Getting started with the STSW-ETHDRV01V1 firmware for servo drive EtherCAT solution

Introduction

The STSW-ETHDRV01V1 firmware package for the STEVAL-ETH001V1 servo drive solution has been built around the STM32F767ZI microcontroller. It implements a position control algorithm using the X-CUBE-MCSDK motor control library (V.5.4.4) to control a PMSM motor rotor position via EtherCAT communication remote control.

Servo drive actuation and digital input/output interface can be managed at the same time.

The connectivity features real-time communication with EtherCAT protocol stack (V. 5.0.8) for slave node and RS485 communication to interface the hardware with a PC or digital encoder supporting BiSS and EnDat protocols.

1 Overview

The STSW-ETHDRV01V1 firmware has been developed using IAR Workbench 8.50 and is compliant with the STM32Cube framework. The key features are:

- Position control algorithm based on X-CUBE-MCSDK (V.5.4.4)
- Supported EtherCAT slave protocol (V.5.0.8)
- Firmware compliant with STM32Cube framework
- BSP support for digital actuation interface
- RS485 interface support

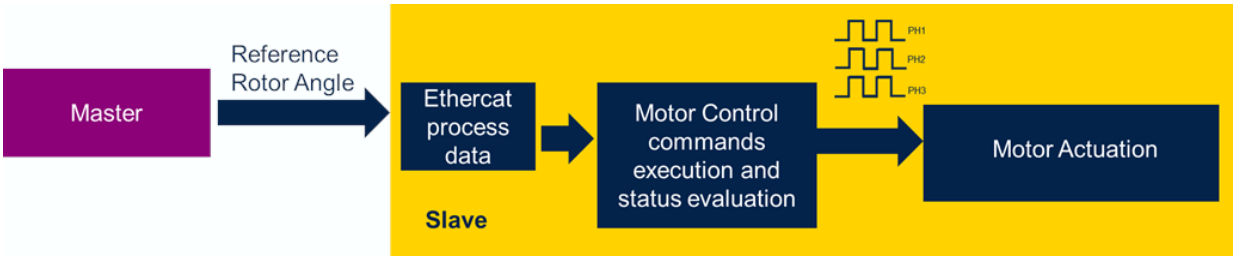
The communication topology is master-slave: the master (not part of this development) can be represented by dedicated hardware or a dedicated software tool like TwinCAT, whereas the slave is implemented by the STEVAL-ETH001V1 evaluation board.

The firmware is able to manage all the blocks included in the hardware solution at the same time, with particular focus on connectivity and motion control.

This firmware section allows a real-time handling of a PMSM motor, implementing a master-slave communication based on the EtherCAT protocol; in particular, the STEVAL-ETH001V1, working as a slave node, receives a data streaming composed of a set command and the rotor position by the master node built with the TwinCAT software tool in this application use case.

Once the slave receives the data, a processing phase is activated on the microcontroller side to modulate the motor drive signal, according to the data and command received.

Figure 1. Motor control process flow



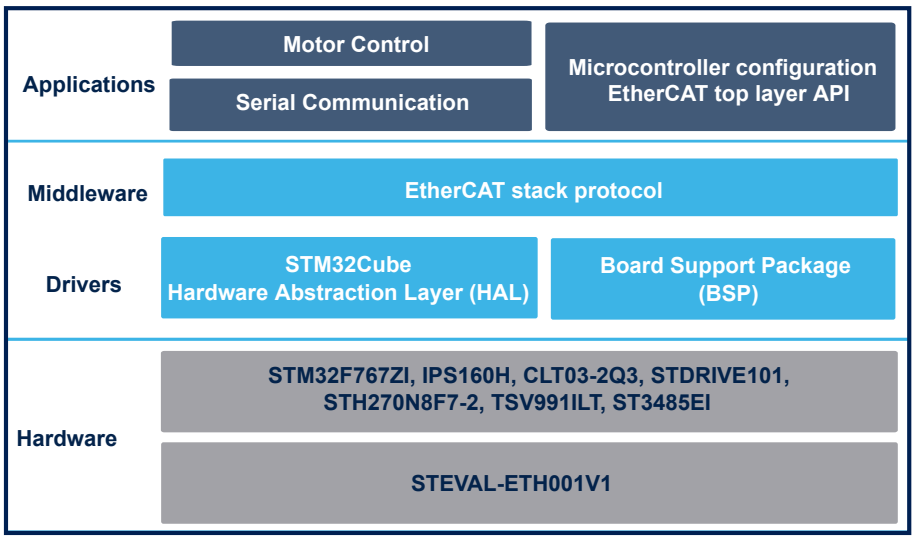
2 Layers

The firmware structure consists of Application, Middleware and Drivers layers.

Each layer is split into different groups, containing the source files for EtherCAT APIs, motor control library and user interface, detailed as follows:

- **Application**
  - User: STM32 configuration, interrupt service routine and motor control APIs
  - CifxApplication: APIs for application layer
  - Protocol: low level APIs for protocol management
- **Middleware**
  - CifxToolkit: protocol stack APIs
  - X-CUBE-MCSDK: motor control library (for further details, refer to UM2392 and UM2374 freely available on www.st.com)
- **Drivers**
  - BSP: the board support package drivers are part of the STM32Cube MCU and MPU packages based on the HAL drivers and provide a set of high-level APIs related to the hardware components and features of the evaluation boards, Discovery kits and STM32 Nucleo boards (for further details about BSP, refer to UM2298 on www.st.com)
  - CMSIS: Cortex microcontroller software interface standard
  - STM32F7xx_HAL_Library

Figure 2. STSW-ETHDRV01V1 architecture



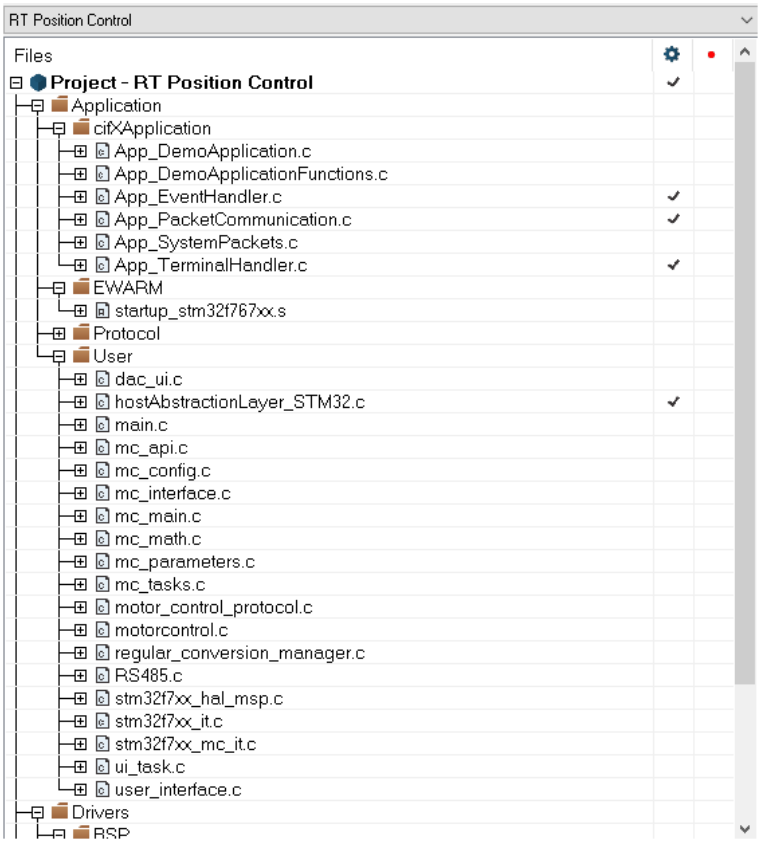
2.1 Application

This layer consists of a set of routines to manage the EtherCAT protocol application and the user application layers.

In particular, as shown in the picture below, it embeds:

- the EtherCAT blocks with the **cifXApplication** routines for all settings to properly manage the communication and data packets, and with the **Protocol** routines to configure the protocol
- the **User** routines for peripheral configuration, top level motor control library management, digital I/O and RS485 management

Figure 3. **Application folder**



2.1.1 EtherCAT protocol

2.1.1.1 cifXApplication

The cifXApplication folder contains all the routines to manage cyclic and acyclic communication.

The acyclic communication manages the event and uses the mailbox channel; it is developed in the "App_PacketCommunication.c", "App_EventHandler.c", and "App_SystemPacket.c" source files.

The cyclic communication manages the process data between master and slave nodes. Moreover, it enables the data channel read/out procedure. It is developed in the "App_DemoApplication.c" and "App_DemoApplicationFunctions.c" source files.

2.1.1.1.1 App_DemoApplication.c

This source file contains the *App_CifxApplicationDemo* application routine called in the main.c after the configuration phase. The routine calls all the subroutines operating at communication level to run the application.

This section must contain all the required user function calls.

Figure 4. **User function call example**

```
/** now the bus is running */
while(tAppData.fRunning && lRet == CIFX_NO_ERROR)
{
  /** check and process incoming packets */
  lRet = Protocol_PacketHandler(&tAppData);

 /** motor control routine */

  main_mc_loop();   <-- Motor control routine


  /** check and process input from terminal (UART console) */
//  App_TerminalHandler(&tAppData);

  /** check for events (DIP switches, changed variable values, etc.) */
//  App_EventHandler(&tAppData);

  HOSTAL_Sleep(1);
}
```

The *main_mc_loop()* routine is defined in the "mc_main.c" source file of the application layer.

The data for inputs and outputs are continuously updated on the bus according to the time scheduler. These buffers are organized according to the structure definition shown in the code below which includes all the motor control commands that a master can exchange with the EtherCAT nodes as well as the feedback coming from the nodes.

These structures are defined in the " App_demoApplication.h " file and can be modified manually according to the master frame protocol, if it is different from the provided example.

```
typedef __HIL_PACKED_PRE struct APP_PROCESS_DATA_INPUT_Ttag
{
uint8_t start;
uint8_t commandID;
float position;
float duration;
} __HIL_PACKED_POST APP_PROCESS_DATA_INPUT_T;

typedef __HIL_PACKED_PRE struct APP_PROCESS_DATA_OUTPUT_Ttag
{
uint8_t done;
uint8_t status;
uint8_t actuation_CH1run;
uint8_t actuation_CH2run;
uint8_t actuation_CH1status;
uint8_t actuation_CH2status;
} __HIL_PACKED_POST APP_PROCESS_DATA_OUTPUT_T;
```

Table 1. **Input/output frame structure variables**

Variable	In/out	Type	Description
start	In	uint8_t	This byte is set to 1 when a new command is sent by the master
commandID	In	uint8_t	The code of the command sent (see Section 5)
position	In	float	The target motor position expressed in radians
duration	In	Float	The duration of the last motor movement command
done	Out	uint8_t	This byte is set to 1 by the slave indicating that the command has been received
status	Out	uint8_t	This byte contains the current state code indicated by the slave (see section 4 for details)

Variable	In/out	Type	Description
CH1run	Out	uint8_t	This value represents the CH1 output state (ON/OFF)
CH2run	Out	uint8_t	This value represents the CH2 output state (ON/OFF)
CH1status	Out	uint8_t	This value represents the CH1 output status (GOOD/FAULT)
CH2status	Out	uint8_t	This value represents the CH2 output status (GOOD/FAULT)

2.1.1.2 *Protocol*

The protocol folder contains all the routines called by the user and by cifxApplication to configure the stack and manage the package transmission and reception through data channels.

2.1.1.3 *User*

This folder contains a set of routines to provide the user with a first access level to interact with the firmware.

The user can:

- call functions for EtherCAT protocol configuration and network controller type detection
- properly configure the STM32 peripherals and GPIOs for the motor control section
- handle RS485 data transmission

At this level, it is possible also to modify the scheduler settings, implemented to properly manage the EtherCAT data packet, by accessing the *hostAbstractionLayer_STM32.c* and changing the STM32 timer *prescaler* and/or the *period* parameters.

2.1.1.3.1 hostAbstractionLayer.c

To modify time scheduler and change communication speed, you have to modify the *TimHandle.Init.Prescaler* or the *TimHandle.Init.Period* to meet the communication speed requirement.

For example, a new time scheduler is set to 5 ms. To get a prescaler value (*uwPrescalerValue)*) lower than 65535, the timer period value (which corresponds to the auto-reload register (ARR) value in the microcontroller timer) has to be calculated as follows:

$$uw\mathrm{Pr}\,escalerValue = SystemCoreClock/Period/Timfreq - 1$$

considering that

$$Period = 100$$
$$Timfreq = 200Hz$$
$$uw\mathrm{Pr}\,escalerValue = 10800$$

and the system core clock is equal to 216 MHz.

The firmware time scheduler default value is 1 ms; the picture below shows the timer parameter settings.

Figure 5. **Time scheduler configuration**

```
92  HOSTAL_RESULT_E HOSTAL_Init(void){
93
94
95          /*setup timer*/
96          /*##-1- Configure the TIM peripheral ###################################
97          /* ------------------------------------------------------------------
116
117         /* Compute the prescaler value to have TIMx counter clock equal to 2kHz
118
119         uint16_t Period=10;
120         uint16_t Timfreq=2000;
121
122
123         /* Set TIMx instance */
124         TimHandle.Instance = nTIMx;
125
126         /* Initialize TIMx peripheral as follows:
127                + Period = 10
128                + Prescaler = SystemCoreClock/Period/Timfreq - 1
129                + Timfreq =2000 Hz
130                + ClockDivision = 0
131                + Counter direction = Up
132          */
133
134         uwPrescalerValue = (uint32_t)((SystemCoreClock/ Period/Timfreq) - 1);
135
136         TimHandle.Init.Period            = Period;
137         TimHandle.Init.Prescaler         = uwPrescalerValue;
138         TimHandle.Init.ClockDivision     = 0;
139         TimHandle.Init.CounterMode       = TIM_COUNTERMODE_UP;
140         TimHandle.Init.RepetitionCounter = 0;
141
142         if (HAL_TIM_Base_Init(&TimHandle) != HAL_OK)
143         {
144                 /* Initialization Error */
145                 Error_Handler();
146         }
```

2.2 Middleware

This layer contains the motor control library and the library to implement the real-time EtherCAT protocol.

2.2.1 EtherCAT protocol stack routines

The protocol stack routines are grouped in the cifXToolkit and organized in subfolders per functionality:

- OSAbstraction
- SerialDPM
- Source
- User

2.2.1.1 OSAbstraction

The operating system abstraction C module (OSAbstraction) can be used to exploit embedded RTOS services, such as memory management or synchronization mechanisms. Moreover, it supports bare metal applications without an OS as in the case of STSW-ETHDRV01V1 firmware. The cifXToolkit middleware component calls the OSAbstraction functions also to manage the SPI initialization and communication between NETX device and STM32.

2.2.1.2 Serial DPM

This layer consists of a set of routines, implemented to handle the communication with different companion IC types belonging to the Hilscher NETX family (NETX10, NETX50, NETX500, NETX51).

The NETX90 communication hardware is the same as the NETX51, so it can use the same communication functions. For this reason, there are no dedicated communication routines for NETX90.

2.2.1.3 *Source*

This layer contains all low-level functions used to manage the communication.

2.2.1.4 *User*

This section contains a set of APIs that can be used to get different information.

2.2.2 Motor control library

The motor control library is a set of source files to manage the field-oriented control of the permanent magnet synchronous motor (PMSM).

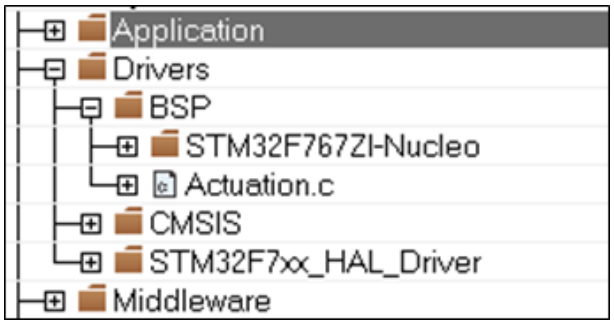
All the required functions related to this type of control are implemented without requiring any modification. The main functions are:

- motor phase current sensing
- rotor position sensing
- torque reference generation based on speed or position control loop
- current control loop based on PI regulators (torque control)
- PWM signal generation required by the inverter

2.3 Drivers

The Drivers layer includes the STM32Cube package libraries (CMSIS and STM32F7xxHAL_Diver) and the BSP routines available in the Actuation.c to manage the digital I/O stage (IPS160H and current limiter).

Figure 6. Drivers folder



2.3.1 Actuation.c

The Actuation.c file contains a set of routines to manage the output stage powered by IPS160H and the digital input powered by CLT03-2Q3.

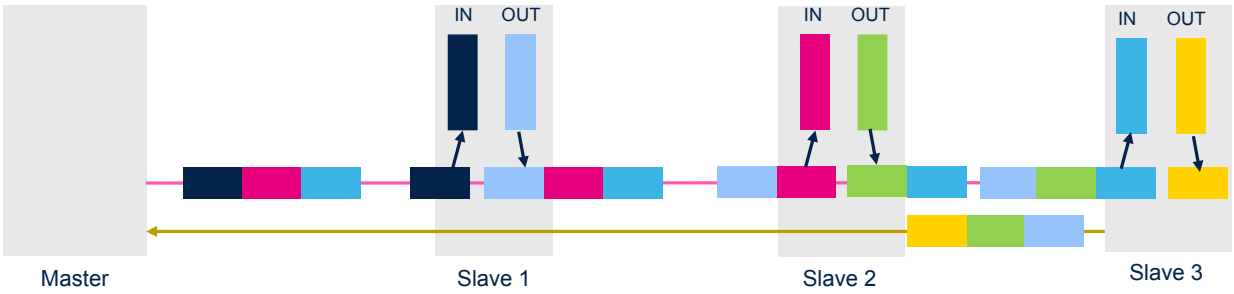
Ther routines available in the source are:

- `void GPIO_Actuation_Init (void)` to initialize all the GPIOs related to the ICs;
- `void Read_DIAG_IPS (void)` to manage the diagnostic information received by the IPS160H and CLT03-2Q3, reading the GPIO state (HIGH level→GOOD status, LOW level→ FAULT);
- `void CLT03_2Q3_Read(void)` to read the input voltage level related to the sensor feedback. For example, considering a use case where a proximity sensor is connected, 0 V on the input and on the MCU side means no object is present, whereas a 24 V on the input and on the MCU side indicates the presence of an object;
- `void IPS160H_Actuate(void)` to drive the IPS160H inputs according to the command received through the RS485 communication;
- `void Update_Actuation_ERT (void)` is used to communicate the output state (ON/OFF) and the output status (GOOD, FAULT) to the master EtherCAT through the RS485 communication.

3 Implementing multi-axial position control

The STSW-ETHDRV01V1 firmware provides an application example implementing master/slave communication based on the EtherCAT protocol. It allows the user to build communication with cyclic and acyclic process data with low jitter and latency. Moreover, thanks to the daisy chain connection, it is possible to connect more than one slave node to the same bus.

Figure 7. Bus data flow diagram



You can modify the code according to your needs, adding source files or changing the existing ones for variables, structures, time scheduler configuration for protocol, etc.

The user editable sections are related to the following layers:

- Application
- Middleware
- Drivers

The source files for the board or IC management must be included in the BSP folder.

The Middleware layer must be used to insert all the user source files related to libraries and other protocol types in the project.

3.1 Requirements

The STSW-ETHDRV01V1 implements the position control of an industrial end node (EtherCAT slave) based on the STEVAL-ETH001V1.

The firmware solution aims at changing the motor rotor position according to the target angle and the master logic. Even if a single position control can be executed on each node firmware, to achieve the synchronization of a multi-axial control, a real-time network is necessary.

To use the STEVAL-ETH001V1 board with the provided slave firmware, you need an EtherCAT master which implements the custom frame explained in Section 5 . The master sends the rotor position and commands to the slave via the custom frame. The firmware package is customized for a PMSM S160-2B305 series.

4 Tool chain

The STSW-ETHDRV01V1 firmware package uses the application example provided by Hilscher with the addition of a library for motor control management.

It has been implemented using the IAR Workbench v8.5.5 and compiled without optimization.

The output converted format is ".hex" and the compiled files are located in the binary folder.

5 How to build an EtherCAT master

The EtherCAT master can be built through dedicated software or hardware.

This application example is based on an EtherCAT master built through dedicated software (TWINCAT). To run the application, a set of instructions has been implemented using structured text language to manage all the commands and data needed to properly drive the motor.

The code is able to manage only one node and then only one motor; the driving flow expects a definition on the rotor position master side and the desired time of usage to make the motor reach the required position.

After this definition, the values are stored in the data frame and the cycling is sent to the slave node; in the meantime, the master manages the communication with the slave node, sending also a set of commands to manage alignment, get status, set the position and properly complete the driving sequence.

Table 2. List of commands

Output Data buffer	Data format	Data type	Data value
Command ID	USINT (one byte)	Alignment	0x01
		Get Status	0x02
		Set Position	0x03
		Stop Motor	0x05
		Reset MCU	0x06
Motor Status	USINT (one byte)	Ready	0x01
		Aligned	0x06
		Positioned	0x07
		Fault	0x08
		Wrong command	0xFF

Rotor position and duration time data format is USINT (four bytes).

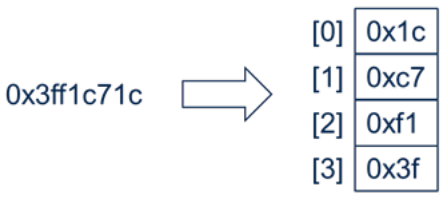
Rotor position and duration time values (expressed in radians and seconds, respectively) must be properly converted into a sequence of 4 bytes according to the procedure described hereafter.

All the real number needs to be rounded to the nearest floating point value (for example, the value 1.88888888888888888888888888888888 is rounded to 1.888888835906982421875).

The floating point value must be converted into the equivalent hexadecimal representation according to IEEE 754; so, 1.888888835906982421875 is represented as 0x3ff1c71c (it is possible to find some online free converter on the web).

The 32-bit hexadecimal value has to be split into four 8-bit values and set in the corresponding four-byte of rotor position or duration time arrays with little-endian memory representation as shown in the figure below.

Figure 8. IEEE 754 binary floating point representation example



These arrays are sent to the EtherCAT node.

In the firmware example, the rotor angle rotation is equal to 6.28 rad and the duration time is 2 seconds.

Table 3. **STSW-ETHDRV01V1 firmware example - rotor angle rotation hexadecimal and decimal values**

HEX	Decimal representation
0x40	64
0xC8	200
0xF5	245
0xC3	195

Table 4. **STSW-ETHDRV01V1 firmware example - duration time hexadecimal and decimal values**

HEX	Decimal representation
0x40	64
0x00	0
0x00	0
0x00	0

Figure 9. **STSW-ETHDRV01V1 firmware example rotor angle and duration**

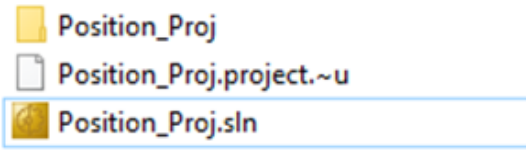
```
Position_Int3:= 64;
Position_Int2:= 200;
Position_Int1:= 245;
Position_Int0:= 195;

Duration_Int3:= 64;
Duration_Int2:= 0;
Duration_Int1:= 0;
Duration_Int0:= 0;
```

5.1 How to run the master EtherCAT

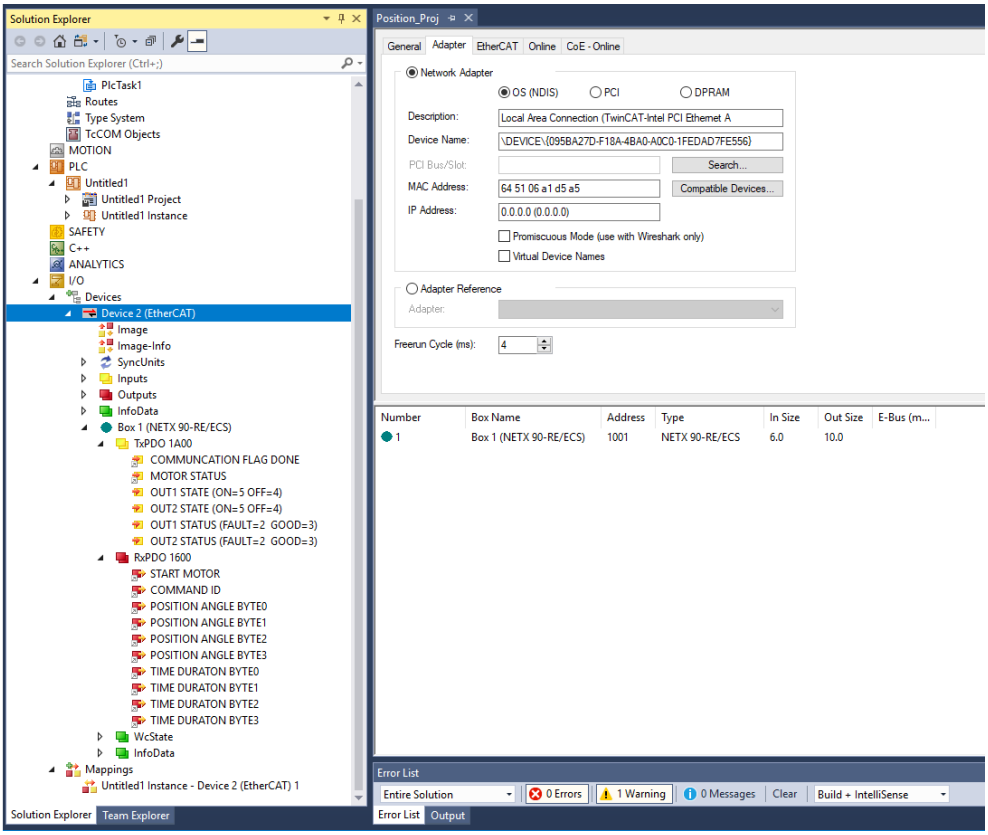
Step 1. Select the solution in the Utilities folder.

Figure 10. **Utilities folder project selection**



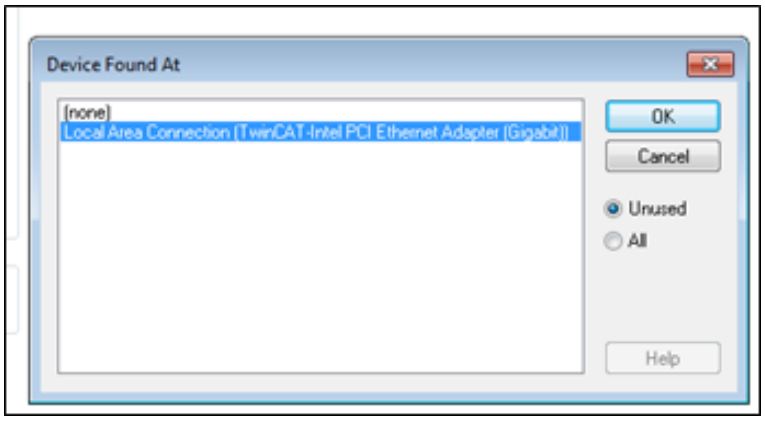
Step 2. Select [**Devices**] in the left window, open [**Adapter**] and click [**Search**].

Figure 11. Device search



Step 3. Select the device name in the [**Device Found**] window.

Figure 12. Device found

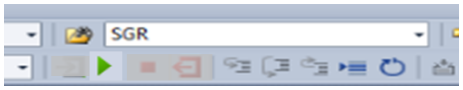


Step 4. Reload the device using the toolbar button ( ).

Step 5. Activate the free running mode clicking on the related button ( ).

Step 6. Click on the green button ( ) and then on the green arrow in the toolbar.

Figure 13. Toolbar green arrow



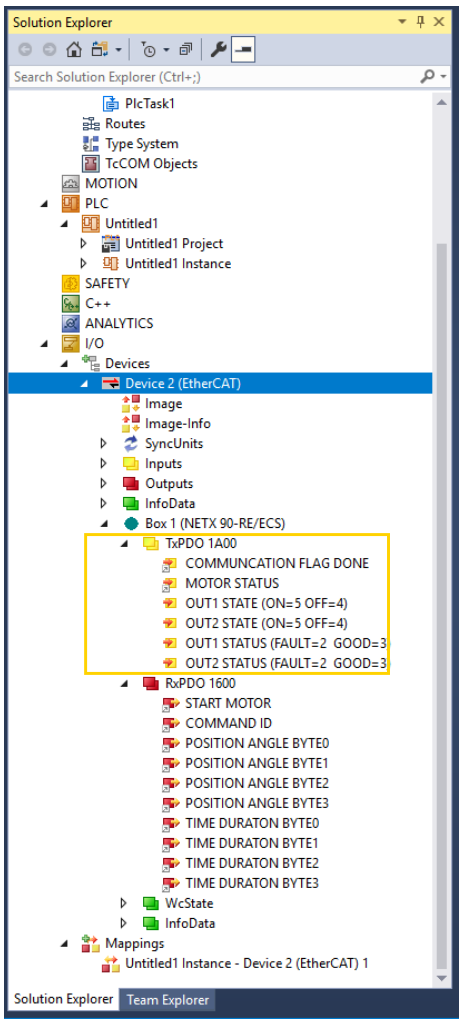
If the procedure has been performed properly, in the TWINCAT, the toolbar appears as shown below.

Figure 14. Toolbar appearance when procedure running is successful

The on-board green LED close to RJ45 connector turns red and the motor turns the rotor to reach the right position.

Observing the TXPDO buffer in TWINCAT, it is also possible to see the feedback related to the digital I/O circuit.

Figure 15. Digital I/O circuit feedback



6 Digital actuation circuit management

Digital actuation circuits have been managed at firmware level by implementing a set of routines useful to read the input voltage forced by a proximity sensor (0 to 24 V) or other references and to handle the digital output device to drive loads.

The interaction of digital input and digital output parts is allowed through a set of string messages exchanged between a PC and the STEVAL-ETH001V1 through the RS485 interface.

To use it, it is necessary to run a hyper terminal on a PC, configure it properly and run the communication as described hereafter.

6.1 How to set up the hyper terminal

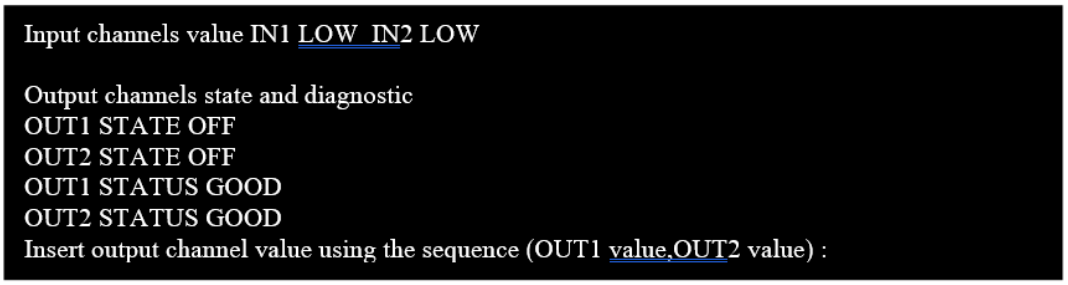
Step 1. Connect the STEVAL-ETH001V1 evaluation board to the PC using U18 stripline connector.

Step 2. Enable the communication by selecting the COM port linked to the RS485 dongle connected to the board.

Step 3. Configure the communication parameters through the `void RS485_Init(void)` routine available in the RS485.c file.

If the communication is successfully established, the below string message appears on the screen:

Figure 16. Successful communication - string message



Note: Default configuration is:

- *Baud rate: 921600*
- *Data: 8 bit*
- *Parity: None*
- *Stop bits: 1 bit*
- *Flow control: none*

When using Tera Term, set:

- *Local echo*
- *Receive : CR*
- *Transmit: CR*

6.2 How to manage the communication

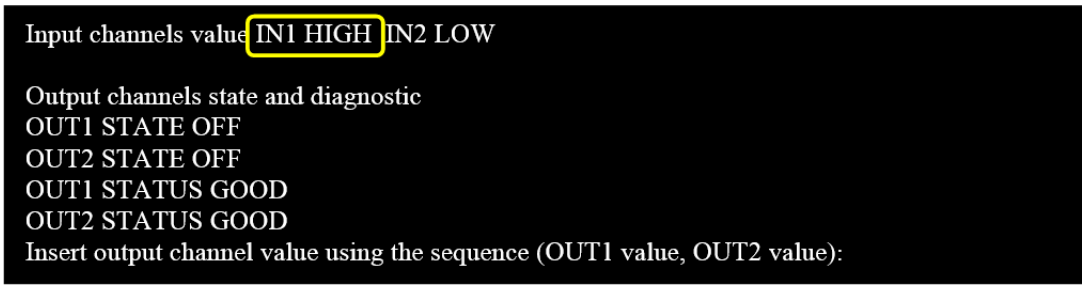
After powering the STEVAL-ETH001V1 evaluation board:

Step 1. Connect a proximity sensor or provide a 24 V to J14 screw connector through a switch.

Step 2. Connect a load to J2 screw connector (refer to UM2807, Section 3.3).

If the connection is related to digital input channel one and digital output channel on, the microcontroller sends the following string message:

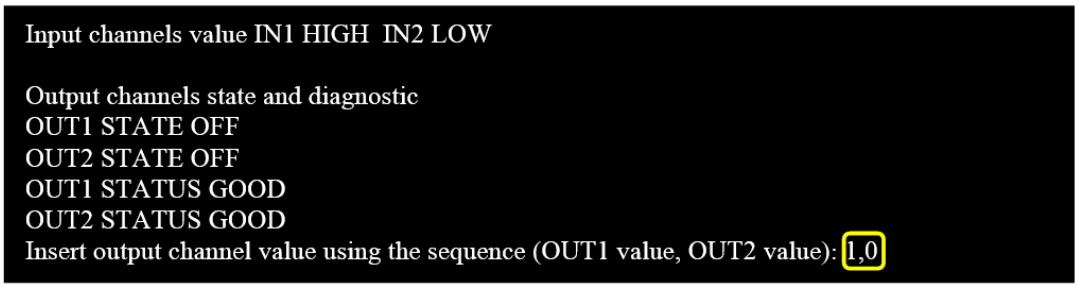
Figure 17. Digital input channel one and digital output channel on connection - microcontroller string message



Step 3. Use the string format suggested, type 1 to turn the output ON or 0 to turn or keep the output OFF. For a correct parsing, use a comma between the two digits and click ENTER.

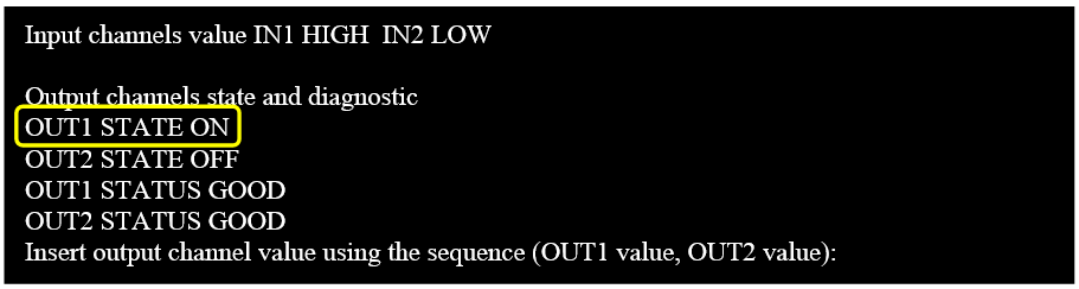
If for example, you type (1, 0)

Figure 18. Output channel value sequence



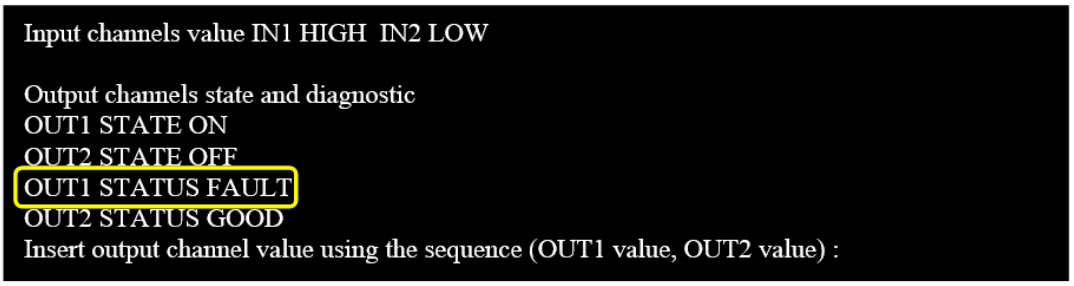
the MCU sends the below message

Figure 19. MCU message



If in this condition the output is going in overload and the IPS160H DIAG pin switches down, the fault condition is detected by the MCU and shown as follows:

Figure 20. Fault condition



6.3 String messages

Table 5. List of communication string messages

String message	value	Description	Note
Input channels value	LOW HIGH	0 V is applied to the CLT03-2Q3 devices 24 V is applied to the CLT03-2Q3 devices	String message sent by the microcontroller to the user
OUTx STATE	OFF ON	Output in OFF condition Output in ON condition	String message sent by the microcontroller to the user
OUTx STATUS	FAULT GOOD	Overload or overtemperature condition detected Normal operation	String message sent by the microcontroller to the user
OUTx value	1 0	Output channel turns ON Output channel turns OFF	String message sent by the user to the MCU

Revision history

Table 6. **Document revision history**

Date	Version	Changes
26-Apr-2021	1	Initial release.

Contents

List of tables

List of figures