Application Note

Realization of Password-Protected Debug Based on Software

Zoey Wei

ABSTRACT

For a low-cost device in M0 like MSPM0C110x, there are no hardware registers available to help implement SWD protection with a password, but only the fully open or completely disabled options can be chosen. However, with the popularity of cybersecurity, more and more applications, especially for the automotive market, are requesting this password-protected function. This application note proposes a new way to realize this security function based on software for a low-cost device, which does not have hardware to support the device.

Table of Contents

Trademarks

1 Cyber Security Requirement Introduction

With the deep integration of network communications, artificial intelligence, internet applications and other technologies with the automotive industry, intelligent connected vehicles have become the strategic direction of the automotive industry. And with the evolution of technology, the information security issues of intelligent connected vehicles are becoming increasingly severe, especially in recent years, there have been many automobile information security recall incidents, which have aroused great concern in the industry.

In June 2023, the United Nations World Forum for Harmonization of Vehicle Regulations (abbreviated as UN/WP.29) issued the world's first mandatory automobile information security regulation R155, namely Cybersecurity.

The series of regulations requires that all models in all EU countries and other OECD countries must pass relevant certification from July 2024. The regulations clearly state that there are process measures to control related risks throughout the vehicle life cycle, including that vehicle manufacturers must have measures to monitor security threats and detect and prevent cyber attacks.

As for China, although this is not mandatory to meet information security requirements, the cars must pass relevant certification as long as the cars are sold in these countries. And, with the increasing attention paid to information security, the demand for this function is also growing.

1.1 Cyber Security Requirement for MSPM0

Based on cybersecurity, automobile manufacturers put forward the following requirements for MCU chips:

1. Need to disable SWD once power up, and only with password can access.
2. No other communication and external trigger can be used, which means BSL is not allowed in this situation.
3. Factory reset can not be used.

Based on this requirement, commonly-used automotive MCUs have integrated hardware to support changing the debug access mode to the encryption mode. The G and L series of MSPM0 has the BOOTCFG0 register to support that.

Table 1-1. BOOTCFG0 Field Descriptions for L and G Series

Bit	Field	Type	Reset	Description
31-16	SWDP_MODE	R/W	AABBh	The serial wire debug port (SW-DP) access policy. This policy sets whether any communication is allowed with the device via the SWD pins (to any DAP). When disabled, no SWD communication is possible regardless of the configuration of the DEBUGACCESS field. 5566h = The SW-DP is fully disabled and no device access is possible via the SW-DP (0x5566 and all other values NOT 0xAABB). AABBh = The SW-DP is enabled and device access is set by the additional policies in NONMAIN.
15-0	DEBUGACCESS	W	AABBh	The debug access policy for accessing the AHB-AP, ET-AP, and PWR-AP debug access ports. Note that if SWDP_MODE is set to DISABLED, then the value of this field is ignored and the debug port remains fully locked. 5566h = Access to AHB-AP, ET-AP, and PWR-AP via SWD is disabled (0x5566 and all other values NOT 0xCCDD or 0xAABB). AABBh = Access to AHB-AP, ET-AP, and PWR-AP via SWD is enabled. CCDDh = Access to AHB-AP, ET-AP, and PWR-AP via SWD is only enabled when the correct password is provided via the DSSM before BCR execution.

Table 1-2. BOOTCFG0 Register Field Descriptions for C Series

Bit	Field	Type	Reset	Description
31-16	SWDP_MODE	R/W	AABBh	The serial wire debug port (SW-DP) access policy. This policy sets whether any communication is allowed with the device via the SWD pins (to any DAP). When disabled, no SWD communication is possible regardless of the configuration of the DEBUGACCESS field. AABBh = Enabled; FFFFh = Disabled (all other values).
15-0	DEBUGACCESS	W	AABBh	The debug access policy for accessing the AHB-AP, ET-AP, and PWR-AP debug access ports. Note that if SWDP_MODE is set to DISABLED, then the value of this field is ignored and the debug port remains fully locked. AABBh = Access to AHB-AP, ET-AP, and PWR-AP via SWD is enabled; FFFFh = Access to AHB-AP, ET-AP, and PWR-AP via SWD is disabled (all other values).

However, for some low-cost MCU like MSPM0 C series, this function is cut off because of cost. This application note explains how to implement encryption debugging through software to make this type of MCU without hardware support also meet the cybersecurity requirements.

Also, not only C but L and G series can also use this software method for a more flexible application.

2 MSPM0 Debug Register Introduction

MSPM0 uses the ARM® M0+ core, allowing the user to follow the procedure described by ARM to switch the device from JTAG to SWD. Like the common ARM core, MSPM0 mainly uses the debugger to transmit data between AP and DP based on the SWD protocol to access the MCU internal.

MSPM0 devices support debugging of processor execution, the device state, and the power state (through EnergyTrace™ technology). The DEBUGSS also provides a mailbox system for communicating with software through SWD.

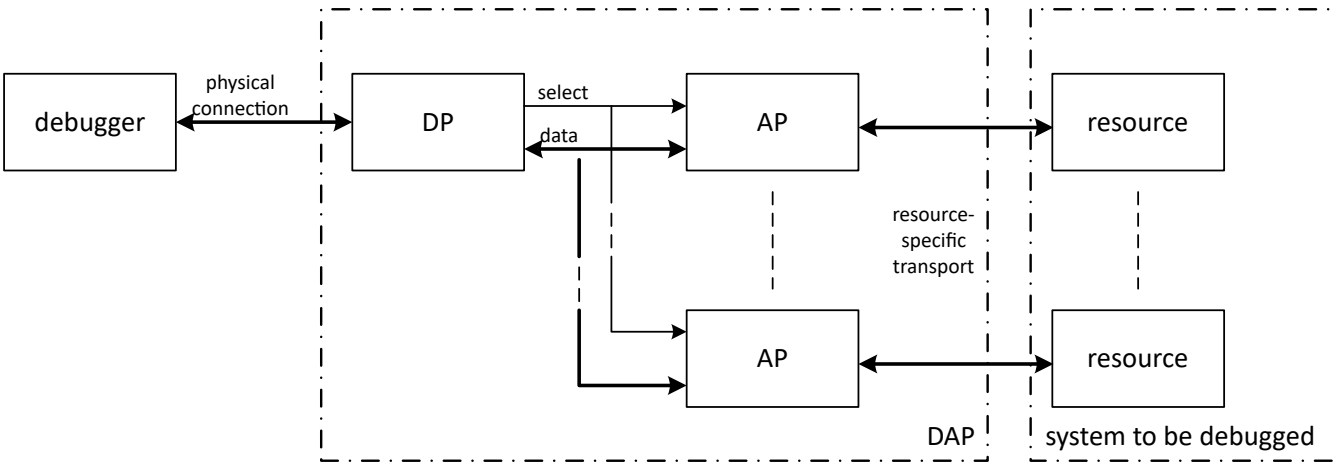

Figure 2-1. ARM Cortex devices Debug Block Diagram

Figure 2-1 shows MSPM0 Debug Sub System Block Diagram. MSPM0 devices support debugging of processor execution, the device state, and the power state (through EnergyTrace technology). The DEBUGSS also provides a mailbox system for communicating with software through SWD.

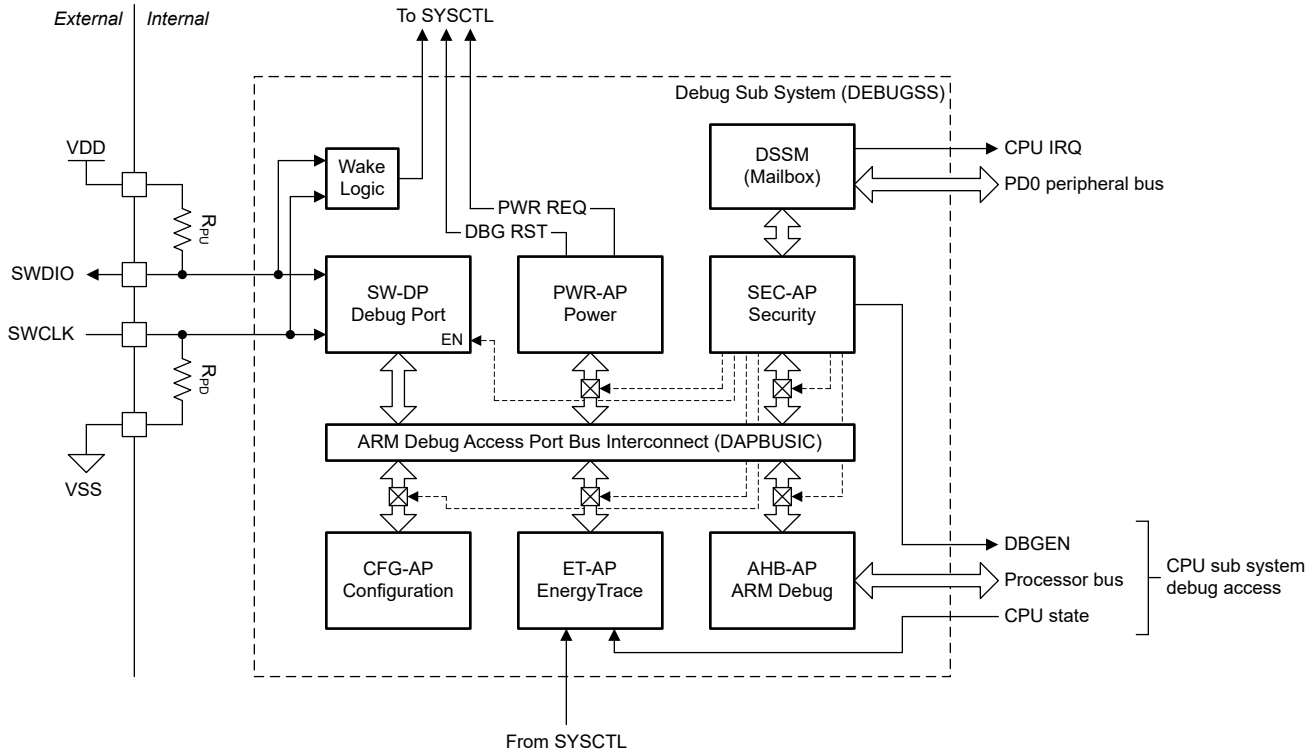

Figure 2-2. Debug Sub System Block Diagram

The SWD physical interface interacts with the Arm serial wire debug port (SW-DP) to gain access to the debug access port bus interconnect (DAPBUSIC) when the SW-DP is enabled.

There are several debug access ports in the DEBUGSS.

Table 2-1. DEBUGSS Access Port Listing

AP	Port Description	Purpose
AHB-AP	MCPUSS debug access port	Debug of the processor and peripherals
CFG-AP	Configuration access port	Access device type information, including the device part number and the device revision.
SEC-AP	Security access port	Access the debug subsystem mailbox(DSSM) for transmission of commands to the device during boot or communicating with software running on the device through SWD.
ET-AP	EnergyTrace™ technology access port	Read the power state data from EnergyTrace technology for power aware debug
PWR-AP	Power access port	Configure the device power states (interfaces with PMCU/SYSCTL), enabling low-power mode handling

The AHB-AP, PWR-AP, and ET-AP provide the complete device debug functionality (processor debug, peripheral and memory bus access, power state control, and processor state). And these can be disabled via BOOTCFG0 register in NONMAIN.

3 Implementation

This design is mainly based on the SWD protocol with XDS110 hardware. Both MCU and mailbox are needed.

3.1 Debugger with Mailbox

The debug subsystem mailbox (DSSM) enables a debug probe to pass messages to the target device through the SWD interface, and the target device can return data to the debug probe. In this design, the CCS script files are provided to enable the debugger to complete the transmission of encrypted identification information where for MCU SEC-AP is needed to communicate with application software through mailbox.

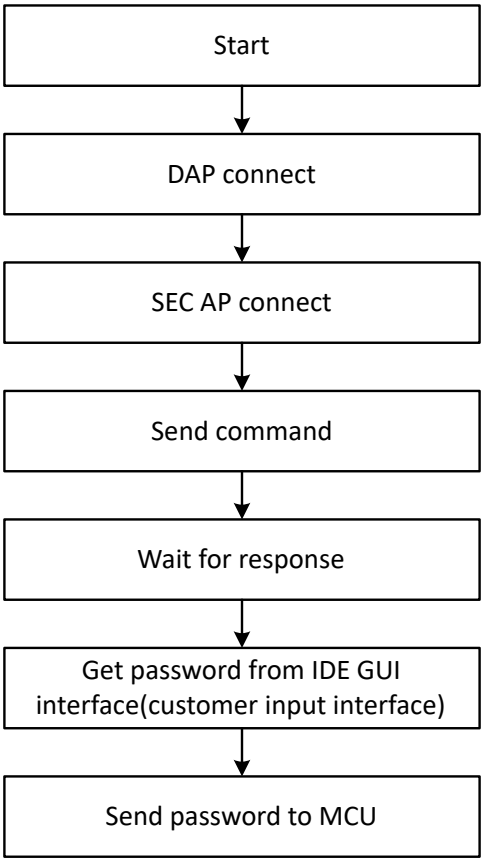

Figure 3-1. Debugger Mailbox Flow Chart

As shown in Figure 2-1, to access the internal resource, SW-DP port and SEC-AP port of the MCU need to be connected based on the SWD protocol firstly.

Then, the debugger sends a password verification command. This command is specified by the protocol. When the MCU is in the password verification stage and the command is consistent with the internal boot setting of MCU, a response signal is sent from MCU. Once receiving the response, the debugger starts to send the password input from external, such as IDE GUI interface to MCU, trying to passing the authentication.

3.2 MCU

3.2.1 Usage and Configuration of Nonmain

MSPM0 utilizes the SEC-AP to communicating with application software by DSSM. The prerequisite for this to work is that the Arm serial wire debug port needs to be retained to make sure that data can be received. Then, the data can be dealt in DSSM. At the same time, to prevent hackers from accessing the MCU app code, the AHB-AP port is disabled in Nonmain.

For factory reset, to prevent the MCU code from being easily cleared, choose to disable in Nonmain.

The most important part of the configuration process is the password setting. Since there is extra space in the Nonmain area of the MSPM0C that is not allocated to the corresponding register, the software password can be stored here. Doing so also allows customers to avoid having to repeatedly burn the password into the MCU when updating the code later.

According to the above, MSPM0C series Nonmain configuration is as shown in Table 3-1.

Table 3-1. Nonmain Register Configuration (MSPM0C Series)

Offset	Acronym	Setting Value	Purpose
41C00000h	BCRCONFIGID	0x00000003	Configuration ID of the BOOTCFG
41C00004h	BOOTCFG0	0xAABBFFFF	The SW-DP is enabled but AHB-AP, ET-AP, PWR-AP are disabled
41C00008h	BOOTCFG3	0xFFFFFFFF	Disable factory reset command;Disable static write protection configuration for Non-Main
41C0000Ch	SWPMAINLOW	0xFFFFFFFF	Disable lower part Flash protection
41C00010h	SWPMAINHIGH	0xFFFFFFFF	Disable higher part Flash protection
41C00014h	Password0	Customization	No hardware register. Use Nonmain remain free area to store the password
41C00018h	Password1		
41C0001Ch	Password2		
41C00020h	Password3		

3.2.2 MSPM0 Software Implementation

The code that implements this function is in the same project file as the user's own application.

According to the Nonmain configuration, the MCU can communicate with the mailbox based on SWD, but cannot directly access the kernel resource using the debug port. Writing software for the MCU can implement password verification based on Mailbox communication.

Table 3-2. DSSM Register Functions

DSSM Register	Description	Debug Probe	Target Device	Actions
TX_DATA	Data buffer	RW	R	TXCTL.TRANSMIT is set on write by the debug probe, and cleared on a read by the target device; TXIFG is also set on a write by the debug probe
TXCTL	Flow control and status	RW	R	None
RX_DATA	Data buffer	R	RW	RXCTL.RECEIVE is set on write by the target device, and cleared on a read by the debug probe; RXIFG is also set on a write by the target device
RXCTL	Flow control and status	R	RW	None

The workflow for MCU is as shown in Figure 3-2.

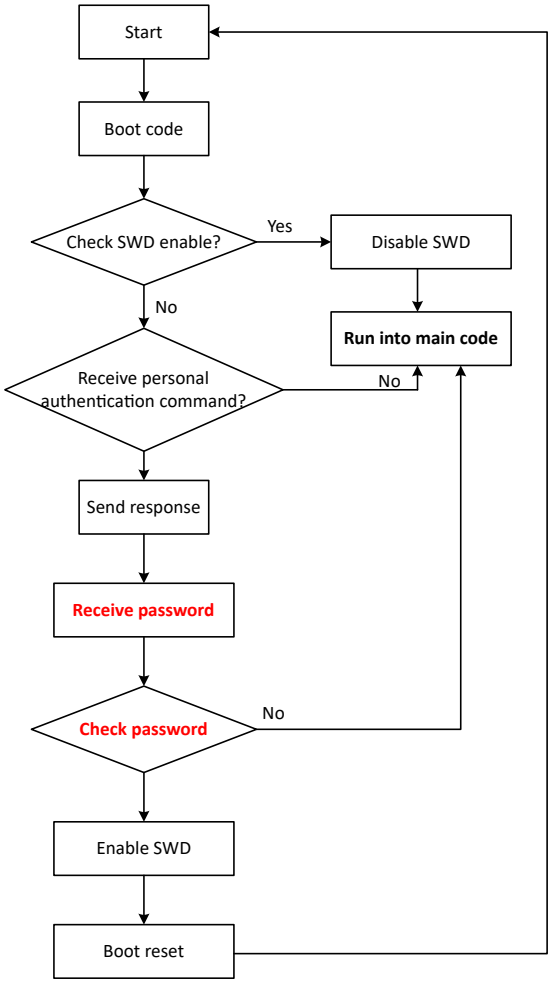

Figure 3-2. MCU Software Flow Chart

Combine with debugger software as shown in Figure 3-3. The following is the process of implementing encryption debugging. After the MCU runs the Boot code, the MCU first checks whether AHB-AP is turned on in the Nonmain configuration. If not, then this means that this is in encryption state at this time, and further check whether there is a password transmission command in the DSSM register. If so, then send a response and use the DSSM register to receive the password.

After receiving the four 32-bit password, the MCU checks whether this is the correct password. If correct, then the Nonmain configuration is changed again to enable AHB-AP port.

For Nonmain to take effect, trigger the MCU to perform a boot reset through software. Then, the MCU is powered back on and detects that the AHB-AP is now turned on. To make sure that the MCU is locked again after this debugging is completed, the Nonmain configuration is changed again after detecting that the AHB-AP port is open. However, after this configuration is completed, the MCU is not reset, so the modified Nonmain does not take effect immediately until debug is finished.

After AHB-AP port is opened with password, MCU runs into customer self code and can succeed to debug, and disable AHB-AP port again.

For usual MCU startup, there is no need to debug; MCU enters into main application code after checking SWD is disabled.

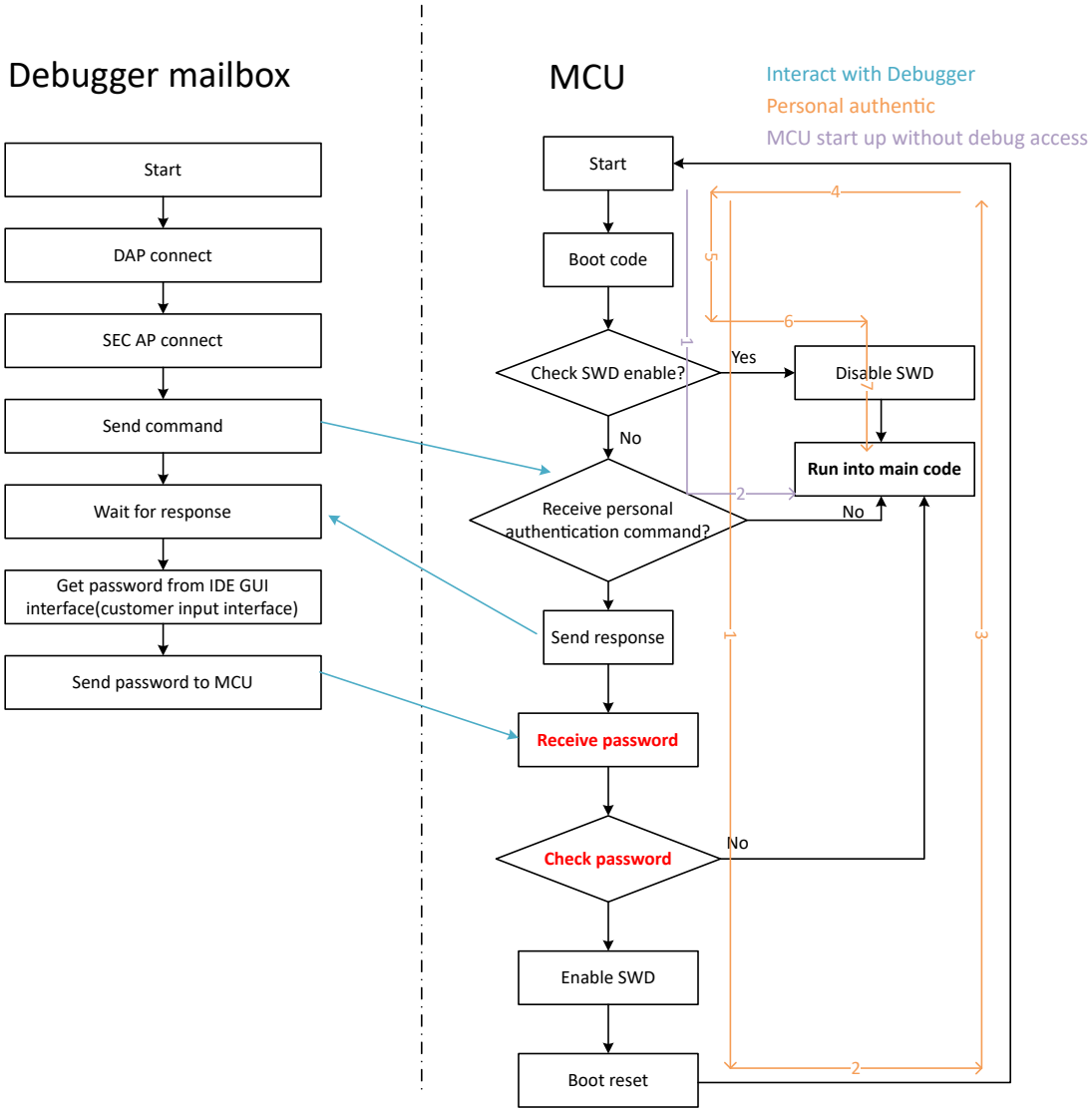

Figure 3-3. MCU and Debugger Flow Chart

During the entire implementation process, Nonmain is modified many times, which also increases the MCU security risk. Specifically, the NONMAIN is a dedicated region of flash memory which stores the configuration data used by the BCR and is related to SWD policies, flash memory, and so on. When modifying Nonmain, if abnormal operation such as power failure causes the Nonmain configuration to be incorrect, then this causes the MCU to become a brick and never be connected. To avoid this problem, a monitor is added to the code to detect whether the configuration is correct after powering on and modifying Nonmain.

Note, that the code for implementing encryption debugging in the MCU is placed in the startup file, and is executed immediately after the MCU is powered on. This jumps to the C Initialization Routine.

4 Execution

This section is a step-by-step instruction on how to implement encryption debugging functions.

Materials used:

- CCS IDE
- One code project
- Two customized CCS scripts
- XDS110 hardware

The customer's own application is in main.c. The specific application connects to the MCU functional software mentioned in Section 3.2.2, and are in the startup file. Also, note that the .cmd file is also changed.

The CCS script is used to set the debugger to send the related mailbox message to support the whole connection work flow and the password sending.

Note

TI has provided the code project and CCS scripts which have the demo code to achieve password-protected debug function. If users need further assistance, then contact a TI representative.

4.1 First Time Flashing

To flash for the first time, users need to configurate NONMAIN to store the password. The settings needed are shown in Figure 4-1.

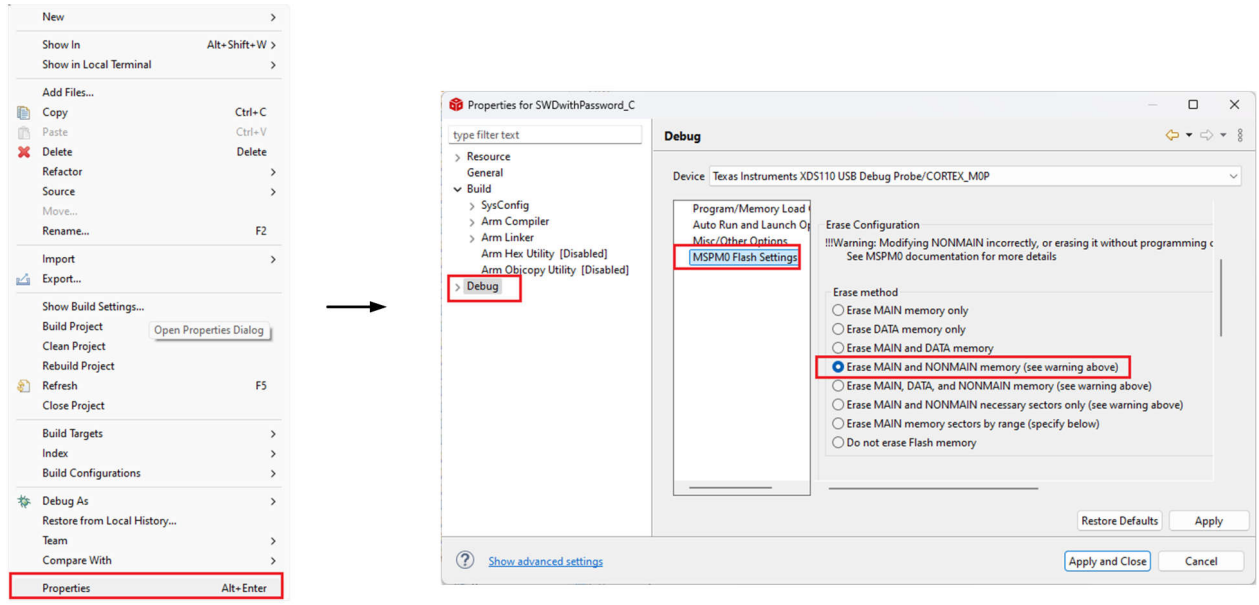

Figure 4-1. Enable Nonmain Flashing

4.2 Access to Locked MCU

If the MCU has been password locked, then the following steps show how to reconnect the MCU and debug a new project.

Since the Nonmain has been configured during the first programming, there is no need to configure Nonmain for subsequent programming. For projects that are burned or debugged later, some changes need to be made. First, disable Nonmain flashing in the project properties, then comment out the Nonmain configuration code in boot_configwithPassword.c, as shown in Figure 4-2 and Figure 4-3.

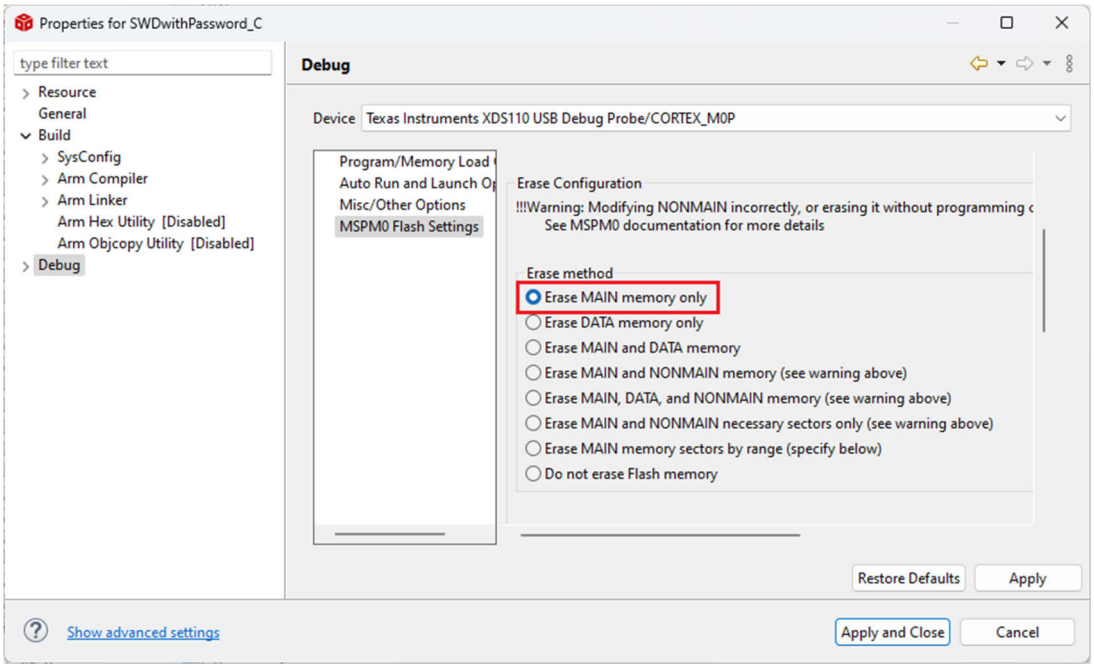

Figure 4-2. Disable Nonmain Flashing

```
/* Bootcode configuration
 * The second time need to be commented */

//PLACE_IN_MEMORY(".BCRConfig")
//const BCR_Config BCRConfig_origin =
//{
//    .bcrConfigID          = 0x3,
//    .debugAccess          = BCR_CFG_DEBUG_ACCESS_DIS,
//    .swdpMode             = BCR_CFG_SWDP_EN,
//    .factoryResetMode     = BCR_CFG_FACTORY_RESET_DIS,
//    .staticWriteProtectionNonMain  = BCR_CFG_NON_MAIN_STATIC_PROT_DIS,
//    .staticWriteProtectionMainLow  = CFG_DEFAULT_VALUE,
//    .staticWriteProtectionMainHigh = CFG_DEFAULT_VALUE,
//    .reserved = 0xFFFFFFFFU,
//    .password0 = DebugAccess_Password0,
//    .password1 = DebugAccess_Password1,
//    .password2 = DebugAccess_Password2,
//    .password3 = DebugAccess_Password3,
//};
```

Figure 4-3. Comment Nonmain Configuration Code

After modification is completed, the following steps need to be performed to unlock and debug the MCU.

1. Connect MCU and open target configuration in CCS to load CCS scripts.

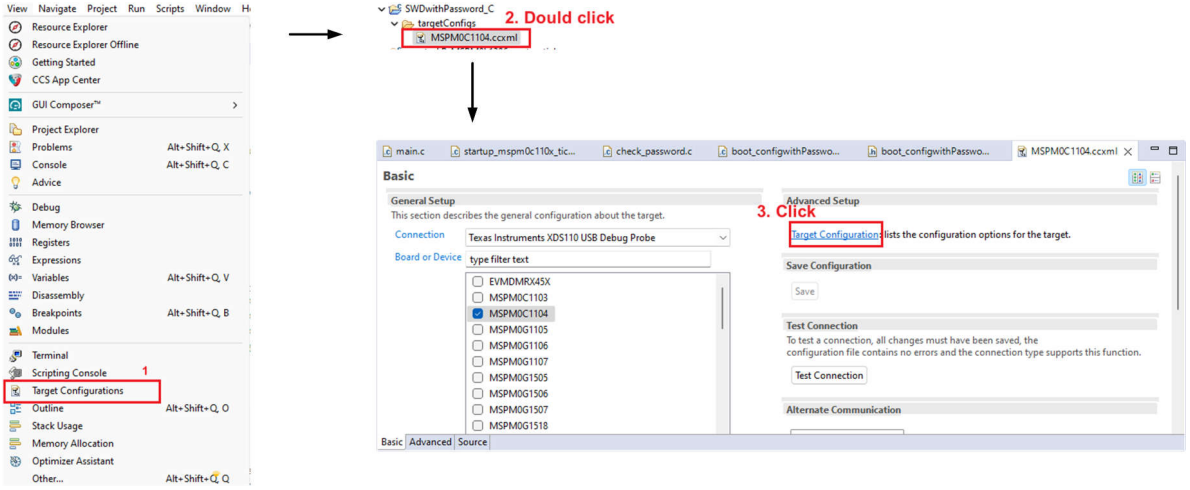

Figure 4-4. Open Target Configuration

2. Add two CCS scripts into project.

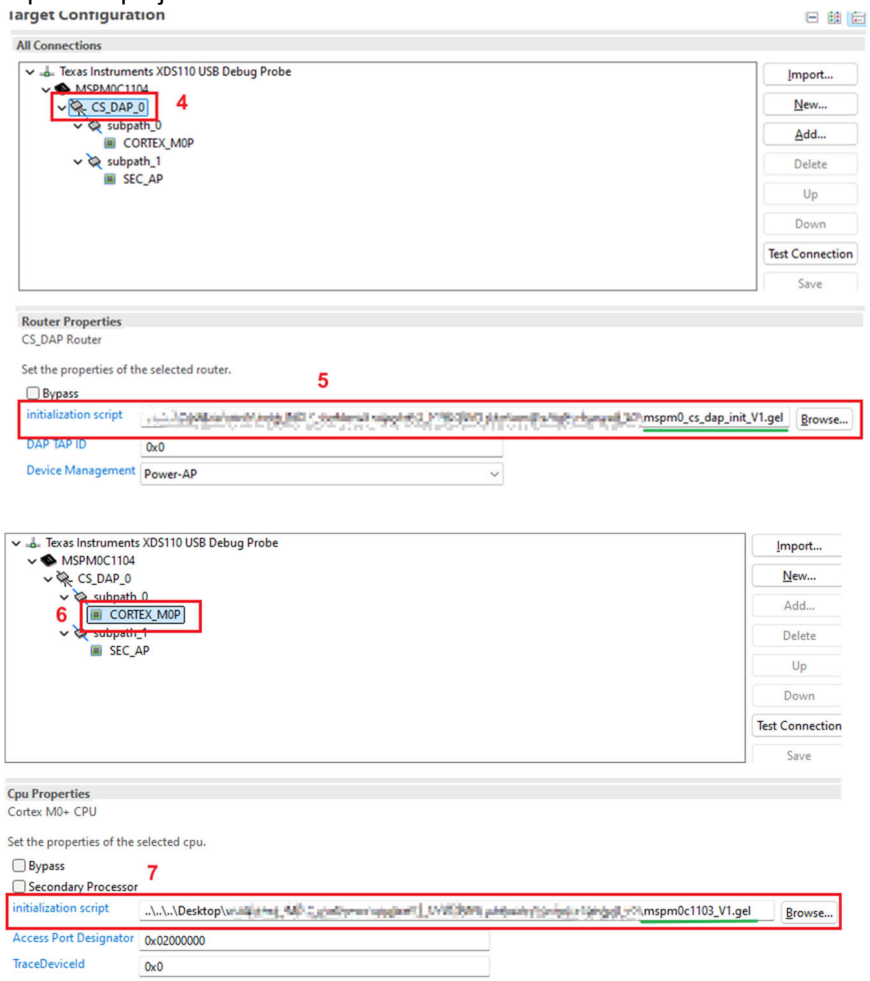

Figure 4-5. Add Scripts

3. Enter in the password by CCS GUI interface and Ctrl + S to save the configuration.

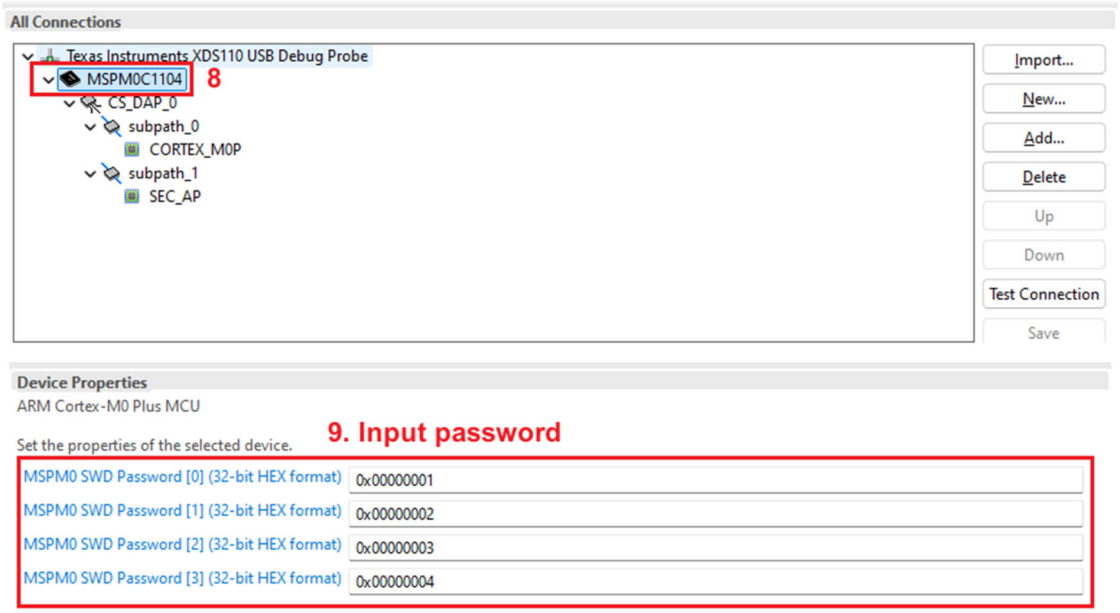

Figure 4-6. Input Password by GUI

4. Launch the configuration.

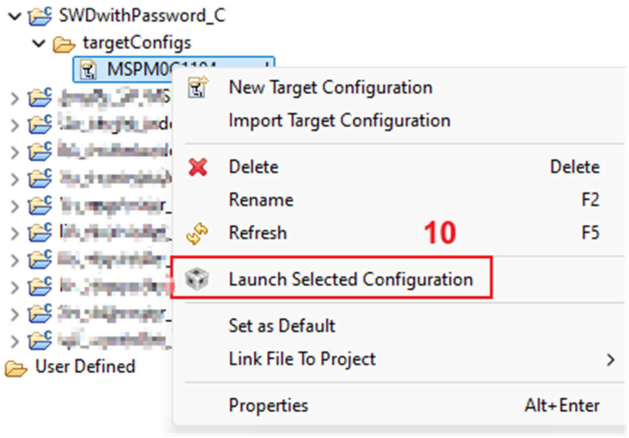

Figure 4-7. Launch the Configuration

5. Use the scripts. Keep pulling down the reset pin, set MCU in reset mode and do not run the code. Then click the MSPM0_PasswordAuthForMSPM0C. Keep pressing the reset button.

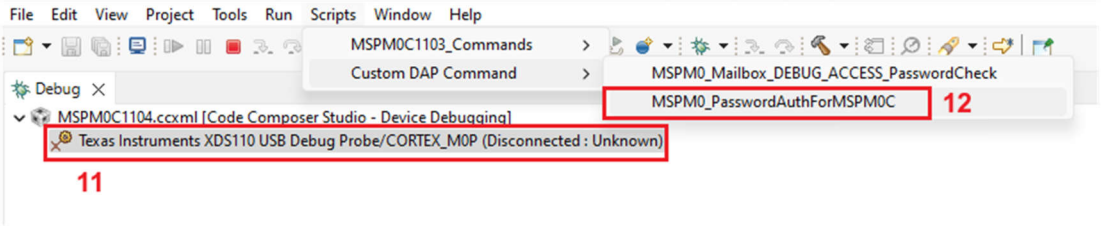

Figure 4-8. Run the Script

6. After this interface appears, which shows the DAP and SEC-AP has been connected and wait for the response, pull up the reset pin.

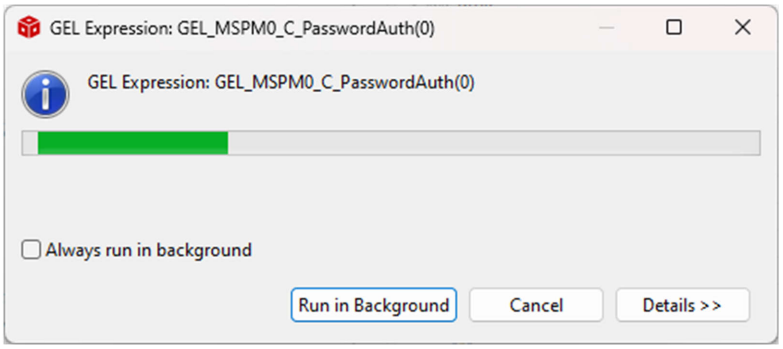

Figure 4-9. Message Showing The Process of Mailbox

7. When reset is pulled high, the debugger tries to connect to the DAP and start authentication of the MCU. If the verification passes, then the following information as shown in Figure 4-10 is displayed. In addition, the password sent by the debugger is also displayed in the console.

```
CS_DAP_0: GEL Output: Attempting SEC_AP connection
CS_DAP_0: GEL Output: Command = 0x000000F0
CS_DAP_0: GEL Output: Respond = 0x0000CAF0
CS_DAP_0: GEL Output: Data 1 Sent = 0x00000001
CS_DAP_0: GEL Output: Data 2 Sent = 0x00000002
CS_DAP_0: GEL Output: Data 3 Sent = 0x00000003
CS_DAP_0: GEL Output: Data 4 Sent = 0x00000004
CORTEX_M0P: GEL Output: FINISHED
```

Figure 4-10. Console Information

8. MCU is unlocked and access the memory or flash code.

Note

Note, that once debugging is finished, execute boot reset to make the Nonmain configuration effective. The MCU is locked again. For easy reset, pull down reset pin for 1s, then power off and on again. Figure 4-11 shows an error message for users who do not have enough password attempts to access the MCU.

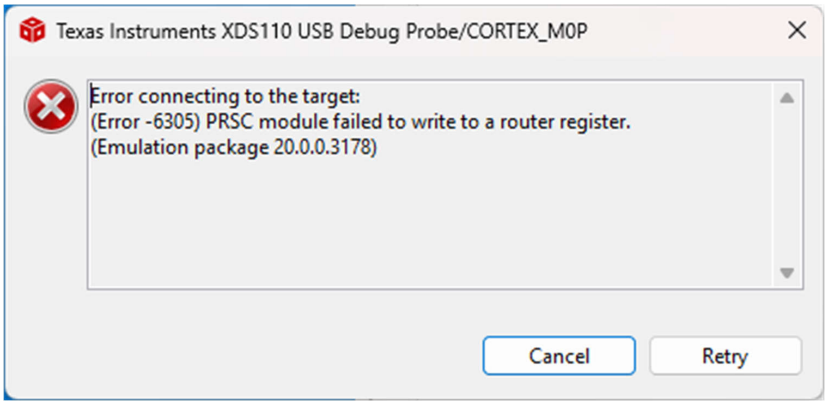

Figure 4-11. Fail to Connect MCU

5 How to Customize Passwords

If users want to change the mailbox communication protocol, then the next section details how the password can be changed.

5.1 Password

1. Demo code - boot_configwithPassword.h

```
#define DebugAccess_Password0                                   (0x00000001)
#define DebugAccess_Password1                                   (0x00000002)
#define DebugAccess_Password2                                   (0x00000003)
#define DebugAccess_Password3                                   (0x00000004)
```

5.2 Password Length

1. Demo code – boot_configwithPassword.h; password

```
#define PASSWORD_WORD_LEN                                           (4U)
#define DebugAccess_Password0                                   (0x00000001)
#define DebugAccess_Password1                                   (0x00000002)
#define DebugAccess_Password2                                   (0x00000003)
#define DebugAccess_Password3                                   (0x00000004)
```

2. Demo code – boot_configwithPassword.h: Password in BCR_Config struct

```
/* Bootcode user configuration structure */
typedef struct
{
    /*! Configuration signature */
    uint32_t bcrConfigID;
    /*! Enable/disable AHB-AP, ET-AP, PWR-AP.
     * One of @ref BCR_CFG_DEBUG_ACCESS */
    BCR_CFG_DEBUG_ACCESS debugAccess;
    /*! Enable/disable SWD port access. One of @ref BCR_CFG_SWDP_MODE */
    BCR_CFG_SWDP_MODE swdpMode;
    /*! The factory reset mode. One of @ref BCR_CFG_FACTORY_RESET */
    BCR_CFG_FACTORY_RESET factoryResetMode;
    /*! Non Main Flash Static Write Protection.
    * One of @ref BCR_CFG_NON_MAIN_STATIC_PROT */
    BCR_CFG_NON_MAIN_STATIC_PROT staticWriteProtectionNonMain;
    /*! Programs static write protection of first 32K bytes.
     * One bit corresponds to one sector, LSB is Sector 0. Setting a bit
     * to 0 disables write, setting a bit to 1 enables write Possible values:
     *    - 0x0 to 0xFFFFFFFF */
    uint32_t staticWriteProtectionMainLow;
    /*! Programs static write protection of first 32K bytes.
     * One bit corresponds to eight sectors. Setting a bit
     * to 0 disables write, setting a bit to 1 enables write Possible values:
     *    - 0x0 to 0xFFFFFFF0 */
    uint32_t staticWriteProtectionMainHigh;
    /*! Reserved */
    uint32_t reserved;
    uint32_t password0;
    uint32_t password1;
    uint32_t password2;
    uint32_t password3;
    //Can add password length if needed
} BCR_Config;
```

3. Demo code – boot_configwithPassword.c: BCRConfig_origin variable to configuration Nonmain

```
PLACE_IN_MEMORY(".BCRConfig")
const BCR_Config BCRConfig_origin =
{
    .bcrConfigID          = 0x3,
    .debugAccess          = BCR_CFG_DEBUG_ACCESS_DIS,
    .swdpMode             = BCR_CFG_SWDP_EN,
    .factoryResetMode     = BCR_CFG_FACTORY_RESET_DIS,
    .staticWriteProtectionNonMain  = BCR_CFG_NON_MAIN_STATIC_PROT_DIS,
    .staticWriteProtectionMainLow  = CFG_DEFAULT_VALUE,
    .staticWriteProtectionMainHigh = CFG_DEFAULT_VALUE,
    .reserved = 0xFFFFFFFU,
    .password0 = DebugAccess_Password0,
    .password1 = DebugAccess_Password1,
    .password2 = DebugAccess_Password2,
    .password3 = DebugAccess_Password3,
};
```

4. Demo code – check_password.c: Para_init function

```
void Para_init(void)
{
    AHPAccess = false;
    BCRConfig_update.bcrConfigID          = 0x3;
    BCRConfig_update.debugAccess          = BCR_CFG_DEBUG_ACCESS_DIS;
    BCRConfig_update.swdpMode             = BCR_CFG_SWDP_EN;
    BCRConfig_update.factoryResetMode     = BCR_CFG_FACTORY_RESET_DIS;
    BCRConfig_update.staticWriteProtectionNonMain  = BCR_CFG_NON_MAIN_STATIC_PROT_DIS;
    BCRConfig_update.staticWriteProtectionMainLow  = CFG_DEFAULT_VALUE;
    BCRConfig_update.staticWriteProtectionMainHigh = CFG_DEFAULT_VALUE;
    BCRConfig_update.reserved = 0xFFFFFFFU;
    BCRConfig_update.password0 = DebugAccess_Password0;
    BCRConfig_update.password1 = DebugAccess_Password1;
    BCRConfig_update.password2 = DebugAccess_Password2;
    BCRConfig_update.password3 = DebugAccess_Password3;
}
```

5. Demo code –check_password.c: Nonmain_check function
6. mspm0_cs_dap_init_V2: Password_LENGTH

```
#define PASSWORD_LENGTH                                         (4U)
```

7. mspm0_cs_dap_init_V2: GEL_MSPM0_C_PasswordAuth(autoReset) function. The password length has been changed and cannot use CCS GUI as human interface, so users need to add an actual password in CCS script and send the debugger to transmit.

6 Summary

This document provides a way to implement an MCU encryption debug, so that devices that do not have the hardware conditions can still meet cybersecurity functions through software. Users are provided with sample programs and scripts that can quickly implement the function with clear steps in the documentation. In addition, the documentation also provides descriptions to tell users how to customize the code.

7 References

- Texas Instruments, MSPM0 C-Series device, webpage
- Texas Instruments, MSPM0 L-Series device, webpage
- Texas Instruments, MSPM0 G-Series device, webpage
- Texas Instruments, Code Composer Studio™ , webpage

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 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