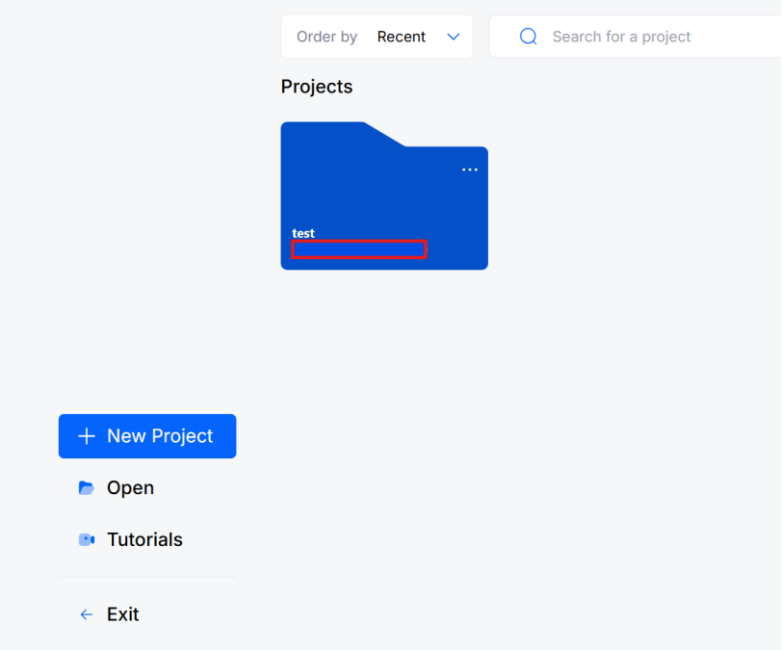
250826

Automazione 2 - Marcia-arresto in Structured Text con OpenPLC Editor

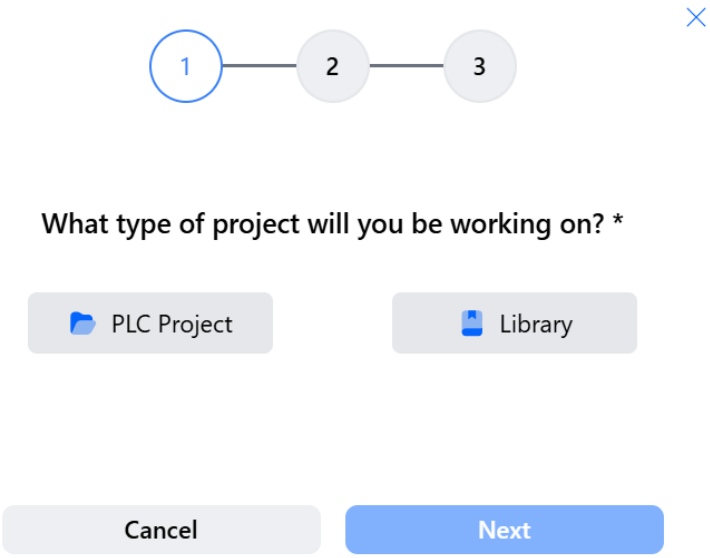
Questa prova realizza un comando marcia-arresto con autoritenuta in Structured Text, lo verifica nel simulatore di OpenPLC Editor e lo trasferisce poi su una scheda basata su ATmega2560. Gli ingressi fisici usati sono i pin 2 e 3; l'uscita è il pin 13, collegato anche al LED integrato della scheda.

1. CREAZIONE DEL PROGETTO

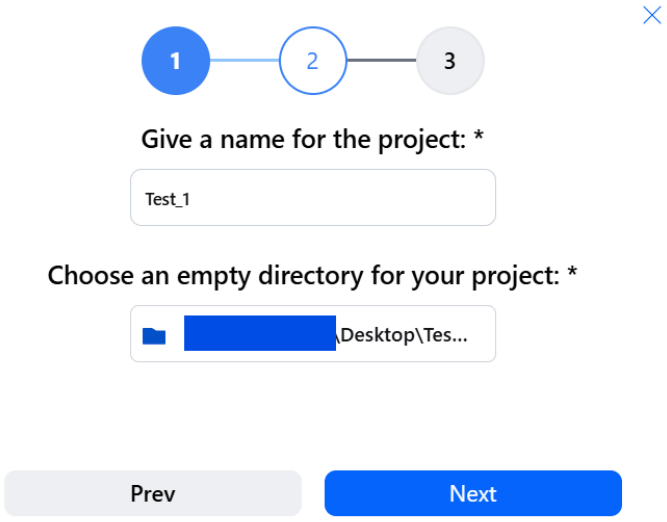
Dalla schermata iniziale di OpenPLC Editor si seleziona New Project.



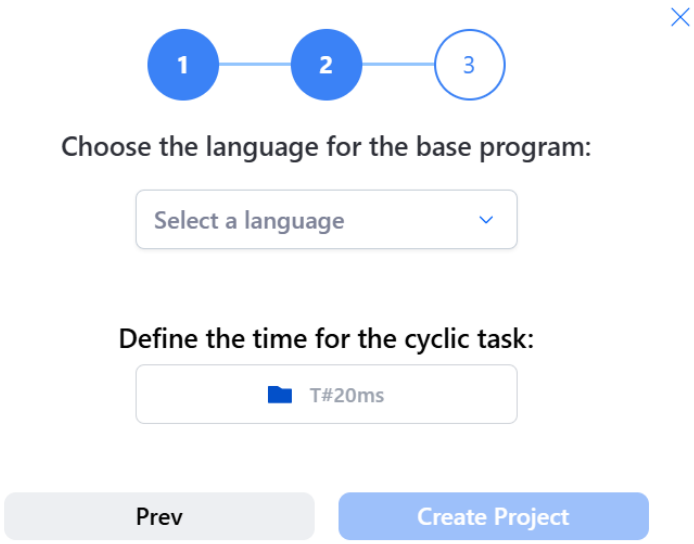
Come tipo di progetto si sceglie PLC Project e si procede con Next.

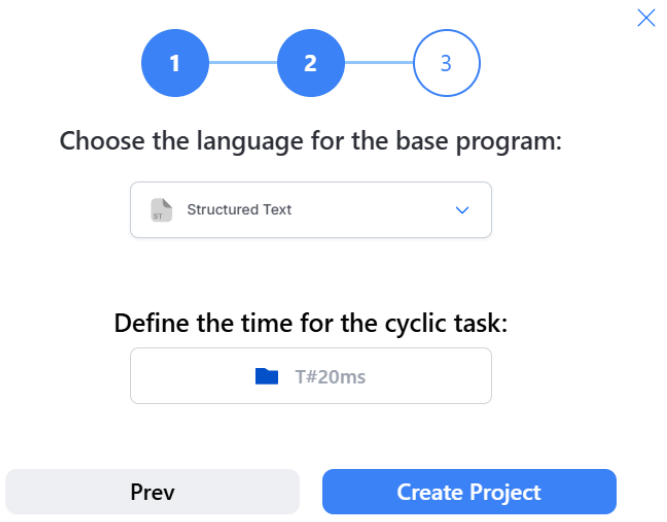


Si assegna quindi un nome al progetto e si indica una cartella vuota nella quale salvarlo. Nell'esempio il progetto è denominato Test_1.

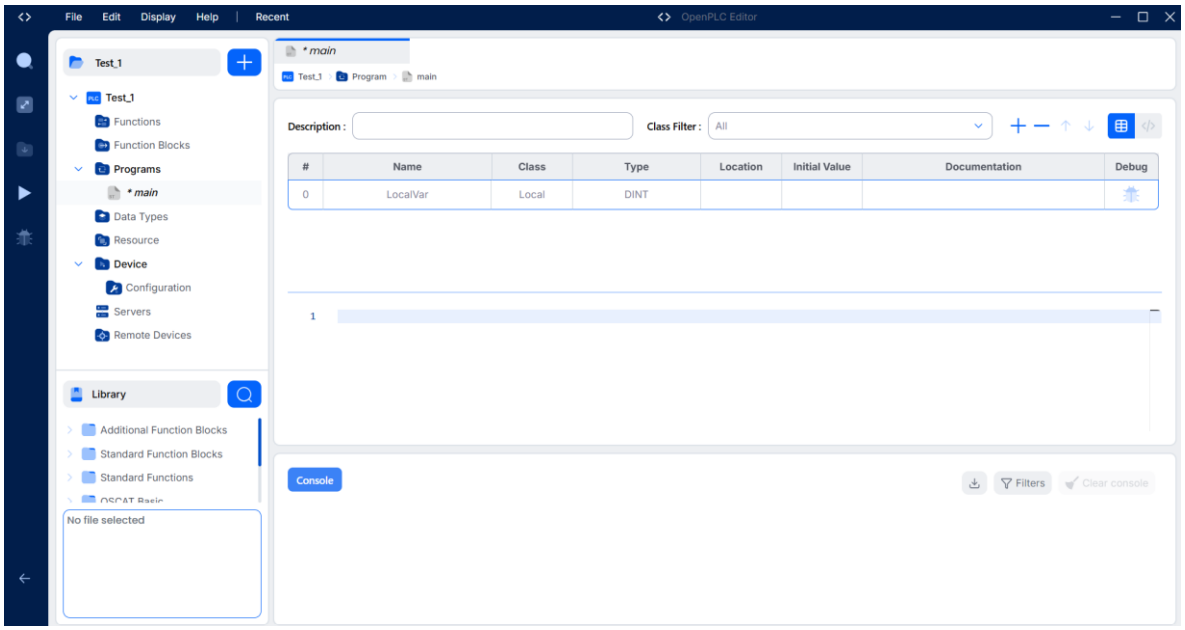


Nella terza schermata si sceglie il linguaggio del programma principale. Per questa prova si utilizza Structured Text. Il task ciclico viene lasciato al valore T#20ms.



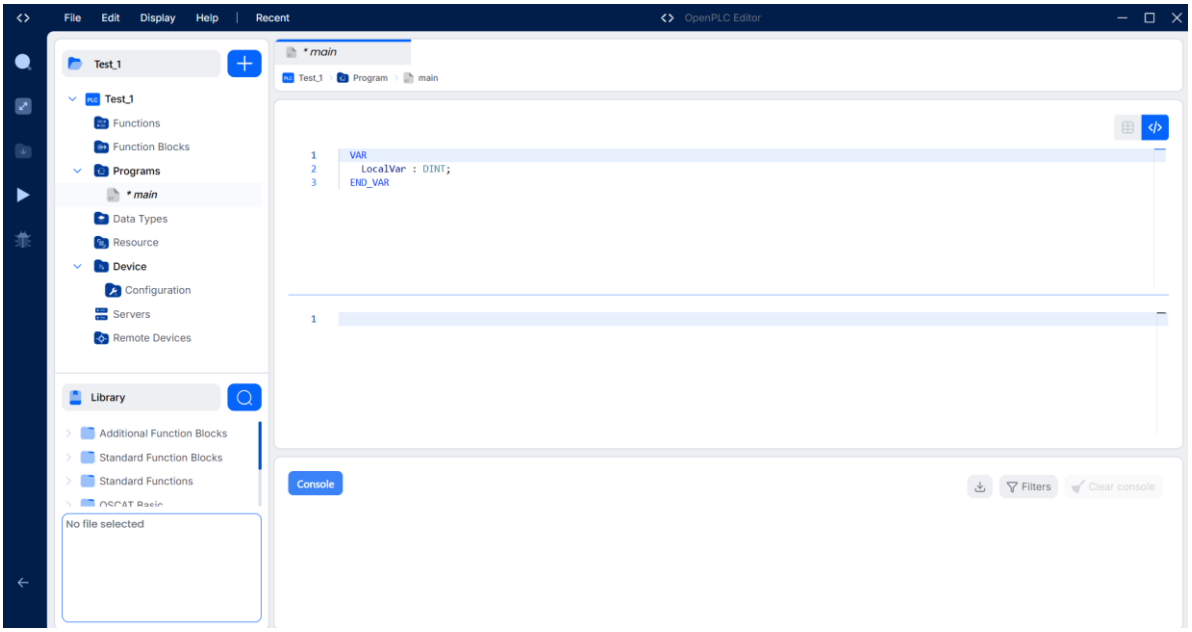


Con Create Project viene aperto automaticamente il programma main. La parte superiore contiene le variabili del POU; la parte inferiore è l'editor del codice ST.



2. DICHIARAZIONE DELLE VARIABILI

Le variabili possono essere gestite in forma tabellare oppure come dichiarazioni IEC. Premendo il pulsante </> a destra della tabella si passa alla modalità testuale.

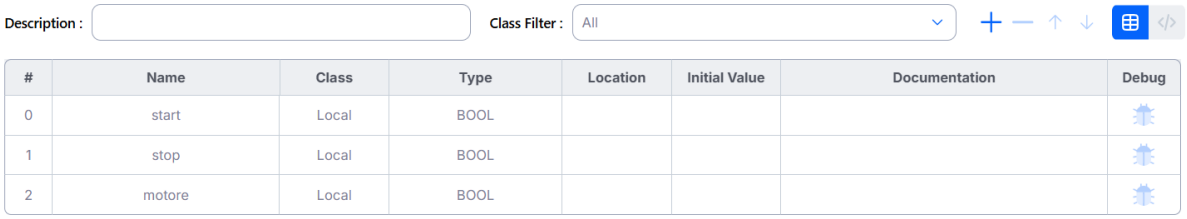


La variabile predefinita viene sostituita con tre variabili booleane:

```
VAR
   start : BOOL;
   stop : BOOL;
   motore : BOOL;
END_VAR
```

```
1    VAR
2    start : bool;
3    stop : bool;
4    motore : bool;
5    END_VAR
```

Ritornando alla vista tabellare, OpenPLC Editor interpreta automaticamente le dichiarazioni: i tre nomi compaiono come variabili Local di tipo BOOL. Non è necessario reinserire manualmente nome, classe e tipo.



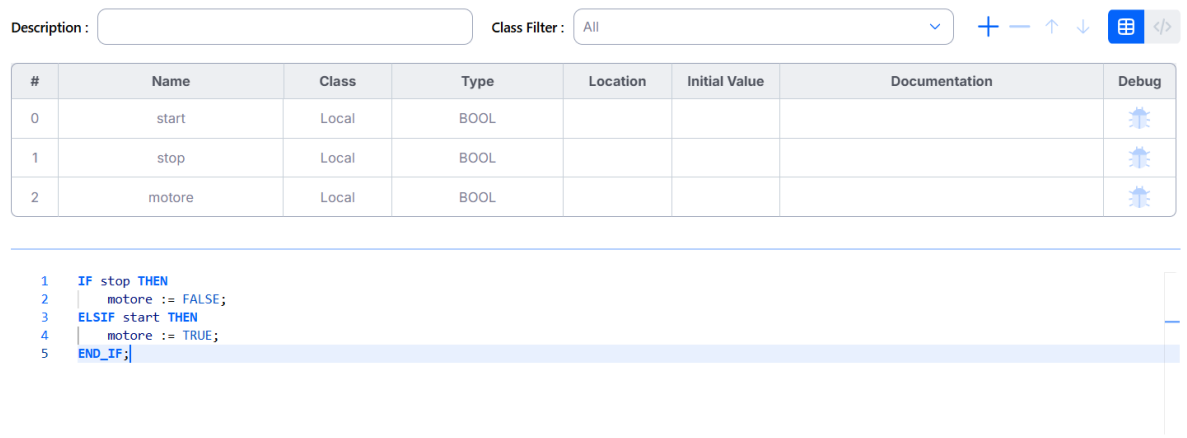
#	Name	Class	Type	Location	Initial Value	Documentation	Debug
0	start	Local	BOOL				
1	stop	Local	BOOL				
2	motore	Local	BOOL				

3. PROGRAMMA MARCIA-ARRESTO

Nel riquadro inferiore si inserisce il programma Structured Text:

```
IF stop THEN
   motore := FALSE;
ELSIF start THEN
   motore := TRUE;
END_IF;
```

La condizione di arresto viene valutata per prima e ha quindi priorità. Quando start e stop sono entrambi FALSE, il programma non assegna un nuovo valore a motore; essendo una variabile Local di un Program, motore conserva il valore tra gli scan. È questa memoria che realizza l'autoritenuta.

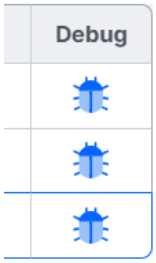


#	Name	Class	Type	Location	Initial Value	Documentation	Debug
0	start	Local	BOOL				
1	stop	Local	BOOL				
2	motore	Local	BOOL				

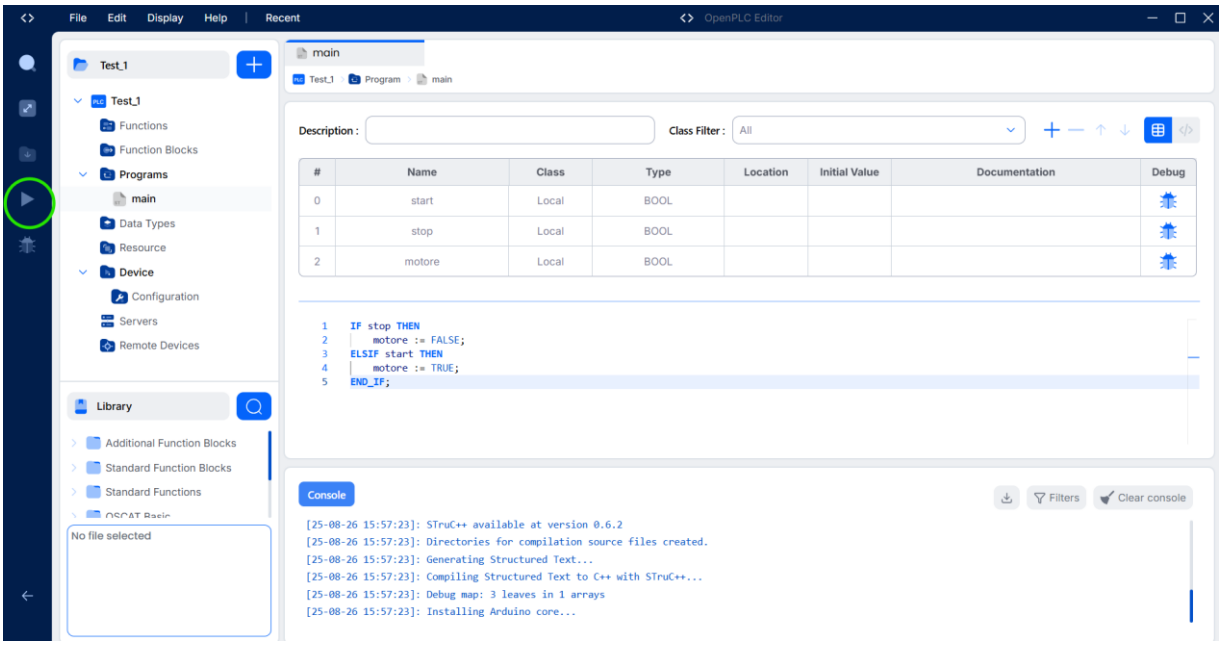
```
1   IF stop THEN
2       motore := FALSE;
3   ELSIF start THEN
4       motore := TRUE;
5   END_IF;
```

4. ATTIVAZIONE DEL DEBUG E SIMULAZIONE

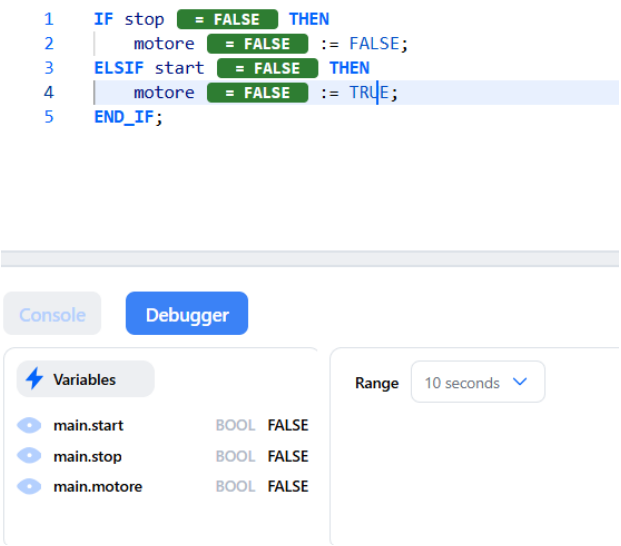
Nella colonna Debug si attivano le tre variabili premendo l'icona a forma di cimice. In questo modo il debugger potrà mostrarne il valore durante l'esecuzione.



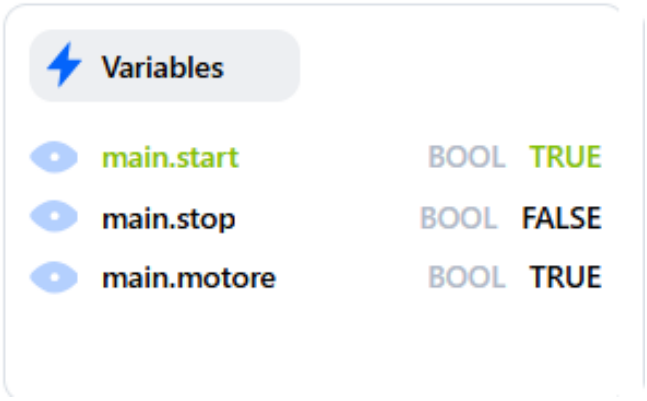
Il simulatore viene avviato con il pulsante Play nella barra verticale a sinistra. OpenPLC salva e compila il progetto; l'avanzamento è mostrato nella Console.



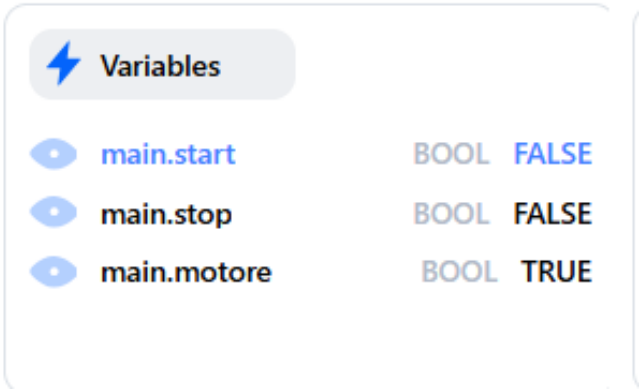
Al termine compare il pannello Debugger. Nella condizione iniziale tutte e tre le variabili sono FALSE.



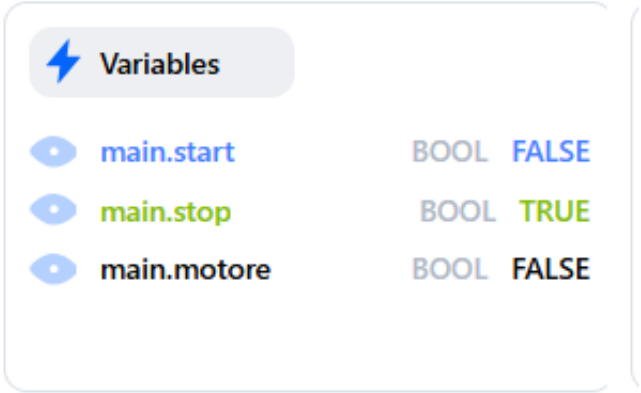
Dal Debugger si forza main.start a TRUE. Il programma porta immediatamente main.motore a TRUE.



Riportando main.start a FALSE, main.motore resta TRUE. Questa è la verifica dell'autoritenuta.

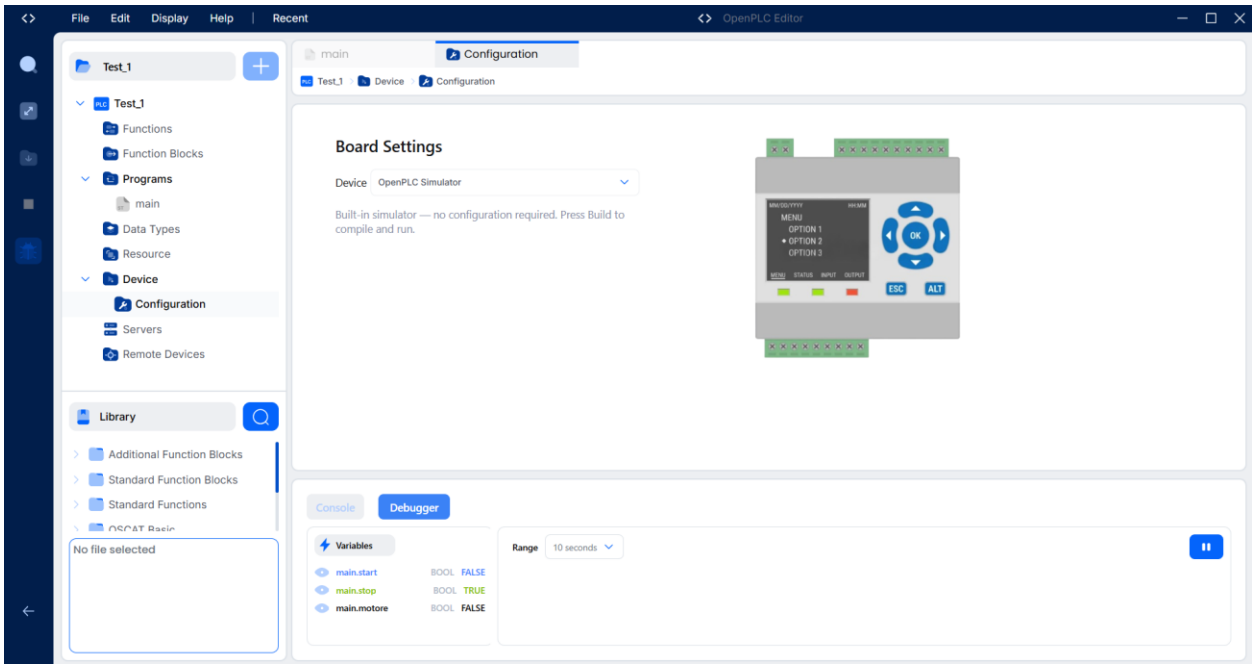


Forzando infine main.stop a TRUE, main.motore passa a FALSE. La funzione di arresto è quindi verificata.



5. PASSAGGIO ALL'HARDWARE REALE

Terminata la simulazione, si apre Device → Configuration. Inizialmente il dispositivo selezionato è ancora OpenPLC Simulator.



Nel menu Device si seleziona la scheda basata su ATmega2560. Dopo averla collegata via USB si aggiorna, se necessario, l'elenco delle porte con la freccia circolare e si seleziona la porta COM corrispondente.

Se la scheda non è ancora presente nell'elenco dei dispositivi, il Desktop Editor permette di installare i pacchetti aggiuntivi delle board prima di procedere.

6. PIN MAPPING

Nella sezione Pin Mapping si associa ogni pin fisico a un indirizzo IEC e a un alias leggibile. Nella prova sono stati utilizzati:

- pin 2 → Digital Input → %IX0.0 → start;
- pin 3 → Digital Input → %IX0.1 → stop;
- pin 13 → Digital Output → %QX0.0 → motore.

Pin Mapping

Pin	Type	Address	Alias
2	Digital Input	%IX0.0	start
3	Digital Input	%IX0.1	stop
13	Digital Output	%QX0.0	motore

Il pin 13 è stato scelto perché sulla scheda basata su ATmega2560 è collegato anche al LED integrato; questo permette di verificare direttamente lo stato dell'uscita senza aggiungere un LED esterno.

7. ASSOCIAZIONE DELLE VARIABILI AGLI INDIRIZZI IEC

Il Pin Mapping definisce il rapporto tra indirizzo IEC e pin fisico. Il programma deve poi associare le proprie variabili agli stessi indirizzi tramite la colonna Location.

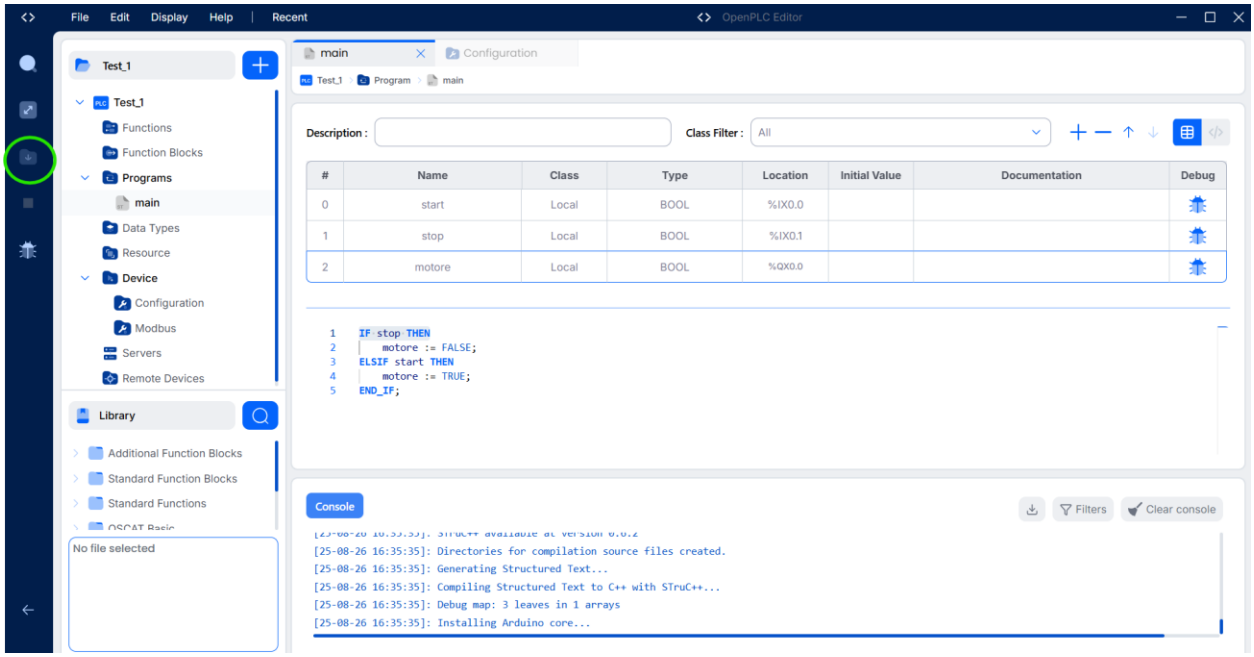
Le tre variabili restano di classe Local. Nella colonna Location si impostano:

- start → %IX0.0;
- stop → %IX0.1;
- motore → %QX0.0.

Nell'interfaccia provata, facendo doppio clic sulla cella Location compare il selettore degli indirizzi disponibili, quindi non è necessario ricordare e digitare ogni indirizzo manualmente.

8. COMPILAZIONE E CARICAMENTO

Il caricamento sull'hardware si avvia dall'icona Build nella barra verticale esterna a sinistra. Per un target Arduino-compatible il Desktop Editor traduce il programma IEC in C++, genera la configurazione necessaria e utilizza la toolchain Arduino per produrre e caricare il firmware.



La procedura è conclusa correttamente quando la Console conferma sia il caricamento sia la compilazione. Nella prova la porta utilizzata è stata COM5.

```
[25-08-26 16:36:33]: Nuova porta di caricamento: COM5 (serial)
[25-08-26 16:36:33]: Arduino upload complete.
[25-08-26 16:36:33]: ------------------------------------------
[25-08-26 16:36:33]: Compilation complete
```

9. PROVA REALE

Con il firmware caricato sulla scheda basata su ATmega2560, la prova sugli I/O fisici ha confermato il comportamento già osservato nel simulatore:

- attivando start sul pin 2, l'uscita motore va a TRUE e il LED sul pin 13 si accende;
- rilasciando start, l'uscita resta TRUE per effetto dell'autoritenuta;
- attivando stop sul pin 3, l'uscita torna FALSE e il LED si spegne.

La prova riguarda esclusivamente segnali logici a bassissima tensione della scheda. Gli ingressi digitali devono essere mantenuti a livelli logici definiti; non devono essere lasciati flottanti durante una verifica stabile.

CONCLUSIONI

La stessa logica è stata verificata prima nel simulatore e poi sull'hardware reale. La sequenza completa è: creazione del progetto, dichiarazione delle variabili, programma ST, debug, simulazione, selezione del dispositivo, Pin Mapping, associazione delle Location, Build/Upload e prova degli I/O. Il test conferma inoltre che, in un Program, le variabili Local possono essere associate direttamente agli I/O fisici mediante la colonna Location.

FONTI

Autonomy Logic — OpenPLC Editor, Quick Start Guide

Autonomy Logic — OpenPLC Editor, Workspace Layout

Autonomy Logic — OpenPLC Editor, ST Editor Features

Autonomy Logic — OpenPLC Editor, Variables & Data Types

Autonomy Logic — OpenPLC Editor, Variable classes inside a POU

Autonomy Logic — OpenPLC Editor, Debugger

Autonomy Logic — OpenPLC Editor, Device Configuration Overview

Autonomy Logic — OpenPLC Editor, Building a project

Arduino — pinout ufficiale della scheda basata su ATmega2560