

使用手册

QEC-PPC-M-XXXT

DM&P Vortex86EX2 处理器

具多功能集成能力的 EtherCAT MDevice。

9 英寸 / 15 英寸面板电脑，搭载四线电阻式触摸屏

（版本 1.2）

版本纪录

日期	版本	说明
2025/8/26	Version1.0	首次发行。
2026/1/9	Version1.1	• 错字修正： 第 118、126、127 与 139 页。QEC MDevice Open-frame series => QEC-PPC-M series MDevice
2026/8/24	Version1.2	• 添加 2.3 Extension I/O 章节。 • 添加 4.12 Arduino Pins Usage 章节。

版权声明

为持续改良产品，本手册内容如有变更恕不另行通知。版权所有，翻印必究。制造商对本文档中可能存在的任何错误不负任何责任，亦不承诺更新或维持本手册内容之实时性。
未经 ICOP Technology Inc. 事前书面同意，本手册之任何部分均不得以任何形式或任何方式进行重制、拷贝、翻译或传播，无论全部或部分。
©Copyright 2026 ICOP Technology Inc.
Ver.1.2 2026 年 8 月

商标声明

ICOP® 为 ICOP Corporation 之注册商标。本文档中出现的其他品牌名称或产品名称，均为其各自所有人之财产与注册商标。文中所提及之所有名称仅供识别之用。
如需更详细的信息，或您对 ICOP 的其他产品有兴趣，欢迎造访我们的官方网站：

- 全球：www.icop.com.tw
- 美国：www.icoptech.com
- 日本：www.icop.co.jp
- 欧洲：www.icoptech.eu
- 中国：www.icop.com.cn

如需技术支持或下载驱动程序，请造访我们的网站：

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

如需 EtherCAT 解决方案服务、技术支持或教学资源，以及 86Duino Coding IDE 500+ 的介绍、功能、语言、函数库等信息，请造访 QEC 网站：

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

本手册适用于 QEC 系列产品。

安全信息

- 请详阅以下安全注意事项。
- 搬运本机时请双手持拿并小心操作。
- 电源输入电压为 +12 至 +50VDC（典型值 +24VDC）。
- 将设备连接至电源插座前，请先确认电源电压是否正确。
- 为避免 QEC 设备发生触电或火灾危险，请保持干燥，并远离水气与湿气。
- 工作温度：0°C 至 +60°C / -20°C 至 +70°C（选配）。
- 使用外置存储设备作为主要操作系统存储介质时，请务必在设备电源关闭的状态下进行插拔。
- 在电源供应器未拔除前，切勿碰触未绝缘的端子或线材。
- 请将 QEC 设备放置在尽可能靠近电源插座的位置以便操作，并避免手臂与周边线材缠绕而造成拉扯。
- 若 QEC 设备将有一段时间不使用，请确实拔除电源，以免暂态过电压造成损坏。

警告!



请勿尝试打开或拆解本产品外壳。如需维修，请联系您的经销商，由合格技术人员进行处理。

目录

Ch. 1 一般信息 1
1.1 简介 2
1.1.1 QEC EtherCAT MDevice 架构 3
1.1.2 硬件平台 4
1.1.3 双系统同步 5
1.1.4 软件支持 6
1.2 规格 7
1.2.1 QEC-PPC-M-090T 7
1.2.2 QEC-PPC-M-150T 9
1.3 尺寸 11
1.3.1 QEC-PPC-M-090T 11
1.3.2 QEC-PPC-M-150T 12
1.4 TFT-LCD 面板检验标准 13
1.5 订购信息 17
1.5.1 订购料号 17
Ch. 2 硬件系统 18
2.1 一般技术数据 19
2.2 总览 20
2.2.1 USB 21
2.2.2 EtherCAT 接口 22
2.2.3 电源状态指示灯 24
2.2.4 电源连接器 25
2.2.5 USB Type C 26
2.2.6 Giga LAN 27
2.2.7 HD Audio 28
2.3 Extension I/O（选配） 29
2.3.1 Arduino 引脚定义 29
2.4 连接器接线 36
2.4.1 将线材接入连接器 36
2.4.2 将线材自连接器移除 36
Ch. 3 硬件安装 37
3.1 安装说明 38
3.1.1 一般准则 38
3.1.2 安装尺寸 42
Ch. 4 快速入门 （本章提供多语版本） 44
4.1 包装清单 47
4.2 硬件配置 48
4.3 软件 / 开发环境 49
4.4 连接至 PC 并设置开发环境 50
4.5 EtherCAT 通信 52

4.5.1 EtherCAT State Machine (ESM) 控制 52
4.5.2 SubDevice 信息 59
4.5.3 Process Data Objects (PDO) 函数 64
4.5.4 CANopen over EtherCAT (CoE) 功能 69
4.5.5 Cyclic Callback 函数 76
4.5.6 Distributed Clock (DC) 配置函数 82
4.5.7 86EVA：EtherCAT 配置工具 87
4.5.8 将 ENI 导入 QEC MDevice 101
4.6 Bootloader Menu 使用说明 108
4.6.1 打开 Bootloader Menu 109
4.6.2 General 页面 112
4.6.3 EtherCAT 页面 115
4.6.4 Security 页面 119
4.6.5 Exit 页面 125
4.7 USB 设备使用说明 126
4.7.1 示例 1：将 .txt 保存至 USB U 盘 127
4.7.2 示例 2：将 .txt 保存至 eMMC 128
4.8 Giga LAN 设置 129
4.8.1 Ethernet 通信 130
4.8.2 Modbus TCP 通信 136
4.9 HMI 设计 138
4.9.1 函数库说明 138
4.9.2 使用图形化 HMI 编辑器：86HMI Editor 139
4.10 音频使用说明 142
4.10.1 示例 1：简易音频播放器 142
4.11 故障排除 144
4.11.1 无法顺利上传代码 144
4.12 Arduino 引脚使用说明 147
4.12.1 GPIO 150
4.12.2 ADC 151
4.12.3 TX/RX 152
4.12.4 RS-485 153
4.12.5 CAN 157
4.12.6 SPI 158
4.12.7 I2C 160
4.12.8 MCM 161
Ch. 5 软件功能 163
5.1 软件说明 164
5.2 EtherCAT 函数列表 165
5.2.1 EtherCAT MDevice 166
5.2.2 EtherCAT SubDevice 168
5.2.3 QEC 系列 SubDevice 175
5.3 更多资源 177
附录 178

A1. 关于 86Duino IDE 的 ENI 配置 179
保修 182

Ch. 1

一般信息

1.1 简介

ICOP 的 QEC-PPC-M 系列（QEC-PPC-M-090T / QEC-PPC-M-150T）是集成面板与触摸 LCD 系统的 EtherCAT MDevice，专为实时、可靠且同步的工业 HMI 控制而设计。各机型皆集成高分辨率的触摸 TFT LCD（9 英寸或 15 英寸），在精巧的机构中提供坚固的外壳与强大的控制核心。

以 86Duino IDE 高效开发

开发环境采用 86Duino IDE，这是一套工业级的类 Arduino 平台，支持 EtherCAT API、图形化编程工具与高级 C/C++ 编程语言，可加速开发、降低人才招募难度并缩短上市时间。除 EtherCAT 之外，QEC-PPC-M 系列亦支持 Modbus TCP、Ethernet TCP/IP 与 CAN bus，提供完整的工业自动化解决方案。

运动控制与 I/O 控制的实时精准度

QEC MDevice 支持 PDO、CoE、FoE、Distributed Clock (DC) 等必要的 EtherCAT 功能，可弹性集成伺服驱动器、数字 I/O 等第三方 EtherCAT 设备。最小 cycle time 为 125 µs，jitter 小于 1 µs（通过 86Duino IDE），非常适合高同步需求的运动控制与 I/O 控制应用。

延伸阅读：EtherCAT MDevice Benchmark

嵌入式规格下的稳固保存、可靠 I/O 与多元连接能力

各机型皆内置 2GB SLC eMMC，确保操作系统稳定运行，并提供充裕空间存放可执行文件、HMI 图形与应用数据，文件可通过 86Duino IDE 部署。IDE 亦集成 LVGL 图形函数库，开发者可直接在 QEC 设备上打造现代化且具交互性的触摸 HMI。

除 EtherCAT 控制之外，QEC-PPC-M 系列亦可监控系统温度、电压与电流，为碳足迹分析与系统寿命推估提供实用数据。 坚固的面板设计（尺寸依机型而异）可弹性集成并支持工业应用的客制化需求。标准工作温度为 0°C 至 +60°C，并提供 -20°C 至 +70°C 的宽温选配。

每台 QEC-PPC-M 皆具备双 EtherCAT 端口（供 redundancy 使用）、一组 Giga LAN 端口、三组 USB 端口、一组 USB 调试端口（供上传 / 调试）与 HD Audio，以上皆可通过标准函数库访问。

1.1.1 QEC EtherCAT MDevice 架构

EtherCAT MDevice 软件主要分为两个部分，分别运行于 Vortex86EX2 CPU 的两个系统上，各自负责下列工作：

- **EtherCAT MDevice Library**
  - 提供 C/C++ 应用程序接口：
    - 初始化接口。
    - 配置设置接口。
    - Process Data (PDO) 访问接口。
    - CAN application protocol over EtherCAT (CoE) 访问接口。
    - File access over EtherCAT (FoE) 访问接口。
    - SubDevice Information Interface (SII) 访问接口。
    - Distributed Clocks (DC) 访问接口。
- **EtherCAT MDevice Firmware**
  - 运行 EtherCAT MDevice Core。
  - 控制 Primary/Secondary Ethernet Driver，负责发送 EtherCAT frame

这些编程为运行于 FreeDOS 操作系统，并使用 DJGPP 环境所提供的 GCC 编译器进行编译。

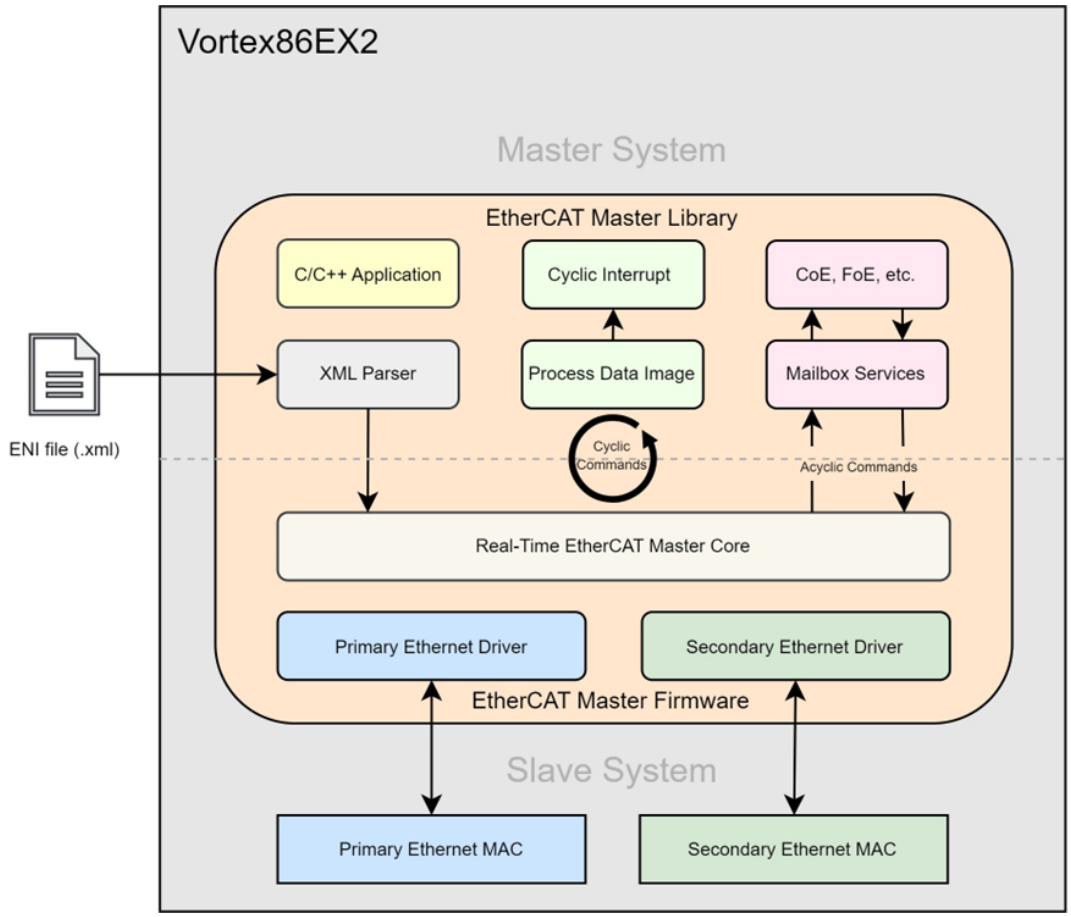
1.1.2 硬件平台

EtherCAT MDevice 软件仅能运行于 DM&P 所生产的 Vortex86EX2 CPU，该处理器采用双系统架构，分为 Master System 与 Slave System，各自运行独立的操作系统，两系统之间通过 Dual-Port RAM 与事件中断进行通信。

两者各自负责的工作如下：

- **Master System**
  - 用户的 EtherCAT 应用程序。
  - 用户的 HMI 应用程序。
  - 用户的 Ethernet 应用程序。
  - 以及其他应用。
- **Slave System**
  - 仅负责运行 EtherCAT MDevice Firmware。

由于大多数应用程序都运行于 Master System，运行于 Slave System 的 EtherCAT MDevice Firmware 不会受到其他应用程序干扰，因此得以专注运行 EtherCAT MDevice Core，确保 EtherCAT 的同步性与实时性。



1.1.3 双系统同步

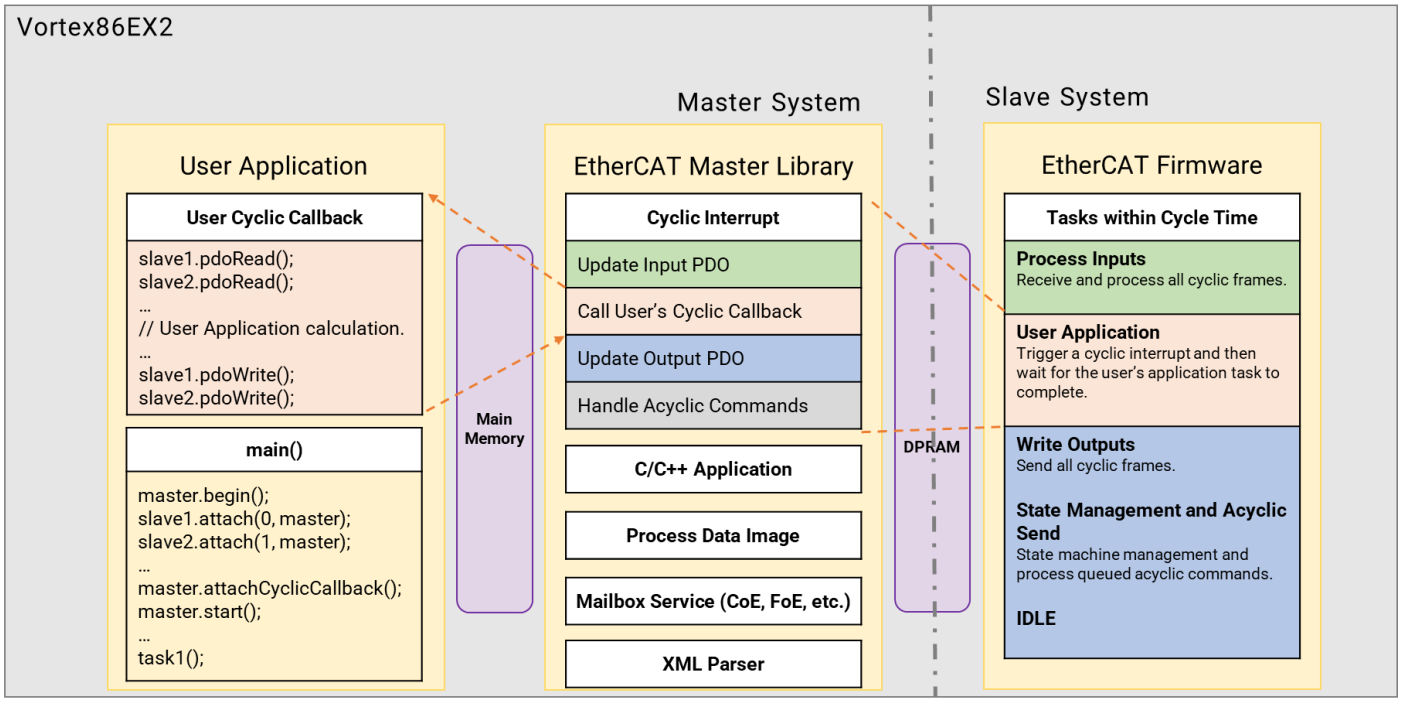
本节主要说明双系统的 PDO 数据同步机制。如下图所示，User Application 与 EtherCAT MDevice Library 区块运行于 Master System，而 Real-Time EtherCAT MDevice Core 则运行于 Slave System。

当 EtherCAT MDevice Core 进入 **Process Inputs** 阶段时，会从 Ethernet Driver 接收所有周期性 frame，并将 Input PDO 数据拷贝到 DPRAM。

进入 **User Application** 阶段后，EtherCAT MDevice Core 会对 Master System 触发 Cyclic Interrupt。Master System 收到 Cyclic Interrupt 后，随即运行 EtherCAT MDevice Library 的中断处理进程：将 Input PDO 数据由 DPRAM 搬移至主内存、调用用户注册的 Cyclic Callback、在 Cyclic Callback 结束后将 Output PDO 数据由主内存搬移至 DPRAM、处理非周期性命令，最后结束中断处理进程。此时 EtherCAT MDevice Core 的 **User Application** 阶段与中断处理进程会同时完成。

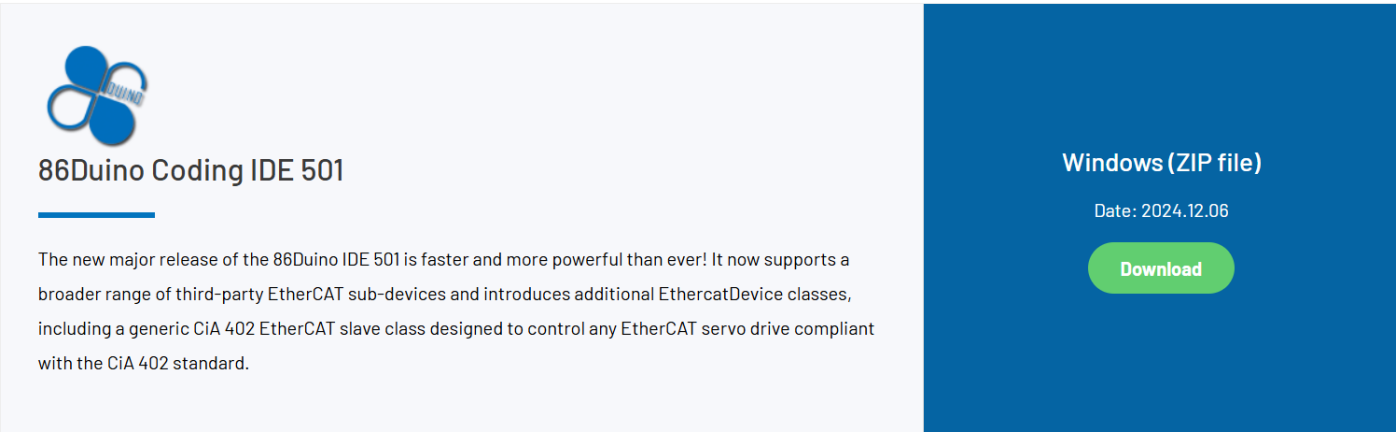
当 EtherCAT MDevice Core 进入 **Write Outputs** 阶段时，会将 Output PDO 数据由 DPRAM 拷贝到 Ethernet Driver 的 DMA 并送出 frame。

上述工作会依照所述流程周期性地循环运行，确保双系统 PDO 数据的同步。



1.1.4 软件支持

86Duino 集成开发环境（IDE）让您能轻松编写代码并上传至 86Duino 开发板与 QEC MDevice，可运行于 Windows、Mac OS X 与 Linux。此开发环境以 Java 编写，并以 Arduino IDE、Processing、DJGPP 等开源软件为基础，可由 https://www.qec.tw/software/ 下载。



QEC MDevice 的软件 86Duino IDE 另提供一套配置工具：**86EVA**，这是一款图形化界面工具，可供用户编辑 EtherCAT 网络的参数，功能如下：

- 扫描 EtherCAT SubDevice。
- 导入 ENI 文件。
- 设置 EtherCAT MDevice。
- 设置 EtherCAT SubDevice。

其他详细功能请参见 86EVA 使用手册。



1.2 规格

1.2.1 QEC-PPC-M-090T

CPU 板规格

CPU	DM&P Vortex86EX2 处理器，Master 533MHz / Slave 400MHz
内存	板载 512MB / 1GB DDRIII
存储设备	32MB SPI Flash / 2GB SLC eMMC
LAN	1Gbps Ethernet RJ45 x1 10/100Mbps Ethernet RJ45 x2（供 EtherCAT 使用）
PoE（选配）	可选配于 10/100Mbps Ethernet x 1，支持 IEEE 802.3AT (Mode B)
I/O 连接器	DC 电源输入 / 输出连接器 x1 USB (Type-C) x1（仅供上传 / 调试） USB 2.0 Host x3 HD Audio x1 RJ45 x3
通信协议	EtherCAT、Modbus、Ethernet 等
Ethernet 标准	IEEE 802.3
控制 Cycle Time	125 μs（最小值）
软件支持	86Duino Coding IDE 501+ （此开发环境以 Java 编写，并以 Arduino IDE、Processing、DJGPP 等开源软件为基础）

*MCM 信号等同于 Arduino 的 PWM 信号。

机构与环境

电源连接器	4-pin 电源输入 / 输出与 2-pin FGND
电源需求	+12 至 +50VDC 电源输入（典型值 +24VDC）
消耗功率	9.6 瓦
工作温度	0 至 +60°C / -20 至 +70°C（选配）
存储温度	-30 ~ +85°C
工作湿度	0% ~ 90% 相对湿度，无凝结
尺寸	242 x 156.5 x 44 mm
重量	1360 g
前面板防护等级	IP65
认证	CE, FCC, VCCI
内部监控	温度、电压、电流、启动时间

LCD 规格

显示类型	9 英寸 WSVGA TFT-LCD
背光模块	LED
显示分辨率	1024 (W) x 600 (H)
对比度	500: 1
显示色彩	262,144
有效显示区域	196.61 (W) x 114.15 (H) mm
可视角	垂直 120°，水平 140°
亮度 (cd/m^2)	300 nits
背光寿命	20,000 小时

触摸屏

类型	模拟电阻式
分辨率	连续式
穿透率	80%
控制器	PS/2 接口
耐用度	100 万次

1.2.2 QEC-PPC-M-150T

CPU 板规格

CPU	DM&P Vortex86EX2 处理器，Master 533MHz / Slave 400MHz
内存	板载 512MB / 1GB DDRIII
存储设备	32MB SPI Flash / 2GB SLC eMMC
LAN	1Gbps Ethernet RJ45 x1 10/100Mbps Ethernet RJ45 x2（供 EtherCAT 使用）
PoE（选配）	可选配于 10/100Mbps Ethernet x 1，支持 IEEE 802.3AT (Mode B)
I/O 连接器	DC 电源输入 / 输出连接器 x1 USB (Type-C) x1（仅供上传 / 调试） USB 2.0 Host x3 HD Audio x1 RJ45 x3
通信协议	EtherCAT、Modbus、Ethernet 等
Ethernet 标准	IEEE 802.3
控制 Cycle Time	125 μs（最小值）
软件支持	86Duino Coding IDE 501+ （此开发环境以 Java 编写，并以 Arduino IDE、Processing、DJGPP 等开源软件为基础）

*MCM 信号等同于 Arduino 的 PWM 信号。

机构与环境

电源连接器	4-pin 电源输入 / 输出与 2-pin FGND
电源需求	+12 至 +50VDC 电源输入（典型值 +24VDC）
消耗功率	13 瓦
工作温度	0 至 +60°C / -20 至 +70°C（选配）
存储温度	-30 ~ +85°C
工作湿度	0% ~ 90% 相对湿度，无凝结
尺寸	354.81 x 280.84 x 55 mm
重量	4000 g
前面板防护等级	IP65
认证	CE, FCC, VCCI

LCD 规格

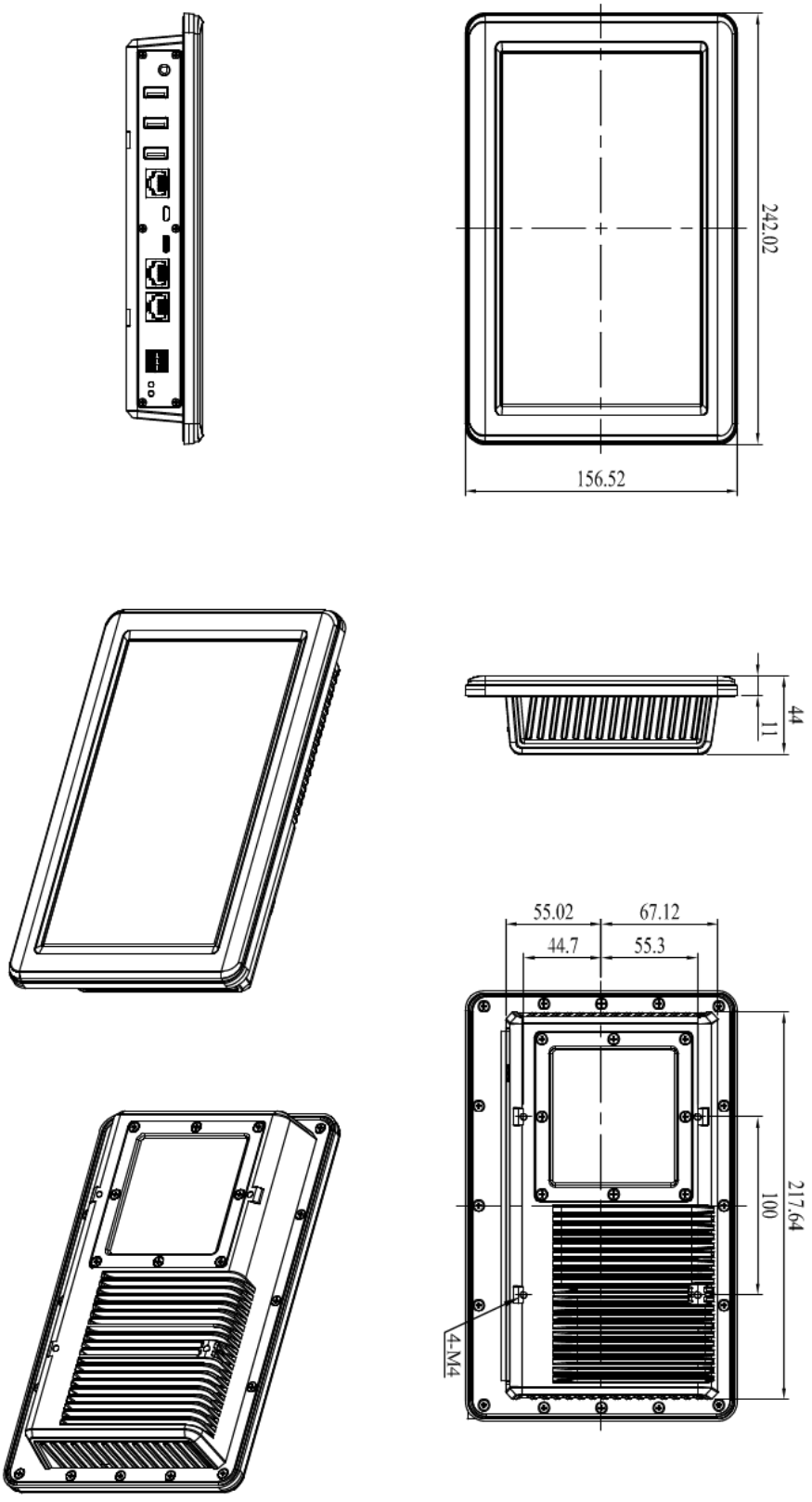
显示类型	15 英寸 XGA TFT LCD
背光模块	LED
像素	1024 (H) x 768 (V)
像素间距 (mm)	0.297 x 0.297 mm
对比度	600: 1
亮度 (cd/m^2)	350 nits
支持色彩	262,144
可视角	垂直 160°，水平 160°
背光寿命	30,000 小时

触摸屏

类型	模拟电阻式
分辨率	连续式
穿透率	80%
控制器	PS/2 接口
耐用度	100 万次

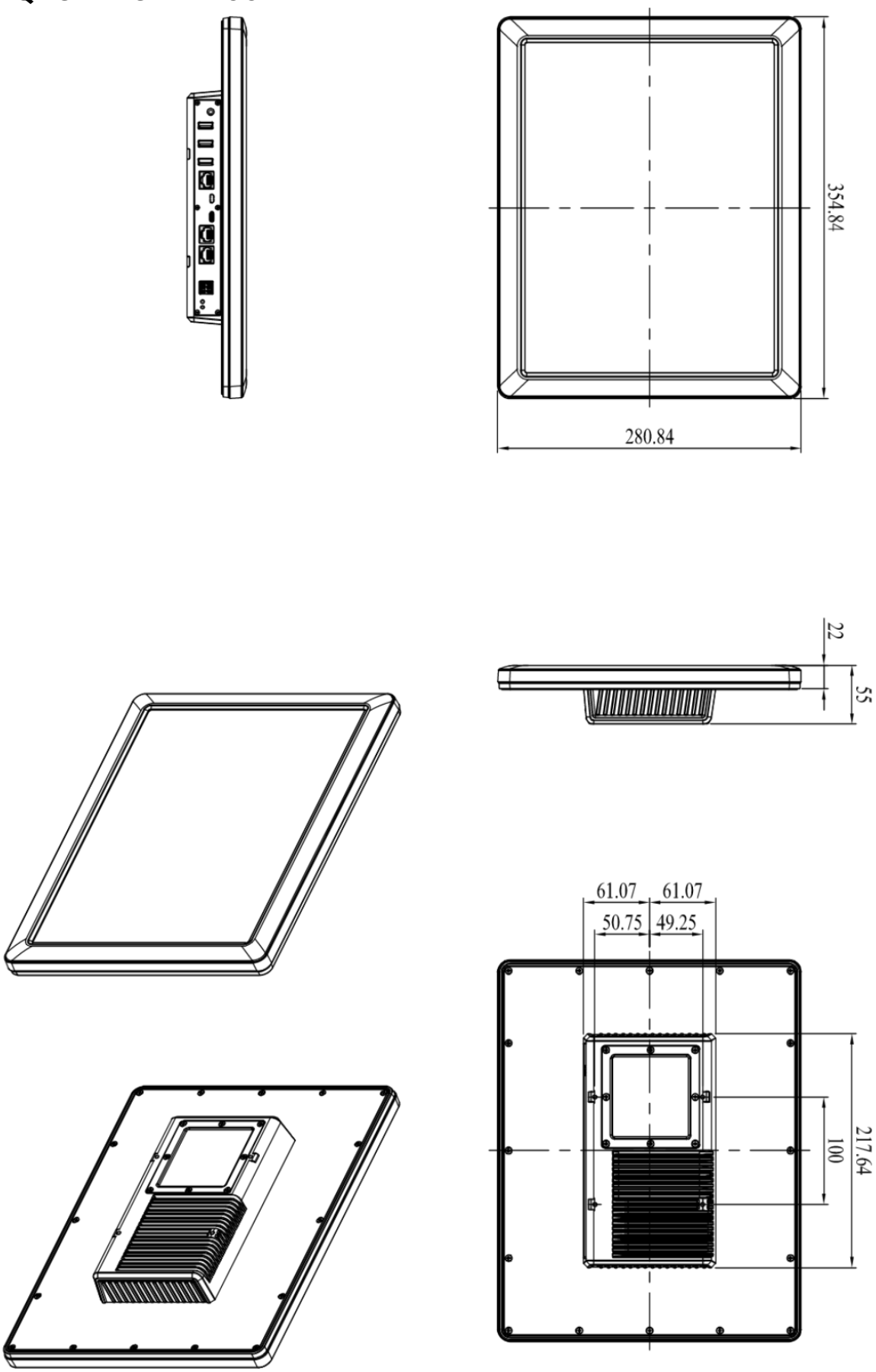
1.3 尺寸

1.3.1 QEC-PPC-M-090T



(单位：mm)

1.3.2 QEC-PPC-M-150T



(单位：mm)

1.4 TFT-LCD 面板检验标准

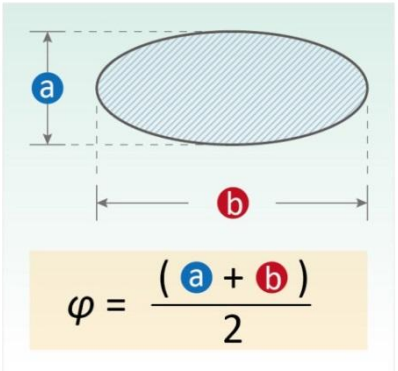
<table>
<tr><th colspan="3">缺陷类型</th><th colspan="7">允收标准</th><th>备注</th></tr>
<tr><td rowspan="11">外观缺陷</td><td rowspan="11">内部</td><td rowspan="3">点状缺陷</td><td colspan="5">φ＜0.15mm</td><td colspan="2">忽略不计</td><td rowspan="3">Note1</td></tr>
<tr><td colspan="5">0.15mm≦φ≦0.5mm</td><td colspan="2">N≦4</td></tr>
<tr><td colspan="5">0.5mm＜φ</td><td colspan="2">N=0</td></tr>
<tr><td rowspan="2">纤维状缺陷</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">偏光板气泡</td><td colspan="5">φ＜0.15mm</td><td colspan="2">忽略不计</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（亮度不均）</td><td colspan="7">通过 6% ND 滤片查看时，轻微 mura 为允收范围。</td><td></td></tr>
<tr><td rowspan="7">电性缺陷</td><td rowspan="2" colspan="2">亮点</td><td colspan="3">A 级</td><td colspan="3">B 级</td><td></td></tr>
<tr><td>C 区</td><td>O 区</td><td>合计</td><td>C 区</td><td>O 区</td><td>合计</td><td>Note3</td></tr>
<tr><td colspan="2">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">暗点</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">总点数</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">两相邻点</td><td>N≦0</td><td>N≦1 对</td><td>N≦1 对</td><td>N≦1 对</td><td>N≦1 对</td><td>N≦1 对</td><td>Note4</td></tr>
<tr><td colspan="2">三点以上相邻点</td><td colspan="7">不允收</td><td></td></tr>
<tr><td colspan="2">线状缺陷</td><td colspan="7">不允收</td><td></td></tr>
</table>

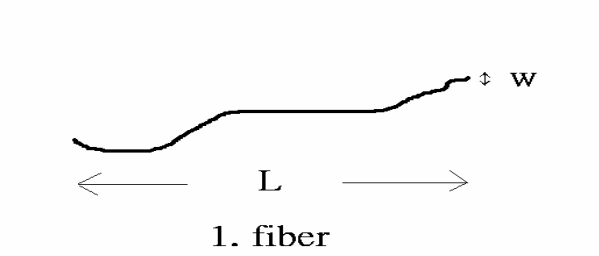
(1) 一个 pixel 由 R、G、B 三个 sub-pixel 组成。(Sub-pixel = Dot)

(2) 在 6% ND 滤片下，轻微亮点为允收范围。

(3) 若需 G0 等级（总点数 N = 0），请洽询您所属区域的业务代表。

[注 1] W：宽度 [mm]；L：长度 [mm]；N：数量；φ：平均直径。

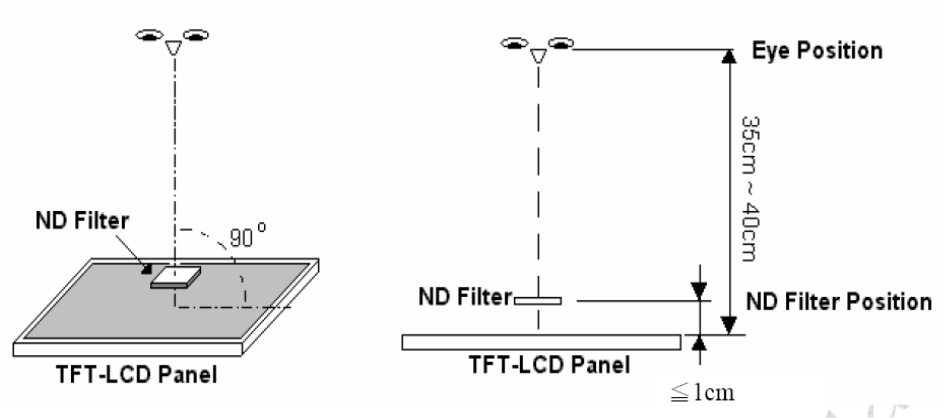




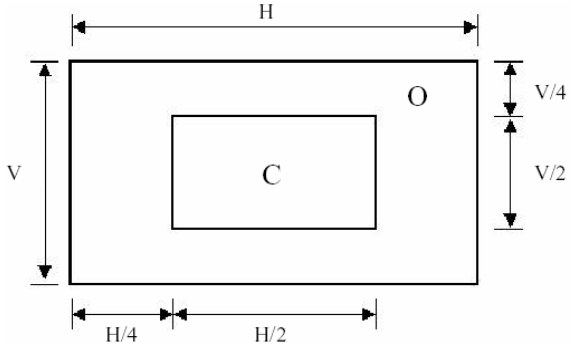
1. fiber

(a) 白点 / 黑点　(b) 偏光板气泡

[注 2] 亮点的定义系通过 6% 穿透率的 ND 滤片判定，如下所示。

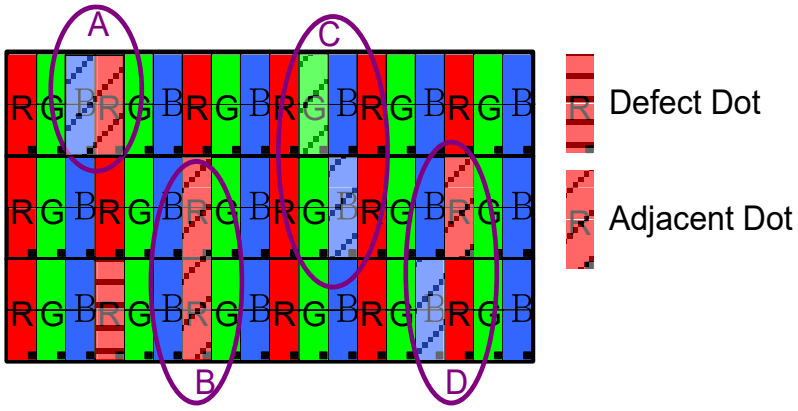


[注 3] 显示区域



C 区：显示区域中央　　**O 区**：显示区域外围

[注 4] 缺陷点与相邻点的判定方式如下。下列情形（A、B、C、D 状态）的相邻缺陷点为允收范围，包含相邻的亮点与暗点，且于总数量中计为 2 个缺陷点。



(1) 未于上述定义、但被认定为问题的缺陷，应由双方共同查看并协商认定。

位于 Black Matrix 上、显示区域之外的缺陷，不视为缺陷亦不计入。

[注 5]根据 LCD 制造商提供的技术信息，若静态画面长时间维持不变，LCD 显示器可能发生残影（image retention）。ICOP 建议客户勿让 LCD 显示同一静态画面超过 4 小时而完全不变动；若 LCD 需长时间打开，亦请启用屏幕保护程序，以避免残影问题。

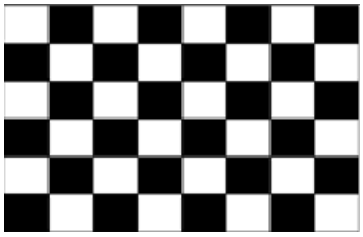
部分残影在 LCD 关闭一段时间后会自行消失，但若 LCD 已发生烙印（screen burn），部分残影将无法回复。

以下为 LCD 制造商的测试结果，供客户参考。

测试项目	条件	备注
高温运行	70℃ ;240hrs	
高温保存	80℃ ; 240hrs	
高温高湿运行	60℃ ; 90%RH ;240hrs	无凝结
低温运行	-20℃ ; 240hrs	背光模块全程打开
低温保存	-30℃ ; 240hrs	
热冲击	–30℃（0.5 小时）~ 80℃（0.5 小时）；200 循环	
残影	25℃ ; 4hrs	**注 5-1**
MTBF	20,000Hrs	

[注 5-1]

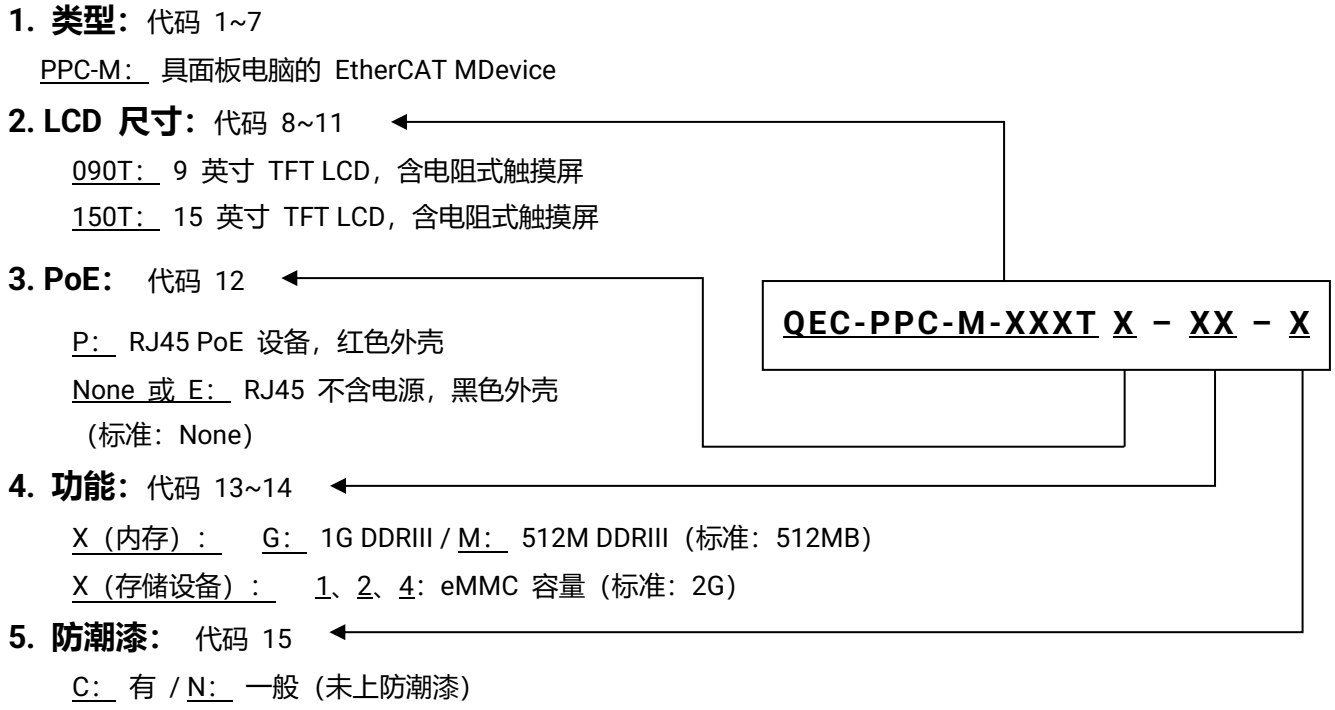
1. 残影测试条件：25 ℃± 2 ℃。
2. 以测试图样持续运行 4 小时后，立即切换为灰阶图样。
3. 5 分钟后，mura 须完全消失。





1.5 订购信息

类型	LCD 尺寸	(*以下为客制化功能，未使用的字段无须填写；若客户无特殊需求，将直接以标准料号出货，例如 **QEC-PPC-M-090T**)					
		PoE	-	功能		-	防潮漆
				内存	存储设备		
QEC-PPC-M	XXXT	X		X	X		X



1. **类型：** 代码 1~7
   PPC-M： 具面板电脑的 EtherCAT MDevice
2. **LCD 尺寸：** 代码 8~11
   090T： 9 英寸 TFT LCD，含电阻式触摸屏
   150T： 15 英寸 TFT LCD，含电阻式触摸屏
3. **PoE：** 代码 12
   P： RJ45 PoE 设备，红色外壳
   None 或 E： RJ45 不含电源，黑色外壳
   （标准：None）
4. **功能：** 代码 13~14
   X（内存）： G： 1G DDRIII / M： 512M DDRIII（标准：512MB）
   X（存储设备）： 1、2、4：eMMC 容量（标准：2G）
5. **防潮漆：** 代码 15
   C： 有 / N： 一般（未上防潮漆）

1.5.1 订购料号

- **QEC-PPC-M-090T：** 搭载 9 英寸 LCD 的 EtherCAT MDevice 面板电脑
- **QEC-PPC-M-090TP：** 搭载 9 英寸 LCD 的 EtherCAT MDevice 面板电脑 / PoE
- **QEC-PPC-M-150T：** 搭载 15 英寸 LCD 的 EtherCAT MDevice 面板电脑
- **QEC-PPC-M-150TP：** 搭载 15 英寸 LCD 的 EtherCAT MDevice 面板电脑 / PoE

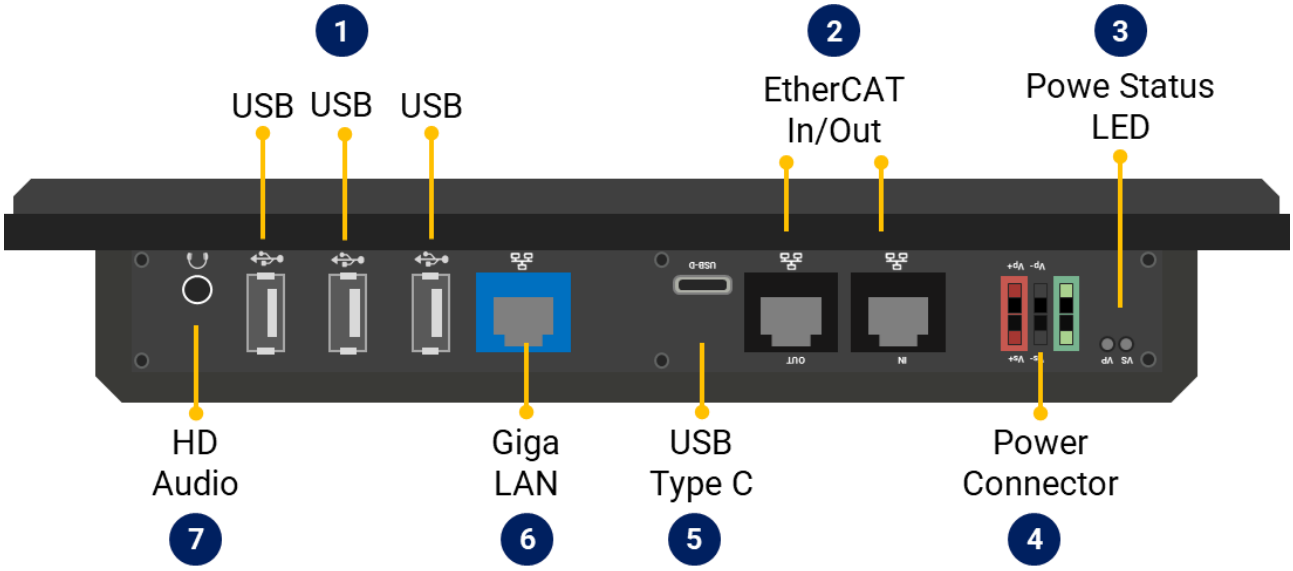
Ch. 2

硬件系统

2.1 一般技术数据

本节说明 QEC-PPC-M 各部位的功能。显示器（屏幕）规格请参见 Ch 1.2 规格。

示例以 **QEC-PPC-M-090T** 说明；**QEC-PPC-M-150T** 的功能完全相同。

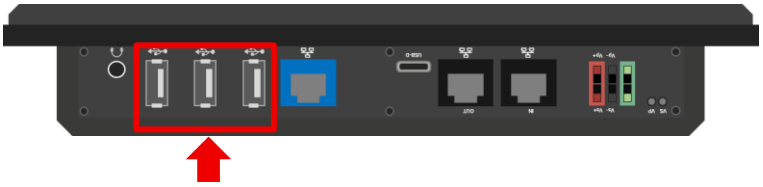


2.2 总览

编号	说明		类型叙述	引脚
1	USB		USB 2.0	-
2	EtherCAT 接口	IN	外置 RJ45 连接器（金手指）	8-pin
		OUT		8-pin
3	电源状态指示灯		外部状态指示灯	
4	电源连接器		接线端子接口	6-pin
5	USB Type-C（调试与上传端口）		USB Type-C	-
6	Giga LAN		外置 RJ45 连接器（金手指）	8-pin
7	HD Audio		外置音频连接器	-

2.2.1 USB

如图所示，共有三组支持热插拔的标准 USB 2.0 端口。



此端口可用于连接：

- USB U 盘：用于文件保存、传输或访问配置数据。
- 键盘 / 鼠标：供 HMI 显示、控制或输入使用。

***USB U 盘使用注意事项：**

- 请确认 USB U 盘已格式化为支持的文件系统（例如 FAT32）。
- 大型文件的传输可能受 USB 2.0 的速度限制。

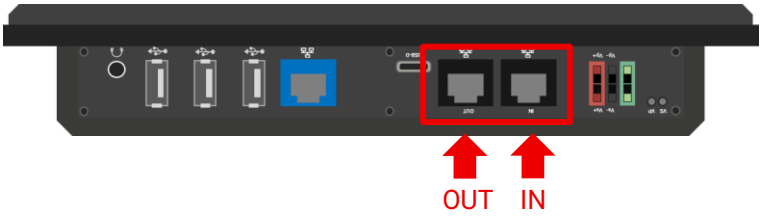
关于 USB 存储设备的更多使用细节，请参见 Ch 4.7 USB 设备使用说明章节。

***键盘 / 鼠标使用注意事项：**

QEC-PPC-M 系列默认支持 LCD 与触摸。若您需要使用键盘与鼠标功能，请于出货前告知我们。

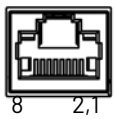
2.2.2 EtherCAT 接口

RJ45 连接器。



关于 EtherCAT 通信的更多使用细节，请参见 Ch 4.5 EtherCAT 通信章节。

2.2.2.1 EC IN

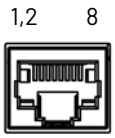


引脚	信号名称	引脚	信号名称
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

*PoE LAN 为红色外壳；一般 LAN 为黑色外壳。

*L4、L5、L7、L8 为选配引脚，供 RJ45 Power IN/OUT 使用。

2.2.2.2 EC OUT



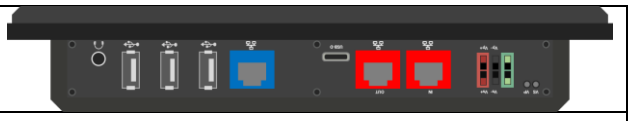
引脚	信号名称	引脚	信号名称
1	LAN2_TX+	2	LAN2_TX-
3	LAN2_RX+	4	VS+
5	VP+	6	LAN2_RX-
7	VS- (GND)	8	VP- (GND)

*PoE LAN 为红色外壳；一般 LAN 为黑色外壳。

*L4、L5、L7、L8 为选配引脚，供 RJ45 Power IN/OUT 使用。

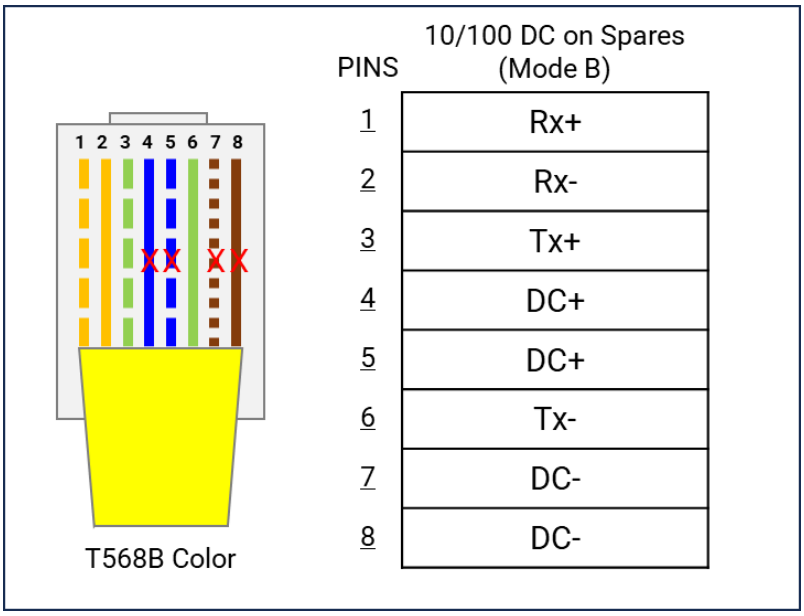
注意. QEC 的 PoE（Power over Ethernet）

在 QEC 产品的安装上，用户可轻易分辨 PoE 与非 PoE：RJ45 外壳为红色者为 PoE 型，外壳为黑色者为非 PoE 型。

	
非 PoE 型	PoE 型

PoE（Power over Ethernet）是一种通过网线供电的功能。QEC 可选配 PoE 功能以减少布线。实际应用上 PoE 需依系统设备选用，因此在评估与测试时请注意下列事项：

1. 当 PoE 设备与非 PoE 设备连接时，务必将网线的第 4、5、7、8 引脚断开（例如支持 PoE 的 QEC EtherCAT MDevice 连接第三方 EtherCAT SubDevice 时）。



2. QEC 的 PoE 功能与 EtherCAT P 不同且互不兼容。QEC 的 PoE 功能以 PoE Type B 为基础。

3. QEC 的 PoE 供电能力最高为 24V/3A。

2.2.3 电源状态指示灯

电源状态指示灯说明。



Vs 为系统电源；Vp 为周边电源与冗余电源。

电源输入为 24V（典型值）。指示灯状态可提供高 / 低电压警示。

***注意：** 请务必同时接上 Vs 与 Vp。

若出现上述任一情形，请通过 info@icop.com.tw 与 ICOP 联系，以进行诊断与处理。

2.2.4 电源连接器

Euroblock 连接器。

4-pin 电源输入 / 输出与 2-pin FGND。



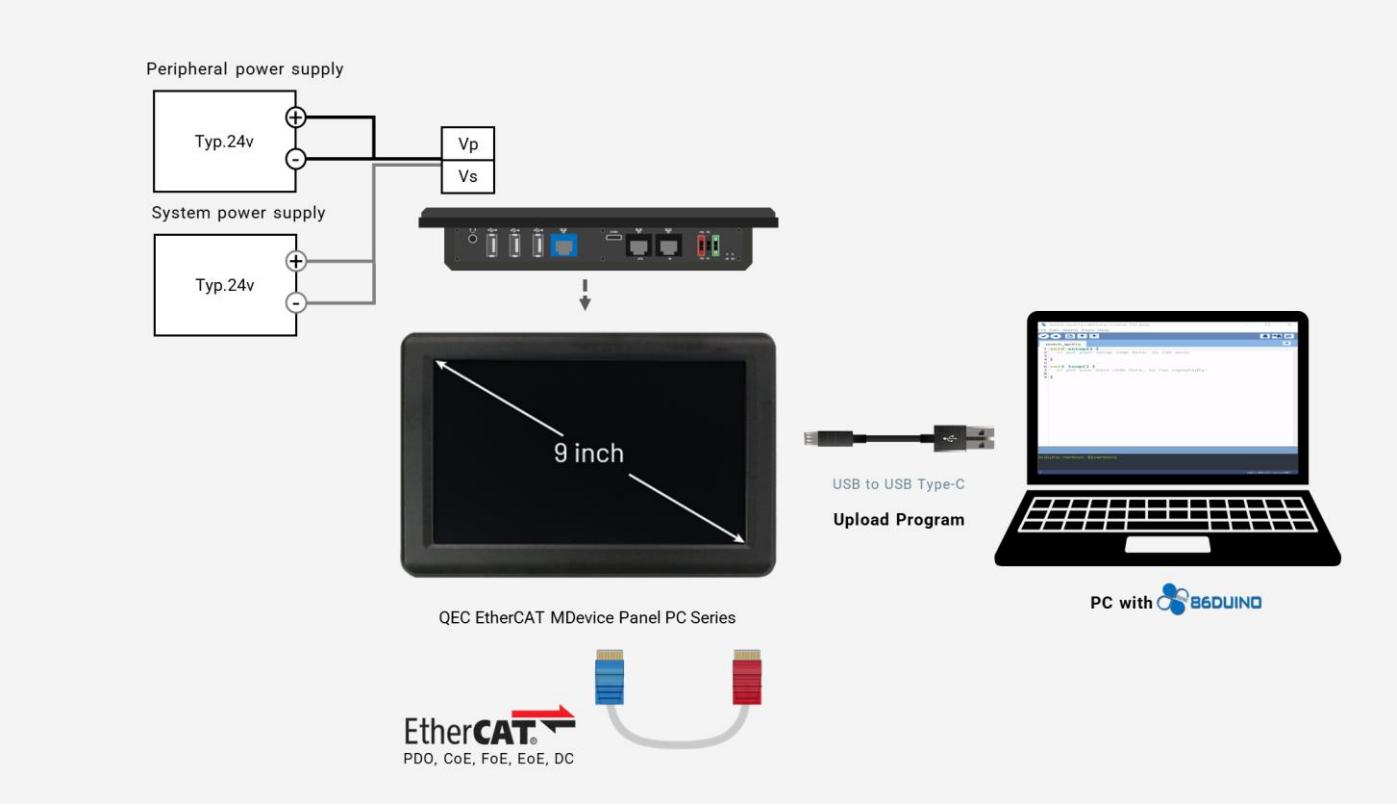
Vs 为系统电源；Vp 为周边电源与冗余电源。

	引脚	信号名称	引脚	信号名称	引脚	信号名称
	1	FGND	3	Vs- (GND)	5	Vs+
	2	FGND	4	Vp- (GND)	6	Vp+

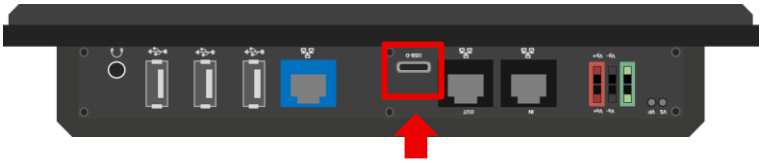
* 电源输入电压为 +12 至 +50VDC（典型值 +24VDC）

2.2.5 USB Type C

QEC-PPC-M 系列具备 USB Type-C 端口，主要用于程序上传与调试。您可使用 USB 转 USB Type-C 连接线将设备连接至电脑，以上传代码并进行系统设置。



USB Type-C 端口的位置如图所示。

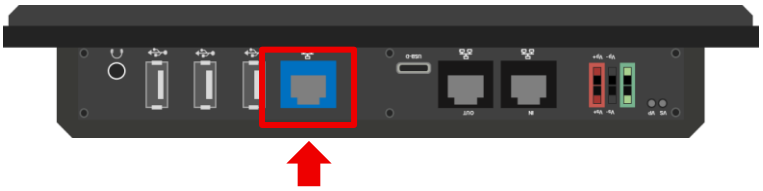


QEC-PPC-M 系列的程序编写与设置方式，请参见 Ch. 4 快速入门。

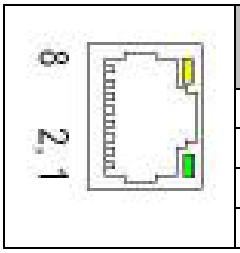
2.2.6 Giga LAN

QEC-PPC-M 系列具备**一组 Giga LAN 端口**，专供一般网络用途的对外 Ethernet 通信使用。

Giga LAN 端口为蓝色。



引脚定义：



引脚	信号名称	引脚	信号名称
L1	GTX+	L2	GTX-
L3	GRX+	L4	GTXC+
L5	GTXC-	L6	GRX-
L7	GRXD+	L8	GRXD-

Giga LAN 端口支持高速 Ethernet 对外通信。若要驱动 Giga LAN，可参考 Ethernet library 或 Modbus Library。

关于 Ethernet 或 Modbus TCP 的更多使用细节，请参见 Ch 4.8 Giga LAN 设置章节。

2.2.7 HD Audio

QEC-PPC-M 系列具备 HD Audio（Line-Out）端口，可连接喇叭或耳机等外部音频设备。

HD Audio 端口的位置如图所示。



兼容于标准 3.5mm 音频接头。

关于 HD Audio 的更多使用细节，请参见 Ch 4.10 音频使用说明章节。

2.3 Extension I/O（选配）

2.3.1 Arduino 引脚定义

如图所示，我们将 QEC MDevice PPC 系列上所有的 Arduino 引脚都保留在外壳内部。



引脚分为两类：**Arduino 标准引脚**与 **86Duino 专属引脚。**

用户可通过 **86Duino Coding IDE 软件**轻松控制这些引脚。若要驱动 Arduino 引脚，可参考 Libraries 与 Language 文档。

***Arduino 引脚使用注意事项：**

在 QEC 系统上使用 `digitalWrite()` 或 `pinMode()` 等 Arduino 风格函数时，必须依照 86Duino 的引脚对应表。

例如，若要控制接在 J5 Pin 3 (GP00) 的 GPIO，程序中应使用引脚编号 2，写法如下：

```
digitalWrite(2, HIGH);
```

请参考下方的 86Duino 引脚配置表，确认物理引脚位置与软件引脚编号之间的正确对应关系。

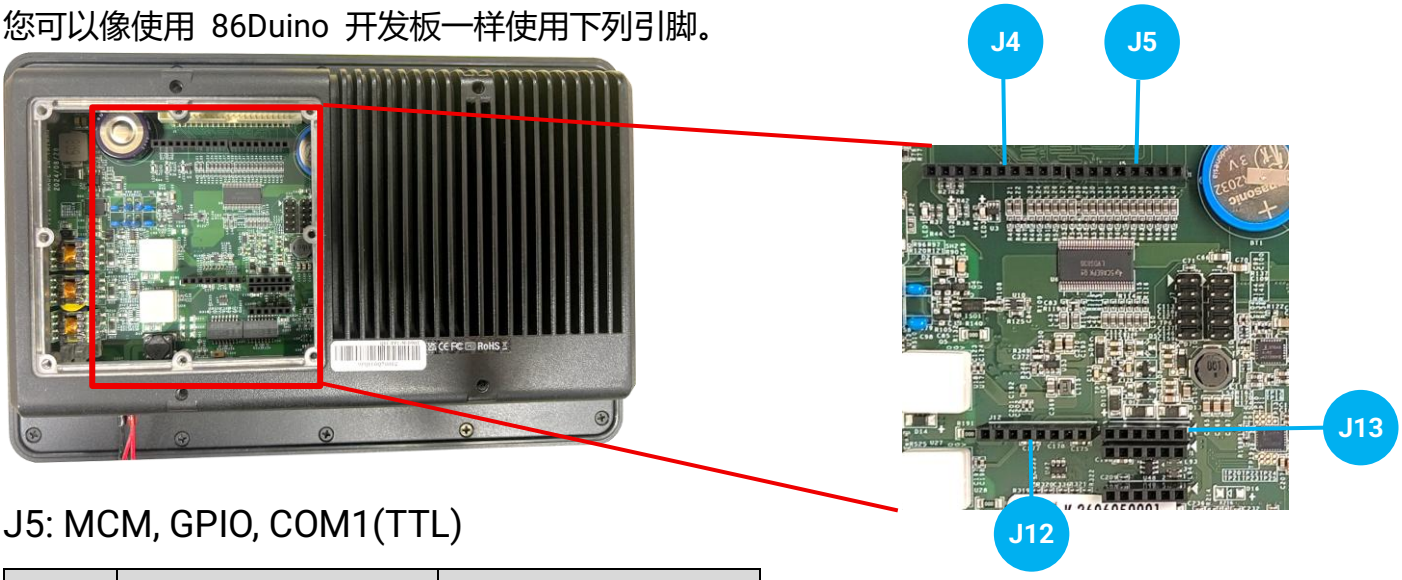
表：86Duino 引脚配置

连接器	引脚	信号	86Duino 引脚配置
J5	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
J4	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	(线材)
	10	I2C0_SCL	(线材)
J13	6	GP40ADC	14 (A0)
	5	GP41ADC	15 (A1)
	4	GP42ADC	16 (A2)
	3	GP43ADC	17 (A3)
	2	GP56ADC	18 (A4)
	1	GP57ADC	19 (A5)
J14	1	-	-
	2	GP35	23
	3	GP36	24
	4	GP37	25
	5	GND	-
	6	VCC	-
J16	1	CAN1_L	(CAN1)
	2	CAN1_H	(CAN1)
	3	GND	-
	4	CAN0_L	(CAN)
	5	CAN0_H	(CAN)
	6	VCC3	-

连接器	引脚	信号	86Duino 引脚配置
J7	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	-	-
J8	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)

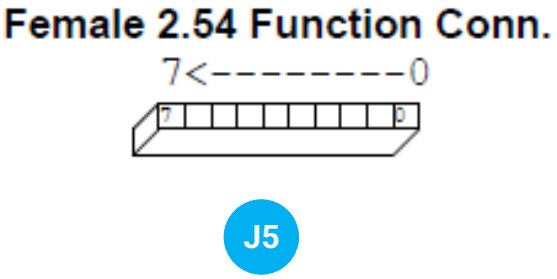
2.3.1.1 Arduino 标准引脚

您可以像使用 86Duino 开发板一样使用下列引脚。



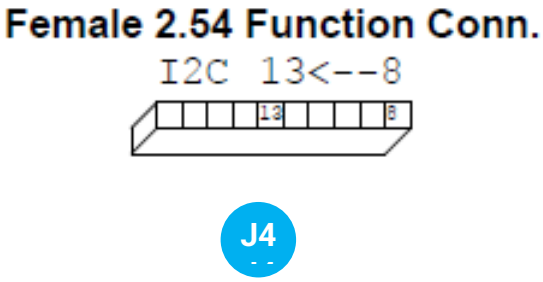
J5: MCM, GPIO, COM1(TTL)

引脚	信号名称	86Duino 引脚
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



J4: I2C0, MCM, GPIO

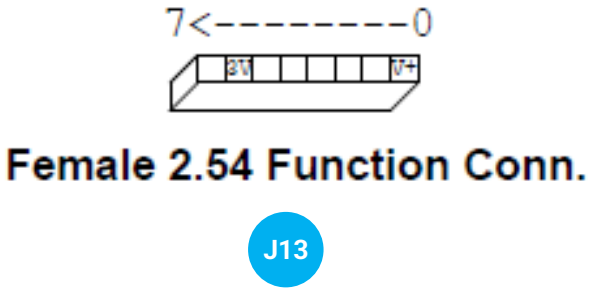
引脚	信号名称	86Duino 引脚
1	GP30	8
2	MCM-9	9
3	MCM-10	10
4	MCM-11	11
5	GP31	12
6	MCM-13	13
7	GND	-
8	-	-
9	I2C0_SDA	（线材）
10	I2C0_SCL	（线材）



*MCM 信号等同于 Arduino 的 PWM 信号。

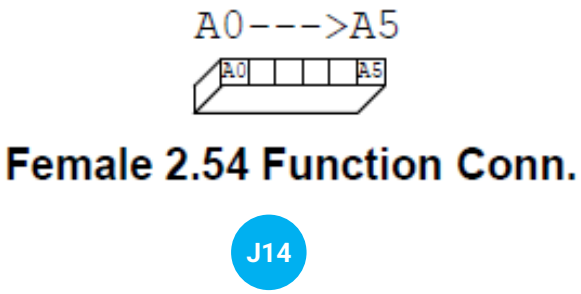
J12: Power source, RESET-

引脚	信号名称	86Duino 引脚
1	VCC	-
2	GND	-
3	GND	-
4	VCC	-
5	VCC3	-
6	RESET-	-
7	VCC3	-
8	-	-



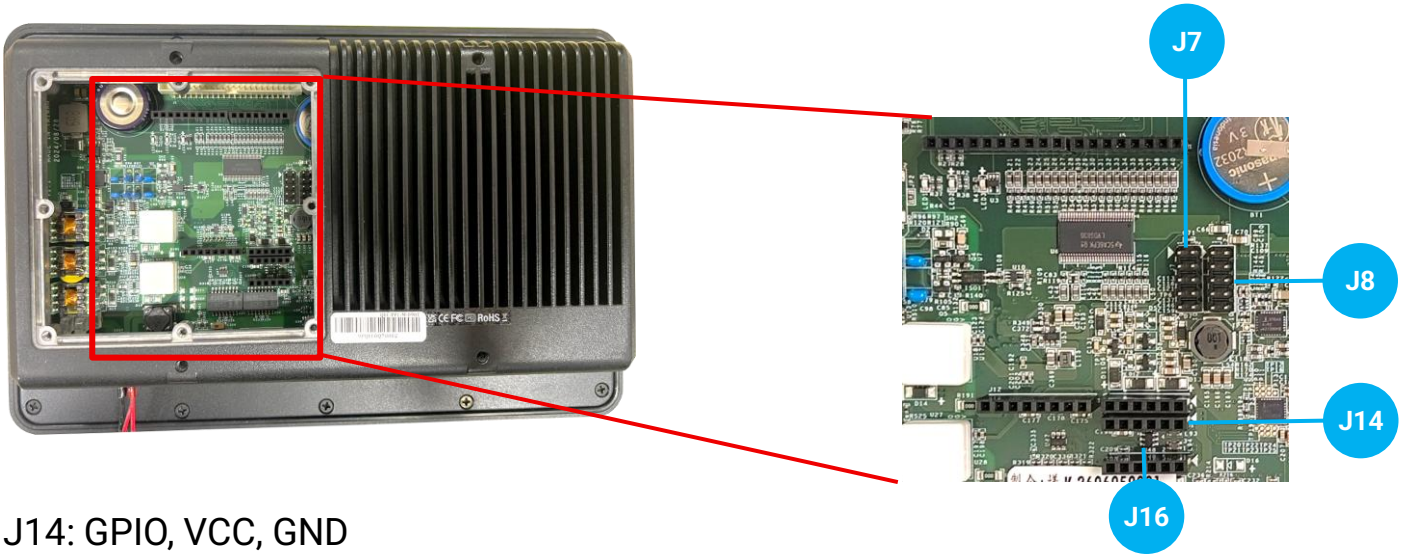
J13: ADC/GPIO

引脚	信号名称	86Duino 引脚
6	GP40ADC	14 (A0)
5	GP41ADC	15 (A1)
4	GP42ADC	16 (A2)
3	GP43ADC	17 (A3)
2	GP56ADC	18 (A4)
1	GP57ADC	19 (A5)



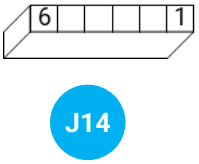
2.3.1.2 QEC 86Duino 专属引脚

QEC MDevice PPC 系列另有提供 86Duino 专属函数库所使用的引脚。



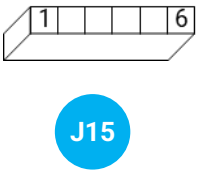
J14: GPIO, VCC, GND

引脚	信号名称	86Duino 引脚
1	-	-
2	GP35	23
3	GP36	24
4	GP37	25
5	GND	-
6	VCC	-



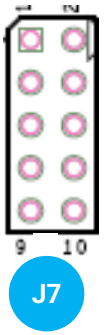
J16: CAN0 and CAN1 bus

引脚	信号名称	86Duino 引脚
1	CAN1_L	(CAN1)
2	CAN1_H	(CAN1)
3	GND	-
4	CAN0_L	(CAN)
5	CAN0_H	(CAN)
6	VCC3	-



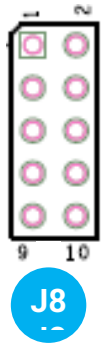
J7: SPI, RESET-

引脚	信号名称	86Duino 引脚
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	-	-



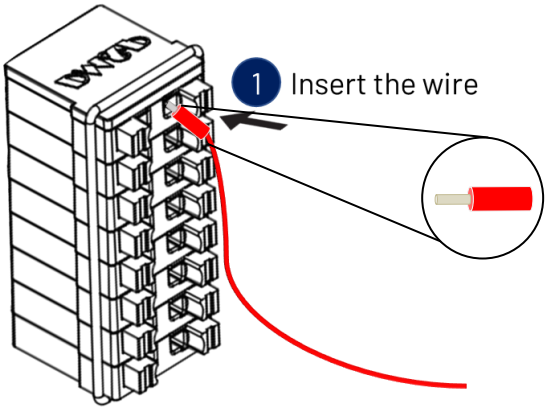
J8: SPI, RESET-, RS485+/-

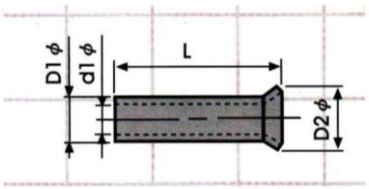
引脚	信号名称	86Duino 引脚
1	SPI1_DI	(SPI1)
2	VCC	-
3	SPI1_CLK	(SPI1)
4	SPI1_DO	(SPI1)
5	RESET-	-
6	GND	-
7	SPI1_CS	(SPI1)
8	-	-
9	RS485+	(Serial485 / SerialCOM)
10	RS485-	(Serial485 / SerialCOM)



2.4 连接器接线

2.4.1 将线材接入连接器

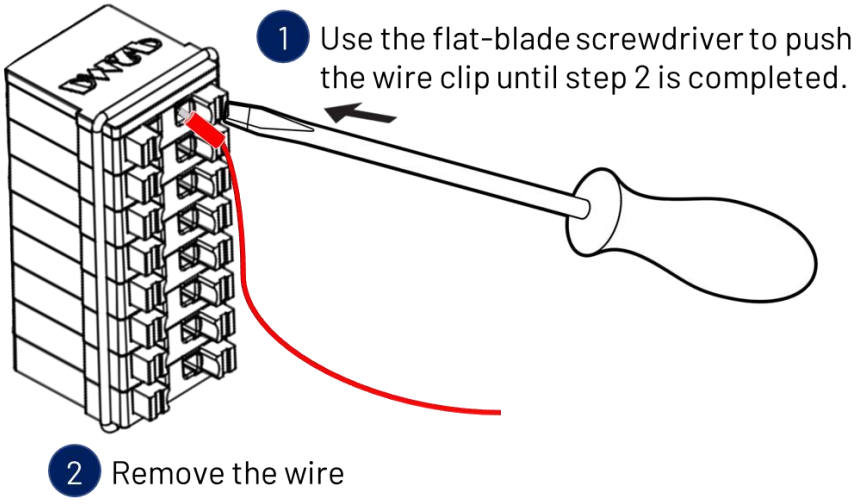




绝缘端子尺寸（mm）

位置	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

2.4.2 将线材自连接器移除



Ch. 3

硬件安装

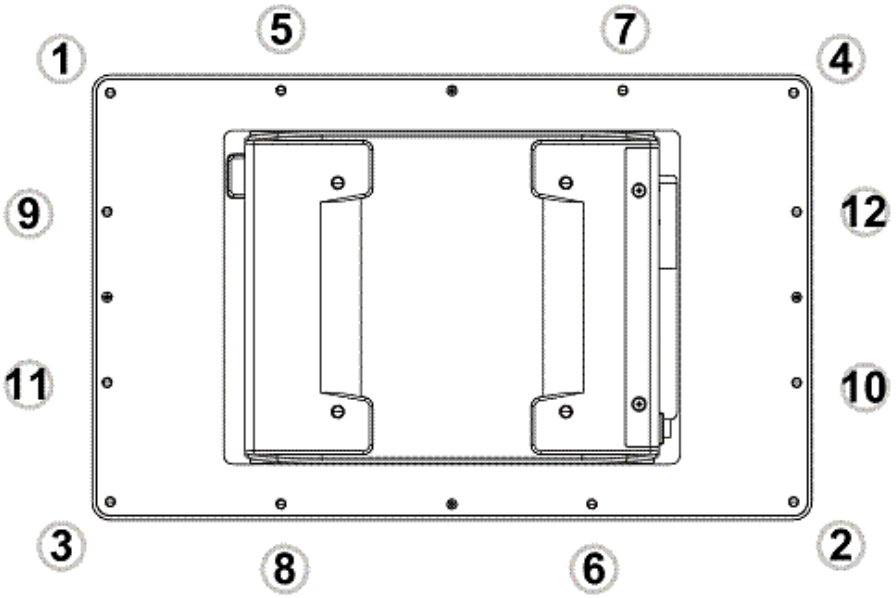
3.1 安装说明

本进程适用于搭载 9 英寸或 15 英寸触摸显示器的 QEC-PPC-M 系列机型。
安装方式。

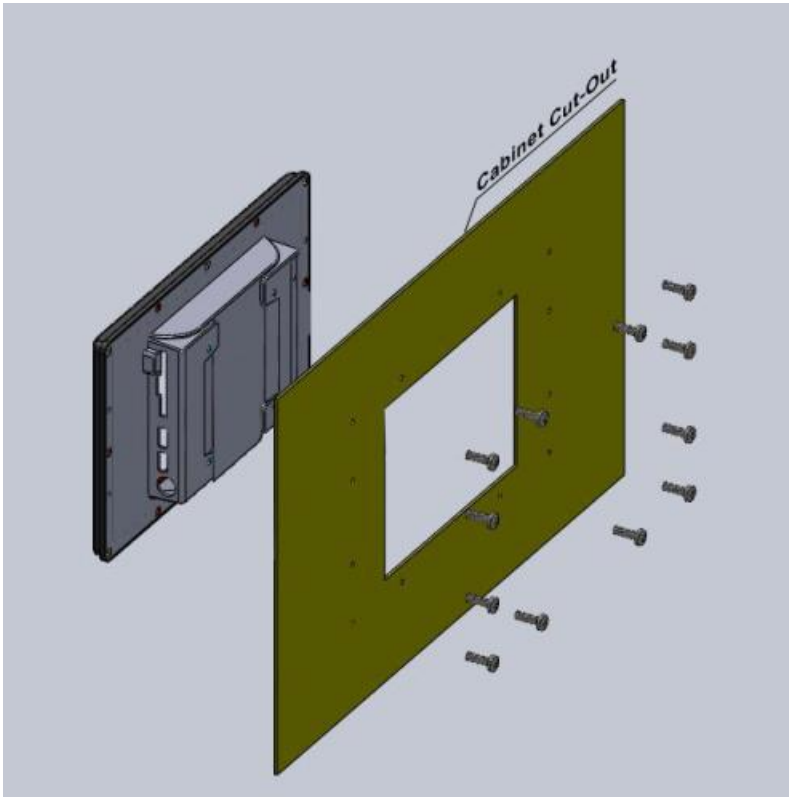
3.1.1 一般准则

采前面板嵌入式安装，含防水垫圈，使用 **M3** 螺丝固定。

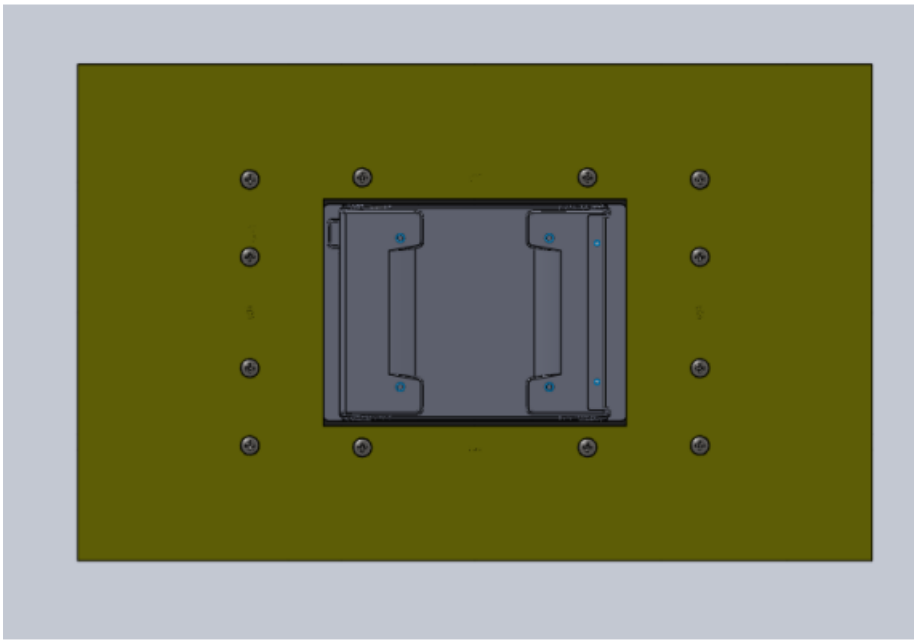
3.1.1.1 QEC-PPC-M-090T



1. 在面板上裁切安装开孔。(注 1)
2. 如下图所示，必要时请以对角顺序检查并拆下十二颗 M3 螺丝。
3. 将 QEC-PPC-M-090T 正面朝下置于干净平整的表面上。
4. 将面板开孔由 QEC-PPC-M-090T 背面套入，直到面板完全贴合于防水垫圈上，并确认螺丝孔与 QEC-PPC-M-090T 上的螺丝孔对齐。
5. 螺丝规格为 M3*L（L = 面板厚度 + 6.0mm）(注 2)
6. 将十二颗 M3 螺丝全部装入螺丝孔。(注 2)
7. 先以手指将 M3 螺丝锁上，再以 M3 螺丝刀依对角顺序锁紧（参见下图)；最大扭矩为 1.18Nm（12 kgf-cm)。

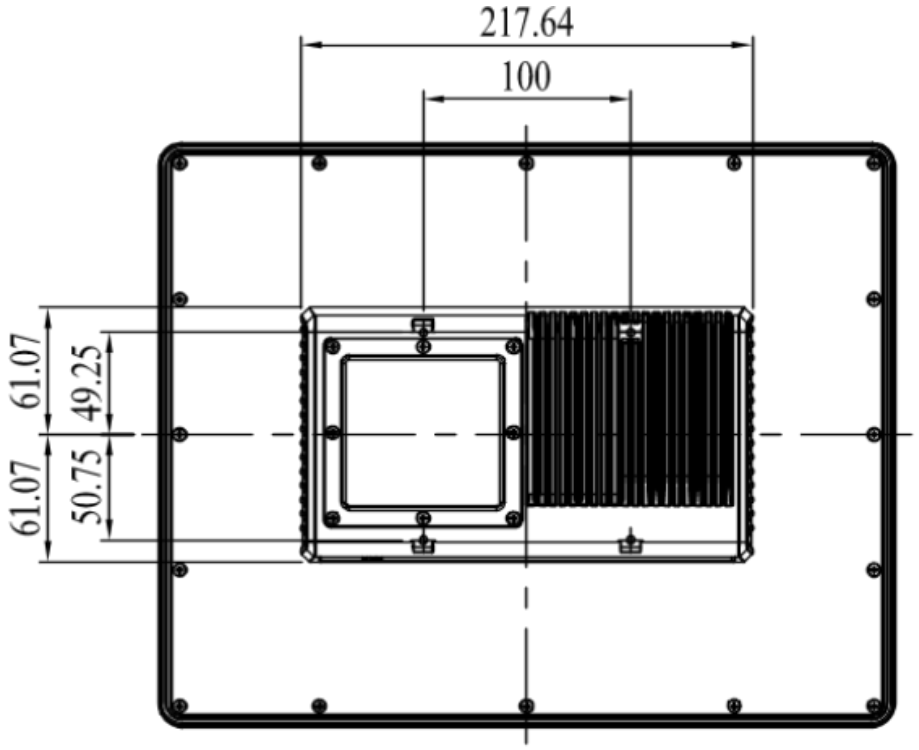


***注 1**：强烈建议由专业机械加工厂进行面板安装开孔的裁切。

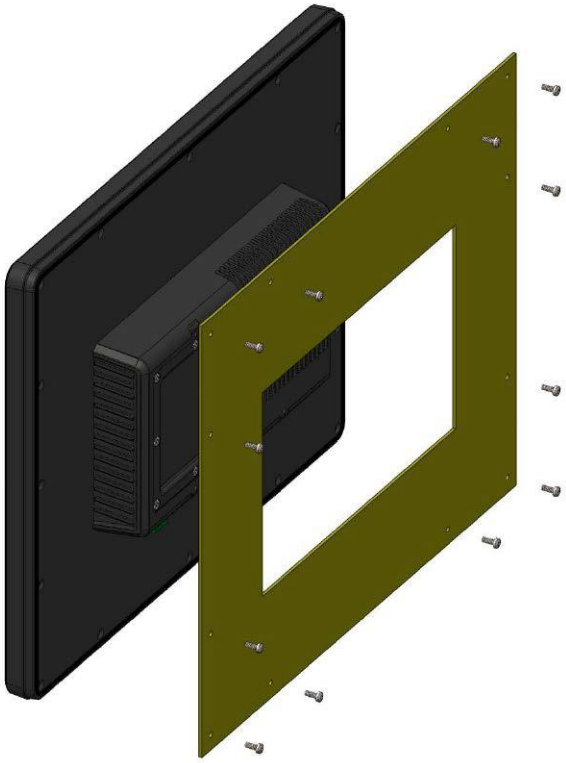


***注 2：** 十二颗 M3 螺丝的长度需依安装面板的厚度而定。例如：QEC-PPC-M-090T 标准 M3 螺丝长度为 6mm；若您的安装面板厚度为 3mm、垫圈厚度为 1mm，则须改用 10mm 的 M3 螺丝。

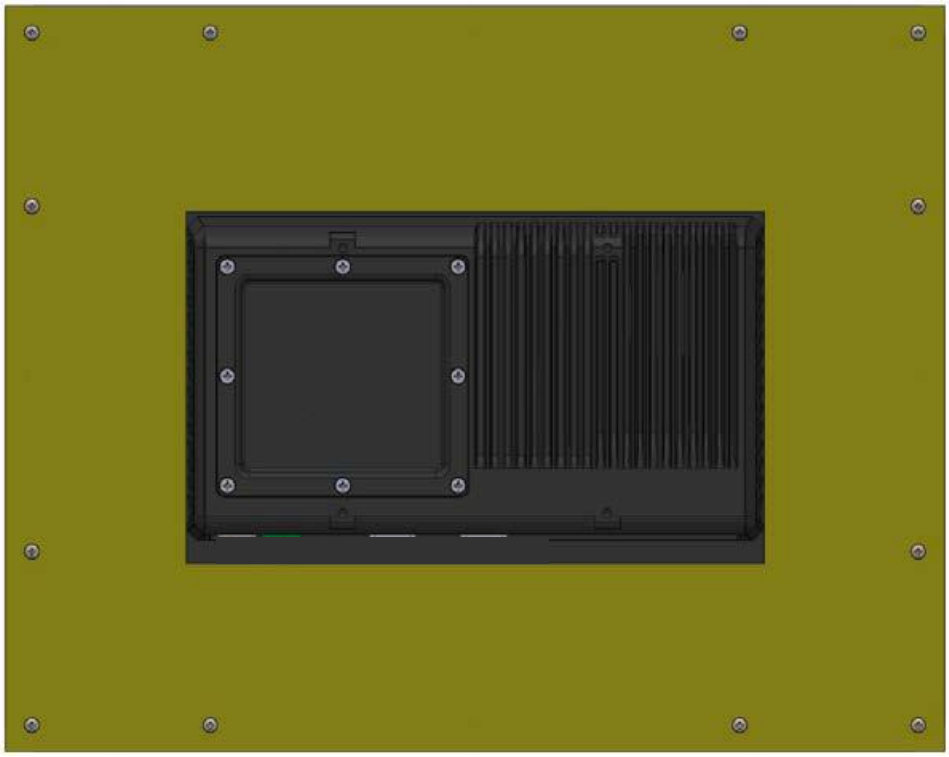
3.1.1.2 QEC-PPC-M-150T



1. 在面板上裁切安装开孔。(注 1)
2. 如下图所示，必要时请以对角顺序检查并拆下十二颗 M3 螺丝。
3. 将 QEC-PPC-M-150T 正面朝下置于干净平整的表面上。
4. 将面板开孔由 QEC-PPC-M-150T 背面套入，直到面板完全贴合于防水垫圈上，并确认螺丝孔与 QEC-PPC-M-150T 上的螺丝孔对齐。
5. 螺丝规格为 M3*L（L = 面板厚度 + 6.0mm）(注 2)
6. 将十二颗 M3 螺丝全部装入螺丝孔。(注 2)
7. 先以手指将 M3 螺丝锁上，再以 M3 螺丝刀依对角顺序锁紧（参见下图)；最大扭矩为 0.44Nm（4.5 kgf-cm)。



*注 1：强烈建议由专业机械加工厂进行面板安装开孔的裁切。

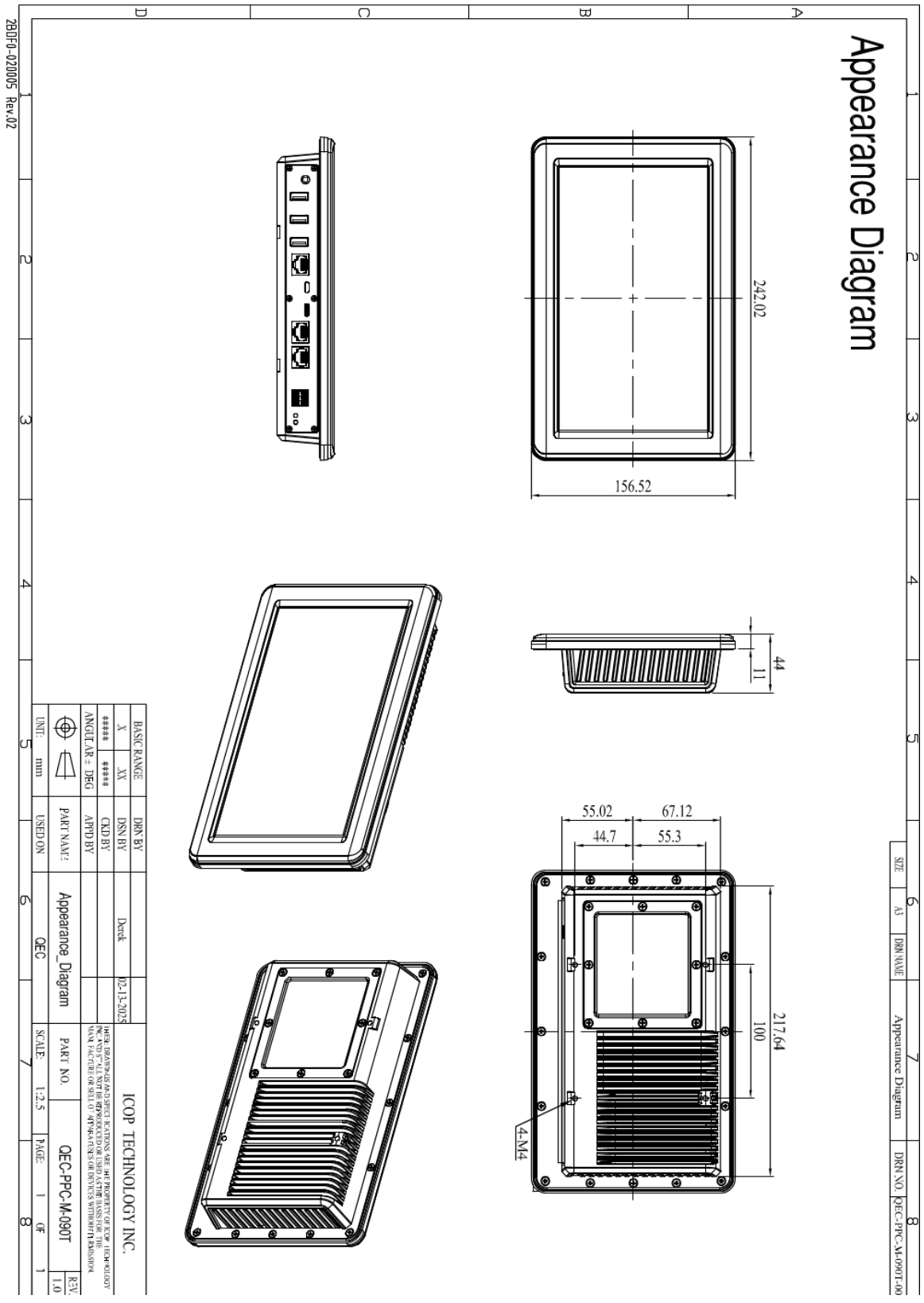


*注 2： 十二颗 M3 螺丝的长度需依安装面板的厚度而定。例如：QEC-PPC-M-150T 标准 M3 螺丝长度为 6mm；若您的安装面板厚度为 3mm、垫圈厚度为 1mm，则须改用 10mm 的 M3 螺丝。

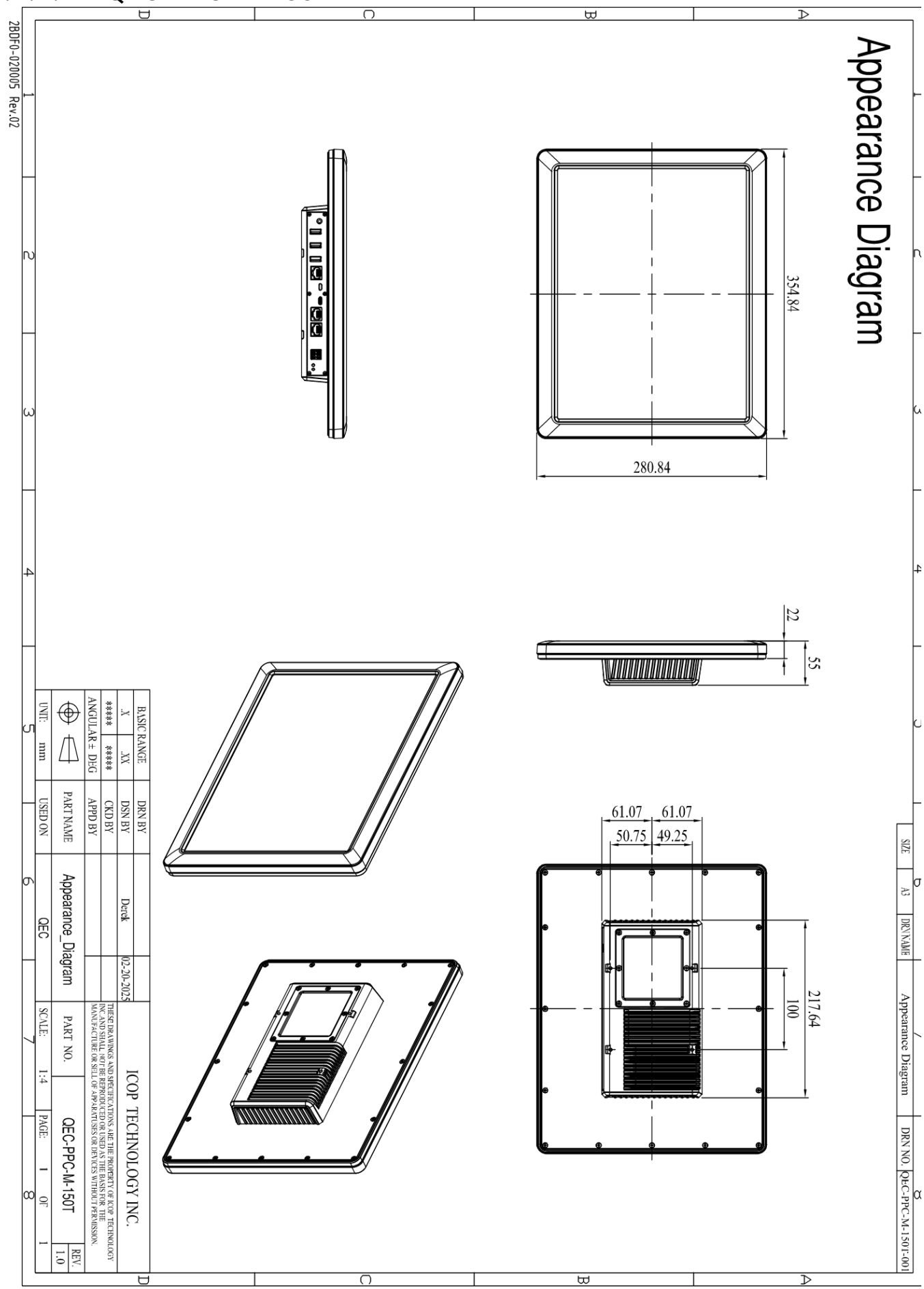
3.1.2 安装尺寸

安装开孔与尺寸总览。

3.1.2.1 QEC-PPC-M-090T



3.1.2.2 QEC-PPC-M-150T



Ch. 4

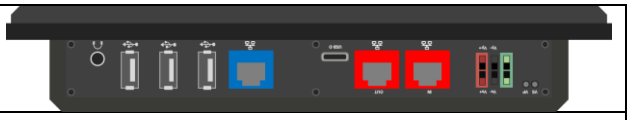
快速入门

（本章提供多语版本）

本章说明如何开始使用 QEC-PPC-M 系列及其开发软件 86Duino Coding IDE 501+。

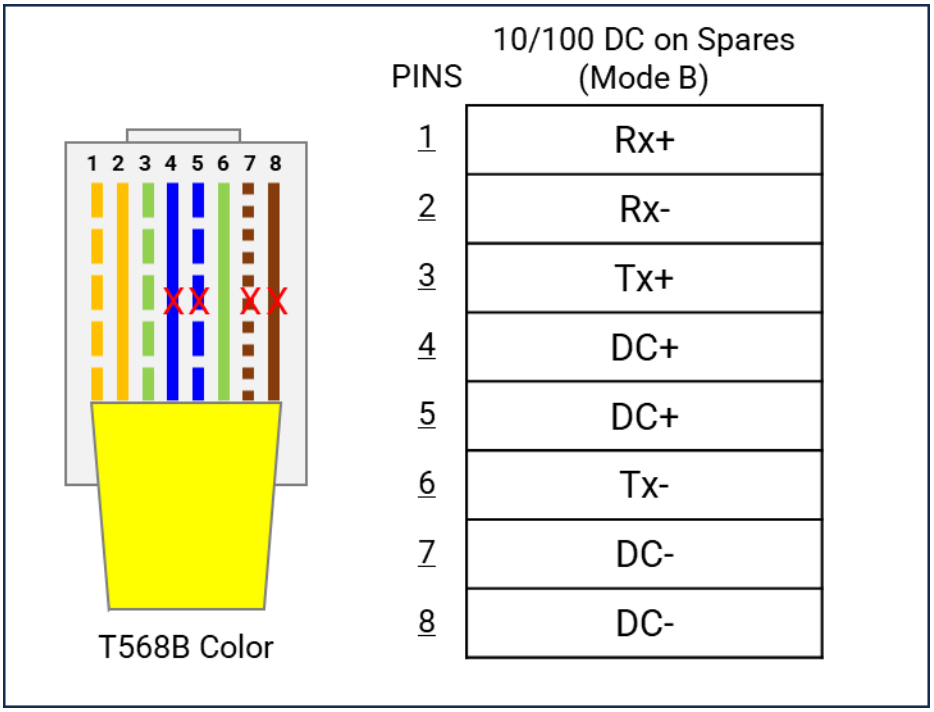
注意. QEC 的 PoE（Power over Ethernet）

在 QEC 产品的安装上，用户可轻易分辨 PoE 与非 PoE：RJ45 外壳为红色者为 PoE 型，外壳为黑色者为非 PoE 型。

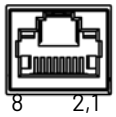
	
非 PoE 型	PoE 型

PoE（Power over Ethernet）是一种通过网线供电的功能。QEC 可选配 PoE 功能以减少布线。实际应用上 PoE 需依系统设备选用，因此在评估与测试时请注意下列事项：

1. 当 PoE 设备与非 PoE 设备连接时，务必将网线的第 4、5、7、8 引脚断开（例如支持 PoE 的 QEC EtherCAT MDevice 连接第三方 EtherCAT SubDevice 时）。



2. QEC 的 PoE 功能与 EtherCAT P 不同且互不兼容。QEC 的 PoE 功能以 PoE Type B 为基础，引脚功能如下：



引脚	信号名称	引脚	信号名称
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

* PoE LAN 为红色外壳；一般 LAN 为黑色外壳。

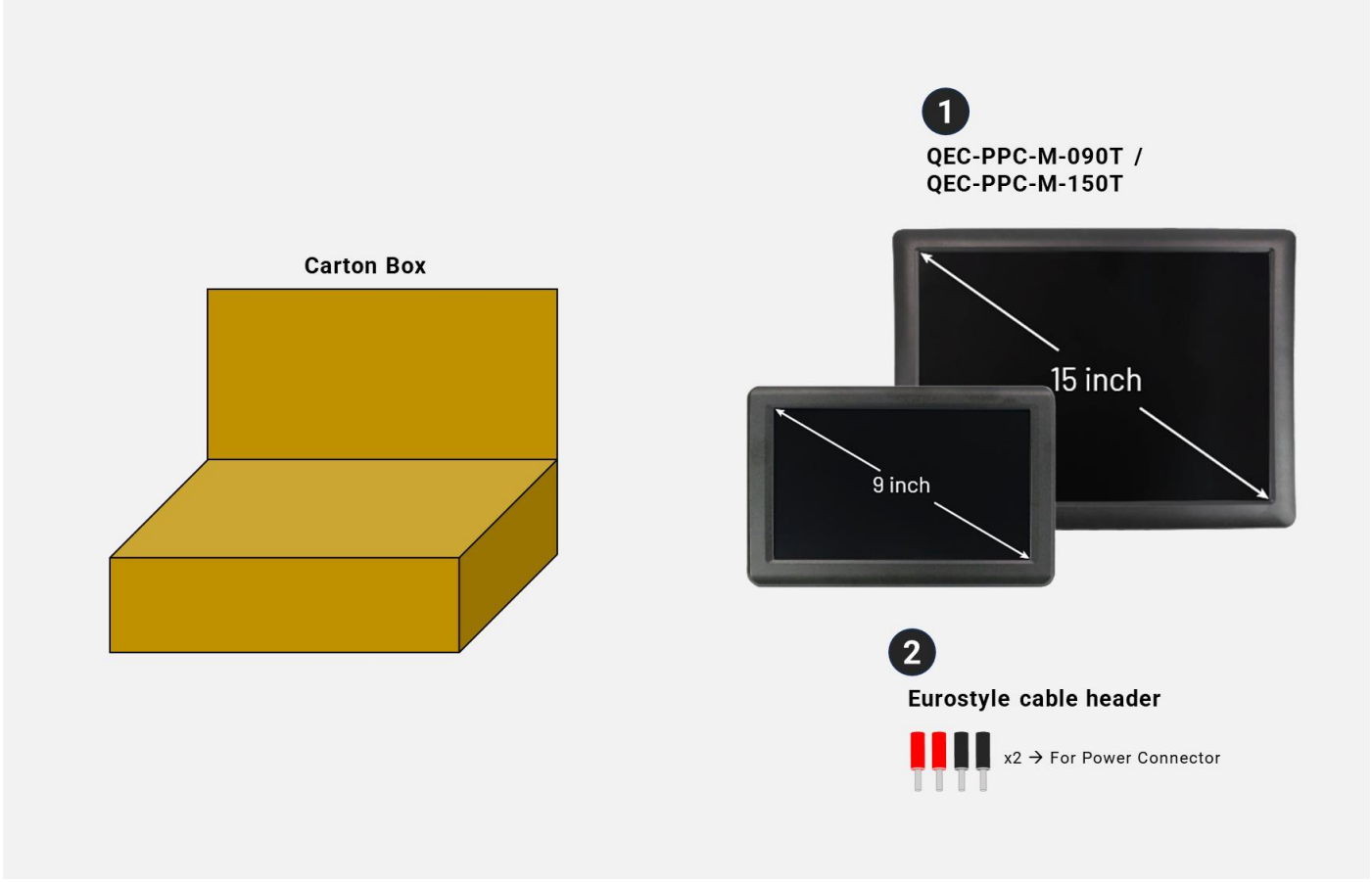
* L4、L5、L7、L8 为选配引脚，供 RJ45 Power IN/OUT 使用。

3. QEC 的 PoE 供电能力最高为 24V/3A。

4.1 包装清单

包装内含下列项目：

1. QEC-PPC-M-090T / 150T 主机 x1
2. 线材组（Euro 型连接器）

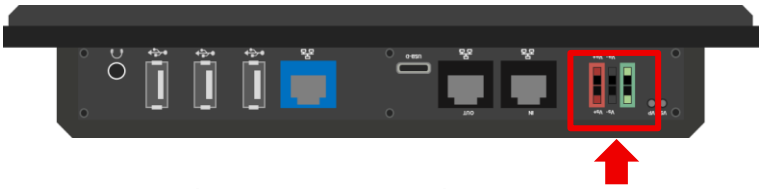


若包装清单有缺件或损坏，请与我们的销售窗口联系。此外，出货用的包材与纸箱建议保留，以供日后存放或运送使用。

4.2 硬件配置

QEC-PPC-M 系列设备出货前已预先安装好开发环境。用户需自行下载软件（请参见 4.3 软件 / 开发环境），并依照本手册完成设备设置。

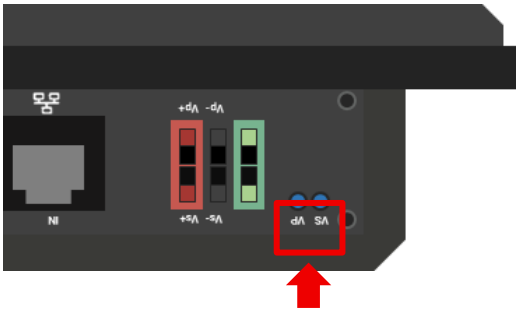
接上电源。



*注意： Vs 为系统电源；Vp 为周边电源与冗余电源。

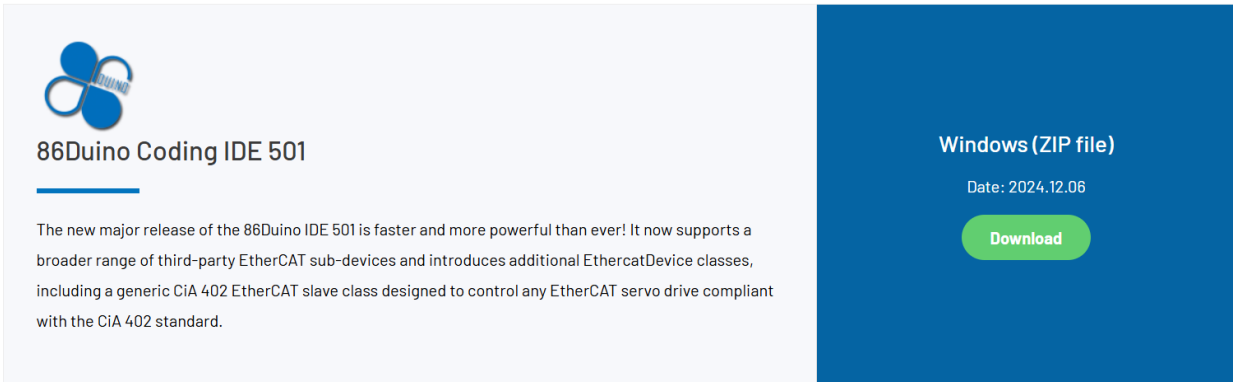
QEC MDevice 具有 Vs 与 Vp 两组电源，两者的电压需求皆为 19V 至 50V 宽电压范围。

上电后，画面即会亮起。



4.3 软件 / 开发环境

请由 https://www.qec.tw/software/ 下载 86Duino IDE。



下载后请将 zip 档解压缩，无须另行安装其他软件，直接双击 **86duino.exe** 即可启动 IDE。



***注意：** 若 Windows 出现警告消息，请点击“其他信息”，再点击一次“仍要运行”按钮。

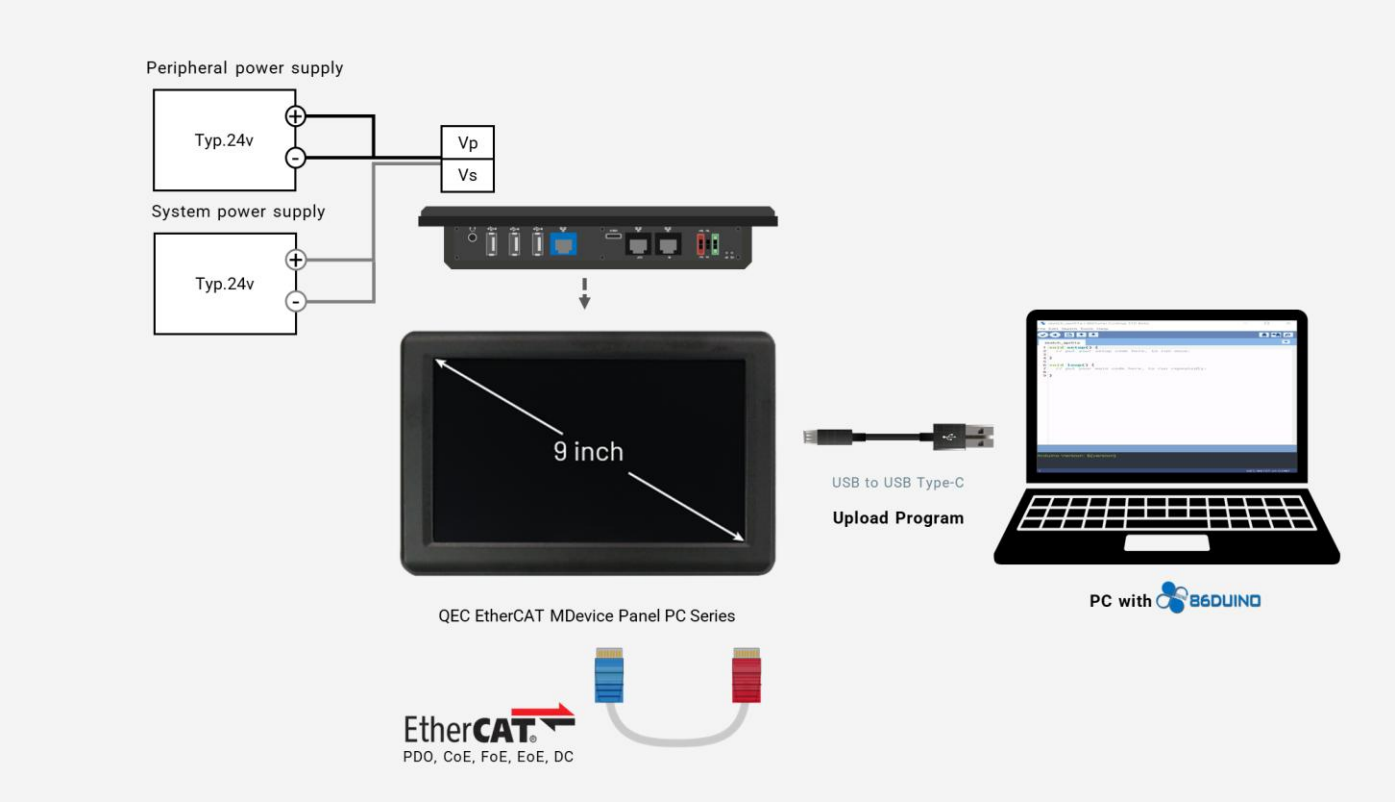
86Duino Coding IDE 501+ 的画面如下。



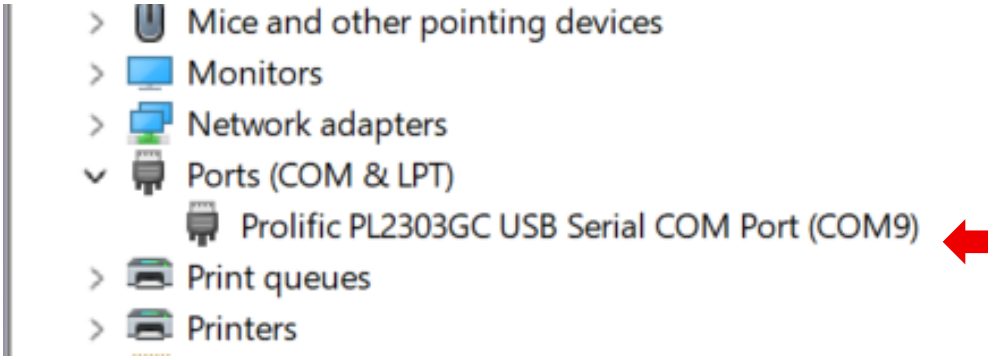
4.4 连接至 PC 并设置开发环境

请依下列步骤完成环境设置：

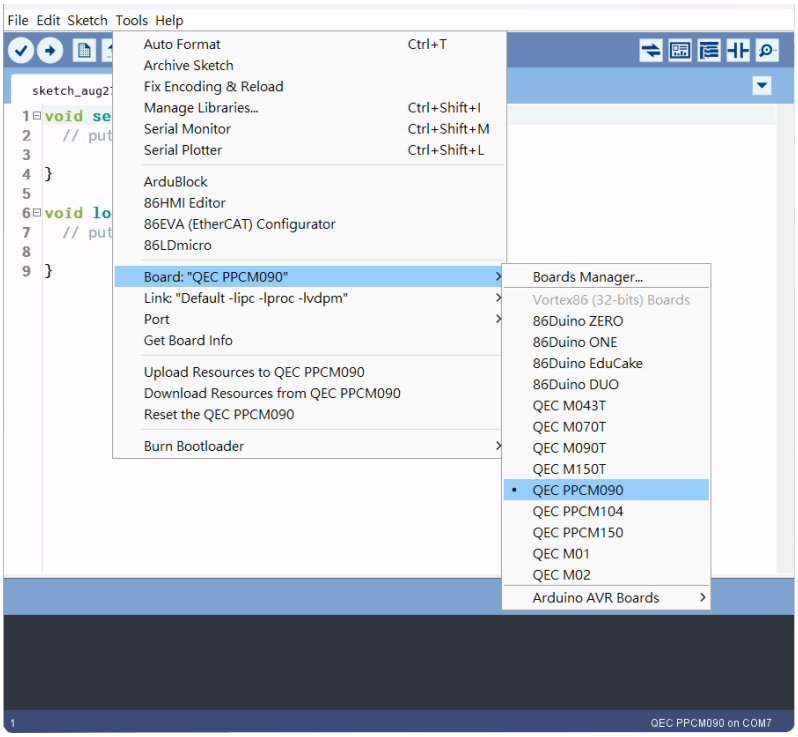
1. 使用 USB Type-C 转 USB 连接线，将 QEC-PPC-M 设备连接至已安装 86Duino IDE 的 PC。



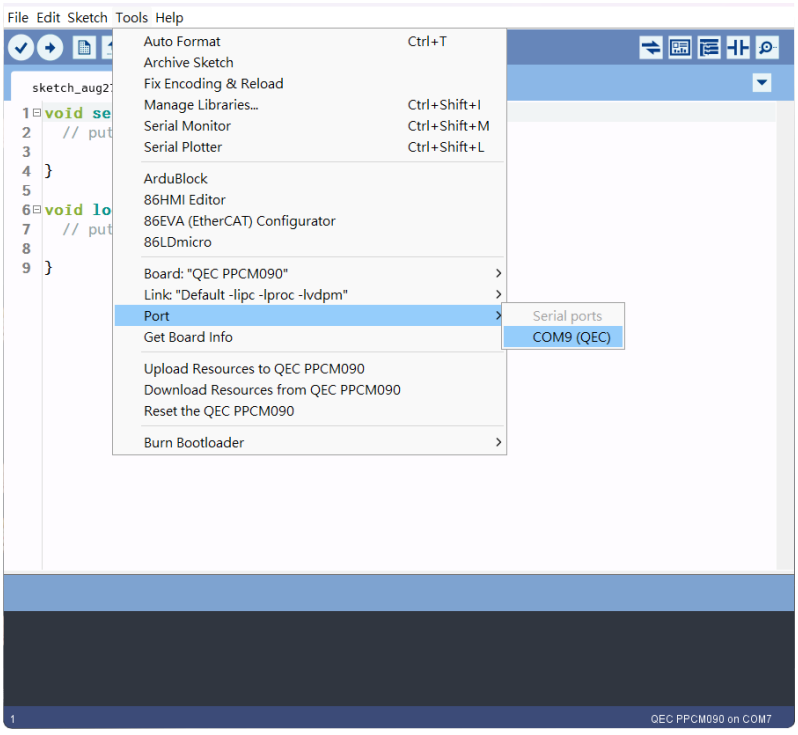
2. 打开 QEC 电源。
3. 在 PC 上打开“**设备管理员**”→“**连接端口（COM 和 LPT)**”并展开，应可看到已检测到“**Prolific PL2303GC USB Serial COM Port (COMx)**”；若未出现，则需安装对应的驱动程序。
（Windows 版 PL2303 驱动程序可由此处下载）



4. 打开 86Duino IDE。
5. 选择正确的开发板：在 IDE 菜单中选择“**Tools**”→“**Board**”→“QEC PPCM090T”（或您所使用的 QEC MDevice 型号）。



6. 选择连接端口：在 IDE 菜单中选择“**Tools**”→“**Port**”，并选择连接 QEC MDevice 的 USB 端口（本例为 COM9 (QEC)）。



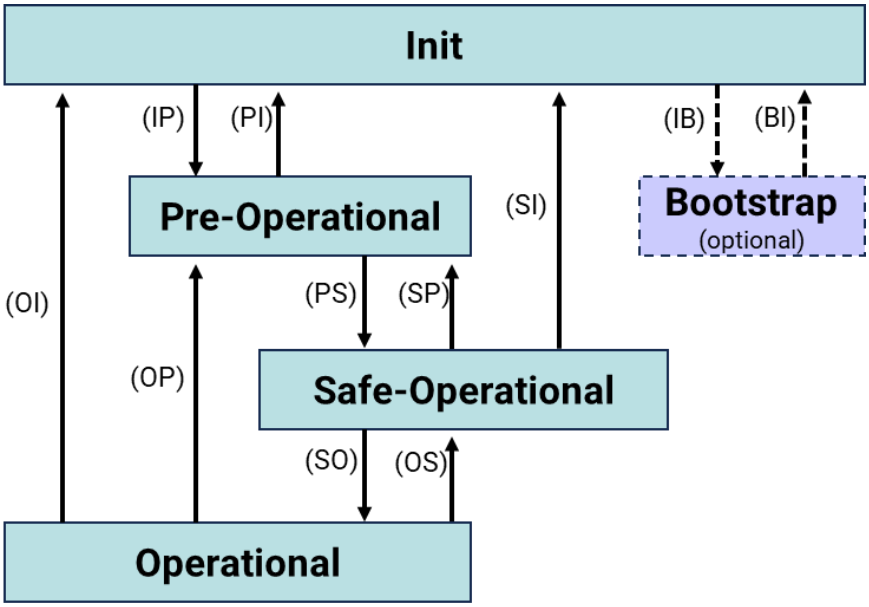
4.5 EtherCAT 通信

本节介绍通过 QEC EtherCAT MDevice 设置 EtherCAT SubDevice 的两种主要方式：直接编写代码，以及搭配 86EVA 使用代码。两种方式皆兼顾弹性与效率，您可依熟悉程度与实际需求选用。

4.5.1 EtherCAT State Machine (ESM) 控制

本节说明如何通过 QEC EtherCAT MDevice 设置 EtherCAT SubDevice 并切换其各种运行状态。理解 EtherCAT 网络中的状态转换与运行流程相当重要。

EtherCAT SubDevice 的状态由 EtherCAT State Machine (ESM) 控制。在不同状态下，EtherCAT SubDevice 可访问或运行的功能各不相同。EtherCAT MDevice 必须在每个状态下对设备送出特定命令，在 SubDevice 开机阶段尤其如此。



状态可区分如下：

- Init
- Pre-Operational
- Safe-Operational 以及
- Operational
- Boot

每个 EtherCAT SubDevice 开机后的正常状态为 OP 状态。

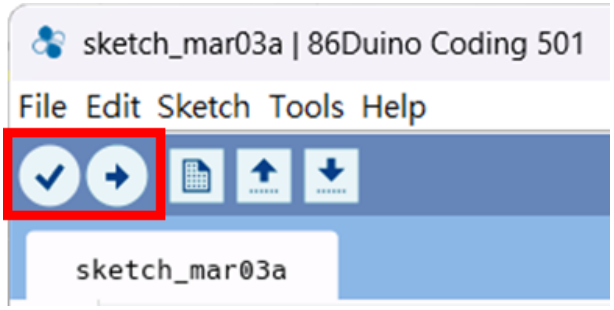
QEC EtherCAT MDevice 可通过 EtherCAT library 在 EtherCAT 网络中进行设置，并在 86Duino IDE 中编写控制动作。86Duino 开发环境主要分为两个部分：`setup()` 与 `loop()`，分别对应初始化与主程序。

```
1  void setup() {
2    // put your setup code here, to run once:
3
4  }
5
6  void loop() {
7    // put your main code here, to run repeatedly
8
9  }
```

操作 EtherCAT 网络前，必须先完成一次配置设置。EtherCAT 设备的流程应由 Init 状态进行到 OP 状态。

若要实现 EtherCAT 通信，用户必须使用 86Duino Coding IDE 501+ 的 EtherCAT Library。详细信息请参见 Ch.5 软件功能。

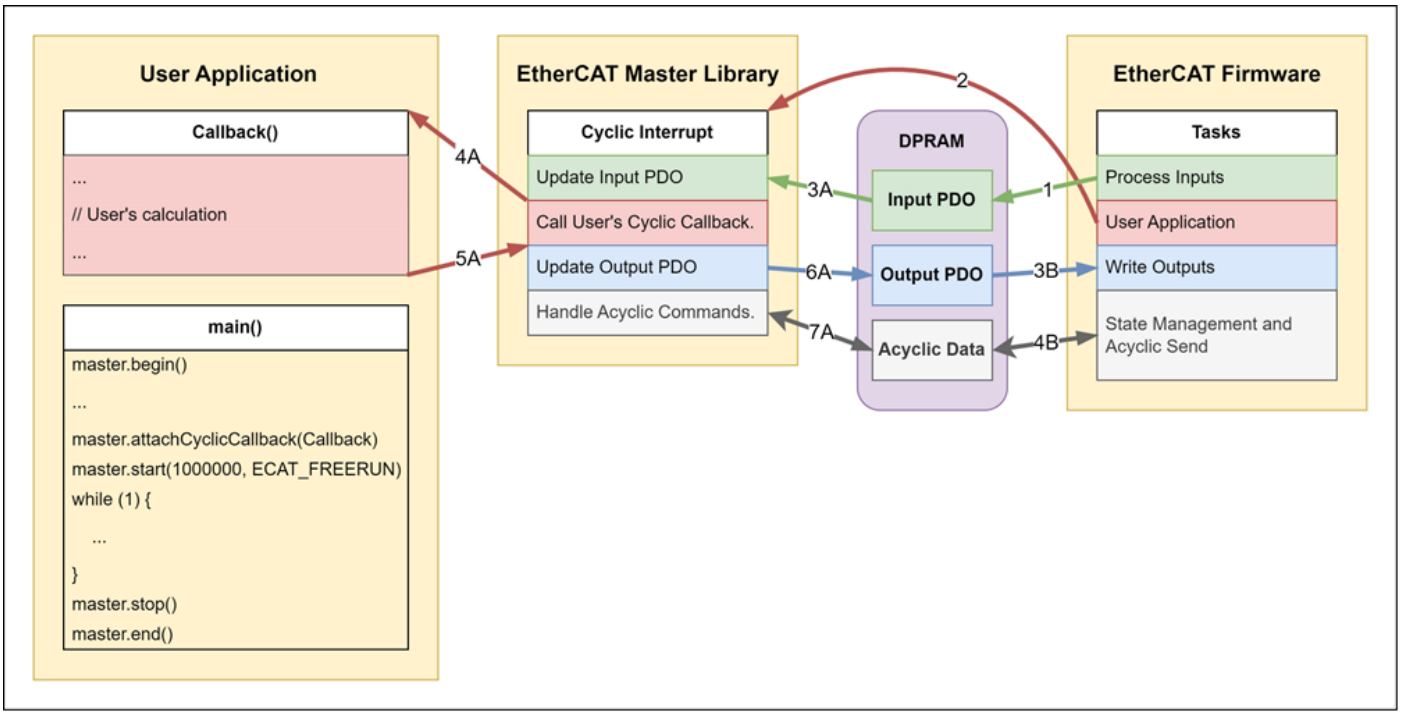
***注意：** 代码编写完成后，请点击工具列的 进行编译；确认编译完成且无错误后，即可点击 进行上传。



4.5.1.1 示例 1：Free Run 模式

在 Free Run 模式下，本地周期由应用控制器的本地计时器中断触发。MDevice 可选择性地修改 cycle time，以改变计时器中断的周期。在 Free Run 模式下，本地周期的运行独立于通信周期与 / 或 MDevice 周期。

QEC MDevice 的 Free Run 模式示意图：



这是未启用双系统同步的 free-run 模式。如图所示，编号箭头代表动作顺序。不过流程在第 3 步会出现分支：EtherCAT firmware 对 EtherCAT MDevice library 触发 cyclic interrupt（第 2 步）后，并不会等待 EtherCAT MDevice library，而是直接继续运行下一个动作。两个系统各自独立运行，彼此不同步。若用户已注册 cyclic callback，cyclic interrupt 会加以调用，如第 4A 步所示。

示例代码如下：

```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

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

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

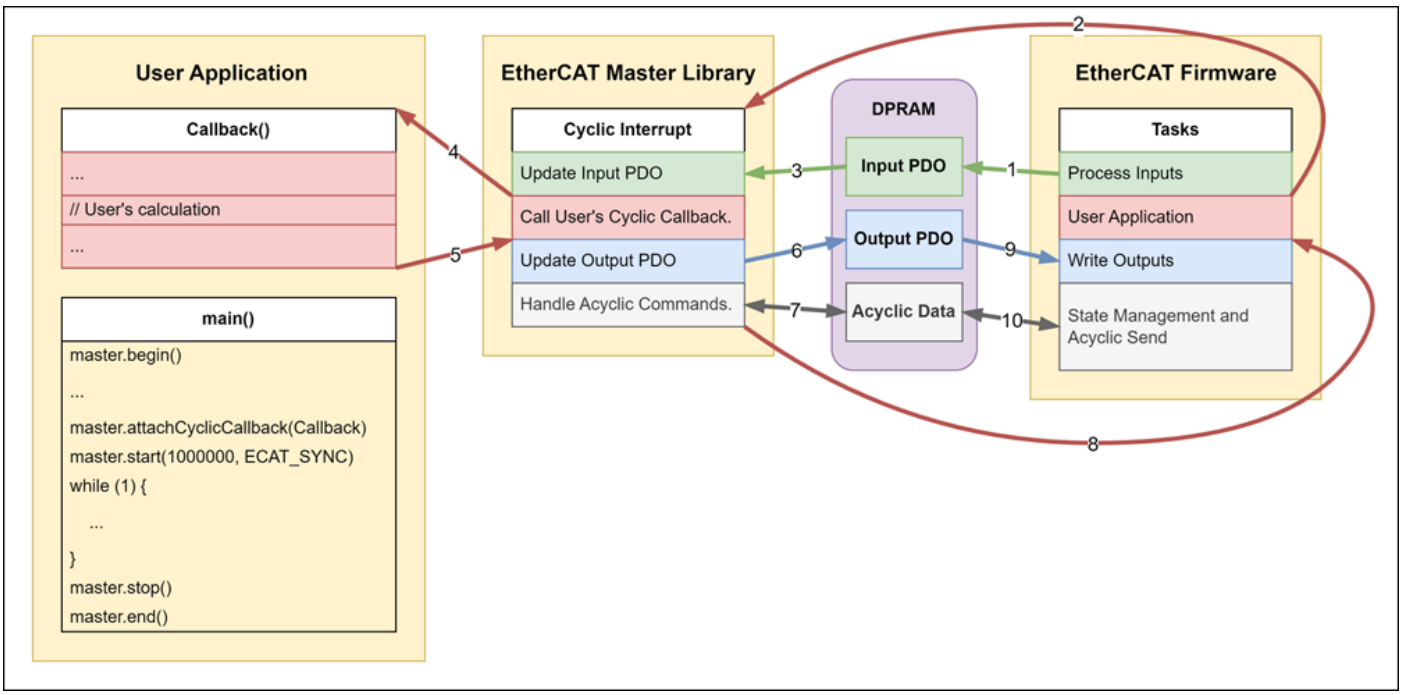
4.5.1.2 示例 2：SYNC 模式

本地周期在收到 SM2 事件（具周期性输出）或 SM3 事件（无周期性输出）时启动。若有输出数据，SubDevice 一般会与 SM2 事件同步；若无输出数据，则与 SM3 事件同步，例如仅有周期性输入的情况。

此模式提供下列选项：

- 与 SM2/3 事件同步
- 与 SM2/3 事件同步，并位移 Input Latch 时间

QEC MDevice 的 SYNC 模式示意图：



此模式提供最高程度的双系统同步。如图所示，编号箭头代表动作顺序，流程中没有分支。EtherCAT firmware 对 EtherCAT MDevice library 触发 cyclic interrupt（第 2 步）后，会等待 EtherCAT MDevice library 回复 ACK（第 8 步）才继续运行下一个动作。若用户已注册 cyclic callback，cyclic interrupt 会加以调用，如第 4 步所示。只要用户在 cyclic callback 中读取当前的输入 process data、完成处理、计算出输出 process data 并写回，当前周期就会将输出 process data 送往 EtherCAT 网络，满足实时控制系统的需求。

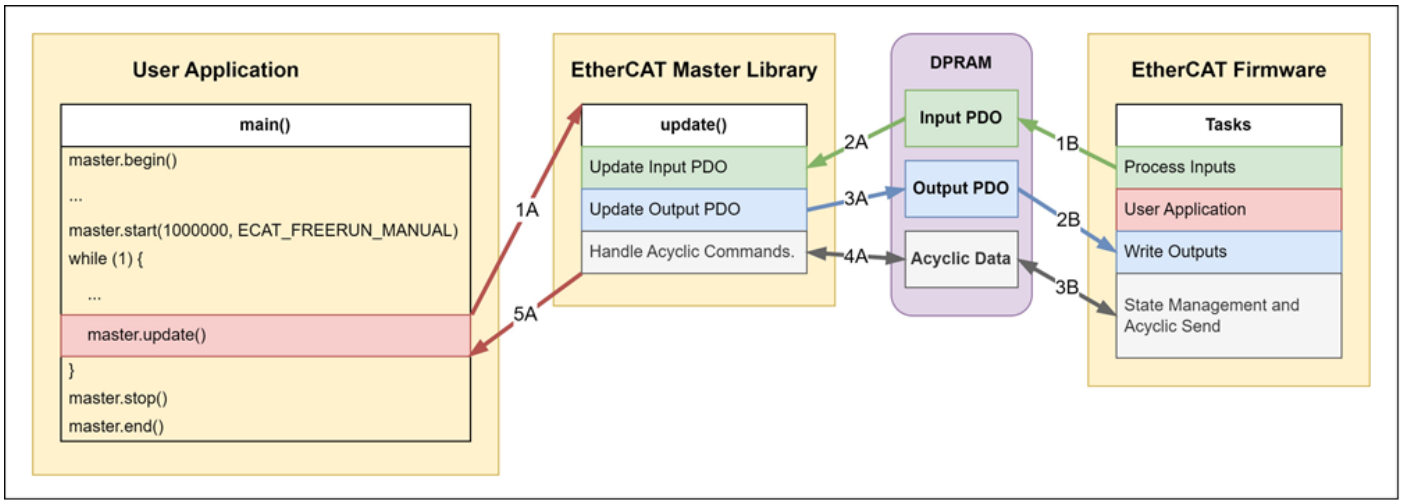
示例代码如下：

```
#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 示例 3：Free Run Manual 模式

这同样是未启用双系统同步的 free-run 模式。与 ECAT_FREERUN 模式的主要差异在于，本模式没有 cyclic interrupt 来更新 process data 与处理非周期性命令，用户必须自行调用 `EthercatMaster::update()` 来更新 process data 并处理非周期性命令。此外，由于本模式没有 cyclic interrupt，cyclic callback 也不会被调用。如图中编号箭头所示，两个系统各自独立运行，彼此不同步。

QEC MDevice 的 Free Run Manual 模式示意图：



示例代码如下：

```
#include "Ethercat.h"

EthercatMaster master;

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

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

4.5.2 SubDevice 信息

QEC MDevice 的 EtherCAT library 提供多个函数，可取得网络上 EtherCAT SubDevice 的相关信息，包括查找网络上的 SubDevice 数量、依序号取得 SubDevice 的 Vendor ID、Product Code 与 Alias Address，以及利用上述信息反查 SubDevice 编号。这些信息可用于判断 SubDevice 的类型，并选用适当的 EtherCAT SubDevice 类别加以挂载。

4.5.2.1 示例 1：使用 EthercatMaster 类别

使用 EthercatMaster 类别显示 SubDevice 信息。

示例代码如下：

```
#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 示例 2：使用 EthercatDevice_Generic 类别

使用 EthercatDevice_Generic 类别显示 SubDevice 信息。

示例代码如下：

```
#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 示例 3：使用 EthercatDevice_CiA402 类别

使用 EthercatDevice_CiA402 类别显示 SubDevice 信息。

示例代码如下：

```
#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);
```

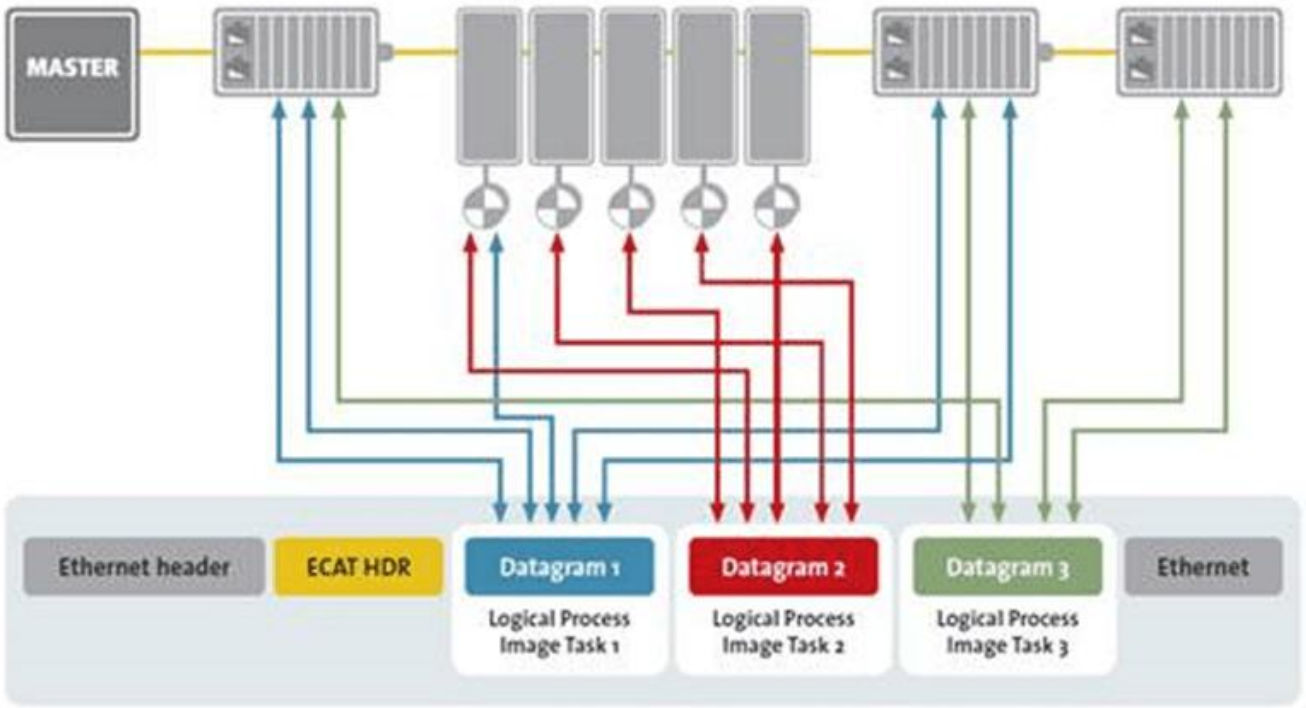
```
    Serial.print("        EoE Details: 0x");
    Serial.println(slave.getEoEDetails(), HEX);

    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) 函数

Process Data 指的是 EtherCAT 网络中 MDevice 与 SubDevice 之间交换的实时通信数据，内容涵盖控制、监控与通信用途的信息。EtherCAT MDevice 会周期性地发送 process data，以控制并监控所有 SubDevice，确保高同步性与低延迟。

EtherCAT SubDevice Controller (ESC) 中的 Fieldbus Memory Management Unit (FMMU) 可将双端口内存映射至逻辑地址。所有 SubDevice 节点会检查 EtherCAT MDevice 送出的 EtherCAT frame，比对 process data 的逻辑地址与 FMMU 中设置的地址；若比对相符，输出的 process data 会被写入双端口内存，而输入的 process data 则会插入 EtherCAT frame 中。

整体而言，process data 是 EtherCAT 技术的核心环节，适用于机器人控制、CNC 控制、自动化控制等领域的实时应用。



EtherCAT：网络封包与数据的交换。(数据源：http://www.ethercat.org/)

4.5.3.1 示例 1：从 Input PDO 读取一个 bit

使用 `pdoBitRead()` 从 Input PDO 读取一个 bit。

一台 16 信道数字输入的 EtherCAT SubDevice 具有 2 byte 的 Input PDO，每个 bit 对应一个数字输入信道。程序会以 1 Hz 的频率印出信道 0 与信道 9 的状态。

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

示例代码如下：

```
#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 示例 2：从 Input PDO 读取 byte 数据

使用 `pdoRead8()` 从 Input PDO 读取数据。

一台 16 信道数字输入的 EtherCAT SubDevice 具有 2 byte 的 Input PDO，每个 bit 对应一个数字输入信道。程序会以 1 Hz 的频率印出信道 0 与信道 9 的状态。

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

示例代码如下：

```
#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 示例 3：将一个 bit 写入 Output PDO

使用 `pdoBitWrite()` 将一个 bit 写入 Output PDO。

一台 16 信道数字输出的 EtherCAT SubDevice 具有 2 byte 的 Output PDO，每个 bit 对应一个数字输出信道。程序会以 1 Hz 的频率切换信道 0 与信道 9 的状态。

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

示例代码如下：

```
#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 示例 4：将 byte 数据写入 Output PDO

使用 `pdoWrite8()` 将数据写入 Output PDO。

一台 16 信道数字输出的 EtherCAT SubDevice 具有 2 byte 的 Output PDO，每个 bit 对应一个数字输出信道。程序会以 1 Hz 的频率切换信道 0 与信道 9 的状态。

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

示例代码如下：

```
#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
}
```

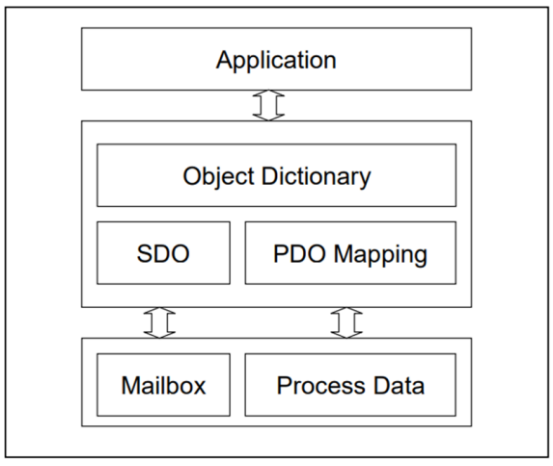
4.5.4 CANopen over EtherCAT (CoE) 功能

CoE（CAN application over EtherCAT）是架构在 EtherCAT 网络之上的 CANopen 协议，可在 EtherCAT 网络上以 CANopen 协议进行通信。Object Dictionary 中存放参数、应用数据，以及 process data 接口与应用数据之间的对应信息（PDO mapping），其项目可通过 Service Data Objects (SDO) 访问。

CANopen 是架构在 Controller Area Network (CAN) bus 之上的高级通信协议，常用于工业应用中控制系统与设备之间的通信。它定义了一组通信对象、数据类型与网络管理功能，以利设备之间的数据交换、配置设置与控制。

CANopen 协议包含下列面向：

- Object Dictionary：定义设备之间交换的所有数据对象与参数。Object Dictionary 涵盖变量、参数、事件、功能等多种类型的对象。
- PDO (Process Data Object)：用于实时数据传输。PDO 让设备之间能以固定周期或事件触发的方式发送数据，实现实时控制与数据交换。
- SDO (Service Data Object)：用于设置与管理设备参数。SDO 提供读取、写入与参数配置等功能，让设备能动态交换配置信息。



SDO 服务主要包含两类命令：SDO command 用于访问 Object Dictionary 中的对象，SDO information command 则用于取得这些对象的详细信息。

4.5.4.1 示例 1：SDO Upload

使用 `sdoUpload8()` 运行 SDO Upload。

`sdoUpload16()`、`sdoUpload32()` 与 `sdoUpload64()` 的用法与 `sdoUpload8()` 相同，差别仅在于回传值的类型。

示例代码如下：

```
#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 示例 2：带 abort code 的 SDO Upload

使用 `sdoUpload()` 运行带 abort code 的 SDO Upload。

对一个不存在的对象发出 SDO Upload 命令以读取数值，预期会取得 0x06020000 的 abort code。关于 abort code 的更多信息，请参见 SDO Abort Code。

示例代码如下：

```
#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 示例 3：SDO Download

使用 `sdoDownload8()` 运行 SDO Download。

`sdoDownload16()`、`sdoDownload32()` 与 `sdoDownload64()` 的用法与 `sdoDownload8()` 相同，差别仅在于输入参数的类型。

示例代码如下：

```
#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 示例 4：带 abort code 的 SDO Download

使用 `sdoDownload()` 运行带 abort code 的 SDO Download。

对一个不存在的对象发出 SDO Download 命令以写入数值，预期会取得 0x06020000 的 abort code。关于 abort code 的更多信息，请参见 SDO Abort Code。

示例代码如下：

```
#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 示例 5：印出 PDO mapping 配置

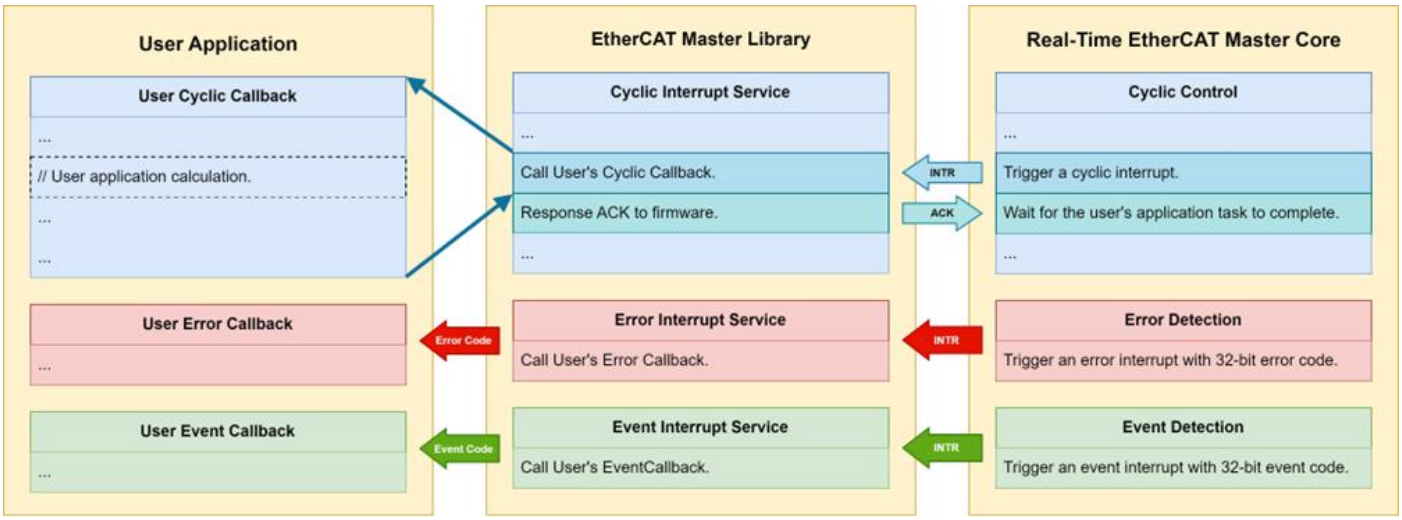
印出 PDO mapping 配置。

示例代码如下：

```
#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 函数



本 library 提供下列三种 callback：

- **Cyclic Callback**
  Cyclic Callback 的用途，是让用户能实现运动控制、CNC 控制、机器人控制等周期性控制系统。Real-Time EtherCAT MDevice Core 会依设置的 cycle time 对 EtherCAT MDevice Library 触发 cyclic interrupt，并等待 ACK 以确保 process data 同步。若用户已注册 Cyclic Callback，系统便会加以调用，以达成周期性控制。

- **Error Callback**
  当 Real-Time EtherCAT MDevice Core 检测到错误时，会触发 error interrupt，并将一组 32-bit 的 error code 传给 EtherCAT MDevice Library。若用户已注册 error callback，系统便会调用该 callback，以告知用户具体的错误内容。

  Error Callback 支持的 error code 如下：

定义	代码	说明
ECAT_ERR_WKC_SINGLE_FAULT	2000001	发生 working counter 错误。
ECAT_ERR_WKC_MULTIPLE_FAULTS	2000002	发生多次 working counter 错误。
ECAT_ERR_SINGLE_LOST_FRAME	2000003	发生 frame 遗失。
ECAT_ERR_MULTIPLE_LOST_FRAMES	2000004	发生多次 frame 遗失。
ECAT_ERR_CABLE_BROKEN	2000007	线路断线。
ECAT_ERR_WAIT_ACK_TIMEOUT	2001000	Firmware 等待 cyclic interrupt ACK 超时。

- **Event Callback**
  当 Real-Time EtherCAT MDevice Core 检测到事件时，会触发 event interrupt，并将一组 32-bit 的 event code 传给 EtherCAT MDevice Library。若用户已注册 event callback，系统便会调用该 callback，以告知用户具体的事件内容。Event Callback 支持的 event code 如下：

定义	代码	说明
ECAT_EVT_STATE_CHANGED	1000001	MDevice 的 EtherCAT 状态已变更。
ECAT_EVT_CABLE_RECONNECTED	1000002	线路已重新连接。

4.5.5.1 示例 1：Cyclic callback

一台 16 信道数字输出的 EtherCAT SubDevice 具有 2 byte 的 Output PDO，每个 bit 对应一个数字输出信道。程序会以 1 Hz 的频率切换信道 0 与信道 9 的状态。

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

示例代码如下：

```
#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 示例 2：Error callback

每秒印出各类错误的发生次数。

示例代码如下：

```
#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 示例 3：Event callback

每秒印出各类事件的发生次数。

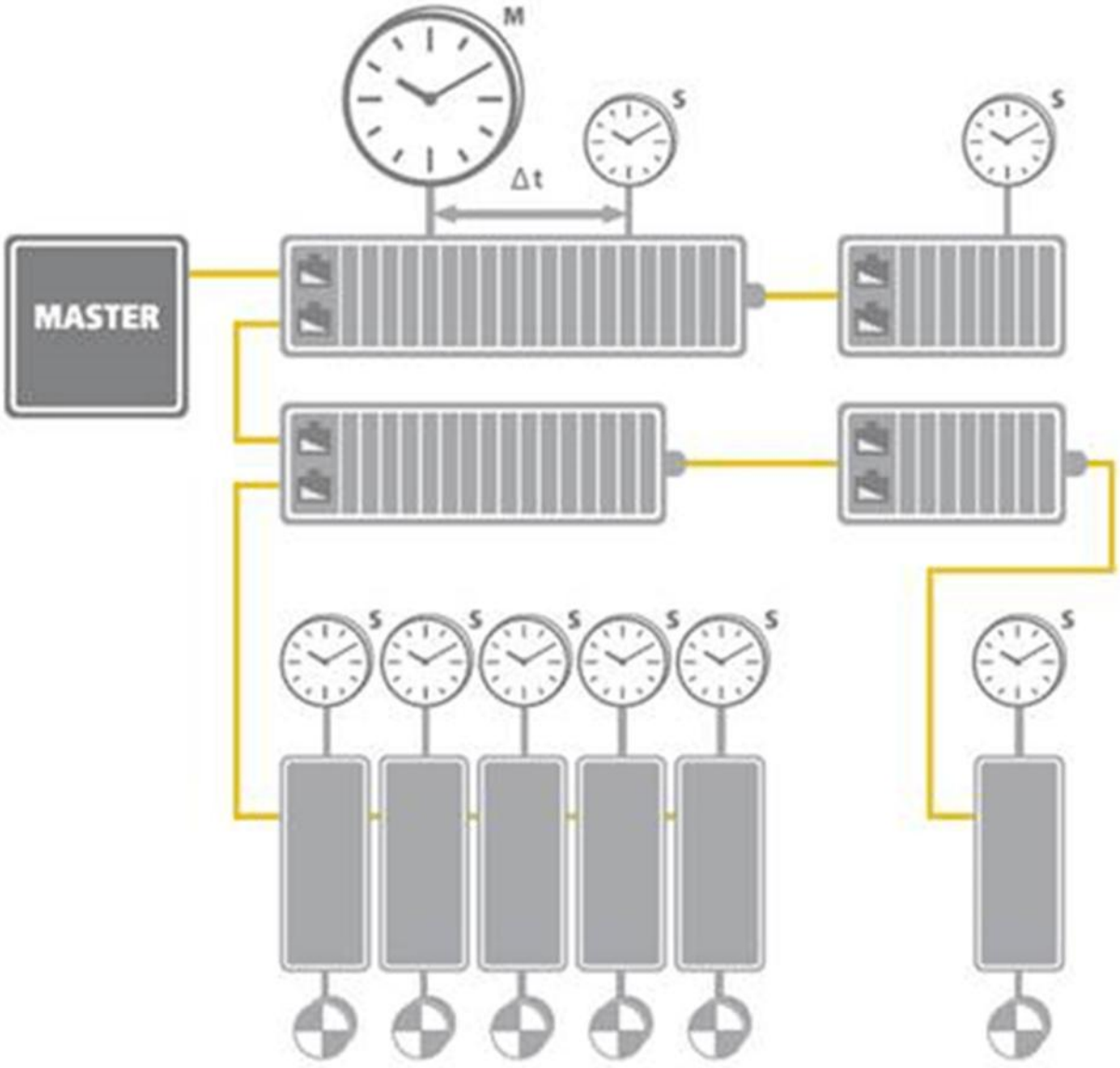
示例代码如下：

```
#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) 配置函数

在空间上分散、却必须同时动作的制程应用中，精确同步格外重要。例如多轴伺服需运行协调动作的应用即属此类。

与完全同步式通信（其质量会立即受通信错误影响）相比，分布式同步时脉对通信系统的 jitter 具有相当高的容忍度。因此 EtherCAT 用于节点同步的解决方案即是以这类 Distributed Clock (DC) 为基础。

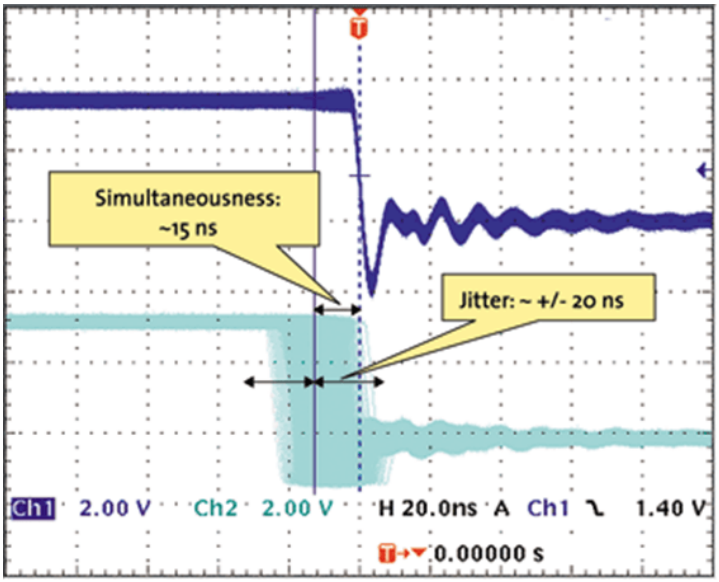


EtherCAT：Distributed Clock (DC) 示意图。(数据源：http://www.ethercat.org/)

EtherCAT 的同步机制以 IEEE-1588 Precision Clock Synchronization Protocol 为基础，并延伸定义出所谓的 Distributed Clock (DC)。简单来说，每一颗 EtherCAT ESC 都维护一组硬件时脉，最小时间间隔为 1 奈秒（共 64 bit）。EtherCAT SubDevice 所维护的时间称为 Local system time。

通过精准的网络时间同步机制与动态时间补偿机制（*1），EtherCAT DC 技术可确保各 EtherCAT SubDevice 的 local system time 之间的时间差维持在 +/- 20 奈秒以内。下图为两台 SubDevice 输出数字信号的示波器画面，可看出两台 EtherCAT SubDevice 的 I/O 信号时间差约为 20 奈秒。

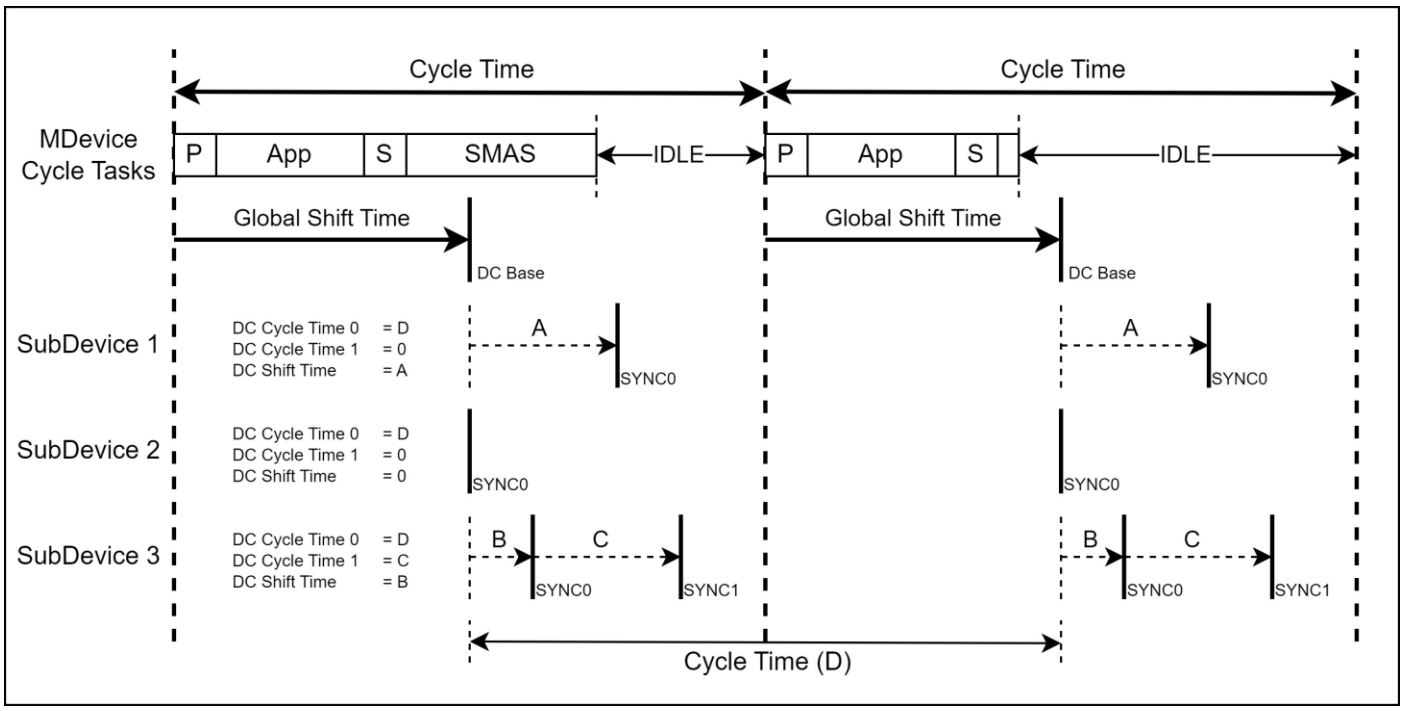
（*1）请参见 EtherCAT 标准文档 ETG1000.4



同步性与同时性：两台分散设备的示波器画面，两者之间相隔 300 个节点与 120 公尺的线材。（数据源：http://www.ethercat.org/）

设置 EtherCAT SubDevice 的 DC 参数。此函数可设置三个 DC 参数：

- ***DC Cycle Time 0*** 用于设置 SYNC0 信号的周期时间，通常与 EtherCAT 通信的 cycle time 一致。
- ***DC Cycle Time 1*** 用于设置 SYNC1 信号的周期时间，也就是 SYNC0 脉波之后所定义的延迟。此参数为选用。
- ***DC Shift Time*** 用于设置 SYNC0 信号相对于 DC Base 的偏移量。



4.5.6.1 示例 1：启用 DC 同步

使用 EthercatDevice_Generic 类别对 CiA 402 的 EtherCAT SubDevice 实现位置控制。CiA 402 的控制模式设置为 cyclic synchronous position mode，并启用 DC 同步以取得精准的时序。

默认的 PDO mapping 如下：

- 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			

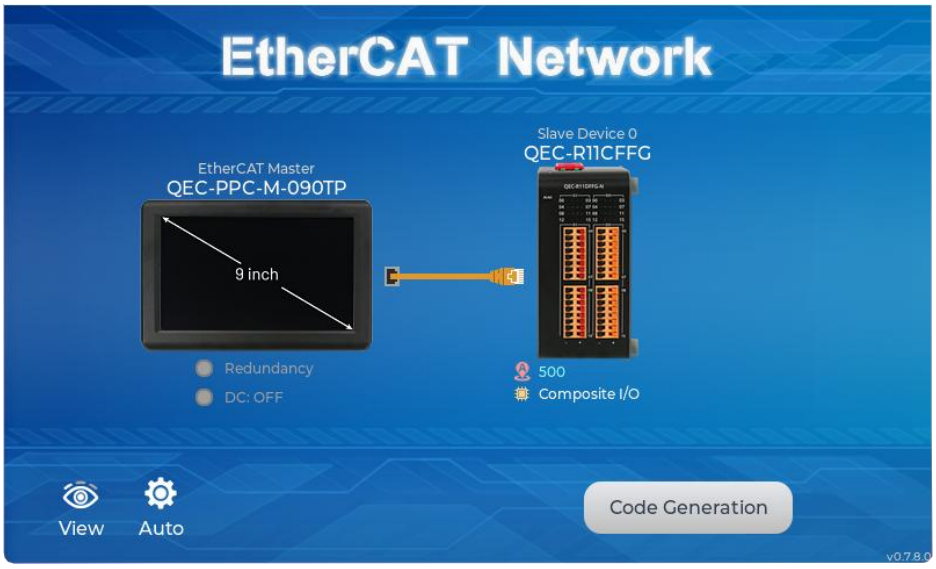
示例代码如下：

```
#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：EtherCAT 配置工具

86EVA 是一套图形化的 EtherCAT 配置工具，以 86Duino IDE 中的 EtherCAT Library 为基础，也是 86Duino 的开发套件之一。用户可用它快速完成 EtherCAT 网络配置并开始编写程序。



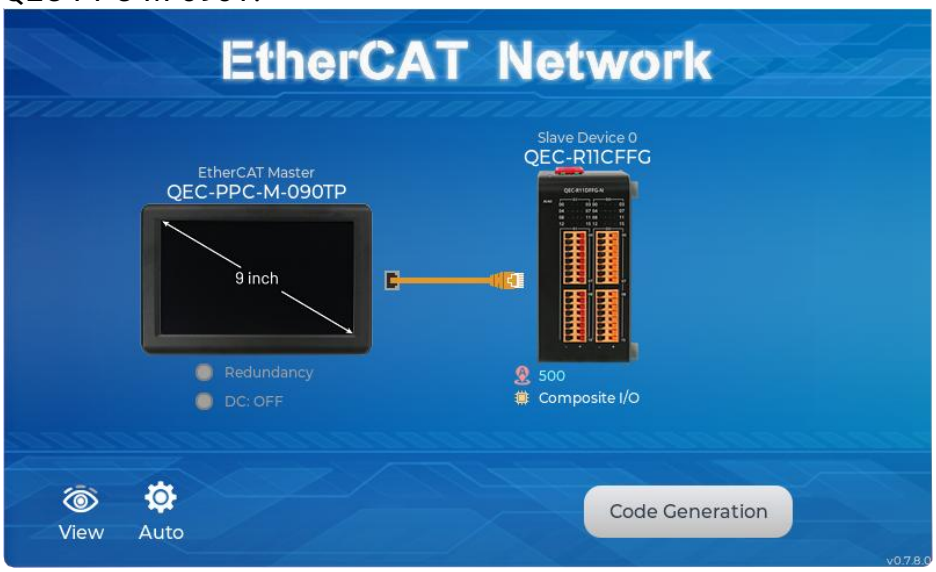
以下关于 86EVA 的说明将聚焦于 QEC EtherCAT MDevice 与 SubDevice，功能包括：

1. 自动产生 Arduino 语言代码（通过 EtherCAT-Based Virtual Arduino）
2. 自动扫描网络上的设备。
3. EtherCAT MDevice 设置：
    - 设置 MDevice 对象名称
    - 设置 Cycle Time
    - 设置 Redundancy 选项
    - 选用 ENI 文件
4. EtherCAT SubDevice 设置：
    - 设置 SubDevice 对象名称
    - 设置 SubDevice Alias Address
    - 可设置 SubDevice 的 I/O Mapping
    - 显示次要设备信息
    - 查看内部信息

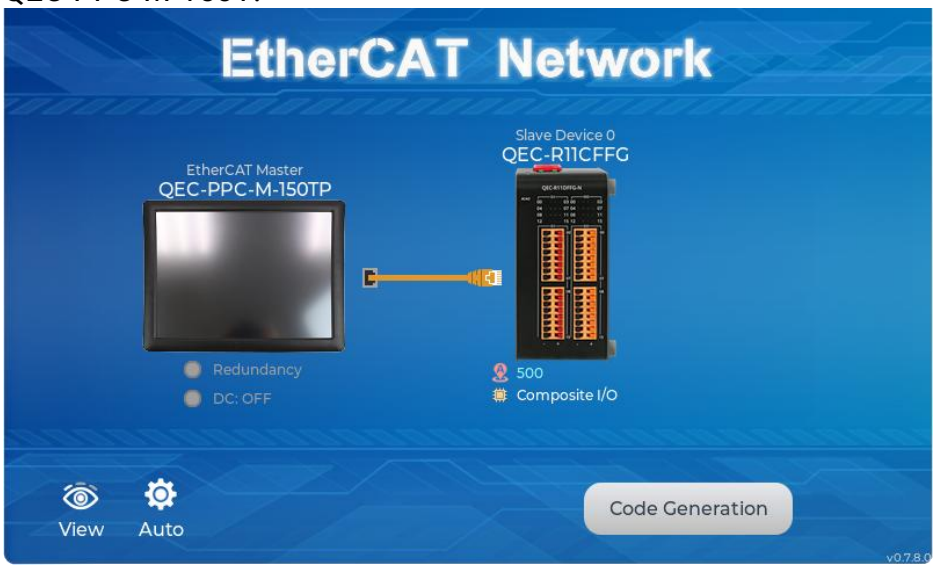
86Duino IDE 开发环境专为 QEC MDevice 设计，涵盖 **QEC-M-01**、**QEC-M-02**、**QEC-M-070T**、**QEC-PPC-M-090T**、**QEC-M-090T**、**QEC-PPC-M-104T**、**QEC-PPC-M-150T** 与 **QEC-M-150T**。

在 86EVA 中完成连接与扫描后，用户即可看到 EtherCAT MDevice 的信息与产品图标。

- QEC-PPC-M-090T:

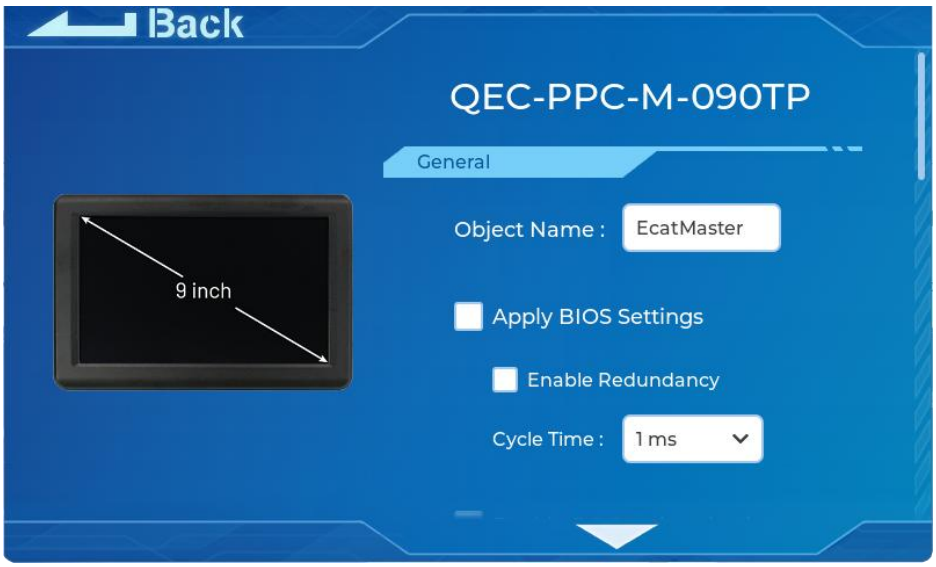


- QEC-PPC-M-150T:



本节以 **QEC-PPC-M-090T** 为例说明。

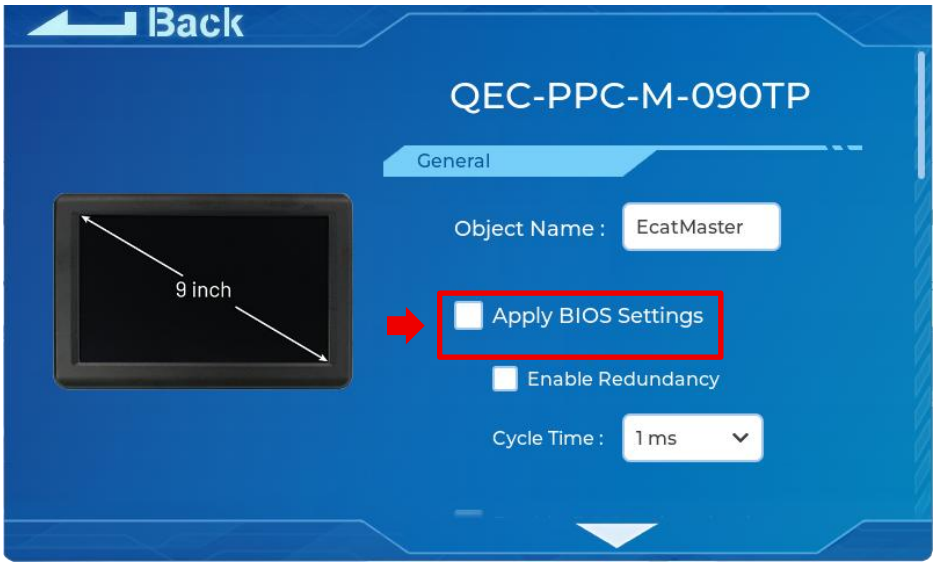
在 QEC-PPC-M-090T 的图标上按两下，即可查看 MDevice 的参数设置。



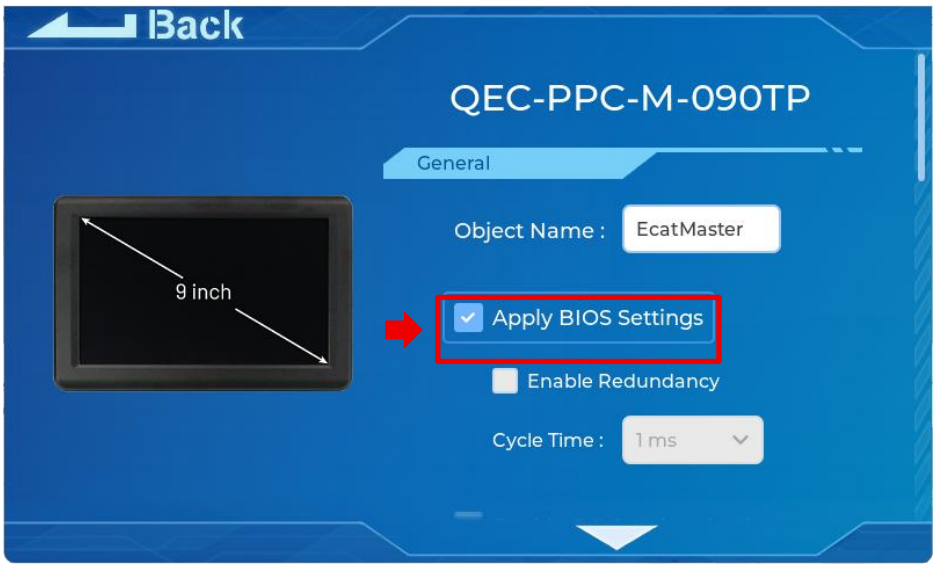
您可以设置 EtherCAT MDevice 对象名称、套用 BIOS 设置、启用 EtherCAT Cable Redundancy 选项、设置 EtherCAT Cycle Time 与 DC 同步功能，并选择 ENI 文件。

关于 86EVA 工具的详细信息，请参见 86EVA 使用手册。

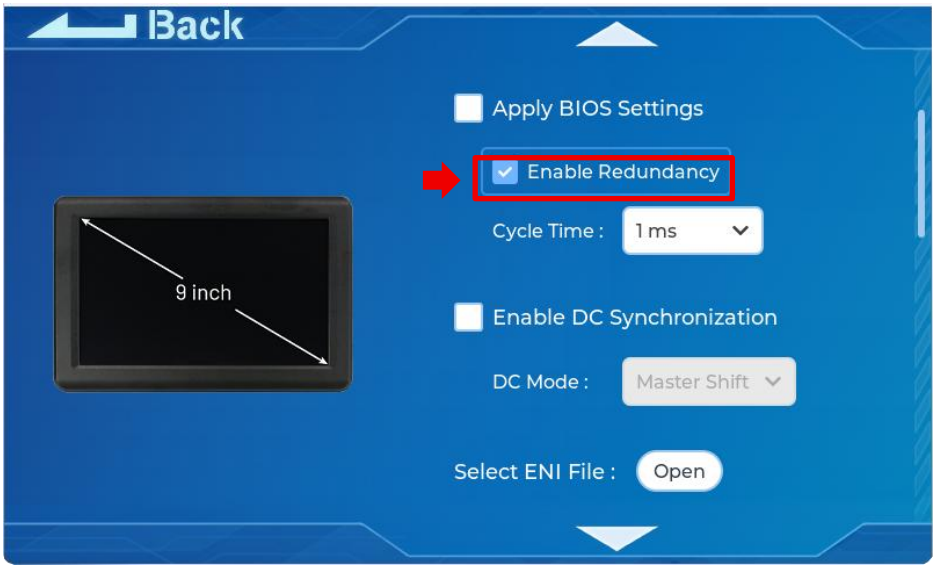
在“**Apply BIOS Settings**”中，您可依照 4.6.3 EtherCAT 页面的说明，以默认值设置 EtherCAT 相关项目。



若用户勾选“**Apply BIOS Settings**”，则复选框“**Enable Redundancy**”与下拉菜单“**Cycle Time**”将无法使用。



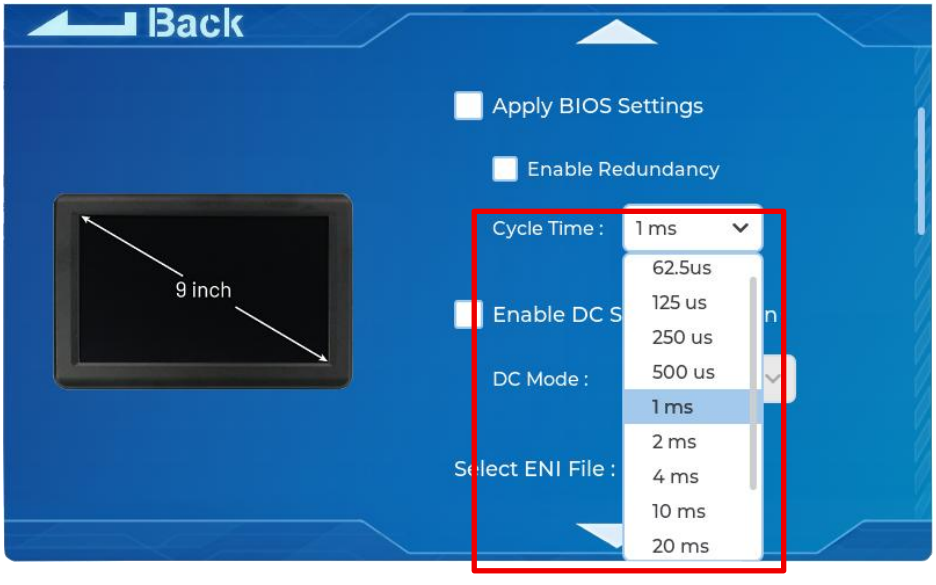
若不使用 BIOS 设置，用户可自行勾选“**Enable Redundancy**”功能。



此功能会在 MDevice 与 SubDevice 之间创建环状拓扑，确保在线材或设备故障时 EtherCAT 通信仍能持续运行。

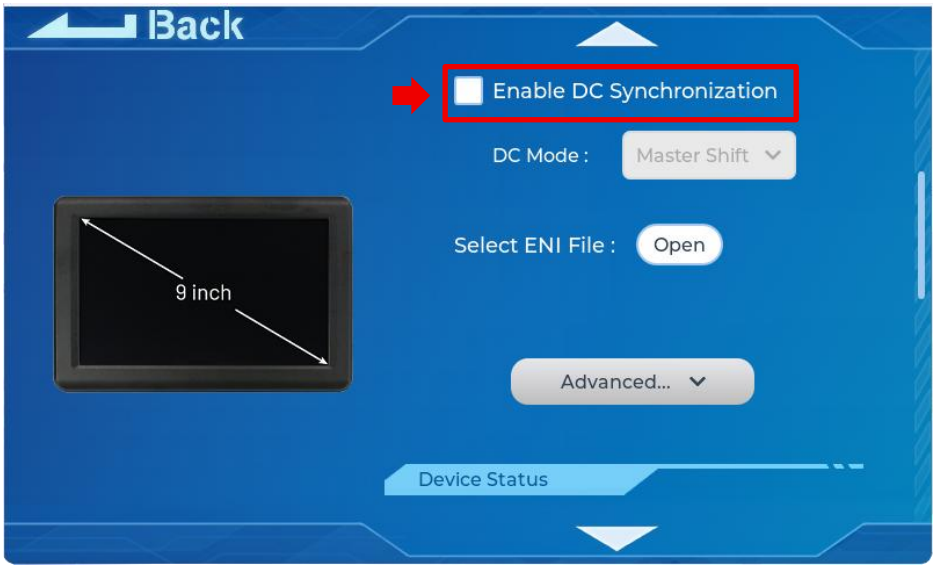
***注意：** 若启用 Redundancy 选项，必须实际接上第二条 EtherCAT 网线以形成环状回路。未接上冗余线材将导致 EtherCAT State Machine 无法切换至 OPERATIONAL 状态，与 SubDevice 的通信也会失败。

若不使用 BIOS 设置，用户可由下拉菜单 **“Cycle time”**选择 EtherCAT 的 cycle time，可选范围为 62.5 µs 至 20 ms。

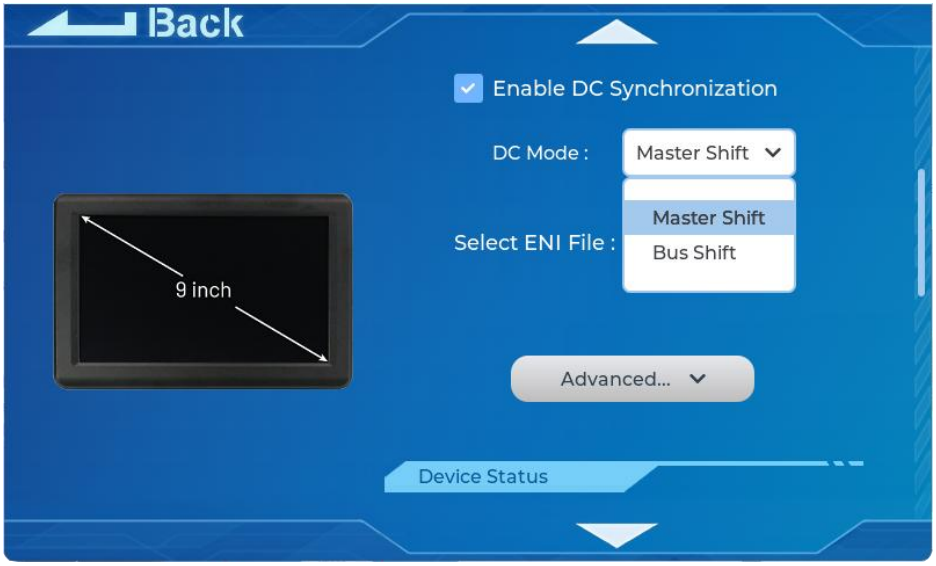


***注意：** cycle time 需配合 SubDevice 的反应时间设置。

在 **“Enable DC Synchronization”**中，您可为目前 EtherCAT 网络上所有支持且能通过 86EVA 验证的 EtherCAT SubDevice 启用 DC 功能。

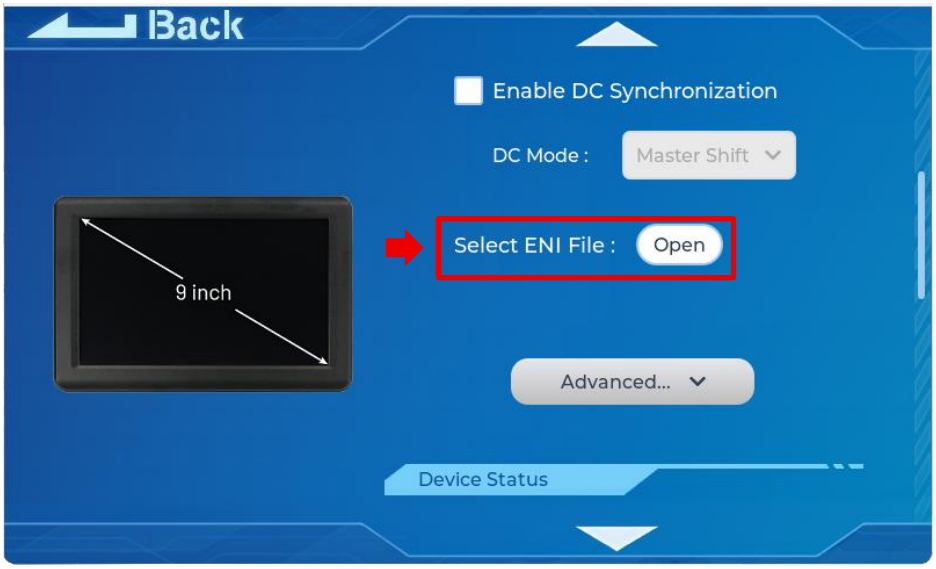


用户亦可由下拉菜单 **“DC Mode”**选择 DC 模式，选项为 “**Bus Shift**”与“**Master Shift**”。



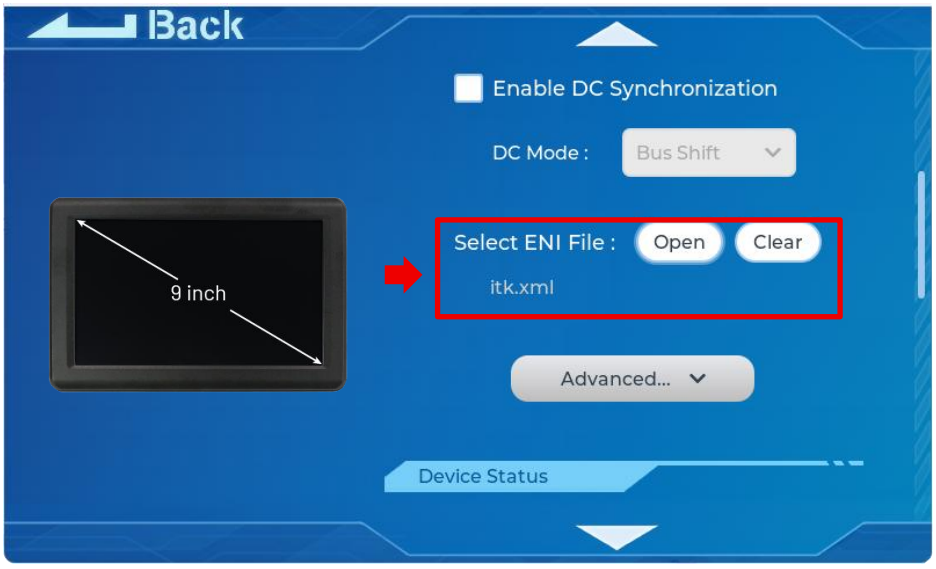
***注意：** DC 功能与模式需依 SubDevice 的规格设置。

在 **“Select ENI File”**中，用户可上传 ENI 文件。关于 ENI 文件的更多信息，可参见 Beckhoff Information System - English。



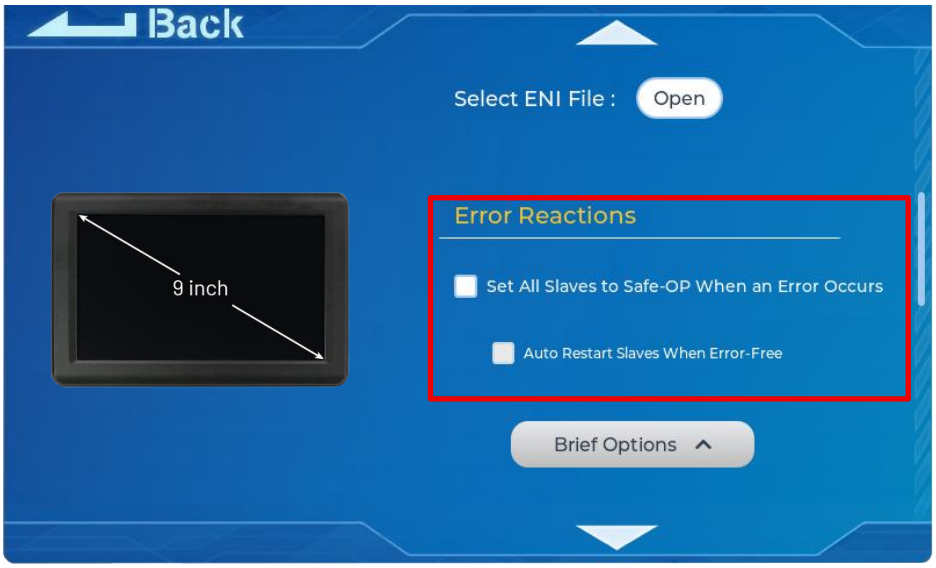
关于如何将 ENI 导入 86EVA，请参见 Ch. 4.5.8 将 ENI 导入 QEC MDevice。

上传 ENI 文件后，**“Select ENI file”**标签下方即会显示文件名称。



点击**“Clear”**按钮即可删除已导入的 ENI 文件。

在 **“Advanced”**区段中，您可设置 EtherCAT 网络的 Error Reactions。这些设置有助于在发生通信错误时提升系统稳定性与故障回复能力。



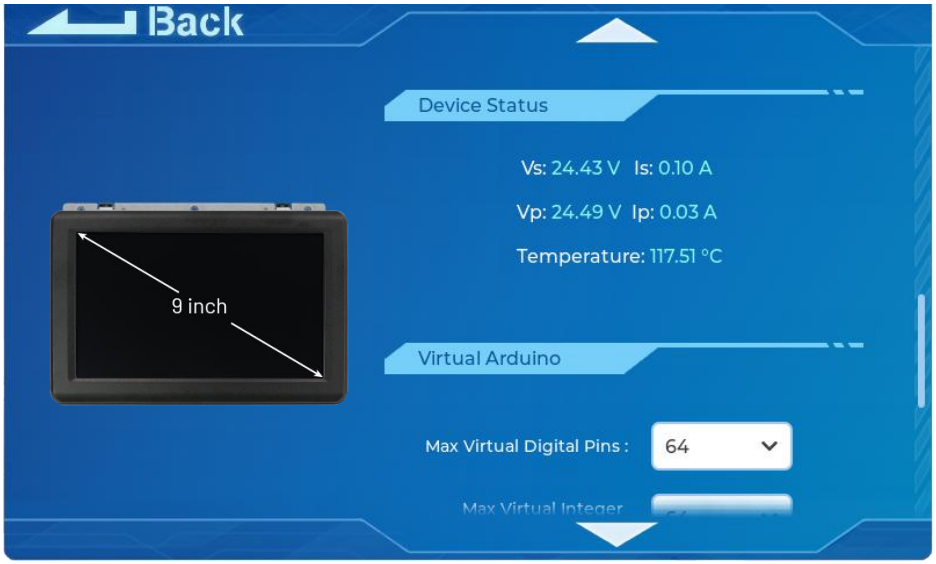
Error Reactions 下有两个选项：

1. **Set All Slaves to Safe-OP When an Error Occurs**
   ：启用后，只要任一 SubDevice 回报错误或 abort code，此选项就会将所有 EtherCAT SubDevice 强制切换至 Safe-Operational (Safe-OP) 状态，让运行期发生故障时有一致且可预期的退回行为。
2. **Auto Restart Slaves When Error-Free**
   ：此选项仅在前一项设置启用时才能勾选。勾选后，一旦错误状况排除，EtherCAT MDevice 会自动尝试恢复通信，并将所有 SubDevice 还原至先前的状态。

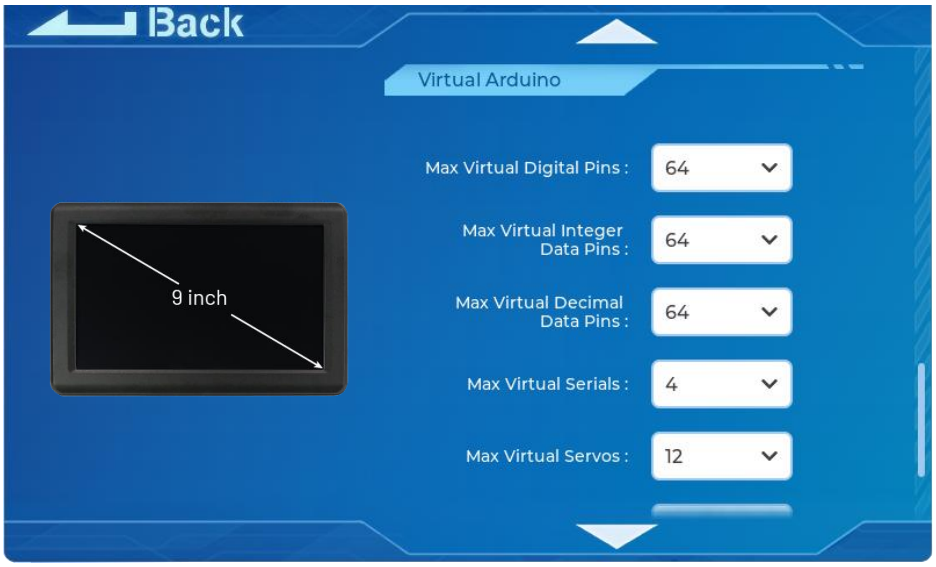
您也可以点击页面下方的**“Brief Options”**按钮，收合 Error Reactions 区段以简化画面。



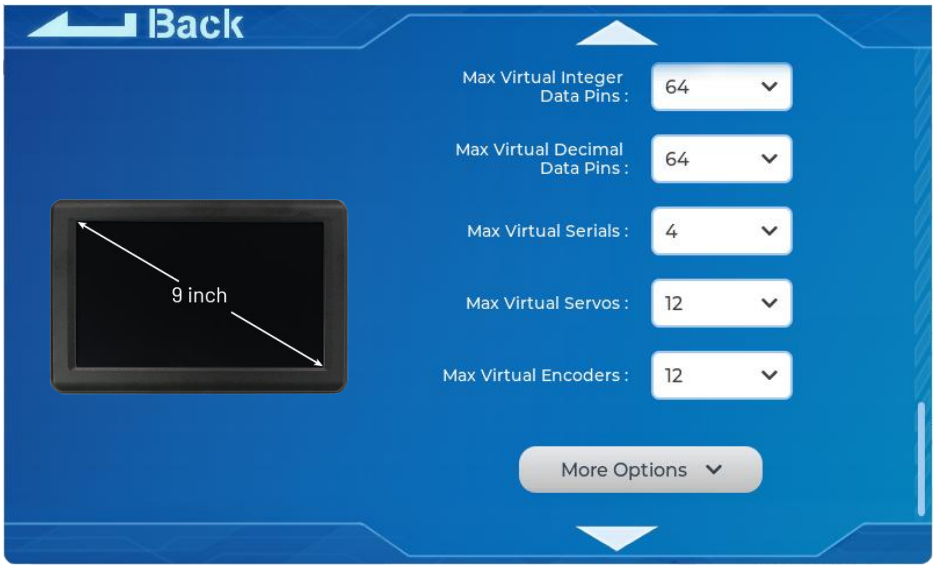
您也可以在 **“Device Status”**区域查看电压、电流与系统温度。



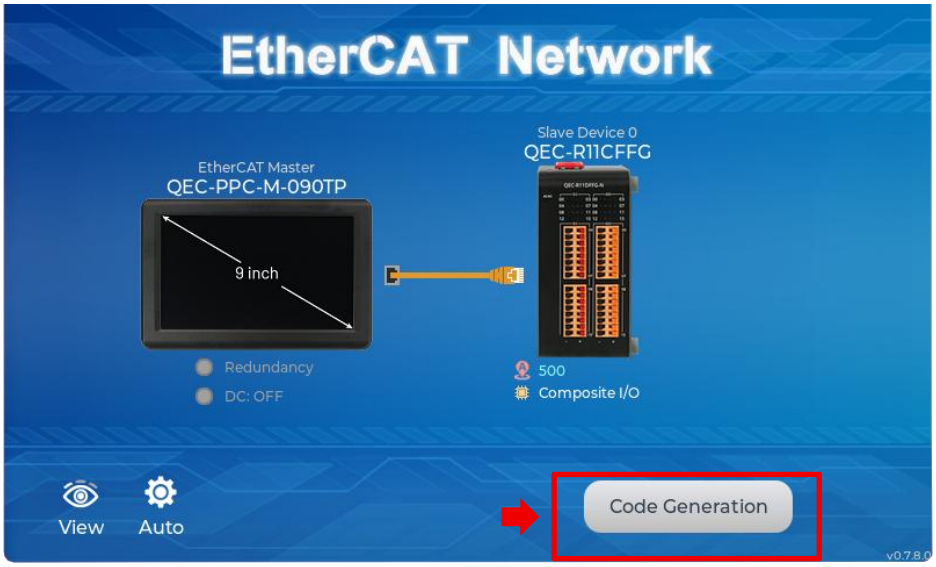
用户亦可设置 **Virtual Arduino** 功能的最大数量。



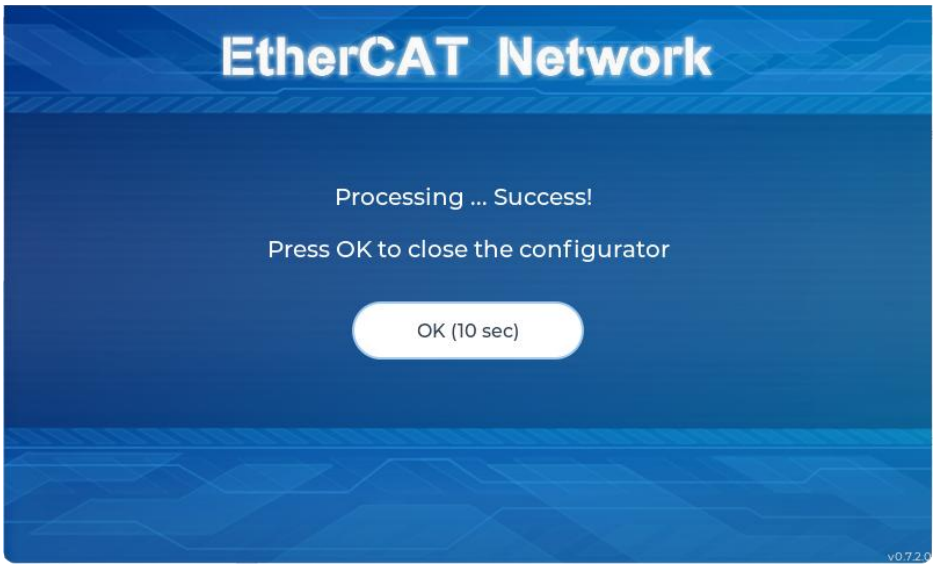
点击 “**More Options**”按钮可显示更多 Virtual Arduino 对象设置。



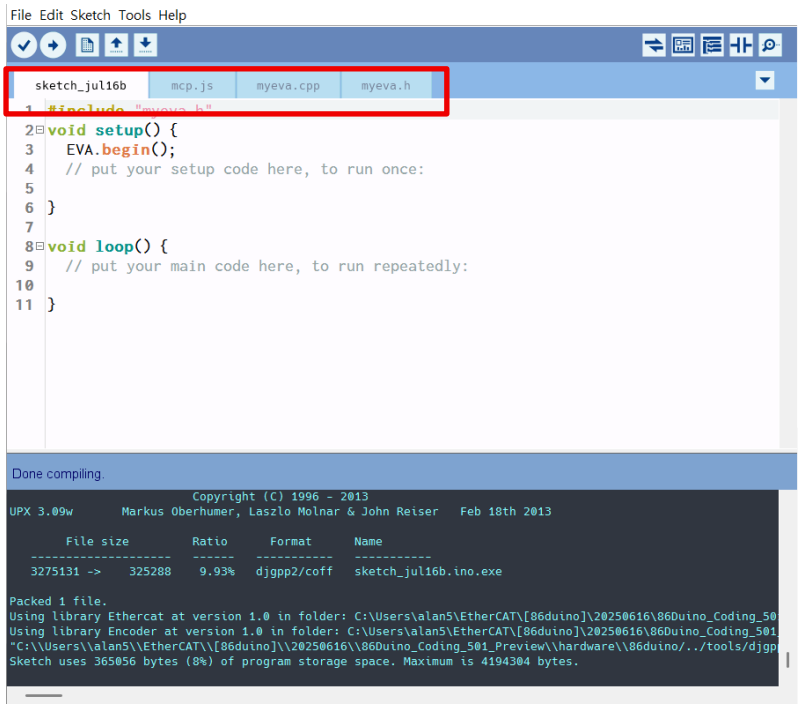
完成 EtherCAT 配置设置后，回到主画面并按下右下角的 **“Code Generation”**按钮。



完成后，双击**“OK”**按钮即可关闭 86EVA，否则窗口将于 10 秒后自动关闭。



产生的代码与文件如下：

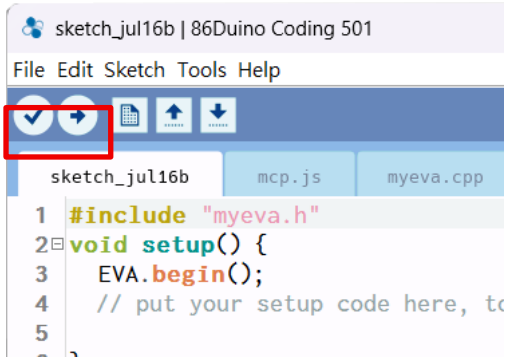


a. sketch_jul23d：主项目（.ino，文件名依您的项目名称而定）。

b. myeva.cpp：86EVA 的 C++ 代码。

c. myeva.h：86EVA 的标头档。

代码完成后，请点击工具列的 进行编译；确认编译完成且无错误后，即可点击 进行上传。上传完成后程序便会开始运行。



更详细的信息请参见 86EVA, EtherCAT-Based Virtual Arduino。

4.5.7.1 故障排除：无法顺利上传代码

当您无法顺利上传代码时，请打开 86EVA 检查 QEC EtherCAT MDevice 的环境是否异常。如下图所示，请尝试更新 QEC EtherCAT MDevice 的环境，更新内容包含下列三个项目：Bootloader、EtherCAT firmware 与 EtherCAT tool。



以下进一步说明更新的操作方式：

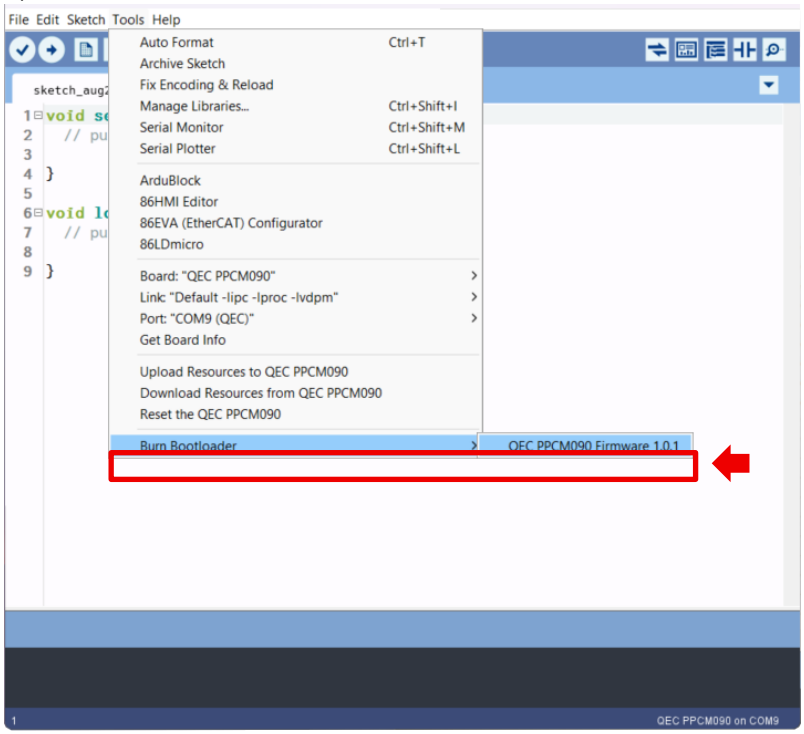
步骤 1：设置 QEC-M

1. 下载并安装 86Duino IDE 501+（或更新版本）：可由 Software 页面下载。
2. 连接 QEC MDevice：使用 USB 连接线将 QEC MDevice 连接至电脑。
3. 打开 86Duino IDE：安装完成后，打开 86Duino IDE 软件。
4. 选择开发板：在 IDE 菜单中选择“**Tools**”→“**Board**”→“**QEC-PPC-M-090T**”（或您所使用的 QEC MDevice 型号）。
5. 选择连接端口：在 IDE 菜单中选择“**Tools**”→“**Port**”，并选择 QEC MDevice 所连接的 USB 端口。

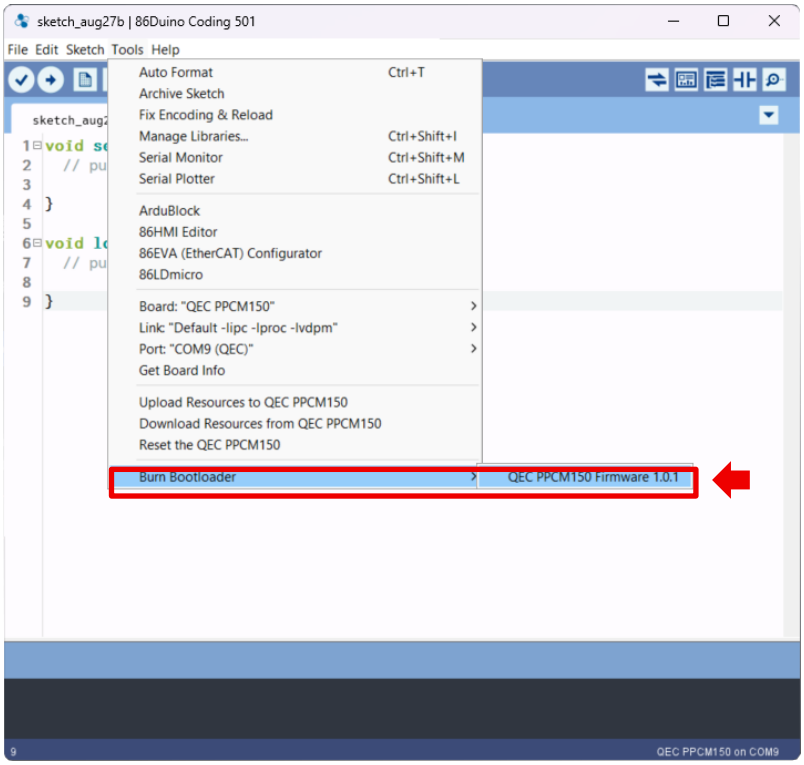
步骤 2：点击“Burn Bootloader”按钮

连接至您的 QEC MDevice 后，选择“**Tools**”→“**Burn Bootloader**”，画面会显示目前选择的 QEC MDevice 名称。点击后即开始更新进程，约需 5 至 20 分钟。

- QEC-PPC-M-090T:



- QEC-PPC-M-150T



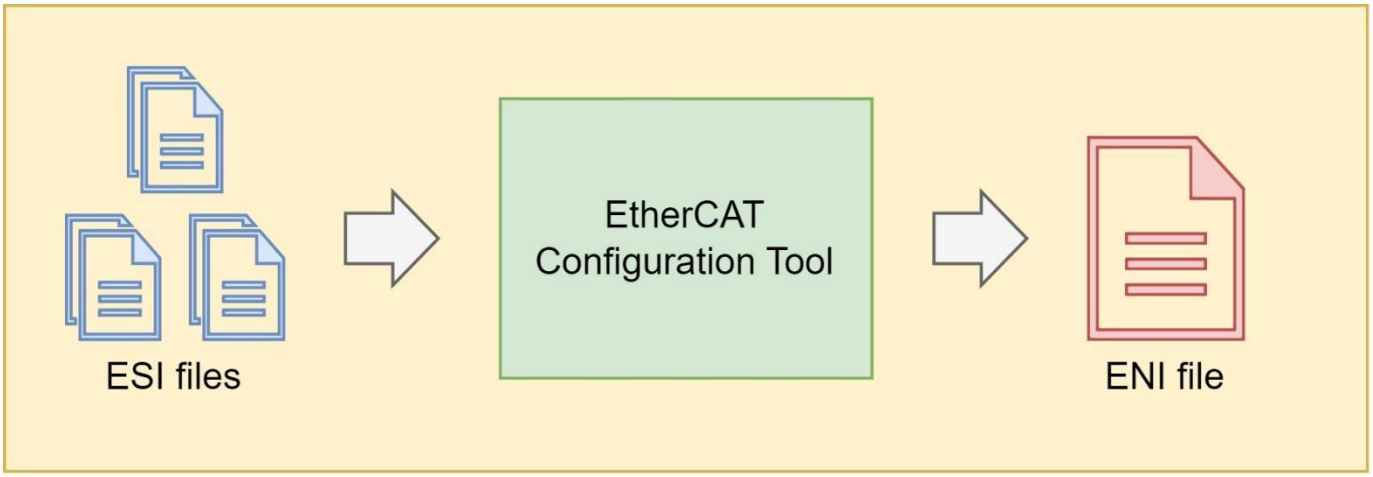
步骤 3：完成更新



完成上述步骤后，您的 QEC-M 设备便已成功更新至最新版本的开发环境。

4.5.8 将 ENI 导入 QEC MDevice

EtherCAT Network Information (ENI) 文件包含设置 EtherCAT 网络所需的必要信息。这份以 XML 为基础的文件，记录了 MDevice 的一般信息，以及所有连接之 SubDevice 的配置。使用 EtherCAT 配置工具，您可以读取 ESI 文件或对网络进行在线扫描以检测所有已连接的 SubDevice，接着设置 PDO mapping、启用 DC 等相关 EtherCAT 设置，并导出 ENI 文件。



EtherCAT Technology Group 要求 EtherCAT MDevice 软件在 Network Configuration 项目中，至少须支持下列其中一种方式：在线扫描（Online Scanning）或读取 ENI（Reading ENI）。
QEC EtherCAT 在 86Duino IDE 501+ 中的 EtherCAT Library 两种方式皆支持。不过在读取 ENI 时，目前 library 仅会由 ENI 文件中截取部分信息用于网络配置。详细说明请参见 A.1 关于 86Duino IDE 的 ENI 配置。

4.5.8.1 方法 1：使用代码

完成 86Duino IDE 环境设置后，请将 ENI 文件放入 USB U 盘，并插入您的 QEC EtherCAT MDevice。

*注意：通过 QEC MDevice 读取 USB U 盘时，路径为 P:\\。）

步骤 1：编写导入代码：

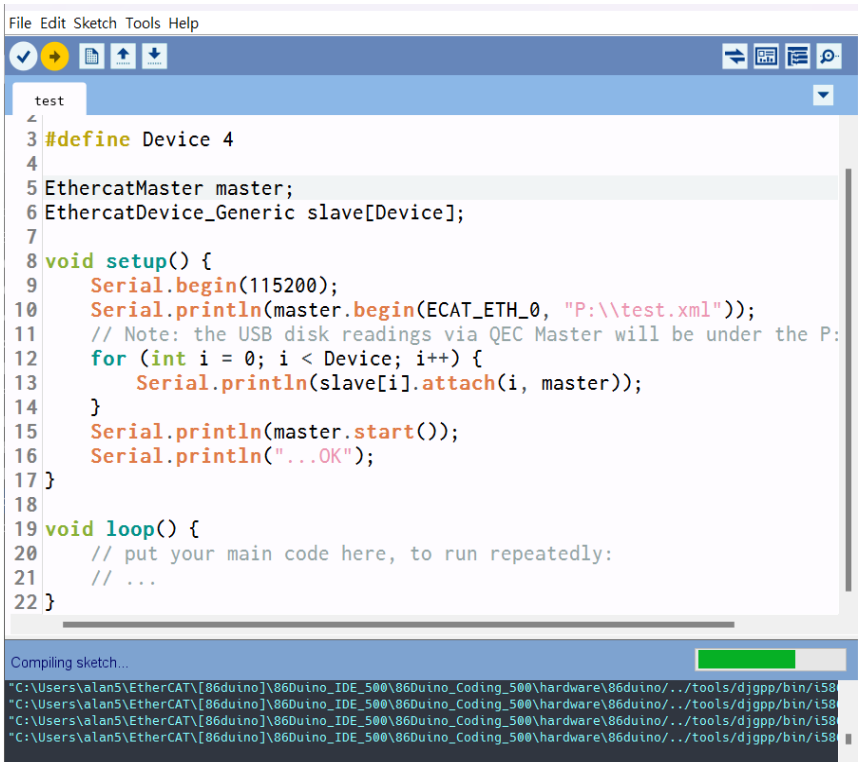
在您的项目代码中，使用 `EthercatMaster::begin()` 函数导入 ENI 文件。

示例代码如下：

```
#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:
    // ...
}
```

步骤 2：上传代码：

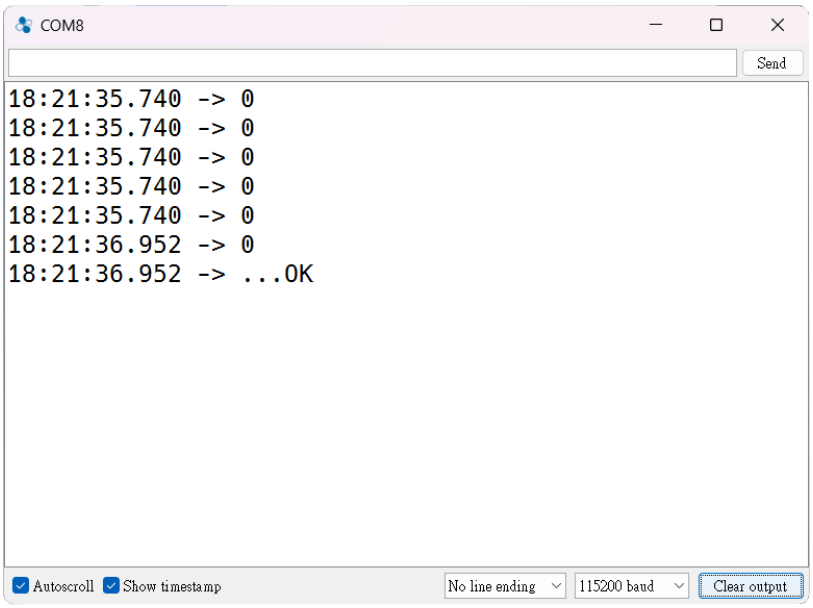
将代码上传至您的 QEC EtherCAT MDevice。



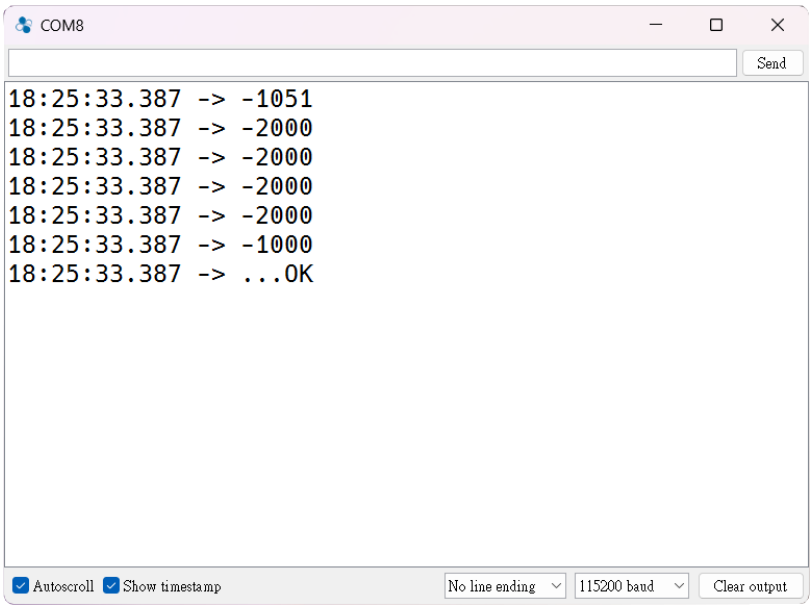
步骤 3：验证 ENI 文件是否导入成功：

通过 Serial Monitor 确认 ENI 文件已正确导入，且网络运行正常。

若 ENI 文件导入成功，Serial Monitor 会显示下列回传值：



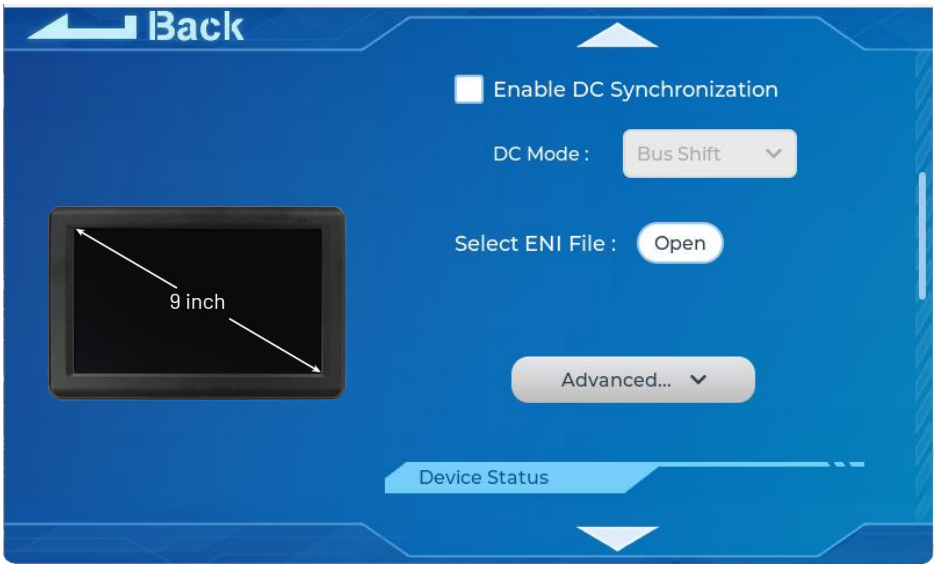
若出现错误，例如 -1051（代表 `ECAT_ERR_MASTER_ENI_MISMATCH`），其他错误码与其代表的意义请参见 EtherCAT API 使用手册。



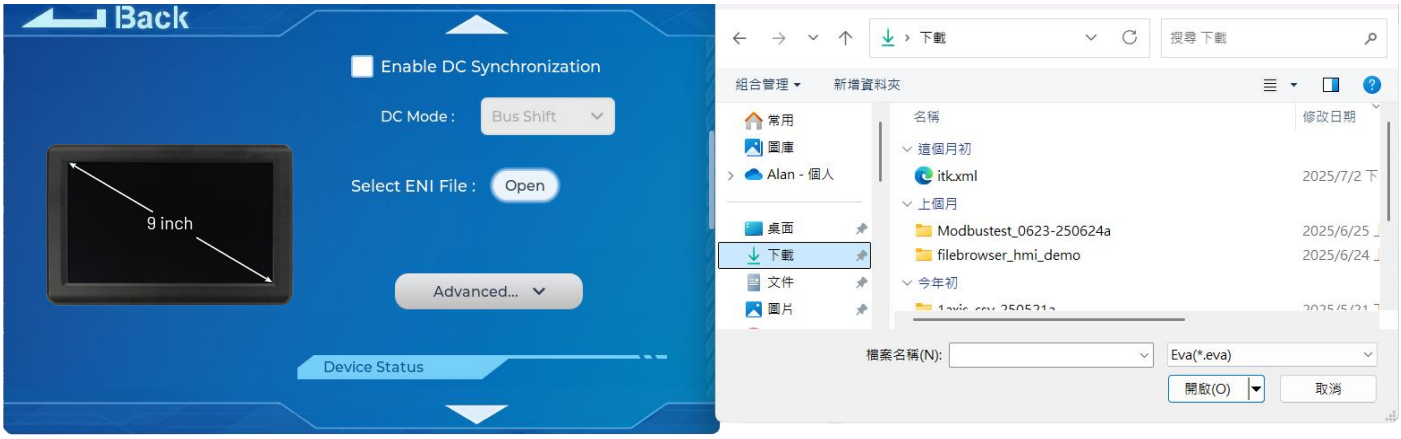
4.5.8.2 方法 2：使用 86EVA

86EVA 的设置方式请参见 Ch 4.5.7 86EVA。

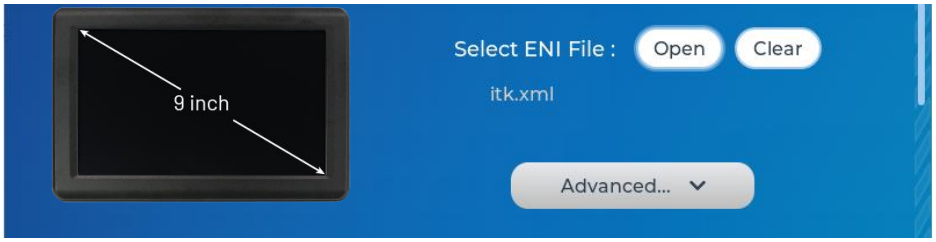
打开 86EVA 并成功扫描 EtherCAT 网络后，请进入 QEC MDevice 的设置页面，并往下卷动至“**Select ENI file**”区段。



点击“**Select ENI file**”旁的“**Open**”按钮，画面会出现打开文件窗口。浏览至您保存 ENI 文件的位置并打开。



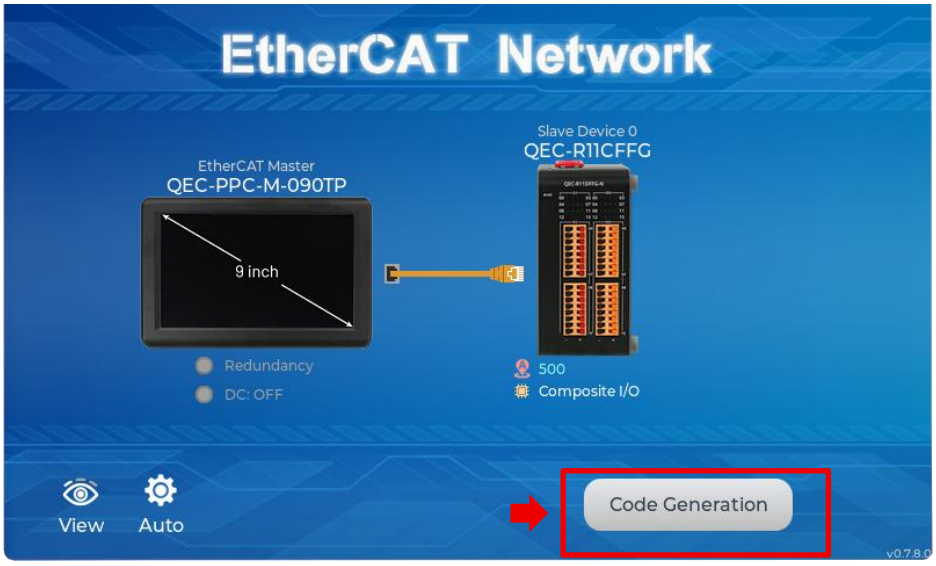
将 ENI 文件导入 86EVA 后，即可看到 ENI 文件名称。



请点击左上角的“**Back**”按钮返回。



回到主画面并按下右下角的“Code Generation”按钮。



完成后，双击 OK 按钮即可关闭 86EVA，否则窗口将于 10 秒后自动关闭。

产生的代码与文件如下：



a. sketch_jul23d：主项目（.ino，文件名依您的项目名称而定）。

b. myeva.cpp：86EVA 的 C++ 代码。

c. myeva.h：86EVA 的标头档。

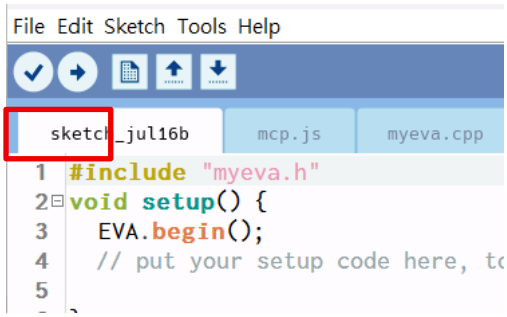
接着开始编写代码，设置您的 EtherCAT 系统。

请在 `void setup() { }` 中的 `EVA.begin();` 之前加入 `Serial.begin(115200);`，如此 EVA 程序的回传值才能显示于 Serial Monitor（波特率 115200）。

示例代码如下：

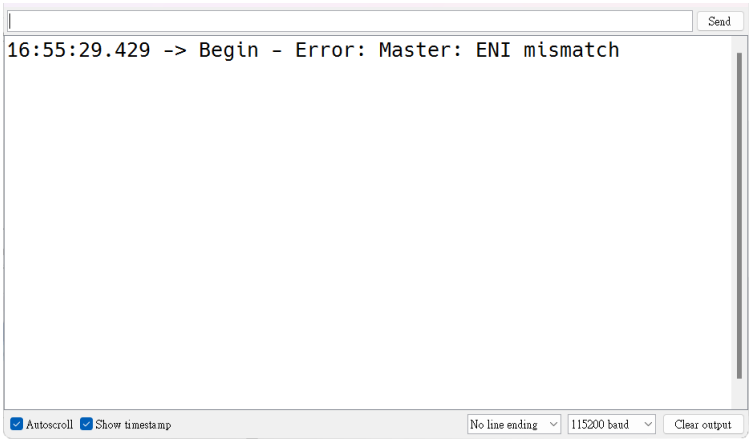
```
#include "myeva.h"

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:
}
```

代码完成后，请点击工具列的  进行编译；确认编译完成且无错误后，即可点击  进行上传。上传完成后程序便会开始运行。



程序上传成功后，必须通过 Serial Monitor 确认 ENI 文件已正确导入，且网络运行正常。

若 ENI 文件导入成功，**Serial Monitor 不会有任何回传**；若发生错误，则会直接将错误回传至 Serial Monitor。



4.6 Bootloader Menu 使用说明

本节介绍 Bootloader Menu，它提供友善的操作界面，供您设置与管理 QEC MDevice。



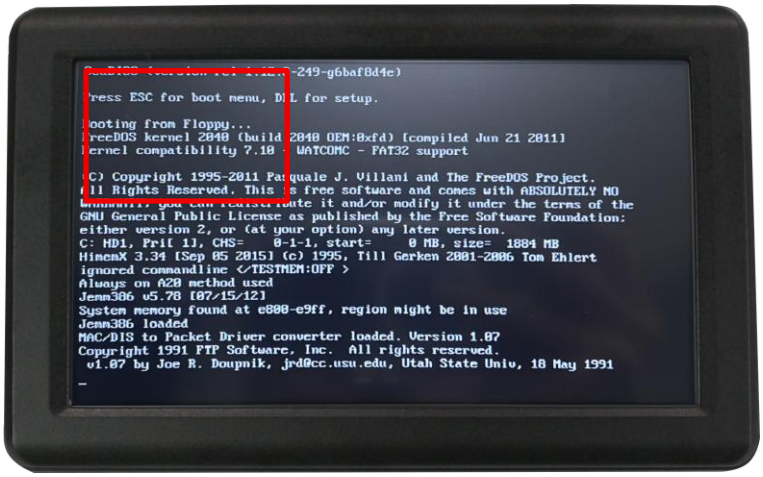
Bootloader 是 86Duino 上电后运行的第一个程序，在 86Duino 硬件与开发环境之间扮演关键的桥梁角色。Bootloader 通过 USB Device / Programming Port 与 86Duino Coding IDE 通信，让用户能将 sketch 程序由开发环境顺畅地上传至 86Duino 硬件。
Bootloader 会接收用户上传的 sketch 程序，并写入 86Duino 设备的板载内存。sketch 上传成功后，Bootloader 会自动运行该程序。

本节详细说明如何有效操作与运用 Bootloader Menu 的各项功能。

本节以 **QEC-PPC-M-090T** 为例说明。

4.6.1 打开 Bootloader Menu

QEC-PPC-M MDevice 开机时，持续触摸 LCD 屏幕左上角即可打开 Bootloader Menu。



持续按住左上角后，QEC MDevice 画面上会出现三个提示列。

第一列会显示“**Keep Pressing**”。



第二列会显示“**Touch Calibration**”。



第三列会显示“**Bootloader Menu**”。

在此列时放开屏幕，即可进入 Bootloader Menu 页面。



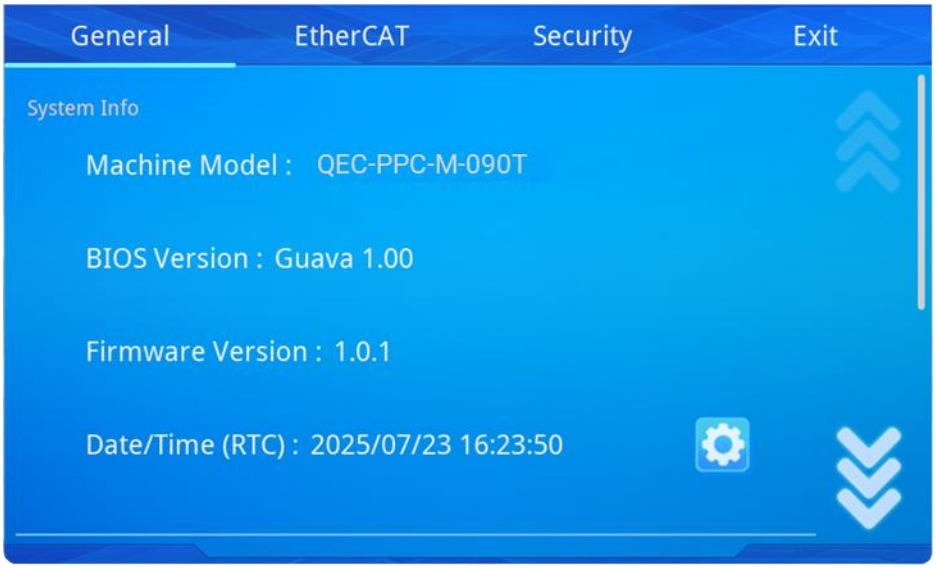
Bootloader Menu 的画面如下图所示。



以下说明仅以 Bootloader 的画面为例。

***注意：** 所有 QEC-PPC-M 系列 MDevice 皆可用相同方式打开 Bootloader Menu。

Bootloader Menu 的上方菜单可让您快速访问主要设置与功能。



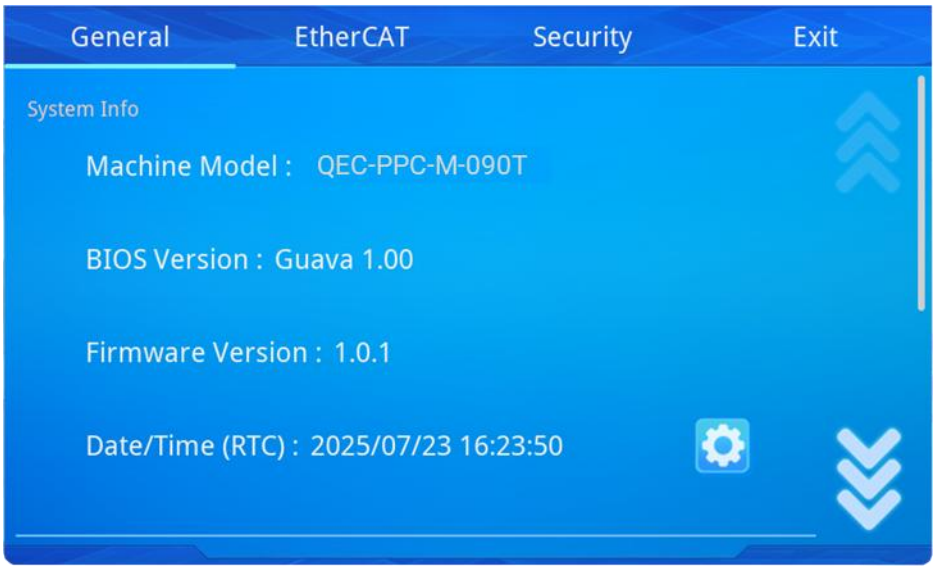
包含下列页签：

1. **General：**
   - 查看系统信息（例如机型型号、BIOS 版本、firmware 版本与 RTC）。
   - 访问基本设置，例如启用 BIOS 菜单或允许 IDE 连接。
2. **EtherCAT：**
   - 设置 BIOS 默认的 EtherCAT cycle time，并启用 / 禁用 redundancy。
3. **Security：**
   - 设置或更新 Bootloader 密码，以保护访问权限。
4. **Exit：**
   - 保存或舍弃变更并重新开机。

4.6.2 General 页面

General 页面显示重要的系统信息，并提供基本设置选项。

4.6.2.1 系统信息

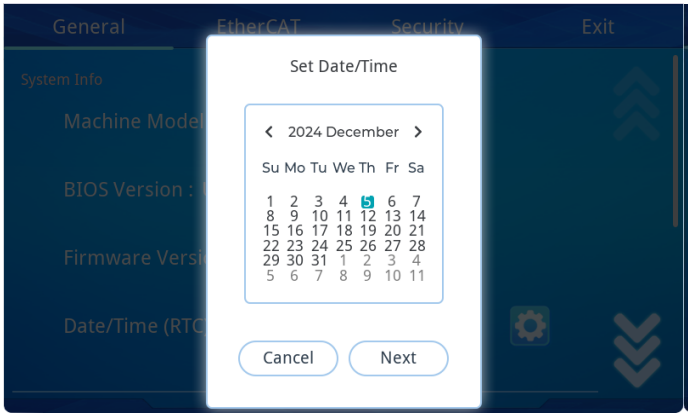


- **Machine Model**：显示目前的硬件机型（例如 QEC-PPC-M-090T、QEC-PPC-M-150T）。
- **BIOS Version**：显示已安装的 BIOS 版本。
- **Firmware Version**：显示目前安装的 firmware 版本。
- **Date/Time (RTC)**：显示系统的实时时脉，用户可自行调整此设置。

***补充说明：**

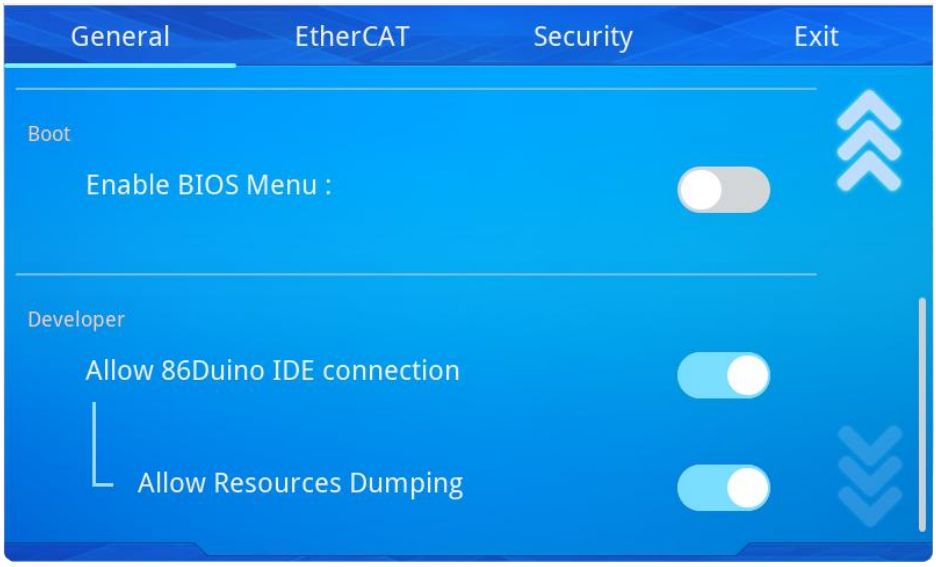
调整 Date/Time (RTC) 的方式：

1. 点击齿轮图标 打开“Set Date/Time”窗口。
2. 通过画面上的选项修改日期与时间设置。
3. 点击 OK 保存变更，或点击 cancel 舍弃变更。





4.6.2.2 Boot 与 Developer 选项



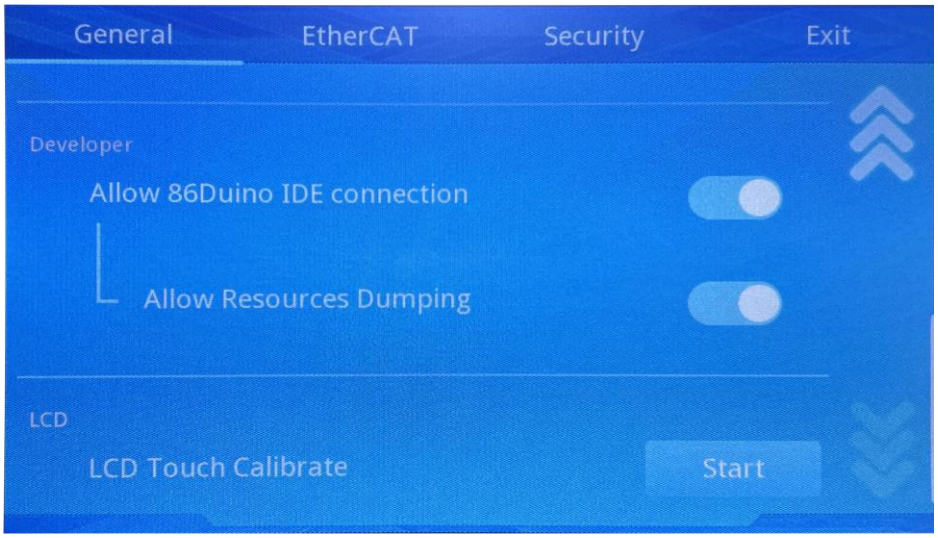
Boot：

- **Enable BIOS Menu**：启用或禁用 BIOS 菜单。

Developer：

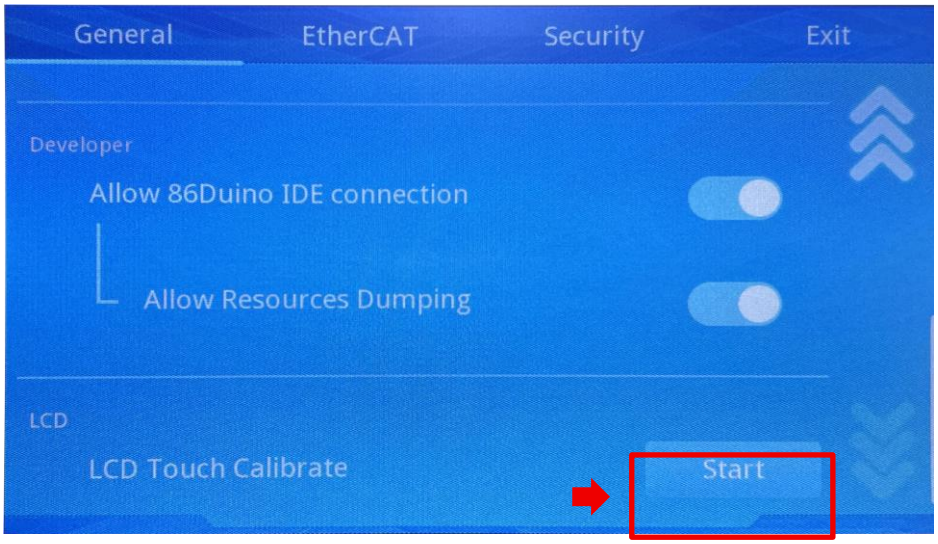
- **Allow 86Duino IDE connection**：供开发用途使用。
- **Allow Resources Dumping**：启用或禁用资源倾印功能，供调试使用。

4.6.2.3 LCD 选项

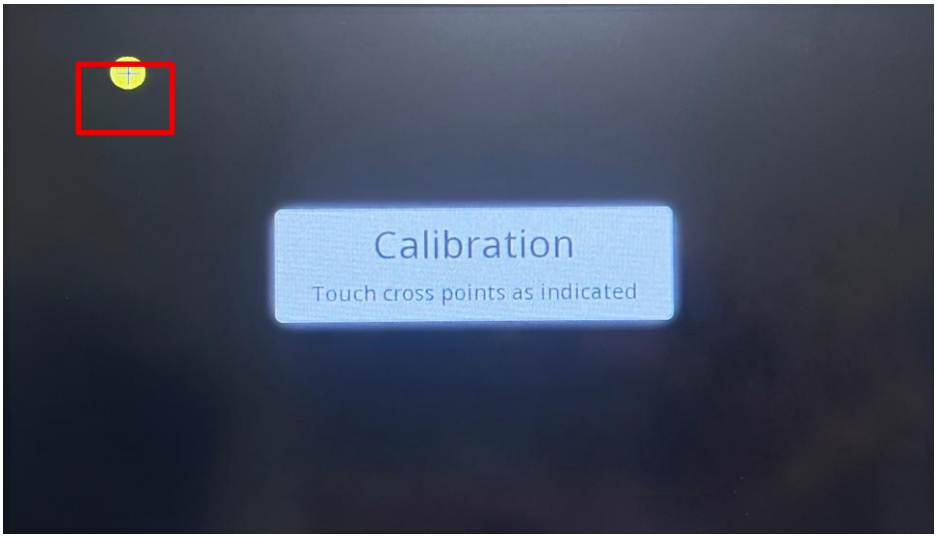


- **LCD Touch Calibrate**：启动 QEC MDevice 的 LCD 触摸校正功能。

点击 LCD Touch Calibrate 的 “**Start**”按钮



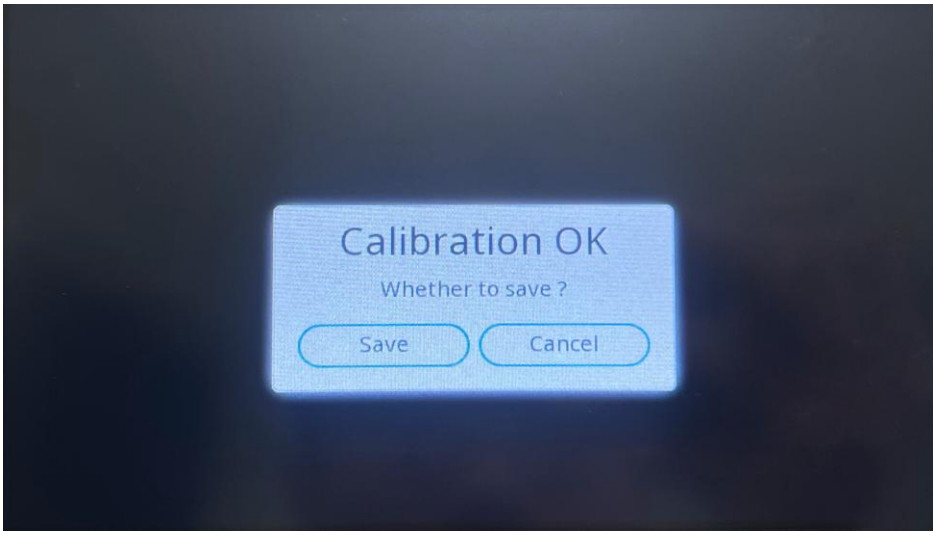
画面四个角落会依序显示四个黄色圆圈，请依序触摸这些圆圈以完成 LCD 触摸校正。



完成四个角落的圆圈后，屏幕中央会再出现一个黄色圆圈，请一并触摸。



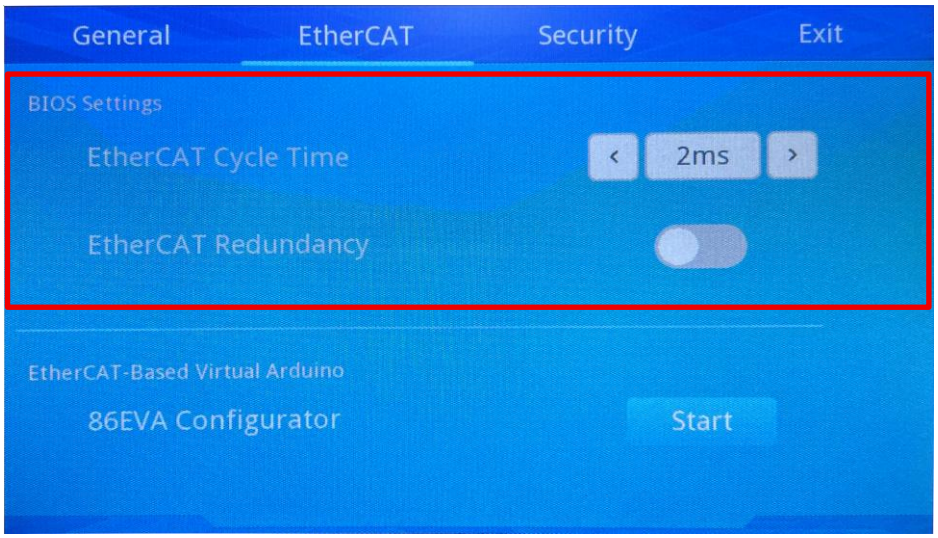
至此即完成 LCD 触摸校正。请点击“**Save**”，即可返回 Bootloader Menu。



4.6.3 EtherCAT 页面

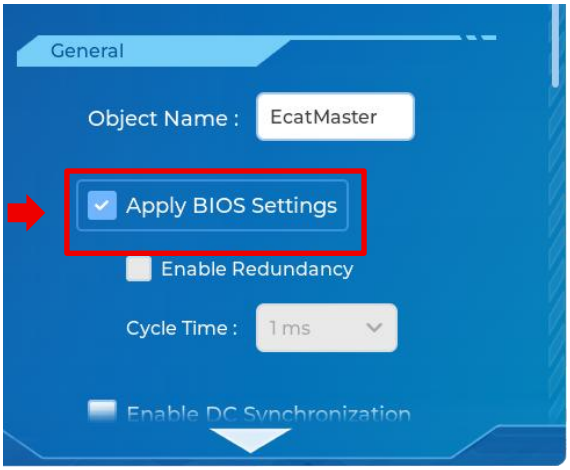
EtherCAT 页面可供用户设置 EtherCAT 与 86EVA Configurator 的相关项目，包括 Cycle Time 与 Redundancy，这些皆属于 BIOS 默认值。

4.6.3.1 BIOS 设置



- **EtherCAT Cycle Time**：设置默认的 EtherCAT 通信 cycle time，可选项目为 62.5μs、125μs、250μs、500μs、1ms 与 2ms。
- **EtherCAT Redundancy**： 启用或禁用默认的 EtherCAT redundancy 配置。

这些 BIOS 设置会直接影响 86EVA Configurator 中的 EtherCAT 配置，如 QEC MDevice 页面所示。

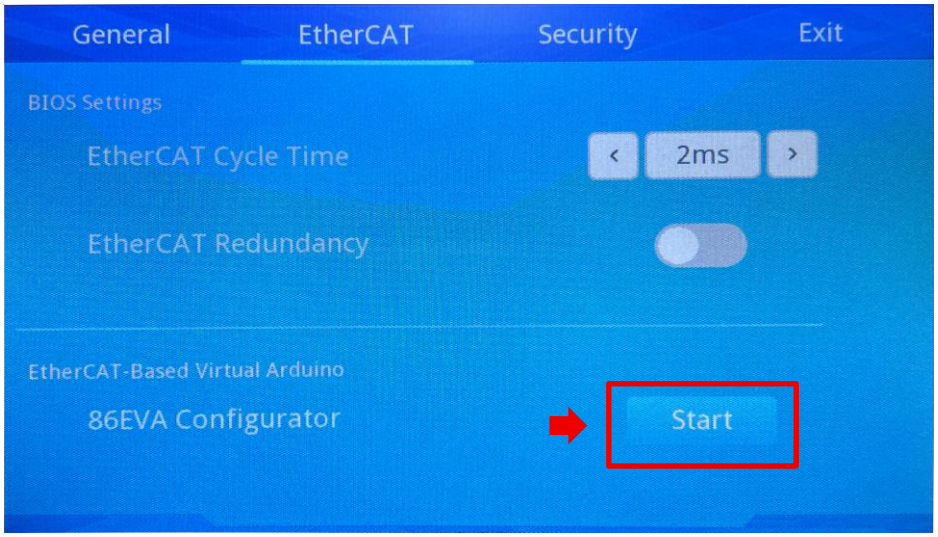


在 EtherCAT 页面启用 redundancy 后，86EVA Configurator 中的 Redundancy 复选框会自动被勾选；而在 BIOS 中设置的 Cycle Time，也会成为 Configurator 中的默认值。

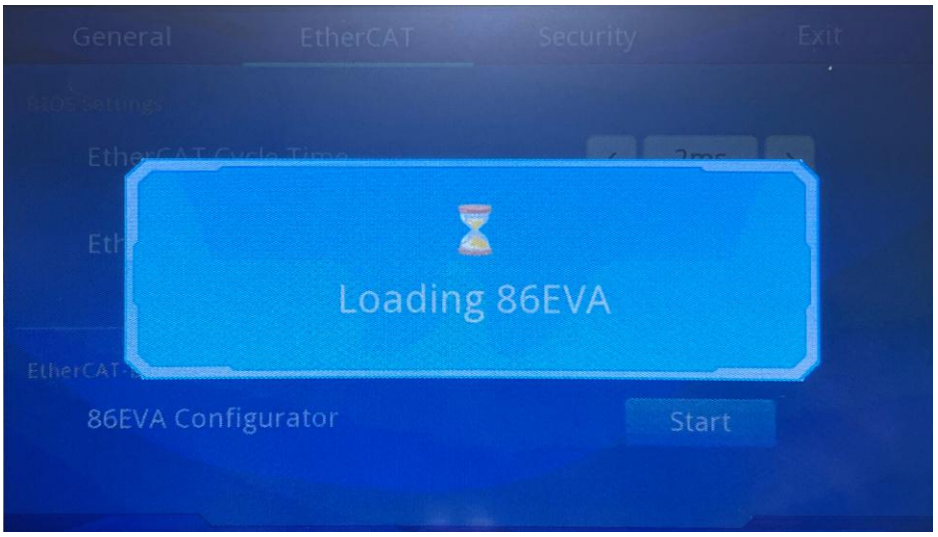
4.6.3.2 EtherCAT-Based Virtual Arduino

此功能可让用户通过 86EVA Configurator 扫描并设置 EtherCAT-Based Virtual Arduino 的对应关系。

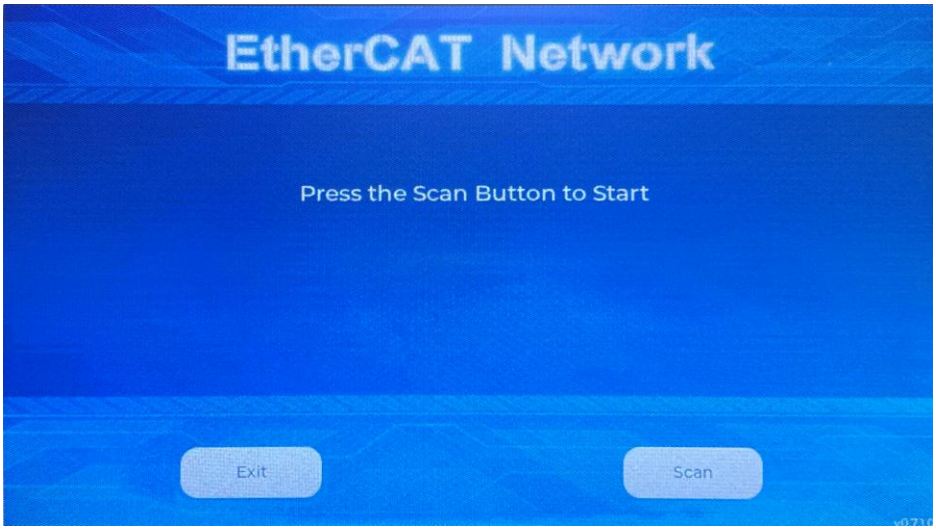
点击“**Start**”按钮即可进入 86EVA 窗口。



点击“**Scan**”按钮即可开始扫描 EtherCAT 网络。



QEC-PPC-M 系列 MDevice 上的 86EVA。

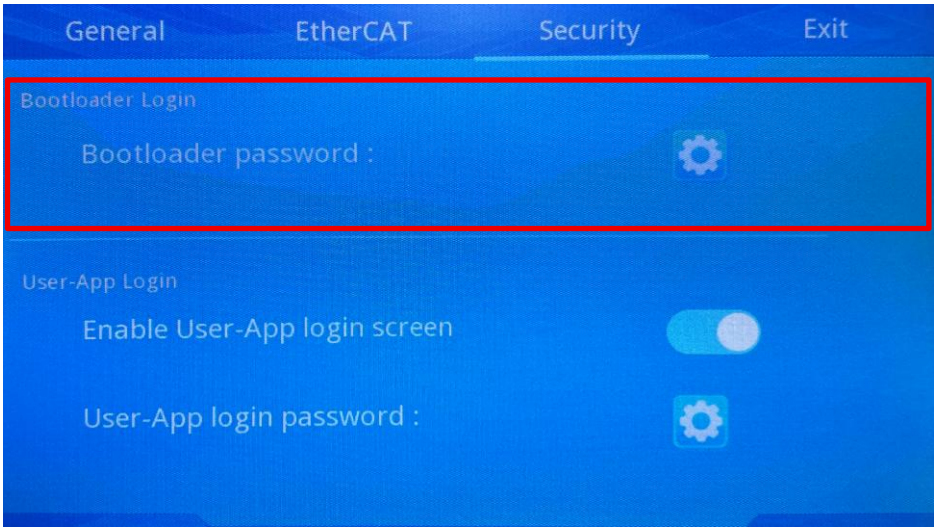


*注意：QEC-PPC-M 系列 MDevice 上的 86EVA Configurator 仍在开发中，目前不建议使用。

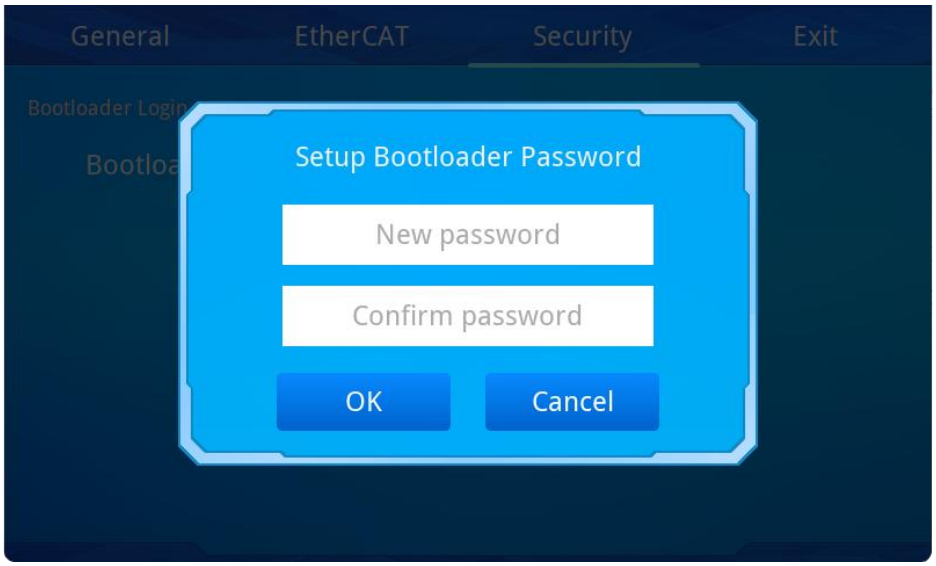
4.6.4 Security 页面

Security 页面可供用户设置与管理 Bootloader 密码，以保护程序与配置设置，确保只有经授权的用户才能访问 Bootloader Menu。

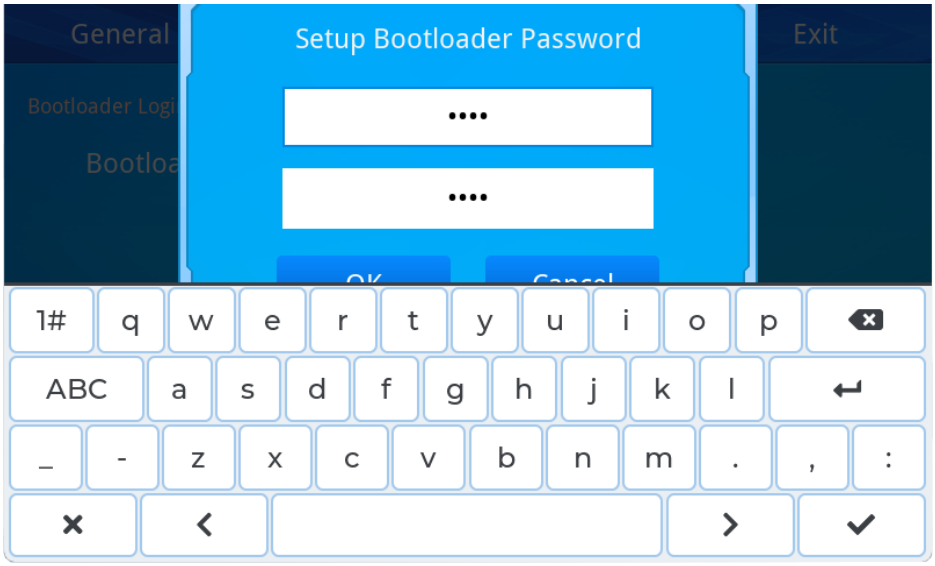
4.6.4.1 Bootloader 登录



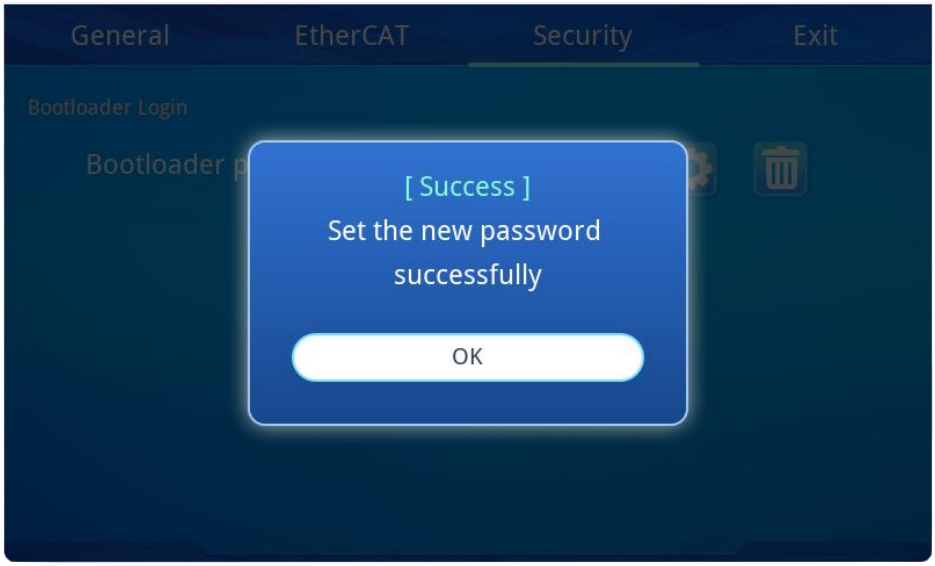
- **Bootloader password**：用户可点击齿轮图标 打开“**Setup Bootloader Password**”窗口，如下图所示。



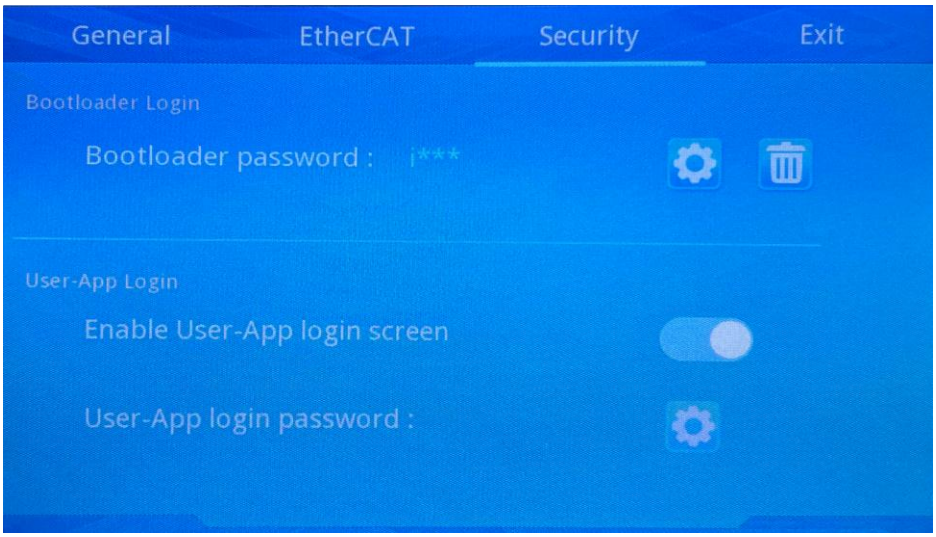
点击文本输入区后，画面上会出现键盘。



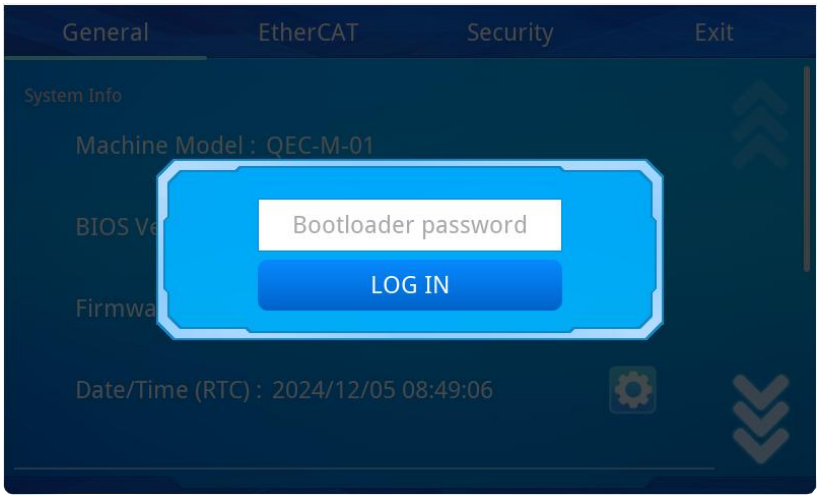
设置完 Bootloader 密码后点击“**OK**”，画面会出现【**Success**】窗口，表示新密码已设置成功。



此时 Security 页面会显示如下。



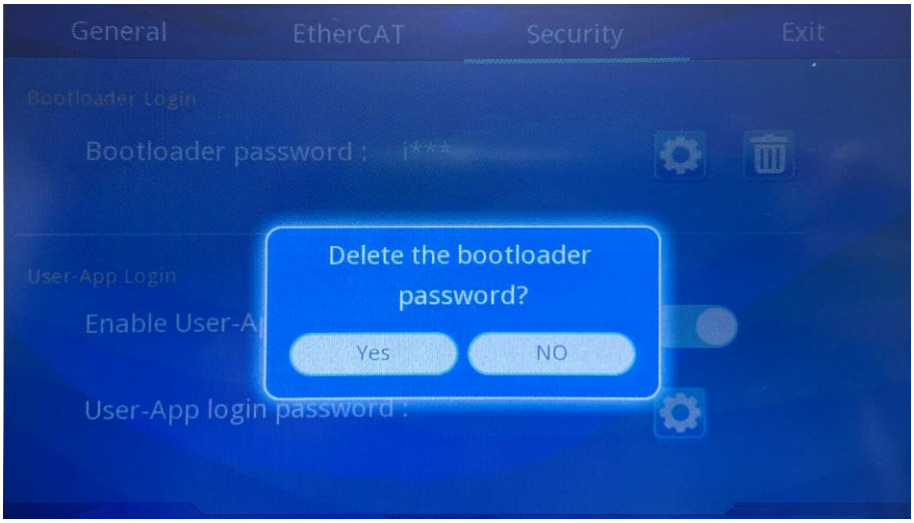
保存 Bootloader Menu 的配置后，密码即会一并保存，下次打开 Bootloader Menu 时便会要求输入。



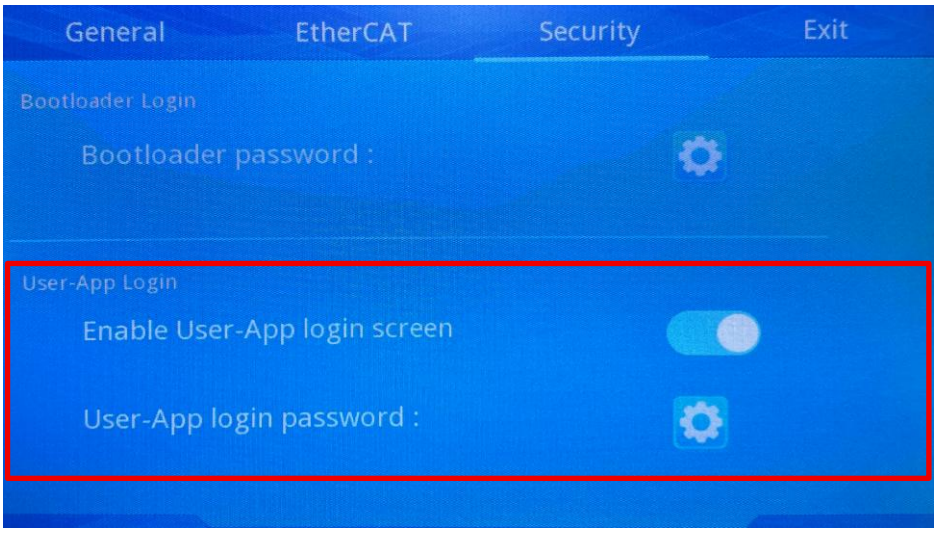
用户可点击齿轮图标变更密码，或点击垃圾桶图标删除密码。



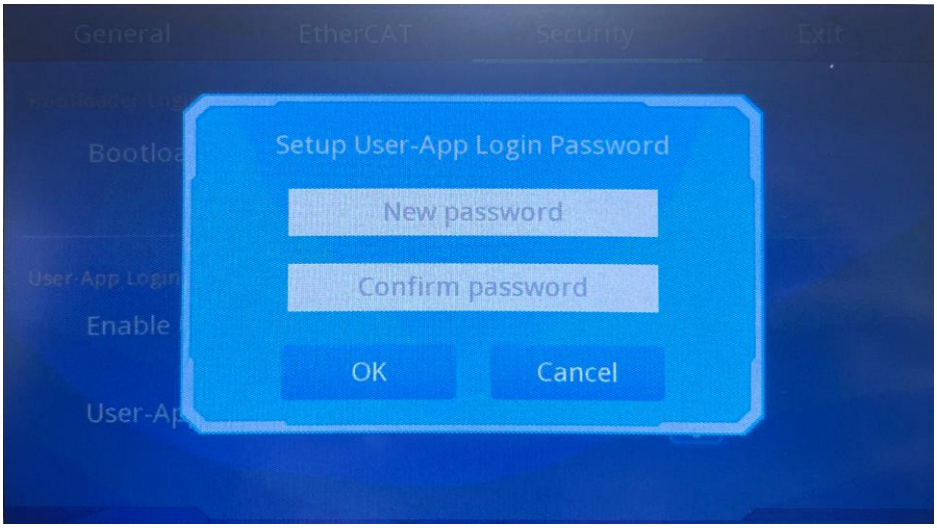
点击垃圾桶图标删除 Bootloader 密码时，画面会显示下列确认窗口。



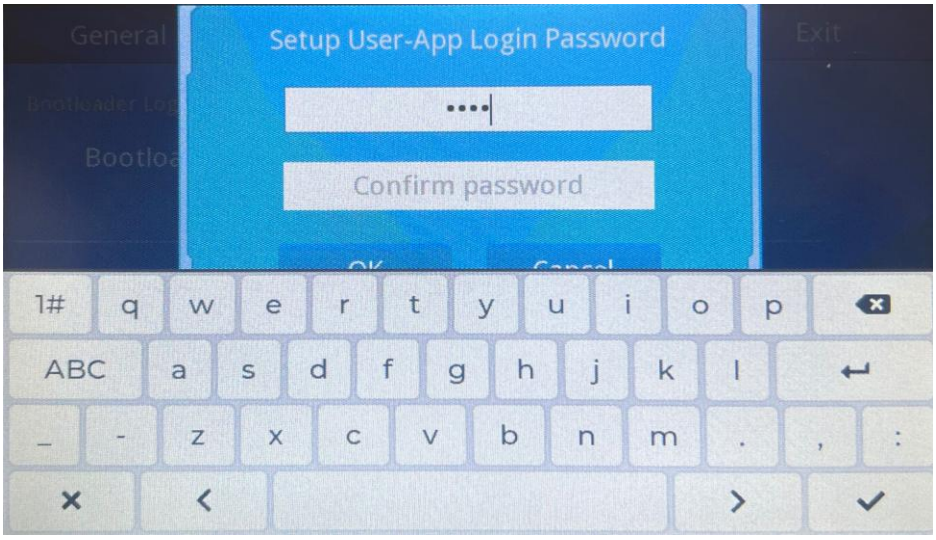
4.6.4.2 User-App 登录



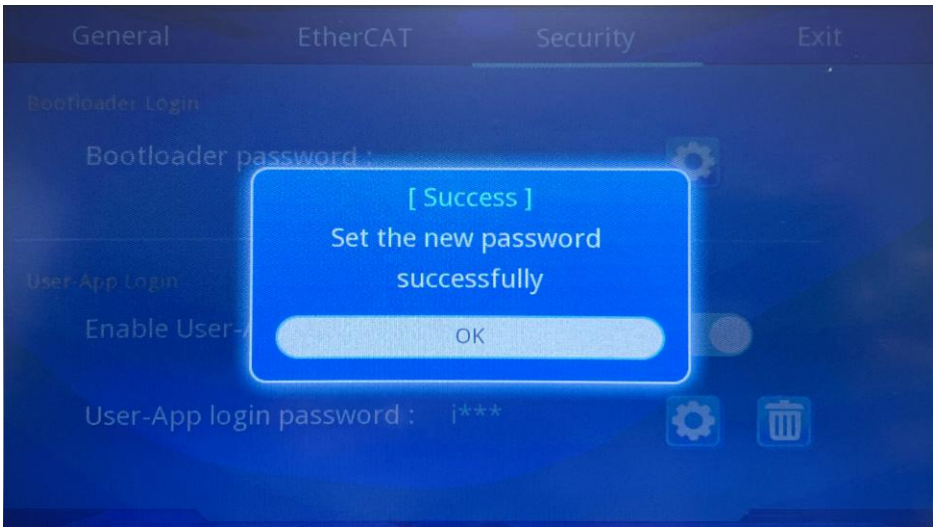
- **Enable User-App login screen**： 勾选此选项后，若已设置密码，QEC MDevice 在运行用户应用程序前会先显示登录画面。
- **User-App login password：** 用户可点击齿轮图标 打开“**Setup User-App Login Password**”窗口，如下图所示。



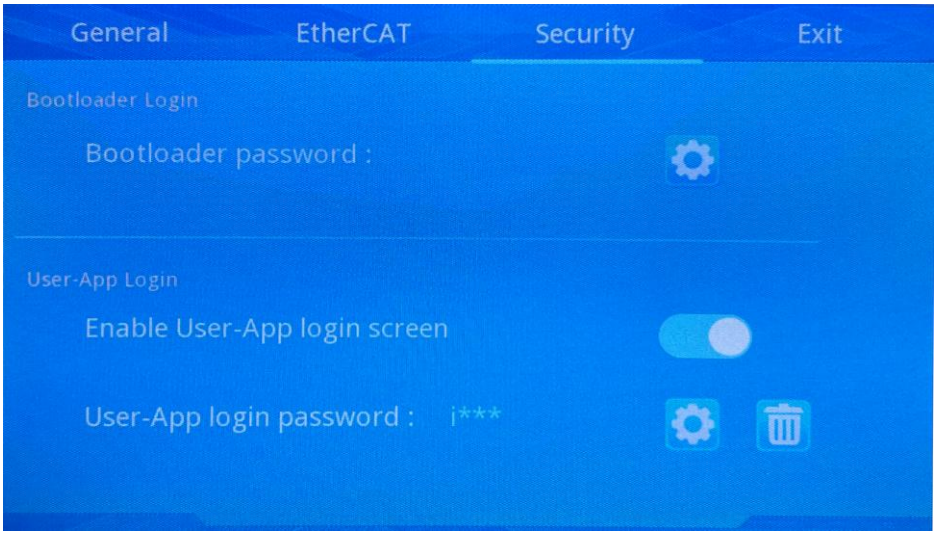
点击文本输入区后，画面上会出现键盘。



设置完 User-App 登录密码后点击“**OK**”，画面会出现【**Success**】窗口，表示新密码已设置成功。

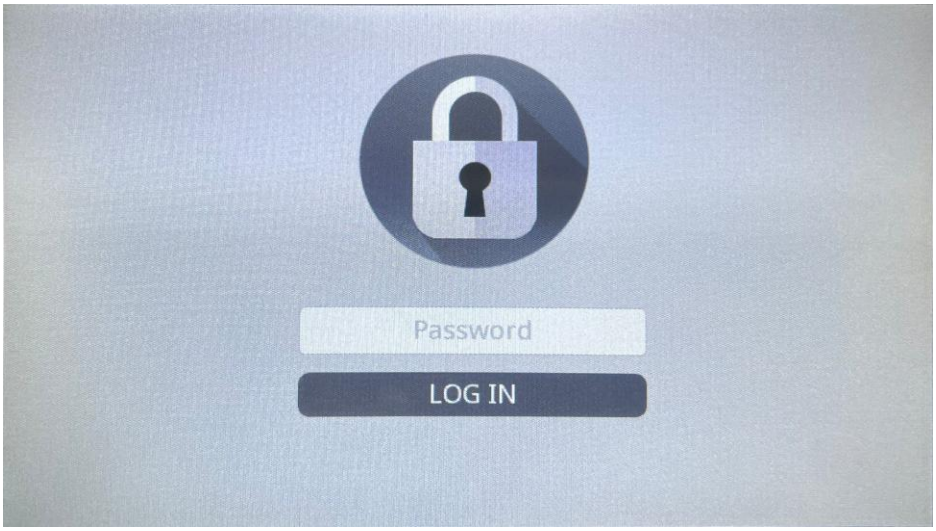


此时 Security 页面会显示如下。



保存 Bootloader Menu 的配置后，密码即会一并保存，下次打开 QEC MDevice 时便会显示 User-App 登录画面。

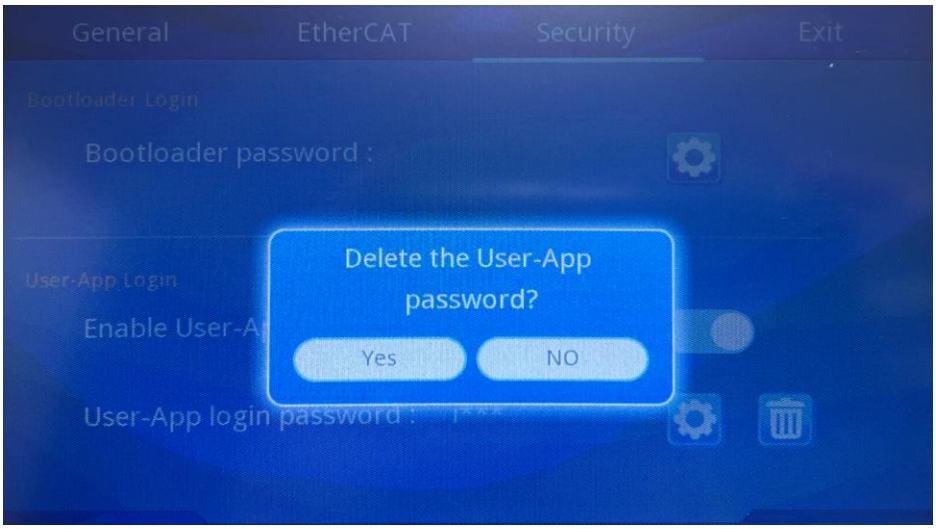
如下图所示。



用户可点击齿轮图标变更密码，或点击垃圾桶图标删除密码。



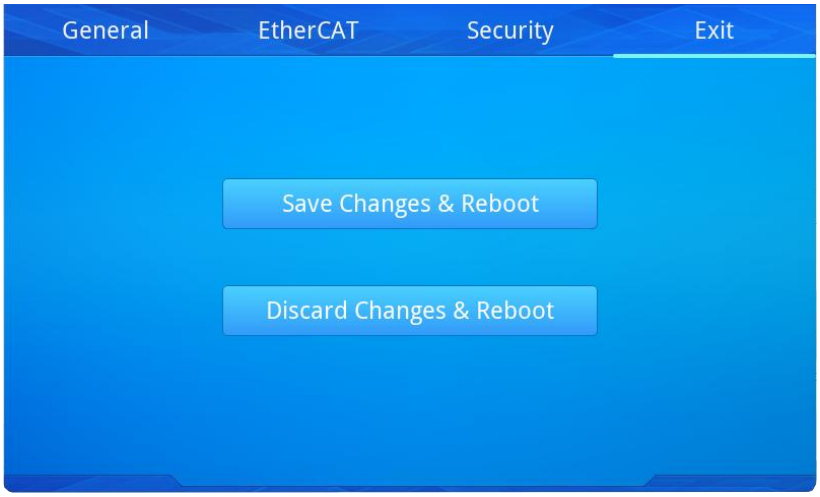
点击垃圾桶图标删除 User-App 登录密码时，画面会显示下列确认窗口。



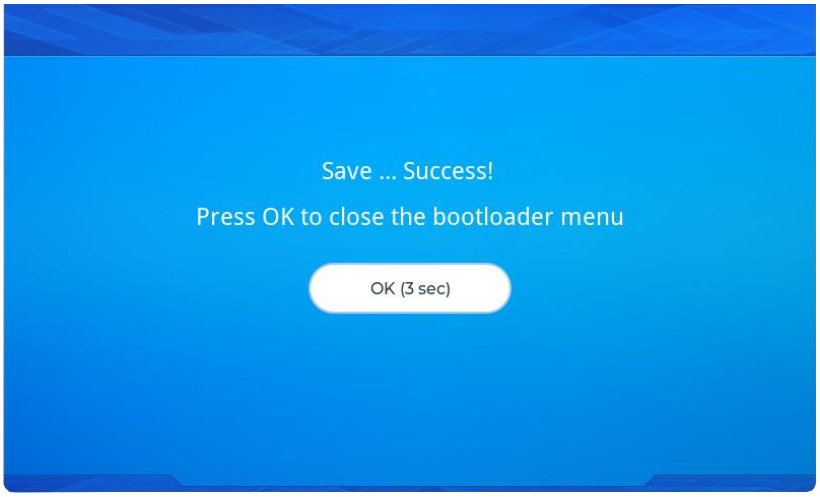
4.6.5 Exit 页面

Exit 页面可供用户在重新启动 QEC MDevice 前，选择保存或舍弃于 Bootloader Menu 中所做的变更，确保所有设置依用户意愿正确套用或还原。

Exit 页面：



- **Save Changes & Reboot：** 选择此项目时，于 Bootloader Menu 中所做的所有变更都会被保存。Bootloader Menu 会显示“Save... Success”确认消息，表示配置已成功保存。



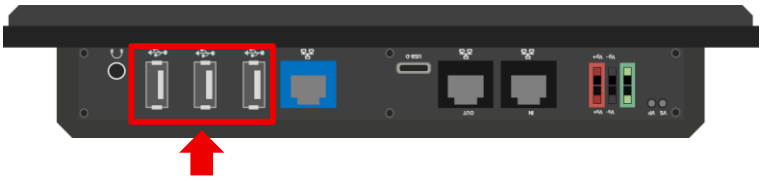
用户可点击 OK 立即关闭 Bootloader Menu，或等待 5 秒后自动关闭。

- **Discard Changes & Reboot：** 选择此项目会舍弃所有尚未保存的变更，并以先前保存的配置重新启动 QEC MDevice。Bootloader Menu 会直接关闭而不进行保存。

4.7 USB 设备使用说明

本节说明如何正确使用 QEC-PPC-M 系列 MDevice 的 USB 设备与文件保存功能。

QEC-PPC-M 系列 MDevice 具备 3 组支持热插拔的标准 USB 2.0 端口，用户可连接 USB 存储设备，用于文件保存、传输或访问配置数据。



用户可利用 SD Library 读写 USB 文件夹，例如通过此 library 创建 .txt 文件。在 QEC-PPC-M 系列上，SD library 可读写 USB 存储设备或 eMMC 存储空间，并支持其上的 FAT16 与 FAT32 文件系统。传入 SD library 函数的文件名可包含以正斜线 / 分隔的路径，例如 “directory/filename.txt”。由于工作目录永远是所选存储设备的根目录，文件名前是否加上斜线都指向同一个文件（例如“/file.txt”等同于“file.txt”）。此 library 支持同时打开多个文件。
不过若要将文件放入特定文件夹，则无法使用 SD Library，用户必须改用标准 C 语言的 `fopen()` 函数手动指定文件位置。

***警告：存储槽使用建议**

QEC-PPC-M 系列 MDevice 具有 A、B、C、P 四个存储槽。

- **A** 与 **B** 槽：保留给系统文件使用。强烈建议用户不要将文件存放于此，否则可能造成系统损毁，需回复原厂设置或重新烧录 32MB flash。
- **C** 槽：此槽对应 eMMC，建议用于安全存放用户文件。
- **P** 槽：此槽对应 USB 设备，同样适合存放用户文件。

4.7.1 示例 1：将 .txt 保存至 USB U 盘

以下示例示范如何以 C 语言的 `fopen()` 创建文件，并将 .txt 档存放至**外置 USB U 盘**，说明如何指定文件位置并确保文件存放于适当的存储槽。

示例代码如下：

```
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:

}
```

***注意：** 请在 QEC MDevice PPC 系列上使用外置 USB U 盘。

4.7.2 示例 2：将 .txt 保存至 eMMC

以下示例示范如何以 C 语言的 `fopen()` 创建文件，并将 .txt 档存放至**内置 eMMC**，说明如何指定文件位置并确保文件存放于适当的存储槽。

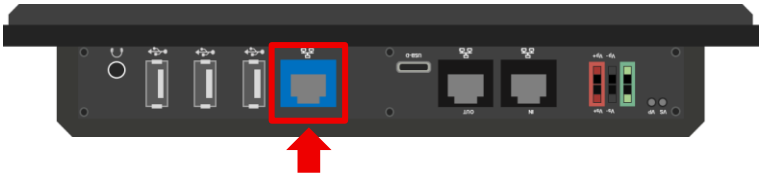
示例代码如下：

```
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 C:\\
   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:

}
```

4.8 Giga LAN 设置

本节介绍 QEC MDevice 的 Giga LAN 设置与控制方式。

QEC-PPC-M 系列具备一组 Giga LAN 端口，专供一般网络用途的对外 Ethernet 通信使用。



***注意：** QEC-PPC-M MDevice 上的 Giga LAN 为蓝色外壳。

若要驱动 Giga LAN，可在 86Duino 环境中使用 Ethernet Library 或 Modbus Library。

Ethernet Library：

- 提供一般网络应用所需的工具，包括：
- 静态与动态 IP 设置。
- TCP/UDP 通信。
- 架设 Web Server。

更多细节请参见 Ethernet Library Documentation。

Modbus Library：

- 可通过 Ethernet 与 Modbus TCP 设备通信。
- 适合工业自动化与监控应用。

更多信息请参见 Modbus Library Documentation。

4.8.1 Ethernet 通信

本节介绍 QEC MDevice 的 Ethernet 设置与控制方式。

QEC-PPC-M 系列的 CPU 内置 10/100/1000Mbps LAN 接口，可通过 Ethernet library 访问，无须另外搭配 Arduino Ethernet Shield。此 library 可作为接受连接的 server，也可作为对外连接的 client。在 86Duino Coding 中，此 library 最多支持四个同时连接（传入、传出或两者混合）；后续版本最多可支持 128 个同时连接。

4.8.1.1 示例 1：以 DHCP 取得并印出 IP

本示例使用 Ethernet library 的 DHCP 扩展功能，通过 DHCP 取得 IP 地址并印出。

示例代码如下：

```
#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() {

}
```

4.8.1.2 示例 2：Web Server

一个简单的 Web Server，用于显示模拟输入引脚的数值。

示例代码如下：

```
#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("Connection: 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 disconnected");
  }
}
```

4.8.1.3 示例 3：DHCP Chat Server

一个简单的服务器，会将收到的消息转发给所有已连接的客户端。您可以用 telnet 连到设备的 IP 地址后输入文本，同时也能在 Serial Monitor 中看到客户端的输入内容。

示例代码如下：

```
#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);
  while (!Serial) {
    ; // wait for serial port to connect.
  }


  // 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);
  }
}
```

4.8.1.4 示例 4：MySQL Connector

本示例示范如何由 QEC MDevice 的 Ethernet 端口连接至 MySQL 服务器。

更多信息与文档请参见 wiki：https://github.com/ChuckBell/MySQL_Connector_Arduino/wiki。

示例代码如下：

```
#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() {

}
```

4.8.2 Modbus TCP 通信

本节介绍 QEC MDevice 的 Modbus TCP 设置与控制方式。

4.8.2.1 示例 1：简易 Modbus Master TCP

本示例使用 Ethernet 与 ModbusMaster library，通过 Ethernet 创建 Modbus TCP 通信。程序会以静态 IP 地址初始化 Ethernet 连接，并设置 Modbus TCP master 与 ID 为 11 的 slave 设备交互。

示例代码如下：

```
#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);
}
```

4.9 HMI 设计

本节示范如何在 QEC-PPC-M 设备（面板电脑系列）上创建基本的人机界面（HMI），可使用 LVGL 函数库或 86Duino Coding IDE 501+ 内附的 86HMI Editor。

我们假设您已完成快速入门指南前面各节的步骤，包括包装清单确认、硬件配置、软件驱动程序安装，并已完成 QEC-PPC-M MDevice 的设置。

4.9.1 函数库说明

LVGL（Light and Versatile Graphics Library）是一套功能强大的开源图形函数库，可为嵌入式系统打造美观且具交互性的图形用户界面（GUI）。将 LVGL 函数库与 86Duino IDE 搭配使用，开发者便能轻松且高效率地设计并实现 HMI。

您可运用按钮、滑杆、图表、图像等其他 LVGL 组件来扩展与客制化 HMI。LVGL 提供丰富的内置组件与客制化选项，可依您的应用需求打造精致且具视觉吸引力的 HMI。



关于 LVGL 的更多细节，请参见 https://docs.lvgl.io/7.11/index.html。

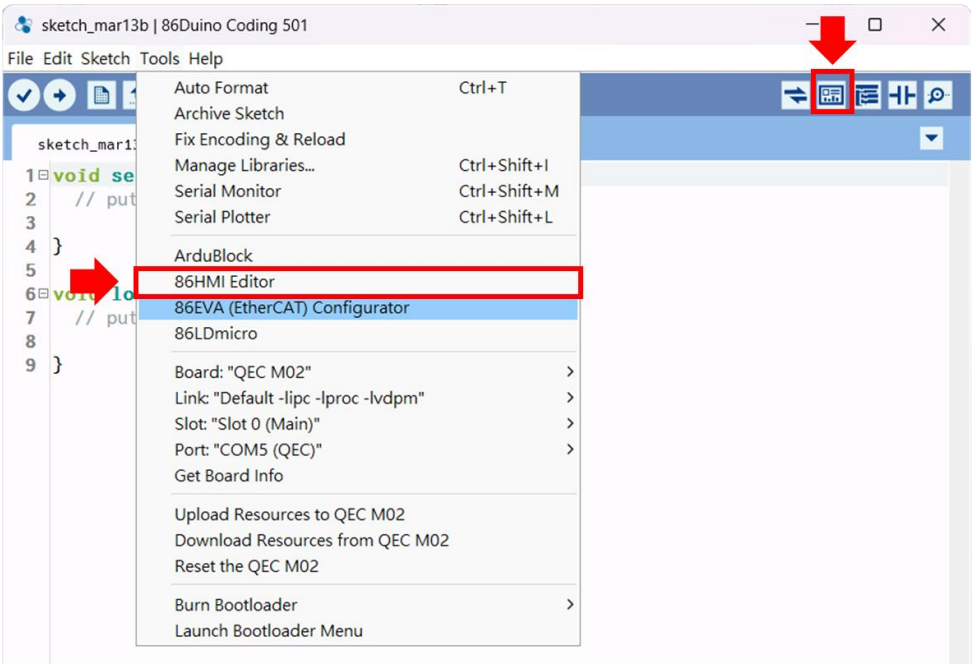
***注意：** 86Duino IDE 501+ 已完成 **LVGL 9.3 版**的测试，用户可自行导入 LVGL 函数库。

4.9.2 使用图形化 HMI 编辑器：86HMI Editor

86HMI Editor 是一套简单易用的 HMI 编辑器，以 LVGL 7.11 版为基础，可快速创建客制化的 HMI。通过 86HMI Editor 的自动产生代码（Auto Code Generation）功能产生 HMI API（应用程序接口），即可达成不需编写程序也能创建 HMI 的效果。

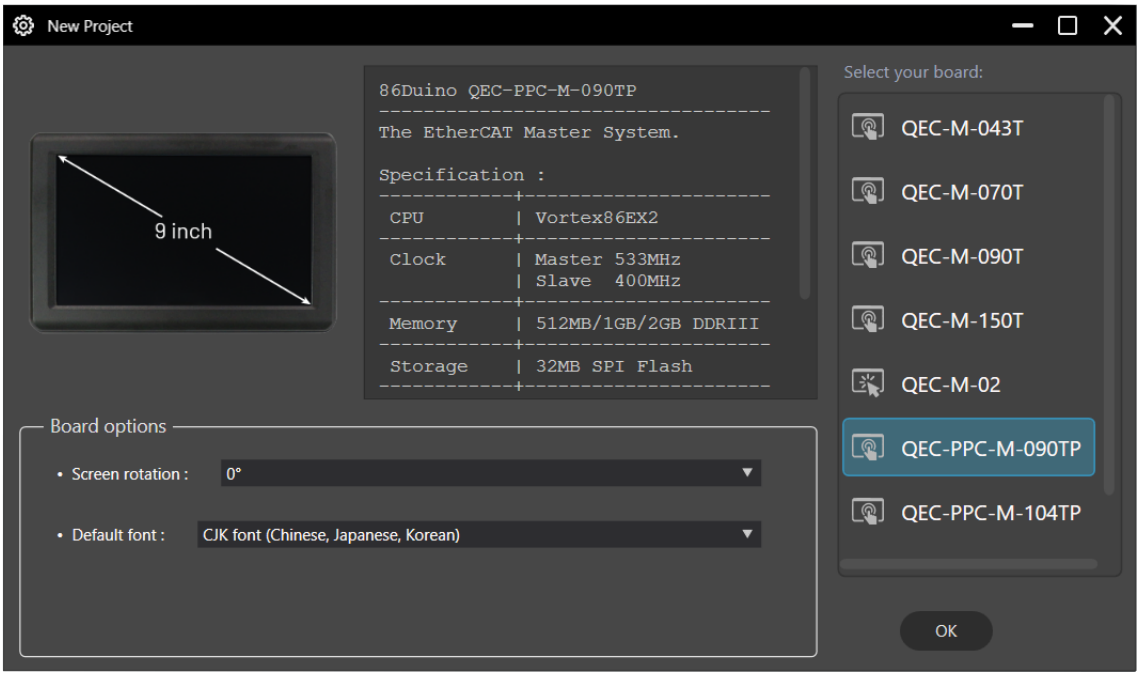
以下是使用 86HMI 为 QEC MDevice 设计 UI 的完整步骤：

1. 通过 86Duino Coding IDE 501+ 打开 86HMI 工具。

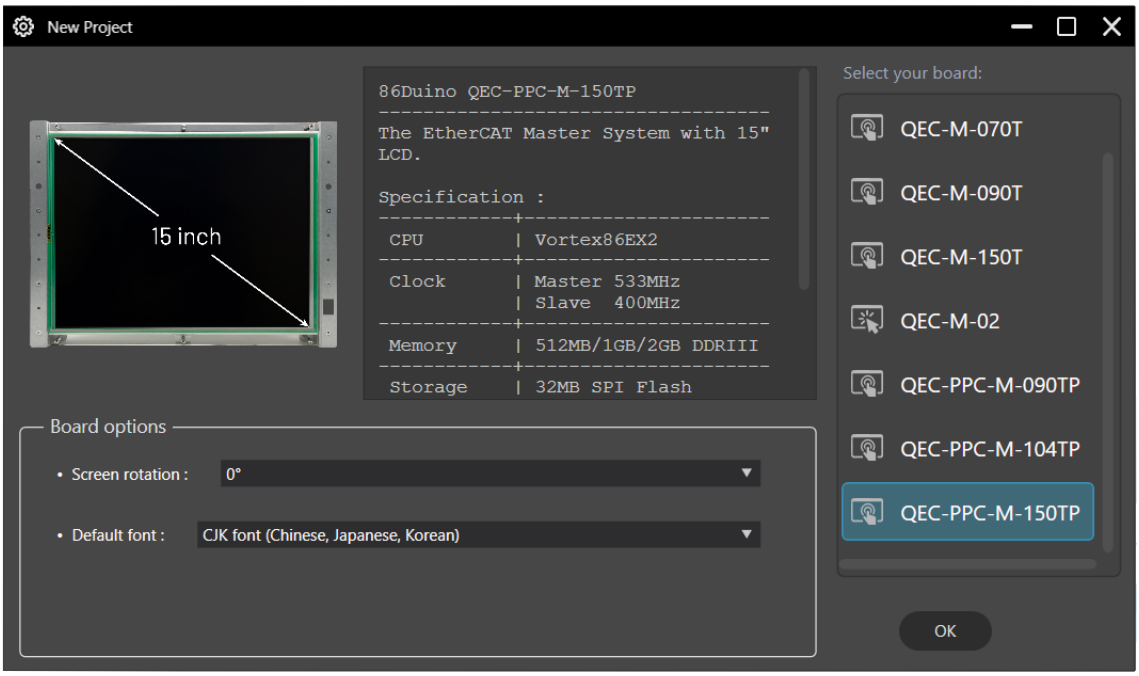


2. 选择您的 QEC MDevice。

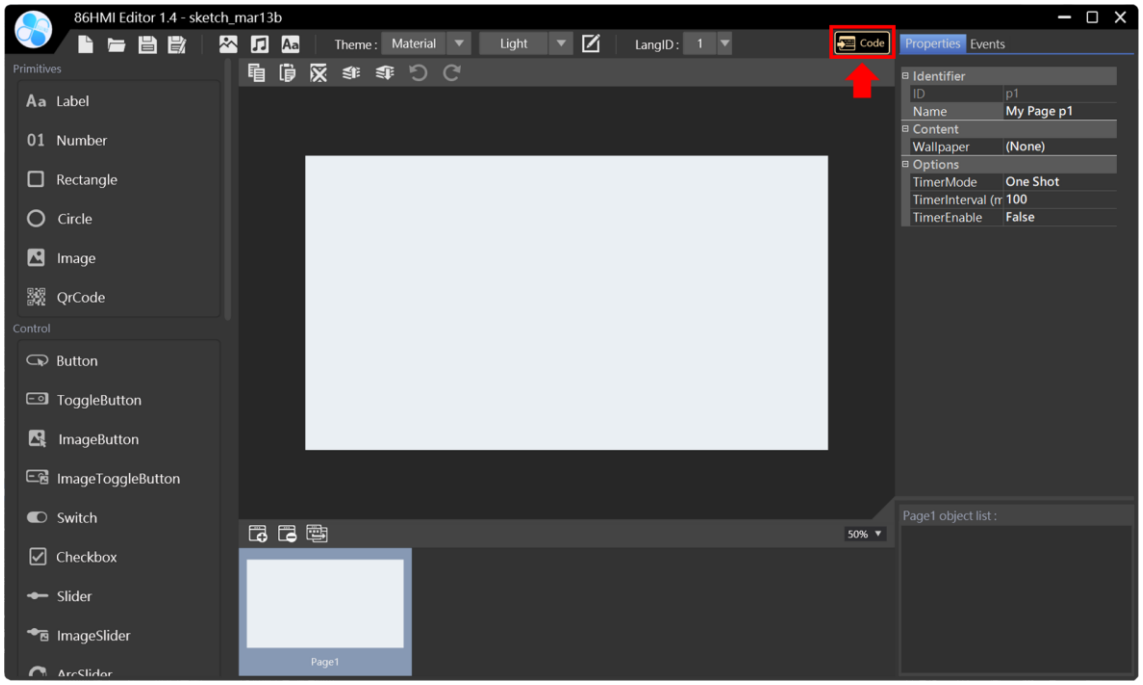
QEC-PPC-M-090T：



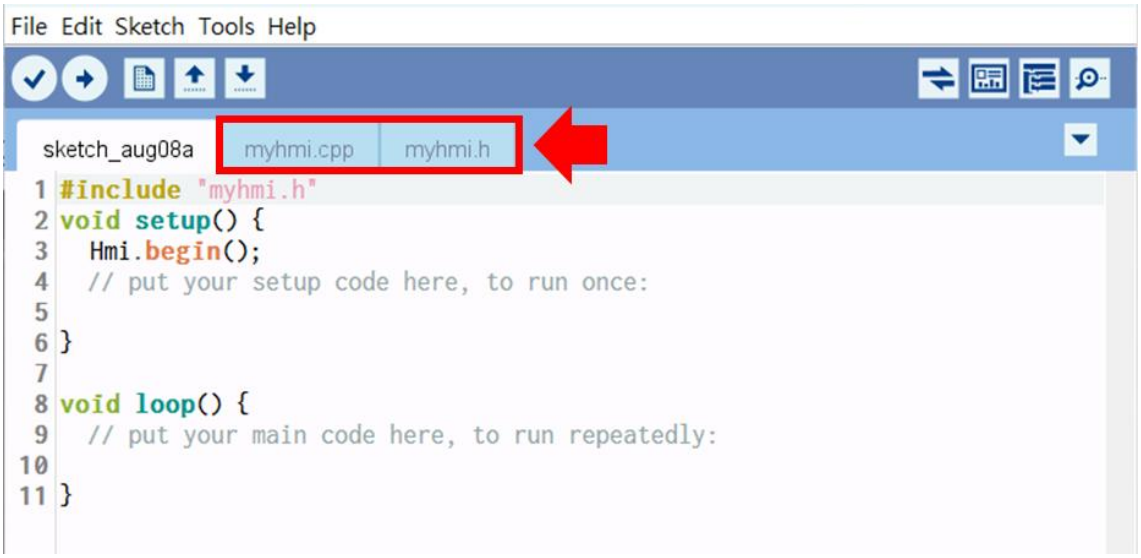
QEC-PPC-M-150T:



3. 接着即可使用左侧菜单开始设计 UI，并点击 **“Code”**按钮，自动将原代码产生回 86Duino IDE。



产生完成后，可在 86Duino IDE 中看到 **myhmi.h** 与 **myhmi.cpp**。

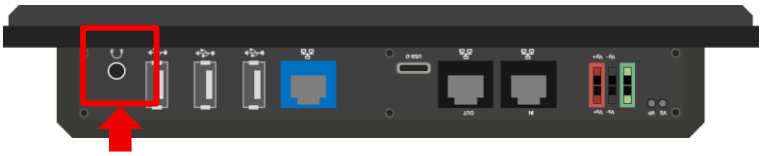


上传完成后，即可在 QEC MDevice 上看到您刚才设计的用户界面。

关于 86HMI Editor 的更多细节，请参见 86HMI 使用手册：
https://www.qec.tw/86duino/86hmi/。

4.10 音频使用说明

本节介绍 QEC MDevice 的 HD Audio 控制方式。
QEC-PPC-M-090T 与 QEC-PPC-M-150T 各具备一组 HD Audio 端口，用户可通过 Audio Library 播放音频。



此 library 自 86Duino Coding 102 起提供，实现了 Arduino Due 的 Audio library API，可访问 QEC-PPC-M-090T/150T 的板载 HD Audio 接口。

4.10.1 示例 1：简易音频播放器

示范如何在 QEC-PPC-M-090T/150T 上使用 Audio library。
本示例示范如何在 QEC-PPC-M-090T/150T 上使用 Audio library，播放存放于 USB 存储设备中的 .wav 音频档。程序会由文件读取音频数据，通过 Audio library 处理后实时输出。

示例代码如下：

```
#include <SD.h>
#include <Audio.h>

void setup() {
  Serial.begin(9600);
  SD.setBank(USBDISK);
  Serial.print("Initializing SD card...");
  if (!SD.begin(4)) {
    Serial.println(" failed!");
    return;
  }
  Serial.println(" done.");
  Audio.begin(88200, 100);
}

void loop() {
  int count=0;
  File myFile = SD.open("test.wav");
  if (!myFile) {
    Serial.println("error opening test.wav");
    while (true);
  }

  const int S=1024;
  short buffer[S];
```

```
  Serial.print("Playing");

  while (myFile.available()) {
    myFile.read(buffer, sizeof(buffer));
    int volume = 1024;
    Audio.prepare(buffer, S, volume);
    Audio.write(buffer, S);
    count++;
    if (count == 100) {
      Serial.print(".");
      count = 0;
    }
  }
  myFile.close();

  Serial.println("End of file. Thank you for listening!");
  while (true) ;
}
```

4.11 故障排除

4.11.1 无法顺利上传代码

当您无法顺利上传代码时，请打开 86EVA 检查 QEC EtherCAT MDevice 的环境是否异常。如下图所示，请尝试更新 QEC EtherCAT MDevice 的环境，更新内容包含下列三个项目：Bootloader、EtherCAT firmware 与 EtherCAT tool。



以下进一步说明更新的操作方式：

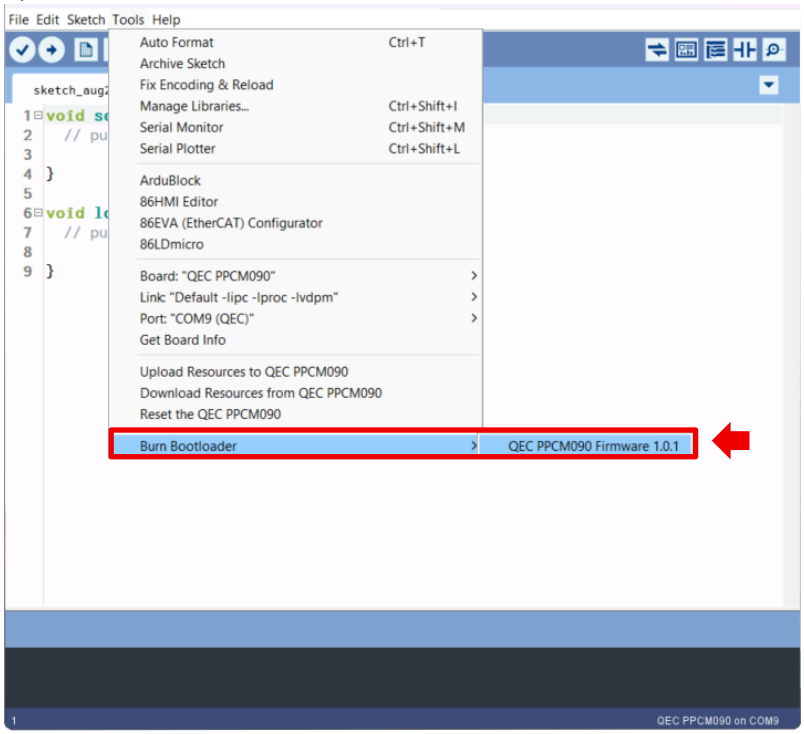
步骤 1：设置 QEC-M

1. 下载并安装 86Duino IDE 501+（或更新版本）：可由 Software 页面下载。
2. 连接 QEC MDevice：使用 USB 连接线将 QEC MDevice 连接至电脑。
3. 打开 86Duino IDE：安装完成后，打开 86Duino IDE 软件。
4. 选择开发板：在 IDE 菜单中选择“**Tools**”→“**Board**”→“**QEC-PPC-M-090T**”（或您所使用的 QEC MDevice 型号）。
5. 选择连接端口：在 IDE 菜单中选择“**Tools**”→“**Port**”，并选择 QEC MDevice 所连接的 USB 端口。

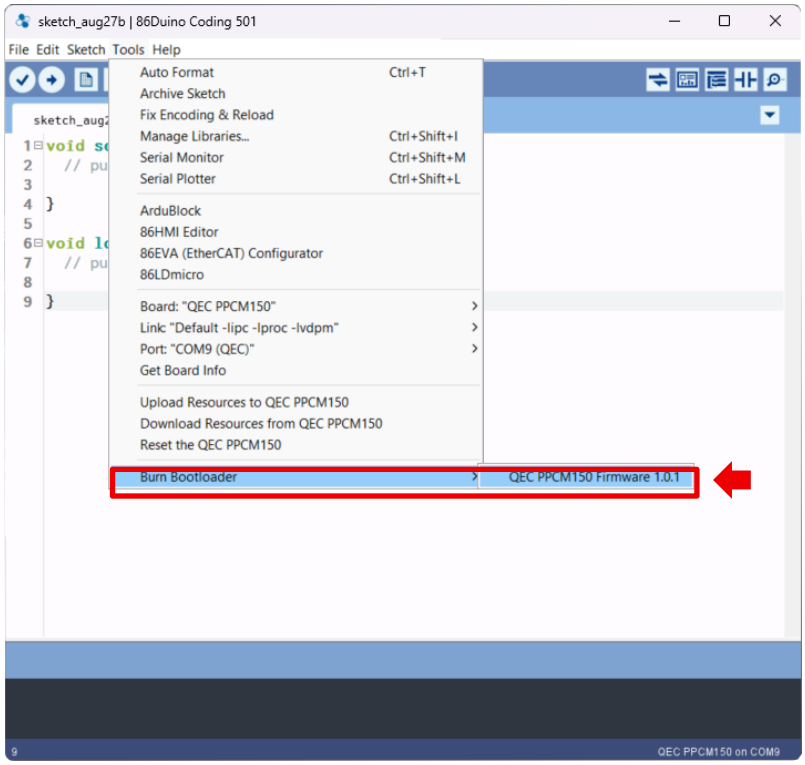
步骤 2：点击“Burn Bootloader”按钮

连接至您的 QEC MDevice 后，选择“**Tools**”→“**Burn Bootloader**”，画面会显示目前选择的 QEC MDevice 名称。点击后即开始更新进程，约需 5 至 20 分钟。

- QEC-PPC-M-090T:



- QEC-PPC-M-150T



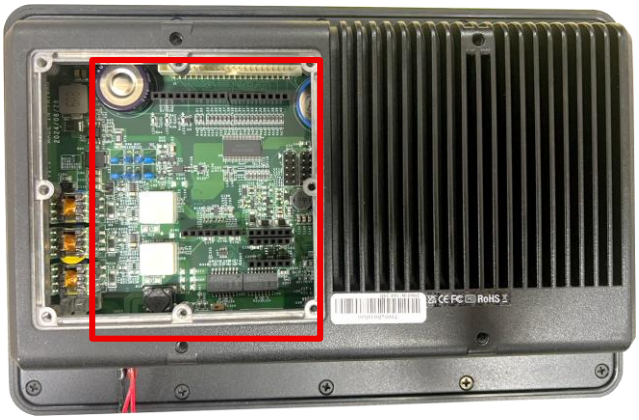
步骤 3：完成更新



完成上述步骤后，您的 QEC-M 便已成功更新至最新版本的开发环境。

4.12 Arduino 引脚使用说明

本节介绍 QEC MDevice 上的 GPIO 引脚配置，包含 Arduino 兼容引脚，以及 CAN、SPI、PWM 等 QEC 专属功能引脚。



QEC 系统采用 86Duino 架构，保留与 Arduino 程序模型的兼容性，用户可使用 `digitalWrite()`、`analogRead()`、`pinMode()` 等标准函数编写控制程序。

不过部分引脚提供 QEC 专属的高级功能，包括硬件 CAN bus、专用 SPI 接口与扩展的 PWM 输出，这些皆非标准 Arduino 开发板所具备。

为确保正确使用，用户编写软件时务必参照 86Duino 的引脚对应表。
请参考下方的 86Duino 引脚配置表，确认物理引脚位置与软件引脚编号之间的正确对应关系。

表：86Duino 引脚配置

连接器	引脚	信号	86Duino 引脚配置
J5	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
J4	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	(线材)
	10	I2C0_SCL	(线材)
J13	6	GP40ADC	14 (A0)
	5	GP41ADC	15 (A1)
	4	GP42ADC	16 (A2)
	3	GP43ADC	17 (A3)
	2	GP56ADC	18 (A4)
	1	GP57ADC	19 (A5)
J14	1	-	-
	2	GP35	23
	3	GP36	24
	4	GP37	25
	5	GND	-
	6	VCC	-
J16	1	CAN1_L	(CAN1)
	2	CAN1_H	(CAN1)
	3	GND	-
	4	CAN0_L	(CAN)
	5	CAN0_H	(CAN)
	6	VCC3	-

连接器	引脚	信号	86Duino 引脚配置
J7	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	-	-
J8	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)

4.12.1 GPIO

QEC MDevice 上的 General Purpose Input/Output (GPIO) 引脚，可通过 86Duino Coding IDE 以熟悉的 Arduino 风格函数进行程序控制。

电气特性：所有 GPxx (GPIO) 与 MCM 引脚的输出为 **VOH = 3.3 V**（逻辑高准位）与 **VOL = 0 V**（逻辑低准位）。每支引脚的输出驱动电流**默认为 8 mA，最大为 16 mA。**

您可参考下列 API 函数以进一步了解：

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

***注意：** 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

```
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
}
```

4.12.2 ADC

QEC MDevice 提供多组模拟输入（ADC）信道。默认情况下，`analogRead()` 会将 0 至 3.3 V 的输入电压对应为 0 至 1023 的整数值（10-bit 分辨率）。调用 `analogReadResolution()` 可切换为 11-bit 分辨率，对应为 0 至 2047 的数值。在默认的 10-bit 分辨率下，每一单位约为 3.3 V / 1024 ≈ 3.2 mV。

您可参考下列 API 函数以进一步了解：

- analogRead()
- analogReference()

***注意：** 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

```
int analogPin = 3;     // potentiometer wiper (middle terminal) connected to
analog pin 3
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
}
```

4.12.3 TX/RX

QEC MDevice 支持一组 UART 接口（一对 TX/RX），遵循标准 UART 协议，可通过 86Duino Coding IDE 中的 `Serial1` 对象访问。

您可参考下列 API 函数以进一步了解：

- Serial

*__注意：__ 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

```
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);
    }
}
```

4.12.4 RS-485

QEC MDevice 支持一组 RS-485 接口，遵循标准 RS-485 协议，可通过 86Duino Coding IDE 中的 `Serial485` 或 `SerialCOM` 对象访问。

您可参考下列 API 函数以进一步了解:

- Serial
- Modbus Library

*__注意:__ 请使用 86Duino 引脚对应表中的正确引脚编号。

4.12.4.1 串行通信

用户可通过 `SerialCOM` 设置 QEC MDevice PPC 系列上的串行端口。

示例代码如下:

```
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();
}
```

主要函数:

- `SerialCOM.begin(baud_rate)` : 以指定的波特率初始化通信。
- `SerialCOM.read()` : 由串行端口读取传入的数据。
- `SerialCOM.write(data)` : 通过串行端口送出数据。

4.12.4.2 Modbus RTU 通信

若需使用更高级的通信协议，RS485 接口可搭配 Modbus Library 使用。

86Duino Coding IDE 支持 Modbus 通信协议。Modbus 是 1979 年发布的工业通信标准，用于可编程逻辑控制器（PLC）等自动化电子设备之间的通信。

Modbus 是以 master/slave 为基础的协议，由一个 master 节点与网络上的多个 slave 节点通信，每个节点各有唯一地址。当 master 节点对指定地址送出封包时，只有对应地址的 slave 节点会接收并解析该封包，并依封包内容运行命令与回应。

86Duino 的 Modbus library 具备下列特性：

- 支持 Modbus RTU、TCP 与 ASCII 子协议。
- 可同时作为 Modbus master 与 Modbus slave 节点运行。
- 支持 Modbus gateway 功能。

详细用法请参见 86Duino Modbus Library Documentation。

4.12.4.2.1 示例 1：以 Modbus Master RTU 输出 8-bit 数据

- 将主机端 RS485 的波特率设为 115200。
- 将 Modbus master 设置为 RTU 模式，并连接至 QEC MDevice 上的主机端 RS485。
- 主循环程序：Modbus slave 的 ID 为 30，具 8 个 Coil。先将奇数编号的 Coil 写入 1、偶数编号的 Coil 写入 0；间隔 1 秒后，再将奇数编号的 Coil 写入 0、偶数编号的 Coil 写入 1，如此持续循环。

示例代码如下：

```
#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
}
```

4.12.4.2.2 示例 2：以 Modbus Master RTU 读取 8-bit 数据

- 将主机端 RS485 的波特率设为 115200。
- 将 Modbus master 设置为 RTU 模式，并连接至主机端 RS485。
- 主循环程序：Modbus slave 的 ID 为 30。持续读取第 4 个 Holding Register 的数值，并每隔 1 秒以 `Serial.print` 印出，如此循环。

示例代码如下：

```
#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/read was attempted
    }
}
```

4.12.5 CAN

QEC MDevice 配备两组 CAN Bus 接口，可直接与各式工业设备通信。这两组 CAN Bus 可通过 86Duino Coding IDE 中的 `CAN` 与 `CAN1` 对象访问。
通过 86Duino CANBus Library，用户可用熟悉的 Arduino 风格函数，经由 CAN 对象轻松收送 CAN frame。

86Duino 平台所内含的 CAN bus 具备下列特性：

- 支持 CAN 2.0A 与 2.0B 标准
- 支持最高 1 Mbps 的数据传输速率
- 支持标准（11-bit）与延伸（29-bit）的 data/remote frame 传输
- 提供 3 组 frame buffer
- 完善的错误检测与处理机制

为降低学习与上手的门槛，本 library 的 API 设计高度贴近 Wire library 与 Seeed Studio 的 CAN bus library。以下为本 library 各 API 的功能与用法说明。

您可参考下列 API 函数以进一步了解：

- CANBus Library

***注意：** 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

```
#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 frame.
    CAN.write(buf, 8);                        // Sent 8 bytes of data
    CAN.endTransmission();                    // End transmission
    delay(10);
}
```

4.12.6 SPI

QEC MDevice 配备两组硬件 SPI 接口，可通过 86Duino Coding IDE 中的 `SPI` 与 `SPI1` 对象访问。这些接口可与 ADC、DAC、flash 内存、显示模块等外部周边进行高速全双工通信。

您可参考下列 API 函数以进一步了解：

- SPI library

*__注意：__ 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

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

void setup() {
  // 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 SPI0_CS low to select the chip:
  SPI.SPICS(0);
  delay(100);
  //  send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  delay(100);
  // take SPI0_CS high to de-select the chip:
  SPI.SPICS(1);
}
```

4.12.6.1 SPI Chip Select (SPI0_CS / SPI1_CS)

在 QEC MDevice 上，chip-select 信号线 SPI0_CS 与 SPI1_CS 并非以 `digitalWrite()` 控制。

请改用 SPI (SPI0) 与 SPI1 对象的 `SPICS()` 方法来致能与解除 chip-select 信号：

- `SPICS(0)` — 将 chip-select 信号线拉为 LOW，以选择目标设备。
- `SPICS(1)` — 将 chip-select 信号线拉为 HIGH，以取消选择设备。

***注意：** 请使用 86Duino 引脚对应表中的正确引脚编号。

以下示例驱动两颗数字电位器，分别接在 SPI0 与 SPI1 上，并让各信道的电阻值在最小与最大之间来回循环。

示例代码如下：

```
#include <SPI.h>

void setup() {
  // initialize SPI:
  SPI.begin();
  SPI1.begin();
}

void loop() {
  // go through the six channels of the digital pot:
  for (int channel = 0; channel < 6; channel++) {
    for (int level = 0; level < 255; level++) {      // min -> max
      digitalPotWrite(channel, level);
      delay(10);
    }
    delay(100);
    for (int level = 0; level < 255; level++) {      // max -> min
      digitalPotWrite(channel, 255 - level);
      delay(10);
    }
  }
}

void digitalPotWrite(int address, int value) {
  SPI.SPICS(0);            // take SPI0_CS low to select
  SPI.transfer(address);
  SPI.transfer(value);
  SPI.SPICS(1);            // take SPI0_CS high to de-select

  SPI1.SPICS(0);           // take SPI1_CS low to select
  SPI1.transfer(address);
  SPI1.transfer(value);
  SPI1.SPICS(1);           // take SPI1_CS high to de-select
}
```

***注意：** 请使用 86Duino 引脚对应表中的正确引脚编号。SPI0_CS 位于 J7 Pin 7，SPI1_CS 位于 J8 Pin 7。

4.12.7 I2C

QEC MDevice 通过 Wire 对象提供 I2C（Inter-Integrated Circuit）接口，可与传感器、EEPROM、RTC 模块、I/O 扩展器等设备进行双线通信。此 library 可让您通过 SDA（数据线）与 SCL（时脉线）引脚与 I2C / TWI 设备通信。

您可参考下列 API 函数以进一步了解：

- Wire library

*__注意：__ 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

```
#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);
}
```

4.12.8 MCM

Multi-Channel PWM Module (MCM) 是 QEC MDevice 的重要硬件特性，可高效且弹性地产生多路 **PWM（Pulse Width Modulation）**信号，也是 86Duino Coding IDE 中 `analogWrite()` 函数的基础。

MCM 特别适用于下列应用：

- 马达转速控制
- LED 调光
- 伺服信号产生
- 一般模拟输出仿真

在我们以 1000Hz PWM 频率实现的 `analogWrite()` 函数中，写入分辨率最高可设为 13。将写入分辨率设为 13 后，即可使用 `analogWrite()` 以 0 至 8191 之间的数值设置 PWM 信号而不会溢出。

您可参考下列 API 函数以进一步了解：

- analogWrite()
- analogWriteResolution()

***注意：** 请使用 86Duino 引脚对应表中的正确引脚编号。

示例代码如下：

```
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
  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, 15));
  Serial.print(", 4-bit PWM value : ");
  Serial.println(map(sensorVal, 0, 1023, 0, 15));

  delay(5);
}
```

***注意：** 若您将 `analogWriteResolution()` 设为超过硬件允许范围的数值，我们的实现会*舍弃*多余的位。例如：在 86Duino 上使用 `analogWriteResolution(16)`，传入 `analogWrite()` 的数值仅会采用前 13 个 bit，最后 3 个 bit 会被舍弃。

若您将 `analogWriteResolution()` 设为低于硬件允许范围的数值，不足的位会补零以凑足所需长度。例如：在 86Duino 上使用 `analogWriteResolution(8)`，86Duino 会在 `analogWrite()` 所使用的 8-bit 数值后补上 5 个零位，以取得所需的 13 bit。

Ch. 5

软件功能

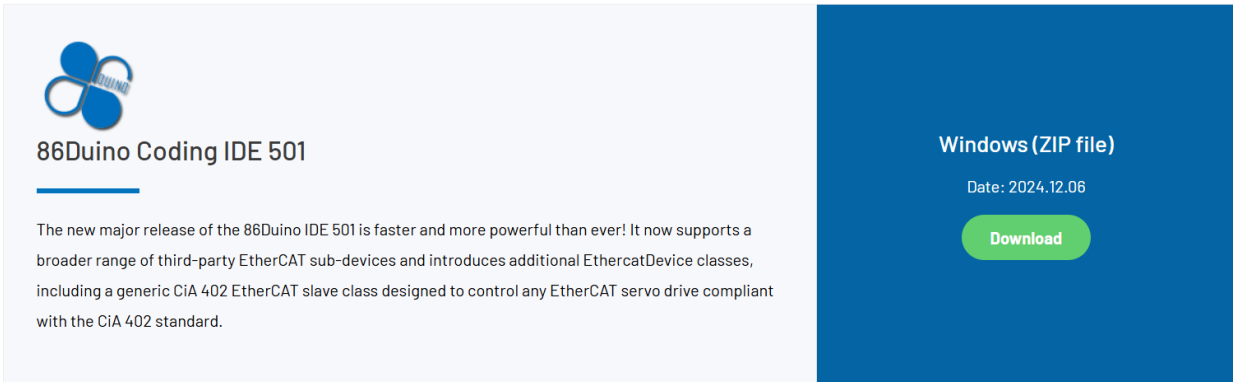
5.1 软件说明

86Duino Coding IDE 501+ 由 QEC 团队开发，专为工业控制系统设计，通过开源的类 Arduino 生态系，将简洁而强大的功能带入工业应用。



86Duino 集成开发环境（IDE）让您能轻松编写代码并上传至 86Duino 开发板，可运行于 Windows、Mac OS X 与 Linux。此开发环境以 Java 编写，并以 Arduino IDE、Processing、DJGPP 等开源软件为基础。

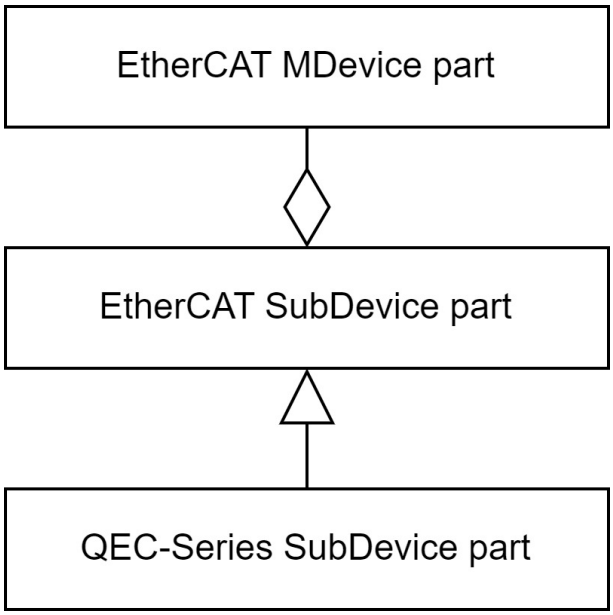
86Duino Coding IDE 501+ 的详细信息请造访 qec.tw。



下载链接：https://www.qec.tw/software/。

5.2 EtherCAT 函数列表

QEC-MDevice 是以 C/C++ 实现的 EtherCAT MDevice library，内含 MDevice、generic SubDevice、CiA 402 SubDevice 等类别，以及专为 QEC 系列 SubDevice 设计的专用类别。这些类别不仅职责划分明确，也已考量日后的扩展性。



这些类别可分为下列三个部分：

- EtherCAT MDevice
  *EtherCAT MDevice 部分*不仅提供多样且弹性的 MDevice 配置与操作函数，也提供多种 EtherCAT SubDevice 操作函数，供 EtherCAT SubDevice 部分调用使用。
- EtherCAT SubDevice
  *EtherCAT SubDevice 部分*提供通用的 EtherCAT SubDevice 类别，可操作 PDO、CoE、FoE 等功能，并包含 CiA 402 SubDevice 的通用类别。
- QEC-Series SubDevice
  *QEC-Series SubDevice 部分*提供 ICOP QEC 系列 SubDevice 的专用函数，让用户能以更友善、更精简的方式编写程序。

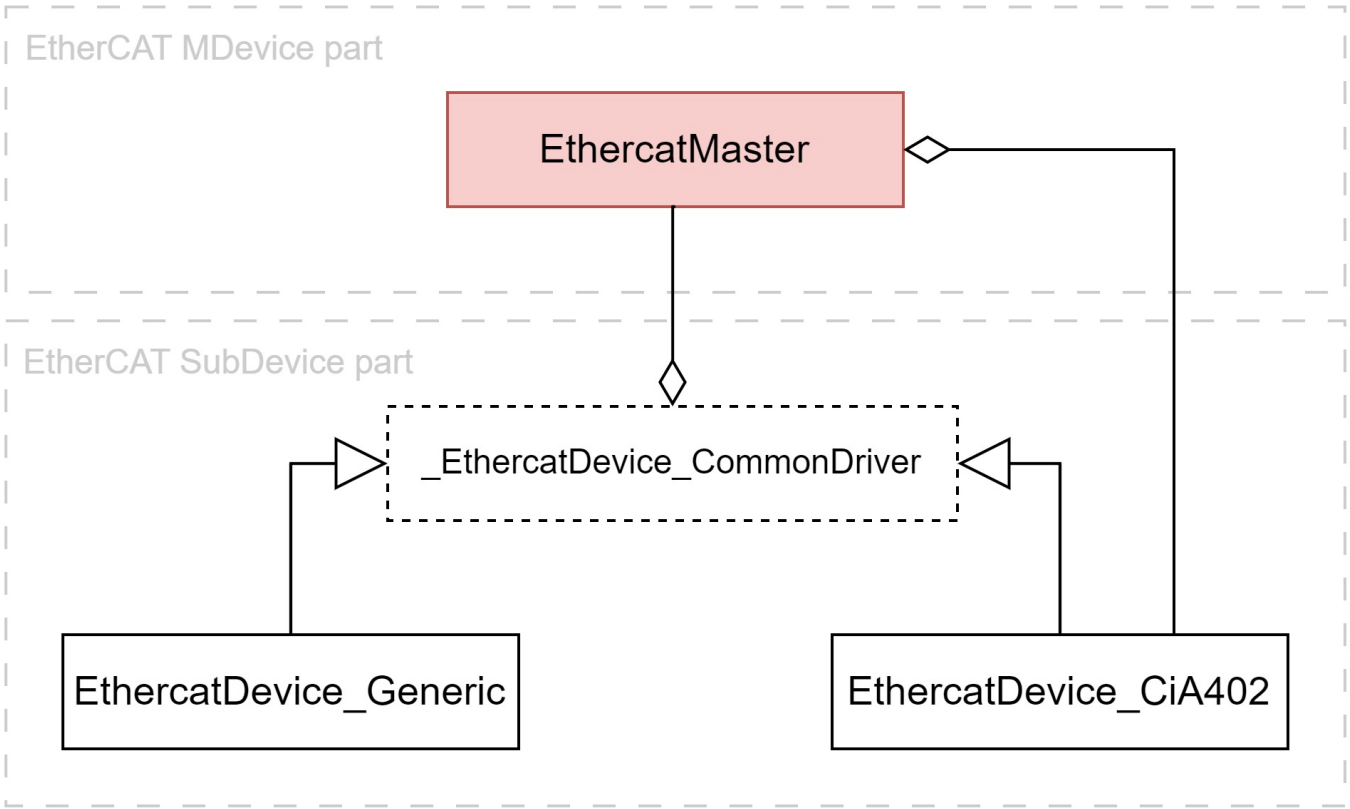
下列清单介绍 QEC MDevice 的 EtherCAT Library API 函数。
API 函数的详细说明请参见 EtherCAT Library API User Manual。

5.2.1 EtherCAT MDevice

EtherCAT MDevice 部分不仅提供多样且弹性的 MDevice 配置与操作函数，也提供多种 EtherCAT SubDevice 操作函数，供 EtherCAT SubDevice 部分调用使用。

EthercatMaster 是 EtherCAT MDevice 部分中唯一的类别，是与 EtherCAT firmware 沟通的关键桥梁。在双系统通信方面，它负责通信接口初始化、周期性交换 process data、处理非周期性传输接口，以及管理中断事件；在 API 方面，它提供 MDevice 初始化、MDevice 控制，以及访问 SubDevice 信息的相关函数。

EtherCAT MDevice 部分与 EtherCAT SubDevice 部分之间的主要类别关系为关联（association），其中 EtherCAT SubDevice 部分相依于 EtherCAT MDevice 部分。EthercatMaster 的类别关系如下图所示：



- EthercatMaster 与 _EthercatDevice_CommonDriver 之间为关联关系，其中 _EthercatDevice_CommonDriver 相依于 EthercatMaster。
- EthercatMaster 与 EthercatDevice_CiA402 之间为关联关系，其中 EthercatMaster 相依于 EthercatDevice_CiA402。

函数:

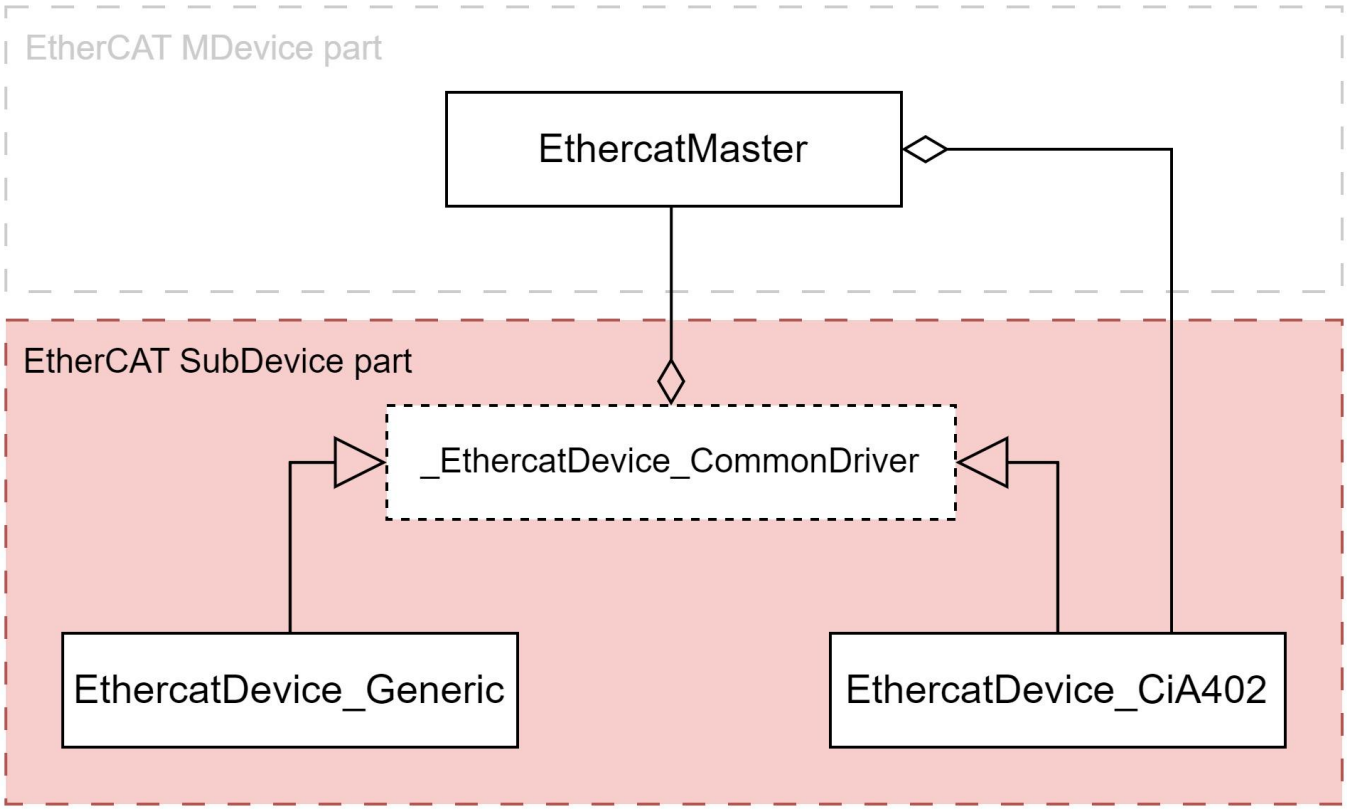
函数名称	说明	可于 Callback 中调用
初始化相关函数		
begin()	初始化 EtherCAT MDevice。	
end()	解除初始化 EtherCAT MDevice。	
isRedundancy()	检查 EtherCAT MDevice 是否已启用 cable redundancy。	O
libraryVersion()	取得 EtherCAT MDevice library 版本。	O
firmwareVersion()	取得 EtherCAT firmware 版本。	O
readSettings()	读取目前的 EtherCAT MDevice 设置。	
saveSettings()	保存 EtherCAT MDevice 设置。	
控制相关函数		
start()	启动 EtherCAT MDevice。	
stop()	停止 EtherCAT MDevice。	
update()	更新 process data 并处理非周期性命令。	O
setShiftTime()	设置 DC-Synchronous 模式的 Global Shift Time。	
getShiftTime()	取得 DC-Synchronous 模式的 Global Shift Time。	O
getSystemTime()	取得目前周期的系统时间。	O
getWorkingCounter()	取得目前周期的 working counter。	O
getExpectedWorkingCounter()	取得预期的 working counter。	O
Callback 相关函数		
attachCyclicCallback()	注册 cyclic callback。	
detachCyclicCallback()	取消注册 cyclic callback。	
attachErrorCallback()	注册 error callback。	
detachErrorCallback()	取消注册 error callback。	
attachEventCallback()	注册 event callback。	
detachEventCallback()	取消注册 event callback。	
errGetCableBrokenLocation1()	于 error callback 中取得断线位置 1。	O[1]
errGetCableBrokenLocation2()	于 error callback 中取得断线位置 2。	O[1]
evtGetMasterState()	于 event callback 中取得 EtherCAT MDevice 状态。	O[2]
SubDevice 信息相关函数		
getSlaveCount()	取得网络上的 SubDevice 数量。	O
getVendorID()	取得指定 SubDevice 的 vendor ID。	O
getProductCode()	取得指定 SubDevice 的 product code。	O
getRevisionNumber()	取得指定 SubDevice 的 revision number。	O
getSerialNumber()	取得指定 SubDevice 的 serial number。	O
getAliasAddress()	取得指定 SubDevice 的 alias address。	O
getSlaveNo()	在网络上寻找符合条件之 EtherCAT SubDevice 的序号。	O

- 注 1：此函数仅能在 error callback 中调用。
- 注 2：此函数仅能在 event callback 中调用。

5.2.2 EtherCAT SubDevice

EtherCAT SubDevice 部分提供通用的 EtherCAT SubDevice 类别，可操作 PDO、CoE、FoE 等功能，并包含 CiA 402 SubDevice 的通用类别。

EtherCAT SubDevice 部分与 EtherCAT MDevice 部分之间的主要类别关系为关联（association），其中 EtherCAT SubDevice 部分相依于 EtherCAT MDevice 部分。如下图所示，_EthercatDevice_CommonDriver 与 EthercatMaster 之间存在关联关系。



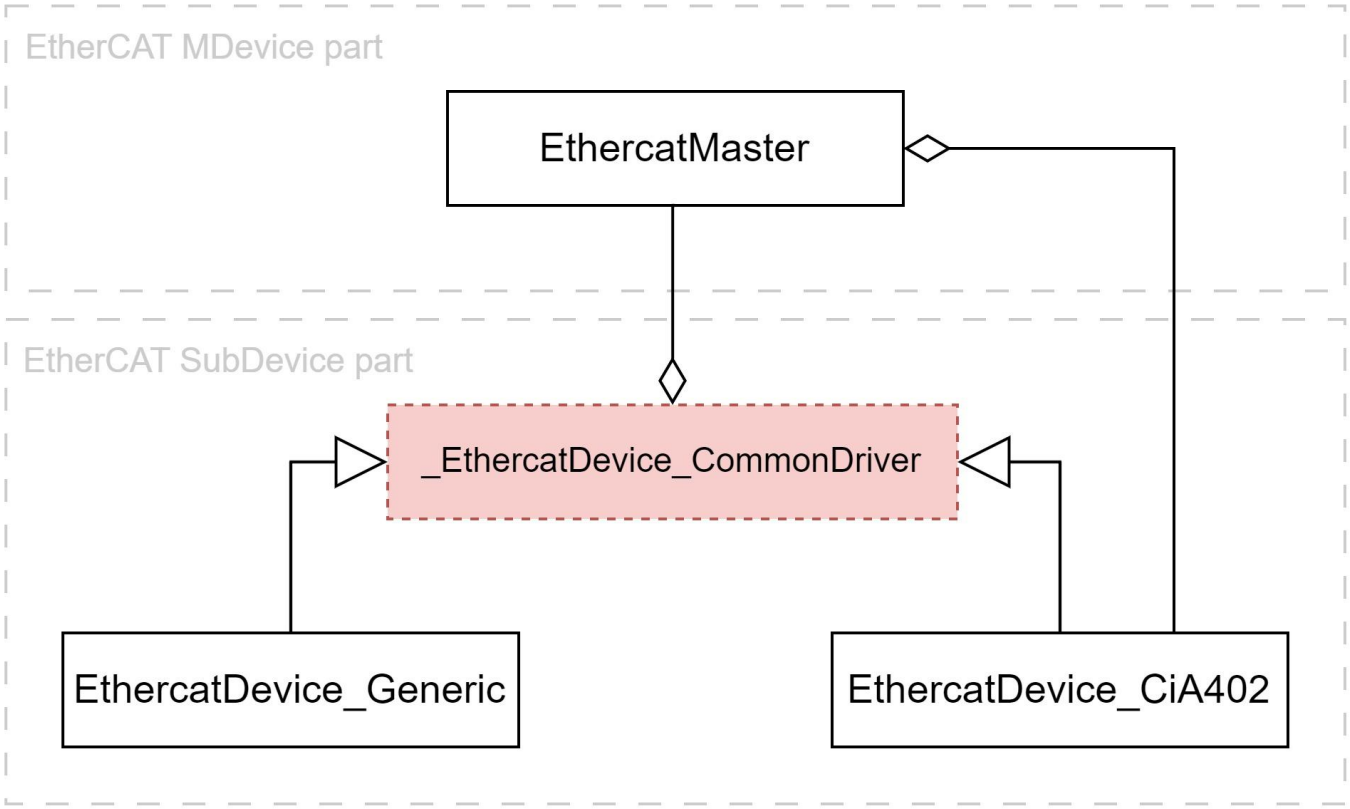
类别：

- _EthercatDevice_CommonDriver
- EthercatDevice_Generic
- EthercatDevice_CiA402

5.2.2.1 _EthercatDevice_CommonDriver

_EthercatDevice_CommonDriver 是一个抽象类别，除了具备访问 SubDevice 信息的函数外，也提供 PDO、SII、CoE、FoE、DC 等多种 EtherCAT 功能的访问方式。所有 EtherCAT SubDevice 类别皆继承自此类别。

_EthercatDevice_CommonDriver 的类别关系如下图所示：



- EthercatMaster 与 _EthercatDevice_CommonDriver 之间为关联关系，其中 _EthercatDevice_CommonDriver 相依于 EthercatMaster。
- 所有其他 EtherCAT SubDevice 类别皆继承自 _EthercatDevice_CommonDriver。

警告： *禁止使用此类别声明对象。*

函数：

函数名称	说明	可于 Callback 中调用
SubDevice 信息相关函数		
getVendorID()	取得 vendor ID。	O
getProductCode()	取得 product code。	O
getRevisionNumber()	取得 revision number。	O
getSerialNumber()	取得 serial number。	O
getAliasAddress()	取得 alias address。	O
getSlaveNo()	取得在 EtherCAT 网络上的序号 ID。	O
getDeviceName()	取得设备名称。	O
getMailboxProtocol()	取得支持的 mailbox 协议类型。	O

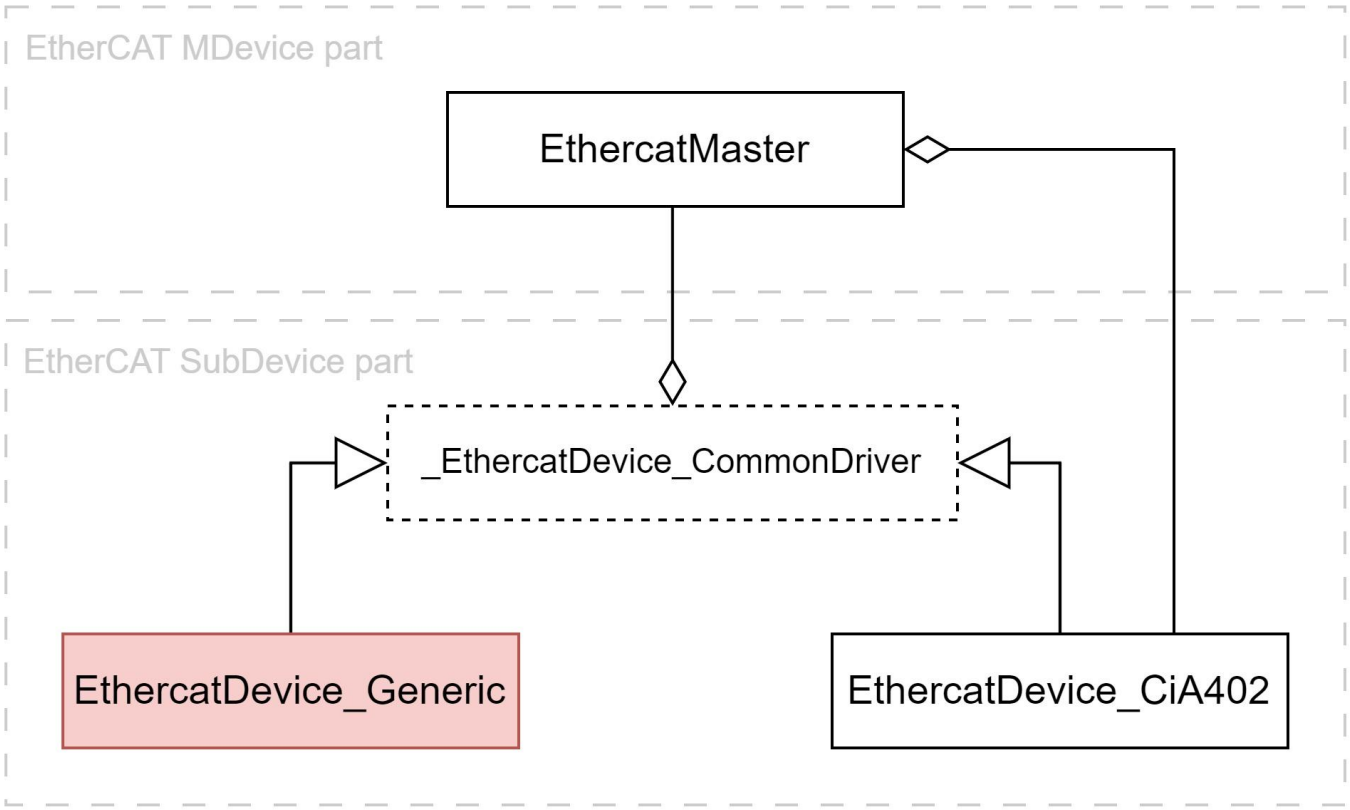
getCoEDetails()	取得 CoE 支持的详细信息。	O
getFoEDetails()	取得 FoE 支持的详细信息。	O
getEoEDetails()	取得 EoE 支持的详细信息。	O
getSoEChannels()	取得支持的 SoE 信道数量。	O
isSupportDC()	检查 EtherCAT SubDevice 是否支持 DC。	O
PDO 访问函数		
pdoBitWrite()	写入 1-bit 输出 process data。	O
pdoBitRead()	读取 1-bit 输入 process data。	O
pdoGetOutputBuffer()	取得输出 process data 的内存指针。	O
pdoGetInputBuffer()	取得输入 process data 的内存指针。	O
pdoWrite()	写入多个 byte 的输出 process data。	O
pdoWrite8()	写入 8-bit 输出 process data。	O
pdoWrite16()	写入 16-bit 输出 process data。	O
pdoWrite32()	写入 32-bit 输出 process data。	O
pdoWrite64()	写入 64-bit 输出 process data。	O
pdoRead()	读取多个 byte 的输入 process data。	O
pdoRead8()	读取 8-bit 输入 process data。	O
pdoRead16()	读取 16-bit 输入 process data。	O
pdoRead32()	读取 32-bit 输入 process data。	O
pdoRead64()	读取 64-bit 输入 process data。	O
CoE 通信函数		
sdoDownload()	将多个 byte 的数据写入对象。	
sdoDownload8()	将 8-bit 数值写入对象。	
sdoDownload16()	将 16-bit 数值写入对象。	
sdoDownload32()	将 32-bit 数值写入对象。	
sdoDownload64()	将 64-bit 数值写入对象。	
sdoUpload()	由对象读取多个 byte 的数据。	
sdoUpload8()	由对象读取 8-bit 数值。	
sdoUpload16()	由对象读取 16-bit 数值。	
sdoUpload32()	由对象读取 32-bit 数值。	
sdoUpload64()	由对象读取 64-bit 数值。	
getODlist()	取得 object dictionary 中既有对象的清单。	
getObjectDescription()	取得对象的 object description。	
getEntryDescription()	取得对象的 entry description。	
FoE 通信函数		
readFoE()	由 EtherCAT SubDevice 读取文件。	
writeFoE()	将文件写入 EtherCAT SubDevice。	
DC 配置函数		
setDc()	设置 DC 参数。	
SII EEPROM 访问函数		
writeSII()	将多个 byte 的数据写入 SII EEPROM。	
writeSII8()	将 8-bit 数值写入 SII EEPROM。	
writeSII16()	将 16-bit 数值写入 SII EEPROM。	

writeSII32()	将 32-bit 数值写入 SII EEPROM。	
readSII()	由 SII EEPROM 读取多个 byte 的数据。	
readSII8()	由 SII EEPROM 读取 8-bit 数值。	
readSII16()	由 SII EEPROM 读取 16-bit 数值。	
readSII32()	由 SII EEPROM 读取 32-bit 数值。	
初始化相关函数		
attach()	初始化此 EtherCAT SubDevice 类别的对象。	
detach()	解除初始化此 EtherCAT SubDevice 类别的对象。	

5.2.2.2 EthercatDevice_Generic

EthercatDevice_Generic 是通用的 EtherCAT SubDevice 类别，可用于控制所有 EtherCAT SubDevice，包括访问 SubDevice 信息、PDO、CoE、FoE、DC 等功能。

EthercatDevice_Generic 的类别关系如下图所示：



- EthercatDevice_Generic 继承自 _EthercatDevice_CommonDriver。

父类别

- _EthercatDevice_CommonDriver

函数

函数名称	说明	可于 Callback 中调用
初始化相关函数		
attach()	初始化此 EtherCAT SubDevice 类别的对象。	
detach()	解除初始化此 EtherCAT SubDevice 类别的对象。	

5.2.2.3 EthercatDevice_CiA402

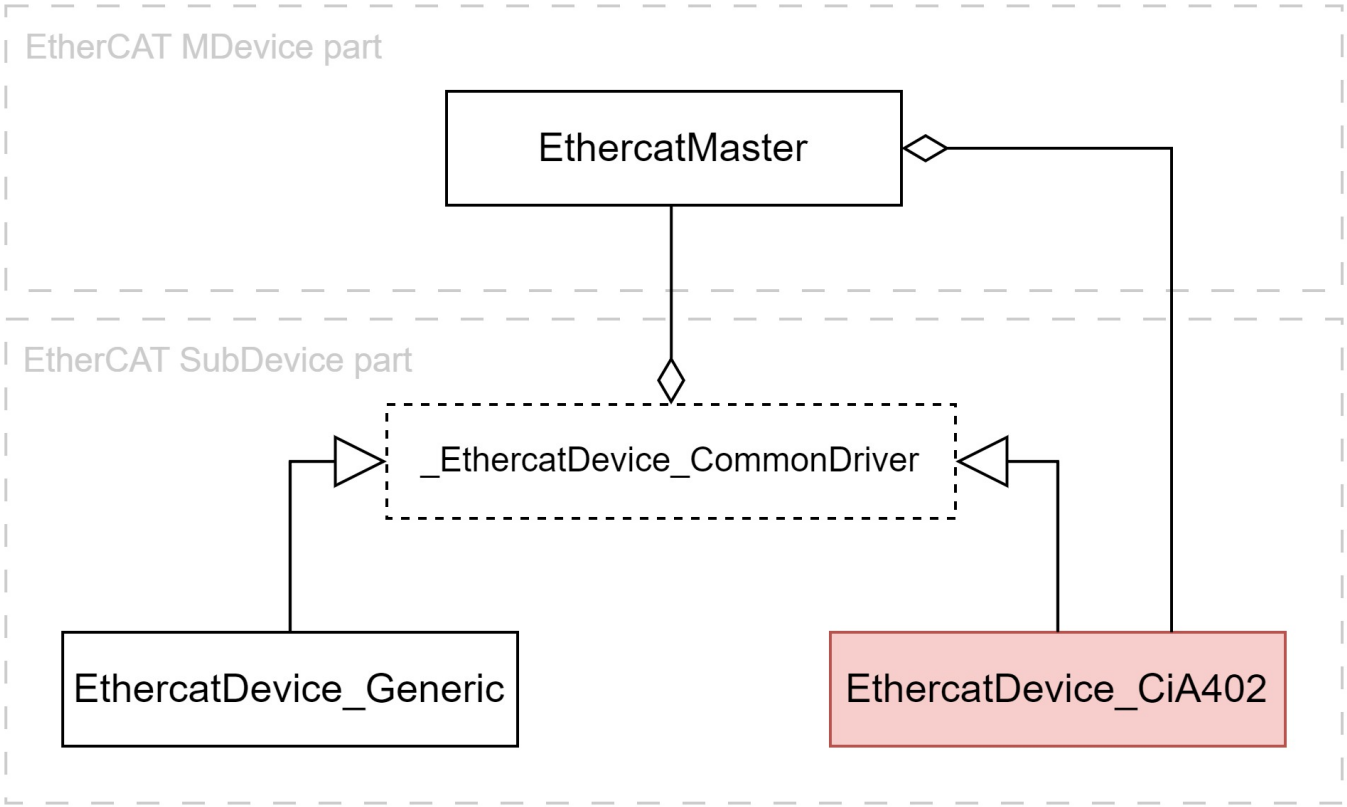
EthercatDevice_CiA402 是通用的 CiA 402 EtherCAT SubDevice 类别，可用于控制任何支持 CiA 402 标准的 EtherCAT 伺服驱动器。它提供常用 CiA 402 对象的访问函数，以及数种 CiA 402 操作模式与功能群组的操作函数，包括：

- **操作模式**
  - Profile Position (pp)
  - Profile Velocity (pv)
  - Profile Torque (tq)
  - Homing (hm)
  - Cyclic Synchronous Position (csp)
  - Cyclic Synchronous Velocity (csv)
  - Cyclic Synchronous Torque (cst)
- **功能群组**
  - Touch Probe

关于 CiA 402 的详细信息，请参见下列文档：

- CiA Draft Standard 402: CANopen device profile drives and motion control
- ETG.6010 Implementation Directive for CiA402 Drive Profile
- 目前所使用之 CiA 402 驱动器设备的使用手册

EthercatDevice_CiA402 的类别关系如下图所示：



- *EthercatDevice_CiA402* 继承自 *_EthercatDevice_CommonDriver*。

父类别

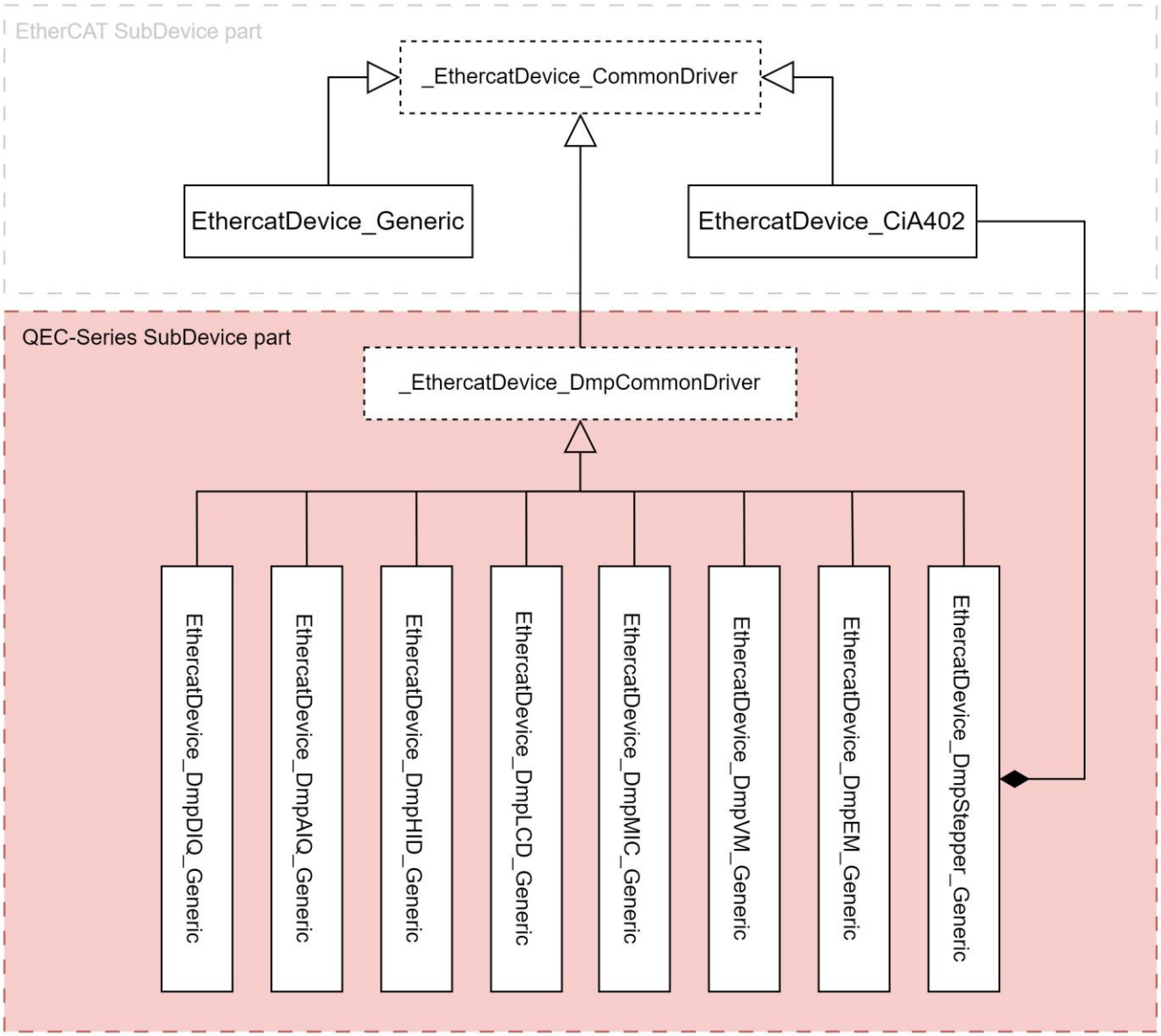
- _EthercatDevice_CommonDriver

关于 CiA 402 API 的详细信息，请参见 EtherCAT Library API User Manual - QEC。

5.2.3 QEC 系列 SubDevice

QEC-Series SubDevice 部分提供 ICOP QEC 系列 SubDevice 的专用函数，让用户能以更友善、更精简的方式编写程序。

QEC-Series SubDevice 部分与 EtherCAT SubDevice 部分之间的主要类别关系为关联（association），其中 QEC-Series SubDevice 部分相依于 EtherCAT SubDevice 部分。如下图所示，_EthercatDevice_DmpCommonDriver 与 _EthercatDevice_CommonDriver 之间存在关联关系。



类别

- _EthercatDevice_DmpCommonDriver
- EthercatDevice_DmpDIQ_Generic
- EthercatDevice_DmpAIQ_Generic
- EthercatDevice_DmpHID_Generic
- EthercatDevice_DmpLCD_Generic
- EthercatDevice_DmpStepper_Generic

5.3 更多资源

若您想进一步了解 86Duino IDE 提供的函数库，或深入研究 86Duino 编程语言的细节，请参见下列链接：

- **EtherCAT Library API User Manual：** QEC MDevice 是兼容于 86Duino Coding IDE 501+ 的 EtherCAT MDevice，可在 EtherCAT MDevice 与 EtherCAT SubDevice 之间提供实时通信。详细信息请参见 https://www.qec.tw/ethercat/api/ethercat-library-api-user-manual/。
- **86Duino IDE Libraries：** 可于 https://www.qec.tw/86duino/libraries/ 查阅 86Duino IDE 支持的完整函数库清单，以及详细文档与示例。
- **86Duino Language Reference：** 可于官方的 86Duino Language Reference（https://www.qec.tw/86duino/86duino-language-reference/）了解 86Duino 编程语言的语法、函数与使用方式。

无论是初学者或资深开发者，这些资源都能为使用 86Duino 平台提供实用的参考。善用这些链接，您将能充分发挥 86Duino IDE 的潜力，打造具创新性的项目。

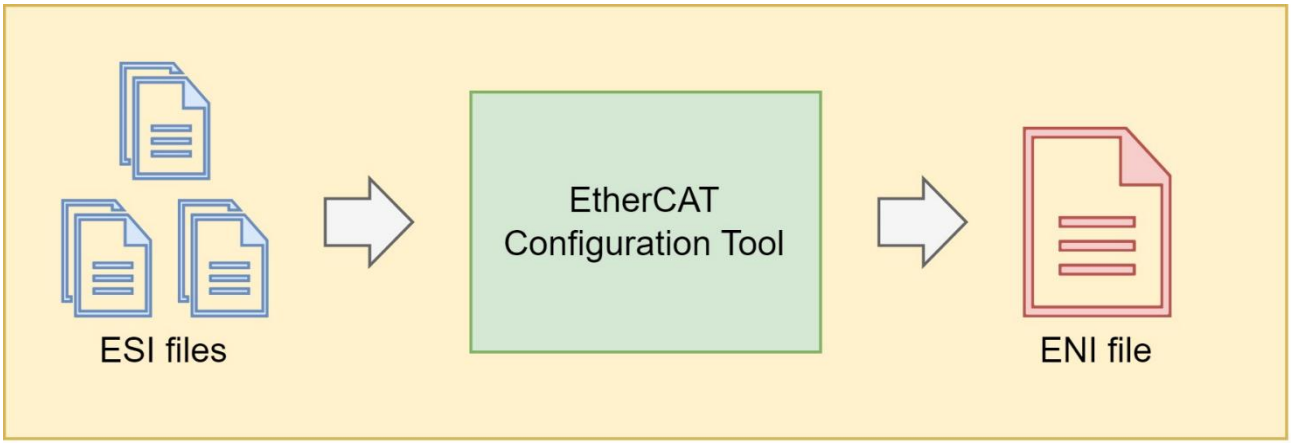
祝您在 86Duino IDE 上开发愉快！

86Duino reference 的文本内容修改自 Arduino reference，并依 Creative Commons Attribution-ShareAlike 3.0 License 授权。reference 中的代码示例则发布至公共领域。

附录

A1. 关于 86Duino IDE 的 ENI 配置

EtherCAT Network Information (ENI) 包含设置 EtherCAT 网络所需的必要信息。这份以 XML 为基础的文件，记录了 EtherCAT MDevice 的一般信息，以及所有连接至该 EtherCAT MDevice 之 SubDevice 的配置。EtherCAT 配置工具会读取 ESI 文件，或对网络进行在线扫描以取得所有 SubDevice，接着用户可设置 PDO mapping、启用 DC 等相关 EtherCAT 设置，并导出 ENI 文件。



EtherCAT Technology Group 规定，EtherCAT MDevice 软件在 Network Configuration 项目中至少须支持下列其中一种方式：在线扫描（Online Scanning）或读取 ENI（Reading ENI）。本 library 两者皆支持。不过在读取 ENI 时，目前仅会由 ENI 文件中截取部分信息用于网络配置。

截取的信息包括下列项目：

EtherCATConfig : Config : SubDevice : Info

- 元素
  - VendorId
  - ProductCode
- 属性
  - Identification : Value
- 用途

  用于检查网络上的 EtherCAT SubDevice 是否与 ENI 文件所指定的 SubDevice 相符，检查规则如下：
  - 检查 ENI 文件中的 SubDevice 数量是否与网络上的 SubDevice 数量相符。
  - 对于 ENI 文件中具有 Identification: Value 属性的 SubDevice，检查网络上是否存在 Alias Address 与 Identification: Value 属性相符，且 Vendor ID 与 Product Code 亦相符的 SubDevice。若存在，即表示比对成功。
  - 对于具有 Identification: Value 属性但比对失败的 SubDevice，或未具备此属性的 SubDevice，则检查网络上相同序号之 SubDevice 的 Vendor ID 与 Product Code 是否相符。

EtherCATConfig : Config : SubDevice : Mailbox

- 元素
  - Send : MailboxSendInfoType : Start
  - Recv : MailboxRecvInfoType : Start
- 用途

  用于设置 EtherCAT SubDevice 的 mailbox Physical Start Address。

EtherCATConfig : Config : SubDevice : Mailbox : CoE

- 元素
  - InitCmds : InitCmd : Index
  - InitCmds : InitCmd : SubIndex
  - InitCmds : InitCmd : Data
  - InitCmds : InitCmd : Timeout
- 属性
  - InitCmds : InitCmd : CompleteAccess
- 用途

  在 EtherCAT state machine 切换至 Pre-Operational 状态后，于 `EthercatMaster::begin()` 中运行 EtherCAT SubDevice 的 CoE 初始化命令。

EtherCATConfig : Config : SubDevice : ProcessData

- 元素
  - Recv : BitLength
  - Send : BitLength
- 用途

  提供 EtherCAT SubDevice 输出与输入 process data 的 bit 长度给 firmware，作为相关配置之用。

EtherCATConfig : Config : SubDevice : ProcessData : Sm

- 元素
  - SyncManagerSettings : StartAddress
  - SyncManagerSettings : ControlByte
  - SyncManagerSettings : Enable
- 用途

  用于设置 EtherCAT SubDevice 之 process data 的 Sync Manager 寄存器。

EtherCATConfig : Config : SubDevice : DC

- 元素
  - CycleTime0
  - CycleTime1
  - ShiftTime
- 用途

  用于设置 EtherCAT SubDevice 的 DC 参数。

保修

本产品自购买日起保修一年，保证于此期间内维持良好运行状态。若本产品于保修期间内发生功能异常，本公司将依下列条款，选择免费更换或维修。本保修不适用于因不当使用、擅自改装、意外或天灾所造成之损坏。对于因使用、误用本产品或无法使用本产品所导致的任何损害、利润损失、储蓄损失，或其他附带性、衍生性损害，供应商概不负责；供应商亦不对任何其他相关方所提出之请求负责。退回商品前，必须先取得供应商的退货授权；您可致电或传真供应商，申请退货授权编号（RMA number）。退回商品时，请务必附上清楚的问题描述。

本手册中出现的所有商标，均为其各自所有人之注册商标。所有规格如有变更，恕不另行通知。
©ICOP Technology Inc. 2026