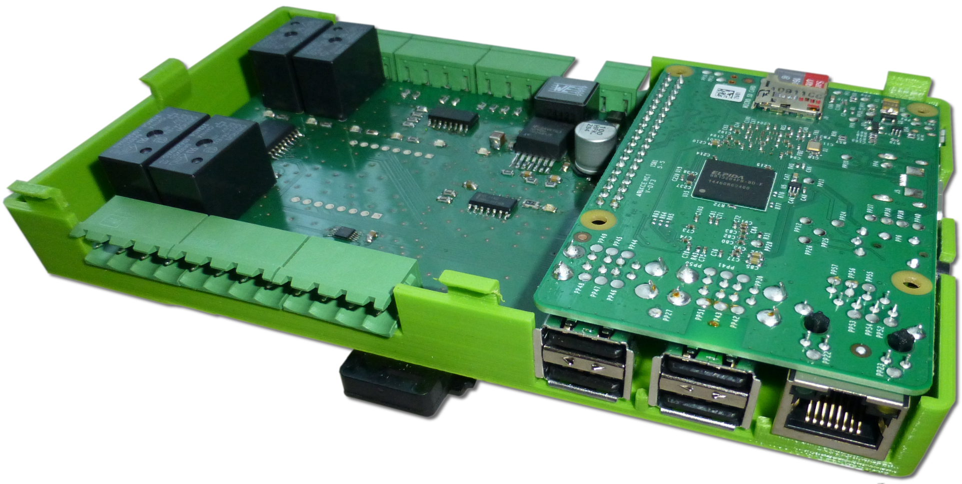
Developer Guide for CONMELEON C1 Hardware

by Herwig Eichler, June 2015



Terms and Conditions

Table of Contents

Illustration Index

1 The CONMELEON Hardware

The CONMELEON IO boards (e.g. the CONMELEON C1) are extensions for the Raspberry Pi B+ or 2B models with 40 pin connection headers. They provide digital inputs and outputs as well as analog ones. Depending on the IO board type, the number of inputs and outputs is different. This guide is explicitly dealing with the CONMELEON C1 hardware.

The CONMELEON C1 board has the following connectors.

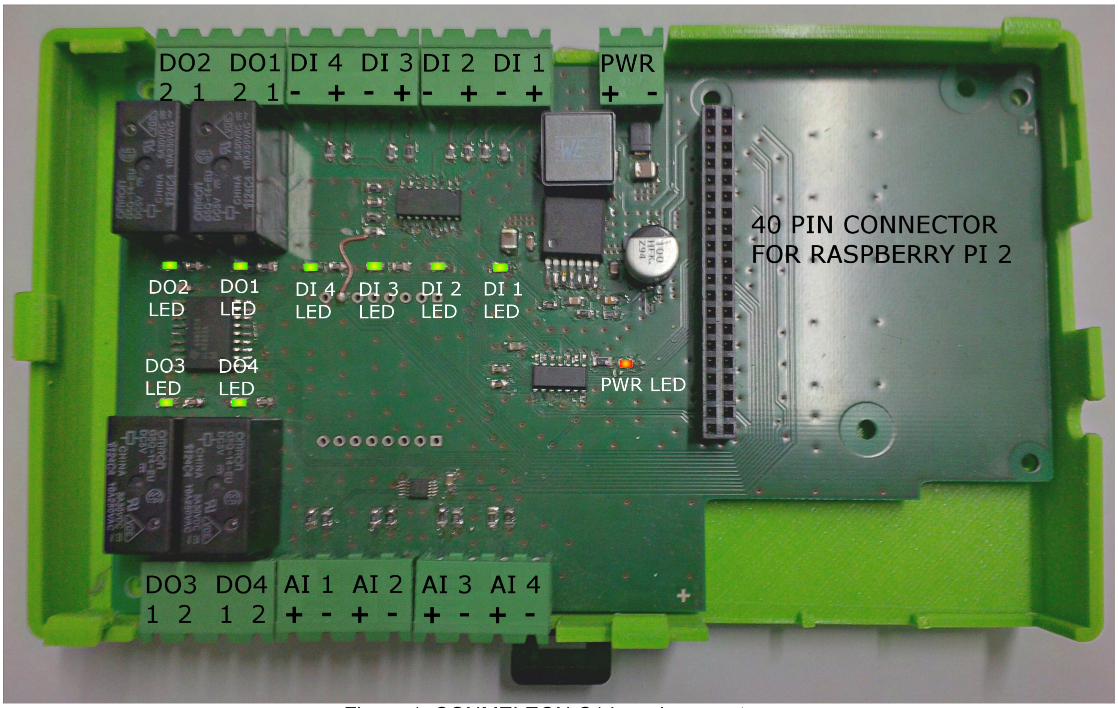


Figure 1: CONMELEON C1 board connectors

Name	Description
PWR	Power supply connector, apply 24V DC, triple check the correct polarity
DI 1	Digital input 1, 24V HTL logic level
DI 2	Digital input 2, 24V HTL logic level
DI 3	Digital input 3, 24V HTL logic level
DI 4	Digital input 4, 24V HTL logic level
DO 1	Digital output 1, relay normally open
DO 2	Digital output 2, relay normally open

DO 3	Digital output 3, relay normally open
DO 4	Digital output 4, relay normally open
AI 1	Analog input 1, 0..10V DC
AI 2	Analog input 2, 0..10V DC
AI 3	Analog input 3, 0..10V DC
AI 4	Analog input 4, 0..10V DC
Raspberry Pi 2 B connector	Attach the Raspberry Pi 2 B to this 40 pin header. The microSD card should face upwards (in direction of PWR) and the ethernet connector should face downwards
PWR LED	This LED glows in red color, if the supply voltage is applied to the PWR connector
DI 1 LED	This LED glows in green color, if the voltage at DI 1 is applied
DI 2 LED	This LED glows in green color, if the voltage at DI 2 is applied
DI 3 LED	This LED glows in green color, if the voltage at DI 3 is applied
DI 4 LED	This LED glows in green color, if the voltage at DI 4 is applied
DO1 LED	This LED glows in green color, if the relay at DO 1 is switched on
DO2 LED	This LED glows in green color, if the relay at DO 2 is switched on
DO3 LED	This LED glows in green color, if the relay at DO 3 is switched on
DO4 LED	This LED glows in green color, if the relay at DO 4 is switched on

The following figure shows the pinout of the Raspberry Pi B+ and 2B (40 pin header) for completeness. Pleas note that the figure shows the pins of the Raspberry Pi, they will be mirrored on the CONMELEON C1 board (Pin 1 3,3V will be on the right and Pin 2 5V will be on the left).

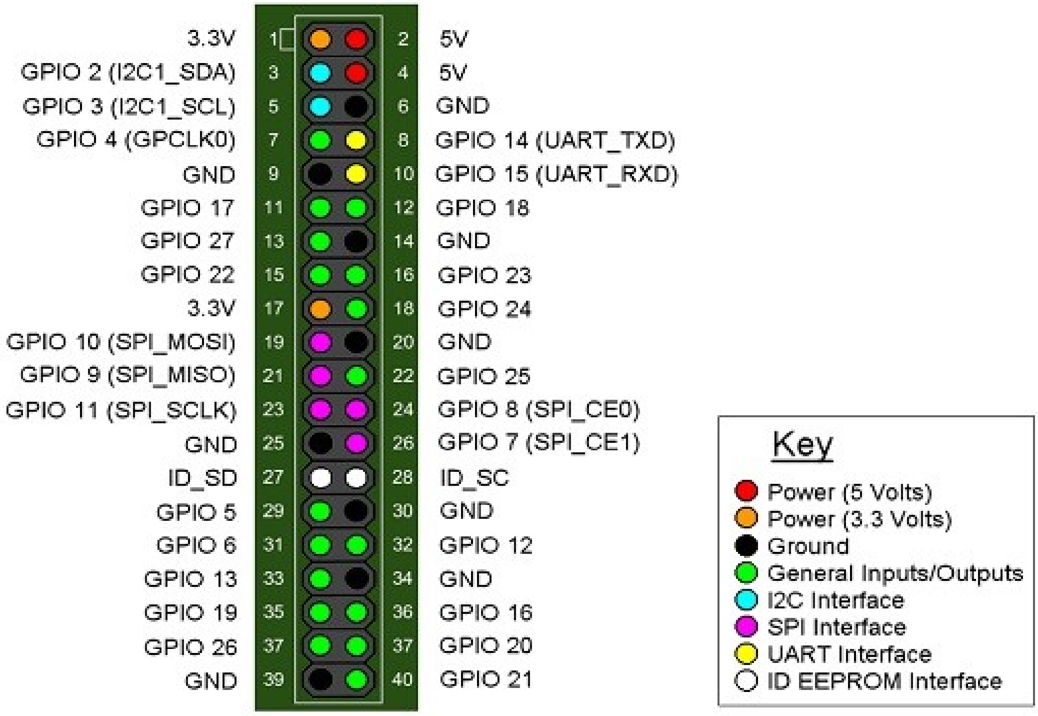


Figure 2: Raspberry Pi 40 Pin connection header pinout (Source: http://www.rs-online.com)

There is also a great pinout at Gadgetoid
http://pi.gadgetoid.com/pinout

2 Digital IO

If you are new to GPIO usage on the Raspberry Pi, the following link might be useful.

https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md

The following table shows the IO assignment of the C1 board to the Raspberry Pi's header connector.

Raspberry Pi Header Pin (physical)	GPIO Number BCM2836	CONMELEON C1 Screw Terminal
38	20	Digital IN 1 (24V)
40	21	Digital IN 2 (24V)
15	22	Digital IN 3 (24V)
16	23	Digital IN 4 (24V)
18	24	Digital OUT 1 (Relay)
22	25	Digital OUT 2 (Relay)
37	26	Digital OUT 3 (Relay)
13	27	Digital OUT 4 (Relay)

We will need the GPIO numbers later, when accessing the digital IO via software.

2.1 Digital Input Circuit

Figure 3 shows the circuit details.

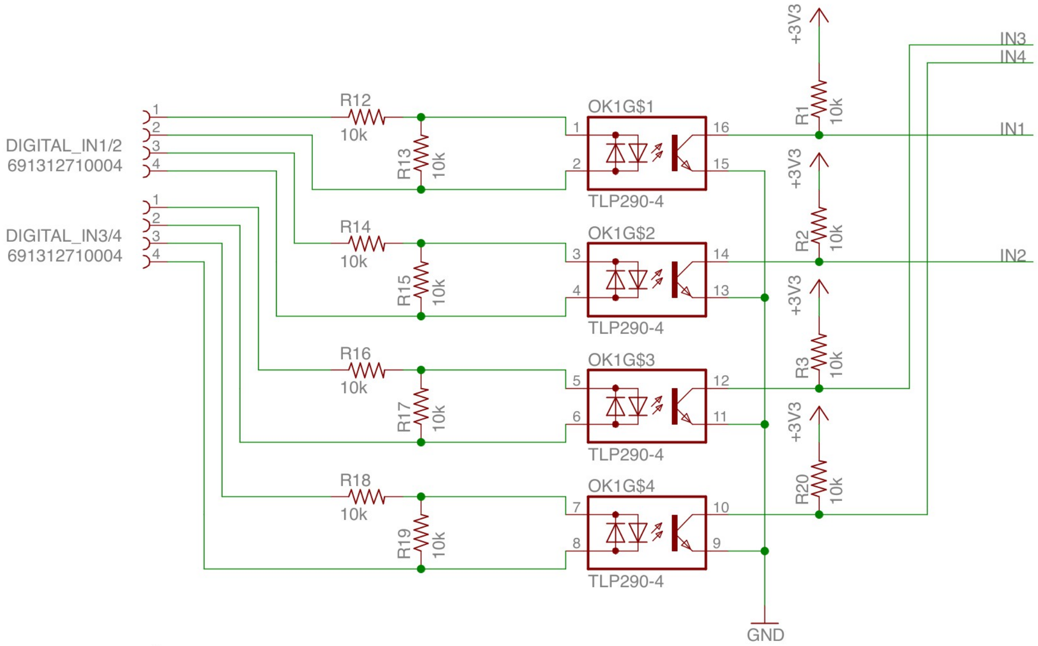


Figure 3: CONMELEON C1 24V Digital Input Circuit

The digital inputs are operated at a HTL level of 24V. Therefore the GPIO inputs of the Raspberry Pi are protected and isolated by Toshiba TLP290 photo couplers.

The resistors R12, R14, R16 and R18 are limiting the forward current for the LED of the photo coupler. R13, R15, R17 and R19 are providing an additional threshold for the forward current to get a slightly higher diode forward voltage than the normal 1,1 V. R1, R2, R3 and R20 are normal pull-up resistors to get a defined logic level at the GPIO pins of the Raspberry Pi. Due to the input circuit, the digital inputs are inverted. For 0V at the input you will get true at the GPIO input and 24V at the input will lead to false.

The voltage at the digital inputs needs to be at least 5,5V to get a logical TRUE.

2.2 Digital Output Circuit

The CONMELEON C1 board has 4 digital outputs, which are able to switch higher loads with built in relays (type OMRON G5Q-14, single pole double throw with 5 VDC rated coil voltage).

The maximum resistive load the relays are able to switch is 3A @ 30VDC or 3A @ 230VAC.

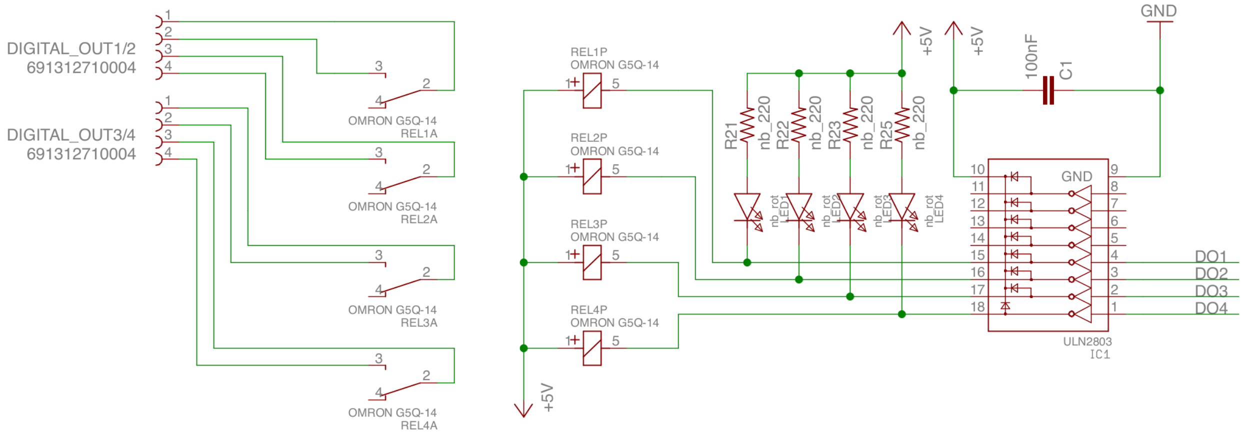


Figure 4: CONMELEON C1 Digital Output Circuit

To be able to switch the inductive load of the relay coil with the Raspberry Pi GPIO's a Darlington transistor array (IC1 ULN2803) is used. The current switching condition of every output is shown with LED's (LED1, LED2, LED3 and LED4). Please note the current limiting resistors R21, R22, R23 and R25 for the LED's to allow a maximum LED current of 20mA. The capacitor C1 is used as a decoupling capacitor to keep off voltage spikes from the power supply pins of IC1.

2.3 Interfacing the GPIO via SYSFS

SYSFS has the advantage that it is part of the Linux Kernel and therefore available on almost every Linux based platform. SYSFS is providing access (reading and writing) via simple file handling.

Per default the GPIO files (e.g. /sys/class/gpio/export) are owned by the *root* user, but also belong to the *gpio* group. If you want to use the remote debugging features in Eclipse CDT, it is quite helpful to add the user, which is accessing the Raspberry Pi to the *gpio* group. SSH into your Raspberry Pi and enter the following command:

```
herwig@CONMELEON ~ $ sudo adduser herwig gpio
Adding user `herwig' to group `gpio' ...
Adding user herwig to group gpio
Done.
```

Please note that with this configuration it is still not possible to access the GPIO's with an executable with any other user than root. This behavior is by default for security reasons.

You can launch a root shell on the Raspberry Pi with the following command:

```
herwig@CONMELEON ~ $ sudo -i
```

Within this shell you can start your executable and you will have full access to the SYSFS GPIO related files.

2.3.1 Programming the GPIO via SYSFS

The programming of the GPIO to use them as digital inputs or outputs is pretty straight forward. In general you have to follow the following procedure:

1. Export the pin to be able to use it
   Write the GPIO pin number (e.g. 20 for DI 1 of the CONMELEON C1 board) to the file */sys/class/gpio/export*
2. Set the pin direction to input or output
   Write “in” or “out” to the file */sys/class/gpio/gpio20/direction*
3. Write or read the pin value
   Read or write “1” for true and “0” for false to the file */sys/class/gpio/gpio20/value*
4. Unexport the pin if you don't need it anymore
   Write the GPIO pin number (e.g. 20 for DI 1 of the CONMELEON C1 board) to the file */sys/class/gpio/unexport*

3 Analog IO

The CONMELEON C1 board provides analog inputs only.

3.1 Analog Input Circuit

The CONMELEON C1 board uses a 12bit 4 channel ΔΣ ADC (Analog Digital Converter) which is connected via SPI bus to the Raspberry Pi. The ADC is a ADS1018 from Texas Instruments, with a programmable data rate up to 3300 samples per second and an internal programmable gain amplifier. The channels are configured in single ended mode and allow an input voltage level from 0V to +10V.

Figure 5 shows the pinout of the ADS1018.

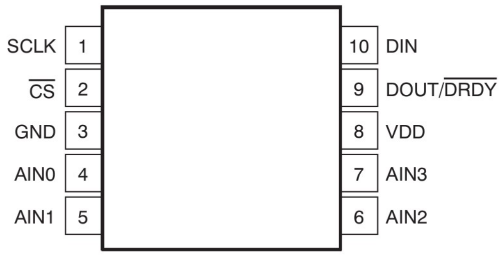


Figure 5: ADS1018 pinout

The maximum input voltage level of the ADS1018 is limited to VDD + 0,3 V. As you can see in Figure 6 the VDD voltage of the CONMELEON C1 board is 3,3V. Therefore a voltage divider (e.g. R4 and R5 for AIN0) is used to scale down the nominal maximum input voltage of 10V to the allowed 3,3V.

Never apply larger voltages than 10V DC to any of the analog inputs of the CONMELEON C1 board! Higher voltages will destroy the ADS1018!

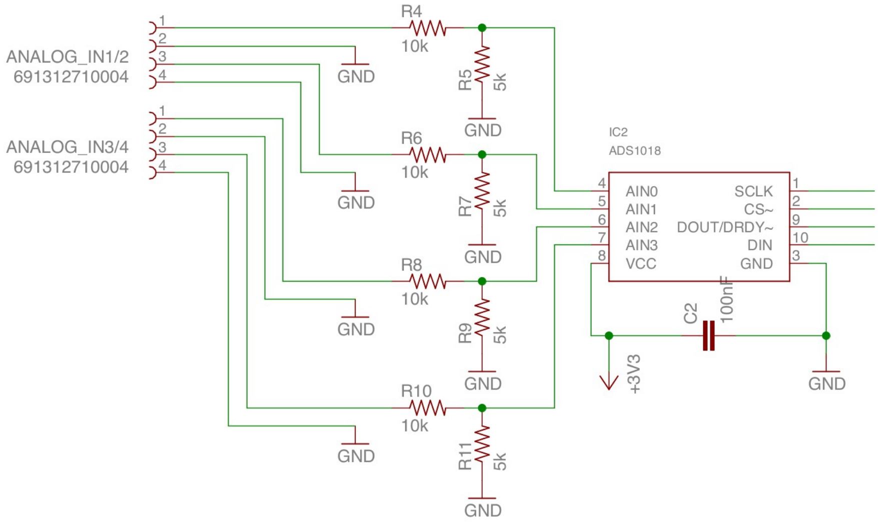


Figure 6: CONMELEON C1 analog input circuit

Raspberry Pi Header Pin (physical)	ADS1018 pin	Description
19	10	DIN / MOSI
21	9	DOUT / MISO
23	1	SCLK
26	2	CS

3.2 Interfacing the SPI Bus with Linux Kernel Device Driver

A nice startup for the SPI bus on the Raspberry Pi is available at https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md

A great tutorial on interfacing an ADC via SPI bus on the Raspberry Pi can be found here http://hertaville.com/interfacing-an-spi-adc-mcp3008-chip-to-the-raspberry-pi-using-c/

Hussam Al-Hertani uses a MCP3008 from Microchip for his tutorial instead of the TI ADS1018, but nevertheless the general procedure is the same.

We will use the linux kernel SPI driver kernel module for accessing the SPI interface. This is slower than driver memory mapping or direct register access of the BCM2836 SoC of the Raspberry Pi 2 B but it is highly portable and for our applications the speed is way enough.

We need to enable the SPI kernel driver first, just in case this is not done yet.

3.2.1 Enabling the SPI Driver

The most easy way to enable the SPI driver is via the raspi-config utility. Which can be started with the following command:

```
herwig@conmeleon ~ $ sudo raspi-config
```

The SPI driver can be found in the advanced options.

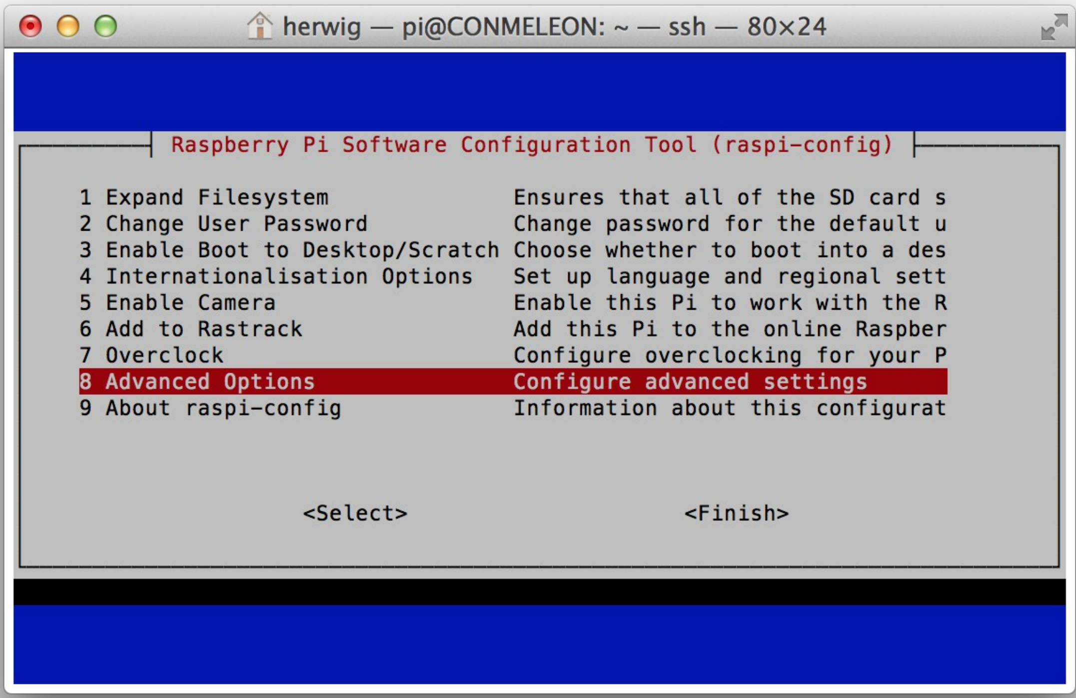


Figure 7: Raspberry Pi Configuration Utility

Just answer that you want to enable the SPI driver and that the kernel module should be loaded at boot time and you are done.

If you want to do the driver enabling the hard way, you can follow the instructions below.

First check if the driver is already loaded. SSH into the device and enter the following command

```
herwig@conmeleon ~ $ ls /dev/spidev*
```

This command should produce the following output

```
herwig@conmeleon ~ $ ls /dev/spidev*
/dev/spidev0.0 /dev/spidev0.1
```

If it is not, you can double check if the kernel driver is loaded with the command

```
herwig@conmeleon ~ $ lsmod
Module                  Size  Used by
uio_pdrv_genirq         2958  0
uio                     8119  1 uio_pdrv_genirq
snd_bcm2835            18850  0
snd_pcm                75388  1 snd_bcm2835
snd_timer              17784  1 snd_pcm
snd                    51667  3 snd_bcm2835,snd_timer,snd_pcm
fuse                   82155  1
ipv6                  333229  22
```

If you don't see an entry called *spi_bcm2708*, you can be 100% sure, that the kernel driver is not loaded.

Kernels starting from version 3.18 don't support the original */etc/modprobe.d/raspi-blacklist.conf* file anymore.

To check your kernel version enter the following command.

```
herwig@conmeleon ~ $ cat /proc/version
Linux version 3.18.7-v7+ (dc4@dc4-XPS13-9333) (gcc version 4.8.3
20140303 (prerelease) (crosstool-NG linaro-1.13.1+bzr2650 - Linaro GCC
2014.03) ) #755 SMP PREEMPT Thu Feb 12 17:20:48 GMT 2015
```

Edit the file */boot/config.txt* instead.

```
herwig@conmeleon ~ $ sudo nano /boot/config.txt
```

Add the line shown in Figure 8 at the bottom of the file.

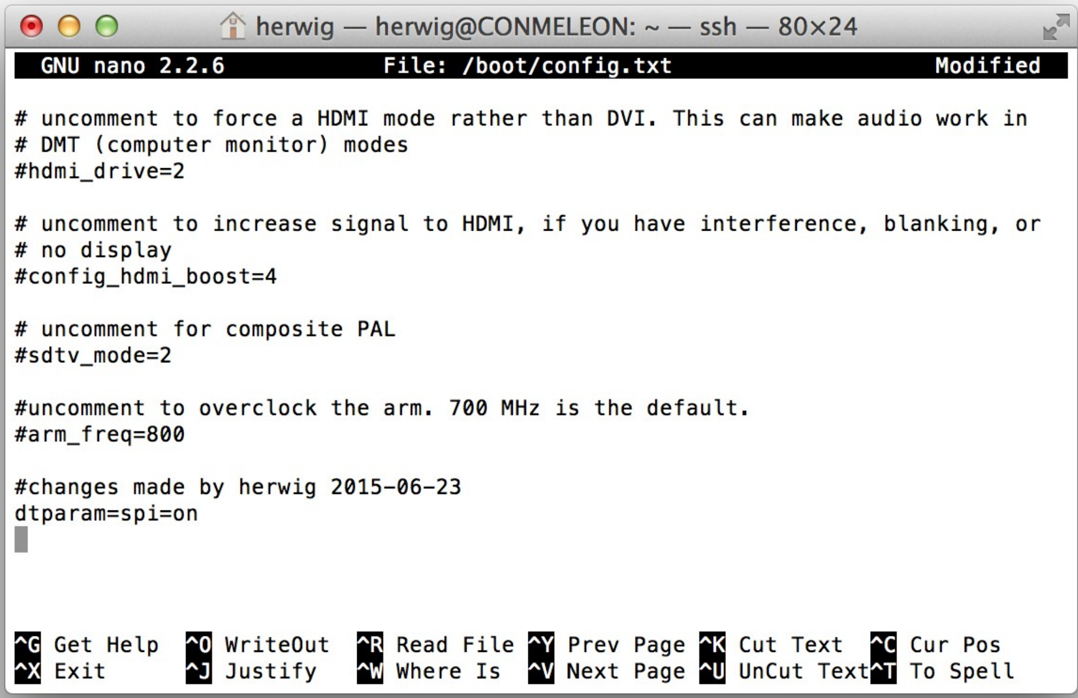


Figure 8: Edit /boot/config.txt

Save the file with Ctrl+O and exit with Ctrl+X.

Reboot the Raspberry Pi with

```
herwig@conmeleon ~ $ sudo reboot now
```

You should now see the two SPI devices */dev/spidev0.0* and */dev/spidev0.1*.

> The chip select input pin of the ADS1018 is connected to pin 26 of the Raspberry Pi, which is CE1. Therefore the ADS1018 can be addressed via /dev/spidev0.1

3.2.2 Create SPI Permissions

Per default the SPI devices on the Raspberry Pi can be accessed with user *root* only. We'd want to change this, because life will be much easier when using the remote debugger of Eclipse CDT for development.

First create a new “spi” group.

```
herwig@conmeleon ~ $ sudo groupadd -f --system spi
```

Next step is to add the user, who should have access to the SPI devices to the new group.

```
herwig@conmeleon ~ $ sudo groupadd -f --system spi

herwig@CONMELEON ~ $ sudo adduser herwig spi
Adding user `herwig' to group `spi' ...
Adding user herwig to group spi
Done.
herwig@CONMELEON ~ $
```

Now we will have to assign the spidev subsystem to the spi group we just created.
For this we will have to create the file */etc/udev/rules.d/90-spi.rules*.

```
herwig@CONMELEON ~ $ cd /etc/udev/rules.d/
herwig@CONMELEON /etc/udev/rules.d $ sudo nano 90-spi.rules
```

Enter the line as shown in Figure 9.

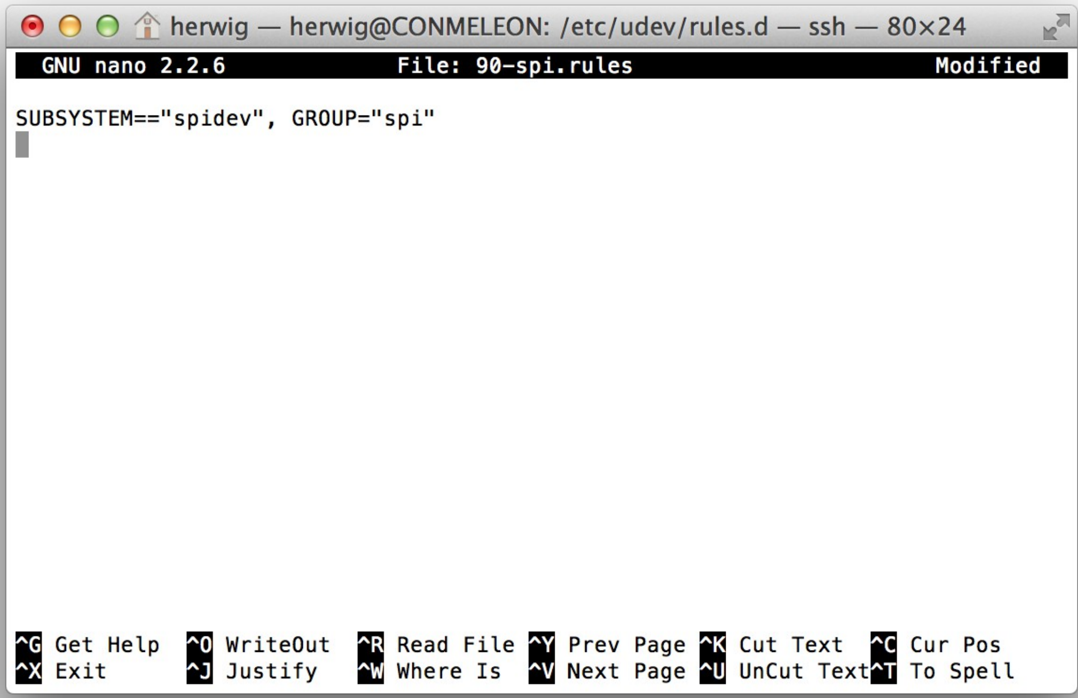


Figure 9: Create 90-spi.rules file for spidev

Save the file with Ctrl+O and exit with Ctrl+X. Reboot the Raspberry Pi to apply the changes.

3.3 Interfacing the SPI Bus with BCM2835 Library

The BCM2835 library is an alternative to the Linux Kernel device driver approach. It is much faster, because it is addressing the BCM2835 (or BCM2836 of the Raspberry Pi 2) registers directly. The disadvantage is, that it is coded especially for the Raspberry Pi and is not portable to other Linux based platforms (e.g. the BeagleBone Black).
To be able to use the library with the BCM2836 of the Raspberry Pi 2 B we will need at least library version 1.39.

The library and a lot of useful documentation can be found at

http://www.airspayce.com/mikem/bcm2835/index.html

3.3.1 Enabling the Device Tree Support

The BCM2835 library needs the support for the device tree enabled to work properly. It is also necessary to disable the SPI Kernel Driver, otherwise it will interfere with the library, which is accessing the registers of the BCM2835 (BCM2836) chip directly. The best way to enable the device tree support is via the *raspi-config* utility. The device tree support can be activated by selecting *8 advanced options* and then *A5 Device Tree*.

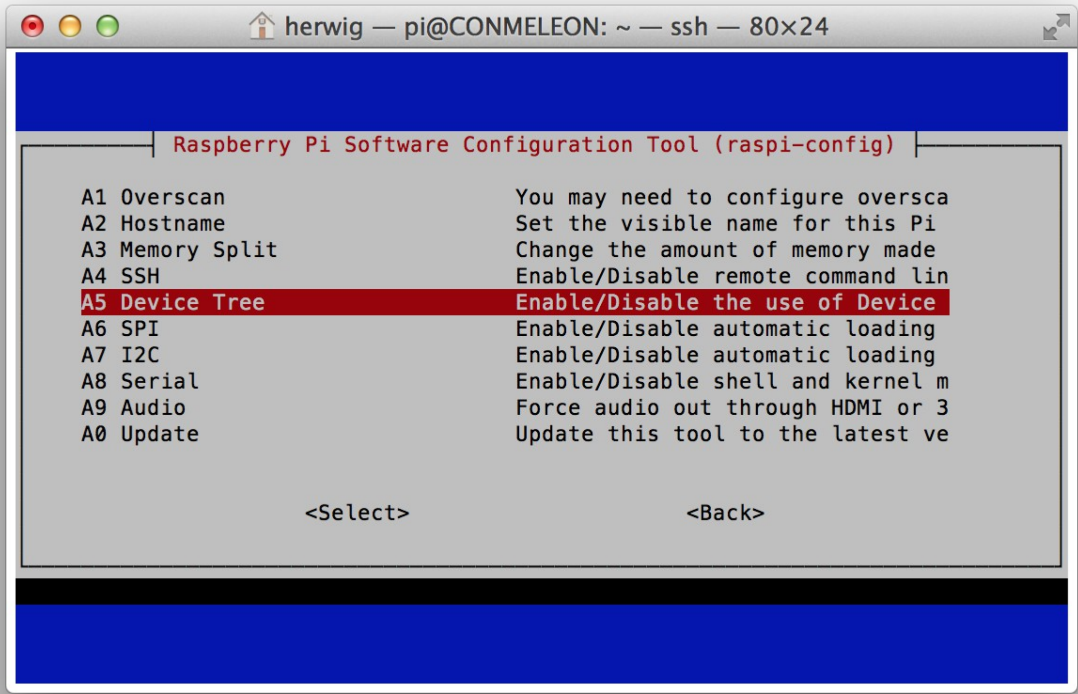


Figure 10: Enable Device Tree Support

Reboot after the change.

Now you can check if the device tree is enabled properly. Change to the */proc/device-tree/soc* directory and check if the *ranges* file is present.

```
herwig@CONMELEON ~ $ cd /proc/device-tree/soc/
herwig@CONMELEON /proc/device-tree/soc $ ls ranges
ranges
herwig@CONMELEON /proc/device-tree/soc $
```

3.3.2 Installing the BCM2835 Library

So first of all the BCM2835 library needs to be built on the Raspberry Pi 2 B.

```
herwig@conmeleon ~ $ wget
http://www.airspayce.com/mikem/bcm2835/bcm2835-1.46.tar.gz
herwig@conmeleon ~ $ tar zxvf bcm2835-1.46.tar.gz
herwig@conmeleon ~ $ cd bcm2835-1.46/
herwig@conmeleon ~/bcm2835-1.46 $ ./configure
herwig@conmeleon ~/bcm2835-1.46 $ make
herwig@conmeleon ~/bcm2835-1.46 $ sudo make check
herwig@conmeleon ~/bcm2835-1.46 $ sudo make install
```

This will build and install the library *libbcm2835.a* into the */usr/local/lib* directory and copy the file *bcm2835.h* to the */usr/local/include* directory. It will also install the kernel module it is using.

3.4 ADS1018 SPI Communication and Configuration

The ADS1018 data sheet is providing lots of information.

One of the important things is the maximum clock rate of the SCLK signal. This value is given with a minimum clock period of 250 ns (parameter t_{sclk} given on page 5 of the data sheet).So we must not set the bus SCLK frequency of the Raspberry Pi higher than 4 MHz. As the maximum data rate of the ADS1018 is 3300 samples per second, a SCLK frequency of around 1 MHz should be fine.

The SPI bus of the Raspberry Pi SoC provides possible clock rates in multiples of two:

clock divider	clock speed
2	125 MHz
4	62,5 MHz
8	31,2 MHz
16	15,6 MHz
32	7,8 MHz
64	3,9 MHz
128	1953 kHz
256	976 kHz
512	488 kHz
1024	244 kHz

2048	122 kHz
4096	61 kHz
8192	30,5 kHz
16384	15,2 kHz
32768	7629 Hz

If you choose a clock rate which is not in the list (e.g. 1 MHz) it will be reduced to the next lower valid setting (e.g. 976kHz) automatically.

The ADS1018 must be operated in SPI mode 1 (clock polarity CPOL = 0, clock phase CPHA = 1). This gives how data will be exchanged between the ADS1018 and the Raspberry Pi SPI bus driver chip.

CPOL = 0	The clock starts in “low” state and the leading edge is rising, the following edge is falling
CPOL = 1	The clock starts in “high” state and the leading edge is falling, the following edge is rising
CPHA = 0	Data is sampled on the leading edge
CPHA = 1	Data is sampled on the trailing edge

So a valid signal on the MISO or MOSI line needs to be present at the rising edge of the clock and will be latched to the internal register at the falling edge of the clock.

3.4.1 ADS1018 Configuration Register Settings

Both the configuration register and the conversion (data) register will be written and read with MSB (Most Significant Byte) first.

The following table shows the configuration register settings recommended for the CONMELEON C1 board.

	bit number	value	name in datasheet	description
MSB	15	0	SS	trigger bit for starting single shot conversion
	14	1	MUX2	channel selection, we use only single ended, AIN0 = 100, AIN1 = 101, AIN2 = 110, AIN3 = 111
	13	0	MUX1	
	12	0	MUX0	
	11	0	PGA2	programmable gain amplifier, we set the full scale voltage to 4,096V = 001
	10	0	PGA1	
	9	1	PGA0	
	8	1	MODE	single shot (1) or continuous (0) sampling
LSB	7	1	DR2	data rate (sampling speed), we set to full speed at 3300 sps
	6	1	DR1	
	5	0	DR0	
	4	0	TS_MODE	read the internal temperature sensor (1), but we will use the normal analog inputs (0)
	3	1	PULL_UP_EN	internal weak pull up resistor active (1) or inactive (0)
	2	0	NOP1	no operation
	1	1	NOP0	
	0	1	NOT USED	always 1

If we convert the MSB and LSB to hex values we will get:

reading channel AI 1 (AIN0 in ADS1018 language):
MSB = 0x43 (C3 with single shot trigger bit 15 set) and LSB = 0xCB

reading channel AI 2 (AIN1):
MSB = 0x53 (D3 with single shot trigger bit 15 set) and LSB = 0xCB

reading channel AI 3 (AIN2):
MSB = 0x63 (E3 with single shot trigger bit 15 set) and LSB = 0xCB

reading channel AI 4 (AIN3):
MSB = 0x73 (F3 with single shot trigger bit 15 set) and LSB = 0xCB

3.4.2 Example Code using BCM2835 Library

The following C/C++ code snippet shows a very easy data transfer to and from the ADS1018 just reading the first input channel AIN0.

```
#include "bcm2835.h"
#include <unistd.h> // needed for usleep() function

int main() {

      if (!bcm2835_init())
              return 1;
      bcm2835_spi_begin();
      bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
      bcm2835_spi_setDataMode(BCM2835_SPI_MODE1);
      bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);
      bcm2835_spi_chipSelect(BCM2835_SPI_CS1);
      bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS1, LOW);

      char TxData[4];
      char RxData[4];

      // we are transferring data in 32 bit mode, according to the ADS1018
      // datasheet, we should write the configuration register two times in a
row

      // read channel AIN0 in single shot mode

      TxData[0] = 0xC3;
      TxData[1] = 0xCB;
      TxData[2] = 0xC3;
      TxData[3] = 0xCB;
      RxData[0] = 0x00;
      RxData[1] = 0x00;
      RxData[2] = 0x00;
      RxData[3] = 0x00;

            // do the whole thing 10 times, just because we can
```

```
	for (int i = 0; i < 10; i++) {
		// configure for a single shot conversion, reading makes no sense
		// because the conversion is not finished yet
		bcm2835_spi_writenb(TxData, 4);
		// data rate is set to 3300 samples per second, so we will get one
every 303 mysec minimum, so we wait for 350 microseconds and now we can also
read the data, the BCM3835 library is not providing a readnb() function, so we
use transfernb instead
		usleep(350);
		// now we remove the single shot conversion trigger bit, because we
don't want to wait another 350 microseconds
		TxData[0] = 0x43;
		TxData[2] = 0x43;
		bcm2835_spi_transfernb(TxData,RxData, 4);
		// the answer of the ADS1018 will give MSB of conversion value in
RxData[0], LSB of conversion value in RxData[1], MSB of current configuration
register RxData[2], LSB of current configuration register RxData[3]
		// please note that the ADS1018 will always return the single shot
trigger bit as 0, so if you transfer the MSB 0xC3 you will receive the MSB 0x43
from the ADS1018
		printf("Read cycle %d, bytes read from ADS1018 channel 0: %2.2X,
%2.2X, %2.2X, %2.2X\n", i, RxData[0], RxData[1], RxData[2], RxData[3]);
	}
		// BCM2835 library clean up
	bcm2835_spi_end();
	bcm2835_close();
	return 0;

}
```

3.4.3 Converting the Read Integer Value to Voltage

The ADS1018 provides the converted value of the input in twos complement value. Due to the fact, that we don't use differential inputs but single ended ones, we will loose one bit of the 12 bits resolution of the ADS1018. Another fact is that the ADS1018 internal reference voltage (the same as the supply voltage by design) for conversion is 3,3 V. The next possible setting of the programmable gain amplifier is 4,096 V so we loose again, because we will not get a full scale output from the ADS1018.

First we need to rearrange our two 8 Bit MSB and LSB value of the conversion register to a new 16 Bit value. This can be done with a little bit shifting and some other magical tricks ;-).

The following C/C++ code snippet deals with this matter:

```
unsigned int	nRawVoltage = 0;
// used for bit shifting, unsigned int will be 16 Bits on most systems, at least
is is on the Raspberry Pi

// copy data register bytes to 16 bit integer
// shift MSB 8 bits to the left and or in LSB
nRawVoltage = (RxData[0] << 8) | RxData[1];

// now shift integer 4 bits to the right, because the 12 bit voltage value of
```

```
the ADS1018 is left aligned
nRawVoltage >>= 4;
```

Now we have our 16 Bit integer value exactly how we wanted it to be.

The next things to consider are the following points:

- We have an input voltage divider in the analog input circuit providing a factor of 3,0 and we want to know the voltage level at the input itself (something between 0 V and 10V) so we need this factor of 3.0 for our voltage calculation.
- We have set a full scale voltage with the programmable gain amplifier of 4,096V
- The reference voltage for conversion is 3,3 V (the supply voltage of the ADS1018)

To make the formula a bit more readable, we will use the following abbreviations.

U_{ref}	ADS1018 reference voltage (3,3V)
U_{fs}	full scale voltage (4,096V)
U_{input}	input voltage to be converted
U_{raw}	16 bit raw value received from the ADS1018
f_d	input voltage divider factor (3,0)
ADC_{res}	ADS1018 resolution (12 bit)

$$U_{input} = \frac{\frac{U_{raw} * f_d * U_{fs}}{U_{ref}} * U_{ref}}{2^{(ADC_{res}-1)}} = \frac{U_{raw} * f_d * U_{fs}}{2^{(ADC_{res}-1)}} = \frac{U_{raw} * 3{,}0 * 4{,}096}{2^{(12-1)}} = U_{raw} * 0{,}006$$

Now the only thing we have to do is convert the 16 bit integer value to a floating point type to be able to calculate.

The following C++ (sorry no static_cast in C language ;-)) code snippet shows a possible conversion function. You will need to pass a 4 byte character array holding the reply of the ADS1018 e.g. RxData from the SPI interface example above:

```
float getVoltageValue(char* paData) {

      float             fFullscaleVoltage = 4.096;
      float             fVoltageDivider = 3.0;
      unsigned int      nRawVoltage = 0;  // used for bit shifting

      // copy data register bytes to 16 bit integer
      // shift MSB 8 bits to the left and or in LSB
      nRawVoltage = (paData[0] << 8) | paData[1];

      // now shift integer 4 bits to the right, because the 12 bit voltage value
of the ADS1018 is left aligned
      nRawVoltage >>= 4;

      return
(static_cast<float>(nRawVoltage))*fVoltageDivider*fFullscaleVoltage/2048.0;
}
```