Sensors board for Raspberry Pi

Overview

Description

This is a small add-on board for the Raspberry Pi, containing six different sensors, an I/O expander chip, eight discrete LEDs and an I/O header. Its purpose is to facilitate experimentation with the sensors and to gain experience of the I^2C bus and the SPI bus without the need for a breadboard and associated wiring.

Sensors

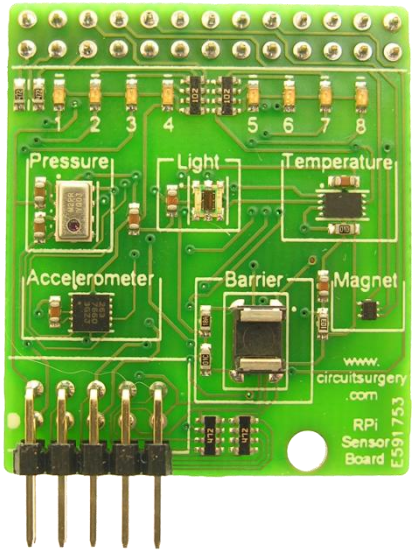
The sensors fitted to the board are as follows (Port expander included for completeness):

Table 1. Sensor connections

Sensor	Device	Connection	Datasheet
Barometric Pressure	MPL115A2	I^2C address 60h	MPL115A2.pdf
Light	APDS-9300	I^2C address 29h GPIO17 (Interrupt)	APDS-9300.pdf
Temperature	LM75B	I^2C address 48h GPIO22 (Overtemp Shutdown)	LM75B.pdf
Accelerometer	MMA7660	I^2C address 4Ch GPIO4 (Interrupt)	MMA7660.pdf
Light Barrier	TCPT1300	GPIO23 (Sender) GPIO24 (Receiver)	TCPT1300.pdf
Hall Effect (Magnetism)	AH1887	GPIO14 (South) GPIO15(North)	AH1887.pdf
Port Expander	MCP23S08	SPI address 20h Device 0, CE0	MCP23S08.pdf

Expansion

The I/O port of the expander chip (fitted to the underside of the board) is connected to the eight LED's situated near the top of the board via a buffer IC. They are also connected directly to the expansion header on the edge of the board to allow the user to connect to the outside world. These are not buffered and should *only* be connected to 3.3volt logic signals.



Header Pin-out (viewed looking towards the board)

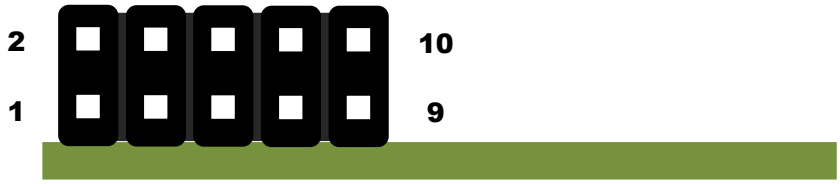


Table 2. Header Pinout

Pin	Function
1	3.3 volts from Raspberry Pi
2	Ground
3	Expansion GPIO 0
4	Expansion GPIO 1
5	Expansion GPIO 2
6	Expansion GPIO 3
7	Expansion GPIO 4
8	Expansion GPIO 5
9	Expansion GPIO 6
10	Expansion GPIO 7

Getting Started

The details that follow make the assumption that the Raspbian Wheezy operating system is installed and that the user has not already installed software and drivers to enable access to the I2C and SPI busses. If you have already done so and are confident with their operation then this section may be skipped.

1. Make sure your operating system is up to date!

Ensure the Pi is connected to the internet and type:

- ***sudo apt-get update***

Let the process complete.

2. Allow access to the I2C and SPI busses

Two files will need modifying. Firstly, the "Modules" file, which specifies the modules to be loaded when the system boots. For the Raspian operating system, run Nano and modify the "Modules" file by entering:

- ***sudo nano /etc/modules***

Add these two lines to the end of the code:

- ***i2c-bcm2708***
- ***i2c-dev***

Type "ctrl+x" and enter "y" to save the file.

Next, modify the ***"/etc/modprobe.d/raspi-blacklist.conf"*** file. This will need to be edited so that the two lines...

- ***blacklist spi-bcm2708***
- ***blacklist i2c-bcm2708***

... will be commented out by adding a "#" in front of them. Again, at the command line prompt, enter:

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

...and then edit the two lines to read:

- ***#blacklist spi-bcm2708***
- ***#blacklist i2c-bcm2708***

Doing this will enable both the I^2C bus and the SPI bus.

3. Install the I2C software tools

Now install the software tools for accessing the I^2C bus.

Type:

- ***sudo apt-get install i2c-tools***

This will install "i2c-tools" software library. Next, add a new user to the I^2C group and restart the system:

- ***sudo adduser pi i2c***
- ***sudo reboot***

When the system has rebooted, you should be able to issue the command...

sudo i2cdetect -y 1 (or ***sudo i2cdetect -y 0*** if you have an early version Raspberry Pi)

... and be presented with a table showing the available I^2C addresses.

In order to be able to access I^2C-tools via Python, the "python-smbus" package is also required. Type:

- ***sudo apt-get install python-smbus***

That completes the basic setup for accessing the I^2C bus.

4. Install the SPI bus software tools

Control of the SPI bus through Python requires the ***python-dev*** package to be installed. Type:

- ***sudo apt-get install python-dev***

Now the python SPI wrapper needed for python to access the port can be installed. Type the following lines:

- ***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***

These lines will:

- Create a new directory called "python-spi"
- Move to the python-spi directory
- Fetch the setup script
- Fetch the SPI device software module
- Install the software device

That completes the setup to allow access to the I2C and SPI busses!

5. Install the sensors board.

Turn your Raspberry Pi off!

Installation of the board is quite easy. Installing the support pillar is not essential but it does add some extra mechanical robustness. Insert the support into the hole of the sensors board. This may be tight, so be firm but take care not to damage the components on the board, especially the light barrier.

Press the connector onto the GPIO port of the Raspberry Pi.

You are now ready to turn the Pi back on and get experimenting!

Demo Software

There are a number of Python scripts that you may download as a Zip file, called sensors-demo.zip, by clicking the link or typing the URL directly: http://www.circuitsurgery.com/sw/sensors-demo.zip. Once downloaded, unpack the files to the directory of your choice (your home directory is a good one!).

These are simple demonstration scripts for each of the sensors. Be aware that they are quite "rough 'n' ready" and could undoubtedly be improved, however they serve to provide a starting point for your own software as well as demonstrating the basic capabilities of the respective sensors.

The SPI Expansion Chip (MCP23008)

Fitted to the underside of the board, this chip is controlled over the SPI bus and is connected to the LEDs and the expansion port. The Python script to demonstrate this is called "sensors-spi-demo.py".

Run the script by typing:-

- ***sudo python sensors-spi-demo.py***

The script will firstly set all ports to output and turn the LEDs off. It will then illuminate each LED in turn, back and forth a couple of times. These conditions will also appear at the expansion header, so you will be able to see the same happen if you connect the expansion header to some LEDs.

The script will now switch the ports to "input" one at a time and the LED's will light as each port is pulled "high" by a 10k ohm resistor. Connecting a port to ground will turn off the corresponding LED and the script will acknowledge the condition by printing the port number connected to ground on the screen. This is an endless loop and will need to be stopped by typing <ctrl+z>.

The Hall Effect (Magnetism) Sensor

This is the simplest of the sensors. The Python script to demonstrate this is called "sensors-hall-demo.py".

Run the script by typing:-

- ***sudo python sensors-hall-demo.py***

Get a magnet and place it close to the "Magnet" area of the Sensors board. The magnetism will be detected by the Hall effect sensor and the polarity will be displayed on the screen and by either the left or right group of four LEDs on the board.

The Infrared Light Barrier

This is again a simple sensor. The Python script for this sensor is "sensors-barrier-demo.py" and, as with the previous script, can be run by typing:

- ***sudo python sensors-barrier-demo.py***

Working with a perforated strip, this script will wait until the strip is placed in the sensor. Gently slide the strip through the gap and the holes will be counted until the script determines that the strip has gone from the sensor. The count will also be shown in binary form on the LEDs at the top of the board.

The Temperature Sensor

Once again, a fairly simple sensor, and again the script "sensors-temp-demo.py" can be run by typing:

- ***sudo python sensors-temp-demo.py***

This script continually prints the temperature on the screen and also presents a coarse representation on the on-board LEDs.

Note that the temperature detected by this sensor will probably be higher than the room temperature. This is normal and is due to the warming effects of the components on the Raspberry Pi.

The Pressure Sensor

This is a slightly more complex sensor in terms of getting a meaningful reading from, although the end result is a simple display of the current atmospheric pressure.

As before, type:

- ***sudo python sensors-press-demo.py***

The script constantly calculates the current atmospheric pressure. Although the output is simple, the script needs to read several registers and do some calculations to arrive at the result. The script is probably more interesting than the output!

The Light Sensor

At the risk of becoming monotonous, the light sensor demo is run by typing:

- ***sudo python sensors-light-demo.py***

The light sensor contains two halves. One "side" senses normal visible light, including the infrared component, and the other senses just the infrared, which may be useful, for example, for detecting the output from an infrared remote control.

When the script is first run, it initialises the sensor and prints confirmation that the sensor has been activated to the screen, after which the light level is displayed on the LEDs as a single illuminated LED moving left to right; the further right, the brighter the light.

The Accelerometer

Once again, the demo for this sensor is run thus:

- ***sudo python sensors-acc-demo.py***

This is probably the most interesting of the six sensors, but also the most complex. With this device you are able to detect movement of the board:

- Tilting and direction of tilt
- Movement and direction of motion
- Shaking
- Tapping

On running, demo script will ask the user to place the Pi on a flat, level surface and tap the board to signify when it has been done. The LEDs are used as a kind of spirit level, starting off in the central position and responding to the tilting of the board. Tilting left or right will cause the LEDs to “slide” left or right. Tilting backwards will make them all light and disappear into the middle, whilst tilting forward will make them reappear. Shaking the Pi will cause the LEDs to light in a pseudo-random fashion and tapping will make them “jitter”.

Detail

The information that follows has, in the main, been taken from the respective device's datasheet, repeated here for ease of reference.

MPL115A2 – Barometric Pressure

The MPL115A2 is an absolute pressure sensor with a digital I2C output with the following features:

- Digitized pressure and temperature information together with programmed calibration coefficients for host micro use.
- Factory Calibrated
- 50 kPa to 115 kPa Absolute Pressure
- ±1 kPa Accuracy
- 2.375V to 5.5V Supply
- Integrated ADC
- I2C Interface (operates up to 400 kHz)
- 7 bit I2C address = 0x60
- Monotonic Pressure and Temperature Data Outputs

Table 3. Device Memory Map

Address	Name	Description	Size (Bits)
0x00	Padc_MSB	10-bit Pressure ADC output value MSB	8
0x01	Padc_LSB	10-bit Pressure ADC output value LSB	2
0x02	Tadc_MSB	10-bit Temperature ADC output value MSB	8
0x03	Tacd_LSB	10-bit Temperature ADC output value LSB	2
0x04	a0_MSB	a0 coefficient MSB	8
0x05	a0_LSB	a0 coefficient LSB	8
0x06	b1_MSB	b1 coefficient MSB	8
0x07	b1_LSB	b1 coefficient LSB	8
0x08	b2_MSB	b2 coefficient MSB	8
0x09	b2_LSB	b2 coefficient LSB	8
0x0A	c12_MSB	c12 coefficient MSB	8
0x0B	c12_LSB	c12 coefficient LSB	8
0x0C	Reserved*	---	---
0x0D	Reserved*	---	---
0x0E	Reserved*	---	---
0x0F	Reserved*	---	---
0x10	Reserved	---	---
0x11	Reserved	---	---
0x12	CONVERT	Start Pressure and Temperature Conversion	---

*These registers are set to 0x00. These are reserved, and were previously utilized as Coefficient values, c11 and c22, which were always 0x00.

I²C Device Read/Write Operations

All device read/write operations are memory mapped. Device actions e.g. "Start Conversions" are controlled by writing to the appropriate memory address location.

For I2C the 7-bit Device Address (from Table 2) has a read/write toggle bit, where the least significant bit is '1' for read operations or '0' for write operations. The Device Address is 0xC0 for a *Write* and the Device Address is 0xC1 for a *Read*.

The most significant bit in the Command tables below is not used and is don't care (X). In examples given it's set to '0'.

Refer to Sensor I2C Setup and FAQ Application Note AN4481 for more information on I2C communication between the sensor and host controller.

Table 4. I²C Write Commands

Command	Binary	HEX(1)
Devices Address + Write bit	1100 0000	0xC0
Start Conversions	X001 0010	0x12

X = Don't care

1 = The command byte needs to be paired with a 0x00 as part of the I2C exchange to complete the passing of Start Conversions.

The actions taken by the part in response to each command are as follows:

Table 5. I2C Write Command Description

Command	Action Taken
Start Conversions	Wake main circuits. Start clock. Allow supply stabilization time. Select pressure sensor input. Apply positive sensor excitation and perform A to D conversion. Select temperature input. Perform A to D conversion. Load the Pressure and Temperature registers with the result. Shut down main circuits and clock.

Table 6. I2C Read Command Description

Command	Binary	Hex(1)
Device Address + Read bit	1100 0001	0xC1
Read Pressure MSB	X000 0000	0x00
Read Pressure LSB	X000 0001	0x01
Read Temperature MSB	X000 0010	0x02
Read Temperature LSB	X000 0011	0x03
Read Coefficient data byte 1	X000 0100	0x04

X = Don't care

1 = The command byte needs to be paired with a 0x00 as part of the I2C exchange to complete the passing of Start Conversions.

APDS-9300 – Ambient Light Photo Sensor

The APDS-9300 is a low-voltage Digital Ambient Light Photo Sensor that converts light intensity to digital signal output capable of direct I2C interface. It has the following features:

- Approximate the human-eye response
- Precise Illuminance measurement under diverse lighting conditions
- Programmable Interrupt Function with User-Defined Upper and Lower Threshold Settings
- 16-Bit Digital Output with I2C Fast-Mode at 400 kHz
- Programmable Analog Gain and Integration Time
- 50/60-Hz Lighting Ripple Rejection
- Low Active Power (0.6 mW Typical) with Power Down Mode

Register Set

The APDS-9300 is controlled and monitored by sixteen registers (three are reserved) and a command register accessed through the serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Table 7.

Table 7. Register Addresses

Address	Register Name	Register Function
--	COMMAND	Specifies register address
0h	CONTROL	Control of basic functions
1h	TIMING	Integration time/gain control
2h	THRESHLOWLOW	Low byte of low interrupt threshold
3h	THRESHLOWHIGH	High byte of low interrupt threshold
4h	THRESHHIGHLOW	Low byte of high interrupt threshold
5h	THRESHHIGHHIGH	High byte of high interrupt threshold
6h	INTERRUPT	Interrupt control
7h	--	Reserved
8h	CRC	Factory test — not a user register
9h	--	Reserved
Ah	ID	Part number/ Rev ID
Bh	--	Reserved
Ch	DATA0LOW	Low byte of ADC channel 0
Dh	DATA0HIGH	High byte of ADC channel 0
Eh	DATA1LOW	Low byte of ADC channel 1
Fh	DATA1HIGH	High byte of ADC channel 1

The mechanics of accessing a specific register depends on the specific I2C protocol used. In general, the COMMAND register is written first to specify the specific control/status register for following read/write operations.

For a full description of the functions of each of the registers, please refer to the datasheet.

LM75B Temperature Sensor

The LM75B is a temperature-to-digital converter which can be configured for different operation conditions. It can be set in normal mode to periodically monitor the ambient temperature, or in shutdown mode to minimize power consumption.

Features include:

- I2C-bus interface with up to 8 devices on the same bus
- Power supply range from 2.8 V to 5.5 V
- Temperatures range from -55 °C to +125 °C
- Frequency range 20 Hz to 400 kHz with bus fault time-out to prevent hanging up the bus
- 11-bit ADC that offers a temperature resolution of 0.125 °C
- Temperature accuracy of:
- ±2 °C from -25 °C to +100 °C
- ±3 °C from -55 °C to +125 °C
- Programmable temperature threshold and hysteresis set points
- Supply current of 1.0 µA in shutdown mode for power conservation
- Stand-alone operation as thermostat at power-up

Register Set

Table 8. Register Table

Register Name	Pointer Value	R/W	POR State	Description
Conf	0x01	R/W	00h	Configuration register: contains a single 8-bit data byte; to set the device operating condition; Default = 0
Temp	0x00	Read Only	n/a	Temperature register: contains two 8-bit data bytes; to store the measured Temp data
Tos	0x03	R/W	0x5000	Overtemperature shutdown threshold register: contains two 8-bit data bytes; to store the overtemperature shutdown Tth(ots) limit; Default = 80 °C
Thyst	0x02	R/W	0x4B00	Hysteresis register: contains two 8-bit data bytes; to store the hysteresis Thys limit; Default = 75 °C

Pointer Register

The Pointer register contains an 8-bit data byte, of which the two LSB bits represent the pointer value of the other four registers, and the other 6 MSB bits are equal to 0, as shown in Table 9 and Table 10. The Pointer register is not accessible to the user, but is used to select the data register for write/read operation by including the pointer data byte in the bus command.

Table 9. Pointer Register

B7	B6	B5	B4	B3	B2	B[1:0]
0	0	0	0	0	0	Pointer Value

Table 10. Pointer Value

B1	B0	Selected Register
0	0	Temperature register (Temp)
0	1	Configuration register (Conf)
1	0	Hysteresis register (Thyst)
1	1	Overtemperature shutdown register (Tos)

Because the Pointer value is latched into the Pointer register when the bus command (which includes the pointer byte) is executed, a read from the LM75B may or may not include the pointer byte in the statement. To read again a register that has been recently read and the pointer has been preset, the pointer byte does not have to be included. To read a register that is different from the one that has been recently read, the pointer byte must be included. However, a write to the LM75B must always include the pointer byte in the statement.

At power-up, the Pointer value is equal to 00 and the Temp register is selected; users can then read the Temp data without specifying the pointer byte.

For a full description of the functions of each of the registers, please refer to the datasheet.

MMA7660 – 3-Axis Orientation/Motion Detection Sensor

The MMA7660FC is a ±1.5 g 3-Axis Accelerometer with Digital Output (I2C) featuring:

- Digital Output (I2C)
- 3mm x 3mm x 0.9mm DFN Package
- Low Power Current Consumption: Off Mode: 0.4 µA,
- Standby Mode: 2 µA, Active Mode: 47 µA at 1 ODR
- Configurable Samples per Second from 1 to 120 samples a second.
- Low Voltage Operation:
- Analog Voltage: 2.4 V - 3.6 V
- Digital Voltage: 1.71 V - 3.6 V
- Auto-Wake/Sleep Feature for Low Power Consumption
- Tilt Orientation Detection for Portrait/Landscape Capability
- Gesture Detection Including Shake Detection and Tap Detection
- Robust Design, High Shocks Survivability (10,000 g)

Register Definitions

Table 11. Register Summary

Address	Name	Definition	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	XOUT	6-bit output value X	-	Alert	XOUT[5]	XOUT[4]	XOUT[3]	XOUT[2]	XOUT[1]	XOUT[0]
0x01	YOUT	6-bit output value Y	-	Alert	YOUT[5]	YOUT[4]	YOUT[3]	YOUT[2]	YOUT[1]	YOUT[0]
0x02	ZOUT	6-bit output value Z	-	Alert	ZOUT[5]	ZOUT[4]	ZOUT[3]	ZOUT[2]	ZOUT[1]	ZOUT[0]
0x03	TILT	Tilt Status	Shake	Alert	Tap	PoLa[2]	PoLa[1]	PoLa[0]	BaFro[1]	BaFro[0]
0x04	SRST	Sampling Rate Status	0	0	0	0	0	0	AWSRS	AMSRS
0x05	SPCNT	Sleep Count	SC[7]	SC[6]	SC[5]	SC[4]	SC[3]	SC[2]	SC[1]	SC[0]
0x06	INTSU	Interrupt Setup	SHINTX	SHINTY	SHINTZ	GINT	ASINT	PDINT	PLINT	FBINT
0x07	MODE	Mode	IAH	IPP	SCPS	ASE	AWE	TON	-	MODE
0x08	SR	Auto-Wake/Sleep and Portrait/Landscape samples per seconds and Debounce Filter	FILT[2]	FILT[1]	FILT[0]	AWSR[1]	AWSR[0]	AMSR[2]	AMSR[1]	AMSR[0]
0x09	PDET	Tap Detection	ZDA	YDA	XDA	PDTH[4]	PDTH[3]	PDTH[2]	PDTH[1]	PDTH[0]
0x0A	PD	Tap Debounce Count	PD[7]	PD[6]	PD[5]	PD[4]	PD[3]	PD[2]	PD[1]	PD[0]
0x0B-0x1F	Factory	Reserved	-	-	-	-	-	-	-	-

NOTE: To write to the registers the MODE bit in the MODE (0x07) register must be set to 0, placing the device in Standby Mode.

For a full description of the functions of each of the registers, please refer to the datasheet.

TCPT1300 – Light Barrier

The TCPT1300X01 is a compact transmissive sensor that includes an infrared emitter and a phototransistor detector, located face-to-face.

This sensor is connected directly to the Raspberry Pi GPIO port. The sender (transmitter) is connected to GPIO 23 (pin 16) and the receiver to GPIO 24 (pin 18).

Setting GPIO23 to logic “1” will switch on the infra red sender, and if the gap between the faces of the sensor is clear, the receiver will pull GPIO24 high (logic “1”). Moving an opaque object into the gap will interrupt the beam and will cause the receiver to switch off, and a logic “0” will be presented to GPIO24.

AH1887 – Hall Effect Switch

This sensor detects the presence of a magnetic field, and also indicates its polarity.

Like the light barrier, it is connected directly to the Raspberry Pi’s GPIO port. Output 1 is connected to GPIO14 (pin 8) and indicates the presence of the South pole of a magnet by pulling the port low (logic “0”). Similarly, Output 2, connected to GPIO15 (pin 10) indicates the presence of a North pole by pulling that pin low.