Sistemi Embedded con Linux e Python

Michele Scafoglieri
Studio Sigma

Linux Day - Roma

26 Ottobre 2013

STUDIO SIGMA

studio tecnico di ingegneria dell'informazione

Ing. Michele Scafoglieri

Email: mscafoglieri@studiosigma.pro
Pec: mscafoglieri@pec.ording.roma.it
Skype: mscafoglieri

Via G. Mattè Trucco 54
00132 Roma (RM) - Italia

P.IVA: IT01231830777
C.F.: SCFMHL70P27L049N

Tel.: (+39) 320 26.75.777
Fax: (+39) 06 92.594.676

www.studiosigma.pro

Agenda

- Di quali sistemi parliamo?
- L'approccio classico
- Perchè Linux?
- Perchè Python?
- Come comunicano?
- Casi di studio:
  - Display per sale di attesa
  - timbracartellino

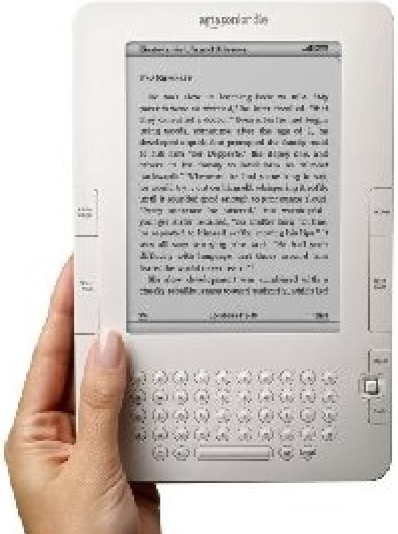
Nikon
D7000

LINKSYS

PHILIPS

MENU

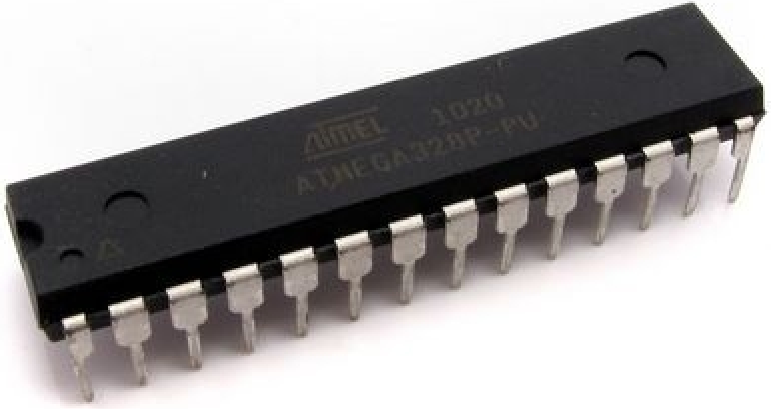
LEXMARK


amazonkindle

SAMSUNG

RIGOL

elna


ATMEL
1020
ATMEGA328P-PU

ATMEL

RENESAS

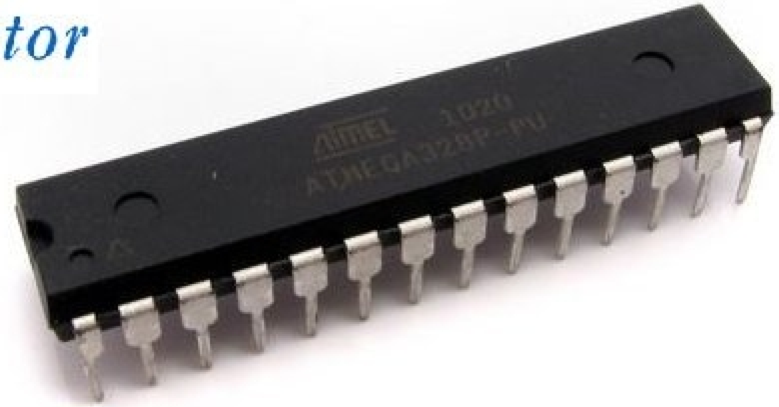
freescale

NXP

maxim integrated

National Semiconductor

ST



TOSHIBA

infineon

intel

CYPRESS
PERFORM

ARM

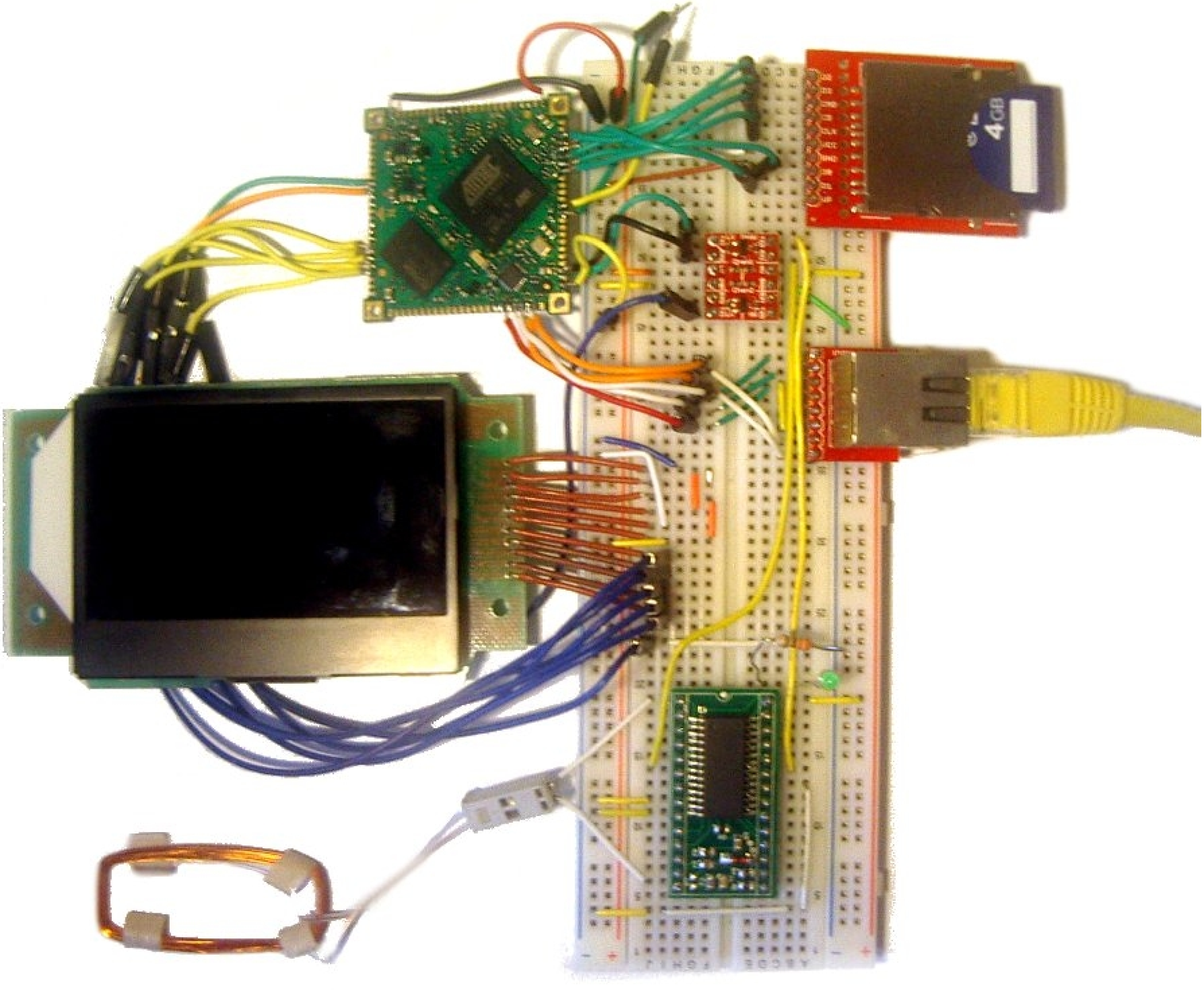
PARALLAX INC
www.parallax.com

FUJITSU

MICROCHIP

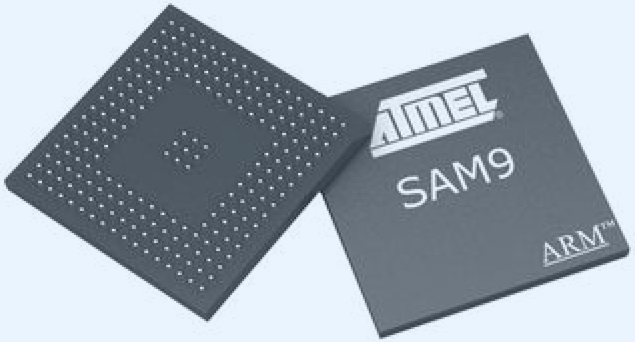
HOLTEK

zilog

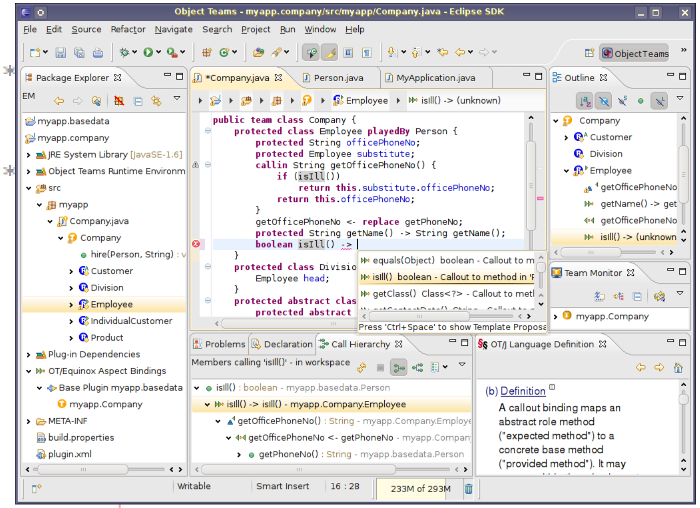


4GB

Tools



```
/****************************************************************************
  Section:
        Header Parsing Configuration
  ***************************************************************************

        // Header strings for which we'd like to parse
        static ROM char * ROM HTTPRequestHeaders[] =
        {
                "Cookie:",
                "Authorization:",
                "Content-Length:"
        };

        // Set to length of longest string above
        #define HTTP_MAX_HEADER_LEN             (15u)

/****************************************************************************
  Section:
        HTTP Connection State Global Variables
  ***************************************************************************/
        #if defined(__18CXX) && !defined(HI_TECH_C)
                #pragma udata HTTP_CONNECTION_STATES
        #endif
        #if defined(HTTP_SAVE_CONTEXT_IN_PIC_RAM)
                HTTP_CONN                               HTTPControlBlock
                #define HTTPLoadConn(a)         do{curHTTPID = (a);}while(0)
        #else
                HTTP_CONN curHTTP;
                static void HTTPLoadConn(BYTE hHTTP);
        #endif
```



Object Teams - myapp.company/src/myapp/Company.java - Eclipse SDK

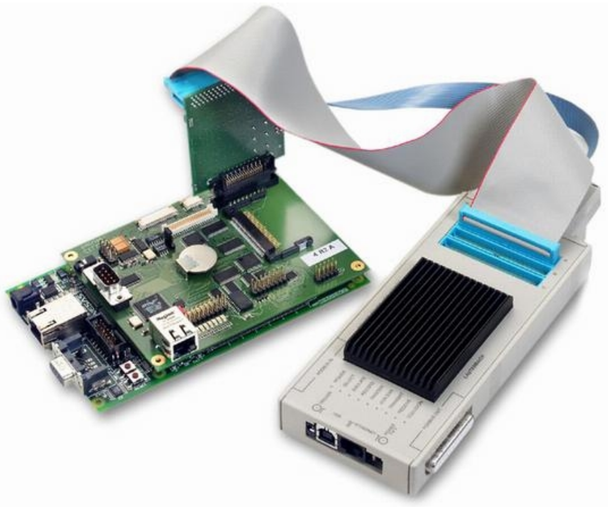
```
public team class Company {
    protected class Employee playedBy Person {
        protected String officePhoneNo;
        protected Employee substitute;
        callin String getOfficePhoneNo() {
            if (isIll())
                return this.substitute.officePhoneNo;
            return this.officePhoneNo;
        }
        getOfficePhoneNo <- replace getPhoneNo;
        protected String getName() -> String getName();
        boolean isIll() -> 
    }
    protected class Divisio
        Employee head;
    }
    protected abstract clas
        protected abstract
```

Members calling 'isIll()' - in workspace

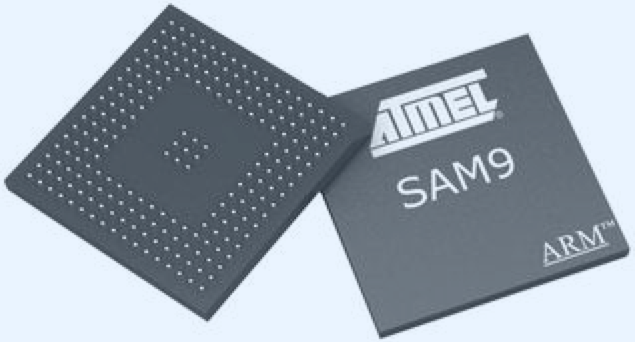
(b) Definition

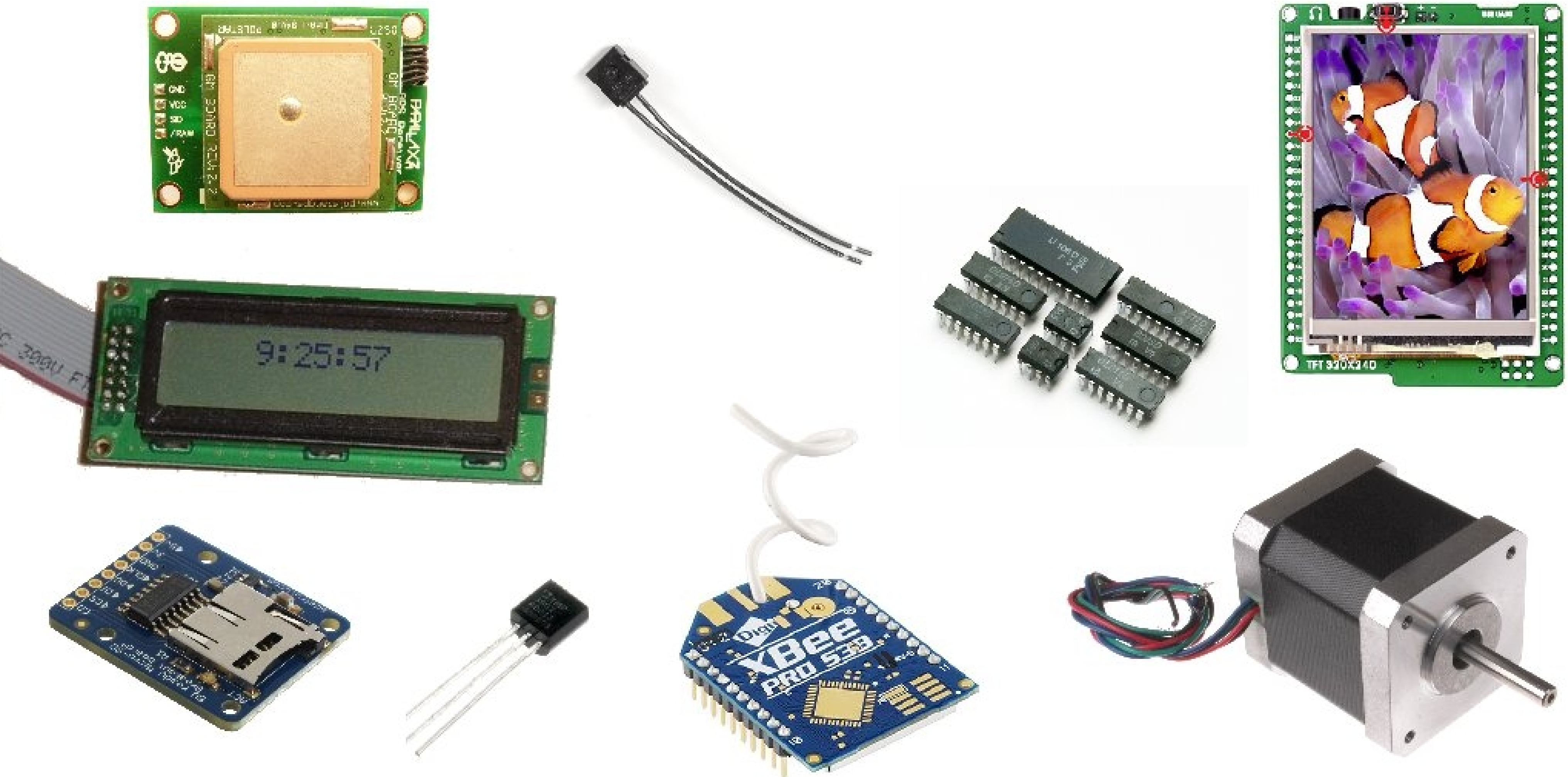
A callout binding maps an abstract role method ("expected method") to a concrete base method ("provided method"). It may

File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
ObjectTeams
Package Explorer
*Company.java
Person.java
MyApplication.java
Outline
Employee
isIll() -> (unknown)
myapp.basedata
myapp.company
JRE System Library [JavaSE-1.6]
Object Teams Runtime Environm
src
myapp
Company.java
Company
hire(Person, String)
Customer
Division
Employee
IndividualCustomer
Product
Plug-in Dependencies
OT/Equinox Aspect Bindings
Base Plugin myapp.basedata
myapp.Company
META-INF
build.properties
plugin.xml
equals(Object) boolean - Callout to m
isIll() boolean - Callout to method in 'P
getClass() Class<?> - Callout to met
Press 'Ctrl+Space' to show Template Proposa
Team Monitor
myapp.Company
Problems
Declaration
Call Hierarchy
OT/J Language Definition
isIll() : boolean - myapp.basedata.Person
isIll() -> isIll() - myapp.Company.Employee
getOfficePhoneNo() : String - myapp.Company.Employe
getOfficePhoneNo <- getPhoneNo - myapp.Compan
getPhoneNo() : String - myapp.basedata.Person
Writable
Smart Insert
16 : 28
233M of 293M

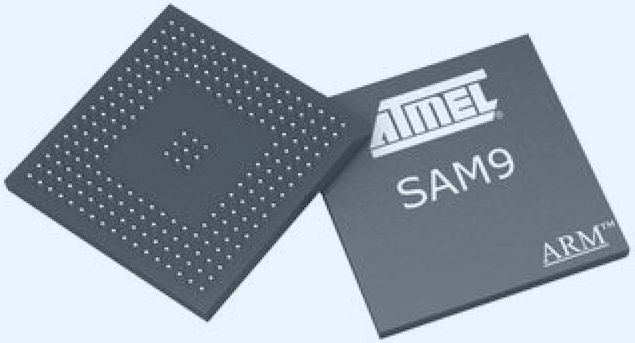


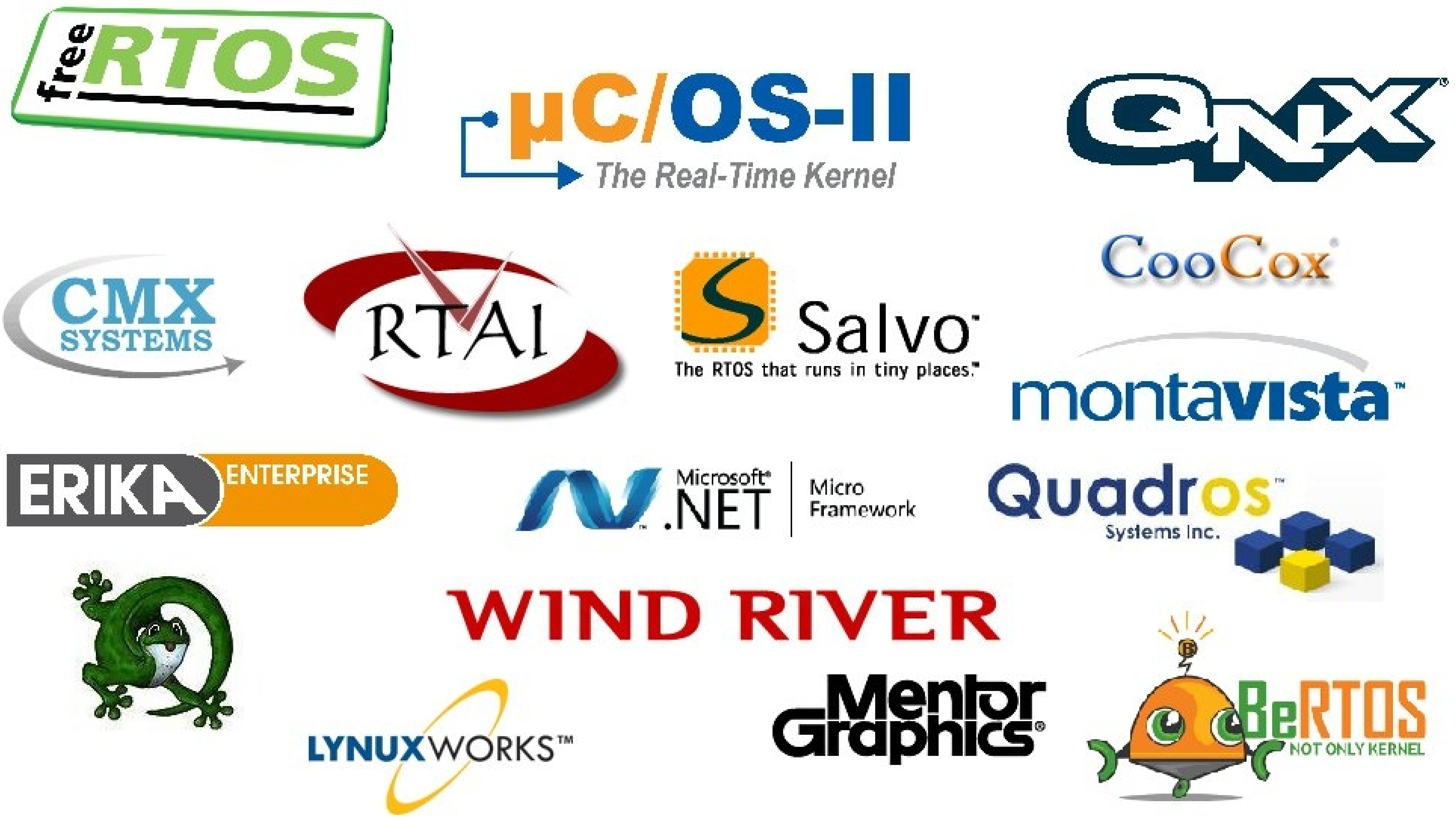
Drivers

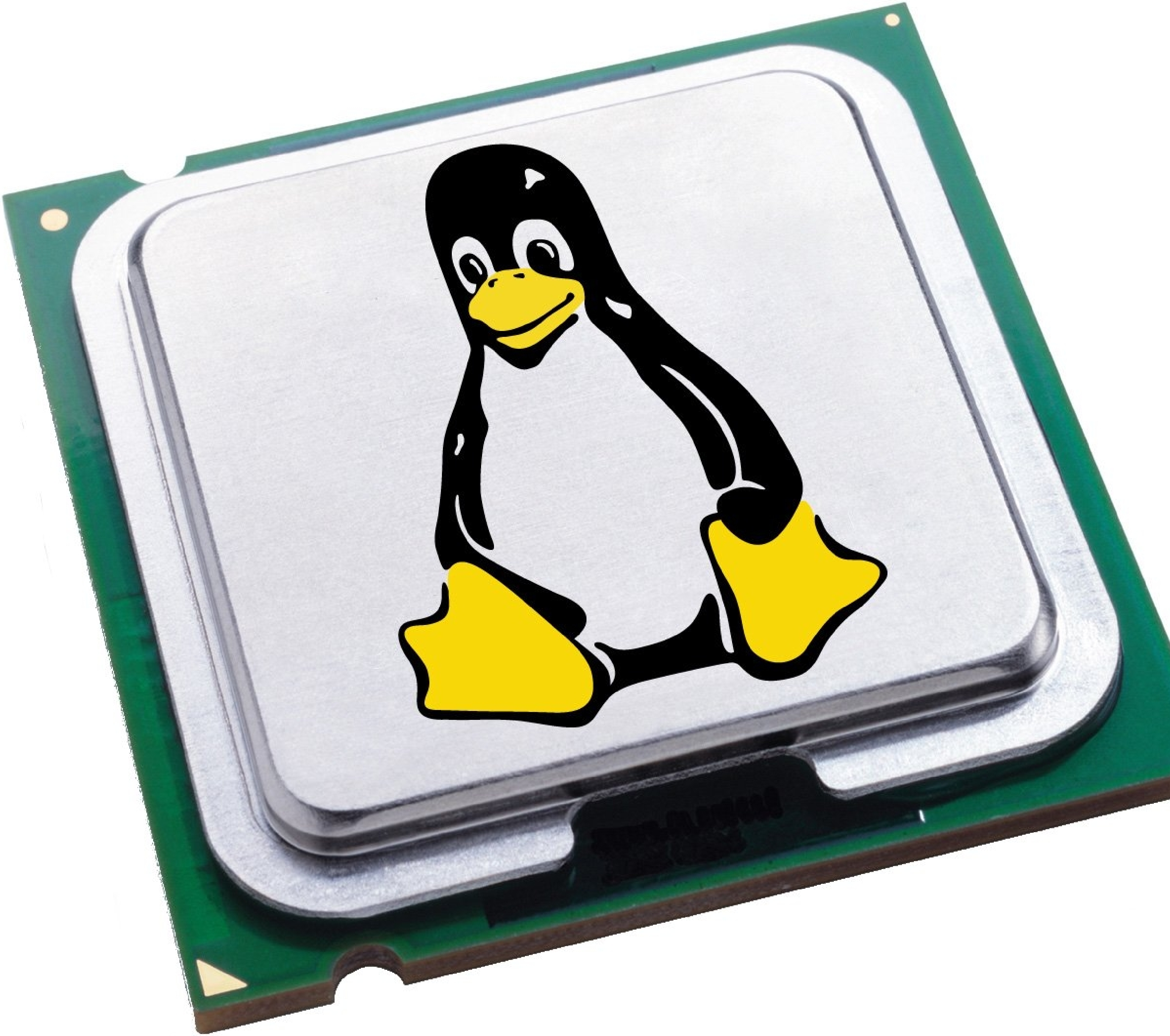




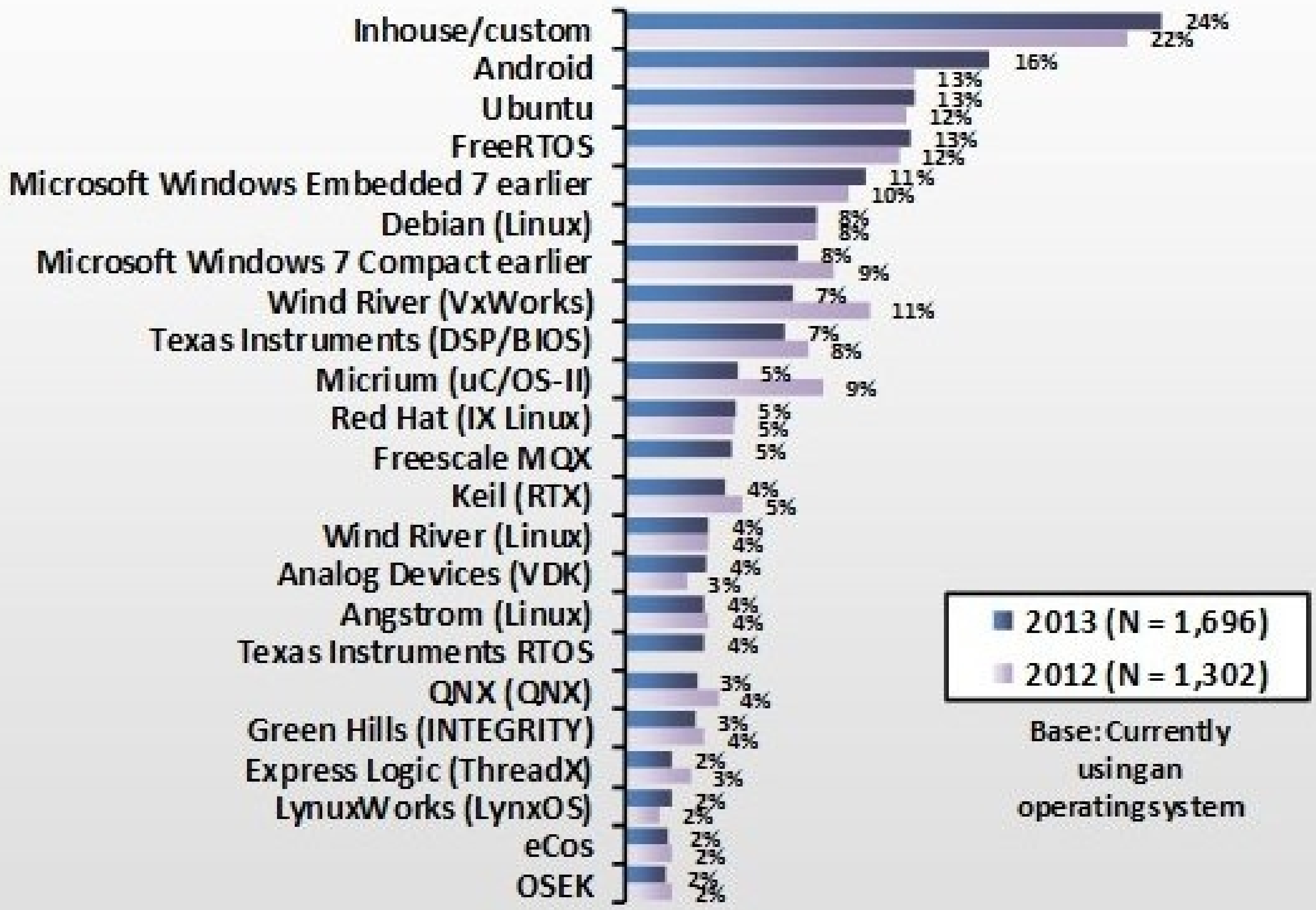
RTOS


ATMEL
SAM9
ARM™


freeRTOS
µC/OS-II
The Real-Time Kernel
QNX®
CMX SYSTEMS
RTAI
Salvo™
The RTOS that runs in tiny places.™
CooCox®
montavista™
ERIKA ENTERPRISE
Microsoft® .NET
Micro Framework
Quadros™
Systems Inc.
WIND RIVER
LYNUXWORKS™
Mentor Graphics®
BeRTOS
NOT ONLY KERNEL

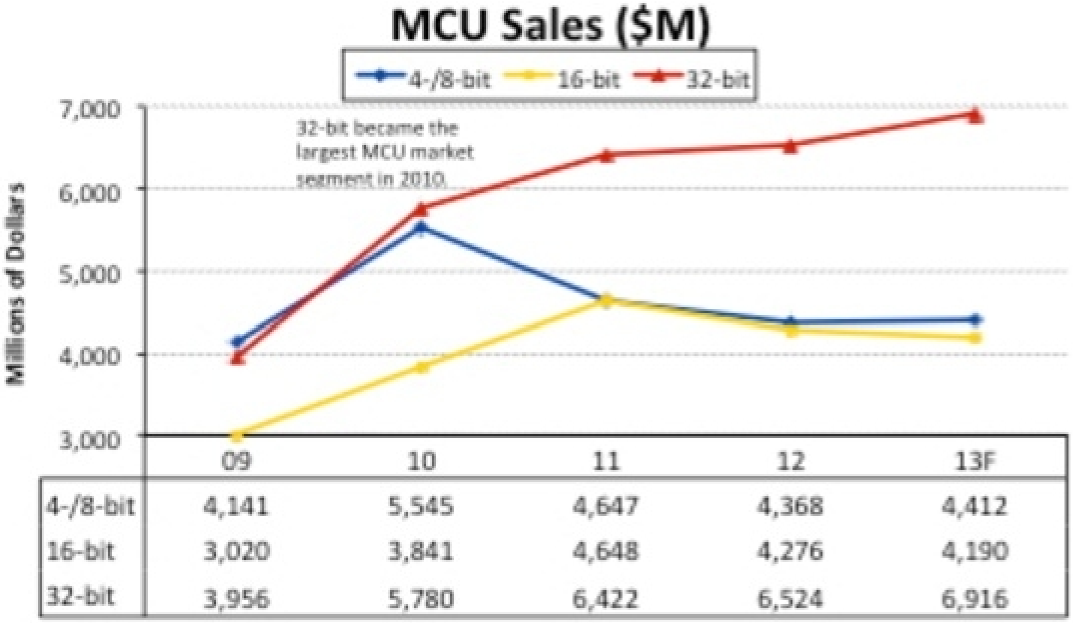


Please select ALL of the operating systems you are currently using.



Base: Currently using an operating system

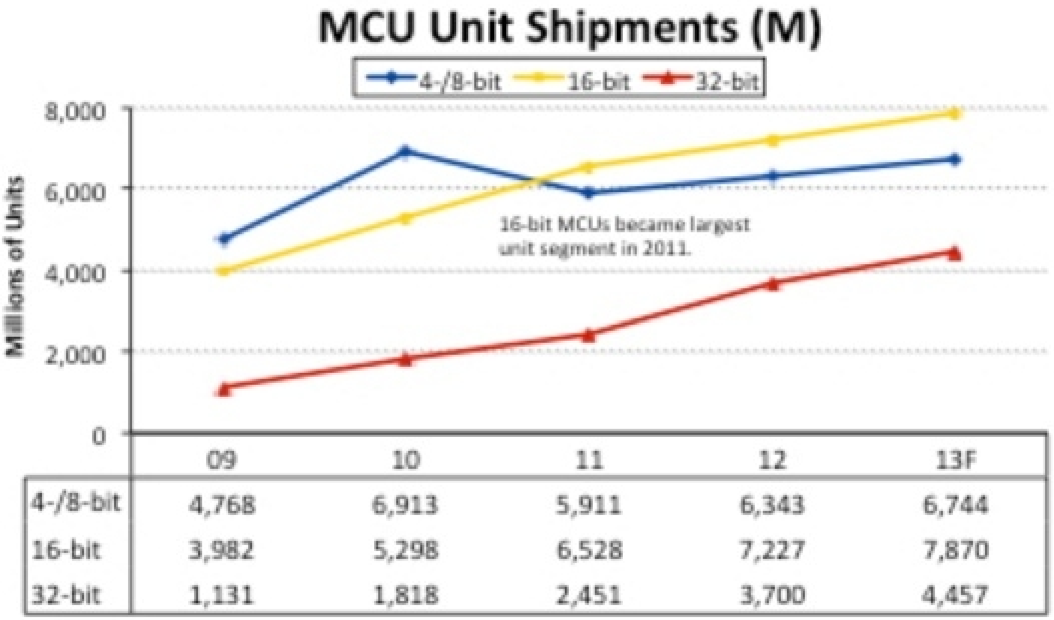
Only Operating Systems that had 2% or more are shown.



MCU Sales ($M)

32-bit became the largest MCU market segment in 2010.

	09	10	11	12	13F
4-/8-bit	4,141	5,545	4,647	4,368	4,412
16-bit	3,020	3,841	4,648	4,276	4,190
32-bit	3,956	5,780	6,422	6,524	6,916

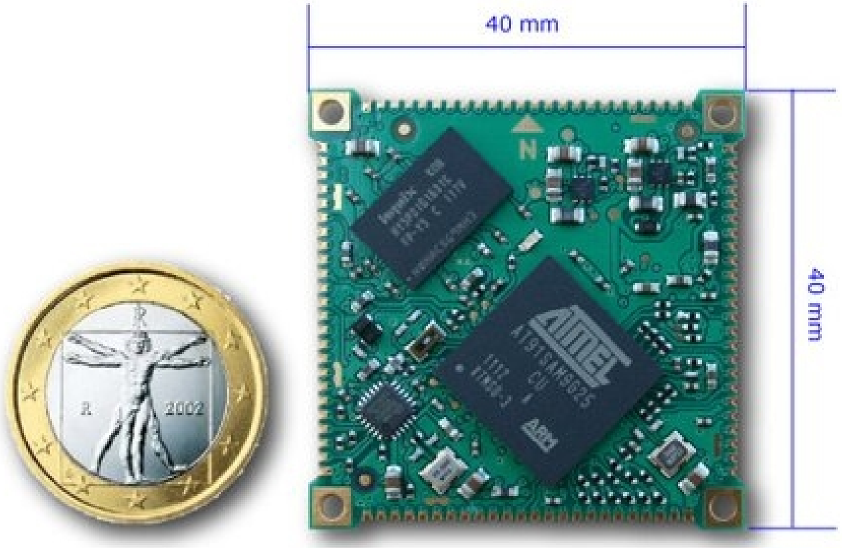


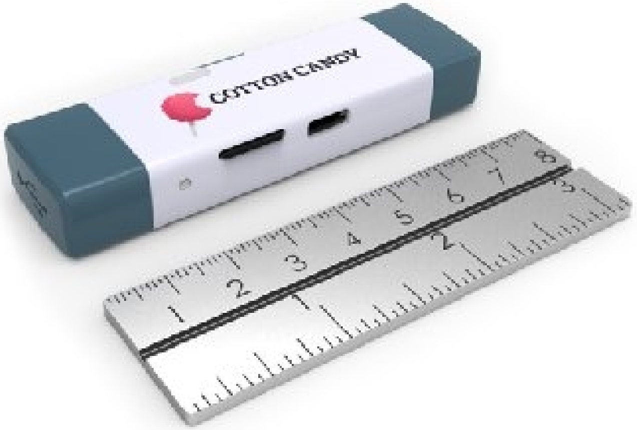
MCU Unit Shipments (M)

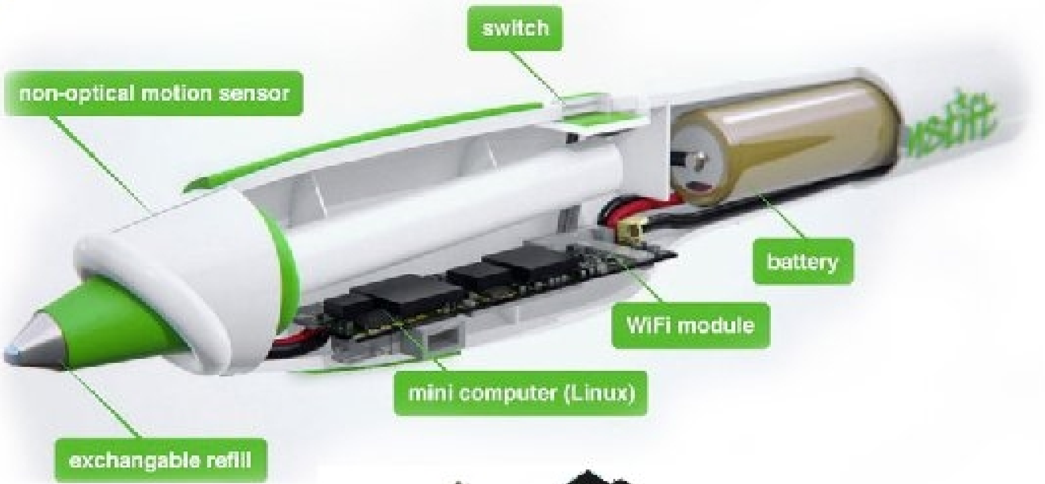
16-bit MCUs became largest unit segment in 2011.

	09	10	11	12	13F
4-/8-bit	4,768	6,913	5,911	6,343	6,744
16-bit	3,982	5,298	6,528	7,227	7,870
32-bit	1,131	1,818	2,451	3,700	4,457

Source: IC Insights

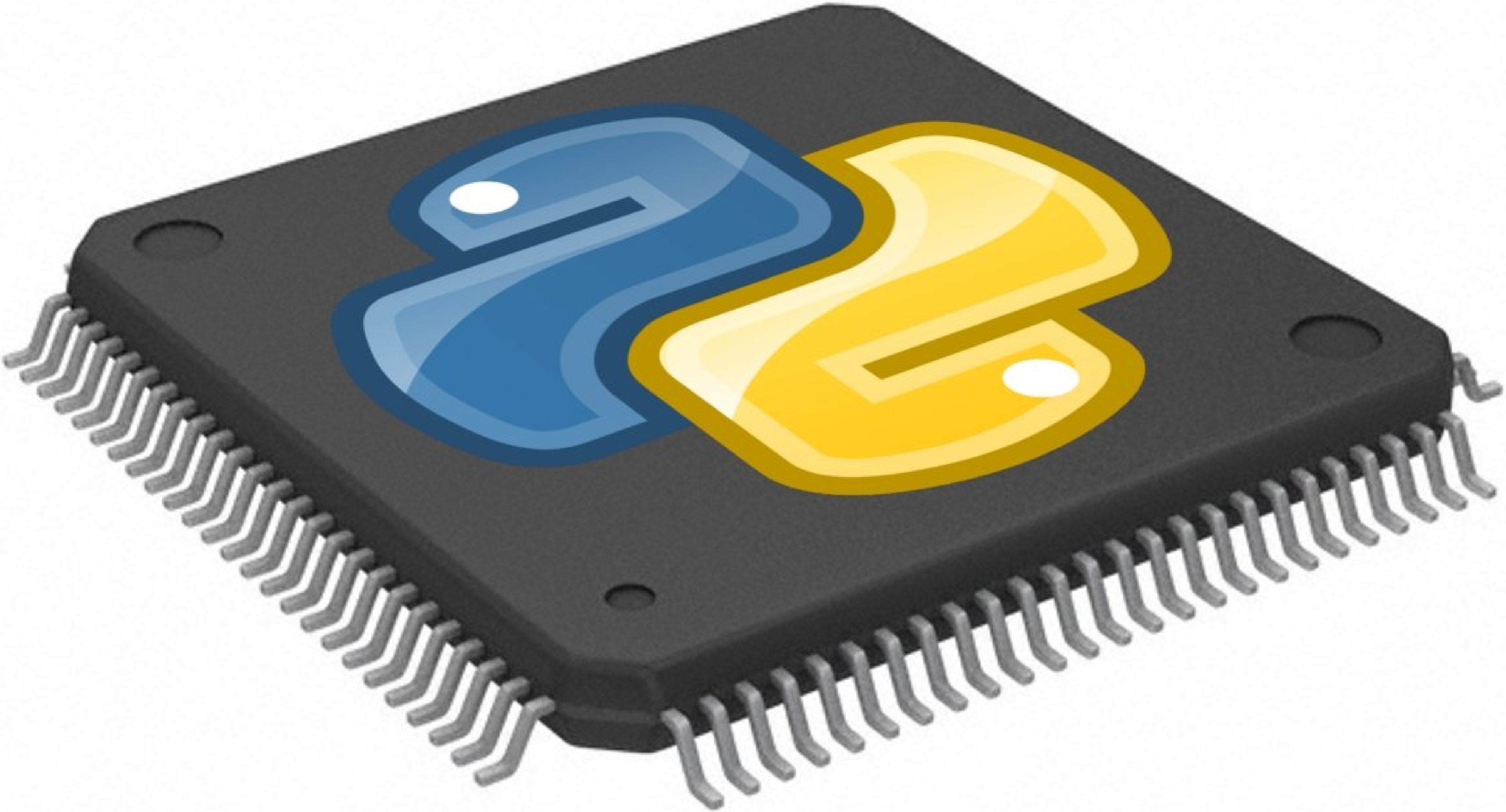




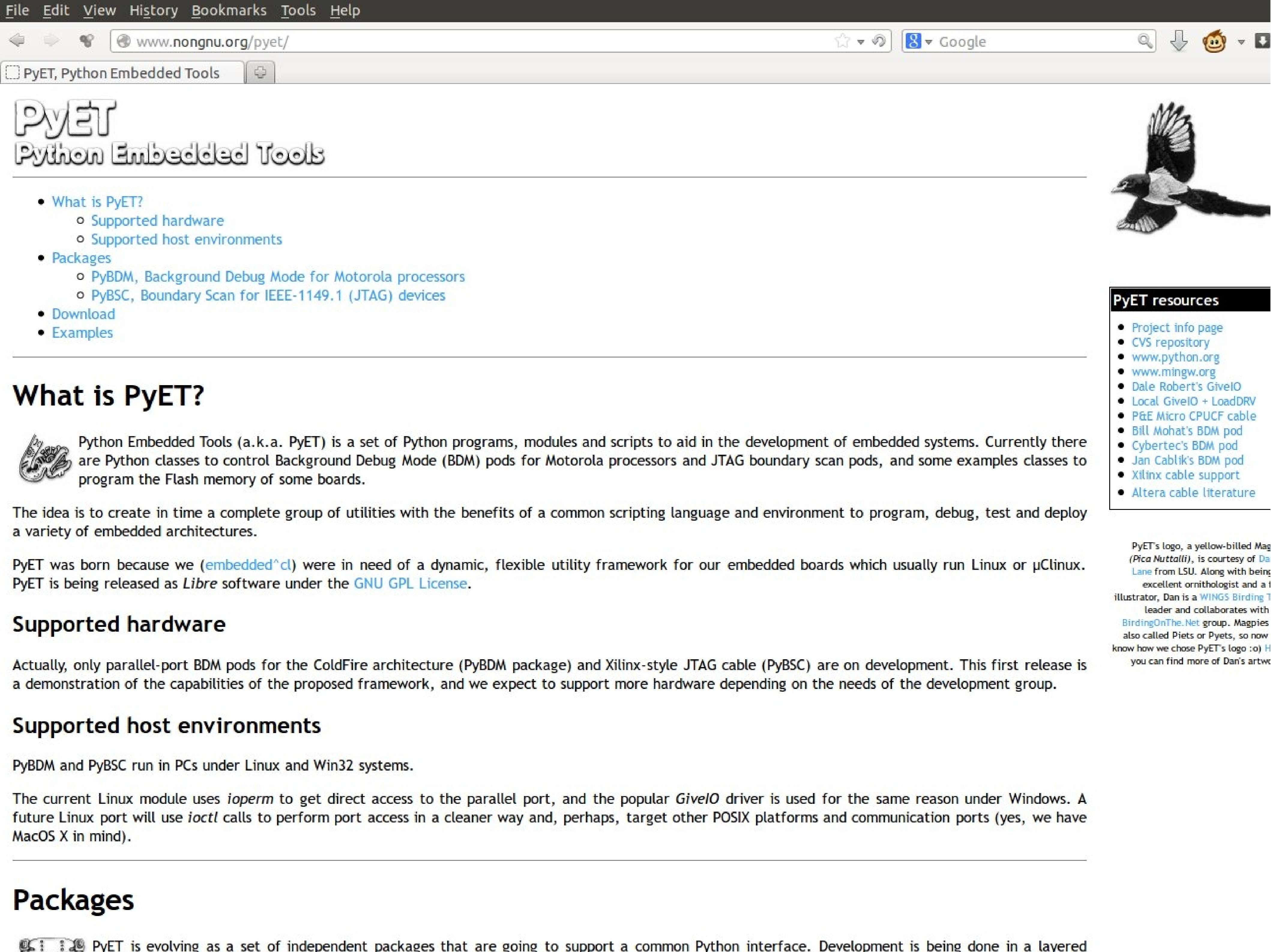


https://raymii.org/s/articles/Small_Linux_PCs.html
http://iqjar.com/jar/an-overview-and-comparison-of-todays-single-board-micro-computers/

Seagate
N S
Seagate
07120



OpenERP
Open Source Management Software
ZOPE
Autodesk
Maya
NASA
SciPy.org
Dropbox
Plone®
YAHOO!®
GROUPS
Arc
ESRI
GIS™
Google™
mercurial
YouTube


File Edit View History Bookmarks Tools Help
www.nongnu.org/pyet/
Google
PyET, Python Embedded Tools

PyET
Python Embedded Tools

- What is PyET?
  - Supported hardware
  - Supported host environments
- Packages
  - PyBDM, Background Debug Mode for Motorola processors
  - PyBSC, Boundary Scan for IEEE-1149.1 (JTAG) devices
- Download
- Examples

What is PyET?

Python Embedded Tools (a.k.a. PyET) is a set of Python programs, modules and scripts to aid in the development of embedded systems. Currently there are Python classes to control Background Debug Mode (BDM) pods for Motorola processors and JTAG boundary scan pods, and some examples classes to program the Flash memory of some boards.

The idea is to create in time a complete group of utilities with the benefits of a common scripting language and environment to program, debug, test and deploy a variety of embedded architectures.

PyET was born because we (embedded^cl) were in need of a dynamic, flexible utility framework for our embedded boards which usually run Linux or µClinux. PyET is being released as *Libre* software under the GNU GPL License.

Supported hardware

Actually, only parallel-port BDM pods for the ColdFire architecture (PyBDM package) and Xilinx-style JTAG cable (PyBSC) are on development. This first release is a demonstration of the capabilities of the proposed framework, and we expect to support more hardware depending on the needs of the development group.

Supported host environments

PyBDM and PyBSC run in PCs under Linux and Win32 systems.

The current Linux module uses *ioperm* to get direct access to the parallel port, and the popular *GiveIO* driver is used for the same reason under Windows. A future Linux port will use *ioctl* calls to perform port access in a cleaner way and, perhaps, target other POSIX platforms and communication ports (yes, we have MacOS X in mind).

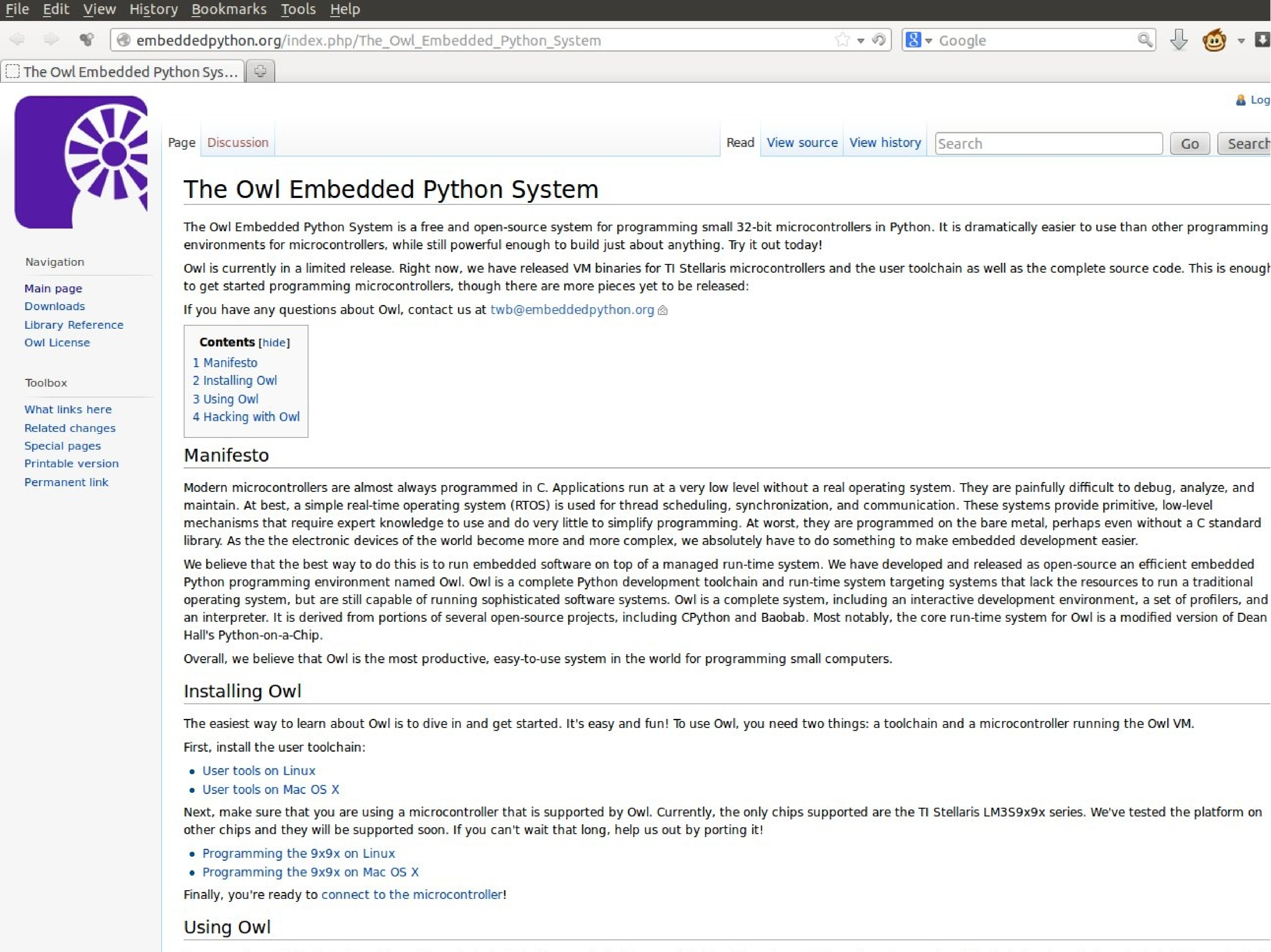
Packages

PyET is evolving as a set of independent packages that are going to support a common Python interface. Development is being done in a layered

PyET resources

- Project info page
- CVS repository
- www.python.org
- www.mingw.org
- Dale Robert's GiveIO
- Local GiveIO + LoadDRV
- P&E Micro CPUCF cable
- Bill Mohat's BDM pod
- Cybertec's BDM pod
- Jan Cablik's BDM pod
- Xilinx cable support
- Altera cable literature

PyET's logo, a yellow-billed *Mag*
(Pica Nuttalli), is courtesy of Da
Lane from LSU. Along with being
excellent ornithologist and a f
illustrator, Dan is a WINGS Birding T
leader and collaborates with
BirdingOnThe.Net group. *Magpies*
also called Piets or Pyets, so now
know how we chose PyET's logo :o) H
you can find more of Dan's artwo



The Owl Embedded Python System

The Owl Embedded Python System is a free and open-source system for programming small 32-bit microcontrollers in Python. It is dramatically easier to use than other programming environments for microcontrollers, while still powerful enough to build just about anything. Try it out today!

Owl is currently in a limited release. Right now, we have released VM binaries for TI Stellaris microcontrollers and the user toolchain as well as the complete source code. This is enougl to get started programming microcontrollers, though there are more pieces yet to be released:

If you have any questions about Owl, contact us at twb@embeddedpython.org

Contents [hide]

1 Manifesto
2 Installing Owl
3 Using Owl
4 Hacking with Owl

Manifesto

Modern microcontrollers are almost always programmed in C. Applications run at a very low level without a real operating system. They are painfully difficult to debug, analyze, and maintain. At best, a simple real-time operating system (RTOS) is used for thread scheduling, synchronization, and communication. These systems provide primitive, low-level mechanisms that require expert knowledge to use and do very little to simplify programming. At worst, they are programmed on the bare metal, perhaps even without a C standard library. As the the electronic devices of the world become more and more complex, we absolutely have to do something to make embedded development easier.

We believe that the best way to do this is to run embedded software on top of a managed run-time system. We have developed and released as open-source an efficient embedded Python programming environment named Owl. Owl is a complete Python development toolchain and run-time system targeting systems that lack the resources to run a traditional operating system, but are still capable of running sophisticated software systems. Owl is a complete system, including an interactive development environment, a set of profilers, and an interpreter. It is derived from portions of several open-source projects, including CPython and Baobab. Most notably, the core run-time system for Owl is a modified version of Dean Hall's Python-on-a-Chip.

Overall, we believe that Owl is the most productive, easy-to-use system in the world for programming small computers.

Installing Owl

The easiest way to learn about Owl is to dive in and get started. It's easy and fun! To use Owl, you need two things: a toolchain and a microcontroller running the Owl VM.

First, install the user toolchain:

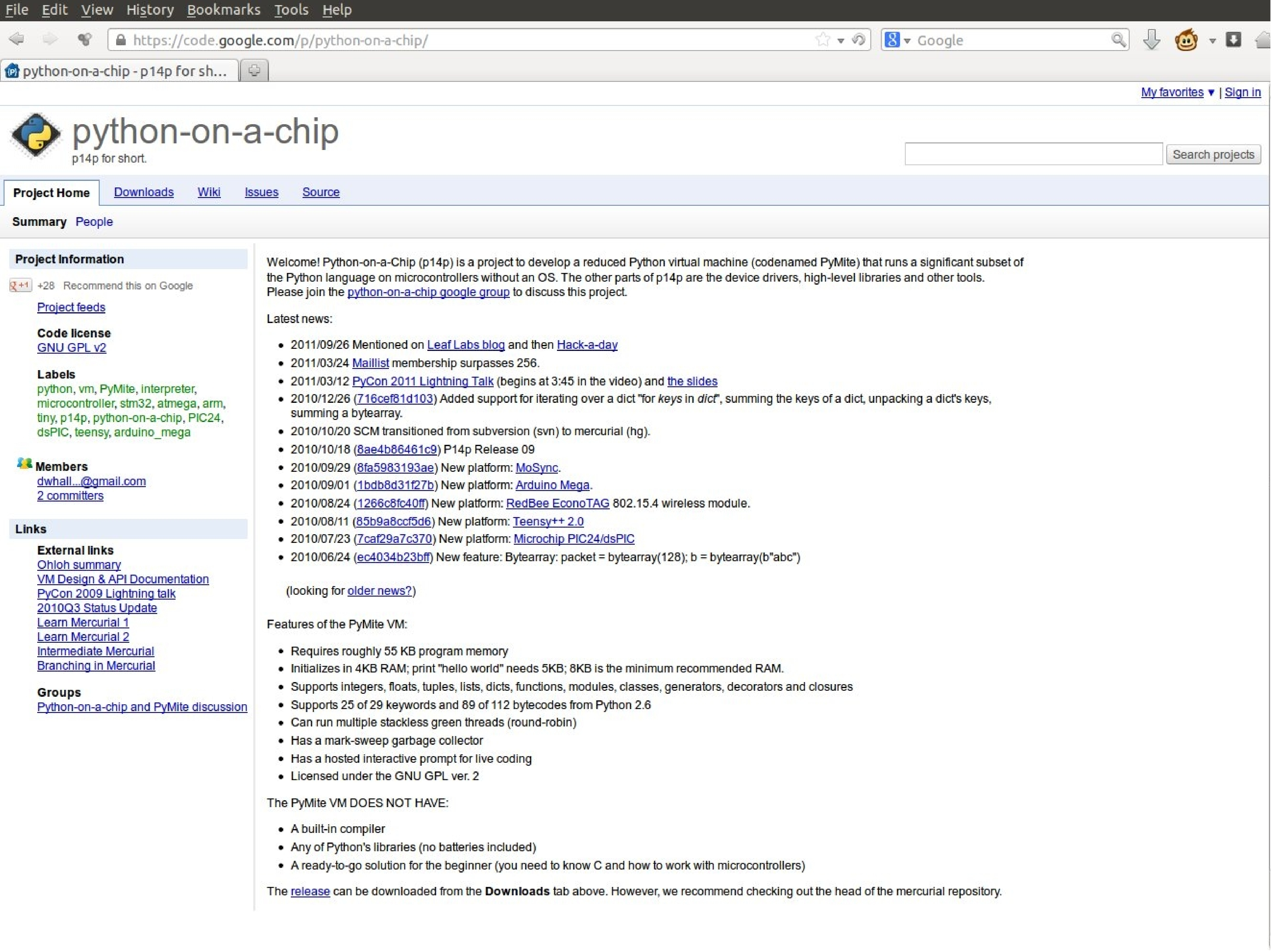
- User tools on Linux
- User tools on Mac OS X

Next, make sure that you are using a microcontroller that is supported by Owl. Currently, the only chips supported are the TI Stellaris LM3S9x9x series. We've tested the platform on other chips and they will be supported soon. If you can't wait that long, help us out by porting it!

- Programming the 9x9x on Linux
- Programming the 9x9x on Mac OS X

Finally, you're ready to connect to the microcontroller!

Using Owl


File Edit View History Bookmarks Tools Help
https://code.google.com/p/python-on-a-chip/
Google
python-on-a-chip - p14p for sh...
My favorites ▾ | Sign in
Search projects
Project Home
Downloads
Wiki
Issues
Source
Summary
People

python-on-a-chip

p14p for short.

Project Information

+28 Recommend this on Google

Project feeds

Code license
GNU GPL v2

Labels
python, vm, PyMite, interpreter, microcontroller, stm32, atmega, arm, tiny, p14p, python-on-a-chip, PIC24, dsPIC, teensy, arduino_mega

Members
dwhall...@gmail.com
2 committers

Links

External links
Ohloh summary
VM Design & API Documentation
PyCon 2009 Lightning talk
2010Q3 Status Update
Learn Mercurial 1
Learn Mercurial 2
Intermediate Mercurial
Branching in Mercurial

Groups
Python-on-a-chip and PyMite discussion

Welcome! Python-on-a-Chip (p14p) is a project to develop a reduced Python virtual machine (codenamed PyMite) that runs a significant subset of the Python language on microcontrollers without an OS. The other parts of p14p are the device drivers, high-level libraries and other tools. Please join the python-on-a-chip google group to discuss this project.

Latest news:

- 2011/09/26 Mentioned on Leaf Labs blog and then Hack-a-day
- 2011/03/24 Maillist membership surpasses 256.
- 2011/03/12 PyCon 2011 Lightning Talk (begins at 3:45 in the video) and the slides
- 2010/12/26 (716cef81d103) Added support for iterating over a dict "for *keys* in *dict*", summing the keys of a dict, unpacking a dict's keys, summing a bytearray.
- 2010/10/20 SCM transitioned from subversion (svn) to mercurial (hg).
- 2010/10/18 (8ae4b86461c9) P14p Release 09
- 2010/09/29 (8fa5983193ae) New platform: MoSync.
- 2010/09/01 (1bdb8d31f27b) New platform: Arduino Mega.
- 2010/08/24 (1266c8fc40ff) New platform: RedBee EconoTAG 802.15.4 wireless module.
- 2010/08/11 (85b9a8ccf5d6) New platform: Teensy++ 2.0
- 2010/07/23 (7caf29a7c370) New platform: Microchip PIC24/dsPIC
- 2010/06/24 (ec4034b23bff) New feature: Bytearray: packet = bytearray(128); b = bytearray(b"abc")

(looking for older news?)

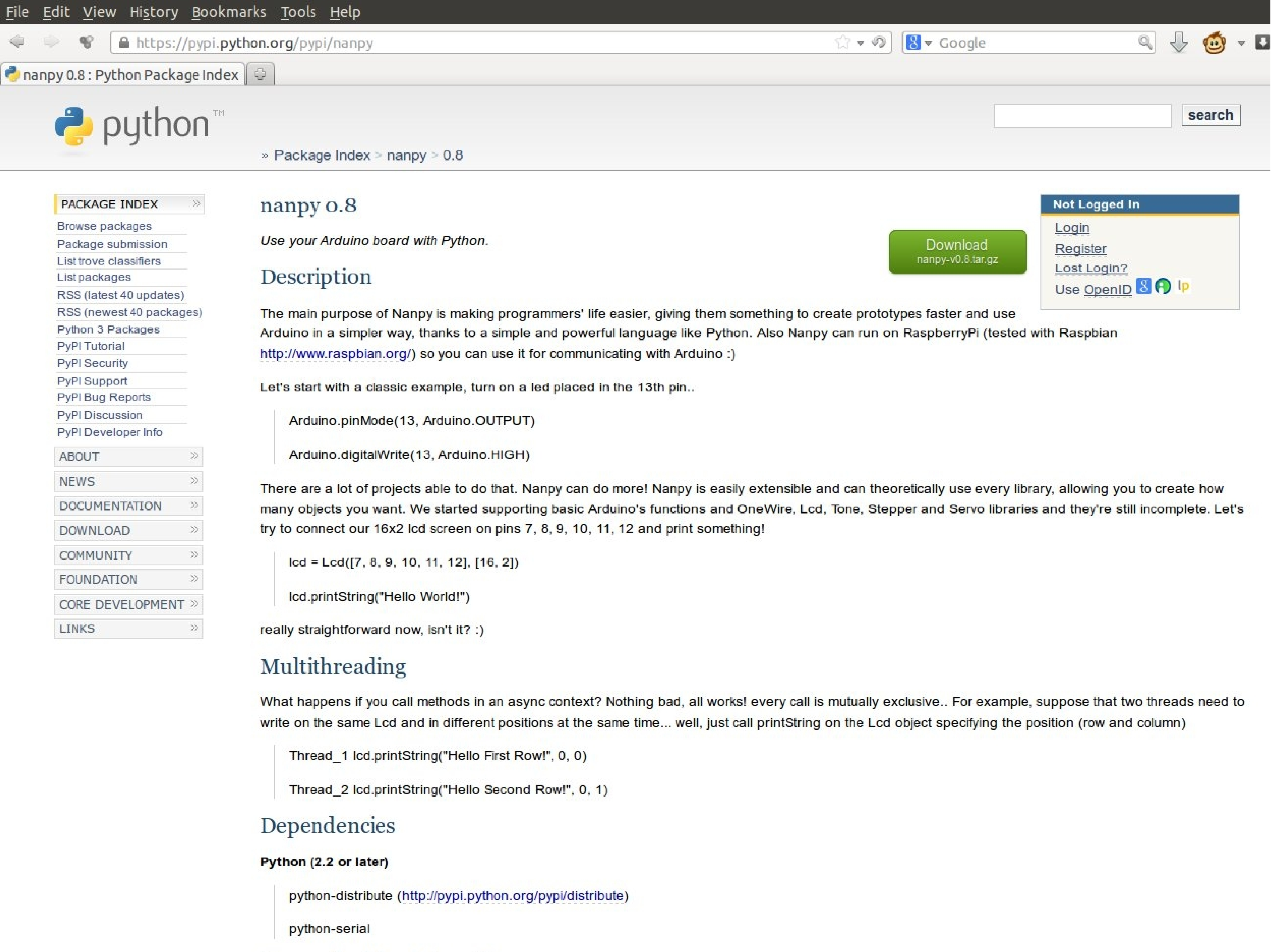
Features of the PyMite VM:

- Requires roughly 55 KB program memory
- Initializes in 4KB RAM; print "hello world" needs 5KB; 8KB is the minimum recommended RAM.
- Supports integers, floats, tuples, lists, dicts, functions, modules, classes, generators, decorators and closures
- Supports 25 of 29 keywords and 89 of 112 bytecodes from Python 2.6
- Can run multiple stackless green threads (round-robin)
- Has a mark-sweep garbage collector
- Has a hosted interactive prompt for live coding
- Licensed under the GNU GPL ver. 2

The PyMite VM DOES NOT HAVE:

- A built-in compiler
- Any of Python's libraries (no batteries included)
- A ready-to-go solution for the beginner (you need to know C and how to work with microcontrollers)

The release can be downloaded from the **Downloads** tab above. However, we recommend checking out the head of the mercurial repository.



nanpy 0.8

Use your Arduino board with Python.

Description

The main purpose of Nanpy is making programmers' life easier, giving them something to create prototypes faster and use Arduino in a simpler way, thanks to a simple and powerful language like Python. Also Nanpy can run on RaspberryPi (tested with Raspbian http://www.raspbian.org/) so you can use it for communicating with Arduino :)

Let's start with a classic example, turn on a led placed in the 13th pin..

> Arduino.pinMode(13, Arduino.OUTPUT)
>
> Arduino.digitalWrite(13, Arduino.HIGH)

There are a lot of projects able to do that. Nanpy can do more! Nanpy is easily extensible and can theoretically use every library, allowing you to create how many objects you want. We started supporting basic Arduino's functions and OneWire, Lcd, Tone, Stepper and Servo libraries and they're still incomplete. Let's try to connect our 16x2 lcd screen on pins 7, 8, 9, 10, 11, 12 and print something!

> lcd = Lcd([7, 8, 9, 10, 11, 12], [16, 2])
>
> lcd.printString("Hello World!")

really straightforward now, isn't it? :)

Multithreading

What happens if you call methods in an async context? Nothing bad, all works! every call is mutually exclusive.. For example, suppose that two threads need to write on the same Lcd and in different positions at the same time... well, just call printString on the Lcd object specifying the position (row and column)

> Thread_1 lcd.printString("Hello First Row!", 0, 0)
>
> Thread_2 lcd.printString("Hello Second Row!", 0, 1)

Dependencies

Python (2.2 or later)

> python-distribute (http://pypi.python.org/pypi/distribute)
>
> python-serial

Main Menu
Calendar
Clock
Contact
Date & Time

Raspberry Pi

CE0168
Telit

Tempi di sviluppo rapidi

Supporto di più paradigmi
di programmazione

Estensibile

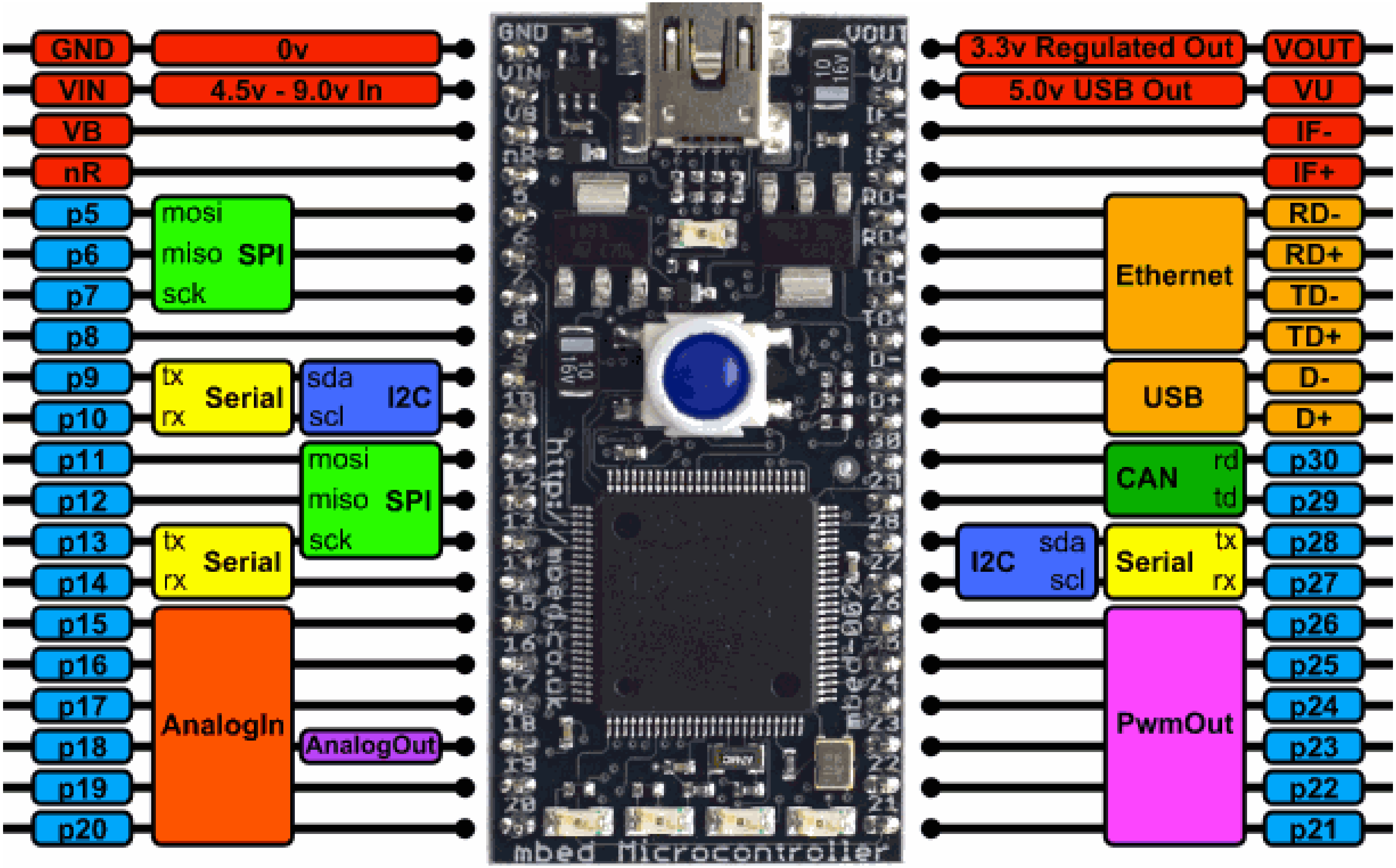
Lento?

Open source

Linguaggio semplice
da imparare

Portabile

Più di 8000 librerie già pronte


GND
0v
VIN
4.5v - 9.0v In
VB
nR
p5
p6
p7
mosi
miso
sck
SPI
p8
p9
p10
tx
rx
Serial
sda
scl
I2C
p11
p12
mosi
miso
sck
SPI
p13
p14
tx
rx
Serial
p15
p16
p17
p18
p19
p20
AnalogIn
AnalogOut
mbed Microcontroller
http://mbed.co.uk
3.3v Regulated Out
VOUT
5.0v USB Out
VU
IF-
IF+
Ethernet
RD-
RD+
TD-
TD+
USB
D-
D+
CAN
rd
td
p30
p29
I2C
sda
scl
Serial
tx
rx
p28
p27
PwmOut
p26
p25
p24
p23
p22
p21

socket

Py SERIAL

PyUSB

pyVisa

GPIO driver
in User Space

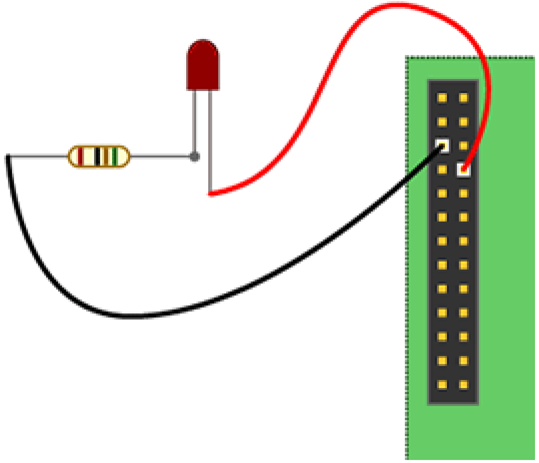
Esportazione

```
echo 11 > /sys/class/gpio/export
echo 12 > /sys/class/gpio/export
```

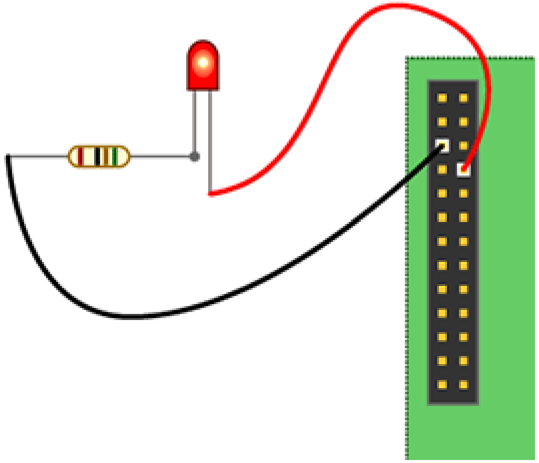
Direzione

```
echo out > /sys/class/gpio/gpio11/direction
echo in > /sys/class/gpio/gpio12/direction
```

Utilizzo



```
echo 0 > /sys/class/gpio/gpio11/value
```



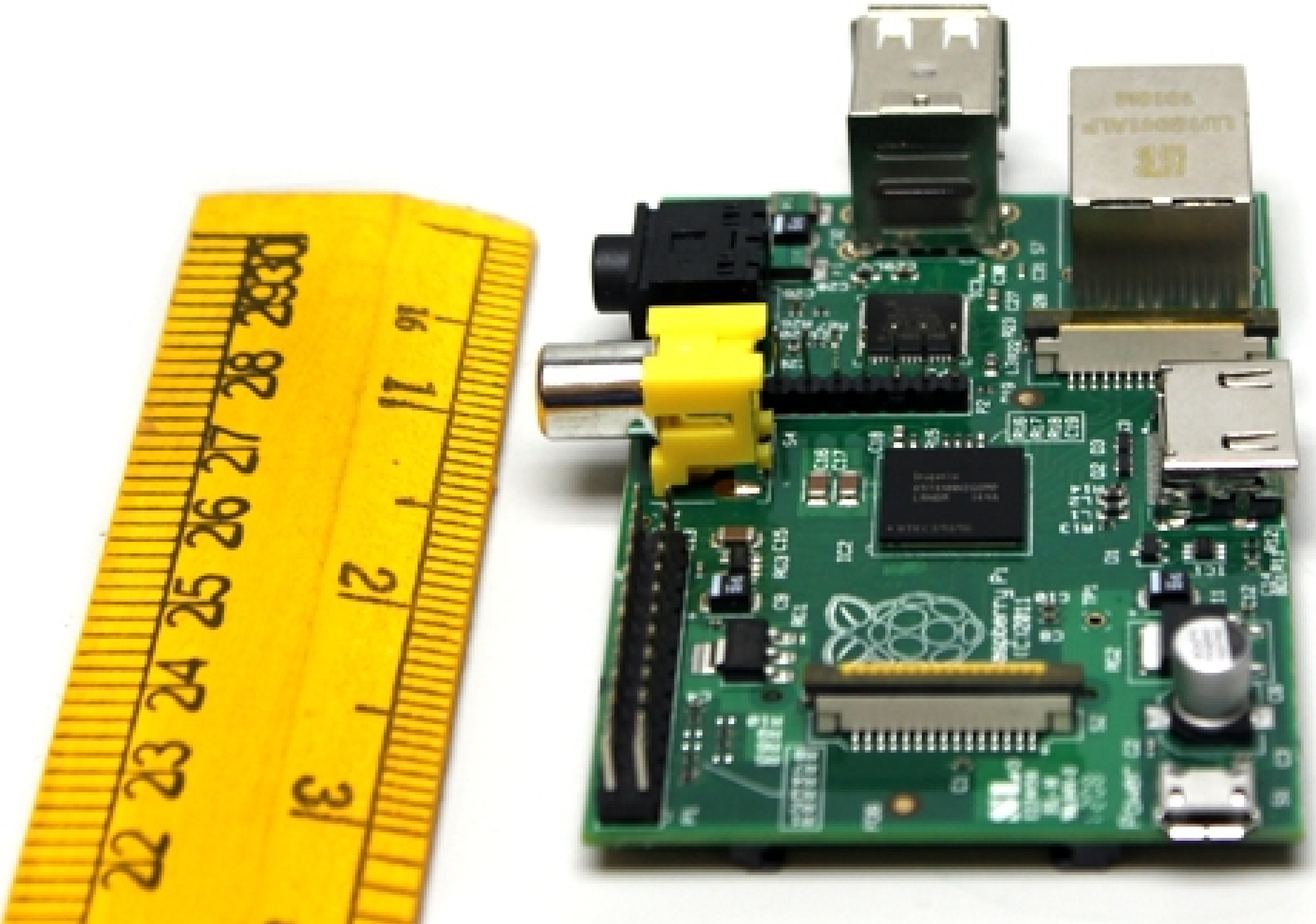
```
echo 1 > /sys/class/gpio/gpio11/value
```

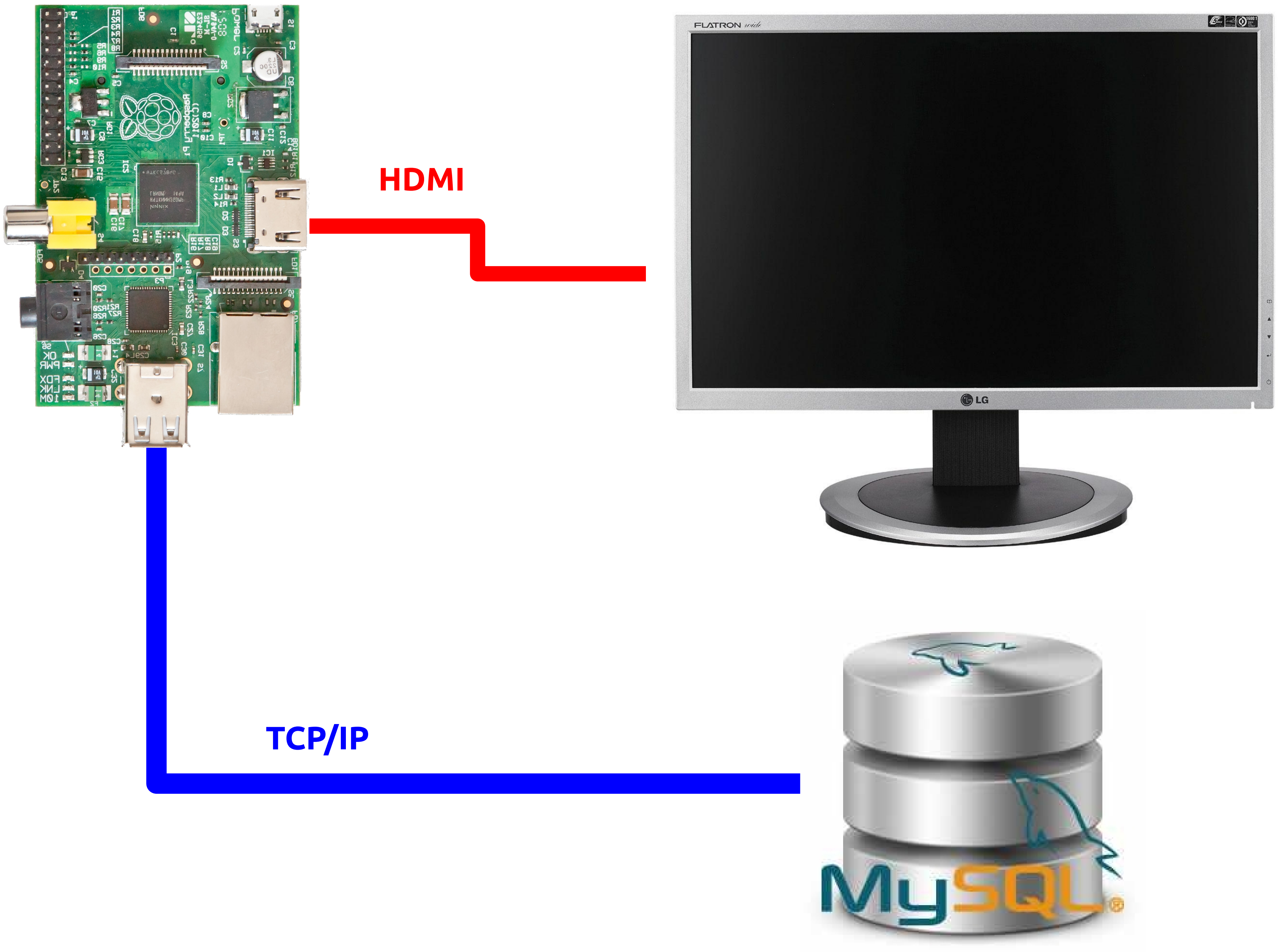
GPIO driver
in Python

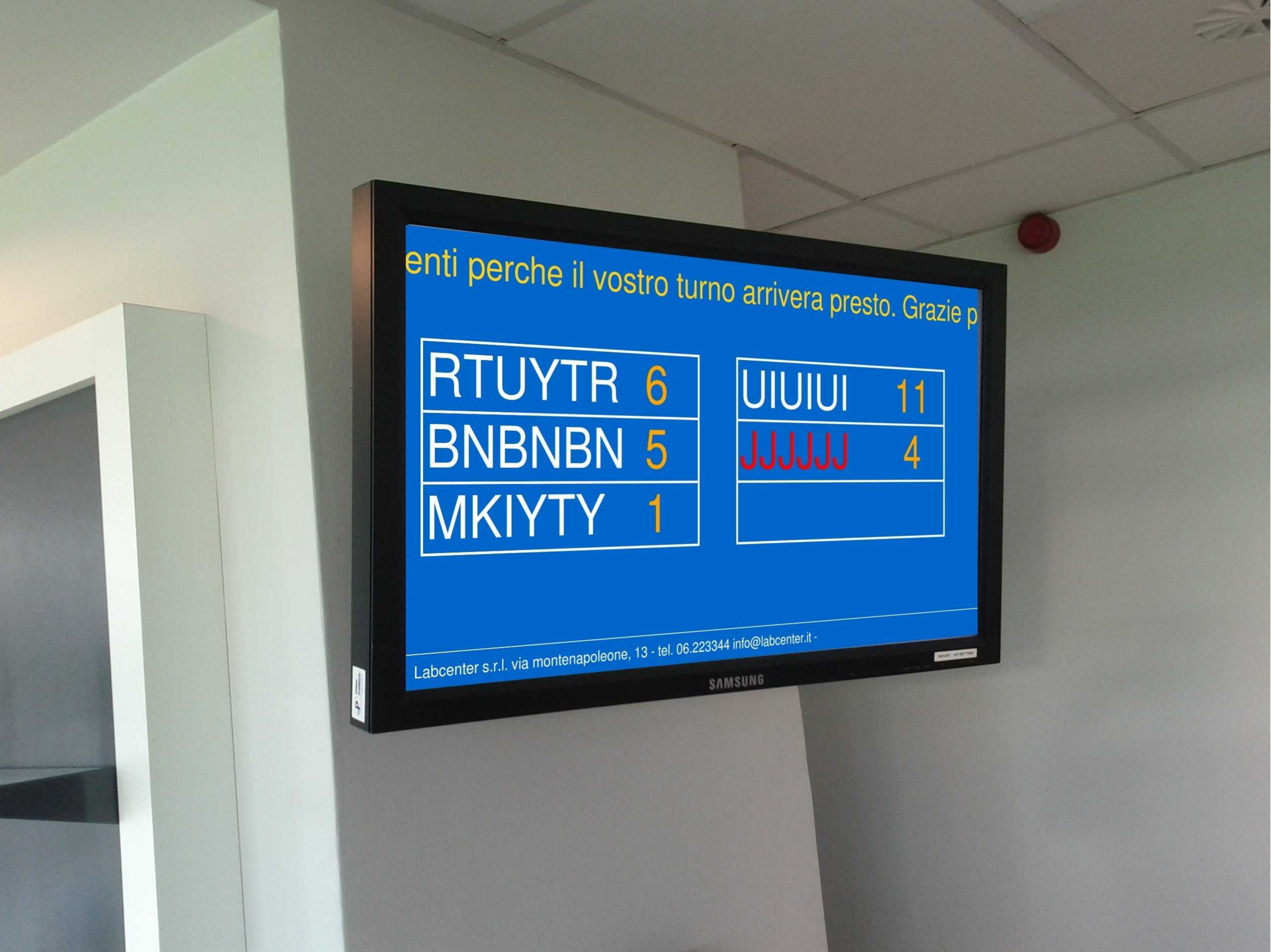
```
def export():
    f = open('/sys/class/gpio/export','w')
    f.write('11')
    f.close()

def direction():
    f = open('/sys/class/gpio/gpio11/direction','w')
    f.write('out')
    f.close()

def pin(value):
    f = open('/sys/class/gpio/gpio11/value','w')
    f.write(str(value))
    f.close()
```

Caso di studio

Display LCD
per sale di attesa




HDMI
TCP/IP
MySQL



enti perche il vostro turno arrivera presto. Grazie p

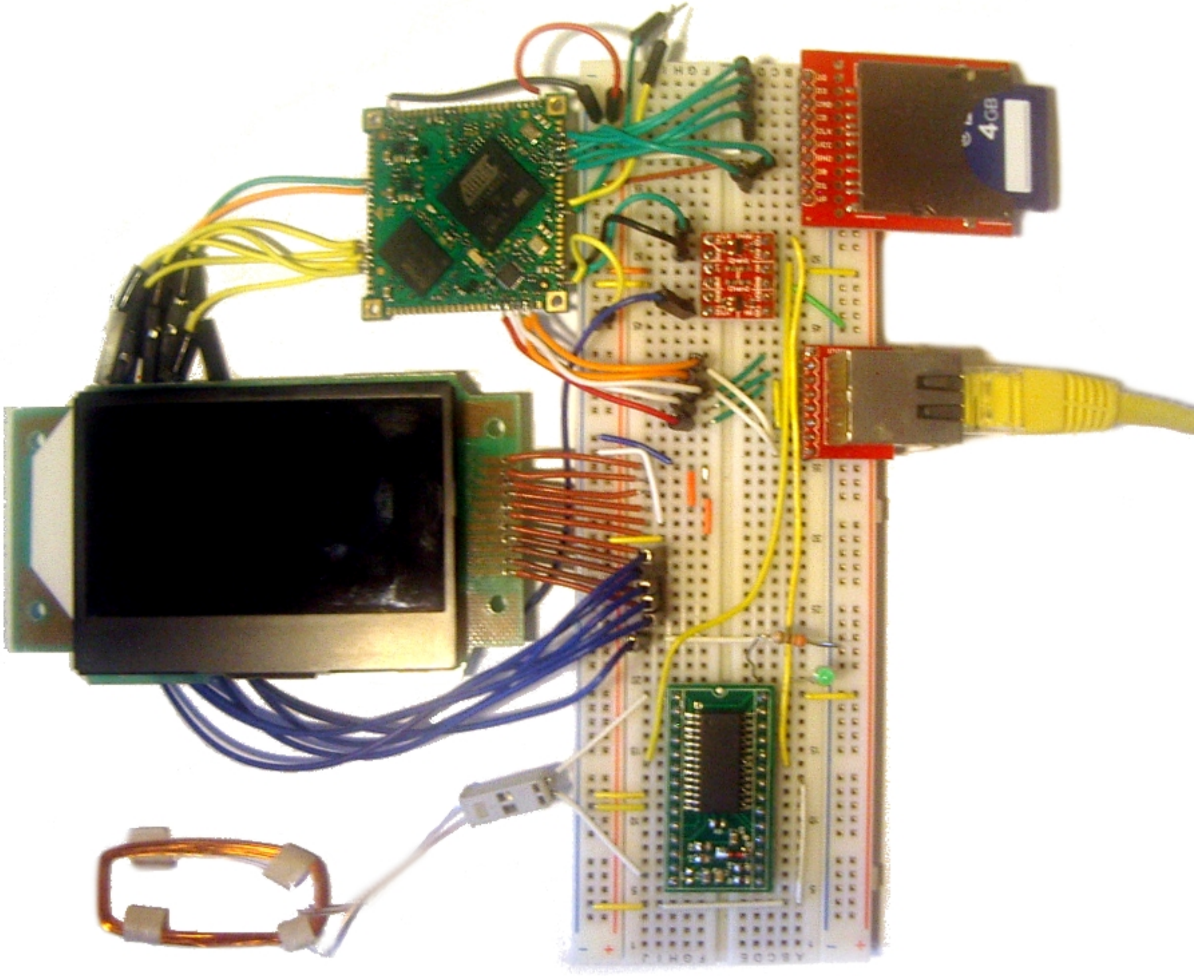
RTUYTR	6
BNBNBN	5
MKIYTY	1

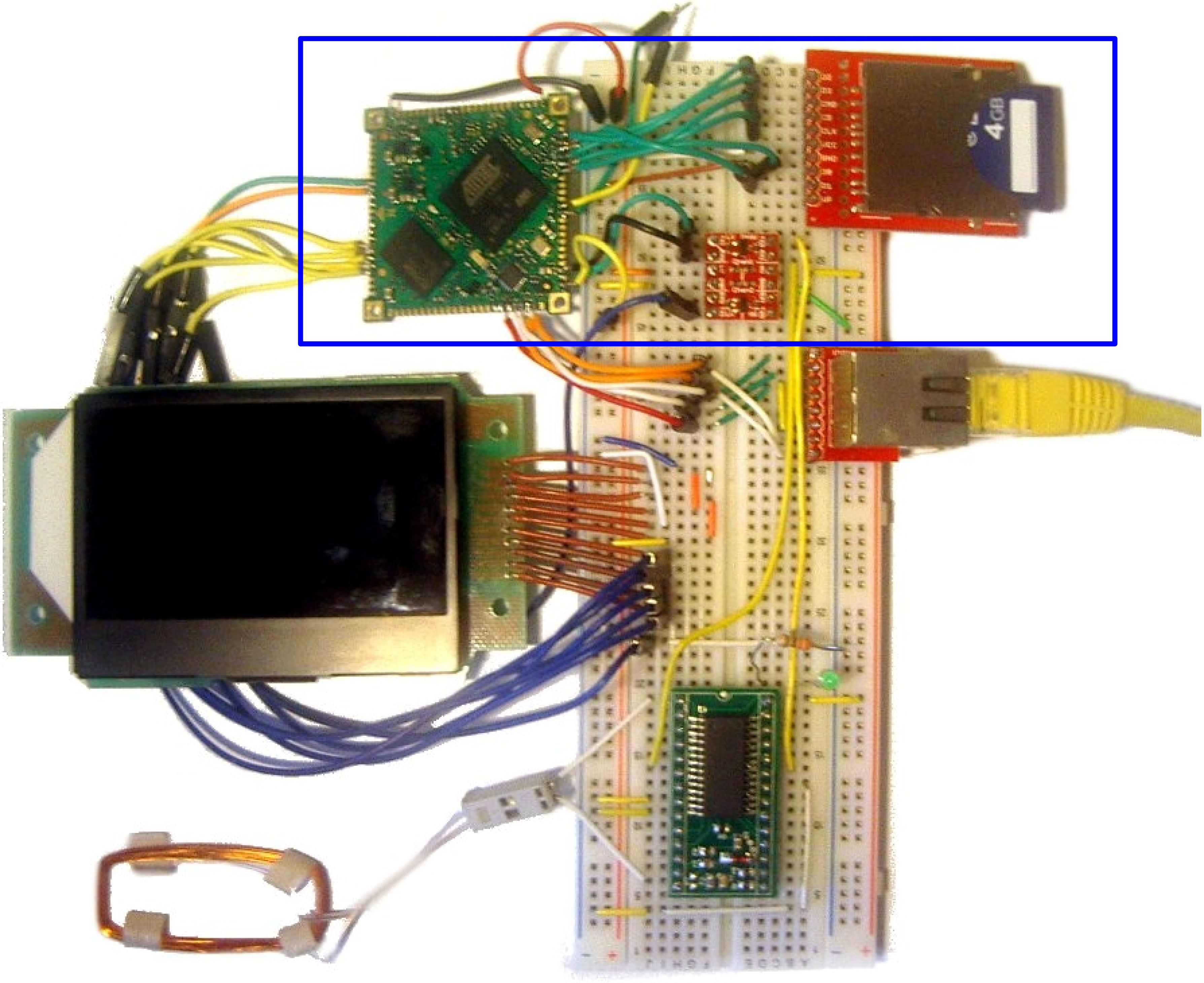
UIUIUI	11
JJJJJJ	4

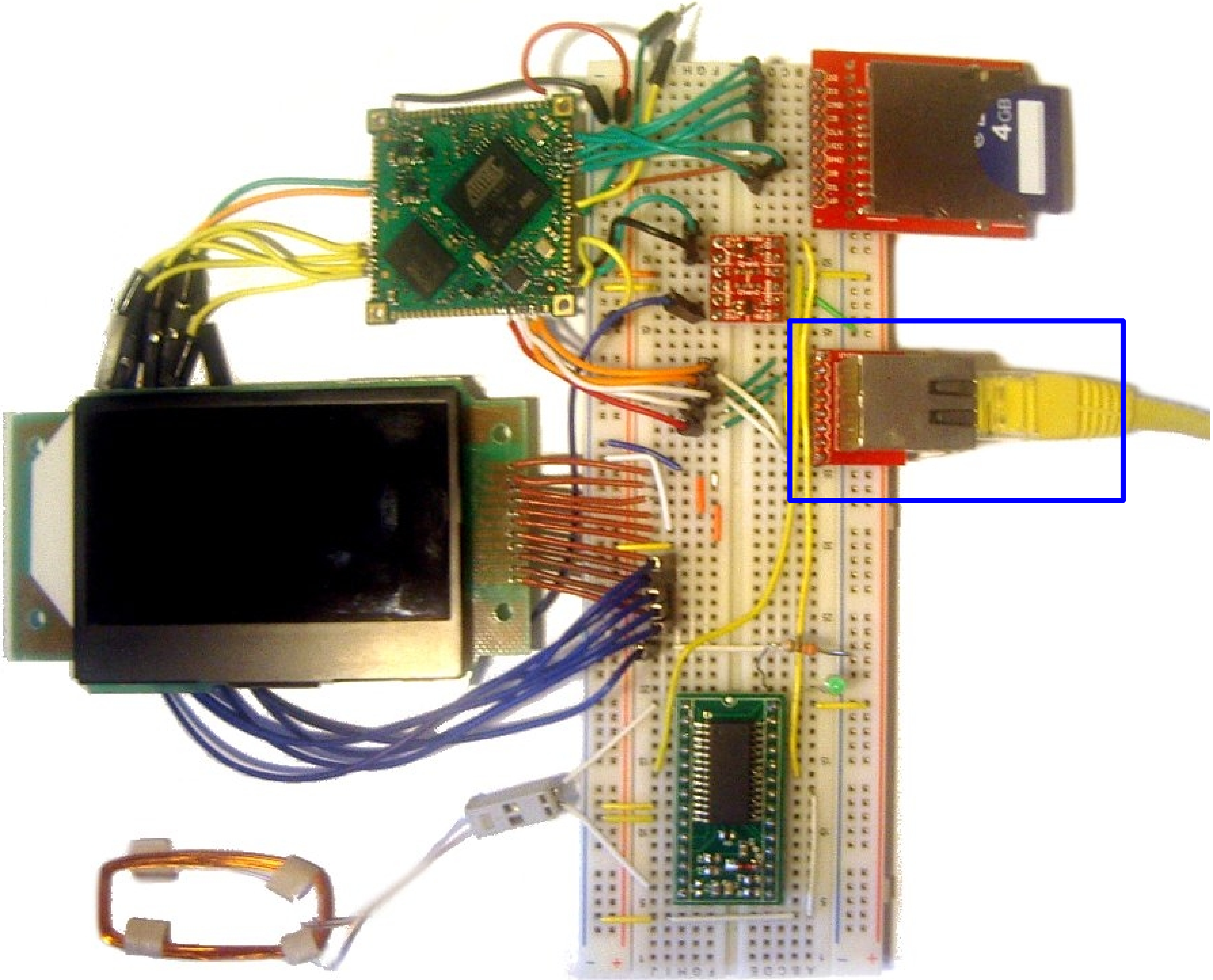
Labcenter s.r.l. via montenapoleone, 13 - tel. 06.223344 info@labcenter.it -

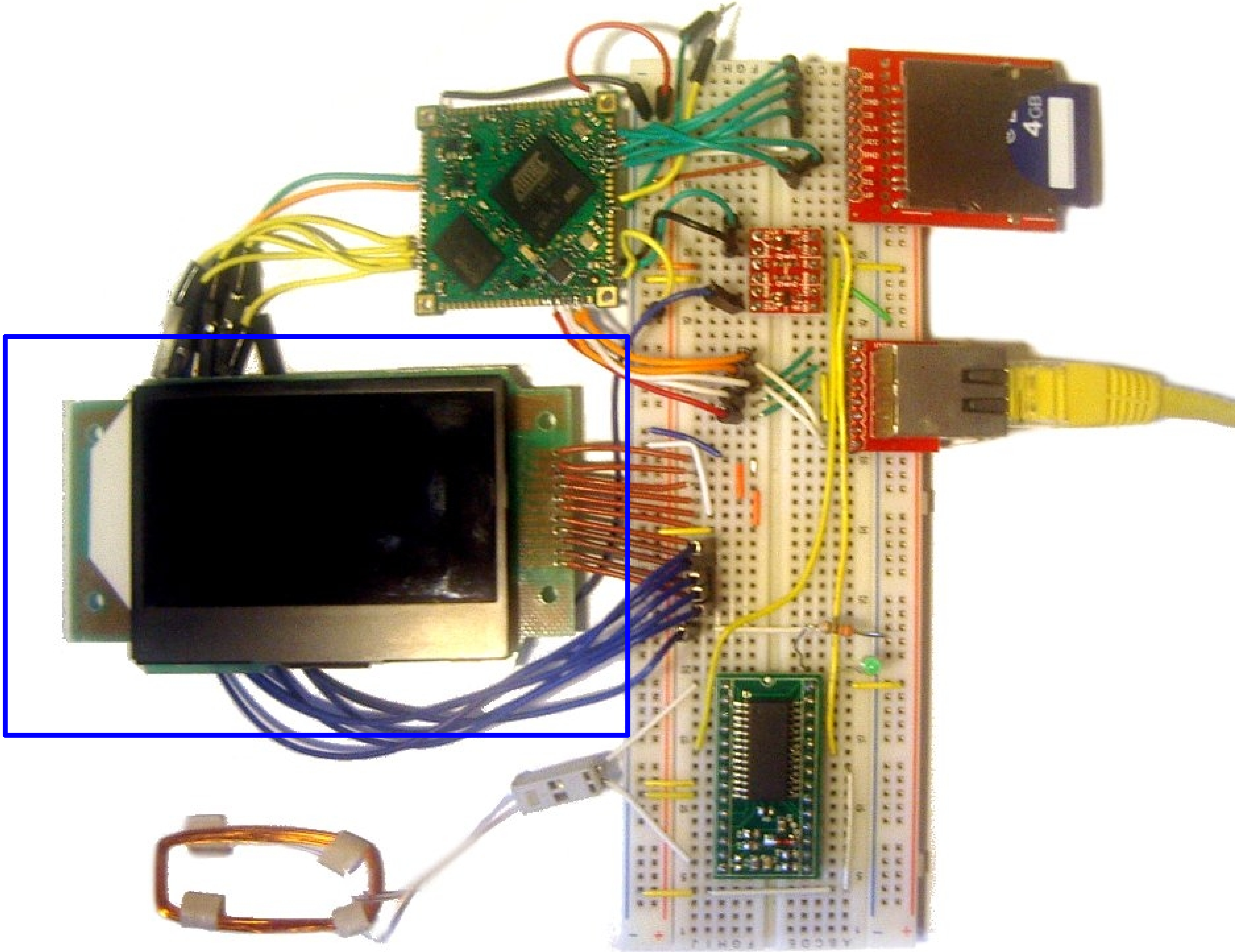
Caso di studio

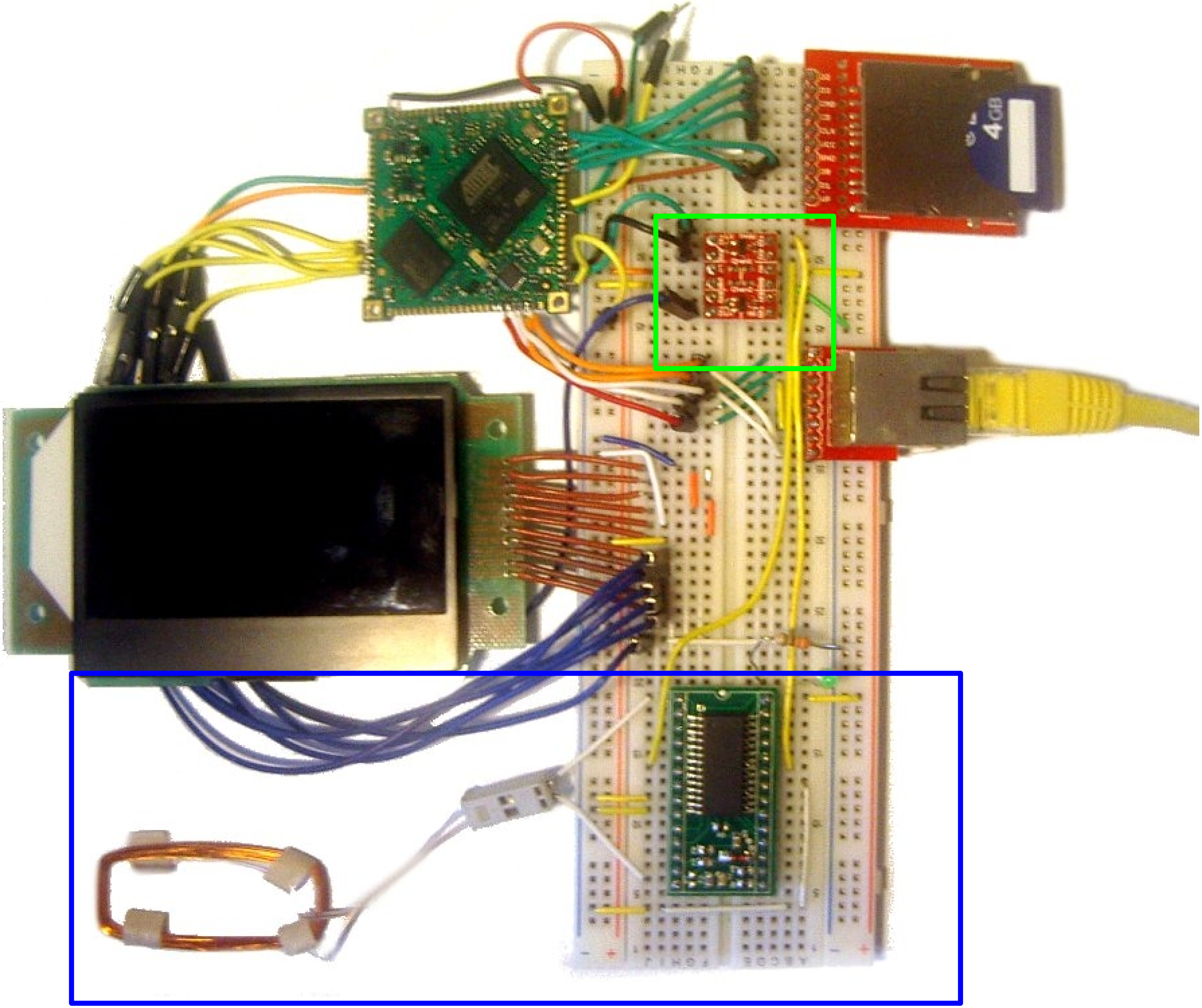
Semplice timbracartellino

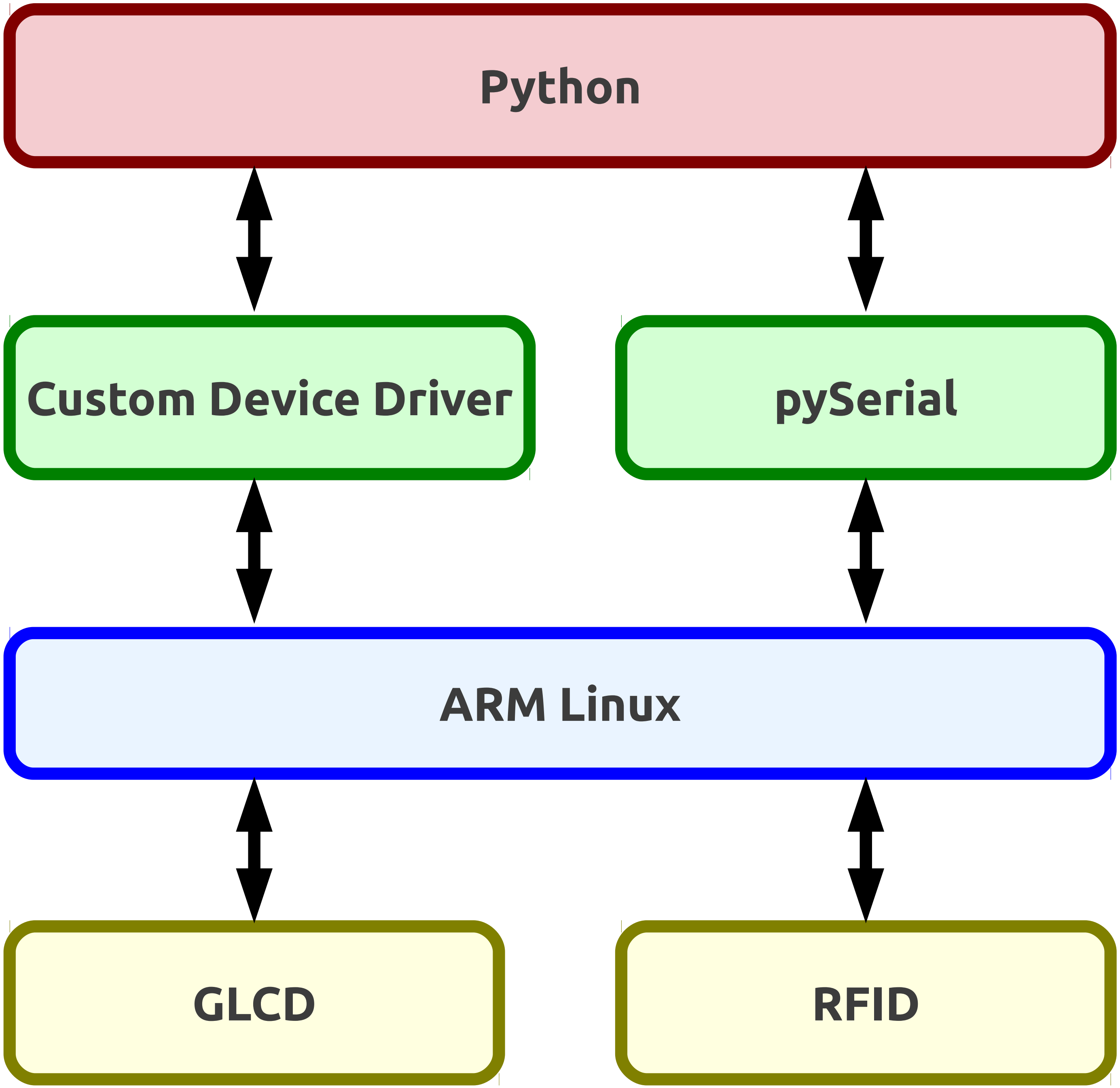


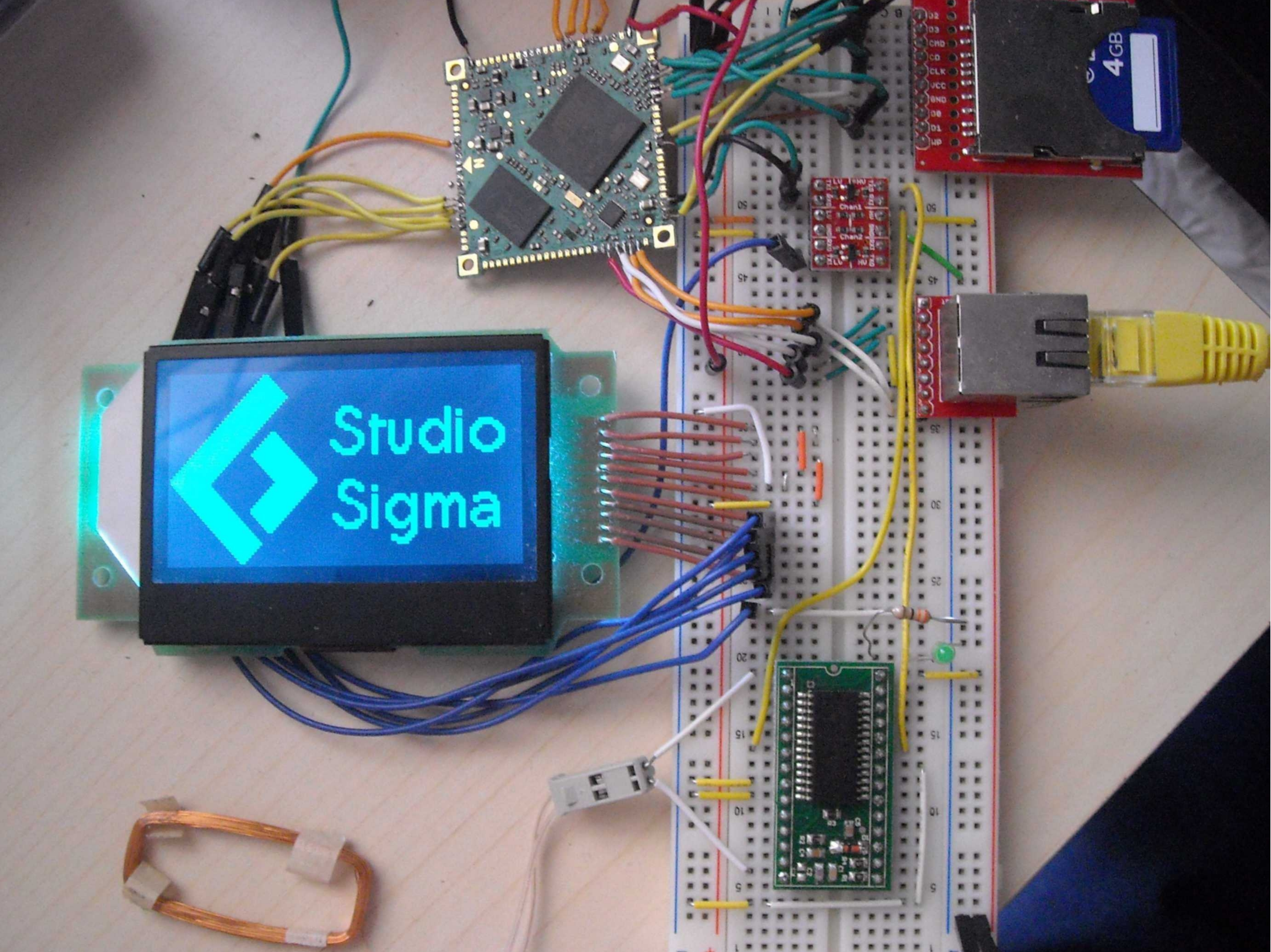

4GB


4GB


4GB


4GB


Python
Custom Device Driver
pySerial
ARM Linux
GLCD
RFID


Studio
Sigma
4GB
D2
D3
CMD
CD
CLK
VCC
GND
D0
D1
WP
Chan1
Chan2
50
45
35
30
25
20
15
10
5

STUDIO SIGMA

studio tecnico di ingegneria dell'informazione

phone
+39 320 26 75 777
mail
info@studiosigma.pro
web
www.studiosigma.pro