RALIM TEK

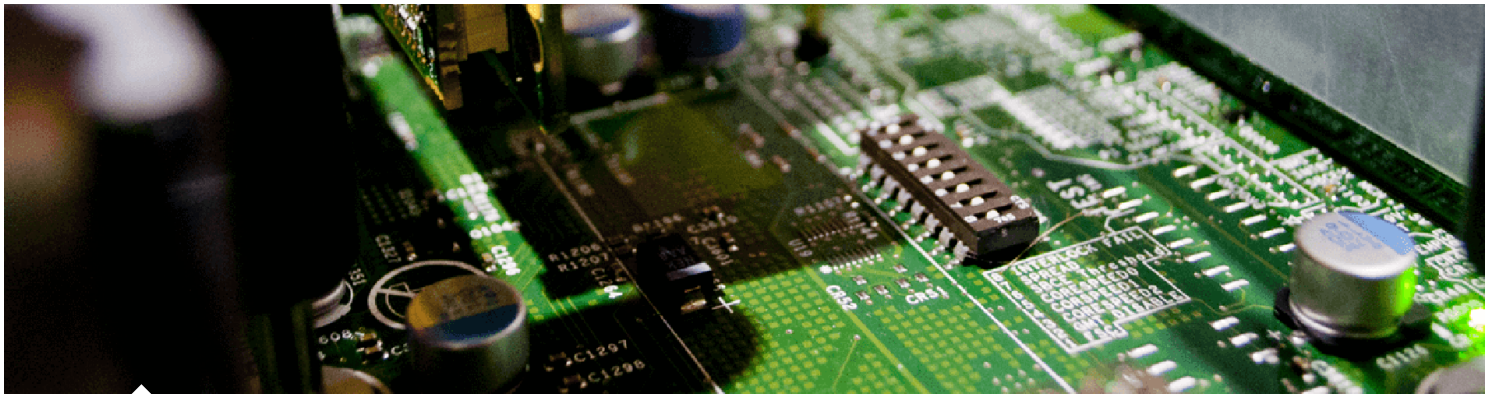

Photo Credit: Ben Brown

Adding a secondary sd card on Raspberry PI

CONTENTS

WARNING

If you are planning to use the SPI method on a raspberry pi 1, compute module 1 or a pi zero, please read the extra instructions at the end of the page. These devices require a driver patch & kernel rebuild. (Its not hard, just slow).

Secondary SD

For one of my upcoming projects featuring a raspberry pi zero, extra internal storage is required and I wanted to keep the USB port free for connecting as a OTG connection.

There are two often talked about methods for mounting a second SD card on the Raspberry PI, these are using the SDIO controller and the SPI interface. I can confirm on kernal 4.4.y both of these methods are usable and can be enabled using the device trees.

SDIO (Faster but only 1 SD card)

Recently on Hack-A-Day, they featured this project RPi WiFi over SDIO that covered excellent details on enabling the second SDIO controller in the raspberry pi. Naturally this controller leads itself to the much simpler task of loading a second SD card to the system as well.

Does it work? - Yes!

Yep it totally works! Currently I am wired up in '1 bit' mode as I only have spi breakout boards handy, but this already provides a write speed of 4MB/sec which is fantastic as a secondary storage device, and plenty fast for most operations.

Wiring

Name	SD Card	Raspberry Pi
VCC	4	17
GND	6	20
CLK/SCLK	5	15
CMD/MOSI	2	16

Name	SD Card	Raspberry Pi
DAT0/MISO	7	18
DAT1	8	22
DAT2	9	37
DAT3/CS	1	13

Yes it is that simple, I would reccomend putting series termination resistors in your connections if you can (~33 ohms).

Software

To enable the system's secondary sdio unit onto the raspberry pi pins you need to add the following to your `/boot/config.txt` file :

1 bit mode (good for SPI sd breakouts)

```
dtoverlay=sdio,poll_once=off,bus_width=1
```

4 bit mode (For everything else)

```
dtoverlay=sdio,poll_once=off
```

Why the poll_once=off ?

Normally linux will only do one set of polling on the bus when the system first boots, but by turning this off the system will keep searching so you can hotswap SD cards (This is also needed for the esp8266 project for different reasons).

SPI

Using the spi controller is slower but may be desireable (for instance if SDIO is in use by a esp8266). If you are on a Pi 1, Compute module 1, or Pi Zero, perform the compiling at the end of this page first, then come back here.

Wiring

Name	SD Card	Raspberry Pi
VCC	4	17
GND	6	20
CLK/SCLK	5	23
CMD/MOSI	2	19
DAT0/MISO	7	21

Name	SD Card	Raspberry Pi
DAT1	8	
DAT2	9	
DAT3/CS	1	24-0/26-1

Two CS lines, one per SD card (all other pins common).

Software

Using the helpful comments at The forum and Device Tree's I was able to construct a working device tree file for mounting the SD card over spi.

First, make sure you are running on an up to date system with the device tree compiler (should be on the newer images automatically I believe)

```
sudo apt update
sudo apt dist-upgrade -y
sudo apt install device-tree-compiler
```

Next, create the file `mmc_spi.dts` and open it for editing. I used nano for this so `nano mmc_spi.dts` works well. Save the file anywhere you wish, I used my home directory.

Insert the following into the file

1 SD card using CS 0

```
/dts-v1/;
/plugin/;

/ {
   compatible = "brcm,bcm2835", "brcm,bcm2836", "brcm,bcm2708", "brcm,bcm2709";

   fragment@0 {
      target = <&spi0>;
      frag0: __overlay__ {
         status = "okay";
         sd1 {
                reg = <0>;
                status = "okay";
                compatible = "spi,mmc_spi";
                voltage-ranges = <3000 3500>;
                spi-max-frequency = <8000000>;
```

```
            };
        };
    };
};
```

Two sd cards using CS 0 & CS 1

```
/dts-v1/;
/plugin/;

/ {
   compatible = "brcm,bcm2835", "brcm,bcm2836", "brcm,bcm2708", "brcm,bcm2709";

   fragment@0 {
      target = <&spi0>;
      frag0: __overlay__ {
         status = "okay";
         sd1 {
                reg = <0>;
                status = "okay";
                compatible = "spi,mmc_spi";
                voltage-ranges = <3000 3500>;
                spi-max-frequency = <8000000>;
         };
         sd2 {
                reg = <1>;
                status = "okay";
                compatible = "spi,mmc_spi";
                voltage-ranges = <3000 3500>;
                spi-max-frequency = <8000000>;
         };

      };
   };
```

```
};
```

Next, compile the created device tree source file with :

```
dtc -@ -I dts -O dtb -o mmc_spi.dtbo mmc_spi.dts
```

This should create the file `mmc_spi.dtbo` which is a compiled device tree blob file. The compiler will generate some warnings, but as of current I do not know the best way to solve these, and the file does work. (Kernel 4.9).

For the system to be able to load this blob at boot, we need to copy it to the folder where the dtbo files are kept.

```
sudo cp mmc_spi.dtbo /boot/overlays/
```

Next we need to enable the overlay to be loaded automatically by the operating system. We can force the system to load this overlay on boot by editing the config.txt file

```
sudo nano /boot/config.txt
```

We want to add `mmc_spi` to the end of the file. If this command is not already in your file anywhere you can just add the line `dtoverlay=mmc_spi` at the end.

After this you should be able to reboot and the system will start looking for sd cards on the spi port for you. Note: If you are using both chip select lines for SD cards, you do NOT want to add `dtparam=spi=on` to the file, as the spi port is already being used for the sd cards.

Creating a filesystem and mounting the SD card

After you have your sd card showing up to your system (check with `ls /dev/mmc*` and look for mmcblk1 etc) we need to format the sd card if it is not formatted already

For this I refer you here , the Ubuntu guide for a new hard drive. The sd cards will show up as extra /dev/mmcblk* entries. Make sure you dont try and wipe your boot sd (usually mmcblk0).

Getting SPI working on a bcm2832 (Pi 1 / compute module 1 / pi zero)

Due to a bug in the SPI DMA driver for the bcm2832, modification of the `spi_mmc.c` driver is required, and a subsequent re-build of the modules (and matching kernal). The modification is simple, however compiling on the pi is reallllllly slow (run it over night) or cross compile. These instructions are lifted from the Raspberry Pi Kernel Building I used cross compiling as I have an Ubuntu server handy, but compiling on device should result in the same details.

Setup on compiling computer

First, download the source code for the pi kernel / firmware.

```
git clone --depth=1 https://github.com/raspberrypi/linux
```

Next, install the needed `bc` command

```
sudo apt install bc
```

Edit the driver to apply the patch

Open the file `linux/drivers/mmc/host/mmc_spi.c` in your favourite editor (I just use plain old nano). In the current file, the line to edit is on line 1394. However it is easy to find for searching for `fail_nobuf1` if the file changes.

You want to change the line :

`if (spi->master->dev.parent->dma_mask) {`

To :

`if(0){`

And then save and close the file. This disables some of the DMA settings and lets the sd card work over spi on the older bcm chipset.

If compiling on the pi

```
cd linux
KERNEL=kernel
make bcmrpi_defconfig
make -j2 zImage modules dtbs
sudo make modules_install
sudo cp arch/arm/boot/dts/*.dtb /boot/
sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/overlays/
sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
sudo scripts/mkknlimg arch/arm/boot/zImage /boot/$KERNEL.img
```

If cross compiling

First you need to install the toolchain.

Setting up the toolchain

`git clone https://github.com/raspberrypi/tools`

I left this in my home folder for this process (ie its path works out to be /home/username/tools/arm-bc2708/...)

You need to edit your `.bashrc` to add the folder to your path so that bash can find the compilers later on.

On x64 bit systems you want to add the following to `$PATH`

`/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin`

otherwise, on x86 (32 bit) you need to add this :

`/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin`

This can be done by opening .bashrc (its in your home folder) in nano. Scroll to the bottom of the file and add the following line

```
PATH=$PATH:/home/userNameHere/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin
```

(or the other path for x86).

Compiling

```
cd linux
KERNEL=kernel
make -j 5 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig
make -j 5 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

Installing onto SD card

Having built the kernel, you need to copy it onto your Raspberry Pi and install the modules; this is best done directly using an SD card reader.

First, use lsblk before and after plugging in your SD card to identify it. You should end up with something like this:

```
sdb
   sdb1
   sdb2
```

with sdb1 being the FAT (boot) partition, and sdb2 being the ext4 filesystem (root) partition.

If it's a NOOBS card, you should see something like this:

```
sdb
  sdb1
  sdb2
  sdb5
  sdb6
  sdb7
```

with sdb6 being the FAT (boot) partition, and sdb7 being the ext4 filesystem (root) partition.

Mount these first, adjusting the partition numbers for NOOBS cards:

```
mkdir mnt/fat32
mkdir mnt/ext4
```

```
sudo mount /dev/sdb1 mnt/fat32
sudo mount /dev/sdb2 mnt/ext4
```

Next, install the modules:

```
sudo make -j 5 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=mnt/ext4 modul
```

Finally, copy the kernel and Device Tree blobs onto the SD card, making sure to back up your old kernel:

```
sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
sudo scripts/mkknlimg arch/arm/boot/zImage mnt/fat32/$KERNEL.img
sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
sudo umount mnt/fat32
sudo umount mnt/ext4
```

Another option is to copy the kernel into the same place, but with a different filename - for instance, kernel-myconfig.img - rather than overwriting the kernel.img file. You can then edit the config.txt file to select the kernel that the Pi will boot into:

```
kernel=kernel-myconfig.img
```

This has the advantage of keeping your kernel separate from the kernel image managed by the system and any automatic update tools, and allowing you to easily revert to a stock kernel in the event that your kernel cannot boot.

Now plug the card back into your Pi and continue with the normal SPI setup, further up the page.

Adding a secondary sd card on Raspberry PI was published on December 10, 2016 and last modified on December 10, 2016.