Technical Disclosure Commons

Defensive Publications Series

June 11, 2019

Board Identifier (BoardID) Locked Firmware Images

Randall R. Spangler

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation

Spangler, Randall R., "Board Identifier (BoardID) Locked Firmware Images", Technical Disclosure Commons, (June 11, 2019)
https://www.tdcommons.org/dpubs_series/2265

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Board Identifier (BoardID) Locked Firmware Images

Abstract:

This publication describes a method for a firmware developer to push out re-writable (RW) firmware images that support legacy-generation, current-generation, and future-generation boards. To do so, the firmware developer uses a minimum of three (3) 32-bit-writable words that describe a board identifier, herein called BoardID. The BoardID contains three (3) fields, which describe a board type, an inverse of the board type, and flags. This BoardID solution allows the firmware developer to push out RW firmware that are not as universal and global as traditional mass-production firmware images and are not as unique as node-locked firmware images. The firmware developer, using this BoardID solution, may push out firmware updates to a subset of boards. Furthermore, any original device manufacturer (ODM) can support existing RW firmware images using this BoardID solution, while increasing their ability to better-manage future board-development phases.

Keywords:

Bit, binary digit, 0, 1, binary value, byte, word, memory, flash memory, storage, flash storage, non-volatile memory, NVM, NVMEM, electrically erasable programmable read-only memory, EPROM, board identifier, BoardID, board number, flag, firmware, permanent software.

Background:

Most computing devices use firmware, which is a specific class of computer software that provides low-level or limited control on a device's hardware. Developers may store firmware in non-volatile memory, such as read-only memory (ROM), erasable programmable read-only memory (EPROM), or flash memory. When firmware was first introduced in the computing world, changing the firmware of the device was not a common practice. Today, however, device manufacturers and firmware developers often push out firmware updates across multiple devices.

Currently, there are two types of re-writable (RW) firmware images: production firmware images and node-locked firmware images. The production firmware images are signed with a production key and run on production (customer) hardware. Device manufacturers and firmware developers share the production firmware images across various devices. Differently said, the same binary code may run on different subsets of devices, such as on Device Type A, on Device Type B, on Device Type C, and so forth. In addition, the production firmware images are rollback-protected, and they are global. That means if any undesired or faulty production firmware image leaks to the public, the firmware developer needs to update all RW firmware images in all types of devices to a newer version that has no known bugs.

Meanwhile, the node-locked firmware images are signed with the developer (not production) key. Firmware developers run the node-locked firmware images on a single device, such as a single notebook. Firmware developers use the node-locked firmware images to produce, test, and debug firmware. Therefore, the node-locked firmware images are not suitable for pushing out firmware updates to the public because the developer can only send a specific node-locked firmware image to a specific device with a unique manufacturer's serial number. When using a

node-locked firmware image, the devices cannot share a binary code. Each device requires its own binary with its own unique signature, and that scales poorly.

It would be desirable and beneficial for a firmware developer to utilize RW firmware images that are not as universal as the production firmware images and are not as unique as the node-locked firmware images. For instance, a firmware developer may want to send a firmware update to only the clients who own Device Type A and not to the clients who own other types of devices because, let us assume, only Device Type A contains a certain firmware bug. Additionally, the firmware developer may want to increase its ability to "dogfood" a new RW firmware image on multiple devices before rolling it out to the clients, where the term "dogfood" or "dogfooding" often refers to testing a product. In this case, using the production firmware images is not prudent, but using the node-locked firmware images is cumbersome and limited in scale.

Description:

Device manufacturers install security chips on their devices, which are often located on a device's motherboard. Among other things, the security chips contain a processor with re-writable (RW) firmware images. The security chips often contain a protected non-volatile memory block, such as in a flash memory. This protected memory block may be a 32-bit-writable word that is not erasable, and it is protected in the RW firmware. A read-only (RO) image verification process uses the first 128 words of the RW firmware. In addition, each RW firmware image header has a corresponding 128-bit INFO_MASK with one (1) bit corresponding to each of the first 128 words in an INFO1 block of the non-volatile memory.

As it is currently done, before verifying a RO image, the mask read-only memory (ROM) verifies that all bits that are one (1) in the INFO_MASK have the expected value in the INFO1

block of the non-volatile memory. The RO performs a similar verification for the RW firmware image using the next 128 words of the INFO1 block. This is done in order to lock out old firmware images for rollback protection. It is worth noting that each bit of the image header indicates whether an entire 32-bit word of the INFO1 block exactly matches the expected value. That means that firmware developers cannot use this portion of non-volatile memory to lock a board identifier, herein called BoardID. An elegant solution may be to use three (3) additional 32-bit-writable words in the INFO1 block of the memory to store the BoardID. The firmware developer, however, may choose to use more than 96 bits in the INFO1 block for the BoardID.

Figure 1 shows how the firmware developer may use three (3) 32-bit-writable words for the BoardID.

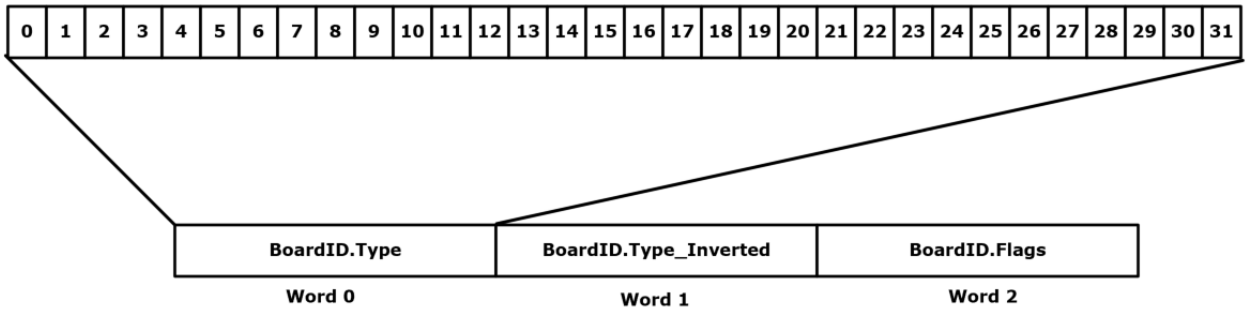

Figure 1

The developer may use three (3) 32-bit-writable words for the BoardID that are stored in the INFO1 block of the non-volatile memory. The non-volatile memory is initially all ones (1s). The ones (1s) can be turned into zeros (0s) with a write operation, but the zeros (0s) can only be turned into ones (1s) by erasing an entire page of the non-volatile memory back to ones (1s). In the case of the INFO1 block, the hardware prevents the page erase. Differently said, the ones (1s)-to-zeros (0s) transition is a one-way operation. A RO cros_loader uses this 96-bit binary to verify the RW firmware image. It is worth noting that in this context, the term "word" may refer to a "field". Below is a brief explanation of the three (3) fields:

- Word 0 — this field defines the BoardID.Type. This field is sufficiently large to store a 4-character-unique-developer code.
- Word 1 — this field defines the bit-inverse value of BoardID.Type, herein called BoardID.Type_inverted. This bit-inverse value prevents an attacker, who may be able to write to the INFO1 block, from converting a board type into one with fewer bits. In other words, it is an added security feature in case the attacker hacks the BoardID.Type. Logically stated:

  BoardID.Type_inverted = ~BoardID.Type, unless BoardID.Type = 0xFFFFFFFF

  where "~" denotes the inversion logic operation.
- Word 2 — this field defines the BoardID.Flags. The firmware developer uses this 32-bit field to lock board builds to a board-development phase or generation.

BoardID.Type Encoding

The BoardID.Type is a 32-bit field, and the developer may store the 4-character-unique-developer code in the following manner:

- Board.Type = 0 — this value means that the factory incorrectly programmed the BoardID and the firmware developer needs to clear this code.
- Board.Type = 1 — this value means that the factory did not program the BoardID, and the user logged in the device before the BoardID was set.
- Board.Type = 0x20202020 to 0x7F7F7F7F — this range of values defines the brand code, and it is stored as an 8-bit American Standard Code for Information Interchange (ASCII) code. The first character of the brand code is the high byte of the BoardID, and it is padded with 0x00 bytes when the unique-developer code is less than four (4) characters. For example, "ZZCR" = 0x5A5A4352.

- Board.Type = 0xFFFFFFFF — this value means that the BoardID has not been programmed. Boards with this BoardID.Type will match headers with any BoardID.Type. Headers with this BoardID.Type will only match unprogrammed boards.
- BoardID.Type = other values — these values remain reserved.

BoardID.Flags Encoding

The BoardID.Flags is a 32-bit field, and the verification process treats all the bits identically. Nevertheless, the firmware developer recommends that an original design manufacturer (ODM) uses this 32-bit field in the following manner:

- Bits 0 to 6, where the least significant bit (LSB) is 0 — the ODM may use these bits to differentiate and describe a board-development phase, and it may do so by clearing the bits (turning a one (1) to a zero (0)) as the ODM progresses in the board-development phases. For example:
  - 1111111 — the ODM did not program the board.
  - 1111000, 1110000, 1100000, 1000000 — the ODM assigns these values to boards under development, where boards with values 1100000 and 1000000 denote a late board-development phase. The ODM may use these boards for debugging or "dogfooding".
  - 0000000 — the ODM assigns this value to a mass production (MP) board.

  It is worth noting that BoardID-locked RW firmware for a later development phase can run on boards produced in an earlier phase, but not the other way around. One reason for such restriction is to eliminate the possibility to run an under-development RW firmware on mass-produced boards.

- Bit 7 — the ODM may use this bit to distinguish boards of an early development phase, but, somehow, someone hacked them to appear as being from a later board-development phase. The ODM assigns a zero (0) to boards in a deployment phase and a one (1) to mass-produced boards.
- Bits 8 to 31 — the ODM may use these bits to describe and distinguish boards with different features. To increase the future value of these bits, the ODM may set current boards with 0x00007F, which provides a convenient way to separate boards that ship with different RW firmware functionality. This way current RW firmware images cannot run on future boards with features that differ from the current ones. In addition, this simplifies testing because the firmware developer need not test every RW firmware they release on specific boards. In cases where the firmware developer adds a new feature that is incompatible with current boards, the firmware developer can change the script to a add a feature bit. The RW firmware images for new boards will require the newly-added feature bit. To reiterate, the new boards will be able to use both the previous and the new RW firmware images, but the current boards will not be able to use the newly-added bit and can safely reject the new RW firmware image.

Programming the BoardID

From a fabrication facility, the ODM ships boards with BoardID set to 0xFFFFFFF 0xFFFFFFF 0xFFFFFFF — all bits set to one (1), regardless of the type of board. When the BoardID is set to all ones (1s), it means that the board is yet to be programmed. In that case, the RO cros_loader skips the BoardID checks and the RW firmware leaves that portion of the INFO1 block unprotected. Otherwise, the RW firmware protects the BoardID in the INFO1 block early in the boot process.

The firmware developer adds a Trusted Platform Module command (*e.g.*, a TPM2 vendor command) so the ODM knows what value to assign to the BoardID. The RW firmware rejects the TPM2 command if the BoardID is not set to all ones (1s). Some example BoardID values with their associated meaning are:

- 0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF — this code means that the BoardID has not been programmed.
- "ABCD" ~"ABCD" 0x00007F7F — this code denotes an "ABCD" development board.
- "ABCD" ~"ABCD" 0x00007F80 — this code denotes an "ABCD" mass-production board.
- "ZZCR" ~"ZZCR" 0x00007F7F — this code denotes an early board-development phase, which may not be used for "dogfooding".
- "FOOB" ~"FOOB" 0x00007F80 — this code denotes a "FOOB" mass-production board. The flags indicate that it is a next-generation board, whose RW firmware is incompatible with current-generation boards.
- 0x0 0x0 0x0 — the RW firmware developer uses this code to check for mismatch in case the BoardID cannot be read because this code will only match an RW firmware image with Type_Mask = Falgs = 0.

Adding BoardID Fields to the RW Header

The RW firmware developer recognizes that for an ODM to adopt this BoardID solution, there needs to be a way to support the existing RW firmware images.

- Word 0 — the ODM can use Hdr.BoardID_type.

- Word 1 — the ODM can use Hdr.BoardID_Type_Mask. 1-bits mean that the corresponding bits in Hdr.BoardID.Type must match BoardID.Type stored in the INFO1 block, while 0-bits are “don't care”.
- Word 2 — the ODM can use Hdr.BoardID_Flags. 1-bits need to match BoardID.Flags stored in the INFO1 block, while 0-bits are “don't care”.

Signing Firmware Images

The ODM sets BoardID.Type and BoardID.Flags at the signing-time and not the build-time. Examples of image header settings are:

- “any code” 0x0 0x0 — this code runs on all boards, current or future.
- “any code” 0x0 0x00007F00 — this code runs on all current-generation boards. The ODM signs this way mass-production RW firmware images, unless there exists a need to be locked to a specific board type.
- “ABCD” 0xFFFFFFFF 0x00007F00 — this code runs on any “ABCD” board.
- “ABCD” 0xFFFFFFFF 0x00007F7F — this code runs only on “ABCD” development boards.
- “ABCD” 0xFFFFFFFF 0x00007F80 — this code runs only on “ABCD” mass-production boards.
- “any code” 0x0 0x00017700 — this code runs on any next-generation boards, but does not run on any current-generation boards.
- “ABCD” 0xFFFF0000 0x00007F00 — this code runs on all current-generation boards where the board type matches “ABxx”, in other words, the board type starts with “AB”.

Potential Issues

Boards that ship without the BoardID programmed

It is also possible that an ODM may accidentally ship boards without programming the BoardID. Theoretically, these boards can boot any BoardID-specific image. To guard against that, a normal operating system (OS) checks the BoardID during every boot. If the BoardID == "all 1 bits" at the time of the boot, the OS will set the BoardID to the following:

- BoardID.Type = brand code from a protected block of the application processor
- BoardID.Type_Inverted = the inverse of the brand code from the protected block of the application processor
- BoardID.Flags = 0xFFFFFF00 and hard-coded flags

The OS uses the bottom byte of the flags for a board-development phase, which in this case is not known, so the OS sets it to 0x00. Meanwhile, the top bits of the flags are hard-coded based on the board generation, so the ODM can hard-code them in the OS image.

Boards that ship with the wrong BoardID programmed

Alternatively, an ODM may mis-program the BoardID under two scenarios:

- The ODM mis-programs the BoardID of a pre-release board by programming it as a mass-production board. In this case, the firmware developer cannot sign BoardID-locked images for such board, therefore, the firmware developer needs to use node-locked images.
- The ODM mis-programs a mass-production board with a pre-release BoardID. In this case, the mass-production board is willing to run BoardID-locked images for that board. If the firmware developer decides to disallow this action, it needs to release a production RW firmware that looks for that BoardID and then writes it to 0x0000000. At that point, the

mass-production board only runs images where Hdr.BoardID_Type_Mask=0, since the type and its inverse no longer matches.

The OS script that checks for a BoardID that is not programmed, also checks if the BoardID does not match the 4-character-unique-developer code in the protected block of the application processor. The firmware developer can track these mismatches like other OS statistics.

In summary, the BoardID solution allows a firmware developer to push out RW firmware that are not as universal and global as traditional mass-production firmware images and are not as unique as node-locked firmware images. The firmware developer may push out firmware updates to a subset of boards. Furthermore, any original device manufacturer (ODM) can support existing RW firmware images using this BoardID solution, while increasing their ability to better-manage future board-development phases.

References:

[1] "Google Security Chip H1." Accessed June 3, 2019. https://2018.osfc.io/uploads/talk/paper/7/gsc_copy.pdf.

[2] "STM32F0x1/STM32F0x2/STM32F0x8 Advanced ARM®-based 32-bit MCUs." January 2017. Accessed June 3, 2019. https://www.st.com/content/ccc/resource/technical/document/reference_manual/c2/f8/8a/f2/18/e6/43/96/DM00031936.pdf/files/DM00031936.pdf/jcr:content/translations/en.DM00031936.pdf.