Site Monitoring using Linux Single Board Computers

Willem A. Schreüder AC0KQ
willem@prinmath.com

HamCon Colorado 2016

This talk is available online at

http://www.prinmath.com/ham/talks/

Single Board Computers

- Full Linux boxes *(today's topic)*
  - Raspberry Pi
  - Beaglebone
- Microcontrollers *(not covered)*
  - Arduino
  - PICAXE
  - BASIC Stamp

Linux SBCs

- Runs a full Linux OS
- Usable stand alone computer or server
- Built in connectivity
  - Ethernet networking
  - USB and serial
  - General purpose IO
- Expandable using daughter boards
- Inexpensive ($50 for a working system)

Linux SBC Applications

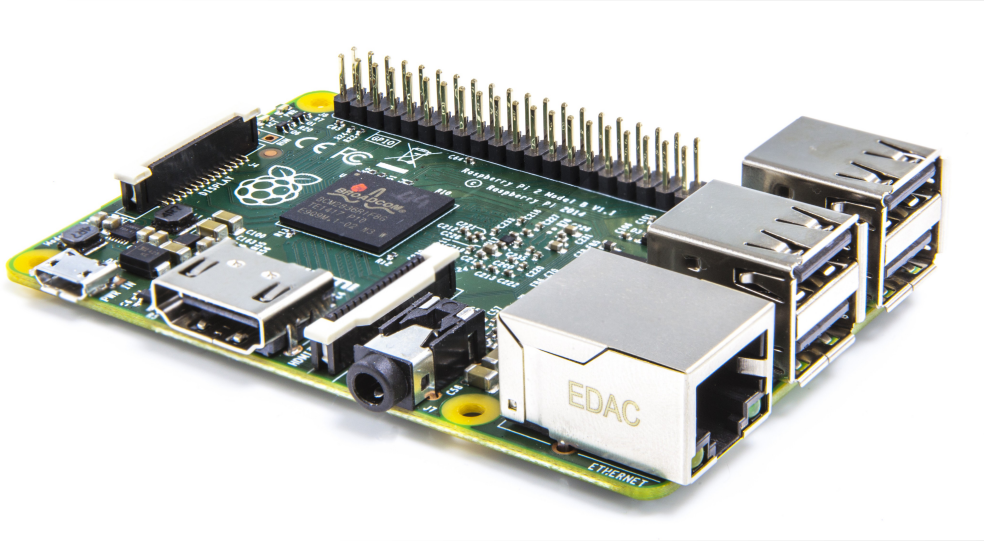
- BPQ
  - Packet, WinLink & APRS
  - TNC daugter board(s)
- AllStarLink
  - Linked Repeaters using URI
- Site Monitor and Control
  - Temperature, voltage, status, relays

Pros and Cons

- Pros
  - Inexpensive
  - No moving parts
  - 5V power
  - Expandable
- Cons
  - SD cards corrupted by bad power

Raspberry Pi

- Most Popular
- Best supported
- rPi2 most powerful
- Lots of USB ports
- Lots of daughterboards
- No analog inputs
- $35 plus SD card

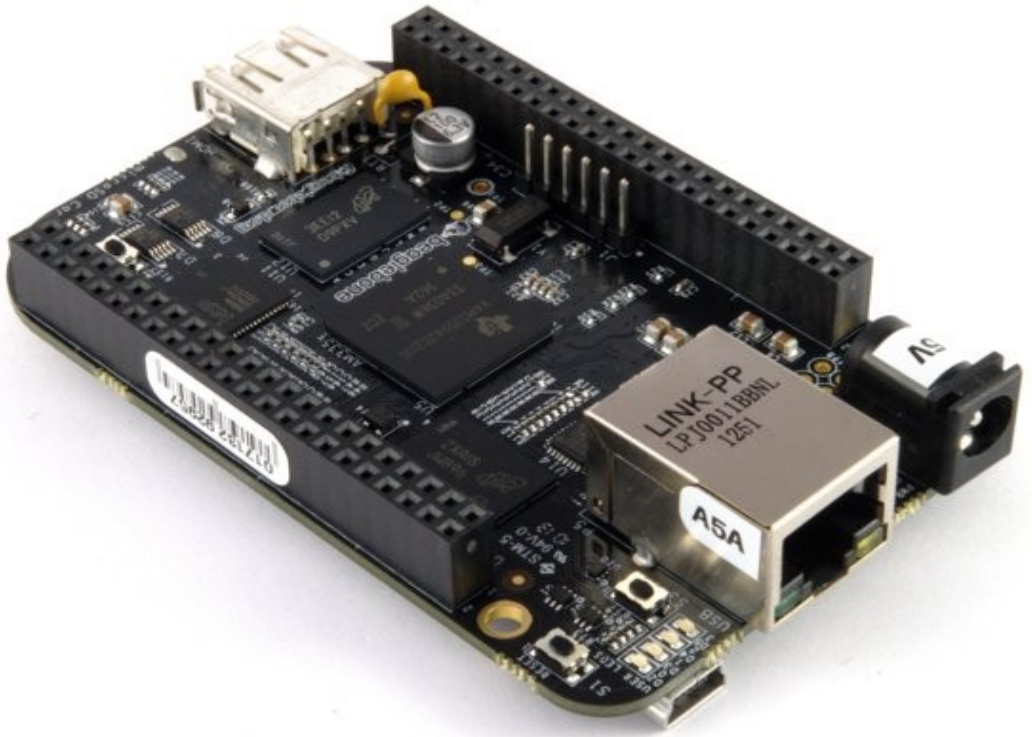


Raspberry Pi models

- Raspberry Pi
  - A/A+ 700MHz CPU & 256MB SDRAM, 1xUSB
  - B 700MHz CPU & 512MB SDRAM, 2xUSB, Ethernet
  - B+ 700MHz CPU & 512MB SDRAM, 4xUSB, Ethernet
  - 2B 900 MHz Quad A7 & 1GB SDRAM, 4xUSB, Ethernet
  - 3B 1.2GHz Quad 64bit & 1GB SDRAM, 4xUSB, Ethernet
- Compute Module
  - 700MHz CPU & 512MB SDRAM
- Zero
  - 1GHz CPU & 512MB SDRAM

BeagleBone

- Less well supported
- Onboard eMMC
- Power & Reset buttons
- More GPIO pins
- 8 analog inputs
- $50 street price



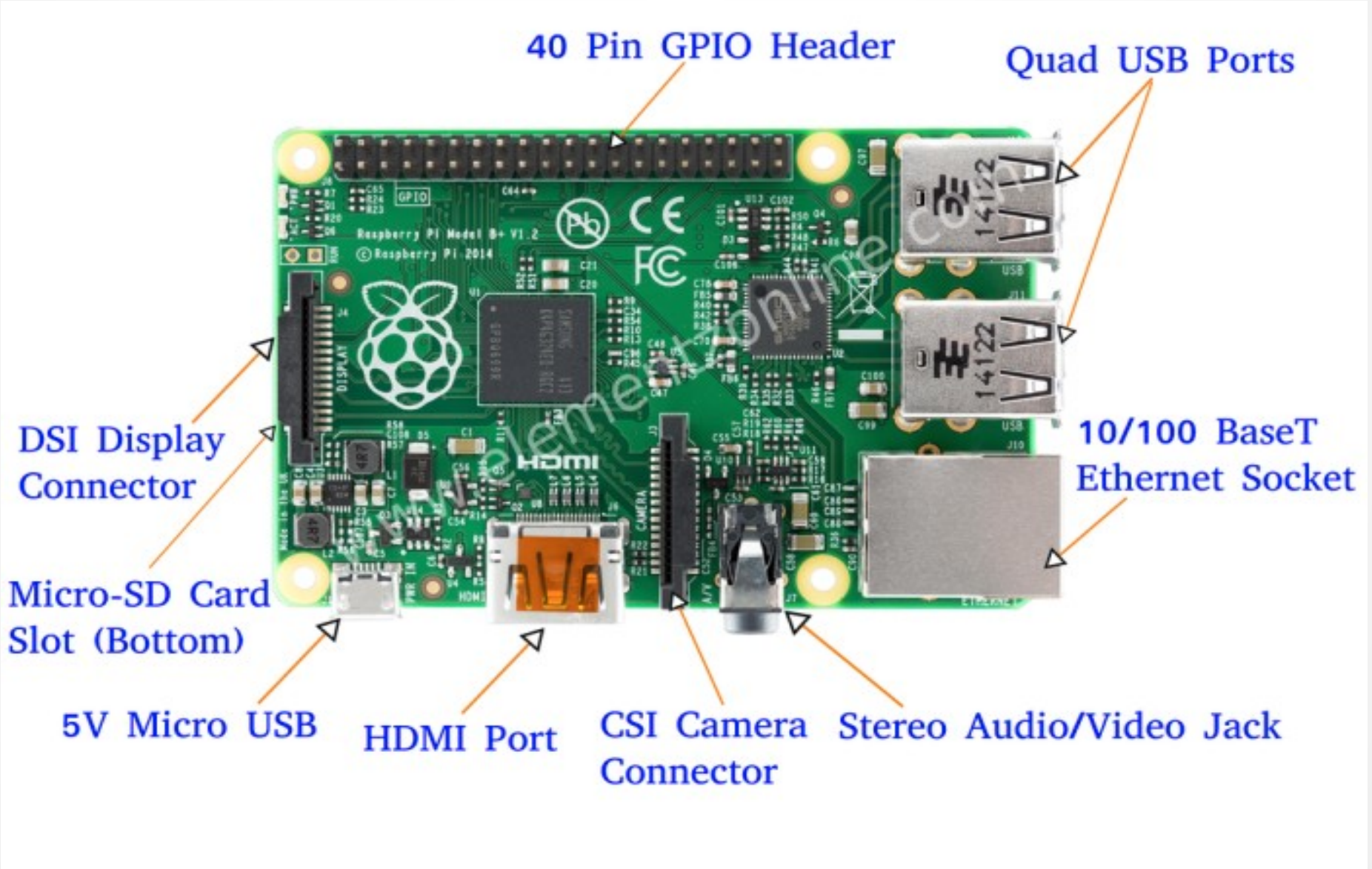
Beaglebone Models

- White
  - Original 720 MHz A8
- Black
  - Most Popular 1GHz A8
- Green
  - Same CPU as Black
  - No barrel power, two Grove connectors

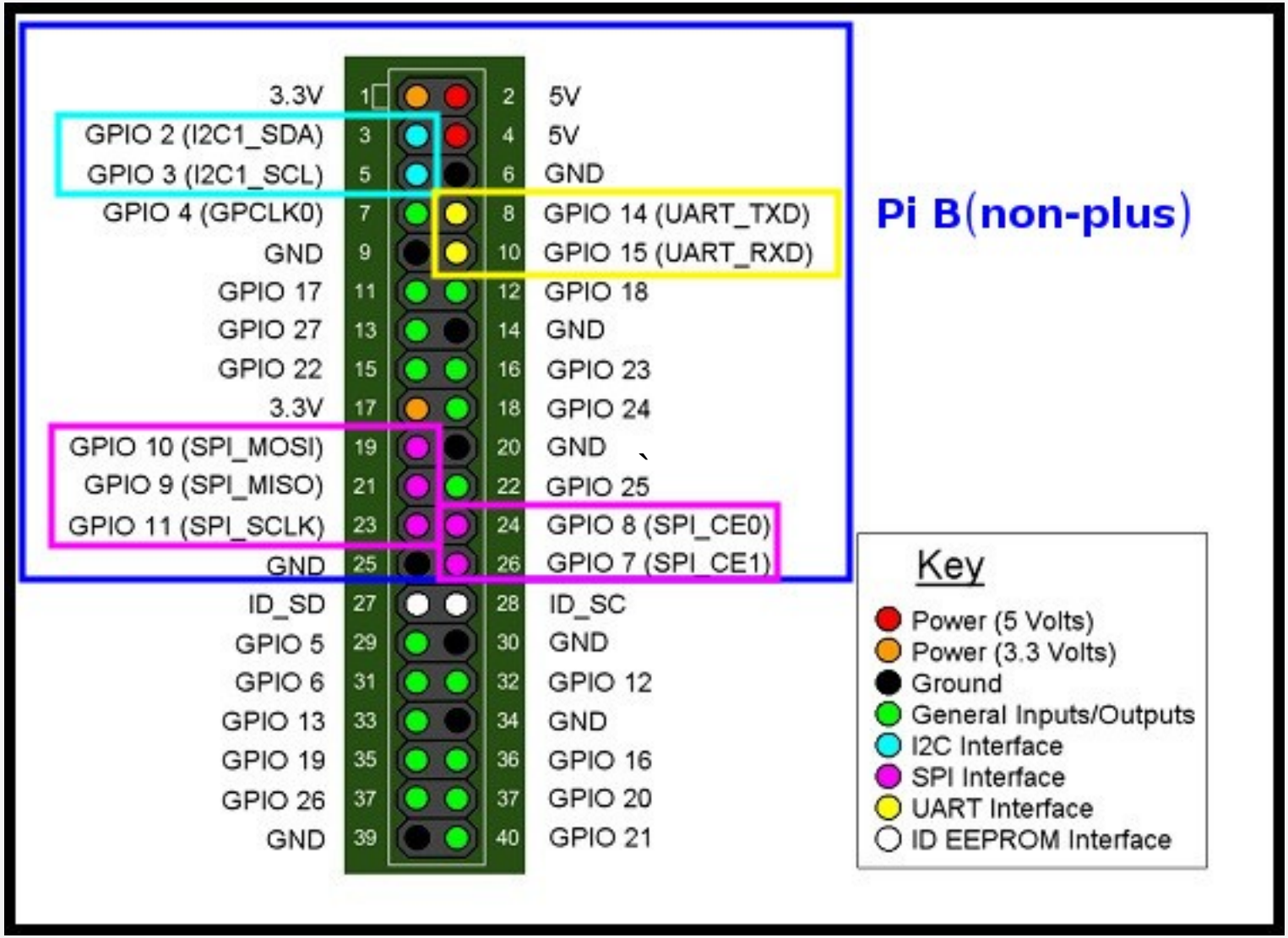
Power and Storage

- Runs on 5V DC
  - Needs clean power
- Micro SD card storage
  - Finite life
  - Bad power kills SD

Raspberry Pi 2B



Raspberry Pi Header



Pins are multiplexed

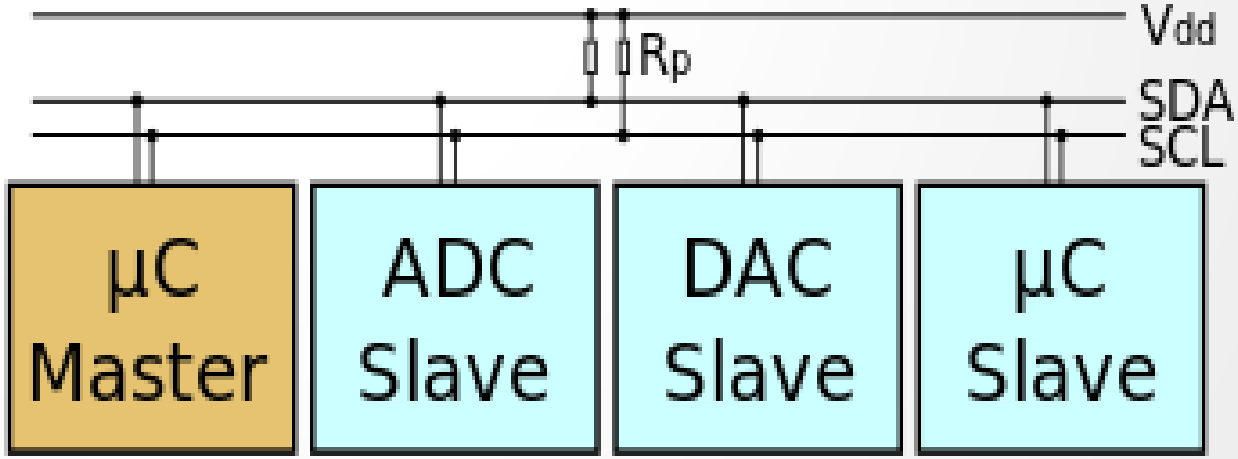
- Pins configured for different uses
- GPIO 14&15 <=> UART TxD/RxD
- GPIO 2&3 <=> I²C SDA&SCL
- GPIO 7&8&9&10&11 <=> SPI MOSI&MISO&SCL&CE0&CE1
- GPIO 18&19 <=> PWM 0&1
- 16-26 GPIO pins

Raspberry Pi Serial

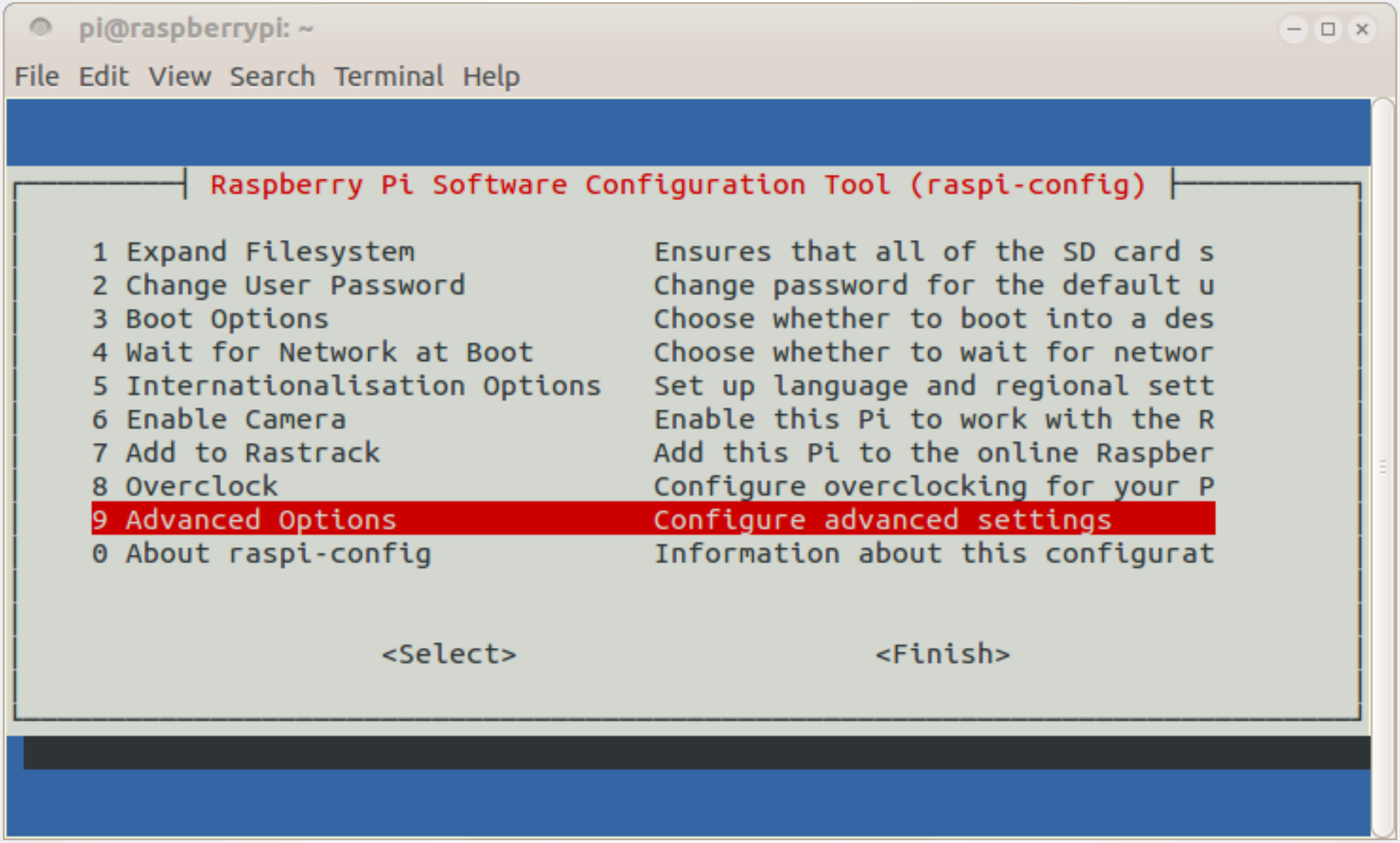
- Single serial port
  - /dev/ttyAMA0
- Speeds up to 115200 bps
- TTL level signals
- By default connected to getty

Raspberry Pi I²C

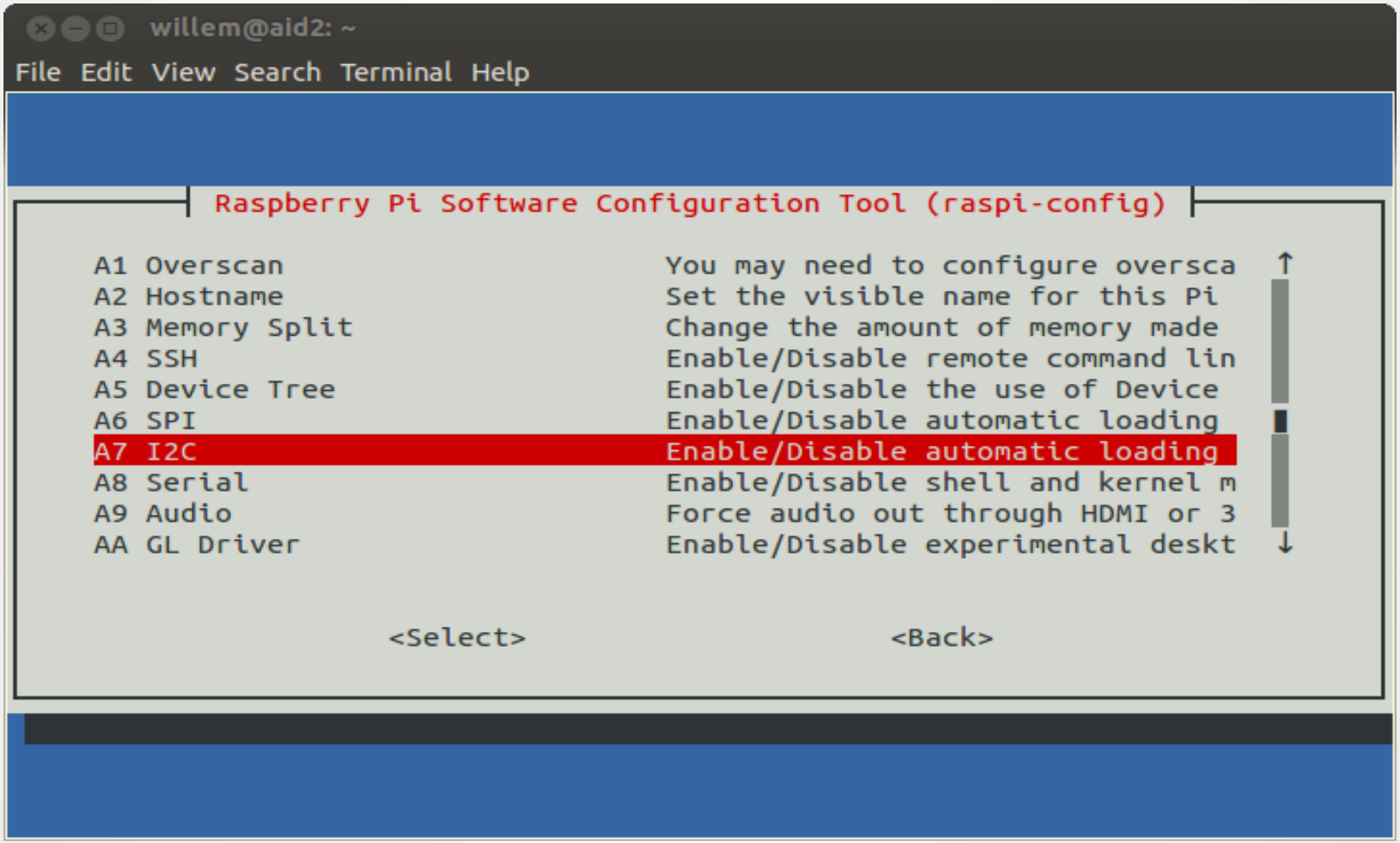
- Inter-Integrated Circuit
  - Serial bus (a.k.a SMBus)
- Default speed 400,000 bps
- rPi has single external I²C bus
  - 127 devices
- Control lines
  - SDA (data)
  - SCL (clock)



Enable I²C with raspi-config 1



Enable I²C with raspi-config 2



Viewing the Bus

```
willem@aid2: ~
File  Edit  View  Search  Terminal  Help
willem@aid2:~ $ sudo apt-get install i2c-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
i2c-tools is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
willem@aid2:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
willem@aid2:~ $
```

i2cdetect outputs

- Scans bus looking for devices

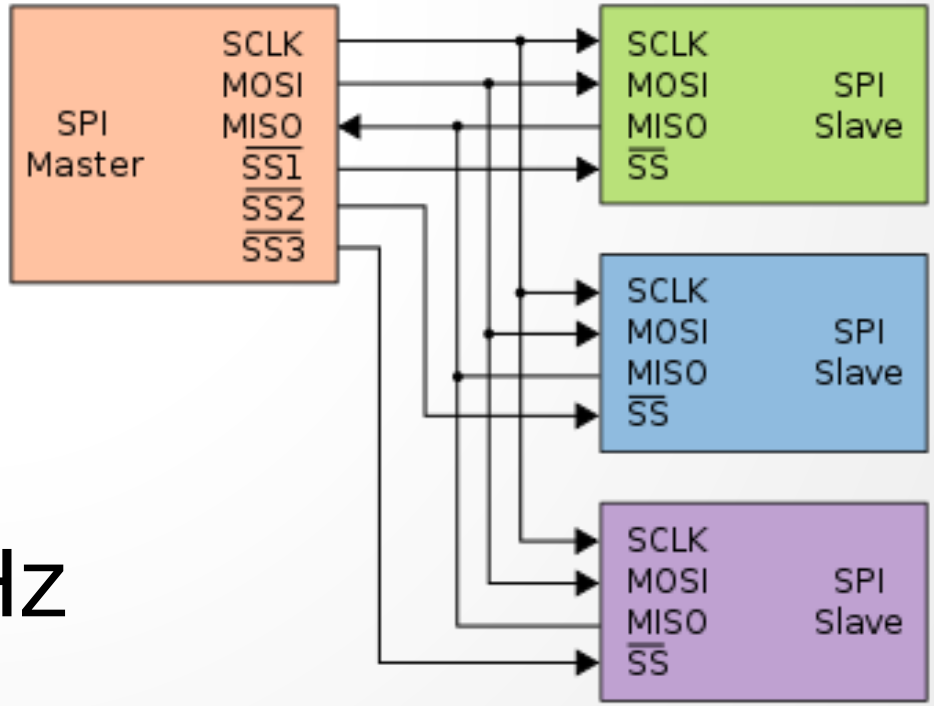
  - - - No Device answered
  - UU Device in use by a driver
  - dd Slave found (dd hex adress)
- Watch for devices in use from user space (e.g. BPQ)
  - can corrupt data

I^2C devices

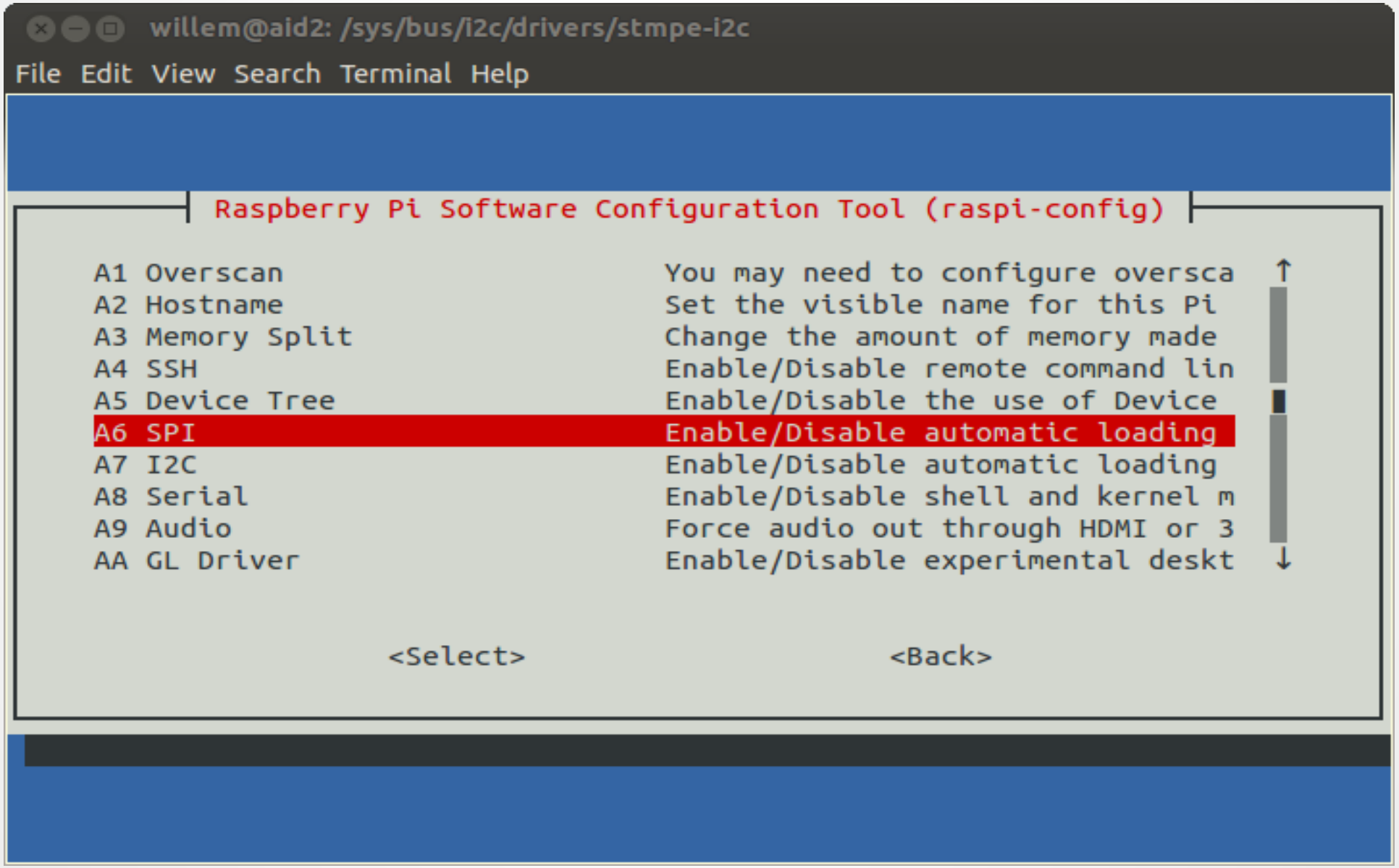
- TNC-Pi
- INA219 current sensor
- Temperature/pressure/RH sensors
- LCD displays
- Accelerometers
- Digitial I/O pins
- Analog<>Digital I/P pins

SPI bus

- Serial Peripheral Interface
- Signals (supports 2 slaves)
  - MasterOutSlaveIn
  - MasterInSlaveOut
  - Clock
  - CE0 (SS1)
  - CE1 (SS2)
- Speeds up to 250 MHz



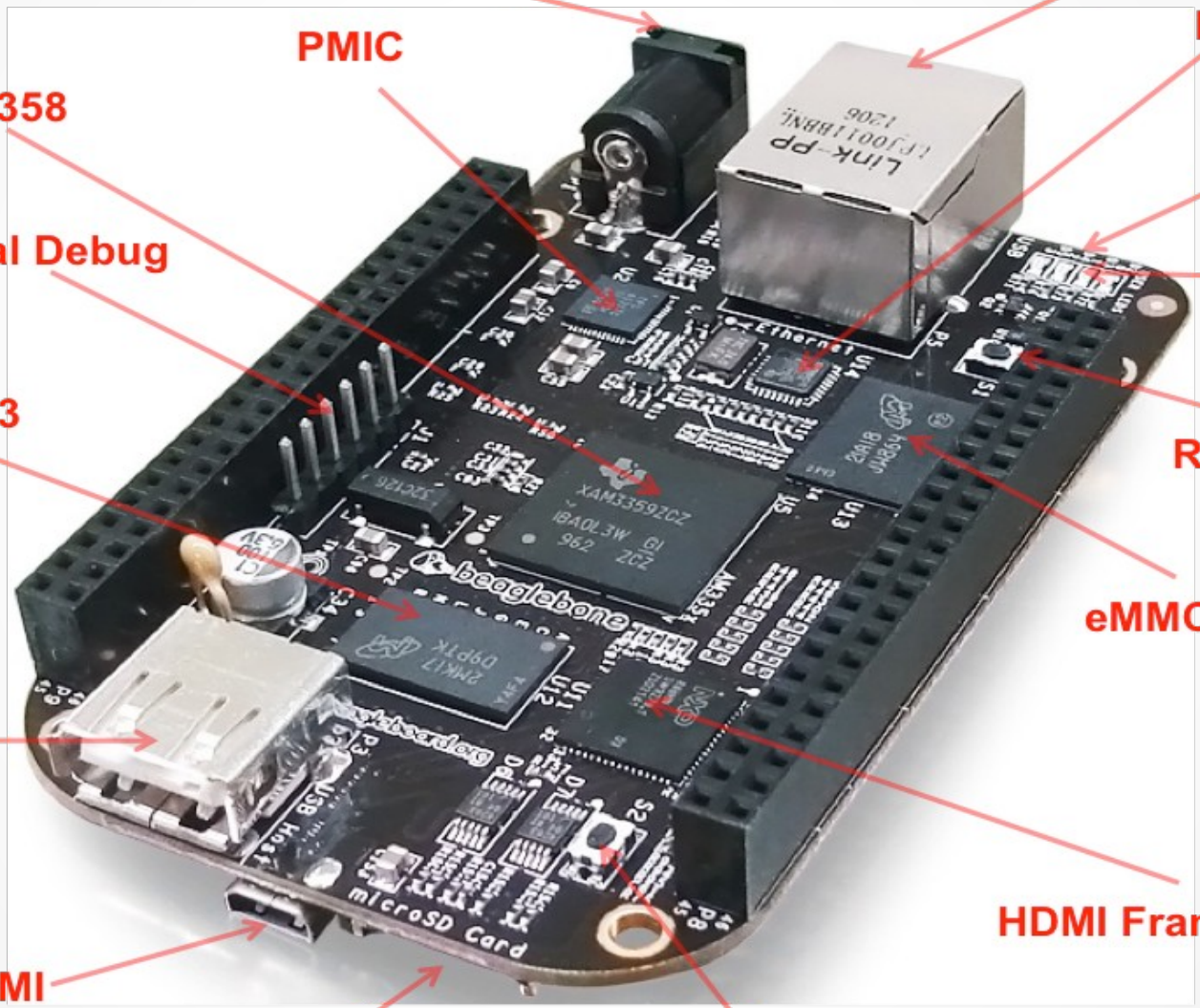
raspi-config enable SPI



SPI Devices

- Faster than I^2C, but uses more pins
- Same devices as I^2C, but adds
  - GPS
  - Ethernet/WiFi/Bluetooth/RFID
  - Memory
- Full duplex

Beagle Bone Black

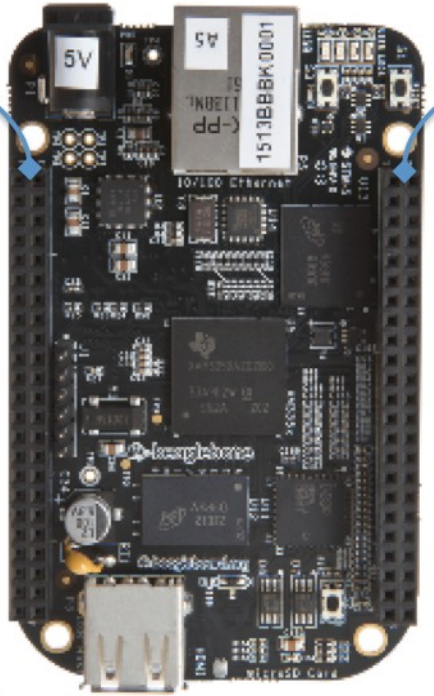


Beagle Bone Black

Cape Expansion Headers

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPI0_CS0	17	18	SPI0_D1
I2C2_SCL	19	20	I2C2_SDA
SPI0_D0	21	22	SPI0_SCLK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SPI1_CS0
SPI1_D0	29	30	GPIO_112
SPI1_SCLK	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPPWM0
DGND	43	44	DGND
DGND	45	46	DGND



LEGEND
Power/Ground/Reset
Available Digital
Available PWM
Shared I2C Bus
Reconfigurable Digital
Analog Inputs (1.8V)

P8

DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DAT0	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

Pins are multiplexed

- Default configuration
  - Power&Reset Buttons
  - 4 serial ports
  - 8 analog inputs (1.8V max)
  - 1 external I^2C bus (127 devices)
  - 19-128 GPIO pins
  - Switched 5V/3.3V DC

Restrictions

- Pins connect directly to CPU
  - Long wires are CPU antennas!
- rPi & BBB GPIO Pins are 3.3 V
  - Max current 16 mA in or out
  - Max combined output current 50 mA
- BB Analog In Pins are 1.8V

Device Tree

- Unix: Everything is a File
- /sys maps to hardware
  - In kernel virtual file system
- Get status by reading
- Set status by writing

Reading analog pins on BBB

- Enable analog pins in device tree

  echo cape-bone-iio>/sys/devices/bone_capemgr.*/slots

- Read value of pin AIN0 in mV

  cat /sys/devices/ocp.*/helper.*/AIN0

  580

- **Voltage on pin AIN0 is 0.580V**

Show pin voltages in Python 1

```
#!/usr/bin/python

for i in range(0,8):
    #  Snarf file
    fd = open("/sys/devices/ocp.3/helper.16/AIN%d" % i)
    text = fd.read()
    fd.close()
    #  Decode voltage
    V = float(text)/1000
    #  Print voltage
    print "AIN%d = %5.3fV" % (i,V)
```

Show pin voltages in Python 2

./aread

```
AIN0 = 1.740V
AIN1 = 1.481V
AIN2 = 1.645V
AIN3 = 0.867V
AIN4 = 0.589V
AIN5 = 0.709V
AIN6 = 0.852V
AIN7 = 1.678V
```

Limitations

- Maximum voltage is **1.8V**
- Use a voltage divider to increase
  - Use 1% or better resistors
  - Max 1 kohm for lower leg
- No analog in on rPi
  - use MCP3008 or similar and SPI

Assigning pins to GPIO

- /sys/class/gpio/export
  - Maps pin to GPIO
  - **echo 18 > /sys/class/gpio/export**
- /sys/class/gpio/unexport
  - Removes pin from GPIO map
  - **echo 18 > /sys/class/gpio/unexport**
- Root access required

Manipulating GPIO

- When mapped to GPIO, a new directory is created for that pin
  - /sys/class/gpio/gpioXX
- Files in this directory controls pin
  - direction = in or out
  - value = 0 or 1

Checking pin value

- In or out?
  - **cat /sys/class/gpio/gpio18/direction**
- High or low?
  - **cat /sys/class/gpio/gpio18/value**

Changing the GPIO direction

- Set pin for input
  - echo in > /sys/class/gpio/gpio18/direction
- Set pin for output
  - echo out > /sys/class/gpio/gpio18/direction

Changing the GPIO value

- Set pin voltage high
  - echo 1 > /sys/class/gpio/gpio18/value
- Set pin for output
  - echo 0 > /sys/class/gpio/gpio18/value

Setting pins at boot

- Edit **/etc/rc.local**
  - Runs at boot time as root
  - Enable pin 18 for output and set high

    echo 18 > /sys/class/gpio/export

    echo out > /sys/class/gpio/gpio18/direction

    echo 1 > /sys/class/gpio18/value
  - Before this runs, values are unpredictable

Setting many pins at boot

- Edit **/etc/rc.local**
  - Set ping 18, 23, 24 and 25 for output and high

```
# Set GPIO ports to out
for n in 18 23 24 25; do
  echo $n > /sys/class/gpio/export
  echo "out" > /sys/class/gpio/gpio$n/direction
  echo 1 > /sys/class/gpio/gpio$n/value
done
```

Turn on pin 5 minutes per hour

- Edit **/etc/crontab**

```
#  This line turns on pin 18 at *:0
0 * * * * root echo 1 > /sys/class/gpio/gpio18/value
#  This line turns off pin 18 at *:5
5 * * * * root echo 0 > /sys/class/gpio/gpio18/value
```

python access to pins

- Import the GPIO package

  import Rpi.GPIO as GPIO

- Name the pins by their GPIO#

  GPIO.setmode(GPIO.BCM)

- Name pins by their board number

  GPIO.setmode(GPIO.BOARD)

python set pins for in/out

- Set pin 18 for output

  GPIO.setup(18,GPIO.OUT)
- Set pins 18,23,24&25 for output

  GPIO.setup([18,23,24,25],GPIO.OUT)
- Set ping 18 for input

  GPIO.setup(18,GPIO.IN)

python set/get pin value

- Set pin 18 high

  GPIO.output(18,1)

- Set pin 18 low

  GPIO.output(18,0)

- Read ping 18 value

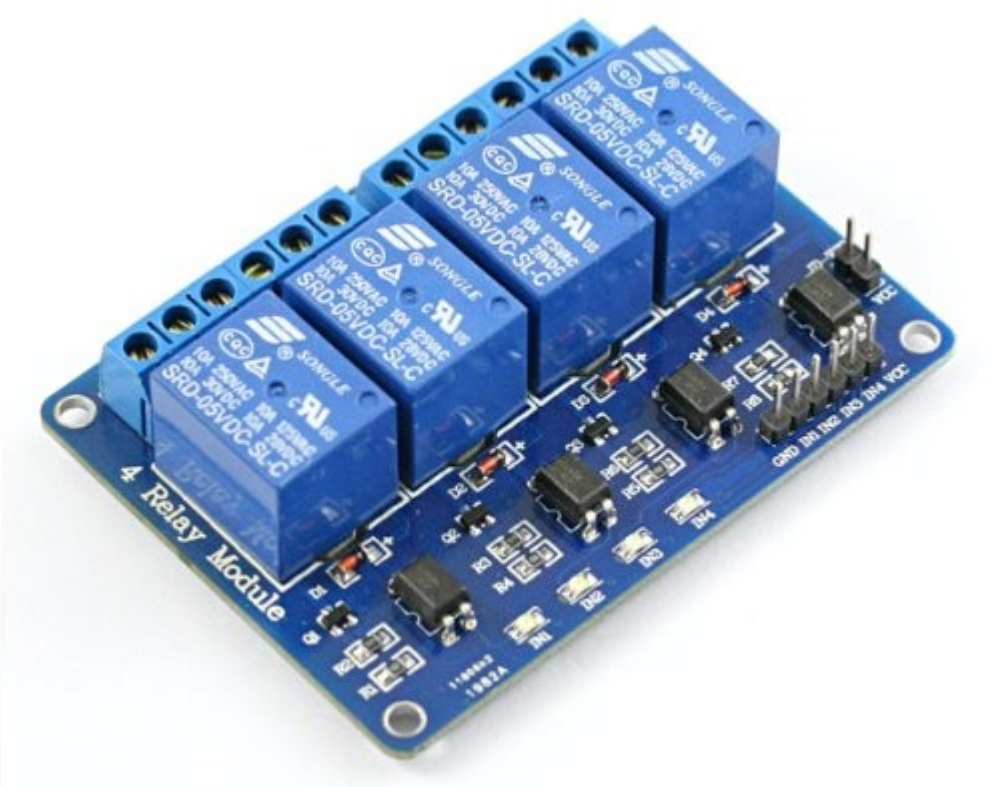
  p18 = GPIO.input(18)

Input pin status

- Set ping 23 to input with pull up
- GPIO.setup(24,GPIO.IN,pull_up_down=GPIO.PUD_UP)
  - ground to activate
- Set pin 24 to input with pull down
- GPIO.setup(24,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
  - Pull up to 3.3V
- A 1k series resistor is typically a good idea

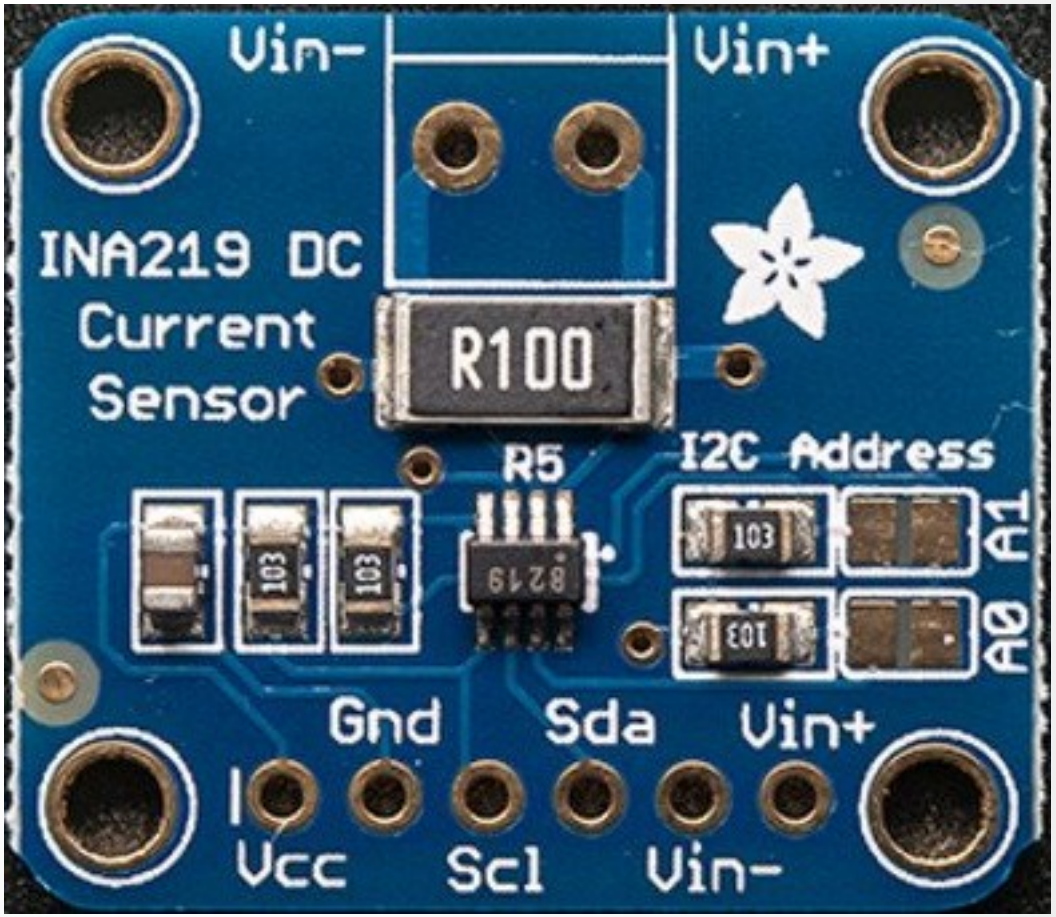
Important Limitations

- GPIO pins are 3.3 V
- Current limited to 16mA
- Opto-isolate relays



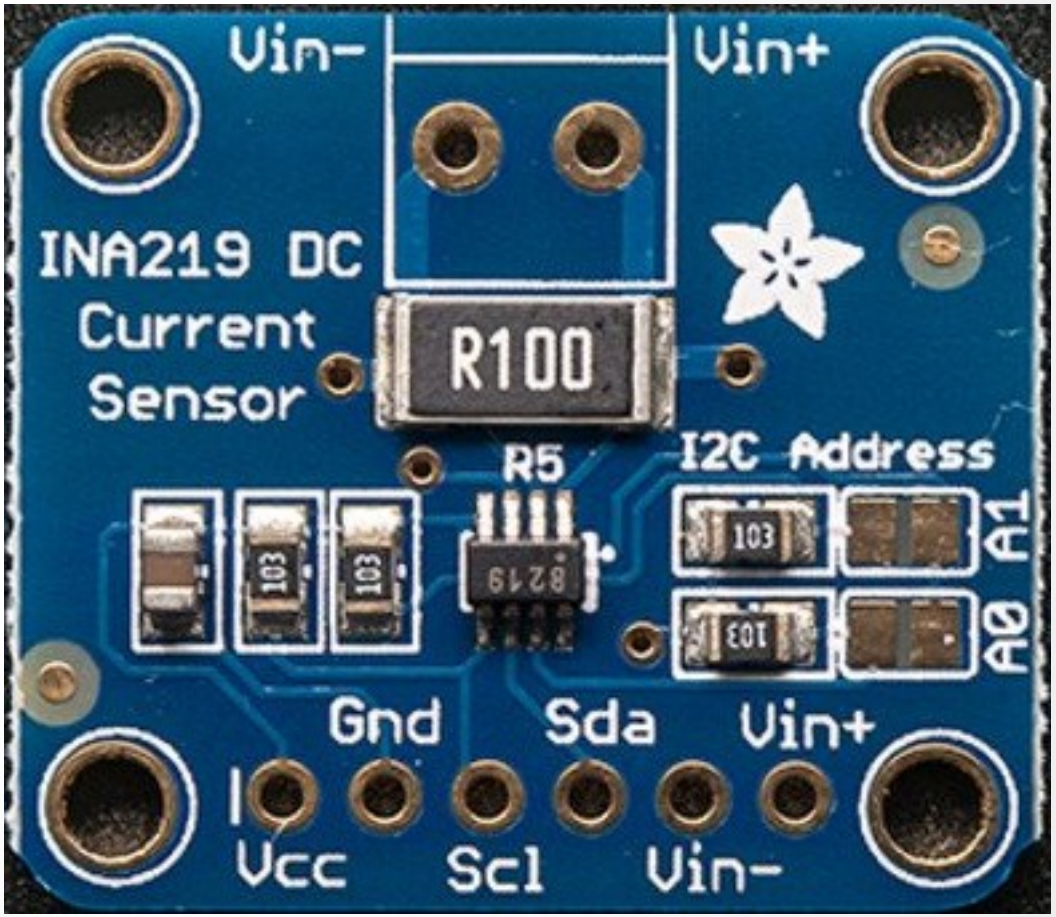
I²C Example: Voltage&Current

- TI INA219 I²C high side monitor
- Max 26V
- Current Sense 40-320mV shunt
- Chip $2.50
- Adafruit $10



Adafruit Breakout

- I²C address 0x40 0x41 0x42 0x43
  - solder jumpers
- 0.1 ohm shunt reads to 3.2A



Python Usage

```
import Subfact_INA219 as INA219
ina = INA219()
V = ina.getBusVoltage_V()
mA =  ina.getCurrent_mA()
```

Digging deeper

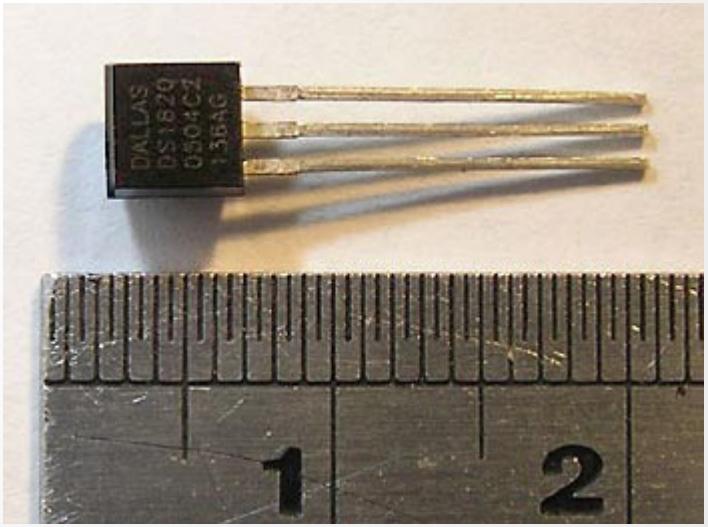
- Subfact_INA219 imports Adafruit_I2C
- Adafruit_I2C imports smbus

- def getBusVoltage():

```
res = i2c.readU16(0x02)
mV =  (res >> 3) * 4
return 0.001*mV
```

Reading 1wire Temperatures

- 1wire uses a single data bus
- Each device has unique address
- DS18S20 is a TO-92 temperature sensor with 0.5C resolution for $2.50
- Can use parasite power (but not on rPi)
  Use 4k7 pullup



Building the Device Tree 1

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

/ {
   compatible = "ti,beaglebone", "ti,beaglebone-black", "ti,beaglebone-green";
   part-number = "BB-W1";
   version = "00A0";

   exclusive-use = "P9.23";

   fragment@0
   {
      target = <&am33xx_pinmux>;
      __overlay__
      {
         bb_w1_pins: pinmux_bb_w1_pins
         {
            pinctrl-single,pins =  <0x68 0x37>; /* gpio1_17,
                     OMAP_PIN_INPUT_PULLUP|OMAP_MUX_MODE7 */
         };
      };
   };
```

Building the Device Tree 2

```
fragment@1
{
   target = <&ocp>;
   __overlay__
   {
      onewire@0
      {
         status        = "okay";
         compatible    = "w1-gpio";
         pinctrl-names = "default";
         pinctrl-0     = <&bb_w1_pins>;

         gpios = <&gpio2 17 0>;
      };
   };
};
};
```

Building the Device Tree 3

- Edit **w1.dts** as shown above
- Compile with device tree compiler

  dtc -O dtb -o w1-00A0.dtbo -b 0 -@ w1.dts

  mv w1-00A0.dtbo /lib/firmware
- Enable

  echo w1 > /sys/devices/bone_capemgr.9/slots

Getting 1wire output

- ls /sys/bus/w1/devices

```
10-000802fba50d
10-000802fbe2f6
10-000802fbf0f9
w1_bus_master1
```

- 10 means it is a DS18S20 temp, the test is a unique serial number

Getting the Data

cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slaves

```
10-000802fbe2f6
10-000802fbf0f9
10-000802fba50d
```

cat /sys/bus/w1/devices/10-000802fbe2f6/w1_slave

```
2c 00 4b 46 ff ff 0e 10 17 : crc=17 YES
2c 00 4b 46 ff ff 0e 10 17 t=21875
```

Temperature of first sensor is 21.875 ºC

Reading Temps in Python 1

```
#  Snarf the slave list  file
fd=open("/sys/bus/w1/devices/w1_bus_master1/w1_master_slaves")
text = fd.read()
fd.close()
#  Split text on line breaks
slaves = filter(None,text.split("\n"))
#  Sort so that order is predictable
slaves.sort()
```

Reading Temps in Python 2

```
#  Blank dictionary
temps = {}
#  Loop over devices
for slave in slaves:
      if slave=="": continue
      #  Snarf device file
      fd = open("/sys/bus/w1/devices/"+slave+"/w1_slave")
      text = fd.read()
      fd.close()
      #  Split lines
      lines = text.split("\n")
      words = lines[1].split(" ")
      #  Get temperature
      C = float(words[9][2:])/1000
      F = 9*C/5+32
     #  Add result to dictionary
      temps[slave] = "%.1fF" % F
```

Observations

- Temperature conversion occurs when you cat the file
  - About 700mS per device
- Temperature reads are best done using a separate thread
- rPi 1wire support in ***raspi-config***

Show and Tell