plcLib Quick Start

Simulation Area

The *Simulation area* lets you run the sketch which is currently loaded in the *Simulator Code editor* area. It implements a simple input/output model with four digital inputs (*X0-X3*), two analogue inputs (*AD0-AD1*) and four digital outputs (*Y0-Y3*). The use of 'generic' pin names and values is called *hardware abstraction*. The currently selected hardware is shown in the centre, which will matter later, when you download your sketch. Inputs and outputs are colour-coded based on value. For clarity, actual numeric values are written next to them.

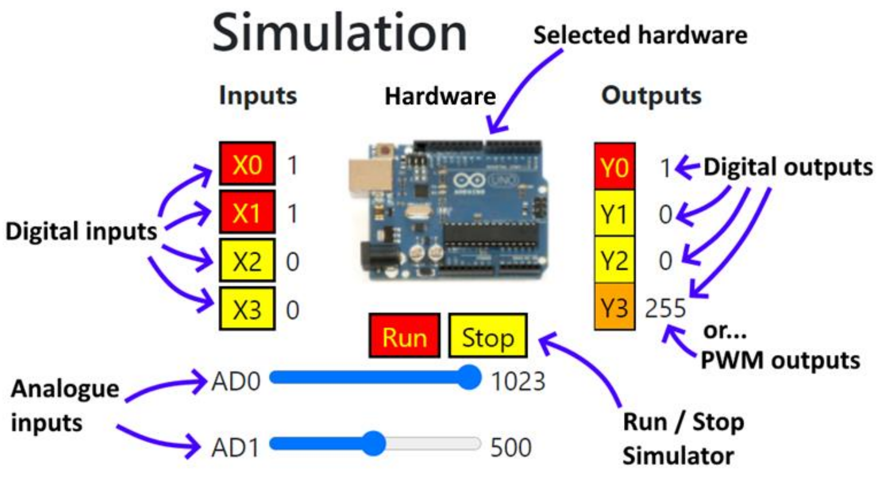
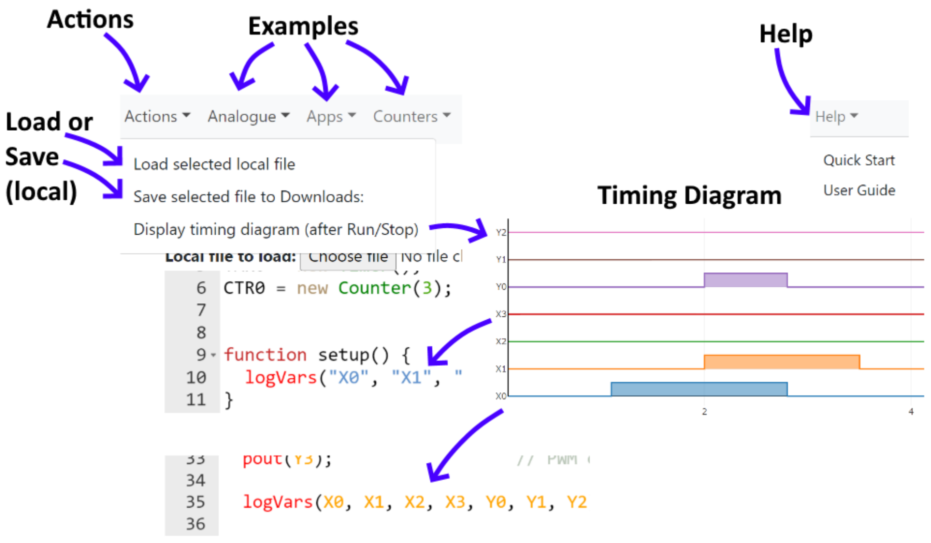


Figure 1. The Simulation Area.

Files and Actions

The pull-down menu allows you to load and test a range of pre-created examples, which are grouped by category. Some areas (such as *Apps*) have a mixture of basic and more advanced examples. User documentation is available from the *Help* area at the right. The *Actions* area has options to load or save your work to the local file system, which works with file selection options at the top of the *Simulator Code* text area. You can also generate *timing diagrams*, by using (both of) the *LogVars* commands below. (Try removing comments from the sample sketch to test this.)



```
CTR0 = new Counter(3);


function setup() {
  logVars("X0", "X1", "
}
```

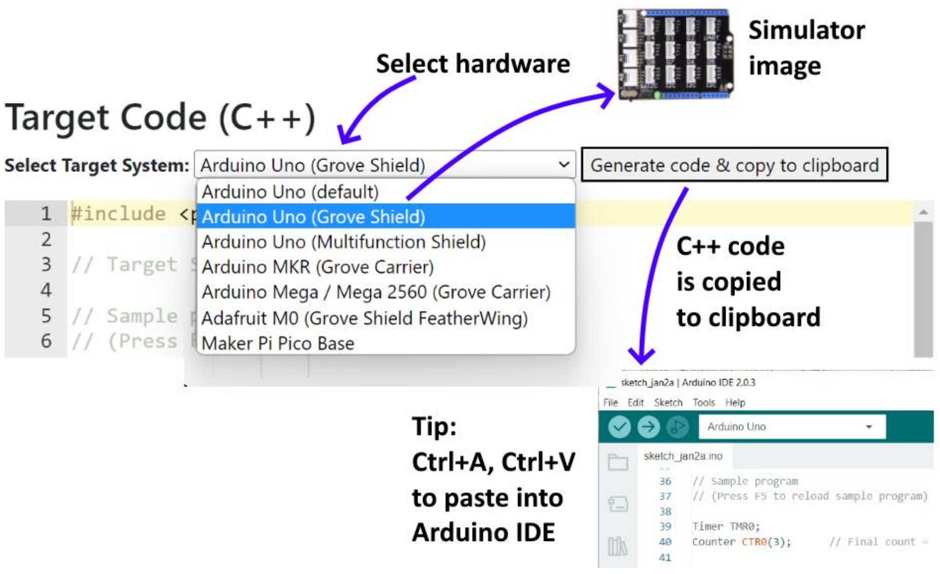
```
  pout(Y3);                    // PWM (

  logVars(X0, X1, X2, X3, Y0, Y1, Y2

```

Figure 2. Actions, examples and help are available from the pull-down menu.

Sending to the Target System

Having previously loaded and tested your sketch, the next step is to ‘convert’ it and then download it to your chosen hardware. The sketch in the *Simulator Code text editor* area is actually a JavaScript program, which is simulated with the aid of a JavaScript version of the plclib library, all of which runs in the browser. This JavaScript sketch needs to be first converted to C++ (although the differences are quite minor), before being downloaded. In addition, the generic I/O pins used earlier, will need to be converted to actual ones, to suit your hardware.

The desired hardware platform should first be selected from the drop-down menu shown below. (This also updates the hardware image in the *Simulation* area.) Next, click the button below to generate the equivalent C++ code in the *Target Code text area*, and at the same time copy the sketch to the clipboard. Finally, switch to the Arduino IDE, replace the existing sketch with the one stored in the clipboard, then compile and download your sketch to the target system.



Select hardware
Simulator image
Target Code (C++)
Select Target System:
Arduino Uno (Grove Shield)
Arduino Uno (default)
Arduino Uno (Grove Shield)
Arduino Uno (Multifunction Shield)
Arduino MKR (Grove Carrier)
Arduino Mega / Mega 2560 (Grove Carrier)
Adafruit M0 (Grove Shield FeatherWing)
Maker Pi Pico Base
Generate code & copy to clipboard
C++ code is copied to clipboard
Tip: Ctrl+A, Ctrl+V to paste into Arduino IDE
sketch_jan2a | Arduino IDE 2.0.3
File Edit Sketch Tools Help
Arduino Uno
sketch_jan2a.ino

```
#include <p

// Target S

// Sample p
// (Press F
```

```
// Sample program
// (Press F5 to reload sample program)

Timer TMR0;
Counter CTR0(3);      // Final count =
```

Figure 3. *Selecting the desired target system and copying code to the Arduino IDE.*

Fault-Finding Tips

1. If your sketch won’t download from the Arduino IDE to the connected target system, check that you have selected the correct *board* and *port* in the Arduino IDE.
2. If it won’t compile, check that you have installed the plcLib library (and any other libraries used by the sketch) and that they are correctly configured. (Check that the library appears under *File* > *Examples*. You could also try running an example sketch from the library.)
3. Start with the simplest possible test sketch (IO > BareMinimum), and check this works correctly on your hardware, before trying anything more complex.
4. If it runs on the target system but results are different from those seen in the Simulator, then first check that you have connected any peripherals to the same pins identified in the sketch. Also check that positive and negative logic input/output settings are correctly configured in the *plcLib.cpp* file, for the chosen hardware. (See the *User Guide* for more details.)
5. Some JavaScript example sketches have built-in debugging instructions, such as `console.log` commands. (If so, the associated filename will typically contain the phrase ‘debug’.) These Javascript debugging commands are automatically converted to equivalent C++ `Serial.println` commands, as part of the ‘conversion’ process. A `Serial.begin` command is also placed into the *setup* function, which will initialise the serial port to 9600 Baud. In this case, you can use the *Serial Monitor* feature to debug your sketch from within the Arduino IDE.