SPI overview

Contents

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 22.02.2021 - 10:10 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 22.02.2021 - 08:34

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.

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

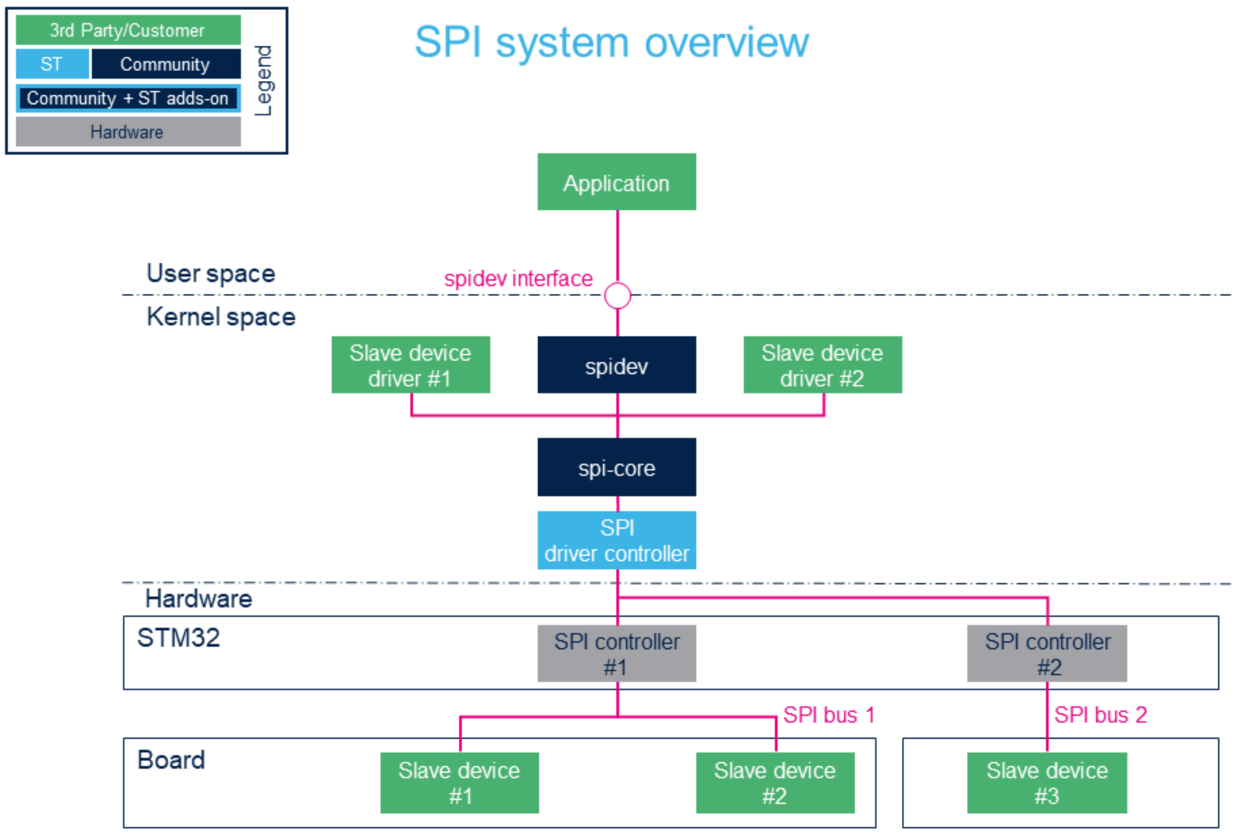
The Linux kernel provides a specific framework for SPI[1] protocol support. The SPI (serial peripheral interface) is a synchronous serial communication interface used for short distance communications, mainly in embedded systems.
This interface was created by Motorola and has become a de facto standard. 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).

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.

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.

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.

'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^{2}C 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

Linux® is a registered trademark of Linus Torvalds.

Serial Peripheral Interface

Board support package

Application programming interface

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

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 26.11.2020 - 13:10 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 26.11.2020 - 11:31

also known as

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

Trusted Platform Module

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 02.11.2020 - 10:48 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 19.10.2020 - 12:09

revreview-invalid

returnto

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 31.03.2021 - 08:47 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 26.03.2021 - 08:44

revreview-invalid

returnto

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 28.01.2021 - 13:45 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 28.01.2021 - 10:36

revreview-invalid

returnto

revreview-invalid

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 04.01.2021 - 10:24 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 21.12.2020 - 10:48

returnto

revreview-invalid

bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-laststable-tag-text: 22.02.2021 - 10:10 / bs-flaggedrevsconnector-addstabledatetochapterheadlinesmodifier-stablerevisiondate-tag-text: 22.02.2021 - 08:34

returnto

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.

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

Contents

1 Framework purpose

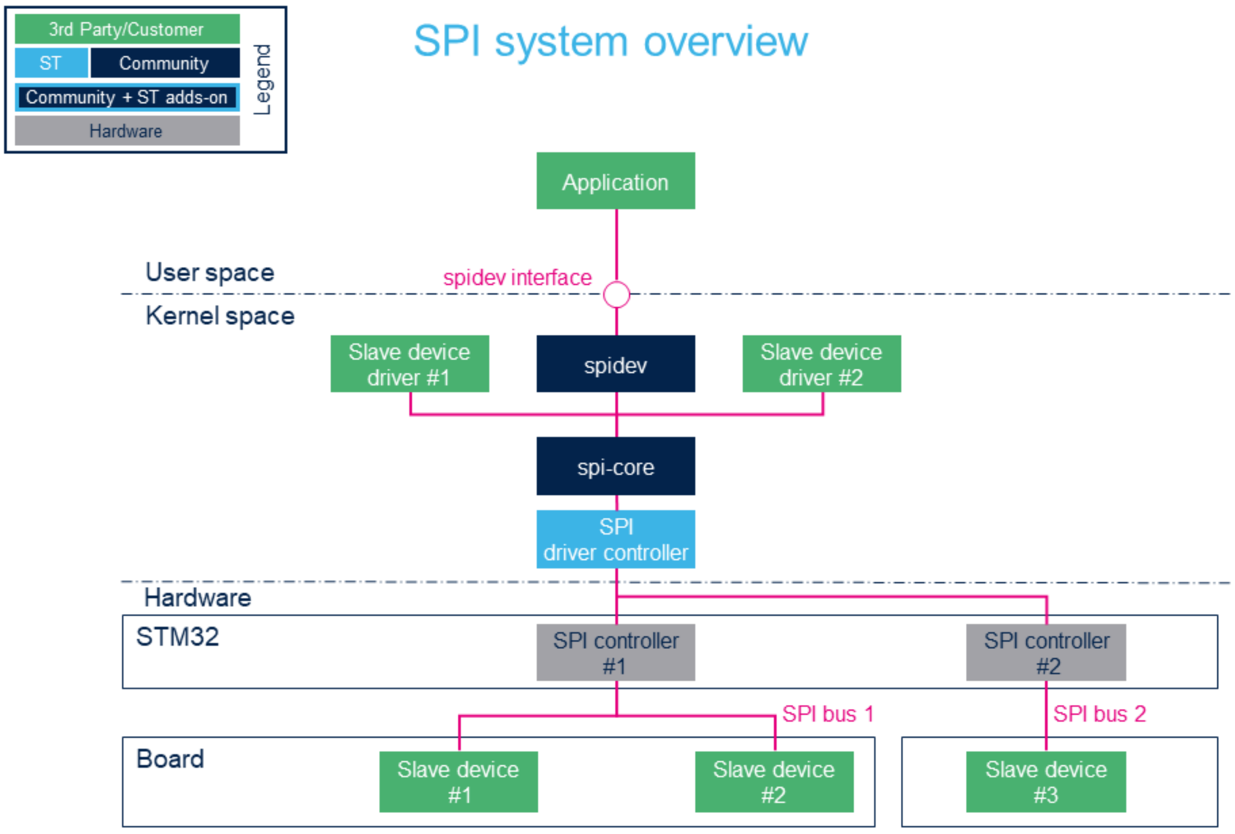
The Linux kernel provides a specific framework for SPI[1] protocol support. The SPI (serial peripheral interface) is a synchronous serial communication interface used for short distance communications, mainly in embedded systems.
This interface was created by Motorola and has become a de facto standard. 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).

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.

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.

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.

'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^{2}C 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

Linux® is a registered trademark of Linus Torvalds.

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