Programming Microchip SAM L11 in Arm MDK

MDK Tutorial

AN315, Summer 2018, V 1.0

feedback@keil.com

The latest version of this document is here: www.keil.com/appnotes/docs/apnt_315.asp

Abstract

The Application Note describes how to partition and use the secure and non-secure domains of Microchip's Arm® Cortex® -M23 based SAM L11 device in Arm MDK. The details are explained with an example project present in the SAML11 device family pack.

Prerequisites

MDK v5.26 provides support for creating and debugging secure and non-secure applications for ARMv8-M based devices, including SAML10 and SAML11 devices from Microchip.

To be able to use the examples provided in this application note, you need to have a valid MDK license. The MDK-Essential edition allows you to develop and debug non-secure applications on Cortex-M23/M33 devices while the MDK-Plus and MDK-Professional editions also allow you to develop and debug secure applications.

Note: there is an evaluation version available for MDK-Professional.

Additionally, the following software packs are required:

- ARM.CMSIS.5.0.1.pack (or higher)
- Keil.SAML11_DFP-1.0.81 (or higher)

Contents

Programming Microchip SAM L11 in Arm MDK 1

Abstract 1

Prerequisites 1

Introduction 2

Secure and non-secure domains with TrustZone for ARMv8-M 2

SAM L11 system partitioning 3

Configuring SAM L11 device in MDK 3

Example: TrustZone for SAML11 No RTOS 5

Multi-project workspace setup 5

Secure project configuration 6

Non-secure project configuration 8

Summary 11

Useful links 11

Introduction

The SAM L11 Microcontroller family from Microchip is based on the Arm® Cortex®-M23 core, featuring TrustZone® for Armv8-M, a programmable environment that provides hardware isolation between certified libraries, IP and application code.

ARM® TrustZone® technology is a System on Chip (SoC) and CPU system-wide approach to security. The TrustZone for ARMv8-M security extension is optimized for ultra-low power embedded applications. It enables multiple software security domains that restrict access to secure memory and I/O to trusted software only.

TrustZone for ARMv8-M:

- preserves low interrupt latencies for both secure and non-secure domains.
- does not impose code overhead, cycle overhead or the complexity of a virtualization-based solution.
- introduces efficient instructions for calls to the secure domain with minimal overhead.

In addition to TrustZone, the SAM L11 security features include an on-board cryptographic module supporting Advanced Encryption Standard (AES), Galois Counter Mode (GCM) and Secure Hash Algorithm (SHA). The secure boot and secure key storage with tamper detection capabilities establish hardware root of trust. It also offers secure bootloader for secure firmware upgrades.

To show a real-world example, in this tutorial a SAML11 Xplained Pro Evaluation Kit is used as a target for the Subsequent Chapters provide details for this example.

Secure and non-secure domains with TrustZone for ARMv8-M

Arm TrustZone for Cortex-M enables the system (Flash, RAM, and peripherals) and the software code to be partitioned in Secure and Non-Secure domains.

Secure software is executed in a secure state and can access both Secure and Non-secure memories and resources, while Non-Secure software is running in non-secure state and can only access Non-secure memories and resources.

Figure 1 shows an example of an embedded application split into a Non-Secure **User project** and a Secure **Firmware project**.

System Start: after power-on or reset, an ARMv8-M system starts code execution in the secure state.

User Application: control can be transferred to the non-secure state to execute user code. This code can only call functions in the secure state, which are marked for execution with the SG (secure gate) instruction and additional memory attributes. Any other attempt to access memory or peripherals that are assigned to the secure

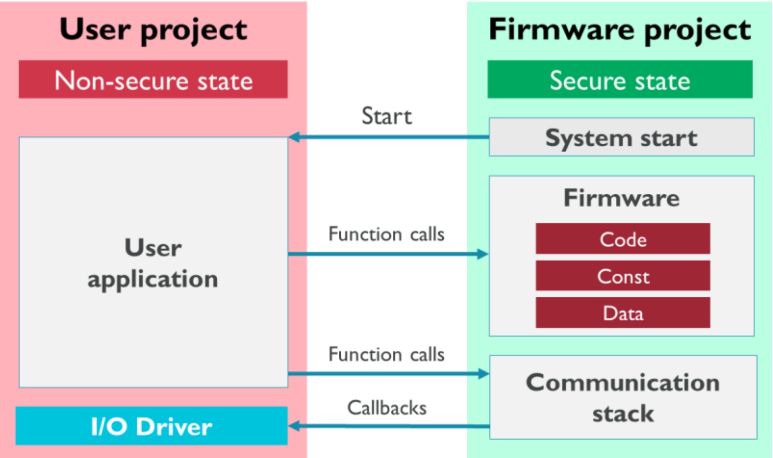


Figure 1 Secure/non-secure embedded application

Firmware callbacks: code running in the secure state can execute code in the non-secure state using call-back function pointers. For example, a communication stack (protected firmware) could use an I/O driver that is configured in user space.

For a real use-case refer to Example: TrustZone for SAML11 No RTOS.

SAM L11 system partitioning

Memory partitioning

Memory partitioning to Non-Secure and Secure domains is done on SAM L11 devices using a centralized **Implementation Defined Attribution Unit (IDAU),** which is a hardware unit external to the core. The Security Attribution Unit (SAU) is not supported.

The IDAU is used to indicate the processor if a memory region is **Secure (S)**, **Non-secure Callable (NSC)**, or **Non-secure (NS)**. It can also mark a memory region to be exempted from security checking.

The IDAU security configuration is done performed via settings in Non-Volatile Memory User Row (UROW), which are read after each reset and are loaded into their respective registers. See section Configuring SAM L11 device in MDK explaining how this configuration is done in µVision.

Peripheral partitioning

The **Peripheral Access Controller (PAC)** configures the security privileges for each individual peripheral in the system. Each peripheral can only be configured either in Secure or in Non-Secure mode.

The PAC security configuration is performed via settings in Non-Volatile Memory User Row (UROW), which are read after each reset and are loaded into their respective registers. See section Configuring SAM L11 device in MDK explaining how this configuration is done in µVision.

Debug access

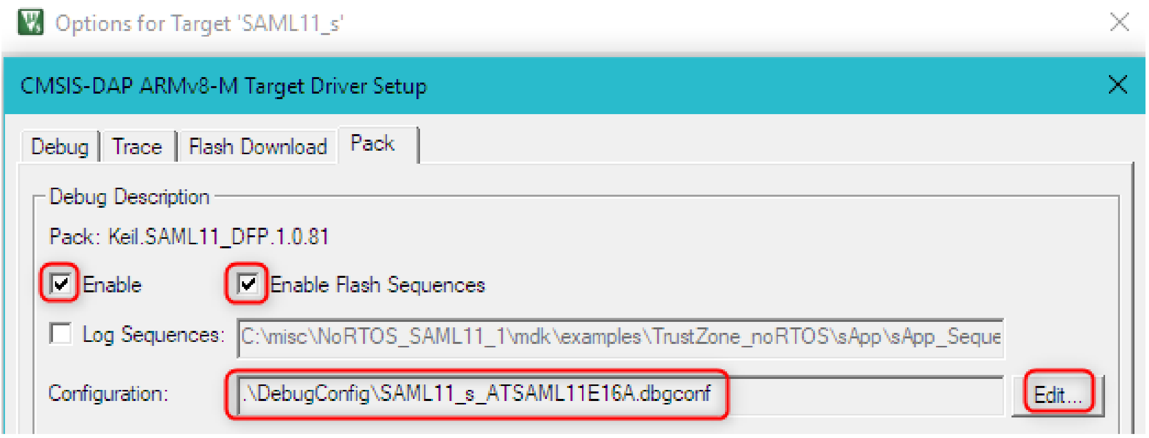
The SAM L11 allows three debug access levels (DAL):

- DAL2: Highest debug level access with no restrictions in term of memory and peripheral accesses.
- DAL1: Access is limited to the Non-Secure memory regions. Secure memory regions accesses are forbidden.
- DAL0: No access is authorized except from the debugger communicating with the Boot ROM in interactive mode.

Configuring SAM L11 device in MDK

The SAM L11 device can be configured in MDK via special debug configuration file. File template is included in the device family pack for SAM L11 and is also present in the example projects for both secure and non-secure applications in .*DebugConfig* directory. The debug configuration file will be copied automatically to the project if not present yet, or in case a new project is created.

The use of such debug configuration file is given via **Options for Target… → Debug → Settings.. → Pack** tab


Options for Target 'SAML11_s'
CMSIS-DAP ARMv8-M Target Driver Setup
Debug
Trace
Flash Download
Pack
Debug Description
Edit...

Pack: Keil.SAML11_DFP.1.0.81

☑ Enable ☑ Enable Flash Sequences

□ Log Sequences: C:\misc\NoRTOS_SAML11_1\mdk\examples\TrustZone_noRTOS\sApp\sApp_Seque

Configuration: .\DebugConfig\SAML11_s_ATSAML11E16A.dbgconf

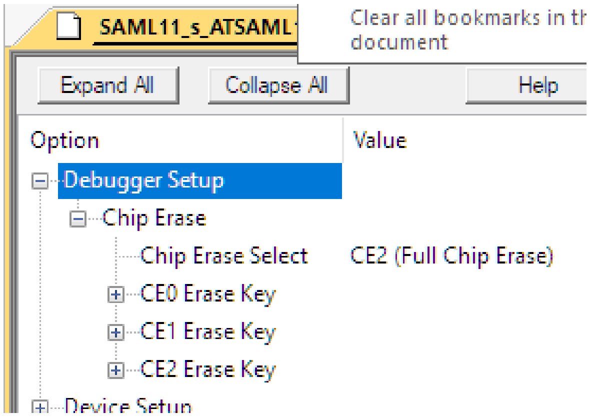
For device programming and debugging the µVision relies on the Debug Descriptions as well as Flash Sequences provided with the SAM L11 device family pack. Hence corresponding options shall be enabled.

Press *Edit..* to open the file in the editor.

The file is best viewed with Configuration Wizard. It organizes configurable parameters into two main categories: Debugger Setup and Device Setup.

Debugger setup parameters

This category contains the chip erase parameters that shall be used by µVision during device programming operations. In particular, the Chip Erase level and Chip Erase keys shall be configured here. Note that these parameters are not being programmed to the target device.



Option	Value
⊟ Debugger Setup	
⊟ Chip Erase	
Chip Erase Select	CE2 (Full Chip Erase)
⊞ CE0 Erase Key	
⊞ CE1 Erase Key	
⊞ CE2 Erase Key	
⊞ Device Setup	

By default, the SAML11 devices are delivered with the Chip Erase keys set at “All 1s” and so are the default values set in the debug configuration file. Note that setting a Chip Erase key to “All 0s” disables corresponding chip erase option. This can be especially critical for CE2 as then it is impossible to get to DAL2 level anymore.

Device setup parameters

In this category user can configure following parameters:

- Debug Access Level
- NVM User Configuration Row (UROW) parameters
- NVM Boot Configuration Row (BOCOR) parameters

These are persistent parameters that get programmed into Non-Volatile Memory (NVM) of the device during debug and load operations.

Some parameters (especially memory and peripheral partitioning) are accessible only by a secure application, while others can be modified also by a non-secure application. The access rights Details about access rights are provided in SAML11 device datasheet.

Note that the parameters in the debug configuration file are not updated from the connected device. µVision only replaces the current device configuration with these values during the next load or debug operation.

Debug Access Level (DAL)

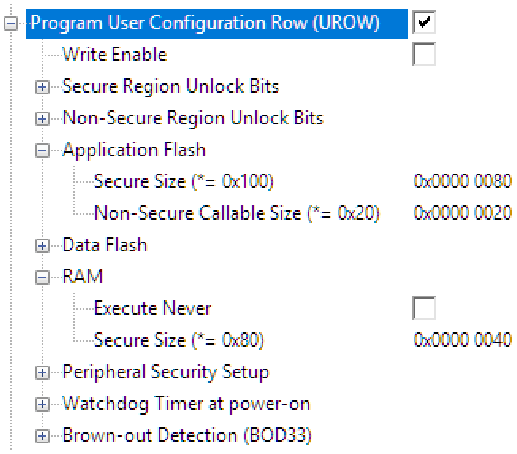
Details about the operation of all debug access levels are provided in SAML11 device datasheet.

NVM UROW settings

NVM User Configuration Row (UROW) contains parameters for the watchdog and brown-out detection as well as settings for memory and peripheral partitioning into secure and non-secure domains.

If programming of the UROW is required, then the *Program User Configuration Row (UROW)* shall be enabled to become active. The parameter *Write Enable* should be enabled as well. After that, individual parameters can be modified.

A detailed explanation of all parameters is available in SAML11 device datasheet.



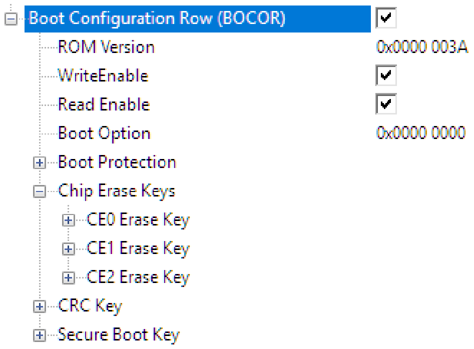
Program User Configuration Row (UROW) ☑
- Write Enable □
- Secure Region Unlock Bits
- Non-Secure Region Unlock Bits
- Application Flash
  - Secure Size (*= 0x100) 0x0000 0080
  - Non-Secure Callable Size (*= 0x20) 0x0000 0020
- Data Flash
- RAM
  - Execute Never □
  - Secure Size (*= 0x80) 0x0000 0040
- Peripheral Security Setup
- Watchdog Timer at power-on
- Brown-out Detection (BOD33)

NVM BOCOR settings

NVM Boot Configuration Row (BOCOR) contains parameters for the boot flash partitioning as well as Chip Erase Keys.

If programming of the BOCOR is required, enable *Program Boot Configuration Row (BOCOR)* to become active. Continue with modifying individual parameters.

A detailed explanation of all parameters is available in SAML11 device datasheet.

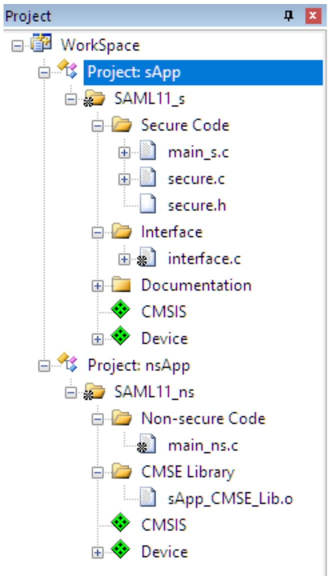


Boot Configuration Row (BOCOR) ☑
- ROM Version 0x0000 003A
- WriteEnable ☑
- Read Enable ☑
- Boot Option 0x0000 0000
- Boot Protection
- Chip Erase Keys
  - CE0 Erase Key
  - CE1 Erase Key
  - CE2 Erase Key
- CRC Key
- Secure Boot Key

Example: TrustZone for SAML11 No RTOS

This example project shows a basic TrustZone configuration on a SAM L11 device. The application demonstrates function calls between secure and non-secure states. The secure application sets up the system and starts the non-secure application which calls a secure function from the non-secure state. All variables used in this application can be viewed in the µVision Debugger Watch window.

Multi-project workspace setup



Project
- WorkSpace
  - Project: sApp
    - SAML11_s
      - Secure Code
        - main_s.c
        - secure.c
        - secure.h
      - Interface
        - interface.c
      - Documentation
      - CMSIS
      - Device
  - Project: nsApp
    - SAML11_ns
      - Non-secure Code
        - main_ns.c
      - CMSE Library
        - sApp_CMSE_Lib.o
      - CMSIS
      - Device

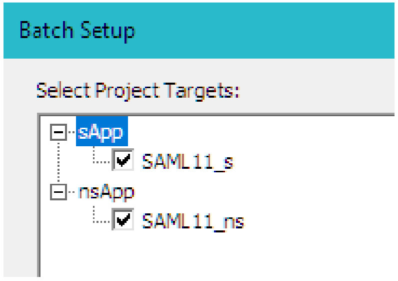
To demonstrate the split into secure and non-secure domains the example application is organized as a multi-project workspace. Once you have opened the project *NoRTOS.uvmpw*, you will see two projects in the *Project* window. The secure project is called *sApp* and the non-secure project is named *nsApp*.

Project Build

The workspace is configured so that a batch operation (build, clean, rebuild) is performed over both projects – first *sApp* and then *nsApp*.

To build both projects at once, go to **Project → Batch Build…** or use the batch build icon. This will generate two firmware images: one image for the secure project (*sApp\Objects\sApp.axf*) and one image for the non-secure project (*nsApp\Objects\nsApp.axf*).

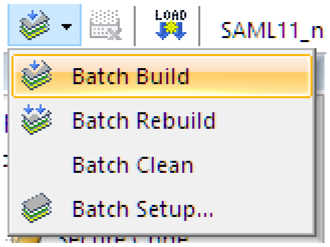
If the *sApp* project is set as active, both secure and non-secure projects will be loaded during flash load and debug. See Secure project configuration section for details.



Batch Setup

Select Project Targets:

- sApp
  - ☑ SAML11_s
- nsApp
  - ☑ SAML11_ns

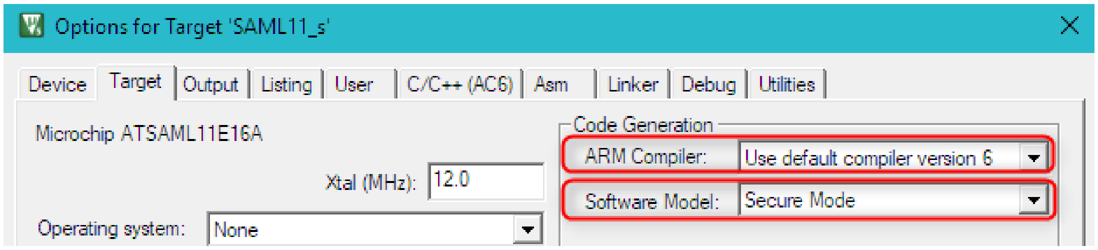


Secure project configuration

The secure project *sApp* requires special configuration to ensure that it gets properly compiled and placed into the secure memory of the device, as well as to enable secure project debug. In addition, device partitioning can be done via the debug configuration file.

Secure mode operation

Secure operation is configured via **Options for Target…** ( ) → **Target** tab → **Software Model→Secure Mode**



Microchip ATSAML11E16A

Xtal (MHz): 12.0

Operating system: None

Code Generation

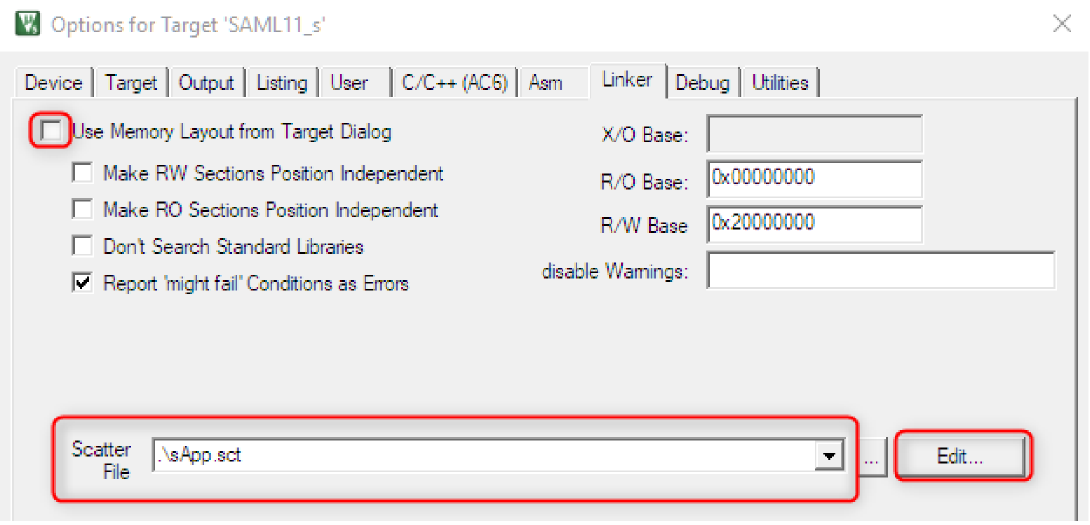
ARM Compiler: Use default compiler version 6

Software Model: Secure Mode

Note that Cortex-M23 device is supported only by Arm compiler version 6.

Memory layout

Scatter file *sApp.sct* is used in **Options for Target…** ( ) → **Linker** tab→ **Scatter File** and describes where the project code and data are placed in the device memory.



□ Use Memory Layout from Target Dialog
□ Make RW Sections Position Independent
□ Make RO Sections Position Independent
□ Don't Search Standard Libraries
☑ Report 'might fail' Conditions as Errors

X/O Base:
R/O Base: 0x00000000
R/W Base 0x20000000
disable Warnings:

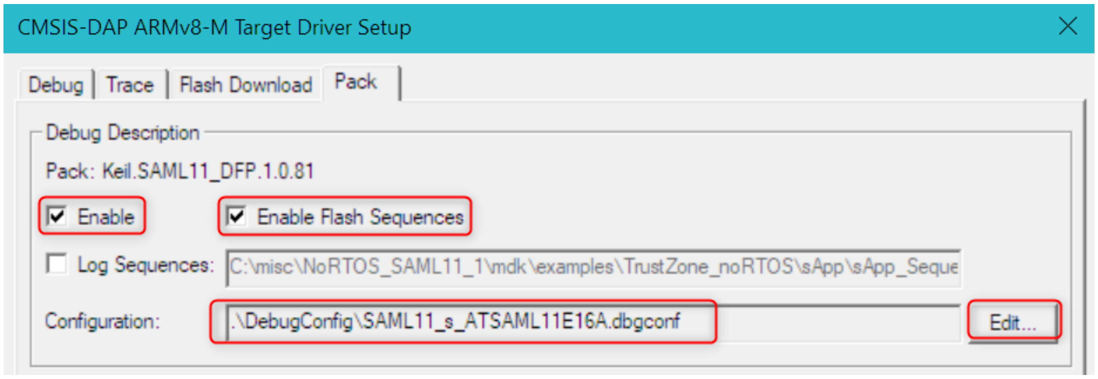
Scatter File: .\sApp.sct

The code from *interface.c* file needs to be accessible from the non-secure project and hence is placed into the non-secure callable area of the Flash memory marked by *(Veneer$$CMSE). The rest of the code is placed in the secure area of the Flash. The secure area of the RAM is used for data.

Note: The *sApp.sct* file itself doesn't partition device memory into secure and non-secure domains. This is done via Debug Configuration file as described below. For correct program operation, the scatter file must use the same memory partitioning as done in the Debug Configuration file.

Debug configuration file

Device-specific debug options as well as device partitioning into secure and non-secure domains is done via the *sApp\.DebugConfig\SAML11_s_ATSAML11E16A.dbgconf* file. The use of such debug configuration file is given via **Options for Target… → Debug → Settings.. → Pack** tab



Debug Description
Pack: Keil.SAML11_DFP.1.0.81
☑ Enable ☑ Enable Flash Sequences
□ Log Sequences: C:\misc\NoRTOS_SAML11_1\mdk\examples\TrustZone_noRTOS\sApp\sApp_Seque
Configuration: .\DebugConfig\SAML11_s_ATSAML11E16A.dbgconf

Note that "Enable" is checked to allow use of Debug Descriptions from the SAM L11 device family pack. To enable flash programming corresponding option "Enable Flash Sequences" shall be checked as well.

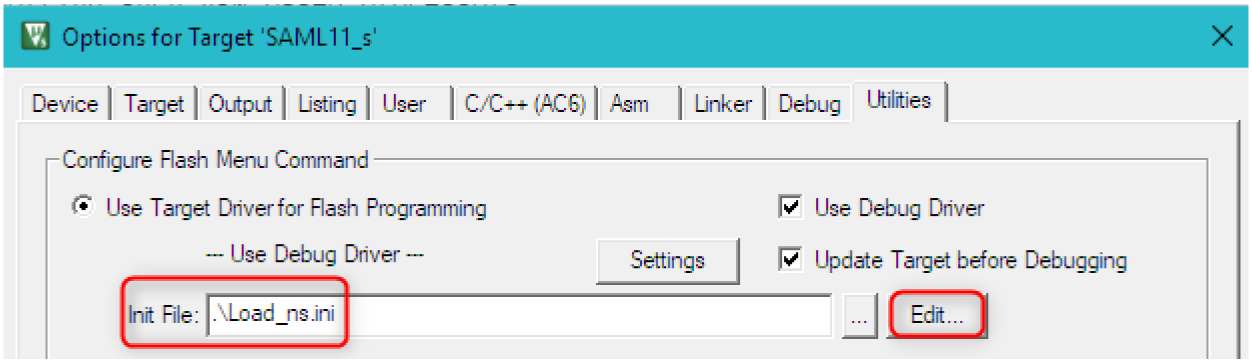
The section Configuring SAM L11 device in MDK explains the concept in detail. In this example, the important settings for the correct program operation are:

- Chip Erase is set to CE2 to allow full chip erase from the *sApp* project
- Chip Erase keys are set to all '1' as they are shipped by default in silicon
- Debug Access Level is set to DAL2 providing secure debug
- Program User Configuration Row (UROW) is enabled:
  - Write Enable bit is enabled to allow UROW changes
  - Application Flash memory is configured to have 32 KB of secure area and 400 B of non-secure callable area.
  - RAM memory is configured to have 8 KB of secure area.

    - Dataflash is configured to have 8 KB of secure area (not used in this project).
    - All peripherals are kept as secure, but none of them are used in the project.
- Program Boot Configuration Row (BOCOR) is enabled
    - Default values are kept without changes.

Loading both projects into Flash

By default, the secure project *sApp* is configured in a way that during the load operation (via **Flash → Download** or with icon) the *nsApp* program is also loaded into Flash. This is done via the *Load_ns.ini* script file in the **Options for Target…** ( ) **→ Utilities** tab:



Configure Flash Menu Command

● Use Target Driver for Flash Programming

--- Use Debug Driver ---

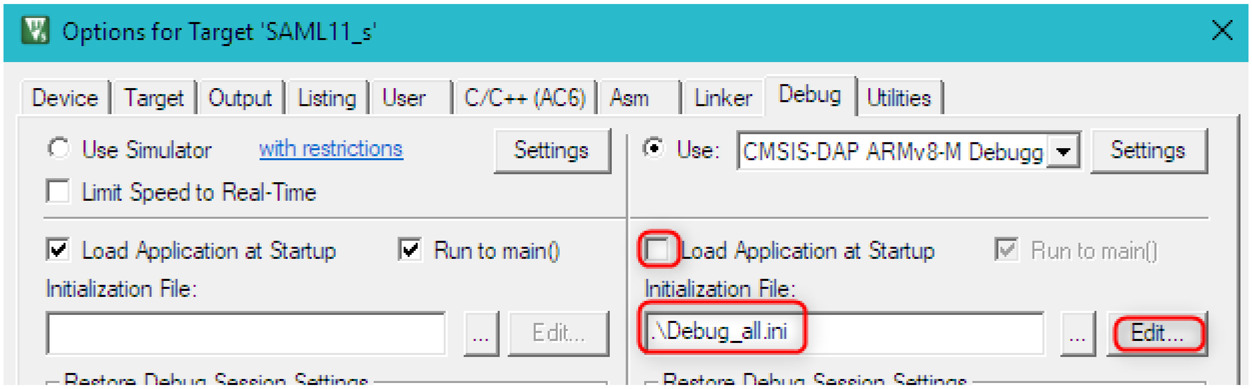
☑ Use Debug Driver

☑ Update Target before Debugging

Init File: .\Load_ns.ini

Debugging both projects

By default, the secure project *sApp* is configured so that during debugging ( ) also the image of the *nsApp* is loaded and thus can be debugged. This is done via the *Debug_all.ini* script file provided in the **Options for Target…** ( ) **→ Debug** tab:



○ Use Simulator

☐ Limit Speed to Real-Time

☑ Load Application at Startup

☑ Run to main()

Initialization File:

● Use: CMSIS-DAP ARMv8-M Debugg

☐ Load Application at Startup

☑ Run to main()

Initialization File: .\Debug_all.ini

Note that the option "Load Application at Startup" is disabled in this case.

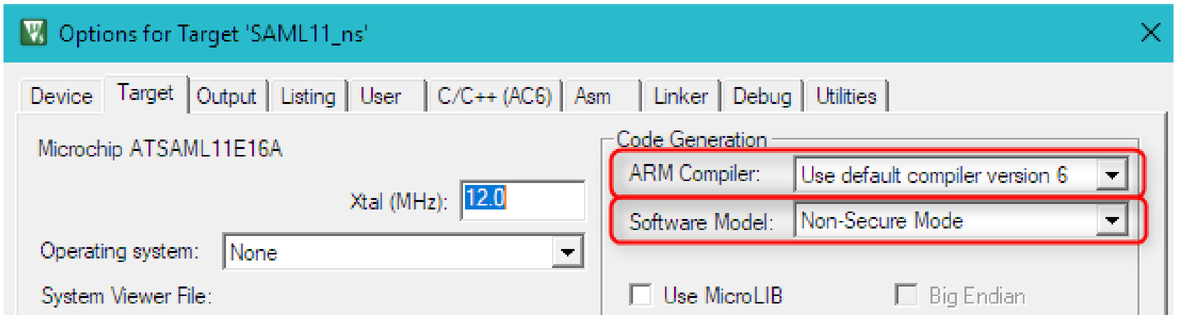
As mentioned in section Configuring SAM L11 device in MDK, for device programming and debugging the µVision relies on the Debug Descriptions as well as Flash Sequences provided with the SAM L11 device family pack. Hence corresponding options in the **Options for Target… → Debug → Settings.. → Pack** tab shall be kept enabled.

Non-secure project configuration

The non-secure project *nsApp* also requires configuration to ensure that it gets properly compiled and placed into the non-secure memory of the device. It is able to use interface shared by the secure project as defined in *interface.h*.

Non-Secure mode operation

Operation in non-secure state is configured via the **Options for Target…** ( ) → **Target** tab → **Software Model→Non-Secure Mode**



Microchip ATSAML11E16A

Xtal (MHz): 12.0

Operating system: None

System Viewer File:

Code Generation

ARM Compiler: Use default compiler version 6

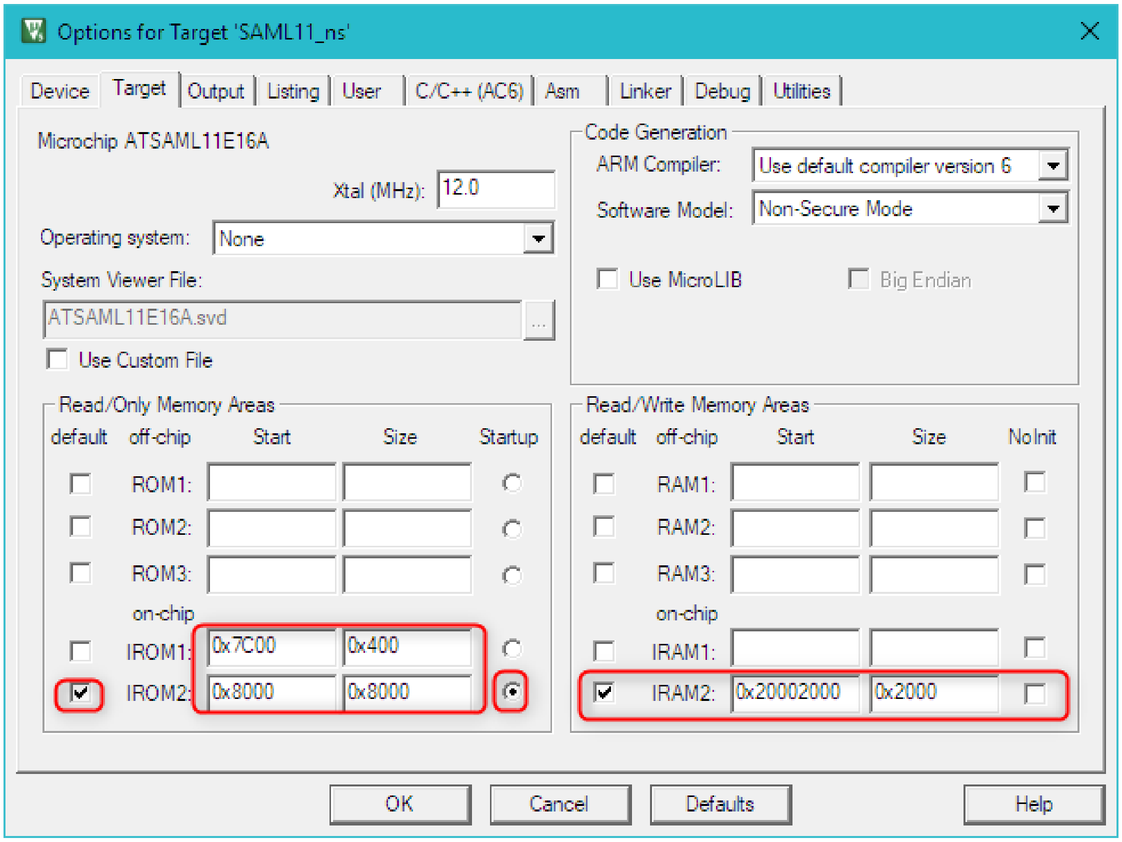
Software Model: Non-Secure Mode

☐ Use MicroLIB ☐ Big Endian

Note that Cortex-M23 device is supported only by Arm compiler version 6.

Memory layout

As the memory configuration of the non-secure project is simple, it can be configured via the memory layout GUI provided in **Options for Target…** ( ) → **Target.** Alternatively, a custom scatter file can be used as well.



Microchip ATSAML11E16A

Xtal (MHz): 12.0

Operating system: None

System Viewer File: ATSAML11E16A.svd

☐ Use Custom File

Code Generation

ARM Compiler: Use default compiler version 6

Software Model: Non-Secure Mode

☐ Use MicroLIB ☐ Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☐	IROM1:	0x7C00	0x400	○
☑	IROM2:	0x8000	0x8000	●

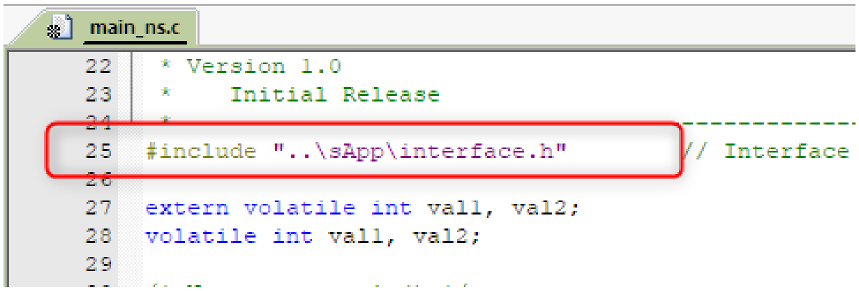
Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☐	IRAM1:			☐
☑	IRAM2:	0x20002000	0x2000	☐

The non-secure callable area of the flash is defined here as well (IROM1). For correct operation, the memory layout needs to correspond to the one actually configured on the device.

Interface to secure application

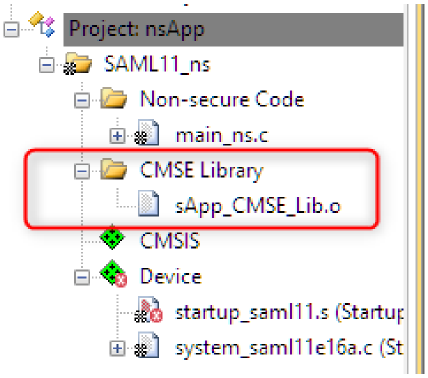
The communication with the secure application is done via the interface provided by the *sApp*. To access it, the *interface.h* file is included in *main_ns.c* file where interface functions *secure_func1* and *secure_func2* are then called.



main_ns.c

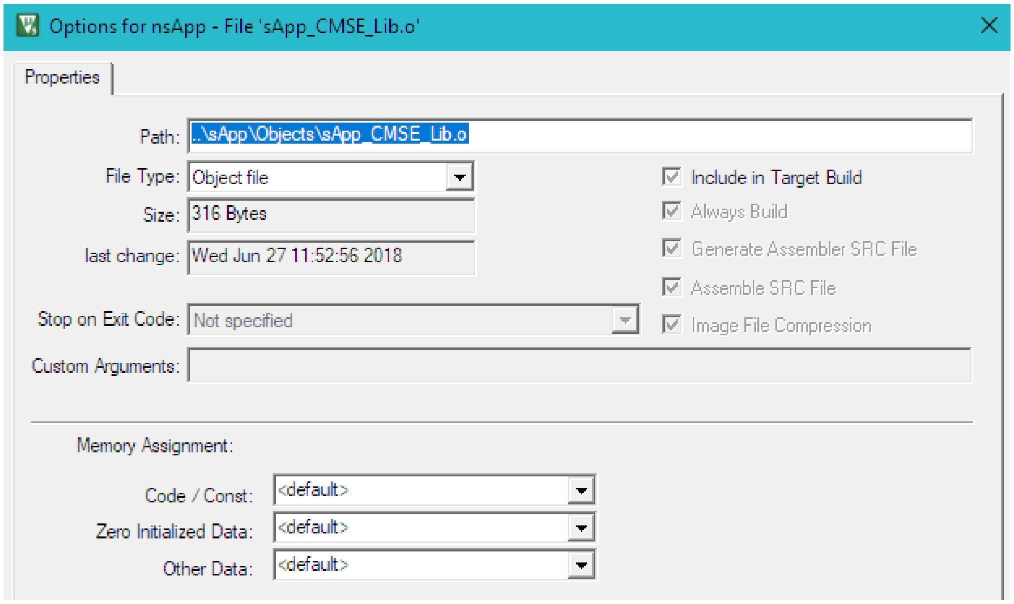
```
 * Version 1.0
 *    Initial Release
 *                                          -------------
#include "..\sApp\interface.h"              // Interface

extern volatile int val1, val2;
volatile int val1, val2;
```

The special object file *sApp_CMSE_Lib.o* that is present in the nsApp is generated by the secure project:



```
Project: nsApp
  SAML11_ns
    Non-secure Code
      main_ns.c
    CMSE Library
      sApp_CMSE_Lib.o
    CMSIS
    Device
      startup_saml11.s (Startup
      system_saml11e16a.c (St
```

The object file name is always created according to the rule `<projectname>_CMSE_Lib.o` in the `Objects` directory of the secure project. Right-click on the *sApp_CMSE_Lib.o* to see its options:



Options for nsApp - File 'sApp_CMSE_Lib.o'

Properties

Path: ..\sApp\Objects\sApp_CMSE_Lib.o

File Type: Object file

Size: 316 Bytes

last change: Wed Jun 27 11:52:56 2018

Stop on Exit Code: Not specified

Custom Arguments:

☑ Include in Target Build

☑ Always Build

☑ Generate Assembler SRC File

☑ Assemble SRC File

☑ Image File Compression

Memory Assignment:

Code / Const: <default>

Zero Initialized Data: <default>

Other Data: <default>

Debug configuration file

Device-specific debug options as well as device partitioning into secure and non-secure domains is done via the *nsApp\.DebugConfig\SAML11_ns_ATSAML11E16A.dbgconf* file. The use of such debug configuration file is given via **Options for Target… → Debug → Settings.. → Pack** tab.

Note that “Enable” is checked to allow use of Debug Descriptions from the SAM L11 device family pack. To enable flash programming corresponding option “Enable Flash Sequences” shall be checked as well.

The section Configuring SAM L11 device in MDK explains the concept in detail. In this example, the important settings for the program operation are:

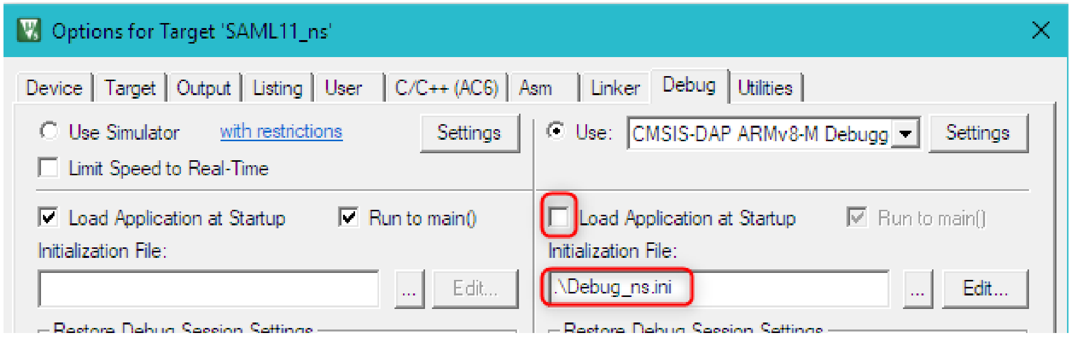
- Chip Erase is set to CE0 to allow erase of non-secure application only

- Chip Erase keys are set to all ‘1’ as they are on the chips shipped by default
- Debug Access Level is set to DAL2 allowing secure debug
- Program User Configuration Row (UROW) is unchecked so UROW doesn’t get changed
- Program Boot Configuration Row (BOCOR) is unchecked so BOCOR doesn’t get changed

Debugging the non-secure project

To debug non-secure application only, select the *nsApp* as an active project in the workspace or open the *nsApp\nsApp.uvprojx* file separately in µVision.

In the **Options for Target…** ( ) → **Debug** tab the debug initialization file is used to instruct the debugger that non-secure application shall be loaded.



Options for Target 'SAML11_ns'

○ Use Simulator with restrictions

□ Limit Speed to Real-Time

☑ Load Application at Startup ☑ Run to main()

Initialization File:

● Use: CMSIS-DAP ARMv8-M Debugg

□ Load Application at Startup ☑ Run to main()

Initialization File: .\Debug_ns.ini

Summary

This application note has described how to set up a secure and non-secure project for Microchip’s SAML11 device family. Dedicated Flash and debug support in MDK is available to successfully program these devices with their enhanced security features.

Useful links

- Microchip SAM L10 Cortex-M23 Tutorial provides a hands-on tutorial on Microchip SAM L10 Cortex-M23 processor and Keil µVision IDE
- Using TrustZone on ARMv8-M explains the features that are available in CMSIS and MDK to utilize the secure and non-secure domains in the ARMv8-M architecture.
- ARMv8-M Architecture Reference Manual gives a complete overview of the ARMv8-M architecture.
- Secure software guidelines for ARMv8-M based platforms lists the requirements when creating secure software for an ARMv8-M based platform.
- ARMv8-M Security Extensions: Requirements on Development Tools explains concepts implemented in compiler toolchains to support the ARMv8-M architecture.