Support SPI in the Linux i.MX RT1170 BSP

Detailed Requirements and Design

rm6911-drad-1_1.doc

TABLE OF CONTENTS

1. Overview

The following is a high-level overview of the problem being resolved by this project:

This project provides support for the LPSPI controller of the i.MX RT1170 processor in the Linux BSP.

2. Requirements

2.1. Detailed Requirements

The following are the requirements for this project:

1. Provide a Linux demo project combining all the requirements in this project.
    - *Rationale*: Needed to let Customer integrate results of this project into target embedded application.
      Implementation: Section: "Design: Demo project".
      Test: Section: "Test Plan: Demo Project".

2. Provide support for the i.MX RT1170 LPSPI controller in Linux.

    - *Rationale*: Explicit Customer requirement.
      Implementation: Section: "Design: Linux LPSPI Device Driver".
      Test: Section: "Test Plan: Linux LPSPI Driver".

3. Provide support for the raw access to the SPI device from user space.

    - *Rationale*: Explicit Customer requirement.
      Implementation: Section: "Design: Raw SPI Device Access".
      Test: Section: "Test Plan: Raw SPI Device Access".

4. Provide support for the standard Linux SPI Flash interface.

    - *Rationale*: Explicit Customer requirement.
      Implementation: Section: "Design: Standard Linux MTD interface to the SPI Flash ".
      Test: Section: "Test Plan: Standard Linux MTD interface to the SPI Flash ".

2.2. Detailed Non-Requirements

The following are the non-requirements for this project that may otherwise not be obvious:

- None

3. Design

3.1. Detailed Design

3.1.1. Design: Demo project

This project will enable the required functionality in the Linux configuration ("embedded project") called `rootfs`, which resides in a `projects/rootfs` directory, relative to the top of the Linux i.MX RT1170 installation.

3.1.2. Design: Linux LPSPI Device Driver

In the i.MXRT1170 SoC, the LPSPI controller is compatible with the same controller of some other SoCs from the i.MX family, so the existing spi-fsl-lpspi.c driver in the Linux sources, which was initially added to support SPI in the i.MX7ULP SoC, will be used to support the LPSPI controller of the i.MXRT1170 SoC.

To enable the driver, the standard CONFIG_SPI along with the CONFIG_SPI_FSL_LPSPI options will be defined in the kernel defconfig file.

In addition to the kernel defconfig, the user must configure the LPSPI controller in DTS. Refer to https://www.kernel.org/doc/Documentation/devicetree/bindings/spi/spi-controller.yaml for details on format of the SPI nodes.

The DTS nodes for all 6 instances of the i.MXRT1170 LPSPI controller will be predefined in arch/arm/boot/dts/imxrt1170.dtsi. The clock driver for the i.MXRT1170 SoC will be updated to support the LPSPI clocks. Interrupts and clocks will be defined in the DTS nodes according to the processor reference manual. All controllers will be configured to use DMA. All the SPI nodes in the .dtsi file will be disabled by default.

Final tuning of the kernel run-time configuration, such as defining the pinctrl settings for a custom connection of an SPI device to the LPSPI controller, defining the chip-select signal, customizing the clocks etc, will be done in the user DTS file.

This project will provide an example for configuring the LPSPI1 controller. On the IMXRT1170-EVK board there is a 512KB SPI Flash device connected to this controller. The controller will be defined in the rootfs.dts.IMXRT117X_NXPEVK file as follows:

```
&lpspi1 {
       pinctrl-names = "default";
       pinctrl-0 = <&pinctrl lpspi1>;
       status = "okay";

       assigned-clocks = <&clks IMXRT1170 CLK ROOT LPSPI1>;
       assigned-clock-parents = <&clks IMXRT1170 CLK PLL2 PFD3>;
       assigned-clock-rates = <99000000>;

       cs-gpios = <&gpio9 28 GPIO_ACTIVE_LOW>;
...
};
...
&iomuxc {
...
       pinctrl lpspi1: lpspi1grp {
               fsl,pins = <
                       IOMUXC_GPIO_AD_28_LPSPI1_SCK                    (MXRT1170_PAD_DSE)
                       IOMUXC GPIO AD 29 GPIO9 IO28                    (MXRT1170 PAD DSE)
                       IOMUXC GPIO AD 30 LPSPI1 SOUT                   (MXRT1170 PAD DSE)
                       IOMUXC GPIO AD 31 LPSPI1 SIN                    (MXRT1170 PAD DSE)
               >;
       };
};
```

This example assigns the i.MXRT1170 pinctrl PADs GPIO_AD_28, GPIO_AD_29, GPIO_AD_30 and GPIO_AD_31 to the LPSPI1 controller and selects the LPSPI1_SCK, GPIO, LPSPI1_SOUT and LPSPI1_SIN functions for these PADs (refer to the board schematics and to the processor reference manual for details on possible IOMUXC configurations).

The example redefines the source for the root clock of the LPSPI1 to PLL2_PDF3. Divider of the root clock is set to generate an 99MHz output frequency (refer to the processor reference manual for details on possible CCM configurations).

The single `chip-select` is configured to the `GPIO9 28` signal in the example.

3.1.3. Design: Raw SPI Device Access

Linux provides a special `spidev` device driver to allow raw accesses to SPI devices from the user space: https://www.kernel.org/doc/Documentation/spi/spidev.

The standard Linux `CONFIG_SPI_SPIDEV` option will be defined in the kernel `defconfig` file to enable the `spidev` driver. The universal compatibility string "linux,spidev" will be added to the `spidev` driver so that it can be used in the project `DTS`.

The following changes will be made to `rootfs.dts.IMXRT117X_NXPEVK` to link the SPI Flash on the `LPSPI1` controller to the `spidev` Linux device:

```
#define LPSPI_USE_SPIDEV
&lpspi1 {
...
#if defined(LPSPI USE SPIDEV)
       spidev: spidev@0 {
               status = "okay";
               compatible = "linux,spidev";
               spi-max-frequency = <33000000>;
               reg = <0>;
       };
...
#endif
};
```

⚠Note that the `spidev` sub-node is defined under the `#if defined(LPSPI_USE_SPIDEV)` condition. This is due to the fact that this projects also provides an alternative way to access the same SPI Flash via the Linux `MTD` interface (refer to the section below), so that the user can switch between 2 interfaces: raw `spidev` or `MTD`, by defining or un-defining the `LPSPI_USE_SPIDEV` pre-processor macro in the `DTS` file.

The following test program `spidev_flash` will be included to the project root file system as an example on how to read the Flash ID of an SPI Flash device from the user-space applications:

```
/*
 * Sample application that makes use of the SPIDEV interface
 * to access an SPI slave device. Specifically, this sample
 * reads a Device ID of a JEDEC-compliant SPI Flash device.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

int main(int argc, char **argv)
{
        char *name;
        int fd;
        struct spi_ioc_transfer xfer[2];
        unsigned char buf[32], *bp;
        int len, status;

        name = argv[1];
```

```
        fd = open(name, O_RDWR);
        if (fd < 0) {
                perror("open");
                return 1;
        }

        memset(xfer, 0, sizeof xfer);
        memset(buf, 0, sizeof buf);
        len = sizeof buf;

        /*
         * Send a GetID command
         */
        buf[0] = 0x9f;
        len = 6;
        xfer[0].tx_buf = (unsigned long)buf;
        xfer[0].len = 1;

        xfer[1].rx_buf = (unsigned long) buf;
        xfer[1].len = 6;

        status = ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
        if (status < 0) {
                perror("SPI_IOC_MESSAGE");
                return -1;
        }

        printf("response(%d): ", status);
        for (bp = buf; len; len--)
                printf("%02x ", *bp++);
        printf("\n");

        return 0;
}
```

3.1.4. Design: Standard Linux MTD interface to the SPI Flash

The standard interface to the SPI Flash in Linux is the Memory Technology Device (`MTD`), on top of which the Flash file system such as `UBIFS` or `JFFS2` is usually used.

The following options will be enabled in the project kernel `defconfig` file to enable support for the `MTD` interface and the `JFFS2` file system: `CONFIG_MTD`, `CONFIG_MTD_BLOCK`, `CONFIG_MTD_SPI_NOR` and `CONFIG_JFFS2_FS`.

The flash node will be defined in the `DTS` as follows:

```
#define LPSPI_USE_SPIDEV
/* uncomment the following line to enable SPI Flash support via the Linux MTD interface */
// #undef LPSPI_USE_SPIDEV
&lpspi1 {
...
#if defined(LPSPI_USE_SPIDEV)
...
#else
        flash0: mx25l1400@0 {
                reg = <0>;
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "jedec,spi-nor";
                spi-max-frequency = <33000000>;
                partition@0 {
                        label = "data";
                        reg = <0x0 0x80000>;
                };
        };
#endif
};
```

⚠Note that the SPI Flash is configured for the `spidev` driver by default. To enable access to the SPI Flash via `MTD`, the user must un-define the `LPSPI_USE_SPIDEV` macro in `rootfs.dts.IMXRT117X_NXPEVK` and rebuild the project.

3.2. Effect on Related Products

This project makes the following updates in the related products:

- None

3.3. Changes to User Documentation

This project updates the following user documents:

- None

3.4. Alternative Design

The following alternative design approaches were considered by this project but then discarded for some reason:

- None

4. Test Plan

4.1. Secure Download Area

The downloadable materials developed by this project are available from a secure Web page on the Emcraft Systems web site. Specifically, proceed to the following URL to download the software materials:

for the i.MX RT1170 BSP:

https://www.emcraft.com/imxrtaddon/imxrt1170/lpspi

The page is protected as follows:

- Login: *`CONTACT EMCRAFT FOR DETAILS`*
- Password: *`CONTACT EMCRAFT FOR DETAILS`*

4.2. Downloadable Files

The following files are available from the secure download area:

- `linux-lpspi.patch` - patch to the Linux kernel sources;
- `projects-lpspi.patch` - patch to the `rootfs` project;
- `rootfs.uImage` - prebuilt bootable Linux image;

4.3. Test Set-Up

4.3.1. Hardware Setup

The following hardware setup is required for the i.MX RT 1170 board:

- The i.MXRT1170 EVK board.

4.3.2. Software Setup

The following software setup is required:

1. Download the files listed in Section: "Downloadable Files" to the top of the Linux i.MX RT installation.
2. Install the BSP, as per the respective "Installing and activating cross development environment" document in the "Software" section on the Emcraft site.
3. From the top of the Linux installation, activate the Linux cross-compile environment by running:

```
$ . ACTIVATE.sh
```

4. From the top of the BSP installation, go to the Linux kernel tree and install the kernel patch, eg:

```
$ cd linux/
$ patch -p1 < ../../linux-lpspi.patch
```

5. From the top of the Linux installation, go to the `projects` sub-directory, and patch the `rootfs` project:

```
$ cd projects/
$ patch -p1 < ../../projects-lpspi.patch
```

6. Build the `rootfs` project:

```
$ cd projects/rootfs
$ make
```

4.4. Detailed Test Plan

4.4.1. Test Plan: Demo Project

Perform the following step-wise test procedure:

1. Go to the `projects/rootfs` directory, build the loadable Linux image (`rootfs.uImage`) and copy it to the TFTP directory on the host:

```
$ cd projects/rootfs
$ make
```

2. Boot the loadable Linux image (`rootfs.uImage`) to the target via TFTP and validate that it boots to the Linux shell:

```
=> run netboot
Using ethernet@40424000 device
TFTP from server 192.168.1.96; our IP address is 192.168.1.86
Filename 'imxrt1170/rootfs.uImage'.
Load address: 0x80007fc0
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
...
/ # uname
Linux
/ #
```

4.4.2. Test Plan: Linux LPSPI Driver

Perform the following step-wise test procedure:

1. Run `dmesg` and verify that there are no error messages related to SPI or LPSPI:

```
/ # dmesg
...
/ # dmesg | grep spi
/ #
```

2. Verify that `spi0` master is registered in the `sysfs`:

```
/ # ls /sys/class/spi_master/
spi0
/ #
```

3. Mount `debugfs` and verify the `lpspi1_root` clock operates at 99MHz frequency:

```
/ # mount -t debugfs debugfs /sys/kernel/debug/
/ # cat /sys/kernel/debug/clk/lpspi1 root/clk parent
pll2_pfd3
/ # cat /sys/kernel/debug/clk/lpspi1_root/clk_rate
99000000
/ #
```

4.4.3. Test Plan: Raw SPI Device Access

1. Make sure the `LPSPI_USE_SPIDEV` macro is defined in the `rootfs.dts.IMXRT117X_NXPEVK` file. Rebuild and reinstall the project. Run the project on the target board.
2. Make sure the `spidev` device exists:

```
/ # ls -l /dev/spidev0.0
crw-------    1 root     root      153,   0 Jan  1 00:00 /dev/spidev0.0
/ #
```

3. Make sure the `spidev_flash` application reads the correct Flash ID over SPI (first 3 bytes must be `0xc2 0x20 0x13` for the `MX25L4006E` Flash installed at the IMXRT1170-EVK board):

```
/ # spidev flash /dev/spidev0.0
response(7): c2 20 13 c2 20 13
/ #
```

4.4.4. Test Plan: Standard Linux MTD interface to the SPI Flash

1. Make sure the `LPSPI_USE_SPIDEV` macro is undefined in the `rootfs.dts.IMXRT117X_NXPEVK` file. Rebuild and reinstall the project. Run the project on the target board.
2. Make sure the `mtd` devices exist:

```
/ # ls /dev/mtd*
/dev/mtd0       /dev/mtd0ro     /dev/mtdblock0
/ #
```

3. Format and mount the SPI Flash device:

```
/ # flash eraseall -j /dev/mtd0
Erasing 4 Kibyte @ 0 -  0% complete.random: crng init done
Erasing 4 Kibyte @ 80000 - 100% complete.Cleanmarker written at 7f000.
/ # mount -t jffs2 /dev/mtdblock0 /mnt/flash/
/ #
```

4. Save the `busybox` binary to the SPI Flash:

```
/ # cp /bin/busybox /mnt/flash/
/ #
```

5. Reboot the target board and make sure the saved `busybox` file is intact :

```
/ # reboot
The system is going down NOW!
...
Starting kernel ...
Booting Linux on physical CPU 0x0
...
```

```
/ # mount -t jffs2 /dev/mtdblock0 /mnt/flash/
/ # /mnt/flash/busybox echo "Hello with busybox from SPI Flash"
Hello with busybox from SPI Flash
/ #
```