SpiDev Documentation

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

Because the SPI device interface is opened R/W, users of this module usually must have root permissions.

This tutorial is written for the use with a Raspberry Pi (Raspbian Wheezy distribution), but it should match for several others, too.

SPI installation

First things first, it's always the best to keep your Raspberry Pi up to date, otherwise some things here may not work. Open a new console and execute the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

If you don't have done this yet, grab some coffee or something else because this may take up to an hour or two.

If you don't have installed it already you need to install the *Python Dev package*:

```
sudo apt-get install python-dev
```

You need to be sure that SPI is enabled in your system (disabled by default because rarely used). Open a new terminal and enter:

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

This will open a text editor. You should see a line like this in this file:

```
spi-bcm2708
```

Put a # before this line to comment it out. Save (Ctrl+O) and exit (Ctrl+X). After a reboot the device should be found.

```
sudo reboot
# --- after reboot ---
lsmod
```

In the list *lsmod* outputs you should find the entry "*spi_bcm2708*" now.

Also you need the *SpiDev Python module*. Open a new terminal and execute the following commands:

```
mkdir python-spi
cd python-spi
wget https://raw.github.com/doceme/py-spidev/master/setup.py
wget https://raw.github.com/doceme/py-spidev/master/spidev_module.c
sudo python setup.py install
```

Now the SPI device should be available to Python.

How to run python scripts

Of course there are several ways. You can either simple save your script and run it using:

```
sudo python myscript.py
```

Or you can mark the file as executable. For this you need to add one line at the top of your script:

```
#!/usr/bin/python
```

Now mark it as executable:

```
sudo chmod +x myscript.py
```

Now you can execute it like any other application:

```
sudo ./myscript.py
```

Examples

Simple output

This example will open SPI and writes a byte (0xAA) to it each 0.1 second until you cancel it with Ctrl+C.

```
import spidev
import time

spi = spidev.SpiDev()                # create spi object
spi.open(0, 1)                       # open spi port 0, device (CS) 1

try:
    while True:
        resp = spi.xfer2([0xAA])     # transfer one byte
        time.sleep(0.1)              # sleep for 0.1 seconds
    #end while
except KeyboardInterrupt:            # Ctrl+C pressed, so…
    spi.close()                      # … close the port before exit
#end try
```

Helpful scripts

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
#end def
```

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()
#end def
```

Member

bits_per_word

Description: Property that gets / sets the bits per word.

Range: 8 .. 16

close

Syntax: `close()`

Returns: `None`

Description: Disconnects the object from the interface.

cshigh

Description: Property that gets / sets if the CS is active high.

loop

Description: Property that gets / sets “loopback configuration”. This is used to test the pcb for example. Anything that gets received will be echoed back.

lsbfirst

Description: Property that gets / sets if LSB should be transferred first or last.

Remarks: Needs Boolean value. However, this property seems to be read only and the value dependences on the endianess of the controller / cpu. The Raspberry Pi can only send MSB first, so you may need to convert the byte(s) manually (see code examples for such a script).

max_speed_hz

Description: Property that gets / sets the maximum bus speed in Hz.

mode

Description: Property that gets / sets the SPI mode as two bit pattern of Clock Polarity and Phase [CPOL|CPHA]

Range: 0b00 = 0 .. 0b11 = 3

open

Syntax: `open(bus, device)`

Description: Connects the object to the specified SPI device.

open(X,Y) will open /dev/spidev-X.Y

readbytes

Syntax: `read(len)`

Returns: `[values]`

Description: Read *len* bytes from SPI device.

threewire

Description: Property that gets / sets “SI/SO signals shared” so you have only 1 data line. Read the data sheets for more details.

writebytes

Syntax: `write([values])`

Returns: `None`

Description: Write bytes to SPI device.

xfer

Syntax: `xfer([values])`

Returns: `[values]`

Description: Perform SPI transaction.

CS will be released and reactivated between blocks. *delay* specifies delay in µsec between blocks.

xfer2

Syntax: `xfer2([values])`

Returns: `[values]`

Description: Perform SPI transaction. CS will be held active between blocks.