Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN220242

Flash Rewriting Procedure for Traveo II Family

Author: Hachisu, Kotaro

Associated Part Family: Traveo™ II Family CYT2/CYT3/CYT4 Series

Associated Code Examples: None

Related Application Notes: see Related Documents.

This application note describes flash rewriting procedure and provides examples on how to use this function for Cypress Traveo II family MCU. The document also explains the feature of eCT40 flash memory, block diagram, sector configuration, and code flash rewriting examples using the SROM API operation.

Contents

1 Introduction

This application note describes flash rewriting procedure for Cypress Traveo II family MCU. Traveo II family has code flash, work flash, and supervisory flash. Code flash is part of flash memory used to store user programs and work flash is used to store critical nonvolatile data or parameters. Supervisory flash is used to store the flash boot code or public key for secure boot operation. Flash programing operations such as “Erase All”, “Program Row”, and so on are executed from the core Arm® Cortex®-M0+ using its IRQ0 interrupt via system calls. This application note describes some use cases with software configuration flow for operations like erase and program of the flash memory.

To understand the functionality described and terminology used in this application note, see the “Code Flash”, “Work Flash”, and “Nonvolatile Memory Programming” chapters of the Architecture Technical Reference Manual (TRM).

2 Overview of Flash Memory

2.1 Features of Flash Memory

Traveo II family of devices use eCT flash memory which is portioned into code, work, and supervisory-flash regions. This application note targets only code and work flash regions. Table 1 lists the features of both code flash and work flash for CYT4B series. See the Device Datasheet for the memory size available for each device.

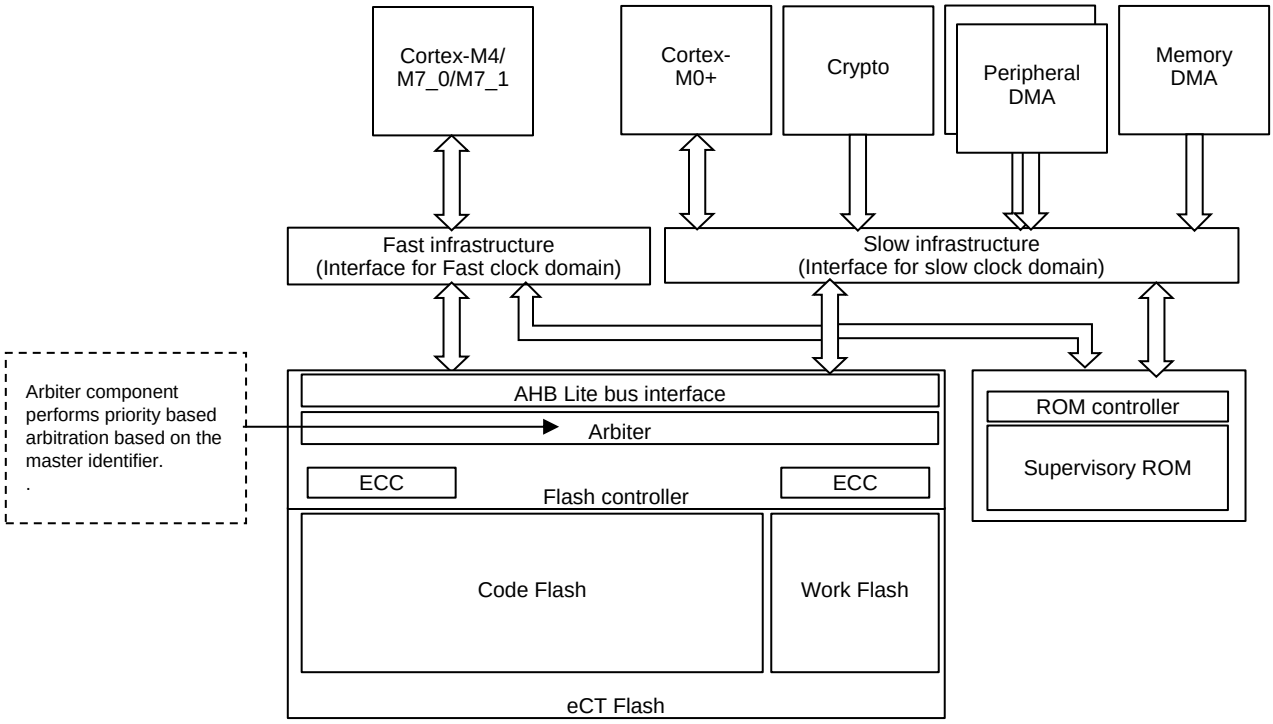
Table 1. Overview of Flash Memory for CYT4B Series

Feature	Code Flash	Work Flash
Memory size	Up to 8384 KB (8128 KB + 256 KB)	Up to 256 KB (192 KB + 64 KB)
Program size	64 bit, 256 bit, 4096 bit	32 bit
ECC function	64 bit + 8 bit Single Error Correction, Double Error Detection (SECDED)	32 bit + 7 bit Single Error Correction, Double Error Detection (SECDED)
Erase sector size	32 KB for large sector and 8 KB for small sector	2 KB for large sector and 128 B for small sector
Security	Supported	Supported
Single Bank and Dual Bank modes	Supported	Supported
Program execution	Supported	Not supported
Reading while programming/erasing	Supported	Supported
Program/Erase cycles/ Data retention time @85ºC	1,000/20 years	250,000/10 years

2.2 Block Diagram

eCT Flash is a part of the CPU subsystem. The Cortex-M4/-M7_0/-M7_1 and CM0+ CPU cores can access eCT Flash via Fast/Slow infrastructures respectively, but according to the design of Traveo II family of devices only CM0+ core can write into code flash and work flash by executing SROM APIs. Thus, user application on CM4/CM7_0/CM7_1 needs to send a command via IPC; then SROM APIs are executed within CM0+ IRQ0 handler.

Figure 1. Block Diagram of Flash Memory I/F



Note: Figure 1 is an example of a few blocks. See the Architecture TRM for details.

Table 2 lists the master identifiers.

Table 2. Arbiter and Bus Master

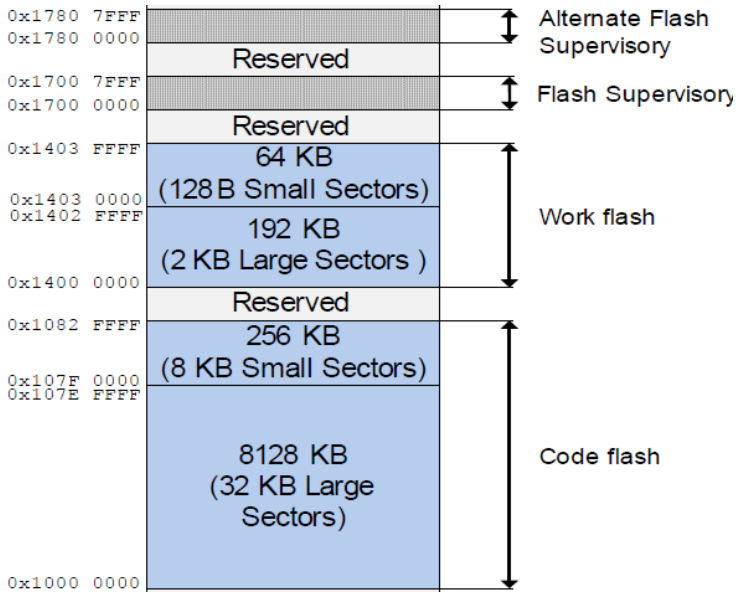
Master Identifier	Bus Master			
	CYT2 Series	CYT3 Series	CYT4B Series	CYT4D Series
0	CM0+ CPU	CM0+ CPU	CM0+ CPU	CM0+ CPU
1	CRYPTO Component	CRYPTO Component	CRYPTO Component	CRYPTO Component
2	P-DMA 0	P-DMA 0	P-DMA 0	P-DMA 0
3	P-DMA 1	P-DMA 1	P-DMA 1	P-DMA 1
4	M-DMA	M-DMA	M-DMA	M-DMA
5	-	SDHC	SDHC	-
9	-	Ethernet 0	Ethernet 1	Ethernet 1
10	-	-	Ethernet 0	Ethernet 0
12	-	-	-	Audio Subsystems
13	-	CM7_1 CPU	CM7_0 CPU	CM7_0 CPU
14	CM4 CPU	CM7_0 CPU	CM7_1 CPU	CM7_1 CPU
15	Test Controller	DAP Tap Controller	Test Controller	Test Controller

2.3 Sector Configuration

2.3.1 Sector Configuration

Figure 2 shows the flash sector configuration for CYT4B series. The CYT4B series has 8128 KB (32 KB large sectors) and 256 KB (8 KB small sectors) of code flash with an additional work flash. Work flash has been optimized to be able to reprogram the data several times more than code flash. Supervisory region stores trim parameters for hard IP, system configuration parameters, protection and security setting, boot script, and so on. Density of flash memory size and sector configuration depends on product specification. For more details, see the Device Datasheet and Architecture TRM.

Figure 2. Sector Configuration of CYT4B Series



2.3.2 Bank Mode and Remap Functionality

2.3.2.1 Single Bank Mode

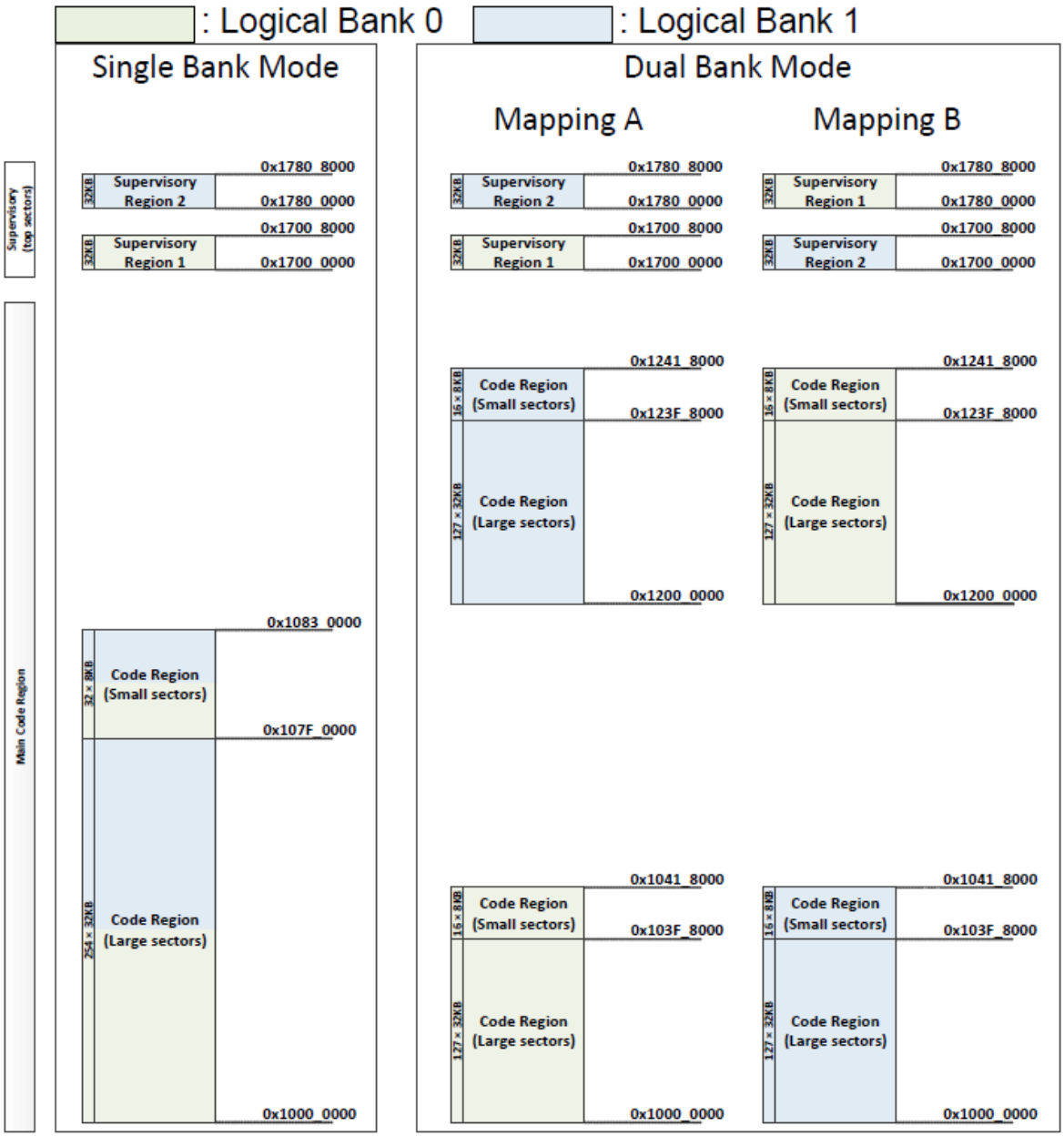
The entire code/work and supervisory logical regions are mapped as single contiguous address regions.

2.3.2.2 Dual Bank Mode

Flash mapping is split into two halves, and each half is presented as a separate address region. In this mode, program execution can be swapped to support same-location firmware updates.

Figure 3 shows the code flash memory mapping for CYT4B series. Code flash and work flash support both single bank mode and dual bank mode. Dual bank mode and read-while-write operation (RWW) can be used for firmware update operation. Density of flash memory size and memory mapping depends on product specification. For more details, see the Device Datasheet and Architecture TRM.

Figure 3. Memory Mapping in Single/Dual Bank Mode for CYT4B Series



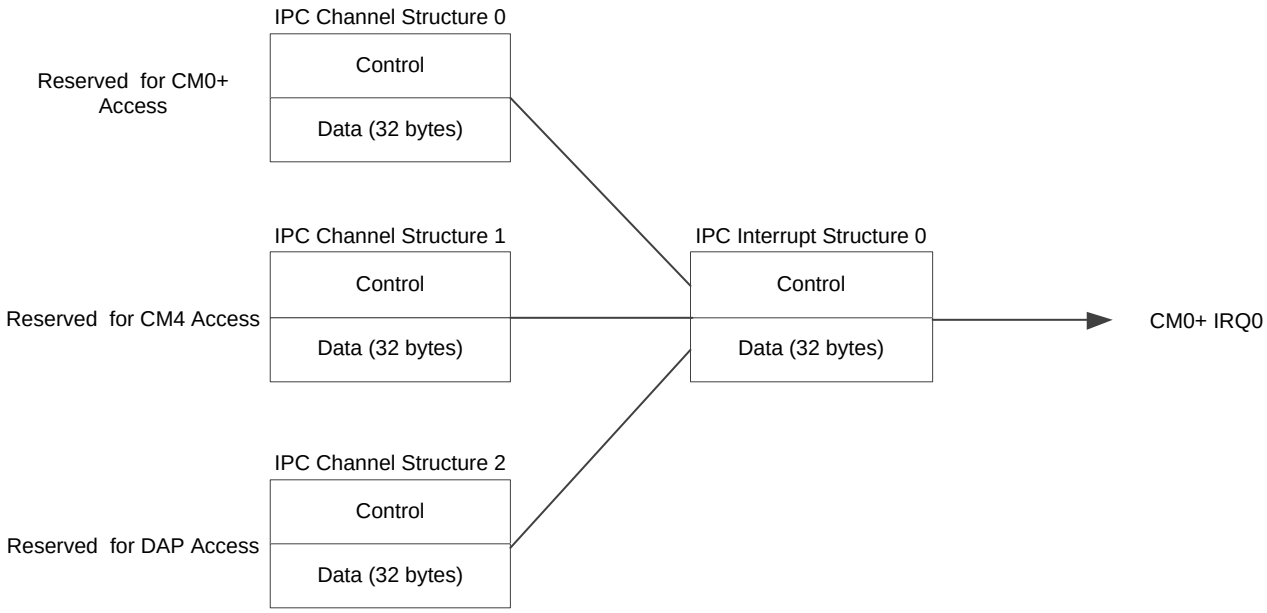
3 Flash Rewriting Procedure

This section shows an example of flash erase of all operation and program row operation that programs the 64-bit data to code flash.

In Traveo II family, there is no automatic algorithm and command sequencer embedded as in Traveo family, but flash operations are implemented as system calls. System calls are executed inside CM0+ IRQ0. You do not have access to modify the SROM code. The CM4/CM7_0/CM7_1 user code requests the system call by acquiring the Inter-processor communication (IPC) and writing the SROM function opcode and parameters to IPC DATA register. As a result, an IRQ0 interrupt is invoked and the requested SROM API is executed.

Figure 4 shows the system call interface using IPC in CYT2B Series. System calls can be performed by CM0+, CM4, or DAP. Each of them has a reserved IPC structure through which they can request CM0+ to perform a system call. When CM4 invokes a system call via CM0+ for erasing/programming the flash memory by user software, IPC structure 1 and IPC interrupt structure 0 are required to be used. See the Device Datasheet and Architecture TRM for supported number of IPC channel structure available for each device.

Figure 4. System Call Interface Using IPC in CYT2B Series



Note: Bus error might occur when any IDE, which supports the option to view live memory, reads flash memory through DAP access during RWW operation. If the error occurs, close the debugger memory view windows or disable the debugger live memory

3.1 Setting Up IRQs for System Calls

The user software is responsible for o correctly setting up CM0+ IRQ0 and IRQ1 interrupts for system call management. The boot code automatically sets CPUSS_CM0_SYSTEM_INT_CTL0. CPU_INT_VALID bit to 1 and CPUSS_CM0_SYSTEM_INT_CTL0. CPU_INT_IDX [2:0] bits to b'000. Hence, the mapping of System Interrupt 0 (IPC Interrupt Structure 0 interrupt) to CM0+ IRQ0 for system calls is done by boot code and CM0+ IRQ0 is triggered by IPC Interrupt Structure 0 interrupt.

However, the software needs to ensure that CM0+ IRQ0 and IRQ1 are enabled and configured with the correct priorities as this is not automatically done by the boot code. Also, the software must ensure that IRQ0 and IRQ1 vector entries in the user CM0+ vector table are identical to the vector entries in the default SROM vector table (addresses 0x00000040 and 0x00000044 respectively). This can be achieved by copying values from the SROM vector table to the user vector table during runtime if it is in RAM, otherwise hard-coded values need to be used and reconfirmed if target MCU or revision changes.

When the CPU is executing code in Thread mode, the CONTROL register can be configured to use Process Stack Pointer (PSP) or Main Stack Pointer (MSP). In Handler mode, MSP is always used. Note that the CPU enters Thread mode and uses MSP when it comes out of reset. Additionally, the software must take special care while setting up the system call interrupts as this depends on the CPU mode (Thread or Handler) of CM0+ and the stack pointer (MSP or PSP) used when the system call was triggered.

Case 1

If the software triggers system calls only when CM0+ is in handler mode, then ensure that the software sets CM0+ IRQ1 with a higher priority than IRQ0. This can be done by setting IRQ0 priority to '1'. By default, IRQ1 priority will be set to '0'.

Case 2

If the software triggers system calls with any other CPU states (for instance Thread Mode and PSP) for CM0+, then the software additionally needs to use another CM0+ interrupt (such as IRQ2) which acts as a manager for system calls. This approach in principle can also be used for any CPU state (including handler mode). So, this is a more generic approach to manage system calls under all CPU states.

This approach to set up CM0+ IRQ includes the following steps:

1. Set up the system call manager function (for example, "Sys_Call_Manager") as IRQ handler for CM0+ IRQ2 in the user vector table.
2. Map the IPC Interrupt Structure 0 interrupt to CM0+ IRQ2.
3. Set the lowest priority to IRQ2 with respect to IRQ0 and IRQ1. Set the same highest priority for IRQ0 and IRQ1. For instance, set the priority of IRQ2 to 1; by default, the priority of IRQ0 and IRQ1 will 0.
4. IRQ2 handler triggers IRQ0 in software.
5. IRQ2 handler clears the Pending Bit of IRQ0.

Thus, the CM0+ vector table will have entries for the first three interrupts as shown in Table 3.

Table 3. Interrupt Handler

Interrupt Number	Handler
…	…
IRQ0	Contents of address (0x00000040)
IRQ1	Contents of address (0x00000044)
IRQ2	Sys_Call_Manager
…	…

Also note that instead of directly assigning "Sys_Call_Manager" as CM0+ IRQ2 handler, it can also be combined with multiple other system interrupts when using a dispatcher implementation.

Here is the pseudo code for the interrupt configuration needed for system call for case 2.

```
/* IRQ2 handler function for IPC Interrupt structure 0 interrupt. This is the system
call manager function */

void Sys_Call_Manager()
```

```
{
  /* Trigger IRQ0 in Software by writing to ISPR register */
  CM0P_SCS_ISPR = 1;

  /* Read back the register to ensure that the write has happened */
  CM0P_SCS_ISPR;

  /* Clear the NVIC Pending bit of IRQ0. This is done as a fallback in case the system
  call was suppressed (e.g., by disabled interrupts) */
  CM0P_SCS_ICPR = 1;

  /* Read back the register to ensure that the write has happened */
  CM0P_SCS_ICPR;

}

/* Application function for interrupt configurations */

 void interrupt_configure()
{
  /* Enable CM0+ IRQ0, IRQ1 and IRQ2 */
  CM0P_SCS_ISER = 7;

  /* Set Priority 0 for IRQ0, IRQ1 and Priority 1 for IRQ2 */
  CM0P_SCS_IPR0 = 0x00400000;

  /* Connect IPC Interrupt Structure 0 Interrupt (System Interrupt 0) to
  IRQ2. The interrupt triggers Sys_Call_Manager */
  CPUSS_CM0_SYSTEM_INT_CTL0.CPU_INT_IDX = 2;

  /* Clear the PRIMASK register to enable the interrupts. This could also
  be done by the application at a later point in time */
  __ASM("cpsie i");

}
```

3.2 Flash Erase All Procedure

Figure 5 shows flash “erase all” procedure from CM4/7 CPU core with the usage of IPC and CM0+ IRQ0 handler. This example shows the erase operation for the entire code flash, but does not include supervisory region and work flash. CM4/7 code is assumed to be placed in or executed from SRAM. The opcode for the “erase all” API is 0x0A. See the chapter “Nonvolatile Memory Programming” of the Architecture TRM for more details of parameters used in the following operation.

Note: The following sections assume IRQ0 triggers system calls which means system calls are triggered when CM0+ is in handler mode. For all other CM0+ CPU states, ensure to set up the interrupts as explained in Setting Up IRQs for System Calls.

3.2.1 Pre-configuration

A. Enable main flash embedded operations (FLASHC_FM_CTL_ECT_MAIN_FLASH_SAFETY_MainFlashWriteEnable = 1) [By CM4/7 user code].

B. Allocate four words as SRAM scratch region (uint32_t SramScratch[4];) [By CM4/7 user code].

Note: If you are using the CYT4B series which supports both of CM7_0 and CM7_1, you should consider cache coherence. SRAM area pointed by "SramScrach"/"data" is shared by multiple cores, thus, the area should not be cacheable.

3.2.2 Requesting the “Erase All” API via CM0+ IRQ0 Handler

A. Acquire lock for IPC1 channel structure [By CM4/7 user code].

B. Update the API parameter to SRAM scratch memory [By CM4/7 user code].

   a) Write the opcode of “erase all” (0x0A) to SramScratch[0] (SramScratch[0] = 0x0A000000;).

- Bits [31:24]: Erase All opcode = 0x0A
- Bits [23:00]: 0x00 (Not used)

C. Assign the SRAM scratch memory to IPC1_DATA0 [By CM4/7 user code].

D. Generate notification event via IPC_NOTIFY register [By CM4/7 user code], subsequently API notification event to CM0+ through IPC0 interrupt structure.

3.2.3 Executing "Erase All" API from CM0+ IRQ0 Handler

A. Detect API notification event via IPC0 interrupt structure [By CM0+ user code].

B. Read the SRAM_SCRACH_ADDR (&SramScratch[0]) from IPC1_DATA0 [By CM0+ IRQ0 handler].

C. Read opcode (0x0A) from SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

D. If opcode is unknown, write the failure code to SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

E. Execute erase all [By CM0+ IRQ0 Handler].

F. Write the success code to SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

G. If result fails, write the failure code to SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

H. Generate release event via IPC_RELEASE register, subsequently API notification event to CM0+ through IPC2 interrupt structure [By CM0+ IRQ0 handler].

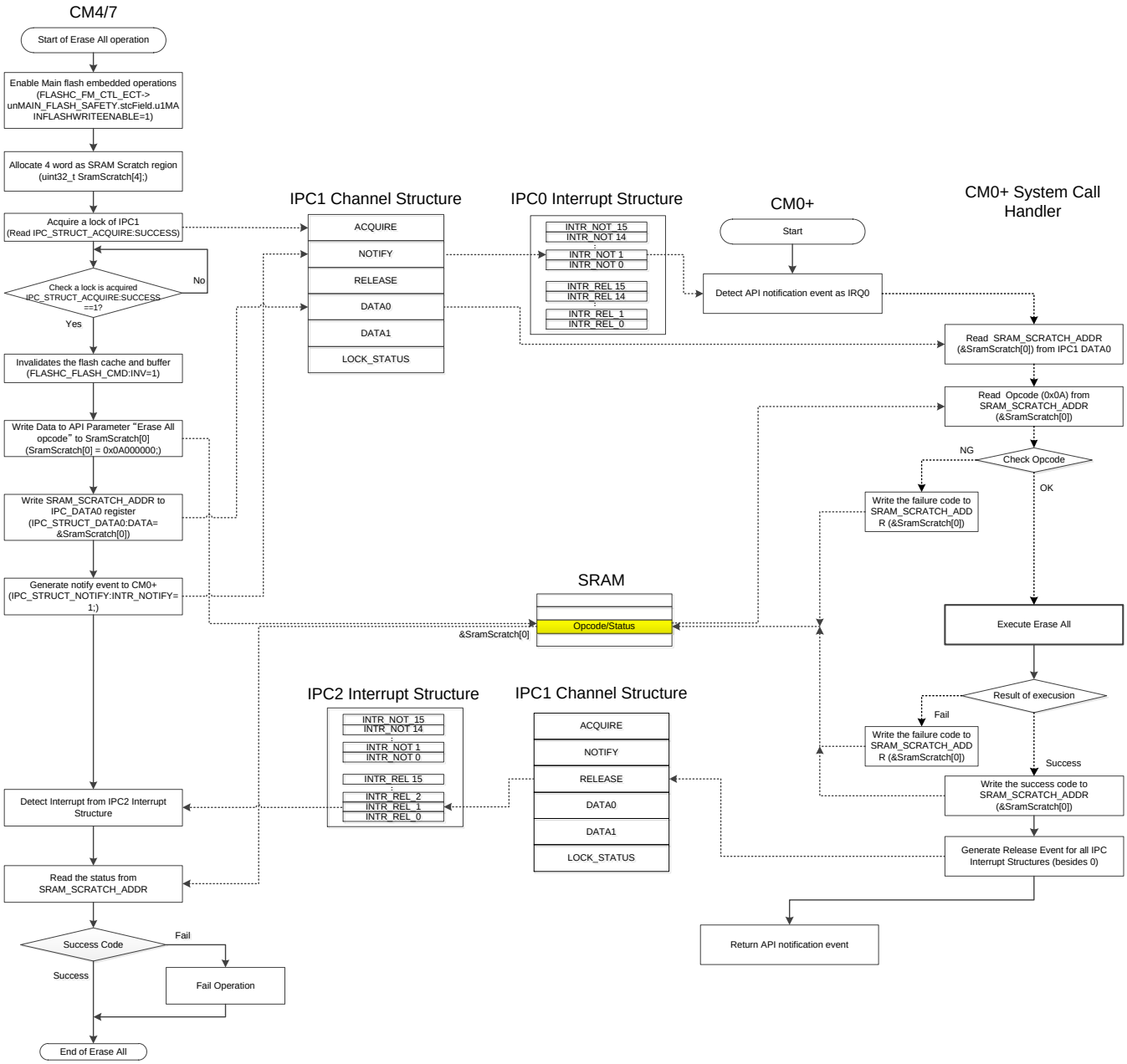
3.2.4 Assessing the Result of "Erase All" Operation from CM0+

A. Detect release interrupt from IPC2 interrupt structure [By CM4/7 user code].

B. Read the status from SRAM_SCRATCH_ADDR (&SramScratch[0]), to get the API operation result [By CM4/7 user code].

C. If status is OK, Erase All operation is completed [By CM4/7 user code].

Figure 5. Flash Erase All Procedure Example



3.3 Program Row Procedure

Figure 6 shows setting example of flash program row operation by CM4/7 and behavior of IPC and CM0+ IRQ0 handler. This example shows the write operation with 64-bit test data into code flash. The address of code flash to be written is 0x1000_0000. The test data is 0x55AA55AA x 2. Program Row API whose opcode is 0x06 is used as system call. See the chapter “Nonvolatile Memory Programming” of the Architecture TRM for more details of parameter used in the following overall operation.

Note: The following sections assume IRQ0 triggers system calls which means system calls are triggered when CM0+ is in handler mode. For all other CM0+ CPU states, make sure to set up the interrupts as explained in Setting Up IRQs for System Calls.

3.3.1 Pre-configuration of Parameter

A. Enable main flash embedded operations (FLASHC_FM_CTL_ECT_MAIN_FLASH_SAFETY_MainFlashWriteEnable = 1) [By CM4/7 user code].

B. Allocate four word as SRAM scratch region (uint32_t SramScratch[4];) [By CM4/7 user code].

C. Allocate double word as data buffer to be written into the code flash (uint32_t data[2];) [By CM4/7 user code].

Note: If you are using the CYT4B series which supports both of CM7_0 and CM7_1, you should consider cache coherence. SRAM area pointed by "SramScrach"/"data" is shared by multiple cores, thus, the area should not be cacheable.

3.3.2 Requesting the “Program Row” API to CM0+ IRQ0

A. Acquire a lock of IPC1 channel structure [By CM4/7 user code]

B. Update the API parameter to SRAM [By CM47 user code].

a) Write the opcode of Program Row (0x06) to SramScratch[0] (SramScratch[0] = 0x06000000).

- Bits [31:24]: Program Row opcode = 0x06
- Bits [23:16]: Skip blank check= 0x00 (Perform blank check)
- Bits [15:08]: Blocking mode=0x00 (non-blocking)
- Bits [08:00]: 0x00 (Not used)

b) Write the data size to SramScratch[1] (SramScratch[1] = 0x00000003;).

- Bits [31:24]: Interrupt mask, only applicable when non-blocking=0x00(FM interrupt mask not set)
- Bits [23:16]: 0x00 (Not used)
- Bits [15:08]: Data location=0x00 (Page latch)
- Bits [08:00]: Data size for code flash=0x03 (64 bits)

c) Write the flash address to be programmed to SramScratch[2] (SramScratch[2] = 0x10000000).

- Bits [31:0]: Flash address to be programmed = 0x10000000

d) Write the SRAM_SCRATCH_DATA_ADDR to SramScratch[3] (SramScratch[3] = &data[0]).

e) Write the 32-bit program data to data[0] (data[0] = 0x55AA55AA).

f) Write the 32-bit program data to data[1] (data[1] = 0x55AA55AA).

C. Assign the SRAM scratch memory to IPC1_DATA0.

D. Generate notification event by writing IPC1_NOTIFY register [By CM4/7 user code], then API notification event to CM0+ through IPC0 interrupt structure.

3.3.3 Executing “Program Row” API in CM0+ IRQ0

A. Detect API notification event via IPC0 interrupt structure [By CM0+ user code].

B. Read the SRAM_SCRATCH_ADDR (&SramScratch[0]) from IPC1_DATA0 [By CM0+ IRQ0 handler].

C. Read Opcode(0x06), program data size, target address to be programmed, program data from SRAM [By CM0+ IRQ0 handler].

D. If opcode is unknown, write the failure code to SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

E. Execute program row operation [By CM0+ IRQ0].

F. If result is successful, write the success code to SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

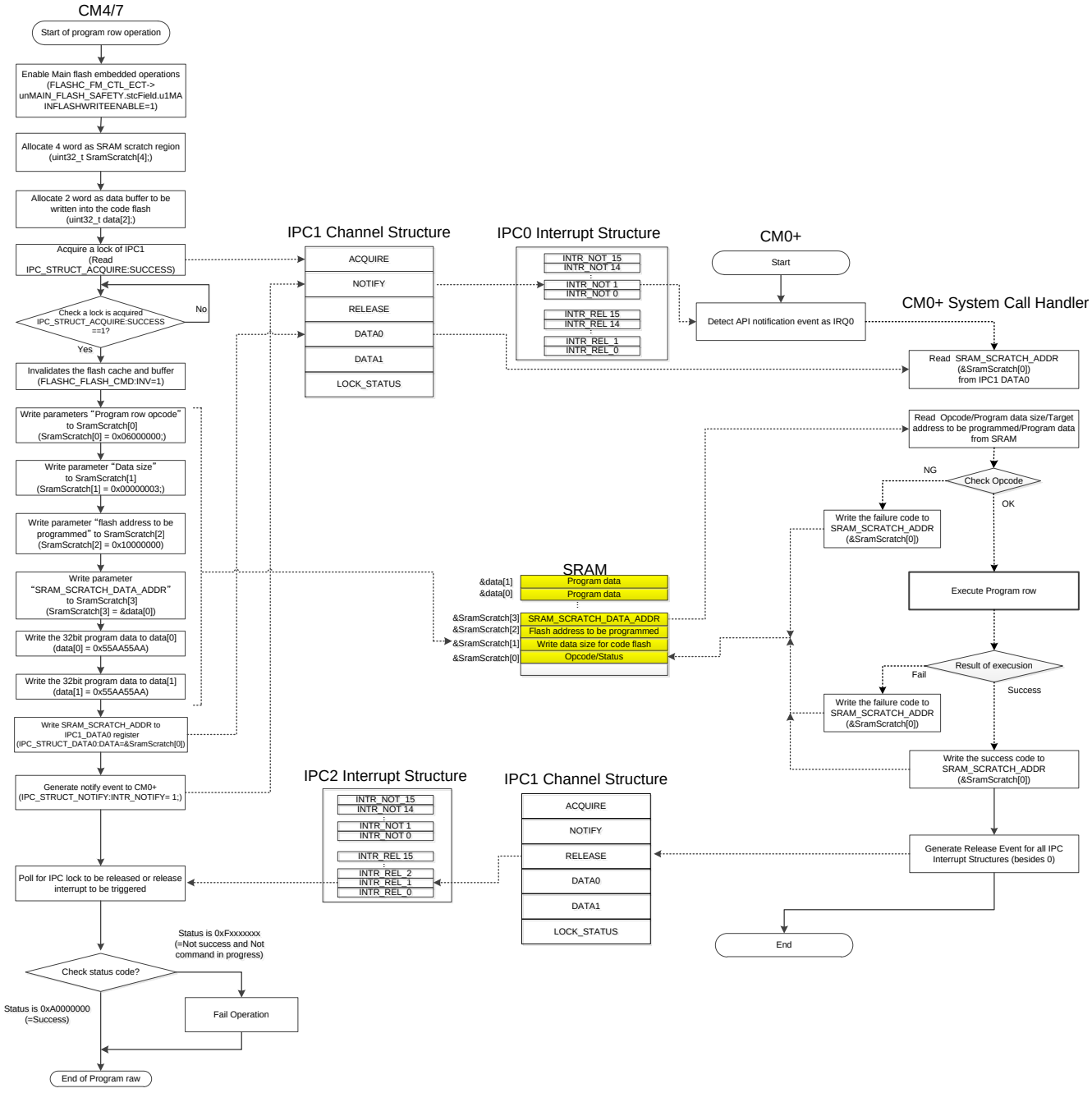
G. If result fails, write the failure code to SRAM_SCRATCH_ADDR (&SramScratch[0]) [By CM0+ IRQ0 handler].

3.3.4 Assessing the Result of "Program Row" Operation from CM0+

A. Detect release interrupt from IPC2 interrupt structure [By CM4/7 user code].

B. If status is OK, Program Row operation is completed [By CM4/7 user code].

Figure 6. Flash Program Row Setting Example



4 Flash Rewriting Using Dual Bank Approach

This chapter shows an example of flash rewriting using flash dual bank approach to enable OTA functionality.

4.1 Concept

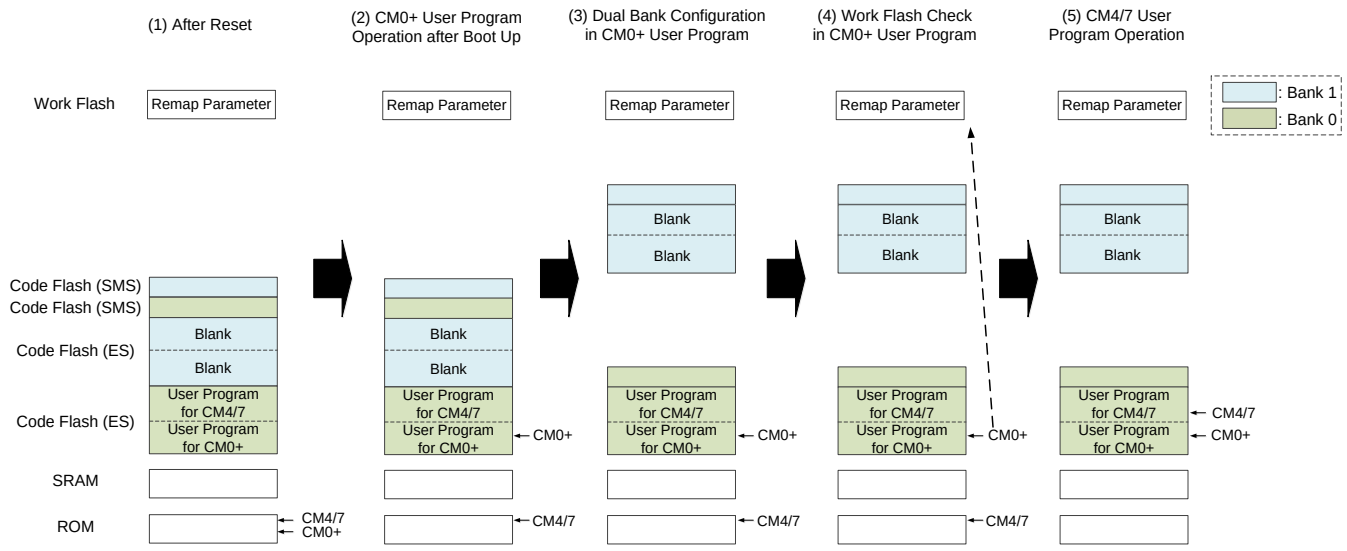
In Traveo II family MCUs, OTA functionality is handled by two separate registers. Flash Main Bank Mode Register is to configure flash bank mode between “Single” and “Dual banks”. The Flash Main Remap register is to configure flash region between “Mapping A and B”. Both Flash Main Bank Mode and Flash Main Remap registers are cleared by reset. ROM and Flash boot will not touch these configurations. In other words, Traveo II family MCU always boots up in single bank mode before CM0+ program starts; then you need to configure dual bank and remap function in your respective program. In this example, remap parameter is stored in work flash and CM0+ program configures the remap after reading from the work flash. Unintended bank switching may cause serious system failure. It is recommended to implement appropriate protection mechanism for system, such as duplicating of switching data or adding check code to prevent unintended bank switching. For safeguard the remap parameter that we store in the work flash, Memory Protection Units (MPU) and Shared Memory Protection units (SMPU) provide the memory protection for work flash. For more detail, see chapter: Protection Unit in Architecture TRM.

4.1.1 Starting First Application

This section explains how flash memory and related functions perform while starting the first application. Figure 7 shows entire process.

1. After reset, CM4/7 and CM0+ start ROM boot. CM4/7 loops by WFI in ROM boot until activation by CM0+.
2. After the completion of ROM boot and Flash boot, CM0+ executes its user program code in the code flash.
3. CM0+ user program code configures the Flash Main Bank Mode to Dual Bank Mode. Flash mapping is split into two halves, and each half is presented as a separate address region.
4. CM0+ user program code checks the remap parameter from certain known work flash area. If the data read from work flash is as expected, CM0+ user program changes the remap control. At the initial state, when no firmware update request is received, no remap control is being executed because the remap parameter will be pointing to the original code.
5. CM0+ use program sets the program counter of CM4/7 user program, then CM4/7 user program in Bank 0 starts.

Figure 7. Starting First Application



4.1.2 Flash Reprogramming

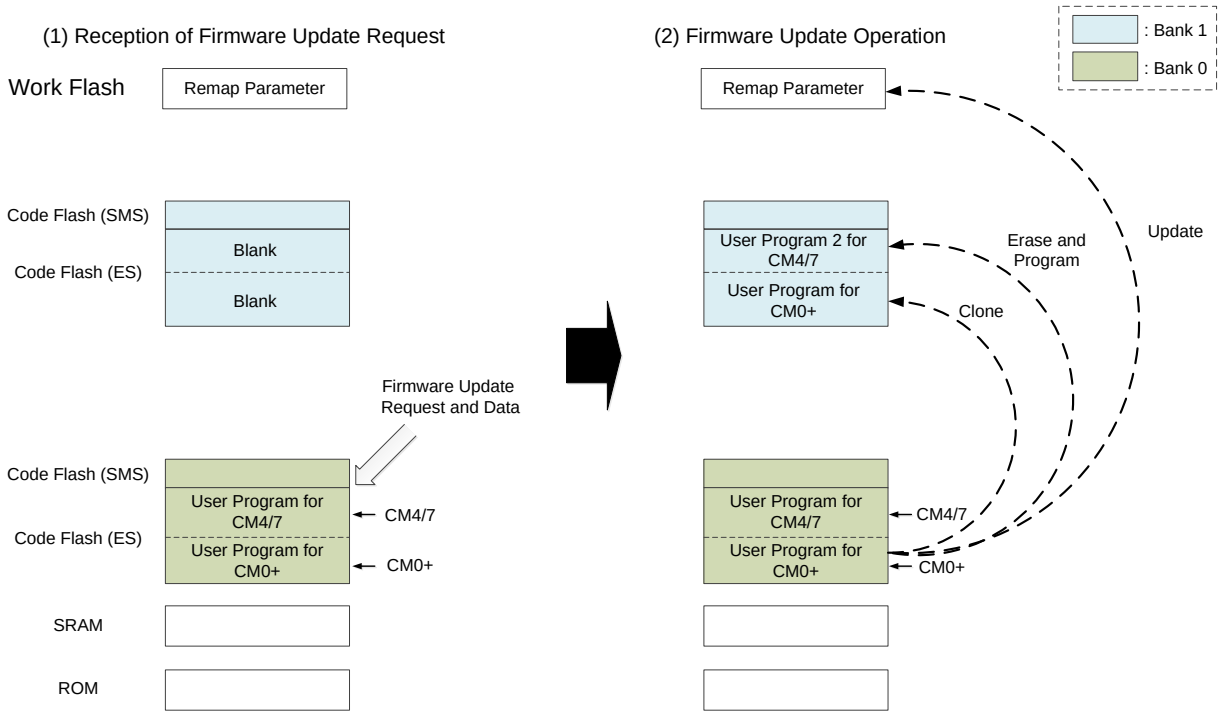
This section explains firmware update using flash dual bank modes. Figure 8 shows the entire process.

1. CM4/7 user program code in Bank 0 receives the firmware update request and data via vehicle LAN such as CAN FD or Ethernet and so on.
2. CM0+ user program starts erasing entire area of code flash Bank 1 and certain work flash area, which contains remap parameter. After the completion of flash erasing, CM0+ user program copies the same CM0+ user program code to code flash Bank 1 and programs the new user program code for CM4/7. Finally, CM0+ updates remap parameter in work flash, which is used to perform switching of the application code at the next reset. Remap parameter would be any key code or start address of user program code and so on depending on the application use case.

   Note: In case of enabling cache on CYT4B series, you should set the MPU as non-cacheable for the following memory.

   - SRAM Scratch area which used by IPC
   - Work Flash area

Figure 8. Flash Reprogramming

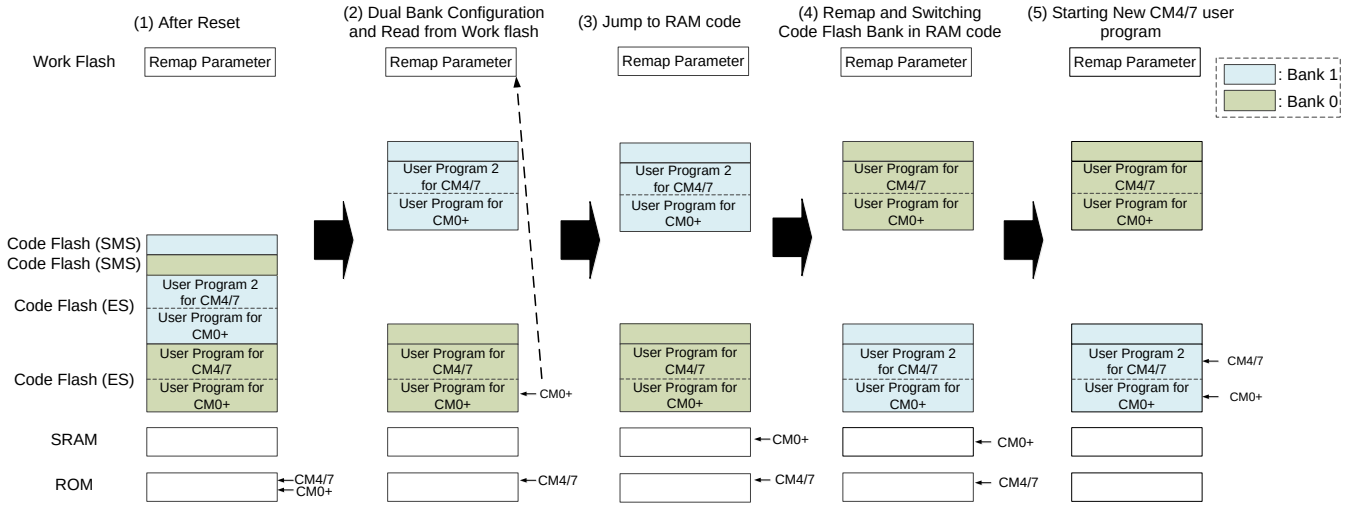


4.1.3 Starting New CM4/7 User Program Code

This section explains the example how new CM4/7 user program code starts after switching the code flash bank. Figure 9 shows entire process.

1. Once new user code is programmed and the remap parameter is updated, user code resets the MCU. After reset, CM4/7 and CM0+ execute the boot code in ROM.
2. CM0+ user program configures the Flash Main Bank Mode to Dual Bank Mode. Flash mapping is split into two halves, and each half is presented as a separate address region. After that, CM0+ user program reads the remap parameter from work flash.
3. If the data read from work flash is an expected value, CM0+ jumps to the SRAM code, then changes the flash region to “Mapping B” by setting the Flash Main Remap register.
4. After the completion of switching operation, address of code flash bank 0 will be swapped with code flash bank 1. CM4/7 and CM0+ can read the flash memory of bank 1 by using the same address.
5. CM0+ jumps back to flash code. At this time, completely same CM0+ user program should be there. CM0+ sets the program counter of CM4/7 to the new user program code, from where its execution is started.

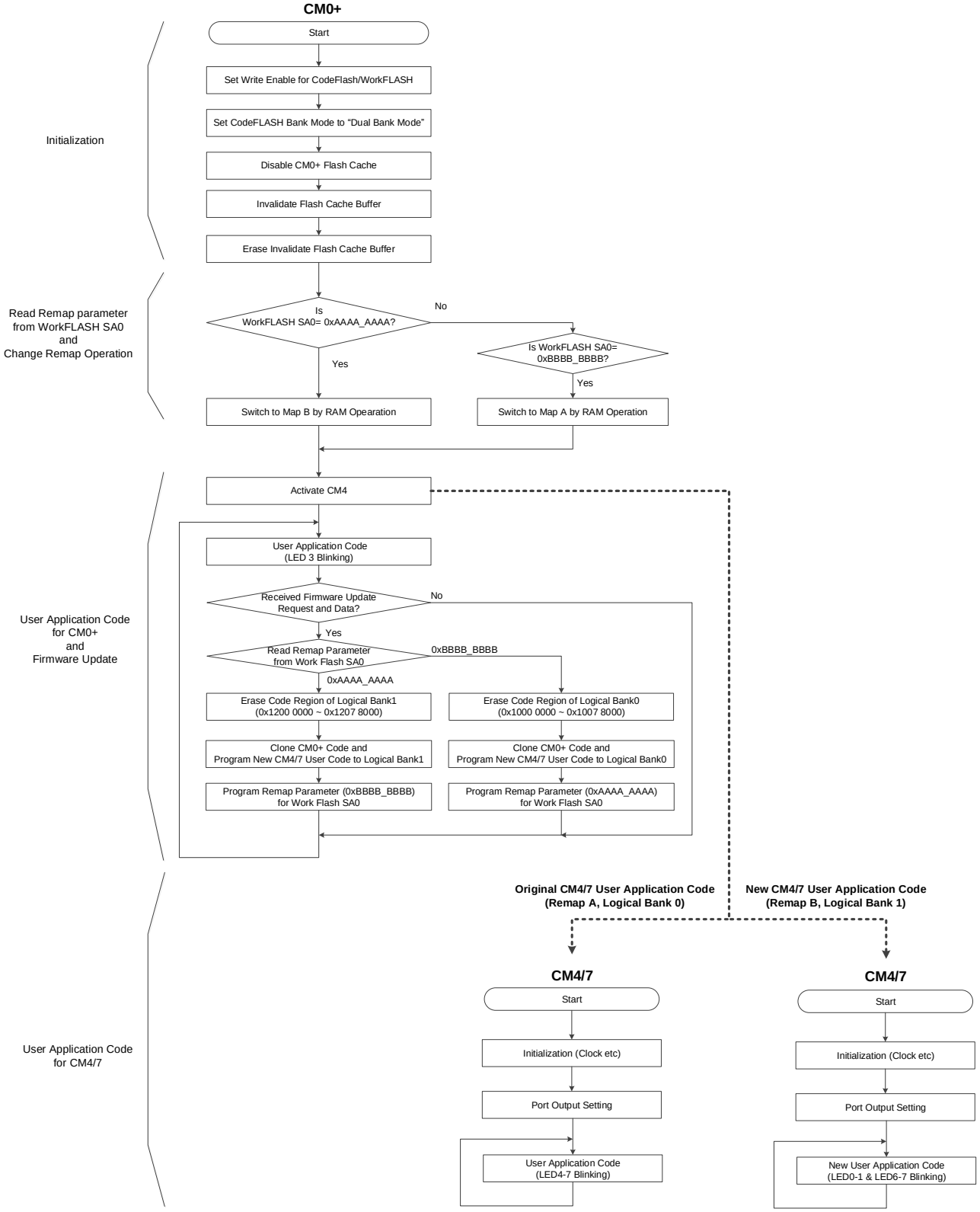
Figure 9. Starting New CM4/7 User Program Code



4.2 Flash Rewriting Procedure using Dual Bank Mode and Remap Function

This section shows the example of flash rewriting procedure using dual bank mode and remap function. In this example, User LED ON/OFF program is updated by the firmware update request. Work flash SA0 sector is used for storing the remap parameters, then CM0+ code reads the data from work flash and configures the mapping. If the data read from work flash is “0xAAAA_AAAA”, mapping A is performed. If the data read is “0xBBBB_BBBB”, mapping B is performed. User application code is swapped and depends on the remap parameter in work flash.

Figure 10. Starting New CM4/7 User Program Code



5 Glossary

Terms	Description
eCT	Embedded Charge-Trap
ECC	Error correcting code
OTA	Over-the-air
RAM	Random access memory
SRAM	Static random-access memory
P-DMA	Peripheral DMA
M-DMA	Memory DMA
IPC	Inter-processer communication
IRQ	Interrupt request
DAP	Debug Access port
RWW	Read-while-write

6 Related Documents

The following are the Traveo II family series datasheets and Technical Reference Manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo II Family
  - CYT3BB/4BB Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo II Family
- Body Controller Entry Family
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- Body Controller High Family
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM) for CYT4BF
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT3BB/4BB
- Cluster 2D Family
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

Document History

Document Title: AN220242 - Flash Rewriting Procedure for Traveo II Family

Document Number: 002-20242

Revision	ECN	Submission Date	Description of Change
**	6083636	03/09/2018	New Application Note.
*A	6398000	06/26/2019	Changed target parts number (CYT2B/CYT4B series) Added Flash Rewriting Procedure using Dual Bank and Remap Function
*B	6740578	03/04/2020	Added a part number CYT2/CYT3/CYT4 series.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709