# VAR-SOM-MX6 - Yocto Thud 2.6.2 based on FSL Community BSP 2.6 with 4.14.78_1.0.0-ga Linux release

Contents [hide]



## 1. Installing required packages

Please make sure your host PC is running Ubuntu 16.04 64-bit and install the following packages:

```
$ sudo apt-get install gawk wget git diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping libsdl1.2-dev xterm

$ sudo apt-get install autoconf libtool libglib2.0-dev libarchive-dev python-git \
sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 \
help2man make gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev \
mercurial automake groff curl lzop asciidoc u-boot-tools dos2unix mtd-utils pv \
libncurses5 libncurses5-dev libncursesw5-dev libelf-dev zlib1g-dev bc rename
```

## 2. Reference documentation

## 2. Reference documentation

- Yocto Project Core - Thud 2.6.2

Documentation is available from www.yoctoproject.org

## 3. Download Yocto Thud based on Freescale Community BSP

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"

$ mkdir ~/bin (this step may not be needed if the bin folder already exists)
$ curl https://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ export PATH=~/bin:$PATH
```

```
$ mkdir ~/var-fslc-yocto
$ cd ~/var-fslc-yocto
```

Now, choose between downloading a release tag, and downloading the latest revision (recommended) and **follow only one of the next two bullet sections**, accordingly:

- **Download a release tag**

Each release in https://github.com/varigit/variscite-bsp-platform/releases corresponds to a tag.
The tags are also listed in https://github.com/varigit/variscite-bsp-platform/tags
To specify a specific release/tag, run the following:

```
$ repo init -u https://github.com/varigit/variscite-bsp-platform.git -b refs/tags/TAG_NAME

For example:
$ repo init -u https://github.com/varigit/variscite-bsp-platform.git -b refs/tags/thud-fslc-4.14.78-mx6-v1.0
$ repo sync -j4
```

or

- **Download the latest revision (recommended)**

```
$ repo init -u https://github.com/varigit/variscite-bsp-platform.git -b thud
$ repo sync -j4
```

## 4. Setup and build Yocto

### 4.1. Supported images

The following images are provided by Variscite for evaluation purpose

- **fsl-image-gui**: Default Variscite demo image with GUI and without any Qt5 content. This image recipe works on all backends for X11, Frame Buffer, Wayland and XWayland and the content is optimized to fit 512MB NAND flash.
- **fsl-image-qt5**: Extends fsl-image-gui image with Qt5 support and various Qt samples for X11, Frame Buffer, Wayland and XWayland backends.

> Will result in image size greater than 512 MB, which will not fit into NAND flash. Use SD card or eMMC to test.

The following images are provided by FSL Community BSP:

- **fsl-image-machine-test**: A console-only image that includes gstreamer packages, Freescale's multimedia packages (VPU and GPU), and test and benchmark applications.
- **fsl-image-mfgtool-initramfs**: Small image to be used with Manufacturing Tool (mfg-tool) in a production environment.
- **fsl-image-multimedia**/**fsl-image-multimedia-full**: A console-only image that includes gstreamer packages and Freescale's multimedia packages (VPU and GPU).

See the list of Yocto Project's reference images in Yocto Project Reference Manual

### 4.2. Supported distros

The following distros can be used:

- **fslc-x11**: Distro for X11 without wayland. This distro include x11 feature and doesn' has wayland support.
- **fslc-framebuffer**: Distro for Framebuffer graphical backend. This distro doesn't include X11 and wayland features.
- **fslc-wayland**: Distro for Wayland without X11. This distro includes wayland feature but doesn't have x11 support.
- **fslc-xwayland**: Distro for Wayland with X11. This distro includes both wayland and X11 emulation features.

Note: Also standard Poky distros can be used

### 4.3. GStreamer support

FSL community BSP comes with gstreamer-imx, a set of GStreamer1.0 plugins for i.MX platform, which make use of the i.MX multimedia capabilities.
Some of the multimedia plugins do not work well with X11 and Wayland backends.
To get the most from gstreamer-imx, it is recommended to use fslc-framebufer distro with one of the demo images

### 4.4. Build X11 GUI demo image

```
$ cd ~/var-fslc-yocto
$ MACHINE=var-som-mx6 DISTRO=fslc-x11 . setup-environment build_x11
```

The above command is only mandatory for the very first build setup: whenever restarting a newer build session (from a different terminal or in a different time), you can skip the full setup and just run

```
$ cd ~/var-fslc-yocto
$ source setup-environment build_x11
```

Optional steps: local.conf customization

launch bitbake:

```
Without Qt content:
$ bitbake fsl-image-gui

Or with Qt content:
$ bitbake fsl-image-qt5
```

NOTE: Some of the blitter-based i.MX GStreamer plugins do not work with X11 and Wayland backends. To get the most of the i.MX GPU/VPU acceleration, use the fslc-framebuffer backend.

### 4.5. Build console-only demo image with Freescale's multimedia packages (VPU and GPU)

```
$ cd ~/var-fslc-yocto
$ MACHINE=var-som-mx6 DISTRO=fslc-framebuffer . setup-environment build_fb
```

Optional steps: local.conf customization

```
Without Qt content:
$ bitbake fsl-image-gui

Or with Qt content:
$ bitbake fsl-image-qt5
```

### 4.6. local.conf customization

#### 4.6.1. Change the downloads directory

Create a /opt/yocto_downloads directory and set its permissions:

```
$ sudo mkdir /opt/yocto_downloads
$ sudo chmod 777 /opt/yocto_downloads/
```

Direct downloads to it, by replacing 'DL_DIR ?= "${BSPDIR}/downloads/"' with 'DL_DIR = "/opt/yocto_downloads/"' in conf/local.conf under your build directory:

```
$ sed -i 's/DL_DIR ?= "${BSPDIR}\/downloads/DL_DIR = "\/opt\/yocto_downloads/g' conf/local.conf
```

#### 4.6.2. Add Qt creator and Eclipse debug support to your images

Append the following to the conf/local.conf file in your Yocto build directory, to add Eclipse debug support to your images:

```
EXTRA_IMAGE_FEATURES = " \
    eclipse-debug \
    ssh-server-openssh \
    "
```

Append the following to the conf/local.conf file in your Yocto build directory, to add Qt creator debug support to your images:

```
EXTRA_IMAGE_FEATURES = " \
    qtcreator-debug \
    ssh-server-openssh \
    "
```

#### 4.6.3. Use systemd instead of SysV init

Append the following to the conf/local.conf file in your Yocto build directory, to use systemd instead of SysV init in your images:

```
DISTRO_FEATURES_append = " systemd"
```

```
DISTRO_FEATURES_BACKFILL_CONSIDERED_append = " sysvinit"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_initscripts = ""
IMX_DEFAULT_DISTRO_FEATURES_append = " systemd"
```

#### 4.6.4. Create a read-only root file system

Append the following to the conf/local.conf file in your Yocto build directory, to create a read-only rootfs:

```
EXTRA_IMAGE_FEATURES += "read-only-rootfs"
```

### 4.7. Build Results

The resulting images are located in tmp/deploy/images/var-som-mx6.

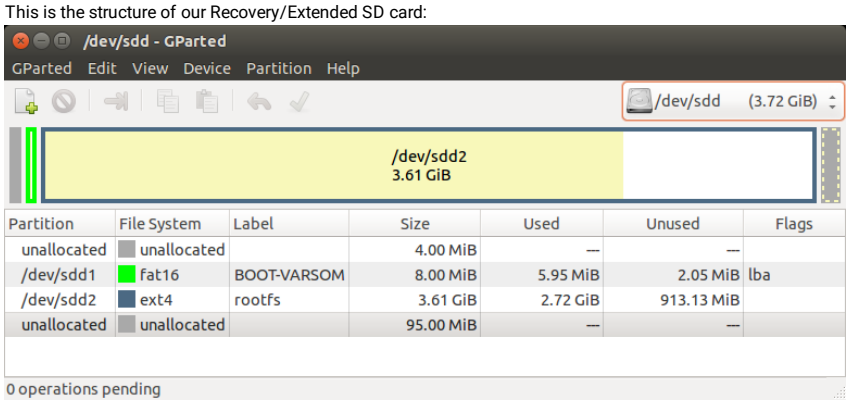
| Image Name | Description |
|---|---|
| fsl-image-gui-var-som-mx6.wic.gz | This image is for SD card boot.<br>It can be flashed as-is on an SD card that can then be used to boot your system,<br>according to the relevant startup-guide of your product<br>(usually requires to press the boot select button, or toggle a DIP switch).<br>For detailed information refer to the Create a bootable SD card section below. |
| fsl-image-gui-var-som-mx6.tar.gz | Tarball with rootfs files.<br>Can be used to create an NFS root file system on the host.<br>See the Yocto Setup TFTP/NFS section for more info.<br>Also used to create our extended SD card.<br>See the Create a bootable SD card section below. |
| fsl-image-gui-var-som-mx6_128kbpeb.ubi | A complete UBI image containing a UBIFS volume, for writing to NAND flash with 128KiB PEB. |
| fsl-image-gui-var-som-mx6_256kbpeb.ubi | A complete UBI image containing a UBIFS volume, for writing to NAND flash with 256KiB PEB. |
| uImage | Linux kernel image, same binary for SD card/eMMC or NAND flash. |
| SPL-sd | SPL built for SD card boot or eMMC boot. |
| SPL-nand | SPL built for NAND flash. |
| u-boot.img-sd | U-Boot built for SD card boot or eMMC boot. |
| u-boot.img-nand | U-Boot built for NAND flash. |

| Device Tree Name | SOM type | Carrier Board type | LCD Type | Evaluation Kit name |
|---|---|---|---|---|
| uImage-imx6q-var-som-cap.dtb | VAR-SOM-MX6_V2 (Quad / Dual) | VAR-MX6CustomBoard | Capacitive touch | VAR-DVK-MX6_V2-PRO<br>VAR-STK-MX6_V2 |
| uImage-imx6q-var-som-res.dtb | VAR-SOM-MX6_V2 (Quad / Dual) | VAR-MX6CustomBoard | Resistive touch | VAR-DVK-MX6_V2-PRO<br>VAR-STK-MX6_V2 |
| uImage-imx6q-var-som-vsc.dtb | VAR-SOM-MX6_V2 (Quad / Dual) | VAR-SOLOCustomBoard | Capacitive LVDS touch | N/A |
| uImage-imx6dl-var-som-cap.dtb | VAR-SOM-MX6_V2 (DualLite / Solo) | VAR-MX6CustomBoard | Capacitive touch | N/A |
| uImage-imx6dl-var-som-res.dtb | VAR-SOM-MX6_V2 (DualLite / Solo) | VAR-MX6CustomBoard | Resistive touch | N/A |
| uImage-imx6dl-var-som-vsc.dtb | VAR-SOM-MX6_V2 (DualLite / Solo) | VAR-SOLOCustomBoard | Capacitive LVDS touch | N/A |
| uImage-imx6qp-var-som-cap.dtb | VAR-SOM-MX6_V2 (QuadPlus / DualPlus) | VAR-MX6CustomBoard | Capacitive touch | N/A |
| uImage-imx6qp-var-som-res.dtb | VAR-SOM-MX6_V2 (QuadPlus / DualPlus) | VAR-MX6CustomBoard | Resistive touch | N/A |
| uImage-imx6qp-var-som-vsc.dtb | VAR-SOM-MX6_V2 (QuadPlus / DualPlus) | VAR-SOLOCustomBoard | Capacitive LVDS touch | N/A |
| uImage-imx6dl-var-som-solo-cap.dtb | VAR-SOM-SOLO / VAR-SOM-DUAL | VAR-MX6CustomBoard | Capacitive touch | N/A |
| uImage-imx6dl-var-som-solo-res.dtb | VAR-SOM-SOLO / VAR-SOM-DUAL | VAR-MX6CustomBoard | Resistive touch | N/A |
| uImage-imx6dl-var-som-solo-vsc.dtb | VAR-SOM-SOLO / VAR-SOM-DUAL | VAR-SOLOCustomBoard | Capacitive LVDS touch | VAR-DVK-SOLO/DUAL<br>VAR-STK-SOLO/DUAL |
| uImage-imx6q-var-dart.dtb | DART-MX6 | VAR-DT6CustomBoard | Capacitive LVDS touch | VAR-DVK-DT6<br>VAR-STK-DT6 |

## 5. Create a bootable SD card

### 5.1. SD card structure

This is the structure of our Recovery/Extended SD card:




The SD card is divided into 3 sections as shown in the picture above:

- The first unallocated 4MiB are saved space for U-Boot. It can be replaced using the dd command as described in the Yocto Build U-Boot section.
- The first partition is a fat16 partition used for the device tree files and the kernel image file. You can copy them as described in the Yocto Build Linux section.
- The second partition is an ext4 partition that contains the complete root filesystem (including the kernel modules).

Note:
The last unallocated area is not used. It is there so that the rootfs will fit on any 4GB SD card, as not all 4GB SD cards are really the same size. If you want, you can use a program such as GParted to resize the roofs partition and make it end at the end of your specific SD card (of course, you can also use SD cards with much bigger capacity than 4GB, and then it makes more sense to resize the partition).
Also, if you create the extended SD card yourself by following the Create an extended SD card section below, and you use the '-a' option, the rootfs partition will end at the end of your specific SD card automatically.

## 5.2. Yocto pre-built bootable SD card

The Yocto build products contains many files as explained in the Build Results section. For example, fsl-image-gui-var-som-mx6.wic.gz, depending on your build. This is a complete image to be flashed directly to an SD card.

Example usage:

```
$ sudo umount /dev/sdX*
```

```
# For GUI-X11 & Qt5-X11
$ cd ~/var-fslc-yocto/build_x11
Or
# For Qt5-FB
$ cd ~/var-fslc-yocto/build_fb

# For fsl-image-gui image (GUI-X11)
$ zcat tmp/deploy/images/var-som-mx6/fsl-image-gui-var-som-mx6.wic.gz | sudo dd of=/dev/sdX bs=1M && sync
Or
# For fsl-image-qt5 image (Qt5-X11 & Qt5-FB)
$ zcat tmp/deploy/images/var-som-mx6/fsl-image-qt5-var-som-mx6.wic.gz | sudo dd of=/dev/sdX bs=1M && sync
```

```
Replace sdX with the right device name. This can be obtained by "dmesg" command on your host Linux PC, after the SD card
```

- Note: Booting your system from an SD card requires pressing the boot-select button, or switching the relevant DIP switch to "Boot from SD card", according to the relevant start-up guide of your system

Drawbacks of the native .wic.gz yocto-built image, (relative to the Recovery/Extended SD card):

- The rootfs partition doesn't use the entire SD card.
- The rootfs partition is not labeled as rootfs.
- The NAND flash and eMMC installation scripts and images are not included.

## 5.3. Create an extended SD card

Variscite provides the var-create-yocto-sdcard.sh script which creates our recovery SD card - an SD card based on the fsl-image-gui filesystem, which also contain the scripts and relevant binaries for installation to the internal storage of the SOM.
Later, you will be able to follow either the more automatic Yocto Recovery SD card guide or the more manual Installing Yocto to the SOM's internal storage guide.

Note:

Note:
This is essentially the same as our pre-built Recovery SD image, with the following main differences:

- The Android recovery (Android-eMMC) is only present on the pre-built SD image, and not on the SD card built from the Yocto script.
- The pre-built image's rootfs partition size is 3700MiB, which is also the default size when using the script, but the script also has an option to set the rootfs partition size to fill the whole free space of the used SD card. Anyway, you can always resize the partition later with an external tool such as gparted.

Naturally, the pre-built image is more straight forward and easier to use, while the script method is easier to customize.

Usage:

- Follow the Setup and build Yocto guide, and bitbake fsl-image-gui.
- Plug-in the SD card to your Linux Host PC, run dmesg and see which device is added (i.e. /dev/sdX or /dev/mmcblkX)

```
$ cd ~/var-fslc-yocto
$ sudo MACHINE=var-som-mx6 sources/meta-variscite-fslc/scripts/var_mk_yocto_sdcard/var-create-yocto-sdcard.sh <options> /
(Replace /dev/sdX with your actual device)
```

```
options:
 -h            Display help message
 -s            Only show partition sizes to be written, without actually write them
 -a            Automatically set the rootfs partition size to fill the SD card
 -r            Select alternative rootfs for recovery images (default: build_x11/tmp/deploy/images/var-som-mx6/fsl-image-
```

```
If you don't use the '-a' option, a default rootfs size of 3700MiB will be used
The '-r' option allows you to create a bootable SD card with an alternative image for the installation to NAND flash or e
Example: "-r tmp/deploy/images/var-som-mx6/fsl-image-qt5-var-som-mx6" -- selected the "Qt5 image with X11" recovery image
```

#### 5.3.1. Create an extended SD card image using a loop device

It is also possible to use the var-create-yocto-sdcard.sh script to create an extended SD card image, while using a loop device instead of attaching a real SD card.

Create an empty file using the following command:

```
$ dd if=/dev/zero of=var-som-mx6-extended-sd.img bs=1M count=3720
```

The above command creates a 3700MiB file representing the SD card.

Attach the first available loop device to this file:

```
$ sudo losetup -Pf var-som-mx6-extended-sd.img
```

To find the actual loop device being used, run:

```
$ losetup -a | grep var-som-mx6-extended-sd.img
```

Write the content to the loop device to generate the SD card image:

```
$ sudo MACHINE=var-som-mx6 sources/meta-variscite-fslc/scripts/var_mk_yocto_sdcard/var-create-yocto-sdcard.sh <options> /
```

(Replace /dev/loopX with your actual loop device, e.g. /dev/loop0)

Detach the loop device from the file:

```
$ sudo losetup -d /dev/loopX
```

To compress the SD card image file use the following command:

```
$ gzip -9 var-som-mx6-extended-sd.img
```

To write the SD card image to a real SD card device use the following command:

```
$ zcat var-som-mx6-extended-sd.img.gz | sudo dd of=/dev/sdX bs=1M && sync
```

(Replace /dev/sdX with your actual SD device, e.g. /dev/sdb)

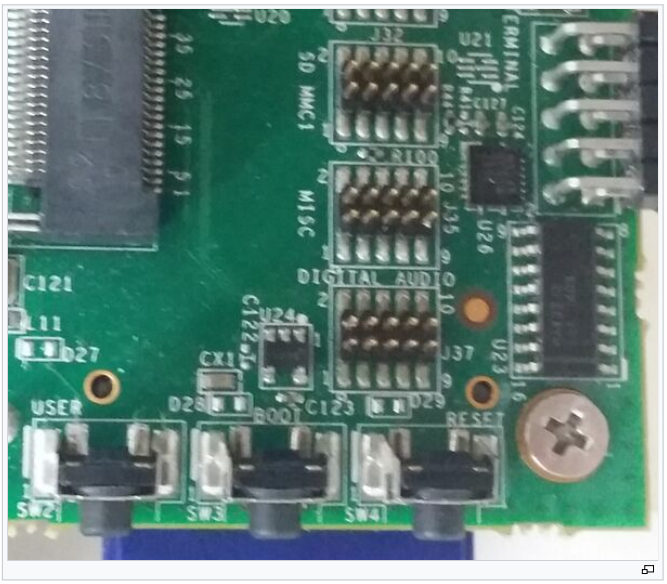
## 6. Boot the board with a bootable SD card

### 6.1. Setting the Boot Mode

Follow the instruction below according to the appropriate carrier board type:

#### 6.1.1. MX6CustomBoard

Booting your MX6CustomBoard system from SD card requires pushing the middle button while powering up the system. See picture below.

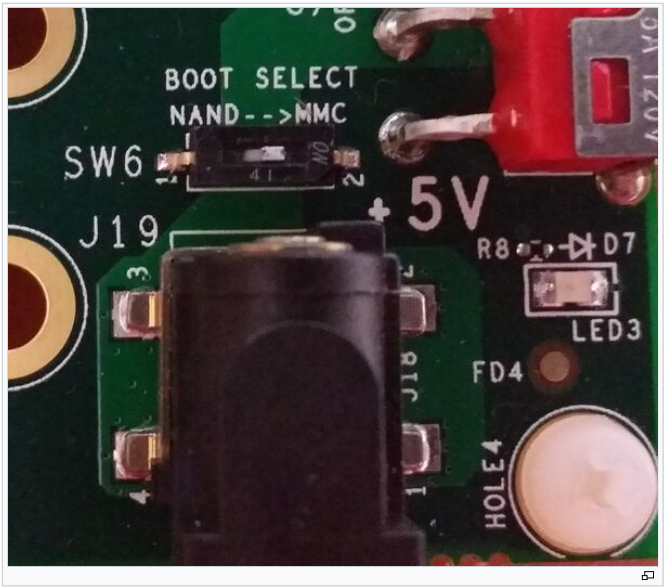



To boot a board using an SD card, follow the steps below:

- Power-off the board.
- Insert the SD card into the SD/MMC slot of the carrier board (DVK)
- Push the middle button (Boot Select) and hold
- Power-up the board
- Release the middle button (Boot Select) after system starts to boot.
- The board will automatically boot into Linux from the SD card

### 6.1.2. SoloCustomBoard

Booting your system requires switching the relevant DIP switch to "Boot from MMC". See picture below.

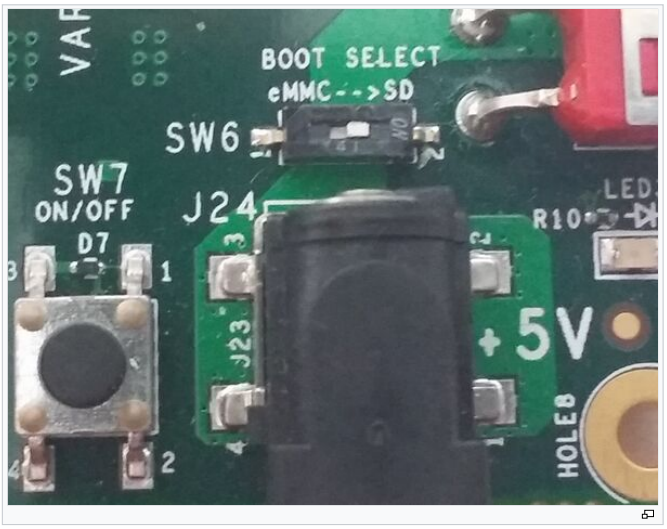



To boot board with SD card, Follow the steps below:

- Power-off the board.
- Insert the SD card into the SD/MMC slot of the carrier board (DVK)
- Switch the relevant DIP switch to "Boot from MMC"
- Power-up board
- The board will automatically boot into Linux from SD card

### 6.1.3. DT6CustomBoard

Booting your system requires switching the relevant DIP switch to "Boot from SD card". See picture below.




To boot board with SD card, Follow the steps below:

- Power-off the board.
- Insert the SD card into the SD/MMC slot of the carrier board (DVK)
- Switch the relevant DIP switch to "Boot from SD card"
- Power-up board
- The board will automatically boot into Linux from SD card

### 6.2. Automatic device tree selection in U-Boot

As shown in the Build Results table above, we have different kernel device trees, corresponding to our different H/W configurations.
We implemented a script in U-Boot's environment, which sets the fdt_file environment variable based on the detected hardware.

#### 6.2.1. Enable/Disable automatic device tree selection

To enable the automatic device tree selection in U-Boot (already enabled by default):

```
$ setenv fdt_file undefined
$ saveenv
```

To disable the automatic device tree selection in U-Boot, set the device tree file manually:

```
$ setenv fdt_file YOUR_DTB_FILE
$ saveenv
```

Useful example: To list all files in the boot partition (where the dtb files are by default) of an SD card:

```
$ ls mmc 0:1
```

## 7. Flash images to NAND/eMMC

Please refer to Yocto NAND Flash Burning guide.

## 8. Yocto Image Customization

### 8.1. Update Yocto Thud to latest revision

From time to time we update the Yocto sources (especially meta-variscite) with new features and bug fixes.
Follow the **Download the latest revision (recommended)** bullet section of the Download Yocto Thud based on Freescale Community BSP step again to update your tree to the latest revision, and rebuild your image.

### 8.2. Update Yocto Thud to a release tag

Follow the **Download a release tag** bullet section of the Download Yocto Thud based on Freescale Community BSP step to update your tree to a release tag, and rebuild your image.

### 8.3. Forcing Clean Build

```
In order to update the kernel, U-Boot and rootfs:
```

```
$ bitbake -c cleanall u-boot-variscite linux-variscite kernel-module-imx-gpu-viv ti-compat-wireless-wl18xx wl18xx-firmwar

for GUI image
$ bitbake -c clean fsl-image-gui

for Qt5 image
$ bitbake -c clean fsl-image-qt5
```

## 8.4. Qt5 for Embedded Linux

To run Qt5 applications without X11 backend the platform specific plugins (e.g. EGLFS or LinuxFB) should be configured with QT_QPA_PLATFORM environment variable or with -platform command-line.

See more information on Qt5 customization for Embedded Linux here

### 8.4.1. Configure EGLFS Plugin

```
export QT_QPA_EGLFS_PHYSICAL_HEIGHT=95
export QT_QPA_EGLFS_PHYSICAL_WIDTH=160
export QT_QPA_EGLFS_HEIGHT=480
export QT_QPA_EGLFS_WIDTH=800
export QT_EGLFS_IMX6_NO_FB_MULTI_BUFFER=1
export QT_QPA_EGLFS_DEPTH=24
export QT_QPA_PLATFORM=eglfs
```

### 8.4.2. Configure Touch Input

When no windowing system is present, the mouse, keyboard, and touch input are read directly via evdev or tslib.

#### 8.4.2.1. Evdev

By default, the Qt5 uses automatic device discovery based on libudev. In case you want to override the default touchscreen configuration the following parameters can be used:

- /dev/input/... - Specifies the name of the input device. When not given, Qt looks for a suitable device either via libudev or by walking through the available nodes.
- rotate - On some touch screens the coordinates must be rotated, which is done by setting rotate to 90, 180, or 270.
- invertx and inverty - To invert the X or Y coordinates in the input events, pass invertx or inverty.

```
export QT_QPA_EVDEV_TOUCHSCREEN_PARAMETERS='/dev/input/touchscreen0'
```

#### 8.4.2.2. Tslib

Tslib is used for resistive single-touch touchscreens and should be pre-configured with:

```
export TSLIB_TSEVENTTYPE='INPUT'
export TSLIB_TSDEVICE='/dev/input/touchscreen0'
export TSLIB_CALIBFILE='/etc/pointercal'
export TSLIB_CONFFILE='/etc/ts.conf'
export TSLIB_CONSOLEDEVICE='none'
export TSLIB_FBDEVICE='/dev/fb0'
export TSLIB_PLUGINDIR='/usr/lib/ts'
export QT_QPA_EGLFS_TSLIB=1
export QT_QPA_FB_TSLIB=1
```

It is recommended to put the above setup inside /etc/profile.d/tslib.sh.

### 8.4.3. Running Qt5 Applications

```
$ cd /usr/share/cinematicexperience-1.0; ./Qt5_CinematicExperience --platform eglfs
$ cd /usr/share/qt5everywheredemo-1.0; ./QtDemo --platform eglfs
$ cd /usr/share/qtsmarthome-1.0; ./smarthome --platform eglfs
```

## 8.5. UBIFS

By default we create ubifs image for 512MB NAND flash size. You can change the size by editing ~/var-fslc-yocto/sources/meta-variscite-fslc/conf/machine/include/variscite.inc
and comment / uncomment the relevant section based on size.

## 8.6. DDR size and Contiguous Memory Allocator

By default Freescale allocates 256MB of RAM to the Contiguous Memory allocator. This is for proper operation of the IPU VPU, X11 etc. On VAR-SOM-SOLO with 256MB DDR RAM size, it will cause a kernel freeze during boot. Adding cma=32MB to the bootargs parameters is required to fix.

# 9. Make changes to the rootfs

The following is usually not the recommended way to work with Yocto.
You should usually create new specific recipes (.bb files) and/or append to specific present recipes by using .bbappend files.
However, if you are not yet experienced enough with Yocto, and you just want to quickly add your files to the resultant file system (or make any other change to it), you can

do it in a general way, by using the following variable:

```
ROOTFS_POSTPROCESS_COMMAND


    Specifies a list of functions to call once the OpenEmbedded build system has created the root filesystem.
    You can specify functions separated by semicolons:

         ROOTFS_POSTPROCESS_COMMAND += "function; ... "

    If you need to pass the root filesystem path to a command within a function, you can use ${IMAGE_ROOTFS},
    which points to the directory that becomes the root filesystem image. See the IMAGE_ROOTFS variable for more informat
```

The functions will be called right after the root filesystem is created and right before it is packed to images (.wic.gz, .ubi, .tar.gz, etc.).

### 9.1. Example

Let's say you have your files that you want to put in the filesystem arranged on your host under a directory called /my_rootfs_additions, like the following:

```
my_rootfs_additions/
├── data
│   ├── example.m4v
│   └── example.bin
├── etc
│   └── example.conf
└── home
    └── root
        └── .example
```

And let's say you want to build the fsl-image-gui image.

Create a file called ~/var-fslc-yocto/sources/meta-variscite-fslc/recipes-images/images/fsl-image-gui.bbappend
with the following content:

```
add_my_files() {
    cp -r /my_rootfs_additions/*  ${IMAGE_ROOTFS}/
}

ROOTFS_POSTPROCESS_COMMAND += "add_my_files;"
```

Now, when you bitbake fsl-image-gui, the files in /my_rootfs_additions will be added to the rootfs (be careful when overwriting files).

## 10. Useful Bitbake commands

Bitbake Cheat Sheet

Useful bitbake commands

i.MX Yocto Project: Itib versus bitbake

Categories: Yocto | VAR-SOM-MX6