DEEPX
DX-M1 & DX-M1M
Yocto Implementation Guide
for Raspberry Pi 5

Revision: v1.1

Document History

Version	Date	Description	Author
v1.1	26.03.06	Revised Edition	JC Yoo
v1.0	25.12.31	Initial Release	SJ Kim JC Yoo

Copyright © 2025 by DEEPX. All rights reserved.

TERMS OF USE

DEEPX Company Limited (DEEPX) reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to DEEPX's terms and conditions of sale supplied at the time of order acknowledgment.

DEEPX assumes no liability for application assistance or customer product designs. Customers are responsible for their products and applications using DEEPX components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

DEEPX does not warrant or represent that any license, either express or implied, is granted under any DEEPX patent right, copyright, mask work right, or other DEEPX intellectual property right relating to any combination, machine, or process in which DEEPX products or services are used. Information published by DEEPX regarding third-party products or services does not imply any license, warranty, or endorsement by DEEPX. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from DEEPX under the patents or other intellectual property of DEEPX.

Reproduction of DEEPX information in DEEPX data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alterations is an unfair and deceptive business practice. DEEPX is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

DEEPX products are not authorized for use in safety-critical applications (such as life support) where a failure of the DEEPX product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of DEEPX products in such safety-critical applications, notwithstanding any application-related information or support that may be provided by DEEPX. Further, Buyers agree to indemnify DEEPX and its representatives from any claims or damages arising from unauthorized use of DEEPX products in safety-critical applications.

DEEPX products are neither designed nor intended for use in military/aerospace applications or environments unless the DEEPX products are specifically designated by DEEPX as military-grade or "enhanced plastic."

DEEPX products are neither designed nor intended for use in automotive applications or environments unless the specific DEEPX products are designated by DEEPX as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, DEEPX will not be responsible for any failure to meet such requirements.

ⓒ DEEPX Co., Ltd.

All rights reserved. No part of this document may be reproduced in any form without written permission from DEEPX Co., Ltd. DEEPX Co., Ltd. reserves the right to change in its products or product specifications to improve function or design at any time, without notice.

Address : 5F, 20, Pangyoyeok-ro 241beon-gil, Bundang-gu, Seongnam-si, Gyeonggi-do, Republic of Korea

Tel : +82-31-603-9877

Fax : +82-31-709-3731

URL : http://www.deepx.ai

Email : tech_support@deepx.ai

Contents

1. Overview 5

2. Yocto Project Environment Setup and Layer Acquisition 6

2.1. Host System Prerequisites 6

2.2. Cloning Poky and Metadata Layer 6

3. Build Environment Initialization and Image Build 8

3.1. Build Environment Initialization and Layer Registration 8

3.2. Customizing Build Configuration (`local.conf`) 10

3.3. Image Build and Verification 13

4. Advanced System Verificadtion 17

4.1. Automatic USB Storage Mount Verification 17

4.2. Networking and Remote Access Configuration 18

4.3. Infrastructure and Layer Review 20

5. Limitations 22

5.1. Application Support (dx-app) 22

5.2. Firmware Upgrades (dx-fw) 22

5.3. System Monitoring Utility (`dxtop`) 22

6. Release Notes (v2.2.0 / 2026-01-16) 23

Copyright © 2025 by DEEPX. All rights reserved.

1. Overview

The Yocto Project is a collaborative open-source framework dedicated to creating custom Linux distributions for embedded systems. It furnishes the essential templates, tools, and methodologies required to generate optimized, lightweight Linux images tailored to specific hardware architectures and operational requirements. This guide utilizes the **Scarthgap (5.0 LTS)** release to build a high-performance AI-enabled environment for the **DEEPX M1 & DX-M1M NPU**.

Key Architectural Components

- **Custom Build System (BitBake)**
  The core engine that automates the entire workflow—comprising source code fetching, cross-compilation, package generation, and final image construction—based on metadata such as recipes and configuration files.
- **Modular Layering Model**
  Project elements are organized through a modular structure. This ensures a clear separation between the core system (**Poky**), hardware support (**meta-raspberrypi**), and specialized AI acceleration (**meta-deepx-m1**), maximizing maintainability and reusability.
- **Hardware-Specific Optimization**
  The framework provides robust support for specialized hardware features. This implementation focuses on enabling **PCIe Gen 3** performance and **GStreamer (GPU-accelerated multimedia)** specifically for the Raspberry Pi 5 to maximize the efficiency of the DEEPX M1 AI module.

Document Scope and Objective

The primary objective of this guide is to detail the methodology for utilizing the Yocto Project to generate an optimized embedded Linux image targeting the **Raspberry Pi 5** equipped with a **DEEPX M1 AI accelerator**.

Copyright © 2025 by DEEPX. All rights reserved.

2. Yocto Project Environment Setup and Layer Acquisition

This chapter provides the essential procedures for establishing a host development environment and acquiring the metadata layers required to build a custom Linux image for the Raspberry Pi 5.

2.1. Host System Prerequisites

A stable and reproducible build environment requires the installation of specific host-side packages. These dependencies are mandatory for the Yocto Project Bitbake infrastructure and header compilation.

Execute the following commands in the terminal.

```
# Update local package index
sudo apt-get update

# Install mandatory Yocto Project build dependencies
sudo apt-get install gawk wget git diffstat unzip texinfo gcc build-essential \
chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils \
iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint \
xterm python3-subunit mesa-common-dev zstd liblz4-tool lz4
```

NOTE. For a comprehensive list of supported Linux distributions and version-specific requirements, refer to the Official Yocto Project Quick Build Documentation .

2.2. Cloning Poky and Metadata Layer

Project initiation involves acquiring **Poky** (the reference build system) alongside the essential hardware support layers (BSP) and the OpenEmbedded ecosystem layers.

Step 1. Initialize Workspace

Create the designated project directory and navigate into it.

```
mkdir ~/yocto
cd ~/yocto
```

Step 2. Acquire Poky (Reference Build System)

Copyright © 2025 by DEEPX. All rights reserved.

Clone the **Scarthgap** branch. Scarthgap is the current LTS(Long Term Support) release of the Yocto Project, ensuring stability and long-term security updates.

```
# Clone the Poky reference repository
git clone -b scarthgap git://git.yoctoproject.org/poky.git
```

Step 3. Acquire Hardware and Accessory Layers

To support the Raspberry Pi 5 hardware and extended software packages, clone the `meta-raspberrypi` BSP layer and the `meta-openembedded` collection.

```
# Ensure you are in the ~/yocto root director
cd ~/yocto

# Clone the OpenEmbedded metadata collection
git clone -b scarthgap git://git.openembedded.org/meta-openembedded

# Clone the Raspberry Pi Board Support Package (BSP) layer
git clone -b scarthgap https://github.com/agherzan/meta-raspberrypi.git
```

Step 4: Verification of Source Acquisition

Upon completion, the `~/yocto` directory should contain the following three primary metadata repositories:

1. `poky/`: The core build engine, Bitbake, and base metadata.
2. `meta-openembedded/`: A collection of sub-layers (`meta-oe, meta-python,` etc.) for extended utility support.
3. `meta-raspberrypi/`: The specific BSP required for Raspberry Pi 5 kernel and bootloader configurations.

The environment is now prepared for layer configuration and image customization.

Copyright © 2025 by DEEPX. All rights reserved.

3. Build Environment Initialization and Image Build

This chapter details the systematic process of initializing the BitBake build environment and configuring hardware-specific metadata to generate an optimized Linux image for the Raspberry Pi 5.

3.1. Build Environment Initialization and Layer Registration

Initializing the BitBake shell environment and registering metadata layers are critical steps to ensure the build system recognizes all hardware-specific recipes and AI acceleration components.

Step 1. Initialize the Build Environment

The initialization script within the `poky` directory must be executed to set up essential environment variables and create (or navigate to) the designated build directory.

```
# Navigate to the project root
cd ~/yocto


# Initialize the BitBake build environment
source poky/oe-init-build-env
```

IMPORTANT. Path Change

Upon successful execution of the `oe-init-build-env` script, your current working directory will automatically change to `~/yocto/build`. All subsequent Bitbake commands must be executed from this location.

Step 2. Register Layer (Base Layers & GStreamer)

The `bitbake-layers add-layer` command is used to register the previously downloaded layers into the build configuration file (`build/conf/bblayers.conf`).

Execute the following commands from the `~/yocto/build` directory.

```
# 1. Register OpenEmbedded base layers
bitbake-layers add-layer ../meta-openembedded/meta-oe
bitbake-layers add-layer ../meta-openembedded/meta-python
bitbake-layers add-layer ../meta-openembedded/meta-networking
bitbake-layers add-layer ../meta-openembedded/meta-multimedia
```

Copyright © 2025 by DEEPX. All rights reserved.

```
# 2. Register Raspberry Pi BSP
bitbake-layers add-layer ../meta-raspberrypi

# 3. Register Multimedia layer for GStreamer support
bitbake-layers add-layer ../meta-openembedded/meta-filesystems
bitbake-layers add-layer ../meta-deepx-m1
```

NOTE. The `meta-deepx-m1` layer will be acquired and registered in the following step to ensure all base dependencies are satisfied first.

Step 3. DEEPX Layer Cloning and Adding

In this step, you will clone the dedicated layer for DEEPX AI accelerator support and add it into the build system.

3-1. Clone the DEEPX Repository

Return to the project root to clone the AI acceleration metadata.

```
# Navigate to project root
cd ~/yocto

# Clone the meta-deepx-m1 layer using the stable scarthgap branch
git clone -b scarthgap https://github.com/DEEPX-AI/meta-deepx-m1.git
```

3-2. Integrate the DEEPX Layer

Once the source is acquired, register the layer to the active build configuration

```
# Return to the build directory
cd ~/yocto/build

# Register the DEEPX layer to the build configuration
bitbake-layers add-layer ../meta-deepx-m1
```

Copyright © 2025 by DEEPX. All rights reserved.

3.2. Customizing Build Configuration (`local.conf`)

Optimizing the graphics stack and integrating the DEEPX AI acceleration runtime is managed through specific modifications within the `local.conf` configuration file. This file acts as the primary instruction set for BitBake to define the target architecture and image features.

Step 1. Target Machine and Graphics Configuration

This step specifies the target hardware and configures the graphics stack to ensure the Weston GUI environment utilizes hardware acceleration on the Raspberry Pi 5.

Modify `~/yocto/build/conf/local.conf` as follows.

```
# ------------------------------------------------------------------
# Target Machine and Graphics Configuration
# ------------------------------------------------------------------

# Specify the Target Machine
MACHINE ??= "raspberrypi5"

# Essential Graphics Configuration for core-image-weston
PACKAGECONFIG:append:pn-mesa = " gallium v3d vc4 gles2 egl wayland"
PACKAGECONFIG:remove:pn-mesa = " x11 glx"

# Enable Wayland and OpenGL while disabling legacy X11 features
# This configuration prevents XWayland from attempting to utilize GLX
DISTRO_FEATURES:append = " wayland opengl pam"
DISTRO_FEATURES:remove = " x11 vulkan directfb"
```

Step 2. DX-RT Integration and Hardware Optimization

This step integrates the DEEPX runtime packages into the image and applies critical performance optimizations for the Raspberry Pi 5's PCIe interface.

Modify `~/yocto/build/conf/local.conf` as follows.

```
# ------------------------------------------------------------------
# DEEPX Runtime (DX-RT) and Application Integration
```

Copyright © 2025 by DEEPX. All rights reserved.

```
# ------------------------------------------------------------


# 1. DEEPX Driver and Runtime Installation
# Include kernel drivers, runtime SDK, and GStreamer streaming elements
IMAGE_INSTALL:append = " dx-driver dx-rt dx-stream dx-stream-sample"
IMAGE_INSTALL:append = " libyuv libonnxruntime"


# 2. Python Runtime and Essential Modules (Critical)
# 'python3-modules' includes the full standard library, and 'python3-numpy' is required for handling AI model I/O tensors.
IMAGE_INSTALL:append = " python3 python3-modules python3-numpy"


# 3. Hardware Optimization and license agreement
IMAGE_FSTYPES:append = " wic.xz"
LICENSE_FLAGS_ACCEPTED += "synaptics-killswitch"


# PCIe performance optimization (disable ASPM and enable Gen 3)
CMDLINE:append = " pcie_aspm=off"
RPI_EXTRA_CONFIG:append = "\ndtparam=pciex1_gen=3\n"
```

IMPORTANT. Python Environment Requirements

To execute the provided `run_model.py` and other sample scripts without `"ModuleNotFoundError"` errors, the image must include the full `python3-modules` stack and the `numpy` package. The configuration above ensures all dependencies are pre-installed in the target rootfs.

Step 3. Configure System Utilities and Initialization Manager (`systemd`)

Integrate `systemd` configuration for USB auto-mounting, remote access (SSH), and modern service management.

```
# ------------------------------------------------------------
# Advanced System Utilities (USB, Networking, systemd)
# ------------------------------------------------------------


# 1. USB Auto-Mounting and Filesystem Support
```

Copyright © 2025 by DEEPX. All rights reserved.

```
IMAGE_INSTALL:append = " usbutils kernel-modules udev-extraconf"
IMAGE_INSTALL:append = " dosfstools mtools exfat-utils fuse-exfat ntfs-3g"


# 2. Remote Access and Networking Tools (SSH and iproute2)
IMAGE_INSTALL:append = " openssh openssh-sftp-server iproute2"
IMAGE_FEATURES += "ssh-server-openssh empty-root-password"


# 3. Init Manager Configuration (systemd Integration)
# usrmerge is required when using systemd in the latest Yocto versions.
DISTRO_FEATURES:append = "usrmerge systemd"
VIRTUAL-RUNTIME_init_manager="systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED += "sysvinit"
VIRTUAL-RUNTIME_initscripts="systemd-compat-units"
```

Step 4. GStreamer and Multimedia Acceleration

To enable high-performance video processing for AI inference, **GStreamer** must be configured with hardware acceleration and commercial license support.

```
# ------------------------------------------------------------------
# GStreamer Commercial License and Hardware Acceleration
# ------------------------------------------------------------------

# Allow building of commercial/restricted plugins
LICENSE_FLAGS_ACCEPTED += "commercial"

# Core GStreamer packages and base tools
IMAGE_INSTALL:append = " gstreamer1.0 gstreamer1.0-plugins-base gstreamer1.0-
plugins-good"

# Essential package for H.264 software decoding (Required for RPi 5)
IMAGE_INSTALL:append = " gstreamer1.0-libav"

# Optional plugins for H.265 HW acceleration and V4L2 features
IMAGE_INSTALL:append = " gstreamer1.0-plugins-bad"
```

Copyright © 2025 by DEEPX. All rights reserved.

```
# Hardware acceleration support via OpenMAX/V4L2 for Raspberry Pi
IMAGE_INSTALL:append = " gstreamer1.0-omx"

# Optimize GStreamer Graphics Backend for Wayland/Weston
# Enables EGL/GLES2 while removing unnecessary X11/GLX dependencies
PACKAGECONFIG:append:pn-gstreamer1.0-plugins-base = " egl gles2 wayland"
PACKAGECONFIG:remove:pn-gstreamer1.0-plugins-base = " x11 glx opengl"
```

3.3. Image Build and Verification

Executing the build process involves running BitBake commands, deploying the resulting image to an SD card, and performing functional verification on the target hardware.

Step 1. Image Build Execution

The final image is generated using the Bitbake command. This process orchestrates the compilation of all layers and recipes defined in your metadata.

1-1. Run the Build Command

Select the image type that fits your requirements. For optimal GStreamer performance and DEEPX integration, the **Weston-based GUI image** is recommended.

```
# Option A: GUI Image (Recommended)
# Provides optimal GStreamer and hardware acceleration
bitbake core-image-weston

# Option B: Console Image
# Use if a GUI is not required
bitbake core-image-minimal
```

NOTE. Error Recovery

If a build error occurs and you need to clean the existing data before rebuilding, use the `-c cleansstate` option.

Copyright © 2025 by DEEPX. All rights reserved.

```
bitbake core-image-weston -c cleansstate
```

CAUTION. Build Duration

The initial build process may take anywhere from 30 minutes to several hours, depending on your host PC's hardware specifications.

1-2. (Optional) Resource Optimization for Low-Spec Hosts

If your host PC lacks sufficient RAM or experiences freezes, add the following lines to your `local.conf` to limit the simultaneous build tasks.

```
BB_NUMBER_THREADS = "8"
PARALLEL_MAKE = "-j 8"
```

Step 2. Image Flashing (Writing to SD Card)

Once the build is complete, the generated `.wic.xz` image file must be written to a physical SD card. This process uses a streamlined command to decompress and write the image simultaneously, saving time and disk space on your host system.

2-1. Prepare and Write the Image File

The build artifacts are located in: `~/yocto/build/tmp/deploy/images/raspberrypi5/`

```
# 1. Navigate to the image deployment directory
cd ~/yocto/build/tmp/deploy/images/raspberrypi5/

# 2. Verify the file list to confirm the presence of the XZ compressed image
ls core-image-weston-raspberrypi5*.wic.xz

# 3. Write to SD card (Optimized Method)
# This command decompresses the .xz file on-the-fly and writes it directly to the SD card.
# REPLACE /dev/sdX with your actual SD card device name (e.g., /dev/sdb).
xz -dc ./core-image-weston-raspberrypi5-raspberrypi5.rootfs.wic.xz | sudo dd of=/dev/sdX bs=4M status=progress
```

CAUTION. Data Loss Warning

Copyright © 2025 by DEEPX. All rights reserved.

Specifying the wrong path for `of=/dev/sdX` (e.g., your host's main drive) will lead to **permanent data loss**. Always use the `lsblk` command to identify the correct device name for your SD card before proceeding.

Step 3. Target Board Verification

Insert the flashed SD card into your Raspberry Pi 5, boot the system, and perform tests to verify the **DX-RT** and **GStreamer** functionality.

3-1. Gstreamer Framework Verification

Verify that the GStreamer framework and its required versions are correctly installed.

```
gst-launch-1.0 --version
```

3-2. Basic Video Output Check

Generate a standard test pattern to confirm the video pipeline and graphics drivers are operational.

```
# Run GStreamer video test pattern
gst-launch-1.0 videotestsrc ! videoconvert ! autovideosink
```

3-3. Hardware Acceleration Test (DX-STREAM)

Test the hardware acceleration capabilities using the DEEPX-specific streamer elements.

```
# Navigate to the DX-STREAM sample directory and run the test script
cd /etc/dx-stream-sample
./run.sh
```

NOTE. For detailed information on `dx-stream` elements, please refer to the **"DXNN Streamer (DX-STREAM) User Manual."**

3-4. Python Environment and DX-RT Verification

Verify the Python runtime environment and **DX-RT (DEEPX Runtime)** can successfully load and execute a model on the target hardware.

```
# Navigate to the DX-RT utility directory
cd /etc/dx-rt
```

Copyright © 2025 by DEEPX. All rights reserved.

```
# Execute the inference test script with a sample DXNN model

python3 run_model.py -m ../dx-stream-sample/YOLO5S_PPU.dxnn
```

NOTE. Successful execution will output the inference results (e.g., detection coordinates or classification labels) to the terminal. This confirms that the Python environment and **DX-RT** libraries are fully operational on the Raspberry Pi 5.

Copyright © 2025 by DEEPX. All rights reserved.

4. Advanced System Verificadtion

This chapter focuses on verifying the advanced utility configurations integrated during the build process. Since all core configurations for USB, networking, and system management were defined in **Chapter 3.2**, this section provides functional tests to ensure the target hardware is operating as intended.

4.1. Automatic USB Storage Mount Verification

The integration of `udev-extraconf` and extended filesystem support (FAT32, NTFS, exFAT) enhances the operational usability of the Weston GUI environment by allowing for seamless, plug-and-play data transfer.

Step 1. Hardware Interaction

To verify the automatic mounting functionality, you must interact with the live Raspberry Pi 5 system using a physical USB storage device.

1. **Insert USB Drive:** Plug a USB flash drive or external hard drive (formatted in FAT32, NTFS, or exFAT) into one of the Raspberry Pi 5 USB ports.
2. **System Detection:** The `udev` daemon will detect the new hardware and trigger the `udev-extraconf` rules to mount the partitions automatically.

Step 2. Verify Mount Point on Target Board

Navigate to the unified media directory to confirm that the system has correctly identified and mounted the storage device.

```
# Verify the automatic mount path on the Raspberry Pi 5 terminal
ls /run/media/
```

NOTE. Success Criteria

A successful integration will display a subdirectory under `/run/media/` named after the device partition (e.g., `sda1` or `sdb1`).

Step 3. File Access and Driver Integrity Test (Optional)

Beyond simple mount verification, listing the files within the drive ensures that the filesystem drivers (such as `ntfs-3g` or `fuse-exfat`) are correctly interacting with the Linux kernel to parse the data structure.

Verification Command

Copyright © 2025 by DEEPX. All rights reserved.

Execute the following command to list the contents of your USB drive. This confirms that the partition is not only mounted but also accessible at the file level.

```
# Replace 'sda1' with your actual mount point identified in Step 2
ls -l /run/media/sda1/
```

Expected Result:

- A detailed list of files and directories stored on the USB drive should be displayed.
- If the command returns an error or an empty list despite the drive having data, it indicates a potential issue with the corresponding filesystem driver or mount permissions.

NOTE. Why these matters

Successful execution of `ls -l` confirms that the NPU application will be able to access AI models or datasets stored on external media without encountering "Permission Denied" or "Unknown Filesystem" errors during runtime.

4.2. Networking and Remote Access Configuration

With **systemd** managing services and **OpenSSH** integrated during the build, the Raspberry Pi 5 is optimized for secure remote management and advanced network diagnostics.

Step 1. Network Connectivity Check

Verify that the system has successfully initialized the network interface and acquired an IP address using the `iproute2` utilities.

```
# Display the status of all network interfaces
ip addr show
```

NOTE. Ensure the ethernet cable is connected or the Wi-Fi is configured. The output should show an assigned IP address for `eth0` or `wlan0`.

Step 2. Remote Management via SSH

Test the OpenSSH server by connecting to the target board from your host PC. This confirms that the `ssh-server-openssh` feature is active.

```
# Connect from your host PC (Replace <target_ip> with the board's actual IP)
```

Copyright © 2025 by DEEPX. All rights reserved.

```
ssh root@<target_ip_address>
```

IMPORTANT. Security Warning

Since the `empty-root-password` feature was enabled in `local.conf` for development, you will not be prompted for a password. Ensure this feature is removed before moving to a production environment.

Step 3. Init Manager Verification (systemd)

Confirm that `systemd` is serving as the primary init manager and that the `usrmerge` file hierarchy is correctly implemented. This ensures the system is capable of modern parallel service management.

3-1. Verify the Global Init Process (PID 1)

In Linux, the first process started by the kernel is assigned **Process ID 1 (PID 1).** By checking which executable is mapped to this ID, you can confirm the active init manager.

Execution Command

```
# Check the command name associated with Process ID 1
ps -p 1 -o comm=
```

Expected Result: `systemd`

NOTE. If the output shows init, the system is still utilizing the legacy `SysVinit`. A successful migration will explicitly return `systemd`, confirming that the configuration in **Chapter 3.2** was applied correctly.

3-2. Verify Service Management Functionality

Test if the `systemctl` utility is operational by checking the status of the SSH daemon.

```
# Monitor the status of the SSH daemon
systemctl status ssh
```

3-3. Verify usrmerge Compatibility

Recent Yocto releases require the **merged-usr** design. Verify that the legacy `/bin` directory is now a symbolic link to `/usr/bin`.

```
# Check the symbolic link for the /bin directory
```

Copyright © 2025 by DEEPX. All rights reserved.

```
ls -ld /bin
```

Expected Result: `lrwxrwxrwx ... /bin -> usr/bin`

4.3. Infrastructure and Layer Review

A standardized workspace and correct layer sequencing are critical for long-term project maintenance, reproducibility, and ensuring that no dependency conflicts occurred during the build process.

Step 1. Verify Active Metadata Layers

Confirm that BitBake is correctly parsing all required metadata. This ensures that the Hardware BSP and DEEPX AI acceleration layers are active and prioritized correctly.

```
# Display the prioritized list of active layers
# Run this from the ~/yocto/build directory
bitbake-layers show-layers
```

Step 2. Validate `bblayers.conf` Baseline

Check that your Host System Prerequisites use **absolute paths** and follows the validated sequence. In the Yocto Project, layers at the bottom of the list have higher priority; therefore, meta-deepx-m1 should be at the end to ensure AI configurations are applied correctly.

Validated `bblayers.conf` Example:

```
BBLAYERS ?= " \
  /home/deepx/yocto/poky/meta \
  /home/deepx/yocto/poky/meta-poky \
  /home/deepx/yocto/poky/meta-yocto-bsp \
  /home/deepx/yocto/meta-raspberrypi \
  /home/deepx/yocto/meta-openembedded/meta-oe \
  /home/deepx/yocto/meta-openembedded/meta-python \
  /home/deepx/yocto/meta-openembedded/meta-networking \
  /home/deepx/yocto/meta-openembedded/meta-multimedia \
  /home/deepx/yocto/meta-openembedded/meta-filesystems \
  /home/deepx/yocto/meta-deepx-m1 \
```

Copyright © 2025 by DEEPX. All rights reserved.

```
"
```

CAUTION. Absolute Paths Requirement

Always use absolute paths in `BBLAYERS`. Relative paths (e.g., `../meta-layer`) are a primary cause of build failures when the environment is re-initialized or accessed from different shell contexts.

Step 3. Standardized Workspace Verification

Verify the physical directory hierarchy. A unified root (`~/yocto`) containing Poky, BSP, and AI layers is essential for collaborative development and path consistency.

Standardized Directory Hierarchy:

- `poky/:` Core build system (Scarthgap LTS).
- `meta-openembedded/:` Extended software metadata (Python, Networking, etc.).
- `meta-raspberrypi/:` Hardware support for Raspberry Pi 5.
- `meta-deepx-m1/:` DEEPX M1 NPU acceleration metadata.
- `build/:` Active build directory containing local.conf and output images.

Programmatic Check

```
# From the project root (~/yocto), verify the essential subdirectories
cd ~/yocto ls -d
poky meta-openembedded meta-raspberrypi meta-deepx-m1 build
```

NOTE. If any directory is missing or misplaced, BitBake will fail to resolve the recipes. If your tree does not match this structure, re-verify the **Repository Cloning** steps in **Section 2**.

Copyright © 2025 by DEEPX. All rights reserved.

5. Limitations

This section outlines the functional constraints and hardware-specific requirements for the current release of the DEEPX Yocto integration. Understanding these limitations is essential for planning development cycles and deployment strategies.

5.1. Application Support (dx-app)

`dx-app` is **not included** in this release and is scheduled for integration in the **next release**.

NOTE. For detailed information on `dx-app` elements, please refer to the **"DXNN Application (DX-APP) User Manual."**

5.2. Firmware Upgrades (dx-fw)

The **M1** and **M1M** modules are updated using the **DX-RT CLI** utility.

However, firmware binaries are **not** distributed publicly. To obtain the appropriate firmware for your specific module revision, please contact a **DEEPX FAE** or an **authorized DEEPX distributor**.

5.3. System Monitoring Utility (`dxtop`)

The `dxtop` real-time NPU monitoring utility is included in this release.

[Known Issue] `dxtop` Execution Error in systemd Environment

- **Symptom:** When running the `dxtop` application via the terminal on a target board configured with `systemd`, the program fails to launch and displays the following error message
  Error opening terminal: `xterm`
- **Cause:** This issue occurs because the `systemd` environment in this Yocto build does not automatically map the default path for the **terminfo database**. Without these terminal environment definitions, ncurses-based applications like `dxtop` cannot initialize the display.
- **Solution:** You can resolve this issue by explicitly setting the `TERMINFO` environment variable to point to the correct path before launching the utility.

Corrective Commands

Enter the following command in the terminal to define the environment variable and launch `dxtop`.

```
# Set the terminal info path and run dxtop
export TERMINFO=/etc/terminfo
dxtop
```

Copyright © 2025 by DEEPX. All rights reserved.

6. Release Notes (v2.2.0 / 2026-01-16)

The following components represent the validated software stack for this release.

(https://github.com/DEEPX-AI/dx-runtime/blob/deba09625ce8b7de8a74c65a5ab8e4089611790a/RELEASE_NOTES.md#npu-driver-v200--210)

DX-Runtime (v2.2.0 / 2026-01-16)

- **DX_FW:** v2.5.0
- **NPU Driver:** v2.1.0
- **DX-RT:** v3.2.0
- **DX-Stream:** v2.2.0
- **DX-APP:** v3.0.0

DX_FW (v2.5.0)

1. Changed

- PCIe Stability: Added PERST# signal wait during initial boot stage.
- Thermal Priority: Changed to Emergency > User > Default.
- Inference Time: Updated inf_time to include both NPU and PPCPU runtimes.

2. Fixed

3. Added

- NPU QoS: Added Quality of Service to the NPU Scheduler.
- Flash Support: Added support for nand 2k flash images.

NPU Driver (v2.0.0 ~ 2.1.0)

1. Changed

- System Stability: Added sleep/reschedule in polling to prevent soft lockups during slow ACKs.
- Performance: Optimized PCIe DMA by reducing CPU dependency (Best with DX-RT SDK v3.2.0+).
- Initialization: Added request ID (req-id) initialization for V3.

2. Fixed

3. Added

Copyright © 2025 by DEEPX. All rights reserved.

DX-RT (v3.2.0)

1. Changed

- Optimization: Improved PCIe DMA sequence and updated Debian OS requirements in the docs.

2. Fixed

- Memory: Reduced device memory footprint for PPU models.

3. Added

- Event Handling: Added RuntimeEventDispatcher (C++ singleton & Python wrapper) for centralized system events.
- CLI Options: Added --profiler and --buffer-count to run_model.py.
- Build Script: Added options to toggle Service and ORT components (--use_service_on/off, --use_ort_on/off).
- Engine Features: Enabled direct .dxnn loading from memory and per-instance I/O buffer configuration.
- Metadata: Added __version__ to the dx_engine namespace.

DX-Stream (v2.2.0)

1. Changed

- Buffer Handling: Updated gst-dxpreprocess to auto-copy buffers when ref_count > 1.
- Metadata Architecture: Modularized metadata files (frame/object/user) and enhanced structures.
- Test & Docs: Refactored test framework for better performance and updated Debian OS requirements.

2. Fixed

- Stability: Resolved race conditions and segfaults in secondary inference mode.
- API/Errors: Migrated to new object metadata APIs and improved compilation error handling in tests.

3. Added

- Python Bindings: Introduced pydxs module with support for DXFrameMeta, DXObjectMeta, and DXUserMeta.
- User Metadata: New API/interface for custom metadata attachment and management.
- Tools & Samples: Added writable_buffer context manager and a new usermeta_app.py sample.
- Testing: Added a dedicated user metadata test suite (test_usermeta.cpp).

Copyright © 2025 by DEEPX. All rights reserved.