

Implementing Run-Time Safety and Security With the C29x Safety and Security Unit



Ibukun Olumuyiwa, Marlyn Rosales Castaneda, and Aishwarya Rajesh

ABSTRACT

The Safety and Security Unit (SSU) is an integrated module in F29x devices that enables run-time functional safety and cybersecurity protections for application code. The features of the SSU enable robust Freedom from Interference (FFI), secure task isolation, debug security and firmware update protections in hardware, maintaining low-latency performance needed for real-time control systems. The [SysConfig](#) tool, provided as part of the [F29 SDK](#), provides an easy-to-use graphical user interface (GUI) for configuring the SSU and enabling safety and security protections in user applications. This application note examines the various features of the SSU, and how embedded system developers can use the SSU Tool within SysConfig to design and implement run-time safety and security in real-time applications.

Table of Contents

1 Introduction	3
2 Supplemental Online Information	3
3 SSU Overview	4
4 Key Concept Definitions	5
5 Safety and Security Goals	6
6 System Design	7
7 Configuring the SSU	10
7.1 Flash SECCFG Region.....	10
7.2 SSU Development Life Cycle.....	10
7.3 Using the SysConfig Tool.....	11
8 Debug Authorization	18
8.1 Password-Based Unlock.....	18
9 Debugging the SSU	20
9.1 Debugging Build Errors.....	20
9.2 Debugging Runtime Errors.....	20
10 SSU Frequently Asked Questions (FAQ)	23
11 Summary	24
12 References	24
13 Revision History	24

List of Figures

Figure 3-1. SSU System Block Diagram (Simplified View).....	4
Figure 6-1. Example of Software Partitioning With SSU.....	8
Figure 7-1. System Security Configuration Page.....	11
Figure 7-2. Application Module Configuration Example.....	13
Figure 7-3. Special Modules Configuration Example.....	14
Figure 7-4. LINK2 Configuration Example.....	15
Figure 7-5. Shared Memory Configuration Example.....	17
Figure 8-1. SSU Mode 3 additional settings.....	18

Trademarks

E2E™, Code Composer Studio™, and C2000™ are trademarks of Texas Instruments.
FreeRTOS® is a registered trademark of Amazon Web Services, Inc.
AUTOSAR® is a registered trademark of AUTOSAR Development Partnership.

All trademarks are the property of their respective owners.

1 Introduction

The Texas Instruments C29 CPU delivers industry-leading performance for real-time control applications. With a 128-bit Very Large Instruction Word (VLIW) architecture, 64-bit fixed-point and floating-point operations, ultra-low latency processing and hardware interrupt prioritization, the C29 is well-equipped to run the most demanding automotive and industrial control applications. The SSU, in concert with the C29 CPU, helps system designers meet the most rigorous modern standards for safety and security in the real-time control applications, without compromising real-time performance. With the SSU, users can achieve true FFI, secure task isolation and advanced debug and firmware update security, while maintaining the same high-speed and low-latency processing needed for the most demanding real-time control systems.

This application note describes how to implement run-time application safety and security in a real-time control system using the C29x CPU and SSU. The C29x, SSU architecture provides dynamic context-sensitive memory protection, secure task isolation with multiple dedicated CPU stack pointers, and multiuser debug ZONES for security.

2 Supplemental Online Information

For detailed descriptions of the C29x CPU and the SSU on a specific device, see the device-specific data sheet and the corresponding Technical Reference Manual (TRM).

This application report was written using the F29H85x family of devices. The [F29 SDK](#) and SysConfig tool support all F29x platform devices.

- [C29x CPU and Instruction Set User's Guide](#)
- [F29H85x and F29P58x Real-Time Microcontrollers Data Sheet](#)
- [F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual](#)
- [C29x Academy: Safety and Security Unit \(SSU\) Chapter](#)

Additional support is provided by the [TI E2E™ Community](#).

3 SSU Overview

The SSU is an integral part of the C29 CPU subsystem, providing context-sensitive memory protection and run-time code isolation, debug security, and secure firmware updates. The SSU acts as a firewall between the C29 CPU and the rest of the system, enforcing user access protection policies and managing debug access and Flash controller operations. Figure 3-1 shows a simplified overview of the SSU as integrated into the C29 subsystem. The features available in the SSU can be used to achieve true FFI in a real-time control system without adding software overhead, which can otherwise negatively impact real-time performance. With the SSU, system designers can combine multiple control and communication functions on the same CPU core, while keeping each function isolated from the others. This can lead to a reduction in the number of cores required to reach system goals, or help achieve higher system safety integrity levels.

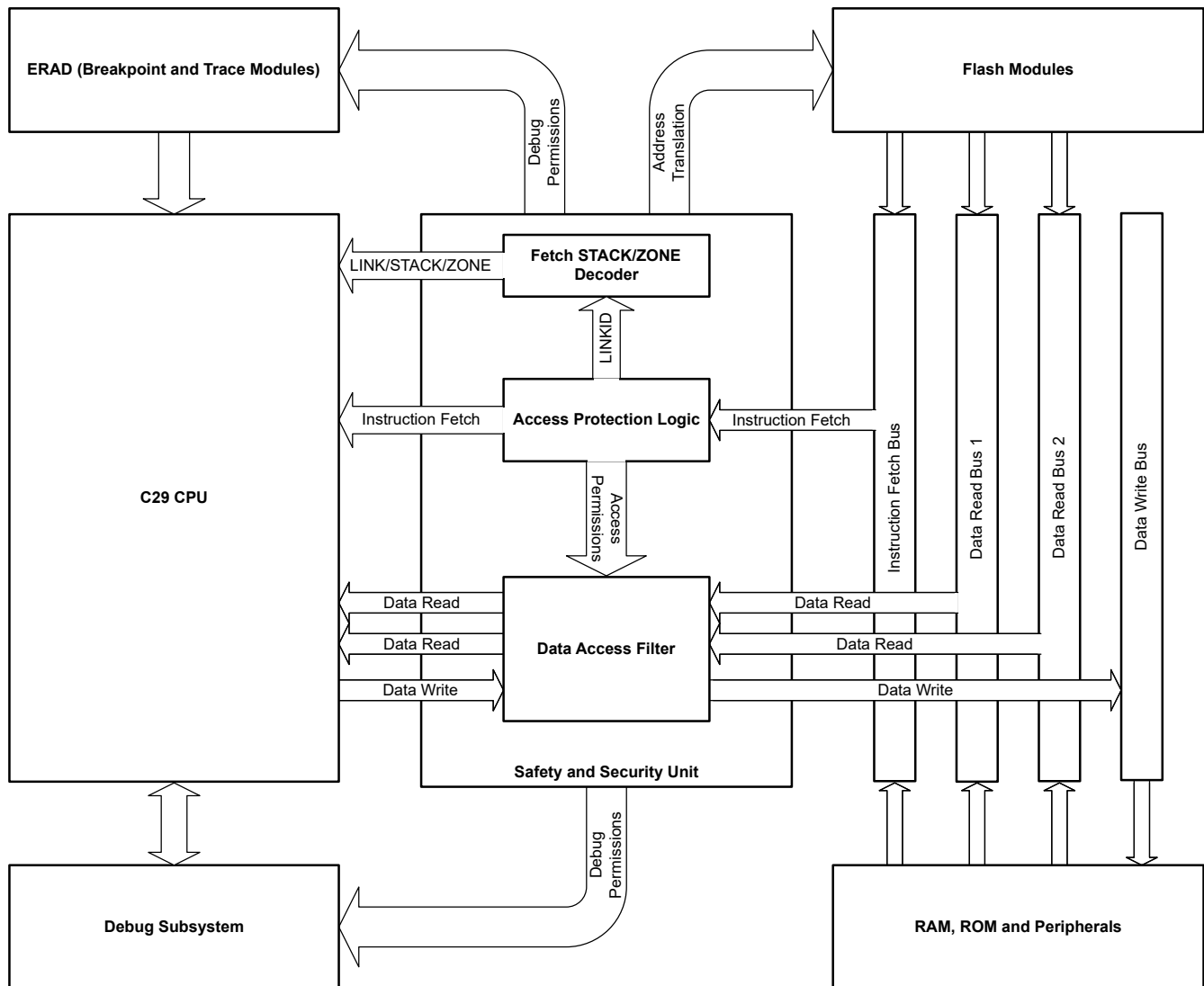


Figure 3-1. SSU System Block Diagram (Simplified View)

4 Key Concept Definitions

This section contains the definitions of key concepts.

Access Protection Range (APR)	This is the basic unit of memory protection in the SSU. An access protection range covers a specific region of Flash memory, SRAM, or peripherals. Each APR defines read and write access permissions for every LINK. An APR can also be configured as a code region, which enables CPU instruction fetches from that memory region.
LINK	In a C29 CPU subsystem, LINKs form the basis for context-sensitive memory protection. Each LINK can represent one or more regions of executable code. The associated LINK identifier is used to determine what data memory regions (APRs) can be accessed by that code.
STACK	STACKs isolate code execution contexts from each other. Each STACK has a dedicated stack pointer in the C29 CPU, and provides hardware safety and security isolation of code from other STACKs. Every LINK belongs to one and only one STACK, but a STACK can contain multiple LINKs.
ZONE	ZONES determine debug and firmware update permissions. While APRs, LINKs, and STACKs are defined independently for each CPU, ZONES span the entire device, excluding the Hardware Security Module (HSM) (which is not governed by the SSU).
HSM	The Hardware Security Manager is a self-contained subsystem within the device that provides key security services, including secure boot, secure storage, debug and firmware update security, and run-time cryptographic services. The HSM is different from the SSU, which is an integral part of the application C29 CPU subsystem. The HSM and SSU perform complementary and orthogonal roles on the device, except for debug authorization: both the HSM and SSU must authorize access to a resource before debug access is enabled to that resource.
Partial Debug	When a ZONE is enabled for partial debug, the user is allowed to debug CPU execution (halt, resume, and view CPU registers), but debug read and write accesses to memories that can otherwise be accessed by LINKs in that ZONE are blocked.
Full Debug	When a ZONE is enabled for full debug, the user can debug the CPU and perform all memory accesses permitted for any LINK within that ZONE.
SECCFG	This is a special Flash region that is designated for storing SSU configuration settings. The values stored in the SECCFG region are loaded into the SSU registers during device boot. Most of these settings cannot be changed during run time, and can only be modified by programming new values into SECCFG and resetting the device.
UPP	User Protection Policy. This is the collection of SSU configuration settings that are programmed into the SECCFG region.
Memory Region	A region of memory configured in SysConfig. This is equivalent to an Access Protection Range (APR).
Module	In SysConfig, a Module consists of a LINK, the code memory regions (executable APRs) that are associated with that LINK, the data memory regions (data APRs) and peripherals that <i>belong</i> to the module, and peripheral interrupts associated with the module. In practice, modules allow the user to organize the application into distinct tasks or partitions that can be isolated from one another for functional safety and security.
Shared Memory	In SysConfig, a Shared Memory consists of one or more APRs that are accessible by multiple Modules. Shared Memories can be used to share data between Modules in a distinct memory range, while maintaining safety protection for other memory regions belonging to those modules.
Sandbox	In SysConfig, a Sandbox consists of a STACK, and can contain one or more Modules.
RTOS	A Real-Time Operating System, such as FreeRTOS® or AUTOSAR®.

5 Safety and Security Goals

The Safety and Security Unit enables system designers to accomplish important safety- and security-related objectives in the design of real-time embedded systems. These objectives include:

1. **Memory Protection:** An essential element of an embedded microcontroller that supports functional safety goals is a Memory Protection Unit, or MPU. An MPU enforces access control rules over memories in the system, to prevent unauthorized reads, accidental overwrites, or unauthorized modifications to code and data. Memory protection plays an important role in maintaining system stability, reliability, and security. The SSU provides advanced MPU functionality that is context-sensitive, switching protections in real time without software intervention.
2. **Freedom from Interference:** In the ISO 26262 standard, which defines functional safety standards for automotive electronics, Freedom from Interference (FFI) is defined as the “*absence of cascading failures between two or more elements that can lead to the violation of a safety requirement.*” A cascading failure occurs when one component in the system fails, and the failure of that component causes a different component in the system to fail; these failures can result in a progressively growing positive feedback loop. The SSU provides mechanisms to fully isolate multiple different system software components from each other, such that a safety failure in one component does not compromise the rest of the application.
3. **Security Isolation:** In addition to safety freedom from interference, the SSU supports security isolation goals, giving each application component a secure execution environment that protects the confidentiality and integrity of code and data assets during run time.
4. **Real-Time Performance:** A critical goal of the SSU is to provide safety and security protections without impact to real-time performance. Memory protection, security isolation, and other SSU functions are all performed in real time without software intervention, eliminating extra latency due to supervisor software overhead. Combined with the industry-leading performance of the C29 CPU, this enables system designers to combine multiple control functions on the same CPU without sacrificing performance, safety or security goals, leading to reduced overall system cost.
5. **Secure Debug and Firmware Updates:** The SSU provides the ability to partition the system software into multiple user debug ZONEs, enabling multiple teams to securely maintain and debug different software components on the same chip. The SSU also manages Flash firmware, controlling which users and code are permitted to perform firmware updates, and enabling mechanisms such as Firmware-Over-The-Air (FOTA) and Live Firmware Update (LFU) with A, B swapping and rollback protection in hardware.

6 System Design

The first step in configuring the SSU for an application is to determine the required system partitioning. The SSU provides three levels of hierarchy for partitioning the application subsystem:

1. **ZONES** : Each ZONE determines debug access for all C29 CPUs on the chip. ZONES are designed to enable multiple code owners or entities to develop and maintain different partitions of an application residing on the same chip. For instance, if a certain aspect of the embedded application is owned and maintained by a third-party vendor, then the system can be divided into two ZONES:
 - a. ZONE1: Primary user ZONE, owned by the primary system developer;
 - b. ZONE2: Secondary user ZONE, owned by the third-party developer.

This partitioning enables the third-party developer to develop, debug and maintain an application function on the same chip without requiring access to the primary user's code and data assets. Furthermore, each user ZONE provides two levels of debug authorization:

- a. Partial debug – CPU debug commands such as halt, step, and breakpoints allowed, but no memory access
- b. Full debug – Access to memory locations is provided as permitted for all LINKs contained within the ZONE.

As an example, a secondary user such as a third-party developer can debug an application module in ZONE2, and also be given partial debug access to ZONE1, so that the secondary user can effectively debug the application in context without having access to the primary user's assets.

Each device has 3 user ZONES available: ZONE1, ZONE2, and ZONE3. ZONE1 is the primary user ZONE; ZONE2 and ZONE3 are secondary user ZONES.

2. **Sandboxes (STACKs)**: Sandboxes provide security and safety isolation within a CPU. Each Sandbox is associated with a STACK in the SSU. Each sandbox has a dedicated physical stack pointer in the CPU that is inaccessible by other sandboxes, and a dedicated stack memory AP region with read/write permissions restricted to only code belonging to that sandbox.

Special C29 CPU gate instructions are required when crossing from one STACK to another. These instructions must be inserted by the compiler at the entry and exit of each function, and at function calls or branches. These mechanisms provide security protection against malware attacks that attempt to redirect code execution or manipulate the stack.

A Sandbox consists of an SSU STACK and everything associated with the STACK, including the stack memory AP region. Each STACK belongs to one ZONE, but a ZONE can contain multiple STACKs

Table 6-1. Predefined STACKs per CPU

STACK #	Description
STACK0	This STACK is reserved for TI internal use and cannot be configured by the user.
STACK1	This STACK is primarily used for bootloaders, but can optionally be associated with other user application code. STACK1 is always associated with ZONE1, and contains only one LINK (LINK1).
STACK2	This is the primary user STACK. STACK2 is always associated with ZONE1. STACK2 always contains LINK2, but can also contain other LINKs.

3. **Application Modules (LINKs)**: An application module is a basic partition of a system application. Each module consists of a single SSU LINK, one or more code memory AP regions containing the code of the LINKs, all data memory AP regions associated with the LINK, and all peripherals and interrupts associated with the Module.

Typically, the code AP regions contain .text and other linker output sections containing code, and the data AP regions contain .bss, .const, and other linker output sections containing data and variables.

Each LINK enables SSU memory protections, providing safety protection from other LINKs in the CPU. Every AP region defines access permissions for each LINK. These permissions are enforced in real time

for every instruction that performs a memory access, depending on the LINK ID instruction. Functions that require safety isolation from each other can be placed in separate Modules. If security isolation is required, then these Modules are placed in separate sandboxes; if not, the Modules can be placed in the same sandbox.

Table 6-2. Predefined LINKs per CPU

LINK #	Description
LINK0	This LINK is reserved for TI internal use and cannot be configured by the user.
LINK1	This LINK is primarily used for bootloaders, but can optionally be associated with other user application code. CPU1.LINK1 has some special fixed permissions that enable access to certain system configuration registers, in addition to AP-defined protections.
LINK2	This is the primary user LINK. CPU1.LINK2 is the system security root LINK (SROOT), and has special fixed permissions that enable access to system configuration registers and override controls. This LINK typically executes privileged host functions at the RTOS level.

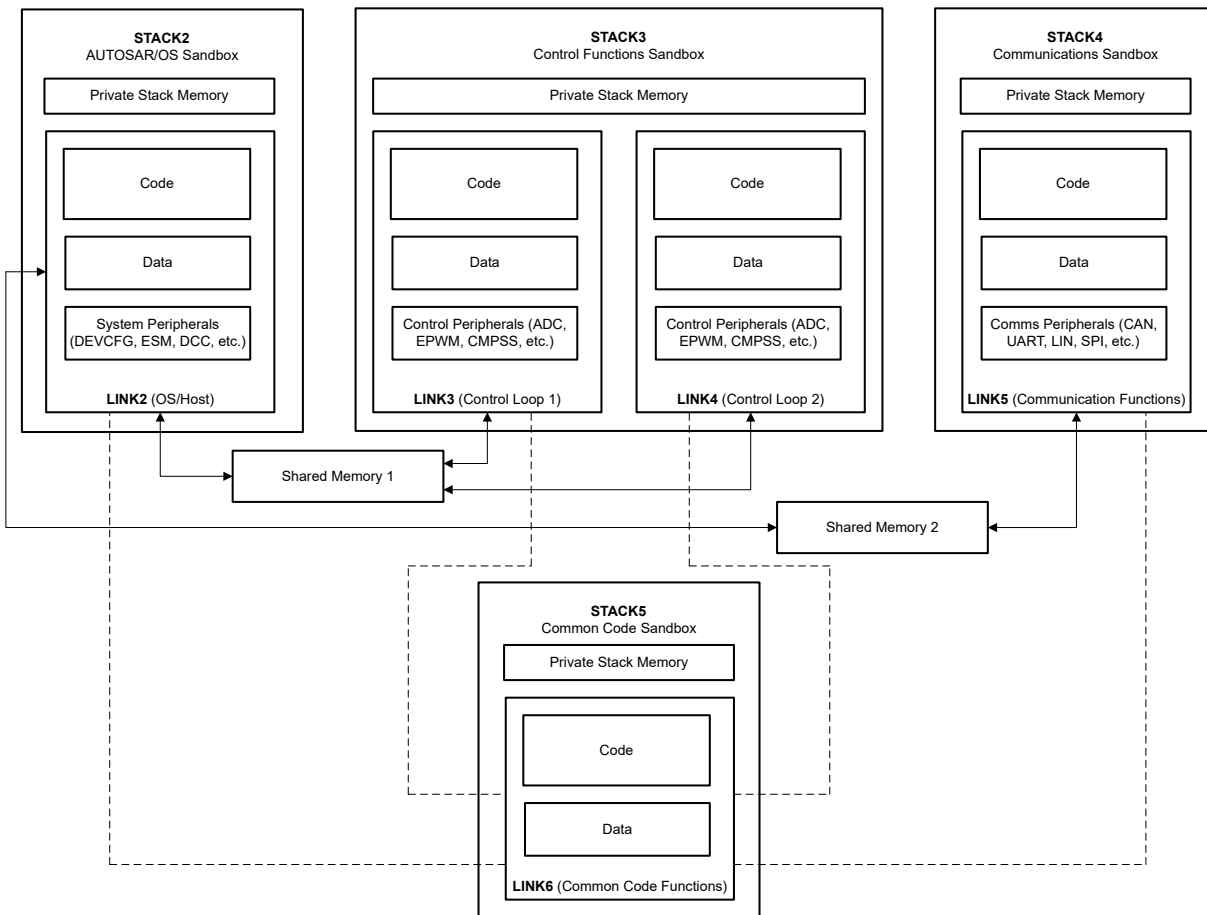


Figure 6-1. Example of Software Partitioning With SSU

Figure 6-1 shows an example of SSU system partitioning on a single-CPU, single-ZONE system and Table 6-3 explains the details behind this configuration.

Table 6-3. Example of SSU Partitioning Explained

STACK	LINK	Details
STACK 2	LINK2	The RTOS runs in STACK2.LINK2, and is responsible for initializing system configuration, setting up peripherals and interrupts, and starting the main execution loop. While various tasks and respective task stacks can be placed in other links, all of the RTOS tasks/stacks need to be within SSU STACK2. If RTINTs are used then the code and data related to the RTINT should be placed in a sepeare SSU STACK other than SSU STACK2.

Table 6-3. Example of SSU Partitioning Explained (continued)

STACK	LINK	Details
STACK 3	LINK3 and LINK 4	There are two control functions, <i>Control Loop1</i> and <i>Control Loop 2</i> . Each of these control functions is placed in a separate Application Module (LINK), and both Application Modules are placed in the same Sandbox. In this system, safety isolation is required between the two control functions, but security isolation is not required between the two.
STACK 4	LINK 5	LINK5 contains the host communication code, such as UART or CAN-FD code. Because data coming from an external interface can potentially pose a security threat to other functions in the system, this Module is placed in a separate Sandbox.
STACK 5	LINK 6	Contains common code functions which are shared between all the other Modules in the system. LINK 6 is defined as the Access Protection Inheritance LINK (APILINK) for other LINKs. The Common Code Module is also placed in a separate sandbox to maintain security isolation from the rest of the system (while maintaining inherited permissions). It is recommended that the common code LINK be placed in its own STACK.

SysConfig includes full support for multicore applications. The built-in memory allocator tool automatically manages memory regions associated with application modules across multiple CPUs, and also manages the allocation of peripherals across the entire device. The SysConfig tool also includes a [Shared Memory feature](#) that enables the definition of memory regions that can be shared between modules on the same CPU or multiple CPUs.

7 Configuring the SSU

7.1 Flash SECCFG Region

The Flash SECCFG region is used for storing the User Protection Policy (UPP). This is a special NONMAIN region of the C29 application Flash banks that is dedicated to SSU configuration and boot settings. The settings programmed into SECCFG are loaded at device start-up into SSU memory-mapped registers, and, in most cases, locked until the next device reset. For each primary CPU in the device (that is, odd-numbered C29 CPU, for example: CPU1, CPU3), there are two SECCFG sectors: the base sector, and the reserve sector. These are designed such that one sector can be erased and programmed with new configuration values while the other is active.

Note

Do not try to erase and reprogram a currently active SECCFG sector. If a device reset occurs during the process of erasing and programming, the device subsequently fails to boot and becomes inoperable. Always program new configurations into the alternate SECCFG sector address. The Flash address translation logic automatically routes this address to the current inactive SECCFG sector during program and erase operations. The SysConfig tool automatically allocates the SECCFG image to the alternate sector in the generated .out file to enable the correct update procedure.

Note

There are four different flash bank modes on F29x devices. Refer to the F29x technical reference manual for more details on the different bank modes. To configure the bank mode properly for the application scenario of interest right click on CPU1 within the 'Connections' while in the CCS Debug view and choose the 'Properties' option. Within the properties window select the 'Flash Settings' drop down. Go to 'Bank Mode' and select the desired bank mode. Click on 'Program Bank Mode'.

To protect the integrity of the SSU user protection policy, the SECCFG sector includes a CRC value that is checked at boot time. This CRC covers access protection settings, LINK and STACK configuration, Flash write and Flash erase protections, Flash update permissions, debug settings, boot settings, and the SSU operating mode. Debug passwords are excluded from this CRC computation.

A comprehensive map of the SECCFG sector is available in the device technical reference manual.

7.2 SSU Development Life Cycle

The SSU can be configured to operate in one of three modes: SSUMODE1, SSUMODE2, and SSUMODE3. These operating modes are intended to facilitate the development process as the user implements safety and security features into a system design. The SSU can be reconfigured to change from any operating mode to any other operating mode, as long as the user has the necessary permissions to update the SECCFG sector.

Units shipped from Texas Instruments start out in SSUMODE1. In this mode, the entire memory map range is mapped to LINK2 (the security root LINK), and all LINKs have full read and write access to all AP-configurable memory regions. Hard-coded protections remain active even in SSUMODE1; however, since all code runs as LINK2, there are effectively no restrictions on user code.

In SSUMODE2, AP region protections are enforced, but debug and Flash update protections remain disabled. For best results, fully validate application functionality in SSUMODE1 first, then implement SSU settings and test them in SSUMODE2. Once validation of run-time safety and security settings is completed, debug passwords can be configured, and the device can be placed in SSUMODE3. In this mode, debug ports are closed by default, authentication is enforced, and Flash update protections are active.

Note

Flash write and Flash erase protections are permanent and cannot be reversed, irrespective of the SSUMODE setting. This feature is intended for use cases where the user needs to make a certain portion of Flash code immutable, for example, for the purpose of implementing a security algorithm. Do not attempt to configure Flash write and Flash erase protections before finalizing device Flash contents.

7.3 Using the SysConfig Tool

SysConfig is a graphical user interface (GUI) tool that provides an easy-to-use method for configuring TI microcontroller products, including F29x Real-Time Control MCUs. SysConfig automatically generates initialization code for the device, including peripherals, interrupts, pin multiplexing initialization, and more. SysConfig also automatically identifies device setup errors and provides helpful guidance to rectify configuration issues, provides graphical visualizations, and enables easy porting of applications between different devices.

C29 SysConfig is available as part of the device [F29 SDK](#), and requires the [SysConfig tool](#), which is delivered built-in with the [Code Composer Studio™ \(CCS\) integrated development environment \(IDE\)](#), and is also available as a standalone tool for use with other development environments.

The SSU tool within C29 SysConfig is a comprehensive tool for defining and implementing SSU protection and isolation within an F29x application. The tool provides an easy-to-use flow for defining application partitions, allocating code and data sections, defining protections for memory and peripherals, and implementing debug security options. The SSU tool automatically generates all the required code and output files required to implement these protections, including the SECCFG sector image, application linker command file, header files, and any required initialization code.

7.3.1 Enabling System Security Configuration

The first step to enabling SSU functionality in an application is to add the *System Security* option in SysConfig, by clicking the (+) button to the right of the module name in the left bar, see [Figure 7-1](#).

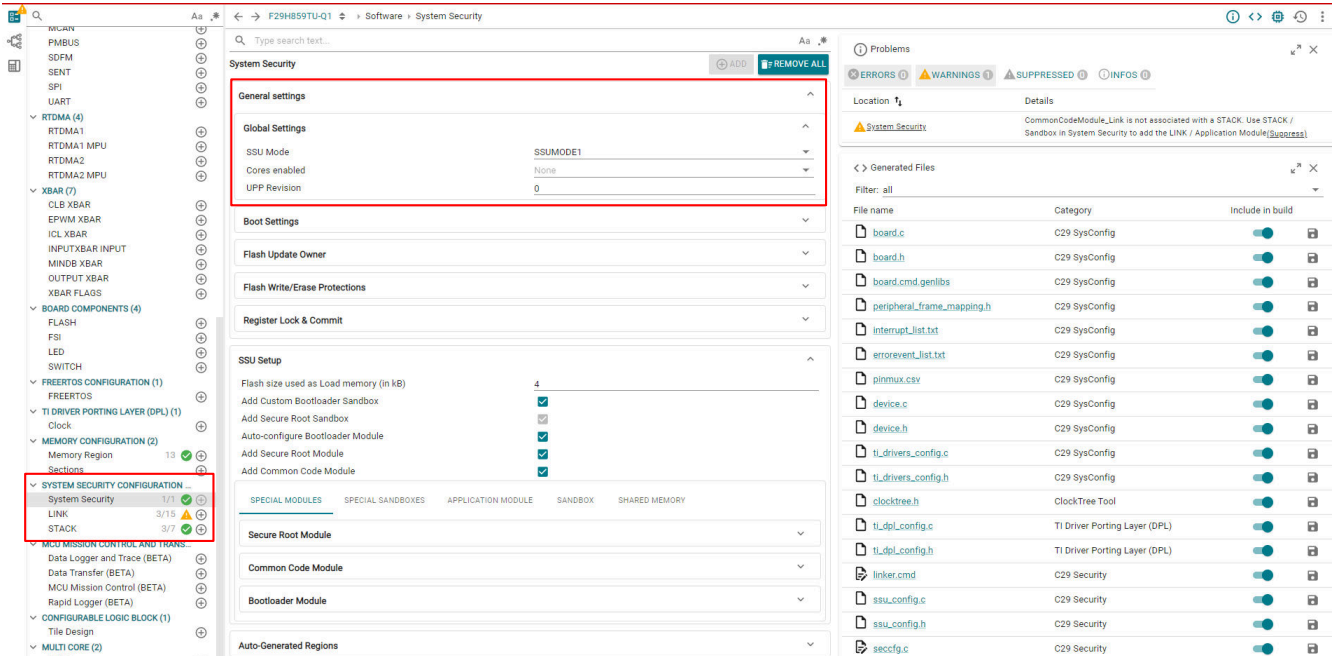


Figure 7-1. System Security Configuration Page

In the *System Security* module, there are several configuration option groups. The first of these is the *General Settings*, which contains settings for SSU operation mode, UPP revision number, and cores enabled.

The *System Security* module also includes configuration options for selecting the device boot mode, configuring Flash update protections, debug passwords (Only visible when SSUMODE 3 is selected as the SSU Mode), and locking SSU registers. Some of these settings, such as debug passwords and Flash update owner settings, require SSUMODE3 operation to take effect.

When the *System Security* module is added in SysConfig, there are other modules that get automatically added such as the LINK, STACK, and Memory Region modules. There is no need to modify the LINK and STACK modules. All of the configuration is done through the *System Security* module. The LINK and STACK modules simply provide a summary of the LINK and STACK configurations selected through the *System Security* module.

The linker command file (.cmd) is also auto generated from SysConfig once the *System Security* module is added. If using an existing project, exclude the .cmd file from the project build to prevent conflicts.

Note

If extra modifications or additions need to be made to the linker command file, use the *Memory Region* and *Sections* modules within the Memory Configuration section of SysConfig.

Note

When the *System Security* module is first added in SysConfig, there is a warning that the CommonCodeModule_Link is not associated with a STACK/SANDBOX in the *System Security* module. To resolve this warning, go to the SANDBOX tab located within the SSU Setup of the *System Security* module and add a SANDBOX with CommonCodeModule_LINK selected as one of the Modules.

7.3.2 Configuring Application Modules

SysConfig provides an easy way to create AP ranges and configure LINK permissions based on object files, libraries, and input sections. When a new Application Module is created, SysConfig automatically creates a LINK, together with a standard set of AP regions:

- A code region that executes out of Flash (ModuleName_codeAPR_Flash)
- [Optional] A code region that executes out of RAM (ModuleName_codeAPR_RAM)
- A variable data region in RAM (ModuleName_dataAPR_RW)
- A read-only data region that can optionally be placed in RAM (ModuleName_dataAPR_RO)

In addition to the standard regions, the user can configure custom section names to be associated with the Application Module, by selecting the *Use Custom Sections* checkbox, and specifying custom sections to be added. SysConfig adds all the defined AP regions to the SSU settings, and configures the associated LINK to have the appropriate permissions for each region. In addition, an output section is created in the linker command file for each AP region, instructing the linker to place input sections in that memory region as configured.

To associate code functions and data with the Application Module, simply add file names to the *Files to be included* input field, minus the file extension. Libraries can also be added to the Module by editing the corresponding input field (with the library file extension included). To select specific objects from a library, use linker command file syntax, for example myLibrary.lib<myFuncs1.o>. That is all that needs to be done: SysConfig automatically assigns the .text, .bss, .data, .rodata, and .const input sections for each object to the corresponding output sections in the linker command file.

To allocate memory to the module, simply specify the amount of memory required for each APR type (Flash code, RAM code, RW data, RO data). SysConfig automatically arranges AP regions in memory, selecting the best memory type as required for minimum wait states. In cases where an Application Module must execute from RAM instead of Flash to meet performance requirements, select the *Place .text section in RAM* checkbox. When this checkbox is selected, SysConfig creates a new RAM code region, and configures the linker command file to load the associated code from Flash at boot and run from RAM. Read-only or constant data such as look-up tables can also be placed in RAM for zero-wait-state access, if desired.

In addition to code and data memory regions, existing peripherals that have been configured through SysConfig can also be auto-allocated to each Application Module. Two drop-down select fields are provided for enabling either read-write access or read-only access to the specified peripherals.

Peripheral interrupts can easily be added using the Interrupts Included field. This option configures the PIPE module to assign the correct execution LINK to the selected peripheral interrupts.

Note

INTs use the designated STACK configured through the INTSP register. This is a configuration applicable to all INTs. Thus, if using one or more INTs, the ISRs have to be placed in the same SSU STACK, that aligns with the STACK chosen in the INTSP register. There are no restrictions when using RTINTs.

The *Memory Regions* module within the Memory Configuration section of SysConfig shows the details of each AP region that has been created for the current Application Module. This frame also provides a few additional configuration options:

- **Use 0 WS Memory only:** Restricts RAM code to zero-wait-state RAM.
- **Create equivalent RTDMA MPU Region:** Creates an MPU region with the same start and end address for DMA transfers.
- **Share with other cores:** Enables a memory region to be used by multiple CPUs.

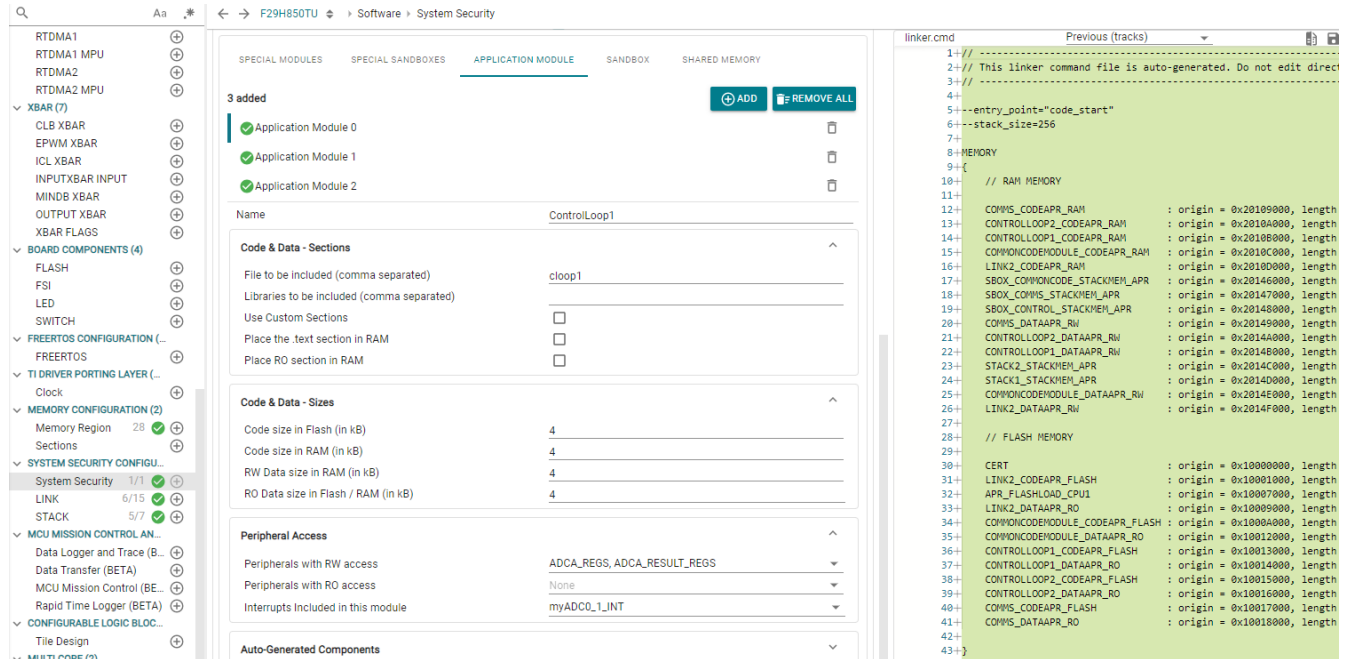


Figure 7-2. Application Module Configuration Example

7.3.3 Configuring Special Modules

SysConfig provides options to directly configure Modules that have predefined functions in the system:

- LINK2 – the system Security Root LINK,
- LINK1 – the Bootloader LINK,
- The Common Code LINK, for access protection inheritance

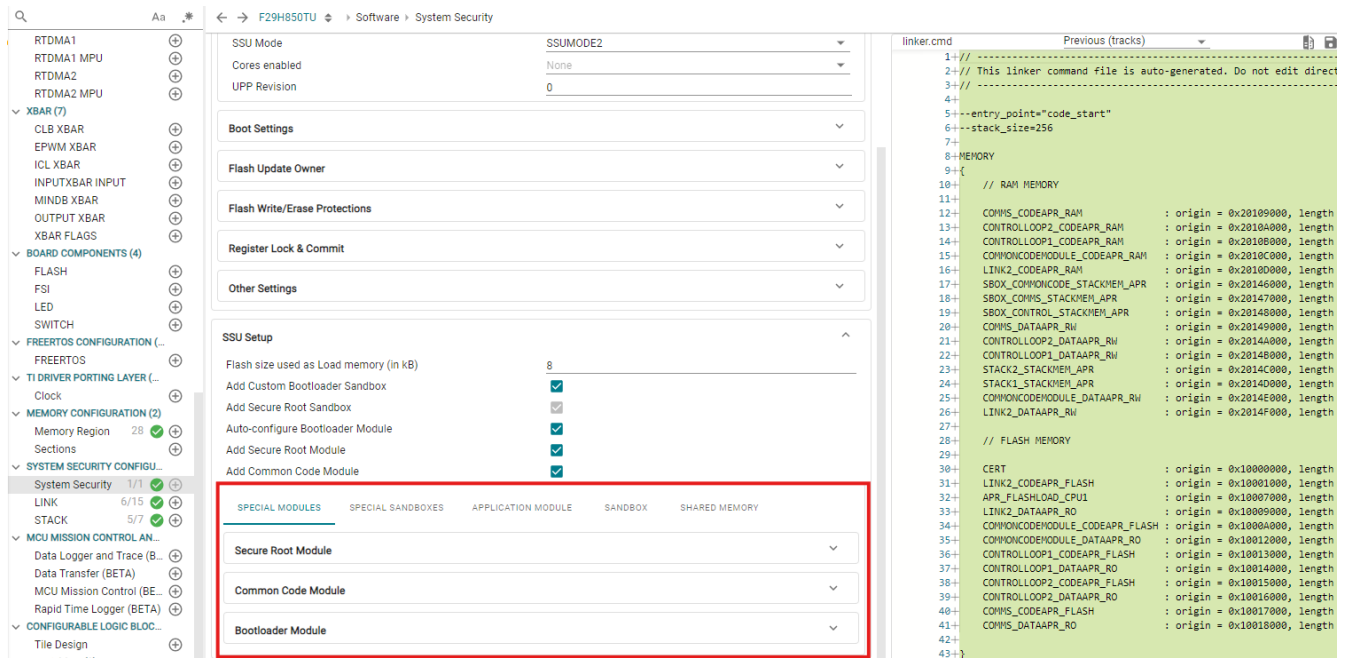


Figure 7-3. Special Modules Configuration Example

After enabling special modules, each module can then be configured by expanding the drop-down box. The following sections explain the special Modules in more detail.

7.3.3.1 LINK2 Configuration

LINK2 is the most secure LINK on each CPU. LINK2 has elevated privileges, including the ability to access secure CPU registers and perform supervisory tasks. In most cases, RTOS layer functions are placed in LINK2. CPU1.LINK2 in particular has special device-wide elevated privileges, including:

- Ability to write to device configuration registers
- Ability to write to certain SSU registers
- Ability to configure the RTDMA, including MPU configuration

Place all code and data sections that are responsible for performing initial system configuration and board configuration and running operating system functions in LINK2. [Table 7-1](#) describes the peripheral registers that are mandatory for LINK2 and why those registers are mandatory.

Table 7-1. Mandatory Peripheral Access Protection for LINK2

Peripheral Register Base	Reason
CPU_SYS_REGS	CPU system registers
PER_CFG_REGS_WD_REGS	Setup watchdog registers
CPUTIMER2_REGS	CPU timer 2 registers for the driver porting layer (DPL)
DCC1_REGS	Clock monitor
ERR_AGG_REGS	Error/NMI Handling
ESM_CPU1_REGS	Error/NMI Handling
ESM_CPU2_REGS	Error/NMI Handling
ESM_CPU3_REGS	Error/NMI Handling
ESM_SYS_REGS	Error/NMI Handling
ESM_SAFETY_AGG_REGS	Error/NMI Handling

SPECIAL MODULES
SPECIAL SANDBOXES
APPLICATION MODULE
SANDBOX
SHARED MEMORY

Secure Root Module ^

Name LINK2

Code & Data - Sections ^

File to be included (comma separated) ssu_ex1_mode2_rtos

Libraries to be included (comma separated)

Use Custom Sections

Default code sections included in Link2 libc.a<boot.c.obj>(.text), libc.a<autoinit.c.obj>(.text), libc.a...

Default sections included in Link2 libc.a<copy_zero_init.c.obj>, libc.a<copy_decompress_lzss...

Place the .text section in RAM

Place RO section in RAM

Code & Data - Sizes ^

Code size in Flash (in kB) 24

Code size in RAM (in kB) 12

RW Data size in RAM (in kB) 4

RO Data size in Flash / RAM (in kB) 4

Peripheral Access ^

Peripherals with RW access CPU_SYS_REGS, PER_CFG_REGS_WD_REGS +7

Peripherals with RO access None

Interrupts Included in this module None

Auto-Generated Components v

Figure 7-4. LINK2 Configuration Example

7.3.3.2 LINK1 Configuration

CPU1.LINK1 is primarily used for bootloaders, and has extra hard-coded privileges to support bootloader functions, such as the ability to write to certain system configuration registers. LINK1 can also be used as a conventional user LINK; however, adding non-bootloader-related code and data to LINK1 is not recommended except as a last resort when all other LINKs are already in use.

When any peripheral boot mode is configured in the *Boot Settings* group, SysConfig automatically configures LINK1 to have access to the respective peripherals required for that boot mode to function, for example, CAN, UART, or SPI. These peripherals are automatically added to the LINK1 module by SysConfig, in addition to certain peripherals that are always required for device boot, such as IPC and the HSM mailbox. [Table 7-2](#) describes the full list of mandatory peripherals required for LINK1 as part of device boot.

Table 7-2. Mandatory Peripheral Access Protection for LINK1

Peripheral Register Base	Reason
PER_CFG_REGS_WD_REGS	Setup for watchdog registers
IPC_CPU1_SEND_REGS_HSM_CH0	Handshake with HSM during device authentication
IPC_CPU1_SEND_REGS_HSM_CH1	Handshake with HSM during device authentication
HSM_MAILBOX	HSM Mailbox for IPC
GPIO_DATA_REGS	Boot mode selection and error status pin configuration
MCANA_MSGRAM	Peripheral boot mode
MCANA_REGS	Peripheral boot mode
UARTA_REGS	Peripheral boot mode
DCC1_REGS	Clock monitor
ERR_AGG_REGS	Error/NMI Handling
ESM_CPU1_REGS	Error/NMI Handling

7.3.3.3 Common Code Link Configuration

7.3.4 Defining Sandboxes

Use Sandboxes in SysConfig to define groups of Application Modules that must have security isolation from other parts of the application. Each Sandbox is associated with an SSU STACK, and contains at least one Application Module, as well as a stack memory AP range. All LINKs associated with the Application Modules in the Sandbox have read-write access to the Sandbox stack memory; all other LINKs have no access. Each Sandbox is associated with one debug ZONE.

SysConfig defines a SECURE_GROUP in the linker command file for each Sandbox. This setting causes the linker to require protected calls for all function calls from other STACKs into the Sandbox STACK. By default, any unprotected call into a SECURE_GROUP causes the linker to generate an error. SysConfig provides an option to auto-generate trampolines and landing calls to satisfy the protected call requirement. When enabling this option, be sure to review the output linker map file to confirm that no undesired cross-STACK trampolines to untrusted code are generated.

Note

Cross-stack trampolines can add latency due to the requirement to save and restore CPU registers to or from stack memory, potentially impacting application performance. For best performance, implement protected function calls directly in application code by adding `__attribute__((c29_protected_call))` to the function definition.

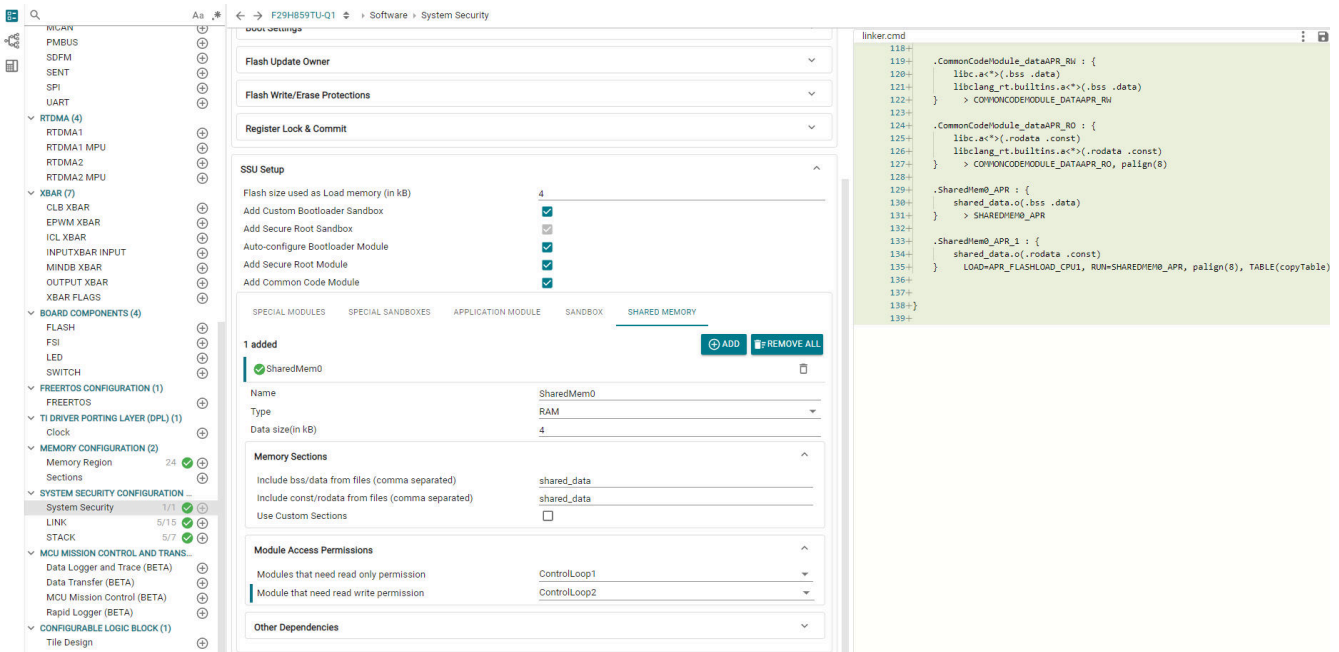
Note

STACK1 configuration can be accessed under the *Special Modules* tab.

7.3.5 Adding Shared Memory

Shared memory regions are special access protection regions (APRs) that are accessible by multiple application modules. In SysConfig, shared memories can be added by selecting the *Shared Memory* tab on the *System Security* page, and clicking the *Add* button. Multiple shared memories can be added, limited by the total number of APRs available on the CPU (including APRs that have been defined for the various Application Modules). For each shared memory, source files can be included; SysConfig adds the .bss, .data, .const, and .rodata sections from these files as configured. Custom section names can also be defined to be included.

For each shared memory, you can select which application modules require read-only permission, and which modules require write permission. SysConfig automatically configures the APR permissions as defined for the LINK for each module.



The screenshot displays the TI Configurator interface for System Security configuration. The 'SHARED MEMORY' section is expanded, showing a table with one entry:

Name	Type	Data size(in kb)
SharedMem0	RAM	4

Below the table, the 'Memory Sections' configuration is shown:

- Include bsa/data from files (comma separated): shared_data
- Include const/rodata from files (comma separated): shared_data
- Use Custom Sections:

The 'Module Access Permissions' section is also visible:

- Modules that need read only permission: ControlLoop1
- Module that need read write permission: ControlLoop2

On the right side, a snippet of linker.cmd shows the memory layout for SharedMem0_APR:

```

118+ .CommonCodeModule_dataAPR_R0 : {
119+ | libc.a*(.bss .data)
120+ | libclang_rt.builtins.a*(.bss .data)
121+ | } > COFFXMCODEMODULE_DATAAPR_R0
122+
123+ .CommonCodeModule_dataAPR_R0 : {
124+ | libc.a*(.rodata .const)
125+ | libclang_rt.builtins.a*(.rodata .const)
126+ | } > COFFXMCODEMODULE_DATAAPR_R0, palign(8)
127+
128+
129+ .SharedMem0_APR : {
130+ | shared_data.o(.bss .data)
131+ | } > SHAREDMEM0_APR
132+
133+ .SharedMem0_APR_1 : {
134+ | shared_data.o(.rodata .const)
135+ | } LOAD=APR_FLASHLOAD_CPU1, RUN=SHAREDMEM0_APR, palign(8), TABLE(copyTable)
136+
137+
138+ }
139+
  
```

Figure 7-5. Shared Memory Configuration Example

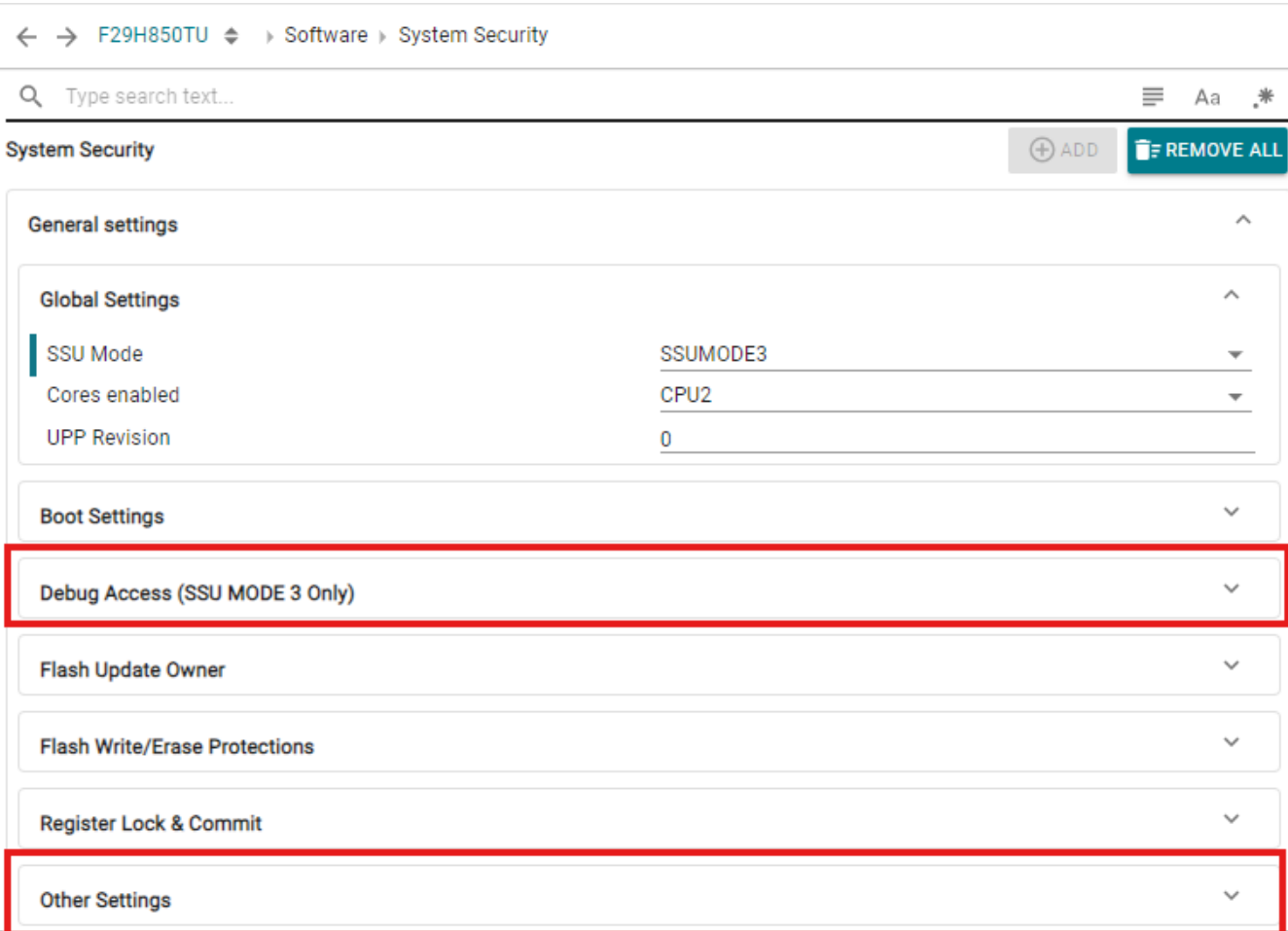
8 Debug Authorization

8.1 Password-Based Unlock

Within the *System Security* module, the SSUMODE can be selected. As shown in [Figure 8-1](#) in SSUMODE 3 secure debug and firmware updates are supported. Thus, to enable debug authorization based on a password entry and match, first select SSUMODE3. After SSUMODE3 has been selected, a section named 'Debug Access (SSU MODE 3 Only)' appears. Within this section, passwords for partial and full debug can be defined for each of the ZONEs. C29 Debug only has one password, hence there is only one entry. These passwords get loaded to SECCFG at the time of boot.

Note

Please verify that the correct access permissions have been set in place when using SSUMODE3. The recommendation is that a user test permissions setup while in SSUMODE2 first. If the device is transitioned to SSUMODE3 without the proper APR configuration then code won't execute properly due to enforced protections.



The screenshot shows the 'System Security' configuration page for device F29H850TU. The breadcrumb trail is 'Software > System Security'. A search bar is present at the top. The 'System Security' section has '+ ADD' and 'REMOVE ALL' buttons. The configuration is organized into several sections:

- General settings** (expanded):
 - Global Settings** (expanded):
 - SSU Mode: SSUMODE3
 - Cores enabled: CPU2
 - UPP Revision: 0
 - Debug Access (SSU MODE 3 Only)** (highlighted with a red box)
 - Flash Update Owner
 - Flash Write/Erase Protections
 - Register Lock & Commit
 - Other Settings** (highlighted with a red box)

Figure 8-1. SSU Mode 3 additional settings

Once the device is in SSUMODE3, all debug access is closed. The passwords setup in SysConfig need to be scanned into the device via the SEC-AP interface to open up the proper debug for the different zones. To do this, modify the SEC-AP gel file with the passwords that were used within SysConfig. To access the SEC-AP gel file in CCS, use the following steps:

- Right-click on any of the visible cores within CCS Debug view in the connections pane and select 'Show all cores'

- Connect to the C29 SEC-AP core
- Go to 'View' in the CCS toolbar menu and choose 'GEL Files'
- Open the provided SEC-AP gel file
- Modify the passwords for the different zones as provided in SysConfig. Writes for the 128-bit passwords are in the following order: [31:0] bits, [63:32] bits, [95:64] bits, [127:96] bits.
- Disconnect from the SECAP interface and reconnect. The passwords are scanned in on connection. Thus, C29 and Zones open if the passwords scanned in through SECAP interface match what was configured through SysConfig.

Upon successful debug open, a connection to the respective C29 cores is possible.

9 Debugging the SSU

9.1 Debugging Build Errors

Table 9-1. Possible Build Errors

Error Example	Description	Resolution
"syscfg/linker.cmd", line 73: error #10099-D: program will not fit into available memory, or the section contains a call site that requires a trampoline that can't be generated for this section, or the section contains padded functions. placement with alignment fails for section ".LINK2_codeAPR_Flash" size 0x5052 page 0. Available memory ranges: LINK2_CODEAPR_FLASH size: 0x2000 unused: 0x1fe0 max hole: 0x1fe0	The defined APR size is not enough.	If the allocated memory for any of the APR types (Flash, RAM, RW, or RO data) is not sufficient, there will be a build error at the time of project build. In order to determine the correct size for an APR, convert the provided size (0x5052) to decimal (20,562). This indicates that the total size of the APR is ~20k. APRs are bound to 4kB so for this particular case, the APR size would need to be set to 24kB.
error #10483-D: unprotected calls from output section ".comms_Module_codeAPR_Flash" to unprotected symbol "UART_writeCharArray" are not allowed: SECURE_GROUP mismatch; the call is in SECURE_GROUP "STACK2_STACK" and the callee is in SECURE_GROUP "sbox_CommonCode_STACK_COMMONCODE"	Cross-stack call is not accompanied by the appropriate entry and exit instructions.	<p>Cross-STACK calls must always be accompanied by appropriate entry and exit instructions. The SSU Tool has an option to specify how to handle unprotected calls. There are three options:</p> <ul style="list-style-type: none"> • Generate Linker Error: Throws an error if there are no appropriate entry and exit instructions (to be setup by the user). • Add linker trampolines and landing pads (Secure): Compiler inserts the appropriate entry and exit instructions for all of the calls to the selected STACK. Registers are preserved across the cross-STACK call. • Add linker trampolines and landing pads (Unsecure, Faster): Compiler inserts the appropriate entry and exit instructions for all of the calls to the selected STACK. Registers are not preserved across the cross-STACK call. <p>Choose the appropriate option to handle cross-STACK entry and exit instructions.</p>

9.2 Debugging Runtime Errors

Various errors, from the SSU and other peripherals are captured and logged through F29x's Error Aggregator Module (EAM) and Error Signaling Module (ESM). For more information on how to view the errors coming from the EAM and ESM in CCS, please refer to the [F29x Error Handling and Debug Guide](#) application report.

Table 9-2. Possible SSU Runtime Errors

Error Example	Description	Resolution
C29xx_CPU1: Error during Flash Programming. Address 0x10D85000, FMSTAT (STATCMD on some devices) 0x00000000, Value 0x00000101 C29xx_CPU1: Filer Loader: Memory write failed: Unknown Error C29xx_CPU1: GEL: File: C:\Users..... ssu_ex1_mode2.out: Load failed.	The following error can be seen when the flash settings are not setup correctly for programming SECCFG.	<p>To allow SECCFG memory to be erased, verify the check of the "Allow NonMain Flash erase before loading data to Flash memory" checkbox within the <i>NonMain Erase Settings</i> section of the Flash Settings. Afterwards, re-flash your program</p> <hr/> <p style="text-align: center;">Note</p> <p>Verify this setting is unselected when not working with SECCFG programming</p> <hr/>

Table 9-2. Possible SSU Runtime Errors (continued)

Error Example	Description	Resolution
<p>CPU1_DW Errors (HP Error Addr = 0x60070018, LP Error Addr = 0x00000000, PC = 0x10010646) SECURITY VIOLATION</p>	<p>The following error indicates that the instruction located at program counter (PC) address 0x10010646 tried to write data to address 0x60070018, but was not able to due to not having sufficient permissions to do so.</p>	<p>To debug this issue, look up the PC address in the CCS Disassembly view (<i>View -> Disassembly</i>). This shows where the program counter is. This gives insights into the code execution (ie, which function and line of that function) the write access was prohibited. For this example, the program counter was located in the 'update_PSFb()' function located within LINK4. Next, open the memory browser in CCS (<i>View -> Memory</i>) and enter in the address 0x60070018). This provides the content in memory that the instruction was going to write to. In this case, 0x60070018 corresponds to UARTA regs. The memory map in the F29x DS can also be utilized to correlate the provided error addresses with register set. In this case, user can look up 0x6007_0000 in the memory map. Thus, LINK4 needs to be given R/W access to UARTA peripheral in order for the code within the 'update_PSFb()' function to write to the UARTA registers.</p> <hr/> <p style="text-align: center;">Note</p> <p>There are many kinds of CPU1 errors such as PR, DR1, DR2, etc. For more details on what these individual types of errors mean please refer to the Error Aggregator chapter within the TRM or the F29x Error Handling and Debug Guide application report.</p> <hr/>
<p>SSU Errors (HP Error Addr = 0x3008000C, LP Error Addr = 0x00000000, PC = 0x00000000) CPU1_SSU_MMR_ACCESS_ERROR</p>	<p>The error showcases that the SSU was not allowed to access one of the SSU memory mapped registers (MMR).</p>	<p>To debug this issue, start by looking up the provided address in the CCS memory browser (<i>View -> Memory</i>). The address 0x3008000C corresponds to LINK2_AP_OVERRIDE. The 'Search' feature within CCS was used to locate all of the instances for which 'SSU_enableLink2APOverride()' was called within the project. In this example, the function 'SSU_enableLink2APOverride()' was used outside of LINK2, causing this error. Use a similar approach when encountering other SSU MMR access errors.</p>

Table 9-2. Possible SSU Runtime Errors (continued)

Error Example	Description	Resolution
During Flashing- "Warning: Target CPU may be stuck in persistent fault state"	Device is in SSUMODE2 and implementing memory protections.	<p>SSU mode needs to be changed to SSU Mode 1 when the SSU is not intended to be used with safety and security protections enabled. To do this, reprogram SECCFG to put device in SSUMODE1 and reprogram to BANKMODE0 in BANKMGMT.</p> <ul style="list-style-type: none"> • Load the default seccfg program to device found. The default seccfg program is located at this path: C:\ti\<f29h85x-sdk_version_x>\source\defseccfgbin • Provide a reset on the EVM or HW • Connect to target and verify that SSUMODE = 0x30 in the SSUGENREGS registers. • In the Flash Settings, program FLASH BANKMODE to BANKMODE 0 or verify that the desired BANKMODE is set. • Disconnect from target in CCS and issue an XRSN on EVM (controlSOM has an XRSN button) if the bankmode was changed. • Connect to target and verify that BANKMODE = 0x3 in the SSUGENREGS or the desired BANKMODE got set correctly.

10 SSU Frequently Asked Questions (FAQ)

Table 10-1. SSU Questions and Answers

#	Question	Answer
1	If modifying an APR during runtime, is it necessary to disable the APR before it is changed?	No, it can be changed while it is enabled.
2	What does the LINK2 AP Override Feature do?	LINK2 gets read and write (R/W) access to all of the APRs that have been enabled or configured even if LINK2 has not been provided with those permissions.
3	Is LINKID a required field when modifying the range for both code and data APRs?	No, LINKID is only required if XE bit is set (code APR).
4	Is it possible to change the START ADDRESS, END ADDRESS, and LINKID fields within the SSU APR registers dynamically during runtime?	Yes, it can be updated during runtime. Only LINK2 has access to modify these registers. LINKx_CFG and STACKy_CFG cannot be modified during runtime.
5	If a file is included, as part of a Module configuration, is the entirety of the file getting captured through APRs?	Yes, code and variables within the file are set as part of APRs for that LINK through the SSU Tool in SysConfig.

11 Summary

The C29x SSU enables advanced safety and security features for real-time control applications, including a novel context-sensitive memory and peripheral-protection feature that eliminates software overhead while switching between tasks or servicing interrupts, and advanced debug security options. With the easy-to-use SysConfig tool, system designers can divide an application into multiple partitions for safety and security isolation, and automatically allocate object files, libraries, and peripherals to each of these partitions. Shared memory resources can also be defined to be used by multiple application modules. SysConfig automatically handles the allocation of memories across the device to various application partitions, with full support for multicore configurations. The tool generates all required output files for implementing the desired protection policy, which can then be built into an application .out image.

12 References

1. Texas Instruments, [C2000™ SysConfig Application Note](#)
2. Texas Instruments, [TI Resource Explorer: C2000™ real-time microcontrollers](#)
3. Texas Instruments, [Code Composer Studio \(CCS\) IDE](#)
 - Integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processor products
 - SysConfig tool is delivered integrated in CCS (built-in SysConfig support)
4. Texas Instruments, [SysConfig Standalone Version](#)
 - SysConfig standalone version can be used alongside other IDEs that do not have a built-in SysConfig tool

13 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from November 30, 2024 to October 31, 2025 (from Revision * (November 2024) to Revision A (October 2025))

	Page
• Updated images to reference the latest Sysconfig version (1.25 at the time of this release).....	5
• System Design : Moved content into tables for Predefined STACKs and LINKs per CPU as well as for the SSU partitioning example.....	7
• Flash SECCFG Region : Added a note for flash bank modes.....	10
• Enabling System Security Configuration : Added notes to discuss warning when adding CommonCodeMode_Link and making modifications for the linker command file.....	11
• Configuring Application Modules : Added a note to discuss INTs and RTINTs with SSU.....	12
• LINK2 Configuration : Added table listing the mandatory peripheral access protection for LINK2.....	14
• LINK1 Configuration : Added table listing the mandatory peripheral access protection for LINK1.....	15
• Added the Debug Authorization chapter.....	18
• Added the Debugging the SSU chapter.....	20
• Added the SSU Frequently Asked Questions (FAQ) chapter.....	23

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated