AN204438

How to Set Up Flash Security for FM0+, FM3, and FM4 MCUs

This application note describes how to enable flash security for FM0+, FM3, and FM4 devices.

1 Introduction

You enable flash security to prevent an external tool from reading the code or data stored in flash memory. This application note describes how to enable flash security for FM0+, FM3, and FM4 devices. Flash security applies to the entire flash memory, including work flash or dual flash if present on the device.

This document describes two methods for enabling flash security, and provides the required flash sector address and data pattern to enable flash security on any FM device.

2 How Flash Security Works

Flash security is typically disabled by default on FM devices. Flash security is enabled when a certain data pattern (e.g. 0x0001) is written to a special flash sector address. The Flash Security Sector Overview section on page 7 lists the data pattern and flash sector address for each FM device.

When flash security is disabled, the user has full access to flash memory using flash programming tools and JTAG debuggers.

When flash security is enabled, flash contents cannot be accessed by external tools. Any read operation by the external tool reads meaningless values. However, user code always has access to the flash memory array. It is not protected by a supervisory mode or a Memory Protection Unit (MPU).

Flash security is not controlled by Boot-ROM or software – it is a part of the hardware macro of the flash memory itself. The only way to unlock flash security is to perform a flash chip erase sequence using a flash programming tool, which handles the order of events for you. The sequence is: if the device has work flash, you must erase work flash first before erasing main flash. For devices with dual flash, erase macro #1 before macro #0.

There are two ways to enable flash security:

- Using an assembly file and modifying the linker command file
- Patching the S-Record or HEX record file

In either case, you use a flash programming tool (serial or USB) to enable flash security by programming the device. You cannot use the JTAG interface if flash security is already enabled.

The following sections describe each approach in detail:

3 Enabling Flash Security Using an Assembly File

3.1 Procedure

1. Create a small assembly file with a memory section for the flash security sector. Define the flash security pattern in this memory section. Precise details vary by IDE.
2. Add this assembly file to the project when you are finished debugging, and just before the final project build.
3. To include this assembly file in your project, modify the linker command file for your IDE.
4. Use a flash programming tool to program flash memory with flash security enabled.

3.1.1 Assembly and Linker File for IAR EWARM

The following assembly code demonstrates how to enable flash security. In this example, the flash security sector address is 0x0010 0000 and the pattern is 0x0001:

```
        MODULE FLASH_SECURITY
        SECTION .flashsec: CONST (2)

        DC16 0x0001

        END
```

The filename is irrelevant. It is not necessary to give it the same name as the assembly module name.

Edit the corresponding linker file (*.icf) and add the following lines:

```
Define region FLASH_SECURITY_region = mem: [from 0x00100000 to
0x00100003];
Define symbol FLASH_SECURITY_ADDRESS = 0x00100000;
Place at address mem: FLASH_SECURITY_ADDRESS {read only
section .flashsec};
```

Assuming that the filename is *flash_security*, this section occurs in the linker map file after build:

```
"A2":                                   0x4
FLASHSEC const    0x00100000    0x2 flash_security.o [1]
                   - 0x00100002    0x2
```

3.1.2 Assembly File for KEIL µVISION

The following assembly code demonstrates how to enable flash security. In this example, the flash security sector address is 0x0010 0000 and the pattern is 0x0001:

```
            AREA    |.ARM.__at_0x00100000|, DATA, READONLY
            DCB 0x01
            DCB 0x00

            END
```

The filename is irrelevant.

Assuming that the filename is *flash_security*, this section occurs in the linker map file after build:

```
      Load Region LR$$.ARM.__at_0x00100000 (Base: 0x00100000, Size:
      0x00000004, Max: 0x00000004, ABSOLUTE)

          Execution Region ER$$.ARM.__at_0x00100000 (Base: 0x00100000, Size:
      0x00000004, Max: 0x00000004, ABSOLUTE, UNINIT)

          Base Addr    Size         Type   AttrIdx    E Section Name
      Object

      0x00100000   0x00000002   Data   RO
      83    .ARM.__at_0x00100000flash_security.o
```

3.1.3 Assembly and Linker File for AtollicTrueStudio

The following assembly code demonstrates how to enable flash security. In this example, the flash security sector address is 0x0010 0000 and the pattern is 0x0001:

```
      .section .flash security, "a"
      .global _flash security
      _flash security:    .long 0x00000001
```

The filename is irrelevant.

Edit the corresponding linker file (*.ld) and add the highlighted line in the MEMORY definition:

```
      /* specify the memory areas */
      MEMORY
      {
        FLASH (rx)      : ORIGIN = 0x00000000, LENGTH = 512K
        RAM (xrw)       : ORIGIN = 0x1FFF8000, LENGTH = 64K
        FLASHSEC (r)    : ORIGIN = 0x00100000, LENGTH = 4
        MEMORY_B1 (rx): ORIGIN = 0x60000000, LENGTH = 0K
      }
```

Add the highlighted lines to the SECTION definitions just before the RAM section definitions:

```
    . = ALIGN (4);
_edata=.;         /* define a global symbol at data end */
  } >RAM AT> FLASH

.flash security:
  {
   . = ALIGN(4);
   _flash security=.;
   KEEP(*(.flash security))
   . = ALIGN (4);
  } >FLASHSEC

  /* uninitialized data section */
  . = ALIGN (4);
```

Assuming that the filename is *flashsec.s*, this section occurs in the linker map file after build:

```
.flash security 0x00100000        0x4
                0x00100000                . = ALIGN (0x4)
                0x00100000                _flash security= .
 *(.flash security)
 .flash security
                0x00100000        0x4 source/flashsec.o
                0x00100004                . = ALIGN (0x4)
```

3.1.4 Assembly and Linker File for GNU Compiler Environment

Refer to Assembly and Linker File for AtollicTrueStudio.

4 Enabling Flash Security by Patching an S-Record or HEX File

4.1 S-Record File Format

A line in the S-Record file format consists typically of the character ‘S’, the type of the line, the byte count of the data payload of this line, an address, the data payload, and a checksum byte.

The type codes of S-Record lines are:

Code	Description	Number of Address Bytes	Data field
S0	Block Header	2	Yes
S1	Data Payload	2	Yes
S2	Data Payload	3	Yes
S3	Data Payload	4	Yes
S5	Record Count	2	No
S7	End of Block	4	No
S8	End of Block	3	No
S9	End of Block	2	No

It is a good idea to put the flash security data before the last S-Record line (`S7-9`):

Assuming that flash security is enabled by setting the address `0x00100000` to the value `0x0001`, then the additional S-Record line is:

```
S20810000001000000E6
```

Detailed explanation of this line:

	Data Payload with 3 Byte Address	Byte Count	3 Byte Address	Data Payload (Big Endian)	Check Sum
S	2	08	100000	01000000	E6

To calculate the checksum: add all bytes after the Sn type and before the checksum byte itself. Bytes with a value of 0x00 are skipped in the calculation. Then, take only the lower byte of the sum and invert the sum:

```
0x08 + 0x10 + 0x01 = 0x19
```

```
0x19 = 0xE6
```

4.2 HEX File Format

A line in the Intel HEX file format for 32-bit architectures consists typically of a colon, the byte count of the data payload of this line, an address, the type of the line, the data payload, and a checksum byte.

The type codes of HEX lines are:

Code	Description
0x00	Data payload
0x01	End of file
0x02	Extended Segment Address
0x03	Start Segment Address
0x04	Extended Linear Address
0x05	Start Linear Address

It is a good idea to put the flash security data before the last HEX line, which is usually:

```
: 00000001FF
```

Assuming that flash security is enabled by setting the address `0x00100000` to the value `0x0001`, then the additional HEX lines are:

```
: 020000040010EA
: 04000000010000000FB
```

Detailed explanation of these lines:

	Data Count	Dummy Address	Extended Linear Address	Upper 16 Bit of Security Address	Check Sum
:	02	0000	04	0010	EA

	Data Count	Lower 16 Bit of Security Address	Data Line	Security Code Data (Big Endian)	Check Sum
:	04	0000	00	01000000	FB

To calculate the checksum, add all bytes after ':' and before the checksum byte itself. Bytes with a value of 0x00 are skipped in the calculation. Take only the lower byte of the sum. Invert the sum and add 1.

Example for first line:

`0x02 + 0x04 + 0x10 = 0x16`

`0x16 = 0xE9`

`0xE9 + 0x01 = 0xEA`

5 Flash Security Sector Overview

The following tables give an overview of the flash security sector address and its activation pattern. Refer to the peripheral manual for cross reference between device type and part number.

5.1 FM0+ Devices

Series	Flash Security Address	Activation Pattern
S6E1A	0x0010_0000	0x0001
S6E1B	0x0010_0000	0x0001
S6E1C	0x0010_0000	0x0001

5.2 FM3 Devices

Series	Flash Security Address	Activation Pattern
CY9Ax3x	0x0010_0000	0x0001
CY9Ax4x	0x0010_0000	0x0001
CY9Ax5x	0x0010_0000	0x0001
CY9Ax10x	0x0010_0000	0x0001
CY9AxAx	0x0010_0000	0x0001
CY9Ax10A	0x0010_0000	0x0001
CY9Ax10K	0x0010_0000	0x0001
CY9AX20L	0x0010_0000	0x0001
CY9Bx10R	0x0010_0000	0x0001
CY9Bx10T	0x0010_0000	0x0001
CY9Bx20J	0x0010_0000	0x0001
CY9Bx20M	0x0010_0000	0x0001
CY9Bx20T	0x0040_0000	0x0001

5.3 FM4 Devices

Series	Flash Security Address	Activation Pattern
CY9BFx6xK/L	0x0040_0000	0x0001
CY9BFx6x M/N/R	0x0040_0000	0x0001
S6E2C	0x0040_0000	0x0001
S6E2D	0x0040_0000	0x0001
S6E2G	0x0040_0000	0x0001
S6E2H	0x0040_0000	0x0001

Document History

Document Title: AN204438 - How to Set Up Flash Security for FM0+, FM3, and FM4 MCUs

Document Number: 002-04438

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MAWI	07/01/2014	Initial Release
*A	5053149	MAWI	12/15/2015	Migrated Spansion Application Note FMx_AN706-00089-1v0-E to Cypress format
*B	5545007	JETT	12/13/2016	Format and language edit, update to latest templates Expanded Flash Security Sector Overview to include all FM types, and added cross reference to this section where useful
*C	5787579	AESATMP8	06/28/2017	Updated logo and Copyright.
*D	6271787	WOFR	08/03/2018	Updated product series names.

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 | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2014-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.