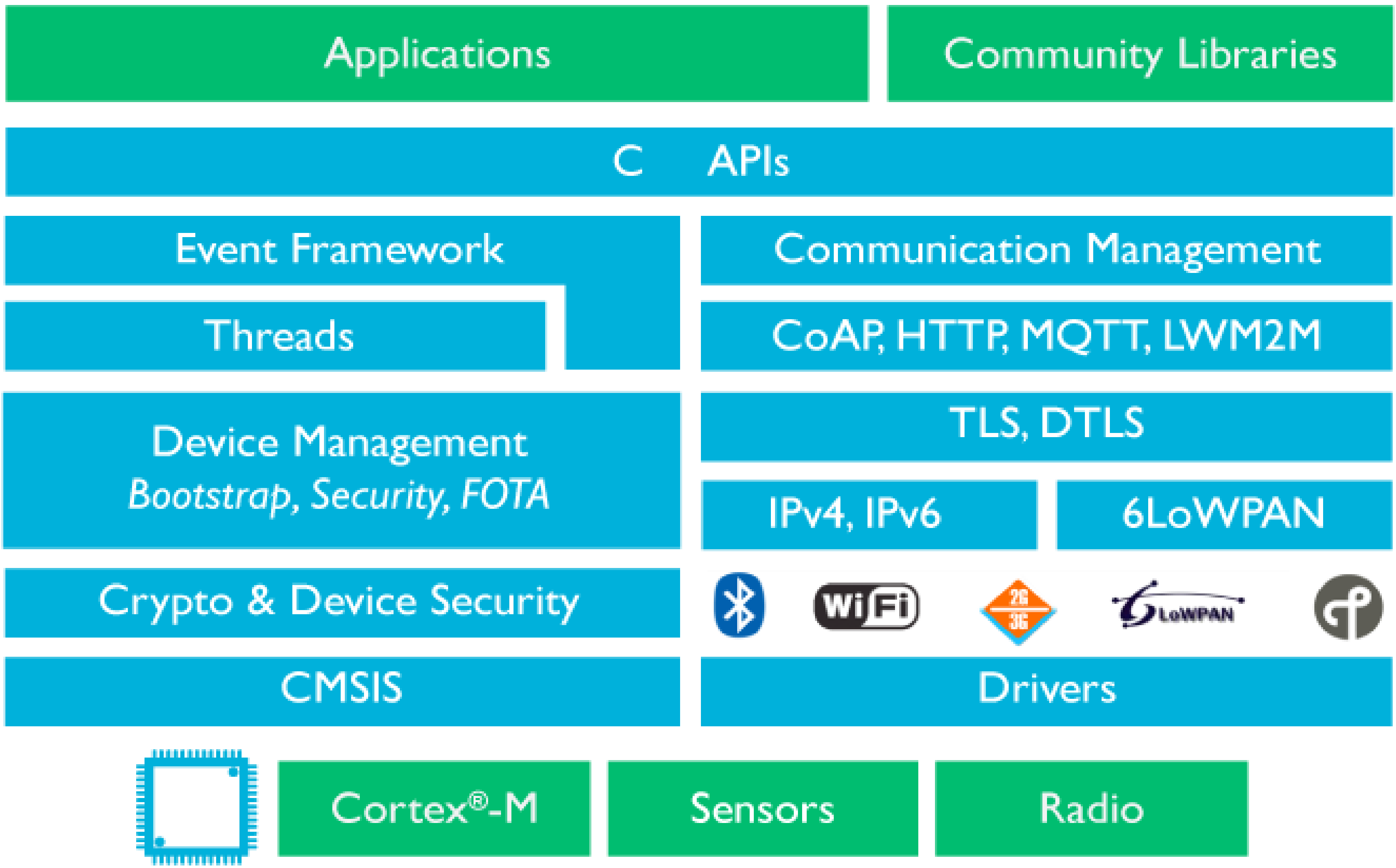
Contiki introduction

Antonio Liñán Colina



https://www.arm.com/products/internet-of-things-solutions/mbed-IoT-device-platform.php

RAM
ROM/FLASH
Processor

<

> 1MB RAM/ROM

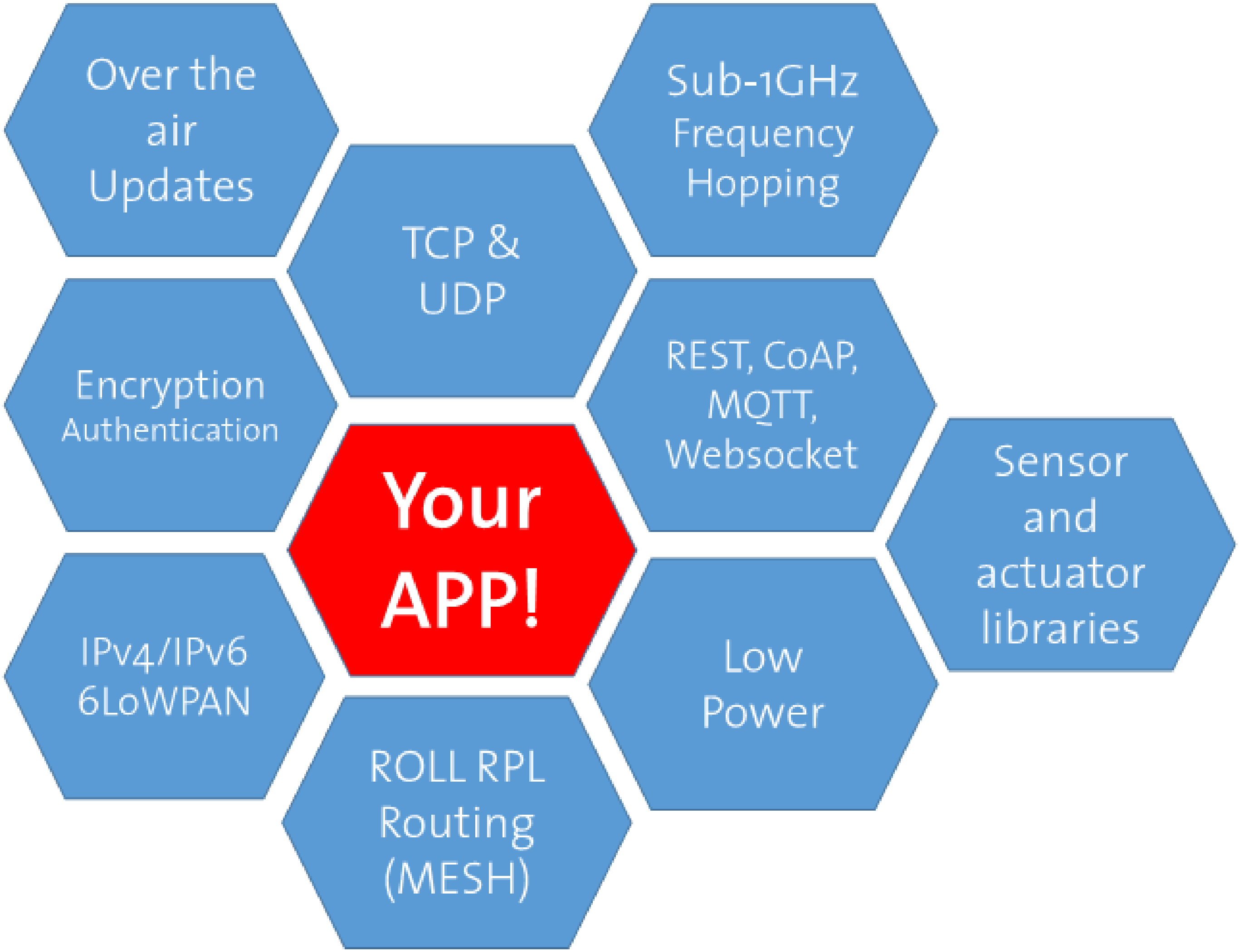
Contiki

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

GCC

open source initiative®

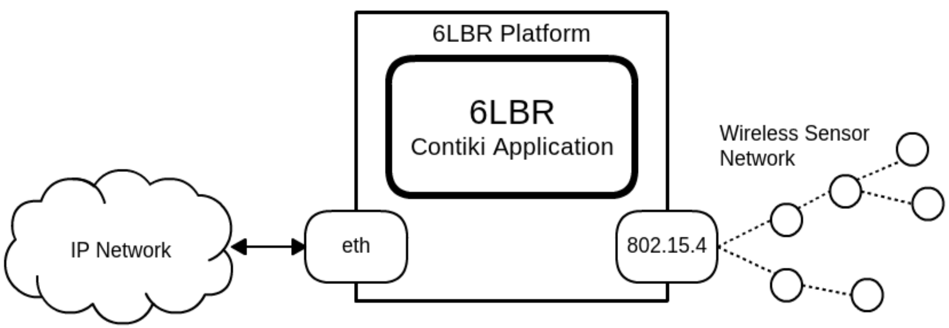

Over the air Updates
Sub-1GHz Frequency Hopping
TCP & UDP
Encryption Authentication
REST, CoAP, MQTT, Websocket
Your APP!
Sensor and actuator libraries
IPv4/IPv6 6LoWPAN
Low Power
ROLL RPL Routing (MESH)

http://www.riva-lighting.de/

thingsquare

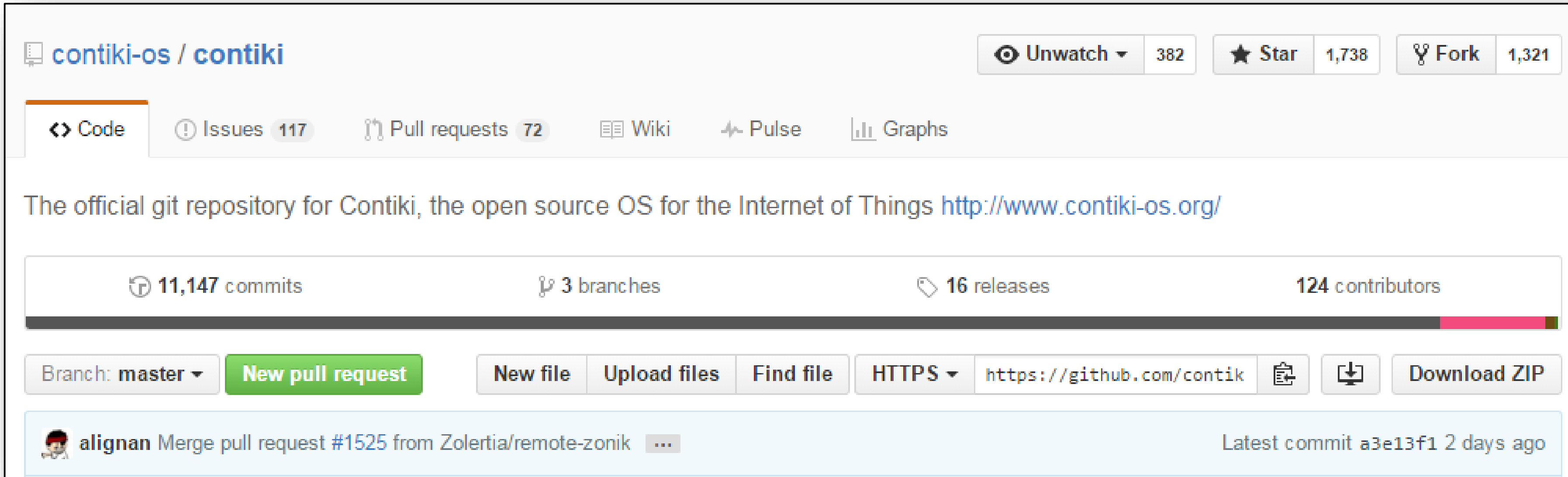
http://www.thingsquare.com

http://www.tado.com



http://cetic.github.io/6lbr/

http://www.lifx.com



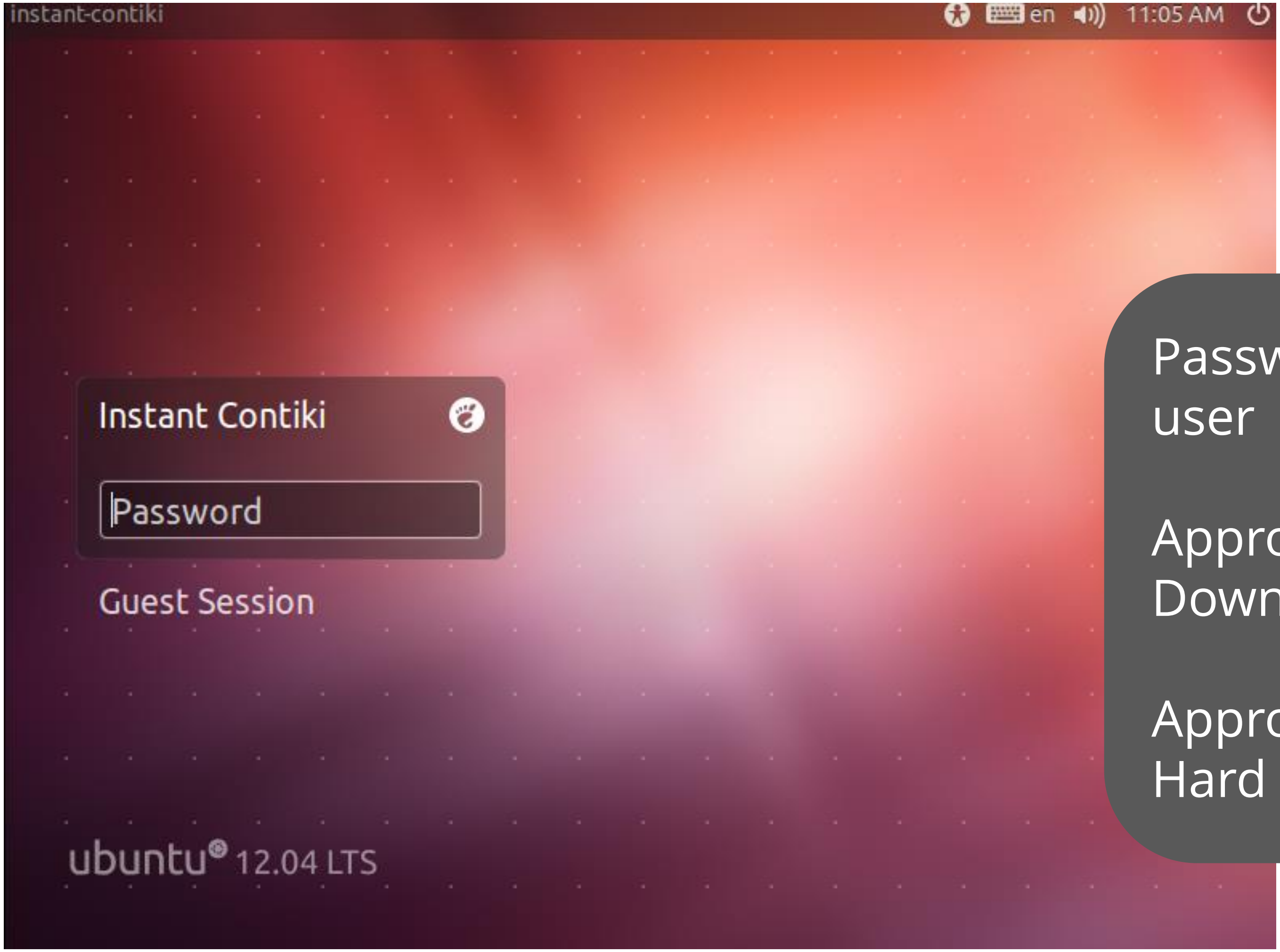
www.contiki-os.org

https://github.com/contiki-os/contiki

Install Contiki

"INSTANT CONTIKI" VIRTUAL MACHINE

VMWare virtualized develop environment



Password is:
user

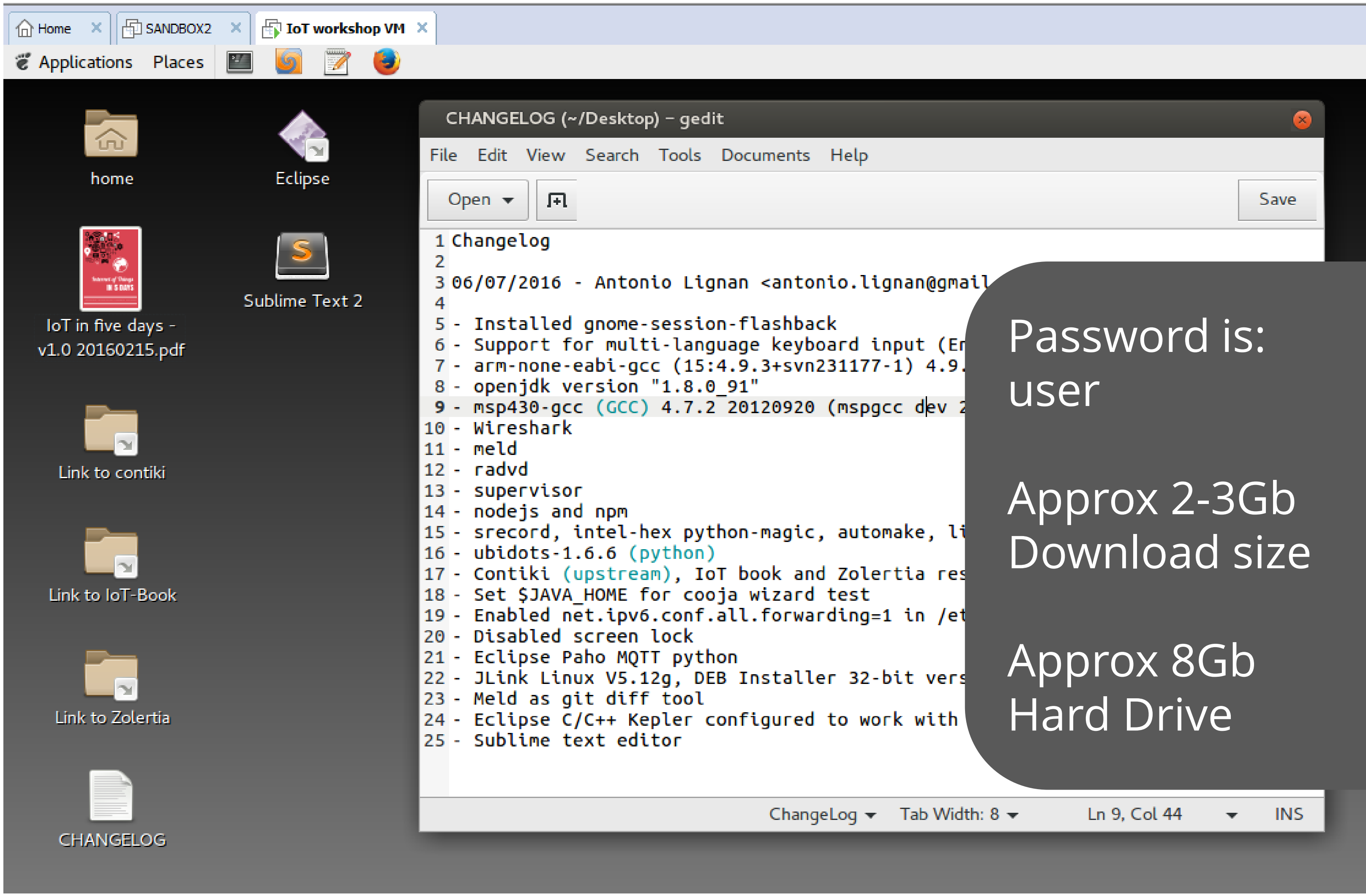
Approx 2-3Gb
Download size

Approx 8Gb
Hard Drive

http://www.contiki-os.org/start.html

"IOT IN FIVE DAYS" VIRTUAL MACHINE

VMWare image built specific for the workshop



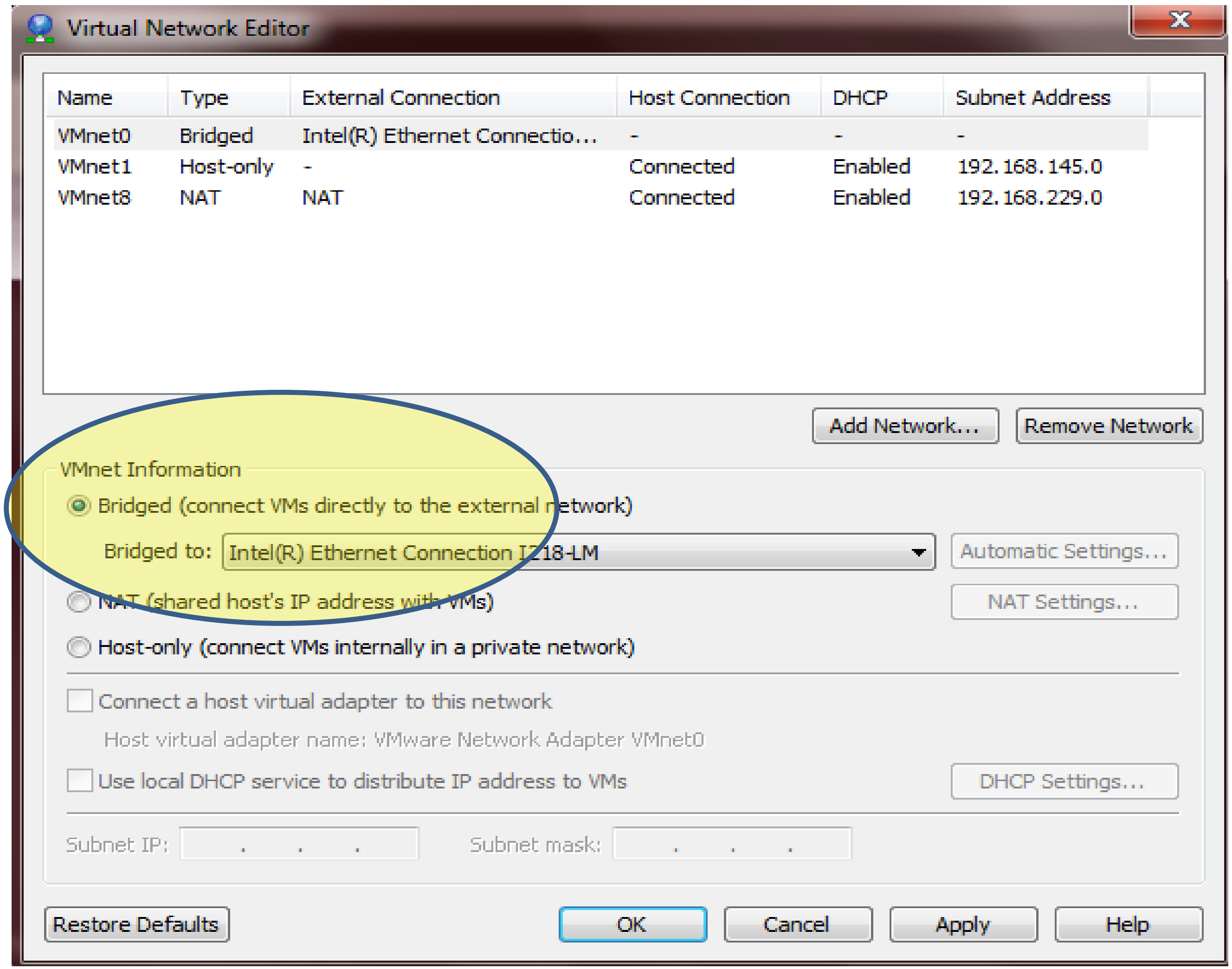
Password is: user

Approx 2-3Gb Download size

Approx 8Gb Hard Drive

https://sourceforge.net/projects/zolertia/files/VM/IoT%20Workshop%20VM.7z

CONFIGURE THE VIRTUAL MACHINE AS "BRIDGED"



Installing Contiki from scratch (Linux)

Install requirements (and extra goodies)

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt-get update
sudo apt-get install gcc-arm-none-eabi gdb-arm-none-eabi
sudo apt-get -y install build-essential automake gettext
sudo apt-get -y install gcc-arm-none-eabi curl graphviz unzip wget
sudo apt-get -y install gcc gcc-msp430
sudo apt-get -y install openjdk-7-jdk openjdk-7-jre ant
sudo apt-get –y install git git-core wireshark
```

Get a Contiki working copy:

```
git clone --recursive https:github.com/alignan/contiki
cd contiki
git submodule –update
git checkout iot-workshop
```

Installing Contiki from scratch (OSX)

Install requirements (only the toolchain)

```
# Install Homebrew - http://brew.sh/
brew tap PX4/homebrew-px4
brew update

# Install GCC Arm Toolchain
brew install gcc-arm-none-eabi-49
```

Get a Contiki working copy:

```
git clone --recursive https:github.com/alignan/contiki
cd contiki
git submodule –update
git checkout iot-workshop
```

OS X

Installing Contiki from scratch (Windows)

Install requirements (only the toolchain)

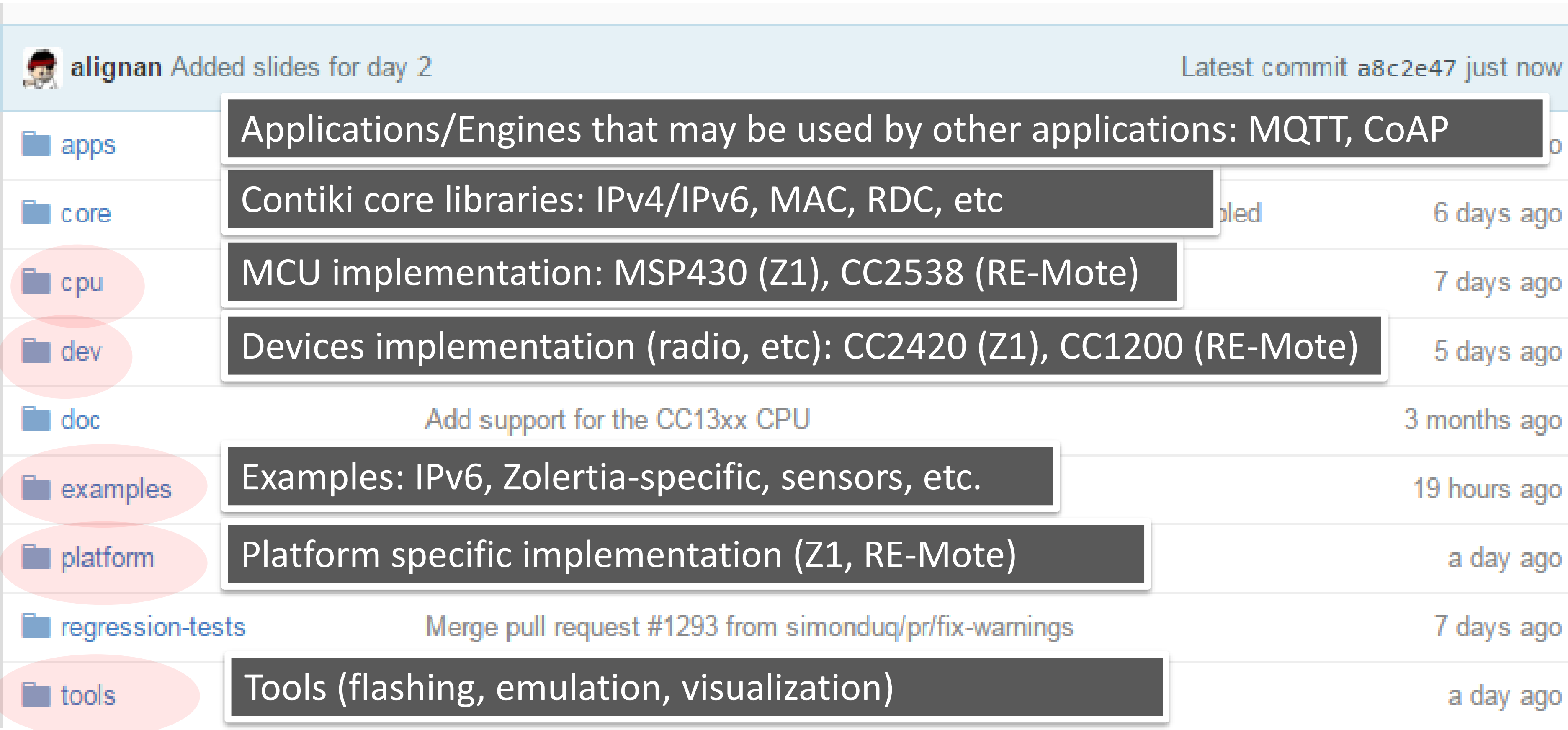
```
# Download ARM toolchain
https://launchpad.net/gcc-arm-embedded/+download

# Download and install MINGW
https://sourceforge.net/projects/mingw/files/latest/download
```

Get a Contiki working copy:

```
git clone --recursive https:github.com/alignan/contiki
cd contiki
git submodule –update
git checkout iot-workshop
```

https://github.com/alignan/IPv6-WSN-book/blob/master/3.asc#windows-installation-re-mote-only

Contiki structure


alignan Added slides for day 2
Latest commit a8c2e47 just now
apps
Applications/Engines that may be used by other applications: MQTT, CoAP
core
Contiki core libraries: IPv4/IPv6, MAC, RDC, etc
6 days ago
cpu
MCU implementation: MSP430 (Z1), CC2538 (RE-Mote)
7 days ago
dev
Devices implementation (radio, etc): CC2420 (Z1), CC1200 (RE-Mote)
5 days ago
doc
Add support for the CC13xx CPU
3 months ago
examples
Examples: IPv6, Zolertia-specific, sensors, etc.
19 hours ago
platform
Platform specific implementation (Z1, RE-Mote)
a day ago
regression-tests
Merge pull request #1293 from simonduq/pr/fix-warnings
7 days ago
tools
Tools (flashing, emulation, visualization)
a day ago

alignan Added optional switch to compile for the Rpi — Latest commit `f5ee9e0` a day ago

..		
apps	…id.h file contains	3 years ago
dev		28 days ago
Makefile.common		a day ago
Makefile.z1	…sing CONTIKI…	a year ago
Makefile.z1sp	Merges Z1SP into Z1 platform	a year ago
README.z1sp	Merges Z1SP into Z1 platform	a year ago
cfs-coffee-arch.h	Coffee no longer uses watchdog calls	5 years ago
contiki-conf.h		17 days ago
contiki-z1-main.c		…nonths ago
contiki-z1-platform.c	Removed all old RCS tags in the Contiki source tree. Those RCS tags a...	3 years ago
node-id.c	Use the Z1 product ID as MAC/Node ID if no value is found in the XMEM	a year ago
platform-conf.h		a year ago

Specific Z1 applications

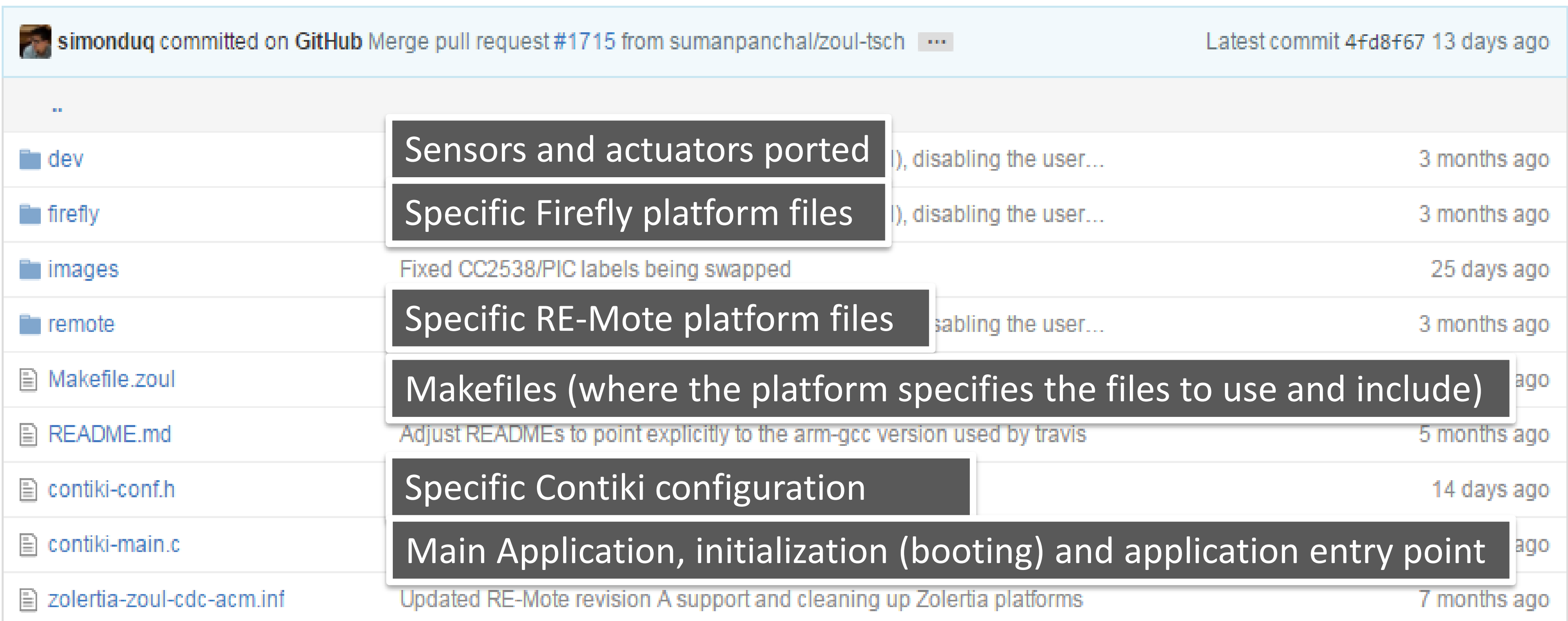
Specific sensors and actuators drivers

Makefiles (where the platform specifies the files to use and include)

Specific Contiki configuration

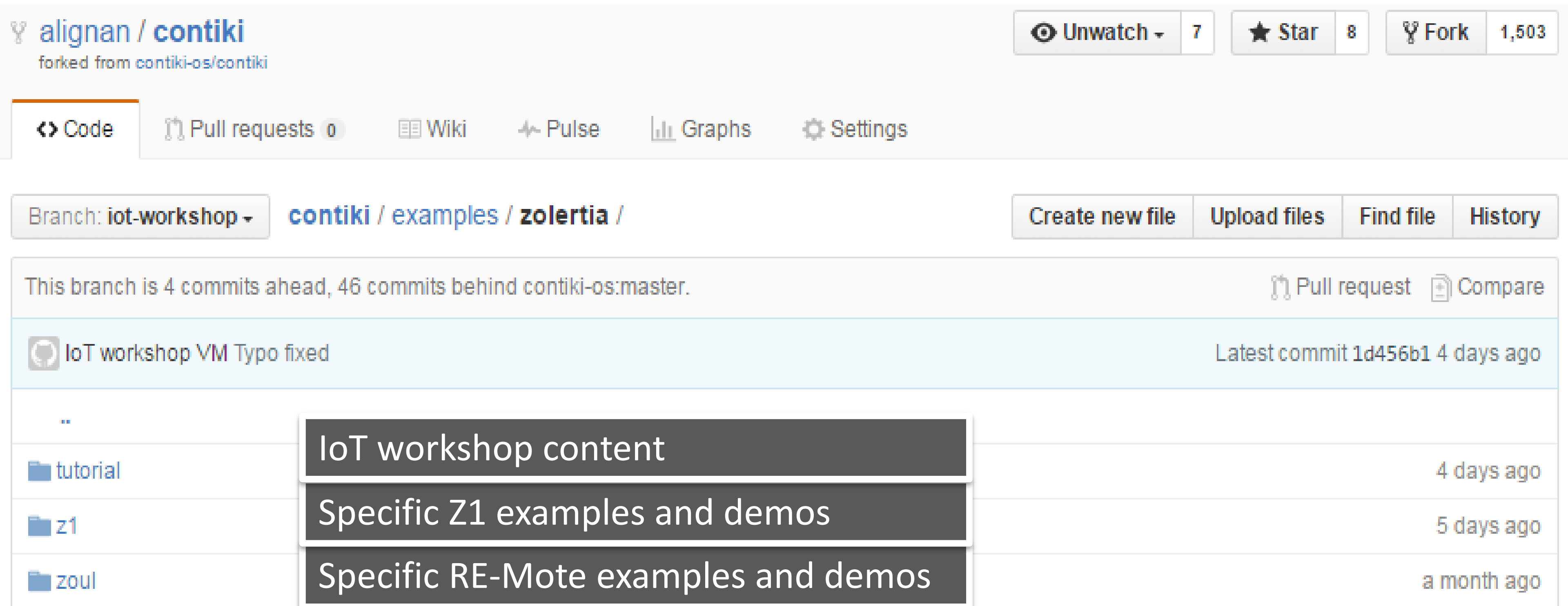
Main Application, Z1 initialization (booting) and application entry point

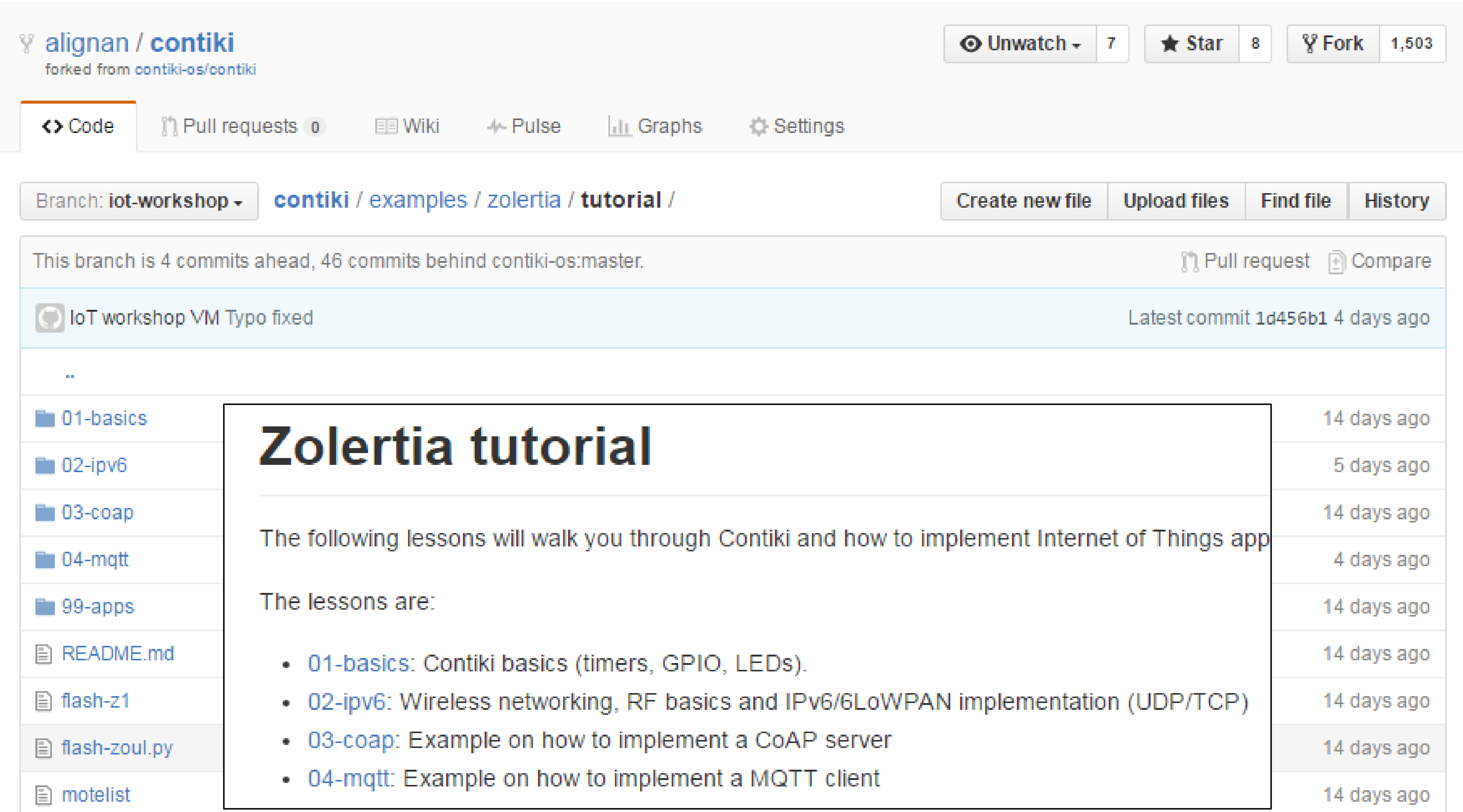
Specific platform configuration (pin-out, peripherals)


simonduq committed on GitHub Merge pull request #1715 from sumanpanchal/zoul-tsch
Latest commit 4fd8f67 13 days ago
..
dev
Sensors and actuators ported
), disabling the user…
3 months ago
firefly
Specific Firefly platform files
), disabling the user…
3 months ago
images
Fixed CC2538/PIC labels being swapped
25 days ago
remote
Specific RE-Mote platform files
sabling the user…
3 months ago
Makefile.zoul
Makefiles (where the platform specifies the files to use and include)
ago
README.md
Adjust READMEs to point explicitly to the arm-gcc version used by travis
5 months ago
contiki-conf.h
Specific Contiki configuration
14 days ago
contiki-main.c
Main Application, initialization (booting) and application entry point
ago
zolertia-zoul-cdc-acm.inf
Updated RE-Mote revision A support and cleaning up Zolertia platforms
7 months ago

ajlozano Allow to use up to 6xADC channels (now hardcoded), disabling the user… ··· Latest commit `a81b400` on 4 Apr

..		
Makefile.remote	Specific RE-Mote Makefile ent feature	6 months ago
README.md	Added support for the RE-Mote on-board Real Time Clock Calendar (RTCC)	6 months ago
antenna-sw.c	Fix code style: platform/zoul	7 months ago
antenna-sw.h	Updated RE-Mote revision A support and cleaning up Zolertia platforms	7 months ago
board.c	Specific RE-Mote initialization	7 months ago
board.h	Allow to use up to 6xADC channels (now hardcoded), disabling the user…	3 months ago
power-mgmt.c	Add support for the RE-Mote on-board power management feature	6 months ago
power-mgmt.h	Add support for the RE-Mote on-board power management feature	6 months ago
rtcc-config.h	Added support for the RE-Mote on-board Real Time Clock Calendar (RTCC)	6 months ago
rtcc.c	Added support for the RE-Mote on-board Real Time Clock Calendar (RTCC)	6 months ago
rtcc.h	Added support for the RE-Mote on-board Real Time Clock Calendar (RTCC)	6 months ago


alignan / contiki
forked from contiki-os/contiki
Unwatch 7
Star 8
Fork 1,503
Code
Pull requests 0
Wiki
Pulse
Graphs
Settings
Branch: iot-workshop
contiki / examples / zolertia /
Create new file
Upload files
Find file
History
This branch is 4 commits ahead, 46 commits behind contiki-os:master.
Pull request
Compare
IoT workshop VM Typo fixed
Latest commit 1d456b1 4 days ago
..
tutorial
4 days ago
z1
5 days ago
zoul
a month ago
IoT workshop content
Specific Z1 examples and demos
Specific RE-Mote examples and demos


alignan / contiki
forked from contiki-os/contiki
Unwatch 7
Star 8
Fork 1,503
Code
Pull requests 0
Wiki
Pulse
Graphs
Settings
Branch: iot-workshop
contiki / examples / zolertia / tutorial /
Create new file
Upload files
Find file
History
This branch is 4 commits ahead, 46 commits behind contiki-os:master.
Pull request
Compare
IoT workshop VM Typo fixed
Latest commit 1d456b1 4 days ago
01-basics
02-ipv6
03-coap
04-mqtt
99-apps
README.md
flash-z1
flash-zoul.py
motelist
14 days ago
5 days ago
14 days ago
4 days ago
14 days ago
14 days ago
14 days ago
14 days ago
14 days ago
Zolertia tutorial
The following lessons will walk you through Contiki and how to implement Internet of Things app
The lessons are:
01-basics: Contiki basics (timers, GPIO, LEDs).
02-ipv6: Wireless networking, RF basics and IPv6/6LoWPAN implementation (UDP/TCP)
03-coap: Example on how to implement a CoAP server
04-mqtt: Example on how to implement a MQTT client

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

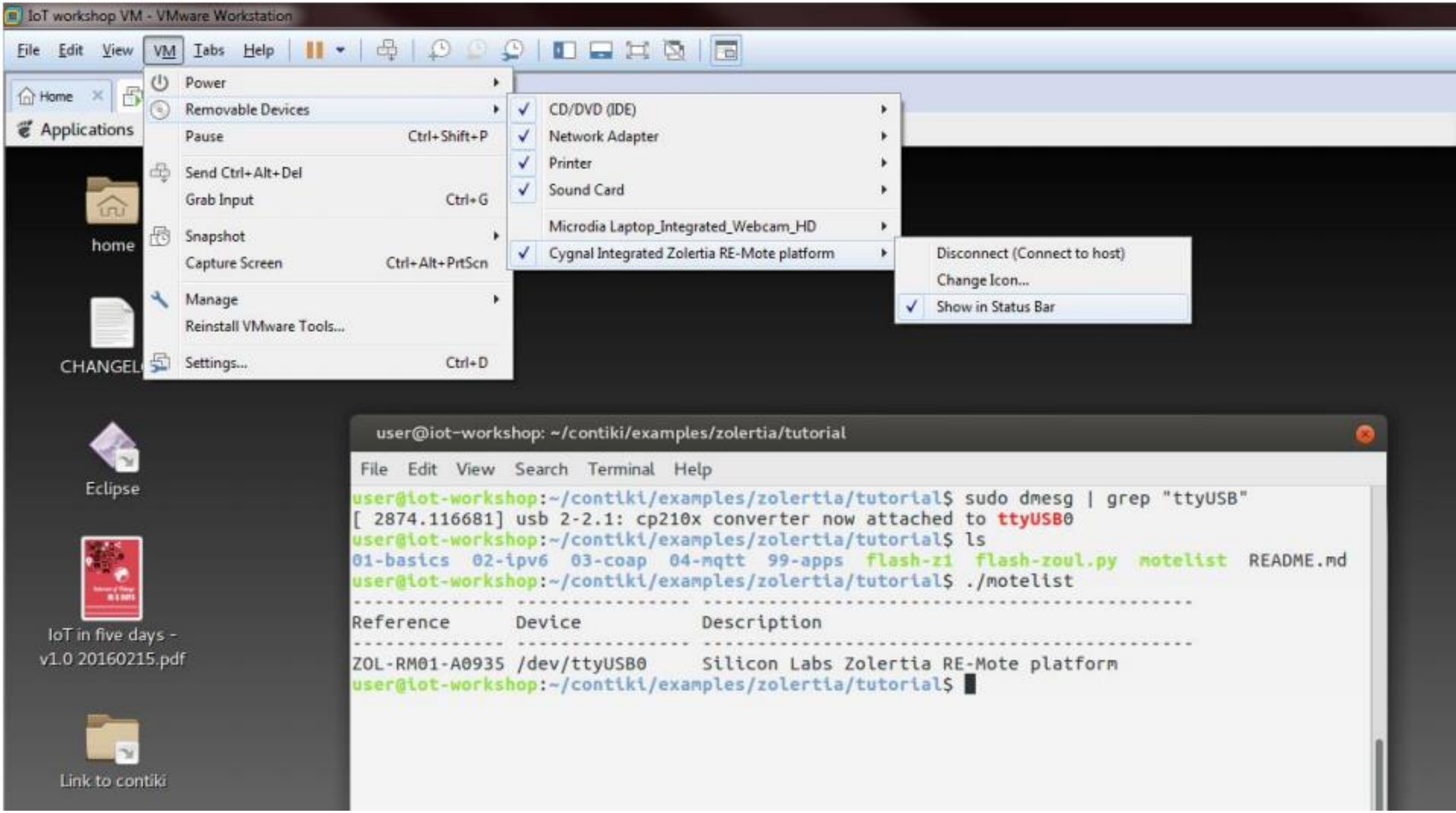
examples/zolertia/tutorial/01-basics

Fixes a problem related to being allowed to write to USB ports as the **/dev/ttyUSB0** used to flash the nodes

```
sudo usermod -a -G dialout user
```

For the change to take effect, you need to logout and log back in
Not required if using the “IoT in five days” virtual image!

VERIFY THE RE-MOTE IS CONNECTED



```
PROCESS(hello_world_process, "Hello world process"); ❶
AUTOSTART_PROCESSES(&hello_world_process); ❷
```

❶ `hello_world_process` is the name of the process and `"Hello world process"` is the readable name of the process when you print it to the terminal.

❷ The `AUTOSTART_PROCESSES(&hello_world_process)` tells Contiki to start that process when it finishes booting.

```
/*---------------------------------------------------------------------------*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(hello_world_process, ev, data) ❶
{
  PROCESS_BEGIN(); ❷
  printf("Hello, world\n"); ❸
  printf("%s\n", hello);
  printf("This is a value in hex 0x%02X, the same as %u\n", num, num);
  PROCESS_END(); ❹
}
```

❶ You declare the content of the process in the process thread. You have the name of the process and callback functions (event handler and data handler).

❷ Inside the thread you begin the process,

❸ do what you want and

❹ finally end the process.

examples/zolertia/tutorial/01-basics/01-hello-world.c

```
CONTIKI_PROJECT = 01-hello-world              ❶
all: $(CONTIKI_PROJECT)                       ❷
CONTIKI = ../../../..                         ❸
include $(CONTIKI)/Makefile.include           ❹
```

❶ Tells the build system which application to compile

❷ If using `make all` it will compile the defined applications

❸ Specify our indentation level respect to Contiki root folder

❹ The system-wide Contiki Makefile, also points out to the platform's Makefile

examples/zolertia/tutorial/01-basics/Makefile

To compile an application:

```
make TARGET=zoul 01-hello-world
```

The **TARGET** is used to tell the compiler which hardware platform it should compile for. For the Z1 platform is “**z1**”, for the RE-Mote (and others zoul-based platforms) is “**zoul**”

You can save the TARGET so next time you don’t have to type it:

```
make TARGET=zoul savetarget
```

This will create a **Makefile.target** file. If no TARGET argument is added in the compilation command line, the compiler will search a valid target in the Makefile.target file (if exists).

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

File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make TARGET=zoul 01-hello-world
  CC        ../../../../cpu/cc2538/./ieee-addr.c
  CC        ../../../../cpu/cc2538/cc2538.lds
  CC        ../../../../cpu/cc2538/./startup-gcc.c
  CC        01-hello-world.c
  LD        01-hello-world.elf
arm-none-eabi-objcopy -O ihex 01-hello-world.elf 01-hello-world.hex
arm-none-eabi-objcopy -O binary --gap-fill 0xff 01-hello-world.elf 01-hello-world.bin
cp 01-hello-world.elf 01-hello-world.zoul
rm 01-hello-world.co obj_zoul/startup-gcc.o
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$
```

To compile and program an application to a RE-Mote:

```
make 01-hello-world.upload
```

The **upload** argument tells the compiler to invoke the programming scripts required to program the compiled binary to the device.

If no specific **PORT** argument is given, it will program all the RE-Motes connected over USB.

```
make 01-hello-world.upload PORT=/dev/ttyUSB0
```

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

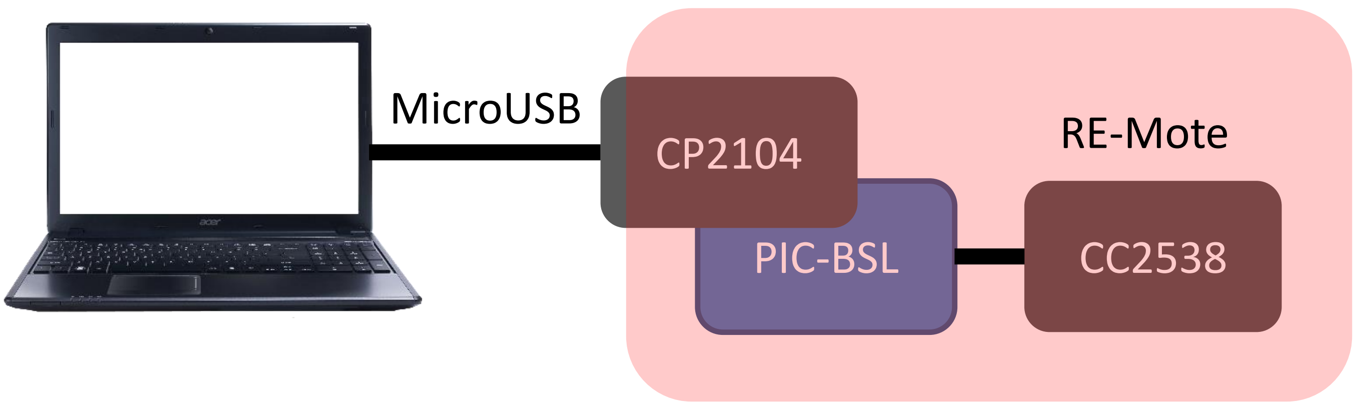
File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make 01-hello-world.upload
using saved target 'zoul'
  CC        ../../../../cpu/cc2538/./ieee-addr.c
  CC        ../../../../cpu/cc2538/cc2538.lds
  CC        ../../../../cpu/cc2538/./startup-gcc.c
  CC        01-hello-world.c
  LD        01-hello-world.elf
arm-none-eabi-objcopy -O binary --gap-fill 0xff 01-hello-world.elf 01-hello-world.bin
python ../../../../tools/cc2538-bsl/cc2538-bsl.py -e -w -v -a 0x00202000 01-hello-world.bin
Opening port /dev/ttyUSB0, baud 500000
Reading data from 01-hello-world.bin
Firmware file: Raw Binary
Connecting to target...
CC2538 PG2.0: 512KB Flash, 32KB SRAM, CCFG at 0x0027FFD4
Primary IEEE Address: 06:15:AB:25:00:12:4B:00
Erasing 524288 bytes starting at address 0x00200000
    Erase done
Writing 516096 bytes starting at address 0x00202000
Write 8 bytes at 0x0027FFF8F00
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0x669e48b1)
rm 01-hello-world.co obj_zoul/startup-gcc.o
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$
```

Programming over the USB is posible as the "**bootloader backdoor**" is enabled as default in Contiki.

The **CP2104** USB to serial converter creates the USB connection, when flashing the image to be programmed is transferred over USB from the Host to the on-board **PIC-BSL** microcontroller, which automatically puts the RE-Mote in "flashing mode" and transfers the image to the RE-Mote's **CC2538**.

The script invoked by the compiler to program the RE-Mote is the **cc2538-bsl**, located in tools/cc2538-bsl. It is already included in Contiki.



https://github.com/JelmerT/cc2538-bsl

MANUALLY ENTER THE FLASHING MODE

Press and hold user button, then press and hold reset button…
release the reset button and then release the user button

https://youtu.be/6vg4f7U_kF4

To visualize the application output in a readable format, the following command is used:

```
make login
```

If no **PORT** is specified, it will open a connection to the first connected device it finds over USB.

```
make login PORT=/dev/ttyUSB0
```

Note you can execute one command after the other:

```
make 01-hello-world.upload PORT=/dev/ttyUSB0 && make login PORT=/dev/ttyUSB0
```

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

File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make login PORT=/dev/ttyUSB0
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200  /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 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
Hello, world
Hello world, again!
This is a value in hex 0xABCD, the same as 43981
```

The next command is similar to make login, but it appends an UNIX **timestamp**:

```
make serialview
```

If no **PORT** is specified, it will open a connection to the first connected device it finds over USB.

```
make serialview PORT=/dev/ttyUSB0
```

You can also list the connected devices:

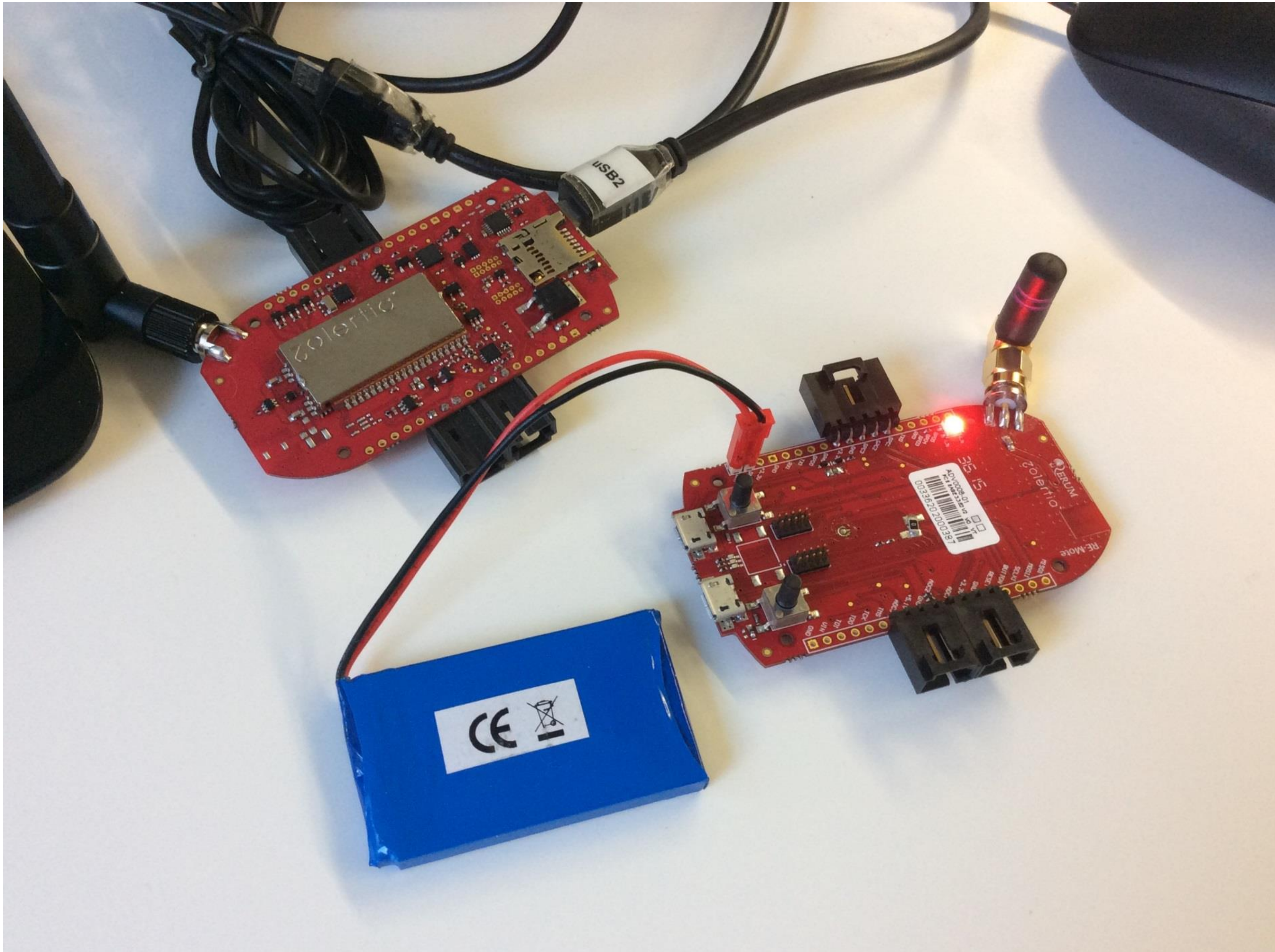
```
make motelist
```

```
user@iot-workshop: ~/contiki/examples/zolertia/tutorial/01-basics
File  Edit  View  Search  Terminal  Help
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make motelist
using saved target 'zoul'
../../../../tools/zolertia/motelist-zolertia
-------------- ---------------- ----------------------------------------------
Reference      Device           Description
-------------- ---------------- ----------------------------------------------
ZOL-RM01-A0935 /dev/ttyUSB0     Silicon Labs Zolertia RE-Mote platform
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make serialview
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0 | ../../../../tools/timestamp
connecting to /dev/ttyUSB0 (115200) [OK]
1467033730 Contiki-3.x-2612-g1d456b1
1467033730 Zolertia RE-Mote platform
1467033730 CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 1
1467033730 System clock: 16000000 Hz
1467033730 I/O clock: 16000000 Hz
1467033730 Reset cause: External reset
1467033730 Rime configured with address 00:12:4b:00:06:15:ab:25
1467033730  Net: sicslowpan
1467033730  MAC: CSMA
1467033730  RDC: ContikiMAC
1467033730 Hello, world
1467033730 Hello world, again!
1467033730 This is a value in hex 0xABCD, the same as 43981
```

Buttons and LEDs

Events and actions can be triggered by pressing the user button: send a message over the radio, take a sensor sample, start a process, etc.

The LEDs (light-emitting diodes) help us to understand what happens in the device, by using different colours and blinking sequences we know when an event is happening, if there are any errors or what happens in our application.



examples/zolertia/tutorial/01-basics/02-led-and-button.c

```
#include "contiki.h"
#include "dev/leds.h"
#include "dev/button-sensor.h"
#include <stdio.h>
/*---------------------------------------------------------------------------*/
PROCESS(led_button_process, "LEDs and button example process");
AUTOSTART_PROCESSES(&led_button_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(led_button_process, ev, data)
{
  PROCESS_BEGIN();
  SENSORS_ACTIVATE(button_sensor);          ❶

  while(1) {                                ❷

    printf("Press the User Button\n");
    PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);   ❸
    /* When the user button is pressed, we toggle the LED on/off... */
    leds_toggle(LEDS_GREEN);                ❹

    /*
     * And we print its status: when zero the LED is off, else on.
     * The number printed when the sensor is on is the LED ID, this value is
     * used as a mask, to allow turning on and off multiple LED at the same
     * time (for example using "LEDS_GREEN + LEDS_RED" or "LEDS_ALL"
     */
    printf("The sensor is: %u\n", leds_get());   ❺
  }

  PROCESS_END();
}
```

The application will wait (as it will be paused) until this event happens, saving processing cycles and energy

examples/zolertia/tutorial/01-basics/02-led-and-button.c

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

File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make login
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 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
Press the User Button
The sensor is: 16
Press the User Button
The sensor is: 0
Press the User Button
The sensor is: 16
Press the User Button
The sensor is: 0
Press the User Button
```

Press and hold the button, the LED will be on

When the button is released, the LED will be off

examples/zolertia/tutorial/01-basics/02-led-and-button.c

```
 *
 * LEDs on the RE-Mote are connected as follows:
 * - LED1 (Red)     -> PD5
 * - LED2 (Green)   -> PD4
 * - LED3 (Blue)    -> PD3
 *
 * LED1 pin shared with EXT_WDT and exposed in JP6 connector
 * LED2 pin shared with UART1 CTS, pin exposed in JP6 connector
 * LED3 pin shared with UART1 RTS, exposed in JP6 connector
 * @{
 */
/*---------------------------------------------------------------------------*/
/* Some files include leds.h before us, so we need to get rid of defaults in
 * leds.h before we provide correct definitions */
#undef LEDS_GREEN
#undef LEDS_YELLOW
#undef LEDS_BLUE
#undef LEDS_RED
#undef LEDS_CONF_ALL

/* In leds.h the LEDS_BLUE is defined by LED_YELLOW definition */
#define LEDS_GREEN    (1 << 4) /**< LED1 (Green) -> PD4 */
#define LEDS_BLUE     (1 << 3) /**< LED2 (Blue)  -> PD3 */
#define LEDS_RED      (1 << 5) /**< LED3 (Red)   -> PD5 */

#define LEDS_CONF_ALL (LEDS_GREEN | LEDS_BLUE | LEDS_RED)

#define LEDS_LIGHT_BLUE (LEDS_GREEN | LEDS_BLUE) /**< Green + Blue (24)       */
#define LEDS_YELLOW     (LEDS_GREEN | LEDS_RED)  /**< Green + Red  (48)       */
#define LEDS_PURPLE     (LEDS_BLUE  | LEDS_RED)  /**< Blue + Red   (40)       */
#define LEDS_WHITE      LEDS_ALL                 /**< Green + Blue + Red (56) */
```

platforms/zoul/remote/board.h

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

File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make login
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 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
Press the User Button
The sensor is: 16
Press the User Button
The sensor is: 0
Press the User Button
The sensor is: 16
Press the User Button
The sensor is: 0
Press the User Button
```

16 in binary is 00010000

Same as (1 << 4)

The LED on should be the Green:

#define LEDS_GREEN (1 << 4) /< LED1 (Green) -> PD4 */**

examples/zolertia/tutorial/01-basics/02-led-and-button.c

```c
/*---------------------------------------------------------------------------*/
#ifndef BUTTON_SENSOR_H_
#define BUTTON_SENSOR_H_
/*---------------------------------------------------------------------------*/
#include "lib/sensors.h"
/*---------------------------------------------------------------------------*/
#define BUTTON_SENSOR "Button"

extern const struct sensors_sensor button_sensor;
/*---------------------------------------------------------------------------*/
extern process_event_t button_press_duration_exceeded;
/*---------------------------------------------------------------------------*/
#define BUTTON_SENSOR_CONFIG_TYPE_INTERVAL      0x0100

#define BUTTON_SENSOR_VALUE_TYPE_LEVEL          0
#define BUTTON_SENSOR_VALUE_TYPE_PRESS_DURATION 1

#define BUTTON_SENSOR_PRESSED_LEVEL             0
#define BUTTON_SENSOR_RELEASED_LEVEL            8
/*---------------------------------------------------------------------------*/
#endif /* BUTTON_SENSOR_H_ */
/*---------------------------------------------------------------------------*/
```

platforms/zoul/dev/button-sensor.h

Always after making a change to the source code, clean the previously compiled objects

```
make clean
```

```
while(1) {
  PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);

  if(button_sensor.value(BUTTON_SENSOR_VALUE_TYPE_LEVEL) ==
                         BUTTON_SENSOR_PRESSED_LEVEL) {

    /* When the user button is pressed, we toggle the LED on/off... */
    leds_toggle(LEDS_GREEN + LEDS_RED);

    printf("The sensor is: %u\n", leds_get());

  } else {
    printf("Press the User Button\n");
  }
}
```

What Color the LEDs will show?
What number is printed when the LEDs are on?
What will happen each time we press the button?

examples/zolertia/tutorial/01-basics/02-led-and-button.c

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

File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make login
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Contiki-3.x-2612-g1d456b1
Zolertia RE-Mote platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 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
The sensor is: 48
Press the User Button
The sensor is: 0
Press the User Button
The sensor is: 48
Press the User Button
The sensor is: 0
Press the User Button
```

Each time we press the button, the LED will toggle on and off showing a yellow-ish color

examples/zolertia/tutorial/01-basics/02-led-and-button.c

```c
/* Start the user button using the "SENSORS_ACTIVATE" macro */
SENSORS_ACTIVATE(button_sensor);

button_sensor.configure(BUTTON_SENSOR_CONFIG_TYPE_INTERVAL,
                        CLOCK_SECOND);

while(1) {
  PROCESS_YIELD();
```

We are waiting for ANY event to happen

```c
  if(ev == sensors_event && data == &button_sensor) {
    if(button_sensor.value(BUTTON_SENSOR_VALUE_TYPE_LEVEL) ==
                           BUTTON_SENSOR_PRESSED_LEVEL) {

      /* When the user button is pressed, we toggle the LED on/off... */
      leds_toggle(LEDS_GREEN + LEDS_RED);

      printf("The sensor is: %u\n", leds_get());

    } else {
      printf("Press the User Button\n");
    }

  } else if(ev == button_press_duration_exceeded) {
    printf("Button pressed for %d ticks [%u events]\n",
          (*((uint8_t *)data) * CLOCK_SECOND),
          button_sensor.value(BUTTON_SENSOR_VALUE_TYPE_PRESS_DURATION));
  }
}
```

examples/zolertia/tutorial/01-basics/02-led-and-button.c

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

File Edit View Search Terminal Help

```
user@iot-workshop:~/contiki/examples/zolertia/tutorial/01-basics$ make login
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
The sensor is: 0
Press the User Button
The sensor is: 48
Press the User Button
The sensor is: 0
Button pressed for 128 ticks [1 events]
Button pressed for 256 ticks [2 events]
Button pressed for 384 ticks [3 events]
Button pressed for 512 ticks [4 events]
Button pressed for 640 ticks [5 events]
Button pressed for 768 ticks [6 events]
Button pressed for 896 ticks [7 events]
Button pressed for 1024 ticks [8 events]
Press the User Button
The sensor is: 48
Press the User Button
```

The press and hold option extends the possible alternatives the application would have to handle the user input

examples/zolertia/tutorial/01-basics/02-led-and-button.c

Timers

Timers allow to execute actions periodically, like measuring a sensor periodically, waiting a few seconds before executing a function, sending hourly reports, etc.

- Simple timer: A simple ticker, the application should check *manually* if the timer has expired. More information at `core/sys/timer.h`.
- Callback timer: When a timer expires it can callback a given function. More information at `core/sys/ctimer.h`.
- Event timer: Same as above, but instead of calling a function, when the timer expires it posts an event signalling its expiration. More information at `core/sys/etimer.h`.
- Real time timer: The real-time module handles the scheduling and execution of real-time tasks, there's only 1 timer available at the moment. More information at `core/sys/rtimer.h`

examples/zolertia/tutorial/01-basics/03-timers.c

```
ticks = 0;
stimer_set(&st, 1);                ❶
while(!stimer_expired(&st)) {      ❷
  ticks++;
}
printf("stimer, ticks: \t%ld\n", ticks);
```

The timer and stimer are computing-intensive, whereas the etimer, ctimer and rtimer generate events/callbacks only when the counter expires, saving processing cycles and energy

```
ticks = 0;
etimer_set(&et, CLOCK_SECOND * 2);

ticks++;
PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);
printf("etimer, now: \t%ld\n", ticks);
```

examples/zolertia/tutorial/01-basics/03-timers.c

```
ticks = 0;
stimer_set(&st, 1);              ❶
while(!stimer_expired(&st)) {    ❷
  ticks++;
}
printf("stimer, ticks: \t%ld\n", ticks);
```

CLOCK_SECOND is a constant defined by each platform, specifies the number of ticks a second has according to the platform's clock. If you want to specify a 5 seconds timer, then use (CLOCK_SECOND * 5), or likewise half a second would be the nearest integer to (CLOCK_SECOND / 2)

```
ticks = 0;
etimer_set(&et, CLOCK_SECOND * 2);

ticks++;
PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);
printf("etimer, now: \t%ld\n", ticks);
```

examples/zolertia/tutorial/01-basics/03-timers.c

Each timer has its own API Aapplication Programming Interface) or commands, but mostly are alike:

```
void etimer_set(struct etimer *t, clock_time_t interval); // Start the timer.
void etimer_reset(struct etimer *t); // Restart the timer from the previous expiration time.
void etimer_restart(struct etimer *t); // Restart the timer from current time.
void etimer_stop(struct etimer *t); // Stop the timer.
int etimer_expired(struct etimer *t); // Check if the timer has expired.
int etimer_pending(); // Check if there are any non-expired event timers.
clock_time_t etimer_next_expiration_time(); // Get the next event timer expiration time.
void etimer_request_poll(); // Inform the etimer library that the system clock has changed.
```

core/sys/etimer.h

```
/* The event timer library */
#include "sys/etimer.h"

/* The seconds timer library */
#include "sys/stimer.h"

/* The regular timer library */
#include "sys/timer.h"

/* The callback timer library */
#include "sys/ctimer.h"

/* The "real-time" timer library */
#include "sys/rtimer.h"
```

Depending on the type of timers we need to use, include a given header and create a timer structure

```
/* The following are the structures used when you need to include a timer, it
 * serves to keep the timer information.
 */
static struct timer  nt;
static struct stimer st;
static struct etimer et;
static struct ctimer ct;
static struct rtimer rt;
```

examples/zolertia/tutorial/01-basics/03-timers.c

```
static void
ctimer_callback_example(void *ptr)
{
  uint32_t *ctimer_ticks = ptr;                          ❶
  printf("ctimer, now: \t%ld\n", *ctimer_ticks);

  /* The real timer allows execution of real-time tasks (with predictable
   * execution times).
   * The function RTIMER_NOW() is used to get the current system time in ticks
   * and RTIMER_SECOND specifies the number of ticks per second.
   */

  (*ctimer_ticks)++;                                     ❷
  rtimer_set(&rt, RTIMER_NOW() + RTIMER_SECOND, 0,       ❸
             rtimer_callback_example, ctimer_ticks);
}
```

RTIMER_NOW() is a macro returning the current tick count
RTIMER_SECOND is a platform-defined constant, number of ticks/second

```
ctimer_set(&ct, CLOCK_SECOND * 2, ctimer_callback_example, &ticks);

/* And we keep the process halt while we wait for the callback timer to
 * expire.
 */

while(1) {
  PROCESS_YIELD();
```

examples/zolertia/tutorial/01-basics/03-timers.c

```
static void
ctimer_callback_example(void *ptr)
{
  uint32_t *ctimer_ticks = ptr;                    1
  printf("ctimer, now: \t%ld\n", *ctimer_ticks);

  /* The real timer allows execution of real-time tasks (with predictable
   * execution times).
   * The function RTIMER_NOW() is used to get the current
   * and RTIMER_SECOND specifies the number of ticks per se
   */

  (*ctimer_ticks)++;                               2
  rtimer_set(&rt, RTIMER_NOW() + RTIMER_SECOND, 0, 3
             rtimer_callback_example, ctimer_ticks);
}
```

```
static void
rtimer_callback_example(struct rtimer *timer, void *ptr)
{
  uint32_t *rtimer_ticks = ptr;
  printf("rtimer, now: \t%ld\n", *rtimer_ticks);

  /* We can restart the ctimer and keep the counting going */
  (*rtimer_ticks)++;
  ctimer_restart(&ct);
}
```

examples/zolertia/tutorial/01-basics/03-timers.c

Print the value of CLOCK_SECOND and RTIMER_SECOND

Turn on only the red LED each time the “tick” counter variable is even, and the blue LED whenever is odd. Use only the ctimer from the example

Stop the timers whenever the user button is pressed

Restart the timer and set the tick variable value to zero when the user button is pressed again

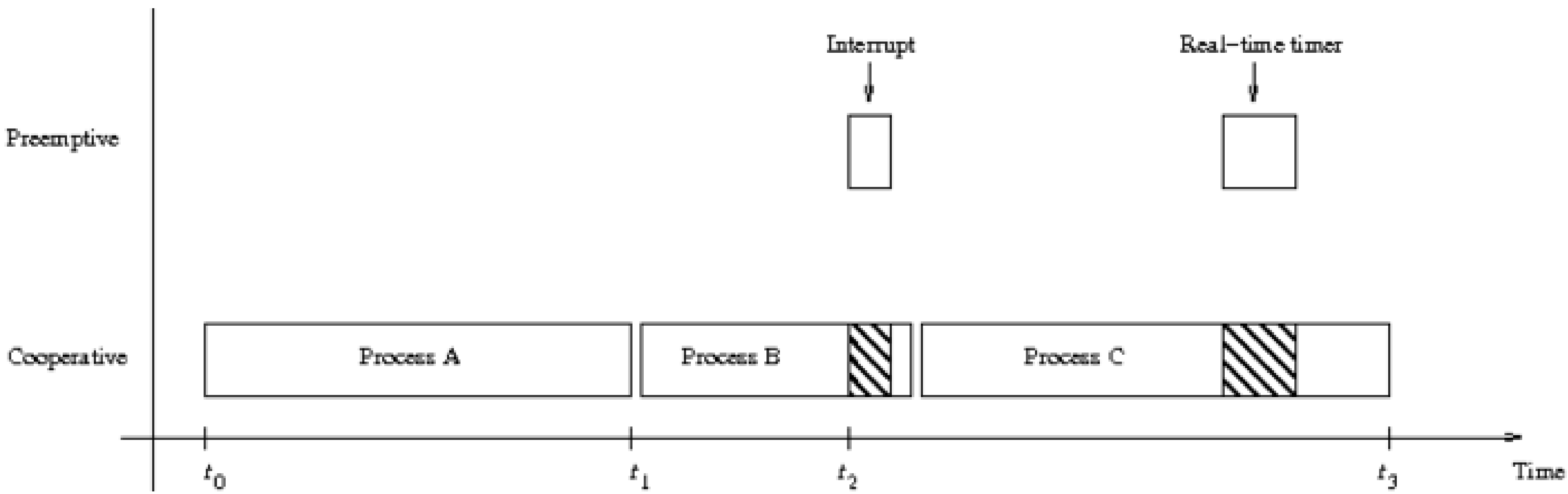
examples/zolertia/tutorial/01-basics/03-timers.c

```c
static void
ctimer_callback_example(void *ptr)
{
  uint32_t *ctimer_ticks = ptr;
  printf("ctimer, now: \t%ld\n", *ctimer_ticks);
  leds_off(LEDS_ALL);

  if((*ctimer_ticks % 2) == 0) {
    leds_on(LEDS_RED);       /* Even value */
  } else {
    leds_on(LEDS_BLUE);      /* Odd value */
  }
  (*ctimer_ticks)++;
  ctimer_reset(&ct);
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(test_timers_process, ev, data)
{
  PROCESS_BEGIN();
  static uint32_t ticks = 0;
  ctimer_set(&ct, CLOCK_SECOND * 2, ctimer_callback_example, &ticks);
  SENSORS_ACTIVATE(button_sensor);

  while(1) {
    PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data == &button_sensor);

    if(button_sensor.value(BUTTON_SENSOR_VALUE_TYPE_LEVEL) ==
                           BUTTON_SENSOR_PRESSED_LEVEL) {
      if(ctimer_expired(&ct)) {
        ctimer_reset(&ct);
      } else {
        ctimer_stop(&ct);
        ticks = 0;
      }
    }
  }
  PROCESS_END();
}
```

examples/zolertia/tutorial/01-basics/03-timers.c

Processes

Contiki has two execution contexts: cooperative and preemptive
Processes are cooperative and sequential, interrupts (button, sensors events) and the real-timer are preemptive.



https://github.com/contiki-os/contiki/wiki/Processes

examples/zolertia/tutorial/01-basics/04-processes.c

```
struct process {
  struct process *next;
  const char *name;
  int (* thread)(struct pt *,
                 process_event_t,
                process_data_t);
  struct pt pt;
  unsigned char state, needspoll;
};
```

The process control block (RAM): contains run-time information about the process such as the name of the process, the state of the process, and a pointer to the process thread of the process.

```
PROCESS(hello_world_process, "Hello world process");
```

```
PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();

  printf("Hello, world\n");

  PROCESS_END();
}
```

The process thread (ROM): the code of the process

https://github.com/contiki-os/contiki/wiki/Processes

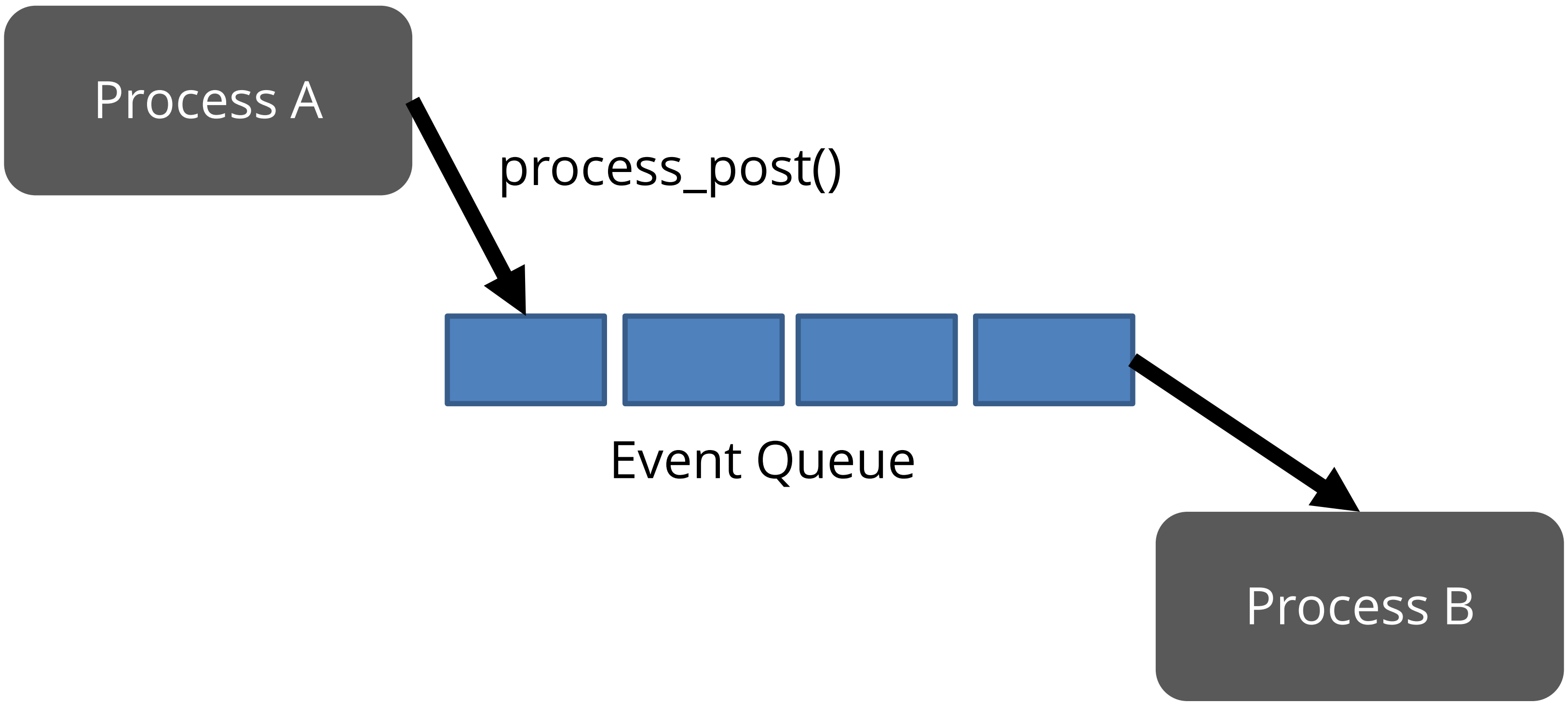
examples/zolertia/tutorial/01-basics/04-processes.c

```
PROCESS_BEGIN(); // Declares the beginning of a process' protothread.
PROCESS_END(); // Declares the end of a process' protothread.
PROCESS_EXIT(); // Exit the process.
PROCESS_WAIT_EVENT(); // Wait for any event.
PROCESS_WAIT_EVENT_UNTIL(); // Wait for an event, but with a condition.
PROCESS_YIELD(); // Wait for any event, equivalent to PROCESS_WAIT_EVENT().
PROCESS_WAIT_UNTIL(); // Wait for a given condition; may not yield the process.
PROCESS_PAUSE(); // Temporarily yield the process.
```

```
#define PROCESS_EVENT_NONE            128
#define PROCESS_EVENT_INIT            129
#define PROCESS_EVENT_POLL            130
#define PROCESS_EVENT_EXIT            131
#define PROCESS_EVENT_CONTINUE        133
#define PROCESS_EVENT_MSG             134
#define PROCESS_EVENT_EXITED          135
#define PROCESS_EVENT_TIMER           136
```

https://github.com/contiki-os/contiki/wiki/Processes

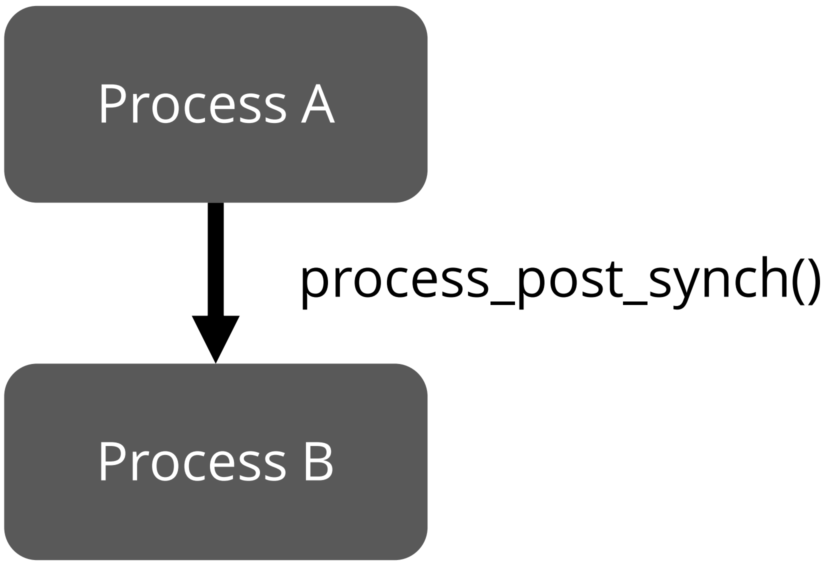
core/sys/process.h




```
/* Send the PROCESS_EVENT_MSG event asynchronously to
   "Example process", with a pointer to the message in the
   array 'msg'. */
process_post(&example_process,
             PROCESS_EVENT_CONTINUE, msg);
```

https://github.com/contiki-os/contiki/wiki/Processes

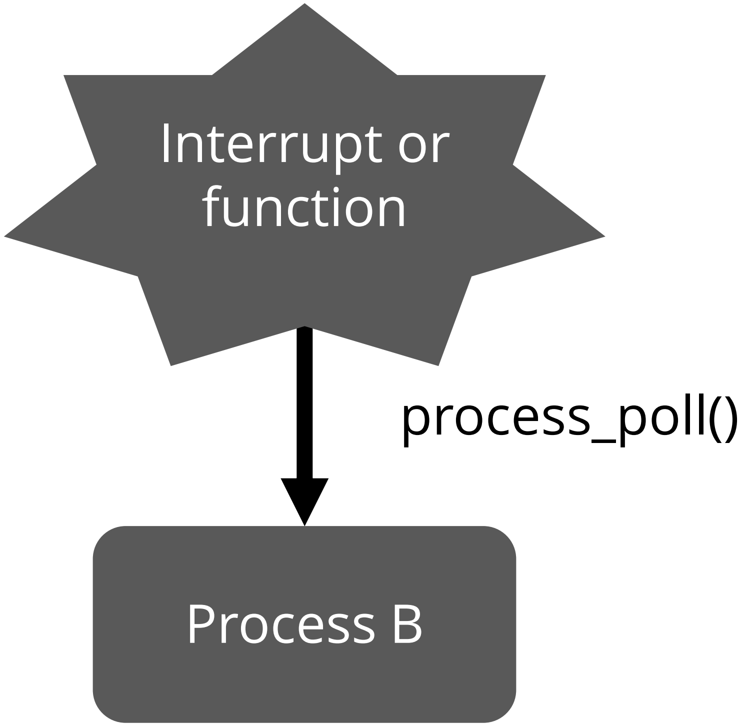
examples/zolertia/tutorial/01-basics/04-processes.c




```
/* Send the PROCESS_EVENT_MSG event synchronously to
   "Example process", with a pointer to the message in the
   array 'msg'. */
process_post_synch(&example_process,
                   PROCESS_EVENT_CONTINUE, msg);
```

https://github.com/contiki-os/contiki/wiki/Processes

examples/zolertia/tutorial/01-basics/04-processes.c

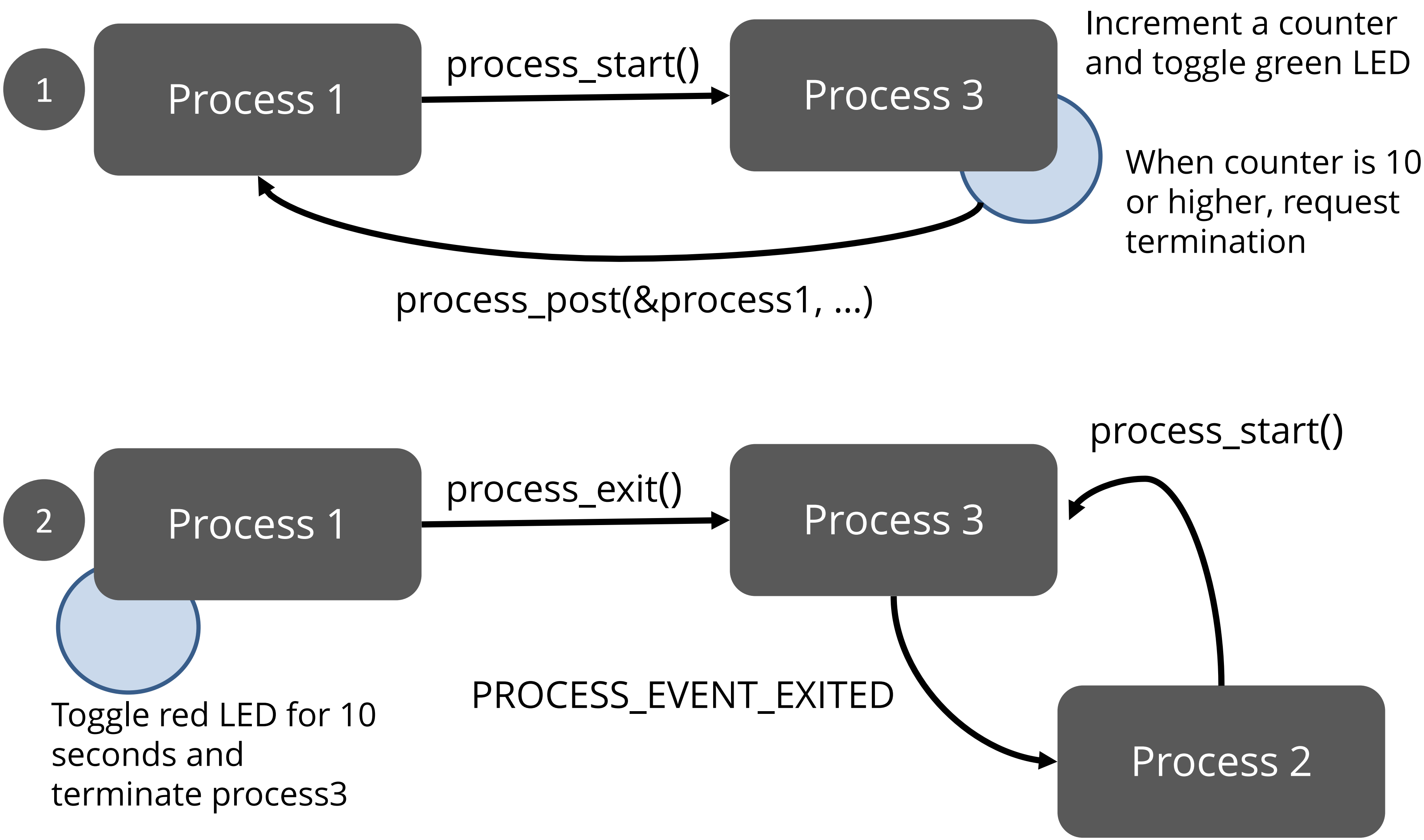


The only function that can be called from preemptive context

```
/* Poll "Example process". */
process_poll(&example_process);
```

https://github.com/contiki-os/contiki/wiki/Processes

examples/zolertia/tutorial/01-basics/04-processes.c



examples/zolertia/tutorial/01-basics/04-processes.c

```
PROCESS(process1, "Main process");
PROCESS(process2, "Auxiliary process");
PROCESS(process3, "Another auxiliary process");

/* But we are only going to automatically start the first two */
AUTOSTART_PROCESSES(&process1, &process2);
```

```
PROCESS_THREAD(process1, ev, data)
{
  PROCESS_BEGIN();
  static uint8_t counter;
  printf("Process 1 started\n");
  process_start(&process3, "Process 1");          1

  while(1) {
    PROCESS_YIELD();                               2
    if(ev == event_from_process3) {                3
      counter = *((uint8_t *)data);                4
      printf("Process 3 has requested shutdown in %u seconds\n", counter);
      etimer_set(&et1, CLOCK_SECOND);
    }

    if(ev == PROCESS_EVENT_TIMER) {                5
      if(counter <= 0) {
        process_exit(&process3);                   6
```

examples/zolertia/tutorial/01-basics/04-processes.c

```
PROCESS_THREAD(process2, ev, data)
{
  PROCESS_BEGIN();
  printf("Process 2 started\n");

  while(1) {
    PROCESS_YIELD();
    if(ev == PROCESS_EVENT_EXITED) {                    ❶
      printf("* Process 3 has been stopped by Process 1!\n");
      etimer_set(&et2, CLOCK_SECOND * 5);               ❷
    }

    if(ev == PROCESS_EVENT_TIMER) {                     ❸
      printf("Process 2 is restarting Process 3\n");
      process_start(&process3, "Process 2");            ❹
    }
  }
```

examples/zolertia/tutorial/01-basics/04-processes.c

```
process_event_t event_from_process3;
```

```
PROCESS_THREAD(process3, ev, data)
{
  PROCESS_BEGIN();
  static char *parent;
  parent = (char * )data;                                    ❶
  static uint8_t counter;                                    ❷

  printf("Process 3 started by %s\n", parent);
  event_from_process3 = process_alloc_event();               ❸
  etimer_set(&et3, CLOCK_SECOND);

  counter = 0;

  while(1) {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et3));
    counter++;
    leds_toggle(LEDS_GREEN);

    if(counter == 10) {
      process_post(&process1, event_from_process3, &counter);  ❹
    }
    etimer_reset(&et3);                                      ❺
  }
  PROCESS_END();
}
```

examples/zolertia/tutorial/01-basics/04-processes.c

Create two processes:

One that increments a counter and toggles a LED. When the counter reaches ten (10), it should print a message to the screen.

A second process that each time the user button is pressed, it will change the color of the LED to be toggled (i.e from Green to Blue, then Red, and repeat).

examples/zolertia/tutorial/01-basics/04-processes.c

Conclusions

You should be able to:

- Create and compile a basic Contiki application
- Program a Zolertia device
- Implements timers, LEDs and buttons in your application
- Understand how Contiki processes and protothreads are implemented
- Create processes and tasks interacting with each other
- Identify connected devices, visualize the debug output in the screen

Antonio Liñán Colina

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

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