NXP Semiconductors

Building Linux Image with QT5 for UDOO Quad

By: Technical Information Center

1 Requirements

1.1 Host machine with Ubuntu 14.04

1.2 UDOO Quad Board

1.3 uSD card with at least 4 GB

1.4 Download documentation at www.nxp.com/imx6q > Click on **Documentation tab** > Look for **Supporting Information** > L3.14.52_1.1.0_LINUX_DOCS

2 Overview and Important Concepts

This document describes how to build an image for UDOO Quad board with QT5 support by using a Yocto Project build environment. **Kernel 3.14.52** will be used in this process. This section briefly explains general information about the tools and some concepts that are used in this process.

2.1 BSP

Board support package is an implementation of specific support code for a given board that conforms to a given operating system. It is commonly built with a bootloader and a root file system.

2.2 Git

Git is a free and open source distributed version control system created by Linus Torvalds in 2005 for development of the Linux kernel. It is designed to handle everything from small to very large projects with speed and efficiency and used for software development and other version control tasks.

2.3 Repo

Repo is a tool built on top of Git that makes it easier to manage projects that contain multiple repositories. Repo is meant to make it easier to work with Git, it complements very well the layered nature of the Yocto Project, making it easier for users to add their own layers to the BSP.

2.4 Yocto - Is not an embedded Linux distribution, it creates a custom one for you

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded software regardless of the hardware architecture.

The NXP (formerly Freescale) Yocto Project BSP Release directory contains the recipes used to build projects and a set of scripts used to set up the environment. The recipes used to build the project come from both the community and NXP. For more information see ***'Freescale Yocto Project User's Guide'*** in the documentation package.

NXP BSPs and Community BSPs can be found in the following links.

NXP BSPs

- git.freescale.com > fsl-arm-yocto-bsp.git

Community BSPs

- https://github.com/Freescale/

3 Host Setup

To build the Linux image and the bootloader in an Ubuntu 14.04 host it is necessary to install essential Yocto Project host packages, the i.MX layers host packages for an Ubuntu 14.04 host and build tools.

```
$ sudo apt-get update

$ sudo apt-get install gawk wget git git-core diffstat unzip texinfo gcc-multilib
build-essential chrpath socat libsdl1.2-dev xterm sed cvs subversion coreutils
texi2html docbook-utils python-pysqlite2 help2man make gcc g++ desktop-file-utils
libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf automake groff curl lzop
asciidoc u-boot-tools phablet-tools gnutls-bin picocom ncurses-dev libtool libxml-
parser-perl fop xsltproc

$ sudo apt-get install gcc-arm-linux-gnueabihf

$ sudo apt-get install device-tree-compiler
```

4 Building Image

NXP along with Yocto Project provides various BSPs with different Linux Kernels for i.MX evaluation and development boards. In this document the NXP BSP Release based on Linux Kenel 3.14.52 was selected.

As mentioned at the beginning of the document the image that is going to be built is based on SABRE-SD board. This chapter explains how to build and deploy such image, following chapters describe which bootloader is used and how to generate a device tree for the UDOO Quad board.

4.1 Install Repo

4.1.1 Create a *bin/* directory in your home directory (if it does not exist) and make sure it is included in environment variable **PATH**.

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
```

NOTE: Alternately you can add the following lines at the end of *.bashrc* file to ensure that the *~/bin* folder is in your **PATH** variable.

```
#Ensure that the ~/bin folder is in PATH variable
export PATH=~/bin:$PATH
```

4.1.2 Download the Repo tool and ensure that it is executable

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

4.2 Initializing Repo Client

4.2.1 Configure Git with your real name and email address, then confirm the information is correct.

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

E.g.

$ git config --global user.name "John Smith"
$ git config --global user.email "john.smith@xcompany.com"
$ git config --list
```

4.2.2 Create an empty directory to hold your working files. In this case, a directory called *fsl-release-bsp_3.14.52* is created for the project. Any name can be used instead of this.

```
$ mkdir WORKING_DIRECTORY
$ cd WORKING_DIRECTORY

E.g.

$ mkdir ~/fsl-release-bsp_3.14.52
$ cd ~/fsl-release-bsp_3.14.52
```

4.2.3 Run `repo init` to bring down the latest version of Repo. You must specify an URL indicating the NXP Yocto Project BSP Release directory and use -b to check out a branch. To see a list of BSP Release branches, go to git.freescale.com > fsl-arm-yocto-bsp.git.

When this process is completed, the source code is checked out into the directory *fsl-release-bsp_3.14.52/sources*. A successful initialization will end with a message stating that Repo is initialized in your working directory. Your client directory should now contain a hidden .repo directory where files such as the manifest will be kept. The repo init is configured for the latest patches in the 3.14.52-1.1.0_ga release line.

```
$ repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-3.14.52-
1.1.0_ga
```

Note: If errors occur during repo initialization, try deleting the *.repo* directory and running the repo initialization command again.

4.2.4 To pull down the source tree to your working directory from the repositories it is necessary to run command `repo sync`. The initial sync operation will take a while to complete, the source files will be located in your working directory under their project names. You can perform repo synchronization, with the command repo sync, periodically to update to the latest code.

```
$ repo sync
```

4.3 Building an Image

4.3.1 NXP provides *fsl-setup-release.sh*, this script simplifies the setup for NXP machines. To use the script it is necessary to indicate the machine to be compiled for as well as the graphical backend (distro) desired. In this guide the machine used is **'imx6qsabresd'**, and the distro is **'fsl-imx-x11'**. To see a full list of NXP machines and distros please see ***'Freescale Yocto Project User's Guide'***.

The syntax for the *fsl-setup-release* script is shown below.

```
$ DISTRO=<distro name> MACHINE=<machine name> source fsl-setup-release.sh -b <build dir>

E.g.

$ DISTRO=fsl-imx-x11 MACHINE=imx6qsabresd source fsl-setup-release.sh -b build-
qt5-imx6qsabresd
```

DISTRO=<distro configuration name> configures the build environment and it is stored in {WORKING_DIRECTORY}/sources/meta-fsl-bsp-release/imx/meta-sdk/conf/distro.

MACHINE=<machine configuration name> points to the configuration file in conf/machine in meta-fsl-arm and meta-fsl-bsp-release.

-b <build dir> specifies the name of the build directory which fsl-setup-release.sh script will use. The local.conf file located in {WORKING_DIRECTORY/build_dir}/conf contains the machine and distro specifications:

MACHINE ??= 'imx6qsabreauto'
DISTRO ?= 'fsl-imx-x11'
ACCEPT_FSL_EULA = "1"

The MACHINE configuration can be changed by editing this file, if necessary.
ACCEPT_FSL_EULA in the *local.conf* file indicates that you have accepted the conditions of the EULA.

4.3.2 The Yocto Project build uses the `bitbake` command, to build an image it is `bitbake <image name>`. The Yocto Project provides some images and additional image recipes are provided in the *meta-fsl-bsp-release-layer***.** The following command is an example on how to build an image (building may take some hours). For a full list of NXP images and recipes please see ***'Freescale Yocto Project User's Guide'***.

```
$ bitbake fsl-image-qt5
```

4.4 Deploying Image

4.4.1 After the build is complete, the created image resides in *<build _directory>/tmp/deploy/images/<machine>*. Each image build creates a U-Boot, a kernel, and an image. Most machine configurations provide an SD card image (*.sdcard*), which contains U-Boot, the kernel and the rootfs completely set up for use on an SD card. To flash an SD card image, follow the next instructions.

First you need to know the name of the disk that is mounted in the Linux host when SD card is inserted and unmount all the partitions of that disk. To do this open a terminal window on your Linux host and type the command ***df -h*** (do not insert your SD card con host), this command will let you know the name of all the mounted units. Then insert your SD card and run the command again to identify the SD card disk name which will appear now on the listed units.

Take note of the name (e.g. */dev/sdc1* or */dev/mmcblk0p1*) and unmount the SD card including all the partitions using command ***umount***.

```
$ umount /dev/<disk>

E.g.

$ umount /dev/mmcblk0p1
$ umount /dev/mmcblk0p2
```

Then use command ***dd*** to deploy image.

```
$ sudo dd if=<image name>.sdcard of=/dev/<disk_partition> bs=1M && sync

E.g.

$ cd ~/fsl-release-bsp_3.14.52/build-qt5-imx6qsabresd/tmp/deploy/images/imx6qsabresd
$ sudo dd if=fsl-image-qt5-imx6qsabresd.sdcard of=/dev/mmcblk0 bs=1M && sync
```

Remove and reinsert SD card to verify that the 2 partitions where created successfully. For more information on deploying a Linux image on a SD card, see section "Preparing an SD/MMC Card to Boot" in the ***'i.MX Linux® User's Guide (IMXLUG)'***.

5 Bootloader

The image deployed in previous chapter provides a bootloader that is meant to work with SABRE-SD board, but not with UDOO. This chapter walks you through the process of overwriting the bootloader with another which will work with UDOO board.

5.1 Download and checkout

5.1.1 First it is necessary to download a copy the bootloader source into your local host.

```
$ cd ~/
$ git clone https://github.com/Freescale/u-boot-fslc.git uboot2016
```

5.1.2 The branch that is going to be used is not the master branch, so it is necessary to change the branch and make sure you are working under 2016.09+fslc.

```
$ cd ~/uboot2016
$ git branch
$ git checkout -b 2016.09+fslc remotes/origin/2016.09+fslc
$ git branch
```

5.2 Build Bootlaoder

5.2.1 To build the bootloader it is necessary to define ARCH and CROSS_COMPILE variables and set UDOO defconfig.

```
$ cd ~/uboot2016
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make udoo_defconfig
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make
```

5.3 Deploy Bootloader into microSD card

5.3.1 Insert in the SD card the host's cardreader and umount the SD card unit as it was done n point 4.4.

NOTE: In this document ***/dev/mmcblk0*** is used, however you may see that in your PC the SD card is mounted on ***/dev/sdX***.

5.3.2 To deploy bootloader execute commands below.

```
$ cd ~/uboot2016
$ sudo dd if=SPL of=/dev/mmcblk0 bs=1k seek=1
$ sudo dd if=u-boot.img of=/dev/mmcblk0 bs=1k seek=69
```

6 Device Tree Generation

A device tree is a data structure used to describe the physical hardware in a system. Once again, the device Tree Blob provided in the image generated on chapter 4 is customized for SABRE-SD and not for UDOO. To generate a valid DTB file it is necessary to pull the UDOO Linux Kernel source and take the necessary *.dts* and *.dtsi* files to build them with our Kernel source.

6.1 Download UDOO Linux Kernel and build DTB

6.1.1 To get a copy of UDOO Linux Kernel source use the following command.

```
$ cd ~/
$ git clone https://github.com/UDOOboard/linux_kernel.git udooKernel
```

6.1.2 Go to *~/udooKernel/arch/arm/boot/dts* and copy the files listed below into *~/fsl_release_bsp_3.14.52/build-qt5-imx6qsabresd/tmp/work-shared/imx6qsabresd/kernel-source/arch/arm/boot/dts*.

- imx6q-udoo-hdmi.dts
- imx6qdl-udoo.dtsi
- imx6qdl-udoo-externalpins.dtsi

6.1.3 Generate *imx6q-udoo-hdmi.dtb* file by executing next commands.

```
$ cd ~/fsl_release_bsp_3.14.52/build-qt5-imx6qsabresd/tmp/work-
shared/imx6qsabresd/kernel-source/
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ make imx_v7_defconfig
$ make imx6q-udoo-hdmi.dtb
```

6.1.4 At this point ***imx6q-udoo-hdmi.dtb*** should've been generated inside *~/fsl_release_bsp_3.14.52/build-qt5-imx6qsabresd/tmp/work-shared/imx6qsabresd/kernel-source/arch/arm/boot/dts*. Finally rename this file as ***imx6q-udoo.dtb*** and copy it in the FAT partition of the SD card and remove all other .dtb files on this partition.

Now your image with Linux Kernel 3.14.52 and QT5 is ready to boot!!!