Application Note

Serial Flash Programming of F29H85x™

Skyler Baumer, Sen Wang, Aakash Kedia, Arush Pant

ABSTRACT

Embedded processors often need to be programmed in situations where JTAG cannot be used to program the target device. In these cases, the engineer needs to rely on peripheral programming designs. C2000™ devices aid in this endeavor through the inclusion of several program loading utilities in ROM. These utilities are useful, but only solve half of the programming problem because the utilities only allow loading application code into RAM. This application note builds on these ROM loaders by using a flash kernel. A flash kernel is loaded to RAM using a ROM loader and is then executed and used to program the on-chip Flash memory of the target device with the end application. A key consideration of using a flash kernel on the F29H85x is the addition of the Hardware Security Module (HSM). This document details one possible implementation for C2000 devices and provides PC utilities to evaluate the design.

Table of Contents

List of Figures

List of Tables

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.
Microsoft Visual Studio® and Windows® are registered trademarks of Microsoft Corporation.
Linux® is a registered trademark of Linus Torvalds.
All trademarks are the property of their respective owners.

1 Programming Fundamentals

Before programming a device, understanding how the non-volatile memory of C2000 devices works is necessary. Flash is a non-volatile memory that allows users to easily erase and re-program. Erase operations set all the bits in a sector to '1' while programming operations selectively set bits to '0'.

Flash operations on all C2000 devices are performed using the CPU. Algorithms are loaded into RAM and executed by the CPU to perform any flash operation. For example, erasing or programming the flash of a C2000 device with Code Composer Studio™ entails loading flash algorithms into RAM and letting the processor execute them. There are no special JTAG commands that are used. All flash operations are performed using the flash application programming interface (API). Because all flash operations are done using the CPU, there are many possibilities for device programming. Irrespective of how the kernels and application are brought into the device, flash is programmed using the CPU.

2 Introduction

ROM bootloader, also referred to as primary bootloader or simply loader, is a small piece of code that resides in the boot-ROM memory of the target device that allows the loading and execution of code from an external host. In most cases, a communication peripheral such as Universal Asynchronous Receiver/Transmitter (UART) or Controller Area Network (CAN) is used to load code into the device rather than JTAG, which requires an expensive specialized tool and is not advisable to use in a commercial setting.

Boot Pins are used to configure different boot modes using various peripherals that determine which ROM loader is invoked. In this application note, the peripheral used is UART. The boot modes that are associated with the boot pins refer to the first instance of the peripheral. For UART, the boot mode is associated with UART**A**.

C2000 devices partially solve the problem of firmware updates by including some basic loading utilities in ROM. Depending on the device and the communications peripherals present, code can be loaded into on-chip RAM using UART, Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and CAN. A subset of these loaders is present in every C2000 device, but the loaders can only load code into RAM. How does one bridge the gap and program the application code into non-volatile memory?

At a high-level, application programming to non-volatile memory like flash requires two steps:

1. Use the ROM bootloader to download a secondary bootloader to RAM.
2. Run the secondary bootloader in RAM to download the application to flash.

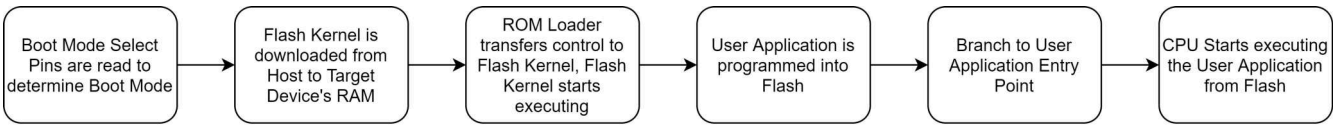

Figure 2-1. Flash Kernel Flow

Although the overarching concept are synonymous, note that there are a few major differences between the flash kernels implemented for C28-based devices and the kernel described in this document.

2.1 Hardware Security Module

The primary difference between the flash kernel design for C28-based devices and the F29H85x flash kernel is the integration of the Hardware Security Module (HSM). The HSM is a subsystem that provides security and cryptographic functions. The C29 CPUs interface with the HSM to perform cryptographic operations required for code authentication, secure boot, secure firmware upgrades, and encrypted run-time communications.

During the UART boot sequence, the HSM is responsible for authentication of the incoming image. For the authentication to succeed, the incoming image must include an X.509 certificate. To properly generate an X.509 certificate with the flash kernel, refer to Section 2.2.

The HSM introduces the concept of different device *security states*. The device states are High Security - Field Securable (HS-FS), High Security - Key Provisioned (HS-KP), and High Security - Security Enabled (HS-SE). By default, the F29H85x device ships with HS-FS. Table 2-1 describes the differences between these three states.

Table 2-1. Device Security State

	HS-FS	HS-KP	HS-SE
C29 boot image (flash kernel)	Secure boot not enforced	Secure boot enforced with customer keys programmed by keywriter	Secure boot enforced with customer keys programmed by keywriter
HSM boot image	Secure boot enforced (with default TI-provided key)	Secure boot enforced with customer keys programmed by keywriter	Secure boot enforced with customer keys programmed by keywriter
C29 JTAG	Open by default	Open by default	Closed by default
SoC firewalls	Open by default	Disabled for HSM and enabled for C29	Disabled for HSM and enabled fro C29
C29 CPU access to C29 flash banks	Enabled	Disabled	Enabled

TI provides OTP (One Time Programmable) Keywriter that can be used to transition an HS-FS device to HS-KP or HS-SE. OTP Keywriter is a combination of TI delivered HSM Run Time Firmware and Tools (Certificate generation) which together when executed on device enables the following:

- Provisioning up to two sets of Customer Keys (both public keys for root of trust of boot images and encryption keys for decryption of boot images).
- Programming of Extended OTP fields which are additional OTP fields available for customer specific usage.
- Programming of OTP fields KEY_COUNT, KEY_REVISION which enables device transition from HS-FS (Field Securable) to HS-KP (Key Provisioned).
- Programming of OTP fields like SWREV for SBL, HSM, APP, SECCFG images which enforce Anti Roll Back checks by Secure Boot on HS-KP as well as HS-SE device.

Once these fields are programmed, the device state is transitioned to HS-KP, boot ROM enforces secure boot with Image Authentication and decryption based on the keys provisioned and configured in the device. Secure boot requires an image to be encrypted (optional) and signed using customer keys. This image is then verified by the SoC using the active MPK Hash (to verify the signature) and the MEK (for decryption).

CAUTION

- This action of burning the keys is irreversible across the fields, so caution needs to be exercised to provide the key values in correct format and correct key configurations.
- The action of programming fields is irreversible and providing incorrect values or configuration can permanently damage the device.

For more information on the OTP Keywriter, request the Restricted Security Package from the F29H85x MCU SDK download page.

The firmware upgrade process differs between HS-FS, HS-KP, and HS-SE devices.

- For details regarding C29 CPU1 firmware upgrade on an HS-FS device, refer to CPU1 Firmware Upgrade (HS-FS).
- For details regarding C29 CPU1 firmware upgrade on an HS-KP/HS-SE device, refer to CPU1 Firmware Upgrade (HS-KP or HS-SE).
- For details regarding HSM firmware upgrade on an HS-KP/HS-SE device, refer to HSM Firmware Upgrade (HS-KP or HS-SE).

2.2 ROM Bootloader

The basic idea of a flash kernel and how the firmware upgrades has been described, the next section details the first step of the process: loading the kernel to RAM via bootROM.

At the beginning, the device boots and, based on the boot mode, decides if the device executes the code already programmed into the Flash memory or load in code using one of the ROM loaders. This application note focuses on the boot execution path when the emulator is not connected.

Note

This section is based on the F29H85x device. Specific information for a particular device can be found in the *ROM Code and Peripheral Booting* section of the device-specific technical reference manual (TRM).

Table 2-2. Default Boot Modes for F29H85x devices

Boot Mode	GPIO72 (default boot mode select pin 1)	GPIO84 (default boot mode select pin 0)
Parallel I/O	0	0
UART	0	1
CAN	1	0
Flash	1	1

After the boot ROM readies the device for use, the device decides where to start executing. In the case of a standalone boot, the device does this by examining the state of two GPIOs (as seen in Table 2-2 , the default choices are GPIO 72 and 84). In some cases, two values programmed into one time programmable (OTP) memory can be examined. In the implementation described in this application note, the UART loader is used, so at powerup GPIO 84 must be forced high and GPIO 72 must be forced low. If this is the case when the device boots, then the UART loader in ROM begins executing and operates at a baud rate of 115200. At this point, the device is ready to receive code from the host.

A major difference between the previous C28-based devices and the F29H85x is the inclusion of the Hardware Security Module (HSM). All boot flows require the HSM to authenticate the incoming image before the boot flows can be executed.

Please see the *ROM code and Peripheral Booting* section of the Technical Reference Manual (TRM) for details on the boot flow. And *Device Boot Flow Diagrams* section, on how the HSM and C29 CPUs communicate during the boot sequence.

2.3 Combined Image with X.509 Certificate

With the inclusion of the HSM, the BootROM expects all incoming images to be in a binary format and be combined with a X.509 certificate. The first 0x1000 bytes of the binary file must contain the key certificate.

Listed below are the default post-build steps provided by UART flash kernel and other SDK examples to generate the combined binary image with X.509 certificate.

For new Code Composer Studio projects, paste the following script in the *Post-build Steps* section under Build category of the project properties. To run the script standalone, find import.mak in the root directory of the device SDK for the default alias of the variable.

RAM post-build steps (for Flash Kernel):

```
${CG_TOOL_OBJCOPY} --strip-all -O binary ${ProjName}.out ${ProjName}.bin
$(PYTHON)  ${COM_TI_MCU_SDK_INSTALL_DIR}/tools/boot/signing/mcu_rom_image_gen.py --image-bin $
{ProjName}.bin --core C29 --swrv 1 --loadaddr 0x200E1000 --sign-key ${COM_TI_MCU_SDK_INSTALL_DIR}/
tools/boot/signing/mcu_custMpk.pem --out-image ${ProjName}.cert.bin --boot RAM --device f29h85x --
debug DBG_SOC_DEFAULT
```

- Produces RAM-based binary image: ${ProjName}.cert.bin
- mcu_rom_image_gen.py ram parameter: --loadaddr 0x200E100 (LDAx RAM), --boot RAM

Flash post-build steps (for Flash application images):

```
${CG_TOOL_OBJCOPY} --remove-section=cert -O binary ${ProjName}.out ${ProjName}.bin
$(PYTHON) ${COM_TI_MCU_SDK_INSTALL_DIR}/tools/boot/signing/mcu_rom_image_gen.py --image-bin $
{ProjName}.bin --core C29 --swrv 1 --loadaddr 0x10001000 --sign-key ${COM_TI_MCU_SDK_INSTALL_DIR}/
tools/boot/signing/mcu_gpkey.pem --out-image ${ProjName}_cert.bin --device f29h85x --boot FLASH --
img_integ no
${CG_TOOL_OBJCOPY} --update-section cert=C29-cert-pad.bin ${ProjName}.out ${ProjName}_cert.out
$(DELETE) ${ProjName}.out C29-cert-pad.bin;
$(RENAME) ${ProjName}_cert.out ${ProjName}.out
```

- Produces Flash-based binary image: ${ProjName}_cert.bin
- mcu_rom_image_gen.py flash parameter: --loadaddr 0x10001000 (Flash entry address), --boot FLASH

Both post build steps shown above generate a certificate for the application, converts the application .out file into binary, and creates a combined binary image with an X.509 certificate.

Additionally, the post-build steps leverages two different keys provided in the SDK:

- mcu_gpkey.pem: General purpose key provided by TI to generate TI key certificate.
- mcu_custMpk.pem: A dummy key that mimics a custom key, can be used as an exemplary custom key to test out Key Provision & Code Provisioning.

Note

To change the generated key certificate with a custom key certificate, provide an alternate key and pass into the python script via parameter *--sign-key*. This is necessary for all flash images concerning HS-KP and HS-SE, and HSMRt for HS-SE.

3 Flash Kernel Implementation

When compared to the C28-based designs, the UART flash kernel for F29H85x is quite different due to the integration of the HSM. A high-level overview of the features implemented in this example is provided below.

- CPU1 DFU (HS-FS only)
  1. Device need to be configured for Bank Mode 0.
  2. Flash kernel in CPU1 programs application to the flash banks.
  3. Certificate is programmed in 0x10000000 - 0x10000FFF, certificate is not used for authentication but rather to infer the size of the kernel binary.
  4. Application can be programmed in the remaining flash addresses, and TI recommends to program codestart at 0x10001000. As 0x10001000 is the flash entry address in flash boot mode.

> **CAUTION**
> For users not concerned with the security features provided by the HSM, simply use the CPU1 DFU flow to upgrade the CPU1 firmware.

- Load HSMRt (prerequisite for Key & Code Provisioning)
  1. Flash kernel in CPU1 receives key provisioning HSM run-time (HSMRt) and places in LDAx RAM.
  2. HSM authenticates HSMRt image and begin executing the run time in LDAx RAM.
- CPU1 Key Provisioning (HS-FS -> HS-KP)
  1. Flash kernel in CPU1 loads HSMRt
  2. Flash kernel receives key certificate and places in LDAx RAM
  3. HSM validates key certificate and programs them to OTP
- CPU1 Code Provisioning (HS-SE only)
  1. Flash kernel in CPU1 loads in HSMRt
  2. Flash Kernel receives the CPU1 application image certificate and shares the same with HSMRt
  3. After successful authentication of the image, HSMRt responds with an acknowledgment, after which flash kernel starts importing the chunk of data via UART into the LDAx memory.
  4. After each 16KB (size of LDAx memory) of data received, the flash kernel sends an HSM requests to program the data for further processing
  5. After all chunks are received and programmed, HSMRt is requested to verify the code programmed in HSM active and dormant banks. When the HSMRt firmware authenticates the programmed image against the certificate, the certificate is further programmed to make sure of a successful boot in the subsequent power cycles.
  6. Upon successful authentication, the HSM programs the firmware to CPU1 flash
- HSM Code Provisioning (HS-KP/HS-SE -> HS-SE)
  1. Flash kernel in CPU1 loads in HSMRt
  2. Flash kernel receives the HSM application image certificate and shares the same with HSMRt
  3. After successful authentication of the image, HSMRt responds with an acknowledgment, after which flash kernel starts importing the chunk of data via UART into the LDAx memory.
  4. After each 16KB (size of LDAx memory) of data received, the flash kernel sends an HSM requests to program the data for further processing
  5. After all chunks are received and programmed, HSMRt is requested to verify the code programmed in HSM active and dormant banks. When the HSMRt firmware authenticates the programmed image against the certificate, the certificate is further programmed to make sure of a successful boot in the subsequent power cycles.
- SECCFG Code Provisioning (HS-KP -> HS-SE)
  1. Flash kernel in CPU1 loads in HSMRt
  2. Flash kernel receives the image certificate and shares the same with HSMR
  3. After successful authentication of the image, HSMRt responds with an acknowledgment, after which flash kernel starts importing the SecCfg data via UART into the LDAx memory
  4. After all the SecCfg data are received and programmed, the HSMRt is requested to verify the SecCfg programmed in the dormant banks with valid counter values. When the HSMRt authenticates the programmed image against the certificate, the certificate is further programmed to make sure of a successful boot in the subsequent power cycles.

5. Note in the case of HS-SE device, the decision of programming of the certificate is made on the swap value of the SSU registers.

> **CAUTION**
>
> Key and code provision both require a RAM-based HSMRt image to be running on the HSM. Nonetheless, the image requires a different key certificate at each stage of provisioning.
>
> In key provisioning, user must provide the HSMRt image with default TI key certificate to complete authentication on an HS-FS device, whereas code provisioning requires an image with user key certificate to complete authentication on an HS-KP/HS-SE device.
>
> The OTP Keywriter firmware described in the Hardware Security Module section contains a binary image that can perform these tasks.

The UART flash kernel is designed for CPU1 while in Bank Mode 0. This means that all flash banks are allocated to CPU1 and the bank swapping feature is not available. For more details on Bank Modes, please see the *Bank Modes and Swapping* section of the TRM. The flash kernel communicates with the host PC application provided in the F29H85x MCU SDK (MCU_SDK_F28H85x > *utilities* > *flash_programmers* > *serial_flash_programmer*) and provides feedback to the host on the receiving of packets and completion of commands given.

After loading the kernel into RAM and executing by the UART ROM bootloader, the kernel first initializes the PLLs, GPIOs, the UARTA module, and the flash module. The UART communication is configured with a baud rate of 115200, similar to the UART boot flow. After this, the kernel begins a while loop, which waits on commands from the host, executes the commands, and sends a status packet back to the host. This while loop breaks when a Run or Reset command is sent.

Commands are sent in a packet described in Table 4-2 and each packet is either acknowledged or not-acknowledged. All commands, except for Run and Reset, send a packet after completion with the status of the operation. The status packet sends a 16-bit status code and 32-bit address. In case of an error, the address in the data specifies the address of the first error. In case of NO_COMMAND_ERROR, the address is 0x1000.

3.1 CPU1 Firmware Upgrade (HS-FS)

To perform a firmware upgrade for CPU1 of an HS-FS device, compile the CPU1_APP build configuration of the project. This can be done by right-clicking the project, hovering over Build Configurations, and selecting CPU1_APP.

In the case of a Device Firmware Upgrade (DFU) command, the following steps take place:

1. The kernel in CPU1 receives the command packet to perform DFU.
2. To prepare for a new application, the entirety of flash is erased before programming.
3. The kernel prepares to receive the X.509 certificate from the host.
4. The kernel verifies that the incoming certificate is of the proper size and format and derives the size of the incoming image. For now, the certificate is stored in RAM.
   a. While there is no authentication on the certificate or the incoming image during this process. The certificate needs to still be generated properly as the kernel infers the image size from the certificate and is authenticated during standalone flash boot routine.
5. If the certificate size and format are accepted, then the kernel prepares to receive and program the new application to flash
6. The kernel programs the incoming application in chunks of 1024 bits. After each chunk is programmed, the kernel verifies that the data and ECC were programmed correctly into flash.
7. After successfully programming the entire image, the kernel then programs the associated certificate into the first 0x1000 addresses of FLC1 B0/B1. With the certificate programmed to this region of flash, the device is able to boot to the existing application in the standalone flash boot mode.

To generate an application image that can be loaded by the kernel, refer to the post-build step of Section 2.3. And see Section 5.1 for more details on usage.

3.2 Key Provision (HS-FS to HS-KP)

To perform key provisioning on CPU1 of an HS-FS device, compile the KP_APP build configuration of the project. This can be done by right-clicking the project, hovering over Build Configurations, and selecting KP_APP.

Key Provision transitions an HS-FS device to an HS-KP device, and the following events occur:

1. BootROM in UART boot mode receives the UART flash kernel and boots the kernel.
2. The kernel in CPU1 receives a command packet to receive the HSMRt image.
3. The kernel prepares to receive an X.509 certificate as part of the combined image from the host.
4. The kernel verifies that the incoming certificate is of the proper size and format and derives the size of the incoming image. For now, the certificate is stored in RAM.
5. The kernel stores HSMRt image in shared LDAx RAM and requested HSM to authenticate.
6. Upon successful authentication, the HSM begins executing the HSMRt in shared LDAx RAM.
7. The kernel then receives a command packet to receive HSM key.
8. The kernel receives key certificate and places in shared LDAx RAM.
9. The kernel notifies HSM that the keys have been received.
10. Upon successful authentication of the key certificate, HSMRt programs the key into OTP.
11. Perform power-on reset to transition the device into HS-KP.

CAUTION

The Keywriter binary image described in Section 2.1 need to be used as the HSMRt.

Refer to Section 5.5 on steps to perform on the host application.

As mentioned inSection 2.1 , the keywriter firmware is used to program customer keys and transition from HS-FS to HS-KP. The keywriter firmware supports the programming of the following key types and fields:

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
SMPKH	Secondary Manufacturer Public Key Hash SMPKH Length: 512 Bits BCH Length: 64 Bits	• Customer primary key set: Public Key Hash • SMPK is 4096-bit customer RSA Public key used for verifying RSA signature included in Boot Images. • SMPKH Value: 512 bit hash of SMPK generated via SHA512 hashing algo which is split into two parts of 256-bits. That is, SMPKH_P1/SMPKH_P2 • BCH Value: 32-bit BCH is computed using algorithm (288, 261, 7) for each part of SMPKH_P1 and SMPKH_P2. This algorithm supports 3 bit error correction • Checks enforced by KeyWriter: N/A	Secure boot active key to validate Root of Trust for boot Image x.509 certificate when key configuration field KEYREV=1
SMEK	Secondary Manufacturer Encryption Key SMEK Length: 256 Bits BCH Length: 32 Bits	• Customer primary key set: Encryption Key • SMEK is 256-bit customer encryption key for Boot Image Encryption • SMEK Value: Original 256 Bit Symmetric Key used for AES-CBC Encryption of Boot Images • BCH Value: 32-bit BCH is computed using algorithm (288, 261, 7) for 256 bit SMEK. This algorithm supports 3 bit error correction • Checks enforced by KeyWriter: N/A	Active key for secure boot to decrypt the boot Image if encrypted and enabled via x509 certificate when key configuration field KEYREV=1

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
BMPKH	Backup Manufacturer Public Key Hash BMPKH Length: 512 Bits BCH Length: 64 Bits	• Customer backup key set: Public Key Hash • **Note** Device supports Backup key pair which must be provisioned by keywriter and can be activated once product is deployed in field by incremental programming of KEVREV field in field. Note that if backup key pair is not provisioned by keywriter on HS-FS device then this is not possible to program backup key pair later once device transitions to HS-SE. • BMPK is 4096-bit customer RSA Public key used for verifying RSA signature included in Boot Images. • BMPKH Value: 512 bit hash of BMPK generated by SHA512 hashing algo which is split into two parts of 256-bits. That is, BMPKH_P1/BMPKH_P2 • BCH Value: 32-bit BCH is computed using algorithm (288, 261, 7) for each part of BMPKH_P1 and BMPKH_P2. This algorithm supports 3 bit error correction • Checks enforced by KeyWriter: N/A	Secure boot active key to validate Root of Trust for boot Image x.509 certificate when key configuration field KEYREV=2
BMEK	Backup Manufacturer Encryption Key BMEK Length: 256 Bits BCH Length: 32 Bits	• Customer backup key set: Encryption Key • BMEK is 256-bit customer encryption key for Boot Image Encryption • BMEK Value: Original 256 Bit Symmetric Key used for AES-CBC Encryption of Boot Images • BCH Value: 32-bit BCH is computed using algorithm (288, 261, 7) for 256 bit BMEK. This algorithm supports 3 bit error correction • Checks enforced by KeyWriter: N/A	Active key for secure boot to decrypt the boot Image if encrypted and enabled via x.509 certificate when key configuration field KEYREV=2

Key	Description	KeyWriter usage notes	Impact on HS-SE Device
KEYCNT	Key count configuration field Length: 16 bits	• Active Key Sets provisioned in the device, and supports following values • If only Primary Key Set is Plan of Record for device (SMPKH /SMEK) • If both Primary Key Set and Backup Key Set is Plan of Record for device (SMPKH /SMEK) and (BMPKH /BMEK) • **Note** This field controls the active Key sets in the device. If the user wants to enable backup Key set, then a mandatory issue is to provision backup key pair and set KEYCNT as 2 by Keywriter. If this field is programmed as 1, then device can only support primary Key set for Secure boot and this field can not be updated later. • Checks enforced by KeyWriter: N/A	Active Key Sets provisioned in the device for Key Manager to decode and setup the Keys.
KEYREV	Key revision configuration field Length: 16 bits	• Current active key revision in the device for Root of Trust and supports following values • For Primary Key Set (SMPK/SMEK) used for Secure Boot (Recommended config for KeyWriter) • For Backup Key Set (BMPK/BMEK) used for Secure Boot • Checks enforced by KeyWriter: N/A	Current active key revision for Secure Boot
MSV	Model specific value Length: 24 bits BCH: 8 bits	• 24 bit Model Specific Value, Customers can program 24 bit values to differentiate product variants using same SoC either in production flow Or in boot flow of the device. • BCH Value: 8-bit BCH is computed using algorithm (32, 24, 8) for 24 bit MSV. This algorithm supports 2 bit error correction	No Impact for Boot ROM, SW needs to comprehend the usage of this field
SWREV-SBL	SBL software revision Length: 64 bits	• Supports 32 values (1 to 32 states with double redundancy) • **Recommended to set initial value of 0x1**	Enables Anti Roll back feature for SBL Image x.509 Certificate via SWRV extension
SWREV-HSM	TIFS-MCU software revision Length: 64 bits	• Supports 32 values (1 to 32 states with double redundancy) • **Recommended to set initial value of 0x1**	Enables Anti Roll back feature for TIFS-MCU Image x.509 Certificate via SWRV extension
SWREV-APP	Application Image software revision Length: 192 bits	• Supports 64 values (1 to 64 states with double redundancy) • Recommended to set initial value of 0x1	No Impact for Boot ROM, TIFS-MCU needs to comprehend the usage of this fields
EXTENDED OTP	Extended OTP array Length: 1664 bits	1664 bit extended otp array for customer usage	No Impact for Boot ROM

3.3 CPU1 Secure Firmware Upgrade (HS-KP/SE to HS-SE)

To perform a secure firmware upgrade on CPU1 of an HS-KP or HS-SE device, compile the CP_APP build configuration of the project. This can be done by right-clicking the project, hovering over Build Configurations, and selecting CP_APP.

To perform a CPU1 secure firmware upgrade on an HS-KP or HS-SE device, the following events occur:

1. BootROM in UART boot mode receives the UART flash kernel and boots the kernel.
2. The kernel in CPU1 receives a command packet to receive the HSMRt image.
3. The kernel prepares to receive an X.509 certificate as part of the combined image from the host.
4. The kernel verifies that the incoming certificate is of the proper size and format and derives the size of the incoming image. For now, the certificate is stored in RAM.
5. The kernel stores HSMRt image in shared LDAx RAM and requested HSM to authenticate.
6. Upon successful authentication, the HSM begins executing the HSMRt in shared LDAx RAM.
7. The kernel receives a command packet to receive CPU1 flash application image.
8. Kernel receives the X.509 image certificate and shares the same with HSMRt.
9. After successful authentication of the image, HSMRt responds with an acknowledgment, after which flash kernel starts importing the chunk of data via UART into the LDAx memory.
10. After each 16KB (size of LDAx memory) of data received, the flash kernel sends an HSM requests to program the data for further processing.
11. After all chunks are received and programmed, HSMRt is requested to verify the code programmed in HSM active and dormant banks. When the HSMRt firmware authenticates the programmed image against the certificate, the certificate is further programmed to make sure of a successful boot in the subsequent power cycles.
12. Upon successful authentication, the HSM programs the firmware to CPU1 flash.
    a. If the device is previously in HS-KP, then the device is transitioned to HS-SE.

Refer to Section 5.6 on steps to perform on the host application.

CAUTION

For HS-KP or HS-SE devices, the DPL interrupt LINK and STACK pointer needs to be set to LINK2 and STACK2, respectively. To adjust the setting, open the syscfg file in CCS, select *Clock* under TI Driver Porting Layer (DPL) section.

The Keywriter binary image described in Section 2.1 needs to be used as the HSMRt.

3.4 HSM Firmware Upgrade (HS-KP/SE to HS-SE)

To perform a firmware upgrade on the HSM of an HS-KP or HS-SE device, compile the CP_APP build configuration of the project. This can be done by right-clicking the project, hovering over Build Configurations, and selecting CP_APP.

To perform a CPU1 firmware upgrade on an HS-KP or HS-SE device, the following events occur:

1. BootROM in UART boot mode receives the UART flash kernel and boots the kernel.
2. The kernel in CPU1 receives a command packet to receive the HSMRt image.
3. The kernel prepares to receive an X.509 certificate as part of the combined image from the host.
4. The kernel verifies that the incoming certificate is of the proper size and format and derives the size of the incoming image. For now, the certificate is stored in RAM.
5. The kernel stores HSMRt image in shared LDAx RAM and requested HSM to authenticate.
6. The kernel receives a command packet to receive HSM flash application image.
7. Kernel receives the X.509 image certificate and shares the same with HSMRt.
8. After successful authentication of the image, HSMRt responds with an acknowledgment, after which flash kernel starts importing the chunk of data via UART into the LDAx memory.
9. After each 16KB (size of LDAx memory) of data received, the flash kernel sends an HSM requests to program the data for further processing.
10. After all chunks are received and programmed, HSMRt is requested to verify the code programmed in HSM active and dormant banks. When the HSMRt firmware authenticates the programmed image against the

certificate, the certificate is further programmed to make sure successful boot in the subsequent power cycles.

11. Upon successful authentication, the HSM programs the firmware to CPU1 flash.
    a. If the device is previously in HS-KP, then the device is transitioned to HS-SE.

Refer to Section 5.6 on steps to perform on the host application.

CAUTION

For HS-KP or HS-SE devices, the DPL interrupt LINK and STACK pointer needs to be set to LINK2 and STACK2, respectively. To adjust the setting, open the syscfg file in CCS, select *Clock* under TI Driver Porting Layer (DPL) section. The Keywriter binary image described in the Hardware Security Module section needs to be used as the HSMRt.

3.5 SECCFG Code Provisioning (HS-KP/SE to HS-SE)

To perform SECCFG programming of an HS-KP or HS-SE device, compile the CP_APP build configuration of the project. This can be done by right-clicking the project, hovering over Build Configurations, and selecting CP_APP.

To program a new image to SECCFG on an HS-KP or HS-SE device, the following events occur:

1. BootROM in UART boot mode receives the UART lash kernel and boots the kernel.
2. The kernel in CPU1 receives a command packet to receive the HSMRt image.
3. The kernel prepares to receive an X.509 certificate as part of the combined image from the host.
4. The kernel verifies that the incoming certificate is of the proper size and format and derives the size of the incoming image. For now, the certificate is stored in RAM.
5. The kernel stores HSMRt image in shared LDAx RAM and requested HSM to authenticate.
6. Upon successful authentication, the HSM begins executing the HSMRt in shared LDAx RAM.
7. The kernel receives a command packet to receive Sec Cfg image.
8. Kernel receives the X.509 image certificate and shares the same with HSMRt.
9. After successful authentication of the image, HSMRt responds with an acknowledgment, after which flash kernel starts importing the Sec Cfg data via UART into the LDAx memory.
10. After all the SecCfg data are received and programmed, the HSMRt is requested to verify the SecCfg programmed in the dormant banks with valid counter values. When the HSMRt authenticates the programmed image against the certificate, the certificate is further programmed to make sure successful boot in the subsequent power cycles
    a. Note in the case of HS-SE device, the decision of programming of the certificate is made on the swap value of the SSU registers.

Refer to Section 5.6 on steps to perform on the host application.

CAUTION

For HS-KP or HS-SE devices, the DPL interrupt LINK and STACK pointer needs to be set to LINK2 and STACK2, respectively. To adjust the setting, open the syscfg file in CCS, select *Clock* under TI Driver Porting Layer (DPL) section. The Keywriter binary image described in Section 2.1 needs be used as the HSMRt.

4 Host Application: UART Flash Programmer

4.1 Overview

The UART flash programmer is a program in command line interface that runs on the host PC and to be interfaced with the bootROM or UART flash kernel on the target device. The program can be easily incorporated into scripting environments for applications like production line programming.

UART flash programmer is written in C++ and can be built as either a Microsoft Visual Studio® or CMake project in Windows® or Linux®. The project and the source can be found in the tools directory of the SDK (*f29h85x-sdk_x_xx_xx_xx > tools> flash_programmers > uart_flash_programmer*).

Two Windows pre-compiled executables are provided:

- uart_flash_programmer.exe (x86_64): The program starts off with sending the SBL (secondary bootloader, in this case the UART flash kernel) image to BootROM in UART boot mode, which BootROM proceeds to hand off the device control to SBL. Then the user is prompted with options to communicate and perform commands in the kernel.
  - **Note**

    BootROM follows a strict state machine sequence of expecting a SBL boot prior to any HSM service, therefore, please use this executable for operations regarding HSM. For example, use this executable for key and code provisioning.
  - Refer to Section 2.2 on the specifics of BootROM.
- uart_flash_programmer_appIn.exe (x86_64): The program bypasses sending kernel to BootROM and goes straight to prompting the user with command options. This is useful to debug the custom kernel.
  - User can either load the kernel directly to the device via CCS or an alternative image load methods.
  - User can un-define the *kernel* macro to include Common.h and rebuild the project to make an executable with this behavior.

For Linux users, a shell script *build_cmake.sh* is provided to automate the CMake build. The default source code generates an executable identical to uart_flash_programmer.exe.

To use this tool to program the F29H85x device, make sure that the target device has been reset and is in the UART boot mode with UART pins connected to the host PC serial port via a UART transceiver. Refer to Section 5.1 for the setup specifics.

The supported parameters can be displayed by supplying -h or --help as a parameter.

```
Syntax:
uart_flash_programmer.exe -d f29h85x -p <COM/tty Port> -k <uart kernel image>.bin -a <CPU1
application image>.bin -e <F29x alternate entry address>-r <HSM runtime image>.bin -f <user HSM
keys>.bin -t <CPU1 application image>.bin -g <HSM application image>.bin -c <sec cfg program
image>.bin -q -w
```

Table 4-1. Supported Parameters

-d, --device <device>	The name of the device to connect and load to. Currently, F29H85x is the only device supported.
-k, --kernel <file>	The file name for the CPU1 flash kernel
-a, --appcpu1 <file>	The file name for CPU1 application image to download via DFU for HS-FS device.
-r, --hsmrt <file>	The file name for RAM-based HSM runtime image. This is required to load the runtime image prior to key and code provisioning.
-f, --hsmkeys <file>	The file name for HSM certificate key image used to convert devices to HS-KP.
-t, --cpappcpu1 <file>	The file name for Flash-based CPU1 application image via code provisioning for devices in HS-KP/HS-SE.
-g, --cpapphsm <file>	The file name for Flash-based HSM application image via code provisioning for devices in HS-KP/HS-SE.
-s, --cpseccfg <file>	The file name for the image used to program SEC CFG section in non-main flash via code provisioning.

Table 4-1. Supported Parameters (continued)

-e, --entry <hex_num>	An optional parameter to override the default entry address for C29 CPU1 application. For example, pass 10001000 for the hex address 0x10001000. TI recommends to use the default 10001000 as the entry address because that is the bootROM flash entry point.
-h, --help	Shows the help dialogue.
-q, --quiet	Quiet mode. Suppress all non-essential printouts.
-l, --log <file>	Log mode. Redirect All non-essential printouts to the specified file. Overrides quiet mode if specified.
-w	Wait for a key press before exiting.

-d, -p, -k are mandatory parameters.

Note

All files programmed to the F29H85x MUST be in binary format and combined with an X.509 certificate. The first 0x1000 bytes of the binary file must contain the certificate. See #none# for the instruction on post-build steps .

4.2 Build UART Flash Programmer with Visual Studio

UART flash programmer can be compiled and run as a Visual Studio project.

1. Navigate to the uart_flash_programmer directory.
2. Double click the uart_flash_programmer.sln to open the Visual Studio project.
3. When Visual Studio opens, select Build → Build Solution.
4. After Visual Studio completes the build, select Debug → uart_flash_programmer properties.
5. Select Configuration Properties →Debugging.
6. Select the input box next to the Command Arguments.
7. Type the parameter arguments. The arguments are described in Section 4.1.
   - Example:

```
-d f29h85x -k flash_kernel.bin -a application.bin -p COM34
```

8. Click Apply and OK.
9. Select Debug → Start Debugging to begin running the project.
10. The output from Visual Studio command prompt needs to match what is shown in Figure 5-1.

4.3 Build UART Flash Programmer with CMake

A shell script, build_cmake.sh is provided to automate the CMake project build. The script encapsulates the steps below on building a CMake project:

- Instructions on building with CMake:
  - In the terminal, proceed to create a build folder and cd into folder via
    - **mkdir build && cd build**
  - Generate CMake artifacts via
    - **cmake -S .. -DCMAKE_BUILD_TYPE={Debug/Release}**
  - Build the CMake project via
    - **cmake --build .**

The generated executable is called *uart_flash_programmer* under the build folder.

Both executable generated by Visual Studio and CMake have the exact same utility and are used in the same fashion. In the subsequent section, uart_flash_programmer.

4.4 Packet Format

Packets are sent in a standard format between the host and device. The packet allows for a variable amount of data to be sent while making sure the correct transmission and reception of the packet. The header, footer and checksum fields help to make sure that the data was not corrupted during transmission. The checksum is the summation of the bytes in the command and data fields.

Note that the a shared *f29h85x_kernel_commands_cpu1.h* are used by both uart flash kernel and the host programmer to synchronize packet macros such as header, footer, nak, ack, command and status error values. Users are welcome to make new or modifications to existing macros.

Table 4-2. Packet Format

Header	Data Length	Command	Data	Checksum	Footer
2 Bytes	2 Bytes	2 Bytes	Length Bytes	2 Bytes	2 Bytes
0x1BE4	Length of Data in Bytes	Command	Data	Checksum of Command and Data	0xE41B

Both the host and device respond to a packet with an ACK or NAK.

Table 4-3. ACK or NAK Values

ACK	NAK
0x2D	0xA5

4.5 Kernel Commands

A brief description of the commands and the associated kernel behavior are provided in Table 4-4.

Table 4-4. CPU1 Kernel Command Flows

Kernel Commands	Command Code	Description
DFU CPU1	0x01	1. Receive the command packet with no data 2. Receive the flash application byte-by-byte 3. Program and verify the application 4. Send flash status packet 5. Send message for the final status
Load HSMRt Image	0x0B	1. Receive the command packet with no data 2. Receive the HSMRt byte-by-byte 3. Place HSMRt in shared LDAx RAM 4. Send flash status packet 5. Wait to receive status of the HSM client 6. Send message for the final status
Load HSM Keys	0x0E	1. Receive the command packet with no data 2. Receive the HSMRt byte-by-byte 3. Place key certificate in shared LDAx RAM so the HSM can program them 4. Waits for HSM authentication status in forms of IPC message 5. Send status packet 6. Forward the status log messages from HSM
Load HSM Code Provisioning Image (firmware upgrade for HSM)	0x0D	1. Receive the command packet with no data 2. Receive HSM firmware byte-by-byte, send the included key certificate for HSM validate 3. Proceed to store the firmware in shared LDAx RAM for HSM to validate in chunks (if the firmware exceeds the size of RAM) 4. Waits for HSM to perform integrity checks on the programmed firmware 5. Send flash status packet 6. Send status log message
Load C29 Code Provisioning Image (firmware upgrade for HS-SE C29)	0x10	Identical to HSM Code Provisioning Image
Program SECCFG section (HS-SE)	0x0C	1. Receive the command packet with no data 2. Receive SECCFG image byte-by-byte and store in shared RAM for the HSM to validate 3. Send status packet 4. Send status log message
Run CPU1	0x09	1. Receive the packet with no data 2. Branch to application entry point
Reset CPU1	0x0A	1. Receive the packet with no data 2. Enable WatchDog and allow to cause a reset

5 Example Usage

The kernel described above is available in the F29H85x MCU SDK at: *mcu_sdk_f29h85x/examples/driverlib/single_core/flash/uart_flash_kernel*. The host application can be found in the same SDK as well (*mcu_sdk_f29h85x /tools/ flash_programmers /uart_flash_programmer*). This section details how the kernel and host programmer can be used together to perform firmware upgrades on an F29H85x device.

5.1 Loading the Flash Kernel onto the Device

The first step of performing a firmware upgrade on the F29H85x is to load the flash kernel to RAM via the ROM bootloader. Make sure to load the appropriate build configuration for the desired behavior. For example, for CPU1 DFU of an HS-FS device, make sure the CPU1_APP build configuration is selected when compiling the kernel. Here are the steps required to achieve this:

1. Configure the Boot Mode Select Pins to put the device in UART boot mode.
2. Reset the device.
3. Send the kernel to the device via the UART host programmer.

These steps are described in more detail below.

5.1.1 Hardware Setup

Set up the device correctly to be able to communicate with the host PC running the UART flash programmer.

1. The first thing to do is make sure the boot mode select pins are configured properly to boot the device to UART boot mode.
2. Next, connect the appropriate UART boot loader GPIO pins to the Rx and Tx pins that are connected to the host PC serial port. A transceiver is often needed to convert a Virtual serial port from the PC to GPIO pins that can connect to the device. On some systems, like the controlSOM, an FTDI chip is used to interface the GPIO pins for UART communication to a USB Virtual serial port. Refer to the device-specific user's guide for the controlSOM to get information on the switch configuration needed to enable UART communication.
3. After the hardware is set up correctly to communicate with the host, reset the device. This is to boot the device to UART boot mode.

5.1.2 Running the UART Flash Programmer

Note

TI recommends to reset the device before running UART flash programmer so bootROM does not timeout while waiting for UART data.

1. Navigate to the folder containing the compiled uart_flash_programmer executable.
2. Run the executable uart_flash_programmer.exe with the following command structure:

```
:> uart_flash_programmer.exe -d f29h85x -p COM41 -k ex3_uart_flash_kernel.bin --appcpu1
c29_cpu1_application.bin --hsmrt HSM_runtimeImage.bin --hsmkeys HSM_customKeyCert.bin --
cpseccfg sec_cfg_cert.bin --cpappcpu1 c29_cpu1_application.bin --cpapphsm hsm_application.bin
```

   The program automatically connects to the device and download the CPU1 kernel (-k option) into RAM. If the certificate is valid and authentication is successful, then the device begins executing the kernel in LPAx RAM. Now, the CPU1 kernel is running and waiting for a packet from the host.
3. The uart_flash_programmer prints the options to the screen to choose from that is sent to the device kernel (see Figure 5-1).
4. The same options is re-printed after succession of the previous operation.

```
F29x UART Firmware Programmer
Copyright (c) 2024 Texas Instruments Incorporated. All rights reserved.

COM51

What operation do you want to perform?
         1-DFU CPU1 (HS-FS only)
         2-Load HSMRt Image (prerequisite for KP & CP)
         3-Load HSM Keys (HS-KP Key Provision)
         4-Program Sec Cfg (HS-SE Code Provision)
         5-Load HSM Image (HS-SE Code Provision)
         6-Load C29 Image (HS-SE Code Provision)
         7-Run CPU1
         8-Reset CPU1
         0-DONE
```

Figure 5-1. UART Flash Programmer Prompt for Next Command After Downloading Flash Kernel to RAM

5.2 CPU1 Device Firmware Upgrade (HS-FS only)

After the UART flash kernel has been loaded onto the device RAM, a device firmware upgrade can be performed. This section discusses how to do this on a HS-FS device.

Figure 6-1 displays the options available when the flash kernel is running in RAM. For a firmware upgrade on an HS-FS device, select DFU CPU1 (option 1). After successful, select Run CPU1 (option 7) to branch from kernel to the newly loaded application. Alternatively, reset CPU1 (option 8) is also valid if the device is set to flash boot mode.

5.3 Convert HS-FS to HS-SE

The following and subsequent sections discusses how to use the flash programmer commands to convert an HS-FS device to an HS-SE device.

In an overarching view, to convert an default HS-FS device, user must first:

- Goes through Key Provision, which converts to an intermediate state of HS-KP. (Key Provisioned, but no image has been flashed thus far).
- Goes through Code Provision, which converts HS-KP device to HS-SE upon any successful flash unto the flash banks.
  - The flash programmer has provided three options to program flash as part of Code Provisioning:
    - CPU1 flash
    - HSM flash
    - SEC CFG flash (part of non-main flash bank)
- After the first Code Provision and the device is in HS-SE, any subsequent Code Provisions are still allowed to program flash. And the device maintains the HS-SE status.

Even though any of the three provided Code Provision option uplifts the device into HS-SE, TI recommends to program SEC CFG first, prior to either CPU1 flash or HSM flash programming.

Over the next subsequent sections, each provision flow is discussed in chronological order.

CAUTION

BootROM follows a state machine sequence of expecting a SBL boot prior to any HSM services, as such, use the regular *non-appln* version of the flash programmer. Refer to Section 4.1 on the difference of the two.

5.4 Loading a RAM-based HSMRt Image

For all functionality associated with Key and Code Provision, loading a RAM-based HSM runtime (HSMRt) image is required.

HSMRt service HSM requests from the flash kernel, and once authenticated, this programs a set of field required to convert the device to a different state. For more information on the fields being programmed, refer to the *OTP Keywriter* section in Section 2.1.

After the flash kernel has been booted onto the device via BootROM in UART boot mode, select Load HSMRt Image (option 2). The host sends a RAM-based HSMRt image to the C29 CPU1, which is subsequently validated by HSM. Upon successful validation, this RAM-based HSMRt begins executing in shared RAM.

Note that two separate HSMRt binary are needed for Key and Code Provision, each with different key certificate. In Key Provision for HS-FS, the key certificate needs to be the default TI-provided key, whereas Code Provision is using the user's custom key certificate.

5.5 Key Provision (HS-FS to HS-KP)

As aforementioned, Key Provision permanently converts HS-FS device to HS-KP by supplying a user key certificate to replace the default TI-provided key certificate.

The process on the Flash Programmer is as follows:

- Start the flash programmer, wait for flash kernel to be downloaded onto the device and prompts the user with options listed in Figure 6-1.
- Select Load HSMRt Image (option 2), wait until completion.
- Select Load HSM keys (option 3), wait until completion.
- Perform a power-on reset.

In this process, HSMRt can use the default TI-signed key certificate as the device is still in HS-FS. Flash kernel won't be authenticated in HS-FS so any certificate suffice, and the HSM key certificate need to be signed by a custom key.

The necessary parameters are -d (--device), -p (--port), -k (--kernel), -r (--hsmrt), -f (--hsmkeys).

For instance:

```
uart_flash_programmer.exe -d f29h85x -p COM41 --kernel ex3_uart_flash_kernel.bin --hsmrt
HSM_runtimeImage.bin --hsmkeys HSM_customKeyCert.bin
```

For procedures and important notes on building the flash kernel, refer to Section 3.2.

5.6 Code Provision (HS-KP/SE to HS-SE)

After Key Provision converts the device into HS-KP, Code Provision can be conducted to either flash CPU1/HSM application or SEC CFG section into the corresponding flash banks.

The process on the Flash Programmer is as follows:

- Start the flash programmer, wait for flash kernel to be downloaded onto the device and prompts the user with options listed in Figure 6-1.
- Select Load HSMRt Image (option 2), wait until completion.
- Select either one of the three options and wait until completion.
  - Program Sec Cfg (option 4)
  - Load HSM Image (option 5)
  - Load C29 CPU1 Image (option 6)
- Continue to perform the other two Code Provision options if needed.

Note that all image (flash kernel, HSMRt, application/sec cfg) requires the user key certificate as part of the binary image. Refer to Section 2.3 on instructions to generate key certificate with custom key.

The necessary parameters are -d (--device), -p (--port), -k (--kernel), and either -t (--cpappcpu1), -g (--cpapphsm), -s (--cpseccfg) depending on the Code Provision options.

For instance:

```
uart_flash_programmer.exe -d f29h85x -p COM41 --kernel ex3_uart_flash_kernel.bin --hsmrt
HSM_runtimeImage.bin --cpseccfg sec_cfg_cert.bin --cpappcpu1 c29_cpu1_application.bin --cpapphsm
hsm_application.bin
```

For procedures and important notes on building the flash kernel for Code Provisioning, refer to Section 3.3.

6 Troubleshooting

Answers to some common issues encountered by users when utilizing the UART flash kernel are provided below.

6.1 General

Question: I cannot find the UART flash kernel projects, where are the projects?

Answer:

Device	Build Configurations	Location
F29H85x	CPU1_RAM	mcu_sdk_f29h85x\examples\driverlib\single_core\flash\uart_flash_kernel

6.2 UART Boot

Question: I cannot download the UART kernel to RAM in UART boot mode, what course of action do I take?

Answer:

- The most common issue users encounter is that the correct boot pins for UART boot mode are not used. For example, on the F29H85x devices, UART boot has five options for GPIO pins to use. Make sure that the pins for the default option are not being used for something else. If the pins are already used, then make sure that another UART boot option is used, so that the device can be connected to another set of pins. Make sure that the UART kernel project uses this UART boot GPIO option as the parameter for UART_GetFunction() as well.
- Make sure there is an associated X.509 certificate in the first 0x1000 bytes of the binary file. Please refer to Section 2.2dev for specific instructions.
- Make sure the user uses a high quality UART transceiver to minimize any issues with the baud rate.
- For baud rate and connection issues, try running UART loopback and echoback examples for the device (user can find these in the MCU SDK driverlib folders for the device in concern).

6.3 Application Load

Question: I can download the kernel to the device successfully, but I cannot successfully load the application to flash. What do I need to check?

Answer:

- Make all sections in the linker cmd file that are allocated to flash are aligned to 512-bit boundaries. This can be done by adding palign(32) to the appropriate sections as shown below.

```
.text : {} > FLASH_RP0, palign(32)
```

- Make sure there is an appropriate X.509 certificate in the first 0x1000 bytes of your binary file. Refer to Section 3.1 for specific instructions.

7 Summary

As applications grow in complexity, the need to fix bugs, add features, and otherwise modify embedded firmware is increasingly critical in end applications. Enabling functionality like this can be easily and inexpensively accomplished through the use of bootloaders.

This application note aims to solve this problem by the introduction of a secondary bootloader (SBL), which is the UART flash kernel in this case. This document discusses the specifics of the kernel and the host application tool found in the F29H85x MCU SDK.

8 References

1. Texas Instruments: *F29H85x and F29P58x Real-Time Microcontrollers*, technical reference manual
2. Texas Instruments, F29H85x and F29P58x Real-Time Microcontrollers, data sheet

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