



PICSimLab 0.8.9

Luis Claudio Gambôa Lopes <lcgamboa@yahoo.com>

Download: [github](#)

[HTML version of documentation](#)

June 19, 2021

Contents

1	Introduction	2
2	Install	4
2.1	Stable version executables download	4
2.2	Install from source	4
2.2.1	Linux	4
2.2.2	Windows	5
2.2.3	macOS	5
2.2.4	Experimental version	5
3	Simulator Interface	6
3.1	Main Window	6
3.2	Interaction with the Board	8
3.3	Command Line	8
3.4	Remote Control Interface	9
4	Boards	13
4.1	Breadboard	15
4.2	McLab1	15
4.3	K16F	16
4.4	McLab2	17
4.5	PICGenios	18
4.6	PQDB	20
4.7	Arduino Uno	20
4.8	Franzininho	21
5	Experimental Boards	22
5.1	Blue Pill	22
5.2	uCboard	22
5.3	gpboard	23
5.4	STM32 H103	23
5.5	X	24
5.6	Curiosity	24
5.7	Curiosity HPC	25

CONTENTS	2
5.8 Xpress	25
6 Serial Communication	27
6.1 Com0com Installation and Configuration(Windows)	27
6.2 tty0tty Installation and Configuration (Linux)	29
7 Debug Support	31
7.1 MPLABX Integrated Debug (picsim and simavr)	31
7.2 Arduino IDE Integration (simavr)	31
7.3 avr-gdb Debug (simavr)	32
7.4 arm-gdb Debug (qemu-stm32)	32
7.5 uCsim Debug	32
8 Tools	33
8.1 Serial Terminal	33
8.2 Serial Remote Tank	33
8.2.1 Sensors and Actuators	34
8.2.2 Communication Protocol	35
8.3 Esp8266 Modem Simulator	36
8.3.1 Supported Commands	37
8.4 Arduino Bootloader	37
8.5 MPLABX Debugger Plugin	37
8.6 Pin Viewer	37
9 Oscilloscope	38
10 Spare Parts	39
10.1 Inputs	42
10.1.1 Encoder	42
10.1.2 Gamepad	43
10.1.3 Gamepad Analogic	44
10.1.4 Keypad	44
10.1.5 MPU6050	46
10.1.6 Potentiometers	46
10.1.7 Potentiometers (Rotary)	47
10.1.8 Push Buttons	48
10.1.9 Push Buttons (Analogic)	48
10.1.10 Switchs	49
10.1.11 Ultrasonic HC-SR04	49
10.2 Outputs	50
10.2.1 7 Segments Display	50
10.2.2 7 Segments Display (Decoder)	50
10.2.3 Buzzer	51
10.2.4 DC Motor	51
10.2.5 LCD hd44780	52
10.2.6 LCD ili9341	53

CONTENTS	3
-----------------	----------

10.2.7 LCD pcf8833	54
10.2.8 LCD pcd8544	54
10.2.9 LCD ssd1306	55
10.2.10 LED Matrix	55
10.2.11 LEDs	56
10.2.12 RGB LED	57
10.2.13 Servo Motor	57
10.2.14 Step Motor	58
10.3 Others	59
10.3.1 ETH w5500	59
10.3.2 IO 74xx595	60
10.3.3 IO MCP23S17	61
10.3.4 IO PCF8574	61
10.3.5 IO UART	61
10.3.6 Jumper Wires	62
10.3.7 MEM 24CXXX	62
10.3.8 RTC ds1307	63
10.3.9 RTC pfc8563	63
10.3.10 SD Card	64
10.3.11 Temperature System	64
10.4 Virtual	65
10.4.1 D. Transfer Function	65
10.4.2 IO Virtual Term	65
10.4.3 Signal Generator	66
10.4.4 VCD Dump	66
10.4.5 VCD Dump (Analogic)	67
10.4.6 VCD Play	67
11 Troubleshooting	69
12 Use with MPLABX	70
12.1 Installing the Necessary Tools	70
12.1.1 Install MPLABX IDE and XC8 Compiler	70
12.1.2 Install PICsimLab	70
12.1.3 How to Install PicSimLab MPLABX Debugger plugin	70
12.2 Configuring a New Project in MPLABX	75
12.2.1 Project Creation	75
12.2.2 File Creation	78
12.2.3 PIC Configuration Bits	79
12.2.4 Code Example	80
12.2.5 Building the Project	81
12.3 Program and Debug PICsimLab With MPLABX	82
12.3.1 Starting PICsimLab	82
12.3.2 Programming PICsimLab	82
12.3.3 Pausing the Program	83
12.3.4 Restarting the Program	84

CONTENTS	4
12.3.5 Running Step by Step	84
12.3.6 Stopping Debugger	85
12.4 This Tutorial in Video	86
13 Creating New Boards	87
13.1 How to Compile PICsimLab	87
13.1.1 In Debian Linux and derivatives	87
13.1.2 Cross-compiling for windows	87
13.2 Creating a New Board	87
13.2.1 Board Hardware and Schematic	88
13.2.2 Board Picture	91
13.2.3 Picture maps	92
13.2.4 Board code	97
13.2.5 Integration with PICsimLab	111
13.2.6 Final Result	112
14 License	115

Chapter 1

Introduction

PICSimLab means Programmable IC Simulator Laboratory

PICSimLab is a realtime emulator of [development boards](#) with integrated MPLABX/avr-gdb debugger. PICSimLab supports some [picsim](#) microcontrollers and some [simavr](#) microcontrollers. PICSimLab have integration with MPLABX/Arduino IDE for programming the boards microcontrollers. As the purpose of PICSimLab is to emulate real hardware it does not have any source code editing support. For code editing and debugging the same tools used for a real board should be used with PICSimLab, such as MPLABX or Arduino IDE.

PICSimLab supports several devices (spare parts) that can be connected to the boards for simulation. As for example LEDs and push buttons for simple outputs and inputs and some more complex ones like the ethernet shield w5500 for internet connection or the color graphic display ili9340 with touchscreen. The the complete list of parts can be accessed in the chapter [Spare Parts](#).

The [experimental version boards](#) supports [uCsim](#), [gpsim](#) and [qemu-stm32](#) simulators in addition to the stable ones.



Chapter 2

Install

2.1 Stable version executables download

If you are on Linux or Windows you can download the last version at:

<https://github.com/lcgamboa/picsimlab/releases>

If you are on macOS you can run PICSimLab using Wine:

1. Download and install ['xquartz'](https://www.xquartz.org)
2. Download and install [Wine](https://dl.winehq.org/wine-builds/macosx/download.html)
3. Download the executable and double-click it to run the installer

2.2 Install from source

2.2.1 Linux

In Debian Linux and derivatives Linux native:

Using a user with permission to run the sudo command:

In first time build:

```
git clone --depth=1 https://github.com/lcgamboa/picsimlab.git
cd picsimlab
./picsimlab_build_all_and_install.sh
```

To recompile use:

```
make -j4
```

2.2.2 Windows

Cross-compiling for Windows (from Linux or [WSL](<https://docs.microsoft.com/windows/wsl/install-win10>) on win10)

In first time build in Debian Linux and derivatives target Windows 64 bits:

```
git clone https://github.com/lcgamboa/picsimlab.git
cd picsimlab
./picsimlab_build_w64.sh
```

To recompile use:

```
make FILE=Makefile.cross -j4
```

For target Windows 32 bits:

```
git clone https://github.com/lcgamboa/picsimlab.git
cd picsimlab
./picsimlab_build_w32.sh
```

To recompile use:

```
make FILE=Makefile.cross_32 -j4
```

2.2.3 macOS

macOS from source

Theoretically it is possible to compile PICSimLab natively on macOS. But I do not have access to any computer with macOS to try to compile and until today nobody has communicated that they managed to do it. (help wanted)

2.2.4 Experimental version

Experimental version

Experimental version can be built using the parameter "exp" on scripts:

```
./picsimlab_build_all_and_install.sh exp
./picsimlab_build_w64.sh exp
./picsimlab_build_w32.sh exp
```

And recompiled using the parameter "exp" on Makefiles:

```
make -j4 exp
make FILE=Makefile.cross -j4 exp
make FILE=Makefile.cross_32 -j4 exp
```

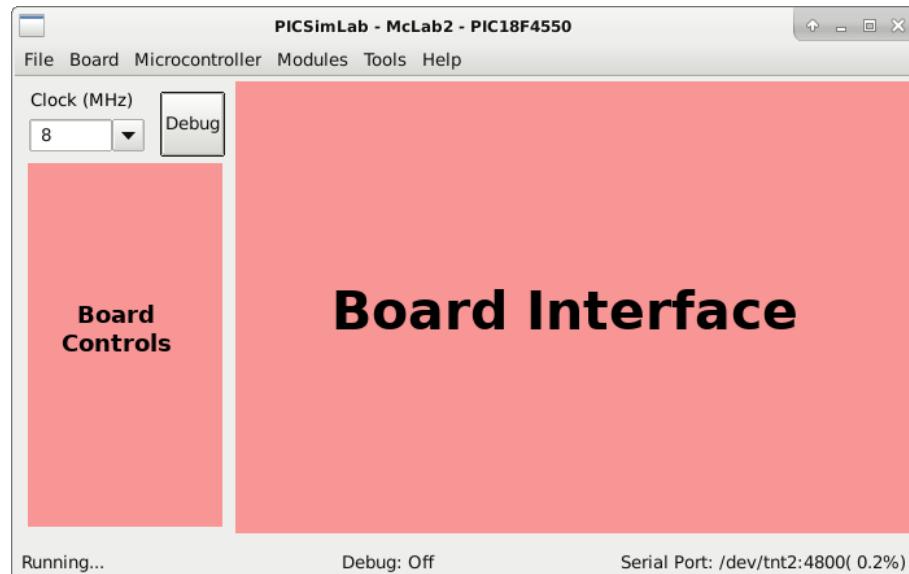
Chapter 3

Simulator Interface

3.1 Main Window

The main window consists of a menu, a status bar, a frequency selection combobox, an on/off button to trigger debugging, some board-specific controls and the part of the board interface itself.

In the title of the window is shown the name of the simulator PICSimLab, followed by the board and the microcontroller in use.



The frequency selection combobox directly changes the working speed of the microcontroller, when the “Clock (MHz)” label goes red indicates that the computer is not being able to run the program in real time for the selected clock. In this case

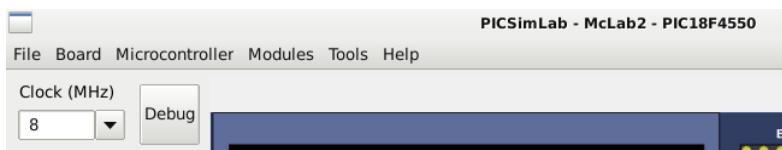
the simulation may present some difference than expected and the CPU load will be increased.

The on/off button to enable debugging is used to enable debugging support, with the active support there is a higher simulation load.

The menus and their functions are listed below:

- File
 - Load Hex - Load .hex files
 - Reload Last - Reload the last used .hex file
 - Save Hex - Save memory in a .hex file
 - Configure - Open the configuration windows
 - Save Workspace - Saves all current workspace settings to a .pzw file
 - Load Workspace - Loads saved settings from a .pzw file
 - Exit
- Board
 - Breadboard - Choose board Breadboard
 - McLab1 - Choose board McLab1
 - K16F - Choose board K16F
 - McLab2 - Choose board McLab2
 - PICGenios - Choose board PICGenios
 - Arduino Uno - Choose board Arduino Uno
- Microcontroller
 - xxxx - Selects the microcontroller to be used (depends on the selected board)
- Modules
 - Oscilloscope - Open the oscilloscope window
 - Spare parts - Open the spare parts window
- Tools
 - Serial Terminal - Open the serial terminal **Cutecom**
 - Serial Remote Tank - Open the **remote tank simulator**
 - Esp8266 Modem Simulator - Open the **Esp8266 Modem Simulator**
 - Arduino Bootloader - Load microcontroller with **Arduino serial bootloader**
 - MPLABX Debugger Plugin - Open the web page to download the **MPLABX Debugger Plugin**
 - Pin Viewer - Open the **Pin Viewer**

- Help
 - Contents - Open the Help window
 - Board - Open the Board Help window
 - Examples - Load the examples
 - About Board - Show message about author and version of board
 - About PICSimLab - Show message about author and version of PICSimLab



The first part of the status bar shows the state of the simulation, in the middle part the status of the debug support and in the last part the name of the serial port used, its default speed and the error in relation to the real speed configured in the microcontroller.

Running... Debug: Off Serial Port: /dev/tnt2:4800(0.2%)

3.2 Interaction with the Board

On the interface area of the board it is possible to interact in some ways:

- Click in ICSP connector to load an .hex file.
- Click in PWR button to ON/OFF the emulator..
- The buttons can be activated through mouse or keys 1, 2, 3 e 4.

3.3 Command Line

PICSimLab supports two command lines format:

One for load a PICSimLab Workspace file (.pzw)

```
picsimlab file.pzw
```

And other for load .hex files

```
picsimlab boardname microcontroller [file.hex] [file.pcf]
```

3.4 Remote Control Interface

The PICSimLab remote control interface supports TCP connections using telnet or nc (netcat).

The default port is 5000 and can be changed in configuration windows.

The 'rlwrap' command can be used for best command edition support in telnet or nc:

```
rlwrap nc 127.0.0.1 5000
```

The supported commands can be shown using the "help" command:

```
help
List of supported commands:
dumpe [a] [s]- dump internal EEPROM memory
dumpf [a] [s]- dump Flash memory
dumpr [a] [s]- dump RAM memory
exit           - shutdown PICSimLab
get ob         - get object value
help           - show this message
info           - show actual setup info and objects
pins           - show pins directions and values
pinsl          - show pins formated info
quit           - exit remote control interface
reset          - reset the board
set ob vl     - set object with value
sync           - wait to syncronize with timer event
version        - show PICSimLab version
```

Ok

The "info" command show all available "objects" and values:

```
info
Board:      Arduino Uno
Processor:  atmega328p
Frequency:   16000000 Hz
Use Spare:  1
    board.out[00] LD_L= 254
part[00]: LEDs
    part[00].out[08] LD_1= 254
    part[00].out[09] LD_2= 30
    part[00].out[10] LD_3= 254
    part[00].out[11] LD_4= 254
    part[00].out[12] LD_5 254
    part[00].out[13] LD_6= 254
    part[00].out[14] LD_7= 254
part[01]: Buzzer
    part[01].out[02] LD_1= 140
```

```

part[02]: Push buttons
  part[02].in[00] PB_1= 1
  part[02].in[01] PB_2= 0
  part[02].in[02] PB_3= 1
  part[02].in[03] PB_4= 1
  part[02].in[04] PB_5= 1
  part[02].in[05] PB_6= 1
  part[02].in[06] PB_7= 1
  part[02].in[07] PB_8= 1

```

Ok

The “pins” command show all pins directions and digital values:

```

pins
pin[01] ( PC6/RST) < 0
pin[02] ( PD0/0) < 1
pin[03] ( PD1/1) < 1
pin[04] ( PD2/2) < 1
pin[05] ( PD3/~3) > 0
pin[06] ( PD4/4) < 1
pin[07] ( +5V) < 1
pin[08] ( GND) < 0
pin[09] ( PB6/X1) < 0
pin[10] ( PB7/X2) < 0
pin[11] ( PD5/~5) < 1
pin[12] ( PD6/~6) < 1
pin[13] ( PD7/7) < 1
pin[14] ( PB0/8) > 0
pin[15] ( PB1/~9) > 0
pin[16] ( PB2/~10) > 0
pin[17] ( PB3/~11) > 0
pin[18] ( PB4/12) < 0
pin[19] ( PB5/13) > 0
pin[20] ( +5V) < 1
pin[21] ( AREF) < 0
pin[22] ( GND) < 0
pin[23] ( PC0/A0) < 0
pin[24] ( PC1/A1) < 0
pin[25] ( PC2/A2) < 0
pin[26] ( PC3/A3) < 0
pin[27] ( PC4/A4) > 0
pin[28] ( PC5/A5) > 0

```

Ok

The “pinsl” command show all pins info in text formatted output:

```

pinsl
28 pins [atmega328p]:
pin[01] D I 0 000 0.000 "PC6/RST "
pin[02] D I 1 200 0.000 "PD0/0 "
pin[03] D I 1 200 0.000 "PD1/1 "
pin[04] D I 1 200 0.000 "PD2/2 "
pin[05] D O 0 007 0.000 "PD3/~3 "
pin[06] D I 1 200 0.000 "PD4/4 "
pin[07] P I 1 200 0.000 "+5V "
pin[08] P I 0 000 0.000 "GND "
pin[09] D I 0 000 0.000 "PB6/X1 "
pin[10] D I 0 000 0.000 "PB7/X2 "
pin[11] D I 1 200 0.000 "PD5/~5 "
pin[12] D I 1 200 0.000 "PD6/~6 "
pin[13] D I 1 200 0.000 "PD7/7 "

```

```

pin[14] D O 0 000 0.000 "PB0/8 "
pin[15] D O 0 000 0.000 "PB1/~9 "
pin[16] D O 0 000 0.000 "PB2/~10 "
pin[17] D O 0 006 0.000 "PB3/~11 "
pin[18] D I 0 000 0.000 "PB4/12 "
pin[19] D O 0 000 0.000 "PB5/13 "
pin[20] P I 1 200 0.000 "+5V "
pin[21] R I 0 000 0.000 "AREF "
pin[22] P I 0 000 0.000 "GND "
pin[23] A I 0 000 0.875 "PC0/A0 "
pin[24] A I 0 000 1.925 "PC1/A1 "
pin[25] A I 0 000 2.700 "PC2/A2 "
pin[26] A I 0 000 4.275 "PC3/A3 "
pin[27] D O 1 179 0.000 "PC4/A4 "
pin[28] D O 1 186 0.000 "PC5/A5 "

```

Ok

You can view one input/output state using the “get” command:

```

get board.out[00]

get part[02].in[01]

```

Its possible use the “get” command to view individual pins state:

```

#digital state
get pin[19]
pin[19]= 0
Ok

#digital mean value (0-200)
get pinm[19]
pin[18]= 100
Ok

#analog state
get apin[25]
apin[25]= 2.700
Ok

#all info
get pinl[13]
pin[13] D I 1 200 0.000 "PD7/7 "

```

And set value of one input using the “set” command:

```

set part[02].in[01] 0
set part[02].in[01] 1

```

Or set value of one pin using the “set” command:

```
#digital  
set pin[10] 2  
  
#analog  
set apin[20] 2.345
```

For windows users [putty telnet client](#) is a good option.

Chapter 4

Boards

PICSimLab currently supports five backend simulators. The stable version supports [picsim](#) and [simavr](#). The experimental version supports [uCsim](#), [gpsim](#) and [qemu-stm32](#) in addition to the stable ones.

The Figure 4.1 shows which cards are based on which backend simulator:

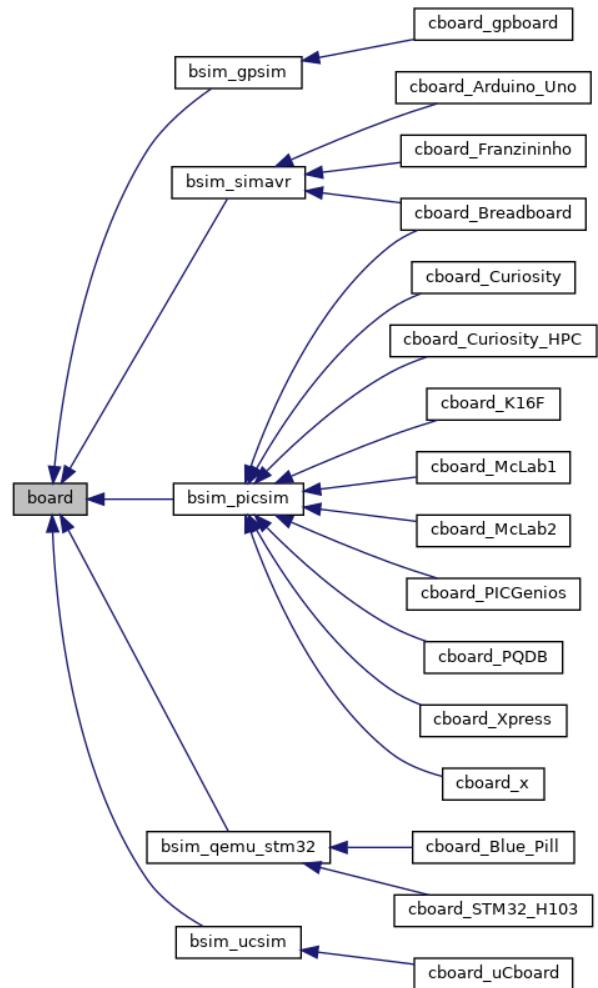


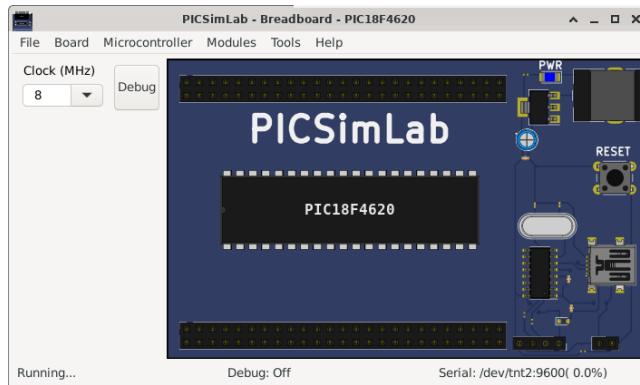
Figure 4.1: Boards backend simulators

The below table show the supported debug interface of each simulator:

Backend	Debug Support
picsim	MPLABX Integrated Debug (see section 7.1)
simavr	MPLABX Integrated Debug (see section 7.1) and remote avr-gdb (see section 7.3)
qemu-stm32	remote arm-gdb (see Chapter 7.4)
uCsim	uCsim remote console (telnet) (see section 7.5)
gpsim	none yet

4.1 Breadboard

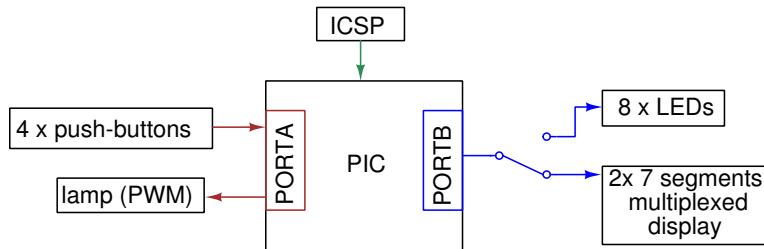
It is a generic board only with reset, serial and crystal circuits and support to multiple microcontrollers of [picsim](#) and [simavr](#).

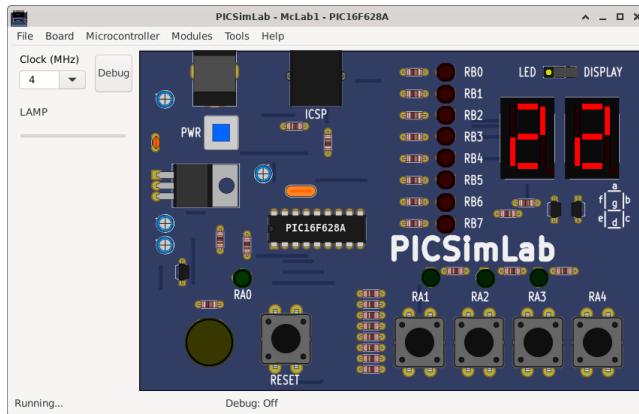


[Examples](#)

4.2 McLab1

It emulates the Labtools development board McLab1 that uses one PIC16F84, PIC16F628 or PIC16F648 of [picsim](#).





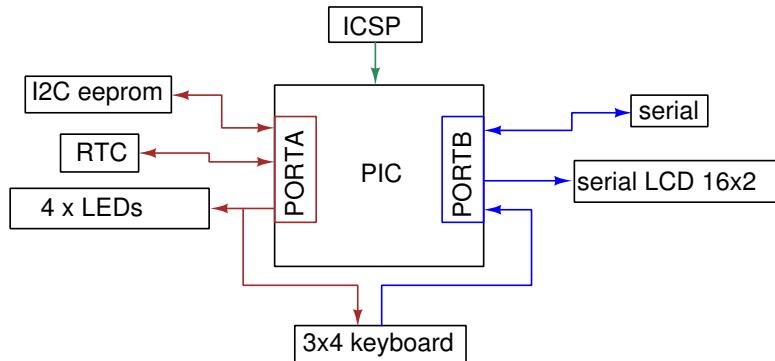
Board McLab1 schematics.

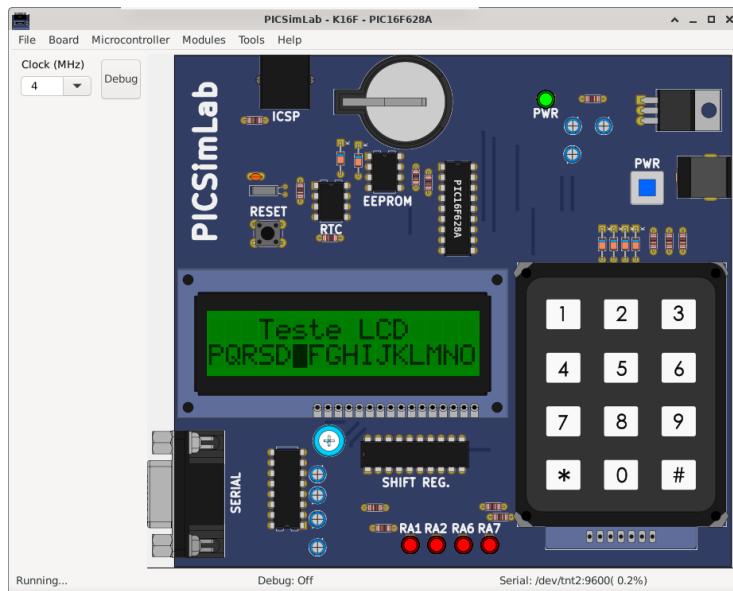
The code examples can be loaded in PICSimLab menu **Help->Examples**.

The source code of board McLab1 examples using [MPLABX](#) and [XC8](#) compiler are in the link: [board_McLab1](#).

4.3 K16F

It emulates an didactic board developed by author that uses one PIC16F84, PIC16F628 or PIC16F648 of [picsim](#).





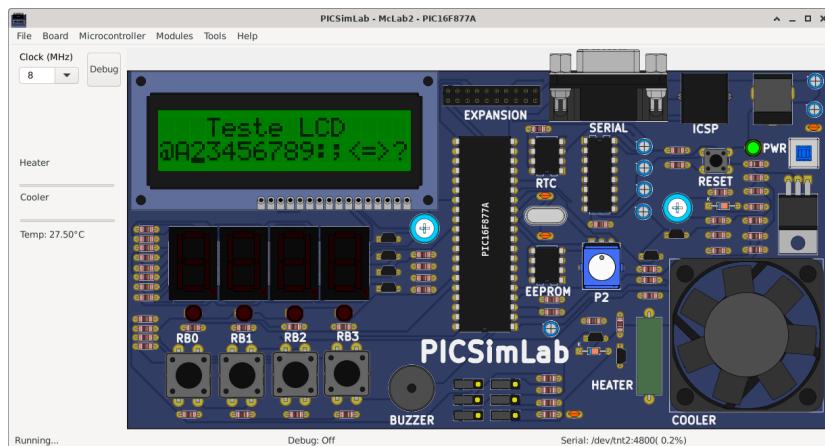
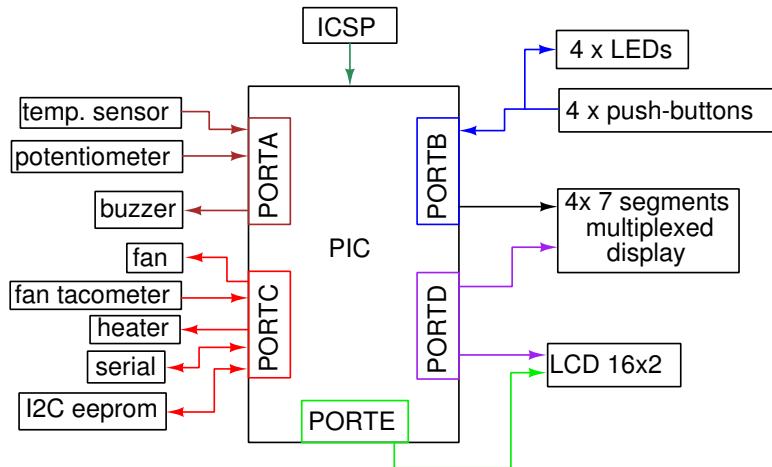
Board K16F schematics.

The code examples can be loaded in PICSimLab menu **Help->Examples**.

The source code of board K16F examples using [MPLABX](#) and [XC8](#) compiler are in the link: [board_K16F](#).

4.4 McLab2

It emulates the Labtools development board McLab2 that uses one PIC16F777, PIC16F877A, PIC18F452, PIC18F4520, PIC18F4550 or PIC18F4620 of [picsim](#).



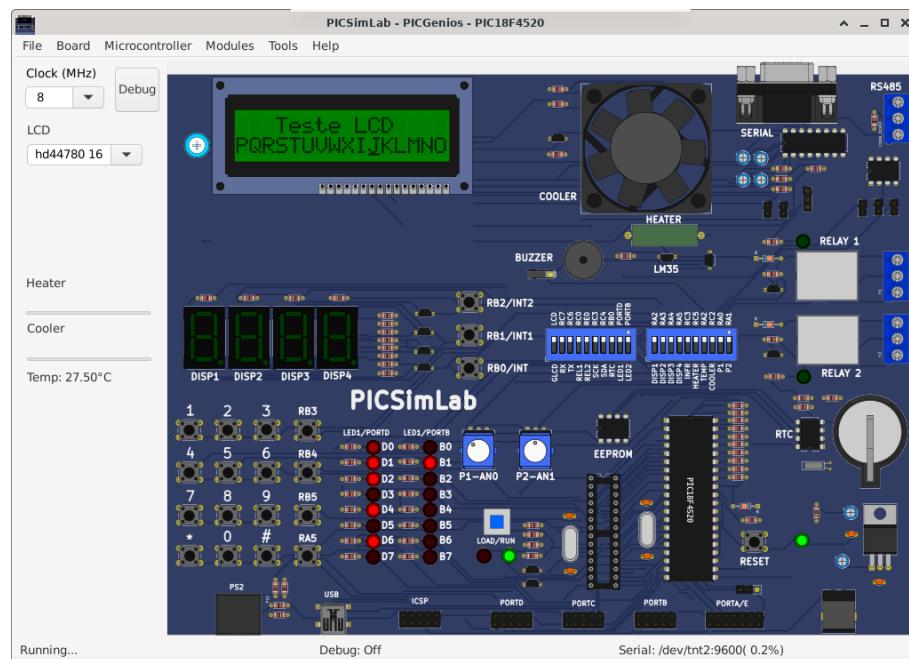
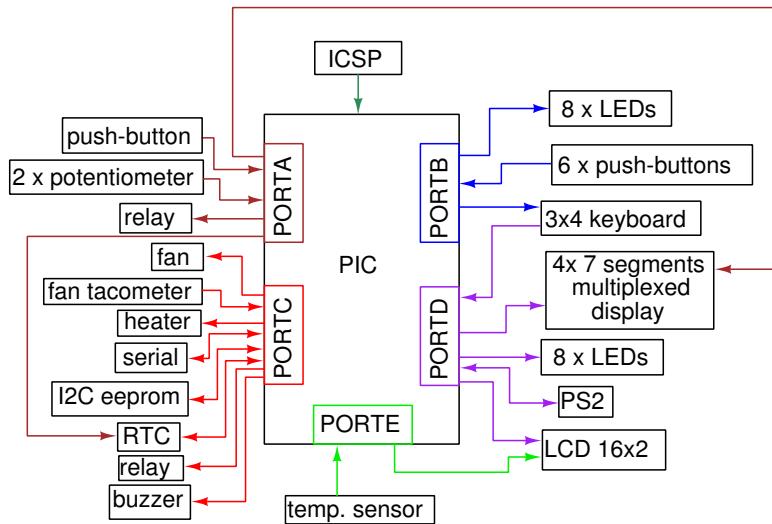
Board McLab2 schematics.

The code examples can be loaded in PICSimLab menu **Help->Examples**.

The source code of board McLab2 examples using **MPLABX** and **XC8** compiler are in the link: [board_McLab2](#).

4.5 PICGenios

It emulates the microgenius development board PICGenios PIC18F e PIC16F Microchip that uses one PIC16F777, PIC16F877A, PIC18F452, PIC18F4520, PIC18F4550 or PIC18F4620 of [picsim](#).



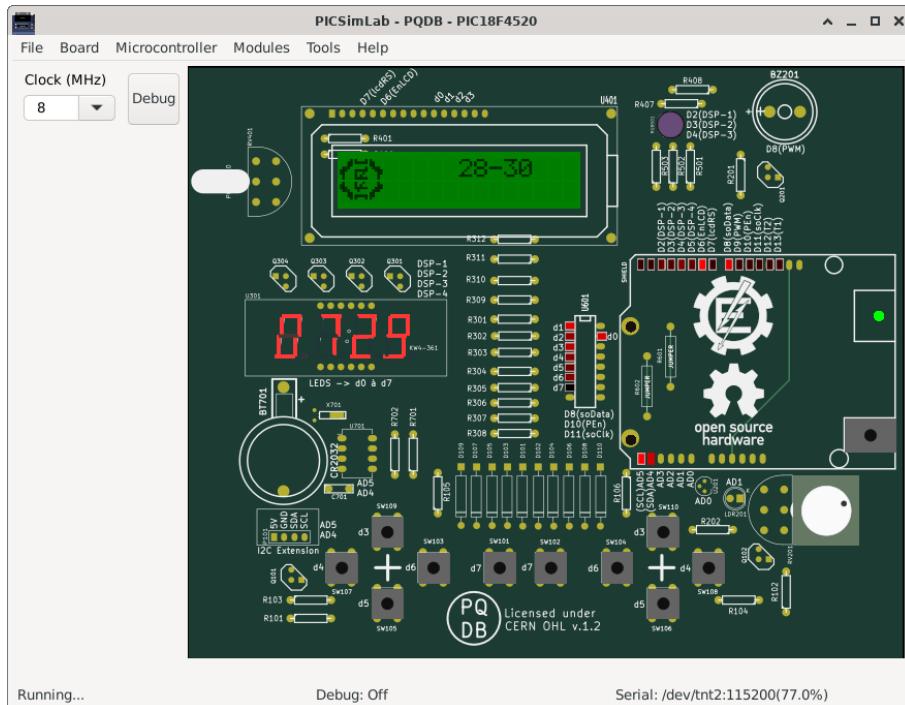
Board PICGenios schematics.

The code examples can be loaded in PICSimLab menu **Help->Examples**.

The source code of board PICGenios examples using **MPLABX** and **XC8** compiler are in the link: [board_PICGenios](#).

4.6 PQDB

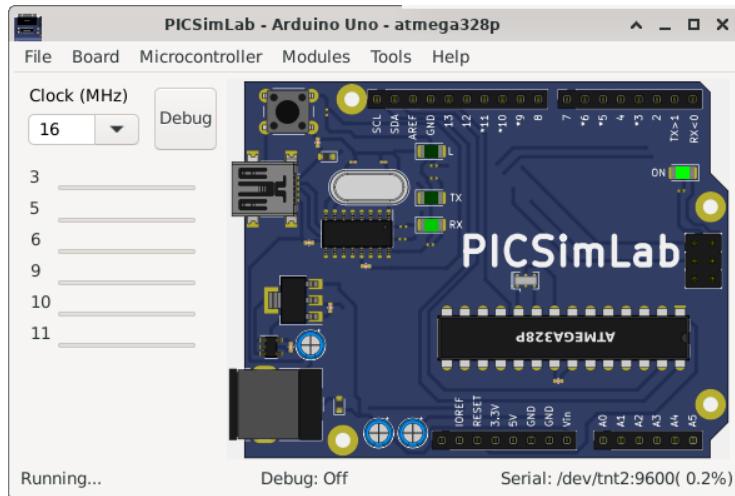
The PQDB board is an opensource/openhardware project, more info at <https://github.com/projetopqdb/>. It was developed to be used with arduino/freedom boards, but adapted to use the microcontroller PIC18F4520 of [picsim](#) on PICSImLab.



[Examples](#)

4.7 Arduino Uno

It emulates the Arduino Uno development board that uses one ATMEGA328P microcontroller of [simavr](#).



Board Arduino Uno schematics.

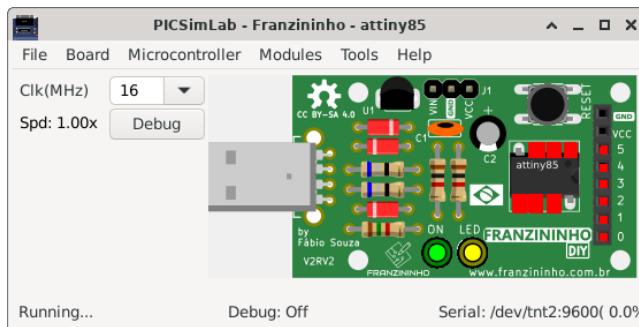
The code examples can be loaded in PICSimLab menu **Help->Examples**.

The source code of board Arduino Uno examples using the [Arduino IDE with avr-gcc](#) are in the link: [board_Arduino_Uno](#).

More information about the Arduino in [www.arduino.cc](#)

4.8 Franzininho

The Franzininho DIY board is an openhardware project, more info at <https://franzininho.com.br/>. It was developed to be used with the microcontroller ATTiny85 of of [simavr](#) on PIC-SIMLab.



Examples

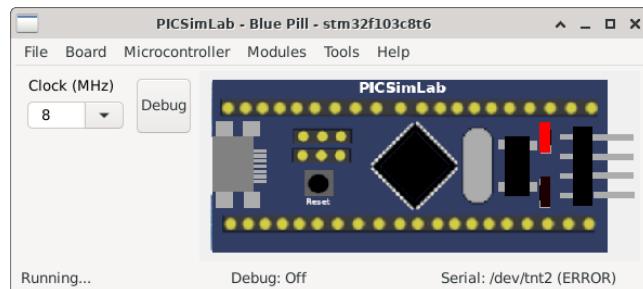
Chapter 5

Experimental Boards

Boards in the experimental phase. Probably with some bugs and missing features.

5.1 Blue Pill

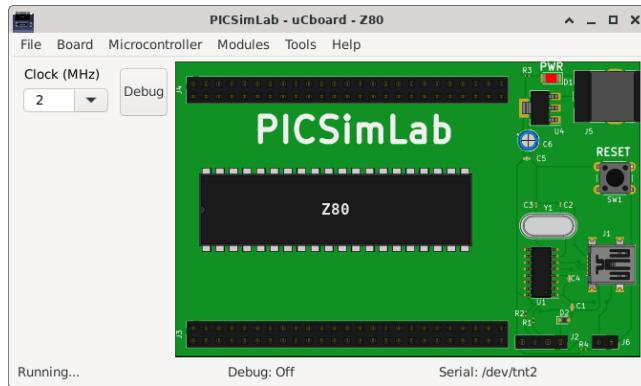
It is a generic board only with reset, serial and crystal circuits and support to stm32f103c8t6 microcontroller of [qemu-stm32](#).



[Examples](#)

5.2 uCboard

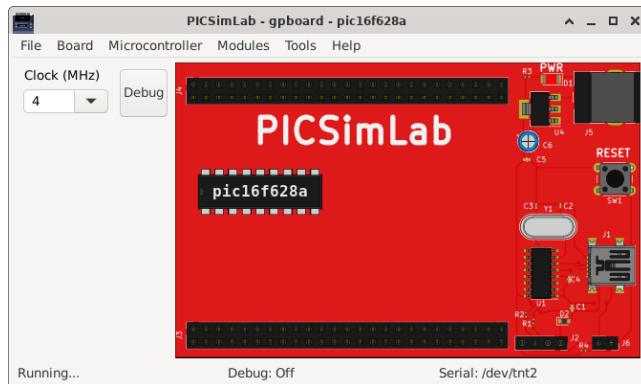
It is a generic board only with reset, serial and crystal circuits and support to multiple microcontrollers (initially C51, Z80 and STM8S103)of [uCsim](#).



[Examples](#)

5.3 gpboard

It is a generic board only with reset, serial and crystal circuits and support to multiple microcontrollers of [gpsim](#).



[Examples](#)

5.4 STM32 H103

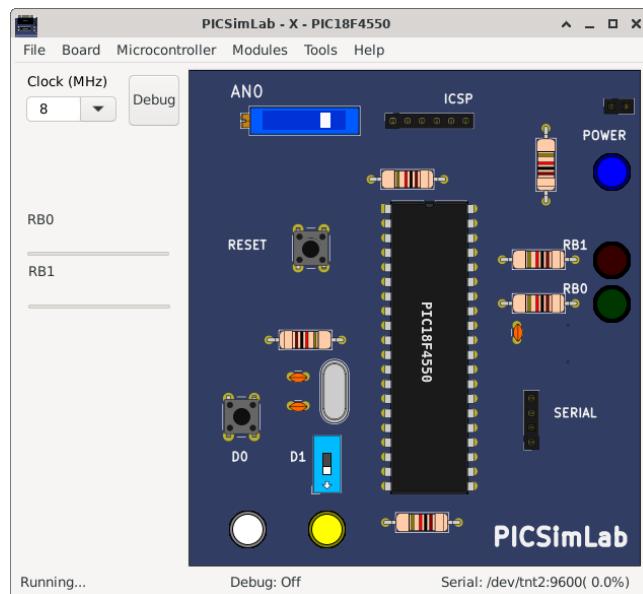
It is a generic board only with reset, one push button, serial and crystal circuits and support to stm32f103rbt6 microcontroller of [qemu-stm32](#).



Examples

5.5 X

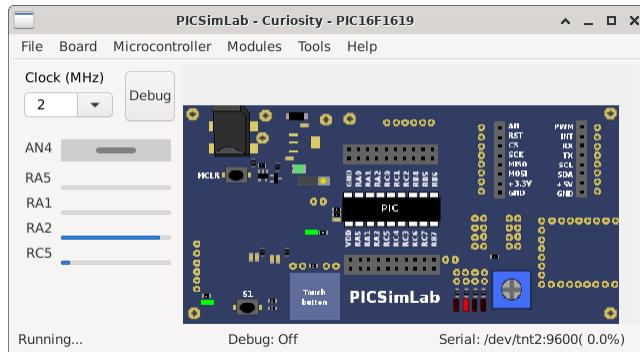
It is a generic board, used as example in [How to Compile PICsimLab and Create New Boards](#).



Examples

5.6 Curiosity

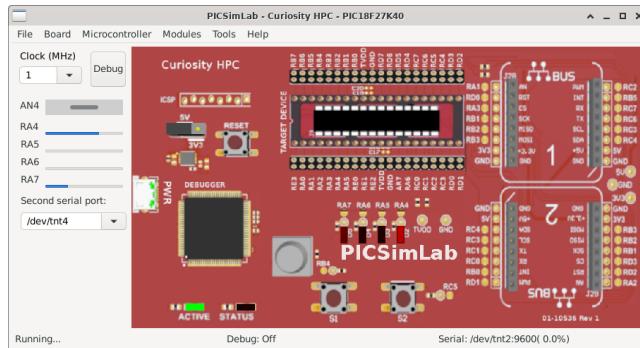
This is a simple PIC microcontroller development board that uses [picsim](#).



[Examples](#)

5.7 Curiosity HPC

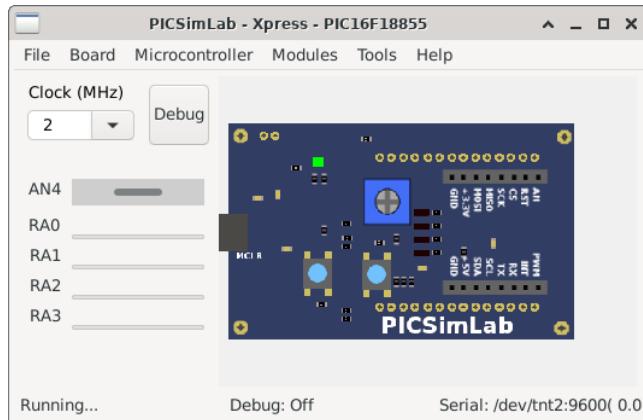
This is a simple PIC microcontroller development board that uses [picsim](#).



[Examples](#)

5.8 Xpress

This is a simple PIC microcontroller development board that uses [picsim](#).



[Examples](#)

Chapter 6

Serial Communication

To use the simulator serial port, install a NULL-MODEM emulator:

- Windows: com0com <http://sourceforge.net/projects/com0com/>
- Linux: tty0tty <https://github.com/lcgamboa/tty0tty>

For communication the PICSimLab should be connected in one port of the NULL-MODEM emulator and the other application connected in the other port. Configuration examples linking PICSimLab to [CuteCom](#) for serial communication:

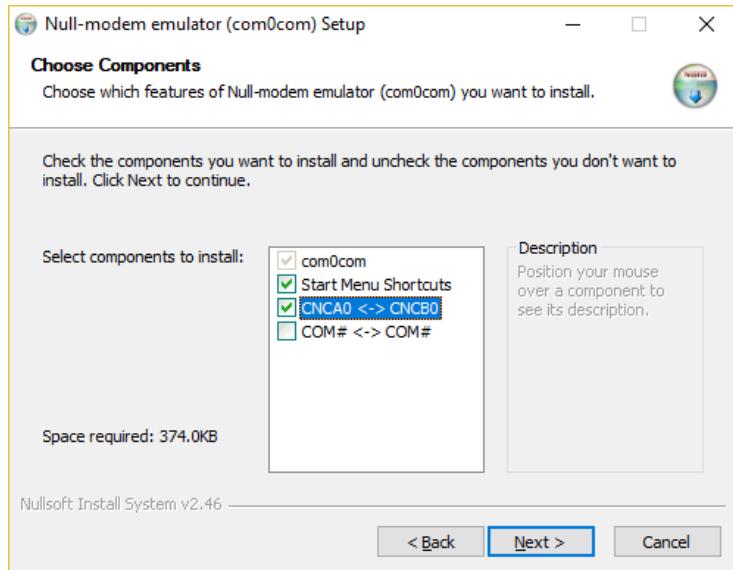
OS	PicsimLab port	CuteCom port	NULL-Modem prog.	Connection
Windows	com1	com2	com0com	com1<=>com2
Linux	/dev/tnt2	/dev/tnt3	tty0tty	/dev/tnt2<=>/dev/tnt3

6.1 Com0com Installation and Configuration(Windows)

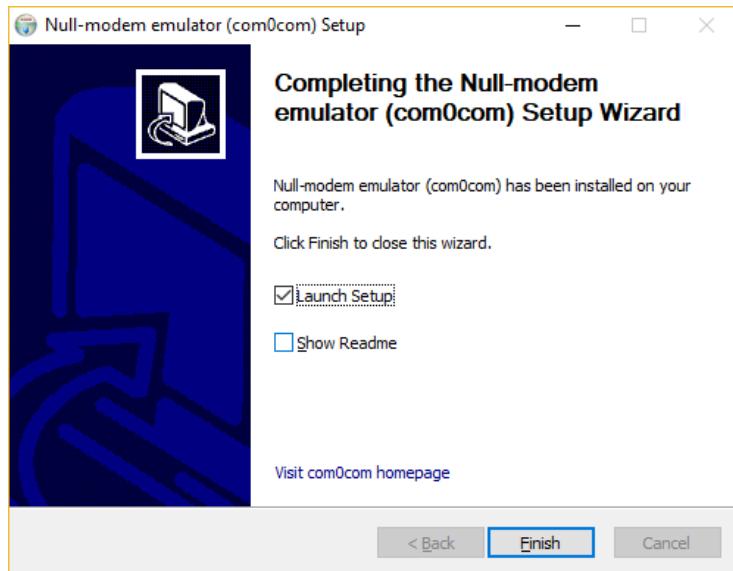
Download the signed version of [com0com](#).

Unzip the downloaded .zip file and run the specific installer of your operating system, x86 for windows 32-bit or x64 for windows 64-bit.

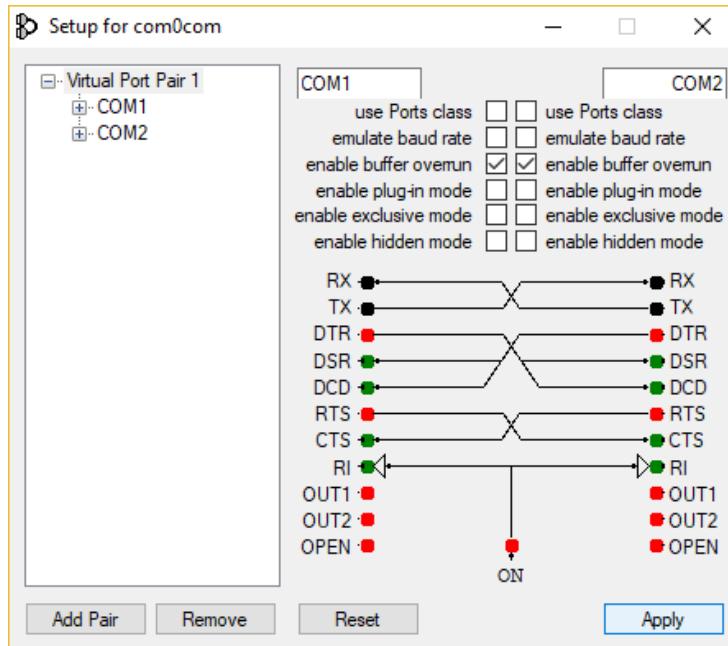
Configure the “choose components” window as the figure below:



In the last configuration window, check the “Launch setup” option:



In the setup window, change the port names to COM1, COM2, COM3 Just check the “enable buffer overrun” option on the two ports, click in the “Apply” button and close the setup. In the configuration shown in the figure below, the COM1 and COM2 ports form a NULL-MODEM connection, where one port must be used by the PICSimLab and another by the application with serial communication.



6.2 tty0tty Installation and Configuration (Linux)

Download the href <https://github.com/lcgamboa/tty0tty/archive/master.zip> tty0tty. Unzip the downloaded folder.

Open a terminal and enter in the `tty0tty/module/` folder and enter the following commands:

```
sudo apt-get update
sudo apt-get -y upgrade
sudo apt-get -y install gcc make linux-headers-`uname -r`
make
sudo make install
```

The user must be in the **dialout** group to access the ports. To add your user to **dialout** group use the command:

```
sudo usermod -a -G dialout your_user_name
```

after this is necessary logout and login to group permissions take effect.

Once installed, the module creates 8 interconnected ports as follows:

```
/dev/tnt0  <=>  /dev/tnt1
/dev/tnt2  <=>  /dev/tnt3
/dev/tnt4  <=>  /dev/tnt5
/dev/tnt6  <=>  /dev/tnt7
```

the connection between each pair is of the form:

TX	->	RX
RX	<-	TX
RTS	->	CTS
CTS	<-	RTS
DSR	<-	DTR
CD	<-	DTR
DTR	->	DSR
DTR	->	CD

Any pair of ports form a NULL-MODEM connection, where one port must be used by the PICSimLab and another by the application with serial communication.

Chapter 7

Debug Support

The type of debug interface depends on the backend simulator utilized.

7.1 MPLABX Integrated Debug (picsim and simavr)

To use the [MPLABX](#) IDE for debug and program the PicsimLab, install the plugin [com-picsim-picsimlab.nbm](#) in MPLABX.

The plugin connect to Picsimlab through a TCP socket using port 1234 (or other defined in configuration window), and you have to allow the access in the firewall.

[Tutorial: how to use MPLABX to program and debug PICsimLab.](#)

It's possible import and debug a Arduino sketch into MPLABX using the [Arduino import plugin](#).

7.2 Arduino IDE Integration (simavr)

For integrated use with the Arduino IDE, simply configure the serial port as explained in the section [6](#) and load the Arduino bootloader. The bootloader can be loaded from the “Tools->Arduino bootloader” menu.

In Windows, considering com0com making a NULL-MODEM connection between COM1 and COM2, simply connect the PICSimLab on the COM1 port (defined in configuration window) and the Arduino IDE on the COM2 port or vice versa.

On Linux the operation is the same, but using for example the ports /dev/tnt2 and /dev/tnt3.

In Linux for the virtual ports to be detected in Arduino it is necessary to replace the library lib/liblistSerialsj.so of the Arduino with a version which support the detection of tty0tty ports, that can be downloaded in the link [listSerialC with tty0tty support](#).

7.3 avr-gdb Debug (simavr)

With debug support enabled you can use avr-gdb to debug the code used in the simulator. Use the configuration window to choose between MDB (MPLABX) or GDB to debug AVR microcontrollers.

Use avr-gdb with the .elf file as the parameter:

```
avr-gdb compiled_file.elf
```

and the command below to connect (1234 is the default port):

```
target remote localhost:1234
```

Graphic debug mode can be made using [eclipse IDE](#) with Sloeber Arduino plugin.

7.4 arm-gdb Debug (qemu-stm32)

With debug support enabled you can use arm-none-eabi-gdb (or gdb-multiarch) to debug the code used in the simulator.

Use arm-none-eabi-gdb with the .elf file as the parameter:

```
arm-none-eabi-gdb compiled_file.elf
```

and the command below to connect (1234 is the default port):

```
target remote localhost:1234
```

Graphic debug mode can be made using [eclipse IDE](#) with Eclipse Embedded CDT.

7.5 uCsim Debug

The uCsim debug console can be accessed with the telnet (1234 is the default port):

```
telnet localhost 1234
```

All [uCsim commands](#) are supported.

For windows users [putty telnet client](#) is a good option to access the uCsim console.

Chapter 8

Tools

8.1 Serial Terminal

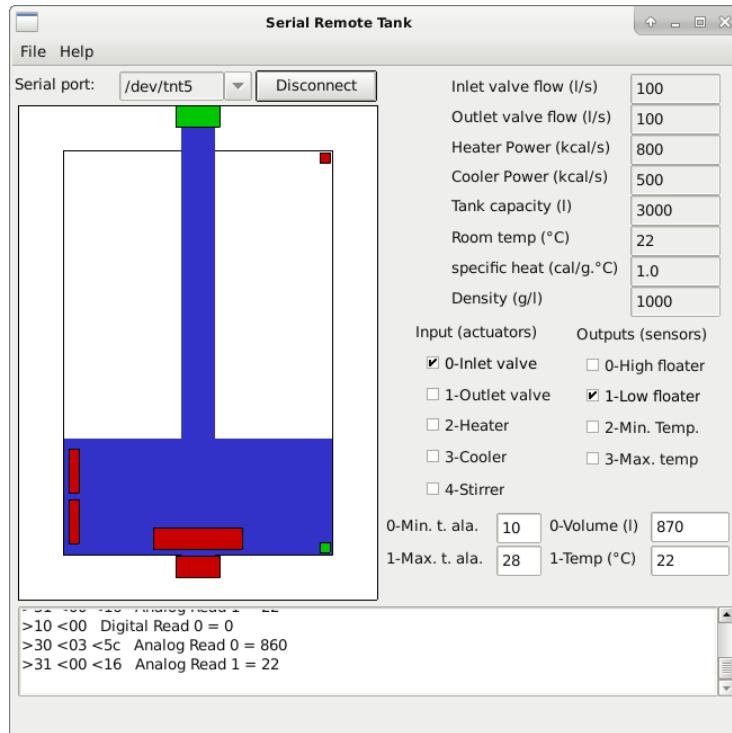
To use this tool with PICSimLab you first need to configure a virtual serial port as described in Chapter: [Serial Communication](#). It is possible to use this tool with a real serial port connected to a real device.

Open the serial terminal [CuteCom](#)

8.2 Serial Remote Tank

To use this tool with PICSimLab you first need to configure a virtual serial port as described in Chapter: [Serial Communication](#). It is possible to use this tool with a real serial port connected to a real device.

Serial Remote Tank



8.2.1 Sensors and Actuators

Actuators

Digital inputs

1. Inlet valve
2. Outlet valve
3. Heater
4. Cooler
5. Stirrer

Analog inputs

1. Minimal temperature alarm trigger level
2. Maximal temperature alarm trigger level

Sensors

Digital outputs

1. High floater
2. Low floater
3. Minimal temperature
4. Maximal temperature

Analog outputs

1. Volume
2. Temperature

8.2.2 Communication Protocol

Writing on Digital Input

Sent one byte in 0x0N hexadecimal format where N is the number of input followed by a second byte with value 0x00 for disable or 0x01 for enable.

Example to turn on the input 2:

```
Serial_write(0x02);
Serial_write(0x01);
```

Reading Digital Output

Sent one byte in 0x1N hexadecimal format where N is the number of output and read one byte. The byte readed have value 0x00 for disable or 0x01 for enable.

Example to read output 3:

```
Serial_write(0x13);
valor=Serial_read(0);
```

Writing on Analog Input

Sent one byte in 0x2N hexadecimal format where N is the number of input followed by two bytes with the 16 bits value.

Example to write the value 230 on analog input 1:

```
Serial_write(0x21);
valor=230;
Serial_write((valor&0xFF00)>>8);
Serial_write(valor&0x00FF);
```

Reading Analog Output

Sent one byte in 0x3N hexadecimal format where N is the number of output and read two bytes to form the 16 bits value.

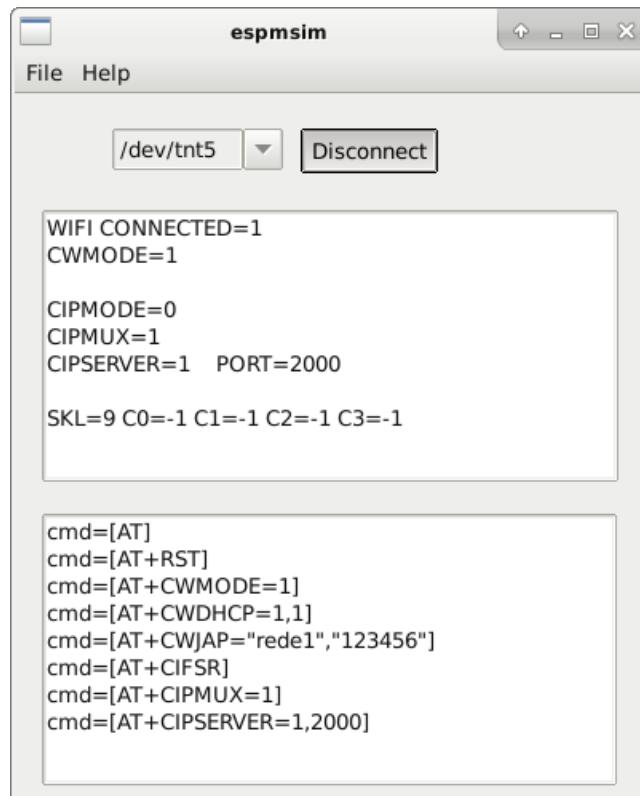
Example to read analog output 2:

```
Serial_write(0x32);
valorh=Serial_read(0);
valorl=Serial_read(0);
valor=(valorh<<8)|valorl;
```

8.3 Esp8266 Modem Simulator

To use this tool with PICSimLab you first need to configure a virtual serial port as described in Chapter: [Serial Communication](#). It is possible to use this tool with a real serial port connected to a real device.

ESP8266 Modem Simulator



8.3.1 Supported Commands

- AT
- AT+RST
- AT+GMR
- AT+CWMODE=1
- AT+CWDHCP=1,1
- AT+CWLAP
- AT+CWJAP="rede1","123456"
- AT+CIFSR
- AT+CIPMUX=1
- AT+CIPSERVER=1,2000
- AT+CIPSEND=0,10
- AT+CIPCLOSE=0

8.4 Arduino Bootloader

To use this tool with PICSimLab you first need to configure a virtual serial port as described in Chapter: [Serial Communication](#).

Load microcontroller with Arduino serial bootloader

8.5 MPLABX Debugger Plugin

- Open the web page to download the plugin

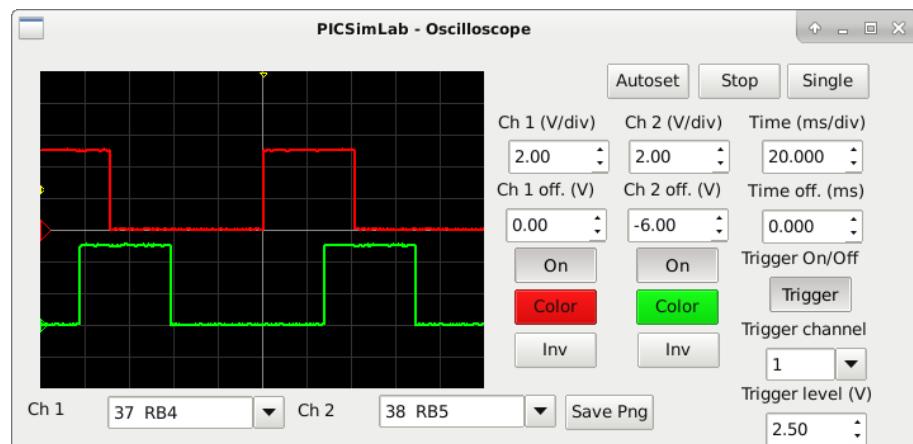
8.6 Pin Viewer

Open the Pin status viewer program

Chapter 9

Oscilloscope

The PICSimLab has a basic two-channel oscilloscope that can be used to view the signal on any pin of the microcontroller. The oscilloscope can be accessed through the “Modules->Oscilloscope” menu.



Chapter 10

Spare Parts

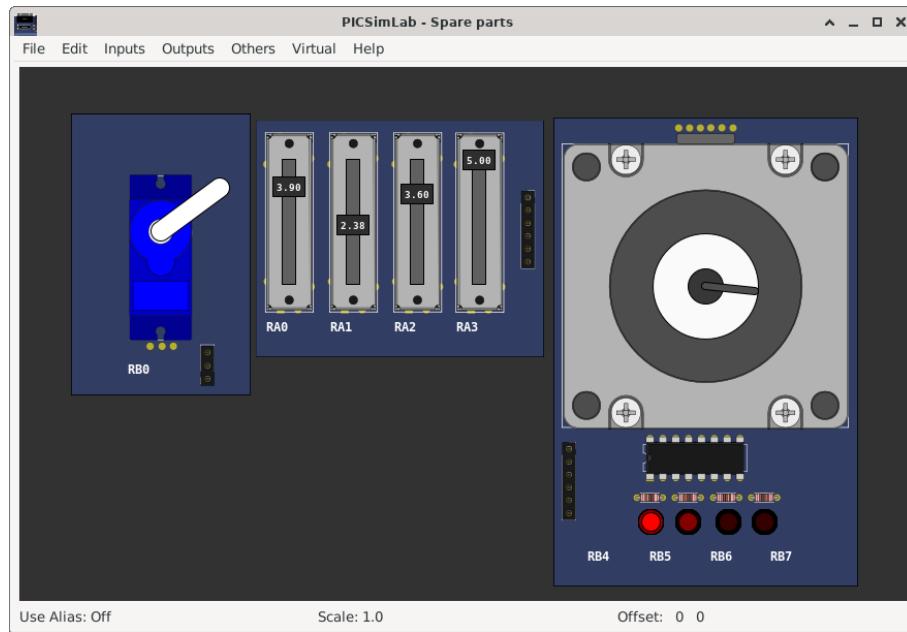
The PICSimLab has a window that allows the connection of spare parts to the microcontroller, it can be accessed through the menu “ Modules-> Spare parts ”.

The main window has the menu with the following functions:

- File
 - New configuration - Clear the spare parts window
 - Save configuration - Saves the current settings of the spare parts into .pcf file
 - Load configuration - Loads the settings from .pcf file
 - Save pin alias - Saves the current pin alias to .ppa text file
 - Load pin alias - Loads the pin alias from .ppa file
- Edit
 - Clear pin alias - Clear the pin alias
 - Toggle pin alias - Enable/Disable pin alias use
 - Edit pin alias - Open current pin alias .ppa file in text editor
 - Reload pin alias - Reload the current .ppa pin alias file (need after edit .ppa file)
 - Zoom in - Increase draw scale
 - Zoom out - Decrease draw scale
- Inputs
 - Encoder - Adds a rotary quadrature encoder with push button
 - Gamepad - Adds a gamepad
 - Gamepad (Analogic) - Adds a gamepad with one analogic output
 - Keypad - Adds one matrix keypad

- MPU6050 - Adds a accelerometer and gyroscope (only raw values)
- Potentiometers - Adds 4 potentiometers
- Potentiometers (Rotary) - Adds 4 rotary potentiometers
- Push Buttons - Adds 8 push buttons
- Push Buttons (Analogic) - Adds 8 push buttons with analog output
- Switchs - Adds eight switchs
- Ultrasonic HC-SR04 - Adds a ultrasonic range sensor
- Outputs
 - 7 Segments Display - Adds four multiplexed 7 segments displays
 - 7 Segments Display (w/dec) - Adds four multiplexed 7 segments displays with decoder
 - Buzzer - Adds a active/passive buzzer
 - DC Motor - Adds a DC motor with H-bridge and quadrature encoder
 - LCD hd44780 - Adds a text display hd44780
 - LCD ili9340 - Adds a color graphic display ili9340 with touchscreen
 - LCD pcd8544 - Adds a monochrome graphic display pcd8544 (Nokia 5110)
 - LCD pcf8833 - Adds a color graphic display pcf8833
 - LCD ssd1306 - Adds a monochrome graphic display ssd1306
 - LED Matrix - Adds a 8x8 LED matrix with MAX72xx controller
 - LEDs - Adds 8 red LEDs
 - RGB LED - Adds one RGB LED
 - Servo Motor - Adds a servo motor
 - Step Motor - Adds a step motor
- Others
 - ETH w5500 - Adds a ethernet shield w5500
 - IO 74xx595 - Adds a 74xx595 SIPO 8 bit shift register
 - IO MCP23S17 - Adds a MCP23S17 serial SPI IO expander
 - IO PCF8574 - Adds a PCF8574 serial I2C IO expander
 - IO UART - Adds a UART serial port
 - Jumper Wires - Adds sixteen jumper wires
 - MEM 24CXXX - Adds a 24CXXX serial I2C EEPROM memory
 - RTC ds1307 - Adds a ds1307 real time clock
 - RTC pfc8563 - Adds a pfc8563 real time clock
 - SD Card - Adds a SD card shield

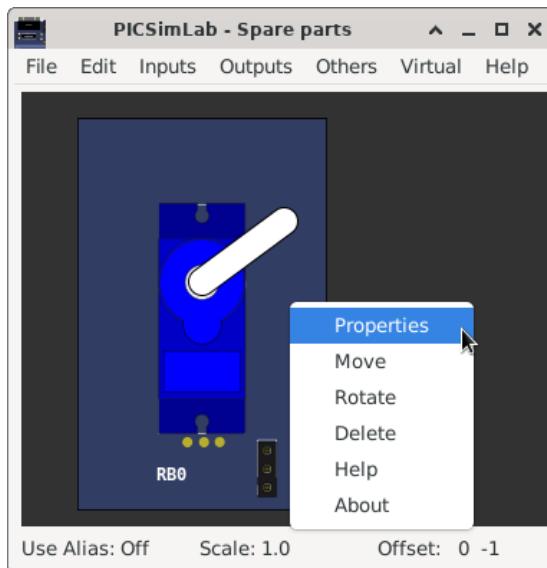
- Temperature System - Adds a temperature control system
- Virtual
 - D. Transfer Function - Adds a discrete transfer function mathematical model
 - IO Virtual term - Adds a virtual serial terminal
 - Signal Generator - Adds a virtual signal generator
 - VCD Dump - Adds a digital value file dump recorder
 - VCD Dump (Analogic) - Adds a analog value file dump recorder
 - VCD Play - Adds a digital value file dump player
- Help
 - Contents - Open Help window
 - About - Show message about author and version



After adding the part, with a right click of the mouse you can access the options menu of the part with the options:

- Properties - Opens the connection settings window
- Move - Unlocks the part to move
- Rotate - Change the orientation of part

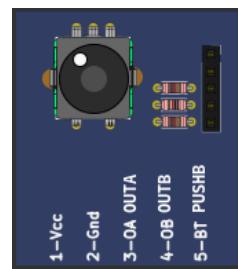
- Delete - Remove part
- Help - Open Help window of part
- About - Show message about author and version of part



10.1 Inputs

10.1.1 Encoder

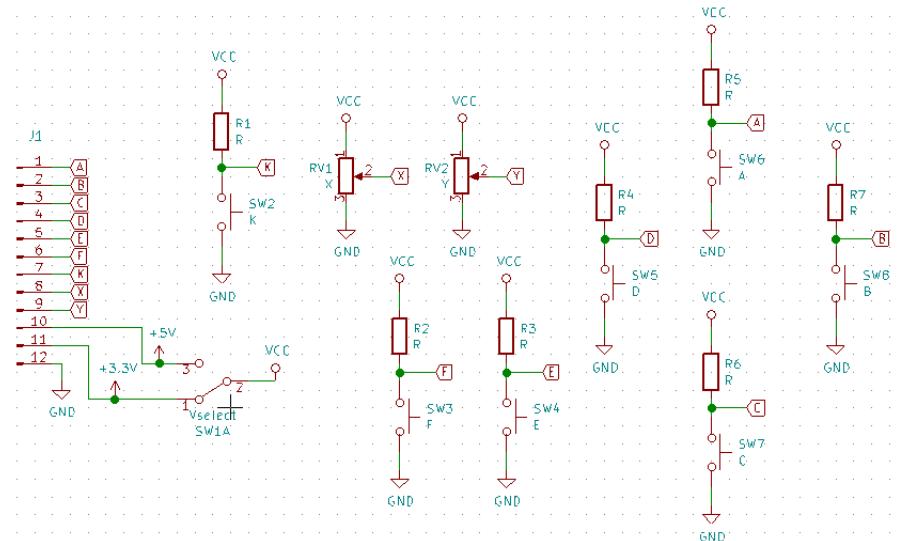
This part is a rotary quadrature encoder with push button. The output is twenty pulses per revolution.



[Examples](#)

10.1.2 Gamepad

This part is a gamepad with two analog axis and 7 push buttons.



The gamepad can be controlled by keyboards keys:

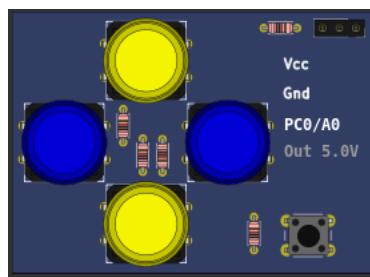
- X axis - keys 'A' and 'D'
- Y axis - keys 'W' and 'S'
- Button A - key 'I'
- Button B - key 'L'
- Button C - key 'K'

- Button D - key 'J'
- Button E - key 'E'
- Button F - key 'O'
- Button K - key 'R'

[Examples](#)

10.1.3 Gamepad Analogic

This part is a gamepad with 5 push buttons and one analogic output.



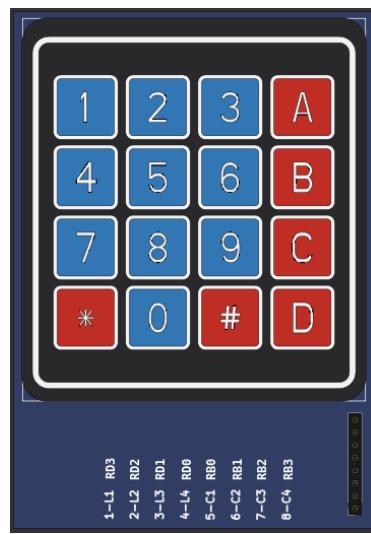
The gamepad can be controlled by keyboards keys:

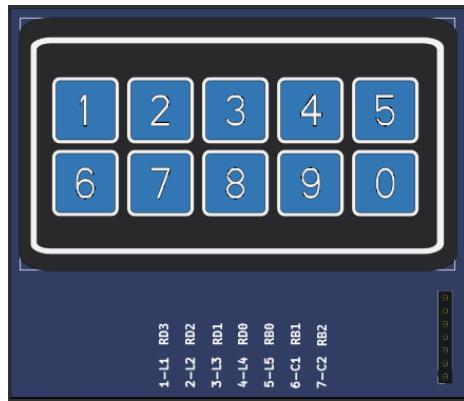
- Button A - key 'L'
- Button B - key 'I'
- Button C - key 'K'
- Button D - key 'J'
- Button E - key 'O'

[Examples](#)

10.1.4 Keypad

It is a matrix keyboard configurable to 4x3 , 4x4 or 2x5 rows/columns.

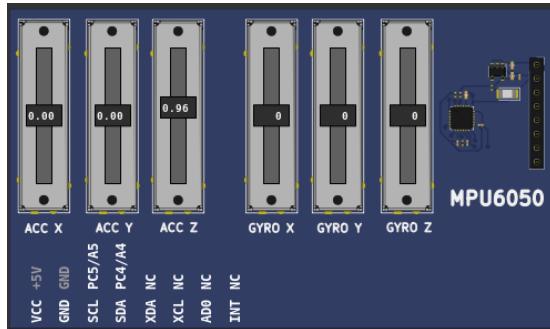




[Examples](#)

10.1.5 MPU6050

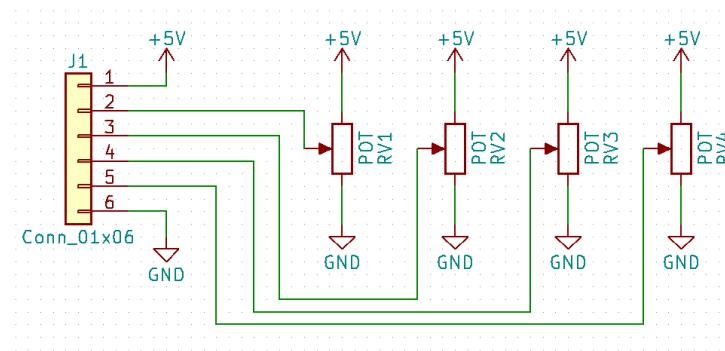
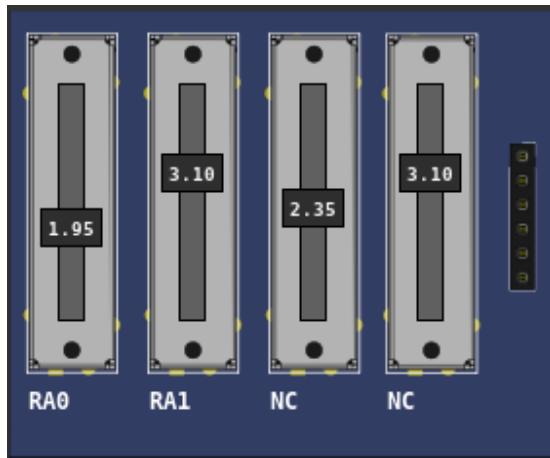
This part is MPU6050 accelerometer and gyroscope with I2C interface. Only raw values are available, DMP is not supported.



[Examples](#)

10.1.6 Potentiometers

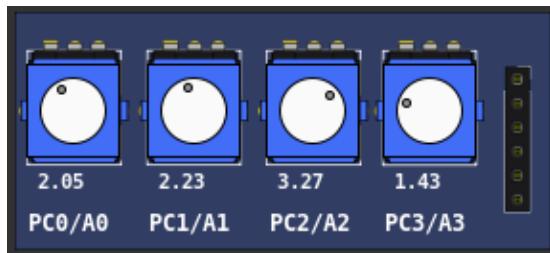
This part is formed by 4 potentiometers connected between 0 and 5 volts, the output is connected to the cursor and varies within this voltage range.

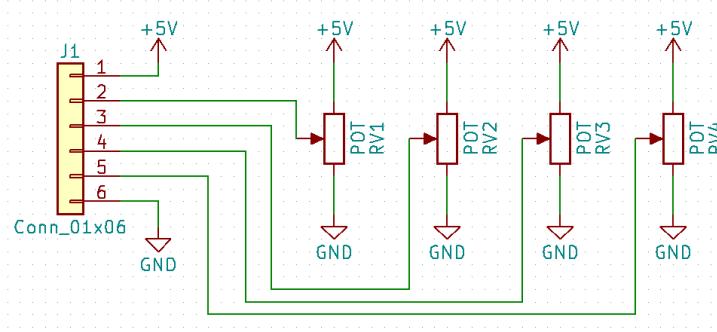


Examples

10.1.7 Potentiometers (Rotary)

This part is formed by 4 rotary potentiometers connected between 0 and 5 volts, the output is connected to the cursor and varies within this voltage range.

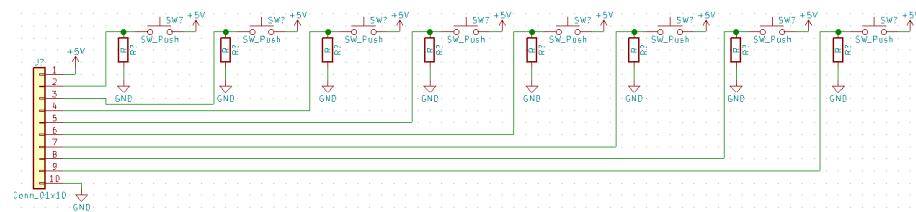




Examples

10.1.8 Push Buttons

This part consists of 8 push buttons. The output active state can be configurable.



Examples

10.1.9 Push Buttons (Analogic)

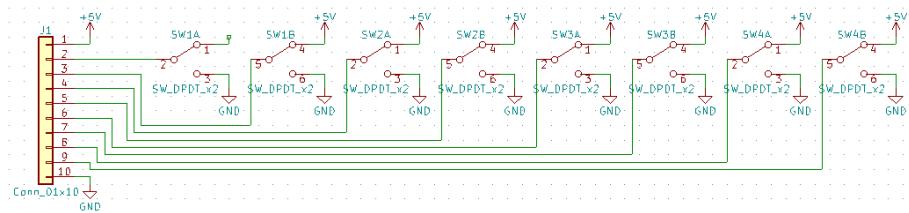
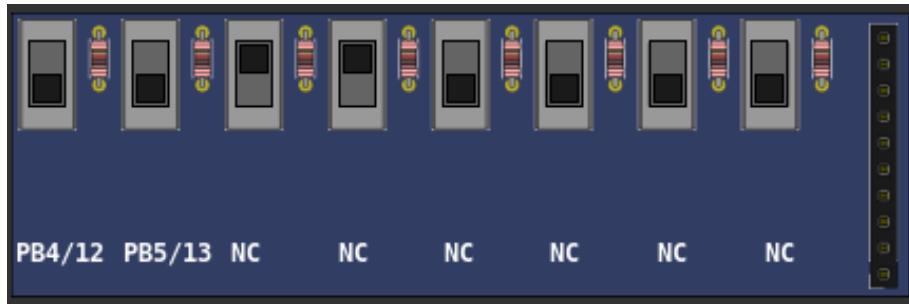
This part consists of 8 push buttons connected in a resistive ladder.



Examples

10.1.10 Switches

This part consists of 8 keys with on or off position (0 or 1).



Examples

10.1.11 Ultrasonic HC-SR04

This part is ultrasonic range meter sensor.

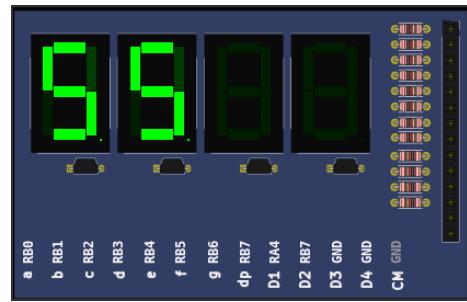


Examples

10.2 Outputs

10.2.1 7 Segments Display

This is a four multiplexed 7 segments displays.

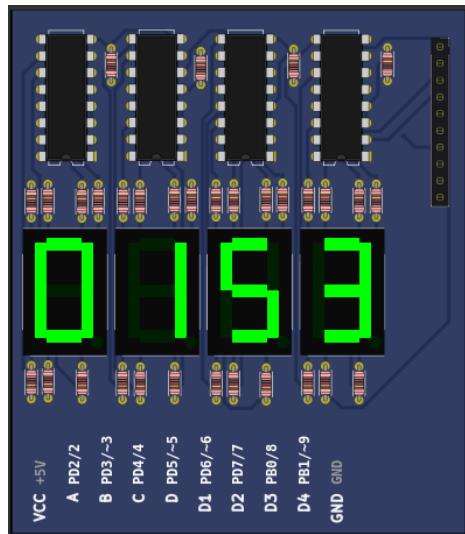


Examples

10.2.2 7 Segments Display (Decoder)

This is a four multiplexed 7 segments displays with BCD to 7 segments decoder (CD4511).





[Examples](#)

10.2.3 Buzzer

This is a active/passive buzzer.



[Examples](#)

10.2.4 DC Motor

This part is DC motor with H-bridge driver and quadrature encoder.



[Examples](#)

10.2.5 LCD hd44780

This part is a text display with 2 (or 4) lines by 16 (or 20) columns.

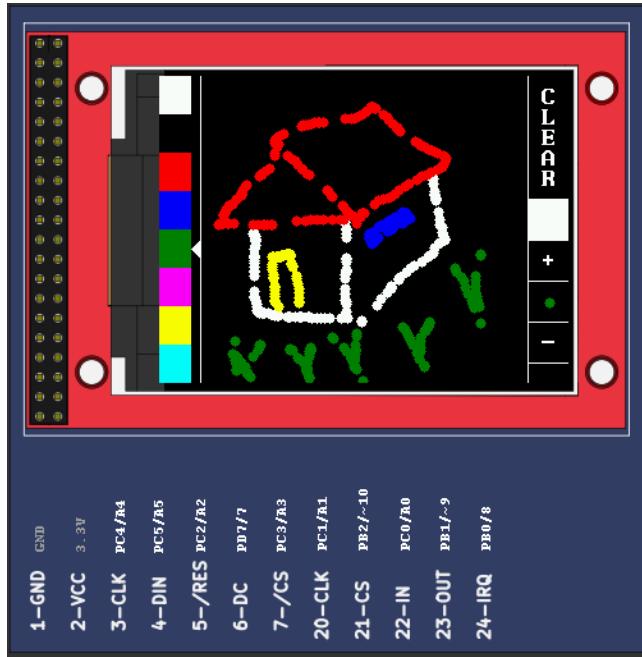




Examples

10.2.6 LCD ili9341

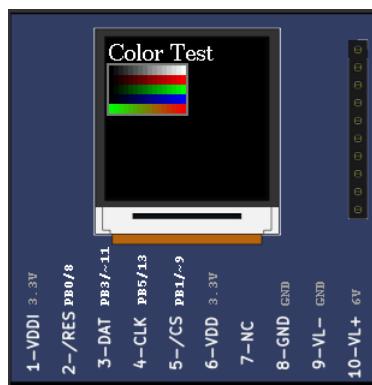
This part is a color graphic display with 240x320 pixels with touchscreen (xpt2046 controller). Only 4 SPI mode and 8 bits parallel mode is available.



[Examples](#)

10.2.7 LCD pcf8833

This part is a color graphic display with 132x132 pixels.



[Examples](#)

10.2.8 LCD pcd8544

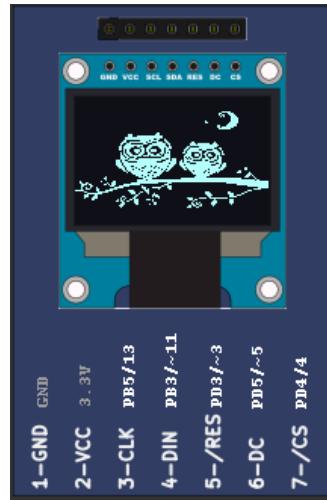
This part is a monochrome graphic display with 48x84 pixels. (Nokia 5110)



[Examples](#)

10.2.9 LCD ssd1306

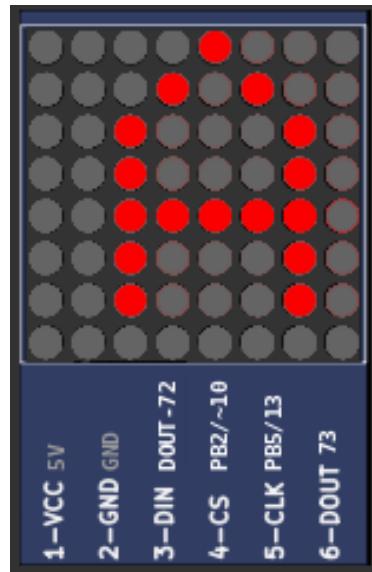
This part is a monochrome oled graphic display with 128x64 pixels. The part suport I2C and 4 SPI serial mode.



[Examples](#)

10.2.10 LED Matrix

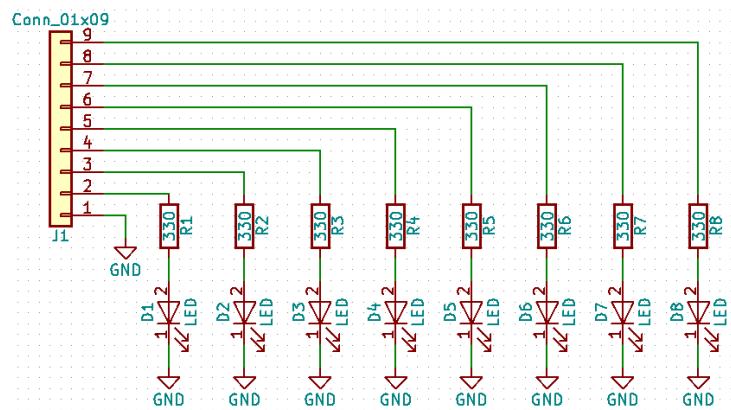
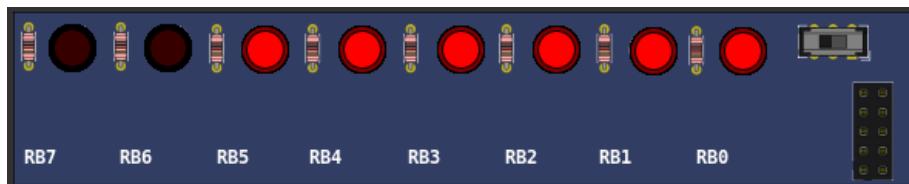
It is a 8x8 LED matrix with MAX72xx controller.



Examples

10.2.11 LEDs

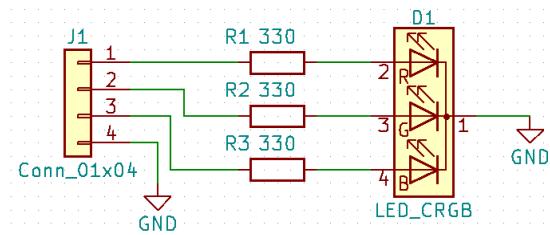
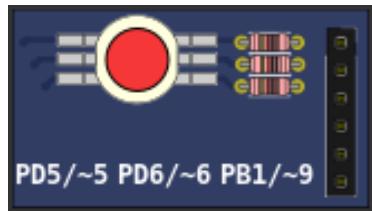
This part is a bar of 8 independent red LEDs.



Examples

10.2.12 RGB LED

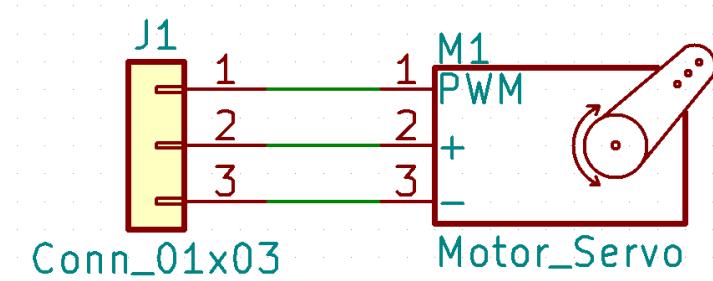
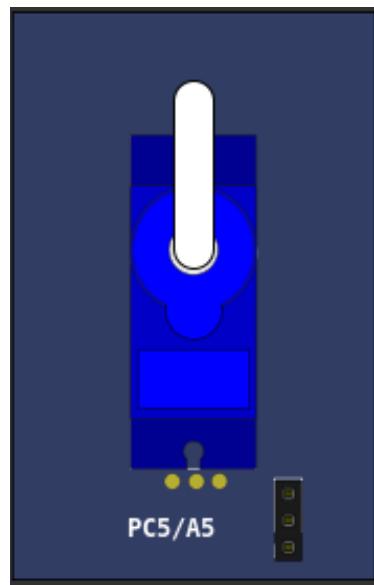
This part consists of a 4-pin RGB LED. Each color can be triggered independently. Using PWM it is possible to generate several colors by combining the 3 primary colors.



Examples

10.2.13 Servo Motor

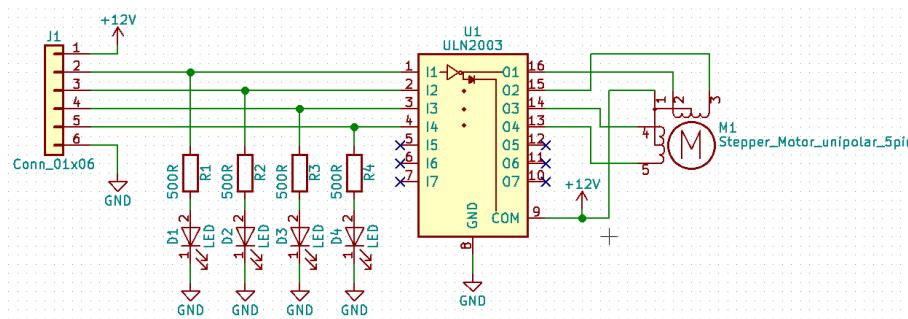
The servo motor is a component that must be activated with a pulse of variable width from 1ms to 2ms every 20 ms. A pulse of 1ms positions the servo at -90°, one from 1.5ms to 0° and one from 2ms to 90°.



Examples

10.2.14 Step Motor

The stepper motor is a component with 4 coils that must be driven in the correct order to rotate the rotor. Each step of the motor is 1.8° .



Examples

10.3 Others

10.3.1 ETH w5500

This part is a ethernet shield w5500 with support to 8 sockets simultaneously.

Only TCP/UDP unicast address sockets is supported. DHCP is emulated and return a fake ipv4 address.

All listening ports below 2000 are increased by 2000 to avoid operational system services ports. For example listening on port 80 becomes 2080.

w5500 Status Legend:

1º Letter - Type	2º Letter - Status	3º Letter - Error
C - Closed	C - Closed	B - Bind
T - TCP	I - Initialized	S - Send
U - UDP	L - Listen	R - Receive
M - MACRAW (don't supported)	S - Syn sent	L - Listen
	E - Established	U - Reuse
	W - Close wait	C - Connecting
	U - UDP	D - Shutdown
	M - MACRAW (don't supported)	

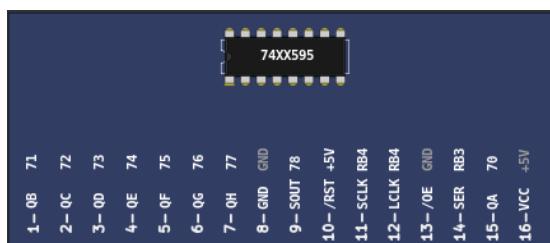
Click on connector to toggle link status.



Examples

10.3.2 IO 74xx595

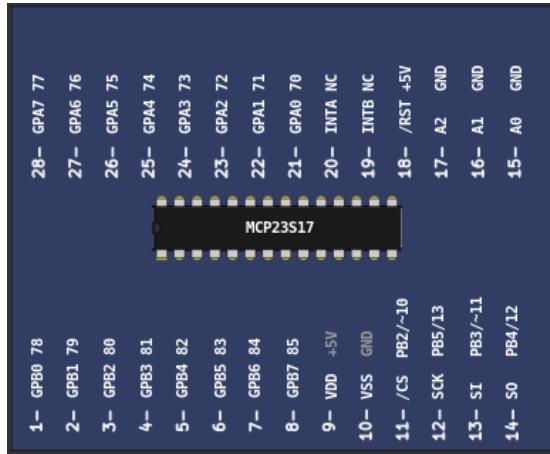
This is one 74xx595 serial input and parallel output 8 bit shift register.



Examples

10.3.3 IO MCP23S17

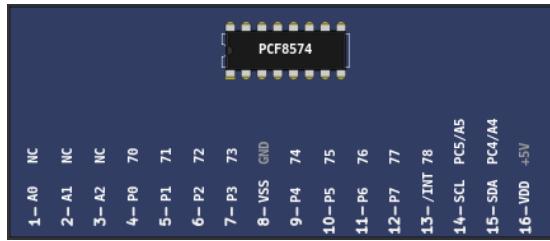
It is a MCP23S17 serial SPI IO expander part.



[Examples](#)

10.3.4 IO PCF8574

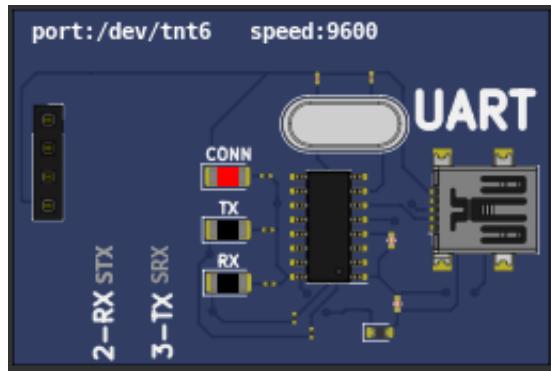
It is a PCF8574 serial I2C IO expander.



[Examples](#)

10.3.5 IO UART

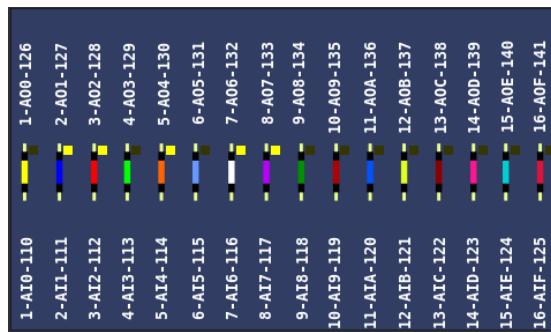
This part is a UART serial port. This part connects the hardware/software UART IO pins of microcontroller to one real/virtual PC serial port. To use virtual port is need to install a virtual port software, as described in [6](#).



[Examples](#)

10.3.6 Jumper Wires

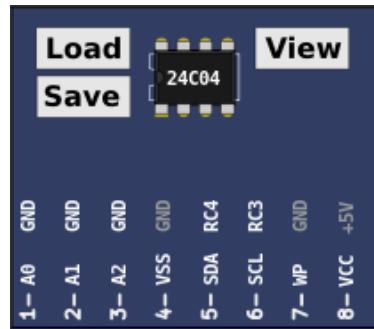
This part are formed by sixteen jumper wires. Each jumper has one input and one output. The jumper input must be connected to one pin output, the jumper output can be connected to multiple pin inputs. The jumper can be used to connect microcontroller pins or make connection between spare parts pins.



[Examples](#)

10.3.7 MEM 24CXXX

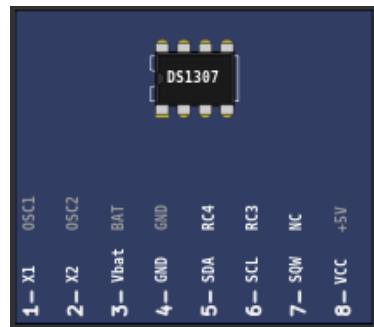
It is a 24CXXX serial I2C EEPROM part. There are support to the models 24C04 and 24C512.



[Examples](#)

10.3.8 RTC ds1307

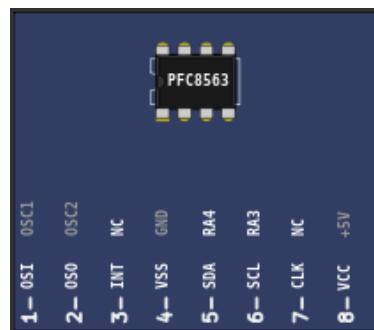
This part is a ds1307 real time clock with serial I2C interface.



[Examples](#)

10.3.9 RTC pfc8563

This part is a pfc8563 real time clock with serial I2C interface.



[Examples](#)

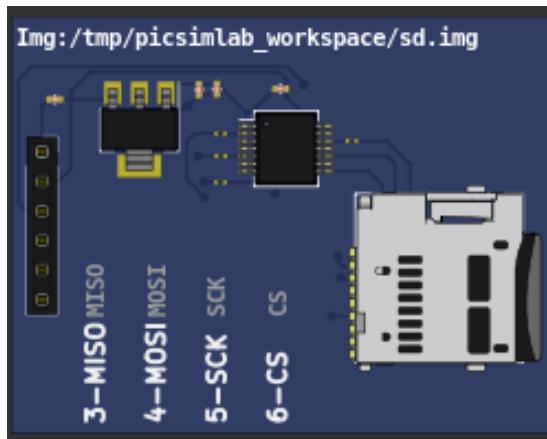
10.3.10 SD Card

This part is a SD Card shield. It's necessary set one sd card file image before use it.
(Click on SD card connector to open file dialog)

On Linux one empty image can be created with this command:

```
dd if=/dev/zero of=sd.img bs=1M count=32
```

This empty image can be used with raw sd card access, to work with FAT file system
the image need to be formatted before the use. (using [SdFormatter.ino](#) for example)

[Examples](#)

10.3.11 Temperature System

This part is a temperature control system. The temperature control system consists of a heating resistor, an LM35 temperature sensor, a cooler and an infrared tachometer.

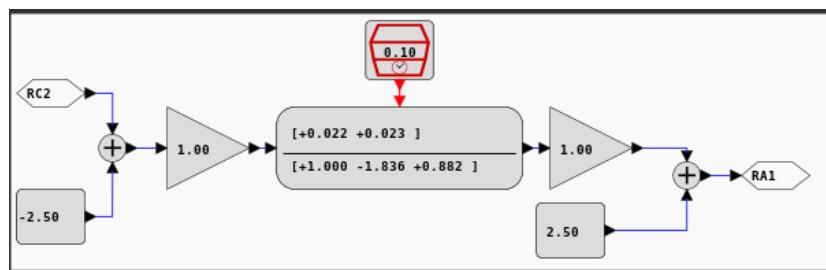


[Examples](#)

10.4 Virtual

10.4.1 D. Transfer Function

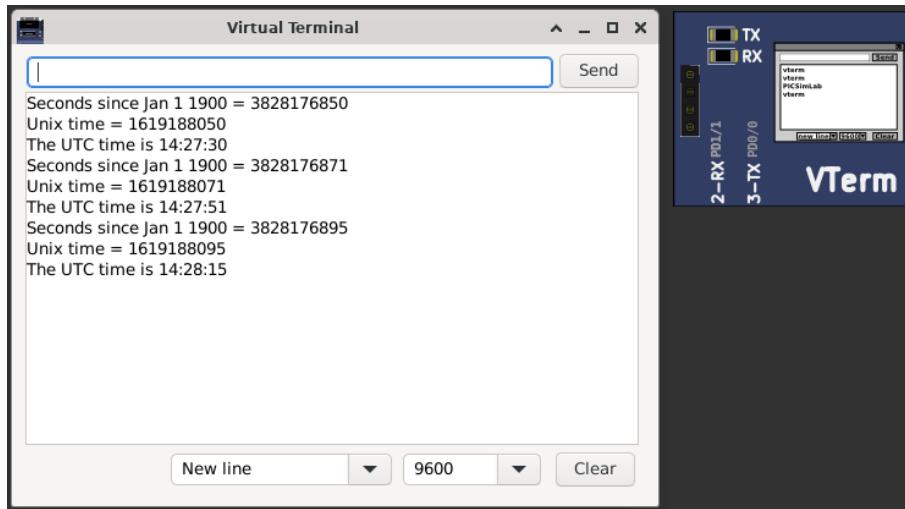
This is a discrete transfer function mathematical model.



[Examples](#)

10.4.2 IO Virtual Term

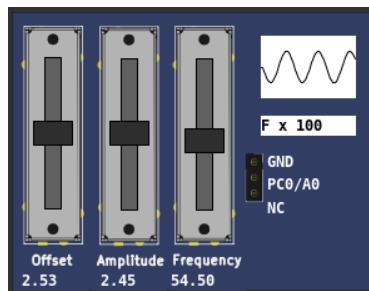
This part is a virtual serial terminal. This part can be used to read and write RX/TX pins UART signals. This part don't need the use or install of virtual serial ports on computer. Clik on terminal picture to open the terminal window.



Examples

10.4.3 Signal Generator

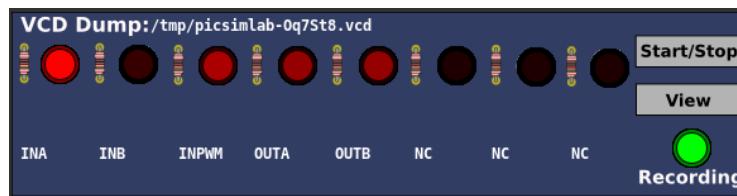
This part is a virtual signal generator with support for sine, square and triangular waves generation with amplitude and frequency adjustment.

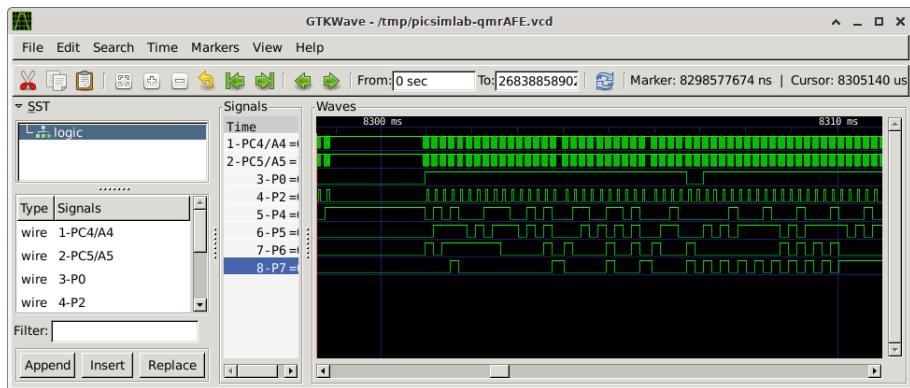


Examples

10.4.4 VCD Dump

This part is a digital value file dump recorder. The file can be visualized with gtkwave.

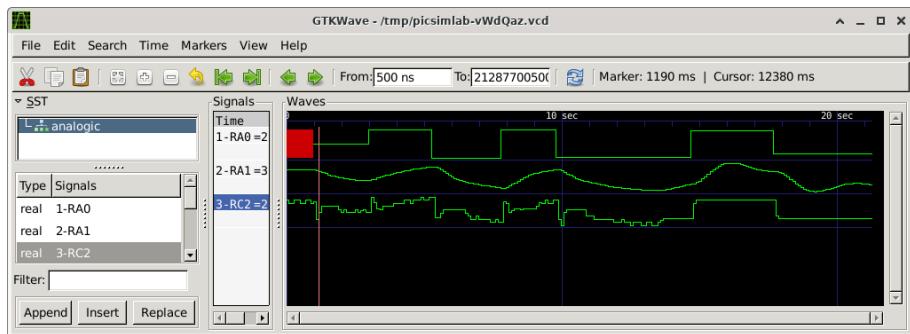
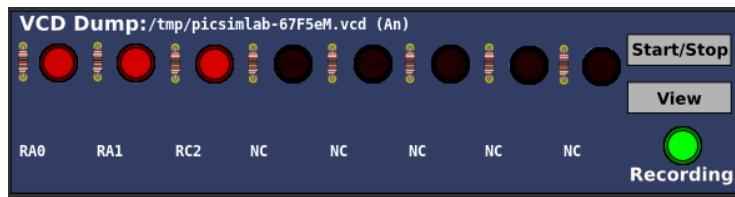




Examples

10.4.5 VCD Dump (Analogic)

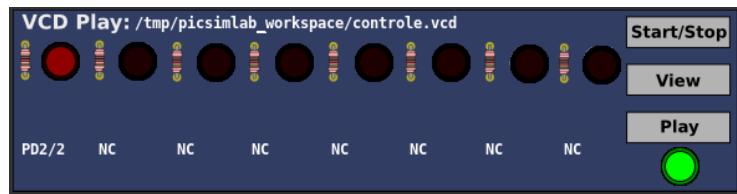
This part is a analog value file dump recorder. The file can be visualized with gtkwave.



Examples

10.4.6 VCD Play

This part play a VCD file saved from VCD Dump part.



Examples

Chapter 11

Troubleshooting

The simulation in PICSimLab consists of 3 parts:

- The microcontroller program
- Microcontroller simulation (made by [picsim](<https://github.com/lcgamboa/picsim>) and [simavr](<https://github.com/buserror/simavr>))
- Simulation of boards and parts

When a problem occurs it is important to detect where it is occurring.

One of the most common problems is the error in the microcontroller program. Before creating an issue, test your code on a real circuit (even partially) to make sure the problem is not there.

Errors in the microcontroller simulation can be detected using code debugging. Any instruction execution or peripheral behavior outside the expected should be reported in the project of simulator used ([picsim](<https://github.com/lcgamboa/picsim>) or [simavr](<https://github.com/buserror/simavr>)).

If the problem is not in either of the previous two options, the problem is probably in PICSimLab. A good practice is to send a source code together with a PICSimLab workspace (.pzw file) to open the issue about the problem.

Chapter 12

Use with MPLABX

Use with MPLABX to program and Debug

12.1 Installing the Necessary Tools

12.1.1 Install MPLABX IDE and XC8 Compiler

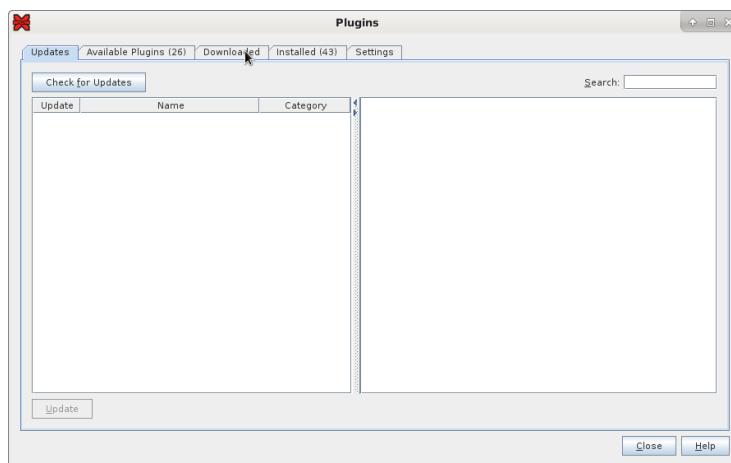
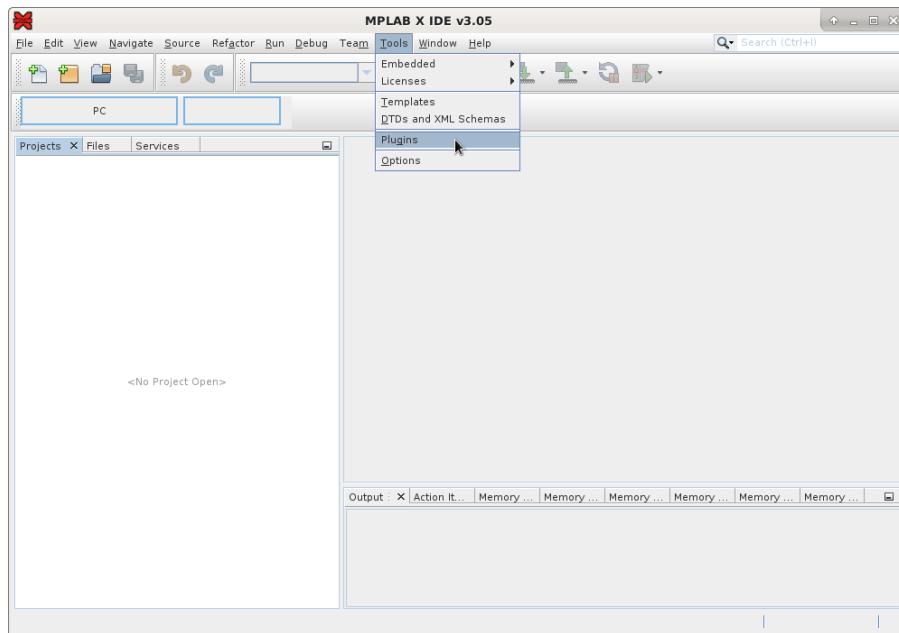
Links for download [MPLABX IDE](#) and [XC8 Compiler](#) installers. Download and install.

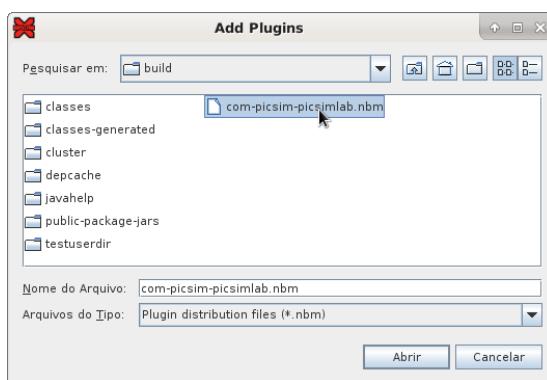
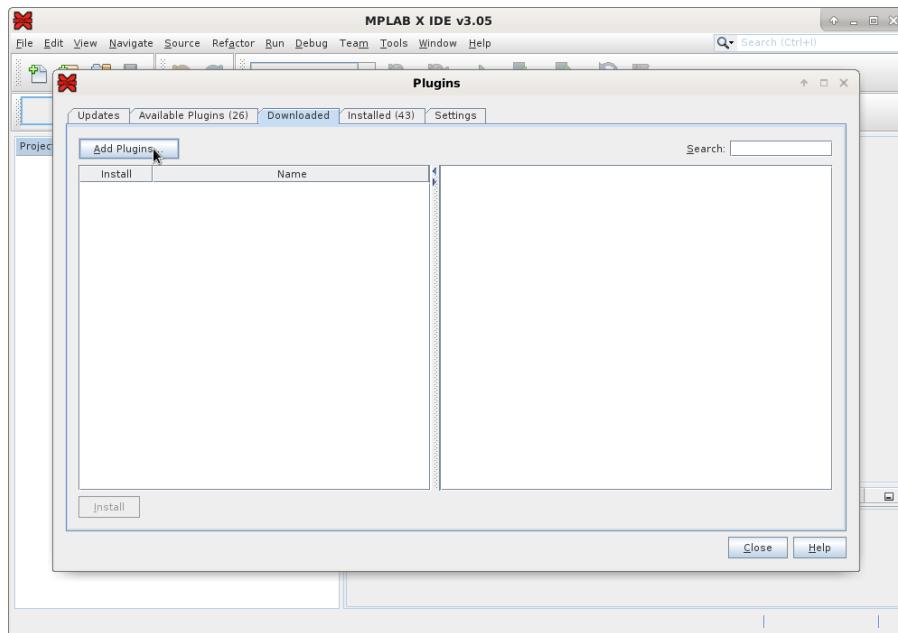
12.1.2 Install PICsimLab

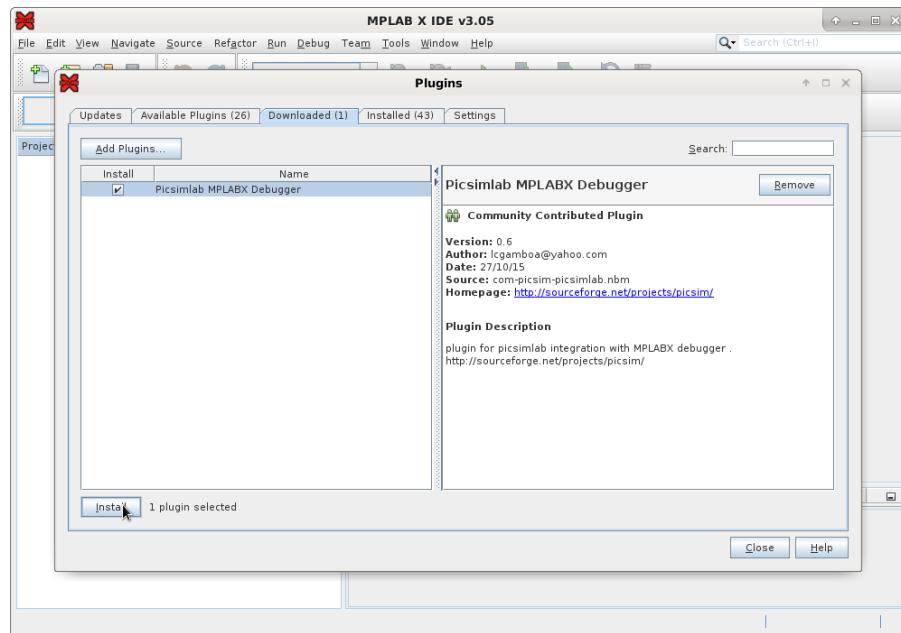
Link for download [PICsimLab-0.6](#) installer. Download and install

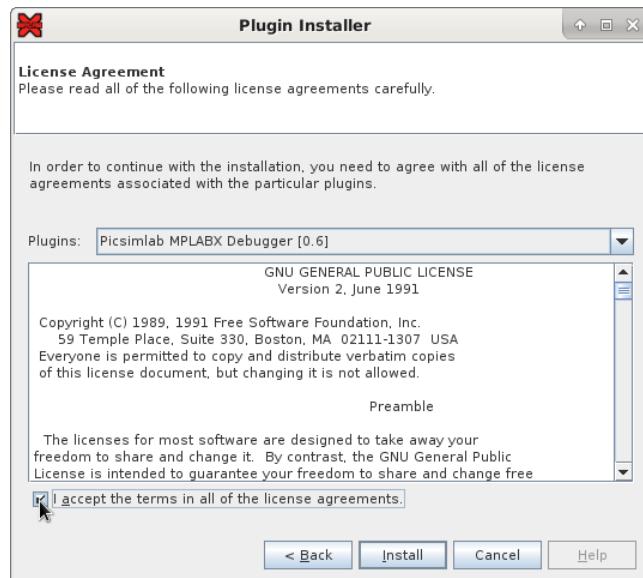
12.1.3 How to Install PicsimLab MPLABX Debugger plugin

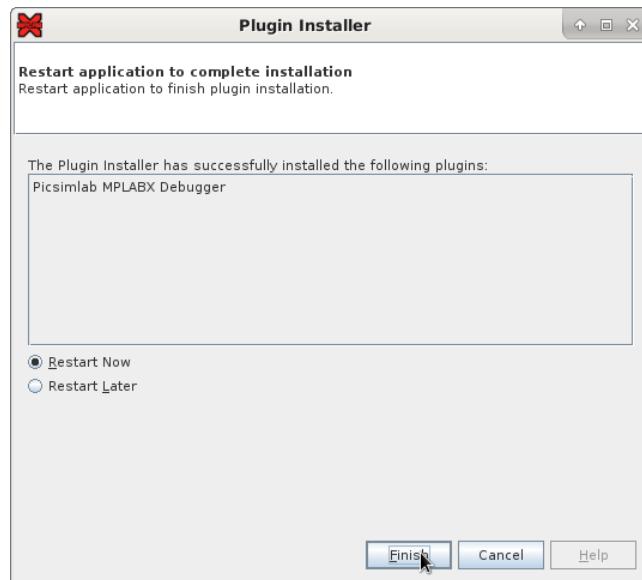
Link for download [PicsimLab MPLABX Debugger plugin \(com-picsim-picsimlab.nbm\)](#)





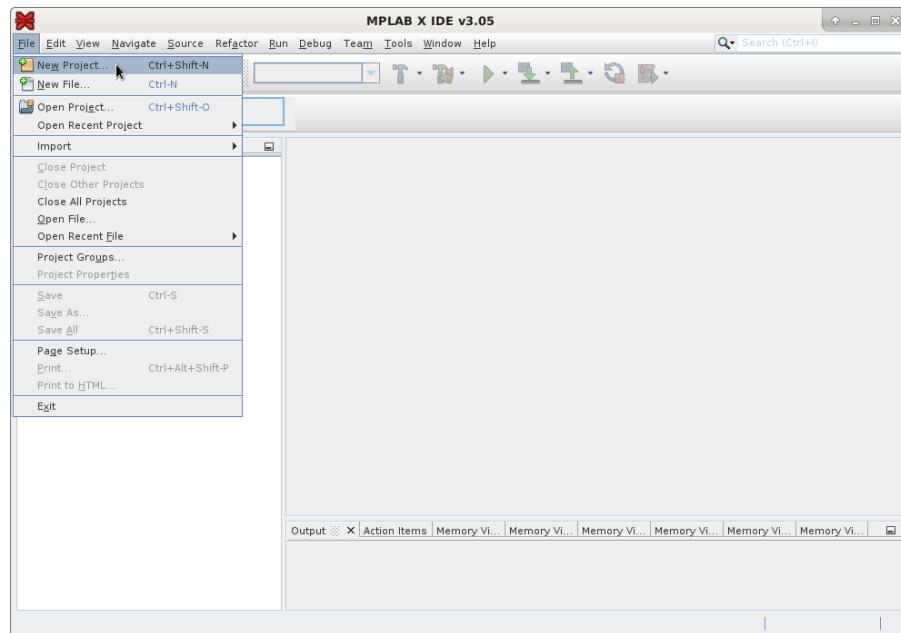


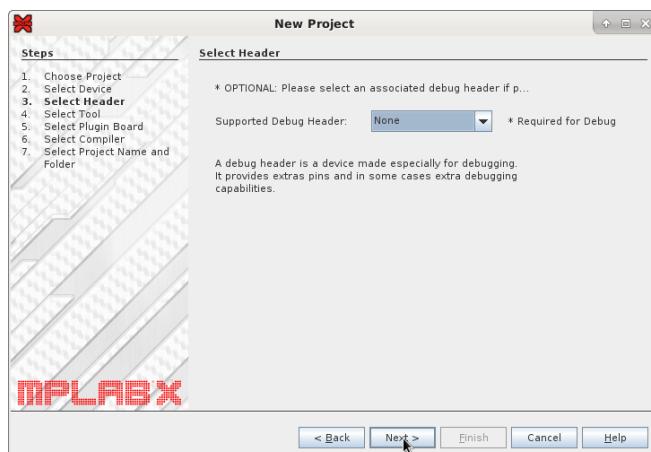
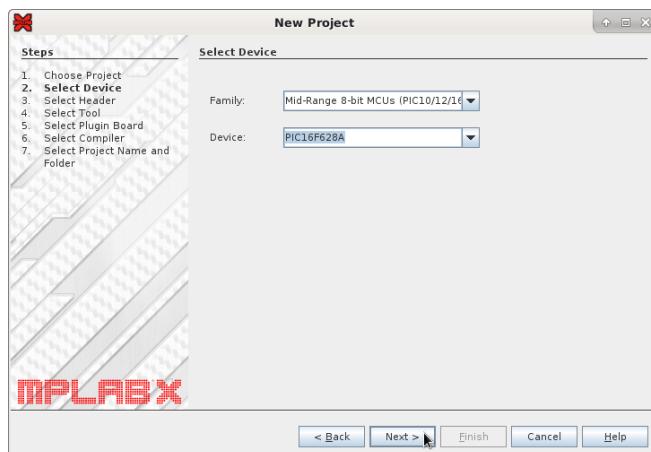
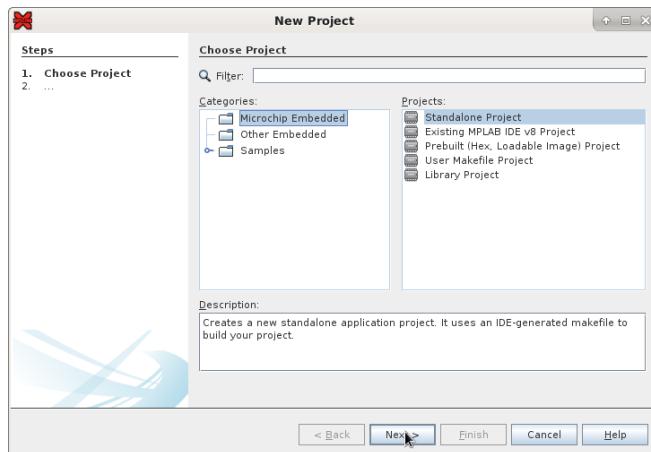


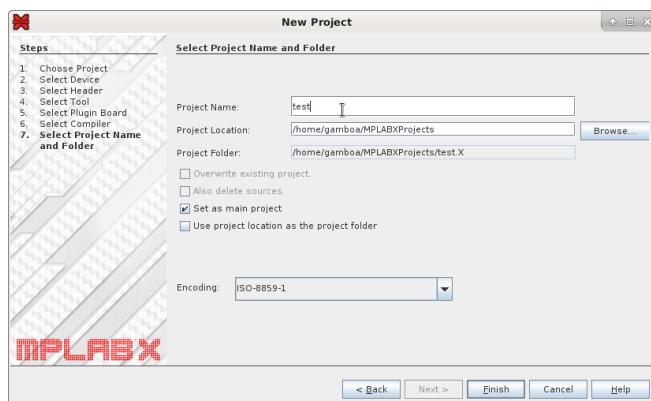
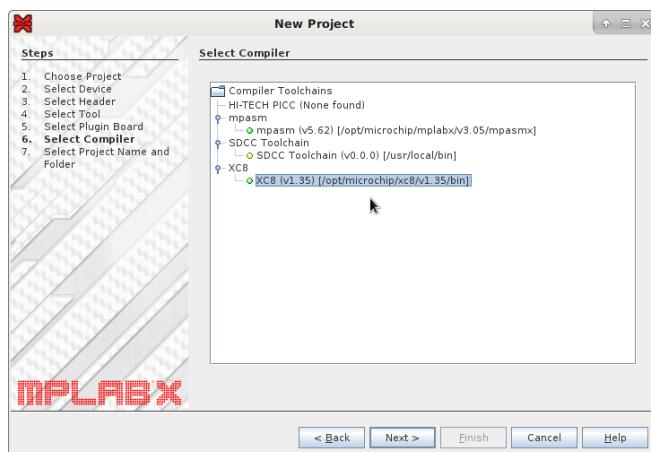
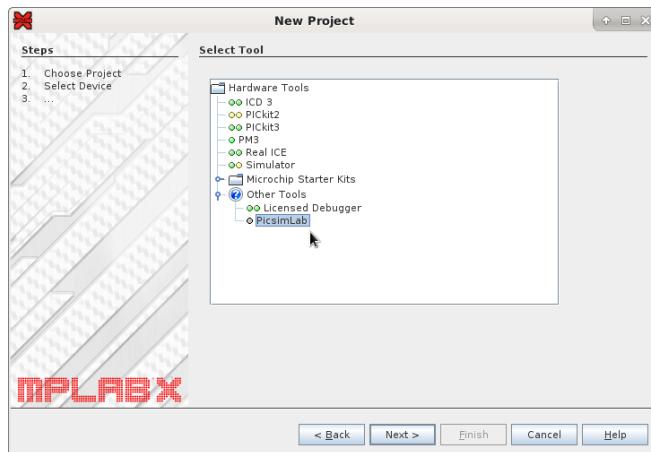


12.2 Configuring a New Project in MPLABX

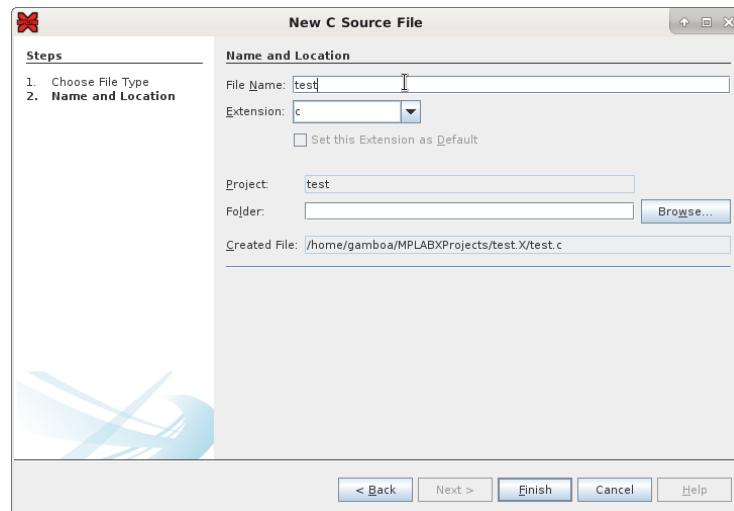
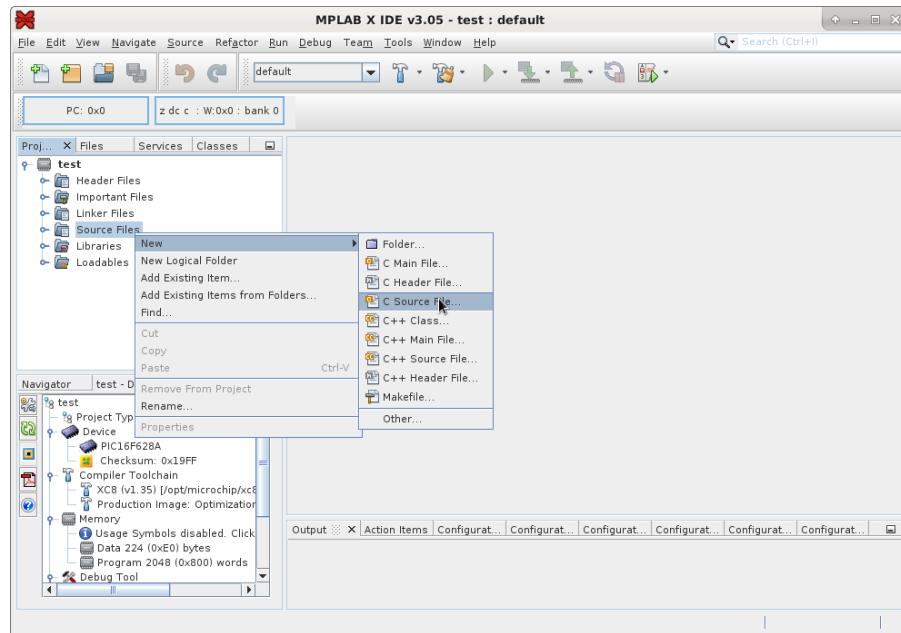
12.2.1 Project Creation



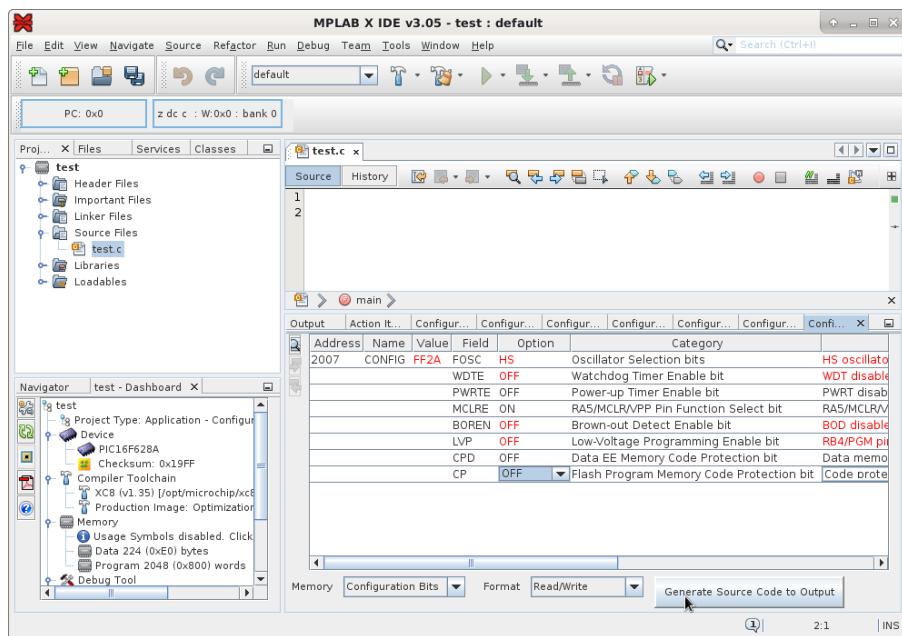
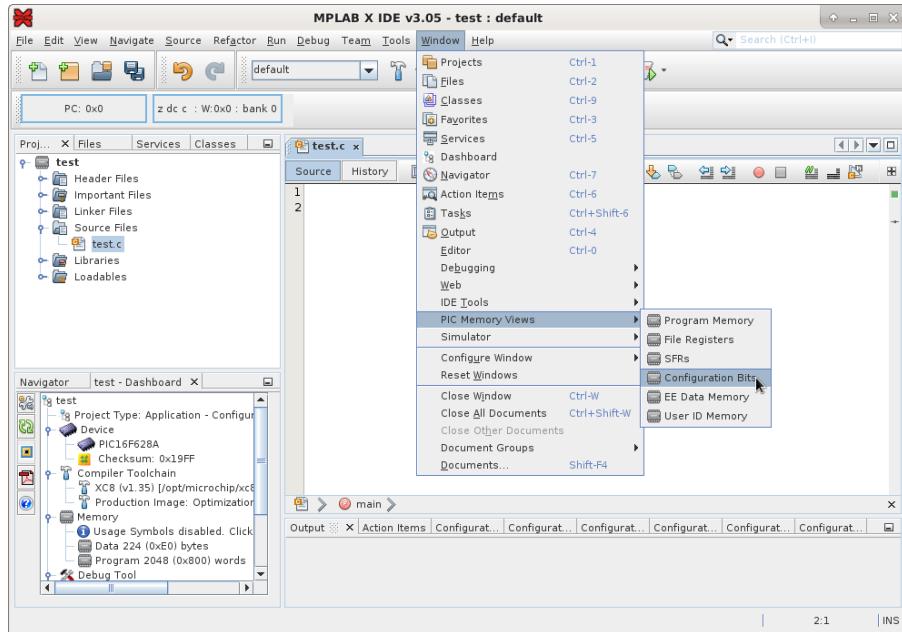


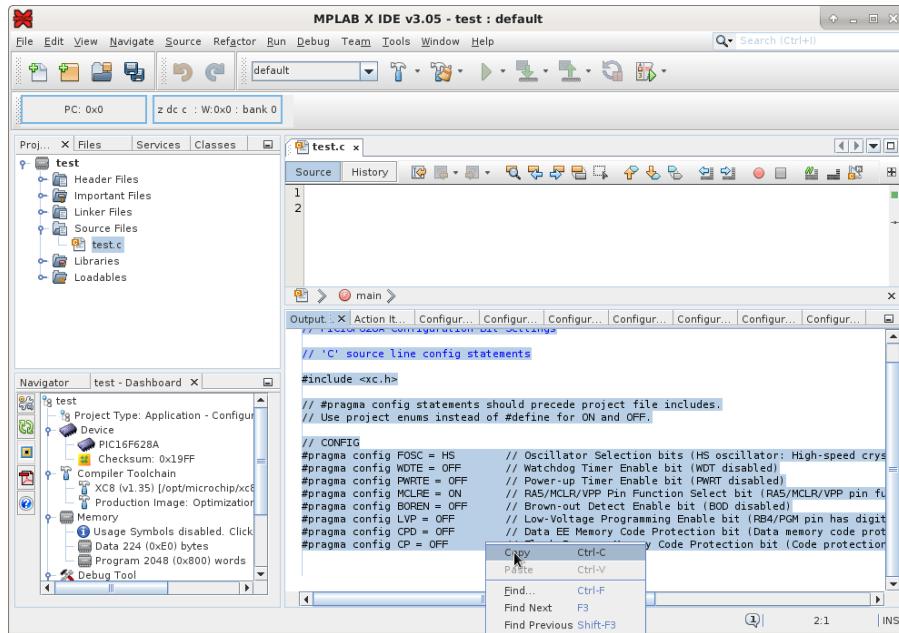


12.2.2 File Creation



12.2.3 PIC Configuration Bits

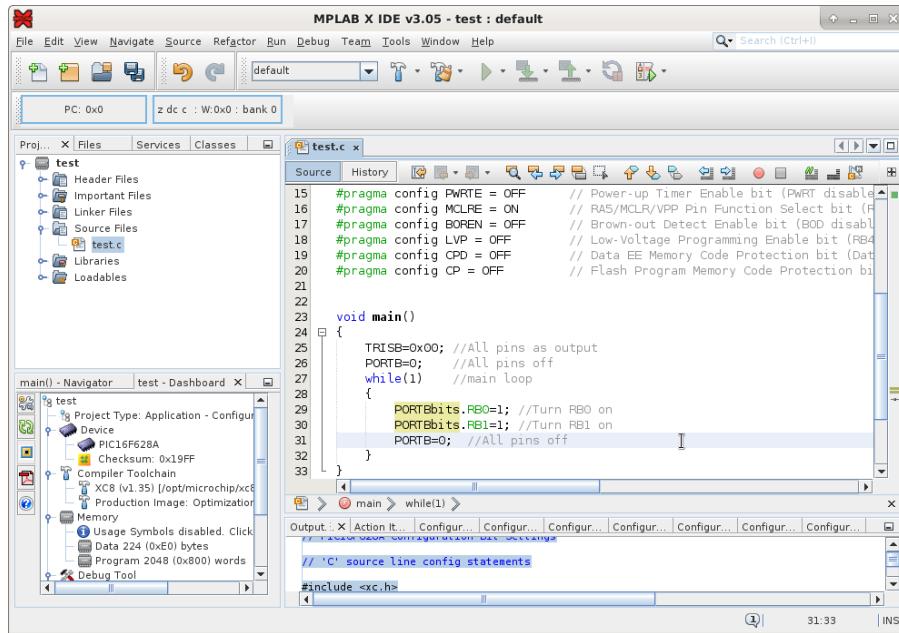




12.2.4 Code Example

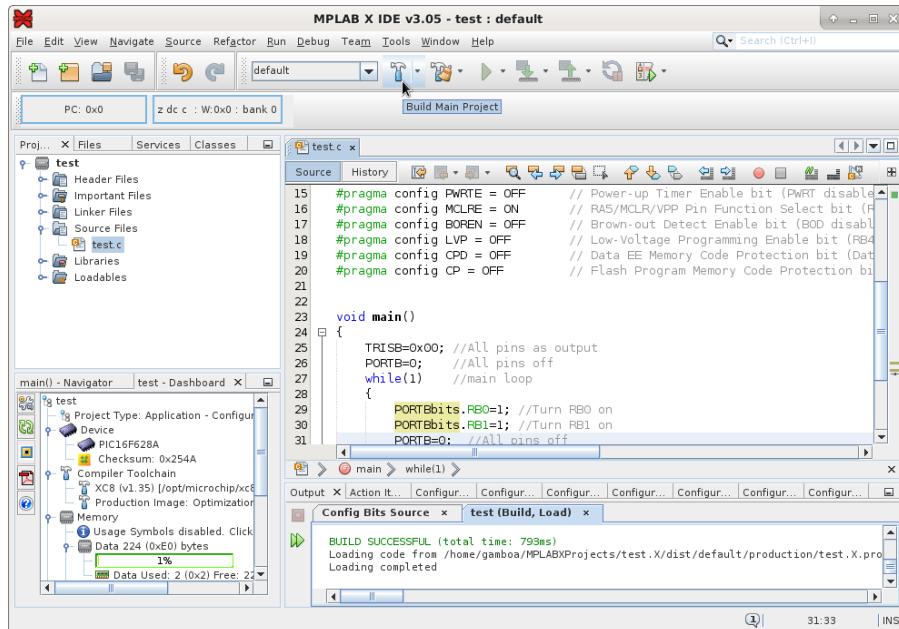
Paste the configuration and this simple code example in test.c:

```
void main()
{
    TRISB=0x00; //All pins as output
    PORTB=0;    //All pins off
    while(1)    //main loop
    {
        PORTBbits.RB0=1; //Turn RB0 on
        PORTBbits.RB1=1; //Turn RB1 on
        PORTB=0;    //All pins off
    }
}
```



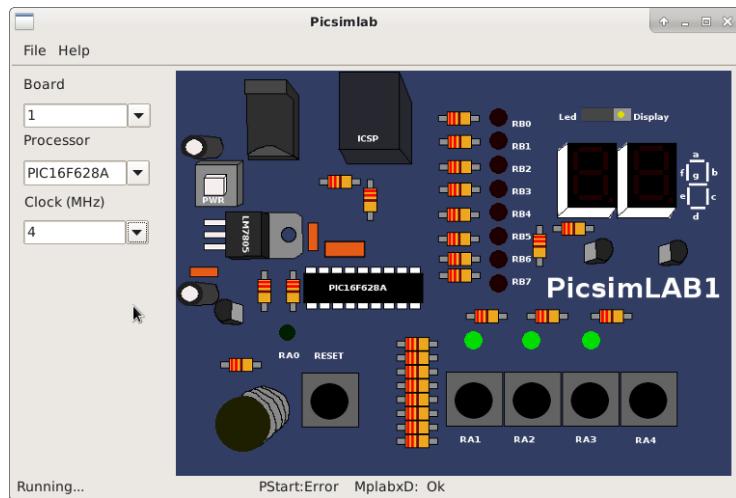
12.2.5 Building the Project

Use the **Build** button and wait for the message “**BUILD SUCCESSFUL**”.



12.3 Program and Debug PICsimLab With MPLABX

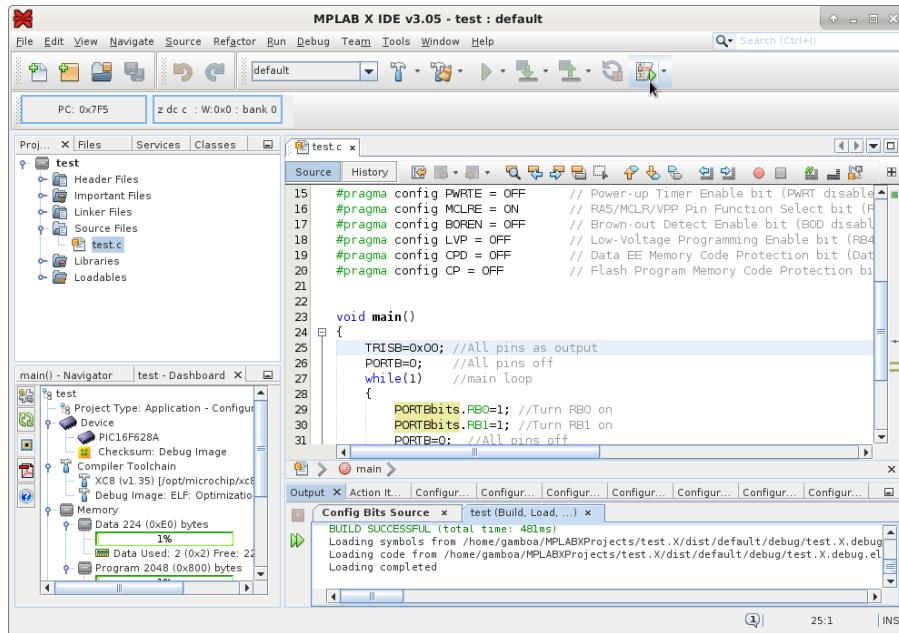
12.3.1 Starting PICsimLab



The plugin connect to Picsimlab through a TCP socket using port 1234, and you have to allow the access in the firewall. Verify in the PICsimLab statusbar the message “MplabxD: Ok”. It’s show debugger server state.

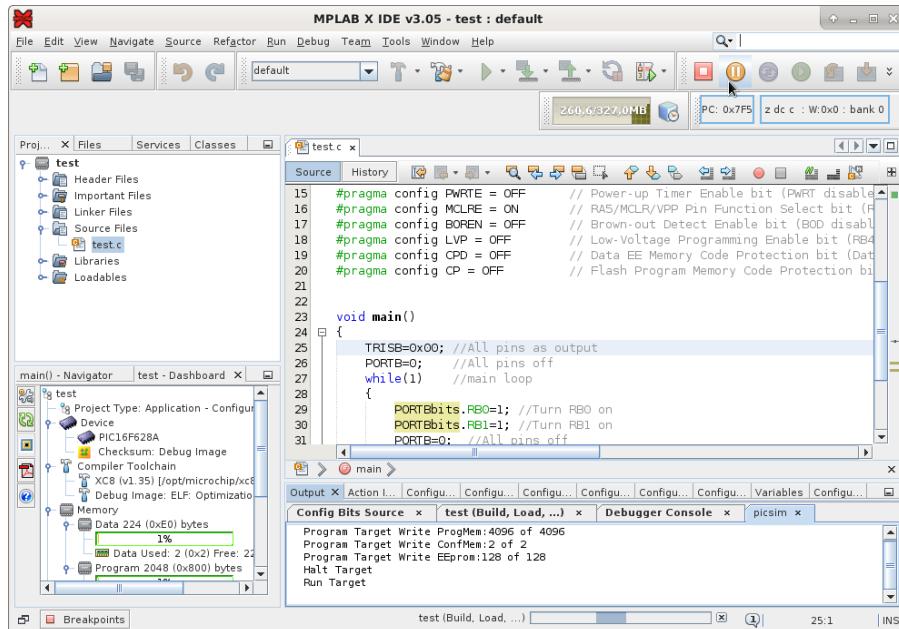
12.3.2 Programming PICsimLab

Use the **Debug** button to programming PICsimLab.



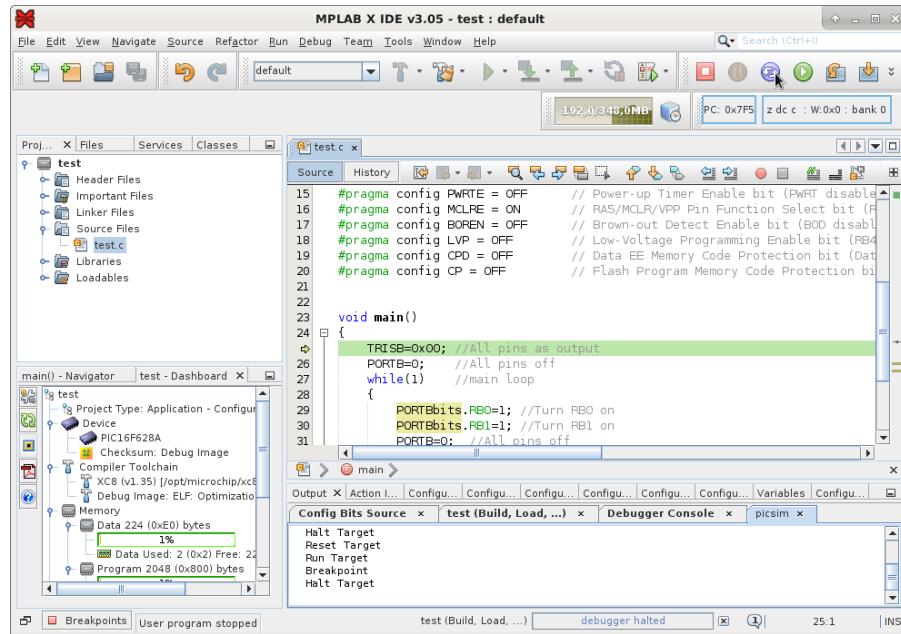
12.3.3 Pausing the Program

Use the **Pause** button to stop the program and inspect the code and memory.



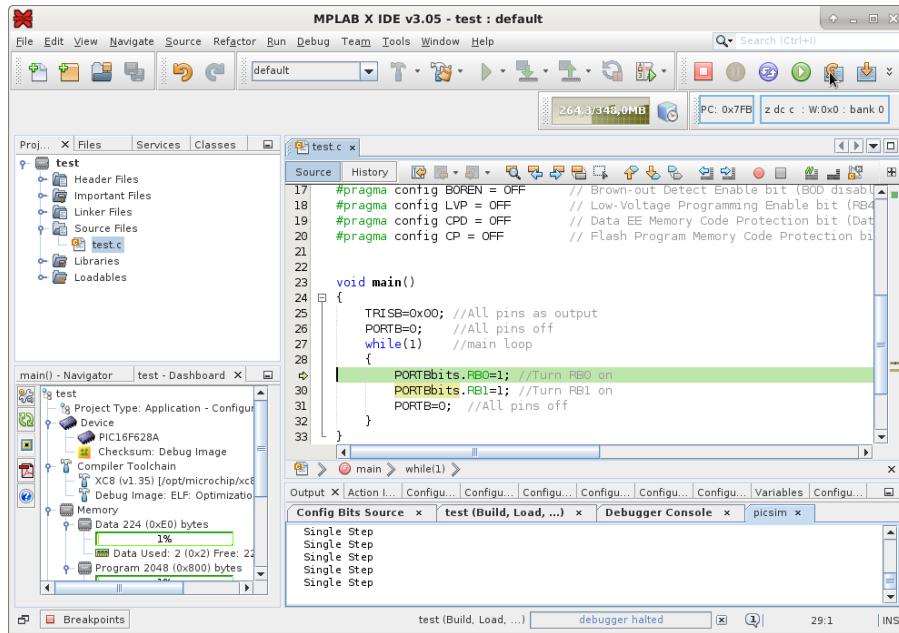
12.3.4 Restarting the Program

Use the **Restart** button to restart the program.

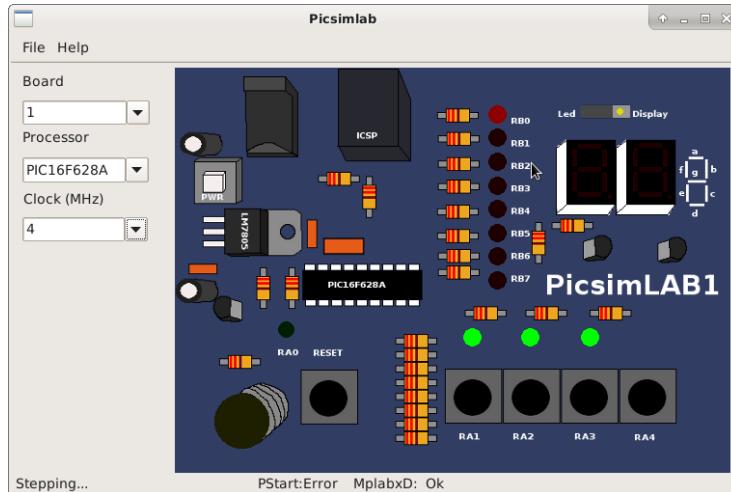


12.3.5 Running Step by Step

Use the **Step** or **Step Over** button to run the program step by step.

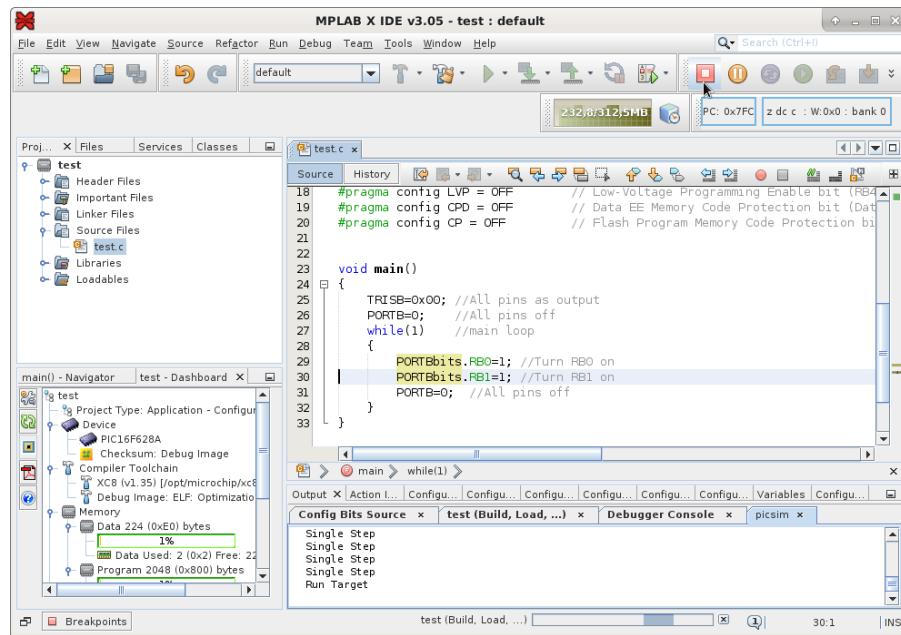


See in the PICsimLab the changes of each step.



12.3.6 Stopping Debugger

Use the **Stop** button to turn off the MPLABX debugger. The program continues running in PICsimLab after MPLABX debugger is stopped.



12.4 This Tutorial in Video

Link for Youtube video version of this tutorial: [How to use MPLABX to program and debug PicsimLab 0.6](#)

Chapter 13

Creating New Boards

13.1 How to Compile PICsimLab

13.1.1 In Debian Linux and derivatives

```
git clone https://github.com/lcgamboa/picsimlab.git  
cd picsimlab  
./picsimlab_build_all_and_deps.sh
```

To build experimental version use the argument “exp” with the *picsimlab_build_all_and_deps.sh* script

13.1.2 Cross-compiling for windows

For Windows 64 bits version from Debian Linux and derivatives or [WSL](#) (Windows Subsystem for Linux) on win10

```
git clone https://github.com/lcgamboa/picsimlab.git  
cd picsimlab  
./picsimlab_build_w64.sh
```

For 32 bits version:

```
git clone https://github.com/lcgamboa/picsimlab.git  
cd picsimlab  
./picsimlab_build_w32.sh
```

To build experimental version use the argument “exp” with the scripts.

13.2 Creating a New Board

The first step is get the schematic and all information about the board hardware. The second step is the creation of five files in PICSimLab dir (consider replace the 'x' of

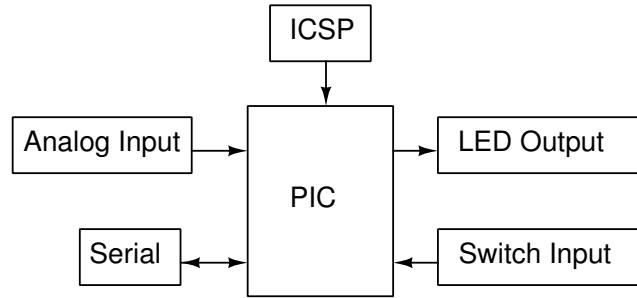
board_x for a number or name in your case):

- Board Picture (share/boards/x/board.png);
- Board input map (share/boards/x/input.map);
- Board output map (share/boards/x/output.map);
- Board header (src/boards/board_x.h);
- Board C++ code (src/boards/board_x.cc);

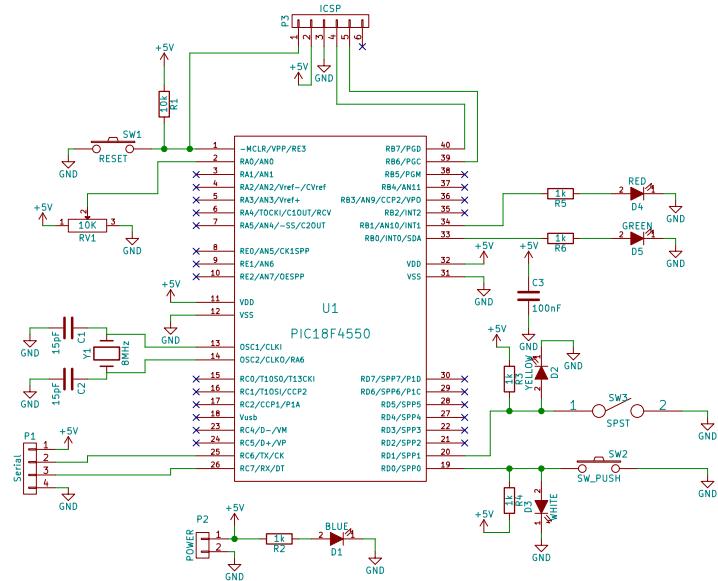
The third and last step is recompiling PICSimLab with new board support.

13.2.1 Board Hardware and Schematic

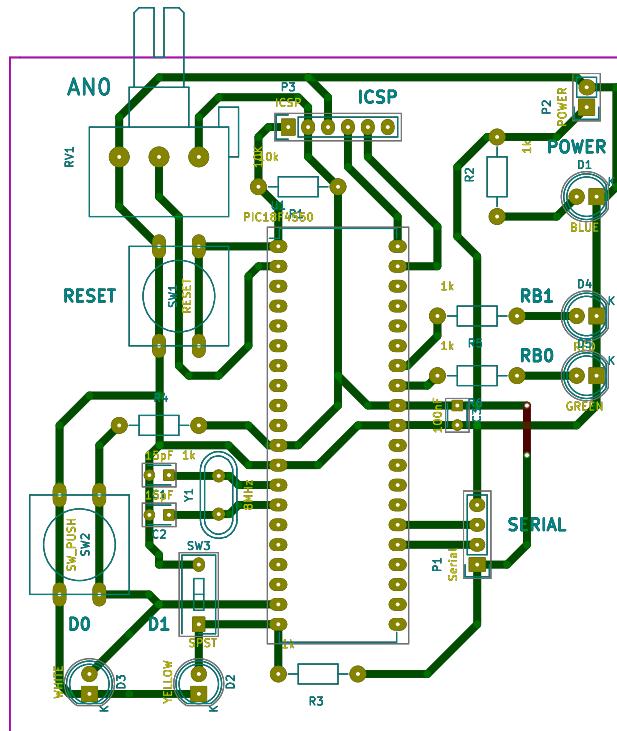
For this tutorial, the board created have the hardware shown in diagram below:



The schematic for the tutorial board made in [Kicad](#).

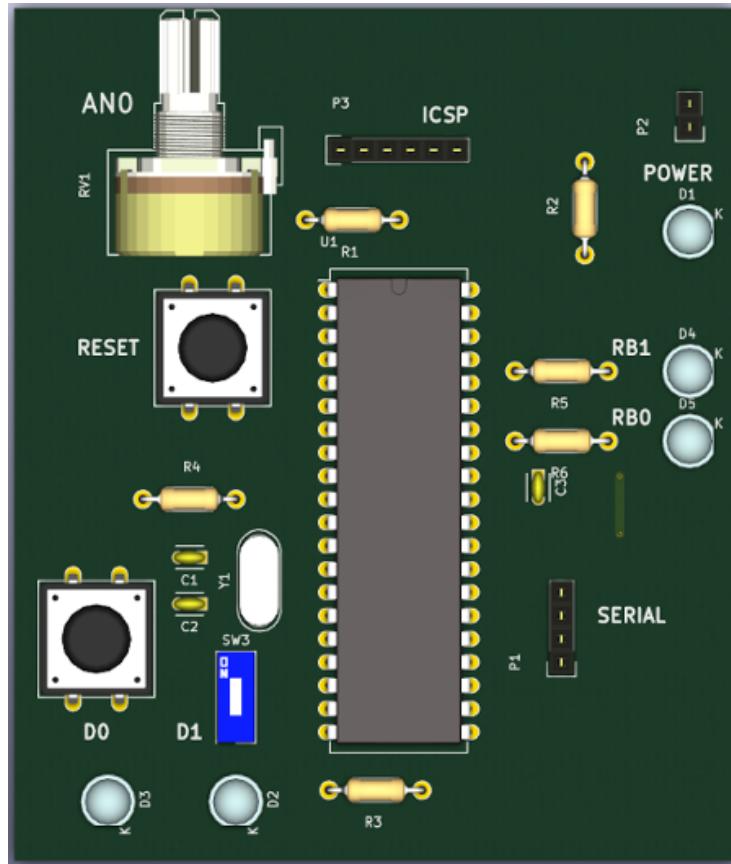


And the PCB layout was made in [Kicad](#) too. The PCB is not necessary if you have a real board.



13.2.2 Board Picture

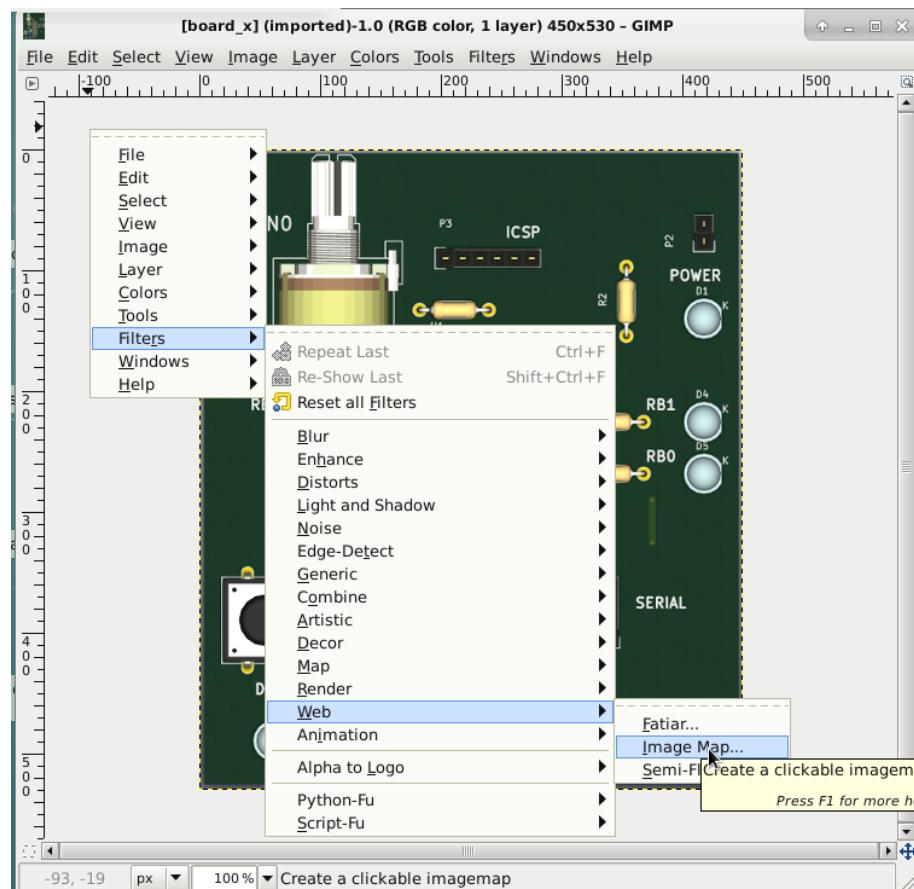
Because the real board of this tutorial never has been built, the board picture was taken from [Kicad](#) 3D viewer. The picture image is saved as “share/board/x/board.png”.



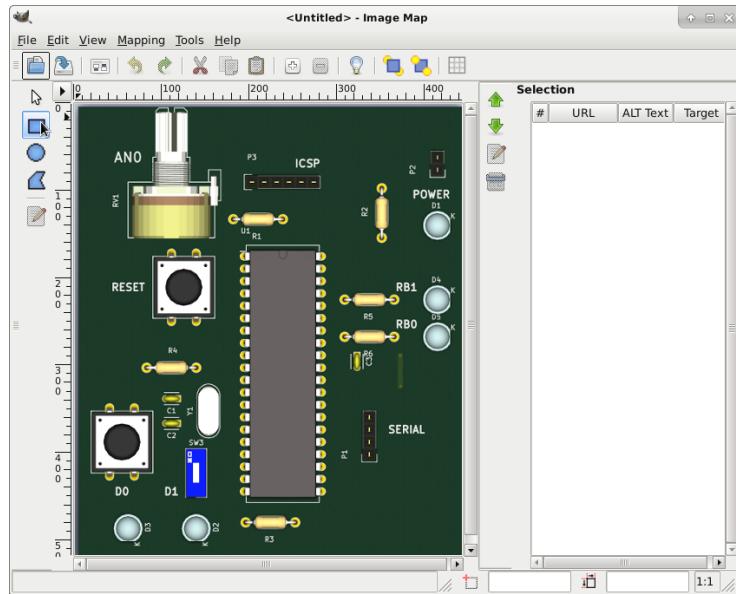
13.2.3 Picture maps

The PICSimLab use two type of image maps. The input map mark the areas in board picture which user can interact (by mouse click). The output map mark the areas in board picture to be redraw according simulator status. The picture maps used for PICSimLab are normal HTML image-map. They can be made by hand or using any software which can handle image maps. The original PICSimLab maps are made using [Gimp image editor](#).

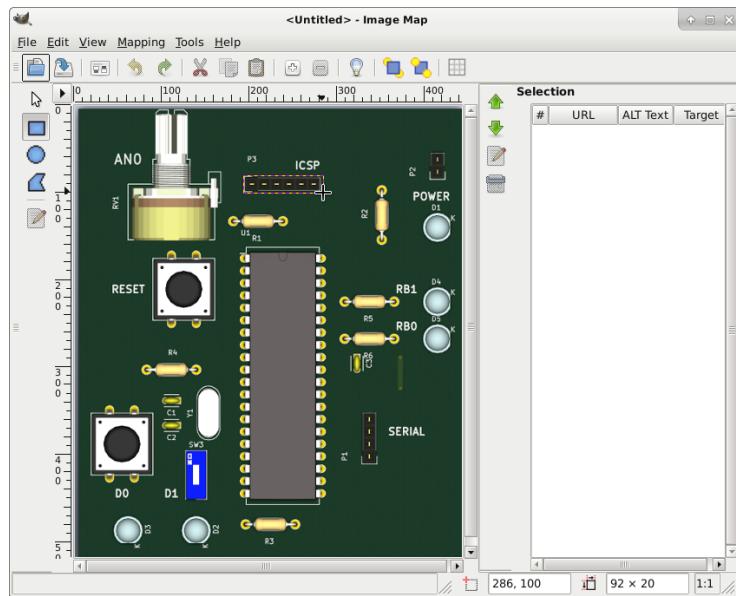
To start, in the GIMP, use the Filters->Web->Image Map to open image map editor window.



Then select rectangle or circle map on toolbar.



And mark the area in picture.



After area is select, in the settings windows select the link type for “Other”.



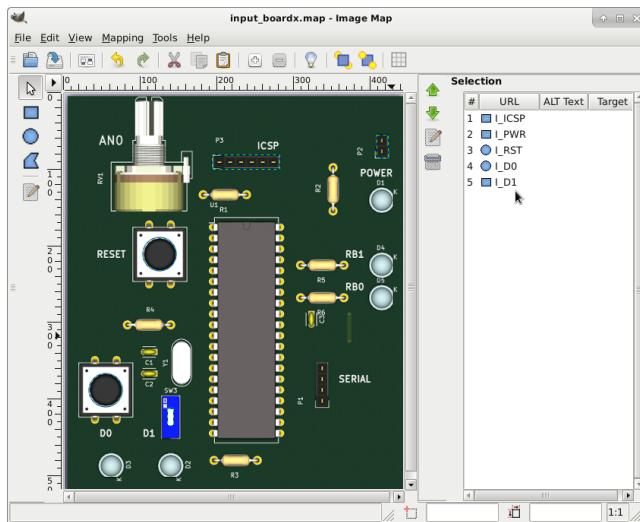
And write the name of area. The name must describe the area function on the board.



Input map

For this tutorial board, five input areas are marked:

- I_ICSP - where user click to load hexfile.
- I_PWR - where user click to turn on/off the board.
- I_RST - Button to reset board.
- I_D0 - Button connected in RD0.
- I_D1 - Switch connected in RD1.



Input map generated by Gimp image map editor and saved as “share/boards/x/input.map”.

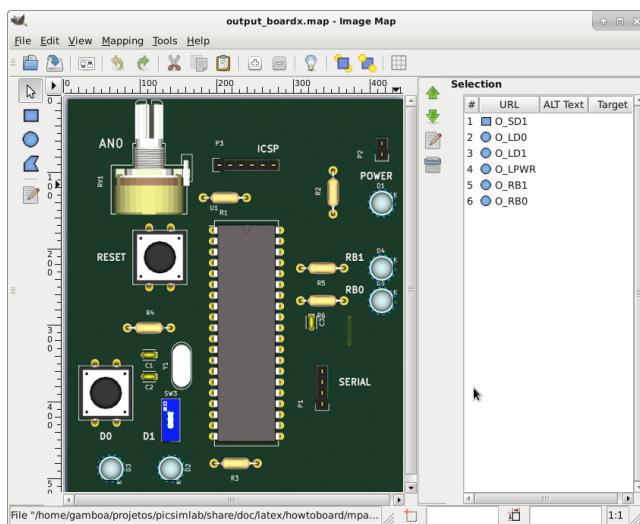
```

1 
2
3 <map name="map">
4 <!-- #$:-Image map file created by GIMP Image Map plug-in -->
5 <!-- #$:-GIMP Image Map plug-in by Maurits Rijk -->
6 <!-- #$:-Please do not edit lines starting with "#" -->
7 <!-- #$VERSION:2.3 -->
8 <!-- #$AUTHOR:lcgambao@yahoo.com -->
9 <area shape="rect" coords="194,80,283,99" href="I_ICSP" />
10 <area shape="rect" coords="411,54,426,84" href="I_PWR" />
11 <area shape="circle" coords="125,211,22" href="I_RST" />
12 <area shape="circle" coords="54,390,22" href="I_D0" />
13 <area shape="rect" coords="135,414,143,436" href="I_D1" />
14 </map>
```

Output map

For this tutorial board, six output areas are marked:

- O_SD1 - draw the switch on/off.
- O_LD0 - draw LED connected in button.
- O_LD1 - draw LED connected in switch.
- O_LPWR - draw power LED indicator.
- O_RB0 and O_RB1 - draw LEDs connected in RB0 and RB1.



Output map generated by Gimp image map editor and saved as “share/boards/x/output.map”.

```

1 
2
3 <map name="map">
4 <!-- #$-:Image map file created by GIMP Image Map plug-in -->
5 <!-- #$-:GIMP Image Map plug-in by Maurits Rijk -->
6 <!-- #$-:Please do not edit lines starting with "#$" -->
7 <!-- #$VERSION:2.3 -->
8 <!-- #$AUTHOR:lcgamboa@yahoo.com -->
9 <area shape="rect" coords="135,414,143,436" href="O_SD1" />
10 <area shape="circle" coords="61,489,17" href="O_LD0" />
11 <area shape="circle" coords="140,489,17" href="O_LD1" />
12 <area shape="circle" coords="418,140,17" href="O_LPWR" />
13 <area shape="circle" coords="418,226,17" href="O_RB1" />
14 <area shape="circle" coords="418,269,17" href="O_RB0" />
15 </map>
```

The kicad project files can be download from github [PICSIMLab repository](#).

13.2.4 Board code

The header file and c++ code file with comments are listed in the next two subsections. This files control the behavior of board in simulator.

board_x.h

[board_x.h online file](#).

[board_x.h online doxygen version](#).

```

1  /* ##### */
2
3  PICsimLab - PIC laboratory simulator
4
5  #####
6
7  Copyright (c) : 2015-2020 Luis Claudio Gambôa Lopes
8
9  This program is free software; you can redistribute it and/or modify
10  it under the terms of the GNU General Public License as published by
11  the Free Software Foundation; either version 2, or (at your option)
12  any later version.
13
14  This program is distributed in the hope that it will be useful,
15  but WITHOUT ANY WARRANTY; without even the implied warranty of
16  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17  GNU General Public License for more details.
18
19  You should have received a copy of the GNU General Public License
20  along with this program; if not, write to the Free Software
21  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
22
23  For e-mail suggestions : lcgamboa@yahoo.com
24  ##### */
25
26 #ifndef BOARD_X_H
27 #define      BOARD_X_H
28
29 #include<lxradi.h>
30
31 #include "board_picsim.h"
32
33 //new board class must be derived from board class defined in board.h
34 class cboard_x:public board_picsim
35 {
36     private:
37         int p_BT1;           //first board switch in RD0
38         int p_BT2;           //second board switch in RD1
39

```

```

40     //controls to be added in simulator window
41     CScroll *scroll1; //scroll for analog input A0
42     CGauge *gauge1;   //gauge to show mean value of R0
43     CGauge *gauge2;   //gauge to show mean value of R1
44     CLabel *label1;   //label of scroll A0
45     CLabel *label2;   //label of gauge R0
46     CLabel *label3;   //label of gauge R1
47
48 public:
49     //Constructor called once on board creation
50     cboard_x(void);
51     //Destructor called once on board destruction
52     ~cboard_x(void);
53     //Return the about info of board
54     String GetAboutInfo(void) {return lxT("L.C. Gamboa \n <lcgamboa@yahoo.com>");}
55     //Called every 100ms to draw board
56     void Draw(CDraw *draw, double scale);
57     void Run_CPU(void);
58     //Return a list of board supported microcontrollers
59     String GetSupportedDevices(void) {return lxT("PIC18F4550,PIC16F877A,");}
60     //Return the filename of board picture
61     String GetPictureFileName(void) {return lxT("x/board.png");}
62     //Return the filename of board picture input map
63     String GetInputMapFile(void) {return lxT("x/input.map");}
64     //Return the filename of board picture output map
65     String GetOutputMapFile(void) {return lxT("x/output.map");}
66     //Reset board status
67     void Reset(void);
68     //Event on the board
69     void EvMouseButtonPress(uint button, uint x, uint y, uint state);
70     //Event on the board
71     void EvMouseButtonRelease(uint button, uint x, uint y, uint state);
72     //Event on the board
73     void EvKeyPress(uint key, uint mask);
74     //Event on the board
75     void EvKeyRelease(uint key, uint mask);
76     void EvOnShow(void) {};
77     //Called every 1s to refresh status
78     void RefreshStatus(void);
79     //Called to save board preferences in configuration file
80     void WritePreferences(void);
81     //Called whe configuration file load preferences
82     void ReadPreferences(char *name, char *value);
83     //return the input ids numbers of names used in input map
84     unsigned short get_in_id(char * name);
85     //return the output ids numbers of names used in output map
86     unsigned short get_out_id(char * name);
87 };

```

```
88      #endif          /* BOARD_X_H */  
89
```

board_x.cc

board_x.cc online file.
board_x.cc online doxygen version.

```

1  /* ##### */
2
3  PICsimLab - PIC laboratory simulator
4
5  #####
6
7  Copyright (c) : 2015-2020 Luis Claudio Gambôa Lopes
8
9  This program is free software; you can redistribute it and/or modify
10 it under the terms of the GNU General Public License as published by
11 the Free Software Foundation; either version 2, or (at your option)
12 any later version.
13
14 This program is distributed in the hope that it will be useful,
15 but WITHOUT ANY WARRANTY; without even the implied warranty of
16 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17 GNU General Public License for more details.
18
19 You should have received a copy of the GNU General Public License
20 along with this program; if not, write to the Free Software
21 Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
22
23 For e-mail suggestions : lcgamboa@yahoo.com
24 ##### */
25
26 //include files
27 #include "../picsimlab1.h"
28 #include "../picsimlab4.h" //Oscilloscope
29 #include "../picsimlab5.h" //Spare Parts
30 #include "board_x.h"
31
32 /* ids of inputs of input map*/
33 enum
34 {
35     I_ICSP, //ICSP connector
36     I_PWR, //Power button
37     I_RST, //Reset button
38     I_D0, //RD0 push button
39     I_D1 //RD1 switch
40 };
41
42 /* ids of outputs of output map*/
43 enum
44 {

```

```

45     O_SD1, //switch position (On/Off)
46     O_LD0, //LED on RD0 push button
47     O_LD1, //LED on RD1 switch
48     O_LPWR, //Power LED
49     O_RB0, //LED on RB0 output
50     O_RB1 //LED on RB1 output
51   };
52   //return the input ids numbers of names used in input map
53
54   unsigned short
55   cboard_x::get_in_id(char * name)
56   {
57     if (strcmp (name, "I_ICSP") == 0) return I_ICSP;
58     if (strcmp (name, "I_PWR") == 0) return I_PWR;
59     if (strcmp (name, "I_RST") == 0) return I_RST;
60     if (strcmp (name, "I_D0") == 0) return I_D0;
61     if (strcmp (name, "I_D1") == 0) return I_D1;
62
63     printf ("Error input '%s' don't have a valid id! \n", name);
64     return -1;
65   }
66
67   //return the output ids numbers of names used in output map
68
69   unsigned short
70   cboard_x::get_out_id(char * name)
71   {
72
73     if (strcmp (name, "O_SD1") == 0) return O_SD1;
74     if (strcmp (name, "O_LD0") == 0) return O_LD0;
75     if (strcmp (name, "O_LD1") == 0) return O_LD1;
76     if (strcmp (name, "O_LPWR") == 0) return O_LPWR;
77     if (strcmp (name, "O_RB1") == 0) return O_RB1;
78     if (strcmp (name, "O_RB0") == 0) return O_RB0;
79
80     printf ("Error output '%s' don't have a valid id! \n", name);
81     return 1;
82   }
83
84   //Constructor called once on board creation
85
86   cboard_x::cboard_x(void)
87   {
88     Proc = "PIC18F4550"; //default microcontroller if none defined in preferences
89     ReadMaps (); //Read input and output board maps
90
91     //controls properties and creation
92     //scroll11

```

```
93 scroll1 = new CScroll ();
94 scroll1->SetFOwner (&Window1);
95 scroll1->SetName (lxT ("scroll1_px"));
96 scroll1->SetX (12);
97 scroll1->SetY (273 - 160);
98 scroll1->SetWidth (140);
99 scroll1->SetHeight (22);
100 scroll1->SetEnable (1);
101 scroll1->SetVisible (1);
102 scroll1->SetRange (100);
103 scroll1->SetPosition (50);
104 scroll1->SetType (4);
105 Window1.CreateChild (scroll1);
106 //gauge1
107 gauge1 = new CGauge ();
108 gauge1->SetFOwner (&Window1);
109 gauge1->SetName (lxT ("gauge1_px"));
110 gauge1->SetX (13);
111 gauge1->SetY (382 - 160);
112 gauge1->SetWidth (140);
113 gauge1->SetHeight (20);
114 gauge1->SetEnable (1);
115 gauge1->SetVisible (1);
116 gauge1->SetRange (100);
117 gauge1->SetValue (0);
118 gauge1->SetType (4);
119 Window1.CreateChild (gauge1);
120 //gauge2
121 gauge2 = new CGauge ();
122 gauge2->SetFOwner (&Window1);
123 gauge2->SetName (lxT ("gauge2_px"));
124 gauge2->SetX (12);
125 gauge2->SetY (330 - 160);
126 gauge2->SetWidth (140);
127 gauge2->SetHeight (20);
128 gauge2->SetEnable (1);
129 gauge2->SetVisible (1);
130 gauge2->SetRange (100);
131 gauge2->SetValue (0);
132 gauge2->SetType (4);
133 Window1.CreateChild (gauge2);
134 //label1
135 label1 = new CLabel ();
136 label1->SetFOwner (&Window1);
137 label1->SetName (lxT ("label1_px"));
138 label1->SetX (12);
139 label1->SetY (249 - 160);
140 label1->SetWidth (60);
```

```
141    label1->SetHeight (20);
142    label1->SetEnable (1);
143    label1->SetVisible (1);
144    label1->SetText (lxT ("AN0"));
145    label1->SetAlign (1);
146    Window1.CreateChild (label1);
147 //label1
148    label2 = new CLabel ();
149    label2->SetFOwner (&Window1);
150    label2->SetName (lxT ("label2_px"));
151    label2->SetX (12);
152    label2->SetY (306 - 160);
153    label2->SetWidth (60);
154    label2->SetHeight (20);
155    label2->SetEnable (1);
156    label2->SetVisible (1);
157    label2->SetText (lxT ("RB0"));
158    label2->SetAlign (1);
159    Window1.CreateChild (label2);
160 //label2
161    label3 = new CLabel ();
162    label3->SetFOwner (&Window1);
163    label3->SetName (lxT ("label3_px"));
164    label3->SetX (13);
165    label3->SetY (357 - 160);
166    label3->SetWidth (60);
167    label3->SetHeight (20);
168    label3->SetEnable (1);
169    label3->SetVisible (1);
170    label3->SetText (lxT ("RB1"));
171    label3->SetAlign (1);
172    Window1.CreateChild (label3);
173 }
174
175 //Destructor called once on board destruction
176
177 cboard_x::~cboard_x (void)
178 {
179 //controls destruction
180    Window1.DestroyChild (scroll1);
181    Window1.DestroyChild (gauge1);
182    Window1.DestroyChild (gauge2);
183    Window1.DestroyChild (label1);
184    Window1.DestroyChild (label2);
185    Window1.DestroyChild (label3);
186 }
187
188 //Reset board status
```

```
189
190 void
191 cboard_x::Reset(void)
192 {
193     pic_reset (1);
194
195     p_BT1 = 1; //set push button in default state (high)
196
197     //write button state to pic pin 19 (RD0)
198     pic_set_pin (19, p_BT1);
199     //write switch state to pic pin 20 (RD1)
200     pic_set_pin (20, p_BT2);
201
202
203     //verify serial port state and refresh status bar
204 #ifndef _WIN_
205     if (pic.serial[0].serialfd > 0)
206     else
207     if (pic.serial[0].serialfd != INVALID_HANDLE_VALUE)
208     #endif
209     Window1.statusbar1.SetField (2, lxt ("Serial: ") +
210                                 String::FromAscii (SERIALDEVICE) + lxt (":") + itoa (pic.serial[0].se
211                                 String ().Format ("%4.1f", fabs ((100.0 * pic.serial[0].serialexbaud
212                                         pic.serial[0].serialbaud) / pic.ser
213     else
214     Window1.statusbar1.SetField (2, lxt ("Serial: ") +
215                                 String::FromAscii (SERIALDEVICE) + lxt (" (ERROR)"));
216
217     if (use_spare)Window5.Reset ();
218 }
219
220 //Called ever 1s to refresh status
221
222 void
223 cboard_x::RefreshStatus(void)
224 {
225     //verify serial port state and refresh status bar
226 #ifndef _WIN_
227     if (pic.serial[0].serialfd > 0)
228     else
229     if (pic.serial[0].serialfd != INVALID_HANDLE_VALUE)
230     #endif
231     Window1.statusbar1.SetField (2, lxt ("Serial: ") +
232                                 String::FromAscii (SERIALDEVICE) + lxt (":") + itoa (pic.serial[0].se
233                                 String ().Format ("%4.1f", fabs ((100.0 * pic.serial[0].serialexbaud
234                                         pic.serial[0].serialbaud) / pic.ser
235     else
236     Window1.statusbar1.SetField (2, lxt ("Serial: ") +
```

```
237                     String::FromAscii (SERIALDEVICE) + lxt (" (ERROR)"));
238
239     }
240
241 //Called to save board preferences in configuration file
242
243 void
244 cboard_x::WritePreferences(void)
245 {
246     //write selected microcontroller of board_x to preferences
247     Window1.saveprefs (lxt ("X_proc"), Proc);
248     //write switch state of board_x to preferences
249     Window1.saveprefs (lxt ("X_bt2"), String ().Format ("%i", p_BT2));
250     //write microcontroller clock to preferences
251     Window1.saveprefs (lxt ("X_clock"), String ().Format ("%2.1f", Window1.GetClock()));
252 }
253
254 //Called whe configuration file load preferences
255
256 void
257 cboard_x::ReadPreferences(char *name, char *value)
258 {
259     //read switch state of board_x of preferences
260     if (!strcmp (name, "X_bt2"))
261     {
262         if (value[0] == '0'
263             p_BT2 = 0;
264         else
265             p_BT2 = 1;
266     }
267     //read microcontroller of preferences
268     if (!strcmp (name, "X_proc"))
269     {
270         Proc = value;
271     }
272     //read microcontroller clock
273     if (!strcmp (name, "X_clock"))
274     {
275         Window1.SetClock (atof(value));
276     }
277 }
278
279 //Event on the board
280
281 void
282 cboard_x::EvKeyPress(uint key, uint mask)
283 {
```

```
285 //if keyboard key 1 is pressed then activate button (state=0)
286 if (key == '1')
287 {
288     p_BT1 = 0;
289 }
290
291 //if keyboard key 2 is pressed then toggle switch state
292 if (key == '2')
293 {
294     p_BT2 ^= 1;
295 }
296
297 }
298
299 //Event on the board
300
301 void
302 cboard_x::EvKeyRelease(uint key, uint mask)
303 {
304     //if keyboard key 1 is pressed then deactivate button (state=1)
305     if (key == '1')
306     {
307         p_BT1 = 1;
308     }
309
310 }
311
312 //Event on the board
313
314 void
315 cboard_x::EvMouseButtonPress(uint button, uint x, uint y, uint state)
316 {
317
318     int i;
319
320     //search for the input area which owner the event
321     for (i = 0; i < inputc; i++)
322     {
323         if (((input[i].x1 <= x)&&(input[i].x2 >= x))&&((input[i].y1 <= y)&&
324                                         (input[i].y2 >= y)))
325         {
326             switch (input[i].id)
327             {
328                 //if event is over I_ISCP area then load hex file
329                 case I_ICSP:
330                     Window1.menu1_File_LoadHex_EvMenuActive (NULL);
331                     break;
```

```

333         //if event is over I_PWR area then toggle board on/off
334     case I_PWR:
335         if (Window1.Get_mcupwr ()) //if on turn off
336         {
337             Window1.Set_mcurun (0);
338             Window1.Set_mcupwr (0);
339             Reset ();
340             p_BT1 = 1;
341             Window1.statusbar1.SetField (0, lxt ("Stoped"));
342         }
343         else //if off turn on
344         {
345             Window1.Set_mcupwr (1);
346             Window1.Set_mcurun (1);
347             Reset ();
348             Window1.statusbar1.SetField (0, lxt ("Running..."));
349         }
350         break;
351         //if event is over I_RST area then turn off and reset
352     case I_RST:
353         if (Window1.Get_mcupwr () && pic_reset (-1))//if powered
354         {
355             Window1.Set_mcupwr (0);
356             Window1.Set_mcurst (1);
357         }
358         p_MCLR = 0;
359         break;
360         //if event is over I_D0 area then activate button (state=0)
361     case I_D0:
362         p_BT1 = 0;
363         break;
364         //if event is over I_D1 area then toggle switch state
365     case I_D1:
366         p_BT2 ^= 1;
367         break;
368     }
369   }
370 }
371 }
372 }
373
374 //Event on the board
375
376 void
377 cboard_x::EvMouseButtonRelease(uint button, uint x, uint y, uint state)
378 {
379   int i;
380

```

```

381 //search for the input area which owner the event
382 for (i = 0; i < inputc; i++)
383 {
384     if (((input[i].x1 <= x)&&(input[i].x2 >= x))&&((input[i].y1 <= y)&&
385                                         (input[i].y2 >= y)))
386     {
387         switch (input[i].id)
388         {
389             //if event is over I_RST area then turn on
390             case I_RST:
391                 if (Window1.Get_mcurst ())//if powered
392                 {
393                     Window1.Set_mcupwr (1);
394                     Window1.Set_mcurst (0);
395
396                 if (pic_reset (-1))
397                 {
398                     Reset ();
399                 }
400             }
401             p_MCLR = 1;
402             break;
403             //if event is over I_D0 area then deactivate button (state=1)
404             case I_D0:
405                 p_BT1 = 1;
406                 break;
407             }
408         }
409     }
410 }
411 }
412
413
414 //Called ever 100ms to draw board
415 //This is the critical code for simulator running speed
416
417 void
418 cboard_x::Draw(CDraw *draw, double scale)
419 {
420     int i;
421
422     draw->Canvas.Init (scale, scale); //initialize draw context
423
424     //board_x draw
425     for (i = 0; i < outputc; i++) //run over all outputs
426     {
427         if (!output[i].r)//if output shape is a rectangle
428         {

```

```

429     if (output[i].id == O_SD1) //if output is switch
430     {
431         //draw a background white rectangle
432         draw->Canvas.SetBgColor (255, 255, 255);
433         draw->Canvas.Rectangle (1, output[i].x1, output[i].y1,
434                                 output[i].x2 - output[i].x1, output[i].y2 - output[i].y1);
435
436         if (!p_BT2) //draw switch off
437         {
438             //draw a grey rectangle
439             draw->Canvas.SetBgColor (70, 70, 70);
440             draw->Canvas.Rectangle (1, output[i].x1, output[i].y1 +
441                                     (int) ((output[i].y2 - output[i].y1)*0.35)), output[i].x2 - output[i].x1,
442                                     (int) ((output[i].y2 - output[i].y1)*0.65));
443         }
444         else //draw switch on
445         {
446             //draw a grey rectangle
447             draw->Canvas.SetBgColor (70, 70, 70);
448             draw->Canvas.Rectangle (1, output[i].x1,
449                                     output[i].y1, output[i].x2 - output[i].x1,
450                                     (int) ((output[i].y2 - output[i].y1)*0.65));
451         }
452     }
453 }
454 else //if output shape is a circle
455 {
456
457     draw->Canvas.SetFgColor (0, 0, 0); //black
458
459     switch (output[i].id) //search for color of output
460     {
461         case O_LD0: //White using pin 19 mean value (RDO)
462             draw->CanvasSetColor (pic.pins[18].oavalue, pic.pins[18].oavalue, pic.pins[18].oavalue);
463             break;
464         case O_LD1: //Yellow using pin 20 mean value (RD1)
465             draw->CanvasSetColor (pic.pins[19].oavalue, pic.pins[19].oavalue, 0);
466             break;
467         case O_LPWR: //Blue using mcupwr value
468             draw->CanvasSetColor (0, 0, 225 * Window1.Get_mcupwr () + 30);
469             break;
470         case O_RB0: //Green using pin 33 mean value (RB0)
471             draw->CanvasSetColor (0, pic.pins[32].oavalue, 0);
472             break;
473         case O_RB1: //Red using pin 34 mean value (RB1)
474             draw->CanvasSetColor (pic.pins[33].oavalue, 0, 0);
475             break;
476     }

```

```
477
478     //draw a circle
479     draw->Canvas.Circle (1, output[i].x1, output[i].y1, output[i].r);
480 }
481 }
482 }
483
484 //end draw
485 draw->Canvas.End ();
486 draw->Update ();
487
488 //RB0 mean value to gauge1
489 gauge1->SetValue (0.4444 * (pic.pins[33].oavalue - 30));
490 //RB1 mean value to gauge2
491 gauge2->SetValue (0.44444 * (pic.pins[32].oavalue - 30));
492 }
493 }
494
495 void
496 cboard_x::Run_CPU(void)
497 {
498     int i;
499     int j;
500     unsigned char pi;
501     const picpin * pins;
502     unsigned int alm[40];
503
504     int JUMPSTEPS = Window1.GetJUMPSTEPS (); //number of steps skipped
505     long int NSTEPJ = Window1.GetNSTEPJ (); //number of steps in 100ms
506
507
508     //reset pins mean value
509     memset (alm, 0, 40 * sizeof (unsigned int));
510
511     //read pic.pins to a local variable to speed up
512     pins = pic.pins;
513
514     //Spare parts window pre process
515     if (use_spare)Window5.PreProcess ();
516
517     j = JUMPSTEPS; //step counter
518     if (Window1.Get_mcupwr ()) //if powered
519         for (i = 0; i < Window1.GetNSTEP (); i++) //repeat for number of steps in 100ms
520     {
521
522         if (j >= JUMPSTEPS) //if number of step is bigger than steps to skip
523         {
524             pic_set_pin (pic.mclr, p_MCLR);
```

```

525     pic_set_pin (19, p_BT1); //Set pin 19 (RD0) with button state
526     pic_set_pin (20, p_BT2); //Set pin 20 (RD1) with switch state
527 }
528
529 //verify if a breakpoint is reached if not run one instruction
530 if (!mplabxd_testbp ())pic_step ();
531 //Oscilloscope window process
532 if (use_scope)Window4.SetSample ();
533 //Spare parts window process
534 if (use_spare)Window5.Process ();
535
536 //increment mean value counter if pin is high
537 if (j < pic.PINCOUNT)
538     alm[j] += pins[j].value;
539
540 if (j >= JUMPSTEPS)//if number of step is bigger than steps to skip
541 {
542
543     //set analog pin 2 (AN0) with value from scroll
544     pic_set_apin (2, ((5.0 * (scroll1->GetPosition ()) /
545                         (scroll1->GetRange () - 1)));
546
547     j = -1; //reset counter
548 }
549 j++; //counter increment
550 }
551
552 //calculate mean value
553 for (pi = 0; pi < pic.PINCOUNT; pi++)
554 {
555     pic.pins[pi].oavalue = (int) (((225.0 * alm[pi]) / NSTEPJ) + 30);
556 }
557
558 //Spare parts window pre post process
559 if (use_spare)Window5.PostProcess ();
560 }
561
562
563 //Register the board in PICsimLab
564 board_init("X", cboard_x);

```

13.2.5 Integration with PICsimLab

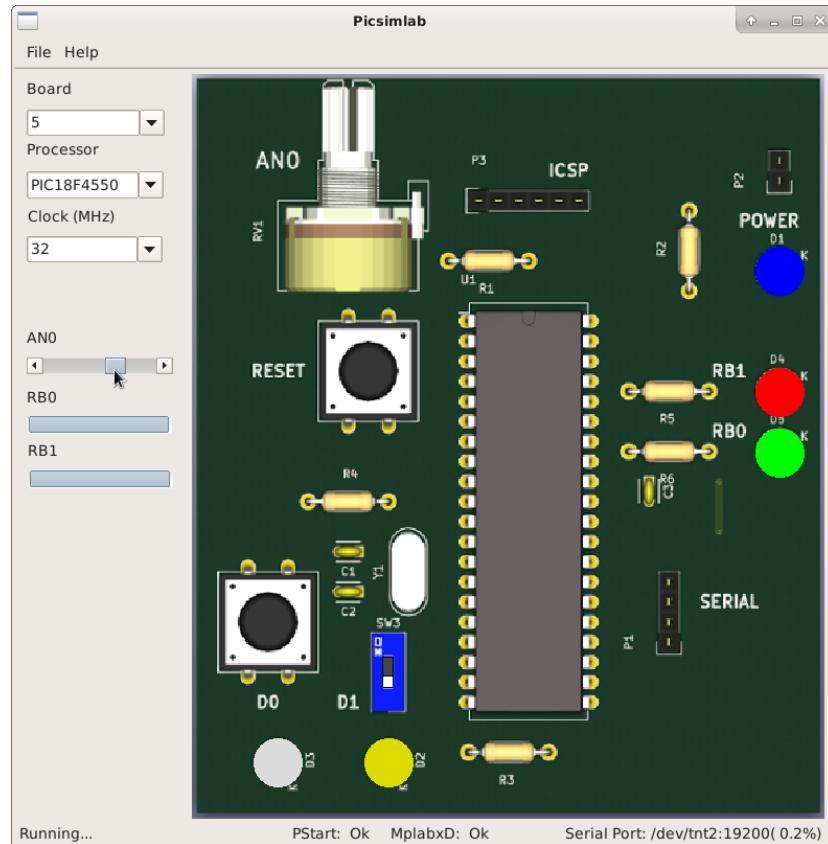
To integration of the new board in PICSimLab, are necessary edit one file.

The file is Makefile.Common. The only change to be made is include object **boards/board_board_x.o** in objects list. They can be added in variable OBJS or OBJS_EXP (used only in experimental version).

After change the Makefile.Common and include the five files created for new board, the PICSimLab can be recompiled, as described in first chapter.

13.2.6 Final Result

The PICSimLab board created for this tutorial are shown in the figure below.



The sample program below can be used to test new board, this code is write for XC8 compiler:

```

1 #include <xc.h>;
2
3 #include "config_4550.h"
4 #include "adc.h"
5 #include "serial.h"
6 #include "itoa.h"
7
8 void main()
9 {

```

```
10     unsigned int val;
11     char buffer[10];
12
13     ADCON1=0x02;
14     TRISA=0xFF;
15     TRISB=0xFC;
16     TRISC=0xBF;
17     TRISD=0xFF;
18     TRISE=0x0F;
19
20     adc_init();
21     serial_init();
22
23
24     while(1)
25     {
26         val=adc_amostra(0);
27
28         if(PORTDbits.RD1)
29         {
30             if(val > 340)
31                 PORTBbits.RB0=1;
32             else
33                 PORTBbits.RB0=0;
34
35             if(val > 680)
36                 PORTBbits.RB1=1;
37             else
38                 PORTBbits.RB1=0;
39         }
40         else
41         {
42             if(PORTDbits.RD0)
43             {
44                 PORTBbits.RB0=1;
45                 PORTBbits.RB1=0;
46             }
47             else
48             {
49                 PORTBbits.RB0=0;
50                 PORTBbits.RB1=1;
51             }
52         }
53     }
54
55     serial_tx_str(itoa(val,buffer));
56     serial_tx_str("\r\n");
57 }
```

58
59 }

Chapter 14

License

Copyright © 2021 Luis Claudio Gambôa Lopes <lcgamboa@yahoo.com>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.