Raspberry Pi 32

Bachelor semester project

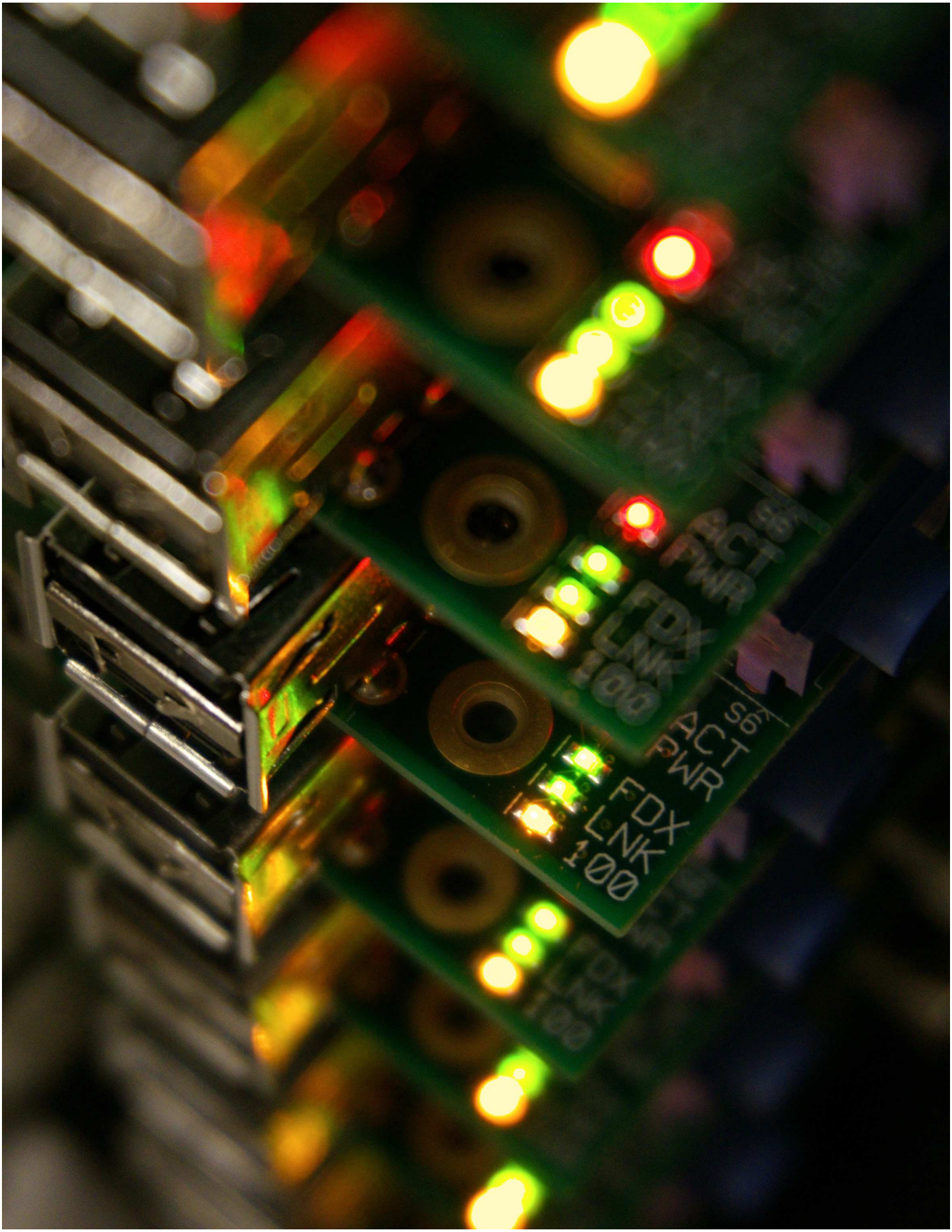
Sidney Bovet

Under the supervision of
René Beuchat
André Schiper
Thomas Ropars

Laboratoire d'Architecture des Processeurs
IN-F Ecublens
CH-1015 Lausanne
http://lsrwww.epfl.ch/

Laboratoire de Systèmes Répartis
IN-F Ecublens
CH-1015 Lausanne
http://lsrwww.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE


ACT
PWR
FDX
LNK
100
S6

Raspberry Pi 32

Bovet Sidney
IN

Introduction

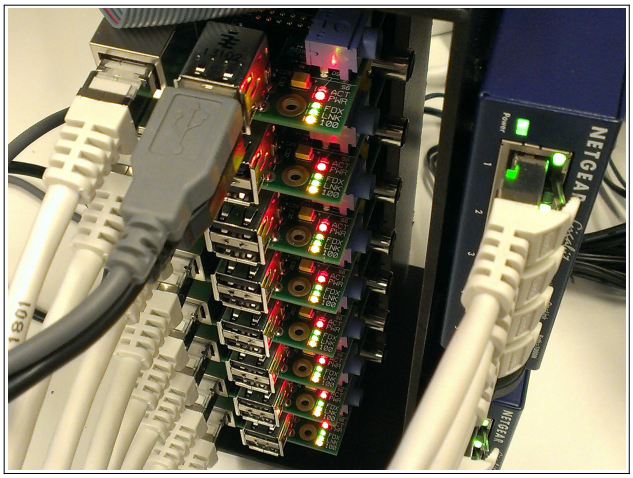
*The idea behind Raspberry Pi 32 is to build a **demonstrative** device of distributed calculus, storage system and other clustering methods and evaluate its performances. This device is made of **Raspberry Pi**, a small computer whose release price was **US$ 35** in 2012.*

Goals

*- Build an **education-oriented** cluster of 32 Raspberry Pi*
*- Study and provide a hardware-based **observation system***
*- Provide a power consumption-**analyzing tool***
*- Set up an implementation of the **message passing interface***

Summary

*Ecological concerns are more and more present and computational resources and their **power consumption** are one of these. It is therefor very useful to be able to quantify how much power various algorithms require – for instance in an embedded system.*



***Education** also is one big part of this process and, as a student, you would be very happy to put the theory (on cluster computing or highly distributed storage systems, for instance) into practice with a real cluster. It often is of great help to deal with more **concrete** things than concepts and simulations.*

Results

*For all of these reasons, Raspberry Pi 32 is a project that aimed to build a **self-evaluating cluster**, capable of measuring how much current each of its nodes is using in a "real time fashion" (that is displaying and keeping track of these data). Besides of that, it is also capable of **notifying through its hardware** interface what it is doing for further investigation. And of course it is capable of heavy **supercomputer calculations**, just as its bigger brothers.*

*It is designed in a way such that it has four independent **modules** of eight nodes, in order for students to work separately in small groups on a single module. And if the experiment requires more nodes or more computational power, they can easily be bound such that the cluster counts 16 or 32 nodes.*

Future considerations

*Building the system is one thing, but it is not enough. It has to be **evaluated** and comparisons have to be made with already-existing systems; it would be foolish to establish relationship with other quantitative metrics without any reference. Remember that the Raspberry Pi was not designed for the use it is made of in this project, and it might use a lot more current than a dedicated device such as computational units of the Human Brain Project.*

A list of further enhancements can be found further in this report.

Table of contents

1 – Introduction

This report is organized in ten sections (including this one) and provides a documented full overview of the work done on the project RaspberryPi32, a **low-cost** supercomputer. After having acknowledged some of the major studies that have been carried out on the same subject, the various components used to build the cluster are described. I then present two of my contributions to this project, namely hardware-based system studying and power consumption self-evaluation.

After that, the process used to validate the system is exhibited. One of the key step in the system validation was to make possible all sort of distributed calculus. We installed MPICH and managed to run a simple distributed application computing an approximation of π.

A quick user guide is also provided, explaining for instance the network map I set up. It also includes two complete examples of use. Then two problems we encountered are set out and, finally, a list of possible enhancements to the project is established.

The footnotes containing hyperlinks in this document are reiterated at the end of it. When the footnote heads to a document, it is also available on the CD-ROM joined to this report.

2 – Related work

This section acknowledges the major studies and works that have been made on the subject we are discussing in this report.

The first work, and probably the most important one for this project, is “Iridis-pi: a low-cost, compact demonstration cluster”. It was published in September 2012 by Simon J. Cox et al. [1]. This paper is a **must-read** for anyone who wants to invest some time in studying Raspberry Pi-based supercomputing.

Also, as pointed out by them, the industrial world begins to invest more and more effort into producing energy-efficient hardware. As an example, Calxeda's ECX-2000 Server-on-Chip family[1] is based upon **ARM** processors.

Other researches have been made on the subject, noticeable one are Andrew K. Dennis' “Raspberry Pi Super Cluster” [2] and Joshua Kiepert's “Creating a Raspberry Pi-Based Beowulf Cluster” [4].

3 – The hardware

Here stand a description of what the cluster is made of. Its structure consists of **Raspberry Pi** powered by a home-made printed circuit and interconnected with Ethernet switches.

Fig. 1: The cluster is made of Raspberry Pi.

1 http://www.calxeda.com/ecx-2000-family/

A useful **schematic representation** of the overall structure of the cluster is available in the annex.

The Raspberry Pi is a **credit card-sized** computer intended for educational use. It is developed by the Raspberry Pi Foundation and was released on 29th February. It is declined in two versions, A and B. The main differences between the two models are as follows:

Model	A	B
Power	2.5 W	3.5 W
Memory	256 MB	512 MB
Price	US$ 25	US$ 35

The model we decided to use for this project is the B because of the low difference in price and, among others, its bigger memory.

Hence the supercomputer that this project works on is made of 32 B-Type Raspberry Pi featuring a Broadcom BCM2835 chip which contains a 700MHz ARM11 CPU, 512MB RAM and a Broadcom VideoCore IV GPU. It exposes two USB 2.0 and one 10/100 Ethernet ports (note that the Ethernet port is wired to an Ethernet-to-USB 2.0 adapter). It also provides a 26-pin header including a UART and eight GPIO pins, as well as I2C and SPI busses.

Along with these boards, René Beuchat designed a **backplane** printed circuit board in order to power the boards. It is 6-to-24V powered (however, the recommended voltage is 10V) and features a DC/DC converter to provide the 5V micro USB and a 16-bit AD7689 analogical to digital converter to evaluate the **power consumption** of each board. See the annex for the design sheets and the *Power consumption* section for more details about the power consumption evaluation process.

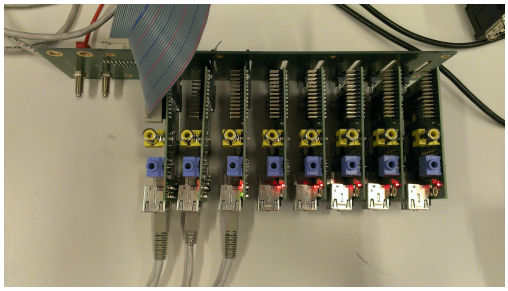


Fig. 2: One eight-board module, nude.

These backplanes home **eight** Raspberry Pi each, so that it is possible to have four smaller devices for a lab session. Figure 2 presents one of these modules.

Each module has two 5-port Netgear Ethernet switches (one of which supports Gigabit) and, for these modules to be really independent, a **structure** was created such that these switches are fixed to the eight Raspberry they are connected to. Figure 3 shows a fully mounted and operational module. The two Ethernet switches are connected with each other using an uplink cat. 6 wire.

When willing to work with the whole cluster, all that's needed is to connect each of the routers to a headend full Gigabit switch, as illustrated in the **schematic representation** of the cluster in the annex.

Fig. 3: A fully mounted module.

4 – Pin toggling

This section presents the solution we developed to allows the user to **monitor** the cluster's state. This can be achieved in two ways, namely through **software** or **hardware**. These two options are discussed and the one we set up is presented: pin toggling. I then compare Java and C in the specific task of changing the logical state of a pin and finish with an example of how this can be used.

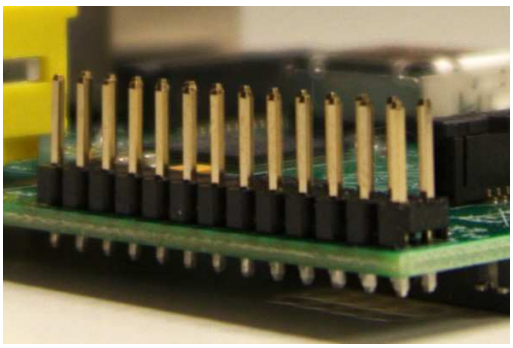


Fig. 4: The 26-pin connector of the Raspberry Pi.

The software-based method has the advantage of being easy to set up and provide a wide range of data, such as the destination of a specific packet. On the other hand, the second one is much faster. And since this is what is needed when studying such a system, the method I focused on was to toggle a pin on the board.

It is then possible to connect a **logic analyzer** such as a USBee[2] device or an oscilloscope to the board(s) in order to monitor the activity of the nodes. One drawback of this method is that one has to include additional function calls to the code to be analyzed, and properly interpret the result.

4.1 – Java or C?

After having decided to perform such pin toggling operation, I decided to perform a small experiment: comparing Java and C. The methodology was quite simple: toggling a pin as fast as possible in both languages and measure the time needed to perform one high/low cycle (i.e. the signal **period**).

The code I used in the Java version relies on a library called Pi4J (available in the annex) and allows easy access to the low-level peripherals of the Raspberry Pi, such as the GPIO pins we are interested into. For the C code I followed a very good article by Pieter-Jan Van de Maele [5] showing how to directly access these pins. The following table shows the two codes involved into pin toggling.

Java	C

`final GpioController gpio =`
`  GpioFactory.getInstance();`
 	
`GpioPinDigitalOutput myPin =`
`  gpio.provisionDigitalOutputPin(`
`  RaspiPin.GPIO_04);`
 	
`for(int i = 0; i<100000; i++) {`
`  myPin.toggle();`
`}`
	`#include "RPI.h"`
	`[...]`
	`if(map_peripheral(&gpio) == -1)`
	`  return -1;`
	`INP_GPIO(4);`
	`OUT_GPIO(4);`
	`[...]`
	`for(i=0; i<1000; ++i)`
	`{`
	`  GPIO_SET = 1 << 4;`
	`  GPIO_CLR = 1 << 4;`
	`}`

2 http://www.usbee.com/rx.html

After having measured the periods, the results are irrevocable; while Java has an average period of 208.77 µs, C performs the same task in 80 ns.

I thus went on using C but of course if you want to analyze a Java code for instance, you will have to place Java calls instead of C code into your program (unless you use a suitable wrapper).

4.2 – Example

To illustrate the use of such pin toggling, I have built a **logical ring**: a set of interconnected devices pass around a token such that only one device at a time possesses the token (this can be used for instance to avoid collisions in network communications). This is based on TCP socket communication.

In the function that receives the token and sends it back I added two function calls so that a specific pin on the Raspberry Pi is high when the devices **holds the token**. The code needed in order to perform this task is in the annex and was provided and explained by Pieter-Jan Van de Maele [5]. If any miscomprehension appears when reading the code, I strongly recommend you take a look at his very good post.

The loop of the code handling the token is very simple and is as follows:

```
while (strncmp(buffer, TOKEN, BUFFER_SIZE)==0)
{
        recv(*socket_in, buffer, BUFFER_SIZE*sizeof(char), 0);
        GPIO_SET = 1 << 4; // set the pin high since we have the token
        send(*socket_out, buffer, BUFFER_SIZE*sizeof(char), 0);
        GPIO_CLR = 1 << 4; // set the pin low since we just sent it back
}
```

By using a logic analyzer, it was made possible to know exactly in what **state** every device was while the token was passed around, as shown on Fig. 5.

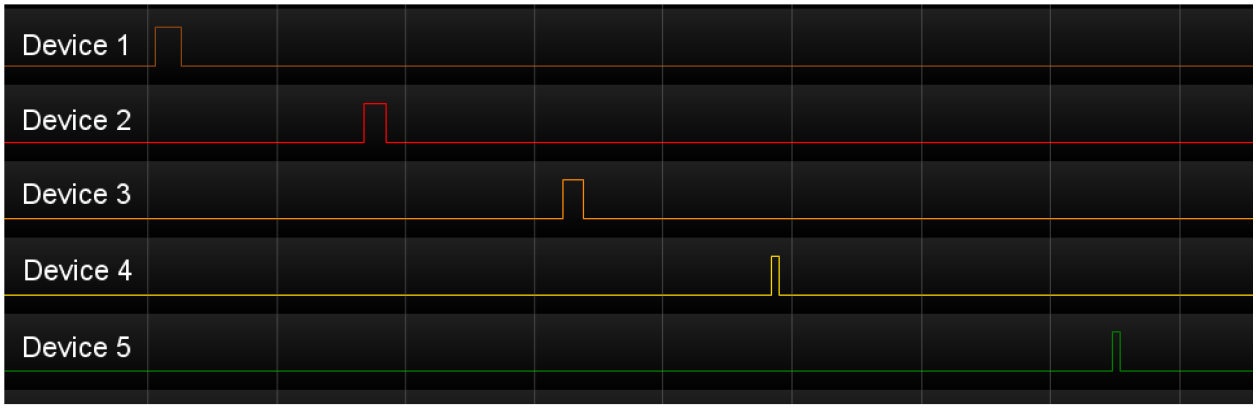


Fig. 5: The hardware state of a pin on five nodes while the token passes by. 400µs per division

The point of this report is not to discuss these results but as a proof of the **efficiency** of this method, here are some interesting metrics:

The average high/low time on a single node is 68.5 µs, the time between the falling edge of a device and the rising edge of the following is 738.2 µs and the total RTT with five nodes is 39.8 ms.

We can easily see how **useful** such data can be if you are trying to study the impact of a certain operation or network configuration on the transmission time of a packet.

5 – Power consumption

This section presents how was set up the application used by the cluster to evaluate its power consumption.

As stated before, the backplanes housing the Raspberry Pi feature an AD7689 analogical to digital (AD) converter[3] providing current consumption measurement on eight channels separately. Since the converter works on voltage, each device has its own current-to-voltage converter. It is possible to communicate with the AD converter using the **SPI** protocol. On figure 2 you can notice that the first device is connected with a flat cable to the backplane, further linked to the AD converter. The C code involved into communicating with the device can be found in the annex. It relies on a library created by Gordon Henderson [3] and his whole site is a very good helper for understanding how to use Wiring Pi on the Raspberry Pi[4].

Being able to talk with the converter is one thing, but it is better to speak the same language. For the converter to deliver its data we have to send a **command word** to it. When reading the code achieving the data polling, keep an eye on page 25 of the data sheet *AD7682_7689.pdf* (in the annex), which tells what the various bits correspond to.

The last step in retrieving values is to **interpret** them: the converter is giving an unsigned value on 16 bits where 0 and 65535 correspond respectively to the ground and its internal reference specified in the command word (4.096V in our case). The problem is that there exists a small offset so in order to be precise I disconnected one Raspberry Pi to get the value corresponding to **zero consumption**.

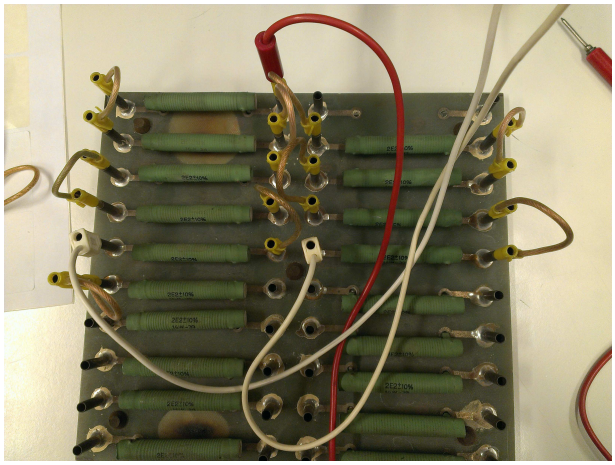


Fig. 6: The resistor used.

It then was interesting to record the output of **1A** of consumption. We can find the resistive load needed using Ohm's law:

$$R = \frac{U}{I} = \frac{5\text{V}}{1\text{A}} = 5\Omega$$

In order to do that in practice (and since the resistors used were 2.2Ω ±10%), I combined them in series and parallel along with a multimeter to obtain the desired resistance.

On the backplanes there is a testing point TP2 which allows us to measure the voltage sent into the devices through the micro USB connector (see in the annex the 5th page of the backplane design sheets or the simplified block design – also on Fig. 13 – to figure out precisely how and where).

3 Data Sheet available in the annex as well as on http://www.analog.com/static/imported-files/data_sheets/AD7682_7689.pdf (January 2014)

4 Especially this article for SPI: https://projects.drogon.net/understanding-spi-on-the-raspberry-pi/

Connecting it through a 5Ω-resistor to the ground allows us to simulate a 1A-consuming device. Fig. 6 shows the resistive load used.

The values I extracted from this experiment are 863 for the zero offset and 7744 for the 1A-consumption; these are also set as constants into the C code of the sequenced polling program. Since these values correspond to current, the last step vas to convert them into **power** units (Watt) by multiplying them by the voltage measured at the same testing point, namely 5.06V.

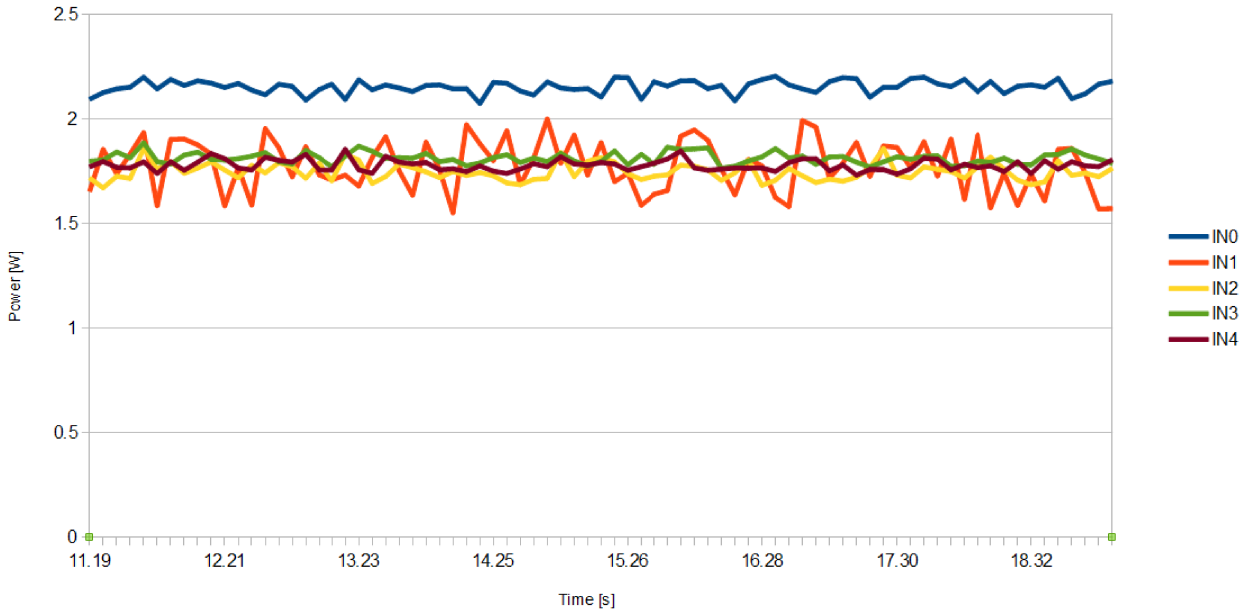


Fig. 7: The power consumption over time of the polling device and four other idle devices.

Regarding the data given by the converter – such as the ones in Fig. 7 – care must be taken when analyzing them:

1. Some values are “noisy”: the orange trace has an average value and standard deviation of 1.76 and 0.13 respectively.
2. The device polling the converter is really busy doing so (the blue trace is the poller, other are idle devices)[5]

Note that, regarding the first point, the “noisiness” of the data is due to two major things. First of all, the ARM11 processor has two **sleeping** modes, namely *standby* and *dormant*. In the first one most of the clocks of the device are disabled while the device itself is still powered up. The second one just keeps the cache and other memories up and the device is shut down. Since it is hard to tell in which mode the CPU enters when awaiting a TCP connection it also is hard to tell if it doesn't perform other small operation, causing the ~0.1W deviation.

However, if the processor was the only source of noisiness we would observe the same noise on all devices and it is not the case. The noisiest trace (the orange one on figure 7) has, as stated before, a standard deviation of 0.13W where the least noisy one only

5 Its higher power consumption can also be explained by the fact that it is connected *via* HDMI to a display.

exhibits 0.02W of standard deviation, for the same average value of 1.77W. René Beuchat left room on the backplanes for **capacitors** to be placed in order to absorb some of this noisiness but further investigations have to be made on the exact cause of it.

I now explain how the program was developed. A more synthetic overview of its functionality can be found in the *User guide* section. It is located in `~/power_monitoring/` and is called `exec_adSequencePolling`.

The first functionality of this program is to display the values in the console (launched with the '`--console`' argument or without anything). The values are refreshed every 0.5s.

One more supported option is '`--java_graph`': it prints in *stdout* the values in a serial manner that is not intended for human use but rather to be read from a Java process. Also in the annex, the Java code launches the polling program in a new process with the `--javaGraph` option and the reads the output in order to display a live-chart of the power consumption using JChart2D[6], a minimal real-time chart-displaying library. However, the Raspberry Pi is not powerful enough to run this program smoothly and it appears that the charting is impractical for lab use.

I then added support for a '`--file FILE`' argument which will then print the output in comma-separated values (CSV) format. This format can for instance be further imported in LibreOffice Calc in order to analyze the data **a posteriori**. The charts shown in Fig. 7 and Fig. 8 come from such a CSV file. Note that in the file, timestamps are missing, but the polling appears to have a sampling rate of 9.81Hz hence it is safe to consider each sample being 101.75ms distant from its neighbors.

6 – System validation

Two elements I used to show that the system is indeed working are now presented. The first one was to use the power consumption mechanism and the second one to make distributed computations possible, which was a crucial element.

6 http://jchart2d.sourceforge.net/

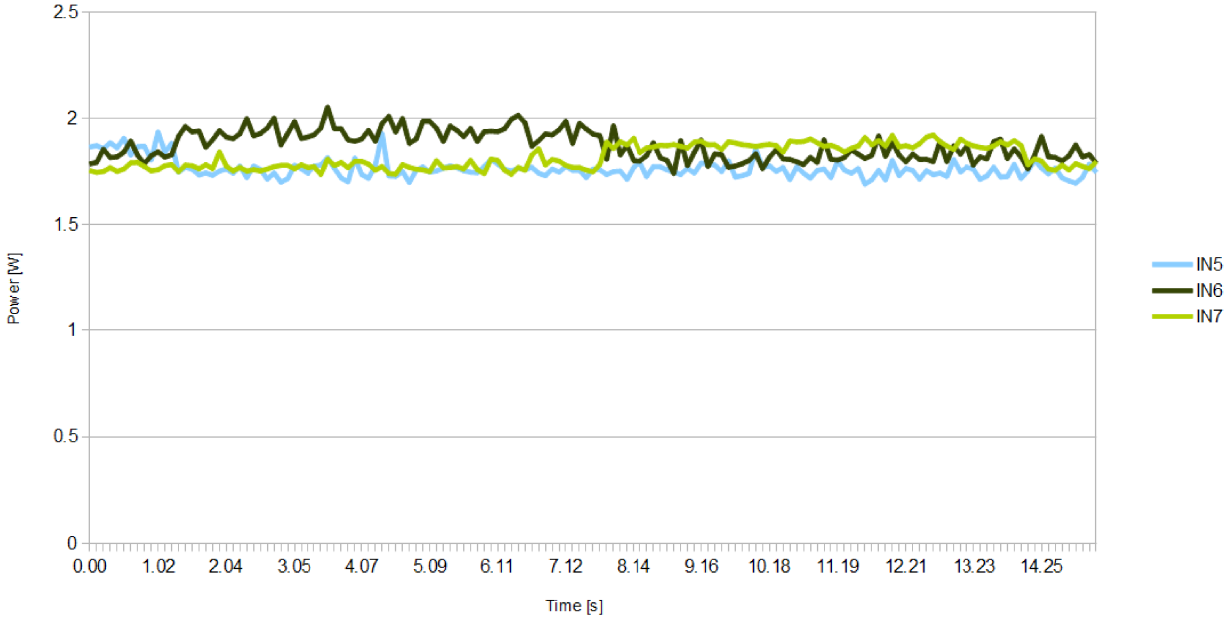


Fig. 8: Trace of the token going from one device to another.

6.1 – Power consumption evaluation

In order to validate the system I first extended the logical ring mentioned in the *Pin toggling* section such that a device receiving the token first has to compute some hard but useless value, just so that it is **busy** for a short amount of time (~6.3 s). I then retrieved the values of the power consumption, which are displayed in Fig. 10.

The first thing we can notice is that it is **working**; the computation is detected and we can see that the token first was in IN5, then passed to IN6 and after a while to IN7. However, the data are a little bit noisy compared for instance to the trace of the token captured by the CPU-usage plot, as shown in Fig. 9. Note regarding figure 9 that it is unlabeled but represent the usage, from 0 to 100%, of the CPU from IN0 and the duration of the peak is exactly the same as on fig. 10.

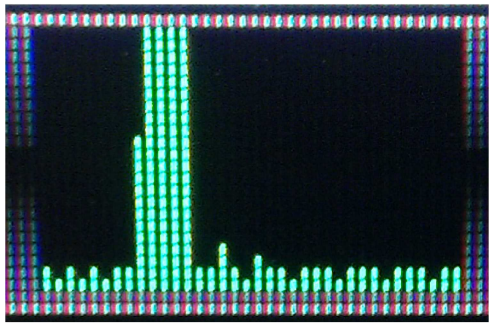


Fig. 9: CPU usage during the flyby of the token.

Also, the device acquiring the power consumption is so busy polling the converter that we cannot even see the token pass through it. On Fig. 7 we should in fact see the blue trace (the polling device) **higher** at the right of the chart than it is at the left, since it received the token in the time covered by this plot and still has it at the end of it. I thus strongly recommend the polling device to be **external** to any experiments made with RaspberryPi32. With this aim in mind I have set up four different SD cards labelled “SPI” and with an IP address of the form `192.168.#module.10`.

Despite the negative sides, the experiment is a success since we were able to tell precisely what impact a certain scheme of computation had on the power consumption of the whole module.

6.2 – Distributed calculus

Since the purpose of this project is to build a supercomputer an important part of the system validation was to be able to compute something in a distributed manner.

We considered using LSR's domain-specific language ***Distal***, which is based on Scala. A few observation led us however to dismiss this possibility. First of all, this language is still in development, making it hard to use for a Bachelor project. Also, since it is based on Scala, it requires a lot of RAM and time to compile (with respect to the performances of the Raspberry Pi), making it hard to use in a lab session. Note however that, once compiled and started, Scala programs run well so it is not impossible to run it on the Raspberry Pi.

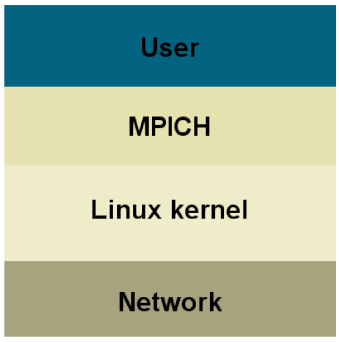


Fig. 10: Position of MPICH in a standard architecture.

This is why we turned to a more classical, yet highly efficient alternative: a message passing interface (**MPI**). Simon J. Cox et al. [1] provided a very good tutorial on installing an implementation of MPI, MPICH, on the Raspberry Pi[7]. This procedure mainly consisted of retrieving the latest sources of MPICH, configuring and building them and finally spreading the built program on all the devices.

A famous example of MPI application is **cpi**, a C program computing pi using numerical integration. The relation used here is:

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

Since we can easily **divide** this integral into tiny parts and compute them separately using the Riemann Integral principle[8], this algorithm performs well in a distributed scheme of computation.

The outcome of this manipulation is that the cluster calculated *pi*, demonstrating that it has the abilities of a real supercomputer.

7 – User guide

This section provide two full examples of typical use of RaspberryPi32 and a small set of instructions on how to use the cluster and its various functionalities.

An interesting file to look at is the schematic representation of the cluster in the annex (*Schematic_Representation.pdf*).

7 http://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton.htm
8 http://en.wikipedia.org/wiki/Riemann_integral

7.1 – Two complete examples

The first example shows how to compile and run a small MPI program.

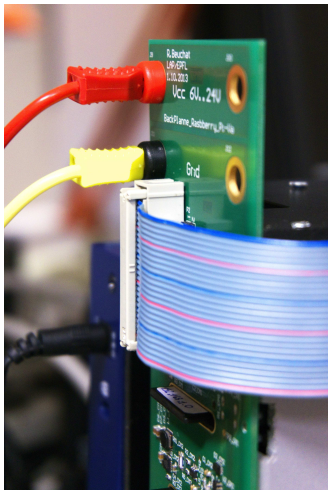


Fig. 11: Electrical connections.

The cluster is powered through the backplanes with a laboratory power supply. It is designed to support voltage between 6V and 24V but it is recommended not to exceed **10V**. The backplanes were first designed so that the modules could be connected in series but a small weakness in the solder implies that it is unsafe to do so. It is therefore recommended to wire the modules in a **star-shaped** manner.

To start using the cluster, first make sure that each Raspberry Pi has an **SD card** plugged in, that the first node is linked to a **keyboard** and a **display** and that the network part (Ethernet wires, switches, etc.) is correctly wired and powered up. Please refer to the cluster's schematic representation available in the annex to see how to link the different parts of the network together.

Once this is set up, the cluster can be powered up (remember that the recommended voltage is **10V**). After a short while the screen asks for a username and a password. The credentials are the default ones on Raspbian (the Linux version that is installed on the Raspberry Pi, a customized version of Debian), namely *pi* and *raspberry* as username and password respectively.

Once logged in, run *`cd mpich_testing`* and then *`nano my_hello_mpi.c`* to launch the text editor. Type in the following code:

```
#include "mpi.h"
#include <stdio.h>

int main(argc, argv)
int argc;
char *argv[];
{
      int rank, size;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      MPI_Comm_size(MPI_COMM_WORLD, &size);
      printf("I am %d of %d\n", rank, size);
      MPI_Finalize();
      return 0;
}
```

Save and exit (Ctrl+X then Y and Enter). Then run

`$ mpicc my_hello_mpi.c -o my_hello_mpi`

to compile, and

`$ sudo cp my_hello_mpi ~/mpich_build/examples`

in order to move the file in the working directory of MPICH.

To finally launch the code on the local machine with two processes, run the following command:

```
$ mpiexec -n 2 ~/mpich_build/examples/my_hello_mpi
> I am 0 of 2
> I am 1 of 2
```

Great! In order to run the code on another machine, we have to transfer it to the other node:

```
$ scp my_hello_mpi pi@node-1-1:~/# or whatever machine you want
                                   to work with
$ ssh pi@node-1-1                # you are now in the other node
$ sudo mv my_hello_mpi mpich_build/examples
$ exit
$ cat > node0and1
 192.168.1.0
 192.168.1.1
 ^C                              # this is Ctrl+C
$ mpiexec -f node0and1 -n 2 ~/mpich_build/examples/my_hello_mpi
> I am 0 of 2
> I am 1 of 2
```

Congratulations, you (may) have just launched your first **distributed** program! To convince yourself, you can launch it once again and look at the LEDs of the network switch; the devices are indeed communicating.

The second example shows how to run the **logical ring** (the version presented in section 6) on three nodes, along with power consumption monitoring.

After having successfully logged in as described in the previous example, execute the following commands:

```
1    $ ./logical_ring/exec_node_with_calculation no 192.168.1.1&
2    $ ssh pi@node-1-1
3    $ ./logical_ring/exec_node_with_calculation no 192.168.1.2&
4    $ exit
5    $ ssh pi@node-1-2
6    $ ./logical_ring/exec_node_with_calculation yes 192.168.1.0
```

Let's see these commands line by line: the first one launches the program on the first node and tells him to **wait** for being contacted (*"no"*) before contacting *192.168.1.1*. Then commands 2 to 4 do the same with the second node and log out of it. Command 5 and 6 do the same with the last node but we tell it it's the **last** one (*"yes"*) and who was the first node.

To monitor the power consumption of your three devices, hit Ctrl+Alt+F1 to open a new terminal. Then log in as previously, type `cd ~/power_monitoring` and execute `./exec_adSequencePolling`. You now can see where your token is!

Note that you could have launched the ring without the first node (for instance between nodes 1, 2 and 3) to let it poll the device. See the *Power consumption* section for more details about the impact of polling on the power consumption and a graphical representation of what you can observe in the console now.

7.2 – Network map

The cluster is **statically** addressed and the IP of each node is written on the SD card. The address scheme is `192.168.#module.#node`[9] (i.e. the address of the second node of the fourth module is `192.168.4.1`). The nodes with IP `192.168.x.0` are the 'masters' of their module: all the other nodes in it have its public RSA key and it can therefore log in and execute things on the other nodes using the `$ ssh 192.168.m.n` command without having to type in a **password**. Moreover the *first* node (`192.168.1.0`) can log in every other device.

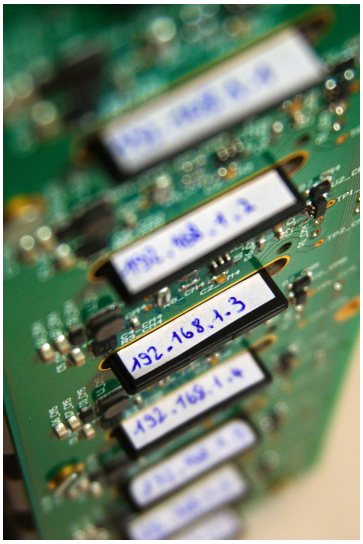


Fig. 12: The static IPs are labelled on the SD cards

This can become very handy when used with a **script**. A small example is a script that safely powers off the whole first module:

```
#!/bin/sh
for I in 1 2 3 4 5 6 7; do
      ssh pi@192.168.1.$i “sudo poweroff”
done
sudo poweroff
```

Note that on the first node, **aliases** were set in `/etc/hosts` such that the nodes are also known by the master as `node-m-n` (where m and n stand for module and node, respectively).

7.3 – Using power consumption displaying

Before using SPI you have to type in the command `$ gpio load spi` on a device on which wiringPi is installed. This command loads the SPI kernel modules.

The program used to start the power consumption display is `~/power_monitoring/exec_adSequencePolling`. As its name suggests, it works in a polling manner and it is therefore recommended for the polling device to be external to the experiment. See the *Power consumption* section for more details and proof of this.

This program accepts various parameters, the first one of which is `--help` and displays the possible usage. Here's a summary of the arguments:

`no argument`	Displays the power consumptions directly in the console, in a human-readable manner. Refreshes every 0.5 seconds.
`--console`	Same as without argument.
`--javaGraph`	Prints the values serially, separated by a comma (suits another Java charting program).

9 Note that, due to technical issues, the modules are one-indexed, while the nodes are zero-indexed.

`--file FILE` Writes the values into FILE, in a CSV format. Samples are taken at ~9.82Hz, i.e. every 101.75ms.

`--help` Displays the help.

If you wish to check the values read by the AD converter or re-calibrate it, fig. 13 can help you figure out how the backplane works and where the two testing points are located.

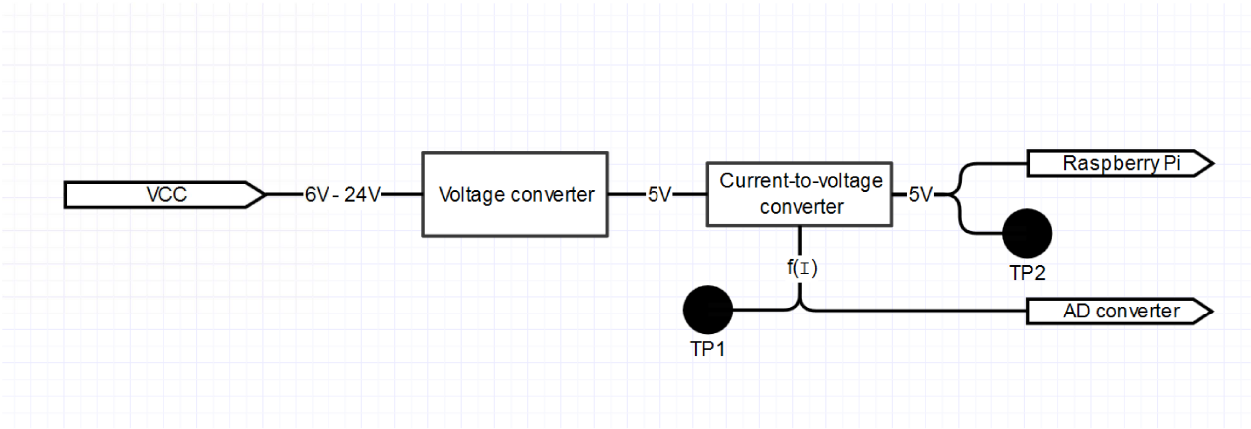


Fig. 13: A simplified block diagram of the power supply circuit of the backplane.

7.4 – Using the logical ring

There is a concrete example of how to use the logical ring at the beginning of this guide.

The program executing the logical ring is located at `/home/pi/logical_ring/` and exists in two versions: the first one toggles a GPIO pin and the second forces the device receiving the token to perform some dummy calculation to increase its power consumption.

The usage is as follows: `./exec_node_with_calculation [yes/no] NEXT_NODE_IP` where [yes/no] indicates whether the current node is the one **closing** the loop. Since I encountered synchronization issues, each node sleeps for a second when contacted by its predecessor, making the ring slow to start. However, once started it works without loss of time, as shown in section 6.

7.5 – Using MPICH

The version currently installed on the cluster is **MPICH 3.0.4** and its build folder is `/home/pi/mpich_build`. Commands such as `mpicc` and `mpiexec` are available. As an example, the command used to compute an approximation of π as described in the *Distributed calculus* section is the following:

```
$ mpiexec -f machinefile -n 2 ~/mpich_build/examples/cpi
Process 0 of 2 is on node-1-0
Process 1 of 2 is on node-1-1
pi is approximately 3.1415926544231318, Error is
0.0000000008333387
```

`machinefile` contains the IP address of the first and second nodes (or their aliases, as

mentioned in the section 7.2 of this guide):

```
node-1-0
node-1-1
```

or

```
192.168.1.0
192.168.1.1
```

See the MPICH documentation for further informations [6].

8 – Problems encountered

Here I establish a list of problems that we encountered during the process of building and setting up the cluster.

There has been one major problem encountered: a **short-circuit** on the backplanes. After investigation, it wasn't possible to clearly determine the cause of the issue, only its location on the board: under or within several DC/DC components.

The solution to this issue was to identify each short-circuiting converter and replace them. This is not yet fully achieved and, by the time of writing, all the experiments were made with the two only functional module, hence making it a 16-node supercomputer. However, each and every part of this project has been developed in a scalable way, such that doubling the number of nodes shouldn't harm its functionality in any way.

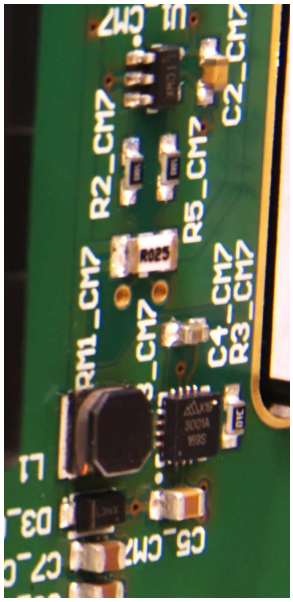


Fig. 14: Components on the backplane.

One other problem was related to the SD cards: these serve as storage unit for the Raspberry Pi and one easy and very convenient way to reflect changes made on, say the software configuration of the devices, is to use a tool such as Win32DiskImager[10] to make a **copy** of the state of the SD card. The problem was that after having set up IP addressing and locally built a suitable version of MPICH (see the *System validation* section for details about the use of MPICH) I found out that the card I was using had slightly more **sectors** to write on than all the others, even if they were of the same model. This implied spending another two hours re-compiling and setting up things on another smaller SD card.

Now that this has been properly done, it is unlikely that one would need to do this again since it is relatively easy to access each node via SSH. See the *User guide, Network map* section for detailed instructions on what and how.

10 http://sourceforge.net/projects/win32diskimager/

9 – Future work

In this section I establish a non-exhaustive list of tasks that can be achieved to improve this project.

9.1 – Performances comparisons

One of the most important things to do, now that the cluster is set up and ready to go, is to see what it is **up to**. It will be interesting to see how well it can do against various architectures, such as regular single-processor PC, GPU-based computation or even FPGA-based ones. And by saying "how well" I mean not only in terms of time performances (which I suppose to be really worse), but also power consumption or fault-tolerance. And those results have to be **scaled**, for instance in terms of *flops/watt* or *flops/US$*.

9.2 – Improve AD-converters polling

The backplanes were designed in such a way that it is possible for a single device to poll the **four** modules together (using SPI's addressing scheme). Since all the modules weren't available until very late in the semester I couldn't achieve this. Also it might be interesting to know exactly what is the **impact** on the power consumption of a device of it being connected to a display or not (as stated on footnote 5).

9.3 – Improve real-time charting

This task comes from the fact that the JChart2D library used, while aimed to be light-weight, could not help in rendering a **smooth** real-time chart. The problem may be solved by optimizing the current version of both the polling program and the charting one or using a more efficient charting program, maybe in C or C++.

9.4 – Ease access to inner pin socket

This is more of a workshop task: when trying to monitor pins of nodes inside the module I found out it was impractical for lab use; the pins are hard to access. I came with a quick solution but it is not elegant and quite inconvenient. So a good thing to improve RaspberryPi32's efficiency in lab sessions would be to provide easy access to all the pins of all the nodes, for instance, by drilling an opening and redirect the 26-pin socket on the side of the module.

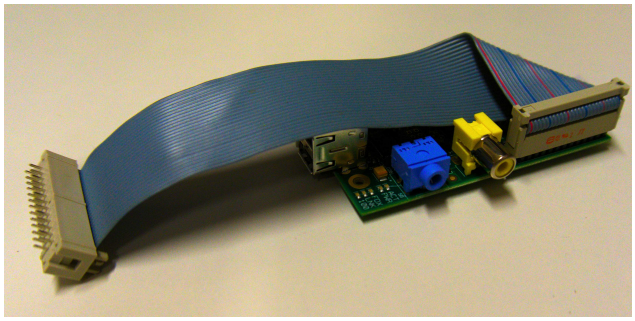


Fig. 15: The quick fix used to access the pins.

9.5 – Keep the User guide *up-to-date*

The title stands for itself; since the first goal of this project is to provide an **educational** device, it must have a proper and efficient guide.

10 – Conclusion

RaspberryPi32 is a long-term project, but the basis are set. With a supercomputer capable of being studied at both an activity and power consumption level, students and professors are well prepared to produce and understand the upcoming innovations in the field of distributed systems study. I do believe this project can be of great educational help and I am happy to have contributed to it.

To conclude this report I would like to thank everyone who was involved in kicking off this interesrting project: André Schiper for having heard my proposition, René Beuchat for the active hardware-related work and Thomas Ropars for the distributed side of the project, as well as André Guignard and André Badertscher for the technical support.

References

1. Cox, Simon J., in Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, Neil S. O'Brien, *Iridis-pi: a low-cost, compact demonstration cluster,* Cluster Computing, June 2013
DOI: 10.1007/s10586-013-0282-7
2. Dennis, Andrew K. *Raspberry Pi Super Cluster*, Pack Publishing, November 2013
ISBN: 1783286199
3. Henderson, Gordon. (n.d.). WiringPi. In *Wiring Pi GPIO Interface library for the Raspberry Pi*. Retrieved October 30, 2013, from http://wiringpi.com/.
4. Kiepert, Joshua. (May 2013). Creating a Raspberry Pi-Based Beowulf Cluster. In *Boise State University*. Retrieved November 19, 2013, from http://coen.boisestate.edu/ece/raspberry-pi/.
5. Van de Maele, Pieter-Jan. (May 2013). Low Level Programming of the Raspberry Pi in C. In *Creativity in Automation & More*. Retrieved October 8, 2013, from www.pieter-jan.com/node/15.
6. Argonne National Laboratory. (n.d.). Guides | MPICH. In *MPICH*. Retrieved January 1, 2014, from http://www.mpich.org/documentation/guides/.

Notable footnotes

1. Link to Calxeda's ECX-2000 ARM-based SoC family webpage.	http://www.calxeda.com/ecx-2000-family/
2. Link to CWAV's USBee RX product suite.	http://www.usbee.com/rx.html
3. Data sheet of the AD converter (see in annex)	http://www.analog.com/static/importedfiles/data_sheets/AD7682_7689.pdf*
4. Gordon Henderson's article *Understanding SPI on the Raspberry Pi*	https://projects.drogon.net/understanding-spi-on-the-raspberry-pi/
6. Link to JChart2D website	http://jchart2d.sourceforge.net/*
7. Link to Southampton's tutorial on installing MPICH on the Raspberry Pi	http://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton.htm**
8. Link to the Wikipedia article *Riemann integral*	http://en.wikipedia.org/wiki/Riemann_integral
10. Link for downloading Win32diskimager	http://sourceforge.net/projects/win32diskimager/**

Some of these files are also available in the *Annex*(*) or *Footnotes*(**) folder of the CD.

11 The terms of the CreativeCommons licence can be found on http://creativecommons.org/licenses/by-nc-sa/4.0/

Annex

Software versions:

Name	*Version*	*Date*
• Raspbian	Debian 7.1	2013-09-25
• MPICH	3.0.4	n.d.
• JChart2D	3.2.2	2011-09-24
• Pi4J	0.0.5	2013-03-18

Additional content:

Filename	Description
AD7682_7689.pdf	The data sheet for the analogical-digital converter.
BackPlane_Raspberry.pdf	The schematics of the backplane.
BackPlane_Simple.svg	A simplified block diagram of the backplane.
JavaCharting.zip	An archive containing the Java code (whole Eclipse project along with the required libraries) performing the power consumption charting. Build path must be set correctly.
LogicalRing.zip	An archive containing the C code of the logical ring with both pin toggling and heavy calculation upon token reception.
PowerMeasurement.zip	An archive containing the C code retrieving the power consumption values from the analogical-digital converter.
Schematic_Representation.pdf	A schematic representation of two bound modules.