Application Note

Ethernet Firmware Over-the-Air using MCU+ SDK on AM243x/AM64x and AM275x Devices

Aryamaan Chaurasia, Prashant Shivhare, Teja Rowthu, Vaibhav Kumar, Meet Thakar

ABSTRACT

This application note describes the implementation of Ethernet-Based Firmware Over-The-Air update capabilities on Texas Instruments Sitara™AM243x/AM64x and AM275x microcontroller families. This design leverages TI's MCU+ SDK infrastructure, including the OSPI drivers and interface, Enet low level driver stack, bootloader framework, and security mechanisms such as authentication and encryption to provide a secure remote firmware update mechanism for industrial and automotive applications. This application note provides system architects and embedded engineers with a comprehensive understanding of how ENET FOTA integrates with TI's software ecosystem.

Table of Contents

1 Introduction 2
2 Mechanism of ENET FOTA 3
3 Ethernet Setup 8
3.1 Linux System 8
3.2 Windows System 8
4 Authenticated FOTA 9
4.1 Signing the Firmware 9
5 Encrypted FOTA 10
5.1 Signing and Encrypting the Firmware 10
6 Software Structure 12
6.1 AM243x/AM64x 12
6.2 AM275x 12
7 Expected Outputs 13
7.1 AM243x/AM64x 13
7.2 AM275x 14
8 Performance Metrics 15
9 Summary 15
10 References 15

Trademarks

Sitara™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

Firmware Over-the-Air (FOTA) is a method of remotely updating the software embedded in electronic devices without requiring physical access or manual intervention. When delivered through Ethernet, FOTA takes advantage of the reliability and speed of wired networking. This approach is particularly useful in industrial automation, smart infrastructure, and any application where devices are deployed at scale or hard to reach locations.

Traditional firmware updates often require a technician or an engineer to connect directly to each device using a serial interface, USB cable, or a dedicated programming tool. This process is time consuming, expensive, and impractical when hundreds or thousands of devices are installed across a factory floor, building, or a city.

Ethernet-Based Firmware Over-The-Air Updates, or ENET FOTA, extends this remote update capability by leveraging standard Ethernet network infrastructure. This approach allows embedded devices equipped with Ethernet connectivity to receive complete firmware updates over the same network they use for operational communication. By utilizing existing network infrastructures, organizations can deploy critical security patches, feature enhancements, and bug fixes across the entire device ecosystem without scheduling maintenance windows or dispatching technicians to physical locations.

2 Mechanism of ENET FOTA

ENET FOTA updates the firmware over Ethernet by receiving a new image, validating the authenticity and integrity using x509 certificates, storing it in non-volatile Flash memory, and then rebooting to run the updated firmware. The process requires minimal user intervention while verifying only authorized updates are applied.

For ENET FOTA, the default Flash Part available on AM243x/AM64x and AM275x devices is the S28HS512T Serial NOR OSPI Flash. The Flash memory layout depicted in Figure 2-1 is used for ENET FOTA:

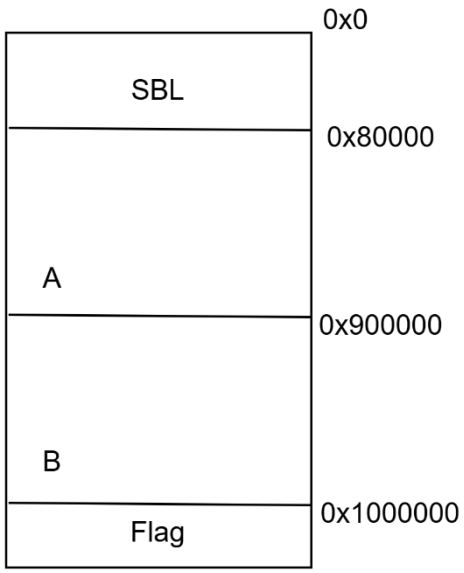


Figure 2-1. Flash Memory Configuration for ENET FOTA

1. The SBL is present at 0x0 by default. This offset remains unchanged, as ROM expects the SBL to be present at the 0x0 offset.
2. The application images will be loaded and/or written to the memory locations A and B, that are starting from the memory locations 0x80000 and 0x900000 respectively.
3. A 1-Byte boot flag stored in 0x1000000 Flash memory indicates which memory location the SBL OSPI bootloader must boot from after reset.

The following steps are performed for ENET FOTA:

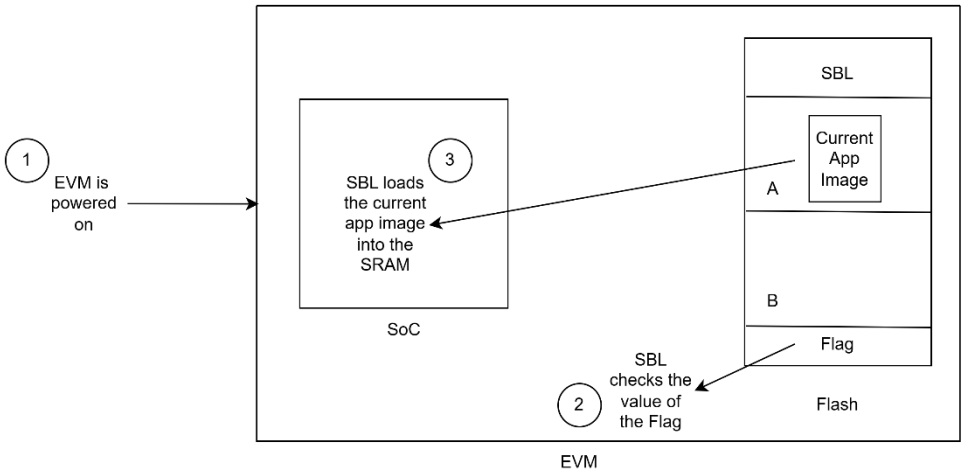


Figure 2-2. Steps 1-3 of ENET FOTA

The EVM is powered on and the SBL checks the value of the flag and loads the current application image present in memory location A into the SRAM.

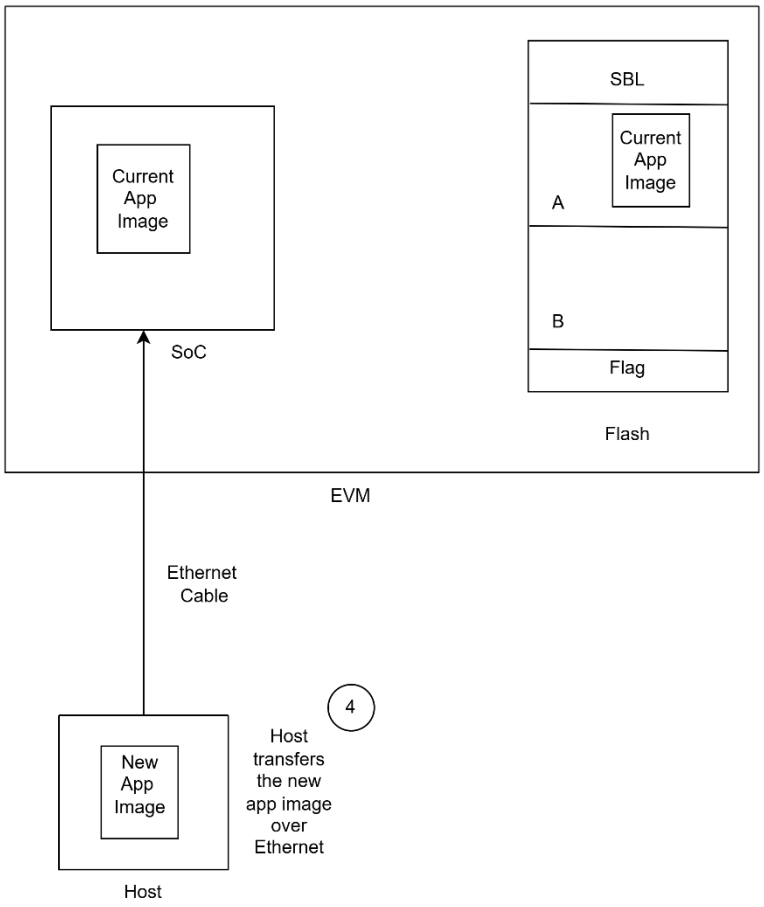


Figure 2-3. Step 4 of ENET FOTA

The host transfers the new application image over Ethernet. The host and target device must be connected to the same local network with compatible network settings. The current application image listens for incoming firmware updates on a predefined UDP port.

The firmware image transmits via UDP in manageable packets. Each packet contains a firmware segment plus sequencing information, allowing the device to reconstruct the complete image regardless of arrival order or retransmission requirements.

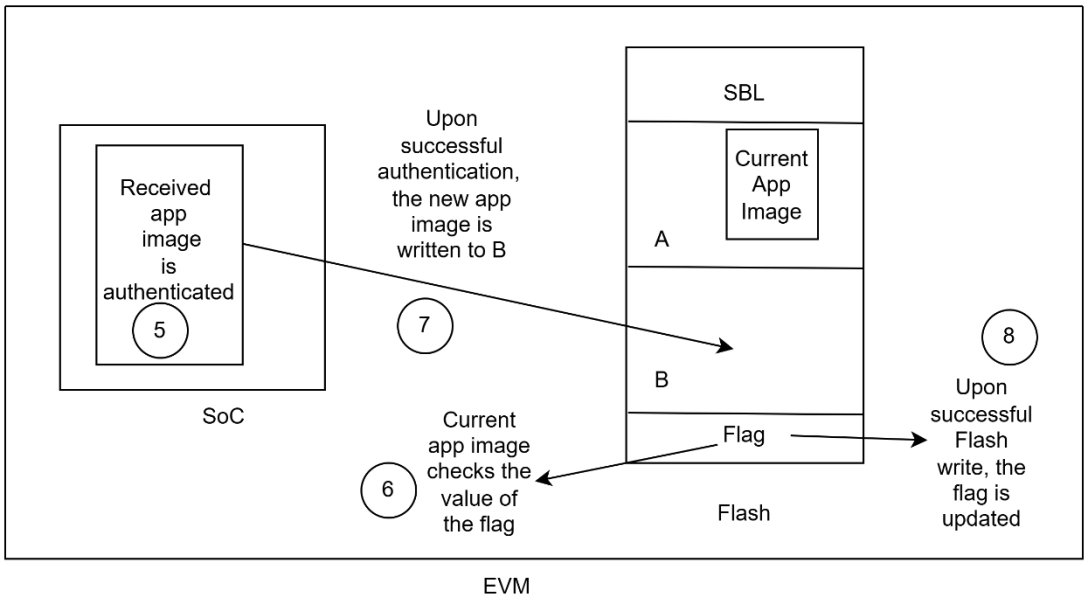


Figure 2-4. Steps 5-8 of ENET FOTA

The application stores incoming packets in DDR memory buffers and acknowledges receipt from the host. Once the complete image is received, the application authenticates the firmware to verify that it originates from a trusted source and has not been modified during transmission.

If authentication fails, the device rejects the firmware. Upon successful authentication, the application writes the new firmware to the new Flash memory location and updates the boot flag only after the write completes successfully.

The application uses a dual Flash memory location where the new firmware writes to the new memory location B while the current memory location A preserves the current working firmware.

If power fails during programming or the new firmware is corrupted, the original firmware remains intact in the current memory location. Since the boot flag updates only after successful completion, any failure results in booting the original firmware, verifying the device remains operational.

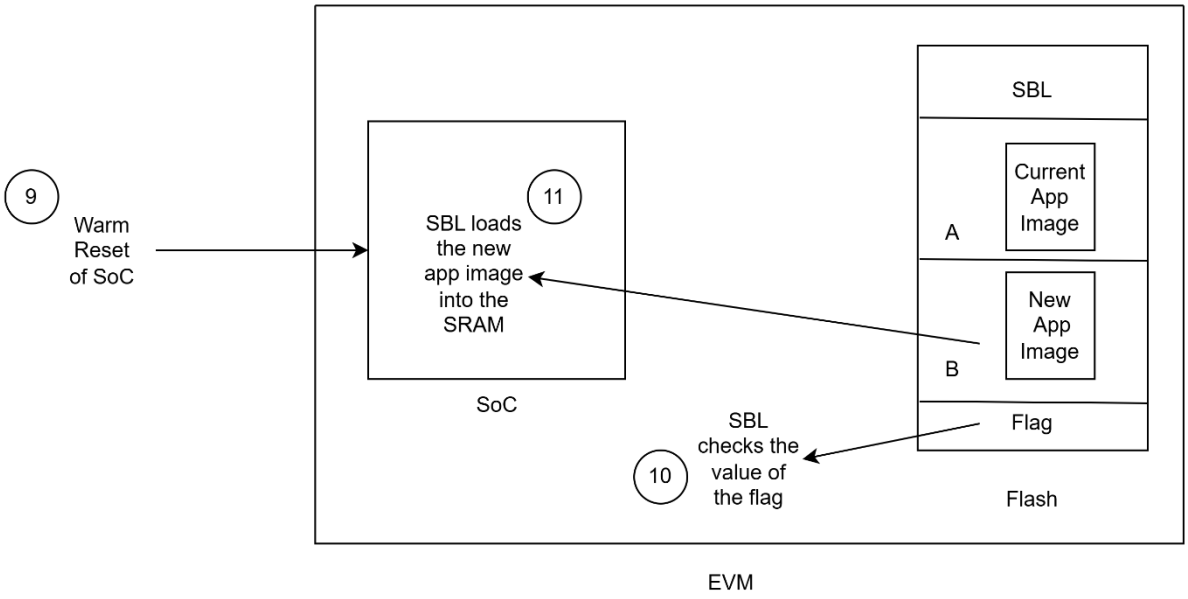


Figure 2-5. Steps 9-11 of ENET FOTA

After the flag is updated successfully, a warm reset of the SoC is performed. The SBL then checks the value of the updated flag and loads the new application image from memory location B into the SRAM.

Similarly, when the new application image is running from memory location B, an incoming application image is stored in memory location A. Since location A is now unused, the old application image previously present is overwritten. The diagrams below represents the switching characteristic of ENET FOTA with the two memory locations A and B.

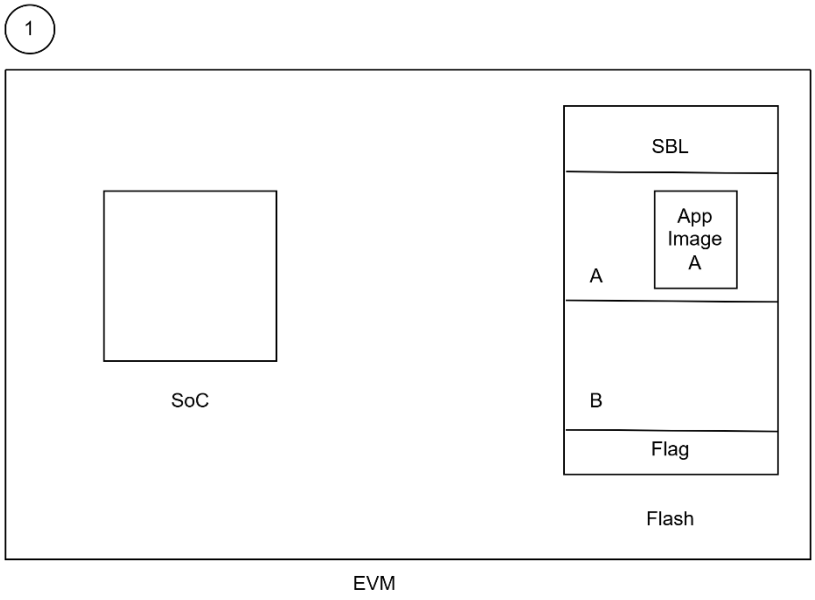


Figure 2-6. Layout 1

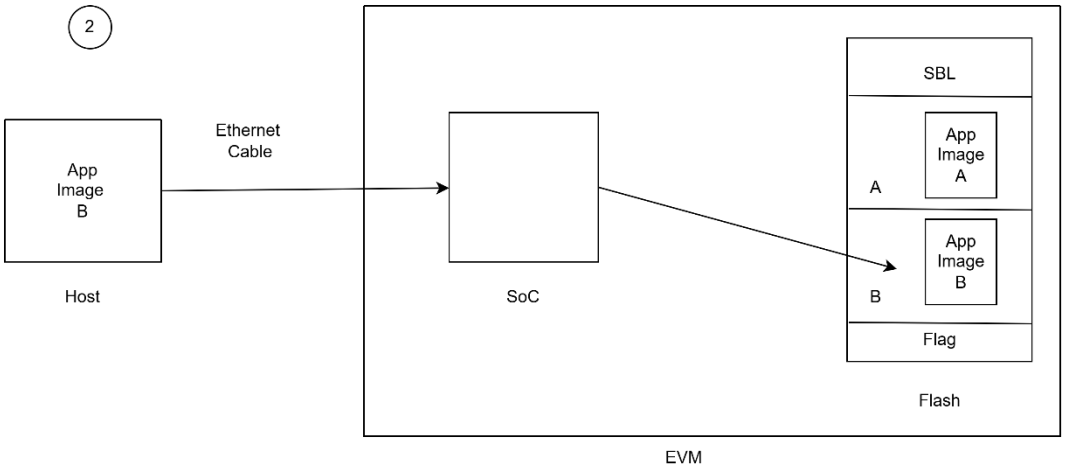


Figure 2-7. Layout 2

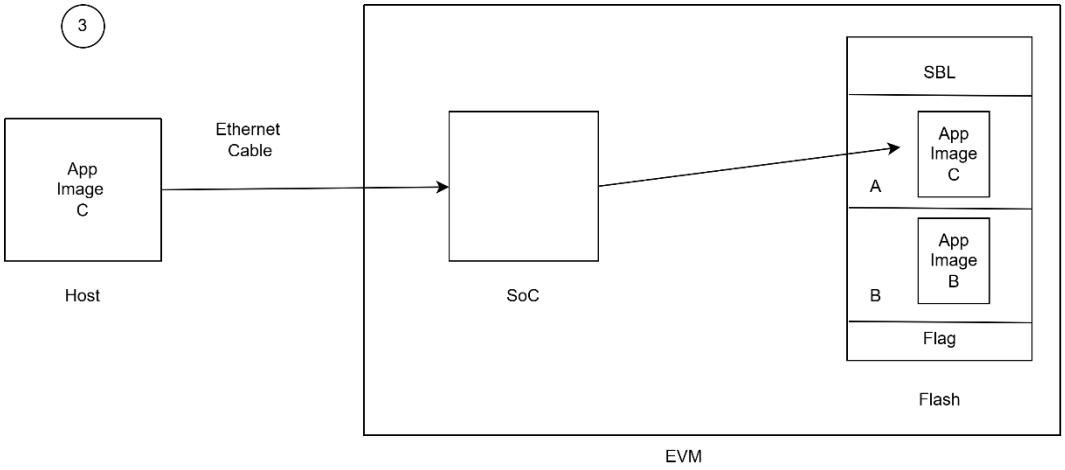


Figure 2-8. Layout 3

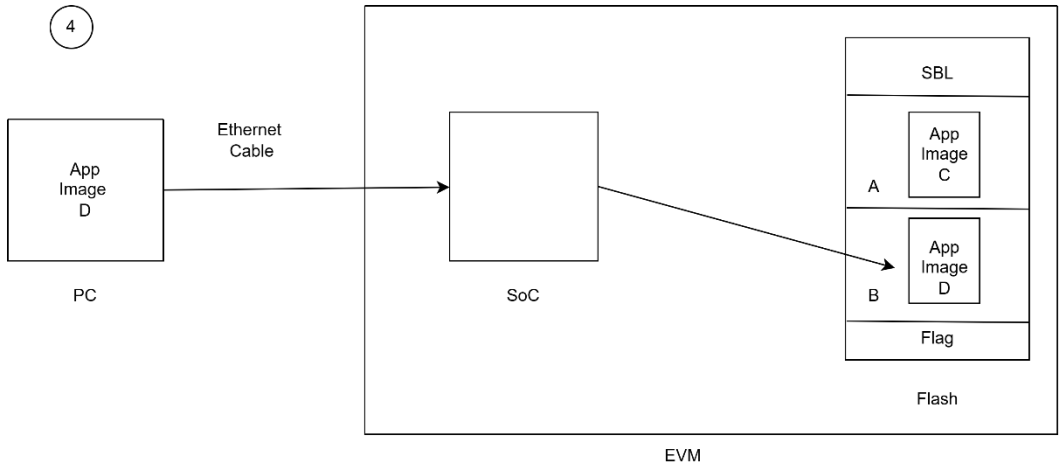


Figure 2-9. Layout 4

3 Ethernet Setup

The following steps should be performed for a successful link between the Host and the EVM:

3.1 Linux System

- Connect the Ethernet cable between the Host and the EVM.
- Assign an IP Address having the same subnet as the EVM, using the following command:

```
sudo ifconfig <interface-name> <Host IP Address> <Subnet mask>
```

1. example:`sudo ifconfig eno1 192.168.0.193 netmask 255.255.255.0`
2. Run the `ip link` or `ifconfig` command to find the name of your network interface.

- Add a static ARP entry with the command below:

```
sudo arp -i <interface-name> -s <IP Address of EVM> <MAC-Address of EVM>
```

1. example: `sudo arp -i eno1 -s 192.168.0.195 44:6b:1f:2c:2d:25`

3.2 Windows System

- Connect the Ethernet cable between the Host and the EVM.
- Open Ethernet settings on your PC >> select corresponding Ethernet adapter.
- Edit the IP settings with the IP address as 192.168.0.193, Subnet Prefix Length as 24 and Gateway as 192.168.0.195
- Make sure to set the connection as private and metered connection is set to off.
- Creating a static ARP entry requires admin privileges. Run the following commands in PowerShell as admin.

```
New-NetNeighbor -InterfaceIndex <ifIndex> -IPAddress '192.168.0.195'
-LinkLayerAddress '<EVM MAC Addr>' -State Permanent
```

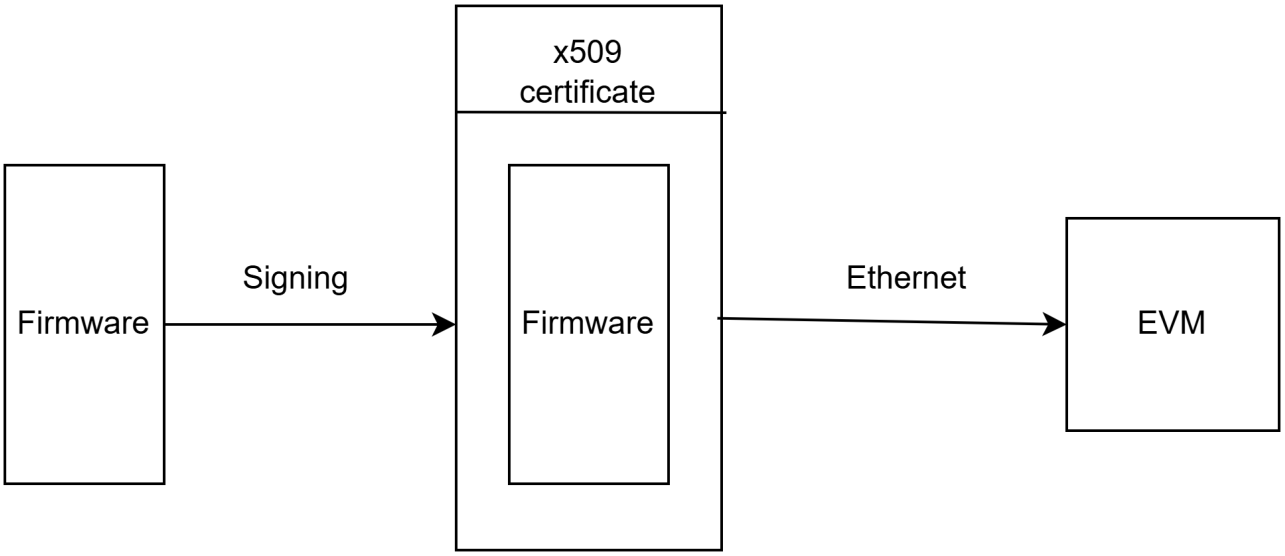
1. Replace <`ifIndex`> with the interface index of the connection between PC and EVM.
2. To find out the interface index corresponding to the Ethernet interface between the PC and the EVM, use the following PowerShell command: `Get-NetAdapter`
3. Replace <`EVM MAC Addr`> with the MAC Address of the EVM, as a continuous string like `70ff761decf2`
4. An example command to set the ARP looks as below:

```
New-NetNeighbor -InterfaceIndex 11 -IPAddress '192.168.0.195' -LinkLayerAddress
'70ff761decf2' -State Permanent
```

Note

For both Windows and Linux systems, open the enet_fota.h file and set the values of enet_host_pc_mac_address, enet_port, enet_source_ip_address, and enet_destination_ip_address according to your setup; then, in the enet_fota_file_transfer.py file, assign the correct IP addresses and port numbers to the hostIP, hostPort, boardIP, and boardPort arguments.

4 Authenticated FOTA



For Authenticated FOTA, the firmware to be updated is post-processed to sign the firmware for authenticity. Once the signed firmware is received on the EVM, it is authenticated before the actual firmware is programmed. If the authentication is successful, the signing certificate is stripped, and the firmware is programmed into the media.

The following steps are required to perform the above flow:

4.1 Signing the Firmware

Run the following commands to sign the firmware:

For AM243x/AM64x devices:

- Change the directory to:`${MCU_PLUS_SDK_PATH}/source/security/security_common/tools/boot/signing`
- Run the following command:

  ```
  python appimage_x509_cert_gen.py --bin </path/to/firmware> --authtype 1 --key $(APP_SIGNING_KEY) --output </path/to/signed_firmware>
  ```

  – This command is found in the Makefile under ti-arm-clang of the example being built.
  – `$(APP_SIGNING_KEY)` can be found in devconfig.mak and the corresponding key based on the device being used can be selected.

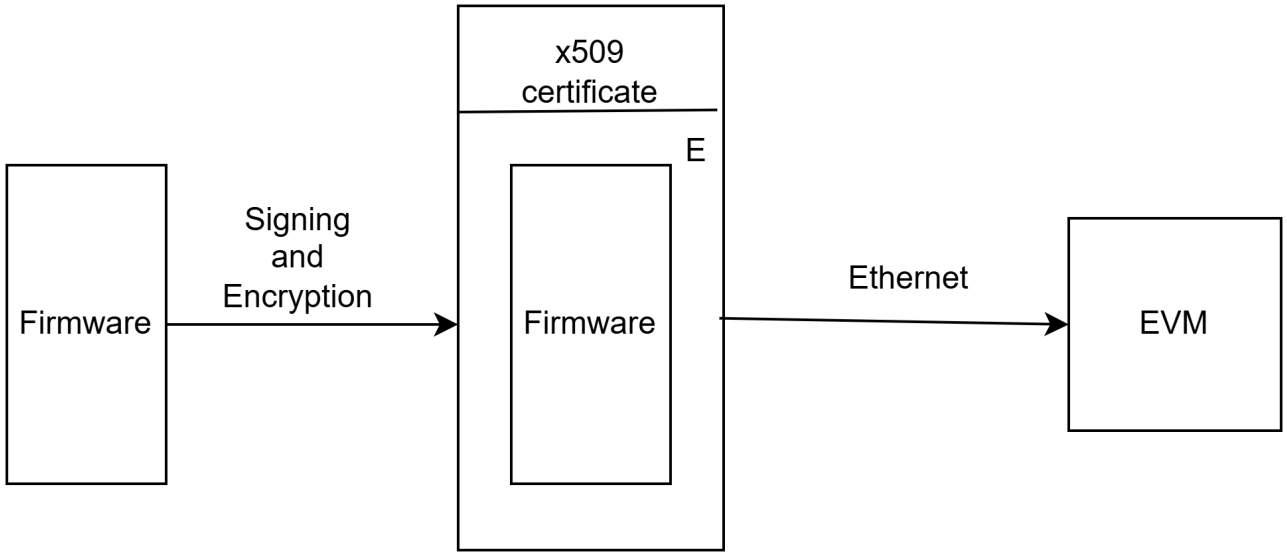
For AM275x devices:

- Change the directory to: `${FreeRTOS_SDK_PATH}/tools/boot/signing/appimage`
- Run the following command:

  ```
  python appimage_x509_cert_gen.py --bin </path/to/firmware> --authtype 1 --key $(APP_SIGNING_KEY) --output </path/to/signed_firmware> --keyversion 1.5
  ```

  – This command is found in the Makefile under ti-arm-clang of the example being built.
  – `$(APP_SIGNING_KEY)` can be found in devconfig.mak and the corresponding key based on the device being used can be selected.

5 Encrypted FOTA



For Encrypted FOTA, the firmware to be updated is post-processed to sign and encrypt the firmware for authenticity and confidentiality. Once the signed firmware is received on the EVM, it is authenticated before the actual firmware is programmed. If the authentication is successful, the signing certificate is stripped, and the firmware is programmed into the media.

The following steps are required to perform the above flow:

5.1 Signing and Encrypting the Firmware

To encrypt the application image to be transferred, run the following python script with the mentioned arguments:

For AM243x/AM64x devices:

- Change the directory to: `${MCU_PLUS_SDK_PATH}/source/security/security_common/tools/boot/signing`
- Run the following command:

  ```
  python appimage_x509_cert_gen.py --bin </path/to/firmware> --authtype 1 --key $(APP_SIGNING_KEY) --enc y --enckey $(APP_ENCRYPTION_KEY) --output </path/to/signed_firmware>
  ```

  – `$(APP_SIGNING_KEY)` and `$(APP_ENCRYPTION_KEY)` can be found in devconfig.mak and the corresponding key based on the device being used can be selected.

For AM275x devices:

- Change the directory to: `${FreeRTOS_SDK_PATH}/tools/boot/signing/appimage`
- Run the following command:

  ```
  python appimage_x509_cert_gen.py --bin </path/to/firmware> --authtype 1 --key $(APP_SIGNING_KEY) --enc y --enckey $(APP_ENCRYPTION_KEY) --output </path/to/signed_firmware> –keyversion 1.5
  ```

  – `$(APP_SIGNING_KEY)` and `$(APP_ENCRYPTION_KEY)` can be found in devconfig.mak and the corresponding key based on the device being used can be selected.

Note

For Authenticated and Encrypted ENET FOTA, signing the new application image with an x509 certificate is a must. This is the recommended practice as the authentication performed on the application side treats the new application image along with the extra certificate as a single blob. This blob is then authenticated, and the extra certificate is stripped out before performing Flash writes. If the new application image does not have the extra certificate, then the logic for removing the extra certificate should be removed in the enet_fota_app.c file. If this is not removed, then the original certificate would be stripped out and the subsequent boot will result in a failure.

6 Software Structure

The Beyond SDK Repository has the code implementation of ENET FOTA.

Note

The ENET FOTA and the SBL OSPI examples are compatible with MCU+SDK 12.00 and FreeRTOS 12.00.

6.1 AM243x/AM64x

- SBL OSPI for ENET FOTA can be found at:
  - Beyond-SDK/am243x/mcupsdk/enet_fota/examples/drivers/boot/sbl_ospi_enet_fota
- ENET FOTA example can be found at:
  - Beyond-SDK/am243x/mcupsdk/enet_fota/examples/enet_fota
- The ENET FOTA example folder also contains the patches folder with the patches that must be applied to the source folder in the MCU+ SDK repository.
  - Ensure that all the libraries are rebuilt after applying the patches and then build the SBL OSPI and ENET FOTA example.
- The python script required for transferring the new application image from the host to the EVM can be found at:
  - Beyond-SDK/am243x/mcupsdk/enet_fota/tools/boot/enet_fota_file_transfer.py
- The cfg file required for running the enet_fota_file_transfer.py python file is:
  - Beyond-SDK/am243x/mcupsdk/enet_fota/tools/boot/sbl_prebuilt/am243x-evm/enet_fota_app.cfg
- Add the location of the new application image in the enet_fota_app.cfg file and save it.
- Flash the respective SBL OSPI and the ENET FOTA application images.
- Run the following command after Powering ON the EVM:
  - python enet_fota_file_transfer.py --cfg=sbl_prebuilt/am243x-evm/enet_fota_app.cfg

6.2 AM275x

- SBL OSPI for ENET FOTA can be found at:
  - Beyond-SDK/am275x/mcupsdk/enet_fota/examples/drivers/boot/sbl_ospi_enet_fota
- SBL OSPI file for SBL OSPI can be found at:
  - Beyond-SDK/am275x/mcupsdk/enet_fota/examples/drivers/boot/common/soc/am275x/sbl_ospi_enet_fota.c
- ENET FOTA example can be found at:
  - Beyond-SDK/am275x/mcupsdk/enet_fota/examples/enet_fota
- The ENET FOTA example folder also contains the patches folder with the patches that msut be applied to the source folder in the MCU+ SDK repository.
  - Ensure that all the libraries are rebuilt after applying the patch and then build the SBL OSPI and ENET FOTA example.
- The python script required for transferring the new application image from the Host to the EVM can be found at:
  - Beyond-SDK/am275x/mcupsdk/enet_fota/tool/boot/enet_fota_file_transfer.py
- The cfg file required for running the enet_fota_file_transfer.py python file is:
  - Beyond-SDK/am275x/mcupsdk/enet_fota/tools/boot/sbl_prebuilt/am275x-evm/enet_fota_app.cfg
- Add the location of the new application image in the enet_fota_app.cfg file and save it.
- Flash the respective SBL OSPI and the ENET FOTA application images.
- Run the following command after Powering ON the EVM:
  - python enet_fota_file_transfer.py --cfg=sbl_prebuilt/am275x-evm/enet_fota_app.cfg

7 Expected Outputs

7.1 AM243x/AM64x

R5 logs:

```
Old app image

Listening for new app image

[ ENETFOTA ] Starting Ethernet Transfer ...
Enabling clocks!
Mdio_open:320
Open MAC port 1
EnetPhy_bindDriver:1942
Open MAC port 2
EnetPhy_bindDriver:1942
PHY 0 is alive
PHY 3 is alive
[ ENETFOTA ] EVM MAC address: 34:08:e1:80:a7:c1
[ ENETFOTA ] Please wait for Linkup ...
Cpsw_handleLinkUp:1423
Cpsw_handleLinkDown:1449
Cpsw_handleLinkUp:1423
[ ENETFOTA ] Linkup Done!
[ ENETFOTA ] Receiving file, please wait ...
[ ENETFOTA ] Status:0
[ ENETFOTA SUCCESS ] Ethernet Transfer Done.
[ ENETFOTA ] Packets Received   :  156
[ ENETFOTA ] Total File Size    :  227864 Bytes
Cpsw_handleLinkDown:1449
Disabling clocks for ENET: 5, inst:0!

Image Authentication Successful

Flashed New App Image

--- Time Profile for ENET FOTA ---
Time Taken for Ethernet Setup:               12.100539 s
Time Taken for Transferring New App Image:   0.116197 s
Time Taken for Authenticating New App Image: 0.001007 s
Time Taken for Flashing New App Image:       1.361955 s

SOC Reset

DMSC Firmware Version 12.0.2--v12.00.02 (Clever Cat)
DMSC Firmware revision 0xc
DMSC ABI revision 4.0

BootLoader1 loaded
KPI_DATA: [BOOTLOADER_PROFILE] CPU Clock        : 800.000 MHz
KPI_DATA: [BOOTLOADER_PROFILE] Boot Media       : NOR SPI FLASH
KPI_DATA: [BOOTLOADER_PROFILE] Boot Media Clock : 166.667 MHz
KPI_DATA: [BOOTLOADER_PROFILE] Boot Image Size  : 217 KB
```

```
SOC Reset

DMSC Firmware Version 12.0.2--v12.00.02 (Clever Cat)
DMSC Firmware revision 0xc
DMSC ABI revision 4.0

BootLoader1 loaded
KPI_DATA: [BOOTLOADER_PROFILE] CPU Clock        : 800.000 MHz
KPI_DATA: [BOOTLOADER_PROFILE] Boot Media       : NOR SPI FLASH
KPI_DATA: [BOOTLOADER_PROFILE] Boot Media Clock : 166.667 MHz
KPI_DATA: [BOOTLOADER_PROFILE] Boot Image Size  : 217 KB
KPI_DATA: [BOOTLOADER_PROFILE] Cores present    :
r5f0-0
KPI_DATA: [BOOTLOADER_PROFILE] SYSFW init                      :      10863us
KPI_DATA: [BOOTLOADER_PROFILE] System_init                     :      13899us
KPI_DATA: [BOOTLOADER_PROFILE] Drivers_open                    :       1664us
KPI_DATA: [BOOTLOADER_PROFILE] Board_driversOpen               :      27558us
KPI_DATA: [BOOTLOADER_PROFILE] Sciclient Get Version           :       9938us
KPI_DATA: [BOOTLOADER_PROFILE] CPU load                        :      39671us
KPI_DATA: [BOOTLOADER_PROFILE] SBL End                         :          3us
KPI_DATA: [BOOTLOADER_PROFILE] SBL Total Time Taken            :     103599us

Image loading done, switching to application ...

New app image

Listening for new app image

[ ENETFOTA ] Starting Ethernet Transfer ...
Enabling clocks!
Mdio_open:320
Open MAC port 1
EnetPhy_bindDriver:1942
Open MAC port 2
EnetPhy_bindDriver:1942
PHY 0 is alive
PHY 3 is alive
[ ENETFOTA ] EVM MAC address: 34:08:e1:80:a7:c1
[ ENETFOTA ] Please wait for Linkup ...
Cpsw_handleLinkUp:1423
[ ENETFOTA ] Linkup Done!
[ ENETFOTA ] Status:0
[ ENETFOTA SUCCESS ] Ethernet Transfer Done.
[ ENETFOTA ] Packets Received   :  0
[ ENETFOTA ] Total File Size    :  0 Bytes
Cpsw_handleLinkDown:1449
Disabling clocks for ENET: 5, inst:0!
```

7.2 AM275x

WKUP R5 logs:

```
BootLoader0 loaded
[BOOTLOADER_PROFILE] Boot Media       : FLASH
[BOOTLOADER_PROFILE] Boot Media Clock : 166.667 MHz
[BOOTLOADER_PROFILE] Boot Image Size  : 219 KB
[BOOTLOADER_PROFILE] Cores present    :
[BOOTLOADER_PROFILE] System_init                    :       2743us
[BOOTLOADER_PROFILE] TIFS init                      :        292us
[BOOTLOADER_PROFILE] Board_init                     :          7us
[BOOTLOADER_PROFILE] FreeRtosTask Create            :        255us
[BOOTLOADER_PROFILE] Drivers_open                   :       5785us
[BOOTLOADER_PROFILE] Board_driversOpen              :        159us
[BOOTLOADER_PROFILE] sciServer_init                 :      15080us
[BOOTLOADER_PROFILE] SBL Drivers_open               :       3205us
[BOOTLOADER_PROFILE] SBL Board_driversOpen          :       5843us
[BOOTLOADER_PROFILE] Sciclient Get Version          :       6604us
[BOOTLOADER_PROFILE] R5FSS0_0 Image Load            :       4508us
[BOOTLOADER_PROFILE] SBL Total Time Taken           :      44485us

Image loading done...
Starting RTOS/Baremetal applications
Sciserver Testapp Built On: Jun  1 2026 14:46:47
Sciserver Version: v2023.11.0.0REL.MCUSDK.MM.NN.PP.bb
RM_PM_HAL Version: vMM.NN.PP
Starting Sciserver..... PASSED

Starting OSPI Bootloader ...

SYSFW ABI: 4.0 (firmware rev 0x000c '12.0.2--v12.00.02 (Clever Cat)')
BootLoader1 loaded
[BOOTLOADER_PROFILE] Boot Media       : FLASH
[BOOTLOADER_PROFILE] Boot Media Clock : 166.667 MHz
[BOOTLOADER_PROFILE] Boot Image Size  : 219 KB
[BOOTLOADER_PROFILE] Cores present    :
[BOOTLOADER_PROFILE] System_init                    :       2556us
[BOOTLOADER_PROFILE] TIFS init                      :        292us
[BOOTLOADER_PROFILE] Board_init                     :          6us
[BOOTLOADER_PROFILE] FreeRtosTask Create            :        255us
[BOOTLOADER_PROFILE] Drivers_open                   :       5785us
[BOOTLOADER_PROFILE] Board_driversOpen              :        160us
[BOOTLOADER_PROFILE] sciServer_init                 :      15082us
[BOOTLOADER_PROFILE] SBL Drivers_open               :       3211us
[BOOTLOADER_PROFILE] SBL Board_driversOpen          :       5828us
[BOOTLOADER_PROFILE] Sciclient Get Version          :       6605us
[BOOTLOADER_PROFILE] R5FSS0_0 Image Load            :       4512us
[BOOTLOADER_PROFILE] SBL Total Time Taken           :      44298us

Image loading done...
Starting RTOS/Baremetal applications
```

R5 logs:

```
Old app image

Listening for new app image

[ ENETFOTA ] Starting Ethernet Transfer ...
Enabling clocks!
EnetAppUtils_reduceCoreMacAllocation: Reduced Mac Address Allocation for CoreId:1 From 4 To 1
Open MAC port 2
EnetPhy_bindDriver:1942
PHY 3 is alive
[ ENETFOTA ] EVM MAC address: 44:6b:1f:2c:2d:25
[ ENETFOTA ] Please wait for Linkup ...
Cpsw_handleLinkUp:1423
Cpsw_handleLinkDown:1449
Cpsw_handleLinkUp:1423
[ ENETFOTA ] Linkup Done!
[ ENETFOTA ] Receiving file, please wait ...
[ ENETFOTA ] Status:0
[ ENETFOTA SUCCESS ] Ethernet Transfer Done.
[ ENETFOTA ] Packets Received   :  158
[ ENETFOTA ] Total File Size    :  229976 Bytes
Cpsw_handleLinkDown:1449
Disabling clocks for ENET: 5, inst:0!

Image Authentication Successful

Flashed New App Image

--- Time Profile for ENET FOTA ---
Time Taken for Ethernet Setup:              12.028640 s
Time Taken for Transferring New App Image:   0.110134 s
Time Taken for Authenticating New App Image: 0.027497 s
Time Taken for Flashing New App Image:       1.407127 s

SOC Reset

New app image

Listening for new app image

[ ENETFOTA ] Starting Ethernet Transfer ...
Enabling clocks!
EnetAppUtils_reduceCoreMacAllocation: Reduced Mac Address Allocation for CoreId:1 From 4 To 1
Open MAC port 2
EnetPhy_bindDriver:1942
PHY 3 is alive
[ ENETFOTA ] EVM MAC address: 44:6b:1f:2c:2d:25
[ ENETFOTA ] Please wait for Linkup ...
```

8 Performance Metrics

Table 8-1. OSPI Configuration

Input Clock Frequency	166 MHz
PHY	Enabled
DMA	Enabled
Protocol	8D-8D-8D

Table 8-2. Time Profile

Time Taken for Ethernet Setup	Approximately 8-12 Seconds
Time taken for transferring new app image	Approximately 0.1 seconds
Time taken for authenticating new app image	Approximately 0.02 seconds
Time taken for writing new app image to Flash	Approximately 1.4 seconds

9 Summary

ENET FOTA provides a powerful, efficient, and secure method for keeping embedded devices up to date. By leveraging the speed and reliability of wired networking, combined with security practices such as authentication and encryption, organizations can maintain large device deployments without the cost and complexity of manual updates. This application note has detailed the mechanism and the steps to implement ENET FOTA on AM243x/AM64x and AM275x devices.

10 References

1. Texas Instruments, *AM243x MCU+ SDK*, webpage.
2. Texas Instruments, *AM275x FreeRTOS SDK*, webpage. .
3. Github, *Texas Instruments Beyond-SDK Repository*, webpage.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

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

Last updated 10/2025