FLASH BOOT FOR TRAVEO II IP BROS

Table of Contents

FLASH BOOT FOR TRAVEO II IP BROS

The information in this document is subject to change without notice.

Published by

Cypress
198 Champion Court
San Jose, CA 95134 USA

Attention please!
The information herein is given to describe certain components or functionalities and shall not be considered as warranted characteristics.

FLASH BOOT FOR TRAVEO II IP BROS

1 PURPOSE/SCOPE

1.1 Purpose

The purpose of the Block Requirements Objective Specification (BROS) is to document the implementation details of the *Traveo II Flash boot* firmware.

1.2 Scope

This document covers the implementation details of the *Traveo II Flash boot* IP throughout the SW/FW IP development process from IPS0 through IPS4.

1.3 How to file Defects or Enhancement Requests against this document

Defects and/or Enhancement Requests against the Flash boot are tracked in Jira.

2 RESPONSIBILITIES:

The Vice President of New Product Development or Senior VP of Product Engineering/Test Engineering or Vice President of IP is responsible for defining the block development and review process.

The Chip Lead and Product Line Design Director have the responsibility for approving the block requirements (Section 4).

The Program Manager of the lead NPP or the lead of the Center of Excellence is responsible for the block requirements and block preparation. This document is meant to be a living document, and updated as the block development proceeds.

3 REFERENCED DOCUMENTS:

00-00064 – CYPRESS RECORD RETENTION POLICY

001-91989 – BUSINESS PROCESS FOR PSOC PART DEFINITION

001-98134 – MXS40 SYSTEM ARCHITECTURE SPECIFICATION (SAS)

002-03298 – MXS40SROMPSOC SOFTIP BLOCK REQUIREMENTS OBJECTIVE SPEC (BROS)

002-11462 – MXS40 CORE PLATFORM VERIFICATION REQUIREMENTS

002-12095 – SNPD SYSTEM EROS – MXS40 BOOTLOADER SDK

002-13924 – AN213924 – PSOC 6 MCU BOOTLOADER SOFTWARE DEVELOPMENT KIT (SDK) GUIDE

FLASH BOOT FOR TRAVEO II IP BROS

002-19314 – TRAVEO(TM) II AUTOMOTIVE BODY CONTROLLER ENTRY FAMILY ARCHITECTURE TECHNICAL REFERENCE MANUAL (TRM)

002-19761 – FLASH BOOT FOR TRAVEO II SEROS

002-20889 – SNPD SYSTEUM EROS - CYMCUELFTOOL

002-22934 – CYMCUELFTOOL 1.0 USER GUIDE

ALXD-22 – HOW TO GENERATE RSA PUBLIC KEY FOR PSOC6 SI *A SECURE BOOT VALIDATION

ALXD-23 – HOW TO CREATE AND BUILD AN APPLICATION IN CYPRESS SECURE APPLICATION FORMAT FOR MXS40 DEVICES

ALXD-25 – HOW TO PROGRAM THE FLASH BOOT INTO THE PSOC *A DEVICES

ALXD-26 – HOW TO BUILD A TOC PART 2 FOR PSOC6 *A DEVICES

ALXD-31 – FLASH BOOT VERSIONING

ALXD-34 – HOW TO CALL FLASH BOOT SHARED FUNCTIONS

ANRY-280 – WHAT IS DUMMY FLASHBOOT

ANRY-299 – HOW TO GENERATE PATCHES FOR TVII

DESP-36 – CYACD2 FILE FORMAT

DOFE-1 – TRANSITION TO NORMAL FOR THE TRAVEO II FAMILY

DOFE-2 – SECURE APPLICATIONS FOR THE TRAVEO II FAMILY AND PSOC6. TRANSITION TO SECURE INSTRUCTIONS.

DOFE-3 – KNOWLEDGE SHARING DUE TO INVESTIGATION PERFORMED FOR THE EFUSE-IP DEFECT (CDT329370 PROGRAM-READ TIMING REQUIREMENT FOR THE TRAVEO II FAMILY DEVICES)

DOFE-5 – RSA3K FEATURE FOR TVII DEVICES. CHANGES FOR FLASH BOOT PROPOSAL.

VENN-36 – SECURE BOOT IN MXS40

VENN-40 – TRAVEO-II BOOTLOADER - INITIAL VERSION

VENN-44 – TRAVEO-II BOOTLOADER - VERSION 3

VENN-45 – SECURE BOOT FOR TRAVEO-II

VLAD-355 – TRAVEO II FLASHBOOT TEST DESIGN REVIEW REPORT

VMEL-96 – FLASHBOOT DRIVER TEST PLAN FOR TRAVEO II

YBER-173 – FLASH BOOT PROJECT REPOSITORY

YBER-178 – TRAVEO II FLASH BOOT FIRMWARE: ARCHITECTURE RESEARCH: CAN AND LIN REQUIREMENTS AND FEASIBILITY STUDY

YBER-179 – TRAVEO II FLASH BOOT DESIGN REVIEW REPORT MEMO

TVII-B-E Family TRM
TVII-B-H Family TRM

TVII-C Family TRM

4 MATERIALS: N/A

5 EQUIPMENT: N/A

6 SAFETY: N/A

7 DEFINITIONS

FLASH BOOT FOR TRAVEO II IP BROS

Term	Definition
RSA	A Public-key cryptosystem which is widely used for data protection
SHE	Secure Hardware Extension
HSM	Hardware Security Module
CAN	Controller Area Network
LIN	Local Interconnect Network
OEM	Original Equipment Manufacturer
NAR	Normal Access Restrictions
SAR	Secure Access Restrictions
DAR	Dead Access Restrictions
CySAF	Cypress Secure Application Format
CyBAF	Cypress Basic Application Format
FIRMWARE IMAGE	A specific format for a firmware module stored in flash. This specific format is described later in this document.
ROM BOOT	ROM code stored at the device address range that starts at address 0x0000_0000. The first code executed when the device is powered on. The first phase of the boot process.
FLASH BOOT	A firmware image stored in SFLASH that provides code for the second phase of the boot process.

FLASH BOOT FOR TRAVEO II IP BROS

SECURE BOOT	Secure Boot is the term used to include the entire secure chain of trust boot process. It includes ROM Boot, Flash boot, and optionally Secure Image.
DAP	Debug Access Port
SFLASH	Supervisory flash. A dedicated flash region used by Cypress to store manufacturing information, hardware trim and wounding information, special user sections, TOC and code for the second phase of the boot process, Flash boot.
USER FLASH	The region of flash expected to store the customer's application and all its associated data.
TOC	(Table Of Contents) This table is broken up into two parts. The first part includes addresses of items frozen in the factory, such as items that are included in the FACTORY_HASH calculation and cannot be changed by the user. TOC part 2 includes addresses of the User Application, Secure Image, and user public key along with other items to be updated by the customer. Both parts are located in SFLASH.
SECURITY IMAGE	A firmware image, located in user flash, that provides the third phase of the Secure Boot process. The Security Image provides additional security for SRAM, Flash, and hardware. Security Image is designed to be modifiable and configurable by the customer.
SECURITY POLICY	The security policy is a set of services provided by Security Image which performs secure operations as needed by the target application. The security policy will differ significantly from application to application, but will generally include functions like Flash boot and Flash bootloading. The security policy runs on the Cortex-M0 security processors and the services are made available via an IPC interface to any other CPU cores in the system.

CYPRESS SECURITY POLICY	Implementation of security policy by Cypress that provides a very basic set of security services. This security policy is provided in the source form and is expected to be the basis of the customer security policy.
USER APP	A firmware image stored in user flash that provides the specific set of functionality defined by the application. The user application does not run in a secure mode, but rather relies on security services from the security policy. Multiple user apps may reside in user flash. A common case is a bootloader user app that runs first, determines if a bootload operation is necessary, and if not transfers control to a second user app with the primary functionality of the device.
FACTORY_HASH	128 most significant bits of the SHA-256 hash value computed to authenticate objects frozen in the factory.
SECURE_HASH	28 most significant bits of the SHA-256 hash value used to authenticate Flash boot and Public key in SECURE and SECURE_WITH_DEBUG life cycle stages. At the time of creating SECURE_HASH, factory frozen objects are authenticated using FACTORY_HASH. This makes sure that the Flash boot authenticated by SECURE_HASH later is the same as the one created in the factory. SECURE_HASH is computed just before transition to SECURE, so Public key needs to be known only then and the OEM provides the Public key.
PUBLIC_KEY	Cryptographic public key used to verify the digital signature of the user application and/or secure image.
IPC	Inter processor communication
NMI	Non-maskable interrupt
WDT	Watchdog timer

FLASH BOOT FOR TRAVEO II IP BROS

MMIO	Memory mapped input/output
FLL	Frequency locked loop
PSVP	Pre-Silicon Validation Platform
WFLASH	Work Region Flash memory
TEE	Trusted Execution Environment
ECC	Error checking and correction mechanism for Traveo II SRAM

8 THEORY OF OPERATION

FLASH BOOT Overview

The Flash boot program flow is shown in **Figures 1, 1.1-1.5**.
ROM boot code will transfer control to Flash boot after its tasks are complete and SFLASH has been validated.
Each section of the flow chart is labeled with an index number (n) to make a reference to it in the next sections. See **Figures 1, 1.1-1.5**.

```
@startuml
:Entry after ROM boot (1);
:Set-up SP (2);
:Initialization (3);
:Validate TOC2 (5);
if (is TOC2 Valid ? (6)) then (no)
   :Branch DEAD (26);
 note right
   Continue on Figure 1.1.
   end note
   detach
   :Interrupts and System Calls (29);
   detach
else (yes)
   if (Do Bootloading ? (8)) then (yes)
      :Branch Bootloader (28);
  note left
  Continue on Figure 1.3.
  end note
      detach
   else (no)
      :Get App #0 Reset Handler (9);
endif
:isAppValid = 0;
if (is Reset Handler Valid ? (10)) then (yes)
   if (is Authenticate App ? (11)) then (yes)
      if (is Public Key Valid ? (12)) then (yes)
         if (is Digital Signature Valid ? (13)) then (yes)
            :isAppValid = 1;
         else (no)
         endif
      else (no)
         :Branch DEAD (26);
         note right
         Continue on Figure 1.1.
         end note
         detach
```

```
      endif
      else (no)
   endif
else (no)
endif
if (isAppValid == 1 ?) then (no)
   :Get App #1 Reset Handler (9);
   if (is Reset Handler Valid ? (10)) then (yes)
      if (is Authenticate App? (11)) then (yes)
         if (is Public Key Valid? (12)) then (yes)
            if (is Digital Signature Valid? (13)) then (yes)
            else (no)
               :Branch DEAD (26);
    note left
               Continue on Figure 1.1.
               end note
               detach
            endif
         else (no)
            :Branch DEAD (26);
            note right
            Continue on Figure 1.1.
            end note
   detach
         endif
      else (no)
      endif
   else (no)
      :Branch DEAD (26);
  note left
      Continue on Figure 1.1.
      end note
      detach
   endif
else (yes)
endif
:Part 2;
note left
Continue on Figure 1.2.
end note
detach
@enduml
```

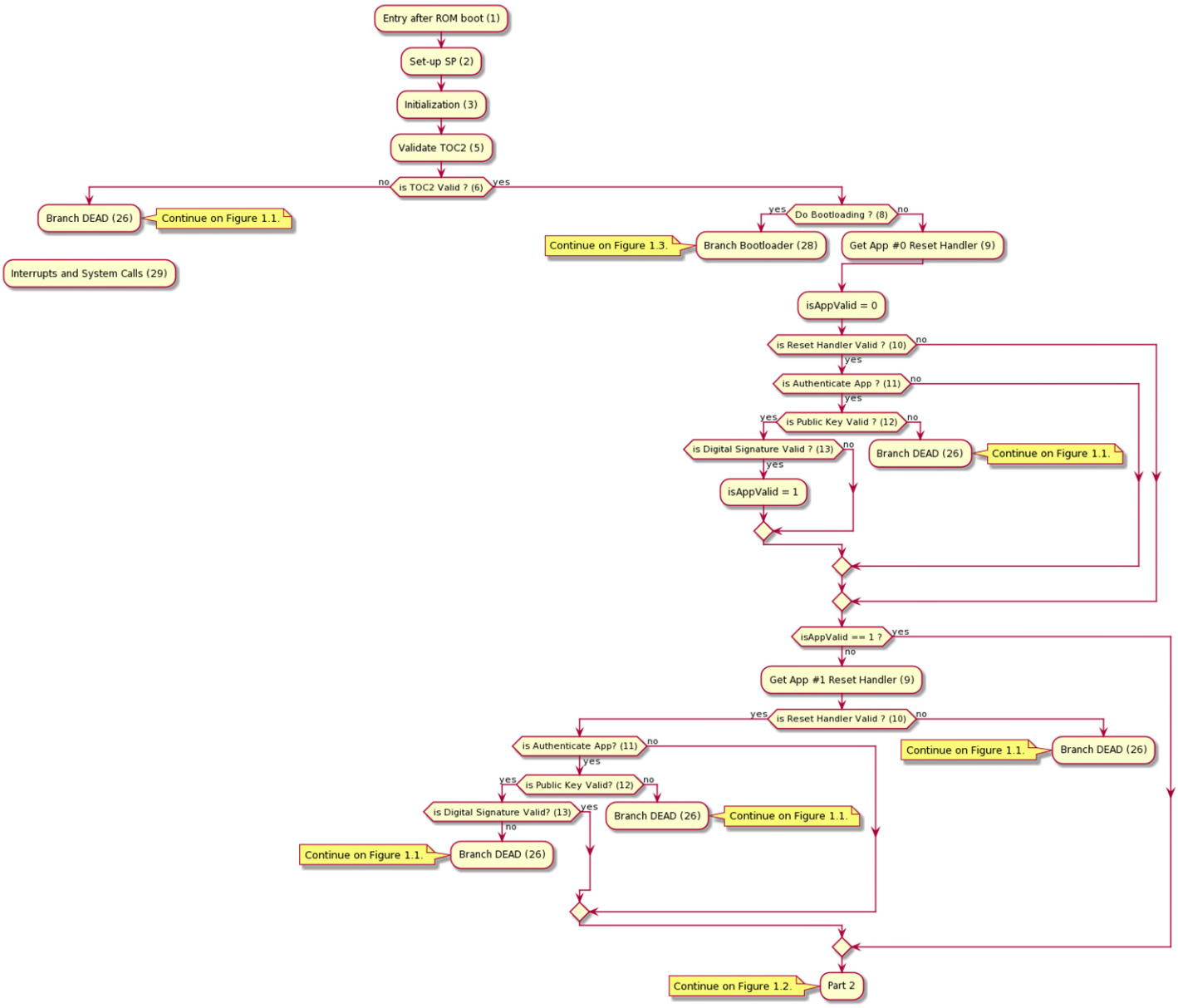


Figure 1. Flash Boot Flow (Part 1)

```
@startuml
:Branch DEAD;
:Set Error Code (30);
if (PROTECTION == VIRGIN ? (31)) then (yes)
   :Set-up SP (2);
   :Idle Loop (25);
   detach
else (no)
endif
if (LifeCycle == SECURE ? (32)) then (yes)
```

```
   :PROTECTION = DEAD (33);
   :Deploy SECURE_DEAD
    Access Restrictions (34);
   :Enable System Calls (14);
   :Set-up DAP from AR (15);
   :Apply System Protection (35);
   :Set PC = 4 (16);
else (no)
   :Do not change PROTECTION (33);
   :Deploy NORMAL_DEAD
   Access Restrictions (34);
   :Enable System Calls (14);
   :Set-up DAP from AR (15);
   :Apply System Protection (35);
   :Set PC = 2 or 4 (16);
endif
if (is DAP Enabled ? (17)) then (yes)
   :Configure SWJ (18);
else (no)
endif
:Set-up SP (2);
detach
@enduml
```

FLASH BOOT FOR TRAVEO II IP BROS

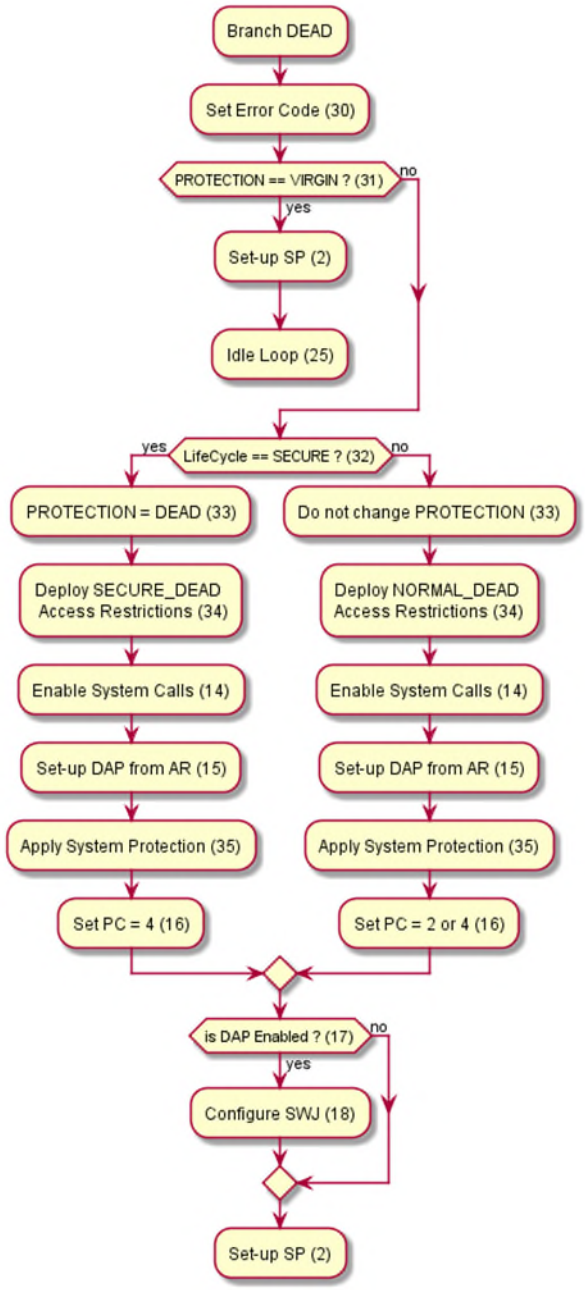


Figure 1.1. Flash Boot Flow (Branch DEAD)

```
@startuml
:Part 2;
:Enable System Calls (14);
:Set-up DAP from AR (15);
:Set PC=2 or 4 (16);
if (is DAP Enabled ? (17)) then (yes)
   :Configure SWJ (18);
```

```
   if (Wake-up from Hibernate ? (19)) then (no)
      :Listen Window (20)
       (default is 20ms);
       if (Test Mode ? (21)) then (yes)
          :Set-up SP (2);
          :Iddle Loop (25);
          detach
       else (no)
       endif
   else (yes)
   endif
else (no)
endif
   if (is Single Core ? (22)) then (yes)
      :Launch CM4 Application (23);
   else (no)
      :Launch CM0 Application (24);
endif
detach
@enduml
```

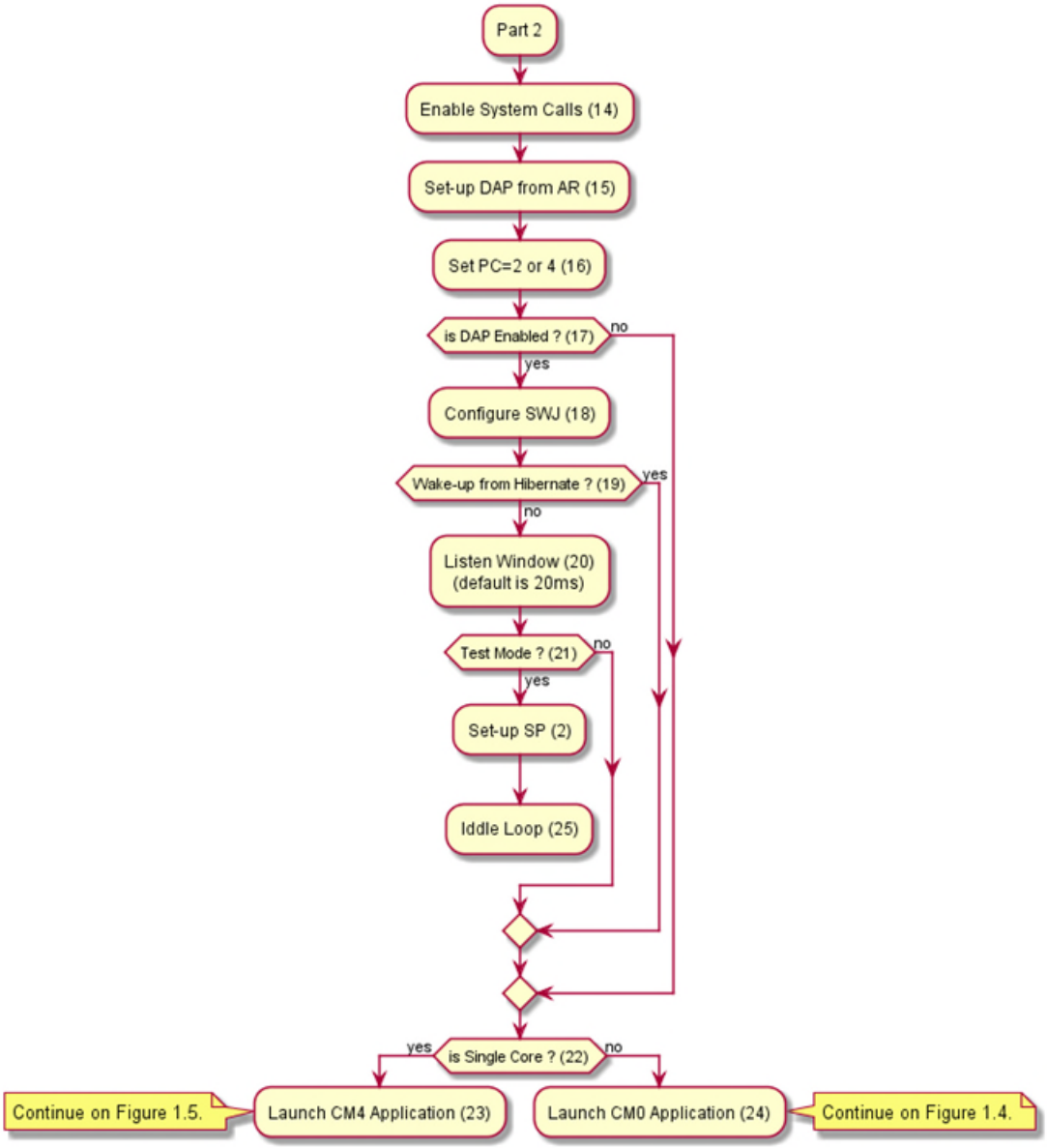


Figure 1.2. Flash Boot Flow (Part 2)

```
@startuml
:Branch Bootloader;
:Disable WDT (40);
:Enable System Calls (14);
:CPUSS.CM0_VECTOR_TABLE_BASE = 0xFFFF_0000
CPUSS.CM4_VECTOR_TABLE_BASE = 0xFFFF_0000 (41);
:Set-up DAP from AR (15);
:Set PC = 2 (16);
if (is DAP Enabled ? (17)) then (yes)
   :Configure SWJ (18);
```

```
   if (Wake-up from Hibernate ? (19)) then (no)
      :Listen Window (20)
        (default is 20ms);
   else (yes)
   endif
   if (Test Mode ? (21)) then (yes)
      :Set-up SP (2);
      :Iddle Loop (25);
      detach
   else (no)
   endif
else (no)
endif
:Launch Bootloader (42);
detach
@enduml
```

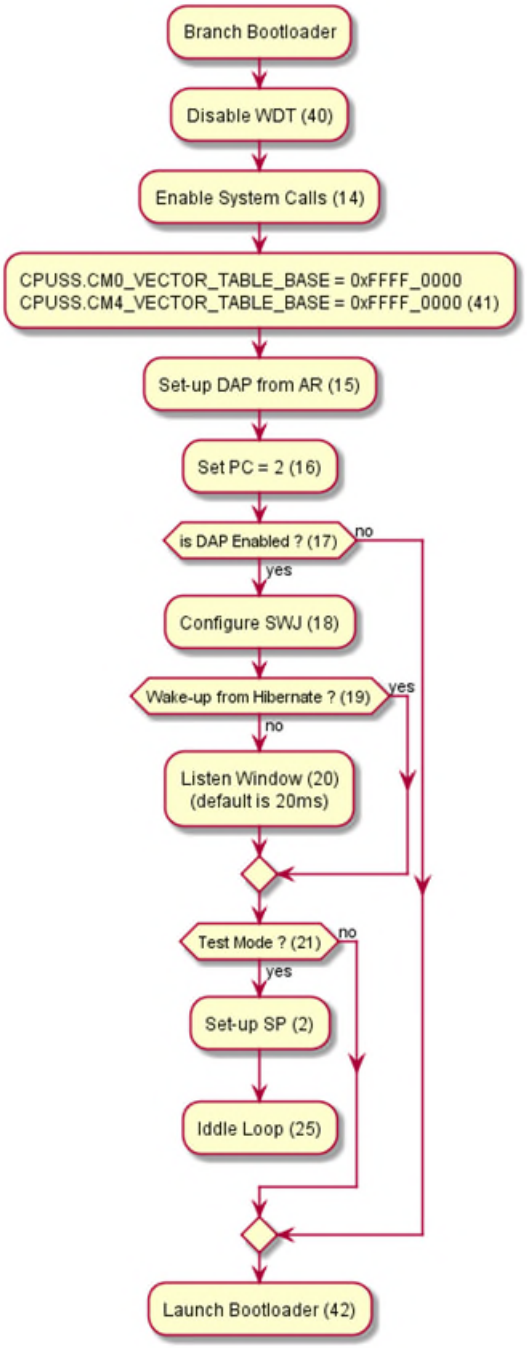


Figure 1.3. Flash Boot Flow (Branch Bootloader)

@startuml
:Launch CM0+
Application;
:CPUSS.CM0_VECTOR_TABLE_BASE = "app addr"
CPUSS.CM4_VECTOR_TABLE_BASE = 0xFFFF_0000 (41);

```
:CM0+ core reset (50);
:Launch user app by ROM boot;
-[dotted]->
:CM0+ user application is launched by ROM boot
            after a CM0+ core reset (51);
detach
@enduml
```

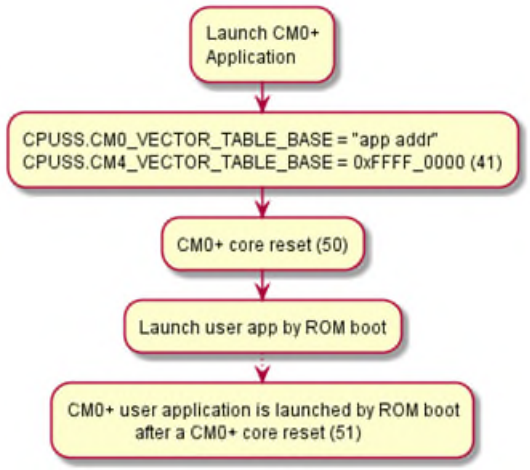


Figure 1.4. Flash Boot Flow (Launch CM0+ App)

```
@startuml
:Launch CM4
Application;
:Set-up SP;
:CPUSS.CM0_VECTOR_TABLE_BASE = 0xFFFF_0000
CPUSS.CM4_VECTOR_TABLE_BASE = "app addr" (41);
:Enable CM4;
repeat
if (is IPC2 lock acquired ? (53)) then (no)
   if (is IPC1 lock acquired ? (53)) then (no)
      :Put CM0+ into Deep Sleep);
   else (yes);
   endif
else (yes);
endif
repeat while(endless loop);
@enduml
```

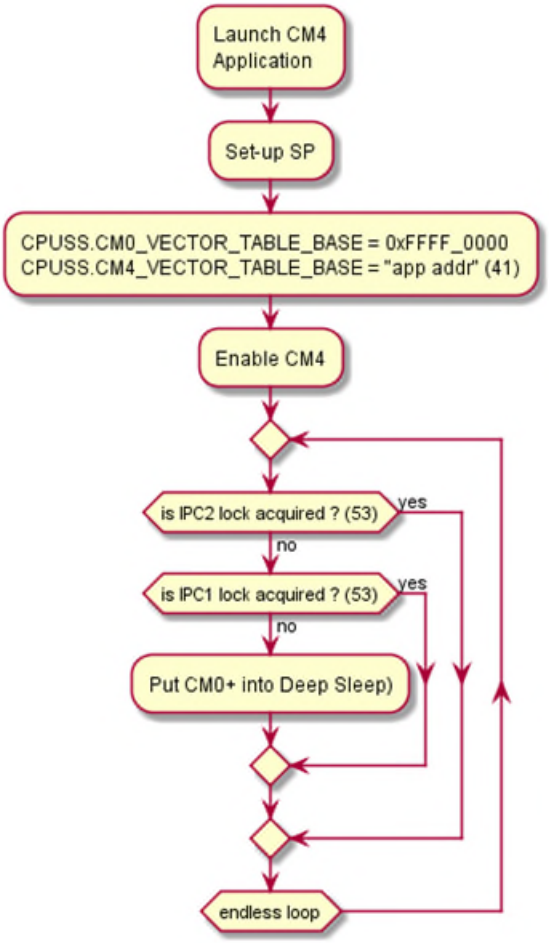


Figure 1.5. Flash Boot Flow (Launch CM4+ App)

8.1.1 Flash Boot summary

This chapter summarizes the Flash boot flow which is illustrated in chapter 8- Theory of Operation

FLASH BOOT main steps _0

The Flash boot is the firmware that runs on the security processor (Cortex-M0+) and is executed after ROM boot has completed basic hardware configuration, trim setting as also validation of the Flashboot image.

FLASH BOOT main steps _1

It validates the user's code in flash, sets up the Debug Access Port (DAP), Enable Systems calls and then jumps to either Secure Image or a User Application depending on the Lifecycle stage, Table of Contents, and application verification.

FLASH BOOT FOR TRAVEO II IP BROS

FLASH BOOT: main steps

Flash boot is a second stage of a secure boot process, it is launched by the ROM boot and has the following goals:

1. Provide the workarounds for the silicon issues which has been discovered after the silicon tapeout. These workarounds are executed at PC=0.
2. Validate TOC2.
3. Setup a debugger configuration: enable/disable DAP, configure SWJ pins, provide a listen window timeout for a debugger to connect.
4. (TVII specific) Implement and launch a one-time bootloader to program data to an empty Main Flash.
5. Authenticate the application if its format is either CySAF or Simplified Secure.
6. Launch a user application or go to an infinite loop after setting respective error code.

FLASH BOOT code location

Flash boot resides in the Supervisory Flash or SFLASH memory.

Flash boot may be updated until the silicon is not converted to NORMAL life-cycle stage.

Most of SFLASH memory can be viewed as OTP (One Time Programmable) that is programmed at Cypress during testing. Some sections of the SFLASH are writable by the customer to store Secure Public Keys and the Table of Context 2 (TOC2) . Trim values used to calibrate different aspects of silicon, such as oscillator and bandgap values are also stored in SFLASH. The latest memory map of the SFLASH can be found in the specific SAS file stored at subversion Link :

Each device has its own SFLASH image that is stored in an SVN folder by this link:
MXS40/Platform/MXS40-SAS/Configuration/

The released and mature silicon families keep the copy of this document in their IP vaults.

FLASH BOOT main steps _2

Flash boot determines if the life cycle state is normal or secure, validate the TOC and optionally validate the firmware before transferring execution to the user CM0+ application code.

The Flash boot is used to ensure that only the code intended for a device from the customer is able to run on a specific platform. Also it ensures that firmware will not execute if it has been modified by a malicious 3rd party. The firmware in Flash is validated using the RSA algorithm. The Public key will be stored in SFLASH and the digital signature is stored in user Flash with the application.

FLASH BOOT main steps _3

Flash boot for Traveo II includes internal CAN and LIN bootloader. The intention of the bootloader is to upload and launch user application named Flash Loader in the SRAM.

FLASH BOOT FOR TRAVEO II IP BROS

The Flash Loader application goal it to program custom application into the User Flash during the OEM serial production with no SWD or JTAG connection.

A user application on CM0+ core is responsible to launch the CM4 application.

8.1.2 Flash boot flow description

This chapter describes in details the Flash boot flow process in Chapter 8- Theory of Operation.

Entry after ROM Boot (1)

FLASH BOOT Overview _step 1: Entry after ROM BOOT
At this point ROM boot has finished all its tasks and transfers the execution for CM0+ code to Flash boot.

Set-up SP (2)

The same Flash boot image is programmed to all the silicon units of the same device family. Within the device family multiple MPNs have different size of SRAM. The SP register value for Flash boot must be at the top of user SRAM minus 2kB. Thus, it is impossible to have SP value know at build time.

FLASH BOOT Overview _step 2: Set-up SP
At the start of Flash boot SP register value is 0. Flash boot sets the value of SP at runtime by reading the CPUSS.WOUNDING and applying this wounding information to the SRAM.

Initialization (3)

FLASH BOOT Overview _step 3.1: Initialization
This function executes a hardware-specific initialization code. Also the Flash boot initializes the SRAM it uses for stack for the device families which support SRAM ECC (i.e. TVII-BE-1M, 2M, and 8M).

FLASH BOOT Overview _step 3.2: Initialization
In this section we apply the workarounds for the silicon issues which are fixed in the Flash boot (sometimes called "patches").

Validate TOC2 (5)

FLASH BOOT Overview _step 5: Validate TOC2
The current procedure to validate TOC2 is to check the following conditions:
SFLASH_TOC2_OBJECT_SIZE <= 512 and SFLASH_TOC2_OBJECT_SIZE >= 8
SFLASH_TOC2_MAGIC_NUMBER == 0x01211220.

See section 8.5.4.2- TOC (Table of Contents) for the details of the TOC2 structure.

Is TOC2 Valid (6)

TOC2 may be in three states:

- VALID, TOC2 structure and CRC are valid.
- ERASED, the first two 32-bit words at the start of TOC2 are equal to the SFLASH erase value and protection mode is either VIRGIN or NORMAL.
  - For SONOS SFLASH the erased value is 0x0000_0000
  - For ECT SFLASH it is 0xFFFF_FFFF.
- CORRUPTED, when both ERASE nor VALID condition are false.

FLASH BOOT Overview _step 6.1: Validate TOC2

If TOC2 state is ERASED then Flash boot uses the default values for all the TOC2 elements instead of reading them from TOC2. The default values are:

- SFLASH_TOC2_FIRST_USER_APP_ADDR is 0x1000_0000 (the start of Flash).
- SFLASH_TOC2_FIRST_USER_APP_FORMAT is 0 (Basic Application Format).
- SFLASH_TOC2_FLAGS 0x0000_0242 for TVII silicon families.

FLASH BOOT Overview _step 6.2: Validate TOC2

The other TOC2 entries shall not be used when TOC2 state is ERASED.

Do Bootloading? (8)

FLASH BOOT Overview _step 8.1: Do Bootloading?

Bootloading triggers in VIRGIN and NORMAL protection modes if all the following conditions are met:

- CPUSS.PROTECTION != SECURE.
- TOC2 state is either ERASED or VALID and SFLASH.TOC2_FLAGS bits FB_BOOTLOADER_DISABLE = 2'b01.
- The first two 32-bit words at 0x1000_0000 are both equal to 0xFFFF_FFFF.

FLASH BOOT Overview _step 8.2: Do Bootloading?

If the bootloader is triggered then Flash boot launches a 1 second wait window for DAP to get connected.

- If the DAP connects to MCU during this wait window the bootloading is not not launched and devices goes into Idle state.
- If the DAP is not connected, the bootloader is launched.

Get App #{0, 1} reset handler (9)

FLASH BOOT Overview _step 9:1: Get Appl {0, 1} Reset Handler

This step may be executed when TOC2 state is either VALID or ERASED (section 8.5.4.4.7- Function ResetHandler).
If TOC2 state is ERASED and CPUSS.PROTECTION=NORMAL then:

7. Application start address is 0x1000_0000
8. Application format is CyBAF.
9. Second application is ignored, thus if a validation of the first application leads to an error the second application is not validated and DEAD branch is executed.

Otherwise TOC2 state is VALID and application parameters are calculated as shown till the end of this section.

FLASH BOOT Overview _step 9.2: Get Appl {0, 1} Reset Handler

Flash boot reads application start address from TOC2 entries:

- SFLASH_TOC2_FIRST_USER_APP_ADDR for App#0.
- SFLASH_TOC2_SECOND_USER_APP_ADDR for App#1.

The application format is stored in TOC2 entries:

- SFLASH_TOC2_FIRST_USER_APP_FORMAT for App #0.
- SFLASH_TOCR_SECOND_USER_APP_FORMAT for App #1

The address of reset handler inside the application depends on the application format. See SAS file *001-98134-RevM-Book2-PartII-BOOT.docx in* 001-98134 describing the application formats.

Is ResetHandler valid? (10)

FLASH BOOT Overview _step 10: Is ResetHandler valid?

Flash boot checks whether the address of the Reset Handler for the user application is inside a valid range.
The valid range is the following: SRAM, SFLASH, Code Flash, WFLASH.

Note The goal of this check is to prevent the hard-fault which otherwise would occur if the reset handler for the user application points to an invalid memory location.

Authenticate App? (11)

FLASH BOOT Overview _step 11: Authenticate App?

Flash boot optionally authenticates a digital signature for the application image based on the TOC2_FLAGS bits APP_AUTH_DISABLE.

Is Public Key Valid (12)

FLASH BOOT Overview _step 12: Is Public Key Valid

The public key structure is filled by the user. Thus, it must be validated to ensure the correctness of the its entries before being used.

Is Digital Signature valid? (13)

FLASH BOOT FOR TRAVEO II IP BROS

The application which is to be launched by Flash boot may be authenticated with RSASSA-PKCS1-v1_5 defined in RFC 3447 The public key used for this operation is stored in the User Public key area of SFLASH. The format of this key is shown below in the Data Structures section.

The application may be in one of three formats, only two of which are validated using a public key. These formats are defined below in the Data Structures section.

FLASH BOOT Overview _step 13.1: Is Digital Signature valid?
Flash boot determines the application format from by TOC part 2 entry at offset +0x10 for App#0 and +0x18 for App#1.

The Crypto block, that may be used for validation, is enabled for the time when the block is used and disabled promptly after application validation is complete to conserve on power.

FLASH BOOT Overview _step 13.2: Is Digital Signature valid?
The diagram below shows the application validation flow.

```
@startuml
partition Application_Bundle {
split
:Binary Code Image;
split again
:Digital Signature;
endsplit
}
split
-> Binary Firmware;
:Binary Code Image (FW)/
:Hash Function
SHA-256|
-> .
**Calculated** Digital
Digest;
split again
-> Digital Signature;
:Digital Signature (HASH)/
:Decrypt
(RSA N-bit)|
note right
Public Key
endnote
-> .
**Decrypted** Digital
Digest;
endsplit
```

:Compare Stored Digest
With Calculated Digest]
if (Do Signatures Match?) then (yes)
: Valid Application;
detach
else (no)
: Invalid Application;
detach
@enduml

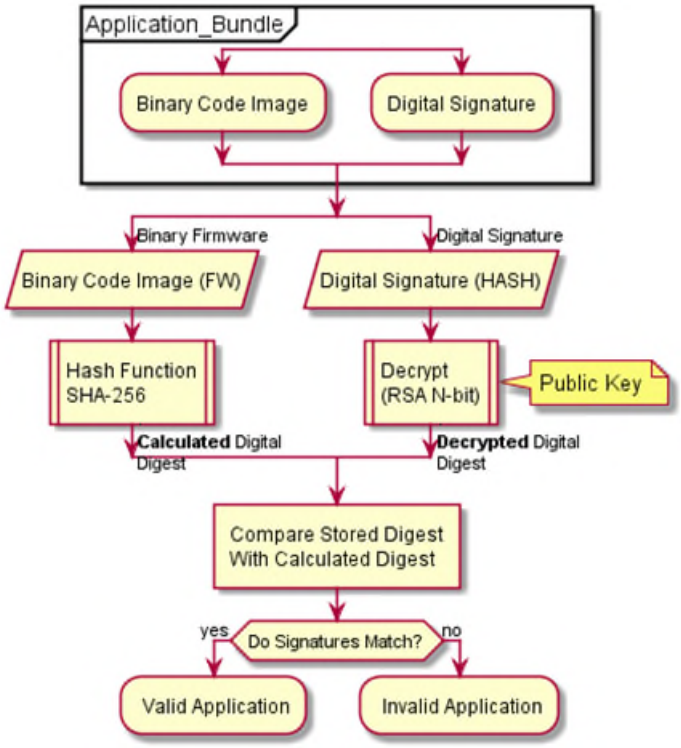


Figure 2. Application Validation

Enable System Calls (14)

FLASH BOOT Overview _step 14: Enable System Calls

At this point the System calls are enabled.

The system calls are functions such as writing to an internal Flash. The function calls are performed via the IPC communication to the CM0+ NMI interrupt. The SROM function *EnableSystemCalls()* is called to enable these system calls.

Set-up DAP from AR (15)

FLASH BOOT FOR TRAVEO II IP BROS

FLASH BOOT Overview _step 15.1: Set-up DAP from AR
Enable or disable DAP based on the current access restrictions (AR).

FLASH BOOT Overview _step 15.2: Set-up DAP from AR
ROM boot function *GetAccessRestrictStruct()* is called to determine which APs (access points, one of CM0+ AP, CM4, and TC) are enabled. Based on this information the proper values are written to CPUSS_AP_CTL register.

Note For a VIRGIN protection mode the set-up of DAP is performed in the ROM boot, thus Flash boot skips this step.

Set PC=2 (16)

ROM boot and Flash boot are being executed in PC=0, system calls are executed in PC=1, all the other code must be executed in PC=2 or higher.

FLASH BOOT Overview _step 16.1: Set Protection Context (PC)
Flash boot is responsible for setting PC=2 before launching a user application or jumping into the idle loop. PC will be set to 2 only in case when Protection state is not 'DEAD'. In DEAD PC will be set to 4.

FLASH BOOT Overview _step 16.2: Set Protection Context (PC)
Flash boot sets PC=4 in SECURE_DEAD branch (when life-cycle stage is SECURE and protection mode is DEAD).

Is DAP enabled (17)

FLASH BOOT Overview _step 17.1: Is DAP enabled
For DEAD and Bootloader branches the result of step (17) is TRUE if CPUSS.AP_CTL enables DAP.

FLASH BOOT Overview _step 17.2: Is DAP enabled
For a common branch (the one which ends by launching a user application) there is an additional check to determine if DAP is enabled. This check reads the TOC2_FLAG bit SWJ_PIN_CTL. If TOC2_FLAGS.SWJ_PINS_CTL is set and CPUSS.AP_CTL enables DAP then the result of step (17) is TRUE.

Configure SWJ (18)

FLASH BOOT Overview _step 18: Configure SWJ
Flash boot uses ROM boot function *ConfigureSWJ()* if it is implemented for the device family, otherwise, this function is implemented in the Flash boot code base. This function configures the GPIO pins to work in SWJ mode. All the JTAG pins must be configured, except the TRST pin for the device families on which it has a silicon issue.

Wake-up from Hibernate (19)

FLASH BOOT Overview _step 19: Wake-Up from Hibernate
If the reason for a reset was "wake from Hibernate", skip the wait window and test mode check.

Listen window (20)

FLASH BOOT FOR TRAVEO II IP BROS

FLASH BOOT Overview _step 20: Listen window

The CPU delays execution for a period of time to allow the debug hardware to acquire the CM0+. The default is 20 ms, but the user may set other delay options. If the Listen window is not needed for the project, the user may set Listen window timeout to 0 ms.

This delay allows the debug hardware to acquire the debug interface before any user code is executed, it helps recovering the device in which a user code re-purposes SWJ pins and goes wild.

Test Mode? (21)

FLASH BOOT Overview _step 21: Test Mode

During the listen window delay, the firmware checks if the SRSS_TST_MODE register has either TEST_MODE or TEST_KEY_DFT_EN bit set. If either bit is set, execution is transferred to an endless loop in SROM. This is done by calling the ROM boot function *BusyWaitLoop()*.

Some programmers use Listen window and set a Test Mode bit to perform either programming or debugging tasks.

Is Single-Core? (22)

FLASH BOOT Overview _step 22: Is Single-Core?

Detects if a MCU is a single-core one. A single-core MCU does not allow a user code to be executed on CM0+.
Flash boot determines if MCU is a single-core by reading SFLASH_SINGLE_CORE_WOUND.

Launch CM4 application (23)

This section is for PSoC6 only. We intentionally left it here for future use.

Flash boot enables the CM4 and sets CPUSS.CM4_VECTOR_TABLE_BASE to point at CM4 interrupts vector table.

Then CM0+ is executing an idle loop in Flash boot. Inside this loop the code polls until neither IPC1 nor IPC2 is locked. It then puts CM0+ into Deep-Sleep power mode.

The goal to check if IPC1 or IPC2 is in a locked state is to forbid placing CM0+ into Deep-Sleep until the system calls are finished.

Launch CM0+ application (24)

FLASH BOOT Overview _step 24: Launch CM0+ application

The procedure to launch a user application is:

1. For CM4: Set CPUSS_CM4_VECTOR_TABLE_BASE to 0xFFFF_0000.
   For CM7: Do not change CMPUSS_CM7_x_VECTOR_TABLE_BASE.
2. Set CPUSS_CM0_VECTOR_TABLE_BASE to the start of the user application interrupts vector table.

3. Perform a CM0+ core reset.
4. After a core-reset is performed a ROM boot is launched (on CM0+).
5. ROM boot checks if CPUSS_PROTECTION != 0, which means ROM boot is launched on CM0+ after a core-reset.
6. If (5) is true, ROM boot sets SP and PC register values from the user interrupt vector table. The address of a user application interrupt vector table is stored at step (1) to CPUSS_CM0_VECTOR_TABLE_BASE.
7. When ROM boot sets PC register value with the user reset handler address, user code starts executing

Idle Loop (25)

FLASH BOOT Overview _step 25.1: Idle Loop

Before going to an idle loop the Flash boot sets the CPUSS_CM0_VECTOR_TABLE_BASE MMIO register to 0xFFFF_0000. This is required by CDT 282643.

FLASH BOOT Overview _step 25.2: Idle Loop

Then CM0+ core is placed into a sleep power mode by calling ROM boot function *BusyWaitLoop()*.

Note Any interrupt (IPC system call or the other interrupt source) may wake up the device so the access restrictions and the other security settings should be properly configured *by the user* for each life-cycle stage.

Branch DEAD (8)

FLASH BOOT Overview _step 26: Branch DEAD

Flash boot goes into a DEAD branch if it detects any error.
The list of the required errors is provided in section **8.1.2.1- Set Error Code (30)**.

The reasons for entering the dead state could be:

TOC2 Issues	tocAddr == CY_FB_TOC2_INVALID in Secure State
	tocAddr == CY_FB_TOC2_ERASED in Secure State
	Cy_FB_GetTocFlagsClockMhz "1" is a special number for invalid delay value
Verify Application Issues	Cy_FB_IsValidKey(tocAddr, publicKeyAddr) == false
	If application verification has failed
	if (vectorTableBase == CY_FB_INVALID_VECTOR_TABLE)
Wounding	Default value for Cy_FB_GetMainFlashSize() and GetWFlashSize()
Bootloading	Multiple error scenarious during bootloading

Branch Bootloader (28)

If the bootloader feature is enabled for the device family and the bootloader launch condition is triggered then Flash boot launches a bootloader by going into this branch. The implementation may implement this branch either as a function call or as launching an application.

FLASH BOOT FOR TRAVEO II IP BROS

Interrupts and System calls (29)

FLASH BOOT Overview _step 29: Interrupts and System Calls

Flash boot supports patching the system calls using the system call patch table. Flash boot does not use any interrupts.

PROTECTION = VIRGIN? (31)

FLASH BOOT Overview _step 31: PROTECTION=VIRGIN?

CPUSS_PROTECTION MMIO register value is being compared to the wished protection mode.

LifeCycle = SECURE? (32)

A life-cycle stage is stored in EFUSE. SAS file MXS40-IP-EFUSE.xlsm in folder: MXS40/Platform/MXS40-SAS/Configuration/provides the description for the EFUSE bits for each device families.

Note Life-cycle stage is not the same as protection mode. In this case, SECURE_WITH_DEBUG life-cycle stage is not equal to SECURE life-cycle stage, but for both of them the protection mode equals to SECURE.

PROTECTION = DEAD (33)

FLASH BOOT Overview _step 33: PROTECTION=DEAD

Flash boot sets CPUSS_PROTECTION to DEAD in the DEAD branch only for SECURE life-cycle stage. For SECURE_WITH_DEBUG, NORMAL, and the other life-cycle stages Flash boot keeps the existing protection mode.

Deploy Access Restrictions (34)

FLASH BOOT Overview _step 34: Deploy Access Restrictions

Flash boot deploys the access restrictions which apply to the new protection mode.
NORMAL_DEAD access restrictions are applied in the case of entering DEAD branch from NORMAL, NORMAL_PROVISIONED, or SECURE_WITH_DEBUG life-cycle stages.
SECURE_DEAD access restrictions are applied in the case of entering DEAD branch from SECURE life-cycle stage.

Assess restrictions are applied by calling ROM boot function *RestrictAccess()*.

Apply System Protection (35)

FLASH BOOT Overview _step 35: Apply System Protection

System Protection Settings are applied. Usually they are applied by ROM boot code before entering Flash boot. But in the case of DEAD branch, the system protection settings may be changed, thus, Flash boot needs to call the ROM boot function *ApplyProtectionSettings()* to reconfigure them.

Disable WDT (40)

FLASH BOOT FOR TRAVEO II IP BROS

FLASH BOOT Overview _step 40: Disable WDT

Bootloader may run for a significantly longer time then WDT timeout, thus WDT must be either a periodically reset or disabled at the start of Bootloader.

Set VECTOR_TABLE_BASE (41)

FLASH BOOT Overview _step 41: Set VECTOR_TABLE_BASE

CPUSS.CM0_VECTOR_TABLE_BASE and CPUSS.CM4_VECTOR_TABLE_BASE registers must be set to 0xFFFF_0000 value to show the debugger that no user application is running.

Launch Bootloader (42)

FLASH BOOT Overview _step 42: Launch Bootloader

A bootloader firmware is launched. Bootloader firmware is a part of Flash boot. Thus, launching it may be as simple as calling a function.

CM0+ core reset (50)

FLASH BOOT Overview _step 50: CM0+ core reset

Flash boot resets CM0+ core by writing to CPUSS_CM0_CTL register.

After the write CM0+ must enter a sleep mode and wait until the core is reset.

Launch a user app by ROM boot (51)

FLASH BOOT Overview _step 51: Launch a user app by ROM boot

Flash boot performs the following to switch to the user application on CM0+:

8. Flash boot sets CPUSS.CM0_VECTOR_TABLE_BASE value with an address of the user CM0+ interrupt vector table.
9. Flash boot performs CM0+ core reset.
10. ROM boot starts up, tests if CPUSS.PROTECTION <> 0 which means ROM boot is launched from a “software reset”.
11. If (3) succeeds, ROM boot sets SP and PC register values from the users interrupt vector table, which is read out from CPUSS.CM0_VECTOR_TABLE_BASE.
12. When ROM boot sets PC register value with the user’s reset handler address, a user code starts executin

8.1.2.1 Set Error Code (30)

FLASH BOOT Overview _step 30: Set Error Code

Flash boot sets an error code into IPC_STRUCT[*TC*].DATA0 register.
TC=2 for MCUs with 1 CM4/CM7 cores,
TC=3 for MCUs with 2 CM4/CM7 cores.
For TVII-BE-1M ** it also sets an error code into the first word of the last available SRAM macro.

Error Name	Value	**Description**
CY_FB_STATUS_SUCCESS	0xA100_0100	Success status value.
CY_FB_STATUS_BUSY_WAIT_LOOP	0xA100_0101	Debugger probe acquired the device in Test Mode. The Flash boot to entered a busy wait loop.
CY_FB_STATUS_BOOTLOADING	0xA100_0101	Bootloading in progress
CY_FB_STATUS_BTLD_OK	0xA100_0102	Bootloading finished successfully
CY_FB_ERROR_INVALID_APP_SIGN	0xF100_0100	App signature validation failed for the device families where Flash boot launches only one application from TOC2. Either app structure or a digital signature is invalid for the device families for which Flash boot may launch either of two apps in TOC2.
CY_FB_ERROR_INVALID_TOC	0xF100_0101	Empty or Invalid TOC
CY_FB_ERROR_INVALID_KEY	0xF100_0102	Invalid Public Key
CY_FB_ERROR_TOC_DATA_DELAY	0xF100_0105	TOC contains invalid listen window delay
CY_FB_ERROR_INVALID_APP0_DATA	0xF100_0107	App structure is invalid, for the device families where Flash

		boot may launch only one app from TOC2.
CY_FB_ERROR_INVALID_PARAM	0xF100_0109	Invalid parameter value.
CY_FB_ERROR_BOOT_LIN_INIT	0xF100_0141	Bootloader error, LIN initialization failed
CY_FB_ERROR_BOOT_LIN_SET_CMD	0xF100_0142	Bootloader error, LinSetCmd() failed
CY_FB_ERROR_BOOT_CAN_INIT	0xF100_0143	Bootloader error, CAN initialization failure
CY_FB_ERROR_BOOT_SECURE	0xF100_0144	Bootloader launched while CPUSS.PROTECTION=SECURE
CY_FB_ERROR_ECC	0xF1001000	The ECC error happened. This error code is OR-ed with other error codes.

8.2 Solution Implementation Overview

The Flash boot is implemented according to the flow described in8- Theory of Operation.

8.2.1 Third Party Sources / Open Source

IAR workbench v8.4 is used to develop a Traveo II Flash boot firmware.

8.2.2 Driver and/or Component Dependencies: N/A

8.2.3 Silicon Dependencies

This firmware solution is built to support Traveo II device family.

Flash boot uses hidden MMIO registers to support device programming, FLL clock configuration, accessing Lifecycle stage EFUSE address etc.

E.g. CPUSS.WOUNDING, CPUSS.AP_CTL.

FLASH BOOT FOR TRAVEO II IP BROS

SRSS configuration is not possible on a PSVP so Flash boot generates two output targets – silicon and PSVP. The difference between targets are an absence of SRSS configuration in PSVP target, and clock frequencies.

8.2.4 Silicon IP blocks Dependencies

The Flash boot firmware is based on mxs40cpuss, mxs40srss, mxs40eFuse, mxs40crypto, mxs40ipc, mxs40can, mxs40lin SoftIP blocks.

8.2.5 Shared Hardware Resources

During the Flash boot run time, there are no other consumers for the available resources.

8.2.6 Traveo II debugging probe support

Cypress Programmer 1.0 supports programming Main Flash, SFLASH, and WFLASH with the following probes:

- MiniProg3/4
- J-Link

The following 3rd party vendors are implementing debugger/programmer for TVII:

- Lauterbach
- iSystem
- IAR
- GHS (Green Hills)

8.2.7 Low-power Modes (LPM) Support & Behavior

FLASH BOOT: Active Mode
Flash boot always runs in Active mode.

TOC 2 contains an option for the user to choose a core clock frequency (8 MHz, 25 MHz, 50 MHz, and the highest valid “clk_slow” frequency) that directly correlates with the power consumption.

8.3 Driver Implementation Overview: N/A

8.4 Component Implementation Overview: N/A

8.5 Firmware Implementation Overview

8.5.1 Public Export Interface

Flash boot source code is not available to the customers unless under NDA, however there are several shared functions that are used by ROM boot and could be used by the user application – Secure image,

TEE, SHE, HSM, etc. The table of pointers to these functions is placed in a fixed location – address(FLASH_BOOT_OBJECT_SIZE0) + 0x40..

FLASH BOOT Public Export Interface _1

The table of pointers to these functions is placed in a fixed location – address(FLASH_BOOT_OBJECT_SIZE0) + 0x40..

FLASH BOOT Public Export Interface _2

Note These functions are reentrant and directly callable by either CM0+ or CM4.

However, they are intended for Cypress internal use and are not guaranteed to be functional for an external use cases, i.e. at some point of time SMPU may be used to reject the read or execute access at PC > 0.

FLASH BOOT Public Export Interface _3

Offset from the start of Flash boot code	Function Name	Comment
+ 0x40	Cy_FB_VerifyApplication	Validates the application signature with RSASSA-PKCS1-v1.5 (2048 bit).
+ 0x44	Cy_FB_IsValidKey	Validates the public key.
+ 0x48	Cy_FB_Crc16ccitt	Computes CRC-16-CCITT.
+ 0x4C	Cy_FB_ValidateToc	Validate TOC2.

8.5.1.1 Cy_FB_VerifyApplication

FLASH BOOT Public Export Interface: Cy_FB_VerifyApplication

Function Description:	**Verifies the application digital secure signature.**
Parameters:	uint32_t address : The start address of an application area to be verified with a secure signature. uint32_t length : The length of an area to be verified. uint32_t signature: The starting address of the signature section inside the application in the flash. cy_stc_crypto_rsa_pub_key_t publicKey: The pointer to a public key structure.
Return Value:	uint32_t: 0 – if the digital secure signature verification of the application fails; 1 – if the digital secure signature verification succeeds.

8.5.1.2 Cy_FB_IsValidKey

FLASH BOOT Public Export Interface: Cy_FB_IsValidKey

Function Description:	**Checks whether the public key structure is valid.**
Parameters:	uint32_t address: The address of TOC2 cy_stc_crypto_rsa_pub_key_t *publicKey: the pointer to public key structure.
Return Value:	uint32_t: 1 if – the public key is valid, 0 – if the public key is invalid.

8.5.1.3 Cy_FB_Crc16ccitt

FLASH BOOT Public Export Interface: Cy_FB_Crc16ccitt

Function Description:	**Computes CRC-16-CCITT with polynomial =0x1021, initial value=0xfff.**
Parameters:	uint32_t address: The address of a data block for CRC calculation. It shall be 32-bit aligned. uint32_t length: Length of a memory block to calculate CRC.
Return Value:	uint32_t: The calculated CRC-16-CCITT for the data.

8.5.1.4 Cy_FB_ValidateToc

FLASH BOOT Public Export Interface: Cy_FB_ValidateToc

Function Description:	**Checks if TOC2 is valid**
Parameters:	uint32_t tocAddress: Address of the Table of Contents (TOC)
Return Value:	uint32_t: CY_FB_TOC_INVALID: If TOC2 in invalid CY_FB_TOC_EMPTY: If TOC2 is empty. The address of the valid TOC2 if the TOC2 is valid

Note These functions are reentrant and directly callable by either CM0+ or CM4. However, they are intended for Cypress internal use and are not guaranteed to be functional for an external use cases, i.e. at some point of time SMPU may be used to reject the read or execute access at PC > 0.

8.5.2 Public Import Interface

Flash boot calls a few ROM boot functions. This strategy gives these benefits:

a) Re-usage of existing tested code.

b) Additional testing of the ROM boot code by Flash boot.

c) The reduction of the Flash boot code size.

FLASH BOOT Public Import Interfaces

ROM boot contains a table of pointers to the functions that are called by Flash boot at address 0x00000D00. The following table contains a list of ROM boot shared functions. Original list is taken from 002-03298 *H section "**Shared Function Table**".

Memory Location	Function Name	Comment
0x00000D00	EnableSystemCalls()	Enables the system calls
0x00000D04	ConfigureSWJ()	Configures the SWD and JTAG pins, sets an AP_CTL value
0x00000D08	BusyWaitLoop()	Executes a busy wait loop, the MCU program counter (PC)

		register is in the ROM code
0x00000D0C	SetUpEfuseRead()	Sets up an EFUSE state machine to allow the EFUSE read operation
0x00000D10	CloseEfuseRead()	Closes the EFUSE state machine, no further EFUSE read operations allowed until the next call to SetUpEfuseRead()
0x00000D14	ReadEfuseByte()	Returns the value read out of the EFUSE byte
0x00000D18	EnterDeadState()	Enters the DEAD state
0x00000D1C	FlashWriteProxy()	A proxy function used by non CM0+ cores to perform a blocking flash write
0x00000D20	NMIHandler()	The NMI handler, typically executed by performing system calls
0x00000D24	ConfigureSWJ_NoAP()	Configures SWJ and JTAG pins without AP_CTL initialization
0x00000D28	ApplySystemProtection()	Configures protection units
0x00000D2C	AssignAllPC_Exit()	Assigns all PC (Protection Context) then goes to a busy wait loop

0x00000D34	RestrictAccess()	This functions enforces restrictions based on the encoding in SFLASH or eFUSE.
0x00000D38	GetAccessRestrictStruct()	This function updates the passed access_restrict_struct with the corresponding content for SFLASH/eFuse based on protection state
0x00000D3C	SetPPU_PC0AccessOnly	This function sets the passed fixed PPU NR to allow access only to PC0 and read only for all other PCs
0x00000D40	ConfigureBootProgPPU	Configures the passed programmable PPU to not allow access to any PC
0x00000D44	ConfigureBootProgPPU_PC2Only	Configures the passed programmable PPU to not allow access to PC = 2
0x00000D48	BlowFuseByteSub	Programs writes a particular byte in eFuse
0x00000D4C	GetFactoryFuseGroupZeros	Returns numbers of zeros in FACTORY eFuse group
0x00000D50	BlowAndCheckFuseBit	For TVll calls the BlowFuseBitSub() and validates the

		programmed bit. In case of failure repeats programming with a longer program strobe. The validation is only occurred for Bootrow fuses. For PSoC 6 the function just calls the BlowFuseBitSub().
0x00000D54	ProgramWorkFlash	This is a syscall to program WorkFlash
0x00000D58	ConfigureBootProgPPU_PC1ReadPCxAll	Configures the passed programmable PPU to not allow write access to any PC above 1

8.5.2.1 EnableSystemCalls

Function Description:	**Enables the system calls**
Parameters:	None
Return Value:	None

8.5.2.2 ConfigureSWJ

Function Description:	**Configures SWD and JTAG pins, sets an AP_CTL value**
Parameters:	None
Return Value:	None

8.5.2.3 BusyWaitLoop

Function Description:	**Executes a busy wait loop, the MCU program counter register is in the ROM code**
Parameters:	None
Return Value:	This function does not return

8.5.2.4 SetUpEfuseRead

Function Description:	**Sets up an EFUSE state machine to allow the EFUSE read operation**
Parameters:	None
Return Value:	None

8.5.2.5 CloseEfuseRead

Function Description:	**Closes the EFUSE state machine, no further EFUSE read operation allowed until the next call to SetUpEfuseRead()**
Parameters:	None
Return Value:	None

8.5.2.6 ReadEfuseByte

Function Description:	**Returns the value read out of the EFUSE data byte**
Parameters:	int32_t address: The full EFUSE byte address, “0x402C0800 + byte_offset” from MXS40-IP-EFUSE.xls
Return Value:	uint8_t: The EFUSE data byte value

8.5.2.7 EnterDeadState

Function Description:	**Enters the DEAD mode**
Parameters:	None
Return Value:	This function does not return

8.5.2.8 FlashWriteProxy

Not used by Flash boot. Used by Flash driver for SDL to perform a blocking flash write on non CM0+ MCU cores.

8.5.2.9 NMIHandler

The NMI handler in the ROM code. Not used by Flash boot. A user secure application, i.e. Secure Image, SHE, HSM, TEE may use this function.

8.5.2.10 ConfigureSWJ_NoAP

Function Description:	**Configures SWJ and JTAG pins without AP_CTL initialization**
Parameters:	None
Return Value:	None

8.5.2.11 ApplySystemProtection

Function Description:	**Configures protection units**
Parameters:	None
Return Value:	None

8.5.2.12 AssignAllPC_Exit

Function Description:	**Assigns all PC (Protection Context) to PC=4 then goes to a busy wait loop**
Parameters:	None
Return Value:	This function does not return

8.5.2.13 RestrictAccess

Function Description:	This functions enforces FLASH restrictions based on the encoding in eFUSE
Parameters:	uint32 lProtState
Return Value:	This function does not return

8.5.2.14 GetAccessRestrictStruct

Function Description:	This function updates the passed access_restrict_struct with the corresponding content for SFLASH/eFuse based on protection state
Parameters:	acccess_restrict_struct_s* - pointer to access restrict struct * uint32 lProtState - lifecycle stage for which restriction is to be applied
Return Value:	This function does not return

8.5.2.15 SetPPU_PC0AccessOnly

Function Description:	This function sets the passed fixed PPU NR to allow access only to PC0 and read only for all other PCs
Parameters:	uint32 lPPU_NR uint32 lReadAllowed - 0,PROT_NOT_ALLOWED - Reads not allowed 1,PROT_ALLOWED - Reads allowed
Return Value:	This function does not return

8.5.2.16 ConfigureBootProgPPU

Function Description:	Configures the passed programmable PPU to not allow access to any PC
Parameters:	None
Return Value:	This function does not return

8.5.2.17 ConfigureBootProgPPU_PC2Only

Function Description:	Configures the passed programmable PPU to not allow access to PC = 2
Parameters:	uint8 cPPU_ID - PPU ID to configure uint32 lAddr - start of memory region that needs to be protected uint8 cRegionSize - memory region size that needs to be protected
Return Value:	This function does not return

8.5.2.18 BlowFuseByteSub

Function Description:	Programs writes a particular byte in eFuse
Parameters:	uint32 lMacroAddr uint32 lByteAddr
Return Value:	This function does not return

8.5.2.19 GetFactoryFuseGroupZeros

Function Description:	Returns CRC of FACTORY eFuse group
Parameters:	None
Return Value:	uint32 lResult - Number of zeros the group

8.5.2.20 BlowAndCheckFuseBit

Function Description:	For TVII calls the BlowFuseBitSub() and validates the programmed bit. In case of failure repeats programming with a longer program strobe. The validation is only occurred for Bootrow fuses. For PSoC 6 the function just calls the BlowFuseBitSub().
Parameters:	uint32 * lMacroAddr,lByteAddr,lBitAddr
Return Value:	status: 0 - function executed successfully; 1 - failed to blow specified fuse bit.

8.5.2.21 ProgramWorkFlash

Function Description:	This is a syscall to program WorkFlash
Parameters:	uint32 ipcStruct
Return Value:	This function does not return

8.5.2.22 ConfigureBootProgPPU_PC1ReadPCxAll

Function Description:	Configures the passed programmable PPU to not allow write access to any PC above 1
Parameters:	uint32 ipcStruct
Return Value:	uint8 cPPU_ID - PPU ID to configure uint32 lAddr - start of memory region that needs to be protected uint8 cRegionSize - memory region size that needs to be protected

8.5.3 API Memory Usage

8.5.3.1 Compile-time stack usage calculation

FLASH BOOT Compiler

Flash boot for All Traveo II devices uses IAR compiler.

FLASH BOOT Linker

IAR linker has an option to calculate the stack usage for each function call in the function call tree at link time. The results of this calculation are stored in the output folder into a file named *stack_usage_<version>.txt* and *stack_usage_<version>_psvp.txt*.

IAR linker flags to enable stack usage calculations:

```
--stack_usage_control stack_control.txt --log call_graph --log_file
./out/stack_usage_$(SUFFIX).txt
```

IAR may not calculate stack usage for some functions, the list of these functions and what to do with them is configured in stack usage control file, a file named *stack_control.txt*.

8.5.4 Internal Firmware Implementation Details

FLASH BOOT FOR TRAVEO II IP BROS

User application may be in either Cypress Secure Application Format (CySAF), Simplified Application Format, or Basic Application Format. The description for these formats is provided in 001-98134, SAS book 2 BOOT, section 1.11.1 FLASH Boot Layout.

8.5.4.1 Bootloading

Traveo II Architecture TRM (002-19314 section named “Flash boot”) contains customer visible part of Bootloader implementation:

- Bootloader packet structure.
- Bootloader commands
- CAN and LIN communication description
- CAN configuration (pins, peripheral configuration)
- LIN configuration (pins, peripheral configuration, switching to Fast Mode)

Note The latest version of 002-19314 is stored on SVN, Flash boot section address is:

B-E Family

https://svn.design.cypress.com:18080/svn/chips/trunk/s40/MXS40/Traveo-II/TVII-B-E-Family/TVII-B-E-1M/WPP/TRM/

B-H Family

https://svn.design.cypress.com:18080/svn/chips/trunk/s40/MXS40/Traveo-II/TVII-B-H-Family/TVII-B-H-8M/WPP/TRM

Cluster Family

https://svn.design.cypress.com:18080/svn/chips/trunk/s40/MXS40/Traveo-II/TVII-C-Family/TVII-C-2D-6M/WPP/TRM

Bootloader SDK is used for implementing the bootloader. Bootloader SDK files from PDL 3.0.1 were ported to TVII-B-E-1M inside the Flash boot implementation. See 002-12095 and 002-13924 for the Bootloader SDK requirements and documentation.

Bootloader Overview

Flash boot is designed to update user application in the Code Flash with the algorithm described on Figure 3.@startuml

```
:Flash Boot Start;
if (Run Bootloader ? (A)) then (yes)
   partition Flashboot
   repeat
```

```
      :Run Internal Bootloader (B);
      :Download "Flash Loader" into RAM (C);
   repeat while (is Flash Loader valid? (D)) is (no)
   ->yes;
   :Switch to Flash Loader (Located in RAM) (E);
   }
   -[dotted]->
   partition Flash_Loader
   :Flash Loader Start;
   repeat
      :Download App into User Flash (F);
      repeat while (is User App #1 Valid? (G)) is (no)
   ->yes;
   :Switch to User App #1;
   }
   -[dotted]->
   partition UserApp#1
   :User App #1 Start (in Flash);
   :May Contain Bootloader (H);
   :Switch to User App #2 (I);
   }
   detach
else (no)
   :Switch to user App #1 (in User Flash);
   detach
endif
@enduml
```

FLASH BOOT FOR TRAVEO II IP BROS

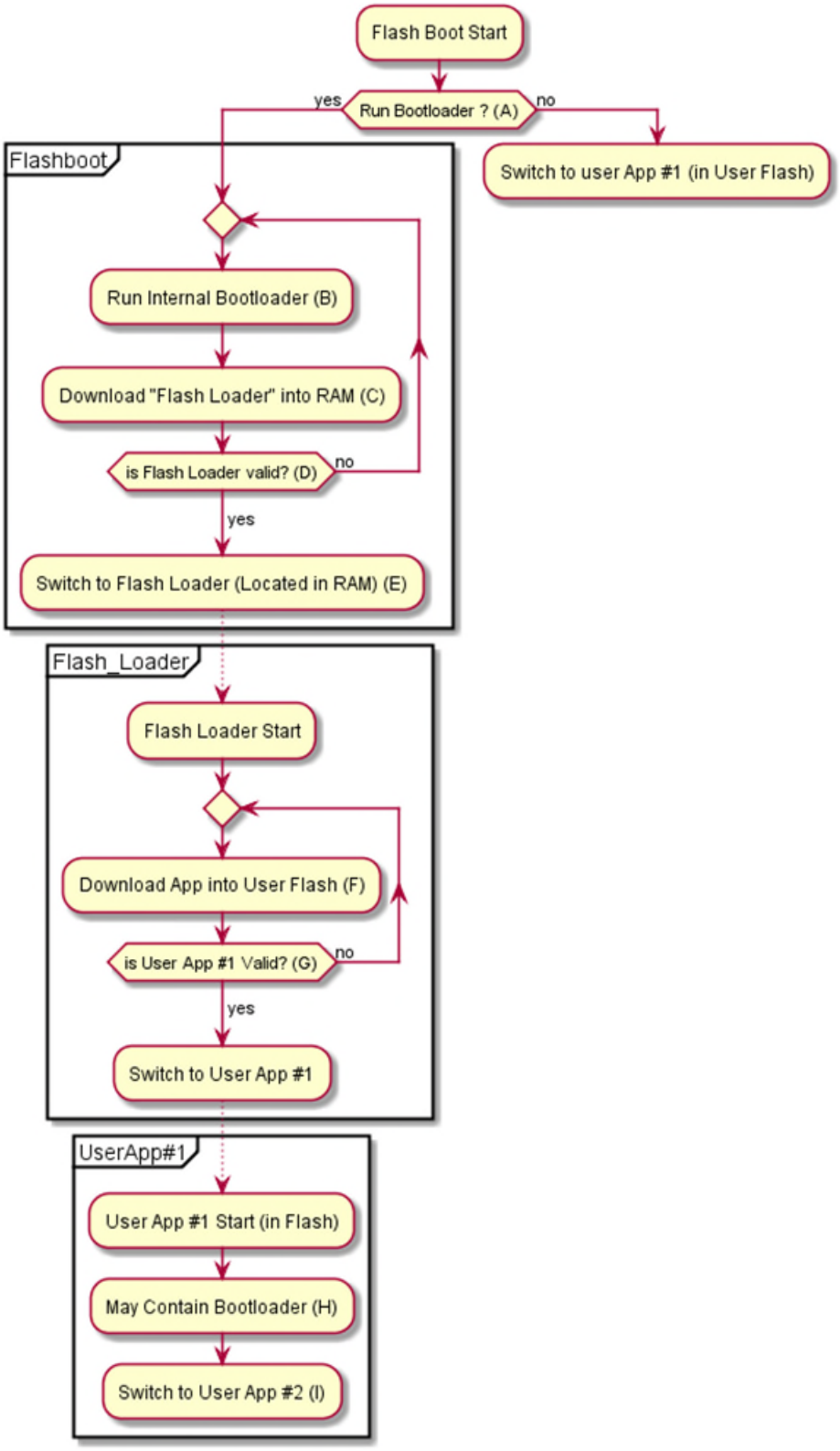


Figure 3. Startup and Bootloading sequence

Bootloader Overview _step (A): Run Bootloader?
(A) Flash boot checks if the internal bootloader should be run.

Bootloader Overview _step (E): Launch the Flash Loader
(D) The Internal bootloader is a part of the Flash boot firmware that has a goal to download the Flash Loader into SRAM (C) and launch it (E).

First 2K is reserved for SROM private SRAM. For FB and SROM stack there is a 4kB dedicated at end of SRAM minus 6KB. The Flash Loader must locates after this region.

Note that the Flash Loader image does not require an encryption because Flash Loader is uploaded into the device by the OEM on their factory setup.

Flash Loader does not require a secure signature for the same reason as for not requiring an encryption. The Flash Loader application format is Basic Application Format with CRC-32C appended to the end of it, the same format is used by Bootloader SDK unsecure applications.

Bootloader Overview: CRC
CRC-32C hash is used for Flash Loader image integrity check only.

Bootloader Overview: Image Storage
The image may be stored anywhere in the SRAM, with the only limitation that it should not overlap with Flash boot stack. that starts at the end of SRAM minus 2KB. The stackstarts at end of SRAM and occupies SRAM location in descending order. And has reserved space until end of SRAM – 4KB.

Bootloader Overview: Set App Metadata
Flash boot bootloader receives the start address and the length of the application from the data of the *Set App Metadata* bootloader command which is the second bootloader command to be sent from the bootloading host to the device.

Bootloader Overview _step (F): Download the user application
(F) The Flash Loader downloads a user application through CAN and LIN communication and stores it into Code Flash, WFlash or any other external memory it supports.

Bootloader Overview _step (G): Check user application integrity
(G) The user application is verified for integrity by the Flash Loader.
If the user application signature verification fails, the Flash loader tries to restart the bootloading and receive a new image.

(H) The user application itself may or may not contain a bootloader. It is up to the user.

Note that only the Flash boot part of the bootloading sequence (A) to (E) is developed as Flash boot firmware, the rest of the sequence is developed by the user.

Bootloader internal: Flowchart

Figure 4 shows the execution flow of the internal bootloader that is executed by the Flash boot.
The Bootloader packets structure is documented in the AN213924 (002-13924 Bootloader SDK User Guide).
The Bootloader SDK requirements are documented in 002-12095 (Bootloader SDK SEROS).
The Bootloading file format (CYACD2) is described in DESP-36 and in AN213924 (002-13924 Bootloader SDK User Guide).@startuml

```
:Flash Boot Start;
repeat
:Configure CAN (1);
if (Enter Bootloader Received ? (2)) then (no)
   :Configure LIN (3);
else (yes)
endif
repeat while (Enter Bootloader Received ? (4)) is (no) not (yes)
:Initialize Bootloader State (5);
repeat
:Wait For Command (6);
if (PROTECTION == SECURE ? (6.1)) then (yes)
   :Branch DEAD;
   detach;
else (no)
 fork
 :Other commands_1;
 note left
 Continue on Figure 4.1.
 end note
 fork again
   :Command
 "Program Data" (14);
   if (is Command Data Valid ? (15)) then (yes)
      :Append Data
  to Buffer (16);
      :Program Data
  to SRAM (17);
```

```
      :Clear Buffer (18);
      if (Program Data Failed ? (19)) then (yes)
         :Send ACK
  Response (20);
      else (no)
         :Send Error
  Response (21);
      endif
   else (no)
      :Send Error
 Responce (21);
   endif
   endfork
endif
@enduml
```

FLASH BOOT FOR TRAVEO II IP BROS

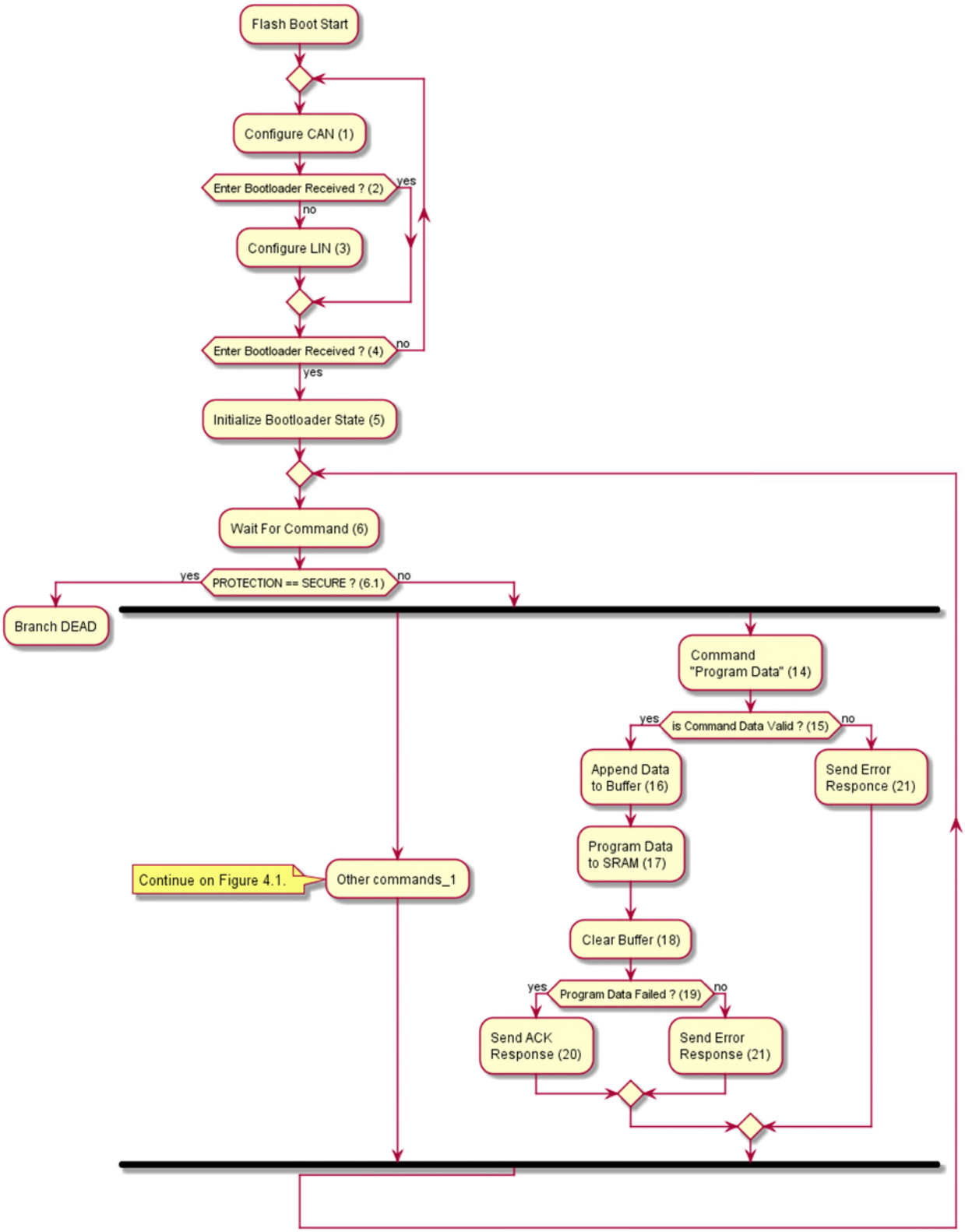


Figure 4. Internal Bootloader Flow

```
@startuml
start
note right
this is the "Other Commands_1" block begin point
end note
  fork
  :Command
"Sync" (36);
  :Clear
Buffer (18);
fork again
  :Command "Verify
App" (35);
  if (is Valid App ? (12)) then (yes)
    :Send ACK
 Response (20);
  else (no)
    :Send Error
 Responce (21);
  endif
fork again
  :Command
"Send Data" (22);
  :Append Data
to Buffer (16);
  :Send ACK
Response (20);
fork again
:Other commands_2;
note right
Continue on Figure 4.1.
end note
endfork
stop
note right
this is the "Other Commands_1" block end point
end note
@enduml
```

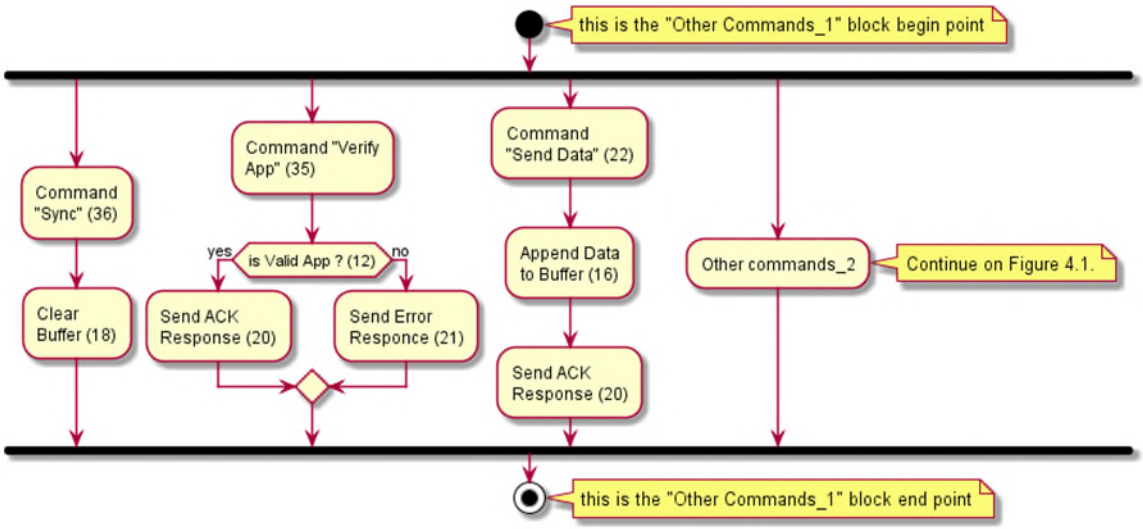


Figure 4.1. Additional commands - part 1

```
@startuml
start
note right
this is the "Other Commands_2" block begin point
end note
  fork
  :Command
"Enter
Bootloader" (7);
  :State = BOOTLOADING (8);
  :Send "Enter Response" (9);
  fork again
  :Command "Exit
Bootloader" (10);
  :State = FINISHED (11);
  if (is Valid App? (12)) then (yes)
     :Switch to
 RAM App (13);
     detach;
  else (no)
  endif
  fork again
  :Command "Send
App Metadata" (32);
  :Send ACK
Response (20);
```

endfork
stop
note right
this is the "Other Commands_2" block end point
end note
@enduml

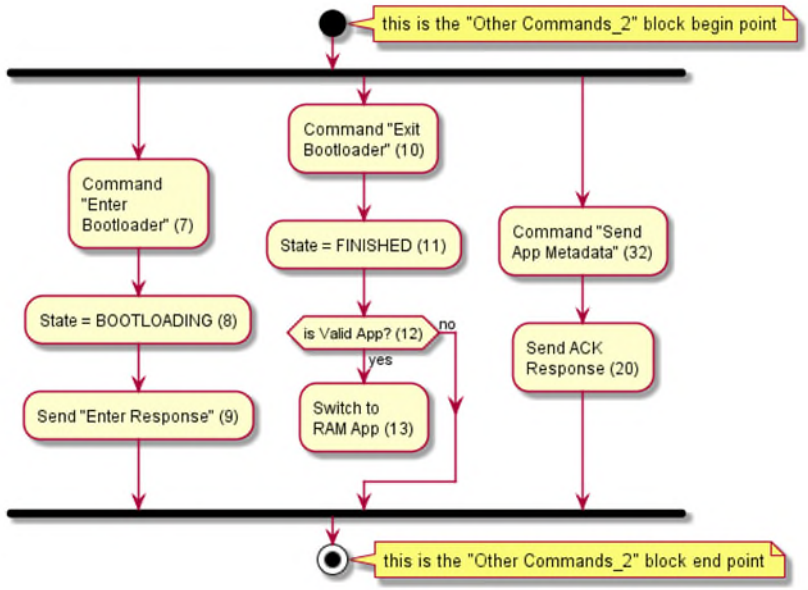


Figure 4.2. Additional commands - part 2

Configure CAN (1)

Bootloader internal _step 1.1: Configure CAN
CAN supports 500 KBPS or 100 KBPS communication speeds.

Bootloader internal _step 1.2: Configure CAN
The polling for an *Enter* bootloader command on CAN starts at 100 KBPS. After timeout if no valid data has been received then poll starts at 500 KBPS speed.

Bootloader internal _step 1.3: Configure CAN
RX, TX, and two ENABLE pins must be present in all the packages.

Bootloader internal _step 1.4: Configure CAN
The ENABLE pins must be strong drive to support CAN transceivers which have pull-up or pull-down resistors in the ENABLE pins internal circuit. Moreover, all the LIN transceivers do have the integrated pull-down resistors in their ENABLE pins circuits.
One ENABLE pin must be active HIGH, the other – active LOW.
Both ENABLE pins must be in a generic GPIO output mode, and shall not use CAN or LIN special function configuration, as they will be used for either CAN or LIN.

Enter bootloader received? (2)

Bootloader internal _step 2: Enter bootloader on CAN received?
If the *Enter* bootloader command is received on CAN communication interface, then continue bootloading using this communication channel. No need to poll the communication on the other interfaces.

Configure LIN (3)

Bootloader internal _step 3.1: Configure LIN
LIN 20 KBPS communication speed must be supported.

Bootloader internal _step 3.2: Configure LIN
LIN 115200 BPS communication speed must be supported.

There are RX, TX, and two ENABLE pins in all device packages.

Bootloader internal _step 3.3: Configure LIN
The ENABLE pins are shared with CAN communication interface.

Enter bootloader received? (4)

Bootloader internal _step 4.1: Enter bootloader on LIN received?
If the *Enter* bootloader command is received on the LIN communication interface, then continue bootloading using this interface. No need to poll the communication on the other interfaces .

Initialize bootloader state (5)

Bootloader internal _step 4.2
Set state variable to the NONE value. In this state, only the *Enter Bootloader* command is accepted, all the other bootloader commands are ignored.

Bootloader internal _step 5: Initialize bootloader state
Initialize and clear programming buffer, *Program Data* and *Send Data* bootloader commands use this buffer.

Wait for Command (6)

Bootloader internal _step 6: Wait for Command

The bootloader waits for a full bootloader command to be received through CAN or LIN.
After this happens, it checks for the command to be valid, if yes, it jumps to the appropriate command.

PROTECTION == SECURE? (6.1)

Bootloader internal _step 6.1: PROTECTION==SECURE?

Bootloader shall compare CPUSS_PROTECTION value against SECURE before handling each bootloader command. If CPUSS_PROTECTION = SECURE when a new bootloader command is received this means the bootloading is performed by a malicious means and must be stopped immediately. Bootloader puts MCU into DEAD state in this case.

Command "Enter Bootloader" (7)

Bootloader internal _step 7: Enter Bootloader

"Enter bootloader" is the first bootloader command to be received by bootloader.
The device checks the command data and responds if it is ready to perform a bootloading.

State = BOOTLOADING (8)

Bootloader internal _step 8: State = BOOTLOADING

The state variable value is assigned to BOOTLOADING.
After this, any other bootloader commands may be handled by the bootloader.

Send Enter Response (9)

Bootloader internal _step 9: Send Enter Response

The Enter bootloader command has a special response to bootloader host. See002-13924 and Bootloader SDK SEROS(002-12095) for details.

Command Exit Bootloader (10)

Bootloader internal _step 10: Command Exit Bootloader

This command is used to inform the bootloader that bootloading has been finished.

State = FINISHED (11)

Bootloader internal _step 11: State = FINISHED
This state variable value indicates that bootloading has finished.
No more bootloading command are processed.

Is Valid App? (12)

Bootloader internal _step 12: Is Valid App?
Application integrity validation and secure signature verification are done by calling Flash boot function *Cy_FB_IsValidApplication()*.
Application has to be in the Cypress Secure Application Format.

Switch to SRAM App (13)

Bootloader internal _step 13: Switch to SRAM App
The Flash Loader is stored and executed from the SRAM.
The procedure to launch a user application is described in Figure 1.4. in section **8- Theory of Operation**.

Command Program Data (14)

Bootloader internal _step 14: Command Program Data
The *Program data* bootloader command is used to program received data into the destination location. In the Flash Loader, it is a dedicated area in the SRAM.

Before storing data, it is verified to be valid – steps (15) and (18).

Note that *Program data* appends data to the bootloading buffer that may be filled with *Send Data*. The buffer is cleared after the data has been written to its destination location.
See 002-13924 for details.

Is Command Data Valid? (15)

Bootloader internal _step 15: Is Program Data Valid?
Data Valid is verified by checking the CRC of the program buffer with the value that is part of the *Program Data* command and by checking that the application start address is inside the allowed SRAM range – from the start of SRAM till the end of SRAM minus 4KB.
The *Program data* bootloader command has a data CRC inside it. This allows data integrity verification. The CRC algorithm is CRC-32C. See 002-12095 for more details.

The application start address and application length are provided by the bootloader host with the *Send App Metadata* bootloader command.

Append data to buffer (16)

Bootloader internal _step 16: Append data to buffer

The data for the *Send Data* command to be appended to the *Program Data* buffer if *Send Data* command is used.

Program Data to SRAM (17)

Bootloader internal _step 17: Program Data to SRAM

Bootloader SDK function *Cy_Bootload_WriteData()* is used for this step.
This function has to be implemented in the Bootloader Application to allow downloading application into the SRAM.

Clear buffer (18)

Bootloader internal _step 18: Clear buffer

Program Data writes the data to its destination location, then it clears the programming buffer, so new data can be written there.

Program Data Failed? (19)

Bootloader internal _step 19: Program Data Failed?

Cy_Bootload_WriteData() may return error code, if this happens then (18) results in “NO”.

Send ACK Response (20)

Bootloader internal _step 20: Send ACK Response

The bootloader sends ACK response to the bootloading host through the CAN or LIN interface.

Send Error Response (21)

Bootloader internal _step 21: Send Error Response

The bootloader sends an error response with error code indicating the failure reason to the bootloader host.

Command Send Data (22)

Bootloader internal _step 22: Command Send Data

This command appends data to the buffer that is used by *Program Data* command.
It may be useful to download complete Flash rows, but if downloading into the SRAM it may be optional.

Command Send App Metadata (32)

Bootloader internal _step 32: Command Send App Metadata

This command provide start address in SRAM and application size for Flash Loader.

Command Verify Application (35)

Bootloader internal _step 35: Command Verify Application

The bootloader host may send the *Verify Application* command to ensure the application is sent is valid.
The device responds with ACK or Error code to this command.

Command Sync (36)

Bootloader internal _step 36: Command Sync

This command is used to reset bootloader state.

FLASH BOOT: CyMCUElfTool

CyMCUElfTool is used to generate CYACD2 files for flash loader. See 002-20889 and 002-22934 for the requirements and documentation.

8.5.4.1.1 Bootloading timeouts and alternating CAN and LIN

FLASH BOOT: Bootloading Timeouts _1

Bootloader polls for *Enter bootloader* command on CAN and LIN pins as following:
1. Bootloader polls for CAN messages at 100 KBPS, if no valid CAN message with *Enter* bootloader command is received during 10 ms it goes to (2). If a valid Enter bootloader command is received, bootloader continues using CAN at 100 KBPS for the next bootloader commands.

FLASH BOOT: Bootloading Timeouts _2

2. Bootloader polls CAN at 500 KBPS for 10 ms duration. If no valid *Enter*bootloader command is

received it goes to (3).

FLASH BOOT: Bootloading Timeouts _3

3. Bootloader polls LIN at 20 KBPS for 150 ms duration. If no valid Enter bootloader command is received it goes to (1).

a. If a valid Enter bootloader has been received, and the next bootloader command is Set Application Metadata, and Set Application Metadata bootloader command has App Id = 0, then bootloader sends an OK response to the bootloading host and continues the bootloading process on LIN at 20 KBPS.

b. If a valid Enter bootloader has been received, and the next bootloader command is Set Application Metadata, and Set Application Metadata bootloader command has App Id = 1, then bootloader sends an OK response to the bootloading host and after this:

i. Performs a toggling on EN(HIGH) and TX pins to enter Fast Mode of LIN. See section **8.5.4.1.3.2** for more info.

ii. Re-configures LIN to 115200 BPS and waits for the next bootloader commands to use this new baud rate.

c. If a valid Enter bootloader has been received, and the next bootloader command is Set Application Metadata, and Set Application Metadata bootloader command has App Id = 2, then bootloader sends an OK response to the bootloading host and after this re-configures LIN to 115200 BPS and waits for the next bootloader commands to use this new baud rate.

FLASH BOOT: Bootloading Timeouts _4

If there were no valid *Enter* bootloader command during 300 seconds, bootloader is stopped. The device goes into *Sleep* power mode.

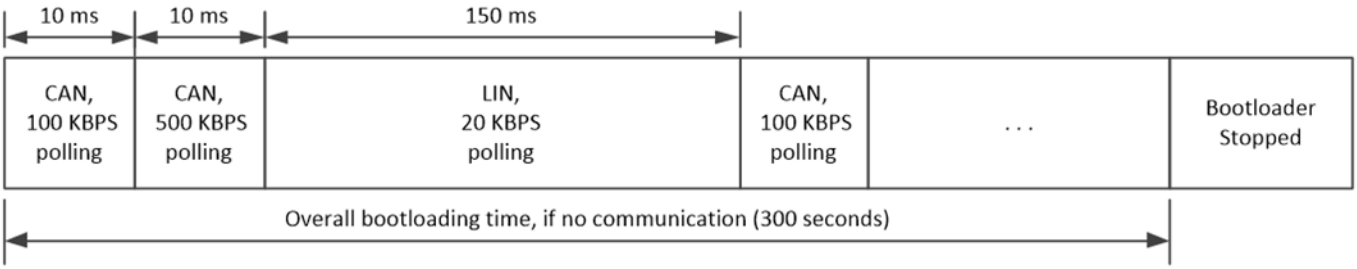


Figure 5. Polling CAN and LIN with no Bootloader commands

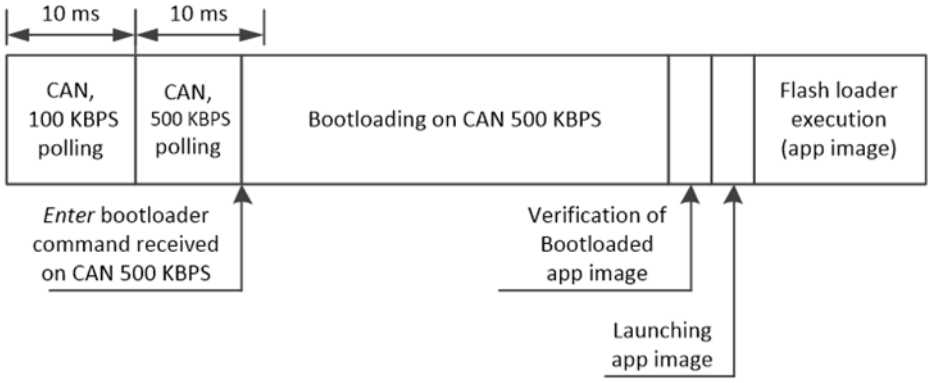


Figure 6. Successful Bootloading on CAN 500 KBPS

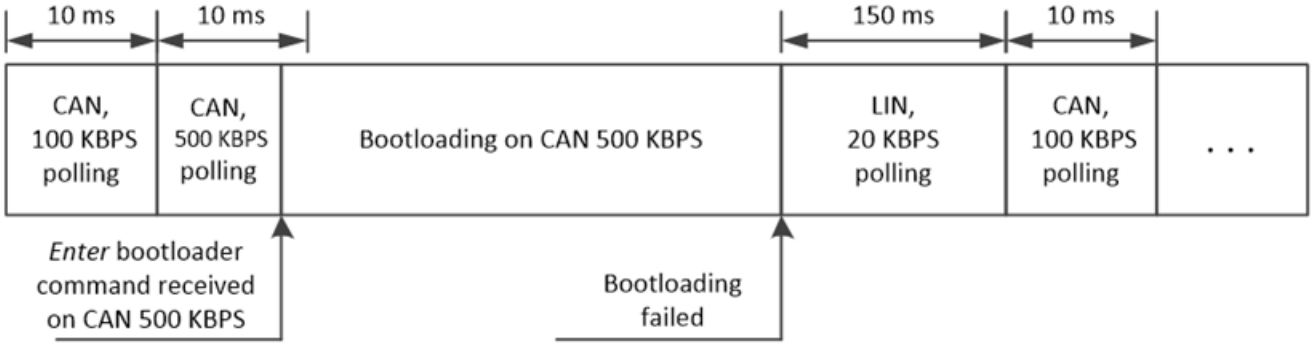


Figure 7. An example of failed Bootloading

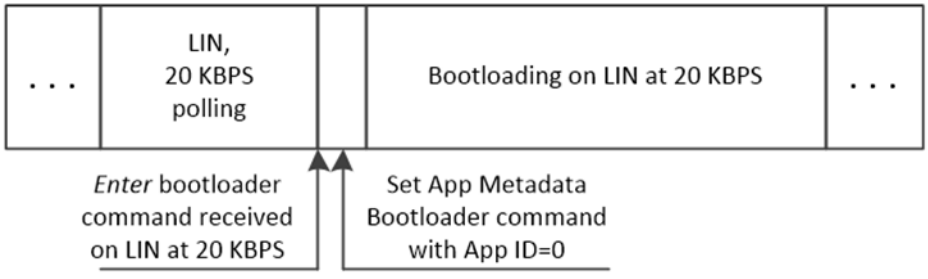


Figure 8. Bootloading on LIN at 20 KBPS for AppID=0

FLASH BOOT FOR TRAVEO II IP BROS

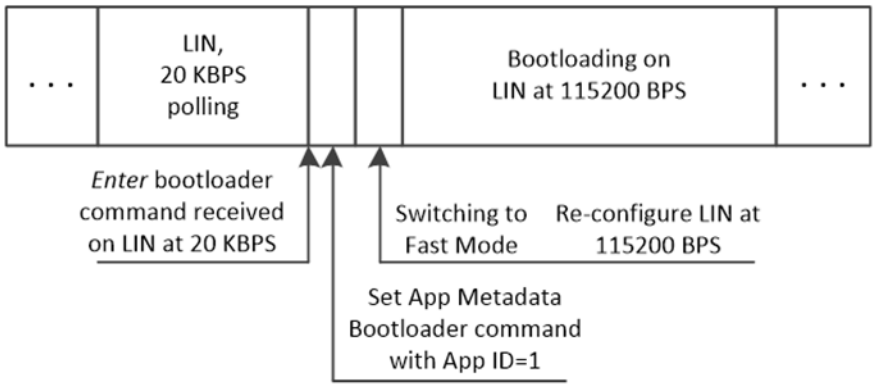


Figure 9. Bootloading on LIN at 115200 BPS for AppID=1

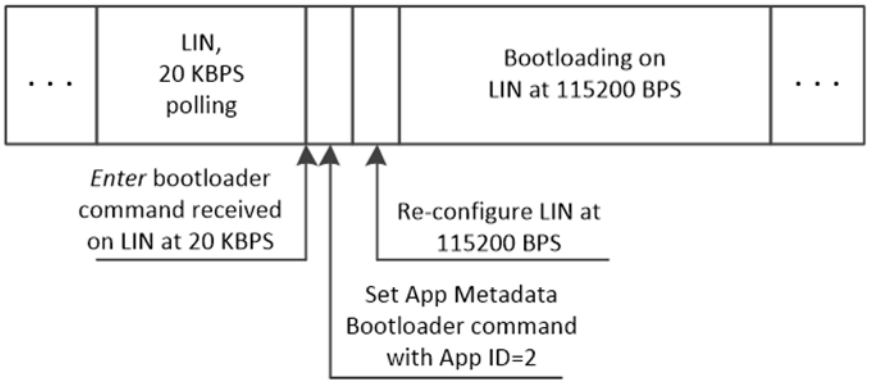


Figure 10. Bootloading on LIN at 115200 BPS for AppID=2

FLASH BOOT: Switching between CAN and LIN bootloading interfaces _1

Figure 11 shows the switching logic between CAN and LIN bootloading interfaces.

```
@startuml
:Start Bootloading (1);
repeat
repeat
:Configure CAN (2);
repeat
   :Read Data (3);
repeat while (is Enter Bootloader Command (4) ? or
has CAN Poll Timeout Passed ? (5)) is (no)
->yes;
if (is Enter Bootloader Command ? (4)) then (no)
   :Configure LIN (6);
```

```
   repeat
      :Read Data (7);
      repeat while (has LIN Poll Timeout Passed ? (4)) is (no) not (yes)
else (yes)
endif
repeat while (is Enter Bootloader Command ? (4)) is (no) not (yes)
while (Received Bootloader Command ? (10)) is (yes)
   :Handle the Command (13);
endwhile
->no;
if (is Overal Bootloading Timeout ? (11)) then (yes)
   :Stop Bootloading (14);
   detach
else (no)
   if (Is Communication Timeout ? (12)) then (yes)
      :Stop Bootloading;
      detach
   else (no)
endif
@enduml
```

FLASH BOOT FOR TRAVEO II IP BROS

FLASH BOOT FOR TRAVEO II IP BROS

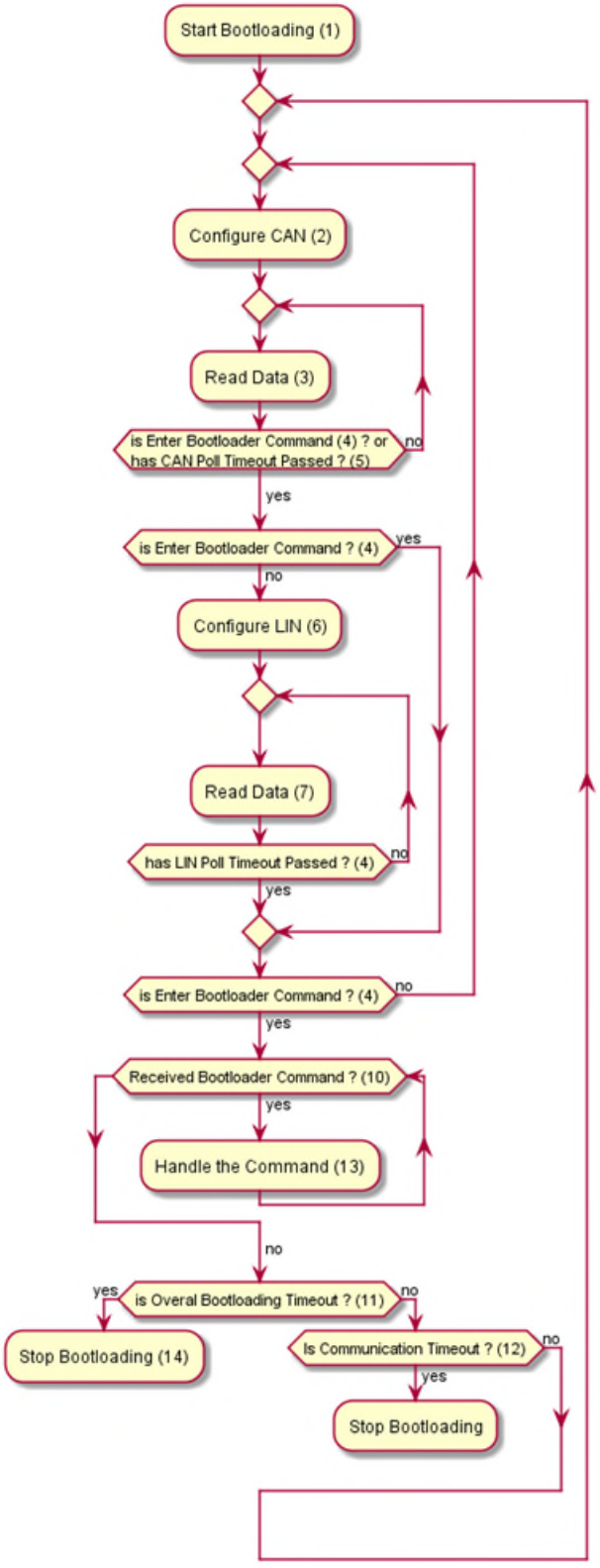


Figure 11. Bootloading switching between CAN and LIN

FLASH BOOT: Switching between CAN and LIN bootloading interfaces _2
(5) A CAN polling timeout is 10 ms at 100 KBPS and 10 ms at 500 KBPS.

FLASH BOOT: Switching between CAN and LIN bootloading interfaces _3
(9) A LIN polling timeout is 150 ms at 20 KBPS.

FLASH BOOT: Switching between CAN and LIN bootloading interfaces _4
(11) An overall bootloading timeout is 300 seconds from the end of the last successful received bootloading command, or from the start of the bootloading if no commands have been received.

FLASH BOOT: Switching between CAN and LIN bootloading interfaces _5
(12) A *Communication timeout* flag is set if no valid bootloader command has been received for 2 seconds.

FLASH BOOT: Switching between CAN and LIN bootloading interfaces _6
(13) LIN by default is configured at 20 KBPS. But if the *Send App Metadata* bootloader command is received with AppID=1, then LIN is reconfigured to 115200 BPS. See section **8.5.4.1.3.2**

8.5.4.1.2 CAN configuration

FLASH BOOT: CAN Configuration

The general CAN configuration for ES10:

- CAN instance CAN0[1]
- Baud rate 100 or 500 KBPS, alternating
- CAN FD mode false (Use CAN classic mode)
- Rx/Tx Buffer Data Size 8 byte
- Rx FIFO Size 4 elements
- Rx FIFO Mode blocking
- Rx message ID 0x1A1u
- Tx message ID 0x1B1u
- CAN clocks for 100 KBPS (for silicon):
    - Prescaler 2.
    - Time Segment 1 39 tq.
    - Time Segment 2 10 tq.

    - SWJ 10 tq.
    - Sampling point 80%.
- Prescalers for 500 KBPS (for silicon):
    - Prescaler 2.
    - Time Segment 1 39 tq.
    - Time Segment 2 10 tq.
    - SWJ 10 tq.
    - Sampling point 80%

Table 1 CAN Pins

Pin	TVII-BE-1M	TVII-BE-2M	TVII-BE-4M	TVII-C2D-4M	TVII-C2D-6M	TVII-C2D-6M DDR	TVII-BH-4M	TVII-BH-8M
RX	P0[3]	P0[3]	P0[3]	P5[5]	P2[4]	P2[4]	P0[3]	P0[3]
TX	P0[2]	P0[2]	P0[2]	P5[4]	P2[3]	P2[3]	P0[2]	P0[2]
EN HIGH	P2[1]	P2[1]	P2[1]	P2[1]	P0[2]	P0[2]	P2[1]	P2[1]
EN LOW	P23[3]	P23[3]	P23[3]	P2[0]	P0[5]	P0[5]	P23[3]	P23[3]

TX, EN HIGH and EN LOW pins have Strond output drive. RX pin is in the High Z mode.

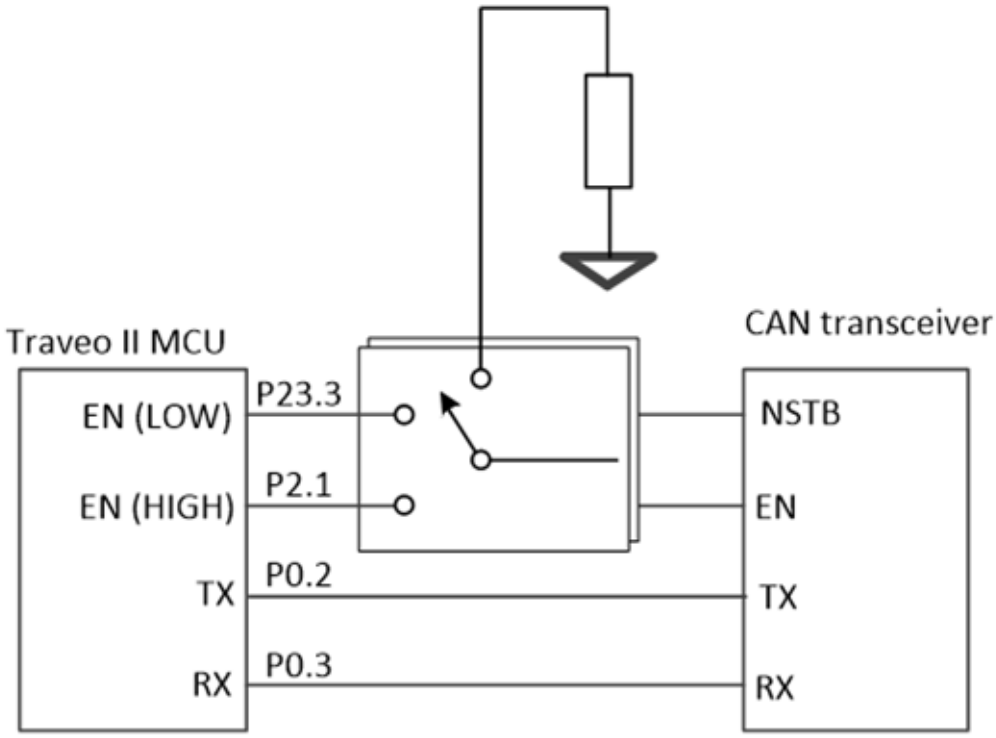


Figure 12. The connection of a CAN transceiver to MCU

Note CAN transceivers use multiple schemes for enable pins, some use active low, while the other active high, with a pull-up/pull-down resistor or without.

8.5.4.1.2.1 CAN timings configuration

The default CM0+ clock for a PSVP is 24 MHz, the default CM0+ clock for a silicon is 50 MHz. Due to this, the CAN timing configuration is defined in a macro within `#if/else CY_FB_OPT_PSVP != 0` blocks.

CAN requires configuring *nominal bit timing* and the *CAN peripheral prescaler*. This configuration if performed by writing to the *TTCANFD.NBTP* register.

See the description of the *TTCANFT.NBTP* register in the SAS file *MXS40-IP-CAN.xlsm* 001-98134 on page *M_TTCAN_Regs*.

One nominal CAN bit consists of N time quants. A prerescaler is used to divide the input clock, which is then divived by N to get a baud rate.
I.e. at 24 MHz peripheral clock, with a CAN prescaler value of 6, and N=8, the baud rate is *24e6 / 6 / 8 = 500000*.

One nominal bit consists of Synchronization Jump Width (SJW) time (shown an SYNC_SEG on Figure), Propagation Time Segment (shown as PROP_SEG), Phase 1 Time Segment (PHASE_SEG1), and Phase 2 Time Segment (PHASE_SEG2).

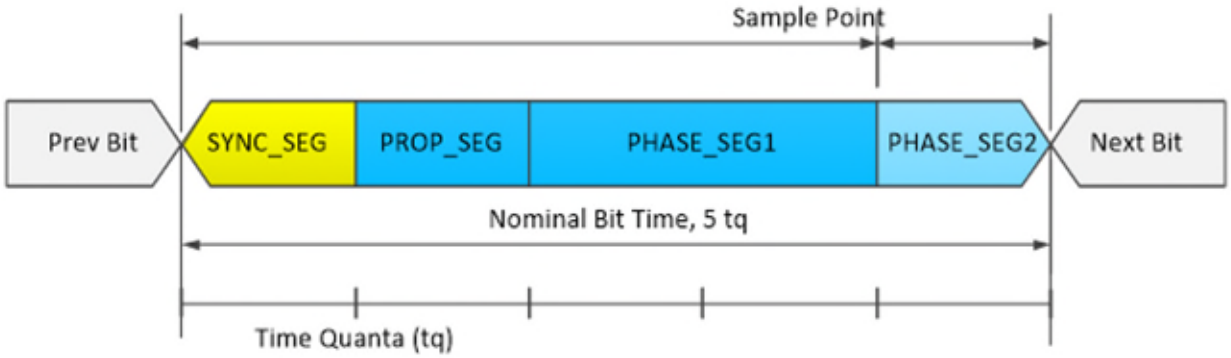


Figure 13. CAN timings configuration

A sample point value is (SYNC_SEG + PROP_SEG + PHASE_SEG1) / NOMINAL_BIT_TIME * 100 %.

To calculate the TTCANFT.NBTP register bits values:

1. Select the PERI frequency (PERI_CLK), and CAN baud rate (BAUD).
2. Select the CAN prescaler, number of time quants (N), SYNC_SEG, PROP_SEG + PHASE_SEG1, and PHASE_SEG2 values to fulfil the following conditions:
   a. PERI_CLK / prescaler / N = BAUD.
   b. N (in time quants) = (SYNC_SEG + PROP_SEG + PHASE_SEG1 + PHASE_SEG2).
   c. The sample point in the range 50 - 90 %.

8.5.4.1.2.2 CAN Timings

FLASH BOOT: CAN Timings

Overall bootloading time measurements for the 32 KB application showed the following results:

CAN baud rate, KBPS	Overall bootloading time for the 32 KB application, seconds
100	8.5
500	5.5

8.5.4.1.2.3 CAN limitations

FLASH BOOT FOR TRAVEO II IP BROS

Flash boot uses IMO with PLL for CM0+ and PERI clocks, CAN IP is affected by these clocks. IMO frequency accuracy is ±1.0%.

FLASH BOOT: CAN limitations _1

CAN at 100 KBPS allows ±1.6% clock frequency deviations. This requirement is fully met in the bootloader.

FLASH BOOT: CAN limitations _2

CAN at 500 KBPS allows up to ±0.5% clock frequency deviations for general use case. This requirement is not met by the bootloader because of a hardware limitations of IMO. Instead, ±1.0% clock frequency deviations are allowed for the following conditions:

- Single end-to-end connection.
- Avoid wire length longer than the spec for 500 KBPS.
- CAN time quanta, prescaler, and sampling point parameters taken from CDT 301902 post #16.

Classical CAN / CAN FD setting	Baud Rate [kbps]	tBit [µsec]	TQ/Bit	tTQ [nsec]	SP > 80% [TQ]	Effective SP [%]	Clock tole-rance [%]	Devia-tion after 10 bit times [nsec]	Total Deviation < RSJWmax?	Timing Margin for other bus compo-nents [nsec]	TJA1043 TXD->bus dom + bus dom->RXD delay (typ) *2 [nsec]	Margin for wire harness [nsec]	Wire harness 1 nF load @ 100 Ohms	Remaining margin > 0?
Cl. CAN	500	2	25	80	20	80%	1,0	200	will cause pertetual re-synch	1720	260	1460	300	1160
Cl. CAN	500	2	24	83,33	20	83%	1,0	200	will cause pertetual re-synch	1717	260	1457	300	1157
Cl. CAN	500	2	20	100	16	80%	1,0	200	will cause pertetual re-synch	1700	260	1440	300	1140
Cl. CAN	500	2	16	125	13	81%	1,0	200	will cause pertetual re-synch	1675	260	1415	300	1115
Cl. CAN	500	2	10	200	8	80%	1,0	200	will cause pertetual re-synch	1600	260	1340	300	1040
Cl. CAN	500	2	8	250	7	88%	1,0	200	ok	1550	260	1290	300	990
CAN FD	500	2	50	40	40	80%	1,0	200	will cause pertetual re-synch	1760	260	1500	300	1200
CAN FD	500	2	40	50	32	80%	1,0	200	will cause pertetual re-synch	1750	260	1490	300	1190

Note SWJ may be used by CAN IP block to adjust clock frequency to the current baud rate, when clock tolerance is within ±1.0% accuracy.

8.5.4.1.3 LIN configuration

FLASH BOOT FOR TRAVEO II IP BROS

FLASH BOOT: LIN Configuration

- The general LIN configuration for ES10:
  - LIN instance LIN1
  - LIN Slave mode
  - Bitrate 20 kbps or 115.2kbps, user configurable.
  - Break Field Length 11
  - Lin Transceiver Auto Enable true
  - Break Delimiter Length 1 bit
  - Stop bit 1
  - PID 45 for Rx and 46 for Tx
  - Checksum Type Classic

Table 2 LIN Pins

Pin	TVII-BE-1M	TVII-BE-2M	TVII-BE-4M	TVII-C2D-4M	TVII-C2D-6M	TVII-C2D-6M DDR	TVII-BH-4M	TVII-BH-8M
RX	P0[0]	P0[0]	P0[0]	P1[7]	P0[1]	P0[1]	P0[0]	P0[0]
TX	P0[1]	P0[1]	P0[1]	P1[6]	P0[0]	P0[0]	P0[1]	P0[1]
EN HIGH	P2[1]	P2[1]	P2[1]	P2[1]	P0[2]	P0[2]	P2[1]	P2[1]
EN LOW	P23[3]	P23[3]	P23[3]	P2[0]	P0[5]	P0[5]	P23[3]	P23[3]

TX, EN HIGH and EN LOW pins have Strond output drive. RX pin is in the High Z mode for all devices except TVII-C2D-4M. For this device is has a resistive pull-up drive mode.

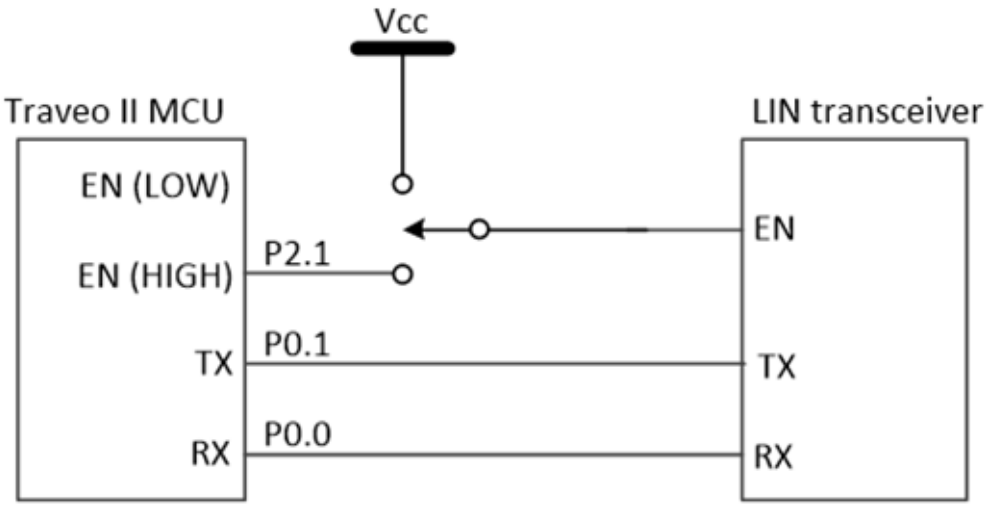


Figure 14. The connection of a LIN transceiver to MCU

8.5.4.1.3.1 LIN prescaler calculation

The LIN IP block requires an additional prescaler to configure the baud rate.

The LIN IP does oversampling, the oversampling count is fixed 16.

`LIN_baud_rate = LIN_clock/16 = PERI_CLK/LIN_divider/16.`

E.g. for PSVP, PERI_CLK = 24 MHz. Thus for the 20 KBPS baud rate

`LIN_divider = PERI_CLK/16/LIN_baud_rate = 24e6/16/20e3 = 75.`

Note, a value divider value assigned to *PERI.DIV8_CTL[]* shall be *divider-1*.

For LIN at 115200 BPS the default PERI clock of 50 MHz is does not allow to

The default PERI clock of 50 MHz and an integer peripheral divider does not allow setting 115200 BPS LIN setting within 0.6%. To overcome these two methods may be used:

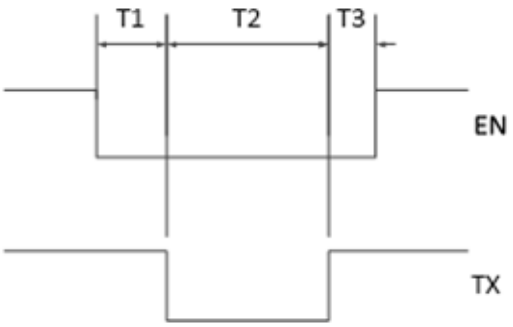
3. Use a fractional divider for LIN peripheral clock.
   This is simpler solution, but fails with LIN masters having clock deviation of 20% and higher. This is not used in the bootloader on TVII families, because the LIN master used for testing has high deviation of LIN clock.
4. Use an integer divider, but adjust PERI clock.
   This is more complicated solution. It increases code size, but gives a bit better tolerance to LIN master having clock deviation of 20% and higher.

8.5.4.1.3.2 Switching between Normal and Fast LIN modes

Some manufacturers of LIN transceivers allow so called “Fast mode” or “Flash mode” which is used mainly for bootloading. Fast mode allows LIN communication speed to be increased to 115200 BPS.

FLASH BOOT: Switching between NORMAL and FAST LIN modes

A special sequence of signals on EN, and TX pins of the LIN transceiver switches it to the Fast mode.



The T1, T2, and T3 limits may vary from manufacture to manufacturer. Flash boot uses average values: T1 = T2 = T3 = 12 us.

Switching from Fast mode to Normal mode is done by applying the same sequence on EN, and TX pins.

8.5.4.1.3.3 Normal vs Fast mode communication time

FLASH BOOT: LIN Timings

Overall bootloading time measurements in Normal and Fast modes for the 32 KB application showed the following results (provided in CDT 326938):

LIN baud rate, KBPS	Overall bootloading time for the 32 KB application, seconds
20.0	56
115.2	28

8.5.4.1.4 Bootloadable Application Requirements

The bootloader in Flash boot updates an application in the SRAM. Then it validates this application and transfers control to it. This application may be a bootloader by itself. However, what the application does is out of scope for Flash boot.

FLASH BOOT: SRAM Initialization
Traveo II SRAM has an error checking and correction (ECC) mechanism, thus Flash boot has to initialize the SRAM with zero before the start of a bootloading process.

The application start address shall be aligned to 256 bytes, the application size shall be:

- Aligned to 4 bytes
- Aligned to CYACD2 file row size.

Bootloadable application shall be in basic application format.

8.5.4.1.5 Security enhancements for bootloading

FLASH BOOT: Security Enhancements _1
To increase a system security it is recommended (CDT 322947) to verify that CPUSS_PROTECTION < SECURE for each bootloading message (read and write). Flash boot runs this check at the start of Cy_Bootload_TransportRead() and Cy_Bootload_TransportWrite().

FLASH BOOT: Security Enhancements _2
If the check fails and CPUSS_PROTECTION equals to SECURE or DEAD then Flash boot goes into the DEAD branch and sets error code CY_FB_ERROR_BOOT_SECURE.

The same check is performed before launching the bootloadable application.

8.5.4.2 TOC (Table of Contents)

The TOC organization and structure is defined in MXS40 SAS (001-98134, Book2-PartII-BOOT), and in the SAS excel file MXS40-IP-SFLASH-(Family).xlsm (MXS40/Platform/MXS40-SAS/Configuration/). These are the family-specific Excel files. The TOC structure is described in Table 1.

Table 1. TOC organization and structure

Offset	Size	Qty	Width	Name	Description	Default
0x7C00		1	32	TOC2_OBJECT_SIZE	Object size in bytes for CRC calculation starting from offset 0x00	0x1FC (508u)

0x7C04		1	32	TOC2_MAGIC_NUMBER	Magic number(0x01211220)	0x01211220u
0x7C08		1	32	TOC2_SMIF_CFG_STRUCT_ADDR	Null terminated table of pointers representing the SMIF configuration structure	0x0u
0x7C0C		1	32	TOC2_FIRST_USER_APP_ADDR	Address of First User Application Object	0x10000000u
0x7C10		1	32	TOC2_FIRST_USER_APP_FORMAT	Format of First User Application Object. 0 - Basic, 1 - Cypress standard & 2 - Simplified	0x0u
0x7C14		1	32	TOC2_SECOND_USER_APP_ADDR	Address of Second User Application Object	0x0u
0x7C18		1	32	TOC2_SECOND_USER_APP_FORMAT	Format of Second User Application Object. 0 - Basic, 1 - Cypress standard & 2 - Simplified	0x0u
0x7C1C		1	32	TOC2_FIRST_CMx_1_USER_APP_ADDR	Address of First CM4 or CM7 core1 User Application Object	0x0u
0x7C20		1	32	TOC2_SECOND_CMx_1_USER_APP_ADDR	Address of Second CM4 or CM7 core1 User Application Object	0x0u
0x7C24		1	32	TOC2_FIRST_CMx_2_USER_APP_ADDR	Address of First CM4 or CM7 core2 User Application Object	0x0u
0x7C28		1	32	TOC2_SECOND_CMx_2_USER_APP_ADDR	Address of Second CM4 or CM7 core2 User Application Object	0x0u

0x7CFC		1	32	CYREG_SFLASH_TOC2_SECURITY_MARKER	Enables security fixes for BH and BE devices if Magic number (~TOC2_MAGIC_NUMBER). CDT363653, CDT367642.	0x0Uu
0x7D00		1	32	TOC2_SHASH_OBJECTS	Number of additional objects to be verified for SECURE_HASH	0x3u
0x7D04		1	32	TOC2_SIGNATURE_VERIF_KEY	Address of signature verification key (0 if none).The object is signature specific key. It is the public key in case of RSA	0x0u
0x7D08		1	32	TOC2_APP_PROTECTION_ADDR	Address of Application Protection	0x17007600U
0x7DF4		1	32	TOC2_REVISION	Indicates TOC2 Revision. It is not used now.	
0x7DF8		1	32	TOC2_FLAGS	Controls default configuration	0x00000242u

Flash boot does not use TOC1, in contrast TOC2 is actively used throughout the Flash boot code base.

Flash boot uses SDL files *tviibe1m.h, cy_ip_sflash.h* to calculate TOC2 location in the following way:

```
CY_FB_TOC2_ADDRESS = ((uint32_t)&SFLASH->unTOC2_OBJECT_SIZE)
```

Flash boot uses macro *CY_FB_TOC2_IDX_<TOC2_MEMBER>* to access each 32-bit TOC2 member. I.e. *CY_FB_TOC2_IDX_USER_APP0* is an index of TOC2 field named TOC2_FIRST_USER_APP_ADDR. The values of *CY_FB_TOC2_IDX_<TOC2_MEMBER>* are calculated from SDL register map, so there is no need to manually recalculate them when TOC2 layout is changed. Although, an SDL shall be update for the SAS changes to be applied.

A user application is responsible for allocating space for and setting up the TOC2.

TOC2 protection is important item for the whole system security. TOC2 should be protected to avoid loading of malicious application by substitution of an RSA public key and malicious application with the hash encrypted by not authorized key. There are several ways to protect TOC2:

1) Use a memory protection unit (MPU/PPU/SwPU).

2) Change the life-cycle stage to SECURE WITH DEBUG, or SECURE.
Note Before transition to SECURE, ensure that TOC1 and TOC2 are included in SECURE_HASH calculation.

Table 2. TOC flags

Bits	Name	HW	SW	Default	Sync	Description
1:0	CLOCK_CONFIG		RW	2		Indicates clock frequency configuration. The clock should stay the same after Flash Boot execution. 0 = 8 MHz, IMO, no FLL 1 = 25 MHz, IMO + FLL 2 = 50 MHz, IMO + FLL (default) 3 = Use ROM boot clocks configuration (100 MHz)
4:2	LISTEN_WINDOW		RW	0		Determines the Listen window to allow sufficient time to acquire debug port. 0 = 20 ms (Default) 1 = 10 ms 2 = 1 ms 3 = 0 ms (No Listen window) 4 = 100 ms
6:5	SWJ_PINS_CTL		RW	2		Determines if SWJ pins are configured in SWJ mode by Flash boot. Note: SWJ pins may be enabled later in the user code. 0 = Do not enable SWJ pins in Flash boot. Listen window is skipped. 1 = Do not enable SWJ pins in Flash boot. Listen window is skipped. 2 = Enable SWJ pins in Flash boot (default). 3 = Do not enable SWJ pins in Flash boot. Listen window is skipped.

8:7	APP_AUTH_CTL		RW	0		Determines if the application image digital signature verification (authentication) is performed: 0 = Authentication is enabled (default). 1 = Authentication is disabled. 2 = Authentication is enabled (recommended). 3 = Authentication is enabled.
10:9	FB_BOOTLOADER_CTL		RW	1		Determine if the internal bootloader in Flash boot is disabled: 0 = Internal bootloader is disabled. 1 = Internal bootloader is launched if the other bootloader conditions are met (default). 2 = Internal bootloader is disabled. 3 = Internal bootloader is disabled.

8.5.4.3 Secure application digital signature verification

Secure booting typically involves authenticating the application flash images using a key-based security protocol defined by a market/application specific standard. In MXS40, this process is implemented in “emulated” one-time-programmable supervisory FLASH such that one device can support multiple different security standards.

The algorithm used for the secure application signature verification is RSASSA-PKCS1-v1.5 defined in RFC 3447.

The RSA public key location is defined by the user and the reference to it is placed in the TOC2. User may place Public Key anywhere in the internal flash, however, to protect the public key from the changes in SECURE life-cycle stage, there is a range of SFLASH reserved for public key. See MXS40-IP-SFLASH.xlsm (MXS40/Platform/MXS40-SAS/Configuration/) file, page “*TVIIXXX_SR_Details*” section “*Public Key*”

The key format structure is the following:

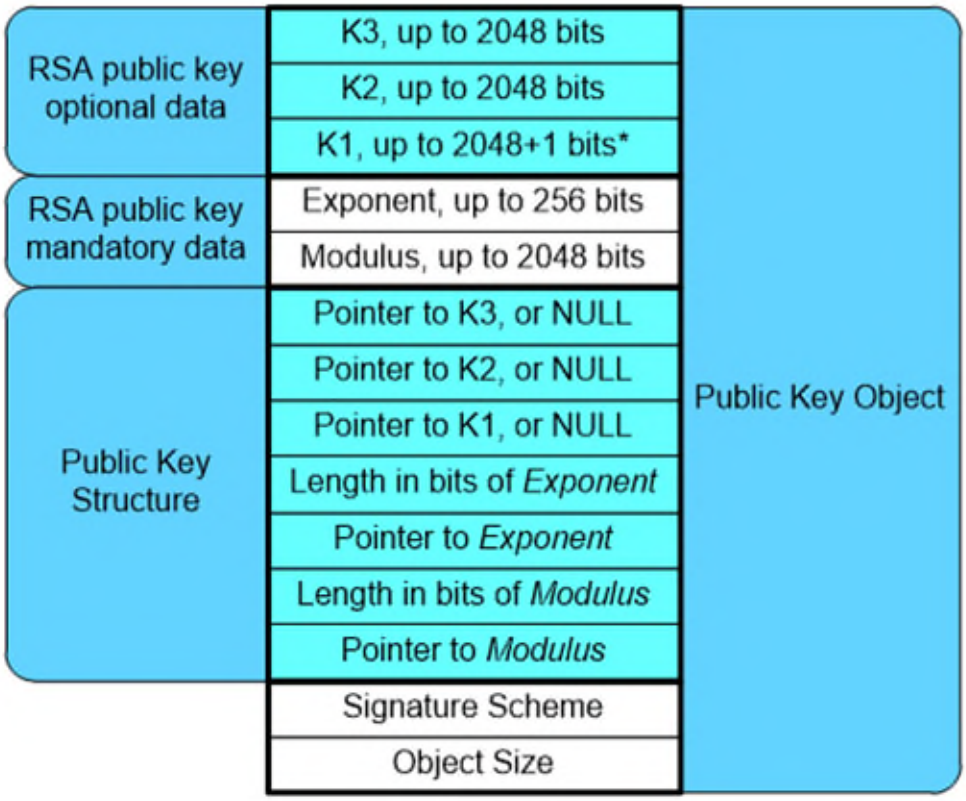


Figure 15. Public Key Format

The length of N is up to 4096 bit, the length of E is up to 256 bit.

The public key data is checked by ROM boot SECURE_HASH computation to guard against any changes when the device is in the SECURE or SECURE_WITH_DEBUG life-cycle stages.

Refer to ALXD-22 for more details on how to generate and store an RSA public key into the SFLASH for PSoC6A-BLE-2 *A secure boot validation. Note that the same steps are applicable for Traveo II with the only difference being a base address of SFLASH.

ALXD-23 describes the steps to build and sign a secure application, it also contains a simple example of a secure application which is used for Flash boot testing. DOFE-5 contains the description of RSA-3K implementation.

The Crypto IP block that may be used for validation, should only be enabled for the time the block is used and disabled promptly after application validation is complete to conserve on power.

The application validation flow is described in 8.1.2

FLASH BOOT FOR TRAVEO II IP BROS

Flash boot public function *Cy_FB_VerifyApplication()* is used to verify a secure application signature implementing the flowchart on Figure 11.
See the function parameters description in Section 8.5.1.1- Cy_FB_VerifyApplication.

Note The application digital signature is byte-reversed before the RSA encryption/decryption function call and the output of RSA encryption/decryption call is also byte reversed. Byte-reversal is done because MXS40 Crypto IP is Little Endian and RFC 3447 defines the digital signature and its RSA decrypted output to be “strings” which are byte reverse to the MXS40 Crypto API input and output.

FLASH BOOT: RSA 3/4K

The RSA 3K and 4K can be enabled in the SFLASH ROW 0 for Traveo devices.
#define CY_FB_SFLASH_FB_FLAGS_ADDRESS (0x170001FCu)
RSA2K is enabled by default.
To enable other options:
#define CY_FB_FLAGS_RSA3K_ENABLE_MASK (1u<<3u)
#define CY_FB_FLAGS_RSA4K_ENABLE_MASK (1u<<2u)
The function static uint32_t GetMaxKeyLength(void); checks these flags and returns appropriate key length. The rest of the FW flow is the same for PSoC and Traveo devices.

FLASH BOOT: Execution time

It is possible to measure the flashboot execution time. It gives an information how long the application verification takes.

The boot time feature can be enabled in the SFLASH ROW 0 for Traveo devices.
#define CY_FB_SFLASH_FB_FLAGS_ADDRESS (0x170001FCu)
This feature is disabled by default.

To enable the boot time measurement option need to write:
#define CY_FB_SFLASH_FB_FLAGS_FB_PIN_CTL_ENABLED (2u << 0u)

The function triggers the pin P1[0] high at start of the flashboot and puts it low at the end of the flashboot. By measuring the difference with application verification enabled and disabled we could calculate the execution time.

This feature also gives an information about the SROM boot time. it would be the time from XRES to the pin goes high. The total device boot time is the time from XRES to the pin's falling edge.

Note: for TVII-C2D-4M device only this pin is P6[0].

8.5.4.4 Specific Function Description

The following list of functions has a relatively complex structure and requires a more elaborate description because 002-19761*on Figure 1* containing a general Flash boot flow does not cover all the detail of the implementation for these functions.

Function Name
Cy_FB_IsValidKey()
Cy_FB_ValidateToc()
Cy_FB_IsAddressValidAndAligned()
Cy_FB_GetAppVectorTable()
Cy_FB_SwitchToApplication()
Cy_FB_GetAppParams()
ResetHandler()

8.5.4.4.1 Function Cy_FB_IsValidKey

FLASH BOOT: Cy_FB_IsValidKey

This function returns *True* if the public key is valid, otherwise it returns *False*.

The public key validation steps:

1. Check if the address of a public key in TOC2 points to a valid memory location. Macro *GET_U32_ELEM_S()*, and functions *IsAddressValid()*, *Cy_FB_IsAddressValidAndAligned()* may be used to determine if the address is pointing to a valid memory location.
2. Check if the address of the *Object Size* member of a public key object is a valid memory location.
3. Check if the *Object Size* value is within the allowed range [MIN, MAX].
   The maximum value is determined from the MXS40-IP-SFLASH.xlsx file (MXS40/Platform/MXS40-SAS/Configuration/), which contains an SFLASH region for the public key object.
4. Check if the address of the last word in the public key object points to a valid memory location.
5. Check if the *Signature Scheme* member of the public key object is valid.
   Valid values are either 0, or 1 per MXS40 SAS.
6. Check if the RSA public key exponent size is less than or equals to 32*8 bits.
   Check if the RSA public key module size is less than or equals to 256*8 bits.
7. Check if the values of RSA public key *modulo* and *exponent* members of the public key structure are inside the memory region for public key object.

8. Validate the RSA optional coefficients (*barretCoefPtr, inverseModuloPtr, rBarPtr*). Their values shall either be zero, or the addresses inside the memory range of the public key object.

8.5.4.4.2 Function Cy_FB_ValidateToc

This function tests if TOC2 and RTOC2 are empty, invalid, or valid.

FLASH BOOT: Cy_FB_ValidateToc _1
If either TOC2 or RTOC2 is valid, return the address of the first valid one.

FLASH BOOT: Cy_FB_ValidateToc _2
If both TOC2 and RTOC2 are empty, return CY_FB_TOC_EMPTY.

FLASH BOOT: Cy_FB_ValidateToc _3
If both TOC2 and RTOC2 are invalid, or one is invalid and the other is empty, return CY_FB_TOC_INVALID.

FLASH BOOT: Cy_FB_ValidateToc _4
TOC2 is *empty* if the TOC2 members named *Object size* and *Magic number* are either both equal to 0x0000_0000, or to 0xFFFF_FFFF.

FLASH BOOT: Cy_FB_ValidateToc _5

TOC2 is *invalid* if any of the following conditions is true:

9. *Magic number* value is not equal to the value defined in MXS40-IP-SFLASH.xlsx (MXS40/Platform/MXS40-SAS/Configuration/).
10. The CRC of the TOC2 data is not valid.
11. *TOC2.TOC2_FIRST_USER_APP_ADDR* is not pointing to a valid memory range, or is not 32-bit aligned.

FLASH BOOT: Cy_FB_ValidateToc _6
TOC2 is *valid* if it is not *empty*, and not *invalid*.

8.5.4.4.3 Function Cy_FB_IsAddressValidAndAligned

FLASH BOOT: Cy_FB_IsAddressValidAndAligned _2
This function returns *True* if the address is pointing within a valid memory range and is 32-bit aligned. Otherwise, it returns *False*.

FLASH BOOT: Cy_FB_IsAddressValidAndAligned _1
A memory range is valid if it is within the SRAM including the MPN wounding, Main Flash including the MPN wounding, the EmEEPROM Flash region, or SFLASH region.

8.5.4.4.4 Function Cy_FB_GetAppVectorTable

FLASH BOOT: Cy_FB_GetAppVectorTable
Returns an address of the interrupts vector table (VT) of a user application for the CM0+ core. If there is no valid CM0+ VT, returns CY_FB_INVALID_VECTOR_TABLE value (0xFFFF_0000).

The CM0+ VT address depends on the application format. See MXS40 SAS (001-98134) for the application formats description, or VENN-36.

8.5.4.4.5 Function Cy_FB_SwitchToApplication

FLASH BOOT: Cy_FB_SwitchToApplication _1
For multi-core MCU this function launches the user application for CM0+, causing Flash boot exit.

It is up to the user application to turn on the other MCU cores.

FLASH BOOT: Cy_FB_SwitchToApplication _2

The switching procedure:

12. Flash boot sets *CPUSS.CM0_VECTOR_TABLE_BASE* value to an address of a CM0+ interrupt vector table for a user application.
13. Flash boot performs a CM0+ core reset.
14. ROM boot starts up, tests if *CPUSS.PROTECTION* <> 0, which means ROM boot is launched after a “software reset”.
15. If (3) succeeds, ROM boot sets SP and PC register values from the user interrupt vector table, the address to which is stored in *CPUSS.CM0_VECTOR_TABLE_BASE*.
16. After ROM boot sets PC register value with the user reset handler address, a user code starts executing.

Note The *CPUSS.PROTECTION* value is cleared to zero on any reset event, except a CM0+ core reset.

8.5.4.4.6 Function Cy_FB_GetAppParams

FLASH BOOT: Cy_FB_GetAppParams _1
Returns *True* if an application parameters are valid for a given application number (0, or 1). Otherwise returns *False*.

FLASH BOOT: Cy_FB_GetAppParams _2

This function does the following:

17. Reads the TOC2 fields “Address of CM0+ First User Application Object” or “Address of CM0+ Second User Application Object” depending on the application number parameter.

18. Reads the TOC2 member *Application Format* for a given application number.
19. Determines application data based on the application format:
    a. The application start address.
    b. The start address within the application for a digital signature verification.
    c. The size in bytes of the application area to be used for a digital signature verification.
    d. The start address of a memory region within the application where the digital signature is stored.
20. If the application format is not basic, get the public key object address from TOC2.
21. Verifies if the basic application format is used when *CPUSS.PROTECTION = SECURE*. If it is not, the function returns *False*.
22. Verifies if the application start address is within the device internal memories (SRAM, Main Flash, SFLASH, EmEEPROM).
23. If the application format is not CyBAF, verifies if:
    a. The application area for which the digital signature is calculated is within the device internal memories.
    b. The application digital signature section is within the device internal memories.

8.5.4.4.7 Function ResetHandler

FLASH BOOT: ResetHandler _1
At the start of this function, the stack pointer (SP) register value is zero, so any stack operations will cause a Hard fault.

To prevent it, this function should be carefully developed. It is not feasible to implement it in the assembly, as the C code is fine. Although, when this function gets modified, a generated assembly code shall be inspected to see if no SP operations are done before the SP is initialized.

FLASH BOOT: ResetHandler _2
The function sets the stack pointer (SP) register value to the top of the SRAM minus X KB, which are reserved for SROM boot.

The X is 0 kB for BE families and 2kB for the rest of TVII devices.

The flashboot stack size is another 2K. So, the first 4K of SRAM is reserved for SROM/Flashboot.

Note, top of SRAM is MPN specific, so wounding (*CPUSS.WOUNDING*) should be applied.

Because Traveo II uses the ECC, Flash boot initializes the SRAM used for the stack.

When the stack is initialized, this function calls the *Cy_FB_Main()* function.

We calculate the actual stack pointer starting from the CY_SRAM_BASE. For all Traveo devices except BE family it is the same as CY_SRAM0_BASE

#define CY_SRAM0_BASE_STACK 0x28000000UL

For "BE" Traveo devices it is shifted up by 2 kB

#define CY_SRAM0_BASE_STACK 0x28000000UL + 0x8000UL

Then we add the size of non-wounded RAM. In the end we subtract the amount of reserved for SROM boot

#define CY_FB_RAM_RESERVED (2048u)

The resulting stack pointer equation is

STACK_POINTER = CY_SRAM0_BASE_STACK + NON_WOUNDED_RAM - CY_FB_RAM_RESERVED

The SRAM base for the entire flashboot project is also shifted by 2 kB for all Traveo devices.

#define CY_FB_SRAM_BASE (CY_SRAM0_BASE + 0x800UL)

8.5.4.4.8 Access Restrictions Handling

Access restrictions(AR) are handled by ROM boot code which sets CPUSS.AP_CTL value based on AR.

FLASH BOOT: Access Restrictions Handling
Flash boot calls ROM boot functions *GetAccessRestrictStruct()* to get the AR for a given protection state (NORMAL, NORMAL_DEAD, SECURE, SECURE_DEAD) and set CPUSS.AP_CTL value based on AR.

This is done because ROM boot sets all AP_CTL.ENABLE_XXX bits independent on AR, CDT 296939.

When CDT is fixed, Flash boot may remove the code which configures AP_CTL and use the default value of this MMIO register.

For switching into DEAD branch, Flash boot may either call ROM boot function *AssignAllPcExit()* directly, or re-implement it, if needed.

8.5.4.4.9 Enabling and Disabling DAP

Traveo II access restrictions allow either temporary or permanent DAP disabling.

Details on Traveo II access restrictions may be found in the following documents:

- MXS40 SAS (001-98134, Book2-PartII-MCU-M4).

- SAS configuration file MXS40-IP-EFUSE.xlsm field named SECURE_ACCESS_RESTRICT on page EFUSE_DATA_MXS40_ver2_Regs.
- 002-03298 *H and newer.

FLASH BOOT: Enabling and Disabling DAP _1
Flash boot function *Cy_FB_EnableDap()* is used to set *CPUSS.AP_CTL* MMIO register value with the corresponding data from the access restriction for a given life cycle stage.

Access restrictions for each life cycle stage are received by calling ROM boot's shared function *GetAccessRestrict()*.

Note In the VIRGIN life-cycle stage the access restrictions are ignored, ROM boot enables the DAP before launching Flash boot, thus Flash boot does not need to do anything.

FLASH BOOT: Enabling and Disabling DAP _2
Flash boot function *Cy_FB_IsDapControlEnabled()* is used to check if the DAP control is enabled. This function reads the *CPUSS.AP_CTL* MMIO register value and returns *True* if any *AP_CTL.XXX_ENABLE* bit is set and the corresponding *AP_CTL.XXX_DISABLE* bit is cleared.

8.5.4.4.10 Restricting Access to DAP

FLASH BOOT: Restricting Accesss to DAP
ROM boot function *RestrictAccess()* is called by Flash boot to set the MPU access restrictions for all the life-cycle stages except VIRGIN.

8.5.4.5 SRSS configuration

Flash boot uses an IMO at its input reference clock for PERI and CM0+. The FLL might be used to source the PERI and CM0+ clocks. *SFLASH.TOC2_FLAGS* contain the bit fields that define the clock settings to be configured by Flash boot.

See MXS40-IP-SFLASH.xlsm (MXS40/Platform/MXS40-SAS/Configuration/) for the *TOC2_FLAGS* description on page *SFLASH.512_ver2_Regs*.

FLASH BOOT: SRSS configuration _1
The clock settings are configured in the function *Cy_FB_ClocksSetup()*.

FLASH BOOT: SRSS configuration _2
For 8 MHz CM0+ clock frequency, the FLL is disabled and bypassed.
For 25 and 50 MHz CM0+ clock frequency, FLL is enabled and configured in *Cy_FB_ClocksSetup()*.

FLASH BOOT FOR TRAVEO II IP BROS

For the MAX CM0+ clock frequency, Flash boot uses the FLL and a peripheral setup configured by ROM boot. This allows eliminating FLL startup time which might take up to 3 us.

FLASH BOOT: SRSS configuration _3

The FLL configuration for 25 and 50 MHz is generated based on the results of the *FLL calculator* tool. FLL calculator is an Microsoft Excel file which may be downloaded from SVN: https://svn.design.cypress.com:18080/svn/chips/trunk/s40/MXS40/Platform/MXS40-SRSS/mxs40srss_ver2/Design/fll_calculator_ver2.xlsm

The SVN revision of the FLL calculator file is documented in the Flash boot file *fb_config.h* in the comment describing the FLL configuration macro definitions.

The following FLL calculator parameters are used for calculation:

Parameter Name	Cell Name	Cell Value	Comment
WCO 32kHz in use	E2	No	The input clock is an IMO
Ref clock frequency (REF):	E3	8000,00	8 MHz is the clock frequency for the IMO.
Target FLL clock frequency	E4	25000 or 50000	The frequency to set up. Either 25 or 50 MHz.
Step size parameter selection	E7	Calculated	The default value.
Optimization	E9	Best lock time	The “Best lock time” is used to optimize for faster startup time.

FLASH BOOT: SRSS configuration _4

PLL is enabled and FLL is disabled when entering the bootloader because PLL has a better accuracy which is critical for the bootloader.

FLASH BOOT: SRSS configuration _5

PLL clock is 50 MHz for LIN at 20 KBPS and for CAN.

FLASH BOOT: SRSS configuration _6
PLL clock is 49.71 MHz for LIN at 115200 BPS.

This allows achieving 115200 BPS ± 0.6 % using an integer peripheral clock divider for a LIN IP.

8.5.4.6 Hardware workarounds

FLASH BOOT - Hardware Workarounds

Flashboot could patch the bugs in the SROM code. If some of the SROM API functions needs to be changed they could be inserted into the Flashboot. In such way we could fix or patch SROM API without re-tapeout. The full list of SROm changes and patches are on
SVN https://svn.design.cypress.com:18080/svn/chips/trunk/s40/MXS40/Platform/mxs40srompsoc/Projects/mxs40srompsoc/proj/ProjectManagement/Plan/Release_Plan.xlsx

CDT 336235 "TVII: ECC RAM error can be triggered on startup if reset occured on flash write"

This patch is applicable for TVII-BE-1M B2, TVII-BE-1M B3, TVII-BE-2M A2, TVII-BE-2M A3, TVII-BE-4M A0, TVII-BH-4M A0, TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B1, TVII-BH-8M B2

Short Description: if the reset is issued during the write row system call execution, the fm_sram ecc error is observed on device boot. By default, the error is not masked fm_sram ecc error on device startup and is masked later during srom execution. The issue is that fm_sram ecc error can occur during the window of device startup and srom masking this erorr. Technically, the mxs40srompsoc code can be updated to mask the error earlier in the srom execution flow, but this will not resolve the issues completely, as error can still be risen before srom masks event.

CDT 337244 "TVII: EraseAll/CheckSum return STATUS_NVM_PROTECTED on wounded main flash"

This patch is applicable for TVII-BE-1M B2, TVII-BE-1M B3, TVII-BE-2M A2, TVII-BE-2M A3, TVII-BE-4M A0, TVII-BH-4M A0,, TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B1, TVII-BH-8M B2

Short Description: Main flash is wounded to half and the EraseAll returns STATUS_NVM_PROTECTED. This behavior is seen when FLASH_MAIN_WOUND =2. For FLASH_MAIN_WOUND =1 this issue is not seen.

The expected behavior is that both the APIs should return STATUS_SUCCESS.

CDT 338178 "mxs40srompsoc_tvii_c_2d_6m_a0: SwitchOverRegulator API not working properly"

FLASH BOOT FOR TRAVEO II IP BROS

This patch is applicable for TVII-C2D-6M A0

Short Description: The API doesn't work. This is the same issue as in CDT338841. Please have a look on 8.5.4.6.4

CDT 338841 "TVII8M: SwitchOverRegulator Issue"

This patch is applicable for TVII-BH-4M A0, TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-4M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B1, TVII-BH-8M B2

Short Description: SwitchOverRegulator API call for PMIC to LDO transition returns success but the Vccd level remain high. This issue is not seen when api parameters are passed in IPC DATA.

CDT 342981 "TVII-4M A0: ConfigureRegulator API failing"

This patch is applicable for TVII-BH-4M A0

Short Description: With the new FB v402, ConfigureRegulator API is failing and returns the STATUS_REGULATOR_ALREADY_ENABLED (0xF00000E2), even for the first time configuration. Fixed by changing the patch table adress in Flashboot.

CDT 343329 "TVII: DebugPowerUpDown SROM API failing with Power down parameter"

This patch is applicable for TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B1, TVII-BH-8M B2

Short Description: the debugPowerUpDown API must restore the previous state of CM7 cores with Power Down parameter also as it do for Power up parameter.

CDT 344750 " Writing DPSLP_REG_DIS=1 causes hang"

This patch is applicable for TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-4M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR,
TVII-BH-8M B2

Short Description: SAS says the write to DPSLP_REG_DIS is optional, which was intended to mean that if PMIC_DPSLP==1 (the option) then turn off the deepsleep regulator. It was interpreted as entirely optional even when PMIC_DPSLP==1.
* As a result, flashboot API does not touch DPSLP_REG_DIS at all.
* We have not yet detected this in PMIC validation because the FB API doesn't have it.
* We know from Mizumoto-san that silicon hangs if SAS sequence is followed, and DPSLP_REG_DIS is written high during PMIC configuration.
* I talked to GPM, and there is really no need to disable the DeepSleep regulator, because its target voltage is

below the PMIC range. This is the same voltage arrangement used for ACTIVE/SLEEP mode when PMIC_DPSLP==0, and we previously confirmed that case is ok.

CDT 350653 "update AdjustReghcVadj for LDO trim change sequence"

This patch is applicable for TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B2

Short Description: This is related to CDT 345351 where it was determined that LDO trim changes must be sequenced. The AdjustReghcVadj function used by SwitchOverRegulators and LoadRegulatorTrims API need an update for this. RevS draft SAS is updated in 4.4.10.4 with the requirements (yellow text).

The circuit requirements are in terms of time (25ns and 125ns), and the worst case clock frequency needs to be used for delays.

* If the API is always run from CM0, fastest clock is 100MHz (10ns/cy ==> 3 cycles and 13 cycles).
* If the API can run on M4 (TVII-B-E and TVII-C-E parts), fastest clock is 160MHz (6.25ns/cy ==> 4 cycles and 20 cycles)
* If the API can run on M7 (TVII-B-H, TVII-C-H/2D, TVII-CH-H parts), fastest clock is 250MHz (4ns/cy ==> 7 cycles and 32 cycles)

Because the processor clock speed is not known, it must be assumed to be the fastest possible speed when calculating the number of cycles to delay. At slow clock speeds, the actual delay may be much longer. Therefore, the API shall calculate the intended value, compare it to the existing register value, and only execute the sequence if they are different. This has the effect of keeping fast wakeups fast (eg. operating on LDO) and the long delay only happens if the PMIC is being enabled/disabled, and the sequencing delays are a negligible part of the overall transition time.

CDT 353288 "TVII-C2D-4M PSVP: TransitionToRMA and OpenRMA API calls result in HardFault"

This patch is applicable for TVII-C2D-4M A0

Short Description: When calling TransitionToRMA and openRMA APIs, IPC_DATA0 return STATUS_HARDFAULT_SYSCALL => 0xF00000F1.

After Flash Boot execution is complete, CM0+ IRQ0 and IRQ1 (syscall IRQs) are also disabled. Application needs to enable these NVICs again if application wants to trigger syscalls. This is true for any lifecycle stage and setting up of syscall IRQs is explained in the TRM (the chapter is also now updated for different use cases - like when we need another IRQ (IRQ2), PSP/MSP Issue and so on..)

If there is an application intended to be executed after successful "OpenRMA" then the application should enable IRQ0/1 just like any Normal use case, if application wants to trigger syscalls. If there is no application to be

launched after "OpenRMA" - then the debugger script should enable IRQ0/1 if syscalls are needed after "OpenRMA". This is expected behavior and we don't need any TRM updates. We have already explained in TRM that whatever be the use-case or lifecycle stage - if syscalls are needed, then IRQ0/1 should be enabled (either by application or by debugger scripts).

CDT 355545 "TVII: C2D 4M: Not enough Stack space for Call_FB_AppVerify function"

This patch is applicable for TVII-BE-1M B2, TVII-BE-1M B3, TVII-BE-2M A2, TVII-BE-2M A3, TVII-BE-4M A0, TVII-C2D-4M A0,
TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B2,

Short Description: FB_AppVerify function (which is part of Flashboot) was modified to support RSA 3k/4K and new version of this function use more stack space than previous version. The overall stack usage by the function is about grown to 1.6 Kbytes (previously it was about 800 bytes) The FB_AppVerify function is used by a FB to verify an Application in SECURE but it is also used by the TransitionToRma and OpenRma SROM system calls. When it is called from the TransitionToRma and OpenRma the function is run on the SROM stack which is 1 Kbytes of size only. Therefore, the OpenRma is now always failing with a new updates applied to FB_AppVerify.

CDT 357276 "TVII-C2D-6M: TransitionToRMA failed with STATUS_SYSCALL_HARDFAULT"

This patch is applicable for TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR

Short Description: "With FB v512, transitionToRMA SROM API returns status 0xF00000F1 i.e. STATUS_SYSCALL_HARDFAULT.With FB v512, transitionToRMA SROM API returns status 0xF00000F1 i.e. STATUS_SYSCALL_HARDFAULT." Fixed in build 515 with the right patches for RMA

CDT 358597 " TVII-C2D-6M: LoadRegulatorTrims API fails with STATUS INVALID_OPCODE"

This patch is applicable for TVII-C2D-6M A0

Short Description: "After debugging we figured out that the patches were not included into the HEX. I fixed the problem in build #520.

CDT 340284 "TVII: Create system call for regulator trim change "

This patch is applicable for TVII-BH-4M A0

Short Description: To create a system call for regulator trim change in SRSS_SECURE region. This is a workaround for a problem with the regulators. Basically, it is a function that copies one of two SFLASH values to PWR_TRIM_HT_PWRSYS_CTL.

FLASH BOOT FOR TRAVEO II IP BROS

CDT 347931 "Handling of CM0_PC_CTL.VALID[3:1] and programming of PC & PC_MASK_15_TO_1 at IP"

This patch is applicable for TVII-C2D-4M A0

Short Description: To extend the current flashboot sequence by the following commands:
5. Set VIDEOSS.INFRA.VRPU.RDx_CTL.PC_MASK_15_TO_1 to allow all PCx
6. Set VIDEOSS.INFRA.VRPU.WRx_CTL.PC_MASK_15_TO_1 to allow all PCx
7. Set VIDEOSS.INFRA.GFX_MPU_RD[x].RD_CTL.PC to same PC that in step 4
8. Set VIDEOSS.INFRA.GFX_MPU_WR[x].WR_CTL.PC to same PC that in step 4

CDT 349003 "Need to add REGHC_PMIC_STATUS_WAIT functionality to ConfigureRegulator API"

This patch is applicable for TVII-BH-4M A1, TVII-C2D-6M A0, TVII-C2D-6M B0, TVII-C2D-6M DRR, TVII-BH-8M B2

Short Description: REGHC_PMIC_STATUS_WAIT value as part of the PWR_REGHC_CTL register is not part of the ConfigureRegulator API.

From validation testing REGHC_PMIC_STATUS_WAIT (set to max value between 2ms - 4ms) is recommended (mandatory for our case) for the SBD isolation configuration.

CDT	Device	Title
336235	TVII-BE-1M B2 TVII-BE-1M B3 TVII-BE-2M A2 TVII-BE-2M A3 TVII-BE-4M A0 TVII-BH-4M A0 TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B1 TVII-BH-8M B2	TVII: ECC RAM error can be triggered on startup if reset occured on flash write

337244	TVII-BE-1M B2 TVII-BE-1M B3 TVII-BE-2M A2 TVII-BE-2M A3 TVII-BE-4M A0 TVII-BH-4M A0 TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B1 TVII-BH-8M B2	TVII: EraseAll/CheckSum return STATUS_NVM_PROTECTED on wounded main flash
338178	TVII-C2D-6M A0	mxs40srompsoc_tvii_c_2d_6m_a0: SwitchOverRegulator API not working properly
338841	TVII-BH-4M A0 TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-4M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B1 TVII-BH-8M B2	TVII8M: SwitchOverRegulator Issue
342981	TVII-BH-4M A0	TVII-4M A0: ConfigureRegulator API failing
343329	TVII-BH-4M A0 TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B1 TVII-BH-8M B2	TVII: DebugPowerUpDown SROM API failing with Power down parameter
344750	TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-4M A0 TVII-C2D-6M B0	Writing DPSLP_REG_DIS=1 causes hang

	TVII-C2D-6M DRR TVII-BH-8M B2	
350653	TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B2	update AdjustReghcVadj for LDO trim change sequence
353288	TVII-C2D-4M A0	TVII-C2D-4M PSVP: TransitionToRMA and OpenRMA API calls result in HardFault
355545	TVII-BE-1M B2 TVII-BE-1M B3 TVII-BE-2M A2 TVII-BE-2M A3 TVII-BE-4M A0 TVII-C2D-4M A0 TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B2	TVII: C2D 4M: Not enough Stack space for Call_FB_AppVerify function
357276	TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR	TVII-C2D-6M: TransitionToRMA failed with STATUS_SYSCALL_HARDFAULT
358597	TVII-C2D-6M A0	TVII-C2D-6M: LoadRegulatorTrims API fails with STATUS INVALID_OPCODE
340284	TVII-BH-4M A0	TVII: Create system call for regulator trim change
347931	TVII-C2D-4M A0	Handling of CM0_PC_CTL.VALID[3:1] and programming of PC & PC_MASK_15_TO_1 at IP

349003	TVII-BH-4M A1 TVII-C2D-6M A0 TVII-C2D-6M B0 TVII-C2D-6M DRR TVII-BH-8M B2	Need to add REGHC_PMIC_STATUS_WAIT functionality to ConfigureRegulator API

Following HW workarounds are implemented in the Flashboot code:

Silicon CDT	Flash boot CDT	Flash boot fix description
295366	300795	**Root cause:** SRSS bandgap reference does not enable S&H mode because it needs a software help for this. Problem causes an increased current consumption, approximately by 20 uA in ACTIVE/SLEEP, and 5.5 uA in DEEP-SLEEP. **Solution summary:** Flash boot sets a bit named BPGREF_LPMODE in SRSS_PWR_CTL2 register which enable S&H mode for SRSS bandgap reference.
320581	324184	Flash boot should not touch CPUSS.CM7_{x}_VECTOR_TABLE_BASE registers to set the state of CM7_{x} cores. Touching these registers when CM7 is powered off may cause an undefined behavior, what was seen in CDT 320581. CM0_ and CM4_VECTOR_TABLE_BASE are set to 0xFFFF_0000 when there is no application, but CM7_{x}_VECTOR_TABLE_BASE stays in the default state.

8.5.4.7 Hard-fault handling

Hard-fault handling is removed from Flash boot due to CDT 325878.
The hard-faults are handled by the Hard-fault handler implemented in the ROM boot code.

8.5.4.8 New and Patched system calls

Flash boot may add new and patch existing system calls. Their list and behavior is provided in this section.

FLASH BOOT FOR TRAVEO II IP BROS

For TVII-BE-1M B0, TVII-BE-2M A0, and TVII-BE-8M A0 flash boot adds 1 new system call and patches 0 existing.

New System calls:

1. WriteDeviceKeys

2. ProgramRow

3. LoadRegulatorTrims

They are imported to Flash boot from ROM boot. They are described in 002-03298 in section "SROM API".

8.5.4.8.1 WriteDeviceKey

The WriteDeviceKeys() system call is originally written only for Flashboot. In other words, it is not ported from SROM.

WriteDeviceKeys system call does the following:

1. Checks that OPCODE bits 0 and 1 are both zero in IPC_STRUCT[*TC*].DATA0. This forces the system call to use memory (SRAM_SCRATCH).
Note *TC* value is 2 for the devices with one CM4 or CM7 core, and 3 for the devices with two CM4 or CM7 cores.

2. Set the return address for a status/error code to SRAM_SCRATCH+0.
Note SRAM_SCRATCH is a value of *IPC_STRUCT[TC].DATA0*, when its bits 0 and 1 are zero.

3. Returns an error code if FF P-bits in EFUSE_DATA.BOOTROW are non-zero.

4. Returns an error code if any single bit in EFUSE_DATA.DEVICE_SECRET_KEY or EFUSE_DATA.DEVICE_SECRET_KEY_ZERO is non-zero

5. Returns an error code if this system call is performed by CM0+ AP or CM4 AP. Only SYS-AP is allowed

6. Returns an error code if CPUSS.PROTECTION != NORMAL.
Note This system call should not work in VIRGIN/SORT protection modes.

7. Returns an error code if the new device secret key it *trivial*. Here a *trivial* key means that all bits of this key are either zero or one.

8. Blows EFUSE_DATA.DEVICE_SECRET_KEY with the device secret key from SRAM_SCRATCH.

9. Returns an error if EFUSE_DATA.DEVICE_SECRET_KEY is not the same as device secret key in SRAM_SCRATCH.

10. Calculates the number of zeros in device secret key in SRAM_SCRATCH.

11. Blows EFUSE_DATA.DEVICE_SECRET_KEY_ZEROS with the value from (10).

12. Returns an error if EFUSE_DATA.DEVICE_SECRET_KEY_ZEROS is not equal to the value received in (10).

13. Blows P-bits in EFUSE_DATA.BOOTROW.

14. Returns an error if EFUSE_DATA.BOOTROW does not have both P-bit correctly blown.

Note This system call ignores the FWPU and FWPU configuration settings when reading or writing EFUSE data.

Note This system call is accessible only in NORMAL protection-mode.

Note This system call is present only in TVII families, except TVII-BE-1M**. It is not present in any PSoC6 device family.

WriteDeviceKeys must contain parameters in SRAM_SCRATCH and SRAM_BUF buffers in SRAM.

Address	Data
IPC[x].DATA0	A pointer to SRAM_SCRATCH
SRAM_SCRATCH+0	OPCODE
SRAM_SCRATCH+4	A pointer to SRAM_BUF
SRAM_BUF+0	Device Secret Key, word #0
SRAM_BUF+4	Device Secret Key, word #1
…	…
SRAM_BUF+28	Device Secret Key, word #7

SRAM_SCRATCH and SRAM_BUF shall be placed in SRAM and shall be 4 byte aligned.

SAM_BUF is used instead of appending the data to SRAM_SCRATCH due to CDT 321414.

OPCODE is 0x3500_0000.

WriteDeviceKey limitations had been found in CDT 324604.

The following decision had been taken:

1. WriteDeviceKey should not be called while CAN/LIN bootloader in Flash boot is being active and enabled.

2. WriteDeviceKey should be called only when CM0+ is clocked by IMO.

8.5.4.8.2 Read Syscall Version

This syscall reads the pre-compiled version of SROM API and returns it to a call...

This syscall reads the pre-compiled version of SROM API and returns it to a caller. It locates in the syscall patch table with the shift of 0x3F from the beginning.The pre-compiled version is taken from this file on SVN https://svn.design.cypress.com:18080/svn/chips/trunk/s40/MXS40/Platform/mxs40srompsoc/Projects/mxs40srompsoc/proj/ProjectManagement/Plan/patches_Versioning.xlsx Based on this document we could always tell what API are patched and what is the version of the patches.

Arguments if IPC_STRUCT.DATA[0]=1:

IPC_STRUCT.DATA	
Bits [6:1]	Bit[0]
Syscall Number	1 (indicates all arguments are passed in DATA)

Arguments if IPC_STRUCT.DATA[0]=0:

IPC_STRUCT.DATA
Bits [31:0]
SRAM_SCRATCH_ADDR (Address of SRAM where the API's parameters are stored. Must be a 32 bit aligned address)

SRAM_SCRATCH	
Bits[31:7]	Bits[6:1]
N/A	Syscall Number

Return if IPC_STRUCT.DATA[0]=1 :

IPC_STRUCT.DATA	
Bits [3:0]	System call version

Return if IPC_STRUCT.DATA[0]=0:

SRAM_SCRATCH	
Bits [3:0]	System call version

The Excel file on SVN and the patch version in the file "fb_common.h" file must be updated every time we are patching the SROM API.

8.5.4.9 MISRA compliance

FLASH BOOT: MISRA Compliance _1
Flash boot C source code passes through a MISRA-C:2012 compliance run by PRQA Framework 2.3.1 framework with QAC-9.4.1 and M3CM-2.3.4 components.

FLASH BOOT: MISRA Compliance _2
The list of the project deviations is stored in *{}/mxs40/src/fb_doxygen.h* file. The format of the MISRA deviations list is the same as in the PDL library – a C language comment, doxygen compatible and containing an XML table.

The C files from SDL and PDL Cypress libraries used by Flash boot are not part of Flash boot MISRA-C:2012 compliance testing according to CDT 298412. **Note** PDL has its own MISRA-C runs, SDL has none.

8.5.4.10 Flash dual banking mode

FLASH BOOT FOR TRAVEO II IP BROS

MXS40 Flash implementations (SONOS, ECT) support dual banking mode. However, the MMIO registers for switching to dual banking mode are non-retention. This means their state is cleared on a hardware reset or on wake-up from Hibernate.

This implies Flash boot is always launched when Flash is in single bank mode.

8.5.5 Flash boot version

Flash boot versioning documentation is maintained in ALXD-31.

The Flash boot version can be found in the MXS40-IP-SFLASH.xlsm file (MXS40/Platform/MXS40-SAS/Configuration/) -> the TVIIBH8M_ByteMap sheet at row 16 and addresses 1700_2018 and 1700_2019. Also, it can be read from SFLASH using OpenOCD.

Address	Value	Name
1700_2018	0xAA	FLASH_BOOT_VERSION_LOW0
1700_2019	0x01	FLASH_BOOT_VERSION_LOW1

In this example table the Flash boot version is 0x01AA which corresponds to 426 in decimal numbers.

8.6 Verification and Validation

8.6.1 Verification by Development

Static unit tests are not applicable because it is a firmware, not a library. However, the Jenkins job for Flash boot performs compiling the source code and generates build reports with the list of build errors and warnings (for more details refer to YBER-173). So, technically this procedure can be considered as an automated static unit test.

Full Flash boot firmware testing has to be done on a PSVP before testing a silicon to prevent a silicon damage in the case of an invalid firmware.

When working with a silicon, ensure the firmware has been tested and remember that EFUSE cannot be returned to the previous state.

The list of all HW registers modified by flashboot is in memo ANRY-386 This list gives an understanding what the impact on HW the flashboot could have. And what is the dependency between the HW and the flashboot.

8.6.2 Verification by Test

Test verification strategy is documented in:

VLAD-355 – TRAVEO II FLASHBOOT TEST DESIGN REVIEW REPORT

002-11462 – MXS40 CORE PLATFORM VERIFICATION REQUIREMENTS

8.6.3 Validation by Test

The formal test plan is documented in the following memo

VMEL-96 *A – FLASHBOOT DRIVER TEST PLAN FOR TRAVEO II

9 QUALITY REQUIREMENTS: N/A

10 RECORDS

Storage location and retention period for records is specified in procedure 00-00064 (Cypress Record Retention Policy).

11 PREVENTIVE MAINTENANCE: N/A

12 POSTING SHEETS/FORMS/APPENDIX: N/A

FLASH BOOT FOR TRAVEO II IP BROS

The following information is a Polarion specific wiki content which is intended to describe the current status of this document in the Polarion ALM system.

Info: NOTE: Document hints are enabled. They can be disabled in this wiki.

Property	Value
Project ID	138944
Project/Product Name	SW_TVII_MXS40BOOT
Document Revision	1.00
Date	2020-03-10
Document Status	InWork
Author	YIDA, ANRY
Approver	

Appendix 1

(This page is not yet in use. Replace this text with your own content. Do ***not*** delete this page.)

13 DOCUMENT HISTORY

REV	ECN No.	Orig. of Changes	Description of change
**	5870442	ALXD	Initial Release.
*A	5993146	ALXD	1. Added numerous ALXD memos to References. 2. Section 8.1, updated from CMAC to SHA256. 3. Section 8.6.2 updated with the references to memo. 4. Sections 8.2.3, 8.5.3, 8.5.4.1, 8.5.5, 8.6.3 fixed TBD milestones. 5. Section 8.5.4.1.1, updated CAN and LIN configuration. 6. Section 8.5.4.2, removed FitBit and Secure Image related text. 7. Section 8.5.4.4, CDT 289764 fix. 8. Added section 8.5.4.1.2 “Bootloadable Application Requirements” 9. Updated section 8.5.4.4. Added a subsection for each function description. 10. Added section 8.5.5, Flash boot version per CDT 292161. 11. Replaced *italic* text with regular for most section bodies.

FLASH BOOT FOR TRAVEO II IP BROS

REV	ECN No.	Orig. of Changes	Description of change
*B	6103629	ALXD	1. CDT 296566 changes throughout the document: a. Section 8.1, updated content. b. Section 8.5.1, fixed a few typos. c. Section 8.5.4.2, updated TOC description. d. Section 8.5.4.3, fixed a vague explanation. e. Section 8.5.4.4.1, fixed function description. 2. CDT 296940, Updated the description for enabling the DAP in Section 8.5.4.5 3. CDT 300366, Update the description for SRSS configuration in Section 8.5.4.6. 4. CDT 300538, Updated CAN and LIN prescalers description in Sections 8.5.4.1.2, and 8.5.4.1.3. Added figures for CAN and LIN transceivers connection to MCU. 5. Section 8.5.4.1. Mentioned the Bootloader SDK and CyMCUElfTool. 6. Section 8.5.4.1.1. Added the figures with the examples of the time flow for CAN and LIN bootloading. 7. Section 8.5.4.4.5. Updated the procedure to launch a user application. 8. Added Section 8.5.4.1.2.2 “CAN limitations” 9. Added Section 8.5.4.6 describing SRSS configuration. 10. Added Section 8.5.4.7 listing the hardware workarounds. 11. Removed references to ALXD-27, 32, 33 and 36. Added references to 002-12095, 002-13924, 002-20889, 002-22934. Added references to YBER-173, VLAD-355. 12. Document proofread changes by RPSM. Updated sections 1.1, 1.3, 7.1, 8.1, 8.2, 8.2.1, 8.2.3, 8.2.4, 8.2.5, 8.2.6, 8.2.7, 8.5.1, 8.5.1.1, 8.5.1.2, 8.5.1.3, 8.5.1.4, 8.5.2, 8.5.2.1-8.5.2.9, 8.5.2.12, 8.5.4, 8.5.4.1, 8.5.4.1.1, 8.5.4.1.4, 8.5.4.2, 8.5.4.3, 8.5

FLASH BOOT FOR TRAVEO II IP BROS

REV	ECN No.	Orig. of Changes	Description of change
			.4.4, 8.5.4.4.1, 8.5.4.4.2, 8.5.4.4.3, 8.5.4.4.4, 8.5.4.4.5, 8.5.4.4.6, 8.5.4.4.7, 8.5.4.5.1, 8.5.4.5.2, 8.5.4.6, 8.5.5, 8.6.1.

FLASH BOOT FOR TRAVEO II IP BROS

REV	ECN No.	Orig. of Changes	Description of change
*C	6219269	ALXD	1. Section 3.12. Added reference to 002-19314. 2. Section 8.1. Updated section content based on the implementation details. CDT 304283 item #1. 3. Section 8.5.1. Clarified on reentrancy and external access to shared functions. CDT 304283 item #2. 4. Replaced absolute SFLASH address with SFLASH object names or offsets. CDT 304283 item #4. 5. Section 8.5.3. Added a description for a stack usage calculation. 6. Section 8.5.4. Removed figure, added reference to 002-19761 instead. CDT 304283 item #3. 7. Section 8.5.4.1. Added reference to 002-19314 for more details on bootloading. 8. Section 8.5.4.5.1. Updated the note text on access restrictions in VIRGIN protection state. CDT 304283 item #5. 9. Section 8.5.5. Removed the section content which is outdated now, added a reference to AXLD-31 which maintains the details on versioning.
*D	6424481	ALXD	1. Section 7.1. Fixed a typo. 2. Section 8.1. Added the description for authentication. 3. Section 8.5.4.4.8. Added references to ROM boot function used to configure CPUSS.AP_CTL based on Access Restrictions. 4. CDT 315120. Section 8.5.4. Added the references to SAS for application formats. 5. Added section 8.5.4.7 Hard-fault handling. 6. CDT 322947. Added section 8.5.4.1.5 Security enhancements for bootloading. 7. Added section 8.5.4.8 New and Patched system calls. 8. CDT 298412. Added section 8.5.4.9 MISRA compliance.

REV	ECN No.	Orig. of Changes	Description of change
			9. Sections 8.5.4.1.2, 8.5.4.1.3. Updated to TVII-BE-1M *A and newer, including figures.
*E	6594007	ANRY, ALXD	1. Added section 8.5.4.1.2.2 which provides CAN bootloading performance measurements from CDT 323734. 2. Updated section 8.5.4.1.3.3. LIN bootloading timing measurements from CDT 323734 3. Section 8.5.4.7 removed the previous content. The new one describes the reason for removal of hard-fault handling in Flash boot (CDT 325878). 4. Section 8.5.4.8.3.3. Added *WriteDeviceKey* limitations. 5. Added section 8.5.4.10, CDT 329610. 6. Sections 8.6.2, 8.6.3. Updated the references to the validation documents. 7. Fixed naming throughout the document: a. *Flashboot* and *Flash Boot* to *Flash boot*. b. *SFlash* to *SFLASH.* c. *RAM* to *SRAM*. d. *Flash* and *User Flash* to *Main Flash.*
*F	6689053	ANRY	1. Section 8.5.4.1.2 Added CAN Enable pins for TVII-C-2D-6M 2. Section 8.5.4.3 Secure application digital signature verification. Add RSA3K and 4K support

FLASH BOOT FOR TRAVEO II IP BROS

REV	ECN No.	Orig. of Changes	Description of change
*G	6720891	ANRY	8.5.4.4.7 Add information about memory usage in different scenarios.
*H	6750188	ANRY	8.5.4.8.31.3 Add information from CDT 328665 about removing some usage limitations from Flashboot 8.5.4.8.1 and 8.5.4.8.2 Add description of ProgramRow API System Call per CDT 344572
*I	6831014	YIDA, ANRY	From this revision rev. *I this document will be updated with a Polarion file version of this document. Updated requirements in "CAN configuration" section (8.6.4.1.2) Updated document structure. Merged SEROS 002-19761 rev. *F into this BROS document. SEROS 002-19761 will be fully reworked from scratch in next revision as a Software requirement document. Updated all flowcharts into UML format. Added new acronyms to the Definitions chapter Added active links into the reference documents, removed obsolete links Added links to relevant DOFE and ANRY memos Added comments to the flowcharts, made them fit MS Word page size Described the TOC2 contents, added information from source code Added the number of current Flash boot revision Updated sections related to CAN and LIN Removed obsolete documents 001-42390, 001-42697, 001-58992, 001-80844. Traveo II is not a PSoC Creator component. Added new requirements in "Secure application digital signature verification" section (8.6.4.3) Added new requirements in "ProgramRow API system call" section (8.5.4.8)

FLASH BOOT FOR TRAVEO II IP BROS

REV	ECN No.	Orig. of Changes	Description of change
			Added new requirements in "WriteDeviceKeys system call" section (8.5.4.8) Rejected duplicate requirements in "ProgramRow API system call" section (8.5.4.8)
*J	6967435	ANRY	1. Add Table1 and Table 2 with the CAN and LIN pinouts 2. Added description about the flashboot execution time (8.5.4.3) 3. In the description of the Reset Handler add about stack reserved for Flashboot (8.5.4.4.7) 4. In the section Bootloading added about reserved SRAM for stack (8.5.4.1) Jira AAPS-225 5. In the section Set Error Code (8.1.2.1) added new error code for ECC Check

FLASH BOOT FOR TRAVEO II IP BROS

REV	ECN No.	Orig. of Changes	Description of change
			(CDT 362423) 6. In the section Definitions changed "PSVP" definition 7. In the Figure 1 changed from "from ROM boot" to "Entry after the ROM boot" 8. In 8.1.1 extend the sentence to "The Flash boot is the firmware that runs on the security processor (Cortex-M0+) and is executed after ROM boot has completed basic hardware configuration, trim setting as also validation of the Flashboot image. " 9. In 8.1.1 add some minor changes to improve text clearness and quality 10. In the description of "**Set-up SP (2)**" add about minus 2kB 11. In the description of "**Set PC=2 (16)**" add following "PC will be set to 2 only in case when Protection state is not 'DEAD'. In DEAD PC will be set to 4." 12. In the description of "**Launch CM0+ application (24)**" reset numeration to 1 13. In 8.1.2.1 add ". IPC=3 only in case of 2 CM4/CM7 cores." 14. In 8.2.6 add MiniProg4 15. Extend "FLASH BOOT Public Import Interfaces" with 10 new API functions 16. In 8.5.4.1 add links to TRMs on SVN for all Traveo families 17. In "Bootloader Overview _step (E): Launch the Flash Loader" extend the description of reserved space 18. In 8.5.4.2 add Family specific Excel files 19. Removed unused error codes 20. In 8.5.4.6 add description of all boot patches
*K	7146919	ANRY	1. In 8.5.4.8 Add description of void Cy_FB_GetSyscallPatchVersion(uint32_t ipcStruct) 2. In FLASH BOOT Overview _step 26: Branch DEAD add reasons for entering the dead state 3. In 8.5.4.2 (TOC2) add description of security marker
*L	7733252	ANRY	Add 8.5.4.8.1 WriteDeviceKey Syscall Description