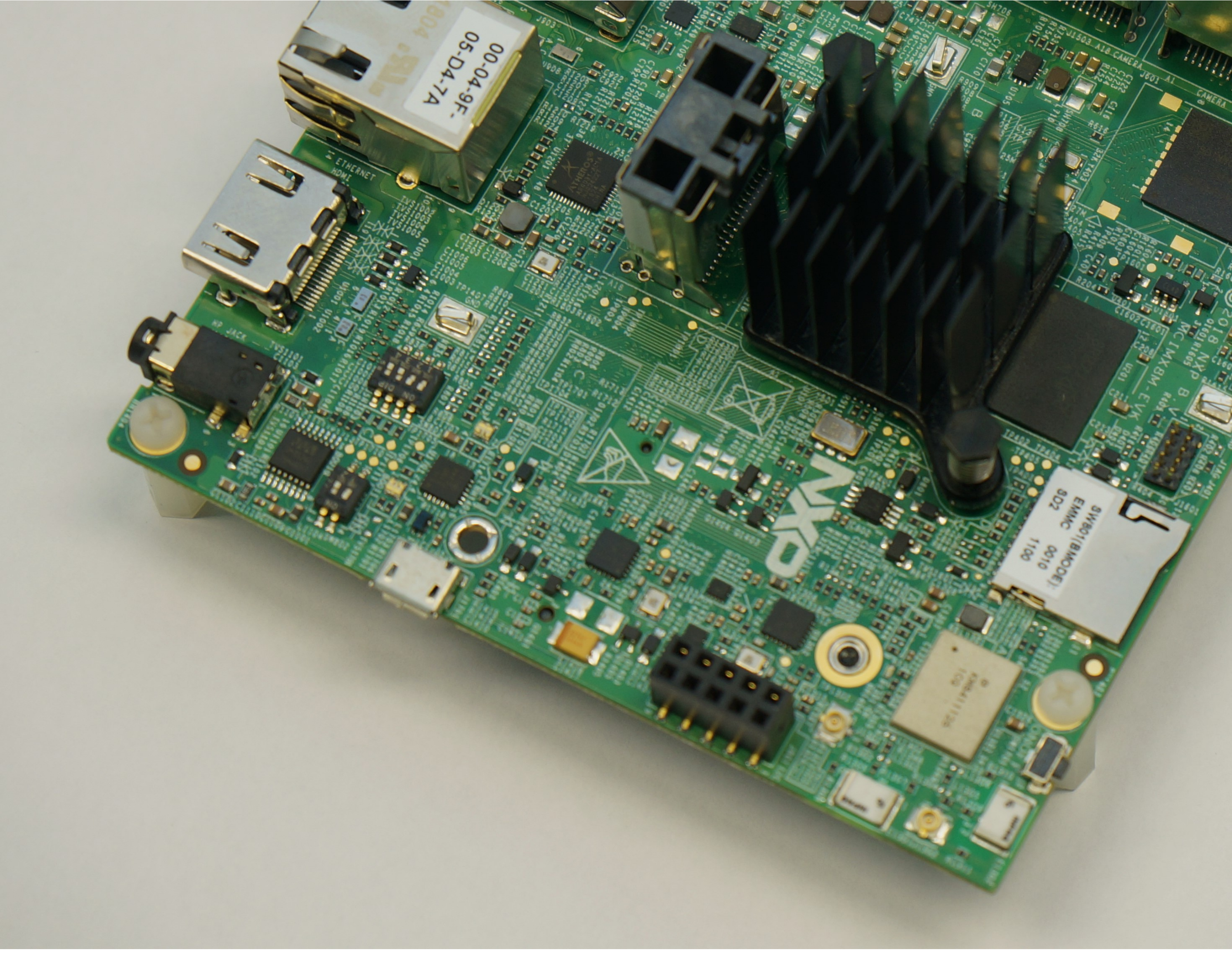


i.MX Windows 10 IoT User’s Guide

for NXP i.MX Platform

Document Number: IMXWGU

Rev. W1.1.0, 6/2022

Contents

1 Overview

User's guide describes the process of building and installing the Windows 10 IoT OS BSP (Board Support Package) for the i.MX platform. It also covers special i.MX features and how to use them.

Guide also lists the steps to run the i.MX platform, including board DIP switch settings, and instructions on the usage and configuration of U-Boot boot loader.

Features covered in this guide may be specific to particular boards or SOCs. For the capabilities of a particular board or SOC, see the *i.MX Windows 10 IoT Release Notes* (IMXWIN10RN).

1.1 Audience

This chapter is intended for software, hardware, and system engineers who are planning to use the product, and for anyone who wants to know more about the product.

1.2 Conventions

This chapter uses the following conventions:

- Courier New font: This font is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.

1.3 How to start

The i.MX Windows 10 IoT BSP is a collection of binary files, source code, and support files you can use to create a bootable Windows 10 IoT image for i.MX development systems.

Before you start, see the Feature List per Board chapter. This section lists all the i.MX boards covered by this BSP and also contains a list of possible features.

1.4 Starting From Prebuilt Binaries

The Prebuild Binary package contains prebuilt release-signed binaries of the drivers and firmware required for Windows 10 IoT Enterprise to run on the NXP i.MX EVK boards. It is the fastest way to get started running on physical hardware.

If you have downloaded the BSP with the Prebuilt Binaries, go to Deploy Windows IoT image to development board,
which will guide you through creating a Windows IoT image that includes the BSP binaries and deploying it to an i.MX EVK board

1.5 Starting From Source Files

The BSP Source Files package contains the source files of the drivers and firmware required for Windows 10 IoT Enterprise to run on NXP i.MX EVK boards. It is intended to be used as a reference for partners that have created their own hardware designs that use the i.MX 8 family of SoCs and need to customize the drivers and firmware for their own design.

If you have downloaded an archive with BSP sources, you will need to first build the Windows drivers and Boot firmware from source before you can create a Windows IoT image and deploy it to your device. Please start from Building Windows 10 IoT for NXP i.MX Processors, which will guide you through the process of the building the Windows drivers and boot firmware from source. Once you have successfully built the driver and firmware binaries, you can go back to the chapter that describes how to Deploy Windows IoT image to development board.

1.6 References

For more information about Windows 10 IoT Enterprise, see Microsoft online documentation.

- http://windowsondevices.com

The quick start guides contain basic information on the board and setting it up. They are available on NXP website.

- i.MX 8M Quad Evaluation Kit Quick Start Guide
- i.MX 8M Mini Evaluation Kit Quick Start Guide
- i.MX 8M Nano Evaluation Kit Quick Start Guide
- i.MX 8M Plus Evaluation Kit Quick Start Guide

Documentation is available online at nxp.com

- i.MX 8 information is at http://www.nxp.com/imx8

1.7 Feature List per Board

Table 1.1: Overview of the currently supported features for every board.

Feature	MCIMX8M-EVK	8MMINILPD4-EVK
BSP name	NXPEVK_IMX8M_4GB	NXPEVK_iMX8M_Mini_2GB
SD Card boot	Y	Y
eMMC boot	Y	Y
Audio	Y	Y
GPIO	Y	Y
I2C	Y	Y
Ethernet	Y	Y
PWM	Y	Y
SD Card	Y	Y
eMMC	Y	Y
SPI (master)	N/A	N/A
Display	Y	Y
UART	Y*	Y*
USB (host)	Y	Y
PCIe	Y	Y
TrEE	Y	Y
Processor PM	Y	Y
Device PM	N**	N**
LP standby	N**	N**
Display PM	Y	Y
DMA	Y	Y
VPU	Y	Y

Legend	Meaning
Y	Enabled
Y*	To enable the UART, the kernel debugger must be disabled by running the following command on the device and rebooting. The UART exposed is the same UART that the kernel debugger uses. `bcdedit /debug off`
N/A	Feature not applicable
N*	Feature not enabled - feature is not available in default board configuration
N**	Feature not enabled - feature is not currently supported
PM	Power management
LP	Low power

Not all features of a given subsystem maybe fully enabled and/or optimized. If you encounter issues with supported features, please open an issue.

2 Deploy Windows IoT image to development board

This chapter will guide you through the deployment process of Windows operating system and boot firmware to the NXP development boards. At a high level, you will go through the following steps:

1. Obtain the BSP drivers + firmware, either by using the prebuilt package, or by building them from source
2. Use the serial downloader to flash the firmware to the board
3. Create a Windows IoT image that merges the BSP drivers with the Windows IoT Operating System
4. Prepare a micro SD card that contains a WinPE installer and the Windows IoT image
5. Boot the board with the prepared micro SD card inserted and wait for Windows IoT to install

Assuming you already have the BSP drivers + firmware ready, this process should take 1-2 hours.

2.1 Hardware prerequisites

- NXP i.MX development board (additional information can be found in Supported boards table in i.MX_Win10_Release_Notes.pdf)
- USB-C cable to connect from device to PC
- microSD card (minimum 8GB)

2.2 Software prerequisites

- Binary drivers and firmware (either downloaded from NXP.com or built locally)
- Windows IoT operating system. There are two options:
  - **Recommended**: Windows 10 IoT Enterprise LTSC 2021 for Arm64
    - *Preview available on Microsoft Collaborate https://partner.microsoft.com/en-us/dashboard/collaborate/packages/12392. Please contact Microsoft at WinIoT-on-NXP@microsoft.com for access*
  - Windows 10 IoT Enterprise 21H2 GAC for Arm64
    - *Please contact a Windows IoT distributor for access*
- Windows ADK 1809 https://docs.microsoft.com/en-us/windows-hardware/get-started/adk-install

  - *Windows ADK 1809 is required for building firmware. If you are only creating a Windows IoT image, you can install Windows ADK for Windows 10, version 2004.*

- Windows PE add-on for the ADK, version 2004. https://docs.microsoft.com/en-us/windows-hardware/get-started/adk-install

  - *Regardless of whether you installed ADK version 1809 or 2004, you should install version 2004 of WinPE.*

2.3 Serial Logging Setup

To help troubleshoot issues during boot, you can use the USB micro-B port on the i.MX EVK boards to output U-Boot and UEFI firmware serial debug logs to a host PC. The USB micro-B port on the EVK presents a virtual serial port to the host PC, which can be read by common Windows serial terminal applications such as HyperTerminal, Tera Term, or PuTTY.

1. Connect the target and the PC using a cable mentioned above.
2. Open Device Manager on the PC and locate the Enhanced Virtual serial device and note the COM port number.
3. Open terminal on the PC. Configure the Enhanced Virtual serial/COM port to 921600 baud/s, 8 bit, one stop bit.

In order for the Host PC to recognize the i.MX device's virtual serial port, you might need to download and install drivers for the i.MX's USB to serial converter.

The i.MX 8M EVK uses the CP210x USB to serial bridge. The CP210x driver can be found here: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers.

The i.MX 8M Mini EVK uses the FT2232D USB to serial bridge. The FT2232D driver can be found under http://www.ftdichip.com/Drivers/VCP.htm.

Note that the order that the serial/COM ports appear in Windows can differ between boards and between Windows hosts.

2.4 Basic Board Setup

NXP i.MX EVK has three different boot modes that determine how the board boots. When the boot mode is set to SD card boot, the board assumes the firmware bootloader is on the SD card. When it's set to eMMC boot, the board assumes the firmware bootloader is on the eMMC. Finally, when the boot mode is set to USB UUU Download, the device goes into serial download mode and can have the eMMC flashed by a Host PC using the uuu.exe serial downloader.

Table 2.1: MCIMX8M-EVK boot DIP cfg

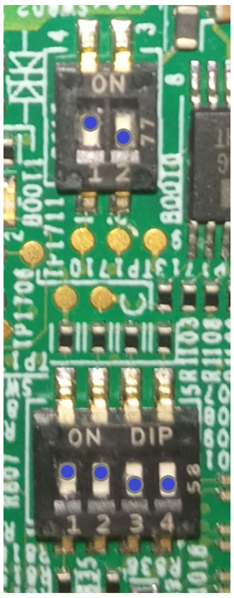
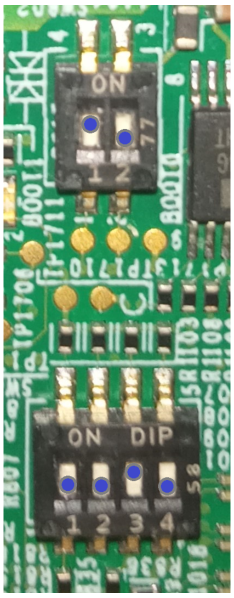
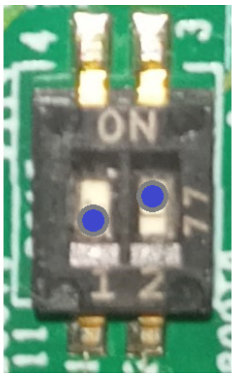
SD card boot	eMMC boot	USB UUU Download
		

Table 2.2: 8MMINILPD4-EVK boot DIP cfg

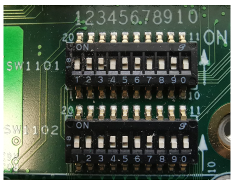
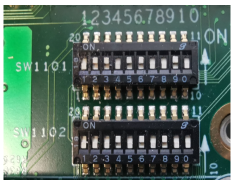
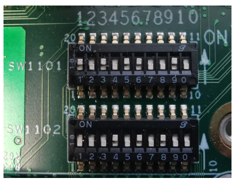
SD card boot	eMMC boot	USB UUU Download
		

Table 2.3: 8MNANOD4-EVK boot DIP cfg

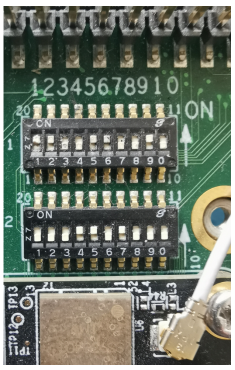
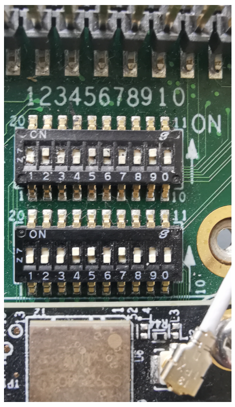
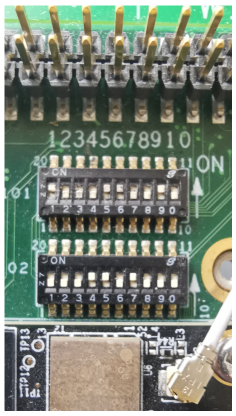
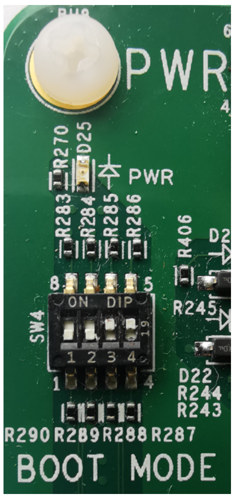
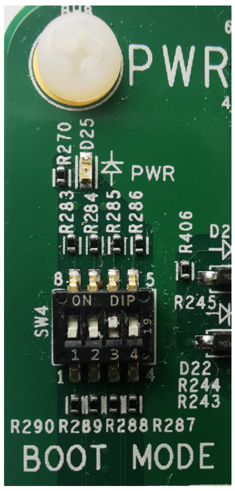
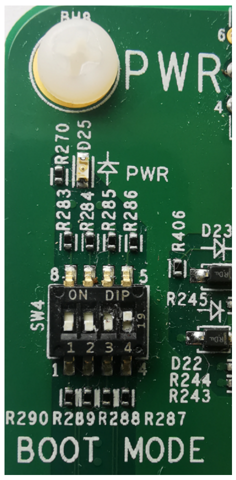
SD card boot	eMMC boot	USB UUU Download
		

Table 2.4: 8MPLUSLPD4-EVK boot DIP cfg

SD card boot	eMMC boot	USB UUU Download
		

2.5 Deploy boot firmware

Prebuilt boot firmware (`firmware.bin`) is included in both binary and source BSP packages. The boot firmware must be deployed to the eMMC on the board before deploying the Windows IoT image, otherwise the bootloader might not check the SD card for the Windows IoT image installer.

2.5.1 Firmware on eMMC

To flash the boot firmware to the eMMC we recommend to download UUU tool (Universal Update Utility) 1.3.191 which was used for BSP testing from https://github.com/NXPmicro/mfgtools/releases/tag/uuu_1.3.191.

Note: The UUU tool requires U-boot with UUU download support enabled. For security reasons, its recommended to disable UUU download support in the U-Boot in final designs

To deploy boot firmware to eMMC via USB download mode follow steps below:

1. Navigate to `firmware` directory. In the source package it is located in `imx-windows-iot/BSP/firmware`.
2. Download UUU tool and put it into `firmware\tools\uuu\uuu.exe`.
3. Begin preparing board by powering it off.
4. Switch the target board to USB download mode using DIP switches.
5. Connect a USB-C cable between your desktop and the board’s USB-C port.
6. Run `flash_bootloader.cmd /device NXPEVK_iMX8M_4GB` inside `firmware` directory.
7. When the process finishes power the board off. Disconnect the USB-C cable connecting board to PC. Not disconnecting from PC may cause errors in later steps.
8. Switch the board to eMMC boot mode. See Basic Board Setup

UUU documentation can be found at https://github.com/NXPmicro/mfgtools.

2.5.2 Firmware on SD card

During active development of the boot firmware, it can be time consuming and error-prone to repeatedly change the dip switches between UUU download mode and eMMC boot mode. To simplify this, the i.MX EVK boards support a SD Card boot mode that allows you to keep the boot firmware on the SD card. However, because the Windows IoT installer uses the SD card, if you need to install a new Windows IoT image, you must go back to flashing the boot firmware to eMMC using UUU download mode and setting the boot mode to boot from eMMC.

To deploy boot firmware to SD card from Windows we recommend to use Cfimager tool from https://www.nxp.com/webapp/Download?colCode=CF_IMAGER.

1. Navigate to `firmware` directory. In the source package it is located in `imx-windows-iot/BSP/firmware`.
2. Download NXP cfimager tool and put it into `firmware/cfimager.exe` or system `%PATH%`.
3. Plug SD card into the computer and run `flash_bootloader.cmd /device NXPEVK_iMX8M_4GB \/target_drive <SD card letter eg. f:>` inside `firmware` directory. The `/device` could be `NXPEVK_iMX8M_4GB`, `NXPEVK_iMX8M_Mini_2GB`, `EVK_iMX8MN_2GB` or `EVK_iMX8MP_6GB`.

4. Begin preparing board by powering it off.
5. Switch the target board to boot form SD card using DIP switches. See Basic Board Setup

2.6 Install Windows IOT Enterprise to eMMC

Currently, the only way to deploy a Windows IoT Enterprise to the onboard eMMC is to use the WinPE OS (Windows Preinstallation Environment) to write the Windows IoT image to eMMC. WinPE is a small Windows manufacturing OS that can be fully loaded and run from memory without using persistent storage. We will create an SD card with WinPE and a Windows IoT image that contains the BSP drivers. The boot firmware will check the SD card and boot WinPE, which will then install the Windows IoT image to the eMMC.

We will list commands for an MCIMX8M-EVK board. The tools and techniques can be adapted to other hardware designs.

2.6.1 Preparing WinPE based installation media on a microSD card

2.6.1.1 Prepare Windows IoT image with prebuilt drivers

1. Download W21H2-X.X.X-imx-windows-bsp-binaries.zip and extract it. If you are building the drivers yourself, follow the steps in the section titled Prepare Windows IoT image with drivers from sources yourself.
2. Open elevated command prompt and navigate to extracted `<imx-windows-bsp-binaries>\IoTEntOnNXP`.
3. Continue to Common steps

2.6.1.2 Prepare Windows IoT image with drivers from sources yourself

1. Build the drivers yourself as described in Building Windows 10 IoT for NXP i.MX Processors. If you are using the pre-built binaries, follow the steps in the section titled Prepare Windows IoT image with prebuilt drivers.
2. After the BSP has been built open elevated command prompt and navigate to `<BSP_DIR>\imx-windows-iot\BSP\IoTEntOnNXP`.
3. Continue to Common steps

2.6.1.3 Common steps

1. Download the Windows IoT Enterprise version ISO mentioned in Software prerequisites.
2. After the ISO file have been downloaded, mount the ISO file and copy out the install.wim from `<DVD mount drive:>\sources\install.wim` to the `IoTEntOnNXP` directory.
3. In the `IoTEntOnNXP` directory run `make-winpe-enterprise.cmd` in elevated prompt. This will create copy of selected install.wim image with injected i.MX drivers.

   Note: Be sure to copy whole command line. Commands with many parameters may be split into multiple lines in the following examples.

   For release signed drivers (e.g. NXP precompiled drivers) you can use:

   ```
   make-winpe-enterprise.cmd /device NXPEVK_iMX8M_4GB /disable_updates /disable_transparency /image ..
       \install.wim
   ```

   During development, you will want to add a few additional development flags to allow test-signed drivers and to enable windbg debugging, as in the below:

   ```
   make-winpe-enterprise.cmd /device NXPEVK_iMX8M_4GB /disable_updates /disable_transparency
       /patch_sdport /test_signing /winpedebug /windebug /image ..\install.wim
   ```

   Notes to parameters:

   - The `/image <WIM_PATH>` could be either absolute or relative to `IoTEntOnNXP\work`. Directory `work` might not exist yet. The script will create it and then enter it.
   - Use `/disable_updates` to prevent Windows from automatically searching for and installing updates, which can slow down a device and reduce available storage.
   - Use `/disable_transparency` to improve UI responsiveness. This is improved in the KB5011543 Windows update.
   - Use `/patch_sdport` to update Sdport.sys. This option should to enabled during development if you are creating an installer for the `19044.1288.211006` Windows OS build, which has a bug in sdport.sys that can cause the blue bug check screen. The patched sdport.sys in this BSP is test signed and will cause blue screen if testsigning is disabled. It should NOT be used in production designs. See limitations in release notes for more information.
   - Use `/winpedebug` to enable Windows PE debug through the serial port.
   - Use `/windebug` to enable Windows IoT debug through the serial port.
   - Use `/test_signing` to enable test signing.
   - The `/device` could be `NXPEVK_iMX8M_4GB`, `NXPEVK_iMX8M_Mini_2GB`, `EVK_iMX8MN_2GB` or `EVK_iMX8MP_6GB`.

4. Now you can deploy the WinPE image to an SD card by `make-winpe-enterprise.cmd /apply \<disk_number>`.

```
make-winpe-enterprise.cmd /device NXPEVK_iMX8M_4GB /apply <disk_number>
```

The `disk_number` of your SD card can be obtained:

- As ‘Disk ###’ using Diskpart.

  ```
  diskpart
  Microsoft DiskPart version 10.0.19041.964

  Copyright (C) Microsoft Corporation.
  On computer: NXL70483

  DISKPART> list disk

    Disk ###  Status         Size     Free     Dyn  Gpt
    --------  -------------  -------  -------  ---  ---
    Disk 0    Online          953 GB   195 GB        *
    Disk 1    Online         7580 MB      0 B        *

  DISKPART> exit

  Leaving DiskPart...
  ```

- From the DeviceID string using wmic.

  ```
  wmic diskdrive list brief
  Caption                  DeviceID             Model                    Partitions  Size
  SDHC Card \\               .\PHYSICALDRIVE1  SDHC Card                  2           7945620480

  # The format is \\.\PHYSICALDRIVE<disk_number>.
  ```

5. WinPE based Windows installer is now deployed on the SD card.

6. After you deploy boot firmware (see Deploy boot firmware) you can insert the prepared SD card installer into your development board and power it on. Board will boot into the installer. Installer will flash Windows IoT Enterprise to the eMMC, then reboot into installed Windows. Installing Windows IoT Enterprise from the SD card can take around 30 minutes, so get some coffee while you wait!

   Note: The WinPE installer renames the `EFI` folder at the root of SD card to `_efi`, which causes UEFI to skip the SD card at boot time. This allows you to keep the SD card inserted across reboots without having Windows IoT reinstalled each reboot. If you wish to boot into the WinPE installer again, you can rename `_efi` back to `EFI`.

7. Make sure to disable sleep in “Power & sleep” settings after the Windows os boots up to avoid unexpected system hangs.

3 Building Windows 10 IoT for NXP i.MX Processors

3.1 Obtaining BSP sources

3.1.1 Required Tools

The following tools are required to follow this chapter:

- git
- git-lfs
- software to unpack zip, gzip and tar archives

3.1.2 The NXP i.MX Windows IoT BSP source package

The Windows IoT BSP for NXP i.MX Processors consists of multiple parts, namely:

1) The NXP i.MX BSP sources package which is available at www.NXP.com. Package contains sources for both the boot firmware and Windows drivers.
2) The iMX firmware and NXP Code Signing Tool (CST) available at www.NXP.com.
3) Open-source parts of boot firmware and tools that are not distributed as part NXP BSP package.
4) The Windows IOT Enterprise operating system provided by Microsoft.

To prepare sources for building BSP follow these steps:

> Note Only steps 1-3 are required for building Windows drivers from source. All steps are required for building firmware from source.

1) Download the NXP i.MX BSP provided as `W<os_version>-imx-windows-bsp-<build_date>.zip` archive from www.NXP.com.

2) Create an empty directory, further referred as `<BSP_DIR>`, and extract the downloaded archive there. Path to this directory should be short as possible containing only letters and underscores, braces and other special characters could cause build errors.

   Shell command `unzip W<os_version>-imx-windows-bsp-<build_date>.zip -d win10-iot-bsp` command can be used to extract the package. Command will create `win10-iot-bsp` directory containing `.git` repository with the BSP release.

3) Populate the directory by running `Init.bat`, to build drivers on Windows host, `Init.sh`, to build firmware on Linux host. Both scripts will checkout sources from the repository by `git \reset --hard`. The `Init.sh` shall also checkout submodules that are required to build i.MX boot firmware by `git submodule update --init --recursive`. During prerelease testing the `Init.sh` executed inside Ubuntu environment run into "server certificate verification failed. CAfile: /etc/ssl/certs/ca-certificates.crt CRLfile: none" error. Problem could be solved by installing `apt-transport-https ca-certificates` and certificate update.

```
sudo apt update ; sudo apt-get install apt-transport-https ca-certificates -y ; sudo
    \update-ca-certificates
```

4) At this point it is possible to build the Windows drivers. Further steps are required only to build i.MX boot firmware (ATF, U-boot and UEFI) from sources.

5) Download and extract the Code Signing Tools (CST) from NXP's website. You will need to create an account on NXP's website to access this tool. Extract the tool inside bsp repository, and rename the newly created folder to cst to get `<BSP_DIR>/cst` folder:

```
tar xf cst-3.1.0.tgz
mv release cst
rm cst-3.1.0.tgz
```

6) Download and extract the iMX firmware from NXP's website and place it to firmware-imx. Extract the tool inside bsp repository, and rename the newly created folder to `firmware-imx` to get `<BSP_DIR>/firmware-imx/firmware/ddr/` in directory tree:

```
chmod +x firmware-imx-8.14.bin
./firmware-imx-8.14.bin
mv firmware-imx-8.14 firmware-imx
rm firmware-imx-8.14.bin
```

7) At this point your directory structure should contain the following folders.

```
- <BSP_DIR>
    |- cst (manually downloaded)
    |- firmware-imx (manually downloaded)
    |- Documentation
    |- MSRSec
    |- RIoT
    |- imx-atf
    |- imx-mkimage
    |- imx-optee-os
    |- imx-windows-iot
    |- mu_platform_nxp
    |- patches
    |- uboot-imx
```

3.2 Building the drivers in the BSP

Before you start building the BSP, you need to have an archive with latest BSP sources from NXP sites downloaded and extracted as described in Obtaining BSP sources chapter.

3.2.1 Required Tools

The following tools are required to build the drivers:

- Visual Studio 2017
- Windows Kits (ADK/SDK/WDK)

3.2.1.1 Visual Studio 2017

- Make sure that you **install Visual Studio 2017 before the WDK** so that the WDK can install a required plugin.
- Download Visual Studio 2017.
- During install select **Desktop development with C++**.
- During install select the following in the Individual components tab. If these options are not available try updating VS2017 to the latest release:
  - **VC++ 2017 version 15.9 v14.16 Libs for Spectre (ARM)**
  - **VC++ 2017 version 15.9 v14.16 Libs for Spectre (ARM64)**
  - **VC++ 2017 version 15.9 v14.16 Libs for Spectre (X86 and x64)**
  - **Visual C++ compilers and libraries for ARM**
  - **Visual C++ compilers and libraries for ARM64**

3.2.1.2 Windows Kits from Windows 10, version 1809

- **IMPORTANT: Make sure that any previous versions of the ADK and WDK have been uninstalled!**
- Install ADK 1809
- Install WDK 1809
  - Scroll down and select Windows 10, version 1809.

  - Make sure that you allow the Visual Studio Extension to install after the WDK install completes.

- If the WDK installer says it could not find the correct SDK version, install SDK 1809
  - Scroll down and select Windows 10 SDK, version 1809 (10.0.17763.0).
- After installing all Windows Kits, restart computer and check if you have correct versions installed in Control panel.

3.2.2 Structure of Windows driver sources

The *imx-windows-iot* - sources of Windows drivers has the following structure:

BSP - Contains boot firmware, driver binaries (Generated at build time.) and scripts needed to deploy BSP to development board.

build - Contains build scripts, and the VS2017 solution file.

components - Contains third party binaries and utility projects.

driver - Contains driver sources.

hals - Contains hal extension sources.

3.2.3 One-Time Environment Setup

Test certificates must be installed to generate driver packages on a development machine.

1. Open an Administrator Command Prompt.
2. Navigate to your BSP and into the folder `imx-windows-iot\build\tools`.
3. Launch `StartBuildEnv.bat`.
4. Run `SetupCertificate.bat` to install the test certificates.

Some tools may not work correctly if LongPath is not enabled, therefore run following command in console:

1. Execute `reg add HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem /v LongPathsEnabled /t REG_DWORD /d 1` command.

3.2.4 Building the drivers

1. Open the solution imx-windows-iot\build\solution\iMXPlatform\iMXPlatform.sln located in the path where you have extracted BSP archive.

Note Microsoft pkggen requires launching Visual Studio 2017 as Administrator. Although driver projects are configured not to generate packages BSP might contain miscellaneous projects that require launching Visual Studio as Administrator to build successfully.

2. Choose `Debug` or `Release` build type.
3. If secure boot feature is enabled it is required to use signed drivers.
4. To build press Ctrl-Shift-B or choose Build -> Build Solution from menu. This will compile all driver packages then `imx-windows-iot\BSP\IoTEntOnNXP\drivers` for deployment.
5. The updated drivers could now be injected into the installation image or manually installed to running development board.
   - To manually install drivers it has to be copied to development board via USB drive, network share, scp, remote desktop. Driver can be installed either by clicking `install` in right click menu of 'inf' file or by devcon command-line utility.
   - For debug use the `.kdfiles` of WinDBG. Then to initiate driver reload either use devcon or reset the board.
   - To create new installation SD card follow Deploy Windows IoT image to development board

3.3 Building ARM64 Firmware

This chapter describes the process of setting up a build-environment to build the latest firmware and update the firmware on development board.

3.3.1 Setting up your build environment

1. Start Linux environment such as:
   - Dedicated Linux system
   - Linux Virtual Machine
   - Windows Subsystem for Linux (WSL setup instructions)

     Note: W-imx-windows-bsp-.zip was validated with Ubuntu 20.04 in WSL and also standalone Ubuntu.
2. Obtain and prepare the BSP sources by following all steps described in Obtaining BSP sources. Use `Init.sh` not `Init.bat` to populate the repository and all submodules.

3. Install or update build tools. The shell commands bellow can be used to do this process on Ubuntu 20.04 or 18.04.

   1. Update package manager.

      ```
      sudo apt-get update
      sudo apt-get upgrade
      ```

   2. If using Ubuntu 18.04 and possibly other older distributions the mono package might be outdated. Causing build to fail. For such distributions add mono repository to the system as described in https://www.mono-project.com/download/stable/#download-lin before installing the mono package.

      The process valid for Ubuntu 18.04 in December 2021:

      ```
      sudo apt install gnupg ca-certificates
      sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
          \3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
      # Optionally key could be downloaded to a file and added manually by 'apt-key add <keyfile>'.
      # Now that certificate is installed we can add official mono repository to repository list.
      echo "deb https://download.mono-project.com/repo/ubuntu stable-bionic main" | sudo tee /etc/apt
          /sources.list.d/mono-official-stable.list
      sudo apt update
      ```

   3. Install required software. Note the mu_project currently requires python 3.8 and higher.

      ```
      sudo apt-get install attr build-essential python3.8 python3.8-dev python3.8-venv
          \device-tree-compiler bison flex swig iasl uuid-dev wget git bc libssl-dev zlib1g-dev
          \python3-pip mono-devel gawk
      ```

   4. Download the Arm64 cross-compiler.

      ```
      pushd ~
      wget https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/aarch64-linux-gnu
          /gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
      tar xf gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
      rm gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz
      # The cross compiler prefix is required to be exported later into AARCH64_TOOLCHAIN_PATH variable.
      # export AARCH64_TOOLCHAIN_PATH=~/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
          /aarch64-linux-gnu-
      popd
      ```

   5. Change directory to the BSP_DIR. Following commands reference the files inside the BSP directory. That BSP_DIR contains extracted W-imx-windows-bsp-.zip.

      ```
      cd <BSP_DIR>
      ```

   6. Project MU strongly suggests use of Python Virtual Environment for each workspace - in this case BSP revision. Separated environments allow workspaces to keep specific Pip module versions without modifying global system state when firmware is compiled.

Note The virtual environment doesn't use system packages. Thus `sudo` must not be used when installing packages using pip.

```
python3.8 -m venv <path to new environment>
source <path to new environment>/bin/activate
eg.:
python3.8 -m venv ~/venv/win_fw_build
source ~/venv/win_fw_build/bin/activate
```

Please note the path to the new environment you chosen. Environment has to be activated before building the firmware.

7. Install required python packages.

   1) Install or update mu_platform Python dependencies using pip.

      ```
      pushd mu_platform_nxp
      pip3 install -r requirements.txt --upgrade
      ```

   2) Install pycryptodome package (successor of pycrypto).

      ```
      pip3 install pycryptodome
      ```

   3) Install pyelftools package.

      ```
      pip3 install pyelftools
      ```

8. Setup Mu platform. (Optional because buildme64.sh does these steps automatically.)

   1) Setup and update submodules.

      ```
      python3 NXP/MCIMX8M_EVK_4GB/PlatformBuild.py --setup
      # or NXP/MCIMX8M_MINI_EVK_2GB/PlatformBuild.py
      ```

      In case you return here facing problems during UEFI build, use `--force` to clean the environment. Make sure to commit or stash all your changes first, `--force` argument performs git reset –hard.

   2) Initialize and update Mu platform dependencies. If this command fails try upgrading mono. The best way to do this is to uninstall mono and reinstall if from its official repository. Process is described at https://www.mono-project.com/download/stable/#download-lin.

      ```
      python3 NXP/MCIMX8M_EVK_4GB/PlatformBuild.py --update
      # or NXP/MCIMX8M_MINI_EVK_2GB/PlatformBuild.py
      ```

9. Return to BSP root.

   ```
   popd
   ```

3.3.2 Building the firmware

This chapter assumes the BSP sources have been prepared as described in Obtaining BSP sources chapter.

To build the boot firmware:

1. Open cmd prompt at BSP_DIR.

   ```
   cd <BSP_DIR>
   ```

2. Activate your python virtual environment created in . (Use the path specified when creating the environment.)

   ```
   source ~/venv/win_fw_build/bin/activate
   ```

3. Export AARCH64_TOOLCHAIN_PATH cross compiler prefix. In this guide toolchain has been placed inside home (~/) drectory.

   ```
   export AARCH64_TOOLCHAIN_PATH=~/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/bin
       /aarch64-linux-gnu-
   ```

4. Optionally, in case of major update, you may need to step into mu_platform_nxp and run `python3 NXP/MCIMX8M_EVK_4GB/PlatformBuild.py` with `--setup --force` and then `--update` manually. To get clean and up-to-date MU build environment. Make sure to stash or commit your changes. Command performs `git reset --hard`.

5. Build the firmware and create firmware.bin. To build the boot firmware execute the `buildme64.sh -b <BOARD_NAME> -t all [-clean] -fw` script provided in BSP_DIR (the root of extracted BSP sources).

   The `buildme64.sh` script bundled in BSP also copies `flash.bin` and `uefi.fit` into `<BSP_DIR>/imx-windows-iot/components/Arm64BootFirmware/<board_name>`. This allows to rebuild only UEFI or U-boot.

   ```
   ./buildme64.sh -b NXPEVK_iMX8M_4GB -t all -c -fw
   ```

   - Use `-b NXPEVK_iMX8M_4GB` or `-b 8M` to select i.MX 8M EVK
   - Use `-b NXPEVK_iMX8M_Mini_2GB` or `-b 8Mm` to select i.MX 8M Mini EVK
   - Use `-t all -t sign_images` to build signed_firmware.bin

6. To deploy `firmware.bin` to the i.MX development board follow the process described in Deploy boot firmware.

3.3.3 Common causes of build errors:

1. ImportError: No module named Crypto.PublicKey.
   - This error is encountered when pycryptodome module is missing. Or in case obsolete pycrypto needs to be removed.
2. Unable to enter directory. Directory doesn't exist.
   - We have run into this problem in case gitmodules were not downloaded completely (eg. MSRSec is empty) or `cst` or `firmware` directories are missing. Try repeating the Obtaining BSP sources step by step.
3. The build fails in WSL while the BSP is located somewhere in `/mnt/c` of the WSL.
   - Try `setfattr -n system.wsl_case_sensitive -v 1 <BSP_DIR>`. OP-Tee also requires symbolic links. We have been able to build boot firmware in `/mnt/c/` on Windows OS version 1909. Workaround is to copy the BSP to WSL filesystem eg. to HOME.
4. RuntimeError: SDE is not current. Please update your env before running this tool.
   - Try to update mono_devel package and repeat setup and update steps for PlatformBuild.py in Setting up your build environment.
5. cp: cannot stat '../../firmware-imx/firmware/ddr/synopsys/lpddr4_pmu_train_*.bin'
   - Please make sure the firmware-imx obtained in Obtaining BSP sources chapter is placed inside the BSP in way the following directories exist `<BSP_DIR>/firmware-imx/firmware/ddr/synopsys`.
6. Can't find elftools module. Probably it is not installed on your system. You can install this module with apt install python3-pyelftools.
   - This error means the `pyelftools` module is not installed inside of the python environment. To install this package into environment use `pip3 install pyelftools` without `sudo`.

4 Signing Firmware

This document offers a high level overview of the various signing keys to construct the High Assurance Boot chain from Boot ROM all the way to Windows IoT Core.

4.1 High Assurance Boot

High Assurance Boot is an NXP security feature to ensure that the Boot ROM will only load code that has been signed with the correct private key. This is accomplished by creating a public/private key-pair with NXP's Code Signing Tool (CST), burning the Super Root Key Hash of the public key into SoC fuses, then signing the SPL binary with the manufacturer's private key. The private keys used for HAB **MUST BE** kept secure and secret because they are the root of trust for your device and any firmware signed with these private keys will be allowed to run.

Testing keys for development have already been generated and are available in the test_keys_no_security folder (../build/firmware/test_keys_no_security). By default all SPLs built in imx-windows-iot are signed with these keys.

- These keys are for development only and **MUST NOT** be used for production!
- To enable High Assurance Boot the Super Root Key Hash of these public keys must be burned into the SoC. This is **permanent** and no other first stage firmware except those signed with this key will be allowed to run.

The test keys folder is selected using the KEY_ROOT define in imx8.mk (../build/firmware/imx8.mk) and the specific keys in the folder are chosen in u-boot_sign.csf-template. The makefile copies u-boot_sign.csf-template into u-boot_sign.csf then fills out some build specific information. Then u-boot_sign.csf is used with CST to sign the SPL binary for High Assurance Boot.

Documentation on generating new keys can be found inside the download for the NXP Code Signing Tool: https://www.nxp.com/webapp/sps/download/license.jsp?colCode=IMX_CST_TOOL

4.2 SPL/U-Boot Flattened Image Tree Loading

SPL is used to load the Flattened Image Tree (FIT) containing OP-TEE and U-Boot Proper. U-Boot proper is used to load the FIT containing UEFI.

The FIT is generated using the mkimage tool and the Image Tree Source (.its) files in the firmware/its folder. The per-board firmware makefiles choose which .its files to use using the UBOOT_OPTEE_ITS and UEFI_ITS variables.

Each .its file contains a list of images to load and a configuration block. SPL and U-Boot use the configuration block to verify the hashes of the stored images when loading them. There are two versions of each .its file, *unsigned.its and* ecc_signed.its. The unsigned.its files don't include a signature in the configuration block, the *ecc_signed.its include an Elliptic-curve Cryptography based signature in the configuration block. The key used by each .its file is configurable by changing the key-name-hint to a different filename.

The FIT test key included in the repository was generated using the following OpenSSL command:

```
openssl ecparam -genkey -name prime256v1 -out ecc_prime256v1_dev_pem.key
```

When imx8.mk uses mkimage on an .its file with a signature in the configuration block it writes out the public key into the Device Tree Blob (dtb) of the firmware that is expected to load the FIT.

As an example this is what happens when uefi.fit is built:

1. The makefile is building uefi.fit with uefi_ecc_signed.its so it signs the FIT with the private key and injects the public key into U-Boot's dt.dtb file.
2. When U-Boot attempts to load any FIT, in this case uefi.fit, it will verify that the FIT has a configuration block.
3. Once U-Boot has verified that uefi.fit has a configuration block it checks to make sure that the configuration block was signed with the private key corresponding to the public key that was built into its own Device Tree Blob.

If the *unsigned.its files are used no public keys are injected into the U-Boot or SPL DTBs. This causes them to not check the presence or signature of the configuration block in the loaded FITs, breaking the High Assurance Boot chain.

4.3 UEFI Secure Boot Keys

Secure Boot is enabled during UEFI by the presence of several variables in the Authenticated Variable store.

Further Reading:

- Microsoft documentation for Secure Boot
- Microsoft documentation for enabling Secure Boot and BitLocker on IoT Core

4.4 OP-TEE TA Signing Key

OP-TEE uses a signing key to verify that all TAs it loads are authorized to run. This key is defined by TA_SIGN_KEY. The default key is optee_os/keys/default_ta.pem.

A different key can be selected by setting the environment variable TA_SIGN_KEY to a different pem file before building OP-TEE. This will require a rebuild of all TAs expected to run on the new build of OP-TEE.

OP-TEE runs a script to extract the public key from the TA_SIGN_KEY pem into optee_os\core\ta_pub_key.c. It then uses the public key information in the shdr_verify_signature function to verify that TAs are correctly signed while loading them.

4.5 Building signed binaries

By default, the buildme64 script does not build signed binaries. In order to create signed_firmware.bin, buildme64.sh must use the -t signed_images:

```
./buildme64.sh -b 8M -t all -fw -t signed_images
```

4.6 Enabling Secure Boot

The default configuration of U-Boot and UEFI do not have Secure Boot enabled by default. U-Boot requires that the config file for the EVK board in the uboot-imx/configs directory be updated adding the following config options:

```
CONFIG_IMX_HAB=y
CONFIG_FIT_SIGNATURE=y
```

and optionally

```
CONFIG_FIT_VERBOSE=y
```

Further reading:

- High Assurance Boot (HAB) – i.MX8M edition

To enable Secure Boot in UEFI, the UEFI must be built using the SECURE profile. Use parameter -t secured_uefi to get signed_firmware.bin with secured UEFI:

```
./buildme64.sh -b 8M -t all -fw -t secured_uefi -t signed_images
```

5 Revision History

Table 5.1: Revision history

Revision number	Date	Substantive changes
W0.9.0	1/2022	Private preview release for i.MX8M platform.
W0.9.1	3/2022	Public preview release for i.MX8M platform.
W1.0.0	4/2022	Public release for i.MX8M and i.MX8M Mini platforms.
W1.1.0	6/2022	Public release for i.MX8M Nano and i.MX8M Plus platforms.