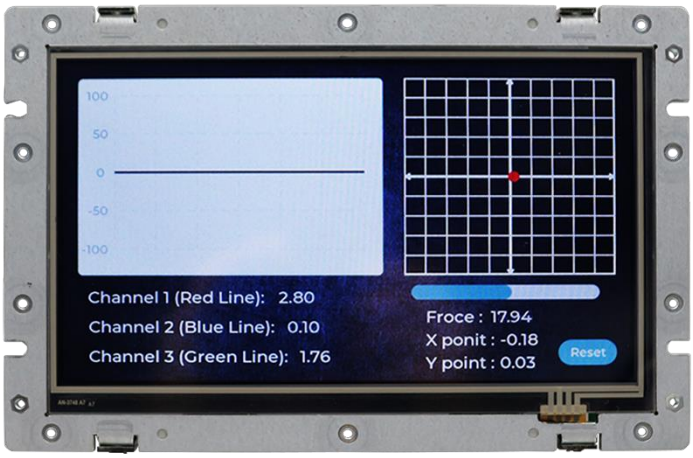


User Manual

QEC-M-070T

DM&P Vortex86EX2 Processor

EtherCAT Master System

7" Open Frame Panel PC with 4-wire Resistive Touch Screen

(Revision 2.3)

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.**
2024/2/26	**Version2.3**	**Add Getting Started.**

COPYRIGHT

Ver.2.3 February, 2024

TRADEMARKS ACKNOWLEDGMENT

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

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

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

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

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

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

This Manual is for the QEC series.

SAFETY INFORMATION

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

WARNING!

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

Content

Ch. 1

General Information

1.1 Introduction

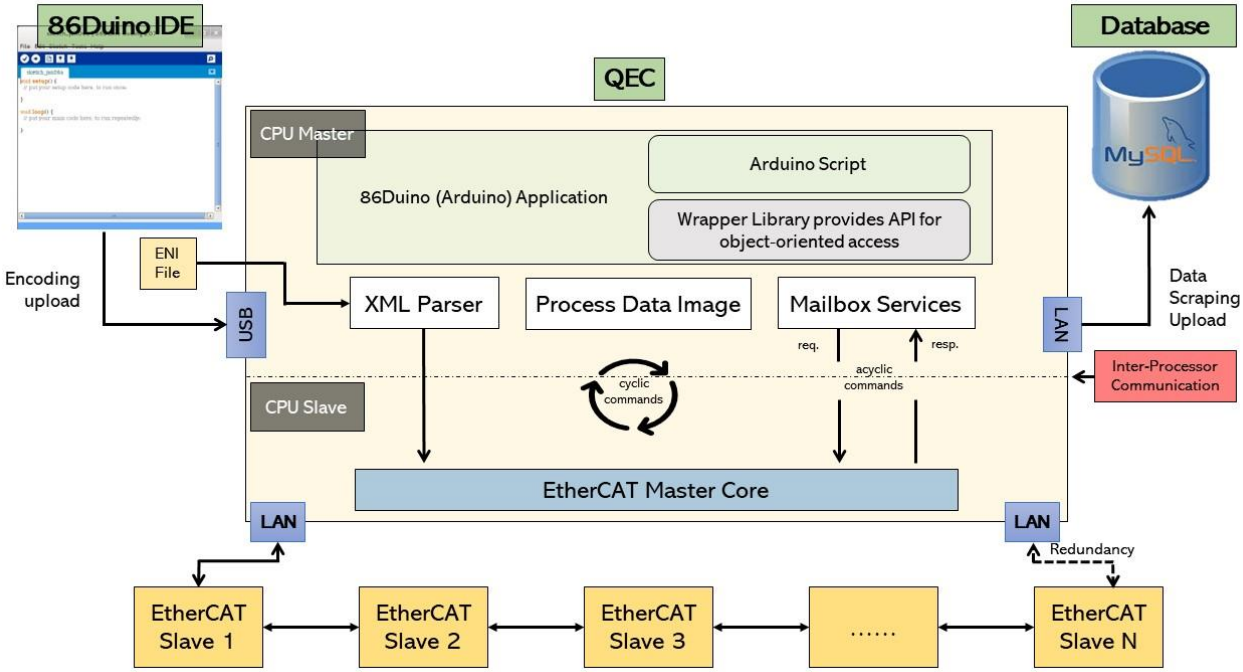
ICOP's QEC-M-070T is an EtherCAT master with a real-time, reliable, and synchronous control equipped with a 7-inch TFT LCD. The industrial Arduino makes it an easy-to-develop, stable, and economical automation system, allowing users to develop EtherCAT with UI quickly.

The QEC master is highly compatible with third-party EtherCAT devices for communication, such as servo, I/O, etc. It supports PDO, CoE, FoE, DC, and EtherCAT cable redundancy to use other EtherCAT slaves flexibly. The QEC master has precise synchronization (min.125μs), and its 86Duino IDE provides less than 1us jitter time in the minimum cycle time; it could apply to highly synchronized and precision automatic applications, like motion control and I/O control. (Read More: EtherCAT Master's Benchmark – QEC)
QEC-M-070T has a built-in high endurance 2GB SLC eMMC, designed to provide a stable and reliable operating system. Users can upload the developed executable files and required images or data, such as HMI images, to the QEC-M-070's SLC via the 86Duino IDE without affecting the performance of the master system. Besides, 86Duino IDE also integrates with the LVGL library to provide an advanced and intuitive approach to user interface design on QEC-M-070T.

QEC-M-070T can also monitor hardware information on temperature, voltage, and current. These features allow users to track the system's carbon footprint and estimate its lifespan. QEC-M-070T's dimension is 186 x 121.05 x 31.05 mm, with its open frame design for easy integration and customization. The operating temperature is from -20°C to +70°C, with an extended option of -40 to +85°C.

QEC-M-070T has two networks for EtherCAT Cable redundancy and one Giga LAN for external network connection. It also offers unrivaled connectivity, featuring 3 USB ports, a MicroUSB port (debug/upload only), VGA output slot, and full Arduino function Pins (GPIO, PWM, SPI, I2C, CAN, etc.); All provide an off-the-shelf API to use.

1.1.1 QEC-M Systems Diagram



1.1.2 Software Support

The 86Duino integrated development environment (IDE) software makes it easy to write code and upload it to QEC-M. 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.

1.2 Specifications

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	
Expansion	Mini PCIe x 1 with Micro SIM Card Holder	
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 Micro USB (Type-B) x1 (Upload/Debug only)	VGA Connector (10-pin) x1 Micro SIM Card Holder x1 MiniPCIe slot 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 (EtherCAT Master Functions: PDO, CoE, DC, Cable-redundancy, 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/-40 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

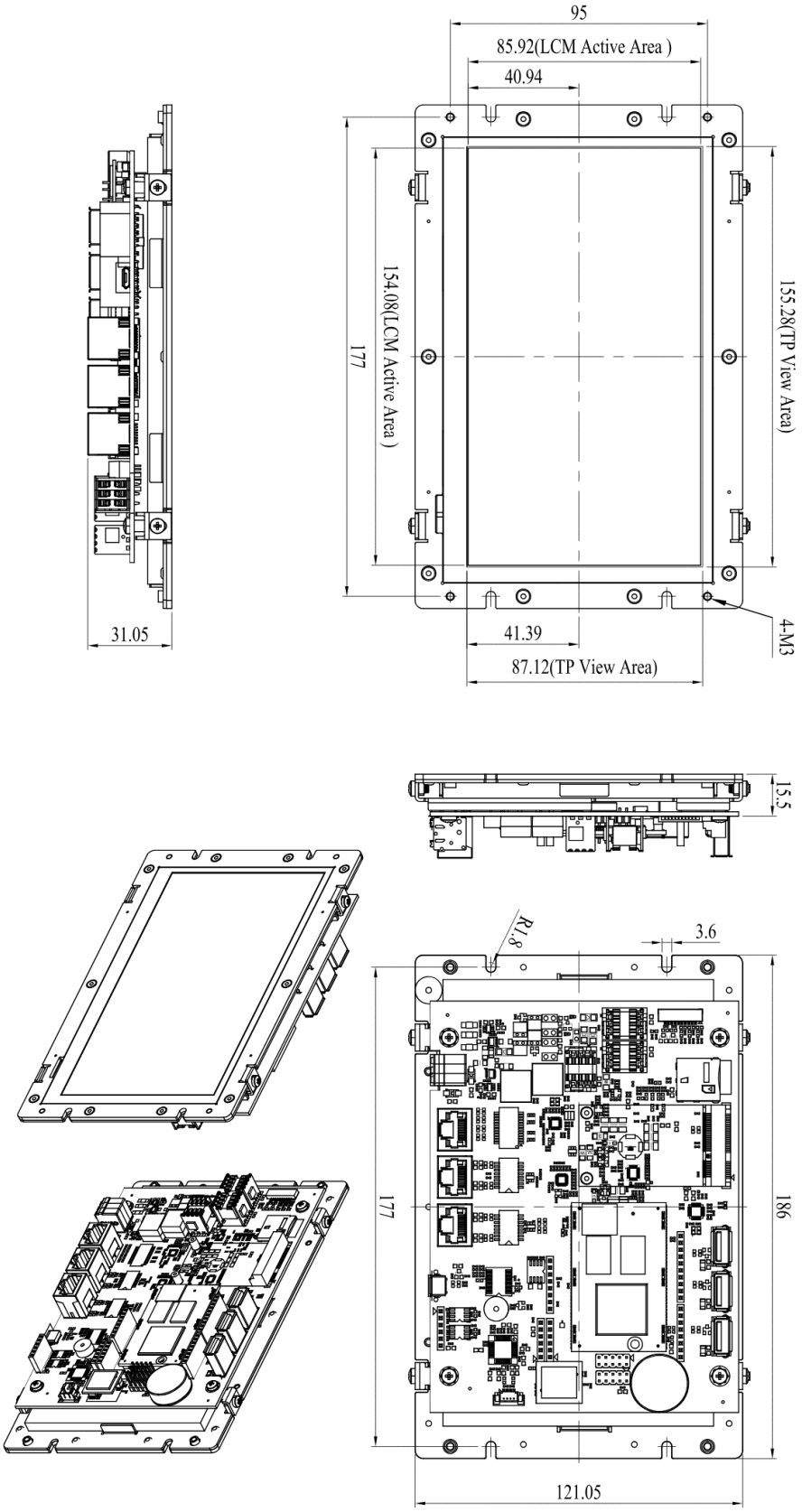
LCD SPECIFICATIONS

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 120°, Horizontal 140°
Backlight Lifetime	20,000 hrs

TOUCHSCREEN

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

1.3 Dimension



(Unit: mm)

1.4 Inspection standard for TFT-LCD Panel

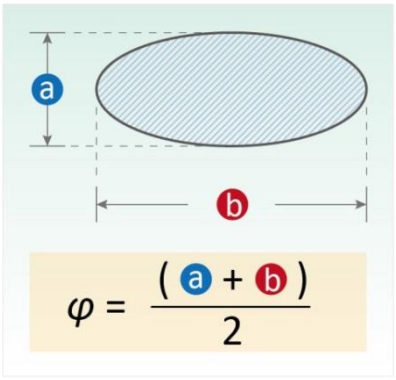
<table>
<tr><th colspan="3">DEFECT TYPE</th><th colspan="7">LIMIT</th><th>Note</th></tr>
<tr><td rowspan="12">VISUAL DEFECT</td><td rowspan="12">INTERNAL</td><td rowspan="3">SPOT</td><td colspan="5">φ＜0.15mm</td><td colspan="2">Ignore</td><td rowspan="3">Note1</td></tr>
<tr><td colspan="5">0.15mm≦φ≦0.5mm</td><td colspan="2">N≦4</td></tr>
<tr><td colspan="5">0.5mm＜φ</td><td colspan="2">N=0</td></tr>
<tr><td rowspan="2">FIBER</td><td colspan="5">0.03mm<W≦0.1mm, L≦5mm</td><td colspan="2">N≦3</td><td rowspan="2">Note1</td></tr>
<tr><td colspan="5">1.0mm＜W, 1.5mm＜L</td><td colspan="2">N=0</td></tr>
<tr><td rowspan="3">POLARIZER BUBBLE</td><td colspan="5">φ＜0.15mm</td><td colspan="2">Ignore</td><td rowspan="3">Note1</td></tr>
<tr><td colspan="5">0.15mm≦φ≦0.5mm</td><td colspan="2">N≦2</td></tr>
<tr><td colspan="5">0.5mm＜φ</td><td colspan="2">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="7">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"></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">DARK 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">TOTAL 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">TWO 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">THREE OR MORE ADJACENT DOT</td><td colspan="6">NOT ALLOWED</td><td></td></tr>
<tr><td colspan="2">LINE DEFECT</td><td colspan="6">NOT ALLOWED</td><td></td></tr>
</table>

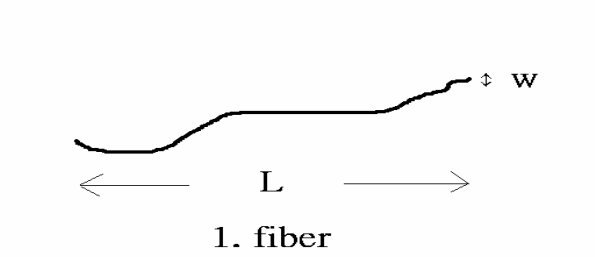
(1) One pixel consists of 3 sub-pixels, including R, G, and B dot. (Sub-pixel = Dot)

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

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

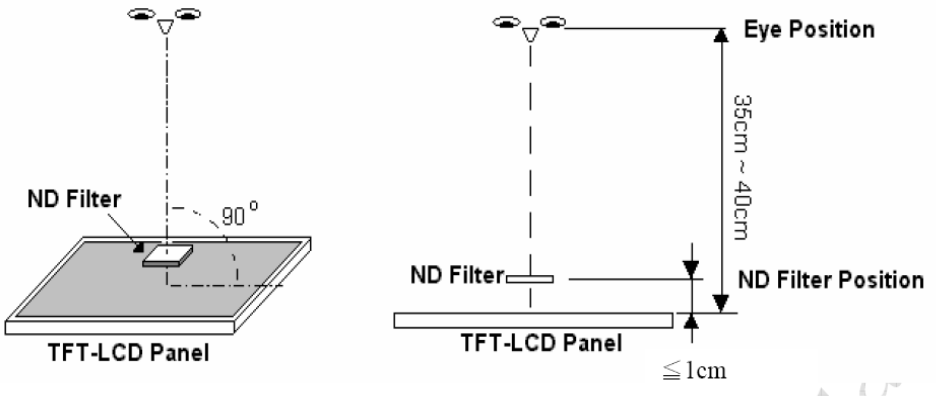
[Note 1] W: Width[mm]; L: Length[mm]; N: Number; φ: Average Diameter.



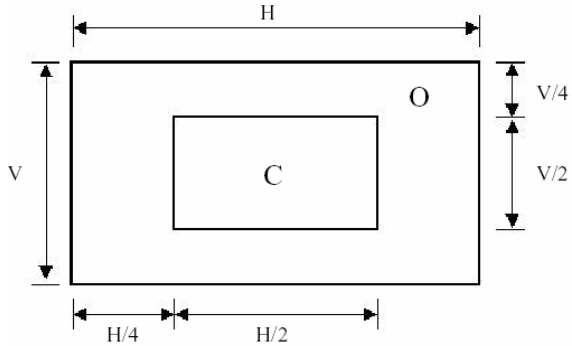


(a) White / Black Spot (b) Polarizer Bubble

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

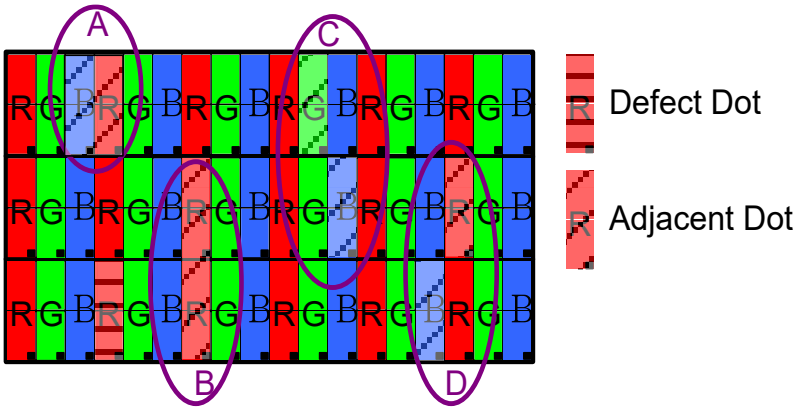


[Note 3] Display area



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.



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.

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	070T	X		X	X		X		X

1. Type: Code 1~4

M: EtherCAT Master

2. LCD size: Code 5~8

070T: 7-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)

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: -40 to +85°C / R: -20 to +70°C (Standard: -20 to +70°C)

6. Coating: Code 13

C: Yes / N: Normal

QEC-M-070T X - XX - X - X

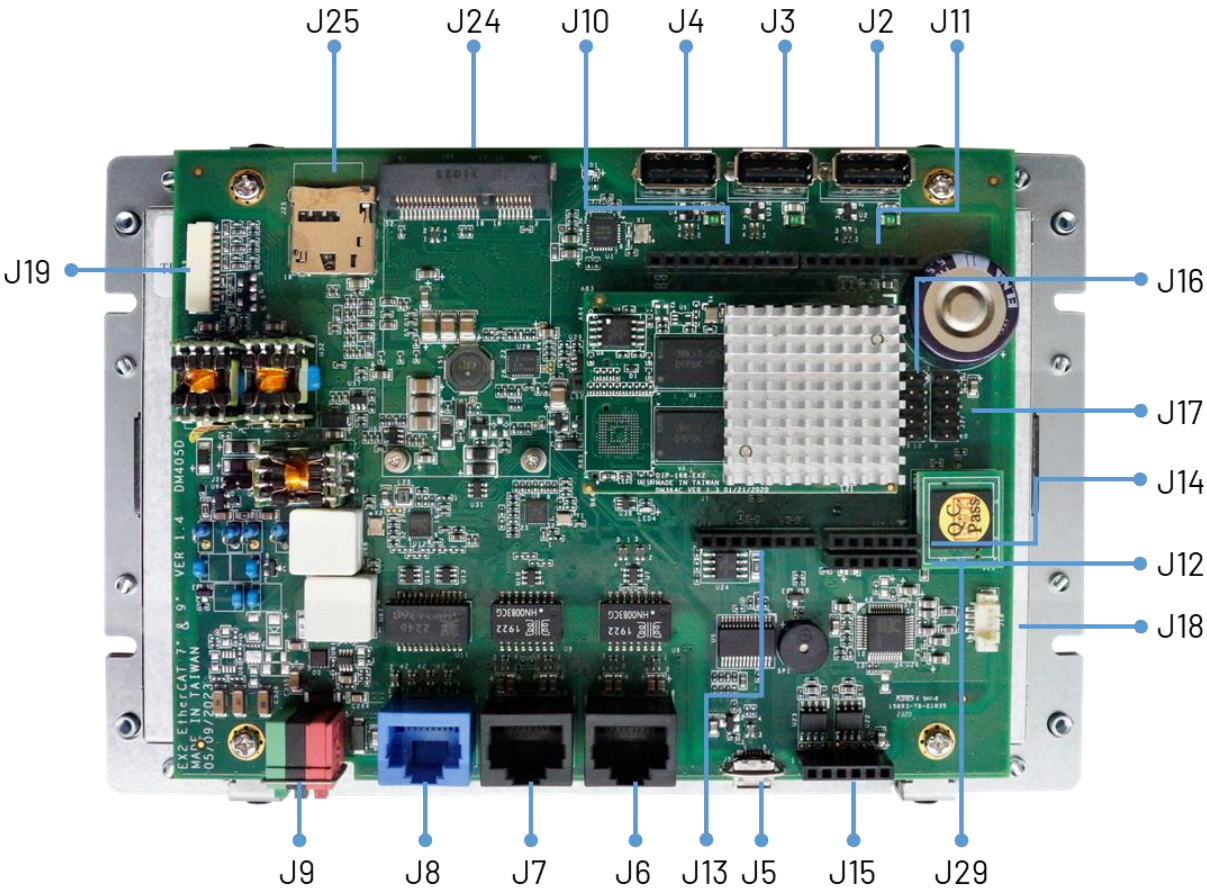
1.5.1 Ordering Part Number

- **QEC-M-070T**: Vortex86EX2 Processor 533MHz-based EtherCAT Master System with 7-inch LCD
- **QEC-M-070TP**: Vortex86EX2 Processor 533MHz-based EtherCAT Master System with 7-inch LCD/PoE
- **QEC-M-070TP-G2-X-C**: Vortex86EX2 Processor 533MHz-based EtherCAT Master System with 7-inch LCD/PoE/1G DDRIII Memory/2G eMMC Storage/Wide Temp./Coating

Ch. 2

Hardware Installation

2.1 CPU Board Outline



2.2 Connector Summary

No.	Description	Type of Connections	Pin #
J2/J3/J4	USB 2.0 Host	1.25mm 5-pin wafer	6-pin
J5	Micro USB	Micro USB Type-B	11-pin
J6	R6040-LAN1	1.25mm 8-pin wafer	8-pin
J7	R6040-LAN2	1.25mm 8-pin wafer	8-pin
J8	Giga LAN	1.25mm 8-pin wafer	8-pin
J9	Power Input Connector	2.00mm 6-pin wafer	6-pin
J10	I2C0, MCM, GPIO	2.54mm female header	10-pin
J11	MCM, GPIO, COM1(TTL)	2.54mm female header	8-pin
J12	GPIO, VCC, GND	2.54mm female header	6-pin
J13	Power source, RESET-	2.54mm female header	8-pin
J14	ADC/GPIO	2.54mm female header	6-pin
J15	CAN0 and CAN1 bus	2.54mm female header	6-pin
J16	SPI, RESET-	2.54mm header	10-pin
J17	SPI, RESET-, RS485	2.54mm header	10-pin
J18	Line-out	1.25mm 4-pin wafer	4-pin
J19	VGA (Reserved and debug used)	1.25mm 10-pin VGA	10-pin
J24	MINI PCIe	Mini PCIe Slot	52-pin
J25	SIM Card Holder	Micro SIM Socket	10-pin
J29A	eMMC TINY MODULE	1.27mm 6-pin header	6-pin
J29B	eMMC TINY MODULE	1.27mm 4-pin header	4-pin

2.3 I/O Connectors

J2/J3/J4: USB 2.0 Host

Pin#	Signal Name
1	VCC
2	MUSBD-
3	MUSBD+
4	GND
5	FGND1

J5: Micro USB Type-B

Pin#	Signal Name
1	USBDET
2	USBDEV2-
3	USBDEV2+
4	-
5	GND

J6: R6040-LAN1 (Secondary EtherCAT)

Pin#	Signal Name
L1	TD+
L2	TD-
L3	RO+
L4	OUT_Us
L5	OUT_Up
L6	RO-
L7	VsGND
L8	VpGND

* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

J7: R6040-LAN2 (Primary EtherCAT)

Pin#	Signal Name
L1	TD+
L2	TD-
L3	RO+
L4	IN_Us
L5	IN_Up
L6	RO-
L7	VsGND
L8	VpGND

* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

J8: Giga LAN

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

J9: Power Input Connector

Pin#	Signal Name
1	FGND
2	FGND
3	VpGND
4	VsGND
5	Vp
6	Vs

J10: I2C0, MCM, GPIO

Pin#	Signal Name
1	GP30
2	MCM-9
3	MCM-10
4	MCM-11
5	GP31
6	MCM-13
7	GND
8	-
9	I2C0_SDA
10	I2C0_SCL

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

J11: MCM, GPIO, COM1 (TTL)

Pin#	Signal Name
1	RXD1#
2	TXD1#
3	GP00
4	MCM-3
5	GP02
6	MCM-5
7	MCM-6
8	GP05

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

J12: GPIO, VCC, GND

Pin#	Signal Name
1	-
2	GP35
3	GP36
4	GP37
5	GND
6	VCC

J13: Power source, RESET-

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

J14: ADC/GPIO

Pin#	Signal Name
1	GP57ADC
2	GP56ADC
3	GP43ADC
4	GP42ADC
5	GP41ADC
6	GP40ADC

J15: CAN0 and CAN1 bus

Pin#	Signal Name
1	CAN1_L
2	CAN1_H
3	GND
4	CAN0_L
5	GAN0_H
6	VCC3

J16: SPI, RESET-

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

J17: SPI, RESET-, RS485

Pin#	Signal Name	Pin#	Signal Name
1	SPI1_DI	2	VCC
3	SPI1_CLK	4	SPI1_DO
5	RESET-	6	GND
7	SPI1_CS	8	-
9	RS485+	10	RS485-

J18: Line-out

Pin#	Signal Name
1	LOUT_R1
2	GND_AUD
3	GND_AUD
4	LOUT_L1

J19: VGA

Reserved and debug used.

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

J24: MINI PCIe

Pin#	Signal Name	Pin#	Signal Name
1	WAKE#	2	+3.3V
3	Reserved / +5V Power-out	4	GND
5	Reserved / +5V Power-out	6	-
7	-	8	SIM-VCC
9	GND	10	SIM-IO
11	REFCLK-	12	SIM-CLK
13	REFCLK+	14	SIM-RST
15	GND	16	SIM-VPP
Mechanical Key			
17	-	18	GND
19	-	20	-
21	GND	22	PERST#
23	PERn0	24	+3.3V
25	PERp0	26	GND
27	GND	28	-
29	GND	30	-
31	PETn0	32	-
33	PETp0	34	GND
35	GND	36	USB_D-
37	GND	38	USB_D+
39	+3.3V	40	GND
41	+3.3V	42	LED_WWAN#
43	GND	44	Reserved / DCD
45	-	46	-
47	-	48	-
49	-	50	GND
51	-	52	+3.3V

J25: SIM Card Holder

Pin#	Signal Name	Pin#	Signal Name
1	SIM-VCC	2	SIM-RST
3	SIM-CLK	4	GND
5	SIM-VPP	6	SIM-IO
7	-	8	-

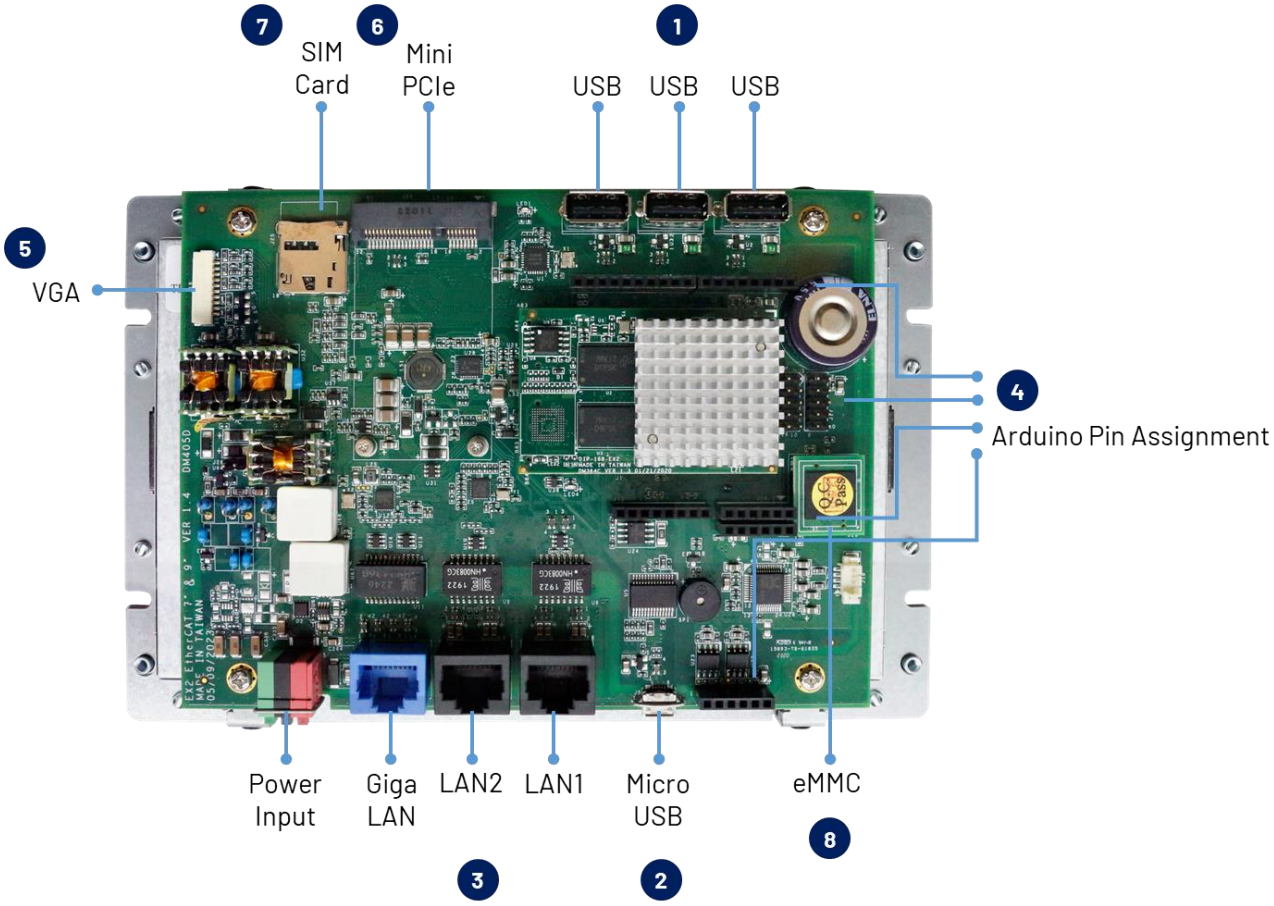
J29A: eMMC Module

Pin#	Signal Name	Pin#	Signal Name
A1	SDA_D0	A2	SDA_D1
A3	SDA_D2	A4	SDA_D3
A5	SDA_CLK	A6	SDA_CMD

J29B: eMMC Module

Pin#	Signal Name	Pin#	Signal Name
B1	VCC3	B2	SDA_CD
B3	VCC3	B4	GND

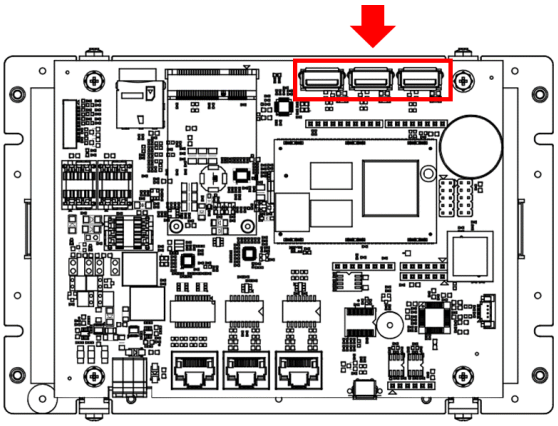
2.4 External I/O Overview



NOTE

1. Standard USB 2.0 Host.
2. Micro USB is mainly for the programming upload.
3. LAN1 and LAN2 are for the EtherCAT communication, and Giga LAN.
4. Arduino Standard Pin and QEC additional Arduino Pin.
5. 10 Pin VGA connector. (Reserved and debug used)
6. Mini PCIe.
7. SIM Card.
8. eMMC Module.
9. Power Input.

2.4.1 USB



Standard USB 2.0 with Hot-plug.

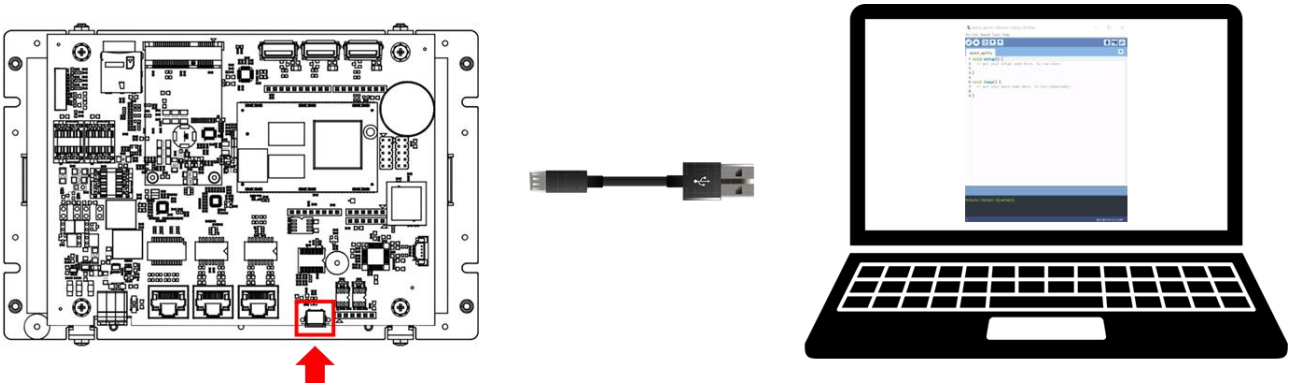
You can plug in the Keyboard, Mouse, or USB dick to control the QEC-M-070T.

For drive USB, you can refer to the following hyperlinks:

- SD library: read USB disk.
- Keyboard Controller Example
- Mouse Controller Example

2.4.2 Micro USB

The Micro USB is mainly for programming upload.



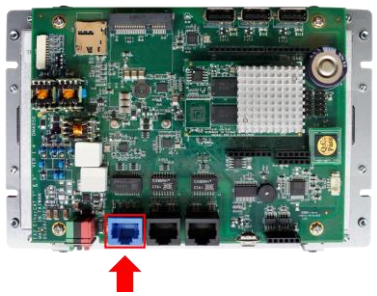
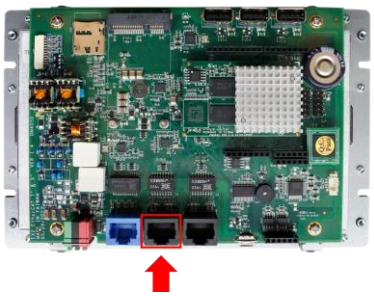
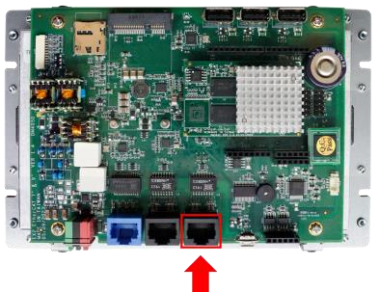
For quick start guide, please see Ch. 3 Getting Started.

2.4.3 LAN1/LAN2/Giga LAN

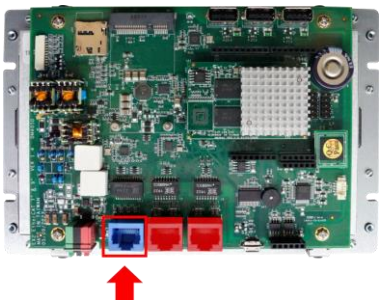
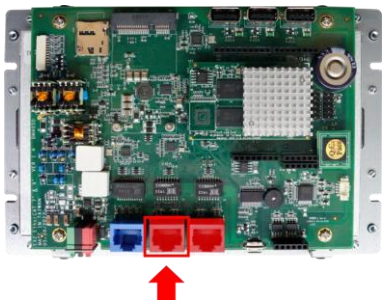
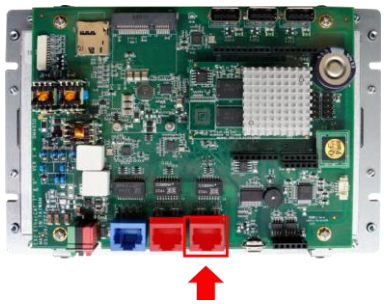
There are three LAN ports in QEC-M-070T, two for EtherCAT communication and one for external Ethernet work. The EtherCAT Lan on the QEC-M divides into Input and Output for cable redundancy.
To drive GigaLAN, you can refer to Ethernet library or Modbus Library.
To drive EtherCAT ports, you can refer to EtherCAT Master API User Manual.

Giga LAN with the Blue Housing; PoE LAN with the Red Housing; Regular LAN with Black Housing.

With Non-PoE:

Giga Lan	EtherCAT Input	EtherCAT Output
		

With PoE:

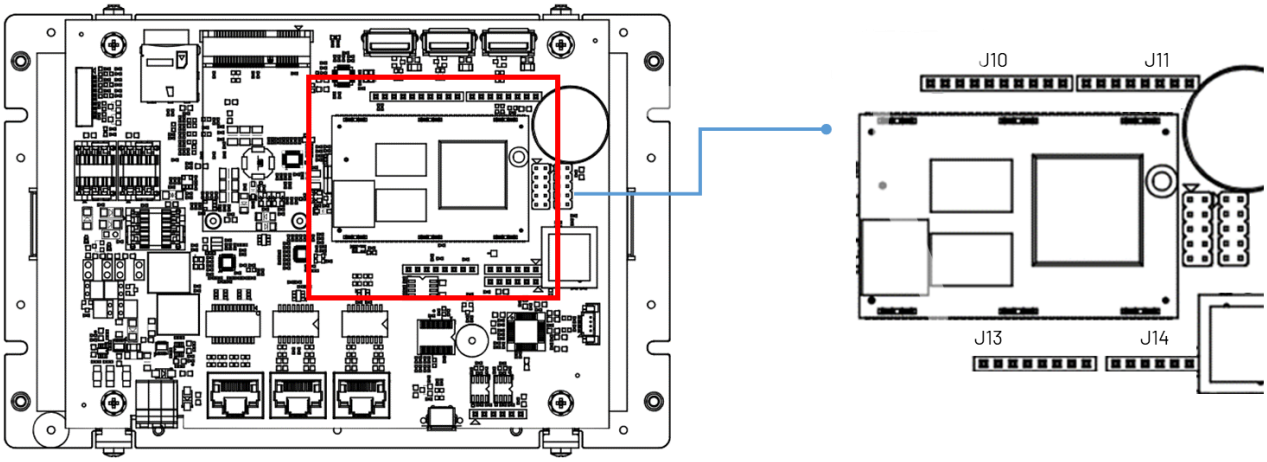
Giga Lan	EtherCAT Input	EtherCAT Output
		

2.4.4 Arduino pin Assignment

We have kept the Arduino pin on the QEC-M-070T. Users can easily control these pins via software (86Duino IDE). To drive Arduino pins, you can refer to Libraries and Language.

Arduino standard pins:

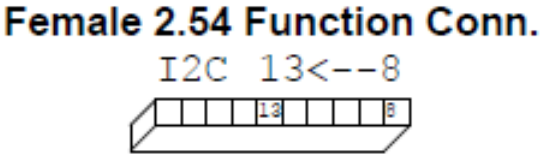
You can use the following pins like 86Duino One board.



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

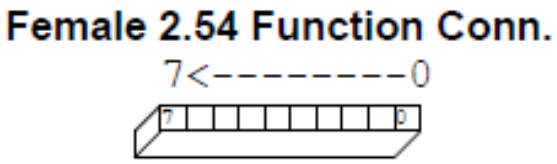
J10: I2C0, MCM, GPIO

Pin#	Signal Name
1	GP30
2	MCM-9
3	MCM-10
4	MCM-11
5	GP31
6	MCM-13
7	GND
8	-
9	I2C0_SDA
10	I2C0_SCL



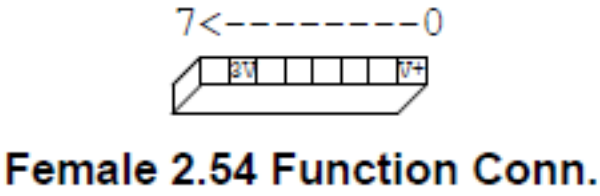
J11: MCM, GPIO, COM1(TTL)

Pin#	Signal Name
1	RXD1#
2	TXD1#
3	GP00
4	MCM-3
5	GP02
6	MCM-5
7	MCM-6
8	GP05



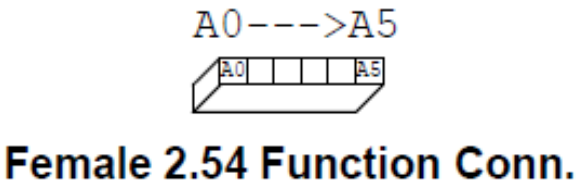
J13: Power source, RESET-

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



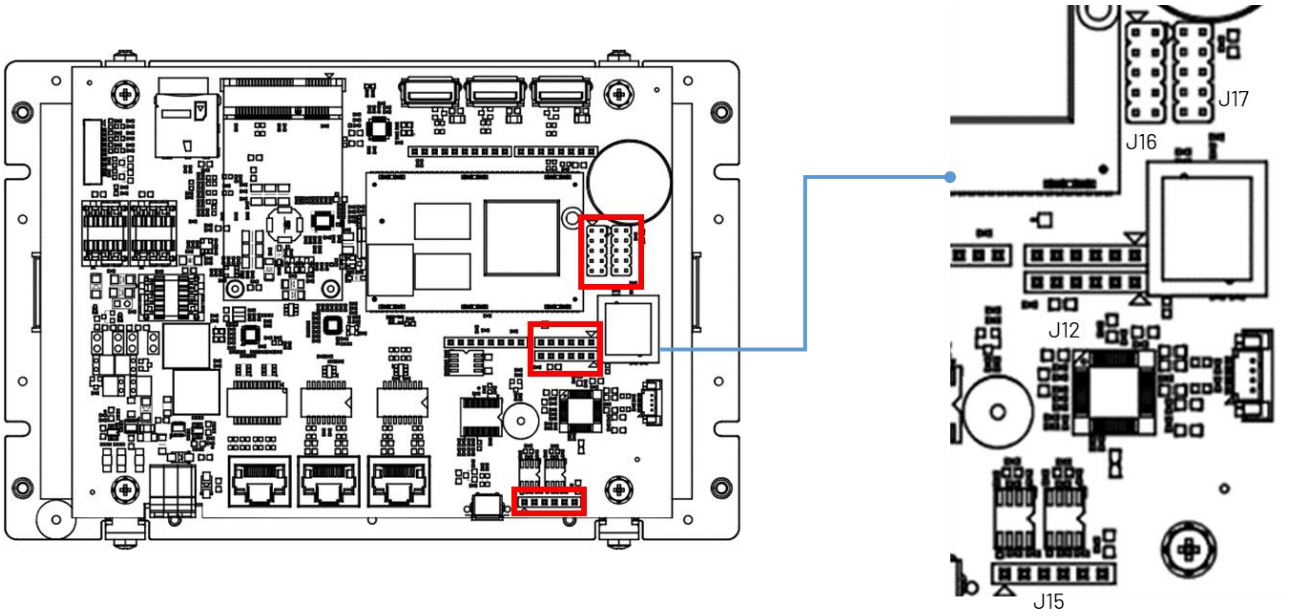
J14: ADC/GPIO

Pin#	Signal Name
1	GP57ADC
2	GP56ADC
3	GP43ADC
4	GP42ADC
5	GP41ADC
6	GP40ADC



QEC Arduino pins:

There are other pins on the QEC-M-070T.
To drive Arduino pins, you can refer to Libraries or please contact ICOP for the details.



J12: GPIO, VCC, GND

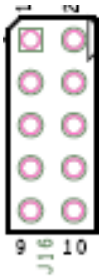
Pin#	Signal Name
1	-
2	GP35
3	GP36
4	GP37
5	GND
6	VCC

J15: CAN0 and CAN1 bus

Pin#	Signal Name
1	CAN1_L
2	CAN1_H
3	GND
4	CAN0_L
5	GAN0_H
6	VCC3

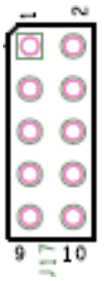
J16: SPI, RESET-

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



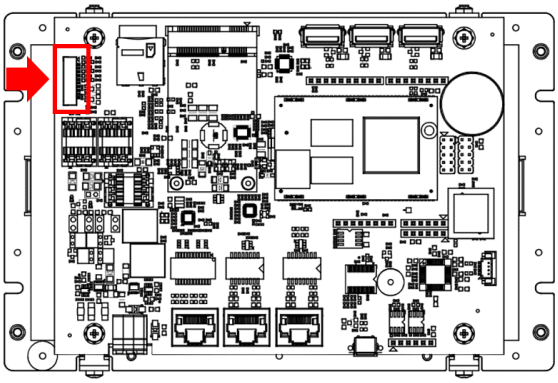
J17: SPI, RESET-, RS485

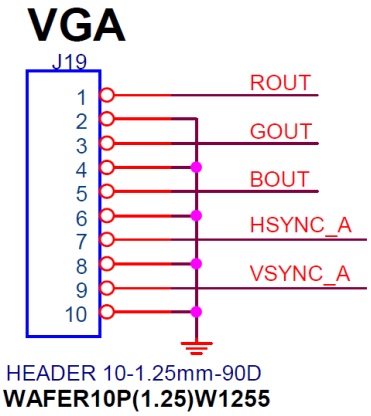
Pin#	Signal Name	Pin#	Signal Name
1	SPI1_DI	2	VCC
3	SPI1_CLK	4	SPI1_DO
5	RESET-	6	GND
7	SPI1_CS	8	-
9	RS485+	10	RS485-



2.4.5 10-pin VGA Connector

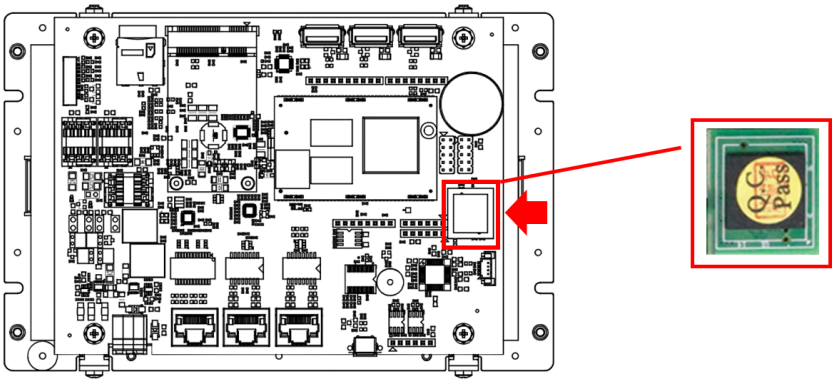
Reserved and debug used.





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

2.4.6 eMMC Module



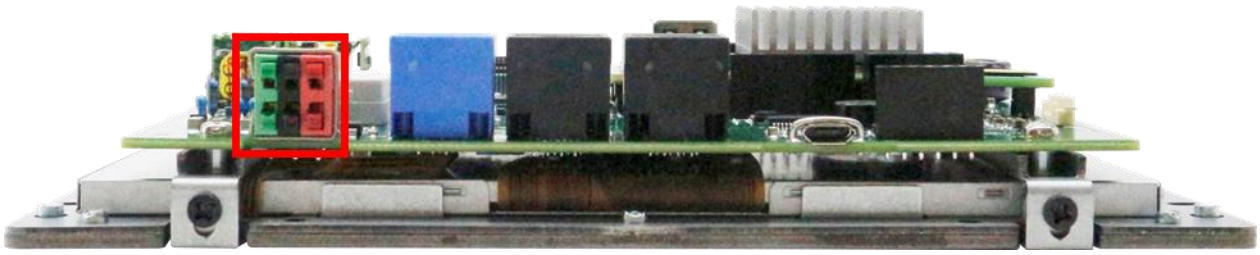
A 2GB eMMC module is onboard by default.
Your 86Duino executable will be uploaded to this eMMC.

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() {

}
```

2.4.7 Power Input

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



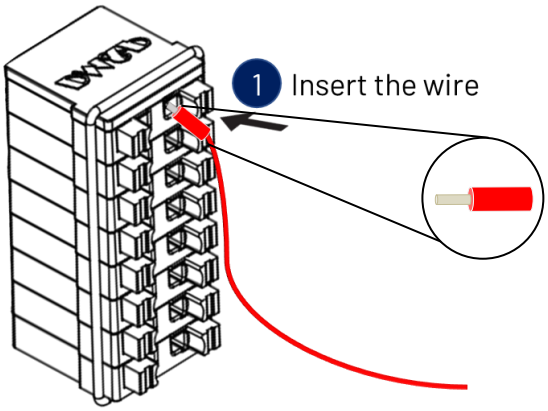
Vs for system power; Vp for peripheral power and backup power.

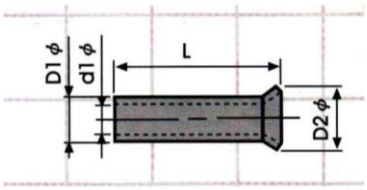
Pin #	Signal Name	Pin #	Signal Name
1	Vs+	2	Vp+
3	Vs- (GND)	4	Vp- (GND)
5	F.G	6	F.G

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

2.5 Wiring to the Connector

2.5.1 Connecting the wire to the connector

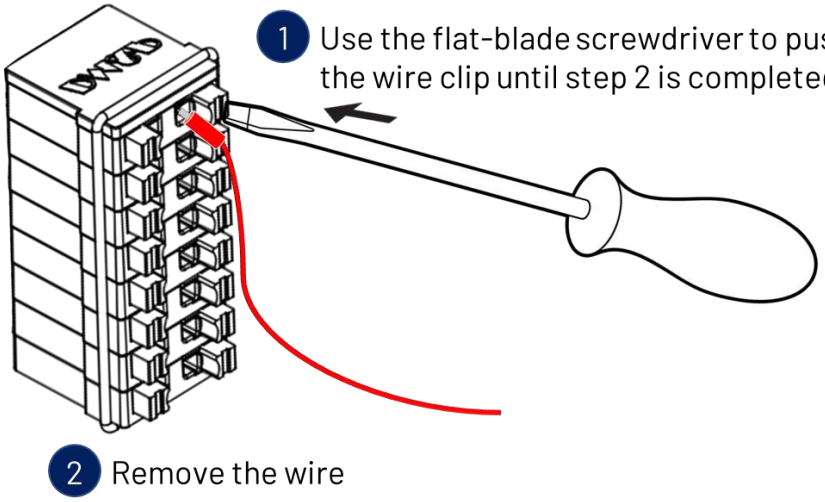




Insulated Terminals Dimensions (mm)

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

2.5.2 Removing the wire from the connector



Ch. 3

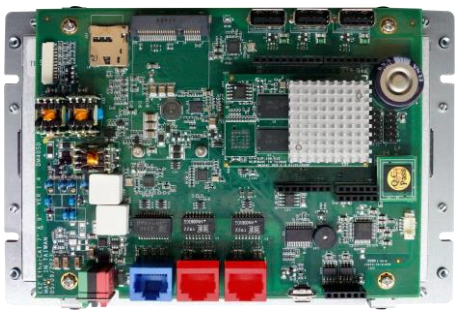
Getting Started

This chapter explains how to start with QEC-M-070T 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

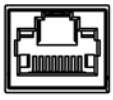


PoE type

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 master connects with a third-party EtherCAT slave).

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.

3.1 Package Contents

The package includes the following items:

- QEC-M-070T
- Cable-set

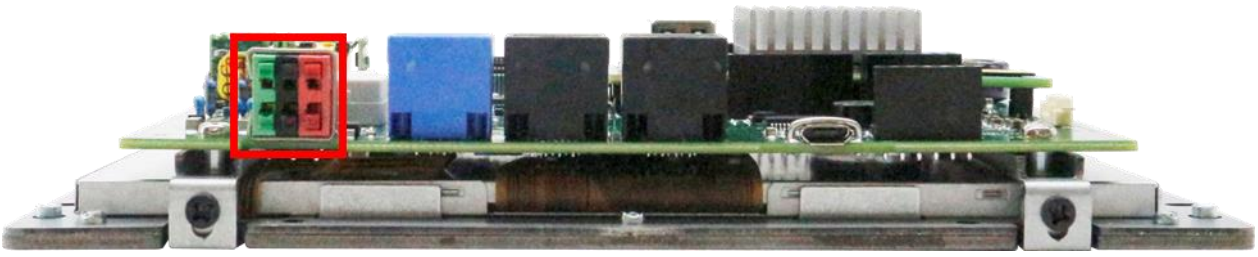
Please get in touch with our sales channels if any of the package items are missing or damaged. Also, feel free to reuse the shipping materials and carton for further storing and shipping needs in the future,

3.2 Hardware Configuration

The development environment will be pre-installed before the QEC-M is shipped to customers. Users must download the software (see 3.3 Software/Development Environment) and follow this user manual to set up the device.

3.2.1 Plugin the power supply

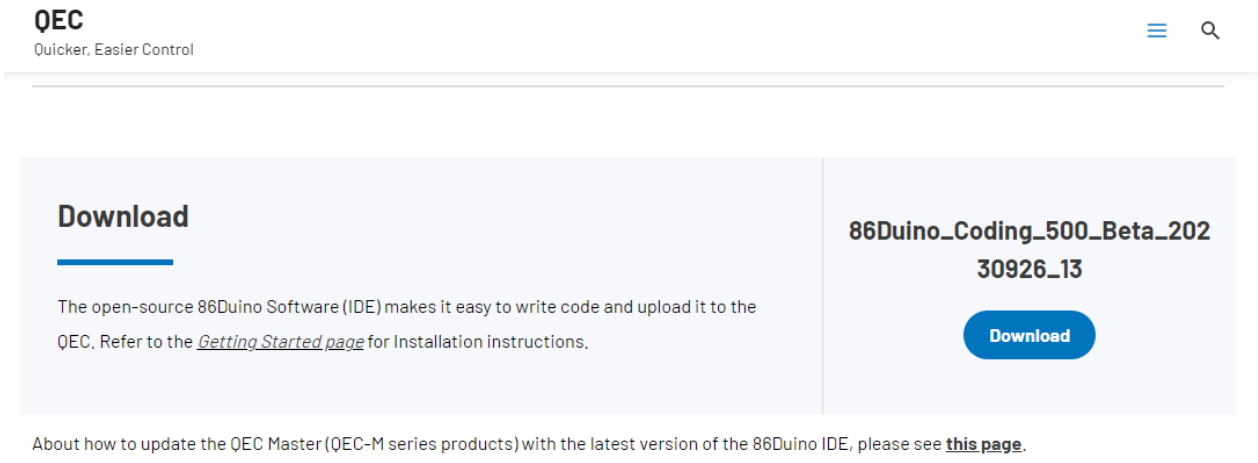
There are two groups of power supplies in QEC-M-070T, Vs and Vp; The voltage requirement for both supplies' ranges from 19V to 50V wide voltage. After powering on, you'll see the power screen light up.



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

3.3 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.

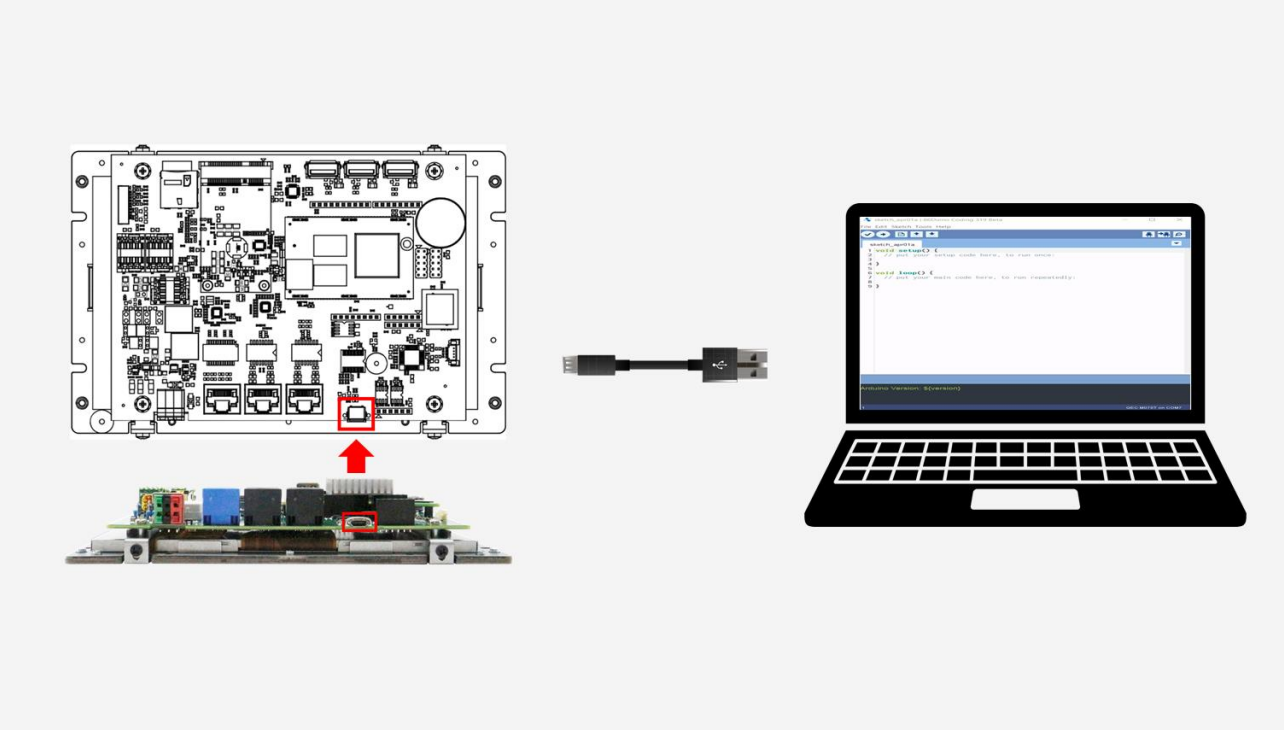
*Note: If Windows displays a warning, click Details once and then click the Continue Run button once.

86Duino Coding IDE 500+ looks like below.

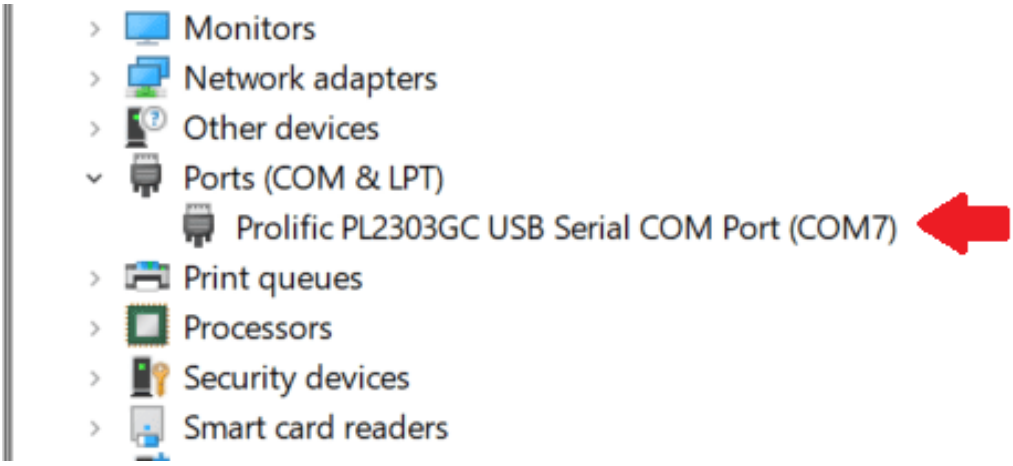
3.4 Connect to your PC and set up the environment

Follow the steps below to set up the environment:

1. Connect the QEC-M-070T to your PC via a Micro USB to USB cable (86Duino IDE installed).

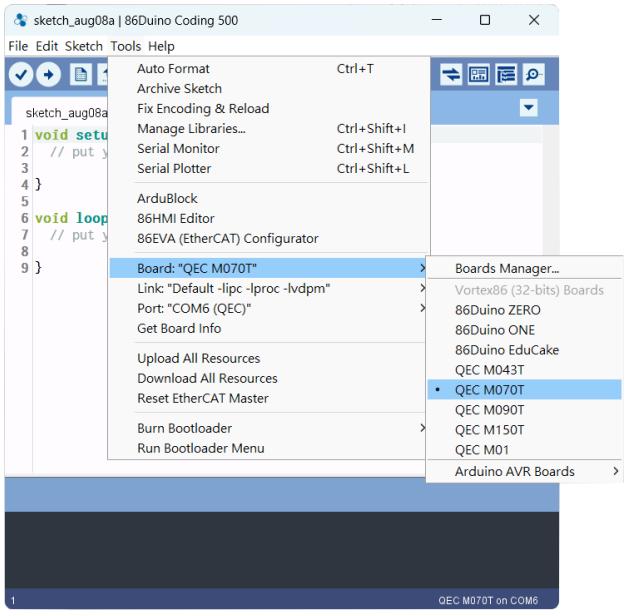


2. Turn on the QEC power.
3. Open "Device Manager" -> "Ports (COM & LPT)" in your PC and expand the ports; you should see that the "Prolific PL2303GC USB Serial COM Port (COMx)" is detected; if not, you will need to install the required drivers.

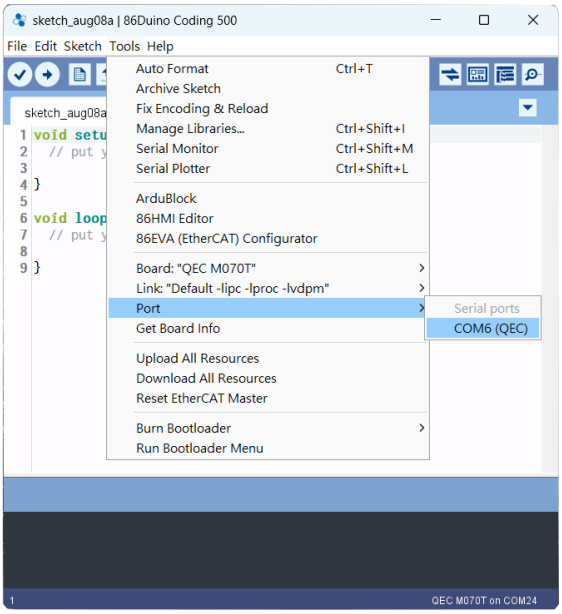


For Windows PL2303 driver, you can download here.

4. Open the 86Duino IDE.
5. Select the correct board: In the IDE's menu, select “Tools” ->” Board”- > QEC M070T (or the QEC-M master model you use).



6. Select Port: In the IDE's menu, select “Tools”- >” Port” and select the USB port to connect to the QEC-M master (in this case, COM6 (QEC)).



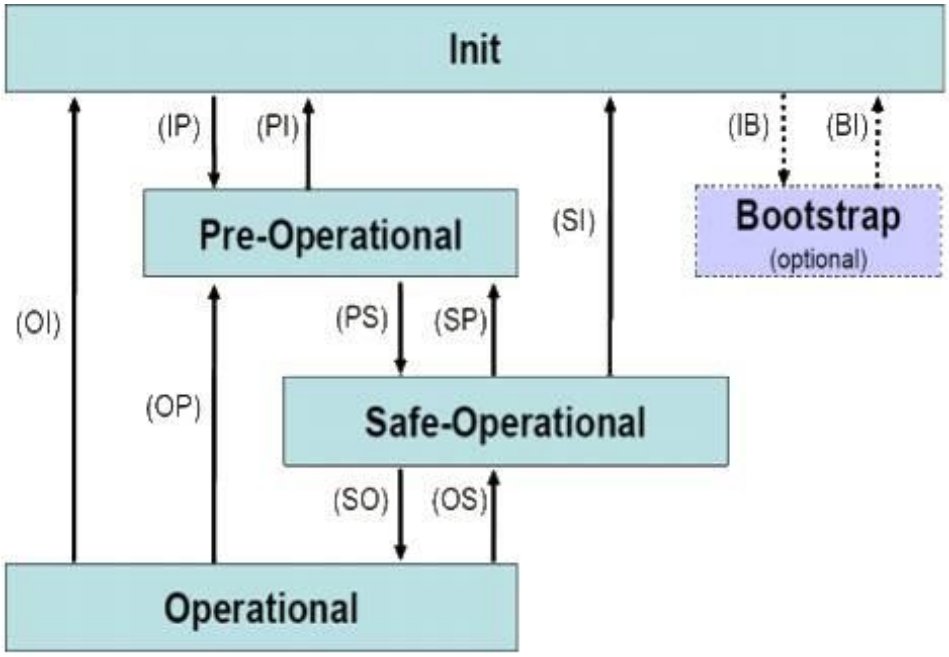
3.5 EtherCAT Development Method

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

3.5.1 Implementing the EtherCAT State Machine

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

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

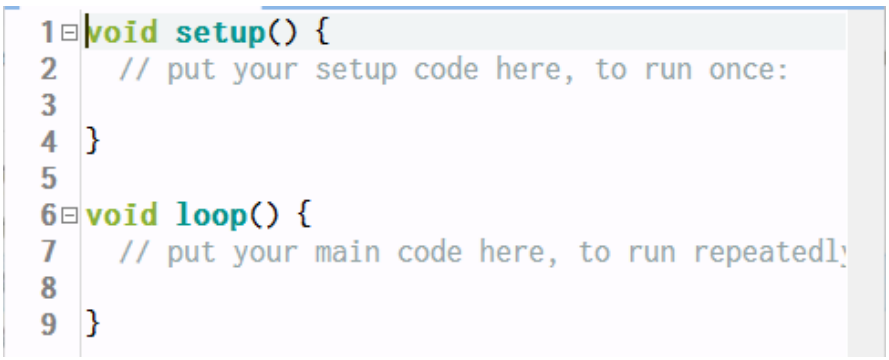


A distinction is made between the following states:

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

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

The EtherCAT master (QEC-M-070T) 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.



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

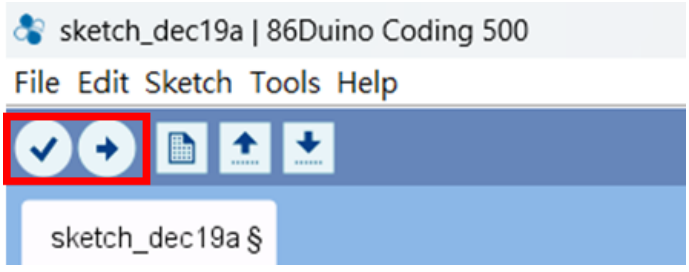
Free-run Mode Code Example:

```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster master; // Create an EtherCAT Master Object
EthercatDevice_Generic slave; // Create an Generic EtherCAT Slave Object

void setup() {
  // EtherCAT Master Initialize. All slaves will enter PRE-OP state if
success.
  master.begin();

  // Specify the EC-Slave number and mount it on the EC-Master.
  slave.attach(0, master);

  // Start EtherCAT Master. All slaves will enter OP state if success.
  // FREERUN Mode, and parameter 1000000 sets the cycle time in nanoseconds
  master.start(1000000, ECAT_FREERUN_AUTO);
}

void loop() {

}
```

SYNC Mode Code Example:

```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster master; // Create an EtherCAT Master Object
EthercatDevice_Generic slave; // Create an Generic EtherCAT Slave Object

void setup() {
  // EtherCAT Master Initialize. All slaves will enter PRE-OP state if
success.
  master.begin();

  // Specify the EC-Slave number and mount it on the EC-Master.
  slave.attach(0, master);

  // Start EtherCAT Master. All slaves will enter OP state if success.
  // Sync Mode, and the parameter 1000000 sets the cycle time in nanoseconds
  master.start(1000000, ECAT_SYNC);
}

void loop() {

}
```

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. The program will run when the upload is complete.

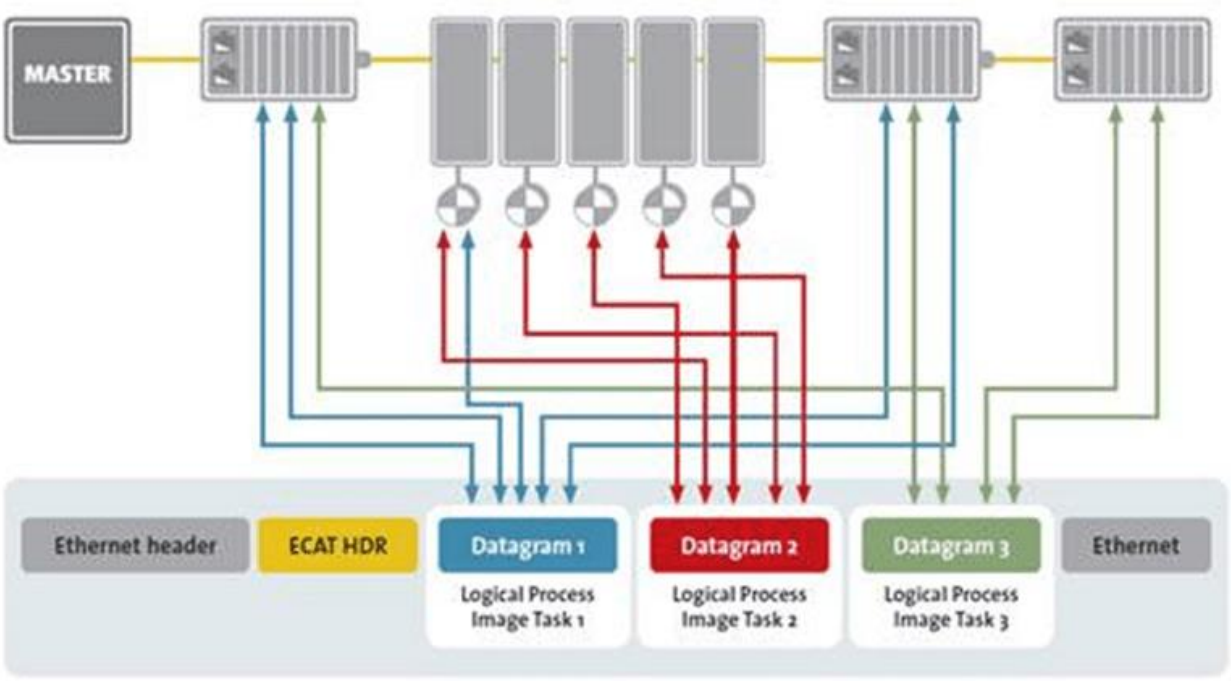


3.5.2 Process Data Objects (PDO) Functions

Process Data Objects (PDOs) are fundamental components within the EtherCAT communication protocol, primarily designed to facilitate efficient and rapid data exchange between an EtherCAT Master and its slave devices. PDOs are instrumental in achieving the high-speed data transmission capabilities that EtherCAT is renowned for, making it a preferred choice for real-time industrial automation and control systems.

PDOs serve as the primary mechanism for exchanging real-time control and feedback data between the EtherCAT Master and slaves. Unlike Service Data Objects (SDOs), which are used for configuring and accessing complex device parameters, PDOs are optimized for quick, cyclic data exchange. This distinction allows EtherCAT networks to maintain low communication latencies and high throughput, essential for real-time applications.

Through addressing mode of EtherCAT and the memory control technology "Fieldbus Memory Management unit (FMMU)" performed by EC-Slaves hardware, we can exchange all data synchronically from all ECAT-Slaves on bus by just passing one single internet packet. The types of data include DIO, AIO and servo motor position, etc. Please refer to the diagram below.



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

Types of PDOs

There are two main types of PDOs: Transmit PDOs (TPDOs) and Receive PDOs (RPDOs).

1. Transmit PDOs (TPDOs): These are used by slave devices to send process data to the EtherCAT Master. TPDOs typically contain status information, sensor readings, or any feedback data that needs to be monitored by the control system.
2. Receive PDOs (RPDOs): Conversely, RPDOs are utilized by the EtherCAT Master to transmit control commands and setpoints to the slaves. This can include motor control commands, actuator positions, or any output data that the control system needs to send to the devices.

Configuration

PDO configuration is a critical process that involves specifying which data points are included in each PDO, their order, and how they are mapped to the device's application objects.

The flexibility of PDO mapping allows for the optimization of data transmission based on the specific needs of the application, contributing to the efficiency of the EtherCAT protocol.

- Static Mapping: In some systems, PDO mapping is fixed by the device manufacturer, limiting the ability to modify which data points are included in a PDO.
- Dynamic Mapping: More commonly, EtherCAT devices support dynamic PDO mapping, enabling users to customize which variables are included in a PDO and in what order. This customization can be achieved through SDOs during the system setup phase, allowing for a highly tailored and optimized communication setup.

PDO Function Code Example:

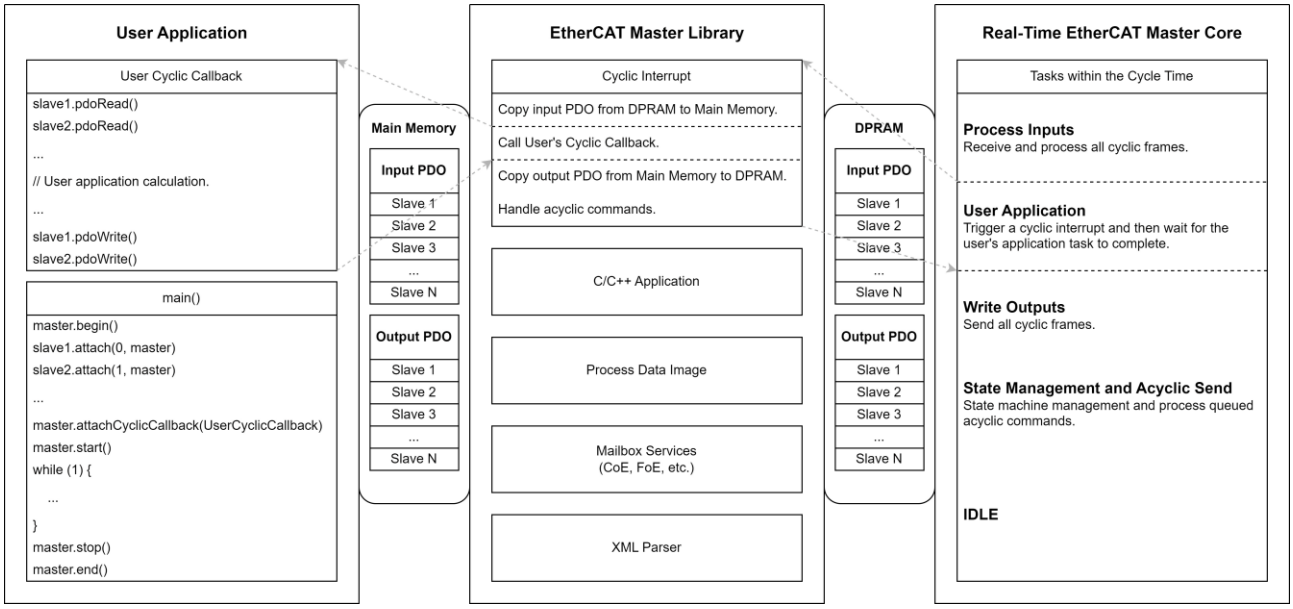
When using a non-QEC slave, you can use the EthercatDevice_Generic function, which can be referenced EthercatDevice Class. We expect the EtherCAT Slave device is Digital IO module.

```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster master; // Create an EtherCAT Master Object
EthercatDevice_Generic slave; // Create an EtherCAT Slave Object

void setup() {
   // Initialize the EtherCAT Master. If successful, all slaves enter PRE-
OPERATIONAL state
   master.begin();

   // Attach the QEC-R11D88H slave device to the EtherCAT Master at position 0
   slave.attach(0, master);

   // Start EtherCAT Master. All slaves will enter OP state if success.
  // Sync Mode, and the parameter 1000000 sets the cycle time in nanoseconds
   master.start(1000000, ECAT_SYNC);
}

void loop() {
   // Write a HIGH value to a bit in the Process Data Output at index 0
   slave.pdoBitWrite(0, HIGH);
   delay(4000); // Wait for 4 seconds

   // Write a LOW value to the same bit in the Process Data Output at index 0
   slave.pdoBitWrite(0, LOW);
   delay(1000); // Wait for 1 second
}
```

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. The program will run when the upload is complete.

3.5.3 Cyclic Callback

Cyclic Callbacks are integral to the efficiency of EtherCAT Master systems, particularly when running on the Vortex86EX2 processor, as depicted in the provided system architecture diagram. They provide the mechanism for periodic task execution at predefined intervals, which is crucial for maintaining data consistency and real-time responsiveness in industrial automation settings.



Understanding Cyclic Callbacks:

In the EtherCAT Master Library, Cyclic Callbacks are designated functions that are invoked by a cyclic interrupt. The diagram illustrates this process as the "User Cyclic Callback," showing how the EtherCAT Master library interacts with the user application. This framework is essential for executing repetitive tasks such as reading and writing PDOs to and from slave devices, as seen in the "User Application" section of the diagram.

Key Notes on Interrupts:

- Responsiveness to interrupts is critical; they must be handled quickly to preserve the system's real-time capabilities.
- Usage of `delay()` within the ISR is not recommended as it can disrupt the timely execution of code, potentially leading to missed data or system delays.
- Variables accessed within an ISR must be declared as volatile to ensure their values are updated in real-time, reflecting changes made by the interrupt.

About Interrupt Service Routines (ISRs):

- Each ISR is unique and defined without parameters, nor do they return values. Their execution is triggered by the system's cyclic interrupt as shown in the architecture diagram.
- The ISR should be as short and efficient as possible to avoid hindering the system's performance, especially since the EtherCAT Master Core has dedicated tasks within the cycle time.
- Given that ISRs in a multi-interrupt system are executed singly, their optimization is imperative to prevent delays and ensure the efficient handling of all interrupts.

EtherCAT Specifics:

- Within the EtherCAT system, the User Cyclic Callback plays a critical role in the timely updating of PDOs. The architecture ensures that PDOs are consistently updated and mapped between the main memory and the DPRAM, aligning with the "Input PDO" and "Output PDO" depicted in the system diagram.
- Although functions like `delay()` and `millis()` may operate as expected within ISRs in this specific environment, it is still advisable to avoid lengthy operations within ISR code to maintain system efficiency and real-time performance.

QEC attach the Cyclic Callback Function Code Example

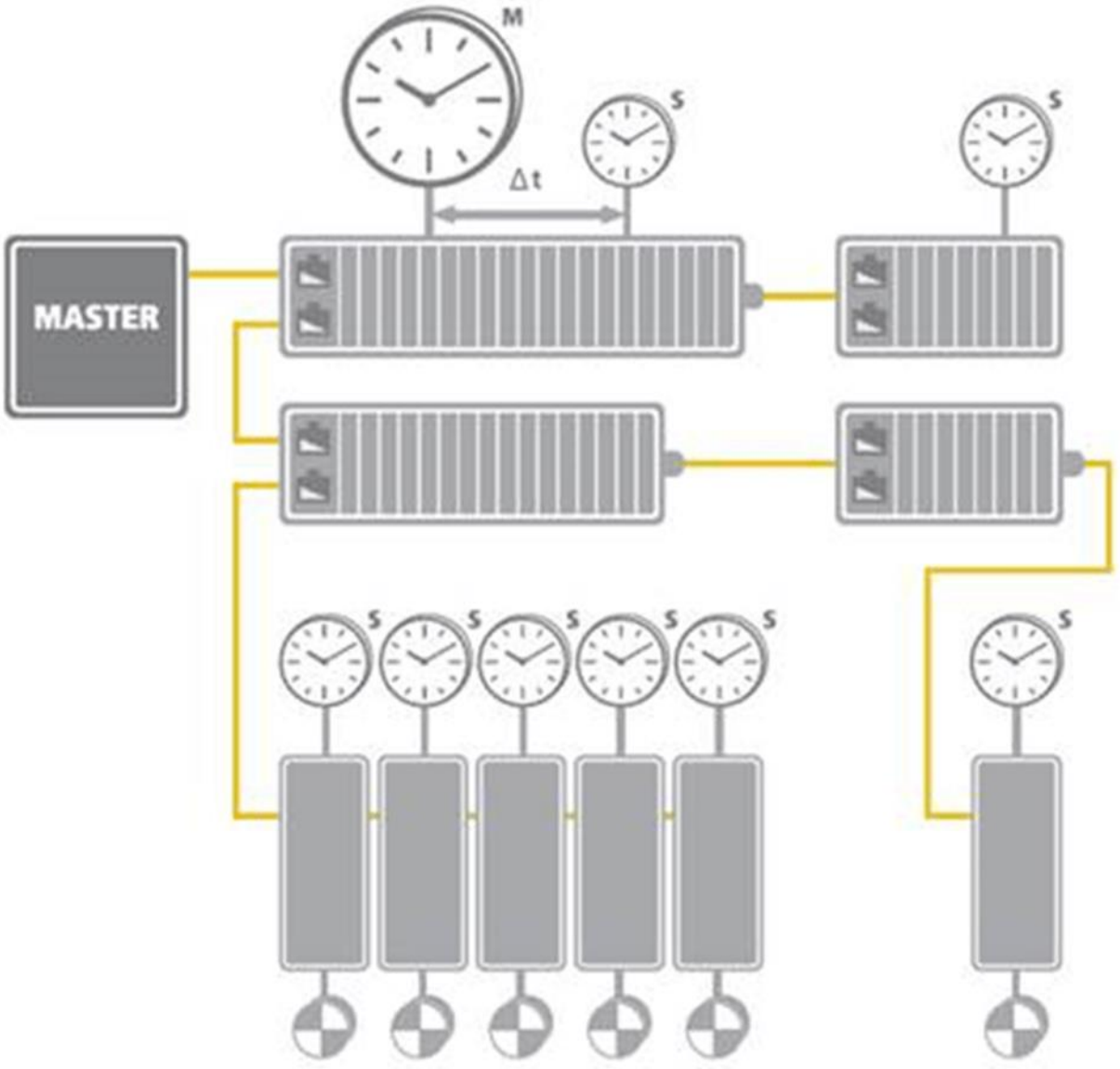
```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster EcatMaster; // Create an EtherCAT Master Object
EthercatDevice_Generic Slave1; // Create an Generic EtherCAT Slave Object

void myCallback() {
  // put your cyclic Callback function here.
}

void setup() {
  // EtherCAT Master Initialize. All slaves will enter PRE-OP state if
success.
  EcatMaster.begin();

  // Specify the EC-Slave number and mount it on the EC-Master.
  Slave1.attach(0, EcatMaster);

  // Set a cyclic callback for the Ethercat Master
  EcatMaster.attachCyclicCallback(myCallback);



  // Start EtherCAT Master. All slaves will enter OP state if success.
  // Sync Mode, and the parameter 1000000 sets the cycle time in nanoseconds
  EcatMaster.start(1000000, ECAT_SYNC);
}

void loop() {

}
```

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. The program will run when the upload is complete, and the LED will start flashing.

3.5.4 Distributed Clock (DC)

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).

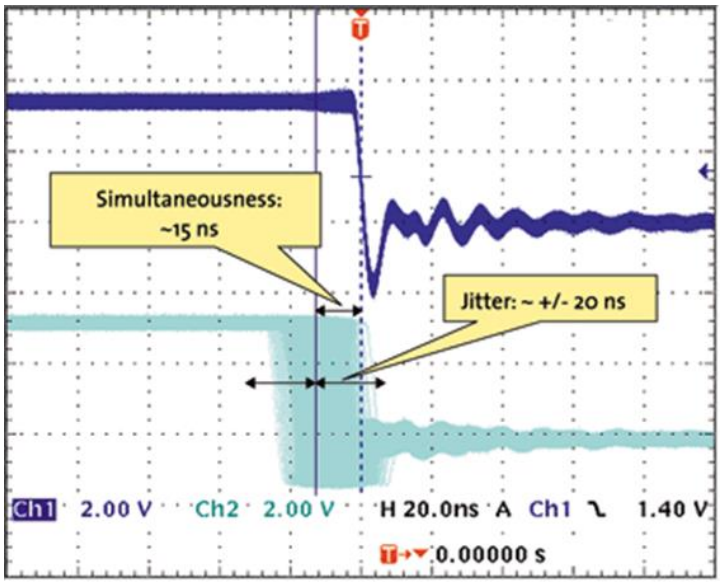


EtherCAT: Illustration of Distributed Clock (DC). (Source of information: http://www.ethercat.org/)

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-Slave 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-Slave local system time is within +/- 20 nano-seconds. The following diagram is a scope view of two slave devices' output digital signals. We can see that the time difference between the I/O signal from two EC-Slaves 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/)

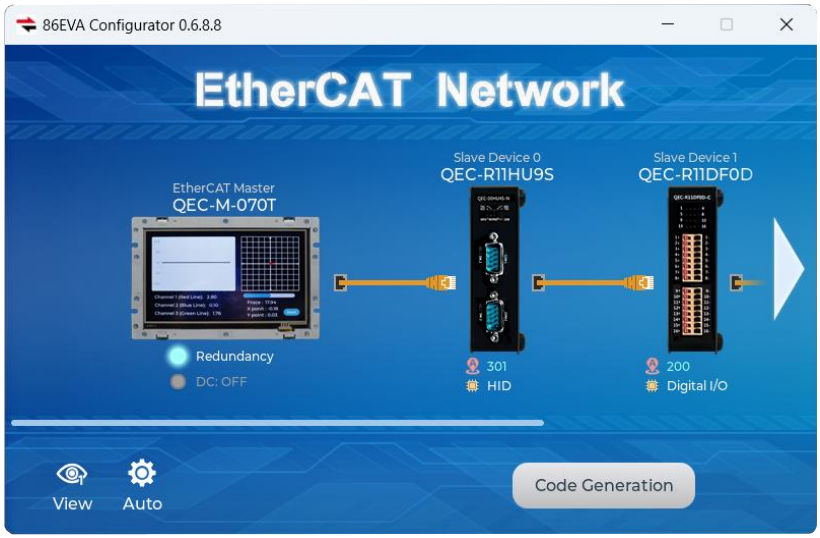
QEC Set DC Code Example

```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster EcatMaster; // Create an EtherCAT Master Object
EthercatDevice_Generic Slave1; // Create a Generic EtherCAT Slave Object
EthercatDevice_Generic Slave2; // Create a Generic EtherCAT Slave Object

void myCallback() {
  // put your cyclic Callback function here.
}

void setup() {
  // EtherCAT Master Initialize. All slaves will enter PRE-OP state if
success.
  EcatMaster.begin();

  // Specify the EC-Slave number and mount it on the EC-Master.
  Slave1.attach(0, EcatMaster);
  Slave1.setDc(1000000); // 1000000 ns = 1 ms

  // Specify the EC-Slave number and mount it on the EC-Master.
  Slave2.attach(1, EcatMaster);
  Slave2.setDc(1000000); // 1000000 ns = 1 ms

  // Set a cyclic callback for the Ethercat Master
  EcatMaster.attachCyclicCallback(myCallback);

  // Start EtherCAT Master. All slaves will enter OP state if success.
  // Sync Mode, and the parameter 1000000 sets the cycle time in nanoseconds
  EcatMaster.start(1000000, ECAT_SYNC);
}
void loop() {

}
```

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. The program will run when the upload is complete.

3.5.5 Use 86EVA with code

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 Slave devices, with features including:

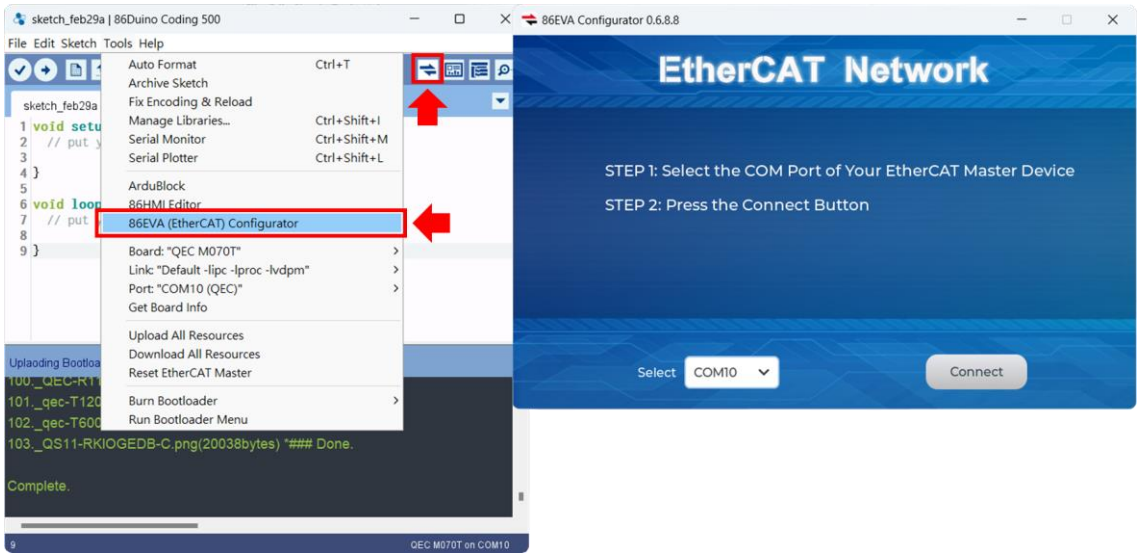
1. Automatically generated Arduino language (via EtherCAT-Based Virtual Arduino)
2. Automatically scan for network devices.
3. EtherCAT Master Settings:
    - Set Master Object Name
    - Set Cycle Time
    - Set Redundancy Options
    - Optional ENI file
4. EtherCAT Slave Settings:
    - Set Slave Object Name
    - Set Slave Alias
    - Slave I/O Mapping can be set
    - Display secondary device information
    - View internal information, including:
        a. Voltage (V)
        b. Current (A)
        c. Temperature (C)
        d. Startup time (hr)

Example

This example uses a QEC-M-070TP to control the QEC-R11D0FD (EtherCAT Slave 16-ch Digital Output) and setting Pin0 of Digital Output to HIGH for 4 seconds and then changing it to LOW for 1 second.

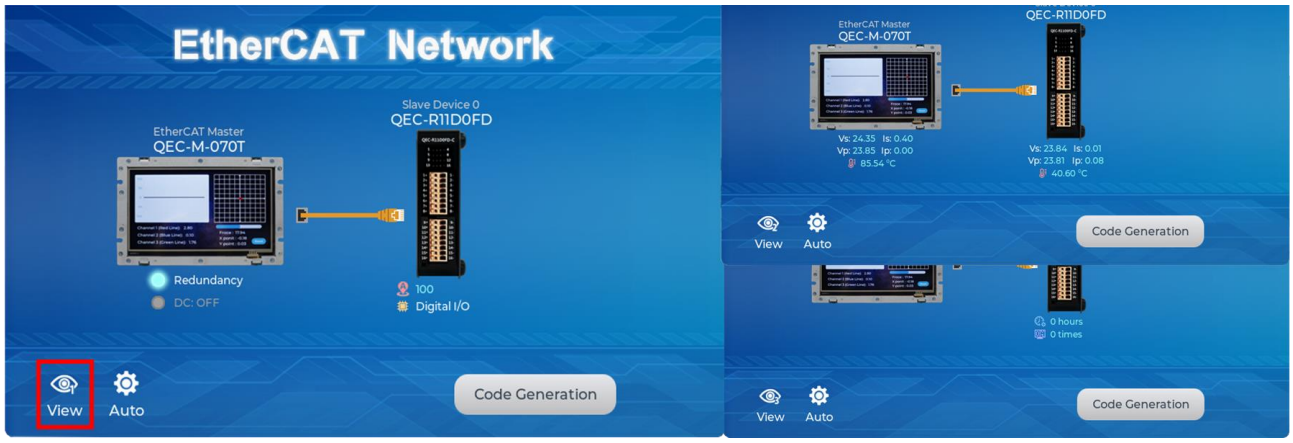
Step 1: Turn on 86EVA and scan

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



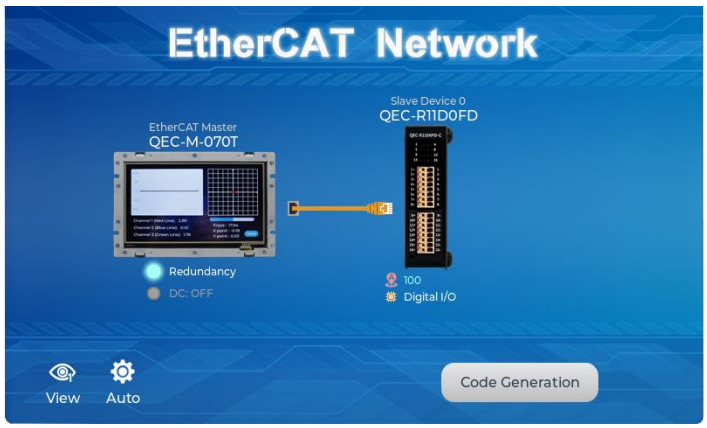
After confirming that the correct COM port (COM10 in this example) has been selected for QEC-M-070TP, press the Connect button to start scanning the EtherCAT network.

The connected devices will be displayed after the EtherCAT network has been scanned.
Press the "View" button in the lower left corner to check the device's status.



Step 2: Set the parameters

Click on the scanned device image to enter the corresponding parameter setting screen.



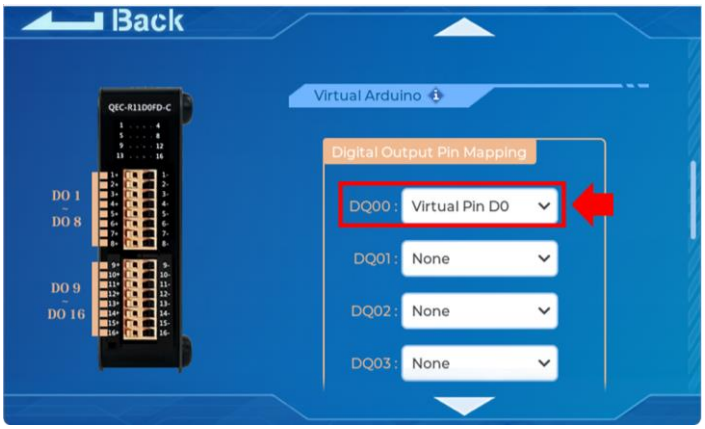
QEC-M-070T:

Click on the image of the QEC-M-070T to see the parameter settings.
If you are developing for the first time, please use the preset settings first and click "**Back**" in the upper left corner to return.

QEC-R11D0FD-N:

Click on the image of the QEC-R11D0FD to see the parameter settings.

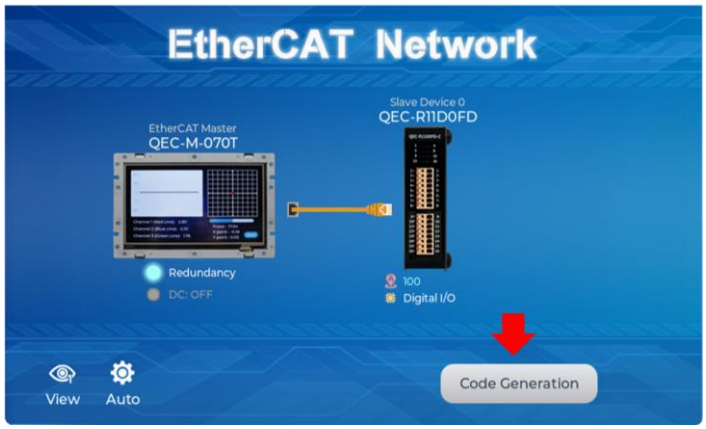
Continue down to the "**Digital Output Pin Mapping"** area. Among them, we select "**Virtual Pin D0**" in the drop-down box of DQ00 of Digital Output Pin Mapping and click "**Back**" in the upper left corner to return.



This action is to set the Digital Output Pin0 of QEC-R11D0FD to the virtual D0 pin of EVA.

Step 3: Generate the code

Once you've set your device's parameters, 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:

- sketch_mar01a: Main Project (.ino, depending on your project name)
- ChatGPT.h: Parameters to provide to ChatGPT referred
- myeva.cpp: C++ program code of 86EVA
- myeva.h: Header file of 86EVA

After 86EVA generates code, the following code will be automatically generated in the main program (.ino), and any of them missing will cause 86EVA not to work.

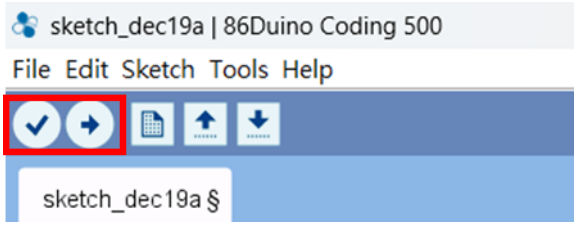
1. `#include “myeva.h”` : Include EVA Header file
2. `EVA.begin()` in `setup();` : Initialize the EVA function

Step 4: Write the code

The following example is setting the D0 pin of EVA (Pin0 for Digital Output) to HIGH for 4 seconds and then changing it to LOW for 1 second.

```
#include "myeva.h" // Include EVA function

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

void loop() {
    // put your main code here, to run repeatedly:
    // Set EVA D0 pin to HIGH
    EVA.digitalWrite(0, HIGH);
    delay(4000); // Wait for 4 seconds

    // Set EVA D0 pin to HIGH
    EVA.digitalWrite(0, LOW);
    delay(1000); // Wait for 1 second
}
```

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. The program will run when the upload is complete, and the LED will start flashing.



3.6 Building an HMI on the QEC-M-070T with 86Duino

This section will demonstrate how to build a basic HMI on the QEC-M-070T using the LVGL library in the 86Duino Coding IDE 500+.
We assume that you have already completed the previous sections of the quick start guide, including Package Contents, Hardware Configuration, Software Driver Installation, and Set up the QEC-M-070T.

3.6.1 Library Instruction

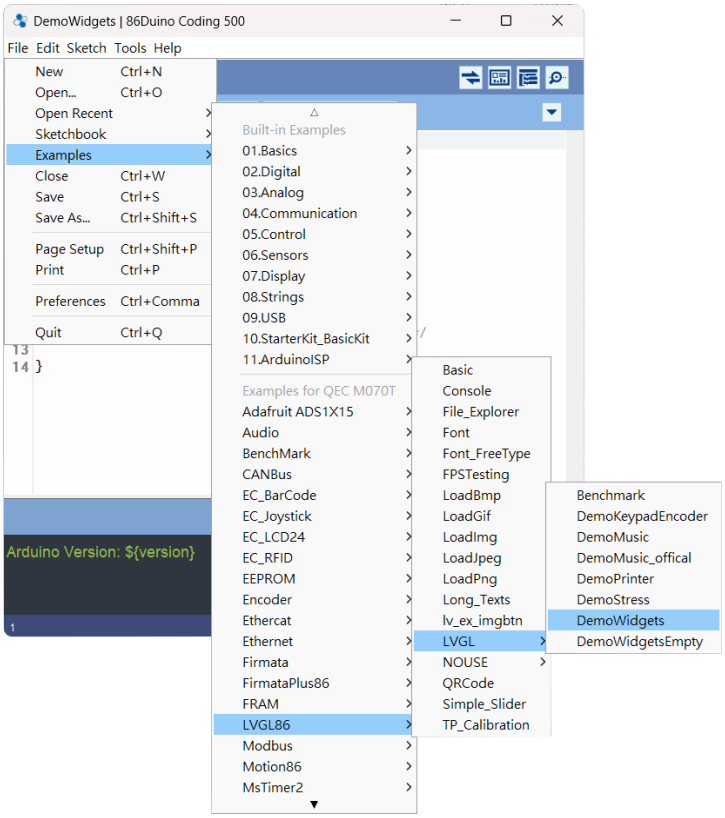
LVGL (Light and Versatile Graphics Library) is a powerful open-source graphics library that enables the creation of attractive and interactive graphical user interfaces (GUIs) for embedded systems. By utilizing the LVGL library in combination with 86Duino IDE, developers can design and implement HMIs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

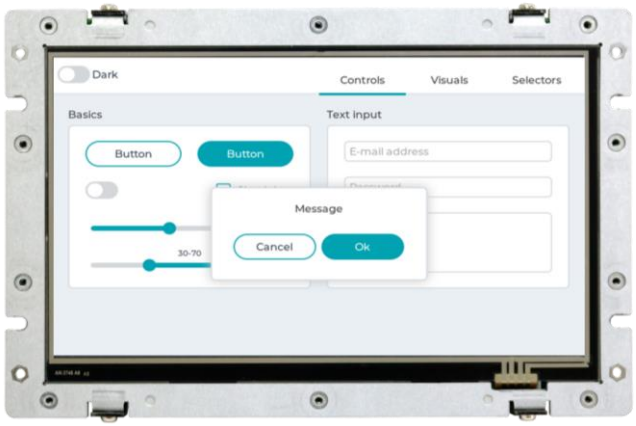
3.6.2 Uploading the LVGL Example

Use the LVGL example “DemoWidgets” to upload the HMI display on QEC-M-070T. Below are the complete steps for uploading the sketch to QEC-M-070T:

1. Open the 86Duino Coding IDE 500+ and connect the QEC-M-070T.
2. Open the Tools menu from Menu bar -> Select the assigned COM port for QEC-M-070T -> Select “QEC M070T” for the board.
3. Select “Examples” in the File menu, go to LVGL86 -> LVGL and open the “DemoWidgets”.



4. Click the “Upload” button to compile your code and upload it to the QEC-M-070T.
5. Then, you can see the DemoWidgets example on QEC-M-070T after the upload is complete.



There are 3 files in the 'DemoWidgets' example.

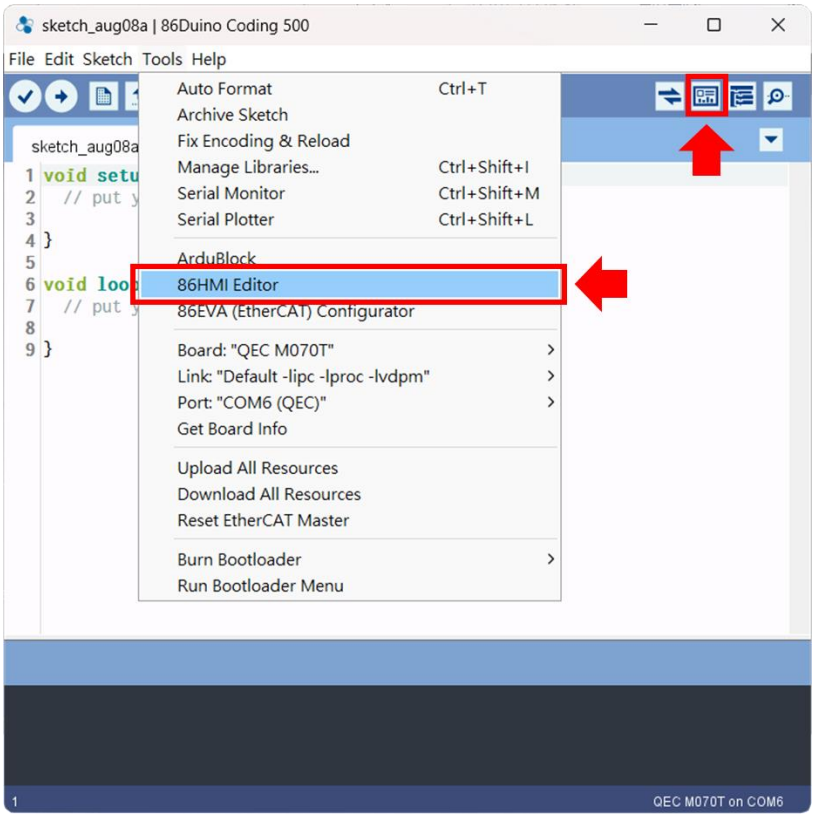
1. DemoWidgets:
   ***.ino** files are your main application files.
2. lv_demo_widgets.c:
   ***.c** files are our C source code files in which our code is written and created by the user.
3. lv_demo_widgets.h:
   ***.h** files are header files that are prewritten for our compiler.

3.6.3 Using the Graphical HMI editor: 86HMI Editor

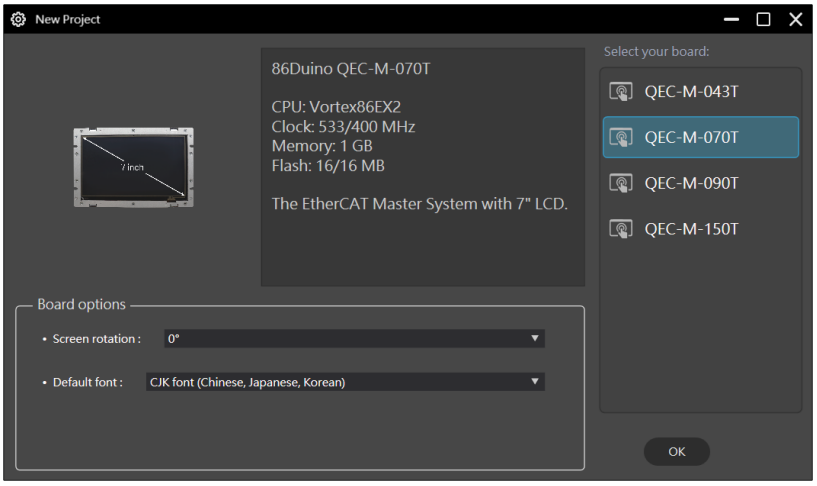
86HMI Editor is an easy-to-use HMI editor that can be used to create a customized HMI quickly. Use the Auto Code Generation function in 86HMI Editor to generate HMI APIs (Application Programming Interface), thus achieving the effect of creating HMIs without writing programs.

Below are the complete steps to design UI to QEC-M-070T using 86HMI:

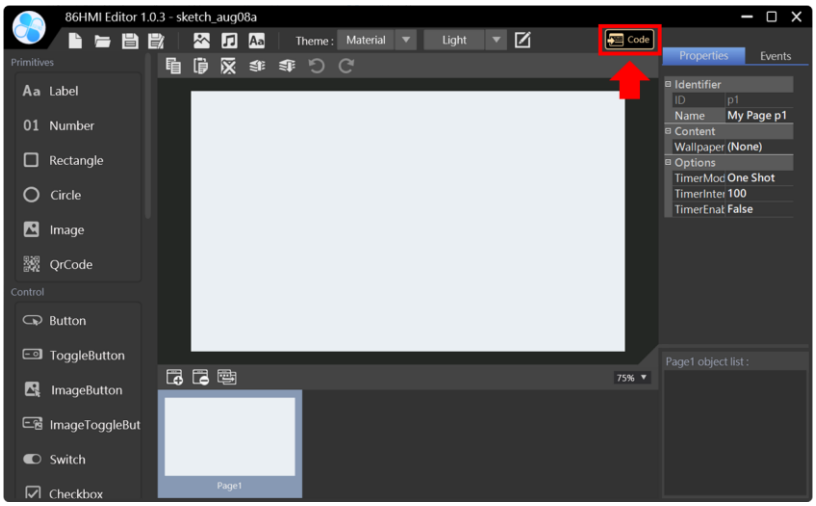
1. Open 86HMI tool via 86Duino Coding IDE 500+.



2. Choose QEC-M-070T.



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.



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

And after you finish uploading, you can see the user interface you just designed on QEC-M-070T.

For more information on how 86HMI can be developed, please write to info@icop.com.tw, call your nearest ICOP Branch, or contact our Worldwide Official Distributor.

Ch. 4

Software Function

4.1 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.
Please visit qec.tw for 86Duino Coding IDE 500+ details.

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

4.2 EtherCAT Function List

The list below introduces the functions of our QEC series products.
Please visit EtherCAT Master API User Manual for API Function details.

4.2.1 EthercatMaster Class Functions

Initialization Functions:

- `begin()` - EtherCAT Master Initialize.
- `end()` - Shutdown EtherCAT Master.
- `isRedundancy()` - Check EtherCAT uses redundancy or not.
- `libraryVersion()` - The version of EtherCAT Master library.
- `firmwareVersion()` - The version of EtherCAT firmware.

Access to slave information Functions:

Get Slave Information (SlaveCount, VendorID, ProductCode, RevisionNumber, SerialNumber, AliasAddress) on EtherCAT bus.

- `getSlaveCount()` - Get EtherCAT Slave Count Number on EtherCAT bus.
- `getVendorID()` - Get the EtherCAT Slave Vendor ID on EtherCAT bus.
- `getProductCode()` - Get the EtherCAT Slave Product Code on EtherCAT bus.
- `getRevisionNumber()` - Get the EtherCAT Slave revision Number on EtherCAT bus.
- `getSerialNumber()` - Get the EtherCAT Slave Serial Number on EtherCAT bus.
- `getAliasAddress()` - Get the EtherCAT Slave Alias Address on EtherCAT bus.
- `getSlaveNo()` - Obtain the sequential ID of the specified slave based on the alias address/vendor ID/product number/revision number/serial number.

Control Functions:

- `start()` - Start EtherCAT Master. All slaves will enter OP state if successful.
- `stop()` - Stop EtherCAT Master.
- `getSystemTime()` - Get system time of current cycle. (The unit is nanosecond)
- `getWorkingCounter()` - Get working counter of current cycle.
- `getExpectedWorkingCounter()` - Get expected working counter. (Fixed value)

Access to master information Functions:

Get Master information, including System/Peripheral power voltage or current.

- `getSystemPowerVoltage()` - Get the System Power Voltage Value of Master. (unit: V)
- `getSystemPowerCurrent()` - Get the System Power Current Value of Master. (unit: A)
- `getPeripheralPowerVoltage()` - Get the Peripheral Power Voltage Value of Master. (unit: V)
- `getPeripheralPowerCurrent()` - Get the Peripheral Power Current Value of Master. (unit: A)

Advanced EtherCAT Master Functions:

The following APIs are still under development and are not recommended for use.

- `attachCyclicCallback()` - Register Cyclic Callback Function.
- `detachCyclicCallback()` - Unregister Cyclic Callback Function.
- `attachErrorCallback()` - Register Error Callback Function.
- `detachErrorCallback()` - Unregister Error Callback Function.

4.2.2 EthercatDevice Class General Functions

Initialization Functions:

- `attach()` - Specify the EC-Slave number and mount it on the EC-Master.
- `detach()` - Uninstall the slave object.

Access to slave information Functions:

- `getVendorID()` - Get EtherCAT Slave Vendor ID.
- `getProductCode()` - Get the EtherCAT Slave Product Code.
- `getRevisionNumber()` - Get the EtherCAT Slave revision Number.
- `getSerialNumber()` - Get the EtherCAT Slave Serial Number.
- `getAliasAddress()` - Get the EtherCAT Slave Alias Address.
- `getSlaveNo()` - Get EtherCAT Slave Number on EtherCAT bus.
- `readSII()` – Read EEPROM.
- `readSII8()` – Read 8-bit EEPROM. (uint8_t)
- `readSII16()` – Read 16-bit EEPROM. (uint16_t)
- `readSII32()` – Read 32-bit EEPROM. (uint32_t)
- `writeSII()` - Write EEPROM.
- `writeSII8()` - Write 8-bit EEPROM. (unit8_t)
- `writeSII16()` - Write 16-bit EEPROM. (uint16_t)
- `writeSII32()` - Write 32-bit EEPROM. (unit32_t)

Process Data Objects (PDO) Functions:

- `pdoBitWrite()` - Write Bit of Process Data Output.
- `pdoBitRead()` - Read Bit of Process Data Input.
- `pdoGetOutputBuffer()` - Get Slave Process Data Output Pointer.
- `pdoGetInputBuffer()` - Get Slave Process Data Input Pointer.
- `pdoWrite()` - Write Slave Process Data Output.
- `pdoWrite8()` - Write 8-bit Slave Process Data Output. (unit8_t)
- `pdoWrite16()` - Write 16-bit Slave Process Data Output. (unit16_t)
- `pdoWrite32()` - Write 32-bit Slave Process Data Output. (unit32_t)
- `pdoRead()` - Read Slave Process Data Input.
- `pdoRead8()` - Read 8-bit Slave Process Data Input. (unit8_t)
- `pdoRead16()` - Read 16-bit Slave Process Data Input. (unit16_t)
- `pdoRead32()` - Read 32-bit Slave Process Data Input. (unit32_t)

CANopen over EtherCAT (CoE) Functions:

- `sdoDownload()` - (CoE) Write the object to EtherCAT Slave device.
- `sdoDownload8()` - (CoE) Write the 8-bit object to EtherCAT Slave device. (unit8_t)
- `sdoDownload16()` - (CoE) Write the 16-bit object to EtherCAT Slave device. (unit16_t)
- `sdoDownload32()` - (CoE) Write the 32-bit object to EtherCAT Slave device. (unit32_t)
- `sdoUpload()` - (CoE) Read the object from EtherCAT Slave device to EtherCAT Master.
- `sdoUpload8()` - (CoE) Read the 8-bit object from EtherCAT Slave device to EtherCAT Master. (unit8_t)
- `sdoUpload16()` - (CoE) Read the 16-bit object from EtherCAT Slave device to EtherCAT Master. (unit16_t)
- `sdoUpload32()` - (CoE) Read the 32-bit object from EtherCAT Slave device to EtherCAT Master. (unit32_t)
- `getODlist()` – Get list of object dictionary indexes of specific class from target EC-Slave.
- `getObjectDescription()` – Return an object description of specific index belong to slave's OD.
- `getEntryDescription()` – Return a description of a single object dictionary Entry.

File over EtherCAT (FoE) Functions:

- `readFoE()` - Read FoE.
- `writeFoE()` - Write FoE.

Distributed Clock (DC) Functions:

- `setDc()` - Configure Distributed Clock (DC) parameters.

4.3 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:

- **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!

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.