

Lab 1

Rafik Zitouni

ECE Paris

rafik.zitouni@ece.fr



Hardware - Zolertia (RE-Mote)

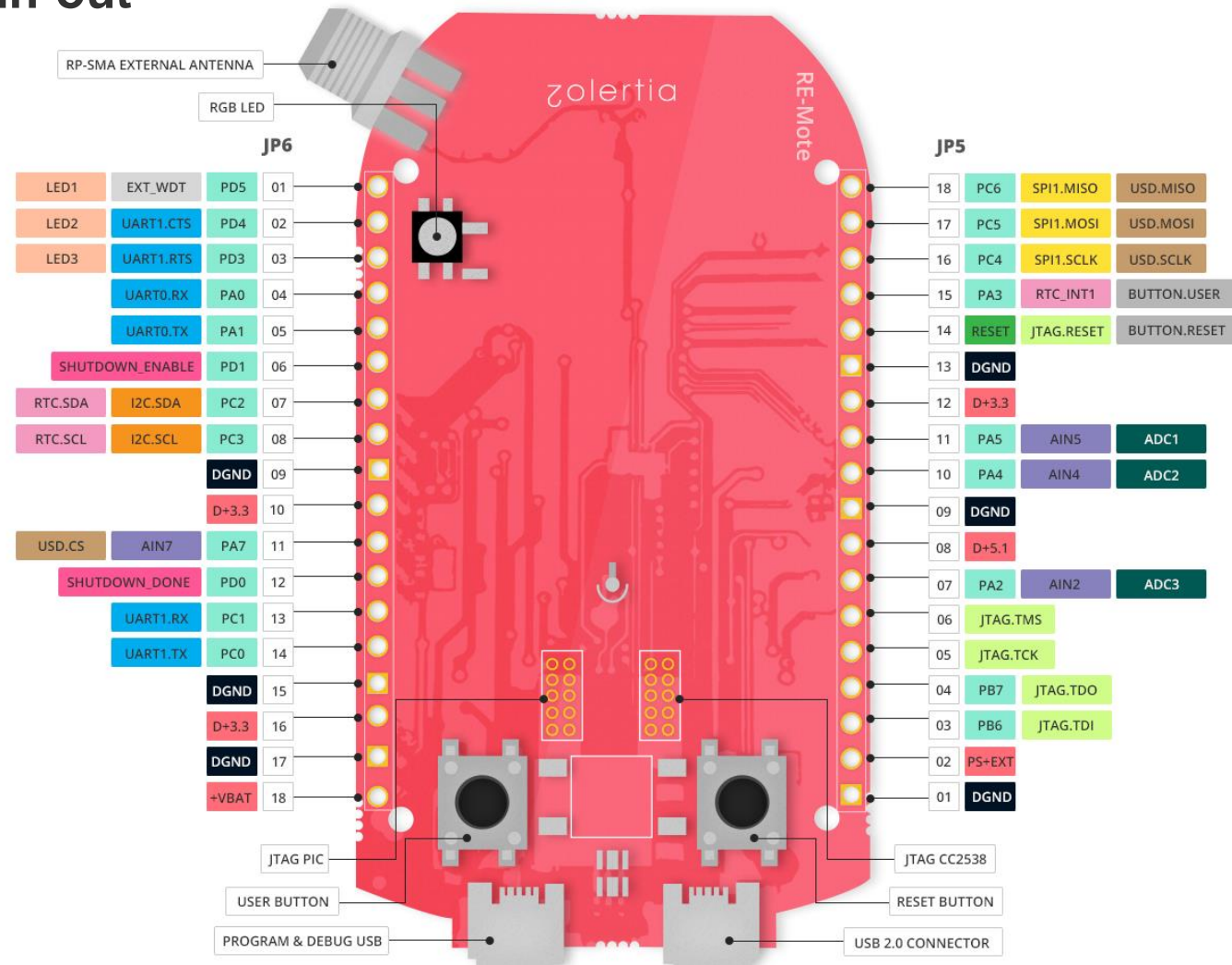
Zolertia Platform

- Zolertia RE-Mote hardware



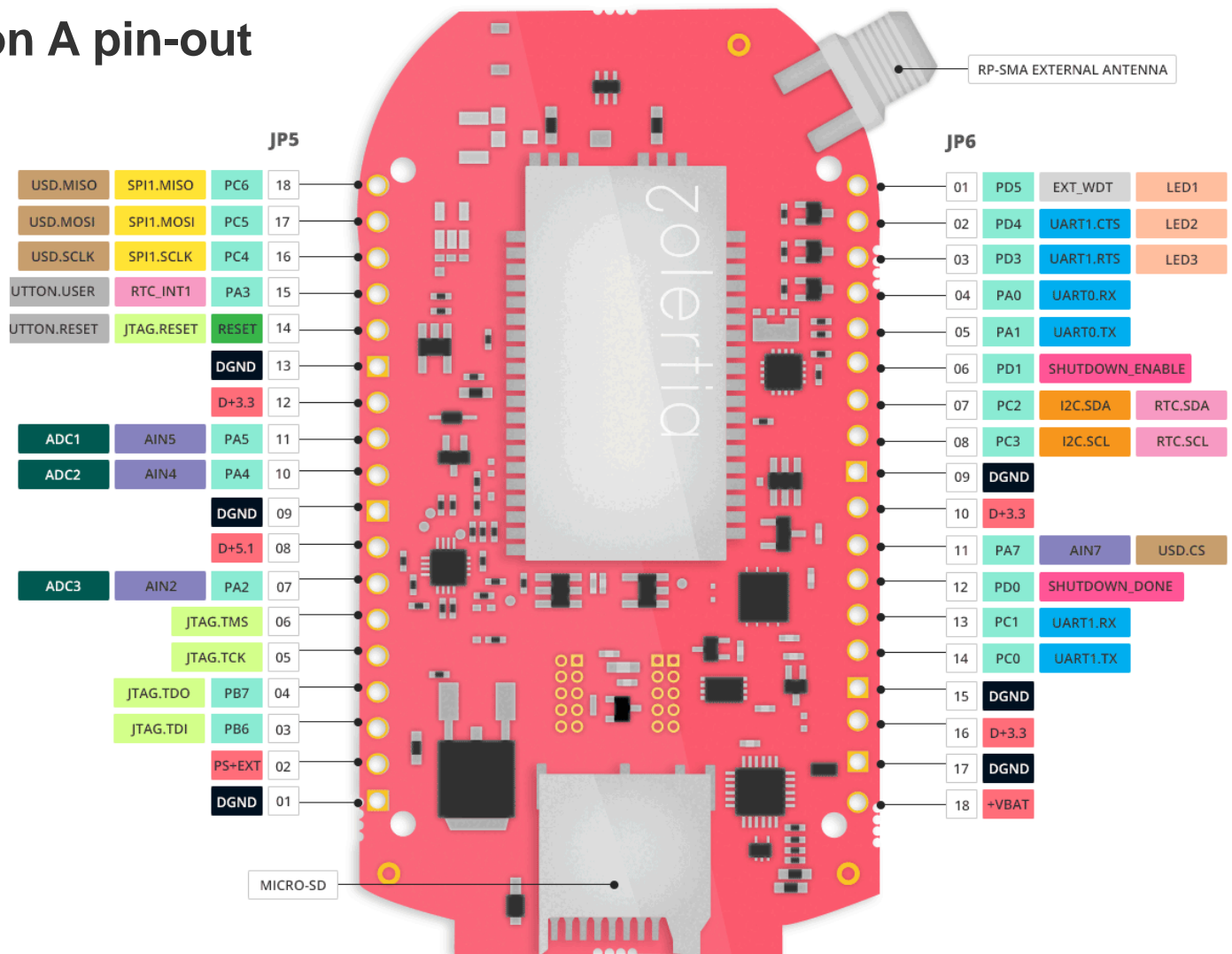
Hardware - Zolertia (RE-Mote)

RE-Mote revision A pin-out



Hardware - Zolertia (RE-Mote)

RE-Mote revision A pin-out

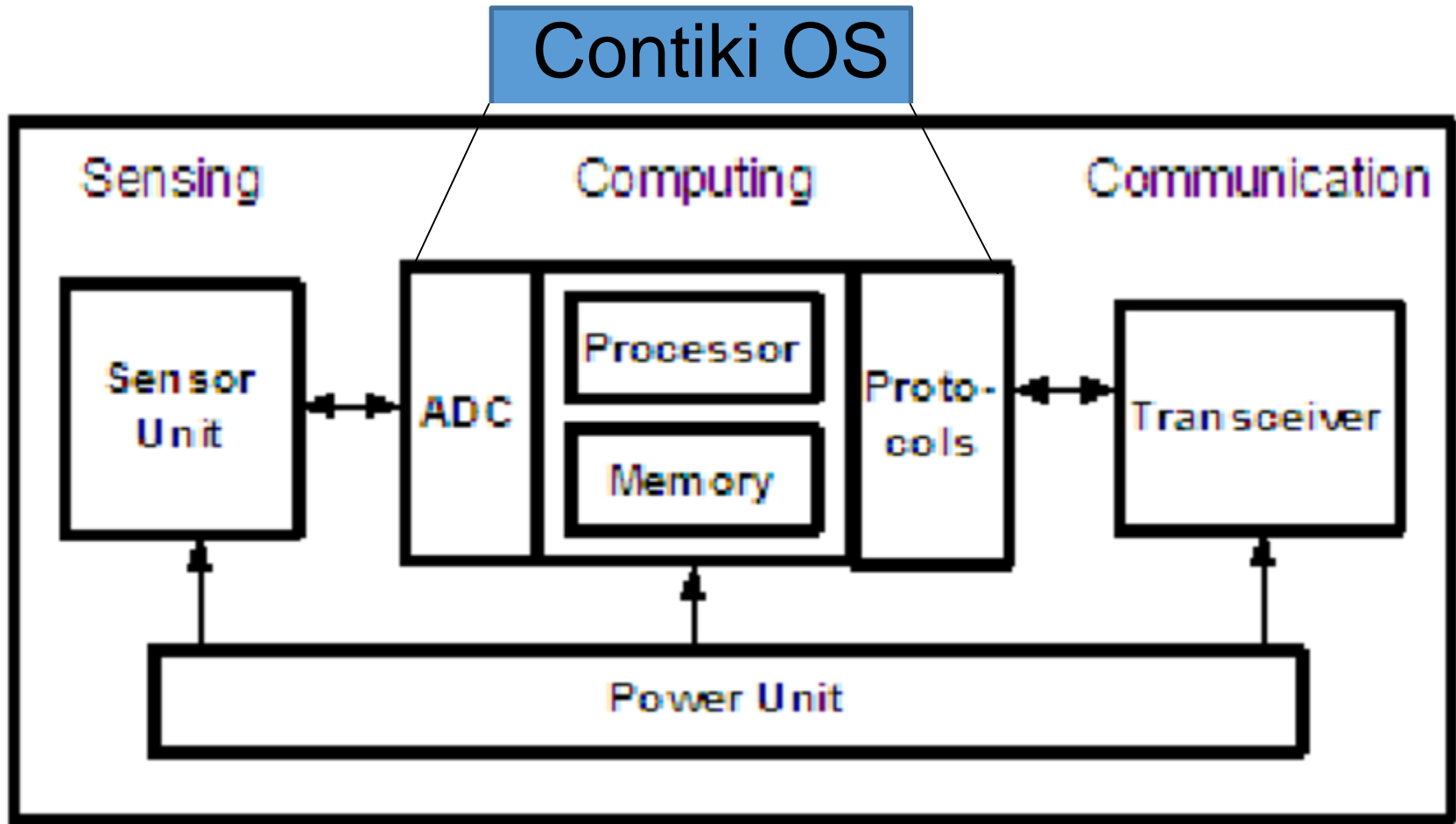


Hardware - Zolertia (RE-Mote)

Zolertia RE-Mote hardware

- ISM 2.4-GHz and 863-950-MHz ISM/SRD bands IEEE 802.15.4 & Zigbee
- **ARM Cortex-M3 32 MHz clock speed, 512 KB flash and 32 KB RAM** (16 KB retention)
- AES-128/256, SHA2 Hardware Encryption Engine
- ECC-128/256, RSA Hardware Acceleration Engine for Secure Key Exchange
- **User and reset button**
- Consumption down to 150 nA using the shutdown mode.
- Built-in battery charger (500 mA), **facilitating Energy Harvesting** and direct connection to **Solar Panels and to standards LiPo batteries.**
- Wide range DC Power input: 3.3-16 V.
- Small form-factor (73 x 40 mm).
- MicroSD (over SPI).
- Programmable RF switch to connect an external antenna either to the 2.4 GHz or to the Sub 1 GHz RF interface through the RP-SMA connector.

Hardware Architecture of Wireless Sensor Networks



Contiki OS

What is Contiki micro OS?

- Open source operating system for Internet of Things
- It supports IPv4, IPv6, 6LowPan
- Standardized C programming
- It supports Cooja simulator

You can get Contiki folder from my github

\$ git clone <https://github.com/zitouni/contiki.git>

Virtual Machine or Native Linux Ubuntu 14.04 64bits

On Virtual Machine every thing is done for you !

On Native Linux, you have to install toolchain

Contiki OS

Toolchain and required dependencies

%support graphviz and make tools

\$ sudo apt-get install curl graphviz unzip wget

\$ sudo apt-get install build-essential automake gettext % *automake*

% support ARM and MSP430 platforms

\$ sudo apt-get install gcc-arm-none-eabi

\$ sudo apt-get -o Dpkg::Options::="--force-overwrite" install gdb-arm-none-eabi

\$ sudo apt-get -y install gcc gcc-msp430

%support Cooja

\$ sudo apt-get install openjdk-7-jdk openjdk-7-jre ant

Test Contiki Installation

% MSP430 toolchain

\$ msp430-gcc --version

%ARM Cortex-M3 toolchain

\$ arm-none-eabi-gcc-4.9.3 --version

Contiki OS

Contiki Structure

Folder	Description	Zolertia files
examples	Ready to build examples	examples/zolertia, examples/cc2538-common
app	Contiki applications	-
cpu	Specific MCU files	msp430, cc2538
dev	External chip and devices	cc2420, cc1200
platform	Specific files and platform drivers	z1, zoul
core	Contiki core files and libraries	-
tools	Tools for flashing, debugging, simulating, etc.	zolertia, sky
doc	Self-generated doxygen documentation	-
regression-tests	nightly regression tests	-

Contiki applications -- Examples

Examples for RE-Mote hardware

examples/zolertia/zoul
examples/cc2538-common

Example1: We can start with a famous Hello world application

\$ cd contiki/examples/hello-world

% If build operation works well, you can run your hello-world.native

\$ sudo make hello-world TARGET=native

% Execute your program

\$./hello-world.native

% Make your application for zoul or RE-Mote

\$ sudo make hello-world.upload TARGET=zoul

% You can save your target

\$ sudo make savetarget=zoul

Contiki applications -- Examples

% If you would like to see what happen in your RE-Mote, tape a following command

\$ sudo make login TARGET=zoul

% If you have an error, please add the following package

\$ sudo apt-get install libc6-i386

*% You should see after introducing your \$ **sudo make login TARGET=zoul***

Hello, world

*% If it doesn't print "Hello, World", push your button **RESET***

If it's OK Go on !

Contiki OS -- Programming

Open your hello-world program

```
$ cd contiki/examples/hello-world/gedit hello-world.c
```

```
#include "contiki.h"
```

The name of the process

```
#include <stdio.h> /* For printf() */
```

Contiki starts process when it finishes booting

```
/*-----*/
```

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

```
AUTOSTART_PROCESSES(&hello_world_process);
```

```
/*-----*/
```

Declare the content of the process in the process thread (event handler and data handler)

```
PROCESS_THREAD(hello_world_process, ev, data)
```

```
{  
  PROCESS_BEGIN();
```

```
  printf("Hello, world\n");
```

Inside the thread you begin the process

```
  PROCESS_END();
```

```
}
```

Contiki OS -- Programming

Open your Makefile to understand it

1) `CONTIKI_PROJECT = hello-world`

Which application to compile

2) `all: $(CONTIKI_PROJECT)`

It will compile the defined applications

3) `CONTIKI = ../..`

Level indentation to contiki folder

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

Add makefile of contiki to the application

Contiki OS -- Programming

Example2: Create your project to make LEDs lightning

You need to create your Makefile in the same folder of your project.

```
#include "contiki.h"
#include "dev/leds.h"
#include <stdio.h>

/*-----*/
PROCESS(test_leds_process, "Test LEDs");
AUTOSTART_PROCESSES(&test_leds_process);
/*-----*/
PROCESS_THREAD(test_leds_process, ev, data)
{
    PROCESS_BEGIN();
    leds_on(LEDS_RED);
    PROCESS_END();
}
```

Contiki OS -- Programming

Available LEDs are :

LEDS_GREEN
LEDS_RED
LEDS_BLUE
LEDS_LIGHT_BLUE
LEDS_YELLOW
LEDS_PURPLE
LEDS_WHITE
LEDS_ALL

3-channel LEDs in a single device, allowing to show any color by the proper combination of Blue, Red and Green.

They are defined in **platforms/zoul/remote/board.h**

Contiki OS -- Programming

Example 2: Printing messages to the console

To debug your programs, you need to see what happening.

Try to see what happen when a ILEDs is lightning using:
Update your program as:

```
#include "contiki.h"
#include "dev/leds.h"
#include <stdio.h>
char hello[] = "hello from the mote!";
/*-----*/
PROCESS(test_leds_process, "Test LEDs");
AUTOSTART_PROCESSES(&test_leds_process);
/*-----*/
PROCESS_THREAD(test_leds_process, ev, data)
{
    PROCESS_BEGIN();
    leds_on(LEDS_RED);
    printf("%s\n", hello);
    printf("The LED %u is %u\n", LEDS_RED, leds_get());
    PROCESS_END();
}
```


%See what happen under your mote
\$ sudo make login target=zoul

Contiki OS -- Programming

Example 3: Adding Button events

- User button is seen as a sensor. It is defined in **core/dev/button-sensor.h** library
- Long press sequences can also be triggered. You can check the library and the example in **platform/zoul/dev/button-sensor.c** and **examples/zolertia/zoul/zoul-demo.c**

Wait() is used to wait for a button to be pressed. When a button is pressed, you get a message printed



```
#include "contiki.h"
#include "dev/leds.h"
#include "dev/button-sensor.h"
#include <stdio.h>
/*-----*/
PROCESS(test_button_process, "Test button");
AUTOSTART_PROCESSES(&test_button_process);
/*-----*/
PROCESS_THREAD(test_button_process, ev, data)
{
    PROCESS_BEGIN();
    SENSORS_ACTIVATE(button_sensor);
    while(1) {
        PROCESS_WAIT_EVENT_UNTIL((ev==sensors_event) &&
                                   (data == &button_sensor));
        printf("I pushed the button!\n");
    }
    PROCESS_END();
}
```

Wireless communication Applications

Broadcast application

- Sending your name to a **base station (or Sink)** with broadcast packets.
- **examples/rime**
- Build and upload application example-abc.
 - Change the string Hello with your name in **packetbuf_copyfrom("Hello", 6);**
 - 6 is the number of characters + 1 (terminator character)
- Log to your sink and see your name in the screen

\$ sudo make login TARGET=zoul

Wireless communication Applications

Unicast application

- It looks like broadcast, but destination address is needed. Open the file **example-unicast.c**. Locate the lines that contain the following:

```
packetbuf_copyfrom("Hello", 5);  
addr.u8[0] = 41;  
addr.u8[1] = 41;
```

- The program is configured to send data to node **41.41**. You need to change this Hex number to the address of your neighboring nodes.
- Compile and upload your program on your Motes

Wireless communication Applications

Data dissemination

- We will disseminate data from a single node to all other nodes when **the on-board user button** is pressed
- The application program **example-trickle.c**
- Try to understand the source code

Other examples

You can try other applications by understanding their functions in **/examples/zolertia/zoul**

Contiki OS -- Programming

Exercise 1:

Switch on the LED when the button is pressed. Switch off the LED when the button is pressed again.

Wireless communication Applications

Exercise 2:

- 1) Switch on the LED when your Mote receives a message. Use a Unicast and Broadcast application.
- 2) Connect your mote's battery to keep it far from your Sink mote (Mote connected to your host). What is the maximum coverage of your wireless communication?
- 3) Increase the output power of your transmitter? Measure the new maximum coverage of your transmitter?
- 4) Can you change the radio frequency channel? How?
Set the new channel equal to 23. Compare the transmission in 23 and 26 channel. Analyze your Packet Error Rate (PER) for each channel?
- 5) How can you transmit in the frequency band 868 MHz? Run the example zolertia/zoul/cc1200-demo and measure the new possible coverage of your zolertia motes. You compare this coverage to the first obtained one.

Wireless communications Re-Mote

Contiki OS folders related to zolertia boards:

- **platform/zoul/remote/dev/board.h** contains hardware settings (I2C pins and ADC channels configurations)
- **platform/zoul/contiki-conf.h** contains UART, MAC, radio channels, IPv6, RIME and network buffer configuration.

Devices addressing

- **RE-Mote** comes with two pre-loaded MAC addresses stored in its internal flash memory
- **platform/zoul/contiki-conf.h** defines the switches between hardcoded addresses.
- If using your own hardcoded address

```
#ifndef IEEE_ADDR_CONF_ADDRESS
#define IEEE_ADDR_CONF_ADDRESS { 0x00, 0x12, 0x4B, 0x00, 0x89, 0xAB, 0xCD, 0xEF }
#endif
```

Wireless communications Re-Mote

Bandwidth and channel

OPTIONS FOR FREQUENCY ASSIGNMENTS			
Geographical regions	Europe	Americas	Worldwide
Frequency assignment	868 to 868.6 MHz	902 to 928 MHz	2.4 to 2.4835 GHz
Number of channels	1	10	16
Channel bandwidth	600 kHz	2 MHz	5 MHz
Symbol rate	20 ksymbols/s	40 ksymbols/s	62.5 ksymbols/s
Data rate	20 kbits/s	40 kbits/s	250 kbits/s
Modulation	BPSK	BPSK	Q-QPSK

Wireless communications Re-Mote

Change channel or frequency

```
#ifndef CC2538_RF_CONF_CHANNEL
#define CC2538_RF_CONF_CHANNEL          26
#endif /* CC2538_RF_CONF_CHANNEL */
```

Try to find **CC2538_RF_CONF_CHANEL** using **grep** command

To change the channel from the application use **RADIO_PARAM_CHANNEL** as follows:

```
rd = NETSTACK_RADIO.set_value(RADIO_PARAM_CHANNEL, value);
```

value can take values from **11 to 26**

rd can be **RADIO_RESULT_INVALID_VALUE** or **RADIO_RESULT_OK**

Wireless communications Re-Mote

- It uses **CC2538** transceiver built-in **2,4 GHz**.
 - The default output power is fixed to 2mW (3 dBm).
 - Check a file **cpu/cc2538/cc2538-rf.h** to change output power

TX Power (dBm)	Value
+7	0xFF
+5	0xED
+3	0xD5
+1	0xC5
0	0xB6
-1	0xB0
-3	0xA1
-5	0x91
-7	0x88
-9	0x72
-11	0x62
-13	0x58
-15	0x42
-24	0x00

Wireless communications Re-Mote

For **863-950 MHz** band, it uses **CC1200** RF transceiver.

TX Power (dBm)	Value
+14	0x7F
+13	0x7C
+12	0x7A
+11	0x78
+8	0x71
+6	0x6C
+4	0x68
+3	0x66
+2	0x63
+1	0x61
0	0x5F
-3	0x58
-40	0x41

These values are recommended
They can be introduced using
dev/cc1200/cc1200-const.h file.

Wireless communications Re-Mote

Checking the wireless link

How to measure the quality of radio link ?

- **ETX metric**, or expected transmission count, is a **measure of the quality of a path between two nodes** in a wireless packet data network.
- An **ETX of one** indicates a perfect transmission medium, where an **ETX of infinity** represents a completely non functional link.
- **RSSI (Received Signal Strenght Indicator)** determine the amount of radio energy received in a given channel. It is measured by :

```
rd = NETSTACK_RADIO.get_value(RADIO_PARAM_RSSI, value);
```

value is pointer to store the RSSI value and **rd** is

RADIO_RESULT_INVALID_VALUE or **RADIO_RESULT_OK**

Wireless communications Re-Mote

Link Quality Indicator (LQI) is a digital value provides by Chipset vendors as an indicator of how well a signal is demodulated, in terms of the strength and quality of the received packets

```
rd = NETSTACK_RADIO.get_value(PACKETBUF_ATTR_LINK_QUALITY, value);
```

value is a variable passed as a pointer to store the LQI and **rd** it will be either **RADIO_RESULT_INVALID_VALUE** or **RADIO_RESULT_OK**