Getting started with the STSW-PLC001 evaluation firmware for the STEVAL-PLC001V1

Introduction

The STSW-PLC001 is the preloaded firmware package for the STEVAL-PLC001V1 board that allows selecting different use cases through the board touchscreen.

You can choose among several use cases: DIDO in which each digital output (DO) mimics the corresponding digital input (DI); Information that displays the board info; Ladder logic, which is a simple ladder logic example; Self-test, which consists of a series of tests for touchscreen display, external memories (SRAM and Flash) and input and output channels with predefined pattern loopback connections; User defined, which contains use cases defined by the user.

The firmware package provides user API functions to invoke board support routines, such as APIs to access each of the 12 digital inputs and 12 digital outputs individually or collectively (per module). Other APIs check faults in input/output modules, control HMI back-light intensity, debug LEDs, enable or disable a module.

The firmware also includes FreeRTOS™ real-time operating system for microcontrollers, as well as sample predefined on/off delay and retentive timers. It also provides source code, STM32CubeMX files, STM32 HAL, and LL libraries to access the hardware.

STM32 TouchGFX graphics engine based HMI displays the board and application status. It also gathers user inputs.

1 Acronyms

Table 1. List of acronyms

Acronym	Description
PLC	Programmable logic controller
API	Application programming interface
HMI	Human machine interface
DI	Digital input
DO	Digital output

2 Getting started

2.1 Overview

The STSW-PLC001 features:

- Framework to use STEVAL-PLC001V1 evaluation board modules including TouchGFX-based HMI
- Simple user APIs to access each of the 12 digital inputs and 12 digital outputs individually or collectively (per module)
- Fault and status reporting
- Board support utility routines and predefined software timers
- Demo use cases including board information mode
- Easily expandable to include user-defined use cases, including ladder logic

The evaluation firmware package is developed around STM32 development environments and libraries to exploit the on-board STM32F746ZGT7 MCU.

The high-performance MCU with ARM Cortex-M7 core manages 12 digital inputs (DI) and 12 digital outputs (DO) as well as the HMI with touch display, based on TouchGFX graphic designer framework.

The DIs are arranged in two modules (I0 and I1) containing, respectively, eight and four channels managed by CLT01-38SQ7 and two CLT03-2Q3 devices. Similarly, the DOs are arranged in two modules (Q0 and Q1) of eight and four channels, respectively, managed by ISO8200AQ and IPS4260L devices. For further information, refer to UM2933 freely available on www.st.com.

The firmware implements few demonstration use cases and board support routines. It can be easily customized and expanded.

The configuration and initialization of the software are based on STM32CubeMX, whereas the graphical user interface is based on TouchGFX, distributed within STM32Cube MCU packages.

For debug, you can use STM32CubeIDE or third-party tools provided by IAR Systems or Keil.

To modify code flashed on the board, use STLINK-V3MINI or STLINK-V3SET in-circuit debugger and programmer for STM32.

2.2 Architecture

The STSW-PLC001 can be divided in two distinct functional blocks linked through one of the demonstration use cases:

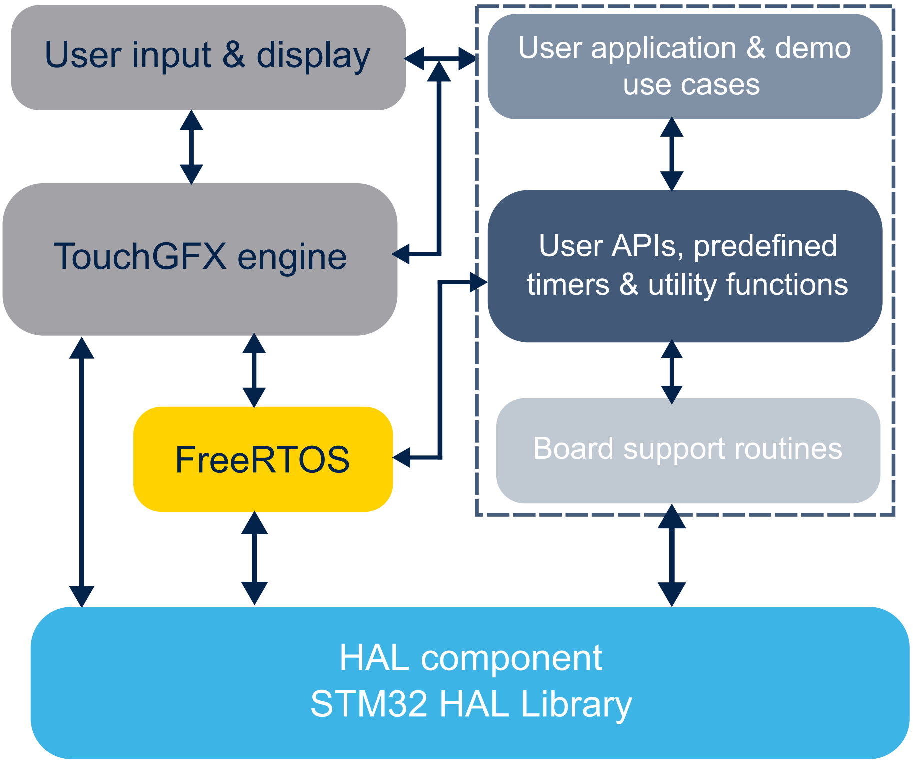
1. Graphical user interface display software to display status, information and to gather user input. User interface is easily upgradable and customizable through versatile TouchGFX framework. A dedicated task is performed via FreeRTOS kernel APIs included in STM32 development environment.
2. Simple user APIs:
   - to access each of the12 digital inputs and 12 digital outputs individually or collectively (per module);
   - to monitor fault and status;
   - to provide board support and utility routines;
   - three different types of predefined PLC timers that use FreeRTOS software timer API. These timer types are on-delay, off-delay, and retentive on-delay timer.
   - to control HMI back-light intensity;

The demonstration firmware implements few simple use-cases. Demonstration use-cases and user APIs are executed in the same task through FreeRTOS kernel APIs.

A task that executes display software performs appropriate communication and synchronization.

You can select a use case among the available ones through the touchscreen interface.

Figure 1. STSW-PLC001 functional block diagram

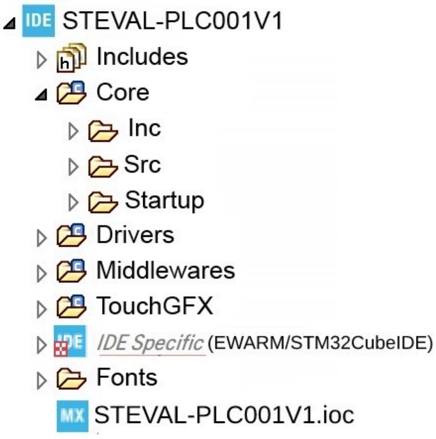


2.3 Folder structure

The figure below shows STSW-PLC001 folder structure, which follows the generic convention for STM32 firmware organization.

The 'Core' folder contains application and underlying routines (surrounded by a dotted block in Figure 1).

Figure 2. STSW-PLC001 folder structure



```
◢ STEVAL-PLC001V1
    ▷ Includes
    ◢ Core
        ▷ Inc
        ▷ Src
        ▷ Startup
    ▷ Drivers
    ▷ Middlewares
    ▷ TouchGFX
    ▷ IDE Specific (EWARM/STM32CubeIDE)
    ▷ Fonts
      STEVAL-PLC001V1.ioc
```

2.4 Graphical user interface with TouchGFX technology

TouchGFX is an advanced, free-of-charge GUI optimized for STM32 microcontrollers. Taking advantage of the STM32 graphical hardware acceleration, architecture, and ecosystem, TouchGFX accelerates the HMI-of-Things revolution through the creation of stunning user interfaces on embedded devices ranging from simple low-color UI applications up to high-resolution and high-color UI applications.

The TouchGFX solution is distributed as an STM32Cube Expansion Package (X-CUBE-TOUCHGFX), which helps users develop their UI application.

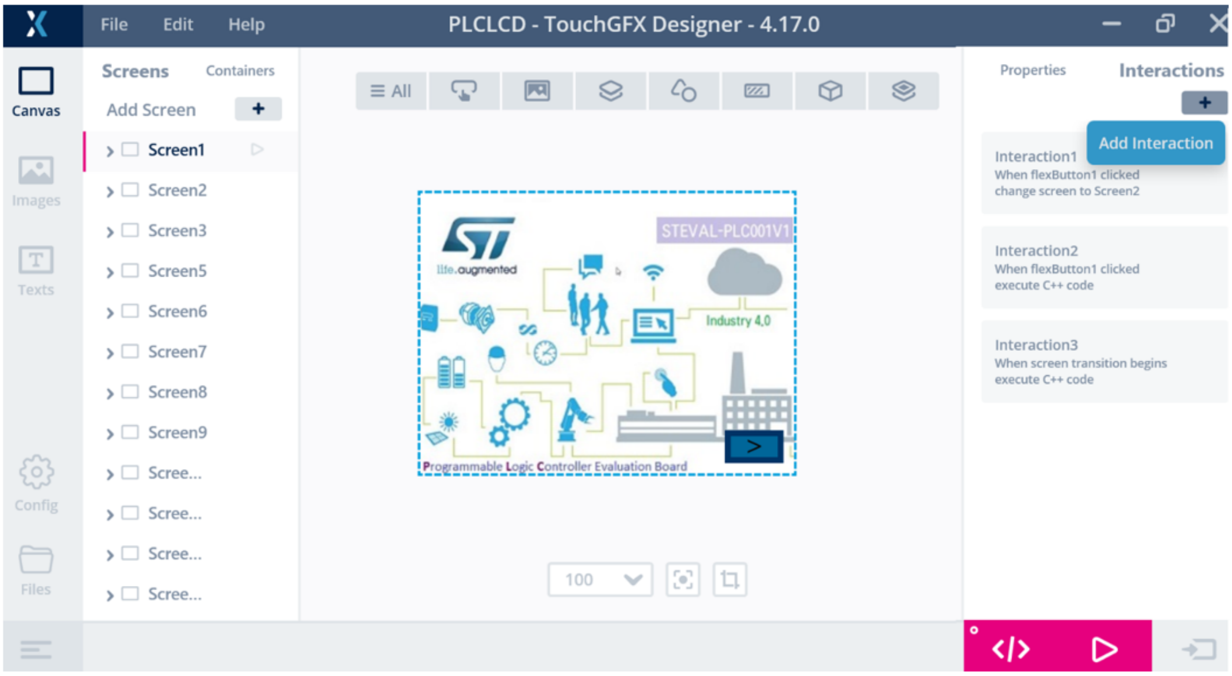
Figure 3. **TouchGFX GUI**

TouchGFX technology main features are:

- smartphone animations (swipe, scroll, 3D effects, video playback, etc.)
- transparency, alpha-blending, antialiased fonts and kerning
- instant interaction from touch display or hard-keys
- easy programing
  - drag-and-drop with automatic code generation
  - more than 30 widgets (swipe container, animated image, shapes, clock, scroll list, etc.)
  - custom triggers and actions
  - easy addition of your own C++ developer code for the creation of a unique UI application
  - support for a variety of IDEs such as IAR Systems Embedded Workbench®, Arm® Keil® MDK, and GCC-based IDEs such as STM32CubeIDE
  - interface for any C code using the Model-View-Presenter pattern
- minimum CPU load and footprints
  - based on STM32 graphic hardware accelerators
  - optimized library footprints requiring at least 20 kbytes of Flash memory and 10 kbytes of RAM
  - partial frame buffering, which minimizes graphic buffer sizes and enables graphic UI support for the STM32 microcontrollers based on the Arm® Cortex®-M0+, M3, M4, M33, and M7 processors
- fast start of UI development
  - STM32 display kits are fully supported in TouchGFX Designer
  - TouchGFX Generator helps users start their own project using STM32 hardware
  - reference demo examples
- part of STM32 ecosystem
  - TouchGFX Engine compiled library runs on any STM32 microcontroller
  - smooth interoperability with STM32Cube MCU packages, STM32CubeMX and STM32CubeIDE with associated project examples

The graphic display is designed using TouchGFX Designer. On-board RAM and Flash memory enhance the display performance and can store multiple display screens. The touch-screen resolution is 800x480 whereas LCD screen resolution is 320x240. Thus, scaling is applied to find the touch corresponding to the point on LCD.

Figure 4. TouchGFX development environment for STEVAL-PLC001V1



Interaction1
When flexButton1 clicked
change screen to Screen2

Interaction2
When flexButton1 clicked
execute C++ code

Interaction3
When screen transition begins
execute C++ code

Figure 5. STEVAL-PLC001V1 board with the display screen created in TocuhGFX environment

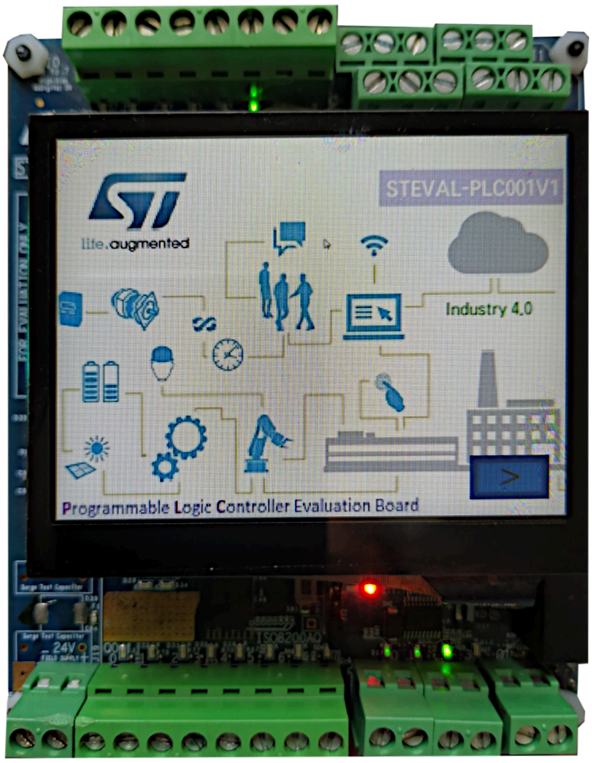
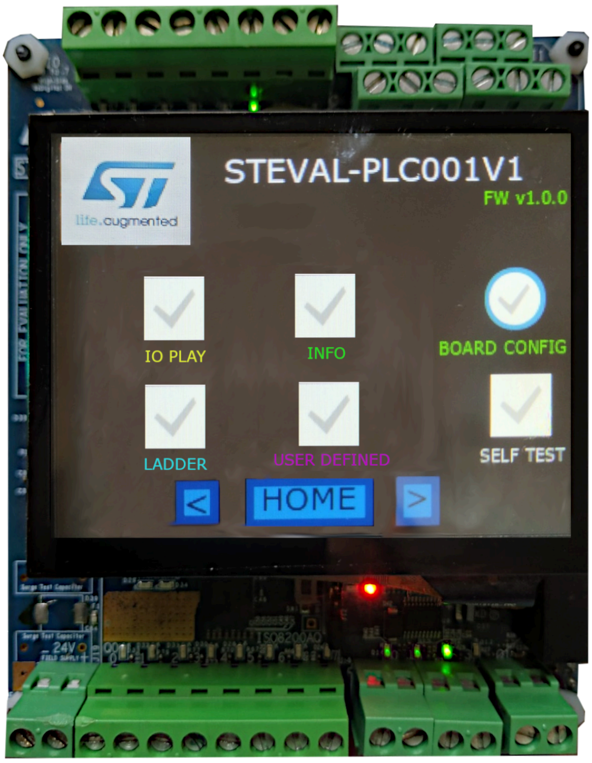


Figure 6. **STEVAL-PLC001V1 application home**



2.5 Timers and ladder logic implementation

2.5.1 Timers

Predefined timers are designed for easy use in ladder logic. However, as they are generic, you can use them in any user application.

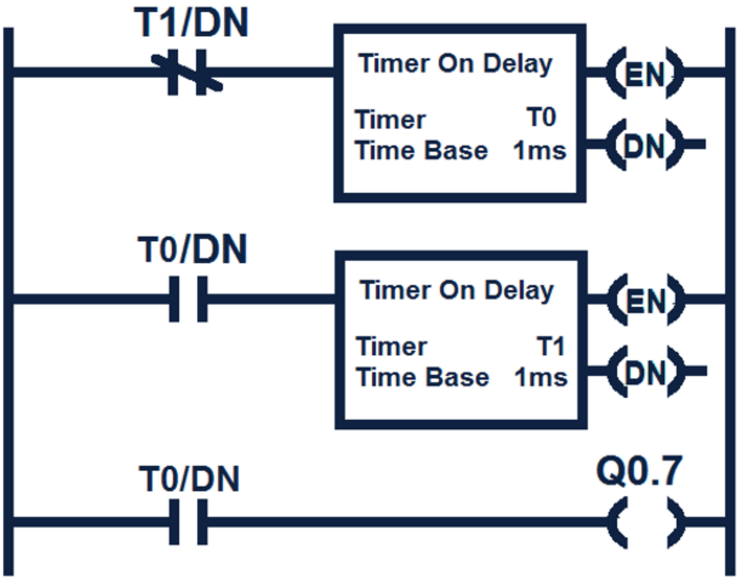
Table 2. **Timer key characteristics**

Timer type	Description
On-delay timer	When you enable the rung of the PLC ladder logic in which the timer is placed, the on-delay timer starts counting. Once the timer count reaches the preset value, the timer done bit is set. If you disable the rung, the timer resets, that is, the done bit and count value is set to 0. On-delay timer is used where the delay is immediately required during the operation. For example, turning the machine on after a fixed time delay, when the Start button is pressed.
Off-delay timer	When you disable the rung in the PLC ladder logic in which the timer is placed, the off-delay timer starts counting. Once the timer count reaches the preset value, the timer done bit is set to 0. If you enable the rung again, the timer count value is set to 0 and the done bit is set to 1. Off-delay of timer is used where the delay is required after the completion of a task. For example, running a cooler fan for several minutes after the machine has stopped.
Retentive timer (RTO)	RTO timer is similar to an on-delay timer. The only difference is that it retains an accumulated value even after you have disabled the rung in which it is placed. When you enable the rung again, the timer starts counting from its retained accumulated value. It requires manual reset to clear the accumulated value. RTO timer could be in applications like paper mills or food processing plants, where it might be necessary to stop the whole process for some time and then start it again from the halted state.

2.5.2 Ladder logic implementation

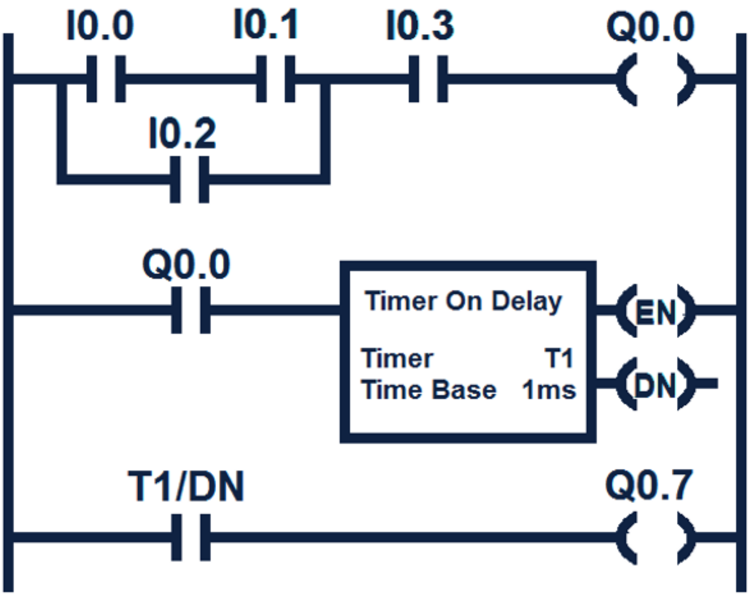
The figure below shows the ladder logic of the output channel toggle, which is the default example in the firmware. No external wiring is required. DO Q0.7 LED blinks due to the periodic toggle in the output channel.

Figure 7. **Output toggle ladder example included in the firmware**



Two complementary on-delay timers are used. This ladder logic is implemented by describing the ladder structure in the form of an array of structures. To illustrate this convention, the figure and the code below show this structure with the help of another ladder diagram.

Figure 8. **Array of structures**



```
int ST_PLC_Ladder [9] [7] =
{
{0, 0, 1, digitalInput,  DI_Channel_I0_0, 0,        XIC  },
{0, 1, 2, digitalInput,  DI_Channel_I0_0, 1,        XIC  },

{0, 0, 2, digitalInput,  DI_Channel_I0_0, 2,        XIC  },
{0, 2, 3, digitalInput,  DI_Channel_I0_0, 3,        XIC  },
{0, 3, 4, digitalOutput, DO_Channel_Q0_0, 0,        0    },
{1, 0, 1, outputStatus,  DO_Channel_Q0_0, 0,        XIC  },
{1, 1, 2, timerOnDelay,  T0,              2000,     0    },
{2, 0, 1, timerStatus,   T0,              timerDone XIC  },
{2, 1, 2, digitalOutput, DO_Channel_Q0_7, 0,        0    },
};
```

- The first column indicates the rung number, starting from 0 (that is, 0, 1, 2, etc.).
- The second column indicates the element node from the user-defined left node of the ladder diagram element. Element node from number should be lower than Element node to number.
- The third column indicates the element node to the user-defined right node of the ladder diagram element.
- The fourth column indicates the element type, which can be digitalInput, digitalOutput, outputStatus, timerStatus, timerOnDelay, timerOffDelay, timerRTO.
- The fifth column indicates the IO / timer identifier, which can be DO_Channel_I0_X for digitalInput, DO_Channel_Q0_X for digitalOutput, or user-defined ID for timer. DO.x and DI.x are physical channels on the board.
- The sixth column indicates the element configuration, which can be timerEnable/timerDone for type timerStatus, preset in milliseconds. For type timerOnDelay/timerOffDelay/timerRTO, it should be set to 0.
- The seventh column indicates PLC contractor type, which can be XIO/XIC for digitalInput, outputStatus and timerStatus element.

ST_PLC_ExecuteLadder() call executes a defined ladder structure. The following sections detail definitions, enumerations, structures, and functions for ladder logic implementation.

3 User API

This is the top-level API visible to the applications through the *stevalplc.h* header file and is a combination of:

- constants that can be:
  - defined: safe to modify, shown with a grey background but in this case the firmware needs to be rebuilt;
  - enumerated: do not reflect the actual register bit positions of devices (although some may match purely by coincidence);
- data structures: can be used entirely or as elements only by some devices, so it is important to note which input or output device uses that part of the data structures;
- functions: a function call might behave slightly different and/or return different error codes depending on which device (and associated set of drivers) is opened.

3.1 Constants

Table 3. Defined constants

Constant	Description
`ST_PLC_FW_VERSION`	Concatenated definition for ST_PLC firmware Major+Minor+Patch versions
`Nb_Total_DI_Channels`	No. of digital input channels on the STEVAL-PLC001V1 board
`Nb_Total_DO_Channels`	No. of digital output channels on the STEVAL-PLC001V1 board
`Nb_Total_DI_Module`	No of digital input modules on the STEVAL-PLC001V1 board
`Nb_Total_DO_Module`	No. of digital output modules on the STEVAL-PLC001V1 board
`ST_PLC_MaxTimers`	Maximum number of PLC timers that can be created
`TotalRungs`	Total number of rungs in ladder logic
`TimCount`	Total number of timers in ladder logic
`TotalElements`	Maximum number of elements in one rung

Table 4. Enumerated constants

Enumeration type	Purpose
`Module_ID`	Identity of a module in the board
`DI_Channel_Name`	Identifies a digital input from the two input modules (i.e., one of 12 (8+4) input channels)
`DO_Channel_Name`	Identifies a digital output from the two output modules having 8+4 channels
`DO_Fault_Name`	Identifies the fault(s) in the two-output module having 6 and 2 fault pins, respectively
`IO_Devices_Name`	It is used to select the IO devices in the STEVAL-PLC001V1 board
`ST_PLC_Err_Code`	It defines the error code used to return to a calling function
`Debug_LedNo`	It is used to select the Debug LED
`DebugLed_State`	It is used to select the state of Debug LED (ON/OFF/REVERSE)
`LCDBackLight_Intensity`	Used to set LCD back-light intensity
`SerialInterface_Config`	Describes exact configuration of the serial peripheral interface used for a device
`ST_PLC_TimerType`	Identifies the type of timer (on-delay, off-delay or retentive on-delay)
`ST_PLC_ElementType`	Identifies the type of element in ladder logic

RELATED LINKS

A.1 Constants on page 14

3.2 Data structure

Table 5. **List of data structures**

Data structure name	Description
IODeviceDescriptor	Contains device name and description of all its interfaces
Module_Status_Map	Stores the status of a device IO channel and its faults as bit field
Module_Status	Holds input/output value and fault status of an IO module. The array (IO_ModuleStatus_Fault[]) of this structure acts as the central repository of input or output status of a module and its fault(pin) status in a predefined bitmap
ST_PLC_Timer	Contains timer handle, ID, preset value, type, done and enable bits, accumulated value for RTO and user-defined callback function
ST_PLC_RungStructure	Contains the rung properties

3.3 Functions

Table 6. **List of functions supported by ST_PLC API**

Function	Description
ST_PLC_Read_AllChannels_InModule	Reads the data from all channels in one of the two input modules
ST_PLC_Write_AllChannels_InModule	Writes the desired 8-bit/4-bit data to the output module
ST_PLC_SetOne_DO	Sets the selected output channel
ST_PLC_ClrOne_DO	Clears the selected output channel
ST_PLC_GetOne_DI	Reads the digital input of each individual channel of the selected input module
ST_PLC_GetOne_DO_Status	Returns the boolean status of a DO channel from memory
ST_PLC_Update_ModuleFaultStatus	Reads the fault status of the chosen module and updates the IO_ModuleStatus_Fault array indexed by Module_ID
ST_PLC_EnableModule	Enables the modules when needed. Applicable for devices with hardware enable pins
ST_PLC_DisableModule	Disables the module. Applicable for devices with hardware enable pins
ST_PLC_Set_HMI_Backlightintensity	Controls the intensity of the LCD display back-light
ST_PLC_Blink_One_DebugLed	Makes one identified debug LED blink for the desired number of time and duration
ST_PLC_Control_DebugLed	Controls the identified debug LED as per parameter supplied (on, off or reverse)
ST_PLC_InitiatizeSTEval	Initializes the board
STEVALPLC_GetFWVersion	Gets firmware version
ST_PLC_TimerCallback	FreeRTOS timer callback
ST_PLC_CreateTimer	Creates a FreeRTOS timer
ST_PLC_StartTimer	This function is used internally to start the timer and should not be called directly
ST_PLC_StopTimer	This function is used internally to stop the timer and should not be called directly

Function	Description
`ST_PLC_ChangeTimerPeriod`	Updates the timer preset value
`ST_PLC_EnableTimer`	Starts or stops the timer depending on its type
`ST_PLC_DisableTimer`	Stops or starts the timer depending on its type
`ST_PLC_RTOReset`	Resets the RTO timer
`ST_PLC_InitializeLadder`	Initializes ladder elements (timer creation, etc.)
`ST_PLC_StartExecution`	Starts ladder execution
`ST_PLC_ExecuteLadder`	Single function to call the `startLadder` execution. This function internally calls `ST_PLC_InitializeLadder` (if required) and `ST_PLC_ExecuteLadder`

Table 7. **List of utility functions**

Function	Description
`ST_PlC_UtilIsNthBitSet`	Checks whether a specific 8-bit data is set
`ST_PLC_Util_ReverseBits`	Interchanges MSb with LSb of 8-bit number
`ST_PLC_UtilsetBitLeft`	Shifts 8-bit data to the left for desired number of times
`ST_PLC_UtilSetBitRight`	Shifts 8-bit data to the right for desired number of times

RELATED LINKS

A.2 Functions on page 16

A.2.1 Utility functions on page 22

4 Demonstration application and use cases

On powering up the board, a user input is expected for the selection of a use case.

The demonstration application implements few simple use cases. You can select one of the following use-cases through the touchscreen interface:

- IO Play: digital outputs (DO) mimic that correspond to digital inputs (DI) as per board symmetry. The IO status is displayed on screen as well as Fault;
- Info: board information and device details are shown. This mode can work even with 5 V supply to USB connector;
- Ladder Logic: simple ladder examples as described in Section 2.5.2 ;
- Self-test: to test board key components and firmware functions as well as RAM, Flash, and touch screen and IO modules. For the IO modules, loop-back connections are needed to test each input and output channel with predefined patterns. Q0.x has to be connected to the corresponding IO.x;
- User defined: contains user-defined use cases. This mode can be used to add code for the user-defined logic. The user can optionally update the display screen to suit a specific application.

Appendix A ST_PLC API reference

The following sections provide detailed information on some of the important definitions, structures, constants, and functions used in the User API. Refer to source code for more information.

A.1 Constants

Module_ID

It indicates the unique identity of modules on the STEVAL-PLC001V1 board.

```
enum Module_ID {
                 Module_ID_STM32F746,
                 Module_ID_CLT01_38SQ7,
                 Module_ID_CLT03_2Q3_X_2,
                 Module_ID_ISO8200AQ,
                 Module_ID_IPS4260L,
                 Moduel_ID_UART_GEN,
                 Module_ID_STLD40DPUR,
                 Module_ID_RK035HQ02_CT814B,
                 Module_ID_MAX
                  };
```

DI_Channel_Name

It selects the digital input from the two input modules having 8+4 channels.

```
enum DI_Channel_Name {
      DI_Channel_I0_0 = (Module_ID_CLT01_38SQ7<<8) | (0x01),
      DI_Channel_I0_1 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<1),
      DI_Channel_I0_2 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<2),
      DI_Channel_I0_3 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<3),
      DI_Channel_I0_4 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<4),
      DI_Channel_I0_5 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<5),
      DI_Channel_I0_6 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<6),
      DI_Channel_I0_7 = (Module_ID_CLT01_38SQ7<<8) | (0x01<<7),

      DI_Channel_I1_0 = (Module_ID_CLT03_2Q3_X_2<<8) | (0x01),
      DI_Channel_I1_1 = (Module_ID_CLT03_2Q3_X_2<<8) | (0x01 <<1),
      DI_Channel_I1_2 = (Module_ID_CLT03_2Q3_X_2<<8) | (0x01 <<2),
      DI_Channel_I1_3 = (Module_ID_CLT03_2Q3_X_2<<8) | (0x01 <<3)

      };
```

DO_Channel_Name

It selects the digital output from the two output modules having 8 and 4 channels, respectively.

```
enum DO_Channel_Name {
      DO_Channel_Q0_0 = (Module_ID_ISO8200AQ<<8) | (0x01),
      DO_Channel_Q0_1 = (Module_ID_ISO8200AQ<<8) | (0x01<<1),
      DO_Channel_Q0_2 = (Module_ID_ISO8200AQ<<8) | (0x01<<2),
      DO_Channel_Q0_3 = (Module_ID_ISO8200AQ<<8) | (0x01<<3),
      DO_Channel_Q0_4 = (Module_ID_ISO8200AQ<<8) | (0x01<<4),
      DO_Channel_Q0_5 = (Module_ID_ISO8200AQ<<8) | (0x01<<5),
      DO_Channel_Q0_6 = (Module_ID_ISO8200AQ<<8) | (0x01<<6),
      DO_Channel_Q0_7 = (Module_ID_ISO8200AQ<<8) | (0x01<<7),

      DO_Channel_Q1_0 = (Module_ID_IPS4260L<<8) | (0x01),
      DO_Channel_Q1_1 = (Module_ID_IPS4260L<<8) | (0x01 <<1),
      DO_Channel_Q1_2 = (Module_ID_IPS4260L<<8) | (0x01 <<2),
      DO_Channel_Q1_3 = (Module_ID_IPS4260L<<8) | (0x01 <<3)

      };
```

DI_Fault_Name

It indicates the unique identity of the available fault pins/bits of the modules.

```
enum DI_Fault_Name {
      DI_Channel_FLT_Q1_0 = (Module_ID_IPS4260L<<8) | (0x01),
      DI_Channel_FLT_Q1_1 = (Module_ID_IPS4260L<<8) | (0x01<<1),
      DI_Channel_FLT_Q1_2 = (Module_ID_IPS4260L<<8) | (0x01<<2),
      DI_Channel_FLT_Q1_3 = (Module_ID_IPS4260L<<8) | (0x01<<3),
      DI_Channel_FLT_Q1_FLT = (Module_ID_IPS4260L<<8) | (0x01<<4),
      DI_Channel_FLT_Q1_OL =(Module_ID_IPS4260L<<8) | (0x01<<5),

      DI_Channel_Q0_PGOOD = (Module_ID_ISO8200AQ<<8) | (0x01),
      DI_Channel_Q0_FLT =(Module_ID_ISO8200AQ<<8) | (0x01 <<1),
};
```

IO_Devices_Name

It selects the IO devices in the STEVAL-PLC001V1 board.

```
enum IO_Devices_Name {
                 IN_CLT01_38SQ7,
                 IN_CLT03_2Q3_Array,
                 OUT_ISO8200AQ,
                 OUT_IPS4260L,
                 IO_UART_GEN ,
                 UI_RK035HQ02_CT814B,
                 XY_NAME_MAX =0xFF
                  };
```

ST_PLC_Usecase

It selects the use-case: DIDO is a simple use-case where the output channels are replicated with the corresponding input channel; DODI is intended for self-test, that is each digital output feedback is linked to the corresponding DI to check the board; Info shows information on the board display.

```
enum ST_PLC_Usecase {
                      ST_PLC_DIDO,
                      ST_PLC_Info,
                      ST_PLC_Info_DIDO,
                      ST_PLC_LadderLogic,
                      ST_PLC_DODI_Selftest,
                      ST_PLC_Userdefined
                      };
```

ST_PLC_Err_Code

It defines the error code used to return a particular type in case of an error in the board.

```
enum ST_PLC_Err_Code {
                     ST_PLC_NO_ERROR,
                       ST_PLC_ERROR_HW_FLT,
                       ST_PLC_ERROR_WrongParameter,
                       ST_PLC_ERROR_UnexpectedOutcome,
                       ST_PLC_ERROR_SPI,
                       ST_PLC_ERROR_Undefined,
                       ST_PLC_ERROR_FeatureNotSupported
                    };
```

Debug_Led

It selects the debug LED.

```
Enum Debug_Led {
               DebugLed1,
               DebugLed2,
               DebugLed3
               };
```

DebugLed_State

It selects the state of the debug LED (ON/OFF/REVERSE).

```
Enum Debug_Led {
                    DebugLed_ON,
                    DebugLed_OFF,
                    DebugLed_REVERSE
                   };
```

LCDBackLight_control

It selects the intensity of the LCD display.

```
Enum LCDBackLight_control {
                          BackLight_Reset,
                          BackLight_25,
                          BackLight_50,
                          BackLight_75,
                          BackLight_100,
                        };
```

ST_PLC_TimerType

It selects the type of PLC timer.

```
enum ST_PLC_TimerType {
ST_PLC_OnDelayTimer,
                          ST_PLC_OffDelayTimer,
                          ST_PLC_RTOTimer
};
```

ST_PLC_ElementType

It selects the type of PLC rung element.

```
enum ST_PLC_ElementType {
digitalInput,
digitalOutput,
timerStatus,
outputStatus,
timerOnDelay,
timerOffDelay,
timerRTO,
timerEnable,
timerDone,
XIO,
XIC
};
```

RELATED LINKS

3.1 Constants on page 10

A.2 Functions

ST_PLC_Read_AllChannels_InModule()

It reads data from either of the two-input modules.

```
ST_PLC_ErrorCode ST_PLC_Read_AllChannels_InModule
(
              uint8_t ModuleName,
              uint8_t *InData
);
```

List of arguments

- `ModuleName`: used to call the input module whose data need to be read
- `Indata`: pointer to the data to be read from the module

List of return values

- `ST_PLC_NO_ERROR`: if either of the input modules is selected, it is returned
- `ST_PLC_ERROR_WrongParameter`: if any other module different from the input module is selected, it is returned

Call sequence

This API function is called after board initialization.

ST_PLC_Write_AllChannels_InModule()

It writes the desired 8 bit/4 bit data in the output module.

```
ST_PLC_ErrorCode ST_PLC_Write_AllChannels_InModule
(
        uint8_t Module_ID,
        uint8_t Data
);
```

List of arguments

- `Module_ID`: selects the output module where the data need to be written
- `Data`: data to be sent to the output module

List of return values

- `ST_PLC_NO_ERROR`: if either of the input modules is selected, it is returned
- `ST_PLC_ERROR_WrongParameter`: if any other module different from the input module is selected, it is returned
-

Call sequence

Generally this ST_PLC API function is called when you need to write the 8-bit/4-bit data into the output module.

Comments

This function writes the data in the selected output module.

If the output module is written without any interruption, then it will return the `ST_PLC_NO_ERROR`.

ST_PLC_SetOne_DO()

This function is used to set each individual pin of the selected output module.

```
ST_PLC_ErrorCode ST_PLC_SetOne_DO
(
   uint32_t Channel_Name
);
```

List of arguments

- `ChannelName`: selects the individual output channel

List of return values

- `ST_PLC_NO_ERROR`: if either of the output modules is selected, it is returned
- `ST_PLC_ERROR_WrongParameter`: if any other module different from the output module is selected, it is returned

Call sequence

Generally this ST_PLC API function is called to set the output module individual pins and return the 8-bit/4-bit data.

ST_PLC_ClrOne_DO()

This function is used to clear or reset each individual pin of the selected output module.

```
ST_PLC_ErrorCode ST_PLC_ClrOne_DO
(
   uint32_t Channel_Name
);
```

List of arguments

- `ChannelName`: selects the individual output channel

List of return values

- `ST_PLC_NO_ERROR`: if either of the output modules is selected, it is returned
- `ST_PLC_ERROR_WrongParameter`: if any other module different from the output module is selected, it is returned

Call sequence

Generally this ST_PLC API function is called to clear the output module individual pins and return the 8-bit/4-bit data.

ST_PLC_GetOne_DI()

This function is used to get the digital input of each individual pin of the selected input module.

```
ST_PLC_ErrorCode   ST_PLC_GetOne_DI
(
  uint32_t Channel_Name
);
```

List of arguments

- ChannelName: selects the individual output channel

List of return values

- ST_PLC_NO_ERROR: in case of no error
- ST_PLC_ERROR_WrongParameter: if any other module different from the output module is selected

Call sequence

This function is called after board initialization.

Comments

Channel_Name is used to call the specified channel of a particular module.

GetOne_DI is used for each individual pin of the input channel.

ST_PLC_GetOne_DO(Status)

This function is used to extract the status of a DO channel from module status stored in the memory.

```
ST_PLC_ErrorCode   ST_PLC_GetOne_DO_Status
(uint32_t Channel_Name,
bool *Outstatus );
```

List of arguments

- Channel_Name: identifies the output channel
- Out: Boolean status to indicate on or off

List of return values

- ST_PLC_NO_ERROR: in case of no error
- ST_PLC_ERROR_WrongParameter: if module ID is incorrect

Call sequence

This function is called after board initialization.

ST_PLC_Update_Module_FaultStatus()

This function reads the fault status of the chosen module and updates the `IO_ModuleStatus_Fault` array, which is indexed by `Module_ID`.

```
ST_PLC_ErrorCode ST_PLC_Update_Module_FaultStatus
(
    uint8_t FaultModule,
    uint8_t *Data
);
```

List of arguments

- `FaultModule`: `Module_ID` of the module

List of return values

- `ST_PLC_NO_ERROR`: if the module has no fault, then this status is returned
- `ST_PLC_ERROR_HW_FLT`: when the hardware fault is encountered in any of the modules then this status is returned
- `ST_PLC_ERROR_WrongParameter`: if any other module different from the output module is selected, it is returned

Call sequence

This function can be called after any input or output function where data read/write does not give the desired result.

Comments

Fault module function can be called only for the modules, which have a dedicated fault pin.

It gives information about the module and not about the pin/ input where the fault has occurred.

ST_PLC_EnableModule ()

This function enables the module.

```
ST_PLC_ErrorCode ST_PLC_EnableModule
(
  uint8_t Module_ID
);
```

List of arguments

- `Module_ID`: takes the ID of the module that needs to be enabled

List of return values

- `ST_PLC_NO_ERROR`: if all the modules are enabled, then this status is returned
- `ST_PLC_ERROR_WrongParameter`: if the module cannot be enabled, then this status is returned

Call sequence

This function can be called at start-up or anytime when a module needs to be initialized.

Comments

This function enables the modules, which have a dedicated hardware pin to enable the module.

ST_PLC_DisableModule()

This function disables the modules.

```
ST_PLC_ErrorCode ST_PLC_DisableModule
(
  uint8_t Module_ID
);
```

List of arguments

- `Module_ID`: takes the module that needs to be disabled

List of return values

- `ST_PLC_NO_ERROR`: if all the modules are disabled, then this status is returned
- `ST_PLC_ERROR_WrongParameter`: if the module cannot be disabled, then this status is returned

ST_PLC_Set_HMI_Backlightintensity()

It is used to control the intensity of the back-light of the LCD display.

```
ST_PLC_ErrorCode ST_PLC_Set_HMI_Backlightintensity
(
   uint8_t Freq,
   uint8_t DutyCycle
);
```

List of arguments

- `Freq`: determines the time period
- `DutyCycle`: determines the percentage of the PWM

List of return values

- `ST_NO_ERROR`: the LCD display shows content with various intensities

Comments

This function changes the intensity of the LCD display.

STEVALPLC_GetFWVersion()

This function returns version of the API.

```
uint32_t STEVALPLC_GetFwVersion
(
Void
);
```

List of return values

- `STEVALPLC_FW_VERSION`: returns the current version of firmware

ST_PLC_TimerCallback()

FreeRTOS timer callback. The timer done bit is set or cleared depending on the type of timer.

```
void ST_PLC_TimerCallback
(
              xTimerHandle xTimer
);
```

List of arguments

- `xTimer`: FreeRTOS kernel passes the timer handle

Call sequence

- This function is called when the FreeRTOS timer counting finishes
- The timer done bit is set or cleared depending on the type of timer
- User callback function is called if not set to NULL

Comments

This function is used by FreeRTOS kernel and should not be called directly.

ST_PLC_CreateTimer()

This function creates a FreeRTOS timer.

```
ST_PLC_TimerHandle_t ST_PLC_CreateTimer
(
int preset,
uint8_t type,
void (*func)(ST_PLC_TimerHandle_t)
);
```

List of arguments

- `Preset`: timer preset value in milliseconds
- `Type`: timer type from `ST_PLC_TimerType`
- `void (*func)(ST_PLC_TimerHandle_t)`: a user callback function pointer with a function that has an argument of type `ST_PLC_TimerHandle_t`, which is a pointer to timer structure. This function is called in `ST_PLC_TimerCallback` after the given timer elapses

List of return values

- `ST_PLC_TimerHandle_t`: pointer to timer structure

Call sequence

- This function is called when you need to create a new timer.

Comments

`ST_PLC_MaxTimers` defines the maximum number of timers that can be created.

ST_PLC_InitializeLadder(void)

This function initializes ladder elements (timer creation, etc.).

```
void ST_PLC_InitializeLadder(void);
```

Call sequence

- This function is called from `ST_PLC_ExecuteLadder` if the ladder has not already been initialized.

Comments

This function is used by ladder API function and should not be called directly.

ST_PLC_StartExecution()

This function starts execution of a ladder.

```
void ST_PLC_StartExecution(void);
```

Call sequence

- This function is called from `ST_PLC_ExecuteLadder` function.

Comments

Other functions, such as ladder API, use this function internally.

RELATED LINKS

3.3 Functions on page 11

A.2.1 Utility functions

ST_PLC_util_ReverseBits() This API function is used to return the firmware version of the library.

```
uint8_t ST_PLC_util_ReverseBits
(
   uint8_t num8b
);
```

List of arguments

- `Num8b`: 8-bit data whose sequence needs to be changed

List of return values

- `Integer value`: 8-bit data sequence in reversed form is returned

Comments

This function is generally called when data is written in the output module and the data sequence needs to be reversed.

STEvalPlcUtilIsNthBitSet This API function is used to check if the bit is set or not.

```
bool STEvalPlcUtilIsNthBitSet
(
    uint8_t data,
    uint8_t pos
);
```

List of arguments

- `Data`: value that needs to be checked
- `pos`: value to be read

Return value

Boolean value 1 or 0.

Comments

This function checks if each bit of the data is set or not.

STEvalPlcUtilsetBitLeft This function is used to set 8-bit data to the left.

```
uint8_t STEvalPlcUtilsetBit
(
     uint8_t tmp_data,
     uint8_t pos,
     bool bit
);
```

List of arguments

- `Tmp_data`: value that needs to be set
- `pos`: position of the bit
- `bit`: 1 or 0

Return value

It returns the 8-bit data that is shifted to the left by specified bits.

Comments

Utility function

STEvalPlcUtilSetBitRight

This function is used to set 8-bit data to the right.

```
uuint8_t STEvalPlcUtilsetBit
(
     uint8_t tmp_data,
     uint8_t pos,
```

List of arguments

- `Tmp_data`: value that needs to be set
- `pos`: position of the bit

Return value

It returns the 8-bit data that is set to the right.

Comments

This function sets the 8-bit data on the right.

Blink_One_DebugLed

This function is used to make one LED blink.

```
void Blink_One_DebugLed
(
   uint8_t DebugLed,
   uint32_t delay,
   uint8_t repeat
 );
```

List of arguments

- `DebugLed`: selects the debug LED that needs to blink
- `Delay`: gives delay
- `Repeat`: calculates the number of times the LED blinks

Return value

None.

Comments

Selects the debug LED that needs to blink.

Blink_Debug_LED2

This function is used to make the debug LED blink as soon as the board starts.

```
void Blink_Debug_LED2
(
   uint32_t delay,
   uint8_t repeat
);
```

List of arguments

- `Delay`: gives delay
- `Repeat`: calculates the number of times the LED blinks

Return value

None.

Comments

Selects the debug LED that needs to blink.

Control_DebugLed

This function is used to control the debug LED.

```
void Blink_One_DebugLed
(
   uint8_t DebugLed,
   uint8_t repeat
 );
```

List of arguments

- `DebugLed`: selects the debug LED that needs to blink
- `Repeat`: intended number of LED blinks

Return value

None.

Comments

Selects the LED that needs to blink.

RELATED LINKS

3.3 Functions on page 11

Revision history

Table 8. **Document revision history**

Date	Revision	Changes
25-Oct-2021	1	Initial release.

Contents

List of tables

List of figures

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.