



PICSimLab 0.8.9

Luis Claudio Gambôa Lopes <lcgamboa@yahoo.com>

Download: [github](#)

[HTML version of documentation](#)

June 20, 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	39
10 Spare Parts	40
10.1 Inputs	43
10.1.1 Encoder	43
10.1.2 Gamepad	44
10.1.3 Gamepad Analogic	45
10.1.4 Keypad	45
10.1.5 LM35	47
10.1.6 MPU6050	47
10.1.7 Potentiometers	48
10.1.8 Potentiometers (Rotary)	49
10.1.9 Push Buttons	49
10.1.10 Push Buttons (Analogic)	50
10.1.11 Switchs	50
10.1.12 Ultrasonic HC-SR04	51
10.2 Outputs	51
10.2.1 7 Segments Display	51
10.2.2 7 Segments Display (Decoder)	51
10.2.3 Buzzer	52
10.2.4 DC Motor	52
10.2.5 LCD hd44780	53

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

10.2.6 LCD ili9341	54
10.2.7 LCD pcf8833	55
10.2.8 LCD pcd8544	55
10.2.9 LCD ssd1306	56
10.2.10 LED Matrix	56
10.2.11 LEDs	57
10.2.12 RGB LED	58
10.2.13 Servo Motor	58
10.2.14 Step Motor	59
10.3 Others	60
10.3.1 ETH w5500	60
10.3.2 IO 74xx595	61
10.3.3 IO MCP23S17	62
10.3.4 IO PCF8574	62
10.3.5 IO UART	62
10.3.6 Jumper Wires	63
10.3.7 MEM 24CXXX	63
10.3.8 RTC ds1307	64
10.3.9 RTC pfc8563	64
10.3.10 SD Card	65
10.3.11 Temperature System	65
10.4 Virtual	66
10.4.1 D. Transfer Function	66
10.4.2 IO Virtual Term	66
10.4.3 Signal Generator	67
10.4.4 VCD Dump	67
10.4.5 VCD Dump (Analogic)	68
10.4.6 VCD Play	68
11 Troubleshooting	70
12 Use with MPLABX	71
12.1 Installing the Necessary Tools	71
12.1.1 Install MPLABX IDE and XC8 Compiler	71
12.1.2 Install PICsimLab	71
12.1.3 How to Install PicSimLab MPLABX Debugger plugin	71
12.2 Configuring a New Project in MPLABX	76
12.2.1 Project Creation	76
12.2.2 File Creation	79
12.2.3 PIC Configuration Bits	80
12.2.4 Code Example	81
12.2.5 Building the Project	82
12.3 Program and Debug PICsimLab With MPLABX	83
12.3.1 Starting PICsimLab	83
12.3.2 Programming PICsimLab	83
12.3.3 Pausing the Program	84

CONTENTS	4
12.3.4 Restarting the Program	85
12.3.5 Running Step by Step	85
12.3.6 Stopping Debugger	86
12.4 This Tutorial in Video	87
13 Creating New Boards	88
13.1 How to Compile PICsimLab	88
13.1.1 In Debian Linux and derivatives	88
13.1.2 Cross-compiling for windows	88
13.2 Creating a New Board	88
13.2.1 Board Hardware and Schematic	89
13.2.2 Board Picture	92
13.2.3 Picture maps	94
13.2.4 Board code	98
13.2.5 Integration with PICsimLab	117
13.2.6 Final Result	117
14 License	120

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
```

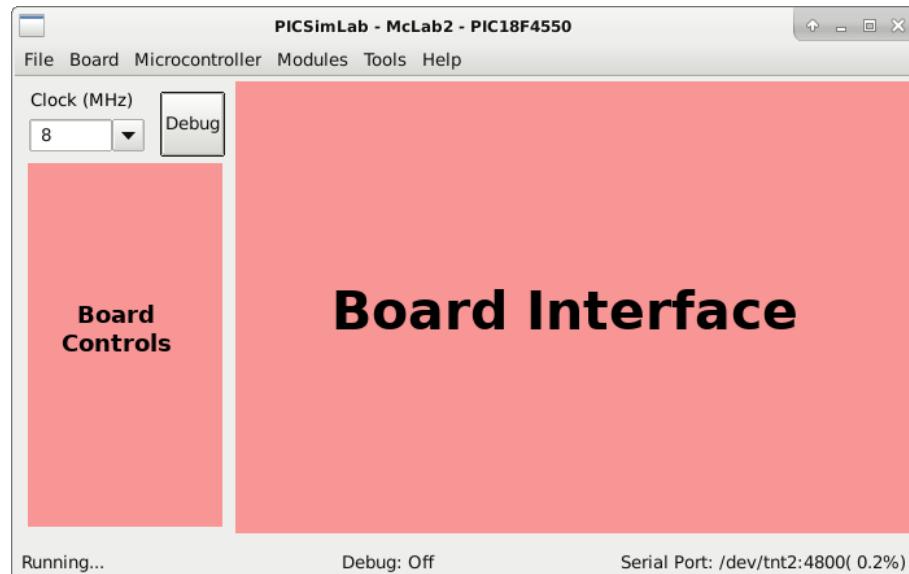
Chapter 3

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

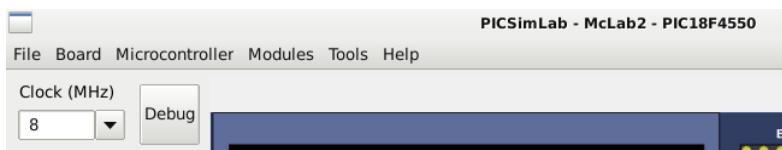
the simulation may present some difference than expected and the CPU load will be increased.

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 remote control interface allows other programs to control the PICSimLab simulation through a TCP/IP socket using text formatted commands.

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

```

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:

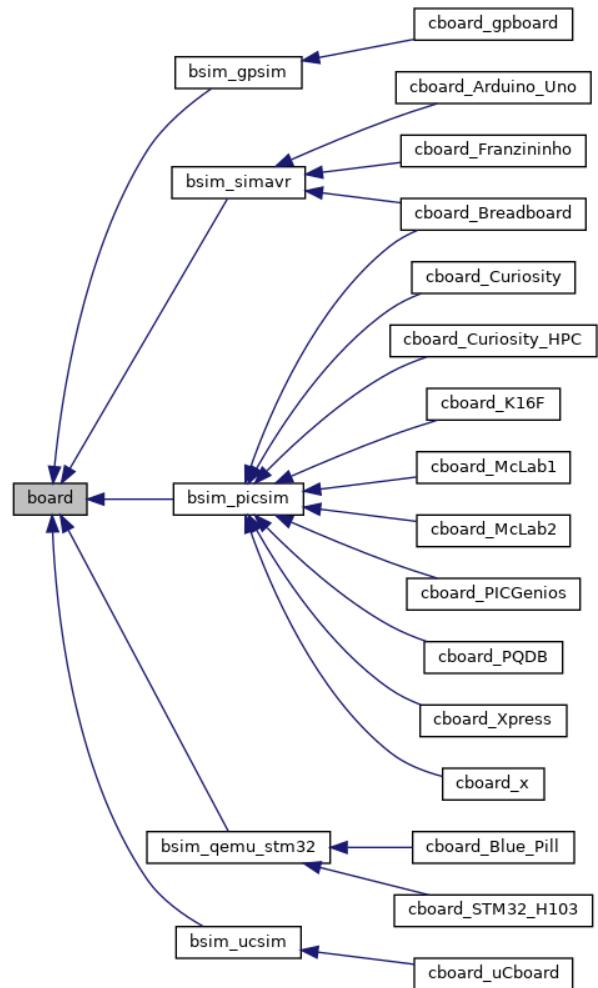


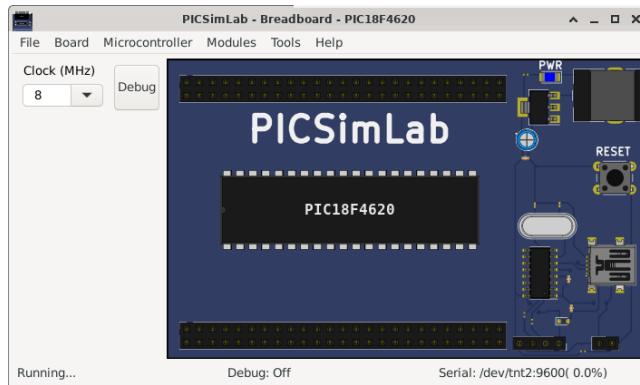
Figure 4.1: Boards backend simulators

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

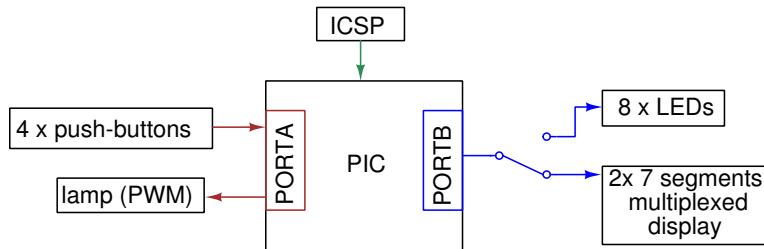
It is a generic board only with reset, serial and crystal circuits and support to multiple microcontrollers of [picsim](#) and [simavr](#).

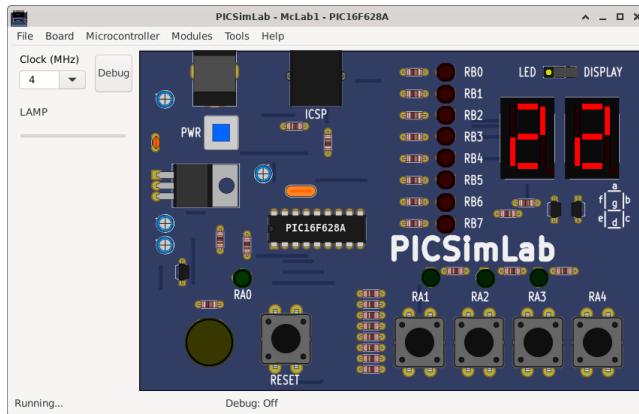


[Examples](#)

4.2 McLab1

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





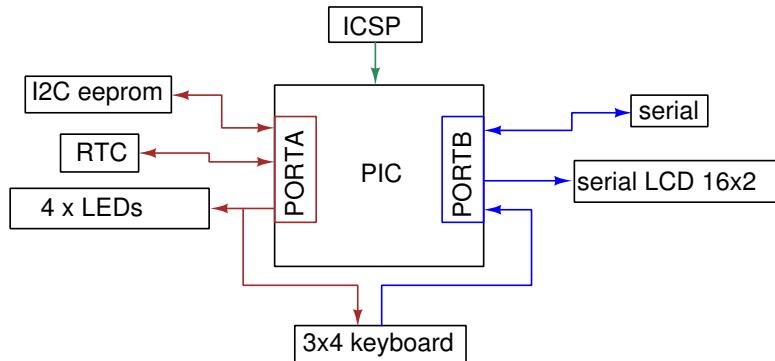
Board McLab1 schematics.

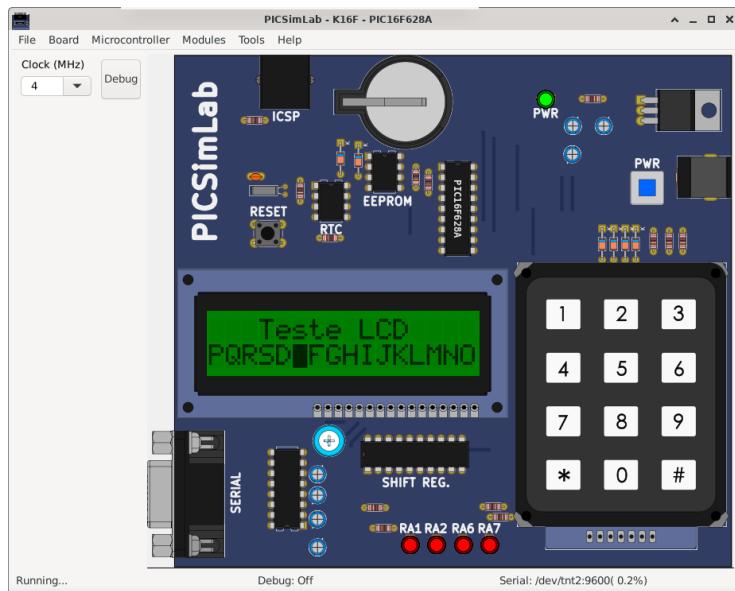
The code examples can be loaded in PICSimLab menu **Help->Examples**.

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](#).





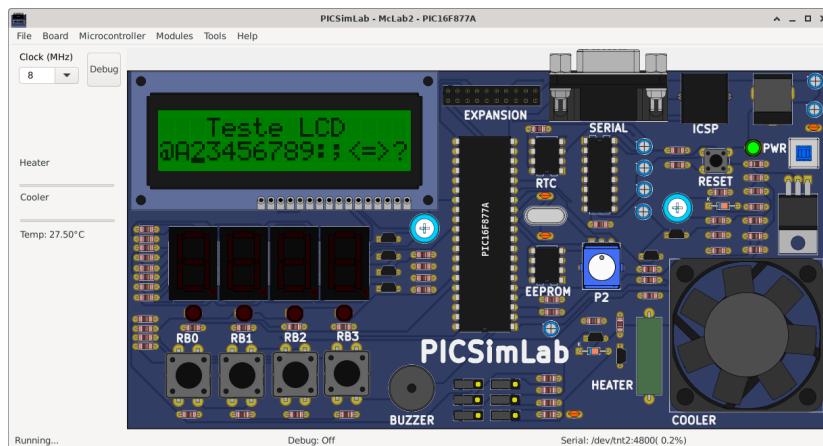
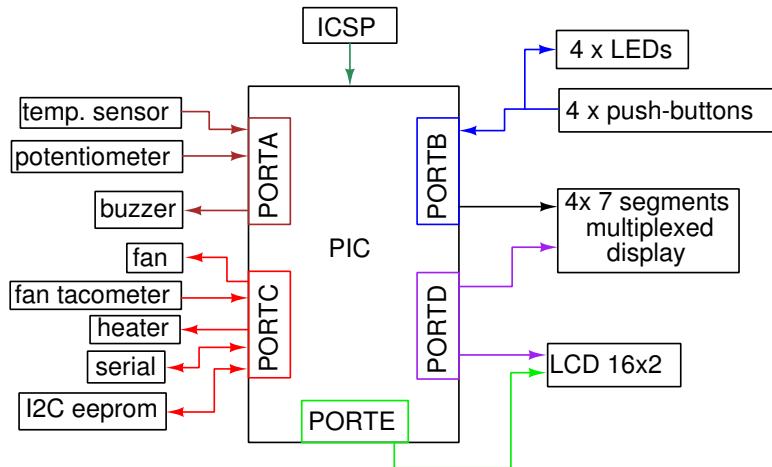
Board K16F schematics.

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](#).



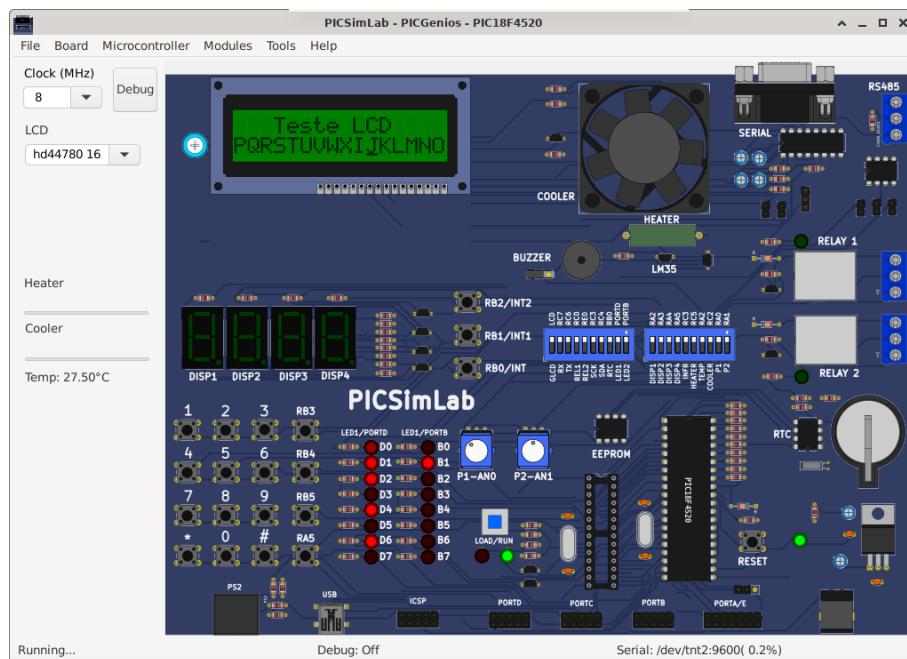
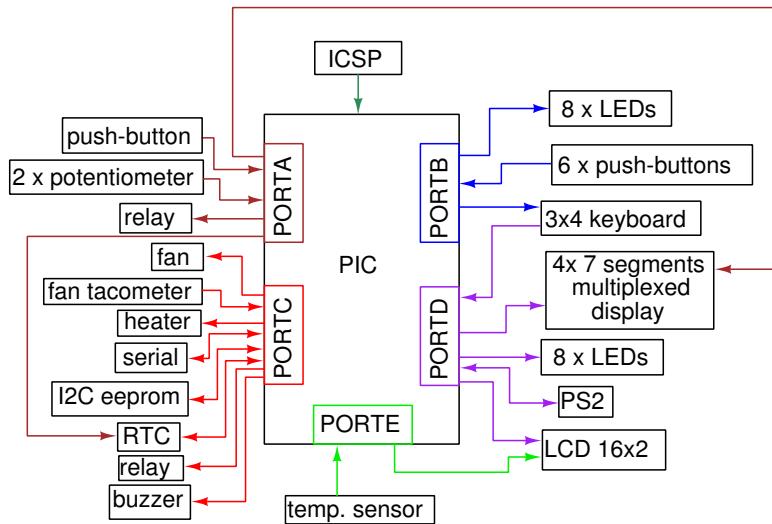
Board McLab2 schematics.

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](#).



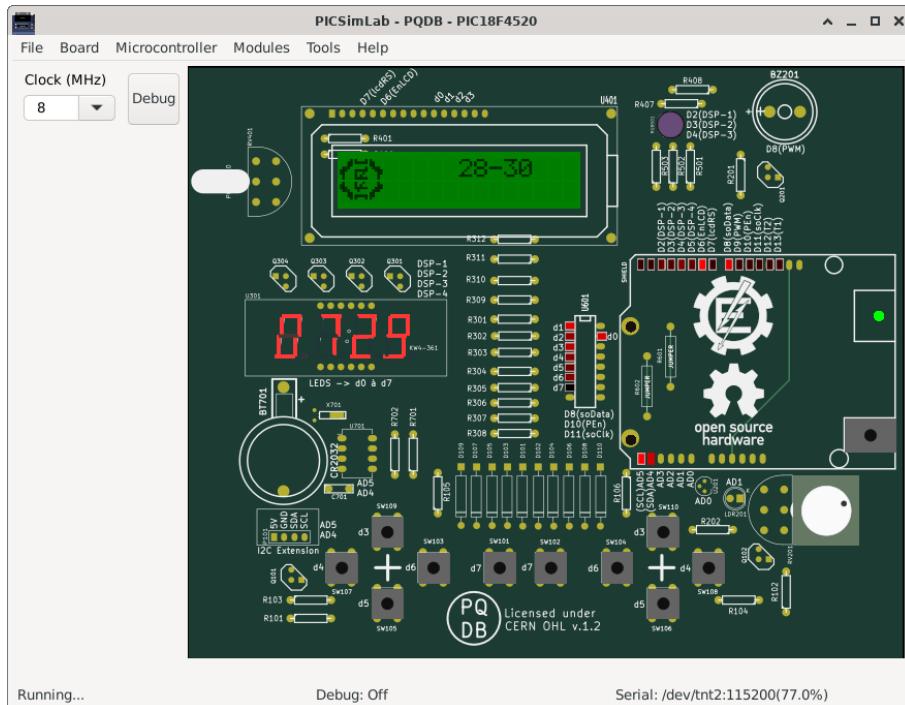
Board PICGenios schematics.

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

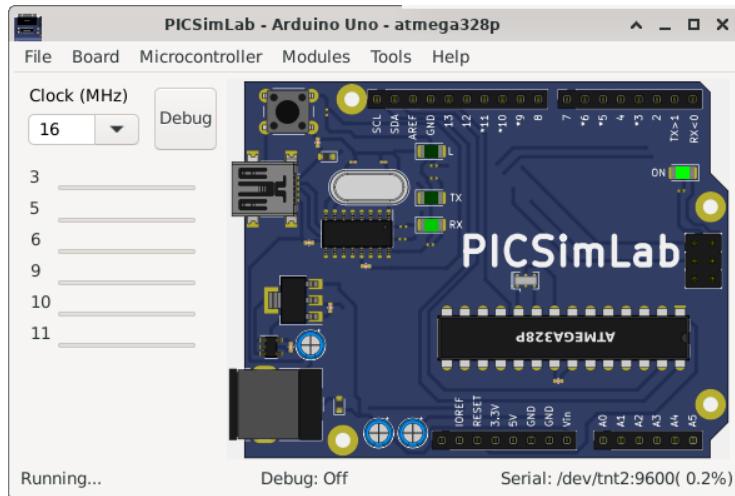
The PQDB board is an opensource/openhardware project, more info at <https://github.com/projetopqdb/>. It was developed to be used with arduino/freedom boards, but adapted to use the microcontroller PIC18F4520 of [picsim](#) on PICSImLab.



[Examples](#)

4.7 Arduino Uno

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



Board Arduino Uno schematics.

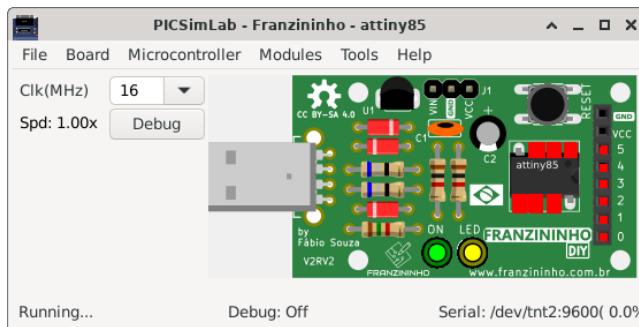
The code examples can be loaded in PICSimLab menu **Help->Examples**.

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

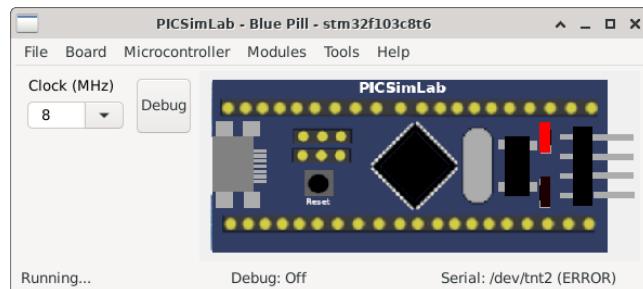
Chapter 5

Experimental Boards

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

5.1 Blue Pill

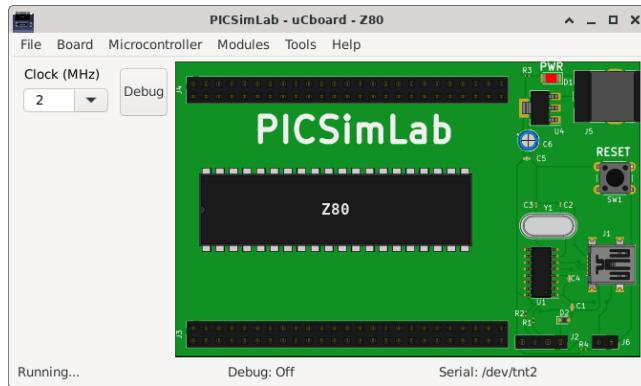
It is a generic board only with reset, serial and crystal circuits and support to stm32f103c8t6 microcontroller of [qemu-stm32](#).



[Examples](#)

5.2 uCboard

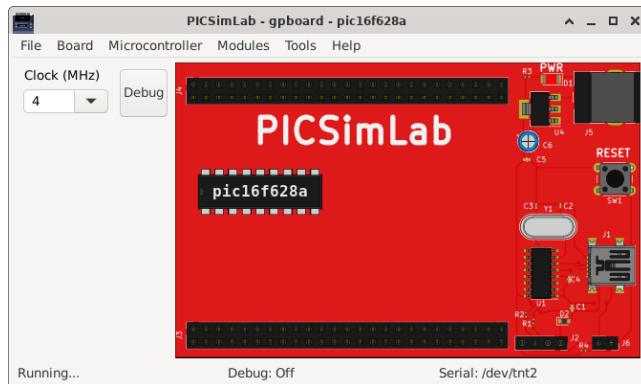
It is a generic board only with reset, serial and crystal circuits and support to multiple microcontrollers (initially C51, Z80 and STM8S103)of [uCsim](#).



[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

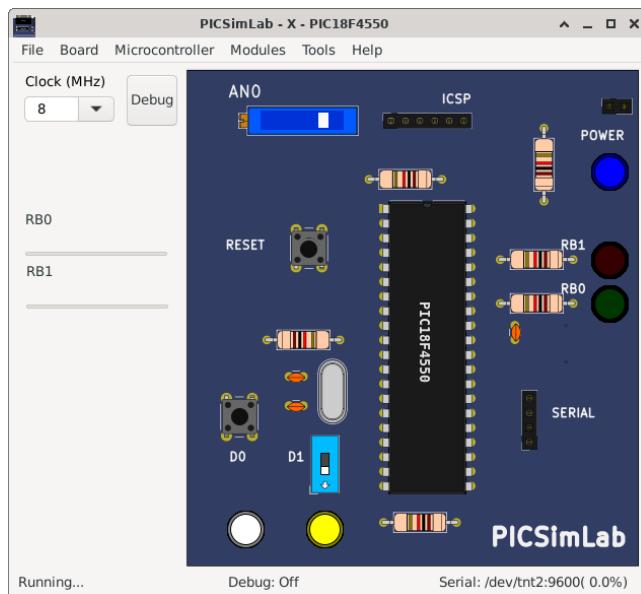
It is a generic board only with reset, one push button, serial and crystal circuits and support to stm32f103rbt6 microcontroller of [qemu-stm32](#).



Examples

5.5 X

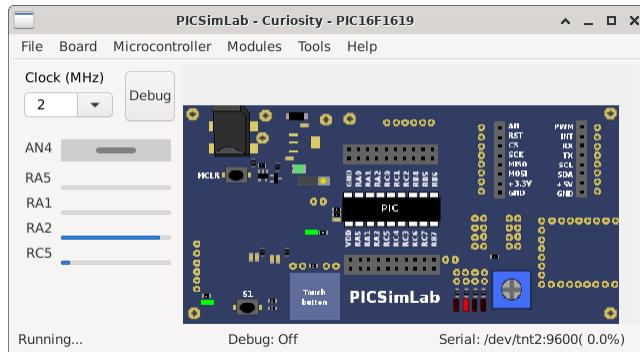
It is a generic board, used as example in [How to Compile PICsimLab and Create New Boards](#).



Examples

5.6 Curiosity

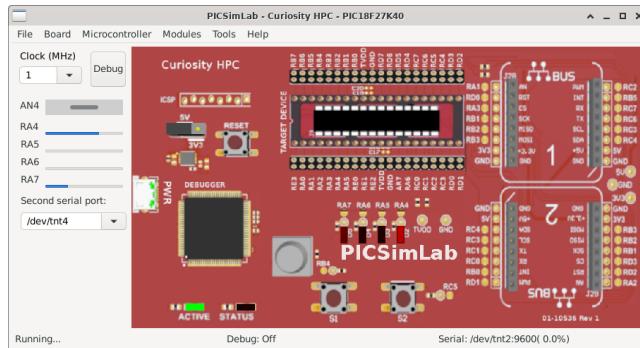
This is a simple PIC microcontroller development board that uses [picsim](#).



[Examples](#)

5.7 Curiosity HPC

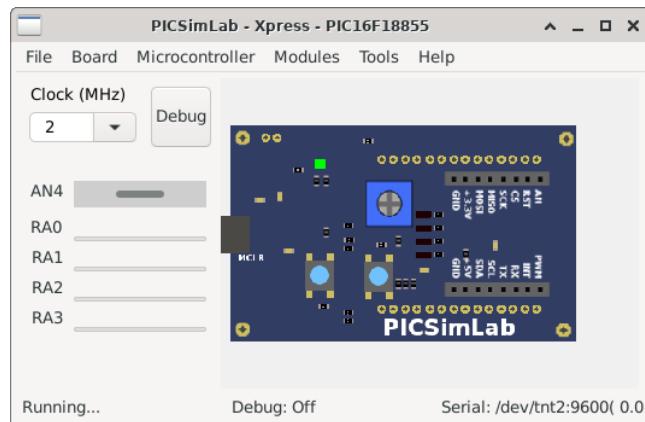
This is a simple PIC microcontroller development board that uses [picsim](#).



[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:

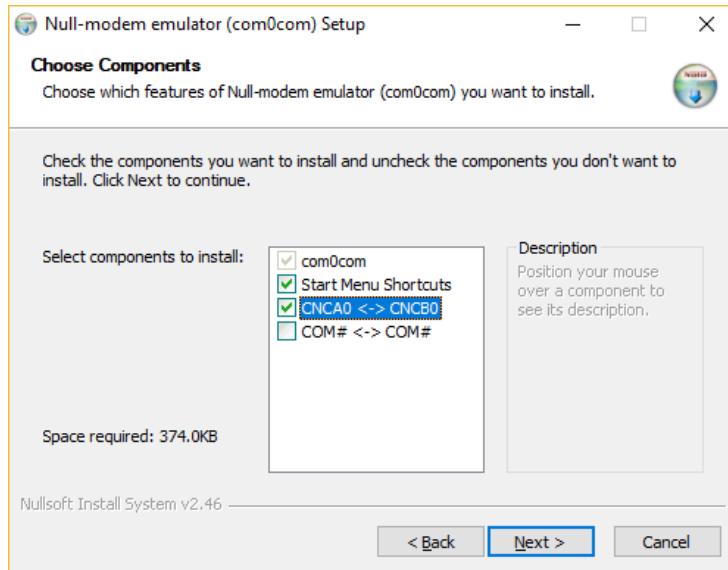
OS	PicsimLab port	CuteCom port	NULL-Modem prog.	Connection
Windows	com1	com2	com0com	com1<=>com2
Linux	/dev/tnt2	/dev/tnt3	tty0tty	/dev/tnt2<=>/dev/tnt3

6.1 Com0com Installation and Configuration(Windows)

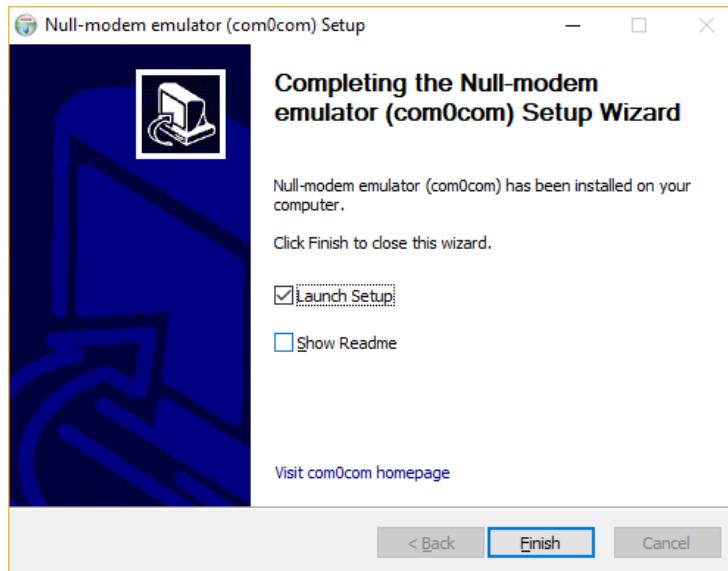
Download the signed version of [com0com](#).

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.

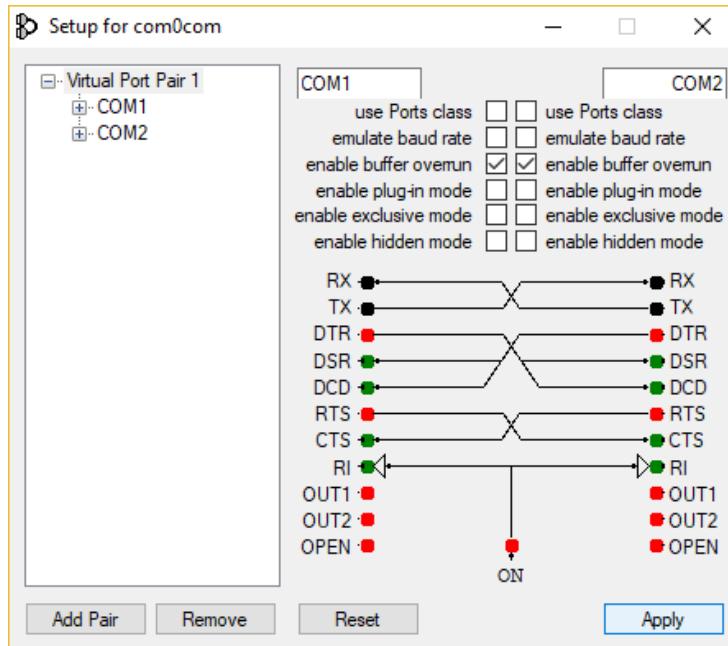
Configure the “choose components” window as the figure below:



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

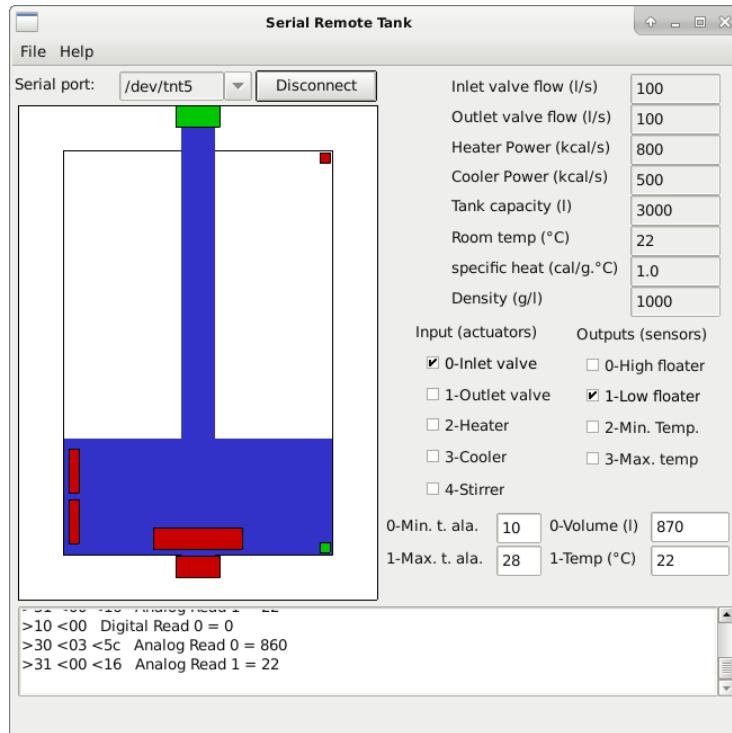
To use this tool with PICSimLab you first need to configure a virtual serial port as described in Chapter: [Serial Communication](#). It is possible to use this tool with a real serial port connected to a real device.

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.

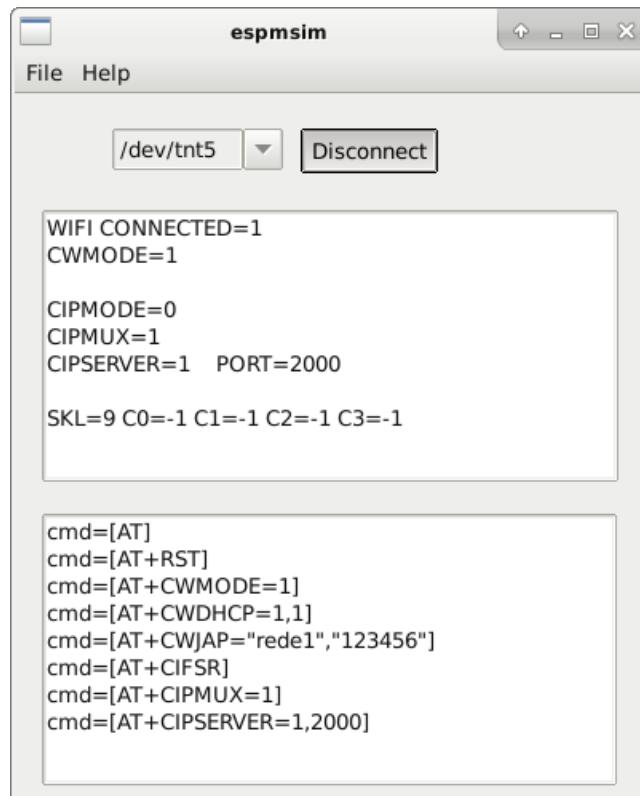
Example to read analog output 2:

```
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

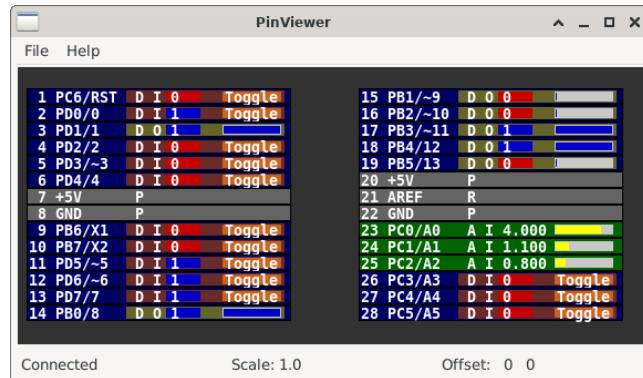
8.5 MPLABX Debugger Plugin

Open the web page to download the plugin

8.6 Pin Viewer

Open the Pin status viewer program.

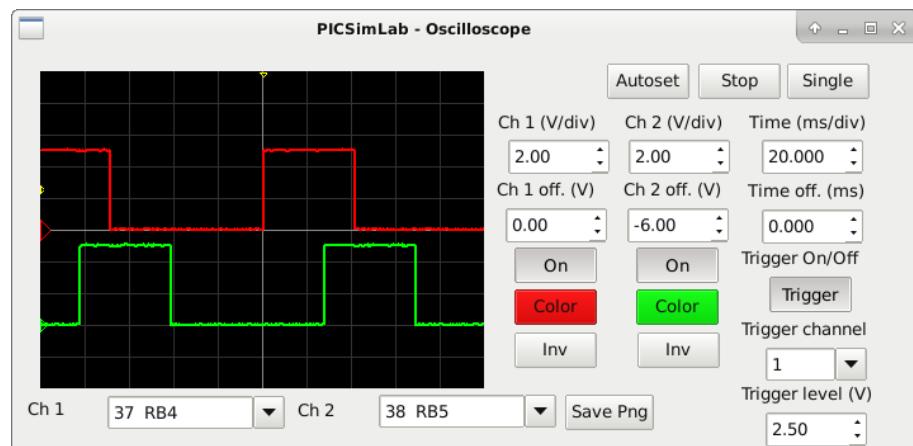
PinViewer connects to PICSimLab through the [rcontrol interface](#) and allows viewing the status and direction of all microcontroller pins. It is also possible to change the state of the digital pins and adjust the voltage value on the analog pins configured as input. Pins configured as outputs also show the average value, useful for evaluating the functioning of PWM outputs.



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

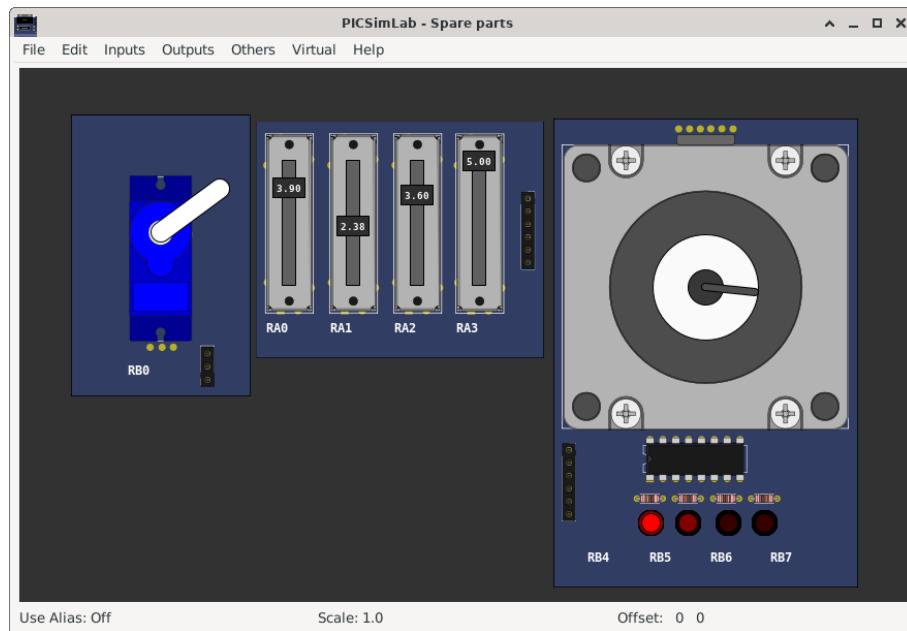
The PICSimLab has a window that allows the connection of spare parts to the microcontroller, it can be accessed through the menu “ Modules-> Spare parts ”.

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

- LM35 - Adds a analog temperature sensor
 - 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

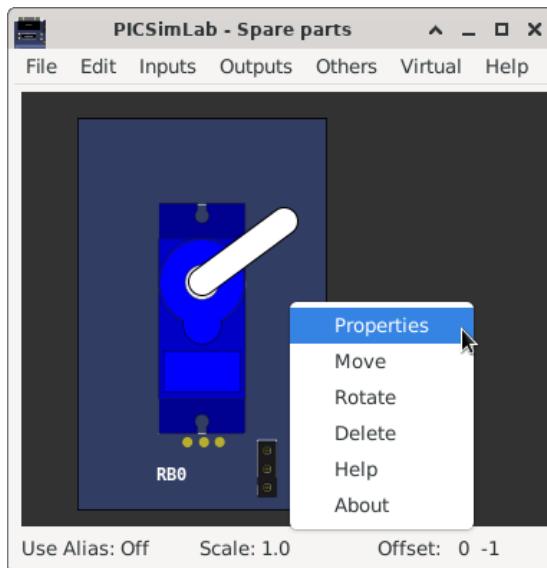
- SD Card - Adds a SD card shield
- Temperature System - Adds a temperature control system
- Virtual
 - D. Transfer Function - Adds a discrete transfer function mathematical model
 - IO Virtual term - Adds a virtual serial terminal
 - Signal Generator - Adds a virtual signal generator
 - VCD Dump - Adds 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

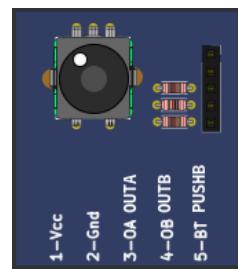
- Delete - Remove part
- Help - Open Help window of part
- About - Show message about author and version of part



10.1 Inputs

10.1.1 Encoder

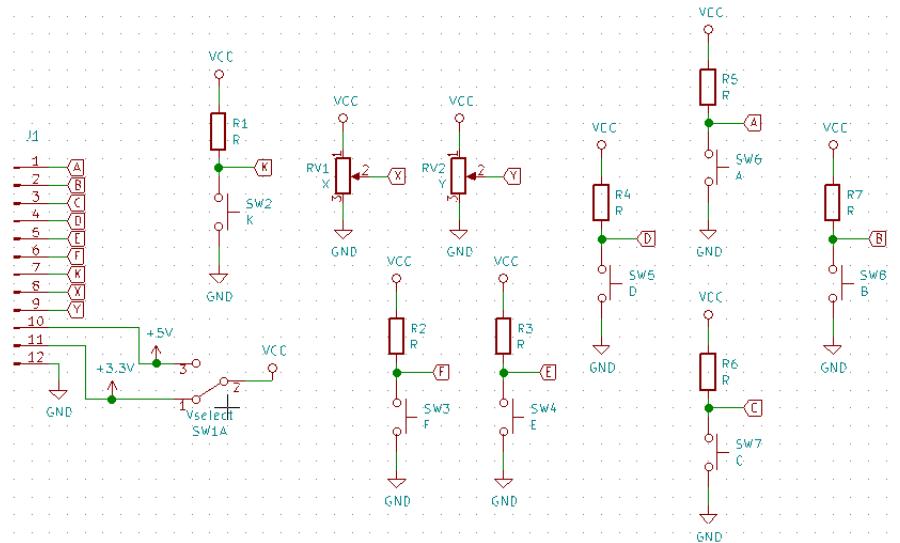
This part is a rotary quadrature encoder with push button. The output is twenty pulses per revolution.



[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:

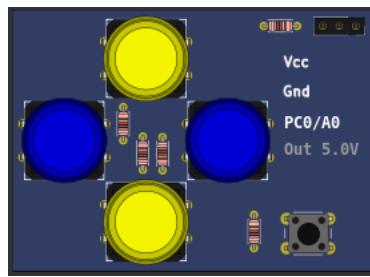
- X axis - keys 'A' and 'D'
- Y axis - keys 'W' and 'S'
- Button A - key 'I'
- Button B - key 'L'
- Button C - key 'K'

- 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.



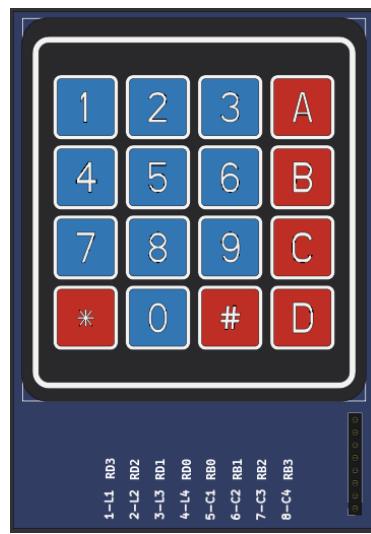
The gamepad can be controlled by keyboards keys:

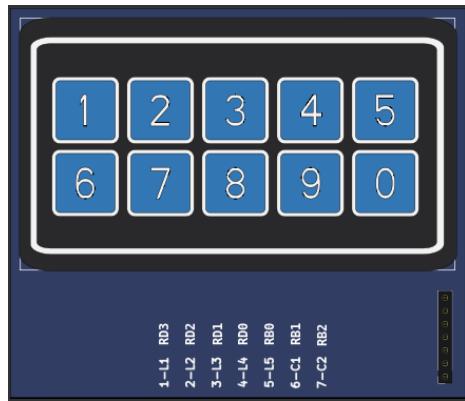
- Button A - key 'L'
- Button B - key 'I'
- Button C - key 'K'
- Button D - key 'J'
- Button E - key 'O'

[Examples](#)

10.1.4 Keypad

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

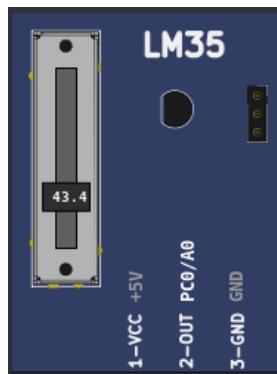




[Examples](#)

10.1.5 LM35

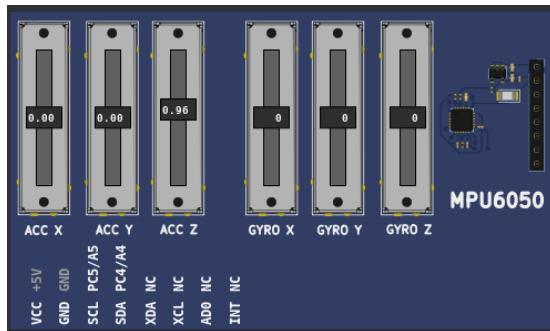
This part is LM35 analog temperature sensor. The voltage output is 10mV/C.



[Examples](#)

10.1.6 MPU6050

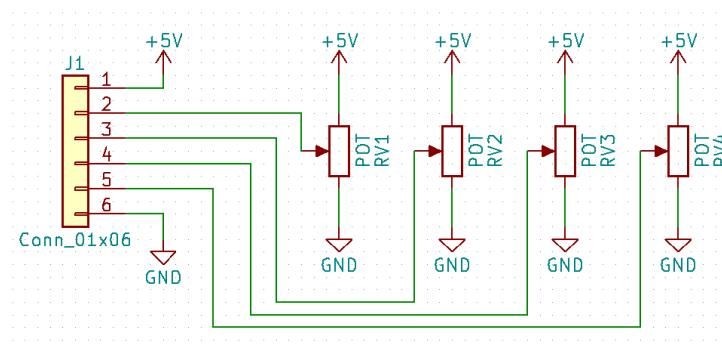
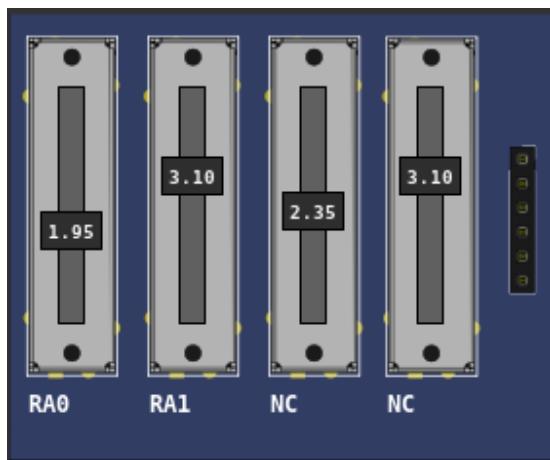
This part is MPU6050 accelerometer and gyroscope with I2C interface. Only raw values are available, DMP is not supported.



Examples

10.1.7 Potentiometers

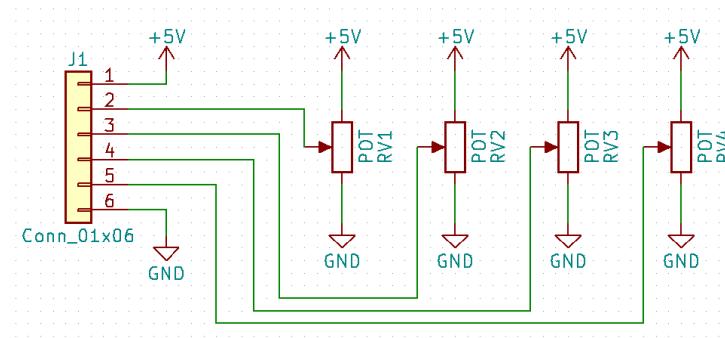
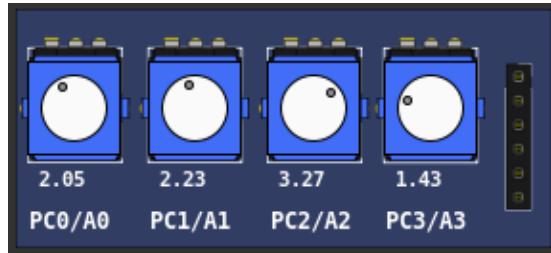
This part is formed by 4 potentiometers connected between 0 and 5 volts, the output is connected to the cursor and varies within this voltage range.



Examples

10.1.8 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.

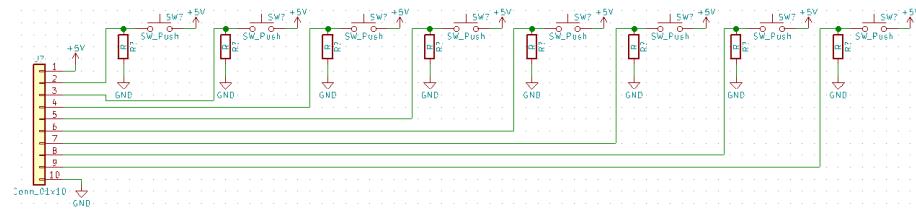


Examples

10.1.9 Push Buttons

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





Examples

10.1.10 Push Buttons (Analogic)

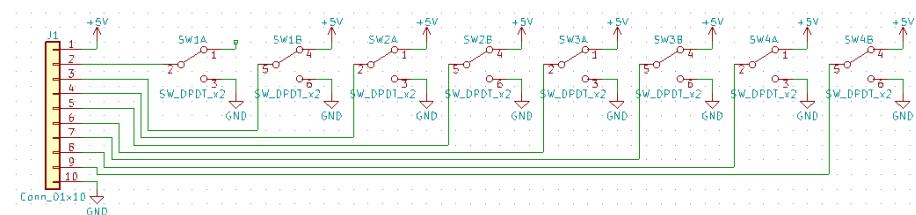
This part consists of 8 push buttons connected in a resistive ladder.



Examples

10.1.11 Switches

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



Examples

10.1.12 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

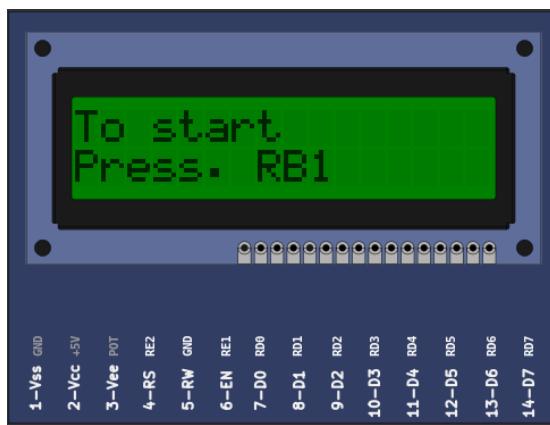
This part is DC motor with H-bridge driver and quadrature encoder.



Examples

10.2.5 LCD hd44780

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

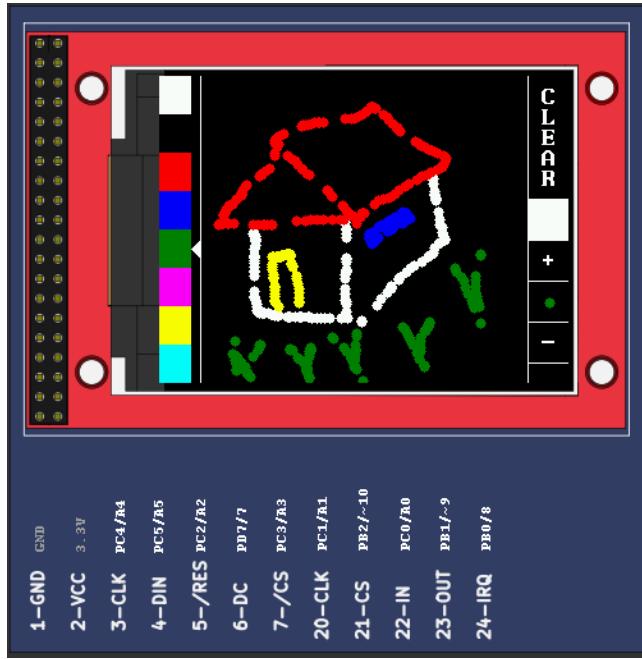




Examples

10.2.6 LCD ili9341

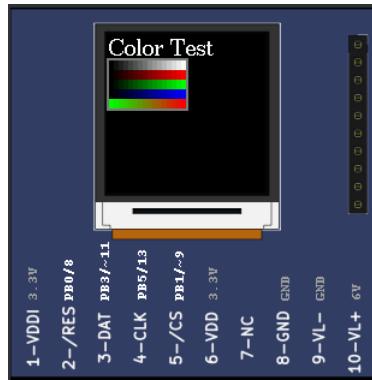
This part is a color graphic display with 240x320 pixels with touchscreen (xpt2046 controller). Only 4 SPI mode and 8 bits parallel mode is available.



[Examples](#)

10.2.7 LCD pcf8833

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



[Examples](#)

10.2.8 LCD pcd8544

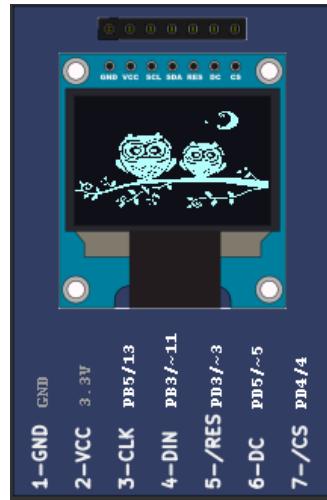
This part is a monochrome graphic display with 48x84 pixels. (Nokia 5110)



[Examples](#)

10.2.9 LCD ssd1306

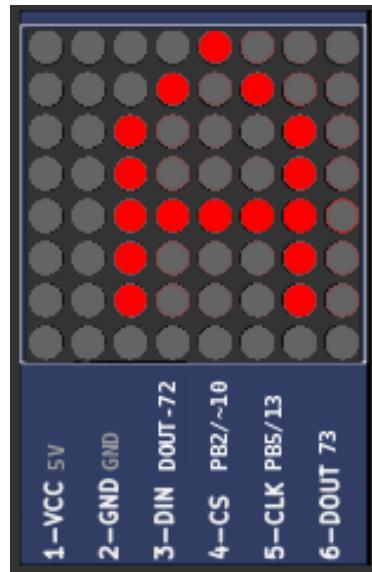
This part is a monochrome oled graphic display with 128x64 pixels. The part support I2C and 4 SPI serial mode.



[Examples](#)

10.2.10 LED Matrix

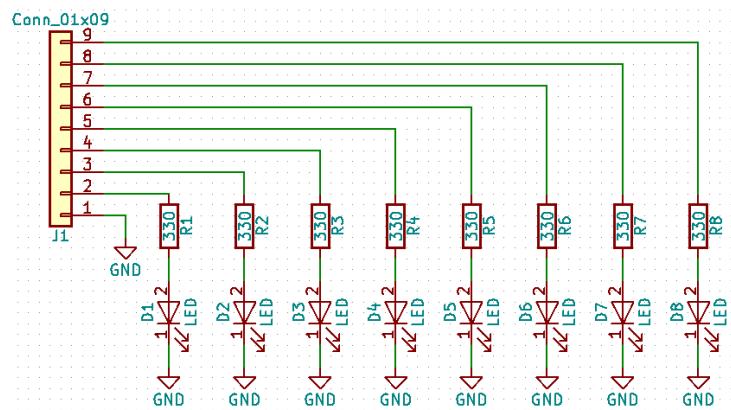
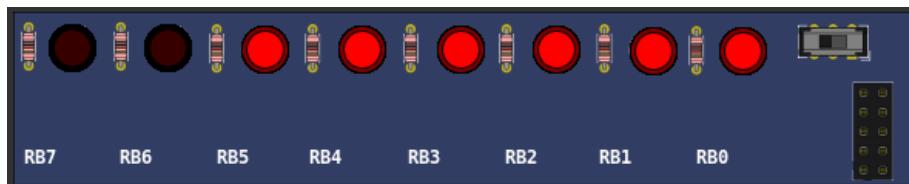
It is a 8x8 LED matrix with MAX72xx controller.



Examples

10.2.11 LEDs

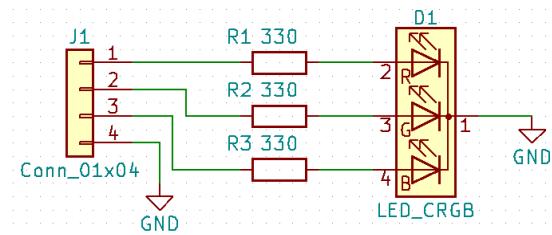
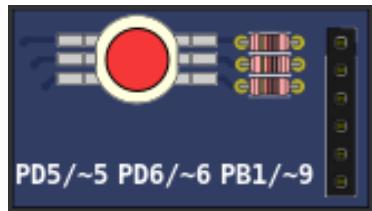
This part is a bar of 8 independent red LEDs.



Examples

10.2.12 RGB LED

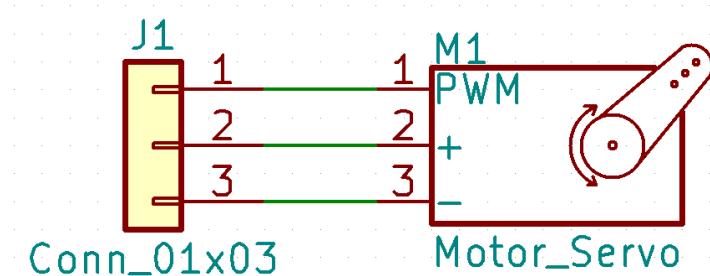
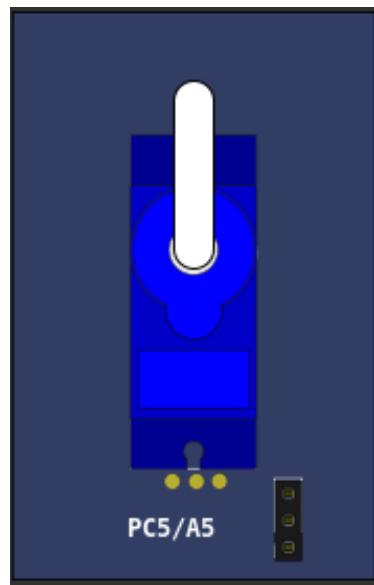
This part consists of a 4-pin RGB LED. Each color can be triggered independently. Using PWM it is possible to generate several colors by combining the 3 primary colors.



Examples

10.2.13 Servo Motor

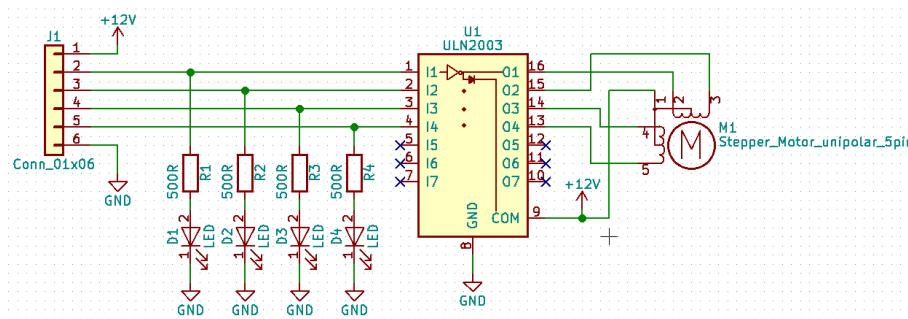
The servo motor is a component that must be activated with a pulse of variable width from 1ms to 2ms every 20 ms. A pulse of 1ms positions the servo at -90°, one from 1.5ms to 0° and one from 2ms to 90°.



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)	

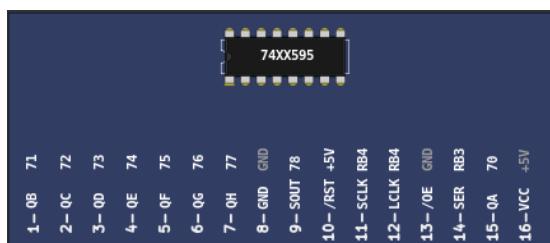
Click on connector to toggle link status.



Examples

10.3.2 IO 74xx595

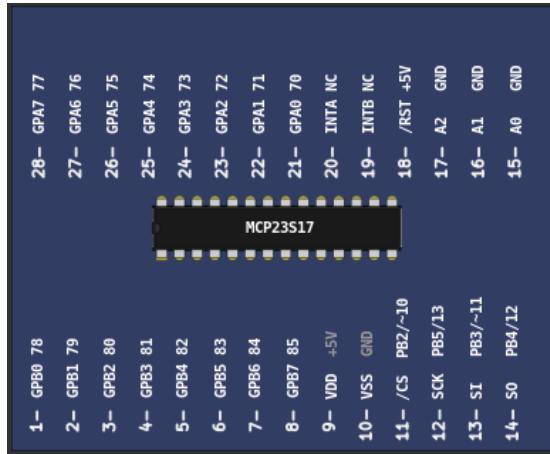
This is one 74xx595 serial input and parallel output 8 bit shift register.



Examples

10.3.3 IO MCP23S17

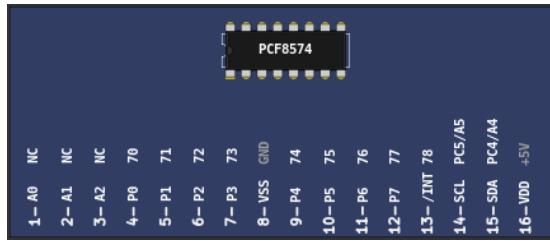
It is a MCP23S17 serial SPI IO expander part.



[Examples](#)

10.3.4 IO PCF8574

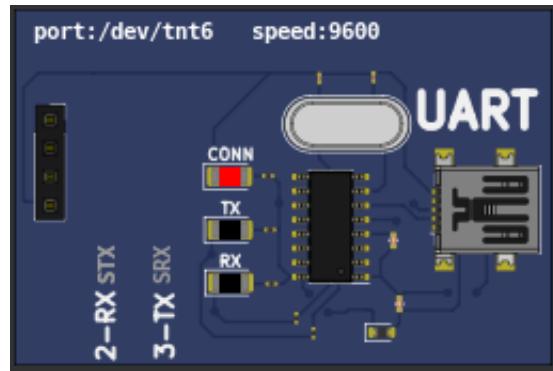
It is a PCF8574 serial I2C IO expander.



[Examples](#)

10.3.5 IO UART

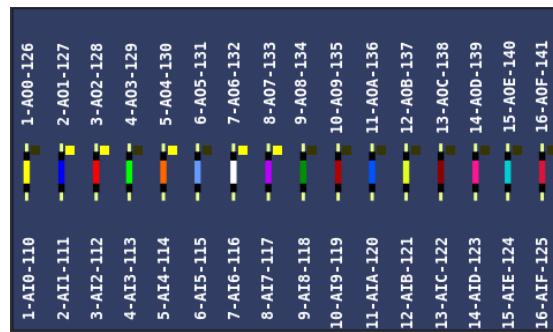
This part is a UART serial port. This part connects the hardware/software UART IO pins of microcontroller to one real/virtual PC serial port. To use virtual port is need to install a virtual port software, as described in [6](#).



[Examples](#)

10.3.6 Jumper Wires

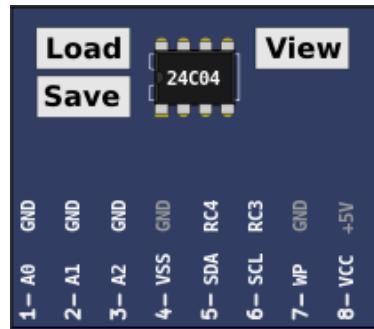
This part are formed by sixteen jumper wires. Each jumper has one input and one output. The jumper input must be connected to one pin output, the jumper output can be connected to multiple pin inputs. The jumper can be used to connect microcontroller pins or make connection between spare parts pins.



[Examples](#)

10.3.7 MEM 24CXXX

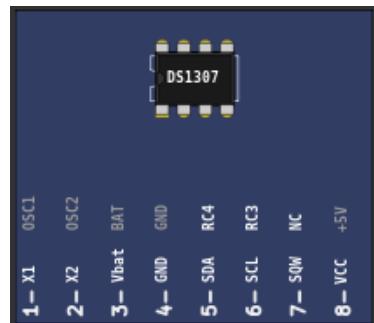
It is a 24CXXX serial I2C EEPROM part. There are support to the models 24C04 and 24C512.



[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](#)

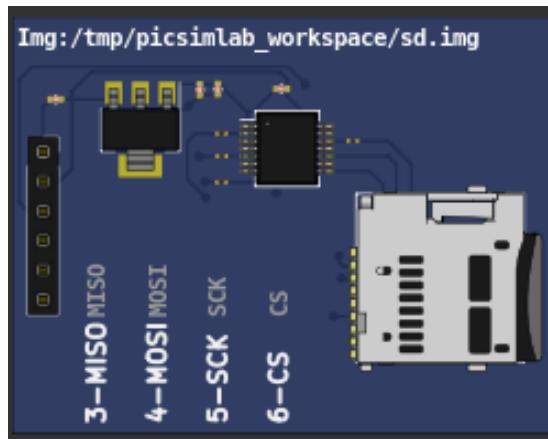
10.3.10 SD Card

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.

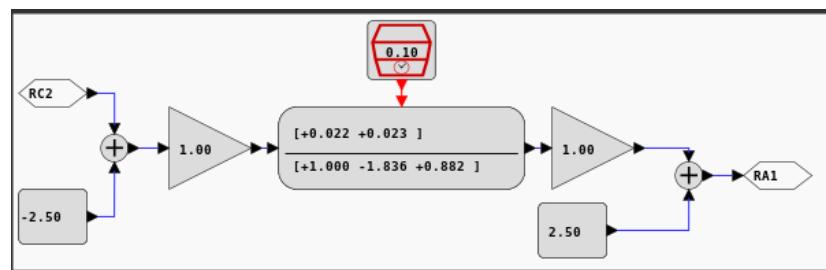


Examples

10.4 Virtual

10.4.1 D. Transfer Function

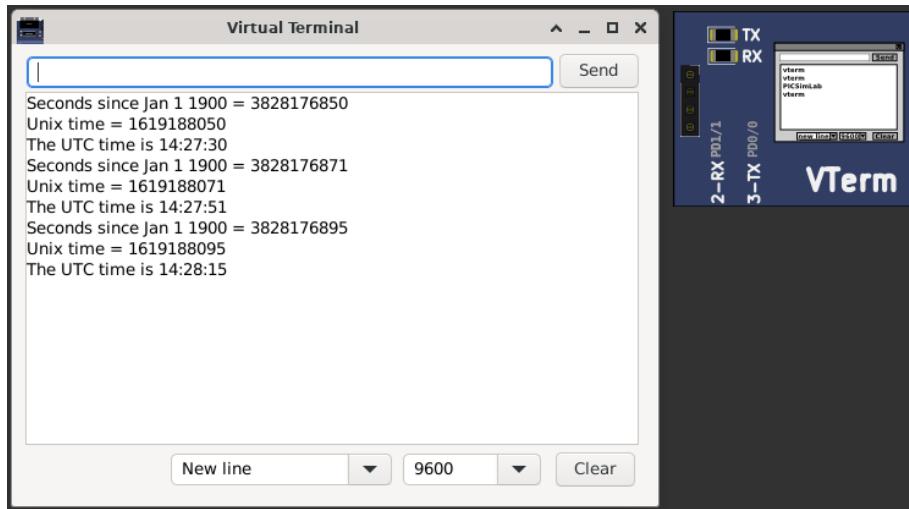
This is a discrete transfer function mathematical model.



Examples

10.4.2 IO Virtual Term

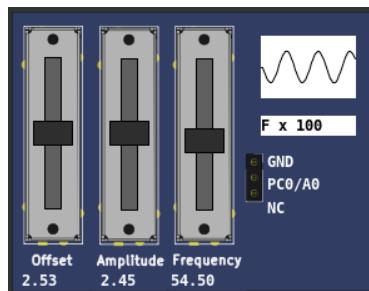
This part is a virtual serial terminal. This part can be used to read and write RX/TX pins UART signals. This part don't need the use or install of virtual serial ports on computer. Clik on terminal picture to open the terminal window.



Examples

10.4.3 Signal Generator

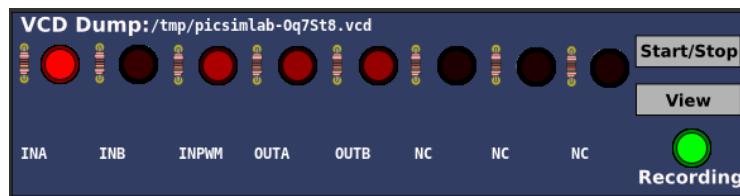
This part is a virtual signal generator with support for sine, square and triangular waves generation with amplitude and frequency adjustment.

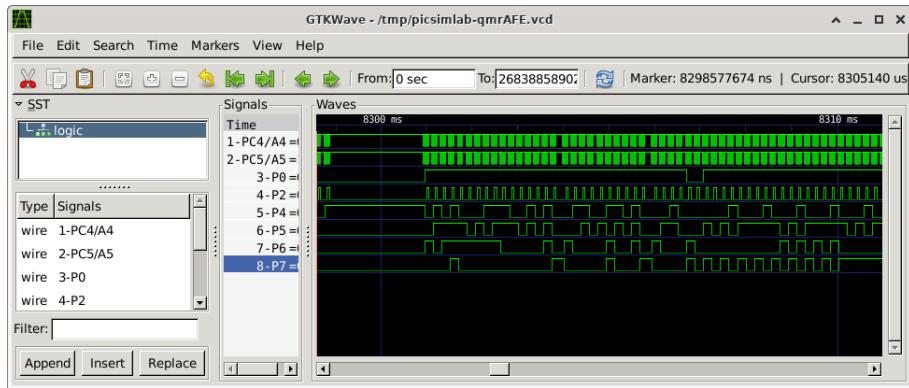


Examples

10.4.4 VCD Dump

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

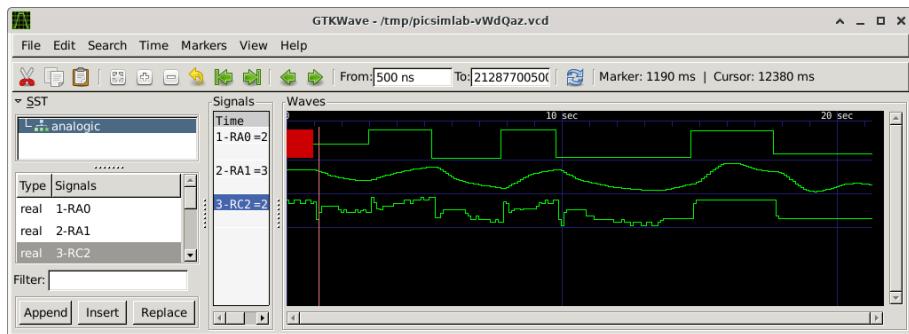
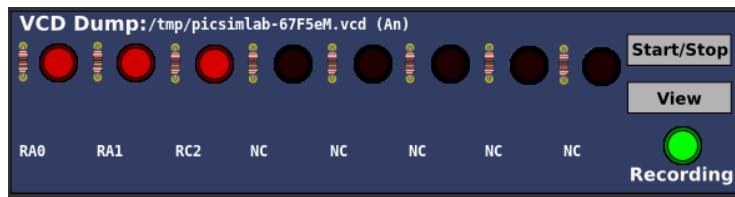




Examples

10.4.5 VCD Dump (Analogic)

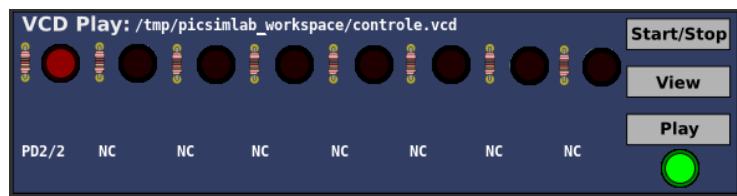
This part is a analog value file dump recorder. The file can be visualized with gtkwave.



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

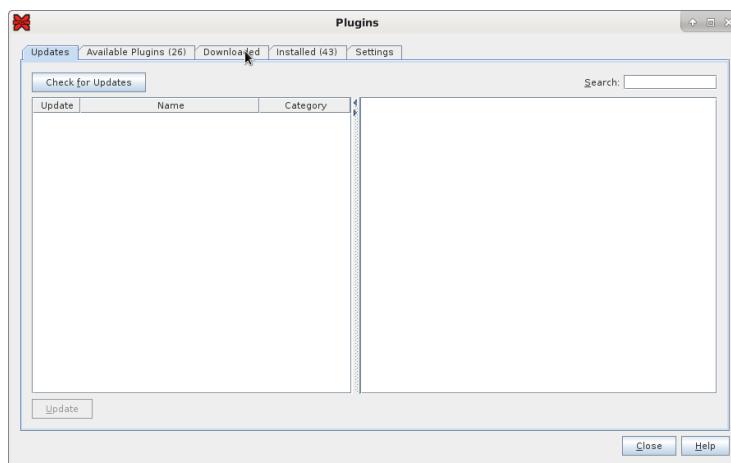
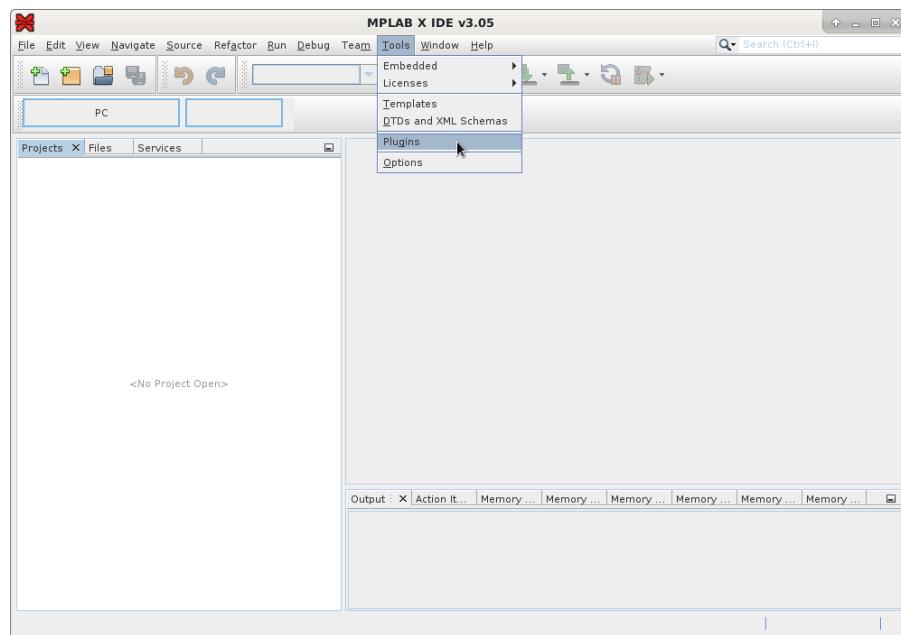
Links for download [MPLABX IDE](#) and [XC8 Compiler](#) installers. Download and install.

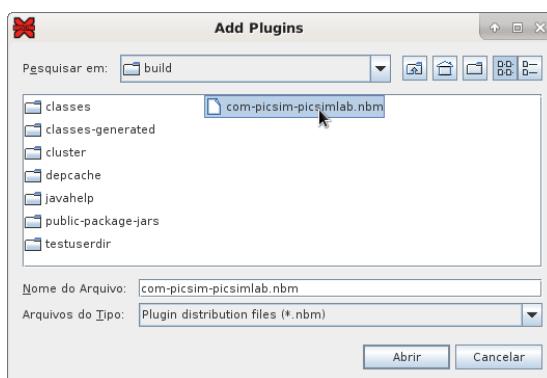
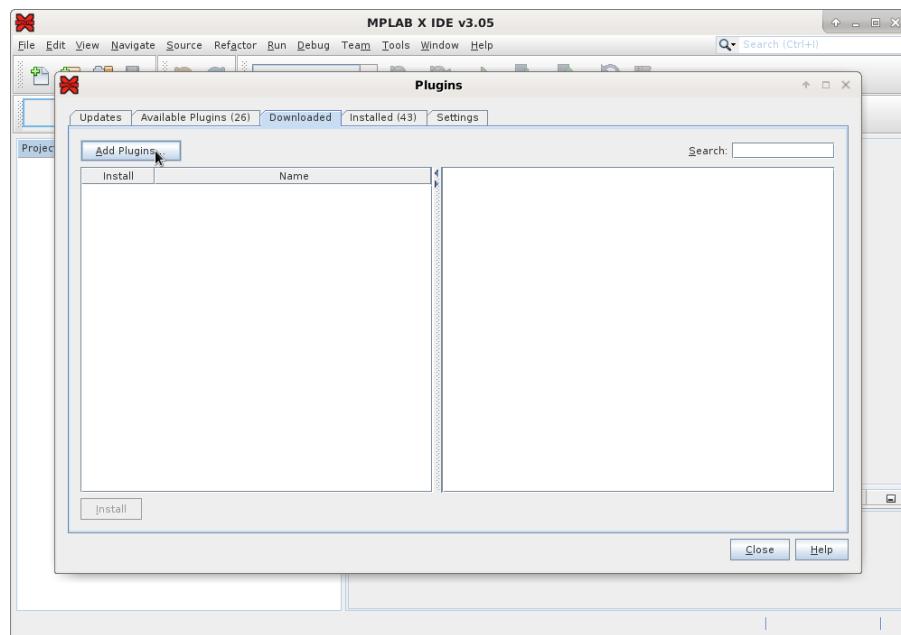
12.1.2 Install PICsimLab

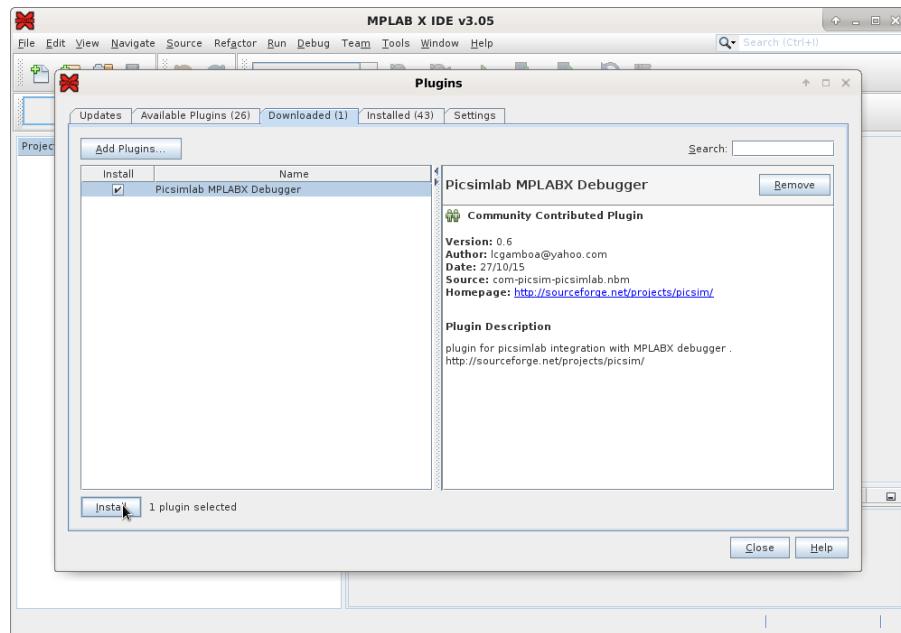
Link for download [PICsimLab-0.6](#) installer. Download and install

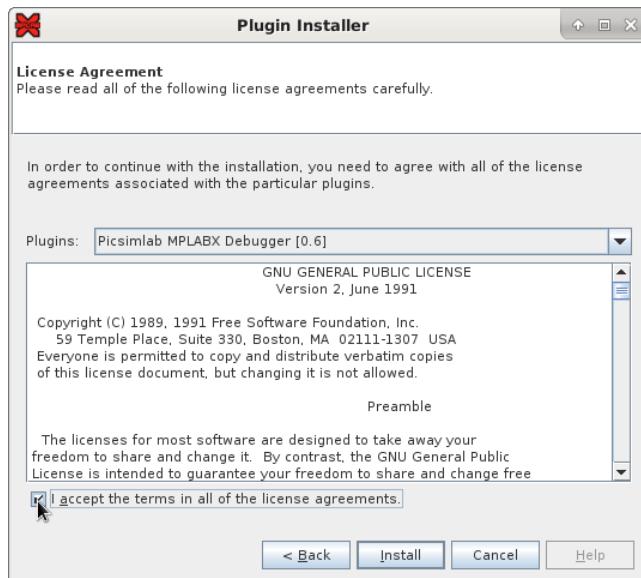
12.1.3 How to Install PicsimLab MPLABX Debugger plugin

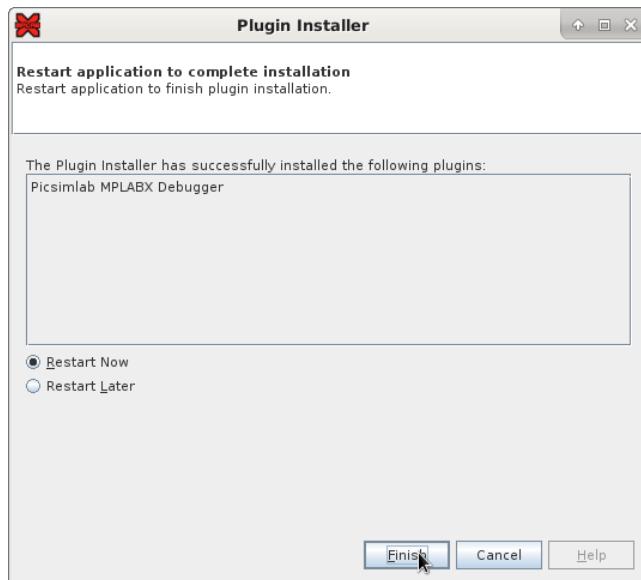
Link for download [PicsimLab MPLABX Debugger plugin \(com-picsim-picsimlab.nbm\)](#)





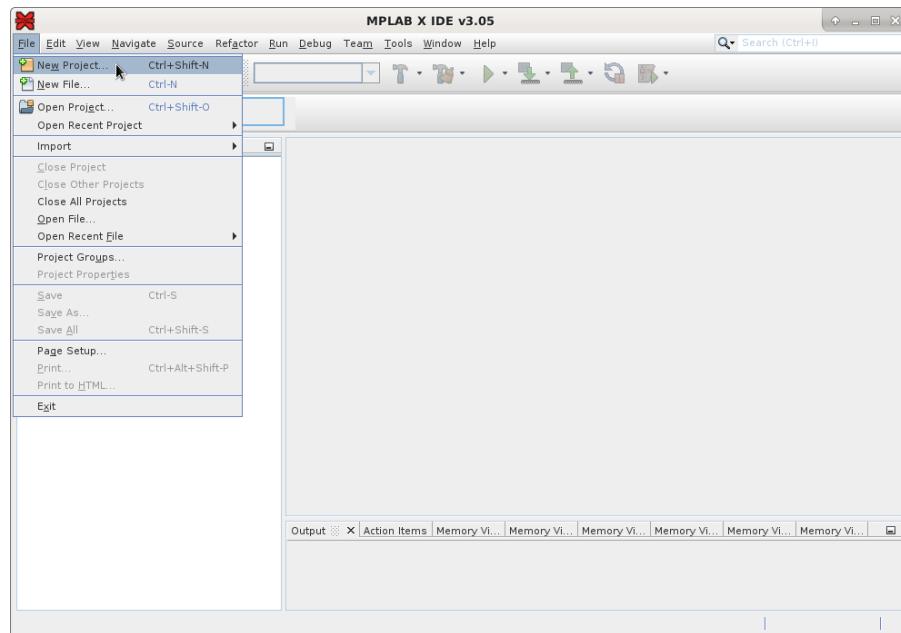


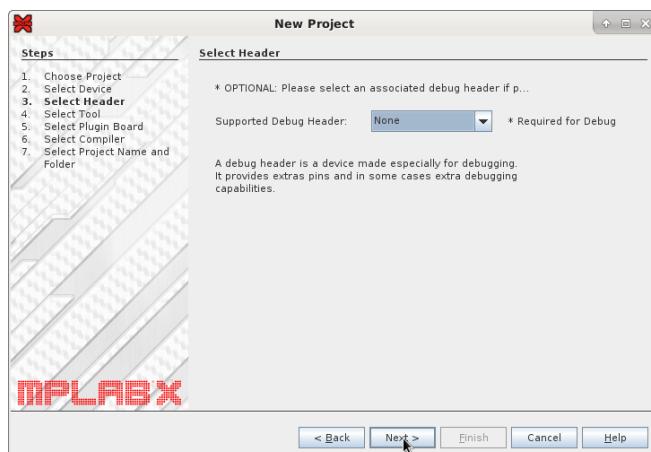
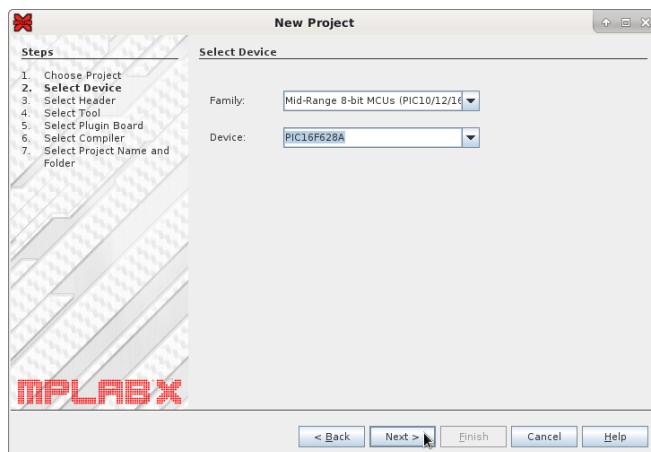
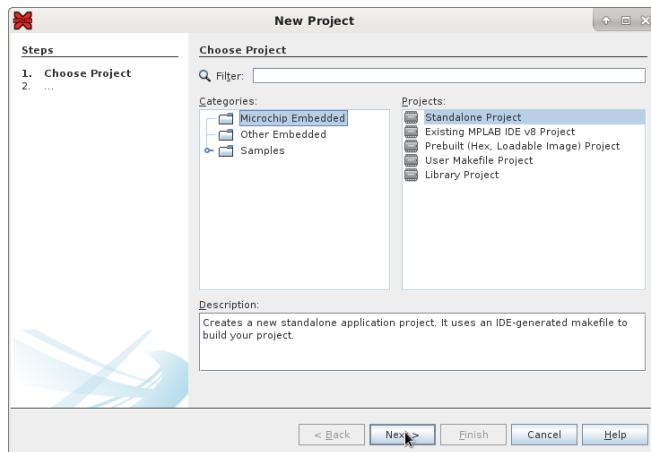


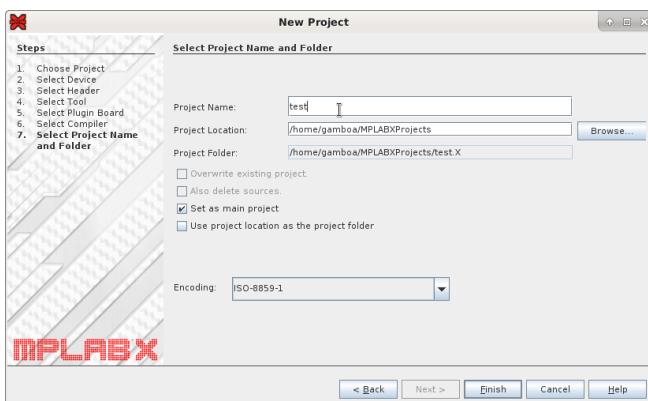
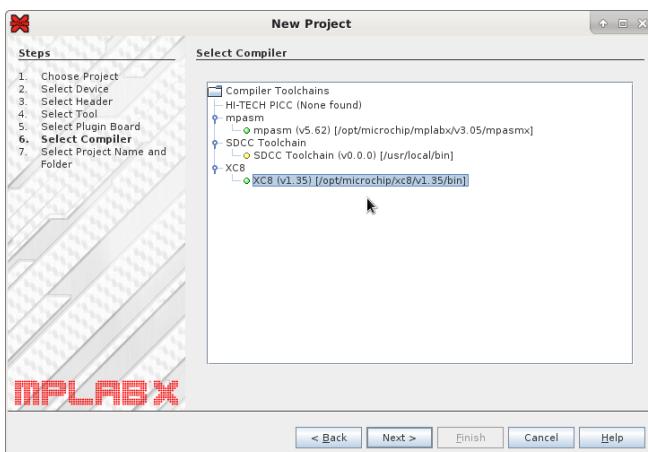
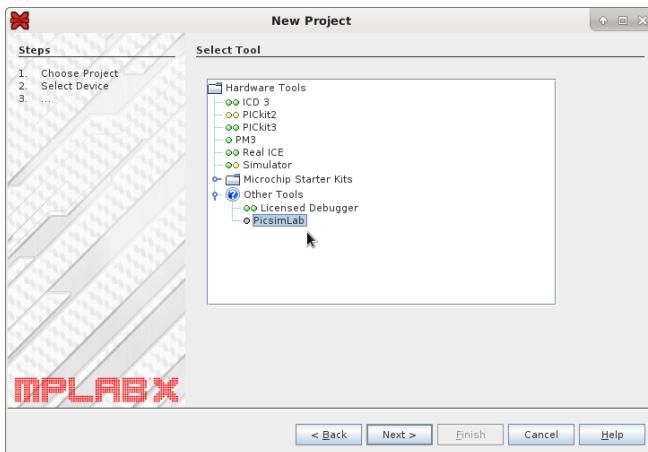


12.2 Configuring a New Project in MPLABX

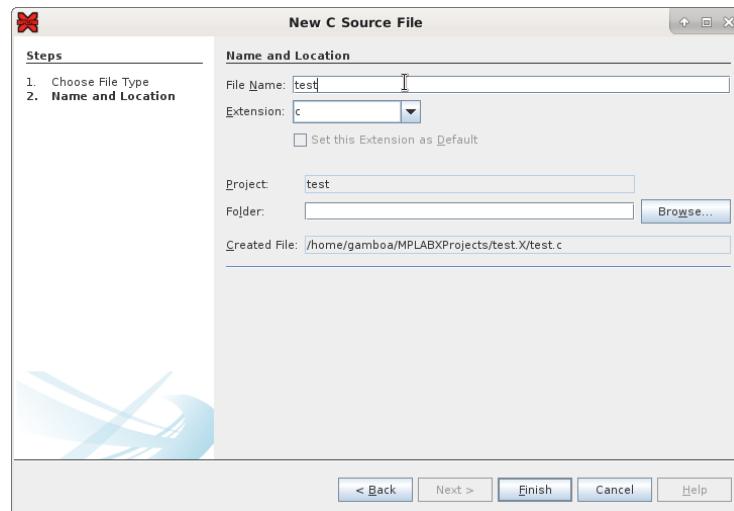
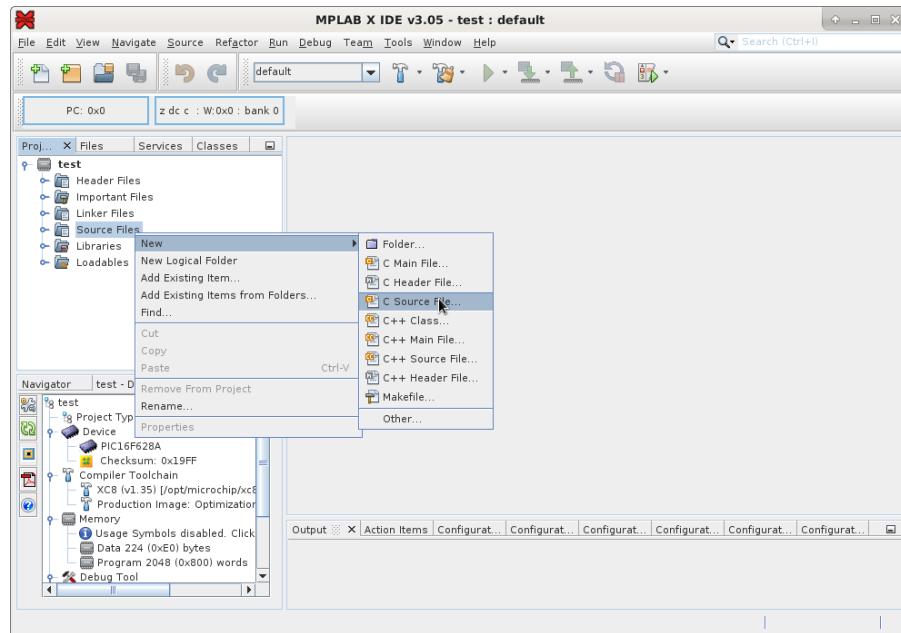
12.2.1 Project Creation



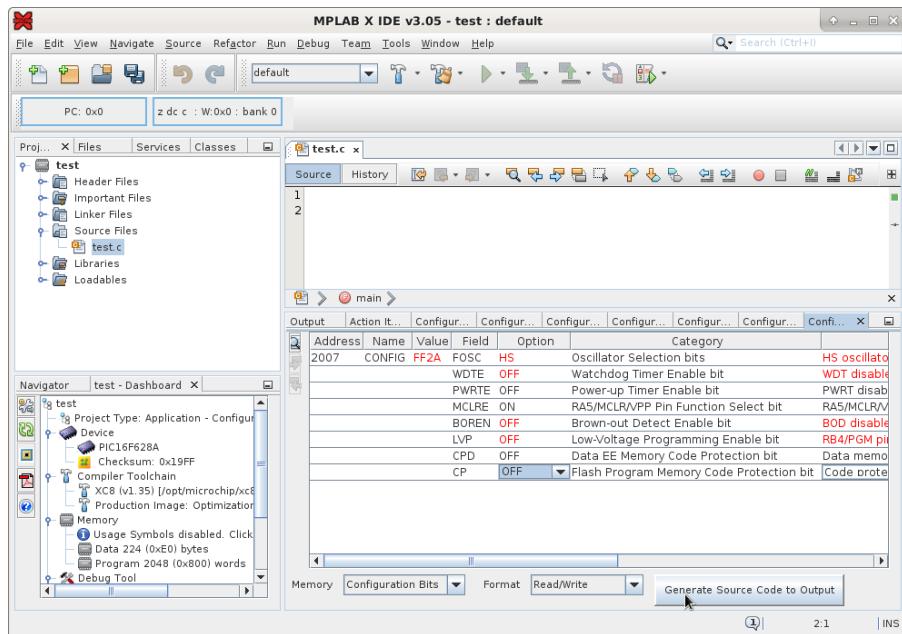
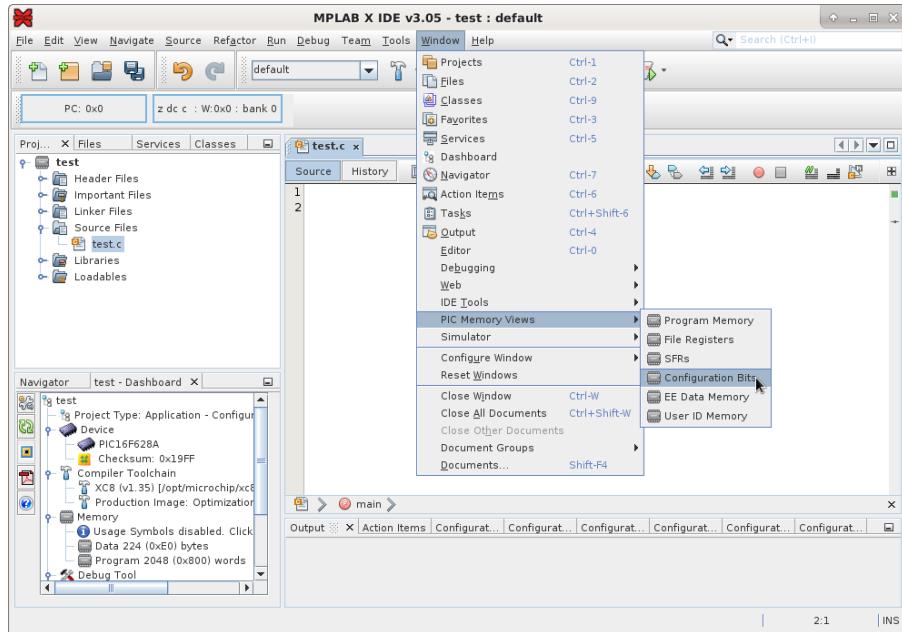


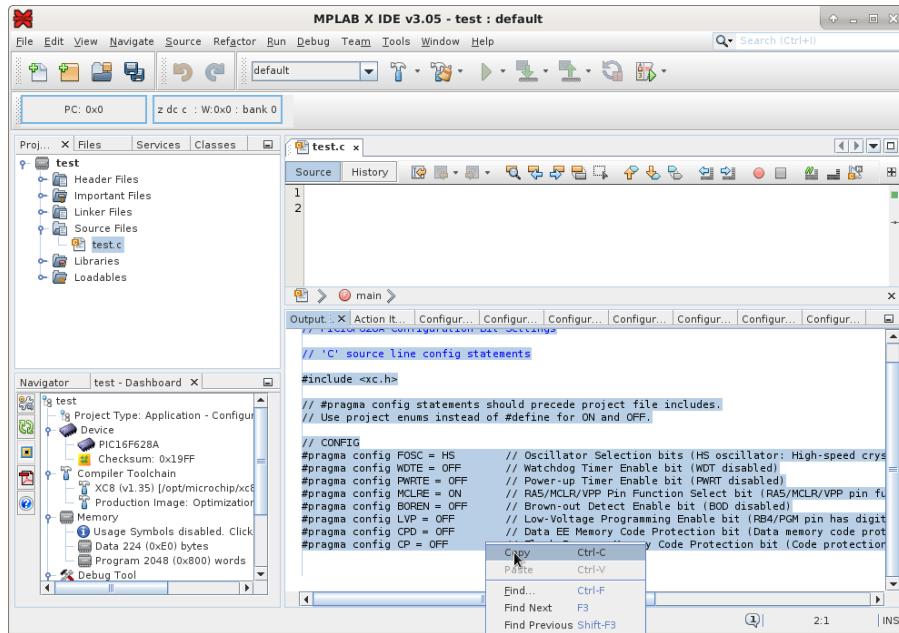


12.2.2 File Creation



12.2.3 PIC Configuration Bits





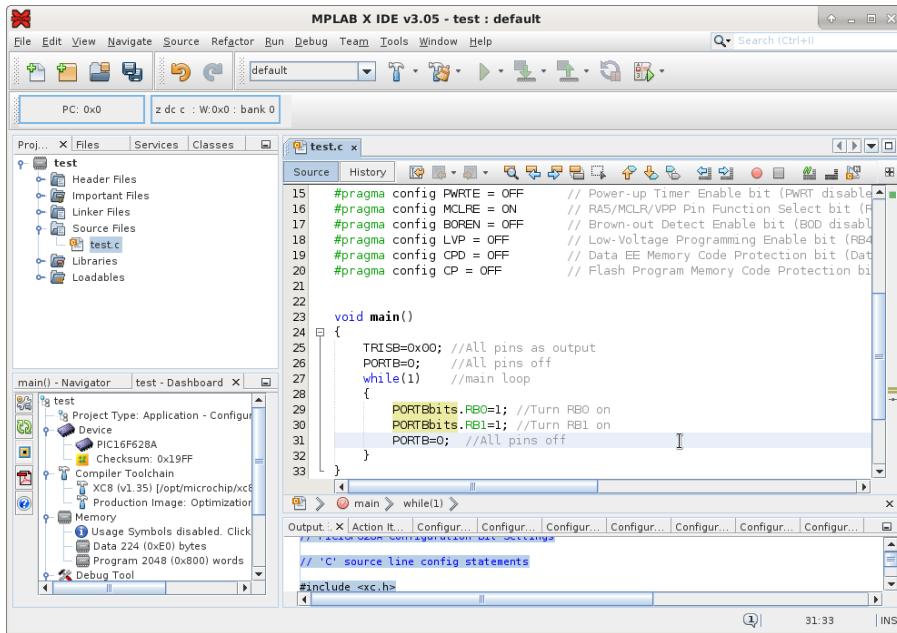
12.2.4 Code Example

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

```

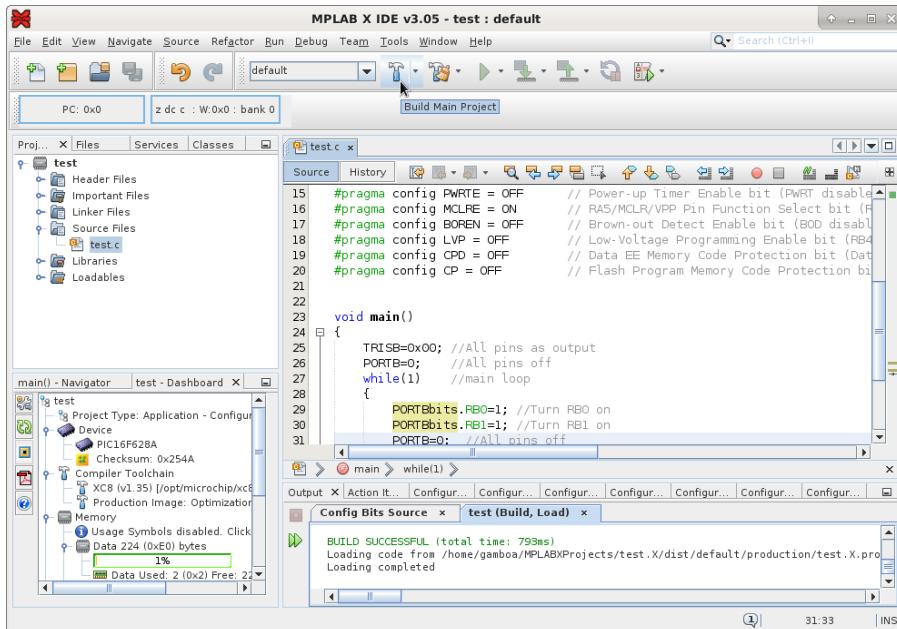
void main()
{
    TRISB=0x00; //All pins as output
    PORTB=0;     //All pins off
    while(1)      //main loop
    {
        PORTBbits.RB0=1; //Turn RB0 on
        PORTBbits.RB1=1; //Turn RB1 on
        PORTB=0; //All pins off
    }
}

```



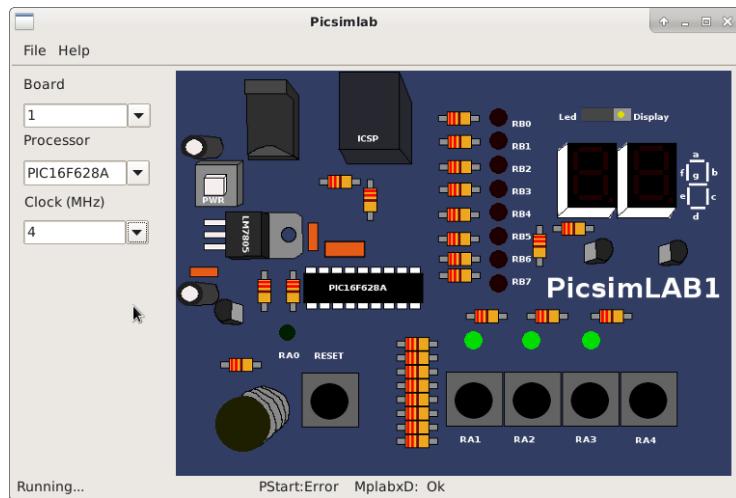
12.2.5 Building the Project

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



12.3 Program and Debug PICsimLab With MPLABX

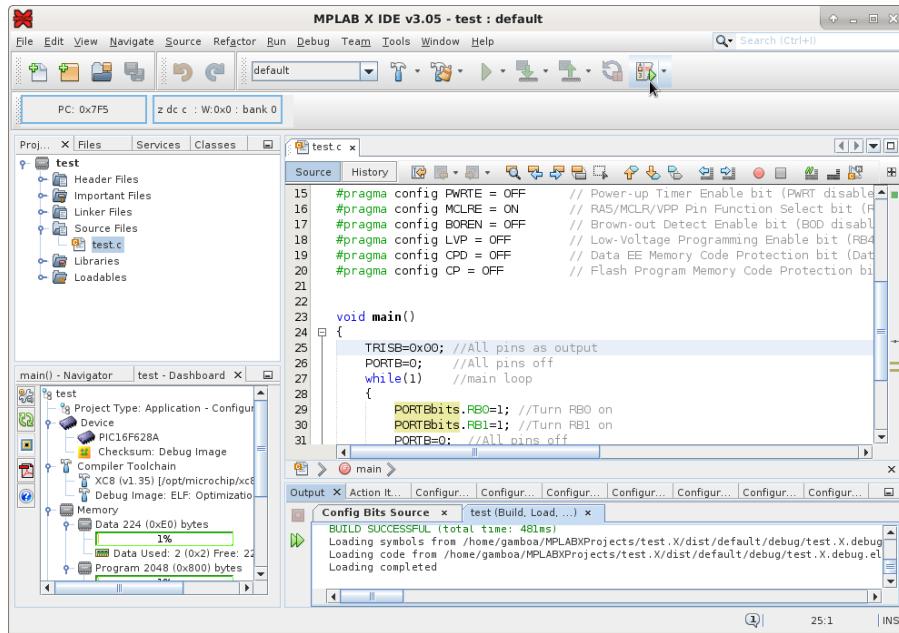
12.3.1 Starting PICsimLab



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.

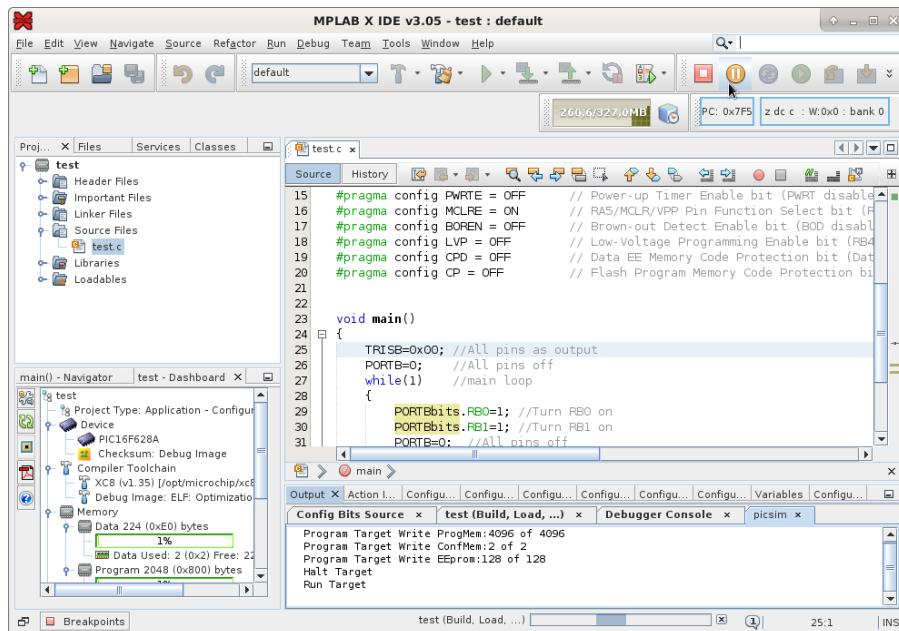
12.3.2 Programming PICsimLab

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



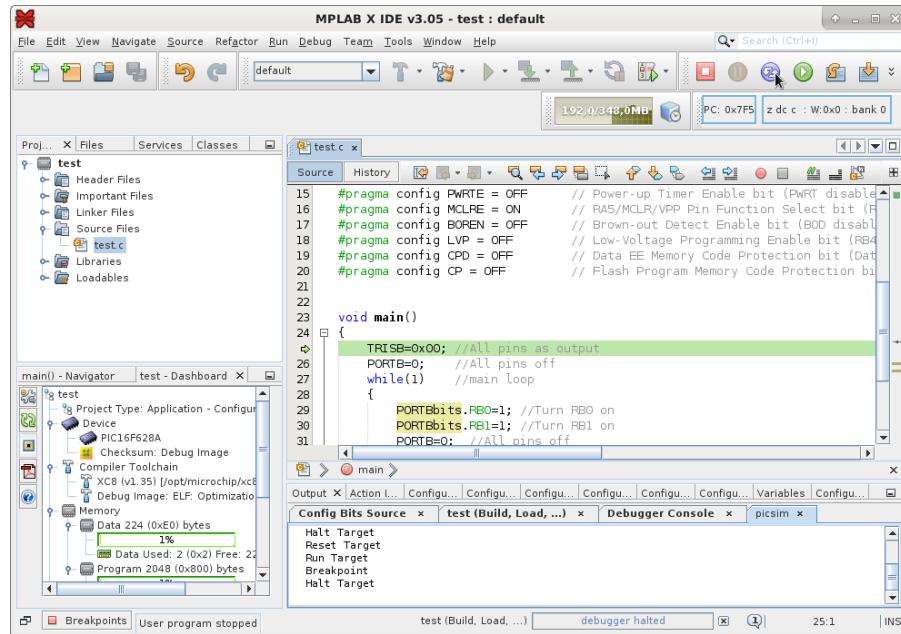
12.3.3 Pausing the Program

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



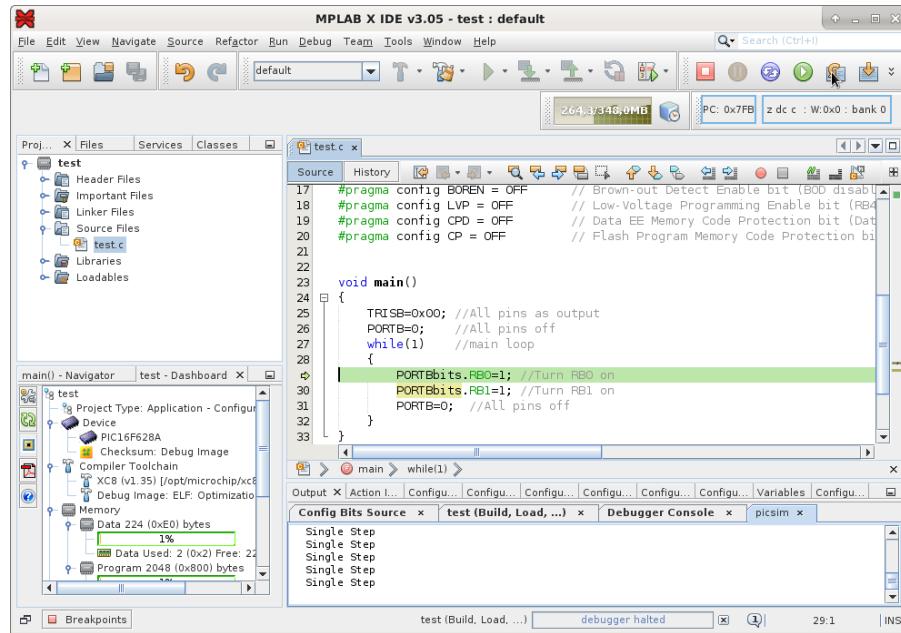
12.3.4 Restarting the Program

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

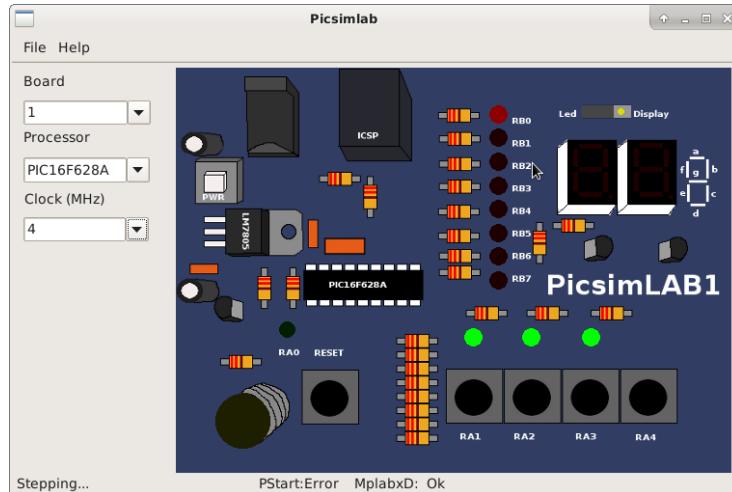


12.3.5 Running Step by Step

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

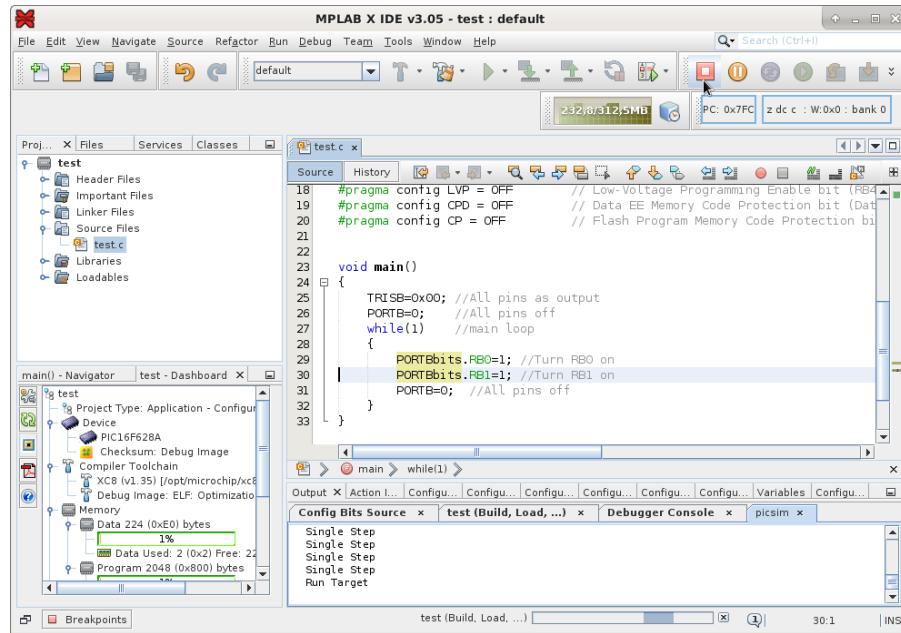


See in the PICsimLab the changes of each step.



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

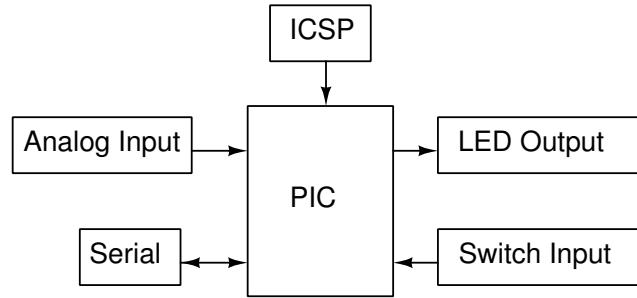
board_x for a number or name in your case):

- 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);

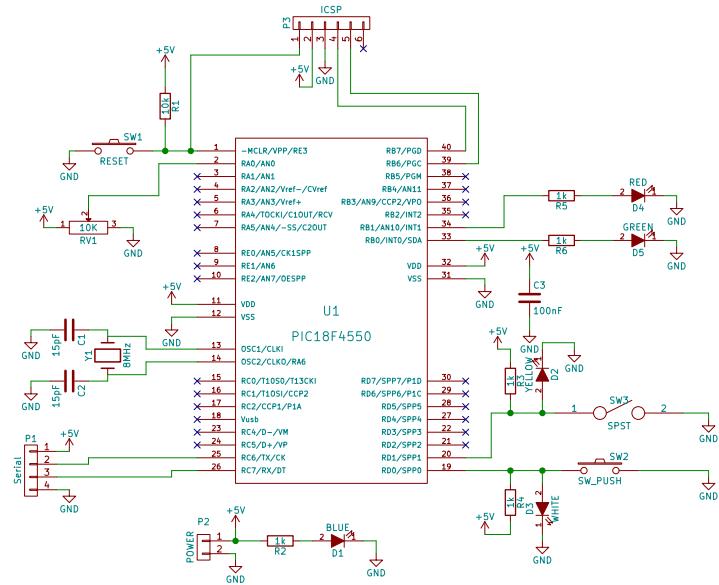
The third and last step is recompiling PICSimLab with new board support.

13.2.1 Board Hardware and Schematic

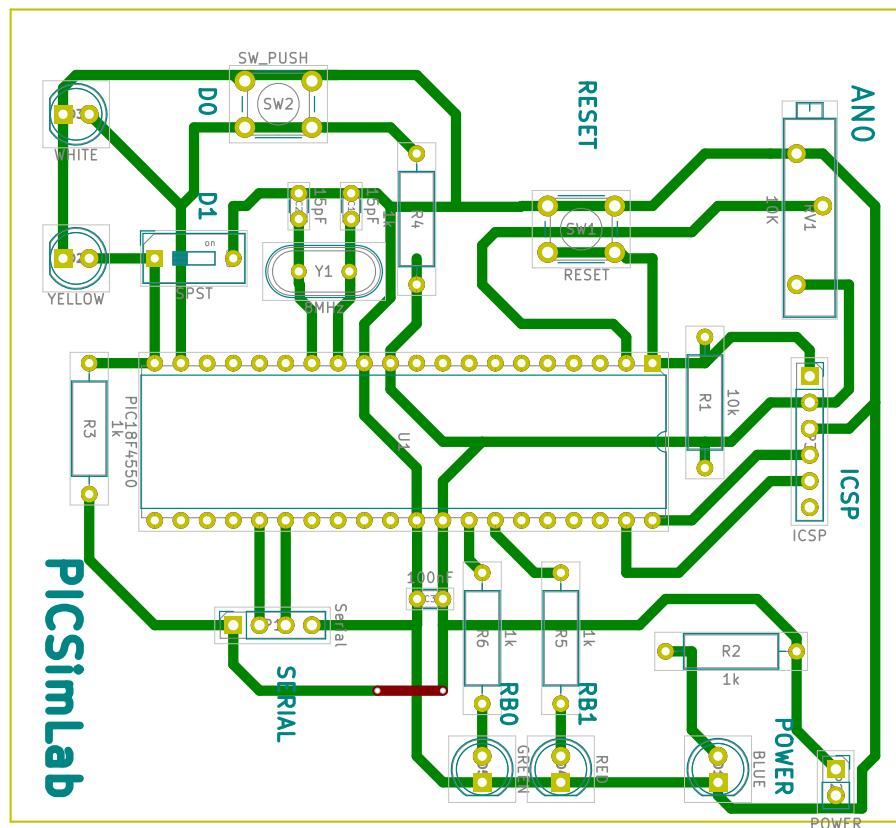
For this tutorial, the board created have the hardware shown in diagram below:



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

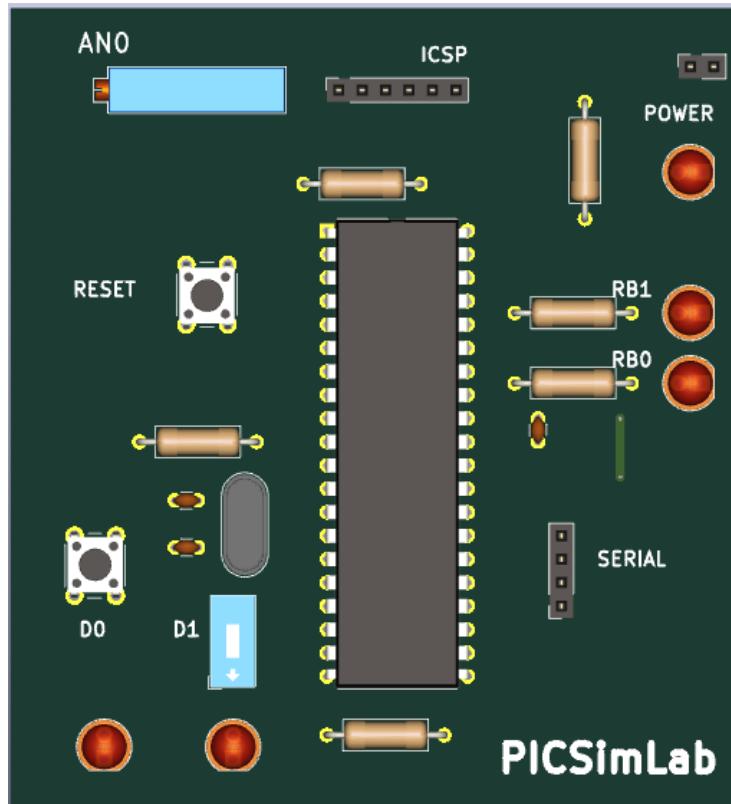


And the PCB layout was made in [Kicad](#) too. The PCB is not necessary if you have a real board.

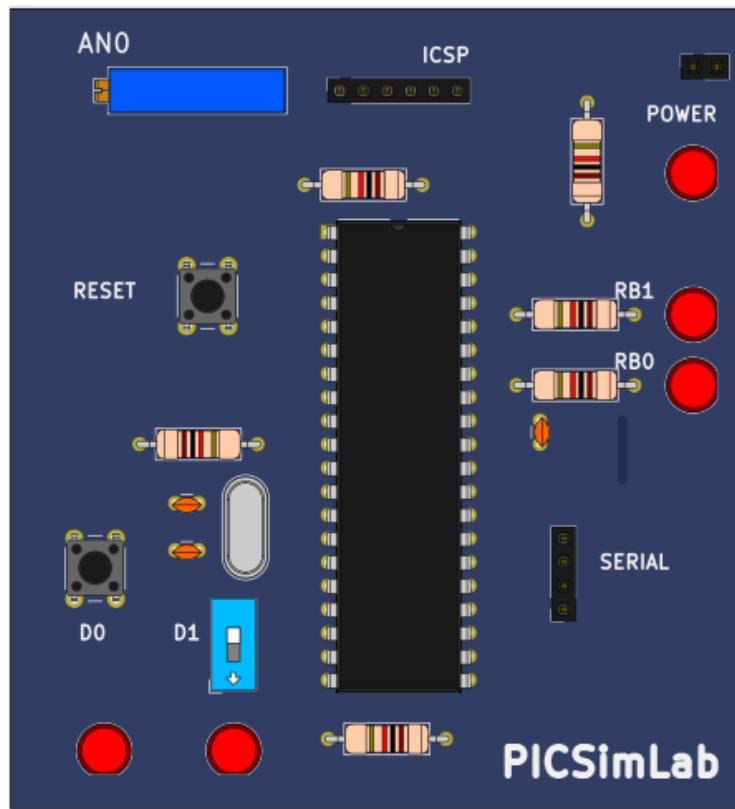


13.2.2 Board Picture

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”.



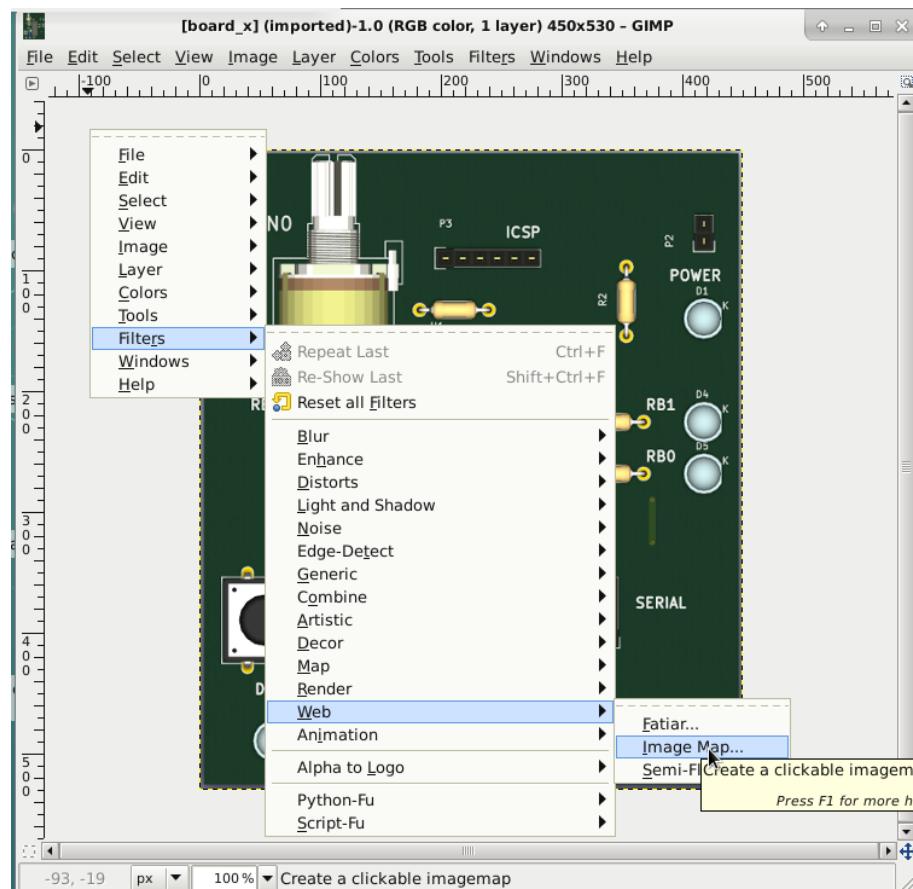
It is also possible to use images in SVG format for better viewing quality. [PCBDraw](#) can be used to convert a Kicad PCB project to an SVG image.



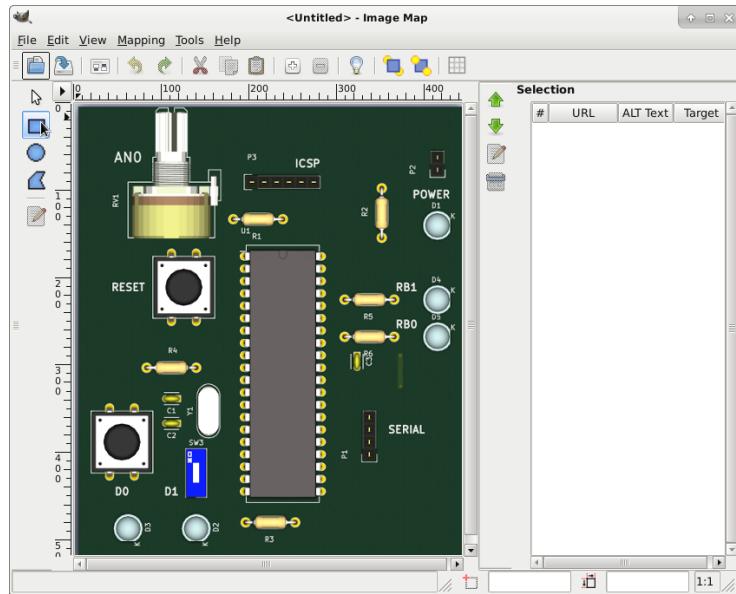
13.2.3 Picture maps

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](#).

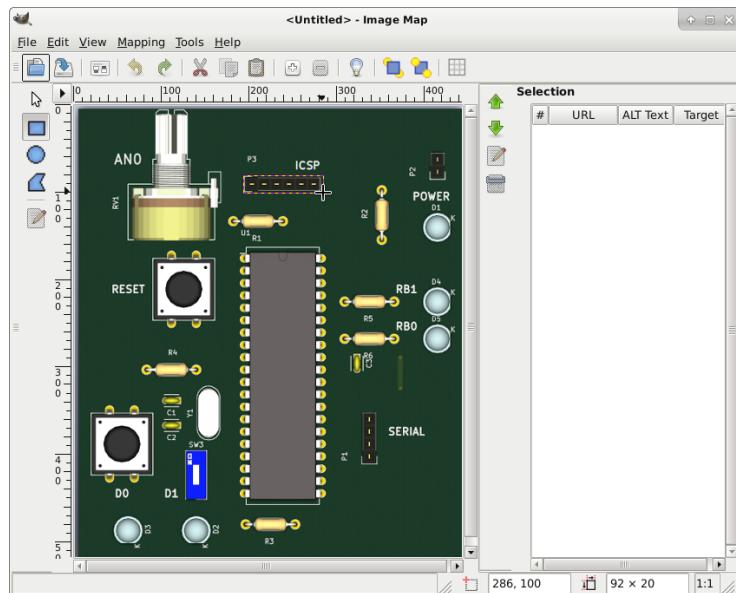
To start, in the GIMP, use the Filters->Web->Image Map to open image map editor window.



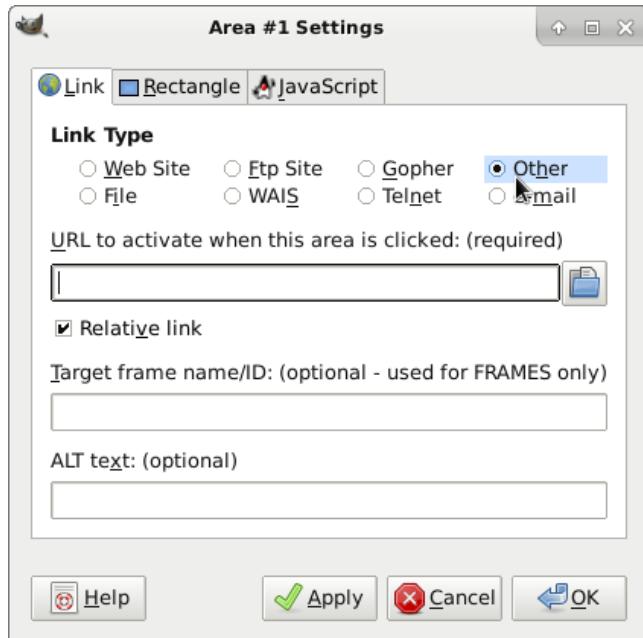
Then select rectangle or circle map on toolbar.



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.



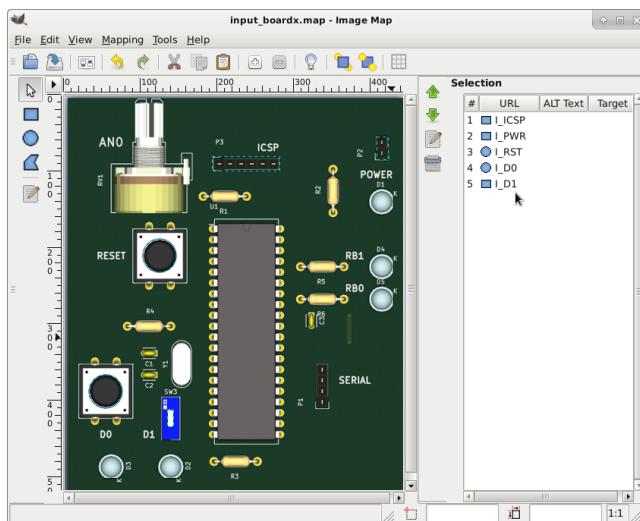
Board map

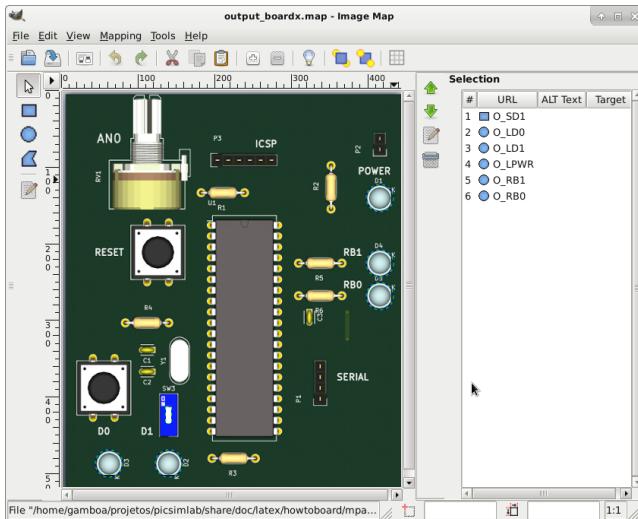
For this tutorial board, five input areas are marked:

- 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.

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.





Board map generated by Gimp image map editor and saved as “share/boards/X/board.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:lcganmboa@yahoo.com -->
9 <area shape="rect" coords="196,45,280,58" href="I_PG_ICSP" />
10 <area shape="rect" coords="409,30,441,46" href="I_SW_PWR" />
11 <area shape="rect" coords="133,379,142,401" href="B_SW_D1" />
12 <area shape="rect" coords="74,42,156,61" href="B_PO_1" />
13 <area shape="rect" coords="105,162,138,195" href="B_PB_RST" />
14 <area shape="rect" coords="37,327,70,360" href="B_PB_D0" />
15 <area shape="circle" coords="59,454,17" href="O_LD_LD0" />
16 <area shape="circle" coords="137,454,17" href="O_LD_LD1" />
17 <area shape="circle" coords="418,102,17" href="O_LD_LPWR" />
18 <area shape="circle" coords="418,189,17" href="O_LD_RB1" />
19 <area shape="circle" coords="418,232,17" href="O_LD_RB0" />
20 <area shape="rect" coords="227,220,247,328" href="O_MP_CPU" />
21 </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) : 2015-2021 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 "bsim_picsim.h"
32
33 #define      BOARD_X_Name "X"
34
35 //new board class must be derived from board class defined in board.h
36 class cboard_x:public bsim_picsim
37 {
38     private:
39         unsigned char p_BT1;           //first board push button in RD0
40         unsigned char p_BT2;           //second board switch in RD1
41
42         //value of potentiometer
43         unsigned char pot1;
44         //flag to control if potentiometer is active

```

```
45     unsigned char active;
46
47     //controls to be added in simulator window
48     CGauge *gauge1;    //gauge to show mean value of RB0
49     CGauge *gauge2;    //gauge to show mean value of RB1
50     CLabel *label1;   //label of gauge RB0
51     CLabel *label2;   //label of gauge RB1
52
53     //Register controls for remote interface called once on board creation
54     void RegisterRemoteControl(void);
55
56     lxColor color1;//LEDs color 1
57     lxColor color2;//LEDs color 2
58     lxFont font;
59
60     public:
61         //Constructor called once on board creation
62         cboard_x(void);
63         //Destructor called once on board destruction
64         ~cboard_x(void);
65         //Return the board name
66         lxString GetName(void) {return lxT(BOARD_x_Name); };
67         //Return the about info of board
68         lxString GetAboutInfo(void) {return lxT("L.C. Gamboa \n <lcgamboa@yahoo.com>"); };
69         //Called ever 100ms to draw board
70         void Draw(CDraw *draw);
71         void Run_CPU(void);
72         //Return a list of board supported microcontrollers
73         lxString GetSupportedDevices(void) {return lxT("PIC16F877A,PIC18F4550,PIC18F4620,"); };
74         //Reset board status
75         void Reset(void);
76         //Event on the board
77         void EvMouseButtonPress(uint button, uint x, uint y,uint state);
78         //Event on the board
79         void EvMouseButtonRelease(uint button, uint x, uint y,uint state);
80         //Event on the board
81         void EvMouseMove(uint button, uint x, uint y, uint state);
82         //Event on the board
83         void EvKeyPress(uint key,uint mask);
84         //Event on the board
85         void EvKeyRelease(uint key,uint mask);
86         //Called ever 1s to refresh status
87         void RefreshStatus(void);
88         //Called to save board preferences in configuration file
89         void WritePreferences(void);
90         //Called whe configuration file load preferences
91         void ReadPreferences(char *name,char *value);
92         //return the input ids numbers of names used in input map
93         unsigned short get_in_id(char * name);
```

```
93     //return the output ids numbers of names used in output map
94     unsigned short get_out_id(char * name);
95 };
96
97 #endif           /* BOARD_X_H */
```

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) : 2015-2021 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_POT1, //potentiometer
36     I_ICSP, //ICSP connector
37     I_PWR, //Power button
38     I_RST, //Reset button
39     I_D0, //RDO push button
40     I_D1 //RD1 switch
41 };
42
43 /* ids of outputs of output map*/
44 enum

```

```

45  {
46    O_POT1, //potentiometer
47    O_RST, //Reset button
48    O_SD1, //switch position (On/Off)
49    O_LD0, //LED on RD0 push button
50    O_LD1, //LED on RD1 switch
51    O_LPWR, //Power LED
52    O_RB0, //LED on RB0 output
53    O_RB1, //LED on RB1 output
54    O_BD0, //RD1 switch
55    O_CPU //CPU name
56  };
57
58 //return the input ids numbers of names used in input map
59
60 unsigned short
61 cboard_x::get_in_id(char * name)
62 {
63   if (strcmp (name, "PG_ICSP") == 0) return I_ICSP;
64   if (strcmp (name, "SW_PWR") == 0) return I_PWR;
65   if (strcmp (name, "PB_RST") == 0) return I_RST;
66   if (strcmp (name, "PB_D0") == 0) return I_D0;
67   if (strcmp (name, "SW_D1") == 0) return I_D1;
68   if (strcmp (name, "PO_1") == 0) return I_POT1;
69
70   printf ("Error input '%s' don't have a valid id! \n", name);
71   return -1;
72 }
73
74 //return the output ids numbers of names used in output map
75
76 unsigned short
77 cboard_x::get_out_id(char * name)
78 {
79
80   if (strcmp (name, "SW_D1") == 0) return O_SD1;
81   if (strcmp (name, "LD_LD0") == 0) return O_LD0;
82   if (strcmp (name, "LD_LD1") == 0) return O_LD1;
83   if (strcmp (name, "LD_LPWR") == 0) return O_LPWR;
84   if (strcmp (name, "LD_RB1") == 0) return O_RB1;
85   if (strcmp (name, "LD_RB0") == 0) return O_RB0;
86   if (strcmp (name, "PB_D0") == 0) return O_BD0;
87   if (strcmp (name, "PO_1") == 0) return O_POT1;
88   if (strcmp (name, "PB_RST") == 0) return O_RST;
89   if (strcmp (name, "MP_CPU") == 0) return O_CPU;
90
91   printf ("Error output '%s' don't have a valid id! \n", name);
92   return 1;

```

```
93    }
94
95 //Constructor called once on board creation
96
97 cboard_x::cboard_x(void):
98     font(10, lxFONTFAMILY_TELETYPE, lxFONTSTYLE_NORMAL, lxFONTWEIGHT_BOLD)
99 {
100     Proc = "PIC18F4550"; //default microcontroller if none defined in preferences
101     ReadMaps (); //Read input and output board maps
102
103     pot1 = 100;
104
105     active = 0;
106
107     //controls properties and creation
108     //gauge1
109     gauge1 = new CGauge ();
110     gauge1->SetFOwner (&Window1);
111     gauge1->SetName (lxT ("gauge1_px"));
112     gauge1->SetX (13);
113     gauge1->SetY (382 - 160);
114     gauge1->SetWidth (140);
115     gauge1->SetHeight (20);
116     gauge1->SetEnable (1);
117     gauge1->SetVisible (1);
118     gauge1->SetRange (100);
119     gauge1->SetValue (0);
120     gauge1->SetType (4);
121     Window1.CreateChild (gauge1);
122     //gauge2
123     gauge2 = new CGauge ();
124     gauge2->SetFOwner (&Window1);
125     gauge2->SetName (lxT ("gauge2_px"));
126     gauge2->SetX (12);
127     gauge2->SetY (330 - 160);
128     gauge2->SetWidth (140);
129     gauge2->SetHeight (20);
130     gauge2->SetEnable (1);
131     gauge2->SetVisible (1);
132     gauge2->SetRange (100);
133     gauge2->SetValue (0);
134     gauge2->SetType (4);
135     Window1.CreateChild (gauge2);
136     //label2
137     label2 = new CLabel ();
138     label2->SetFOwner (&Window1);
139     label2->SetName (lxT ("label2_px"));
140     label2->SetX (12);
```

```
141    label2->SetY (306 - 160);
142    label2->SetWidth (60);
143    label2->SetHeight (20);
144    label2->SetEnable (1);
145    label2->SetVisible (1);
146    label2->SetText (lxT ("RB0"));
147    label2->SetAlign (1);
148    Window1.CreateChild (label2);
149    //label3
150    label3 = new CLabel ();
151    label3->SetFOwner (&Window1);
152    label3->SetName (lxT ("label3_px"));
153    label3->SetX (13);
154    label3->SetY (357 - 160);
155    label3->SetWidth (60);
156    label3->SetHeight (20);
157    label3->SetEnable (1);
158    label3->SetVisible (1);
159    label3->SetText (lxT ("RB1"));
160    label3->SetAlign (1);
161    Window1.CreateChild (label3);
162 }
163
164 //Destructor called once on board destruction
165
166 cboard_x::~cboard_x (void)
167 {
168     //controls destruction
169     Window1.DestroyChild (gauge1);
170     Window1.DestroyChild (gauge2);
171     Window1.DestroyChild (label2);
172     Window1.DestroyChild (label3);
173 }
174
175 //Reset board status
176
177 void
178 cboard_x::Reset (void)
179 {
180     pic_reset (1);
181
182     p_BT1 = 1; //set push button in default state (high)
183
184     //write button state to pic pin 19 (RD0)
185     pic_set_pin (19, p_BT1);
186     //write switch state to pic pin 20 (RD1)
187     pic_set_pin (20, p_BT2);
188 }
```

```
189 //verify serial port state and refresh status bar
190 #ifndef _WIN_
191     if (pic.serial[0].serialfd > 0)
192     #else
193         if (pic.serial[0].serialfd != INVALID_HANDLE_VALUE)
194     #endif
195         Window1.statusbar1.SetField (2, lxT ("Serial: ") +
196                                     lxString::FromAscii (SERIALDEVICE) + lxT (":") + itoa (pic.serial[0].
197                                     lxString () .Format ("%4.1f", fabs ((100.0 * pic.serial[0].serialexbau
198                                     pic.serial[0].serialbaud) / pic.s
199
200     else
201         Window1.statusbar1.SetField (2, lxT ("Serial: ") +
202                                     lxString::FromAscii (SERIALDEVICE) + lxT (" (ERROR)"));
203
204     if (use_spare)Window5.Reset ();
205
206     RegisterRemoteControl ();
207 }
208
209 //Register variables to be controled by remote control
210
211 void
212 cboard_x::RegisterRemoteControl (void)
213 {
214     for (int i = 0; i < inputc; i++)
215     {
216         switch (input[i].id)
217         {
218             case I_D0:
219                 input[i].status = &p_BT1;
220                 break;
221             case I_D1:
222                 input[i].status = &p_BT2;
223                 break;
224             case I_POT1:
225                 input[i].status = &pot1;
226                 break;
227         }
228     }
229
230     for (int i = 0; i < outputc; i++)
231     {
232         switch (output[i].id)
233         {
234             case O_RB0:
235                 output[i].status = &pic.pins[32].oavalue;
236                 break;
```

```

237     case O_RB1:
238         output[i].status = &pic.pins[33].oavalue;
239         break;
240     case O_LD0:
241         output[i].status = &pic.pins[18].oavalue;
242         break;
243     case O_LD1:
244         output[i].status = &pic.pins[19].oavalue;
245         break;
246     }
247 }
248 }
249
250 //Called every 1s to refresh status
251
252 void
253 cboard_x::RefreshStatus(void)
254 {
255     //verify serial port state and refresh status bar
256 #ifndef _WIN_
257     if (pic.serial[0].serialfd > 0)
258 #else
259     if (pic.serial[0].serialfd != INVALID_HANDLE_VALUE)
260 #endif
261     Window1.statusbar1.SetField (2, lxT ("Serial: ") +
262                                 lxString::FromAscii (SERIALDEVICE) + lxT (":") + itoa (pic.serial[0].
263                                         lxString ().Format ("%4.1f", fabs ((100.0 * pic.serial[0].serialexbau-
264                                         pic.serial[0].serialbaud) / pic.s
265     else
266     Window1.statusbar1.SetField (2, lxT ("Serial: ") +
267                                 lxString::FromAscii (SERIALDEVICE) + lxT (" (ERROR) "));
268
269 }
270
271 //Called to save board preferences in configuration file
272
273 void
274 cboard_x::WritePreferences(void)
275 {
276     //write selected microcontroller of board_x to preferences
277     Window1.saveprefs (lxT ("X_proc"), Proc);
278     //write switch state of board_x to preferences
279     Window1.saveprefs (lxT ("X_bt2"), lxString ().Format ("%i", p_BT2));
280     //write microcontroller clock to preferences
281     Window1.saveprefs (lxT ("X_clock"), lxString ().Format ("%2.1f", Window1.GetClock ()));
282     //write potentiometer position to preferences
283     Window1.saveprefs (lxT ("X_pot1"), lxString ().Format ("%i", pot1));
284 }
```

```
285 //Called whe configuration file load preferences
286
287
288 void
289 cboard_x::ReadPreferences(char *name, char *value)
290 {
291     //read switch state of board_x of preferences
292     if (!strcmp (name, "X_bt2"))
293     {
294         if (value[0] == '0')
295             p_BT2 = 0;
296         else
297             p_BT2 = 1;
298     }
299     //read microcontroller of preferences
300     if (!strcmp (name, "X_proc"))
301     {
302         Proc = value;
303     }
304     //read microcontroller clock
305     if (!strcmp (name, "X_clock"))
306     {
307         Window1.SetClock (atof (value));
308     }
309
310     //read potentiometer position
311     if (!strcmp (name, "X_pot1"))
312     {
313         pot1 = atoi (value);
314     }
315 }
316
317
318 //Event on the board
319
320 void
321 cboard_x::EvKeyPress(uint key, uint mask)
322 {
323     //if keyboard key 1 is pressed then activate button (state=0)
324     if (key == '1')
325     {
326         p_BT1 = 0;
327         output_ids[0_BD0]->update = 1;
328     }
329
330     //if keyboard key 2 is pressed then toggle switch state
331     if (key == '2')
332     {
```

```

333     p_BT2 ^= 1;
334     output_ids[O_SD1]->update = 1;
335 }
336
337 }
338
339 //Event on the board
340
341 void
342 cboard_x::EvKeyRelease(uint key, uint mask)
343 {
344 //if keyboard key 1 is pressed then deactivate button (state=1)
345 if (key == '1')
346 {
347     p_BT1 = 1;
348     output_ids[O_BD0]->update = 1;
349 }
350
351 }
352
353 //Event on the board
354
355 void
356 cboard_x::EvMouseButtonPress(uint button, uint x, uint y, uint state)
357 {
358
359 int i;
360
361 //search for the input area which owner the event
362 for (i = 0; i < inputc; i++)
363 {
364     if (((input[i].x1 <= x)&&(input[i].x2 >= x))&&((input[i].y1 <= y)&&
365                                         (input[i].y2 >= y)))
366     {
367
368         switch (input[i].id)
369         {
370             //if event is over I_ISCP area then load hex file
371             case I_ICSP:
372                 Window1.menu1_File_LoadHex_EvMenuActive (NULL);
373                 break;
374             //if event is over I_PWR area then toggle board on/off
375             case I_PWR:
376                 if (Window1.Get_mcupwr ()) //if on turn off
377                 {
378                     Window1.Set_mcurun (0);
379                     Window1.Set_mcupwr (0);
380                     Reset ();

```

```

381         p_BT1 = 1;
382         Window1.statusbar1.SetField (0, lxt ("Stoped"));
383     }
384     else //if off turn on
385     {
386         Window1.Set_mcupwr (1);
387         Window1.Set_mcurun (1);
388         Reset ();
389         Window1.statusbar1.SetField (0, lxt ("Running..."));
390     }
391     output_ids[O_LPWR]->update = 1;
392     break;
393     //if event is over I_RST area then turn off and reset
394     case I_RST:
395         if (Window1.Get_mcupwr () && pic_reset (-1))//if powered
396         {
397             Window1.Set_mcupwr (0);
398             Window1.Set_mcurst (1);
399         }
400         p_RST = 0;
401         output_ids[O_RST]->update = 1;
402         break;
403         //if event is over I_D0 area then activate button (state=0)
404     case I_D0:
405         p_BT1 = 0;
406         output_ids[O_BD0]->update = 1;
407         break;
408         //if event is over I_D1 area then toggle switch state
409     case I_D1:
410         p_BT2 ^= 1;
411         output_ids[O_SD1]->update = 1;
412         break;
413     case I_POT1:
414     {
415         active = 1;
416         pot1 = (x - input[i].x1)*2.77;
417         if (pot1 > 199) pot1 = 199;
418         output_ids[O_POT1]->update = 1;
419     }
420     break;
421   }
422 }
423 }
424 }
425 }
426 //Event on the board
427
428

```

```

429 void
430 cboard_x::EvMouseMove(uint button, uint x, uint y, uint state)
431 {
432     int i;
433
434     for (i = 0; i < inputc; i++)
435     {
436         switch (input[i].id)
437         {
438             case I_POT1:
439                 if (((input[i].x1 <= x)&&(input[i].x2 >= x))&&((input[i].y1 <= y)&&(input[i].y2 >= y)))
440                 {
441                     if (active)
442                     {
443                         pot1 = (x - input[i].x1)*2.77;
444                         if (pot1 > 199)pot1 = 199;
445                         output_ids[O_POT1]->update = 1;
446                     }
447                 }
448                 break;
449             }
450         }
451     }
452
453 //Event on the board
454
455 void
456 cboard_x::EvMouseButtonRelease(uint button, uint x, uint y, uint state)
457 {
458     int i;
459
460     //search for the input area which owner the event
461     for (i = 0; i < inputc; i++)
462     {
463         if (((input[i].x1 <= x)&&(input[i].x2 >= x))&&((input[i].y1 <= y)&&
464                                         (input[i].y2 >= y)))
465         {
466             switch (input[i].id)
467             {
468                 //if event is over I_RST area then turn on
469                 case I_RST:
470                     if (Window1.Get_mcurst ())//if powered
471                     {
472                         Window1.Set_mcupwr (1);
473                         Window1.Set_mcurst (0);
474
475                     if (pic_reset (-1))
476                     {

```

```
477             Reset ();
478         }
479     }
480     p_RST = 1;
481     output_ids[O_RST]->update = 1;
482     break;
483 //if event is over I_D0 area then deactivate button (state=1)
484 case I_D0:
485     p_BT1 = 1;
486     output_ids[O_BD0]->update = 1;
487     break;
488 case I_POT1:
489 {
490     active = 0;
491     output_ids[O_POT1]->update = 1;
492 }
493 break;
494 }
495 }
496 }
497
498 }
499
500
501 //Called every 100ms to draw board
502 //This is the critical code for simulator running speed
503
504 void
505 cboard_x::Draw(CDraw *draw)
506 {
507     int update = 0; //verifiy if updated is needed
508     int i;
509
510
511 //board_x draw
512 for (i = 0; i < outputc; i++) //run over all outputs
513 {
514     if (output[i].update)//only if need update
515     {
516         output[i].update = 0;
517
518         if (!update)
519         {
520             draw->Canvas.Init (Scale, Scale);
521         }
522         update++; //set to update buffer
523
524         if (!output[i].r)//if output shape is a rectangle
```

```

525     {
526     if (output[i].id == O_SD1)//if output is switch
527     {
528         //draw a background white rectangle
529         draw->Canvas.SetBgColor (255, 255, 255);
530         draw->Canvas.Rectangle (1, output[i].x1, output[i].y1,
531                             output[i].x2 - output[i].x1, output[i].y2 - output[i].y1);
532
533     if (!p_BT2) //draw switch off
534     {
535         //draw a grey rectangle
536         draw->Canvas.SetBgColor (70, 70, 70);
537         draw->Canvas.Rectangle (1, output[i].x1, output[i].y1 +
538                             ((int) ((output[i].y2 - output[i].y1)*0.35)), output[i].x2 - output[i].x1 +
539                             ((int) ((output[i].y2 - output[i].y1)*0.65)));
540     }
541     else //draw switch on
542     {
543         //draw a grey rectangle
544         draw->Canvas.SetBgColor (70, 70, 70);
545         draw->Canvas.Rectangle (1, output[i].x1,
546                             output[i].y1, output[i].x2 - output[i].x1,
547                             ((int) ((output[i].y2 - output[i].y1)*0.65)));
548     }
549     }
550     else if (output[i].id == O_BD0)
551     {
552         draw->CanvasSetColor (102, 102, 102);
553         draw->Canvas.Circle (1, output[i].cx, output[i].cy, 10);
554         if (p_BT1)
555         {
556             draw->CanvasSetColor (15, 15, 15);
557         }
558     else
559         {
560             draw->CanvasSetColor (55, 55, 55);
561         }
562         draw->Canvas.Circle (1, output[i].cx, output[i].cy, 8);
563     }
564     else if (output[i].id == O_RST)
565     {
566         draw->CanvasSetColor (102, 102, 102);
567         draw->Canvas.Circle (1, output[i].cx, output[i].cy, 10);
568
569         if (p_RST)
570         {
571             draw->CanvasSetColor (15, 15, 15);
572         }

```

```

573     else
574     {
575         draw->Canvas.SetColor (55, 55, 55);
576     }
577     draw->Canvas.Circle (1, output[i].cx, output[i].cy, 8);
578 }
579 else if (output[i].id == O_POT1)
580 {
581     draw->Canvas.SetColor (0, 50, 215);
582     draw->Canvas.Rectangle (1, output[i].x1, output[i].y1, output[i].x2 - output[i].x1, output[i].y2);
583     draw->Canvas.SetColor (250, 250, 250);
584     draw->Canvas.Rectangle (1, output[i].x1 + pot1 / 2.77, output[i].y1 + 2, 10, 15);
585 }
586 else if (output[i].id == O_CPU)
587 {
588     draw->CanvasSetFont (font);
589     int x, y, w, h;
590     draw->CanvasSetColor (26, 26, 26);
591     draw->Canvas.Rectangle (1, output[i].x1, output[i].y1, output[i].x2 - output[i].x1, output[i].y2);
592
593     draw->CanvasSetColor (230, 230, 230);
594     w = output[i].x2 - output[i].x1;
595     h = output[i].y2 - output[i].y1;
596     x = output[i].x1 + (w / 2) + 7;
597     y = output[i].y1 + (h / 2) + (Proc.length ());
598     draw->Canvas.RotatedText (Proc, x, y, 270);
599 }
600 }
601 else //if output shape is a circle
602 {
603     draw->Canvas.SetFgColor (0, 0, 0); //black
604
605     switch (output[i].id)//search for color of output
606     {
607         case O_LD0: //White using pin 19 mean value (RD0)
608             draw->Canvas.SetBgColor (pic.pins[18].oavalue, pic.pins[18].oavalue, pic.pins[18].oavalue);
609             break;
610         case O_LD1: //Yellow using pin 20 mean value (RD1)
611             draw->Canvas.SetBgColor (pic.pins[19].oavalue, pic.pins[19].oavalue, 0);
612             break;
613         case O_LPWR: //Blue using mcupwr value
614             draw->Canvas.SetBgColor (0, 0, 200 * Window1.Get_mcupwr () + 55);
615             break;
616         case O_RB0: //Green using pin 33 mean value (RB0)
617             draw->Canvas.SetBgColor (0, pic.pins[32].oavalue, 0);
618             break;
619         case O_RB1: //Red using pin 34 mean value (RB1)
620             draw->Canvas.SetBgColor (pic.pins[33].oavalue, 0, 0);

```

```
621         break;
622     }
623
624     //draw a LED
625     color1 = draw->Canvas.GetBgColor ();
626     int r = color1.Red () - 120;
627     int g = color1.Green () - 120;
628     int b = color1.Blue () - 120;
629     if (r < 0)r = 0;
630     if (g < 0)g = 0;
631     if (b < 0)b = 0;
632     color2.Set (r, g, b);
633     draw->Canvas.SetBgColor (color2);
634     draw->Canvas.Circle (1, output[i].x1, output[i].y1, output[i].r + 1);
635     draw->Canvas.SetBgColor (color1);
636     draw->Canvas.Circle (1, output[i].x1, output[i].y1, output[i].r - 2);
637   }
638 }
639 }
640 //end draw
641
642 if (update)
643 {
644   draw->Canvas.End ();
645   draw->Update ();
646 }
647
648 //RB0 mean value to gauge1
649 gauge1->SetValue ((pic.pins[33].oavalue - 55) / 2);
650 //RB1 mean value to gauge2
651 gauge2->SetValue ((pic.pins[32].oavalue - 55) / 2);
652
653 }
654
655 void
656 cboard_x::Run_CPU(void)
657 {
658   int i;
659   int j;
660   unsigned char pi;
661   const picpin * pins;
662   unsigned int alm[40];
663
664   int JUMPSTEPS = Window1.GetJUMPSTEPS (); //number of steps skipped
665   long int NSTEP = Window1.GetNSTEP () / MGetPinCount (); //number of steps in 100ms
666
667
668 //reset pins mean value
```

```
669     memset (alm, 0, 40 * sizeof (unsigned int));
670
671 //read pic.pins to a local variable to speed up
672 pins = pic.pins;
673
674 //Spare parts window pre process
675 if (use_spare)Window5.PreProcess ();
676
677 j = JUMPSTEPS; //step counter
678 if (Window1.Get_mcupwr ()) //if powered
679     for (i = 0; i < Window1.GetNSTEP (); i++) //repeat for number of steps in 100ms
680     {
681
682         if (j >= JUMPSTEPS)//if number of step is bigger than steps to skip
683         {
684             pic_set_pin (pic.mclr, p_RST);
685             pic_set_pin (19, p_BT1); //Set pin 19 (RD0) with button state
686             pic_set_pin (20, p_BT2); //Set pin 20 (RD1) with switch state
687         }
688
689         //verify if a breakpoint is reached if not run one instruction
690         if (!mplabxd_testbp ())pic_step ();
691         //Oscilloscope window process
692         if (use_oscope)Window4.SetSample ();
693         //Spare parts window process
694         if (use_spare)Window5.Process ();
695
696         //increment mean value counter if pin is high
697         alm[i % pic.PINCOUNT] += pins[i % pic.PINCOUNT].value;
698
699         if (j >= JUMPSTEPS)//if number of step is bigger than steps to skip
700         {
701
702             //set analog pin 2 (AN0) with value from scroll
703             pic_set_apin (2, (5.0 * pot1 / 199));
704
705             j = -1; //reset counter
706         }
707         j++; //counter increment
708     }
709
710     //calculate mean value
711     for (pi = 0; pi < pic.PINCOUNT; pi++)
712     {
713         pic.pins[pi].oavalue = (int) (((200.0 * alm[pi]) / NSTEP) + 55);
714     }
715
716 //Spare parts window pre post process
```

```

717     if (use_spare) Window5.PostProcess ();
718
719     //verifiy if LEDS need update
720     if (output_ids[O_LD0]->value != pic.pins[18].oavalue)
721     {
722         output_ids[O_LD0]->value = pic.pins[18].oavalue;
723         output_ids[O_LD0]->update = 1;
724     }
725     if (output_ids[O_LD1]->value != pic.pins[19].oavalue)
726     {
727         output_ids[O_LD1]->value = pic.pins[19].oavalue;
728         output_ids[O_LD1]->update = 1;
729     }
730     if (output_ids[O_RB0]->value != pic.pins[32].oavalue)
731     {
732         output_ids[O_RB0]->value = pic.pins[32].oavalue;
733         output_ids[O_RB0]->update = 1;
734     }
735     if (output_ids[O_RB1]->value != pic.pins[33].oavalue)
736     {
737         output_ids[O_RB1]->value = pic.pins[33].oavalue;
738         output_ids[O_RB1]->update = 1;
739     }
740 }
741
742
743
744 //Register the board in PICSimLab
745 board_init(BOARD_x_Name, cboard_x);

```

13.2.5 Integration with PICsimLab

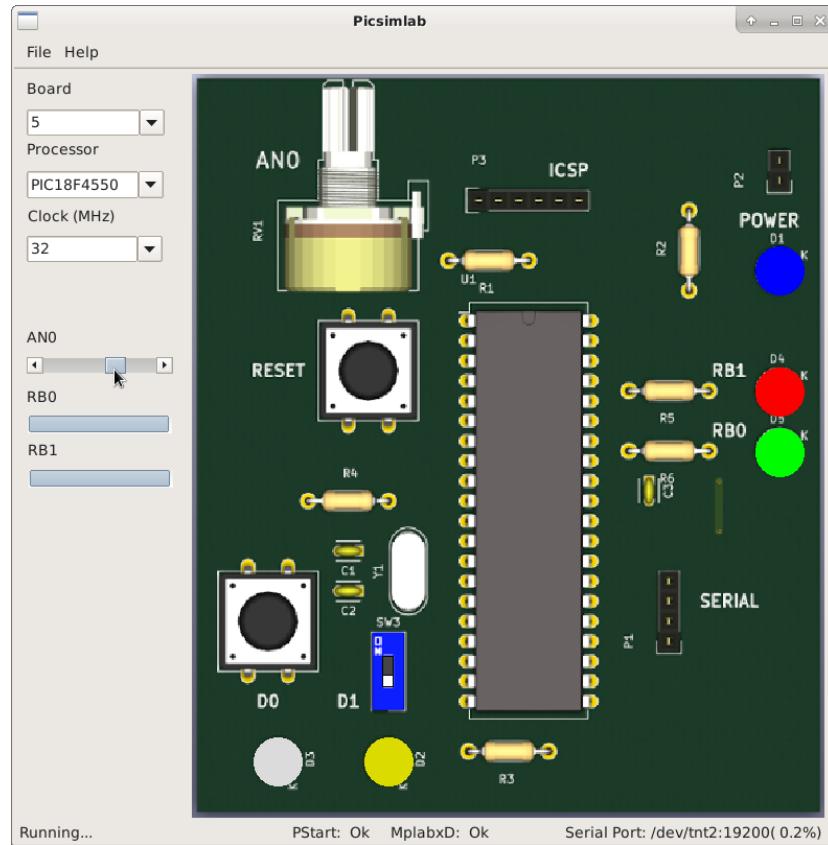
To integration of the new board in PICSimLab, are necessary edit one file.

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         serial_tx_str(itoa(val,buffer));
55         serial_tx_str("\r\n");
56     }
57
58
59 }
```

Chapter 14

License

Copyright © 2021 Luis Claudio Gambôa Lopes <lcgambboa@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.