | 3rd byte of flash address |
| 0xXX | 4th byte of flash address |
| 0xXX | 1st byte of data |
| ….. | data |
| 0xXX | Last byte of data |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to download the data to the RAM or SPI flash. This command can be used to download the boot ram code into the SPI flash. The sequences are:

1. Download boot ram code into the RAM at address 0x200000,
2. Issue “CMD\_BOOTRAM\_ENABLE” command which will acquire the SPI flash interface functions from boot ram,
3. Issue “CMD\_FLASH\_ERASE” command to erase first 4K bytes of SPI flash,
4. Download boot ram code again by specifying address at 0x300000 which is SPI flash starting address.

Besides the above, this command can also be used for the “no external SPI flash” platform. The sequences are:

1. Download boot ram code into the RAM at address 0x200000. The boot ram code can be modified to just deal with cold boot branch.
2. Download application code into the RAM above the boot ram code,
3. Issue “CMD\_GO” to branch to boot ram which in turn can branch to application code.

## CMD\_FLASH\_ERASE

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x10 | Command code. |
| 0x08 | Flash address and size |
| 0x00 |  |
| 0xXX | 1st byte of the starting flash address to erase |
| 0xXX | 2nd byte of the starting flash address to erase |
| 0xXX | 3rd byte of the starting flash address to erase |
| 0xXX | 4th byte of the starting flash address to erase |
| 0xXX | 1st byte of the erase size |
| 0xXX | 2nd byte of the erase size |
| 0xXX | 3rd byte of the erase size |
| 0xXX | 4th byte of the erase size |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to erase SPI flash sector. Upon receive this command, the boot rom will call into boot ram SPI flash erase interface function to perform erasing.

***Message chart:***

Host Device

CMD\_ACK

CMD\_FLASH\_ERASE

## CMD\_BOOTRAM\_ENABLE

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x11 | Command code. |
| 0x00 | This command has no data |
| 0x00 |  |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to acquire the SPI flash interface functions from boot ram. If successful, the device will return an “ack” to the host. Otherwise, it will return a “nak”. Host can start programming to flash after it receive an “ack” from device.

***Message chart:***

Host Device

CMD\_ACK

CMD\_BOOTRAM\_ENABLE

## CMD\_AUTH\_REQ

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x12 | Command code. |
| 0x00 | This command has no data |
| 0x00 |  |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to request authentication from device. Upon receiving this command, boot rom will generate a 32 bytes plain text to return to host. The host should then sign this plain text by using ECDSA (curve p-256) and send the signature back to the device for verification.

## CMD\_AUTH\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x13 | Command code. |
| 0x20 | 32 bytes of plain text |
| 0x00 |  |
| 0xXX | LSB of plain text |
| …. | Bytes of plain text |
| 0xXX | MSB of plain text |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to return a 32 bytes plain text for host to generate signature.

## CMD\_AUTH\_VERIFY

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x14 | Command code. |
| 0x40 | 64 bytes of signature |
| 0x00 |  |
| 0xXX | LSB of signature |
| …. | Bytes of signature |
| 0xXX | MSB of signature |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to provide a signature for device to verify. If verification successful, the device will return “ack”. Otherwise it will return “nak”.

***Message chart:***

Host Device

CMD\_ACK

CMD\_AUTH\_REQ

CMD\_AUTH\_RSP

CMD\_ACK

CMD\_AUTH\_VERIFY

## CMD\_READ\_ROM\_VER

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x15 | Command code. |
| 0x00 | This command has no data |
| 0x00 |  |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to read boot rom version number.

## CMD\_READ\_ROM\_VER\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x16 | Command code. |
| 0x04 | Rom version length |
| 0x00 |  |
| 0x00 | 1st byte of version number |
| 0x01 | 2nd byte of version number |
| 0x2A | 3rd byte of version number |
| 0x60 | 4th byte of version number |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the device to return current bot rom version number (0x602A0100).

***Message chart:***

Host Device

CMD\_ACK

CMD\_READ\_ROM\_VER

CMD\_READ\_ROM\_VER\_RSP

## CMD\_CHANGE\_UART\_BAUD\_RATE

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x17 | Command code. |
| 0x04 | Uart baud rate length |
| 0x00 |  |
| 0xXX | 1st byte of baud rate |
| 0xXX | 2nd byte of baud rate |
| 0xXX | 3rd byte of baud rate |
| 0xXX | 4th byte of baud rate |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to change the UART interface baud rate. This command has no return; therefore, host should wait at least 20 milliseconds before issuing any new commands. The maximum baud rate that IN602 can support is 2Mbps.

## CMD\_CALC\_HASH

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x18 | Command code. |
| 0x08 | Hash length |
| 0x00 |  |
| 0xXX | 1st byte of address |
| 0xXX | 2nd byte of address |
| 0xXX | 3rd byte of address |
| 0xXX | 4th byte of address |
| 0xXX | 1st byte of size |
| 0xXX | 2nd byte of size |
| 0xXX | 3rd byte of size |
| 0xXX | 4th byte of size |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to calculate hash. If enable flash encryption, bootloader decrypt data on flash. The address and size must be 4bytes aligned. Device return 32 bytes hash in CMD\_CALC\_HASH\_RSP.

***Message chart:***

Host Device

CMD\_ACK

CMD\_CALC\_HASH

CMD\_CALC\_HASH\_RSP

## CMD\_CALC\_HASH\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x19 | Command code. |
| 0x20 | Hash length |
| 0x00 |  |
| 0xXX | LSB of hash |
| …. | Bytes of hash |
| 0xXX | MSB of hash |
| 0x7E | End sync byte. |

## CMD\_CALC\_CRC

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x1A | Command code. |
| 0x0C | CRC length |
| 0x00 |  |
| 0xXX | 1st byte of address |
| 0xXX | 2nd byte of address |
| 0xXX | 3rd byte of address |
| 0xXX | 4th byte of address |
| 0xXX | 1st byte of size |
| 0xXX | 2nd byte of size |
| 0xXX | 3rd byte of size |
| 0xXX | 4th byte of size |
| 0xXX | 1st byte of qspi offset |
| 0xXX | 2nd byte of qspi offset |
| 0xXX | 3rd byte of qspi offset |
| 0xXX | 4th byte of qspi offset |
| 0x7E | End sync byte. |

***Description:***

This command is issued by the host to calculate hash. If enable flash encryption, bootloader decrypt data on flash. The address and size must be 4bytes aligned. Device return 32 bytes hash in CMD\_CALC\_HASH\_RSP.

***Message chart:***

Host Device

CMD\_ACK

CMD\_CALC\_CRC

CMD\_CALC\_CRC\_RSP

## CMD\_CALC\_HASH\_RSP

***Format:***

|  |  |
| --- | --- |
| Byte | Comment |
| 0x7E | Start sync byte. |
| 0x1B | Command code. |
| 0x04 | Hash length |
| 0x00 |  |
| 0xXX | 1st byte of CRC32 |
| 0xXX | 2nd byte of CRC32 |
| 0xXX | 3rd byte of CRC32 |
| 0xXX | 4th byte of CRC32 |
| 0x7E | End sync byte. |