Freescale Semiconductor
Application Note

Document Number:AN4533
Rev. 0, 06/2012

Robust Over-the-Air Firmware Updates Using Program Flash Memory Swap on Kinetis Microcontrollers

**by: Maclain Lobdell
Automotive, Industrial, & Multi-Market Solutions Group**

Contents

1 Introduction

The program flash swap feature available on Kinetis microcontrollers provides a robust means for remote over-the-air system software upgrades. This application note describes how to take advantage of the swap feature. Common questions are addressed and an application example is provided.

This application note is targeted toward system programmers developing applications that can utilize flash swap. Methods for over-the-air communications are mentioned, but are not the focus of this application note.

For additional important information on flash swap, review the reference manual for the respective devices discussed in this application note.

2 Application requirements

A reliable over-the-air system update process is required for many applications. A mechanism is needed to update system software/firmware remotely to fix bugs and make important software improvements. Critical applications, such as electrical metering, require extremely minimal system downtime during updates, must be tolerant of communication errors, and should have no risk of problems causing the system to cease to operate. The update process must be tolerant of difficult environments. In many applications with unreliable electrical power, the system update process must provide consistent operation in the presence of power loss.

Updating system software remotely requires new code to be transmitted through a communication interface (for example, Wi-Fi, Zigbee®, UART, SPI, I2C, USB, Ethernet, and so on). A software uploader program executing on the system must receive and program the code to local memory. Typically, in-factory programming is accomplished with a local debug connection like JTAG or Background Debug Mode (BDM), however these connections are not possible remotely. It is economically infeasible to send a technician out to a system in the field for reprogramming. Therefore, a reliable over-the-air update process is mandatory.

2.1 Remote firmware update systems

2.1.1 Traditional remote update systems

Traditional remote firmware updating systems rely on a bootloader application that executes on a system reset and selects the application to run or executes an application update routine. A remote server sends new firmware to the system to program into its local memory.

In most cases, when the system update process is started, the main application is halted. The main application code is then erased and reprogrammed. There is only one copy of the main application, therefore if there were undetected errors when the new code was received and programmed, the system may not operate correctly and could stop operating until a new application is downloaded. The worst case scenario is when the system is unresponsive and unable to be forced into bootloader mode for system updates.

Advantages

- Simple to implement
- Less flash memory is required because no backup copy of application is maintained

Disadvantages

- The main application must stop during the update process
- Not possible to go back to known working application
- May not be tolerant of power loss during the update process

2.1.2 Systems with code back-up

A system can be implemented with a complete software backup held in memory. If a serious error is detected, the system can revert to the backup copy of the main application. A bootloader could be used to select the correct copy of the main application to execute.

Advantages

- In a multitasking operating system it is possible to continue to execute the main application tasks while background tasks are running to update the new copy of the application.
- Backup copy of code. Possible to revert back to the known working application.

Disadvantages

- Additional memory space required to store backup copy.
- Requires a bootloader (startup routine that selects which application to run).

2.1.3 Systems using flash memory swap

In devices with two or more internal flash blocks that support swap, the memory base address of each flash block can be exchanged. The address location of each flash block will thus be swapped in the logical memory map of the device. After a reset, the built-in flash swap system essentially selects which software executes by the location of the flash block in the logical memory map. This allows a code back-up system with the added ease of programming. You can execute out of one block while erasing/programming the other block. On Kinetis devices, the flash swap system monitors/controls all the steps of switching from the old application to new; there is an added assurance of reliable operation in the case of a power loss during one of those steps.

Advantages

- Ease of programming. Application always executes out of the lower block in the memory map.
- Power loss tolerant.
- No bootloader required. No delay to the start of the main application.
- Well suited for a multi-tasking OS. Minimal application downtime. In a multi-tasking system it is possible to continue to execute the main application tasks while background tasks are running to update the new copy of the application.
- Backup copy of code. Possible to revert to the known working application.

Disadvantages

- Additional flash memory space required to store backup copy.

3 Program flash swap overview

Kinetis devices are available with internal flash memory for program and data storage. For devices with two or more program flash (p-flash) memory blocks that have the p-flash swap feature, the system programmer can configure the logical memory map of the p-flash space such that either of the two physical p-flash blocks can exist at relative address 0x0000. The flash swap feature is not mandatory.

This is particularly useful because on Kinetis devices the default vector table is located at address 0x0000. When the processor exits reset, it fetches the initial stack pointer (SP) from address 0x0000 and the program counter (PC) from address 0x0004. Thus, swapping the base address of the two p-flash blocks allows the system to either boot from p-flash block 0 or p-flash block 1 because either block can be located at base address 0x0000.

The swap system is controlled by issuing swap control commands that are similar to other flash commands on Kinetis devices. The swap system requires initialization before swap is possible.

3.1 Key facts about p-flash swap

- The block that is currently located at relative address 0x0000 is known as the active block. This is also referred to as the lower block. The other block is known as the nonactive block or upper block.
- The typical use-case is for the system to execute from one block (active block) while reprogramming the other block (nonactive block). In most applications, the system application code is limited to fit within one flash block (active block).
- After reprogramming the nonactive block and when it is ready for execution, the user can complete the swap process to swap the blocks. The nonactive block becomes the active block and vice versa.
- A system reset begins code execution from the active block (the block located at base address 0x0000).
- The same procedure for swapping back and forth between blocks is used.

3.1.1 Kinetis (100 MHz) swap details

Kinetis 100 MHz devices with two p-flash blocks support swapping of the two blocks. The SWAP bit of the FTFL_FCNFG register indicates which p-flash block is located at address 0x0000. The state of the SWAP flag is set by the swap system within the flash module during the reset sequence.

The SWAPPFLSH bit in the SIM_FCFG2 register indicates if the swap system has been initialized and thus potentially ready to be swapped.

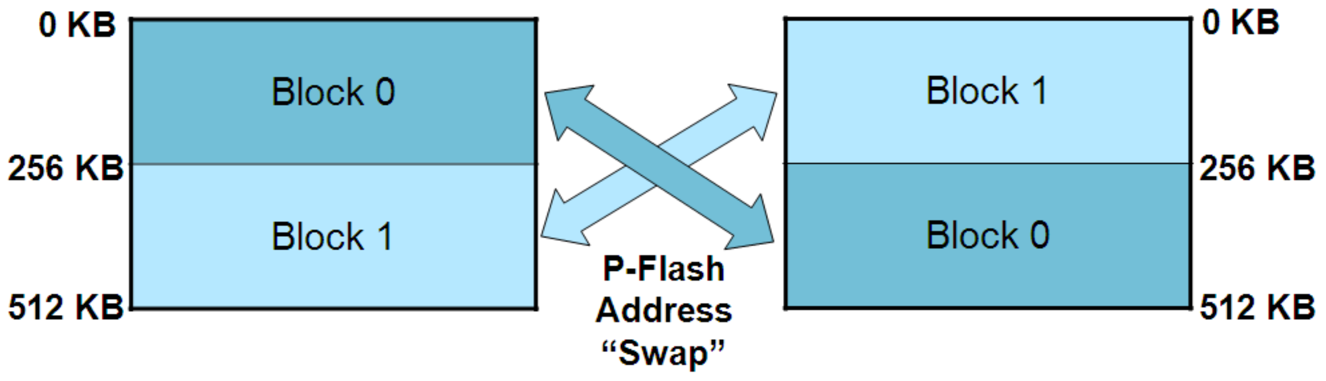


Figure 1. P-flash swap (Kinetis 100 MHz)

3.1.2 Kinetis (120/150 MHz) swap details

Kinetis 120/150 MHz devices have two flash configuration options that support swap. Each are discussed below.

The SWAP bit of the FTFL_FCNFG register indicates which p-flash block/half is located at address 0x0000. The state of the SWAP flag is set by the swap system within the flash module during the reset sequence.

3.1.2.1 512 KB p-flash option

Kinetis 120/150 MHz devices with two p-flash blocks (512 KB p-flash) support swapping between the two p-flash blocks. This is similar to the 100 MHz Kinetis devices with 512 KB.

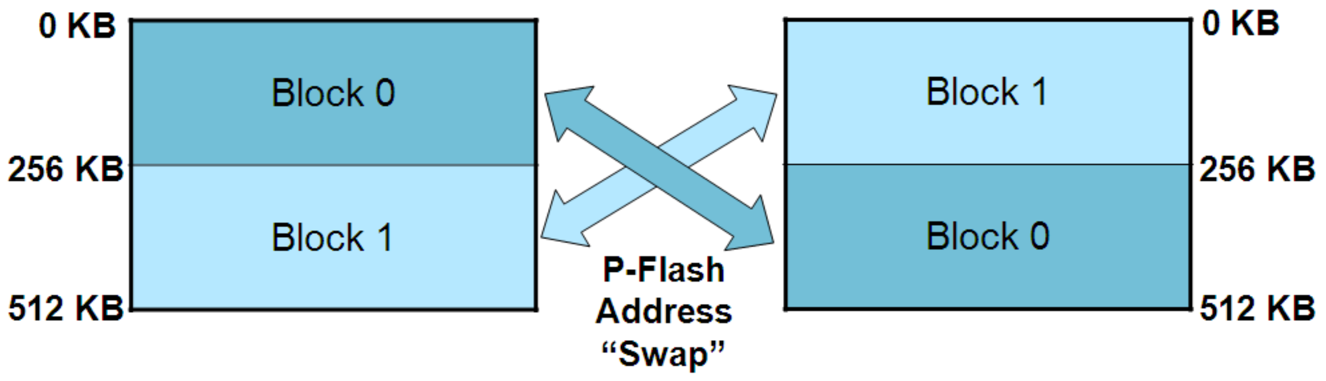


Figure 2. P-flash swap (Kinetis 120/150 MHz, 512K p-flash)

3.1.2.2 1 MB p-flash option

Kinetis 120/150 MHz devices with four p-flash blocks (1 MB) support swapping between two 2-block halves (p-flash blocks 0–1 and 2–3).

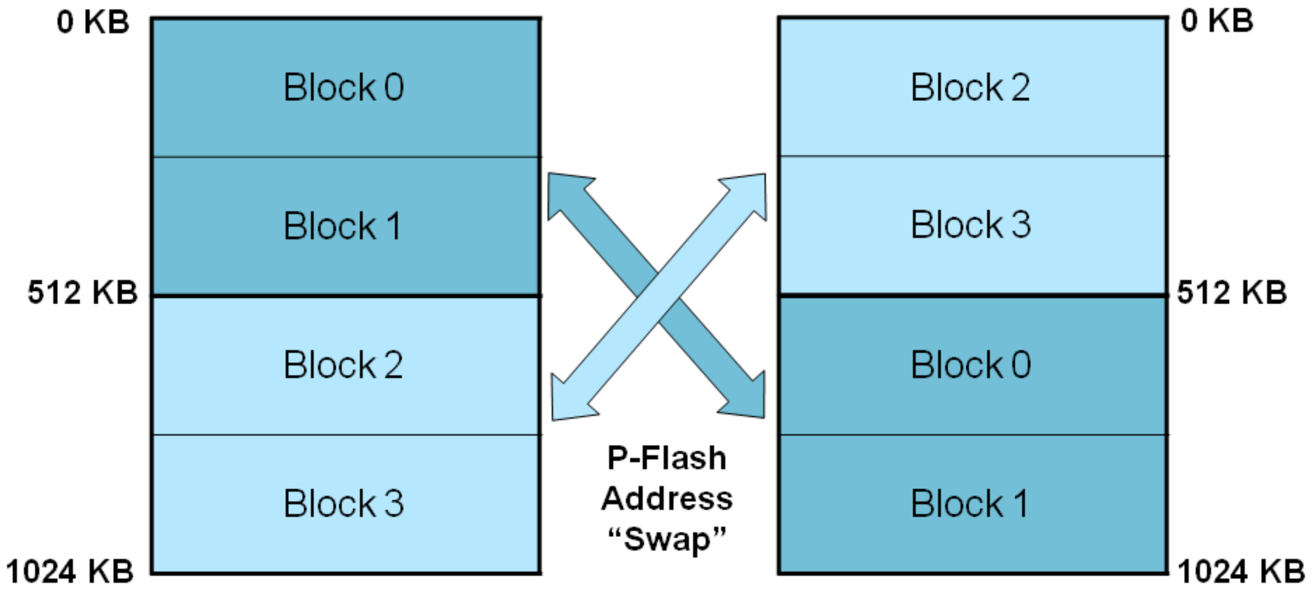


Figure 3. P-flash swap (Kinetis 120/150 MHz, 1M p-flash)

3.2 Swap command overview

There are four options to the swap command.

1. Set Swap Indicator Address (Initialize Swap system)
2. Set Swap in *Update* (Prepare) Mode
3. Set Swap in *Complete* Mode
4. Report Swap Status

Set Swap Indicator Address Command

This command initializes the swap system. When issuing this command, you must provide an address for the flash swap indicators (discussed below). This command sets the swap enable word and swap indicator address which is used by the swap system. This step is required only once when for the first time a swap is performed.

Set Swap in *Update* Mode

This command programs the flash swap indicator value to indicate to the system that content updates of the upper block has been planned. Additionally, it un-protects the sector holding the flash swap indicator within the nonactive block to allow it to be erased. Erasing this sector during the Update (or Update-Erased) state is a requirement of the swap process.

Set Swap in *Complete* Mode

This command programs the flash swap indicator value to indicate to the system that reprogramming the nonactive block is complete and is ready to swap the flash blocks. This command is to be issued after the contents of the nonactive (upper) block are erased/reprogrammed as required to update the system software. This includes erasing the sector in the nonactive block that contains the flash swap indicator.

Report Swap Status

This command checks the status of the swap system. It returns the current swap state, current swap block status (which block is currently located at address 0x0000), the next swap block status (which block will be located at 0x0000 after a system reset), and if there are any errors.

It is recommended to issue this command in the system initialization code that executes on a system reset. This will help the system software identify if there were any errors with the swap system that could potentially be caused by power loss during a swap command execution.

See Error handling for more details.

3.2.1 Execute Swap command from internal SRAM

The Swap command must be executed from an SRAM routine to avoid a read-while-write violation. A read-while-write violation is possible if code is executing out of a flash block in which an erase or program operation is taking place. Since the flash swap system programs the flash swap indicators located in both flash blocks, it is necessary that the code is not executing out of the flash when the swap commands are issued.

See example software that accompanies this application note for details.

3.2.2 Address passed to all swap commands

The address required by all the swap commands is the active block swap indicator address that was set by the initialization command. The same address must be used for each swap command. It does not matter whether you are swapping from block 0 to 1 or block 1 to 0, and so on.

See Flash swap indicators for more details.

4 Swap steps

4.1 Swap procedure

The procedure for swapping the flash blocks is simple. The same procedure for swapping back and forth between the blocks is used.

4.1.1 First swap

1. First initialize the system by issuing the initialization command. This step is required only once when for the first time a swap is performed. When first initialized, the swap system goes straight to *Update-Erased*.
2. Erase the nonactive (upper) block.
3. Reprogram the nonactive (upper) block with new software.
4. Issue the command to set the system to the *Complete* state. The swap will take effect after a reset (including a software reset).
5. After a reset, the blocks are swapped and the swap system comes up in the *Ready* state.

4.1.2 Procedure for additional swaps

After the first swap has been completed, the process starts with the *Ready* state.

1. Issue the command to set the system to the *Update* state.
2. Erase the nonactive (upper) block. Once the erase is complete, the system will automatically move to the *Update-Erased* state.
3. Reprogram the software in the nonactive block.
4. Issue the command to set the system to the *Complete* state.
5. Reset the microcontroller (any reset including software reset).
6. After a reset, the blocks are swapped and the swap system comes up in the *Ready* state.

4.2 Erasing the nonactive upper block

You can use block erase or sector erases. It is only specifically required to erase the flash swap indicator sector in the nonactive block. However, to update the software in the nonactive block, you must first erase it. Once the erase is complete, the system will automatically move to the *Update-Erased* state.

Swapping between blocks without erasing/reprogramming the entire nonactive block is possible. You just need to erase the sector with the flash swap indicator (when in the *Update* state). This is useful for swapping back to a known good application in the nonactive block.

See Flash swap indicators for more details.

4.3 Order of the steps

Freescale recommends that the new code be uploaded to the nonactive (upper) block when the swap system is in the *Update-Erased* state and before the system is moved to the *Complete* state. Thus, if there is a power loss during the process, the swap system will know that it was in the middle of updating and it should revert to the previous swap state that was known to be good.

The order that the flash swap command options are issued is important. The command options cannot be issued out of sequence or an error can occur. See the section Error handling for details.

4.4 Summary of typical procedure to swap (after the first swap)

1. Check Status
2. *Ready -> Update*
3. Erase upper block (or just sector with flash swap indicator in the nonactive upper block)
4. *Update -> Update-Erased* (automatic)
5. Reprogram upper block – check contents of upper block (skip over swap indicator location when programming)
6. Swap *Update-Erased -> Complete*
7. Reset
8. Complete -> Ready (automatic)

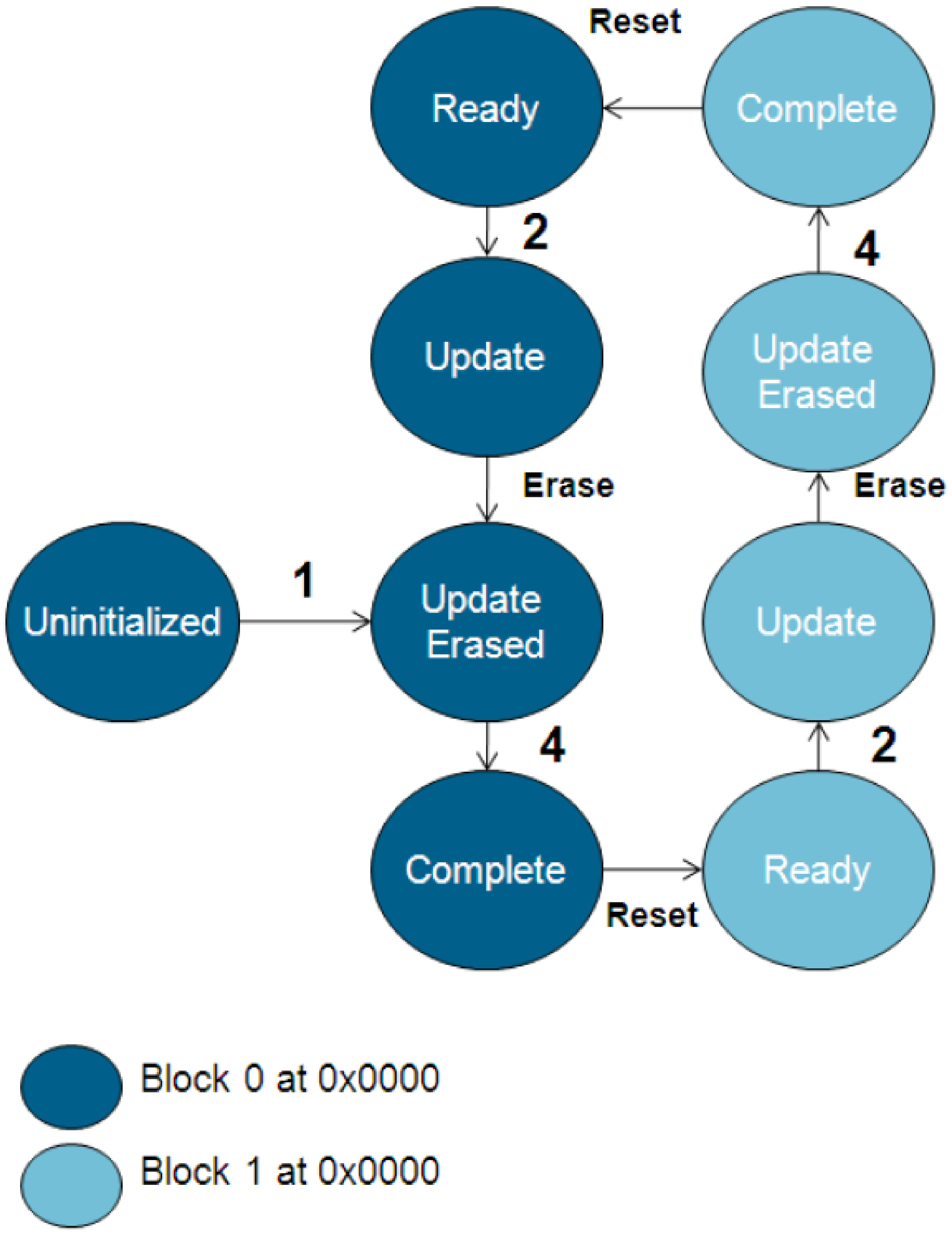


Figure 4. Flash Swap (Kinetis 120/150 MHz, 1M p-flash)

5 Swap details

5.1 Flash swap indicators

The flash swap indicators (indicator0 and indicator1) are two locations within the p-flash memory space. One is within the lower half and the other in the upper half of the total p-flash. The two flash swap indicators are located at the same address offset relative to the base of each p-flash half. The system programmer specifies where the flash swap indicators are to be located when initializing the flash swap system.

What is the purpose of the flash swap indicators?

The flash swap indicators are used by the swap system to store codes that the system uses to ensure robust operation in the presence of potential power losses during swap operations. The swap system programs the flash swap indicators with specific codes at each step of the swap process. This provides the system with a tracking mechanism so that it can reliably determine the swap status on a system reset.

On a reset, the swap system interrogates the swap indicators to determine the swap state and whether there was any interruption during a swap flow. You have to do nothing with the swap indicators other than erase them during the *Update* stage of the swap process.

How big are the flash swap indicators?

2 bytes within each p-flash block

What should be programmed to the flash swap indicators?

The system programmer does not need to program the flash swap indicators or scrutinize their contents because the flash swap system handles this automatically. However, the swap state report mapping table in the reference manual provides details of the codes programmed into the swap indicators. The table describes codes during each step in the swap process or in cases where an error was detected (possible power loss during a swap).

Where to locate the flash swap indicators?

The system programmer specifies where the flash swap indicators are to be located when initializing the flash swap system. Any flash sector can be used except the first two sectors which contain the vectors and flash configuration fields. Depending on the device, the swap indicator must be 32-bit, 64-bit, or 128-bit aligned.

Consider for example a device that contains two p-flash blocks of 256 KB (block 0: 0x00000-0x3FFFF, block 1: 0x40000-0x7FFFF) and sector size of 2 KB (0x800). If the swap indicators are placed in the last sector of the flash blocks, the flash swap system will designate 0x3F800 as the swap indicator location for block 0 and 0x7F800 as the swap indicator location for block 1. In this example, the address provided to the swap system is simply 0x3F800.

NOTE

It is recommended to place the flash swap indicators in the last sector of the p-flash block. This will allow the largest contiguous flash area for the application code.

Where is the flash swap indicator address (offset) stored?

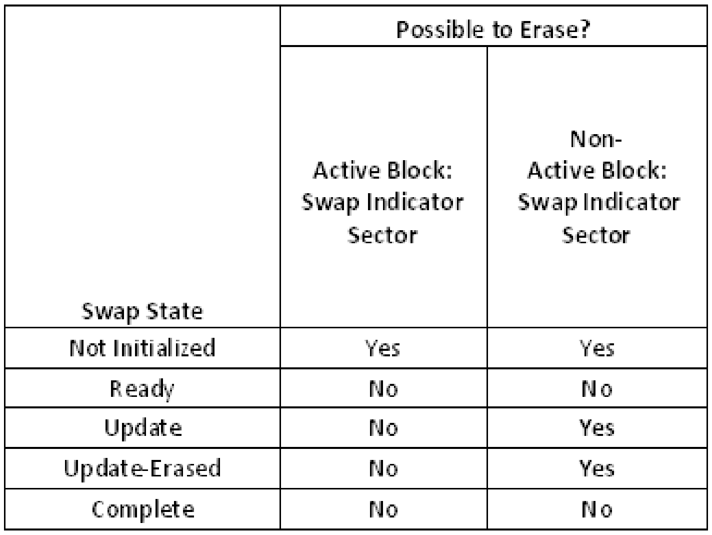
The address of the flash swap indicators (offset address from the base of each block) is stored in an area of flash called the program flash block 1 IFR.

Can the flash swap indicator location be changed?

The flash swap indicator address cannot be modified unless the Erase All Blocks command is issued that clears the swap system back to uninitialized (on supported devices). Therefore, it is important not to forget the address provided to the swap system during swap initialization.

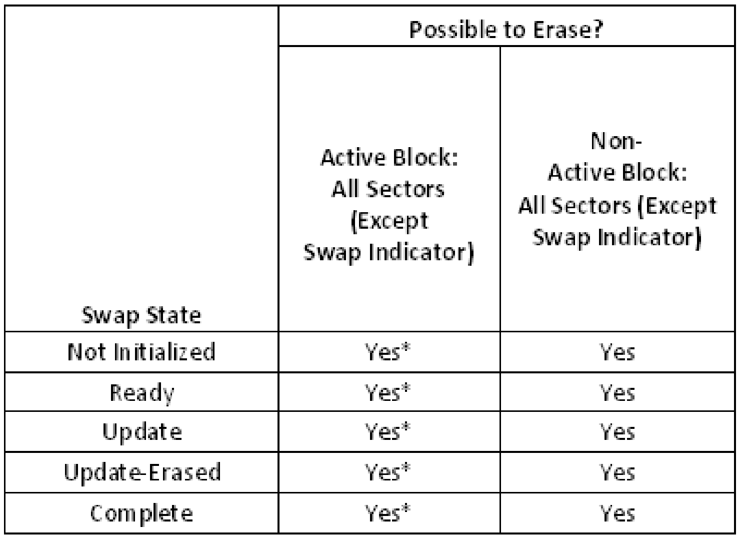
Flash Swap Indicator Protection

Once set, the flash swap indicator locations cannot be programmed by the user. Additionally, the sector containing the flash swap indicator in the active block cannot be erased. The sector containing the flash swap indicator in the nonactive block can only be erased when the swap system is in the *Update* or *Update-Erased* state. Be sure not to program over the swap indicators as this causes an error.



	Possible to Erase?	
Swap State	Active Block: Swap Indicator Sector	Non-Active Block: Swap Indicator Sector
Not Initialized	Yes	Yes
Ready	No	No
Update	No	Yes
Update-Erased	No	Yes
Complete	No	No

Figure 5. When it is allowable to erase the flash swap indicator sector



	Possible to Erase?	
Swap State	Active Block: All Sectors (Except Swap Indicator)	Non-Active Block: All Sectors (Except Swap Indicator)
Not Initialized	Yes*	Yes
Ready	Yes*	Yes
Update	Yes*	Yes
Update-Erased	Yes*	Yes
Complete	Yes*	Yes

Figure 6. When it is allowable to erase sectors other than the flash swap indicator sectors

* You can erase any sector other than the swap indicator sectors at any time. However, you must not erase within the block that you are currently executing. In rare cases in which you want to erase the active block, jump to a subroutine in RAM and execute from RAM.

5.2 Swap enable field (word)

A swap enable field also known as swap enable word is also stored in the Program Flash Block 1 IFR. This location is programmed by the swap system when the swap initialization command is issued. The contents of this location simply tell the swap system that swap has been initialized and that a valid flash swap indicator address has been specified.

5.3 Common questions

Why is swap so complicated? - Power Loss Tolerance

Reading the swap documentation may lead you to the question, “why is swap so complicated?” The answer is power loss tolerance. In less robust systems, if the power is lost at a vulnerable time, the system could become corrupted. For systems using the flash memory swap feature, if a power loss occurs during the swap process, the built-in system detects errors and reliably determines the last known valid flash block configuration.

Are systems using swap limited to applications that fit within one of the two flash blocks (half the total flash memory size)?

In most use cases, the system application will be limited to executing within one flash block (active block). However, the swap system does not restrict code from executing in the nonactive block. Therefore, code can execute from either block at any time.

It is possible to use advanced techniques to expand the size of the application over the size of half the p-flash and still use flash swap. This would sacrifice some protection against problems, but gain more room for code. This is outside the scope of this application note.

Can the sector with the flash swap indicator be used for data/program storage?

Typically, the user specifies an unused flash sector to place the flash swap indicators. However, the swap system will not disturb the other data in the flash sector containing the indicators. So, that space can be used for program or data storage. The space should be designated in the linker file accordingly. See Important considerations for details.

5.4 Important considerations

Always program the vector table and flash configuration fields

You must program the interrupt vector table and flash configuration fields (FSEC, FOPT, and so on) to the nonactive upper block before swapping. If not, the vector table and flash configuration will not be set correctly in the new active block after the swap. This will result in unwanted configurations and could potentially secure the device.

Project linker file considerations

No special considerations are required in the code. Just link your code as it would normally fill the lower block. It is recommended to allocate a section in the linker file for the active (lower) flash swap indicator location to avoid the application attempting to overwrite it.

5.5 Error handling

There are two main situations you should check for errors.

5.5.1 Issue Swap Status command option after a reset

Run the Swap Status command option to report the swap status after any reset to detect the following potential conditions.

1. **Possible power loss during firmware update / swap process: Swap state not Ready**

   A reset occurred in the middle of the firmware update process. This is evident if the system is in the Update or Update-Erased state after a reset. Typically, the system will be in the Ready state unless in process of updating the firmware.

   If this happens, continue the swap process from the current swap state. It is recommended to erase and reprogram firmware in the nonactive block to ensure that it is programmed fully.

2. **Possible power loss during swap command: Error flag set**

   A reset occurred in the middle of the firmware update process that resulted in corrupted flash swap indicator values. This condition is reported by the MGSTAT0 error flag in the FSTAT register after running the Swap Status command option. The swap system has been designed to be fully tolerant of this condition. However, the flash swap indicators need to be cleaned up. This is accomplished by running through the swap procedure once.

5.5.2 Check error flags after issuing a swap command

Check error flags ACCERR and MGSTAT0 in the FSTAT register for errors documented in the Swap Control Command Error Handling table found in the reference manual for the device. It is recommended to check for errors after any command, not just swap.

6 Multitasking application overview

In a multitasking operating system, the main tasks can continue running while new firmware is being loaded by a separate uploader/programmer task. The software uploader task sets the swap system to *Update* mode, receives new application code (from a communication interface), programs it to the non-active upper program flash block, then sets the swap system to complete the swap. After the system is reset, the flash blocks are swapped and the new application boots up. Thus the application downtime is very minimal. The process can be repeated again as long as the new application also has the uploader/programmer/swap task. The requirements for the application would be that it fits within half the program flash and that the software has a task that can receive and program the new application and perform the swap.

Main Application Tasks

- Continue running during new firmware upload

Uploader Task

When activated, the uploader task performs the following:

- Issues command to put swap system to the *Update* state
- Erases the upper block sectors to be reprogrammed
- Receive new line of application code (from SCI, TCP/IP, and so on)
- Programs new line of code to the upper flash block
- Checks the code for errors
- Issues command to put swap in *Complete* state
- Performs software reset
- After the system is reset, the flash blocks are swapped and the new application boots up
- The process can be repeated again with the exact same steps

7 Software example

A software example is provided with this application note to demonstrate the swap feature. The example is not based on a multi-tasking operating system.

7.1 Software components

7.1.1 Flash Driver Software

The example uses the Flash Driver Software for Kinetis Microcontrollers (C90TFS_FLASH_DRIVER). The drivers are useful for compiling into an embedded application to add flash control capability. The example uses version 0.2.9 (beta) which has updated flash swap support.

7.1.2 Kinetis sample code

The Kinetis sample code for the K60 100 MHz version (KINETIS512_SC) was used as the basis for the example. Both an IAR and CodeWarrior project is provided. The flash drivers were added into the projects.

Supported Integrated Development Environments (IDE)

- IAR Embedded Workbench for ARM version 6.30 or later
- CodeWarrior for MCUs version 10.1

7.2 Hardware

Kinetis K60 100 MHz MCU Module (TWR-K60N512) with MCU device marked 4N30D or later required. See note below.

NOTE

Kinetis 100 MHz revision 1.4 or later required. The mask set number marked on the package is 4N30D. Subsequent revisions may be marked 5N30D, and so on. Revisions earlier than revision 1.4, may not support swap or may disable the Erase-All-Blocks command after swap initialized. Thus, on those early revisions, some debuggers will not be able to re-download to flash if they rely on the Erase-All-Blocks command to pre-erase the flash. Additionally, it was not possible to set the swap system back to uninitialized on those early devices.

7.3 Anatomy of the software

7.3.1 Flash driver swap functions

The flash driver provides two functions.

7.3.1.1 PFlashGetSwapStatus

Located in the file *PFlashGetSwapStatus.c*. This function returns the swap state details and checks whether any errors were detected in the swap state determination.

7.3.1.2 PFlashSwap

These are located in the files: *PFlashSwap.c*. This function performs the swap control commands.

7.3.2 Callback function

The calling application has the ability to pass a callback function to the PFlashSwap function. This callback function is then called by the PFlashSwap driver function after each time it advances the swap state. When the PFlashSwap function calls the callback function, it passes the swap state back to the calling application. Using this information, the application can determine whether to continue to the next swap state.

Additionally, this provides an excellent place to perform the firmware update. When the swap driver reports to the callback function that the swap system is in the *Update* state, the callback function downloads new system software (firmware), and erases/reprograms the upper block as needed.

7.4 Running the demo

NOTE

The following instructions assume previous installation and basic knowledge of the IDE and terminal utility. See Quick Start Guide for TWR-K60N512 for details available in http://www.freescale.com.

7.4.1 Instructions

1. Download and install the demo software. Install to a directory of your choice. This directory will be referred to **<Install Dir>**.
2. Plug in the USB cable between your computer and the board to provide power and debugger connection.
3. Open the PE Terminal Utility as described in the Quick Start Guide. Ensure the baud rate is 115200, no parity, and 8 bits. Click **Open Serial Port**.
4. Open IAR or CodeWarrior.
5. Open the swap demo project

   IAR
   - Open the workspace **<Install Dir>\build\iar\swap_demo\swap_demo.eww**

   CodeWarrior
   - Set the workspace to **<Install Dir>**
   - Import the project in the folder **<Install Dir>\build\cw\swap_demo\**
6. Select the target configuration

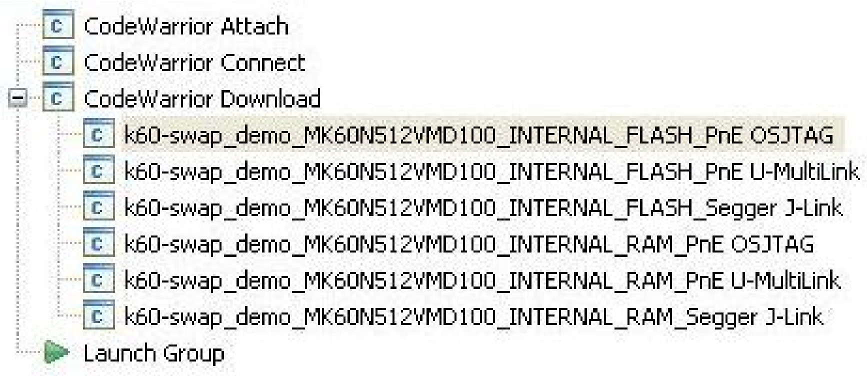
   IAR
   - Select the **FLASH_512KB_PFLASH** configuration.

   CodeWarrior
   - Select the **MK60N512VMD100_INTERNAL_FLASH** configuration
7. Compile the project
8. Download and Debug

   IAR
   - Click the Download and Debug button

CodeWarrior

- Open Debug Configurations
- Select **k60-swap_demo_MK60N512VMD100_INTERNAL_FLASH_PnE_OSJTAG**



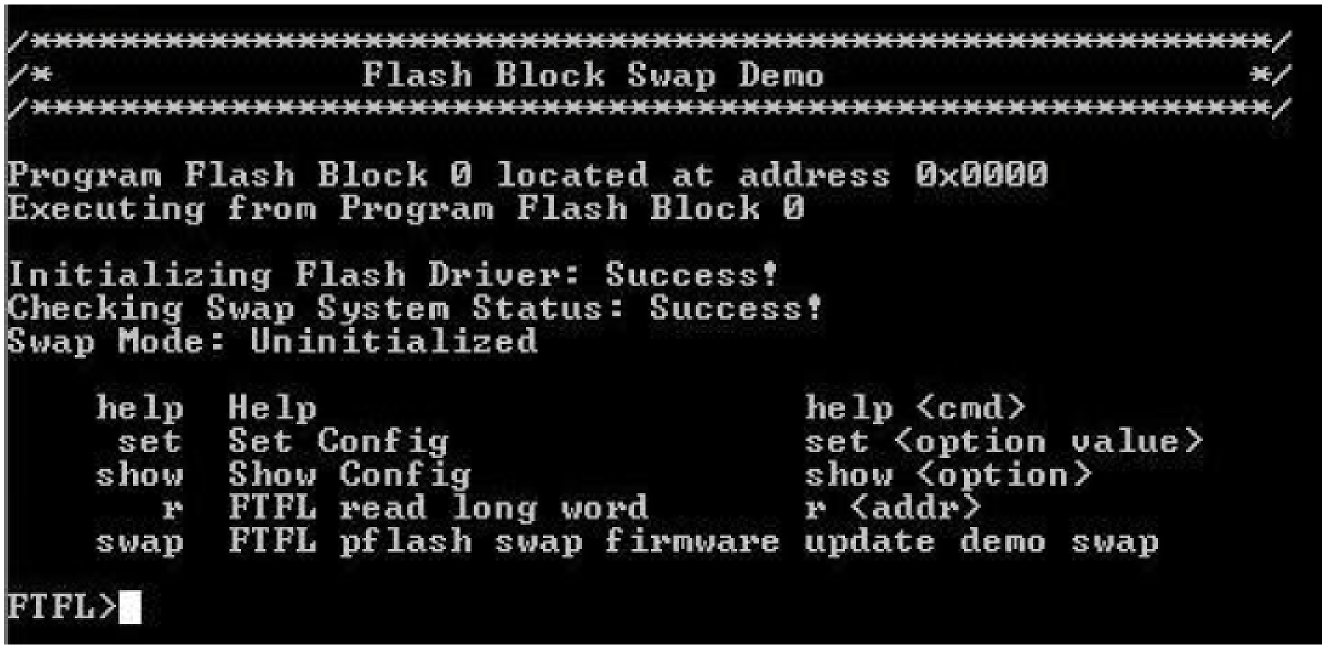
```
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
    k60-swap_demo_MK60N512VMD100_INTERNAL_FLASH_PnE OSJTAG
    k60-swap_demo_MK60N512VMD100_INTERNAL_FLASH_PnE U-MultiLink
    k60-swap_demo_MK60N512VMD100_INTERNAL_FLASH_Segger J-Link
    k60-swap_demo_MK60N512VMD100_INTERNAL_RAM_PnE OSJTAG
    k60-swap_demo_MK60N512VMD100_INTERNAL_RAM_PnE U-MultiLink
    k60-swap_demo_MK60N512VMD100_INTERNAL_RAM_Segger J-Link
Launch Group
```

Figure 7. Debug configurations

- Click Debug

9. In the PE Terminal Utility, the output will be displayed to the screen.

   First, the code checks which block is located at address 0x0000.
   Next, the code issues the swap command to check the swap system status. The first time you execute the code, the swap system will report Uninitialized.



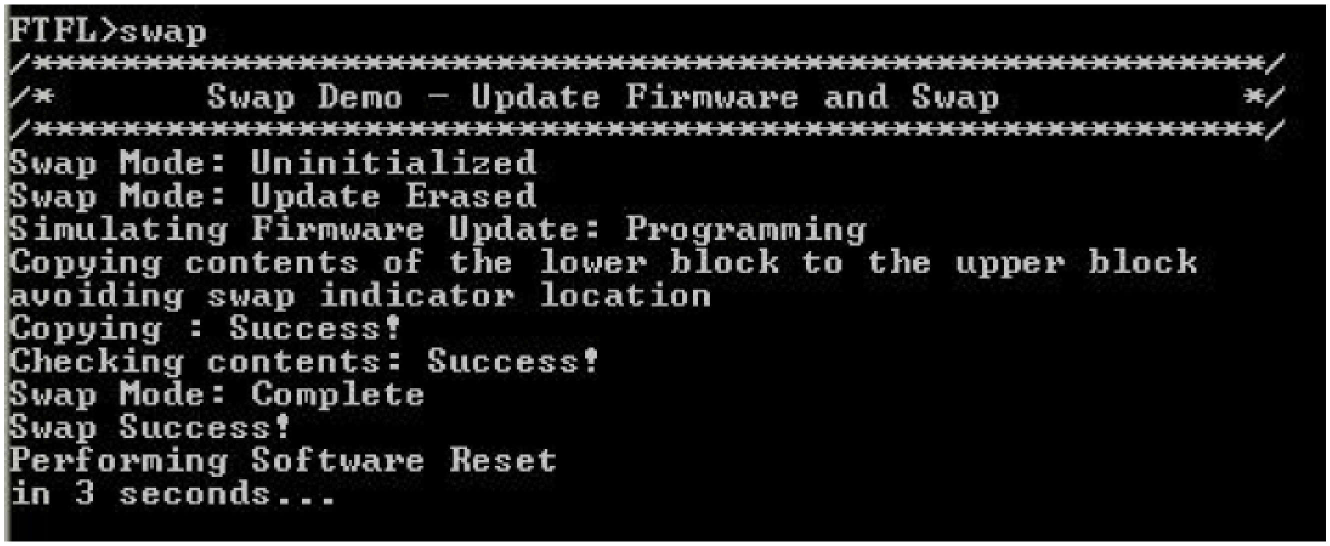
```
/*************************************************************/
/*                   Flash Block Swap Demo                  */
/*************************************************************/

Program Flash Block 0 located at address 0x0000
Executing from Program Flash Block 0

Initializing Flash Driver: Success!
Checking Swap System Status: Success!
Swap Mode: Uninitialized

    help  Help                    help <cmd>
     set  Set Config              set <option value>
    show  Show Config             show <option>
       r  FTFL read long word     r <addr>
    swap  FTFL pflash swap firmware update demo swap

FTFL>
```

Figure 8. Terminal utility output: before first swap

10. The code executes a command line interface. Type swap to start the swap demo. The swap driver will sequence the swap system through each swap state and call a callback function that reports the swap state back to the calling function.
11. When the swap system is in the *Update* state, the callback function simulates a firmware update, by erasing and reprogramming the nonactive (upper) block by copying the contents of the active (lower) block to the nonactive (upper) block.

NOTE

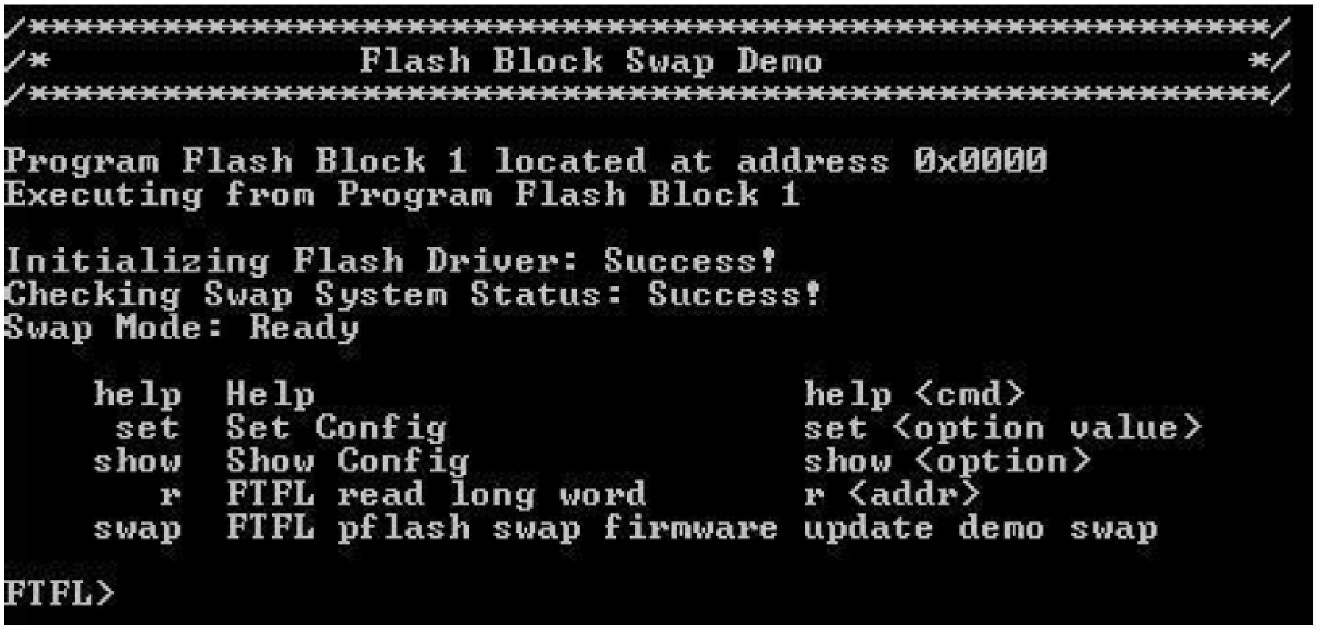
This example places the flash swap indicator in the last sector of each block. Programming the flash swap indicator is not allowed, so when programming the nonactive (upper) block, this location is avoided.



```
FTFL>swap
/*************************************************************/
/*        Swap Demo - Update Firmware and Swap              */
/*************************************************************/
Swap Mode: Uninitialized
Swap Mode: Update Erased
Simulating Firmware Update: Programming
Copying contents of the lower block to the upper block
avoiding swap indicator location
Copying : Success!
Checking contents: Success!
Swap Mode: Complete
Swap Success!
Performing Software Reset
in 3 seconds...
```

Figure 9. Terminal utility output: during first swap

12. The code executes a software reset in 3 s to complete the swap.



```
/*************************************************************/
/*                 Flash Block Swap Demo                    */
/*************************************************************/

Program Flash Block 1 located at address 0x0000
Executing from Program Flash Block 1

Initializing Flash Driver: Success!
Checking Swap System Status: Success!
Swap Mode: Ready

    help  Help                        help <cmd>
     set  Set Config                  set <option value>
    show  Show Config                 show <option>
       r  FTFL read long word         r <addr>
    swap  FTFL pflash swap firmware  update demo swap

FTFL>
```

Figure 10. Terminal utility output: after first swap

13. When the reset occurs, the code will boot up again. The flash blocks are now swapped. Block 1 is located at address 0x0000. The swap state is now in the Ready state. Type the swap command again to swap back to block 0.

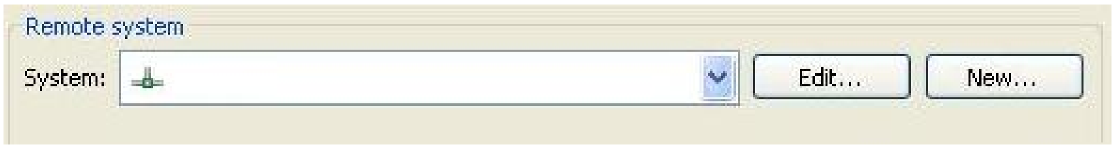
NOTE

If the debugger is connected when the software reset occurs, the debugger will halt. Either, resume the debugger or disconnect it and press the reset button on the board.

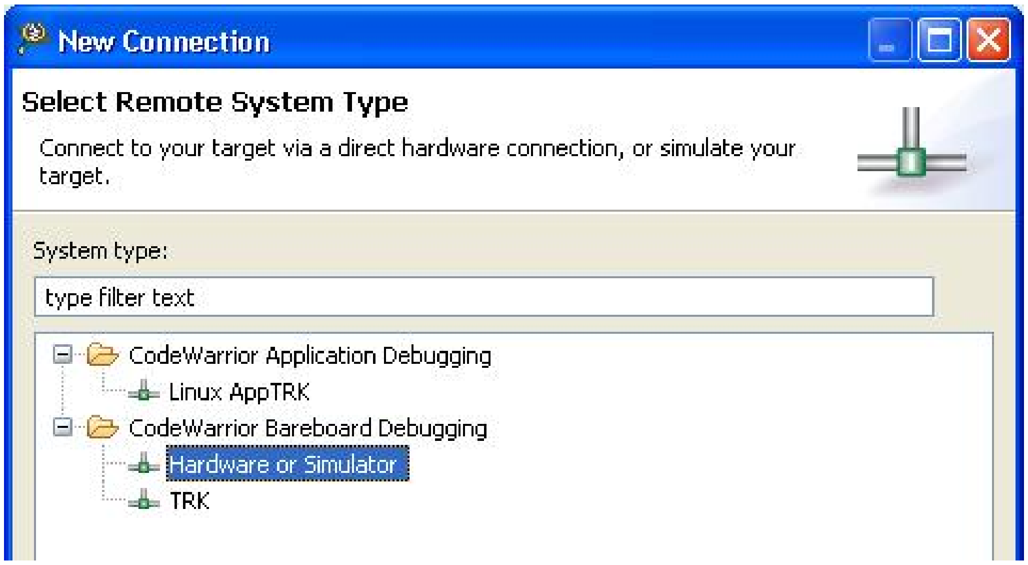
7.4.2 Setting up Remote System (CW 10.1)

The first time you run the demo in CodeWarrior 10.1, it may ask you to set up a Remote System. This is not required with later versions of CodeWarrior.

1. Open Debug Configurations, under Remote System, click **New…**



2. The New Connection window will open. Under CodeWarrior Bareboard Debugging, select **Hardware or Simulator**.


New Connection

Select Remote System Type

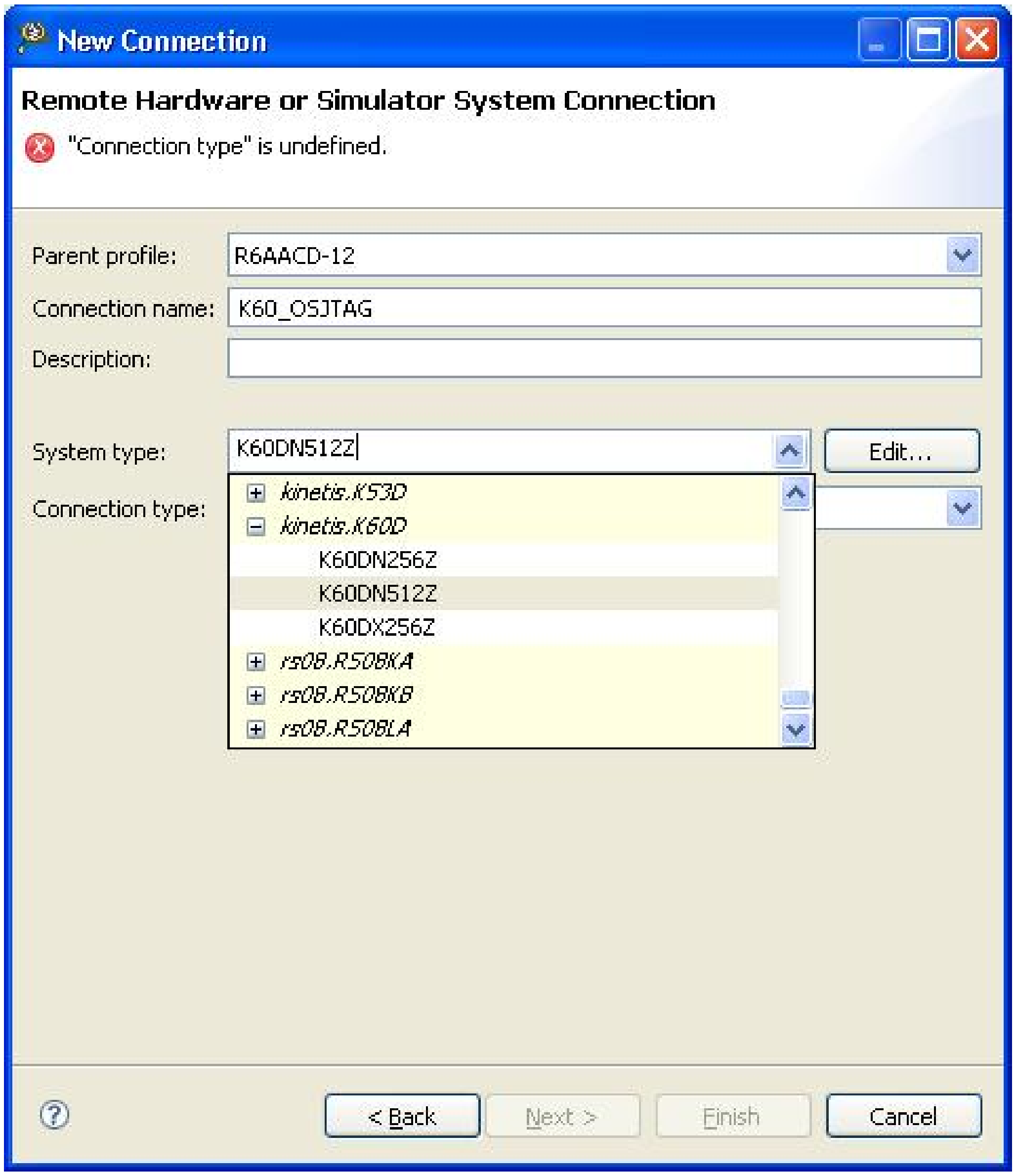
Connect to your target via a direct hardware connection, or simulate your target.

System type:

type filter text

- CodeWarrior Application Debugging
  - Linux AppTRK
- CodeWarrior Bareboard Debugging
  - Hardware or Simulator
  - TRK

3. Write in a name of your choice (e.g. K60_OSJTAG) for the **Connection name** and select **K60DN512Z** for **System** type. Click **Finish**.



New Connection

Remote Hardware or Simulator System Connection

"Connection type" is undefined.

Parent profile: R6AACD-12

Connection name: K60_OSJTAG

Description:

System type: K60DN512Z

Connection type:

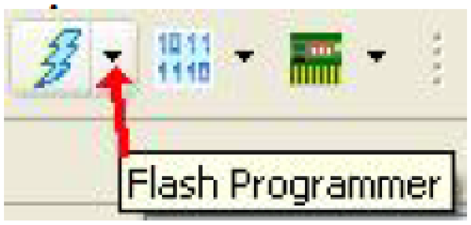
kinetis.K53D
kinetis.K60D
K60DN256Z
K60DN512Z
K60DX256Z
rs08.RS08KA
rs08.RS08KB
rs08.RS08LA

7.4.3 Recovering connection problem after running demo

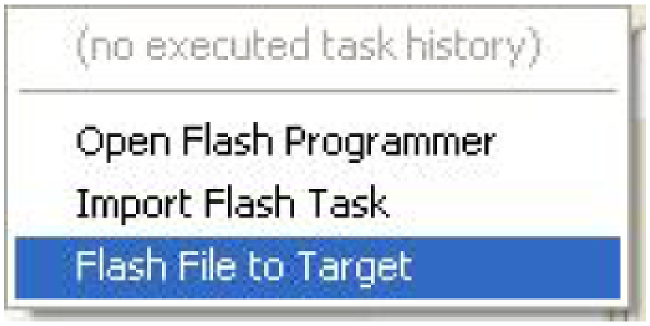
There is a potential issue in which CodeWarrior cannot erase/reprogram the device after the flash swap demo is executed. This is not related to the silicon revision.

To work around this, use the *Flash File to Target* tool to erase the flash. The *Flash File to Target* tool uses the Erase-All command which will erase the entire flash and clear the swap system back to uninitialized.

1. Open the Flash File to Target too by clicking the arrow next to the lightning bolt icon.



2. Select **Flash File to Target**



3. Click **Erase Device**

8 Conclusion

Program Flash Memory Swap is one of the many attractive features of Freescale's Kinetis microcontrollers. The Swap feature is ideal for systems that require a very robust firmware update mechanism that provides the safety of a backup copy and minimal application downtime. Program Flash Swap is simple to use and reduces software complexity. It is well suited for multi-tasking operating systems such as Freescale's complimentary MQX RTOS.

For further resources, please review the software example provided with this application note as well as the References section.

9 References

1. Kinetis Microcontroller documentation, software, and tools available at http://www.freescale.com/kinetis.
2. Flash Driver Software for Kinetis Microcontrollers. Keyword search C90TFS_FLASH_DRIVER at http://www.freescale.com.
3. Kinetis Sample Code. Keyword search KINETIS512_SC at http://www.freescale.com.

Document Number: AN4533
Rev. 0, 06/2012