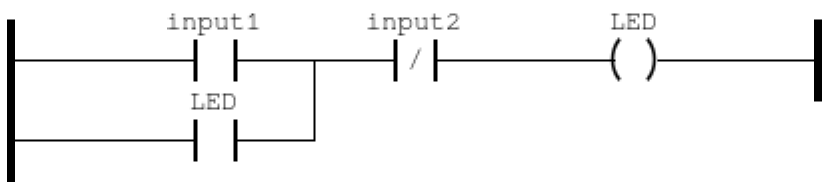
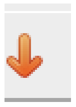
LAB 4

Matthew Townsend
3/7/24

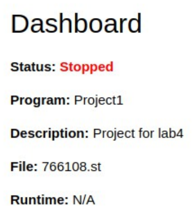
In this lab we used the ladder logic we made in Lab 2 and applied it to a physical circuit. To do this we used the openplc runtime at localhost:8080 to process the circuit and give it power. The logic we used to create the powered circuit was this. Two buttons and the light with a power pass through for the led if it on.



First, we have to compile the program to be used in the runtime through pressing the orange arow



The openplc runtime site gives us a dashboard to monitor and add new circuits to it. This allows us to verify it is working the way we intend it to.



Dashboard

Status: Stopped

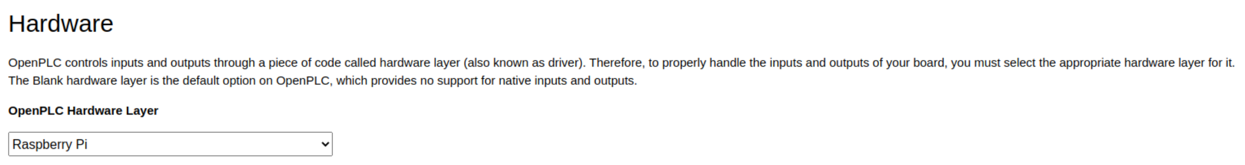
Program: Project1

Description: Project for lab4

File: 766108.st

Runtime: N/A

First, we need to select the hardware that is being used, for us we are using a raspberry. This will use the proper pin layout, and which is needed to have the system function properly.



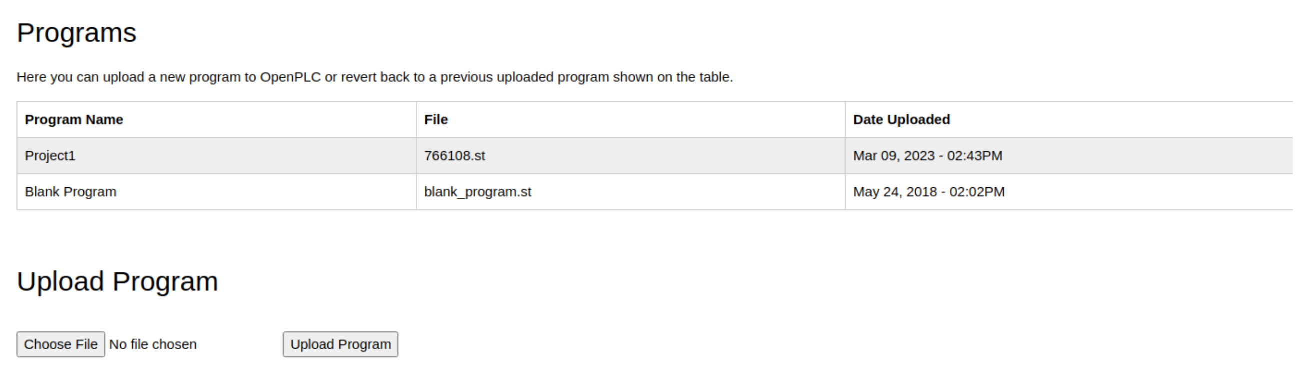
Hardware

OpenPLC controls inputs and outputs through a piece of code called hardware layer (also known as driver). Therefore, to properly handle the inputs and outputs of your board, you must select the appropriate hardware layer for it. The Blank hardware layer is the default option on OpenPLC, which provides no support for native inputs and outputs.

OpenPLC Hardware Layer

Raspberry Pi

A long the side of the site there is a menu which will allow us to do those functions and the programs tab will allow us to upload the program file to the software to run



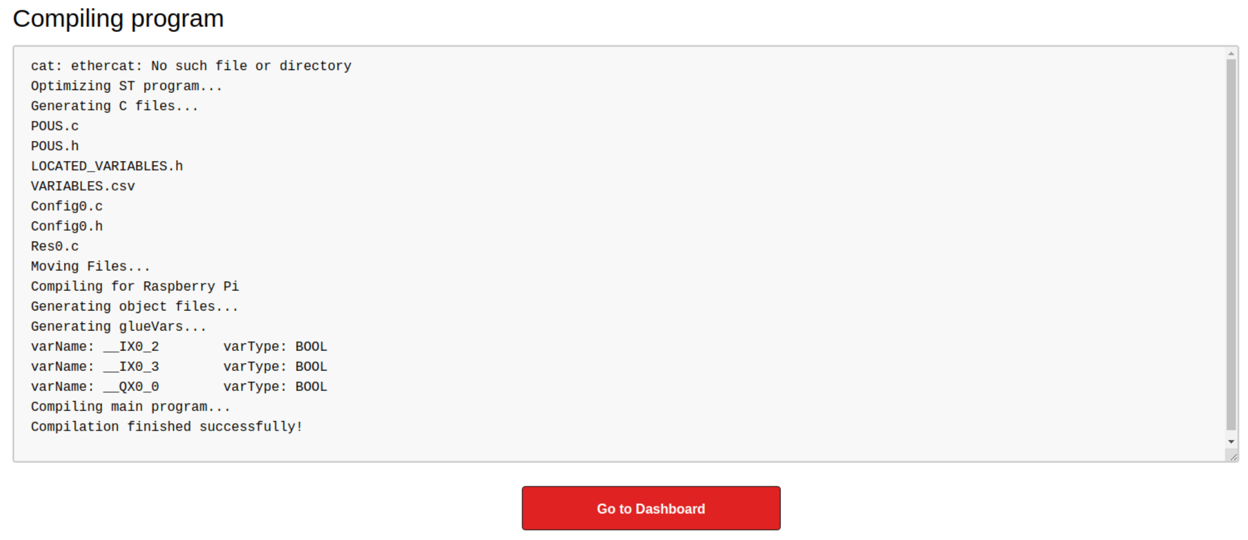
Programs

Here you can upload a new program to OpenPLC or revert back to a previous uploaded program shown on the table.

Program Name	File	Date Uploaded
Project1	766108.st	Mar 09, 2023 - 02:43PM
Blank Program	blank_program.st	May 24, 2018 - 02:02PM

Upload Program

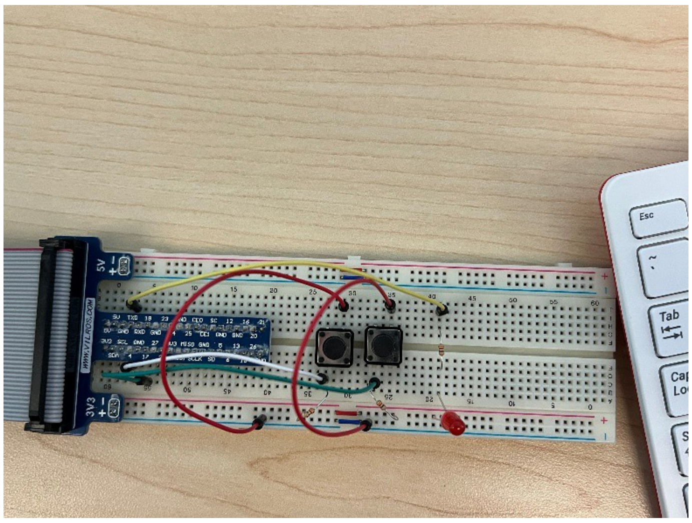
Press upload and then this will appear and wait for it to finish



Compiling program

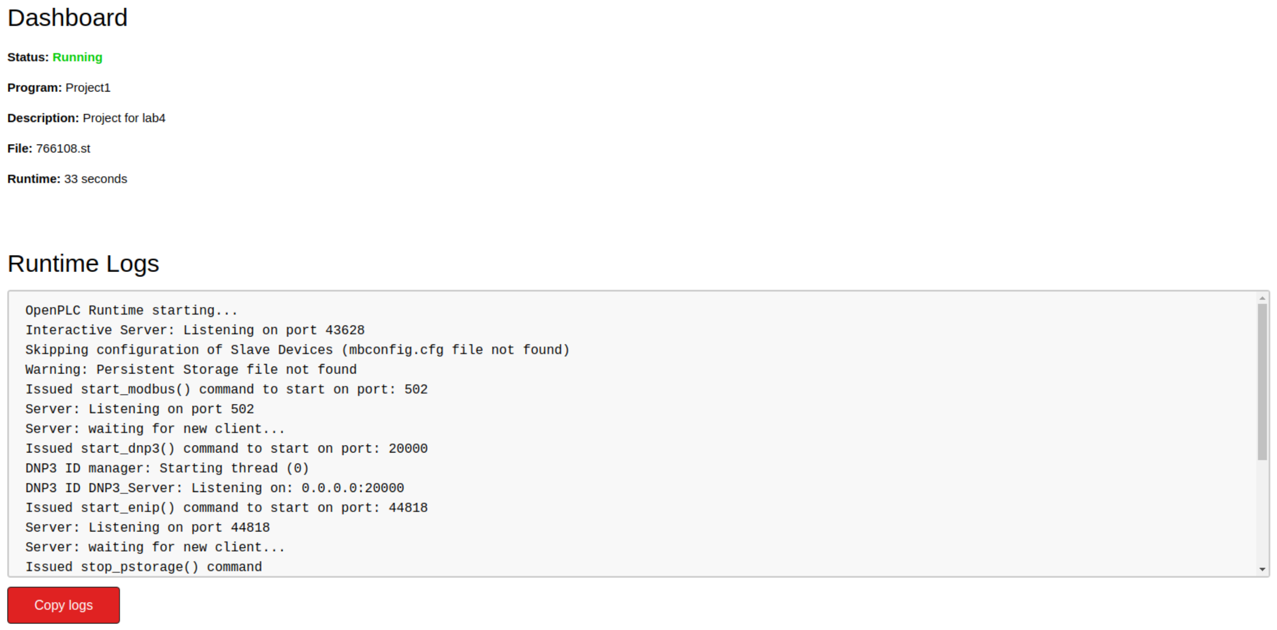
```
cat: ethercat: No such file or directory
Optimizing ST program...
Generating C files...
POUS.c
POUS.h
LOCATED_VARIABLES.h
VARIABLES.csv
Config0.c
Config0.h
Res0.c
Moving Files...
Compiling for Raspberry Pi
Generating object files...
Generating glueVars...
varName: __IX0_2        varType: BOOL
varName: __IX0_3        varType: BOOL
varName: __QX0_0        varType: BOOL
Compiling main program...
Compilation finished successfully!
```

Next comes building the physical circuit itself with the bread board, breakout board, wires, buttons and lights



After the circuit is built should look like this in the end, wires from 4 and 17 to the two buttons and a wire from TXD to the resistor in line with the LED, two red cables giving it power, and two more resistors grounding it. Then we shut down the pi and plug in the ribbon cable, one end in the breakout board and the other to the pi itself. Then power it back online.

Next, we return to localhost:8080 where we can view the dashboard and finally press run on the program hopefully running it. To verify it started we can view the monitor as well as pressing button 1 to turn the LED on.



Dashboard

Status: Running

Program: Project1

Description: Project for lab4

File: 766108.st

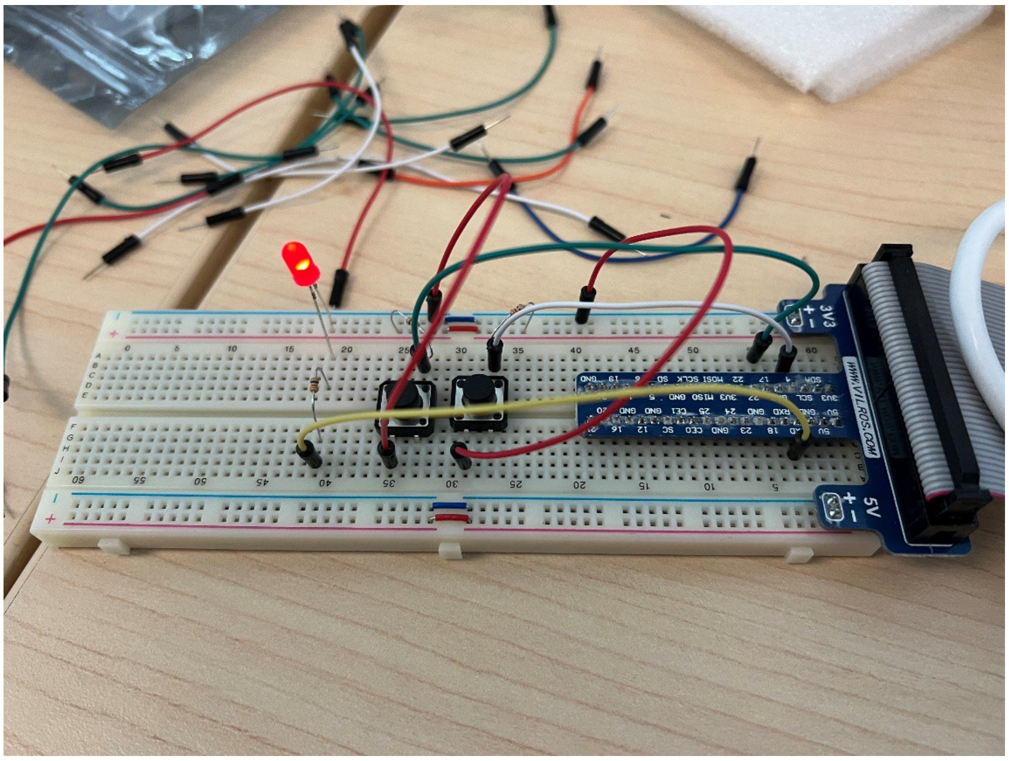
Runtime: 33 seconds

Runtime Logs

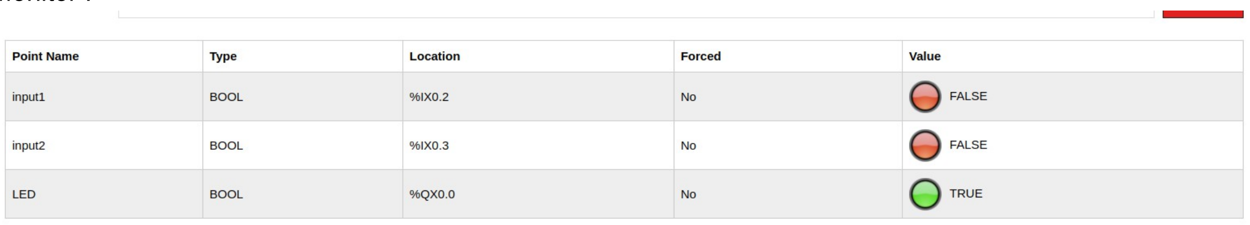
```
OpenPLC Runtime starting...
Interactive Server: Listening on port 43628
Skipping configuration of Slave Devices (mbconfig.cfg file not found)
Warning: Persistent Storage file not found
Issued start_modbus() command to start on port: 502
Server: Listening on port 502
Server: waiting for new client...
Issued start_dnp3() command to start on port: 20000
DNP3 ID manager: Starting thread (0)
DNP3 ID DNP3_Server: Listening on: 0.0.0.0:20000
Issued start_enip() command to start on port: 44818
Server: Listening on port 44818
Server: waiting for new client...
Issued stop_pstorage() command
```

To verify its proper functionality, we can use the monitor tab to check this

Button one:

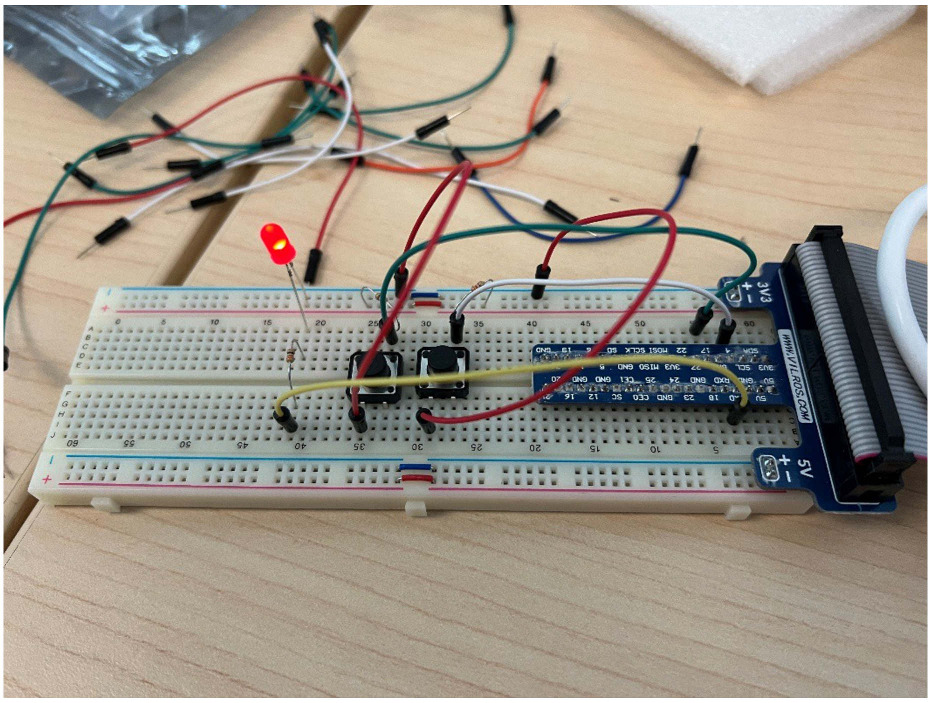


Monitor :

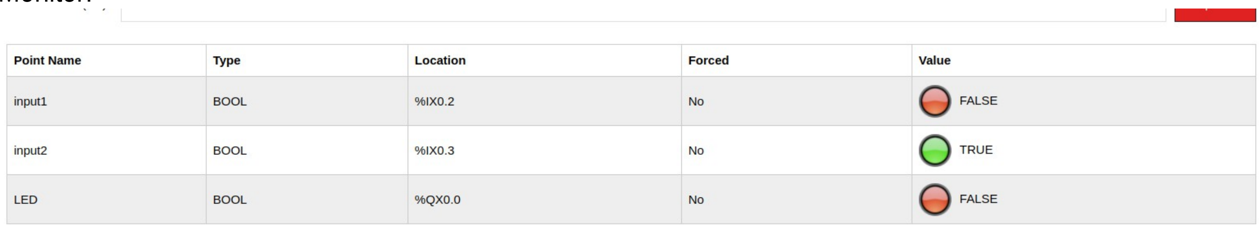


Point Name	Type	Location	Forced	Value
input1	BOOL	%IX0.2	No	🔴 FALSE
input2	BOOL	%IX0.3	No	🔴 FALSE
LED	BOOL	%QX0.0	No	🟢 TRUE

BUTTON 2:

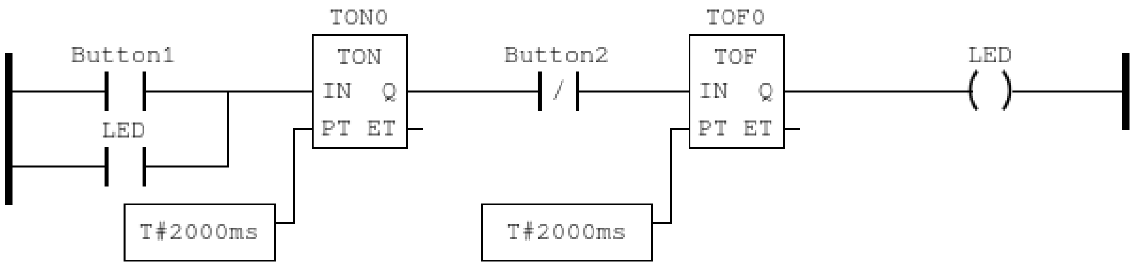


Monitor:

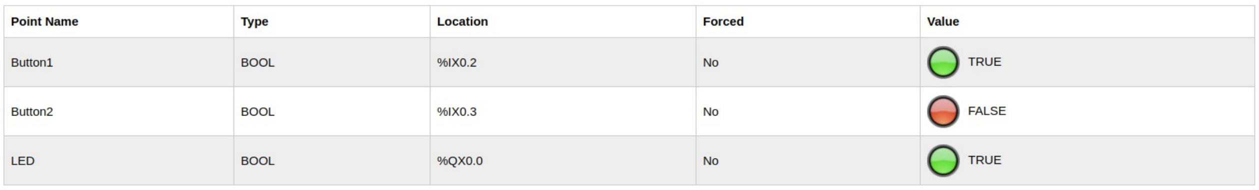


Point Name	Type	Location	Forced	Value
input1	BOOL	%IX0.2	No	🔴 FALSE
input2	BOOL	%IX0.3	No	🟢 TRUE
LED	BOOL	%QX0.0	No	🔴 FALSE

After the completion of the standard on/off circuit we are instructed to add a delay on and delay off function blocks to the program. This will delay the led from turning on and off when properly implemented such as here:

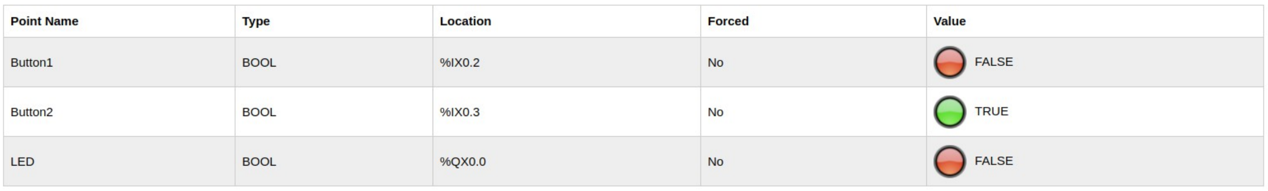


Here the buttons function the same as the previous program but when button one is pressed there is a 2 second delay to turn on the led which means you must hold the button for two seconds before the light turns on,



Point Name	Type	Location	Forced	Value
Button1	BOOL	%IX0.2	No	🟢 TRUE
Button2	BOOL	%IX0.3	No	🔴 FALSE
LED	BOOL	%QX0.0	No	🟢 TRUE

On the monitor page it will look like this, the button is pressed then the light turns true after the short delay. The functionality carries over to Turing it off with the TOF0 function block with the same delay of 2 seconds. Button 2 must be held for two seconds to turn off the light.



Point Name	Type	Location	Forced	Value
Button1	BOOL	%IX0.2	No	🔴 FALSE
Button2	BOOL	%IX0.3	No	🟢 TRUE
LED	BOOL	%QX0.0	No	🔴 FALSE

After being held the led will turn negative, as well as negating the led contact below button 1 in the program.
The design of the physical circuit did not change nor did the led so it will look the same as the previous program while using the new program.