The Tutorial Book

Have fun with PIC microcontrollers, Jal v2 and Jallib

Sébastien Lelong

2009

Jallib Group

Contents

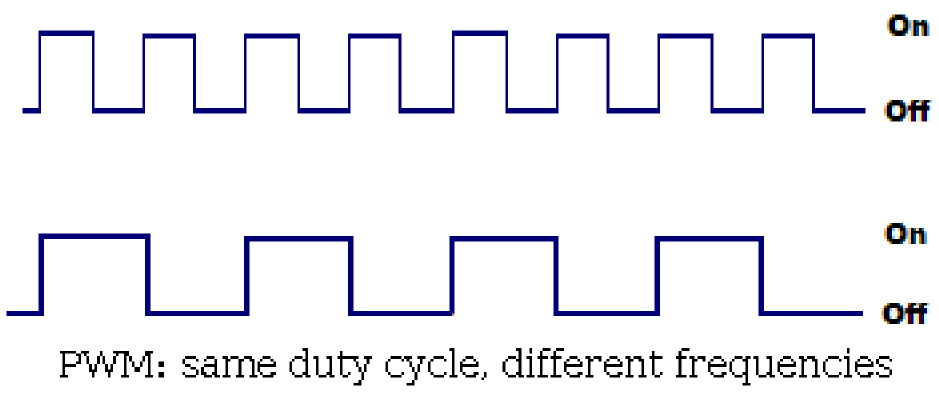
Having fun with PWM and a LED (part 1)

In this "Step-by-Step" tutorial, we're going to (try to) have some fun with PWM. PWM stands for *Pulse Width Modulation*, and is quite weird when you first face this (this was at least my first feeling).

So, how does PWM look like ?...

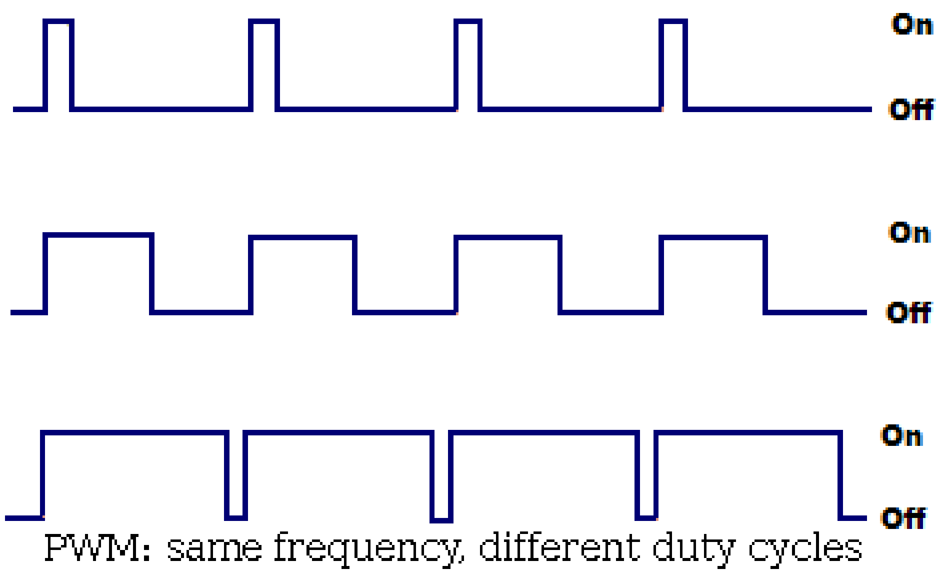
PWM is about switching one pin (or more) high and low, at different frequencies and duty cycles. This is a on/off process. You can either vary:

- the **frequency**,
- or the **duty cycle**, that is the proportion where the pin will be high



Both have a 50% duty cycle (50% on, 50% off), but the upper one's frequency is twice the bottom

Figure 1: PWM: same duty cycle, different frequencies.



Three different duty cycle (10%, 50% and 90%), all at the same frequency

Figure 2: PWM: same frequency, different duty cycles

But what is PWM for ? What can we do with it ? Many things, like:

- producing variable voltage (to control DC motor speed, for instance)
- playing sounds: duty cycle is constant, frequency is variable

- playing PCM wave file (PCM is Pulse Code Modulation)
- ...

One PWM channel + one LED = fun

For now, and for this first part, we're going to see how to *control the brightness of a LED*. If simply connected to a pin, it will light at its max brightness, because the pin is "just" high (5V).

Now, if we connect this LED on a PWM pin, maybe we'll be able to control the brightness: as previously said, *PWM can be used to produce variable voltages*. If we provide half the value (2.5V), maybe the LED will be half its brightness (though I guess the relation between voltage and brightness is not linear...). Half the value of 5V. How to do this ? Simply **configure the duty cycle to be 50% high, 50% low**.

But we also said *PWM is just about switching a pin on/off*. That is, either the pin will be 0V, or 5V. So how will we be able to produce 2.5V ? Technically speaking, we won't be able to produce a real 2.5V, but if PWM frequency is high enough, then, on the average, and from the LED's context, it's as though the pin outputs 2.5V.

Building the whole

Enough theory, let's get our hands dirty. Connecting a LED to a PWM pin on a 16f88 is quite easy. This PIC has quite a nice feature about PWM, it's possible to select which pin, between RB0 and RB3, will carry the PWM signals. Since I use *tinybootloader* to upload my programs, and since tiny's fuses are configured to select the RB0 pin, I'll keep using this one (if you wonder why tinybootloader interferes here, *read this post*).

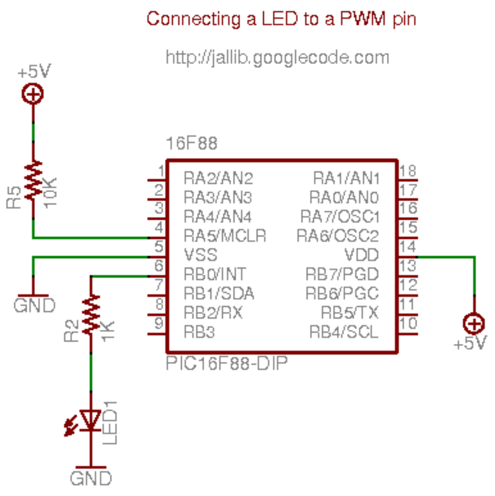
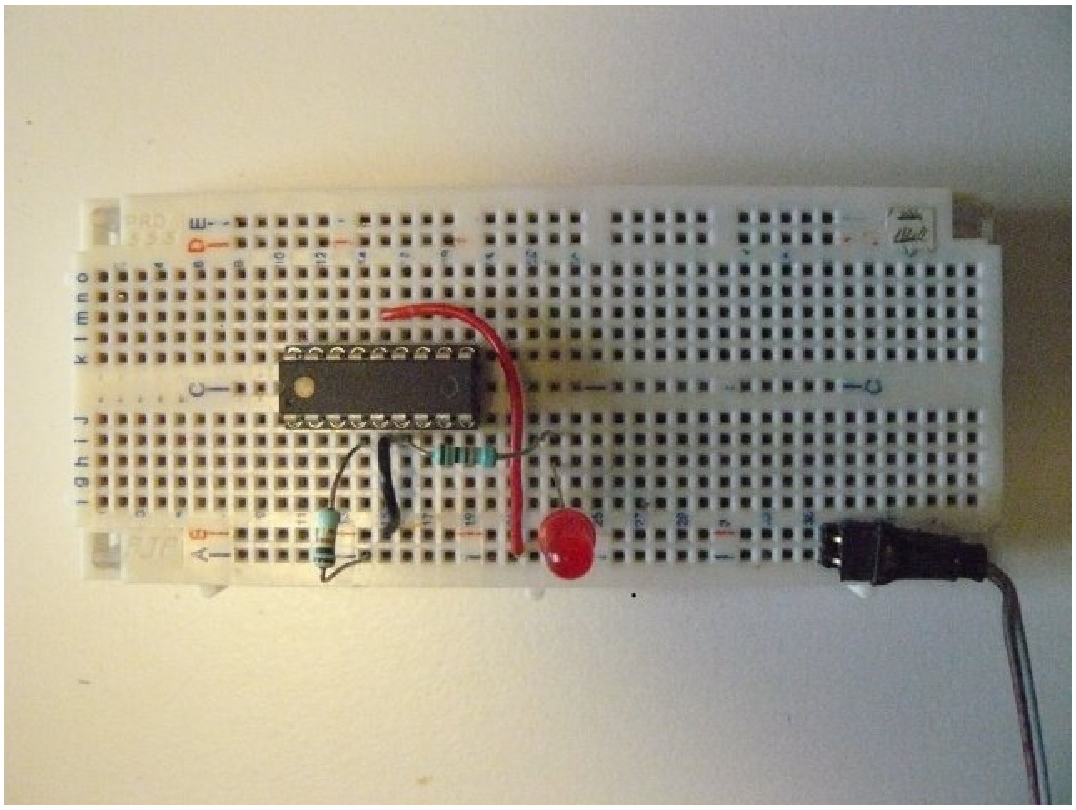
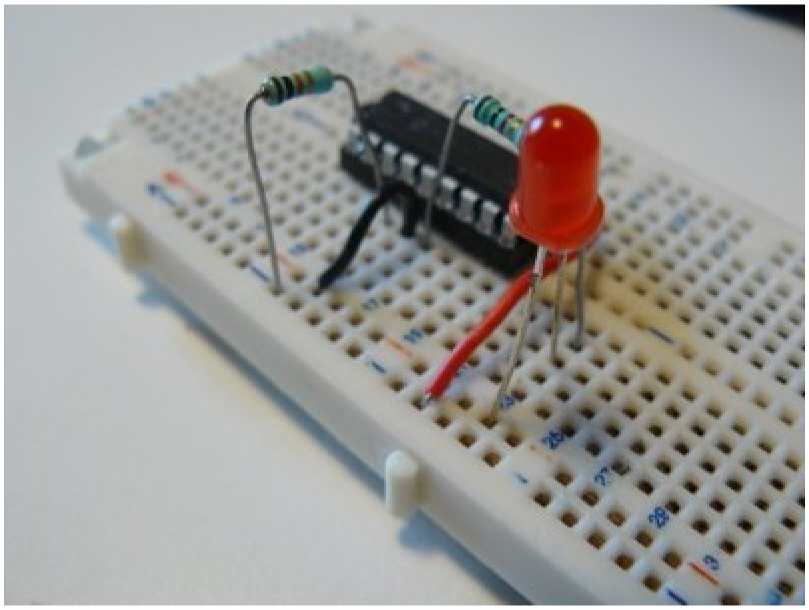


Figure 3: Connecting a LED to a PWM pin

On a breadboard, this looks like this:



The connector brings +5V on the two bottom lines (+5V on line A, ground on line B).



LED is connected to RB0

Writing the software

For this example, I took one of the 16f88's sample included in jallib distribution (*16f88_pwm_led.jal*), and just adapt it so it runs at 8MHz, using internal clock. It also select RB0 as the PWM pin.

So, step by step... First, as we said, we must select which pin will carry the PWM signals...

```
pragma target CCP1MUX      RB0     -- ccp1 pin on B0
```

and configure it as output

```
var volatile bit pin_ccp1_direction is pin_b0_direction
pin_ccp1_direction = output
-- (simply "pin_b0_direction = output" would do the trick too)
```

Then we include the PWM library.

```
include pwm_hardware
```

Few words here... This library is able to handle **up to 10 PWM channels** (PIC using CCP1, CCP2, CCP3, CCP4, ... CCP10 registers). Using conditional compilation, it **automatically selects the appropriate underlying PWM libraries**, for the selected target PIC.

Since 16f88 has only one PWM channel, it just includes "pwm_ccp1" library. If we'd used a 16f877, which has two PWM channels, it would include "pwm_ccp1" *and* "pwm_ccp2" libraries. What is important is it's transparent to users (you).

OK, let's continue. We now need to configure the **resolution**. What's the resolution ? Given a frequency, the **number of values you can have for the duty cycle** can vary (you could have, say, 100 different values at one frequency, and 255 at another frequency). Have a look at the datasheet for more.

What we want here is to have the **max number of values we can for the duty cycle**, so we can select the exact brightness we want. We also want to **have the max frequency** we can have (ie. no pre-scaler).

```
pwm_max_resolution(1)
```

If you read the *jalapi documentation* for this, you'll see that the frequency will be 7.81kHz (we run at 8MHz).

PWM channels can be turned on/off independently, now we want to activate our channel:

```
pwm1_on()
```

Before we dive into the forever loop, I forgot to mention PWM can be used in **low or high resolution**. On *low resolution*, duty cycles values range from *0 to 255* (8 bits). On *high resolution*, values range from *0 to 1024* (10 bits). In this example, we'll use low resolution PWM. For high resolution, you can have a look at the other sample, *16f88_pwm_led_highres.jal*. As you'll see, there are very few differences.

Now let's dive into the loop...

```
forever loop
   var byte i
   i = 0
   -- loop up and down, to produce different duty cycle
   while i < 250 loop
      pwm1_set_dutycycle(i)
      _usec_delay(10000)
      i = i + 1
   end loop
   while i > 0 loop
      pwm1_set_dutycycle(i)
      _usec_delay(10000)
      i = i - 1
   end loop
   -- turning off, the LED lights at max.
   _usec_delay(500000)
   pwm1_off()
   _usec_delay(500000)
   pwm1_on()

end loop
```

Quite easy right ? There are *two main waves*: one will light up the LED progressively (0 to 250), another will turn it off progressively (250 to 0). On each value, we set the duty cycle with `pwm1_set_dutycycle(i)` and wait a little so we, humans, can see the result.

About the result, how does this look like ? See this video: *http://www.youtube.com/watch?v=r9_TfEmUSf0*

"I wanna try, where are the files ?"

To run this sample, you'll need the *last jallib pack* (at least 0.2 version). You'll also find the exact code we used *here*.

Sébastien Lelong

Interfacing a Sharp GP2D02 IR ranger

Sharp IR rangers are widely used out there. There are many different references, depending on the beam pattern, the minimal and maximal distance you want to be able to get, etc... The way you got results also make a difference: either **analog** (you'll get a voltage proportional to the distance), or **digital** (you'll directly get a digital value). This nice article will explain these details (and now I know GP2D02 seems to be discontinued...)

Overview of GP2D02 IR ranger

GP2D02 IR ranger is able to measure distances between approx. 10cm and 1m. Results are available as digital values you can access through a dedicated protocol. One pin, Vin, will be used to act on the ranger. Another pin, Vout, will be read to determine the distance. Basically, getting a distance involves the following:

1. First you wake up the ranger and tell it to perform a distance measure
2. Then, for each bit, you read Vout in order to reconstitute the whole byte, that is, the distance
3. finally, you switch off the ranger

The following timing chart taken from the datasheet will explain this better.

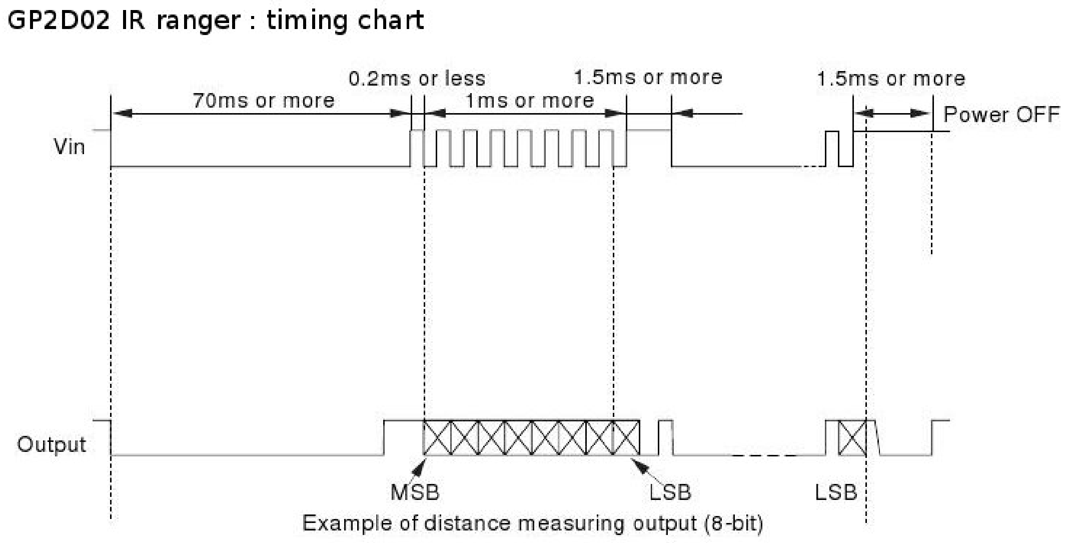
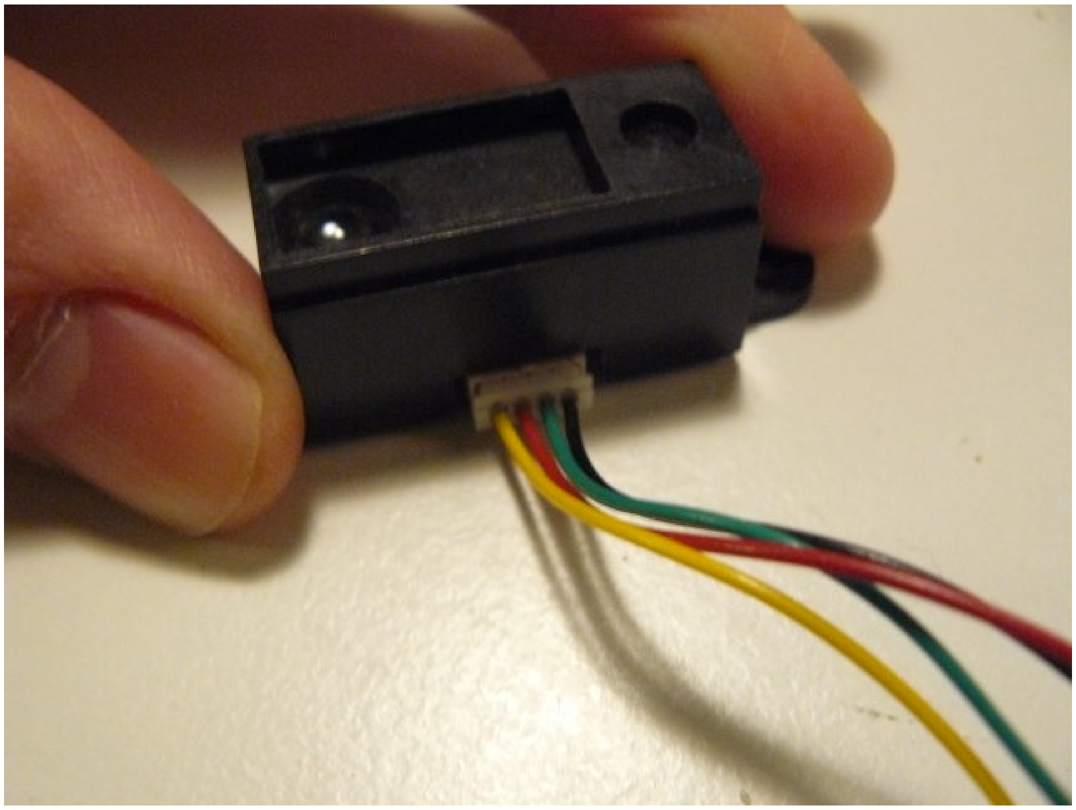


Figure 4: GD2D02 IR ranger : timing chart

Note: the distances obtained from the ranger aren't linear, you'll need some computation to make them so.

Sharp GP2D02 IR ranger looks like this:



- *Red* wire is for +5V
- *Black* wire ground
- *Green* wire is for Vin pin, used to control the sensor
- *Yellow* wire is for Vout pin, from which 8-bits results read

(make a mental note of this...)

Interfacing the Sharp GP2D02 IR ranger

Interfacing such a sensor is quite straight forward. The only critical point is **Vin** ranger pin can't handle high logic level of the PIC's output, *this level mustn't exceed 3.3 volts*. A **zener diode** can be used to limit this level.

Note: be careful while connecting this diode. Don't forget it, and don't put it in the wrong side. You may damage your sensor. And I'm not responsible for ! You've been warned... That's said, I already forgot it, put it in the wrong side, and thought I'd killed my GP2D02, but this one always got back to life. Anyway, be cautious !

Here's the whole schematic. The goal here is to collect data from the sensor, and light up a LED, more or less according to the read distance. That's why we'll use a LED driven by PWM.

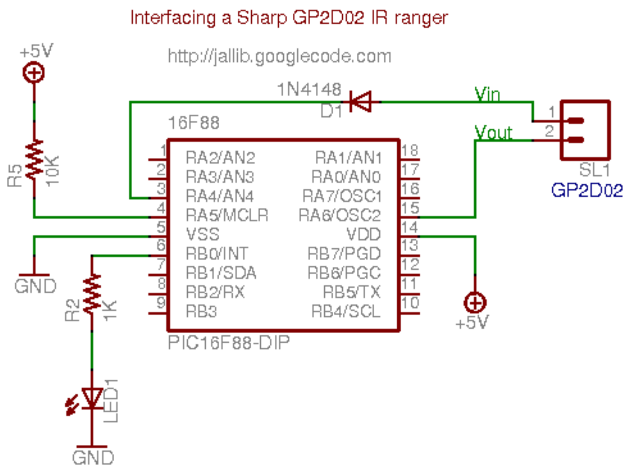
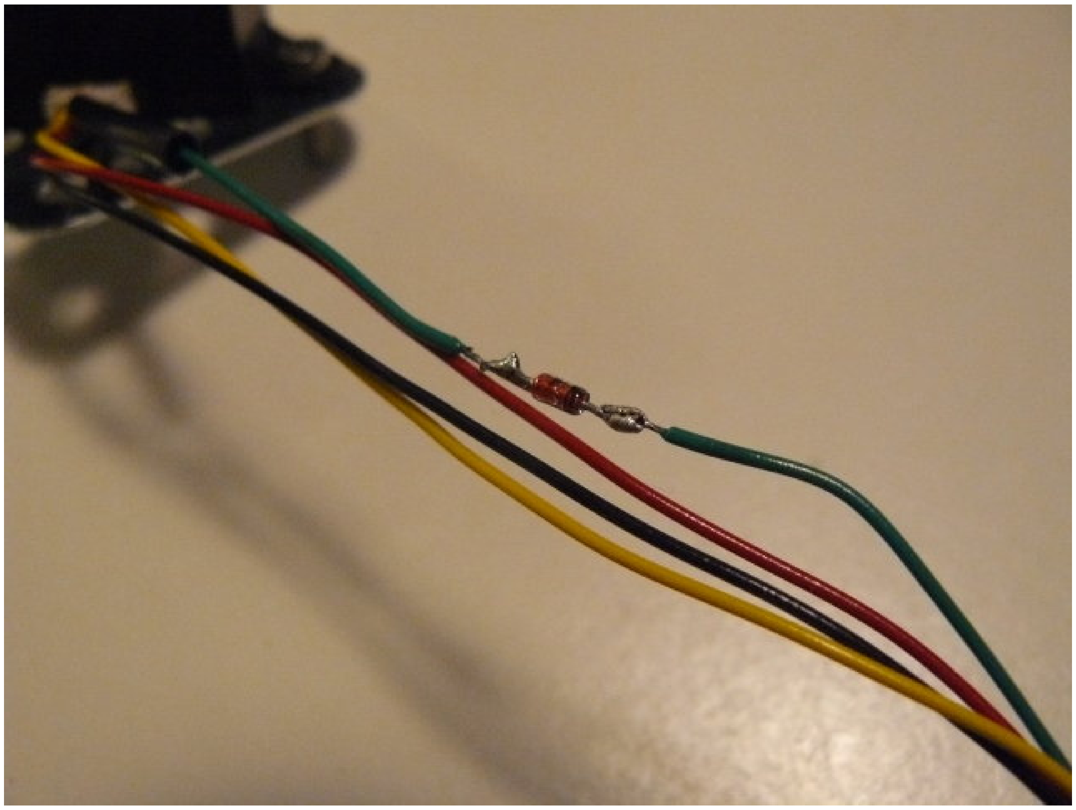
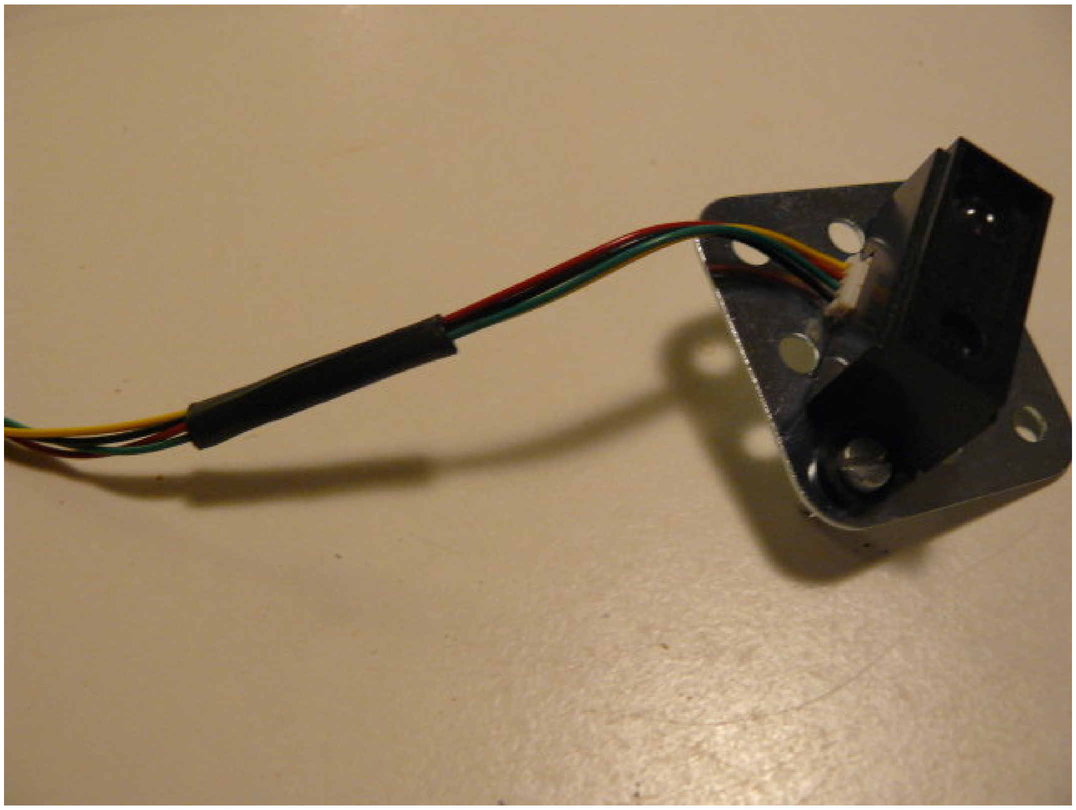


Figure 5: Interfacing Sharp GP2D02 IR range : schematic

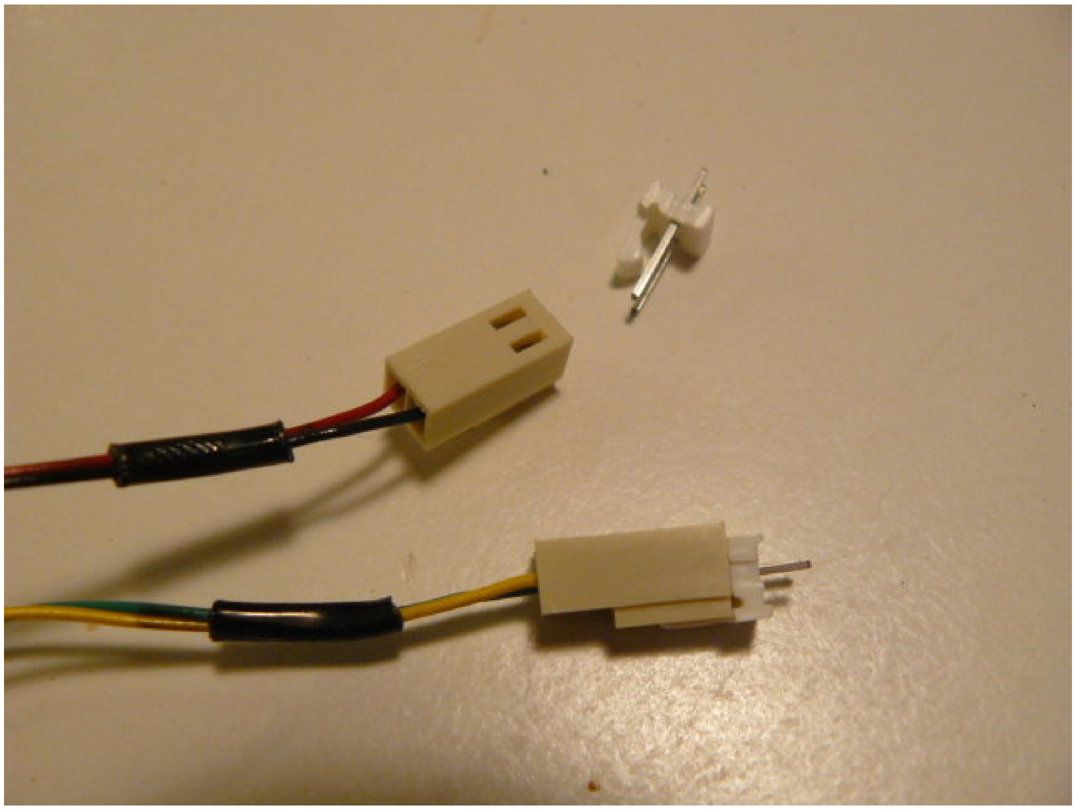
Here's the ranger with the diode soldered on the green wire (which is Vin pin, using your previously created mental note...):



I've also added thermoplastic rubber tubes, to cleanly join all the wires:



Finally, in order to easily plug/unplug the sensor, I've soldered nice polarized connectors:



Writing the program

jallib >=0.3 contains a library, *ir_ranger_gp2d02.jal*, used to handle this kind of rangers. The setup is quite straight forward: just declare your Vin and Vout pins, and pass them to the `gp2d02_read_pins()`. This function returns the distance as a raw value. Directly passing pins allows you to have multiple rangers of this type (many robots have many of them arranged in the front and back sides, to detect and avoid obstacles).

Using PWM libs, we can easily make our LED more or less bright. In the mean time, we'll also transmit the results through a serial link.

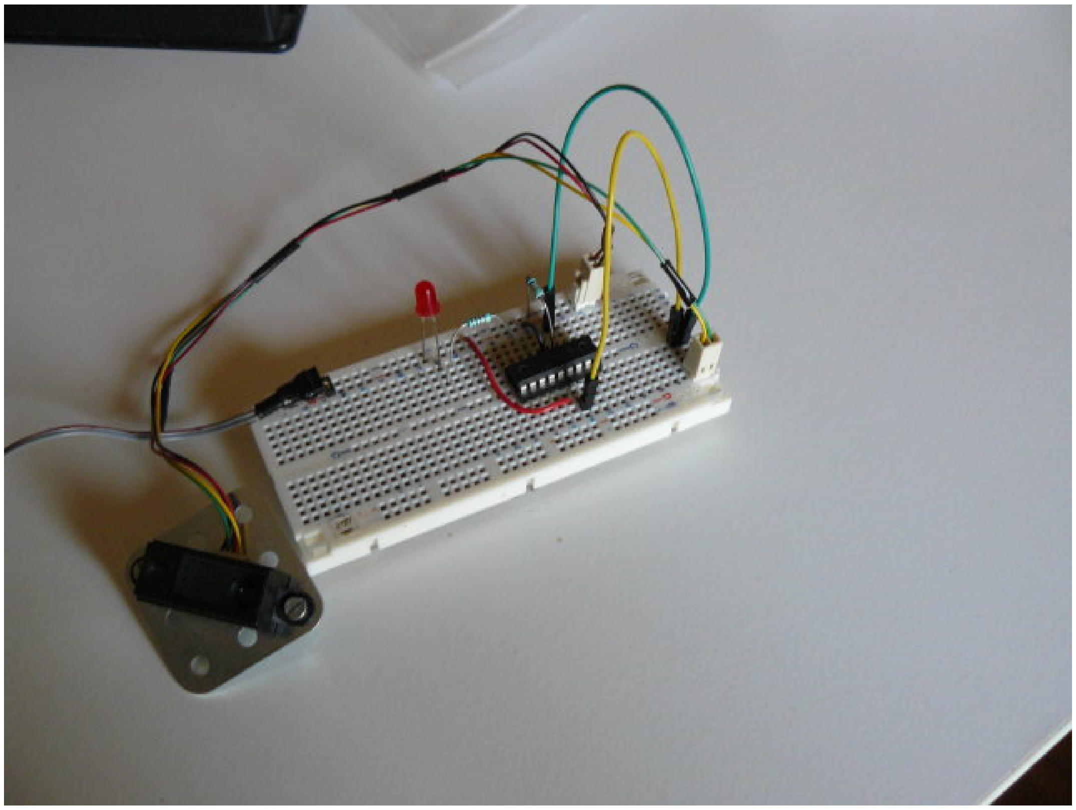
```
var volatile bit gp2d02_vin is pin_a4
var volatile bit gp2d02_vout is pin_a6
var bit gp2d02_vin_direction is pin_a4_direction
var bit gp2d02_vout_direction is pin_a6_direction
include ir_ranger_gp2d02
-- set pin direction (careful: "vin" is the GP2D02 pin's name,
-- it's an input for GP2D02, but an output for PIC !)
gp2d02_vin_direction = output
gp2d02_vout_direction = input

var byte measure
forever loop
   -- read distance from ranger num. 0
   measure = gp2d02_read_pins(gp2d02_vin,gp2d02_vout)
   -- results via serial
   serial_hw_write(measure)
   -- now blink more or less
   pwm1_set_dutycycle(measure)
end loop
```

Note: I could directly pass `pin_A4` and `pin_A6`, but to avoid confusion, I prefer using *aliases*.

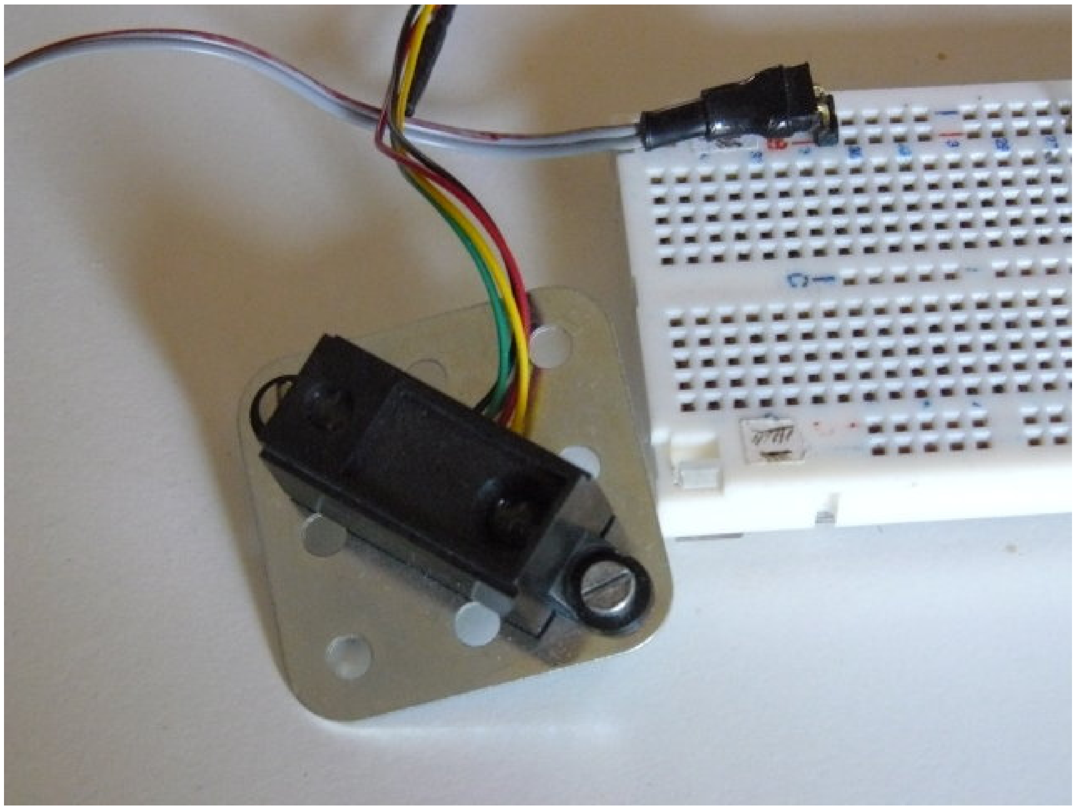
A sample, *16f88_ir_ranger_gp2d02.jal*, is available in *jallib SVN repository*jallib released packages, and also in , starting from version 0.3. You can access downloads *here*.

Building the whole on a breadboard

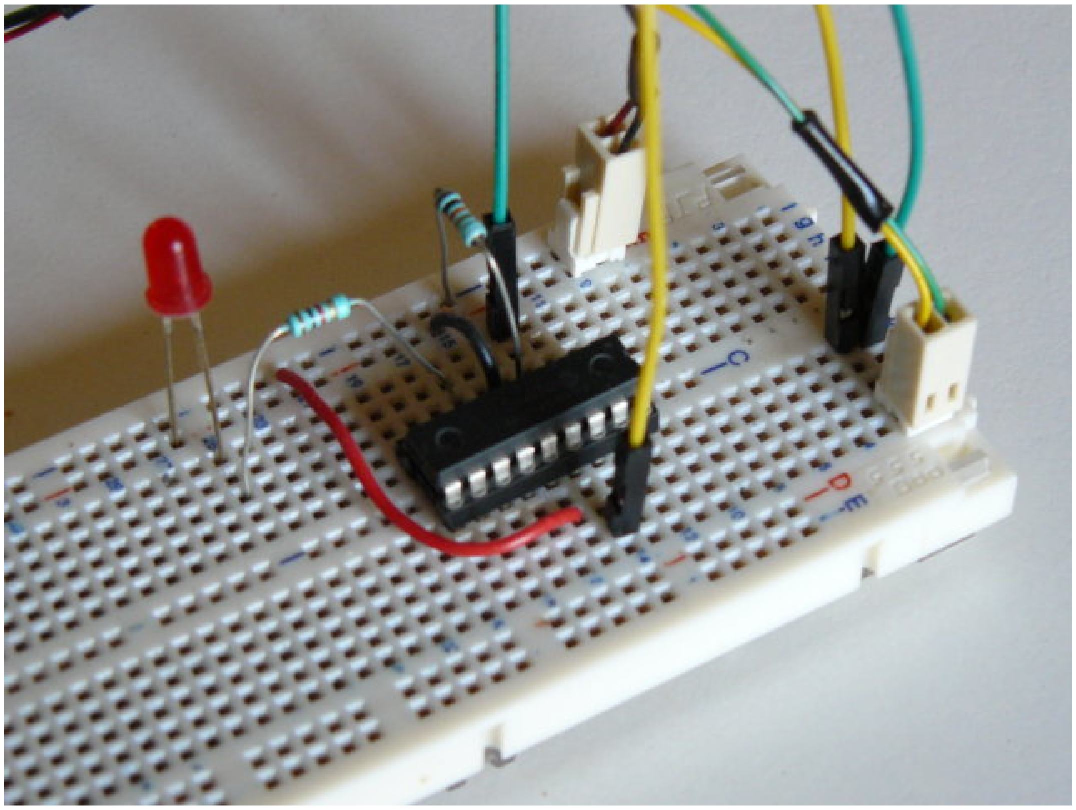
Building the whole on a breadboard



I usually power two tracks on the side, used for the PIC and for the ranger:



Using the same previously created mental note, I connected the yellow Vout pin using a yellow wire, and the green Vin pin using a green wire...



Testing (and the video)

Time to test this nice circuit ! Power the whole, and check no smoke is coming from the PIC or (and) the ranger. Now get an object, like you hand, more or less closed to the ranger and observe the LED, or the serial output... Sweet !

http://www.youtube.com/watch?v=l5AZwv7LzyM

Sébastien Lelong