User Manual

QEC-M-XXXT

DM&P Vortex86EX2 Processor

EtherCAT MDevice with multi-functionality.

7”/9”/15” Open Frame Panel PC with 4-wire Resistive Touch Screen

(Revision 3.1)

REVISION

DATE	VERSION	DESCRIPTION
2022/03/18	Version1.0A	New Release.
2022/06/04	Version1.0B	Edit EMC Description.
2023/08/08	Version2.0	Updated Product Specifications.
2023/10/26	Version2.1	Updated EtherCAT RJ45.
2023/10/31	Version2.2	Updated Arduino Pins and LCD Specifications.
2024/02/26	Version2.3	Add Getting Started.
2025/06/09	Version3.0	• Combine QEC-M-070T, QEC-M-090T, QEC-M-150T in one documentation. • Add Arduino Pins Information and usage.
2025/07/30	Version3.1	• Updated 15” LCD spec to IVO panel. • Removed DCM module for QEC-M-150T/TP. • Added IPS LCD Mura Note for 9” panel. • Deleted USB disk supported format. • Revised operating temp. ranges for different LCD sizes. • Corrected and emphasized J14 Arduino pin mapping. • Clarified EtherCAT port directions and FGND label.

COPYRIGHT

Ver.3.1 July, 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 7
1.1 Introduction 8
1.1.1 QEC EtherCAT MDevice Architecture 9
1.1.2 Hardware Platform 10
1.1.3 Dual-System Synchronization 11
1.1.4 Software Support 12
1.2 Specifications 13
1.2.1 QEC-M-070T 13
1.2.2 QEC-M-090T 15
1.2.3 QEC-M-150T 17
1.3 Dimension 20
1.3.1 QEC-M-070T 20
1.3.2 QEC-M-090T 21
1.3.3 QEC-M-150T 22
1.4 Inspection standard for TFT-LCD Panel 23
1.5 Ordering Information 27
1.5.1 Ordering Part Number 27
Ch. 2 Hardware System 28
2.1 General Technical Data 29
2.2 General Summary 30
2.2.1 USB 31
2.2.2 Arduino Pin Assignment 32
2.2.3 eMMC 39
2.2.4 USB Type C 40
2.2.5 EtherCAT Interface 41
2.2.6 Giga LAN 43
2.2.7 Power Connector 44
2.2.8 VGA Connector 45
2.3 Wiring to the Connector 46
2.3.1 Connecting the wire to the connector 46
2.3.2 Removing the wire from the connector 46
Ch. 3 Hardware Installation 47
3.1 Mounting Instructions 48
3.1.1 General Guidelines 48
3.1.2 Mounting Dimensions 49
Ch. 4 Getting Started (This chapter is available in multiple languages) 52
4.1 Package Contents 55
4.2 Hardware Configuration 56
4.3 Software/Development Environment 57
4.4 Connect to your PC and set up the environment 58

4.5 EtherCAT Communication 60
4.5.1 EtherCAT State Machine (ESM) Control 60
4.5.2 SubDevice Information 67
4.5.3 Process Data Objects (PDO) Functions 73
4.5.4 CANopen over EtherCAT (CoE) Functions 78
4.5.5 Cyclic Callback Functions 85
4.5.6 Distributed Clock (DC) Configuration Functions 92
4.5.7 86EVA, an EtherCAT Configuration Tool 97
4.5.8 Import ENI to QEC MDevice 111
4.6 Bootloader Menu Usage 118
4.6.1 Turn on Bootloader Menu 119
4.6.2 General Page 122
4.6.3 EtherCAT Page 126
4.6.4 Security Page 129
4.6.5 Exit Page 135
4.7 Arduino Pins Usage 136
4.7.1 Expansion Board: EC-TBV-ADAPT-KIT 138
4.7.2 GPIO 142
4.7.3 ADC 143
4.7.4 TX/RX 144
4.7.5 RS-485 145
4.7.6 CAN 149
4.7.7 SPI 151
4.7.8 I2C 153
4.7.9 MCM 154
4.8 USB Device Usage 156
4.8.1 Example 1: Save .txt in USB disk 157
4.8.2 Example2: Save .txt in EMMC storage 158
4.9 Giga LAN Configuration 159
4.9.1 Ethernet Communication 160
4.9.2 Modbus TCP Communication 166
4.10 HMI Design 168
4.10.1 Library Instruction 168
4.10.2 Using the Graphical HMI editor: 86HMI Editor 169
Ch. 5 Software Function 171
5.1 Software Description 172
5.2 EtherCAT Function List 173
5.2.1 EtherCAT MDevice 174
5.2.2 EtherCAT SubDevice 176
5.2.3 QEC-Series SubDevice 183
5.3 Additional Resources 185
Appendix 186

A1. About ENI Configuration in 86Duino IDE..... 187
Warranty..... 190

Ch. 1

General Information

1.1 Introduction

ICOP's QEC-M series (QEC-M-070T / QEC-M-090T / QEC-M-150T) are EtherCAT MDevice with Open-frame and Touch LCD system, designed for real-time, reliable, and synchronized industrial HMI control. Each model integrates a high-resolution touch-enabled TFT LCD (7", 9", or 15"), providing an embedded Open-frame form factor and powerful control core within a compact system.

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 series also supports Modbus, Ethernet TCP/IP, and CAN bus, providing a complete industrial automation solution.

Real-Time Precision for Motion and I/O Control
The QEC MDevice supports essential EtherCAT functions including PDO, CoE, FoE, Distributed Clocks (DC), and etc., ensuring flexible integration with third-party EtherCAT devices such as servo drives and digital I/O. With a minimum cycle time of 125 µs and jitter of less than 1 µs (via the 86Duino IDE), it is ideal for highly synchronized motion and I/O control applications.
Read More: EtherCAT MDevice Benchmark

Robust Storage, Reliable I/O, and Versatile Connectivity with Embedded form factor
Each model features a built-in 2GB SLC eMMC, ensuring stable OS operation and offering ample storage for executables, HMI graphics, and application data. Files can be deployed via the 86Duino IDE. The IDE also integrates the LVGL graphics library, allowing developers to create modern and interactive touchscreen HMIs directly on the QEC device.

In addition to EtherCAT control, QEC-M models monitor system temperature, voltage, and current—providing useful data for carbon footprint analysis and system lifespan estimation. The Open-frame design (dimensions vary by model) supports flexible integration and customization for industrial use. Standard operating temperature is -20°C to +70°C, with an extended option from -40°C to +85°C.

Each QEC-M unit features dual EtherCAT ports (for redundancy), one Giga LAN port, three USB ports, a USB debug port (for upload/debug), and full Arduino-compatible pins including PWM, SPI, I^2C, and CAN, which all can accessible control via off-the-shelf Library.

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.

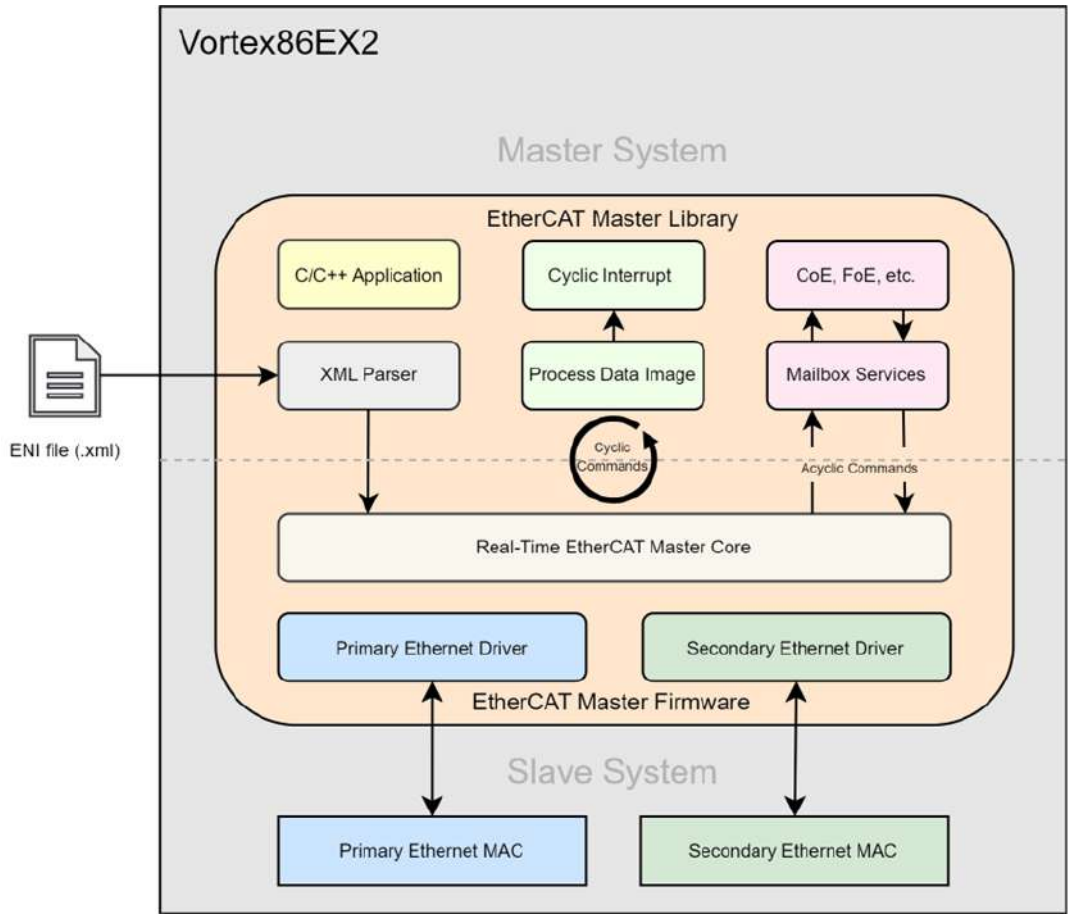
1.1.2 Hardware Platform

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

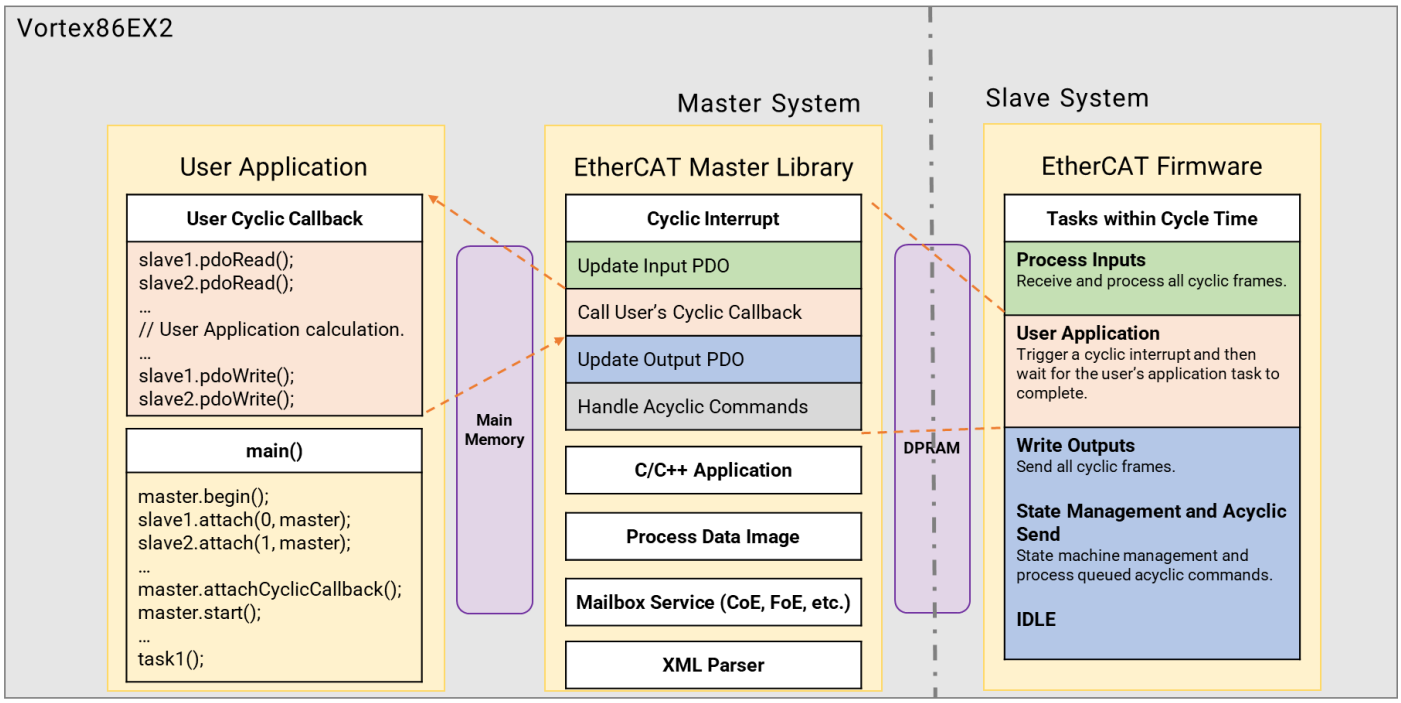
The primary focus of this section is the synchronization of dual-system PDO data. As illustrated in the diagram below, the User Application and EtherCAT MDevice Library blocks run on the Master System, while the Real-Time EtherCAT MDevice Core runs on the Slave System.

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.

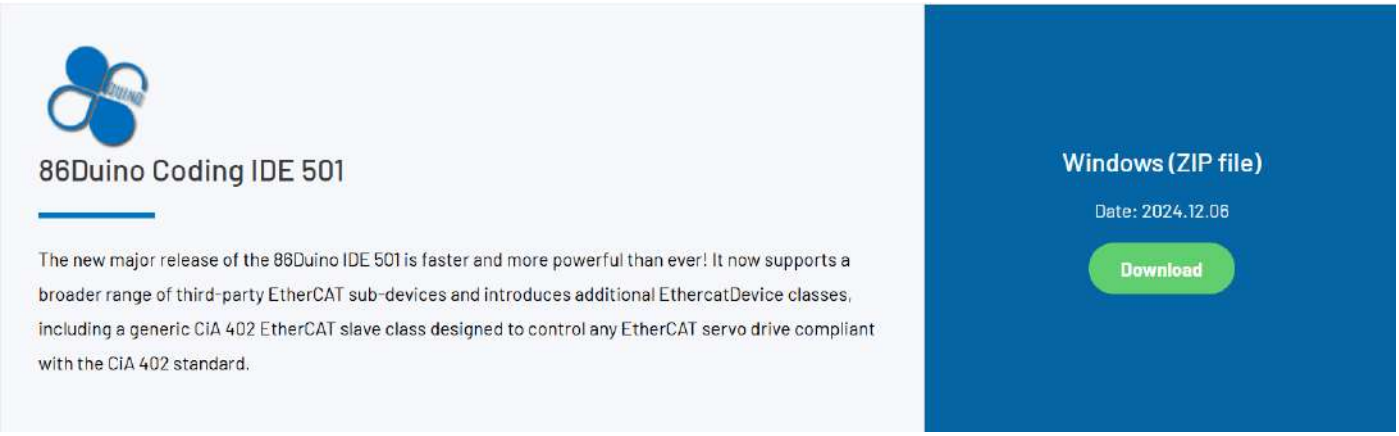
When the EtherCAT MDevice Core reaches the **Write Outputs** stage, it copies Output PDO data from DPRAM to the Ethernet Driver's DMA and sends frames.

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

1.2.1 QEC-M-070T

CPU BOARD SPECIFICATIONS

CPU	DM&P Vortex86EX2 Processor, Master 533MHz/Slave 400MHz
Memory	512MB/1GB DDRIII Onboard
Storage	32MB SPI Flash /2GB SLC eMMC
LAN	1Gbps Ethernet RJ45 x1 10/100Mbps Ethernet RJ45 x2 for EtherCAT
I/O Connector	2.54mm 2-pin header for Power Connector 1.25mm 4-pin header for EXT I2C TFT Driver 1.25mm 4-pin wafer for Line-Out Power DC Input/Output Connector x1 USB (Type-C) x1 (Upload/Debug only) VGA Connector (10-pin) x1 USB 2.0 Host x3 RJ45 x3
Arduino Compatible Connector	2.54mm 10-pin female header for I2C0, MCM, GPIO 2.54mm 8-pin female header for MCM, GPIO, COM1 (TTL) 2.54mm 8-pin female header for Power source 2.54mm 6-pin female header for ADC/GPIO 2.54mm 6-pin female header for GPIO, VCC and GND 2.54mm 6-pin female header for CAN0 and CAN1 bus 2.54mm 10-pin header for SPI, RESET- 2.54mm 10-pin header for SPI, RESET-, RS485
Protocol	EtherCAT, Modbus, Ethernet, CAN bus, etc.
Ethernet Standard	IEEE 802.3
Control Cycle Time	125 μs (min.)
Software Support	86Duino Coding IDE 500+ (The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software)

*MCM signal is equivalent to Arduino's PWM signal.

MECHANICAL & ENVIRONMENT

Power Connector	6-pin Power Input /Output
Power Requirement	+19 to +50VDC Power Input (Typ. +24VDC)
Power Consumption	8 Watt
Operating Temp.	-20 to +70°C/-30 to +85°C (Option)
Storage Temp.	-30 ~ +85°C
Operating Humidity	0% ~ 90% Relative Humidity, Non-Condensing
Dimension	186 x 121.05 x 31.05 mm
Weight	520 g
Internal Monitoring	Temperature, Voltage, Current, Start-up time

LCD SPECIFICATIONS

Display Type	7” WVGA TFT LCD
Backlight Unit	LED
Display Resolution	800(W) x 480(H)
Brightness (cd/m^2)	400 nits
Contrast Ratio	500: 1
Display Color	16.7M
Pixel Pitch (mm)	0.0642 (W) × 0.1790 (H) mm
Viewing Angle	Vertical 120º, Horizontal 140º
Backlight Lifetime	20,000 hrs

TOUCHSCREEN

Type	Analog Resistive
Resolution	Continuous
Transmittance	80%
Controller	PS/2 interface
Durability	1 million

1.2.2 QEC-M-090T

CPU BOARD SPECIFICATIONS

CPU	DM&P Vortex86EX2 Processor, Master 533MHz/Slave 400MHz
Memory	512MB/1GB DDRIII Onboard
Storage	32MB SPI Flash /2GB SLC eMMC
LAN	1Gbps Ethernet RJ45 x1 10/100Mbps Ethernet RJ45 x2 for EtherCAT
I/O Connector	2.54mm 2-pin header for Power Connector 1.25mm 4-pin header for EXT I2C TFT Driver 1.25mm 4-pin wafer for Line-Out Power DC Input/Output Connector x1 USB (Type-C) x1 (Upload/Debug only) VGA Connector (10-pin) x1 USB 2.0 Host x3 RJ45 x3
Arduino Compatible Connector	2.54mm 10-pin female header for I2C0, MCM, GPIO 2.54mm 8-pin female header for MCM, GPIO, COM1 (TTL) 2.54mm 8-pin female header for Power source 2.54mm 6-pin female header for ADC/GPIO 2.54mm 6-pin female header for GPIO, VCC and GND 2.54mm 6-pin female header for CAN0 and CAN1 bus 2.54mm 10-pin header for SPI, RESET- 2.54mm 10-pin header for SPI, RESET-, RS485
Protocol	EtherCAT, Modbus, Ethernet, CAN bus, etc.
Ethernet Standard	IEEE 802.3
Control Cycle Time	125 μs (min.)
Software Support	86Duino Coding IDE 500+ (The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software)

*MCM signal is equivalent to Arduino's PWM signal.

MECHANICAL & ENVIRONMENT

Power Connector	6-pin Power Input /Output
Power Requirement	+19 to +50VDC Power Input (Typ. +24VDC)
Power Consumption	8 Watt
Operating Temp.	-20 to +70°C
Storage Temp.	-30 ~ +85°C
Operating Humidity	0% ~ 90% Relative Humidity, Non-Condensing
Dimension	245 x 152.2 x 32.55mm
Weight	1.07Kg
Internal Monitoring	Temperature, Voltage, Current, Start-up time

LCD SPECIFICATIONS

Display Type	9” WVGA TFT LCD
Backlight Unit	LED
Display Resolution	800(W) x 480(H)
Contrast Ratio	500: 1
Display Color	16.7M
Active Area	198 (W) x 111.696 (H) mm
Viewing Angle	Vertical 120^{o}, Horizontal 140^{o}
Brightness (cd/m^{2})	300 nits
Backlight Lifetime	20,000 hrs

TOUCHSCREEN

Type	Analog Resistive
Resolution	Continuous
Transmittance	80%
Controller	PS/2 interface
Durability	1 million

1.2.3 QEC-M-150T

CPU BOARD SPECIFICATIONS

CPU	DM&P Vortex86EX2 Processor, Master 533MHz/Slave 400MHz
Memory	512MB/1GB DDRIII Onboard
Storage	32MB SPI Flash /2GB SLC eMMC
LAN	1Gbps Ethernet RJ45 x1 10/100Mbps Ethernet RJ45 x2 for EtherCAT
I/O Connector	2.54mm 2-pin header for Power Connector 1.25mm 4-pin header for EXT I2C TFT Driver 1.25mm 4-pin wafer for Line-Out Power DC Input/Output Connector x1 USB (Type-C) x1 (Upload/Debug only) VGA Connector (10-pin) x1 USB 2.0 Host x3 RJ45 x3
Arduino Compatible Connector	2.54mm 10-pin female header for I2C0, MCM, GPIO 2.54mm 8-pin female header for MCM, GPIO, COM1 (TTL) 2.54mm 8-pin female header for Power source 2.54mm 6-pin female header for ADC/GPIO 2.54mm 6-pin female header for GPIO, VCC and GND 2.54mm 6-pin female header for CAN0 and CAN1 bus 2.54mm 10-pin header for SPI, RESET- 2.54mm 10-pin header for SPI, RESET-, RS485
Protocol	EtherCAT, Modbus, Ethernet, CAN bus, etc.
Ethernet Standard	IEEE 802.3
Control Cycle Time	125 μs (min.)
Software Support	86Duino Coding IDE 500+ (The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software)

*MCM signal is equivalent to Arduino's PWM signal.

MECHANICAL & ENVIRONMENT

Power Connector	6-pin Power Input /Output
Power Requirement	+19 to +50VDC Power Input (Typ. +24VDC)
Power Consumption	10 Watt
LCD Display Power	DCM-12V: Input 15~36Vdc and Output 12V@Max.2.5A
Operating Temp.	-20 to +70°C
Storage Temp.	-30 ~ +85°C
Operating Humidity	0% ~ 90% Relative Humidity, Non-Condensing
Dimension	369 x 278 x 49.25 mm
Weight	2.3 Kg
Internal Monitoring	Temperature, Voltage, Current, Start-up time

LCD SPECIFICATIONS

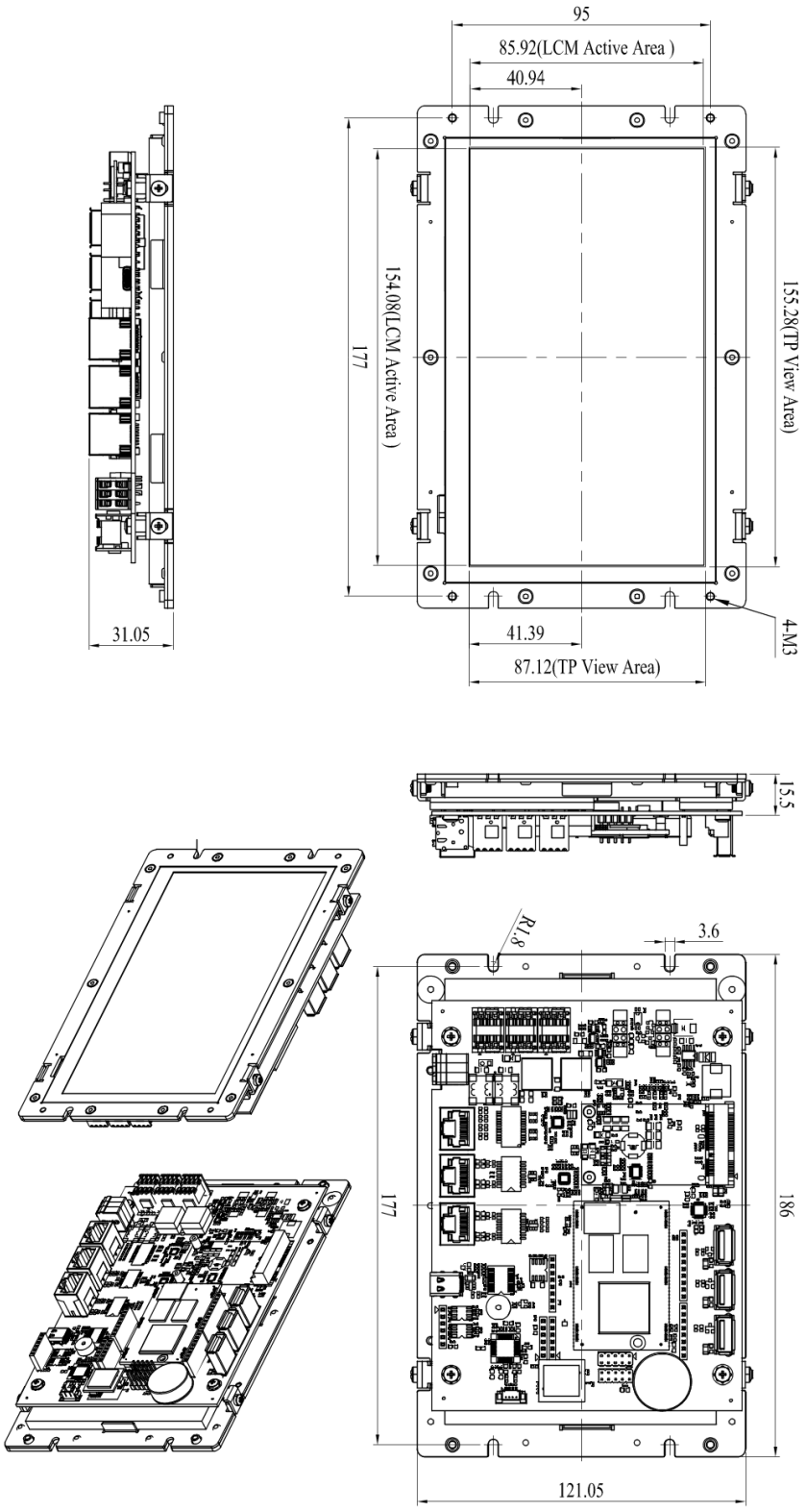
Display Type	15” TFT LCD
Light Bar Unit	LED, Non-Replaceable
Active Area	304.13 (H) x 228.10 (V)
Pixels	1024 (H) x 768 (V)
Pixel Pitch (mm)	0.297 x 0.297 mm
Pixel Arrangement	R.G.B. Vertical Stripe
Display Mode	TN, Normally Black
Nominal Input	3.3 V (Typ.)
Power Consumption	14.1 W (Max.)
Electrical Interface	1 channel LVDS
Contrast Ratio	800: 1
Brightness (cd/m^2)	400 nits
Support Color	16.7M colors
Viewing Angle	Vertical 170°, Horizontal 170°
Backlight Lifetime	30,000 hrs

TOUCHSCREEN

Type	Analog Resistive
Resolution	Continuous
Transmittance	80%
Controller	PS/2 interface
Durability	1 million

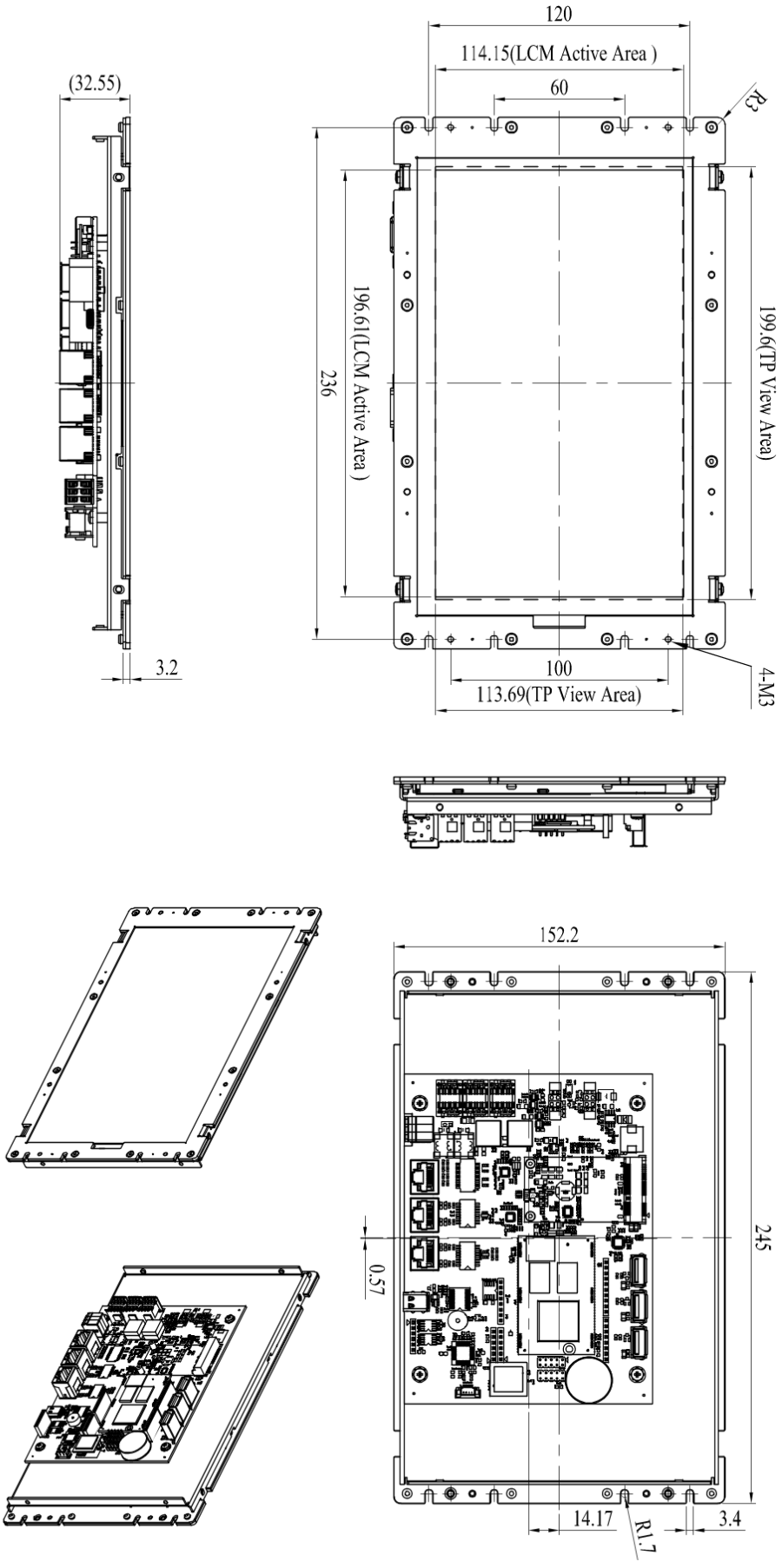
1.3 Dimension

1.3.1 QEC-M-070T



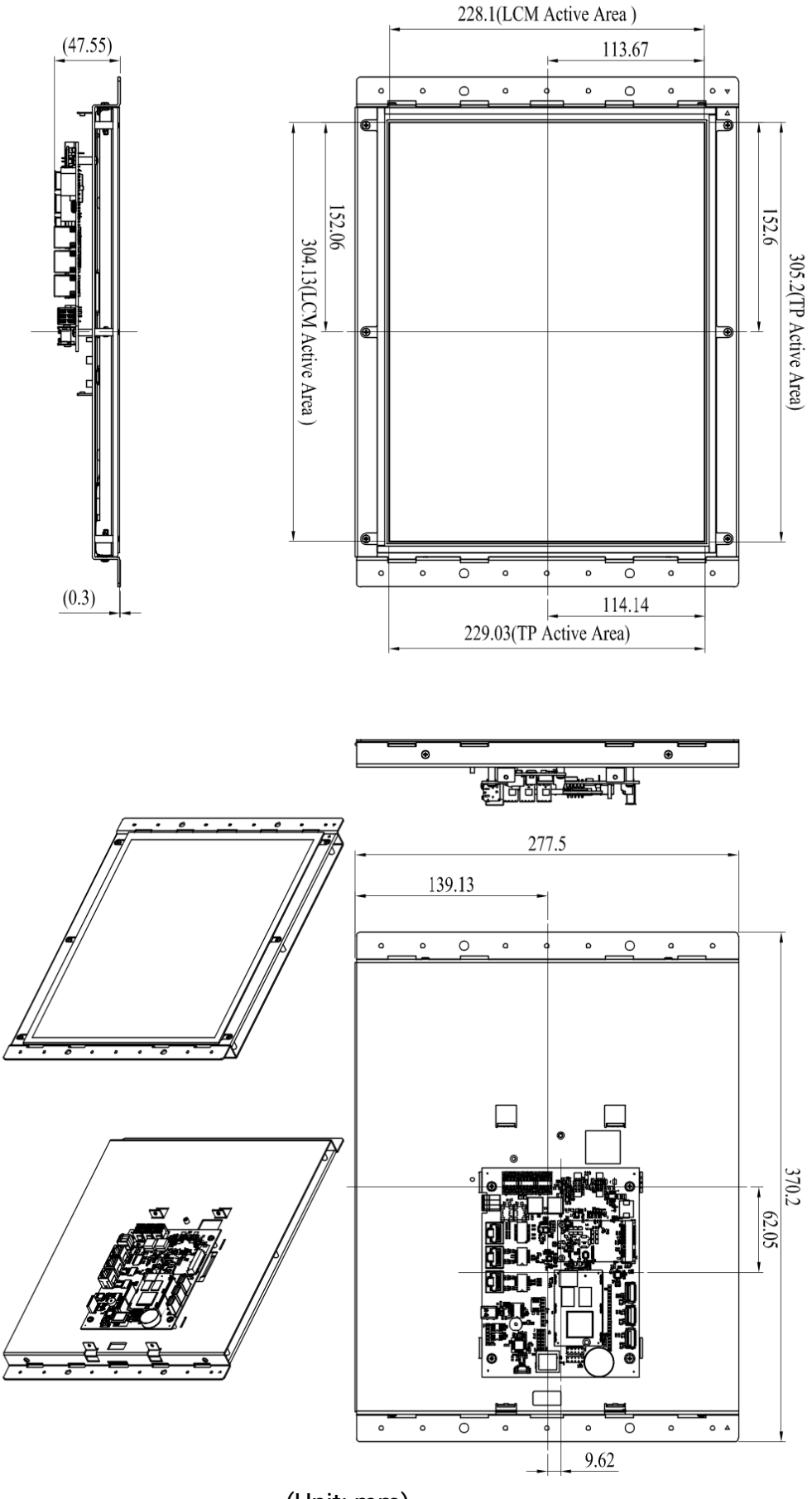
(Unit: mm)

1.3.2 QEC-M-090T



(Unit: mm)

1.3.3 QEC-M-150T



(Unit: mm)

1.4 Inspection standard for TFT-LCD Panel

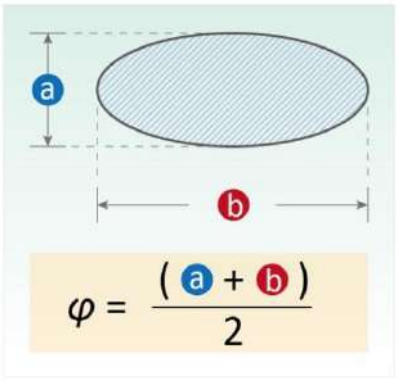
<table>
<tr><th colspan="3">DEFECT TYPE</th><th colspan="7">LIMIT</th><th>Note</th></tr>
<tr><td rowspan="11">VISUAL DEFECT</td><td rowspan="11">INTERNAL</td><td rowspan="3">SPOT</td><td colspan="4">φ＜0.15mm</td><td colspan="3">Ignore</td><td rowspan="3">Note1</td></tr>
<tr><td colspan="4">0.15mm≦φ≦0.5mm</td><td colspan="3">N≦4</td></tr>
<tr><td colspan="4">0.5mm＜φ</td><td colspan="3">N=0</td></tr>
<tr><td rowspan="2">FIBER</td><td colspan="4">0.03mm<W≦0.1mm, L≦5mm</td><td colspan="3">N≦3</td><td rowspan="2">Note1</td></tr>
<tr><td colspan="4">1.0mm＜W, 1.5mm＜L</td><td colspan="3">N=0</td></tr>
<tr><td rowspan="3">POLARIZER BUBBLE</td><td colspan="4">φ＜0.15mm</td><td colspan="3">Ignore</td><td rowspan="3">Note1</td></tr>
<tr><td colspan="4">0.15mm≦φ≦0.5mm</td><td colspan="3">N≦2</td></tr>
<tr><td colspan="4">0.5mm＜φ</td><td colspan="3">N=0</td></tr>
<tr><td>Mura</td><td colspan="7">It’ OK if mura is slight visible through 6%ND filter</td><td></td></tr>
<tr><td rowspan="8">ELECTRICAL DEFECT</td><td colspan="2" rowspan="2">BRIGHT DOT</td><td colspan="3">A Grade</td><td colspan="3">B Grade</td><td></td></tr>
<tr><td>C Area</td><td>O Area</td><td>Total</td><td>C Area</td><td>O Area</td><td>Total</td><td>Note3</td></tr>
<tr><td colspan="2">DARK DOT</td><td>N≦0</td><td>N≦2</td><td>N≦2</td><td>N≦2</td><td>N≦3</td><td>N≦5</td><td>Note2</td></tr>
<tr><td colspan="2">TOTAL DOT</td><td>N≦2</td><td>N≦3</td><td>N≦3</td><td>N≦3</td><td>N≦5</td><td>N≦8</td><td></td></tr>
<tr><td colspan="2">TWO ADJACENT DOT</td><td colspan="3">N≦4</td><td>N≦5</td><td>N≦6</td><td>N≦8</td><td>Note2</td></tr>
<tr><td colspan="2">THREE OR MORE ADJACENT DOT</td><td>N≦0</td><td>N≦1 pair</td><td>N≦1 pair</td><td>N≦1 pair</td><td>N≦1 pair</td><td>N≦1 pair</td><td>Note4</td></tr>
<tr><td colspan="2">LINE DEFECT</td><td colspan="7">NOT ALLOWED</td><td></td></tr>
</table>

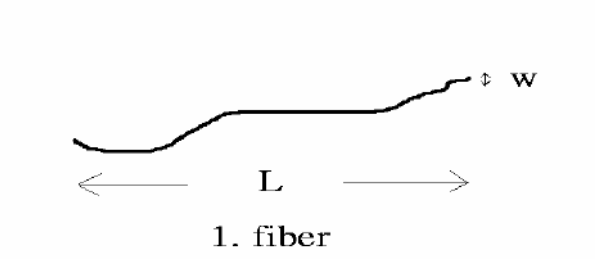
(1) One pixel consists of 3 sub-pixels, including R, G, and B dot. (Sub-pixel = Dot)

(2) Little bright Dot acceptable under 6% ND-Filter.

(3) If require G0 grand (Total dot N≦0), please contact region sales.

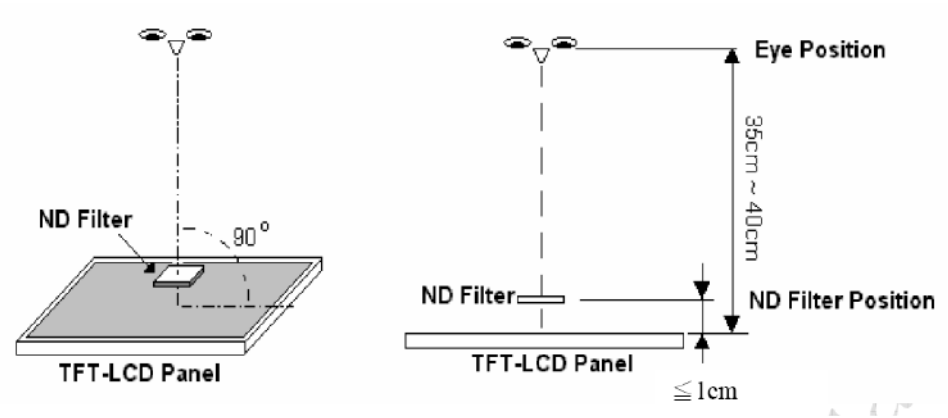
[Note 1] W: Width[mm]; L: Length[mm]; N: Number; φ: Average Diameter.



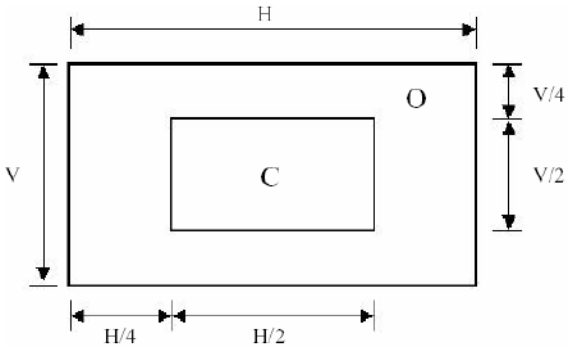


(a) White / Black Spot (b) Polarizer Bubble

[Note 2] Bright dot is defined through 6% transmission ND Filter as following.

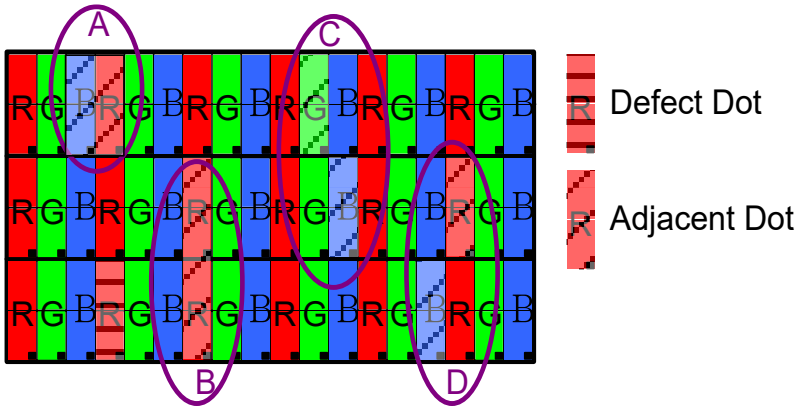


[Note 3] Display area



C Area: Center of display area **O Area**: Outer of display area

[Note 4] Judge the defect dot and the adjacent dot as following. Allow below (as A, B, C and D status) adjacent defect dots, including bright and dark adjacent dot. And they will be counted 2 defect dots in total quantity.



(1) The defects that are not defined above and considered to be problem shall be reviewed and discussed by both parties.

Defects on the Black Matrix, out of Display area, are not considered as a defect or counted.

[Note 5] According to the technical information from LCD manufacturer, the image retention may happen on LCD display if the static image is kept for a period of time without any change. ICOP will suggest customers not to have static image on LCD for over 4 hours without any image movement and also enable screensaver to avoid image sticking issue if LCD displays need to be kept on for a long time.

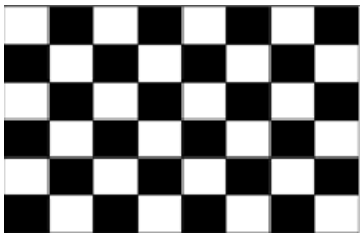
Some Image retention issue will disappear when LCD display is turned off for a period of time, but some image retention may be not reversible when LCD encounters screen burn.

The following is LCD manufacturer's test result for customers' reference.

TEST ITEMS	CONDITIONS	NOTE
High Temperature Operation	70℃ ;240hrs	
High Temperature Storage	80℃ ; 240hrs	
High Temperature High Humidity Operation	60℃ ; 90%RH ;240hrs	No condensation
Low Temperature Operation	-20℃ ; 240hrs	Backlight unit always turn on
Low Temperature Storage	-30℃ ; 240hrs	
Thermal Shock	−30℃(0.5hr) ~ 80℃(0.5hr) ; 200 Cycles	
Image Sticking	25℃ ; 4hrs	**Note 5-1**
MTBF	20,000Hrs	

Note 5-1

1. Condition of Image Sticking test：25 ℃± 2 ℃.
2. Operation with test pattern sustained for 4 hrs, then change to gray pattern immediately.
3. After 5 mins, the mura must be disappeared completely.



1.5 Ordering Information

Type	LCD size	(Below is the customization function, the unfilled fields do not need to be filled in; if the customer does not require, it will be directly shipped standard material number, such as QEC-M-070T)							
		PoE	-	Feature		-	Wide Temp.	-	Coating
				Memory	Storage				
QEC-M	XXXT	X		X	X		X		X

1. Type: Code 1~4

M: EtherCAT MDevice

2. LCD size: Code 5~8

070T: 7-inch TFT LCD with Restive touchscreen

090T: 9-inch TFT LCD with Restive touchscreen

150T: 15-inch TFT LCD with Restive touchscreen

3. PoE: Code 9

P: RJ45 PoE Device, Red Plastic Housing

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

(Standard: None)

QEC-M-XXXT X – XX – X - X

4. Feature: Code 10~11

X (Memory): G: 1G DDRIII / M: 512M DDRIII (Standard: 512MB)

X (Storage): 1, 2, 4: eMMC size (Standard: 2G)

5. Wide Temp.: Code 12

X: -30 to +85°C for QEC-M-070T(P) option only / R: -20 to +70°C (Standard: -20 to +70°C)

6. Coating: Code 13

C: Yes / N: Normal

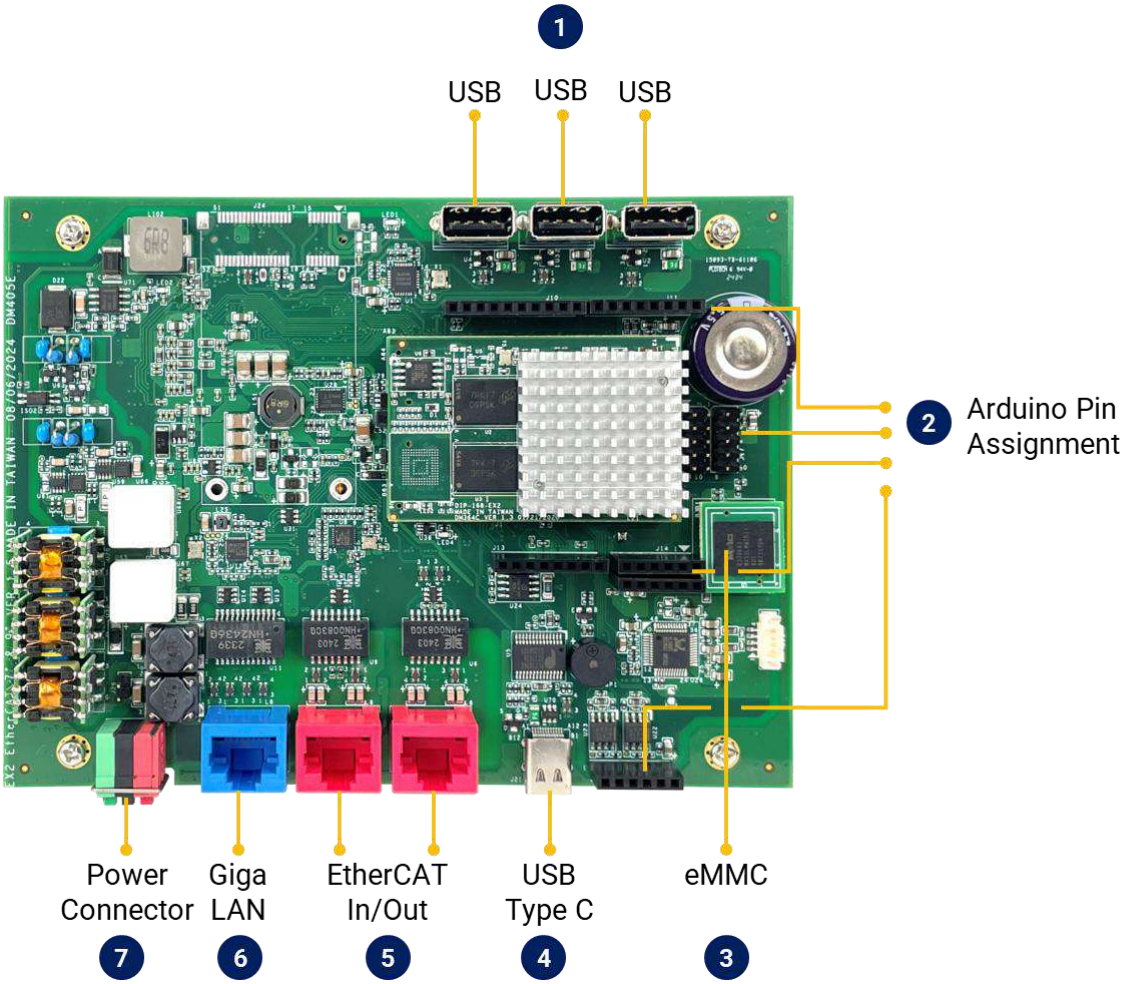
1.5.1 Ordering Part Number

- **QEC-M-070T**: EtherCAT MDevice System with 7-inch LCD
- **QEC-M-070TP**: EtherCAT MDevice System with 7-inch LCD/PoE
- **QEC-M-090T:** EtherCAT MDevice System with 9-inch LCD
- **QEC-M-090TP:** EtherCAT MDevice System with 9-inch LCD/PoE
- **QEC-M-150T:** EtherCAT MDevice System with 15-inch LCD
- **QEC-M-150TP:** EtherCAT MDevice System with 15-inch LCD/PoE

Ch. 2

Hardware System

2.1 General Technical Data

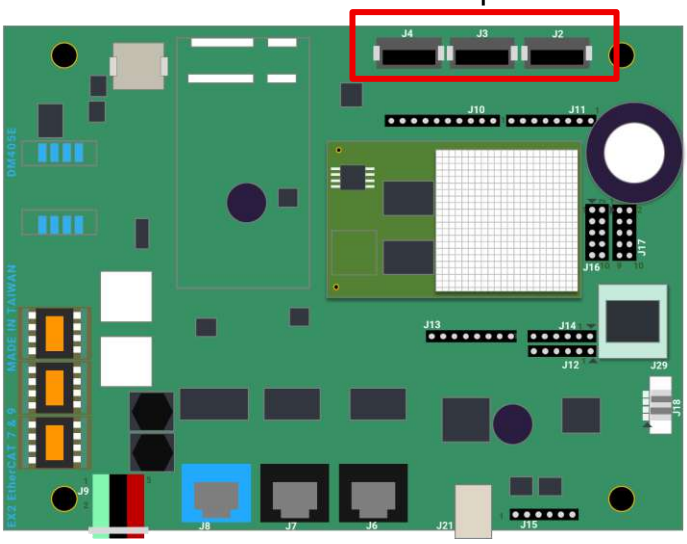


2.2 General Summary

No.	Description		Type Narrative	Pin #
1	USB		USB 2.0	-
2	Arduino Pin Assignment		Arduino Pins functions, including I^2C, MCM, GPIO, RS232/485, CAN, etc.	64-pin
3	eMMC		2GB eMMC module	-
4	USB Type-C (Debug and Upload port)		USB Type-C	-
5	EtherCAT Interface	IN	External RJ45 Connector (Gold finger)	8-pin
		OUT		8-pin
6	Giga LAN		External RJ45 Connector (Gold finger)	8-pin
7	Power Connector		Terminal Block Interface	6-pin
8	VGA Connector		1.25mm 10 pin VGA	10-pin

2.2.1 USB

There are three Standard USB 2.0 ports with hot-plug support as shown in the diagram.



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.

***Notes on USB Disk Usage:**

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

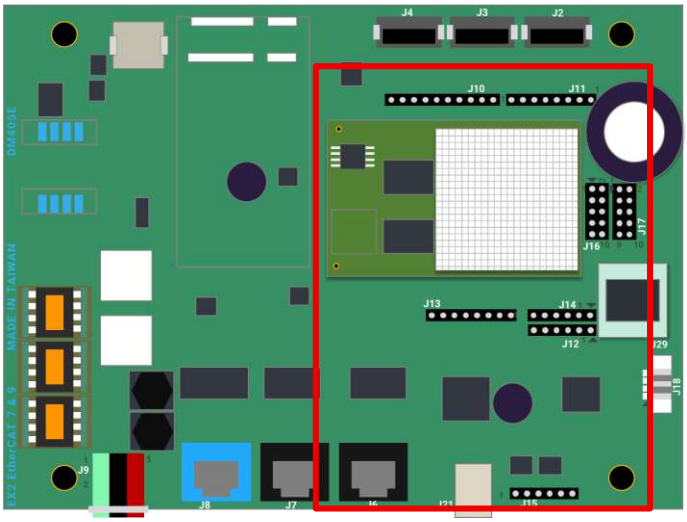
For more details on using USB storage devices, please refer to the Ch 4.8 USB Device Usage Section.

***Notes on Keyboard/Mouse Usage:**

Due to the QEC MDevice Open-frame series supports LCD and Touch. If you would like to use keyboard and mouse functions, please let us know before shipping.

2.2.2 Arduino Pin Assignment

We have kept all the Arduino pins on the QEC MDevice Open-frame Series as shown in the diagram.



There are two types of pins: **Arduino standard pins** and **86duino-only pins**.

Users can easily control these pins via the **86Duino IDE software**. To drive Arduino pins, you can refer to the Libraries and Language.

***Notes on Arduino Pins Usage:**
When using Arduino-style functions such as `digitalWrite()` or `pinMode()` on the QEC system, users must follow the 86Duino pin mapping.
For example, if you want to control the GPIO connected to J11 Pin 3 (GP00), you should use pin number 2 in your code, like this: `digitalWrite(2, HIGH);`

Refer to the 86Duino pin configuration table below to ensure the correct mapping between physical pin locations and software pin numbers.

Table: 86Duino pin configuration

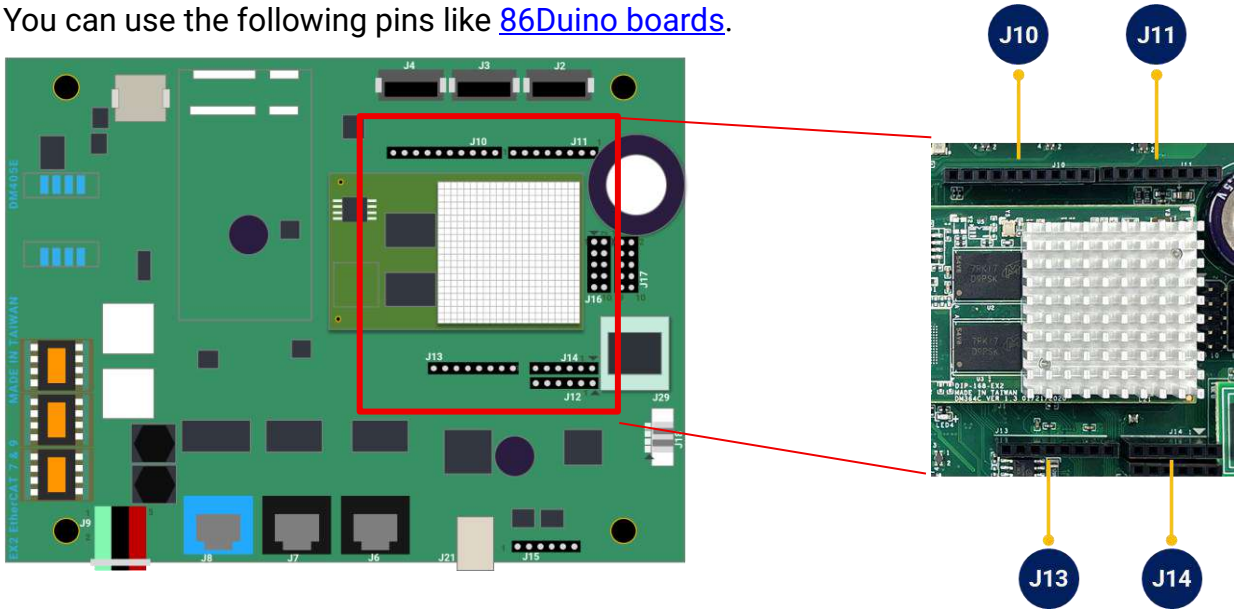
Connector	Pins	Signal	86Duino Pin Configuration
J11	1	RXD1#	0 (Serial1)
	2	TXD1#	1 (Serial1)
	3	GP00	2
	4	MCM-3	3
	5	GP02	4
	6	MCM-5	5
	7	MCM-6	6
	8	GP05	7
J10	1	GP30	8

J10	2	MCM-9	9
	3	MCM-10	10
	4	MCM-11	11
	5	GP31	12
	6	MCM-13	13
	7	GND	-
	8	-	-
	9	I2C0_SDA	(Wire)
	10	I2C0_SCL	(Wire)
J14	**6**	GP40ADC	14 (A0)
	5	GP41ADC	15 (A1)
	4	GP42ADC	16 (A2)
	3	GP43ADC	17 (A3)
	2	GP56ADC	18 (A4)
	1	GP57ADC	19 (A5)
J12	1	-	-
	2	GP35	23
	3	GP36	24
	4	GP37	25
	5	GND	-
	6	VCC	-
J15	1	CAN1_L	(CAN1)
	2	CAN1_H	(CAN1)
	3	GND	-
	4	CAN0_L	(CAN)
	5	CAN0_H	(CAN)
	6	VCC3	-
J16	1	SPI0_DI	(SPI)
	2	VCC	-
	3	SPI0_CLK	(SPI)
	4	SPI0_DO	(SPI)
	5	RESET-	-
	6	GND	-
	7	SPI0_CS	(SPI)
	8	-	-
	9	-	-
	10	-	-
J17	1	SPI1_DI	(SPI1)
	2	VCC	-
	3	SPI1_CLK	(SPI1)
	4	SPI1_DO	(SPI1)

J17	5	RESET-	-
	6	GND	-
	7	SPI1_CS	(SPI1)
	8	-	-
	9	RS485+	(Serial485 / SerialCOM)
	10	RS485-	(Serial485 / SerialCOM)

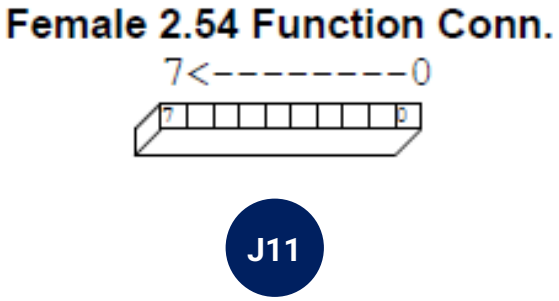
2.2.2.1 Arduino Standard Pins

You can use the following pins like 86Duino boards.



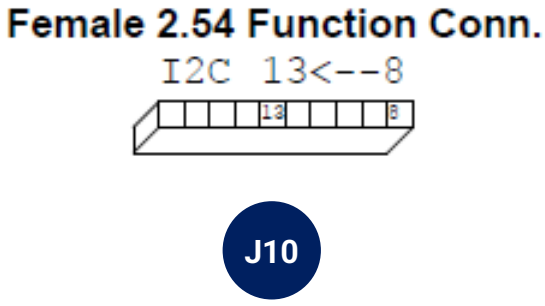
J11: MCM, GPIO, COM1(TTL)

Pin#	Signal Name	86Duino Pin
1	RXD1#	0 (Serial1)
2	TXD1#	1 (Serial1)
3	GP00	2
4	MCM-3	3
5	GP02	4
6	MCM-5	5
7	MCM-6	6
8	GP05	7



J10: I2C0, MCM, GPIO

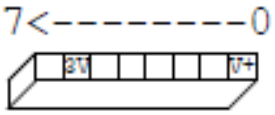
Pin#	Signal Name	86Duino Pin
1	GP30	8
2	MCM-9	9
3	MCM-10	10
4	MCM-11	11
5	GP31	12
6	MCM-13	13
7	GND	-
8	-	-
9	I2C0_SDA	(Wire)
10	I2C0_SCL	(Wire)



*MCM signal is equivalent to Arduino's PWM signal.

J13: Power source, RESET-

Pin#	Signal Name	86Duino Pin
1	VCC	-
2	GND	-
3	GND	-
4	VCC	-
5	VCC3	-
6	RESET-	-
7	VCC3	-
8	-	-

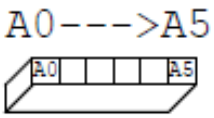


Female 2.54 Function Conn.

J13

J14: ADC/GPIO

Pin#	Signal Name	86Duino Pin
6	GP40ADC	14 (A0)
5	GP41ADC	15 (A1)
4	GP42ADC	16 (A2)
3	GP43ADC	17 (A3)
2	GP56ADC	18 (A4)
1	GP57ADC	19 (A5)

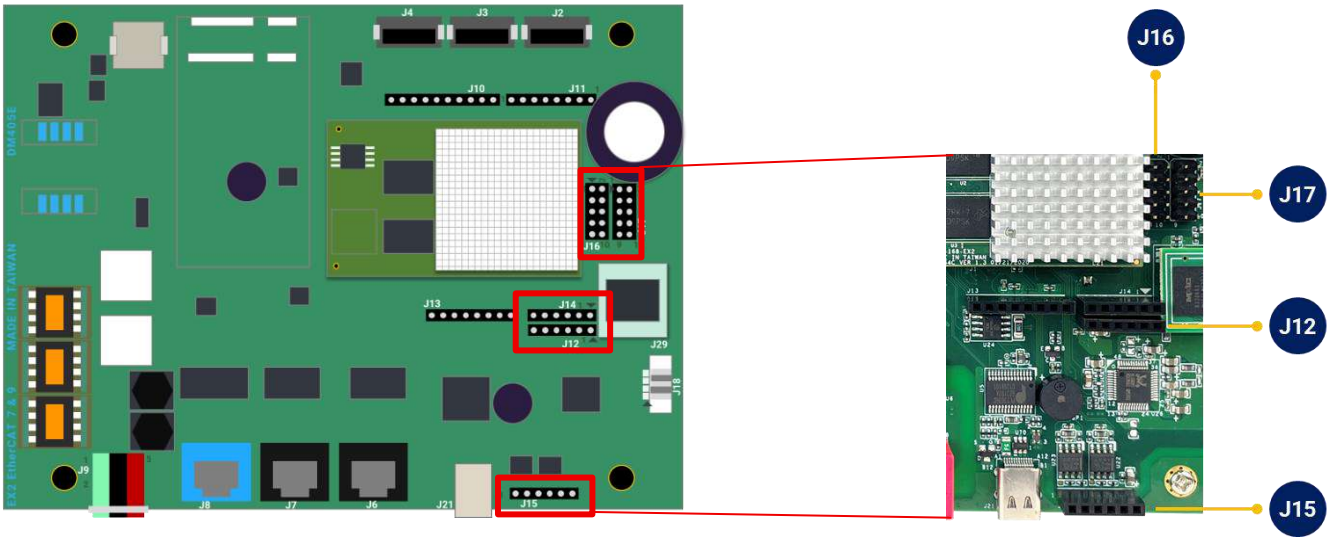


Female 2.54 Function Conn.

J14

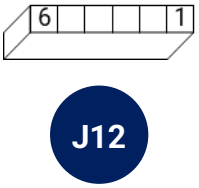
2.2.2.2 QEC 86Duino Only Pins

There are other pins for 86Duino Only Libraries on the QEC MDevice Open-frame Series.



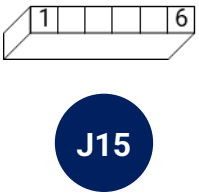
J12: GPIO, VCC, GND

Pin#	Signal Name	86Duino Pin
1	-	-
2	GP35	23
3	GP36	24
4	GP37	25
5	GND	-
6	VCC	-



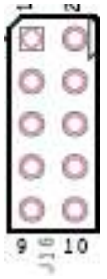
J15: CAN0 and CAN1 bus

Pin#	Signal Name	86Duino Pin
1	CAN1_L	(CAN1)
2	CAN1_H	(CAN1)
3	GND	-
4	CAN0_L	(CAN)
5	CAN0_H	(CAN)
6	VCC3	-



J16: SPI, RESET-

Pin#	Signal Name	86Duino Pin
1	SPI0_DI	(SPI)
2	VCC	-
3	SPI0_CLK	(SPI)
4	SPI0_DO	(SPI)
5	RESET-	-
6	GND	-
7	SPI0_CS	(SPI)
8	-	-
9	-	-
10	-	-



J16

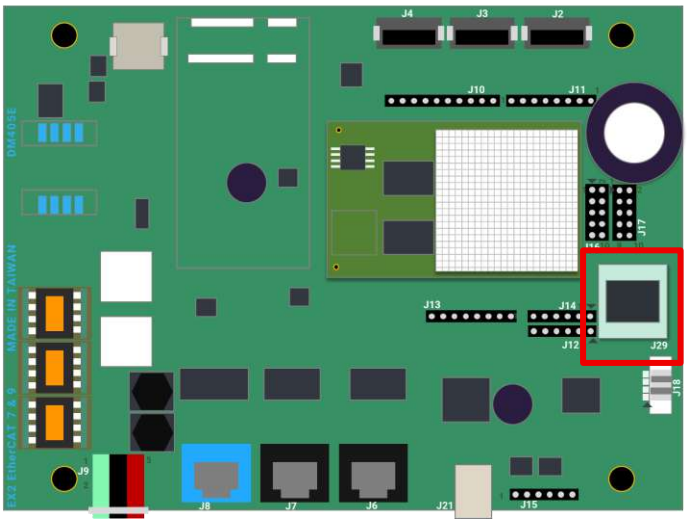
J17: SPI, RESET-, RS485+/-

Pin#	Signal Name	86Duino Pin
1	SPI1_DI	(SPI1)
2	VCC	-
3	SPI1_CLK	(SPI1)
4	SPI1_DO	(SPI1)
5	RESET-	-
6	GND	-
7	SPI1_CS	(SPI1)
8	-	-
9	RS485+	(Serial485 / SerialCOM)
10	RS485-	(Serial485 / SerialCOM)

J17

2.2.3 eMMC

There is a 2GB eMMC module is onboard by default, as shown in the diagram.



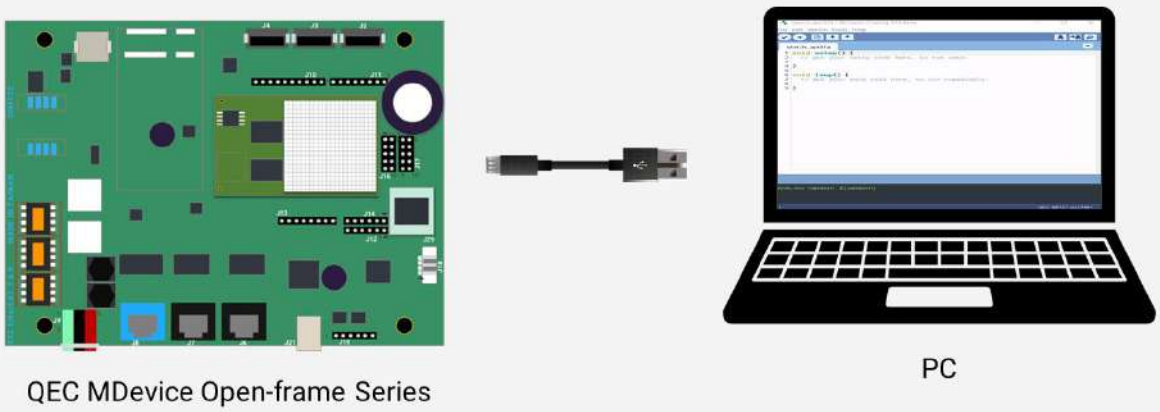
***Notes:** Your 86Duino executable will be uploaded to this eMMC.

To save data to this eMMC, you can refer to the SD library and set the following code:

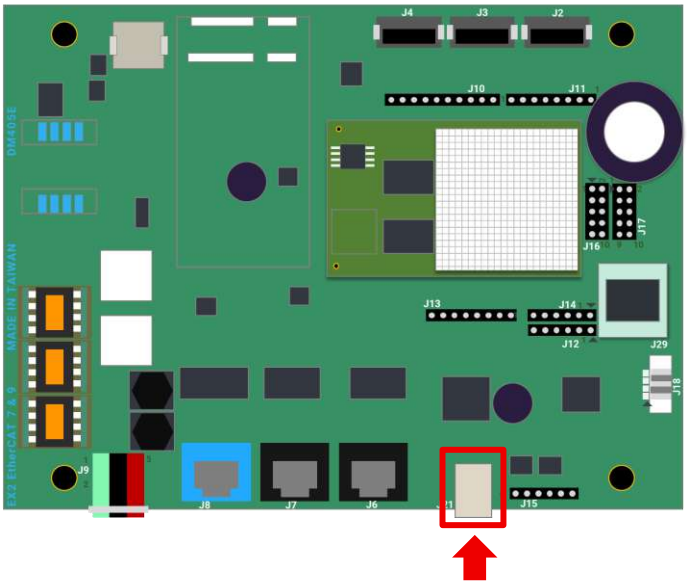
```
#include <SD.h>

void setup() {
  // ...
  SD.setBank(EMMCDISK);
}

void loop() {
  // …
}
```

2.2.4 USB Type C

The QEC MDevice Open-frame Series feature 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.



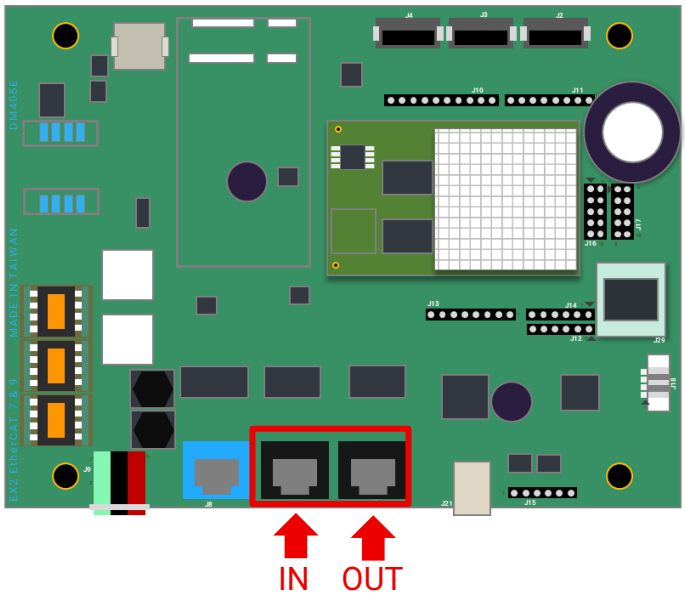
The USB Type-C port is located as shown in the diagram.



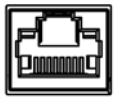
For instructions on programming and setting up the QEC MDevice Open-frame Series, please refer to Chapter 4: Getting Started.

2.2.5 EtherCAT Interface

RJ45 Connectors.



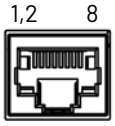
EC IN



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

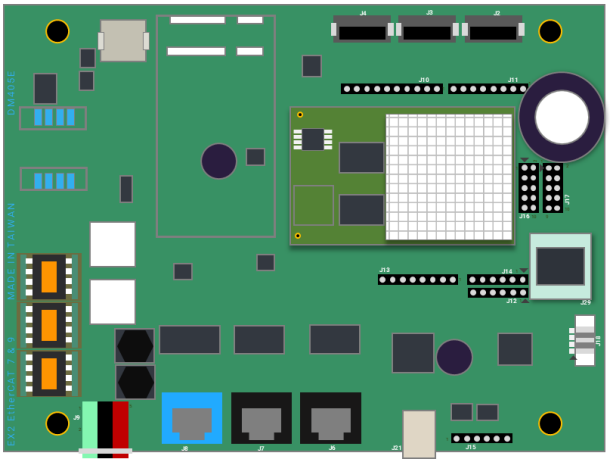
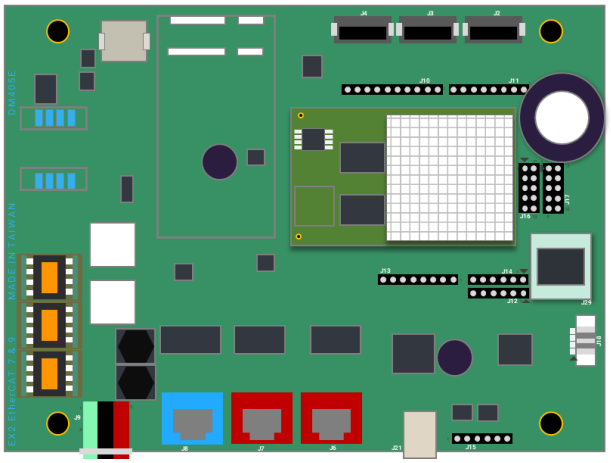


Pin #	Signal Name	Pin #	Signal Name
1	LAN2_TX+	2	LAN2_TX-
3	LAN2_RX+	4	VS+
5	VP+	6	LAN2_RX-
7	VS- (GND)	8	VP- (GND)

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

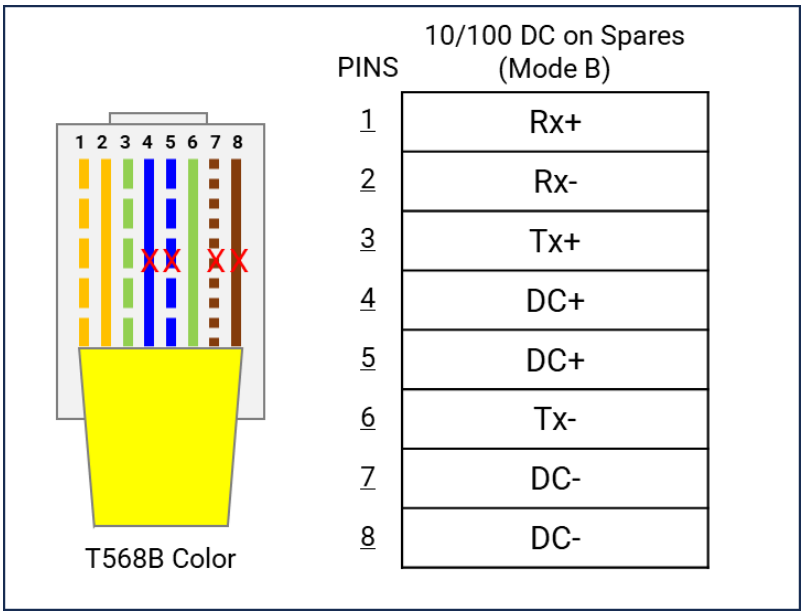
Note. QEC's PoE (Power over Ethernet)

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).



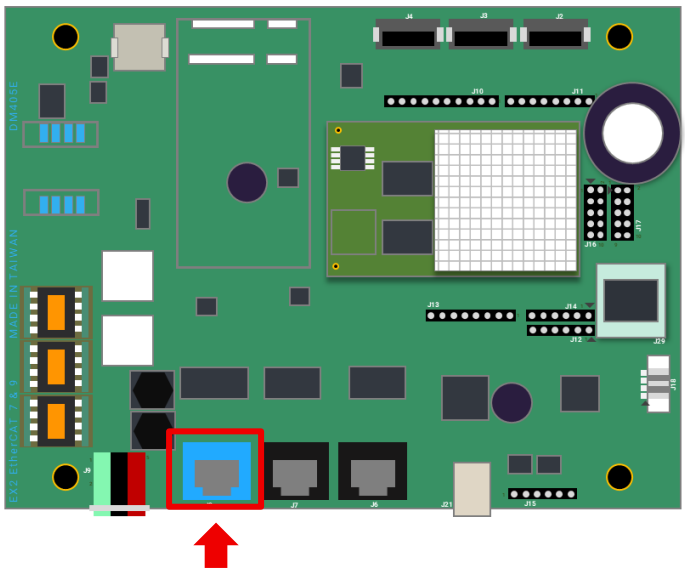
2. The PoE function of QEC is different and incompatible with EtherCAT P, and the PoE function of QEC is based on PoE Type B.

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

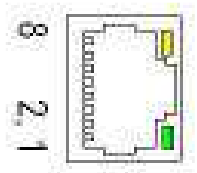
2.2.6 Giga LAN

The QEC MDevice Open-frame Series feature **one Giga LAN port** and is dedicated to external Ethernet communication for general network use.

The Giga LAN port is blue.



Pin Definitions:



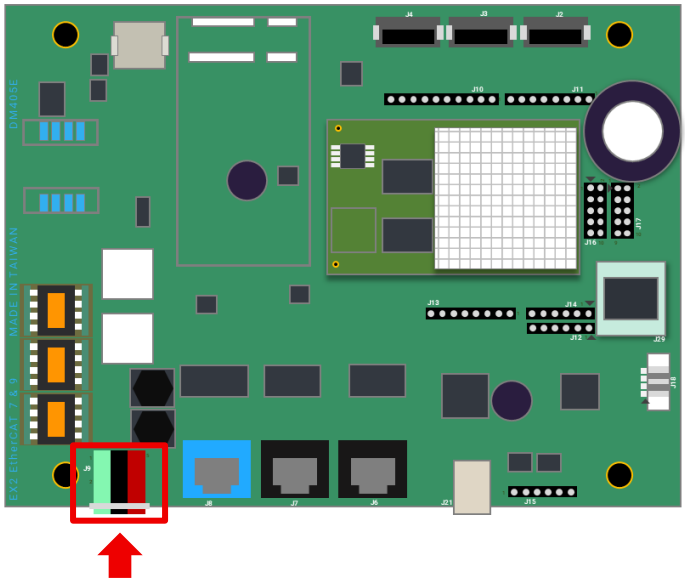
Pin #	Signal Name	Pin #	Signal Name
L1	GTX+	L2	GTX-
L3	GRX+	L4	GTXC+
L5	GTXC-	L6	GRX-
L7	GRXD+	L8	GRXD-

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.

2.2.7 Power Connector

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



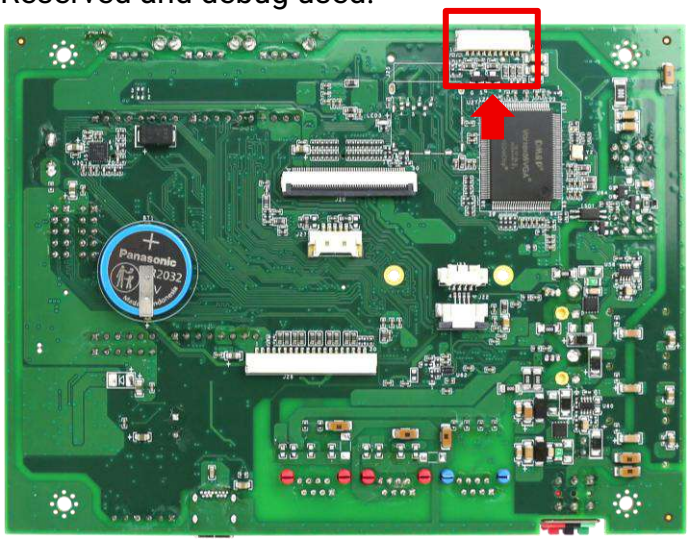
Vs for system power; Vp for peripheral power and backup power.

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

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

2.2.8 VGA Connector

Reserved and debug used.



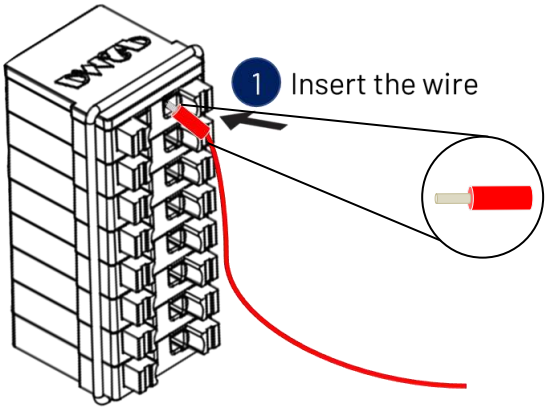
The pin configuration of a VGA Connector includes 10 pins, where each pin and its function are discussed below.

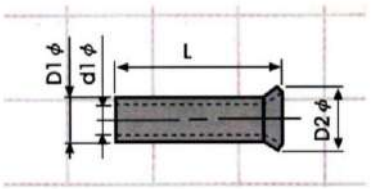
Pins Assignment:

Pin#	Signal Name
1	ROUT
2	GND
3	GOUT
4	GND
5	BOUT
6	GND
7	HSYNC_A
8	GND
9	VSYNC_A
10	GND

2.3 Wiring to the Connector

2.3.1 Connecting the wire to the connector

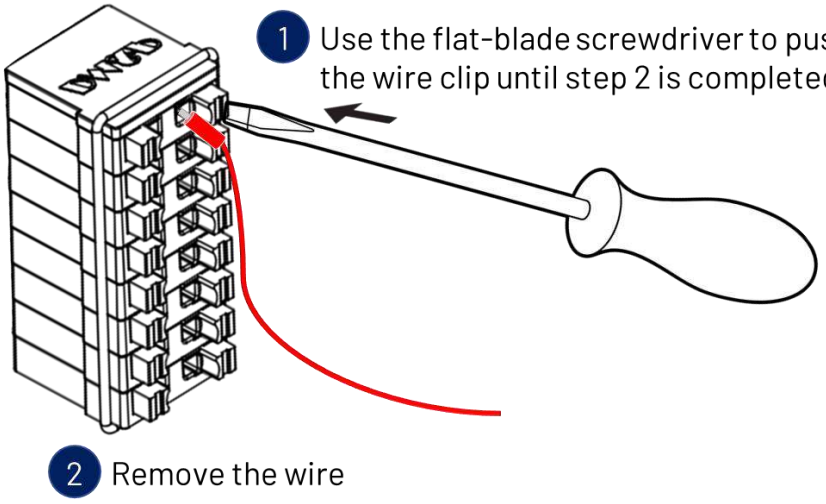




Insulated Terminals Dimensions (mm)

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



Ch. 3

Hardware Installation

3.1 Mounting Instructions

All QEC MDevice Open-frame series adopt an open-frame design and provide multiple mounting holes (**M3** and **M6**) to support panel mounting, rear bracket installation, or integration into custom enclosures.

3.1.1 General Guidelines

1. Use M3 screws for standard mounting holes (typically at corners).
2. Use M6 screws for reinforced mounting (available on QEC-M-090T and QEC-M-150T).
3. Ensure the panel surface is flat and grounded to prevent mechanical stress and ESD.
4. Leave at least 10 mm space behind the unit for airflow and cable bending radius.

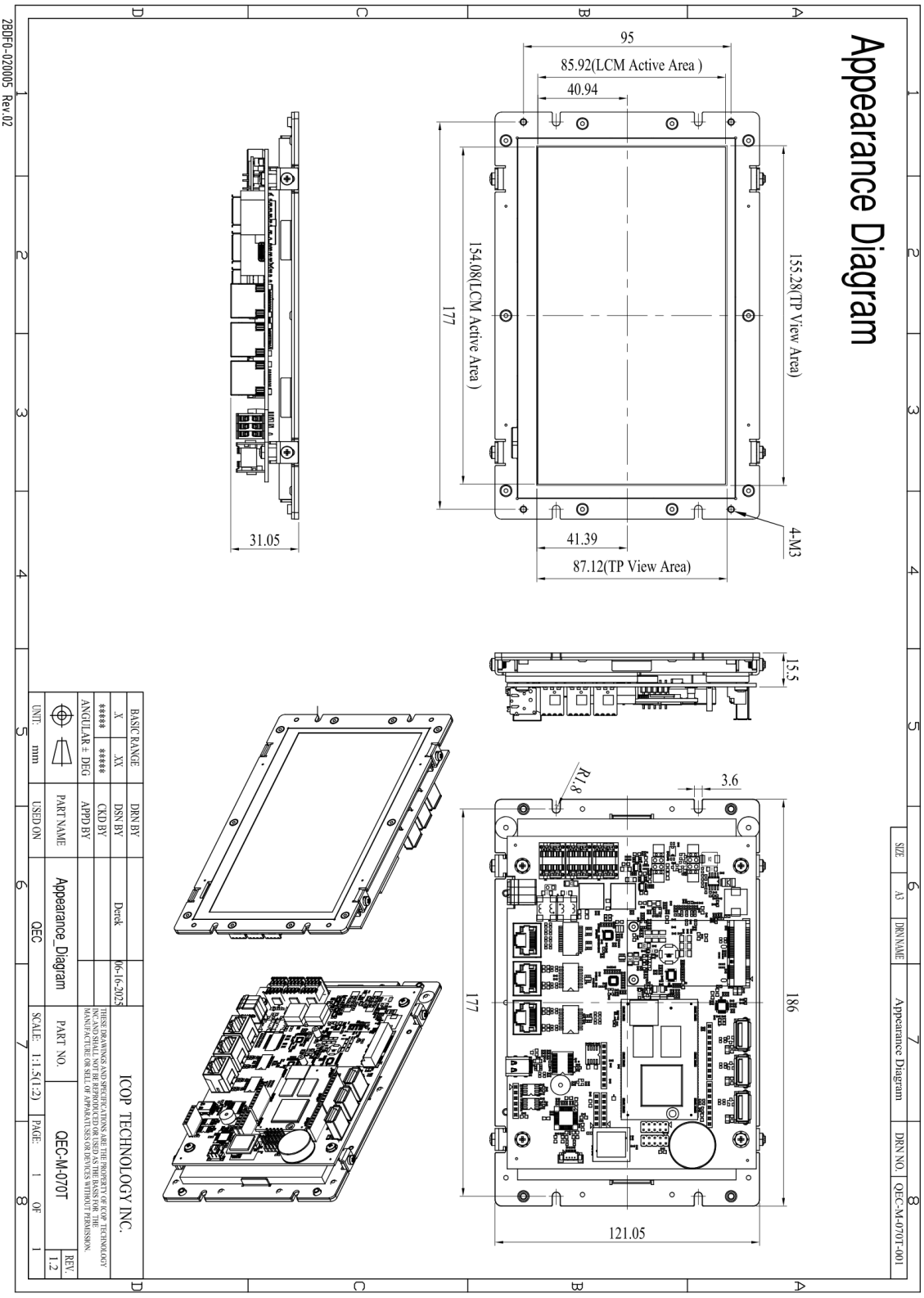
***Note**:

- Operating temperature: -20°C ~ +70°C (standard), -30°C ~ +85°C (extended)
- Avoid installing under direct sunlight or near heat-generating components.
- Protect the touchscreen with a front bezel if mounted in harsh environments.

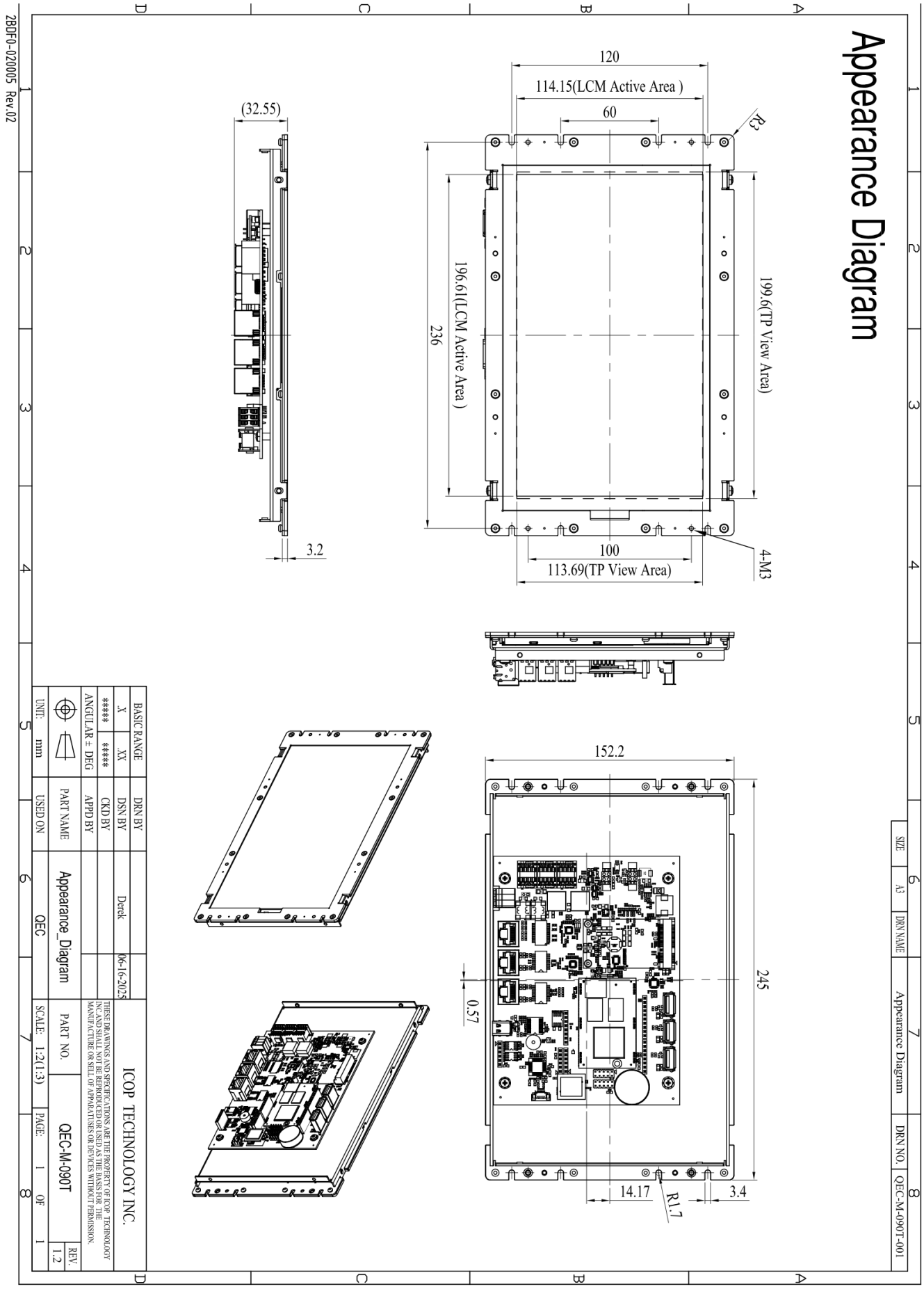
3.1.2 Mounting Dimensions

Mounting Hole and Dimensions Overview.

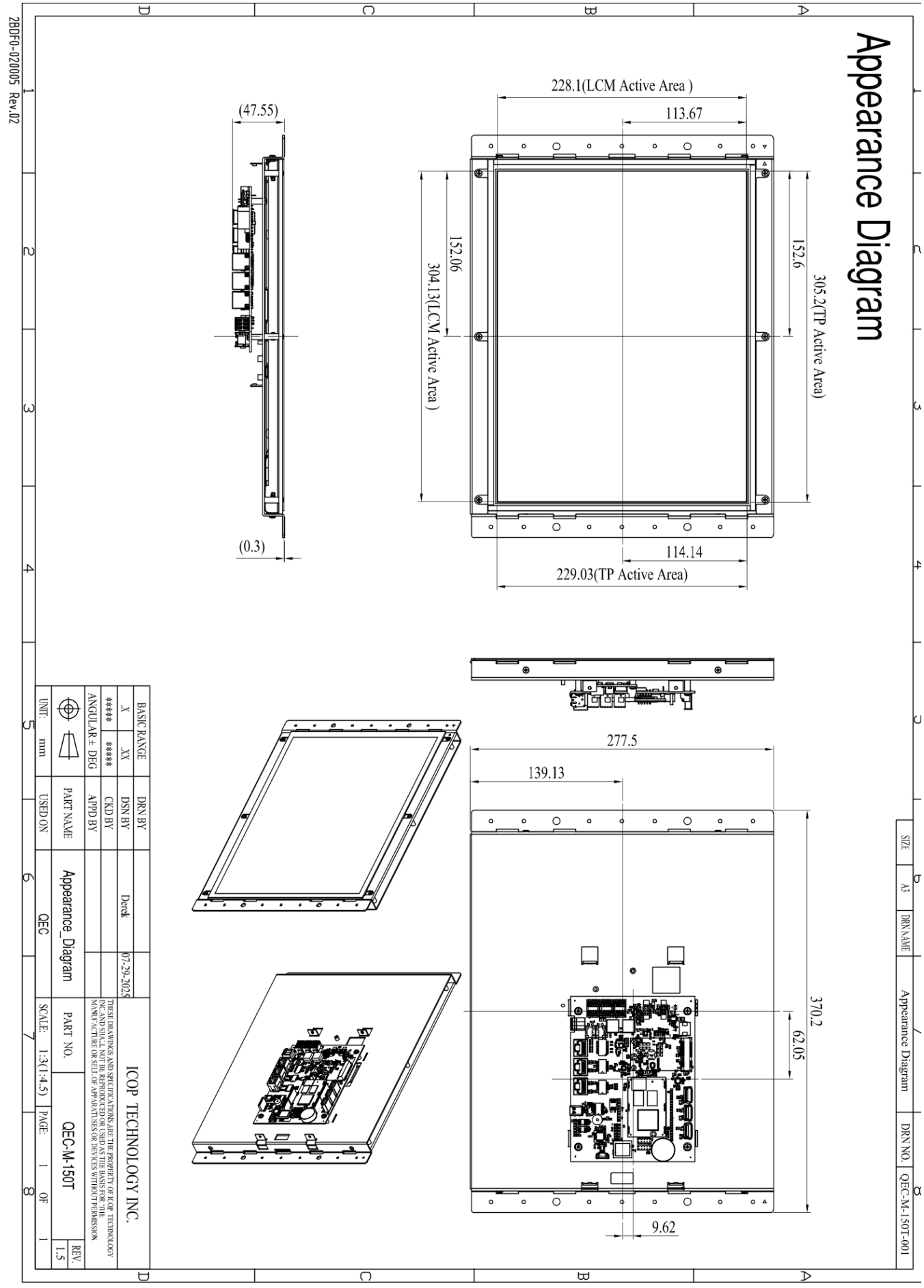
3.1.2.1 QEC-M-070T



3.1.2.2 QEC-M-090T



3.1.2.3 QEC-M-150T



Ch. 4

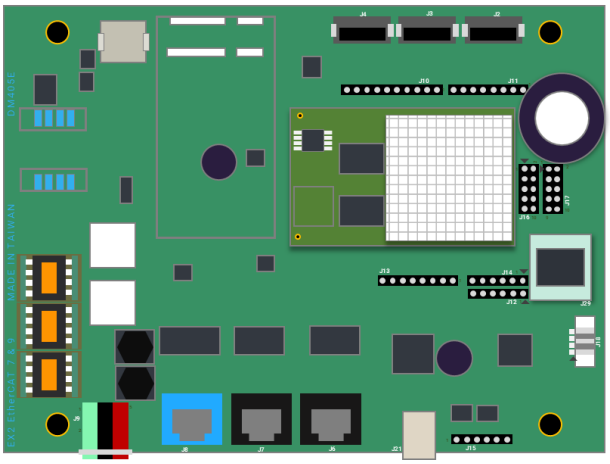
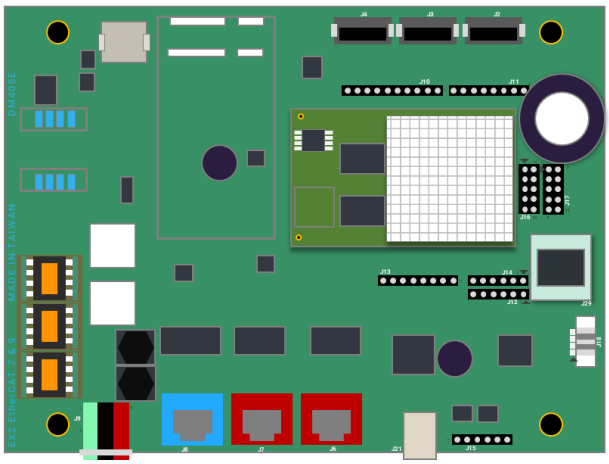
Getting Started

(This chapter is available in multiple languages)

This chapter explains how to start with the QEC MDevice Open-frame series and its software, 86Duino Coding IDE.

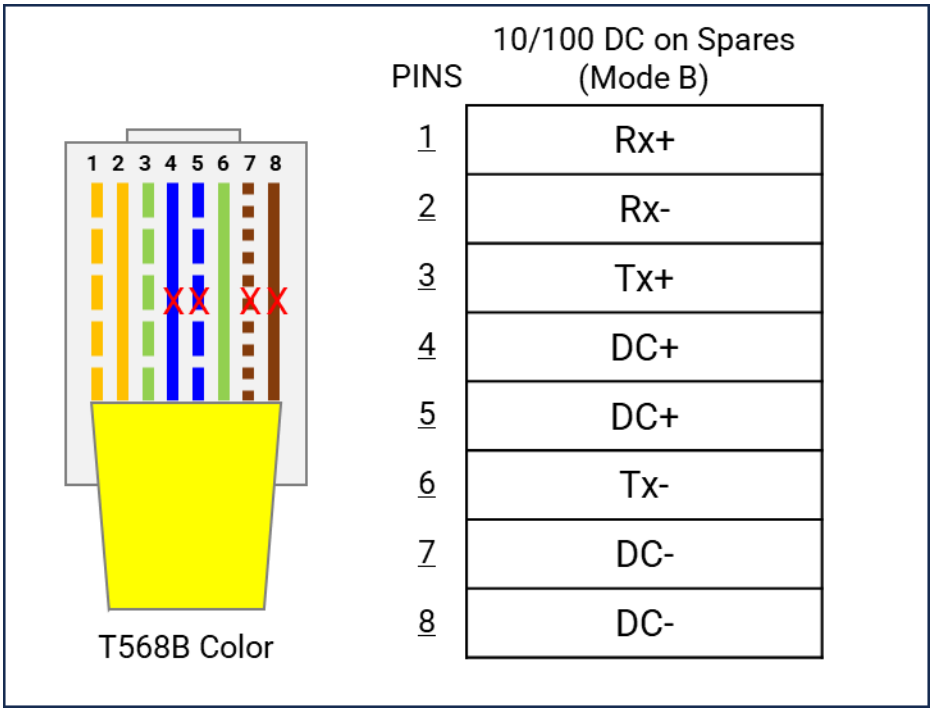
Note. QEC's PoE (Power over Ethernet)

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.

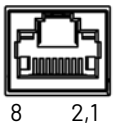
	
Non-PoE type	PoE type

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)

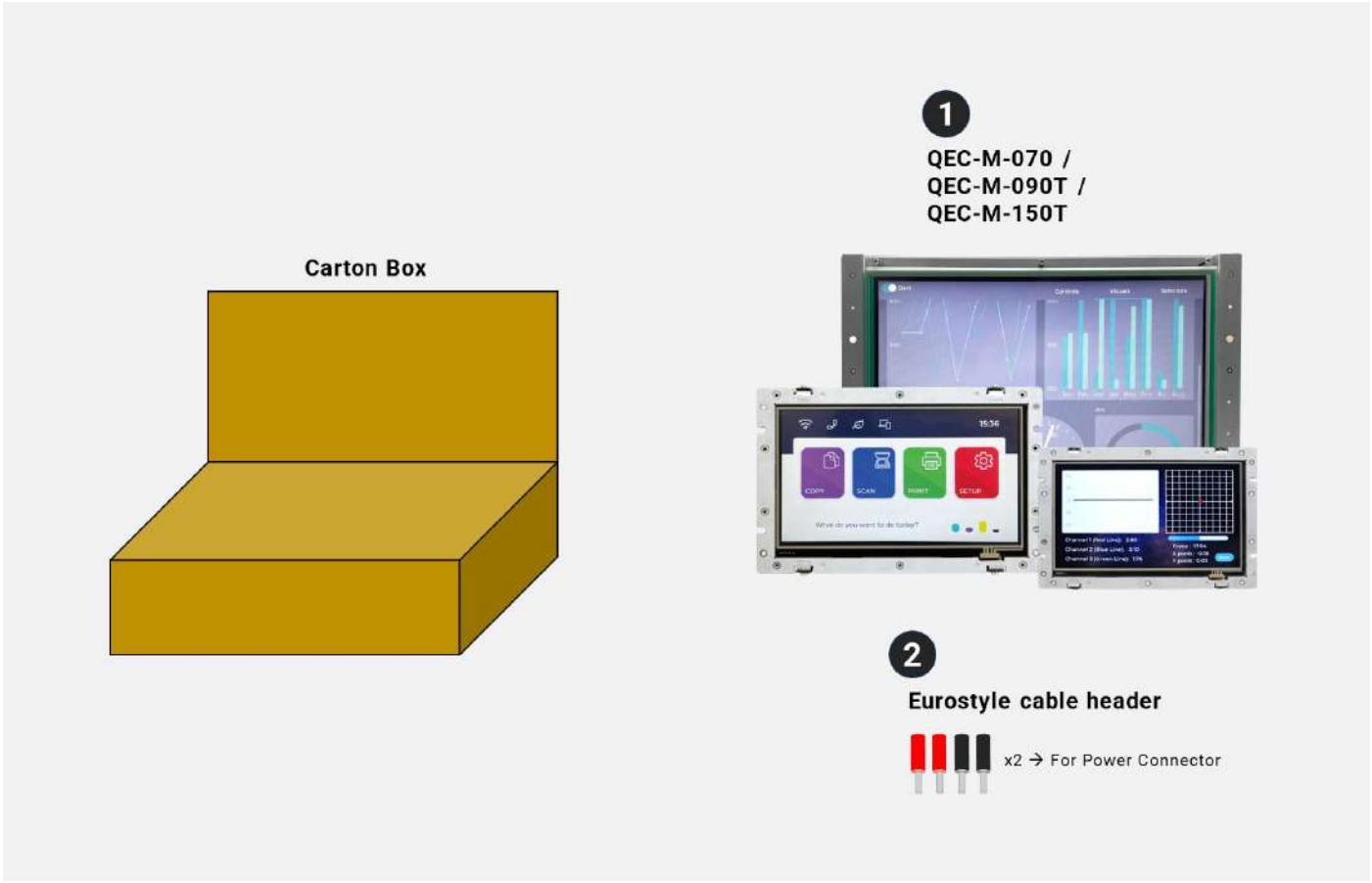
* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

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

4.1 Package Contents

The package includes the following items:

1. QEC-M-070T / 090T / 150T unit x1
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

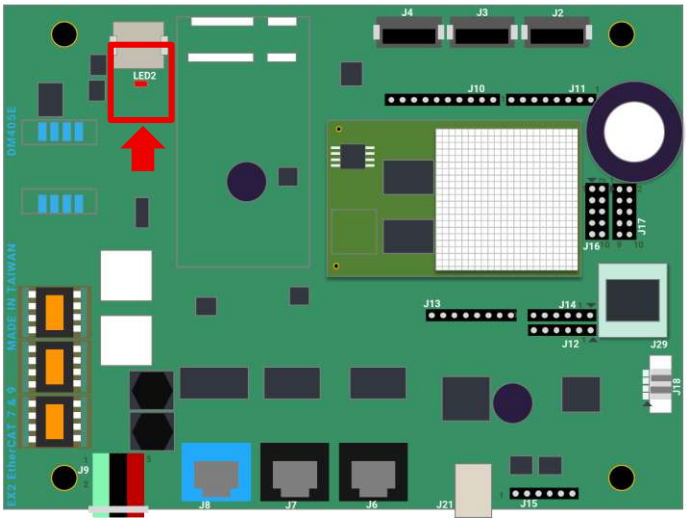
The development environment will be pre-installed before the QEC MDevice is shipped to customers. Users must download the software (see 4.3 Software/Development Environment) and follow this user manual to set up the device.

Plug in the power supply.

*__Note:__ Vs for system power; Vp for peripheral power and backup power.

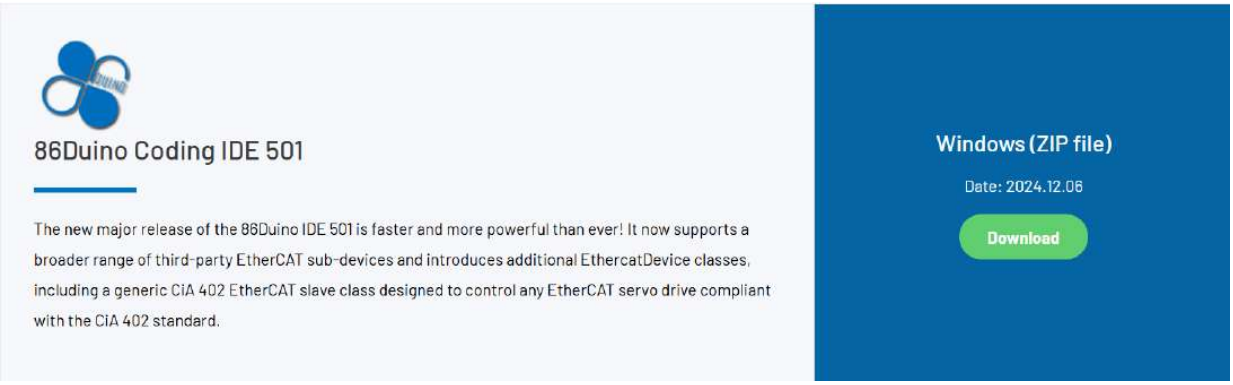
There are two groups of power supplies in QEC MDevice, Vs and Vp; The voltage requirement for both supplies' ranges from 19V to 50V wide voltage.

After powering on, you'll see the power screen light up.



4.3 Software/Development Environment

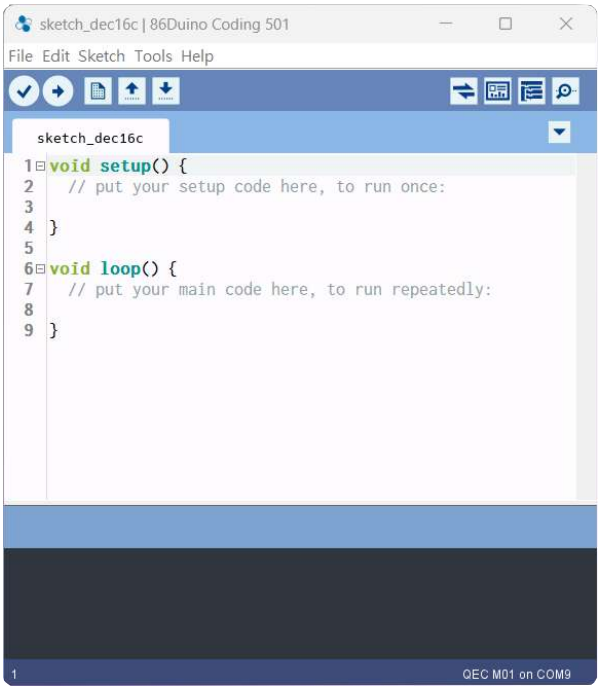
Download 86duino IDE from https://www.qec.tw/software/.



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.

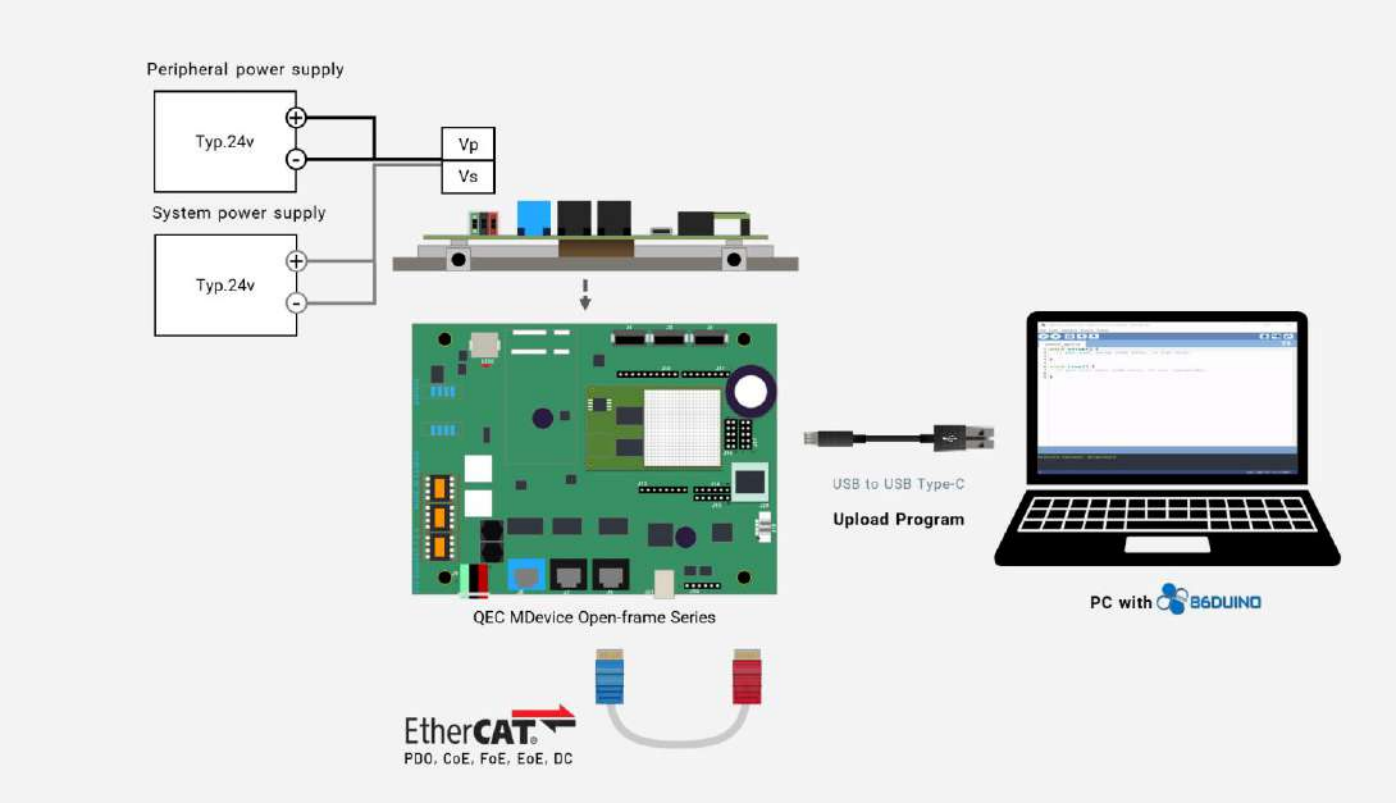
86Duino Coding IDE 501+ looks like below.



4.4 Connect to your PC and set up the environment

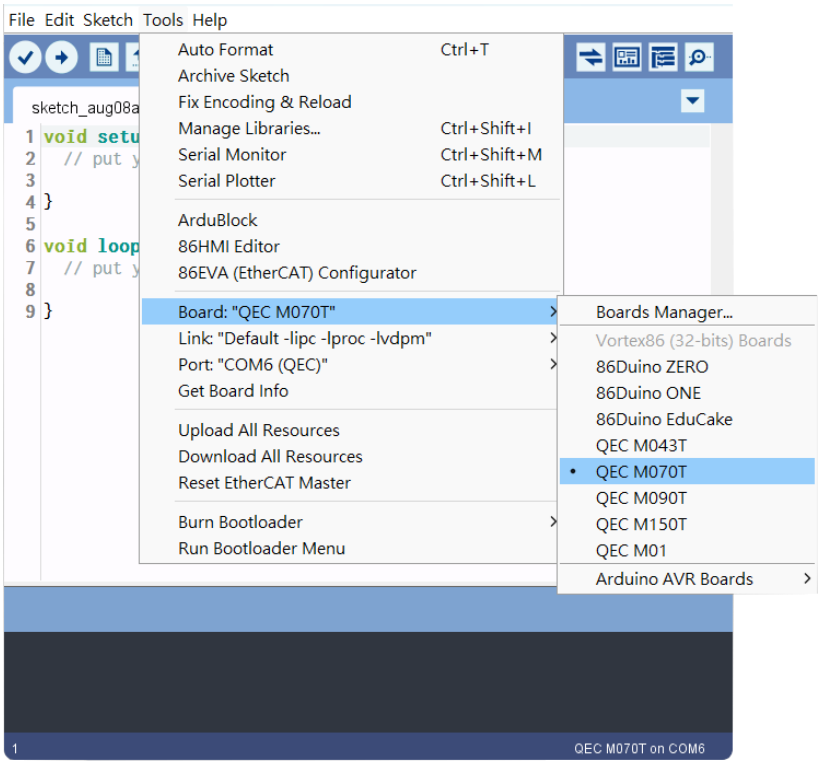
Follow the steps below to set up the environment:

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

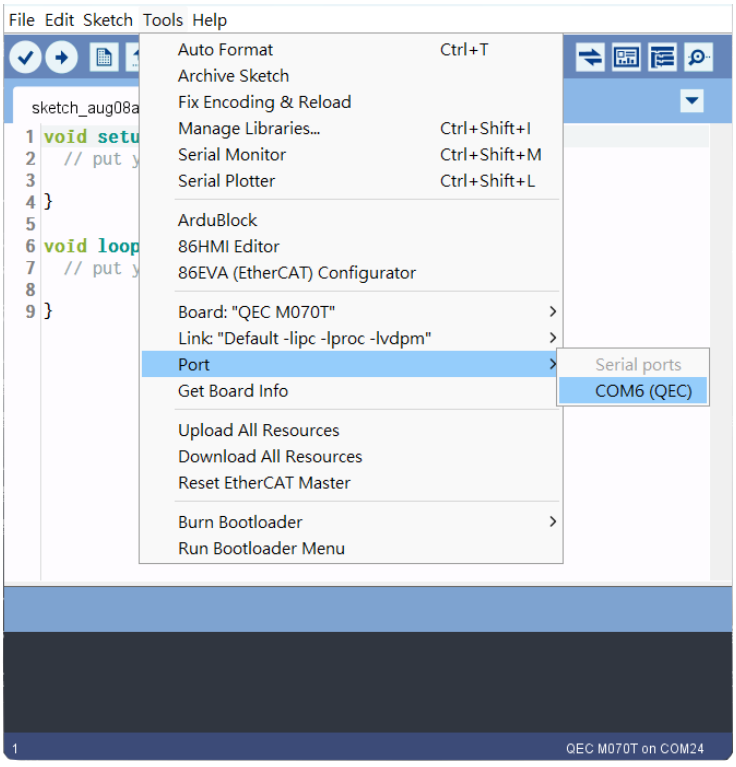


2. Turn on the QEC power.
3. Open "**Device Manager**" -> "**Ports (COM & LPT)**" in your PC and expand the ports; you should see that the "**Prolific PL2303GC USB Serial COM Port (COMx)**" is detected; if not, you will need to install the required drivers.
   (For Windows PL2303 driver, you can download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

4. Open the 86Duino IDE.
5. Select the correct board: In the IDE's menu, select “**Tools**” ->” **Board**”- > “QEC M070T” (or the QEC 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 MDevice (in this case, COM6 (QEC)).



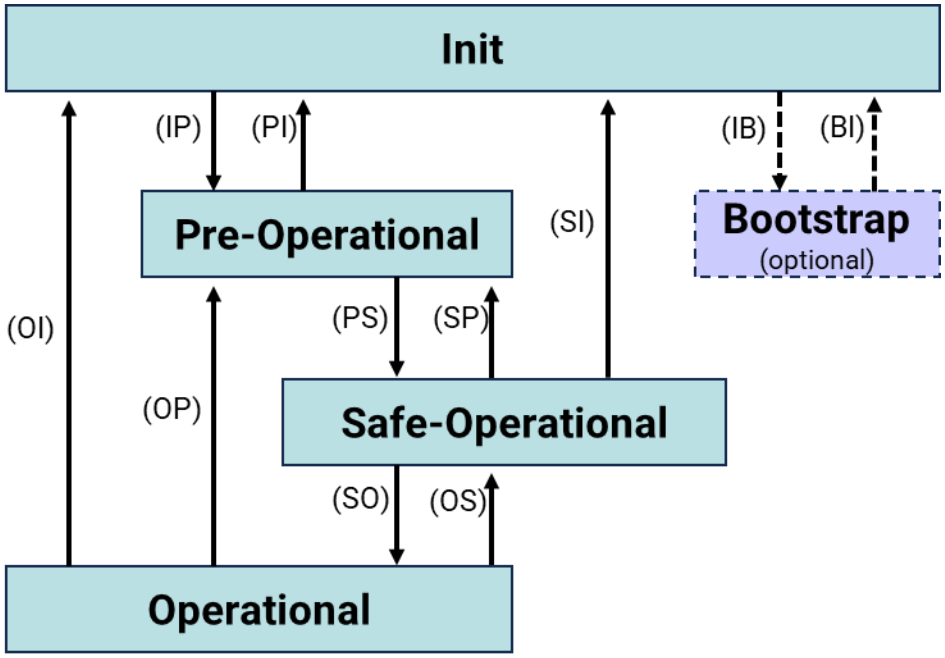
4.5 EtherCAT Communication

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.

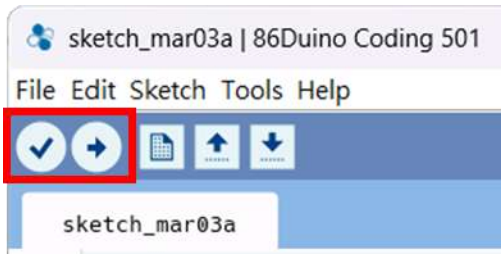
The QEC EtherCAT MDevice can be configured in the EtherCAT network via the EtherCAT library and programmed with the control action in the 86Duino IDE. The 86Duino development environment has two main parts: `setup()` and `loop()`, which correspond to initialization and main programs.

```
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.

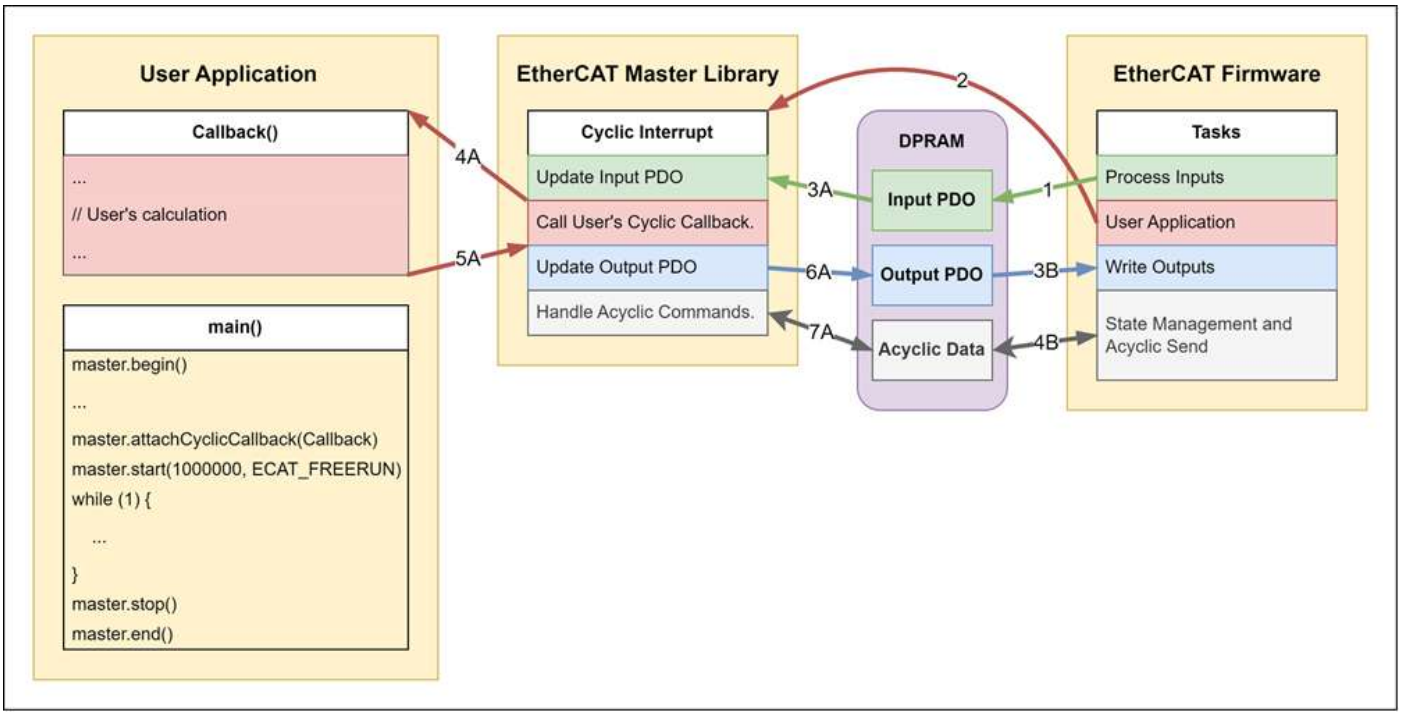
***Note:** Once the code is written, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload.



4.5.1.1 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:

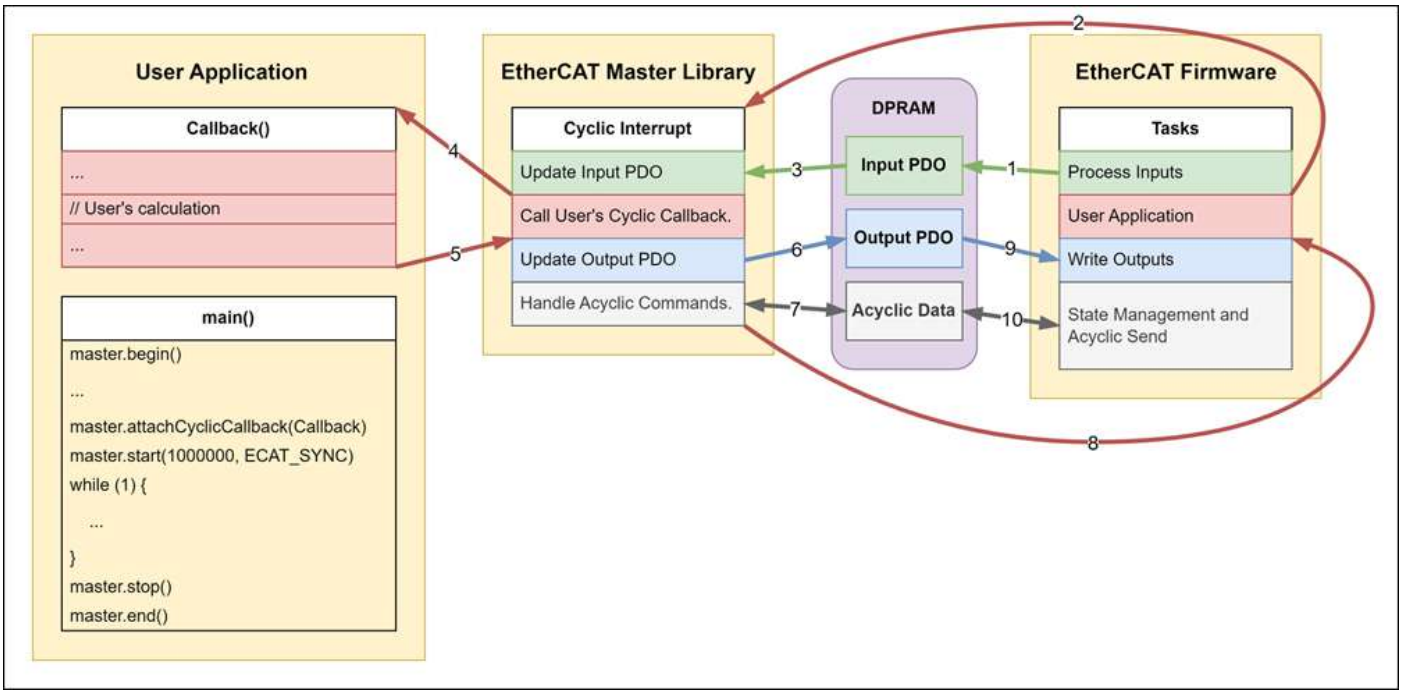
```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

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

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

4.5.1.2 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.

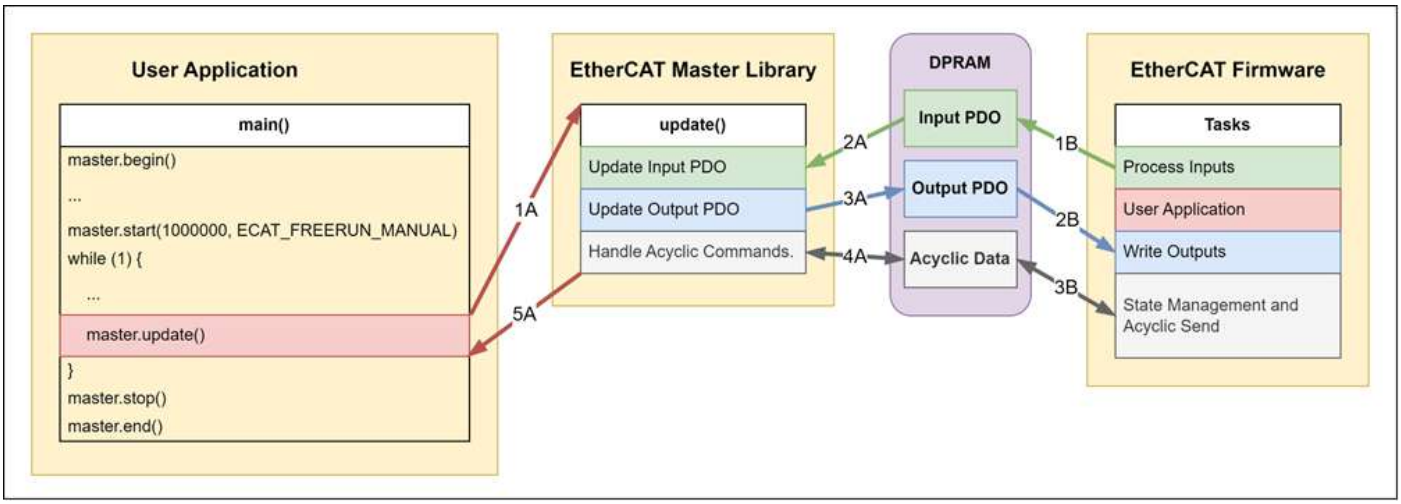
Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

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

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

4.5.1.3 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.

4.5.2.1 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:

}
```

4.5.2.2 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:

}
```

4.5.2.3 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);
```

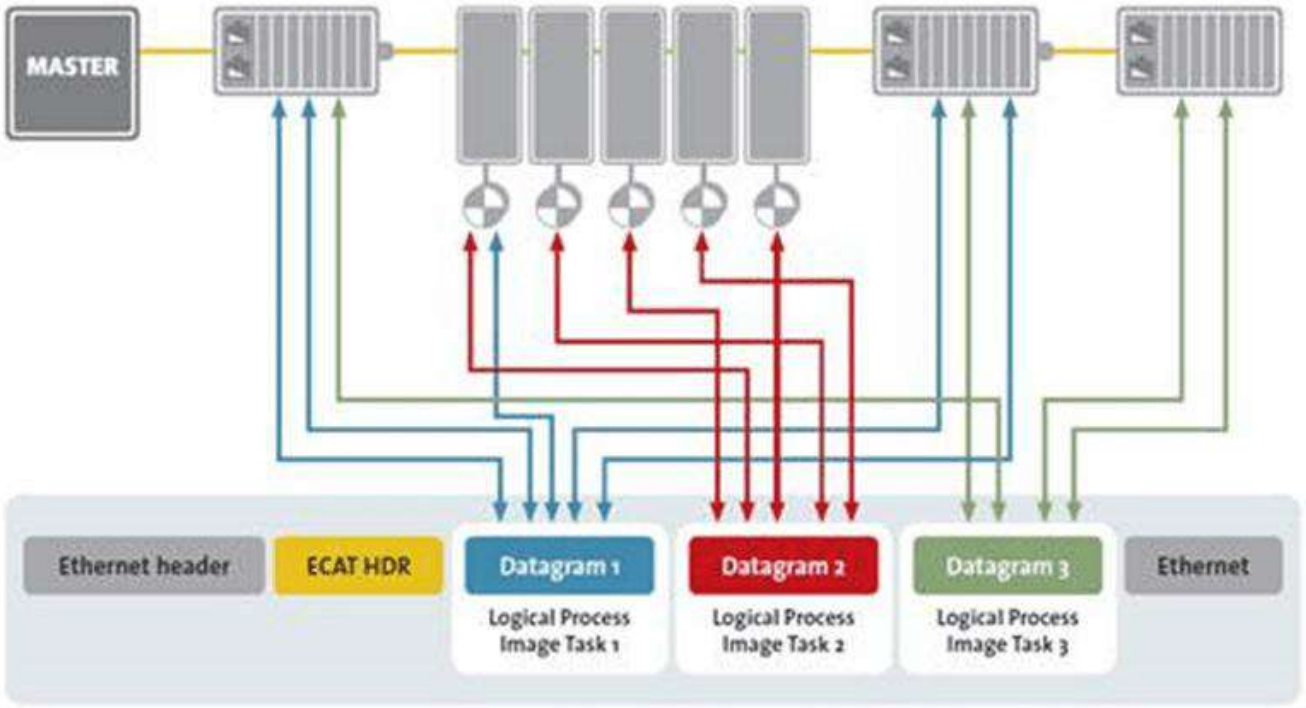
```
    Serial.print("      Alias Address: ");
    Serial.println(slave.getAliasAddress());

    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/)

4.5.3.1 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
}
```

4.5.3.2 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
}
```

4.5.3.3 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
}
```

4.5.3.4 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
}
```

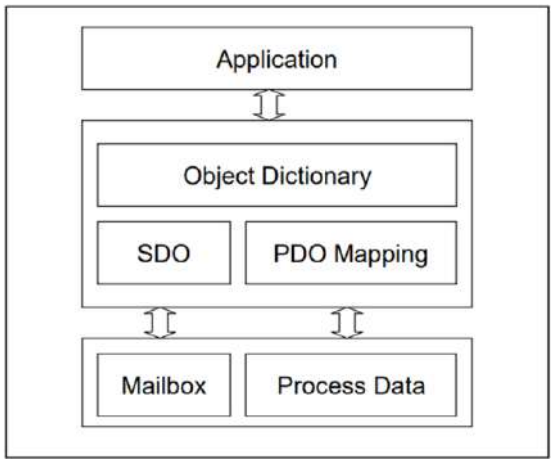
4.5.4 CANopen over EtherCAT (CoE) Functions

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.

4.5.4.1 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:

}
```

4.5.4.2 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:

}
```

4.5.4.3 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:

}
```

4.5.4.4 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:

}
```

4.5.4.5 Example 5: Print the PDO mapping configuration

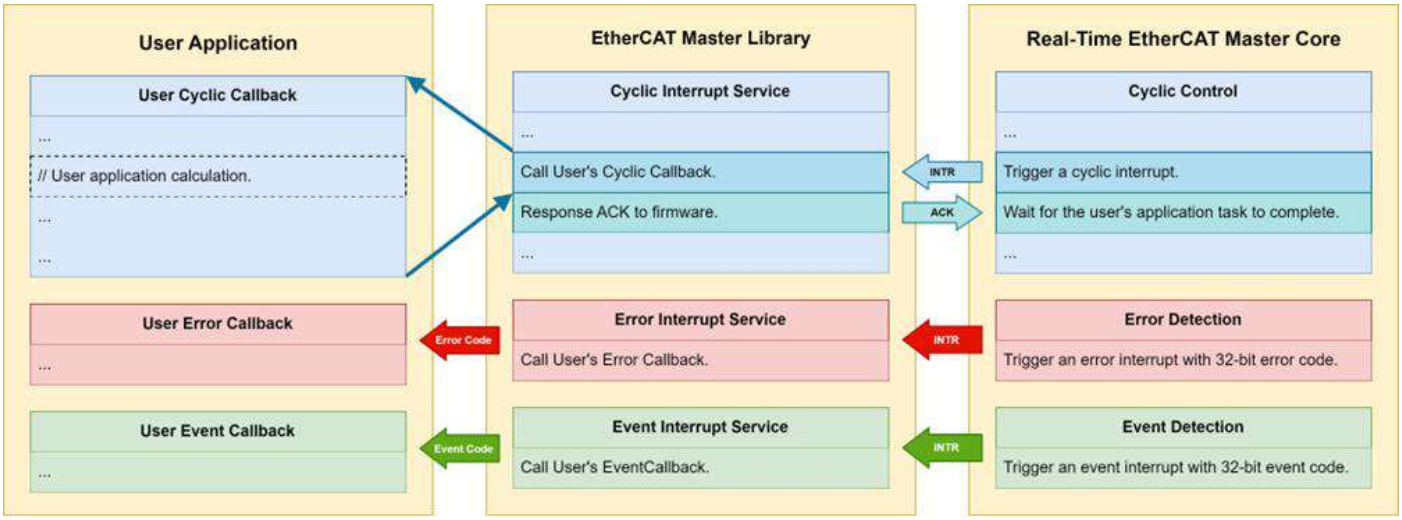
Print the PDO mapping configuration.

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.

4.5.5.1 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:

}
```

4.5.5.2 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
}
```

4.5.5.3 Example 3: Event callback

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

Here is the example code:

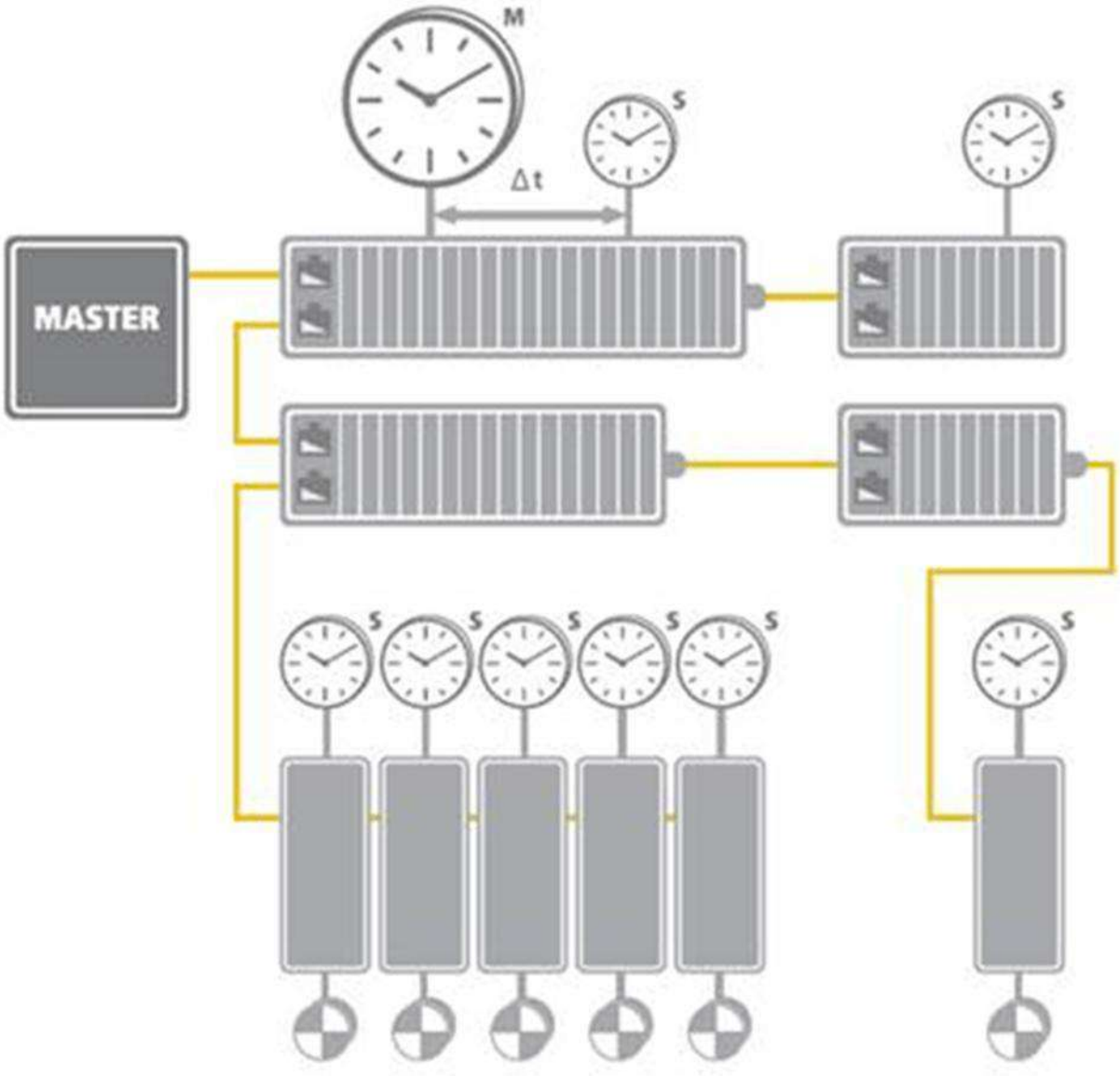
```
#include "Ethercat.h"

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).

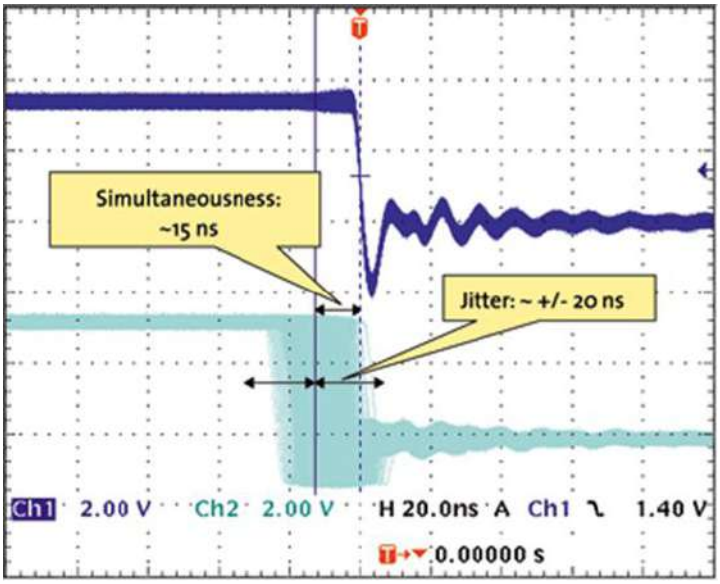


EtherCAT: Illustration of Distributed Clock (DC). (Source of information: http://www.ethercat.org/)

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.

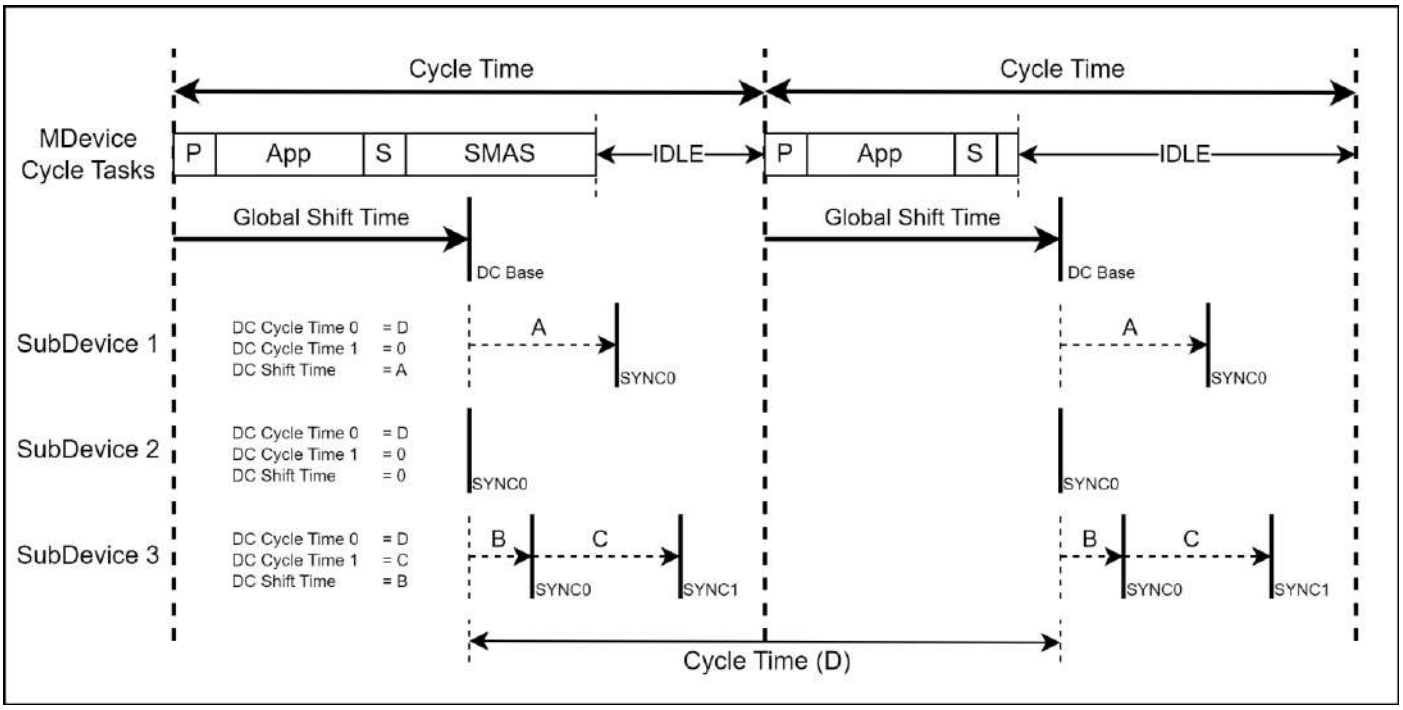
(*1) Please refer to EtherCAT standard document ETG1000.4



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.



4.5.6.1 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			

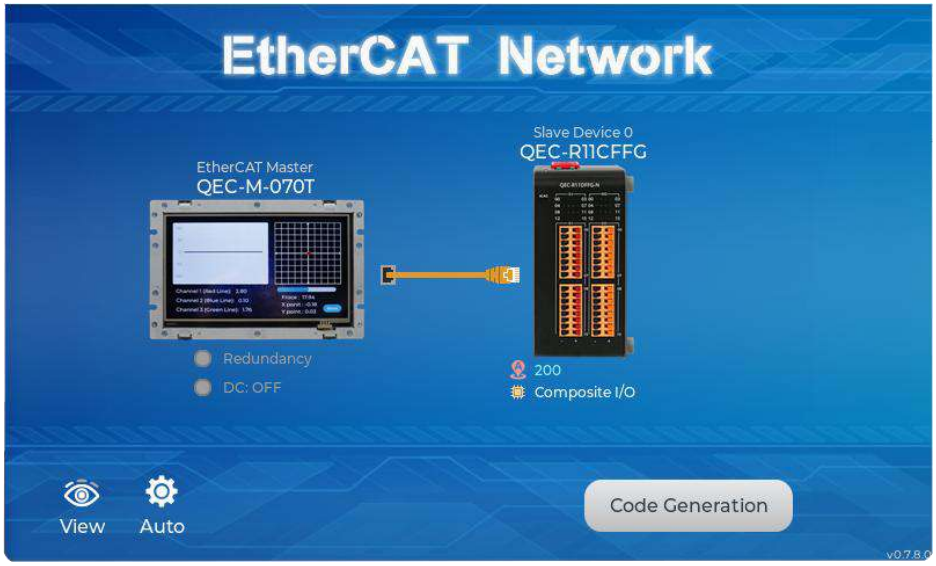
Here is the example code:

```
#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.



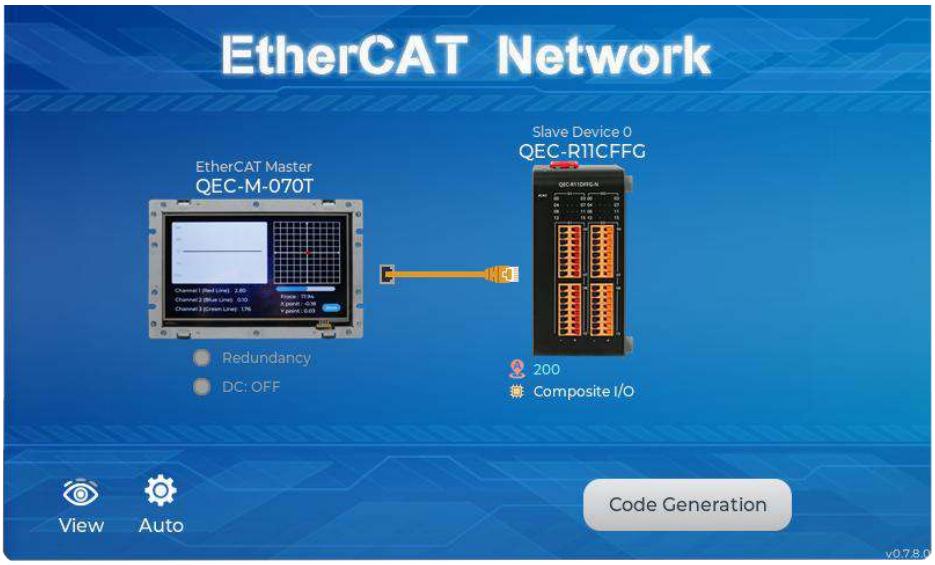
The following information about 86EVA will focus on QEC EtherCAT MDevice and SubDevices, with features including:

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-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-070T:



- QEC-M-090T:

- QEC-M-150T:

In this section, we will use the **QEC-M-070T** as an example.
Press twice on the image of the QEC-M-070T to see the MDevice's 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.

For more detailed information about the 86EVA tool, please refer to the 86EVA User Manual.

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

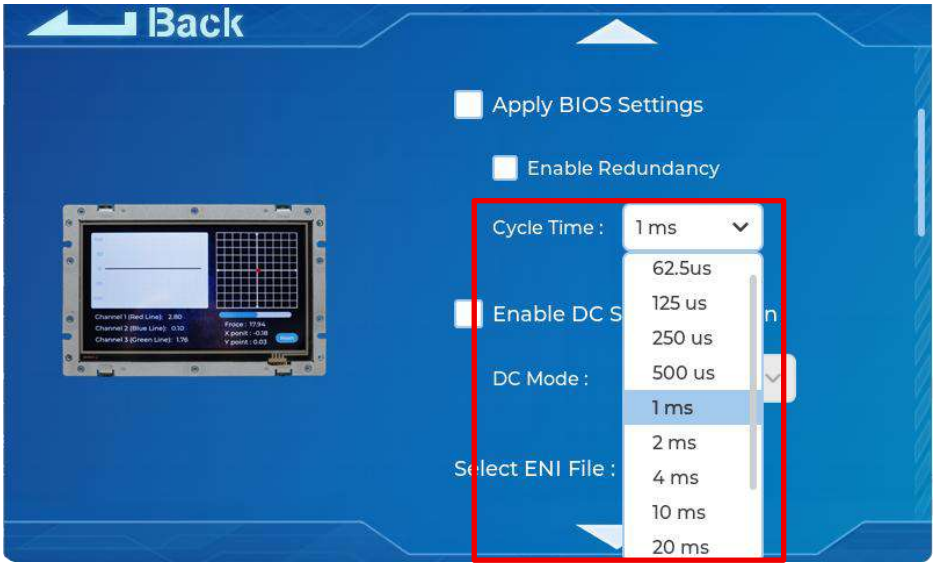
If users select **“Apply BIOS Settings”**, then the select box **“Enable Redundancy”** and the dropdown menu **“Cycle Time”** will not be available.

While not using the BIOS settings, users can select to “**Enable Redundancy**” function.

This feature ensures continued EtherCAT communication in the event of a cable or device failure by creating a ring topology between the MDevice and SubDevices.

***Note:** If the Redundancy option is enabled, you must physically connect the second EtherCAT cable to complete the ring. Failure to connect the redundancy cable will prevent the EtherCAT State Machine from transitioning to the OPERATIONAL state, and communication with SubDevices will fail.

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.

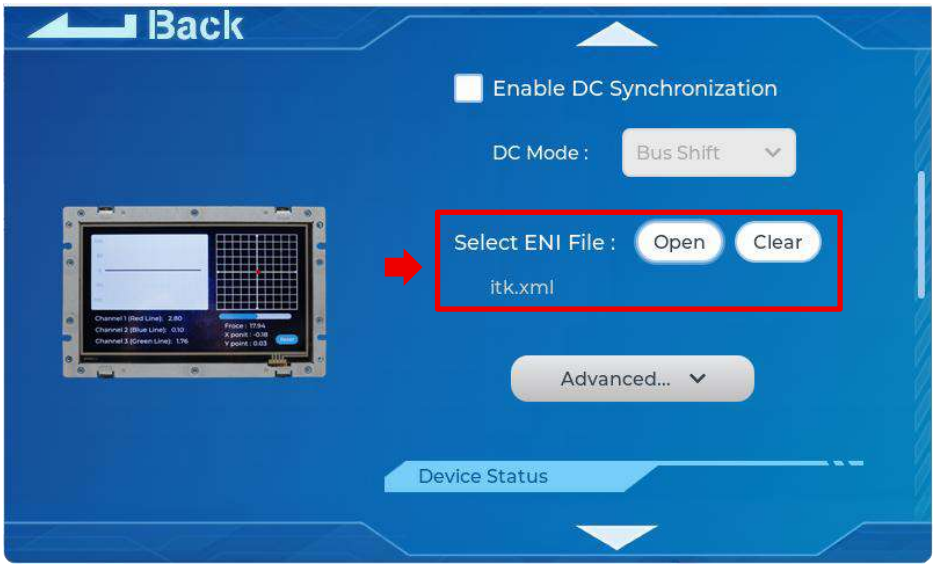
Users can also select DC mode by the dropdown menu **“DC Mode”**, with options “**Bus Shift**” and “**Master Shift**”.

***Note:** the DC function and mode need to follow the SubDevice specification.

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

For how to import the ENI into the 86EVA, please refer to Ch. 4.5.8 Import ENI to QEC MDevice.

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



You can click the “**Clear**” button to delete the ENI file that you imported.

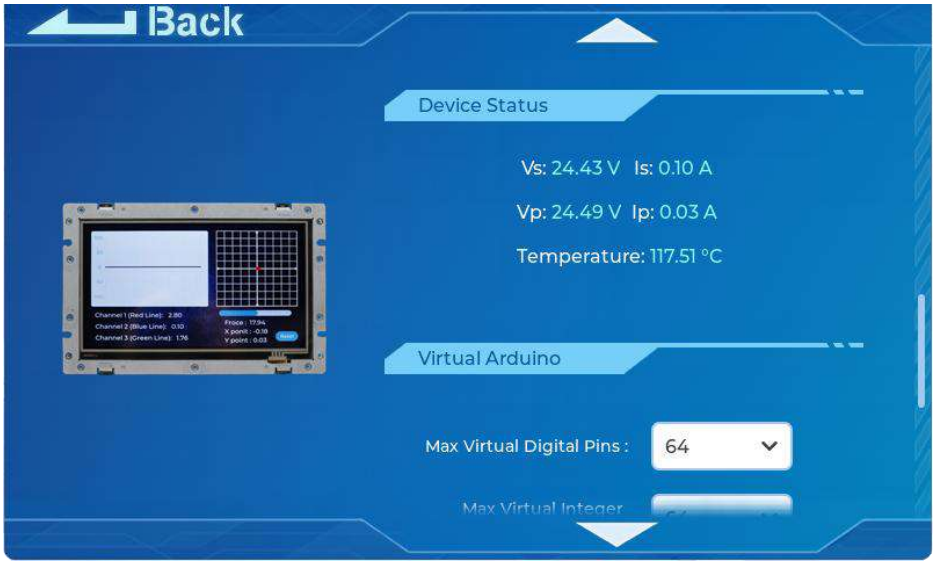
In **“Advanced”** section, you can configure Error Reactions for the EtherCAT network. These settings help improve system stability and fault recovery when a communication error occurs.

Two options are available under Error Reactions:

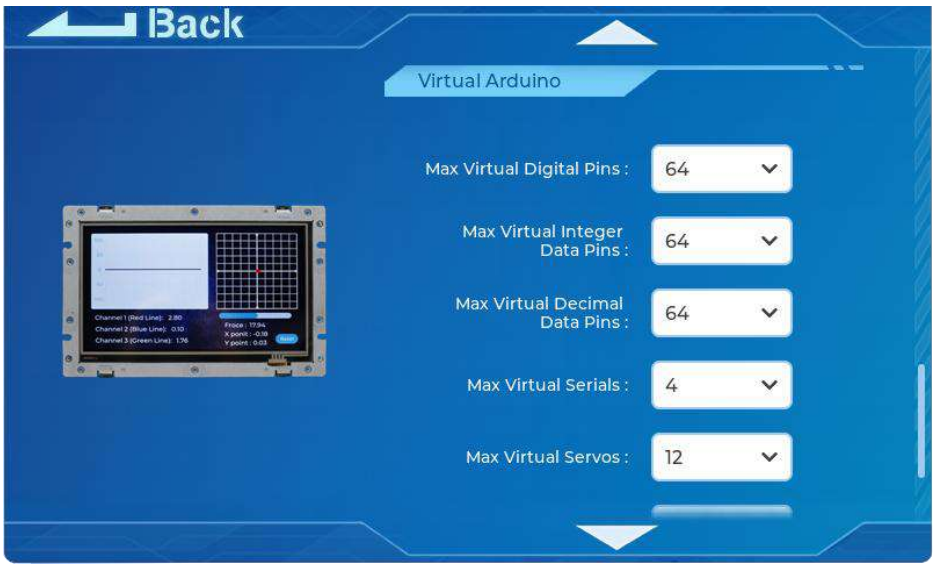
1. **Set All Slaves to Safe-OP When an Error Occurs**
   When enabled, this option forces all EtherCAT slave devices into Safe-Operational (Safe-OP) state if any slave reports an error or abort code. This provides a unified and predictable fallback behavior during runtime faults.
2. **Auto Restart Slaves When Error-Free**
   This option can only be enabled if the first setting is active. When selected, the EtherCAT master will automatically attempt to restore communication and return all slaves to their previous state once the error condition clears.

You can also click the “**Brief Options**” button at the bottom of the page to collapse the Error Reactions section and simplify the view.

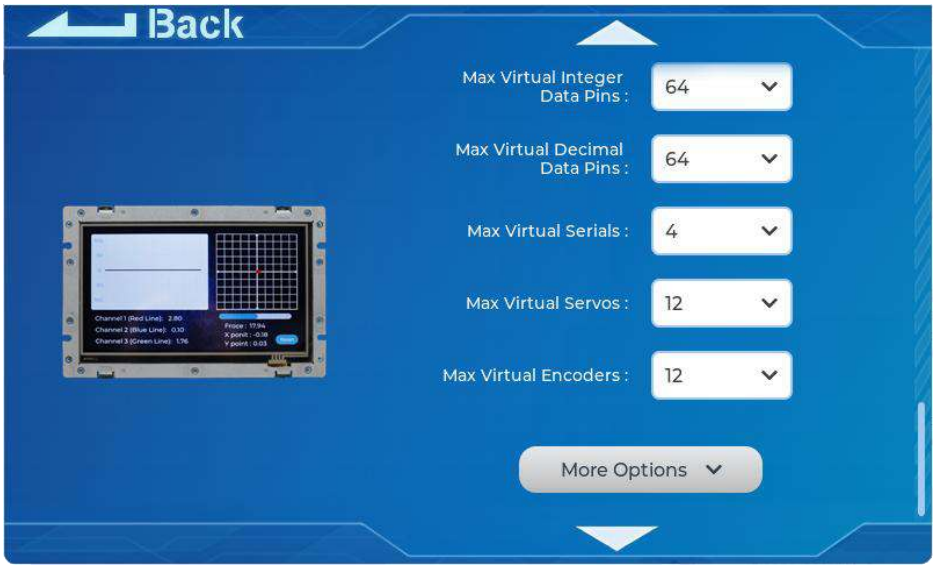
You can also see the voltage, current and system temperature in the **“Device Status”** area.



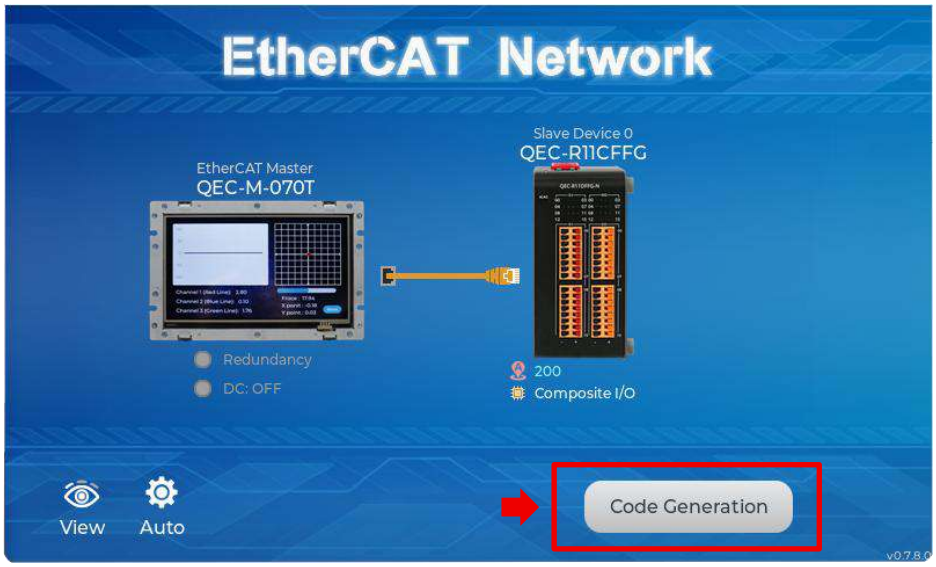
Users can also set the maximum number of the **Virtual Arduino** functions.



You can click the “**More Options**” button to show more Virtual Arduino object settings.



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

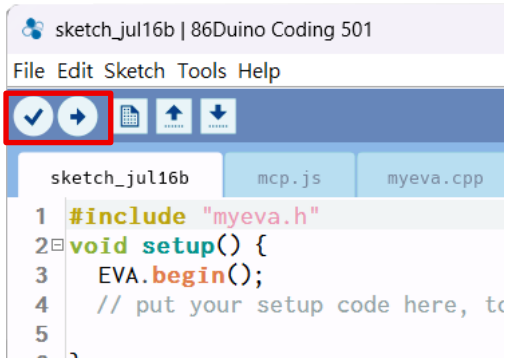


When you're done, double-click the “**OK**” button to turn off 86EVA, or it will close in 10 seconds.

The generated code and files are as follows:

a. sketch_jul23d: Main Project (.ino, depending on your project name).
b. myeva.cpp: C++ program code of 86EVA.
c. 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:

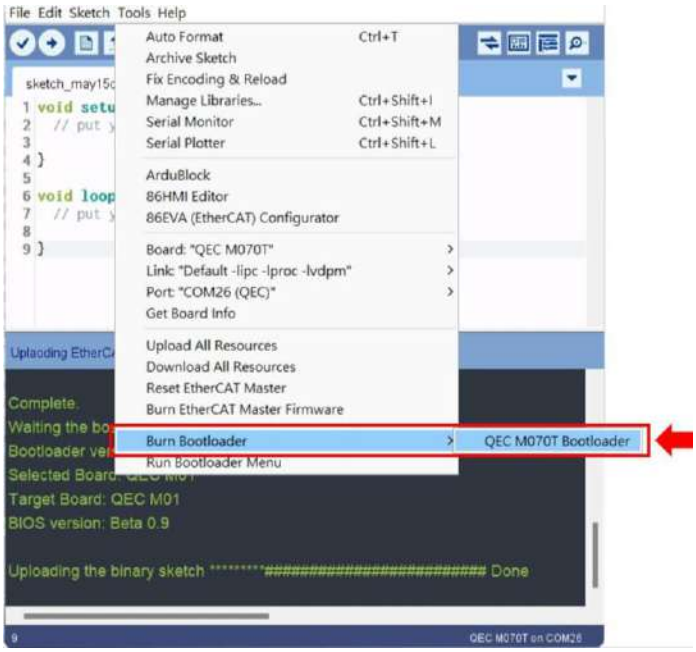
Step 1: Setting up QEC-M

1. Download and install 86Duino IDE 500 (or a newer version): You can download it from Software.
2. Connect the QEC MDevice: Use a USB cable to connect the QEC MDevice 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-070T**" (or the specific model of QEC MDevice you are using).
5. Select Port: From the IDE menu, choose "**Tools**" -> "**Port**" and select the USB port to which the QEC MDevice is connected.

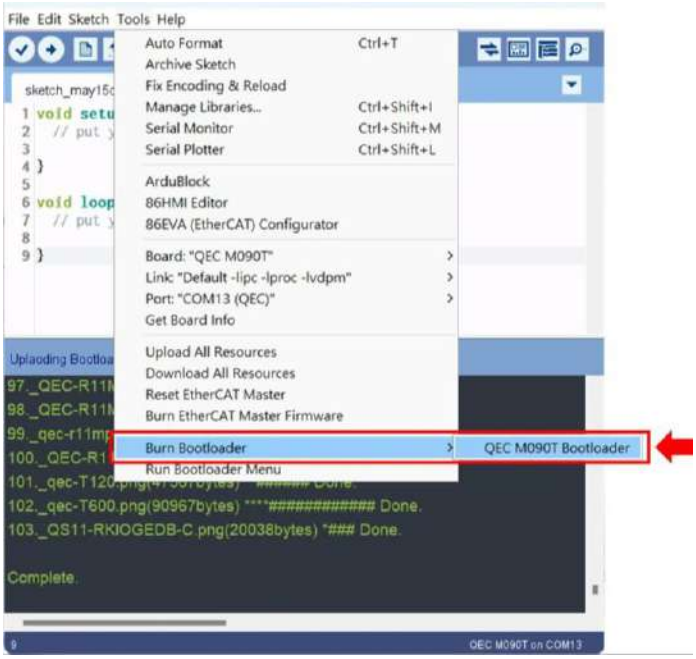
Step 2: Click “Burn Bootloader” button

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

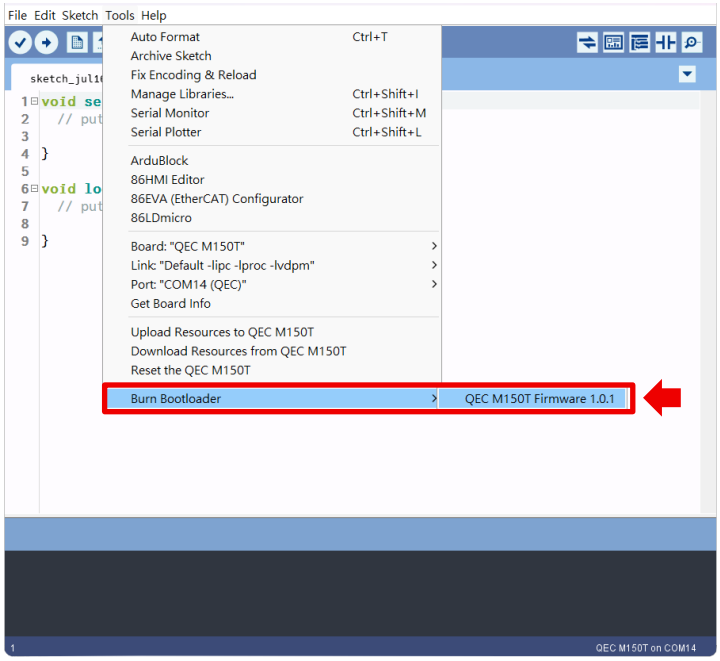
- QEC-M-070T:



- QEC-M-090T:



- QEC-M-150T

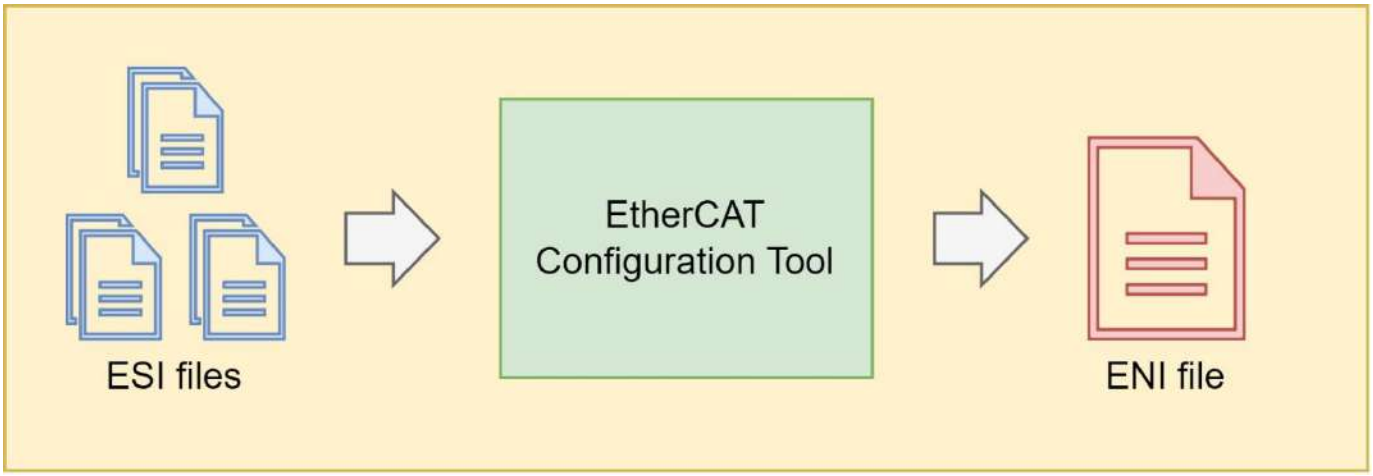


Step 3: Complete the Update

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 MDevice

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.

4.5.8.1 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.)

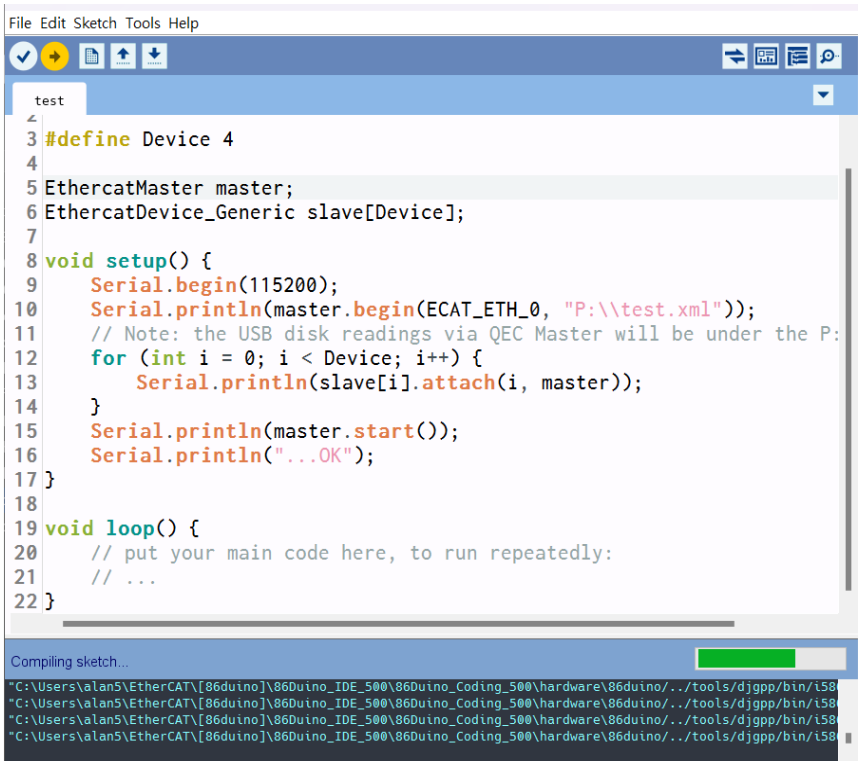
Step 1: Write the Import Code:

- 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:
    // ...
}
```

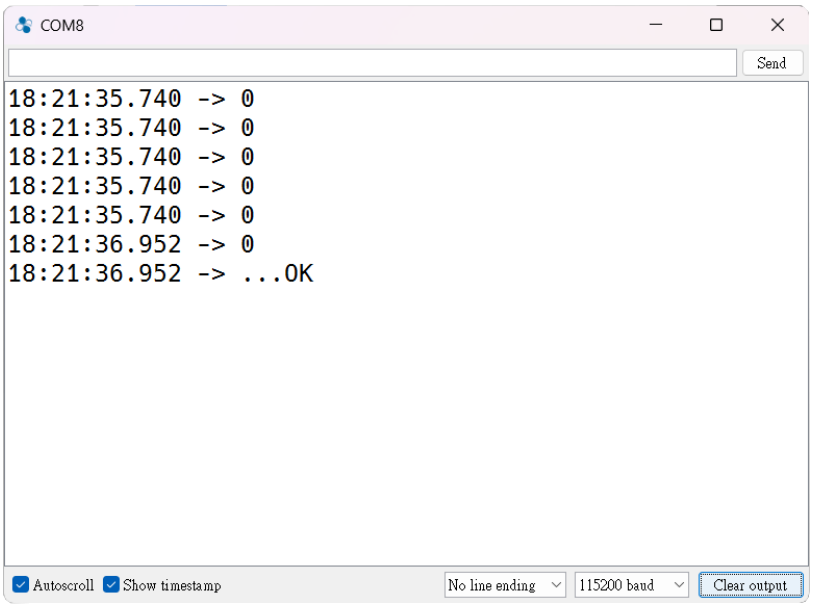
Step 2: Upload the Code:

- Upload the code to your QEC EtherCAT MDevice.

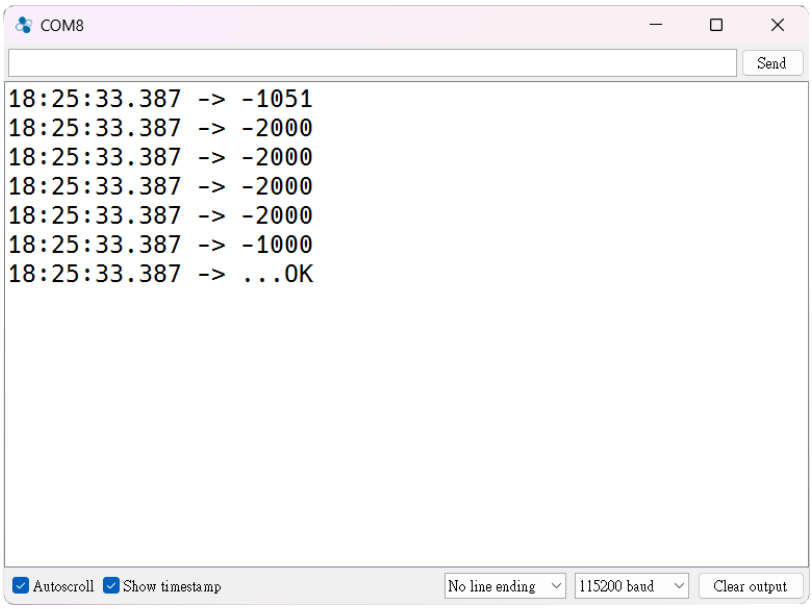


Step 3: Verify the ENI File Import:

- 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:



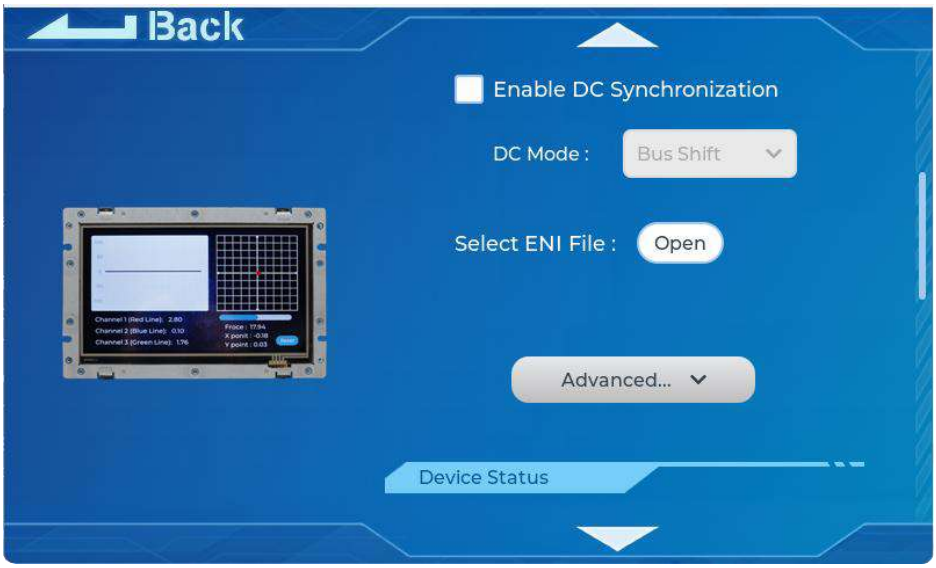
- If there is an error, such as -1051 (which means `ECAT_ERR_MASTER_ENI_MISMATCH`), refer to the EtherCAT API user manual for other error codes and their meanings.



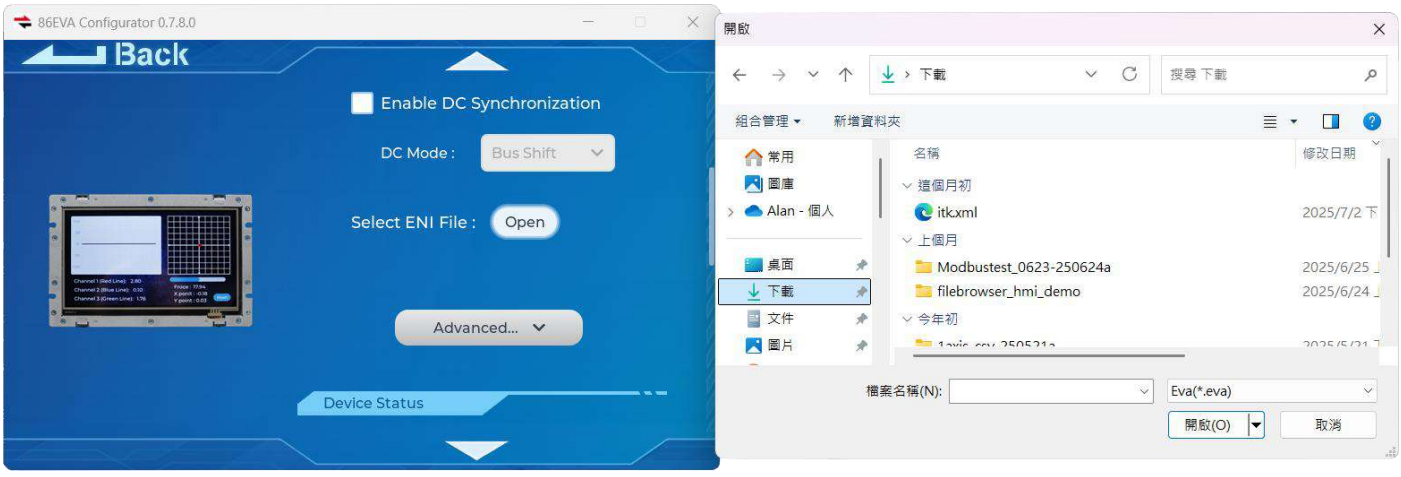
4.5.8.2 Method 2: Using 86EVA

Please refer to the Ch 4.5.7 86EVA for 86EVA setup.

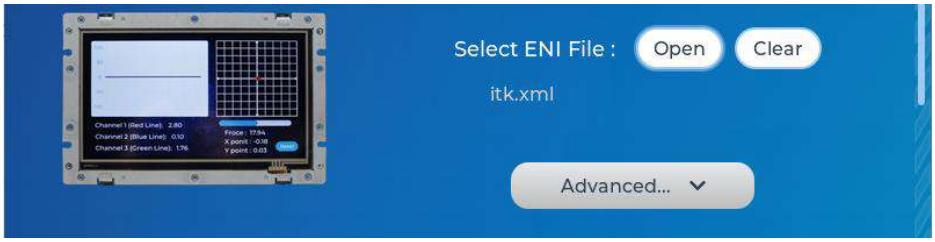
After you turn on 86EVA and scan the EtherCAT Network successfully, please enter to the QEC MDevice setting page, and go down to the "**Select ENI file**" section.



Click on the "**Open**" button next to the "**Select ENI file**", and you will see the open file window. Browse where you saved the ENI file, and open it.

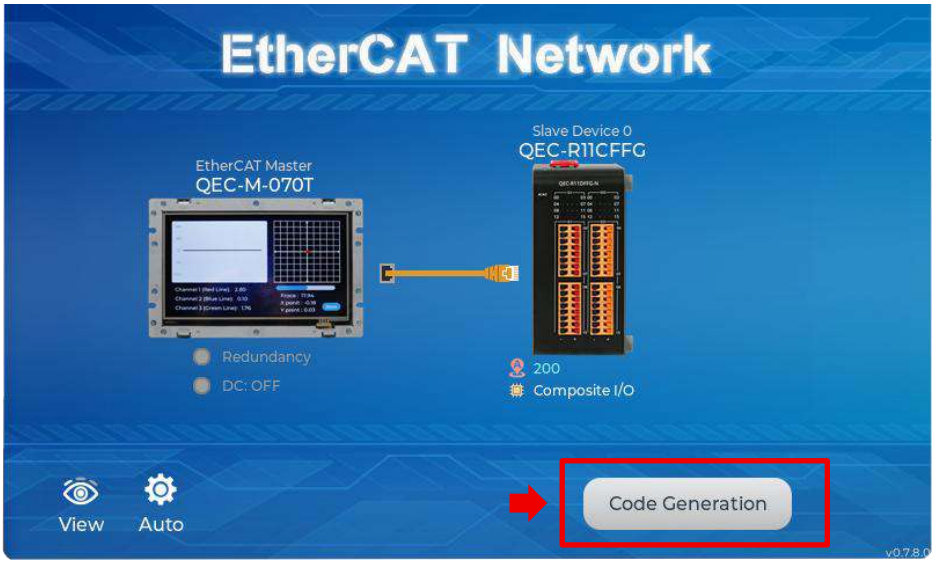


After you import the ENI file into the 86EVA, you can see the ENI file name.

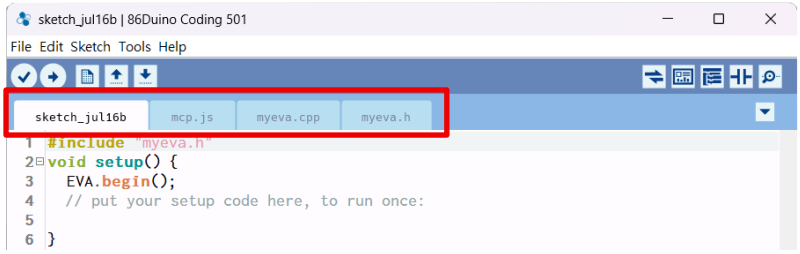


Please click “**Back**” button in the upper left corner to return.

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:

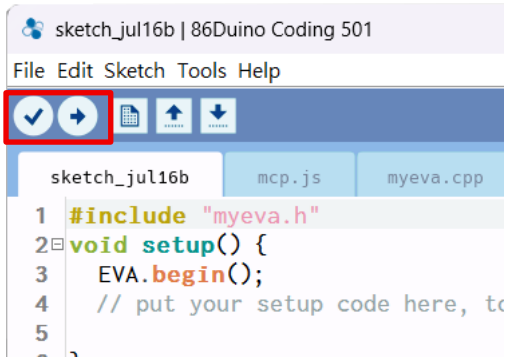


a. sketch_jul23d: Main Project (.ino, depending on your project name).
b. myeva.cpp: C++ program code of 86EVA.
c. myeva.h: Header file of 86EVA.

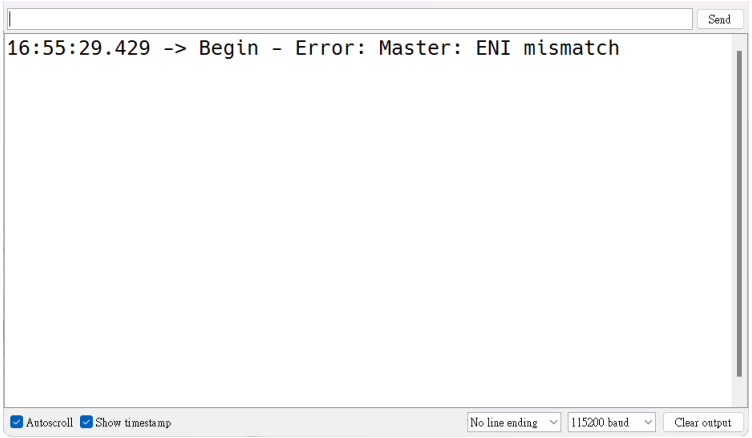
Then, we start to write the code to config your EtherCAT system.
Please insert `Serial.begin(115200);` before `EVA.begin();` in `void setup() { }`, so the EVA program return value can display in Serial Monitor (in baud rate 115200).

```
#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.



After we uploaded the program successfully, we have to 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.

In this section, we will use the **QEC-M-070T** as an example.

4.6.1 Turn on Bootloader Menu

The Bootloader Menu can be opened by touching the left-top side of the LCD screen when your QEC Open-frame MDevice is opening.

You will see the three bars on your QEC MDevice after you keep pressing the left-top side. It'll show "**Keep Pressing**" in the first bar.

Shows "**Touch Calibration**" in the second bar.

Shows “**Bootloader Menu**” in the third bar.
You can leave the LCD in this bar, then you’ll enter the Bootloader Menu page.

You can see the bootloader menu like in the picture below.

In the following introduction, I’ll only use the Bootloader picture.

***Note:** The Bootloader Menu can be opened in the same way on all QEC MDevice Open-frame series.

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.

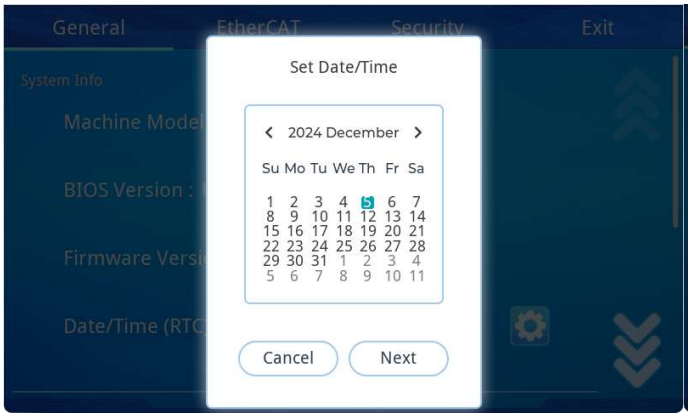
4.6.2.1 System Info

- **Machine Model**: Displays the current hardware model (e.g., QEC-M-070T, QEC-M-150T).
- **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.

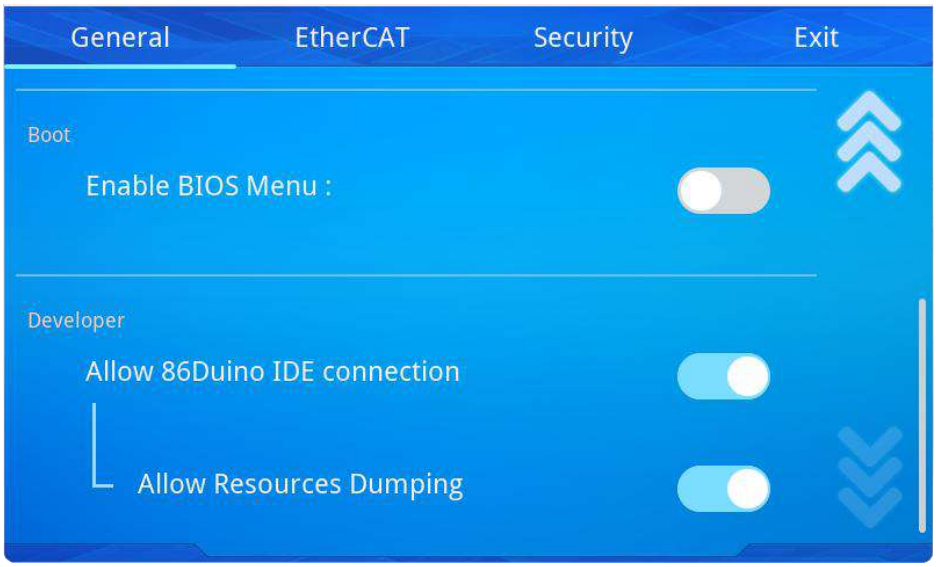
***Additional Information:**

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.



4.6.2.2 Boot and Developer Options



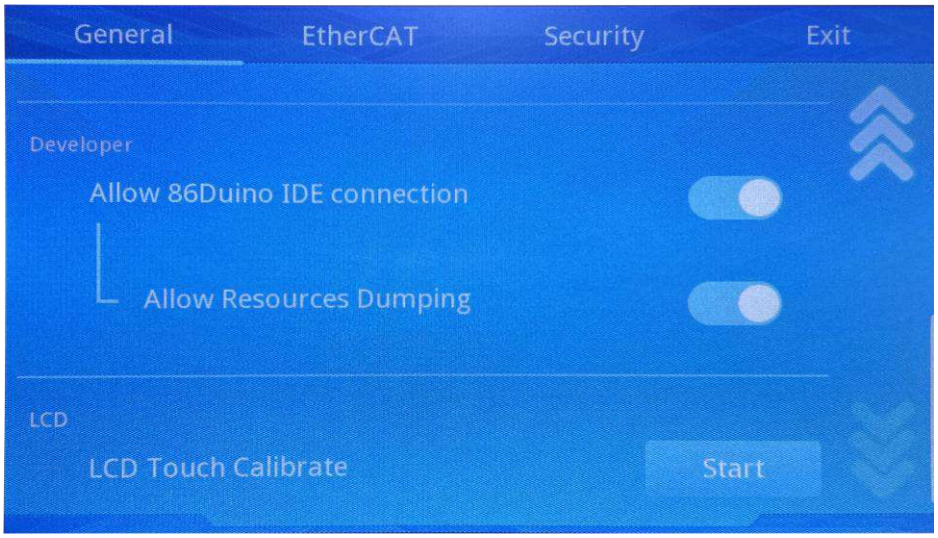
Boot:

- **Enable BIOS Menu**: Enable or disable BIOS Menu.

Developer:

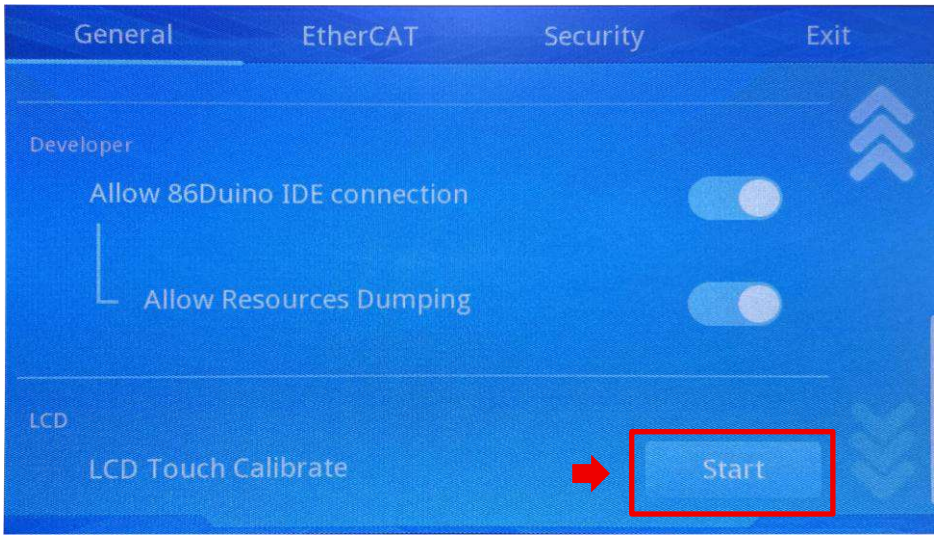
- **Allow 86Duino IDE connection**: for development purposes.
- **Allow Resources Dumping**: Enable or disable Resources Dumping for debugging.

4.6.2.3 LCD Option



- **LCD Touch Calibrate**: Start LCD Touch Calibrate function for your QEC MDevice.

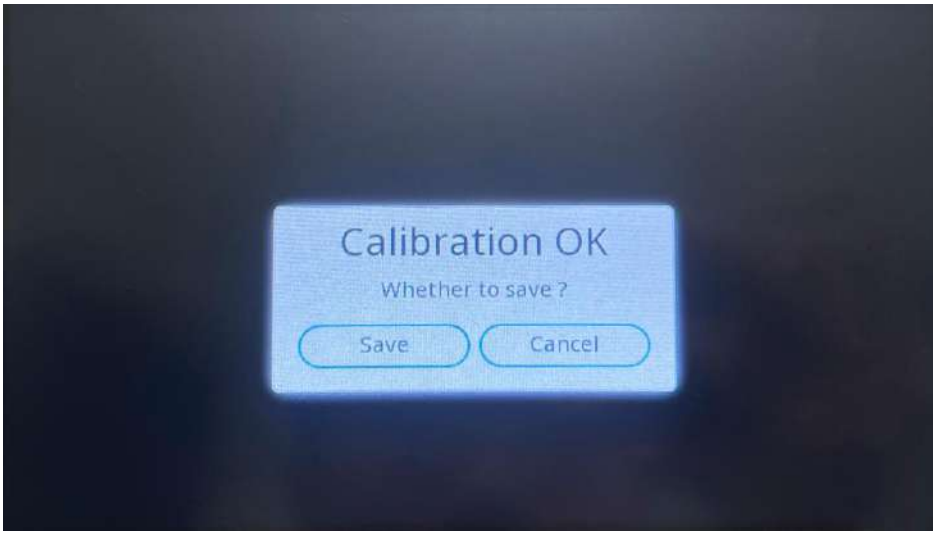
Click the "**Start**" button of LCD Touch Calibrate



It'll have four yellow circles displayed on the four corners by step. You have to touch the circles to calibrate the LCD touch.

After you finish the yellow circles, there is a yellow circle displayed in the center of the screen. Please also touch it.

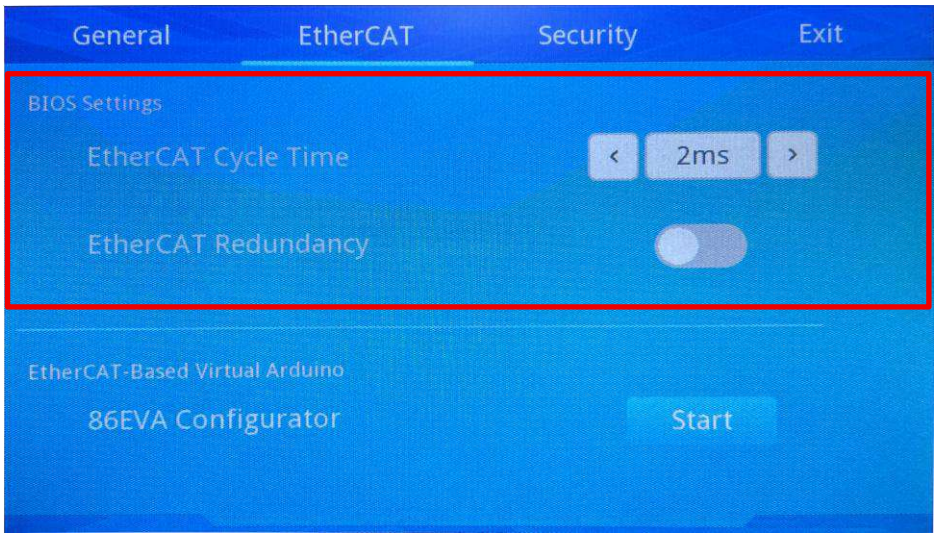
You have now finished the LCD touch Calibration. Please click “**Save**”, and you’ll return to the Bootloader Menu.



4.6.3 EtherCAT Page

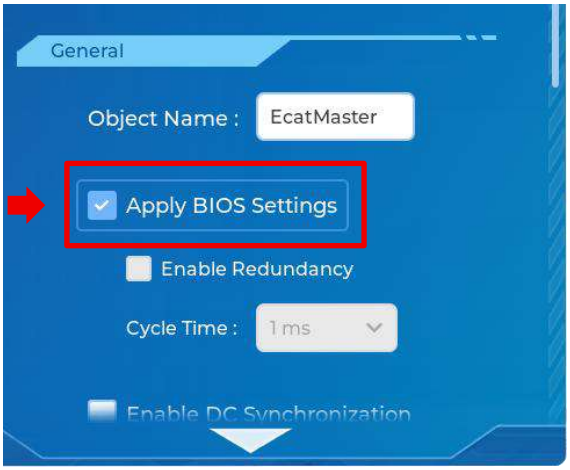
The EtherCAT Page allows users to configure EtherCAT settings and 86EVA Configurator, including Cycle Time and Redundancy, which are part of the BIOS defaults.

4.6.3.1 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.

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

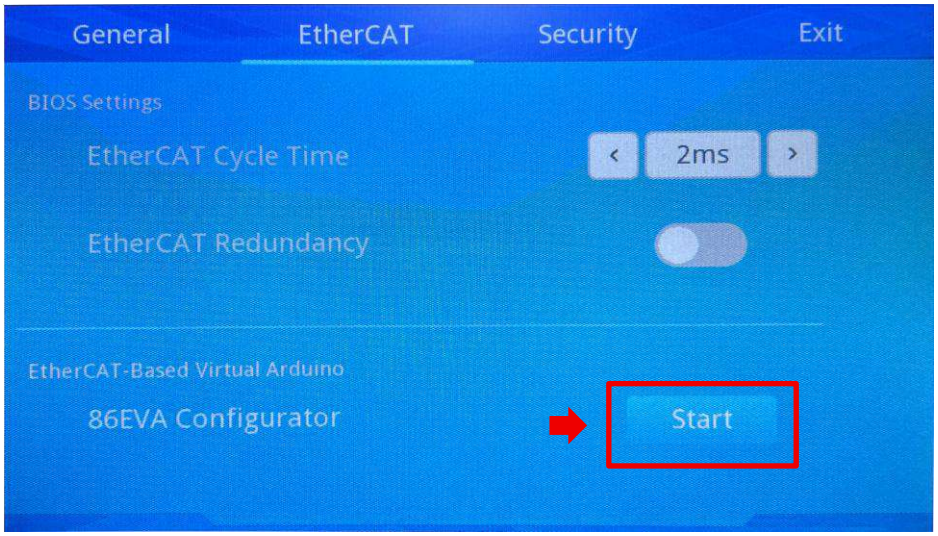


Enabling redundancy in the EtherCAT Page will automatically enable the Redundancy checkbox in the 86EVA Configurator, and setting the Cycle Time in the BIOS will apply it as the default value in the configurator.

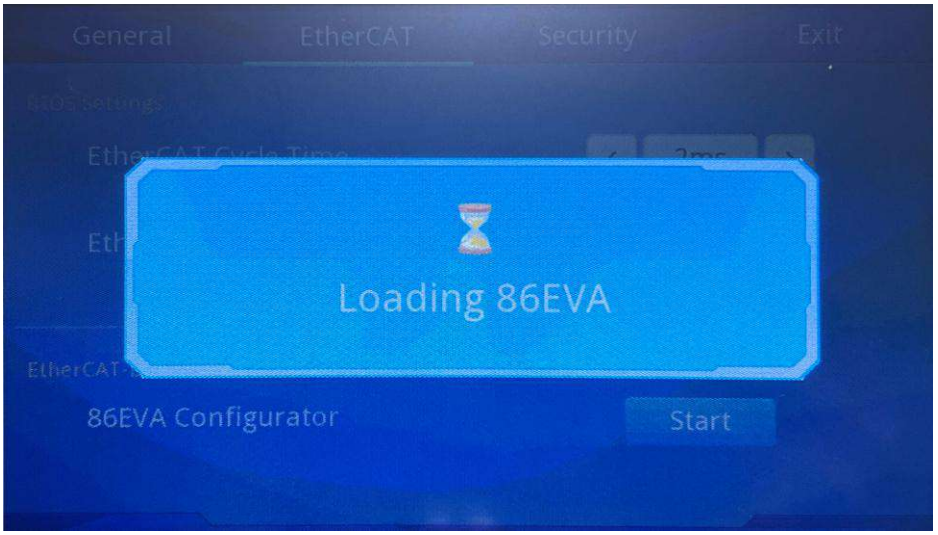
4.6.3.2 EtherCAT-Based Virtual Arduino

This function allows users to use 86EVA Configurator to scan and set the EtherCAT-Based Virtual Arduino mapping.

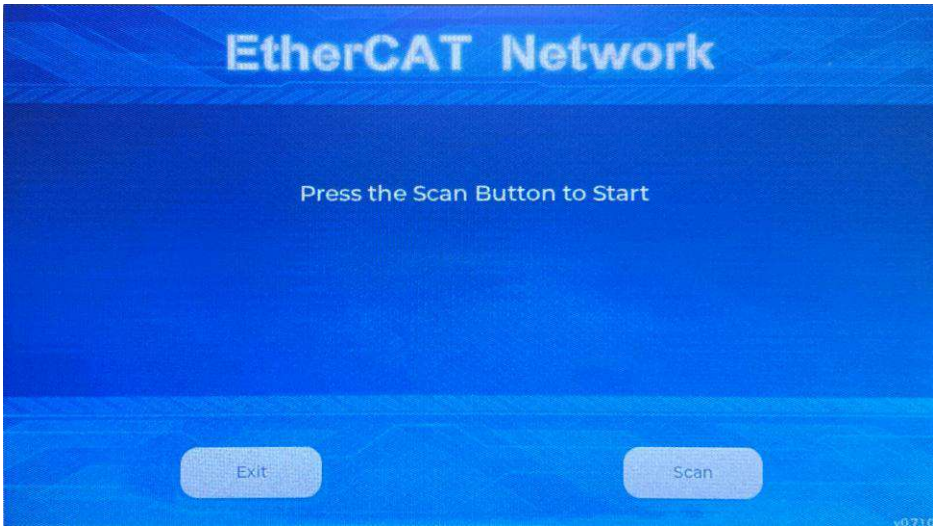
You can click the “**Start**” button to enter the 86EVA windows.



And you can click the "**Scan**" button to start the EtherCAT Network scanning.



86EVA on your QEC MDevice Open-frame series.

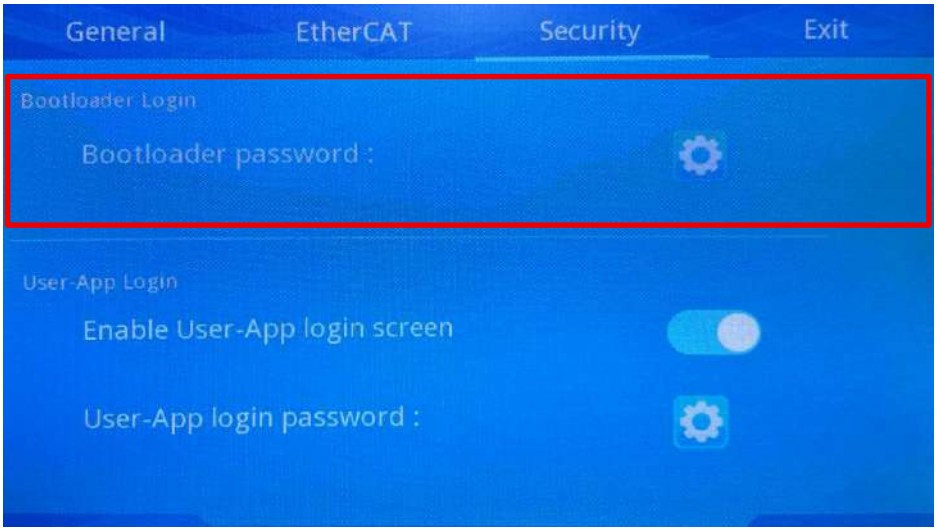


*__Note__: This 86EVA Configurator on the QEC MDevice Open-frame series is still under development. Do not suggest using.

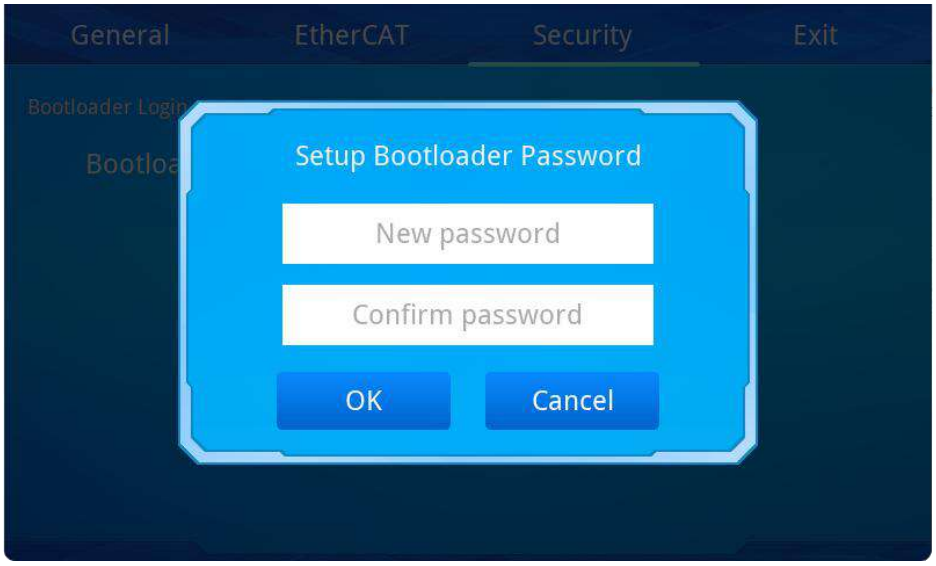
4.6.4 Security Page

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.

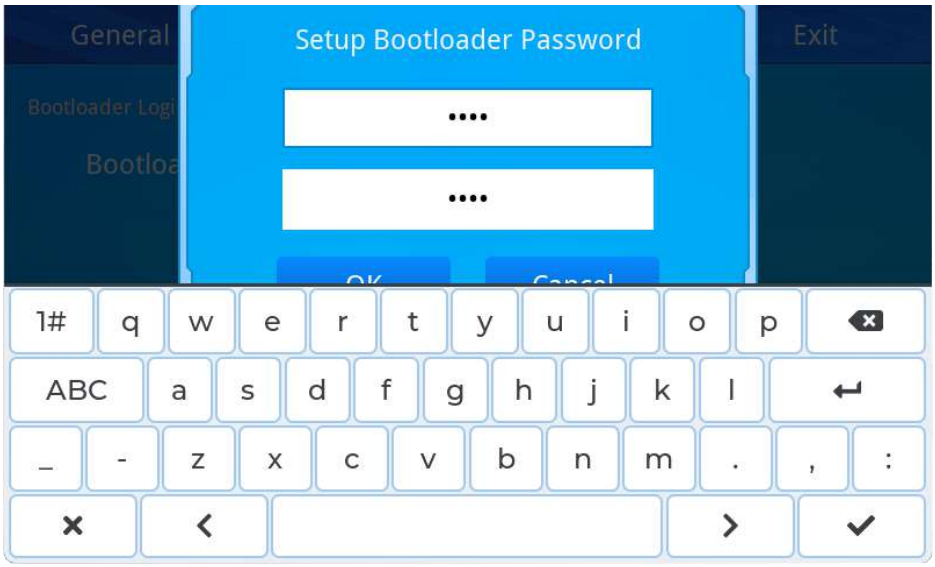
4.6.4.1 Bootloader Login



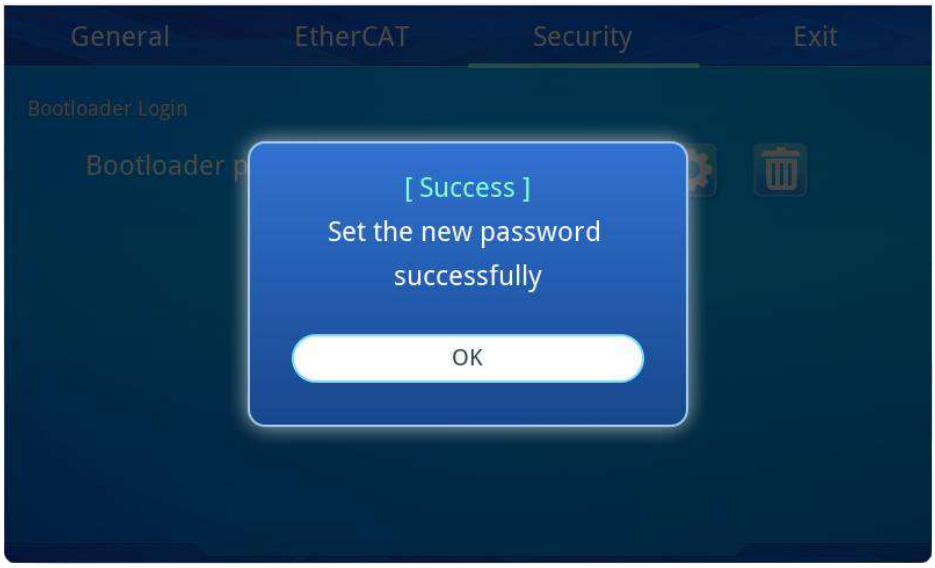
- **Bootloader password**: Users can click gear icon to open the "**Setup Bootloader Password**" window, as shown below.



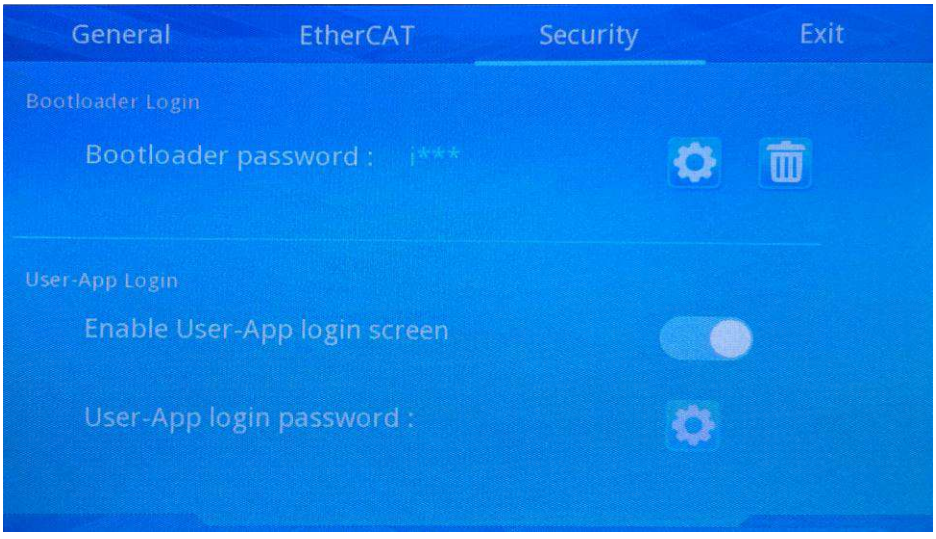
After clicking the text input area, the keyboard will appear on the windows.



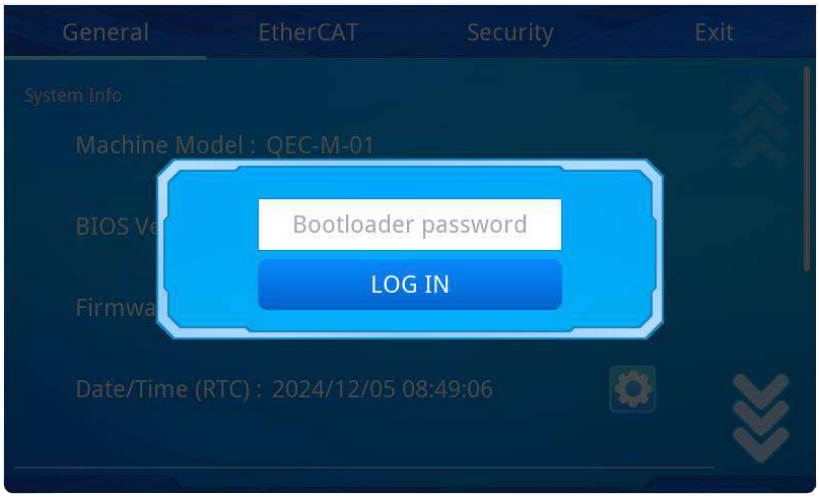
When users finish setting the bootloader password, click "**OK**". A **[Success]** window will appear to let the users know that the new password has been set successfully.



Then, the Security page will look like this.

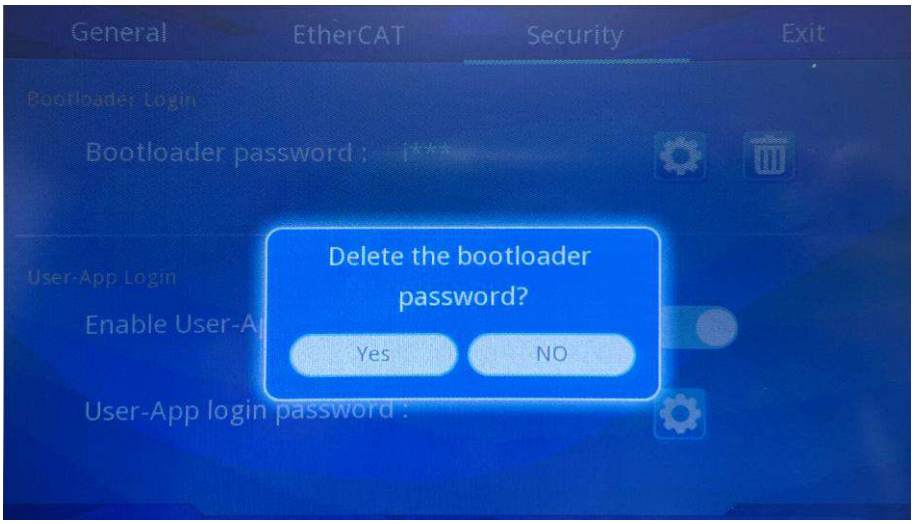


Once users save the bootloader menu configuration, the password will be saved, and the bootloader menu will show when users open it next time.



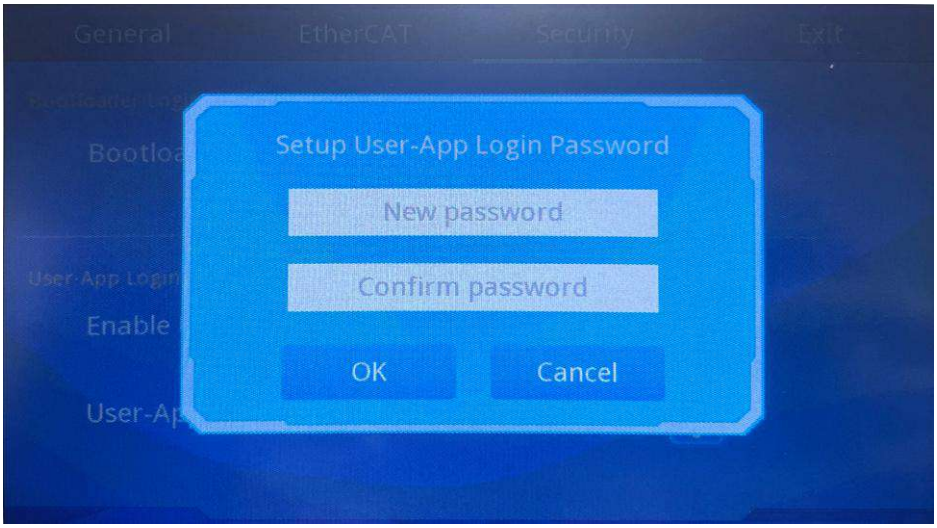
Users can click the gear icon to change the password and click the garbage can icon to delete the password.

Once you click the garbage can icon to delete the bootloader password, it'll display the following confirm windows.

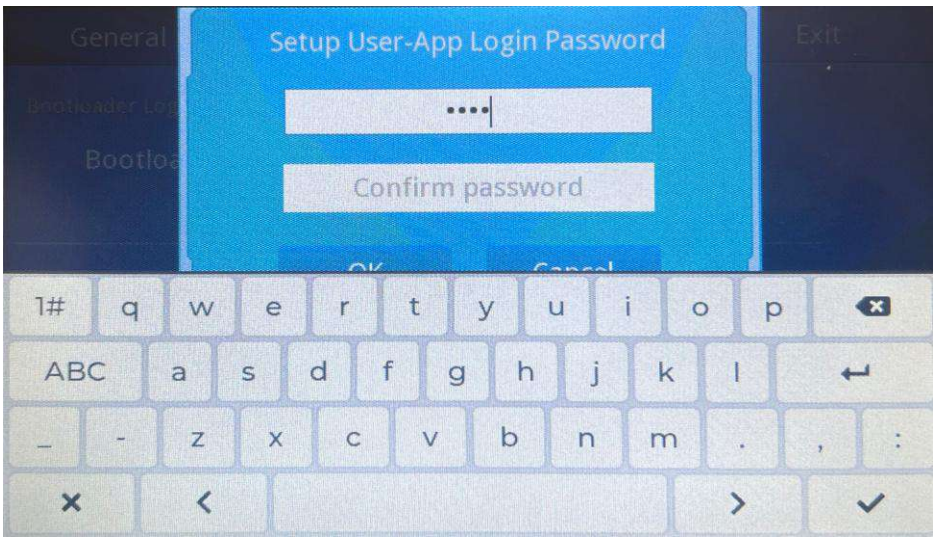


4.6.4.2 User-App Login

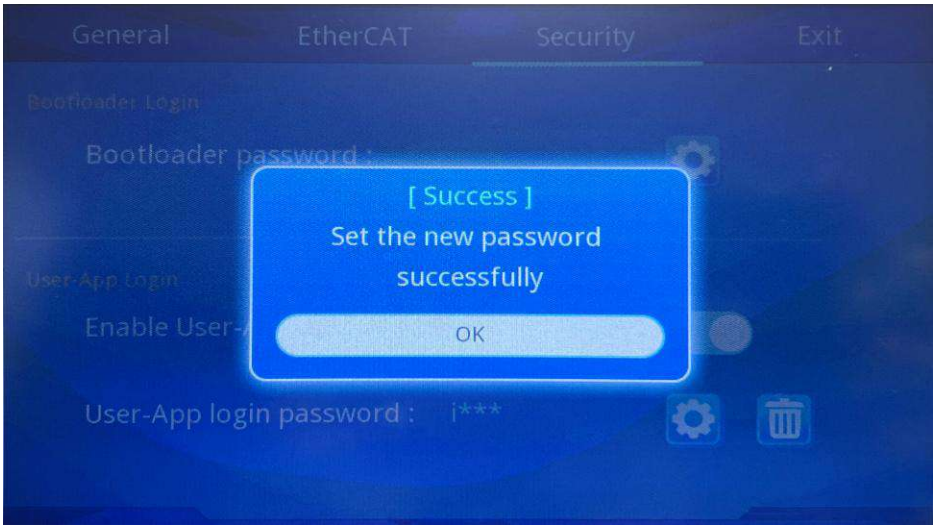
- **Enable User-App login screen**: If users select this option, a login screen appears before the user application operates in the QEC MDevice, provided a password has been set.
- **User-App login password:** Users can click gear icon to open the "**Setup Bootloader Password**" window, as shown below.



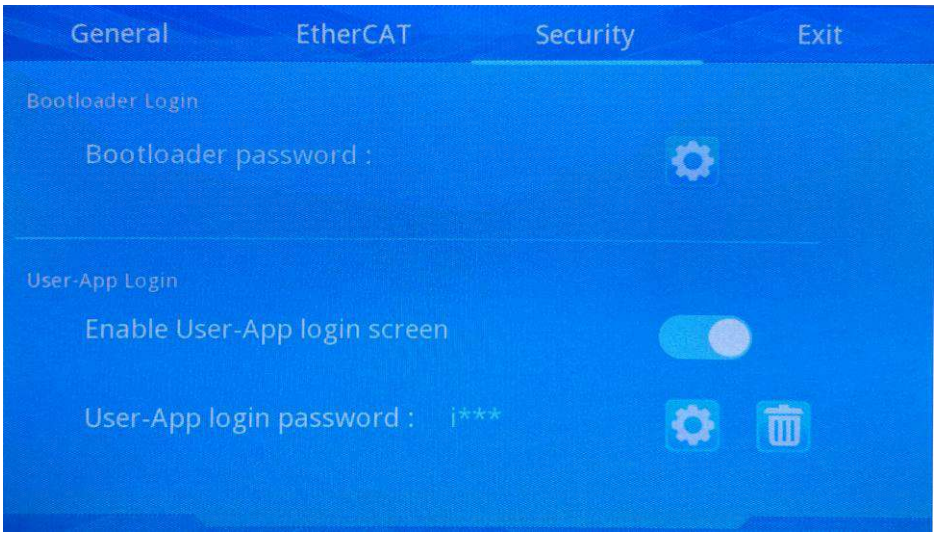
After clicking the text input area, the keyboard will appear on the windows.



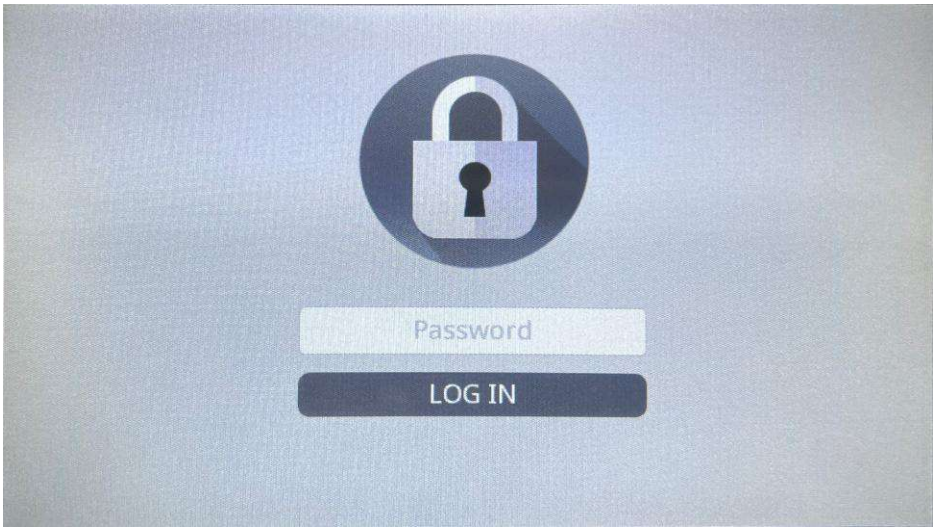
When users finish setting the User-App Login password, click "**OK**". A **[Success]** window will appear to let the users know that the new password has been set successfully.



Then, the Security page will look like this.

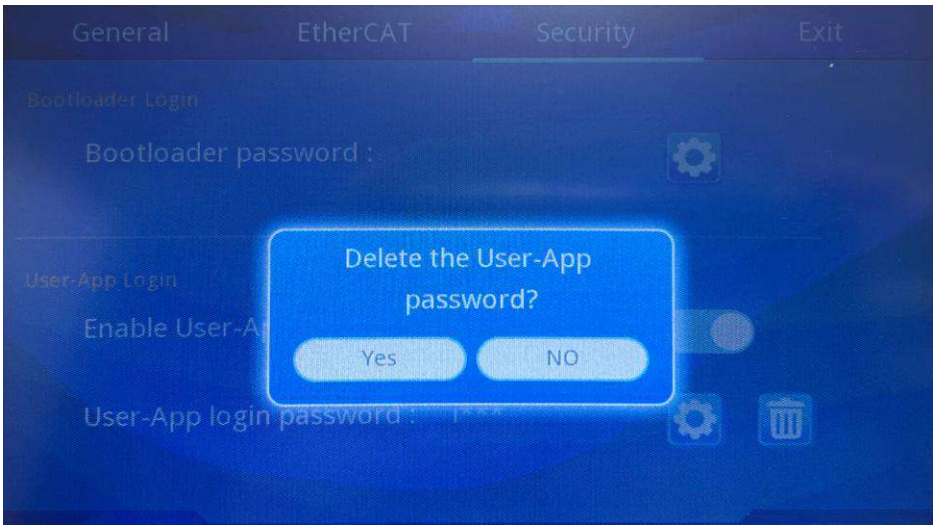


Once users save the bootloader menu configuration, the password will be saved, and the User-App Login screen will show when users open their QEC MDevice next time.
Like the below picture.



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

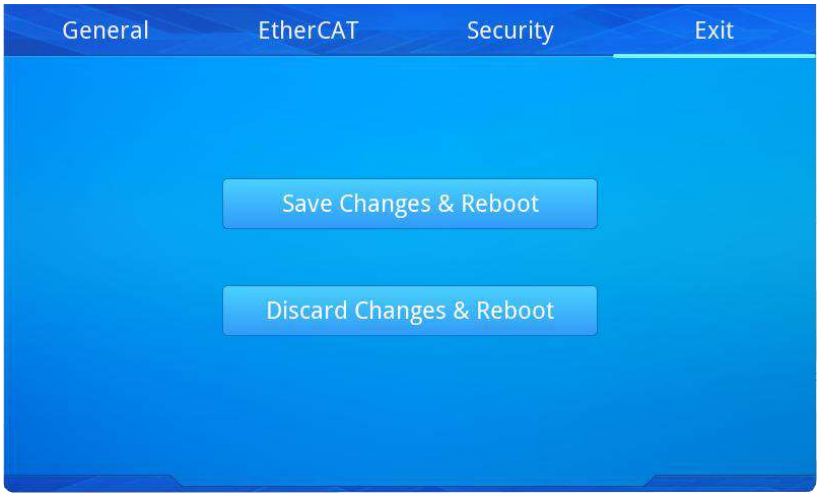
Once you click the garbage can icon to delete the bootloader password, it'll display the following confirm windows.



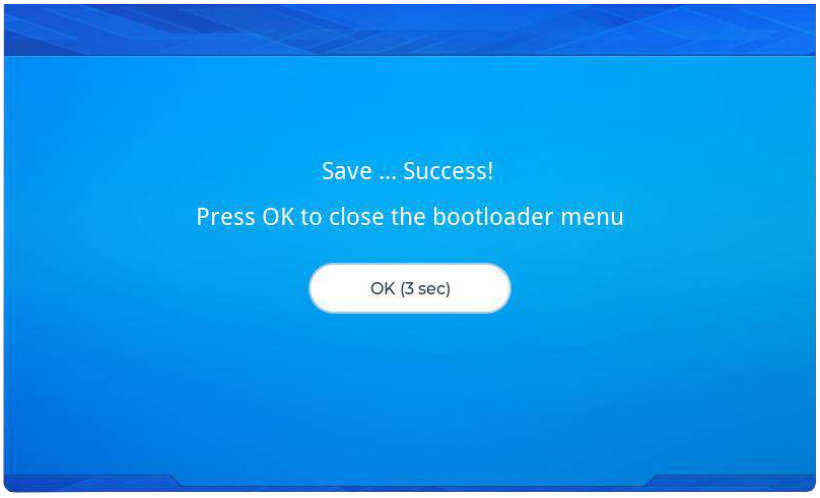
4.6.5 Exit Page

The Exit Page allows users to save or discard changes made in the Bootloader Menu before rebooting the QEC MDevice. 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.

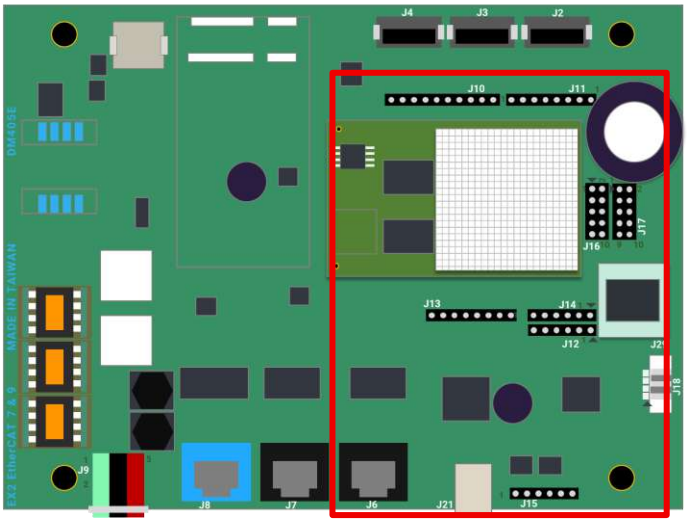
  Users can click OK to close the Bootloader Menu immediately, or wait for the automatic close in 5 seconds.

- **Discard Changes & Reboot:** Selecting this option will discard any unsaved changes and reboot the QEC MDevice with the previously saved configuration.

The Bootloader Menu will close directly without saving.

4.7 Arduino Pins Usage

This section introduces the GPIO pin configuration on the QEC MDevice, including both Arduino-compatible pins and QEC-exclusive functional pins such as CAN, SPI, and PWM.



The QEC system adopts the 86Duino architecture, which retains compatibility with Arduino's programming model, allowing users to write control programs using standard functions such as `digitalWrite()`, `analogRead()`, and `pinMode()`. However, some pins provide advanced functions unique to QEC, including hardware CAN bus, dedicated SPI interfaces, and extended PWM outputs, which are not available on standard Arduino boards.

To ensure correct usage, users must refer to the 86Duino pin mapping when writing software. Refer to the 86Duino pin configuration table below to ensure the correct mapping between physical pin locations and software pin numbers.

Table: 86Duino pin configuration

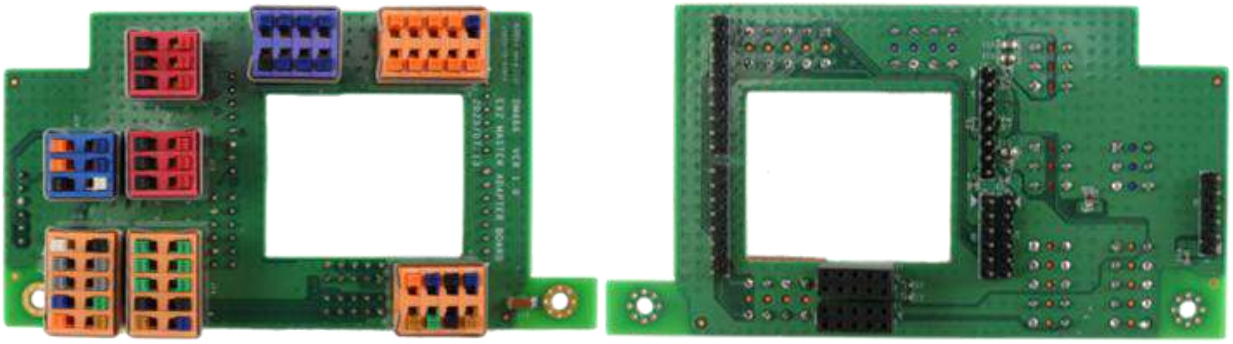
Connector	Pins	Signal	86Duino Pin Configuration
J11	1	RXD1#	0 (Serial1)
	2	TXD1#	1 (Serial1)
	3	GP00	2
	4	MCM-3	3
	5	GP02	4
	6	MCM-5	5
	7	MCM-6	6
	8	GP05	7
J10	1	GP30	8
	2	MCM-9	9
	3	MCM-10	10
	4	MCM-11	11

J10	5	GP31	12
	6	MCM-13	13
	7	GND	-
	8	-	-
	9	I2C0_SDA	(Wire)
	10	I2C0_SCL	(Wire)
J14	**6**	GP40ADC	14 (A0)
	5	GP41ADC	15 (A1)
	4	GP42ADC	16 (A2)
	3	GP43ADC	17 (A3)
	2	GP56ADC	18 (A4)
	1	GP57ADC	19 (A5)
J12	1	-	-
	2	GP35	23
	3	GP36	24
	4	GP37	25
	5	GND	-
	6	VCC	-
J15	1	CAN1_L	(CAN1)
	2	CAN1_H	(CAN1)
	3	GND	-
	4	CAN0_L	(CAN)
	5	CAN0_H	(CAN)
	6	VCC3	-
J16	1	SPI0_DI	(SPI)
	2	VCC	-
	3	SPI0_CLK	(SPI)
	4	SPI0_DO	(SPI)
	5	RESET-	-
	6	GND	-
	7	SPI0_CS	(SPI)
	8	-	-
	9	-	-
	10	-	-
J17	1	SPI1_DI	(SPI1)
	2	VCC	-
	3	SPI1_CLK	(SPI1)
	4	SPI1_DO	(SPI1)
	5	RESET-	-
	6	GND	-
	7	SPI1_CS	(SPI1)

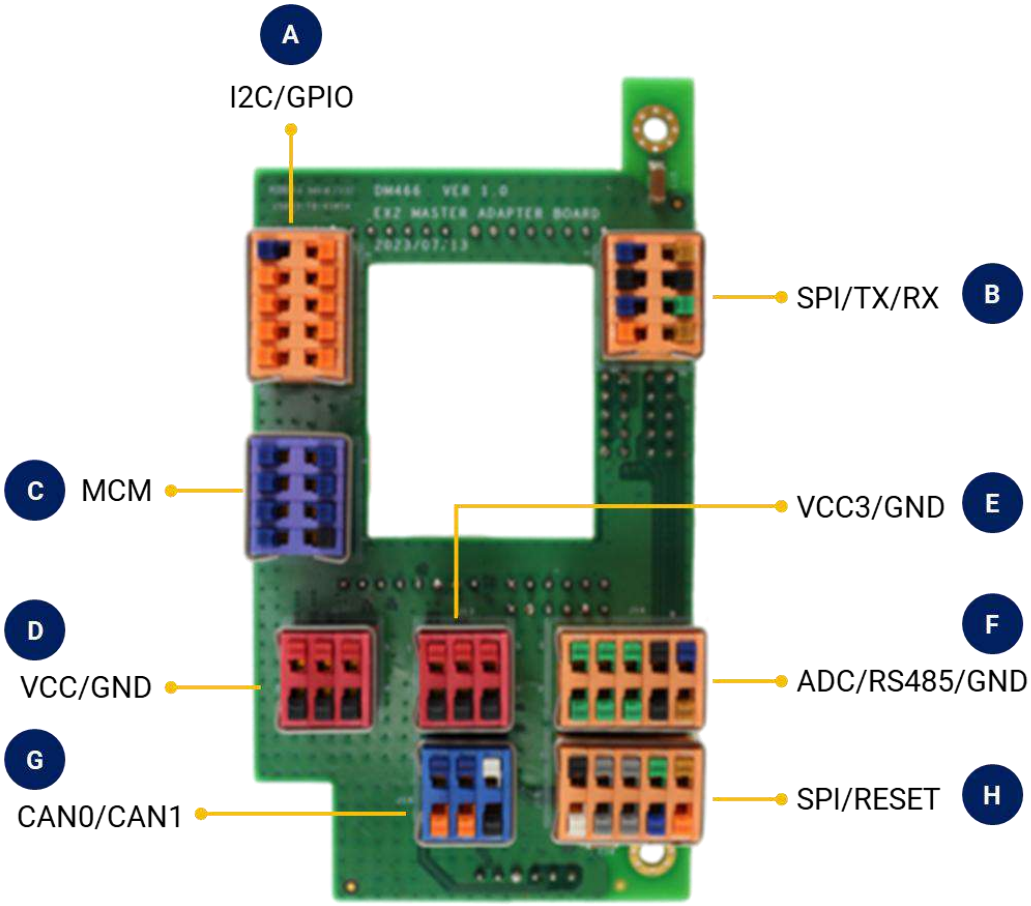
J17	8	-	-
	9	RS485+	(Serial485 / SerialCOM)
	10	RS485-	(Serial485 / SerialCOM)

4.7.1 Expansion Board: EC-TBV-ADAPT-KIT

For advanced application purposes, you can also purchase the following expansion board to transfer the Arduino pins from a female connector to a European-type connector.



4.7.1.1 Pin Definition



A. I2C/GPIO

Connector	Pins	Pins
A	I2C0_SCL	I2C0_SDA
	GP00	GP02
	GP05	GP30
	GP31	GP35
	GP36	GP37

B. SPI/TX/RX

Connector	Pins	Pins
B	TXD1	RXD1
	-	-
	SPI0_CS	SPI0_CLK
	SPI0_DO	SPI0_DI

C. MCM

Connector	Pins	Pins
C	MCM-3	MCM-5
	MCM-6	MCM-9
	MCM-10	MCM-11
	MCM-13	GND

D. VCC/GND

Connector	Pins	Pins	Pins
D	VCC	VCC	VCC
	GND	GND	GND

E. VCC3/GND

Connector	Pins	Pins	Pins
E	VCC3	VCC3	VCC3
	GND	GND	GND

F. ADC/RS485/GND

Connector	Pins	Pins	Pins	Pins	Pins
F	GP41ADC	GP43ADC	GP47ADC	GND	RS485+
	GP40ADC	GP42ADC	GP56ADC	GND	RS485-

G. CAN0/CAN1

Connector	Pins	Pins	Pins
G	CAN0_H	CAN1_H	RESET-
	CAN0_L	CAN1_L	GND

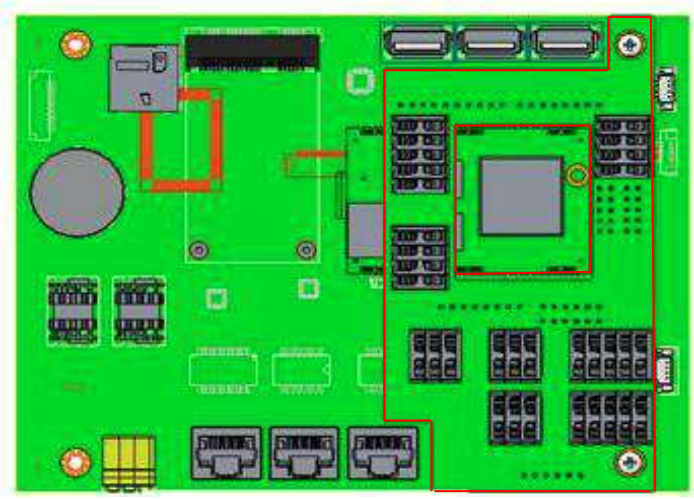
H. SPI/RESET

Connector	Pins	Pins	Pins	Pins	Pins
H	GND	-	-	SPI1_CLK	SPI1_DI
	RESET-	-	-	SPI1_CS	SPI1_DO

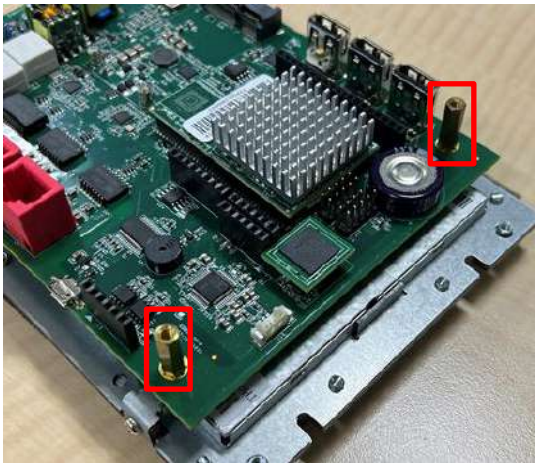
***Note:** To ensure correct usage, users must refer to the 86Duino pin mapping when writing software. Refer to the 86Duino pin configuration table to ensure the correct mapping between physical pin locations and software pin numbers.

4.7.1.2 Assembly Instructions

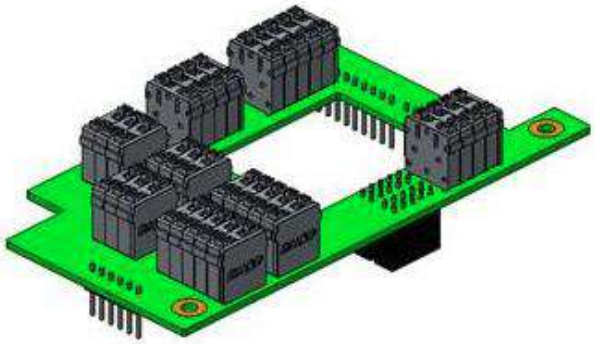
1. Place the QEC MDevice Open-frame face down with the monitor side facing downward, then position the EC-TBV-ADAPT-KIT on top of it as shown in the image.

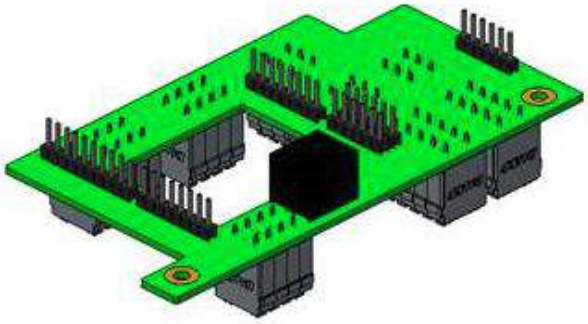


2. Insert two 1.5cm copper pillars into the screw holes to secure the baseboard and the screen.



3. Ensure that the PHX*XX (2.54)/MALE pins are properly aligned with the corresponding PHX*XX (2.54)/FEMALE connectors





4. Gently press the EC-TBV-ADAPT-KIT board until it is stable, then tighten the screws and copper pillars to fully secure the kit.

4.7.2 GPIO

The General Purpose Input/Output (GPIO) pins on the QEC MDevice can be programmed using familiar Arduino-style functions via the 86Duino IDE.

You can refer to the following API functions to learn more:

- pinMode()
- digitalWrite()
- digitalRead()

***Note:** Use the correct pin number from the 86Duino Pin Mapping Table.

Here is the example code:

```
int ledPin = 13;               // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                 // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                 // waits for a second
}
```

4.7.3 ADC

The QEC MDevice provides multiple Analog Input (ADC) channels (11-bit analog to digital converter). This means that it will map input voltages between 0 and 3.3 volts into integer values between 0 and 1023 (or 2047 by calling analogReadResolution() to set the 11-bit resolution). This yields a resolution between readings of: 3.3 volts / 1024 units or, .0032 volts (3.2 mV) per unit.

You can refer to the following API functions to learn more:

- analogRead()
- analogReference()

***Note:** Use the correct pin number from the 86Duino Pin Mapping Table.

Here is the example code:

```
int analogPin = 3;     // potentiometer wiper (middle terminal) connected to
analog pin 3
                      // outside leads to ground and +5V
int val = 0;           // variable to store the value read

void setup()
{
  Serial.begin(9600);          //  setup serial
}

void loop()
{
  val = analogRead(analogPin);    // read the input pin
  Serial.println(val);           // debug value
}
```

4.7.4 TX/RX

The QEC MDevice supports one TX/RX signal. It follows the standard UART protocol and can be accessed through the `Serial1` objects in the 86Duino IDE.

You can refer to the following API functions to learn more:

- Serial

*__Note:__ Use the correct pin number from the 86Duino Pin Mapping Table.

Here is the example code:

```
int incomingByte = 0;   // for incoming serial data

void setup() {
    Serial1.begin(9600);     // opens serial port, sets data rate to 9600 bps
}

void loop() {
    // send data only when you receive data:
    if (Serial1.available() > 0) {
            // read the incoming byte:
            incomingByte = Serial1.read();

            // say what you got:
            Serial1.print("I received: ");
            Serial1.println(incomingByte, DEC);
    }
}
```

4.7.5 RS-485

The QEC MDevice supports one RS-485 pin. It follows the standard RS-485 protocol and can be accessed through the `Serial485` or `SerialCOM` objects in the 86Duino IDE.

You can refer to the following API functions to learn more:

- Serial
- Modbus Library

*__Note:__ Use the correct pin number from the 86Duino Pin Mapping Table.

4.7.5.1 Serial Communication

Users can use `SerialCOM` to configure the serial port on the QEC MDevice Open-frame series.

Here is the example code:

```
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.

4.7.5.2 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

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

4.7.5.2.1 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
}
```

4.7.5.2.2 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
    }
}
```

4.7.6 CAN

The QEC MDevice is equipped with two CAN Bus interfaces, allowing direct communication with various industrial devices. These two sets of CAN Bus can be accessed through the `CAN` or `CAN1` objects in the 86Duino IDE.

With the 86Duino CANBus Library, users can easily send and receive CAN frames through the CAN object using familiar Arduino-style functions.

The CAN bus included in the 86Duino platform includes the following features:

- Support CAN 2.0A and 2.0B standards
- Support up to 1 Mbps data transmission speed
- Support standard (11-bit) and extended (29-bit) data/remote frame transmission
- Provide 3 set of frame buffers
- Extensive error detection and handling mechanism

To simplify the effort to learn and become familiar with this library, the API design for this library is closely resembling the Wire library and CAN bus library from Seeed Studio. Following are description for the function and usage for the API in this library.

You can refer to the following API functions to learn more:

- CANBus Library

*__Note:__ Use the correct pin number from the 86Duino Pin Mapping Table.

Here is the example code:

```
#include <CANBus.h>
unsigned char buf[8] = {0, 1, 2, 3, 4, 5, 6, 7};

void setup() {
    Serial.begin(115200);
    CAN.begin(CAN_500KBPS); // Configure CAN bus transmission speed to
500KBPS
}

void loop() {
    CAN.beginTransmission(0x00, CAN_STDID);    // Set the CAN device ID to
0x00, using standard data grame.
    CAN.write(buf, 8);                        // Sent 8 bytes of data
    CAN.endTransmission();                    // End transmission
    delay(10);
}
```

4.7.7 SPI

The QEC MDevice is equipped with two hardware SPI interfaces, which can be accessed via the `SPI` and `SPI1` objects in the 86Duino IDE. These interfaces enable high-speed, full-duplex communication with external peripherals such as ADCs, DACs, flash memory, display modules, and more.

You can refer to the following API functions to learn more:

- SPI library

*__Note:__ Use the correct pin number from the 86Duino Pin Mapping Table.

Here is the example code:

```
// inslude the SPI library:
#include <SPI.h>

// set pin 10 as the chip select for the digital pot:
const int chipSelectPin = 10;

void setup() {
  // set the chipSelectPin as an output:
  pinMode(chipSelectPin, OUTPUT);

  // initialize SPI:
  SPI.begin();
}

void loop() {
  // go through the six channels of the digital pot:
  for (int channel = 0; channel < 6; channel++) {
    // change the resistance on this channel from min to max:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, level);
      delay(10);
    }
    // wait a second at the top:
    delay(100);

    // change the resistance on this channel from max to min:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, 255 - level);
```

```
      delay(10);
    }
  }
}

void digitalPotWrite(int address, int value) {
  // take the SS pin low to select the chip:
  digitalWrite(chipSelectPin, LOW);
  delay(100);

  //  send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  delay(100);

  // take the SS pin high to de-select the chip:
  digitalWrite(chipSelectPin, HIGH);
}
```

4.7.8 I2C

The QEC MDevice provides an I2C (Inter-Integrated Circuit) interface through the Wire object, enabling two-wire communication with devices such as sensors, EEPROMs, RTC modules, and I/O expanders. This library allows you to communicate with I2C / TWI devices via the SDA (data line) and SCL (clock line) pins.

You can refer to the following API functions to learn more:

- Wire library

*__Note:__ Use the correct pin number from the 86Duino Pin Mapping Table.

Here is the example code:

```
#include <Wire.h>

byte val = 0;

void setup()
{
  Wire.begin(); // join i2c bus
}

void loop()
{
  Wire.beginTransmission(44); // transmit to device #44 (0x2c)
                              // device address is specified in datasheet
  Wire.write(val);              // sends value byte
  Wire.endTransmission();     // stop transmitting

  val++;        // increment value
  if(val == 64) // if reached 64th position (max)
  {
    val = 0;    // start over from lowest value
  }
  delay(500);
}
```

4.7.9 MCM

The **Multi-Channel PWM Module (MCM)** is a key hardware feature of the QEC MDevice that enables efficient and flexible generation of multiple **PWM (Pulse Width Modulation)** signals. It serves as the foundation of the `analogWrite()` function in the 86Duino IDE.

MCM is particularly useful in applications that require:

- Motor speed control
- LED dimming
- Servo signal generation
- General analog output emulation

In our implementation of the `analogWrite()` function with a 1000Hz PWM frequency, you can set the write resolution up to 13. And by setting the write resolution to 13, you can use `analogWrite()` with values between 0 and 8191 to set the PWM signal without rolling over.

You can refer to the following API functions to learn more:

- analogWrite()
- analogWriteResolution()

*__Note:__ Use the correct pin number from the 86Duino Pin Mapping Table.

```
void setup(){
  // open a serial connection
  Serial.begin(9600);
  // make our digital pin an output
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
}

void loop(){
  // read the input on A0 and map it to a PWM pin
  // with an attached LED
  int sensorVal = analogRead(A0);
  Serial.print("Analog Read) : ");
  Serial.print(sensorVal);

  // the default PWM resolution
  analogWriteResolution(8);
  analogWrite(11, map(sensorVal, 0, 1023, 0 ,255));
```

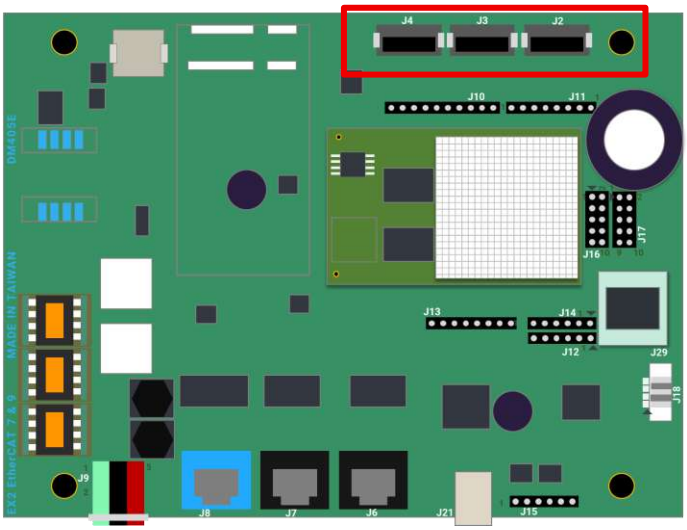
```
  Serial.print(" , 8-bit PWM value : ");
  Serial.print(map(sensorVal, 0, 1023, 0 ,255));

  // change the PWM resolution to 12 bits
  // the full 12 bit resolution is only supported
  // on the Due
  analogWriteResolution(12);
  analogWrite(12, map(sensorVal, 0, 1023, 0, 4095));
  Serial.print(" , 12-bit PWM value : ");
  Serial.print(map(sensorVal, 0, 1023, 0, 4095));

  // change the PWM resolution to 4 bits
  analogWriteResolution(4);
  analogWrite(13, map(sensorVal, 0, 1023, 0, 127));
  Serial.print(", 4-bit PWM value : ");
  Serial.println(map(sensorVal, 0, 1023, 0, 127));

  delay(5);
}
```

***Note:** If you set the `analogWriteResolution()` value to a value higher than the allowed capabilities, our implementation will *discard* the extra bits. For example: using the 86Duino with `analogWriteResolution(16)`, only the first 13 bits of the values passed to `analogWrite()` will be used and the last 3 bits will be discarded.
If you set the `analogWriteResolution()` value to a value lower than the allowed capabilities, the missing bits will be padded with zeros to fill the required size. For example: using the 86Duino with `analogWriteResolution(8)`, the 86Duino will add 5 zero bits to the 8-bit value used in `analogWrite()` to obtain the 13 bits required.

4.8 USB Device Usage

This section ensures the proper use of USB devices and file storage on the QEC MDevice Open-frame series.

The QEC MDevice Open-frame series features 3 Standard USB 2.0 ports 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 MDevice Open-frame series 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.

4.8.1 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 your QEC MDevice Open-frame.

4.8.2 Example2: Save .txt in EMMC storage

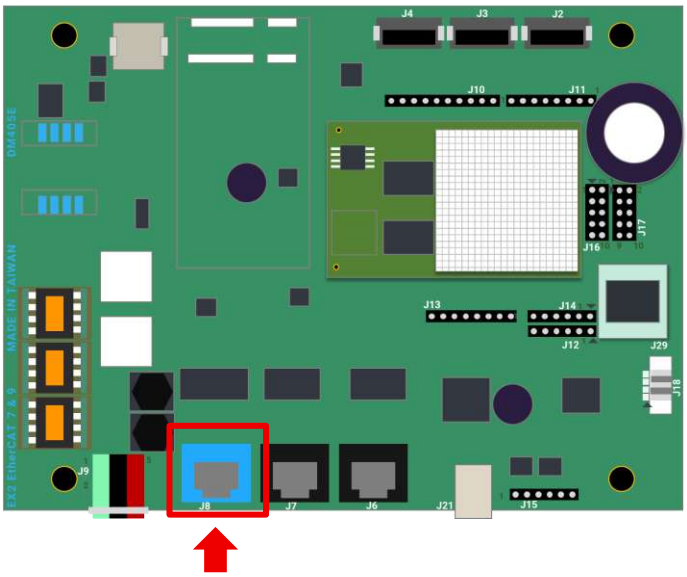
Below is an example of creating a file using `fopen()` in C language. We save a .txt file on an **internal EMMC storage**. This demonstrates specifying the file location and ensuring safe storage in the appropriate slot.

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 MDevice Open-frame series features one Giga LAN port dedicated to external Ethernet communication for general network use.



***Note:** The Giga LAN on the QEC MDevice Open-frame is in blue housing.

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.

4.9.1.1 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);

  // 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() {

}
```

4.9.1.2 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");
  }
}
```

4.9.1.3 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);
  }
}
```

4.9.1.4 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.

4.9.2.1 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 HMI Design

This section demonstrates how to build a basic Human-Machine Interface (HMI) on the QEC MDevice (Open-frame Series), using either the LVGL library or the 86HMI Editor included 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 Open-frame MDevice.

4.10.1 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.

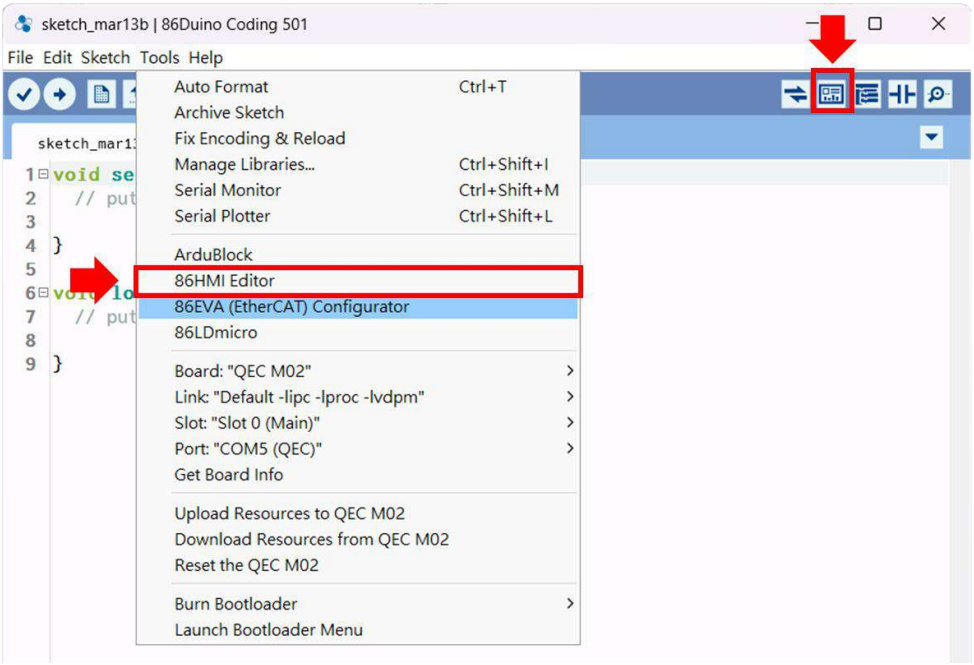
***Note:** 86Duino IDE 501+ has already tested the **LVGL version 9.3**, so users can import the LVGL library manually.

4.10.2 Using the Graphical HMI editor: 86HMI Editor

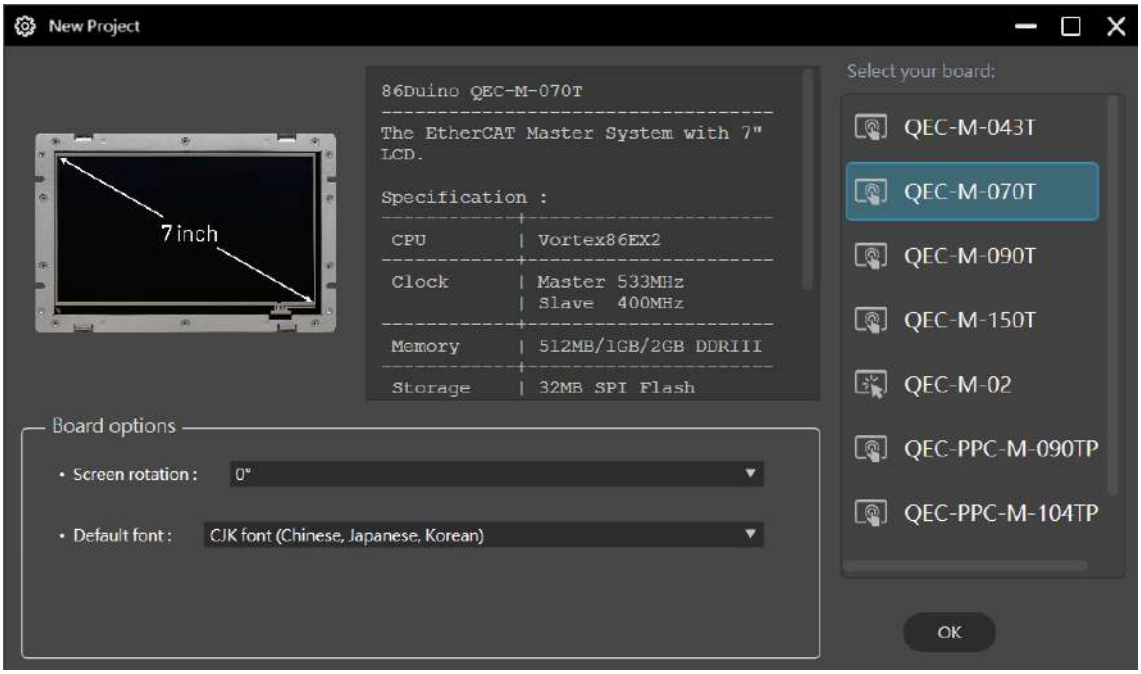
86HMI Editor is an easy-to-use HMI editor based-on LVGL version 7.11, that can be used to create a customized HMI quickly. Use the Auto Code Generation function in 86HMI Editor to generate HMI APIs (Application Programming Interface), thus achieving the effect of creating HMIs without writing programs.

Below are the complete steps to design UI to QEC MDevice using 86HMI:

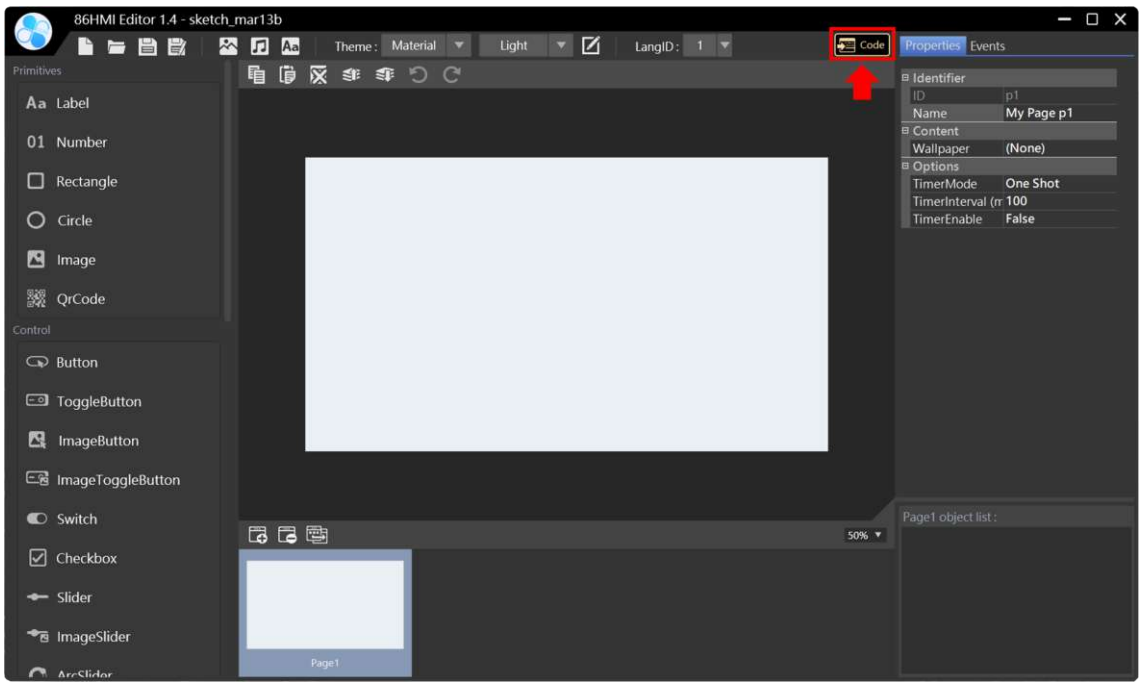
1. Open 86HMI tool via 86Duino Coding IDE 501+.



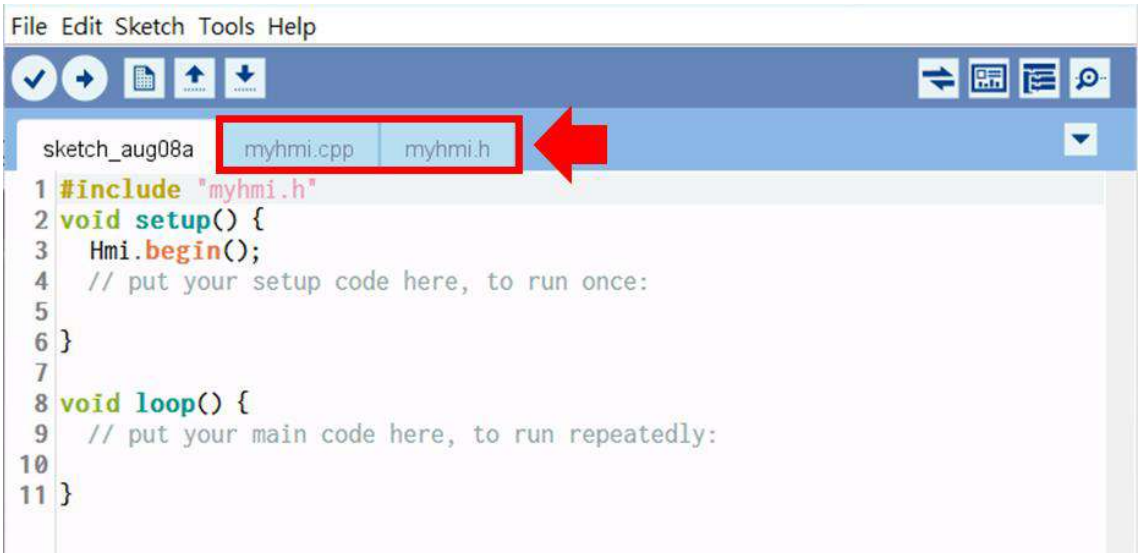
2. Choose your QEC MDevice Open-frame.



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 MDevice.

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

Ch. 5

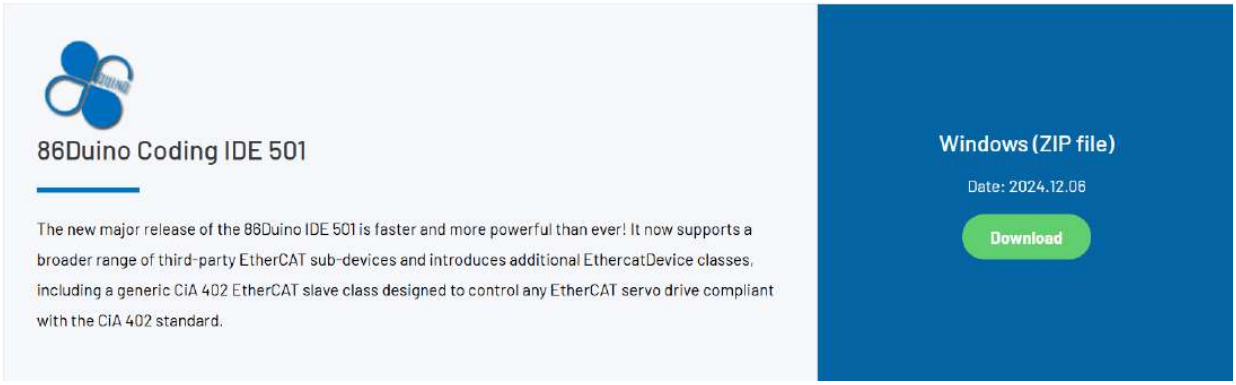
Software Function

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.

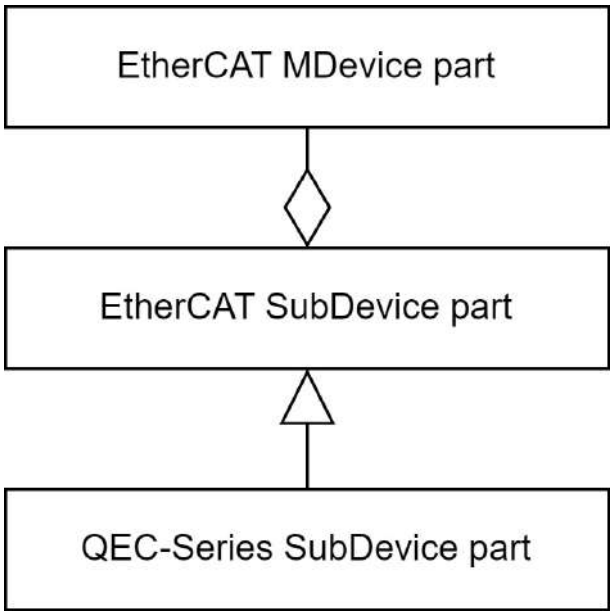
Please visit qec.tw for 86Duino Coding IDE 501+ details.



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.

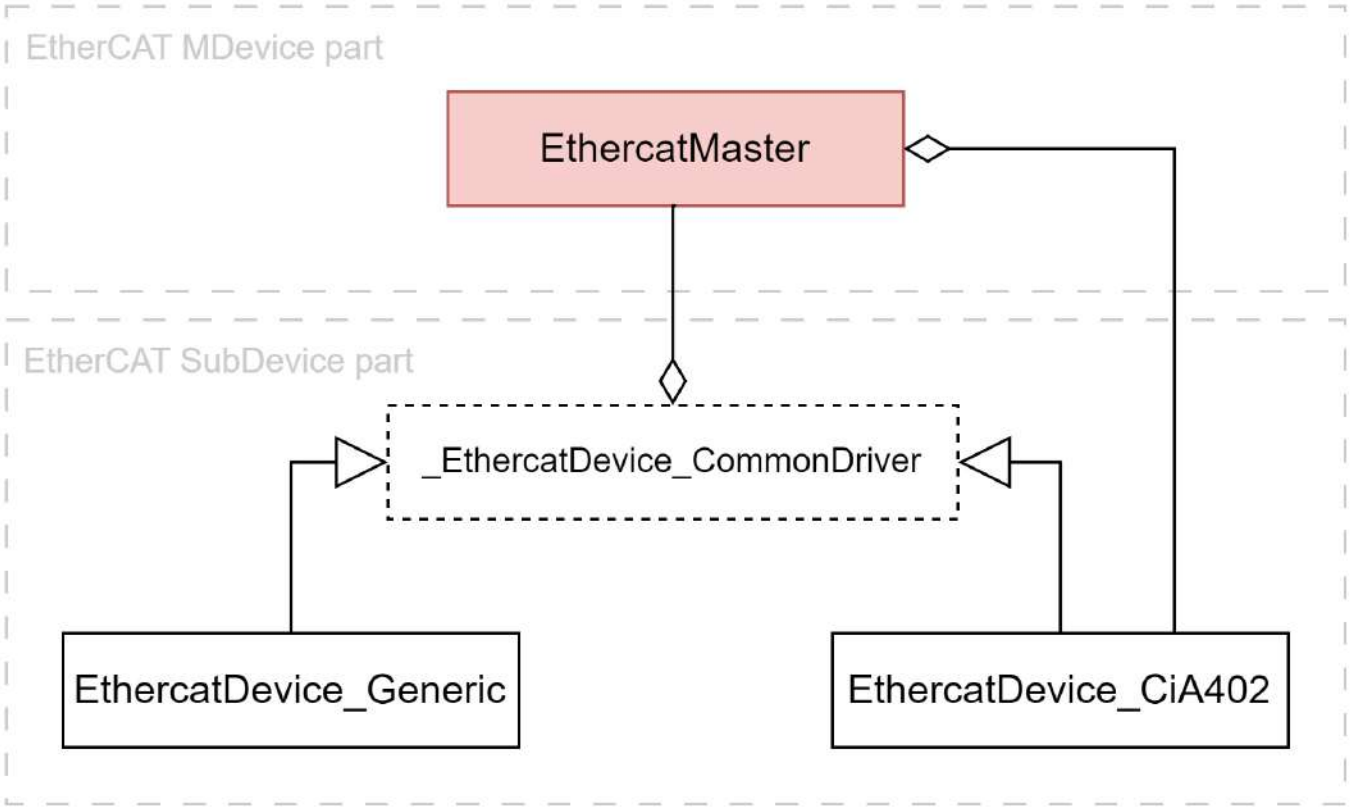
The list below introduces the EtherCAT Library API functions of our QEC MDevice.
Please visit the EtherCAT Library API User Manual for details on the API Function.

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:

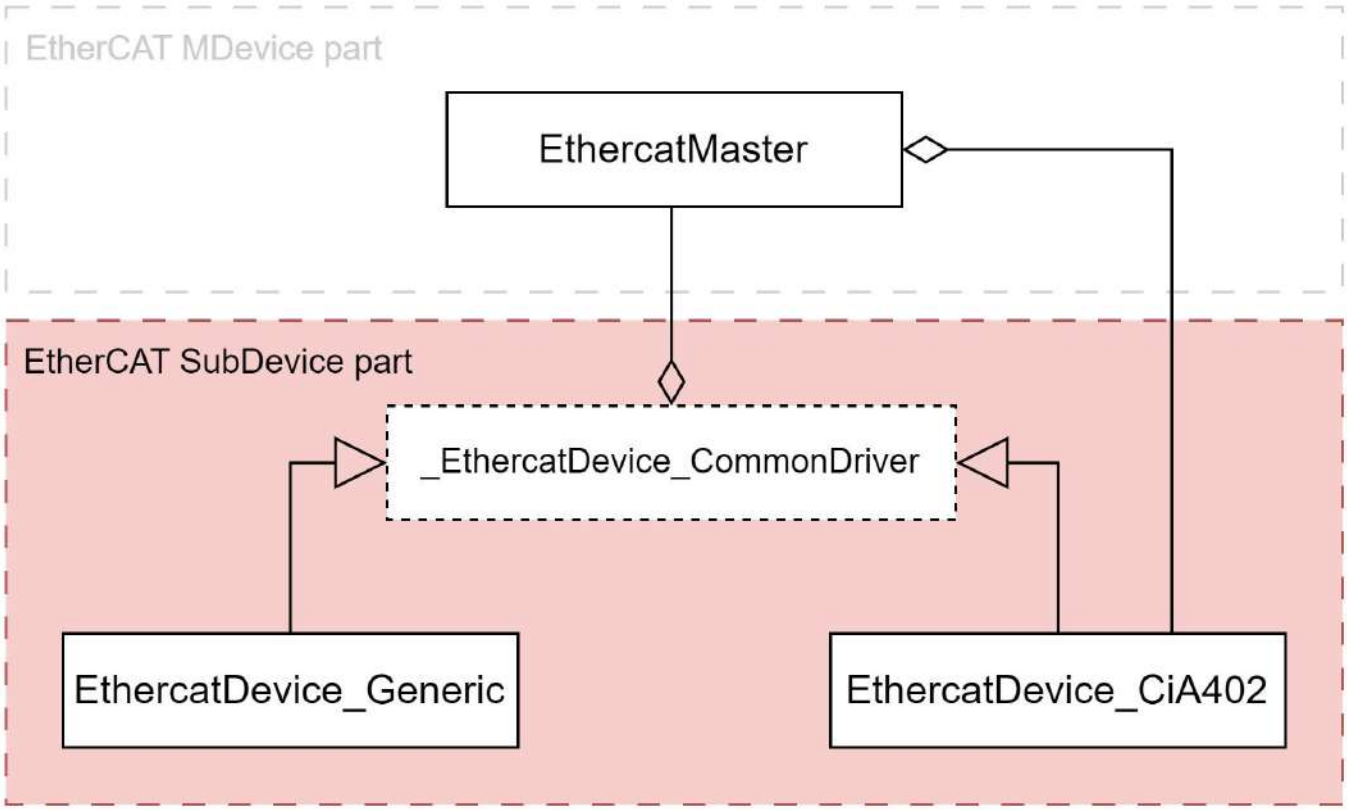
Function Name	Description	Callback Available
Initialization-related functions		
begin()	Initialize the EtherCAT MDevice.	
end()	Deinitialize the EtherCAT MDevice.	
isRedundancy()	Check if the EtherCAT MDevice has cable redundancy enabled.	O
libraryVersion()	Get the EtherCAT MDevice library version.	O
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.



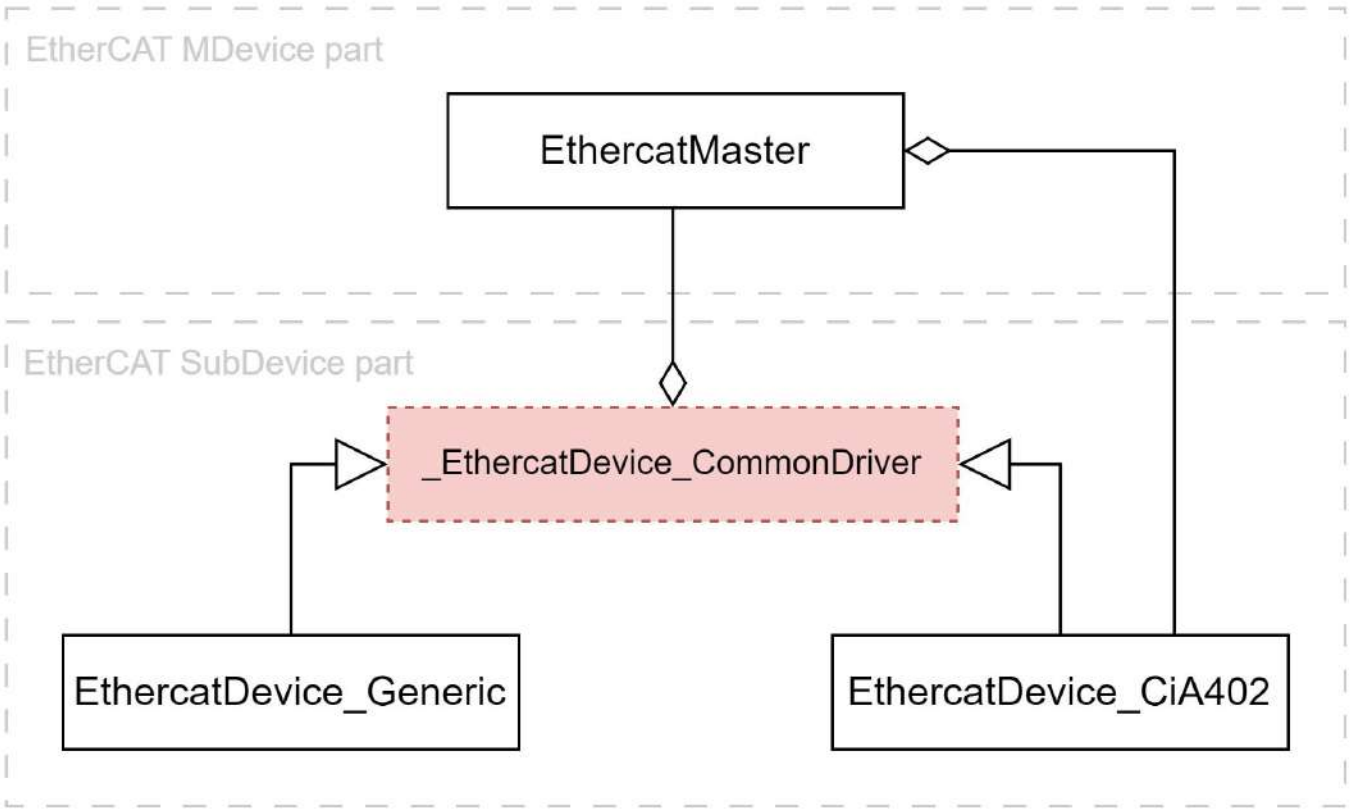
Classes:

- _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

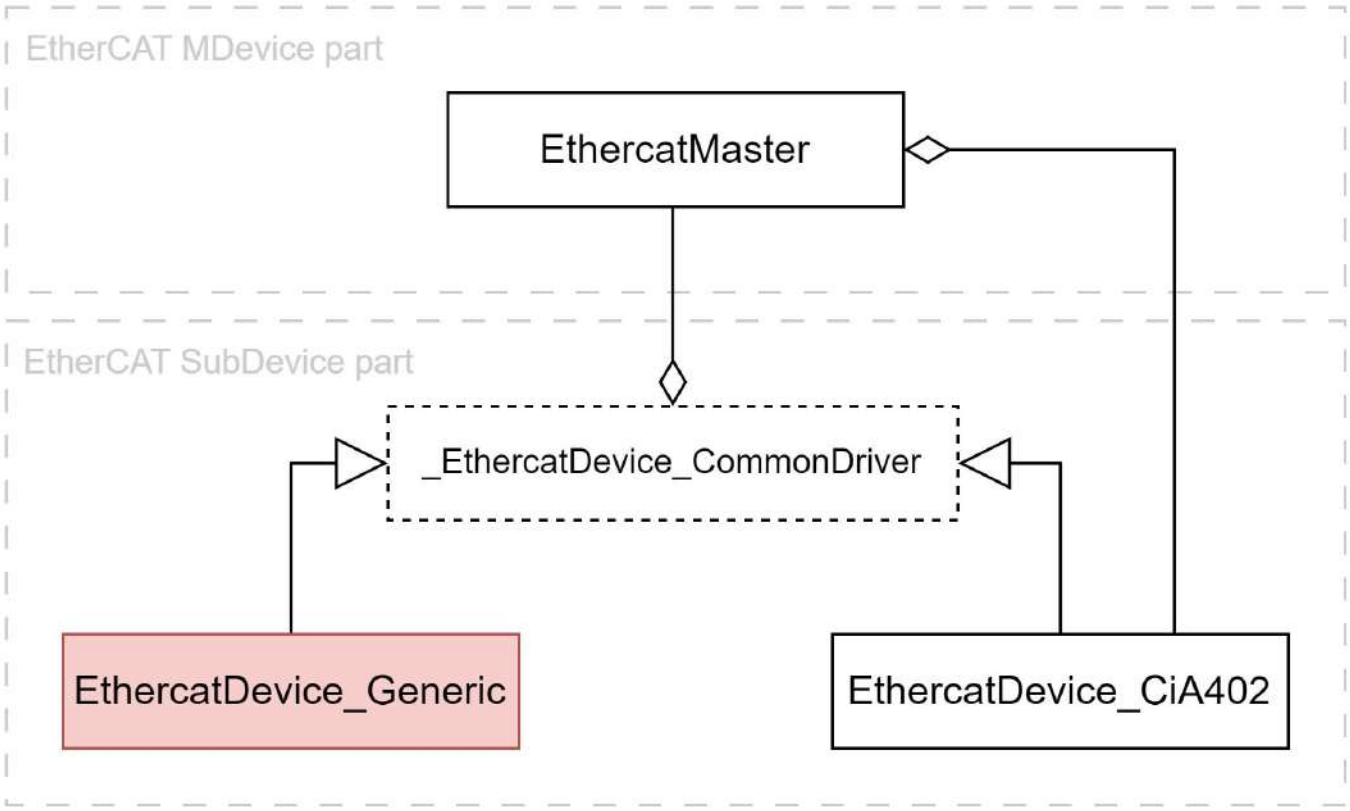
getMailboxProtocol()	Get the supported mailbox protocol types.	O
getCoEDetails()	Get the details about CoE supported.	O
getFoEDetails()	Get the details about FoE supported.	O
getEoEDetails()	Get the details about EoE supported.	O
getSoEChannels()	Get the number of SoE channels supported.	O
isSupportDC()	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

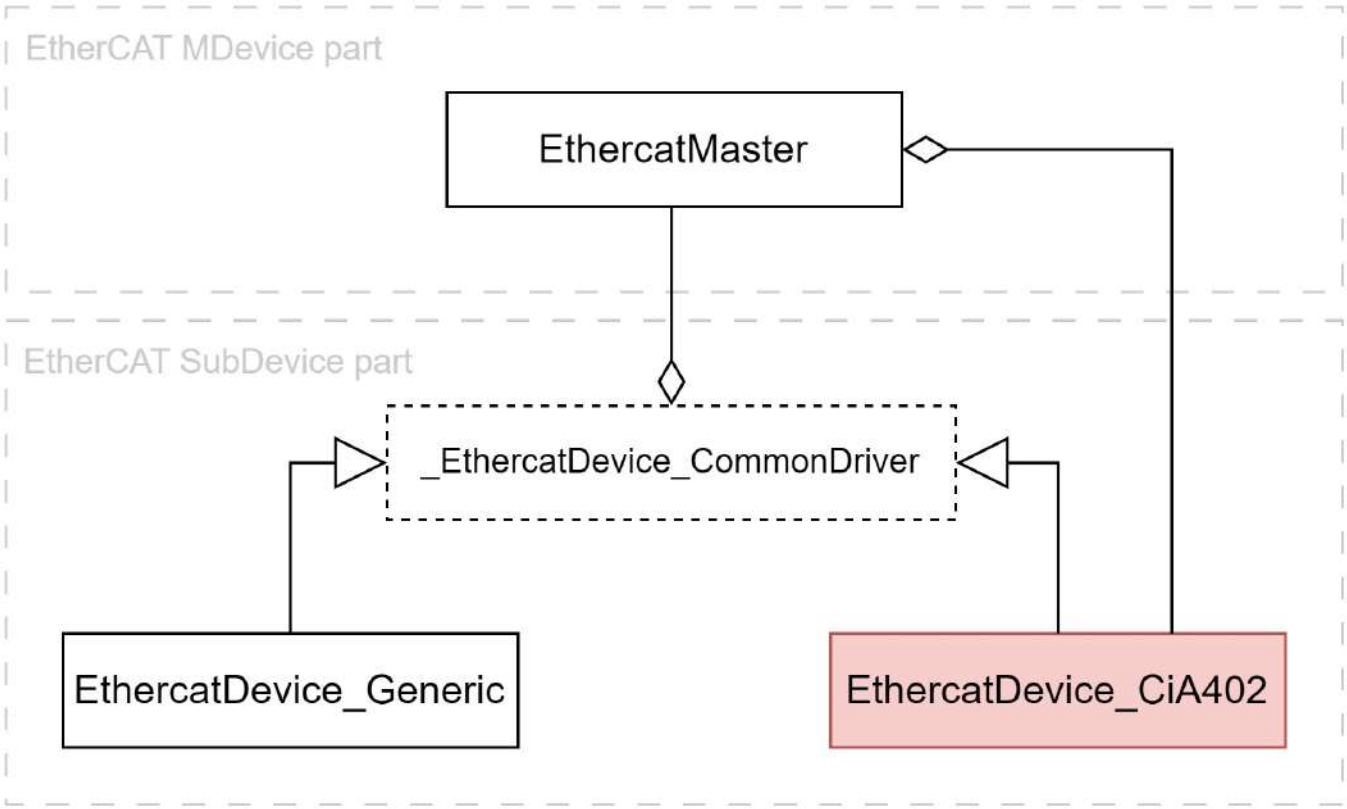
EthercatDevice_CiA402 is a generic CiA 402 EtherCAT SubDevice class designed to control any EtherCAT servo drive that supports the CiA 402 standard. It provides access functions for commonly used CiA 402 objects and operation functions for several CiA 402 operation modes and function groups, including:

- **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

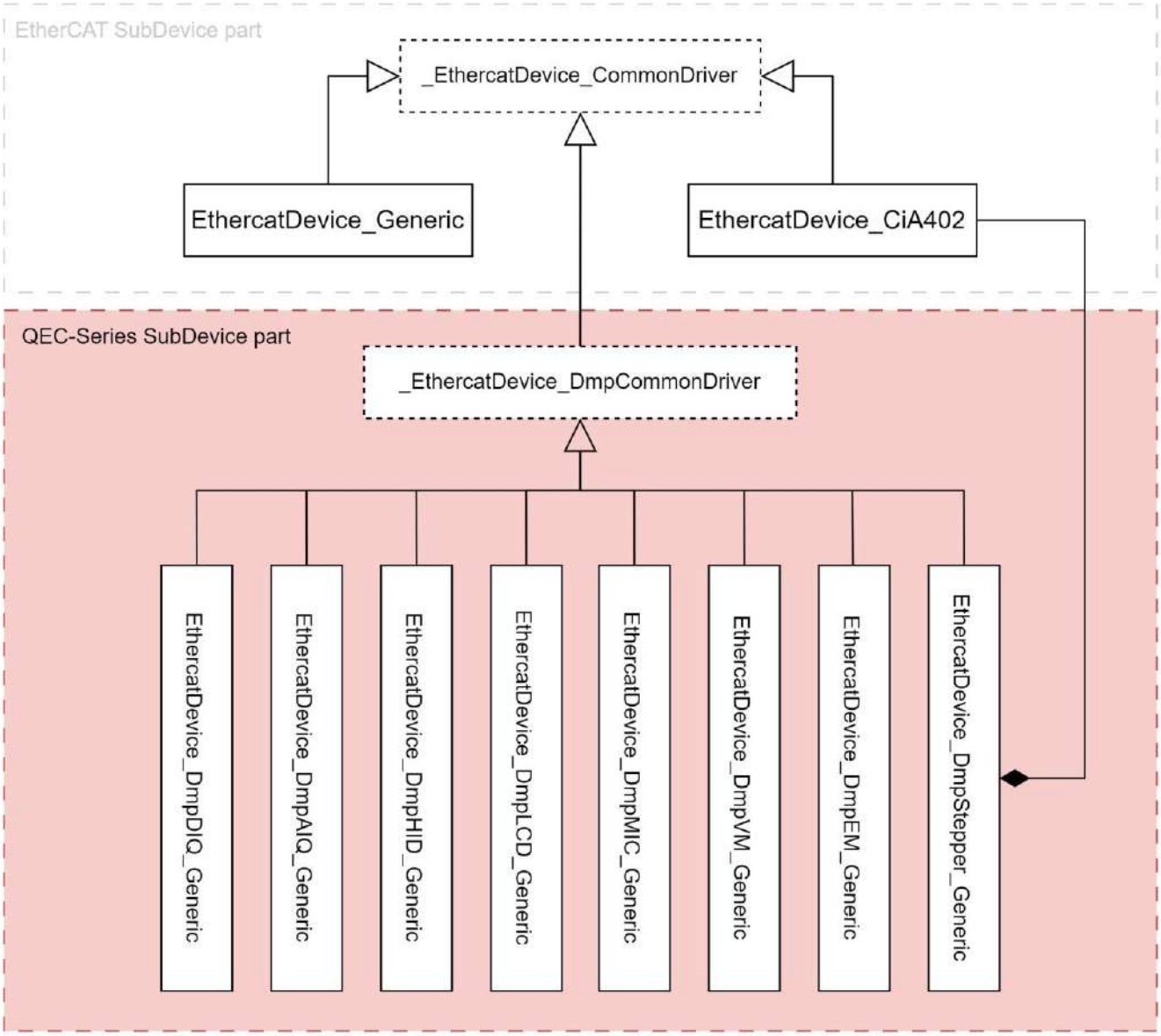
- _EthercatDevice_CommonDriver

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/.

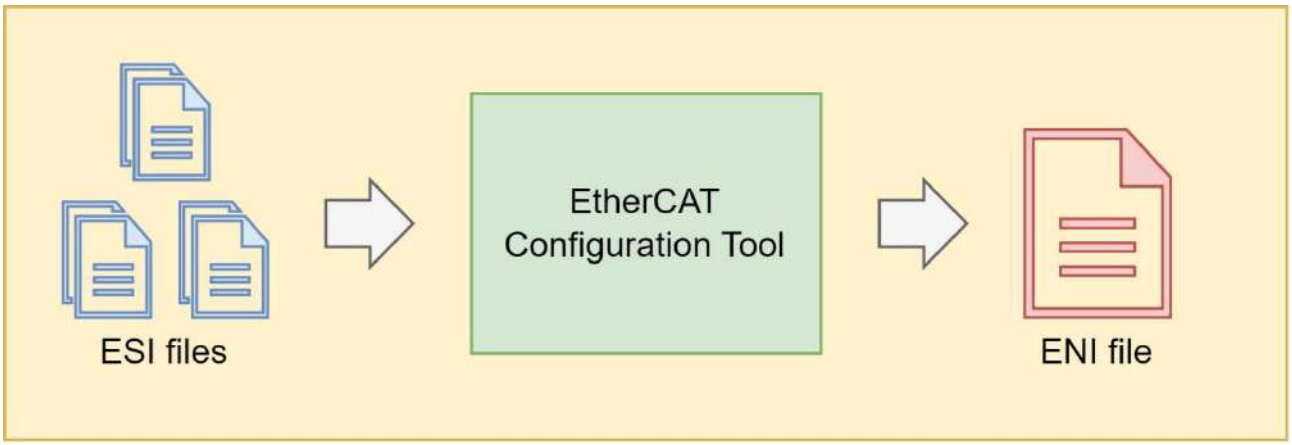
These resources provide valuable information for both beginners and experienced developers using the 86Duino platform. By exploring these links, you can harness the full potential of 86Duino IDE and create innovative projects.

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.