SFP Transceivers Reprogramming V1.0

Wei

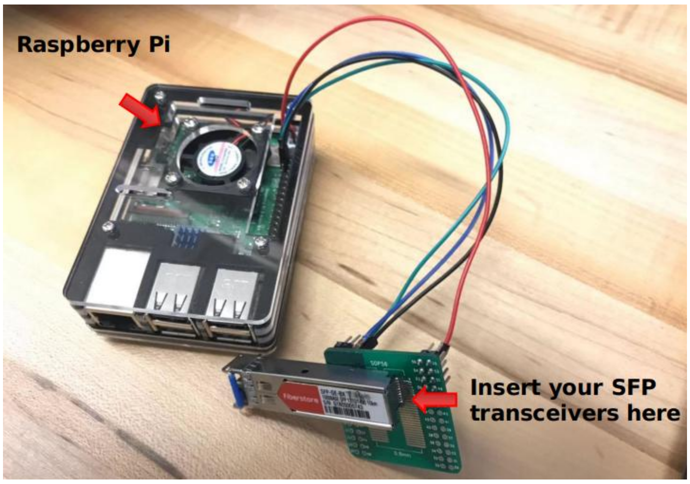
01-06-2020

This document is about how to reprogram a sfp transceiver.
You need some hardware and software support.

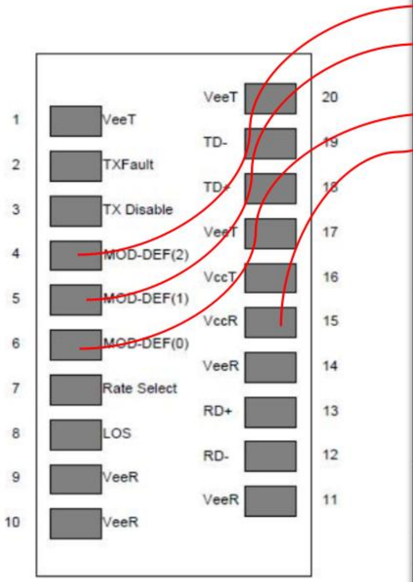
1. Hardware

We use a Raspberry Pi for sfp transceivers reprogramming, so that's the key hardware you need. At the same time, We also need a sfp connector, which is another hardware you need. Here is a picture of the hardware.

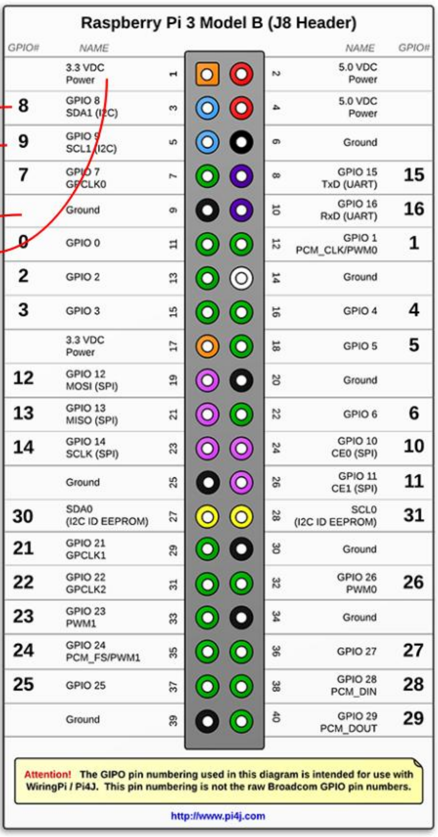
At last, don't forget to prepare for a monitor(with HDMI lane), a keyboard and a mouser, because Raspberry Pi is like a tiny computer, on which you can run some python code.



The connection between the raspberry pi and sfp connector is as follow.



Pin out of SFP transceivers



Raspberry Pi 3 Model B (J8 Header)

GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1	2	5.0 VDC Power	
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power	
9	GPIO 9 SCL1 (I2C)	5	6	Ground	
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (UART)	15
	Ground	9	10	GPIO 16 RxD (UART)	16
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0	1
2	GPIO 2	13	14	Ground	
3	GPIO 3	15	16	GPIO 4	4
	3.3 VDC Power	17	18	GPIO 5	5
12	GPIO 12 MOSI (SPI)	19	20	Ground	
13	GPIO 13 MISO (SPI)	21	22	GPIO 6	6
14	GPIO 14 SCLK (SPI)	23	24	GPIO 10 CE0 (SPI)	10
	Ground	25	26	GPIO 11 CE1 (SPI)	11
30	SDA0 (I2C ID EEPROM)	27	28	SCL0 (I2C ID EEPROM)	31
21	GPIO 21 GPCLK1	29	30	Ground	
22	GPIO 22 GPCLK2	31	32	GPIO 26 PWM0	26
23	GPIO 23 PWM1	33	34	Ground	
24	GPIO 24 PCM_FS/PWM1	35	36	GPIO 27	27
25	GPIO 25	37	38	GPIO 28 PCM_DIN	28
	Ground	39	40	GPIO 29 PCM_DOUT	29

Attention! The GIPO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

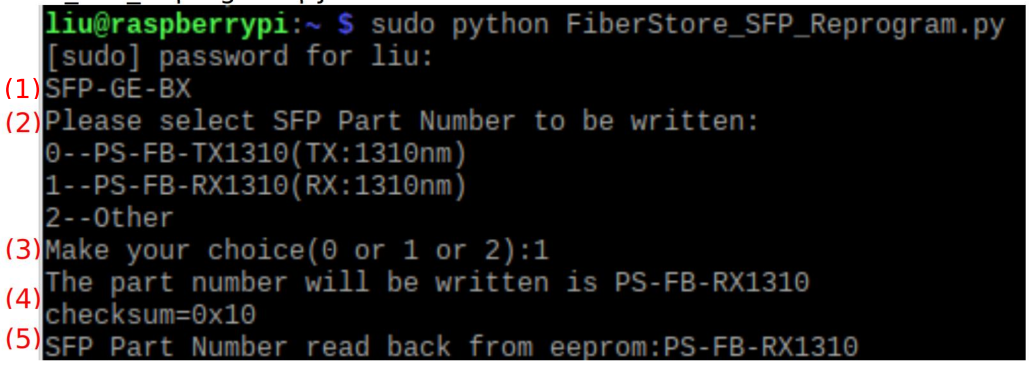
http://www.pi4j.com

Pin out of Raspberry Pi

2. Software

Some python code are designed for reprogramming sfp transceivers, and they are FiberStore_SFP_Reprogram.py and SFP_WR_V1_1.py.

You need to copy these two py files to your raspberry pi, and then run FiberStore_SFP_Reprogram.py as root.



```
liu@raspberrypi:~ $ sudo python FiberStore_SFP_Reprogram.py
[sudo] password for liu:
SFP-GE-BX
Please select SFP Part Number to be written:
0--PS-FB-TX1310(TX:1310nm)
1--PS-FB-RX1310(RX:1310nm)
2--Other
Make your choice(0 or 1 or 2):1
The part number will be written is PS-FB-RX1310
checksum=0x10
SFP Part Number read back from eeprom:PS-FB-RX1310
```

When you run the python script, there are some information shown:

(1) The original PN of the sfp transceiver;

(2) The PN you want to write to your sfp transceiver:

There are two default PN: PS-FB-TX1310 and PS-FB-RX1310, or you want write what you want[the length should not be more than 16 bytes].

(3) Make your choice. (We write “PS-FB-RX1310” to the sfp transceiver, so we select ‘1’)

(4) Raspberry Pi is writing PN to sfp transceiver, and calculate the checksum, and write it to sfp transceiver at the same time.

(5) Raspberry Pi reads back the PN from sfp transceiver, so that you can confirm it.

The time of writing PN is about 5s.

3. Part Number

As dan suggested, the PN should show who made the transceiver, and that vendor's part number, and the PN length should not more than 16 bytes, so the PN is designed like this:

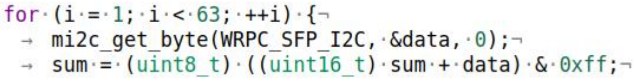
PS-FB-TX1310

Which means **P**ano**S**eti-**F**iber**S**tore-**TX** wavelength is **1310**nm.

4. Checksum

During the test, I found that WR-LEN doesn’t care about the checksum, but WR working on quabo checks the checksum, so it’s better to write the correct checksum to sfp transceviers, when you reprogram it.

The checksum the 64th byte in the sfp transceiver flash, and here is the code about how to calculate it.



```
for (i = 1; i < 63; ++i) {
	mi2c_get_byte(WRPC_SFP_I2C, &data, 0);
	sum = (uint8_t) ((uint16_t) sum + data) & 0xff;
```

It just sums the first 63 bytes, and the lowest byte is the checksum value.

It’s easy, but important, or WR on quabo can’t detect your reprogrammed sfp transceivers.