

Renesas Synergy™ Platform

Synergy Secure Data at Rest

Introduction

This application project discusses the considerations for secure **Data at Rest** design in an embedded system and provides guidelines on how to use the Synergy MCU security features to implement a secure Data at Rest solution. Secure **Data at Rest** refers to the features and services provided to protect sensitive data residing on a device which may or may not be modifiable.

Synergy MCUs offer data encryption, authentication schemes, and read/write and write-once access protection from CPU and bus masters for secure Data at Rest designs. In addition, Synergy MCUs provide security functions that disable control of certain security-related peripherals from non-secure software access.

Renesas has recently released a Secure Boot Manager (SBM) application project where the secure boot loader serves as the root of trust for the MCU device. **Data at Rest application guidelines when using SBM are not covered in this application project.**

For internal flash applications, this application project provides usage examples for sensitive data read protection, write protection, read/write protection, write-once protection, and write-once with read protection. For internal SRAM usage, this application project provides usage examples for read and write protection.

Upon completion of this guide, you will be able to use the secure Data at Rest features and solutions provided by Synergy in your own design, configure the secure components correctly for the target application, and write code using the included application example code as a reference. This guide provides a step-by-step operational flow for setting up the memory access features to efficiently use the security features of the Synergy MCU. More detailed hardware feature and API descriptions are available in the hardware user manual and the *Synergy Software Package (SSP) User's Manual* (see References section).

Required Resources

Development tools and software

- e² studio ISDE v6.2.1 or greater (www.renesas.com/synergy/e2studio)
- Visual Studio 2017 Community Version (<https://visualstudio.microsoft.com/downloads/>)
- Synergy Software Package (SSP) 1.5.3 or later (www.renesas.com/synergy/ssp)
- Segger J-link® USB driver (<https://www.segger.com/downloads/J-Link/>)
- Download and install Renesas Flash Programmer V3 using www.renesas.com/us/en/products/software-tools/tools/programmer/flash-development-toolkit-programming-gui.html#downloads

Hardware

- Renesas Synergy PK-S5D9 kit (www.renesas.com/synergy/pk-s5d9)
- Workstation running Windows 7 or 10; the Tera Term console, or similar application
- Two Micro USB cables

Prerequisites and Intended Audience

This application project assumes you have some experience with the Renesas e² studio ISDE and Synergy Software Package (SSP). Before you perform the procedures in this application note, follow the procedure in the *SSP User Manual* to build and run the Blinky project. Doing so enables you to become familiar with e² studio and the SSP and validates that the debug connection to your board functions properly. In addition, this application project assumes you have some knowledge on S5D9 flash memory organization and customizing e² studio linker script.

The intended audience are users who are interested in developing secure Data at Rest solutions with Synergy MCUs.

Contents

1. Secure Data Overview.....	5
1.1 Sensitive Data at Rest Topology	5
1.2 Data at Rest Security Measures.....	5
1.2.1 Data Encryption	5
1.2.2 Data Access Control.....	5
1.2.2.1 Read Protection.....	6
1.2.2.2 Write Protection.....	6
1.2.2.3 Read/Write Protection	6
1.2.2.4 Write-Once Protection	6
1.2.2.5 Write-Once and Read Protection	6
1.3 Secure Data at Rest Risk Profile and Attack Surface Analysis.....	7
1.4 Secure Data at Rest with and without Secure Boot Loader.....	7
2. Synergy MCU Features for Secure Data at Rest	7
2.1 Overview of Synergy Security Elements	7
2.2 Security MPU.....	8
2.2.1 Secure Data Regions	8
2.2.1.1 Secure Program	10
2.2.1.2 Secure Access Monitor and Protection.....	10
2.3 Flash Access Window (FAW).....	12
2.3.1 Using the Security MPU and FAW for Code Flash Write Protection.....	12
2.4 Debugging Security Considerations	13
2.5 Notes on ARM MPU, Bus Master MPU, Bus Slave MPU	14
2.6 Other Security Elements	14
2.6.1 Secure Crypto Engine (SCE)	14
2.6.1.1 Security Algorithms	14
2.6.1.2 Other Crypto Security Features	14
3. Configuring the Security Elements	14
3.1 Overview of Synergy MCU Option-Setting Memory.....	14
3.2 Configuring the Security MPU.....	15
3.2.1 Setting up the Security MPU Registers.....	15
3.2.2 Locating Secure Code/Data to a Specific Memory Region.....	16
3.2.3 Resetting the Security MPU registers	17
3.3 Configuring the FAW	17
3.3.1 Setting up the FAW Region.....	17
3.3.2 Clearing the FAW Regions.....	17
3.4 Permanent Locking of the FAW Region.....	18
3.5 Setting up the Security Control for Debugging.....	18
3.5.1 Methods of Setting the OSIS ID Code	18

3.5.1.1	Using the Synergy Configurator and J-Link Debugger	18
3.5.1.2	Using a J-Link script.....	19
3.5.1.3	Using the factory bootloader	19
3.5.2	Method of Setting up the OSIS ID Code for Debugging	19
3.5.3	Method of Resetting the OSIS ID code.....	20
4.	Operational Flow using Security MPU and FAW	20
4.1	Internal Flash and SRAM Read Protection	20
4.2	Internal Flash Write Protection.....	21
4.2.1	Operational Flow	22
4.3	Internal Flash and SRAM Read/Write Protection.....	22
4.3.1	Operational Flow	22
4.4	Internal Flash Write-Once Protection	23
4.4.1	Operational Flow	23
4.5	Internal Flash Write-Once and Read Protection	23
4.5.1	Operational Flow	24
4.6	Operation Notes	24
4.6.1	Memory Allocation.....	24
4.6.1.1	Space between Secure and Non-secure Instructions.....	24
4.6.1.2	Do Not Assign the Memory Mirror Region to the Secure Region	24
4.6.1.3	Startup Area of Security MPU Regions.....	24
4.6.2	Limitations on Programming the Option-Setting Memory	24
4.6.3	Factory Bootloader Accessibility	24
4.6.4	Access Secure Function from Non-Secure Functions	24
4.6.5	Debugger Access to the Security MPU Regions.....	25
5.	Security Application e ² studio Projects: Internal Flash and SRAM	25
5.1	Project 1 e ² studio project: Internal Flash and SRAM Read Write Protection.....	25
5.1.1	Software Architecture Overview	25
5.1.2	Memory Allocation Arrangement.....	26
5.1.3	Functionality Description	28
5.1.4	Establishing and Running Software from Secure SRAM Region.....	29
5.1.5	Importing and Building the Project	30
5.1.6	Hardware Setup	30
5.1.7	Verifying the Secure Functionalities.....	30
5.1.7.1	Summary of the functionalities provided in this application project:	34
5.1.8	Migrating to other Synergy MCUs	34
5.2	Project 2: e ² studio Project: Reset the Security MPU and FAW setting.....	35
5.3	Project 3: PC Application to Permanently Lock the FAW	35
5.4	Example Reset J-Link Script for the Security MPU and FAW.....	37
5.5	J-Link Scripts for Resetting OSIS ID code for S5D9.....	38

- 6. Secure Data at Rest Next Steps..... 41
 - 6.1 Secure Data Encryption and Authentication 41
 - 6.2 External Storage Secure Data at Rest 41
 - 6.3 Example using the Security MPU Security Functions 41
- 7. References 41
- Revision History 43

1. Secure Data Overview

With the dawn of AI, IoT, and Cloud connectivity, digital data security became the number one priority when protecting trade secrets and personal privacy.

Secure data technology includes Data in Transit and Data at Rest. **Data in Transit**, or data in motion is data actively moving from one location to another, such as across the internet or through a private network. Data protection in transit is the protection of data while it is traveling from network to network or being transferred from a local storage device to a cloud storage device. **Data at Rest** is data that is not actively moving from device to device or network to network, such as data stored on a hard drive, laptop, flash drive, embedded memory, or archived/stored in some other way. Data protection at rest aims to secure inactive data stored on any device or network. This application project focuses on Data at Rest design in an embedded environment using a Synergy MCU.

Data at Rest protection uses **Data Encryption** and **Data Access Control** as major security measures. This application project provides detailed steps to establish the Write Once and Read/Write Access Control for a Synergy MCU as well as guidelines for applying these security features to a wide range of secure Data at Rest applications. Data Encryption is not covered in this release and may be provided in later release.

SBM (Secure Boot Manager) is the reference Synergy secure boot solution. (See the References section for direct links to this solution). When considering securing Data at Rest, one should consider the impact of using a secure boot solution to the overall application design. Detailed discussion on Data at Rest with SBM usage is beyond the scope of this application project.

1.1 Sensitive Data at Rest Topology

In an embedded system, secure data can reside in volatile data storage (MCU's internal SRAM or external SDRAM) or non-volatile data storage (such as MCU's internal flash storage, external QSPI storage, and external EEPROM storage.) As part of the application security design, one must consider the topology of the data based on its use case. As an example, in a medical device some data (like blood pressure measured every 5 minutes) can be stored in volatile memory while other types of data (daily blood pressure averages) may need to be stored in non-volatile memory for future use. One should consider the nature of the data and therefore determine its topology before beginning the design as this decision will have an impact on securing the data.

With the first release of this application project, example code will be available to demonstrate how to secure data that resides in internal SRAM or internal flash storage. The follow-on release of this application project will describe the methodology and provide sample projects to secure data in external storage (QSPI, EEPROM, etc.).

1.2 Data at Rest Security Measures

Encryption and **Access Control** are two of the main secure Data at Rest protection schemes that will be discussed in this application project. These two schemes apply to both volatile and non-volatile storage types including internal and external storage. With the first release of this application project, access control for internal storage (both volatile and non-volatile) will be covered with example projects.

1.2.1 Data Encryption

Data Encryption is widely used in both secure **Data at Rest** and **Data in Transit**.

Securing internal data through encryption is increasingly becoming a necessity for small ARM Cortex MCUs as these devices are used more in networking and communication applications. Secure Data at Rest almost always means the data is encrypted or certain protocols exist that include encryption to protect the data from unauthorized access. The SCE (Secure Crypto Engine) feature of Synergy devices are, for example, used to generate encrypted private keys.

An example use of encryption of Data at Rest is encryption of data in external storage. An embedded system could use an AES key to encrypt sensitive data and code that resides on the external storage. Upon successful authentication, the external code data can be decrypted and used.

1.2.2 Data Access Control

Increased demands for device connectivity as well as increased complexity in embedded systems result in more potential attack surfaces exposed. Controlled access to the secure data effectively reduces the attack surface, thus increasing system security. The following is a brief introduction to possible uses cases where access controls provided in Synergy can be applied.

1.2.2.1 Read Protection

Sensitive data and code residing in flash and SRAM can have read protection properties set such that only software granted with read permission can access them. Synergy MCUs have a Security MPU unit that can help establishing sensitive regions with read protection.

- Section 2.2 introduces the Security MPU's functionality
- Section 3.2 provides the configuration methods for the Security MPU
- Section 4.1 explains the operational flow of establishing read protection using the Security MPU.

1.2.2.2 Write Protection

It is important to protect sensitive data from being maliciously modified or erased. Volatile and non-volatile data can be write protected to avoid unauthorized writes by using the memory options setting in a Synergy device. There are two ways to establish flash write protection:

- The Synergy Security MPU can disable the flash erase and write access from non-secure software access. See section 2.2 for details.
- The Synergy MCU Flash Access Window (FAW) can protect sensitive flash data from being modified by secure and non-secure software.
 - Section 2.3 introduces the FAW's functionality
 - Section 3.3 provides the configuration methods for the FAW
 - Section 4.2 explains the operational flow of establishing write protection using the FAW.

1.2.2.3 Read/Write Protection

Read/Write protection reduces the attack surface from malware and IP theft. For internal flash data, similar to write protection, there are two ways Synergy devices can provide read/write protection:

- The Synergy Security MPU can disable read and write access to the security MPU flash and SRAM regions from non-secure software.
- When the Security MPU and FAW are used together, the sensitive data in flash can be read & write protected from both secure and non-secure software. An example project for this use case is provided in this release.
 - Section 4.3 explains the operational flow of establishing read & write protection using the Security MPU and FAW.

1.2.2.4 Write-Once Protection

In some use cases, sensitive Data at Rest needs to be protected from access or alteration for the lifetime of the device. For example, a secure boot loader must be immutable for the lifetime of the product. For use cases where the data resides on internal flash, FAW settings can be programmed to provide write-once protection.

It is important to note the implication of using a secure boot manager in your design. If your end application will use a secure bootloader (such as Renesas's SBM – Secure Boot Manager), then special consideration must be taken to include the write-once data memory regions for the application with the regions that the secure bootloader reserves for its own. This must be done because setting the Flash Access Window (FAW) properties to implement the write once policy can only be done once for the lifetime of the device. In other words, once the FAW policies are programmed, they cannot be changed.

- Section 4.4 explains the operational flow of establishing write-once secure data and program usage.

1.2.2.5 Write-Once and Read Protection

Write-once protected data can be optionally read protected. When handling sensitive data, read protection can be provided to the write-once protected flash data such that only secure software can read the contents. For Synergy MCUs, this is realized by using both the Security MPU and FAW.

- Section 4.5 explains the operational flow of establishing write-once and read protection for Synergy MCUs.

1.3 Secure Data at Rest Risk Profile and Attack Surface Analysis

To fully consider and design for secure Data at Rest in an embedded environment, one should thoroughly consider the following topics:

1. Consider who will have access to the sensitive data in the embedded system.
2. Consider if the CPU bus can access the sensitive data.
3. Consider if other bus masters can access the sensitive data. If so, determine which peripheral the bus master connects to, and what entity this peripheral communicates with.
4. Consider if the debugger can access the sensitive data.
5. Consider robustness of the application design such that there are measures taken against the application itself to accidentally damage the sensitive data by overwriting the security policies and measures in place.

Reducing the attack surface helps in all of the above situations. Securing the entire MCU's memory may not effectively enhance the overall data security, since a larger attack surface translates to a higher chance that hackers will find a weak point. A good guideline for securing sensitive data is to design the application such that only the minimum amount of data is secured, and access is controlled through strategic interfaces.

The analysis of a system's risk profile and attack surface is outside of the scope of this application note; however, the security measures offered by Synergy will be introduced to help in reducing the attack surface and minimizing the system's risk profile.

1.4 Secure Data at Rest with and without Secure Boot Loader

Whether your system will or will not use a secure bootloader solution will have an impact on how to implement secure Data at Rest.

As mentioned before in section 1.2.2 Data Access Control, there may be a need for the application to protect sensitive data for the lifetime of the device and a secure boot manager such as the Renesas SBM is being used. In this case, one must consider the setting of FAW (Flash Access Window) properties appropriately to ensure that the application-specific sensitive data can be allocated in the unmodifiable by FAW region and programmed onto the MCU at the same time the secure boot manager is programmed during the manufacturing provisioning process. In addition, if the application must protect sensitive data with read protection only, the Security MPU region needs to be appropriately considered to ensure that the application-specific read-protected sensitive data can be allocated within the Security MPU region.

It is worth noting that Synergy devices allocate the Security MPU setting in the first block of the user flash (which is a block of 8 KB flash). As a result, if there is a need to permanently lock the Security MPU settings, the first 8KB of internal flash needs to be part of the write-once flash region. SBM is an example of such use case. Please refer to the specific device user manual for further details.

2. Synergy MCU Features for Secure Data at Rest

Synergy MCUs provide a rich set of hardware features to address secure Data at Rest needs. From a high level, Synergy MCUs support the following data protection features:

- Write Once access control for internal flash
- Read/write access control for internal flash
- Read/write access control for internal SRAM
- Data access protection from bus masters and debuggers
- Hardware data encryption support for both volatile and non-volatile, internal and external storage
- Security functions to disable certain peripherals from being accessed by a non-secure program

This chapter will introduce the hardware features that can support the above security features without getting into the operational details. Details on the configurations of these hardware components are provided in chapter 3.

Integration of the MCU internal data access control with encryption support for external data provides a consistent level of security support for the entire embedded system.

2.1 Overview of Synergy Security Elements

Below is a list of security elements on the Synergy MCUs.

- Security MPU
- Flash Access Window (FAW)
- Secure Crypto Engine (SCE)
- Debug Protection with OSIS

Table 1 summarizes the availability of the secure elements on the Synergy MCUs.

Table 1. Secure Elements of Synergy MCUs

MCU Feature	S7	S5	S3	S1 except S124	S124
Security MPU	n/a	yes	yes	yes	n/a
FAW	yes	yes	yes	yes	yes
SCE	yes (*)	yes (*)	yes (*)	yes (*)	yes (*)
OSIS Register	yes	yes	yes	yes	yes

Note *: Please refer to section 2.6.1 for the details on the SCE support for the Synergy MCU families.

This section will also explain how the ARM MPU, bus master MPU, and bus slave MPU operation relate to Data at Rest from a high level.

2.2 Security MPU

The Security MPU feature of Synergy devices implements a set of versatile data security policies by creating isolation between different software and hardware components. Identification of secure and non-secure program and data is based on address location.

The Security MPU settings are read and applied before the reset vector is fetched, and therefore apply before any code is executed.

Note: Security MPU memory features exists on every Synergy device except S7G2 and S124.

2.2.1 Secure Data Regions

Figure 1 describes the available secure data regions and secure program regions provided by the Security MPU. Each region is defined by a pair of start and end addresses and an enable bit.

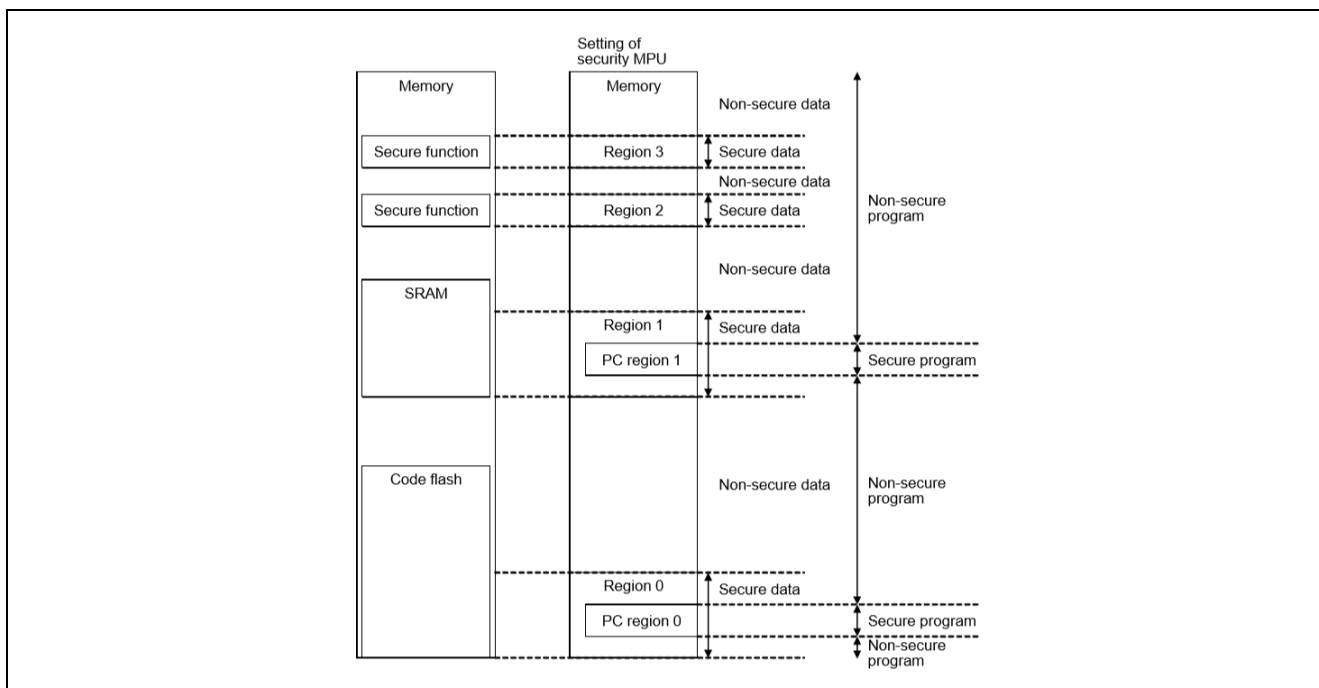


Figure 1. Security MPU Secure Regions

The Security MPU provides four secure data regions, as shown in Figure 1.

- One secure flash data region (located in the MCU's code flash region). Note that in terms of Security MPU usage, the secure flash data region can contain both secure flash data and secure flash program.

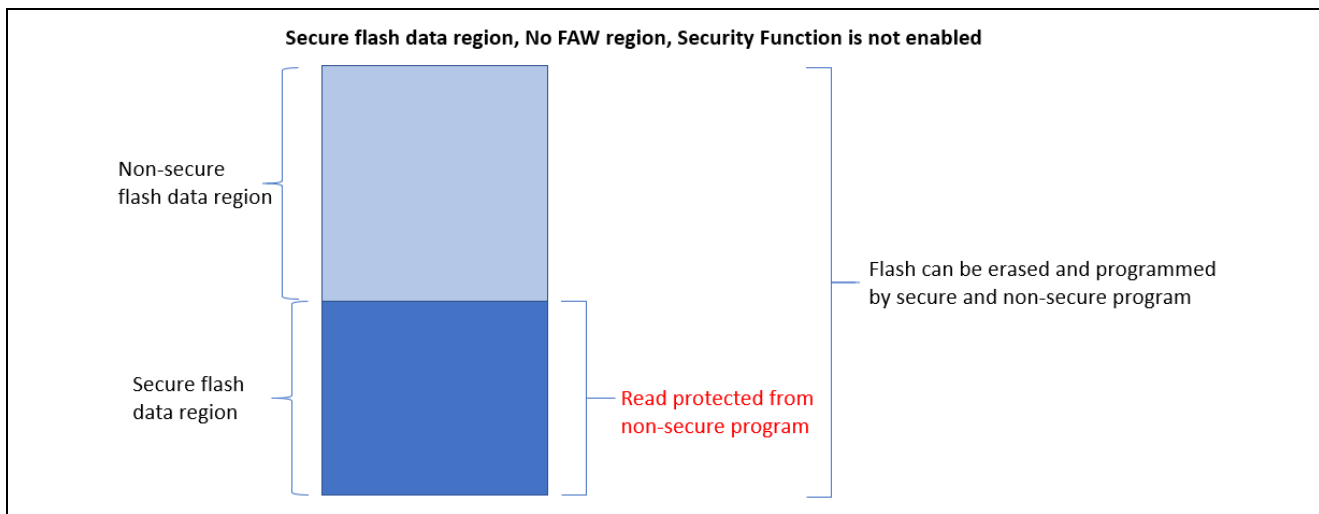


Figure 2. Secure flash data region

- One secure SRAM data region. Note that in terms of Security MPU usage, secure SRAM data region can contain both secure SRAM data and secure SRAM program.

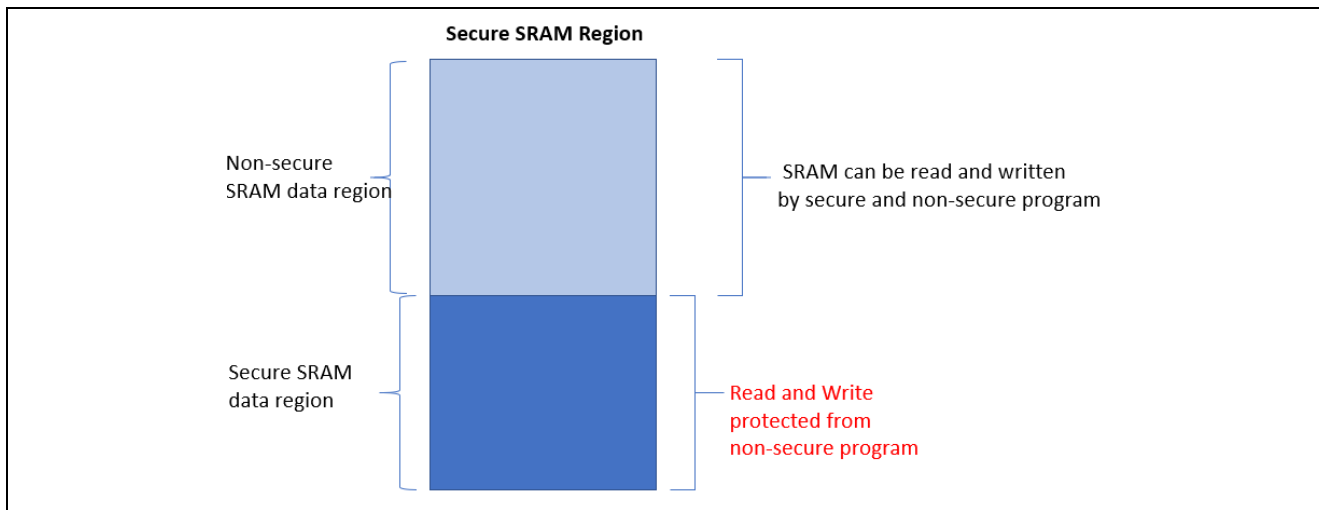


Figure 3. Secure SRAM data region

- Two security function regions
 - Security function region for the SCE (Secure Crypto Engine): address 0x400C0000 to 0x400DFFFF
 - This region maps to the Synergy MCU's internal peripheral bus 7
 - When this security function is enabled, the SCE cannot be controlled from a non-secure program
 - Security function for the flash (code and data) erase and program (E/P): address 0x40100000 to 0x407FFFFF
 - This region maps to the Synergy MCU's internal peripheral bus 9
 - When this security function is enabled, the flash cannot be put in erase and program mode from non-secure program
 - Note that this security function does not control TSN even though it is in the same memory space.

Addresses	Bus	Area
0000 0000h to 01FF FFFFh	Memory bus 1, 3	Code flash memory
1FFE 0000h to 1FFF FFFFh	Memory bus 2, 3	SRAMHS
2000 0000h to 2003 FFFFh	Memory bus 4	SRAM0
2004 0000h to 200F FFFFh	Memory bus 5	SRAM1 and Standby SRAM
4000 0000h to 4001 FFFFh	Internal peripheral bus 1	Peripheral I/O registers
4004 0000h to 4005 FFFFh	Internal peripheral bus 3	
4006 0000h to 4007 FFFFh	Internal peripheral bus 4	
4008 0000h to 4009 FFFFh	Internal peripheral bus 5	
400C 0000h to 400D FFFFh	Internal peripheral bus 7	
400E 0000h to 400F FFFFh	Internal peripheral bus 8	Graphic IPs (JPEG, GLCDC, and DRW)
4010 0000h to 407F FFFFh	Internal peripheral bus 9	Flash memory (in P/E*1), data flash memory, and TSN
6000 0000h to 67FF FFFFh	External bus	QSPI area
8000 0000h to 97FF FFFFh	External bus	CS area and SDRAM area

Figure 4. Security Function Regions

2.2.1.1 Secure Program

The Security MPU provides two secure program regions, each defined by a pair of start and end Program Counter (PC) addresses and an enable bit. There can be one secure flash program region and one secure SRAM program region. For security purpose, it is recommended that the secure program regions are located within the secure flash and secure SRAM data region respectively as shown in Figure 1.

Note: The secure program region can reside outside of the secure flash and secure SRAM region; however, in this case, there is a security hole. The secure program in Flash and SRAM can be read by non-secure software.

Non-secure program and secure program can make function calls into each secure and non-secure region, which provides efficiency in embedded system design.

2.2.1.2 Secure Access Monitor and Protection

- The Security MPU monitors and protects secure flash and secure SRAM data regions (Region 0 and Region 1) from unauthorized CPU read from the D Code bus (Data access), as shown in Figure 5. The I Code bus (Instruction access) is not monitored. This allows secure instructions to be accessed by non-secure program.
- The Security MPU monitors and protects secure SRAM data and security function regions from system bus access (e.g. debugger access), as shown in Figure 5.
- The Security MPU monitors and protects all four secure data regions from access from the three bus master groups. See the hardware user manual for definition of the bus master groups.

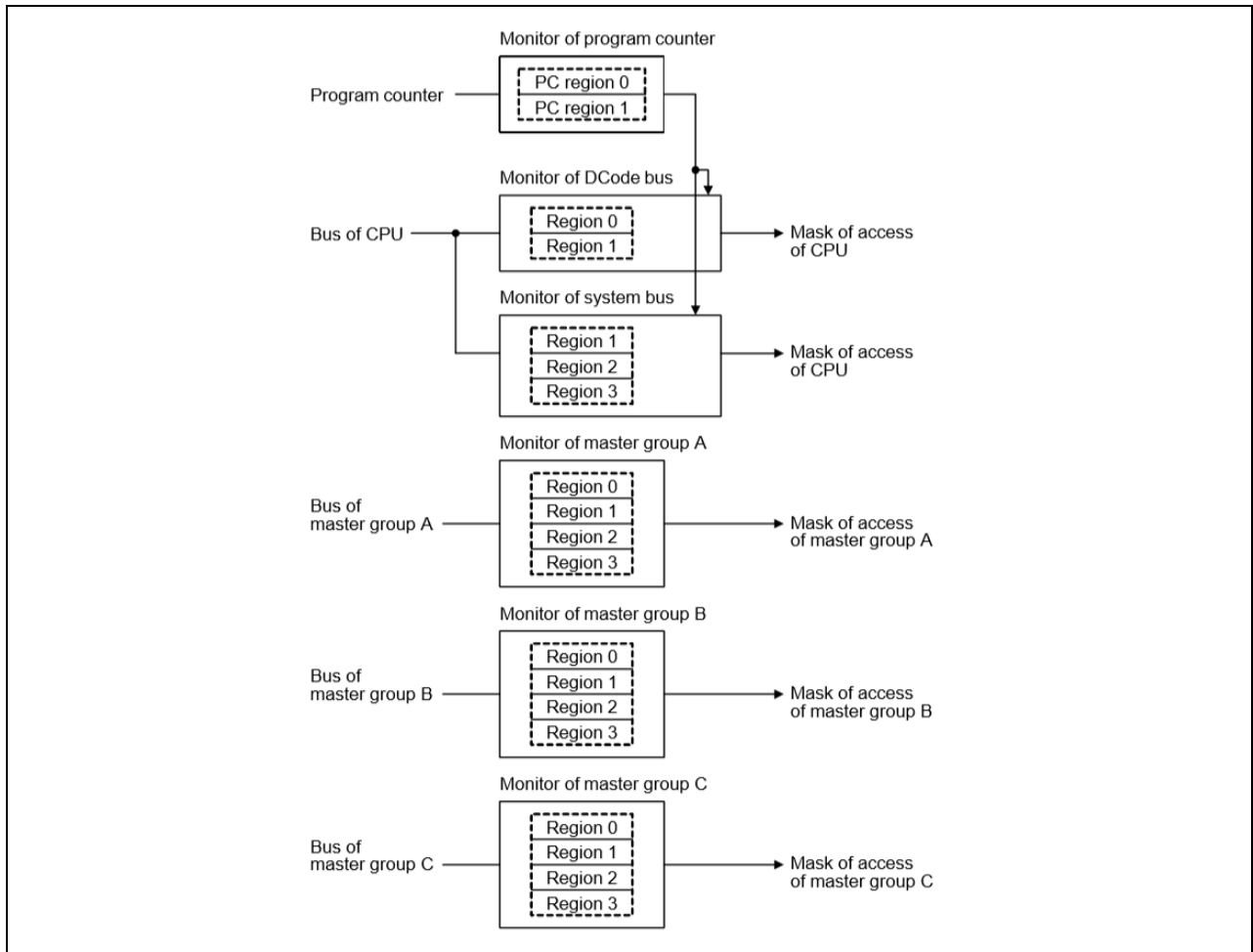




Figure 5. Data Protection for Security MPU Regions

The table in Figure 6 describes the details of CPU access and debugger access to the secure and non-secure regions.

-  indicates that the bus master on the left is granted access to the region on the top
-  indicates that the bus master on the left is not granted access to the region on the top

Note: The secure program located in the SRAM region other than the SRAMHS region cannot be fetched by a non-secure program..

Configuration details of the Security MPU are described in section 3.2.

PC value	Master access	Slave region 0		Slave region 1				Slave region 2/3	
		FLASH		SRAMHS		Other than SRAMHS		E2P/Flash Cnt./TSIP	
		Sec	Non-sec	Sec	Non-sec	Sec	Non-sec	Sec	Non-sec
Secure	CPU (instruction)	✓	✓	✓	✓	✓	✓	-	-
Non-secure	CPU (instruction)	✓	✓	✓	✓	✗	✓	-	-
Secure	CPU (operand)	✓	✓	✓	✓	✓	✓	✓	✓
Non-secure	CPU (operand)	✗	✓	✗	✓	✗	✓	✗	✓
Secure	Debugger	✗	✓	✗	✓	✗	✓	✗	✓
Non-secure	Debugger	✗	✓	✗	✓	✗	✓	✗	✓
[Don't Care]	DMAC/DTC and others	✗	✓	✗	✓	✗	✓	✗	✓

Figure 6. CPU and Debugger Access Summary

2.3 Flash Access Window (FAW)

The Flash Access Window (FAW) defines one contiguous flash region within the MCU flash space. Within this region, the flash erase and write operation is allowed from both secure and non-secure program. The access window is only valid in the program flash area. The access window provides protection in self-programming mode, serial programming mode, and on-chip debug mode.

The FAW region setting uses flash block boundaries as the start and end addresses of the flash access window. There is one bit (FSPR bit) in the FAW control register that, when cleared, disables any possible update to the flash access window setting for the lifetime of the device. The internal flash write-once protection uses this bit to implement the write-once protection.

The configuration area cannot be written while DTC/DMAC, EDMAC, or LCDC/2DG/JPEG are enabled. Refer to section 3.3 for the configuration details on the FAW.

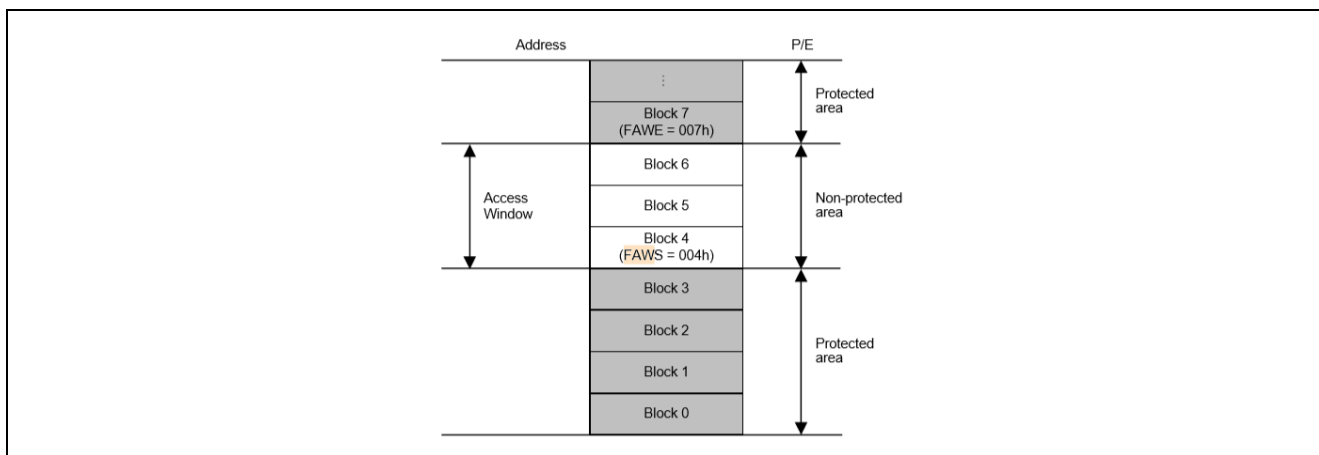


Figure 7. Flash Access Window

2.3.1 Using the Security MPU and FAW for Code Flash Write Protection

As explained in section 1.2.2, there are two ways to establish flash region write protection. In this application project, the FAW is used to establish flash write protection. Figure 8 summarizes the code flash read, erase and programming access control when the Security MPU is used in conjunction with the FAW, and the security function is not enabled.

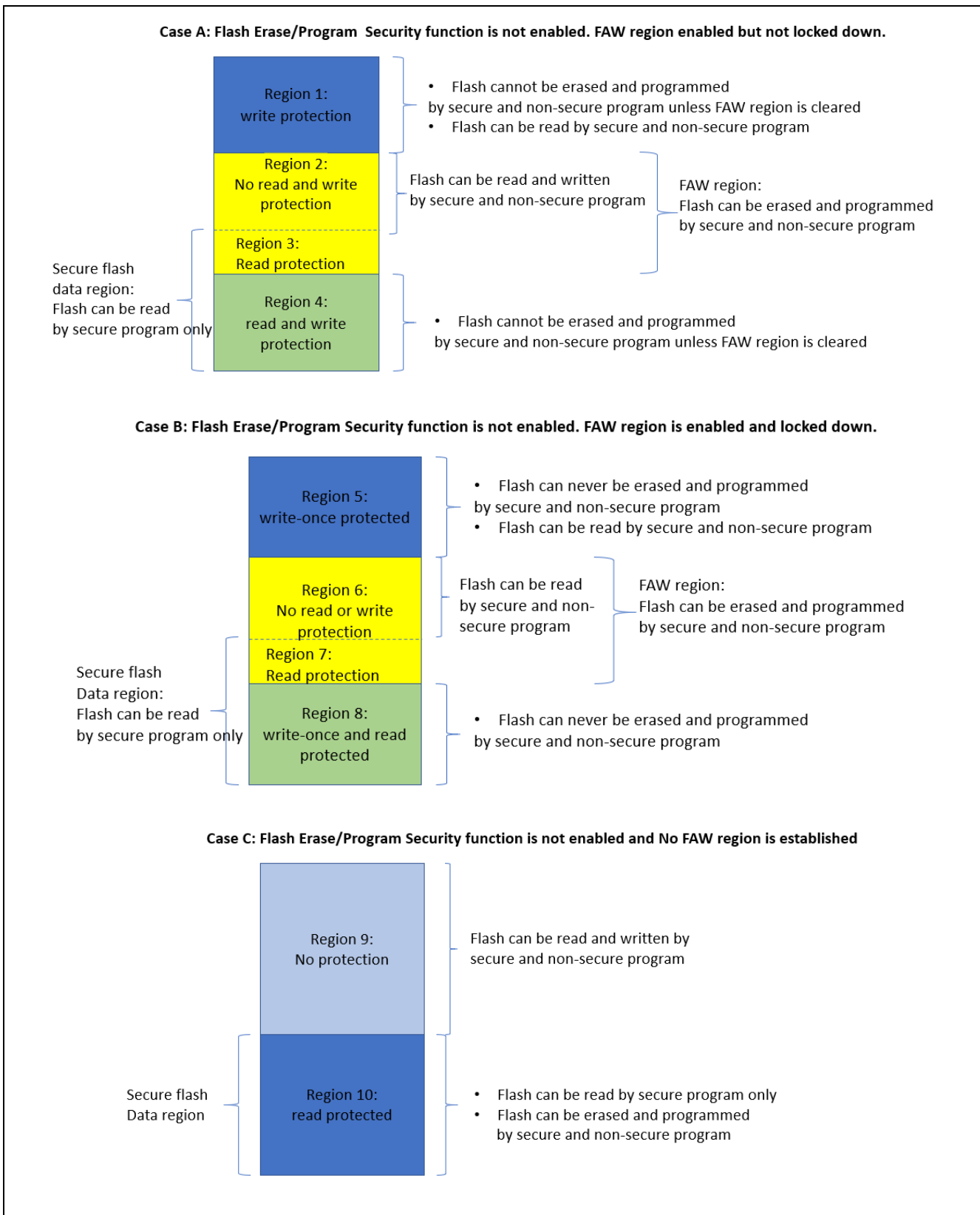


Figure 8. Code Flash Read, Erase and Program Control when Security Function is not enabled

2.4 Debugging Security Considerations

The Synergy MCU provides OCD/serial programmer ID code protection with the OSIS register. The OSIS register stores the ID for ID code protection of the OCD/serial programmer. Once the OSIS register ID code is set for the MCU, the user must provide the matching ID code when connecting an OCD/serial

programmer. If the ID code matches, debugging is allowed; otherwise, the debugging process is not allowed. Configuration of the OSIS ID code protection is described in section 3.5.

2.5 Notes on ARM MPU, Bus Master MPU, Bus Slave MPU

This section explains how the MPU, Bus Master MPU, and Bus Slave MPU relate to Data at Rest design. Please refer to the ARM Cortex technical user manual to understand the definition and settings of the ARM MPU. Please refer to Synergy MCUs hardware user manuals to understand the definition and settings of the Bus Master and Bus Slave MPUs.

While these three MPUs intend to catch inadvertent accesses to the regions defined by these MPUs, they do not provide protection of reading and updating the register settings from non-secure program. The register settings of these MPUs are not protected from reading by a debugger nor by non-secure programs. Secure and non-secure programs can follow correct procedures to update the registers.

In addition, the MPU regions defined by these three MPUs do not provide the same level of security compared to the protection provided by the Security MPU:

- A debugger can access the protected regions
- A read-protected region (without write protection) can be written by secure and non-secure code
- A write-protected region (without read protection) can be read by secure and non-secure code
- A read/write-protected region cannot be read nor written by either secure or non-secure code

2.6 Other Security Elements

2.6.1 Secure Crypto Engine (SCE)

The Secure Crypto Engine (SCE) hardware block on Synergy MCUs provides data encryption and authentication capability. Below are the encryption and authentication algorithms supported:

2.6.1.1 Security Algorithms

- Symmetric algorithms: AES, 3DES and ARC4
- Asymmetric algorithms: RSA, DSA and ECC
- MCU family support status:
 - S5 and S7 MCU families supports all the above symmetric and asymmetric algorithms
 - S3 and S1 supports AES algorithm only

Configuration details of the SCE is outside the scope of this application project. Please refer to the hardware user manual and SSP user manual for operational details.

2.6.1.2 Other Crypto Security Features

- TRNG (True Random Number Generator)
- Hash value generation: SHA1, SHA224, SHA256, GHASH, and MD5
- 128-bit unique ID
- MCU family support:
 - S5 and S7 MCU families support this entire group of features
 - S3 supports TRNG and GHASH
 - S1 supports TRNG only

3. Configuring the Security Elements

This section explains the detailed process to setup the Security MPU, FAW, and the OSIS ID register to provide the desired security level for Data at Rest security design. It also explains how to undo the security feature settings in the development stage. In addition, steps needed to lock the security features in the MCU are provided.

3.1 Overview of Synergy MCU Option-Setting Memory

The Security MPU and the FAW registers are in the MCU's Option-Setting Memory, as shown below. The option-setting memory is in the flash user area. Setting these registers requires different procedures compared to erasing and programming the other part of the code and data flash area. Figure 9 is an example location of the option-setting memory based on S5D9. Other MCUs can have the FAW register

located at a different location. Please refer to the hardware manual for the exact location for the MCU involved.

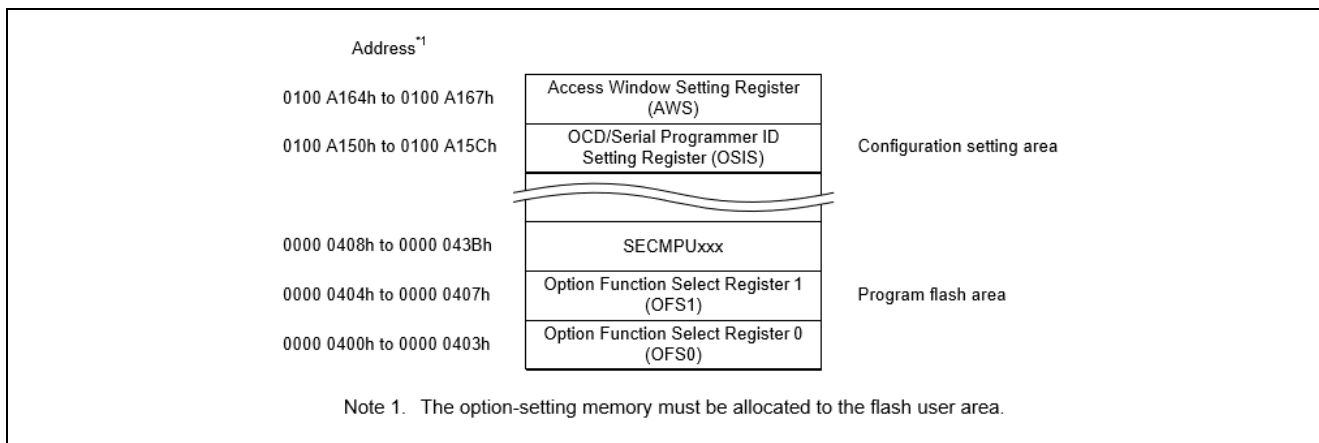


Figure 9. Option-setting memory area

As we can see from Figure 9, the Security MPU registers are allocated in the first sector of the program flash area. The FAW register and OSIS register settings are allocated in the configuration setting area.

With SSP v1.5.3 or lower, we recommend using the method below to set up the Security MPU and FAW registers:

- Use the Synergy configurator to define the Security MPU registers. Section 3.2.1 describes this method.
- Use the SSP flash driver API to configure the FAW register when locking the register settings is not required. Use the factory bootloader to lock the FAW register setting. Section 3.3 describes this method.

Other methods of setting the Security MPU and FAW registers include:

- Use a J-Link script.
- Use a serial port to communicate with the MCU’s factory bootloader and use the factory bootloader’s utilities (section 3.4 describes using this method to clear the locking bit FSPR and hence permanently lock the FAW register setting).
- Self-programming for locking the FAW register (SSP v1.5.3 or earlier does not support this functionality).

3.2 Configuring the Security MPU

This section describes the steps needed to set up and reset the Security MPU registers.

3.2.1 Setting up the Security MPU Registers

Through the Synergy configurator SSP BSP tab, users can define and enable the four secure data regions and the two secure program regions.

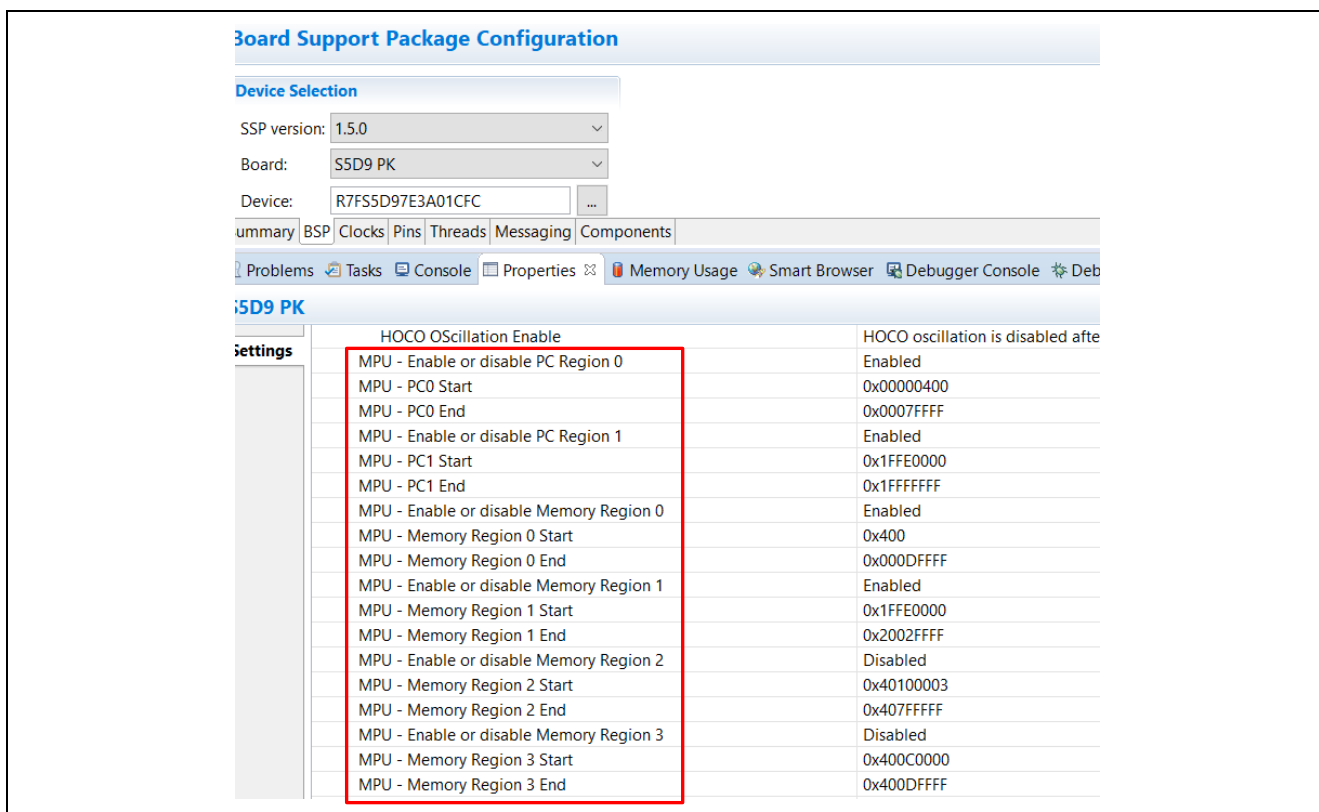


Figure 10. Security MPU Configuration

The following list describes all the regions defined by the above property fields (in bold).

- Secure flash program
 - **PC0 Start** and **PC0 End**: program counter region for the secure flash program
 - **Enable or disable PC Region 0**: enable or disable the secure flash program
- Secure SRAM program
 - **PC1 Start** and **PC1 End**: program counter region for the secure SRAM program
 - **Enable or disable PC Region 1**: enable or disable the secure SRAM program
- Secure flash data region
 - **Memory Region 0 Start** and **Memory Region 0 End**: secure flash data region start and end addresses
 - **Enable or disable Memory Region 0**: enable or disable the secure flash data region
- Secure SRAM data region
 - **Memory Region 1 Start** and **Memory Region 1 End**: secure SRAM data region start and end addresses
 - **Enable or disable Memory Region 1**: enable or disable the secure SRAM data region
- Security Function Region 1
 - **Memory Region 2 Start** and **Memory Region 2 End**: security function 1 start and end addresses (must be the address space for internal peripheral bus 7 or 9, see description below)
 - **Enable or disable Memory Region 2**: enable or disable the security function 1
- Security Function Region 2
 - **Memory Region 3 Start** and **Memory Region 3 End**: security function 2 start and end addresses (must be the address space for internal peripheral bus 7 or 9, see description below)
 - **Enable or disable Memory Region 3**: enable or disable security function 2

See section 4.1 for the operational flow of using the Security MPU in an application project.

3.2.2 Locating Secure Code/Data to a Specific Memory Region

In addition to setting up the Security MPU registers, one must must allocate the secure code and secure data to the intended memory region. This is achieved by customizing the linker script. When you have regions of

secure and non-secure program, the default linker is not sufficient. When designing with the Security MPU, you must define multiple flash and/or SRAM regions in the linker script.

Please refer to section 5.1.2 for an example of locating secure code/data to a specific memory region.

3.2.3 Resetting the Security MPU registers

Resetting the Security MPU registers is possible if the Security MPU registers are located in the FAW region, or if they are located outside the FAW region but the FAW region is not permanently locked down.

Resetting the Security MPU requires a flash erase operation for the first block of the MCU flash. Below are the steps to reset the Security MPU setting with a Synergy e2 studio project:

1. Create an SRAM project to execute from an SRAM region other than SRAMHS
2. Open the flash HAL driver
3. If the Security MPU flash region is not part of the FAW region, clear the FAW register setting first. See section 3.3.2 to understand how to clear the FAW region.
4. Erase the first flash block using the flash HAL driver
5. Close the flash HAL driver

Reference the example e² studio project `secure_data_at_rest_pk_s5d9` described in section 5.2 for the implementation of these steps.

3.3 Configuring the FAW

This section describes the steps to set up and clear the FAW regions, and to permanently lock the FAW setting. The configuration area cannot be written while DTC/DMAC, EDMAC, and LCDC/2DG/JPE are enabled. With SSP v1.5.3 or later, to disable DMAC/DTC, close all modules using DMAC/DTC and call the following code:

To disable DMAC/DTC, close all modules using DMAC/DTC and do the following:

```
ssp_err_t err;
ssp_feature_t ssp_feature = {{{ssp_ip_t} 0U}};
ssp_feature.id = SSP_IP_DTC;
err = R_BSP_ModuleStopAlways(&ssp_feature);
```

Figure 11. Prepare for FAW setup

3.3.1 Setting up the FAW Region

After disabling DTC/DMAC, EDMAC, and LCDC/2DG/JPE, call the following SSP FAW API to set up the FAW:

```
err = g_flash0.p_api->accessWindowSet(g_flash0.p_ctrl, FAW_START, FAW_END);
```

- `g_flash0.p_ctrl` is the instance of this flash HAL driver.
- `FAW_START` is the start address of the FAW window.
- `FAW_END` is the address of the next block acceptable for programming and erasure defined by the access window.

Note: This API always intends to disable the FAW locking. If the FAW is permanently locked prior to running this API, this API will **NOT** function as expected, the FAW region cannot be updated anymore.

3.3.2 Clearing the FAW Regions

After disabling DTC/DMAC, EDMAC, and LCDC/2DG/JPE, call the following SSP FAW API to clear the FAW setting:

```
err = g_flash0.p_api->accessWindowClear(g_flash0.p_ctrl);
```

The `accessWindowClear` API clears the FAW window setting, essentially disabling the FAW control by setting the start address and end address to 0. This routine will work only if the FAW is not permanently locked.

Follow the steps below to clear the FAW setting in the development stage (prior to clear FSPR) in a Synergy e² studio project:

1. Create an SRAM project to execute from an SRAM region other than SRAMHS

2. Open the flash HAL driver
3. Clear the FAW setting using `accessWindowClear`
4. Close the flash HAL driver

3.4 Permanent Locking of the FAW Region

Permanently locking the FAW region prevents a user from updating the FAW region. This step is needed to establish the Write-Once flash region.

There are two use cases when a user may want to consider permanently locking the FAW region:

1. To lock critical runtime data in real time when the device is deployed to the field (the use case without secure bootloader).
2. To lock critical data/code during the manufacturing stage (the use case with secure bootloader)

SSP v1.5.3 and earlier does not support locking the FAW to protect the FAW register from further update. Therefore, the first use case cannot be supported with SSP v1.5.3 and earlier.

For manufacturing usage, to permanently lock the FAW register, we can use the factory bootloader routines, which implement the FAW register setting and clearing procedure. This application project provides an example PC application `final_program_installer` to program the application and permanently lock the FAW.

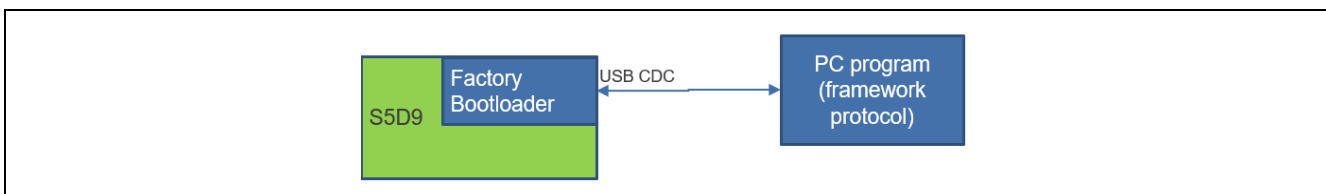


Figure 12. Permanently lock the FAW

3.5 Setting up the Security Control for Debugging

The OCD/Serial Programming ID Setting Register (OSIS) stores the ID code protection for the OCD/serial programmer. Upon connection, an OCD/serial programmer must provide an ID value so that the MCU can determine whether to permit the connection.

Below are the brief rules to set up the ID Code. Please refer to the hardware user manual for details.

- OSIS
 - 128 Bit register holds the ID for authenticating the debugger/programmer
 - SCI/USB boot mode programming also uses this ID code
- Bit[127] = 0 totally disables the debug & factory programming
- Bit[126] = 1 allows the debug & factory programming on ID code match
 - The Flash is erased on connecting in factory programming mode
 - If the FAW window is permanently locked, the flash cannot be erased.
 - If the Security MPU is enabled for the flash block, the Factory Bootloader is disabled.
- Bit[126] = 0 allows debug and factory programming on ID code match

There are multiple ways user can set up the OSIS ID Code:

- Use the Synergy Configurator with the support of the J-Link debugger.
 - Note that J-Link driver V6.34 and higher is required
- Use Renesas Flash Programming Tool (RFP)
 - See the Reference section for the RFP download link
- Use a J-Link script
- Use the factory bootloader
- Self-programming (not currently supported by SSP 1.5.3 or earlier)

3.5.1 Methods of Setting the OSIS ID Code

3.5.1.1 Using the Synergy Configurator and J-Link Debugger

The Synergy configurator allows the user to set up the OSIS ID code through the SSP property setting shown in Figure 13.

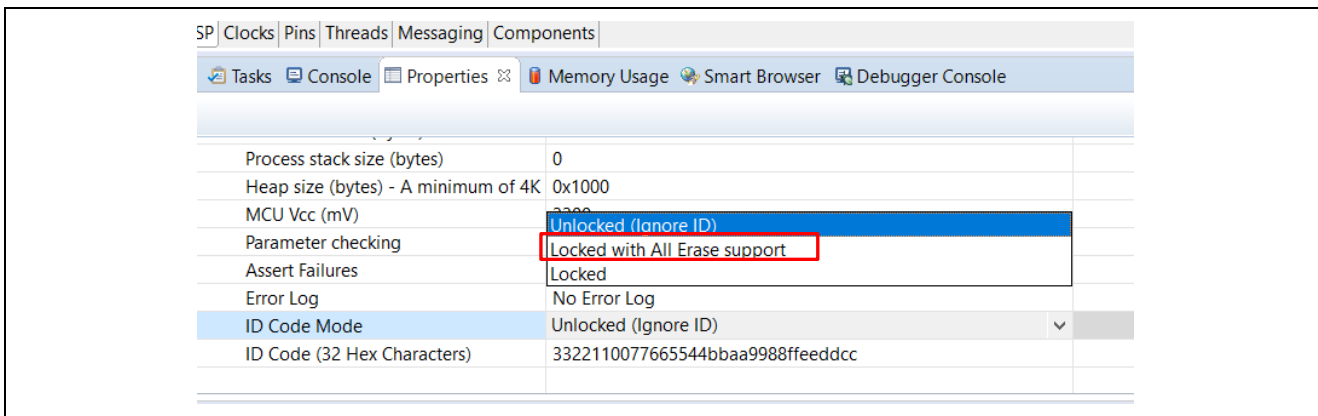


Figure 13. Setting up OSIS ID using Synergy Configurator

- During development stage, it is recommended to choose “Locked with All Erase support” for the **ID Code Mode** selection. Section 5.5 provides an example a J-Link script which erases the entire flash to reset the ID code setting to all FFs.
- When “Locked” is selected for the ID Code Mode, the device will be disabled from debugging and factory programming. Since self-programming is not supported with SSP1.5.3 and earlier, the device will effectively permanently be disabled from debugging and factory programming.

3.5.1.2 Using a J-Link script

The J-Link script `segger` can setup the OSIS ID. This application project does not provide an example script for this usage.

3.5.1.3 Using the factory bootloader

The Renesas SBM is an example of this use case. See the Renesas SBM Application Project.

3.5.2 Method of Setting up the OSIS ID Code for Debugging

With e² studio, a user can set up the ID code using this interface:

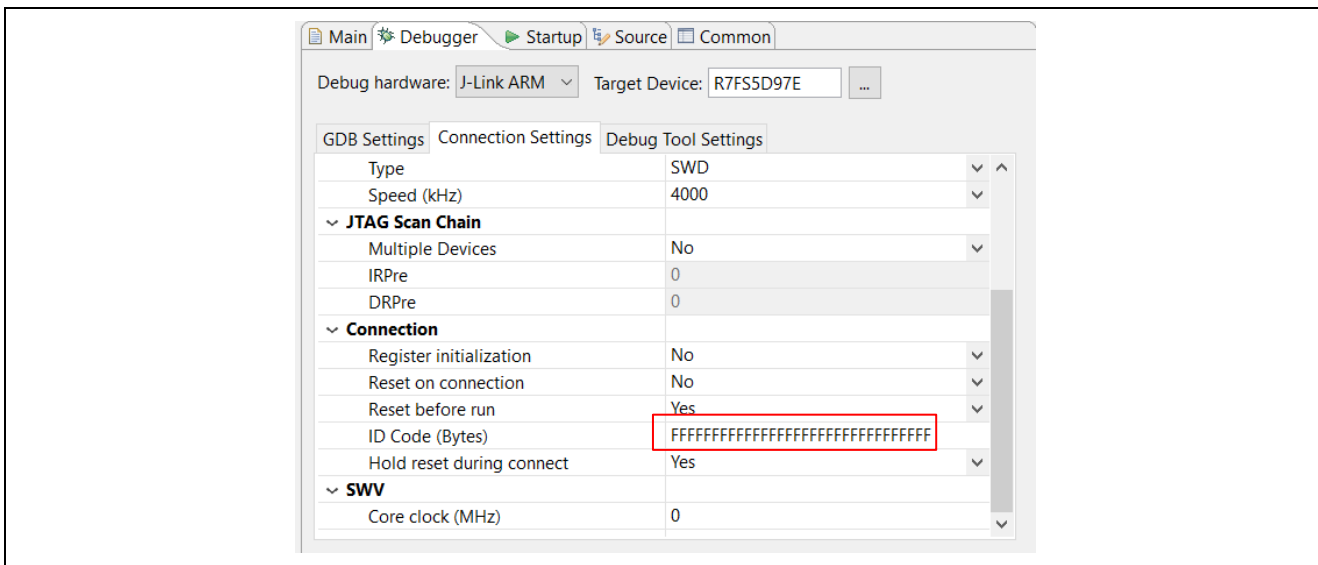


Figure 14. Setting up the ID code with e² studio

With IAR Embedded Workbench for Renesas Synergy, a user can set up the ID code from this interface:

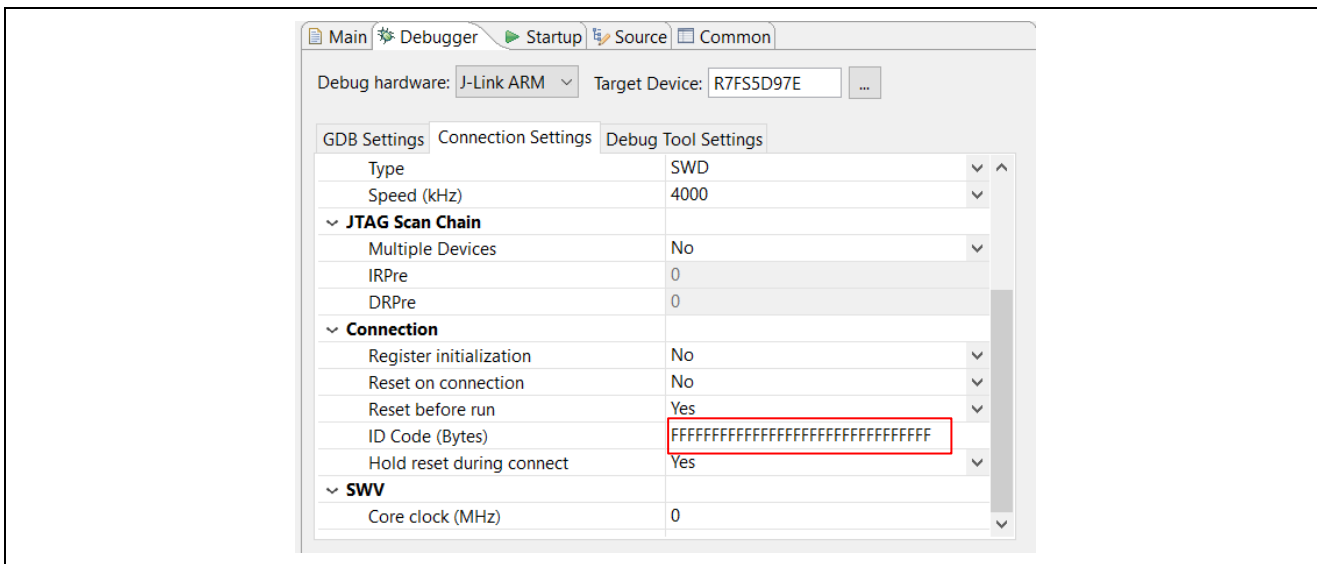


Figure 15. Setting up the ID code with IAR

3.5.3 Method of Resetting the OSIS ID code

Below are the methods of resetting the OSIS ID code:

Using J-Link script

This application project includes an example J-Link script to reset the OSIS ID code by erasing the entire flash. Please reference section 5.5 for details on the operation.

Using factory bootloader

User can use the RFP routine to reset the OSIS ID.

Self-programming

Resetting the OSIS ID code with self-programming will be supported in later SSP releases. User applications should provide this functionality before setting up the intended OSIS ID to fully use this functionality.

4. Operational Flow using Security MPU and FAW

This section describes the operational flow for designing a secure application using the Synergy Security MPU and FAW.

4.1 Internal Flash and SRAM Read Protection

When the Security MPU enables the secure flash and secure SRAM regions, these regions gain read protection:

- Secure SRAM regions are read & write protected from non-secure flash and SRAM programs
- Secure flash regions are read protected from non-secure flash and SRAM programs
- The bus master and debuggers are disabled from access to the secure region

The high-level operational flow of establishing the Security MPU regions is described by the flow diagram in Figure 16. Region 3 and Region 7 in Figure 8 are examples of internal flash read protection.

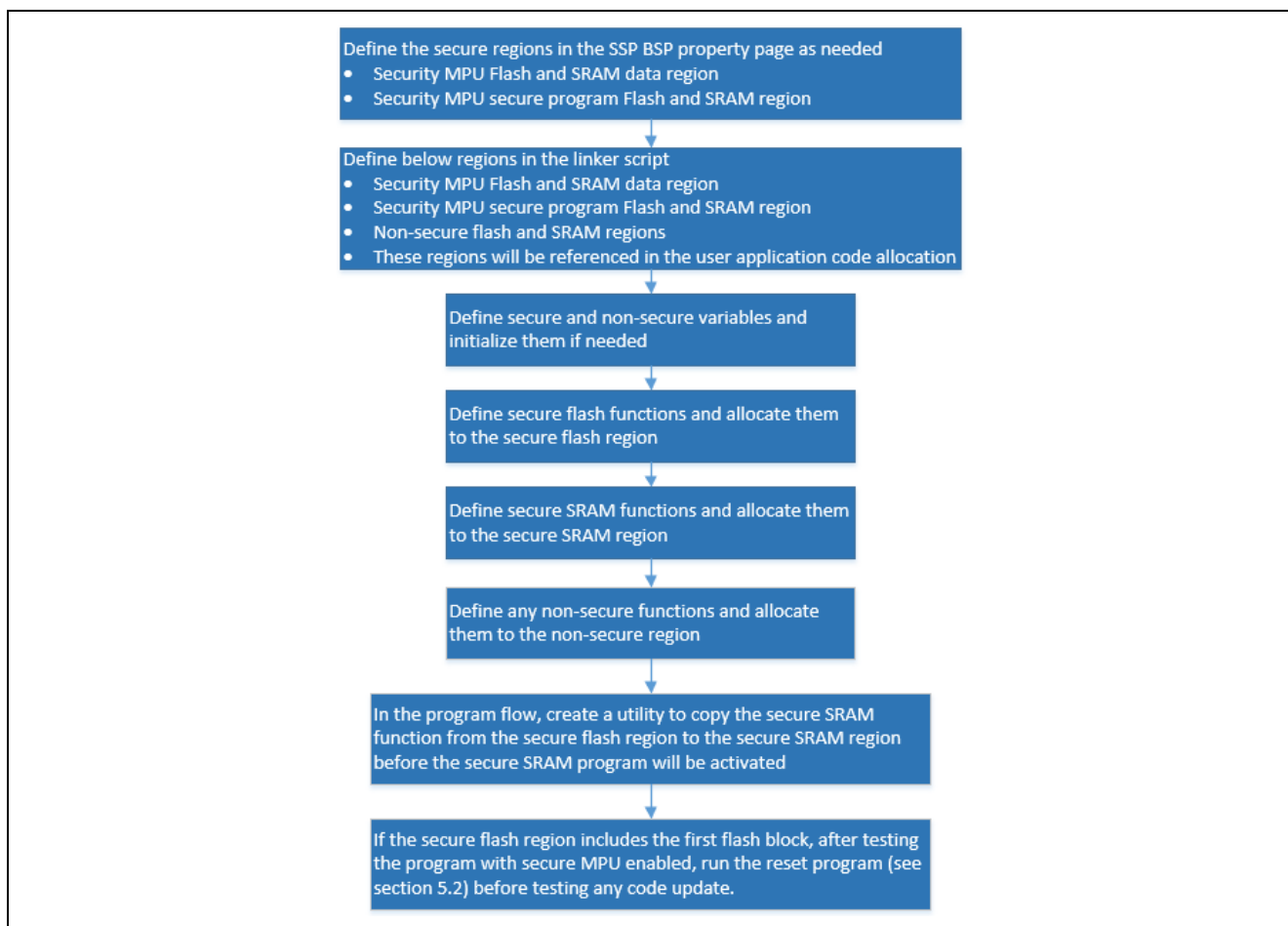


Figure 16. Operational flow for using the Security MPU

The following is a summary of setting up the Security MPU for the MCU:

- Use the configurator BSP tab to define secure regions
- Use the linker script to define the secure regions and non-secure regions
- Establish the secure and non-secure functions
- Copy the secure SRAM function (if defined) from the secure flash region to the secure SRAM region
- Test the system

Note: Once the Security MPU is enabled, the MCU needs to run the reset routine (refer to section 5.2) whenever a new program needs to be programmed on the device.

4.2 Internal Flash Write Protection

As described in section 4.1, the internal SRAM region gains write protection when the Security MPU is enabled for the secure SRAM region. The internal flash region can gain write protection via two methods:

- Set up the FAW region such that it does not include the secure flash region. The entire region (whether it is part of the secure flash region or non-secure flash region) that is not included in the FAW region will have erase and write protection from **secure and non-secure program**. Region 1 in Figure 8 is an example of internal flash write protection.
- When the Security Function for internal peripheral bus 9 is enabled, the entire flash region, including secure and non-secure flash regions, is protected from erasing and programming by **non-secure flash and non-secure SRAM program**.

In this application project, only the first method is described. See section 5.1 for an example of the implementation.

4.2.1 Operational Flow

The SSP API `accessWindowSet` (see section 3.3.1) must be called from the user application to set up the erase and write enabled region. The rest of the flash region will be disabled for erase and write from both secure and non-secure programs.

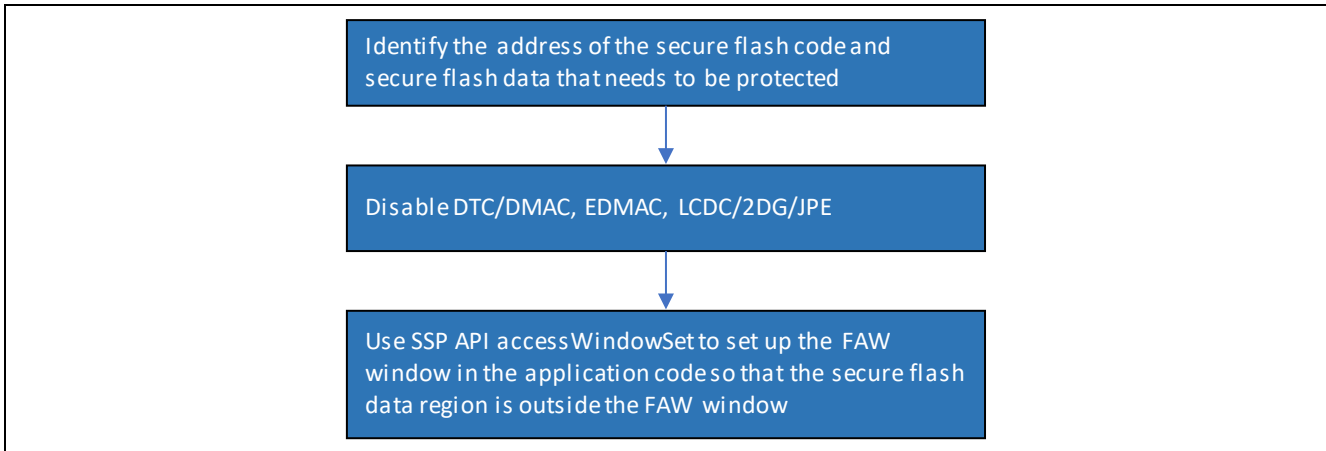


Figure 17. Establish write protected internal flash region

4.3 Internal Flash and SRAM Read/Write Protection

When the Security MPU is enabled for the secure SRAM region, the secure SRAM gains read/write protection from non-secure flash and non-secure SRAM programs.

If the write-protected flash region is part of the secure flash region, this secure flash region is read & write protected from non-secure program and write protected from secure program. Region 4 in Figure 8 is an example of internal flash read and write protection.

4.3.1 Operational Flow

When the internal SRAM region gains read protection by setting up the secure SRAM region, it also gains write protection by the same process. Reference section 4.1 for details on the operational flow.

If the write protected internal flash region is set up via the FAW, and it is also part of the secure flash data region, this section of the secure flash region is erase & write protected from the secure and non-secure programs. The entire secure flash region is read protected from non-secure program.

Reference section 5.1 for an example of the implementation of internal SRAM and flash read & write protection.

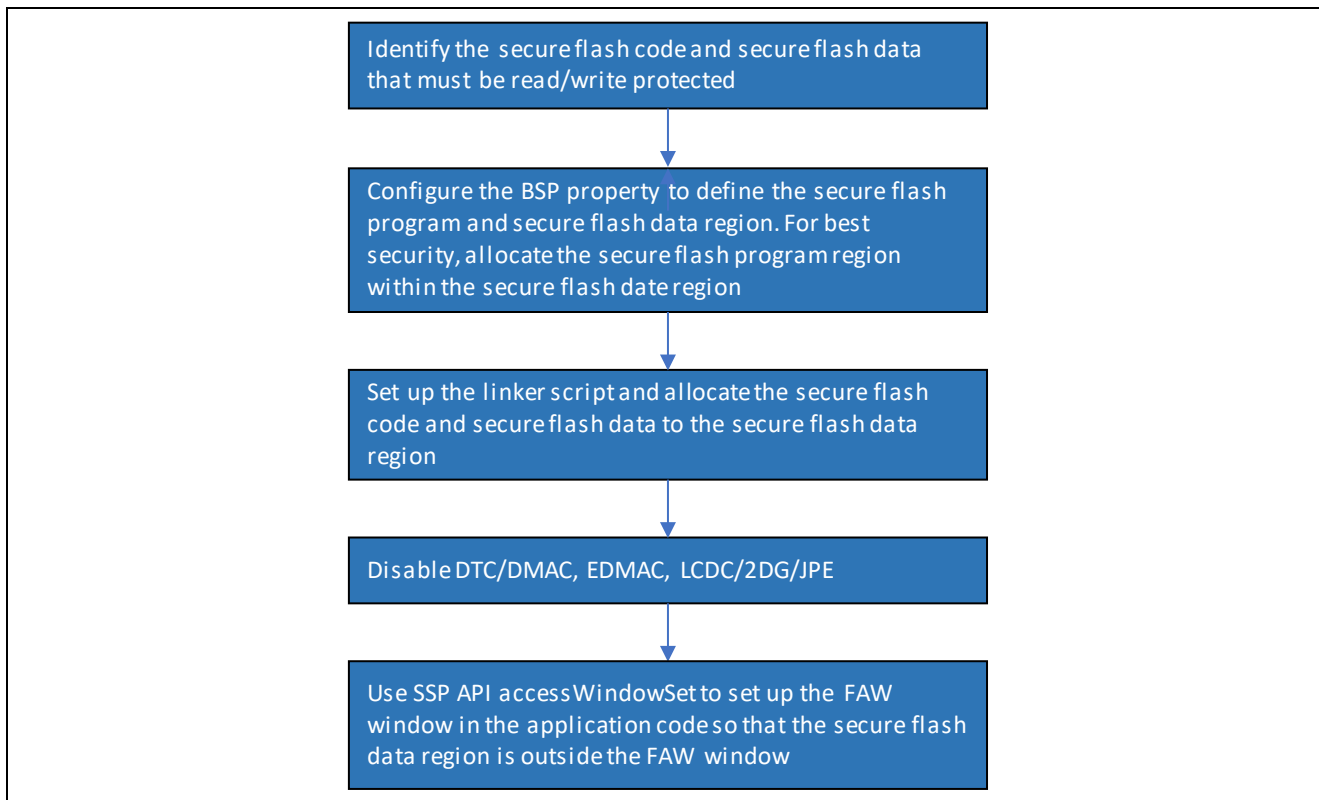


Figure 18. Establish read/write protected internal flash region

4.4 Internal Flash Write-Once Protection

When the FSPR bit is cleared in the FAW register, the internal flash region which is not included in the FAW window becomes write-once protected from secure and non-secure program. The content within this write-once protected region cannot be altered for the lifetime of the device. Region 5 in Figure 8 is an example of internal flash write-once protection.

4.4.1 Operational Flow

Before performing the write-once protection:

- Fully test the critical data and/or code which the application intends to lock down
- Ensure that no other critical data and/or code need to be assigned to the write-once protected region

Figure 19 is the operational flow when operating in a manufacturing stage. Reference section 5.3 for an implementation of the write-once protection.

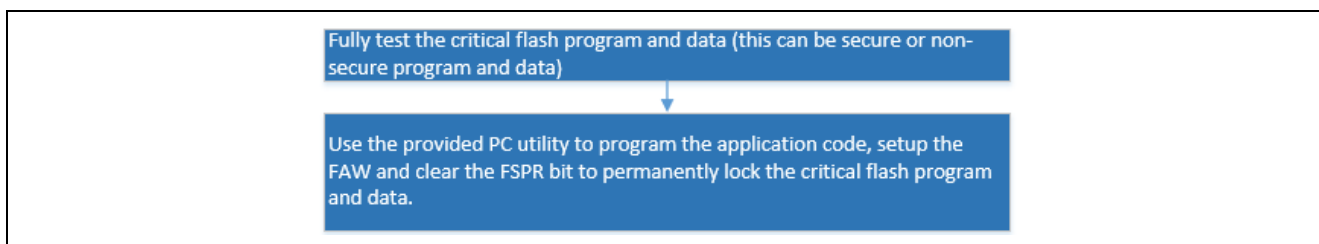


Figure 19. Write-once Protection

As explained in section 3.4, runtime write-once protection by locking the FAW register is not supported by SSP1.5.3 or earlier.

4.5 Internal Flash Write-Once and Read Protection

The write-once and read protection can be achieved for secure flash program and data. When a secure flash region gains read/write protection by setting up the Security MPU and FAW region, a user can permanently lock the FAW window such that the corresponding region can never be erased and programmed by any tools, and it is read protected from non-secure software. Erasing and programming this flash region is not possible via the factory boot loader, J-Link script, or self-programming.

Region 8 in Figure 8 is an example of write-once and read protected internal flash region.

4.5.1 Operational Flow

For internal secure flash applications, a user can follow these operational steps to set up regions that are write-once from secure and non-secure program and read protected from non-secure program:

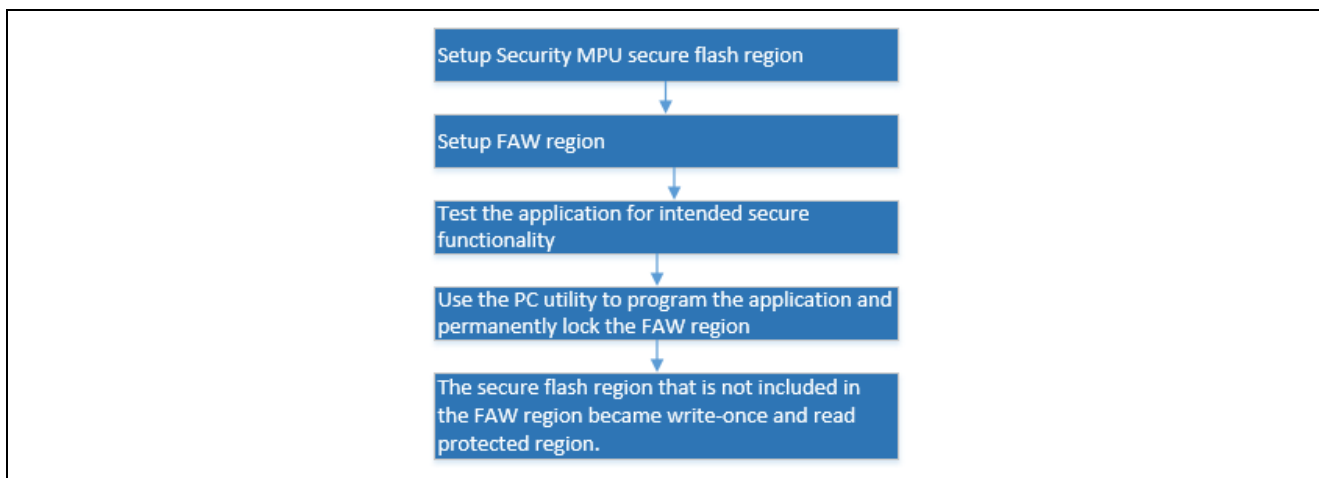


Figure 20. Write-once and Read Protection

Refer to section 5.3 for an example implementation of locking down the FAW region.

4.6 Operation Notes

4.6.1 Memory Allocation

4.6.1.1 Space between Secure and Non-secure Instructions

An address space of greater than 12 bytes is required between the last instruction of a non-secure program and the first instruction of a secure program. If a secure program is stored the area following the non-secure program without the gap, the secure program might be fetched as non-secure program. See hardware user manual for details.

4.6.1.2 Do Not Assign the Memory Mirror Region to the Secure Region

Setting of the memory mirror space (0200 0000h to 027F FFFFh) for MMF is prohibited.

4.6.1.3 Startup Area of Security MPU Regions

The start address area of the secure flash data region cannot be set at the vector table area.

The start address of the secure SRAM data region cannot be set at the stack area or the vector table area.

4.6.2 Limitations on Programming the Option-Setting Memory

Reference the hardware user manual section “Changing the option-setting memory by self-programming” and “Debugging through an OCD or programming by a flash writer” to understand the limitations on programming the Option-Setting Memory for flash access window setup and OSIS ID setup.

The application code included in this application project follows the guidelines provided in the user manual.

4.6.3 Factory Bootloader Accessibility

If the Security MPU is enabled, the factory bootloader is not available anymore. Reset the Security MPU register settings is needed to re-enable the factory bootloader. Refer to section 5.2 for a Security MPU reset example implementation.

4.6.4 Access Secure Function from Non-Secure Functions

Secure functions can be called from non-secure functions (see Figure 6 in 2.2).

- The application project implemented in this release calls the secure function directly from the non-secure function.
- Special considerations are needed when accessing secure functions located in a secure bootloader:
 - Define a group of secure function entry points (function pointers) in the secure flash data region
 - Access the secure function from a function pointer stored in the secure function table

4.6.5 Debugger Access to the Security MPU Regions

When the debugger is operating in debug mode, viewing of the Security MPU flash and secure SRAM region contents are disabled. With e² studio, when the secure region is viewed from the 'memory' view, the values are shown as "0".

5. Security Application e² studio Projects: Internal Flash and SRAM

This section introduces the example projects for using the Security MPU and FAW

- Access secure flash and secure SRAM data from secure flash and SRAM code
- Access secure flash and secure SRAM data from non-secure flash and non-secure SRAM code

There are three example projects included for the internal flash and SRAM secure Data at Rest application:

- Project 1: e² studio Synergy project (`secure_data_at_rest_pk_s5d9.zip`) that demonstrates:
 - Secure flash data read protection
 - Secure and non-secure flash data write protection
 - Secure SRAM read and write protection
- Project 2: e² studio Synergy project (`reset_flash_pk_s5d9.zip`) that demonstrates
 - Reset of the Security MPU
 - Reset of the FAW region
- Project 3: PC-based Visual Studio application (`programInstaller.sln`) that downloads the application created from project 1 and provides the option to permanently lock the FAW via the factory bootloader on the MCU, thus realizing the write-once protection of the regions not included in the FAW.

In addition, there are several example J-Link scripts included in this application project which performs the Security MPU, FAW and OSIS ID code resetting.

Section 5.4 explains a J-Link script that resets the FAW and Security MPU registers before the FAW region is permanently locked down.

Section 5.5 explains the procedures for setting up the OSIS ID with an e² studio auto generated blinky project and the usage of the J-Link scripts to reset the OSIS ID. The reset OSIS ID scripts is used in conjunction of the blinky project to demonstrate an end to end development for debugging security handling with the Synergy MCUs using SSP1.5.3.

5.1 Project 1 e² studio project: Internal Flash and SRAM Read Write Protection

5.1.1 Software Architecture Overview

Project 1 project implements four groups of the software components that demonstrate the flash and SRAM read and write protection.

- Console Interface
- Secure flash code
- Secure SRAM code
- Non-secure flash code

There is one console thread in this application software using the USB CDC console framework from SSP. The console thread runs in the non-secure flash region. It monitors user input from the COM terminal and activates the corresponding secure flash/SRAM and non-secure flash application code. See Figure 21.

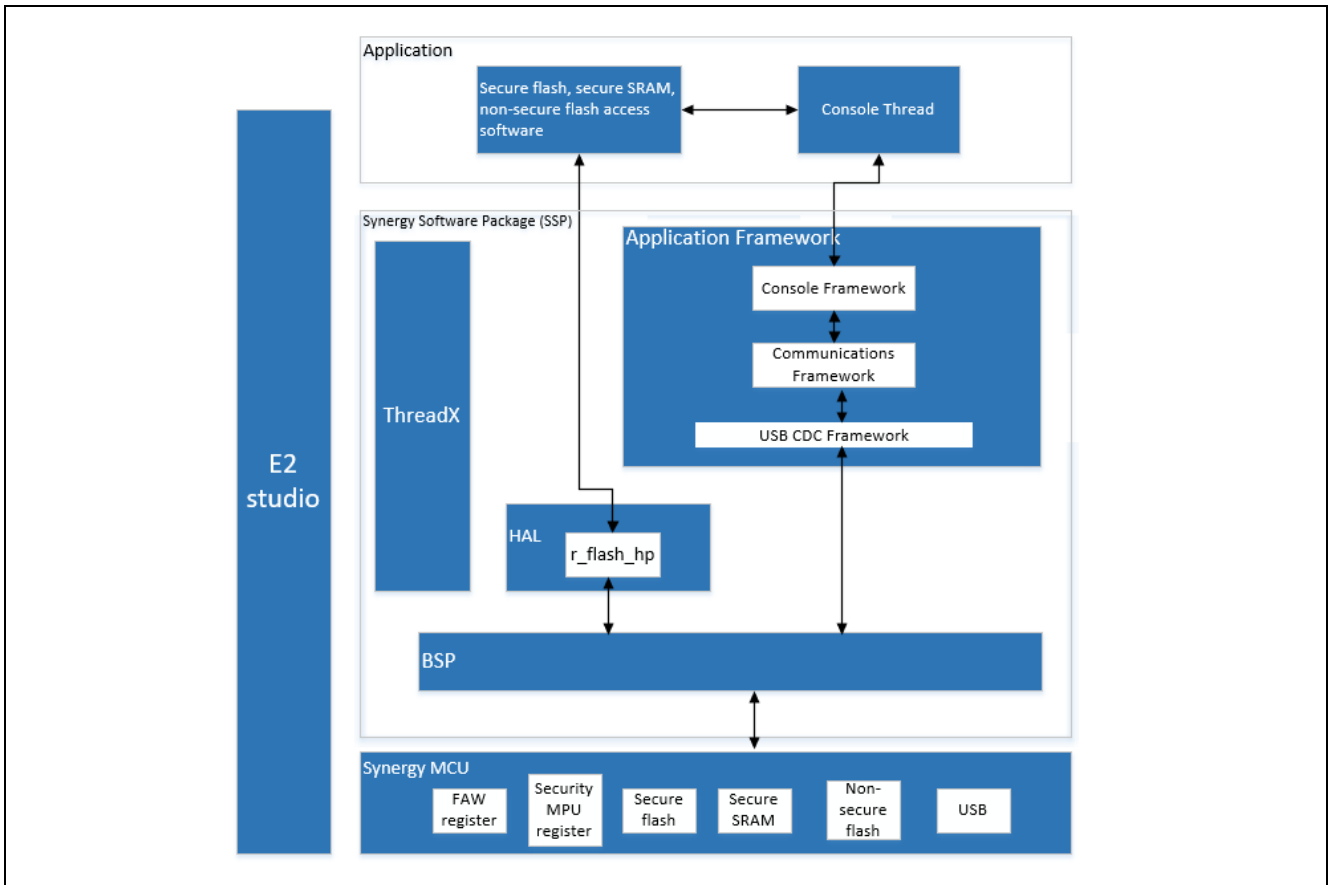


Figure 21. Project 1 Software Block Diagram

5.1.2 Memory Allocation Arrangement

The memory map of the system is controlled by the linker script `\script\S5D9.ld`. The software uses macros defined in `secure_definitions.h` to allocate the different secure and non-secure code.

Figure 22 shows the flash memory allocation based on the flash software components.

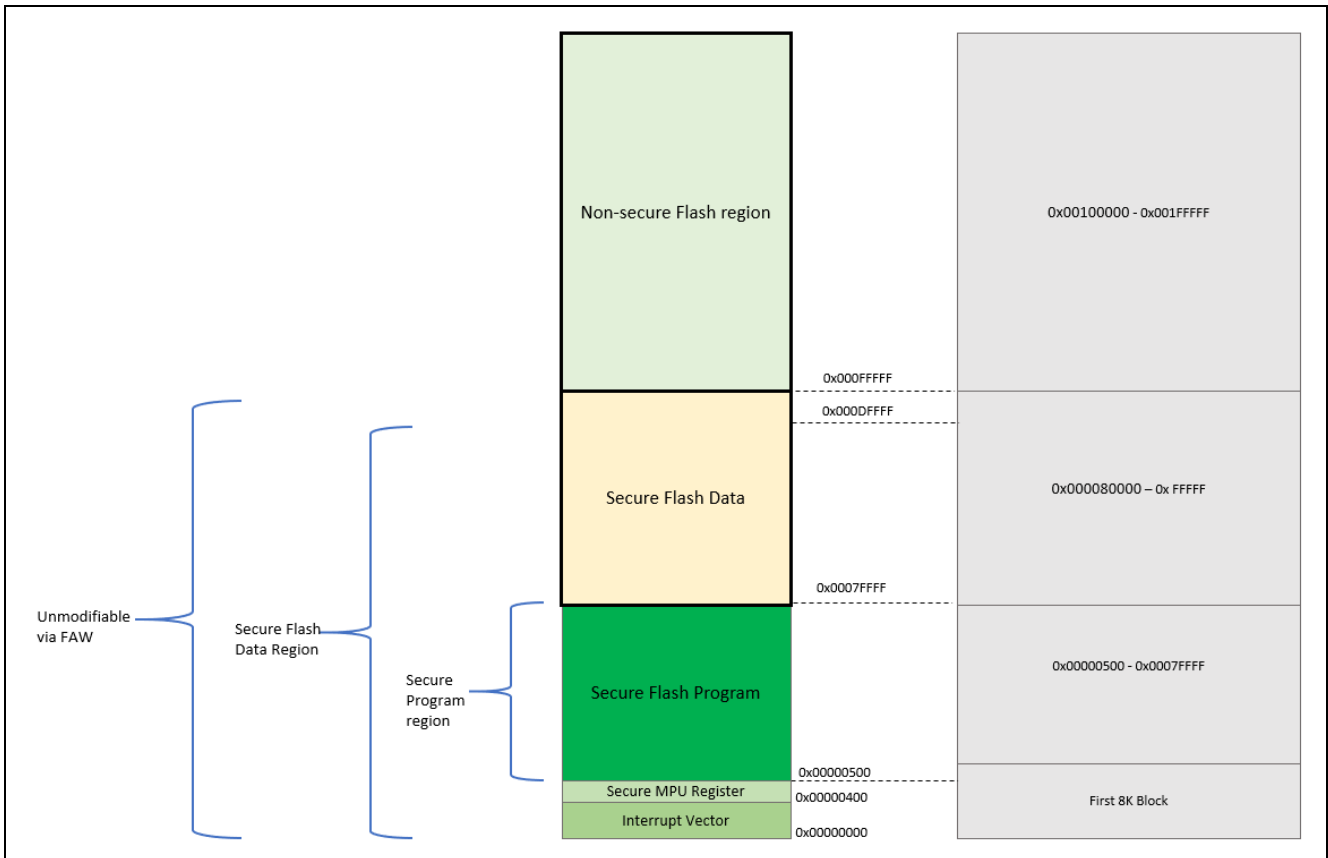


Figure 22. Project 1 Internal Flash

Figure 23 shows the SRAM memory allocation based on the different SRAM software components.

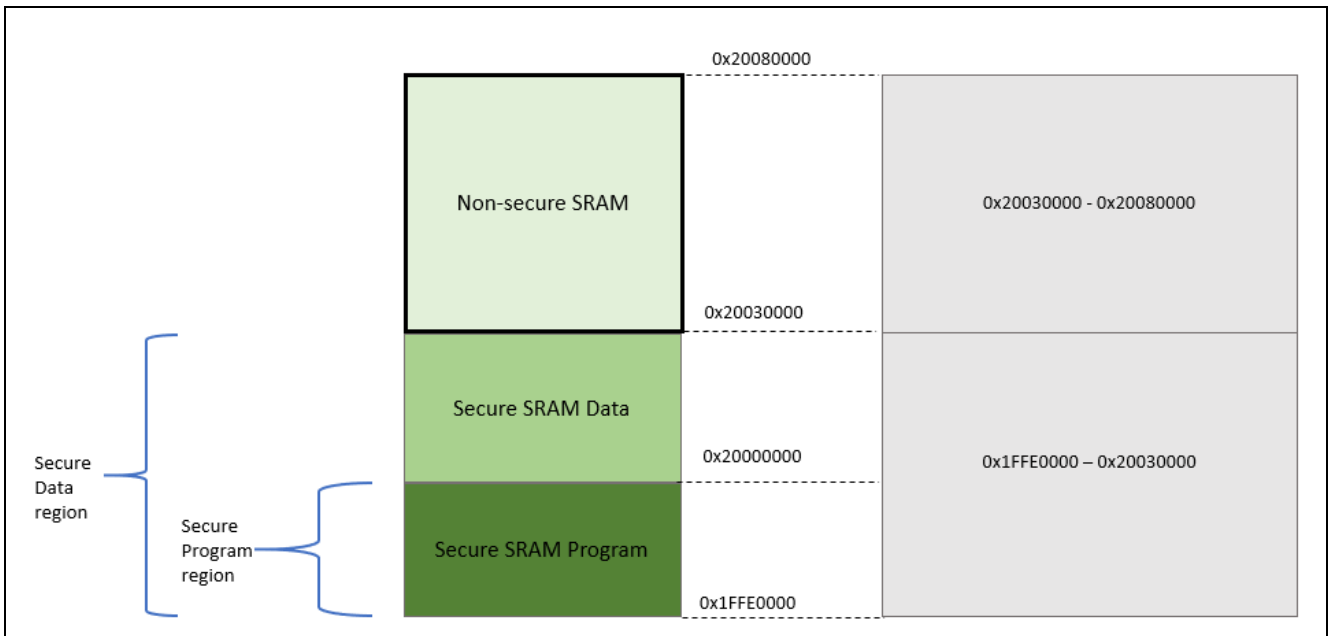


Figure 23. Project 1 Internal SRAM

Below are the memory regions defined by the linker script (\script\S5D9.ld)

```

/* Linker script to configure memory regions. */
MEMORY
{
  VECTOR_TABLE (rx) : ORIGIN = 0x00000000, LENGTH = 0x400 /* vector table, 1024 bytes */
  SECURE_PROGRAM (rx) : ORIGIN = 0x00000400, LENGTH = 0x007FC00 /* secure flash program*/
  SECURE_DATA (rx) : ORIGIN = 0x00008000, LENGTH = 0x0080000 /* secure flash data */
  FLASH (rx) : ORIGIN = 0x00100000, LENGTH = 0x0100000 /* non-secure flash code */
}

```

```

SECURE_RAM_PROGRAM (rwx)  : ORIGIN = 0x1FFE0000, LENGTH = 0x0020000 /* secure SRAM program */
SECURE_RAM (rwx)         : ORIGIN = 0x20000000, LENGTH = 0x0030000 /* secure SRAM data */
RAM (rwx)                : ORIGIN = 0x20030000, LENGTH = 0x0050000 /* non-secure SRAM program and data
*/
DATA_FLASH (rx)         : ORIGIN = 0x40100000, LENGTH = 0x0010000 /* 64K */
QSPI_FLASH (rx)        : ORIGIN = 0x60000000, LENGTH = 0x4000000 /* 64M, Change in QSPI section
below also */
SDRAM (rwx)            : ORIGIN = 0x90000000, LENGTH = 0x2000000 /* 32M */
}

```

Below are the memory assignment macros used by the secure software and data sections defined by `secure_definitions.h`:

```

#define SECURE_PROGRAM __attribute__((section (".secure_text")))
#define SECURE_CONST __attribute__((section (".secure_rodata")))
#define SECURE_DATA __attribute__((section (".secure_data")))
#define SECURE_BSS __attribute__((section (".secure_bss")))
#define SECURE_SRAM_PROGRAM __attribute__((section (".secure_sram_program")))

```

Below are the test variables that are used to demonstrate the protection on secure flash data and secure SRAM data.

Global variables are used to allow read and write directly from non-secure program to simulate reading and writing to the secure data area.

Table 2. Test Variables

Data Type	Global Variable Name	Comment for data type	Comment on functionality
Secure flash data example	<code>s_dataConst</code>	Constant	No access function is provided. Access to this variable from non-secure program is not valid.
Secure initialized SRAM data example	<code>s_dataInit</code>	Initialized	No access function is provided. Access to this variable from non-secure program is not valid.
Secure uninitialized SRAM data example	<code>s_dataWritten</code>	Uninitialized	Write to this variable from non-secure flash program and show the write is not valid.

5.1.3 Functionality Description

(a) Console user interface

This section of code controls the CLI input and output.

(b) Secure flash program

- Read and write from the secure flash program to the secure SRAM and non-secure SRAM region
— Show access to all SRAM regions are granted
- Read from the secure flash program to the secure flash data region and non-secure flash data region
— Show access to all flash regions are granted

(c) Secure SRAM program

- Read and write from the secure SRAM program to the secure SRAM and non-secure SRAM region
— Show access to all SRAM regions are granted
- Read and write from the secure SRAM program to the secure SRAM and non-secure SRAM region
— Show access to all flash regions are granted

(d) Non-secure flash program

- Read and write from the non-secure flash program to the secure SRAM and non-secure SRAM region
 - Show access to the non-secure SRAM region is granted
 - Show access to the secure SRAM region is not granted
- Read and write from the non-secure flash program to the secure and non-secure flash region
 - Show that before the FAW is set up, write access to all flash regions is granted. This is an example of internal flash read protection (see section 4.1)
 - Show that after the FAW is set up, write access to the non-FAW region is not granted. This is an example of internal flash write protection (see section 4.2).
 - If this non-FAW region is part of the Security MPU region, this is an example of internal flash read & write protection (see section 4.3)
 - If we lock the FAW setting (via clearing the FSPR bit):
 - The above flash write-protected region becomes a write-once region (see section 4.4)
 - The above flash read/write protected region becomes a write-once and read protected region (see section 4.5)

Special Note on the macro definition for `SECURE_PADDING`: This macro inserts the gap between secure and non-secure code execution as requested by the Security MPU architecture as described in section 4.6.1.1.

Figure 24 shows the source files grouped by functionality.

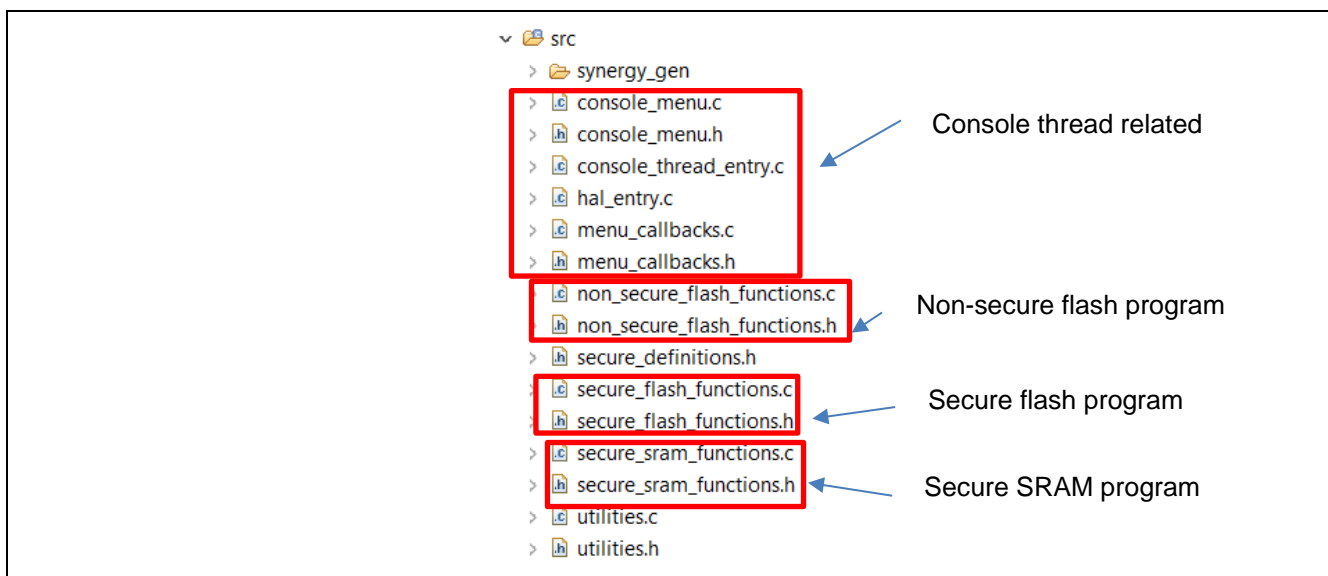


Figure 24. Source code

5.1.4 Establishing and Running Software from Secure SRAM Region

Running secure programs from the secure SRAM region involves the following actions:

- Set up the secure SRAM region from the linker script, as shown in section 5.1.2
- Relocate the secure SRAM application code (`.secure_sram_program` region) from the secure flash region to the secure SRAM region in the linker script

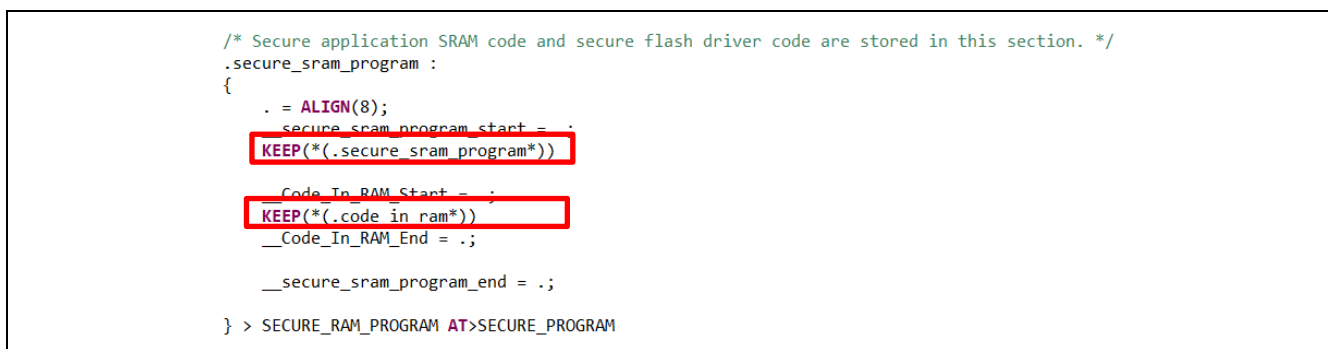


Figure 25. Allocate code to secure SRAM

- Copy the secure SRAM code from the secure flash region to the secure SRAM region using application code. Refer to the `secure_sram_section_copy` function (in `secure_flash_functions.c`) for implementation
 - The `.code_in_ram` SSP code section relocates the flash writing routine to RAM. In this application project, we allocated this section of the code in the secure SRAM section as shown above.

5.1.5 Importing and Building the Project

See the *Renesas Synergy™ Project Import Guide* (r11an0023eu0121-synergy-ssp-import-guide.pdf), included in this package, for instructions on how to import the `secure_data_at_rest_pk_s5d9` project into e² studio, build, and run the project.

This project expects the USB CDC Driver, <https://www.renesas.com/us/en/products/synergy/software/add-ons/usb-cdc-drivers.html>. Make sure to download and install it.

5.1.6 Hardware Setup

1. Connect J5 using the micro USB cable to the workstation to provide power and debugging capability using the on-board debugger.
2. Connect J19 using the micro USB cable to the workstation to provide communication with the terminal program (for example, Tera Term)

5.1.7 Verifying the Secure Functionalities

Open the terminal program. The following uses Tera Term as an example. Connect to the Synergy USB Communications port.

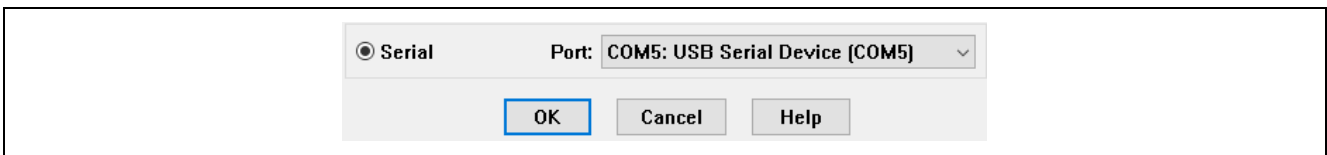


Figure 26. Communication through USB CDC Console

Step 1: View the available main menu items.

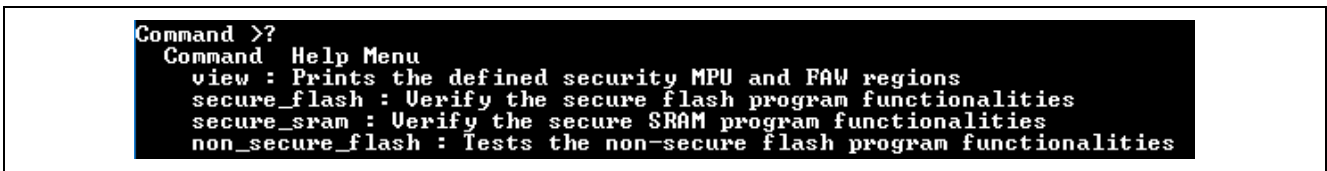


Figure 27. Main menu

Step 2: View the default Security MPU and FAW settings (Function `read_secure_settings` performs this task).



Figure 28. Security setting

Step 3: View the secure flash functions (functions running in secure flash region), reading and writing to various memory regions using secure flash functions.

```

Command >secure_flash
Secure flash code >?
Secure flash code Help Menu
~ : Back to root menu
^ : Up one menu level
read : Read secure & non_secure areas from secure code
write : Write secure & non_secure areas from secure code
setup_faw : Setup the FAW from secure code
reset_faw : Reset the FAW from secure code

```

Figure 29. Secure flash program operation menu

Reading the secure flash, secure SRAM, non-secure flash, non-secure SRAM region from the secure flash functions and confirm that read accesses are granted. Function `secure_code_read()` verifies this functionality.

```

Command >secure_flash
Secure flash code >read
*****

Secure code reads...running from secure flash

PASS! secure flash program can read secure flash
PASS! secure flash program can read secure ram
PASS! secure flash program can read non-secure flash
PASS! secure flash program can read non-secure ram

```

Figure 30. Secure flash program read

Perform writing of the secure SRAM, non-secure SRAM regions from the secure flash functions and confirm the write accesses are granted. Function `secure_code_write()` performs this function.

```

Secure flash code >write
*****

Secure code writes...running from secure flash

PASS! secure flash program can write secure sram data
PASS! secure flash program can write non-secure sram data

```

Figure 31. Secure flash program writes

Set up the FAW using the `setup_faw` command. Figure 32 shows the default FAW region set up by the example project. Return to main menu to confirm FAW is set up.

```

Secure flash code >setup_faw
*****

Setup Flash Access Window...running from flash

PASS secure program => faw is setup from 0x100000 to 0x1FFFFFF
*****

Secure flash code >~
Command >view
*****

Security MPU settings
PC Region 0:          enabled [0x    500 : 0x   7FFFF]
PC Region 1:          enabled [0x1FFE0000 : 0x1FFFFFFF]
Secure Region 0:      enabled [0x    400 : 0x   DFFFF]
Secure Region 1:      enabled [0x1FFE0000 : 0x2002FFFF]
Secure Region 2:      disabled
Secure Region 3:      disabled

Access Window Setting Register (AWS)
Access Window:        enabled [0x 100000 : 0x 200000]

```

Figure 32. Set up the FAW (from 0x100000 to 0x1FFFFFF)

Reset the FAW using the `reset_faw` command from secure flash program and return to main menu to confirm the FAW region is disabled.

```

Command >secure_flash
Secure flash code >reset_faw
*****

Reset Flash Access Window...running from flash

PASS secure program => faw is reset

*****
Secure flash code >~
Command >view
*****
Security MPU settings
PC Region 0:      enabled [0x   500 : 0x  7FFFF]
PC Region 1:      enabled [0x1FFE0000 : 0x1FFFFFFF]
Secure Region 0:  enabled [0x   400 : 0x  DFFFF]
Secure Region 1:  enabled [0x1FFE0000 : 0x2002FFFF]
Secure Region 2:  disabled
Secure Region 3:  disabled

Access Window Setting Register <AWS>
Access Window:    disabled

```

Figure 33. Reset the FAW

Step 4: View the secure SRAM functions (functions running out of the secure SRAM region), perform reading and writing to various memory regions using secure SRAM functions:

```

Command >secure_sram
Secure sram code >?
Secure sram code Help Menu
^ : Back to root menu
~ : Up one menu level
read : Read secure & non_secure areas from secure sram code
write : Write secure & non_secure areas from secure sram code

```

Figure 34. Secure SRAM program

Read the secure flash, secure SRAM, non-secure flash, non-secure SRAM region from the secure SRAM functions and confirm read accesses are granted. Function `secure_sram_code_read` performs this function.

```

Secure sram code >read
*****

Secure sram code reads...running from secure sram

PASS! secure sram program can read secure flash
PASS! secure sram program can read secure ram
PASS! secure sram program can read non-secure flash
PASS! secure sram program can read non-secure ram

```

Figure 35. Secure SRAM program read

Write to the secure flash, secure SRAM, non-secure flash, non-secure SRAM regions from the secure SRAM functions **with FAW region enabled** from 0x100000 to 0x1FFFFFF. Function `secure_sram_code_write` performs this function.

```

Secure sram code >write
*****

Secure sram code writes...running from secure sram
Access Window:      enabled [0x 100000 : 0x 200000]
PASS! secure sram program can write secure sram data
PASS! secure sram program can write non-secure sram data
PASS! secure sram program cannot write to secure flash region 0xb0000
      which is unmodifiable by FAW
PASS! secure sram program can write to non-secure flash 0x120000
      which is modifiable by FAW

```

Figure 36 Secure SRAM program write

Return to the main CLI menu:

```

*****
Secure sram code >~
Command >

```

Figure 37. Return to main menu

Step 5: View the non-secure flash functions (functions running out of non-secure flash region), read and write to various memory regions using non-secure flash functions:

```

Command >non_secure_flash
Non-secure flash code >?
Non-secure flash code Help Menu
~ : Back to root menu
^ : Up one menu level
read : Read secure & non_secure areas from non_secure code
write : Write secure & non_secure areas from non_secure code

```

Figure 38. Non-secure flash program operations

Read the secure flash, secure SRAM, non-secure flash, non-secure SRAM region from the non-secure flash functions and confirm that reading secure memory is not valid. Function `non_secure_code_read` performs this function.

```

Non-secure flash code >read
*****
non-secure code reads...running from non-secure flash
PASS! non-secure program cannot read secure flash
PASS! non-secure program cannot read secure sram
PASS! non-secure program can read non-secure flash
PASS! non-secure program can read non-secure sram

```

Figure 39. Read operation from non-secure flash program

Write the secure flash, secure SRAM, non-secure flash, non-secure SRAM regions from non-secure flash program **with FAW region enabled** from 0x100000 to 0x1FFFFFF and show that secure flash memory should be protected with FAW to prevent non-secure code writing. Function `non_secure_code_write` performs this function.

```

Non-secure flash code >write
*****
non-secure flash code tests...running from non-secure flash
Access Window:          enabled [0x 100000 : 0x 200000]
PASS! non-secure flash program cannot write secure sram data
PASS! non-secure flash program can write to non-secure sram data
PASS! non-secure flash program cannot write to secure flash region 0xb0000
      which is unmodifiable by FAW
PASS! non-secure flash program can write to non-secure flash region 0x120000
      which is modifiable by FAW

```

Figure 40. Write operation from non-secure flash program

Step 6: Return to the secure function menu to clear the FAW setting and verify the operation using the `view` command:

```
Secure flash code >reset_faw
*****
Reset Flash Access Window...running from flash
PASS secure program => faw is reset
*****
Secure flash code >~
Command >view
*****
Security MPU settings
PC Region 0:      enabled [0x 500 : 0x 7FFFF]
PC Region 1:      enabled [0x1FFE0000 : 0x1FFFFFFF]
Secure Region 0:  enabled [0x 400 : 0x DFFFF]
Secure Region 1:  enabled [0x1FFE0000 : 0x2002FFFF]
Secure Region 2:  disabled
Secure Region 3:  disabled

Access Window Setting Register <AWS>
Access Window:    disabled
```

Figure 41. Confirm FAW is reset

5.1.7.1 Summary of the functionalities provided in this application project:

- Establish read/write protection to the secure flash and secure SRAM regions using Security MPU and FAW setting.
- Shield secure data from the debugger. For demonstration, try to view the memory of the secure data region from the debugger, value “0” is presented. The protection from debugger is effective after first power recycle.

Address	0 - 3	4 - 7	8 - B	C - F
000000000000400	00000000	00000000	00000000	00000000
000000000000410	00000000	00000000	00000000	00000000
000000000000420	00000000	00000000	00000000	00000000
000000000000430	00000000	00000000	00000000	00000000
000000000000440	00000000	00000000	00000000	00000000
000000000000450	00000000	00000000	00000000	00000000
000000000000460	00000000	00000000	00000000	00000000
000000000000470	00000000	00000000	00000000	00000000
000000000000480	00000000	00000000	00000000	00000000
000000000000490	00000000	00000000	00000000	00000000
0000000000004A0	00000000	00000000	00000000	00000000
0000000000004B0	00000000	00000000	00000000	00000000
0000000000004C0	00000000	00000000	00000000	00000000
0000000000004D0	00000000	00000000	00000000	00000000
0000000000004E0	00000000	00000000	00000000	00000000
0000000000004F0	00000000	00000000	00000000	00000000
000000000000500	00000000	00000000	00000000	00000000
000000000000510	00000000	00000000	00000000	00000000
000000000000520	00000000	00000000	00000000	00000000

Figure 42. Secure data region is not viewable from debugger

- Establish a blueprint of how to implement secure flash and secure SRAM regions with read & write protection of the secure SRAM and secure flash regions.

5.1.8 Migrating to other Synergy MCUs

As each MCU group and each MCU within a group may have different internal flash and SRAM size. Following below steps to migrate this application.

1. Modify the BSP Security MPU region settings to fit the memory size
2. Modify the linker script to take care of the new secure flash and SRAM regions defined in the BSP tab.
3. Modify the FAW region range if needed in the application code. As the application code size is fairly small, Other application code does not need modification in most cases.

5.2 Project 2: e² studio Project: Reset the Security MPU and FAW setting

This e² studio project `reset_flash_pk_s5d9` provides an example to reset the Security MPU and FAW setting using the SSP `r_flash_hp` HAL driver. The user is advised to run this routine every time they update a project with the Security MPU enabled or load a new project.

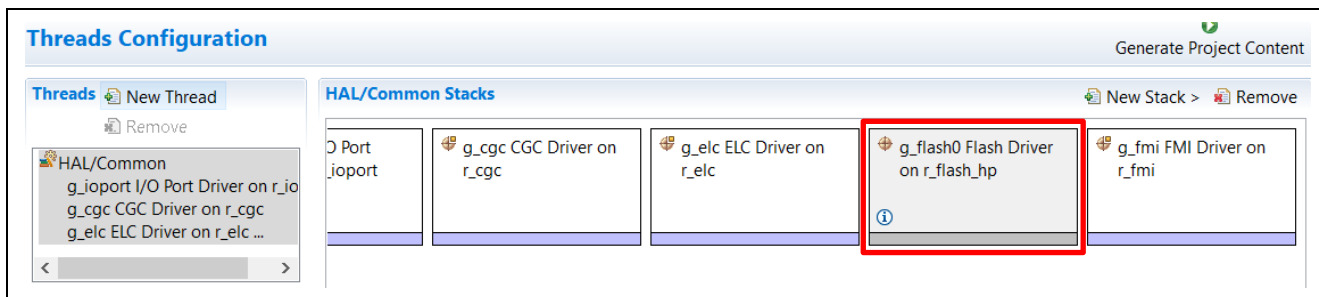


Figure 43. Flash Driver on `r_flash_hp`

Before the flash driver can write to the configuration area, you must close the DTC/DMAC, EDMAC, LCDC/2DG/JPEG usage and call the code as described in Figure 11. Prepare for FAW setup

. After these preparations, call the routines in Figure 44 to clear the FAW region setting.

```
static volatile ssp_err_t err;
SSP_PARAMETER_NOT_USED(err);

//Configure and open the Flash API
err = g_flash0.p_api->open(g_flash0.p_ctrl, g_flash0.p_cfg);
APP_ERR_TRAP(err);

//Remove any prior access window
err = g_flash0.p_api->accessWindowClear(g_flash0.p_ctrl);
APP_ERR_TRAP(err);

//erase SECMPUAC register setting
err = g_flash0.p_api->erase(g_flash0.p_ctrl, secureMPU_reg, 1);
APP_ERR_TRAP(err);

//close flash API
err = g_flash0.p_api->close(g_flash0.p_ctrl);
APP_ERR_TRAP(err);
```

Figure 44. Resetting Security MPU and FAW settings

5.3 Project 3: PC Application to Permanently Lock the FAW

The workstation utility that permanently locks the FAW uses the MCU's factory loader as shown in Figure 12. For PK-S5D9, to enter factory bootloader mode, change the Jumper J1 position from 1-2 to position 2-3.

Normal Boot: If J1 is in **position 1-2**, the S5D9 starts execution from internal flash (ROM). This is the normal mode of operation described above.

Factory Boot Loader Mode: If J1 is in **position 2-3**, the S5D9 boots with the factory bootloader in USB program mode, which enables you to load a program directly to the internal microcontroller flash through the USB device interface (CDC Class).

To use this tool, follow the steps below:

Step 1: Exercise the default evaluation content which programs the default program to pk-s5d9. The default program `installer.exe` under `\PC program\final_program_installer_script` does not lock down the FAW. It does set up the FAW region.

1. Put the Synergy MCU in factory bootloader mode.
2. Open the device manager and find the enumerated COM port number for "Synergy USB Boot".
3. Run the `deploy_app.bat` file and provide the enumerated COM port number as the argument. See example output in Figure 45:

```
C:\ PC program\final_program_installer_script>deploy_app.bat 12

C:\Synergy_SW\iotsg_repository\SSP Applications\Secure Data at Rest\Phase 1\PC
program\final_program_installer_script>REM script to download application and setup FAW

C:\Synergy_SW\iotsg_repository\SSP Applications\Secure Data at Rest\Phase 1\PC
program\final_program_installer_script>echo off
Parsed SREC file: secure_data_at_rest_pk_s5d9.srec (0x0..0x10D477)
Initialising flash programming connection to port COM12
Established flash programming connection
Disabling flash access window (0x0..0xFFFFFFFF)
Flash access window is now disabled
Erasing flash
Flash erased OK
Writing flash for secure_data_at_rest_pk_s5d9.srec region (bufferOffset=0x0, 0x10D440
bytes)
Flash region written OK
Flash Access Window configured for start=0x100000, end=0x1fffff
Completed OK
```

Figure 45. Program the application without locking the FAW region

Note: Every time before reprogramming the application, the reset routine must be performed successfully.

Step 2: Proceed with locking the FAW region if needed by following steps:

1. Open the Visual Studio project program_installer.sln (\PC program\final_program_installer_src\pc\apps\programInstaller\programInstaller.sln).
2. Open the project properties. Under C/C++>Preprocessor, add a Preprocessor definition for `ACTIVATE_THE_FSPR`.

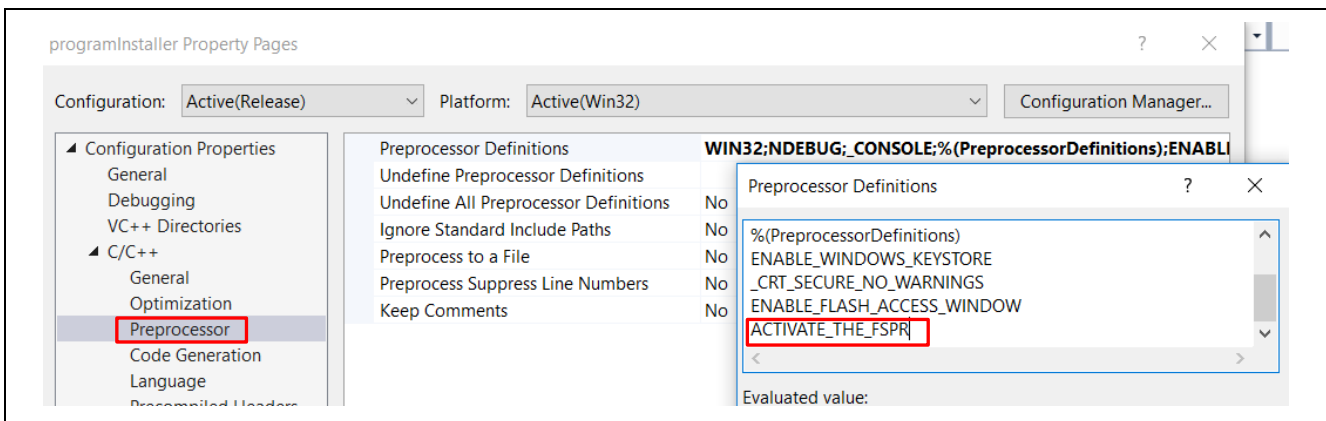


Figure 46. Set up the FAW locking

3. Compile Program_Installer project.
If you are using Visual Studio 2017 professional or enterprise version, you will run into following error message.

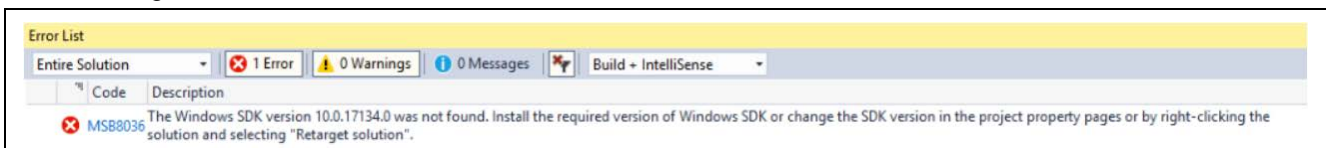


Figure 47. Compiling the Program Installer

To resolve this error message, right-click on the project and select **Retarget Projects**. Then choose the SDK version you are using. Proceed to compiling.

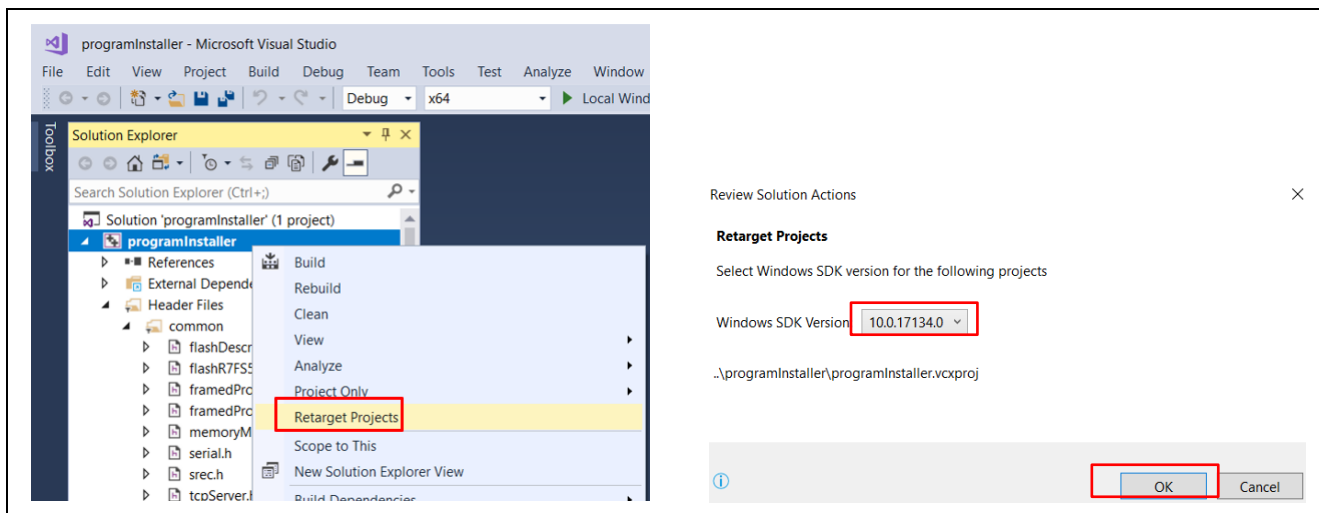


Figure 48. Zip File Contents

4. Compile the application Project 1 `secure_data_at_rest_pk_s5d9` with ID code region removed. This is a limitation of this tool. If you do not change `secure_data_at_rest_pk_s5d9`, there is no need to take this action as there is a copy of the default `.srec` file included in the script folder.
5. Put the generated `secure_data_at_rest_pk_s5d9.srec` in the same folder as the `deploy_app.bat` file (`\PC program\final_program_installer_script`).
6. Proceed to use the same steps from 1 to 3 in Step 1 in this section to program the final application.

Warning: The unmodifiable flash region will be permanently locked down with the current image (from 0x0 to 0xFFFFF).

```
Writing flash for secure_data_at_rest_pk_s5d9.srec region (bufferOffset=0x0, 0x10D440 bytes)
Flash region written OK
Flash Access Window configured for start=0x100000, end=0x1fffff
Flash access window is now LOCKED
Completed OK
```

Figure 49. Lock the FAW region

5.4 Example Reset J-Link Script for the Security MPU and FAW

Unzip `Reset_Scripts.zip` and locate the Security MPU and FAW region reset J-Link Script (`SMPU_AWS_Reset_Script.zip`) which can also reset the Security MPU and FAW settings before the FSPR bit is cleared (that is, the FAW region is permanently locked down). Unzip `SMPU_AWS_Reset_J-Link_Script.zip` to reveal two files (Figure 50).

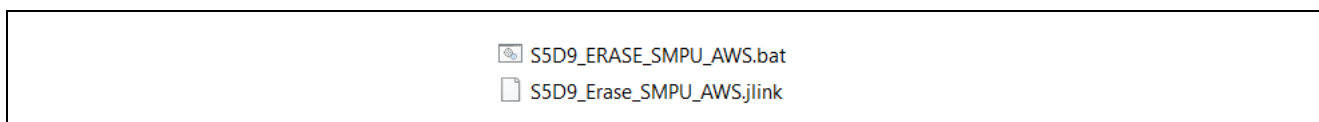


Figure 50. Security MPU and FAW reset script

Open `S5D9_ERASE_SMPU_AWS.bat` and update the Segger J-Link version to match your J-Link version.

```
S5D9_ERASE_SMPU_AWS.bat - Notepad
File Edit Format View Help
set PATH=%PATH%;"C:\Program Files (x86)\SEGGER\JLink_V634f"
jlink -CommanderScript S5D9_Erase_SMPU_AWS.jlink
pause
```

Figure 51. Update the Segger J-Link version

The script performs the following functions:

1. Reset the device
2. Reprogram the Flash Access Window configuration area (0x0100A160) to all FFs
3. Reset the device
4. Erase the first block of code flash
5. Reset the device

To run the reset script, put S5D9 into Normal Boot mode and open a command line window, navigate to the location of the .bat file, and run the S5d9_ERASE_SMPU_AWS.bat.

5.5 J-Link Scripts for Resetting OSIS ID code for S5D9

Unzip the source code Reset_Scripts.zip and locate the OSIS_ID_Resetting_Script.zip. Unzip OSIS_ID_Resetting_Script.zip to reveal the scripts for resetting OSIS ID:

- ID_Code_ERASEALL_S5D9.bat
- IDCode_ALeRASE_RV40_ph2.jlinkscript
- S5D9_connect.jlink

Follow the steps below to exercise these reset scripts.

Step 1: If you don't have the Segger tools installed, go to <https://www.segger.com/downloads/J-Link/> and download & install the latest J-Link software.

Step 2: Copy the newly installed J-LinkARM.dll to (e² studio installation folder)\DebugComp\Synergy\ARM\Segger

Note: J-Link version V6.34 and higher is required for setting OSIS ID with the IDE.

Step 3: Create a blinky project using the e² studio.

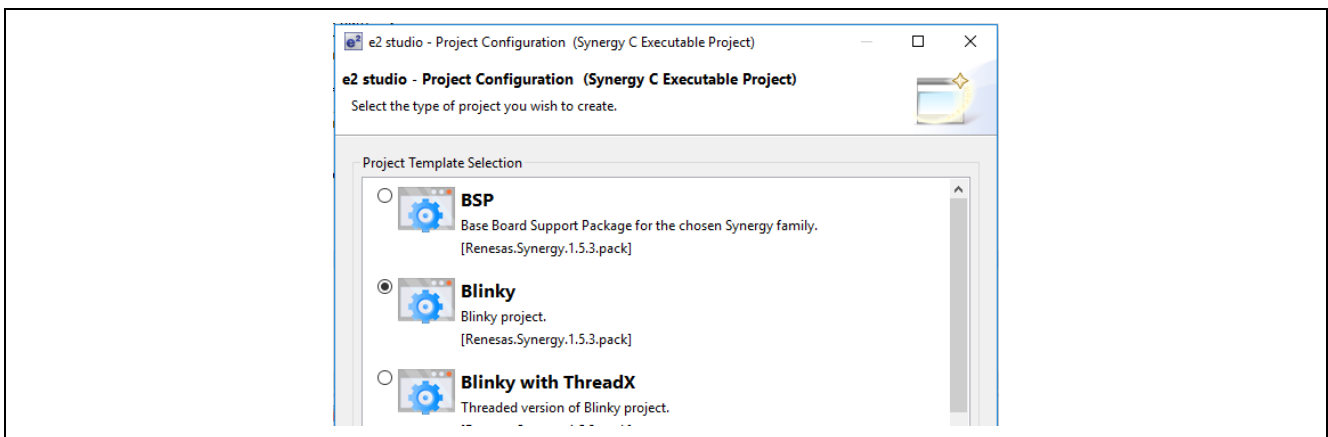


Figure 52. Create a blinky project

Step 4: Setup the ID code in the "Property" field under the BSP tab. Note bit 127 and bit 126 are both set to 1.

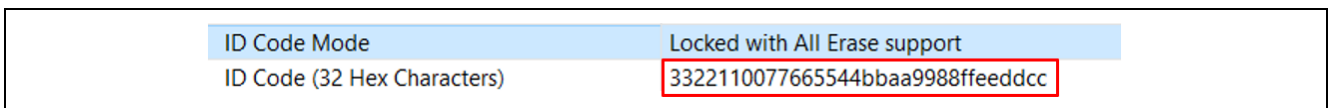


Figure 53. Setup ID code in configurator

Step 5: Click **Generate Project Content**, and compile the project.

Step 6: Open the debug configuration **Debugger->Connection Settings**, input the OSIS ID below. Click **Apply->Debug**:

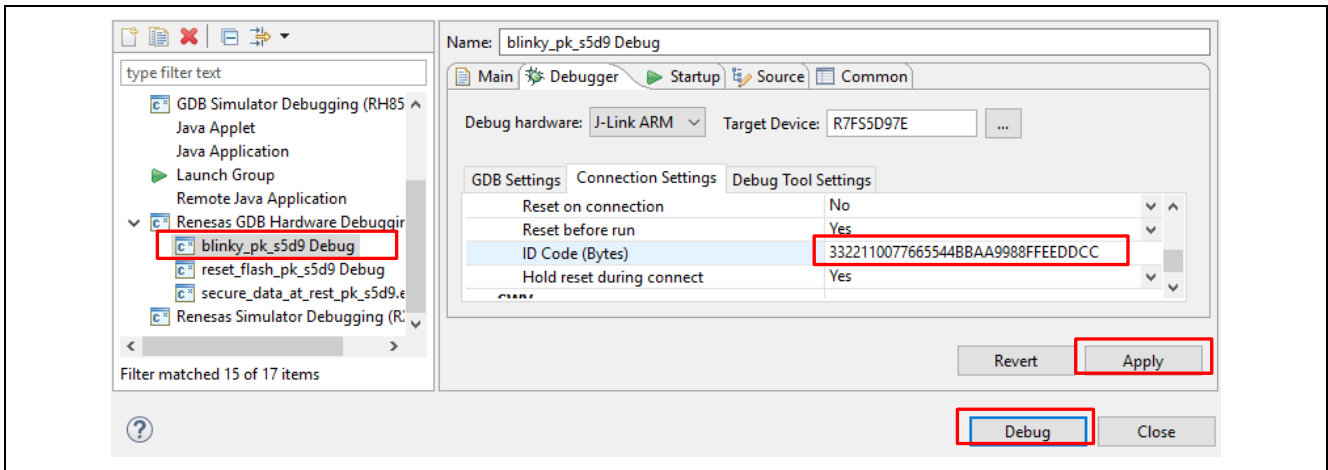


Figure 54. Setup matching ID code in debug configuration

Step 7: Debug the project

Step 8. Stop the project.

Step 9. Open a command prompt and navigate to the folder where you extracted the script files.

Step 10. Set the path to the version of J-Link that you have installed:

```
set PATH=%PATH%;"C:\Program Files (x86)\SEGGER\JLink_V641a"
```

Step 11. Run the batch file ID_Code_ERASEALL_S5D9.bat you should see output similar to the output below:

```

C:\MyProjects\Data_At_Rest\delivery_01_17\OSIS_ID_Resetting_Script>jlink -JLinkScriptFile IDCode_ALeRASE_RV40_ph2.Jli
nkScript -CommanderScript S5D9_connect.jlink
SEGGER J-Link Commander V6.41a (Compiled Nov 27 2018 14:10:26)
DLL version V6.41a, compiled Nov 27 2018 14:09:51

J-Link Command File read successfully.
Processing script file...

J-Link connection not established yet but required for command.
Connecting to J-Link via USB...O.K.
Firmware: J-Link OB RX621-ARM-SWD V1 compiled Mar  8 2017 13:46:30
Hardware version: V2.10
S/N: 708005016
VTref=3.300V

Selecting SWD as current target interface.

Selecting 4000 kHz as target interface speed

Device "R7FS5D97C" selected.

Connecting to target via SWD
InitTarget() start

Synergy ALeRASE J-Link script InitTarget

ALeRASE CMD
SWO:

DAP-CtrlStat: 0x00000040
DAP-CtrlStat: 0xF0000040
DAP-CtrlStat: 0xC0000040
RESET assert
RESET de-assert
SWO:

DAP-CtrlStat: 0xF0000040
MCUSTAT: 0x00000000
MCUSTAT: 0x00000002
RESET assert
RESET de-assert
DAP-CtrlStat: 0xF0000040
DAP-CtrlStat: 0xF0000040
MCUSTAT: 0x00000001
InitTarget() end
Found SW-DP with ID 0x5BA02477
Scanning AP map to find all available APs
AP[2]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x24770011)
AP[1]: APB-AP (IDR: 0x44770002)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Found Cortex-M4 r0p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E00E000, CID: B105E00D, PID: 000BB00C SCS-M7
ROMTbl[0][1]: E001000, CID: B105E00D, PID: 003BB002 DWT
ROMTbl[0][2]: E002000, CID: B105E00D, PID: 002BB003 FPB
ROMTbl[0][3]: E000000, CID: B105E00D, PID: 003BB001 ITM
ROMTbl[0][4]: E004000, CID: B105900D, PID: 000BB9A1 TPIU
ROMTbl[0][5]: E0041000, CID: B105900D, PID: 000BB925 ETM
ROMTbl[0][6]: E0042000, CID: B105900D, PID: 002BB908 CSTF
ROMTbl[0][7]: E0043000, CID: B105900D, PID: 001BB961 TMC
ROMTbl[0][8]: E0044000, CID: B105F00D, PID: 001BB101 TSG

Synergy ALeRASE J-Link script SetupTarget

MCUSTAT: 0x00000001
MCUCTRL: 0x00000000
SYOCDR: 0x00000080
Memory Address 0: 0xFFFFFFFF
OSIS 1: 0xFFFFFFFF
OSIS 2: 0xFFFFFFFF
OSIS 3: 0xFFFFFFFF
OSIS 4: 0xFFFFFFFF
Cortex-M4 identified.

Script processing completed.

```

Figure 55. Erase entire flash and reset the OSIS ID code

Step 13. Reset the OSIS ID to all FFs in the configurator for the blinky project (see Figure 53) and compile the project.

Step 14. Reset the OSIS ID for the debugger configuration to all FFs (see Figure 54) and confirm the debugging works.

6. Secure Data at Rest Next Steps

6.1 Secure Data Encryption and Authentication

A future release of this application project will cover secure data encryption and authentication.

6.2 External Storage Secure Data at Rest

A future release of this application project will cover external storage data encryption and authentication as well as access control.

6.3 Example using the Security MPU Security Functions

As explained in section 2.2, there are two security functions that can be controlled by the Security MPU.

One security function controls the access to the Secure Crypto Engine (SCE). When this security function is enabled, only secure program can access the SCE.

The other security function controls the access to the code and data flash Erase and Programming (E/P). When this security function is enabled, only secure program can perform internal flash E/P.

A future release of this application project will provide an example usage of these security functions.

7. References

1. MQTT and TLS application project: <https://www.renesas.com/us/en/software/D6002391.html>
2. Secure Boot Manager application project: <https://www.renesas.com/us/en/software/D6002619.html>
3. Renesas Flash Programmer: <https://www.renesas.com/us/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html#downloads>
4. *ARM®v7-M Architecture Reference Manual* (ARM DDI 0403D)
5. *ARM® Cortex®-M4 Processor Technical Reference Manual* (ARM DDI 0439D)
6. *ARM® Cortex®-M4 Devices Generic User Guide* (ARM DUI 0553A)

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.31.19	—	First release document

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
 - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
 - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.