MaaXBoard Mini Yocto Software Development Guide V2.0

Revision History

Ver.	Note	Author	Release Date
V1.0	Initial version	Paul/Sandy	20200904
V1.1	Updated Yocto 3.0	Nick	20210322
V1.2	Updated Zeus branch for meta-maaxboard	Nick	20211014
V1.3	update the Build Configure	Nick	20211203
V2.0	Updated Yocto to Langdale(4.1), BSP_VERSION to lf-6.1.1-1.0.0, Converts the file format to markdown	Lily	20230712

Catalog

Chapter 1 Build with Yocto

1.1 Setup Build Environment

To setup the build environment need:

- Hardware: At least 300GB of disk space and 8GB of RAM
- Software: Ubuntu 64bit OS, 18.04 LTS version or later LTS version (Ubuntu Desktop or Ubuntu Server version). You could also run the Ubuntu 64 bit OS on virtual machine.

The following packages are required for the development environment. The required packages can be installed using the bash script below:

```
$ sudo apt-get update
$ sudo apt-get install -y \
wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa
libsdl1.2-dev \
pylint3 xterm rsync curl gawk zstd lz4 locales bash-completion
```

Install repo

```
$ mkdir -p ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=~/bin:$PATH
```

Set Git configuration:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "you@example.com"
```

1.2 Fetch Source Code

1.2.1 Download meta layers from NXP

```
$ mkdir ~/imx-yocto-bsp
$ cd ~/imx-yocto-bsp
$ repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-langdale -m
imx-6.1.1-1.0.0.xml
$ repo sync
```

1.2.2 Download MaaXBoard Mini Source Code

To get the source code of MaaXBoard serial boards, download the repository form GitHub:

```
$ cd ~/imx-yocto-bsp/sources
$ git clone https://github.com/Avnet/meta-maaxboard.git -b langdale meta-
maaxboard
```

1.3 Build

1.3.1 Build Configuration

Configure using script

MaaXBoard Mini provides a script, ***maaxboard-setup.sh***, that simplifies the setup for MaaXBoard serial boards. If you want to create a new build folder or set the configuration for the first time, run the command:

```
$ cd ~/imx-yocto-bsp
$ MACHINE=maaxboard-mini source sources/meta-maaxboard/tools/maaxboard-setup.sh -
b maaxboard-mini/build
```

If you want to build in an existing build folder, use the following command:

```
$ cd ~/imx-yocto-bsp
$ source sources/poky/oe-init-build-env maaxboard-mini/build
```

1.3.2 Build

Execute the following command to build a Weston Wayland image:

```
$ bitbake avnet-image-full
```

After the build has successfully completed, the output files are deployed in:

~/imx-yocto-bsp/maaxboard-mini/build/tmp/deploy/images/maaxboard-mini/

imx-boot	Bootloader Image
avnet-image-full-maaxboard-mini-xxxx.rootfs.wic	System image, this includes: Linux kernel, DTB and root file system.
Image	Kernel image
maaxboard-mini.dtb	MaaXBoard Mini device tree binary
overlays	MaaXBoard Mini device tree overlay binary
avnet-image-full-maaxboard-mini-xxxx.rootfs.tar.zst	System image compressed archive file

Chapter 2 Standalone Build of u-Boot and Kernel

This chapter describes how to build U-boot and Kernel using SDK or ARM GCC in a standalone environment.

2.1 Cross-compile tool chain

The cross-compile tool chain could use Yocto SDK or ARM GCC.

2.1.1 ARM GCC

Download the tool chain for the A-profile architecture on arm Developer GNU-A Downloads page. It is recommended to use the 10.3 version for this release. You can download the "gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu.tar.xz", and then unzip and install it into a directory.

```
$ mkdir toolchain
$ tar -xJf gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu.tar.xz -C
./toolchain
```

Use the following command to check that the toolchain can be directly run.

```
$ cd toolchain/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/
$ ./aarch64-none-linux-gnu-gcc -v
```

When using ARM GCC to compile a project, first execute the following command to configure environment variables:

```
$ TOOLCHAIN_PATH=$HOME/toolchain/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-
gnu/bin
$ export PATH=$TOOLCHAIN_PATH:$PATH
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-none-linux-gnu-
```

2.1.2 Yocto SDK

Generate an SDK from the Yocto Project build environment with the following command after generating the image in the previous chapter.

```
$ bitbake avnet-image-full -c populate_sdk
$ ls tmp/deploy/sdk/
```

The generated file is ***fsl-imx-wayland-lite-glibc-x86_64-avnet-image-full-armv8a-maaxboard-mini-toolchain-6.1-langdale.sh***, and execute this script to install the SDK. The default location is /opt but can be placed anywhere on the host machine.

```
$ sudo ./imx-yocto-bsp/maaxboard-mini/build/tmp/deploy/sdk/fsl-imx-wayland-lite-glibc-x86_64-avnet-image-full-armv8a-maaxboard-mini-toolchain-6.1-langdale.sh
NXP i.MX Release Distro SDK installer version 6.1-langdale
===========================================================
Enter target directory for SDK (default: /opt/fsl-imx-wayland-lite/6.1-langdale):
The directory "/opt/fsl-imx-wayland-lite/6.1-langdale" already contains a SDK for this architecture.
If you continue, existing files will be overwritten! Proceed [y/N]? y
Extracting SDK......done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.

```
 $ . /opt/fsl-imx-wayland-lite/6.1-langdale/environment-setup-armv8a-poky-linux
```

2.2 Build U-Boot in a standalone environment

2.2.1 Get the source code and firmware

To get the source code of u-boot, imx-atf and imx-mkimage, execute the following command:

```
$ mkdir tmp
$ cd tmp
$ git clone https://github.com/Avnet/uboot-imx.git -b maaxboard_lf-6.1.1-1.0.0
$ git clone https://github.com/Avnet/imx-atf.git -b maaxboard_lf-6.1.1-1.0.0
$ git clone https://github.com/Avnet/imx-mkimage.git -b maaxboard_lf-6.1.1-1.0.0
```

Download the firmware-imx, decompress and accept NXP EULA when running

```
$ wget https://www.nxp.com.cn/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.18.bin
$ chmod +x firmware-imx-8.18.bin
$ ./firmware-imx-8.18.bin
```

Execute the 'ls' command to view the tmp directory:

```
$ ls tmp
```

firmware-imx-8.18 firmware-imx-8.18.bin imx-atf imx-mkimage uboot-imx

So far, the required source code and firmware have been prepared.

2.2.2 Compile script

Create a bash script in the tmp directory and change the file mode:

```
$ cd tmp
$ touch make_mx8m_uboot.sh
$ chmod 766 make_mx8m_uboot.sh
```

Copy the following content into the make_mx8m_uboot.sh script:

```
#!/bin/bash
RM_GCC_VERSION=10.3
if [ "${ARM_GCC_VERSION}" == "SDK" ] ; then
        source /opt/fsl-imx-wayland-lite/5.4-zeus/environment-setup-aarch64-poky-
linux
else
        ## gcc 10.3 default
        TOOLCHAIN_PATH=$HOME/toolchain/gcc-arm-10.3-2021.07-x86_64-aarch64-none-
linux-gnu/bin
        export PATH=$TOOLCHAIN_PATH:$PATH
        export ARCH=arm64
        export CROSS_COMPILE=aarch64-none-linux-gnu-
fi

# check required applications are installed
command -v dtc >/dev/null 2>&1 || { echo "ERROR: Cannot find dtc, run 'sudo apt
install device-tree-compiler' please"; exit; }
command -v ${CROSS_COMPILE}gcc >/dev/null 2>&1 || { cat << EEOF
ERROR: ${CROSS_COMPILE}gcc not found,
please install the toolchain first
and export the enviroment like "export PATH=\$PATH:your_toolchain_path"
EEOF
exit;}
help() {
        bn=`basename $0`
        cat << EOF
usage : $bn <option>
options:
-h display this help and exit
-mx8m build u-boot.imx for the MaaXBoard baord
-mini build u-boot.imx for the MaaXBoard mini baord
-nano build u-boot.imx for the MaaXBoard nano baord
-clean clean the build files for all projects
Example:
./$bn -mx8m
./$bn -mini
./$bn -nano
EOF
}

SOC_TYPE="mx8m"
WORKPWD=$(pwd)
mk_clean()
```

```
{
        cd $WORKPWD
        make clean -C imx-atf/
        make clean -C imx-mkimage/
        (cd imx-mkimage/ && git clean -f -d)
        make distclean -C uboot-imx/
        rm u-boot*.imx
}

[ $# -eq 0 ] && help && exit
while [ $# -gt 0 ]; do
        case $1 in
                -h) help; exit ;;
                -mx8m) echo ${SOC_TYPE};;
                -mini) SOC_TYPE="mx8m_mini"; echo ${SOC_TYPE};;
                -nano) SOC_TYPE="mx8m_nano"; echo ${SOC_TYPE};;
                -cl*) mk_clean ; exit ;;
                *) echo "-- invalid option -- "; help; exit;;
        esac
        shift
done

mk_uboot()
{
        cd uboot-imx/
        if [ "${SOC_TYPE}" == "mx8m_mini" ] ; then
                make maaxboard-mini_defconfig
        elif [ "${SOC_TYPE}" == "mx8m_nano" ] ; then
                make maaxboard-nano_defconfig
        else
                make maaxboard_defconfig
        fi
        make -j4
        [ $? -ne 0 ] && echo "Failed in uboot-imx ..." && exit
        cd $WORKPWD
}

IMX_FW_NAME="firmware-imx-8.18"
mk_firmware()
{
        cd $WORKPWD
        [ -e ${IMX_FW_NAME}/firmware/sdma/sdma-imx7d.bin ] && return
        echo unpack ${IMX_FW_NAME}.bin
        ./${IMX_FW_NAME}.bin
}

mk_atf()
{
        cd imx-atf/
        case ${SOC_TYPE} in
                mx8m) echo "build atf for imx8mq"; make PLAT=imx8mq bl31;;
                mx8m_mini) echo "build atf for imx8mm"; make PLAT=imx8mm bl31;;
                mx8m_nano) echo "build atf for imx8mn"; make PLAT=imx8mn bl31;;
        esac
        [ $? -ne 0 ] && echo "Failed in imx-atf ..." && exit
        cd $WORKPWD
}
```

```
mk_imxboot()
{
        cp ${IMX_FW_NAME}/firmware/ddr/synopsys/ddr4*.bin imx-mkimage/iMX8M
        cp uboot-imx/tools/mkimage imx-mkimage/iMX8M/mkimage_uboot
        cp uboot-imx/u-boot-nodtb.bin imx-mkimage/iMX8M/
        cp uboot-imx/spl/u-boot-spl.bin imx-mkimage/iMX8M/
        if [ "${SOC_TYPE}" == "mx8m_mini" ] ; then
                cp imx-atf/build/imx8mm/release/bl31.bin imx-
mkimage/iMX8M/bl31.bin
                cp uboot-imx/arch/arm/dts/maaxboard-mini.dtb imx-
mkimage/iMX8M/imx8mm-ddr4-evk.dtb
                cd imx-mkimage/
                make SOC=iMX8MM flash_ddr4_evk
        elif [ "${SOC_TYPE}" == "mx8m_nano" ] ; then
                cp imx-atf/build/imx8mn/release/bl31.bin imx-
mkimage/iMX8M/bl31.bin
                cp uboot-imx/arch/arm/dts/maaxboard-nano-mipi.dtb imx-
mkimage/iMX8M/imx8mn-ddr4-evk.dtb
                cd imx-mkimage/
                make SOC=iMX8MN flash_ddr4_evk
        else
                cp ${IMX_FW_NAME}/firmware/hdmi/cadence/signed_hdmi_imx8m.bin
imx-mkimage/iMX8M
                cp imx-atf/build/imx8mq/release/bl31.bin imx-
mkimage/iMX8M/bl31.bin
                cp uboot-imx/arch/arm/dts/maaxboard.dtb imx-mkimage/iMX8M/imx8mq-
ddr4-val.dtb
                cd imx-mkimage/
                make SOC=iMX8M flash_ddr4_val
        fi
        [ $? -ne 0 ] && echo "Failed in imx-mkimage ..." && exit
        cd $WORKPWD
}

## main loop ###
cd $WORKPWD
mk_uboot
mk_firmware
mk_atf
mk_imxboot
cp -f imx-mkimage/iMX8M/flash.bin ./u-boot-${SOC_TYPE}.imx
echo ""
echo "---Finished--- the boot image is u-boot-${SOC_TYPE}.imx"
exit
```

Execute the script to build:

```
$ ./make_mx8m_uboot.sh -mini
$ ls -t
firmware-imx-8.18      imx-atf      linux-imx                u-boot-
mx8m_mini.imx
firmware-imx-8.18.bin  imx-mkimage  make_mx8m_uboot.sh   uboot-imx
```

The boot image for Maaxboard Mini is u-boot-mx8m_mini.imx in the current directory.

Execute the following command to download the boot image into the SD card:

```
$ sudo dd if=u-boot-mx8m_mini.imx of=/dev/sdX bs=1k seek=33 conv=fsync
```

Where: /dev/sdX is the device node of the SD card.

2.3 Build Kernel in a standalone environment

Get the Linux source code

```
$ git clone https://github.com/Avnet/linux-imx.git -b maaxboard_lf-6.1.1-1.0.0
```

Check that the environment variables are correctly set :

```
$ echo $CROSS_COMPILE $ARCH
```

Build the kernel sources

```
$ cd linux-imx
$ make distclean
$ make maaxboard-mini_defconfig
$ make -j4
```

Execute the 'ls' command to view the Image and dtb files after compilation.

```
$ ls arch/arm64/boot/Image
$ ls arch/arm64/boot/dts/freescale/maaxboard*dtb
```

Execute the following command to compile the kernel modules, and install the modules to rootfs in the current directory.

```
$ make modules
$ make modules_install INSTALL_MOD_PATH=./rootfs
```

Perform the following four steps to program the kernel image and module into the SD card:

- Connect the SD card to the Ubuntu desktop system and mount the first partition (FAT format) and the second partition (EXT4 format) of the SD card respectively
- Copy Image and dtb files to the first partition of the SD card
- Copy ./rootfs//lib/modules to lib/ in the second partition of the SD card
- Unmount the two partitions of the SD and remove the SD card

Chapter 3 System Power on and Boot up

The default version of MaaXBoard Mini supports SD Card. Avnet also provide eMMC version for users to customize. To program the generated new Bootloader and System image files into MaaXBoard Mini's eMMC or SD card memory, or for guidance on power-up MaaXBoard Mini, the boot-up process, and how to exercise the supported BSP features of MaaXBoard Mini, please refer to ***MaaXBoard-Mini-Linux-Yocto-UserManual***.

Chapter 4 Appendix

4.1 Hardware Documents

For the detail hardware introduction, please refer to ***MaaXBoard Mini Hardware user manual***.

4.2 Software Documents

MaaXBoard Mini supports Yocto Linux system, for additional information, please refer to the following documents:

- ***MaaXBoard Mini Linux Yocto UserManual***

  - Describes how to boot MaaXBoard Mini and aspects of the BSP functionality

- ***MaaXBoard Mini Linux Yocto Development Guide***

  - Detailed guidance on how to rebuild the Linux system image(This document)

4.4 Contact Information

- Website: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/maaxboard/maaxboard-mini