1 Introduction

This document provides a step by step guide to get a full DIY digital station for model trains running on an ARM device. The current development is based on my initial ideas to make rocrail (www.rocrail.net) run on a raspberry pi (www.raspberrypi.org/).

The old document with such ideas – "Setting up rocpi v1.93" – is still available in the same repository where you have found this manual.

Nowadays, the target HW used as basis for this project has shifted towards the odroid platform (http://www.hardkernel.com/main/main.php), specifically the C1 (C1+) board, thought all the instructions in this document are most probably equally suitable for the C2 board.

The operating system used as basis has also shifted towards arch linux (https://archlinuxarm.org/), which in addition to be faster, more configurable and more updated than other tailored-made OS for other platforms – as Raspbian, it is available for multiple ARM devices. This allows to maintain this manual with detailed instructions both for odroid and raspberry pi (for testing purposes).

The basic software to manage the trains layout is still rocrail, though, since the rpicd software emulates standard central stations, any other control software could be used too.

The distribution of rpicd has also evolved towards the use of dockers (https://www.docker.com/) as main way to keep the software updated and independent on the HW and OS platform

The final objective is to get the ARM device running on a very simple configuration, starting rocrail as a daemon, and enabling the launching of rocview through the activation of a digital input. The messages from the rocrail – or any other control software – are received by the rpicd daemon and sent to a ucontroller in a DIY board (rpic), which translates the messages into the signals used to command trains and accessories. The rpic board is designed to provide also an s88, I2C and a CAN bus connection – not tested yet.

My intention is to use this set up with a touch screen (Packard bell Viseo 200T) and an extra Loconet board for the sensors in the layout (http://wiki.rocrail.net/doku.php?id=mgv101-en) .

This document also contains specific configuration details – not only with regards to the specific HW I intend to use – but also with regards to specific SW set-ups.

1.1 Setting up your ARM device

I will be assuming you are running the last version of the arch distro for arm. You can get it here:

- *Odroid C1 and C1+: https://archlinuxarm.org/platforms/armv7/amlogic/odroid-c1*
- *Odroid C2 (not tested): https://archlinuxarm.org/platforms/armv8/amlogic/odroid-c2*
- *RPI: https://archlinuxarm.org/platforms/armv6/raspberry-pi*
- *RPI 2 (not tested): https://archlinuxarm.org/platforms/armv7/broadcom/raspberry-pi-2*
- *RPI3 (not tested): https://archlinuxarm.org/platforms/armv8/broadcom/raspberry-pi-3*

Follow the instructions on the site and you will not have any problem. The Arch distro default user and passwords are alarm/alarm and root/root. SSH connection is activated by default.

Eventually, I will publish my own image of my system.

Some of the additional things you probably will like to do are:

- *Set a fix IP.*
- *Setting up the sound.*
- *Increase the video signal at the hdmi port. Some TVs and screens needs it.*
- *Get rid of the big black borders around the image. Again, a typical problem with some TVs and screens.*
- *Share a folder.*
- *Clean some of the daemons starting with the system, so it starts up quicker.*
- *Change the name of your device.*
- *Backup your system.*
- *Recover a lost password*

All these procedures can be found in section 4.1 of this document.

1.2 Prerequisites

1.2.1 Software dependencies

If you decide to use the docker option to run rpicd, the prerequisites are quite limited:

```
git
docker
docker-compose
```

By downloading the rpicd source, the rest of dependencies will be covered.

If, you decide to go for a self-compilation of rpicd, the dependencies are not much higher, and you will only need to include the `gcc` *and* `make` *package*

Everything is installed with the package manager of arch (pacman)

```
pacman -S git docker docker-compose gcc make
```

1.2.2 Releasing the use of the serial port

1.2.2.1 RPI

Source: http://logicalgenetics.com/serial-on-raspberry-pi-arch-linux/

There are some differences with regards to the procedure in raspbian:

1. Disable serial output during boot

Edit /boot/cmdline.txt using your favorite editor.

```
sudo nano /boot/cmdline.txt
```

Remove all chunks of text that mention ttyAMA0 but leave the rest of the line intact. Bits to remove look like:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

2. Disable the console on the serial port

This was the new bit for me. The process used to involve commenting out a line in /etc/innitab but that file is long gone.

Systemd uses links in /etc to decide what to start up, so once you find the right one, removing it is easy. You can find the files associated with consoles by doing:

```
ls /etc/systemd/system/getty.target.wants/
```

One of the entries clearly refers to ttyAMA0. It can be removed using the following command:

```
sudo systemctl disable serial-getty@ttyAMA0.service
```

The serial port in RPI is ttyAMA0

1.2.2.2 ODRDOID

In Odroid, there are up to three serial ports. Two of them are easily available: ttyS0 and ttyS2. The one at the GPIO header is TTYS2, and it is released already of any used. Actually, the console is linked to the TTYS0 port- which is not at the GPIO header.

The serial port in ODROID ttyS2

In case of willing to use the ttyS0 port, a similar approach to the RPI installation should be followed:

Edit /boot/boot.ini as root and comment the following line.

```
#setenv condev "console=tty0 console=ttyS0,115200n8"   # on both
```

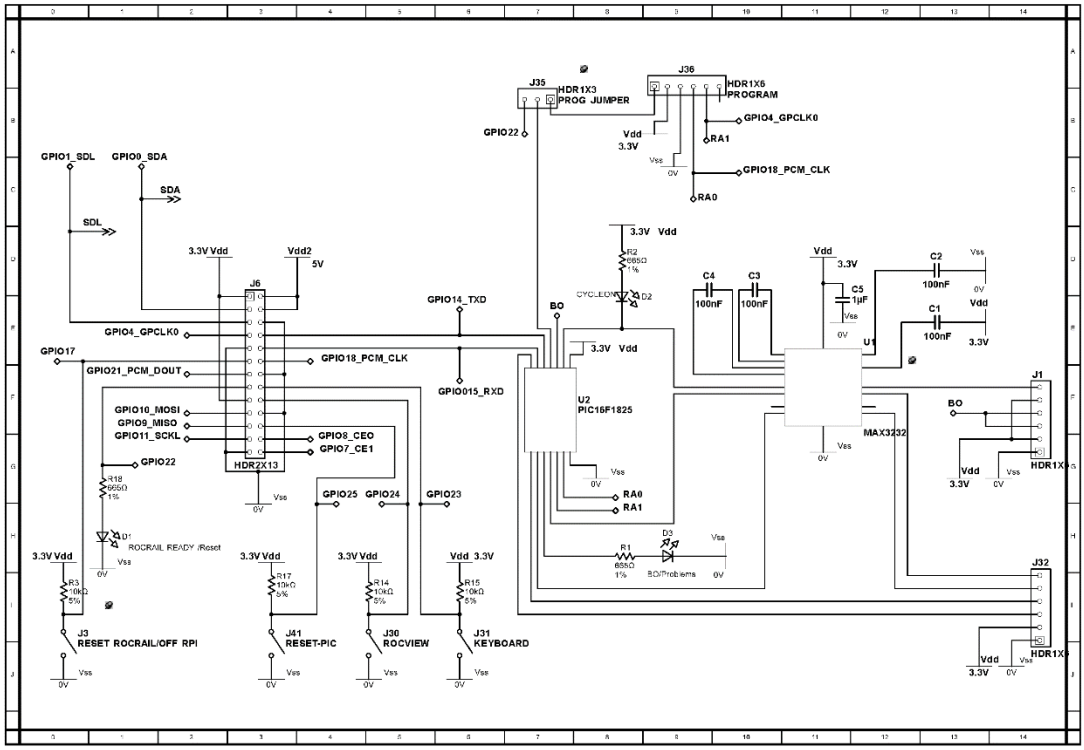
2 RPIC

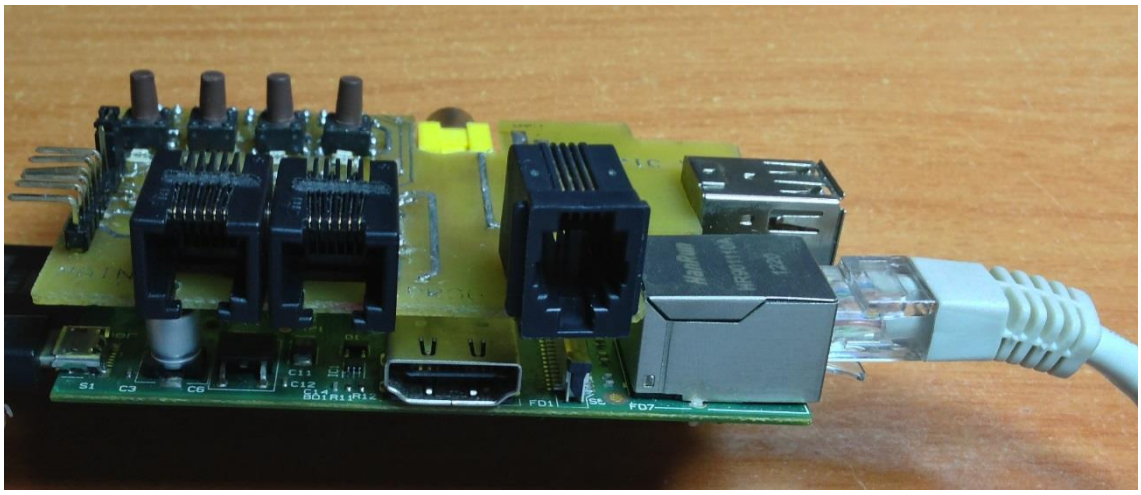
The raspberry and PIC (RPIC) is a very basic electronic circuit to have a multiprotocol central station running in the raspberry pi.

Yes, it is yet another solution for managing model trains. The main difference with other solutions is that RPIC is born to be used jointly with an ARM device in order to have a self-supported platform to generate the right signals to the train layout. The objective was to complement the raspberry pi with the minimum pieces of HW to get a full control center.

RPIC is basically code running on a Microchip microcontroller (currently a pic16f1825 or a PIC16F18326)

I have developed different versions of it. The first one, hereafter, was designed for a RPI1, with four buttons to program different actions. I used this board for more than one year without any problem.



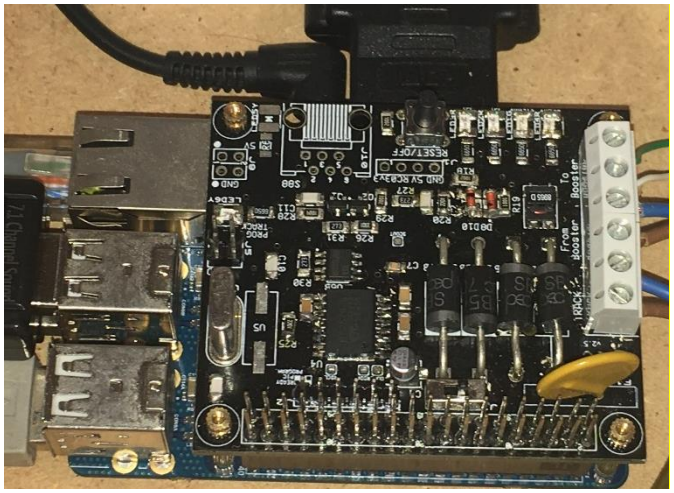


The solution has evolved to the form factor of an RP2 (and following versions) and odroid. This gives much more possibilities, both in terms of CPU capacity and GPIO availability.

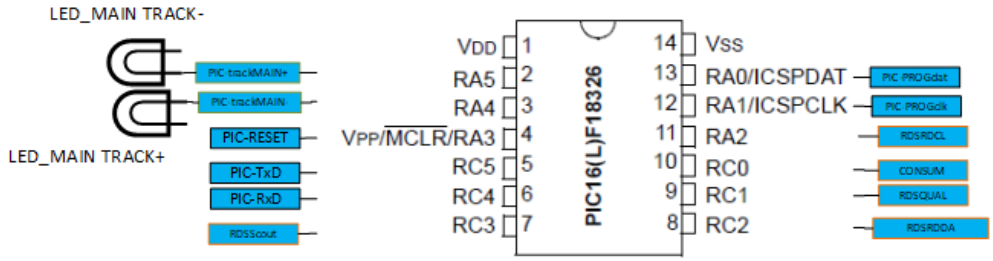
Both the firmware and hardware can be downloaded from here: https://gitlab.com/RailPIC/rpic

2.1 Hardware

Current version (v2.53) includes in a single board the design for the processing of signals to the booster, supporting Motorola, NMRA (DCC) and MMX (MM4) protocols. In addition, it provides the necessary adaptors to handle the feedback from MMX and an S88 Bus – still not fully developed. The rpic board also offers a pin header to activate a program track and let the rest of pins at the GPIO available for a second board – not developed – offering I2C and CAN communications.



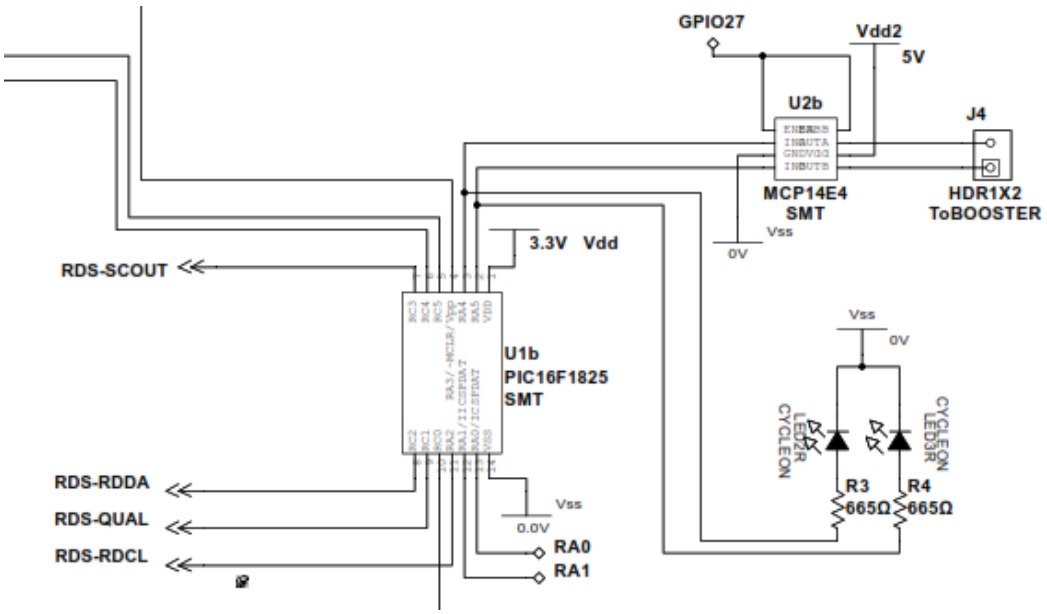
2.1.1 uController



The uController can be a PIC 16f1825 or a PIC 16f18326. There are cheap components. The second one has more memory for the program and the queue of locos that is used to handle the signals to the track – for more information look in details the firmware code. I have been developing using the first platform, which seems enough for most of the functionalities, I only started using the second uController when I first run out of memory to implement the last features related to the programming commands. Since these last commands have not been completed yet, both platforms are valid.

There are, for sure, other uControllers that could work. In the Microchip family, any controller working at least at 32Mhz (8MIPS), with a serial port and enough memory to hold the program (at least 14KB) could handle the signals generated with my code. It would be also necessary enough I/O pins for the signals coming from the RDS (MMX) detector and two generate the signals to the track. Obviously, there could be alternatives to generate the signals with just one pin – in my previous developments, this approach was used too - … but it would be necessary to work a little bit on the code to make it work. Similarly, the program should be able to be adapted to work on Arduino, or in any ATMEL uController… but by the time I started coding everything, Arduino was not yet in the market, and Microchip was the family of uControllers I had more experience with.

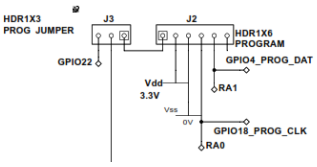
The signal to the track, which go -3V3 to 3V3 out of the uController, is adapted with a power MOSFET driver (MCP14E4) to go -5V to 5V with enough power to feed a booster, even if it is not optocoupled. I tried different approaches to generate the suitable signal: MAX3232 in first version or even feeding directly from the uController the booster…but the mosfet driver seems the best options: it enables the use of different boosters (with different requirements) and protects the uController if a higher current is required (by a short circuit for instance).

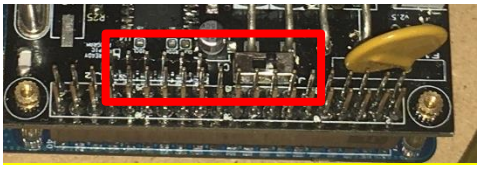


2.1.2 Programmer header

In order to program the uController, there are actually two options:

a) *Direct programming from the ARM device. This option is currently only available for the PIC 16f1825. It depends on a script in python, and is had been used for developing process.*
b) *An external Programmer directly attached to the uController.*





The software necessary for both options is documented in section 2.3.

In the rpic board it is important to note that the PROG JUMPER enables the activation of the PROGRAM header or the activation or the normal operation. When the switch is set to PROGRAM position (close to the PROGRAM header) the ARM device is not governing the reset pin, and therefore will not operate the uController.

2.1.3 Consum/Cut OFF header

The rpic board also provides a header prepared to retrieve consumption information (to get feedback from program commands) or to send a cut/off signal. The header is connected to RC0 pin, and it is currently used for developing purposes.

Originally, it was left prepared to gather external information – that was also the reason to provide power pins in the headers. Currently it is being used to generate a cut off signal for external boosters. The only reason is that my miniBOOSTER design was not considering originally the possibility of generating the cutoff signal autonomously – because I did not need it. Now that I am experimenting with railcom, I wanted a quick platfrom for experiments.

Thus, currently the header has not a definitive purpose, and it is mainly used to:

a) *Explore the feedback from consumption (program commands, not yet developed)*
b) *Generate a cut off signal to see if I can get a Railcom system.*

Obviously, it would be difficult – not impossible – to make both uses work on the same pin… but it could be possible with some sequential use of the pin.

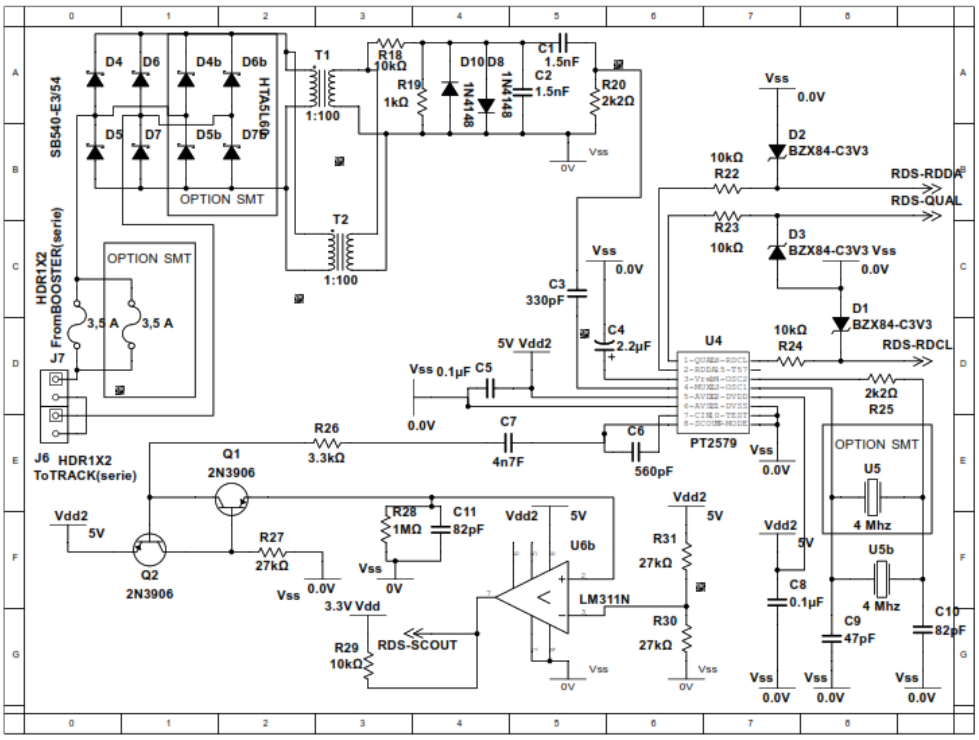
2.1.4 RDS Decoder (MMX)

The MMX decoding part takes most of the surface of the RPIC board. Mainly because, since I was exploring for the right solution, I planned for different footprint of some parts (SMT or through hole). The

circuit relies on the use of the PT2579 as RDS decoder circuit. The definition of the MFX protocol was obtained from: "Das mfx-Schienenformat", Stefan Krauß, Version 1 2009-02-01 and Rainer Mueller web site (http://www.alice-dsl.net/mue473/mfxrahmen.htm).

All the elements are there to:

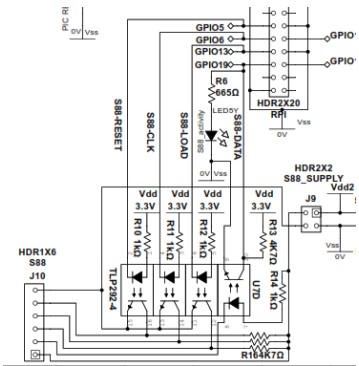
1. *adapt the signal (including a fuse for protection of the transformers)*
2. *Decode the signal with the RDS decoder*
3. *Adapt the signal to be red by the uController (with Zener Diodes and with the LM311N as comparator.*



The SCOUT signal is the one used for the simplest ACK on feedback from MMX, whilest the RDCL and RDDA contain the data on more complex responses.

2.1.5 S88

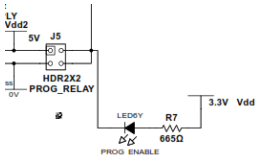
Though the supporting code is not developed yet, the rpic board provides for the necessary circuit to connect an S88 bus directly controlled from the GPIO of your arm device. The circuit is optocoupled, and through the S88 supply header it is possible to feed it with an external power source. In case of not willing to isolate the circuit, it is also possible to feed the circuit – with a couple of jumpers – to Vdd2 in your arm device (5V).



The circuit is similar to many others used with a raspberry pi to emulate an S88 bus.

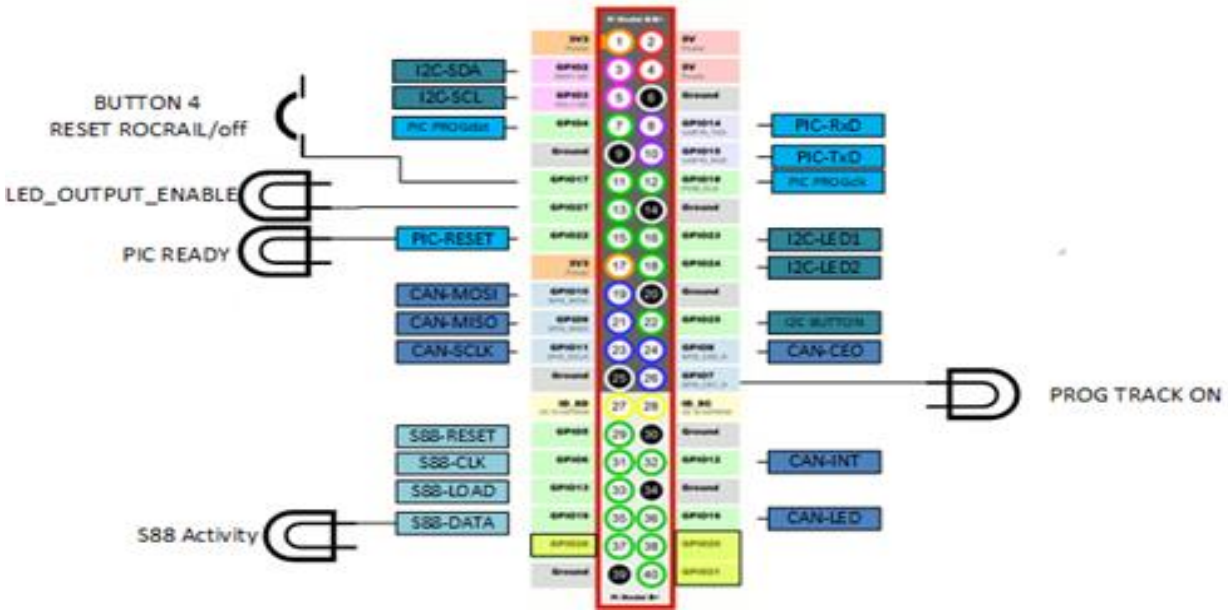
2.1.6 Programming Track

The Programming TRACK sub-circuit is the simplest one in the rpic board. It simply takes GPIO07 and offers the pin with a couple of power pins (5V and GND) to enable the connection of a relay board where it would be possible to switch the signal from the operation track to the programming track.



The first prototype included two different connectors to feed an operational track and a programming track – in parallel. However, developing the ode on the uController to enable both tracks to be running in parallel was too challenging, and I decided to adopt a more conservative approach. Now, the signal being generated is unique, and through this connector, and a external relay board, it is possible to switch from one track to another. Something like this (less than 1€) makes the magic:

2.1.7 GPIO Header



The GPIO Header in your ARM device has been left accessible to enable the connection of further boards.

The distribution of pines can be found hereafter. There is a number of pins reserved for I2C and CAN (see next section). Also pins reserved to the uController operation and the S88 operation, and finally a number of pins used for information (LEDS) and input – a button that can be used to send a reset signal and turn off the ARM device -only available currently when using rpicd as daemon, not as docker.

It is noteworthy mentioning that GPIO 20, 21 and 26 are not planned for use, as they are only analog inputs in odroid – whilst RPI2 and above use them for digital I/O.

2.1.8 Expansion board

The expansion board has not been developed yet and I do not know if I ever will commit to such effort as I do not plan to need it for my layout. It basically considers the development of and I2C and CAN sub-circuit similar to the ones used by some other rail modellers.

2.2 Firmware

The current features included in the firmware are:

- *Support for MM1 and MM2 protocol*
- *Support for NMRA (long and short addresses) – 14,28 and 128 steps*
- *Support for MMX (MM4) (not yet reading responses, but with the UID of the loco, it is possible to register the loco and send commands to it.)*
- *Support for NMRA accessories (no extended accessories so far)*
- *Support for writing CV.*

There are some other eastern eggs. The development is not completed and coming features include:

- *Full support to reading and writing CVs.*
- *MM accessories, MM1Fx and the special protocols MM3, MM4 and MM5 available in ddx (under demand).*
- *Complete the development to read MMX feedback.*

2.2.1 RPIC and RPICD interface. Protocol definition

The communications between the rpic board and the rpicd are managed through the serial port of the uController and the ARM device.

I have tried to define a serial interface simple enough to implement all the commands expected by the uController, but at the same time robust and effective not to interrupt the processing at the ucontroller and the arm device.

As a general rule, the communication in asynchronous from the arm device to the uController. The rpicd daemon sends command and the uController reacts with some feedback (from a simple ack to some additional information when more info is needed.

There are 7 types of basic commands coded by three lower bits on the first byte of a message. The other 5 bits are used to code the length of the message (up to 32 Bytes, though I have tried to keep comms below 10 bytes not to .block the rpic operation when receiving new commands from the serial stack).

COMMAND	Coding	Purpose
SYSTEM	0	Commands to manage the HW and some system options for MMX handling
ACCNMRA	1	Accessories in NMRA
ACCMM	2	Accessories in Märklin Motorloa
LOCOMM	3	Locomotives in Märklin Motorola
LOCODCC	4	Locomotives in NMRA
LOCOMMX	5	Locomotives in MMX
PROG	6	Programing options (for all protocols handled)
CONFIG	7	Commands to configure the HW

2.2.1.1 SYSTEM Commands

The SYSTEM Commands specify different types of functionalities. Some of the commands are specified to be sent between rpic and rpicd, whilst others are just used to mark the status of the rpic serial stack or to respond with and ACK error.

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
CYCLEOFF	4	0	0	RANDOM	ChecksumMESSAGE							
ERROR1 (main error)	0	0	1	Not to be send… used as CODE added to return CHECKSUM								
NEWCMD	0	0	2	Not to be send. Status								
ERROR10 (error address)	0	0	10	Not to be send… used as CODE added to return CHECKSUM								
ERROR20 (command not available)	0	0	20	Not to be send… used as CODE added to return CHECKSUM								
OVERLOAD (overload_short)	*0*	*0*	*50*	*Not to be sent from RPIC, but from rpicd to stations*								
STARTUS (Status consumption)	*0*	*0*	*51*	*Not to be sent from RPIC, but from rpicd to stations*								
PTON	4	0	100	RANDOM	ChecksumMESSAGE							
PTREADY_MM	4	0	101	RANDOM	ChecksumMESSAGE							
PTREADY_NMRA	4	0	102	RANDOM	ChecksumMESSAGE							
PTREADY_MMX	4	0	103	RANDOM	ChecksumMESSAGE							
PTREADY_ACC_MM	4	0	104	RANDOM	ChecksumMESSAGE							
PTREADY_ACC_NMRA	4	0	105	RANDOM	ChecksumMESSAGE							
SYSTEM_HALT	4	0	120	RANDOM	ChecksumMESSAGE							
EMERGENCY_BREAK	6	0	121	TYPE_LOCO	AddressH	AddressL	ChecksumMESSAGE					
REMOVE_FROM_CYCLE	6	0	122	TYPE_LOCO	AddressH	AddressL	ChecksumMESSAGE					
CHANGE_LOC_TYPE	7	0	123	TYPE_LOCO	AddressH	AddressL	NEW_TYPE	Checksum MESSAGE				
SW_TIME	5	0	124	TimeH(ms)	TimeL(ms)	ChecksumMESSAGE						
RESET	4	0	125	RANDOM	ChecksumMESSAGE							
S88_POLLING	3	0	140	Not to be sent from RPIC, but from rpicd to stations								

S88_EVENT	3	0	141	Not to be sent from RPIC, but from rpicd to stations								
PTOFF	4	0	200	RANDOM	ChecksumMESSAGE							
CYCLEON	4	0	250	RANDOM	ChecksumMESSAGE							
PING_VERIFY	9	0	150	AddressH	AddressL	U31-24	U26-13	U15-8	U7-0	Checksum MESSAGE	Checksum MESSAGE	Checksum MESSAGE
LOK_DISCOVERY	9	0	151	0X00	RANGE	U31-24	U26-13	U15-8	U7-0			
BIND_NADR	9	0	152	AddressH	AddressL	U31-24	U26-13	U15-8	U7-0			
INIT_MFX	11	0	153	AddressH	AddressL	TYPE	FN	U31-24	U26-13	U15-8	U7-0	Checksum MESSAGE

In red the commands that are still under development.

Moreover, the commands above have a translation (in most cases) to the commands sent from a MCS2 station.

TYPE	*BUFFER1 TYPE*	*MCS2_CMD*	*MCS2_Text*
CYCLEOFF	0	*0x00-0x00*	*System Stop*
ERROR1 (main error)	1		
NEWCMD	2		
ERROR10 (error address)	10		
ERROR20 (command not available)	20		
OVERLOAD (overload_short)	*50*	*0x00-0x0A*	*System-Overload*
STARTUS (Status consumption)	*51*	*0x00-0x0B*	
PTON	100		
PTREADY_MM	101		
PTREADY_NMRA	102		
PTREADY_MMX	103		
PTREADY_ACC_MM	104		
PTREADY_ACC_NMRA	105		
SYSTEM_HALT	120	*0x00-0x02*	*System-Halt*

EMERGENCY_BREAK	121	*0x00-0x03*	*System-Emergency Break*
REMOVE_FROM_CYCLE	122	*0x00-0x04*	*System-CycleOFF*
CHANGE_LOC_TYPE	123	*0x00-0x05*	*System-lok-prot*
SW_TIME	124	*0x00-0x06*	*System-SW-TIME*
RESET	125	*0x00-0x80*	*System-RESET*
S88_POLLING	140	*0x20*	*S88_POLLING*
S88_EVENT	141	*0X22*	*S88_EVENT*
PTOFF	200	*0x00-0x01*	*System GO*
CYCLEON	250	*0x06*	*Verify*
PING_VERIFY	150	*0x02*	*Lok-Discovery*
LOK_DISCOVERY	151	*0x04*	*Bind*
BIND_NADR	152	*0x00-0x01*	*System GO*
INIT_MFX	153		

2.2.1.2 ACCNMRA Command

The ACCNMRA Command only specify 1 type of sub-command, to activate/deactivate a NMRA accessory decoder of 8 ports.

TYPE	BUFFER 0 LENGTH/COMMAND	BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
ACCNMRA	6	1	0	AddressH	AddressL	BYTE FUNCTION1	ChecksumMESSAGE					

The command has a translation to the command sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
ACCNMRA	0	0x16	Accessories

2.2.1.3 ACCMM Command

The ACCNMM Command only specify 1 type of sub-command, to activate/deactivate a MM accessory decoder of 8 ports.

This is one of the commands that t is not implemented at all, and that will only be developed under demand, as I do not plan to use MM accessory decoders.

Furthermore, this command has some challenges in terms of timing, as the bit codinf for the signal requires the shortest pulse of all the protocols supported (aorund 13us),

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
ACCMM	5	2	0	AddressL	BYTE FUNCTION1	ChecksumMESSAGE						

The command has a translation to the command sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
ACCNMRA	0	0x16	Accessories

2.2.1.4 LOCOMM Commands

The LOCOMM Commands specify different types of MM protocols and functionalities. Some of the commands are specified to be sent from rpic to the tracks, and are only defined for signalization purpose (they do not have to be stored in the loco queue of commands).

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
LOCOMM1	6	3	1	AddressL	Speed+direction	BYTE FUNCTION1	ChecksumMESSAGE					
LOCOMM1Fx	5	3	2	AddressL	BYTE FUNCTION1	ChecksumMESSAGE						
LOCOMM1R	0	3	3	AddressL		Not to be sent, minimum information needed to build this packet. Not to be sotred						
LOCOMM2	6	3	4	AddressL	Speed+direction	BYTE FUNCTION1	ChecksumMESSAGE					
LOCOMM2Fx	0	3	5	AddressL	Speed+direction	BYTE FUNCTION1	FUNCTION GROUP	Not to be sent, minimum information needed to build this packet				
LOCOMM3	6	3	6	AddressL	Speed+direction	BYTE FUNCTION1	ChecksumMESSAGE					
LOCOMM3Fx	7	3	7	AddressL	Speed+direction	BYTE FUNCTION1	FUNCTION GROUP	Checksum MESSAGE				
LOCOMM4	6	3	8	AddressL	Speed+direction	BYTE FUNCTION1	ChecksumMESSAGE					
LOCOMM4Fx	7	3	9	AddressL	Speed+direction	BYTE FUNCTION1	FUNCTION GROUP	Checksum MESSAGE				
LOCOMM5	6	3	10	AddressL	Speed+direction	BYTE FUNCTION1	ChecksumMESSAGE					
LOCOMM5Fx	7	3	11	AddressL	Speed+direction	BYTE FUNCTION1	FUNCTION GROUP	ChecksumMESSAGE				

As it can been seen only MM1 and MM2 is supported. The MM3-MM5 protocols defined by some other command stations based on the production of serial commands (as DDX or DDW) are specified but not yet developed as I do not have locos to test them and the definition of the protocol is not correctly documented.

In red the commands that are still under development.

Moreover, the commands above have a translation (in most cases) to the commands sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
LOCOMM1	1	0x08-0x0A	Speed&Dir
LOCOMM1Fx	2		
LOCOMM1R	3		
LOCOMM2	4	0x08-0x0A	Speed&Dir
LOCOMM2Fx	5	0x0C	function
LOCOMM3	6	0x08-0x0A	Speed&Dir
LOCOMM3Fx	7	0x0C	function
LOCOMM4	8	0x08-0x0A	Speed&Dir
LOCOMM4Fx	9	0x0C	function
LOCOMM5	10	0x08-0x0A	Speed&Dir
LOCOMM5Fx	11	0x0C	function

2.2.1.5 LOCODCC Commands

The LOCODCC Commands specify different types of NMRA (DCC) protocols and functionalities. In this case, all commands have been developed, and all commands are sent from the rpicd to the rpic. The advantage of having a fully open and documented protocol enabled a fast development of the different options..

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
LOCODCCN14	5	5	100	AddressL	Speed+direction	Checksum MESSAGE						
LOCODCCN28	5	5	101	AddressL	Speed+direction	Checksum MESAGE						
LOCODCCN128	5	5	102	AddressL	Speed+direction	Checksum MESSAGE						
LOCODCCNfx	6	5	103	AddressL	BYTE FUNCTION1	FUNCTION GROUP	Checksum MESSAGE					
LOCODCCL14	6	5	104	AddressH	AddressL	Speed+direction	Checksum MESSAGE					
LOCODCCL28	6	5	105	AddressH	AddressL	Speed+direction	Checksum MESSAGE					
LOCODCCL128	6	5	106	AddressH	AddressL	Speed+direction	Checksum MESSAGE					
LOCODCCLfx	7	5	107	AddressH	AddressL	BYTE FUNCTION1	FUNCTION GROUP	Checksum MESSAGE				

As it can been seen there is a different type of command depending on the number of steps (14,28,128) and the size of the address (Normal or Long)

The commands above have a translation to the commands sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
LOCODCCN14	*100*	*0x08-0x0A*	*Speed&Dir*
LOCODCCN28	*101*	*0x08-0x0A*	*Speed&Dir*
LOCODCCN128	*102*	*0x08-0x0A*	*Speed&Dir*
LOCODCCNfx	*103*	*0x0C*	*function*
LOCODCCL14	*104*	*0x08-0x0A*	*Speed&Dir*
LOCODCCL28	*105*	*0x08-0x0A*	*Speed&Dir*
LOCODCCL128	*106*	*0x08-0x0A*	*Speed&Dir*
LOCODCCLfx	*107*	*0x0C*	*function*

2.2.1.6 LOCOMMX Commands

The LOCOMMX Commands specify different types of MMX functionalities. The basic difference is the number of functions available. Originally, the proposal was to differentiate also on the size of the address and the steps of the motor, but I decided that the firmware would be responsible of selecting the best type of commands to be sent to the tracks depending on the size of the address and the steps being transmitted. In this case, all commands have been developed and tested.

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
LOCOMMX_F4	*7*	*4*	*200*	*AddressH*	*AddressL*	*Speed+direction*	*BYTE FUNCTION1*	*Checksum MESSAGE*				
LOCOMMX_F8	*7*	*4*	*201*	*AddressH*	*AddressL*	*Speed+direction*	*BYTE FUNCTION1*	*Checksum MESSAGE*				
LOCOMMX_F16	*8*	*4*	*202*	*AddressH*	*AddressL*	*Speed+direction*	*BYTE FUNCTION2*	*BYTE FUNCTION1*	*Checksum MESSAGE*			
LOCOMMX_FN	*7*	*4*	*203*	*AddressH*	*AddressL*	*Speed+direction*	*Fn+BYTE FUNCTION*	*Checksum MESSAGE*				

The MFX (MM4) protocol defines actually much more granularity when it comes to define the commands sent to the tracks:

- *Addresses can be defined with 7, 9, 11 and 14 bits.*
- *Speed and direction can be defined with 4 and 8 bits.*
- *And the functions with 4, 8, 16 bits, ow with a generic package that can specify up to 127 functions (7bits).*

The commands above have a translation to the commands sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
LOCOMMX_F4	200	0x08-0x0A	Speed&Dir
LOCOMMX_F8	201	0x08-0x0A	Speed&Dir
LOCOMMX_F16	202	0x08-0x0A	Speed&Dir
LOCOMMX_FN	203	0x08-0x0A	Speed&Dir

2.2.1.7 PROG Commands

The PROG Commands specify the different types of commands for each of the protocol. At this stage only the NMRA commands have been developed, and from them, only the WRITE commands have been tested. The MM and MMX commands will be developed soon.

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
NMRA_SERVICE_MODE_ADDRESS_write	4	6	0	Value	Checksum MESSAGE							
NMRA_SERVICE_MODE_ADDRESS_read	4	6	1	Value	Checksum MESSAGE							
NMRA_SERVICE_MODE_REGISTER_write	5	6	2	Registry	Value	Checksum MESSAGE						
NMRA_SERVICE_MODE_REGISTER_read	5	6	3	Registry	Value	Checksum MESSAGE						
NMRA_SERVICE_MODE_PAGE_write	6	6	4	Registry	Page	VALUE	Checksum MESSAGE					
NMRA_SERVICE_MODE_PAGE_read	6	6	5	registry	Page	VALUE	Checksum MESSAGE					
NMRA_SERVICE MODE_DIRECT_write	6	6	6	CV_H	CV_L	Value	Checksum MESSAGE					
NMRA_SERVICE MODE_DIRECT_verify	6	6	7	CV_H	CV_L	Value	Checksum MESSAGE					
NMRA_SERVICE_MODE_DIRECT_bit	6	6	8	CV_H	CV_L	Value/bit	Checksum MESSAGE					
ACCNMRAwrite	8	6	50	AddressH	AddressL	CV_H	CV_L	VALUE	Checksum MESSAGE			
ACCNMRAverify	8	6	51	AddressH	AddressL	CV_H	CV_L	VALUE	Checksum MESSAGE			
ACCNMRAbit	8	6	52	AddressH	AddressL	CV_H	CV_L	value/bit	Checksum MESSAGE			
LOCODCCNwrite	7	6	53	AddressL	CV_H	CV_L	value	Checksum MESSAGE				

LOCODCCNverify	7	6	54	AddressL	CV_H	CV_L	value	Checksum MESSAGE				
LOCODCCNbit	7	6	55	AddressL	CV_H	CV_L	value/bit	Checksum MESSAGE				
LOCODCCLwrite	8	6	56	AddressH	AddressL	CV_H	CV_L	VALUE	Checksum MESSAGE			
LOCODCCLverify	8	6	57	AddressH	AddressL	CV_H	CV_L	VALUE	Checksum MESSAGE			
LOCODCCLbit	8	6	58	AddressH	AddressL	CV_H	CV_L	value/bit	Checksum MESSAGE			
MM	2	6	100									
MMX	2	6	150									

In red the commands that are still under development.

Moreover, the commands above have a translation (in most cases) to the commands sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
NMRA_SERVICE_MODE_ADDRESS_write	0		
NMRA_SERVICE_MODE_ADDRESS_read	1		
NMRA_SERVICE_MODE_REGISTER_write	2	0x10	WRITE
NMRA_SERVICE_MODE_REGISTER_read	3	0x0e	READ
NMRA_SERVICE_MODE_PAGE_write	4		
NMRA_SERVICE_MODE_PAGE_read	5		
NMRA_SERVICE MODE_DIRECT_write	6	0x10	WRITE
NMRA_SERVICE MODE_DIRECT_verify	7	0x0e	READ
NMRA_SERVICE_MODE_DIRECT_bit	8	0x10	WRITE
ACCNMRAwrite	50	0x10	WRITE
ACCNMRAverify	51	0x0e	READ
ACCNMRAbit	52	0x10	WRITE
LOCODCCNwrite	53	0x10	WRITE
LOCODCCNverify	54	0x0e	READ

LOCODCCNbit	55	0x10	WRITE
LOCODCCLwrite	56	0x10	WRITE
LOCODCCLverify	57	0x0e	READ
LOCODCCLbit	58	0x10	WRITE
MM	100	0x10	WRITE
MMX	150	0x10	WRITE

2.2.1.8 CONFIG Commands

The CONFIG Commands specify different actions affecting the configuration of the rpic board. It also includes commands that the rpicd daemon must support to emulate a MCS2.. but these commands are still under research..

TYPE	BUFFER 0 LENGTH/COMMAND		BUFFER1 TYPE	BUFFER 2	BUFFER 3	BUFFER 4	BUFFER 5	BUFFER 6	BUFFER 7	BUFFER 8	BUFFER 9	BUFFER 10
TESTON	4	7	50	RANDOM	Checksum MESSAGE							
TESTOFF	4	7	200	RANDOM	Checksum MESSAGE							
VERSION	4	7	210	RANDOM	Checksum MESSAGE							
ZENTRALE_UID_WRITE	7	7	150	U31-24	U26-13	U15-8	U7-0	Checksum MESSAGE				
ZENTRALE_UID_READ	4	7	151	RANDOM	Checksum MESSAGE							
SESSION_N_WRITE	5	7	152	S14-8	S7-0	Checksum MESSAGE						
SESSION_N_READ	4	7	153	RANDOM	Checksum MESSAGE							
ENABLEPROT	4	7	10	ENABLE_PROT	Checksum MESSAGE							
CONF_DEVICES	0	7	12	Not to be send to rpic... I think it is from RPICD to oher devices... not clear how to use								
PING_SW	0	7	13	Not to be send to rpic... I think it is from RPICD to oher devices... not clear how to use								
CONFIG_STATUS	0	7	14	Not to be send to rpic... I think it is from RPICD to oher devices... not clear how to use								
REQUESTCONF	0	7	15	Not to be send to rpic... I think it is from RPICD to oher devices... not clear how to use								
CONFIG_DATA	0	7	16	Not to be send to rpic... I think it is from RPICD to oher devices... not clear how to use								
CONFIG_AUT	0	7	17	Not to be send to rpic... I think it is from RPICD to oher devices... not clear how to use								

In red the commands that are still under development.

Moreover, the commands above have a translation (in most cases) to the commands sent from a MCS2 station.

TYPE	BUFFER1 TYPE	MCS2_CMD	MCS2_Text
TESTON	50		
TESTOFF	200		
VERSION	210		
ZENTRALE_UID_WRITE	150		
ZENTRALE_UID_READ	151		
SESSION_N_WRITE	152	0x00-0x09	System_sessionNr
SESSION_N_READ	153		
ENABLEPROT	10	0x00-0x08	System_EnableProt
CONF_DEVICES	12	0x00-0x0C	System_devices
PING_SW	13	0X30	Can PING
CONFIG_STATUS	14	0x3a	CONFIF_STATUS
REQUESTCONF	15	0x40	REQUEST_CONF_DATA
CONFIG_DATA	16	0X42	CONFIG_DATA
CONFIG_AUT	17	0X60	CONFIG_AUT

2.2.2 Loco Queue

One of the most important aspects on the rpic firmware is the way the information to be sent to the tracks – regarding the Locomotive message – is stored. In any central station it is necessary to refresh continuously the information to the locomotives. Therefore, a stack with the status of the locomotive is formed and in a never-ending loop the locomotives receive a refresh on their status. This helps not only to keep power in the tracks, but also to correct potential losses of message, as the messages received are retransmitted in loop.

The rpic stack is formed with a queue of 800 bytes (16f1825) or 1800 bytes (16f18326) which enables storing a large number of locos depending on the used protocols. For instance, it could store 90 NMRA, 20 MM and 10 MMX locos, or any other combination considering that as average the number of bytes per loco is 6 for MM and NMRA and 12 for MMX.

Details on the number of bytes needed per packaged stored and the information that is being stored can be found hereafter.

	Position Com, + Type		Position+1	Position+2	Position+3	Position+4	Position+5																					
LOCOMM1	3	1	AddressL	Speed+ direction	BYTE FUNCTION1	Next1	Next0																					
LOCOMM2	3	4	AddressL	Speed+ direction	Fgroup|BYTE FUNCTION1	Next1	Next0																					
LOCODCCN14	5	100	AddressL	Speed+ direction	Next1	Next0																						
LOCODCCN28	5	101	AddressL	Speed+ direction	Next1	Next0																						
LOCODCCN128	5	102	AddressL	Speed+ direction	Next1	Next0																						
LOCODCCNfx	5	103	AddressL	BYTE FUNCTION1	FUNCTION GROUP	Next1	Next0																					
LOCODCCL14	5	104	AddressH	AddressL	Speed+direction	Next1	Next0																					
LOCODCCL28	5	105	AddressH	AddressL	Speed+direction	Next1	Next0																					
LOCODCCL128	5	106	AddressH	AddressL	Speed+direction	Next1	Next0																					
LOCODCCLfx	5	107	AddressH	AddressL	BYTE FUNCTION1	FUNCTION GROUP	Next1	Next0																				
LOCOMMX_F4	4	200	AddressH	AddressL	Speed+direction	F1	U31-24	U26-13	U15-8	U7-0	Next1	Next0																
LOCOMMX_F8	4	201	AddressH	AddressL	Speed+direction	F1	U31-24	U26-13	U15-8	U7-0	Next1	Next0																
LOCOMMX_F16	4	202	AddressH	AddressL	Speed+direction	F2	F1	U31-24	U26-13	U15-8	U7-0	Next1	Next0															
LOCOMMX_FN	4	203	AddressH	AddressL	Speed+direction	NF	F16	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	U31-24	U26-13	U15-8	U7-0	Next1	Next0

It is noteworthy mentioning that the last type of package LOCOMMX_FN, the number of bytes is variable, with a worst scenario (128 function bits) of 27 bytes. This package has been defined in this way (storing the status of all function bits) in order to refresh the status of all function bits within the queue. If it was decided to only refresh the status of the functions being activated/deactivated the size in queue could be reduced. Moreover, if only the last message was considered to be stored in the queue, the size of the package could be reduced to 12 (as LOCOMMX_F16)… but then the large capacity of LOCOMMX_FN to handle large amount of functions would be lost.

Thist ype of message has not been tested with any loco supporting such functionality (yet).

2.3 Programming the firmware and start using the rpic

2.3.1 Programing with an external programmer

Using am external programmer is always possible. Actually, if the uController has not been programmed before and the Low Voltage Programming is not available, this will be the only way to program the board.

REMEMBER TO UNPLUG THE RPIC BOARD FROM THE RASPBERRY PI AND CHANGE THE JUMPER FOR PROGRAMMING.

Personally, I use a PICkit2 programmer (below a picture with jumper position and header connection). Obviously any external programmer can be used, matching the right pins to the PICkit header. For the PIC 16f18326 you may need a most modern version, as the PICkit3.

2.3.2 Programing from ARM device with rpicd docker

The simplest solution to program the 16f1825 is to make use of the rpicd docker image – see next sections. If you are using rpicd through a docker, it is already provided with all the necessary tools to program the uController.

In future versions of rpicd, I will provide for a user interface trhough the rpicd web, but currently, it is already possible to use the functionality in this way:

1. *Make sure the rpicd container is running (if you follow the instructions in section 3.1.5, it will be running)*
2. *In a command shell, type the following:*

```
docker exec -it rpicd bin/sh
```

This will open a shell within the rpicd container. Now it is easy.

```
cd LVP
./programrpic.sh ../sw/release/XC8_16F1825/rpic.hex
```

You can exit the shell now CTRL+D

2.3.3 Direct programing from the arm device

If the ucontroller has never been programmed before, you can try to program directly from the raspberry pi. I am providing a modification from a script made by Daniel Perron, suitable to program the 16f1825.

In case of error look for the dependencies explained in next sections.

Source: https://github.com/danjperron/A2D_PIC_RPI

1. *Download the last version of RPIC and programing script (you only need the most recent in master (--depth=1)*

```
git clone --depth=1 https://gitlab.com/RailPIC/rpic.git
```

2. Navigate to the programing folder

```
cd rpic/LVP
```

3. Run the install script
   BE SURE THE JUMPER FOR PROGRAMMING IS IN NORMAL POSITION

```
sudo ./install
```

Now, if everything went OK, you should be able to start using the rpic. If you receive any error message, or simply the ucontroller has been programed before blocking the capability to be reprogram at low voltage (LVP), there is always the option to move the jumper to PROGRAM POSITION, and make use of a microchip programmer as pickit.

By default, the install script installs the python dependences explained hereafter. If, in any case you still have problems installing them (typical issue of back compatibility), you can follow the instructions hereafter

2.3.3.1 Dependency with intelHex

One potential error you can receive from the programming script in python is a missing library needed to read the firmware file. This is solved with the intelhex-1.5.tar.gz file included in the LVP folder.

1. Go to the LVP folder and unzip the module

```
tar -vxf intelhex-1.5.tar.gz
```

2. Go to the new folder unzipped and run the setup

```
cd intelhex-1.5
sudo python setup.py install
```

Now you can use directly the program script with the file for the firmware:

```
sudo ./programrpic.sh ../sw/release/XC8_16F1825/rpic.hex
```

2.3.3.2 Dependency with RPi.GPIO

One potential error you can receive from the programming script in python is a missing library needed to work with the firmware file. This is solved with the RPi.GPIO-0.6.2.tar.gz file included in the LVP folder.

1. Go to the LVP folder and unzip the module

```
tar -vxf RPi.GPIO-0.6.2.tar.gz
```

2. Go to the new folder unzipped and run the setup

```
cd RPi.GPIO-0.6.2
sudo python setup.py install
```

Now you can try again to run the install script (2.3) or use directly the program script with the file for the firmware:

```
sudo ./programrpic.sh ../sw/release/XC8_16F1825/rpic.hex
```

3 RPIC Daemon (RPICD)

The rpicd is the daemon part of the railpic project. It is essentially the server responsible of communicating to the rpic board through the serial port of the ARM device.

I have explored several options for rpicd: acting as a rocrail client, emulating an ECOS… and finally emulating a CS2.

All the explored paths remain active in the daemon – you can explore the code for durther details – but only the CS2 option is being kept under development, as it gives flexibility to the railppic project to be interface by different software control.

3.1 Configuration options (rpicd.xml)

The rpicd is configured through an xml file.

An example is provided with the code. The parameters stated are the ones rpicd take as default. In case of not being included in the configuration file, these are the values that will be used.

```
<rpicd log="rpicd.log" port="8015">
    <!-- Settings for rpicd -->
    <rpiconfig resetoffpin = 0 ledreadyrestpin = 3 sec4halt = 4
rpiserial="/dev/ttyS2" maxserialtry="5" sec4resetpic="2" msecserialtry="500"
progtrackpin="11" ledsteadypin="2"/>
    <rocclient IP="localhost" port="8051" locodepot="true"
parser="false"></rocclient>
    <commandparser port="15731" maxclientes="3" locopath=""
type="mcs2"></commandparser>
    <speedcurve>
    <lc id="BR44">
      <speedcurvestep step="0" nsleep="0" psleep="100"/>
      <speedcurvestep step="1" nsleep="0" psleep="100"/>
      <speedcurvestep step="5" nsleep="0" psleep="100"/>
    </lc>
    <lc id="130TB SNCF">
      <speedcurvestep step="0" nsleep="3000" psleep="0"/>
      <speedcurvestep step="1" nsleep="0" psleep="1500"/>
      <speedcurvestep step="2" nsleep="0" psleep="1000"/>
      <speedcurvestep step="3" nsleep="0" psleep="2000"/>
      <speedcurvestep step="4" nsleep="0" psleep="100"/>
      <speedcurvestep step="5" nsleep="0" psleep="100"/>
      <speedcurvestep step="6" nsleep="0" psleep="500"/>
      <speedcurvestep step="7" nsleep="0" psleep="100"/>
      <speedcurvestep step="8" nsleep="0" psleep="500"/>
      <speedcurvestep step="9" nsleep="0" psleep="300"/>
      <speedcurvestep step="10" nsleep="0" psleep="500"/>
      <speedcurvestep step="11" nsleep="0" psleep="300"/>
      <speedcurvestep step="12" nsleep="0" psleep="200"/>
      <speedcurvestep step="13" nsleep="0" psleep="200"/>
      <speedcurvestep step="14" nsleep="0" psleep="100"/>
    </lc>
    <lc id="Bomberos">
      <speedcurvestep step="0" nsleep="0" psleep="100"/>
    </lc>
    <lc id="V160">
      <speedcurvestep step="0" nsleep="0" psleep="100"/>
    </lc>
    </speedcurve>
    <mmx>
      <lc id="D XII" UID="0xFFF49080"></lc>
      <lc id="T3-89" UID="0x7DFA8DFC"></lc>
    </mmx>
</rpicd>
```

3.1.1 Log Configuration path and port

```
<rpicd log="rpicd.log" port="8015">
```

Within the root element of the configuration file, it is possible to identify the path to the log file and the port used for the web interface.

The access to different rpicd.xml files can be stated when calling the rpicd daemon

```
sudo ./rpicd /path/to file/rpicd.xml
```

Note that rpicd must be run as root, or at least with enough privileges to access the GPIO.

3.1.2 ARM configuration option (rpiconfig)

```
<rpiconfig resetoffpin = 0 ledreadyrestpin = 3 sec4halt = 4 rpiserial="/dev/ttyS2"
maxserialtry="5" sec4resetpic="2" msecserialtry="500" progtrackpin="11"
ledsteadypin="2"/>
```

The interface to the GPIO header and serial port is specify withing the rpiconfig element. It is noteworthy mentioning that the rpicd uses the GIPO numbering defined by the wiringPi library – thanks Gordon Hendersons for this grate library.

P1: The Main GPIO connector:

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
—	—	3.3v	1 | 2	5v	—	—
8	R1:0/R2:2	SDA	3 | 4	5v	—	—
9	R1:1/R2:3	SCL	5 | 6	0v	—	—
7	4	GPIO7	7 | 8	TxD	14	15
—	—	0v	9 | 10	RxD	15	16
0	17	GPIO0	11 | 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 | 14	0v	—	—
3	22	GPIO3	15 | 16	GPIO4	23	4
—	—	3.3v	17 | 18	GPIO5	24	5
12	10	MOSI	19 | 20	0v	—	—
13	9	MISO	21 | 22	GPIO6	25	6
14	11	SCLK	23 | 24	CE0	8	10
—	—	0v	25 | 26	CE1	7	11
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin

The rpiconfig element enables to configure the following options:

- ***rsetoffpin=0**. This is the GPIO pin connected to the button taking care of requesting a ucontroller reset and the shutdown of the ARM device. If using the rpic board, **this value should not change**.*
- ***ledreadyrestpin = 3**. This is the GPIO connected to the reset pin of the uController and the led informing the ucontroller is ready. If using the rpic board, **this value should not change**.*

- ***sec4halt = 4.*** *Number of seconds that are necessary to push the button (resetoffpin) to activate the shut down command to the arm device.*
- ***sec4resetpic="2.*** *Number of seconds the uController is kept under reset conditions.*
- ***rpiserial="/dev/ttyS2"****. Serial port of the arm device. If using the docker option, it is not relevant. Otherwise, it is necessary to match the serial port available in the odroid (see section 1 for details on the serial port available for each ARM device.)*
- ***maxserialtry="5"****. Number of tries to send a serial command, before error.*
- ***msecserialtry="500"****. Milliseconds to wait between attempts to send a serial command.*
- ***progtrackpin="11"****. This is the GPIO pin connected to the header to activate a programming track. If using the rpic board,* ***this value should not change****.*
- ***ledsteadypin="2"****. This is the GPIO pin connected to the led informing rpicd is ready to work. If using the rpic board,* ***this value should not change****.*

3.1.3 Rocclient configuration option (rocclient)

```
<rocclient IP="localhost" port="8051" locodepot="true" parser="false"></rocclient>
```

The definition of rpicd as a client of Rocrail is one of the firsr options I explored, though is currently deprecated, it is still used by default to retrieve the list of locomotives to be managed by rpicd. The parameters are:

- ***IP="localhost".*** *The IP of the Rocrail server.*
- ***port="8051" .*** *The port to connect to the Rocrail server*
- ***locodepot="true".*** *The main use of the rocclient nowdays is to retrieve the list of locomotives. There is still no other option to get the list of locos to be handled. In the future they could be included as part of the rpicd.xml file.*
- ***parser="false".*** *The commands sent from the Rocrail server to its clients can be parsed. This is an old functionality that I am no longer supporting… but it is kept in case someone wants to explore it.*

3.1.4 Command Parser configuration option (commandparser)

```
commandparser port="15731" maxclients="3" locopath="" type="mcs2"></commandparser>
```

Currently rpicd support two type of parsers, emulating two types of command statiosn. The only one that is maintained ins MCS2 (Märklin Central Station 2).The parameters are:

- ***type="mcs2".*** *Only two options – mcs2 and ecos. ECOS is no longer maintained.*
- ***port="15731".*** *Port that is open to receive the messages.*
- ***maxclients="3".*** *Maximum number of clients that will be accepted.*
- ***locopath="".*** *Not yet completed, this paremeter will specify a path to a file containing the locomotives to be handled.*

rpicd was designed to be able to use more than one commad parser, so if in the future a new parser is needed, it will be always possible to include a second line in the rpicd.xm file defining a second parser.

3.1.5 Speedcurve

```
<speedcurve>
    <lc id="BR44">
      <speedcurvestep step="0" nsleep="0" psleep="100"/>
      <speedcurvestep step="1" nsleep="0" psleep="100"/>
      <speedcurvestep step="5" nsleep="0" psleep="100"/>
    </lc>
</speedcurve>
```

The speedcurve functionality is offered to automatically adjust the seed of a loco. It is necessary to specify the id of the handled loco. Then it is possible to specify only the steps affected. Rpicd will send the commands that are stated in the file, waiting ***n*** *milisseconds when goingto the* ***next*** *(nslepp) or* ***previous*** *(pnsleep) step. The curve is symmetric in forward and backward direction, and if tracths the zero step with a psleep defined, it means that rpicd must send twice the zero speed command waiting n milliseconds.*

There is no limit in the number of locomotives that can be defined. There are only two restrictions:

- *The id defined must match the id of the locodepot defined.*
- *The biggest steo must be coherent with the number of steps defined at the locodepot. If a 14 steps loco uses a speedcurve with 28 steps, the uController will not accept the commands of the steps beyond 14.*

3.1.6 MMX

```
<mmx>
      <lc id="D XII" UID="0xFFF49080"></lc>
</mmx>
```

This is the part of the rpicd file where the UIDs of the mmx locomotives are stored. Storing the UIDs of the registered locomotives facilitate the use of the mmx functionalities – avoiding to find out the UIDs of any mmx locomotive in the track.

The UID is essential to be able to send commands to the mmx locomotives. The only restriction to be considered, once again, is that the id defined must match the id of a locomotive defined in the locodepot.

3.2 Running RPICD within a docker

Running as a docker is the simplest option to make use of the rpicd. I am currently providing a functional docker for arm6 (raspberry pi 1) and amr7 (odroid and newer raspberry pi). The steps to get it are quite simple if you have followed the instruction to install docker and docker-compose.

1. *Enable and start the docker engie*

```
sudo systenctl start docker.service
sudo systenctl enable docker.service
```

2. *Go to a new folder and create a docker-compose.yml file.*

```
cd ~
mkir rpicd-docker
cd rpicd-docker
nano docker-compose.yml
```

3. *Copy the following parameters into docker-compose.yml (selecting arm6 or arm7)*

```
version: '3'

services:
  rpicd:
    container_name: rpicd
    image: raildocker/rpicd:arm6
    volumes:
      - config:/config
#First part should match the named volume at the end of the file,
#Second part (the docker part) should match the WORKDIR ENVRIRONMENT at the
environment section
      - /sys:/sys
#Uncomment to run on x86 directories
#      - /usr/bin/qemu-arm-static:/usr/bin/qemu-arm-static
    ports:
```

```
      - "8015:8015"
    environment:
      - WORKDIR=/config      #It should match the mapping part of the docker
(second part) in the volume define above
    restart: always
    devices:
      - /dev/gpiomem:/dev/gpiomem
      - /dev/ttyAMA0:/dev/ttyS2


volumes:
  config:         #it should match the volume defines on top.
    driver: local
```

4. Start the docker (it will be restarted automatically every time to turn on your arm device)

```
Docker-compose up
```

3.2.1 Docker-compose in details

The docker compose file includes a number of parameters that can help adapting the docker to different environments.

1. Service definition.

```
services:
  rpicd:
    container_name: rpicd
    image: raildocker/rpicd:arm6
```

The main parameter that can change is the arm6/arm7 tag to the docker that must be selected ending on the ARM targeted device.

2. Volume definition.

```
    volumes:
      - config:/config
#First part should match the named volume at the end of the file,
#Second part (the docker part) should match the WORKDIR ENVRIRONMENT at the
environment section
      - /sys:/sys
#Uncomment to run on x86 directories
#      - /usr/bin/qemu-arm-static:/usr/bin/qemu-arm-static
```

There are three volumes that can be mounted (some of them mandatory):

- **config:/config** -> it matches the config folder withinthe docker (second part) with a persistent folder in the arm device. By default, a volume that the docker engine will maintain and that can be found at /var/lib/docker/volumes/rpicd-docker_config/_data. This volume is mandatory, but you can select any other folder to store the basic rpicd cofig files and access it from the arm device. For instance: /home/user/rpicd_files:/config.
- **/sys:/sys** -> it mates the /sys filesystem. It is mandatory to be able to access the GPIO.
- **/usr/bin/qemu-arm-static:/usr/bin/qemu-arm-static** ->it is possible to run the docker from a non-arm device by uncommenting this volume (if you have installed qemu at your host device).

Note that the first volume requires also to specify the docker volume at the end of the file:

```
volumes:
  config:          #it should match the volume defines on top.
     driver: local
```

And it is also necessary to include the following variable in environment:

```
    environment:
      - WORKDIR=/config      #It should match the mapping part of the docker
(second part) in the volume define above
```

3. Port definition.

```
    ports:
      - "8015:8015"
```

The port defines where the web interface will be reachable. The host port (first part) can be modified and select another port.

4. Restart option

```
restart: always
```

The docker by default restart automatically. This is something that can be changed to no, on-failure, always and unless-stopped

5. Devices definition

```
    devices:
      - /dev/gpiomem:/dev/gpiomem
      - /dev/ttyAMA0:/dev/ttyS2
```

This section is again mandatory as it enables the access to the GPIO and the serial port. The only part that need to be configure is the serial port in the host part (left part). It is necessary to adapt to the serial port available at the ARM device: ttyAMA0 in rpi and ttyS2 for odroid (check section 1.2.2 for more details).

3.2.2 Dependencies with railhome

One of the eastern eggs provided by the railpic project is the possibility of running a docker that automatically synchronize to a git server the config files in used. It is of interest when the /config folder contains all the information related to a specific layout – not only related to the rpicd docker.

To make use of the railhome docker, the docker-compose.yml described above must be adapted to cinldue the following two parameters:

```
    depends_on:
      - "railhome"


    environment:
      - WORKDIR=/config      #It should match the mapping part of the docker (second
part) in the volume define above
      - RAILHOME=yes          #To sync with a git repository -see railhome images.
```

Then, two launch the railhome docker, it is necessary to proceed in this way:

1. Go to a new folder and create a docker-compose.yml file.

```
cd ~
mkir railhome-docker
```

```
cd railhome-docker
nano docker-compose.yml
```

1. Copy the following parameters into docker-compose.yml (selecting arm6 or arm7) services:

```
  railhome:
    container_name: railhome
    image: raildocker/railhome:arm7
    volumes:
      - config:/config
#First part should match the named volume at the end of the file,
#Second part (the docker part) should match the WORKDIR ENVRIRONMENT at the environment section
#This folder should be the same as the one set in rpicd as config folder.

      - /home/$USER/.ssh/id_rsa:/root/.ssh/id_rsa       #User launching the container must have previous access to the git repository

#See https://docs.gitlab.com/ee/ssh/ for more details
#Uncomment to run on x86 directories
      - /usr/bin/qemu-arm-static:/usr/bin/qemu-arm-static

    environment:
      - WORKDIR=/config      #It should match the mapping part of the docker (second part) in the volume define above
      - GITTRUSTHOST=gitlab.com                                  #To generate the fingertprint on first run. It s recommendable to check the logs on the first run
      - GITURL=
#To update with the git repository you want to sync
      - GITUSER=
#user and mail for the git user
      - GITMAIL=
    restart: always
    stop_grace_period: 90s
volumes:
  config:         #it should match the volume defines on top.
     driver: local
```

The variables to be defined are GITTRUSTHOST, GITURL, GITUSER, GITMAIL, Note the stop_grace_period: 90s. When syncing with a git repository, it may take some time, and it is necessary that the docker service does not stop railhome before it finishes its task.

Currently railhome monitors the status of rpicd to see when it can start syncing. This is control with same semaphore files at WORKDIR. If you need to sync files from other control software, you may need to modify the entrypoint script to monitor new semaphore files - and be sure such control software generates the right fles (probably by creating your own init scripts.

5. Start the docker (it will be restarted automatically every time to turn on your arm device)

```
docker-compose up
```

3.3 Directly running rpicd

It is always possible to compile rpicd and run directly the software. I have kept the external dependences to the minimum – just wiringPi- in order to facilitate the adoption of the code to any kind of system.

3.3.1 Installing the WiringPi library

rpicd only requires the installation of the wiringPi library by Gordon Hendersons. There are binaries compiled for both RPI and ODROID, but in case of need, it is also possible to compile the library from source.

3.3.1.1 RPI

To obtain WiringPi:

```
sudp pacman -S wiringpi
```

3.3.1.2 ODROID (not tested)

To obtain WiringPi:

```
sudp pacman -S wiringc1
```

3.3.1.3 From Source

Source: https://projects.drogon.net/raspberry-pi/wiringpi/

1. To obtain WiringPi using GIT:

```
~~git clone git://git.drogon.net/wiringPi~~
git clone https://github.com/WiringPi/WiringPi.git
```

Unfortunately Gordon discontinued the wiringPI library because of some ungrateful users. It is necessary then to rely on some unofficial mirros.

2. To build/install there is a new simplified script:

```
cd wiringPi
./build
```

3.3.2 Compiling and Installing RPICD

The daemon and the source code are provided in git.

1. Download the last version of the code

```
cd ~
git clone --depth=1 https://gitlab.com/RailPIC/rpicd.git
```

2. compile the code:

```
cd rpicd
make
```

3. Optionally, install the daemon service unit. The script, will enable and start it.

```
sudo ./install
```

3.4 Kiosk option

The ñast eastern egg is the possibility of launching a browser as a kiosk to see the rpicd GUI interface. This will also facilitate a minimum configuration of X Server to launch other graphical instances.

Simply run the kiosk script provided with the rpicd code, and all dependences will be installed,

```
cd rpicd
sudo ./kiosk
```

4 Tuning up your ARM device

4.1 Installing some basic support tools

4.1.1 Sudo

Source: https://wiki.archlinux.org/index.php/Sudo#Using_visudo

The installation of sudo is quite easy… the configutation not so much:

```
su root
pacman -S sudo
```

Edit as root (su root) the sudoers file… which can only be edited with its own editor

```
visudo
```

The vi manual can be found here: http://www.washington.edu/computing/unix/vi.html

```
MOVING THE CURSOR
     h            left one space
     j            down one line
     k            up one line
     l            right one space

  i            insert text left of cursor
  x         delete carácter
ESC         go back to the editor  mode
:w                  save file
:q                  exit
```

Remove the comment and adjust the users alias:

```
User_Alias      ADMINS =alarm
```

Provide Access to all commands to this alias:

```
ADMINS ALL=(ALL) ALL
```

Save and exit.

4.1.2 Screen

The screen tool enables to recover an on-going ssh session. Quite useful if long compilations are required.

To install it, simply:

```
sudo pacman -S screen
```

To use it, simply run

```
screen
```

And to recover and ongoing session:

```
screen -r
```

4.2 General settings

4.2.1 Chaging default user and password

It is highly recommended to change the default passwords, both for the alarm user and root

```
passwd
```

4.2.2 Set a fix IP

Source https://wiki.archlinux.org/index.php/Systemd-networkd#Wired_adapter_using_a_static_IP

Edit /etc/systemd/network/eth0.network and change DHCP=yes with:

```
Address=192.168.2.41/24
Gateway=192.168.2.5
```

Reboot and ready

4.2.3 Change the name of your device

Source: http://www.simonthepiman.com/how_to_rename_my_raspberry_pi.php

In order to change the name to rpic (or anything you want) you need to edit the /etc/hostname file.

```
sudo nano /etc/hostname
```

which should look like the entry below

```
alarmpi
```

and change it to

```
rpic
```

4.2.4 Change welcome messahe

```
sudo nano /etc/motd
```

4.2.5 Local time and keuboard

```
timedatectl set-timezone Europe/Madrid
```

and for the keyboard, edit /etc/vconsole.conf:

```
KEYMAP=es
```

4.2.6 SSH autolongin

Source: http://www.windowstipspage.com/putty-auto-login-ssh-keys/

Generate Keys

(Below steps should be executed on the computer where ssh server is running.). Firstly, set the right permissions on the ***.ssh*** directory. This is the place where we store the generated keys. Run the following commands.

```
chmod 700 .ssh
chmod 600 .ssh/authorized_keys
```

If authorized_keys file does not exist, create it with the below command.

```
touch .ssh/authorized_keys
```

Now run the command ***ssh-keygen***. This will generate public and private keys. You will be prompted to enter some passphrase, you can leave it empty if you want.

```
[user@linux-pc ~]$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/home/user/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/user/.ssh/id_rsa.

Your public key has been saved in /home/user/.ssh/id_rsa.pub.

The key fingerprint is:

78:94:6d:83:66:78:81:89:89:59:fb:1a:aa:a7:c2:e8 user@linux-pc

[user@linux-pc ~]$
```

Move the public key to ***.ssh/authorized_keys*** file.

```
cat .ssh/id_rsa.pub >> .ssh/authorized_keys
```

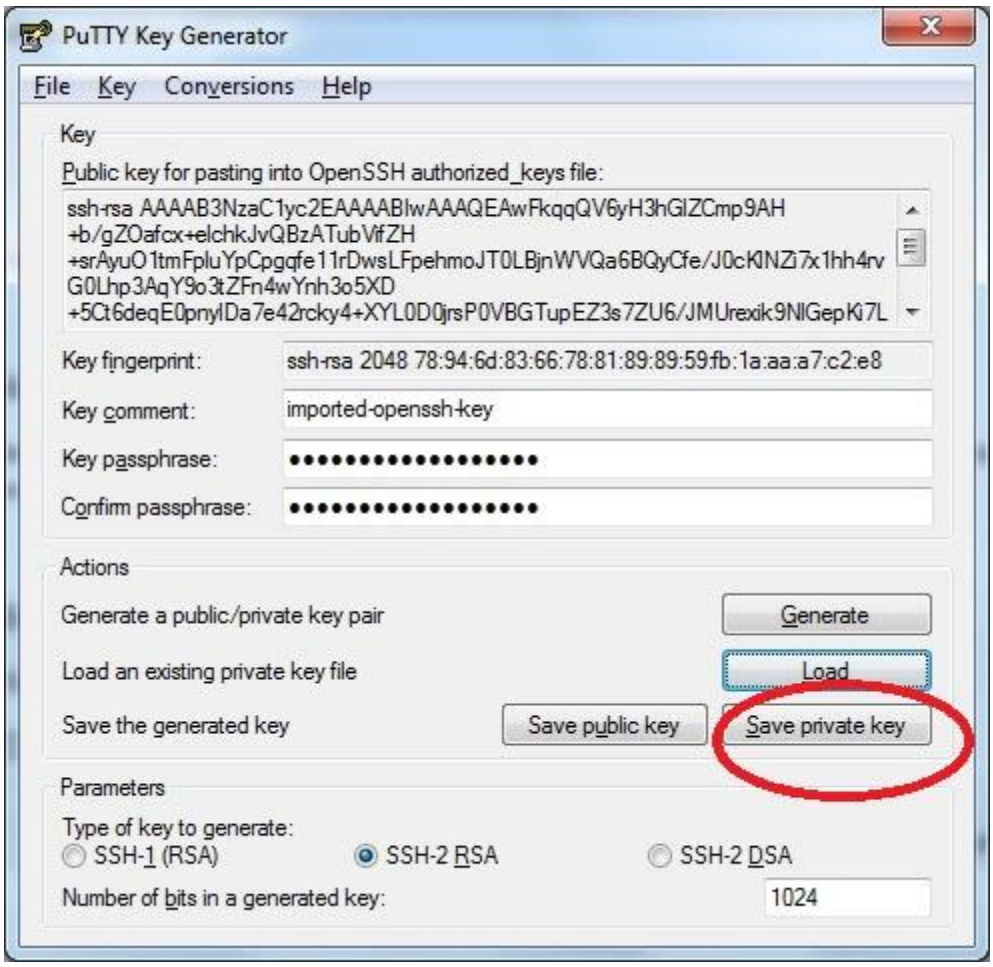
Make sure to append the key ('>>'), otherwise existing keys in the ***authorized_keys*** will be deleted.

Configure Putty Client

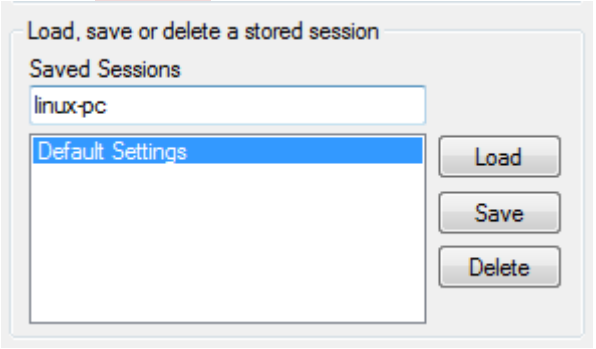
1. *Copy the private key generated on the server to the client computer from where you will be connecting to the server.*

2. *The private key we have is open-ssh key and putty does not understand this. We need to convert this private key to something that putty understands. For this we will use the tool PuttyGen. Click on the link to download this file.*
3. *Launch* **PuttyGen** *tool.*

4. *Click on the Load button, and select the private key file we copied from the server.*
5. *You will also need to enter the pass phrase you used for generating the keys in ssh-key command.*
6. *Save the generated private key using the '***Save private key***' button.*



7. *Now launch Putty tool.*
8. *Go to the node* **Connection** -> **Data** *. In the text box '* **Auto-login user name** *' enter your login name on the server*
9. *Go to the node* **Connection** -> **SSH** -> **Auth**
10. *Click on* **browse** *button and add the path for the private key file which we saved in step 6.*
11. *Now go to the '* **Session** *' in the settings. Enter a new name under '* **Saved sessions** *' and save it.*



12. *Now onwards whenever you want to connect to this server, you can select this session and lick on '* **open** *' button. If you have chosen empty passphrase in the key generation, ssh*

connection will be established automatically. Otherwise, you will be asked to enter the passphrase and then connection will be established.

4.2.7 Setting up the sound (rpi instructions)

Source:

http://elinux.org/R-Pi_Troubleshooting#Sound_does_not_work_at_all.2C_or_in_some_applications

In Wheezy the sound is set up by default and operates on the hdmi port. You may need to redirect the sound to the headphones output.

```
sudo amixer cset numid=3 1
```

To get the sound back to hdmi:

```
sudo amixer cset numid=3 2
```

And you can check if it is working:

```
sudo aplay /usr/share/sounds/alsa/Front_Center.wav
```

4.2.8 Increase the video signal at the hdmi port (rpi instructions)

Source: http://elinux.org/R-Pi_Troubleshooting#Interference_visible_on_a_HDMI_or_DVI_monitor

Some TVs and screens need it.

1. *Edit the rpi configuration file.*

   ```
   sudo nano /boot/config.txt
   ```

2. *Add the following line to the configuration file, replace de value already there with 4, or uncomment the line.*

   ```
   config_hdmi_boost=4
   ```

3. *Exit saving.*

   ```
   Press Control-x
   Press y
   Press [enter]
   ```

4. *After exiting the editor, restart using the command.*

   ```
   sudo reboot
   ```

4.2.9 Get rid of the big black borders around the image (rpi instructions)

Again, this is a typical problem with some TVs and screens.

1. *Edit the rpi configuration file.*

   ```
   sudo nano /boot/config.txt
   ```

2. *Add the following line to the configuration file, replace de value already there with 1, or uncomment the line.*

   ```
   disable_overscan=1
   ```

3. *Exit saving.*

   ```
   Press Control-x
   Press y
   ```

```
Press [enter]
```

4. After exiting the editor, restart using the command.

```
sudo reboot
```

4.2.10 Share a folder.

Source: https://wiki.archlinux.org/index.php/Samba

1. First you need to install all the needed samba libraries.

```
sudo pacman -S samba
```

2. Second step is to comfigure samba. Use the file hereafter

```
sudo nano /etc/samba/smb.conf
```

```
# This is the main Samba configuration file. You should read the
# smb.conf(5) manual page in order to understand the options listed
# here. Samba has a huge number of configurable options (perhaps too
# many!) most of which are not shown in this example
#
# Any line which starts with a ; (semi-colon) or a # (hash)
# is a comment and is ignored. In this example we will use a #
# for commentry and a ; for parts of the config file that you
# may wish to enable
#
# NOTE: Whenever you modify this file you should run the command "testparm"
# to check that you have not made any basic syntactic errors.
#
#======================= Global Settings =====================================
[global]
# THE LANMAN FIX
client lanman auth = yes
client ntlmv2 auth = no

# workgroup = NT-Domain-Name or Workgroup-Name
   workgroup = HOME

# server string is the equivalent of the NT Description field
   server string = RPIC samba

# This option is important for security. It allows you to restrict
# connections to machines which are on your local network. The
# following example restricts access to two C class networks and
# the "loopback" interface. For more examples of the syntax see
# the smb.conf man page
```

```
;   hosts allow = 192.168.1. 192.168.2. 127.

# if you want to automatically load your printer list rather
# than setting them up individually then you'll need this
;   printcap name = /etc/printcap
;   load printers = yes

# It should not be necessary to spell out the print system type unless
# yours is non-standard. Currently supported print systems include:
# bsd, sysv, plp, lprng, aix, hpux, qnx
;   printing = bsd

# Uncomment this if you want a guest account, you must add this to /etc/passwd
# otherwise the user "nobody" is used
;  guest account = pcguest

# this tells Samba to use a separate log file for each machine
# that connects
   log file = /var/log/samba/%m.log

# Put a capping on the size of the log files (in Kb).
   max log size = 50

# Security mode. Most people will want user level security. See
# security_level.txt for details.
   security = user
# Use password server option only with security = server
;   password server = <NT-Server-Name>

# Password Level allows matching of _n_ characters of the password for
# all combinations of upper and lower case.
;  username level = 8

# You may wish to use password encryption. Please read
# ENCRYPTION.txt, Win95.txt and WinNT.txt in the Samba documentation.
# Do not enable this option unless you have read those documents
;  encrypt passwords = yes
;  smb passwd file = /etc/samba/smbpasswd

# The following are needed to allow password changing from Windows to
# update the Linux sytsem password also.
# NOTE: Use these with 'encrypt passwords' and 'smb passwd file' above.
```

```
# NOTE2: You do NOT need these to allow workstations to change only
#        the encrypted SMB passwords. They allow the Unix password
#        to be kept in sync with the SMB password.
;  unix password sync = Yes
;  passwd program = /usr/bin/passwd %u
;  passwd chat = *New*UNIX*password* %n\n *ReType*new*UNIX*password* %n\n
*passwd:*all*authentication*tokens*updated*successfully*

# Unix users can map to different SMB User names
;  username map = /etc/samba/smbusers

# Using the following line enables you to customise your configuration
# on a per machine basis. The %m gets replaced with the netbios name
# of the machine that is connecting
;   include = /etc/samba/smb.conf.%m

# Configure Samba to use multiple interfaces
# If you have multiple network interfaces then you must list them
# here. See the man page for details.
;   interfaces = 192.168.12.2/24 192.168.13.2/24

# Configure remote browse list synchronisation here
#  request announcement to, or browse list sync from:
#                                                                    a specific host o
;   remote browse sync = 192.168.3.25 192.168.5.255
# Cause this host to announce itself to local subnets here
;   remote announce = 192.168.1.255 192.168.2.44

# Browser Control Options:
# set local master to no if you don't want Samba to become a master
# browser on your network. Otherwise the normal election rules apply
;   local master = no

# OS Level determines the precedence of this server in master browser
# elections. The default value should be reasonable
;   os level = 33

# Domain Master specifies Samba to be the Domain Master Browser. This
# allows Samba to collate browse lists between subnets. Don't use this
# if you already have a Windows NT domain controller doing this job
;   domain master = yes
```

```
# Preferred Master causes Samba to force a local browser election on startup
# and gives it a slightly higher chance of winning the election
;   preferred master = yes

# Use only if you have an NT server on your network that has been
# configured at install time to be a primary domain controller.
;   domain controller = <NT-Domain-Controller-SMBName>

# Enable this if you want Samba to be a domain logon server for
# Windows95 workstations.
;   domain logons = yes

# if you enable domain logons then you may want a per-machine or
# per user logon script
# run a specific logon batch file per workstation (machine)
;   logon script = %m.bat
# run a specific logon batch file per username
;   logon script = %U.bat

# Where to store roving profiles (only for Win95 and WinNT)
#        %L substitutes for this servers netbios name, %U is username
#        You must uncomment the [Profiles] share below
;   logon path = \\%L\Profiles\%U

# All NetBIOS names must be resolved to IP Addresses
# 'Name Resolve Order' allows the named resolution mechanism to be specified
# the default order is "host lmhosts wins bcast". "host" means use the unix
# system gethostbyname() function call that will use either /etc/hosts OR
# DNS or NIS depending on the settings of /etc/host.config, /etc/nsswitch.conf
# and the /etc/resolv.conf file. "host" therefore is system configuration
# dependant. This parameter is most often of use to prevent DNS lookups
# in order to resolve NetBIOS names to IP Addresses. Use with care!
# The example below excludes use of name resolution for machines that are NOT
# on the local network segment
# - OR - are not deliberately to be known via lmhosts or via WINS.
; name resolve order = wins lmhosts bcast

# Windows Internet Name Serving Support Section:
# WINS Support - Tells the NMBD component of Samba to enable it's WINS Server
;   wins support = yes

# WINS Server - Tells the NMBD components of Samba to be a WINS Client
```

```
#                                                                       Note: Samba can b
;   wins server = w.x.y.z

# WINS Proxy - Tells Samba to answer name resolution queries on
# behalf of a non WINS capable client, for this to work there must be
# at least one                                                          WINS Server on th
;   wins proxy = yes

# DNS Proxy - tells Samba whether or not to try to resolve NetBIOS names
# via DNS nslookups. The built-in default for versions 1.9.17 is yes,
# this has been changed in version 1.9.18 to no.
   dns proxy = no

# Case Preservation can be handy - system default is _no_
# NOTE: These can be set on a per share basis
;  preserve case = no
;  short preserve case = no
# Default case is normally upper case for all DOS files
;  default case = lower
# Be very careful with case sensitivity - it can break things!
;  case sensitive = no

#============================ Share Definitions ==============================
;[homes]
;   comment = Home Directories
;   browseable = no
;   writable = yes

# Un-comment the following and create the netlogon directory for Domain Logons
; [netlogon]
;   comment = Network Logon Service
;   path = /home/netlogon
;   guest ok = yes
;   writable = no
;   share modes = no


# Un-comment the following to provide a specific roving profile share
# the default is to use the user's home directory
;[Profiles]
;    path = /home/profiles
;    browseable = no
```

```
;    guest ok = yes


# NOTE: If you have a BSD-style print system there is no need to
# specifically define each individual printer
#[printers]
#   comment = All Printers
#   path = /var/spool/samba
#   browseable = no
# Set public = yes to allow user 'guest account' to print
#   guest ok = no
#   writable = no
#   printable = yes

# This one is useful for people to share files
;[tmp]
;   comment = Temporary file space
;   path = /tmpuu
;   read only = no
;   public = yes

# A publicly accessible directory, but read only, except for people in
# the "staff" group
;[public]
;   comment = Public Stuff
;   path = /home/samba
;   public = yes
;   read only = yes
;   write list = @staff

# Other examples.
#
# A private printer, usable only by fred. Spool data will be placed in fred's
# home directory. Note that fred must have write access to the spool directory,
# wherever it is.
;[fredsprn]
;   comment = Fred's Printer
;   valid users = fred
;   path = /homes/fred
;   printer = freds_printer
;   public = no
;   writable = no
```

```
;   printable = yes

# A private directory, usable only by fred. Note that fred requires write
# access to the directory.
[railhome]
   comment = ROCHOME conf folder
   path = /home/alarm/railhome
   valid users = alarm
   public = no
   writable = yes
   printable = no
   browseable = yes
   create mask = 777
   guest ok = no
   read only = no
# a service which has a different directory for each machine that connects
# this allows you to tailor configurations to incoming machines. You could
# also use the %u option to tailor it by user name.
# The %m gets replaced with the machine name that is connecting.
;[pchome]
;  comment = PC Directories
;  path = /usr/pc/%m
;  public = no
;  writable = yes

# A publicly accessible directory, read/write to all users. Note that all files
# created in the directory by users will be owned by the default user, so
# any user with access can delete any other user's files. Obviously this
# directory must be writable by the default user. Another user could of course
# be specified, in which case all files would be owned by that user instead.
;[public]
;   path = /usr/somewhere/else/public
;   public = yes
;   only guest = yes
;   writable = yes
;   printable = no

# The following two entries demonstrate how to share a directory so that two
# users can place files there that will be owned by the specific users. In this
# setup, the directory should be writable by both users and should have the
# sticky bit set on it to prevent abuse. Obviously this could be extended to
# as many users as required.
```

```
;[myshare]
;   comment = Mary's and Fred's stuff
;   path = /usr/somewhere/shared
;   valid users = mary fred
;   public = no
;   writable = yes
;   printable = no
;   create mask = 0765
```

3. Set user and password

```
sudo pdbedit -a -u alarm
smbpasswd alarm
```

4. Restart/enable samba

```
sudo systemctl enable smb
sudo systemctl start smb
```

4.2.11 Clean some of the daemons starting with the system, so it starts up quicker (rpi instructions)

Source: http://theos.in/desktop-linux/removing-unwanted-startup-debian-files-or-services/

In order to speed up the boot process as much as possible, you can deactivate some daemons. An easy way to do it is through the rcconf tool.

```
sudo apt-get install rcconf
```

The following list of daemons is candidate to be removed… I have not tested, so feedback is welcome:

- Lightdm. If no desktop is going to be used, it seems it could be removed. However, I have no idea on the effect when trying to launch a stand-alone rocview windows.
- Motf. I would assume, removing this will only affect the message of the day… but I have not tested.
- Rsync. As far as I know, this is not in use… unless being used for updates of the rpi.
- X11-common. Again, in case of not planning to use rocview.

4.2.12 Backup your system (rpi instructions)

Sources: http://www.raspberrypi.org/phpBB3/viewtopic.php?f=29&t=12540
http://www.raspberrypi.org/phpBB3/viewtopic.php?f=29&t=10543
http://doc.ubuntu-es.org/Clonar_discos_duros
http://bytelchus.com/clonar-discos-con-cualquier-linux-con-comando-dd/

Easiest way from windows is to make use of the same tool used to install the rpi image in your sd card: Win32DiskImager. This time, instead of writing, just read.

It may take some time, depending on the size of your card.

The only problem with this solution is that it is actually making a backup of the entire card (even then non formatted clusters). This means that if you are not using the entire sd-card – quite usual if you want to keep compliance between cards – you cannot use this method.

For instance, two 8GB cards may not have the exact same size. In order to keep the backups interchangeable, it is mandatory to backup always to the smallest sieze of the SD cards… and this cannot be done with Win32DiskImage. The solution is on Linux:

1. Check for the SD card partition

```
df
```

You will get something like this:

```
/dev/sda1       37024320 19763972  15534896  56% /
none                   4        0         4   0% /sys/fs/cgroup
udev              503204        4    503200   1% /dev
tmpfs             102556     1144    101412   2% /run
none                5120        0      5120   0% /run/lock
none              512760       76    512684   1% /run/shm
none              102400       20    102380   1% /run/user
/dev/sdb1       20510332    44992  19400432   1% /media/agonia/TEMPORAL
/dev/sdc2        7425512  5637320   1393100  81% /media/agonia/f10ba0bd-17e0-
4800-b0af-19bb2ed45acd
/dev/sdc1          57288    19072     38216  34% /media/agonia/9DCF-4197
```

The SD card in this case is sdc (the two partition)

2. *Make a copy*

```
dd if=/dev/sdc of=partitionimage.dd
```

3. *Restore a copy with:*

```
dd if=partitionimage.dd of=/dev/sdc
```

There could be some errors as result.. do not panic, the copy is well done, the erros may arise when copying from the big to the small SD card.

4.2.13 Recover a lost password (rpi instructions)

Soure: http://rpi.tnet.com/project/faqs/resetpasswd

*Put the card in another machine and edit cmdline.txt on the fat partition. Add **init=/bin/sh** to the end of the line of text that it already has and save the file.*

Example

Before it might look like:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1
root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```

After you add the command it will look like:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1
root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait init=/bin/sh
```

Put the card back in the Pi and apply power. The system should start booting and then pretty quickly drop you at a command prompt.

Reset the pi User password

At this command prompt type:

```
mount -o remount,rw /
passwd pi
(enter a new password)
sync
exec /sbin/init
```

The boot process will then continue and then pretty quickly drop you at a command prompt again.

*Re-edit the cmdline.txt file in /boot and now remove the **init=/bin/sh** from cmdline.txt.*

From that point on, when you reboot you should be able to use the new password you set for the pi user.

4.2.14 SWAP

https://wiki.archlinux.org/index.php/Swap

For some action (as get the entire git repository). Small devices as the first versions of the RPI may have some memory problems. In those cases it is necessary to create a reasonable swap partition

```
su root
fallocate -l 1G /swapfile
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
```

and `nano /etc/fstab`

```
/swapfile none swap defaults 0 0
```

4.3 Specific settings for my layout

Hereafter I include specific customization for my layout.

4.3.1 Two IPs for Loconet Ethernet bridge

Source: http://www.nish.com/2007/12/how-to-set-multiple-ips-on-debian/

To replace the s88 bus for sensors in the layout, I use Loconet, and specifically the GCA101(MGV101) board designed by Peter Giling and Fred Jansen (http://wiki.rocrail.net/doku.php?id=mgv101-en).

The manual of this board explains the benefits of having the Loconet Bridge in a segregated subnet. The instructions to get multiple IPs on debian can be found hereafter (you will need a fix IP a explained in section 4.2.1):

1. *Backup the current version of the interfaces file.*

```
sudo cp /etc/network/interfaces /etc/network/interfaces.sav
```

2. *Edit the file.*

```
sudo nano /etc/network/interfaces
```

3. *Add the following lines.*

```
auto eth0:0
iface eth0:0 inet static

address 192.168.0.10
netmask 255.255.255.0
broadcast 192.168.0.255
```

4. *Restart the network.*

```
sudo /etc/init.d/networking restart
```

4.3.2 Touchscreen

Fortunately the Packard Bell Viseo200T I am using is already recognized by debian distro installed in rpi. Thus, there is no need for any hard work.

However, in order to fine tune the screen. There are a number of things that can be done.

4.3.2.1 Install a touch keyboard

I am using the following one: http://homepage3.nifty.com/tsato/xvkbd/#mainmenu

First thing to do, install it:

```
sudo apt-get install xvkbd
```

Second thing, configure your keyboard layout by default.

Source: http://forum.tinycorelinux.net/index.php?topic=2189.0

```
cat /etc/X11/app-defaults/XVkbd-spanish >> .Xdefaults
```

To launch it, you only need to:

1. *Call it from a graphical session, from a terminal.*

   ```
   xvkbd
   ```

2. *Launch it from HW.*
   WIP: The idea is to launch it and completement the HW launch of rcview.. but I do not know if it would be possible
3. *Launch it from Rocview.*
   WIP:
   Launching it from rocview does not work. I have created a button launching the external program, and I get an error related to not having access to graphical.
   http://forums.wxwidgets.org/viewtopic.php?f=1&t=20228

4.3.2.2 Emulate right click

WIP: http://www.touch-base.com/documentation/Virtual%20Keyboards.htm
http://www.conan.de/touchscreen/evtouch.html
http://forum.xbmc.org/showthread.php?tid=137852
http://home.eeti.com.tw/web20/eg/about_EETI.html

4.3.3 Video tuner

WIP

4.3.4 Speech Synthesis

Source: http://elinux.org/RPi_Text_to_Speech_(Speech_Synthesis)

http://wiki.rocrail.net/doku.php?id=text-en

Please check first you are capable of reproducing sounds (section 4.2.3). Then you need to follow these steps:

1. *Install a player*

   ```
   sudo apt-get install mplayer
   ```

2. *Sort out the mplayer error message, edit file /etc/mplayer/mplayer.conf using:*

   ```
   sudo nano /etc/mplayer/mplayer.conf
   ```

   Add a line:

   ```
   nolirc=yes
   ```

And here, we have two options:

a) Espeak
I. Install Espeak with:

```
sudo apt-get install espeak
```

II. Test Espeak with: Spanish female voice, emphasis on capitals (-k), speaking slowly (-s) using direct text:-

```
espeak -ven+f3 -k5 -s150 "I've just picked up a fault in the AE35 unit"
```

b) Google (internet connection required)

I. Create a file sonido.sh (the same that will be used later on by rocrail) with:

```
cd /opt/rocrail
sudo nano speech.sh
```

II. Add these lines to the file and save it (in nano editor use CTRL-O writeOut)

```
#!/bin/bash
say() { local IFS=+;/usr/bin/mplayer -ao alsa -really-quiet -
noconsolecontrols
"http://translate.google.com/translate_tts?tl=en&q=$*"; }
say $*
```

III. Add execute permissions to your script with:

```
chmod u+x speech.sh
```

IV. Test it using:

```
./speech.sh Hola Mundo.
```

Personally, I prefer the google option.. it sounds better. Now, you only need to follow the same process as described in rocrail forum to play with the text.

4.3.5 Git Only the required files

Some repositories are quite large – as the rpicd repository – and we only need to use part of the files at the reposioty.

Source: https://askubuntu.com/questions/460885/how-to-clone-git-repository-only-some-directories

With git>=1.7

```
git init RPICD
cd RPICD
git remote add -f origin <url>

git config core.sparseCheckout true

echo "rpicd/" >> .git/info/sparse-checkout
echo "rpic/sw/rpic/commands.h" >> .git/info/sparse-checkout
git pull --all
```

5 Control Software

5.1 Rocrail

I normally use windows and eclipse to compile rocrail. I was not so familiar with linux, so the instructions hereafter are mainly lessons learnt when trying to get rocrail running.

I decided to go the "easiest way" and compile directly on the ARM device. Cross-compiling from linux seems to be a standard procedure for the people developing for the raspberry pi, so it should be to achieve it … but I do not know how.

5.1.1 Compiling Rocrail

Source: http://wiki.rocrail.net/doku.php?id=build-en

This is also quite easy with the information in the rocrail web site:

1. *Request access to the sources in the Rocrail Forum.*
2. *Download the sources.*

```
git clone <granted link> Rocrail
```

3. *Make the files.*

```
make fromtar
```

4. *Install as root.*

```
su root
make install
```

*Rocrail is installed in **/opt/rocrail**.*
Alternatively, if you want to build exclusively the server, you now have a nice feature:

1. *Make the files.*

```
make server
```

5.1.2 Running Rocrail

5.1.2.1 Preparing the environment

5.1.2.1.1 Tune-up the rocrail.ini file (in case of problem)

Since v4177, it seems mandatory to provide a path to the libraries for the centrals (included ddx). If you get into this problem, you can easily detect it, since you will get errors opening ddx.so and in the clients you will receive a message "IID not set".

```
cd /opt/rocrail
sudo nano rocrail.ini
```

Replace

```
libpath=""
```

by

```
libpath="/opt/rocrail/"
```

5.1.2.2 Launching rocrail

Running rocrail should be easy at this point. Just remember it has to be run with root privileges.

```
cd /opt/rocrail/
```

```
sudo ./rocrail
```

I have tried to run it from another folder, but I always get errors related to dependences with other files. Since in the typical setting, you will be running rocrail as a daemon you will not usually need to follow the above procedure.

5.1.2.3 Running rocrail as a daemon

The daemons in arch linux works differently to raspbian.

Source: https://wiki.archlinux.org/index.php/Systemd

I have created the following unit to start automatically rocrail:

```
[Unit]
Description= Rocrail Service
Requires=network.target

[Service]
User=root
WorkingDirectory=/opt/rocrail
ExecStart=/opt/rocrail/rocrail
PIDFile=/opt/rocrail/rocrailpid

[Install]
WantedBy=multi-user.target
```

It has to be places in /etc/systemd/system. And then enable and star t as any other service in arch linux. The file can be found at the rpicd repository (daemons folder)

```
sudo systemctl start rocrail.service
sudo systemctl enable rocrail.service
```

To stop the service, stop is the key work (instead of start).

5.1.2.4 Customizing the daemon

This section is not necessary to run the daemon. The configuration hereafter serve to make the daemon look like the rest of processes in startup, and to get a copy of rocrail trace in a second terminal (tty2).

5.1.2.4.1 Colors and messages

If you see the boot sequence of the rpi, you will see that all services are marked with an [OK], [fail],[info], etc. tag, depending if they are launched correctly. In order to get this feature, you would need to edit the rocraild file in /etc/init.d/.

Note that the following code is just to make the message look nice. Adapting the daemon to the debian template would probably require more modifications:

1. *Open the file.*

   ```
   sudo nano /etc/init.d/rocraild
   ```

2. *Include the following line after the header to get access to log functions:* ***. /lib/lsb/init-functions*** *It should look like this.*

   ```
   ## END INIT INFO
   ```

```
. /lib/lsb/init-functions

rocraild_BIN=/opt/rocrail/rocrail
```

3. Then replace the echo messages with the log messages.
From:

```
if [ ! -x $rocraild_BIN ] ; then
        echo -n "Rocrail not installed ! "
        exit 5
fi

case "$1" in
    start)
        if [ ! -e $rocraild_PID ] ; then
                echo "Starting Rocrail"
        else
                echo "rocraild.pid already exists ! "
                exit 5
        fi
        su - root -c "$rocraild_SH"
        ;;
    stop)
        if [ -e $rocraild_PID ] ; then
                echo "Shutting down Rocrail"
        else
                echo "Rocrail not running or missing PID File ! "
                exit 5
        fi
        su - root -c "kill `head $rocraild_PID`"
        su - root -c "rm $rocraild_PID"
        ;;
*)
```

To:

```
if [ ! -x $rocraild_BIN ] ; then
        log_daemon_msg "Rocrail not installed !" "rocrail"
        log_end_msg 1
        exit 5
fi

case "$1" in
    start)
        log_daemon_msg "Starting Rocrail" "rocrail"
```

```
        if [ ! -e $rocraild_PID ] ; then
                su - root -c "$rocraild_SH"
        else
                log_end_msg 1
         echo "rocraild.pid already exists ! "
                exit 5
        fi
        log_end_msg 0
        ;;
    stop)
        log_daemon_msg  "Shutting down Rocrail" "rocrail"
        if [ -e $rocraild_PID ] ; then
        su - root -c "kill `head $rocraild_PID`"
        su - root -c "rm $rocraild_PID"
     else
                log_end_msg 1
         echo "Rocrail not running or missing PID File ! "
                exit 5
        fi
        log_end_msg 0
        ;;
*)
```

5.1.2.4.2 Restart option

In order to restart the daemon – useful for the reset button, it is necessary to modify the rocrail daemon:

1. *Open the file.*

```
sudo nano /etc/init.d/rocraild
```

2. *Include this lines at the end, just between the last ;; and the *)*

```
   reset)
        su - root -c  "/etc/init.d/rocraild stop"
        sleep 2
        su - root -c "/etc/init.d/rocraild start"


;;
```

5.1.2.4.3 Copy of nohup in tty2

I want to use the second terminal (tty2) as screen to check the daemon operations. There are several ways to do it, and it is even possible to activate/deactivate the output to tty2, but I have decided to make it a default monitoring screen. In this way, pressing CTRL+ALT+F2 it is possible to monitor the daemon (not start and stop, but see what it is doing).

Additionally, I am removing all messages when starting the daemon, even the ones related to nohup.out file.

1. *To get this last feature you need to edit the rocraild.sh file.*

```
sudo nano /opt/rocrail/rocraild.sh
```

2. Change the original file from.

```
#!/bin/bash
cd /opt/rocrail/
rm -f nohup.out
nohup ./rocrail -l /opt/rocrail&
echo "$!" > rocraild.pid
```

To:

```
#!/bin/bash
cd /opt/rocrail/
rm -f nohup.out
touch nohup.out
nohup ./rocrail -l /opt/rocrail >nohup.out 2>&1 &
echo "$!" > rocraild.pid
tail -f nohup.out >/dev/tty2&
```

5.1.2.5 Controlling the daemon from another computer

It is always possible to control the daemon from another computer through ssh, just open a session and use the start and stop commands in section 5.1.2.3.

*Additionally, if you want to see how the daemon is working, you can use the last command in the previous section (**tail -f nohup.out**).*

If you want to see what the daemon is actually sending to tty2, you need:

1. Install linuxvnc.

```
sudo apt-get install linuxvnc
```

2. Grab terminal 2 with linuxvnc.

```
sudo linuxvnc 2
```

3. Make use of your favorite vnc tool to check tty2 (port in linuxvnc by default is 5900.

5.1.3 Running Rocview

Source:
https://wiki.archlinux.org/index.php/Xinitrc#Starting_applications_without_a_window_manager
http://archlinuxarm.org/forum/viewtopic.php?f=31&t=2877

In order to run rocview, it is necessary to install

```
xorg-server
xorg-xinit
```

If you want to run rocview to the maximum size of your screen, you need to edit your rocview.ini file (example for 1080p)

```
<window x="0" y="0" cx="1920" cy="1080"/>
```

5.1.3.1 Set-up for raspberry pi

CHECK if the same solution as for odroid is valid

Additionally, for rpi it is needed to install:

```
xf86-video-fbdev
```

Then, it is necessary to copy a valid xinit conf. file:

```
cp /etc/X11/xinit/xinitrc ~/.xinitrc
```

And finally, if the fie es edited and included a final line with:

```
exec /opt/rocrail/rocview
```

Every time startx is called, rocview will be launched, without additional window manager

5.1.3.2 Set-up for odroid

Sources: https://unix.stackexchange.com/questions/85383/how-to-start-a-second-x-session

https://wiki.archlinux.org/index.php/Xinit

https://en.wikibooks.org/wiki/Guide_to_X11/Starting_Sessions

Additionally, for odroid it is needed to install:

```
xf86-video-odroid-c1
```

And it is possible to run wocview with

```
sudo xinit /opt/roctail/rocview
```

Notes:

1.This needs to be run as root. I have tried to run it as a different user, but I can not. It is possible to modify the /etc/X11/ Xwrapper.config and add `allowed_users = anybody` and `needs_root_rights = no`. But then the problem is with /tmp/.X0-lock.

2. Once it is running, if you close the app or kill it, the Xserver seems to crash. The workaround is to before getting out, change to another console (CTRL+ALT+Fn), and go back to the console with rocview (ALT+Fn). For some reason, this avoids the Xserver to crash. I have tried as work around to launch xinit with a script launching xterm and rocview, butt after exiting any of the two prgrams… it simply crashes.

```
exec /bin/xterm &
exec /opt/rocrail/rocview
```

3. With this minimal configuration starx does not work.

5.1.3.2.1 Running rocview from an SSH session

Source: https://askubuntu.com/questions/47642/how-to-start-a-gui-software-on-a-remote-linux-pc-via-ssh

From a ssh session :

```
export DISPLAY=:0
```

And then ans on previous section

```
sudo xinit /opt/roctail/rocview
```

The same notes apply. The advantage is that it is always possible to launch agan xinit.

5.1.3.3 Set-up for dockers

In addition to the dependencies explained before, it is necessary to install xorg-xhost and run the following command to enable xterminal from anywhere:

```
xhost +
```

The most easy option is actually to run the kiosk script for rpicd andit will create the necessary environment to launch rocview.

5.1.3.4 Launching rocview from hardware

WIP

The idea is to use one of the general purpose IO pins at the ARM device to activate a script with the above mentioned procedure to start rocview.

5.2 CS2

Apparently, it is not possible to run wine on ARM devices yet in order to launch CS2.exe (ref)

https://forum.odroid.com/viewtopic.php?f=52&t=5924
https://wiki.winehq.org/ARM#Running_Windows.2Fx86_Applications

But it is possible to run the server side and connect from a windows machine

6 Next steps

Some ideas to be check:

- *Use other GPIOs for other functions:*
  - *Make the install process for rpic simpler, creat a C program to do it and not rely on python scripts.*
  - *Capture through the Loconet the consumption of different boosters.*

7 Acknowledges

This guide has been created with information from different sources. I have tried to reference all of them within the text.