AN2698

Secure UART Bootloader for SAM L11

Introduction

Many modern embedded systems require application image updates to fix errors or support new features. A small piece of code can be added to the main application to provide the ability to download updates, replacing the old firmware of the device. This code is often called as Bootloader, as its role is to load a new program at boot. A Bootloader always resides in the memory to make it possible for the device to be upgraded at any time. Therefore, it must be as small as possible.

Protection of intellectual property plays a key role in this process. Microcontrollers have robust firmware protection mechanisms; however, the firmware is vulnerable to interception during the data transfer from the external source.

One way to solve this problem is to use a secure Bootloader and distribute only encrypted images of the firmware to the public. This document describes the design and operation of a secure Bootloader for the SAM L11 devices, and the encryption algorithm.

Table of Contents

1. Features

The secure UART Bootloader has the following features:

- Secure
- Small size (2 KBytes)
- Uses UART RX and TX pins and a Bootloader Entry pin
- Running out of SRAM allows self-updating
- Simultaneous writing to Flash and next Buffer Reception to speed up the update process
- Application integrity verification
- Source code is available, which can be customized to user requirements

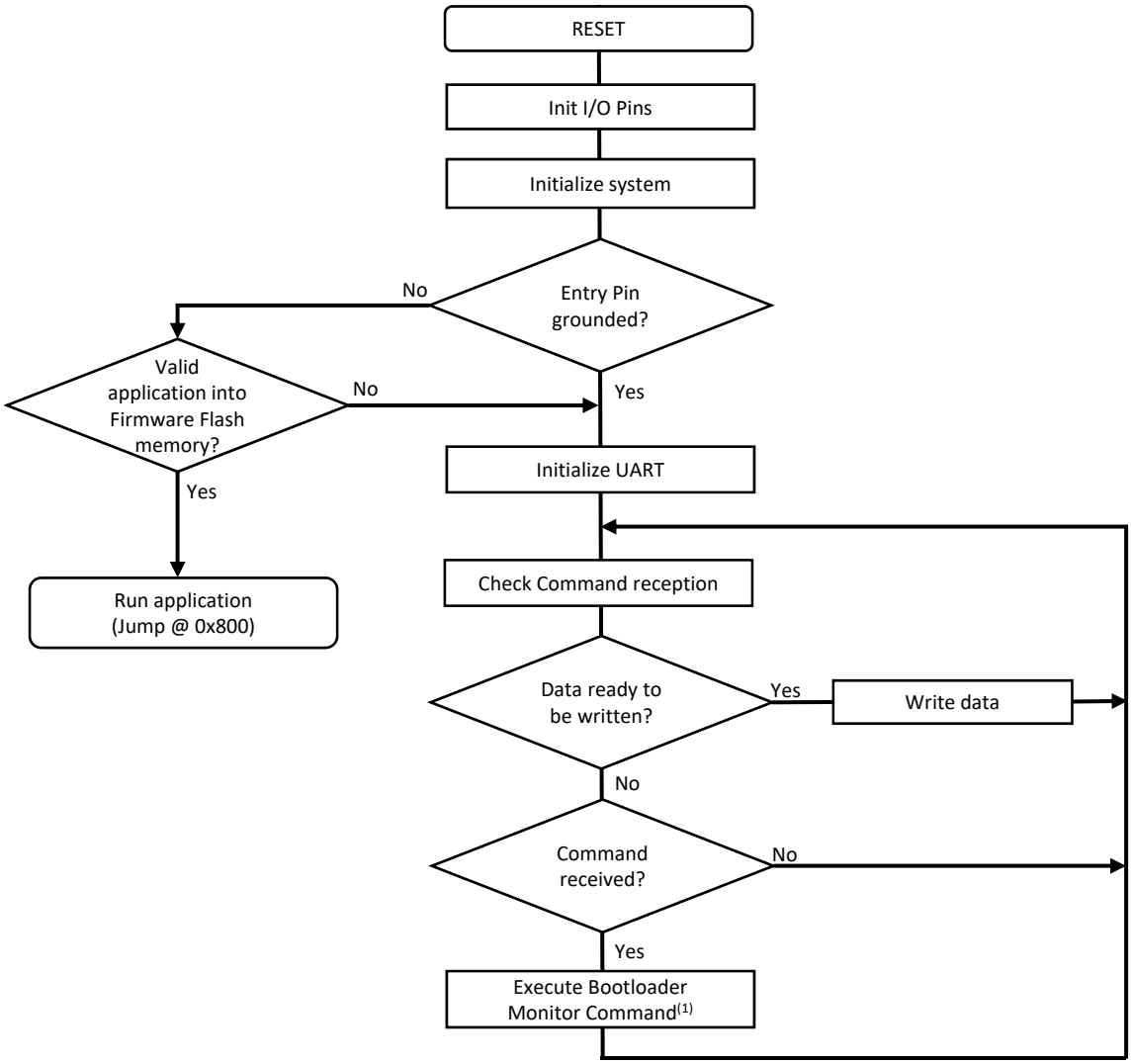
2. Bootloader Implementation

2.1 Bootloader Flow

The startup sequence of the Bootloader is as follows:

- Initialization:
  - I/O Pins initialization
  - Cache is disabled
  - Wait States = 2
  - Performance level = 2
  - 16 MHz clock
  - Security key for AES-128 = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 }
- Entry methods condition check
- Initialization
  - UART initialization
- Bootloader Monitor execution (if one entry method is verified)
  - Application integrity check (SHA-256) using the Verify command

Figure 2-1. UART Bootloader Flowchart



Note:

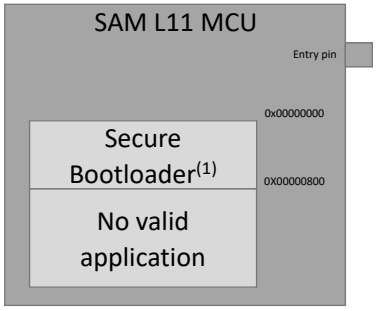
1. The Application image integrity is checked (SHA-256) using the Verify command.

2.2 Method of Entry

The Bootloader can be invoked in two different ways:

1. The Bootloader will run automatically if there is no valid application in the application Flash memory region. Refer to Application Image Configuration for more details.

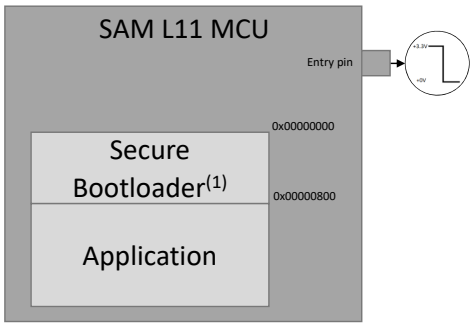
   Figure 2-2. First Bootloader Entry Method



   Note: 1. Bootloader is running out of the SAM L11 Secure Flash (BOOT Region).

2. The Bootloader commands monitor will run on external request if the value of the Bootloader Entry pin is low when the Bootloader execution starts. The entry pin takes priority over any other method of entry.

   Figure 2-3. Second Bootloader Entry Method



   Note: 1. Bootloader is running out of the SAM L11 Secure Flash (BOOT Region).

2.3 Application Image Configuration

An application is considered valid if:

- The size of the field specified in the application image is less than the maximum application size for the device,
  AND
- If the size is a multiple of 256 bytes minus 32 bytes

The image size value must be written at the offset 0x10 from the application image start offset. Normally this value is occupied by a reserved exception vector (ARM vector table), therefore there is no possibility of it interfering with the application code.

Additionally, the application image integrity is verified by a SHA-256 hash. The hash value must be located in the 32 bytes following the application image.

2.4 Hardware Configuration

The UART pins used by the Bootloader are listed in the following table.

Table 2-1. Hardware Configuration

Device	UART Tx	UART Rx	Entry Pin
SAM L11	PA16	PA17	PA19

The Bootloader Entry pin has an active low level. The value of the pin is sampled at the beginning of the Bootloader execution. Although the internal pull-up resistor is enabled before sampling the pin, it is recommended that the Bootloader Entry pin be pulled up externally for improved noise immunity.

The UART setting used by the secure Bootloader is 115200, 8, N,1.

Note: These are the default settings used by the Bootloader, but it can be modified by the user as desired for more flexibility.

2.5 Device Configuration

To ensure full chain of trust protection of sensitive information, the NVM Boot Configuration Row (BOCOR) must have the following settings:

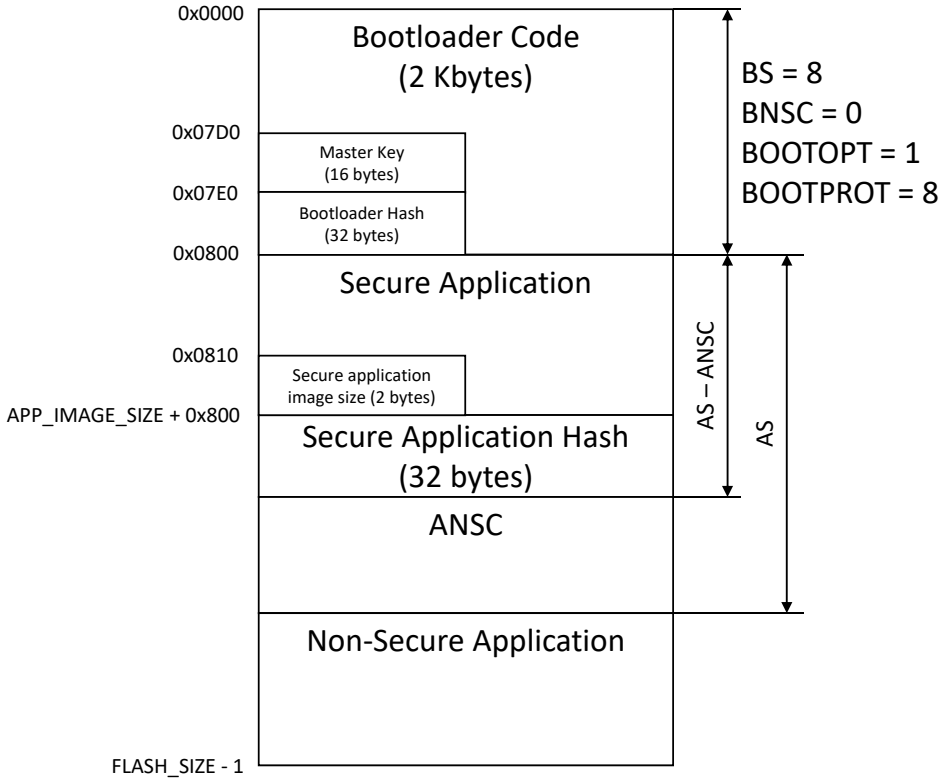
- Boot Flash Secure Size (BS) = 8
- Boot Flash Non-Secure Callable Size (BNSC) = 0
- Boot Protection size (BOOTPROT) = 8
- Boot Option (BOOTOPT) = 1
- Boot Configuration CRC (BOCORCRC) = Correct BOCOR CRC32 value
- Boot Configuration Row Hash (BOCORHASH) = Correct BOCOR SHA-256 value

Both the BOCORCRC and BOCORHASH must be computed and programmed, so that the Boot ROM acknowledges the application. Refer to the Boot ROM section of the SAM L11 product data sheet for more details.

Additionally, User Row bit RXN (RAM is eXecute Never), must be cleared before executing the Bootloader, as the Bootloader runs from the RAM to allow self-update.

The specified fuse settings results in the memory map are shown in the following figure:

Figure 2-4. SAM L11 Flash Memory Mapping



3. Bootloader Monitor Commands

All Bootloader commands have the same general format, as shown in the following table.

Table 3-1. Bootloader Commands

Command ID	Guard Value	Data 0	…	Data N
1 byte	4 bytes	4 bytes	…	4 bytes

The number and meaning of the data words varies with the command. All data words must be sent in the little-endian (LSB first) format.

The Guard Value must be a constant value of 0x2b620bc3, which provides additional protection against spurious commands.

All bytes of the command frame must be sent within 100 ms of each other. After 100 ms of idle time, the incomplete command is discarded, and the Bootloader returns to waiting for a new Command ID. This behavior allows the host to re-synchronize if there was synchronization loss.

The Bootloader understands the following commands:

1. `Unlock (0xA0)`
2. `Data (0xA1)`
3. `Verify (0xA2)`
4. `Reset (0xA3)`

The `Unlock` command must be issued before the first `Data` command and has the following payload:

- Data 0 – Starting Offset
- Data 1 – Image Size
- Data 2 – Data 5 – Nonce

The Starting Offset is the offset from the beginning of the Flash memory. To upgrade the Bootloader, this value must be set to zero.

The application image offset is device-dependent and valid values are listed in the following table. The image offset must be aligned at an Erase Unit Size boundary, which is also device-dependent. The image size must be in increments of Erase Unit bytes.

Table 3-2. Valid Values for Application Image Offset

Device	Application Offset, bytes	Erase Unit Size, bytes
SAM L11	2048	256

The Nonce is an arbitrary 16-byte value that must never repeat for a given key. The number may be an incrementing counter, a sequence of random bytes, or a combination of both. In case random numbers are used, ensure that they are derived from a reliable source of entropy. Details on the use of this number for encryption and authentication are outlined in Security Procedures.

The `Data` command is used to send image data, and has the following payload:

- Data 0 – Starting Offset
- Data 1 – Data N – Image Data (Erase Unit Size bytes)
- Data N+1 – Data N+17 – Image MAC (16 bytes)

A starting offset must be located inside the region previously unlocked through the `Unlock` command. Any attempts to request the write outside of the unlocked region will result in an error, and the supplied data will be discarded.

The image data must be encrypted and authenticated. The image Message Authentication Code (MAC) ensures that the data is authenticated and is not damaged during transfer. MAC also ensures that the data belongs to the current location and the current file. Details on the generation and use of this number for authentication and encryption are outlined in Security Procedures.

This Bootloader supports simultaneous Flash memory write and reception of the next block of data. The next block of data may be transmitted as soon as the status code is returned for the first one.

Due to this behavior, the status code for the last block will be sent before this block is written into the Flash memory. To ensure that this block is written, the host must send another command and wait for the response. Therefore, the `Verify` or `Reset` command must be sent after the last block of data.
Note: Block will not be written if the MSB of the data address is 0x8xxx_xxxx. This allows to perform a verification without programming operation.

The `Verify` command is used to verify the image data and has no payload. The image integrity during the transfer is ensured through the MAC, and the Bootloader reads back the data after it was written into the Flash memory. The `Verify` command may be issued at any time during the image upload as it will return the current value of the verification status.

The `Reset` command is used to exit the Bootloader and run the application, except when the Bootloader entry is done using the Bootloader entry pin. In the specific case, the entry pin value must be set high once the programming is done and a hardware reset must be performed to run the application.

The `Reset` command has the following payload:

- Data 0 – Arbitrary Value 0
- Data 1 – Arbitrary Value 1
- Data 2 – Arbitrary Value 2
- Data 3 – Arbitrary Value 3

The supplied arbitrary values are passed to the application in the first four locations in the SRAM.

4. Bootloader Monitor Response Codes

The Bootloader will send a single character response code in response to each command. Sequential commands can only be sent after the response code is received for a previous command, or after a 100 ms time-out without a response.

The following are possible response codes:

- OK (0x50) – Command received and processed successfully
- ERROR (0x51) – There were errors during the processing of the command
- INVALID (0x52) – Invalid command is received
- VERIFY_OK (0x53) – MAC verification is successful
- VERIFY_FAIL (0x54) – MAC verification failed

5. Programming Algorithm

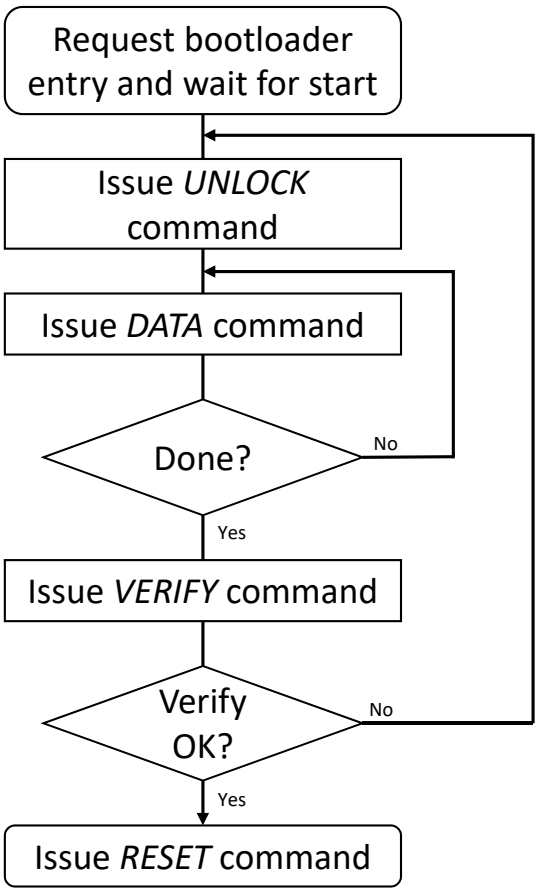
After issuing each command, the host must wait for the response code for at least 100 ms. If no response code is received during this time, the command may be considered lost and may be repeated again.

The host controller must perform the following actions to update the firmware:

1. Request the Bootloader entry.
2. Wait for at least 50 ms for the Bootloader to start.
3. Issue the `Unlock` command with the required image parameters.
4. Send the `Data` command with the application data.
5. If the MSB of the address is set to 0x8xxx_xxxx, jump to 7.
6. Application data is programmed. Repeat item 4 until the entire image is written.
7. Issue the `Verify` command and check the response code.
8. If the response code is VERIFY_FAIL, then repeat the update starting from the item 3.
9. Issue the `Reset` command.

This algorithm is illustrated in the following figure:

Figure 5-1. Programming Algorithm



6. Security Procedures

The security procedures described in this section are presented from the point of view of the host. The bootloader performs similar procedures, but in reverse order.

This bootloader is using the CBC-MAC based on the AES-128 for command and data authentication and the AES-128 in Output Feedback (OFB) mode for encryption of the transferred data.

Security and authentication of the data is based on the master key stored in the last 16 bytes of the bootloader. If the key needs to be updated after the initial programming of the bootloader image, the entire bootloader must be replaced. This is necessary because the key value is included in the SHA-256 hash of the bootloader. The bootloader is updated the same way as a regular application, hence it requires full security processing. This means that the old key must be used for encryption and authentication of the new key. If the old key is lost, then there is no possibility to recover the device using this bootloader.

The default encryption key is `00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f`.

The proper security practice is to never use the master key directly. Instead, derive a session key from the master key and other information specific to a given session or application image.

The procedure to obtain the session key is as follows:

```
GenerateSessionKey(StartingOffset, ImageSize, MasterKey)
    Temp = (StartingOffset || ImageSize || 0x00000000 || 0x00000000)
    SessionKey = AES(AES(Nonce, MasterKey) ^ Temp, MasterKey)
```

Where, '||' is a concatenation operator, 0x00000000 is a 32-bit zero padding value, and AES(Key, Data) is AES-128 operation over 16 bytes of Data using the Key. Starting Offset and Image Size must match to the corresponding values passed through the Unlock command.

The resulting session key is unique to a specific image and location in the Flash memory. The Nonce is used to eliminate possible duplication of the session keys, because typically application images will be located at the same offset, and have roughly the same size.

The session key is 16 bytes long and is used for encryption and authentication operations.

The encryption and authentication routine operates on blocks of data aligned at the Erase Block Size boundary. It uses a command header (Block Offset) to ensure that authenticated data cannot be written into a different location. The encryption algorithm splits the data into 16 byte blocks, and encrypts them using AES-128 algorithm. After the block of data is encrypted, it needs to be authenticated, since the encryption is susceptible to cipher text modifications that are not detectable during the decryption stage.

```
EncryptAndAuthenticateBlock(SessionKey, BlockOffset, Data)
    Result = GuardValue || BlockOffset
    EncIV  = (offset || 0x00000000 || 0x00000000 || 0x00000000)
    MacTemp = (offset || 0x00000000 || 0x00000000 || 0x00000000)
    MAC = AES(MacTemp, SessionKey)
    for Block in Data:
        EncTemp = AES(EncIV, SessionKey)
        EncBlock = Block ^ EncTemp
        EncIV = EncTemp
        MAC = AES(MAC ^ EncBlock,  SessionKey)
        Result = Result || EncBlock
    return Result || MAC
```

Python scripts based on these security procedures are provided to help with generating and programming the encrypted images. Refer to 7. PC Utilities for Working with the Bootloader (Encrypted Images) for more details.

7. PC Utilities for Working with the Bootloader (Encrypted Images)

A number of evaluation utilities are provided with the Bootloader. The process is split into preparing an encrypted image that can be freely distributed over open channels, and an actual update.

7.1 Software Requirements

Python 2.X is required to execute the scripts provided along with this Application Note. The latest version of the Python is available for download from the following location: https://www.python.org/downloads/.

Note:

- In addition to Python, the pySerial module is required to enable the access to the serial COM port of the computer. Refer to https://pypi.org/project/pyserial/ to download latest 2.X version.
- In addition to Python, the pyCrypto module is required to add hash functions (SHA-256) and various encryption algorithms (AES, DES, …). Refer to https://pypi.org/project/pycrypto/ to download latest 2.X version.

7.2 Encrypt.py Utility

The encryption utility, `encrypt.py`, will add the suffix `.enc` to the input file name to obtain the name of the output file. Each invocation will produce different file contents, because a random Nonce is generated for each invocation. This utility has the following syntax:

Options:

```
-h, --help            show this help message and exit
-r, --raw    encrypt as raw data (do not add application hash and application image size)
-k KEY, --key=KEY         encryption key (16 bytes separated with ':')
-f FILE, --file=FILE     binary file to program
-o OFFS, --offset=OFFS     destination offset (default 0x800)
```

Example invocation (single line):

- File Programming:

```
python encrypt.py -k 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f -f test_app_l11.bin -o
0x800
```

- For verification without Programming operation, refer to Programming Algorithm for more information:

```
python encrypt.py -k 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f -f test_app_l11.bin -o
0x80000800
```

Figure 7-1. Successful Application Encryption Linked @ 0x800

```
C:\Users\M50534\Documents\Cortex-M23\Omega_SAM_L1x\Bootloader for SAM L10L11\Secure UART Bootloader for S
AM L11\bootloader_uart_l11_sec\tools>python encrypt.py -k 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f
 -f Secure_project_prog.bin -o 0x800
Done
```

⚠ CAUTION **-r** option is mandatory to use for any binaries other than the application image binary file, as it prevents the addition of the application image size and the hash value to the binary.

Example invocation (single line):

```
python encrypt.py -k 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f -f bootloader_l11.bin -o
0x0000 -r
```

7.3 Boot.py Utility

The update utility, `boot.py`, takes an image and uploads it over the serial port, which has the following syntax:

Options:

```
-h, --help             show this help message and exit
-v, --verbose         enable verbose output
-i PATH, --interface=PATH     communication interface
-f FILE, --file=FILE     binary file to program
--boot              enable write to the bootloader area
```

Example invocation (single line):

```
python boot.py -v -i COM12 -f test_app_l11.bin.enc
```

The `--boot` option is necessary if an encrypted image has an offset less than the Bootloader size. This is an additional protection to prevent accidental overwrite of the Bootloader.

Figure 7-2. Successful Programming of an Application Linked at 0x800

```
C:\Users\M50534\Documents\Cortex-M23\Omega_SAM_L1x\Bootloader for SAM L10L11\Secure UART Bootloader for SAM L11\bootloader_uart_l11_sec\tools>python boot.py -v -i COM9 -f NS_LEDblink_prog.bin.enc
Unlocking
Uploading 29 blocks at offset 2048 (0x800)
... block 1 of 29
... block 2 of 29
... block 3 of 29
... block 4 of 29
... block 5 of 29
... block 6 of 29
... block 7 of 29
... block 8 of 29
... block 9 of 29
... block 10 of 29
... block 11 of 29
... block 12 of 29
... block 13 of 29
... block 14 of 29
... block 15 of 29
... block 16 of 29
... block 17 of 29
... block 18 of 29
... block 19 of 29
... block 20 of 29
... block 21 of 29
... block 22 of 29
... block 23 of 29
... block 24 of 29
... block 25 of 29
... block 26 of 29
... block 27 of 29
... block 28 of 29
... block 29 of 29
Verification
... success
Rebooting
Done!
```

7.4 Key_Update.py Utility

The `key_update.py` is a utility to set or update the encryption key in the Bootloader image. This utility calculates and appends the SHA-256 hash of the Bootloader image. The file is updated in-place and has the following syntax:

Options:

```
-h, --help             show this help message and exit
-k KEY, --key=KEY        encryption key (16 bytes separated with ':')
-f FILE, --file=FILE     binary file to program
```

Example invocation (single line):

```
python key_update.py -k 0:1:2:3:4:5:6:7:8:9:0:1:2:3:4:5 -f bootloader_l11.bin
```

7.5 Troubleshooting Guide

7.5.1 Verify Python Version

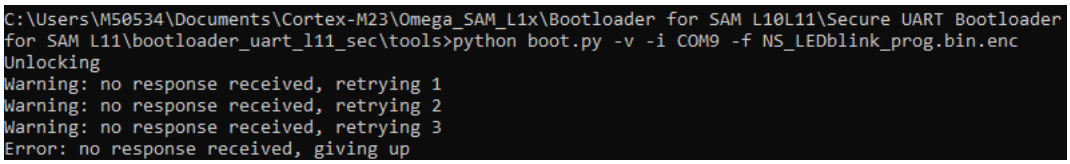
There might be a situation, the `boot.py` call leads to an error message.

Ensure that the Python 2.X path is defined in the environment variables after installation, and is used instead of Python 3.X, even if version 3.X is already installed on the host, otherwise `boot.py` cannot be called by the python sequence.

7.5.2 Check Bootloader Execution with Terminal

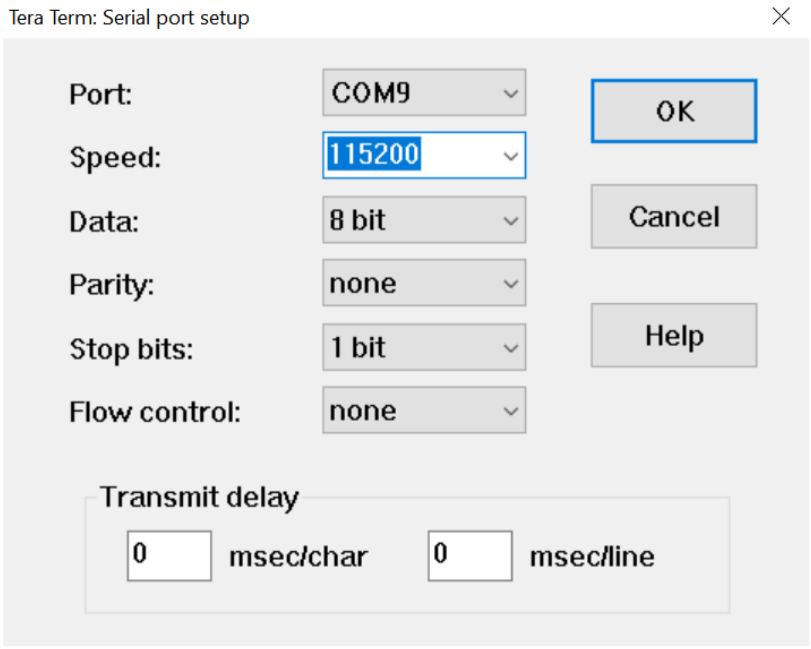
It may happen that the host does not receive any response from the Bootloader as shown in the following figure:

Figure 7-3. No Response from Bootloader



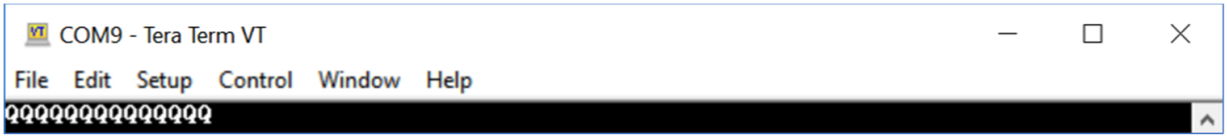
In that case a terminal can be used with the following settings to verify whether the Bootloader is running on the device or not.

Figure 7-4. Bootloader COM Port Settings



If the Bootloader is running, “Q” characters must appear when typing any character on the terminal, as shown below:

Figure 7-5. Bootloader Response Through Terminal



Note: The terminal must be connected to the COM port that is dedicated to the UART pins and not the SAM L11 device. If not, verify the following points to resolve this issue:

- If there is already a valid firmware in the Flash memory, ensure the Bootloader entry pin is grounded, then reset the board
- Check that UROW RXN fuse bit is cleared

8. Revision History

Revision A - May 2018
This is the initial release of this document.

Revision B - February 2019
The following updates and changes were made:

Section	Updates
Introduction	Updated the text with new material.
Features	Updated features and corrected erroneous text.
Bootloader Implementation	New Chapter Heading.
Bootloader Flow	New Chapter added.
Method of Entry	Updated existing content with new images and added new notes.
Hardware Configuration	Updated existing chapter with new text, and moved some content to new chapters.
Device Configuration	Updated existing chapter with a new flow chart for SAM L11 Flash Memory Mapping.
Programming Algorithm	Updated chapter with a new Algorithm flow chart and additional text.
PC Utilities for Working with the Bootloader (Encrypted Images)	Removed outdated text and updated the chapter with new sub chapters: • Software Requirements • Encrypt.py Utility • Boot.py Utility • Key_Update.py Utility • Troubleshooting Guide • Verify Python Version • Check Bootloader Execution with Terminal

Revision C - February 2019
Added ISBN which was missing in the Revision B doc.

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-4188-5

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820