Application Note

Hardware Programming and Debugger Guide for MSPM0

Henry Nguyen

ABSTRACT

This application note describes the debug subsystem, resets of MSPM0, and the differences in the flashctl between device families. The debug subsystem is an entity that is separate from the M0+ core present in the MSPM0. Through properly utilizing the debug subsystem in the MSPM0, the user can access the M0+ core in a low-power state and recover it in cases of misconfiguration since it is separate from the M0+ core. By acquiring an understanding of the MSPM0's debug subsystem, flashctl, and resets you can achieve an ideal debugging and programming environment with the MSPM0.

Table of Contents

1 Introduction to the Debug Subsystem and MSPM0 2
1.1 Access Ports of MSPM0 2
1.2 Behaviors With the MSPM0 in a Blank/Low-Power State 3
2 Proper SWD Initialization Sequence 3
3 PWR-AP 3
3.1 Enabling Low-Power Mode Debugging With MSPM0 4
3.2 Modifying the Reset Behavior of MSPM0 4
3.3 Register View 5
4 SEC-AP 6
4.1 DSSM Commands 6
4.2 DSSM Flow 7
4.3 Register View 8
5 Understanding Flash in MSPM0 9
5.1 Protection of Flash Memory Across MSPM0 9
5.2 Clearing the STATCMD Register 9
5.3 Ideal Programming Flow for MSPM0 10
6 The Resets of MSPM0 11
7 Summary 11
8 References 11

List of Figures

Figure 1-1. Debug Sub System Block Diagram 2
Figure 2-1. MSPM0 SWD Initialization Sequence 3
Figure 4-1. DSSM Command Flow 7
Figure 5-1. MSPM0 Programming Sequence 10

List of Tables

Table 3-1. DPREC0 Low-Power Mode Configuration Bits 4
Table 3-2. RST CTL Bit Configurations 4
Table 3-3. PWR-AP Register View 5
Table 4-1. DSSM Command Table 6
Table 4-2. SEC-AP Register View 8
Table 5-1. Protection Registers for MSPM0 Flashctl 9
Table 6-1. Sysctl Reset Registers 11

Trademarks

1 Introduction to the Debug Subsystem and MSPM0

The MSPM0 is a low-power MCU that offers a dense feature set at a low cost. To balance the performance and power consumption of the peripherals, they are separated into two separate power domains [1] known as PD0 and PD1. Peripherals within the PD1 domain consist of the CPU, memories, and high performance peripherals and PD0 consist of the low speed, low-power peripherals. Upon entering a low-power mode stronger than SLEEP, PD1 peripherals are disabled to decrease power consumption. This causes the AHB bus to not be discoverable, however, within the MSPM0 contains a peripheral known as the debug subsystem that allows the AHB to be discoverable again. By having the debug subsystem separate from the M0+ core, it provides the debugger or programmer a method of regaining access to the device in scenarios of misconfiguration or low-power state. Accessing the device while in a low-power state or reconfiguring it into a "known state" is done through a set of registers known as the access ports. This application note goes in-depth with the SEC-AP and PWR-AP specifically as they are the vital components for enabling the formerly discussed features. Alongside the debug subsystem, the flashctl and its different protection scheme between devices families, and unique reset via the AIRCR is discussed as well.

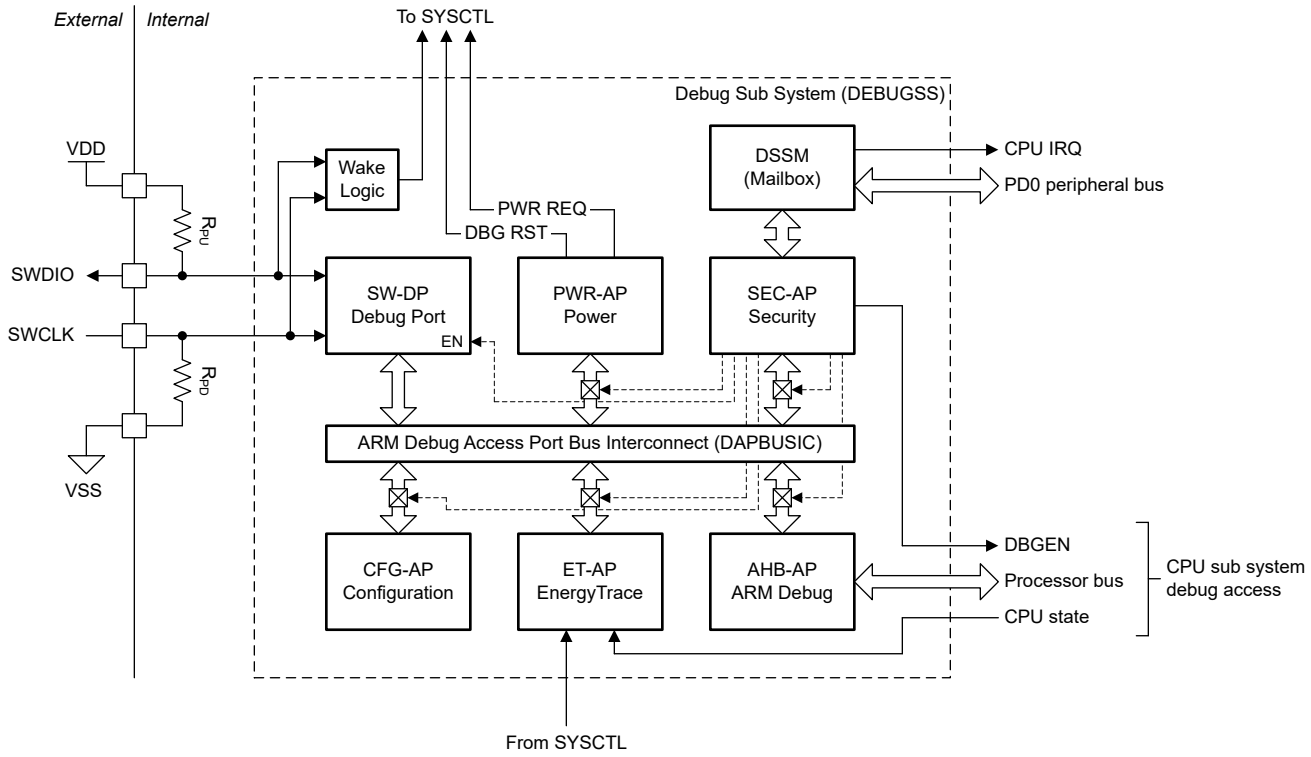

Figure 1-1. Debug Sub System Block Diagram

1.1 Access Ports of MSPM0

MSPM0 devices contain a total of five access ports shown in the Figure 1-1 that the user can directly interface with using the debug access port (DAP). Each access port has their own unique functionality that provides the user with the ability of configuring and reading the device outside of the M0+ core.

1.1.1 Advance High-Performance Bus Access Port

Also known as the AHB-AP, this access port provides the user with a bridge from the DAP to the M0+ core that allows the user to directly interface with the device through direct memory access.

[1] For more information on power domains, see the *MSPM0L110x Mixed-Signal Microcontrollers Data Sheet*.

1.1.2 Configuration Access Port

Also known as the CFG-AP, this access port provides the user with device information and its current operation status.

1.1.3 Security Access Port

Also known as the SEC-AP, this access port provides the user with the ability to communicate to the device bootcode to erase Flash memory and re-enable access to the device.

1.1.4 EnergyTrace™ Access Port

Also known as the ET-AP, this access port provides the user with the ability to read the power state data.

1.1.5 Power Access Port

Also known as the PWR-AP, this access port provides users with a method of configuring the device power state and reset behavior.

1.2 Behaviors With the MSPM0 in a Blank/Low-Power State

A consequence of putting an MSPM0 device into a low-power mode greater than SLEEP, is the AHB-AP becoming undiscoverable. This is due to the AHB being part of PD1. Upon entering DEEPSLEEP (STOP and STANDBY), PD1 is disabled, preventing the AHB-AP from being discoverable. Besides application code putting the device into a low-power state, having empty flash also puts the device into STANDBY0.

When the main memory in any MSPM0 device is empty and powered on for ~5-10 seconds, bootcode comes in and populates SRAM with content and then begins executing from it. The program that populates SRAM puts the device into STANDBY0 and in turn revokes AHB-AP access. To allow the AHB-AP to become visible again, the user must utilize the PWR-AP to power the device. Modifying the PWR-AP, seen in Figure 1-1, provides the user debug access while the CPU stays in a low-power state.

2 Proper SWD Initialization Sequence

MSPM0 uses the Arm® M0+ core, allowing the user to follow the procedure described by Arm to switch the device from JTAG to SWD.

Upon executing the described sequence by Arm to switch from JTAG to SWD, the Wake Logic unit seen in sends a wake-up request to the device allowing the IDCODE to be read regardless of any low-power state the device could be in and for the access ports to be available as well.

It is best practice to have the SWJ-DP state machine in a known state before beginning any operation. Before going in between states, perform a line reset then begin the SWD to JTAG sequence, this is to ensure the lines are in reset and have been initialized to a known state. Then perform the line reset and then JTAG to SWD sequence, upon doing so the Wake Logic unit sends a wake-up signal to the CPU allowing the IDCODE to be read even when the device is in SHUTDOWN mode. To see what should be done when implementing the SWD initialization sequence, see the flow chart in Figure 2-1.

Figure 2-1. MSPM0 SWD Initialization Sequence

3 PWR-AP

The PWR-AP is an access port that contains two registers known as the DPREC0 and SPREC. These registers can be utilized to enable low-power mode debugging, modify reset behavior, and perform resets externally.

3.1 Enabling Low-Power Mode Debugging With MSPM0

MSPM0 contains an access port known as the PWR-AP. It is utilized to re-enable the AHB-AP to regain access to the M0+ core when the device is in a low-power state.

To access the device and maintain a connection while it is in a low-power state, the user must write a 1 to these bits. The register view can be seen in Table 3-3.

Table 3-1. DPREC0 Low-Power Mode Configuration Bits

DPREC0 Bit	Description
FRC ACT (bit 20)	Forces device out of low-power mode
IHIB SLP (bit 3)	Does not allow system to go into low-power mode

Writing to the FRC ACT bit forces the device out of the low-power state allowing the AHB-AP to be discoverable again and writing to IHIB SLP maintains connection to the device even when the CPU has a request to go into DEEPSLEEP mode. It is mandatory that these bits are enabled when adding support for MSPM0. If FRC ACT and IHIB SLP are not written to then access to the M0+ while it is in low-power mode is not possible.

3.2 Modifying the Reset Behavior of MSPM0

With the RST CTL bits of the DPREC0 register, it is possible to modify the behavior seen post-reset. Within Table 3-2 contains all the possible configurations of the RST CTL bits (16:14).

Table 3-2. RST CTL Bit Configurations

RST CTL Mode	Bit Configuration
Wait For Debug	001b
Halt On Reset	100b
Default Reset	000b

3.2.1 Wait for Debug

Wait for debug is the most important configuration to implement when adding support for MSPM0 to any toolchain. Any reset performed upon the device post-configuration will still reset only the peripherals defined by the reset level, but it will remain in the reset-handler post reset. This is to ensure that the device does not go back into the application unless the user wishes to continue with application execution. This configuration is especially useful when flashing a device or debugging an application as it throws the device back into a known state and halts it.

3.2.2 Halt on Reset

Halt on reset will immediately halt the device upon any form of reset performed on it post-configuration. This is also useful in a similar manner to wait for debug but it does not force the device to remain halted at the reset-handler.

3.2.3 INRST Behavior

When the wait for debug sequence is executed, bit 17 within DPREC0 will be set as 1 to indicate it is in wait for debug mode. To be released from the wait for debug state the user must write a 1 to the INRST bit to be released from the wait for debug state.

3.3 Register View

Table 3-3. PWR-AP Register View

APSEL	AP-NAME	ADDR	BANK	INDEX	REG-NAME	BIT 31	BIT 30	BIT 29	BIT 28	BIT 27	BIT 26	BIT 25	BIT 24	BIT 23	BIT 22	BIT 21	BIT 20	BIT 19	BIT 18	BIT 17	BIT 16	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
4	PWR	0x00	0	0	DPREC0	Reserved											IHIBSLP	Reserved		INRST	RST CTL			Reserved										FRCACT	Reserved			
		0xF0	15	0	SPREC	Reserved																															SYSRST
		0xFC	15	3	IDR	Access point ID																															

4 SEC-AP

When a user misconfigures their clock, disable debug access via nonmain [2], or program incorrect values into the CRC registers of nonmain, this disables access to the MSPM0. An understanding of the SEC-AP is vital when adding support for MSPM0 as it is utilized for device recovery in cases of peripheral or nonmain misconfiguration. Previously said recovery is done by sending a Debug Sub-System Mailbox (DSSM) command into the mailbox and then executing it by performing a BOOTRST. It is also possible to unlock debug access in the scenario that it is disabled with or without a password while debug port access is still enabled.

4.1 DSSM Commands

When sending a DSSM command to the mailbox, it is serviced by the bootcode that begins executing upon performing a BOOTRST and only a BOOTRST. If a BOR or POR is executed while a DSSM command is in the mailbox it will be wiped out and not serviced as the reset levels will power cycle the power domains. It is important to note that if a password is enabled for any DSSM command, it will not fully execute the command until the password sequence has been fully executed. Once the command has been sent to the mailbox, the user then has a window of two seconds to send the password that matches the password set within nonmain. If successful only then the command will be fully processed. For all possible DSSM commands [3], see the Table 4-1. For the flow when implementing support for the DSSM commands and the register view, see Table 4-2.

Table 4-1. DSSM Command Table

DSSM Command	DSSM Value
Factory Reset	0x020Ah
Mass Erase	0x020Ch
Password Authentication	0x030Eh
Data Exchange	0x00EEh
Wait For Debug	0x0206h

4.1.1 Factory Reset

When the value for the "Factory Reset" command is sent to the mailbox and processed. All content within the main and nonmain memory will be wiped out, then nonmain will be re-populated with its default content.

This command is useful in the cases of:

- Nonmain misconfiguration that is not too severe.
- Disabled debug access.
- Peripheral/device misconfiguration (for example, overclocked PLL, non-serviced wwdt, or double hard-fault).

4.1.2 Mass Erase

When the value for the "Mass Erase" command is sent to the mailbox and processed. All content within the main memory will be wiped out and nonmain remains untouched

Similar to factory reset this command is useful in the case of:

- Peripheral/device misconfiguration (for example, overclocked PLL, non-serviced wwdt, or double hard-fault).

4.1.3 Password Authentication

When the value for the "Password Authentication" command is sent to the mailbox and processed. Only after the password has been processed, debug access will be unlocked.

4.1.4 Data Exchange

Data exchange is the only DSSM command that does not require a reset to process the command. If the scenario (e.g factory reset, mass erase, or password authentication) requires a password. After sending the command to the mailbox and performing a reset, the user must begin sending the password to the TXDATA register. After each word 0x00EEh must be written to the TXCTL register.

[2] For more information about nonmain, see the *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*.
[3] Mass erase, password authentication, and data exchange cannot be executed by the MSPM0C and MSPS device families.

4.1.5 Wait for Debug

When the command for the "Wait for Debug" command is sent to the mailbox and processed. Similar to wait for debug described previously in Section 3.2, it resets the peripherals defined by the reset level and then forces the device into the reset handler. However, when performing the wait for debug sequence via the mailbox, it requires the INRST bit to be cleared after executing the command to fully execute the command the command.

4.1.6 Custom DSSM Command

It is also possible for the user to create their own DSSM command and have it perform actions defined by the user. This can be done by having the debugger communicate to the M0+ core through the TXDATA and TXCTL. The core can then receive messages from the debugger and send responses back by using the RXDATA and RXCTL registers for the debugger to read. It is also possible to configure CPU interrupt events for activity seen in the TX_DATA buffer, RX_DATA buffer, and DAP connection.

4.2 DSSM Flow

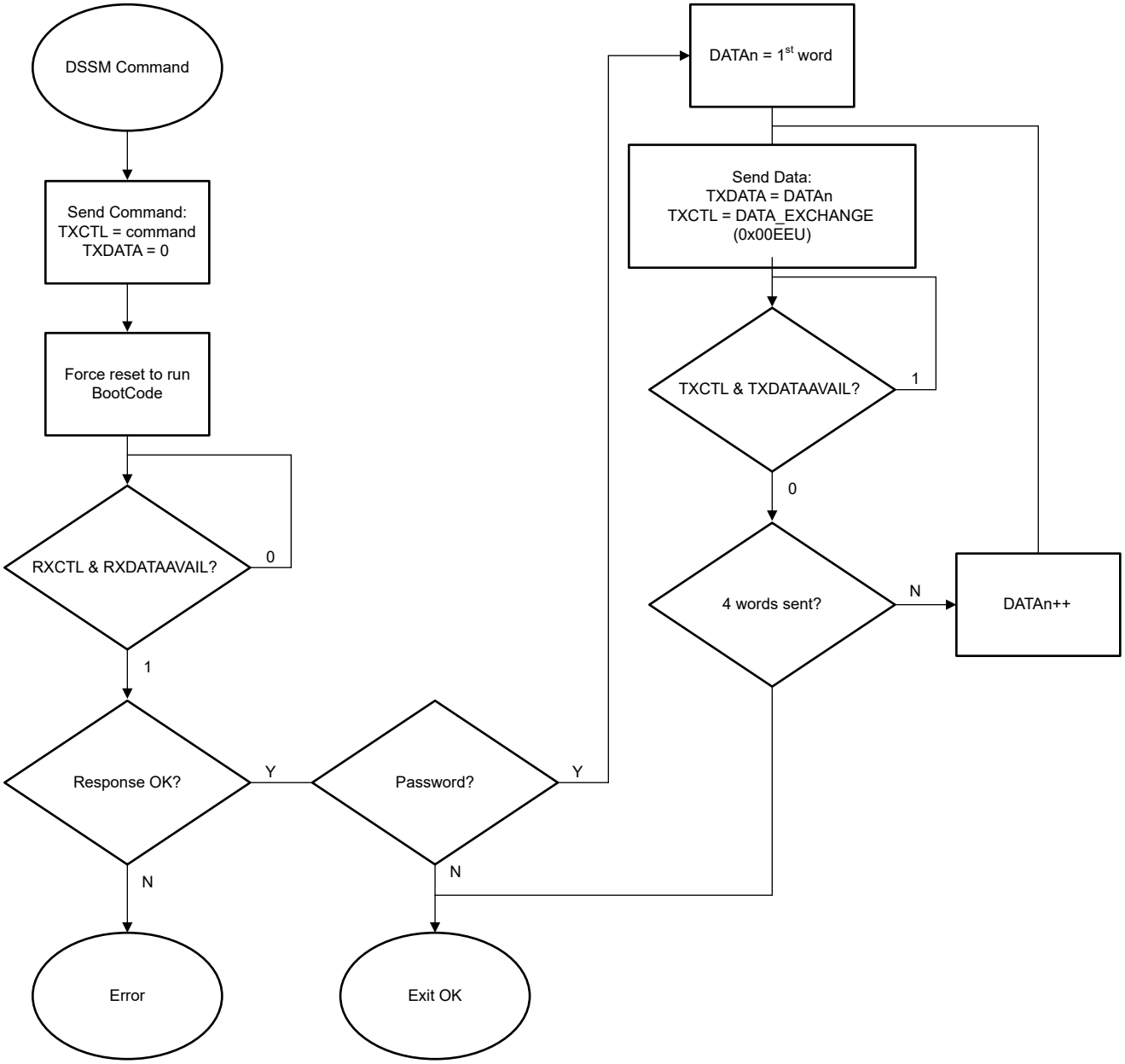

Figure 4-1. DSSM Command Flow

4.3 Register View

Table 4-2. SEC-AP Register View

APSEL	AP-NAME	ADDR	BANK	INDEX	REG-NAME	BIT 31	BIT 30	BIT 29	BIT 28	BIT 27	BIT 26	BIT 25	BIT 24	BIT 23	BIT 22	BIT 21	BIT 20	BIT 19	BIT 18	BIT 17	BIT 16	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
2	SEC	0x00	0	0	TXDATA	TX DATA																															
		0x04	0	1	TXCTL	TX CONTROL																															TXVLD
		0x08	0	2	RXDATA	RX DATA																															
		0x0C	0	3	RXCTL	RESERVED																								RX CONTROL							RXVLD
		0xFC	15	3	IDR	Access point ID																															

5 Understanding Flash in MSPM0

Memory within MSPM0 devices are organized into banks, with each bank of memory containing sectors of memory that are 1kB each in size. With certain variants of MSPM0 they can contain several banks of memory as well. This is important to understand when implementing the programming algorithm for flash of MSPM0 [4], as it is vital that the nuances between devices are well understood. It is also important to isolate any flash operation between different memory spaces such as the nonmain and main memory into their own programming operation. This section of the application note goes into further details on how to handle flash across the MSPM0 family in regular debugging to production environment.

5.1 Protection of Flash Memory Across MSPM0

In all regions of Flash memory in MSPM0, they will have a protection register known as CMDWEPROTx. They are dynamic registers that will unprotect sectors of memory according to which bit the user sets to zero. The memory remains unprotected until the user performs a flash operation. Upon performing any flash operation, the CMDWEPROTx register used automatically re-protects the memory by setting all bits back to one. Table 5-1 shows the protection registers used for all devices.

Table 5-1. Protection Registers for MSPM0 Flashctl

CMDWEPROTx Register	MSPM0L11XX / MSPM0L13XX	MSPM0G1X0X / MSPM0G3X0X	MSPM0C110X / MSPS003FX	MSPM0L122X / MSPM0L222X	All Future MSPM0 SOCs
CMDWEPROTA	x	x	x	x	
CMDWEPROTB	x	x		x	x
CMDWEPROTNM	x	x	x	x	x

5.2 Clearing the STATCMD Register

As the MSPM0 device family expands with newer SOCs, the best practice when adding support for flash evolves as well. With all variants of the MSPM0, it is best practice to clear the STATCMD register before any flash operation is performed. The register is cleared by writing the clear status command (0x00000005h) to the CMDTYPE register and then executing it. Upon execution of the command, any previous content in the STATCMD register will be cleared. It is important to note that clearing the STATCMD register will also reset the CMDWEPROTx registers previously discussed.

An example of clearing the STATCMD register when attempting to perform a sector erase can be seen in the following steps:

1. Write the clear status (0x00000005h) command to the CMDTYPE register.
2. Execute the command by writing the execute key (0x00000001h) to the CMDEXEC register.
3. Poll for completion by checking the STATCMD register.
   a. Upon completion STATCMD will be empty.
4. Unprotect the sector of memory by setting the desired bit within CMDWEPROTx to zero.
5. Set the CMDTYPE register perform an erase that is a sector in size (0x00000042h).
6. Set the CMDADDR register equal to the desired address of operation.
7. Execute the command by writing the execute key (0x00000001h) to the CMDEXEC register.
8. Poll for completion by checking the STATCMD register.

[4] For a broader understanding of the flashctl and SDK for example code, see an MSPM0.

5.3 Ideal Programming Flow for MSPM0

When creating the flashloader for any MSPM0 device, it is always vital that the flash algorithm performs modifications of each memory region separately. For example, if the user's application contains content for both the main and nonmain memory. Then the tool must take care of erasing and programming each memory region separately. This is to limit the exposure of the nonmain region as much as possible from other factors such as noise. Due to all the security configuration living within the region of nonmain memory, these cautions must be taken when attempting to modify it. For the ideal flow when programming, see Figure 5-1.

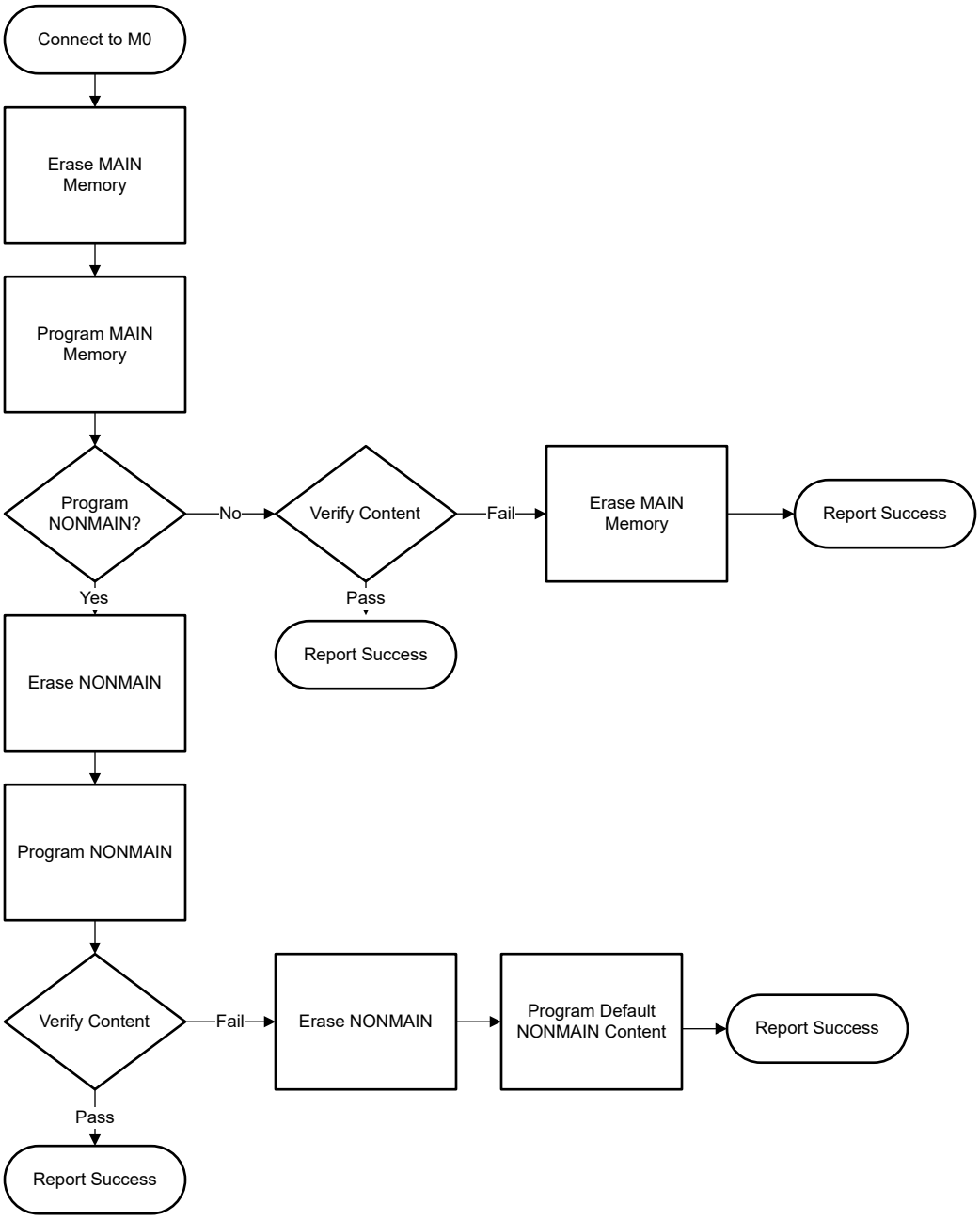

Figure 5-1. MSPM0 Programming Sequence

6 The Resets of MSPM0

Understanding the unique implementation of resets within MSPM0 is important when adding support. When performing a reset via the AIRCR it performs a reset only on the CPU and no other peripherals. To perform a system reset, it is best to utilize the SYSCTL module to execute the reset. This is done by using the Table 6-1 and following these steps:

1. Write the system level reset (0x00000000h) to the Reset Level register. This is the value for system reset.
2. Write the reset key (0xE4000001h) to the Reset Command register to execute the reset.

Adding support for is to ensure that the device is able to exit the bootcode after being empty for any period of time. It is also possible to perform a system reset externally by writing to the SYS RST bit of the SPREC inside the PWR-AP. The system reset performed by the SPREC is lower than a system reset via SYSCTL but higher than a CPU reset.

Table 6-1. Sysctl Reset Registers

Register	Address
Reset Level	0x400B0300h
Reset Command	0x400B0304h

7 Summary

This application note provides an introduction to the debug subsystem, flashctl, and the unique resets of MSPM0. Through the understanding provided by the document any third-party can easily pick up MSPM0 and implement a robust solution to their toolchain. This ensures the customer experience with the tool is as ideal as possible.

8 References

- Texas Instruments: *MSPM0L110x Mixed-Signal Microcontrollers Data Sheet*
- Texas Instruments: *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265