User Manual

QEC-M-02

DM&P Vortex86EX2 Processor

EtherCAT MDevice with Isolated 32-ch DIO, USB, GLAN, COM port, HDMI display.

(Revision 1.1)

REVISION

Date	Version	Description
2025/03/12	Version1.0	First Release.
2025/03/31	Version1.1	• Update Power LED Error Indication in page 20. • Add “(This chapter is available in multiple languages)” in Ch.4 Getting Started. • Delete the extra “•” in Ch.4.5.4 CANopen over EtherCAT (CoE) Functions.

COPYRIGHT

Ver.1.1 March, 2025

TRADEMARKS ACKNOWLEDGMENT

For more detailed information or if you are interested in other ICOP products, please visit our official websites at:

- Global: www.icop.com.tw
- USA: www.icoptech.com
- Japan: www.icop.co.jp
- Europe: www.icoptech.eu
- China: www.icop.com.cn

For technical support or drivers download, please visit our websites at:

- https://www.icop.com.tw/resource_entrance

For EtherCAT solution service, support or tutorials, 86Duino Coding IDE 500+ introduction, functions, languages, libraries, etc. Please visit the QEC website:

- QEC: https://www.qec.tw/

This Manual is for the QEC series.

SAFETY INFORMATION

- Read these safety instructions carefully.
- Please carry the unit with both hands and handle it with caution.
- Power Input voltage +19 to +50VDC Power Input (Typ. +24VDC)
- Make sure the voltage of the power source is appropriate before connecting the equipment to the power outlet.
- To prevent the QEC device from shock or fire hazards, please keep it dry and away from water and humidity.
- Operating temperature between -20 to +70°C/-40 to +85°C (Option).
- When using external storage as the main operating system storage, ensure the device's power is off before connecting and removing it.
- Never touch un-insulated terminals or wire unless your power adaptor is disconnected.
- Locate your QEC device as close as possible to the socket outline for easy access and avoid force caused by the entangling of your arms with surrounding cables from the QEC device.
- If your QEC device will not be used for a period of time, make sure it is disconnected from the power source to avoid transient overvoltage damage.

WARNING!

DO NOT ATTEMPT TO OPEN OR TO DISASSEMBLE THE CHASSIS (ENCASING) OF THIS PRODUCT. PLEASE CONTACT YOUR DEALER FOR SERVICING FROM QUALIFIED TECHNICIAN.

CONTENT

Ch. 1

General Information

1.1 Introduction

The ICOP QEC-M-02 is an EtherCAT MDevice powered by the Vortex86EX2 processor, featuring two independent cores that efficiently handle real-time fieldbus communication and user applications.

Efficient Development with 86Duino IDE
The development environment utilizes 86Duino IDE, an industrial Arduino-like platform that supports EtherCAT API, graphical programming tools, and high-level C/C++ programming, enabling rapid development while reducing hiring challenges and time to market. Beyond EtherCAT, the QEC-M-02 also supports Modbus, Ethernet TCP/IP, and CAN bus, providing a complete industrial automation solution.

Real-Time Precision for Motion and I/O Control
With a precise synchronization cycle of 125µs (min.) and jitter time under 1µs, the QEC-M-02 is designed for highly synchronized motion and I/O control applications, making it ideal for precision automation. *(Read More: EtherCAT MDevice Benchmark)*

Robust Storage, Reliable I/O, and Versatile Connectivity with compact size
For storage and stability, the QEC-M-02 integrates 2GB SLC eMMC, ensuring a reliable operating system. Users can easily upload executable files, HMI images, or data via 86Duino IDE without impacting master system performance.
The QEC-M-02 includes 16 isolated digital input channels and 16 isolated digital output channels, supporting NPN and PNP sink configurations with a safe operating frequency of up to 3000 Hz. The digital output supports load voltages up to 60VDC, with a typical current of 500mA per channel and a peak current of 1000mA. Isolation protection is rated at 2500 vrms (input) and 1500 vrms (output), ensuring reliability and safety in industrial environments.

For display functionality, the QEC-M-02 features an HDMI output (1280 x 720 x 256) and supports LVGL and the 86HMI editor, making it an excellent solution for HMI applications.
In terms of connectivity, the QEC-M-02 offers EtherCAT, Giga LAN, USB, COM, HDMI, a rotary switch, and a RUN/STOP switch, providing flexible integration options with off-the-shelf APIs for automation projects.

Measuring 107.45 x 49 x 77.31 mm and weighing only 370g, the QEC-M-02 supports DIN-rail mounting with a robust Euroblock terminal block for easy installation and wiring. It operates in a temperature range of -20°C to +70°C (with an optional -40°C to +85°C variant) and features built-in monitoring for temperature, voltage, current, and startup time, ensuring optimal system performance in industrial environments.

1.1.1 QEC EtherCAT MDevice Architecture

The EtherCAT MDevice software is primarily divided into two parts, each running on the respective systems of the Vortex86EX2 CPU. They are responsible for the following tasks:

- **EtherCAT MDevice Library**
  - Provides C/C++ application interfaces:
    - Initialization interface.
    - Configuration interface.
    - Process Data (PDO) access interface.
    - CAN application protocol over EtherCAT (CoE) access interface.
    - File access over EtherCAT (FoE) access interface.
    - SubDevice Information Interface (SII) access interface.
    - Distributed Clocks (DC) access interface.
- **EtherCAT MDevice Firmware**
  - Executes the EtherCAT MDevice Core.
  - Controls the Primary/Secondary Ethernet Driver, sending EtherCAT frames

The programs are designed to run on the FreeDOS operating system and have been compiled using the GCC compiler provided by the DJGPP environment.

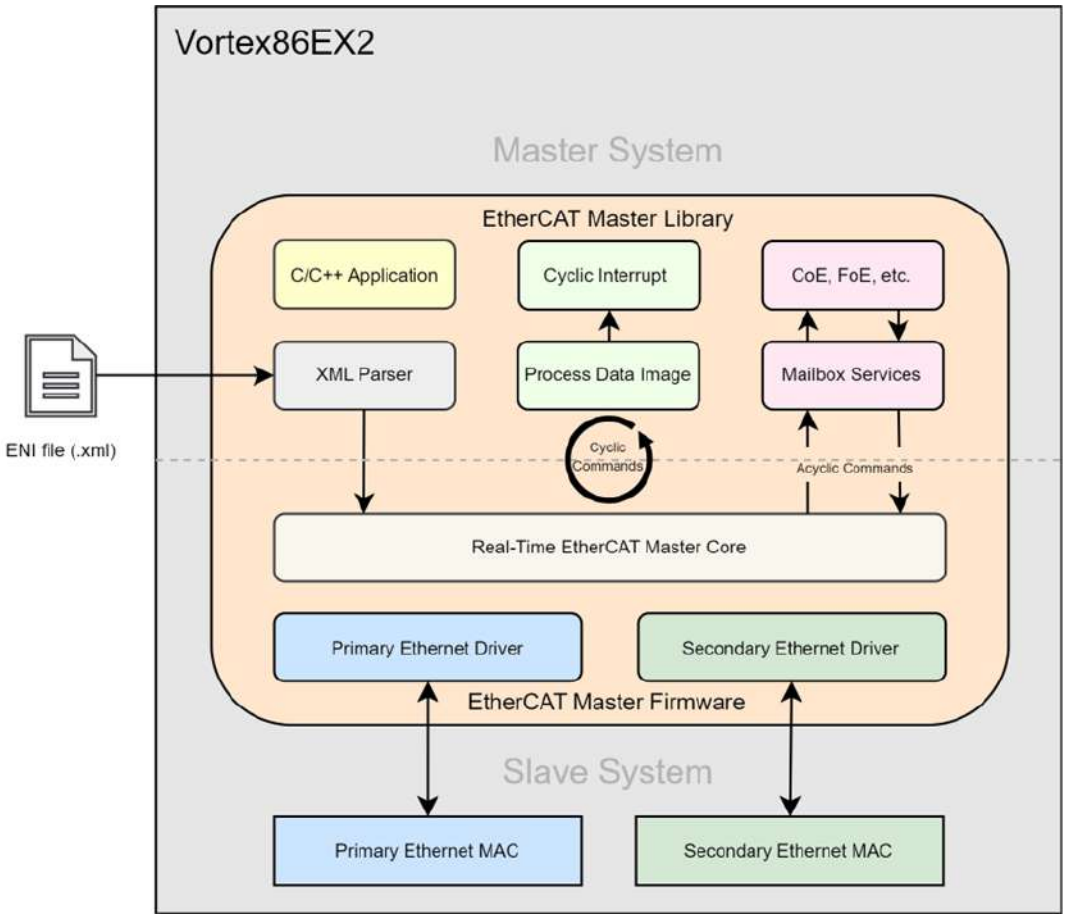
1.1.2 Hardware Platform

The EtherCAT MDevice software only runs on the Vortex86EX2 CPU produced by DM&P, which features a dual-system architecture. It is divided into Master System and Slave System, each running its own operating system, with communication between systems facilitated by Dual-Port RAM and event interrupts.

Their respective tasks are as follows:

- **Master System**
  - User's EtherCAT application.
  - User's HMI application.
  - User's Ethernet application.
  - And so on.
- **Slave System**
  - Only responsible for running the EtherCAT MDevice Firmware.

As most applications run on the Master System, the EtherCAT MDevice Firmware running on the Slave System is free from interference by other applications. This setup allows it to focus on executing the EtherCAT MDevice Core, ensuring the synchronization and real-time capabilities of EtherCAT.



1.1.3 Dual-System Synchronization

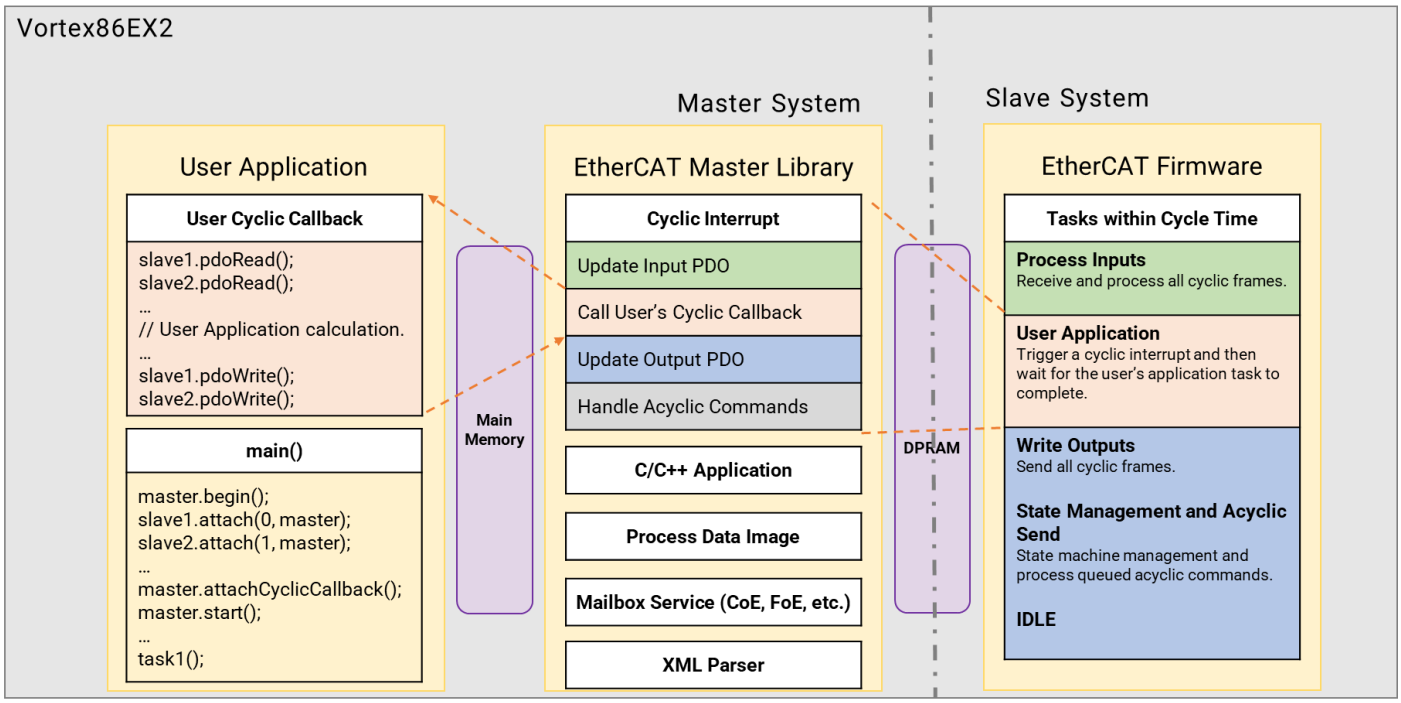
The primary focus of this section is the synchronization of dual-system PDO data. As illustrated in the diagram below, the User Application and EtherCAT MDevice Library blocks run on the Master System, while the Real-Time EtherCAT MDevice Core runs on the Slave System.

When the EtherCAT MDevice Core reaches the **Process Inputs** stage, it receives all cyclic frames from the Ethernet Driver and copies Input PDO data to the DPRAM.

Upon reaching the **User Application** stage, the EtherCAT MDevice Core triggers a Cyclic Interrupt to the Master System. Upon receiving the Cyclic Interrupt, the Master System executes the interrupt handling procedure of the EtherCAT MDevice Library. It moves Input PDO data from DPRAM to Main Memory, calls the user-registered Cyclic Callback, transfers Output PDO data from Main Memory to DPRAM after the Cyclic Callback completes, processes acyclic commands, and concludes the interrupt handling procedure. At this point, both the EtherCAT MDevice Core's **User Application** and the interrupt handling procedure are completed simultaneously.

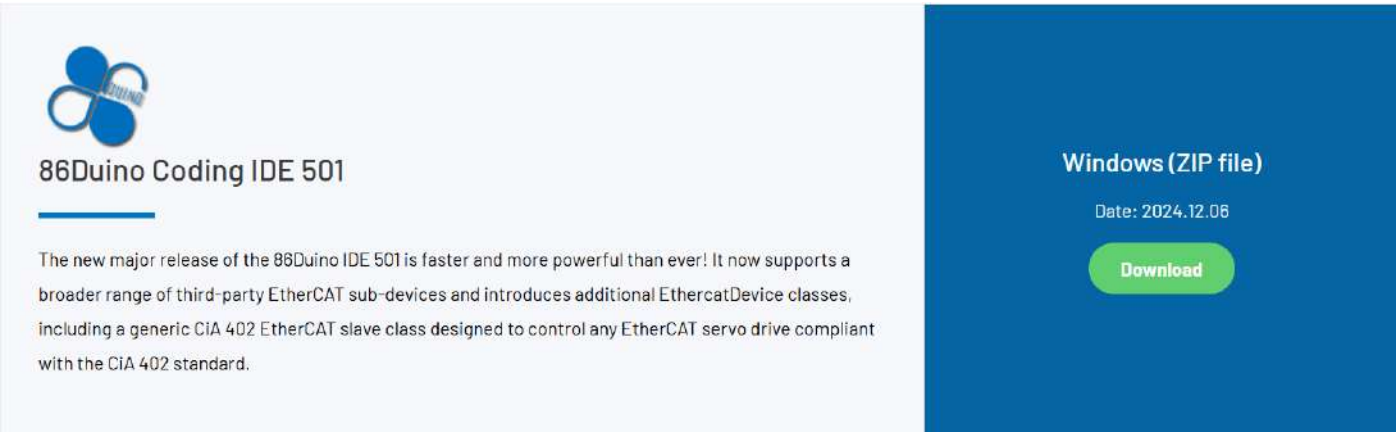
When the EtherCAT MDevice Core reaches the **Write Outputs** stage, it copies Output PDO data from DPRAM to the Ethernet Driver's DMA and sends frames.

These tasks are executed periodically in a cyclic manner, following the outlined procedural steps, ensuring the synchronization of dual-system PDO data.



1.1.4 Software Support

The 86Duino integrated development environment (IDE) software makes it easy to write and upload code to 86Duino boards and QEC MDevice. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software, which can be downloaded from https://www.qec.tw/software/.



QEC MDevice software, 86Duino IDE, also offers a configuration utility: **86EVA**, a graphic user interface tool for users to edit parameters for the EtherCAT network; its functions are as follows:

- EtherCAT SubDevice scanning.
- Import ENI file.
- Setting EtherCAT MDevice.
- Configure EtherCAT SubDevices.

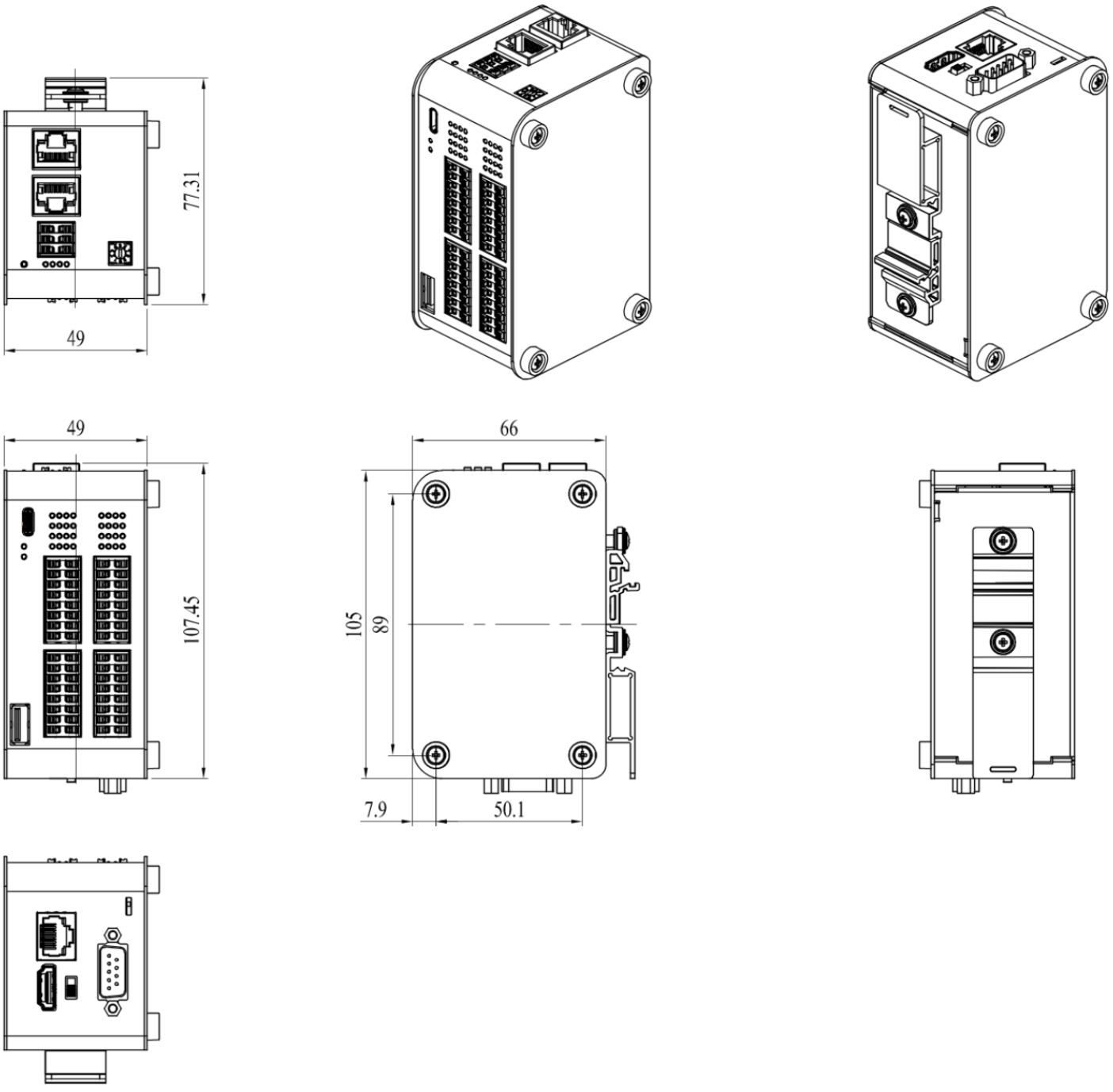
For other detailed functions, please refer to the 86EVA User Manual.

1.2 Specifications

Hardware			
CPU	DM&P Vortex86EX2 Processor, Master 533MHz/Slave 400MHz		
Memory	512MB/1GB DDRIII Onboard		
Storage	32MB SPI Flash/2GB SLC eMMC		
Display	HDMI 2.0 (Resolution: 1280 x 720 x 256)		
LAN	1Gbps Ethernet RJ45 x1, 10/100Mbps Ethernet RJ45 x2 for EtherCAT		
I/O Connector	Power DC Input/Output Connector x1 USB (Type-C) x1 (Upload/Debug only) Rotary switch (Max. 16 modes) x1	32-channel isolated GPIO (Option) COM port for RS232/485 x1 Run/Stop switch x1	HDMI port x1 USB 2.0 x1 LAN x3
Digital I/O	Digital Input	Digital Output	
Channels	16 (Isolated)	16 (Isolated)	
Input Type	Sink (Supports both NPN and PNP configurations)		
Frequency	Safe Operating Frequency: ≤3000 Hz Rise Time (tr): 6 μs, Fall Time (tf): 6 μs	Operating Frequency: 1000Hz Turn-On Time: 0.25 ms, Turn-Off Time: 20 μs	
Load Voltage	+24V (Options: 24V or 48V)	+60Vdc	
Load Current	-	Typ. 500mA (peak 1000mA)	
Isolation Protection	2500 Vrms	1500 Vrms	
Serial Port			
Interface Mode	RS232/RS485 (D-Sub 9-pin)		
Data transfer rate (bps)	2400,4800,9600,14400,19200,38400,57600,115200		
Data width (bit)	5/6/7/8		
Hardware Flow Control	CTS/RTS		
General			
Protocol	EtherCAT, Ethernet, Modbus, etc.		
EtherCAT Cycle Time	125 μs (min.)		
Power Connector	4-pin Power Input/Output & 2-pin FGND		
Power Requirement	+19 to +50VDC Power Input (Typ. +24VDC)		
Power Consumption	6W		
Operating Temperature	-20 to +70°C/-40 to +85°C (Option)		
Dimension	107.45 x 49 x 77.31 mm		
LED Indicator	PWR, RUN, LINK, Digital status		
Rotary switch	Max. 16 modes		

Weight	370 g
Mounting	DIN-Rail
Internal Monitoring	Temperature, Voltage, Current, Start-up time
Software Support	86Duino Coding IDE 501+

1.3 Dimension

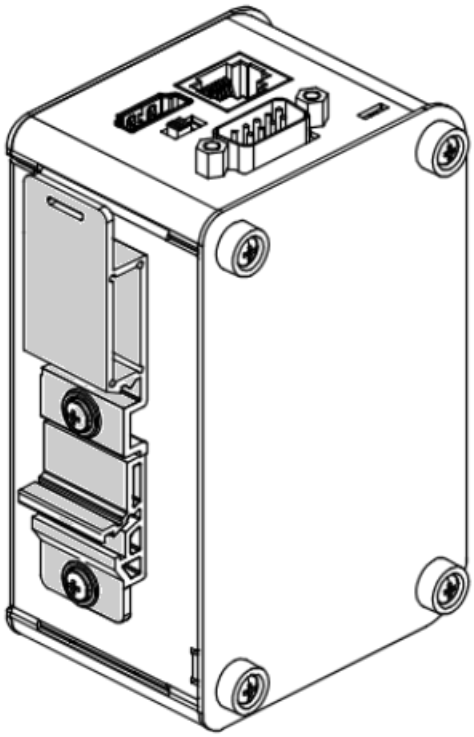


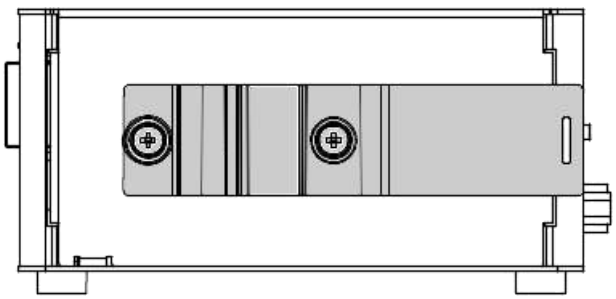
(Unit: mm)

1.4 Mounting Instruction

QEC-M-02 is an easy-install design to help you maintain your modules easily. Please refer to Ch.3.1 DIN-Rail installation.

- **DIN-Rail**





1.5 Ordering Information

P/N	-	Type	PoE	-	Feature			-	Wide Temp.	-	Coating
					Digital	Channel					
QEC-M		02	X		D	X	X		X		X

1. P/N: Code 1~4

M: EtherCAT MDevice.

2. Type: Code 5~6

02: PLC.

3. PoE: Code 7

P: RJ45 PoE Device, Red Housing

None or E: RJ45 w/o power, Black Housing

(Standard: None)

4. Feature: Code 8~10

D: Digital I/O

X: 0 ,4 ,8, C (12), F (16), Z (24), T (32) input channels

X: 0 ,4 ,8, C (12), F (16), Z (24), T (32) output channels

5. Wide Temp.: Code 11

X: -40 to +85°C.

None or R: -20 to +70°C (Standard: -20 to +70°C).

6. Coating: Code 12

C: Yes.

None or N: Normal

QEC-M-02 X – DXX – X - X

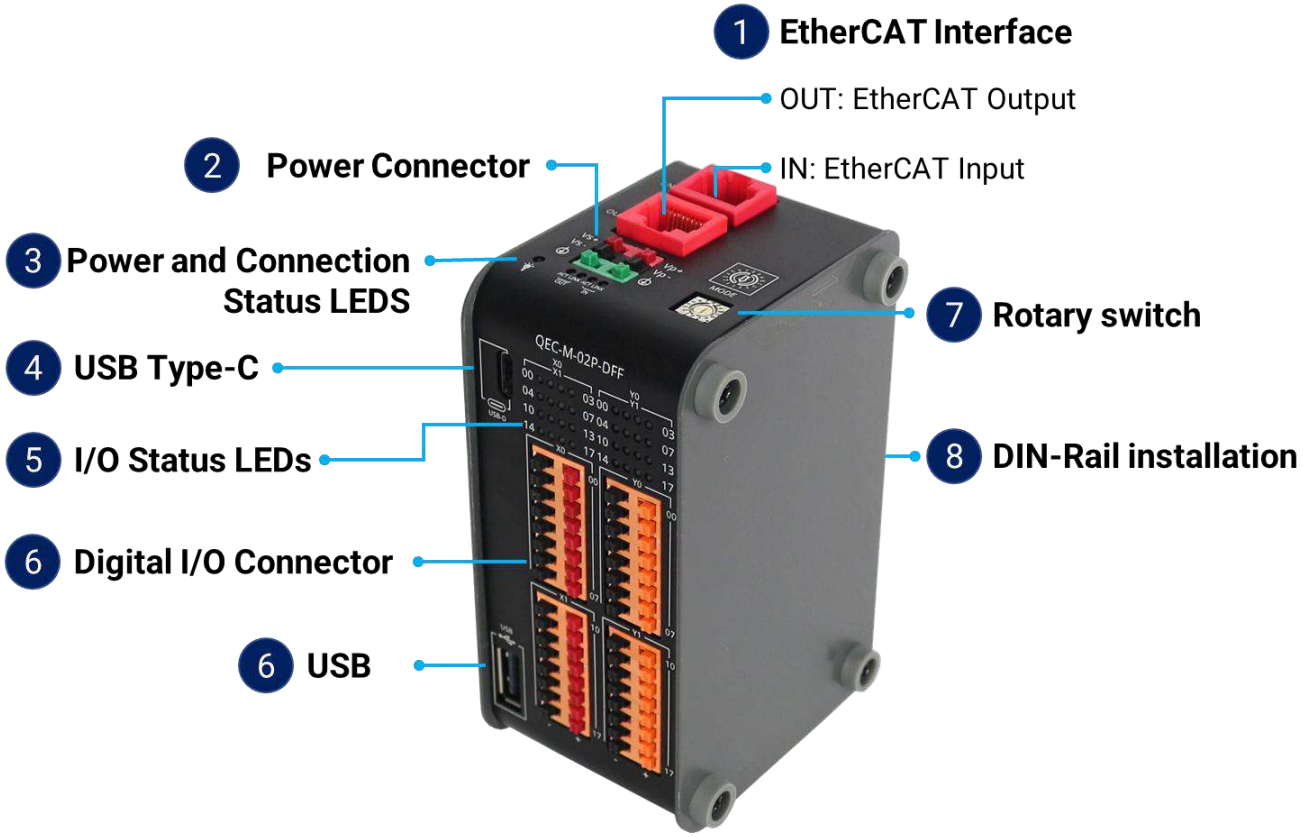
1.5.1 Ordering Part Number:

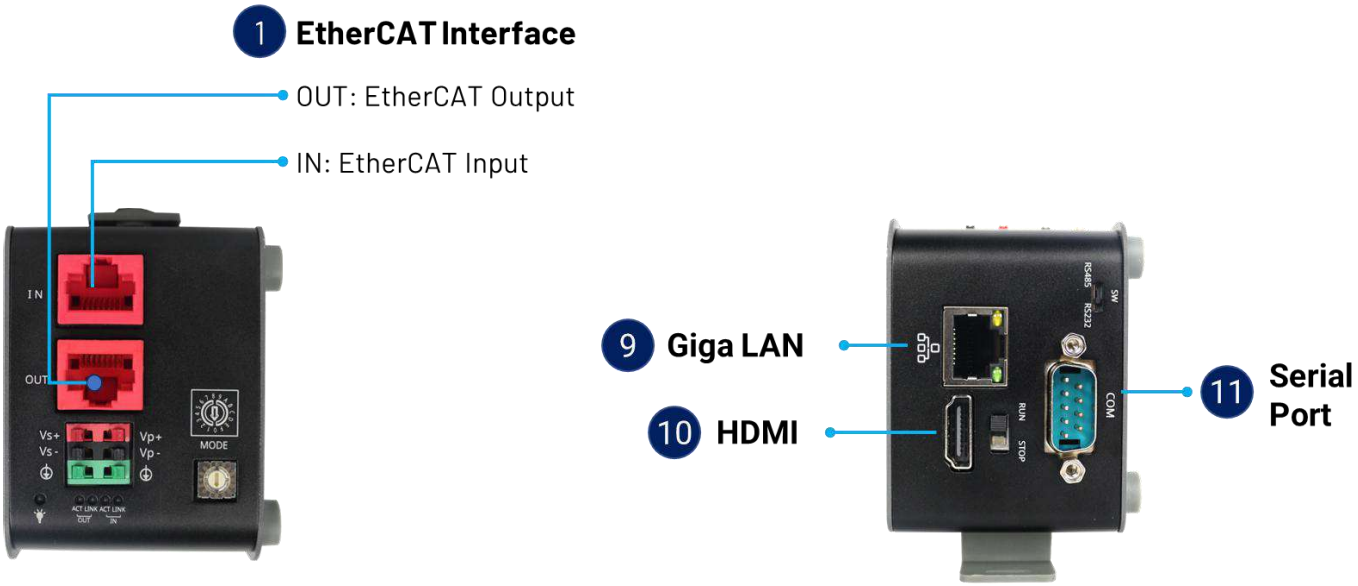
- **QEC-M-02-DFF**: EtherCAT MDevice with Isolated Digital Input 16-ch/Output 16-CH / USB / GLAN / HDMI / COM / Rotary switch.
- **QEC-M-02P-DFF**: EtherCAT MDevice with Isolated Digital Input 16-ch/Output 16-CH / USB / GLAN / HDMI / COM / Rotary switch / POE.

Ch. 2

Hardware System

2.1 General Technical Data



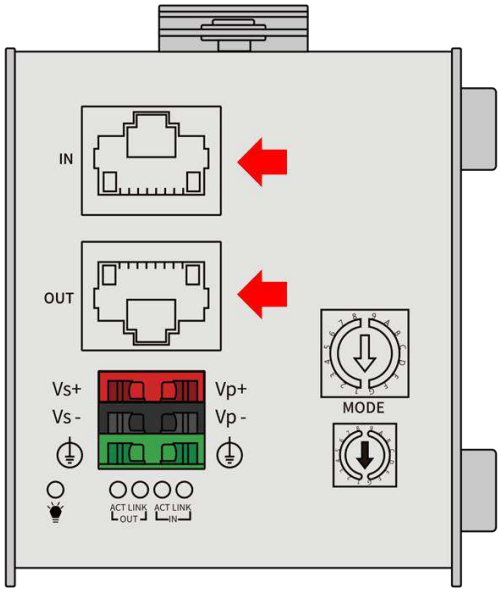


2.2 General Summary

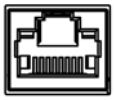
No.	Description		Type Narrative	Pin #
1	EtherCAT Interface	IN	External RJ45 Connector (Gold finger)	8-pin
		OUT		8-pin
2	Power Connector		Terminal Block Interface	6-pin
3	Power and Connection Status LEDs		External Status LEDs	-
4	USB Type-C (Debug and Upload port)		USB Type-C	-
5	I/O Status LEDs		External Status LEDs	-
6	Digital I/O Connector		External Status LEDs	-
7	Rotary switch		External Rotary switch	-
8	DIN-Rail		-	-
9	Giga LAN		External RJ45 Connector (Gold finger)	8-pin
10	HDMI		HDMI 2.0	-
11	Serial Port		DB9 serial port	9-pin

2.2.1 EtherCAT Interface

RJ45 Connectors.



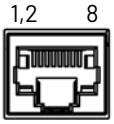
EC IN



Pin #	Signal Name	Pin #	Signal Name
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

EC OUT

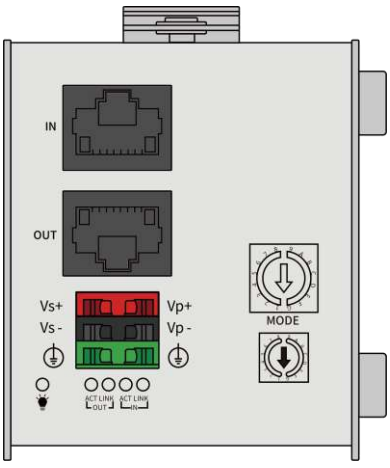


Pin #	Signal Name	Pin #	Signal Name
1	LAN2_TX+	2	LAN2_TX-
3	LAN2_RX+	4	VS+
5	VP+	6	LAN2_RX-
7	VS- (GND)	8	VP- (GND)

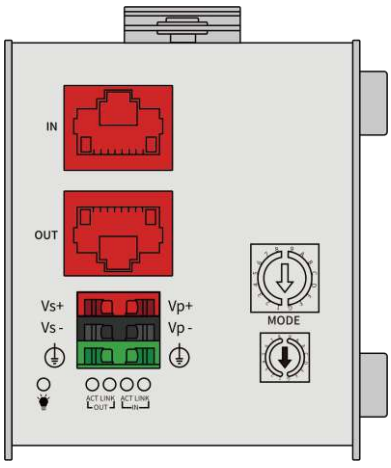
* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

Note. QEC's PoE (Power over Ethernet)

In QEC product installations, users can easily distinguish between PoE and non-PoE: if the RJ45 house is red, it is PoE type, and if the RJ45 house is black, it is non-PoE type.



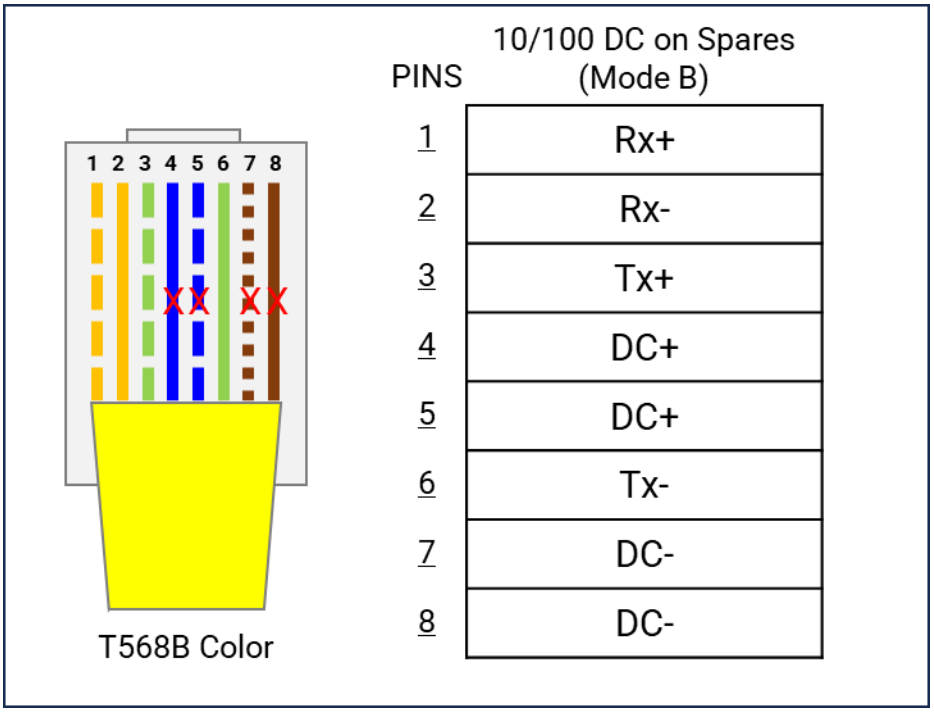
Non-PoE type



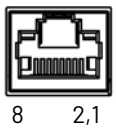
PoE type

PoE (Power over Ethernet) is a function that delivers power over the network. QEC can be equipped with an optional PoE function to reduce cabling. In practice, PoE is selected based on system equipment, so please pay attention to the following points while evaluating and testing:

1. When connecting PoE and non-PoE devices, make sure to disconnect Ethernet cables at pins 4, 5, 7, and 8 (e.g., when a PoE-supported QEC EtherCAT MDevice connects with a third-party EtherCAT SubDevice).



2. The PoE function of QEC is different and incompatible with EtherCAT P, and the PoE function of QEC is based on PoE Type B, and the pin functions are as follows:



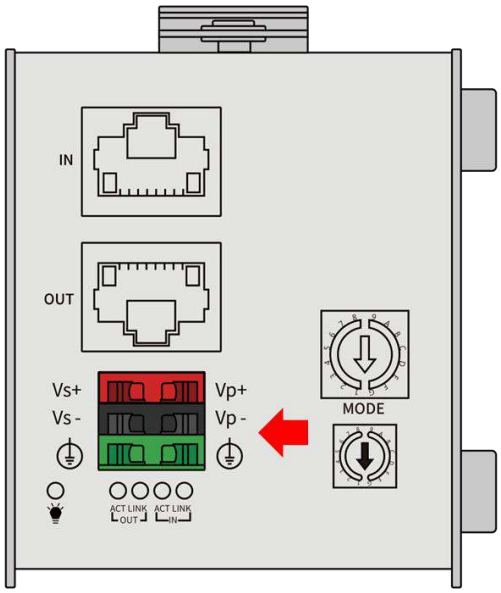
Pin #	Signal Name	Pin #	Signal Name
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

3. QEC's PoE power supply is up to 24V/3A.

2.2.2 Power Connector

Euroblock Connectors.
4-pins Power Input/Output & 2-pins FGND.



Vs for system power; Vp for peripheral power and backup power.

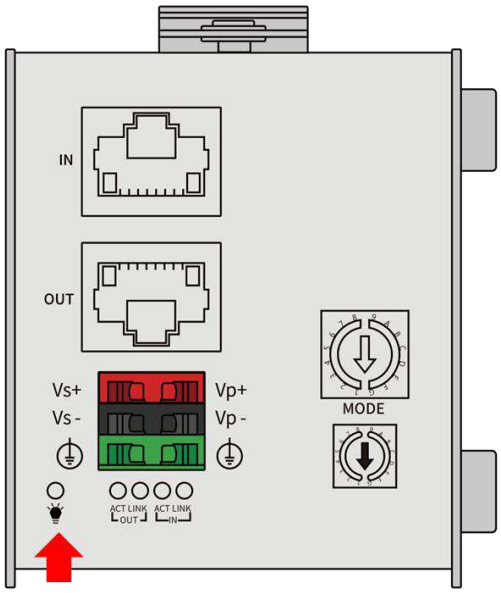
Pin #	Signal Name	Pin #	Signal Name
1	Vs+	2	Vp+
3	Vs- (GND)	4	Vp- (GND)
5	F.G	6	F.G

* Power Input voltage +19 to +50VDC Power Input (Typ. +24VDC)

2.2.3 Power and Connection Status LEDs

Power and connection status LEDs information.

Power Status LED



Power input is 24V (typical). The LED status provide high/low voltage warning.

Notation	States	Condition	Description
PWR	Green LED On	Voltage <= 50V and >= 45V Voltage <= 26V and >= 19V	When Vs and Vp voltages are confirmed to be normal, the Green LED will remain steady on.
	Green LED On Red LED On	Voltage < 45V and > 26V Voltage < 19V and > 12V	LEDs will alternately flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.
	Orange LED On	Voltage > 50V or < 12V	Orange LED (Green + Red) will continuously flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.

* Vs power status will be displayed first.

Power LED Error Indication: During startup, the red Power LED may flash to indicate a system error.

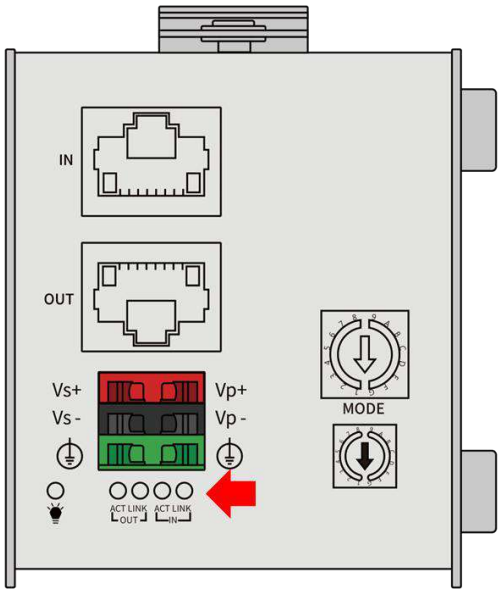
If the LED shows abnormal flashing patterns (e.g., long light or short flashes), please stop using the device and contact ICOP technical support immediately at info@icop.com.tw.
This helps avoid misjudgment and ensures proper troubleshooting.

Power ERROR Code table (Red LED Flashing Display (2 seconds/cycle)).

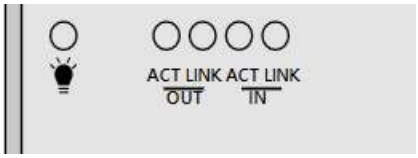
Long Light	Short Flash	Description
0 Long Light	Bootloader test passed, proceeding to APP stage.	
	1 short flash	EtherCAT chip communication failed.
	2 short flashes	EtherCAT chip internal RAM test failed.
	5 or 6 short flashes	Quartz oscillator abnormality.
1 Long Light	Bootloader active, APP not yet executed.	
	1 short flash	Internal SRAM failed.
	2 short flashes	APP software CHECKSUM failed.
2 Long Lights	Not yet defined.	

* Note: This table is for reference only. If any of the above patterns occur, please contact ICOP for diagnosis and resolution via email at info@icop.com.tw.

Connection Status LEDs



There are two LED for each LAN ports.



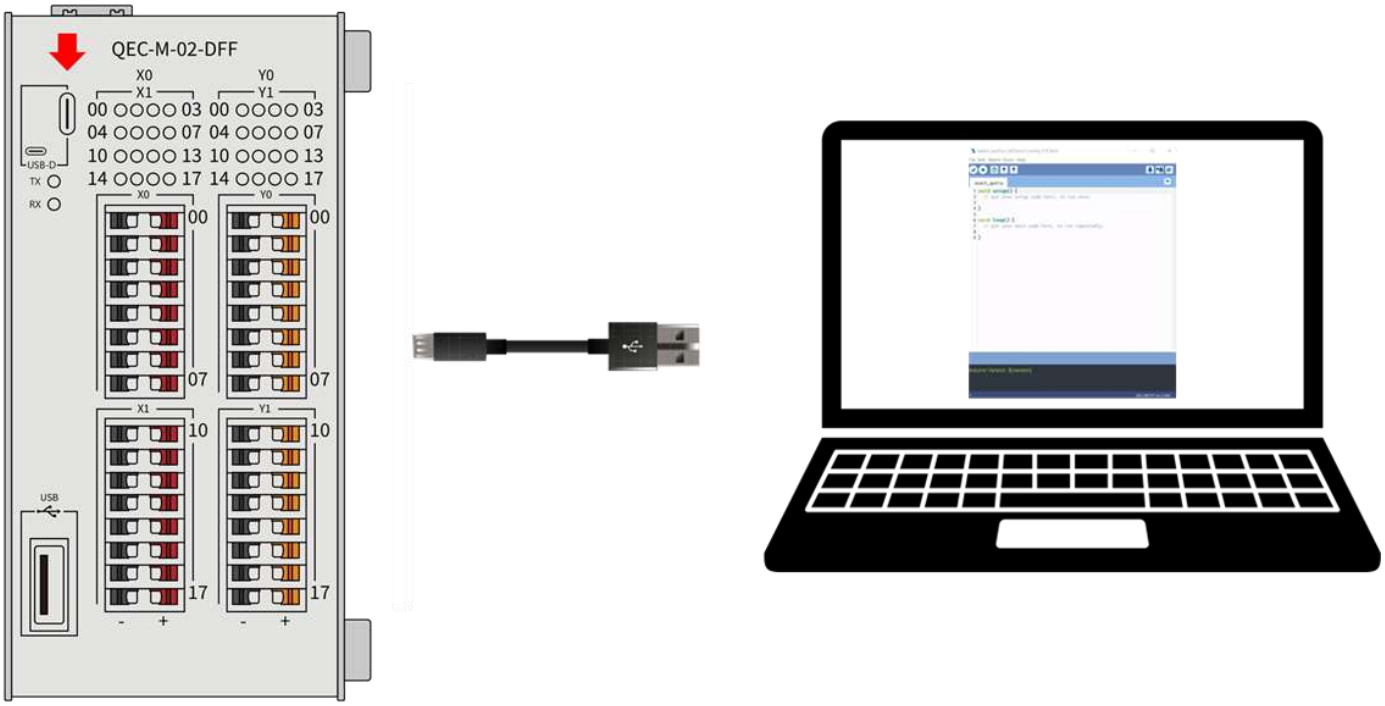
Please refer to the table below for the LAN port LED indications.

LAN	Notation	Color	States	Description
OUT	ACT	Green	Green LED Blinking	Data Activity
	LINK	Orange	Orange LED On	100Mbps Connection
IN	ACT	Green	Green LED Blinking	Data Activity
	LINK	Orange	Orange LED On	100Mbps Connection

2.2.4 USB Type-C

The QEC-M-02 features a USB Type-C port primarily used for programming uploads and debugging. You can connect the device to a computer using a USB to USB Type-C cable to upload code and configure the system.

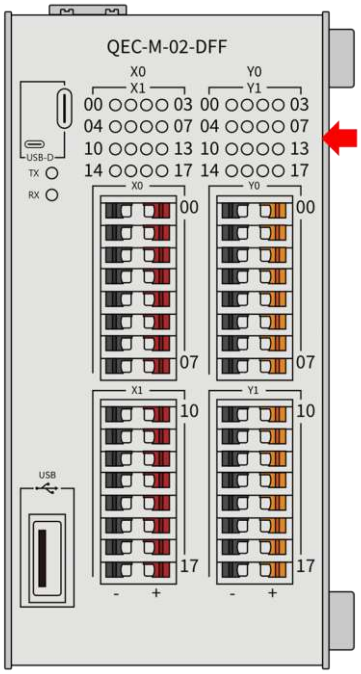
The USB Type-C port is located on the device's front panel (as shown in the diagram) and is labeled as USB-D.



For instructions on programming and setting up the QEC-M-02, please refer to Chapter 4: Getting Started.

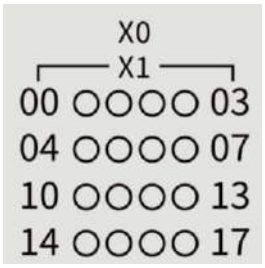
2.2.5 I/O Status LEDs

The I/O status LEDs for the Digital Input and Output.



Digital Input

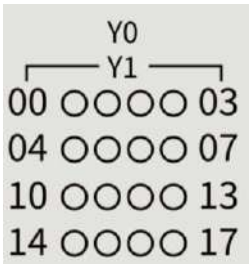
The LEDs of 16 digital inputs are 0 to 15, individually indicating the status of the 16 digital channels.



Notation	Color	States	Description
X	-	Off	Digital input status is “Off”
	Green	On	Digital input status is “On”

Digital Output

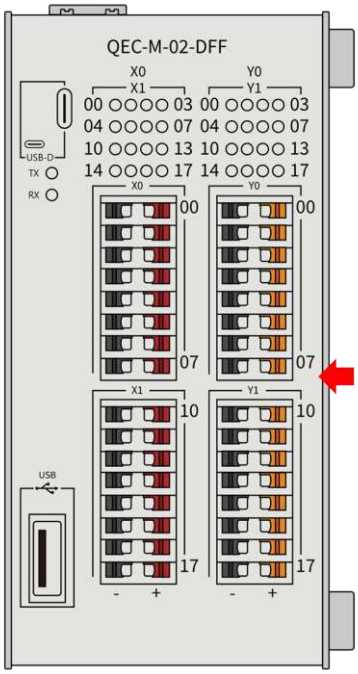
The LEDs of 16 digital outputs are 0 to 15, individually indicating the status of the 16 digital channels.



Notation	Color	States	Description
Y	-	Off	Digital output status is “Off”
	Green	On	Digital output status is “On”

2.2.6 Digital I/O Connector

The Isolated Digital Input and Digital Output connector description.



The Digital I/O specification table:

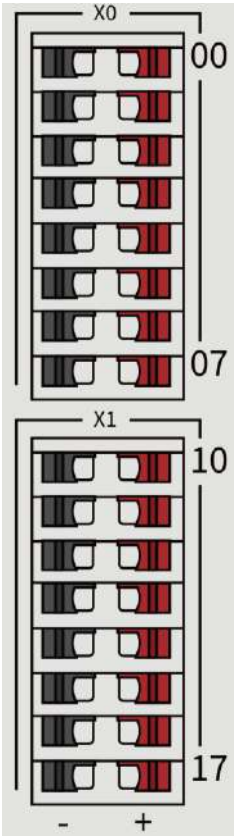
Digital I/O	Digital Input	Digital Output
Channels	16 (Isolated)	16 (Isolated)
Input Type	Sink (Supports both NPN and PNP configurations)	
Frequency	Safe Operating Frequency: ≤3000 Hz Rise Time (tr): 6us, Fall Time (tf): 6us	Operating Frequency: 1000Hz Turn-On Time: 0.25ms, Turn-Off Time: 20us
Load Voltage	+24V (Options: 24V or 48V)	+50Vdc
Load Current	-	Typ. 500mA (peak 1000mA)
Isolation Protection	2500 Vrms	1500 Vrms

Color Define:

- Digital Input: Positive (Red), Negative (Black).
- Digital Output: Positive (Orange), Negative (Black).

Digital Input

Digital Input: Positive (Red), Negative (Black).

Pin #	Signal Name		Pin #	Signal Name
00	X00-		00	X00+
01	X01-		01	X01+
02	X02-		02	X02+
03	X03-		03	X03+
04	X04-		04	X04+
05	X05-		05	X05+
06	X06-		06	X06+
07	X07-		07	X07+
08	X10-		08	X10+
09	X11-		09	X11+
10	X12-		10	X12+
11	X13-		11	X13+
12	X14-		12	X14+
13	X15-		13	X15+
14	X16-		14	X16+
15	X17-		15	X17+

Digital Input Load Voltage:

- Maximum Load Voltage: Load Voltage+24V (Options: 24V or 48V)

Channel Isolation:

- Isolated Channels: 16

Frequency

- Safe Operating Frequency: ≤3000 Hz (Rise Time (tr): 6us, Fall Time (tf): 6us)

I/O Type:

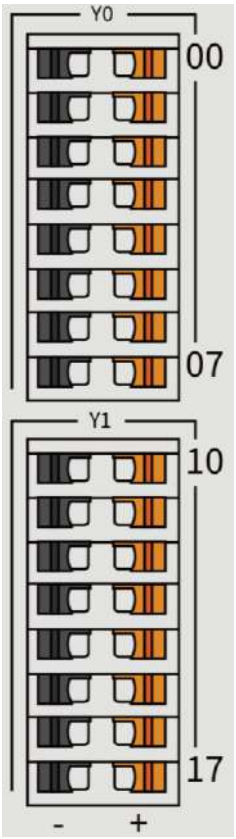
- Type: Sink (Supports both NPN and PNP configurations)

Isolation Protection

- 2500 Vrms

Digital Output

Digital Output: Positive (Orange), Negative (Black).

Pin #	Signal Name		Pin #	Signal Name
00	Y00-		00	Y00+
01	Y01-		01	Y01+
02	Y02-		02	Y02+
03	Y03-		03	Y03+
04	Y04-		04	Y04+
05	Y05-		05	Y05+
06	Y06-		06	Y06+
07	Y07-		07	Y07+
08	Y10-		08	Y10+
09	Y11-		09	Y11+
10	Y12-		10	Y12+
11	Y13-		11	Y13+
12	Y14-		12	Y14+
13	Y15-		13	Y15+
14	Y16-		14	Y16+
15	Y17-		15	Y17+

Digital Output Load Voltage:

- Maximum Load Voltage: +50Vdc

Digital Output Load Current:

- Typ. 500mA (peak 1000mA)

Channel Isolation:

- Isolated Channels: 16

Frequency

- Operating Frequency: 1000Hz (Turn-On Time: 0.25ms, Turn-Off Time: 20us)

I/O Type:

- Type: Sink (Supports both NPN and PNP configurations)

Isolation Protection

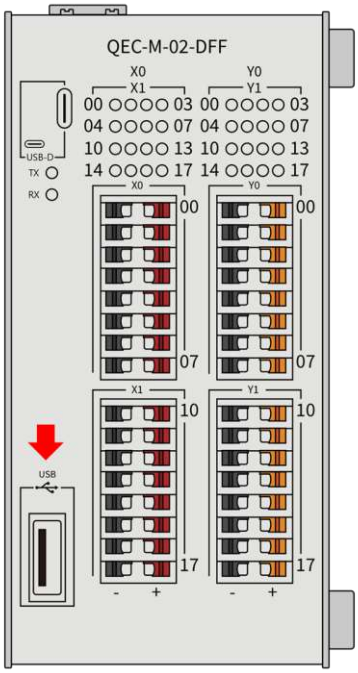
- 1500 Vrms

2.2.7 USB

The QEC-M-02 has a Standard USB 2.0 port with hot-plug support. You can use this port to connect:

- USB Disk: For file storage, transfer, or accessing configuration data.
- Keyboard/Mouse: For HMI display, control, or enter.

The USB port is located on the device's front panel (as shown in the diagram) and is labeled as USB.



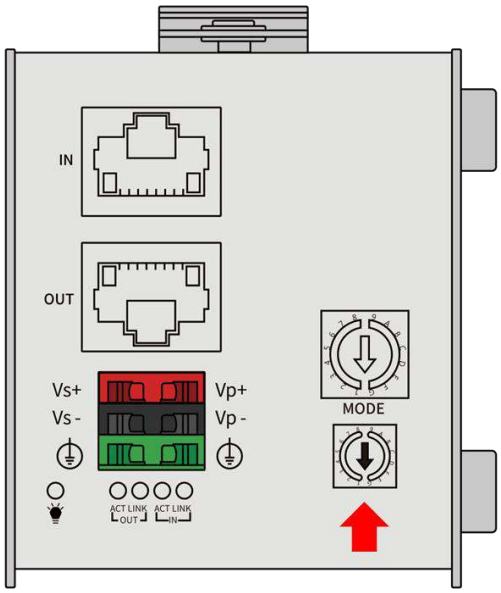
*Notes on USB Disk Usage:

- Ensure the USB disk is formatted in a supported file system (e.g., FAT32 or NTFS).
- Large file transfers may be subject to USB 2.0 speed limitations.

For more details on using USB storage devices, please refer to the Ch 4.8 USB Device Usage Section.

2.2.8 Rotary switch

The Rotary Switch on the QEC-M02 is used to assign device addresses for network communication and operational modes. The switch is labeled from 0 to 9 and A to F, with each memory for each position.

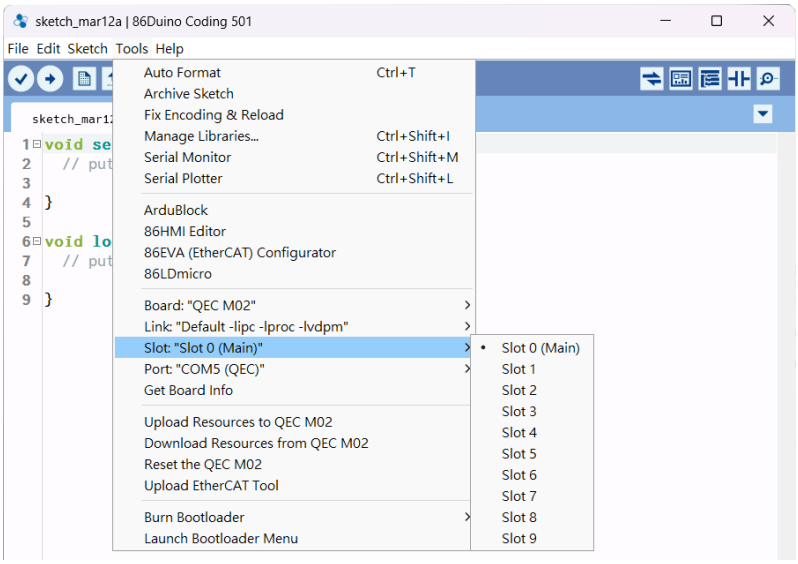


User-Configurable Addresses (0–9)

Positions 0 to 9 are available for user-defined configurations. Users can set the switch to any of these positions to assign a unique address for the device.

- Default Setting: The switch is set to 0 by default, unless otherwise specified.
- Usage: Turn the switch with a small flathead screwdriver to the desired address.

User can use 86Duino Coding IDE 501+ to select the address slot that what to upload the program.



Manufacturer-Reserved Positions (A–F)

Positions A to F are reserved for manufacturer use and are not accessible for user configuration. These positions are utilized for diagnostics, internal testing, and advanced device functions.

* Tampering Warning: Modifying the switch to these positions without manufacturer authorization may cause unexpected behavior or void the warranty.

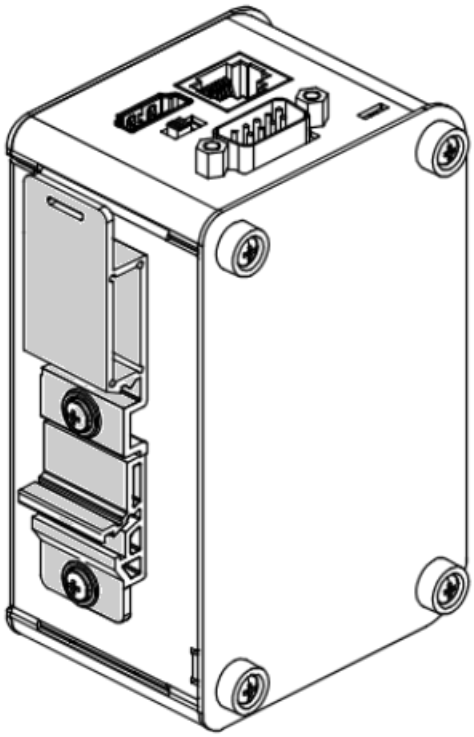
Rotary Switch Operation Steps

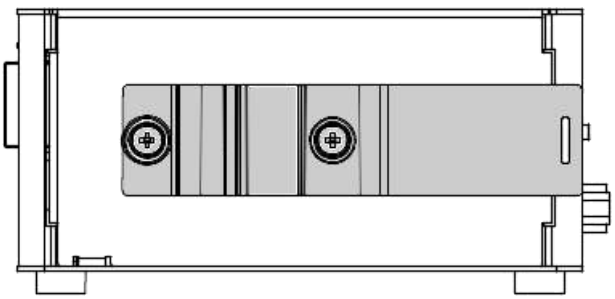
1. Turn off the power before adjusting the rotary switch.
2. Use a small flathead screwdriver to rotate the switch to the desired position.
3. Power on the device and confirm the selected address.

2.2.9 DIN-Rail installation

QEC-M-02 is an easy-install design to help you maintain your modules easily. Please refer to Ch.3.1 DIN-Rail installation.

- **DIN-Rail**

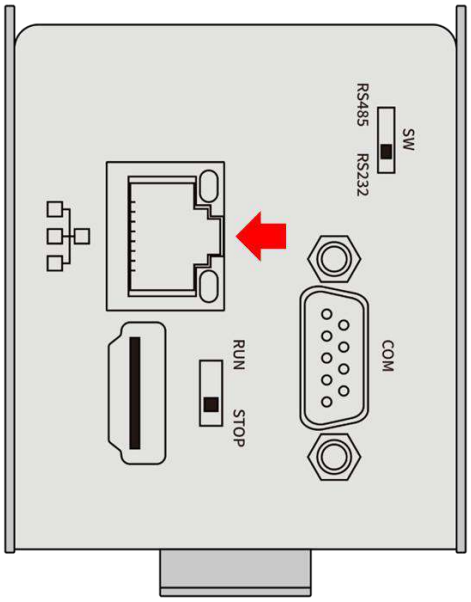




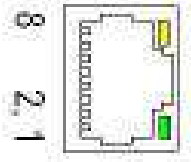
2.2.10 Giga LAN

The QEC-M-02 features one Giga LAN port and is dedicated to external Ethernet communication for general network use.

The Giga LAN port is located on the device's bottom side (as shown in the diagram).



Pin Definitions:



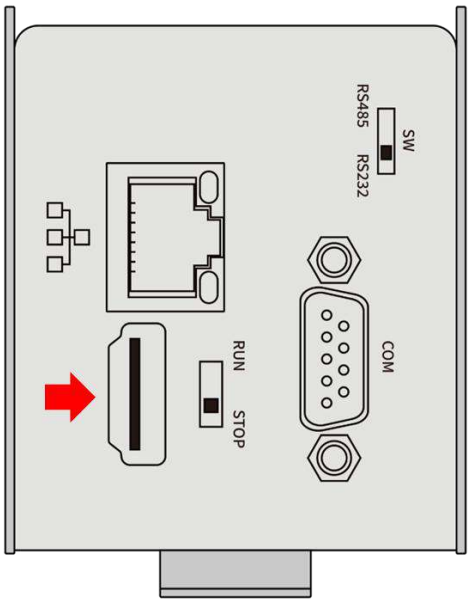
Pin #	Signal Name	Pin #	Signal Name
L1	GTX+	L2	GTX-
L3	GRX+	L4	GTXC+
L5	GTXC-	L6	GRX-
L7	GRXD+	L8	GRXD-

The Giga LAN port supports high-speed Ethernet for external communication. To drive GigaLAN, you can refer to Ethernet library or Modbus Library.

For more details on using Ethernet or Modbus TCP, please refer to the Ch 4.9 Giga LAN Configuration section.

2.2.11 HDMI

The QEC-M-02 is equipped with an HDMI 2.0 connector.



- Display Resolution: 1280 x 720 x 256
- Supported Libraries: Compatible with LVGL and 86HMI Editor for graphical interface design.

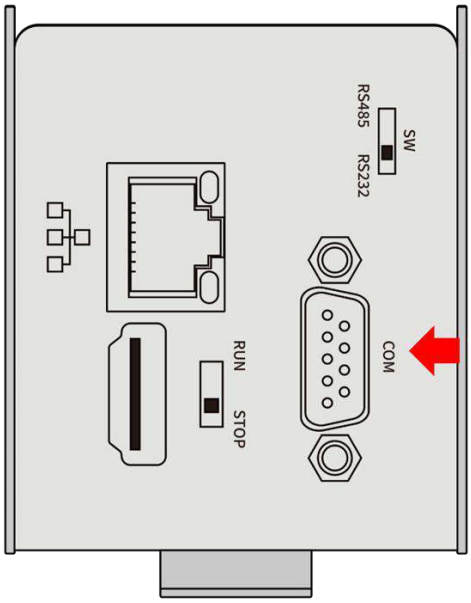
To display content on an HDMI-connected monitor, use the LVGL library or the 86HMI Editor in the 86Duino IDE. These tools allow you to create intuitive and dynamic visual interfaces for your applications.

Ensure the HDMI cable is securely connected to avoid signal loss or display issues.

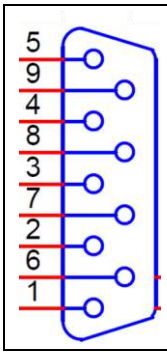
For more details on using HDMI Display, please refer to the Ch 4.11 HDMI Display: LVGL section.

2.2.12 Serial Port

Standard DB9 connector for serial communication interface.
Supports RS232 or RS485; users can use a DIP switch to change the serial communication interface.



Serial Port



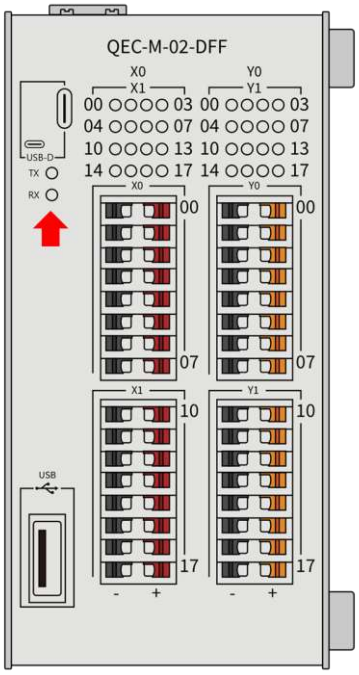
No.	Pin Assignment	No.	Pin Assignment
1	RS485-	6	DSR
2	RS485+/RXD	7	RTS
3	TXD	8	CTS
4	DTR	9	VCC
5	GND	-	-

* Note: RS232 and RS485 cannot be used simultaneously.

The Serial specification table:

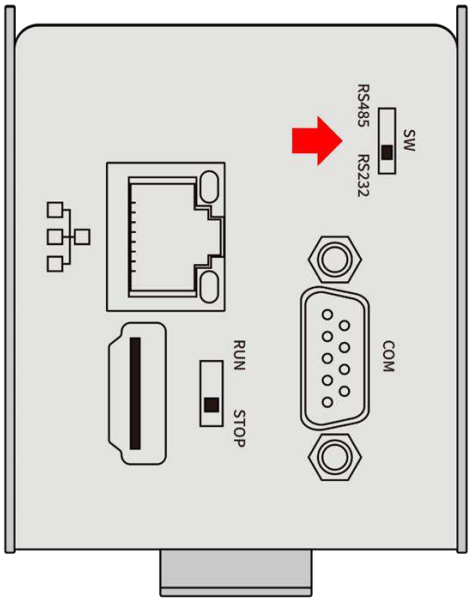
Serial Port	
Interface Mode	RS232/RS485 (D-Sub 9-pin)
Data transfer rate (bps)	2400,4800,9600,14400,19200,38400,57600,115200
Data width (bit)	5/6/7/8
Hardware Flow Control	CTS/RTS

While the Serial Port's signal works, the tx/rx LEDs on the front side will flash.



Switch

DIP switch for changing serial communication to RS232 or RS485.

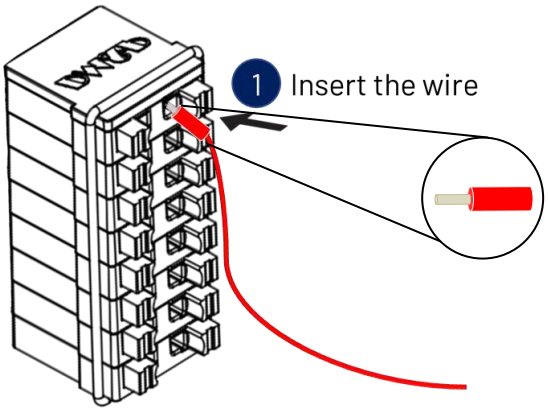


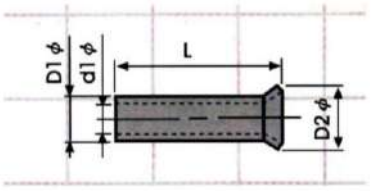
Notation	States
RS485	RS232 Enable
RS232	RS485 Enable

* Note: RS232 and RS485 cannot be used simultaneously.

2.3 Wiring to the Connector

2.3.1 Connecting the wire to the connector

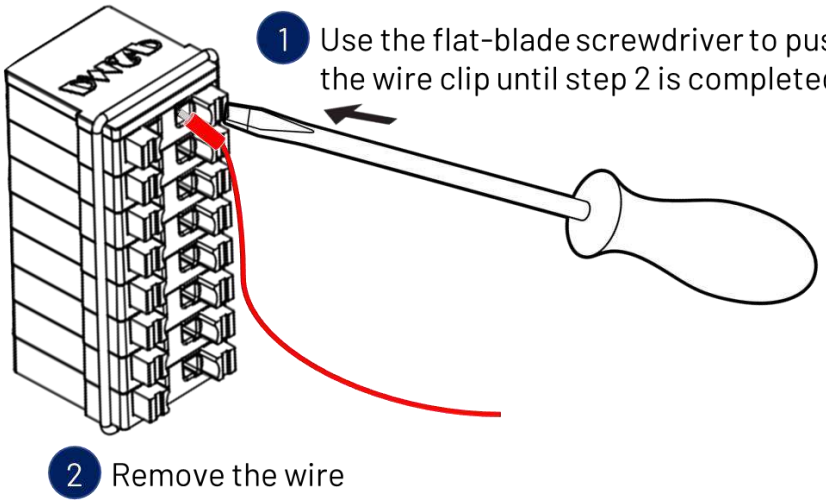




Insulated Terminals Dimensions (mm)

Position	L	ØD1	Ød1	ØD2
CN 0.5-6	6.0	1.3	1.0	1.9
CN 0.5-8	8.0	1.3	1.0	1.9
CN 0.5-10	10.0	1.3	1.0	1.9

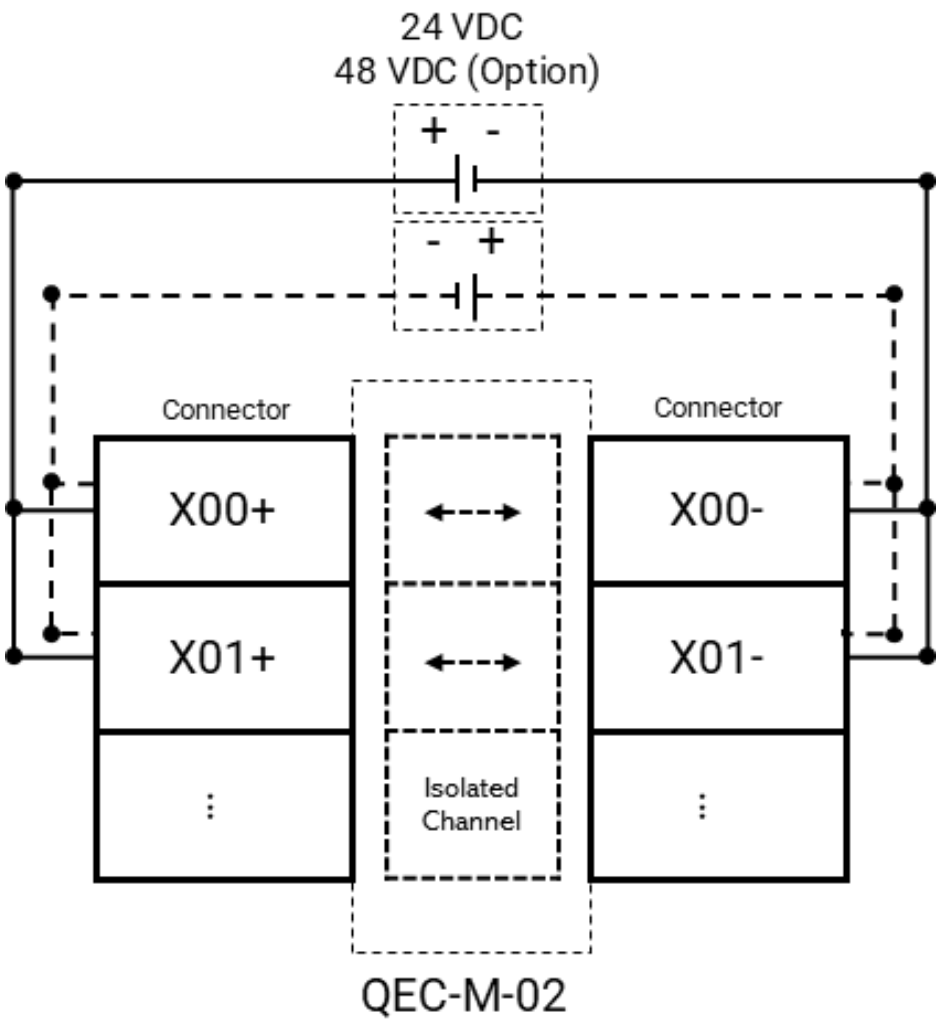
2.3.2 Removing the wire from the connector



2.3.3 Application Wiring

Digital Input

Example for Basic Digital Input Operation.

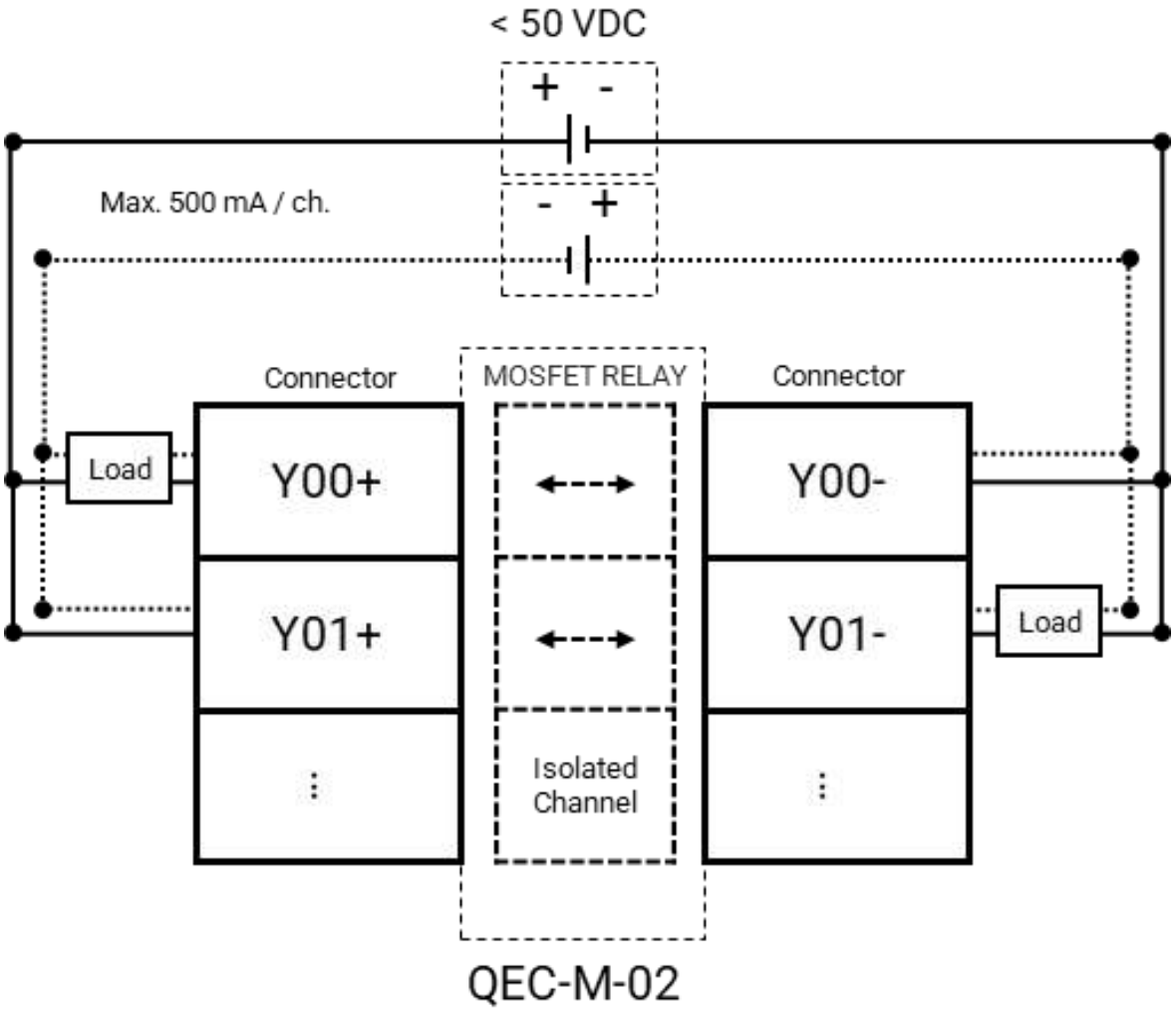


Digital Input Specification:

- Maximum Load Voltage: Load Voltage+24V (Options: 24V or 48V)
- Safe Operating Frequency: ≤3000 Hz (Rise Time (tr): 6us, Fall Time (tf): 6us)
- Type: Sink (Supports both NPN and PNP configurations)
- Isolation Protection: 2500 Vrms

Digital Output

Example for Basic Digital Output Operation.



Digital Output Specification:

- Maximum Load Voltage: +50Vdc
- Typ. 500mA (peak 1000mA)
- Operating Frequency: 1000Hz (Turn-On Time: 0.25ms, Turn-Off Time: 20us)
- Type: Sink (Supports both NPN and PNP configurations)
- Isolation Protection: 1500 Vrms

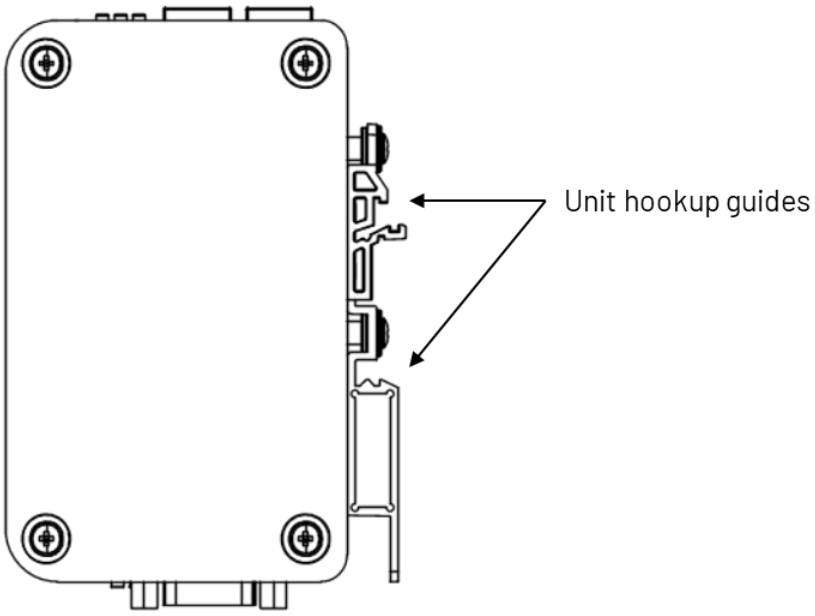
Ch. 3

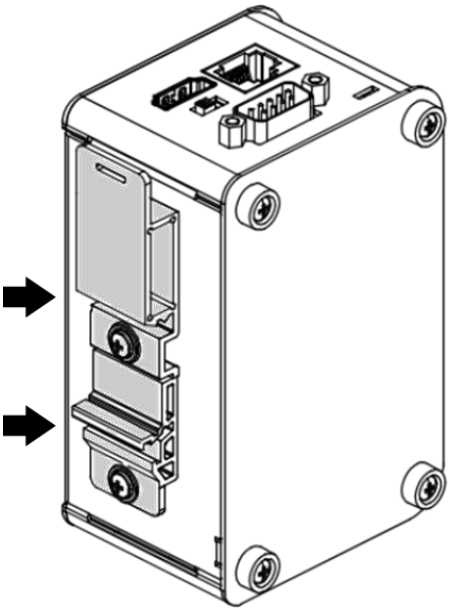
Hardware Installation

This section describes how to install QEC-M-02. Please turn OFF the power supply before you mount QEC-M-02. Always mount QEC-M-02 one at a time.

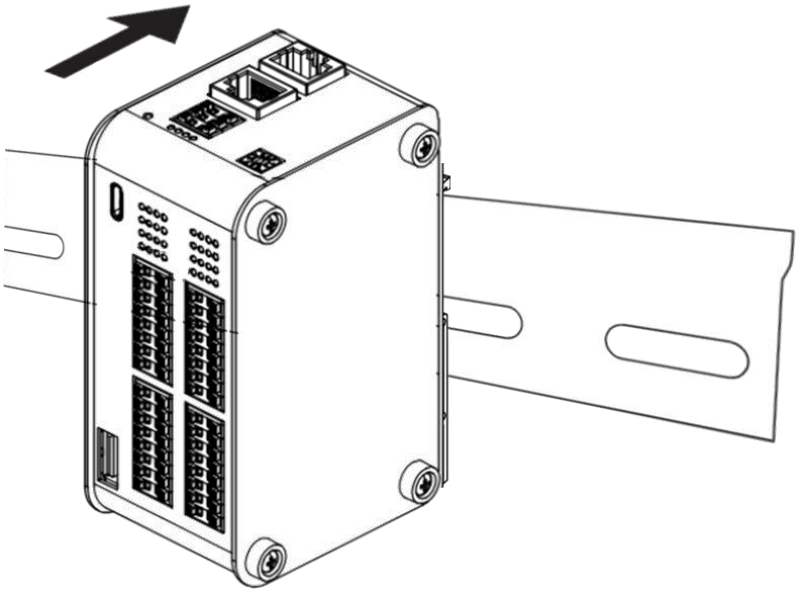
3.1 DIN-Rail installation

Slide in the QEC-M-02 on the hookup guides and press the QEC-M-02 with a certain amount of force against the DIN track until the DIN track mounting hook lock into place.





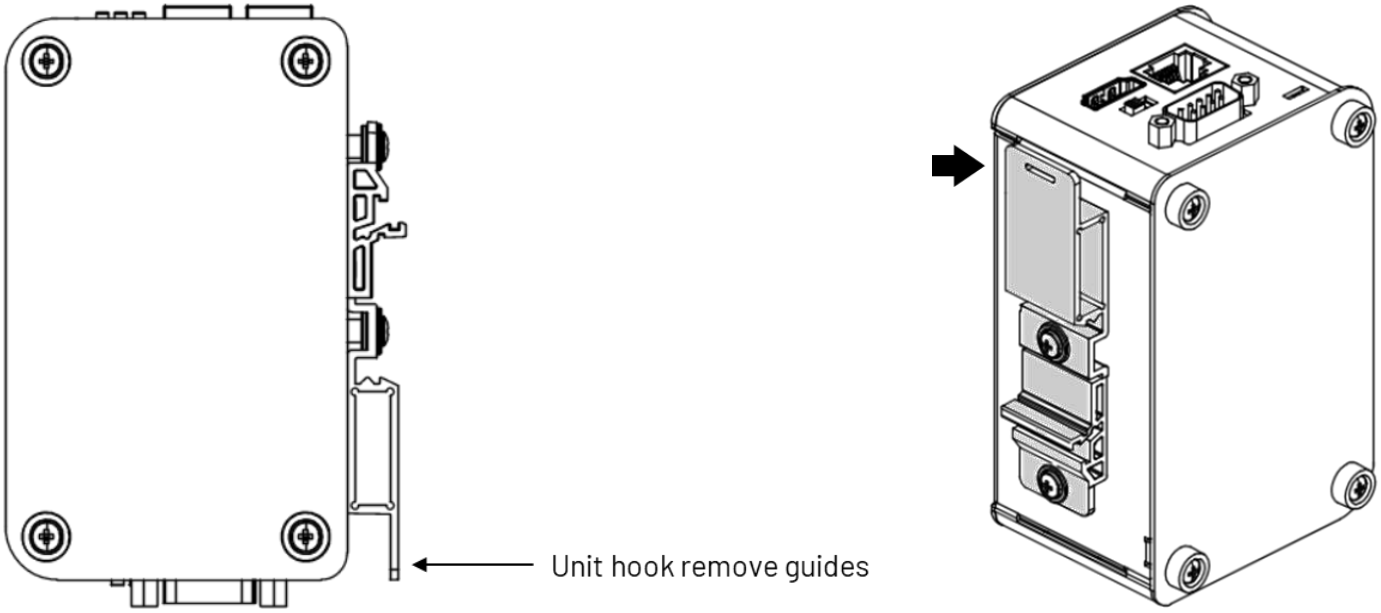
When you mount the QEC-M-02, releasing the DIN track mounting hook on the QEC-M-02 is unnecessary. After you mount the QEC-M-02, make sure it is locked to the DIN track.



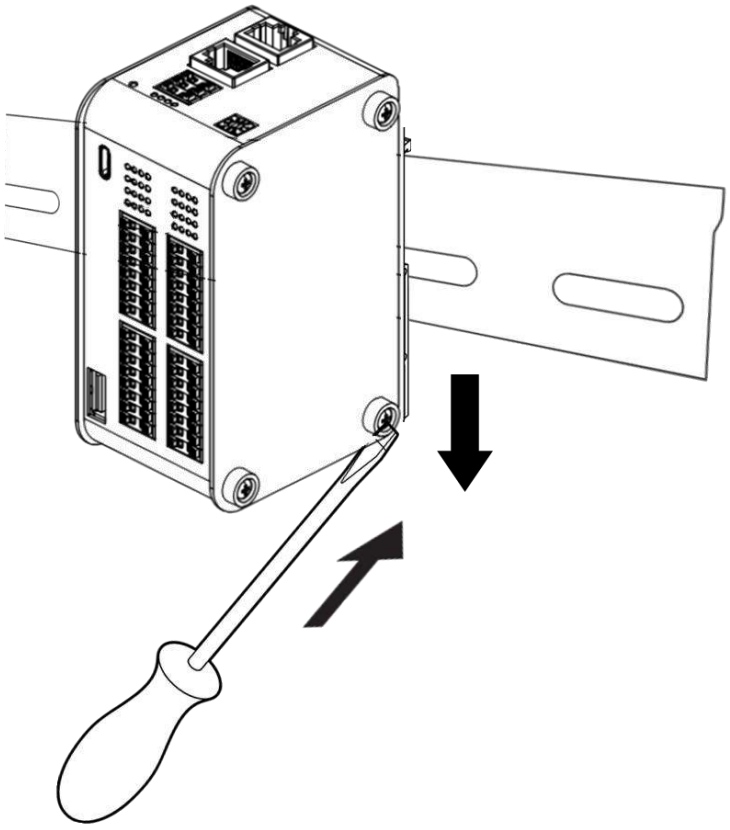
*Note: Always turn OFF the Unit power supply before connecting and removing the QEC-M-02.

3.2 Removing QEC-M-02 Unit

Use a flat-blade screwdriver to remove the DIN Track mounting hook on the unit.



Pull down the flat-blade screwdriver against the DIN track until the mounting hook being removed from the track.



Ch. 4

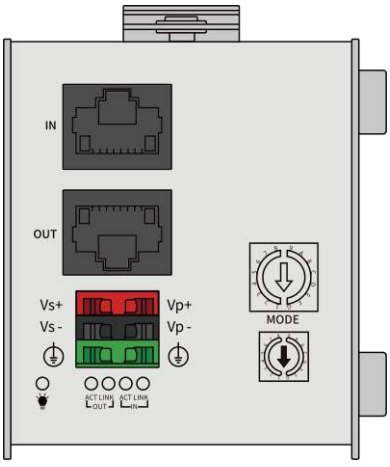
Getting Started

(This chapter is available in multiple languages)

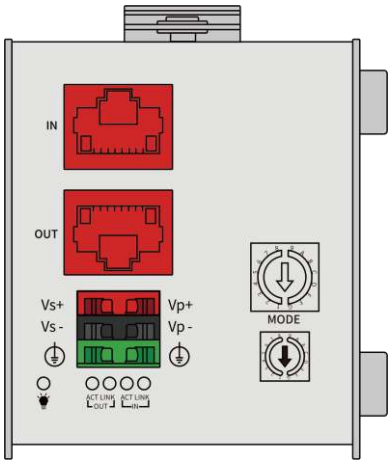
This chapter explains how to start with QEC-M-02 and its software, 86Duino Coding IDE.

Note. QEC's PoE (Power over Ethernet)

In QEC product installations, users can easily distinguish between PoE and non-PoE: if the RJ45 house is red, it is PoE type, and if the RJ45 house is black, it is non-PoE type.



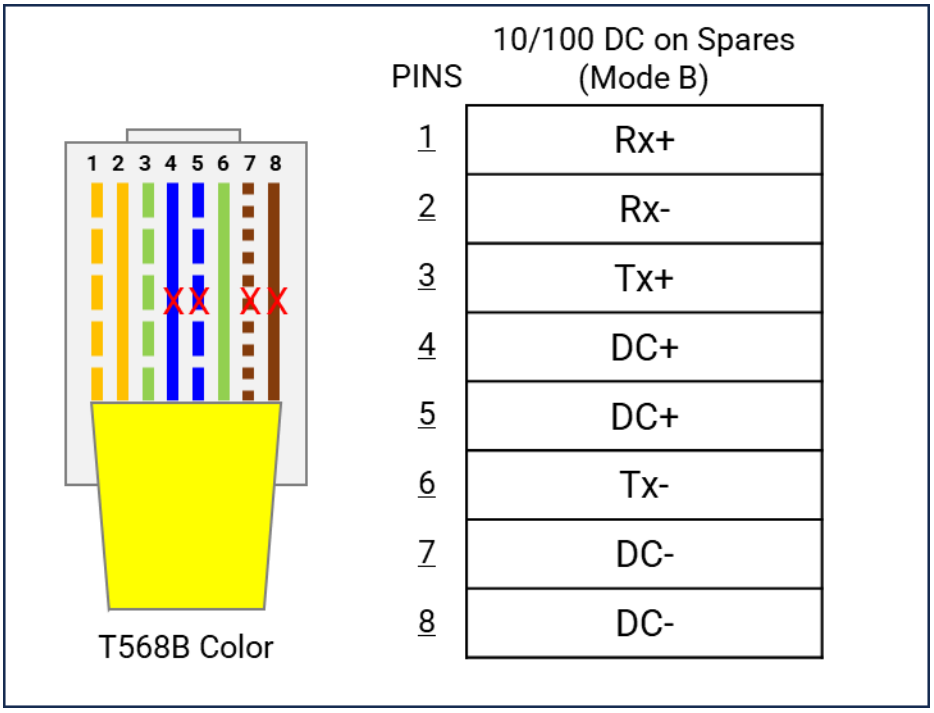
Non-PoE type



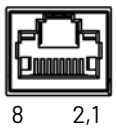
PoE type

PoE (Power over Ethernet) is a function that delivers power over the network. QEC can be equipped with an optional PoE function to reduce cabling. In practice, PoE is selected based on system equipment, so please pay attention to the following points while evaluating and testing:

1. When connecting PoE and non-PoE devices, make sure to disconnect Ethernet cables at pins 4, 5, 7, and 8 (e.g., when a PoE-supported QEC EtherCAT MDevice connects with a third-party EtherCAT SubDevice).



2. The PoE function of QEC is different and incompatible with EtherCAT P, and the PoE function of QEC is based on PoE Type B, and the pin functions are as follows:



Pin #	Signal Name	Pin #	Signal Name
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

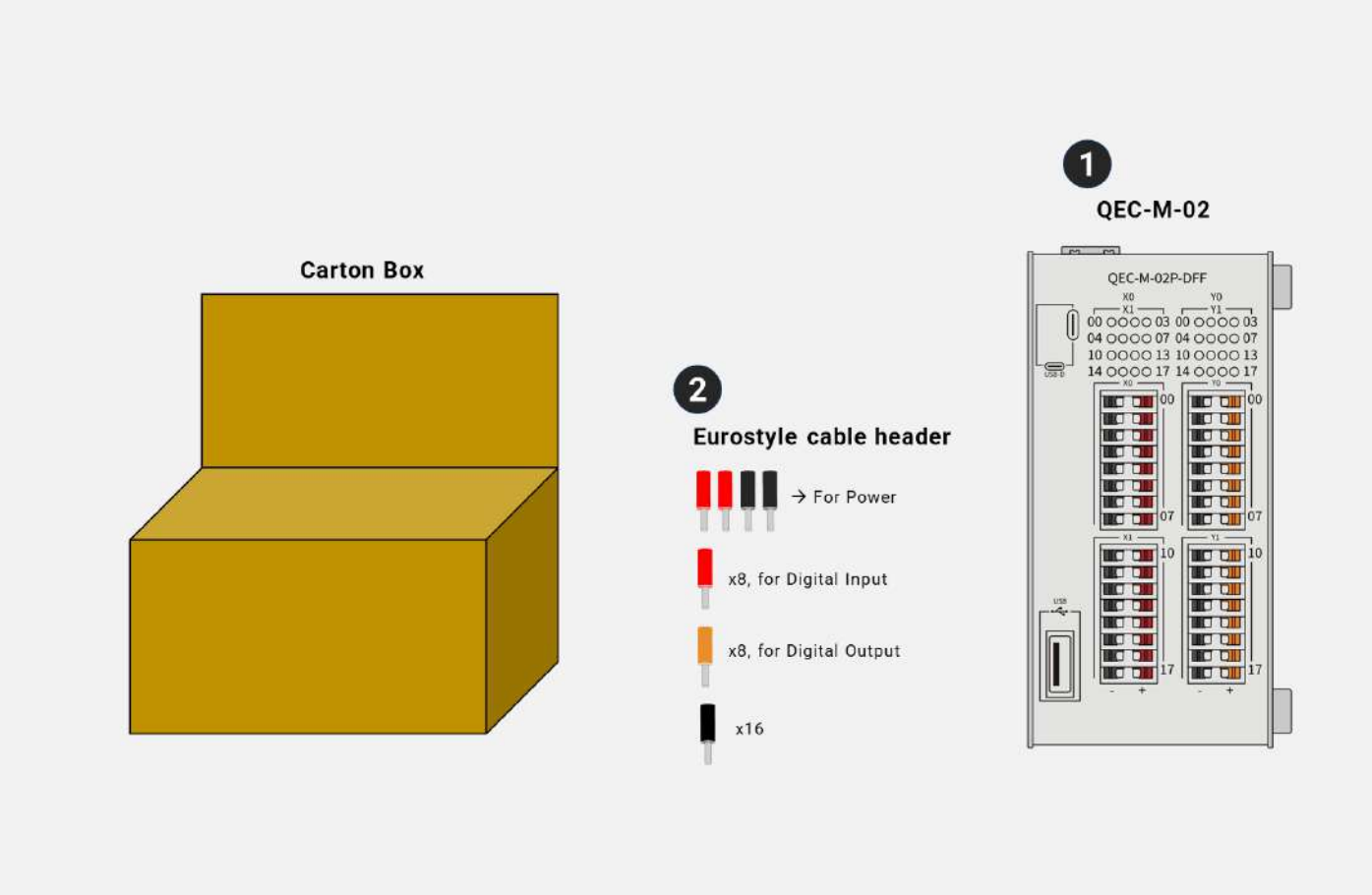
* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

3. QEC's PoE power supply is up to 24V/3A.

4.1 Package Contents

The package includes the following items:

1. QEC-M-02
2. Cable-set (Euro-type connector)

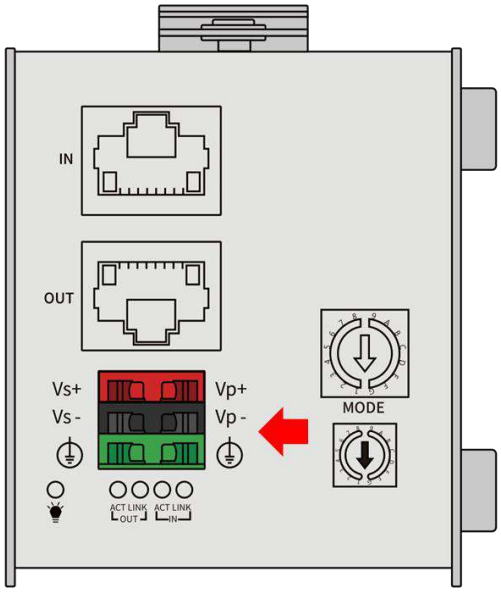


Please get in touch with our sales channels if any of the package items are missing or damaged. Also, feel free to reuse the shipping materials and cartons for further storing and shipping needs in the future.

4.2 Hardware Configuration

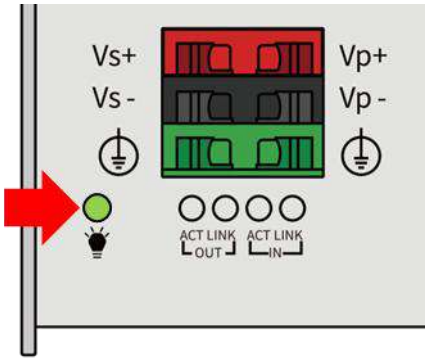
The development environment will be pre-installed before the QEC-M-02 is shipped to customers. Users must download the software (see 4.3 Software/Development Environment) and follow this user manual to set up the device.

Plug in the power supply.



There are two groups of power supplies in QEC-M-02, Vs and Vp. The voltage requirement for both supplies' ranges from 19V to 50V wide voltage.

After powering it on, the power LED lights up (green).



Power input is 24V (typical). The LED status provide high/low voltage warning.

Notation	States	Condition	Description
PWR 	Green LED On	Voltage <= 50V and >= 45V Voltage <= 26V and >= 19V	When Vs and Vp voltages are confirmed to be normal, the Green LED will remain steady on.
	Green LED On Red LED On	Voltage < 45V and > 26V Voltage < 19V and > 12V	LEDs will alternately flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.
	Orange LED On	Voltage > 50V or < 12V	Orange LED (Green + Red) will continuously flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.

* Vs power status will be displayed first.

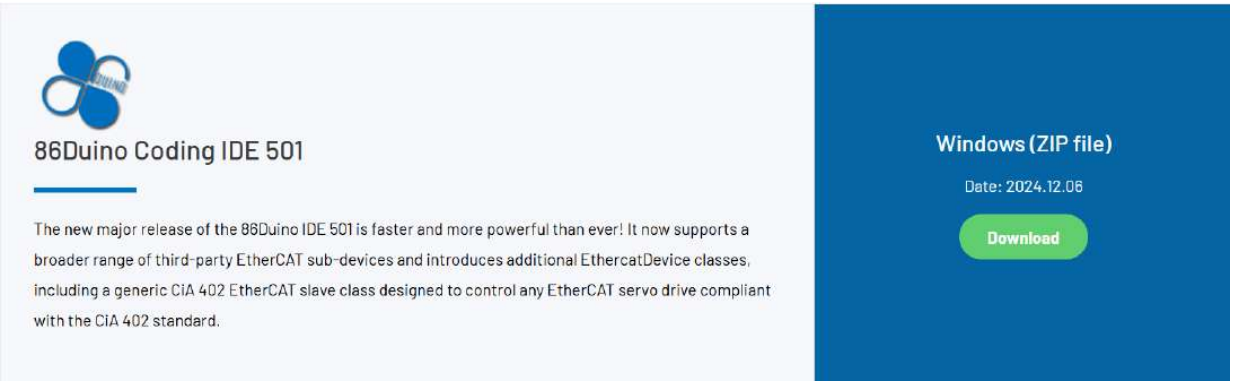
Power ERROR Code table (Red LED Flashing Display (2 seconds/cycle))：

Long Light	Short Flash	Description
0 Long Light	After microchip completes the Bootloader test, it proceeds to the APP program stage.	
	1 short flash	Microchip communication with the EtherCAT chip failed.
	2 short flashes	EtherCAT chip internal RAM test failed.
	5 short flashes	Quartz oscillator on the board abnormality.
1 Long Light	Indicates the microchip Bootloader stage during startup, APP program not yet executed.	
	1 short flash	microchip internal SRAM failed.
	2 short flashes	APP software CHECKSUM failed.
2 Long Lights	Not yet defined.	

* Note: If you encounter any of the above abnormal states, please contact us.

4.3 Software/Development Environment

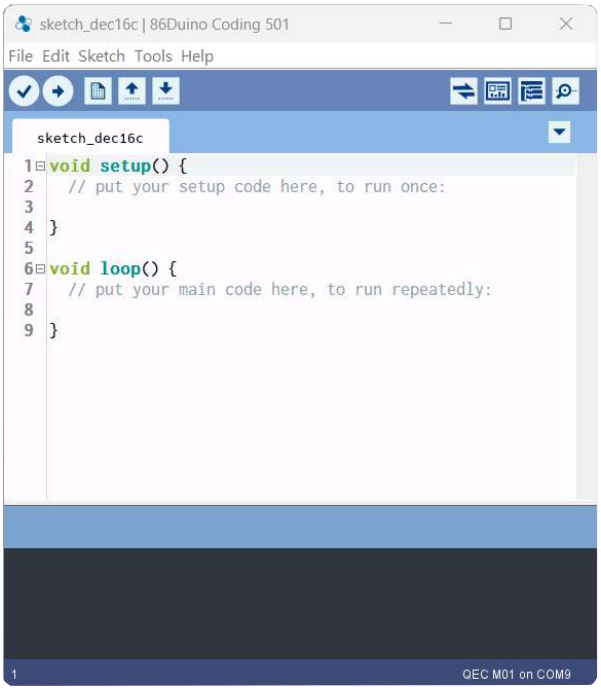
Download 86duino IDE from https://www.qec.tw/software/.



After downloading, please unzip the downloaded zip file, no additional software installation is required, just double-click **86duino.exe** to start the IDE.

* Note: If Windows displays a warning, click Details and then click the Continue Run button once.

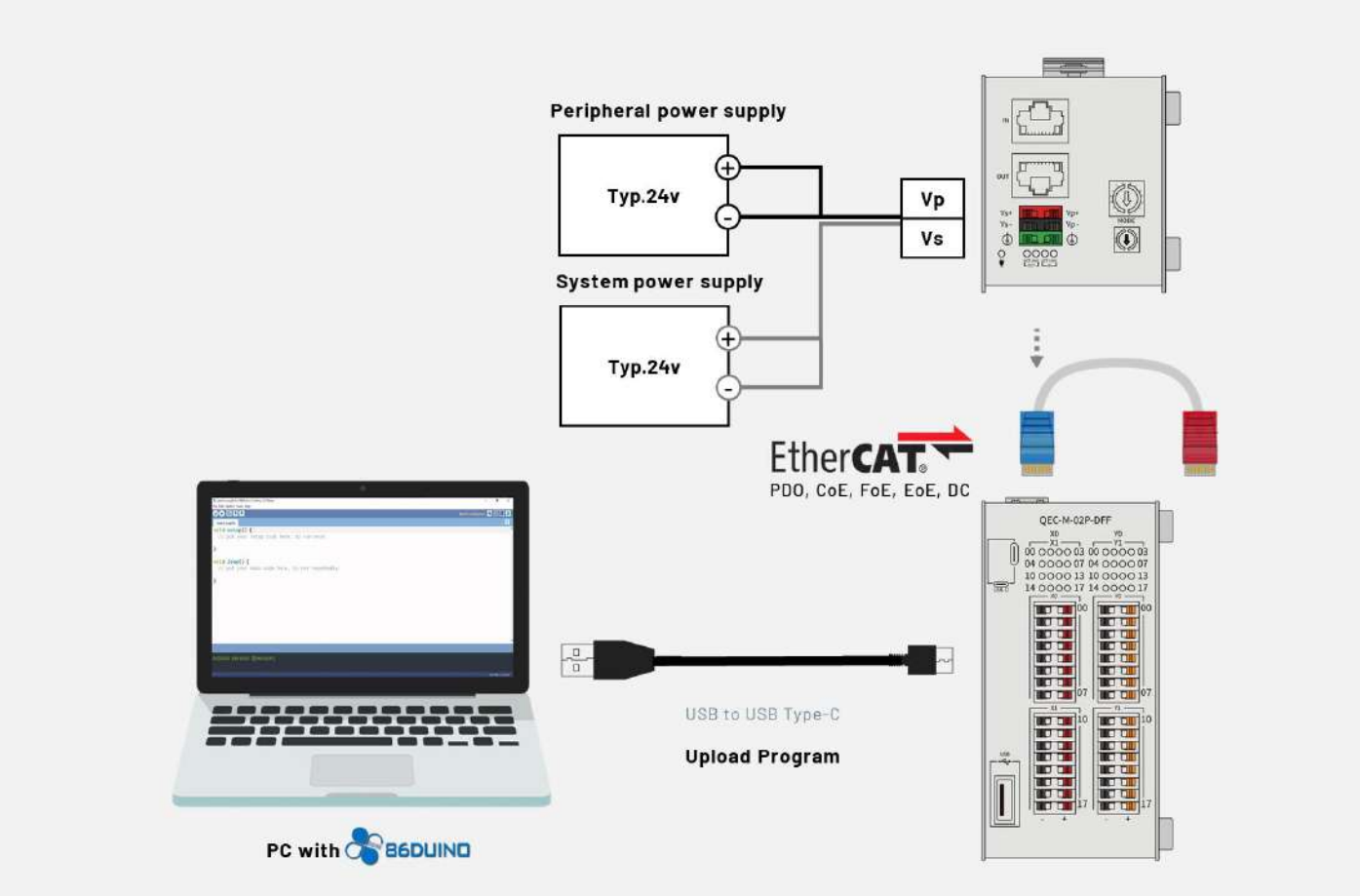
86Duino Coding IDE 501+ looks like below.



4.4 Connect to your PC and set up the environment

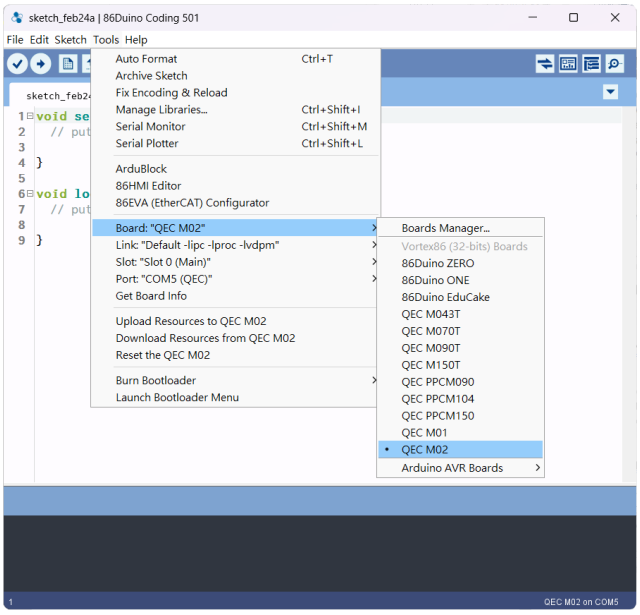
Follow the steps below to set up the environment:

1. Connect the QEC-M-02 to your PC via a USB Type-C to USB cable (86Duino IDE installed).

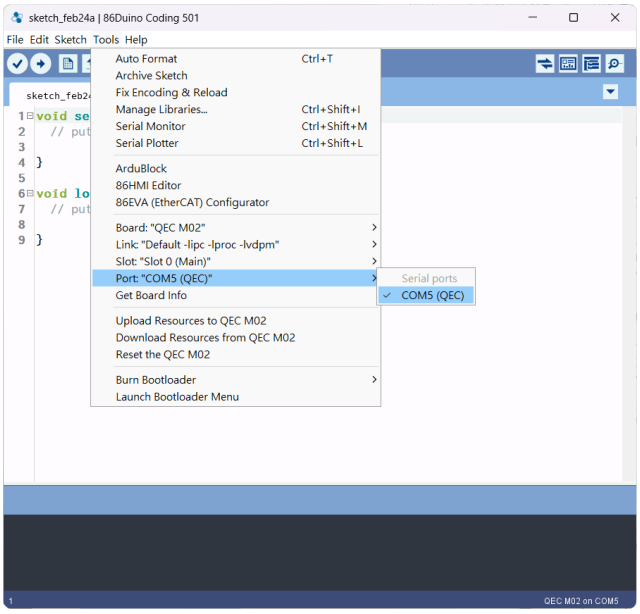


2. Turn on the QEC power.
3. Open “Device Manager” -> “Ports (COM & LPT)” in your PC and expand the ports; you should see that the “Prolific PL2303GC USB Serial COM Port (COMx)” is detected; if not, you will need to install the required drivers.
   (For Windows PL2303 driver, you can download here)

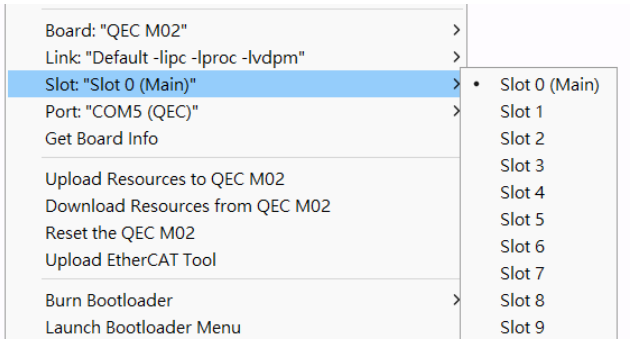
4. Open the 86Duino IDE.
5. Select the correct board: In the IDE's menu, select “Tools” ->” Board” -> “QEC-M-02” (or the QEC-M MDevice model you use).



6. Select Port: In the IDE's menu, select “Tools” -> ”Port” and select the USB port to connect to the QEC-M MDevice (in this case, COM5 (QEC)).



7. Slot Port: In the IDE’s menu, select “Tools” -> “Slot” -> “Slot 0 (Main)” to upload the program to the first main register.



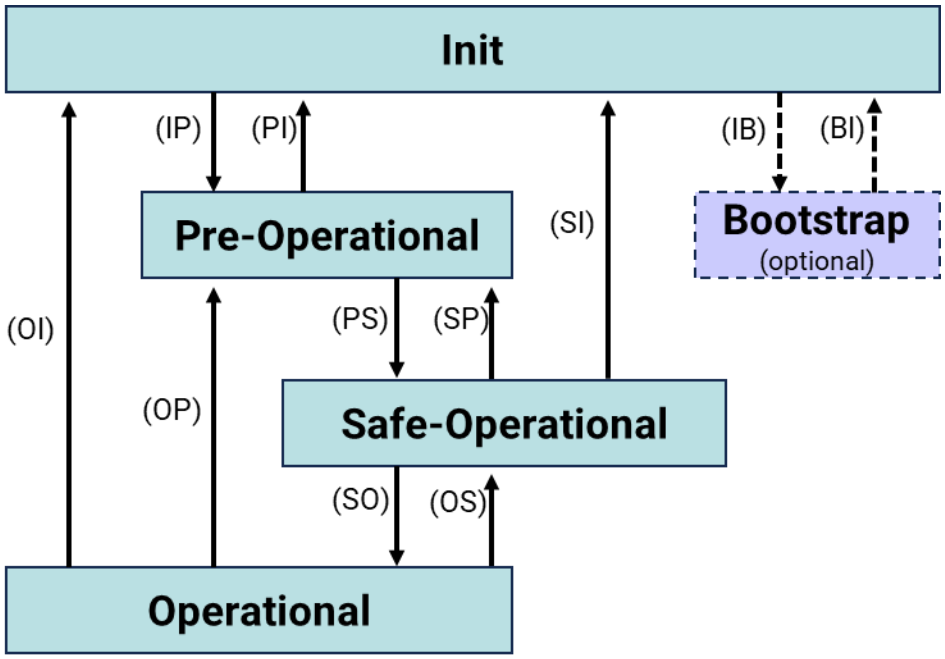
4.5 EtherCAT Communication

This section introduces two primary methods to configure your EtherCAT SubDevice through the QEC EtherCAT MDevice: Write code and Use 86EVA with code. Both methods are designed to offer flexibility and efficiency depending on your familiarity and requirements.

4.5.1 EtherCAT State Machine (ESM) Control

To set up and transition EtherCAT SubDevice through various operational states using the QEC EtherCAT MDevice. It is crucial for understanding the state transitions and operational flow within an EtherCAT network.

The state of the EtherCAT SubDevice is controlled via the EtherCAT State Machine (ESM). Depending upon the state, different functions are accessible or executable in the EtherCAT SubDevice. Specific commands must be sent by the EtherCAT MDevice to the device in each state, particularly during the bootup of the SubDevice.



A distinction is made between the following states:

- Init
- Pre-Operational
- Safe-Operational and
- Operational
- Boot

The regular state of each EtherCAT SubDevice after bootup is the OP state.

The EtherCAT MDevice (QEC-M-02) can be configured in the EtherCAT network via the EtherCAT library and programmed with the control action in the 86Duino IDE. The 86Duino development environment has two main parts: `setup()` and `loop()`, which correspond to initialization and main programs.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly

}
```

Before operating the EtherCAT network, you must configure it once. The process should be from Init to OP state in EtherCAT devices.

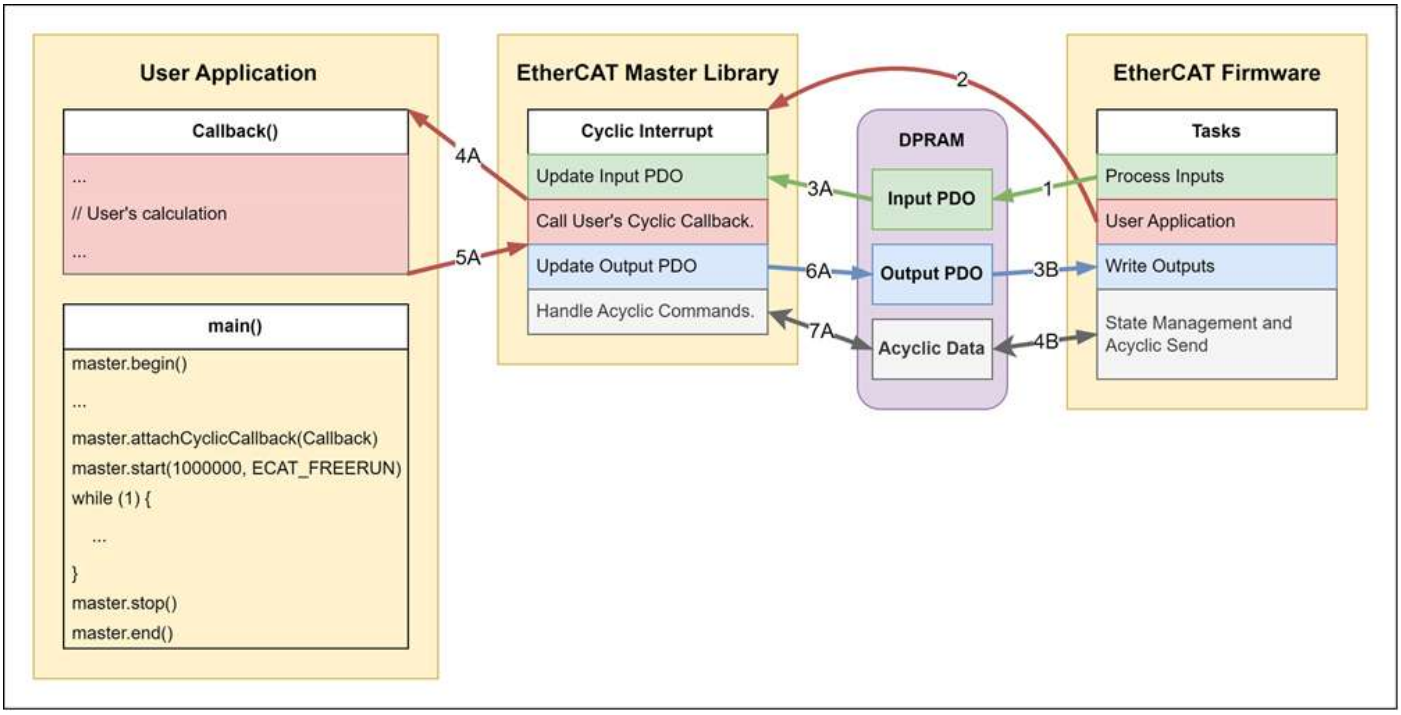
To implement EtherCAT communication, users must use the EtherCAT Library of 86Duino Coding IDE 501+. For detailed information, please refer to Ch.5 Software Function.

*Note: Once the code is written, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload.

Example 1: Free Run Mode

In "Free Run" mode the local cycle is triggered through a local timer interrupt of the application controller. The cycle time can be modified by the MDevice (optional) in order to change the timer interrupt. In "Free Run" mode the local cycle operates independent of the communication cycle and/or the MDevice cycle.

The Free Run Mode diagram for QEC MDevice:



This is the free-run mode without dual-system synchronization. As shown in the diagram, the numbered arrows indicate the sequence of operations. However, a branching occurs at step 3 because, after the EtherCAT firmware triggers a cyclic interrupt to the EtherCAT MDevice library (step 2), it does not wait for the EtherCAT MDevice library and directly continues with the next action. The two systems operate independently, with no synchronization. If the user has registered a cyclic callback, the cyclic interrupt will call it, as shown in step 4A.

Here is the example code:

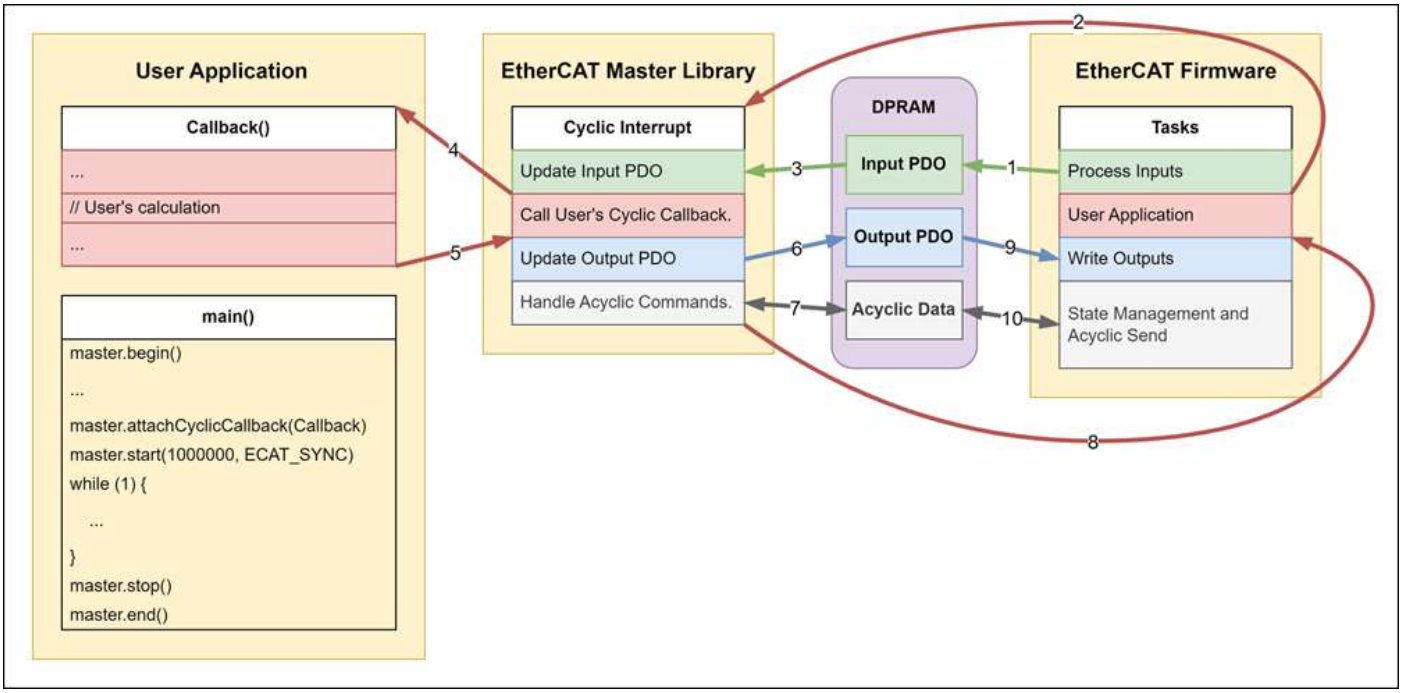
```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

void setup() {
  master.begin();
  master.attachCyclicCallback(CyclicCallback);
  master.start(1000000, ECAT_FREERUN);
}

void loop() {
  // ...
}
```

Example 2: SYNC Mode

The local cycle is started when the SM2 event [with cyclical outputs] or the SM3 event [without cyclical outputs] is received. If the outputs are available, the SubDevice is generally synchronized with the SM2 event. If no outputs are available, the SubDevice is synchronized with the SM3 event, e.g. for cyclical inputs.
In this mode the following options are available:

- Synchronous with SM2/3 event
- Synchronous with SM2/3 event, shifting of the "Input Latch" time

The SYNC Mode diagram for QEC MDevice:



This mode offers the highest level of dual-system synchronization. As shown in the diagram, the numbered arrows indicate the sequence of operations, with no branching present. After the EtherCAT firmware triggers a cyclic interrupt to the EtherCAT MDevice library (step 2), it waits for an ACK response from the EtherCAT MDevice library (step 8) before proceeding with the next action. If the user has registered a cyclic callback, the cyclic interrupt will call it, as shown in step 4. As long as the user reads the current input process data within the cyclic callback, processes it, calculates the output process data, and writes it back, the current cycle will send the output process data to the EtherCAT network, fulfilling the requirements of real-time control systems.

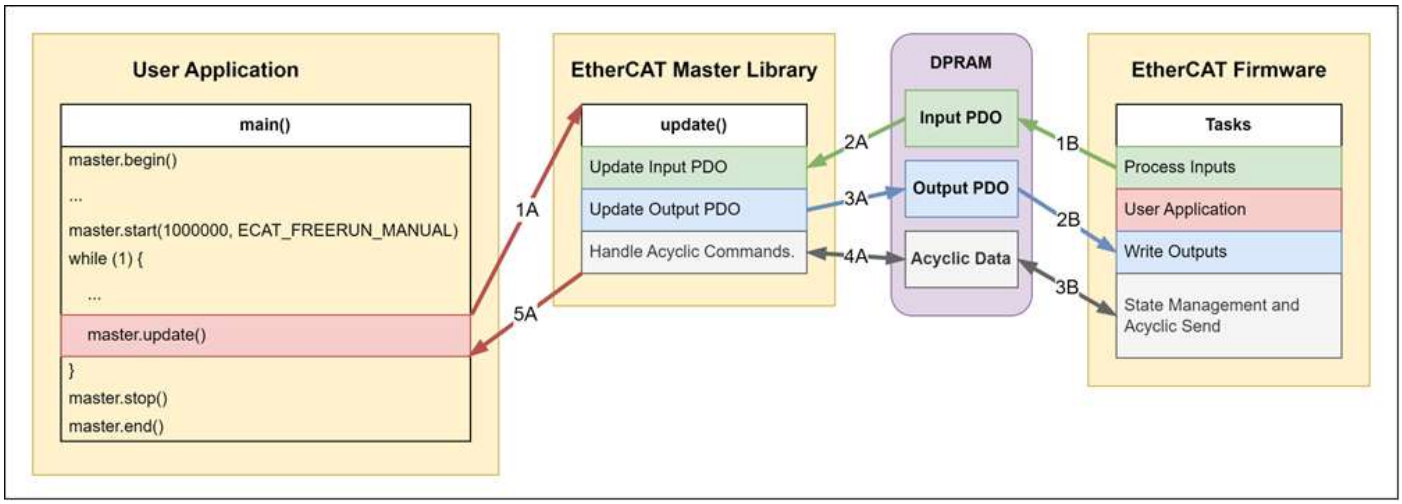
Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

void setup() {
  master.begin();
  master.attachCyclicCallback(CyclicCallback);
  master.start(1000000, ECAT_SYNC);
}

void loop() {
  // ...
}
```

Example 3: Free Run Manual Mode

This is also a free-run mode without dual-system synchronization. The primary difference from the ECAT_FREERUN mode is that there is no cyclic interrupt to update process data and handle acyclic commands. Instead, the user must manually call `EthercatMaster::update()` to update process data and handle acyclic commands. Additionally, since there is no cyclic interrupt in this mode, the cyclic callback will not be called. As indicated by the numbered arrows in the diagram, the two systems operate independently, with no synchronization.

The Free Run Manual Mode diagram for QEC MDevice:



Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

void setup() {
  master.begin();
  master.start(1000000, ECAT_FREERUN_MANUAL);
}

void loop() {
  // ...
  master.update();
}
```

4.5.2 SubDevice Information

The QEC MDevice's EtherCAT library provides functions to obtain information about EtherCAT SubDevice devices on the network. These include querying the number of SubDevices on the network, retrieving a SubDevice's Vendor ID, Product Code, and Alias Address by its sequence number, and reverse querying the SubDevice number using the information above. This information is used to identify the type of SubDevice and choose the appropriate EtherCAT SubDevice class to attach.

Example 1: Using EthercatMaster class

Show SubDevice information using EthercatMaster class.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.begin(); // Initialize EtherCAT Master in Pre-OP state

  Serial.println("Starting EtherCAT Master...");

  // Print Out All Slave Information
  for (int i = 0; i < master.getSlaveCount(); i++) {
    Serial.print("Slave ");
    Serial.print(i);
    Serial.print(" VID: ");
    Serial.print(master.getVendorID(i), HEX);
    Serial.print(", PID: ");
    Serial.print(master.getProductCode(i), HEX);
    Serial.print(", Rev: ");
    Serial.print(master.getRevisionNumber(i), HEX);
    Serial.print(", Ser: ");
    Serial.print(master.getSerialNumber(i), HEX);
    Serial.print(", Alias: ");
    Serial.print(master.getAliasAddress(i));
```

```
    Serial.println();
  }
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 2: Using EthercatDevice_Generic class

Show SubDevice information using EthercatDevice_Generic class.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

char name[256];

void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.begin(); // Initialize EtherCAT Master in Pre-Operational state

  for (int i = 0; i < master.getSlaveCount(); i++) {
    slave.attach(i, master); // Attach the slave to the master
    Serial.print("Slave ");
    Serial.println(i);

    Serial.print("               Name: ");
    Serial.println(slave.getDeviceName(name, 256));

    Serial.print("          Vendor ID: 0x");
    Serial.println(slave.getVendorID(), HEX);

    Serial.print("       Product Code: 0x");
    Serial.println(slave.getProductCode(), HEX);

    Serial.print("    Revision Number: 0x");
    Serial.println(slave.getRevisionNumber(), HEX);

    Serial.print("      Serial Number: 0x");
    Serial.println(slave.getSerialNumber(), HEX);

    Serial.print("      Alias Address: ");
    Serial.println(slave.getAliasAddress());
```

```
    Serial.print("    Mailbox Protocol: 0x");
    Serial.println(slave.getMailboxProtocol(), HEX);

    Serial.print("        CoE Details: 0x");
    Serial.println(slave.getCoEDetails(), HEX);

    Serial.print("        FoE Details: 0x");
    Serial.println(slave.getFoEDetails(), HEX);

    Serial.print("        EoE Details: 0x");
    Serial.println(slave.getEoEDetails(), HEX);

    Serial.print("       SoE Channels: ");
    Serial.println(slave.getSoEChannels());

    Serial.print("       DC Supported: ");
    Serial.println(slave.isSupportDC());
  }
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 3: Using EthercatDevice_CiA402 class

Show SubDevice information using EthercatDevice_CiA402 class.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 slave;

char name[256];

void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.begin();  // Initialize EtherCAT Master in Pre-Operational state

  for (int i = 0; i < master.getSlaveCount(); i++) {
    // Attach the slave to the master
    if (slave.attach(i, master) < 0) {
      continue; // Skip this slave if attachment fails
    }

    Serial.print("Slave ");
    Serial.println(i);

    Serial.print("               Name: ");
    Serial.println(slave.getDeviceName(name, 256));

    Serial.print("          Vendor ID: 0x");
    Serial.println(slave.getVendorID(), HEX);

    Serial.print("       Product Code: 0x");
    Serial.println(slave.getProductCode(), HEX);

    Serial.print("    Revision Number: 0x");
    Serial.println(slave.getRevisionNumber(), HEX);

    Serial.print("      Serial Number: 0x");
    Serial.println(slave.getSerialNumber(), HEX);
```

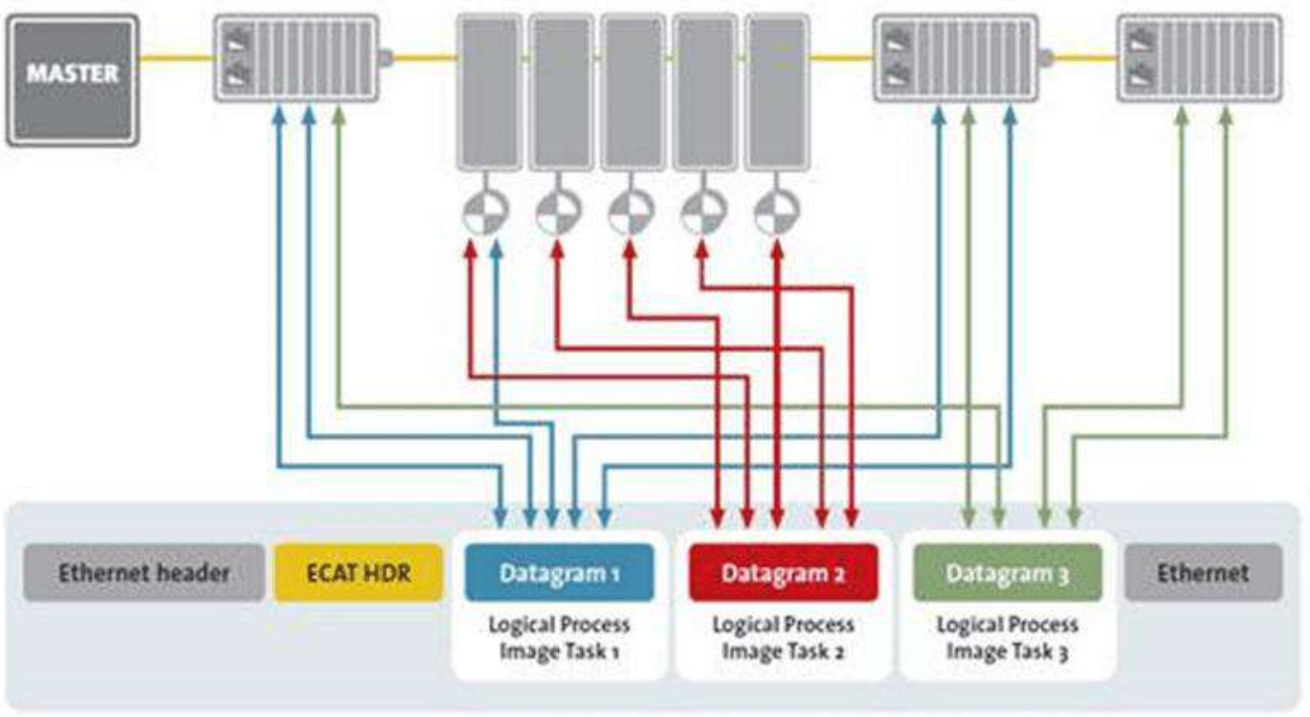
```
    Serial.print("      Alias Address: ");
    Serial.println(slave.getAliasAddress());

    Serial.print("    Mailbox Protocol: 0x");
    Serial.println(slave.getMailboxProtocol(), HEX);

    Serial.print("        CoE Details: 0x");
    Serial.println(slave.getCoEDetails(), HEX);

    Serial.print("        FoE Details: 0x");
    Serial.println(slave.getFoEDetails(), HEX);

    Serial.print("        EoE Details: 0x");
    Serial.println(slave.getEoEDetails(), HEX);

    Serial.print("       SoE Channels: ");
    Serial.println(slave.getSoEChannels());

    Serial.print("       DC Supported: ");
    Serial.println(slave.isSupportDC());
  }
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

4.5.3 Process Data Objects (PDO) Functions

Process Data refers to real-time communication data exchanged between the MDevice and Sub-device in an EtherCAT network. This data includes information used for control, monitoring, and communication purposes. The EtherCAT MDevice cyclically transmits process data to control and monitor all SubDevices, ensuring high synchronization and low latency.

The Fieldbus Memory Management Units (FMMU) in the EtherCAT SubDevice Controller (ESC) can mapping dual-port memory to logical address. All SubDevice nodes check the EtherCAT frames sent by the EtherCAT MDevice, comparing the logical address of the process data with the configured address in the FMMU. If a match is found, the output process data is transferred to dual-port memory, and the input process data is inserted into the EtherCAT frame.

Overall, process data is an essential part of EtherCAT technology and is suitable for real-time applications in robot control, CNC control, automation control, and other fields.



EtherCAT: Exchange of internet packets and data. (Source of information: http://www.ethercat.org/)

Example 1: Read a bit data from Input PDO

Read a bit from Input PDO using `pdoBitRead()`.
A 16-channel digital input EtherCAT SubDevice has 2-byte Input PDOs, with each bit corresponding to a digital input channel. The states of channels 0 and 9 will be printed with a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
    Serial.begin(115200); // Initialize Serial Monitor
    while (!Serial);      // Wait for Serial Monitor to be ready

    master.begin();       // Initialize EtherCAT Master
    slave.attach(0, master); // Attach the first EtherCAT slave to the master
    master.start();       // Start EtherCAT communication
}

void loop() {
    // Read and display the state of PDO bits
    Serial.print("Bit0 => ");
    Serial.print(slave.pdoBitRead(0)); // Read PDO bit 0
    Serial.print(", Bit9 => ");
    Serial.println(slave.pdoBitRead(9)); // Read PDO bit 9

    delay(1000); // Wait for 1 second
}
```

Example 2: Read a byte data from Input PDO

Read data from Input PDO using `pdoRead8()`.
A 16-channel digital input EtherCAT SubDevice has 2-byte Input PDOs, with each bit corresponding to a digital input channel. The states of channels 0 and 9 will be printed with a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  Serial.begin(115200); // Initialize Serial Monitor
  while (!Serial);      // Wait for Serial Monitor to be ready

  master.begin();       // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master
  master.start();       // Start EtherCAT communication
}

void loop() {
  // Read specific bits using pdoRead8 and print their values
  Serial.print("Bit0 => ");
  Serial.print((slave.pdoRead8(0) >> 0) & 1); // Read PDO byte 0, extract bit 0
  Serial.print(", Bit9 => ");
  Serial.println((slave.pdoRead8(1) >> 1) & 1); // Read PDO byte 1, extract bit 9

  delay(1000); // Wait for 1 second before reading again
}
```

Example 3: Write a bit data to Output PDO

Write a bit to Output PDO using `pdoBitWrite()`.
A 16-channel digital output EtherCAT SubDevice has 2-byte Output PDOs, with each bit corresponding to a digital output channel. Channels 0 and 9 will be toggled at a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();       // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master
  master.start();       // Start EtherCAT communication
}

void loop() {
  // Set Bit 0 and Bit 9 to 1
  slave.pdoBitWrite(0, 1); // Write 1 to Bit 0
  slave.pdoBitWrite(9, 1); // Write 1 to Bit 9
  delay(1000);             // Wait for 1 second

  // Set Bit 0 and Bit 9 to 0
  slave.pdoBitWrite(0, 0); // Write 0 to Bit 0
  slave.pdoBitWrite(9, 0); // Write 0 to Bit 9
  delay(1000);             // Wait for 1 second
}
```

Example 4: Write a byte data to Output PDO

Write data to Output PDO using `pdoWrite8()`.
A 16-channel digital output EtherCAT SubDevice has 2-byte Output PDOs, with each bit corresponding to a digital output channel. Channels 0 and 9 will be toggled at a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
    master.begin();       // Initialize EtherCAT Master
    slave.attach(0, master);
    master.start();       // Start EtherCAT communication
}

void loop() {
    // Write 0x01 to PDO byte 0 and 0x02 to PDO byte 1
    slave.pdoWrite8(0, 0x01); // Set byte 0 to 0x01
    slave.pdoWrite8(1, 0x02); // Set byte 1 to 0x02
    delay(1000);              // Wait for 1 second

    // Write 0x00 to PDO byte 0 and byte 1
    slave.pdoWrite8(0, 0x00); // Set byte 0 to 0x00
    slave.pdoWrite8(1, 0x00); // Set byte 1 to 0x00
    delay(1000);              // Wait for 1 second
}
```

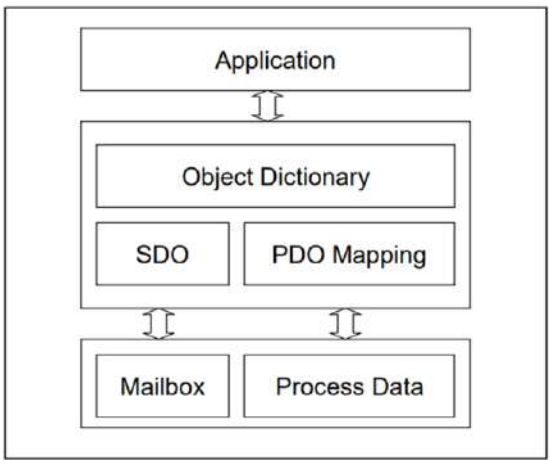
4.5.4 CANopen over EtherCAT (CoE) Functions

CoE (CAN application over EtherCAT) is a CANopen protocol based on the EtherCAT network. It enables communication using the CANopen protocol over EtherCAT networks. The Object Dictionary contains parameters, application data and the mapping information between process data interface and application date (PDO mapping). Its entries can be accessed via Service Data Objects (SDO).

CANopen is a high-level communication protocol based on the Controller Area Network (CAN) bus, commonly used for communication between control systems and devices in industrial applications. It defines a set of communication objects, data types, and network management functions to facilitate data exchange, configuration, and control between devices.

The CANopen protocol includes the following aspects:

- Object Dictionary
  Defines all data objects and parameters exchanged between devices. The object dictionary encompasses various types of objects such as variables, parameters, events, and functions.
- PDO (Process Data Object)
  Used for real-time data transmission. PDOs allow devices to transmit data between each other in a fixed or event-triggered manner, enabling real-time control and data exchange.
- SDO (Service Data Object)
  Used for configuring and managing device parameters. SDOs provide functionalities for reading, writing, and parameter configuration, allowing devices to dynamically exchange configuration information.



The SDO services primarily consist of two types of commands. The SDO command is utilized for accessing objects stored in the Object Dictionary, while the SDO information command is employed to retrieve details about these objects.

Example 1: SDO Upload

SDO Upload using `sdoUpload8()`.
The usage of `sdoUpload16()`, `sdoUpload32()`, and `sdoUpload64()` is similar to `sdoUpload8()`, except for the difference in the return value type.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.begin();       // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master

  // Read and print the value of SDO 0x1C12.0
  Serial.print("1C12h.0 => ");
  Serial.println(slave.sdoUpload8(0x1C12, 0x00)); // Upload 8-bit SDO data from
0x1C12.0

  // Read and print the value of SDO 0x1C13.0
  Serial.print("1C13h.0 => ");
  Serial.println(slave.sdoUpload8(0x1C13, 0x00)); // Upload 8-bit SDO data from
0x1C13.0
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 2: SDO Upload with abort code

SDO Upload using `sdoUpload()` with abort code.
Initiate an SDO Upload command to read a value from a non-existent object, expecting an abort code of 0x06020000. For more information about abort codes, please refer to SDO Abort Code.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t abortcode;   // Variable to store the abort code
uint8_t value;       // Variable to store the uploaded value

void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.begin();      // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master

  // Attempt to upload SDO data
  if (slave.sdoUpload(0xFFFF, 0xFF, &value, sizeof(value), &abortcode) ==
ECAT_ERR_DEVICE_COE_ERROR) {
    Serial.print("Abort Code: 0x");
    Serial.println(abortcode, HEX); // Print the abort code in hexadecimal format
  }
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 3: SDO Download

SDO Download using `sdoDownload8()`.
The usage of `sdoDownload16()`, `sdoDownload32()`, and `sdoDownload64()` is similar to `sdoDownload8()`, except for the difference in the input parameter types.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);

  slave.sdoDownload8(0x1C12, 0x00, 0);
  slave.sdoDownload8(0x1C13, 0x00, 0);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 4: SDO Download with abort code

SDO Download using `sdoDownload()` with abort code.
Initiate an SDO Download command to write a value to a non-existent object, expecting an abort code of 0x06020000. For more information about abort codes, please refer to SDO Abort Code.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t abortcode;
uint8_t value;

void setup() {
  Serial.begin(115200);

  master.begin();
  slave.attach(0, master);

  if (slave.sdoDownload(0xFFFF, 0xFF, &value, sizeof(value), &abortcode) ==
ECAT_ERR_DEVICE_COE_ERROR) {
    Serial.print("Abort Code: 0x");
    Serial.println(abortcode, HEX); // Print the abort code in hexadecimal format
  }
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 5: Print the PDO mapping configuration

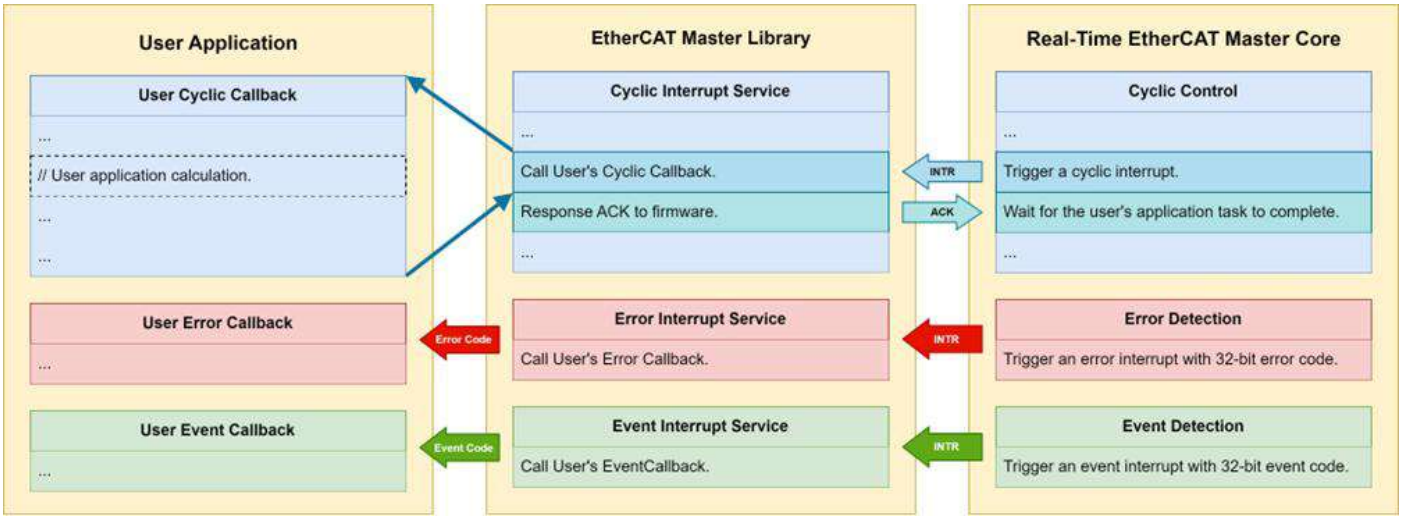
Print the PDO mapping configuration.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t assign_nr, mapping_nr;
uint16_t mapping;
uint32_t entry;

void setup() {
  Serial.begin(115200);

  master.begin();
  slave.attach(0, master);

  // Process RxPDO Assignments
  assign_nr = slave.sdoUpload8(0x1C12, 0x00);
  for (int m = 0; m < assign_nr; m++) {
    mapping = slave.sdoUpload16(0x1C12, m + 1);
    Serial.print("  RxPDO");
    Serial.print(m + 1);
    Serial.print(" (");
    Serial.print(mapping, HEX);
    Serial.println("h)");

    mapping_nr = slave.sdoUpload8(mapping, 0x00);
    for (int n = 0; n < mapping_nr; n++) {
      entry = slave.sdoUpload32(mapping, n + 1);
      Serial.print("    ");
      Serial.print(entry, HEX);
      Serial.println("h");
    }
  }

  // Process TxPDO Assignments
  assign_nr = slave.sdoUpload8(0x1C13, 0x00);
```

```
  for (int m = 0; m < assign_nr; m++) {
    mapping = slave.sdoUpload16(0x1C13, m + 1);
    Serial.print("  TxPDO");
    Serial.print(m + 1);
    Serial.print(" (");
    Serial.print(mapping, HEX);
    Serial.println("h)");

    mapping_nr = slave.sdoUpload8(mapping, 0x00);
    for (int n = 0; n < mapping_nr; n++) {
      entry = slave.sdoUpload32(mapping, n + 1);
      Serial.print("    ");
      Serial.print(entry, HEX);
      Serial.println("h");
    }
  }
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

4.5.5 Cyclic Callback Functions



This library provides three types of callbacks as follows:

- **Cyclic Callback**
  The purpose of the Cyclic Callback is to allow users to implement periodic control systems such as motion control, CNC control, and robot control. The Real-Time EtherCAT MDevice Core triggers cyclic interrupts to the EtherCAT MDevice Library at specified cycle time, then waiting for an ACK to ensure process data synchronization. If a user has registered a Cyclic Callback, it will be invoked to achieve periodic control.

- **Error Callback**
  When the Real-Time EtherCAT MDevice Core detects an error, it will trigger an error interrupt and pass a 32-bit error code to the EtherCAT MDevice Library. If the user has registered an error callback, the system will invoke that callback to inform the user of the specific error.

  The error codes supported by the Error Callback are as follows:

Definition	Code	Description
ECAT_ERR_WKC_SINGLE_FAULT	2000001	Working counter fault occurred.
ECAT_ERR_WKC_MULTIPLE_FAULTS	2000002	Multiple working counter faults occurred.
ECAT_ERR_SINGLE_LOST_FRAME	2000003	Frame was lost.
ECAT_ERR_MULTIPLE_LOST_FRAMES	2000004	Frames were lost multiple times.
ECAT_ERR_CABLE_BROKEN	2000007	The cable is broken.
ECAT_ERR_WAIT_ACK_TIMEOUT	2001000	Firmware timeout waiting for cyclic interrupt ACK.

- **Event Callback**
  When the Real-Time EtherCAT MDevice Core detects an event, it triggers an event interrupt and passes a 32-bit event code to the EtherCAT MDevice Library. If the user has registered an event callback, the system will invoke that callback to inform the user of the specific event.
  The event codes supported by the Event Callback are as follows:

Definition	Code	Description
ECAT_EVT_STATE_CHANGED	1000001	The EtherCAT state of the MDevice has changed.
ECAT_EVT_CABLE_RECONNECTED	1000002	The cable has been reconnected.

Example 1: Cyclic callback

A 16-channel digital output EtherCAT SubDevice has 2-byte Output PDOs, with each bit corresponding to a digital output channel. Channels 0 and 9 will be toggled at a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;
int toggle = 0;       // Toggle variable
int cycle_count = 0;  // Cycle count variable

void myCallback() {
  if (++cycle_count < 1000) // Increment and check the cycle count
    return;
  cycle_count = 0;         // Reset cycle count

  toggle = !toggle;        // Toggle the state
  slave.pdoBitWrite(0, toggle); // Write the toggle value to Bit 0
  slave.pdoBitWrite(9, toggle); // Write the toggle value to Bit 9
}

void setup() {
  master.begin();       // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master

  master.attachCyclicCallback(myCallback); // Attach cyclic callback
  master.start(1000000); // Start EtherCAT Master with 1 ms cycle time
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Example 2: Error callback

Print the count of each type of error once per second.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

// Error counters
int wkc_single_fault_cnt = 0;
int wkc_multiple_faults_cnt = 0;
int single_lost_frame_cnt = 0;
int multiple_lost_frames_cnt = 0;
int cable_broken_cnt = 0;
int wait_ack_timeout_cnt = 0;

// Error callback function
void myErroCallback(uint32_t errorcode) {
  switch (errorcode) {
    case ECAT_ERR_WKC_SINGLE_FAULT:
      wkc_single_fault_cnt++;
      break;
    case ECAT_ERR_WKC_MULTIPLE_FAULTS:
      wkc_multiple_faults_cnt++;
      break;
    case ECAT_ERR_SINGLE_LOST_FRAME:
      single_lost_frame_cnt++;
      break;
    case ECAT_ERR_MULTIPLE_LOST_FRAMES:
      multiple_lost_frames_cnt++;
      break;
    case ECAT_ERR_CABLE_BROKEN:
      cable_broken_cnt++;
      break;
    case ECAT_ERR_WAIT_ACK_TIMEOUT:
      wait_ack_timeout_cnt++;
      break;
  }
}
```

```
void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.attachErrorCallback(myErroCallback); // Attach error callback
  master.begin();      // Initialize EtherCAT Master
  master.start();      // Start EtherCAT communication
}

void loop() {
  // Print error counts to Serial Monitor
  Serial.print("ECAT_ERR_WKC_SINGLE_FAULT     = ");
  Serial.println(wkc_single_fault_cnt);
  Serial.print("ECAT_ERR_WKC_MULTIPLE_FAULTS  = ");
  Serial.println(wkc_multiple_faults_cnt);
  Serial.print("ECAT_ERR_SINGLE_LOST_FRAME    = ");
  Serial.println(single_lost_frame_cnt);
  Serial.print("ECAT_ERR_MULTIPLE_LOST_FRAMES = ");
  Serial.println(multiple_lost_frames_cnt);
  Serial.print("ECAT_ERR_CABLE_BROKEN        = ");
  Serial.println(cable_broken_cnt);
  Serial.print("ECAT_ERR_WAIT_ACK_TIMEOUT     = ");
  Serial.println(wait_ack_timeout_cnt);

  delay(1000); // Wait 1 second before the next print
}
```

Example 3: Event callback

Print the count of each type of event once per second.

Here is the example code:

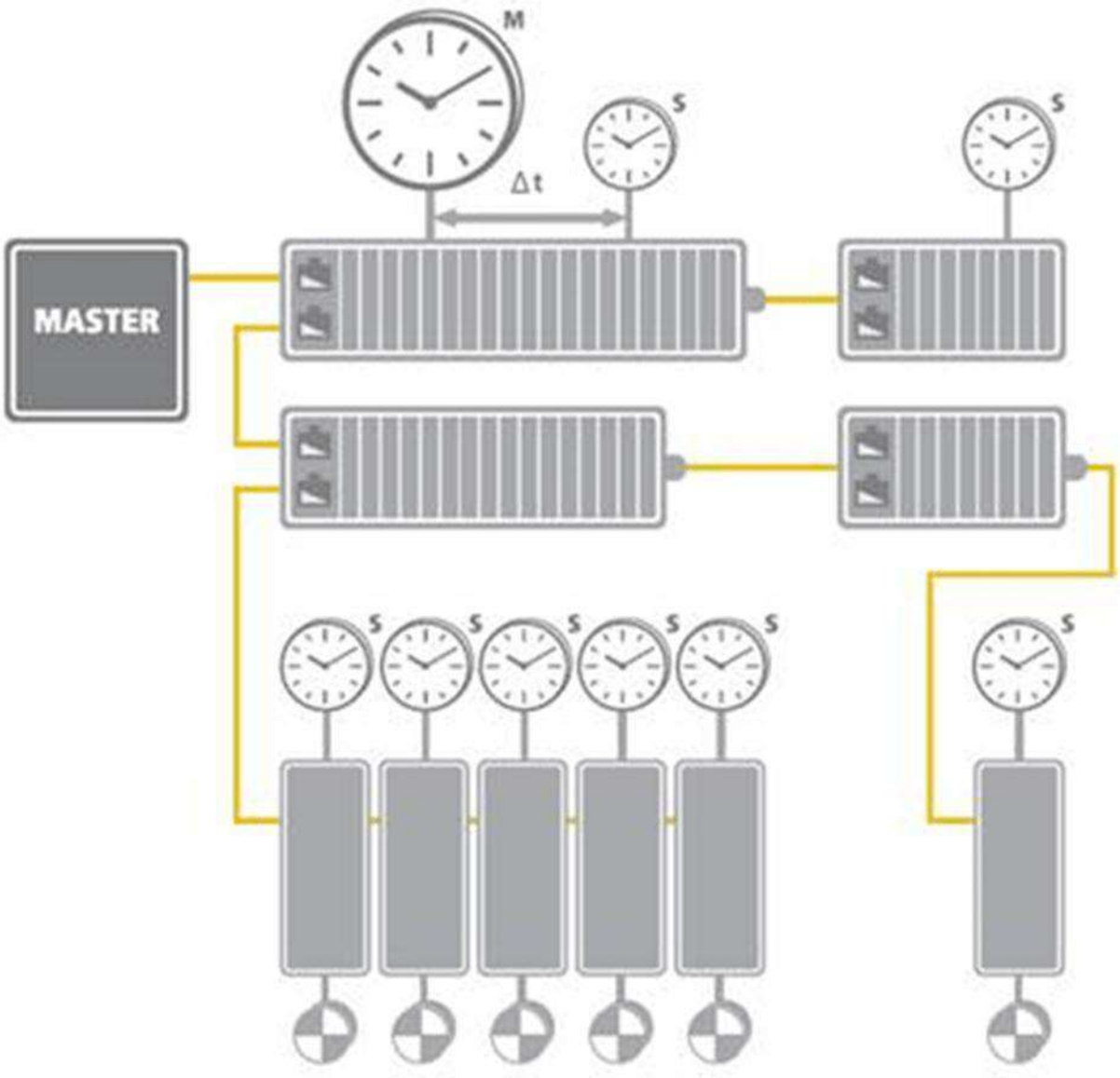
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

// Event counters
int state_changed_cnt = 0;
int cable_reconnected_cnt = 0;

// Event callback function
void myEventCallback(uint32_t eventcode) {
  switch (eventcode) {
    case ECAT_EVT_STATE_CHANGED:
      state_changed_cnt++;
      break;
    case ECAT_EVT_CABLE_RECONNECTED:
      cable_reconnected_cnt++;
      break;
  }
}

void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.attachEventCallback(myEventCallback); // Attach event callback
  master.begin();      // Initialize EtherCAT Master
  master.start();      // Start EtherCAT communication
}

void loop() {
  // Print event counts to Serial Monitor
  Serial.print("ECAT_EVT_STATE_CHANGED    = ");
  Serial.println(state_changed_cnt);
  Serial.print("ECAT_EVT_CABLE_RECONNECTED = ");
  Serial.println(cable_reconnected_cnt);
```

```
  delay(1000); // Wait 1 second before the next update
}
```

4.5.6 Distributed Clock (DC) Configuration Functions

In applications with spatially distributed processes requiring simultaneous actions, exact synchronization is particularly important. For example, this is the case for applications in which multiple servo axes execute coordinated movements.

In contrast to completely synchronous communication, whose quality suffers immediately from communication errors, distributed synchronized clocks have a high degree of tolerance for jitter in the communication system. Therefore, the EtherCAT solution for synchronizing nodes is based on such distributed clocks (DC).

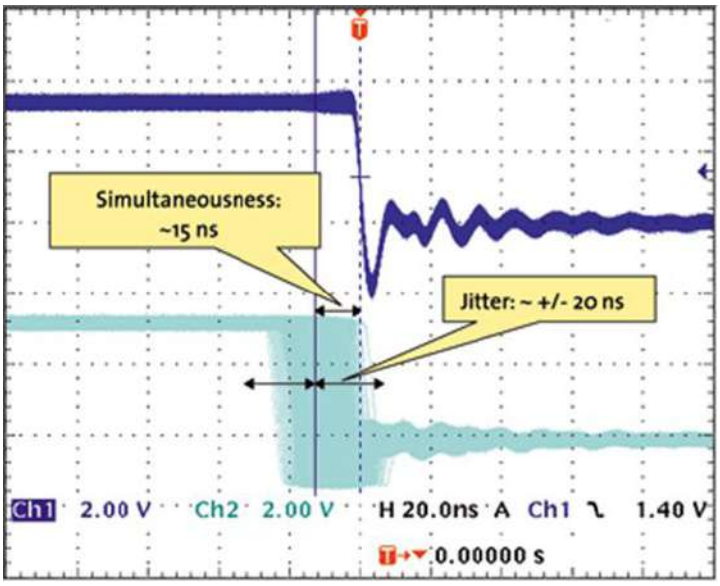


EtherCAT: Illustration of Distributed Clock (DC). (Source of information: http://www.ethercat.org/)

The synchronization mechanism of EtherCAT is based on IEEE-1588 Precision Clock Synchronization Protocol and extends the definition to so-called Distributed-clock (DC). To put it simple, every EtherCAT ESC maintains a hardware-based clock and the minimum time interval is 1 nano-second (64 bits in total). The time maintained by EC-SubDevice is called Local system time.

With accurate internet time synchronization mechanism and dynamic time compensation mechanism (*1), EtherCAT DC technology can guarantee that the time difference among every EC-SubDevice local system time is within +/- 20 nano-seconds. The following diagram is a scope view of two SubDevices' output digital signals. We can see that the time difference between the I/O signal from two EC-SubDevices is around 20 nano-seconds.

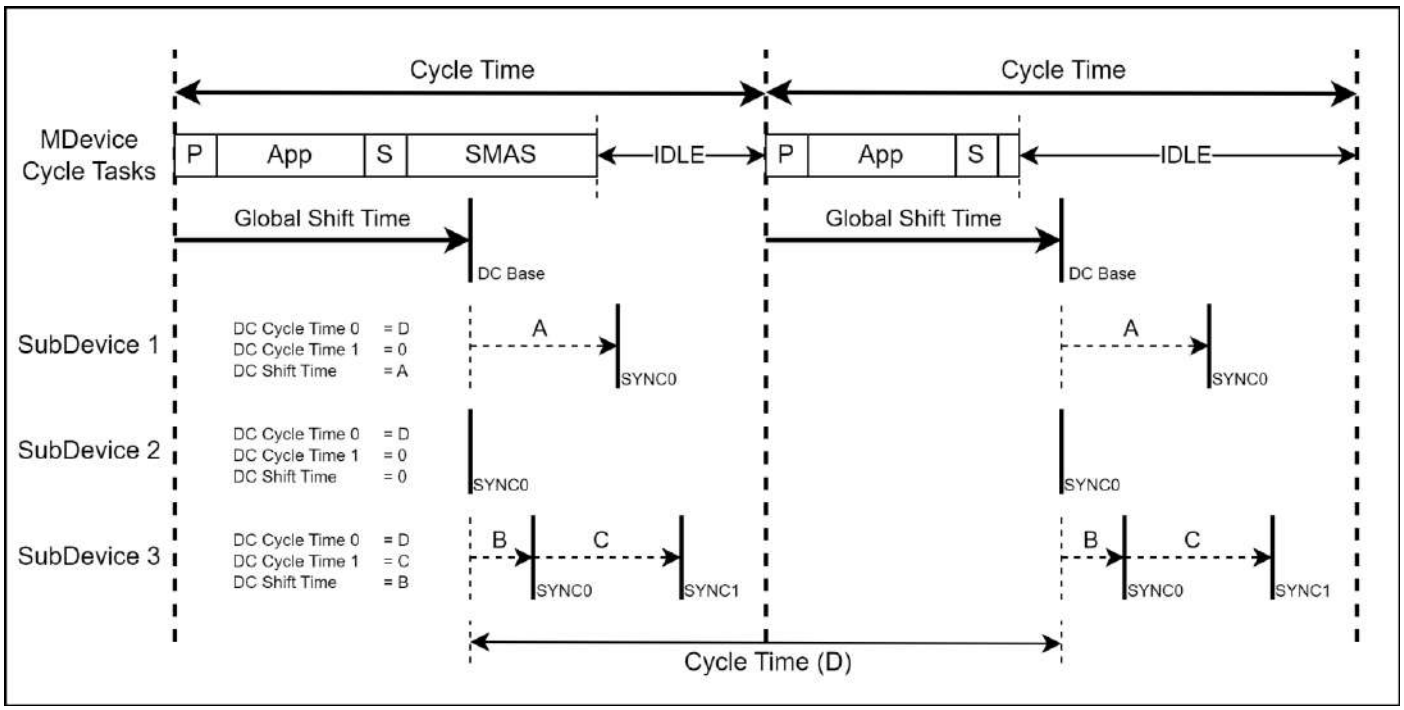
(*1) Please refer to EtherCAT standard document ETG1000.4



Synchronicity and Simultaneousness: Scope view of two distributed devices with 300 nodes and 120 m of cable between them. (Source of information: http://www.ethercat.org/)

Configure DC parameters of the EtherCAT SubDevice. This function has three DC parameters to configure:

- ***DC Cycle Time 0*** is used to set the cycle time for the SYNC0 signal, typically aligned with the EtherCAT communication cycle time.
- ***DC Cycle Time 1*** is used to set the cycle time for the SYNC1 signal, which refers to the delay defined after the SYNC0 pulse. This parameter is optional.
- ***DC Shift Time*** is used to set the offset of the SYNC0 signal relative to the DC Base.



Example 1: Enable DC synchronization

Implementing position control on a CiA 402 EtherCAT SubDevice using the EthercatDevice_Generic class. The CiA 402 control mode is set to cyclic synchronous position mode, and DC synchronization is enabled for precise timing.

The default PDO mapping is as follows:

- Output PDO (RxPDO)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Controlword		Target Position			

- Input PDO (TxPDO)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Statusword		Position Actual Value			

Here is the example code:

```
#include <Ethercat.h>

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t position = 0;

// Cyclic callback function
void myCyclicCallback() {
  // Check if the drive is in the correct state
  if ((slave.pdoRead8(0) & 0x6F) != 0x27)
    return;

  // Increment and write the new position
  slave.pdoWrite32(2, position += 1000);
}

void setup() {
  master.begin();       // Initialize EtherCAT Master

  slave.attach(0, master);   // Attach the first EtherCAT slave
  slave.setDc(1000000);      // Set Distributed Clock synchronization to 1 ms
  slave.sdoDownload8(0x6060, 0x00, 8); // Set operation mode to CSP (Cyclic
Synchronous Position)
```

```
  master.attachCyclicCallback(myCyclicCallback); // Attach the cyclic callback
  master.start(1000000, ECAT_SYNC); // Start EtherCAT Master with 1 ms cycle time
and synchronization

  // Initialize position and control word
  slave.pdoWrite32(2, position = slave.pdoRead32(2)); // Set initial position
  slave.pdoWrite8(0, 0x80); // Reset fault
  delay(1000);
  slave.pdoWrite8(0, 0x06); // Switch to "Shutdown" state
  delay(1000);
  slave.pdoWrite8(0, 0x07); // Switch to "Switch On" state
  delay(1000);
  slave.pdoWrite8(0, 0x0F); // Switch to "Operation Enable" state
  delay(1000);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

4.5.7 86EVA, an EtherCAT Configuration Tool

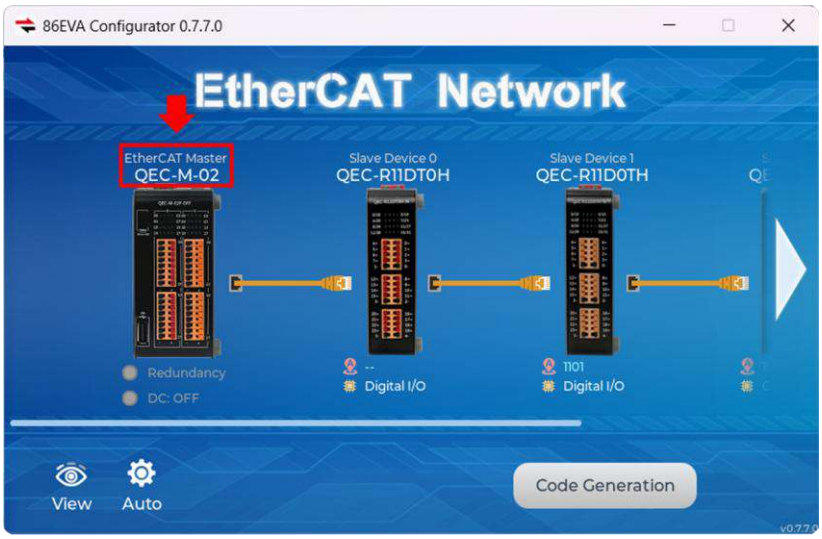
86EVA is a graphical EtherCAT configurator based on the EtherCAT Library in the 86Duino IDE and is one of the development kits for 86Duino. The user can use it to configure the EtherCAT network quickly and start programming.

The following information about 86EVA will focus on QEC EtherCAT MDevice and SubDevices, with features including:

1. Automatically generated Arduino language (via EtherCAT-Based Virtual Arduino)
2. Automatically scan for network devices.
3. EtherCAT MDevice Settings:
    - Set MDevice Object Name
    - Set Cycle Time
    - Set Redundancy Options
    - Optional ENI file
4. EtherCAT SubDevice Settings:
    - Set SubDevice Object Name
    - Set SubDevice Alias Address
    - SubDevice I/O Mapping can be set
    - Display secondary device information
    - View internal information

The 86Duino IDE development environment is specifically designed for the QEC MDevices, which includes QEC-M-01, QEC-M-02, QEC-M-043T, QEC-M-070T, QEC-PPC-M-090T, QEC-M-090T, QEC-PPC-M-104T, QEC-PPC-M-150T, and QEC-M-150T. After connecting and completing the scanning process in 86EVA, users can view the information of the EtherCAT MDevice with the product image.

- QEC-M-02:

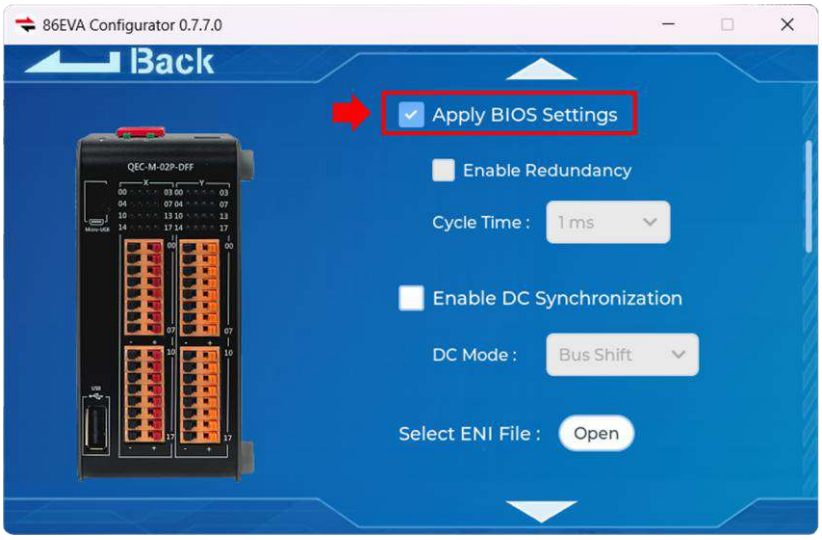


Press twice on the image of the QEC-M-02 to see the parameter settings.

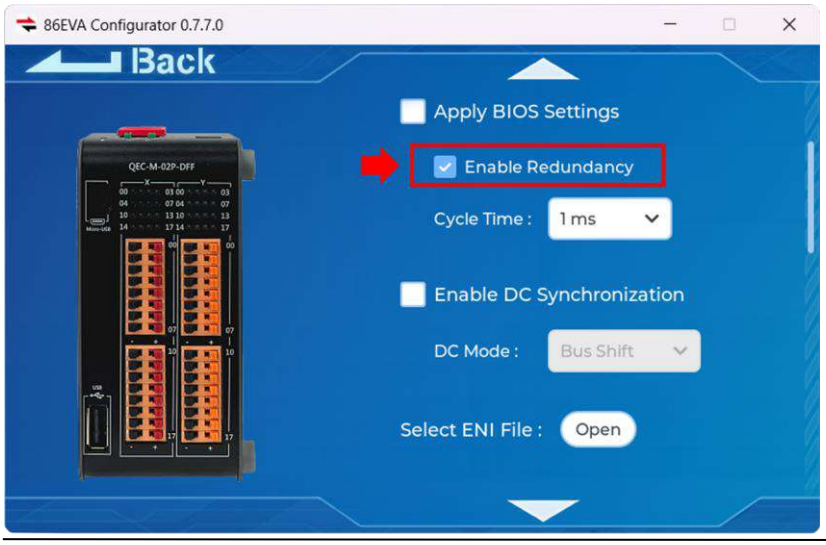
You can set the EtherCAT MDevice Object Name, Apply BIOS Settings, Enable the EtherCAT Cable Redundancy option, set EtherCAT Cycle Time and DC synchronization function, and select the ENI file.

In **“Apply BIOS Settings”**, you can follow Chapter 4.6.3 EtherCAT Page to configure the EtherCAT settings by default.

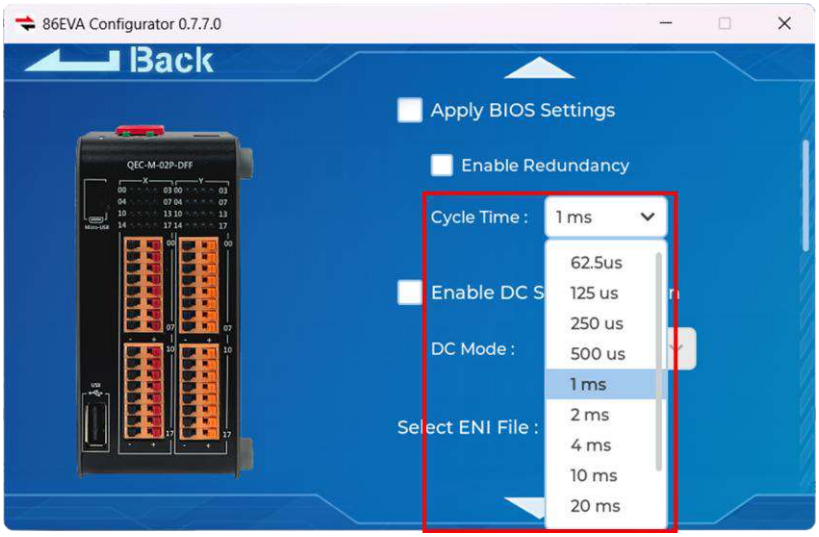
If users select **“Apply BIOS Settings”**, then the select box **“Enable Redundancy”** and the dropdown menu **“Cycle Time”** will not be available.



While not using the BIOS settings, users can select to enable the redundancy function.

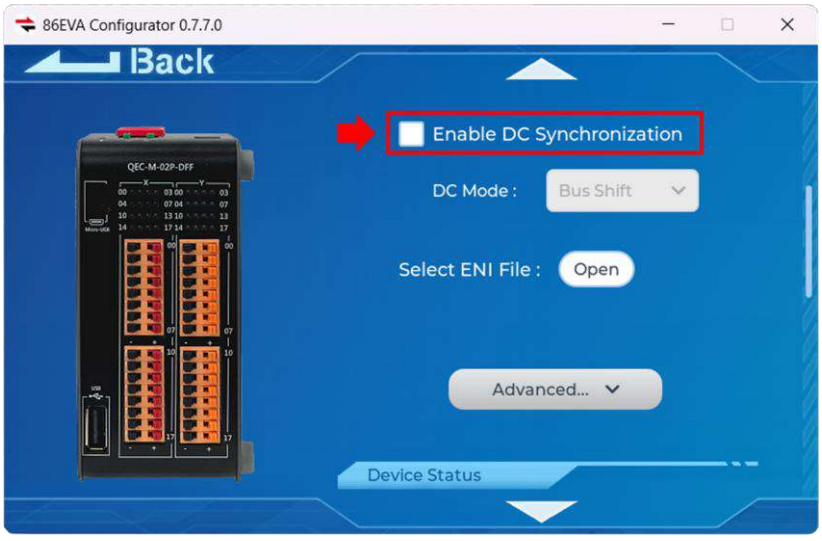


While not using the BIOS settings, users can select EtherCAT's cycle time from the dropdown menu **"Cycle time"**, with options from 62.5 µs to 20 ms.

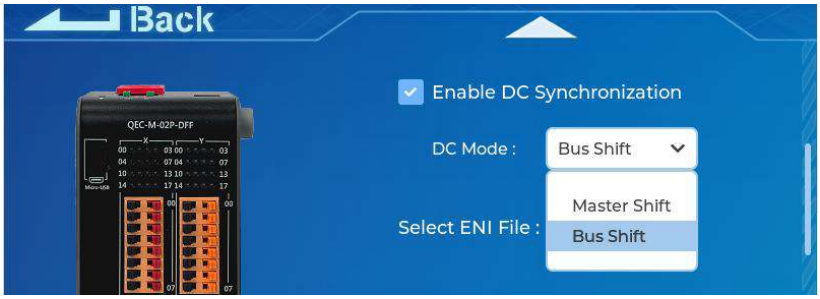


*Note that the cycle time needs to follow the SubDevice's response time.

In **"Enable DC Synchronization,"** you can enable the DC function for all EtherCAT SubDevices on the current EtherCAT network that supports it and can be certificated by 86EVA.

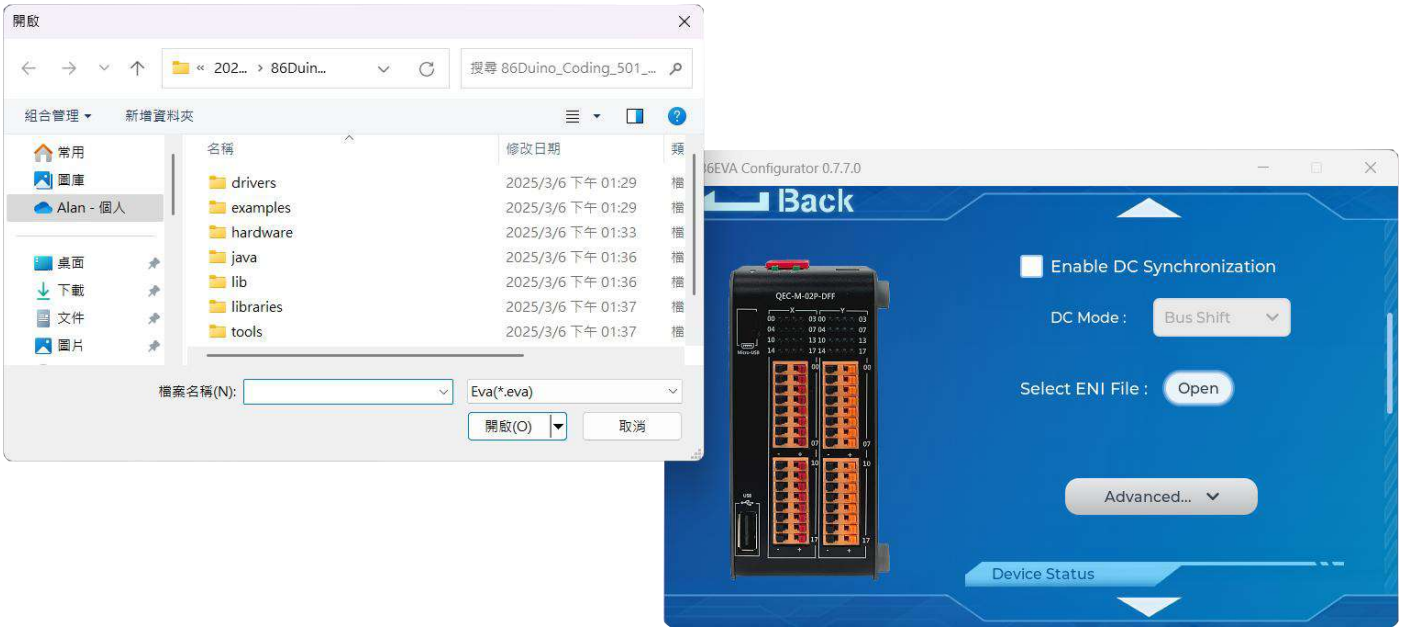


Users can also select DC mode by the dropdown menu **"DC Mode"**, with options "Bus Shift" and "Master Shift".



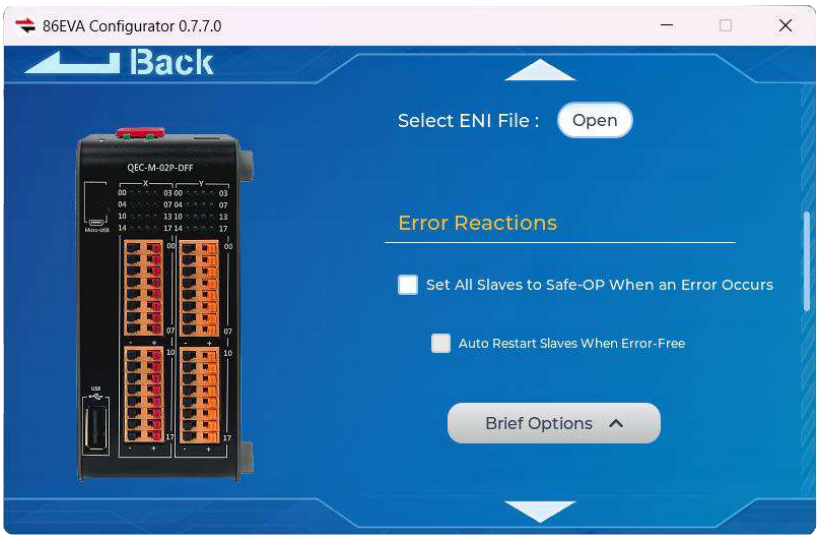
In **“Select ENI File”**, users can upload the ENI file. For more information about ENI file, you can refer to [Beckhoff Information System - English](#).

After you click the **“Open”** button, 86EVA will show file upload window to select the ENI file you need.

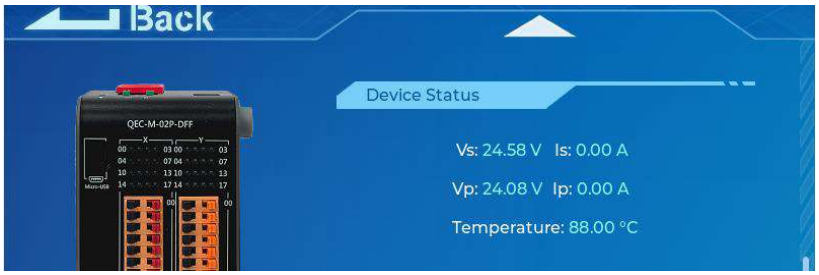


And it’ll show the file name under the **“Select ENI file”** label after you upload the ENI file.

In **“Advanced”**, you can set **“Error Reactions”** for EtherCAT Network, including “Set All Slaves to Safe-OP When an Error Occurs” and “Auto Restart When Error-Free”.

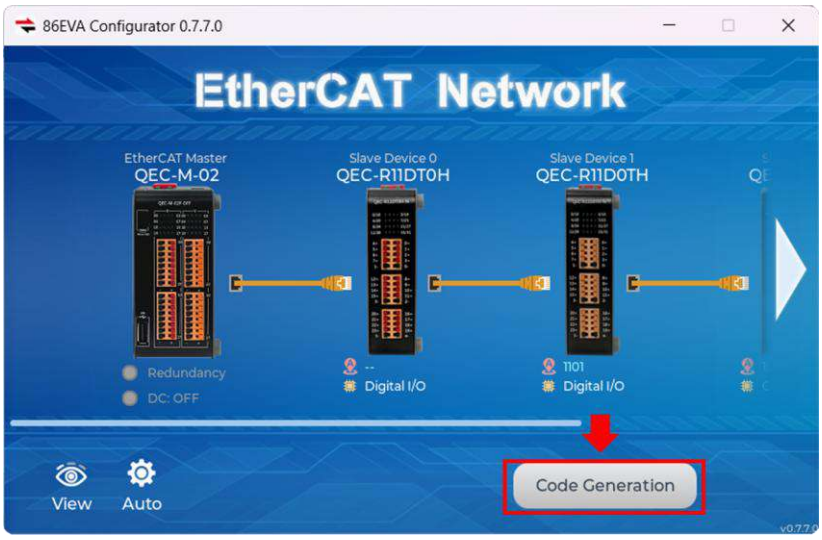


You can also see the voltage, current and system temperature in the **“Device Status”** area.



Users can also set the maximum number of the Virtual Arduino functions.

Once you’re finished your EtherCAT configuration, go back to the home screen and press the "**Code Generation**" button in the bottom right corner.



When you're done, double-click the OK button to turn off 86EVA, or it will close in 10 seconds.

The generated code and files are as follows:

a. sketch_jul23d: Main Project (.ino, depending on your project name).
b. GPT.h: Parameters to provide to ChatGPT referred.
c. myeva.cpp: C++ program code of 86EVA.
d. myeva.h: Header file of 86EVA.

Once the code is completed, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload. The program will run when the upload is complete.

For more detailed information, please refer to 86EVA, EtherCAT-Based Virtual Arduino.

4.5.7.1 Troubleshooting : Cannot Successfully Upload the code

When you are unable to successfully upload code, please open 86EVA to check if your QEC EtherCAT MDevice's environment is abnormal. As shown in the figure below, please try updating your QEC EtherCAT MDevice's environment, which will include the following three items: Bootloader, EtherCAT firmware, and EtherCAT tool.

Now, we will further explain how to proceed with the update:

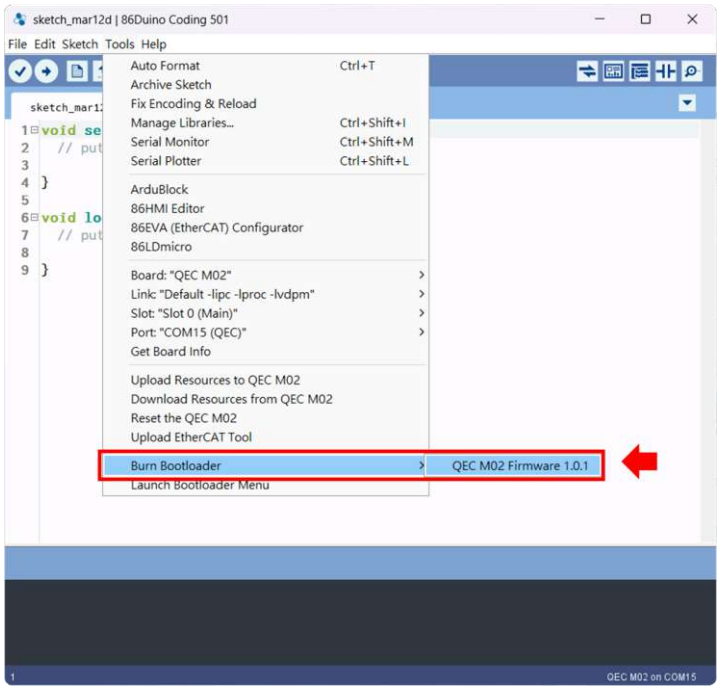
Step 1: Setting up QEC-M

1. Download and install 86Duino IDE 500 (or a newer version): You can download it from Software.
2. Connect the QEC-M: Use a USB cable to connect the QEC-M to your computer.
3. Open 86Duino IDE: After the installation is complete, open the 86Duino IDE software.
4. Select Board: From the IDE menu, choose "Tools" > "Board" > "QEC-M-02" (or the specific model of QEC-M you are using).
5. Select Port: From the IDE menu, choose "Tools" > "Port" and select the USB port to which the QEC-M is connected.

Step 2: Click “Burn Bootloader” button

After connecting to your QEC-M product, go to “Tools”> “Burn Bootloader”. The currently selected QEC-M name will appear. Clicking on it will start the update process, which will take approximately 5-20 minutes.

- QEC-M-02:

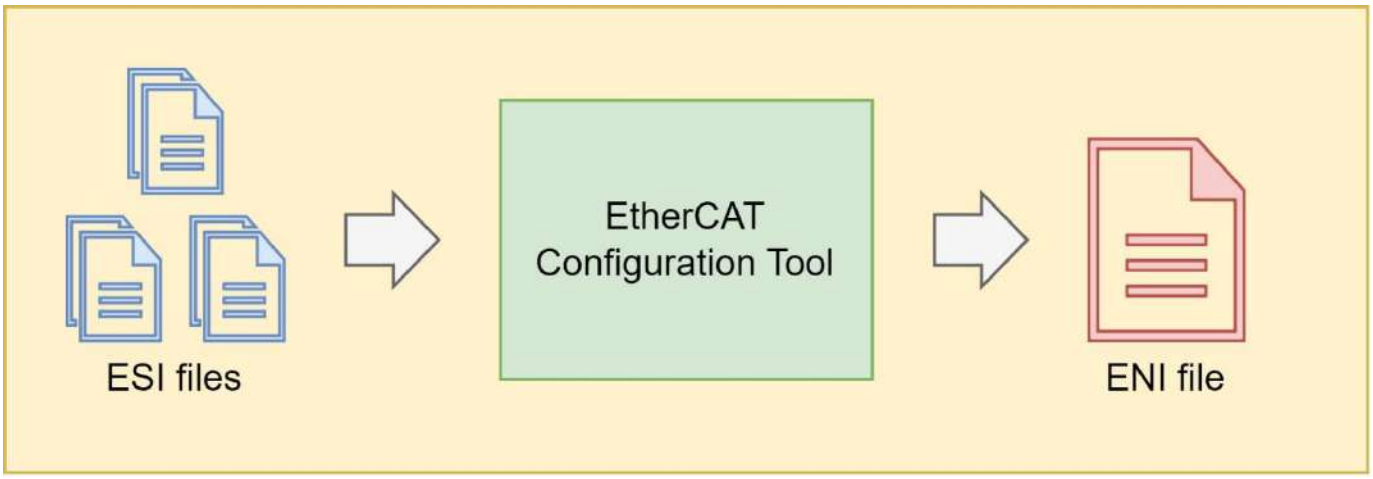


Step 3: Complete the Update

After completing the above steps, your QEC-M has been successfully updated to the latest version of the development environment.

4.5.8 Import ENI to QEC-M-02

The EtherCAT Network Information (ENI) file contains the essential settings needed to configure an EtherCAT network. This XML-based file includes general information about the MDevice and the configurations of every SubDevice connected to it. Using the EtherCAT Configuration Tool, you can read ESI files or perform an online scan of the network to detect all connected SubDevices. You can then configure relevant EtherCAT settings, such as PDO mapping and enabling DC, and export the ENI file.



The EtherCAT Technology Group requires that the EtherCAT MDevice Software support at least one of the following methods in the Network Configuration section: Online Scanning or Reading ENI.
The EtherCAT Library in the 86Duino IDE 500+ of QEC EtherCAT supports both methods. However, when reading ENI, the library currently extracts only partial information from the ENI file for network configuration. For more details, please refer to A.1 About ENI Configuration in 86Duino IDE.

Method 1: Using Code

After setting up your 86Duino IDE environment, please put the ENI file on a USB disk and insert it into your QEC EtherCAT MDevice.

*Note: the USB disk readings via QEC MDevice will be under the P:\\ path.)

Step 1: Write the Import Code:

- In your project code, use the `EthercatMaster::begin()` function to import the ENI file.
- Example code:

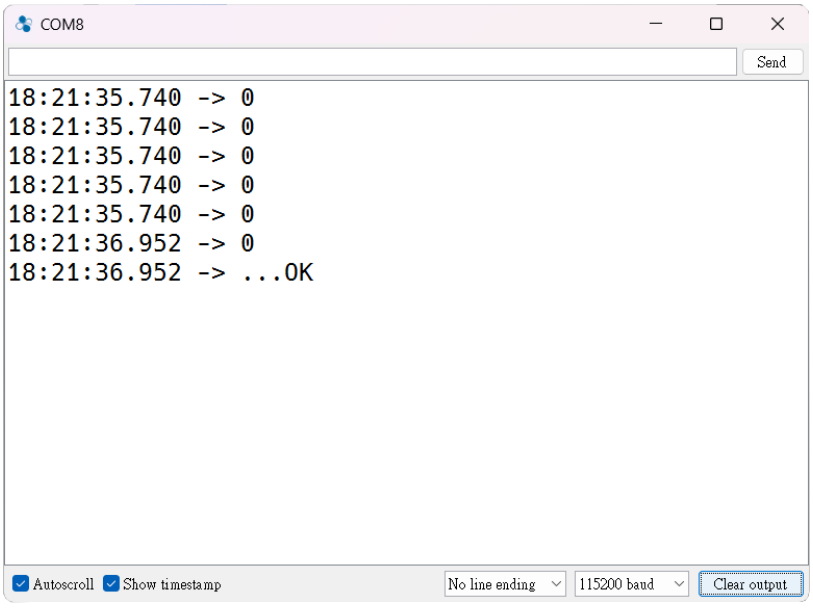
```
#include "Ethercat.h"

#define Device 4

EthercatMaster master;
EthercatDevice_Generic slave[Device];

void setup() {
    Serial.begin(115200);
    Serial.println(master.begin(ECAT_ETH_0, "P:\\test.xml"));
    // Note: the USB disk readings via QEC Master will be under the P:\\ path.
    for (int i = 0; i < Device; i++) {
        Serial.println(slave[i].attach(i, master));
    }
    Serial.println(master.start());
    Serial.println("...OK");
}

void loop() {
    // put your main code here, to run repeatedly:
    // ...
}
```

Step 2: Upload the Code:

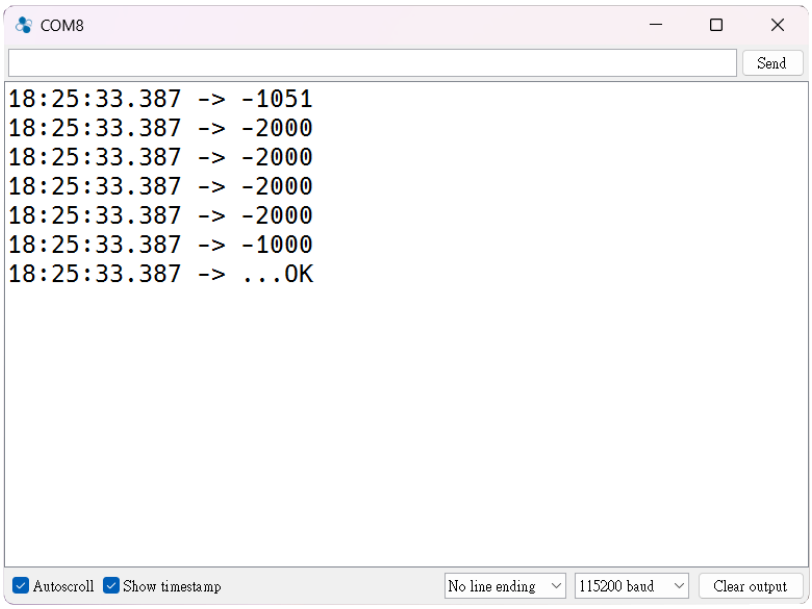
- Upload the code to your QEC EtherCAT MDevice.

Step 3: Verify the ENI File Import:

- Verify that the ENI file is correctly imported and the network is operational via Serial Monitor.
- If the ENI file is imported successfully, you will see the following return value in Serial Monitor:



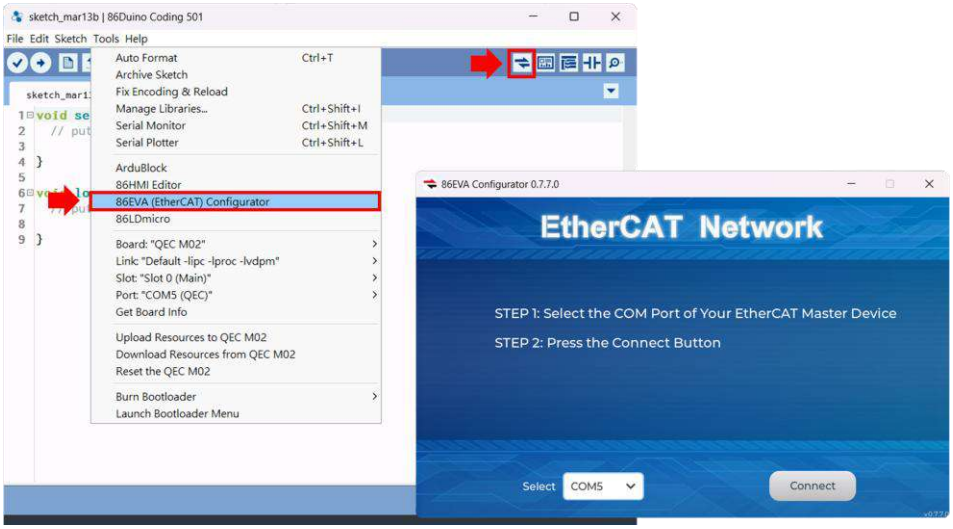
- If there is an error, such as -1051 (which means `ECAT_ERR_MASTER_ENI_MISMATCH`), refer to the EtherCAT API user manual for other error codes and their meanings.



Method 2: Using 86EVA

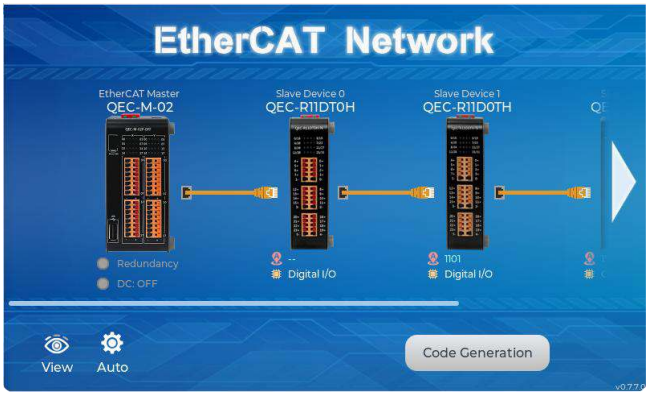
Step 1: Turn on 86EVA and scan:

- The 86EVA tool can be opened via the following buttons.



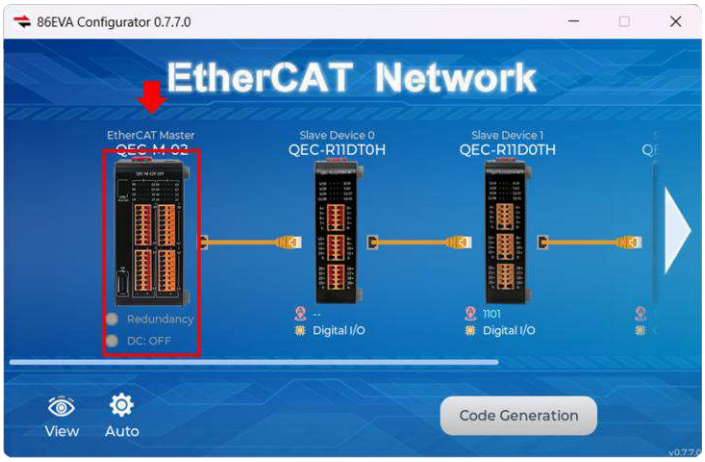
- Once you have confirmed that the correct COM port has been selected of QEC-M-02, press the Connect button to start scanning the EtherCAT network.

- The connected devices will be displayed after the EtherCAT network has been scanned.



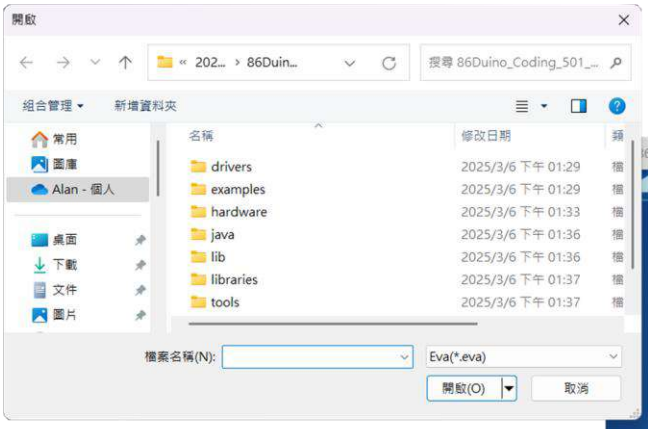
Step 2: Import ENI file:

- Press twice on the QEC-M-02 to enter the corresponding parameter setting screen.

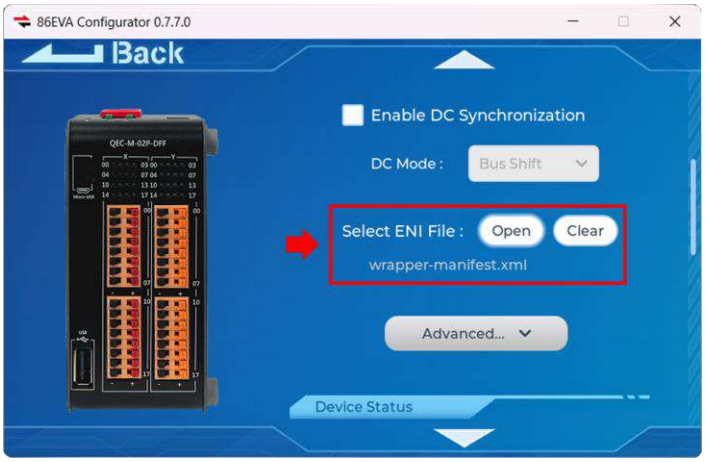


- You can see the parameter settings for the QEC-M-02.

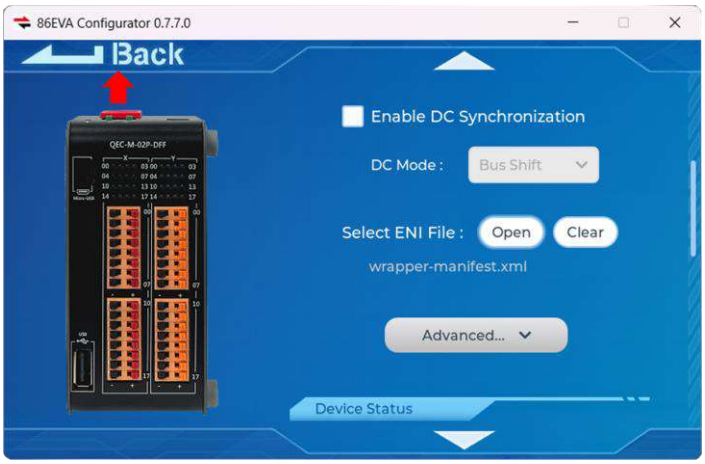
- Click on the “Open” button next to the “Select ENI file” in the General area.



- Browse where you saved the ENI file in Chapter 2, and open it.
- After you import the ENI file into the 86EVA, you can see the ENI file name.

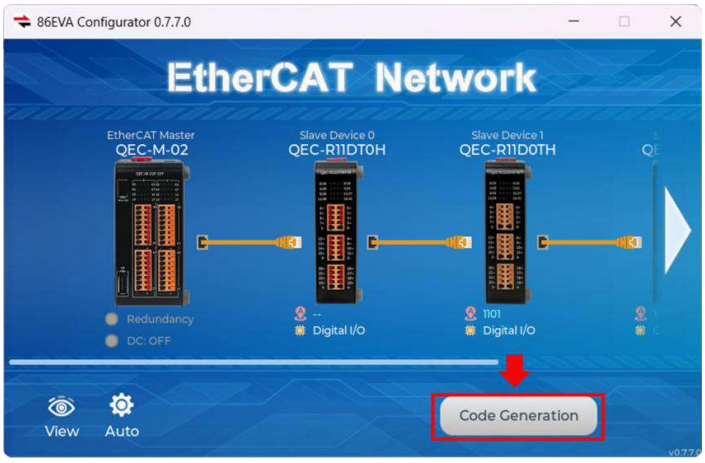


- Please click “Back” button in the upper left corner to return.



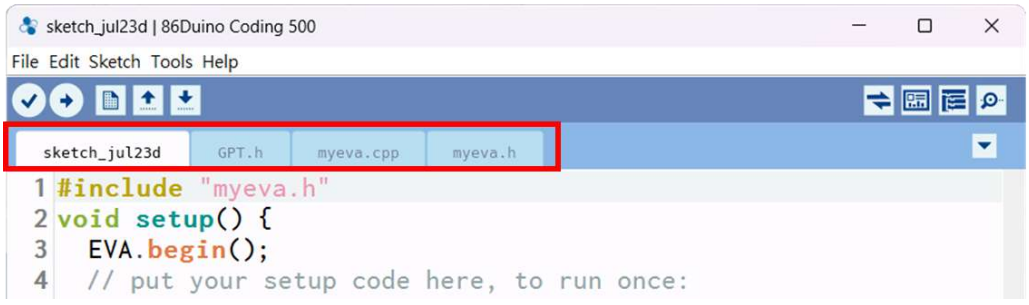
Step 3: Generate the code:

- Go back to the home screen and press the "Code Generation" button in the bottom right corner.



- When you're done, double-click the OK button to turn off 86EVA, or it will close in 10 seconds.

- The generated code and files are as follows:



a. sketch_jul23d: Main Project (.ino, depending on your project name).
b. GPT.h: Parameters to provide to ChatGPT referred.
c. myeva.cpp: C++ program code of 86EVA.
d. myeva.h: Header file of 86EVA.

Step 4: Write the code:

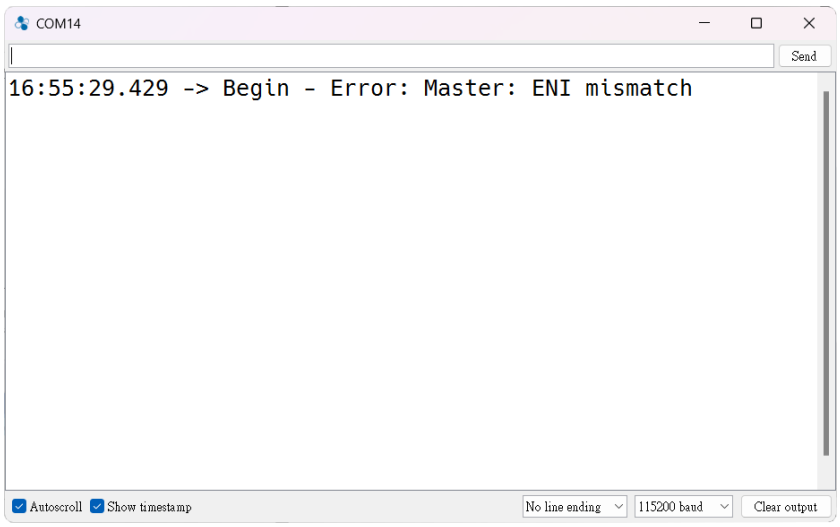
- Please insert `Serial.begin(115200);` before `EVA.begin();` in `void setup() {}`, so the EVA program return value can display in Serial Monitor (in baud rate 115200).

```
#include "myeva.h"

void setup() {
  Serial.begin(115200);
    EVA.begin();
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

- Once the code is completed, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload. The program will run when the upload is complete.

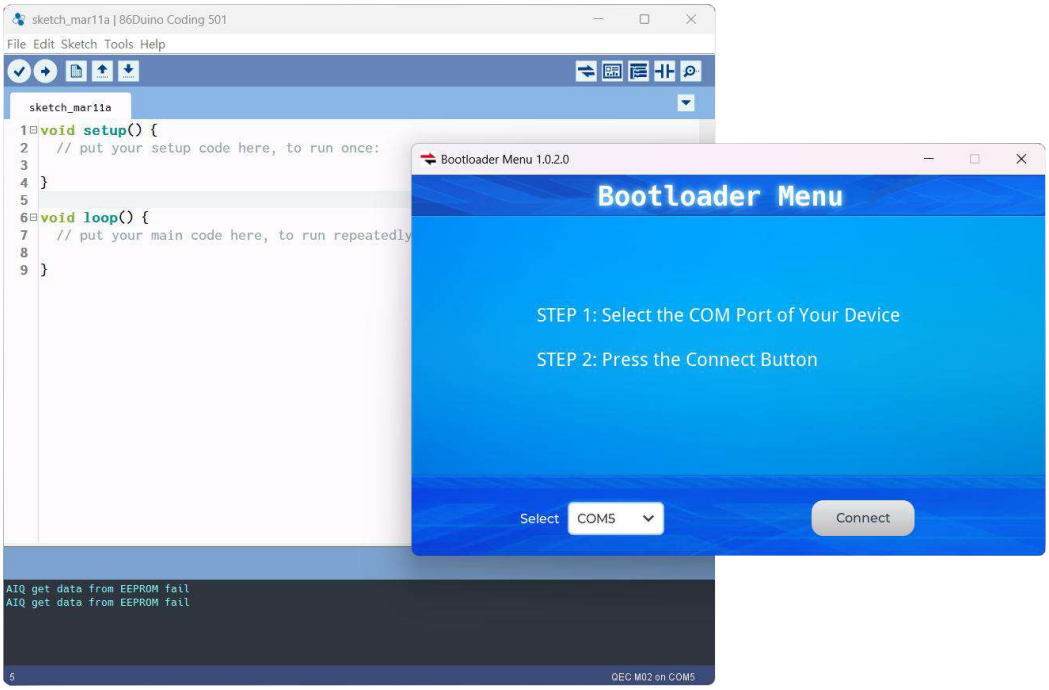
Step 5: Verify the ENI File Import:

- Verify that the ENI file is correctly imported and the network is operational via Serial Monitor.
- If the ENI file is imported successfully, the Serial Monitor will return nothing; if there is an error, it will return the error directly to the Serial Monitor.



4.6 Bootloader Menu Usage

This section introduces the Bootloader Menu, which provides a user-friendly interface for configuring and managing your QEC MDevice.

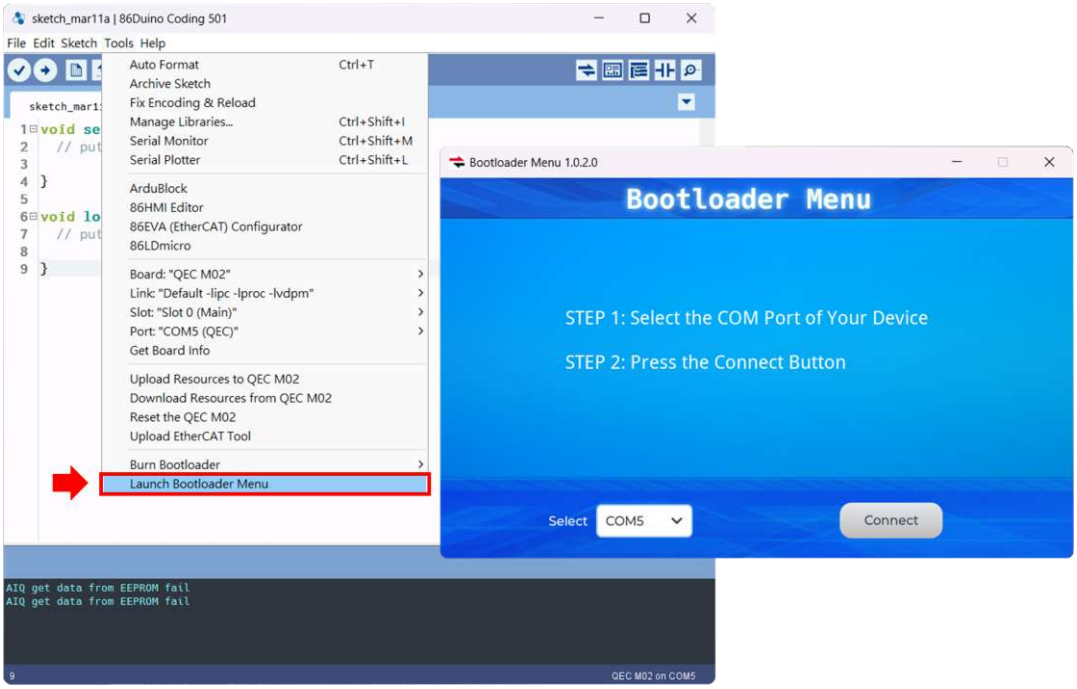


The Bootloader is the first program executed when the 86Duino powers on. It is critical in bridging the 86Duino hardware and the development environment. The Bootloader communicates with the 86Duino Coding IDE via the USB Device/Programming Port. It allows users to upload their sketch programs seamlessly from the development environment to the 86Duino hardware.
The Bootloader accepts user-uploaded sketch programs and writes them to the 86Duino device's onboard memory. After successfully uploading a sketch, the Bootloader executes the program automatically.

This section is a detailed guide to navigating and using the bootloader menu's features effectively.

4.6.1 Turn on Bootloader Menu

The Bootloader Menu can be opened via the following buttons.



Please select the correct COM port and then click the “Connect” button.

Once you have confirmed that the correct COM port has been selected of QEC-M-02, press the Connect button to start the bootloader menu.

After the connection, you can see the bootloader menu like below the picture.

The Top Menu in the Bootloader Menu provides easy access to key settings and features.

It has the following tabs:

1. General:
    - View system info (e.g., model, BIOS version, firmware version, and RTC).
    - Access basic settings like enabling the BIOS menu or allowing IDE connections.
2. EtherCAT:
    - Configure the BIOS default EtherCAT cycle time and enable/disable redundancy.
3. Security:
    - Set or update the bootloader password to secure access.
4. Exit:
    - Save or discard changes and reboot the device.

4.6.2 General Page

The General Page displays essential system information and provides basic configuration options.

System Info:

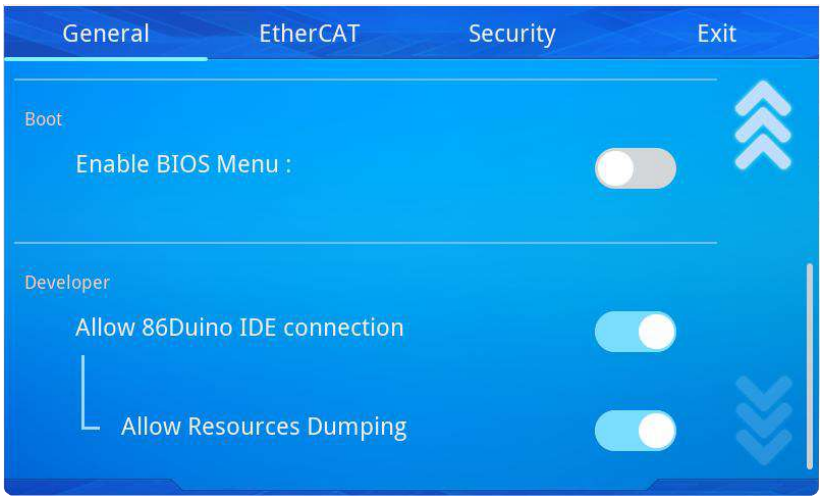
- Machine Model: Displays the current hardware model (e.g., QEC-M-02).
- BIOS Version: Displays the installed BIOS version.
- Firmware Version: Shows the current firmware version installed.
- Date/Time (RTC): Displays the system's real-time clock. Users can adjust this setting.

Additional Information:

To adjust the Date/Time (RTC):

1. Click the gear icon to open the "Set Date/Time" window.
2. Use the displayed options to modify the date and time settings.
3. Click OK to save the changes, or cancel to discard them.

Boot and Developer Options:



Boot:

- Enable BIOS Menu: Enable or disable BIOS Menu.

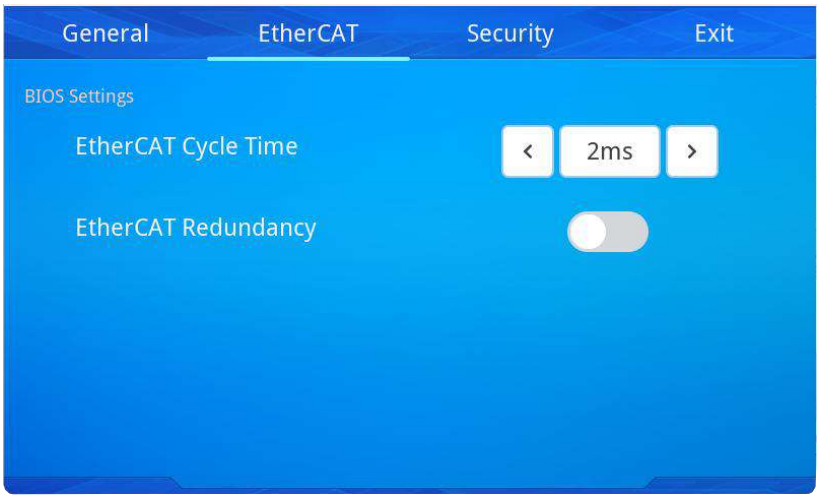
Developer:

- Allow 86Duino IDE connection: for development purposes.
- Allow Resources Dumping: Enable or disable Resources Dumping for debugging.

4.6.3 EtherCAT Page

The EtherCAT Page allows users to configure EtherCAT settings, including Cycle Time and Redundancy, which are part of the BIOS defaults.

BIOS Settings:



- EtherCAT Cycle Time: Configures the default EtherCAT communication cycle time. Options available: 62.5µs, 125µs, 250µs, 500µs, 1ms, and 2ms.
- EtherCAT Redundancy: Enables or disables the default EtherCAT redundancy configuration.

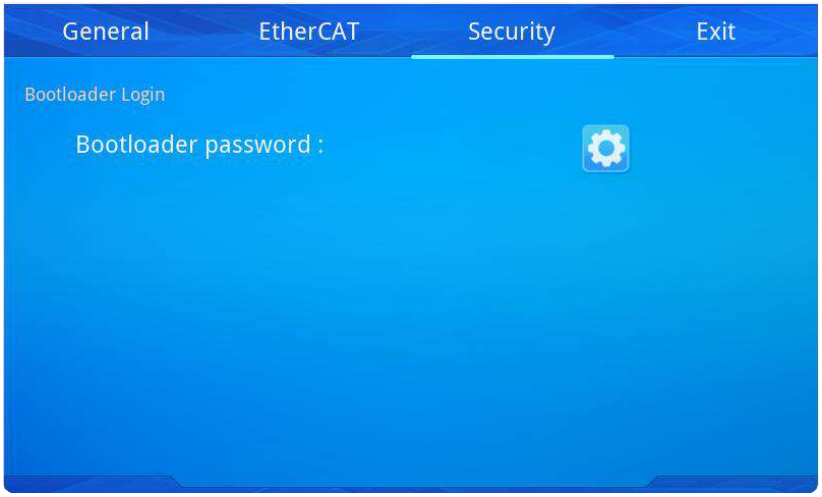
Additional Information:
These BIOS settings directly influence the EtherCAT configuration inside the 86EVA Configurator, as shown in the QEC MDevice page.

Enabling redundancy in the EtherCAT Page will automatically enable the Redundancy checkbox in the 86EVA Configurator, and setting the Cycle Time in the BIOS will apply it as the default value in the configurator.

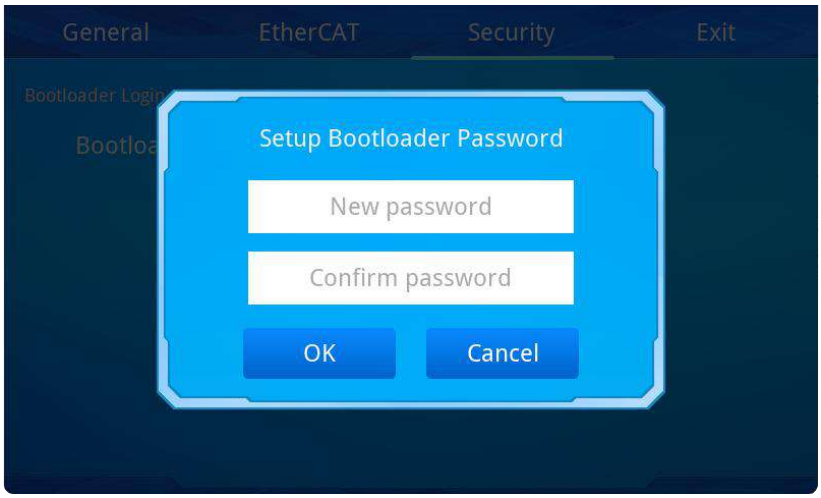
4.6.4 Security Page

The Security Page allows users to set and manage a bootloader password to protect the program and configuration. This feature ensures only authorized users can access the Bootloader Menu.

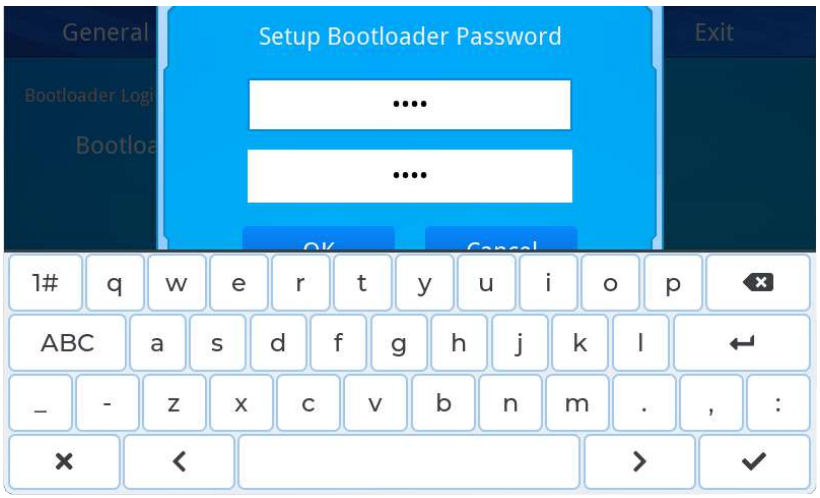
Bootloader Login:



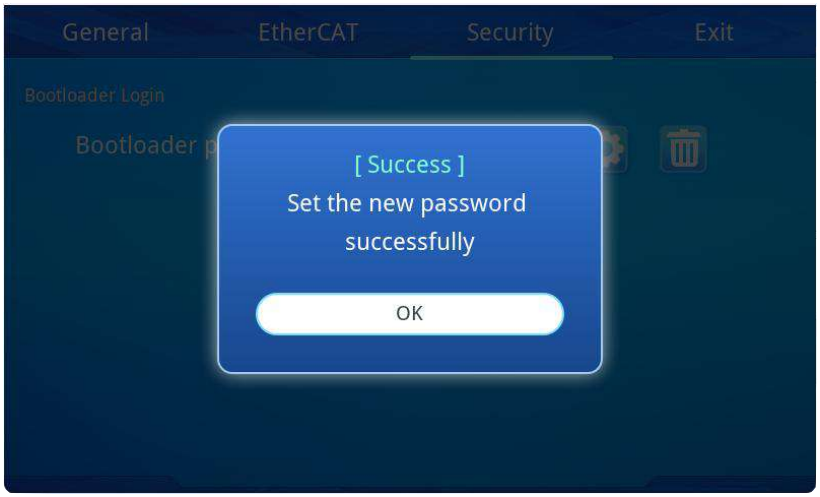
- Bootloader password: Users can click gear icon to open the “Setup Bootloader Password” window, as shown below.



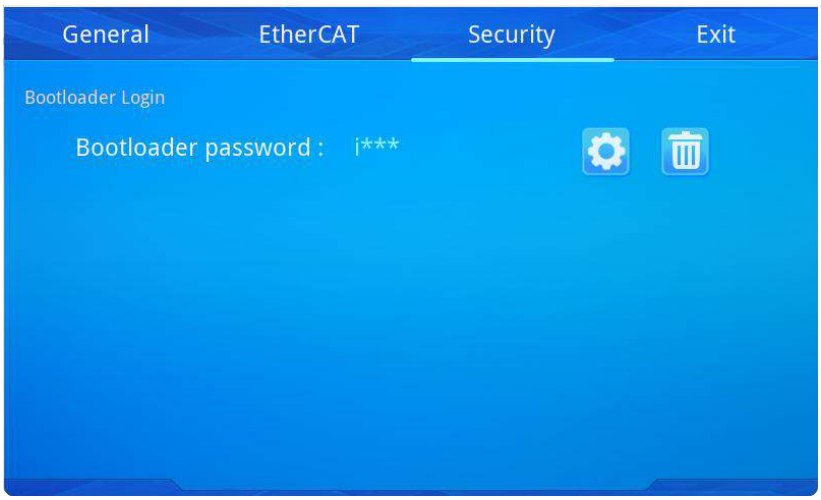
After clicking the text input area, the keyboard will appear on the windows.



When users finish setting the bootloader password, click “OK”. A [Success] window will appear to let the users know that the new password has been set successfully.

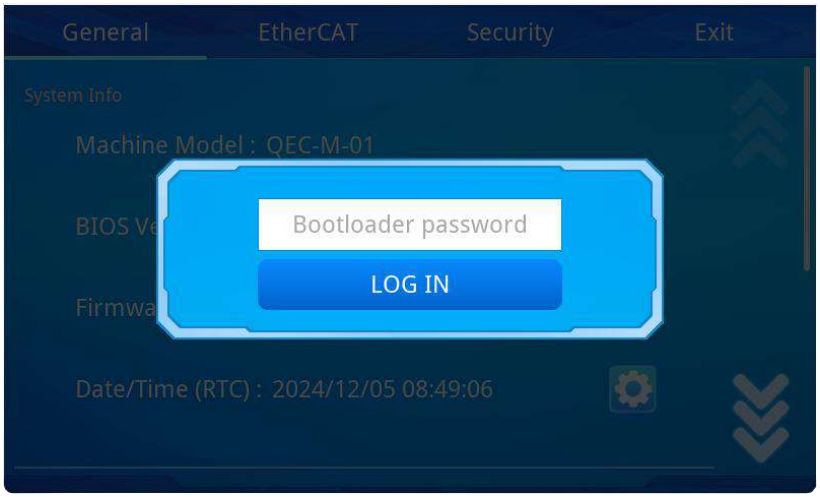


Then, the Security page will look like this.



Users can click the gear icon to change the password and click the garbage can icon to delete the password.

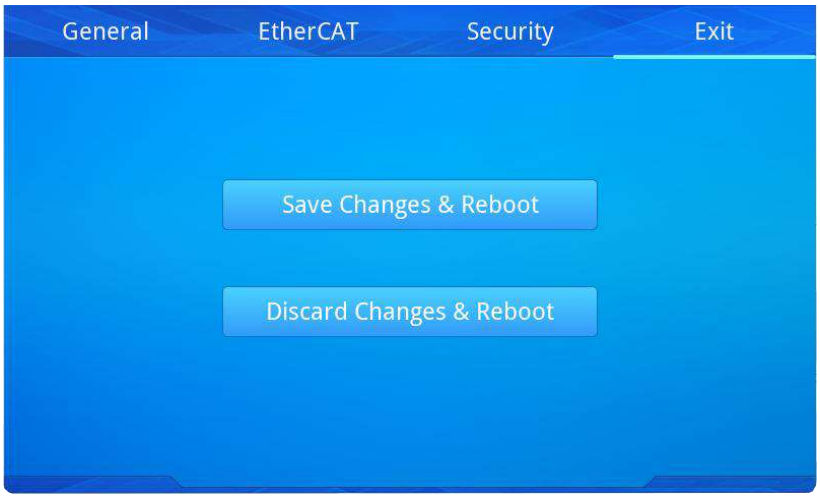
Once users save the bootloader menu configuration, the password will be saved, and the bootloader menu will show when users open it next time.



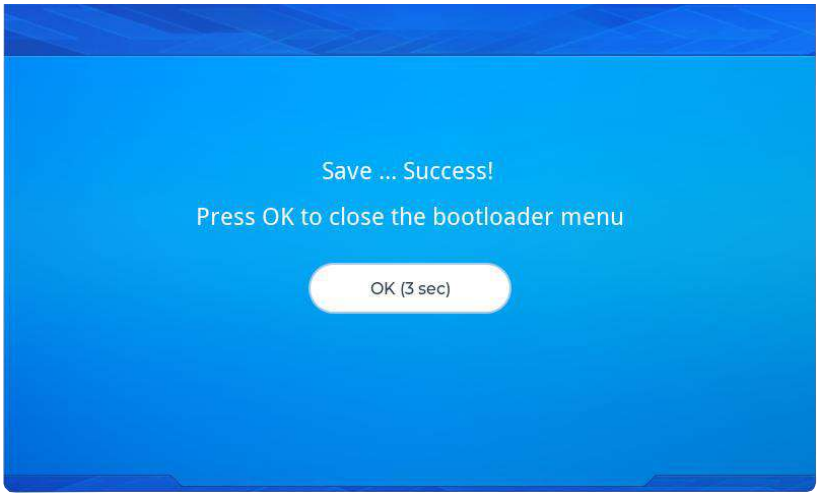
4.6.5 Exit Page

The Exit Page allows users to save or discard changes made in the Bootloader Menu before rebooting the QEC-M-02. This ensures all settings are properly applied or reverted based on user preference.

Exit Page:



- **Save Changes & Reboot:** If users select this option, all changes made in the Bootloader Menu will be saved. The Bootloader Menu will display a confirmation message: "Save... Success" to let users know that the configuration has been successfully saved.

  Users can click OK to close the Bootloader Menu immediately, or wait for the automatic close in 5 seconds.

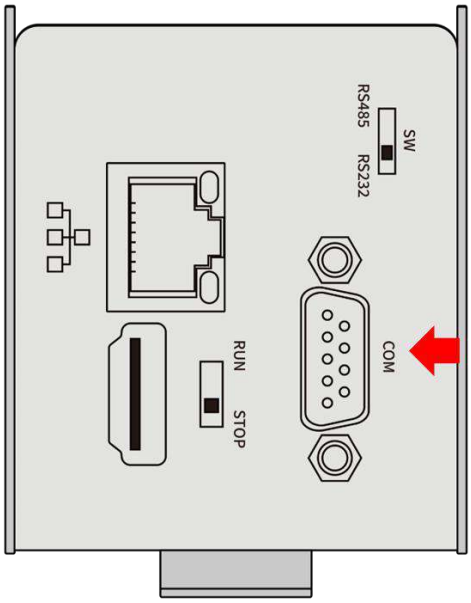
- **Discard Changes & Reboot:** Selecting this option will discard any unsaved changes and reboot the QEC-M-02 with the previously saved configuration.
  The Bootloader Menu will close directly without saving.

4.7 Serial Port Communication

This section provides an overview of the serial communication capabilities of your QEC-M-02 device.

The QEC-M-02 is equipped with a standard DB9 connector, which supports both RS-232 and RS-485 serial communication. Users can switch between RS-232 and RS-485 using a DIP switch located on the device.

DB9 Serial Port Interface:



* Note: RS232 and RS485 cannot be used simultaneously.

RS-232 is a standard for serial communication transmission of data, originally introduced in 1960. For more detailed information about RS232, please refer to https://en.wikipedia.org/wiki/RS-232.

RS485 is a versatile serial communication standard, suitable for long-distance, noise-resistant data transmission in industrial environments. For more detailed information about RS485, please refer to http://en.wikipedia.org/wiki/RS-485.

In the 86Duino environment, serial communication can be implemented using the Serial Library for basic data transmission or the Modbus Library for protocol-based communication.

4.7.1 Serial Communication

To use the serial communication, refer to the Serial Library documentation.
Users can use `SerialCOM` to configure the serial port on the QEC-M-02 device.

Below is a simple example:

```
void setup() {
  SerialCOM.begin(115200);
  Serial.begin(115200);
  Serial.println("Start");
}

void loop() {
  SerialCOM.write("hello Serial");
  delay(10);

  for (int i = 0; i < 11; i++) {
    while (SerialCOM.available() < 1);
    Serial.print(SerialCOM.read());
    Serial.print(",");
  }
  Serial.println();
}
```

Key Functions:

- `SerialCOM.begin(baud_rate)` : Initializes communication with the specified baud rate.
- `SerialCOM.read()` : Reads incoming data from the serial port.
- `SerialCOM.write(data)` : Sends data through the serial port.

For more information, refer to the 86Duino Serial Library Reference.

4.7.2 Modbus RTU Communication

For more advanced communication protocols, the RS485 interface can be used with the Modbus Library.

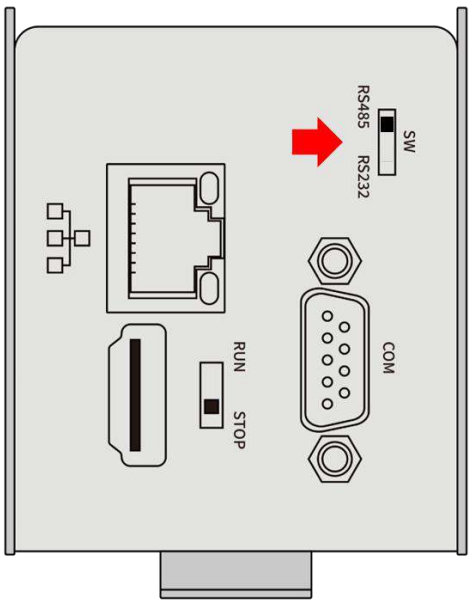
86Diuno IDE supports the Modbus communication protocol, an industrial communication standard published in 1979 for communication between automated electronic devices such as Programmable logic controllers (PLCs).
Modbus is a master/slave based protocol where a master node communicates with multiple slave nodes on the network. Each node has a unique address. When the master node sends a packet to the specified address, only the slave node at the corresponding address receives and parses the packet and executes and responds to commands based on the packet contents.

86Duino's Modbus library has the following features:

- It supports Modbus RTU, TCP, and ASCII sub-protocols.
- Runs as both Modbus master and Modbus slave nodes.
- Support Modbus gateway function

This section provides the foundation for implementing robust RS485 communication using the QEC-M-02. For additional examples and advanced configurations, consult the linked documentation.

*Note that please switch Serial port to RS485 mode.



For detailed usage, visit the 86Duino Modbus Library Documentation.

Example 1: Send Modbus Master RTU 8 bit data output

- Set the host RS485 baud rate to 115200.
- Configure the Modbus master to use RTU mode and connect it to the host RS485 on QEC MDevice.
- Main Loop Program: The Modbus slave has an ID of 30 and 8 Coils. Write 1 to the odd-numbered Coils and 0 to the even-numbered Coils. After a 1-second interval, write 0 to the odd-numbered Coils and 1 to the even-numbered Coils. Repeat this process in a continuous loop.

Here is the example code:

```
#include "Modbus.h"

ModbusMaster bus;

void setup() {
  Serial.begin(3000000);
  SerialCOM.begin(115200);    // Set the Serial baud rate
  bus.begin(MODBUS_RTU, SerialCOM);    // Initialize Modbus RTU
}

void loop() {
  static bool state = false;
  uint16_t coils = state ? 0xAAAA : 0x5555;    // Odd coils 1, even coils 0 and vice
versa
  bus.writeMultipleCoils(30, 0, 8, &coils);
  state = !state;
  delay(1000);        // Wait for 1 second
}
```

Example 2: Read Modbus Master RTU 8 bit data input

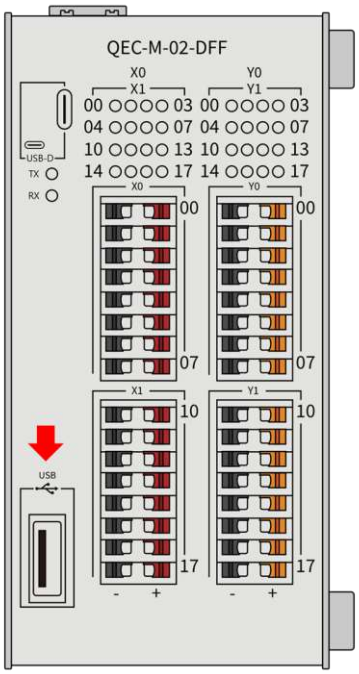
- Set the host RS485 baud rate to 115200.
- Configure the Modbus master to use RTU mode and connect it to the host RS485.
- Main Loop Program: The Modbus slave has an ID of 30. Continuously read the value of the 4th Holding Register and print it using `Serial.print` every 1 second in a loop.

Here is the example code:

```
#include "Modbus.h"

ModbusMaster bus;

void setup() {
  Serial.begin(3000000);
  SerialCOM.begin(115200);    // Set the serial baud rate
  bus.begin(MODBUS_RTU, SerialCOM);    // Initialize Modbus RTU
}

void loop() {
  static uint32_t lastTime = 0;
  uint32_t currentTime = millis();
    // Check if one second has passed
    if (currentTime - lastTime >= 1000) {
      uint16_t data[1];
      uint8_t result = bus.readHoldingRegisters(30, 4, 1, data);
      if (result == 0) {
        Serial.print("Holding Register 4 Value: ");
        Serial.println(data[0]);
      } else {
        // Handle error
      }
      lastTime = currentTime;   // Update the last time a write was attempted
    }
}
```

4.8 USB Device Usage

This section ensures proper usage of the USB device and file storage on the QEC-M-02.
The QEC-M-02 features a Standard USB 2.0 port with hot-plug support, allowing users to connect USB storage devices for file storage, transfer, or accessing configuration data.



Users can utilize the SD Library to read from and write to the USB folder. For example, it is possible to create .txt files using this library. The SD library allows for reading from and writing to SD cards in the MicroSD/SD slot of 86Duino. The library supports FAT16 and FAT32 file systems on standard SD cards and SDHC cards. The file names passed to the SD library functions can include paths separated by forwarding slashes, /, e.g. “directory/filename.txt“. Because the working directory is always the root of the SD card, a name refers to the same file whether or not it includes a leading slash (e.g., “/file.txt” is equivalent to “file.txt“). The library supports opening multiple files.
However, for placing files in a specific folder, the SD Library cannot be used. Instead, users must rely on the standard C language function `fopen()` to specify the file's location manually.

Warnings: Slot Recommendations:
The QEC-M-02 has four storage slots: A, B, C, and P.

- A and B Slots:These are reserved for system files. Users are strongly discouraged from storing files in these slots, as doing so could corrupt the system and require a factory reset or hardware reprogramming of the 32MB flash.
- C Slot: This slot refers to the EMMC and is recommended for storing user files safely.
- P Slot: This slot refers to the USB device and is also suitable for user file storage.

Example 1: Save .txt in USB disk

Below is an example of creating a file using `fopen()` in C language. We save a .txt file on an **external USB disk**. This demonstrates specifying the file location and ensuring safe storage in the appropriate slot.

Here is the example code:

```
char* strs[] = {"Hello, world", "Hello, world2", "Hello, world3"};

void setup() {
   FILE *fp;
   char str[256];

   Serial.begin(115200);

   // Write strings to the test.txt file in the P slot
   fp = fopen("P:\\test.txt", "w");
   for (int i=0; i<sizeof(strs)/sizeof(char*); i++)
      fprintf(fp, "%s\n", strs[i]);
   fclose(fp);

   // Read the strings and print them
   fp = fopen("P:\\test.txt", "r");
   while (fgets(str, sizeof(str), fp)!=NULL ) {
      Serial.print(str);
   }
   fclose(fp);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

*Note: Please use an external USB disk in the QEC-M-02 device.

Additional Example: Save .txt in EMMC storage

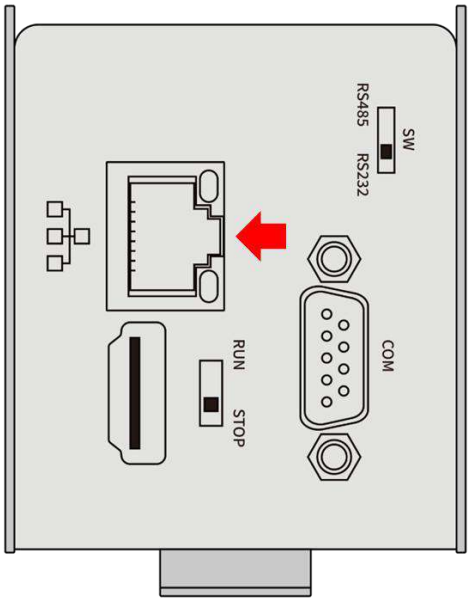
Below is an example of creating a file using `fopen()` in C language. We save a .txt file on an **internal EMMC storage**. This demonstrates specifying the file location and ensuring safe storage in the appropriate slot.

Here is the example code:

```
char* strs[] = {"Hello, world", "Hello, world2", "Hello, world3"};

void setup() {
   FILE *fp;
   char str[256];

   Serial.begin(115200);

   // Write strings to the test.txt file in the P slot
   fp = fopen("C:\\test.txt", "w");
   for (int i=0; i<sizeof(strs)/sizeof(char*); i++)
      fprintf(fp, "%s\n", strs[i]);
   fclose(fp);

   // Read the strings and print them
   fp = fopen("C:\\test.txt", "r");
   while (fgets(str, sizeof(str), fp)!=NULL ) {
      Serial.print(str);
   }
   fclose(fp);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

4.9 Giga LAN Configuration

This section introduces the Giga LAN configuration and control for your QEC MDevice.
The QEC-M-02 features one Giga LAN port dedicated to external Ethernet communication for general network use.



To drive the Giga LAN, you can utilize either the Ethernet Library or the Modbus Library in the 86Duino environment.

Ethernet Library:

- Provides tools for general networking, including:
- Static and dynamic IP configuration.
- TCP/UDP communication.
- Hosting web servers.

For more details, refer to the Ethernet Library Documentation.

Modbus Library:

- Allows communication with Modbus TCP devices using Ethernet.
- Ideal for industrial automation and monitoring applications.

For more information, refer to the Modbus Library Documentation.

4.9.1 Ethernet Communication

This section introduces the Ethernet configuration and control for your QEC MDevice.
The CPU of the QEC-M series contains a built-in 10/100/1000Mbps LAN interface, which can access via the Ethernet library. The Arduino Ethernet Shield isn’t needed. This library can serve as either a server accepting incoming connections or a client making outgoing ones. In 86Duino Coding, the library supports up to four concurrent connections (incoming or outgoing or a combination); and in later versions, up to 128 concurrent connections.

Example 1: DHCP-based IP printer

This sketch uses the DHCP extensions to the Ethernet library to get an IP address via DHCP and print the address obtained.

Here is the example code:

```
#include <Ethernet.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
EthernetClient client;

void setup() {
  Serial.begin(115200);
  while (!Serial);
  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0)
    Serial.println("Failed to configure Ethernet using DHCP");

  // print your local IP address:
  Serial.print("My IP address: ");
  for (byte thisByte = 0; thisByte < 4; thisByte++) {
    // print the value of each byte of the IP address:
    Serial.print(Ethernet.localIP()[thisByte], DEC);
    Serial.print(".");
  }
  Serial.println();
}

void loop() {

}
```

Example 2: Web Server

A simple web server that shows the value of the analog input pins.

Here is the example code:

```
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);
EthernetServer server(80);

void setup() {
  Serial.begin(115200);
  while (!Serial);

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connnection: close");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
```

```
          // add a meta refresh tag, so the browser pulls again every 5 seconds:
          client.println("<meta http-equiv=\"refresh\" content=\"5\">");
          // output the value of each analog input pin
          for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
    Serial.println("client disonnected");
  }
}
```

Example 3: DHCP Chat Server

A simple server that distributes any incoming messages to all connected clients. To use telnet to your device's IP address and type. You can see the client's input in the serial monitor as well. Using an Arduino Wiznet Ethernet shield.

Here is the example code:

```
#include <Ethernet.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };

IPAddress ip(192,168,1, 177);
IPAddress dnsserver(192,168,1, 1);
IPAddress gateway(192,168,1, 1);
IPAddress subnet(255, 255, 0, 0);

// telnet defaults to port 23
EthernetServer server(23);
boolean gotAMessage = false; // whether or not you got a message from the client
yet

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  // this check is only needed on the Leonardo:
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  // start the Ethernet connection:
  Serial.println("Trying to get an IP address using DHCP");
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // initialize the ethernet device not using DHCP:
    Ethernet.begin(mac, ip, dnsserver, gateway, subnet);
  }
  // print your local IP address:
  Serial.print("My IP address: ");
  ip = Ethernet.localIP();
  for (byte thisByte = 0; thisByte < 4; thisByte++) {
```

```
    // print the value of each byte of the IP address:
    Serial.print(ip[thisByte], DEC);
    Serial.print(".");
  }
  Serial.println();
  // start listening for clients
  server.begin();

}

void loop() {
  // wait for a new client:
  EthernetClient client = server.available();

  // when the client sends the first byte, say hello:
  if (client) {
    if (!gotAMessage) {
      Serial.println("We have a new client");
      client.println("Hello, client!");
      gotAMessage = true;
    }

    // read the bytes incoming from the client:
    char thisChar = client.read();
    // echo the bytes back to the client:
    server.write(thisChar);
    // echo the bytes to the server as well:
    Serial.print(thisChar);
  }
}
```

Example 4: MySQL Connector

This example demonstrates connecting to a MySQL server from a QEC MDevice Ethernet port. For more information and documentation, visit the wiki: https://github.com/ChuckBell/MySQL_Connector_Arduino/wiki.

Here is the example code:

```
#include <Ethernet.h>
#include <MySQL_Connection.h>

byte mac_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server_addr(10,0,1,35);  // IP of the MySQL *server* here
char user[] = "root";              // MySQL user login username
char password[] = "secret";        // MySQL user login password

EthernetClient client;
MySQL_Connection conn((Client *)&client);

void setup() {
  Serial.begin(115200);
  while (!Serial); // wait for serial port to connect
  Ethernet.begin(mac_addr);
  Serial.println("Connecting...");
  if (conn.connect(server_addr, 3306, user, password)) {
    delay(1000);
    // You would add your code here to run a query once on startup.
  }
  else
    Serial.println("Connection failed.");
  conn.close();
}

void loop() {

}
```

4.9.2 Modbus TCP Communication

This section introduces the Modbus TCP configuration and control for your QEC MDevice. Please refer to 4.7.2 Modbus RTU Communication about 86Duino IDE Modbus Library.

Example 1: Simple Modbus Master TCP

This example uses the Ethernet and ModbusMaster libraries to establish Modbus TCP communication over Ethernet. The program initializes an Ethernet connection with a static IP address and configures a Modbus TCP master to interact with a slave device with ID 11.

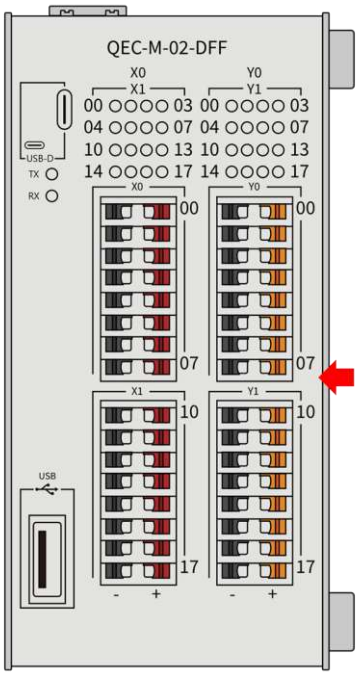
Here is the example code:

```
#include <Ethernet.h>
#include <Modbus.h>

ModbusMaster bus;
ModbusMasterNode node;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 101);
IPAddress serverIp(192, 168, 1, 102);

int led = 0;
uint32_t value = 0;

void setup()
{
  Ethernet.begin(mac, localIp);

  /* Modbus TCP Mode via Ethernet. */
  bus.begin(MODBUS_TCP, serverIp);

  /* Slave node initialize. */
  node.attach(11, bus);
}

void loop()
{
  uint16_t reg[2];
```

```
  /* Write 1 coil to address 0 of the slave with ID 11. */
  node.writeSingleCoil(0, led);

  /* Write 2 word to holding registers address 16 of the slave with ID 11. */
  reg[0] = value & 0xFFFF;
  reg[1] = (value >> 16) & 0xFFFF;
  node.setTransmitBuffer(0, reg[0]);
  node.setTransmitBuffer(1, reg[1]);
  node.writeMultipleRegisters(16, 2);

  /* Read 2 word from holding registers address 16 of the slave with ID 11. */
  node.readHoldingRegisters(16, 2);
  Serial.print("From Node 1 Holding Register: ");
  value = node.getResponseBuffer(0) | (node.getResponseBuffer(1) << 16);
  Serial.println(value);

  /* Read 1 word from input registers address 2 of the slave with ID 11. */
  node.readInputRegisters(2, 1);
  Serial.print("             Input Register: ");
  Serial.print(node.getResponseBuffer(0));
  Serial.println();

  led = !led;
  value++;

  delay(1000);
}
```

4.10 Digital IO Configuration

This section introduces the digital IO pin control for your QEC-M-02 device.



Users can control QEC-M-02's digital input (X) and digital output (Y) using standard Arduino code. The QEC-M-02 is compatible with the 86Duino language, allowing seamless integration with familiar Arduino functions. For more details on supported functions and syntax, please refer to the 86Duino Language Reference: https://www.qec.tw/86duino/86duino-language-reference/.

The QEC-M-02 provides 16 digital input channels and 16 digital output channels, assigned as follows:

- Digital Input Channels → X0 and X1
  - X0: X00 to X07
  - X1: X10 to X17
- Digital Output Channels → Y0 and Y1
  - Y0: Y00 to Y07
  - Y1: Y10 to Y17

For the digital pin configuration and index assignment, please refer to Chapter 2.2.6 Digital I/O Connector.

*Note:

- Ensure proper grounding and correct voltage to avoid damage.
- Use relay modules for high-power devices.

Example 1: Read Digital Input X00

Read the digital input pin X00 on QEC-M-02

Here is the example code:

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  int inputState = digitalRead(X00);
  Serial.print("X00 State: ");
  Serial.println(inputState);
  delay(500);
}
```

Example 2: Write Digital Output Y10

Write the digital output pin Y10 on QEC-M-02

Here is the example code:

```
void setup() {
  // ...
}

void loop() {
  digitalWrite(Y10, HIGH);
  delay(4000);
  digitalWrite(Y10, LOW);
  delay(1000);
}
```

Example 3: Read Digital Input (X00) and Control Digital Output (Y10)

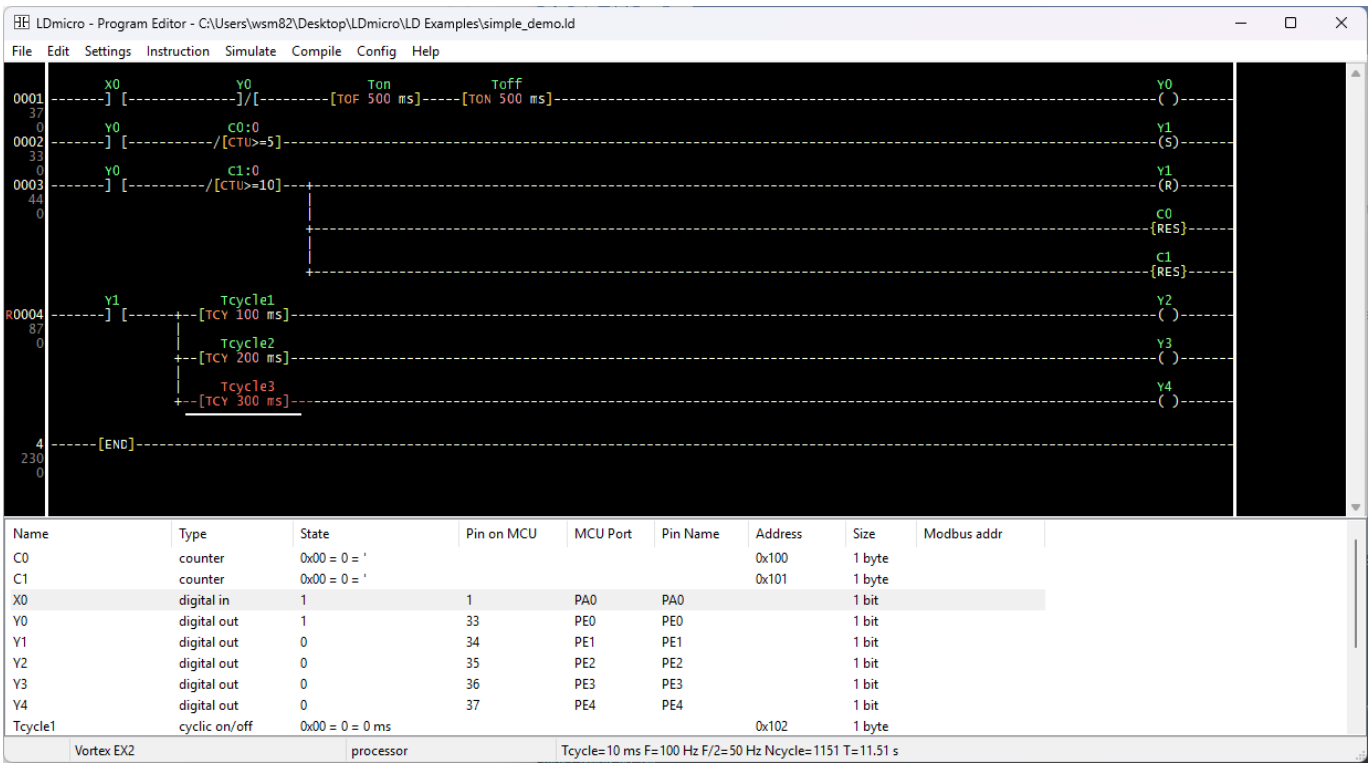
Read digital input pin X00 and control digital output pin Y10 on QEC-M-02

Here is the example code:

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  int inputState = digitalRead(X00);
  Serial.print("X00 State: ");
  Serial.println(inputState);

  if (inputState == HIGH) {
    digitalWrite(Y10, HIGH);
  } else {
    digitalWrite(Y10, LOW);
  }

  delay(500);
}
```

4.10.1 PLC Ladder Diagram Tool: LDmicro

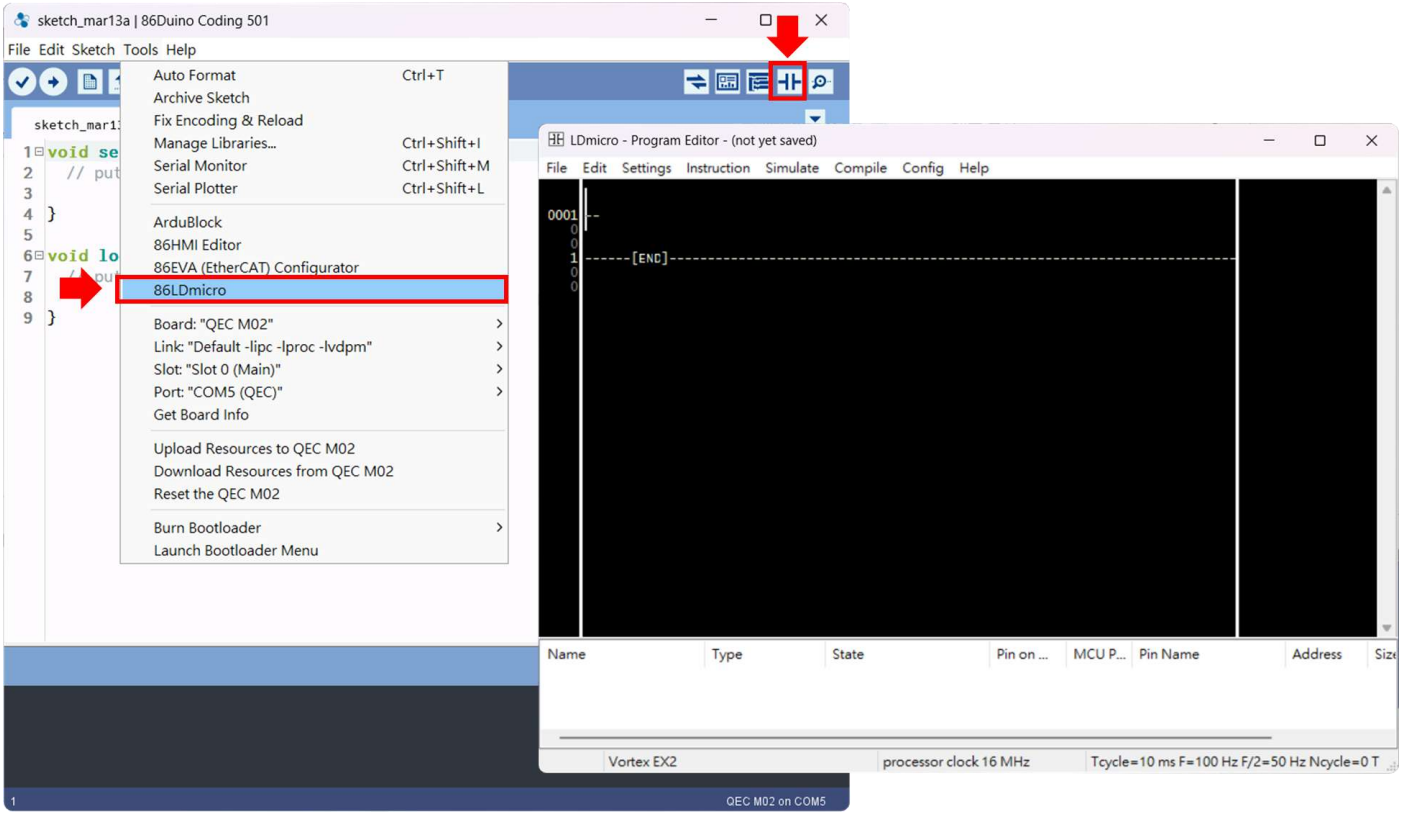
LDmicro is a PLC ladder diagram programming tool integrated into the 86Duino IDE. It is used to configure QEC-M-02 using ladder logic, making it an ideal choice for users familiar with traditional PLC programming.



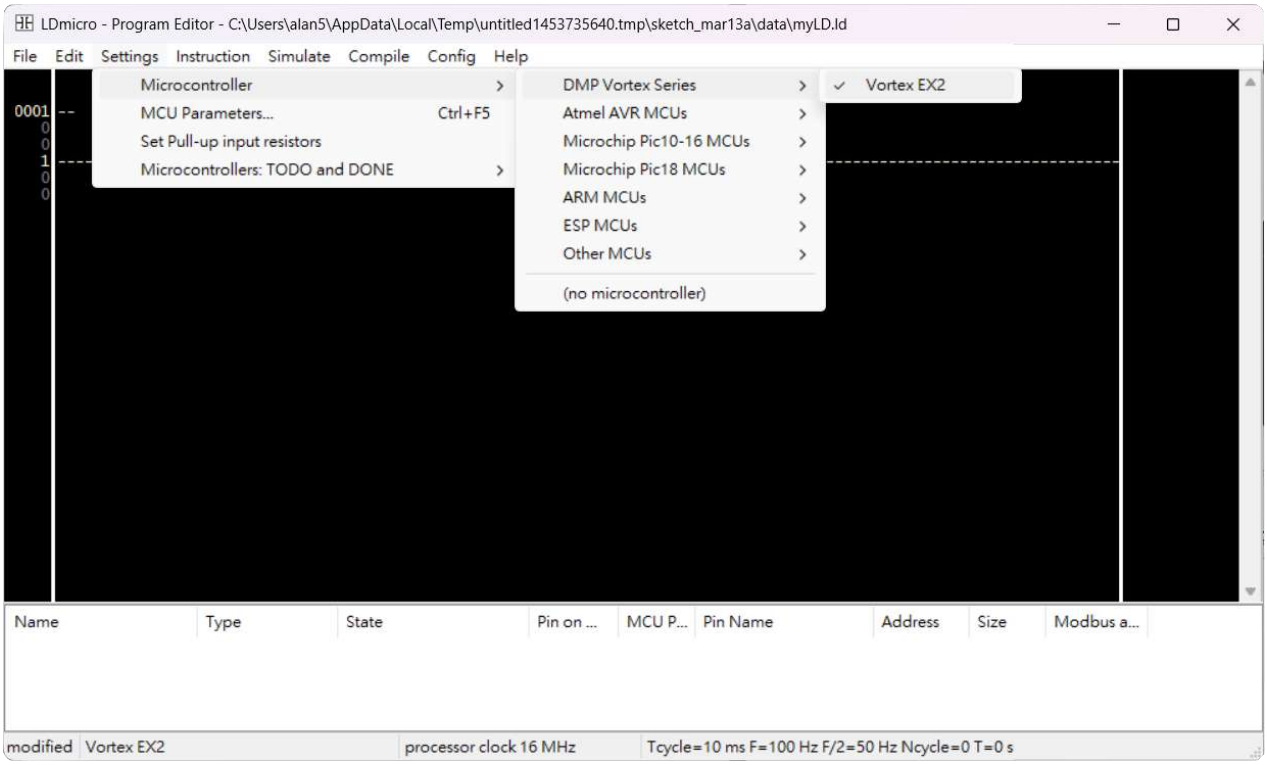
LDmicro generates native code for certain Microchip PIC16 and Atmel AVR microcontrollers. Usually, software for these microcontrollers is written in a programming language like assembler, C, or BASIC. A program in one of these languages comprises a list of statements. These languages are powerful and well-suited to the architecture of the processor, which internally executes a list of instructions.

The LDmicro tool in the 86Duino IDE is specifically designed for QEC MDevices, including QEC-M-02. Once the ladder logic design is completed in LDmicro, it automatically generates 86Duino-compatible code, which can be uploaded directly to the QEC-M-02.

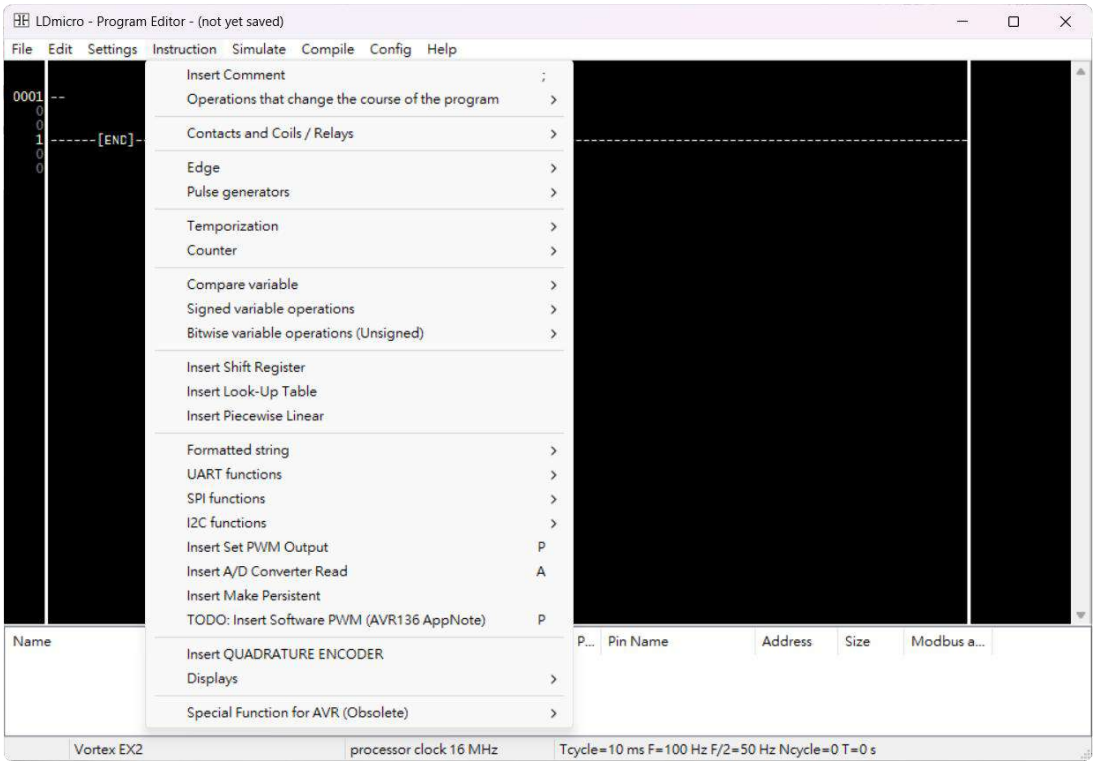
Users can use the following buttons to open the LDmicro.



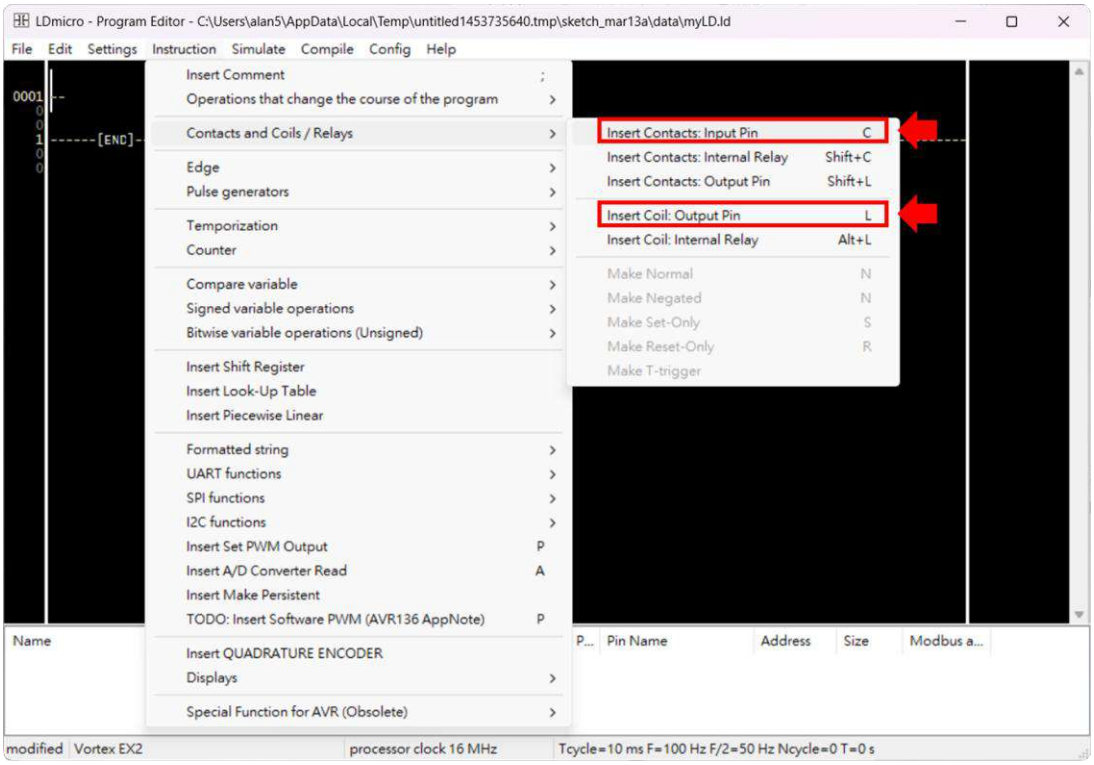
In LDmicro, users can check the platform currently using **“Settings > MicroController > DMP Vortex Series”**. Inside, there is a **VEX2** option, which is selected by default.



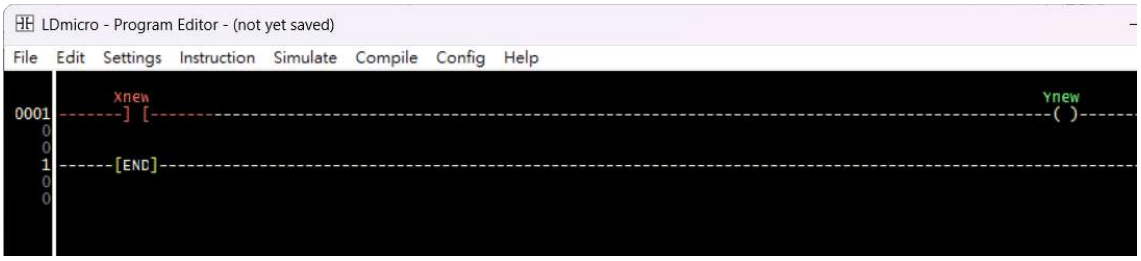
Users can use the ladder logic functions in the **“Instruction”** menu, which provides various options for inserting logic elements into ladder diagrams.



For example, users can use **“Insert Contacts: Input Pin”** to control digital input pins (X) and **“Inset Coil: Output Pin”** to control digital output pins (Y) in the **“Contacts and Coils / Relays”**.

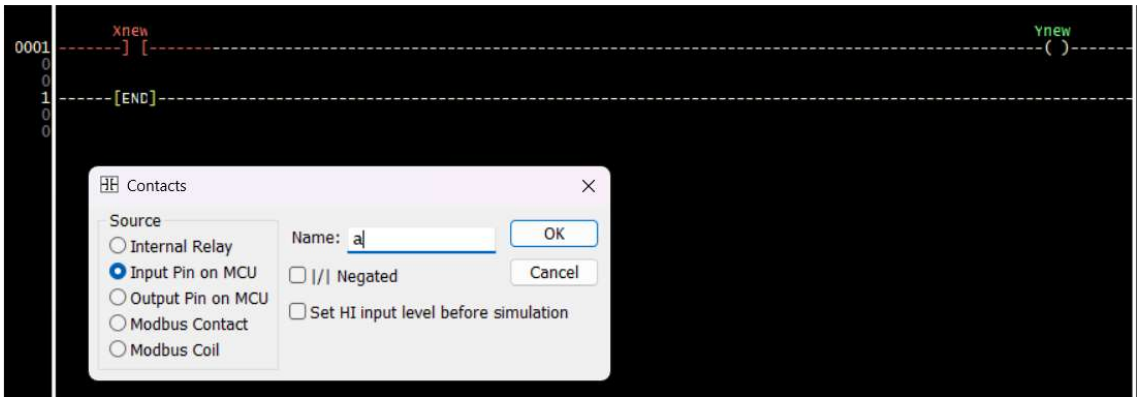


After you create **“Insert Contacts: Input Pin”** and **“Inset Coil: Output Pin”**, Xnew and Ynew will appear. In this example, we use **“Insert Contacts: Input Pin”** (Xnew) in the front.

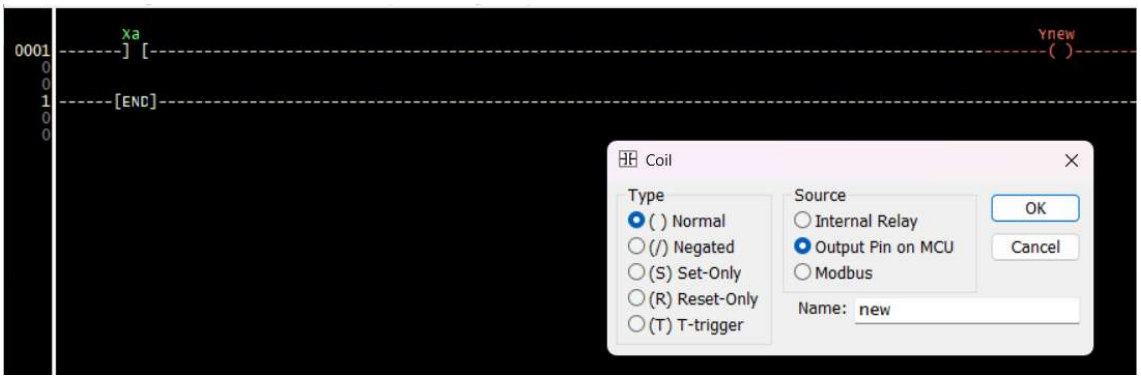


Users need to rename the object's name. We set the input to "Xa" and the output to "Ya". Users can double-click the object, and a setup window will appear, allowing them to set the source and name for the object.

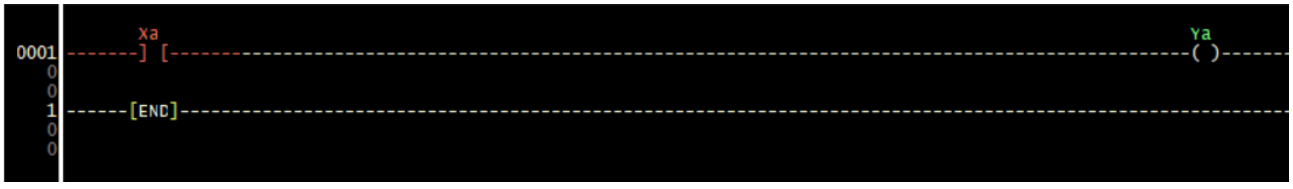
Input Pin X:



Output Pin Y:

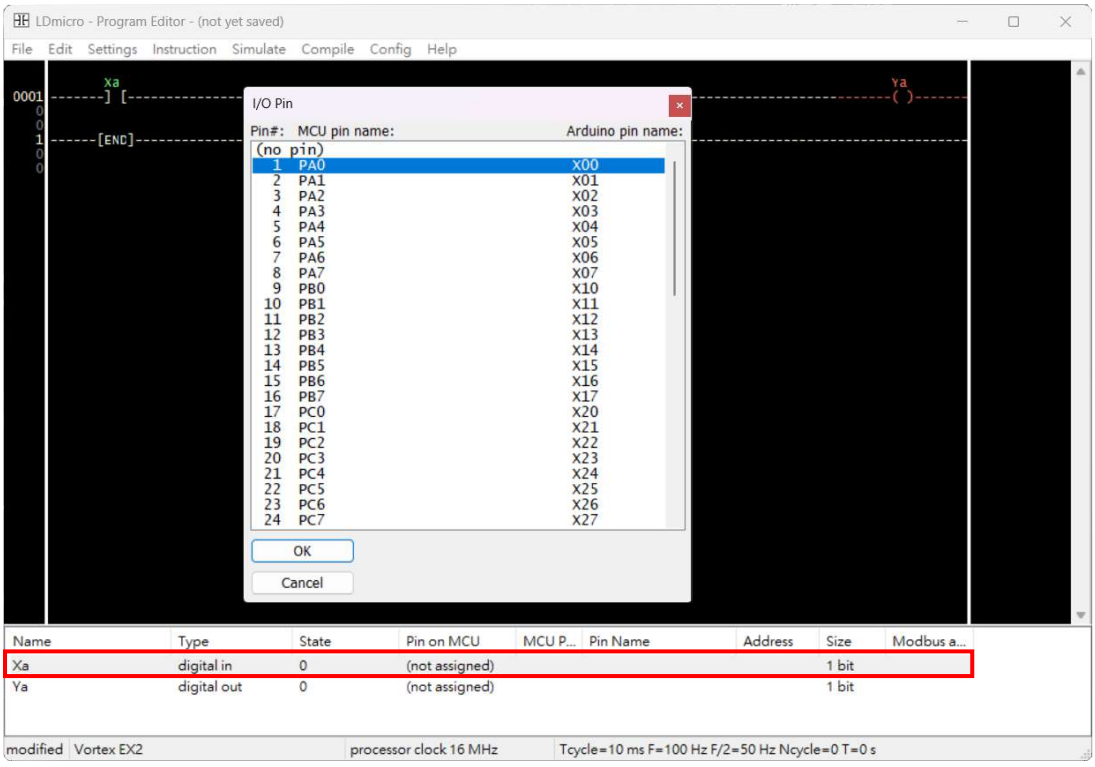


After changing the names of the objects, they look like the image below.

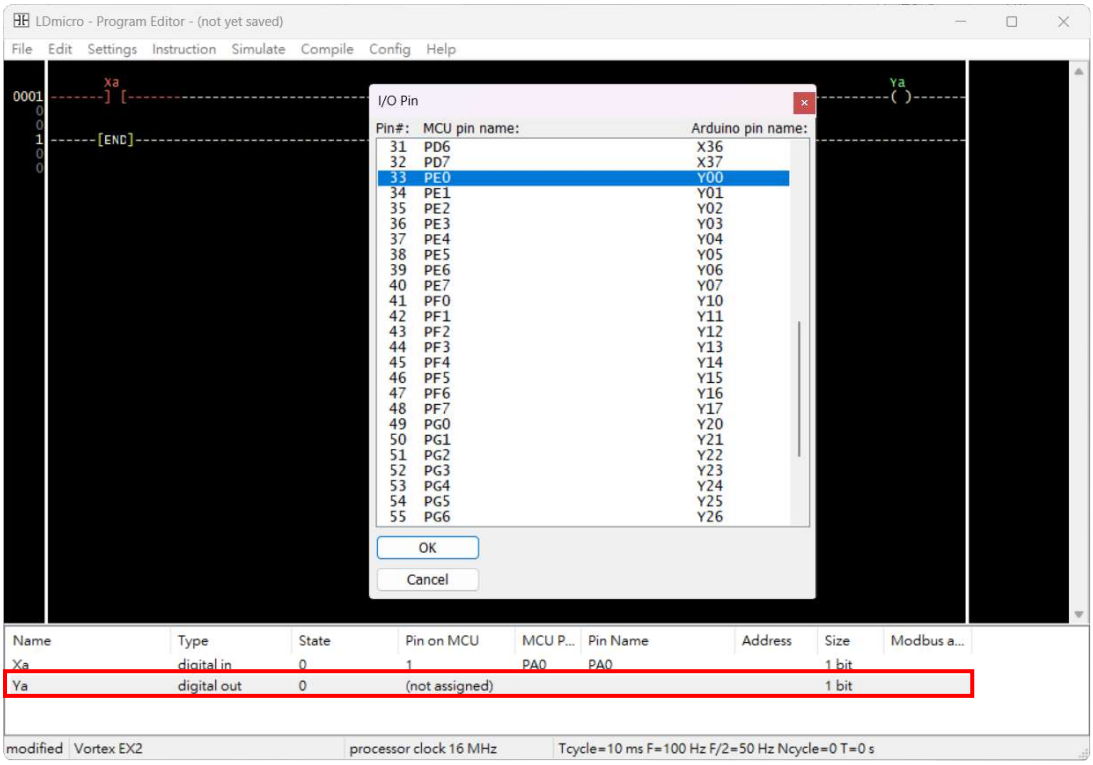


Then, users can select the pin assignment to the QEC-M-02 by clicking the list at the bottom.
In this example, we configure Xa to X00 and Ya to Y00.
To learn the digital pins assignment of QEC-M-02, refer to Chapter 2.2.6 Digital I/O Connector.

Xa:

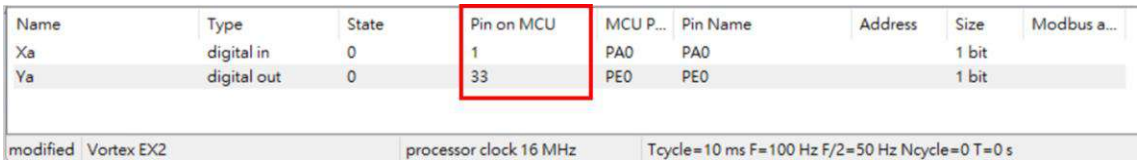


Ya:

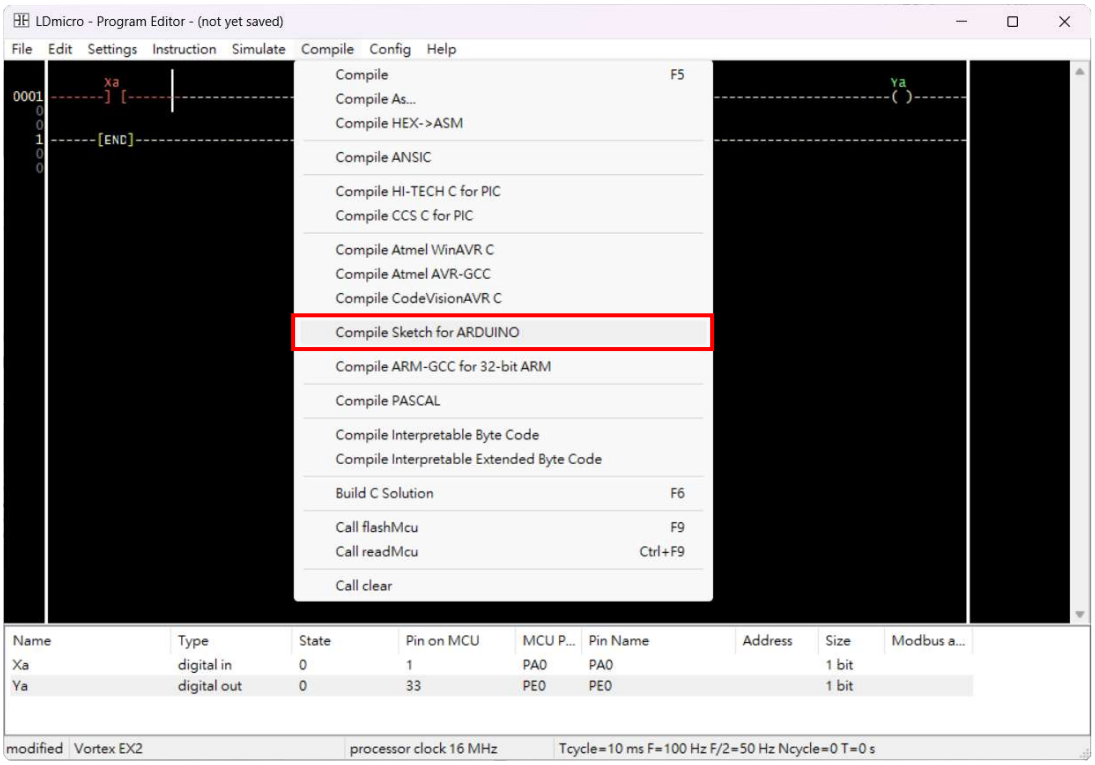


*Note: If the contact's name is kept in "**Xnew**" or "**Ynew**", it can't be assigned to the hardware pin.

The bottom windows will show the pin assignment in the **"Pin on MCU"** column.



After configuration, users need to click the **"Compile Sketch for ARDUINO"** button in the **"Compile"** menu so LDmicro can automatically generate the 86Duino code for your project.

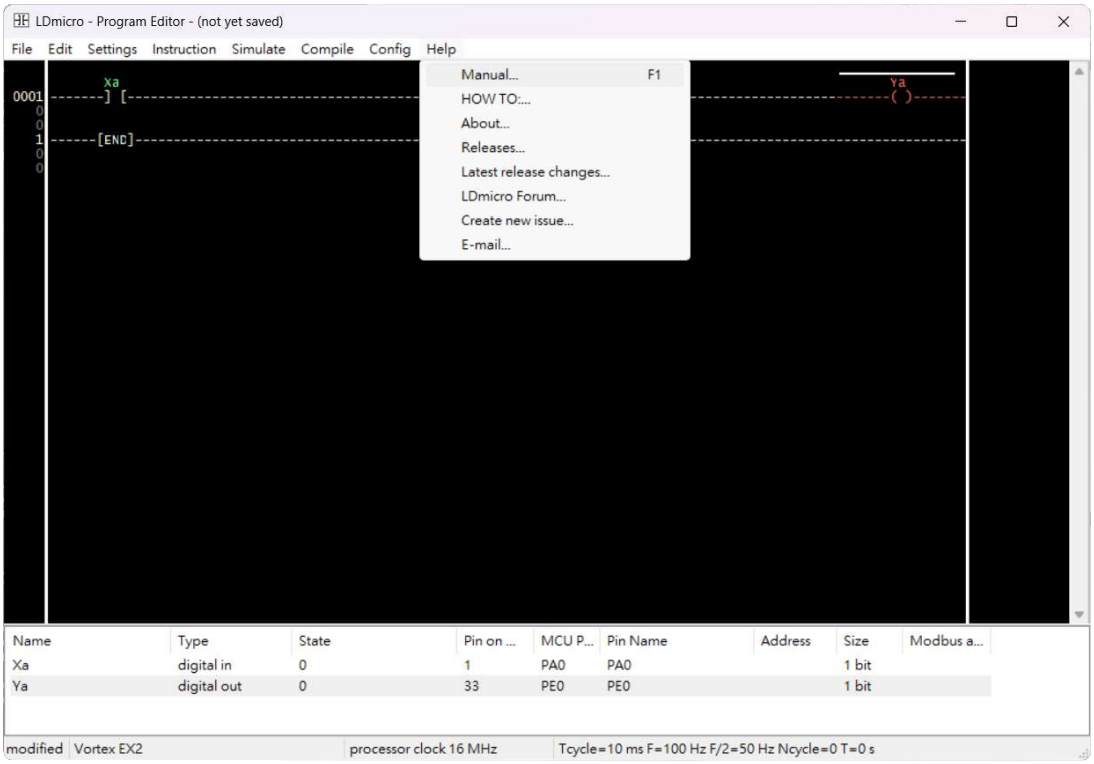


The generated code and files are as follows:

a. sketch_mar13a: Main Project (.ino, depending on your project name).
b. ladder.h: LDmicro parameters header file.
c. sketch_mar13a.cpp: C++ program code of LDmicro.
d. sketch_mar13a.h: Header file of LDmicro.

Once the code is completed, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload. The program will run when the upload is complete.

LDmicro Help Menu



The Help menu in LDmicro provides quick access to documentation and support:

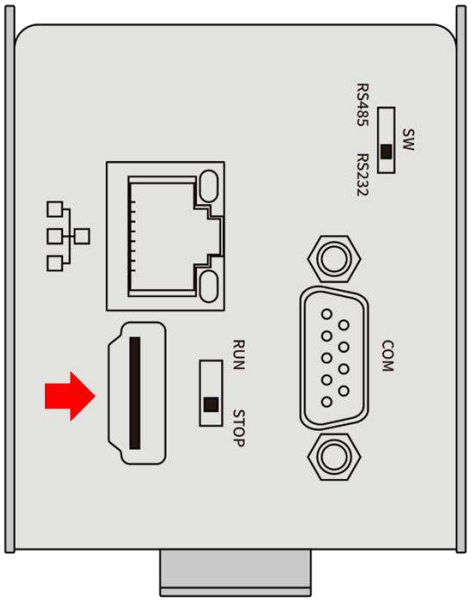
- Manual (F1) – Opens the LDmicro User Manual.
- HOW TO... – Step-by-step guides for common tasks.
- About... – Displays software version info.
- Releases & Latest Changes – Shows updates and release notes.
- LDmicro Forum – Connects to the online community for discussions.
- Create New Issue – Report bugs or request features.
- E-mail... – Contact support via email.

4.11 HDMI Display: LVGL

This section will demonstrate how to build a basic HMI on the QEC-M-02 using use the LVGL library or the 86HMI Editor in the 86Duino Coding IDE 501+.

We assume that you have already completed the previous sections of the quick start guide, including Package Contents, Hardware Configuration, Software Driver Installation, and set up the QEC-M-02.

There is a HDMI 2.0 port on the bottom side of QEC-M-02. Users can connect it to a monitor to show the display designed by users.



- Display Resolution: 1280 x 720 x 256

4.11.1 Library Instruction

LVGL (Light and Versatile Graphics Library) is a powerful open-source graphics library that enables the creation of attractive and interactive graphical user interfaces (GUIs) for embedded systems. By utilizing the LVGL library in combination with 86Duino IDE, developers can design and implement HMIs with ease and efficiency.

You can expand and customize the HMI by utilizing other LVGL widgets such as buttons, sliders, graphs, images, and more. LVGL provides a wide range of built-in widgets and customization options to create sophisticated and visually appealing HMIs tailored to your specific application.

For more details about LVGL, please see https://docs.lvgl.io/7.11/index.html.

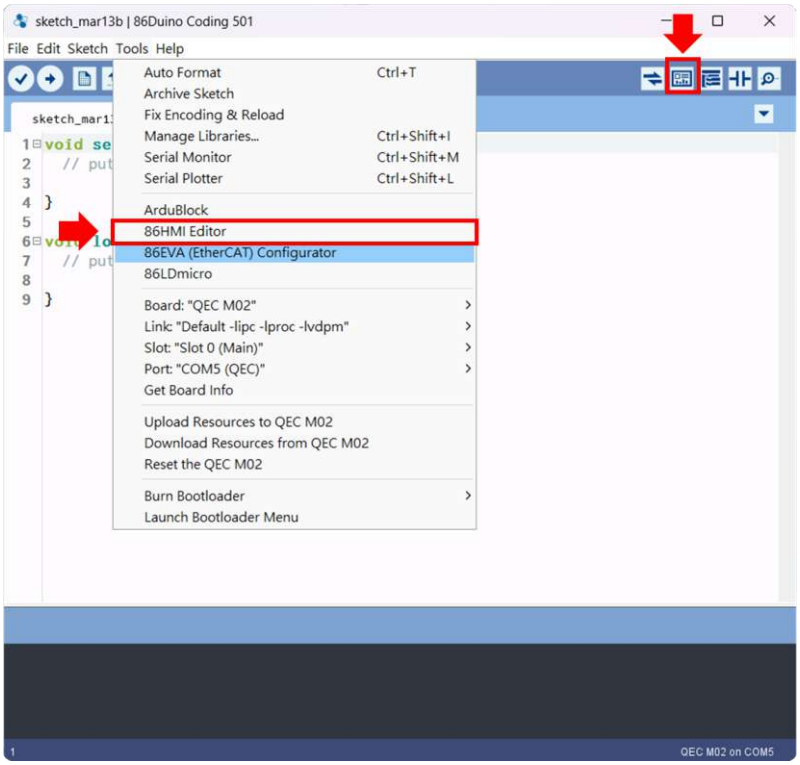
*Note: 86Duino IDE 501+ has already tested the LVGL version 9.3, so users can import the LVGL library manually.

4.11.2 Using the Graphical HMI editor: 86HMI Editor

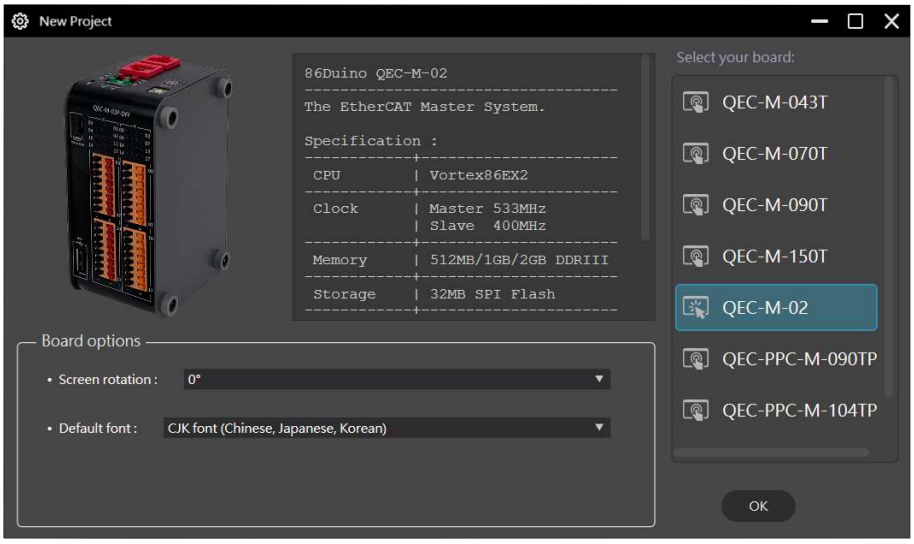
86HMI Editor is an easy-to-use HMI editor based-on LVGL version 7.11, that can be used to create a customized HMI quickly. Use the Auto Code Generation function in 86HMI Editor to generate HMI APIs (Application Programming Interface), thus achieving the effect of creating HMIs without writing programs.

Below are the complete steps to design UI to QEC-M-02 using 86HMI:

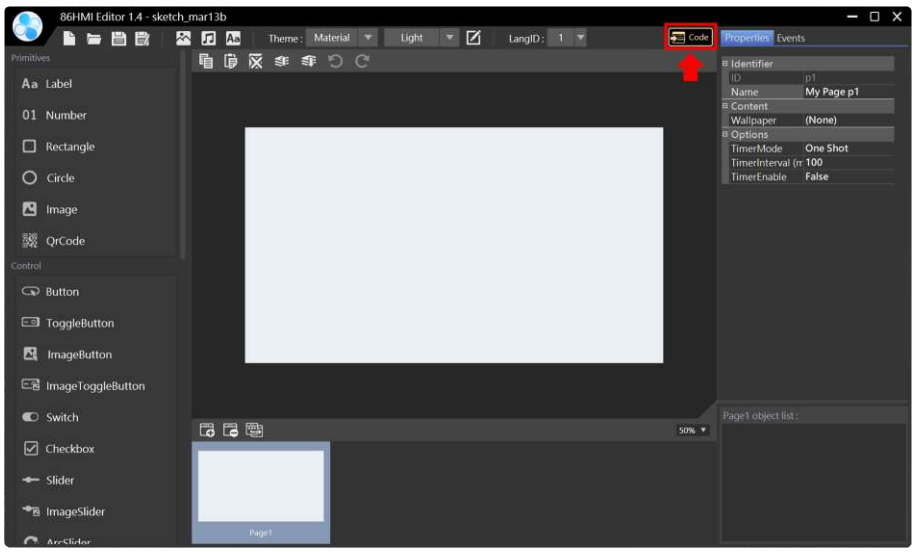
1. Open 86HMI tool via 86Duino Coding IDE 501+.



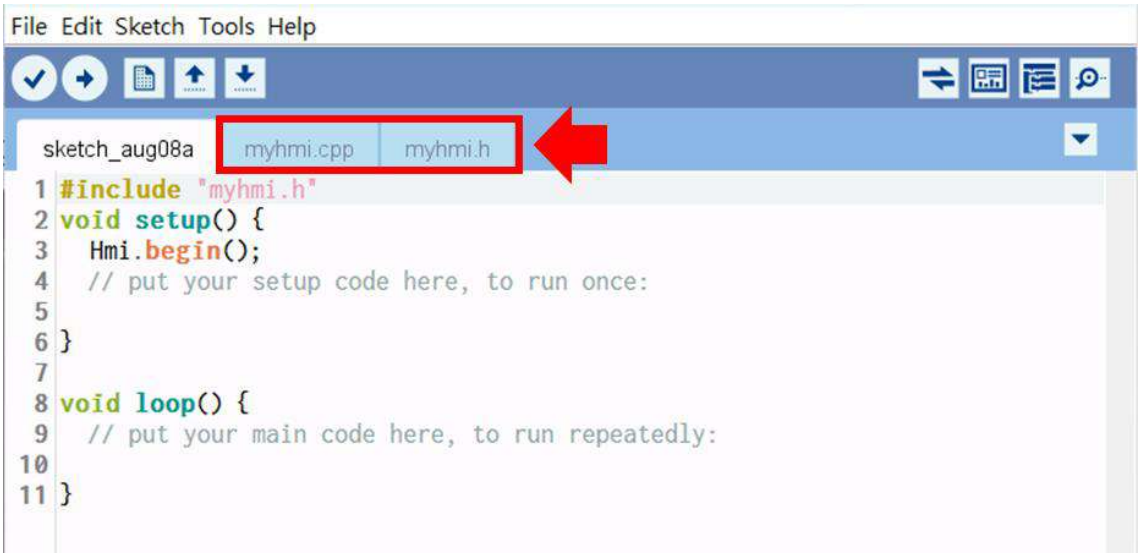
2. Choose QEC-M-02.



3. Then, you can use the left menu to start designing your UI and click the "**Code**" button to generate the source code back to 86Duino IDE automatically.



After generating, you can see myhmi.h and myhmi.cpp in 86Duino IDE.



And after you finish uploading, you can see the user interface you just designed on QEC-M-02.

For more details about 86HMI Editor, please refer to the 86HMI user manual: https://www.qec.tw/86duino/86hmi/.

Ch. 5

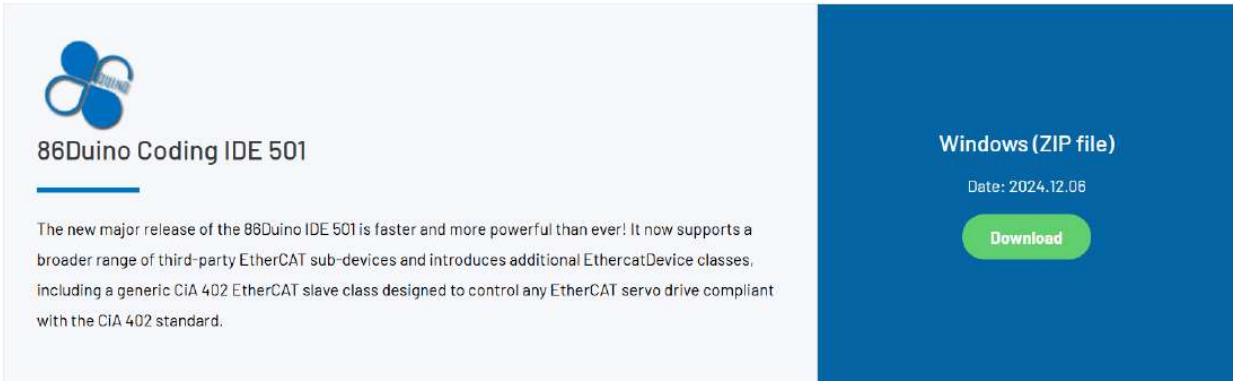
Software Function

5.1 Software Description

The 86Duino Coding IDE 500+ developed by the QEC team designed specifically for industrial-field control systems, bringing simple and powerful functions into related industrial fields through the open-source Arduino.

The 86Duino integrated development environment (IDE) software makes it easy to write code and upload it to 86Duino boards. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software.

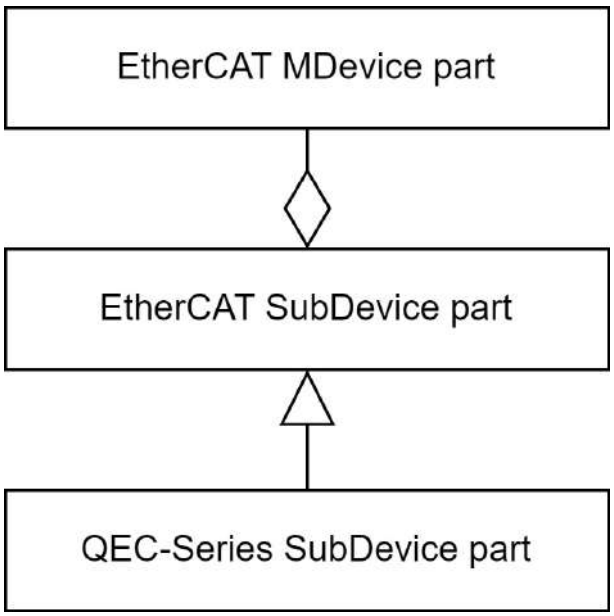
Please visit qec.tw for 86Duino Coding IDE 501+ details.



You can Download here: https://www.qec.tw/software/.

5.2 EtherCAT Function List

QEC-MDevice is an EtherCAT MDevice library implemented in C/C++, which includes classes for the MDevice, generic SubDevice, CiA 402 SubDevice, and dedicated classes for QEC series SubDevices. These classes not only have clearly defined responsibilities but also consider future extensibility.



These classes can be divided into three parts as follows:

- EtherCAT MDevice
  The *EtherCAT MDevice part* not only provides various and flexible MDevice configuration and operation functions but also offers diverse EtherCAT SubDevice operation functions for invocation by the EtherCAT SubDevice part.
- EtherCAT SubDevice
  The *EtherCAT SubDevice part* provides generic EtherCAT SubDevice classes, which can operate functions such as PDOs, CoE, FoE, and also includes CiA 402 SubDevice generic class.
- QEC-Series SubDevice
  The *QEC-Series SubDevice part* provides dedicated functions for ICOP's QEC series SubDevices, enabling users to code in a more user-friendly and concise manner.

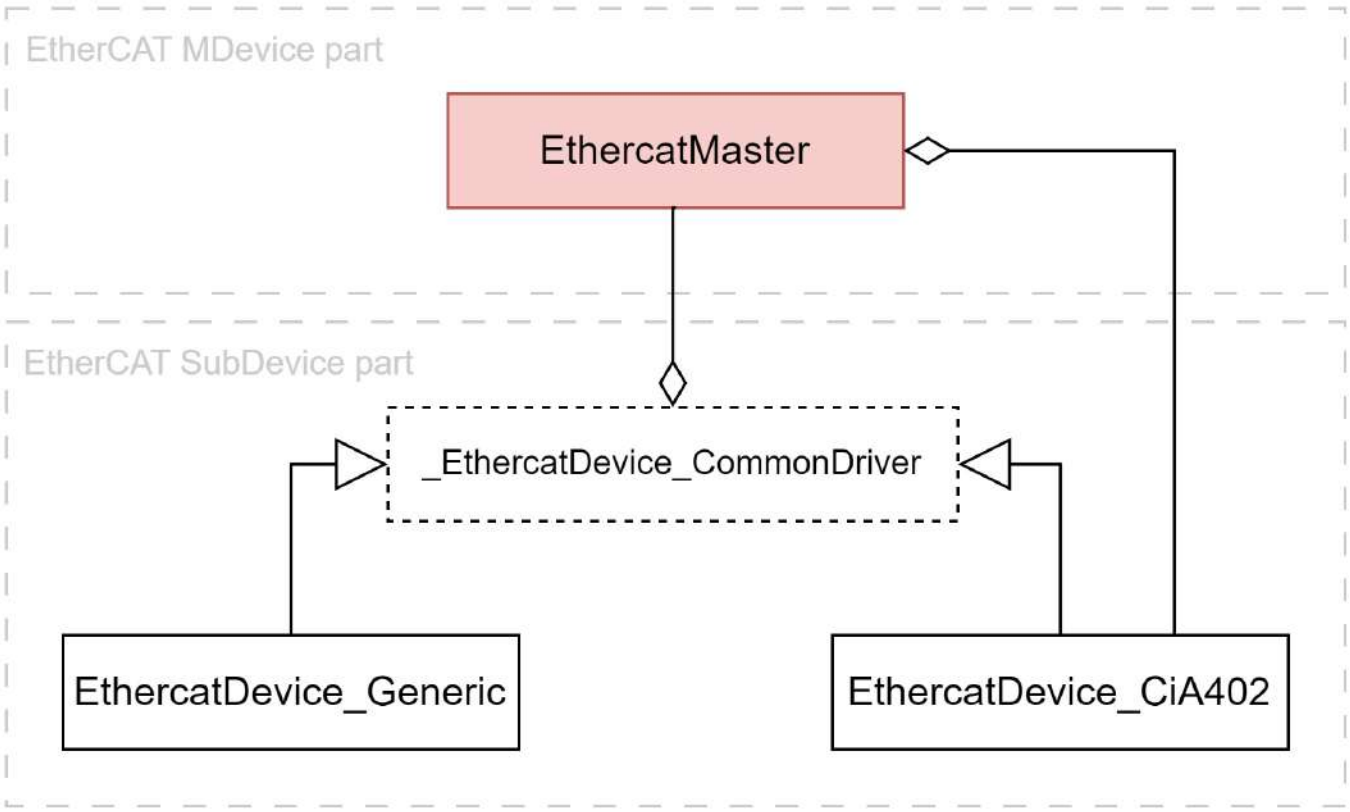
The list below introduces the EtherCAT Library API functions of our QEC MDevice.
Please visit the EtherCAT Library API User Manual for details on the API Function.

5.2.1 EtherCAT MDevice

The EtherCAT MDevice part not only provides various and flexible MDevice configuration and operation functions but also offers diverse EtherCAT SubDevice operation functions for invocation by the EtherCAT SubDevice part.

EthercatMaster is the only class in the EtherCAT MDevice part, it serves as a crucial communication bridge with the EtherCAT firmware. In the Dual-System communication aspect, its responsibilities include communication interface initialization, process data exchange cyclically, handling acyclic transfer interfaces, and managing interrupt events. In the API aspect, it provides functions related to MDevice initialization, MDevice control, and access to SubDevice information.

The main class relationship between the EtherCAT MDevice part and the EtherCAT SubDevice part is association, with the EtherCAT SubDevice part depending on the EtherCAT MDevice part. The class relationships of EthercatMaster are illustrated in the following diagram:



- There is an association between EthercatMaster and _EthercatDevice_CommonDriver, with _EthercatDevice_CommonDriver depending on EthercatMaster.
- There is an association between EthercatMaster and EthercatDevice_CiA402, with EthercatMaster depending on EthercatDevice_CiA402.

Functions:

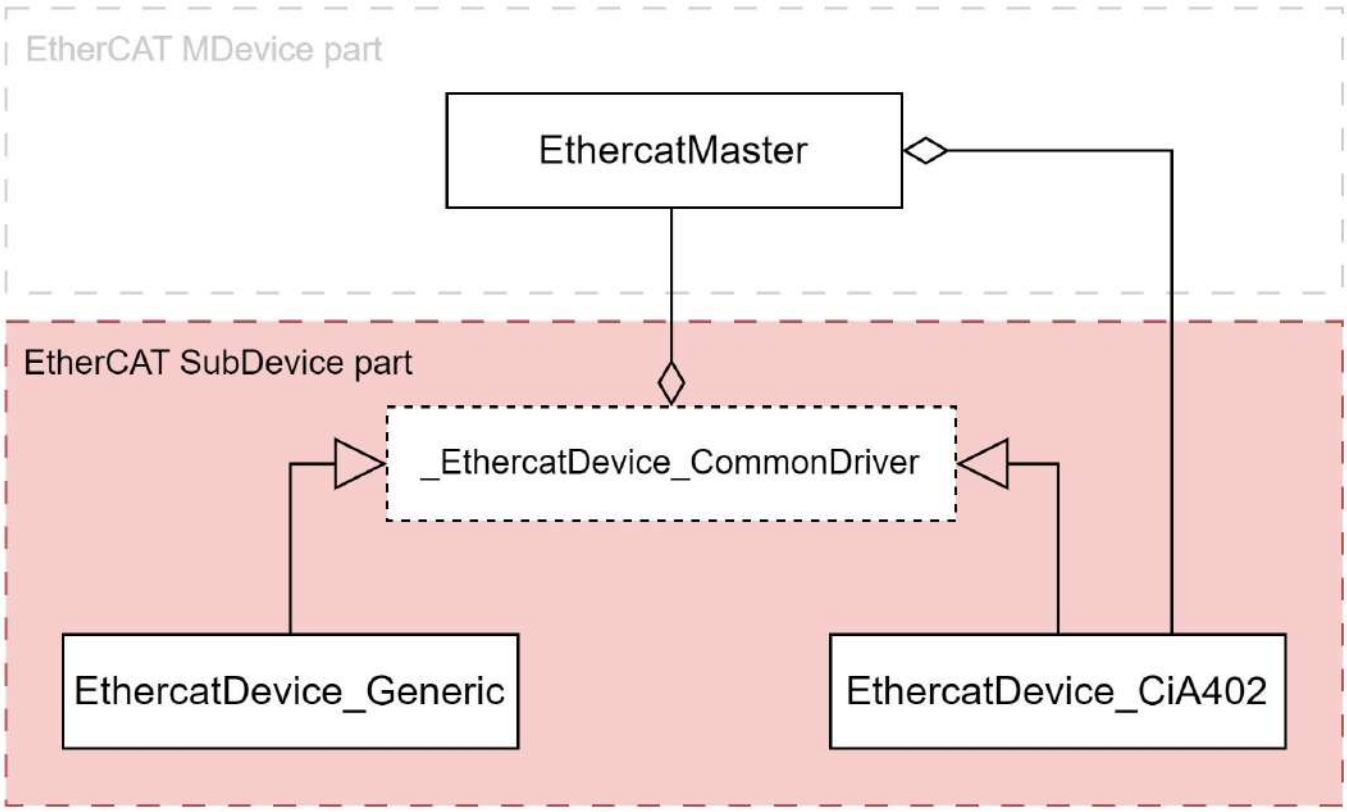
Function Name	Description	Callback Available
Initialization-related functions		
begin()	Initialize the EtherCAT MDevice.	
end()	Deinitialize the EtherCAT MDevice.	
isRedundancy()	Check if the EtherCAT MDevice has cable redundancy enabled.	O
libraryVersion()	Get the EtherCAT MDevice library version.	O
firmwareVersion()	Get the EtherCAT firmware version.	O
readSettings()	Read the current EtherCAT MDevice settings.	
saveSettings()	Save the EtherCAT MDevice settings.	
Control-related functions		
start()	Start the EtherCAT MDevice.	
stop()	Stop the EtherCAT MDevice.	
update()	Update process data and handle acyclic commands.	O
setShiftTime()	Set the Global Shift Time for DC-Synchronous mode.	
getShiftTime()	Get the Global Shift Time for DC-Synchronous mode.	O
getSystemTime()	Get the system time of the current cycle.	O
getWorkingCounter()	Get the working counter for the current cycle.	O
getExpectedWorkingCounter()	Get the expected working counter.	O
Callback-related functions		
attachCyclicCallback()	Register a cyclic callback.	
detachCyclicCallback()	Unregister cyclic callback.	
attachErrorCallback()	Register an error callback.	
detachErrorCallback()	Unregister error callback.	
attachEventCallback()	Register an event callback.	
detachEventCallback()	Unregister event callback.	
errGetCableBrokenLocation1()	Get the cable broken location 1 in error callback.	O[1]
errGetCableBrokenLocation2()	Get the cable broken location 2 in error callback.	O[1]
evtGetMasterState()	Get the EtherCAT MDevice state in event callback.	O[2]
SubDevice information related functions		
getSlaveCount()	Get the number of SubDevices on the network.	O
getVendorID()	Get the vendor ID of the specified SubDevice.	O
getProductCode()	Get the product code of the specified SubDevice.	O
getRevisionNumber()	Get the revision number of the specified SubDevice.	O
getSerialNumber()	Get the serial number of the specified SubDevice.	O
getAliasAddress()	Get the alias address of the specified SubDevice.	O
getSlaveNo()	Find the sequence number of the matching EtherCAT SubDevice on the network.	O

- Note 1: This function can only be called in error callback.
- Note 2: This function can only be called in event callback.

5.2.2 EtherCAT SubDevice

The EtherCAT SubDevice part provides generic EtherCAT SubDevice classes, which can operate functions such as PDOs, CoE, FoE, and also includes CiA 402 SubDevice generic class.

The main class relationship between the EtherCAT SubDevice part and the EtherCAT MDevice part is association, with the EtherCAT SubDevice part depending on the EtherCAT MDevice part. As shown in the diagram below, there is an association relationship between _EthercatDevice_CommonDriver and EthercatMaster.



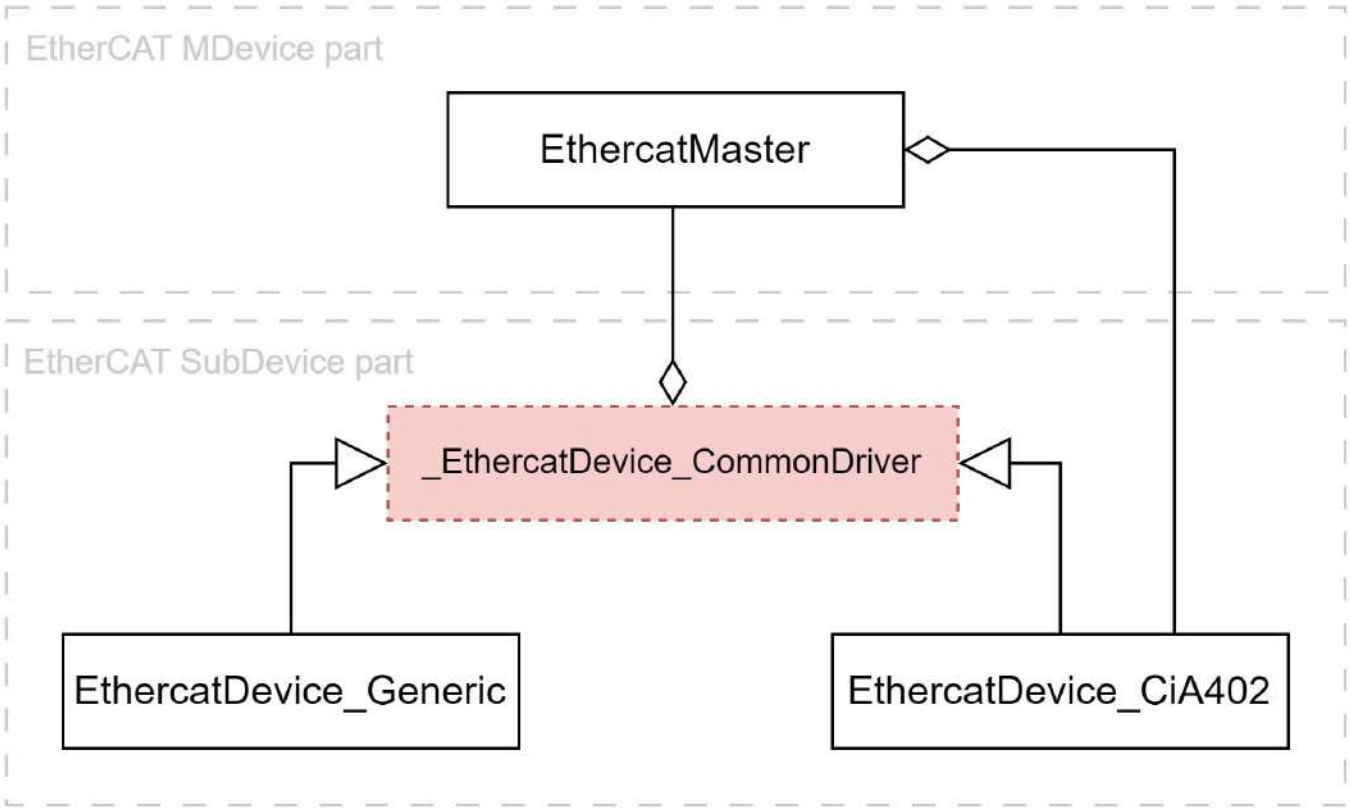
Classes:

- _EthercatDevice_CommonDriver
- EthercatDevice_Generic
- EthercatDevice_CiA402

5.2.2.1 _EthercatDevice_CommonDriver

_EthercatDevice_CommonDriver is an abstract class that not only features functions for accessing SubDevice information but also provides various EtherCAT function access methods, including PDO, SII, CoE, FoE, DC, etc. All EtherCAT SubDevice classes inherit from it.

The class relationships of _EthercatDevice_CommonDriver are illustrated in the following diagram:



- There is an association between EthercatMaster and _EthercatDevice_CommonDriver, with _EthercatDevice_CommonDriver depending on EthercatMaster.
- All other EtherCAT SubDevice classes inherit from _EthercatDevice_CommonDriver.

WARNING: *Prohibited from declaring objects using this class.*

Functions:

Function Name	Description	Callback Available
SubDevice information related functions		
`getVendorID()`	Get the vendor ID.	O
`getProductCode()`	Get the product code.	O
`getRevisionNumber()`	Get the revision number.	O
`getSerialNumber()`	Get the serial number.	O
`getAliasAddress()`	Get the alias address.	O
`getSlaveNo()`	Get the sequence ID on the EtherCAT network.	O

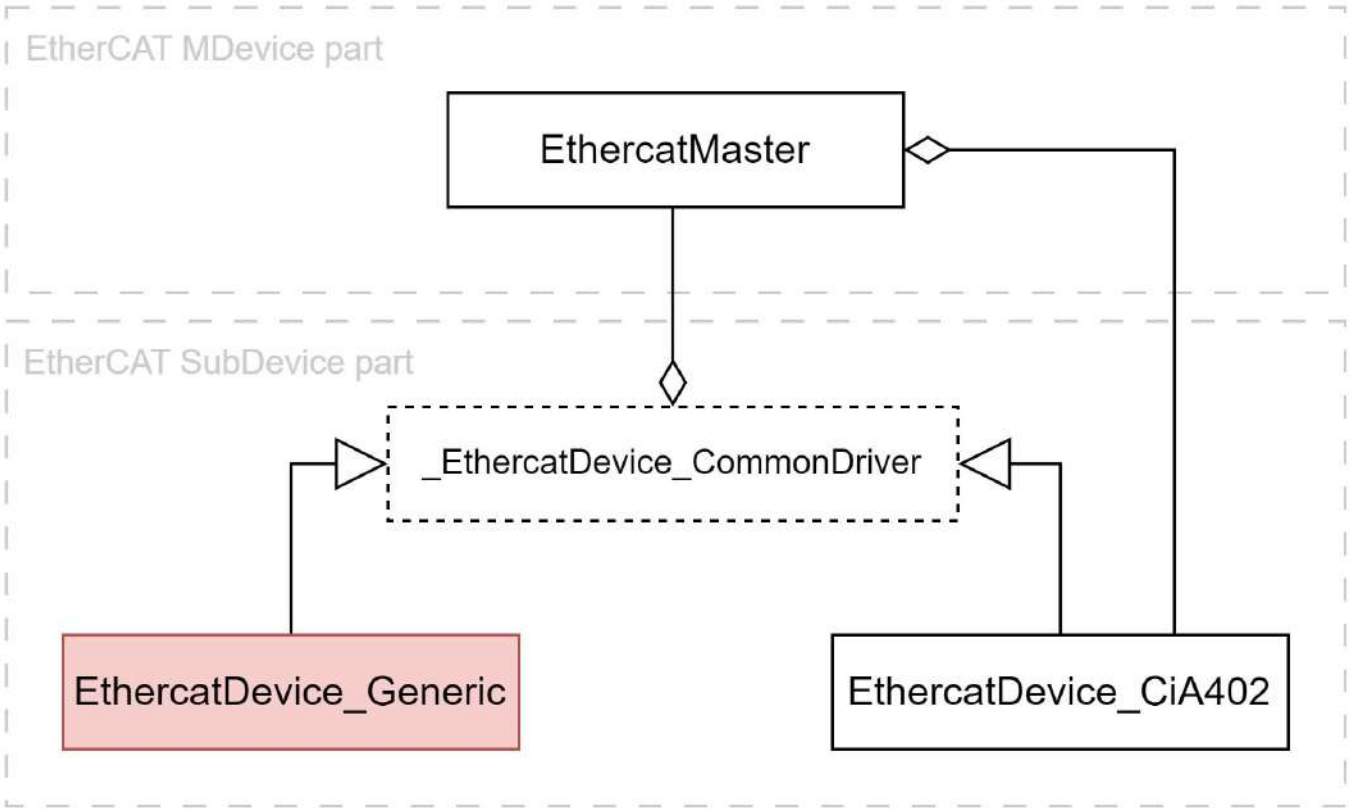
getDeviceName()	Get the device name.	O
getMailboxProtocol()	Get the supported mailbox protocol types.	O
getCoEDetails()	Get the details about CoE supported.	O
getFoEDetails()	Get the details about FoE supported.	O
getEoEDetails()	Get the details about EoE supported.	O
getSoEChannels()	Get the number of SoE channels supported.	O
isSupportDC()	Check if the EtherCAT SubDevice has DC supported.	O
PDO access functions		
pdoBitWrite()	Write 1-bit output process data.	O
pdoBitRead()	Read 1-bit input process data.	O
pdoGetOutputBuffer()	Get the memory pointer of output process data.	O
pdoGetInputBuffer()	Get the memory pointer of input process data.	O
pdoWrite()	Write multiple bytes of output process data.	O
pdoWrite8()	Write 8-bit output process data.	O
pdoWrite16()	Write 16-bit output process data.	O
pdoWrite32()	Write 32-bit output process data.	O
pdoWrite64()	Write 64-bit output process data.	O
pdoRead()	Read multiple bytes of input process data.	O
pdoRead8()	Read 8-bit input process data.	O
pdoRead16()	Read 16-bit input process data.	O
pdoRead32()	Read 32-bit input process data.	O
pdoRead64()	Read 64-bit input process data.	O
CoE communication functions		
sdoDownload()	Write multiple bytes of data to the object.	
sdoDownload8()	Write 8-bit value to the object.	
sdoDownload16()	Write 16-bit value to the object.	
sdoDownload32()	Write 32-bit value to the object.	
sdoDownload64()	Write 64-bit value to the object.	
sdoUpload()	Read multiple bytes of data from the object.	
sdoUpload8()	Read 8-bit value from the object.	
sdoUpload16()	Read 16-bit value from the object.	
sdoUpload32()	Read 32-bit value from the object.	
sdoUpload64()	Read 64-bit value from the object.	
getODlist()	Get a list of objects existing in the object dictionary.	
getObjectDescription()	Get the object description of the object.	
getEntryDescription()	Get the entry description of the object.	
FoE communication functions		
readFoE()	Read a file from the EtherCAT SubDevice.	
writeFoE()	Write a file to the EtherCAT SubDevice.	
DC configuration functions		
setDc()	Configure DC parameters.	

SII EEPROM access functions		
writeSII()	Write multiple bytes of data to the SII EEPROM.	
writeSII8()	Write 8-bit value to the SII EEPROM.	
writeSII16()	Write 16-bit value to the SII EEPROM.	
writeSII32()	Write 32-bit value to the SII EEPROM.	
readSII()	Read multiple bytes of data from the SII EEPROM.	
readSII8()	Read 8-bit value from the SII EEPROM.	
readSII16()	Read 16-bit value from the SII EEPROM.	
readSII32()	Read 32-bit value from the SII EEPROM.	
Initialization-related functions		
attach()	Initialize the object of this EtherCAT SubDevice class.	
detach()	Deinitialize the object of this EtherCAT SubDevice class.	

5.2.2.2 EthercatDevice_Generic

EthercatDevice_Generic is a generic EtherCAT SubDevice class that can be used to control all EtherCAT SubDevices, including accessing SubDevice information, PDO, CoE, FoE, DC, and more.

The class relationships of *EthercatDevice_Generic* are illustrated in the following diagram:



- EthercatDevice_Generic inherits from _EthercatDevice_CommonDriver.

Base Class

- _EthercatDevice_CommonDriver

Functions

Function Name	Description	Callback Available
Initialization-related functions		
attach()	Initialize the object of this EtherCAT SubDevice class.	
detach()	Deinitialize the object of this EtherCAT SubDevice class.	

5.2.2.3 EthercatDevice_CiA402

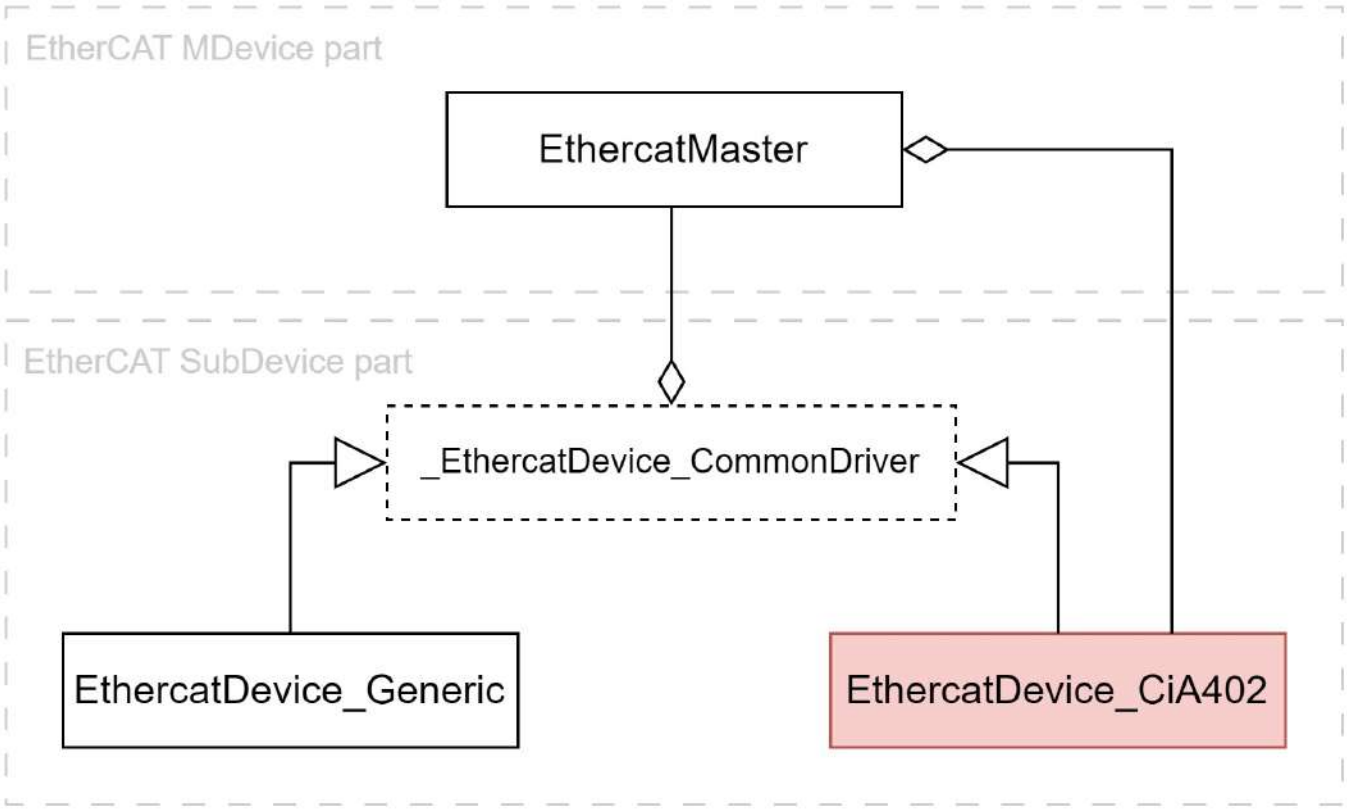
EthercatDevice_CiA402 is a generic CiA 402 EtherCAT SubDevice class designed to control any EtherCAT servo drive that supports the CiA 402 standard. It provides access functions for commonly used CiA 402 objects and operation functions for several CiA 402 operation modes and function groups, including:

- **Operation Modes**
  - Profile Position (pp)
  - Profile Velocity (pv)
  - Profile Torque (tq)
  - Homing (hm)
  - Cyclic Synchronous Position (csp)
  - Cyclic Synchronous Velocity (csv)
  - Cyclic Synchronous Torque (cst)
- **Function Groups**
  - Touch Probe

For more detailed information about CiA 402, please refer to the following documents:

- CiA Draft Standard 402: CANopen device profile drives and motion control
- ETG.6010 Implementation Directive for CiA402 Drive Profile
- User manual for the currently used CiA 402 drive device

The class relationships of EthercatDevice_CiA402 are illustrated in the following diagram:



- *EthercatDevice_CiA402* inherits from *_EthercatDevice_CommonDriver*.

Base Class

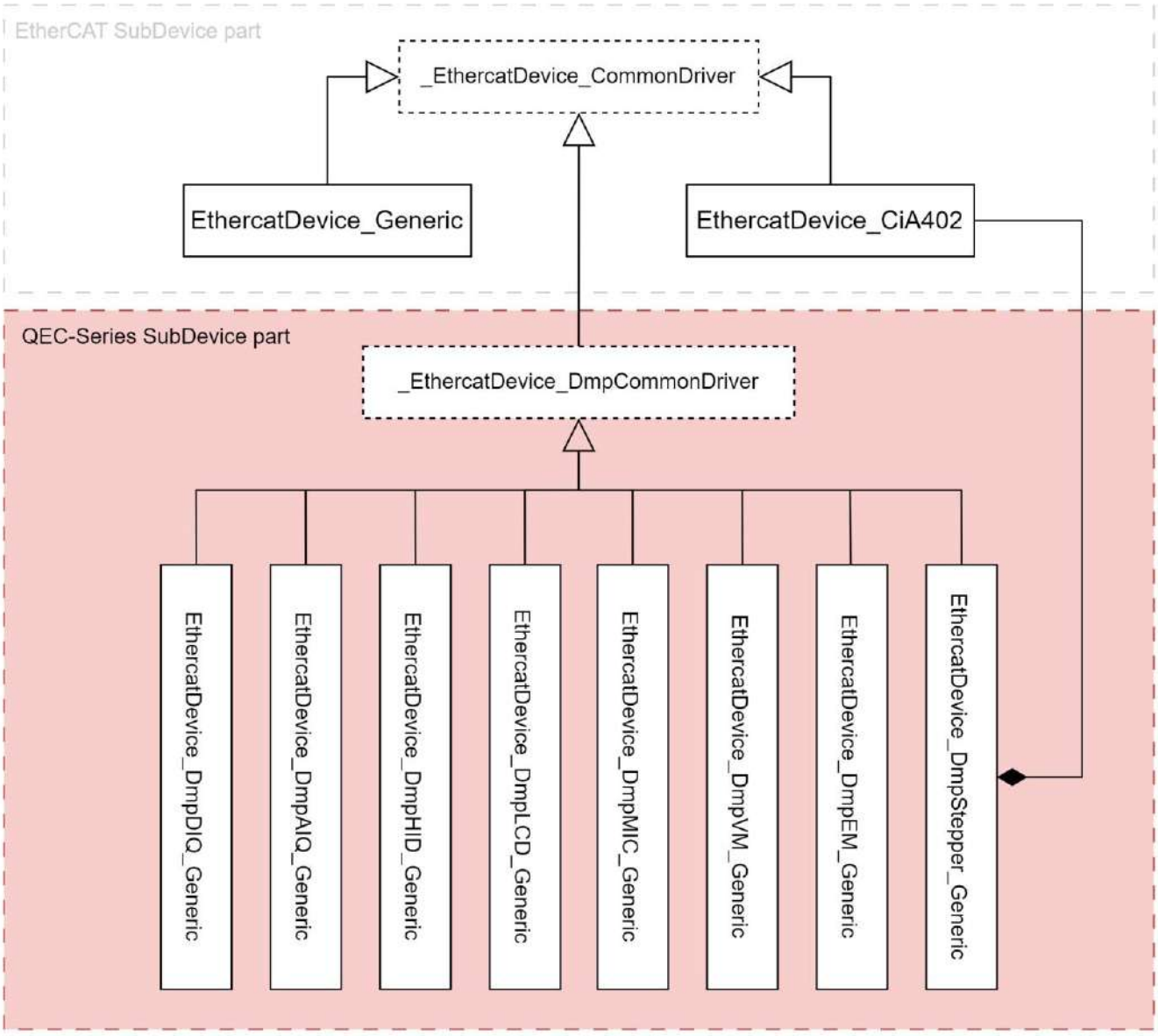
- _EthercatDevice_CommonDriver

For more detailed information about CiA 402 API, please refer to the EtherCAT Library API User Manual - QEC.

5.2.3 QEC-Series SubDevice

The QEC-Series SubDevice part provides dedicated functions for ICOP's QEC series SubDevices, enabling users to code in a more user-friendly and concise manner.

The main class relationship between the QEC-Series SubDevice part and the EtherCAT SubDevice part is association, with the QEC-Series SubDevice part depending on the EtherCAT SubDevice part. As shown in the diagram below, there is an association relationship between _EthercatDevice_DmpCommonDriver and _EthercatDevice_CommonDriver.



Classes

- _EthercatDevice_DmpCommonDriver
- EthercatDevice_DmpDIQ_Generic

- EthercatDevice_DmpAIQ_Generic
- EthercatDevice_DmpHID_Generic
- EthercatDevice_DmpLCD_Generic
- EthercatDevice_DmpStepper_Generic

5.3 Additional Resources

If you want to learn more about the libraries available in the 86Duino IDE or explore the details of the 86Duino programming language, please visit the following links:

- **EtherCAT Library API User Manual:** QEC MDevice is an EtherCAT MDevice compatible with 86Duino Coding IDE 500+. It offers real-time EtherCAT communication between EtherCAT MDevice and EtherCAT Sub-devices. Please refer detailed information at https://www.qec.tw/ethercat/api/ethercat-library-api-user-manual/.

- **86Duino IDE Libraries:** Find an extensive list of libraries supported by 86Duino IDE, along with detailed documentation and examples at https://www.qec.tw/86duino/libraries/.

- **86Duino Language Reference:** Learn about the 86Duino programming language, including its syntax, functions, and usage in the official 86Duino Language Reference at https://www.qec.tw/86duino/86duino-language-reference/.

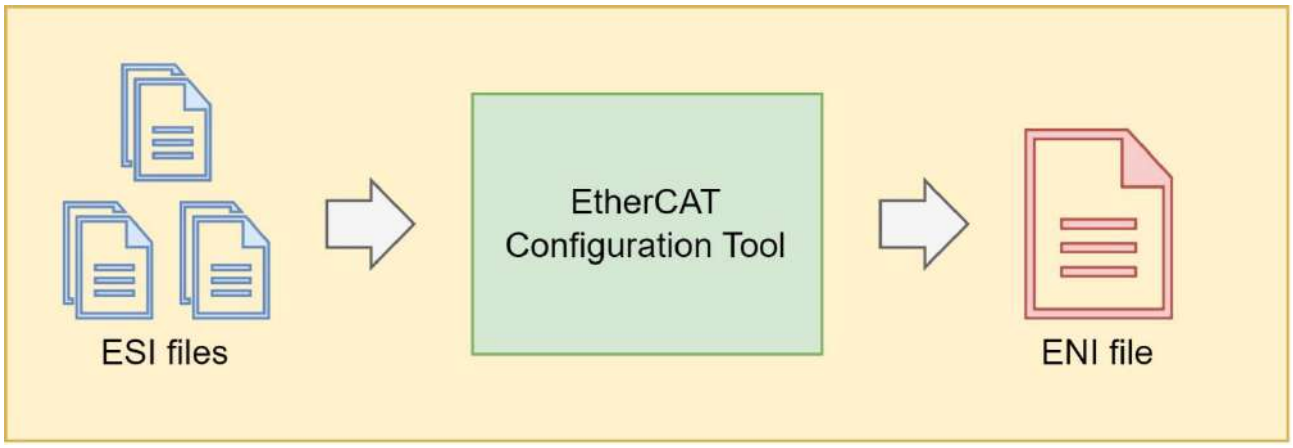
These resources provide valuable information for both beginners and experienced developers using the 86Duino platform. By exploring these links, you can harness the full potential of 86Duino IDE and create innovative projects.

Happy coding with 86Duino IDE!

Appendix

A1. About ENI Configuration in 86Duino IDE

The EtherCAT Network Information (ENI) contains the necessary settings to configure an EtherCAT network. The XML-based file contains general information about the EtherCAT MDevice and the configurations of every SubDevice connected to the EtherCAT MDevice. The EtherCAT Configuration Tool reads the ESI files or online scans the network for all SubDevices, then user can configures relevant EtherCAT settings, such as PDO mapping and enabling DC, and then export the ENI file.



The EtherCAT Technology Group specifies that the EtherCAT MDevice Software must support at least one of the following in the Network Configuration section: Online Scanning or Reading ENI. This library, however, supports both. In the case of Reading ENI, this library currently extracts only partial information from the ENI file for network configuration.

The extracted information includes the following:

EtherCATConfig : Config : SubDevice : Info

- Elements
  - VendorId
  - ProductCode
- Attribute
  - Identification : Value
- Purpose
  Used to check whether the EtherCAT SubDevices on the network match the SubDevices specified in the ENI file. The checking rules are as follows:
  - Check if the number of SubDevices in the ENI file matches the number of SubDevices on the network.
  - For SubDevices in the ENI file with the Identification: Value attribute, check if there are SubDevices on the network with matching Alias Address and Identification: Value attribute, as well as Vendor ID and Product Code. If such SubDevices exist, it indicates a successful match.
  - For SubDevices with the Identification: Value attribute that fail to match, or those without this attribute, check if the Vendor ID and Product Code of the SubDevices with the same sequence number on the network match.

EtherCATConfig : Config : SubDevice : Mailbox

- Elements
  - Send : MailboxSendInfoType : Start
  - Recv : MailboxRecvInfoType : Start
- Purpose
  Used to configure the mailbox Physical Start Address of an EtherCAT SubDevice.

EtherCATConfig : Config : SubDevice : Mailbox : CoE

- Elements
  - InitCmds : InitCmd : Index
  - InitCmds : InitCmd : SubIndex
  - InitCmds : InitCmd : Data
  - InitCmds : InitCmd : Timeout
- Attribute
  - InitCmds : InitCmd : CompleteAccess
- Purpose
  After switching the EtherCAT state machine to the Pre-Operational state, execute the CoE initialization commands for the EtherCAT SubDevice in `EthercatMaster::begin()`.

EtherCATConfig : Config : SubDevice : ProcessData

- Elements
  - Recv : BitLength
  - Send : BitLength
- Purpose
  The bit length of the output process data and input process data of an EtherCAT SubDevice is provided to the firmware for relevant configuration.

EtherCATConfig : Config : SubDevice : ProcessData : Sm

- Elements
  - SyncManagerSettings : StartAddress
  - SyncManagerSettings : ControlByte
  - SyncManagerSettings : Enable
- Purpose
  Used to configure the Sync Manager registers for the process data of an EtherCAT SubDevice.

EtherCATConfig : Config : SubDevice : DC

- Elements
  - CycleTime0
  - CycleTime1
  - ShiftTime
- Purpose
  Used to configure the DC parameters of an EtherCAT SubDevice.

Warranty

This product is warranted to be in good working order for a period of one year from the date of purchase. Should this product fail to be in good working order at any time during this period, we will, at our option, replace or repair it at no additional charge except as set forth in the following terms. This warranty does not apply to products damaged by misuse, modifications, accident or disaster. Vendor assumes no liability for any damages, lost profits, lost savings or any other incidental or consequential damage resulting from the use, misuse of, originality to use this product. Vendor will not be liable for any claim made by any other related party. Return authorization must be obtained from the vendor before returned merchandise will be accepted. Authorization can be obtained by calling or faxing the vendor and requesting a Return Merchandise Authorization (RMA) number. Returned goods should always be accompanied by a clear problem description.