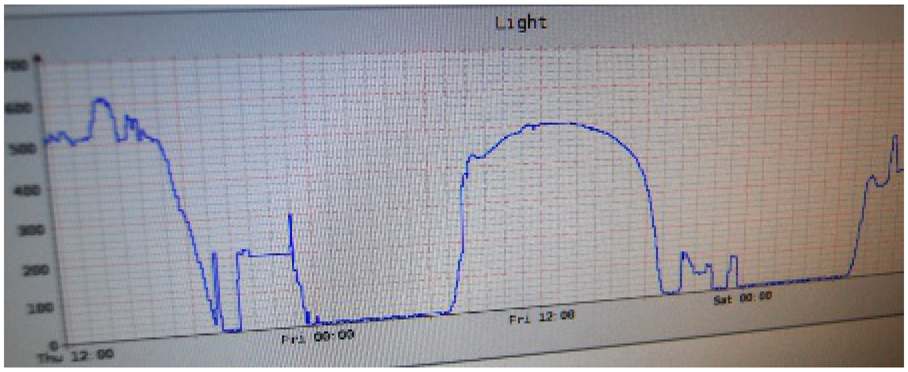
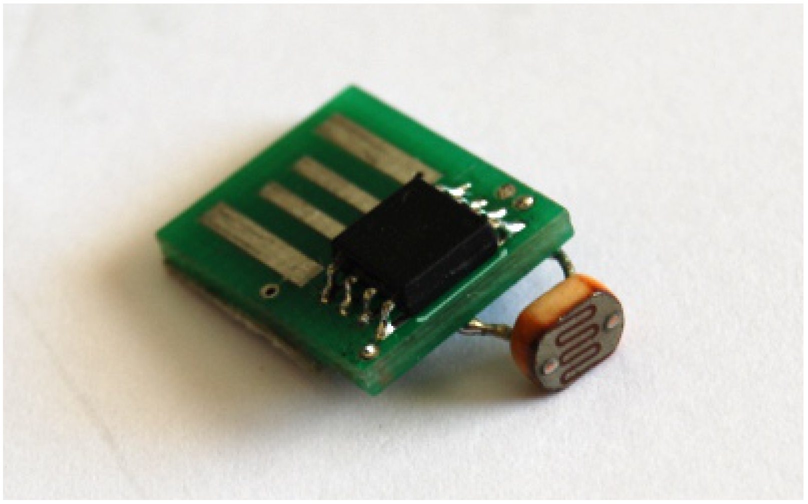
USB SENSORS WITH AT TINY MICROCONTROLLERS

Working with embedded electronics, you will eventually ends up with some sensor between your hands, here I'll show how to make a graph out of it!



This project involves a light sensor, a tiny 8-pin AVR USB key with the V-USB stack, a GNU/Linux system and rrdtool.



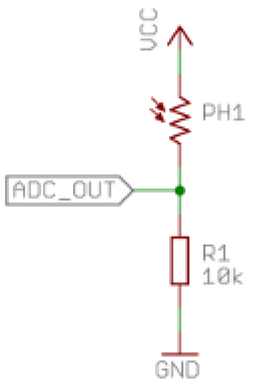
Sensors

If your are working with some digital sensor (like I2C or SPI ones) you just have to connect it to your system and write the protocol, but if you are using an analog sensor, you probably need to do some work on the signals before feeding it to the microcontroller's ADC.

In case of the ATtiny45 and alike, you have a 10bit ADC with three possible voltage references: the external VCC, a 2.56V and a 1.1V internal bandgap.

If you are designing a signal adaption circuit and you want to make reliable measurements, don't forget to include a low-pass filter and a high impedance buffer for the sensor signal, just in case the ADC input doesn't satisfy your needs.

In this case, the light sensor I had works like a variable resistor, so it was just wired in series with a 10kOhm SMD resistor.



Also, the microcontroller includes an internal thermal diode which, when calibrated, can be used to make temperature measurements.

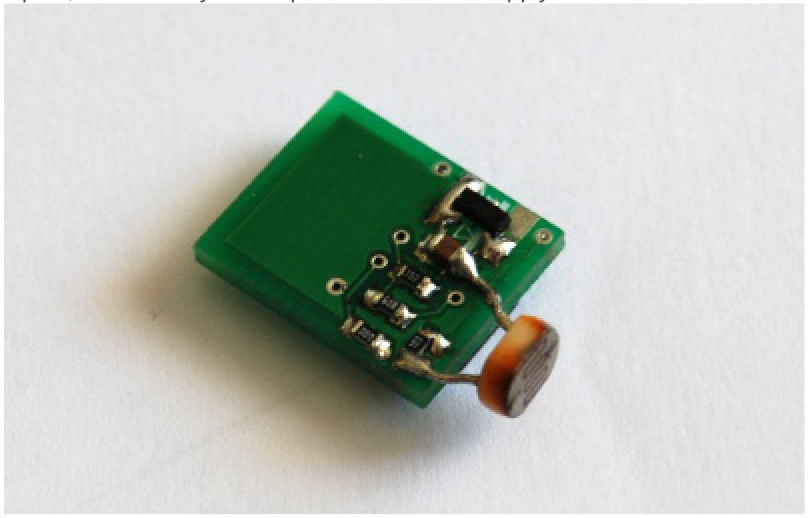
Of course, if you want to make real measurements out of the sensor, you also have to calibrate and linearize it, but that's out of the scope of this post.

USB Interface

When interfacing a sensor with a generic non-embedded GNU/Linux system, the easiest way is to use an USB interface. The free V-USB stack from obdev.at is well suited or this kind of application, as it allows to implement a low-speed USB device with any AVR microcontroller.

The stack has a really low memory footprint, and there are many examples of how to use it for many different purposes.

To implement the sensor, I reused one of my avr-micro-usb boards, which had two SMD pads for GND and an analog inputs, and an easy solder point for the VCC supply.



Firmware

The firmware for the microcontroller is based on the custom-class provided with the V-USB library documentation, and implements two type of “usbFunctionSetup” requests: CUSTOM_RQ_ADC0, to read the light sensor level, and CUSTOM_RQ_ADC1 to read the temperature sensor input.

The ADC routine is the following:

```
uint16_t adc_get (uint8_t cfg)
{
  ADCSRA = ( (1 << ADEN) | /* enable           */
             (0 << ADSC) | /* start conversion */
             (0 << ADATE) | /* free running     */
```

```
                 (1 << ADIF) | /* clear interrupts */
                 (0 << ADIE) | /* interrupt enable */
                 (1 << ADPS2) | (0 << ADPS1) | (0 << ADPS0) );

  ADMUX = cfg;

  /* wait Vref to stabilize */
  _delay_ms(1);

  ADCSRA |= _BV(ADSC);

  loop_until_bit_is_clear(ADCSRA, ADSC);

  ADCSRA = 0x00; /* ADC disable */

  return ADCW;
}
```

Note the delay necessary to let the bandgap reference circuit stabilize. Also, try to turn the internal reference off when not used as it takes quite a lot of power.

To read the value from the Linux system, you can use the *usbtool* utility from the V-USB source code's tools directory, which allows you to quickly send a custom-class USB device request.

As an example, to request the light sensor reading, just run:

```
./usbtool -P "USB Sensor" control in vendor 0 1 0 0

0x64 0x01
```

Graphics

Now that you have the data on your system, let's see how to record that and print out a graphic.

Rrdtool is a free software package to record and display historical data with a round-robin database, which is a fixed-size database files which keeps track of data at fixed time intervals for a fixed time span.

The rrdtool website is full of tutorials, so I'll just skip to the database creation:

```
#!/bin/sh

DB=$HOME/light/light.rrd

# 5 minutes points, 48 hours data
# 30 minutes points, 25 days data
# 2 hours points, 2 months data
# 24 hours points, 2 years data

rrdtool create $DB \
  DS:light:GAUGE:600:U:U  \
  DS:temperature:GAUGE:600:U:U  \
  RRA:AVERAGE:0.5:1:576   \
  RRA:AVERAGE:0.5:6:720   \
  RRA:AVERAGE:0.5:24:720  \
  RRA:AVERAGE:0.5:288:730
```

That’s to keep track of two GAUGE variables, light and temperature, with the time span specified in the script comments.

Then, to update the database with the new real-time data, use usbtool and rrdupdate with some glue logic round, as in:

```
#!/bin/sh

DB=$HOME/light/light.rrd
RRDUPDATE=rrdupdate

light=$( $HOME/light/usbtool -P "USB Sensor" control in vendor 0 1 0 0 | \
        sed 's/0x//g' | awk '{print "0x" $2 $1}' )
light=$( printf %d $light )

temp=$( $HOME/light/usbtool -P "USB Sensor" control in vendor 0 2 0 0 | \
        sed 's/0x//g' | awk '{print "0x" $2 $1}' )
temp=$( printf %d $temp )
temp=$(( $temp - 294 ))

$RRDUPDATE $DB N:$light:$temp
```

This script is run automatically every minute using a crontab job.

The last thing to do is to generate a graphic out the stored data, as in:

```
#!/bin/sh

DB=$HOME/light/light.rrd
RRDTOOL=rrdtool
OUT=/opt/lighttpd/www/htdocs/rrd
OPTS="-w 700 -h 200"

$RRDTOOL graph $OUT/light.png $OPTS -l 0 --start -2d \
  DEF:light=$DB:light:AVERAGE \
  LINE1:light#0000ff:light

$RRDTOOL graph $OUT/temperature.png $OPTS -u 30 -l 0 --start -2d \
  DEF:temperature=$DB:temperature:AVERAGE \
  LINE1:temperature#00ff00:temperature
```

Again, the graphic can easily be updated periodically with a crontab job.

Source : http://fabiobaltieri.com/2011/12/17/usb-sensors-avr/