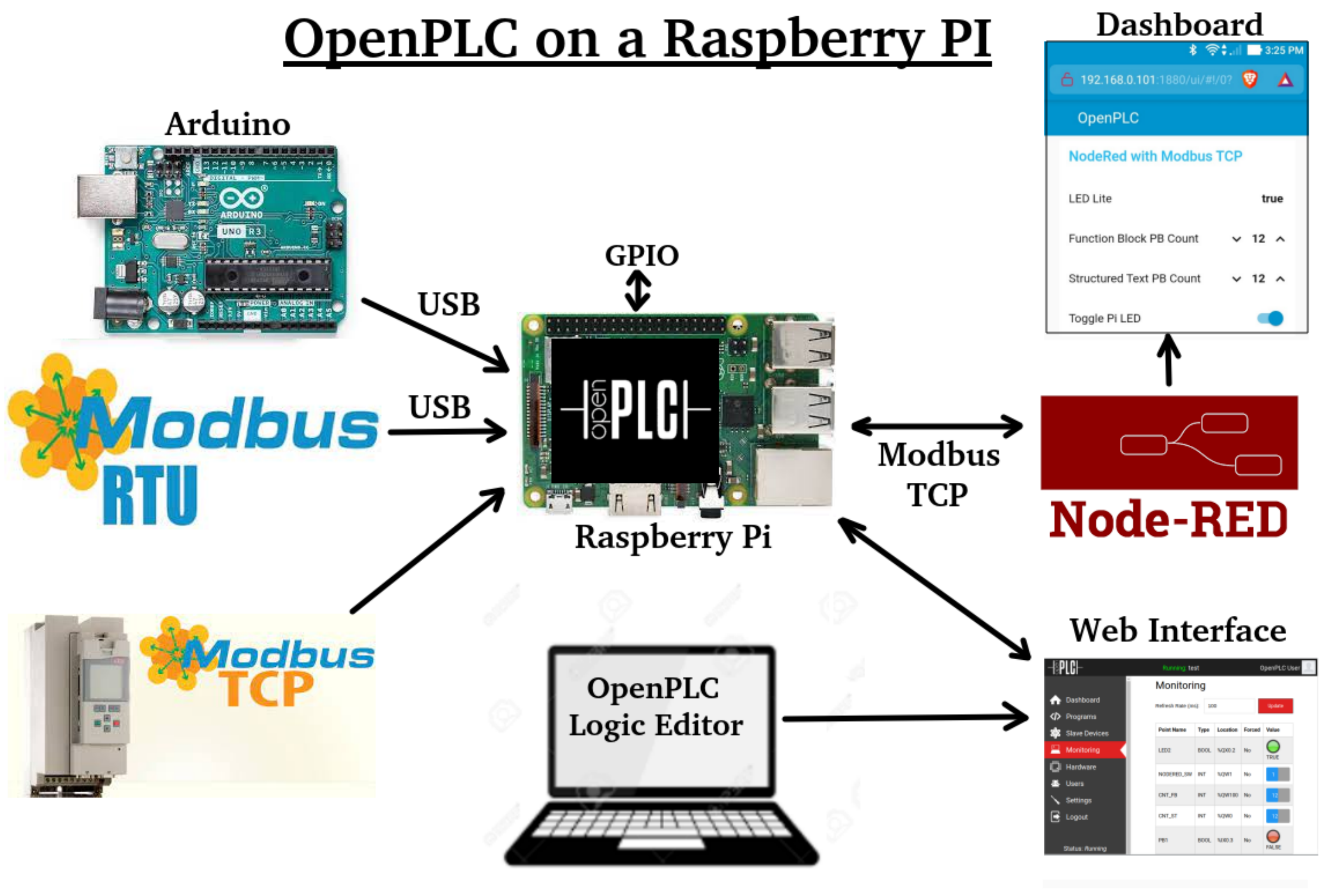
Fun Tech Projects

# OpenPLC on a Raspberry Pi



For home automation projects there are a lot of good software packages like Home Assistant and Node Red that can be used to control and view sensors and devices.

If you are interested in looking at an industrial controls approach to your automation projects then OpenPLC is a good package to consider.

A PLC (Programmable Logic Controller) is an industrial hardened hardware device that manages I/O and logic using the IEC 61131-3 standard.

OpenPLC (https://www.openplcproject.com/) is open source software that runs on a Raspberry Pi, Linux or Windows PC and it offers users a great way to learn industrial control concepts, programming languages and communications protocols.

In this article I will create three small Raspberry Pi projects using the IEC 61131-3 ladder logic, function blocks and structure text programming languages. Finally I will have these projects pass their data via Modbus TCP to a Node Red dashboard.

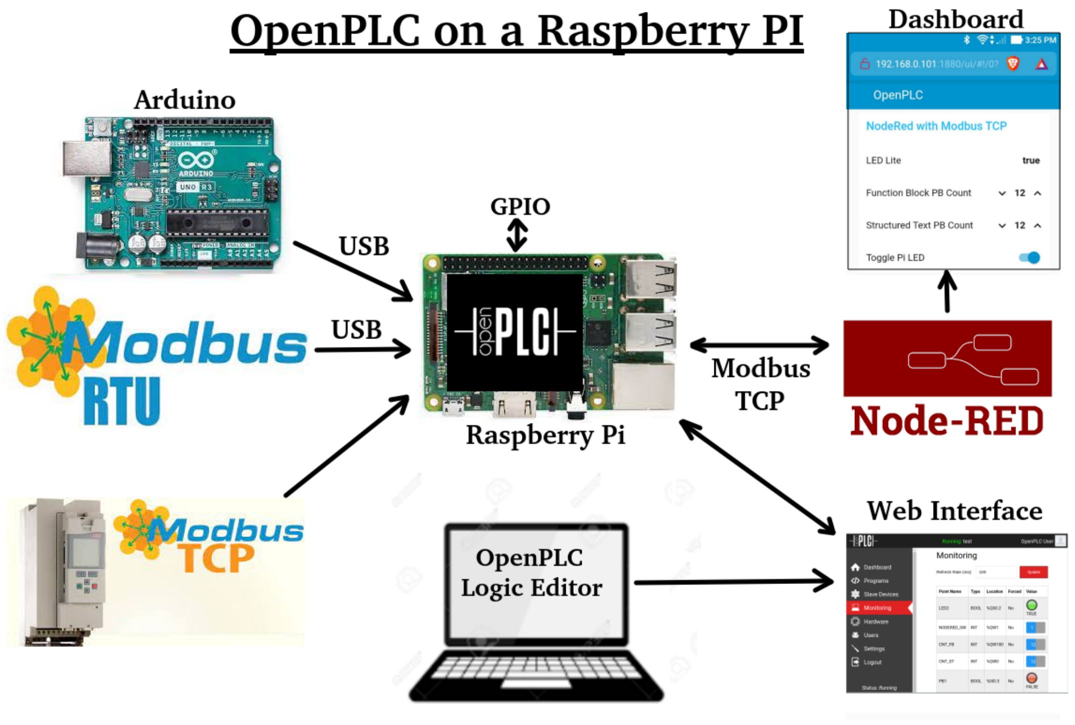
## Getting Started

The OpenPLC software comes in three packages, a logic editor, the runtime component, and a graphic builder. See https://www.openplcproject.com/getting-started/ (https://www.openplcproject.com/getting-started/) for specific instructions for your installation.

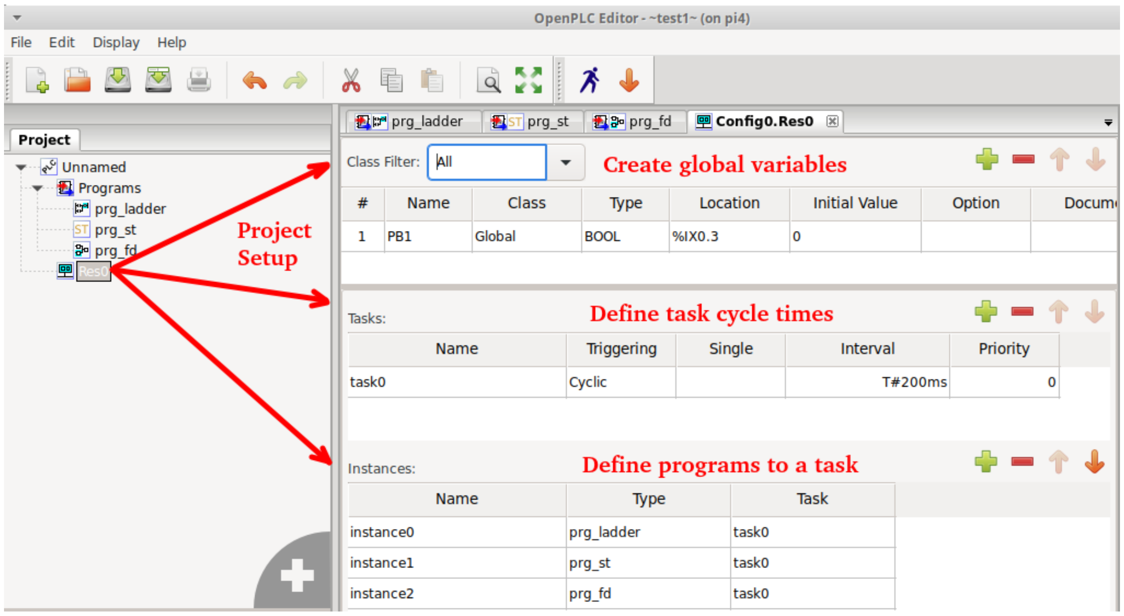
For my installation I put the OpenPLC editor on my Ubuntu PC so that I could do remote configuration. I loaded the OpenPLC runtime on a Raspberry PI. The OpenPLC runtime web interface is used to load and monitor logic.

I didn't install the OpenPLC graphic builder instead I used Node-Red Dashboards as my final user interface

OpenPLC has a good number of optional communications packages and slave I/O components. A typical layout could be as below.



For my application I created a project with three programs; a ladder program, a function block program and a structure text program. The Resource object (*Res0*) defines global variables that can be used by all programs, and the task cycle times. This is a small project so I put all the programs into the same task execution (*task0*). For larger project I might put all my digital logic into a fast task execution (20ms) and my analog logic into a slower task execution (250ms).



Class Filter: All

| # | Name | Class | Type | Location | Initial Value | Option | Docum |
|---|---|---|---|---|---|---|---|
| 1 | PB1 | Global | BOOL | %IX0.3 | 0 | | |

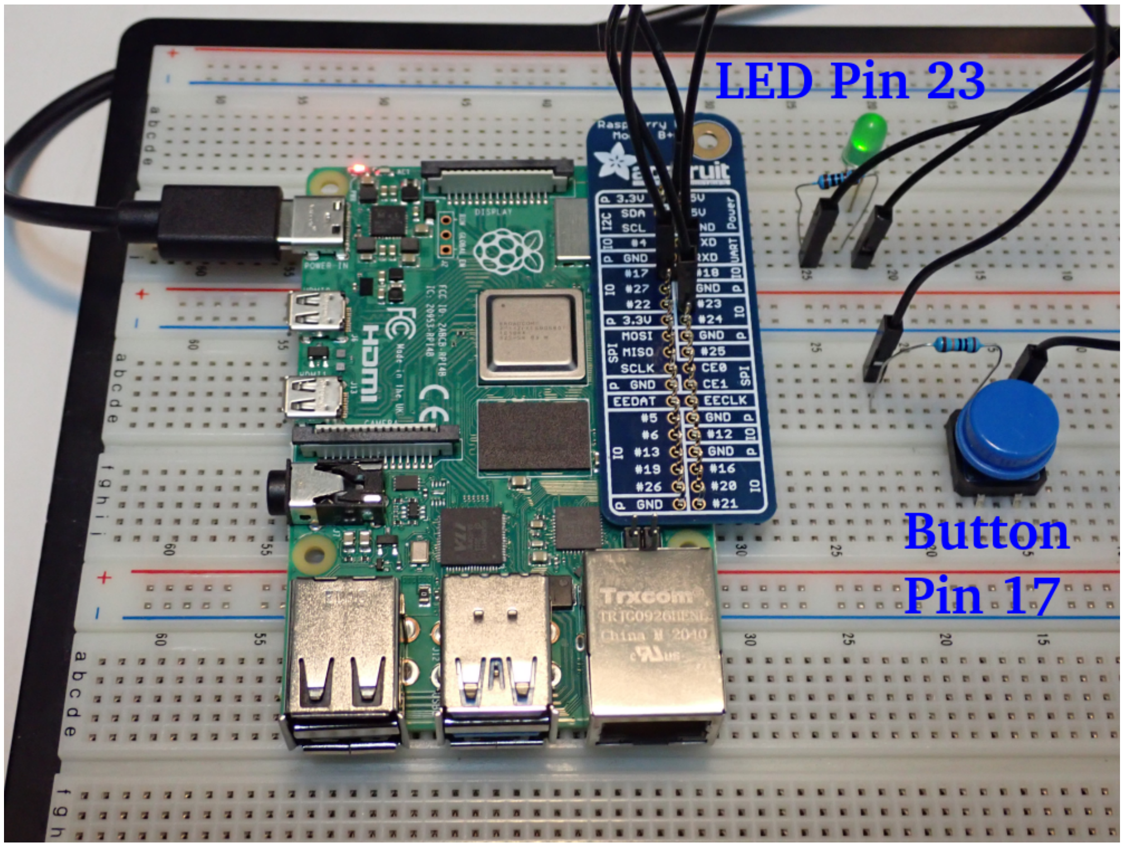
Tasks:

| Name | Triggering | Single | Interval | Priority |
|---|---|---|---|---|
| task0 | Cyclic | | T#200ms | 0 |

Instances:

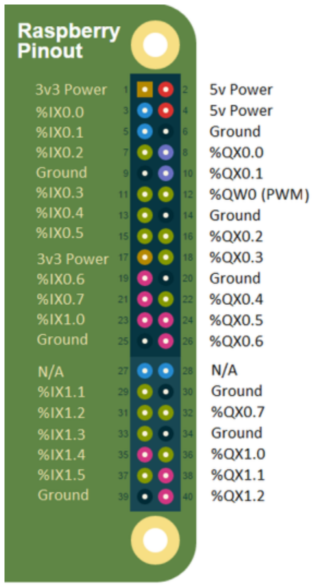
| Name | Type | Task |
|---|---|---|
| instance0 | prg_ladder | task0 |
| instance1 | prg_st | task0 |
| instance2 | prg_fd | task0 |

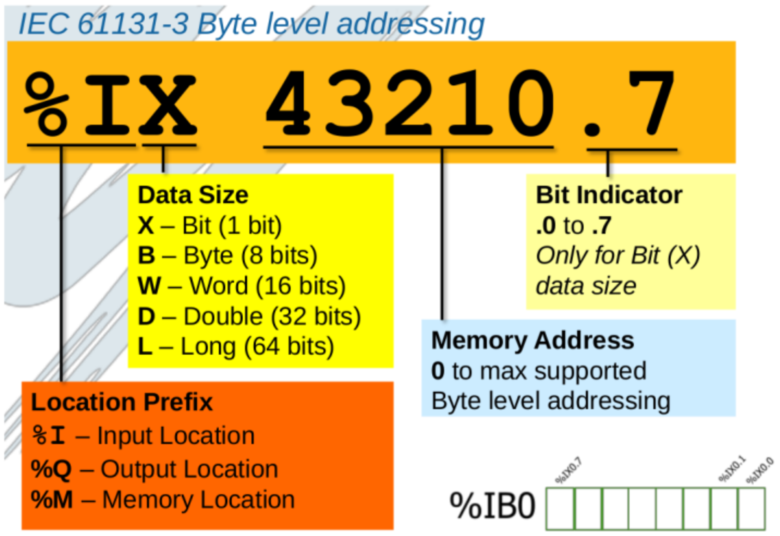
I setup the Raspberry Pi with a push button on pin 17 and an LED on pin 23.



On the Raspberry Pi GPIO pins are referenced using the IEC 61131-3 addressing. So the pushbutton at BCM pin 17 (physical pin 11) is addressed by %IX0.3 , an input bit on bus 0 at bit 3. The LED at BCM pin 23 (physical pin 16) is addressed by %QX0.2 , as output bit on bus 0 bit 2.

It’s important to note that OpenPLC has allocated all the left side (odd) pins as inputs and all the right side (even) pins as outputs.

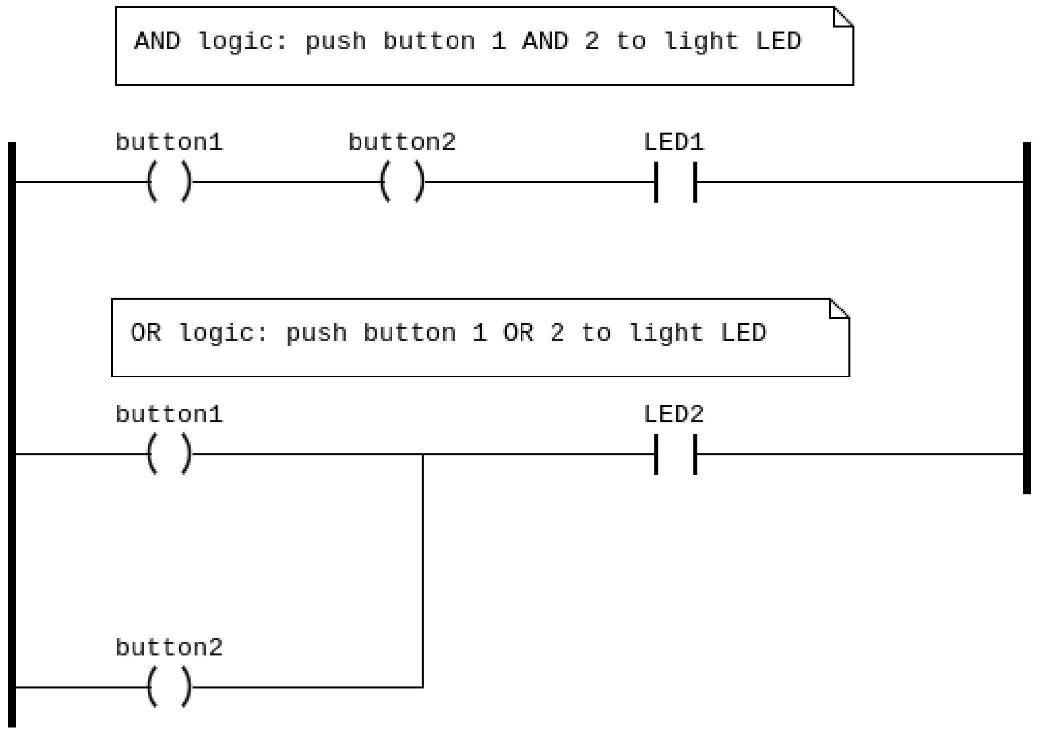




## Ladder Diagrams (LD)

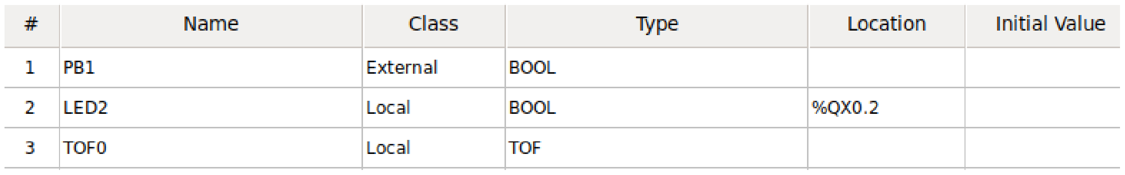
Ladder logic was the first IEC 61131-3 programming languages, it was developed as a graphic representation for circuit diagrams of relay logic hardware. The term "ladder" comes from the fact that the logic looks a little like a ladder with the left side having a vertical power rail and a a vertical ground rail on the right side, then there are a series of horizontal lines or "rungs" wiring hardware components between the rails.

Most electricians feel very comfortable using Ladder logic and it is a good programming method for managing digital logic. If you come from a programming background Ladder logic may feel a little strange at first, for example to do AND / OR logic to light an LED would be:

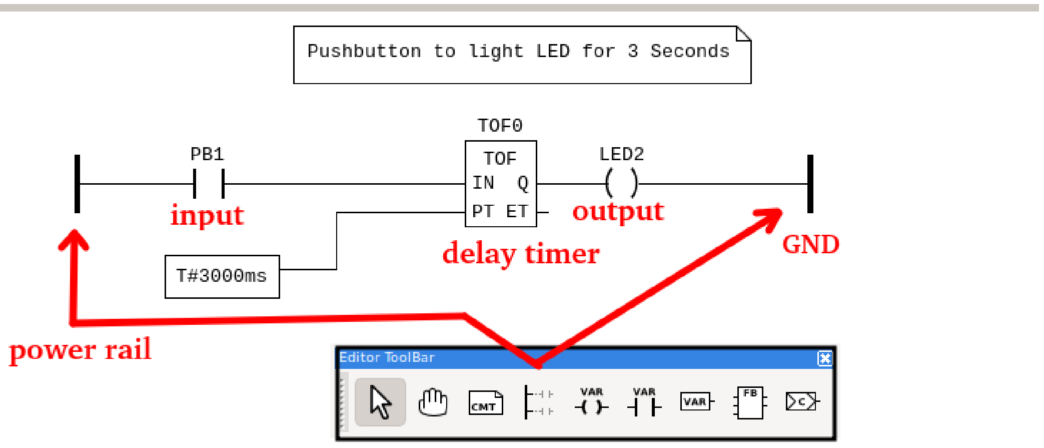

AND logic: push button 1 AND 2 to light LED
button1
button2
LED1
OR logic: push button 1 OR 2 to light LED
button1
LED2
button2

For my Ladder program I wanted to light an LED for 3 seconds with a single push of a button. In the OpenPLC editor I referenced an external variable PB1 (it's defined in Resource object *Res0*) and I created two local variables, LED2, my output LED and TOF0, an off delay timer.

IEC 61131-3 has a wide range functions that can be used in Ladder rungs. In this example a TOF function was inserted after the push button, and the time parameter is wired in variable.



| # | Name | Class | Type | Location | Initial Value |
|---|---|---|---|---|---|
| 1 | PB1 | External | BOOL | | |
| 2 | LED2 | Local | BOOL | %QX0.2 | |
| 3 | TOF0 | Local | TOF | | |



## Function Block Diagrams (FBD)

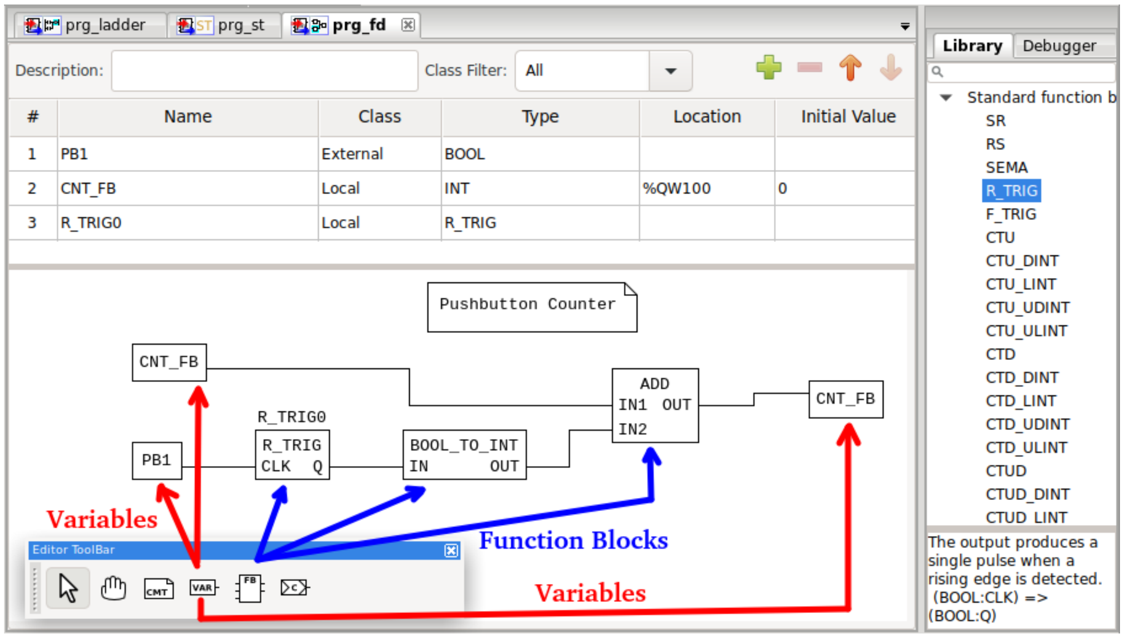
One of limitations of Ladder logic is that managing analog logic can be a little messy, for this reason Function Block Diagrams (FBD) was developed.

If you feel comfortable using graphic programming applications like Node-Red then you shouldn't have any problems working in Function Block Diagrams.

For my FBD program I wanted to count the number of times the LED was lite and output the value to a Modbus hold register.

Like in the Ladder program the external PB1 variable is referenced. A new output CNT_FB is defined as an output word on bus 100, %QW100.

The FBD uses a Rising Edge Trigger (R_TRIG) to catch when the LED turns on. The output from R_TRIG is a boolean so the value is converted to an INT and added to the value of CNT_FB.



| # | Name | Class | Type | Location | Initial Value |
|---|---|---|---|---|---|
| 1 | PB1 | External | BOOL | | |
| 2 | CNT_FB | Local | INT | %QW100 | 0 |
| 3 | R_TRIG0 | Local | R_TRIG | | |

The output produces a single pulse when a rising edge is detected. (BOOL:CLK) => (BOOL:Q)

## Structured Text (ST)

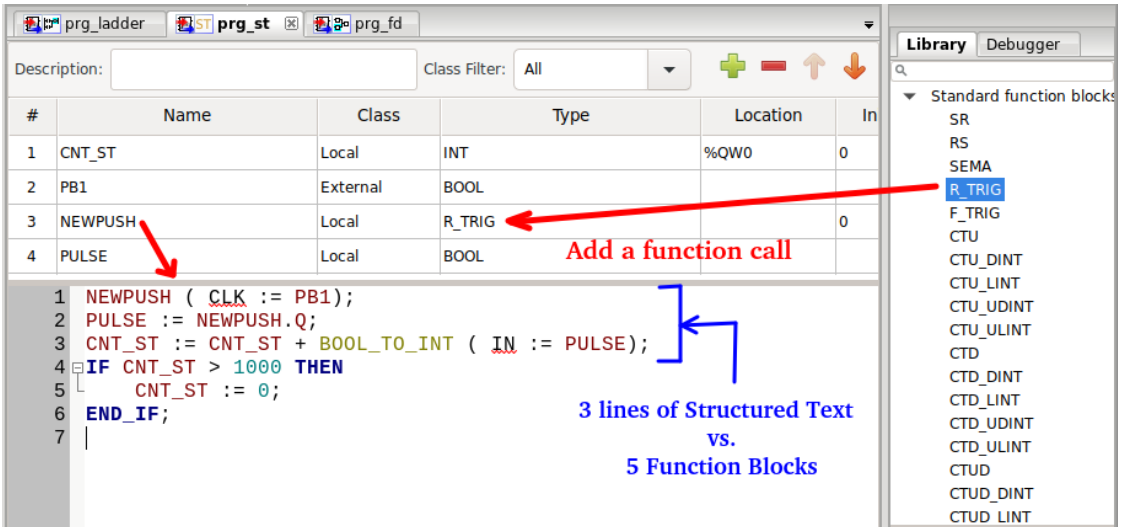
One of the advantages of Function Block Diagrams is that it is very readable and somewhat self documenting. The downside of FBD is that it can be messy for complex conditional logic.

Structured Text (ST) was developed as a programming option that can work along with the other 61131-3 languages. Structure Text is block structured and syntactically resembles Pascal.

For my Structured Text program I wanted to do the same functionality that was done in the earlier Function Block Diagram program. To do the same functionality in ST as the FBD programs it only took 3 lines of code vs. 5 Function Blocks.

In my ST program I added a simple IF condition to reset the push button counter if the value reached 1000.

It's important to note that library functions such as R_TRIG are available in all the 61131-3 programming languages. It is also possible to create your own custom functions in one programming language and they can then we used in all the other languages.



| # | Name | Class | Type | Location | In |
|---|---|---|---|---|---|
| 1 | CNT_ST | Local | INT | %QW0 | 0 |
| 2 | PB1 | External | BOOL | | |
| 3 | NEWPUSH | Local | R_TRIG | | 0 |
| 4 | PULSE | Local | BOOL | | |

```
NEWPUSH ( CLK := PB1);
PULSE := NEWPUSH.Q;
CNT_ST := CNT_ST + BOOL_TO_INT ( IN := PULSE);
IF CNT_ST > 1000 THEN
    CNT_ST := 0;
END_IF;
```

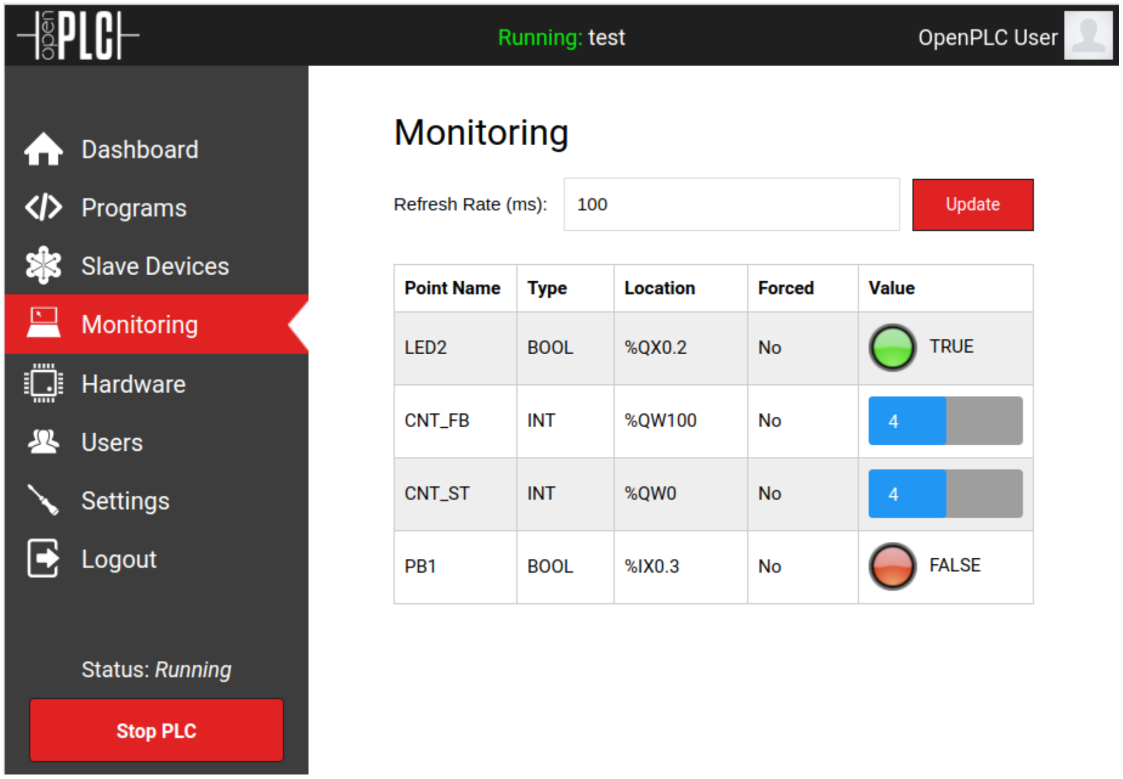
## Running OpenPLC Programs

After the three programs have been compiled and saved they can be install into the OpenPLC runtime application. To manually start the runtime application:

```
pi@pi4:~ $ cd OpenPLC_v3
pi@pi4:~/OpenPLC_v3 $ sudo ./start_openplc.sh &
```

The OpenPLC runtime will start a Web application on port 8080 on the Raspberry Pi. After logging into the web interface, the first step is to select the "Hardware" option and set the *OpenPLC Hardware Layer* to "Raspberry Pi". Next select the "Programs" option and upload the OpenPLC configuration file. After a new configuration file is uploaded and compiled, the final step is to press the "Start PLC" button.

The "Monitoring" option can be used to view the status of variables in the PLC configuration.



Monitoring

Refresh Rate (ms): 100 Update

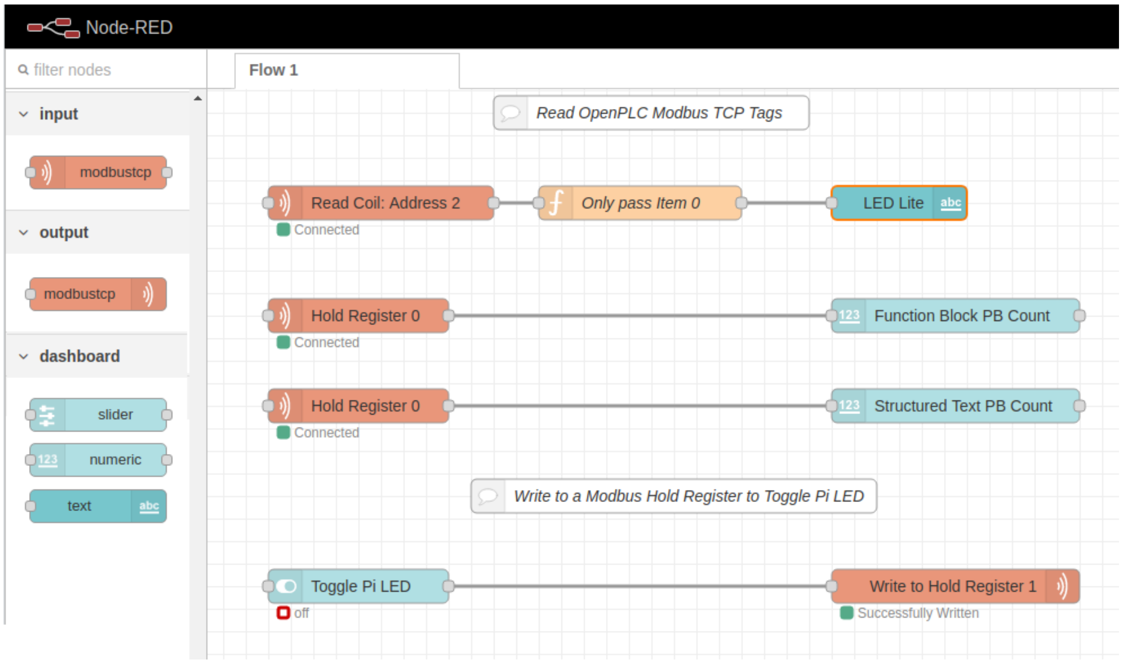
| Point Name | Type | Location | Forced | Value |
|---|---|---|---|---|
| LED2 | BOOL | %QX0.2 | No | 🟢 TRUE |
| CNT_FB | INT | %QW100 | No | 4 |
| CNT_ST | INT | %QW0 | No | 4 |
| PB1 | BOOL | %IX0.3 | No | 🔴 FALSE |

## Modbus with Node-Red

Modbus was the earliest and most common communication protocol used to connect Industrial devices together. Modbus can be used on serial interfaces (Modbus RTU) or on Ethernet networks (Modbus TCP), both are supported by OpenPLC.

Node-Red has a number of Modbus TCP nodes that can be used. I found that : ***node-red-contrib-modbustcp*** worked well for my application. New nodes can be added to Node-Red using the “Manage Palette” option.

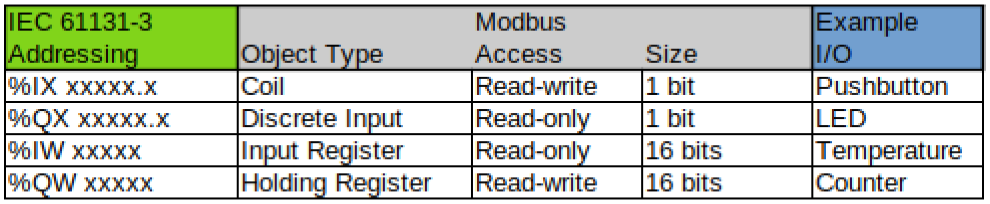
A simple Node-Red application that can monitor the LED and counter statuses would use three ***modbustcp input*** nodes and a ***text*** and two ***numeric*** nodes.



The Modbus read call returns 16 bits of information, so a small function was created ("Only pass Item 0") to change the msg payload to be just the first item in the array:

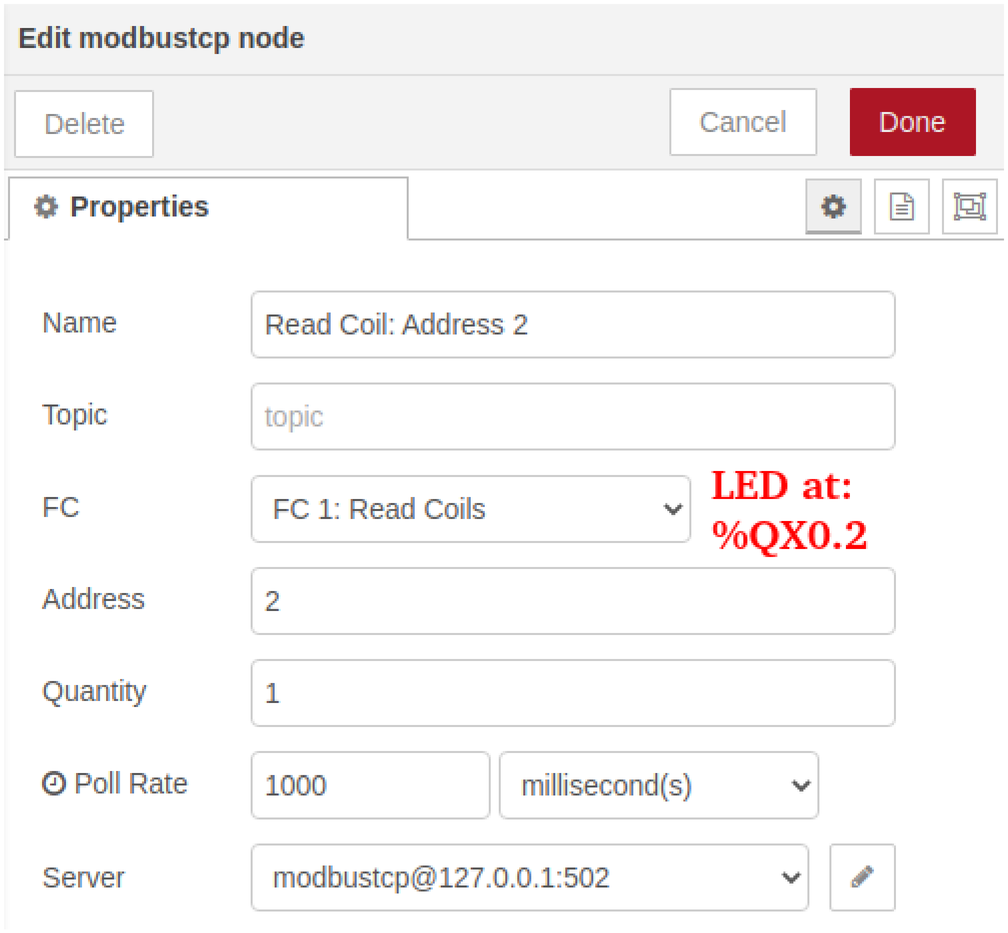
```
msg.payload = msg.payload[0];
return msg;
```

Modbus supports 4 object types; coils, discrete inputs, input registers and holding registers.



| IEC 61131-3 Addressing | Object Type | Modbus Access | Size | Example I/O |
|---|---|---|---|---|
| %IX xxxxx.x | Coil | Read-write | 1 bit | Pushbutton |
| %QX xxxxx.x | Discrete Input | Read-only | 1 bit | LED |
| %IW xxxxx | Input Register | Read-only | 16 bits | Temperature |
| %QW xxxxx | Holding Register | Read-write | 16 bits | Counter |

For this project the LED's IEC addressing is %QX0.2 and this would be a coil at address 2. The Function Block counter (CNT_FB) address of %QW100 is a Hold Register of 100, (CNT_ST is a Hold Register of 0).



Edit modbustcp node

Name: Read Coil: Address 2

Topic: topic

FC: FC 1: Read Coils

LED at: %QX0.2

Address: 2

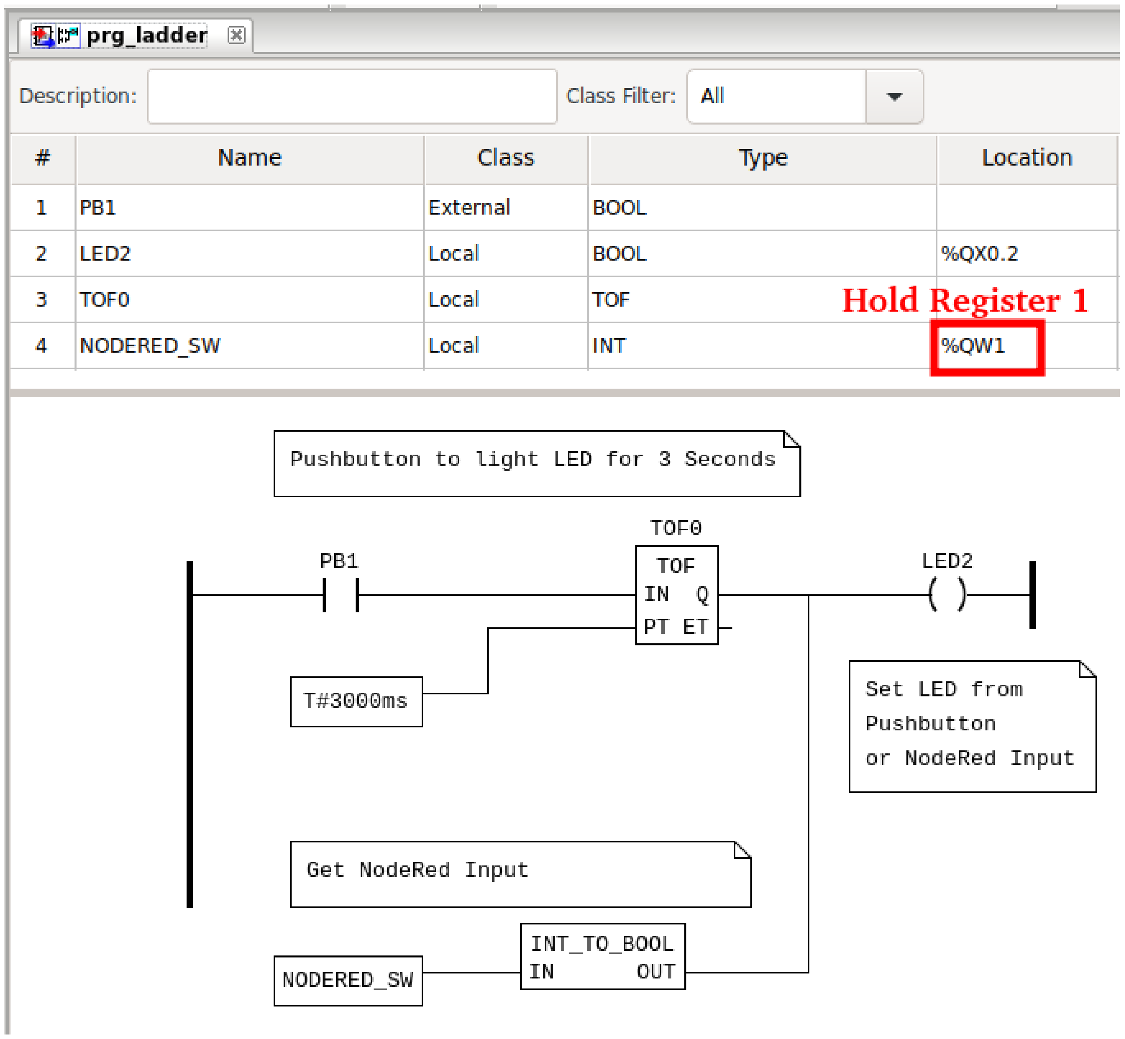
Quantity: 1

Poll Rate: 1000 millisecond(s)

Server: modbustcp@127.0.0.1:502

## Modbus Writing from Node-Red

The Ladder logic program was updated to light the LED from either the push button or a hold register. The hold register (%QW1) is an integer so the value is converted to a boolean then "OR"-ed with the push button interface.

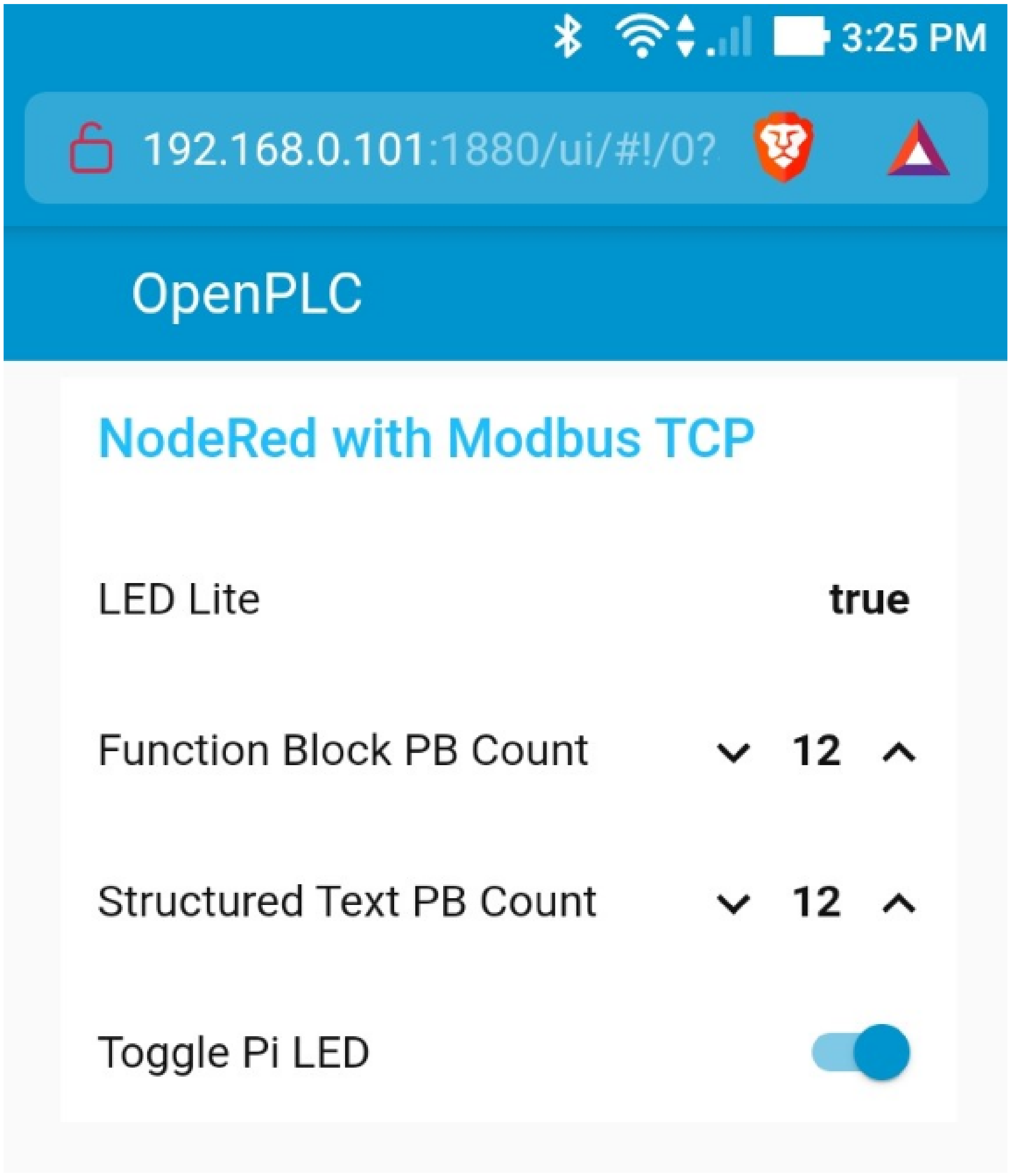


| # | Name | Class | Type | Location |
|---|---|---|---|---|
| 1 | PB1 | External | BOOL | |
| 2 | LED2 | Local | BOOL | %QX0.2 |
| 3 | TOF0 | Local | TOF | |
| 4 | NODERED_SW | Local | INT | %QW1 |

Pushbutton to light LED for 3 Seconds
TOF0
PB1
TOF
IN Q
PT ET
LED2
T#3000ms
Set LED from Pushbutton or NodeRed Input
Get NodeRed Input
INT_TO_BOOL
IN OUT
NODERED_SW

On Node-Red a ***slider*** node is used to pass a 0/1 to a ***modbus tcp output*** node, that write to hold register 1.

The Node-Red web dashboard is accessed at: http://your_rasp_pi:1880/ui/ (http://your_rasp_pi:1880/ui/)



NodeRed with Modbus TCP

LED Lite true

Function Block PB Count 12

Structured Text PB Count 12

Toggle Pi LED

## Final Comments

OpenPLC is an excellent testing and teaching tool for industrial controls.

## 3 thoughts on “OpenPLC on a Raspberry Pi”

**Francisco** says:
**MAY 4, 2022 AT 5:28 AM**

Congrats on a great project indeed with loads of room! would like to share insights on our project and discuss with you,

**REPLY**

**Pete Metcalfe** says:
**MAY 4, 2022 AT 8:29 PM**
Hi,
Sure, sounds good. I used to work in industrial controls, OpenPLC is cool and it's a great freeware package, but it's lacking in what's being used in most plant operations.

I'd be interested to hear where your vision and ideas are.

Cheers
Pete

**REPLY**

**Josef Bernhardt** says:
**AUGUST 22, 2022 AT 6:37 AM**
A great Info of openplcproject
https://www.elektor.com/plc-programming-with-the-raspberry-pi-and-the-openplc-project

**REPLY**

Start a Blog at WordPress.com.