SpiDev Documentation

(*Draft version* 2020-07-12)

1. Table of Content

1. Description

This module defines an object type that allows SPI transactions on hosts running the Linux kernel. The host kernel must have SPI support and SPI device interface support. All of these can be either built-in to the kernel, or loaded from modules.

This document is written for any Raspberry Pi model running recent versions of Raspbian Buster and the current (July 2020) version of Raspberry Pi OS. It could apply to other systems using different Linux distributions with slight changes.

Details can be found at https://sigmdel.ca/michel/ha/rpi/spi_on_pi_en.html.

2. SpiDev Installation

In a Python3 virtual environment, the spidev module can be installed with a simple `pip` command:

```
(spipy) pi@raspberrypi:~ $ pip install spidev
```

The module can be installed in the default Python3 installation with the following:

```
pi@raspberrypi:~ $ sudo apt install python3-spidev
```

From `.../site-packages/spidev-3.4.dist-info/METADATA`

```
Name: spidev
Version: 3.4
Summary: Python bindings for Linux SPI access through spidev
Home-page: http://github.com/doceme/py-spidev
```

3. SPI kernel drivers

By default, the SPI kernel drivers are not loaded in `Raspbian Buster Lite`. That can be done on a one-off basis with the `dtparam` utility:

```
pi@raspberrypi:~ $ sudo dtparam spi=on
```

This will create two spi devices: `/dev/spidev0.0` and `/dev/spidev0.1`.

To add the driver automatically at each boot, edit the `/boot/config.txt` file or use the `raspi-config` utility.

Signal	GPIO pin	Physical pin
SPI0_MOSI	10	19
SPI0_MISO	9	21
SPI0_SCLK	11	23
SPI0_CEO_N	8	24
SPI0_CE1_N	7	26

SPI0 connections

Raspberry Pi models with 40 ping GPIO headers have a second SPI bus which can be enabled with the `dtoverlay` utility:

```
pi@raspberrypi:~ $ sudo dtoverlay spi1-3cs
```

This will create three devices: `/dev/spidev1.0` , `/dev/spidev1.1` and `/dev/spidev1.2`.

Signal	GPIO pin	Physical pin
SPI1_MOSI	20	38
SPI1_MISO	19	35
SPI1_SCLK	21	40
SPI1_CEO_N	18	22
SPI1_CE1_N	17	11
SPI1_CE2_N	16	36

SPI1 connections

Other overlays are available with fewer chip select signals (`spi1-2cs` and `spi1-cs`).

The Raspberry Pi model 4 and the Compute Module has even more SPI buses.

4. Permission

Because the SPI device interface is used to read and write, users of a SPI device node must have root permissions. However, in Raspbian Buster and Raspberry Pi OS members of the `spi` group have access to the interface and the default user is a member of that group. Hence it will not be necessary to use the `sudo` prefix when running a Python script that imports `spidev`.

5. Examples

Simple output

This example will open SPI and writes a byte (0x3A) every tenth of a second until Ctrl+C is pressed.

```
import spidev
import time

spi = spidev.SpiDev(0, 1)     # create spi object connecting to /dev/spidev0.1
spi.max_speed_hz = 250000     # set speed to 250 Khz

try:
   while True:                # endless loop, press Ctrl+C to exit
```

```
      spi.writebytes([0x3A])  # write one byte
      time.sleep(0.1)         # sleep for 0.1 seconds

finally:
   spi.close()                # always close the port before exit
```

Reverse bits

This script will reverse the bit ordering in one byte (if you are not able to change LSB / MSB first to your needs.

```
def ReverseBits(byte):
   byte = ((byte & 0xF0) >> 4) | ((byte & 0x0F) << 4)
   byte = ((byte & 0xCC) >> 2) | ((byte & 0x33) << 2)
   byte = ((byte & 0xAA) >> 1) | ((byte & 0x55) << 1)
   return byte
```

Print bytes

This script will print out a byte array in a human readable format (hexadecimal). This is often useful during debugging.

```
def BytesToHex(Bytes):
   return ''.join(["0x%02X " % x for x in Bytes]).strip()
```

6. Class Constructors

spidev()

Syntax: `myspi = spidev()`

Remark: This is the base constructor which does not connect the object to a system SPI device. It will be necessary to use the open() method to associate the object with a device.

spidev(bus, cs)

Syntax: `myspi = spidev(bus, cs)`

Remark: this is an overloaded constructor that creates the object and connects it to a system SPI device. So

```
myspi = spidev(bus, cs);
```

and

```
myspi = spidev();
myspi.open(bus, cs);
```

are equivalent.

On the Raspberry Pi, the `max_speed_hz` attribute should be set to a reasonable value (less than or equal to 32 MHz) after the spidev object is

connected to a system SPI device either with the `spidev(bus, cs)` constructor or the `open(bus, cs)` method.

7. Class Attributes

bits_per_word

Description: Number of bits per word.

Default: 8

Range: 8 .. 16

Restrictions: Read-only on the Raspberry Pi.

cshigh

Description: If true the chip select signal is active high otherwise it is active low.

Default: False (chip select is active low).

loop

Description: If true the “loop back configuration” is enabled.

Default: False

Remarks: Read-only on the Raspberry Pi. This is used for test purposes; anything that gets received will be echoed back (presumably that means anything received on MISO line is echoed to the MOSI line. A loop back on a single SPI port can be done by connecting its MISO and MOSI lines together.

lsbfirst

Description: Boolean property that specifies if a word is transmitted with the least significant bit first or not (which, of course, means that it would be transmitted most significant bit first).

Default: False (most significant bit first)

Restrictions: Read-only in Raspbian Buster. The default value seems to depend on the bit endianness of the system. The Raspberry Pi can only send most significant bit first, so it may be necessary to convert the byte(s) manually (see code examples for such a script) when a slave device expects to receive bytes least significant bit first and sends its own data in the same way.

max_speed_hz

Description: Property that specifies the maximum bus speed in Hz.

Default: 125000000

Remarks: **The 125 MHz default value is not sustainable on a Raspberry Pi and it must be changed to a reasonable value**. The maximum speed appears to be about 32 MHz on the Raspberry Pi.

There is a debate about permissible speed values, with some insisting that the speed must be a power of 2, while others argue that it can be a multiple of 2. Tests at least partially confirm that the latter is correct. It was possible to set the speed at 3800 Hz, which appears to be a lower limit, and at 4800 Hz. Neither of these values is a power of 2.

mode

Description: the SPI mode as a two-bit pattern of Clock Polarity and Clock Phase.

Default: 0

Range: 0 to 3

Mode	clock polarity (CPOL)	clock phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

threewire

Description: If true, the SPI device is set in three wire mode which means the MISO and MOSI signals share a single date line. In effect, the protocol becomes half duplex.

Default: False

Remark: Not tested!

8. Class Methods

close

Syntax: `close()`

Returns: None

Description: Disconnects the object from the system SPI device.

open

Syntax: `open(bus, cs)`

Description: Connects the object to the specified system PI device.

Example: `open(1,2)` will open /dev/spidev1.2

Remark: Of course, the kernel driver for the device must be loaded.

readbytes

Syntax: `readbytes(len)`

Returns: [values]

Description: Read a list of `len` values (bytes) from SPI device.

writebytes

Syntax: `writebytes([values])`

Returns: None

Description: Write a list of bytes to the SPI device.

Remark: Fails if the list of values is bigger than the maximum buffer size. In Raspbian Buster this limit is 4096 bytes. For longer lists use `writebytes2`.

This is not a SPI transaction, all data read from the MISO signal is discarded and the list of values is not modified.

writebytes2

Syntax: `writebytes2([values])`

Returns: None

Description: Write a list of bytes of any size to the SPI device.

Remarks: Unlike `writebytes`, which fails if the list of values is greater than the maximum buffer size, `writebytes2` handles long lists by breaking it into chunks sent out one at a time. The size of each chunk is equal to the maximum buffer size except perhaps for the last one which could be smaller. The chip select signal is released between each chunk.

Also, `writebytes2` understands the buffer protocol so it can accept numpy byte arrays for example without need to convert them with `tolist()` first. This offers much better performance when transferring frames to SPI displays for instance.

This is not a SPI transaction, all data read from the MISO signal is discarded and the list of values is not modified.

xfer

Syntax: `rcvd = xfer([values])`

`rcvd = xfer([values], speed)`

`rcvd = xfer([values], speed, delay)`

`rcvd = xfer([values], speed, delay, bits)`

Returns: [values]

Description: Perform a SPI transaction: a list of bytes is written to the SPI device and as each byte in that list is sent out, it is replaced by the data simultaneously read from the SPI slave device over the MISO line.

Remarks: Fails if the list of values is bigger than the maximum buffer size which is 4,096 bytes. To send longer lists of values use `xfer3`.

While the documentation says that the “chip select signal will be released and reactivated between blocks”, this does not make much sense as this function sends only one block of data with at most 4096 bytes of data in Raspbian Buster.

If `speed` is not specified, the SPI clock is set to the `maximum_speed_hz` attribute. If the speed is specified (in Hz) it will be used for that single transaction and will not modify the value of `maximum_speed_hz` attribute of the object.

If `delay` is not specified, assume the value is 0. The chip select line remains asserted for a short time after all the data has been transmitted, however, if a delay is specified, the chip select will remain asserted for that additional time (in microseconds).

The `bits` (per word) parameter can have only one value, 8, in the Raspberry Pi.

xfer2

Syntax: `rcvd = xfer2([values])`

`rcvd = xfer2([values], speed)`

`rcvd = xfer2([values], speed, delay)`

`rcvd = xfer2([values], speed, delay, bit`

Remark: The documentation says “Contrary to `xfer`, the chip select signal will be held active between blocks” but this does not seem to be borne out. There does not seem to be any difference between `xfer` and `xfer2`.

The speed, delay and bit options have the same effect as in the `xfer` function as explained above.

xfer3

Syntax: `rcvd = xfer3([values])`

`rcvd = xfer3([values], speed)`

`rcvd = xfer3([values], speed, delay)`

`rcvd = xfer3([values], speed, delay, bit`

Returns: [values]

Description: Perform a SPI transaction: a list of bytes of arbitrary size is written to the SPI device and as each byte in that list is sent out, it is replaced by the data simultaneously read from the SPI slave device over the MISO line.

Remark: Unlike `xfer` and `xfer2`, which fail if the list of values is greater than the maximum buffer size, `xfer3` handles a long list by breaking it into chunks sent out one at a time. The size of each chunk is equal to the maximum buffer size except perhaps for the last one which could be smaller.

The chip select signal is released between each chunk for approximately a fifth of a milliseconds and indefinitely after the last chunk of data is sent. If a `delay` is specified, the chip select line will remain asserted for that period of time at the end of the transmission of each block of data before being released.

9. Maximum SPI Buffer Size

The data to be sent out by a SPI device using any of the `writebytes` and `xfer` methods is buffered. The size of the latter is seemingly specified in the `/sys/module/spidev/parameters/bufsiz`.

```
pi@rasberrypi:~ $ cat /sys/module/spidev/parameters/bufsiz
4096
```

That value can be changed by adding a `spidev.bufsiz=xxxx` option (where `xxxx` is the needed buffer size) in the `/boot/cmdline.txt` file. Remember to keep the content of that file on a single line. The system has to be rebooted for this change to take effect.

No matter what value is assigned to `bufsiz`, the `writebytes`, `xfer` and `xfer2` functions use a fixed buffer size of 4096 bytes and will not transfer any more data. The `writebytes2` and `xfer3` will use the specified buffer size when sending bigger lists.