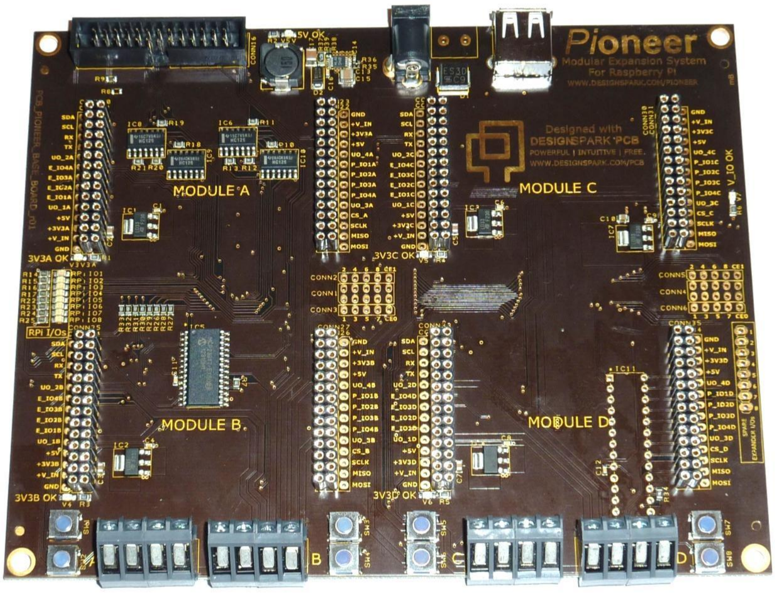
PiGo™

www.DesignSpark.com/PiGo

Modular Expansion System for Raspberry Pi

User manual

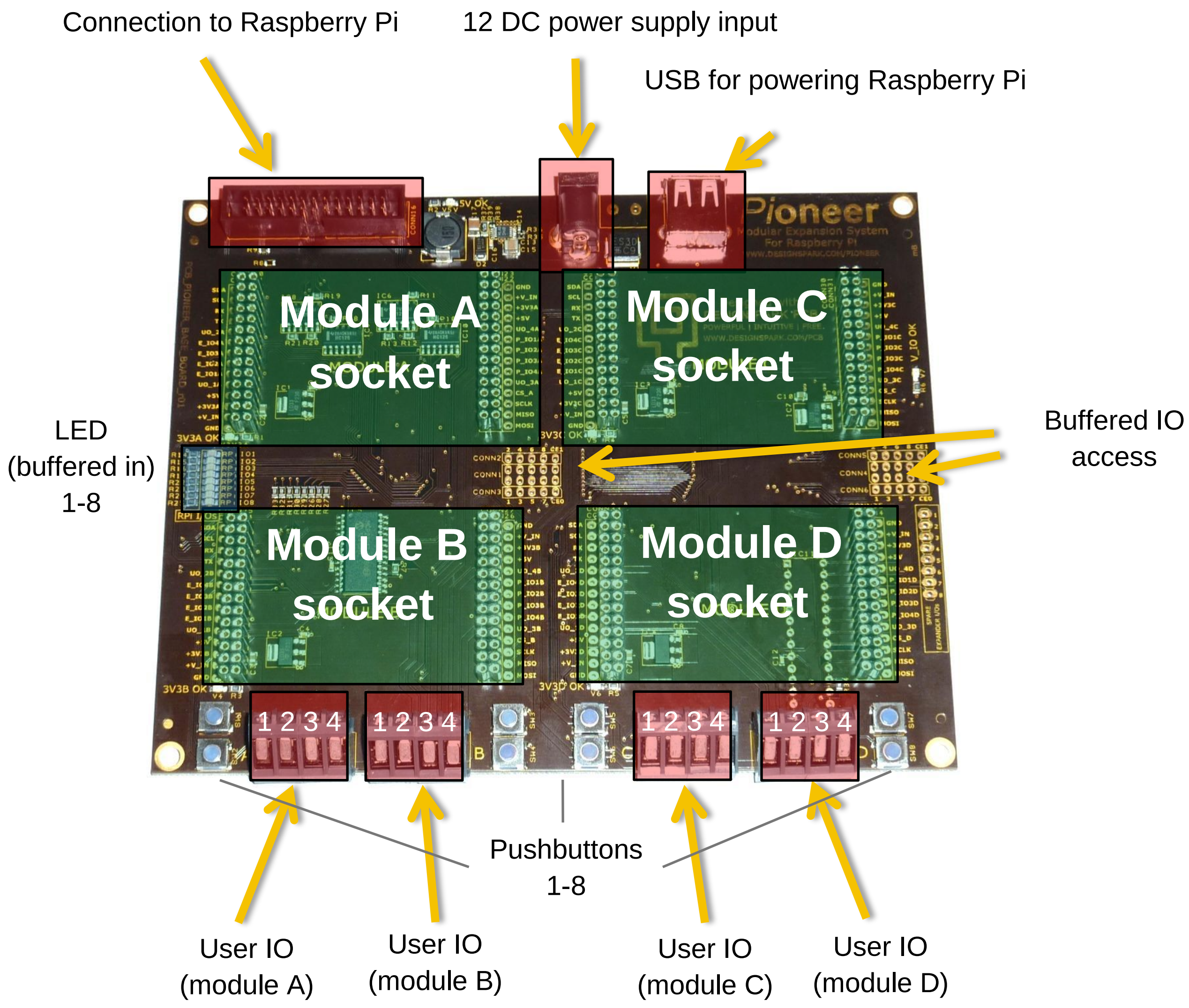


Overview

PiGo extension board is an IO extension board for Raspberry Pi. The board connects to the P1 GPIO connector on the Raspberry Pi board using a flat cable and expands the IO capability of the Raspberry Pi. PiGo extension board features support for multiple module boards, demonstrating the use of PWM for motor control, analog-to-digital and digital-to-analog conversions via SPI bus, general purpose digital IO using I^{2}C bus and serial interface to Arduino or similar device.

Architecture and physical specification

Base board layout



Electrical specifications

The PiGo base board is powered by 12 VDC external power supply via 2.1 mm jack power input connector. There is also an optional 2-way screw terminal that can be soldered if bare wire connection is preferred instead of the jack socket.

The PiGo base board contains 5 VDC switch mode power supply, capable of delivering 2 A of continuous output current. Host USB connector is connected to 5 V power supply directly and can be used to power the Raspberry Pi board directly from the PiGo base board with standard USB cable.

There are four 3.3 V low drop out regulators, one for each module board, capable of continuously delivering 200 mA per each.

The following table shows the current available for module boards

	Using Raspberry Pi model A (1x USB) (consumes max. 500 mA)	**Using Raspberry Pi model B (2x USB + Ethernet) (consumes max. 700 mA)**
Available current at 3.3 V	4 x 0.2 A = **0.8 A**	4 x 0.2 A = **0.8 A**
Available current at 5.0 V	**0.7 A**	**0.5 A**

Raspberry Pi GPIO connector

The following two tables display the basic available functions of pins, found on the Raspberry Pi GPIO connector that is used to connect PiGo board to Raspberry Pi.

Pin Number	**Pin Name Rev1**	**Pin Name Rev2**	**Alt 0 Function**	**Other Alternative Functions**	**PiGo board pin function**
P1-02	5V0	5V0			/
P1-04	5V0	5V0			/
P1-06	GND	GND			GND
P1-08	GPIO 14	GPIO 14	UART0_TXD	ALT5 = UART1_TXD	PiTx (UART)
P1-10	GPIO 15	GPIO 15	UART0_RXD	ALT5 = UART1_RXD	PiRx (UART)
P1-12	GPIO 18	GPIO 18		ALT4 SPI1_CE0_N ALT5 = PWM0	GPIO1
P1-14	GND	GND			GND
P1-16	GPIO23	GPIO23		ALT3 = SD1_CMD ALT4 = ARM_RTCK	GPIO2
P1-18	GPIO24	GPIO24		ALT3 = SD1_DATA0 ALT4 = ARM_TDO	GPIO3
P1-20	GND	GND			GND
P1-22	GPIO25	GPIO25		ALT4 = ARM_TCK	GPIO4
P1-24	GPIO08	GPIO08	SPI0_CE0_N		PiCE0 (SPI)
P1-26	GPIO07	GPIO07	SPI0_CE1_N		PiCE1 (SPI)

Pin Number	Pin Name Rev1	Pin Name Rev2	Alt 0 Function	Other Alternative Functions	PiGo board pin function
P1-01	3.3 V	3.3 V			
P1-03	GPIO 0	**GPIO 2**	I2C0_SDA	I2C0_SDA / I2C1_SDA	PiSDA (I^2C)
P1-05	GPIO 1	**GPIO 3**	I2C0_SCL	I2C0_SCL / I2C1_SCL	PiSCL (I^2C)
P1-07	GPIO 4	GPIO 4		GPCLK0	GPIO5
P1-09	GND	GND			GND
P1-11	GPIO17	GPIO17		ALT3 = UART0_RTS, ALT5 = UART1_RTS	GPIO6
P1-13	GPIO21	**GPIO27**	PCM_DIN	ALT5 = GPCLK1	GPIO7
P1-15	GPIO22	GPIO22		ALT3 = SD1_CLK ALT4 = ARM_TRST	GPIO8
P1-17	3.3 V	3.3 V			/
P1-19	GPIO10	GPIO10	SPI0_MOSI		PiMOSI (SPI)
P1-21	GPIO9	GPIO9	SPI0_MISO		PiMISO (SPI)
P1-23	GPIO11	GPIO11	SPI0_SCLK		PiSCLK (SPI)
P1-25	GND	GND			GND

Module connectors

The PiGo board can accept up to four PiGo peripheral boards, extending the functionality of the base board with a series of attachable modules. Each module is mounted in the space with the appropriate pair of connectors that are found on PiGo board, as seen in the board overview graphics above. Each module receives a

Module A

Left socket			Right socket
Pi$_{SDA}$	I^2C SDA	Power Supply ground	GND
Pi$_{SCL}$	I^2C SCL	PiGo DC input voltage	V$_{IN}$
Pi$_{RX}$	UART Rx	3.3 V power supply (max. 200 mA)	V$_{3.3\ V}$
Pi$_{TX}$	1UART Tx	5 V power supply	V$_{5\ V}$
U$_{IO2A/B/C/D}$	User IO 2	User IO 4	U$_{IO3A/B/C/D}$
EXT10/12/14/16	Ext GPIO	Free IO	P$_{IO1A/B/C/D}$
EXT9/11/13/15	Ext GPIO	Free IO	P$_{IO2A/B/C/D}$
EXT2/4/6/8	Ext GPIO	Free IO	P$_{IO3A/B/C/D}$
EXT1/3/5/7	Ext GPIO	Free IO	P$_{IO4A/B/C/D}$
U$_{IO1A/B/C/D}$	User IO 1	User IO 3	U$_{IO4A/B/C/D}$
V$_{5\ V}$	Save as V$_{5\ V}$ right	SPI Chip Select	CS$_{A/B/C/D}$
V$_{3.3\ V}$	Same as V$_{3.3\ V}$ right	SPI SCLK	Pi$_{SCLK}$
V$_{IN}$	Same as V$_{IN}$ right	SPI MISO	Pi$_{MISO}$
GND	Same as GND right	SPI MOSI	Pi$_{MOSI}$

Library installation

The library installation requires Raspberry Pi to be connected to the internet. Follow the steps below to install the library and demo application.

1. Download the PiGoPackage.zip file and extract it to any folder on the Raspberry Pi
2. Open console (LXTerminal), navigate to the folder, where the above zip file was extracted and type

```
sudo sh install.sh
```

3. The above step will install the library and enable the I2C bus on the Raspberry Pi device. To complete the installation, reboot the device by executing

```
sudo reboot
```

4. Start the demo by executing the following commands in the folder, where the file from step 1 was extracted to

```
sudo sh /Demo/enableRoot.sh
sudo python /Demo/PiGo_demo.py
```

Note: PiGo library accesses the Raspberry Pi's hardware directly and requires root access to run properly. Due to root user not having proper X11 configuration file, the enableRoot.sh script must be executed first to allow the demo application to display a graphical user interface.

Please refer to UART section below to enable UART (disable debug interface).

Available peripherals and buses

GPIO - General Purpose (digital) Input/Output

GPIO pins offer basic digital input/output functionality. The behavior of these pins is controlled by the application that is using the pins – they can be configured to function as digital input or as digital output on request at run time. Raspberry Pi exposes 17 GPIO pins on the GPIO connector, however, only 8 of them are used as digital GPIO pins by the PiGo board, while other are configured with special functions (SPI bus, I^{2}C bus, UART interface). Because the GPIO pins on the Raspberry Pi board are not protected against overvoltages or shorts, PiGo board provides protection and amplification of 8 GPIO pins, which can be accessed as Buffered GPIO pins.

PiGo board also contains a GPIO port extender on the I^{2}C bus

Buffered Raspberry Pi GPIOs

The following schematic shows the PiGo board IO protection, amplification and visualization circuit. All of the 8 buffered GPIO have a dedicated pushbutton switch for usage with digital input function and a LED (light emitting diode) for pin state visualization. The pushbutton switch is connected between the GPIO pin and ground and uses the pull-up resistor of the Raspberry Pi board. When the switch is released and selected pin is configured as digital input, the input reads a digital 1, while pressed switch pulls the voltage level on the pin to digital 0.

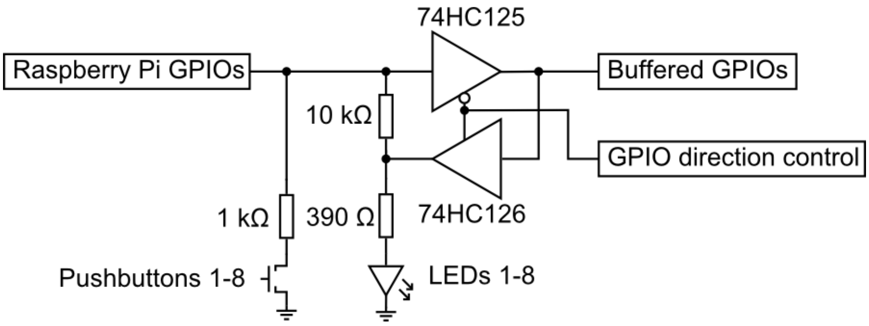


Figure 1: Buffered GPIO schematics

The function of the buffered GPIO circuit is set by the GPIO direction control signal. When this signal is low, the digital output functionality is selected and when the signal is high, the digital input functionality is selected.

Attention: the GPIO direction control signal only controls the buffered GPIO pin function on the PiGo board. To use the buffered GPIO pins successfully, Raspberry Pi's GPIO pin function must be properly configured also. However, Raspberry Pi GPIOs are additionally protected by GPIO function mismatch by a 10 kΩ resistor.

The following table shows the logic states of the buffered GPIO circuit. Bold font marks the active signal source.

Raspberry Pi GPIO state	Pushbutton state	GPIO direction control	Buffered GPIO state
0 (output)	/	0 (output)	0
1 (output)	/	0 (output)	1
0	/	1	**0**
1	/	1	**1**
0	**Pressed**	1	0
1	**Released**	1	1

Buffered GPIO pins are accessible on dedicated connectors CONN1 and CONN4.

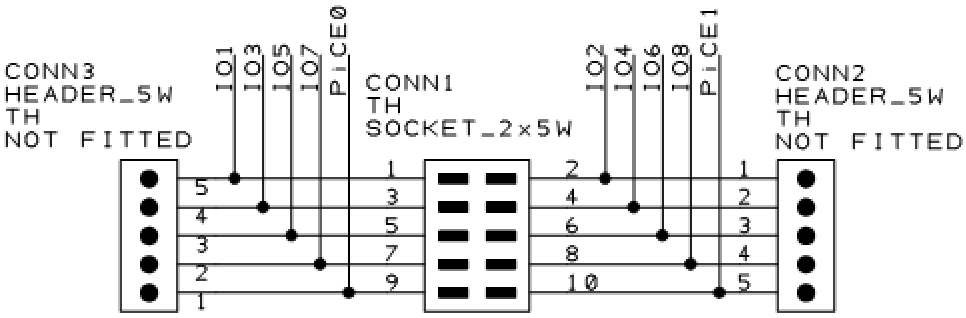


Figure 2: CONN1/2/3 connectors pinout

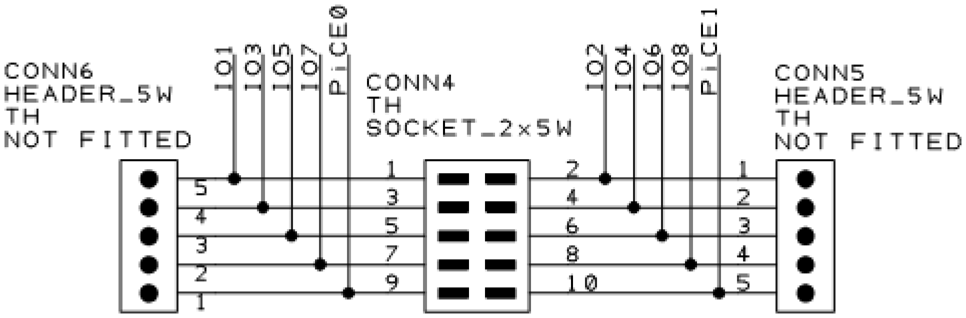


Figure 3: CONN4/5/6 connectors pinout

Using the buffered GPIO with PiGo Python library

Function and description	**setIOdir** - Set buffered IO direction
Arguments	**IONr**: Buffered IO pin (0-7) **IODir**: Buffered IO direction (0 for output, 1 for input) **SetValues**: if 0, no I2C operation is executed (default: 1)
Returns	None
Example	*brd = PiGoBoard()* *brd.setIOdir(0, 0) # Set buffered IO pin 0 as output*

Function and description	**getIOdir** - Return buffered IO direction
Arguments	**IONr**: Buffered IO pin (0-7)
Returns	Direction of the buffered IO pin
Example	*brd = PiGoBoard()* *brd.setIOdir(0, 0) # Set buffered IO pin 0 as output* *…* *brd.getIOdir(0) # Returns the IO pin 0 direction*

Function and description	**setIO** - Set buffered IO state
Arguments	**IONr**: Buffered IO pin (0-7) **IOvalue**: Set buffered digital output state
Returns	None
Example	*brd = PiGoBoard()* *brd.setIOdir(0, 0) # Set buffered IO pin 0 as output* *brd.setIO(0, 1) # Set buffered IO pin 0 to HIGH*

Function and description	**getIO** - Return buffered IO value
Arguments	**IONr**: Buffered IO pin (0-7)
Returns	State of the input pin
Example	*brd = PiGoBoard()* *brd.getIOdir(0, 1) # Set buffered IO pin 0 as input* *brd.getIO(0) # Read the state of the IO pin 0*

Extended GPIOs

PiGo board contains two additional 16-channel GPIO extenders MCP23017 on the I^2C bus. The extended GPIOs are used to drive the digital inputs and outputs of the modules, set the Buffered GPIOs direction input, while additional 8 channels are accessible to user.

Using the buffered GPIO with PiGo Python library

The Python library gives access to Ext1-15 extended GPIOs via the following functions

Function and description	**setExtIOdir** - Set external IO direction
Arguments	**ExtPinNr**: External IO number (0-15) **PinDir**: Pin direction (0 for output, 1 for input) **SetValues**: if 0, no I2C refresh operation is executed (default: 1)
Returns	None
Example	*brd = PiGoBoard()* *brd.setExtIOdir(0, 0) # Set Ext IO pin 0 as output*

Function and description	**getExtIOdir** - Return external IO direction
Arguments	**ExtPinNr**: External IO number (0-15)
Returns	Pin direction
Example	*brd = PiGoBoard()* *brd.getExtIOdir(0, 0) # Returns the Ext IO pin 0 direction*

Function and description	**setExtIO** - Set external IO value
Arguments	**ExtPinNr**: External IO number (0-15) **PinValue**: Output pin state **SetValues**: if 0, no I2C refresh operation is executed (default: 1)
Returns	None

Example	`brd = PiGoBoard()`
	`brd.setExtIOdir(0, 0)# Set Ext IO pin 0 as output`
	`brd.setExtIO(0, 1)    # Set Ext IO pin 0 to HIGH`

Function and description	**getExtIO** - Get external IO value
Arguments	**ExtPinNr**: External IO number (0-15) **ReadValues**: if 0, no I2C refresh operation is executed (default: 1)
Returns	State of the input pin

Example	`brd = PiGoBoard()`
	`brd.getExtIOdir(0, 1)   # Set Ext IO pin 0 as input`
	`brd.getExtIO(0)    # Read the state of the Ext IO pin 0`

UART

UART is an abbreviation for Universal Asynchronous Receiver/Transmitter, most commonly used in conjunction with RS-232 communication standard (also known as serial port). In the world of personal computers, serial ports are almost extinct and replaced by USB ports, however, they still have a major role in embedded systems due to the simplicity and robustness.

The Universal Asynchronous Receiver/Transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires.

There is an UART available in the Broadcom SoC and its RxD and TxD lines are available on the extension connector on pins , but this serial port is configured as debug output port by default. To use this port for general purpose communication from Raspberry Pi's user space, the following steps must be followed:

- Disable kernel boot messages at startup and kernel debugging. Do this by removing the parameters console=ttyAMA0,115200 and kgdboc=ttyAMA0,115200, set in /boot/cmdline.txt. Follow these steps:

  ```
  sudo nano /boot/cmdLine.txt
  ```

  - a simple text editor opens. Use the cursor keys to navigate and delete both parameters console=ttyAMA0,115200, kgdboc=ttyAMA0,115200
  - Press Ctrl+X, then Y to save changes and Enter to confirm the filename.

- Next is to remove the login prompt. Do this by commenting the line T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100 in the file /etc/inittab. Follow these steps:

  ```
  sudo nano /etc/inittab
  ```

  - use cursor keys to navigate to the end of the file, where you will find the line T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
  - Insert a # character at the beginning of the line to comment-out and disable the line
  - Press Ctrl+X, then Y to save changes and Enter to confirm the filename.

- Reboot to activate the changes

  ```
  sudo reboot
  ```

Once all of those functions are disabled, you can use /dev/ttyAMA0 like any normal linux serial port, and you wont get any unwanted traffic confusing the attached devices.

Using UART with PiGo Python library

The Python library gives access to UART via the following functions

Function and description	**serialOpen** - Open serial port using pySerial module
Arguments	**baud**: baudrate (default: 9600) **port**: serial port name (default: '/dev/ttyAMA0') **timeoutValue**: read timeout in seconds (default: 1) **bytesizeValue**: length of the data byte (default: serial.EIGHTBITS) - possible values: serial.FIVEBITS, serial.SIXBITS, serial.SEVENBITS, serial.EIGHTBITS **parityValue**: parity check (default: PARITY_NONE) - possible values: serial.PARITY_NONE, serial.PARITY_EVEN, serial.PARITY_ODD, serial.PARITY_MARK, serial.PARITY_SPACE) **stopbitsValue**: number of stop bits (default: serial.STOPBITS_ONE) - possible values: serial.STOPBITS_ONE, serial.STOPBITS_ONE_POINT_FIVE, serial.STOPBITS_TWO
Returns	None
Example	*brd = PiGoBoard()* *brd.serialOpen(9600, parityValue=serial.PARITY_EVEN)*

Function and description	**serialClose** - Close serial port
Arguments	None
Returns	None

Example	*brd = PiGoBoard()*
	brd.serialOpen(9600, parityValue=serial.PARITY_EVEN)
	…
	brd.serialClose()

Function and description	**serialWrite** - Write to serial port
Arguments	**data**: data to write to serial port
Returns	None
Example	*brd.serialWrite("Pi to the world: 3.141…")*

Function and description	**serialkbhit** - Check serial port input buffer (non-blocking)
Arguments	None
Returns	True if data is in the input buffer
Example	*if brd.serialkbhit():* *data = brd.serialRead(1)*

Function and description	**serialRead** - Read from serial port
Arguments	**length**: number of bytes to read from serial port
Returns	Array of length-bytes from pySerial manual: If a timeout is set it may return less characters as requested. With no timeout it will block until the requested number of

	bytes is read.
Example	*data = brd.serialRead(10)*

Usage examples for UART:

- Simple user interface via terminal
- Connection to a variety of different embedded systems
- GSM communication
- Reading GPS data from GPS receiver

I²C

I²C ("eye-squared cee" or "eye-two-cee" Inter-Integrated Circuit; generically referred to as "two-wire interface") is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, cellphone, or other electronic device.

I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other voltages are permitted. The I²C reference design has a 7-bit or a 10-bit (depending on the device used) address space.

I²C lines SDA and SCL are available on pins P1-03 and P1-05 of the P1 connector on the Raspberry Pi. The software support for I²C is already included in the new Raspbian distributions, however it is deactivated by default. To enable the I2C bus support, comment-out the line *blacklist i2c-bcm2708*:

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

Insert a # character at the beginning of the last line (blacklist i2c-bcm2708) to comment-out and disable the line. Press Ctrl+X, then Y to save changes and Enter to confirm the filename. Next edit the modules file by

```
sudo nano /etc/modules
```

and add ‘i2c-dev’ (without quotes) to a new line. Press Ctrl+X, then Y to save changes and Enter to confirm the filename.

To access the I2C bus from the command line (console), the i2c-tools package is most commonly used. The i2c-tools package contains a heterogeneous set of I^{2}C tools for Linux: a bus probing tool, a chip dumper, register-level SMBus access helpers, EEPROM decoding scripts, EEPROM programming tools and a python module for SMBus access. To install the i2c-tools package, make sure that the Raspberry Pi is connected to the internet and execute the following command

```
sudo apt-get install i2c-tools
```

If you get a 404 error do an update first and run the install again.

```
sudo apt-get update
```

To access the I2C bus without sudo, add a new user to the i2c group:

```
sudo adduser pi i2c
```

Finally, reboot Raspberry Pi to activate the changes

```
sudo reboot
```

After reboot, you can execute the I2C bus scan to see if any devices are connected to the bus.

```
i2cdetect -y 0
```

The following table will be displayed if no devices are found.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Using I^{2}C with PiGo Python library

I^{2}C bus can be accessed via the following three library calls

Function and description	**I2CsetTargetAddress** - Set target I2C device address
Arguments	**addr**: I2C device address (in the range from 0 to 127)
Returns	None

Example	`brd = PiGoBoard()`
	`brd.I2CsetTargetAddress(0x48)`

Function and description	**I2CWrite** - Write to I2C device
Arguments	**data**: data to be written to I2C device
Returns	True if data was acknowledged, False otherwise

Example	`brd = PiGoBoard()`
	`brd.I2CWrite([0])`

Function and description	**I2CRead** - Read from I2C device
Arguments	**numBytes**: number of bytes to be read from I2C device
Returns	Array of data if data was send by the device, empty array otherwise
Example	see below

```
lib.I2CsetTargetAddress(0x48)
if lib.I2CWrite([0]) == True:
  result = lib.I2CRead(2)
  # Second element holds the 0.5 deg. C
  if (result[1] < 0):
    halfDegree = 5
  else:
    halfDegree = 0

  print "LM75: " + str(result[0]) + "." + 
str(halfDegree) + " deg. C"
```

```
else:
  print "LM75 was not found at the specified address!"
```

Usage examples for I²C:

- Accessing a variety of different sensors

SPI

SPI bus is a synchronous serial data link standard that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. The main feature of this bus is that at the same time as the data frame is being sent out by the master bit-by-bit, the response frame is being received. Each write operation therefore equates to one read operation. Multiple slave devices are allowed with either individual slave select (chip select) lines or by daisy-chaining the slave devices.

SPI uses 3 main lines – serial clock (SCLK), master-out-slave-in (MOSI) and master-in-slave-out (MISO). These main SPI lines are available on the Raspberry Pi P1 connector at pins 19, 21 and 23. Raspberry Pi also features two slave select lines (SPI0_CE0_N and SPI0_CE1_N) that can be used to select one of the slave devices that the master (Raspberry Pi) wants to communicate with.

SPI serial bus clock is adjustable by configuring a SPI clock divider. The core clock frequency of 250 MHz is divided by a SPI clock divider to produce the clock to drive the SCLK pin. The divider can be set to any even number from 2 to 65536. This means that SPI frequencies from 3.814 kHz to 125 MHz are supported.

Using SPI with PiGo Python library

SPI bus can be accessed via the following library calls

Function and description	**SPIinit** - Initialize SPI interface
Arguments	**SPIdivider**: a constant that is used to divide the original 250 MHz clock for SPI, default value is 1024, giving 244 kHz SPI clock
Returns	None

Example	`brd = PiGoBoard()`
	`brd.SPIinit(250 # Initialize SPI bus with 1 MHz clock`

Function and description	**SPIsetCS** - Set chip select and polarity of it
Arguments	**CS**: chip select constant (BCM2835_SPI_CS0 for the primary chip select, BCM2835_SPI_CS1 for the secondary chip select) **polarity**: polarity of the chip select signal - whether the chip select pin is to be active HIGH
Returns	None

Example	`brd = PiGoBoard()`
	`brd.SPIinit(250) # Initialize SPI bus with 1 MHz clock`
	`brd.SPIsetCS(PiGoBoardData.BCM2835_SPI_CS0, 0)`

Function and description	**SPItransfer** - SPI transfer data
Arguments	**data**: array of integers (8-bit) to be written to target SPI device
Returns	an array of integers, returned by the SPI device, with the same size as provided in data

Example	`brd = PiGoBoard()`
	`brd.SPIinit(250) # Initialize SPI bus with 1 MHz clock`
	`read = brd.SPItransfer([0xAA, 0x55])# Transfer 2 bytes`

Function and description	**SPIread** - SPI read data
Arguments	**numBytes**: number of bytes to be read from SPI device
Returns	an array of integers, returned by the SPI device, with the same size of numBytes

Example	`brd = PiGoBoard()`
	`brd.SPIinit(250) # Initialize SPI bus with 1 MHz clock`
	`read = brd.SPIread(2) # Read 2 bytes`

Remark	Due to SPI bus protocol this command must write the same amount of date to the bus in order to receive the data.

Usage examples for SPI:

- Feeding data to shift registers (serial to parallel conversion)
- Read data from shift registers (parallel to serial conversion)
- Fast bus connection for data exchange
- Access to a variety of different sensors

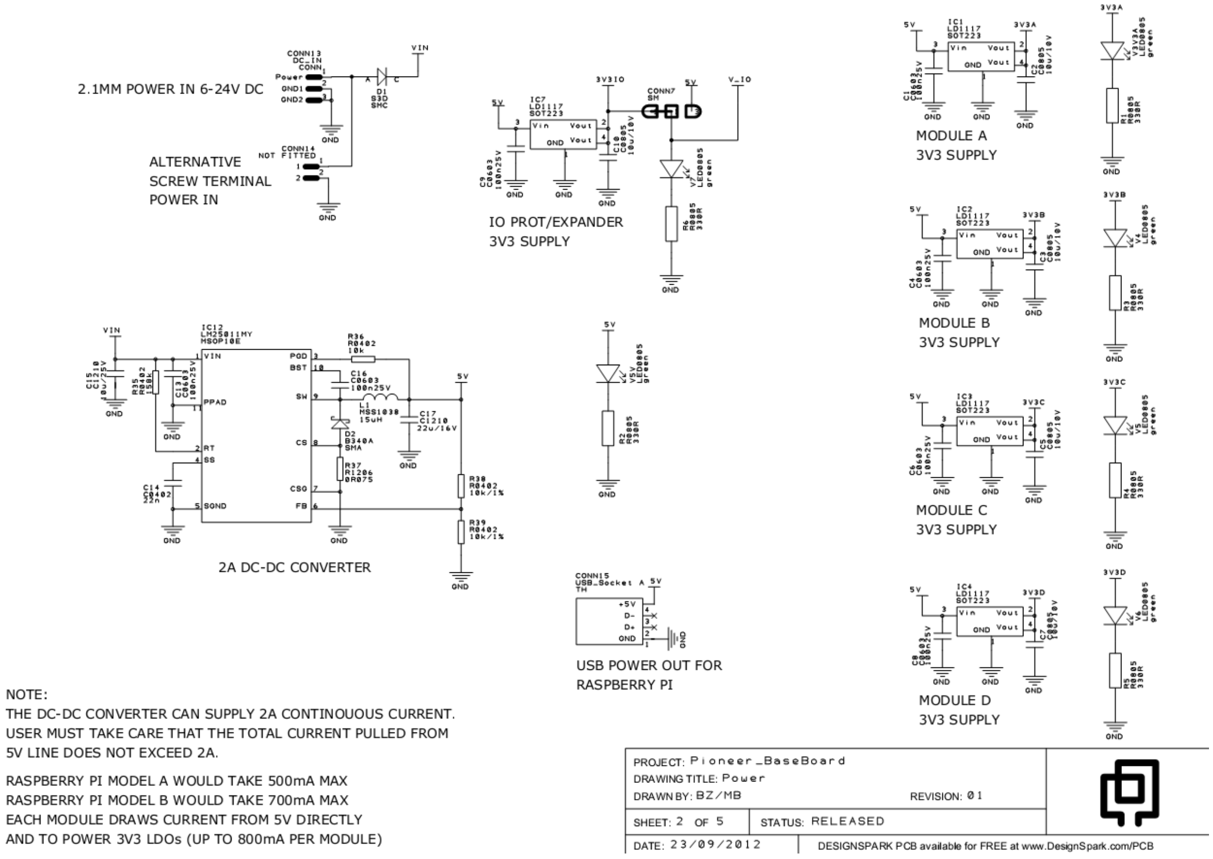
Frequently asked questions

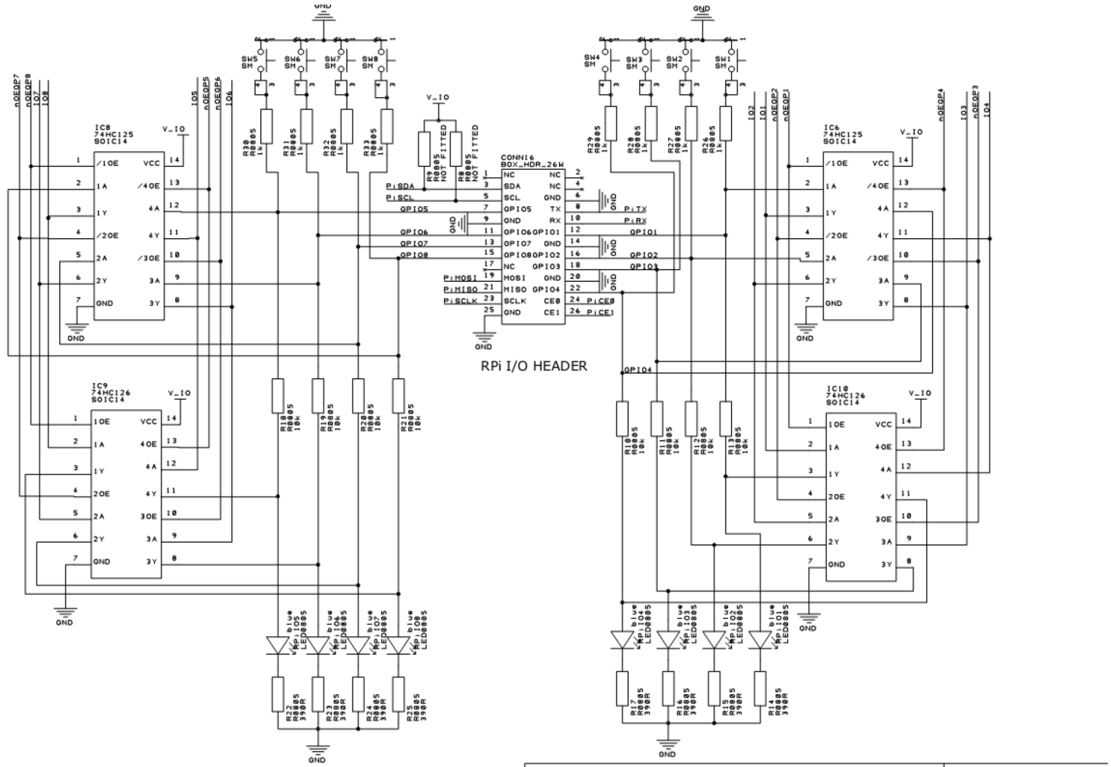
When I start the python script with sudo, the 'Couldn't connect to display…' error appears.

Most probably, no X11 authorization settings are present for the root user. To solve the problem, copy the settings of user pi to user root by executing the following line in the command line:

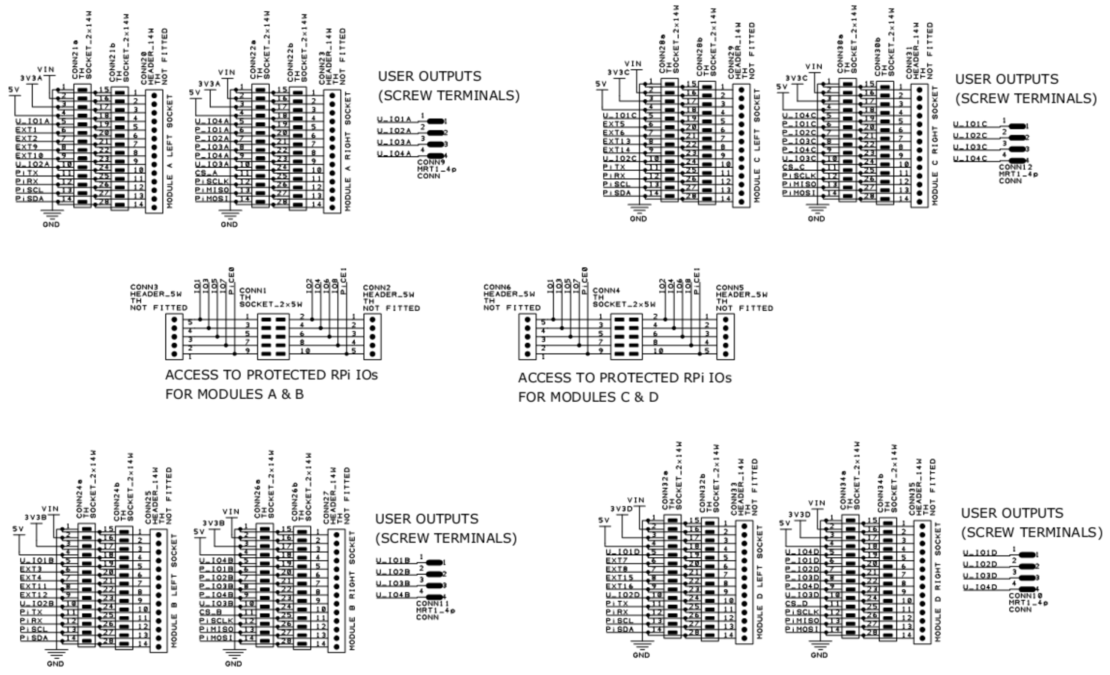
```
sudo cp ~pi/.Xauthority ~root/
```

Board schematics





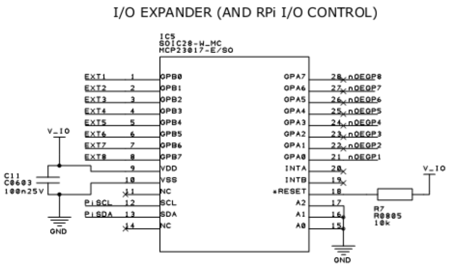
PROJECT: Pioneer_BaseBoard
DRAWING TITLE: Pi IO Protection
DRAWN BY: BZ/MB REVISION: 01
SHEET: 3 OF 5 STATUS: RELEASED
DATE: 23/09/2012 DESIGNSPARK PCB available for FREE at www.DesignSpark.com/PCB

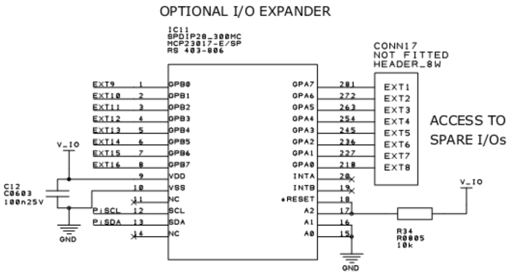


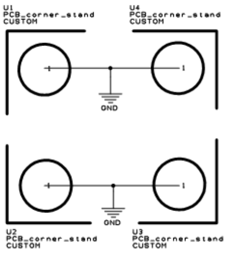
NOTE:
EACH MODULE IS EQUIPPED WITH 2x14 WAY SOCKETS. WHILE MODULE SITS IN THE INNER ROWS, THE OUTER ROWS ARE LINKED IN PARALLEL PROVIDING EASY PLUG-IN ACCESS TO ALL MODULE PINS.

THERE ARE ADDITIONAL PARALLEL PADS PROVIDED FOR SOLDERING CONNECTIONS TO ALL MODULE PINS

PROJECT: Pioneer_BaseBoard
DRAWING TITLE: Module sockets
DRAWN BY: BZ REVISION: 01
SHEET: 4 OF 5 STATUS: RELEASED
DATE: 23/09/2012 DESIGNSPARK PCB available for FREE at www.DesignSpark.com/PCB







PROJECT: Pioneer_BaseBoard	
DRAWING TITLE: IO Expanders	
DRAWN BY: BZ	REVISION: 01
SHEET: 5 OF 5	STATUS: RELEASED
DATE: 23/09/2012	DESIGNSPARK PCB available for FREE at www.DesignSpark.com/PCB

Sources

- http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
- http://elinux.org/RPi_Serial_Connection
- http://en.wikipedia.org/wiki/I%C2%B2C
- http://www.skpang.co.uk/blog/archives/575