Causing Hardware Action Directly from the Pi's GPIO Pins

Caution: The best practice for working with the Raspberry Pi and low-current peripheral boards is to wear a ground strap on your wrist and not work in rooms with synthetic rugs. I don't wear a strap and have not knowingly blown any circuits, but I work in a room with a old natural rug. In very low humidity of freezing weather the danger is greater.

The diagram at http://yosemitefoothills.com/ACM_Workshop_Notes/GPIOPinGuide.png shows the 40 GPIO (General-Purpose Input/Output) pins of the Raspberry Pi A+, Raspberry Pi B+, and Raspberry Pi 2 B. Most are directly connected to the processor and there is no recovery if their maximum electrical specifications are exceeded.

Specifically,

The power pins 1, 2, 4, and 17 must never be grounded or connected to another power source.

The ground pins must not be directly connected to power source although they may be used to “sink” a few mA (milliamperes) or maybe even a few hundreds of mA.

All the remaining pins should never be allowed to have more than 2 mA flow into or out of them. You need to know Ohm's Law and the properties of what you are planning to connect to the pins.

The GPIO pins are connected directly to corresponding pins on the processor. Many of them serve multiple purposes as are summarized in the diagram.

It is strongly recommended that you turn off your Raspberry Pi when you connect circuits to the pins. Only after **carefully checking** your connections, should you then turn it on to try it out.

Add-on boards often will have buffers between the GPIO pins and external components. Those buffers provide some measure of protection. Buffers also usually provide greater current output and current sinking capability.

When set to input signals, the GPIO pins are designed to sense either a digital “hi” (“1”) level of more than 1.3 V and a “low” (“0”) level of less than 0.8 V. The input current drawn by the pins on input is very low, in the µA (microampere) range. This high sensitivity to small currents makes unconnected input pins respond to the static electricity of moving hands near leads.

When set to output signals, the GPIO pins easily provide about 2 mA of current while holding a digital “hi” level at near 3.3 V and pulling down a digital “low” level to near 0 V. An ordinary 2 mA LED with a 1000 ohm resistor in series works nicely. It draws 1.5 mA when the LED is lit. The longer lead of the LED is the positive lead. Connecting the LEDs without the resistor in series will almost certainly permanently destroy the LED and might damage the Pi's GPIO pins!

The total current drawn from the two 3.3 V power pins should not be more than about 50 mA.
The total current drawn from the 5 V power pins should not be more than 300 mA.

More detailed and precise information is available at

http://elinux.org/RPi_Low-level_peripherals#General_Purpose_Input.2FOutput_.28GPIO.29

and engineering specifications are here:

http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications

The programs and many others mentioned below will be provided in a 761 MB file that will be passed out at the workshop using a USB drive. It is too big for the slow uploading speed of my web site.

Blinking LEDs – The "Hello, World!" of Hardware Programming

My blinking LED demo (programs `pinToggle.py` and `pinToggle2.py`) are connected as shown at the bottom of the diagram. Each LED has a 1000 ohm resistor in series with the long lead of the LED connected to a pin (green to GPIO 17, yellow to GPIO 27 and red to GPIO 22). The free ends of the resistors are all connected to the ground. A push button is connected between GPIO 10 and ground. In the program GPIO 17, 27, and 22 are set to output, and GPIO 10 is set to input. The programs are run by doing

```
sudo ./pinToggle.py
```

and

```
sudo ./pinToggle2.py
```

from within the `/home/pi/Programming/Python/NoGraphics/GPIO/` directory.

The first program simply turns the LEDs on for 5 seconds and then turns them off.

The second program causes them to blink together unless the button is pressed in which case they blink sequentially. Pressing <`Ctrl-C`> is needed to stop the program.

Using an Ultrasonic Sensor

The common type of inexpensive ultrasonic sensor is designed for 5 V use. It can be used with a Raspberry Pi as long as its echo output voltage is lowered to 3.3 V max by a resistive voltage divider. Such a divider is constructed by connecting one end of a 1500 ohm resistor to ground and one end of a 1000 ohm resistor to the "Echo" output terminal. The free ends of the two resistors are then connected together and connected to the wire going to the Pi terminal slated for the echo input.

Other than that complication, the "VCC" terminal of the sensor is connected to one of the two 5 V pins on the Pi GPIO header, the "GND" terminal of the sensor is connected to one of the Pi's ground terminals, and the "Trig" terminal of the sensor is connected to a GPIO pin slated for output. Even though the Pi output to the "Trig" input of the sensor is only 3.3 V (instead of the nominal 5V for the sensor), it is sufficient to make it trigger.

The connections I used are shown in the diagram at the lower-right side. I made a correction to my original diagram which stated that the divider needed to be on the trigger line, but the correct place is on the echo output as shown in the diagram on the web site link given above.

The Python program `ultrasonic.py` makes it work and is run by doing:

```
sudo ./ultrasonic.py
```

from within the `/home/pi/Programming/Python/NoGraphics/GPIO/` directory.

The program makes the trigger pin go "Hi" for 10 μs (microseconds), then measures the amount of time that the echo pin goes to a "Hi" level. From that time interval, it calculates a distance. It then repeats that measurement sequence 1000 times until it stops and cleans up its pin settings.