Contiki: sensors and actuators

Antonio Liñán Colina

Contiki

The Open Source OS for the Internet of Things

- Architectures: 8-bit, 16-bit, 32-bit
- Open Source (source code openly available)
- IPv4/IPv6/Rime networking
- Devices with < 8KB RAM
- Typical applications < 50KB Flash
- Vendor and platform independent
- C language
- Developed and contributed by Universities, Research centers and industry contributors
- +10 years development

Zolertia RE-Mote

Zolertia RE-Mote (Zoul inside)

- ARM Cortex-M3, 32MHz, 32KB RAM, 512KB FLASH
- Double Radio: ISM 2.4GHz & 863-925MHz, IEEE 802.15.4-2006/e/g
- Hardware encryption engine and acceleration
- USB programing ready
- Real-Time Clock and Calendar
- Micro SD slot and RGB colors
- Shutdown mode down to 150nA
- USB 2.0 port for applications
- Built-in LiPo battery charger to work with energy harvesting and solar panels
- On-board RF switch to use both radios over the same RP-SMA connector
- Pads to use an external 2.4GHz over U.Fl connector, o solder a chip antenna


ADC1 (3.3V)
3-pin connector
2.54 mm
ADC3 (5.1V)
3-pin connector
2.54 mm
RESET BUTTON
zolertia™
USER BUTTON
RP-SMA connector
external antenna
I2C/SPI sensors
(5-pin connector 2.54mm)


zolertia
USER BUTTON
RESET BUTTON
Micro USB
programming & debugging
Micro USB
for USB 2.0 applications
Micro SD

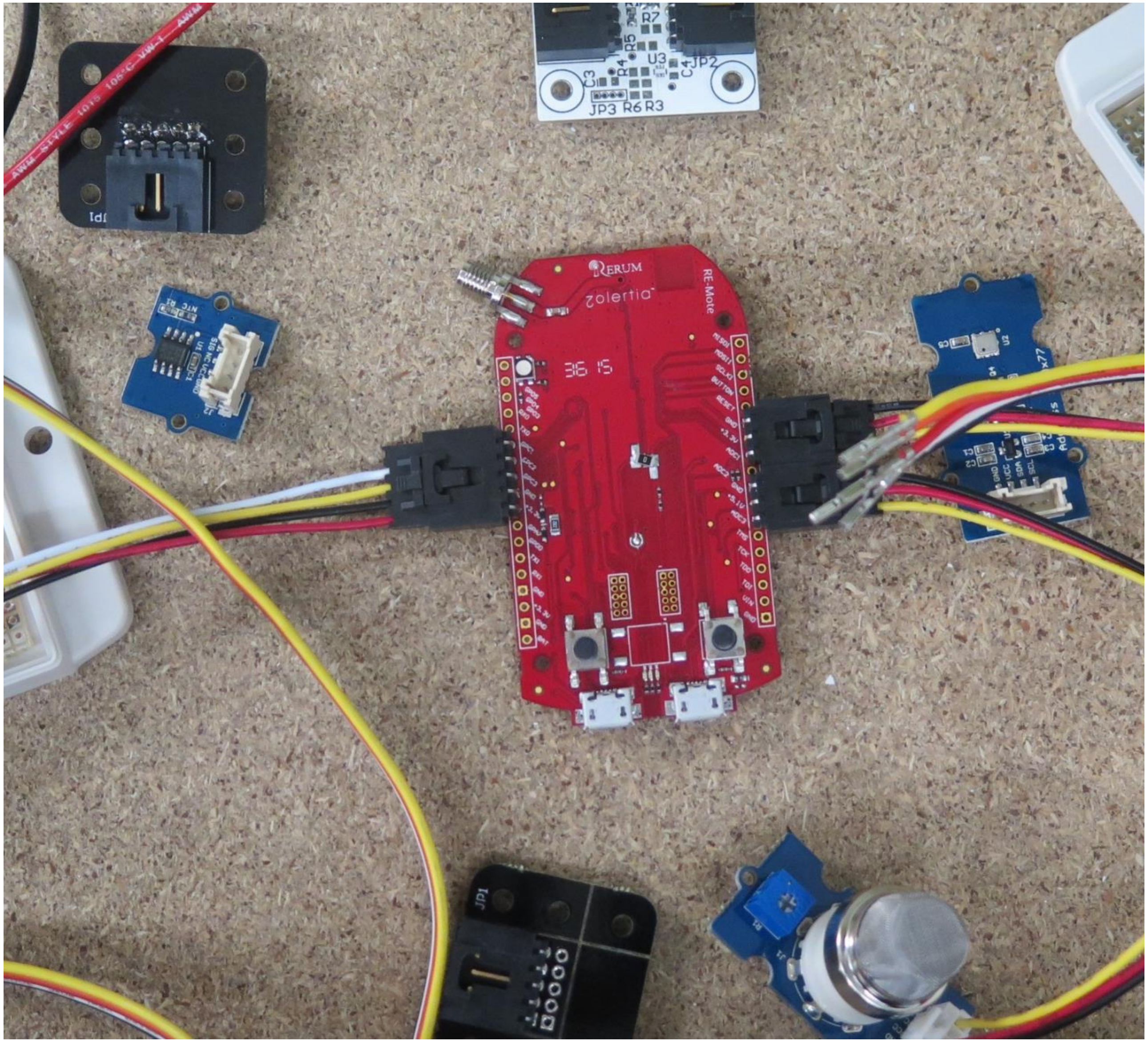

zolertia
thingsquare
Zoul
zolertia

01-basics

examples/zolertia/tutorial/01-basics

Sensors

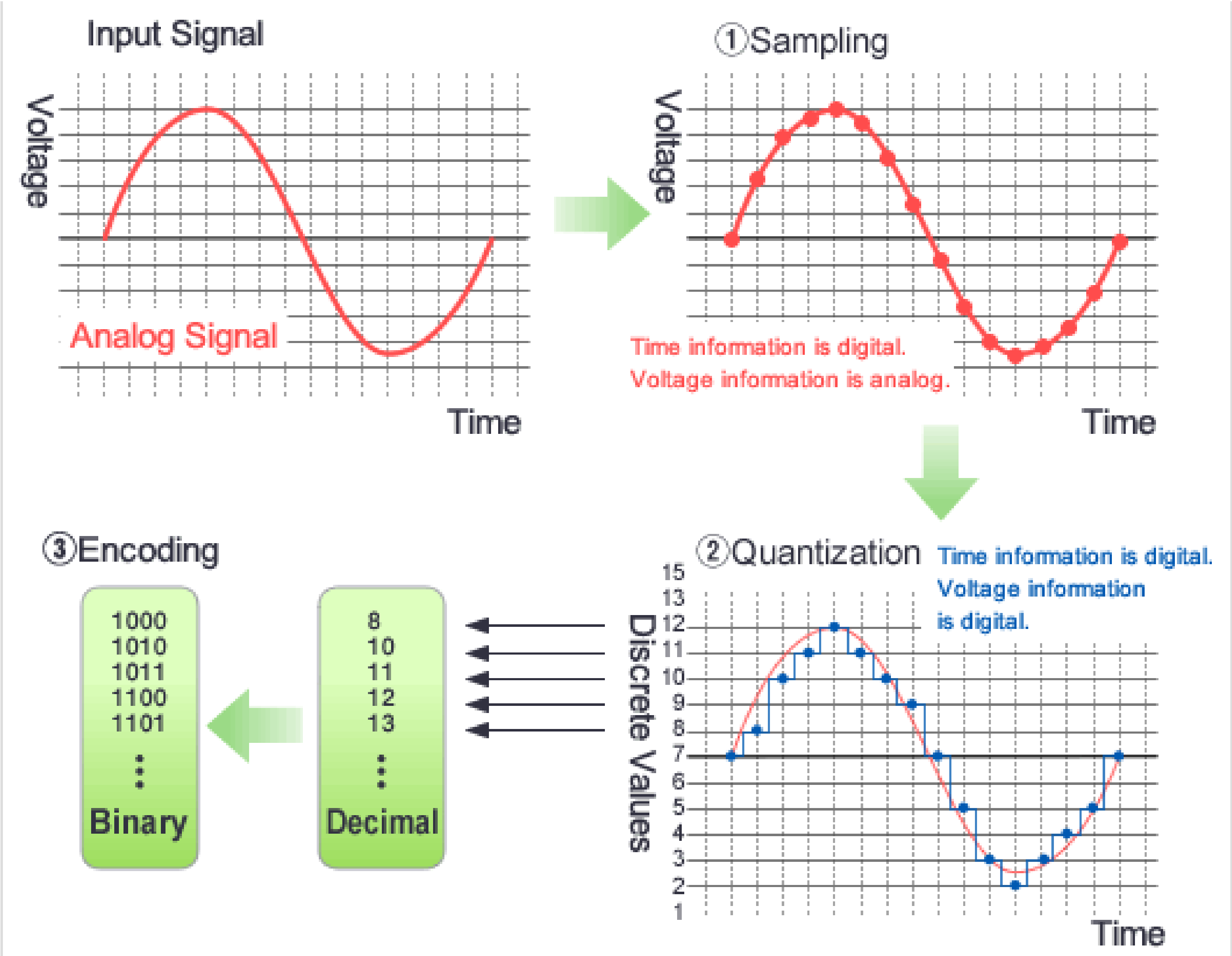
A sensor is a transducer whose purpose is to sense or detect a characteristic of its environment, providing a corresponding output, generally as an electrical or optical signal, related to the quantity of the measured variable.



examples/zolertia/tutorial/01-basics

Analogue sensors

Analogue sensors typically require being connected to an ADC (Analogue to Digital Converter) to translate the analogue (continuous) reading to an equivalent digital value in millivolts.



https://www.renesas.com/en-us.html

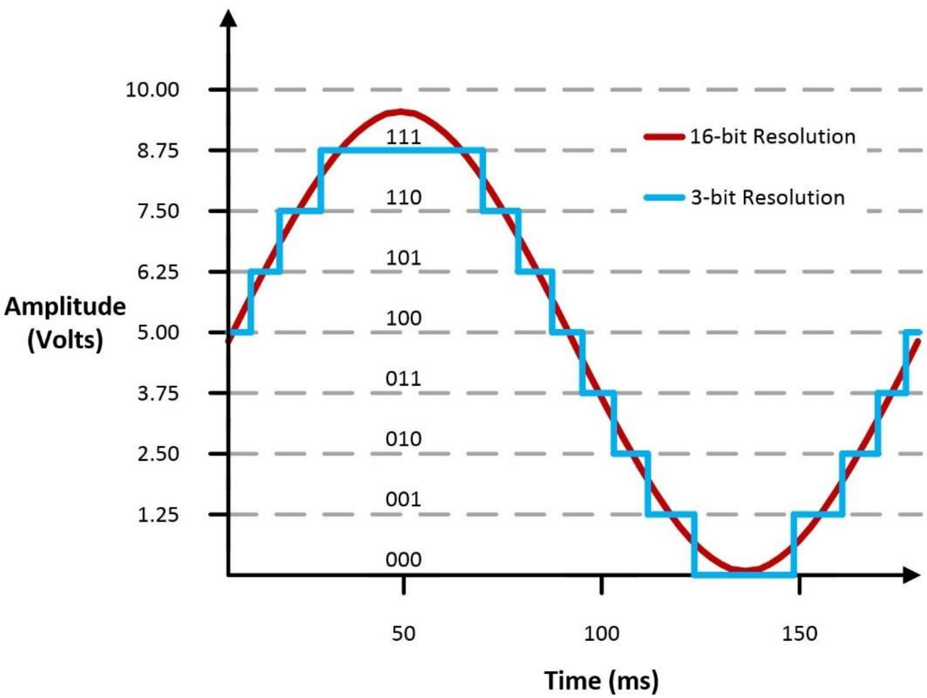
The quality and resolution of the measure depends on both the ADC resolution (10-12 bits in the Z1 and RE-Mote) and the sampling frequency.

As a rule of thumb, the sampling frequency must be at least twice that of the phenomenon you are measuring.

i.e human speech (which may contain frequencies up to 8 kHz), sample at 16 kHz.

The ADC provides a count value (the analogue sensor quantized value). Depending on the sensor we may want to use the count value, or its equivalent voltage value.

$$Vin = \frac{ADC \times Vref}{ADCres} = \frac{ADC \times 3000\,mV}{(1 \ll 12)} = \frac{ADC\ x\ 3000\,mV}{4096}$$



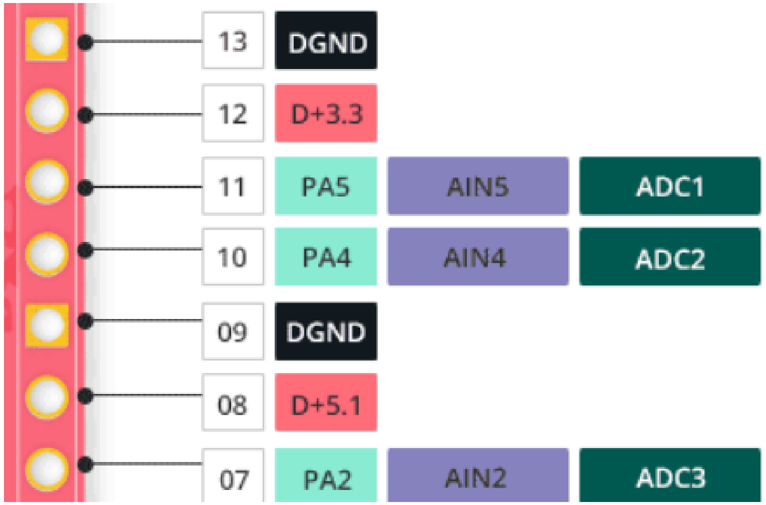
The higher the resolution, higher the sampling time and the size of the value.

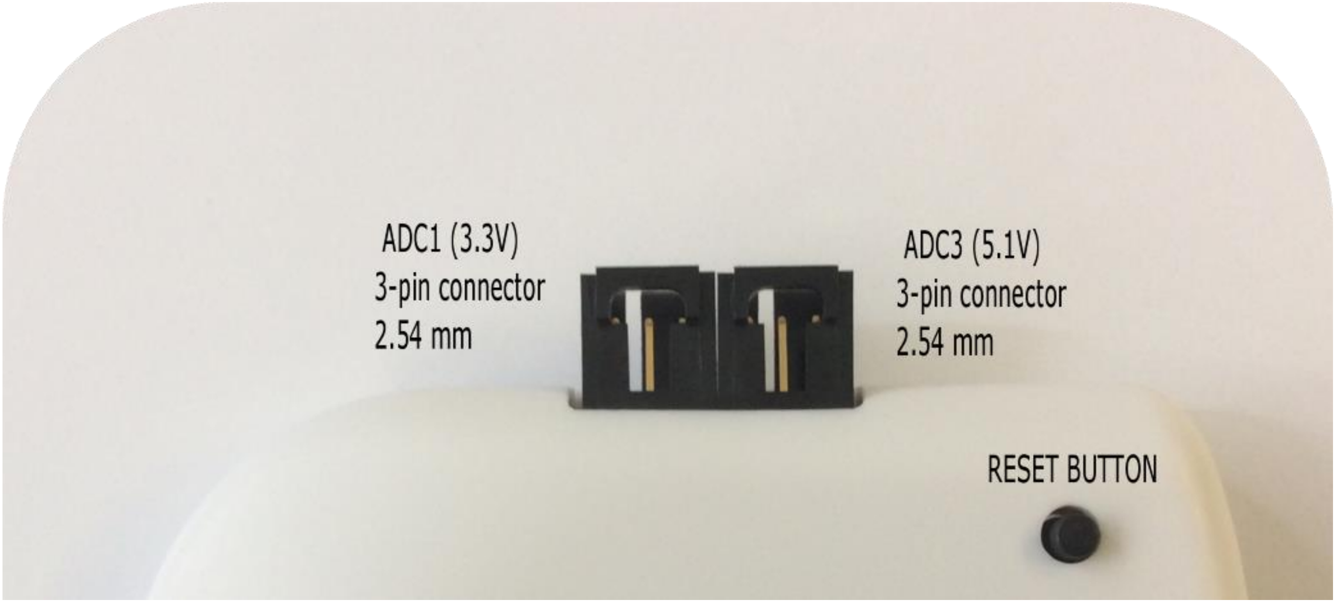
The accuracy and required resolution helps to choose the number of required bits to quantized

examples/zolertia/tutorial/01-basics

Normally the conversion formula (to obtain the actual measuring units, i.e *lux* or *celsius* from the voltage value) are provided by the sensor manufacturer. In other cases, it is required to obtain by measuring and correlating with another calibrated source.

Depending on the accuracy on the sensor, we would either use actual units (i.e 1024 *lux*), or just characterize thresholds (i.e dark, bright). For the later, using a cheap sensor might be enough for the application requirements.



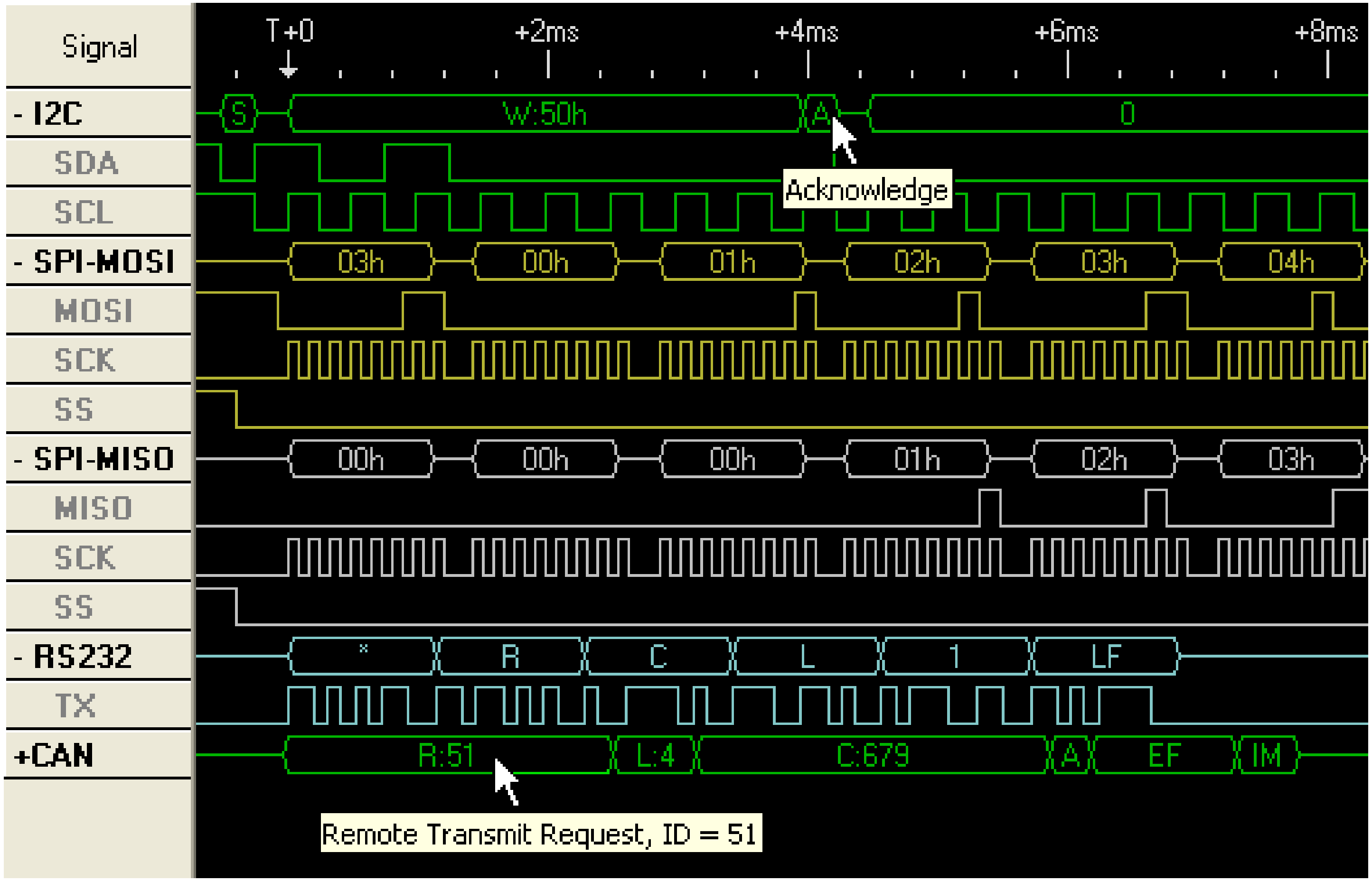


examples/zolertia/tutorial/01-basics

Digital sensors

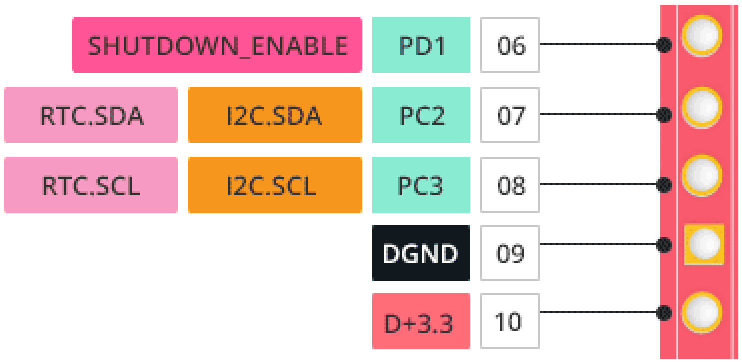
A digital sensor is an electronic or electrochemical sensor, where data conversion and transmission are done digitally.

Digital sensors are normally interfaced over a digital communication protocol such as I2C, SPI, 1-Wire, Serial or depending on the manufacturer, a proprietary protocol normally on a ticking clock.

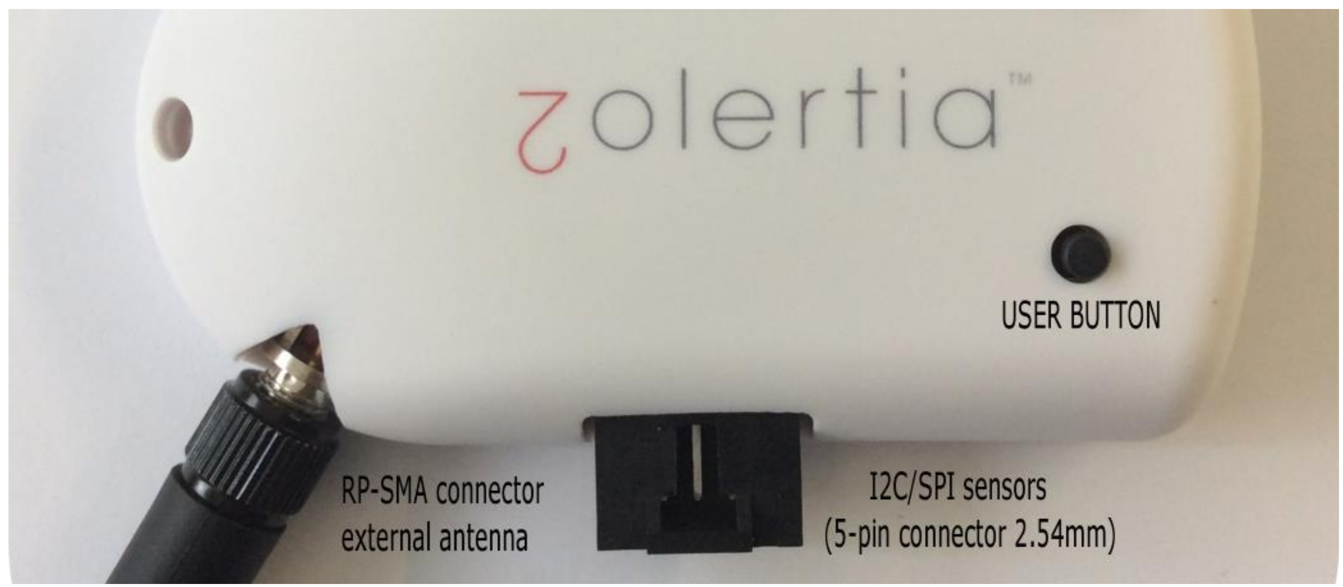


http://www.pctestinstruments.com/

examples/zolertia/tutorial/01-basics



The default configuration matches the I2C, but it can be reconfigured as SPI, even UART. Pins in the ARM Cortex-M3 CC2538 can be multiplexed to different controllers



examples/zolertia/tutorial/01-basics

Digital sensors allow a more extended set of commands (turn on, turn off, configure interrupts, resolution, etc).

With a digital light sensor for example, you could set a threshold value and let the sensor send an interrupt when reached, without the need for continuous polling.

Remember to check the specific sensor information and data sheet for more information.

examples/zolertia/tutorial/01-basics

Sensors in Contiki

Contiki defines a common API to implement and use both sensors and actuators.

Normally platform-specific sensor implementations are located in the platform’s /dev folder.

Contiki has ADC, I2C, SPI and UART libraries, normally sensor and actuators implementations use these, so the porting effort is greatly minimized.

Macro to define a sensor

```
SENSORS_SENSOR (sensor, SENSOR_NAME, value, configure, status);
```

Sensor structure and API

```
struct sensors_sensor {
   char *       type;
   int          (* value)     (int type);
   int          (* configure) (int type, int value);
   int          (* status)    (int type);
};
```

core/lib/sensors.h

Default sensors constants and macros

```
/* some constants for the configure API */
#define SENSORS_HW_INIT 128 /* internal - used only for initialization */
#define SENSORS_ACTIVE 129 /* ACTIVE => 0 -> turn off, 1 -> turn on */
#define SENSORS_READY 130 /* read only */

#define SENSORS_ACTIVATE(sensor) (sensor).configure(SENSORS_ACTIVE, 1)
#define SENSORS_DEACTIVATE(sensor) (sensor).configure(SENSORS_ACTIVE, 0)
```

platforms/zoul/dev/zoul-sensors.c

General structure to link the defined sensors

```
SENSORS (&button_sensor, &vdd3_sensor, &cc2538_temp_sensor, &adc_sensors);
```

SENSORS structure expanded

```
const struct sensors_sensor *sensors[] = {
  &button_sensor,
  &vdd3_sensor,
  &cc2538_temp_sensor,
  &adc_sensors,
  ((void *)0)
};
```

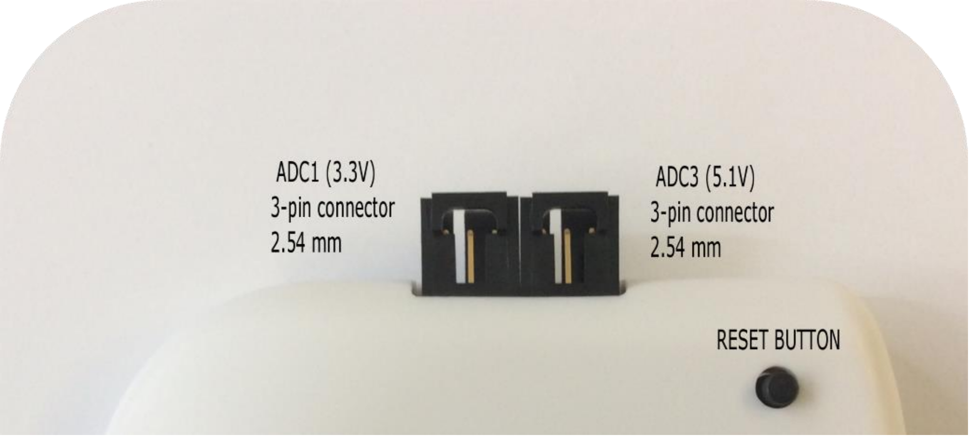
platforms/zoul/dev/zoul-sensors.c

General structure to link the defined sensors

```
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR(adc_zoul, ADC_ZOUL, value, configure, status);
/*---------------------------------------------------------------------------*/
```

How to use the *adc_zoul* API:

```
/* The ADC zoul library configures the on-board enabled ADC channels, more
 * information is provided in the board.h file of the platform
 */
adc_zoul.configure(SENSORS_HW_INIT, ZOUL_SENSORS_ADC_ALL);
printf("ADC1 = %u mV\n", adc_zoul.value(ZOUL_SENSORS_ADC1));
printf("ADC3 = %u mV\n", adc_zoul.value(ZOUL_SENSORS_ADC3));
```

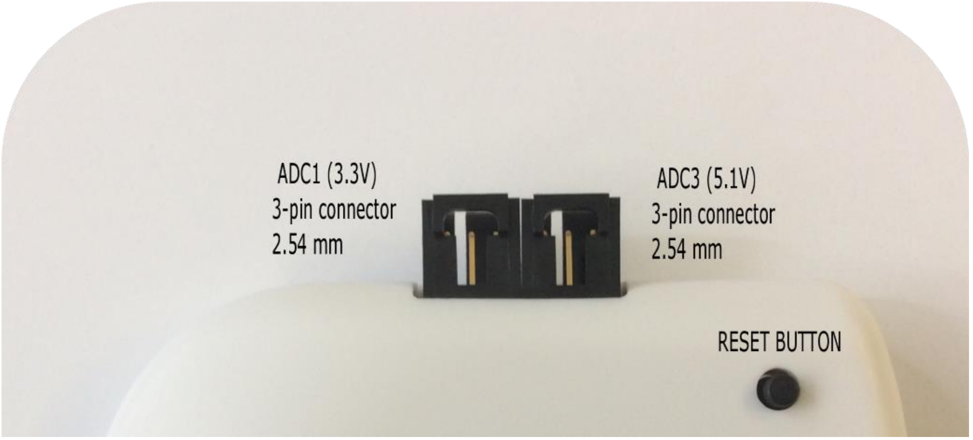


examples/zolertia/tutorial/01-basics/06-adc.c

General structure to link the defined sensors

```
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR(adc_zoul, ADC_ZOUL, value, configure, status);
/*---------------------------------------------------------------------------*/

/* The ADC zoul library configures the on-board enabled ADC channels, more
 * information is provided in the board.h file of the platform
 */
adc_zoul.configure(SENSORS_HW_INIT, ZOUL_SENSORS_ADC_ALL);
printf("ADC1 = %u mV\n", adc_zoul.value(ZOUL_SENSORS_ADC1));
printf("ADC3 = %u mV\n", adc_zoul.value(ZOUL_SENSORS_ADC3));
```

You can enable one or more sensors at the same time. Pins in the PORT A (PA) are used as ADC. The ADC can be configured using mask values

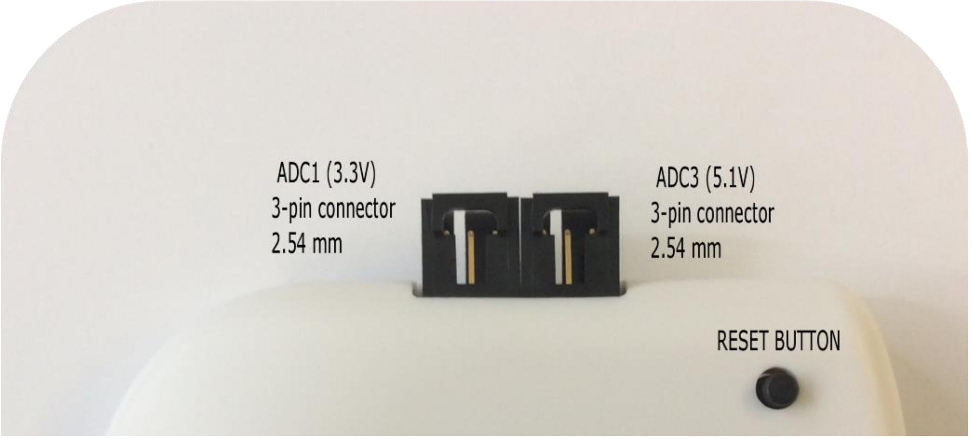


examples/zolertia/tutorial/01-basics/06-adc.c

General structure to link the defined sensors

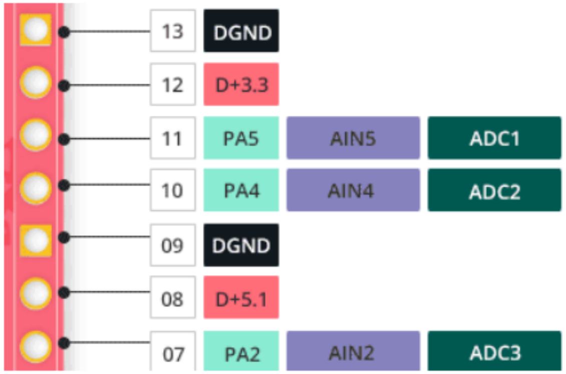
```
/*---------------------------------------------------------------------------*/
SENSORS_SENSOR(adc_zoul, ADC_ZOUL, value, configure, status);
/*---------------------------------------------------------------------------*/

/* The ADC zoul library configures the on-board enabled ADC channels, more
 * information is provided in the board.h file of the platform
 */
adc_zoul.configure(SENSORS_HW_INIT, ZOUL_SENSORS_ADC_ALL);
printf("ADC1 = %u mV\n", adc_zoul.value(ZOUL_SENSORS_ADC1));
printf("ADC3 = %u mV\n", adc_zoul.value(ZOUL_SENSORS_ADC3));
```

The returned value is in milliVolts with one extra precision digit, no need to convert from ADC raw count to voltage



examples/zolertia/tutorial/01-basics/06-adc.c

```
#define ADC_SENSORS_PORT          GPIO_A_NUM /**< ADC GPIO control port */

#ifndef ADC_SENSORS_CONF_ADC1_PIN
#define ADC_SENSORS_ADC1_PIN      5            /**< ADC1 to PA5, 3V3    */
#else
#if ((ADC_SENSORS_CONF_ADC1_PIN != -1) && (ADC_SENSORS_CONF_ADC1_PIN != 5))
#error "ADC1 channel should be mapped to PA5 or disabled with -1"
#else
#define ADC_SENSORS_ADC1_PIN ADC_SENSORS_CONF_ADC1_PIN
#endif
#endif

#ifndef ADC_SENSORS_CONF_ADC3_PIN
#define ADC_SENSORS_ADC3_PIN      2            /**< ADC3 to PA2, 5V     */
#else
#if ((ADC_SENSORS_CONF_ADC3_PIN != -1) && (ADC_SENSORS_CONF_ADC3_PIN != 2))
#error "ADC3 channel should be mapped to PA2 or disabled with -1"
#else
#define ADC_SENSORS_ADC3_PIN ADC_SENSORS_CONF_ADC3_PIN
#endif
#endif
```



platforms/zoul/remote/board.h

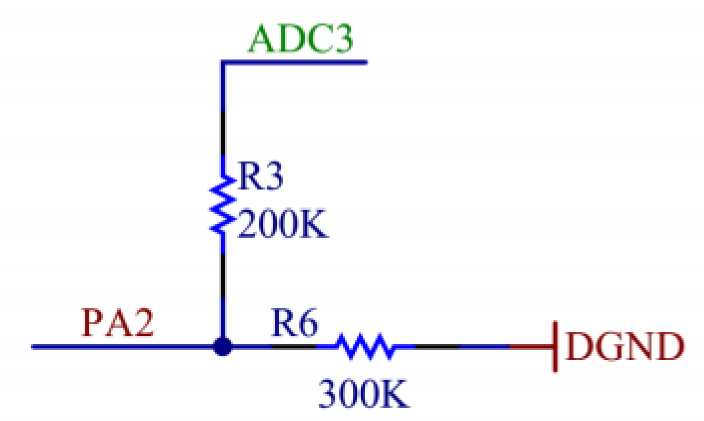
```
#ifndef ADC_SENSORS_CONF_ADC1_PIN
#define ADC_SENSORS_ADC1_PIN      5              /**< ADC1 to PA5, 3V3    */
#else
#if ((ADC_SENSORS_CONF_ADC1_PIN != -1) && (ADC_SENSORS_CONF_ADC1_PIN != 5))
#error "ADC1 channel should be mapped to PA5 or disabled with -1"
#else
#define ADC_SENSORS_ADC1_PIN ADC_SENSORS_CONF_ADC1_PIN
#endif
#endif
```

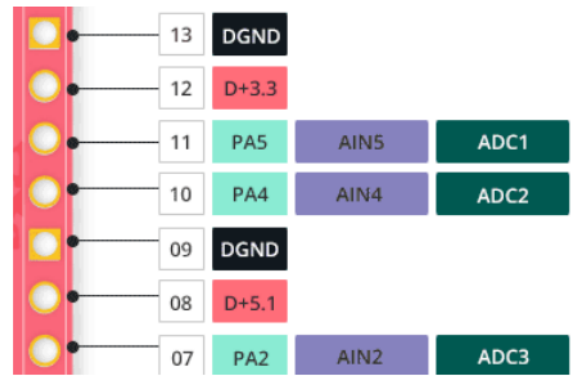
```
/*
 * PA0-PA3 are hardcoded to UART0 and the user button for most Zolertia
 * platforms, the following assumes PA0-1 shall not be used as ADC input, else
 * re-write the below definitions
 */
#define ZOUL_SENSORS_ADC_MIN        2  /**< PA1 pin mask */

/* ADC phidget-like connector ADC1 */
#if ADC_SENSORS_ADC1_PIN >= ZOUL_SENSORS_ADC_MIN
#define ZOUL_SENSORS_ADC1           GPIO_PIN_MASK(ADC_SENSORS_ADC1_PIN)
#else
#define ZOUL_SENSORS_ADC1           0
#endif
```

```
#define ZOUL_SENSORS_ADC_ALL     (ZOUL_SENSORS_ADC1 + ZOUL_SENSORS_ADC2 + \
                                  ZOUL_SENSORS_ADC3 + ZOUL_SENSORS_ADC4 + \
                                  ZOUL_SENSORS_ADC5 + ZOUL_SENSORS_ADC6)
```

platforms/zoul/dev/adc-zoul.c

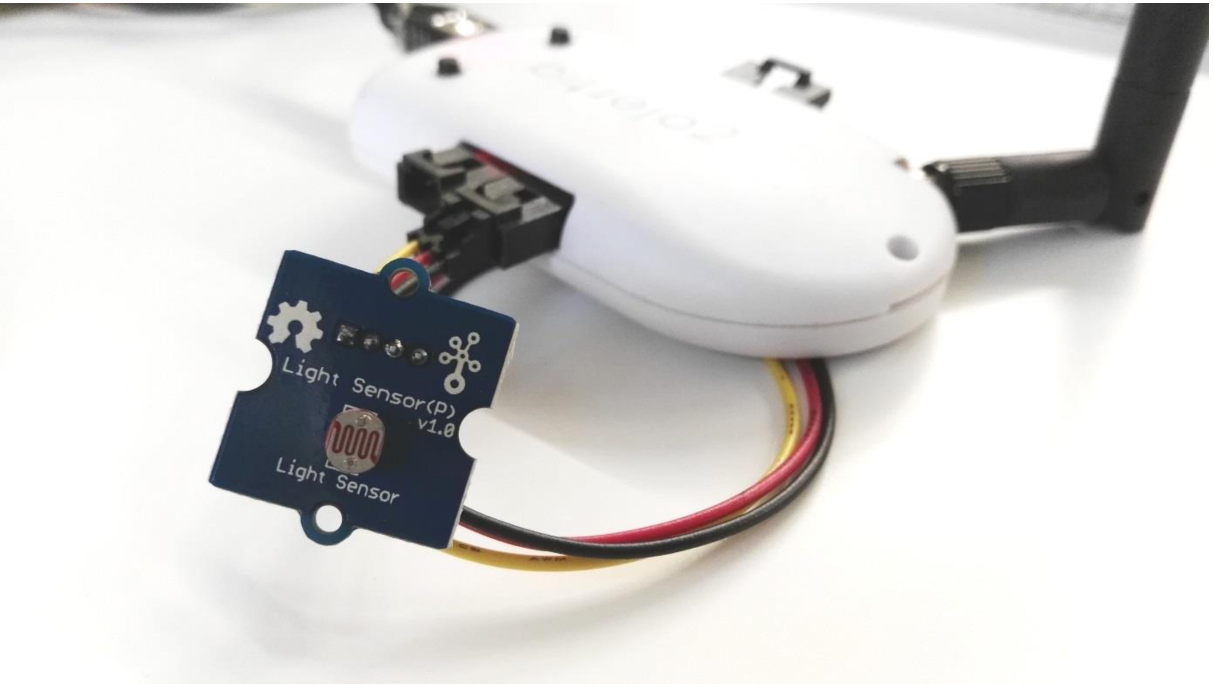
Note 3V and 5V analogue sensors can be connected, even if the RE-Mote works at 3V. The ADC3 channel has a voltage divider





$$V_{adc3(5V)} = \frac{V_{adc3(3V)} * 5}{3}$$

platforms/zoul/dev/adc-zoul.c

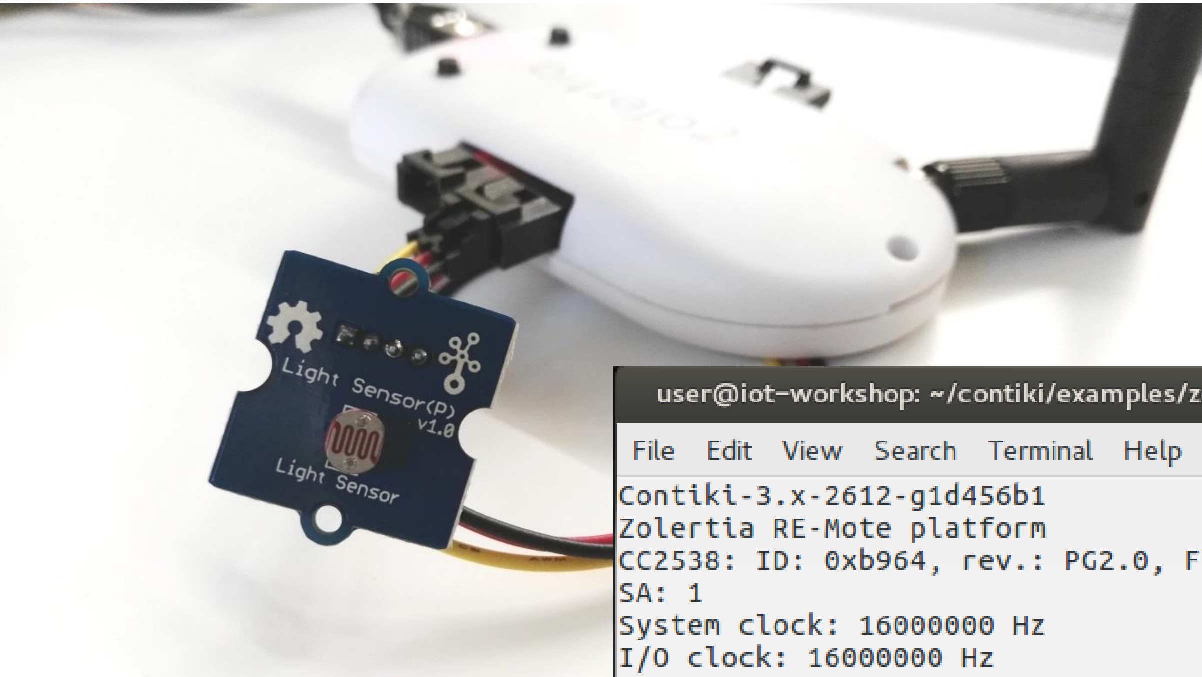


Connect an analogue sensor to the ADC1 connector (3V), and compile and program the 06-adc example:

```
make 06-adc.upload && make login
```

http://zolertia.io/product/sensors/analog-light-sensor
http://www.seeedstudio.com/wiki/Grove_-_Light_Sensor

examples/zolertia/tutorial/01-basics/06-adc.c



Specifications

- Voltage: 3-5V
- Supply Current: 0.5-3mA
- Light resistance: 20KΩ
- Dark resistance: 1MΩ
- Response time: 20-30 secs
- Peak Wavelength: 540 nm
- Ambient temperature: -30~70℃
- LDR Used: GL5528

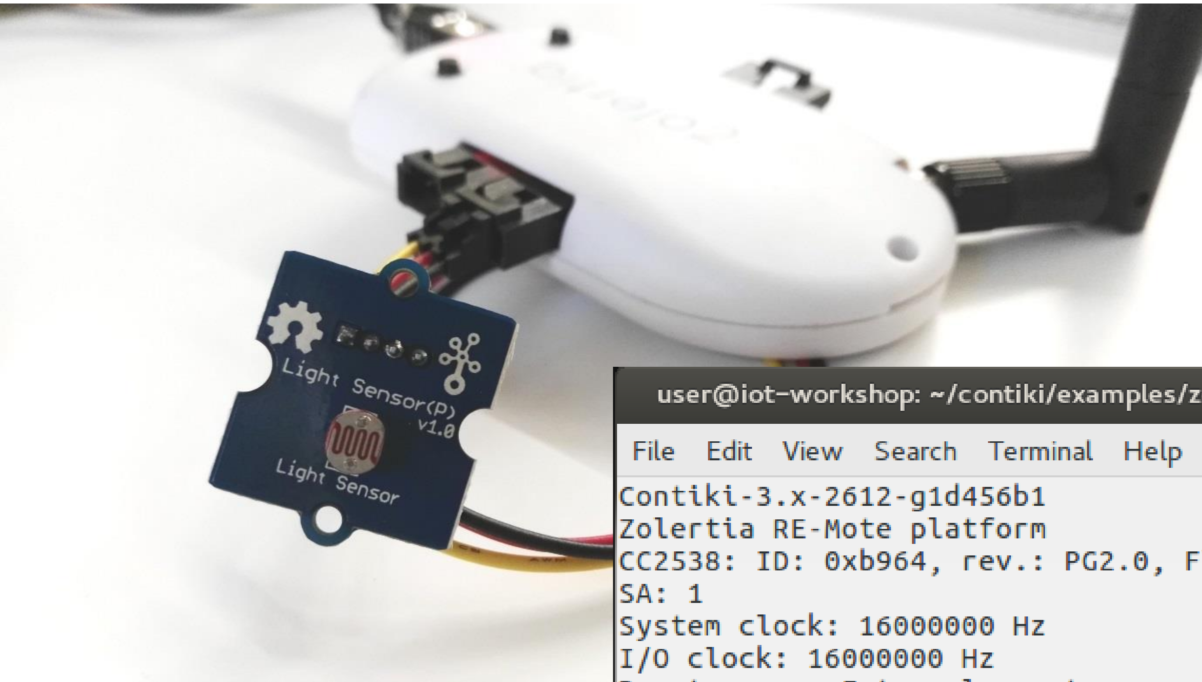
user@iot-workshop: ~/contiki/examples/zolertia/tu

File Edit View Search Terminal Help

```
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1,
SA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
 RDC: ContikiMAC
ADC1 = 216 mV
ADC3 = 1492 mV
ADC1 = 272 mV
ADC3 = 1516 mV
ADC1 = 128 mV
ADC3 = 1488 mV
ADC1 = 64 mV
ADC3 = 1520 mV
ADC1 = 64 mV
ADC3 = 1492 mV
ADC1 = 8704 mV
```

The light sensor is covered Voltage: 0.0064-0.021V, in dark environments the sensor resistance is up to 1MΩ, so the voltaje will decrease as shown

examples/zolertia/tutorial/01-basics/06-adc.c



Specifications

- Voltage: 3-5V
- Supply Current: 0.5-3mA
- Light resistance: 20KΩ
- Dark resistance: 1MΩ
- Response time: 20-30 secs
- Peak Wavelength: 540 nm
- Ambient temperature: -30~70℃
- LDR Used: GL5528

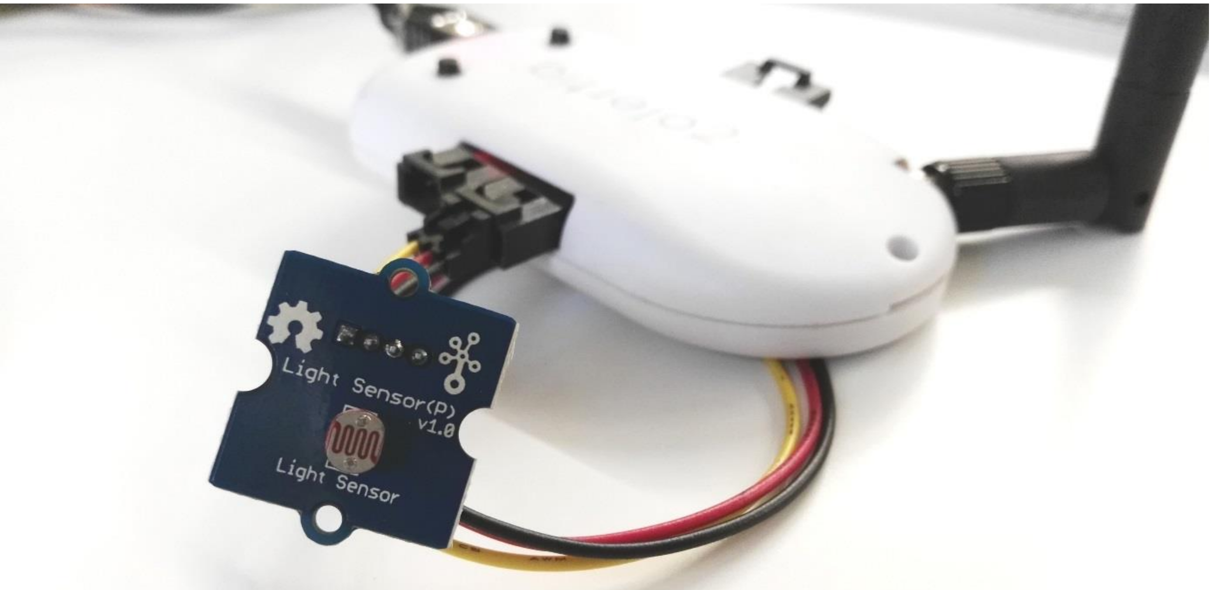
user@iot-workshop: ~/contiki/examples/zolertia/tu

File Edit View Search Terminal Help

```
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1,
SA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
 RDC: ContikiMAC
ADC1 = 21068 mV
ADC3 = 1504 mV
ADC1 = 20792 mV
ADC3 = 1496 mV
ADC1 = 21068 mV
ADC3 = 1508 mV
ADC1 = 20804 mV
ADC3 = 1496 mV
ADC3 = 1492 mV
ADC1 = 8704 mV
```

Light sensor with no cover. Voltage: 2.10-2.08V, in light environments the resistance of the sensor decreases (20KΩ and lower), so the voltage will be higher as shown

examples/zolertia/tutorial/01-basics/06-adc.c



Specifications

- Voltage: 3-5V
- Supply Current: 0.5-3mA
- Light resistance: 20KΩ
- Dark resistance: 1MΩ
- Response time: 20-30 secs
- Peak Wavelength: 540 nm
- Ambient temperature: -30~70℃
- LDR Used: GL5528

```
Rsensor=(float)(1023-sensorValue)*10/sensorValue;
```

As we have voltage values we need the actual ADC count, for the 0.21V (dark):

$$\text{ADC} = \frac{Vin \times ADCres}{Vref} = \frac{21mV \times 4096}{3000mV} = 28.672$$

Then use the above formula to get the sensor resistance (in KΩ), note we use 4095 as we use a 12-bits ADC):

$$Rsensor = \frac{(4095 - 28.672) \times 10}{28.672} = 1418{,}22K\Omega$$

examples/zolertia/tutorial/01-basics/06-adc.c

Specification	Light resistance (10Lux) (KΩ)	Dark resistance (MΩ)	γ^{100}_{10}	Response time (ms)		Illuminance resistance Fig. No.
				Increase	Decrease	
Φ5 series	5-10	0.5	0.5	30	30	2
	10-20	1	0.6	20	30	3
	20-30	2	0.6	20	30	4
	30-50	3	0.7	20	30	4
	50-100	5	0.8	20	30	5
	100-200	10	0.9	20	30	6

Specifications

- Voltage: 3-5V
- Supply Current: 0.5-3mA
- Light resistance: 20KΩ
- Dark resistance: 1MΩ
- Response time: 20-30 secs
- Peak Wavelength: 540 nm
- Ambient temperature: -30~70℃
- LDR Used: GL5528

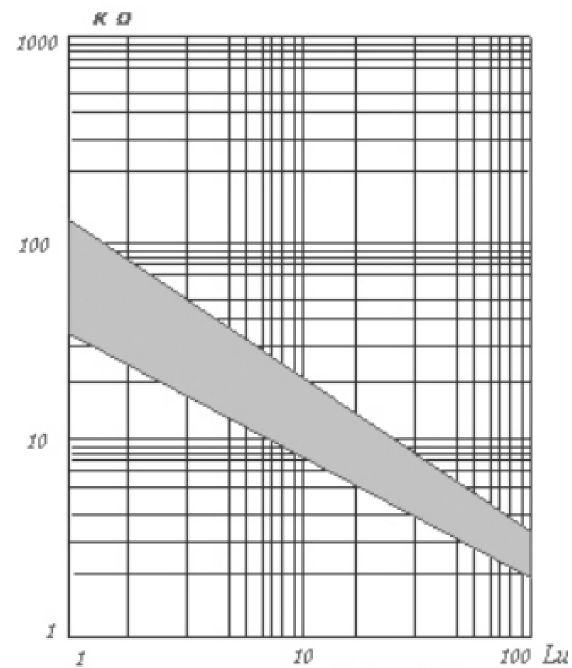


Fig.2

An approximated formula using the curves in the LDR datasheet:

$$\text{Light} = 63 \times Rsensor^{-0.7} = 63 \times 1418{,}22^{-0.7} = 0{,}39\ lux$$

https://forum.arduino.cc/index.php?topic=331679.0

examples/zolertia/tutorial/01-basics/06-adc.c

Specification	Light resistance (10Lux) (KΩ)	Dark resistance (MΩ)	γ_{10}^{100}	Response time (ms)		Illuminance resistance Fig. No.
				Increase	Decrease	
Φ5 series	5-10	0.5	0.5	30	30	2
	10-20	1	0.6	20	30	3
	20-30	2	0.6	20	30	4
	30-50	3	0.7	20	30	4
	50-100	5	0.8	20	30	5
	100-200	10	0.9	20	30	6

Specifications

- Voltage: 3-5V
- Supply Current: 0.5-3mA
- Light resistance: 20KΩ
- Dark resistance: 1MΩ
- Response time: 20-30 secs
- Peak Wavelength: 540 nm
- Ambient temperature: -30~70℃
- LDR Used: GL5528

An approximated formula using the curves in the LDR datasheet:

$$\text{Light} = 63 \times Rsensor^{-0.7} = 63 \times 1418{,}22^{-0.7} = 0{,}39\ lux$$

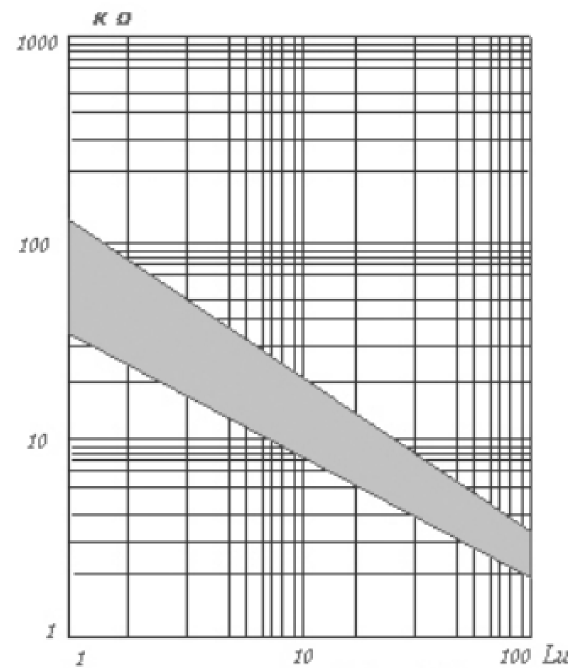


Fig.2

What is the lux value of the light environment (the one proportional to 2.10V) in previous slides? What would be the maximum lux value?

https://forum.arduino.cc/index.php?topic=331679.0

examples/zolertia/tutorial/01-basics/06-adc.c

Light (lux) value for the previous example

Example lux values	
Clear night, no moon:	.002 lx
Clear night, full moon:	.27-1 lx
Family living room:	50 lx
Sunrise/sunset:	300-500 lx
Overcast day:	1000 lx
Daylight:	10000-25000 lx
Direct sunlight:	32000-130000 lx

$$\text{ADC}= \frac{Vin \times ADCres}{Vref} = \frac{2100mV \times 4096}{3000mV} = 2867.2$$

$$Rsensor = \frac{(4095 - 2867{,}2) \times 10}{2867{,}2} = 4{,}28K\Omega$$

$$\text{Light (lux)}= 63 \times Rsensor^{-0.7} = 63 \times 4{,}28^{-0.7} = 22{,}76\ lux$$

Maximum lux value

$$Rsensor = \frac{10}{3000} = 0{,}003K\Omega$$

$$\text{Light (lux)}= 63 \times Rsensor^{-0.7} = 63 \times 0{,}003^{-0.7} = 3675{,}83\ lux$$

http://www.globalspec.com/

examples/zolertia/tutorial/01-basics/06-adc.c

Exponential and logarithms are processing-expensive operations.

Instead of calculating the lux values, one could use the correlated voltage or ADC count value in the application (to trigger an alarm if the room is dark), or send over the radio the ADC count and let the remote server/application to do the math

The RE-Mote has two on-board sensors: voltage level and core temperature:

```
batt = vdd3_sensor.value(CC2538_SENSORS_VALUE_TYPE_CONVERTED);
printf("VDD = %u mV\n", (uint16_t)batt);

temp = cc2538_temp_sensor.value(CC2538_SENSORS_VALUE_TYPE_CONVERTED);
printf("Core temperature = %d.%u C\n", temp / 1000, temp % 1000);
```

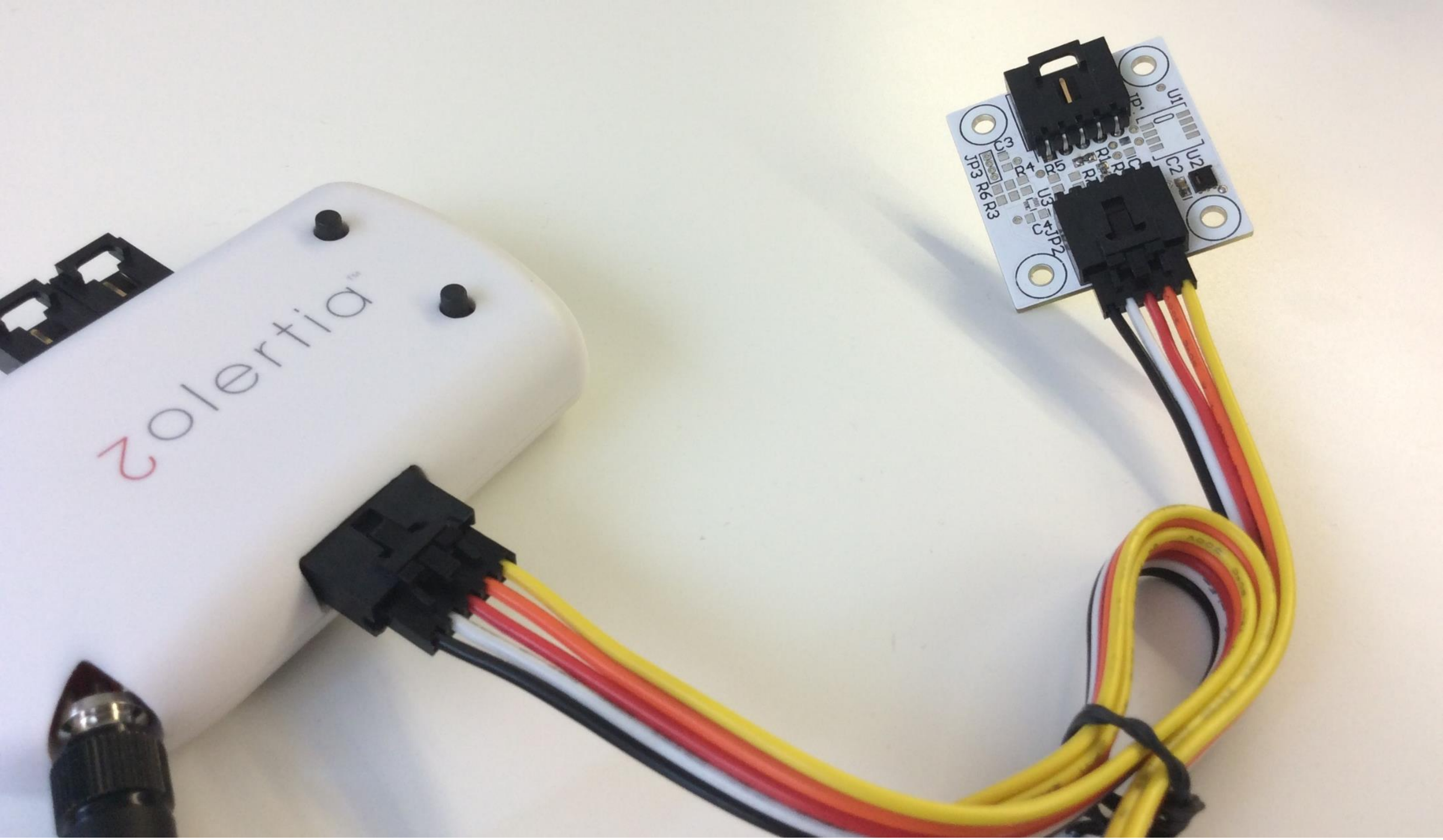
user@iot-workshop: ~/contiki/examples/zolertia/tutorial/01-basics

File Edit View Search Terminal Help

```
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1,
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: External reset
Rime configured with address 00:12:4b:00:06:15:ab:25
 Net: sicslowpan
 MAC: CSMA
 RDC: ContikiMAC
VDD = 3301 mV
Core temperature = 34.47 C
VDD = 3297 mV
Core temperature = 34.285 C
VDD = 3300 mV
```

examples/zolertia/tutorial/01-basics/05-onboard-sensors.c

External digital sensors can be connected to the RE-Mote



https://www.sensirion.com/products/humidity-sensors/humidity-temperature-sensor-sht2x-digital-i2c-accurate/

examples/zolertia/zoul/testsht25.c

SHT21 temperature and humidity sensor (digital, I2C)

Parameter	Value
Sensor type	Temperature and Humidity
Supply Voltage [V]	2.1 - 3.6
Energy Consumption	3.2 uW (at 8 bit, 1 measurement / s)
Max current consumption	**300 uA**
Data range	0-100 % RH (humidity), -40-125°C (temperature)
Max resolution	14 bits (temperature), 12 bits (humidity)

https://www.sensirion.com/products/humidity-sensors/humidity-temperature-sensor-sht2x-digital-i2c-accurate/

platforms/zoul/dev/sht25.c

```
DEFINES+=PROJECT_CONF_H=\"project-conf.h\"

CONTIKI_PROJECT = zoul-demo test-tsl2563 test-sht25 test-power-mgmt
CONTIKI_PROJECT += test-bmp085-bmp180 test-motion test-rotation-sensor
CONTIKI_PROJECT += test-grove-light-sensor test-grove-loudness-sensor
CONTIKI_PROJECT += test-weather-meter test-grove-gyro test-lcd test-iaq
CONTIKI_PROJECT += test-pm10-sensor test-vac-sensor test-aac-sensor
CONTIKI_PROJECT += test-zonik

CONTIKI_TARGET_SOURCEFILES += tsl2563.c sht25.c bmpx8x.c motion-sensor.c
CONTIKI_TARGET_SOURCEFILES += adc-sensors.c weather-meter.c grove-gyro.c
CONTIKI_TARGET_SOURCEFILES += rgb-bl-lcd.c pm10-sensor.c iaq.c zonik.c

all: $(CONTIKI_PROJECT)

CONTIKI = ../../..
include $(CONTIKI)/Makefile.include
```

Add the sensor's library to the application's Makefile

examples/zolertia/zoul/Makefile

```
PROCESS_BEGIN();
SENSORS_ACTIVATE(sht25);

/* Check if the sensor voltage operation is over 2.25V */
if(sht25.value(SHT25_VOLTAGE_ALARM)) {
  printf("Voltage is lower than recommended for the sensor operation\n");
  PROCESS_EXIT();
}

/* Configure the sensor for maximum resolution (14-bit t
 * relative humidity), this will require up to 85ms for
 * integration, and 29ms for the relative humidity (this
 * setting at power on).  To achieve a faster integratio
 * of a lower resolution, change the value below accordi
 */
sht25.configure(SHT25_RESOLUTION, SHT2X_RES_14T_12RH);

/* Let it spin and read sensor data */

while(1) {
  etimer_set(&et, CLOCK_SECOND);
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
  temperature = sht25.value(SHT25_VAL_TEMP);
  printf("Temperature %02d.%02d ºC, ", temperature / 100, temperature % 100);
  humidity = sht25.value(SHT25_VAL_HUM);
  printf("Humidity %02d.%02d RH\n", humidity / 100, humidity % 100);
}
PROCESS_END();
```

```
Starting 'SHT25 test'
Temperature 23.71 ºC
Humidity 42.95 % RH
Temperature 23.71 ºC
Humidity 42.95 % RH
Temperature 23.71 ºC
Humidity 42.95 % RH
Temperature 23.71 ºC
Humidity 42.98  %RH
```

examples/zolertia/zoul/test-sht25.c

```c
static int
configure(int type, int value)
{
  uint8_t buf[2];

  if((type != SHT25_ACTIVE) && (type != SHT25_SOFT_RESET) &&
     (type != SHT25_RESOLUTION)) {
    PRINTF("SHT25: option not supported\n");
    return SHT25_ERROR;
  }

  switch(type) {
  case SHT25_ACTIVE:
    if(value) {
      i2c_init(I2C_SDA_PORT, I2C_SDA_PIN, I2C_SCL_PORT, I2C_SCL_PIN,
               I2C_SCL_NORMAL_BUS_SPEED);

      /* Read the user config register */
      if(sht25_read_user_register() == SHT25_SUCCESS) {
        enabled = value;
        return SHT25_SUCCESS;
      }
    }
```

When enabling the sensor, the I2C is initialized and the sensor configuration loaded

platforms/zoul/dev/sht25.c

```
case SHT25_SOFT_RESET:
  buf[0] = SHT2X_SOFT_RESET;
  if(sht25_write_reg(&buf[0], 1) != SHT25_SUCCESS) {
    PRINTF("SHT25: failed to reset the sensor\n");
    return SHT25_ERROR;
  }
  clock_delay_usec(SHT25_RESET_DELAY);
  return SHT25_SUCCESS;

case SHT25_RESOLUTION:
  if((value != SHT2X_RES_14T_12RH) && (value != SHT2X_RES_12T_08RH) &&
     (value != SHT2X_RES_13T_10RH) && (value != SHT2X_RES_11T_11RH)) {
    PRINTF("SHT25: invalid resolution value\n");
    return SHT25_ERROR;
  }

  user_reg &= ~SHT2X_RES_11T_11RH;
  user_reg |= value;
  buf[0] = SHT2X_UREG_WRITE;
  buf[1] = user_reg;

  if(sht25_write_reg(buf, 2) == SHT25_SUCCESS) {
    PRINTF("SHT25: new user register value 0x%02X\n", user_reg);
    return SHT25_SUCCESS;
  }

default:
  return SHT25_ERROR;
}

return SHT25_ERROR;
```

The soft reset command will restart the sensor in case of failure

The resolution can be configured for both the temperature and humidity sensors

platforms/zoul/dev/sht25.c

```c
/*---------------------------------------------------------------------------*/
static uint16_t
sht25_read_reg(uint8_t reg, uint8_t *buf, uint8_t num)
{
  if((buf == NULL) || (num <= 0)) {
    return SHT25_ERROR;
  }

  i2c_master_enable();
  if(i2c_single_send(SHT25_ADDR, reg) == I2C_MASTER_ERR_NONE) {
    if(i2c_burst_receive(SHT25_ADDR, buf, num) == I2C_MASTER_ERR_NONE) {
      return SHT25_SUCCESS;
    }
  }
  return SHT25_ERROR;
}
```

The I2C API is used to read and write to the sensor's registers

```c
static int16_t
sht25_convert(uint8_t variable, uint16_t value)
{
  int16_t rd;
  uint32_t buff;

  /* Clear the status bits */
  buff = (uint32_t)(value & ~SHT25_STATUS_BITS_MASK);

  if(variable == SHT25_VAL_TEMP) {
    buff *= 17572;
    buff = buff >> 16;
    rd = (int16_t)buff - 4685;
  } else {
    buff *= 12500;
    buff = buff >> 16;
    rd = (int16_t)buff - 600;
    rd = (rd > 10000) ? 10000 : rd;
  }
  return rd;
}
```

The sensor's values are converted into readable units: Celsius and Relative Humidity, with 2 precision digits

platforms/zoul/dev/sht25.c

Actuators

An actuator is a device that allow us to interact with the physical world, and change the state of a given variable. A light switch turns a light bulb on and off, an electrovalve can control the flow of water, an electronic lock can open or shut a door

General Input/Output Pins (GPIO)

The General input/output pins are generic pins used either as input or output (0-3.3V), useful in case you need to actuate on something, or read the digital voltage level as high/low (3.3V or 0V).

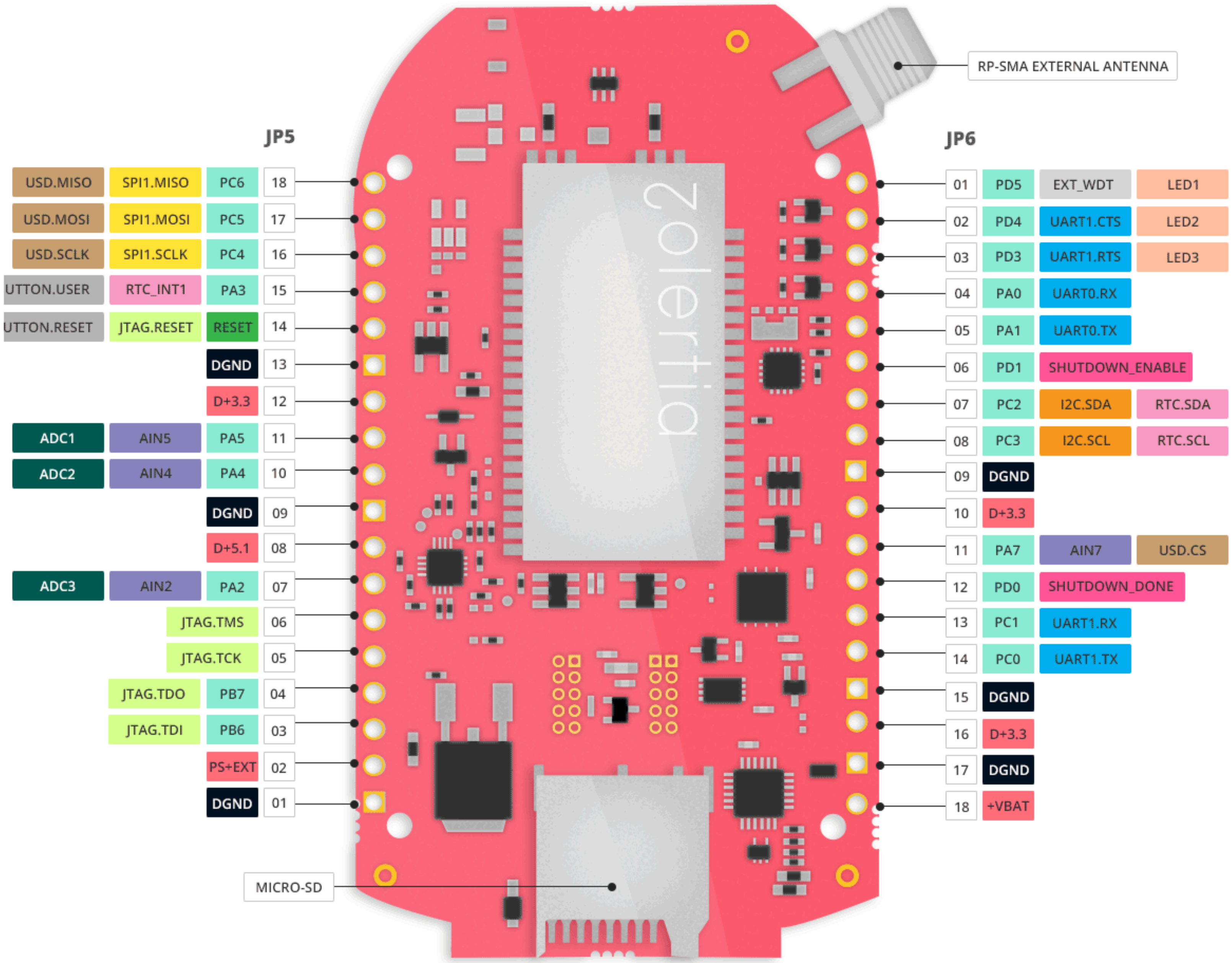
Normally the electrovalves, relays, switchs are bi-state, so a GPIO pin can be used to control.

The LEDs are driven by GPIOs configured as output, changing its value from high/low turns the LEDs on/off (depending if used a pull-up or pull-down resistors).

The user button previously shown is a GPIO configured as input, when pressed its value changed, and its status is read by the MCU.

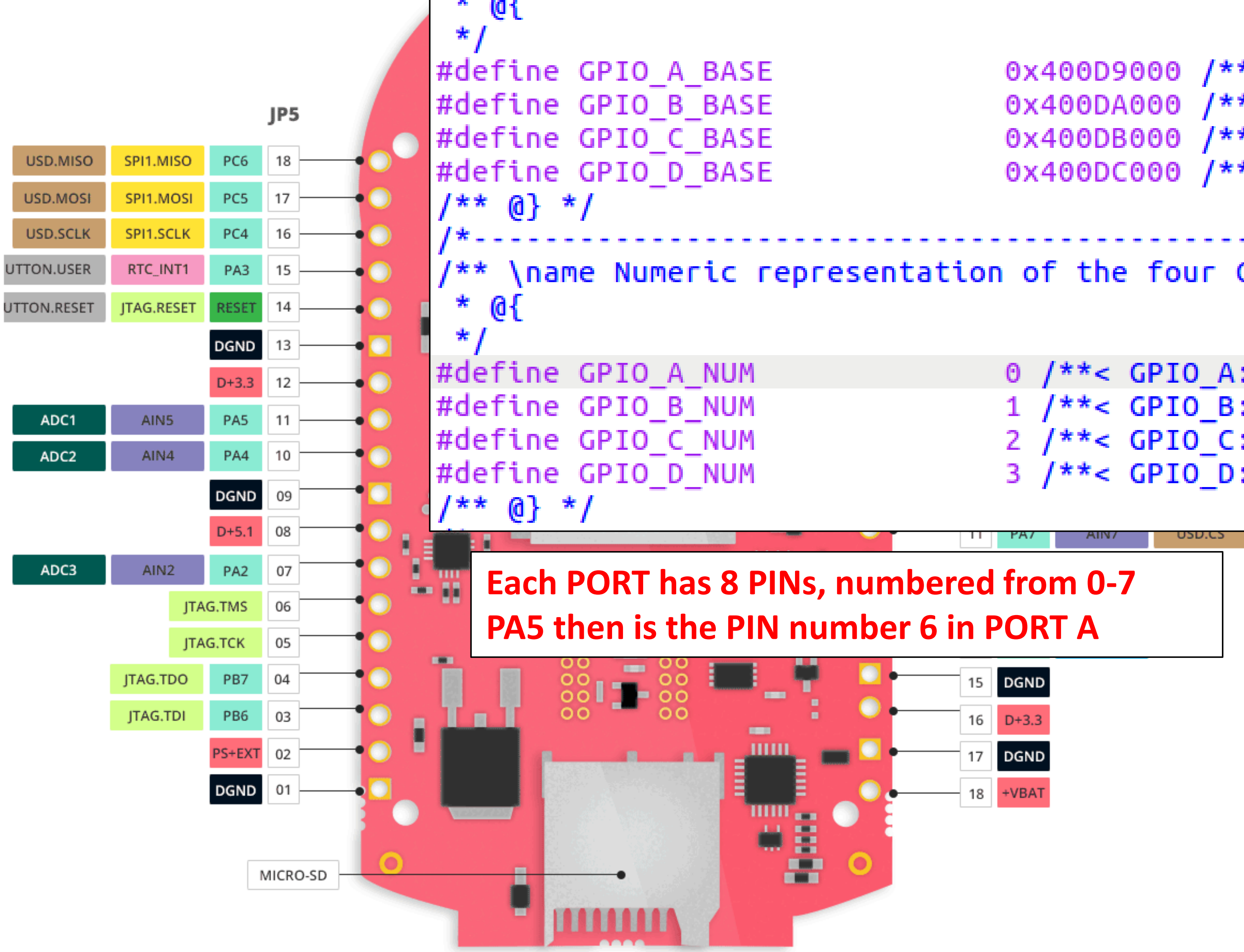
examples/zolertia/tutorial/01-basics

Antonio Liñán, Zolertia. 2016 - CC-NC-SA 4.0



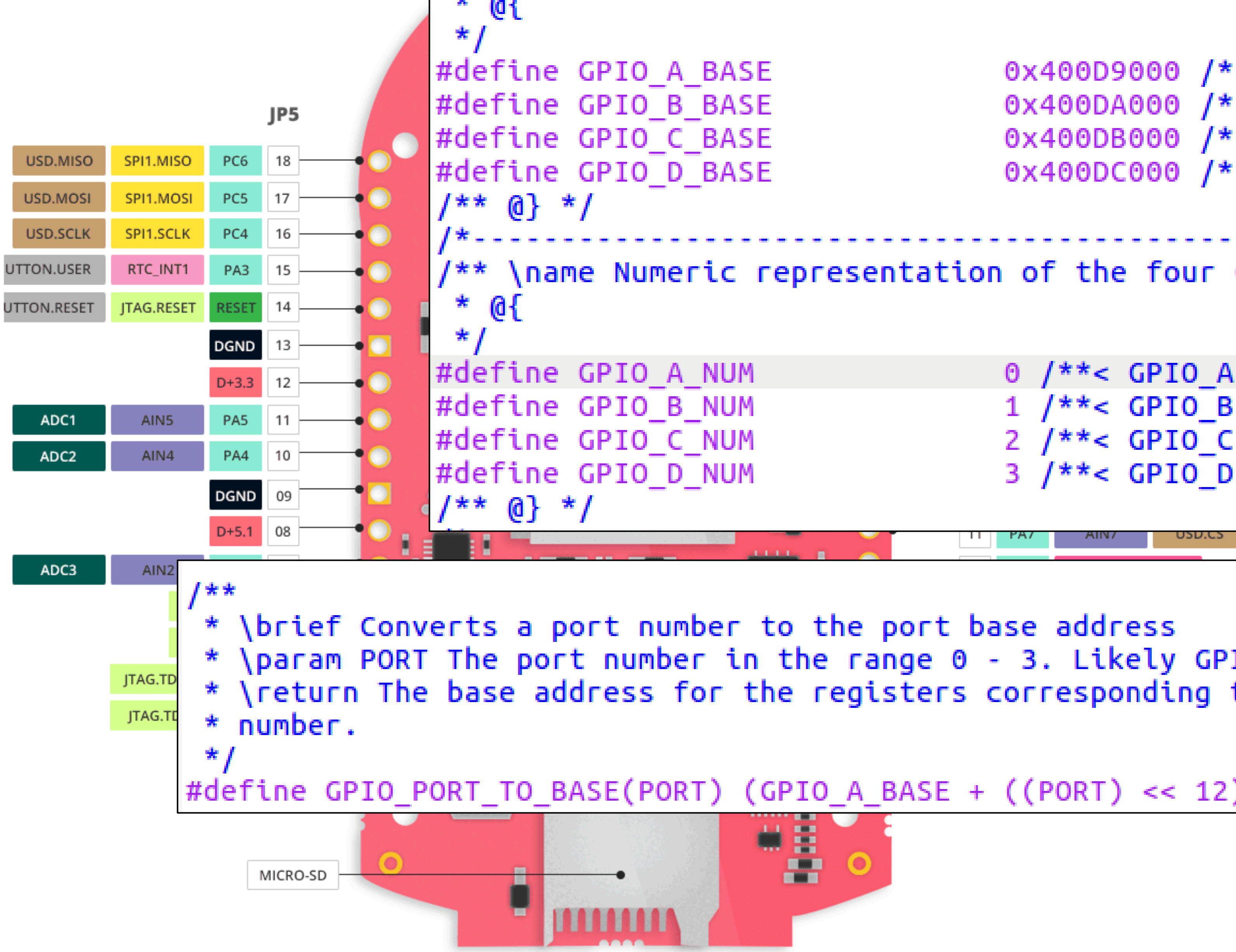
examples/zolertia/tutorial/01-basics

```
/** \name Base addresses for the GPIO register instances
  * @{
  */
#define GPIO_A_BASE              0x400D9000 /**< GPIO_A */
#define GPIO_B_BASE              0x400DA000 /**< GPIO_B */
#define GPIO_C_BASE              0x400DB000 /**< GPIO_C */
#define GPIO_D_BASE              0x400DC000 /**< GPIO_D */
/** @} */
/*---------------------------------------------------------------
/** \name Numeric representation of the four GPIO ports
  * @{
  */
#define GPIO_A_NUM               0 /**< GPIO_A: 0 */
#define GPIO_B_NUM               1 /**< GPIO_B: 1 */
#define GPIO_C_NUM               2 /**< GPIO_C: 2 */
#define GPIO_D_NUM               3 /**< GPIO_D: 3 */
/** @} */
```



Each PORT has 8 PINs, numbered from 0-7
PA5 then is the PIN number 6 in PORT A

examples/zolertia/tutorial/01-basics




```c
/** \name Base addresses for the GPIO register instances
 * @{
 */
#define GPIO_A_BASE              0x400D9000 /**< GPIO_A */
#define GPIO_B_BASE              0x400DA000 /**< GPIO_B */
#define GPIO_C_BASE              0x400DB000 /**< GPIO_C */
#define GPIO_D_BASE              0x400DC000 /**< GPIO_D */
/** @} */
/*---------------------------------------------------------------
/** \name Numeric representation of the four GPIO ports
 * @{
 */
#define GPIO_A_NUM               0 /**< GPIO_A: 0 */
#define GPIO_B_NUM               1 /**< GPIO_B: 1 */
#define GPIO_C_NUM               2 /**< GPIO_C: 2 */
#define GPIO_D_NUM               3 /**< GPIO_D: 3 */
/** @} */
```

```c
/**
 * \brief Converts a port number to the port base address
 * \param PORT The port number in the range 0 - 3. Likely GPIO_X_NUM.
 * \return The base address for the registers corresponding to that port
 * number.
 */
#define GPIO_PORT_TO_BASE(PORT) (GPIO_A_BASE + ((PORT) << 12))
```

examples/zolertia/tutorial/01-basics

GPIOs in a PORT can be accessed via masks, as each PORT has 8 pins, it is similar to an 8-bit variable

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	1	0	1	0	0

In binary : 00010100
In hexadecimal : 0x14
In decimal : 20

If this were the pins in PORT B, it would mean PA2 and PA4 can be configured in a single operation

PIN 3 mask value would be (1 << 3), the equivalent to 8

```
/**
 * \brief Converts a pin number to a pin mask
 * \param PIN The pin number in the range [0..7]
 * \return A pin mask which can be used as the PIN_MASK argument of the macros
 * in this category
 */
#define GPIO_PIN_MASK(PIN) (1 << (PIN))
```

```
/* The masks below converts the Port number and Pin number to base and mask
#define EXAMPLE_PORT_BASE  GPIO_PORT_TO_BASE(GPIO_A_NUM)
#define EXAMPLE_PIN_MASK   GPIO_PIN_MASK(5)

/* We tell the system the application will drive the pin */
GPIO_SOFTWARE_CONTROL(EXAMPLE_PORT_BASE, EXAMPLE_PIN_MASK);

/* And set as output, starting low */
GPIO_SET_OUTPUT(EXAMPLE_PORT_BASE, EXAMPLE_PIN_MASK);
GPIO_SET_PIN(EXAMPLE_PORT_BASE, EXAMPLE_PIN_MASK);
```

```
while(1) {
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

  if(GPIO_READ_PIN(EXAMPLE_PORT_BASE, EXAMPLE_PIN_MASK)) {
    GPIO_CLR_PIN(EXAMPLE_PORT_BASE, EXAMPLE_PIN_MASK);
  } else {
    GPIO_SET_PIN(EXAMPLE_PORT_BASE, EXAMPLE_PIN_MASK);
  }

  leds_toggle(LEDS_GREEN);
  etimer_reset(&et);
}

PROCESS_END();
```

examples/zolertia/tutorial/01-basics/07-gpio.c

Change the value of the PIN in the example, and toggle the LED3 (see in the previous slides which PORT/PIN values corresponds to it)

If available, use a multimeter to measure the voltage values in the pins when high and low.

If available, connect a relay and try to turn on and off a lamp

examples/zolertia/tutorial/01-basics/07-gpio.c

Conclusions

You should be able to:

- Understand how analogue sensors and ADC works
- Implement an analogue sensor, convert ADC count to voltaje values, and convert to actual unit values
- Measure the core temperature and voltaje of the RE-Mote platform
- Understand Contiki's sensor's API
- Understand how digital sensor works
- Use the grove's analogue light sensor (P), and SHT21 digital temperature and humidity sensor
- Understand how GPIO works and the pin distribution on the RE-Mote platform
- Implement an actuator using GPIOs

Antonio Liñán Colina

alinan@zolertia.com
antonio.lignan@gmail.com

Twitter: @4Li6NaN
LinkedIn: Antonio Liñan Colina
github.com/alignan
hackster.io/alinan