MiruRLB – Relay board

Revision 007 20160229

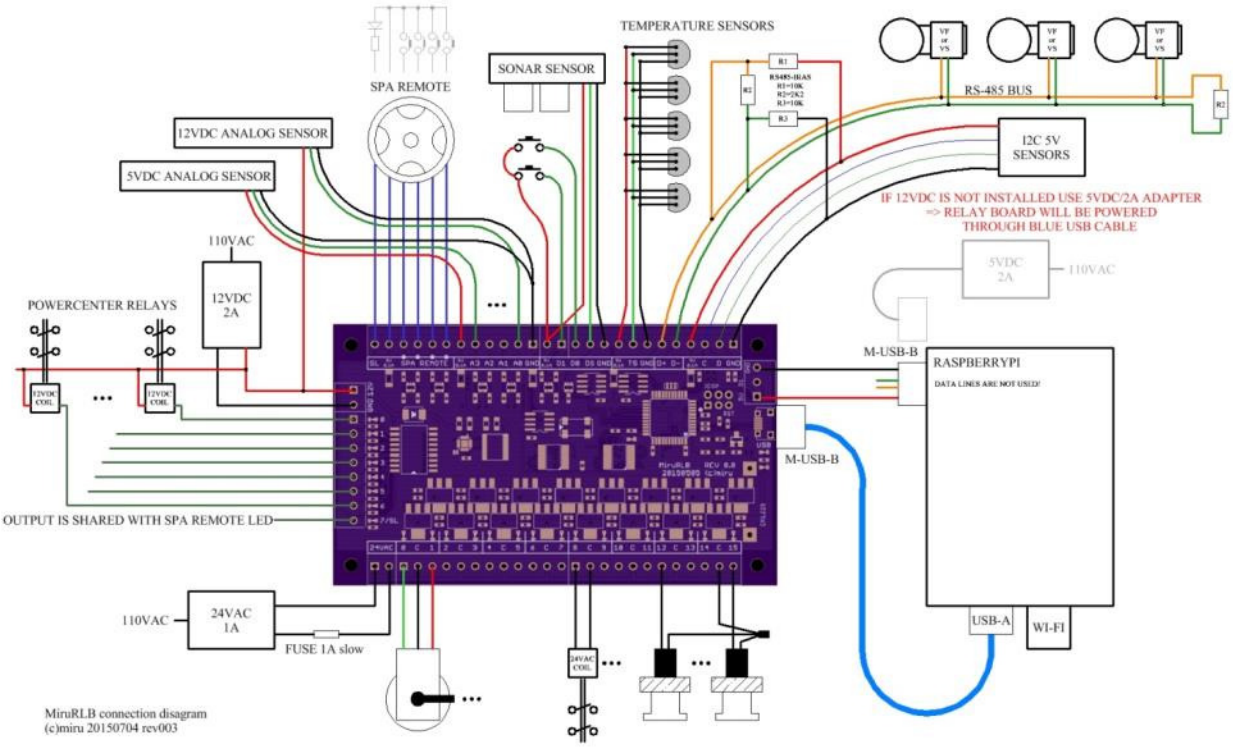


Table of Contents

Introduction 4

Hardware 5

- Power input/output 5
- Relay and Valve interface 5
- Spa remote interface 5
- Analog inputs 6
- Digital sensor terminals 6
- RS485 half-duplex interface 6
- I2C bus interface 6
- Header 6
- MOV installation 6
- Wiring diagram 7

Firmware 8

- Communication 8

Configuration 8
System status report 9
Commands 10
12VDC Relay status get/set 10
Controller clock 10
I2C bus command 10
Spa remote control LED status 11
Smart pump communication 11
Configuration parameter get/set 11
24VAC Relay status get/set 11
System status report 11
Scan one wire bus for temperature sensors 12
Valve status get/set 12
Software 13
Host 13
Linux installation on a RaspberryPi 13
Device node /dev/rlbXXXXXXXXXXXXXX 13
Server rlb 14
Client rlb 14
Log processor utility 'l2grlb' 15
Web interface 15
Multiple relay boards 15
Multiple board software concept 15
Multiple board how-to setup from scratch 16
Multiple board addressing 16

Introduction

Pool controllers are fairly complex and their manufacturers cloud their implementation by marking almost everything 'proprietary'. This insures high margins on fairly simple hardware and explains why pool controllers are difficult to integrate into whole house automation systems.

Today's market expects more open control systems in the sense of accessibility by computers, tablets or smart phones, which most 'sitting' manufacturers choose to ignore or implement by yet another piece of proprietary hardware and/or software.

The advent of the RaspberryPi, a full blown low cost ($35) Linux system on a small footprint circuit board, does provide the necessary environment to implement a pool controller that is as open and connectible as you make it. The only thing that it is missing is a versatile hardware interface to pool equipment, which requires a lot of I/O points.

The MiruRLB is a controller board which is meant to bridge the gap between physical hardware and software allowing alternate solutions for a pool controller. The intent is to handle any hardware related tasks, provide some local processing power to process events that have to be dealt with in a timely fashion and give the ability to deal with 'smart' pumps by communication.

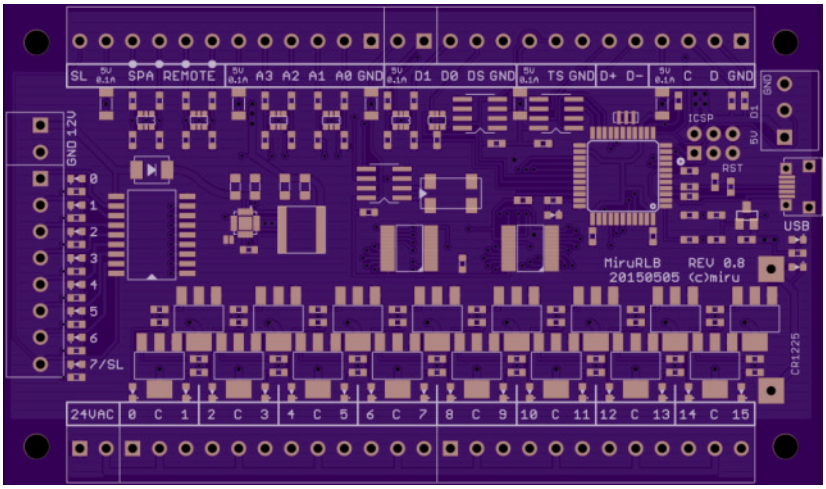
Pool valves, as well as sprinkler valves, run on 24VAC, the MiruRLB has outputs that can turn that voltage on or off under software control , provides sensor interfaces with translation of measurement results and 'knows' how to talk to 'smart' pumps. The current I/O features are:

- 16 24VAC relay outputs, that can be configured in pairs for pool valve operation
- 8 open collector outputs for high side 12VDC relays
- One wire bus interface for DS18x20 temperature sensors
- I2C master interface (5V) for sensor expansions
- 4 analog inputs for pressure sensors, PH probes or ORP probes
- 3 digital sensor inputs, one configured to handle a ping sensor for level measurements
- 1 RS485 interface, used to communicate with 'smart' pumps
- remote spa interface, it handles a 'standard' spa remote with 4 buttons and a LED, it is used in many pool/spa installations

The MiruRLB communicates with a host processor through USB. The host is envisioned to be a computer of some kind running one of the popular operating systems and capable of 'running the show' by means of one or more programs. It is expected that a program on the host can do all the interfacing with the Internet. The firmware of the MiruRLB is not Internet aware nor has access to resources to do it.

Hardware

Here is a look at a MiruRLB board of recent vintage:



Power input/output

The connector labeled 12V is the power feed for the board. The feed is used to power the voltage regulator for the processor and the power connector on the right side of the board. If the 12V power feed is not present, the processor is switched to USB power.

Relay and Valve interface

The connector on the left labeled 24VAC is the power feed for the relay outputs labeled 0 … 15 and C at the bottom of the picture. The firmware supports two addressing schemes for the 24VAC outputs. The outputs can be addressed as individual relays: r 0…15 or as valves: v 0…7. The 'vlm' configuration parameter of the firmware determines the addressing scheme.

valve	0		1		2		3		4		5		6		7	
relay	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Once an output pair is designated as a valve by means of the 'vlm' configuration variable, the firmware will never activate both outputs at the same time. Here is an example for vlm = 83 (hex):

valve	v 0		v 1												v 7	
relay					r 4	r 5	r 6	r 7	r 8	r 9	r 10	r 11	r 12	r 13		
output	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Spa remote interface

SPA REMOTE 0 is an input which activates when it is shorted to SP
SPA REMOTE 1 is an input which activates when it is shorted to SP
SPA REMOTE 2 is an input which activates when it is shorted to SP
SPA REMOTE 3 is an input which activates when it is shorted to SP

SL is expected to be the bottom end of a LED connected to SP
there is a power feed terminal for the remote, it supplies 5V/0.1A.

Analog inputs

Analog inputs are labeled A0 and A3. The inputs use 10K-10K resistors in series and a pair of diodes to protect the processor input.

Digital sensor terminals

Digital sensor terminals are labeled D1, D0 and DS. DS is the I/O for a 3 pin sonar level sensor.

RS485 half-duplex interface

The RS485 half-duplex bus interface pins are labeled D+ and D-. Depending on the installation there might be a requirement to bias the physical bus (see paragraph 'Wiring diagram'). If the bus cables are very long, one might have to use the 'official' termination with 120 Ohms on each end and pull up and pull down resistors of 523 Ohms. Most installations have short wires and the bus works well with 2K2 termination on each end or a single 1K termination at the relay board end and two 10K pull-up/down resistors. An idle bus is supposed to show a voltage of at least 0.270V between D+ and D-. This will ensure that all equipment connected to the bus will detect it as 'idle'.

I2C bus interface

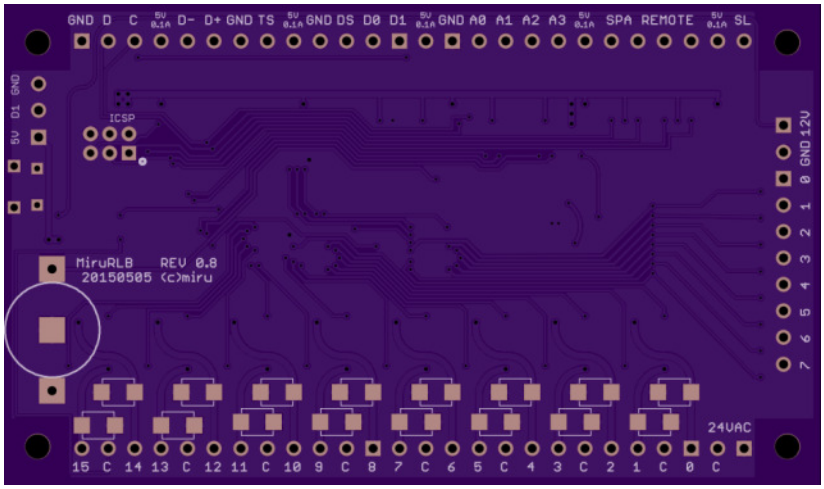
The I2C bus interface allows for extension of sensors that use its protocol.

Header

The 2x3 header location has an ICSP connector layout and is used to load a boot loader on the processor during manufacturing of the board. Once the boot loader is installed reprogramming of the firmware through the USB interface is supported.

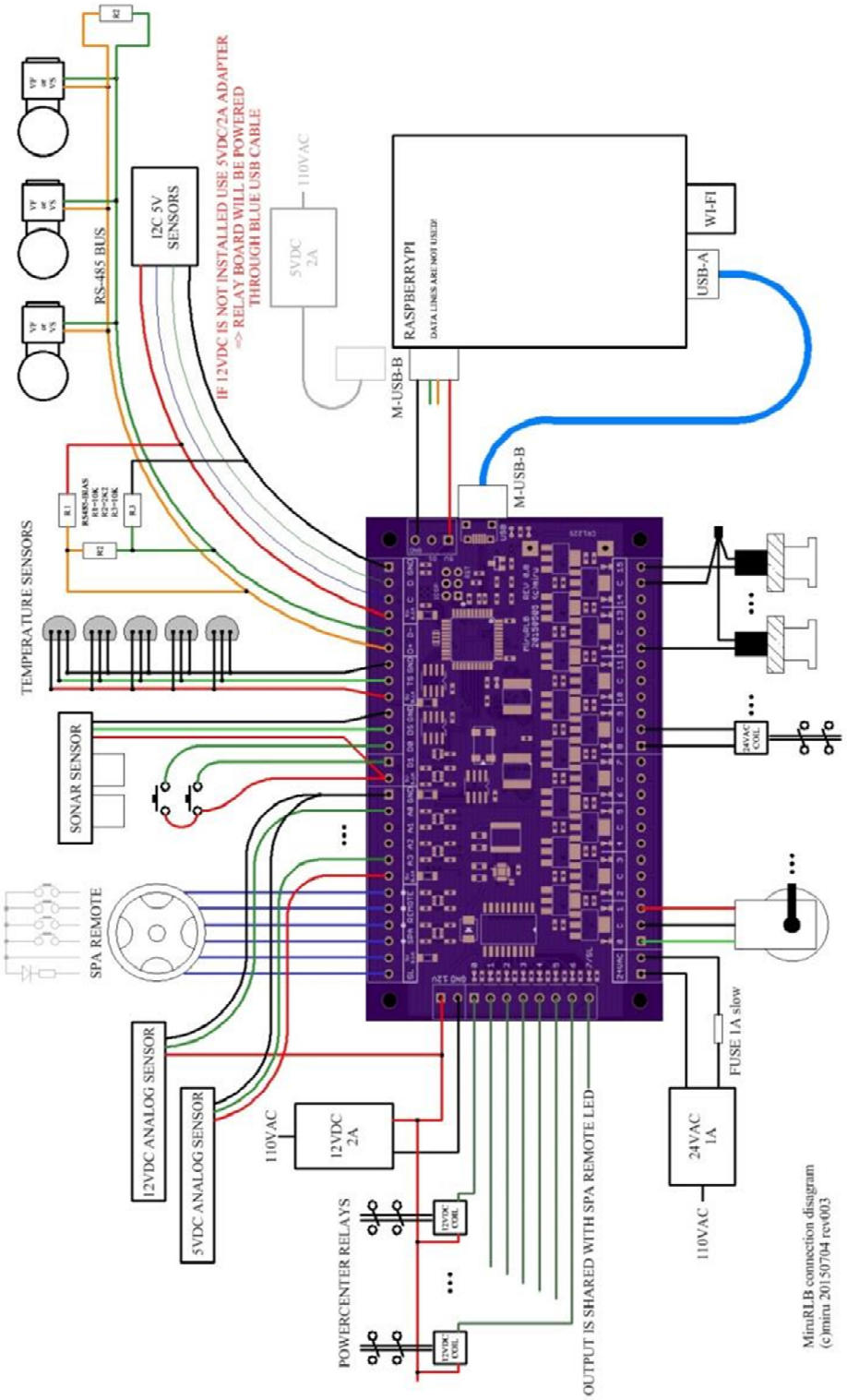
MOV installation

The backside of the board provides mounting points for MOVs in case they are needed.



Wiring diagram

Here is a diagram of the envisioned use of the board:



Firmware

Communication

The communication protocol to the host uses the ASCII character set. Records exchanged are variable length and terminated by a line feed '\n', a preceding '\r' is ignored. Data field separators are blanks ' ' or commas ','. The rationale behind this choice is to eliminate any big/little endian interpretation ambiguities as well as data field size problems. The data field order is implicit to cut down on the interpretation effort at the recipient side of a message.

Configuration

The MiruRLB maintains configuration parameters in EEROM.

Name	Default value	Description	Notes
vlm	83	Valve mask	Valves occupy a pair of relay outputs
vcf	00000000	Valve mask binary	Like vlm, but binary (0…7)
ctl	00 (hint: 14)	This controllers address	RS485 bus, hexadecimal address
Analog inputs			
ap0	0,10000	Offset and scale of analog input 0	Analog input, terminated with two 10K resistor in series
ap1	0,10000	Offset and scale of analog input 1	Analog input, terminated with two 10K resistor in series
ap2	0,10000	Offset and scale of analog input 2	Analog input, terminated with two 10K resistor in series
ap3	0,10000	Offset and scale of analog input 3	Analog input, terminated with two 10K resistor in series
aps	0,16000	Offset and scale of analog input S	Measures 12V power feed at bottom of resistor divider (D12->22K-M-10K-GND)
Temperature sensors			
tsu	F	Temperature reporting units	(Fahrenheit or Celsius)
ts0	MiruRLB ID	Temperature sensor address	Controller board sensor ID
ts1	000000000000	Temperature sensor address	hexadecimal
ts2	000000000000	Temperature sensor address	hexadecimal
ts3	000000000000	Temperature sensor address	hexadecimal
ts4	000000000000	Temperature sensor address	hexadecimal
ts5	000000000000	Temperature sensor address	Hexadecimal (sonar temperature)
srv	00 (hint: 30)	Time server address	RS485 bus, hexadecimal address
son	0.00	Sonar reference [inch]	US sensor reference
adj	0	Clock adjustment	Internal wall clock speed adjustment, negative values speed it up, positive values slow it down.

Note: greyed out parameters are 'hidden' parameters, which can be retrieved and set by name only.

System status report

The MiruRLB reports status every 30 seconds. At the point of this writing the status vector has the following fields:

Field	Name	Example	Description
0	ID	S	Record identifier
1	SEQ	023	Sequence number that runs from 000 … 255
2	MODE	0	Mode
3	DOW	0	Day of week (1– Sunday … 7 – Saturday)
4	WCK	23:59:59	Wall clock (24 hour clock)
5	RTC	5	RTC status
6	ERR	00	Hex error status Bit / Hex / Description 0 / 01 / One Wire Bus (temperature sensors) 1 / 02 / Wall clock 2 / 04 / Level measurement 3 / 08 / I2C Bus 4 / 10 / 24VAC feed 5 / 20 / TBA 6 / 40 / TBA 7 / 80 / TBA
7	VLV	00…..0	Valve status, a '.' indicates the valve is not configured
8	RLY	….0000000000..	Relay status, a '.' indicates the output is assigned to a valve
9	PCN	00000000	Power center status
10	AP0	0	Analog input 0
11	AP1	0	Analog input 1
12	AP2	0	Analog input 2
13	AP3	0	Analog input 3
14	APS	12565	DC supply voltage
15	TS0	69.4F	Temperature sensor 0 / Controller temperature
16	TS1	0F	Temperature sensor 1
17	TS2	0F	Temperature sensor 2
18	TS3	0F	Temperature sensor 3
19	TS4	0F	Temperature sensor 4
20	TS5	0F	Temperature sensor 5
21	ILV	0.00	Level measurement [inch] instant
22	RSC	000000	Remote Spa Control and digital input status
23	ALV	0.00	Level measurement [inch] average

Commands

The command interface to the firmware is not meant to be human friendly; it is designed to interpret commands from another computer. However, using ASCII commands enables human interaction. Most commands are lower case (exceptions are the B, V and R multi I/O point set commands).

12VDC Relay status get/set

Syntax:	**b [<#> [0|1]]**	
<#>	Relays are numbered 0 … 6. A relay output can have 2 different states. Relay 7 is shared with the LED output of the SPA remote control; it is manipulated through the 'l' command.	
0	Output off	
1	Output on	
Syntax:	**B [ss...]**	
Note:	This command can be used to set relays in parallel B 0000000 turns all relays off B 1111111 turns all relays on B ….0.0 turns relay 4 and 6 off The '.' is a place holder and means 'don't touch that output'	

Controller clock

Syntax:	**c <dow> <hour> <min> <sec> [<day> <month> <year>]**
<dow>	Day of week 1 Sunday 2 Monday 3 Tuesday 4 Wednesday 5 Thursday 6 Friday 7 Saturday
<hour>	0 … 23
<min>	0 … 59
<sec>	0 … 59
<day>	1 … 31
<month>	1 … 12
<year>	0 … 99
	Note: this command is rarely ever used directly and is intercepted when used with the rlb server. The rlb server has a 'C' command that will set the clock of the relay board to the system time of the rlb server host. A lower case 'c' command to the rlb server will print the clock offset and make a suggestion how to adjust the ADJ parameter in the configuration.

I2C bus command

Syntax:	**i [<adr> [<nrc> [<hex>]]]**
<adr>	Hexadecimal 7 bit I2C address
<nrc>	Number of bytes to read from device
<hex>	Data to write to device, use hexadecimal e.g. 010203ff would send 4 bytes 01 02 03 and ff

Spa remote control LED status

Syntax:	`l [<0|1|2>]`
0	Off
1	On
2	Blink

Smart pump communication

Syntax:	`m<#> <arg>`
<#>	Pump number 1…16. An 'm?' iterates through all 16 possible pump addresses and attempts to get status.
<arg>	argument

arg	Description
0	Stop pump
1…4	Run VS/VF pump External Control Program 1…4
15…70	Run VF pump with specified GPM
450…3450	Run VS pump with specified RPM
s	Get status

Note: The pump sequencer messages start with an uppercase 'M' followed by the pump number and a comma separated list of record fields. The second field identifies the record type, 'S' is a status report, 'E' is an error report.

Configuration parameter get/set

Syntax:	`p [<var> [arg [arg …]]`
<var>	See Configuration section for variable names
Note:	A 'p' command without arguments will output the complete parameter vector.

24VAC Relay status get/set

Syntax:	`r [<#> [0|1]]`	
<#>	Relays are numbered 0 … 15. A relay output can have 2 different states.	
0	Output off	
1	Output on	
Syntax:	`R [ss...]`	
Note:	This command can be used to set relays in parallel. The system in the example has 3 valves and therefore 10 relays: R ….0000000000.. turns all relays off R ….1111111111.. turns all relays on R ….0..0…… turns relay 4 and 7 off The '.' is a place holder and means 'don't touch that output'. Setting an output that is assigned to a valve is considered an error. '.'s at the end of the string don't need to be entered.	

System status report

Syntax:	`s`
	See Status Report section for field definitions

Scan one wire bus for temperature sensors

Syntax:	**t**
Note:	Example report of a system that has 2 sensors connected, one is assigned to temperature sensor slot 0, the other is unknown to the system: T,ts0,TS109D9B7A0208004D,26004B46FFFF0B10,66.3F

Note: to include the data of a temperature sensor into the system status vector, its address needs to be assigned to one of the TSx variables in the configuration.

Valve status get/set

Syntax:	**v [<#> [0|1|2]]**	
<#>	Valves are numbered 0 … (NVL-1). A valve output can have 3 different states.	
0	Output 1 is off Output 2 is off	A rather unconventional state for a pool controller, since it is expected that always one of the outputs is energized. If a valve output is in this state the valve cannot be operated by its local switch. This state can be used during power up of the controller to sequentially power the valves.
1	Output 1 is **on** Output 2 is off	
2	Output 1 is off Output 2 is **on**	
Syntax:	**V [ss...]**	
Note:	This command can be used to set valves in parallel. The system in this example has 3 valves, valve 0, valve 1 and valve 7: V 00…..0 turns the valves off (unusual state for pool controller) V 11…..1 activates output 1 for on each valve (‘normal’ state for a pool controller) V 21…..1 activates output 2 for valve 0 and output 1 for valve 1 and 7 Using the V command to set output bits that are not assigned to a valve is considered an error. The ‘.’ is a place holder and means ‘don’t touch that output pair’. Using the V command to set an output that is not assigned to a valve is considered an error. ‘.’s at the end of the string don’t need to be entered.	

Software

Host

The software for the MiruRLB is implemented as firmware running on the MiruRLB and a server program that needs to run on the host of the USB port the MiruRLB is connected to.

Linux installation on a RaspberryPi

The project started with a Raspberry Pi running 2015-02-16-raspian-wheezy.zip and migrated later to 2015-11-21-raspian-jessie-lite.zip. The .zip files contain images for the Micro SD of a RaspberryPi.
We use a Linux system to unzip the archive and to copy the result to a Micro SD:

```
sh> 2015-11-21-raspian-jessie-lite.zip
sh> sudo dd if=2015-11-21-raspian-jessie-lite.img of=/dev/MicroSD bs=4M
```

For a Windows host there are several 'free' programs that can be used to accomplish the same thing. Use the Windows built in 'Extract All…' to decompose the archive and something like Win32DiskImager to copy the image to the Micro SD.

Transfer the Micro SD to a RaspberryPi and power it up **without** Internet connection or a relay board. The lite version boots into a console interface. We usually disable the GUI since our systems do not have a monitor, keyboard or a mouse when they are deployed. We make it boot into console mode, set the hostname, time zone, expand the file system and reboot it. Then we plug it into the network so it has Internet access. This Debian distribution uses 'systemd' , which is not suited for the project, it terminates rlb servers started by the udev mechanism for some obscure reason.

The software for the MiruRLB comes as a .zip archive with the name rlbXXX.zip, where XXX stands for the revision of the archive. We log in as user 'pi' and place the archive in the home directory. Then the installation is done by unzipping the archive, changing directory and running the install script:

```
sh> unzip rlbXXX.zip
sh> cd rlbXXX
sh> sudo debian-jessie-lite-init.sh
sh> sudo ./install.rlb
sh> sudo reboot
```

Note: the first time around the system is not aware of the MiruRLB, subsequent installations will check and update the firmware of the MiruRLB with the help of 'avrdude'. This means, if you want to update the firmware on the MiruRLB now, the time is after the reboot. Change directory to rlbXXX and rerun the install script, no need to reboot this time.

Device node /dev/rlbXXXXXXXXXXXXXX

To Linux the MiruRLB is an USB device, it enumerates it as a serial device. The MiruRLB firmware emulates a device that is recognized by Linux as a ttyUSB…. To help identifying the MiruRLB the firmware uses a distinct 'bcdDevice' identifier of 0x0802. This can be detected by the udev system of Linux. We add a rule file to the system during installation that will map a MiruRLB to a specific device. The rules file is called '51-rlb.rules', the name follows the conventions of the udev system. You can use a different file name if your system is setup with a different paradigm. The rule file has two lines:

```
KERNEL==”ttyUSB*”, ATTRS{bcdDevice}==”0802”, SYMLINK+=”rlb$ATTR{serial},
RUN+=”/usr/local/sbin/rlbplg”, MODE=”0666”

KERNEL==”ttyACM*”, MODE=”0666”
```

This causes the plug in of a MiruRLB to create a symbolic link in /dev for the board and makes it accessible for regular users. The link is named /dev/rlbXXXXXXXXXXXXXX, with X.. being the serial number of the board. The script “rlbplg” is run by the udev system when a board is plugged in. It will launch an rlb server instance for each board.

The second line makes sure the boot loader of the board enumerates to a device that has access permissions for a regular user. This way a firmware update does not require super user privileges.

Server rlb

The rlb server/client comes as a precompiled binary for the RasperryPi. In a single relay board installation rlb as a server will write logfiles in directory /var/log/rlb if it exists and is read/write accessible by the application. rlb generates a log file for every day, the file names are formed from the date as yymmdd.log .

Client rlb

When the rlb server is running the rlb program can be used as a client to the server. This means, the following command would retrieve the status of the relay board if the server instance is running:

```
sh> rlb s
S,004,0,1……
```

Use the rlb client to issue commands to the relay board from a PHP web application. The distribution of the software has a web server example you can study and run to get a basic idea on how it is intended to work.

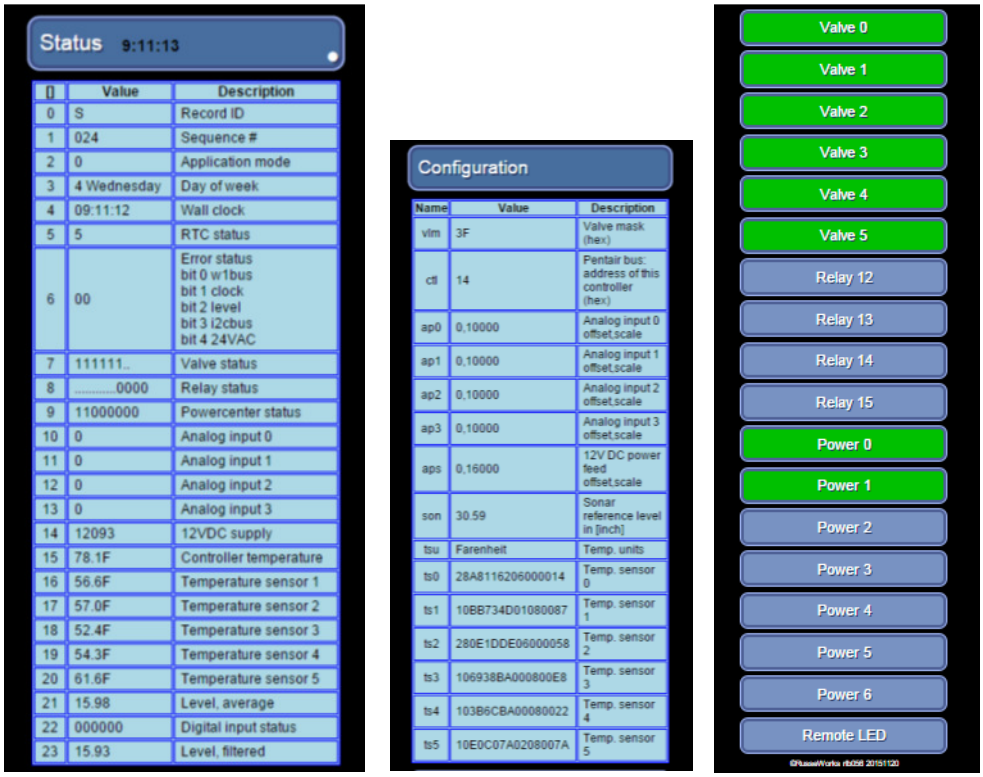
The rlb server has a UDP command port (13330) intended to be used by the client and/or a controller application. An application can sign up to receive all messages from the relay board as they occur. This is referred to as the ‘asynchronous’ interface.
The application creates a datagram socket and sends a two byte message (0x07 0x00) to the server port. The server will respond with a two byte message (0x06 <#>). From now on the server will relay any message from the relay board to that socket.
From time to time the server will check if a socket is still alive. To do that it will send a two byte message (0x05 0x00) and expects a two byte message back (0x06 0x00). If there is no reply to the 0x05 message, the server will drop the socket from its distribution list. To disconnect ‘orderly’ from the server distribution list send the following two bytes (0x7f 0x00), the server will respond with (0x06 0x00).
The server sends a (0x04 0x00) message to all sockets on its distribution list before it exits.
An application can send a (0x07 0x01) message to get on the distribution list for the next message from the relay board. The socket goes off the distribution list as soon as a message is sent. This is the equivalent of a ‘long poll’ request.

Log processor utility 'l2grlb'

The log processor program can generate data for charts. The data is output in JavaScript format. The program knows how to access the log files and can parse and extract particular data fields for a specific date/time range.

Web interface

The software distribution comes with an example web site that shows some possible aspects for the use of the MiruRLB. The installation script installs a link to the website for the local apache server. This way you can reference it from a browser by http://<host>/rlb . Here are some screen shots:



Multiple relay boards

The software supports multiple relay boards connected to the same host. This way one can easily extend the number of I/O points of a system.

Multiple board software concept

The board servers will be named 'rlb1', 'rlb2', 'rlb3' … opposed to just 'rlb' in a single board installation. To provide consistent mapping of a board to a particular server postfix the software uses a database contained in '/etc/rlb.cfg'. This database is created and modified by 'rlbmap' and consulted by 'rlbplg' when a new board is plugged into the system. The database is a map between a relay board serial number and a server postfix. You should use 'rlbmap' to modify '/etc/rlb.cfg', this way there is less pain.

Multiple board how-to setup from scratch

1) Power up the system **without** any relay board connected.
2) The command `sudo rm –f /etc/rlb.cfg` will remove an existing database if it exists
3) Plug in the relay board you consider #1 and issue the command `sudo rlbmap add 1` This will create the database and permanently map that particular board to 'rlb1' and restart the board to make the mapping effective.
4) You can check that it worked by issuing `rlb1 s` once the relay board has rebooted. If your command shell complains that it can't find the command issue `rehash` to force your command shell to reevaluate programs it can execute and try it again. The reason for this is that 'rlbmap' might just have created the command and the command shell still thinks it knows it all.
5) Plug in the relay board you consider #2 and issue the command `sudo rlbmap add 2` and do step 4) replacing 'rlb1' by 'rlb2'.
6) Keep going, the maximum index is 9 for now.

Multiple board addressing

The 'rlb' server /client examines with what name it got started. If it finds a postfix, it modifies its idea of port number to 13330+postfix. This way a server named 'rlb5' would provide service on port 13335 and a client named 'rlb5' will contact port 13335 for service.