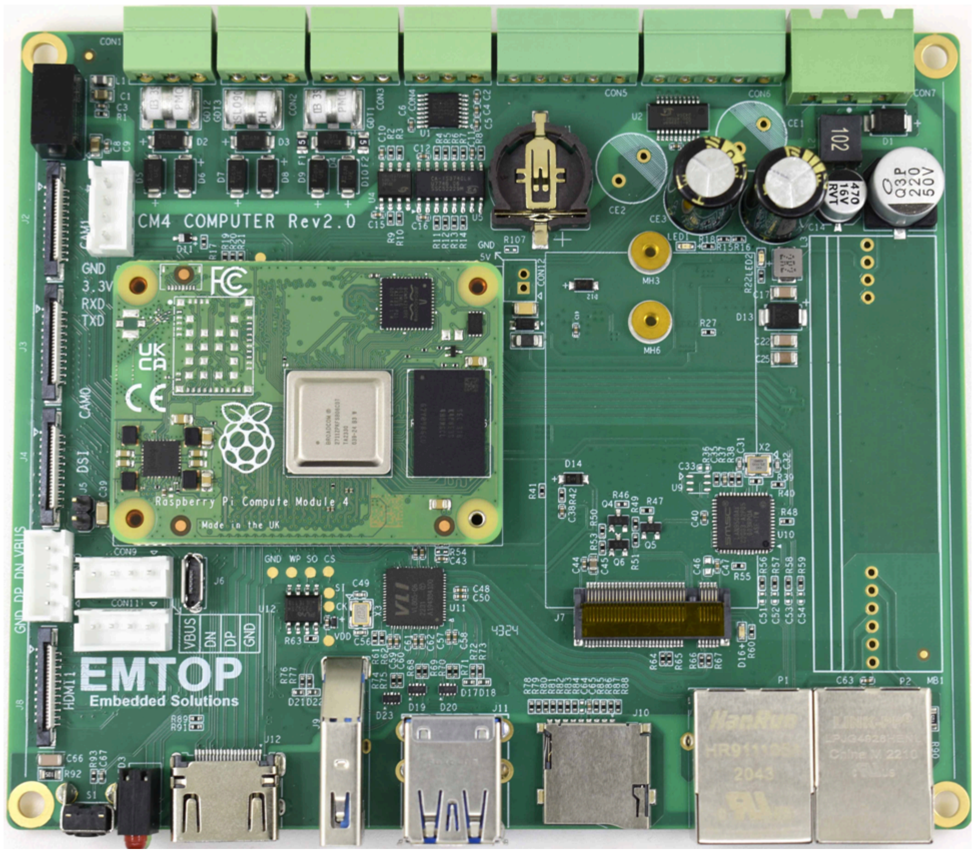
CM4-ET-IND
OpenPLC User Manual



Version: 0.1
2025-09-09

Revision History

Version	Date	Description
0.1	2025-09-09	Initial Release

Contents

1. Product Overview

1.1 Introduction

1.2 Resource

2. Linux Operation System

2.1 Make A Bootable TF Card

1. Unxz image file ***CM4-ET-IND-SD-REVXX.img.xz*** with WinRAR, to generate .img file;
2. Write image file ***CM4-ET-IND-SD-REVXX.img*** into SD card with Win32DiskImager;
3. Install the SD card on ARM board;
4. Connect the UART debug port to PC. Open the serial port [115200, 8N1] with terminal tool such as PuTTY, SecureCRT etc.
5. Connect the power cable and power up the board [12V/2A];
6. After a while, about 1 ~ 2 seconds, the terminal window will show the booting message as below:

```
Debian GNU/Linux 12 raspberrypi ttyS0

My IP address is 192.168.3.176 fe80::9c42:d2bc:94b8:ced1

raspberrypi login:
```

Note

- Default Login Account: ***pi*** with password ***raspberry***
- The screen [HDMI or DSI] will show “Welcome to Raspberry Pi” dialog after the first boot. It will do some basic settings, such as language, keyboard and create user account.

3. OpenPLC Editor

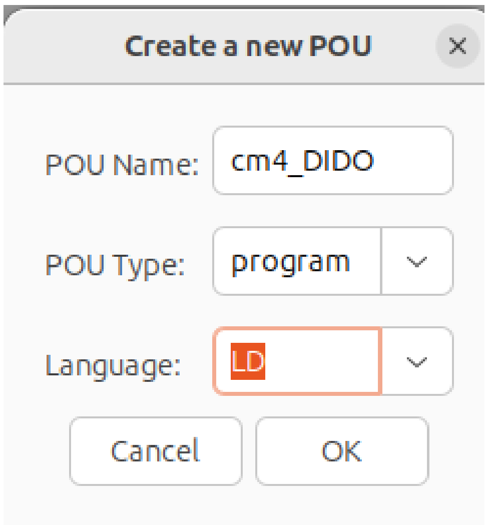
3.1 Install OpenPLC Editor on PC

OpenPLC Editor can be installed under Windows, MacOS and Linux platform. Please visit ***https://autonomylogic.com/download*** to download the correct version to run on your own PC platform.

Windows installation package: ***Tools/OpenPLCEditor-v1.3-Windows.zip***

3.2 Create An Example Project

1. Launch ***OpenPLC Editor*** program
2. Click File -> New:


Create a new POU

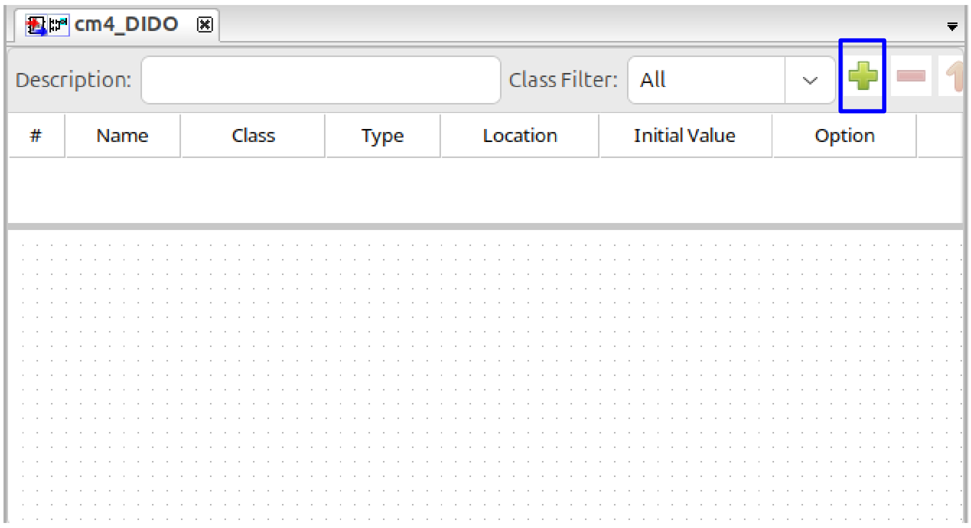
POU Name: cm4_DIDO

POU Type: program

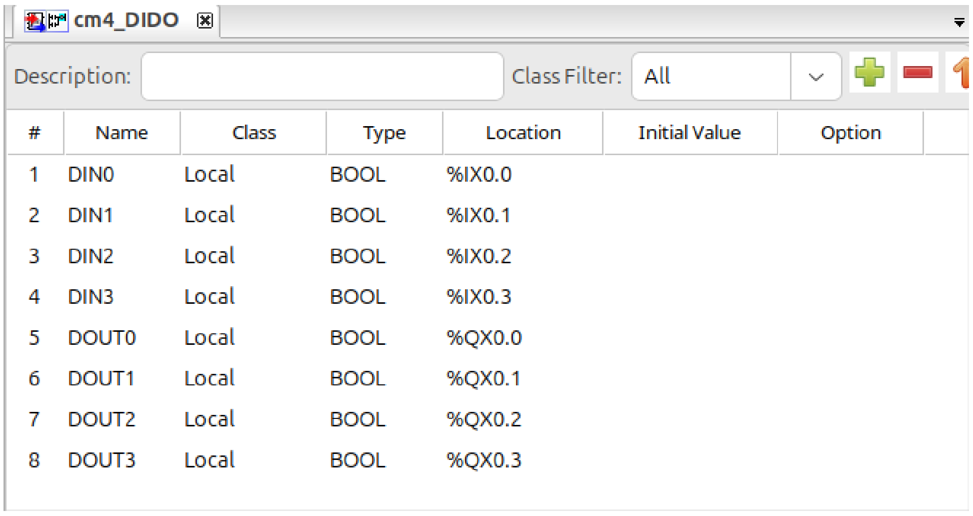
Language: LD

Figure 3-1: Create New Project

3. Add Variables



#	Name	Class	Type	Location	Initial Value	Option



cm4_DIDO

Description: Class Filter: All

#	Name	Class	Type	Location	Initial Value	Option
1	DIN0	Local	BOOL	%IX0.0		
2	DIN1	Local	BOOL	%IX0.1		
3	DIN2	Local	BOOL	%IX0.2		
4	DIN3	Local	BOOL	%IX0.3		
5	DOUT0	Local	BOOL	%QX0.0		
6	DOUT1	Local	BOOL	%QX0.1		
7	DOUT2	Local	BOOL	%QX0.2		
8	DOUT3	Local	BOOL	%QX0.3		

Figure 3-3: Add Variables

4. Create Ladder Logic Latch Circuit

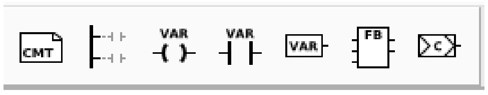
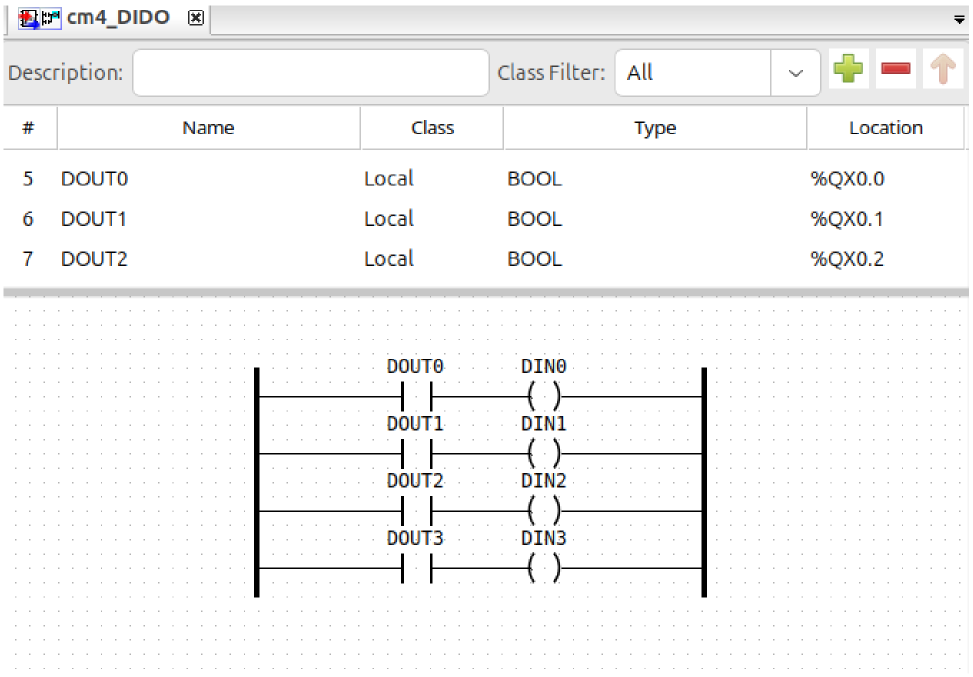


Figure 3-4: Circuit Tools



cm4_DIDO

Description: Class Filter: All

#	Name	Class	Type	Location
5	DOUT0	Local	BOOL	%QX0.0
6	DOUT1	Local	BOOL	%QX0.1
7	DOUT2	Local	BOOL	%QX0.2

Figure 3-5: Target Logic Circuit

The target logic circuit means using DINx get the DOUTx output status.

3.3 Build and Generate Program for OpenPLC Runtime

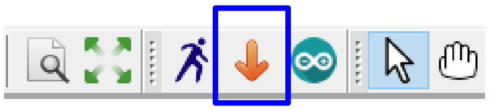


Figure 3-6: Generate Program for OpenPLC Runtime

Save as ***cm4_dido.st***.

3.4 Setup ARM Board Circuit

In order to make DINx and DOUTx controllable duiring test, we should connect them together on ARM board as below figure shows.

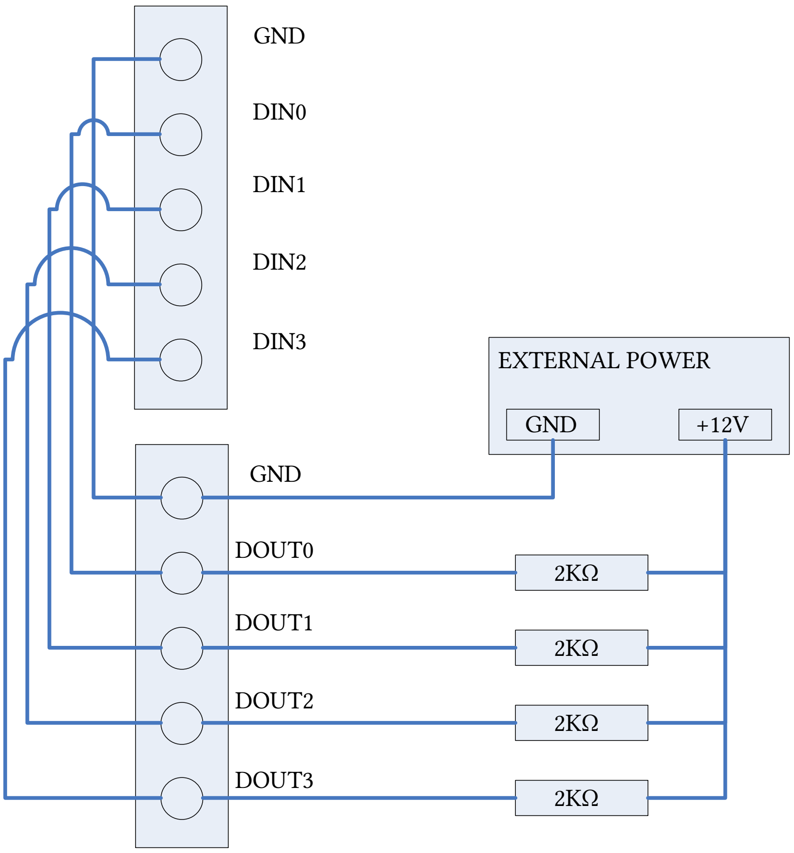


Figure 3-7: Board Circuit

4. OpenPLC Runtime

4.1 Install OpenPLC Runtime on ARM Board

Note

- User can write CM4-ET-IND-OPENPLC-REVXX.img to MicroSD card with win32diskimager to skip the installation steps.

Connect the board to Internet, and follow the below steps to install OpenPLC runtime.

```
pi@raspberrypi:~$ git clone https://github.com/thiagoralves/OpenPLC_v3.git
```

```
pi@raspberrypi:~$ cd OpenPLC_v3
```

```
pi@raspberrypi:~$ sudo ./install.sh rpi

Installing OpenPLC on Raspberry Pi
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://archive.raspberrypi.com/debian bookworm InRelease [55.0 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 k]
Get:4 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
......
LOCATED_VARIABLES.h
VARIABLES.csv
Config0.c
Config0.h
Res0.c
Including Siemens S7 Protocol via snap7
Moving Files...
Compiling for Linux
Generating object files...
Generating glueVars...
Compiling main program...
Compilation finished successfully!
```

```
pi@raspberrypi:~$ sudo reboot
```

User should get the ARM board IP address, eg. 192.168.3.176. Open web browser on PC and visit ***http://192.168.3.176:8080***.

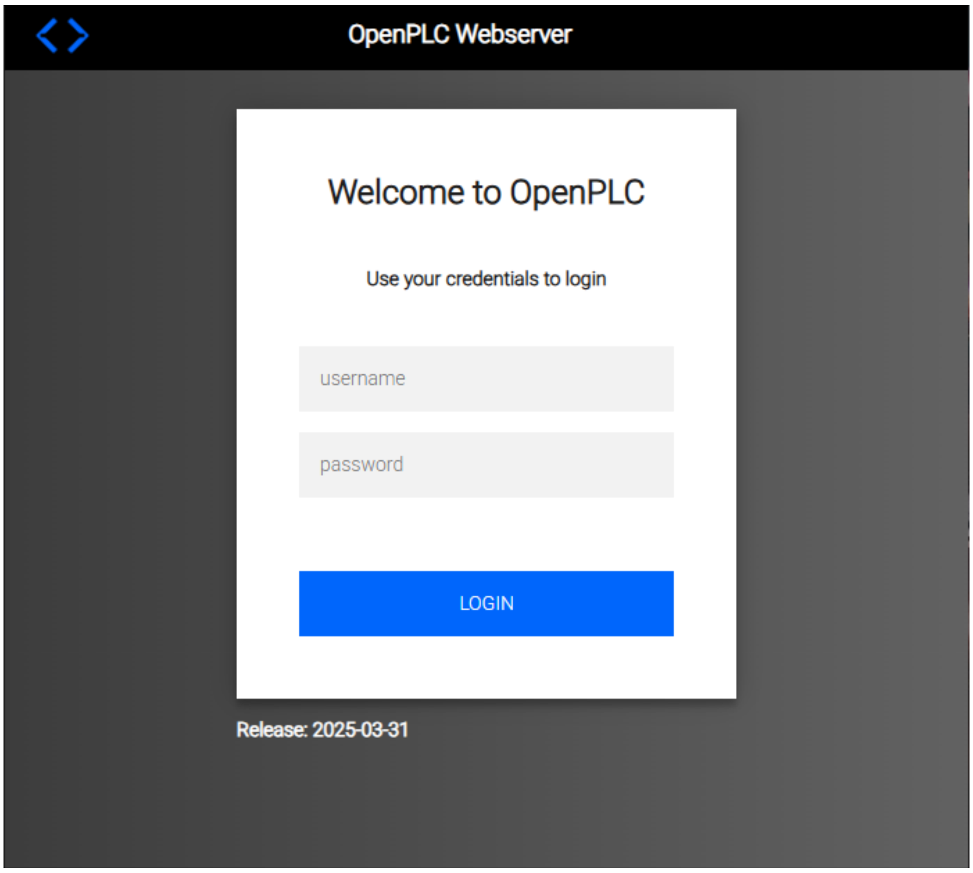
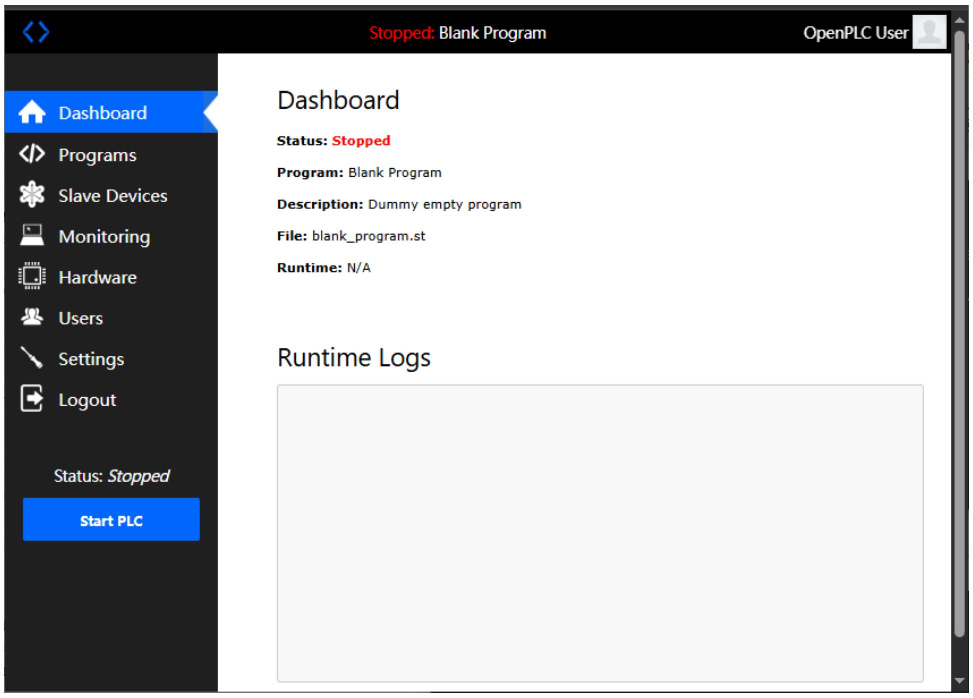


Figure 4-8: OpenPLC Login

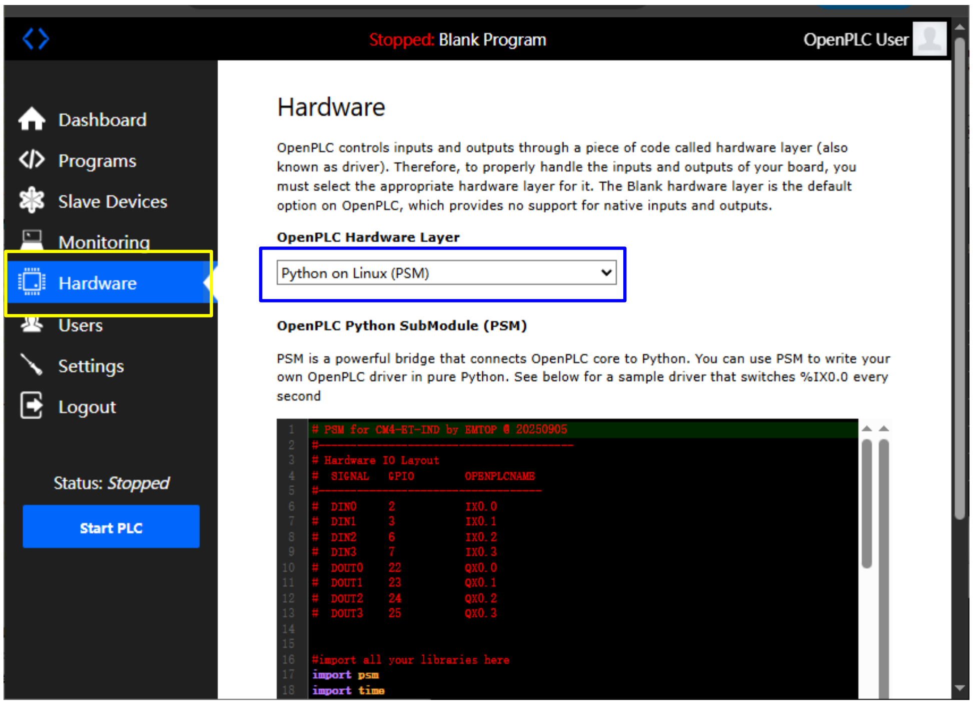
Login username and password are ***openplc***.



Status: Stopped
Program: Blank Program
Description: Dummy empty program
File: blank_program.st
Runtime: N/A

Figure 4-9: OpenPLC Mainwindow

4.2 Setup OpenPLC Python SubModule [PSM]



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

Python on Linux (PSM)

OpenPLC Python SubModule (PSM)

PSM is a powerful bridge that connects OpenPLC core to Python. You can use PSM to write your own OpenPLC driver in pure Python. See below for a sample driver that switches %IX0.0 every second

```
# PSM for CM4-ET-IND by EMTOP @ 20250905
#-------------------------------------
# Hardware IO Layout
#  SIGNAL   GPIO        OPENPLCNAME
#-------------------------------
#  DIN0     2           IX0.0
#  DIN1     3           IX0.1
#  DIN2     6           IX0.2
#  DIN3     7           IX0.3
#  DOUT0    22          QX0.0
#  DOUT1    23          QX0.1
#  DOUT2    24          QX0.2
#  DOUT3    25          QX0.3


#import all your libraries here
import psm
import time
```

Figure 4-10: Setting PSM

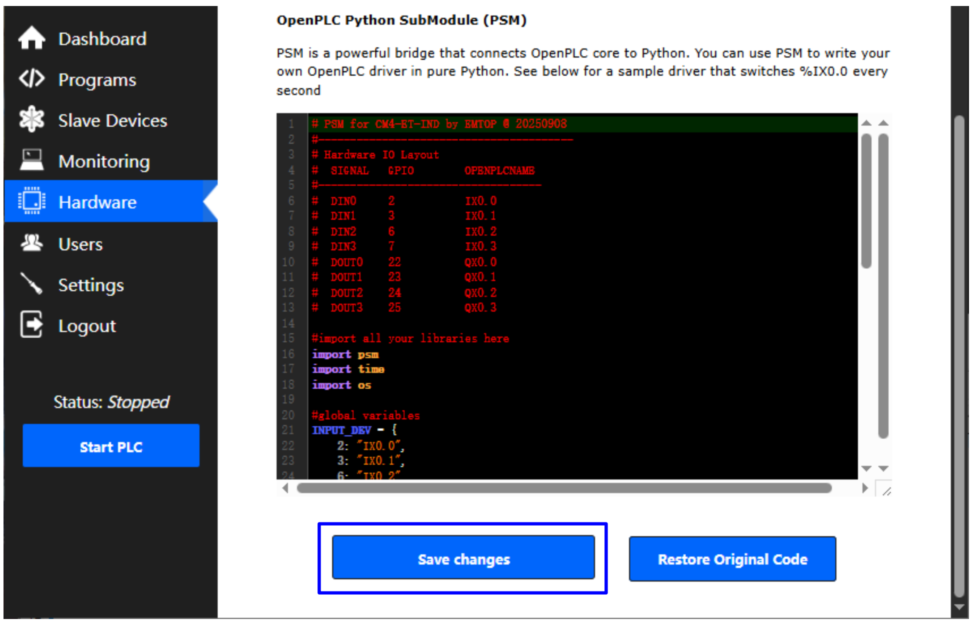
Open ***Source/Hardware-PSM-REVX.X.txt*** and paste its content to the dialog.

PSM

```
# PSM for CM4-ET-IND by EMTOP @ 20250908
#-----------------------------------------
# Hardware IO Layout
#  SIGNAL   GPIO        OPENPLCNAME
#-----------------------------------
#  DIN0     2           IX0.0
#  DIN1     3           IX0.1
#  DIN2     6           IX0.2
#  DIN3     7           IX0.3
#  DOUT0    22          QX0.0
#  DOUT1    23          QX0.1
#  DOUT2    24          QX0.2
#  DOUT3    25          QX0.3

#import all your libraries here
import psm
import time
import os

#global variables
INPUT_DEV = {
    2: "IX0.0",
    3: "IX0.1",
    6: "IX0.2",
    7: "IX0.3",
}

OUTPUT_DEV = {
    22: "QX0.0",
    23: "QX0.1",
    24: "QX0.2",
    25: "QX0.3",
}

def hardware_init():
    #Insert your hardware initialization code in here
    psm.start()

def update_inputs():
    #place here your code to update inputs
    for index, name in INPUT_DEV.items():
        command = os.popen('gpioget 0 {}'.format(index))
        psm.set_var(name, int(command.read()))

def update_outputs():
    #place here your code to work on outputs
    for index, name in OUTPUT_DEV.items():
        a = psm.get_var(name)
        os.system('gpioset 0 {}={}'.format(index, 1 if a else 0))

if __name__ == "__main__":
    hardware_init()
    while (not psm.should_quit()):
        update_inputs()
        update_outputs()
        time.sleep(0.05) #You can adjust the psm cycle time here
    psm.stop()
```

The PSM script binds the ***Location*** name in OpenPLC Editor with GPIOs on ARM board.

Click ***Save Changes***. Then click ***Go to Dashboard***.



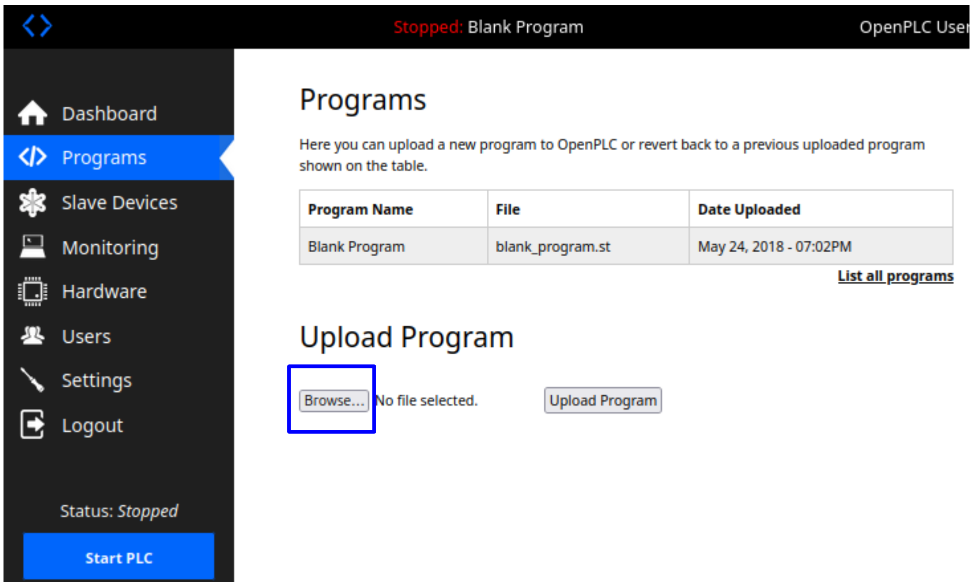
OpenPLC Python SubModule (PSM)

PSM is a powerful bridge that connects OpenPLC core to Python. You can use PSM to write your own OpenPLC driver in pure Python. See below for a sample driver that switches %IX0.0 every second

```
# PSM for CM4-ET-IND by EMTOP @ 20250908
#-----------------------------------
# Hardware IO Layout
#  SIGNAL   GPIO        OPENPLCNAME
#-------------------------------
#  DIN0     2           IX0.0
#  DIN1     3           IX0.1
#  DIN2     6           IX0.2
#  DIN3     7           IX0.3
#  DOUT0    22          QX0.0
#  DOUT1    23          QX0.1
#  DOUT2    24          QX0.2
#  DOUT3    25          QX0.3

#import all your libraries here
import psm
import time
import os

#global variables
INPUT_DEV = {
    2: "IX0.0",
    3: "IX0.1",
    6: "IX0.2"
```

Figure 4-11: Save PSM

4.3 Setup OpenPLC Program



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
Blank Program	blank_program.st	May 24, 2018 - 07:02PM

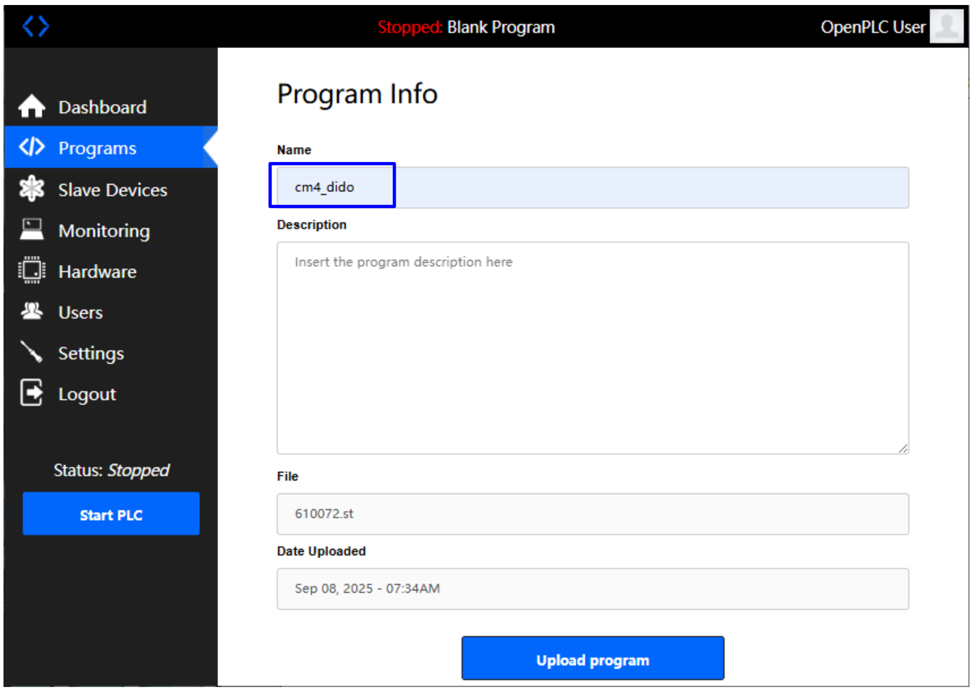
Upload Program

No file selected.

Figure 4-12: Choose Program

Choose the target file ***cm4_dido.st*** created by OpenPLC Editor, then click the button ***Upload Program***.

Git it a name, click the button ***Upload Program***.



Program Info

Name: cm4_dido

Description: Insert the program description here

File: 610072.st

Date Uploaded: Sep 08, 2025 - 07:34AM

Figure 4-13: Upload Program

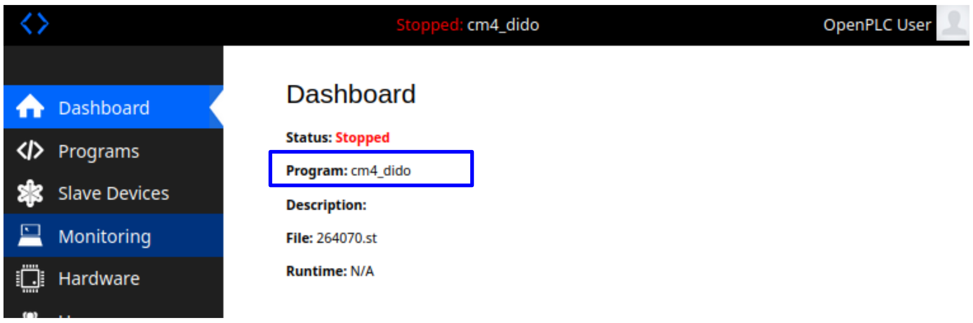
Wait until it completes successfully, and ***Go to Dashboard***.

Compiling program

```
Generating C files...
POUS.c
POUS.h
LOCATED_VARIABLES.h
VARIABLES.csv
```

```
Config0.c
Config0.h
Res0.c
Including Siemens S7 Protocol via snap7
Moving Files...
Compiling for Linux
Generating object files...
Generating glueVars...
varName: __IX0_0  varType: BOOL
varName: __IX0_1  varType: BOOL
varName: __IX0_2  varType: BOOL
varName: __IX0_3  varType: BOOL
varName: __QX0_0  varType: BOOL
varName: __QX0_1  varType: BOOL
varName: __QX0_2  varType: BOOL
varName: __QX0_3  varType: BOOL
Compiling main program...
Compilation finished successfully!
```

Now we can see the current program already changes to the target program ***cm4_dido***.



Dashboard

Status: Stopped

Program: cm4_dido

Description:

File: 264070.st

Runtime: N/A

Figure 4-14: Dashboard

4.4 Start OpenPLC

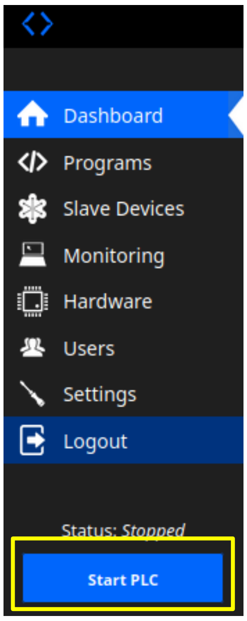


Figure 4-15: Start OpenPLC

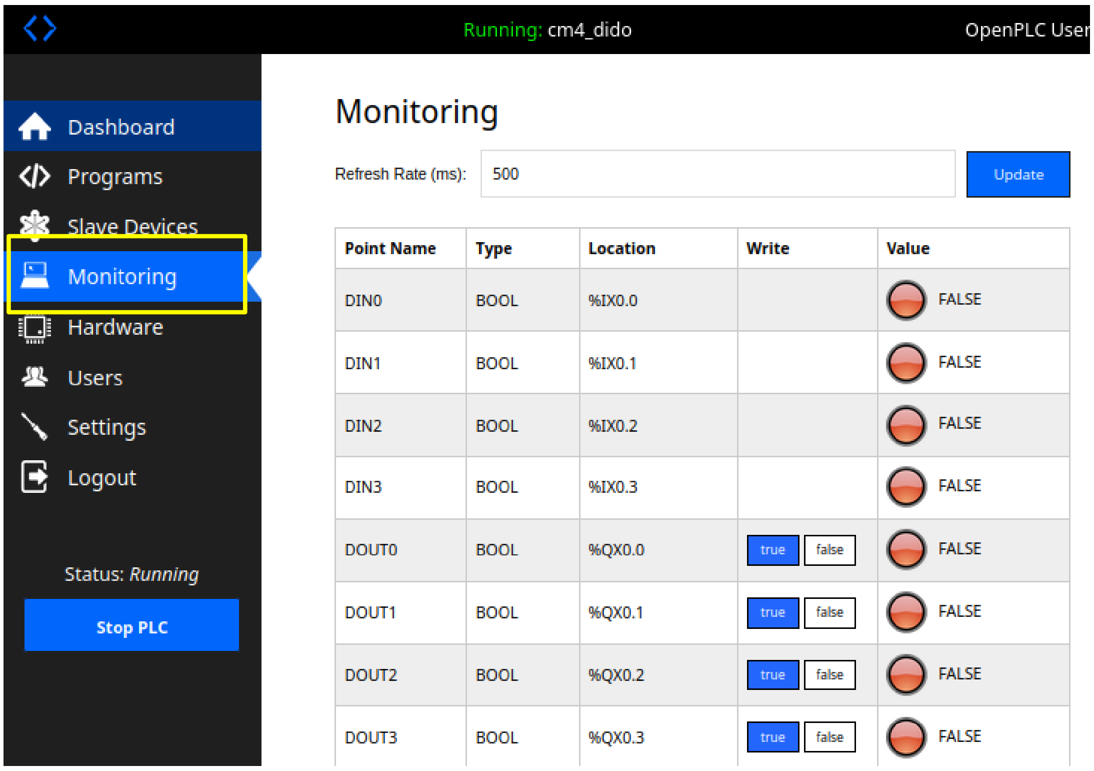
Runtime Logs

```
OpenPLC Runtime starting...
Interactive Server: Listening on port 43628
PSM: Starting PSM...
Issued start_modbus() command to start on port: 502
Server: Listening on port 502
Server: waiting for new client...
Issued stop_dnp3() command
Issued start_enip() command to start on port: 44818
Server: Listening on port 44818
Server: waiting for new client...
PSM: Connected to PSM
Skipping configuration of Slave Devices (mbconfig.cfg file not found)
Persistent Storage is empty
```

If it doesn't report any fatal error, it starts successfully.

4.5 Monitoring

Let's open ***Monitoring*** window, it shows all variables we designed in OpenPLC Editor.



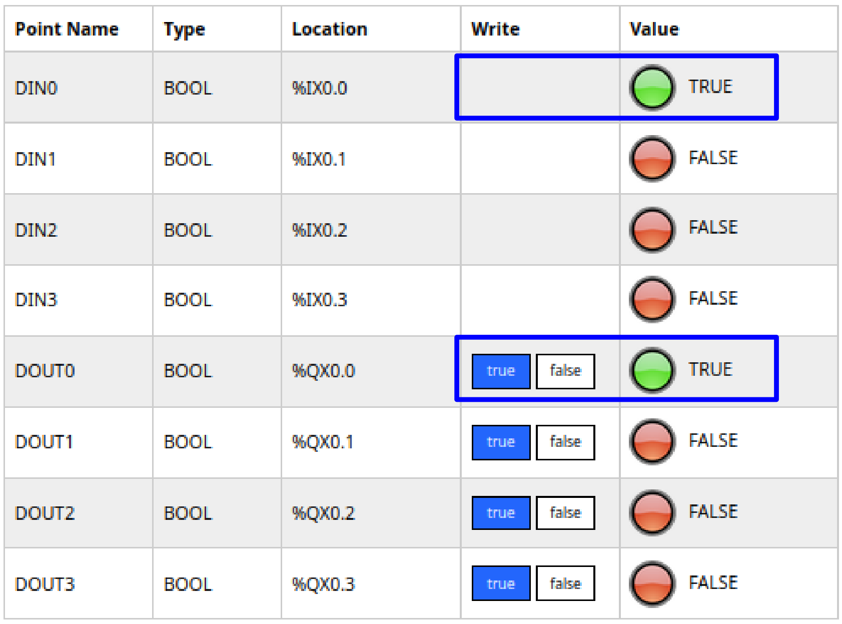
Monitoring

Refresh Rate (ms): 500

Point Name	Type	Location	Write	Value
DIN0	BOOL	%IX0.0		FALSE
DIN1	BOOL	%IX0.1		FALSE
DIN2	BOOL	%IX0.2		FALSE
DIN3	BOOL	%IX0.3		FALSE
DOUT0	BOOL	%QX0.0	true false	FALSE
DOUT1	BOOL	%QX0.1	true false	FALSE
DOUT2	BOOL	%QX0.2	true false	FALSE
DOUT3	BOOL	%QX0.3	true false	FALSE

Figure 4-16: Monitoring Window

We can click the ***true*** and ***false*** button to change the DOUTx output status. Example, when DOUT0 output is changed, the DIN0 will change accordingly.



Point Name	Type	Location	Write	Value
DIN0	BOOL	%IX0.0		🟢 TRUE
DIN1	BOOL	%IX0.1		🔴 FALSE
DIN2	BOOL	%IX0.2		🔴 FALSE
DIN3	BOOL	%IX0.3		🔴 FALSE
DOUT0	BOOL	%QX0.0	true false	🟢 TRUE
DOUT1	BOOL	%QX0.1	true false	🔴 FALSE
DOUT2	BOOL	%QX0.2	true false	🔴 FALSE
DOUT3	BOOL	%QX0.3	true false	🔴 FALSE

Figure 4-17: Monitoring Variables

5. Appendix