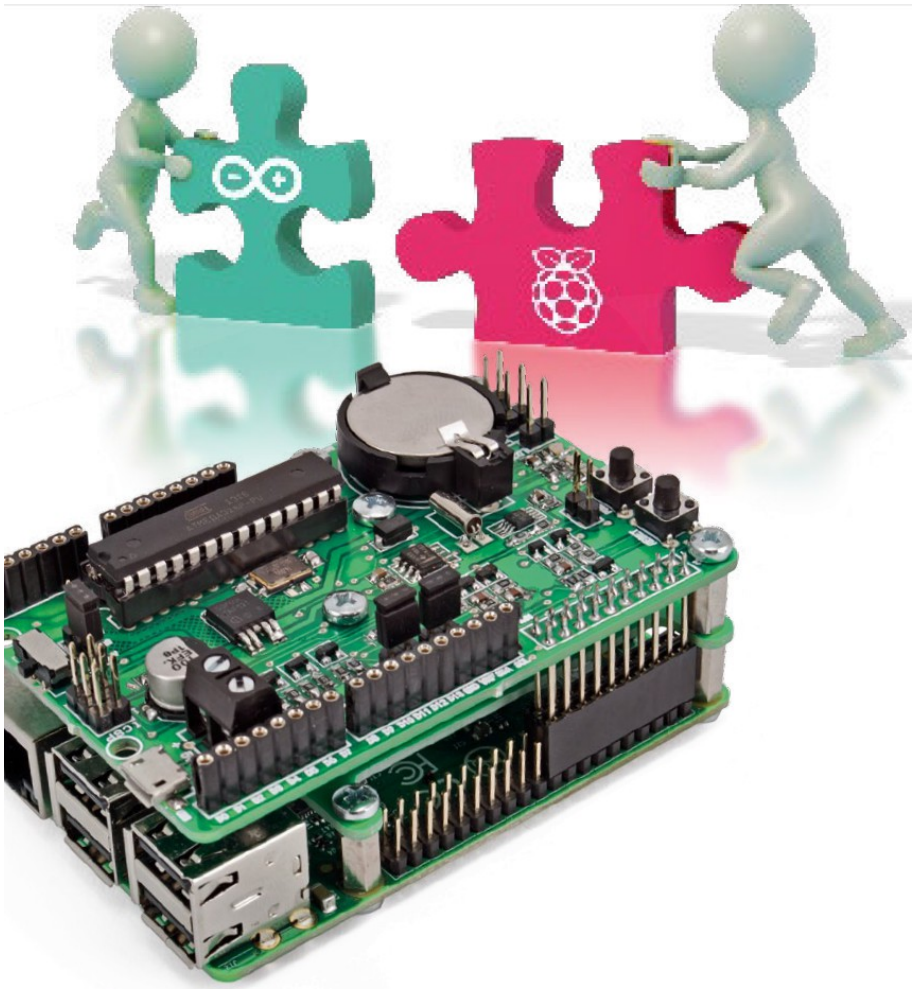
Raspberry and Arduino : RandA

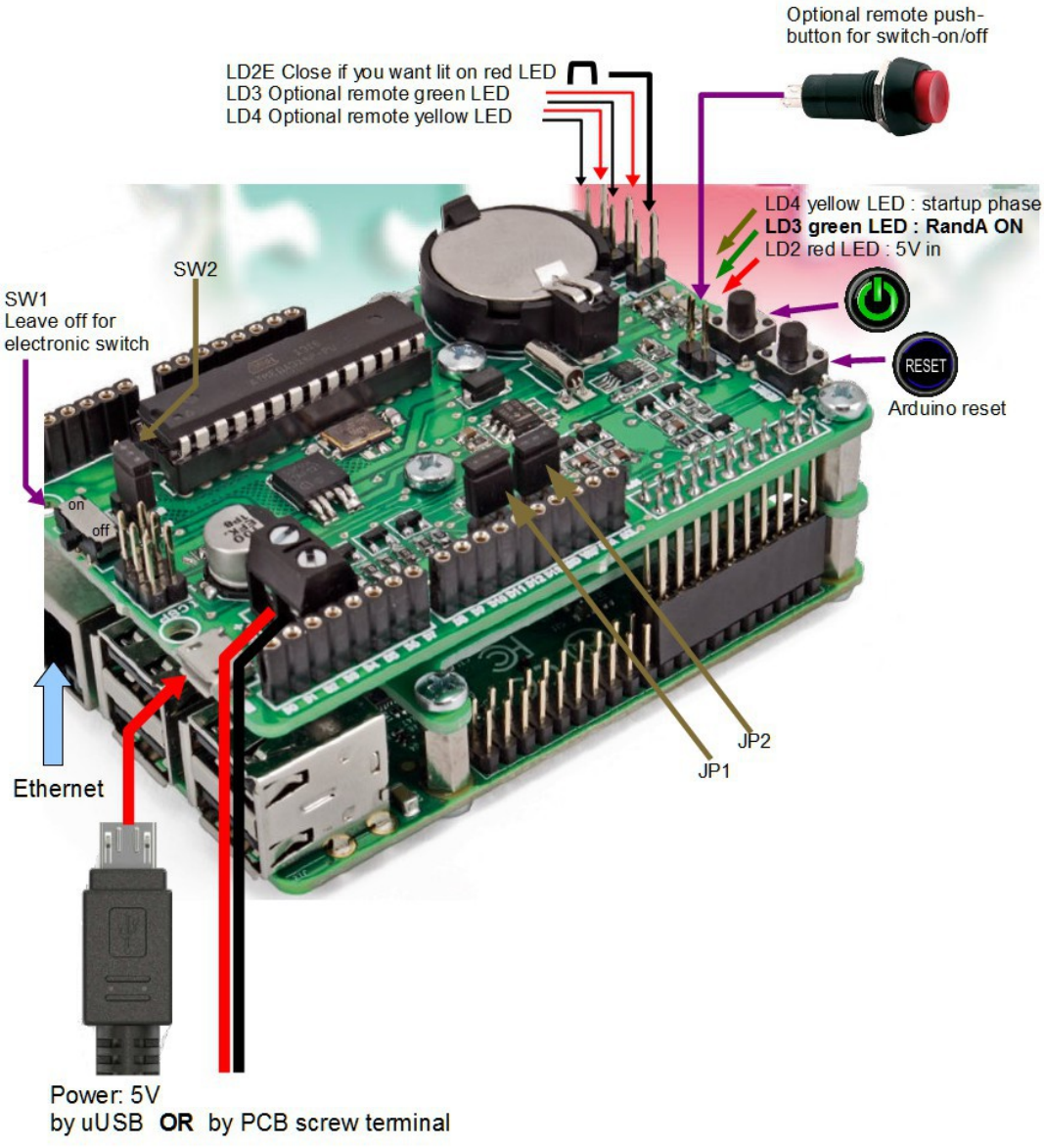
Handbook

Author: Daniele Denaro



Index

Links


Optional remote push-button for switch-on/off
LD2E Close if you want lit on red LED
LD3 Optional remote green LED
LD4 Optional remote yellow LED
LD4 yellow LED : startup phase
LD3 green LED : RandA ON
LD2 red LED : 5V in
SW2
SW1
Leave off for electronic switch
on
off
RESET
Arduino reset
JP2
JP1
Ethernet
Power: 5V
by uUSB OR by PCB screw terminal

RandA basic

Raspberry and Arduino are very known experimental platforms. Therefore there are a lot of trials of combined use. RandA system pretends to be the best solution for this purpose. If Raspberry is a complete system with a serious O.S. like Linux, however Arduino is a very simple I/O management system and it has a useful IDE for programming. Moreover, Arduino can rely on several add on "shields" for a lot of different purposes.

RandA (Raspberry and Arduino) allows the two systems collaboration by three ways:

a) Arduino operates like a Raspberry programmable device; and its program (sketch) is loaded by Raspberry on the basis of its purposes. Arduino compiled sketches library has to lie on Raspberry, ready to be uploaded.

b) Arduino operates like a Raspberry controller and can send commands to use the complex features of a Linux system.

c) As previous point, but Arduino is always ON and can switch ON/OFF Raspberry in addition.

Actually, RandA system provides a power supply management too. A Real Time Clock, with wake-up, and a circuit for soft shutdown or start-up are included on RandA. So, RandA is not just an Arduino hardware but also a Raspberry upgrade.

The hardware link, between the two systems, is made by the serial port that both have. But we also try to link the two different software environments.

RandA software includes:

a) Software for power management.

b) Software for Raspberry-Arduino communication.

c) Software for utilization assistance.

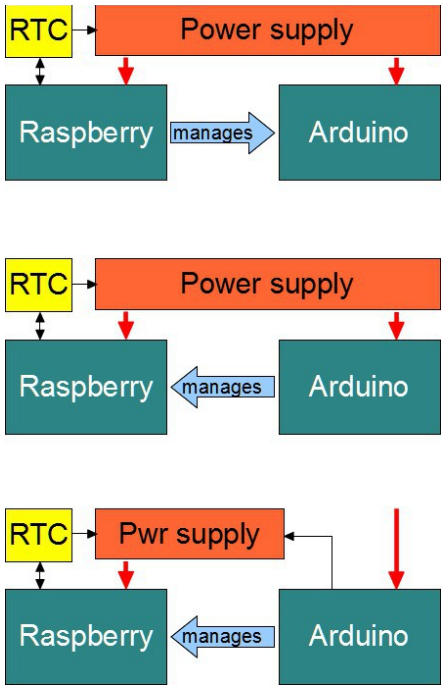


Fig. 1 - Different managements

Substantially you can:

1. switch ON/OFF the complete system by a push button; that is, a background process on Raspberry starts the shutdown procedure and power off;
2. use prepared command on Raspberry to set RTC and the automatic wake-up;
3. use a modified Arduino IDE that detects a remote port on LAN, so you can upload sketches remotely; that is, a background process on Raspberry introduces itself on LAN and realizes a remote serial port;
4. use a Arduino library to write sketches that sends command to Raspberry or writes/reads files; that is, a background process on Raspberry can detect required attention on serial port and listen commands;
5. use prepared command on Raspberry to upload compiled sketch on Arduino;
6. use prepared command on Raspberry to send email;

But you can also use a WEB server (Tomcat based) for manage Raspberry an Arduino with basic functionality. And, of course, you have the complete Linux system to utilize.

RandA hardware

RandA is a “shield” for Raspberry that uses its connector bus for communication and power management.

Push button is joined with power circuit and with GPIO24 for shutdown.

The RTC is linked with Raspberry by I^2C bus.

RTC alarm manages power switch as well as the push button.

But also Arduino pin D4 is connected with power switch. So Arduino can switch power ON/OFF.

Arduino is linked with Raspberry by serial port (D0,D1-GPIO14,GPIO15).

But, also, Arduino Reset pin is connected with GPIO23 Raspberry pin. So, Raspberry can reset Arduino in this way.

Finally, Arduino pin D2 is connected with GPIO22 to request attention for command protocol starting.

An hardware switch can disable software power switch for independent supply settling.

Another switch can disable software power switch just for Arduino. So Arduino can be supplied for ever.

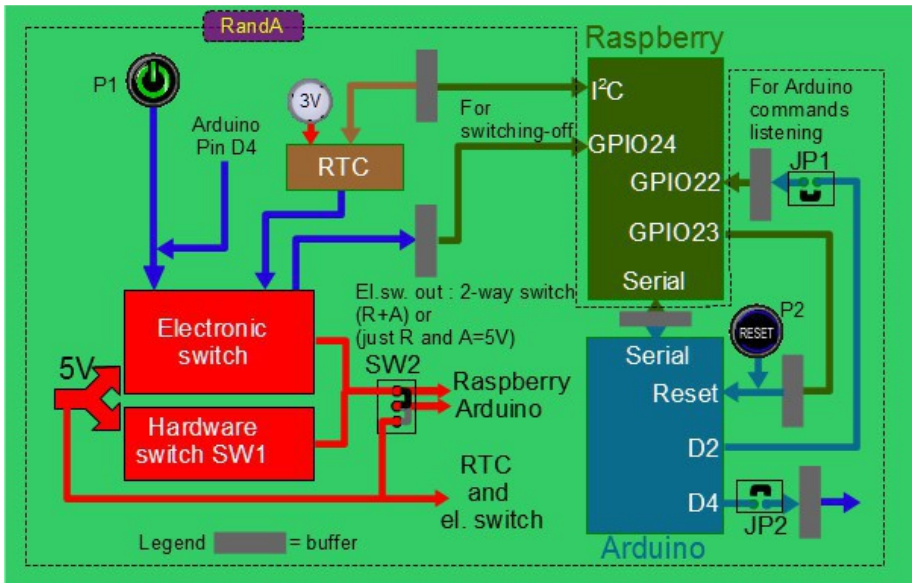


Fig. 2 - RandA functional blocks

So, if you want to switch on/off together Raspberry and Arduino, by RTC or by P1 leave SW2 as in Fig.2. Power-on can be made by push-button or by programmed RTC. Power-off can be made by push-button (that start shutdown) or by Linux command “sudo shutdown -h now) or by Arduino D4.

If you want to leave Arduino always powered, and switch on/off just Raspberry, move connector down (Fig.2 SW2) and close JP2 (for Raspberry switching-on by Arduino).

If you want to disable Raspberry listening for Arduino commands, open IP1 (or disable software).

RTC module

The RTC chip is DS1339. The I^2C bus finds the IC on 0xFFD address (0xFFD0 for reading and 0xFFD1 for writing). SCL and SDA lines use the voltage converters (3V<->5V) realized by the T6 and T7 CMOS transistors.

The DS1339 IC is supplied by 5V power socket (uUSB or pins), but it has also a continuity 3V battery. Its inactivated alarm switch off the power supply setting high the gate of power P channel CMOS T7.

The alarm activation, sets to low level the gate of T7, and T7 powers the system. Because the on condition of DS1339 alarm, the circuit needs a low pass filter.

The alarm can be programmed in terms of day, hour(0-23), minute.(0-59) Where day can be month day (1-31) or week day(1-7). But it can be programmed also just as hour and minutes or minutes only.

The DS1339 has two alarms, but only the second one is used by the installed software.

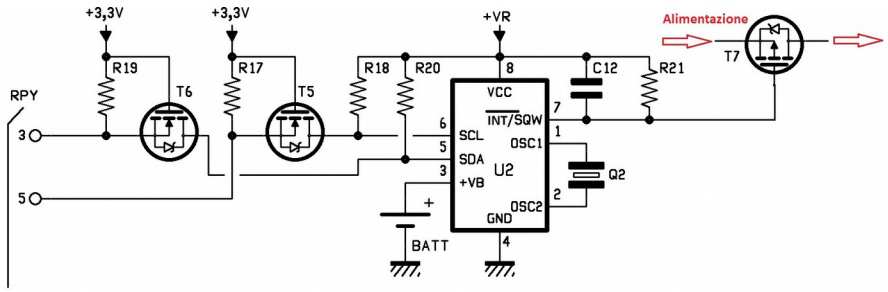


Fig. 3 - RTC diagram

C programs manages RTC and its alarm. These programs use the “wiringPi” library. A copy of these C programs, with sources, are located on “cworkspace” directory as examples.

A Java version of these programs is provided as well, as examples on “jworkspace” directory. And the “pi4j” library is also installed.

Power switching module

The 5 Volt power supply goes in through the uUSB connector or through the screwed terminal. Be careful because it doesn't exist fuse any more! The supply is managed by the T7 CMOS power transistor acting as a switch. T7 is controlled by RTC or by the push button circuit. The push button circuit is made by a 555 classic timer IC (used as flip-flop) and a couple of transistors.

For ON phase, the 555 timer switches on the T7. But the timer is reset by a starting program on Raspberry that activates alarm putting to low level the T7 gate. In this way Raspberry confirms the Power ON and it is ready to detect the push button for switching off .

For OFF phase, a raspberry program intercepts the changing state on GPIO24 (pin 18) and start the shutdown procedure that at the last step runs a PowerOff C program. This program inactivates alarm and T7 CMOS transistor.

In this way, the power switch can be driven by the push button (also several in parallel), or by the RTC alarm or by the Arduino D4 pin.

A green led LD3 flags the power-on , and a yellow led LD4 flags the start-up time. When Raspberry completes its start-up, the LD4 goes off. Leds can be replicated remotely.

The switch SW1 makes power ON for ever, bypassing T7.

The switch SW2 bypasses T7 only for Arduino, allowing independent Arduino supply.

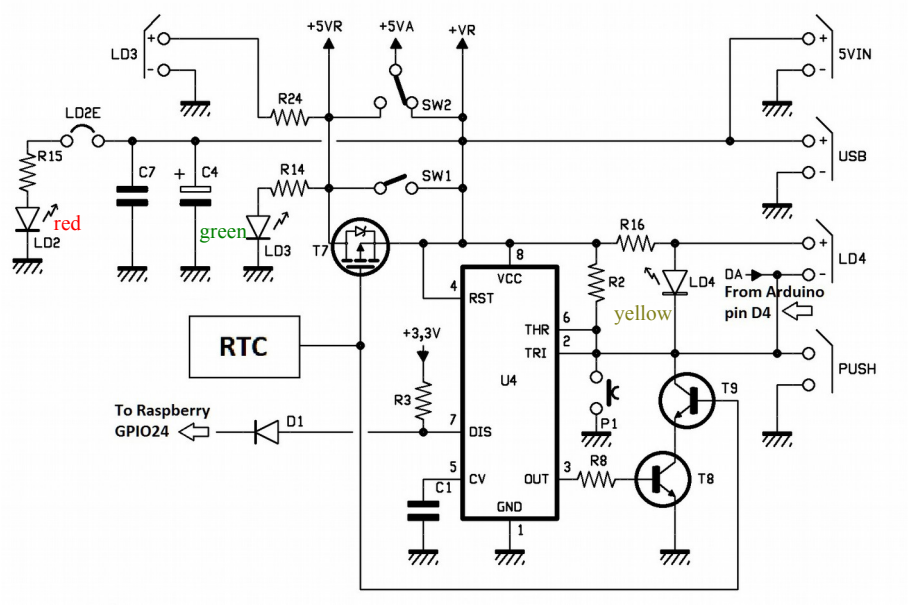


Fig. 4 - Power switching

Arduino module

Arduino module contains a ATmega328 CPU with oscillator and standard Arduino strip connectors. Buffers, as voltage adapter, are made by transistors for:

- Arduino Reset (inp); linked with GPIO23 (out)(pin 16) by capacitor
- Serial Arduino RX (D0); linked with Serial Raspberry TX (GPIO14 – pin 8)
- Serial Arduino TX (D1); linked with Serial Raspberry RX (GPIO15 – pin 10)
- Arduino pin D2 (out); linked with GPIO22 (inp) (pin 15)
- Arduino pin D4 (out)(by capacitor); parallel with push button.

The Reset command is also used for uploading. Actually, in Arduino Uno, uploading by boot-loader is made using a reset signal to alert boot-loader. But in Arduino Uno this reset signal is simulated by the USB circuit that now is missing. So the installed procedure, for uploading, simulates Reset signal by GPIO23 setting.

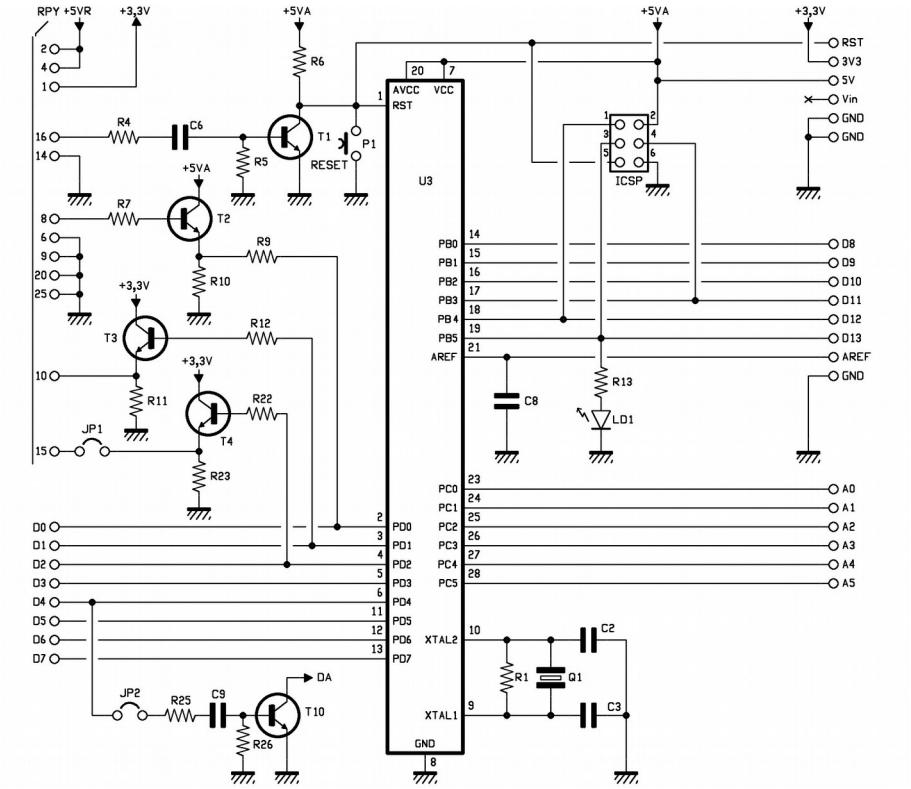


Fig. 5 - Arduino module

Raspberry used pins

Summary of used Raspberry pins:

- 5V (pin 2,4) (inp) Power from RandA
- 3.3V (pin 1) (out) From Raspberry voltage regulator
- GND (pin 6,9,14,20,25) Ground
- GPIO2 (pin 3) for SDA (I^2C bus)
- GPIO3 (pin 5) for SCL (I^2C bus)
- GPIO14 (pin 8) (out) for serial TX to RX Arduino
- GPIO15 (pin 10) (inp) for serial RX from TX Arduino
- GPIO22 (pin 15) (inp) for commands listen
- GPIO23 (pin 16) (out) Arduino Reset
- GPIO24 (pin 18) (inp) for push button shutdown detecting.

RandA software and installation

RandA shield needs some software for activating its functionalities. This software is distributed as a zip file and contains two kinds of programs.

1. Software for modified Arduino IDE; PC based. It allows to program remote Arduino on LAN
2. Software for Randa functions that has to be installed on Raspberry with Raspbian OS.

But PC software and documentation is also included in Rasberry archives and extracted on /home/pi/RandA directory for backup purpose..

Software PC based

This software modifies the Arduino IDE V1.0.5 that now can list remote serial ports too. Actually, for this purpose, the IDE communicates with RandA software present on local network and can manage this ports as well as local ports. A Remote Invocation Method (RMI) is utilized, just modifying the RXTXcomm Java library used in Arduino IDE (RXTXcomm.jar in .\lib directory).

Now, when IDE starts, it collects remote ports on LAN and names these ports as "nnn.nnn.nnn.nnn/Arduino". Where nnn.nnn.nnn..nnn is the address of RandA on local network.

Unfortunately, also a couple of Java classes has to be substitute in IDE software: AvrdudeUploader and Uploader. Both this classes are located in pde.jar (exactly in processing\app\debug path). So, pde.jar too has to be substitute (in .\lib directory).

The sketches can be compiled locally and uploaded to remote Arduino (RandA). Complied sketches are uploaded but also copied to a Raspberry directory (/home/RArduinoUpload) for further uses.

Finally, a Arduino programming library is included for Raspberry commands sending. This RAComm library activates commands protocol listen program on Raspberry, and can send commands, receive answer, and use files, from Arduino sketches. An help and examples are provided.

RAComm library is described further on.

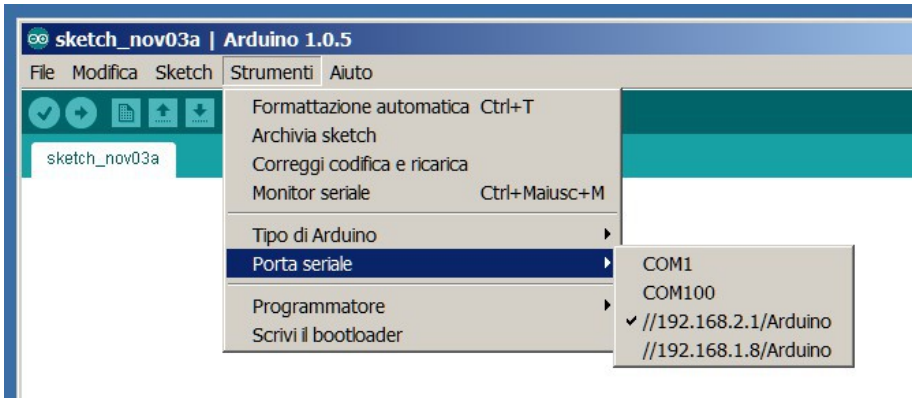


Fig. 6-Remote port

Note that both modified IDE, Remote Arduino IDE and local (Raspberry) Arduino IDE, save compiled sketches with its name and .hex extension to /home/RArduinoUpload (the first one) and to /home/ArduinoUpload (the second one), in addition to upload process. This is a modified behaviour compared to standard Arduino IDE.

Raspberry software

This software comes as the "RandAinstall.tar.gz" archives and the installation script file "RandAinstall.sh". A ReadMe file is also included.

The Raspberry side software is configured for Raspbian OS distribution only, and can be logically divided in these major sections.

1. Software for RandA functions: push button management, power switch management and Arduino commands listen. Substantially start-up and shutdown procedure (i.e. /etc/rc.local).
2. Software for RMI and remote port management. This software is located in /home/pi/raspduino directory.
3. Commands (scripts and C programs) for RandA management. Just a few of these programs are located in /etc directory but a great many of them are in /home/pi/bin directory.
4. C and Java examples in /home/pi/workspace directory.
5. Arduino compiled sketches (ready to be uploaded) in /home/RArduinoUpload and /home/ArduinoUpload directories. The first one collects remote compiled sketches, and the other one the locally compiled sketches (by Raspbian version of Arduino IDE) . These directory can also contain text files (same name of sketches but extension .descr) that explain the compiled sketches. Compiled sketches can be uploaded by a command or by Web server. Another directory ("home/pi/bin/sketch4cmd") for compiled sketches is provided. This directory is used to archive sketches used by Linux command that uses Arduino like a programmable device.
6. Web server (Tomcat based). This server is completely contained on /home/Tomcat directory. The Web server is automatically started and allows a simple remote management of RandA. You can disable Web server putting a comment char (#) at its dedicated line of start-up procedure: /etc/rc.local.
7. Programs development environments for Arduino (IDE), C (codeblocks) and Java. This software is not contained on installation archive, but it is automatically installed at the end of installation script. Note that Arduino Raspberry IDE is downloaded from the Debian site but it is modified by the installation script itself because the uploading procedure changes. Moreover, the Arduino IDE now, after uploading, copies compiled sketches to /home/Aupload directory. (N.B. At the present Java IDE is not downloaded – Eclipse is heavy but, above all, its installation, at present, overwrite Java installation)

Further software for general use is also installed :

- wiringPi: library and utility for Raspberry I/O management in C
- pi4j: equivalent library in Java
- javamail: library for email in Java. Used by SendMail command.

RandA commands (mostly in /home/pi/bin)

- The rc.local script (in /etc) is a standard Linux start-up procedure for customer initialization. In this case it contains several start-up step (i.e. execution of script or program in background mode) for RandA functionalities..
- ButtonOff.sh and PowerOff in /etc directory. The first one is a script that runs in background mode and is activated when GPIO24 changes status. In this case it runs Linux shutdown command. But the shutdown procedure finds the ups-monitor file in /etc/init.d, and executes it. In the end, ups-monitor runs PowerOff program that set alarm and switches off the power.
- **ArduLoad** : is a script that uploads compiled sketch to Arduino. Use: ArduLoad *sketch-path*
- **ArduIO**: Arduino IO management. It uses sketch SerialRasp.

- **ArduInt**: Stop until condition on digital or analogical pin. It uses SerialStop.
- **ResetRandA**: Arduino reset.
- **GetRTC**: C program that reads clock. It has help (-h).
- **SetRTC**: C program that sets clock. It has help (-h).
- **SetRestartAt**: C program that sets alarm. It has help (-h).
- **ResetAlarm**: C program that resets alarm (to the unreachable pattern 0 0 0)
- **SetSysClock**: C program that sets system clock with RTC values
- StartListenCmd and ExecSCmd. The first one is a script running in background, that reacts to changing status on GPIO22 and runs the program ExecSCmd listening serial port for command execution protocol. (Arduino RAComm library uses it). StartListenCmd uses also StartStopCmd script for serial port synchronization. StartListenCmd is started by /etc/rc.local.
- **SendMail**: Java program that sends email. It as help (-h). But it needs the Mail.properties file customization (SMTP server , user name, passw etc.).
- GetSketchName: just if sketch implements itsMe() function.
- Commands: list of these principal commands. If with param -h print helps.

Note that communication with Arduino can be made also just by direct string sending through serial port. For example : echo “WD13=1” > /dev/ttyS0 . This command ligths up Led (pin 13) if sketch “serialRasp” has been uploaded on Arduino.

Further scripts are defined just to run useful graphical programs in detached mode: explorer, Arduino, codeblocks etc.

Let us to suggest you using a remote SSH console program like MobaXterm that includes a X server. In this way you can use Raspberry (and RandA) as well as with wired keyboard and screen. Because Raspbian OS distribution comes with SSH server started you can connect PC with Raspberry on LAN and use console. Graphical programs automatically open windows on host PC screen, but we remind you the limits of Raspberry hardware. So, heavy IDE like codeblocks needs time to initialize itself, and it should appear do nothing even for a minute. But after start-up phase you can use this environment without problem (but no so fast as on PC!).

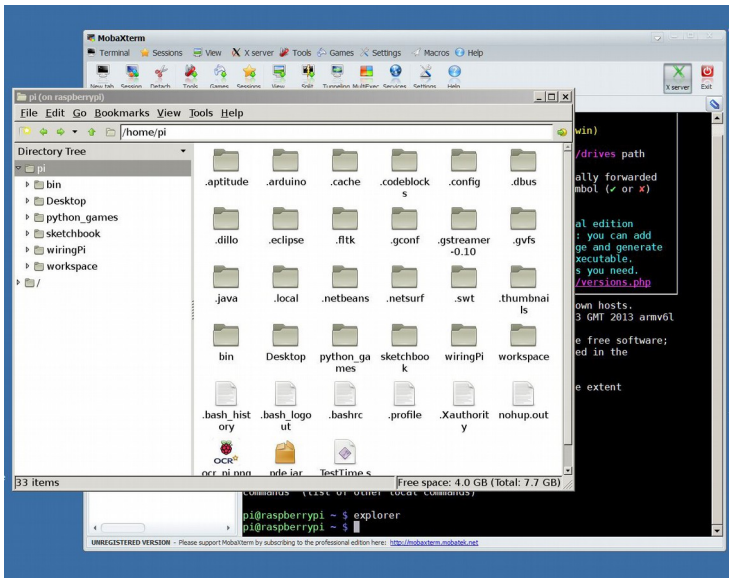


Fig. 7- Example of graphical console on PC

Arduino library for sending commands

The RAComm library is installed on remote Arduino IDE and on local Arduino IDE. With this library you can insert Raspberry dialogue into sketches.

Library functions (see Help.html for more details) :

- begin : it starts the listen program on Raspberry by the D4->GPIO22 pin
- isReady : just to verify if program is ready to accept input
- sendCommand : it sends string to Raspberry to be executed
- getAnswer : it reads a record of reply from Raspberry. When return NULL it means no more records
- close : it ends dialogue and Raspberry listen command release serial port
- openFile : it open a file on Raspberry to write or read (just one at time)
- readRec : it reads a record (new line terminated) from opened file. If it returns NULL it means no more records.
- writeRec : it writes a record to opened file
- closeFile : close opened file
- openConsole : open a simulated Arduino console (note that serial port is now used for commands protocol) . This console is achieved by X terminal Linux utility for now.
- readConsole : read record from simulated console. It is no blocking. So if no data, it returns NULL
- writeConsole : write a string to the simulated console
- setEcho : it opens a simulated console and echoes everything for debugging

The ExecSCmd program can have running arguments and help (-h). These arguments can activate echo on output for debugging. See help for details.

RandA programming

IDEs for several languages and environments are provided for programming directly on Raspberry: Arduino IDE, codeblocks for C and IDE for Java. Python is already included in Raspbian distribution.

Even if you can use this graphical IDE on Raspberry, we suggest you to use Raspberry just for C and Python programming. For Arduino programming, is more useful to use remote IDE on PC. Java is a portable language, so, it is more convenient to program on PC using heavy but powerful IDE like Eclipse and download jar file or war file later.

Codeblocks C IDE is a good compromise between power and weight. Be patient at starting phase because, further on, its use is not so slow and is really powerful.

WEB server

A Web Server is included in RandA software. If you don't like it, you can deactivate its automatic start putting a comment character (#) on concerning line in /etc/rc.local script. You can also delete the entire directory /home/apache-tomcat-7.0.47.

The server is started and stopped by the scripts : startWebS.sh and stopWebS.sh.

Because server uses Java, it must know where to find the Java home directory on Raspberry. This directory name can change in Raspbian OS distributions.

So, if server doesn't start, you should have to change the JAVA_HOME environment variable in startWebS.sh and stopWebS.sh.

Actually, this Web Server is an application server and it is Tomcat based (using standard port 80). It contains a menu of basic RandA management "Servlets" and a simple web based Raspberry console.

The aim of this server is to allow a WAN basic management of RandA, besides local network .

The menu is configured for protected access, but it is not enabled for default. To enable the access protection you have to edit file:

"/home/apache-tomcat-7.0.47/webapps/RandA/WEB-INF/web.xml"

and uncomment the protection section.

User name and password are defined in file:

"/home/apache-tomcat-7.0.47/conf/tomcat-users.xml"

For now, one role "randa" is defined and it has username "randa" and password "randa".

Session timeout is 30 minutes for default. But you can change it by editing the file:

"/home/apache-tomcat-7.0.47/conf/web.xml" (section <session-config>)

Basic static pages (like index.html) are located on directory:

"/home/apache-tomcat-7.0.47/webapps/ROOT"

where you can simply add your static page or modify the present pages.

Directory "…/webapps/RandA" holds the menu applications and …/webapps/sshwebproxy holds the console application.

You can add your Java application using Tomcat manager. Tomcat manager is at the address

"http://.../manager" (where … is for RandA IP address)

using username "tomcat" and password "tomcat". Tomcat manager let you upload war file and install it in very simple way.

But if you don't like to develop complex Java web applications, we have configured Tomcat server to manage also CGI scripts.

The configuration file

"/home/apache-tomcat-7.0.47/conf/web.xml"

contains the section "<servlet><servlet-name> cgi</servlet-name>..." that enable and set cgi management. In this case, "/bin/bash" is defined as script interpreter program. But you can change it with another interpreter like Python, for instance.

CGI scripts have to be put on the directory

"/home/apache-tomcat-7.0.47/webapps/ROOT/WEB-INF/cgi"

and referenced by the address

"http://...../cgi-bin/*script-name*"

Two examples of bash scripts are provided: testcgi.sh and testcgi2.sh

A index page is provided on server. This page has two links : to RandA menu or to WEBconsole.

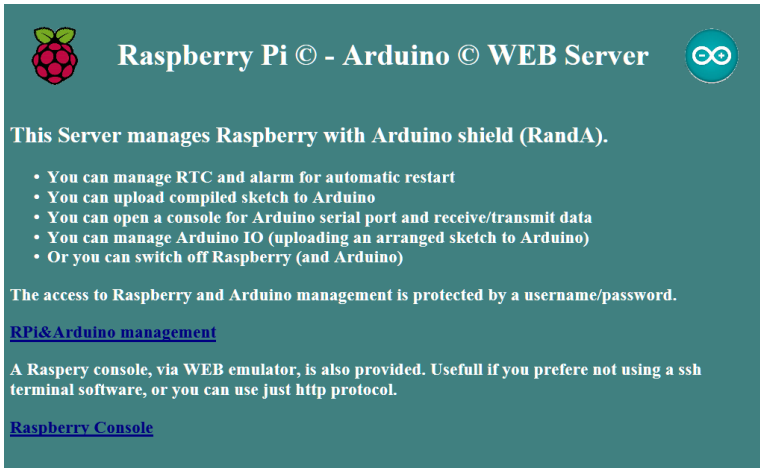


Fig. 8-index.html

RandA WEB menu

RandA web applications:

- Compiled sketches upload to Arduino. This application can choose between the two directories assigned to be repository of compiled sketches and let you select a sketch and upload it. Application, also, show or let you edit description of sketch.
- RTC setting.
- Restart (RTC alarm) setting.
- Arduino console. This application let you communicate with Arduino serial port (if any sketch that support it is uploaded)
- Arduino I/O management. This application let you to manage input/output pin if a preconfigured sketch “SerialRaspIO.hex” was previously uploaded.
- Switch off system. Shutdown of Raspberry with power switch off.

WEB Raspberry console

This application (author: Eric Daugherty) let you have a very basic WAN access to Raspberry OS. You can run Linux command or script and upload or download files. Actually, this application aim is to give you an access to Raspberry system even if you are not in local net and you can't use a SSH console program as MobaXterm.

This application is protected by username “randa” and password “randa”. But because is a Linux console simulation, this application ask you also a Raspberry username (def. “pi”) and password (def. “raspberry”).

Note that command sending has to be activated by button (not just by <return> key).

Programming and use

Depending on your expertise and preference, you can use RandA as a powerful Arduino or as a Raspberry with an intelligent I/O device.

RandA communication

Raspberry is the middleman. So, every communication with Arduino is dispatched by Raspberry, using interfacing software (remote IDE, uploading command or customer sketches)

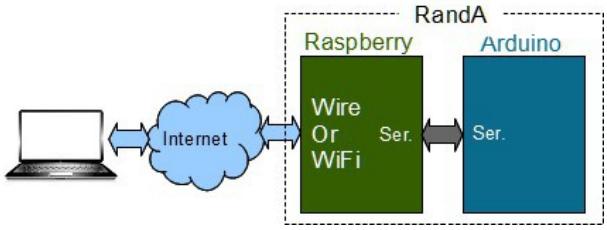


Fig. 9 - Communication

Arduino point of view

Use remote IDE and command library.

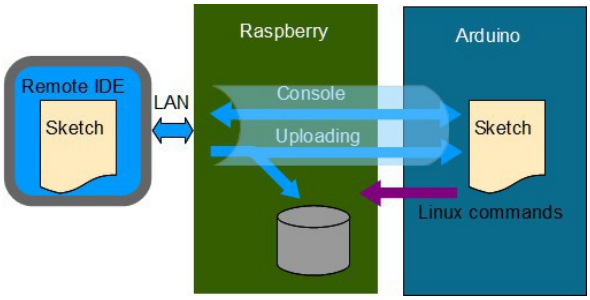


Fig. 10 - Arduino point of view

Raspberry point of view

Use Linux SSH console (X window, if possible). You can use local IDE or remote IDE again.

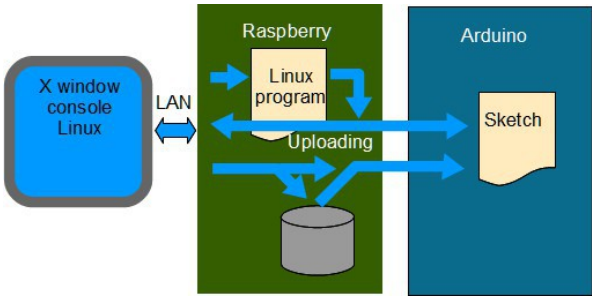


Fig. 11 - Raspberry point of view

Examples

Sketches in RAComm library

- BasicRACommExample : at every reset, starts dialogue with Raspberry and sends command to get time-stamp.
- ReadWeb : uses command “curl” to ask weather site for some weather values.
- SaveData : reads analogical and digital values every hour and writes it on a log file.
- SendMail : uses RTC to wakeup one time at day, reads analogical value, writes it on a log file and sends an e-mail with the last 5 records of the log file.
- ExampleSwitchOff : this sketch needs Arduino always powered. It checks if status is changed in respect to a analogical value (if < threshold). In this case it switches Raspberry on (if finds it off) and write event on a log file. A the end, it switches Raspberry off again if it was off.

The sub-directory “examples-notRAComm” contains sketches that don't use RAComm. Therefore you can use IDE console or Web console page to interact with sketch.

- TestSerial : You can change blinking time, sending value by serial communication.
- SerialRaspIO : Web page Arduino I/O management uses this sketch.

Sketches in /home/pi/bin/sketch4cmd

These sketches are used by Linux command “ArduIO” and “ArduInt”

- SerialRasp : is like SerialRaspIO except for the itsMe() function.
- SerialStop : stalls until it verifies a condition on digital or analogical value. (i.e. it doesn't reply by serial port until a condition)

Script examples

These very short scripts use Linux RandA commands to manage Arduino I/O.

- ScriptExamples4IO.sh : bash script that lights Arduino led if analogical value < threshold.
- PyExample4IO.py : same function but with Python script.

Two script (bash) are also in /home/apache-tomcat-7.0.47/webapps/ROOT/WEB-INF/cgi as CGI script examples (see Web Server paragraph).

Programs

Every command in /home/pi/bin (or in /etc) has its source (with comment) in /home/workspace/cworkspace.

Java programs in /home/bin/JavaCommands have sources in /home/workspace/jworkspace.