UG10155

Debian Linux SDK User Guide

Rev. LDLSDK_26.06 — 26 June 2026 **User guide**

Document information

Information	Content
Keywords	UG10155, Debian, Linux, SDK, Flexbuild
Abstract	The Debian Linux SDK Distribution is a Debian-based Linux enablement software for NXP series processors that are based on Arm cores to provide quick evaluation for customers.

1 Overview

Debian is a free Operating System (OS), also known as Debian GNU/Linux. It provides a wide range of application software, and comes with a total of over 118,000 packages, precompiled software bundled up in a nice format for easy installation for various machines or embedded devices.

The Debian Linux SDK distribution composes of NXP-specific custom components and open source software developed by the community-supported Debian Project. It aims to provide an easy-to-use and convenient development solution for users' quick evaluation with widely available deb packages on the ARM64 platforms of NXP.

The Debian Linux SDK distribution uses Flexbuild (a flexible and easy-to-use build system developed by NXP) to generate the BSP composite firmware (including ATF, U-Boot, OP-TEE, kernel, DTB, peripheral firmware, initramfs) and Debian images. It compiles NXP-specific hardware-accelerated components for various hardware blocks and peripherals (such as GPU, NPU, VPU, ISP, SEC, Wi-Fi/Bluetooth, and Audio) based on Debian runtime dependencies.

Users can use Flexbuild to easily build Debian-based RootFS, Linux kernel, BSP components, and miscellaneous Userspace applications for various use cases (like graphics, multimedia, networking, connectivity, security, and AI/ML) to streamline the system build with flexible customization and efficient CI/CD. Flexbuild Git repository is available at GitHub.

Users can also use the `flex-installer` tool to easily install various Distros to the target storage device (SD/eMMC card or USB/SATA/NVMe disk).

1.1 Features

1.1.1 Supported features on i.MX 8 series and i.MX 95 series

- Weston Desktop supported, Gnome Desktop supported alternatively.
- HDMI monitor display
- DSI MIPI Touchscreen display
- Desktop GUI with GPU acceleration
- Multimedia video playback with VPU codec
- MIPI CSI Camera OS08A20 with ISP (only on i.MX 8M Plus EVK)
- MIPI CSI Camera OV5640
- Web browsers (Chromium)
- Qt6 application support
- Wi-Fi + Bluetooth
- eIQ TensorFlow Lite support
- Gstreamer support
- DPDK for networking acceleration
- GoPoint Demos support in both GUI and TUI mode

1.1.2 Supported features on i.MX 93 series

- Debian 13 Weston Desktop
- HDMI monitor display
- LVDS Touchscreen display (only on i.MX 93 EVK)
- CSI MIPI Camera AP1302 with ISP
- Wi-Fi + Bluetooth
- eIQ TensorFlow Lite support

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Gstreamer support
- DPDK for networking acceleration
- GoPoint Demo support in both GUI and TUI mode

1.1.3 Supported features on i.MX 91 series

- Debian 13 Weston Desktop
- Wi-Fi + Bluetooth

1.1.4 Supported feature matrix on Layerscape platforms

The following table shows the features that are supported in the Layerscape Debian Linux SDK v25.12 release.

Note: *Refer to the legend below to decipher the entries:*

- ***Y****: Feature is supported by the software.*
- */: Feature is not supported by the software.*
- ***NA****: Hardware feature is not available.*

Table 1. Key features matrix on Layerscape platforms

Feature	LS1028A	LS1043A	LS1046A	LX2160A
64-bit User space, LE	Y	Y	Y	Y
40b phys mem	Y	Y	Y	Y
Data Plane Development Kit (DPDK)	Y	Y	Y	Y
Hugetlbfs	Y	Y	Y	Y
Management Complex	NA	NA	NA	Y
Open Portable Trust Execution Environment (OP-TEE)	Y	Y	Y	Y
SEC engine	Y	Y	Y	Y
Time Sensitive Network (TSN)	Y	NA	NA	NA
USDPAA Applications	NA	/	/	NA
Trusted Firmware-A (TF-A)	Y	Y	Y	Y
GUI Desktop	Y	N	N	N

1.2 Supported machines

The currently supported machines are listed below. For the latest list of supported machines, see the *Debian Software Release Notes* (document RN00266).

- i.MX 8M Mini EVK
- i.MX 8M Plus EVK
- i.MX 8M Plus FRDM
- i.MX 8QuadMax MEK
- i.MX 91 EVK
- i.MX 91 FRDM
- i.MX 91S FRDM
- i.MX 93 EVK
- i.MX 93 FRDM

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- i.MX 95 15x15 EVK
- i.MX 95 15x15 FRDM
- i.MX 95 19x19 EVK
- i.MX 95 19x19 FRDM-pro
- LS1028ARDB
- LS1043ARDB
- LS1046ARDB
- LX2160ARDB

2 Quick Start with Debian

For first-time users, it is recommended to use the `deploy-wizard.sh` tool to simplify the setup process.

To deploy Debian onto the target board, perform the following steps:

1. Download the tool.
   Download the deployment wizard script:

   ```
   $ wget https://www.nxp.com/lgfiles/sdk/<version>/deploy-wizard.sh
   ```

   Example:

   ```
   $ wget https://www.nxp.com/lgfiles/sdk/lsdk2606/deploy-wizard.sh
   ```

2. Grant the execute permission.
   Make the script executable:

   ```
   $ chmod a+x deploy-wizard.sh
   ```

3. Launch the deployment wizard.
   Run the script to start the interactive deployment process:

   ```
   $ ./deploy-wizard.sh
   ```

4. Follow the interactive prompts.
   The wizard then guides you through the deployment process with a series of prompts:
   - Select the distro version and variant.
     Choose the distribution image you want to install.
   - Select the target machine.
     Pick the appropriate target platform from the supported list.
   - Select the target storage device.
     Choose the destination device from the detected storage devices.
   - Confirm the deployment.
     Review your selections and confirm to start the installation.

   After confirmation, the tool automatically invokes `flex-installer` and deploy the selected image to the target device.
   Note: *Double-check the selected storage device to avoid overwriting unintended disks.*
5. Boot the board.
   a. Remove the SD card from the host machine and insert it into the target board.
   b. Configure the board's boot switches to boot from the SD card.
   c. Power on the board.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

d. After the system boots up, run the `debian-post-install-pkg` script on the target board to upgrade it to a full-featured rootfs.

3 Host Setup

To achieve the expected behavior of the Flexbuild Project on a Linux host machine, install the packages and utilities described below. An important consideration is the hard disk space required on the host machine. For example, when building on a machine running Ubuntu, the minimum required hard disk space is about 30 GB. It is recommended to provide at least 50 GB, which is sufficient to compile all images together.

The recommended minimum Ubuntu version is 22.04 or later.

3.1 Docker

To meet compilation consistency requirements, Flexbuild projects must be compiled within Docker. The Flexbuild project provides the `make docker` command to build or attach to a Docker image. All compilation tools and required packages are installed and prepared during the initial Docker build. The host machine must provide the Docker tool.

Follow the instructions below to set up a host build machine using Docker.

1. Install Docker.
   There are various methods of installing Docker, for example, by docker script:

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

2. Run Docker without `sudo`.
   To work better with Docker without `sudo`, add your user to the Docker group:

```
$ sudo usermod -aG docker <your_user>
```

   Log out and log back in so that your group membership is re-evaluated.
3. Create a Docker configuration file at `~/.docker/config.json` and enter the following:

```
{
"proxies":
    {
     "default":
         {
          "httpProxy":"http://proxy.example.com:80"
         }
    }
}
```

   Note: *Replace the `example` proxy with your proxy information.*
4. Create the Docker service.

```
$ sudo mkdir -p /etc/systemd/system/docker.service.d
$ sudo vim /etc/systemd/system/docker.service.d/http-proxy.conf
```

   Add the following:

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:80/"
Environment="NO_PROXY=localhost,someservices.somecompany.com"
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Restart the Docker:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

3.2 Host packages

In addition to Docker, the host requires `make`, `git`, and `python3`. Run the following command to install them:

```
$ sudo apt install -y git python3 make qemu-user-static binfmt-support docker-
buildx
$ sudo update-binfmts --enable qemu-aarch64
```

4 Image Build

4.1 Source code structure

The Flexbuild project source code is organized in a well-structured directory hierarchy with clear separation of components and build artifacts.

The following is the Flexbuild project source code directory structure with comments.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
flexbuild/
├── .config                          // Main configuration file (generated)
├── Makefile                         // Main build file with all build targets
├── Kconfig                          // Kconfig configuration definitions
│
├── configs/                         // Configuration files
│   ├── board/                       // Board-level configurations
│   │   ├── lx2160ardb.conf          // Layerscape LX2160A RDB board config
│   │   ├── imx8mpevk.conf           // i.MX8M Plus EVK board config
│   │   ├── LS_boot.txt              // Boot script for Layerscape
│   │   ├── IMX_boot.txt             // Boot script for i.MX
│   │   └── ...                      // Other board configs
│   ├── sdk.yml                      // SDK components configuration (YAML)
│   └── .sdk.cfg                     // Parsed SDK configuration (generated)
│
├── tools/                           // Build tools and scripts
│   ├── flex-builder                 // Main shell build script
│   ├── flex-installer               // System installation tool
│   ├── deploy-wizard.sh             // An interactive wizard for deployment
│   ├── mkfw-wic.py                  // Create bootable firmware WIC images
│   ├── kconfig_hooks.py             // Kconfig post-processing hooks
│   ├── gen_defconfig.py             // Generate defconfig from board name
│   ├── gen_config_from_file.py      // Generate .config from config file
│   ├── distro_debian                // Generate initial rootfs
│   └── parse_yaml.py                // YAML configuration parser
│
├── include/                         // Makefile include files
│   ├── utils.mk                     // Utility functions and macros
│   ├── download_repo.mk             // Git/wget download rules
│   ├── patch_apply.mk               // Patch application rules
│   └── func.mk                      // Shell functions in makefile form
│
├── src/                             // Source build rules
│   ├── Makefile                     // Component-specific build rules
│   ├── bsp/                         // BSP component build rules
│   ├── apps/                        // Application component build rules
│   ├── linux/                       // Linux kernel build rules
│   └── system/                      // System script and configuration
│
├── build_<version>/                 // Build output root (generated)
│   ├── apps/                        // Applications build output
│   │   └── apps_debian_<machine>/   // Machine-specific apps installation
│   ├── bsp/                         // BSP build output
│   │   ├── atf/                     // ARM Trusted Firmware output
│   │   ├── uboot/                   // U-Boot bootloader output
│   │   └── rcw/                     // Reset Configuration Word output
│   ├── linux/                       // Linux kernel build output
│   │   └── linux/arm64/             // Kernel images
│   │       ├── LS/                  // Layerscape platform output
│   │       └── IMX/                 // i.MX platform output
│   ├── rfs/                         // Root filesystem
│   │   └── rootfs_<ver>_debian_<mach>// Complete Debian rootfs
│   ├── images/                      // Final packaged images
│   │   ├── rootfs_*.tar.zst         // Compressed rootfs image
│   │   ├── boot_*_<ver>.tar.zst     // Boot partition image
│   │   ├── rootfs_*.wic.zst         // WIC format disk image
│   │   ├── firmware_*.img           // Composite FW (Layerscape)
│   │   └── <machine>_*-flash.bin    // Boot firmware (i.MX)
│   └── boot/                        // Boot script directory
│
├── components_<version>/            // Component source code (generated)
│   ├── linux/                       // Linux kernel source
│   │   └── linux/                   // Kernel source repository
│   ├── bsp/                         // BSP component source
│   │   ├── atf/                     // ARM Trusted Firmware source
│   │   ├── uboot/                   // U-Boot source
│   │   ├── rcw/                     // RCW source (Layerscape)
│   │   └── imx-mkimage/             // i.MX image creation tool
│   └── apps/                        // Application source code
│       ├── multimedia/              // Multimedia components
│       ├── graphics/                // Graphics components
│       ├── networking/              // Networking components
│       ├── security/                // Security components
│       ├── ml/                      // Machine Learning components
│       ├── gopoint/                 // GoPoint components
│       └── utils/                   // Utility components
│
├── patch/                           // Patch files
├── logs/                            // Build logs (generated)
└── dl/                              // Download cache (generated)
```

Figure 1. Source code structure

4.2 Three output images

Flexbuild compiles and generates three distinct images required by system deployment.

4.2.1 BSP firmware image

The BSP firmware image serves as the boot image for the target platform.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- **Layerscape platform:**
  On Layerscape platforms, the BSP firmware image includes U-Boot, Arm Trusted Firmware (ATF), Reset Configuration Word (RCW), and various firmware components. The following table lists the detailed layout and size of each component.

Table 2. Unified 64 MB layout of Layerscape BSP composite image on SD/NOR/QSPI/XSPI media

Firmware definition		Max. size	Flash Offset	SD Start Block#
RCW + PBI + BL2 (`bl2.pbl`)		1 MB	0x00000000	0x00008
ATF FIP Image (`fip.bin`) BL31 + BL32 + BL33		4 MB	0x00100000	0x00800
Bootloader environment		1 MB	0x00500000	0x02800
Secure boot headers		2 MB	0x00600000	0x03000
DDR PHY FW or reserved		512 kB	0x00800000	0x04000
Fuse provisioning header		512 kB	0x00880000	0x04400
DPAA1 FMAN ucode		256 kB	0x00900000	0x04800
QE firmware or DP firmware		256 kB	0x00940000	0x04A00
Ethernet PHY firmware		256 kB	0x00980000	0x04C00
Script for flashing image		256 kB	0x009C0000	0x04E00
DPAA2-MC or PFE firmware		3 MB	0x00A00000	0x05000
DPAA2 DPL		1 MB	0x00D00000	0x06800
DPAA2 DPC		1 MB	0x00E00000	0x07000
Device tree(needed by uefi)		1 MB	0x00F00000	0x07800
Kernel + DTB	`lsdk2606_poky_tiny_LS_arm64.itb`	16 MB	0x01000000	0x08000
Initramfs		30 MB	0x02000000	0x10000
CA or other uses		2 MB	0x03e00000	0x1F000

Note: *Some platforms may include only a subset of the components listed in the table.*
This image can be generated using the following command:

```
$ make bsp
```

Image naming convention:

```
firmware_<machine>_<boot_mode>_[sec].img
```

Examples:

– For the LS1028ARDB platform with non-secure SD boot:

```
firmware_ls1028ardb_sdboot.img
```

– For the LS1043ARDB platform with secure NOR boot:

```
firmware_ls1043ardb_norboot_sec.img
```

- **i.MX Platform:**
  On i.MX platforms, the BSP firmware image includes ATF, U-Boot, and various firmware components, such as secure firmware, HDMI firmware, and DDR firmware.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

This image can be generated using the following command:

```
$ make flash.bin
```

Image naming convention:

```
<machine>-<boot_mode>-flash.bin
```

Examples:
For the i.MX 8M Plus EVK SD boot:

```
imx8mpevk-sd-flash.bin
```

4.2.2 Boot partition image

The boot partition image is an archive file containing all necessary boot partition contents.

This image includes the kernel image, Device Tree Blob (DTB), and the boot script file `boot.scr`. For secure boot on Layerscape platforms, the boot partition image also includes cryptographically signed files.

The boot partition image can be generated using the following command:

```
$ make boot
```

Image naming convention:

```
boot_<SOCFAMILY>_arm64_lts_<kernel_ver>.tar.zst
```

Examples:

- For i.MX platforms:

```
boot_IMX_arm64_lts_6.12.49.tar.zst
```

- For Layerscape platforms:

```
boot_LS_arm64_lts_6.12.49.tar.zst
```

4.2.3 Root file system partition image

The rootfs partition image is an archive file containing all user space applications, libraries, and data.

Image naming convention:

```
rootfs_<version>_debian_<machine>.tar.zst
```

Examples:

For i.MX 95 15x15 EVK platforms:

```
rootfs_lsdk2606_debian_imx95-15x15-evk.tar.zst
```

4.3 Overall build pocess

Only three steps are required to generate all three images.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

1. Create or attach to the Docker container.

```
$ git clone https://github.com/nxp/flexbuild
$ cd flexbuild
$ make docker
```

2. Create the `.config` file (choose one of the following options).
   - Select the target machine and applications (if needed) manually.

```
$ make menuconfig
```

   - Use the default configuration.

```
$ make <machine>_defconfig
```

   - Use an existing configuration file.

```
$ make config CONFIG=<config_file>
```

   Note: *Do not edit the `.config` file, as it may cause compilation errors.*
3. Build all required images (BSP + boot + rootfs).

```
$ make all
```

   The following sections provide detailed descriptions of each step.

4.3.1 Creating the Docker environment

All builds must be performed inside the Docker container. Therefore, the first step in the build process is to create or attach to a Docker container using the `make docker` command.

The execution flow of the `make docker` command is illustrated in the diagram below.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

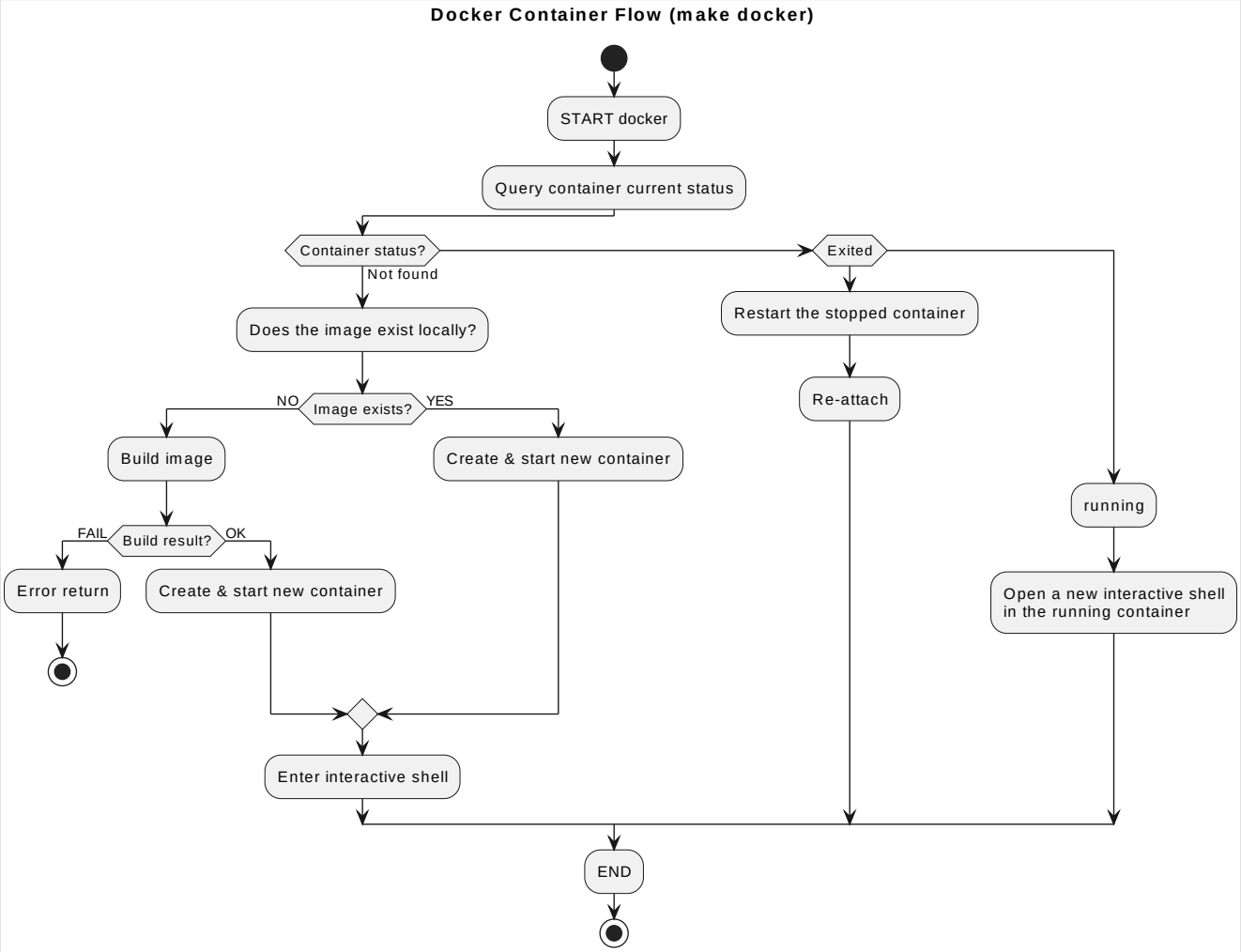

Figure 2. Flowchart of the `make docker` process

After this command completes, you will be inside the Docker container as the root user, ready to execute subsequent build commands. The command prompt will appear as:

```
[root@fbdebian flexbuild] $
```

4.3.2 Creating the configuration file

You can use any one of the following three methods to create the `.config` file.

4.3.2.1 Creating a `.config` file manually

1. Run `$ make menuconfig`.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

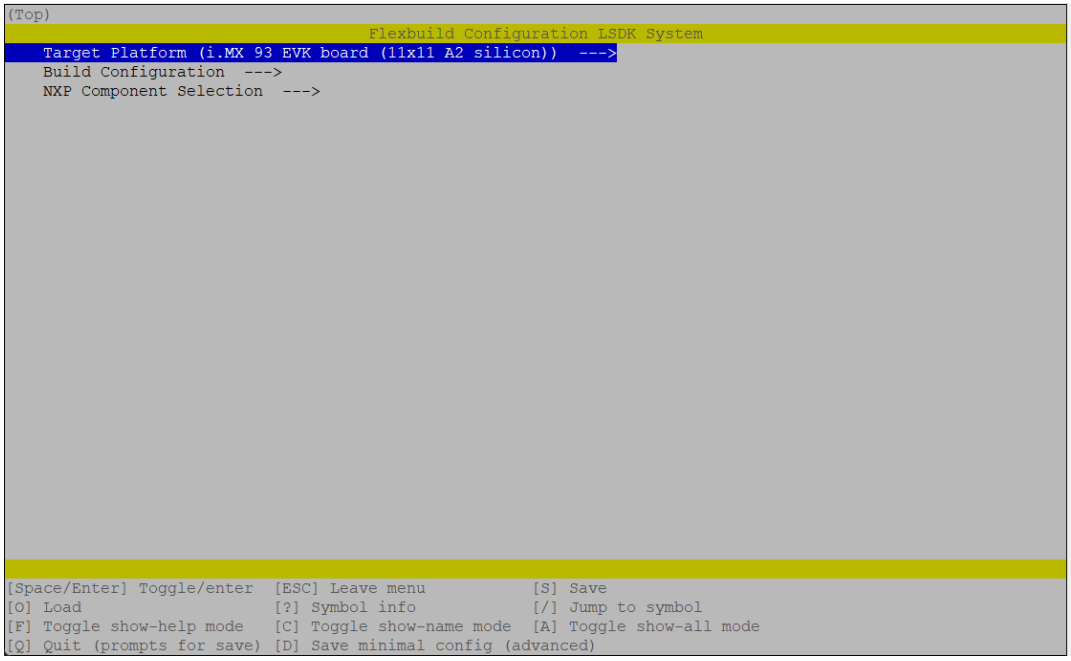

Figure 3. Flexbuild Main menu

2. Navigate to the **Target Platform** menu and select the platform you want to build.

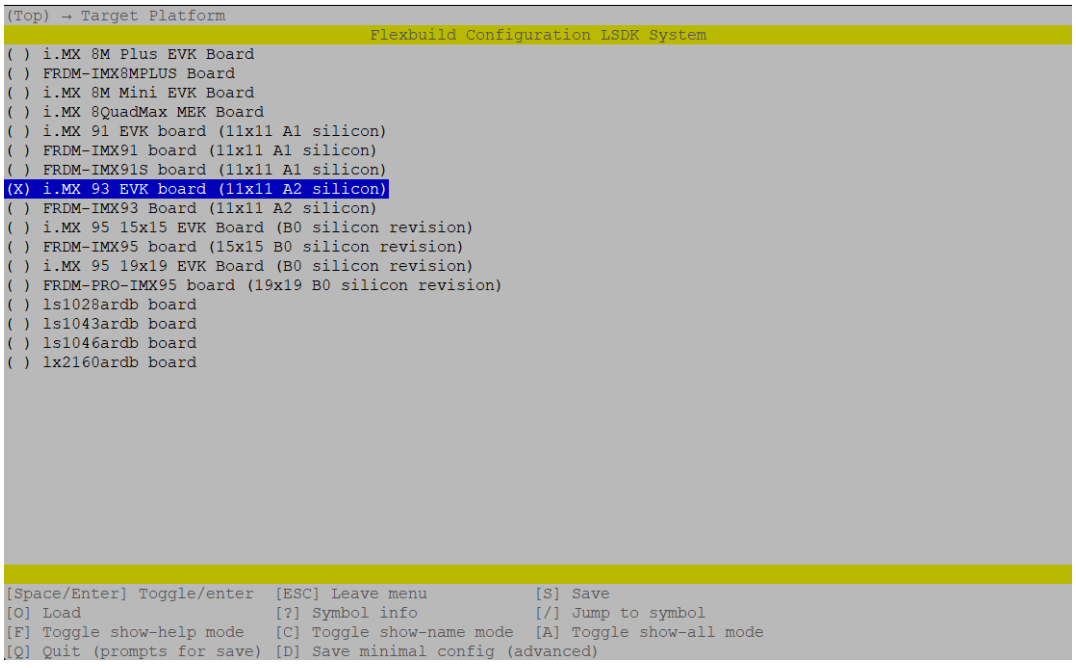

Figure 4. Flexbuild Target Platform menu

3. Navigate to the **Build Configuration** menu to modify build options.
   Use the arrow keys to navigate to the desired option and press the **F** key to view the detailed help for that option.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

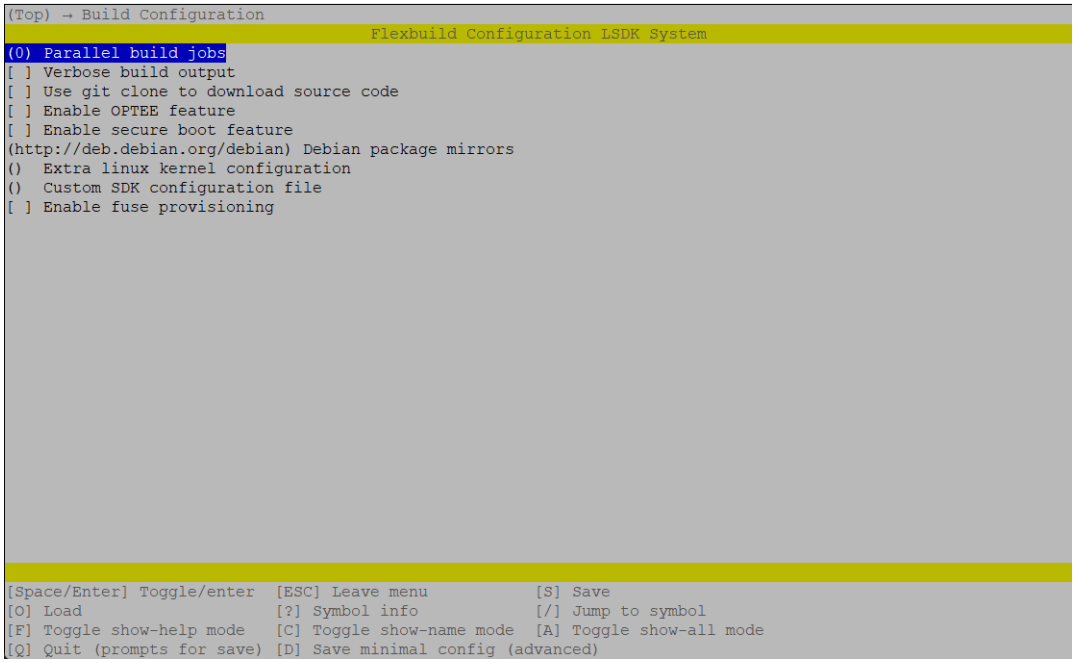

Figure 5. Flexbuild Build Configuration menu

4. Navigate to the **NXP Component Selection** menu to select or deselect components.

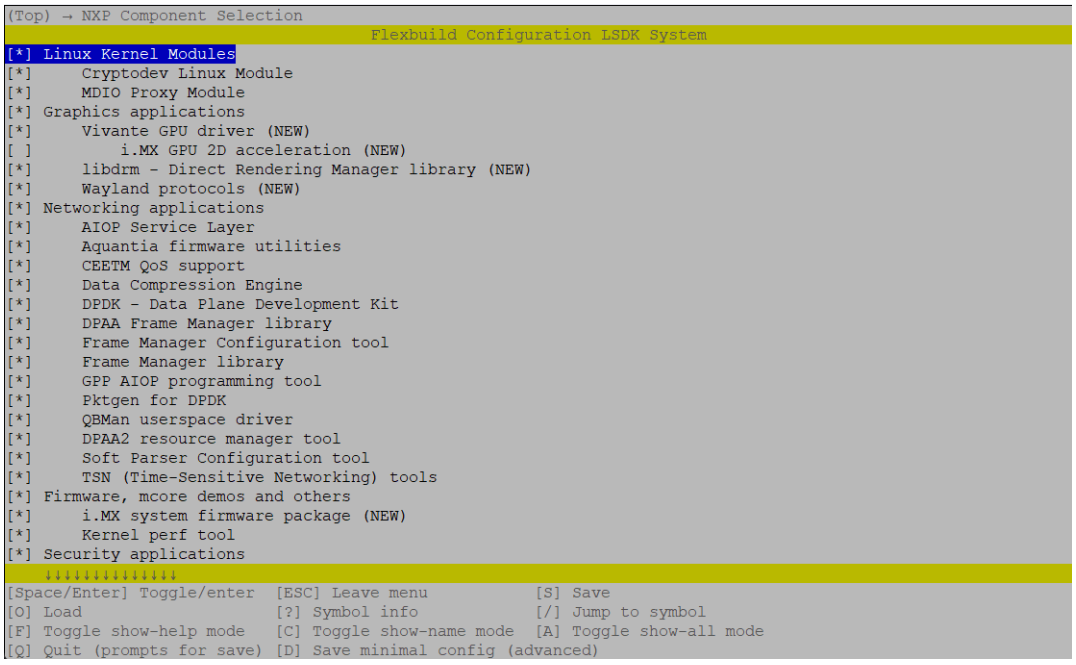

Figure 6. Flexbuild NXP Component Selection menu

Note:

- *The menu may vary depending on the distribution version.*
- *It is recommended to use the default values unless you have specific requirements.*

After saving and exiting the menu, the `.config` file is automatically generated with your configuration saved.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

4.3.2.2 Using the default values to create a `.config` file

To generate the configuration file using the default settings, use the following command:

```
$ make <machine>_defconfig
```

Example:

To generate the default configuration file for i.MX 93 EVK:

```
$ make imx93evk_defconfig
```

4.3.2.3 Creating a `.config` file based on an existing file

Use the following command to create a `.config` file based on an existing configuration file:

```
$ make config CONFIG=<config_file>
```

Example:

```
$ echo "CONFIG_SOC_IMX8MPEVK=y" > myconfig
$ echo "CONFIG_KEEP_GIT=y" >> myconfig
$ make config CONFIG=myconfig
```

Note: *An existing configuration file must specify a machine configuration (for example,* `CONFIG_SOC_IMX8MPEVK=y`*). Otherwise,* `CONFIG_SOC_IMX93FRDM=y` *is used by default.*

4.3.3 Building the three images

After the `.config` file is generated, build all the three required images in one step using the following command:

```
$ make all
```

The execution flow of this command is illustrated in the diagram below.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

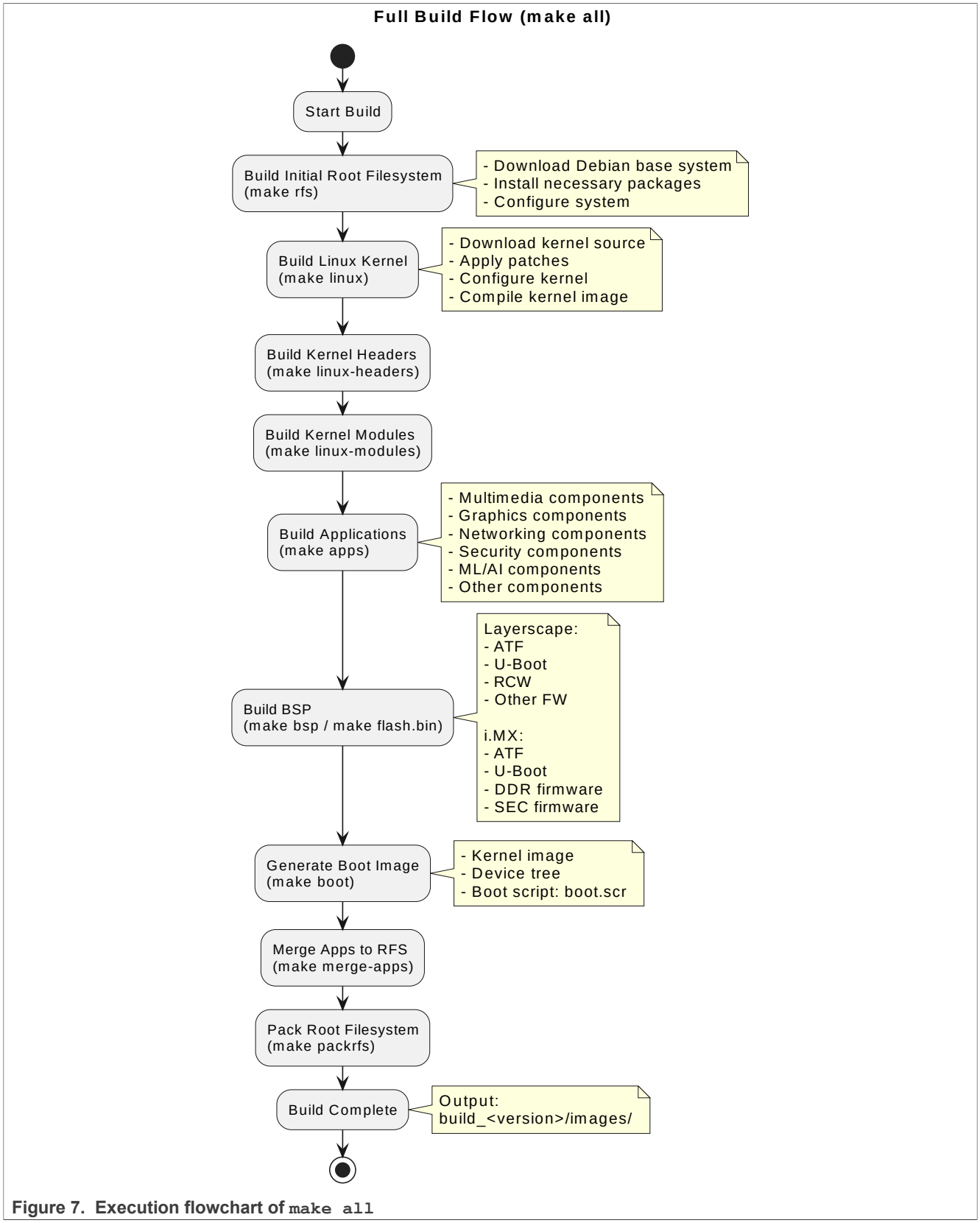

Figure 7. Execution flowchart of `make all`

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

After the command completes successfully, the three expected images are generated in the `build_<version>/images/` directory:

- **BSP image**: `firmware_<machine>_<boot_mode>_[sec].img` or `<machine>-sd-flash.bin`
- **Boot image**: `boot_<SOCFAMILY>_arm64_lts_<kernel_ver>.tar.zst`
- **Rootfs image**: `rootfs_<version>_debian_<machine>.tar.zst`

The three files can be deployed to the target board using the `flex-installer` tool.

4.4 Detailed build process for individual images

Assuming that the Docker container has already been created and the `.config` file has been prepared, the following sections describe how to build each of the three images.

4.4.1 Building the BSP composite firmware image

The BSP firmware image can be generated with the following command.

- **Layerscape platform**

```
$ make bsp
```

The program flowcharts are illustrated below.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

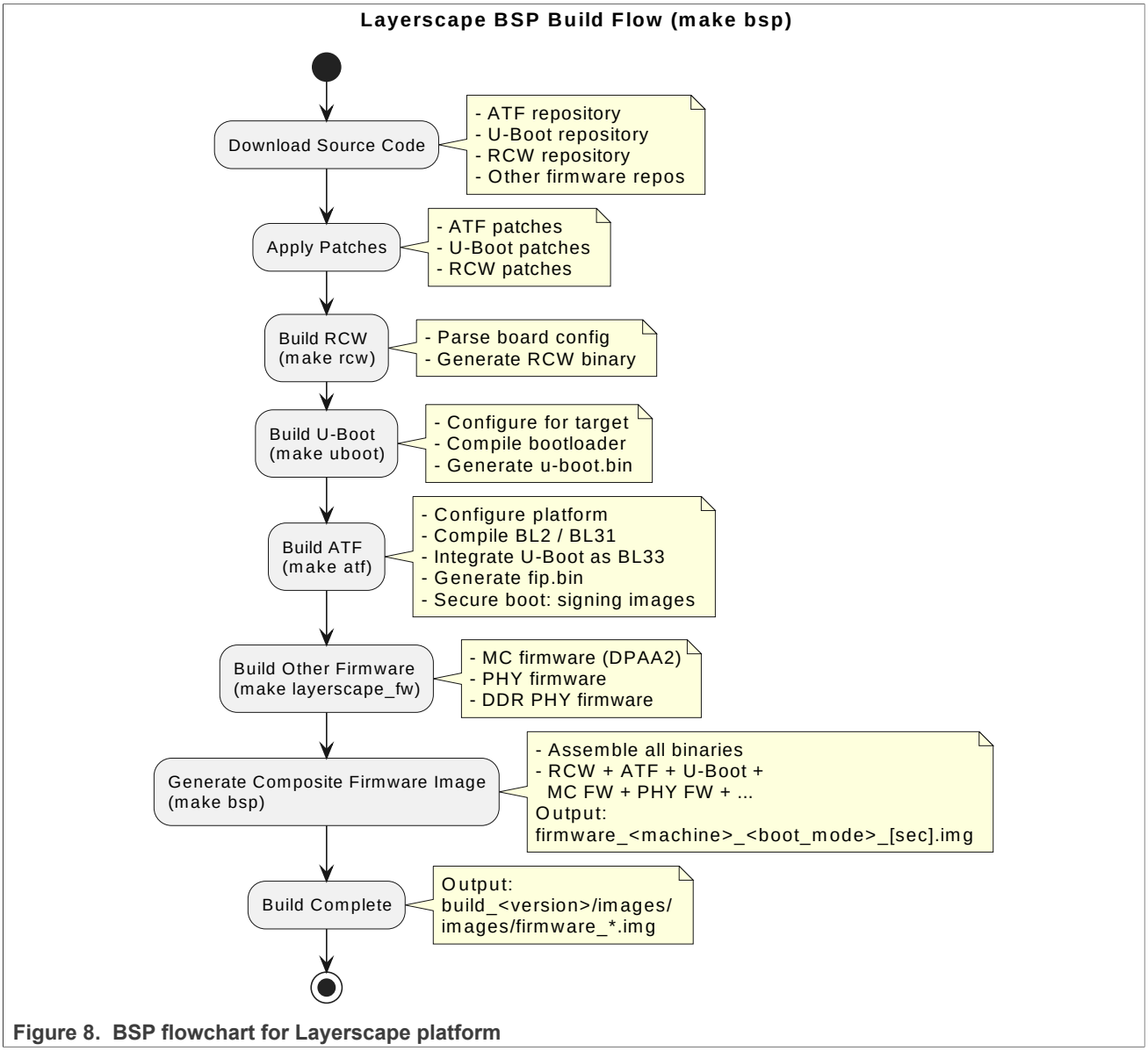

Figure 8. BSP flowchart for Layerscape platform

- **i.MX platform**

```
$ make flash.bin
```

The program flowchart is illustrated below.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

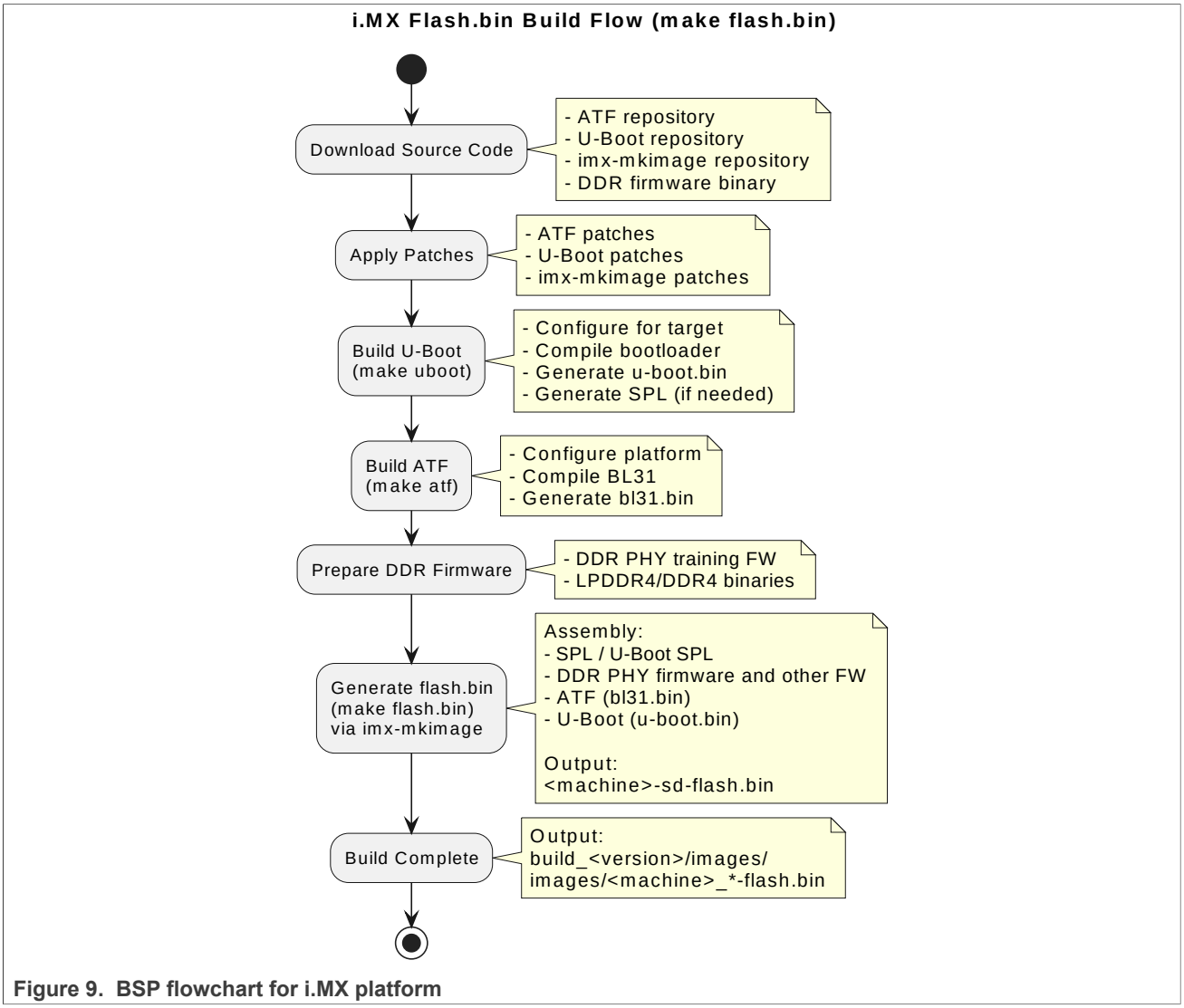

Figure 9. BSP flowchart for i.MX platform

4.4.2 Building the boot partition image

The boot partition image can be generated with the following command.

```
$ make boot
```

The program flowchart is illustrated below.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

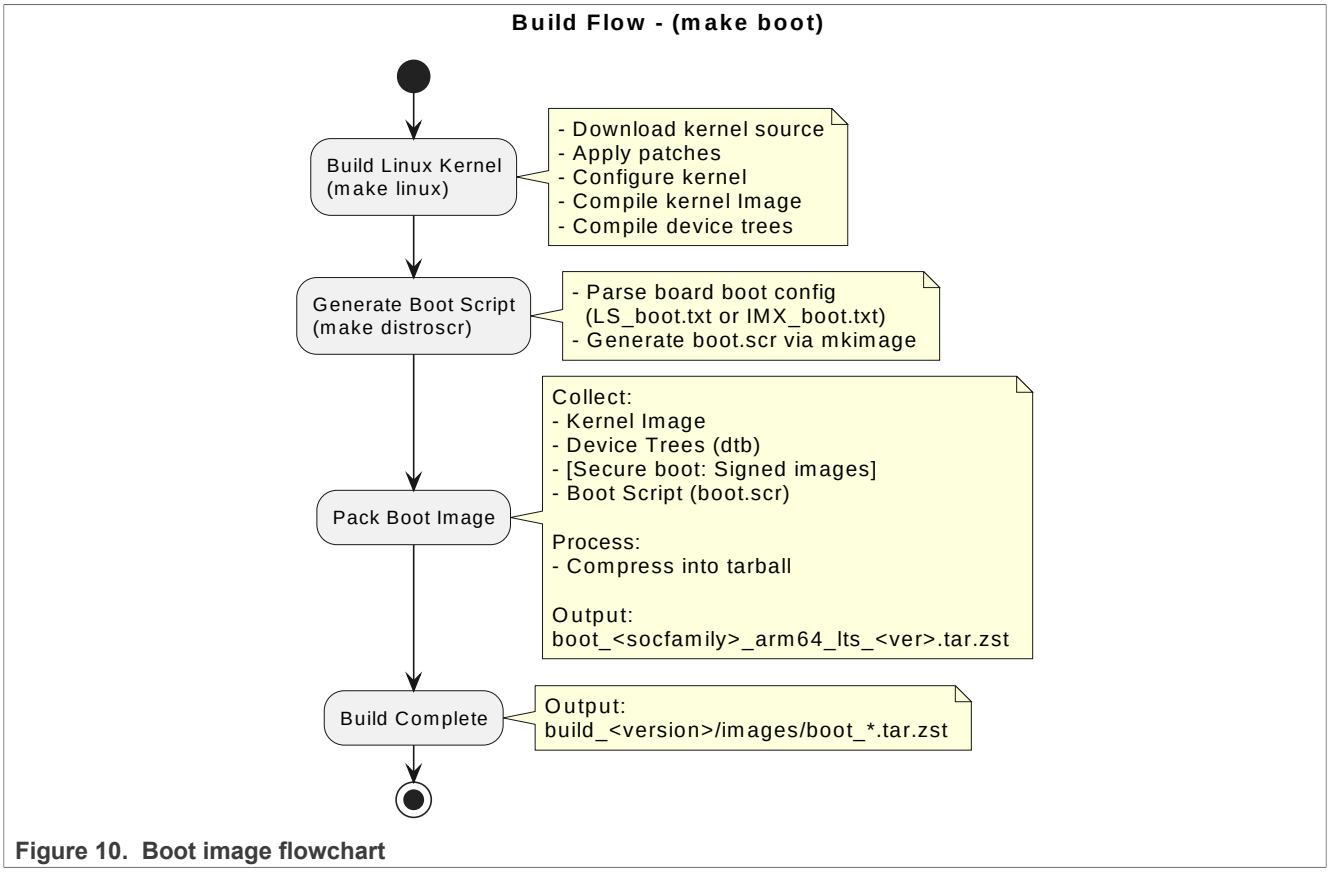

Figure 10. Boot image flowchart

4.4.3 Building the root filesystem image

The rootfs partition image is generated using the following sequence of commands.

```
$ make rfs && make linux && make linux-headers && make linux-modules
$ make apps && make merge-apps && make packrfs
```

The program flowchart is illustrated below.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

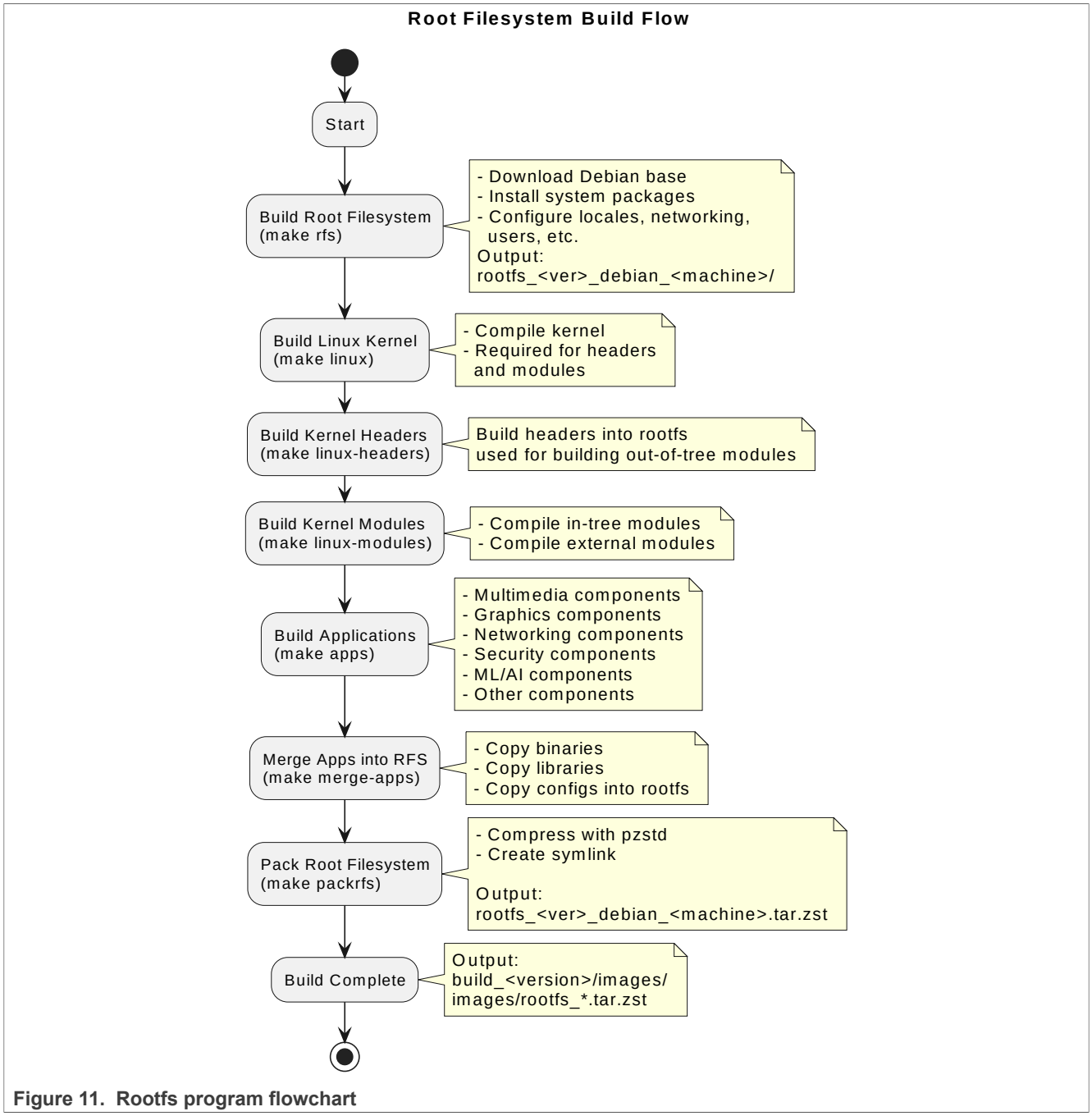

Figure 11. Rootfs program flowchart

If an application is modified later, you can regenerate the rootfs using the following sequence of commands:

```
$ make <app_name> && make merge-apps && make packrfs
```

4.5 Other build targets

The following commands provide additional build functionality and utilities (assume `.config` has already been generated).

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Display commonly used make targets with brief descriptions:

```
$ make Or $ make help
```

- Launch the interactive menu to configure Linux kernel options:

```
$ make linux-menuconfig
```

- Build a specific application or component:

```
$ make <app_name>
```

Example:

```
$ make weston
```

- List all the currently supported machines:

```
$ make list
```

- Generate a WIC disk image ready to flash to SD card or eMMC:

```
$ make wic
```

Note: *This command only packages the three required system files into a `.wic.zst` file. It does not perform any compilation. Therefore, before running this command, ensure that the required three files already exist (for example, `make all` has been executed successfully).*

- Display all applications enabled in the current configuration:

```
$ make show-enabled-apps
```

- List all available applications/components, including disabled ones:

```
$ make list-all-apps
```

- Remove Linux kernel build artifacts:

```
$ make clean-linux
```

- Remove BSP (U-Boot, ATF, etc.) build artifacts:

```
$ make clean-bsp
```

- Remove build artifacts for all applications/components:

```
$ make clean-apps
```

- Remove boot partition build artifacts:

```
$ make clean-boot
```

- Remove configuration files:

```
$ make clean-config
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Remove root filesystem (rootfs) build output:

```
$ make clean-rfs
```

- Remove all build artifacts, including source code (full reset):

```
$ make distclean
```

- Check and install required build dependencies on the host machine:

```
$ make host-dep
```

- Generate a FIT format `.itb` boot image (kernel, DTB, ramdisk, etc.):

```
$ make itb
```

4.6 Debugging guide

This section describes how to debug in the FlexBuild project, including log level control, debug options, and Makefile debugging methods.

4.6.1 `LOG_LEVEL` - Log Level Control

`LOG_LEVEL` is used to control the verbosity of log output during the build process.

Usage

```
$ make <target> LOG_LEVEL=<level>
```

Log Level description

- `LOG_LEVEL=0`: Verbose output, shows more build details.
- `LOG_LEVEL=1`: Minimal output, only critical information and errors.
- `LOG_LEVEL=2`: Mute all the output (default level).

Example

```
$ make uboot LOG_LEVEL=0
```

4.6.2 `MKDEBUG` - Makefile debug mode

`MKDEBUG` enables Makefile debug output to help trace the execution of build rules.

Usage

```
$ make <target> MKDEBUG=y
```

Features

When `MKDEBUG=y` is enabled:

- Shows execution of each Makefile rule.
- Prints variable expansion process.
- Outputs detailed information before command execution.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Example

```
$ make uboot MKDEBUG=y
```

4.6.3 `FBDEBUG` - FlexBuild debug mode

`FBDEBUG` is a FlexBuild-specific debug option used to debug Flexbuild scripts themselves.

Usage

```
$ make <target> FBDEBUG=y
```

Features

When `FBDEBUG=y` is enabled:

- Shows shell script execution trace (set `-x`).
- Prints function call stack.
- Displays the configuration file loading process.
- Outputs environment variables and path information.

Example

```
$ make itb FBDEBUG=y
```

In addition to the methods mentioned above, you can also directly use the debugging features provided by the `make` itself. For example:

- Shows all the debug information:

```
$ make -d <target>
```

- Prints Make's internal database, showing all rules and variable definitions:

```
$ make -p <target>
```

- Dry run (Print commands without executing):

```
$ make -n <target>
```

- Trace specific target:

```
$ make --debug=v <target>
```

5 Image deployment

This section describes how to deploy NXP distro images (including firmware, boot images, and root filesystem) onto target storage devices, such as SD cards, eMMC, USB drives, or SATA drives. Deployment can be performed from either a Linux host machine, a Windows host machine, or directly on an Arm board running TinyLinux.

The following table lists the partition information of the storage medium after deployment.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 3. Default partitions of the SD/USB/SATA storage media installed by flex-installer

Region 1 Partition Table 32K/33K	Region 2 Raw 64 - 256 MiB Composite firmware	Region 3 EXT4 512 MiB Boot Partition-1	Region 4 EXT4 Remaining RootFS Partition-3
MBR or GPT	Bootloader env Firmware	kernel & dtb distro boot.scr modules & firmware	Debian RootFS

5.1 Deploying the pre-built images

Pre-built images are officially released by NXP and hosted on the NXP distribution server. The `flex-installer` tool can automatically download and install these pre-built images onto a target storage device.

Note:

- *Only the Debian Base rootfs is released. You must run the `debian-post-install-pkg` (already included in Debian Base rootfs) script on the target board to upgrade it to a full-featured rootfs.*
- *For a detailed pre-built image list of the release, see the Debian Software Release Notes (RN00266)*

5.1.1 Deploying on a Linux host machine

5.1.1.1 Deploying using the `flex-installer` tool

The `flex-installer` tool is a general-purpose distro installer provided by the Flexbuild system. It supports automatic downloading, partitioning, formatting, and installation of distro images.

Prerequisites:

- A Linux host machine (x86_64 or aarch64) with the following required commands installed: `parted`, `curl`, `sudo`, `fuser`, `mount`, `mkfs.ext4`, `zstd`, `udevadm`, `partprobe`.
- A target storage device (SD card, USB drive, or SATA drive) connected to the host machine.
- Network access to the NXP distribution server (www.nxp.com/lgfiles/sdk/<version>).
- Root or Sudo privileges.

1. The `flex-installer` can be obtained using the following command:

   ```
   $ wget https://www.nxp.com/lgfiles/sdk/<version>/flex-installer
   ```

   For example:

   ```
   $ wget https://www.nxp.com/lgfiles/sdk/lsdk2606/flex-installer
   ```

   Grant the execution permission to it:

   ```
   $ chmod a+x flex-installer
   ```

2. Identify the target storage device.
   Insert the target SD card or USB drive into the Linux host machine, and then identify the device name:

   ```
   $ lsblk
   ```

   The device name is typically `/dev/sdX` (for USB/SATA drives) or `/dev/mmcblk0` (for SD/eMMC cards).
   Note: *Be careful to identify the correct device to avoid data loss.*
3. Partition and format the target storage device.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Before installing images, the target device must be partitioned and formatted:

```
$ ./flex-installer -i pf -d /dev/sdX
```

This creates the default partition layout (`2P=512M:-1` for LSDK2606, or `3P=512M:8G:-1` for LSDK24xx/25xx), which includes:

- A raw region (first 256 MB) reserved for firmware.
- A boot partition (512 MB, ext4, labeled boot).
- A root filesystem partition (remaining space, ext4, labeled system).
- (For LSDK24xx/25xx) An additional backup partition.

4. Automatically download and install distro images.
   Use the `auto` instruction to automatically download and install the default distro (Debian) for a specific machine:

```
$ ./flex-installer -i auto -m <machine> -d /dev/sdX
```

Replace `<machine>` with the target board name.
To list all the supported machines, run:

```
$ ./flex-installer -i list
```

The following are common examples of using auto instruction.

- **Example: Deploy Debian distro to an i.MX 8M Plus EVK through the SD card:**

```
$ ./flex-installer -i auto -m imx8mpevk -d /dev/sdX
```

This command will:

  - Download the BSP firmware (`firmware_imx8mpevk_sdboot.img` or `imx8mpevk-sd-flash.bin`), the boot partition image (`boot_IMX_arm64_lts_<kernel_ver>.tar.zst`), and the root filesystem image (`rootfs_<version>_debian_base_arm64.tar.zst`) from the NXP distribution server.
  - Partition and format the target device (if not already properly partitioned).
  - Program the BSP firmware to the raw region of the SD card.
  - Extract the boot image into the boot partition.
  - Extract the root filesystem into the root partition.
  - Configure UUID-based mount entries for the boot partition.

- **Example: Deploy a specific distro version:**

```
$ ./flex-installer -i auto:lsdk2512 -m imx8mpevk -d /dev/sdX
```

- **Example: Override the default rootfs with a custom one during auto deployment:**

```
$ ./flex-installer -i auto -m imx8mpevk -d /dev/sdX -r
 rootfs_lsdk2606_debian_imx8mpevk.tar.zst
```

- **Example: Use a custom distribution server URL:**

```
$ ./flex-installer -i auto -m imx8mpevk -d /dev/sdX -u https://my-internal-
server/
```

5. Boot the board.
   a. Remove the SD card from the host machine and insert it into the target board.
   b. Configure the board's boot switches to boot from the SD card.
   c. Power on the board.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

d. If you used Debian Base rootfs, run the `debian-post-install-pkg` script on the target board to upgrade it to a full-featured rootfs.

5.1.1.2 Deploying using the `deploy-wizard.sh` Tool

The `deploy-wizard.sh` tool provides an interactive, menu-driven interface for deploying distro images. It wraps `flex-installer` in a user-friendly interface to guide users through a step-by-step deployment process.

Prerequisites:

- Same as that for the `flex-installer` tool (see Section 5.1.1.1).

1. Get the tools:

```
$ wget https://www.nxp.com/lgfiles/sdk/<version>/deploy-wizard.sh
```

For example:

```
$ wget https://www.nxp.com/lgfiles/sdk/lsdk2606/deploy-wizard.sh
```

2. Grant the execution permission to it:

```
$ chmod a+x deploy-wizard.sh
```

Usage:

```
$ ./deploy-wizard.sh
```

The wizard will interactively prompt the user to:

1. Select the distro version and variant to install.
2. Select the target machine from a list of supported machines.
3. Select the target storage device from detected devices.
4. Confirm the operation before proceeding.

The `deploy-wizard.sh` internally invokes `flex-installer` with the appropriate arguments to perform the actual installation. It is recommended for users who are less familiar with the command-line options of `flex-installer`.

5.1.2 Deploying on Arm board running TinyLinux

The `flex-installer` tool also supports deployment directly on an Arm board that is running TinyLinux (a minimal root filesystem booted through NFS or RAM).

The usage is similar:

```
$ ./flex-installer -i auto -m <machine> -d /dev/<device_name>
```

Note: *`<device_name>` can be `mmcblk0`, `sda`, or other valid storage devices.*

5.1.3 Deploying on a Windows host machine

This section describes how to deploy NXP i.MX/Layerscape platform firmware images to an SD card on a Windows host machine.

Prerequisites:

Hardware requirements

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Windows 10/11 PC
- SD card reader (USB or built-in)
- MicroSD card (16 GB or larger recommended, Class 10 or UHS-I)

Software requirements

- **Python 3.7+** (for running `mkfw-wic.py`)
- **balenaEtcher** (for flashing images)

1. Install the required software.
   Install Python 3.
   a. Visit Python official website.
   b. Download Python 3.x Windows installer.
   c. Run the installer and check "Add Python to PATH".
   d. Verify installation:

```
C:\ python --version
Expected output:
Python 3.11.x
```

   Install balenaEtcher.
   a. Visit balenaEtcher official website.
   b. Download Windows version (`.exe` installer).
   c. Run the installer and follow the prompts to complete the installation.
2. Download the `mkfw-wic.py` tool.
   Get the tool from the Flexbuild repository github.com/nxp/flexbuild.git.
   `mkfw-wic.py` is located in the `tools/` directory.
3. Create the firmware WIC image.
   Use the following command to auto-download and create the WIC image.

```
C:\ python mkfw-wic.py -m <machine>
```

   For example:

```
C:\ python mkfw-wic.py -m imx8mpevk
```

   The WIC file `imx8mpevk_<version>_fw.wic` is generated.
4. Flash the WIC image with balenaEtcher.
   a. Insert the SD card.
      Connect the microSD card to your Windows PC through a card reader.
   b. Launch balenaEtcher.
      i. Open the balenaEtcher application
      ii. The interface shows three steps: **Flash from file** → **Select target** → **Flash!**
   c. Select the WIC image file.
      i. Click the **Flash from file** button.
      ii. Browse and select the generated WIC file (such as `imx8mpevk_lsdk2606_fw.wic`).
   d. Select the target SD card.
      i. Click the **Select target** button
      ii. Select your SD card device from the list.
      - **Warning**: Carefully verify the device to avoid data loss!

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Tip: Identify the SD card by its capacity

e. Start flashing.
   i. Click the **Flash!** button.
   ii. If the UAC (User Account Control) prompt appears, click **Yes** to authorize.
   iii. Wait for flashing to complete (progress bar shows status).

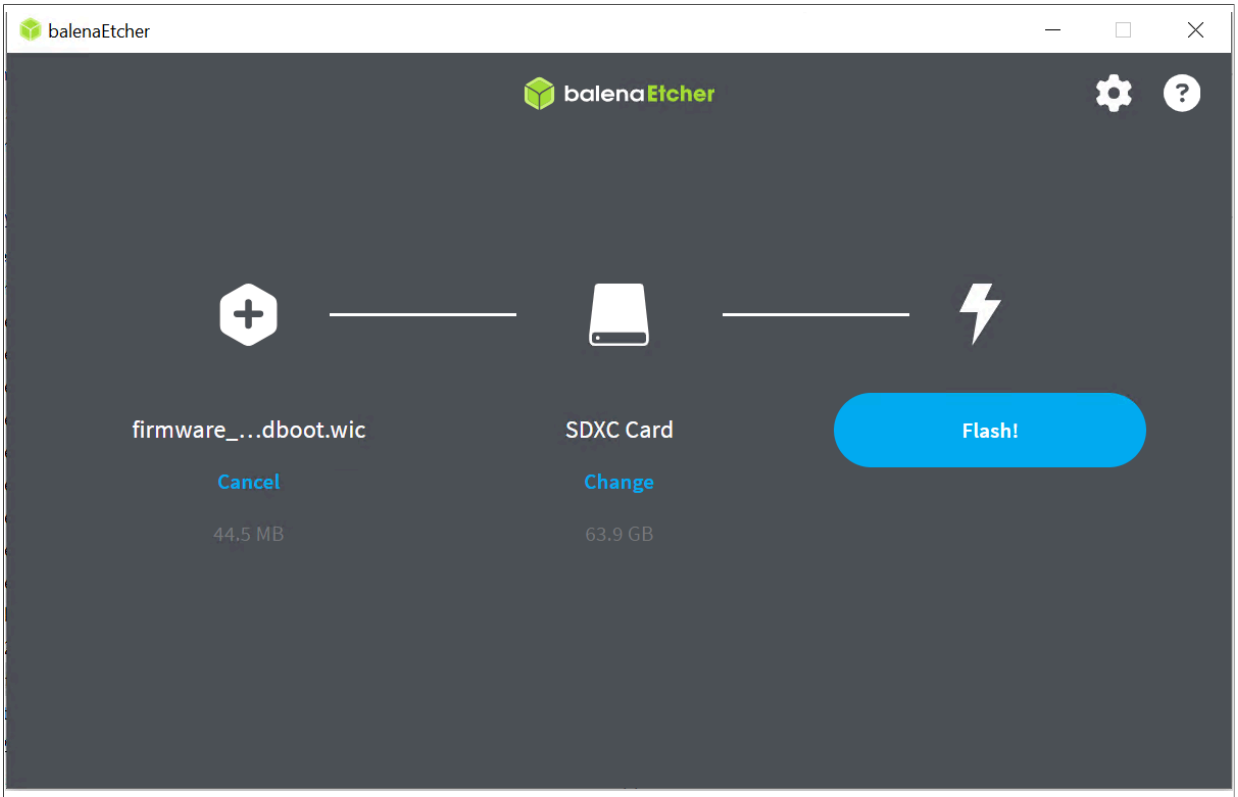

Figure 12. Flashing image by the Etcher tool

5. Boot up TinyLinux and install the Debian Distro by `flex-installer` as follows.
   a. Unplug the SD card from the Windows host and plug it into the target board. Then, set the DIP switch for SD boot if needed.
   b. After powering on the target board, run the following commands under the U-Boot prompt to boot TinyLinux.

```
For i.MX board:
=> setenv tinylinux 'mmc read 0xa0000000 0x4000 0x1f000 && bootm
 a0000000#imx8mpevk'
=> saveenv; run tinylinux

For Layerscape board:
=> run sd_bootcmd
or
=> mmc read a0000000 0x8000 0x1f000 && bootm a0000000#<board_name>
e.g.,
=> setenv tinylinux 'mmc read a0000000 0x8000 0x1f000 && bootm
 a0000000#lx2160ardb'
=> saveenv; run tinylinux
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

c. Log in to the TinyLinux with the username `root`, set up the network on the board, and install the Debian Distro by the following commands:

```
root@TinyLinux:~#  udhcpc -i eth0
(DHCP dynamic IP or manually set static IP)

root@TinyLinux:~#  flex-installer -i pf -d /dev/mmcblk1
(format SD card)

root@TinyLinux:~#  flex-installer -i auto -d /dev/mmcblk1 -m imx8mpevk
(automatically download and install Debian-base, boot, and firmware
 images.)
```

This takes several minutes to download and install the i.MX Debian base RootFS image, boot tarball image, and BSP firmware image onto the SD card.
After finishing the installation in TinyLinux, run the reboot command to reset the system to log into the Debian base system.

d. Once logging into the Debian base system with the username `debian` or `root`, run the following commands to install extra packages (Optionally, set the HTTP proxy for apt in `/etc/apt/apt.conf.d/proxy.conf` if needed in your network environment).

```
$ dhclient -i end0
(setup Ethernet network interface by DHCP or setting it manually)

$ sudo date -s "12 DEC 2024 15:00:00"
(setting correct system time is required)

$ debian-post-install-pkg
(install NXP-specific desktop packages and extra deb  packages for GNOME
 GUI or Weston Desktop version)
```

This step installs the prebuilt board-specific hardware driver components and extra deb packages in about half an hour.

After finishing the installation, run the reboot command to boot up the Debian system.

5.2 Deploying self-built images

If you have built your own distro images using `flexbuild`, you can deploy them to the target storage device using the methods described below.

5.2.1 Deploying using the `flex-installer` Tool

After building images with Flexbuild, the output images are typically located in the `build_<version>/images/` directory.

You can use `flex-installer` with the `install` instruction by providing the image files directly to deploy them.

1. Partition and format the target device (if needed).
   If the target device has not been partitioned and formatted yet, run:

```
$ ./flex-installer -i pf -d /dev/sdX
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Optionally, specify a custom partition layout:

```
$ ./flex-installer -i pf -d /dev/sdX -p 3P=512M:8G:-1
```

2. Install all images (`firmware`, `boot`, and `rootfs`) at once.

```
$ ./flex-installer -d /dev/sdX -m <machine> \
  -f firmware_<machine>_sdboot.img ( or <machine>-sd-flash.bin) \
  -b boot_<SOCFAMILY>_arm64_lts_<kernel_ver>.tar.zst \
  -r rootfs_<version>_debian_<machine>.tar.zst
```

This command will:

- Program the BSP composite firmware image to the raw area of the SD card with the appropriate offset (automatically determined based on the machine type: 33K for i.MX 8M Quad and i.MX 8M Mini, 32K for other i.MX 8 and i.MX 9, 4K for Layerscape).
- Extract the boot partition image into the boot partition.
- Extract the root filesystem image into the root partition.
- Clean up irrelevant device tree files from the boot partition, keeping only those matching the target machine.
- Set up boot scripts file `boot.scr` and `symlinks` (for `lsdk24xx` and `lsdk25xx`).
- Configure the UUID-based boot mount and `fstab` entries.

5.2.2 Deploying the WIC image

A WIC (Yocto-style whole-image) image is a single file that contains the complete disk layout, including the partition table, firmware, boot partition, and root filesystem.

1. Create a WIC image.
   Use the `mkwic` instruction to create an all-in-one SD card WIC image:

```
$ ./flex-installer -i mkwic -m <machine> \
  -f firmware_<machine>_sdboot.img  or <machine>-sd-flash.bin \
  -b boot_IMX_arm64_lts_<kernel_ver>.tar.zst \
  -r rootfs_<version>_debian_<machine>.tar.zst
```

This command will:

- Create a raw image file (`rootfs_<version>_debian_<machine>.wic`) with a default size of 10240 MB (10 GB).
- Attach the image to an available loop device.
- Partition and format the loop device with the default loop device partition configuration (`2P=512M:-1` or `3P=512M:64M:-1`).
- Program the firmware, boot image, and rootfs into the respective partitions.
- Detach the loop device.
- Compress the WIC image with `pzstd` to produce `rootfs_<version>_debian_<machine>.wic.zst`.
- Remove the uncompressed WIC image to save disk space.

You can also specify a custom partition layout for the WIC image, for example:

```
$ ./flex-installer -i mkwic -m imx8mpevk \
  -f imx8mpevk-sd-flash.bin \
  -b boot_IMX_arm64_lts_6.12.49.tar.zst \
  -r rootfs_lsdk2606_debian_imx8mpevk.tar.zst \
  -p 3P=512M:8G:-1
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Note:

- *The total partition size cannot exceed the WIC image size limit (default 10240 MB). If a larger image is needed, modify the `DEFAULT_SDCARD_IMG_SIZE_MB` variable in the script.*
- *WIC image can also be generated during the compilation stage using the following command:*

```
$ make wic
```

2. Flash the WIC image to the SD card.

```
$ zstd -d rootfs_<version>_debian_<machine>.wic.zst
$ sudo dd if=rootfs_<version>_debian_<machine>.wic of=/dev/sdX bs=1M
 conv=fsync status=progress
```

3. Boot the board
   Insert the SD card into the target board and power it on.

5.2.3 Other usages of `flex-installer`

Beyond the primary deployment scenarios described above, the `flex-installer` tool provides several additional capabilities.

- **Partition and Format Only**
  To partition and format a storage device without installing any images, use the `pf` instruction:
  – Default partition layout:

```
$ ./flex-installer -i pf -d /dev/sdX
```

  – Custom partition layout (up to 4 partitions):

```
flex-installer -i pf -d /dev/sdX -p 4P=800M:6G:8G:-1
```

  The partition specification format is `<N>P=<size1>:<size2>:...:<sizeN>`, where:
  – `<N>` is the number of partitions (must be between 2 and 4).
  – `<sizeX>` is the size of each partition in MB (M) or GB (G).
  – `-1` for the last partition means "use all remaining disk space."
  When configured with 4 partitions, the partition details are shown in the following table.

Table 4. Partition layout details

Component	Location	Description
Raw firmware region	Beginning of disk	First 256 MB reserved for composite firmware (not a partition).
Partition 1 (boot)	After raw region	Boot partition containing kernel, device trees, and boot scripts. Labeled boot.
Partition 2 (system)	After Partition 1	Root filesystem partition. Labeled system.
Partition 3 (data)	After Partition 2 (if configured)	Back up an additional data partition. Labeled data3.
Partition 4 (data)	After Partition 3 (if configured)	Additional data partition. Labeled data4.

  Note: *The maximum number of primary partitions supported is 4 (MBR partition table limitation). Partition table type is `msdos` (MBR).*
- **Install individual images selectively.**

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

You can also install only specific images without affecting others:

– Install only the root filesystem:

```
$ ./flex-installer -r rootfs_<version>_debian_<machine>.tar.zst -d /dev/sdX
```

– Install only the boot partition image:

```
$ ./flex-installer -b boot_<SOCFAMILY>_arm64_lts_<kernel_ver>.tar.zst -d /
dev/sdX -m <machine>
```

– Install only the BSP composite firmware:

```
$ ./flex-installer -f firmware_<machine>_sdboot.img -d /dev/sdX -m <machine>
```

- **Install to a non-default partition.**
  By default, the boot partition is partition 1 and the root partition is partition 2.
  You can override these defaults.
  Install rootfs to the third partition:

```
$ ./flex-installer -r rootfs_<version>_debian_<machine>.tar.zst -R 3 -d /dev/
mmcblk0
```

  After installation, boot the board and run the following in U-Boot command to boot from the custom partition:

```
=> setenv devpart_root 3; boot
```

- **Download images only.**
  To download distro images from the NXP distribution server without installing them:

```
$ ./flex-installer -i download -m <machine>
```

  Download a specific distro version:

```
$ ./flex-installer -i download:lsdk2512 -m <machine>
```

  The image is then downloaded to the current working directory. If an image file already exists in the current directory, it will not be re-downloaded.
- **List supported machines and environments.**

```
$ ./flex-installer -i list
```

- **Print the version information.**

```
$ ./flex-installer -v
```

  Sample output:

```
flex-installer version: lsdk2606
```

- **Force installation.**
  By default, the `flex-installer` prompts for confirmation before overwriting existing data on the target device. To skip this confirmation and force the installation:

```
$ ./flex-installer -i auto -m imx8mpevk -d /dev/sdX -F
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

The `-F` (or `--force`) option bypasses all confirmation prompts and proceeds directly with partitioning, formatting, and installation. **Use with caution**, as this will destroy all the existing data on the target device.

- **Command-line options reference.**
  The following table lists all the available command-line options.

Table 5. `flex-installer` command-line options

Option	Long Option	Argument	Description
-i	--instruction	<instruction>	Instruction to execute: `auto`, `pf`, `install`, `download`, `list`, `mkwic`
-b	--bootimg	<boot>	Path to the boot partition image file
-r	--rootfsimg	<rootfs>	Path to the root filesystem image file
-f	--firmware	<firmware>	Path to the BSP firmware image file
-d	--device	<device>	Target storage device (such as `/dev/sdX`, `/dev/mmcblk0`)
-m	--machine	<machine>	Target machine/board name
-p	--partition	<partition>	Custom partition layout (such as `3P=512M:8G:-1`)
-B	--bootpart	<num>	Override default boot partition number
-R	--rootpart	<num>	Override default root partition number
-u	--url	<url>	Override default distro server URL
-o	--offset	<offset>	Firmware write offset (such as 1k, 4k, 32k, 33k)
-F	--force	(none)	Force installation without confirmation prompts
-v	--version	(none)	Print version information
-h	--help	(none)	Display usage help

Note: *The `flex-installer` tool automatically determines the firmware write offset based on the target machine type. Generally, there is no need to explicitly specify the `-o` or `--offset` option.*
The following table lists the default offsets.

Table 6. Default offset for platforms

Machine type	Offset	Description
i.MX 8M Mini	33 KB	i.MX 8M Mini
i.MX 8 (other), i.MX 9	32 KB	i.MX 8 and i.MX 9 platforms
Layerscape (LS, LX, LA)	4 KB	All Layerscape platforms

5.3 Troubleshooting

The following table lists common issues and solutions.

Table 7. Common Issues and Solutions

Issue	Cause	Solution
Device `/dev/sdX` not found.	Target device is not connected or the device name is incorrect.	Verify the device with `lsblk` and ensure it is connected.
HTTP returned `000` or connection errors.	No network access to the distribution server.	Check the network connectivity, proxy settings, and DNS resolution.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 7. Common Issues and Solutions*...continued*

Issue	Cause	Solution
Missing required commands.	Required tools are not installed.	Install missing packages: `apt install parted curl zstd e2fsprogs udev`.
Total partition size exceeds the WIC image size limit.	Custom partition layout is too large for the WIC image.	Reduce partition sizes or increase `DEFAULT_SDCARD_IMG_SIZE_MB`.
Failed to install rootfs on the running root filesystem.	Attempting to overwrite the currently running OS partition.	Use a different target partition or run from TinyLinux.
The number of partitions cannot exceed 4.	More than 4 partitions specified.	Reduce the number of partitions to 4 or fewer.
Installation interrupted (Ctrl +C).	User interruption or error.	The tool automatically cleans up mounted filesystems and loop devices on exit.

6 FAQs

6.1 How to burn i.MX Debian Linux SDK images generated by Flexbuild to SD card or eMMC device by UUU tool?

Here takes i.MX 8M Plus EVK as an example.

1. Build Debian images by Flexbuild.

```
$ make imx8mpevk_defconfig && make all
```

2. Generate the `.wic` image by `flex-installer` as follows:

```
$ flex-installer -i mkwic -m imx8mpevk \
      -f build_<version>/images/<machine>-sd-flash.bin \
      -b build_<version>/images/boot_IMX_arm64_<lts_version>.tar.zst \
      -r build_<version>/rfs/rootfs_<lsdk_version>_debian_<machine>
```

3. Set the DIP switch to USB serial download mode and burn images by UUU.

```
$ uuu -b emmc_all build_<version>/images/imx8mpevk-sd-flash.bin
rootfs_<versino>_debian_imx8mpevk.wic.zst (compressed format, or '-b sd_all'
 for SD)
```

6.2 How to fix the build issue with log "/usr/bin/apt-key: cannot create /dev/null: Permission denied" when running `bld rfs` in Flexbuild on certain Linux host?

This issue might be caused by the settings of disk access permission. You can change `nodev` to `dev`, `noexec` to `exec`, `nosuid` to `suid` in `/etc/fstab` on the host machine and reboot the host. Then run the following commands:

```
make clean-rfs
make rfs && make merge-apps && make packrfs
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

6.3 How to install the Chinese input method package on board?

1. Set the local Locale.

```
$ dpkg-reconfigure locales
```

   Choose **en_US.UTF-8** and **zh_CN.UTF-8**.
   Press the **Tab** key to choose **OK** for confirmation, and then choose **en_US.UTF-8** locale.
2. Install the Chinese fonts.

```
$ sudo apt-get update
$ sudo apt-get install xfonts-intl-chinese wqy*
```

3. Install the Chinese input method package.

```
$ sudo apt-get install ibus ibus-gtk ibus-pinyin ibus-libpinyin
```

4. Reboot the Debian system. Choose **Settings** -> **Keyboard**. Click **+** in **Input Source**. Add an input source, and choose **Chinese** (Intelligent Pinyin).
5. Run the `gnome-tweaks` command to enable the buttons of minimization and maximization on the left title bar of the window.

6.4 Where is the source code stored?

Note: *You must have successfully compiled a complete machine previously. Otherwise, the source directories do not exist. This means that you should have run a command similar to:* `$ make all`.

- Kernel and kernel modules related source code is located in: `<flexbuild dir>/components_<version>/linux/linux/`.
- BSP (including U-Boot, ATF, RCW, etc.) related code is located in: `<flexbuild dir>/components_<version>/bsp`.
- Applications related code is categorized into graphics, multimedia, machine learning, and other relevant categories, and is stored in their respective subdirectories within: `<flexbuild dir>/components_<version>/apps/`.

6.5 How to modify and recompile the source code

- To modify the Linux or Device Tree Blob (DTB):
  1. Navigate to the Linux source directory:

```
$ cd components_<version>/linux/linux/
```

  2. Update your source code or DTB files.
  3. Go back to the previous directory:

```
$ cd -
```

  4. Compile the Linux kernel:

```
$ LOG_LEVEL=0 make linux
```

  5. The compiled output is in the `build_<version>/linux/linux/arm64/<IMX or LS>/` directory. Copy the necessary image or DTB files to your target board for use.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Note: *To use the `flex-installer` tool to flash the newly generated kernel image or DTB, run the following command to generate the boot image `boot_<SOCFAMILY>_arm64_lts_<kernel_ver>.tar.zst`:*

```
$ LOG_LEVEL=0 make boot
```

- To modify the U-Boot or ATF Code:
  1. Navigate to the U-Boot or ATF source directory:

     ```
     $ cd components_<version>/bsp/uboot/ or components_<version>/bsp/atf/
     ```

  2. Update the source code.
  3. Compile U-Boot (replacing with your specific machine name):

     ```
     $ LOG_LEVEL=0 make bsp // for Layerscape Or
     $ LOG_LEVEL=0 make flash.bin // for IMX
     ```

  4. The compiled results is in `build_<version>/images/`.
- To modify and use the Application Source Code:
  1. Navigate to the application module's directory (using `imx_lib` as an example):

     ```
     $ cd components_<version>/apps/multimedia/imx_lib/
     ```

  2. Modify the relevant code. Go back to the previous directory:

     ```
     $ cd -
     ```

  3. Compile the application module:

     ```
     $ make imx_lib
     ```

  4. The compiled code is located under `build_<version>/apps/apps_debian_<machine>/`. Usually, the compiled output of a single component consists of multiple files, making them inconvenient for standalone use. In such cases, perform the following steps:
     a. Merge applications into the target root filesystem:

        ```
        $ make merge-apps
        ```

     b. Repackage to generate a new root filesystem:

        ```
        $ make packrfs
        ```

     The new root filesystem is located at `build_<version>/images/rootfs_<version>_debian_<machine>.tar.zst`, which can then be installed using `flex-installer`.

6.6 How to modify code using patches

Follow the steps below to apply patches.

1. Create the corresponding directory (if the patch directory is not created yet).
2. Place your new patch:
   Copy your generated patch file into the `patch/<components>` directory, alongside with any existing patches.
3. Clean the old code of component:

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

This ensures that during your recompilation, the system applies all patches, including the new one. Take `imx_lib` as an example:

```
$ rm -rf components_<version>/apps/multimedia/imx_lib/ && make imx_lib
```

4. Recompile the source code:
   See Section 6.5 for recompilation instructions.
   The new patch is then automatically applied during this process.

6.7 How to add or remove a deb package in Flexbuild

Besides adding or removing a deb package by `sudo apt install <package>` or `sudo apt remove <package>` directly on the Debian system on the target board, users can also add or remove a deb package in/from Flexbuild during the build stage for customization.

If there is already an existing Debian RootFS on the host machine, run the following commands to install a new deb package or remove a deb package:

```
$ sudo chroot build_<version>/rfs/rootfs_<version>_debian_<machine> apt install
 <package>
$ sudo chroot build_<version>/rfs/rootfs_<version>_debian_<machine> apt remove
 <package>
$ make packrfs
(pack the target Debian RootFS as .tar.zst if needed)
```

6.8 How to add a new custom component in Flexbuild

To add a new component called `hello_world`, perform the following steps:

1. Set the relevant URL and tag/commit information for the new component.
   You can edit `configs/sdk.yml` to set the repository URL with a tag or commit for the `hello_world` Git tree if needed.
2. Create a makefile `src/apps/<subsystem>/hello_world.mk` to add the build object for this component.
   According to the various types of the build system (e.g., `make`, `cmake`, `meson`) in the new component, refer to the following examples to add the `hello_world.mk` file.
   - For building with `make`, refer to `src/apps/utils/imx_test.mk`.
   - For building with `cmake`, refer to `src/apps/graphics/gputop.mk`.
   - For building with `meson`, refer to `src/apps/multimedia/cheese.mk`.
3. Build the new component based on the target ARM64 Debian RootFS.

   ```
   $ make hello_world
   ```

4. Merge the new component into the target Debian RootFS.

   ```
   $ make merge-apps
   ```

5. Pack the target Debian RootFS.

   ```
   $ make packrfs
   ```

6.9 How to add a new board in Flexbuild

To add a custom board called `myboard`, perform the following steps:

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

1. Fetch the source Git repositories of various components in Flexbuild for the first time.

```
$ git clone https://github.com/nxp/flexbuild
$ cd flexbuild
$ make docker
```

2. (Optional) Add the board-specific BSP related patches for the custom board if needed.
   See Section 6.6 for how to apply patches.
3. Add configurations for a custom board in Flexbuild.
   - Add a configuration file in `configs/board/<board>.conf`.
     Copy an existing configuration file of a similar board and make necessary changes in the new `.conf` file.
   - (Optional) Add a node for the new board in `configs/linux/linux_arm64_<SOCFAMILY>.its` to generate the `.itb` image.
   - Add the `myboard` information to: `configs/board/Kconfig`.
     For example,

```
config SOC_MYBOARD
    bool "my board"
    help
        my sample board
```

   - Add `myboard` to a compatible SoC in `Kconfig`. For example, `myboard` is compatible with i.MX 8M Plus EVK:

```
config SOC_IMX8MP
    bool
    default y if SOC_IMX8MPEVK || SOC_IMX8MPFRDM || SOC_MYBOARD
```

   - Add the `myboard` machine name in `Kconfig`:

```
config MACHINE_NAME
    string
    default "myboard" if SOC_MYBOARD
```

4. Build the BSP composite firmware image for the new board.

```
$ bld clean-bsp
(optionally, to clean the obsolete bsp images)
$ make myboard_defconfig
$ make bsp // for Layerscape Or
$ make flash.bin // for IMX
$ make boot
```

   This generates the `firmware_myboard_sdboot.img or myboard-sd-flash.bin` and `boot_<SOCFAMILY>_arm64_lts<kernel_ver>.tar.zst` images for the new board.
5. Build application components based on Debian RootFS if needed.

```
$ make rfs
$ make apps
$ make merge-apps
$ make packrfs
```

6. Deploy the Distro image on the SD card. Add `myboard` support in `flex-installer`:
   a. Add `myboard` to variable: `imx_machine_list` or `ls_machine_list in flex-installer`.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

b. Update the following code as needed:

```
prefix_to_keep=""
case "$machine" in
     imx8*)
         prefix_to_keep="${machine:0:6}" ;;
     imx9*)
         prefix_to_keep="${machine:0:5}" ;;
     lx*|ls*|la*)
         prefix_to_keep="fsl-${machine:0:6}" ;;
     *)
         echo "Unsupported machine type: $machine"
         exit 1 ;;
esac
sudo find "$bootpartition" -maxdepth 1 -type f -name "*.dtb" ! -name
 "${prefix_to_keep}*.dtb" -delete 2>/dev/null || true
```

c. After updating `flex-installer` tools, you can deploy your image with it:

```
$ flex-installer -i pf -d <device>
(partition and format SD card)
$ cd build_<version>/images
$ flex-installer -d /dev/<dev_name> -m myboard \
-f firmware_<machine>_sdboot.img or <machine>-sd-flash.bin \
-b boot_<SOCFAMILY>_arm64_lts_<kernel_ver>.tar.zst \
-r rootfs_<version>_debian_myboard.tar.zst
7. Boot
```

7. Boot up Debian on the board.
   Plug the SD card in the target board and power it on. It automatically boots the Debian system.
   Under U-Boot, if the automated Distro boot is not supported on the board, boot it manually by setting the appropriate U-Boot environment.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

7 Related Documentation

For more information about productions, see the following documentations:

Note: *Obtain the documents UG10163, UG10166, RM00293, UG10159, and UG10165 from the webpage Embedded Linux for i.MX Applications Processors.*

- *i.MX Linux User's Guide* (UG10163)
  Provides information on installing U-Boot and Linux OS and using i.MX-specific features.
- *i.MX Machine Learning User's Guide* (UG10166)
  Provides the machine learning information.
- *i.MX Linux Reference Manual* (RM00293)
  Provides information on Linux drivers for i.MX.
- *i.MX Graphics User's Guide* (UG10159)
  Describes the graphics features.
- *i.MX Porting Guide* (UG10165)
  Provides the instructions on porting the BSP to a new board.
- *i.MX 8M Plus EVK Quick Start Guide* (8MPLUSEVKQSG)
- *i.MX 8M Mini EVK Quick Start Guide* (8MMINIEVKQSG)
- *i.MX 93 EVK Quick Start Guide* (IMX93EVKQSG)
- *Layerscape Debian Linux SDK User Guide* (UG10143)
- i.MX 8 information is at www.nxp.com/imx8.
- i.MX 91 information is at www.nxp.com/imx91.
- i.MX 93 information is at www.nxp.com/imx93.
- i.MX 95 information is at www.nxp.com/imx95.

8 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2026 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

9 Revision History

The following table provides the revision history for this document.

Revision history

Document ID	Release date	Description
UG10155 v.LDLSDK_ 26.06	26 June 2026	Updated the Linux information to be compatible with LF6.12.49_2. 2.0, and added the i.MX 95 19x19 support.
UG10155 v.LDLSDK_ 25.12	26 December 2025	Updated the Linux information to be compatible with LF6.12.20_2.0.0.
UG10155 v.LDLSDK_ 25.06	14 July 2025	Added Section 6.4, Section 6.5, and Section 6.6.
UG10155 v.LDLSDK_ 25.06	27 June 2025	Release for Debian Linux SDK v25.06.
UG10155 v.LDLSDK_ 24.12	19 May 2025	Corrected the typo of the revision number from "IDLSDK_24.12" to "LDLSDK_24.12".
UG10155 v.IDLSDK_ 24.12	30 December 2024	Release for Debian Linux SDK v24.12.
UG10155 v.IDLSDK_ 24.06	16 August 2024	Initial release for Debian Linux SDK v24.06.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

eIQ — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

TensorFlow, the TensorFlow logo and any related marks — are trademarks of Google Inc.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Contents

1 Overview 2
1.1 Features 2
1.1.1 Supported features on i.MX 8 series and i.MX 95 series 2
1.1.2 Supported features on i.MX 93 series 2
1.1.3 Supported features on i.MX 91 series 3
1.1.4 Supported feature matrix on Layerscape platforms 3
1.2 Supported machines 3
2 Quick Start with Debian 4
3 Host Setup 5
3.1 Docker 5
3.2 Host packages 6
4 Image Build 6
4.1 Source code structure 6
4.2 Three output images 7
4.2.1 BSP firmware image 7
4.2.2 Boot partition image 9
4.2.3 Root file system partition image 9
4.3 Overall build pocess 9
4.3.1 Creating the Docker environment 10
4.3.2 Creating the configuration file 11
4.3.2.1 Creating a .config file manually 11
4.3.2.2 Using the default values to create a .config file 14
4.3.2.3 Creating a .config file based on an existing file 14
4.3.3 Building the three images 14
4.4 Detailed build process for individual images 16
4.4.1 Building the BSP composite firmware image 16
4.4.2 Building the boot partition image 18
4.4.3 Building the root filesystem image 19
4.5 Other build targets 20
4.6 Debugging guide 22
4.6.1 LOG_LEVEL - Log Level Control 22
4.6.2 MKDEBUG - Makefile debug mode 22
4.6.3 FBDEBUG - FlexBuild debug mode 23
5 Image deployment 23
5.1 Deploying the pre-built images 24
5.1.1 Deploying on a Linux host machine 24
5.1.1.1 Deploying using the flex-installer tool 24
5.1.1.2 Deploying using the deploy-wizard.sh Tool 26
5.1.2 Deploying on Arm board running TinyLinux 26
5.1.3 Deploying on a Windows host machine 26
5.2 Deploying self-built images 29
5.2.1 Deploying using the flex-installer Tool 29
5.2.2 Deploying the WIC image 30
5.2.3 Other usages of flex-installer 31
5.3 Troubleshooting 33
6 FAQs 34
6.1 How to burn i.MX Debian Linux SDK images generated by Flexbuild to SD card or eMMC device by UUU tool? 34
6.2 How to fix the build issue with log "/usr/bin/apt-key: cannot create /dev/null: Permission denied" when running bld rfs in Flexbuild on certain Linux host? 34
6.3 How to install the Chinese input method package on board? 35
6.4 Where is the source code stored? 35
6.5 How to modify and recompile the source code 35
6.6 How to modify code using patches 36
6.7 How to add or remove a deb package in Flexbuild 37
6.8 How to add a new custom component in Flexbuild 37
6.9 How to add a new board in Flexbuild 37
7 Related Documentation 40
8 Note About the Source Code in the Document 40
9 Revision History 41
Legal information 42

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2026 NXP B.V. **All rights reserved.**