A comprehensive guide to using, programming & flashing the BITtalino R-IoT WiFi sensor module

Ver 1 (March-July 2017)

Emmanuel FLETY - Prototypes & Engineering Team (PIP) - IRCAM

Edition 1.0 - March 2017

Introduction

The R-IoT module is 7th generation of IRCAM's wireless sensor digitizing unit, an essential tool linking motion sensing, gestural recognition and Live Performance Arts.

Since 2001, IRCAM aimed to provide low latency, high data-rate & resolution and stage compatible devices capable of streaming gestural information to the computer from multiple performers. As development progressed, we also targeted a smaller form factor and longer runtime.

Lastly, as the Makers movement arose in 2007-2008, we also privileged a platform approach for our devices, allowing a standardized firmware development and upload scheme to allow collaborators to easily work and customize the unit without dealing with low-level, microcontroller detailed programming.

As a result, we selected a core processing unit capable of implementing all those aspects The R-IoT module is based upon the CC3200 chip from Texas Instrument. Its core feature is compatible with Energia, a branch of the Arduino IDE that allows programming TI processors with the easiness of the Arduino look & feel.

http://energia.nu/

Just like the Arduino system for Atmel microcontrollers, Energia is essentially a code warper on top of the C API from Texas Instrument to program the core processor though R-IoT can still be programmed with TI tools chain & Code Composer.

The CC3200 is actually the combination of two ARM processors, one taking care of the WiFi modem and network stack, the second being the application processor running the user code.

The CC3200 embeds a bootloader triggered upon reset / power on by an external momentary switch. The firmware is uploaded to the chip using a simple a 3.3V USB serial cable such as the TTL-232R-3V3 from FTDI (Farnell part # 1329311).

BITalino R-IoT particulars

The BITalino R-IoT is a slightly different version of the former R-IoT from IRCAM's lab team ISMM. Our partnership with pluX aimed to refine a few aspects of the original design to match with other populations of users, and some new use cases too.

The BITalino R-IoT was made compatible with the BITalino ecosystem and sensor platform as a WiFi transmitter, bridge and data hub. It captures pre-digitized BITalino sensors and packs them in an Open Sound Control message in addition of the IMU data, which are sent in a separate OSC message.

In addition, a new form factor of the module allows for soldering additional sensors or accessories (switch, LED, piezo) to expand its possibilities. Finally, an RGB LED was added for an enhanced visual interaction with the user, easing the boot & connection sequence (wifi connection, streaming).

The BITalino R-IoT exports only some of the available I/O's from the CC3200 for matters of size / form factor. It remains small enough to be installed anywhere, on the body, on an instrument or in an object. With the additional I/Os, which include 3 analog inputs and 3 PWM output and 10 GPIOs total (including the RGB LED), it can be adapted to various contexts.

An i^2c bus is also exported allowing for a complete chain of accessories to be talked to, from LED controllers to additional sensors or I/O extenders.

The BITalino R-IoT also embeds a 9 axis digital sensor featuring a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer, allowing for instance the onboard computation of the absolute orientation of the module in space. The sensor is attached to the SPI port to grab the 16 bit motion data at high speed.

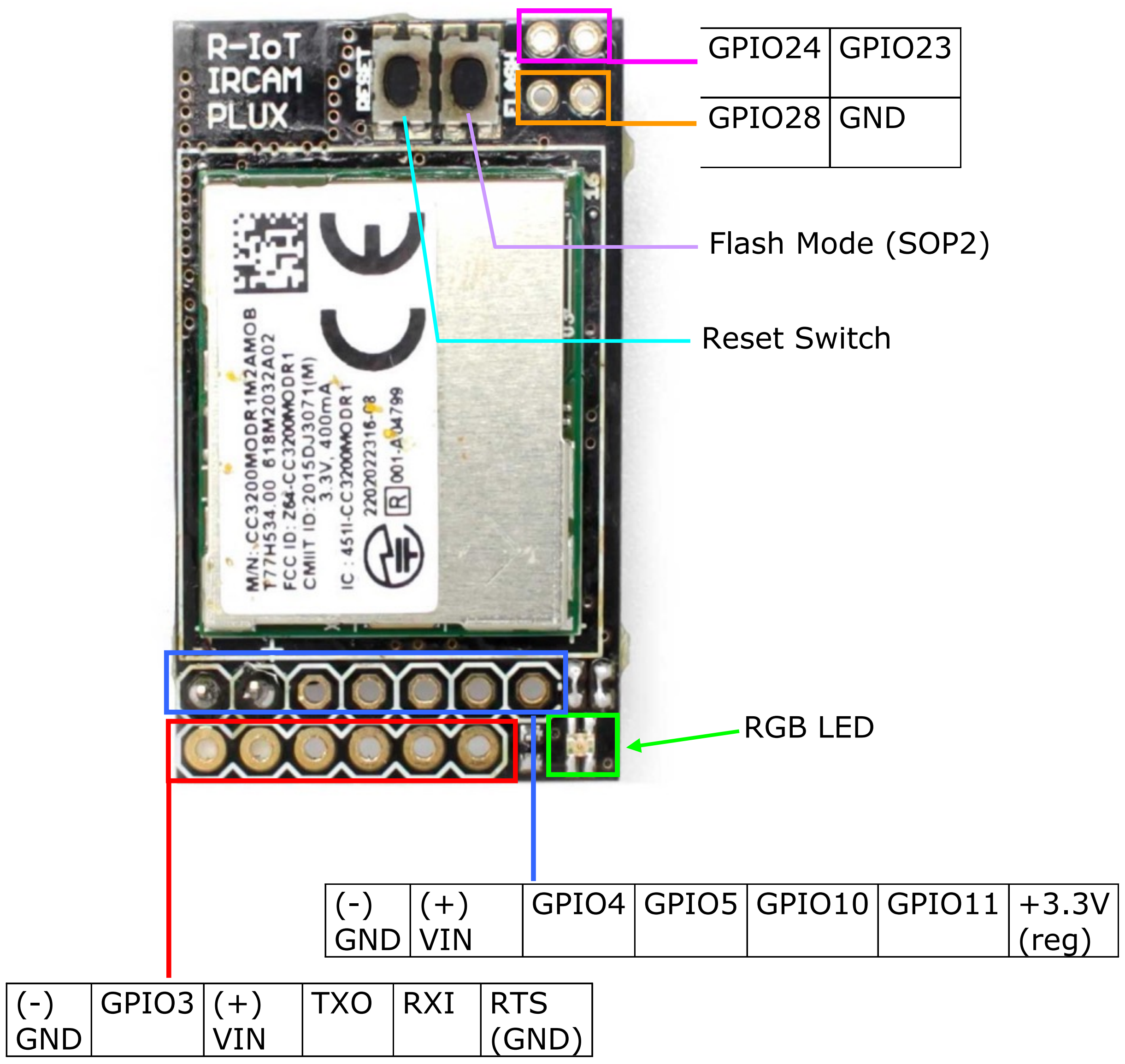
The IMU sensor data are locally processed, analyzed and streamed over WiFi (UDP, Open Sound Control) by the module using its firmware, a program compiled using the Energia tool chain.

Lastly, the BITalino R-IoT edition is meant to be used either as a replacement of the Bluetooth module provided with the BITalino sensor board or as a standalone using (as a wireless WiFi IMU). The pinout of the module reflects this specificity compared to the original R-IoT module and features a 6 pin 2.54mm header that provides the power supply and serial communication to the BITalino board or to a FTDI USB serial cable (with crossed / null-modem wiring).

To ease the reading of this document, we will refer from now to the BITalino R-IoT as R-IoT.

Module pinout and wiring

Front View

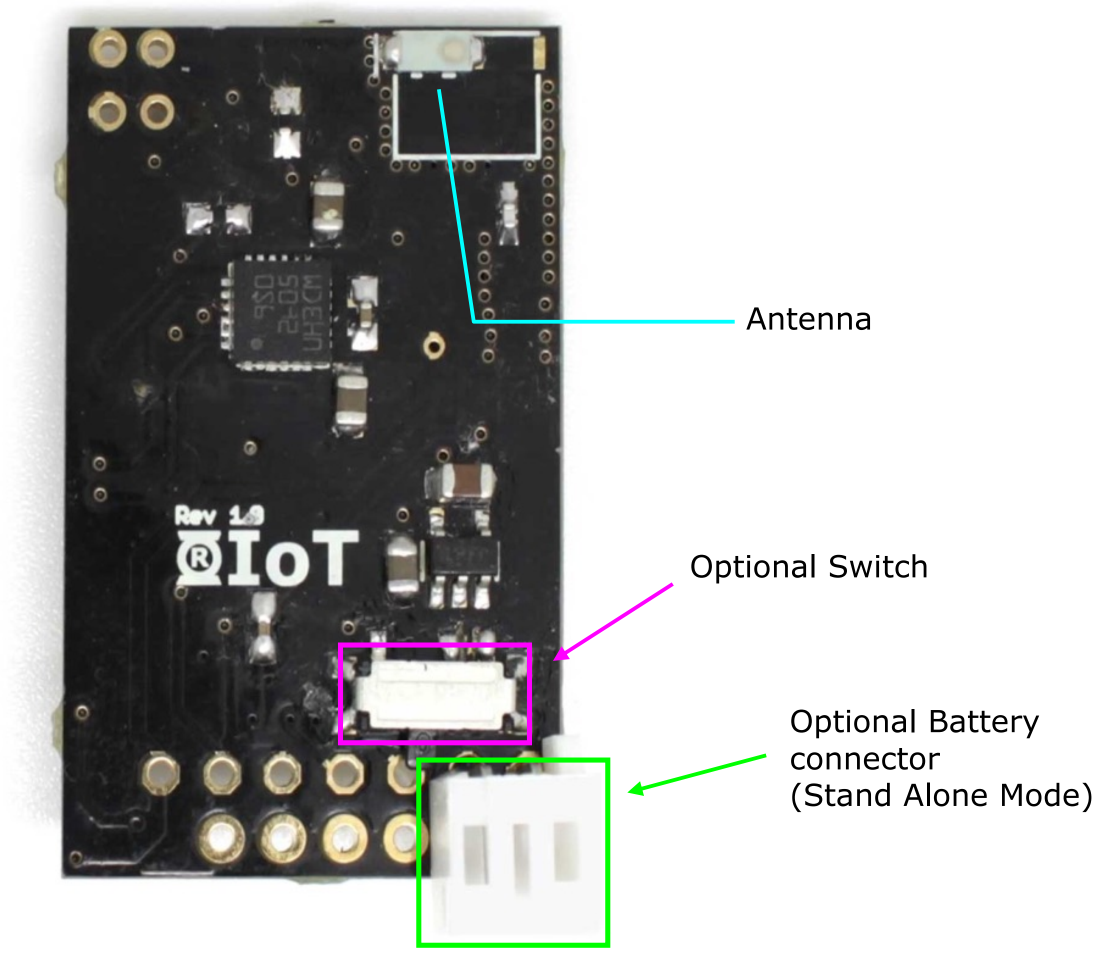


GPIO24	GPIO23
GPIO28	GND

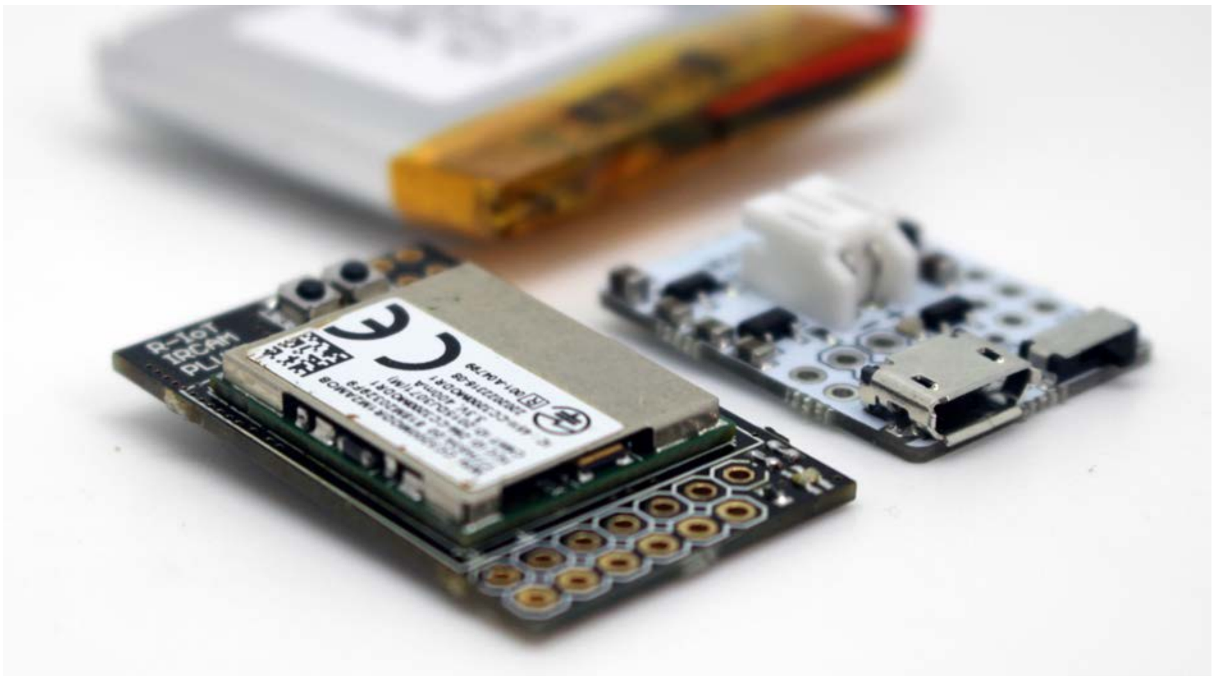
(-) GND	(+) VIN	GPIO4	GPIO5	GPIO10	GPIO11	+3.3V (reg)

(-) GND	GPIO3	(+) VIN	TXO	RXI	RTS (GND)

Back View


Antenna
Optional Switch
Optional Battery connector (Stand Alone Mode)
Rev 1.0
IoT

Installing and powering R-IoT with BITalino



Battery Charging

The battery should be charge using the accessory (right on the above picture) using the provided USB cabe.

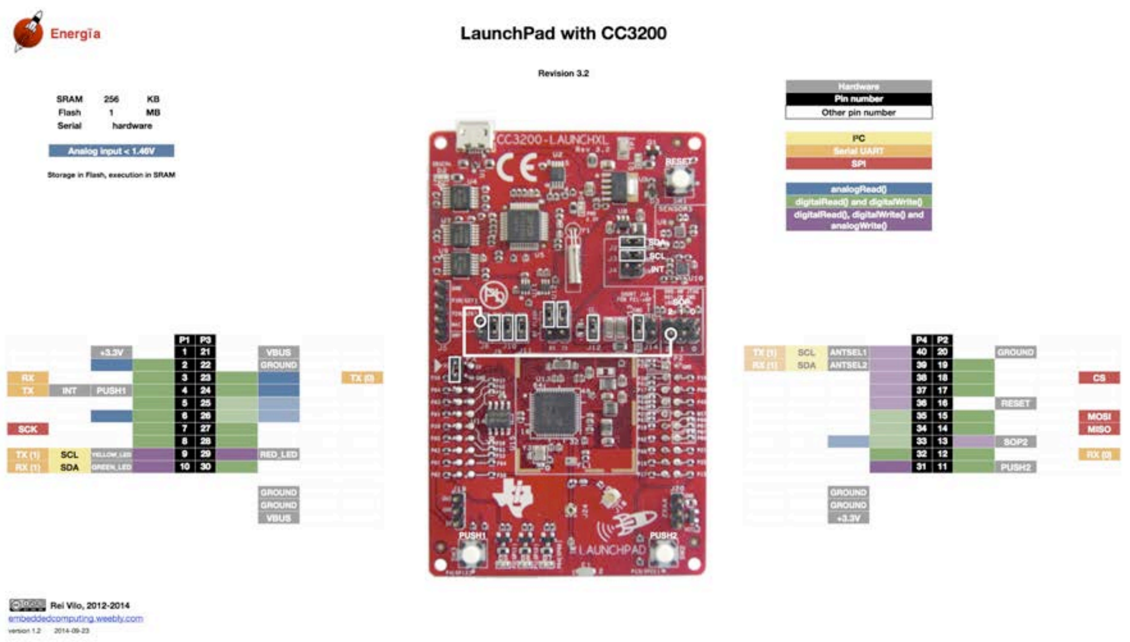
Install

The charged battery must be then connected to the R-IoT

Understanding the physical platform

We have developed the R-IoT platform using initially the Texas Instrument CC3200 launchpad, a development board for which Energia has standardized and named / numbered all the I/Os based on the physical header and pinhead connectors of the dev board.

http://energia.nu/pin-maps/guide_cc3200launchpad/



In order to remain 100% compatible with the former pin numbering, we kept the same numbering logic of the CC3200 I/Os in our code and firmware. The following picture details which I/Os are exported along with their #. That number is the same used to access to the physical pin from the program written in Energia.

However, to keep the I/O number user's friendly, a series of #define provide the translation of the GPIO number and the launchpad board.

R-IoT configuration & setup

Default configuration and OSC messages

R-IoT default firmware implements a WiFi Access Point (AP mode) and a Web server to configure is wireless parameters such as the WiFi network name (SSID), the Open Sound Control UDP port to send data to, the recipient computer IP address etc.

The default SSID is "riot"; for a quick install, the simplest is to configure your router and computer rather than edit the R-IoT configuration in first place (see further).

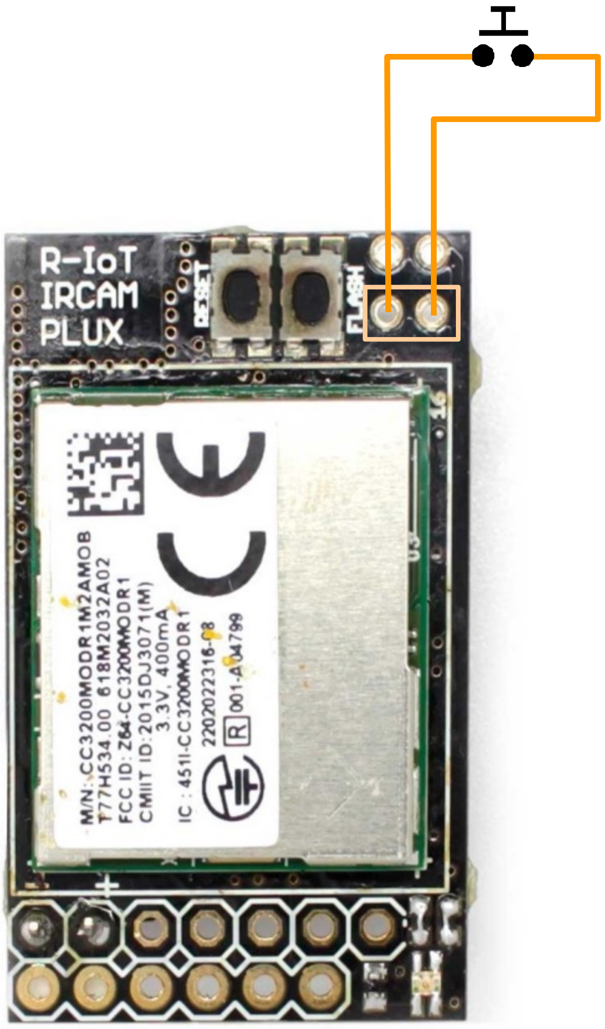
By default, The R-IoT will be sending **OSC data** to the DEST IP (destination IP address) **192.168.1.100** on port **8888.** The other default parameters are reported on page 9 (bottom figure)

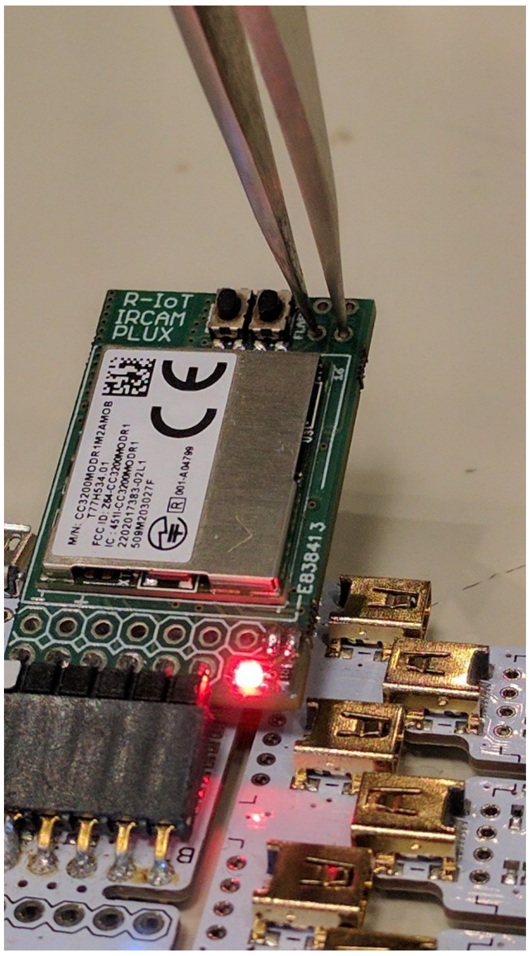
The OSC message structure is described in section R-IoT Streaming, Sensor Fusion and Analysis, on page

Changing the default configuration

To trigger the configuration mode, GPIO28 must be shorted when the module is powered up.

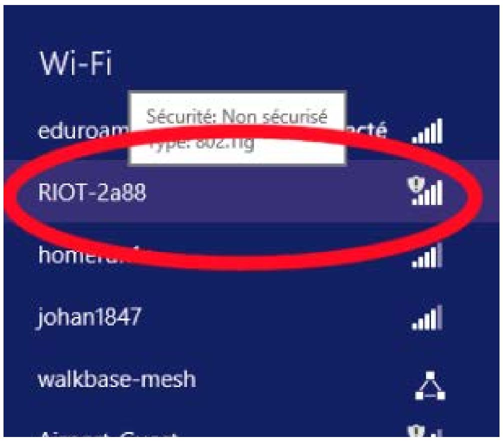
A switch ca be wired to those pins or, if only a casual configuration is required, a pair of sharp tweezers can used to short the 2 pads.



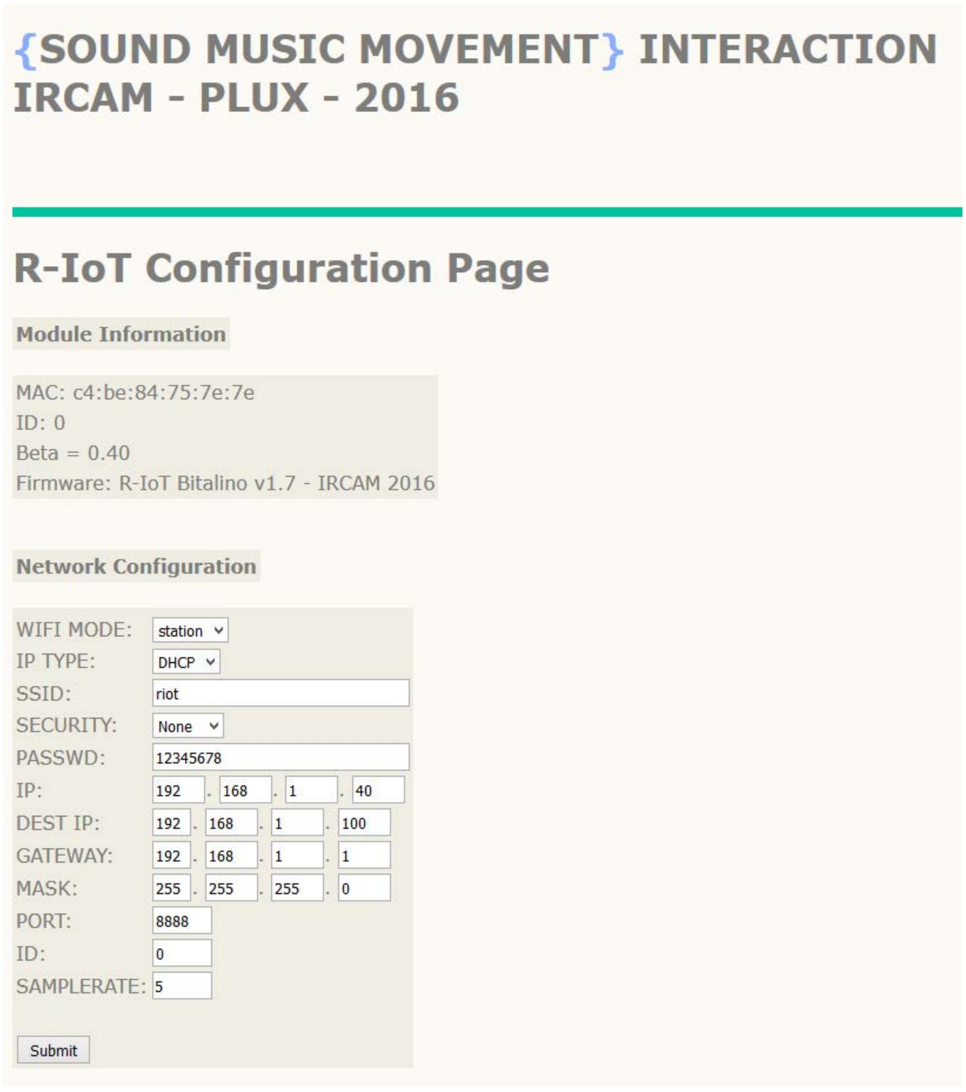


The onboard RGB led will turn red and will start flashing rapidly. After one second, it will turn blue then green indicating the R-IoT has created the WiFi access point.

The Wifi network name can be easily identified in the list of available SSID and starts with RIOT (in uppercase) followed by a 4 digit random number.



- Set your computer in DHCP and lookup the wifi network proposed by the module
- Connect to the network (no security needed)
- Open your web browser and open the URL http://192.168.1.1
- That should take you to the web page hosted by the module where you can configure its network behavior & parameters



{SOUND MUSIC MOVEMENT} INTERACTION
IRCAM - PLUX - 2016

R-IoT Configuration Page

Module Information

MAC: c4:be:84:75:7e:7e
ID: 0
Beta = 0.40
Firmware: R-IoT Bitalino v1.7 - IRCAM 2016

Network Configuration

WIFI MODE:	station
IP TYPE:	DHCP
SSID:	riot
SECURITY:	None
PASSWD:	12345678
IP:	192 . 168 . 1 . 40
DEST IP:	192 . 168 . 1 . 100
GATEWAY:	192 . 168 . 1 . 1
MASK:	255 . 255 . 255 . 0
PORT:	8888
ID:	0
SAMPLERATE:	5

The R-IoT connects to the network name set in the SSID field above. It should match the network name set in your WiFi access point (see further in this document for configuring your AP).

Most of the default settings should work with your WiFi infrastructure. We suggest keeping the R-IoT in station mode (AP is provided by the infrastructure router).

If some security is absolutely needed, WPA-2 can be enabled along with a password or passphrase, however security uses more bandwidth and can be trickier to configure for novice users. If the goal is to prevent unwanted access to the local network during sensor streaming and collection, it is actually easier to setup a MAC address filtering in the Access Point itself. In addition, the SSID beaconing can be disabled (also on the Access point) to prevent finding the name of the network, therefore allowing connection only to parties who know the said SSID name.

In DHCP mode, there's no need to fill up the gateway and mask fields, those will be provided by their DHCP server.

The only other important parameters to be chosen are the recipient IP address (the IP of the computer receiving the data) and the UDP port.

The ID is used to ease the data routing when identifying the R-IoT in the software application. We suggest using both a port **and** ID based routing in order to ensure the proper identification of the data flow source. A simple module numbering / convention makes it easy to remember.

For example :

- R-IoT module #0 uses port #8000
- R-IoT module #1 uses port #8001
- R-IoT module #8 uses port #8008

etc..

In the case the software application has limited flexibility for choosing the UDP port (or when there's a single UDP port available for all input flows), the ID is then use to separate and route the different modules.

Once the parameters are all set in the configuration page, click submit and setup will be saved in the non-volatile memory of the R-IoT module, which can be rebooted to apply the new settings (cycle the power supply or press the reset switch).

WiFi and Computer setup

As explained above, the R-IoT can work (almost) out-of-the-box once the computer and and WiFi access point have been configured to work together.

The R-IoT can operate on both a local network or over internet if necessary. As already said above, the R-IoT it will be sending data to its DEST IP (destination IP address), identified in the field of the configuration web page, which is defaulted to 192.168.1.100.

Therefore, a local network must be created using the AP router, and the proper destination address set on the computer.

Change Computer IP

On MacOS, go in the system preferences then network preferences. Select the ethernet connection and change the IP address method to "manually" and set it to 192.168.1.100. Set the network mask to 255.255.255.0.

On Windows, use the network control center and adjust the card parameter (select the ethernet connection then TCP/IP parameter) and use the same values as above.

On both operating systems, you can also use network profiles that allows quick switching between your regular internet configuration (ie using DHCP for most of us) and a dedicated sensor configuration to play with the R-IoT over a local network (using a manually set IP address). MacOS has a profile system in its network preferences. On Windows you might have to install a free software to manage the profiles, like NetsetMan.

If you further need to change the destination IP, just make sure the R-IoT module is also configured to send to the right one. Obviously, the selected IP address must match the network base address (192.168.1.XXX in our case) so that the OSC messages reach the recipient.

Configure the WiFi Access point and router

We open a can of worm here as it's not "so trivial" for non-experimented users. Configuring a device IP address while not knowing which IP address has been already assigned to it makes the whole thing more complicated than it looks at first glance.

To make it more confusing each brand or manufacturer tend to have its own default configuration. We will attempt to explain the configuration process in layman's words but you will possibly need to refer to the user's manual of the Access Point you've purchased to find out about the defaults.

Access the WiFi Access point the first time

An access point is most of the time pre-configured to be only a very few steps away from running out-of-the-box. The essential information to know (provided by the user's manual of the device) is its IP address, so that you can connect to the device configuration page using your favorite browser.

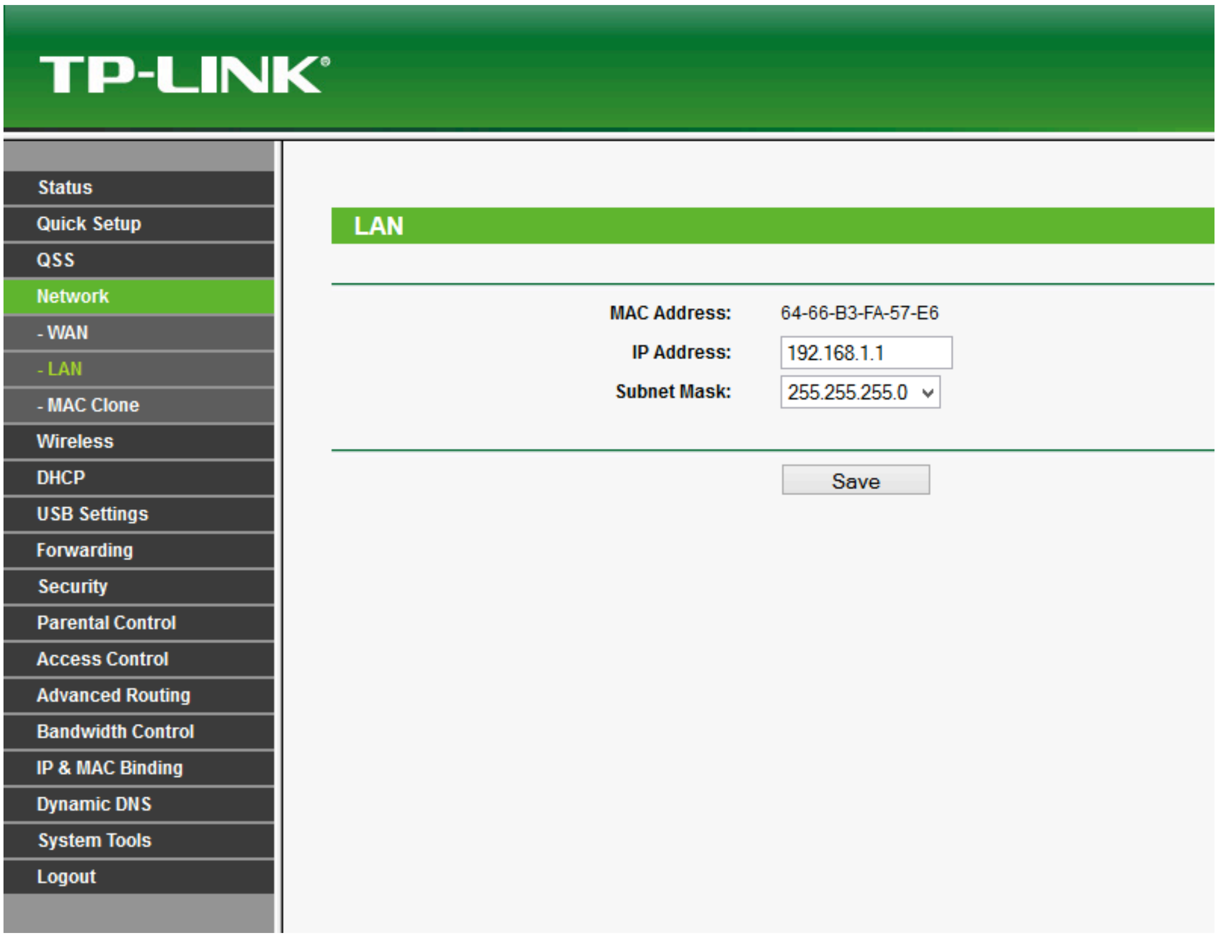
When taken straight out of the box, the AP is usually configured with DHCP enabled for clients and with a fixed IP address which can be (most of the time) 192.168.1.1 or 192.168.0.1 **or in the case or the mini TP-link MR3020 3G/Wifi router, 192.168.0.254.**

If the AP indeed provides a DHCP service, you can leave your default configuration (like when using Internet). If the AP doesn't use DHCP, you will need a manual address to be able to find it on the local network. In the example above, you would use for instance 192.168.0.XXX where the final IP address number would be anything between 1 and 253.

Once you're all set, you can access the configuration page of the AP, open your browser and connect to the default address of the device (192.168.0.254 in the example above).

You will be usually prompted for a login and password which is explicitly provided in the AP user's manual. The TP-link either ship with admin/admin or admin (login) and no password (leave blank).

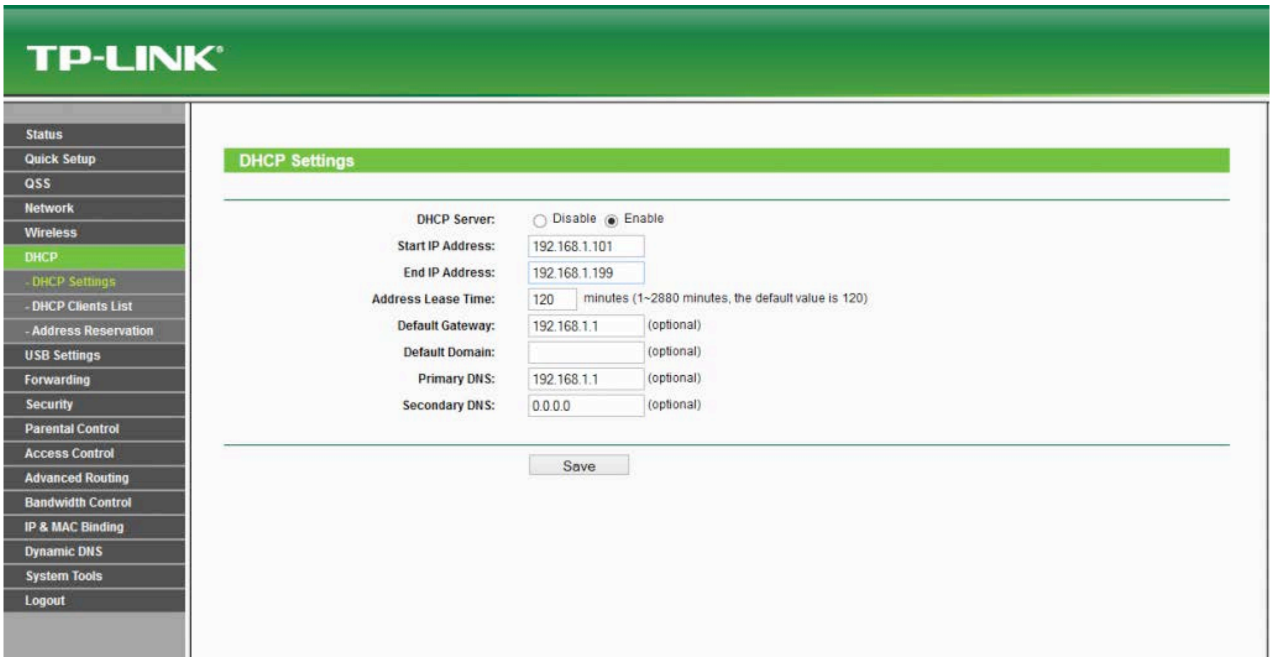
The default configuration of R-IoT is to connect to a computer with the fixed IP address 192.168.1.100. If the router / AP configuration is set to 192.168.0.1, there will be an address mismatch, therefore go in the Network -> LAN page and change the AP address to 192.168.1.1 or 192.168.1.254 which is kind of standard



LAN

MAC Address: 64-66-B3-FA-57-E6
IP Address: 192.168.1.1
Subnet Mask: 255.255.255.0

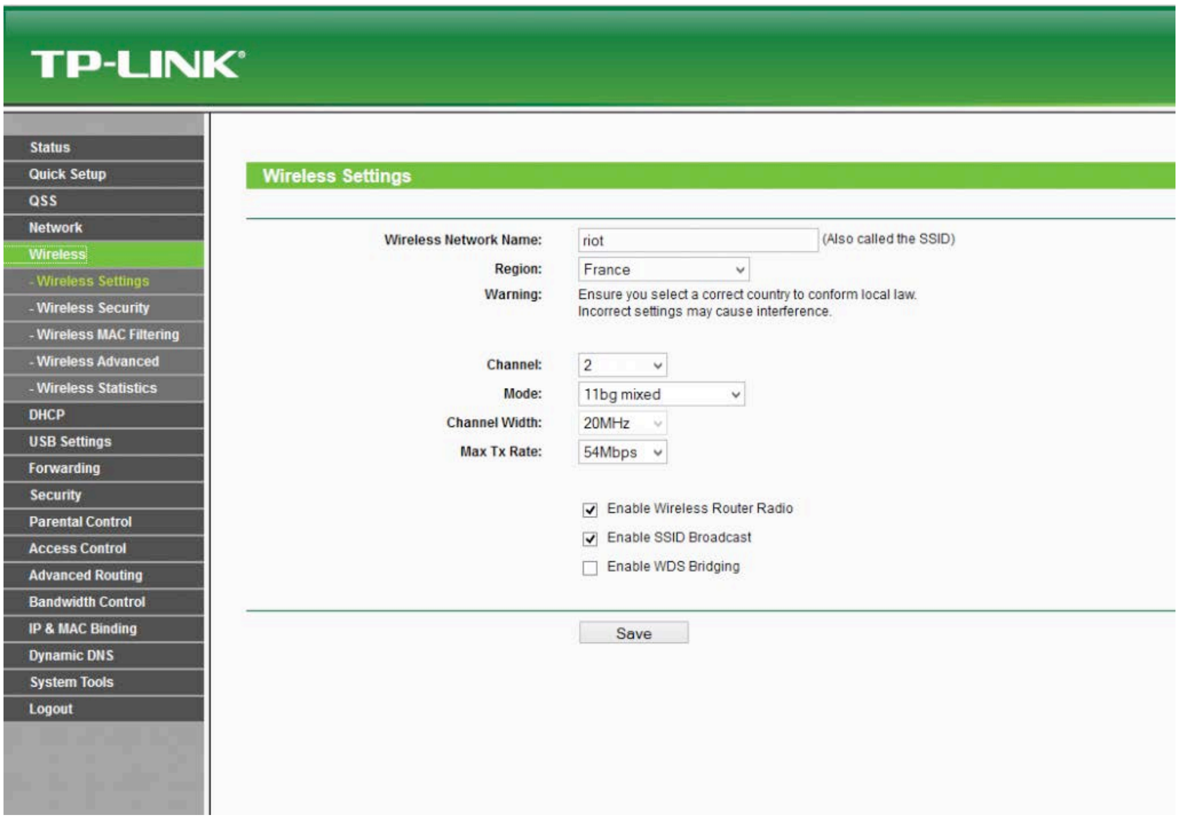
That will usually require a reboot of the AP, proceed and log in once again.
– Tune the DHCP settings. Most of the time the DHCP will be active on addresses from 192.168.1.100 to 199. Change this from 110 to 150, anything to avoid having the default destination IP address (192.168.1.100) to be in the DHCP lease pit. Alternatively you can leave it in the DHCP area and reserve the address based on the MAC address of your computer (see DHCP address reservation)



DHCP Settings

DHCP Server:	Disable [✓ Enable]
Start IP Address:	192.168.1.101
End IP Address:	192.168.1.199
Address Lease Time:	120 minutes (1~2880 minutes, the default value is 120)
Default Gateway:	192.168.1.1 (optional)
Default Domain:	(optional)
Primary DNS:	192.168.1.1 (optional)
Secondary DNS:	0.0.0.0 (optional)

Finally go in the Wireless settings, change the SSID of your network. If you are using the R-IoT defaults, use riot as the network name.
Select a wifi channel that is possibly different than your neighbour. Finally, go in the security section and disable encryption (WPA2 encryption is supported but not recommended on first use).



Wireless Settings

Wireless Network Name:	riot (Also called the SSID)
Region:	France
Warning:	Ensure you select a correct country to conform local law. Incorrect settings may cause interference.
Channel:	2
Mode:	11bg mixed
Channel Width:	20MHz
Max Tx Rate:	54Mbps

- [✓] Enable Wireless Router Radio
- [✓] Enable SSID Broadcast
- [ ] Enable WDS Bridging

Once the router is configured with this 192.168.1.xxx address scheme and DHCP, the computer IP address can be set to 192.168.1.100, the default destination address the R-IoT module sends data to.

Local Network Conventions and Standards

This IP address configuration might sound confusing at first but you won't need to go back to it most of the time. You've just created what is possibly your first local network (congratulations !) and moving between Internet and your sensors will soon become some easy gym !

There's no absolute standard on what to use as your IP address and local network IP scheme. "Our" IRCAM standard using a 192.168.1.100 recipient IP address inherits from the hardware we designed back in 2004 using Linksys WiFi routers which were on 192.168.1.XXX by default.

Moreover, another classic network numbering uses the scheme 10.0.0.XXX or similar. The important thing is to understand your 3 devices (the R-IoT, the wifi access point and the computer) must be on the same scheme to be able to "see each other". In addition, the R-IoT must know to which address it has to send data to (the computer IP).

Ideally you will label your AP with a stiff sticker to remember its IP and find an easy way to remember the ID and port configuration of your R-IoT module.

Once all set, the best practice is to keep the AP and the R-IoT modules paired and stored together to avoid any mismatch.

R-IoT Streamin, Sensor Fusion and Analysis

Sensor fusion

Calibration of the sensors and the Data fusion allowing for computing the quaternions and Euler angles are provided. The data fusion is implemented using the open-source code provided by Sebastian Madgwick[1].

R-IoT code repository

Several code examples dedicated to the R-IoT platform are available on the Bitalino GIT repository https://github.com/BITalinoWorld and on Ircam's GIT repository: https://github.com/Ircam-R-IoT

The repository contains the main firmware, which achieves several analysis on the sensor data and allow the configuration of the module (IP address, UDP port, module ID) via a web server or the USB serial port.

Other simpler examples show how to write dedicated code for the R-IoT platform for specific motion analysis for instance.

Being a development platform, the R-IoT module role isn't bound to motion analysis or sensor data streaming. Using the additional I/Os and analog input and the WiFi modem, the unit can be turned into an efficient Internet Of Things (IoT) object, a miniature web server, a car alarm or a plant watering system.

Receive sensors data from the module

Below is an data receiving in max-msp. When used along with BITalino sensors, the module forges 2 single OSC packets containing all the exported data (IMU + BITalino). When used in standalone mode, only the IMU data are streamed within a single OSC message.

The IMU OSC message starts with **/<ID>/raw** followed by a list of 21 float numbers that split as the following (physical units sent as I.S. units)

- 3 axis accelerometer (1 float per axis) {-8 ; +8} g
- 3 axis gyroscope {-2 ; +2} °/s
- 3 axis magnetometer {-2 ; +2} gauss
- Temperature °C

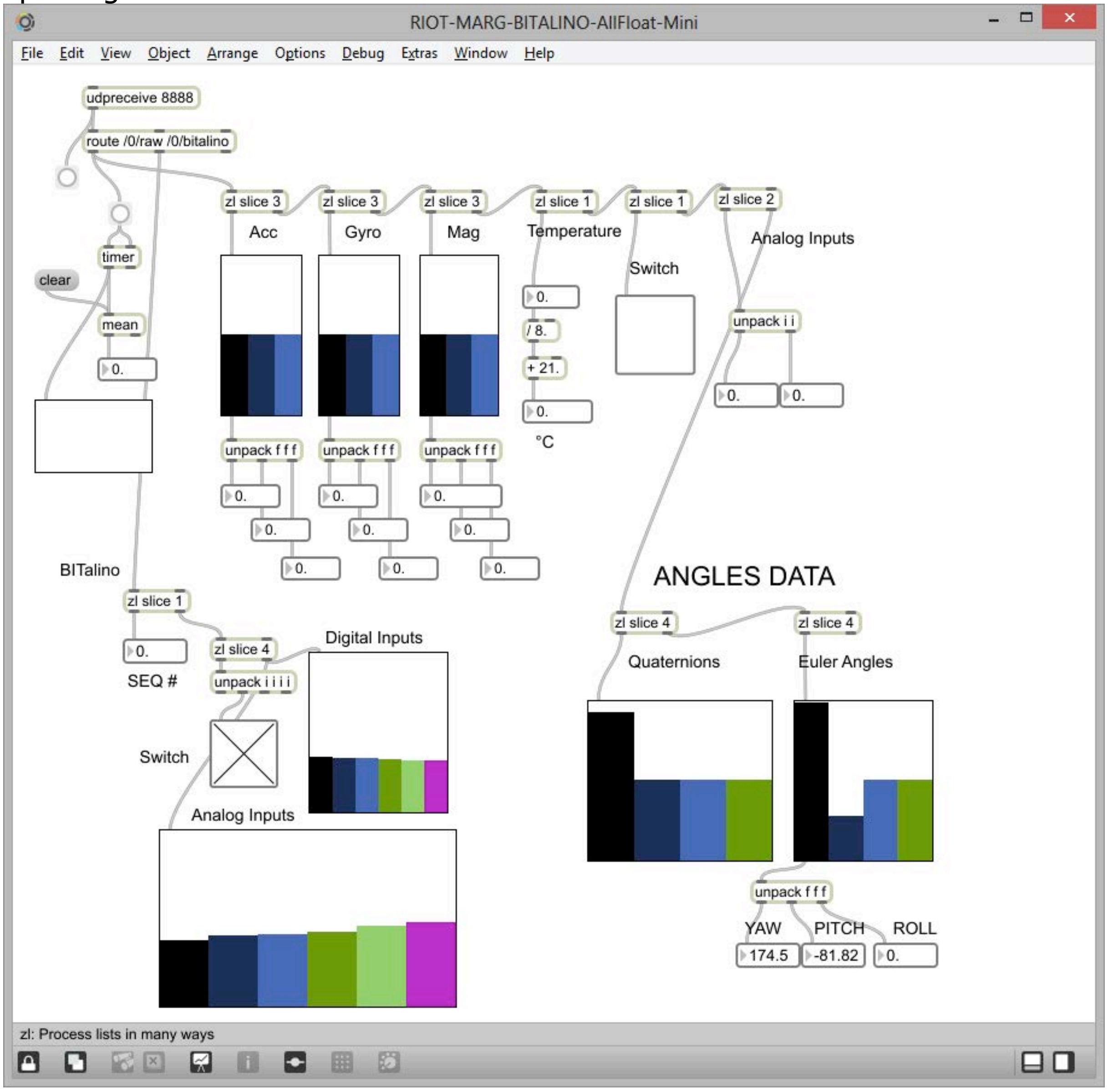
[1] See http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/ for more details

- Switch (GPIO28) {0 / 1}
- Analog Inputs (GPIO3 & GPIO4) {0 ; 4095}
- Quaternions {-1 ; 1}
- Euler Angles and Heading {-180 ; 180} °

The BITalino OSC message starts with **/<ID>/bitalino** followed by a list of 11 integer numbers that split like the following:

- Sequence number {0 ; 15}
- 4 digital inputs I1 to I4 {0 ; 1}
- 6 analog inputs A1 to A6 {0 ; 1023}

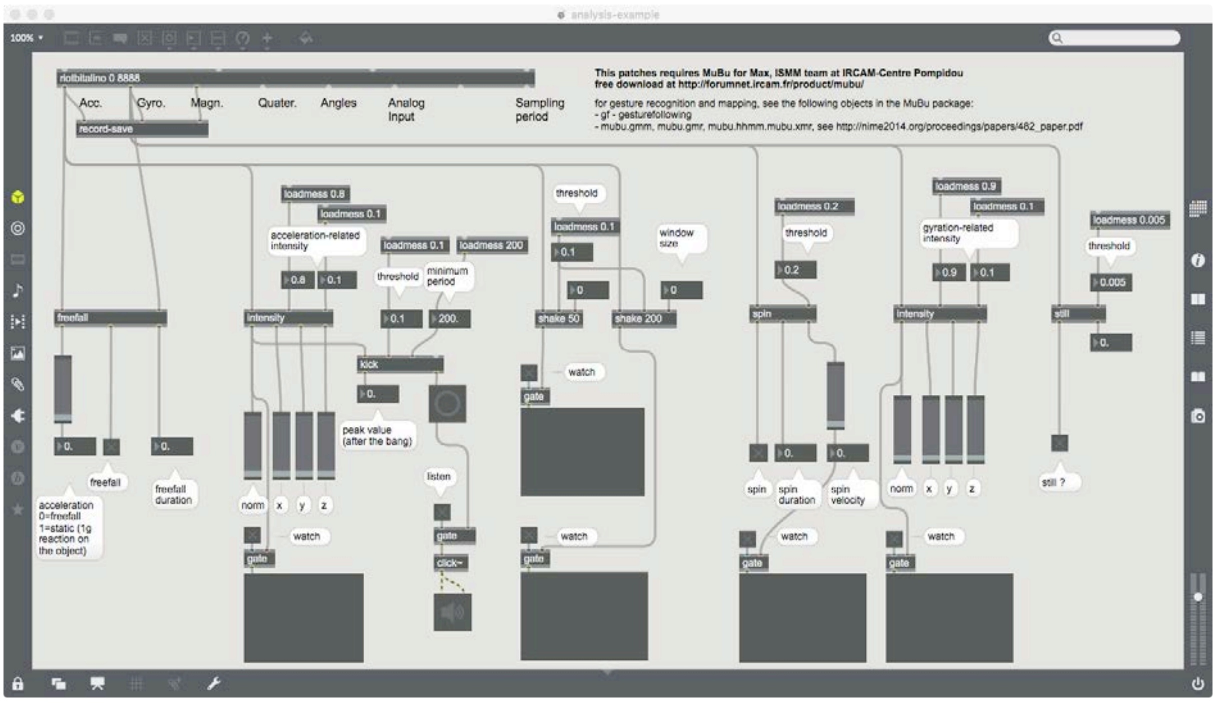
Below an example Max/MSP patch shows the easiness of receiving and splitting the data flow :



Max abstractions for the R-IoT Bitalino

Several abstractions for Max (Cycling'74) are provided in the github https://github.com/Ircam-R-IoT

For example the following abstraction analysis-example.maxpat shows example of various small abstraction that can be used to transform the raw accelerometer and gyroscope in intensity parameters and detecting kicks. These abstraction uses the free Max library MuBu available here: http://forumnet.ircam.fr/product/mubu/


This patches requires MuBu for Max, ISMM team at IRCAM-Centre Pompidou
free download at http://forumnet.ircam.fr/product/mubu/

for gesture recognition and mapping, see the following objects in the MuBu package:
- gf - gesturefollowing
- mubu.gmm, mubu.gmr, mubu.hhmm.mubu.xmr, see http://nime2014.org/proceedings/papers/462_paper.pdf

Programming the R-IoT

Install the Energia IDE

1. visit http://energia.nu/ and look for the download section
2. download energia version 17
3. On windows, simply unzip the Energia folder on your main hard drive. On Mac, place the folder-program in the Application folder.

Install the USB serial port driver

The R-IoT unit uses an external USB serial port to dialog with the computer, such as the TTL-232R-3V3 from FTDI (Farnell part # 1329311).

Install the drivers by visiting FTDI download section (look for VCP driver)

http://www.ftdichip.com/Drivers/VCP.htm

On windows, it will create a COM port. On Mac OS, the user sometimes must create the port by going in the network preferences (select standard NULL modem).

Customize the IDE

In order to compile the "full" firmware (currently named BITalino R-IoT 1.8), the default linker file used by the Energia tool chain must be modified as the reserved heap size / stack is too high when compiling big programs.

On windows:

- open the location of your Energia folder, such as C:\Program Files (x86)\energia-0101E0017\
- keep going into hardware\cc3200\cores\cc3200
- edit the file cc3200.ld

On Mac OS:

- open the location of your Energia folder, usually in the Application folder
- OSX applications are actually a folder. Right click on the app icon and select "show package contents"

- browse to Contents/Resources/Java/hardware/cc3200/cores/cc3200
- edit the file cc3200.ld

In the .ld file, simply edit the first line and change the heap size to 0x0008000 and save the file:

```
HEAP_SIZE = 0x0007500;
```

Install the Firmware and Examples

The full firmware uses 2 libraries that need to be installed to allow compiling the code : the SFLS library (File System for the CC3200 in order to retain parameters in non volatile memory) and the Bitalino basic library (a small wrapper to ease talking to the BITalino hardware).

Those libraries must be placed in the Documents\Energia\libraries folder (PC or Mac).

Use Energia IDE

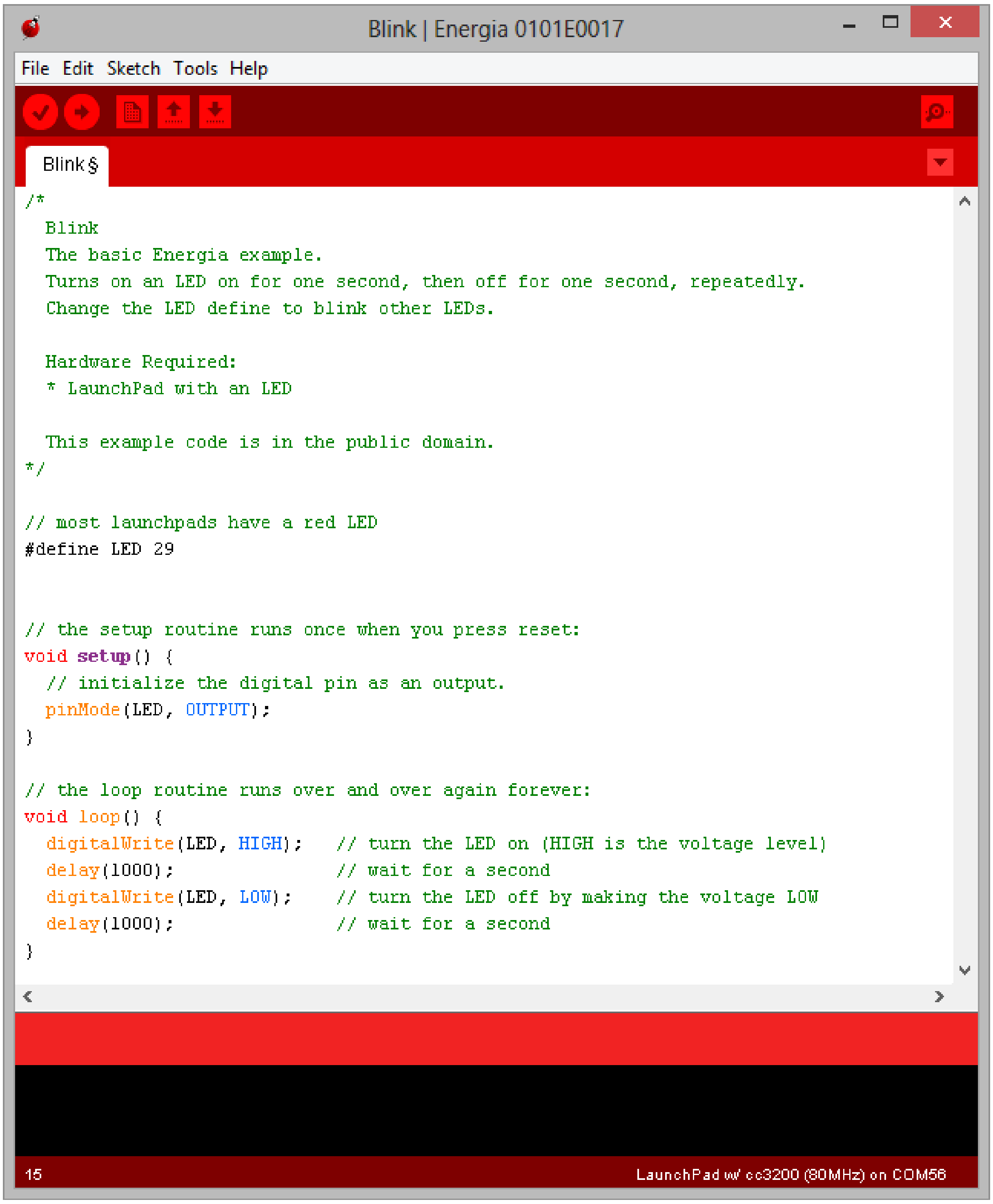
Launch Energia. A blank sketch appears. A sketch is composed of two essential functions, setup() and loop() which is equivalent to the main function in traditional C language. Energia, just like Arduino uses C and C++, along with a basic API and set of classes to access the hardware in what became the Arduino standard.

https://www.arduino.cc/en/Reference/HomePage

The setup function, called prior the main loop, is used to configure the module hardware, default behaviors and everything that requires some initialization before executing the main program.

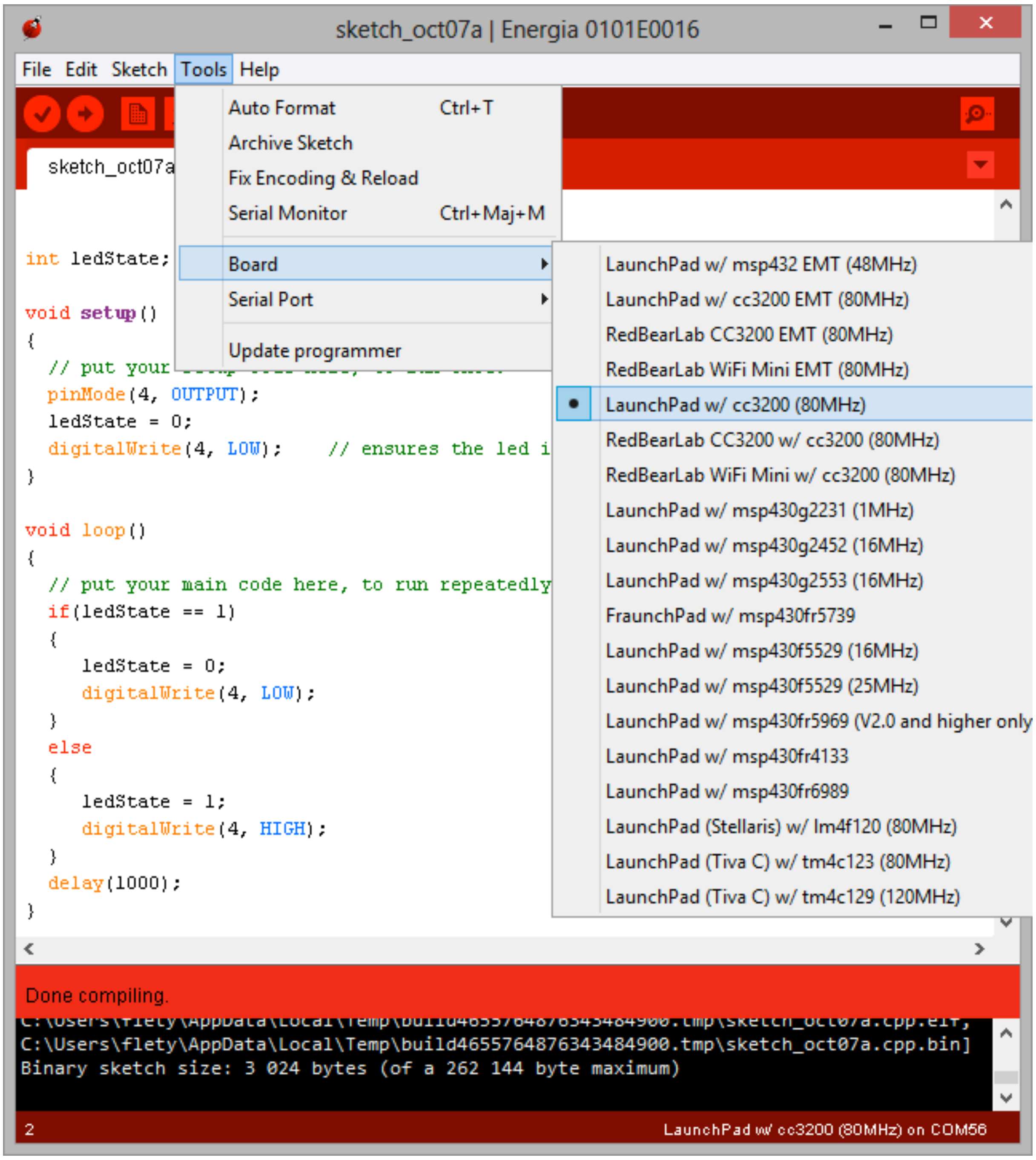
Once returning from the setup function, the loop function is called repetitively and is equivalent to any traditional endless program loop used on embedded platforms (a while(1) statement within the main function).

Below an example of blinking endlessly the red LED of the module (GPIO I/0 #9, launchpad pin 29):


Blink | Energia 0101E0017
File Edit Sketch Tools Help
Blink §
15
LaunchPad w/ cc3200 (80MHz) on COM56

```
/*
  Blink
  The basic Energia example.
  Turns on an LED on for one second, then off for one second, repeatedly.
  Change the LED define to blink other LEDs.

  Hardware Required:
  * LaunchPad with an LED

  This example code is in the public domain.
*/

// most launchpads have a red LED
#define LED 29


// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(LED, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Before compiling, select the proper target in the Tools -> Board menu and select the launchPad w/ CC3200 (80 MHz).


sketch_oct07a | Energia 0101E0016
File Edit Sketch Tools Help
sketch_oct07a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Maj+M
Board
Serial Port
Update programmer
LaunchPad w/ msp432 EMT (48MHz)
LaunchPad w/ cc3200 EMT (80MHz)
RedBearLab CC3200 EMT (80MHz)
RedBearLab WiFi Mini EMT (80MHz)
● LaunchPad w/ cc3200 (80MHz)
RedBearLab CC3200 w/ cc3200 (80MHz)
RedBearLab WiFi Mini w/ cc3200 (80MHz)
LaunchPad w/ msp430g2231 (1MHz)
LaunchPad w/ msp430g2452 (16MHz)
LaunchPad w/ msp430g2553 (16MHz)
FraunchPad w/ msp430fr5739
LaunchPad w/ msp430f5529 (16MHz)
LaunchPad w/ msp430f5529 (25MHz)
LaunchPad w/ msp430fr5969 (V2.0 and higher only
LaunchPad w/ msp430fr4133
LaunchPad w/ msp430fr6989
LaunchPad (Stellaris) w/ lm4f120 (80MHz)
LaunchPad (Tiva C) w/ tm4c123 (80MHz)
LaunchPad (Tiva C) w/ tm4c129 (120MHz)

```
int ledState;

void setup()
{
  // put your
  pinMode(4, OUTPUT);
  ledState = 0;
  digitalWrite(4, LOW);     // ensures the led i
}

void loop()
{
  // put your main code here, to run repeatedly
  if(ledState == 1)
  {
     ledState = 0;
     digitalWrite(4, LOW);
  }
  else
  {
     ledState = 1;
     digitalWrite(4, HIGH);
  }
  delay(1000);
}
```

Done compiling.

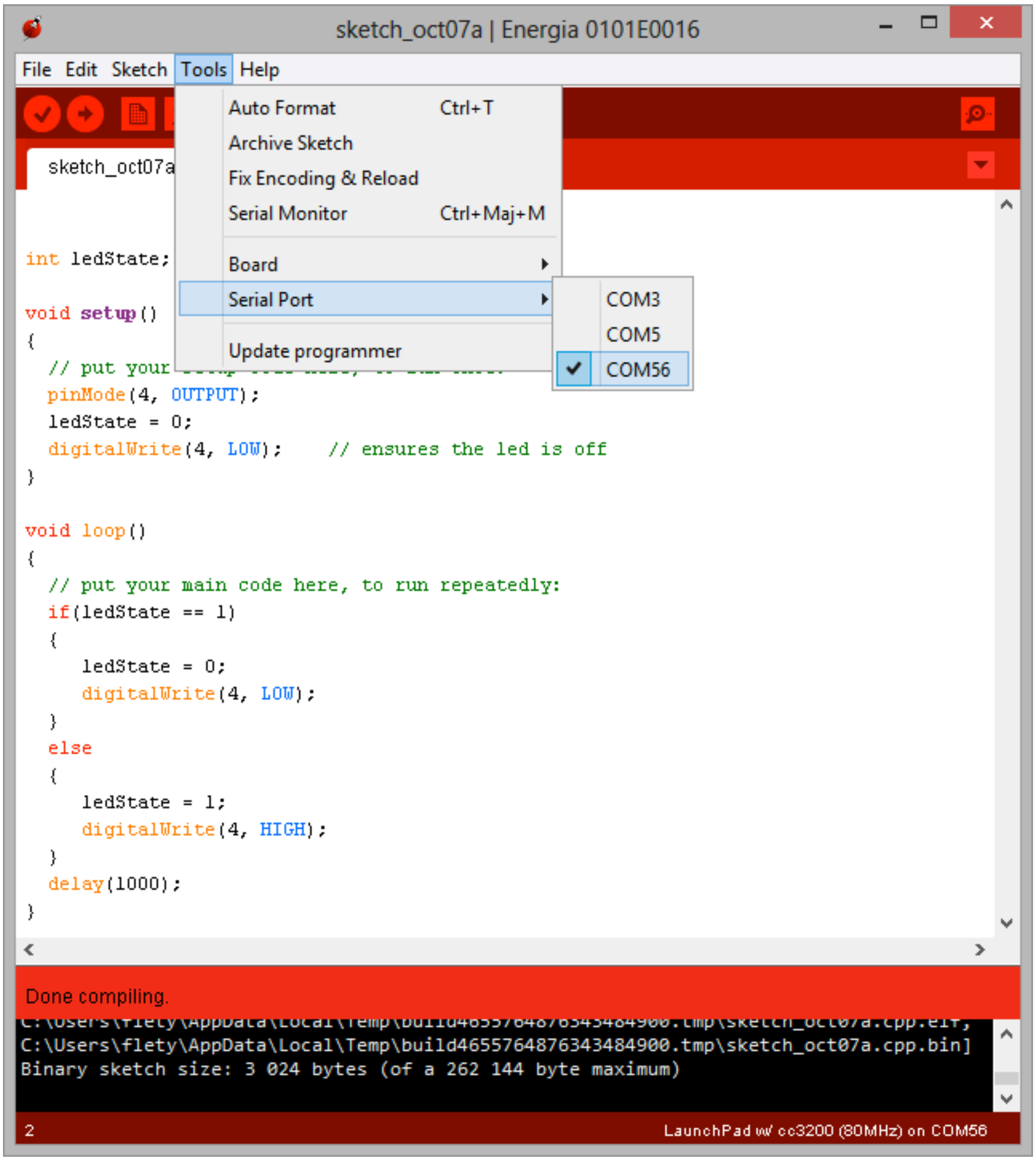
```
C:\Users\flety\AppData\Local\Temp\build4655764876343484900.tmp\sketch_oct07a.cpp.elf,
C:\Users\flety\AppData\Local\Temp\build4655764876343484900.tmp\sketch_oct07a.cpp.bin]
Binary sketch size: 3 024 bytes (of a 262 144 byte maximum)
```

2 LaunchPad w/ cc3200 (80MHz) on COM56

To compile to program, simply click on the left-most icon (tick).

Flash the firmware

To flash the code on the platform first plug the USB serial cable in a USB port then select the matching COM port in Energia Tool->Serial port menu.

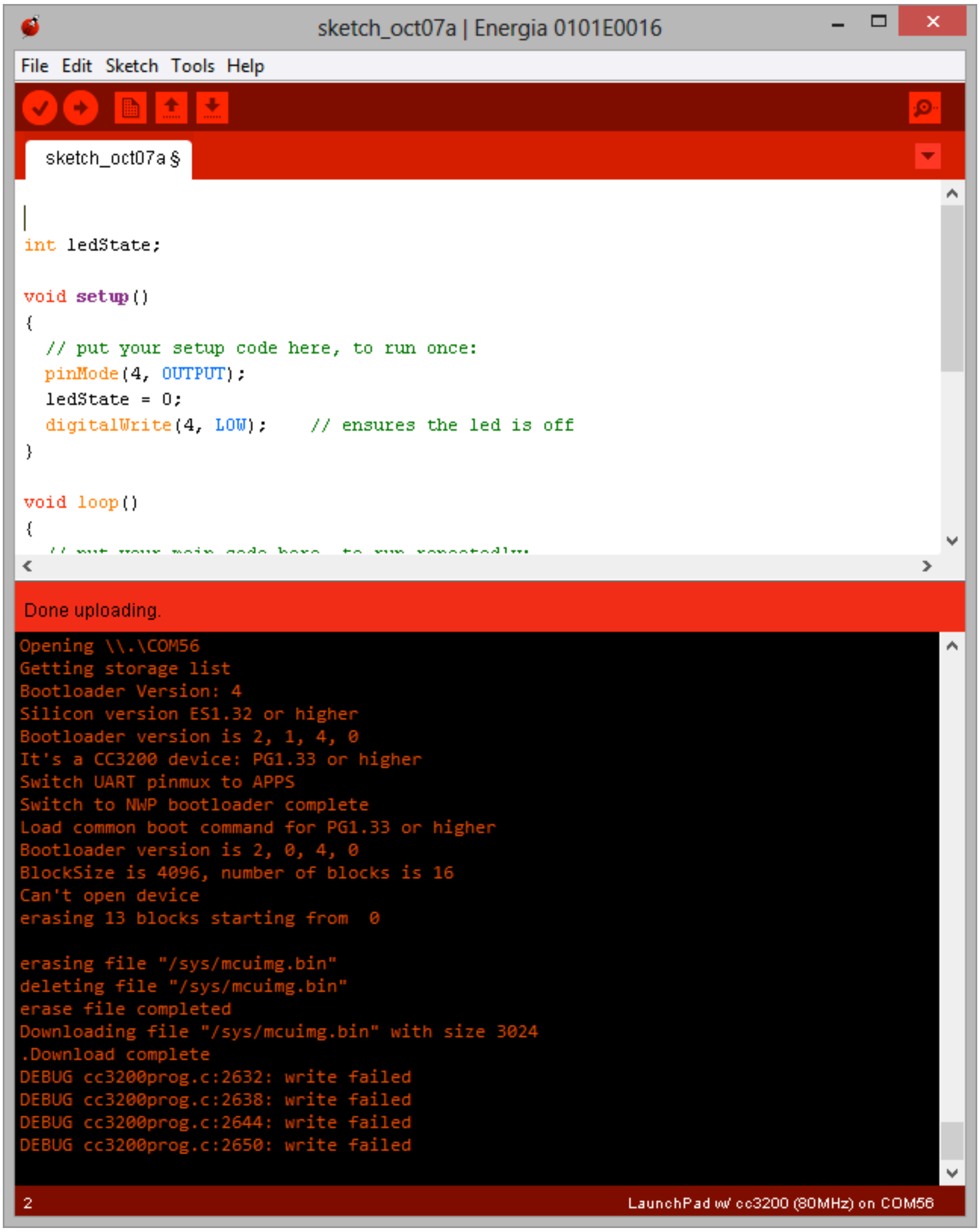


```
int ledState;

void setup()
{
  // put your
  pinMode(4, OUTPUT);
  ledState = 0;
  digitalWrite(4, LOW);     // ensures the led is off
}

void loop()
{
  // put your main code here, to run repeatedly:
  if(ledState == 1)
  {
     ledState = 0;
     digitalWrite(4, LOW);
  }
  else
  {
     ledState = 1;
     digitalWrite(4, HIGH);
  }
  delay(1000);
}
```

```
C:\Users\flety\AppData\Local\Temp\build4655764876343484900.tmp\sketch_oct07a.cpp.elf,
C:\Users\flety\AppData\Local\Temp\build4655764876343484900.tmp\sketch_oct07a.cpp.bin]
Binary sketch size: 3 024 bytes (of a 262 144 byte maximum)
```

With the module powered on, even already executing the code present in the chip, press both the flash and reset switches and click on the upload icon (arrow). It will first compile the code. Once compiling is over, the "uploading" text will be printed above the console. Release the reset switch while keeping the flash switch pressed. After a while, you'll get a done uploading message with the following log in the console of the IDE.

The DEBUG write fail messages are normal, as the bootloading program tries also to set the chip in debug mode while our platform doesn't have any JTAG debugger / port exported.


sketch_oct07a | Energia 0101E0016
File Edit Sketch Tools Help
sketch_oct07a §

```
int ledState;

void setup()
{
  // put your setup code here, to run once:
  pinMode(4, OUTPUT);
  ledState = 0;
  digitalWrite(4, LOW);     // ensures the led is off
}

void loop()
{
```

Done uploading.

```
Opening \\.\COM56
Getting storage list
Bootloader Version: 4
Silicon version ES1.32 or higher
Bootloader version is 2, 1, 4, 0
It's a CC3200 device: PG1.33 or higher
Switch UART pinmux to APPS
Switch to NWP bootloader complete
Load common boot command for PG1.33 or higher
Bootloader version is 2, 0, 4, 0
BlockSize is 4096, number of blocks is 16
Can't open device
erasing 13 blocks starting from  0

erasing file "/sys/mcuimg.bin"
deleting file "/sys/mcuimg.bin"
erase file completed
Downloading file "/sys/mcuimg.bin" with size 3024
.Download complete
DEBUG cc3200prog.c:2632: write failed
DEBUG cc3200prog.c:2638: write failed
DEBUG cc3200prog.c:2644: write failed
DEBUG cc3200prog.c:2650: write failed
```

Once there, the module can be reset (cycle the on-off switch, or press the reset onboard momentary switch) to leave flashing mode and execute the freshly uploaded code. With the above code, the blue LED should flash once per second.

Acknowledgements

This R-IoT has been developed initially by the ISMM and PiP teams at IRCAM-Centre Pompidou (STMS Lab IRCAM-CNRS-UPMC) within the CoSiMa project (funded by ANR) and the H2020 MusicBricks and Rapid-Mix projects funded by the European Union’s Horizon 2020 research and innovation programme.

The Bitalino R-IoT has been developed as a collaboration between IRCAM and PluX, with the support of from the European Union’s Horizon 2020 research and innovation programme under grant agreement N°644862 RAPID-MIX.