Category:SPI

Contents

Stable: 17.06.2020 - 15:26 / Revision: 16.01.2020 - 08:41

Category:SPI

Stable: 17:06.2020 - 10:20 | Revision: 16:01.2020 - 08:41

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to the Linux® **SPI** software framework.

It is recommended to first read the SPI overview article.

Serial Peripheral Interface

Pages in category "SPI"

The following 3 pages are in this category, out of 3 total.

- How to use SPI from Linux userland with spidev
- SPI device tree configuration
- SPI overview

Stable: 26.11.2020 - 13:10 / Revision: 26.11.2020 - 11:31

A quality version of this page, approved on *26 November 2020*, was based off this revision.

Contents

1 Article purpose

Linux®SPI framework offers several ways to access SPI peripherals. Among them, the spidev framework enables to easily control an SPI peripheral straight from Linux® user space.
Before going further in this document, the reader might be interested in having a look at the SPI overview article that describes how to use an SPI when the peripheral is assigned to Linux®:

- How to configure an SPI device through the board device tree (example using "spidev")
- How to perform data transfers in userland

2 SPI hardware tests (loopback MOSI / MISO)

Short-circuit the SPI bus MISO and MOSI lines to create a loopback enables the bus to receive the same data it is sending. This is an interesting solution to quickly perform basic tests as well as performance tests.

On the STM32MP157X-DKX discovery board, MOSI and MISO signals are accessible via the D12 and D11 pins of the STM32MP157x-DKx_-_hardware_description#Arduino_Uno_connector.

3 DT and kernel configuration

To be able to control the SPI device from Linux® user space, the *User mode SPI device driver support* must be enabled. Its configuration is described in the SPI_overview#Kernel_configuration.
In addition, the device tree must be customized to expose the SPI peripheral via the spidev framework. The overall device tree configuration is described in SPI_device_tree_configuration.

In our example of MOSI/MISO loopback on the STM32MP157X-DKX Discovery board, the stm32mp157a-dk1.dts[1] device tree, which already includes the board skeleton for spi4, must be customized as follows:

- Activate the SPI controller by setting its status to *okay*.
- Add a spidev child node.
  - Enable spidev by adding a compatible *spidev*.
  - Add a **reg** property, required for the SPI framework but not meaningful in this case since chip select is not defined and loopback is used.
  - Configure the bus speed for SPI communications by setting the **spi-max-frequency** property.

```
&spi4 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&spi4_pins_a>;
    pinctrl-1 = <&spi4_sleep_pins_a>;
    status = "okay";

    spidev@0{
        compatible = "spidev";
        reg = <0>;
        spi-max-frequency = <4000000>;
    };
};
```

4 SPI unitary tests using spidev_test

spidev_test, available within the Linux® kernel, is a test tool enabling to perform tests via the spidev interface.

4.1 Source code

The Linux® kernel spidev_test tool source code can be found under tools/spi[2] directory:

- tools/spi/spidev_test.c

4.2 Installation on your target

The Linux® kernel SPI tools are not embedded by default in OpenSTLinux distribution. They can be compiled independently and then installed on the target (see How to build Linux kernel user space tools).

4.3 List of spidev options

The *spidev_test* tool has the following options:

```
Board $> spidev_test -h
Usage: spidev_test [-DsbdlHOLC3vpNR24SItx]
  -D --device   device to use (default /dev/spidev1.1)
  -s --speed    max speed (Hz)
  -d --delay    delay (usec)
  -b --bpw      bits per word
  -i --input    input data from a file (e.g. "test.bin")
  -o --output   output data to a file (e.g. "results.bin")
  -l --loop     loopback
  -H --cpha     clock phase
  -O --cpol     clock polarity
  -L --lsb      least significant bit first
  -C --cs-high  chip select active high
  -3 --3wire    SI/SO signals shared
  -v --verbose  Verbose (show tx buffer)
  -p            Send data (e.g. "1234\xde\xad")
  -N --no-cs    no chip select
  -R --ready    slave pulls low to pause
  -2 --dual     dual transfer
  -4 --quad     quad transfer
  -S --size     transfer size
  -I --iter     iterations
  -t --txonly   simplex tx transfer
  -r --rxonly   simplex rx transfer
```

4.4 Example of 32-byte transfer in Full-duplex with loopback

```
Board $> spidev_test -D /dev/spidev0.0 -v
                        spi mode: 0x0
                        bits per word: 8
```

```
                        max speed: 500000 Hz (500 KHz)
                        TX | FF FF FF FF FF FF 40 00 00 00 00 95 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF F0 0D  | ......@.... .................. .
                        RX | FF FF FF FF FF FF 40 00 00 00 00 95 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF F0 0D  | ......@.... .................. .
```

5 References

- arch/arm/boot/dts/stm32mp157a-dk1.dts
- tools/spi , Linux®SPI tools source code

Linux® is a registered trademark of Linus Torvalds.

Serial Peripheral Interface

Device Tree

Transmit

Receive

Stable: 27.01.2021 - 16:26 / Revision: 27.01.2021 - 16:22

A quality version of this page, approved on *27 January 2021*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the *SPI internal peripheral*[1] **when the peripheral is assigned to Linux®OS**, and in particular:

- how to configure the STM32 SPI peripheral
- how to configure the STM32 external SPI devices present either on the board or on a hardware extension.

The configuration is performed using the **device tree mechanism**[2].

It is used by the *STM32 SPI Linux® driver* that registers relevant information in the SPI framework.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The SPI bus and its associated device are represented by:

- The *Generic device tree bindings for SPI buses*[3]
- The *STM32 SPI controller device tree bindings*[4]

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

At device level, each SPI controller is declared as follows:

```
spi1: spi@44004000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "st,stm32h7-spi";
        reg = <0x44004000 0x400>;
        interrupts = <GIC_SPI 35 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&rcc SPI1_K>;
        resets = <&rcc SPI1_R>;
        dmas = <&dmamux1 37 0x400 0x05>,
               <&dmamux1 38 0x400 0x05>;
        dma-names = "rx", "tx";
        power-domains = <&pd_core>;
        status = "disabled";
};
```

Warning

This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.

Refer to the DTS file: stm32mp151.dtsi[5]

3.2 DT configuration (board level)

```
&spi1 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&spi1_pins_a>;
        pinctrl-1 = <&spi1_sleep_pins_a>;
        cs-gpios = <&gpioz 3 0>;
        status = "okay";

        foo@0 {
                compatible = "spi-foo";
                reg = <0>; /* CS #0 */
                spi-max-frequency = <10000000>;
        };
};
```

There are two levels of configuration:

- Configuration of the SPI bus properties:
  - **pinctrl-0&1** configuration depends on hardware board configuration and on how the SPI devices are connected to MOSI, MISO and Clk pins.
    More details about pin configuration are available here: Pinctrl device tree configuration
  - **cs-gpios** represents the list of GPIOs used as chip selects. Since ecosystem release v2.1.0 ⓘ, this property is optional. Prior to ecosystem release v2.1.0 ⓘ, native controller chip select defined by a NULL value was not supported by STM32MP1 SPI driver. More details about GPIO configuration are available here: GPIO device tree configuration
  - **dmas**: by default, DMAs are specified for all SPI instances. This is up to the user to **remove** them if they are not needed. ***/delete-property/*** is used to remove DMA usage for SPI. Both **/delete-property/dma-names** and **/delete-property /dma** have to be inserted to get rid of DMAs.
- Configuration of the properties of the SPI device connected on the bus:
  - **compatible** represents the name of the SPI device driver.
  - **reg** represents the index of the gpio chip select associated to this SPI device.
  - **spi-max-frequency** represents the maximum SPI clocking speed for the device (in Hz).

For more information about SPI bus and SPI device bindings, please refer to spi-controller.yaml[3]

3.3 DT configuration example

Example: of an external TPM device:

```
&spi1 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&spi1_pins_a>;
        pinctrl-1 = <&spi1_sleep_pins_a>;
        cs-gpios = <&gpioz 3 0>;
        status = "okay";

        st33zp24@0 {
                compatible = "st,st33htpm-spi";
                reg = <0>; /* CS #0 */
                spi-max-frequency = <10000000>;
        };
};
```

The above example registers a TPM device on spi1 bus, selected by chip select 0 also known as GPIO-Z3. This instance is compatible with the driver registered with the same compatible property (st,st33htpm-spi).

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

Please refer to the following links for additional information:

- SPI internal peripheral
- Device tree
- 3.03.1 Documentation/devicetree/bindings/spi/spi-controller.yaml , Generic device tree bindings for SPI buses
- Documentation/devicetree/bindings/spi/spi-stm32.txt
- arch/arm/boot/dts/stm32mp151.dtsi

Serial Peripheral Interface

Linux® is a registered trademark of Linus Torvalds.

Operating System

Device Tree

Generic Interrupt Controller

Device Tree Source (in software context) or Digital Temperature Sensor (in peripheral context)

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Direct Memory Access

Trusted Platform Module

Stable: 22.02.2021 - 10:09 / Revision: 22.02.2021 - 08:28

A quality version of this page, approved on *22 February 2021*, was based off this revision.

This article gives basic information about the Linux®SPI framework and STM32 SPI driver installation. It explains how to use the SPI and more specifically:

- how to activate the SPI interface on a Linux®BSP
- how to access the SPI from kernel space
- how to access the SPI from user space.

Warning

While the STM32 SPI controller supports both master and slave modes, the STM32 Linux driver currently only supports SPI master mode.

Contents

1 Framework purpose

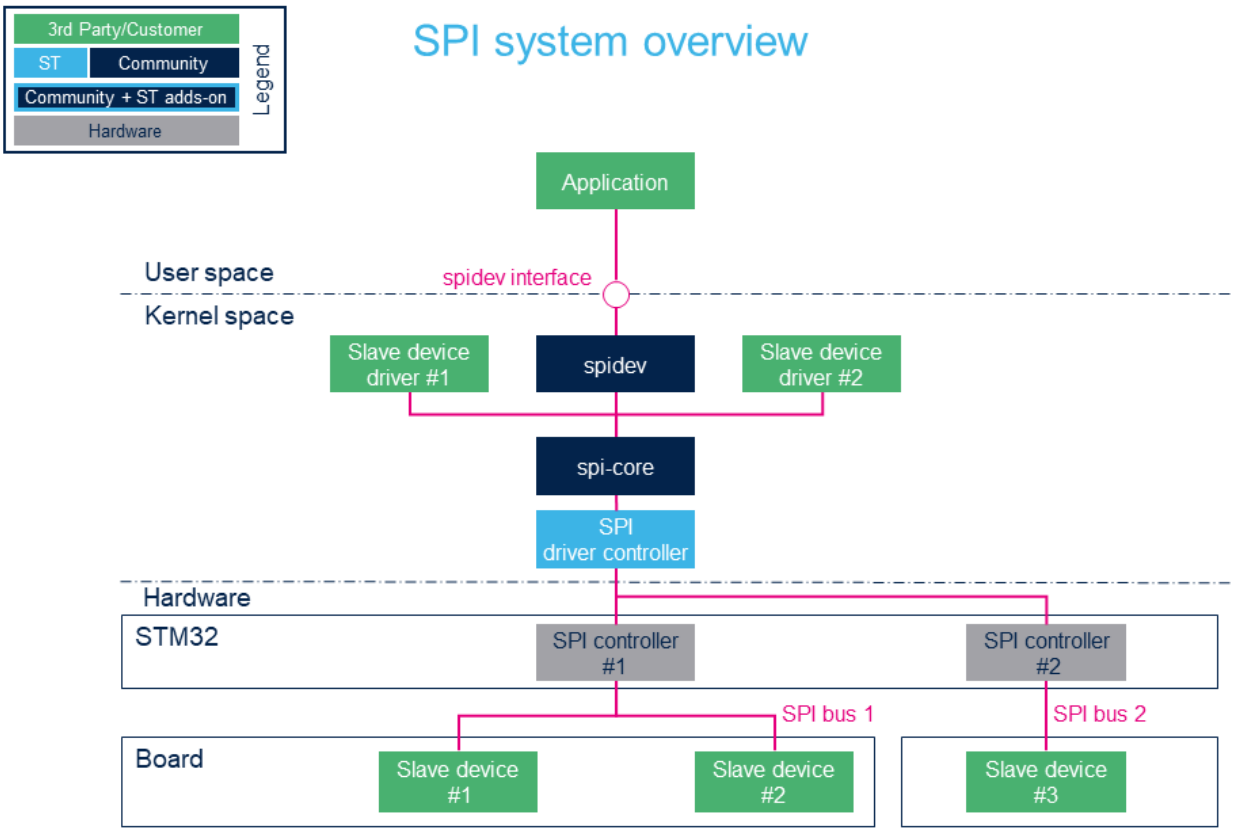
The Linux kernel provides a specific framework for SPI[1] protocol support. The SPI (serial peripheral interface) is a synchronous serial communication interface used for short distance communications, mainly in embedded systems.
This interface was created by Motorola and has become a de facto standard. As it is not defined by a consortium such as I^2C, there are different signal names and signal polarity modes.
SPI devices communicate either in Full duplex, Half duplex, or Simplex (rx/tx) mode using a master-slave architecture with a single master.

The Linux kernel SPI framework provides a complete support for SPI master (the Linux kernel initiates SPI messaging on the bus) and more recently for SPI slave (the Linux kernel answers to requests from the bus master).

See [2] for an introduction on the Linux kernel SPI framework.

2 System overview

The user can add many SPI external devices around the microprocessor device, to create a custom board. Each external device can be accessed through the SPI from the user space or the kernel space.



2.1 Component description

2.1.1 Board external SPI devices

Slave devices 'X' are physical devices (connected to the STM32 microprocessor via an SPI bus) that behave as slaves with respect to the STM32.
The STM32 is the SPI bus master.
A chip select signal allows selecting independently each slave device.

2.1.2 STM32 SPI internal peripheral controller

The STM32 SPI controller handles any external SPI devices connected to the same bus.
The STM32 microprocessor devices usually embed several instances of the SPI internal peripheral allowing to manage multiple SPI buses.
For more information about STM32 SPI internal peripherals, please refer to SPI_internal_peripheral#SPI_main_features

2.1.3 spi-stm32

The STM32 SPI controller driver offers an ST SPI internal peripheral abstraction layer to the spi-core.
It defines all the SPI transfer methods to be used by the SPI core base.

2.1.4 spi-core

spi-core is the "brain of the communication": it instantiates and manages all buses and peripherals.

- As stated by its name, this is the SPI engine. It is also in charge of parsing device tree entries both for adapter and devices.

It implements the standard SPI modes: 0, 1, 2 and 3.

2.1.5 Slave device drivers

This layer represents all the drivers associated to physical peripherals.

2.1.6 spidev

spidev is the interface between the user and the peripheral. This is a kernel driver that offers a unified SPI bus access to the user space application using this dev-interface API. See API Description for examples.

2.1.7 Application

The application can control all peripherals thanks to the *spidev* interface.

2.2 API description

2.2.1 User space application

The user space application uses a kernel driver (spidev) for SPI transfers through the devfs.
Let's take the example of an SPI device connected to bus B with chip select C. The *spidev* driver provides the following interfaces:

- /dev/spidevB.C: character special device created by "udev" that is used by the user space application to control and transfer data to the SPI device.

Supported system calls : open(), close(), read(), write(), ioctl(), llseek(), release().

Constant	Description
SPI_IOC_RD_MODE, SPI_IOC_WR_MODE	Gets/sets SPI transfer mode
SPI_IOC_RD_LSB_FIRST, SPI_IOC_WR_LSB_FIRST	Gets/sets bit justification used to transfer SPI words.
SPI_IOC_RD_BITS_PER_WORD, SPI_IOC_WR_BITS_PER_WORD	Gets/sets the number of bits in each SPI transfer word.
SPI_IOC_RD_MAX_SPEED_HZ, SPI_IOC_WR_MAX_SPEED_HZ	Gets/sets the maximum SPI transfer speed in Hz.

Supported ioctls commands

The table above shows only the main commands. Additional commands are defined in the framework (see **dev-interface API**[3] for a complete list).

2.2.2 Kernel space peripheral driver

The kernel space peripheral driver communicates with SPI devices and uses the following **SPI core API**: [4]

3 Configuration

3.1 Kernel configuration

Enable SPI support (SPI framework and STM32 SPI driver) in the kernel configuration through the Linux Menuconfig tool: Menuconfig or how to configure kernel.

```
[x] Device Drivers
    [x] SPI support
        *** SPI Master Controller Drivers ***
        [x] STMicroelectronics STM32 SPI controller
        *** SPI Protocol Masters ***
        [x] User mode SPI device driver support
```

This can be done manually in your kernel:

```
CONFIG_SPI=y
CONFIG_SPI_MASTER=y
CONFIG_SPI_STM32=y
CONFIG_SPI_SPIDEV=y
```

Drivers (controller and peripheral) can be compiled as a kernel module (selected by the 'm' kernel configuration file) or directly into the kernel (aka built-in) (selected by the 'y' kernel configuration file).

Information

If a slave device is involved in the boot process, the drivers required to support it are considered as critical and must be built into the kernel

3.2 Device tree configuration

Please refer to SPI device tree configuration.

4 How to use the framework

Detailed information on how to write an SPI slave driver to control an SPI device are available in the Linux kernel documentation [5].

User-space examples can be found in How to use SPI from Linux userland with spidev.

5 How to trace and debug the framework

5.1 How to trace

5.1.1 Activating SPI framework debug messages

To get verbose messages from the SPI Framework, activate "Debug support for SPI drivers" in the Linux kernel via menuconfig Menuconfig or how to configure kernel.

```
[x] Device Drivers
    [x] SPI support
        [x] Debug support for SPI drivers
```

This is done manually in your kernel .config file:

```
CONFIG_SPI=y
CONFIG_SPI_DEBUG=y
CONFIG_SPI_MASTER=y
...
```

the debug support for SPI drivers (CONFIG_SPI_DEBUG) compiles all the SPI files located in Linux kernel drivers/spi folder with DEBUG flag.

Information

Reminder: *loglevel* needs to be increased to 8 by using either boot arguments or the *dmesg -n 8* command through the console

5.1.2 Dynamic trace

A detailed dynamic trace is available in How to use the kernel dynamic debug

```
Board $> echo  "file spi* +p" > /sys/kernel/debug/dynamic_debug/control
```

This command enables all the traces related to the SPI core and drivers at runtime.
A finer selection can be made by choosing only the files to trace.

Information

Reminder: *loglevel* needs to be increased to 8 by using either boot arguments or the *dmesg -n 8* command through the console

5.2 How to debug

5.2.1 Detecting SPI configuration

5.2.1.1 sysfs

When a peripheral is instantiated, the spi-core and the kernel export several files through the sysfs :

- **/sys/class/spi_master/spix** shows all the instantiated SPI buses, '**x**' being the SPI bus number.

Warning

'x' may not match the SPI internal peripheral index as it depends on device probing order.

- **/sys/bus/spi/devices** lists all the instantiated peripherals. For example, the repository named **spi0.0** corresponds to the peripheral connected to SPI bus 0 and chip select 0. Below an example representing the "TPM" device:
- **/sys/bus/spi/drivers** lists all the instantiated drivers. The **tpm_tis_spi/** repository is the driver of TPM 2.0. The **spidev/** repository is the generic driver of SPI user mode.

```
/sys/bus/spi/devices/spi0.0/
            /
            /drivers/tpm_tis_spi/spi0.0/
            /drivers/spidev/...
```

```
/sys/class/spi_master/spi0/spi0.0
                      /spi1/
                      /spi2/
```

5.2.2 devfs

If the *spidev* driver is compiled into the kernel, the repository **/dev** contains all SPI device entries. They are numbered spi**x.y** where:

- 'x' is the SPI bus number
- 'y' is the chip select index on the bus.

Unlike i2c-dev which allows full access to the I^2C bus, the *spidev* offers direct access to the SPI device identified by its chip select signal defined in the device tree node.

Below example shows user mode SPI device on SPI bus 4, chipselect 2.

```
/dev/spi4.2
```

For more information, please refer to the spidev documentation [3].

6 Source code location

- The SPI framework is available under drivers/spi
- The STM32 SPI driver is available under drivers/spi/spi-stm32.c
- The user API for the SPI bus is available under include/linux/spi/spi.h and SPI dev is include/uapi/linux/spi/spidev.h .

7 References

- https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface
- https://bootlin.com/doc/training/linux-kernel/
- 3.03.1 Documentation/spi/spidev.rst dev-interface API
- Serial Peripheral Interface (SPI)
- Documentation/spi/spi-summary.rst Linux kernel SPI framework summary

Serial Peripheral Interface

Board support package

Application programming interface

Device File System (See https://en.wikipedia.org/wiki/Device_file#DEVFS for more details)

also known as

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Trusted Platform Module