# Contiki: Basic Programming

**1st Workshop on Appl. of WSN for Env. Monitoring in Developing Countries**

Trieste (Italy), 02.03.2011

# Contiki: Basic Programming

- **Programming Examples**
  - Hello world! (I)
  - Timers
  - Hello world! (II)
  - Using the on-board LEDs
  - Reading data from the sensors

# Hello World! (I)

- **hello_world.c**

```
#include "contiki.h"
#include <stdio.h>

...
printf("Hello world!\n");
...
```

**contiki.h needs to be always included!**

**Include stdio.h to be able to use the printf**

- One must include the definition file of any device / code used by our application
- Radio, LEDs, Sensors, File System, …

# Hello World! (I)

## ➢ hello_world.c

```
#include "contiki.h"
#include <stdio.h>

PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        ...
        printf("Hello world!\n");
        ...
}
```

**Definition of the process (in the header of a program)**

**Processes to be loaded at boot time**

**Specific code of the process**

- An application can host more than one process
- PROCESS(name, strname)
- PROCESS_THREAD(name, ev, data)

# Hello World! (I)

## ➢ hello_world.c

```
#include "contiki.h"
#include <stdio.h>

PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        printf("Hello world!\n");
        PROCESS_END();
}
```

Specify an action when a process exits (cleanup code)

Always delimit the beginning and the end of a process!

# Hello World! (I)

## ➢ hello_world.c

```
#include "contiki.h"
#include <stdio.h>
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        printf("Hello world!\n");
        while(1) {
            PROCESS_WAIT_EVENT();
        }
        PROCESS_END();
}
```

**Unbounded wait: if no event comes, the program remains stuck here forever**

# Hello World! (I)

## ➢ hello_world.c

```
#include "contiki.h"
#include <stdio.h>
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        printf("Hello world!\n");
        while(1) {
            PROCESS_WAIT_EVENT();
        }
        PROCESS_END();
}
```

```
Boano@PC-Boano ~
$ ./contiki-2.x/tools/sky/serialdump.exe -t -b115200 /dev/com15
connecting to /dev/com15 (115200) [OK]
1269440398'647|Rime started with address 153.37
1269440398'704|MAC 99:25:00:00:00:00:00:00 Contiki 2.4 started.
1269440398'723|nullmac sicslowmac, channel check rate 128 Hz, channel 26
1269440398'739|Starting 'Hello world process'
1269440398'751|Hello world!
```

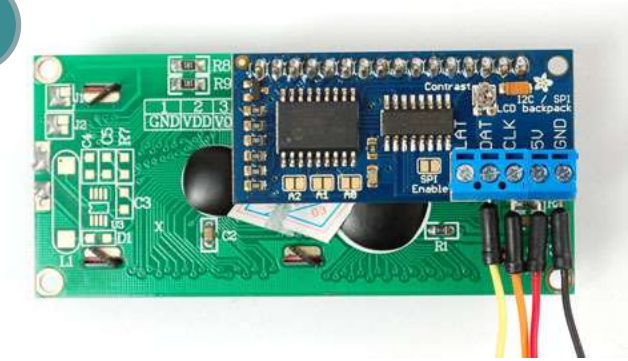
# Contiki: Basic Programming

- **Programming Examples**
  - Hello world! (I)
  - Timers
  - Hello world! (II)
  - Using the on-board LEDs
  - Using the on-board button sensor
  - Reading data from the sensors

# Contiki: Basic Programming

- **Contiki has an Event-driven Kernel**
  - Three types of events
    1. Timer events
    2. External events
    3. Internal events
  - Event information
    - Process
    - Event type
    - Data

2



**Peripheral devices with interrupt capabilities connected to I/O pins**

1

**Timer events**

3

**Interprocess communication**

# Contiki Timers

- **Three types of timers**
  - Passive timer
    - ***struct timer*** (only keeps track of its expiration)
  - Active timer
    - ***struct etimer*** (sends an event when it expires)
    - ***struct ctimer*** (calls a function when it expires)
  - Real-time timer
    - ***struct rtimer*** (calls a function at an exact time)
    - Typically used for accurate time measurements

Check: contiki\core\sys\Xtimer.h

# Contiki Timers

- **etimer**
  - Timer declaration
    ```
    static struct etimer et;
    ```
  - Setting/Stopping the timer
    ```
    etimer_set(&et, AMOUNT_OF_TIME);
    etimer_stop(&et);
    ```
  - Resetting the timer
    ```
    etimer_reset(&et);
    ```
  - Other operations
    ```
    etimer_adjust(&et, AMOUNT_OF_TIME);
    etimer_pending();  ← Checks if there is any non-expired event
    clock_time_t  next_expiration_time();
    ```

**CLOCK_SECOND is the reference time for one second**
(#define CLOCK_CONF_SECOND in platform-conf.h)

Check: contiki\core\sys\etimer.h

# Contiki Timers

- **etimer**
  - Important notes
    - Divide/multiply CLOCK_SECOND by multiples of two
    - Do not use the etimer in interrupt handlers (it posts events!)

Check: contiki\core\sys\etimer.h

# Contiki Timers

- **ctimer**
  - Defines a callback function to be called when timer expires
  - Timer and callback definition

```
static struct ctimer timer1;
static void ctimer1_callback(void *ptr);
```

  - Definition of the callback function

```
static void ctimer1_callback(void *ptr) {
        …
}
```

  - Activating/Deactivating the timer

```
ctimer_set(&timer1, AMOUNT_OF_TIME, ctimer1_callback, NULL);
ctimer_stop(&timer1);
```

Check: contiki\core\sys\ctimer.h

# Contiki Timers

- **rtimer**
  - Counts the amount of ticks
  - Useful for accurate time measurements
  - Frequency depends on the clock used
  - Timer declaration

```
rtimer_clock_t now;
```

  - Instruction delay measurement

```
now = RTIMER_NOW();
/* Instructions to be measured here */
now = (RTIMER_NOW() – now);
printf("duration = %d/%d\n", now, RTIMER_SECOND);
```

**Expect an overflow if you need to measure several seconds!**

Check: contiki\core\sys\rtimer.h

# Contiki: Basic Programming

- **Programming Examples**
  - Hello world! (I)
  - Timers
  - Hello world! (II)
  - Using the on-board LEDs
  - Using the on-board button sensor
  - Reading data from the sensors

# Hello World! (II)

## ➢ hello_world_reloaded.c

- We want to print “Hello world!” every two seconds continuously

```
#include "contiki.h"
#include <stdio.h>

PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();

        ...

        PROCESS_END();
}
```

?

# Hello World! (II)

## ➢ hello_world_reloaded.c

- We want to print “Hello world!” every two seconds continuously

```
#include "contiki.h"
#include <stdio.h>
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        static struct etimer et;
        while(1) {
                etimer_set(&et, (2*CLOCK_SECOND));
                printf(“Hello world!\n”);
        }
        PROCESS_END();
}
```

**We add an etimer that fires every two seconds**

**Are we done?**

# Hello World! (II)

## ➢ hello_world_reloaded.c

- We want to print “Hello world!” every two seconds continuously

```
#include "contiki.h"
#include <stdio.h>
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        static struct etimer et;
        while(1) {
                etimer_set(&et, (2*CLOCK_SECOND));
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
                printf(“Hello world!\n”);
        }
        PROCESS_END();
}
```

**We need to wait for the timer to expire or the program will loop continuously!**

# Hello World! (II)

## ➢ hello_world_reloaded.c

- We want to print “Hello world!” every two seconds continuously

```
#include "contiki.h"
#include <stdio.h>
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
PROCESS_THREAD(hello_world_process, ev, data)
{
        PROCESS_EXITHANDLER()
        PROCESS_BEGIN();
        static struct etimer et;
        while(1) {
                etimer_set(&et, (2*CLOCK_SECOND));
                PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
                printf(“Hello world!\n”);
        }
        PROCESS_END();
}
```

```
Boano@PC-Boano ~
$ ./contiki-2.x/tools/sky/serialdump.exe -t -b115200 /dev/com15
connecting to /dev/com15 (115200) [OK]
1269442461'525|Rime started with address 153.37
1269442461'532|MAC 99:25:00:00:00:00:00:00 Contiki 2.4 started.
1269442461'565|nullmac sicslowmac, channel check rate 128 Hz
1269442461'584|Starting 'Hello world process'
1269442462'546|Hello world!
1269442463'560|Hello world!
1269442464'548|Hello world!
```

# Contiki: Basic Programming

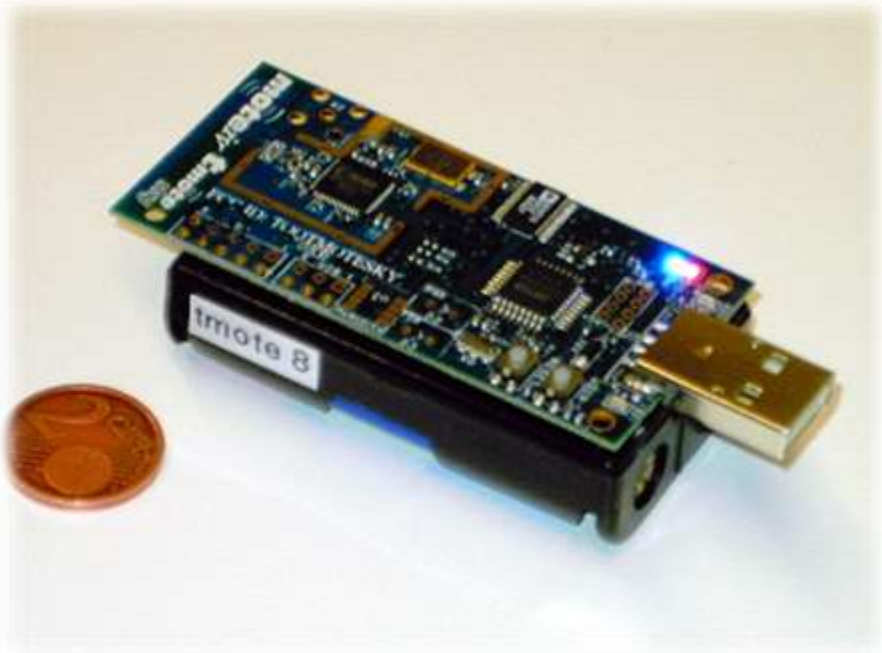
- **Programming Examples**
  - Hello world! (I)
  - Timers
  - Hello world! (II)
  - Using the on-board LEDs
  - Using the on-board button sensor
  - Reading data from the sensors

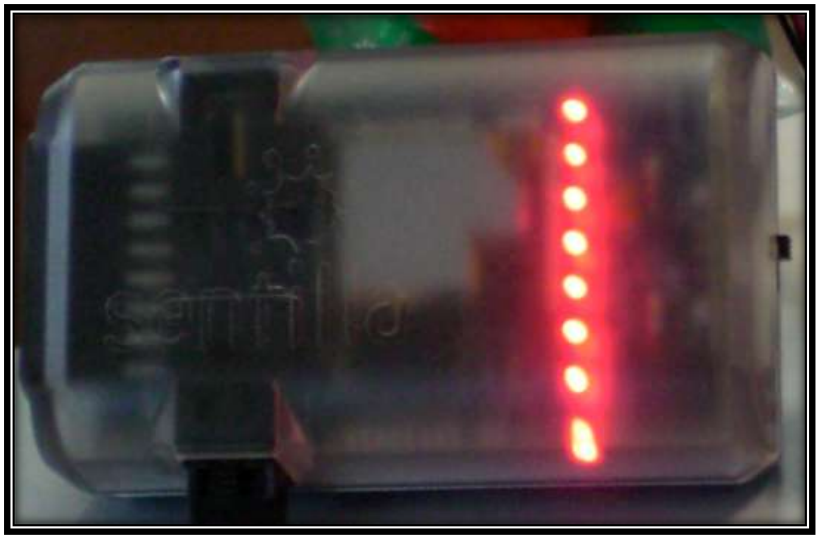
# Using the on-board LEDs

- **What is a LED?**
  - Light Emitting Diode
  - Semiconductor light source
  - Typically used as indicator signals/lamps
  - Sensor nodes typically have many
    - Debugging, testing, …

# Using the on-board LEDs

- **Sentilla Tmote Sky vs Sentilla JCreate**
  - Sentilla Tmote Sky has 3 LEDs (Red, Blue, Green)
  - The Sentilla JCreate has 8 red LEDs

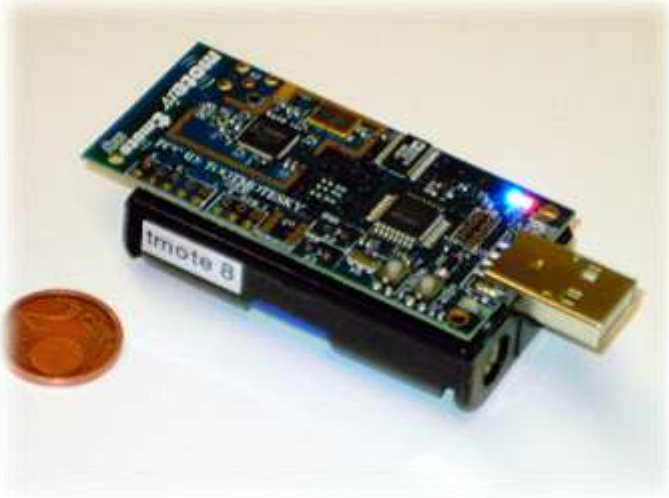


# Using the on-board LEDs

- **Sentilla Tmote Sky and Zolertia Z1**
  - Primitives

    leds_on(unsigned char leds);
    leds_off(unsigned char leds);
    leds_toggle(unsigned char leds);
    …




  - Available LEDs defined in the specification file

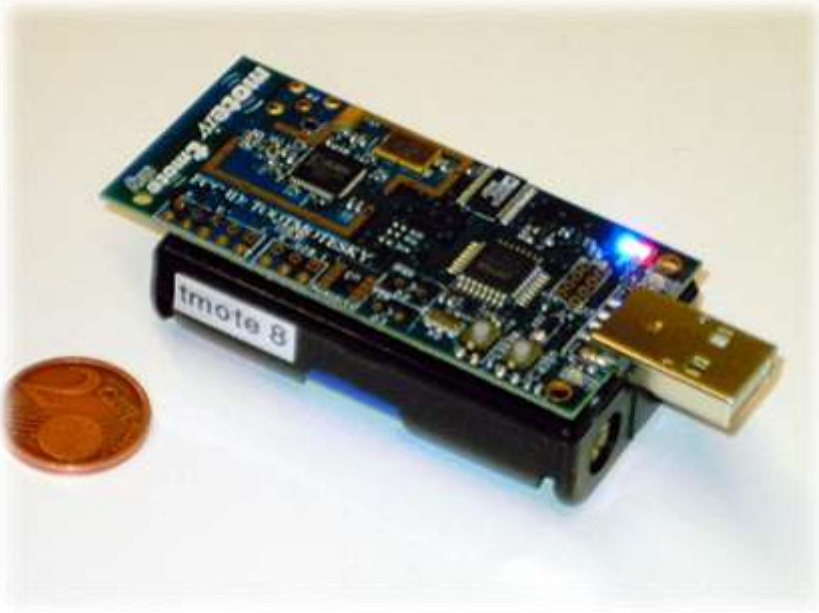
```
#define LEDS_GREEN 1
#define LEDS_YELLOW 2
#define LEDS_RED 4
#define LEDS_BLUE LEDS_YELLOW
#define LEDS_ALL 7
```

**#include "dev/leds.h"**

Check: contiki\core\dev\leds.h

# Using the on-board LEDs

- **Sentilla Tmote Sky and Zolertia Z1**
  - Switching on the red LED only
    leds_on(LEDS_RED);
  - Switching off the red LED only
    leds_off(LEDS_RED);
  - Switching on all the LEDs
    leds_on(LEDS_ALL);
  - Switching off all the LEDs
    leds_off(LEDS_ALL);
  - …




Check: contiki\core\dev\leds.h

# Using the on-board LEDs

- **Sentilla JCreate**
  - Use the sentilla_leds.h and sentilla_leds.c files
  - They provide a set of basic API to access the LEDs
  - Must be included in the main file and in the makefile
  - Primitives:

```
// Showing a binary number on the LEDs
static void sentilla_led_binary(int num);

// LEDs initialization
void sentilla_leds_init();

// Turning on or off a specific LED
void sentilla_led_number(int led_number, int onoff);
```

Check: contiki\core\dev\leds.h

# Using the on-board LEDs

## ➢ sentilla_leds.h

```
#include "contiki.h"
#include "dev/leds.h"

// LEDs initialization
void sentilla_leds_init();

// Display a binary number on the LEDs
// Example: sentilla_led_binary(10); shows 0x0A on the LEDs
void sentilla_led_binary(int number);

// Turning on or off a specific LED
// Example: sentilla_led_number(5,0); // Turns off led number 5
// Example: sentilla_led_number(5,1); // Turns on led number 5
void sentilla_led_number(int led_number, int onoff);
```

## ➢ sentilla_leds.c

```
#include "sentilla_leds.h“
void sentilla_led_binary(int number) {
  LEDS_PxOUT = ~number;
}
void sentilla_leds_init() {
  LEDS_PxDIR = 0xff;
  LEDS_PxOUT = 0xff;
}
void sentilla_led_number(int led_number, int onoff) {
  int val = (1 << led_number);
  if(onoff == 1) {
                LEDS_PxOUT &= ~val;
  }
  else {
                LEDS_PxOUT |= val;
  }
}
```

Check: contiki\core\dev\leds.h

# Using the on-board LEDs

- **Displaying a binary number with the LEDs in the JCreate**
  - introduction_display_binary.c

```
#include "contiki.h"
#include "sentilla_leds.h" // Needed to use the LEDs
PROCESS(introduction_display_binary, "Display a binary number");
AUTOSTART_PROCESSES(&introduction_display_binary);
PROCESS_THREAD(introduction_display_binary, ev, data) {
   PROCESS_EXITHANDLER()
   PROCESS_BEGIN();
    sentilla_leds_init(); // Initializing the LEDs
    static struct etimer et; // Definition of the timer
    int num = 0;
    while(1) {
         etimer_set(&et, CLOCK_SECOND/16); // Setting the etimer to be expired in 62.5 milliseconds
         PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et)); // Wait for the etimer expiration
         sentilla_led_binary(num++); // Displaying a binary number on the LEDs
         if(num >= 255) num = 0; // If we exceed the maximum number, restart from 0
    }
   PROCESS_END();
}
```

**There is an error in this program…**

Check: contiki\core\dev\leds.h

# Using the on-board LEDs

## ➢ Displaying a binary number with the LEDs in the JCreate

- introduction_display_binary.c

**Remember the protothread limitations!**

```
#include "contiki.h"
#include "sentilla_leds.h" // Needed to use the LEDs
int num = 0; // Global variable
PROCESS(introduction_display_binary, "Display a binary number");
AUTOSTART_PROCESSES(&introduction_display_binary);
PROCESS_THREAD(introduction_display_binary, ev, data) {
    PROCESS_EXITHANDLER()
    PROCESS_BEGIN();
     sentilla_leds_init(); // Initializing the LEDs
     static struct etimer et; // Definition of the timer
     while(1) {
          etimer_set(&et, CLOCK_SECOND/16); // Setting the etimer to be expired in 62.5 milliseconds
          PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et)); // Wait for the etimer expiration
          sentilla_led_binary(num++); // Displaying a binary number on the LEDs
          if(num >= 255) num = 0; // If we exceed the maximum number, restart from 0
     }
    PROCESS_END();
}
```

Check: contiki\core\dev\leds.h

# Using the on-board LEDs

## ➢ Playing with the LEDs with the Sentilla Jcreate

- KITT car (Knight Rider, 1982 TV series)

```
#include "contiki.h"
#include "net/rime.h" // Needed to use the ctimers
#include "sentilla_leds.h" // Needed to use the LEDs
#define PERIOD (CLOCK_SECOND/16) // Defining timer period
// Definition of the timer
static struct ctimer timer1;
static void ctimer1_callback(void *ptr);
// Global variables
int num = 1, direction = 0;
// Timer callback: changing the LEDs
static void ctimer1_callback(void *ptr) {
   sentilla_led_binary(num); // Displaying a binary number on the LEDs
   if(direction == 0)
        num = num << 1;
   else
        num = num >> 1;
   if(num == 128) direction = 1;
   if(num == 1) direction = 0;
   ctimer_set(&timer1, PERIOD, ctimer1_callback, NULL); // Important!
}
```

```
PROCESS(introduction_kitt, "KITT!");
AUTOSTART_PROCESSES(&introduction_kitt);
PROCESS_THREAD(introduction_kitt, ev, data) {
  PROCESS_EXITHANDLER()
  PROCESS_BEGIN();
   sentilla_leds_init(); // Initializing the LEDs
   // Setting the timer
   ctimer_set(&timer1, PERIOD, ctimer1_callback, NULL);
   while(1) {
     PROCESS_WAIT_EVENT();
   }
  PROCESS_END();
}
```

# Contiki: Basic Programming

- **Programming Examples**
  - Hello world! (I)
  - Timers
  - Hello world! (II)
  - Using the on-board LEDs
  - Using the on-board button sensor
  - Reading data from the sensors

# Using the On-board Button Sensor

- **A button generates an interrupt event**
  - Typically close to the reset button
  - Not available in the Sentilla JCreate ☹
  - Available in the Tmote Sky and Z1 platforms
  - Having access to a reset button is sometimes vital for debugging and testing

    **make sky-reset && make login**

    **make z1-reset && make login**

**User Button**

**Tip: you can simulate the user button with MSPsim**
(make TARGET=sky application_name.mspsim)

Check: contiki\core\dev\ button-sensor.h

# Using the On-board Button Sensor

## ➢ Example: toggling the LEDs when pressing the user button

```
#include "contiki.h"
#include "dev/button-sensor.h"
#include "dev/leds.h“
#include <stdio.h>

PROCESS(example_button, "Example with user-button");
AUTOSTART_PROCESSES(&example_button);
PROCESS_THREAD(example_button, ev, data) {
  PROCESS_EXITHANDLER()
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(button_sensor);
   while(1) {
     PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);
     printf("button pressed\n");
     leds_toggle(LEDS_ALL);
   }
  PROCESS_END();
}
```

Check: contiki\core\dev\ button-sensor.h
Check: contiki\examples\sky\test-button.c

# Contiki: Basic Programming

- **Programming Examples**
  - Hello world! (I)
  - Timers
  - Hello world! (II)
  - Using the on-board LEDs
  - Using the on-board button sensor
  - Reading data from the sensors

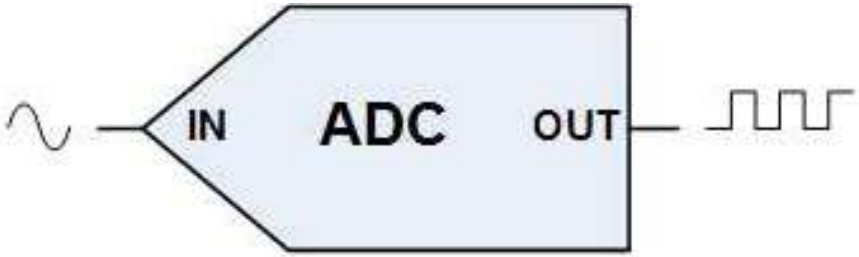
# Reading Data from the Sensors

- **Which sensors are typically available?**
  - Analog and Digital sensors
  - Sentilla Tmote Sky
    - Temperature and Humidity (Sensirion SHT11)
    - Light sensor (Hamamatsu Corp.)
    - Battery and temperature sensors (via MSP430)
  - Sentilla JCreate and Zolertia Z1
    - Battery and temperature sensor s
    - Triaxial accelerometer

# Reading Data from the Sensors

## ➢ Accessing an analog sensor

- Typical sequence of steps to be performed
  1. Activate the sensor and get the raw value from the ADC
  2. Derive the voltage that was converted as digital number (requires knowledge of voltage reference and ADC resolution)
  3. Derive from the measured voltage the physical quantity
  4. [Perform calibration]

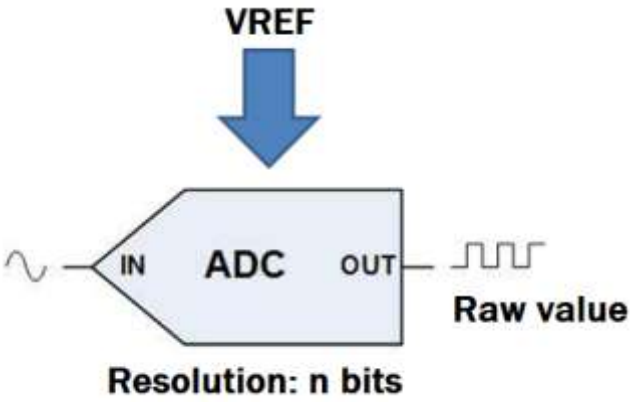


# Reading Data from the Sensors

## ➢ Accessing an analog sensor

- Typical sequence of steps to be performed
  1. Activate the sensor and get the raw value from the ADC
  2. Derive the voltage that was converted as digital number (requires knowledge of voltage reference and ADC resolution)
  3. Derive from the measured voltage the physical quantity
  4. [Perform calibration]

$$IN\ [V] = (OUT*VREF)/2^n$$



# Reading Data from the Sensors

- **Example of analog sensor: MSP430's battery voltage**
  - Zolertia Z1 platform

```
PROCESS(application, "Test application");
AUTOSTART_PROCESSES(&application);
PROCESS_THREAD(application, ev, data)
{
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(battery_sensor);
  while (1) {
    …
  }
  SENSORS_DEACTIVATE(battery_sensor);
  PROCESS_END();
}
```

Activate the sensor

# Reading Data from the Sensors

- **Example of analog sensor: MSP430's battery voltage**
  - Zolertia Z1 platform

```
PROCESS(application, "Test application");
AUTOSTART_PROCESSES(&application);
PROCESS_THREAD(application, ev, data)
{
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(battery_sensor);
  while (1) {
    uint16_t raw_value = battery_sensor.value(0);
    …
  }
  SENSORS_DEACTIVATE(battery_sensor);
  PROCESS_END();
}
```

Get the raw value from the A/D Converter

# Reading Data from the Sensors

- **Example of analog sensor: MSP430's battery voltage**
  - Zolertia Z1 platform

```
PROCESS(application, "Test application");
AUTOSTART_PROCESSES(&application);
PROCESS_THREAD(application, ev, data)
{
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(battery_sensor);
  while (1) {
    uint16_t raw_value = battery_sensor.value(0);
    float voltage = (raw_value*2.500*2)/4096;
    printf("Battery: %i (%ld.%03d V)\n", raw_value, (long) voltage, (unsigned) ((voltage -floor(voltage))*1000));
  }
  SENSORS_DEACTIVATE(battery_sensor);
  PROCESS_END();
}
```

Derive the voltage

# Reading Data from the Sensors

- **Example of analog sensor: MSP430's battery voltage**
  - Zolertia Z1 platform

```
#include "contiki.h"
#include "dev/battery-sensor.h"
#include <stdio.h> /* For printf() */
#include <math.h> /* For floor() */
```

```
PROCESS(application, "Test application");
AUTOSTART_PROCESSES(&application);
PROCESS_THREAD(application, ev, data)
{
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(battery_sensor);
  while (1) {
    uint16_t raw_value = battery_sensor.value(0);
    float voltage = (raw_value*2.500*2)/4096;
    printf("Battery: %i (%ld.%03d V)\n", raw_value, (long) voltage, (unsigned) ((voltage -floor(voltage))*1000));
  }
  SENSORS_DEACTIVATE(battery_sensor);
  PROCESS_END();
}
```

```
float floor(float x){
  if(x>=0.0f) return (float) ((int)x);
  else return(float)((int)x-1);
}
```

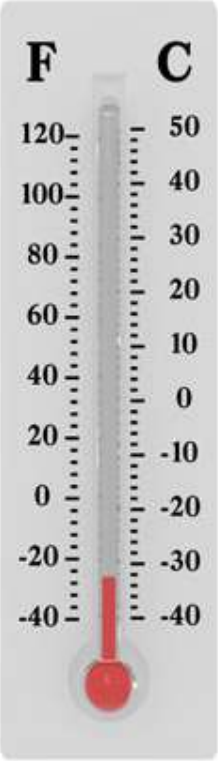
To add support for the Math library, add in the end of the makefile:

```
CUSTOM_RULE_LINK=1
%.$(TARGET): %.co $(PROJECT_OBJECTFILES) $(PROJECT_LIBRARIES) contiki-$(TARGET).a
$(LD) $(LDFLAGS) $(TARGET_STARTFILES) ${filter-out %.a,$^} ${filter %.a,$^} $(TARGET_LIBFILES) -o $@ -lm
```

# Reading Data from the Sensors

- **Measuring temperature with the Sentilla JCreate**
  - Skeleton

```
// Activate the sensor
… (see slide 43)
// Get the raw value
… (see slide 43)
// Get the reference voltage and ADC resolution
… check (ADC12CTL0 & REF2_5V) …
// Convert the raw value to the input voltage of the ADC
…
// Convert the voltage to temperature
(see MSP430 datasheet)
```




Check: MSP430 datasheet, pag 359

# Reading Data from the Sensors

## ➢ Accessing a digital sensor

- Usually a typical sequence of steps to be performed
  1. Connecting to the digital pins ($I^2C$, SMBus, ...)
  2. Convert the raw value to the physical quantity
  3. Calibrate the reading

- Getting the raw temperature from the SHT11 (Tmote Sky)
  - Typical primitives

    **sht11_init();**
    **sht11_temp();**
    **sht11_humidity();**

  - Raw value conversion

    **Relative Humidity: RH = -4 + 0.0405 * raw - raw$^2$ * 2.8e$^{-6}$**
    **Temperature in Celsius: T = -39.60 + 0.01*raw**

Check: contiki\core\dev\sht11.h

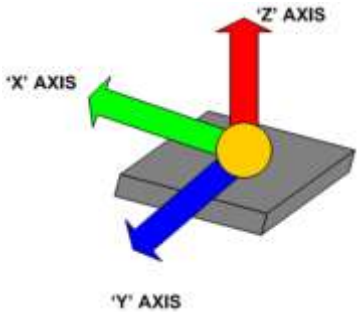
# Reading Data from the Sensors

## ➢ Accessing a digital sensor

- Example: accessing the accelerometer on the Sentilla JCreate

```
SENSOR_ACTIVATE(acc_sensor);
SENSOR_DEACTIVATE(acc_sensor);
acc_sensor.value(x);
```

0 ≤ x ≤ 2: accelerometer axis
x = 3: MSP430 temperature sensor

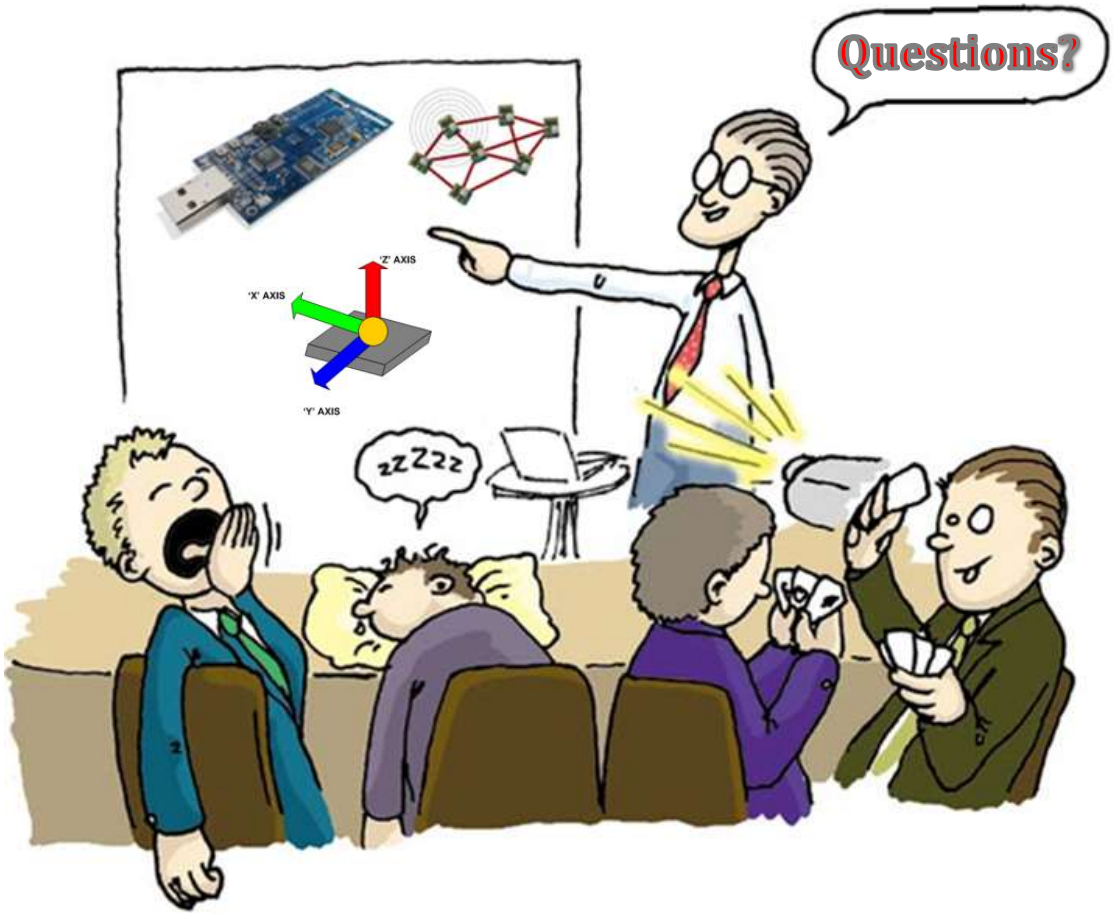



- Example: accessing the accelerometer on the Zolertia Z1

```
accm_read_axis(X_AXIS);
accm_read_axis(Y_AXIS);
accm_read_axis(Z_AXIS);
…
```

Check: contiki\platform\jcreate\dev\acc-sensor.h
Check: contiki\platform\z1\dev\adxl345.h

# References

- **Contiki**
  - A lightweight and flexible operating system for tiny networked sensors
    http://www.sics.se/~adam/dunkels04contiki.pdf
  - Official website
    http://www.sics.se/contiki
  - Contiki examples folder
    contiki-2.x/examples/…

# Questions & Answers