Yocto Linux BSP for SMARC-sAMX6i R01.00

1. Revision history

Revision	Date	Author	Description
1.00	04/12/2014	Viktor Krasnov	Release notes for R01.00

2. Table of contents

Table of Contents

3. Introduction

This document describes Linux Board Support Package (BSP) for Kontron SMARC-sAMX6i module. It provides:

- a summary of BSP features;
- build and installation notes;
- listing of the release package contents.

4. Supported features

- Freescale i.MX6 Quad, Dual, DualLite and Solo processors;
- Integrated Vivante Graphics core(2D/3D acceleration, Hardware accelerated Video);
- Primary display on either LVDS or LCD module's interface. Both 18-bit and 24-bit displays are supported;
- Secondary display on HDMI;
- HDMI Audio;
- Backlight and power control for primary display;
- Ethernet interface(10/100/1000 Mbps);
- SATA interface (for i.MX6 Quad and Dual processors only);
- On-module eMMC flash;
- PCIe interfaces (PCIe-A for modules without PCIe switch, PCIe-A/PCIe-B/PCIe-C for modules with PCIe switch);
- USB OTG on module's USB1 interface;
- USB host on module's USB2 and USB3 interfaces;
- Suspend/Resume;
- SDIO interface;
- 4 module's serial ports;
- Built-in temperature sensor;
- 12 SMARC GPIO lines;
- Watchdog;
- 5 SMARC I2C buses(I2C_GP, I2C_PM, HDMI DDC, I2C_CAM, I2C_LCD);
- 2 SMARC SPI buses;
- On-module SPI flash;
- External USB touch screens (eGalax USB TouchController and Atmel maXTouch Digitizer).

5. Unsupported features

See chapter “8. Known issues”

6. BSP Components

File name in delivery	Description
smarc-samx6i-support-R01.00.diff	Patch that adds support of Kontron SMARC-sAMX6 module to FSL Community BSP1.6 (Daisy)
smarc-samx6i-kernel-and-device-tree-binaries-R01.00.tar.gz	This archive contains prebuilt Linux kernel, corresponding modules and device tree files for different h/w configurations. These binaries can be used to boot SMARC-sAMX6 module to evaluate functionality. ***Note:*** it's highly recommended to use binaries, which are automatically created during build of rootfs image, instead of files provided in this archive.
md5sum	Md5sum hashes for all files in the delivery.
EULA	End-user license agreement.

7. Installation and set up procedures

The BSP is based on FSL Community BSP (http://freescale.github.io/), which is based on Yocto 1.6 Daisy. Due to legal limitations imposed by FREESCALE SEMICONDUCTOR SOFTWARE LICENSE AGREEMENT a rootfs image can't be provided and should be created manually - please refer to instructions below. Furthermore, to avoid conflict between this rootfs image and prebuilt binaries for kernel and device tree tables, which are provided with this BSP in smarc-samx6i-kernel-and-device-tree-binaries-R01.00.tar.gz archive, it's highly recommended to use kernel and device tree binaries built alongside the rootfs image.

When all images/binaries are ready, it's possible to deploy them to the SMARC-sAMX6i module.

7.1. Building BSP bootable images(rootfs, kernel, device tree)

1. Make sure that your host development system meets the following requirements: http://www.yoctoproject.org/docs/1.6/mega-manual/mega-manual.html#intro-requirements
2. Download the FSL community BSP to your host:
   - Install the repo utility:
     ```
     mkdir ~/bin
     curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
     chmod a+x ~/bin/repo
     ```
   - Download the BSP source code:
     ```
     PATH=${PATH}:~/bin
     mkdir fsl-community-bsp
     cd fsl-community-bsp
     repo init -u https://github.com/Freescale/fsl-community-bsp-platform -b daisy
     repo sync
     ```
3. Change current directory to sources/meta-fsl-arm-extra/, and add support of SMARC-sAMX6i to the previously downloaded Freescale BSP by applying the BSP patch:
   ```
   cd sources/meta-fsl-arm-extra/
   patch -p1 < /path/to/smarc-samx6i-support-R01.00.diff
   cd ../..
   ```
4. Set up Yocto build environment:
   ```
   MACHINE=smarc-samx6i . ./setup-environment build
   ```
5. Now, it's possible to build the appropriate rootfs BSP image using, for example, the following command:

```
bitbake core-image-sato
```

Please refer to https://community.freescale.com/docs/DOC-95421 for the list of all available types of rootfs images.

6. After build completes, results are available in tmp/deploy/images/smarc-samx6i/ directory. This directory will contain:
   - a tarball with rootfs image of appropriate type(e.g. core-image-sato-smarc-samx6i.tar.bz2);
   - a kernel image with modules(e.g. uImage-smarc-samx6i.bin and modules-smarc-samx6i.tgz);
   - device tree BLOBs (see section below for details).

Please refer to official Yocto and Freescale documentation for more information on build process:

https://www.yoctoproject.org/documentation
http://freescale.github.io/

Note: if previous version of SMARC-sAMX6i BSP has been already installed, there is no need to do complete download and rebuild. Instead:

1. From the build directory, remove previous kernel build by executing:
   ```
   bitbake -c clean linux-smx6
   ```
2. Change current directory to the BSP directory, revert previous BSP patch, and apply new BSP patch:
   ```
   cd sources/meta-fsl-arm-extra/
   patch -R -p1 < /path/to/smarc-samx6i-support-XXXXXX.diff
   patch -p1 < /path/to/smarc-samx6i-support-R01.00.diff
   cd ../..
   ```
3. Rebuild kernel and filesystem images. E.g:
   ```
   bitbake core-image-sato
   ```

7.2. Choosing the appropriate Device tree table for deployment

The SMARC-sAMX6i BSP uses device tree based hardware description. The following device tree tables are built by default and can be found in tmp/deploy/images/smarc-samx6i/ directory:

1. uImage-imx6q-smx6-lcd.dtb - should be used on SMARC sAMX6i module with i.MX6 Quad or i.MX6 Dual processor, without PCIe switch, when module's LCD interface is used for primary display;
2. uImage-imx6q-smx6-lcd-pcieswitch.dtb - should be used on SMARC sAMX6i module with i.MX6 Quad or i.MX6 Dual processor, with PCIe switch, when module's LCD interface is used for primary display;
3. uImage-imx6q-smx6-lvds.dtb - should be used on SMARC sAMX6i module with i.MX6 Quad or i.MX6 Dual processor, without PCIe switch, when module's LVDS interface is used for primary display;
4. uImage-imx6q-smx6-lvds-pcieswitch.dtb - should be used on SMARC sAMX6i module with i.MX6 Quad or i.MX6 Dual processor, with PCIe switch, when module's LVDS interface is used for primary display;
5. uImage-imx6dl-smx6-lcd.dtb – should be used on SMARC sAMX6i module with i.MX6 DualLite or i.MX6 Solo processor, without PCIe switch, when module's LCD interface is used for primary display;
6. uImage-imx6dl-smx6-lvds.dtb – should be used on SMARC sAMX6i module with i.MX6 DualLite or i.MX6 Solo processor, without PCIe switch, when module's LVDS interface is used for primary display.

Note 1: all DTB files above are intended to be used only with Kontron SMARC Evaluation carrier board Rev A-C.

Note 2: The following information may be useful for further BSP customization:

1. Since device tree language does not provide any sane way to define different hardware options (such as different device topologies depending on GPIO input or kernel command line) within same DTB file, a C preprocessor is utilized to generate different DTB files depending on macros.
2. Device tree parts for SMARC sAMX6i module and for Kontron SMARC evaluation carrier board are described in separate source files: arch/arm/boot/dts/imx6qdl-smx6.dtsi and arch/arm/boot/dts/kontron-smarc-evaluation-carrier.dtsi;
3. The imx6qdl-smx6.dtsi file contains description for SMARC sAM6Xi module. In case of BSP customization this file should be included into new dts file after defining some of the following macros:
   - SMX6_CPU_IMX6Q – should be defined to build a device tree for SMARC sAMX6i module with i.MX6 Quad or i.MX6 Dual processor;
   - SMX6_CPU_IMX6DL – should be defined to build a device tree for SMARC sAMX6i module with i.MX6 DualLite or i.MX6 Solo processor;
   - SMX6_PANEL_LCD – should be defined if module's LCD interface is used for primary display;
   - SMX6_PANEL_LVDS – should be defined if module's LVDS interface is used for primary display;
   - SMX6_PCIE_SWITCH – should be defined if module has PCIe switch;
   - SMX6_HIDE_EMMC – may be defined to avoid registering on-board eMMC device;
   - SMX6_HIDE_SPI_FLASH – may be define to avoid registering on-board SPI flash;
   - SMX6_HDCP – may be defined to enable HCDP content protection protocol on HDMI interface.
4. The mx6qdl-smx6.dtsi file also defines several labels that could be used in carrier's device tree to define devices connected to module's buses or to refine device properties without need to alter module description fie:
   - smarc_i2c_cam, smarc_i2c_lcd, smarc_i2c_gp, smarc_i2c_hdmi, smarc_i2c_pm – for I2C buses;
   - smarc_spi0, smarc_spi1 – for SPI busses;
   - smarc_sdmmc, smarc_sdio - for SD/MMC interfaces,;
   - smarc_ser0, smarc_ser1, smarc_ser2, smarc_ser3 – for serial ports.

   Please refer to kontron-smarc-evaluation-carrier.dtsi for examples on how these labels could be used.
5. All example device tree tables (like uImage-imx6q-smx6-lcd.dtb), which were described above, use imx6qdl-smx6.dtsi with macros defined per file's names, and include kontron-smarc-evaluation-carrier.dtsi to describe devices on carrier board.
6. After creation of new dts file it should be placed into arch/arm/boot/dts/ directory in kernel sources. Once foo.dts is there, corresponding DTB file could be built by executing '`make foo.dtb`' from kernel tree top directory.

7.3. Deploying created images to SATA, SD card or USB flash/disk and booting from it

Note: Kernel and its modules are automatically built into rootfs image during build process, so only a device tree table should be manually integrated into this image.

1. Attach installation media(SATA, microSD card or USB flash/disk) to Linux host computer and determine a device node by analyzing system logs:
   ```
   dmesg | tail | grep sd
   ```
2. (optional) Repartition the installation media using tools like fdisk or parted and create at least one partition. First partiton size should be not less than 2Gb.
3. Format first partition on installation media to ext3 filesystem:
   ```
   mkfs.ext3 PARTITION_DEVICE
   ```

4. Mount first partition:
   ```
   mount PARTITION_DEVICE MOUNTPOINT
   ```
5. Extract the rootfs image to mounted partition. E.g:
   ```
   tar xjpf /path/to/core-image-sato-smarc-samx6i.tar.bz2 -C
   MOUNTPOINT
   ```
6. Copy an appropriate device tree table to root directory of extracted archive. E.g.:
   ```
   cp /path/to/uImage-imx6q-smx6-lcd.dtb MOUNTPOINT/
   ```
7. Unmount first partition and extract the installation media from host:
   ```
   umount PARTITION_DEVICE
   ```
8. Configure serial connection(see “5.5 U-Boot Access and Startup” in SMARC-sAMX6i User's Guide), then insert the installation media into SMARC-sAMX6 and power on/reset the board.
9. Interrupt the boot process and check U-boot environment variables with printenv. If needed, modify paths to kernel image and device tree table. E.g:
   ```
   setenv uimage /boot/uImage-smarc-samx6i.bin
   setenv fdtfile /uImage-imx6q-smx6-lcd.dtb
   ```
10. To boot the board, execute the following commands depending on which type of installation media is used:
    - For USB flash/disk(Please note that only USB #2 channel is supported by U-boot):
      ```
      setenv bootargs console=ttymxc0,115200 root=/dev/sda1 rw
      usb start
      usb dev 0
      ext2load usb 0:1 11000000 ${fdtfile}
      ext2load usb 0:1 10800000 ${uimage}
      bootm 10800000 - 11000000
      ```
    - For SATA(mSATA) disk:
      ```
      setenv bootargs console=ttymxc0,115200 root=/dev/sda1 rw
      sata init
      sata dev 0
      ext2load sata 0:1 11000000 ${fdtfile}
      ext2load sata 0:1 10800000 ${uimage}
      bootm 10800000 - 11000000
      ```
    - For SD(microSD) card:
      ```
      setenv bootargs console=ttymxc0,115200 root=/dev/mmcblk0 rw
      mmc dev 1
      ext2load mmc 1:1 11000000 ${fdtfile}
      ext2load mmc 1:1 10800000 ${uimage}
      bootm 10800000 - 11000000
      ```
    - For external eMMC device connected to SMARC eMMC interface:
      ```
      setenv bootargs console=ttymxc0,115200 root=/dev/mmcblk0 rw
      mmc dev 0
      ext2load mmc 0:1 11000000 ${fdtfile}
      ext2load mmc 0:1 10800000 ${uimage}
      bootm 10800000 - 11000000
      ```
11. After board booted up, login as user root. No password is needed.

Please refer to U-boot user's guide for more detailed instructions:
http://www.denx.de/wiki/DULG/Manual

7.4. Booting from NFS server

Note: Setting up NFS booting is highly dependent on Linux distribution used on host. For more details about setting up NFS, DHCP and TFTP daemons consult your distribution documentation.

1. On Linux host computer create a directory for NFS root (say /nfsroot) and extract the rootfs image into it. E.g:
   ```
   tar  xjpf /path/to/core-image-sato-smarc-samx6i.tar.bz2 -C
   /nfsroot
   ```
2. Then edit file /nfsroot/etc/network/interfaces and add following string at the end of file:
   ```
   iface eth0 inet manual
   ```
3. And add NFS root directory to /etc/exports and start NFS server.
4. Copy kernel from extracted rootfs image(/boot/uImage-smarc-samx6i.bin) into TFTP server root directory (/var/lib/tfptboot or /tftpboot depending on Linux distribution). Configure and start TFTP server.
5. Copy an appropriate device tree table(e.g. uImage-imx6q-smx6-lcd.dtb) into TFTP server root directory (/var/lib/tfptboot or /tftpboot depending on Linux distribution).
6. Configure DHCP server to provide IP configuration for SMARC-sAMX6 board. And start it.
7. Configure serial connection(see “5.5 U-Boot Access and Startup” in SMARC-sAMX6i User's Guide), then power on the board. Interrupt autoboot process by pressing any key on serial console and check U-boot environment variables with printenv . If needed, modify corresponding variables with setenv command and boot Linux from TFTP server on host by executing:
   ```
   setenv bootargs console=ttymxc0,115200 root=/dev/nfs rw ip=dhcp
   nfsroot=<host ip>:<path to unpacked rootfs on host>
   setenv serverip <host ip>
   setenv ipaddr <samx6i ip>
   setenv fdtfile <device tree BLOB copied to tftp>
   setenv uimage <kernle copied to tftp>
   tftp 11000000 ${fdtfile}
   tftp 0x10800000 ${uimage}
   bootm 0x10800000 - 11000000
   ```
   For example:
   ```
   setenv bootargs console=ttymxc0,115200 root=/dev/nfs rw ip=dhcp
   nfsroot=192.168.11.161:/opt/smarc,v3,tcp
   setenv serverip 192.168.11.161
   setenv ipaddr 192.168.11.189
   setenv fdtfile dtb3
   setenv uimage uImage
   tftp 11000000 ${fdtfile};
   tftp 0x10800000 ${uimage};
   bootm 0x10800000 - 11000000
   ```
8. After board booted up, login as user root. No password is needed.

Please refer to U-boot user's guide for more detailed instructions:
http://www.denx.de/wiki/DULG/Manual

7.5. Deploying created images to onboard eMMC and booting from it

Note: Kernel and its modules are automatically built into rootfs image during build process, so only a device tree table should be manually integrated into this image.

1. Boot SMARC-sAMX6 either from SATA, SD card, USB flash/disk or from NFS as was described above.
2. Insert another USB flash or SD card into Linux host and copy the rootfs image and appropriate device tree table to it. E.g.:
   ```
   mount PARTITION_DEVICE MOUNTPOINT
   cp /path/to/core-image-sato-smarc-samx6i.tar.bz2 MOUNTPOINT/
   cp /path/to/uImage-imx6q-smx6-lcd.dtb MOUNTPOINT/
   umount MOUNTPOINT/
   ```

3. Then attach the USB flash or SD card with the rootfs image to SMARC-sAMX6 and mount it:
```
mount USB_PARTITION_DEVICE MOUNTPOINT_EXT
```
4. Determine device node for onboard eMMC by analyzing fdisk and dmesg output:
```
dmesg
fdisk -l
```
5. (optional) Repartition the onboard eMMC using tools like fdisk or parted and create at least one partition. First partiton size should be no less than 2Gb.
6. Format first partition on install media to ext3 filesystem:
```
mkfs.ext3 EMMC_PARTITION_DEVICE
```
7. Mount first partition:
```
mount EMMC_PARTITION_DEVICE MOUNTPOINT_EMMC
```
8. Extract rootfs image to mounted partition:
```
tar xvjf MOUNTPOINT_EXT/core-image-sato-smarc-samx6i.tar.bz2 -C
MOUNTPOINT_EMMC
```
9. Copy the device tree table to root directory of extracted archive. E.g.:
```
cp MOUNTPOINT_EXT/uImage-imx6q-smx6-lcd.dtb MOUNTPOINT_EMMC/
```
10. Unmount mounted media:
```
umount MOUNTPOINT_EMMC
umount MOUNTPOINT_EXT
```
11. Configure serial connection(see “5.5 U-Boot Access and Startup” in SMARC-sAMX6i User's Guide), then insert the installation media into SMARC-sAMX6 and power on/reset the board.
12. Interrupt the boot process and check U-boot environment variables with printenv. If needed, modify paths to kernel image and device tree table. E.g:
```
setenv uimage /boot/uImage-smarc-samx6i.bin
setenv fdtfile /uImage-imx6q-smx6-lcd.dtb
```
13. To boot the board, execute the following commands:
```
setenv bootargs console=ttymxc0,115200 root=/dev/mmcblk0 rw
mmc dev 2
ext2load mmc 2:1 11000000 ${fdtfile}
ext2load mmc 2:1 10800000 ${uimage}
bootm 10800000 – 11000000
```
14. After board booted up, login as user root. No password is needed.

Please refer to U-boot user's guide for more detailed instructions:
http://www.denx.de/wiki/DULG/Manual

7.6. Booting kernel and device tree table from SPI flash

First of all, the kernel image and appropriate device tree table should be written to SPI flash. The easiest way to achieve this is to:

1. Boot SMARC-sAMX6 either from SATA, SD card, USB flash/disk, onboard eMMC or from NFS as was described above;

2. Determine where the binaries for the booted kernel and device tree table are located (usually they can be found in /boot and / directories).

3. Write these binaries to SPI flash with appropriate offset using dd command. E.g.:
```
# dd if=/dtb.ds.lvds of=/dev/mtdblock2 bs=1
47844+0 records in
47844+0 records out
47844 bytes (48 kB) copied, 1.4024 s, 34.1 kB/s
# dd if=/uImage of=/dev/mtdblock2 bs=1 seek=100000
5121712+0 records in
5121712+0 records out
5121712 bytes (5.1 MB) copied, 155.792 s, 32.9 kB/s
```

WARNING: if internal SPI flash is used to load U-Boot it's possible to brick the board if incorrect partitions and offsets are chosen!

4. Determine the size of written binaries in hex format. E.g.:

```
dtb.ds.lvds(47844 bytes):     BAE4
uImage(5121712 bytes):        4E26B0
```

5. Reboot the board and interrupt the boot process and check U-boot environment variables with printenv.

6. To boot the kernel and device tree table from the SPI, execute the following commands:

```
sf probe
setenv bootargs console=ttymxc0,115200
root=/dev/<device_with_rootfs> rw
sf read 11000000 <dtb_addr_on_spi> <dtb_size>
sf read 10800000 <kernel_addr_on_spi> <kernel_size>
bootm 10800000 – 11000000
```

For example:

```
sf probe
setenv bootargs console=ttymxc0,115200 root=/dev/sda1 rw
sf read 11000000 0xd0000 bae4  Note: 0xd0000 is an offset of
mtdblock2 partition on SPI flash.
sf read 10800000 0xe86a0 4e26b0  Note: 0xe86a0==0xd0000+0x186a0.
And 0x186a0 is an offset with which the uImage was written to
mtdblock2 partition(100000 in dec).
bootm 0x10800000 - 11000000
```

Please refer to U-boot user's guide for more detailed instructions:
http://www.denx.de/wiki/DULG/Manual

7.7. Integration of prebuilt binary kernel and device tree binaries into manually built rootfs image

As it was mentioned earlier it's not possible to provide a prebuilt rootfs image due to legal restrictions imposed by FREESCALE SOFTWARE LICENSE AGREEMENT. Thus only binaries for kernel, kernel modules and device tree tables are provided in the smarc-samx6i-kernel-and-device-tree-binaries-R01.00.tar.gz archive. It's recommended to integrate these images into manually built rootfs image only when it's needed to verify some SMARC-sAMX6 functionality. In all other cases kernel and device tree tables that were built alongside rootfs image should be used.

The following instruction describes how to integrate the prebuilt kernel and device tree table into manually built rootfs:

1. Attach installation media(SATA, microSD card or USB flash/disk) to Linux host computer and determine a device node by analyzing system logs:

```
dmesg | tail | grep sd
```

2. (optional) Repartition the installation media using tools like fdisk or parted and create at least one partition. First partiton size should be no less than 2Gb.

3. Format first partition on installation media to ext3 filesystem:

```
mkfs.ext3 PARTITION_DEVICE
```

4. Mount first partition:

```
mount PARTITION_DEVICE MOUNTPOINT
```

5. Extract the rootfs image to mounted partition. E.g:

```
tar xjpf /path/to/core-image-sato-smarc-samx6i.tar.bz2 -C
MOUNTPOINT
```

6. Extract the archive with prebuilt binaries to some directory:

```
tar xzpf /path/to/smarc-samx6i-kernel-and-device-tree-binaries-
R01.00.tar.gz -C SOME_DIRECTORY
```

7. Copy prebuilt kernel and appropriate device tree table to the partition with extracted rootfs image. E.g.:

```
cp SOME_DIRECTORY/uImage-smarc-samx6i.bin MOUNTPOINT/boot/
cp SOME_DIRECTORY/uImage-imx6q-smx6-lcd.dtb MOUNTPOINT/
```

8. Install modules for the previously copied kernel. E.g.:

```
tar xzpf SOME_DIRECTORY/modules-smarc-samx6i.tgz -C MOUNTPOINT/
```

9. Unmount first partition and extract the installation media from host:

```
umount PARTITION_DEVICE
```

10. Now it's possible to connect this installation media to SMARC-sAMX6, configure U-oot environment variables and boot the board according to instructions provided in the corresponding chapter above.

8. Implementation notes

8.1. Display

SMARC-sAMX6 provides hardware control for two displays – primary display over LCD or LVDS, and secondary display over HDMI. Using LCD and LVDS ports to drive different displays at the same time is technically possible but not supported by this BSP.

Primary display port should be selected by defining either SMX6_PANEL_LCD or SMX6_PANEL_LVDS when building device tree.

Framebuffer device /dev/fb0 corresponds to primary display's background, framebuffer device /dev/fb1 corresponds to primary display's overlay layer.

Display parameters are configured in device tree. This release defaults to 32bpp framebuffer, 24-bit hardware interface, 1280x800 panel resolution, 60 Hz refresh rate. These could be changed either directly inside imx6qdl-smx6.dtsi, or in local dts file by overriding properties in &mxcfb1 and/or &ldb nodes, e.g.:

```
&mxcfb1 {
        mode_str = "640x480M@60";
};
```

See Documentation/fb/modedb.txt file in kernel sources for video mode definition syntax.

To use 18-bit hardware interface, use below device tree code:

```
&mxcfb1 {
        interface_pix_fmt = "RGB666";
};
/lcd {
        default_ifmt = “RGB666”;
};
```

Also, depending on carrier board used, it could be required to set up some jumpers. Refer to carrier's board's documentation for details.

Framebuffer device /dev/fb2 corresponds to secondary display's background, framebuffer device /dev/fb3 corresponds to secondary display's overlay layer. Secondary display's overlay layer is available only on imx6q/imx6d based modules where dedicated IPU is used for secondary display; on imx6dl/imx6s based module there is only one IPU that can drive only one overlay layer.

HDMI modes are detected automatically using EDID protocol. However, freescale's HDMI driver has a mode filter that only allows modes defined by CEA-861 standard. This includes modes used by SD- and HD video, but not modes used by legacy computer displays. This could cause older displays connected over hdmi-to-dvi converter to run at 640x480 resolution.

Support for HDCP content protection protocol is available but untested and disabled by default. To enable it, device tree should be rebuilt with SMX6_HDCP macro defined.

There are two backlight devices defined for primary display:

```
# ls /sys/class/backlight/
backlight.17 lcdpower.18
```

Device named 'backlight.17' is pwm-backlight. It has sysfs attribute 'brightness' that could be used to get or change current backlight brightness. Brightness is an integer in 0...100 range. When brightness is set to non-zero value, SMARC LCD_BLKT_EN line is automatically asserted; when brightness is set to zero value, SMARC LCD_BLKT_EN line is automatically deasserted .

Device named 'lcdpower.18' is used to control SMARC LCD_VDD_EN line. It's 'brightness' attribute accepts only two values – 0 to disable LCD_VDD_EN and 1 to enable LCD_VDD_EN.

Both backlight devices are connected to primary display and automatically disable their output when primary display is blanked or closed.

8.2. SPI

Devices connected to SPI must be described in device tree, in appropriate sections. To support that, imx6qdl-smx6.dtsi defines 'smarc_spi0' and smarc_spi1' labels. With these, device definition looks like :

```
&smarc_spi0 {
        <device>@<cs> {
                reg = <cs>;
                /* other attributes */
        };
};
```

See internal flash definition in imx6qdl-smx6.dtsi as an example.

Chip selects configured in device tree should match physical configuration:

1. For SPI1, chip selects should be used as-is: 0 for SMARC CS0, 1 for SMARC CS1.
2. For SPI0, situation is more complex:
   - If SMARC BOOT_SEL[2:0] inputs are set to '011' (that means “boot from external SPI flash”), then internal SPI flash is hidden, and SMARC chip selects should be used as-is; device tree for this configuration should be built with SMX6_HIDE_SPI_FLASH macro defined;
   - If SMARC BOOT_SEL[2:0] inputs are set differently, then chip select 0 selects internal SPI flash, chip select 1 selects SMARC CS0, and chip select 2 selects SMARC CS1; device tree for this configuration should be built with SMX6_HIDE_SPI_FLASH macro not defined.

8.3. I2C

Devices connected to I2C buses must be described in device tree, in appropriate sections. To support that, imx6qdl-smx6.dtsi defines 'smarc_i2c_*' labels. With these, device definitions look like:

```
&smarc_i2c_pm {
        at24@50 {
                compatible = "at24,24c32";
                pagesize = <8>;
                reg = <0x50>;
        };
};
```

Please use this method, do not rely on i2c bus numbers – numbers may change.

Remember that Linux uses 7-bit i2c addresses (i.e. without r/w bit attached at right).

8.4. USB

SMARC USB1 and USB2 are enabled as USB hosts, and SMARC USB0 is enabled as USB OTG.

USB OTG interface role is determined automatically based on cable type connected. When micro-A cable is connected, system dynamically registers an USB host. When micro-A cable is disconnected, system automatically unregisters that host. When micro-B cable is connected, system works as USB device, device type depends on current usb gadget module loaded. E.g. g_mass_storage.ko must be loaded to provide mass storage device functionality. If none usb gadget module is loaded, system won't respond on micro-B cable connection.

Since SMARC-sAMX6 hardware has non-standard hardware connection of USB overcurrent detection signal, overcurrent detection is not supported.

8.5. PCIe

SMARC connector has 3 PCIe interfaces (pcie-a, pcie-b and pcie-c). However, only pcie-a is utilized by SMARC-sAMX6 module (unless module has optional PCIe switch installed). This means that on Kontron SMARC evaluation carrier, only one mPCIe connecter is usable, another mPCIe connector and PCIe 1x slot are not.

PCIe devices connected to available interface should be visible in 'lspci' output and work as soon as device driver is available in the kernel (statically or over module load).

For modules with PCIe switch, device tree must be built with SMX6_PCIE_SWITCH macro defined. Without that PCIe won't work properly.

8.6. GPIO

SMARC GPIO0..GPIO11 lines are mapped to Linux gpio64..gpio75. These GPIOs could be controlled over sysfs, as documented in Documentation/gpio.txt file in kernel sources. For example, following shell script blinks carrier's LED connected to GPIO10 line:

```
echo 74 > /sys/class/gpio/export
while true; do
      echo low > /sys/class/gpio/gpio74/direction
      sleep 1
      echo high > /sys/class/gpio/gpio74/direction
      sleep 1
done
```

Known issue: on some module/carrier combinations, reading values of some GPIOs configured as output do not match programmed GPIO value. However, physical GPIO state is ok (i.e. as programmed).

8.7. Suspend/resume

To move system into low power mode, mode name should be written to /sys/power/state attribute.

Supported modes as 'standby' and 'mem'. Former corresponds to i.MX 6 stop mode, later corresponds to i.MX 6 deep sleep mode. See Freescale's documentation at http://www.freescale.com/infocenter/topic/i.MX6QLXRM/3523887.html for more information.

Any device capable of generating interrupts could be configured as wakeup source, by altering device's power/wakeup attribute in sysfs. See Documentation/power/devices.txt file in kernel sources for more information. By default, power, sleep and lid buttons are configured as wakeup sources.

In this release, suspending system won't drive SMARC CARRIER_STBY# line low.

9. Known issues

- Not supported:
  - I2S Audio Codec and SPDIF: corresponding wrapper for the driver of I2S Codec has not been ported yet to the BSP kernel;
  - CAN: very probable it's a h/w issue as as similar behavior was observed earlier with similar carrier boards. Also CAN doesn't work on existing h/w with old Linux BSP, where this functionality was successfully verified;
  - KEAPI;
  - Battery Charger BQ24171;
  - MLB150 serial interface;
  - CARRIER_STBY# signal/pin is not driven low during suspend;
  - WDT_TIME_OUT# signal/pin. When pinmuxing for this pin is properly configured, the board resets all the time. Very probably it's caused by “Power on Timer” feature of i.MX 6: BSP doesn't update this register for 16 seconds, so the WDT_TIME_OUT# line automatically became active and CPLD resets the board as this signal is routed to it.
  - Force recovery jumper: this jumper should be used only for U-boot recovery;
  - Dual Display mode for X server is not supported by the video driver: https://community.freescale.com/thread/311751. Wayland should be used instead;
  - SIM slot is not functional. Only GSM modems with integrated slot for SIM card (like Novatel EU850D GSM modem) are supported.
- Not tested:
  - SPDIF, AUDIO_MCLK, I2S0, I2S1, I2S2 buses: lack of the I/O mezzanine. These buses are also routed to the I2S codec, which is not supported in this release too;
  - CSI Camera(CAM0 and CAM1 buses): corresponding driver has been integrated into the BSP but there are no mezzanine and camera sensor to test with;
  - Carrier eMMC: lack of eMMC mezzanine;
  - HDCP: lack of appropriate hardware and content;
  - PCIE_WAKE# signal;
  - SLEEP# signal: lack of I/O mezzanine;
  - RESET_OUT#, RESET_IN# are routed to the CPLD and can't be controlled by software.
- Linux kernel fails to boot on SMARC sAMX6i module with 2GB of RAM. To work around this issue please change the address where FDT will be stored by executing the following command in U-Boot:

```
setenv fdt_high 0x20000000
```

- Low network performance for Gigabit Ethernet adapter: 200Mbits/sec for TCP and 300Mbits/sec for UDP.
- Power is not turned off after software shutdown: the problem has not been investigated yet. The same situation was observed with WEC7 BSP.
- On some module/carrier combinations, reading values of some GPIOs configured as output do not match programmed GPIO value. However, physical GPIO state is ok (i.e. as programmed).
- Not all 18-bit panels work correctly, sometimes colors are broken (this may be caused by invalid jumper setting and|or invalid device tree properties).
- Vivante video driver has a limited support of OpenGL(some functions like GLX_EXT_swap_control are not supported and the perfromance is poor): https://lists.yoctoproject.org/pipermail/meta-freescale/2014-April/008254.html
- AIGLX(accelerated indirect glx) is not supported by the Vivante video driver: https://community.freescale.com/message/312735#312735

- Unstable work of USB OTG driver:
  - Mass Storage Gadget: kernel sometimes hangs during intense file transfer when is used;
  - Ethernet Gadget: Very often a network connection to host is lost during UDP stress test in CDC EEM mode.

10. Changelog

R01.00: Initial release.