

ECM-XFU-SK USER GUIDE

版本 : V.1.1.2

日期 : 2023.11

目錄

第 1 章 硬體腳位及使用說明	2
1.1 ECM-XFU-SK V1.3 腳位說明	3
1.1.1 P8/P9 腳位說明	3
1.1.2 J1/J2腳位說明	7
1.1.3 J3腳位說明	7
1.1.4 J7腳位說明(ADC/DAC)	8
1.1.5 J8腳位說明(GPIO)	8
1.1.6 J9腳位說明(ENC)	8
1.1.7 J29腳位說明 (Reset)	9
1.1.8 J33腳位說明(SPI)	9
1.2 ECM-XFU-SK 燈號顯示	10
1.3 ECM-XFU-SK V1.3 尺寸圖	10
第 2 章 MCU透過SPI介面範例	12
2.1 STM32範例列表	12
2.2 STM32範例環境建立導引(NUCLEO-F401RE)	12
2.3 實際接線範例	27
2.4 Nuvoton範例列表	29
2.5 Nuvoton新唐範例環境建立導引	30
2.6 Nuvoton修改範例程式為實際應用導引	32
2.7 常見錯誤說明	36
2.8 402自動切換與手動切換方法	36
第 3 章 Beaglebone透過SPI介面範例說明	38
3.1 設置Beaglebone系統	38
3.2 測試範例	39
第 4 章 Windows Visual Stdio設置導引	42
4.1 Visual Stdio C++	42
第 5 章 ECM-XFU ITE整合測試環境說明	45
5.1 ITE簡介	45
5.2 一般操作說明	45
5.3 SDO操作說明	50
5.4 透過Load ENI修改RxPDO及TxPDO配置說明	52
5.5 透過SDO修改RxPDO及TxPDO配置說明	54
5.6 更新韌體 (Update firmware)	55
5.7 SII editor	56
5.8 XFU週邊IO (Peripheral)	57
5.9 評估版 與 授權版	57
5.10 代碼產生工具(Code Generator)	57

第 1 章 硬體腳位及使用說明

Digital Input and Output

- 16 non-isolated input/output channels
- Input Voltage: 0V / 3.3V
- Output Voltage: 0V / 3.3V

Serial Peripheral Interface(SPI)

- One sets of SPI controllers
- Voltage level: 0V / 3.3V

EtherCAT

- Data transfer medium: Ethernet cable (CAT5e), shield type: S/STP or S/UTP
- Ethernet interface: 1x RJ-45
- Data transfer rate: 100Mbps
- Protocol: EtherCAT

USB Interface

- USB 2.0 with on-chip transceiver
- Implements USB HID class

Environment

- Operating temperature: 0°C to 65°C

1.1 ECM-XFU-SK V1.3 腳位說明

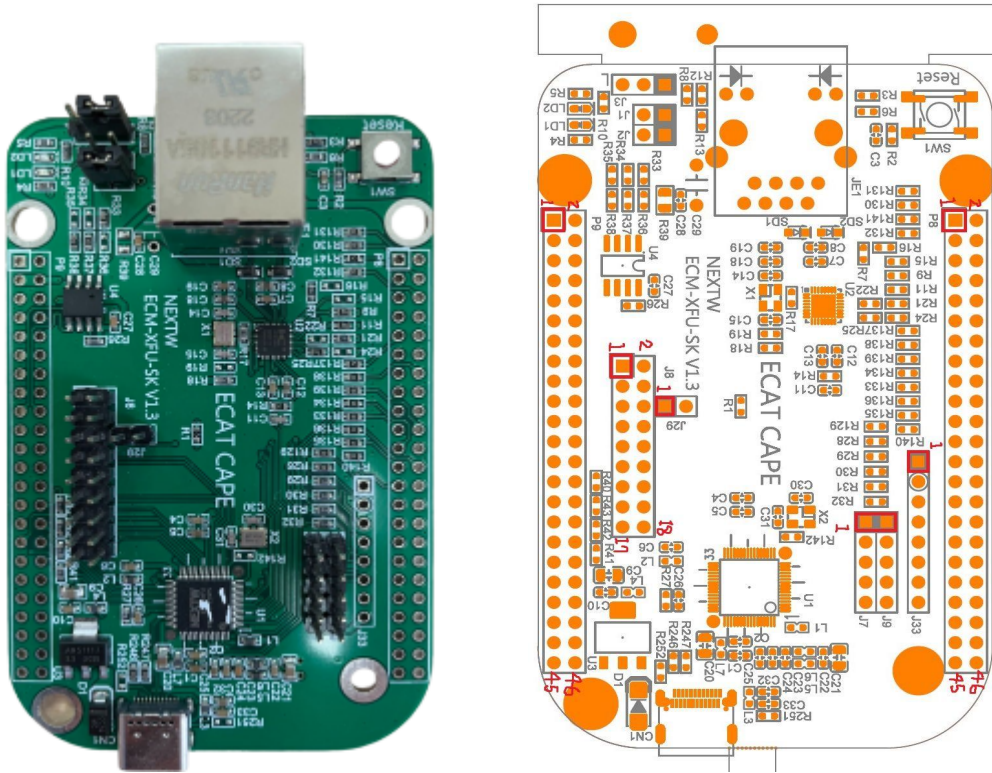
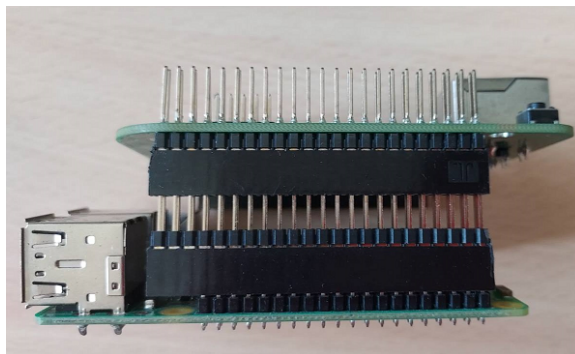
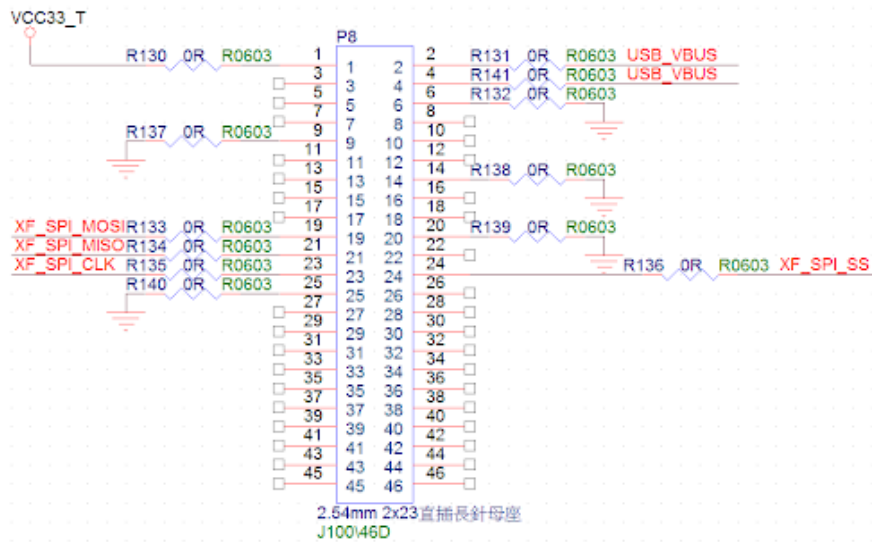


圖 1-1

1.1.1 P8/P9 腳位說明

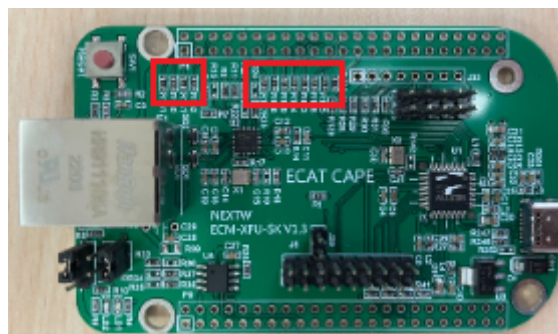
ECM-XFU-SK與Rasperberry Pi對接，需依照下圖將P8 / P9 上件，即可直接與Rasperberry PI的J8對接，PIN1對接PIN1



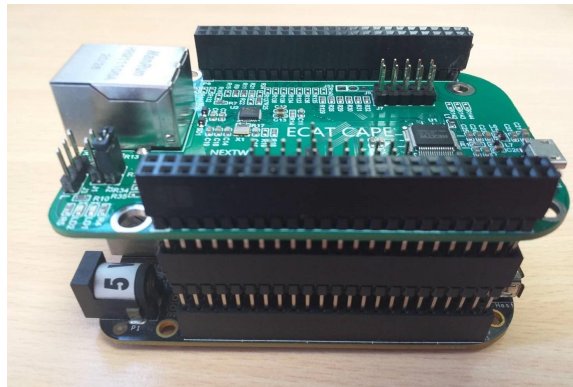


ECM-XFU-SK預設R130~R141均有上件，Raspberry PI透過P8的相關腳位與ECM-XFU進行SPI的傳輸。

ECM-XFU-SK與Beaglebone對接，須先把靠近P8的電阻R131、R130、R141、R132、R137、R138、R139、R134、R133、R136、R135、R140電阻移除後(紅色方框)，避免影響BeagleBone本身的pin腳



再將P8/P9上件後，將 ECM-XFU-SK PIN1對接Beaglebone PIN1即可，P8/P9與Beaglebone的P8/P9對接即可。



注意P8/P9排針上件方向(上下)在對接Raspberry Pi與對接BeagleBone時不同, 恰相反

ECM-XFU-SK V1.3 P9腳位定義如下表, - 代表空腳:

P9			
PIN45	-	PIN46	-
PIN43	-	PIN44	-
PIN41	-	PIN42	-
PIN39	-	PIN40	-
PIN37	-	PIN38	-
PIN35	-	PIN36	-
PIN33	-	PIN34	-
PIN31	XF_SPI_CLK	PIN32	-
PIN29	XF_SPI_MOSI	PIN30	XF_SPI_MISO
PIN27	-	PIN28	XF_SPI_SS
PIN25	-	PIN26	-
PIN23	-	PIN24	-
PIN21	-	PIN22	-
PIN19	-	PIN20	-

PIN17	AM335X_I2C0_SCL	PIN18	AM335X_I2C0_SDA
PIN15	-	PIN16	-
PIN13	-	PIN14	-
PIN11	-	PIN12	-
PIN9	-	PIN10	-
PIN7	SYS_5V	PIN8	SYS_5V
PIN5	-	PIN6	-
PIN3	VDD_3V3B	PIN4	VDD_3V3B
PIN1	GND	PIN2	GND

ECM-XFU-SK V1.3的P8腳位定義如下表，可直接與Raspberry pi對接，
- 代表空腳：

P8(V1.3)			
PIN45	-	PIN46	-
PIN43	-	PIN44	-
PIN41	-	PIN42	-
PIN39	-	PIN40	-
PIN37	-	PIN38	-
PIN35	-	PIN36	-
PIN33	-	PIN34	-
PIN31	-	PIN32	-
PIN29	-	PIN30	-
PIN27	-	PIN28	-
PIN25	GND	PIN26	-
PIN23	XF_SPI_CLK	PIN24	XF_SPI_SS
PIN21	XF_SPI_MISO	PIN22	-
PIN19	XF_SPI_MOSI	PIN20	GND

PIN17	-	PIN18	-
PIN15	-	PIN16	-
PIN13	-	PIN14	GND
PIN11	-	PIN12	-
PIN9	GND	PIN10	-
PIN7	-	PIN8	-
PIN5	-	PIN6	GND
PIN3	-	PIN4	(USB_VBUS)
PIN1	VCC33	PIN2	(USB_VBUS)

注意：若要與Beaglebone對接，須先把靠近P8的電阻R131、R130、R141、R132、R137、R138、R139、R134、R133、R136、R135、R140電阻移除後，方能對接。

1.1.2 J1/J2腳位說明

*若要由ECM-XFU-SK的USB供電給其它外接板 3.3V，J1的PIN1與PIN2請用跳線帽對接短路。

*透過P8/P9對接外接板Beaglebone時，若要由外接板Beaglebone的SYS_5V供電給ECM-XFU-SK，J2的PIN1與PIN2請用跳線帽對接短路。

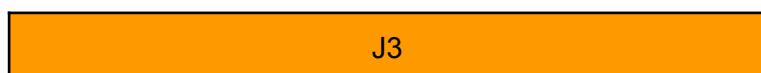
J1	PIN2	VDD_3V3B	PIN1	VCC33
J2	PIN2	SYS_5V	PIN1	USB_VBUS

1.1.3 J3腳位說明 (韌體更新)

一般使用模式: J3 PIN2空接或使用跳線帽連接J3 PIN1與J3 PIN2

韌體更新模式: 使用跳線帽連接J3 PIN2與J3 PIN3

* 僅於初始上電(或Reset)時依據J3的狀態進入指定的模式



PIN1	VCC33(Pull-up resistor 4.7K in series)
PIN2	CONFIG/LED
PIN3	GND(Pull-down resistor 4.7K in series)

1.1.4 J7腳位說明(ADC/DAC)

J7	
PIN1	5V
PIN2	ADC_IN
PIN3	DAC_OUT
PIN4	DAC_ST
PIN5	GND

1.1.5 J8腳位說明(GPIO)

ECM-XFU-SK V1.3 J8			
PIN1	VCC33	PIN2	GND
PIN3	GPIO14	PIN4	GPIO15
PIN5	GPIO12	PIN6	GPIO13
PIN7	GPIO10	PIN8	GPIO11
PIN9	GPIO8	PIN10	GPIO9
PIN11	GPIO6	PIN12	GPIO7
PIN13	GPIO4	PIN14	GPIO5
PIN15	GPIO2	PIN16	GPIO3
PIN17	GPIO0	PIN18	GPIO1

1.1.6 J9腳位說明(ENC)

J9	
PIN1	5V
PIN2	ENCI
PIN3	ENCA
PIN4	ENCB
PIN5	GND

1.1.7 J29腳位說明 (Reset)

J29可外接Button, 當Button按下, 即RESETn與GND導通, 系統會RESET

J29	PIN2	GND	PIN1	RESETn
-----	------	-----	------	--------

1.1.8 J33腳位說明(SPI)

若要使用ST板(ex:NUCLEO-F401RE)可直接從PIN1開始接
D8,D9,D10,D11,D12,D13,GND,AVDD

J33	
PIN1	INT1
PIN2	INT0
PIN3	XF_SPI_SS
PIN4	XF_SPI_MOSI
PIN5	XF_SPI_MISO
PIN6	XF_SPI_CLK
PIN7	GND
PIN8	VCC33

1.2 ECM-XFU-SK 燈號顯示

LD1	綠燈	亮	尚未有命令 或 CRC錯誤	滅	CRC正確
LD2	紅燈	亮	電源正常	滅	無電源
網口	橘燈	恆亮	網路連線 速度100MHz	滅	無連線
網口	綠燈	亮	連線但無傳輸	閃爍	資料傳輸

1.3 ECM-XFU-SK V1.3 尺寸圖

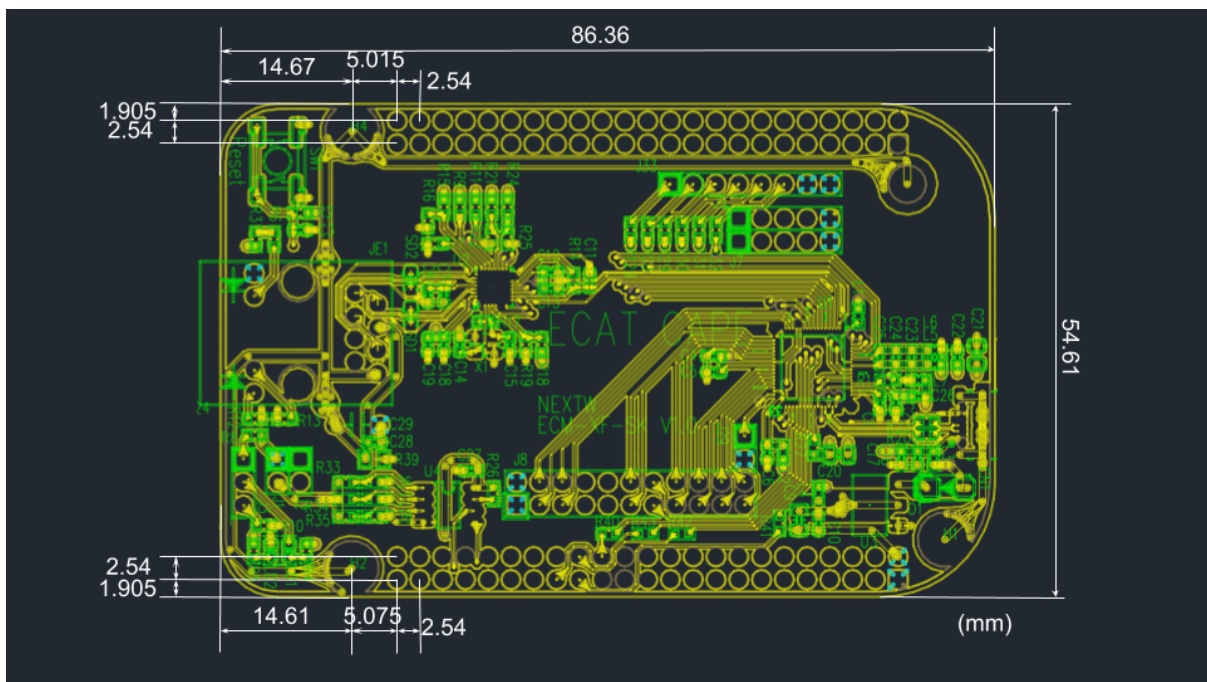


圖1-3

第 2 章 MCU透過SPI介面控制ECM-XF範例

2.1 範例架構與主要變數介紹

ECM資料夾存放與ECM-XF相關的程式，其他資料夾則存放與MCU硬體相關的程式。ECM資料夾下ecmxfudrv為應用層API轉換為ECM命令，資料夾includes為相關API的標頭檔(.h File)，資料夾platform為與硬體相關的API，如SPI傳輸、延遲Delay等，example資料夾則為各種不同應用的主程式檔。

以下介紹主要檔案內容

- ECM\example\ex_home\main.c 驅動器從站回零點(Homing)範例
- ECM\example\ex_csp\main.c 驅動器從站週期位置命令範例
- ECM\example\ex_pp\main.c 驅動器從站目標位置令範例
- ECM\example\ex_pt\main.c 驅動器從站目標扭力令範例
- ECM\example\ex_pv\main.c 驅動器從站目標速度令範例
- ECM\example\ex_rta\main.c Real Time Application, 設定條件比對特定TxPDO內容值，若滿足條件則更改特定RxPDO內容值範例
- ECM\example\utility\utility.c 各範例共用之工具程式
- ECM\example\ex_XXXX\PdoDefine.h 定義EtherCAT Slave週期性資料傳輸內容(RxPDO、TxPDO)

以下介紹進入OP模式前的設定流程

1. ECM_InitLibrary: 設定SPI Data Size與CRC Type, 回傳IC內版本號
=> 必要命令, 若未通過(返回值為0), 代表硬體線路未接受
2. ECM_EcatInit: 初始化從站, 設定週期時間與分散時鐘
=> 必要命令, 必須連結從站且網路孔燈亮起後才能執行, 注意, 設定值必須是從站可以支援的設定值
3. ECM_EcatSlvCntGet: 取得連線從站數量
=> 非必要命令, 若返回的從站數量與實際不同, 請檢查各站網路接線, 或待全部的網口Link燈亮起, 再進行ECM_EcatInit.
4. ConfigDrive: 配置從站的RxPDO及TxPDO (定義於Utility)
=> 視需求及從站特性而定。並非所有從站均能由主站配置RxPDO與TxPDO, 請先參閱從站說明。
5. ECM_SetTxFIFOCnt / ECM_SetRxFIFOCnt: 設定RxPDOFIFO/TxPDOFIFO數量
=> 非必要命令, RxPDOFIFO/TxPDOFIFO數量預設均為64筆, 可透過此命令變更數量
6. ECM_CheckMEMSpace: 檢查設定值記憶體空間是否足夠 (定義於Utility)
=> 非必要但建議命令, 檢查1) SPI Data Size是否則足夠放下所有RxPDO / TxPDO. 2) 記憶體空間是否滿足TxFIFO及RxFIFO的空間需求
7. ECM_InitFIFO: 初始化FIFO

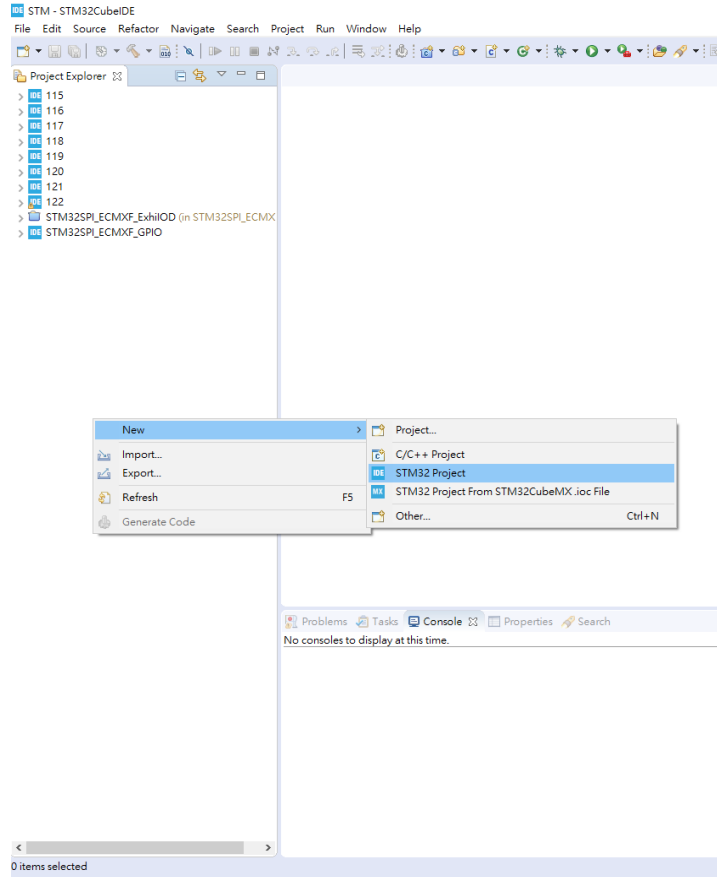
- => 必要命令, 初始化FIFO前請確認各設定值沒有超過記憶體容量
8. ECM_EnableFIFO: 開啟或關閉FIFO
=> 非必要命令, 初始化FIFO後預設開啟FIFO功能, 當進入OP時就會開始透過FIFO收發資料, 可透過此命令關閉FIFO, 待進入有需求時再開啟
 9. ECM_FifoRxPdoSizeGet/ECM_FifoTxPdoSizeGet: 取得RxPdoSize / TxPdoSize
=> 非必要但建議命令, 取得所有從站的RxPDO / TxPDO大小, 可比對是否與程式中傳送/接收的資料結構大小相符
 10. SetDriverCmdOpMode: 設定驅動器的操作模式 (定義於Utility)
=> 驅動器專用命令, 非必要命令但建議, 設定驅動器的操作模式(0x6060), 驅動器所支援的操作模式請參考驅動器手冊說明
 11. SetStateAndCheck: 切換etherCAT模式 (Init / perOP / safeOP / OP) (定義於Utility)
=> 切換etherCAT模式有特定的順序, 請參閱etherCAT技術手冊
 12. ECM_CheckDCStable: 檢查EtherCAT的分散時鐘(DC)是否已經穩定
=> 非必要命令, 當etherCAT模式由PerOP切換至SafeOP時, 若有啟用分散時鐘(DC)機制, 則分散時間會開始校正同步時間, 透過此命令可確認分散時鐘校正是否已經完成

以下介紹進入OP模式後的主要命令

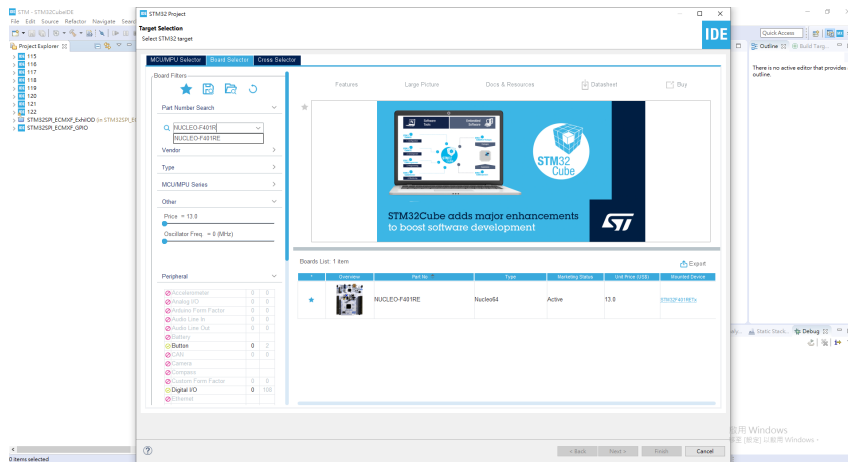
1. AlignmentAndWaitServoOn: 將目標位置指定為目前位置, 並激磁 (定義於Utility)
=> 非必要命令, 當操作模式為CSP時, 需先將目標位置指定為目前位置, 再激磁
2. ECM_EcatPdoFifoDataExchangeAdv: 透過FIFO進行PDO的交換
=> 必要命令, 進入OP後, 透過此命令交換FIFO中的PDO資料

2.2 STM32範例環境建立導引(NUCLEO-F401RE)

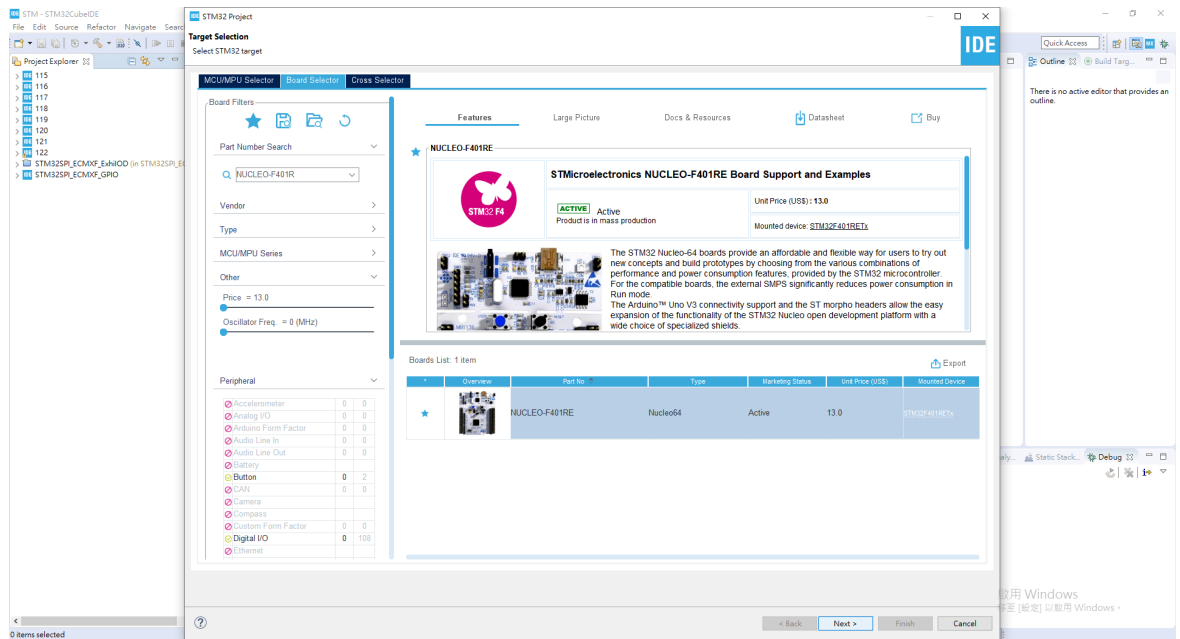
1. 開啟STM32Cube IDE.
2. 在左側視窗點擊滑鼠右鍵並選擇”STM32 Project”



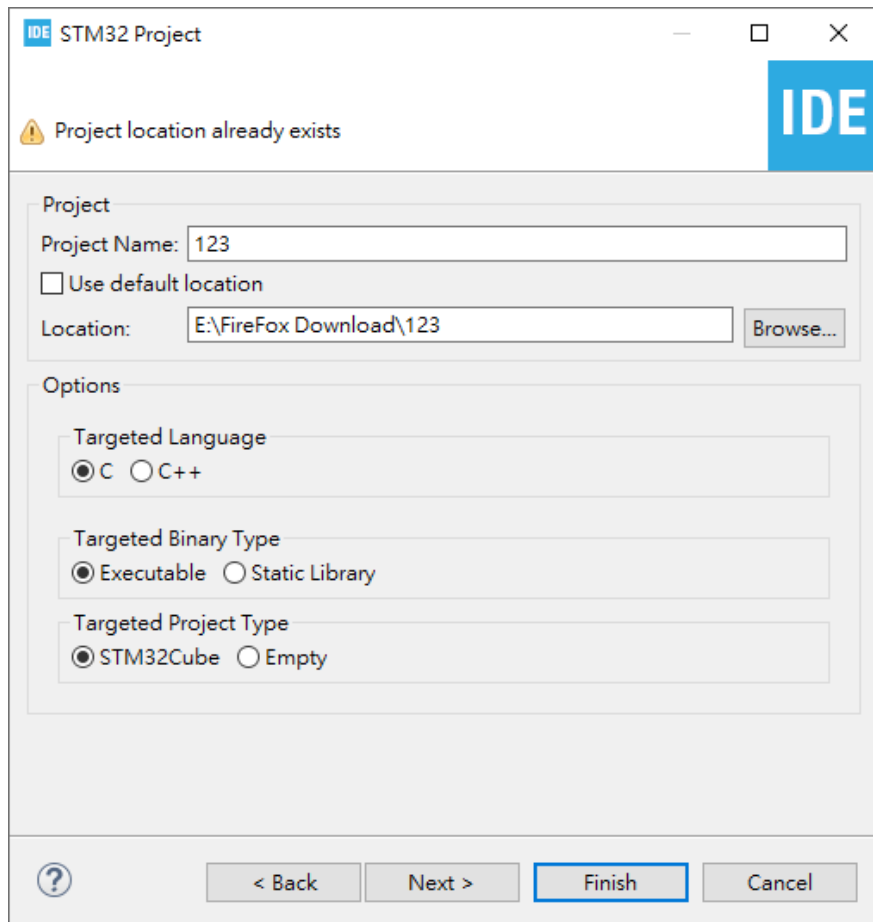
3. 選擇所使用的板子。這邊例子以STM32 NUCLEO-F401RE為例



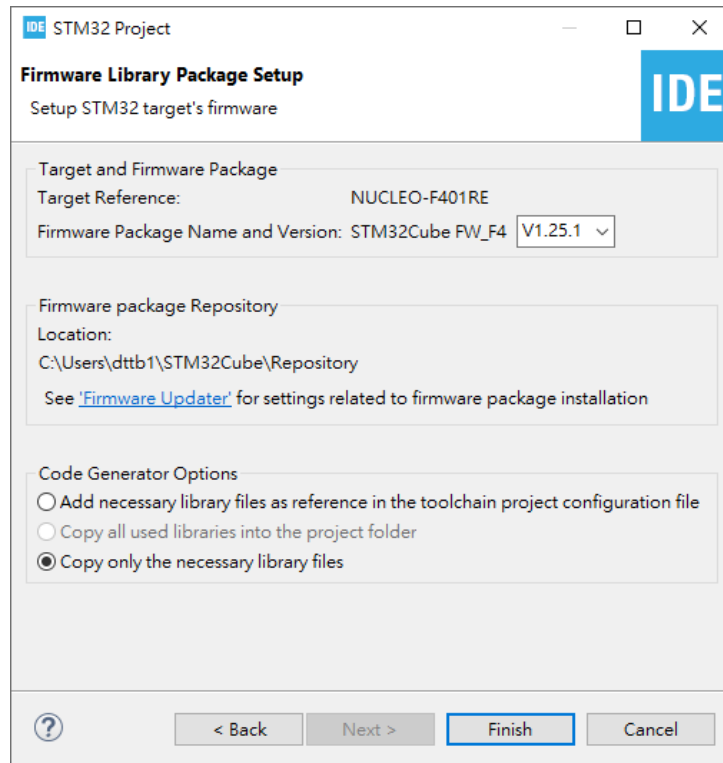
4. 選擇完成後點選下一步



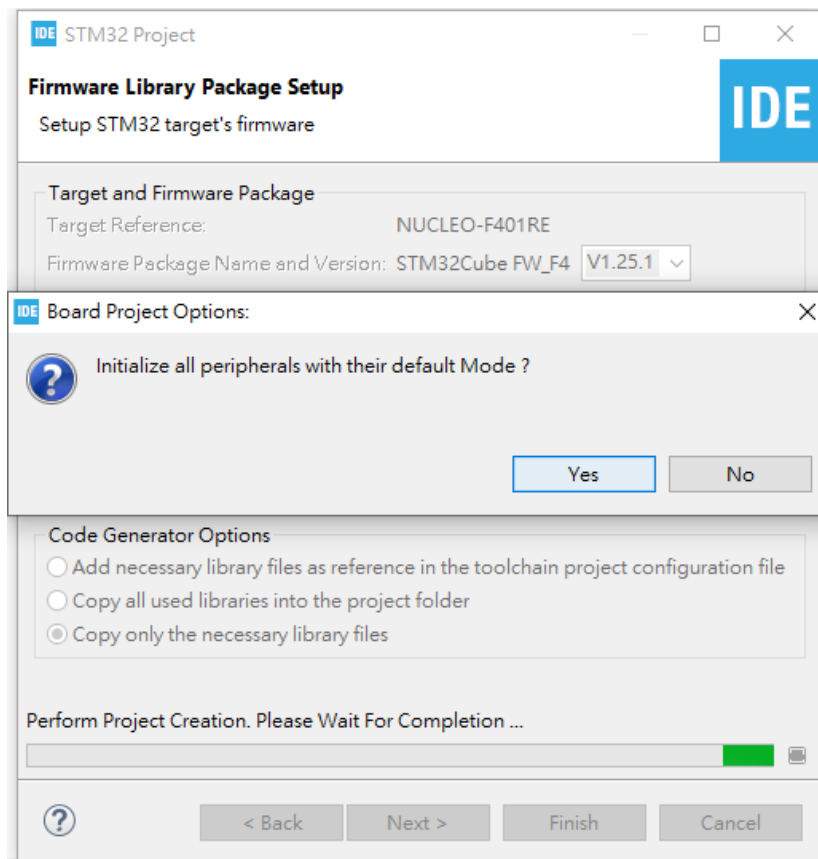
5. 輸入專案名稱與專案位置



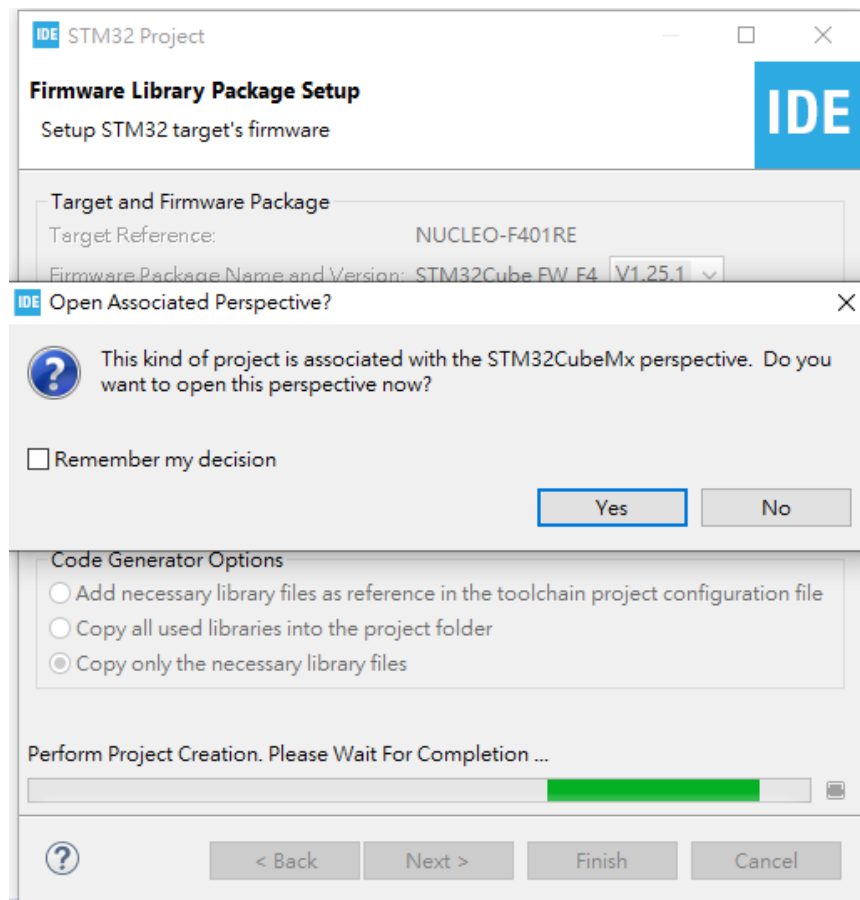
6. 選擇韌體並點選完成。如果沒有目前所選擇的版本，系統會自動下載所選擇的版本



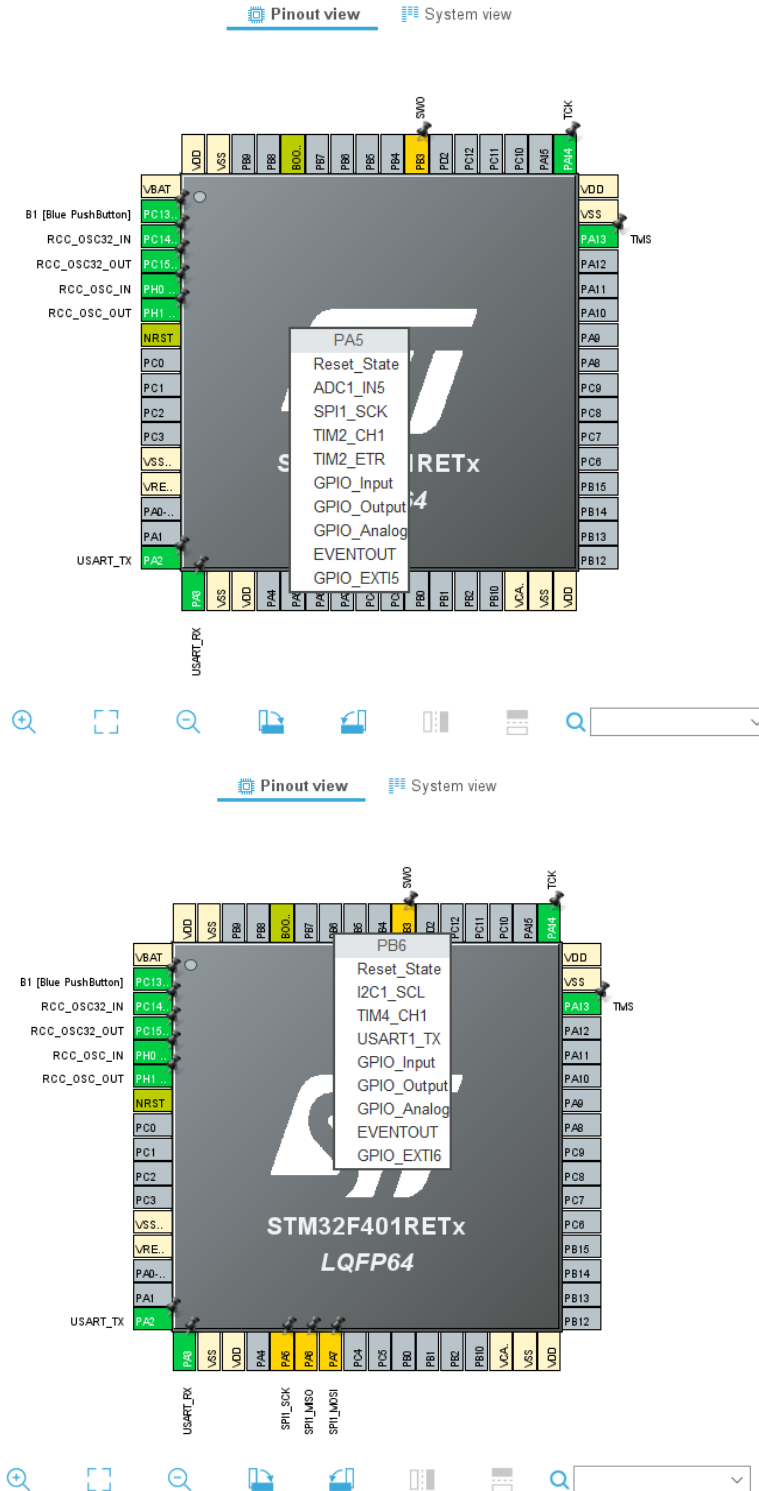
7. 視窗詢問初始化所有外部裝置至預設模式時選擇"是"



8. 詢問開啟STM32CubeMx時選擇"是"並開始定義腳位



9. 在腳位定義圖中，選擇PA5為SPI1_SCK、PA6為SPI1_MISO、PA7為SPI1_MOSI與PB6為GPIO_Output



10. 點選左側“System Core”對GPIO進行進階設定，將“GPIO output level”調整至“High”

The screenshot displays the 'GPIO Mode and Configuration' window. On the left, the 'System Core' category is expanded, and 'GPIO' is selected. The main area shows a configuration table for various peripherals, with 'GPIO' checked. Below this is a table of signal configurations:

Pin ...	Signal o...	GPIO o...	GPIO m...	GPIO P...	Maximu...	User La...	Modified
PB6	n/a	High	Output ...	No pull-...	Low		<input checked="" type="checkbox"/>
PC13-A...	n/a	n/a	Externa...	No pull-...	n/a	B1 [Blu...	<input checked="" type="checkbox"/>

Below the table, the 'PB6 Configuration' panel is visible, showing the following settings:

- GPIO output level: High
- GPIO mode: Output Push Pull
- GPIO Pull-up/Pull-down: No pull-up and no pull-down
- Maximum output speed: Low
- User Label: (empty field)

11. 點選"Timers"中的TIM2並將"Clock Source"改為"Internal Clock"後，將下方"Counter Period"數值改為"0xFFFFFFFF"

The screenshot displays the STM32CubeMX configuration tool for the TIM2 timer. The left sidebar shows the 'Timers' category expanded, with TIM2 selected. The main configuration area is divided into 'Mode' and 'Configuration' sections.

Mode Section:

- Slave Mode: Disable
- Trigger Source: Disable
- Clock Source: Internal Clock
- Channel1: Disable
- Channel2: Disable
- Channel3: Disable
- Channel4: Disable
- Combined Channels: Disable
- Use ETR as Clearing Source
- XOR activation

Configuration Section:

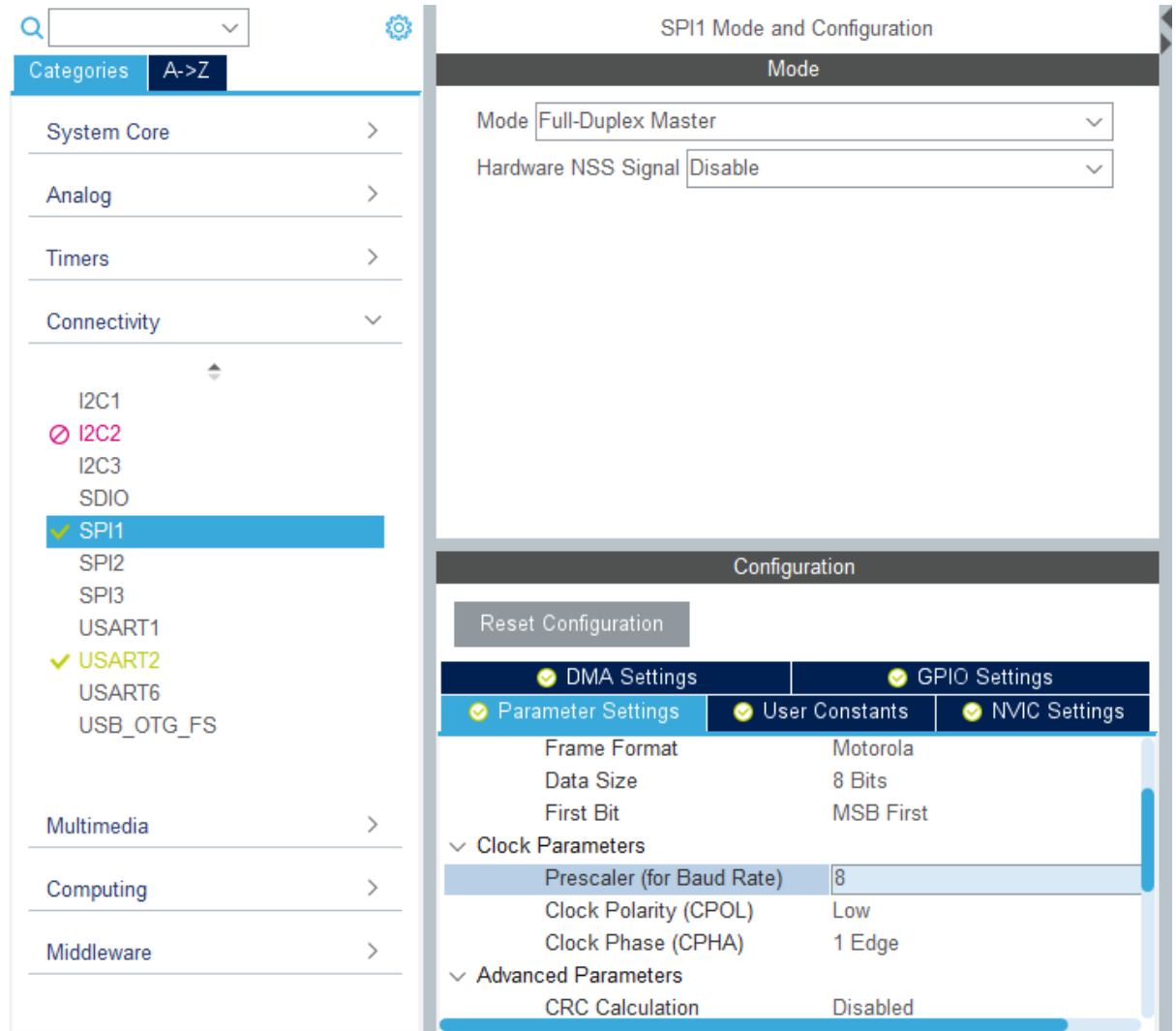
- Reset Configuration
- User Constants (checked)
- NVIC Settings (checked)
- DMA Settings (checked)
- Parameter Settings (checked)

Configure the below parameters :

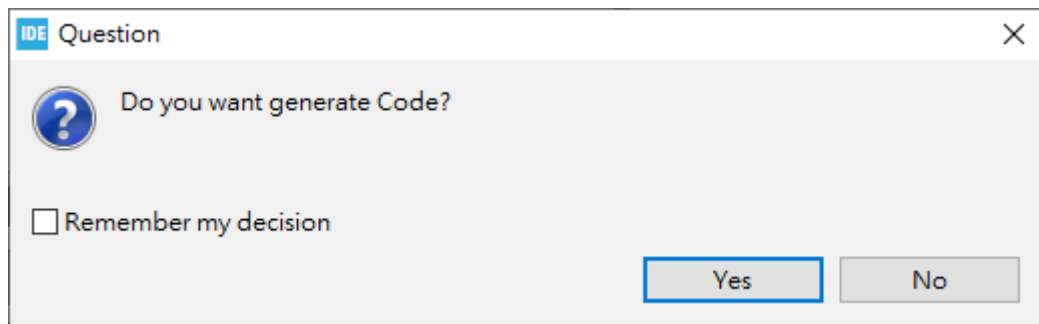
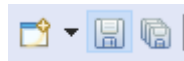
Search (Ctrl+F)

- Counter Settings
 - Prescaler (PSC - 16 bits val... 0
 - Counter Mode Up
 - Counter Period (AutoReload... 0xffffffff
 - Internal Clock Division (CKD) No Division
 - auto-reload preload Disable

12. 接下來跳至“Connectivity”設定“SPI1”，模式設定“Full-Duplex Master”並將鮑率設置在 2~8之間



13. 完成上面步驟後，點擊儲存標示儲存並自動產生程式



14. 如果在生成的main.c “Private variables”中看到剛設定的SPI、TIM2表示有之前的設定有完成

```
/* Private variables -----*/
SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;
```

15. 點開“main.h”並依下圖加入下方程式碼

```
/* USER CODE BEGIN Includes */
#include "platform.h"
#include "EcmUsrDriver.h"
/* USER CODE END Includes */

/* Exported macro -----*/
/* USER CODE BEGIN EM */
#ifndef PRINTF
#define PRINTF( str, ... ) \
    do{ \
        int n; \
        n = sprintf( printbuf, (str), ##_VA_ARGS_ ); \
        HAL_UART_Transmit( &huart2, (uint8_t *)printbuf, n, 0xffffffff); \
    }while(0)
#endif
#ifndef GETCHAR
#define GETCHAR userGetchar
#endif
/* USER CODE END EM */

/* USER CODE BEGIN EFP */
extern UART_HandleTypeDef huart2;
extern char printbuf[];
/* USER CODE END EFP */
```

16. 開啟“main.c”並加入下方程式碼

```
/* USER CODE BEGIN PFP */
char printbuf[128];
int main_ini(void);
/* USER CODE END PFP */

/* USER CODE BEGIN 2 */
main_ini();
/* USER CODE END 2 */

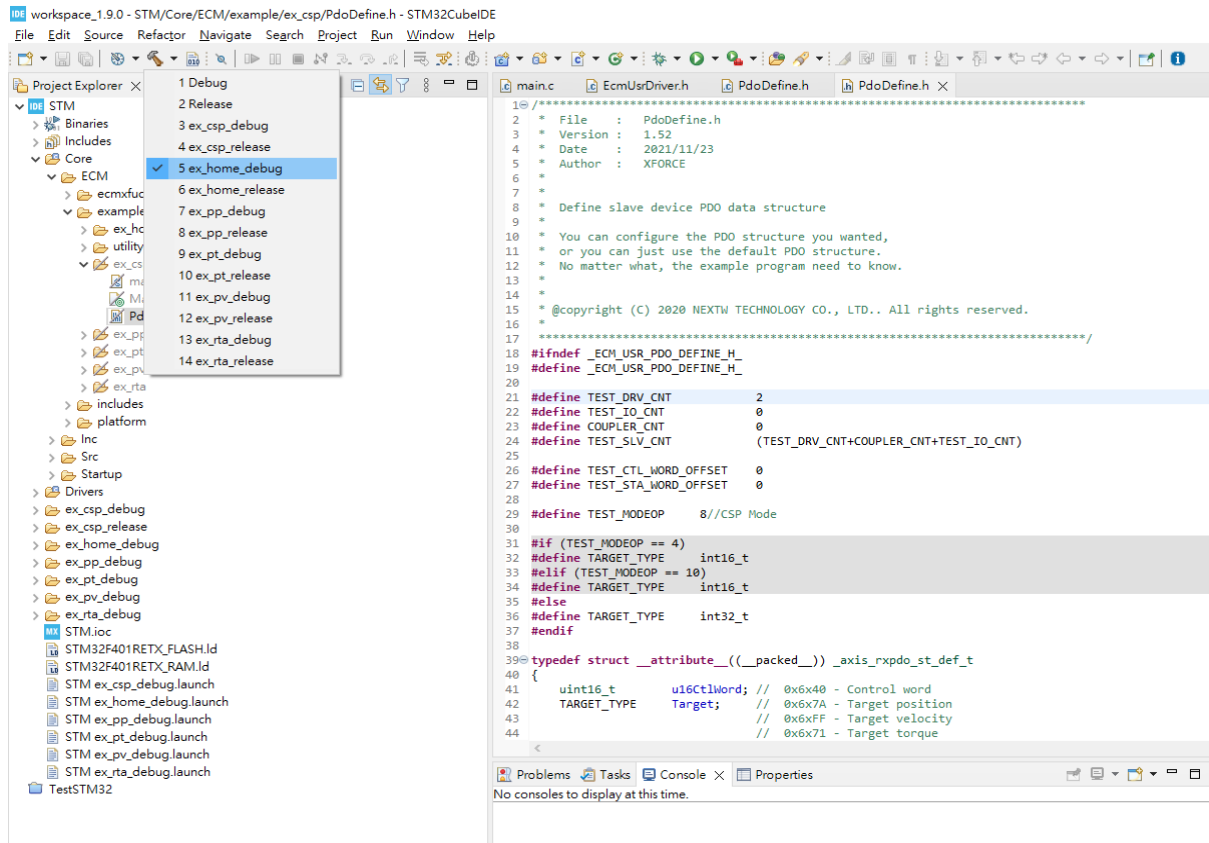
/* USER CODE BEGIN TIM2_Init 2 */
HAL_TIM_Base_Start(&htim2);
/* USER CODE END TIM2_Init 2 */
```

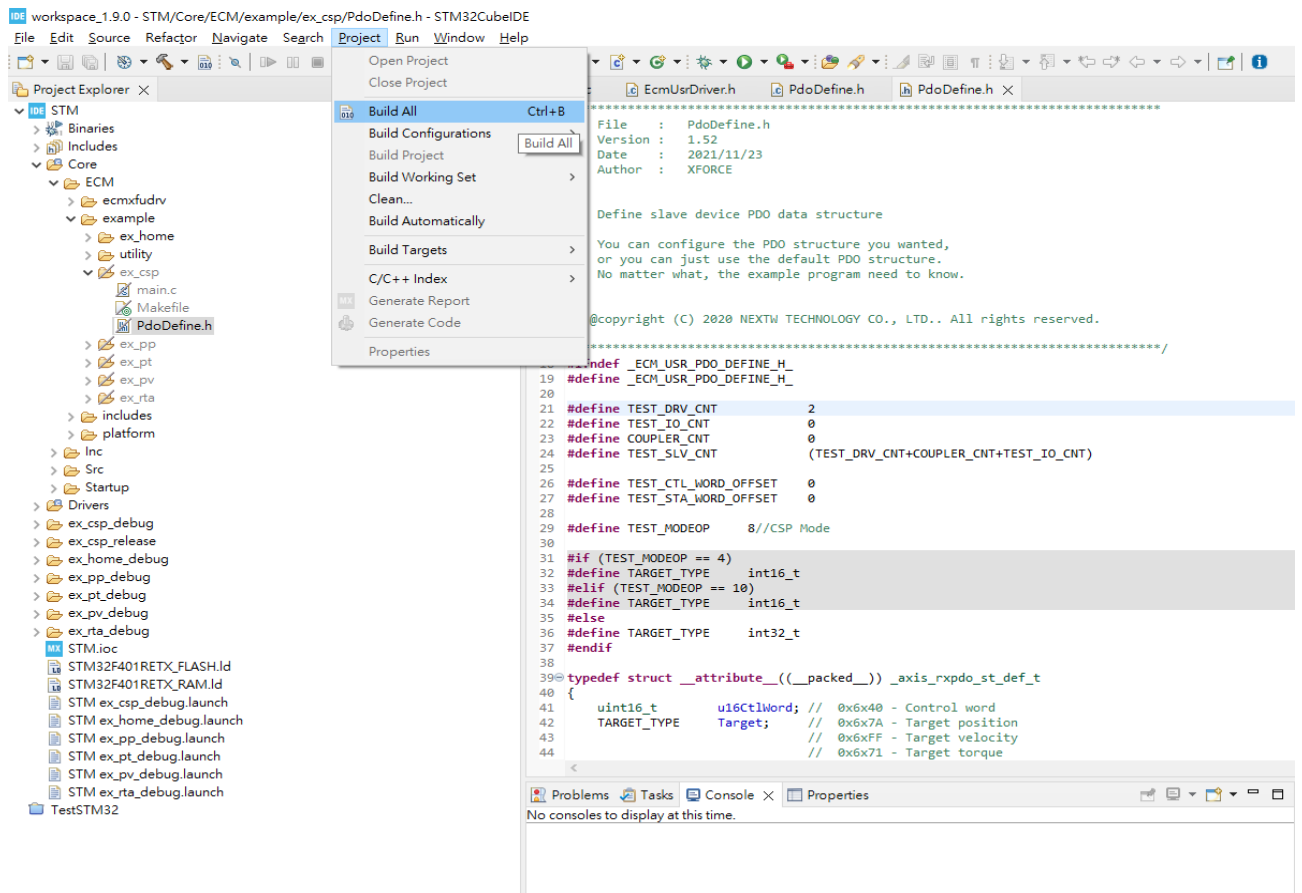
17. 複製EcmDriver.h、EcmUsrDriver.h、PdoDefine.h、platform.h與Utility.h(部分範例)貼入“Inc”資料夾

18. 複製crc32.c、EcmUsrDriver.c、main_ini.c(主程式:部分範例為不同名)與platform.c、Utility.c(部分範例)貼入Src資料夾

19. 已準備好執行範例程式

20. 點擊槌子圖示，並選取欲執行的程式來進行Build，或是點擊Project底下的Build all來建置程式

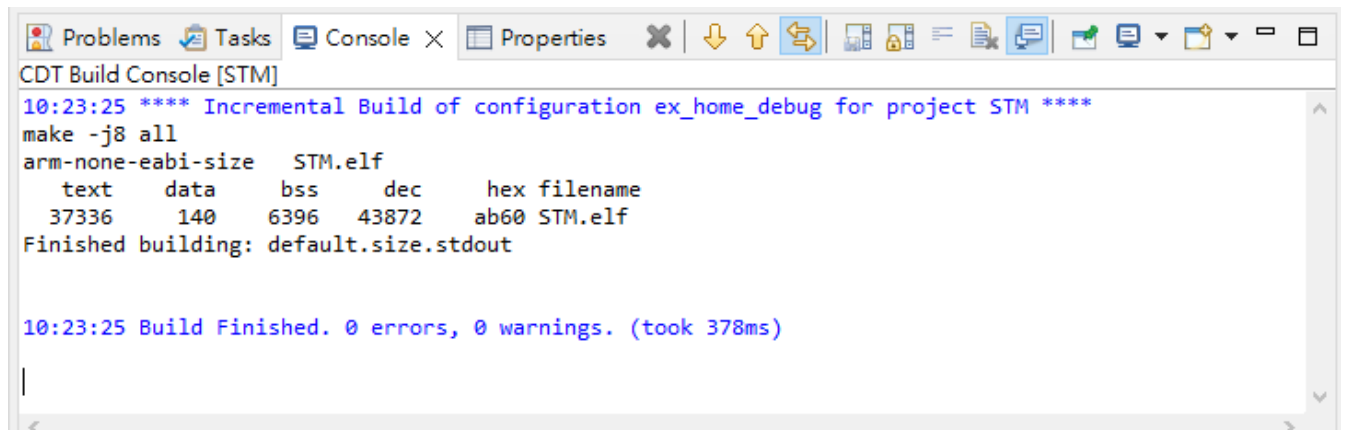




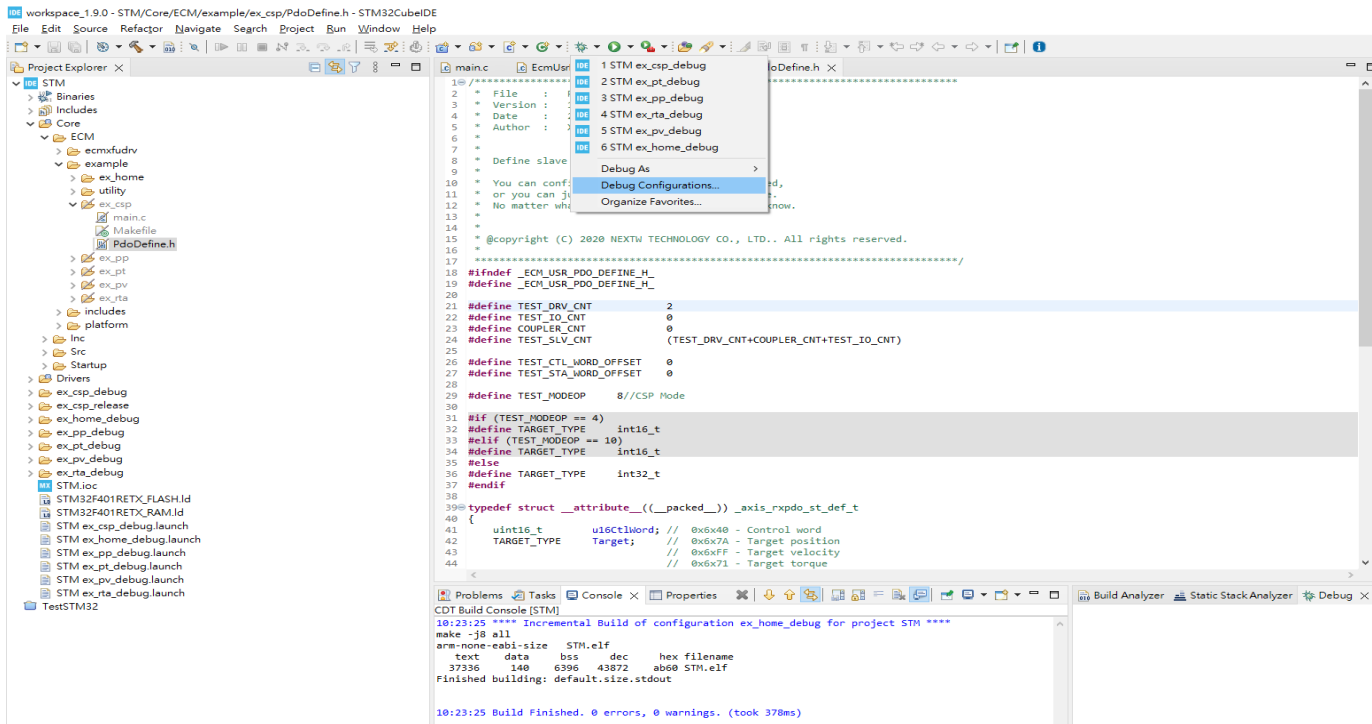
點擊後會開始建置build

在Console欄中顯示完成訊息

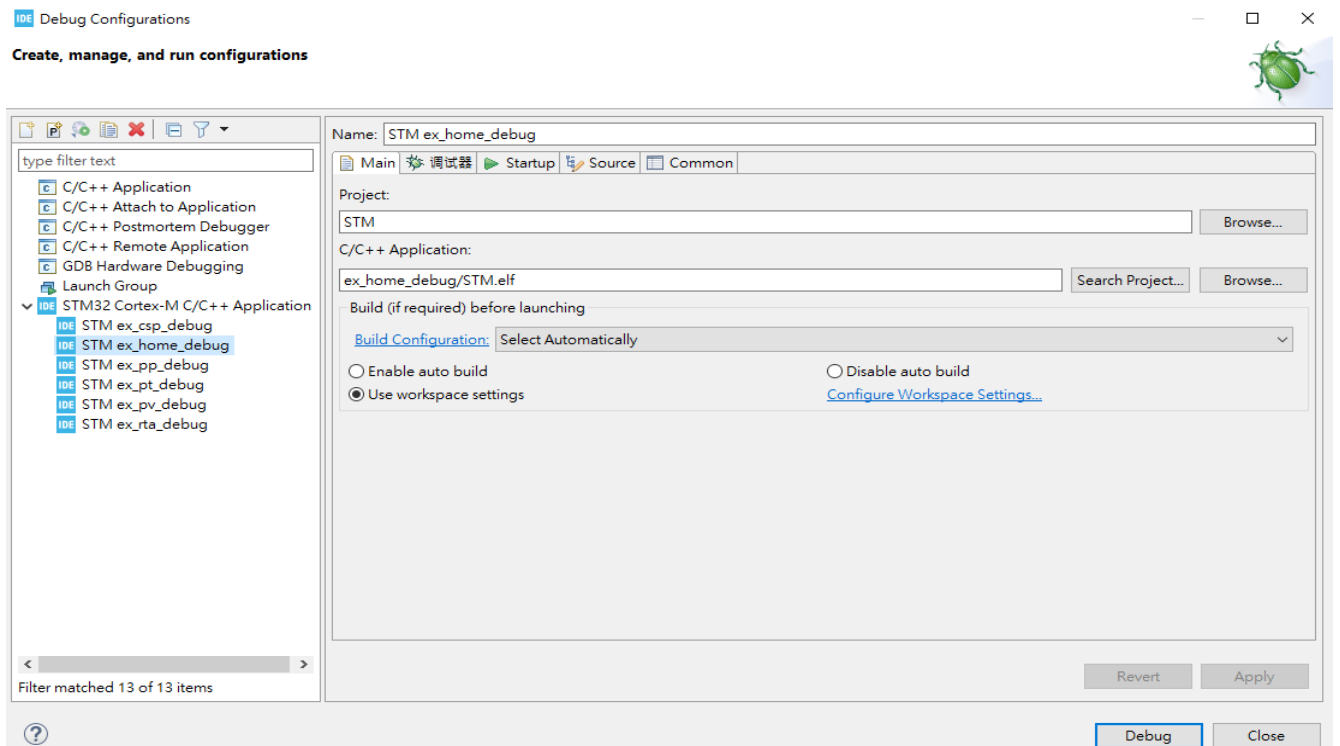
(若有錯誤請檢查錯誤原因並排除, 注意程式路徑不可包含非英文語系)



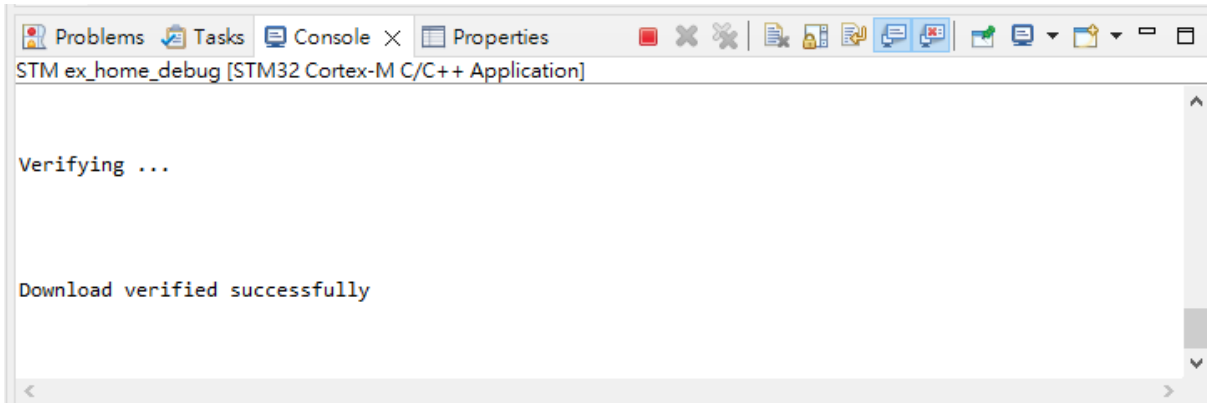
21. 下一步要進行Debug, 在上方工具列中點擊蟲子的圖示, 並點選Debug Configurations.



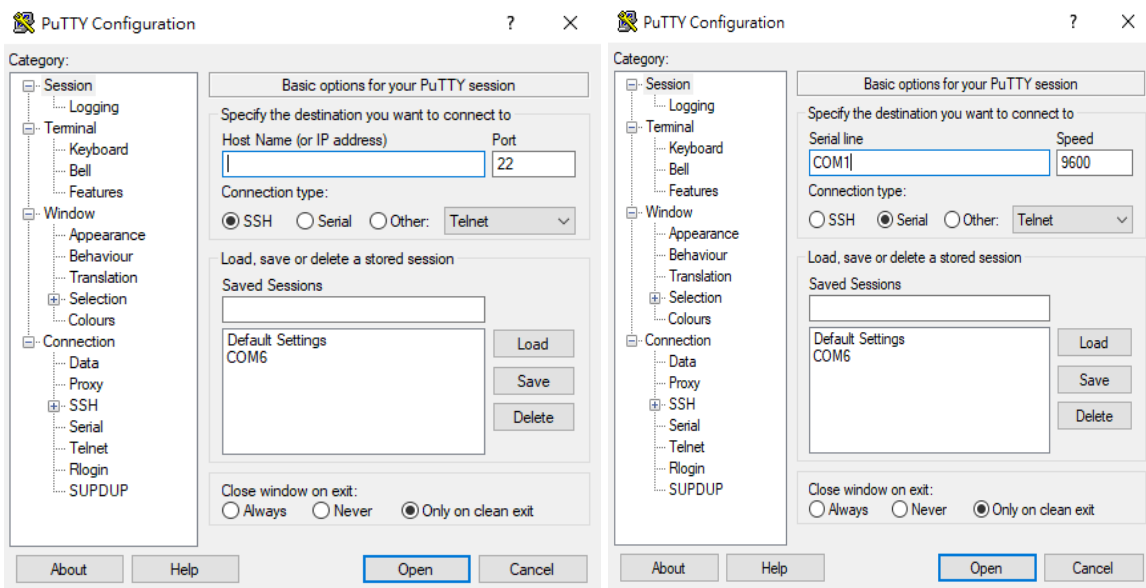
在Debug Configurations中點選STM32, 並選擇要Debug的程式再按下Debug就會開始偵錯



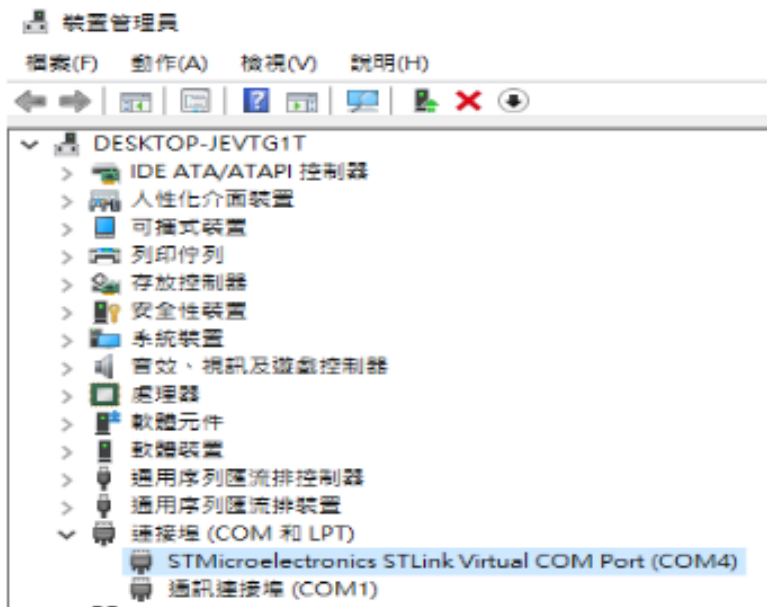
程式會下載至MCU中並在Console顯示相關訊息



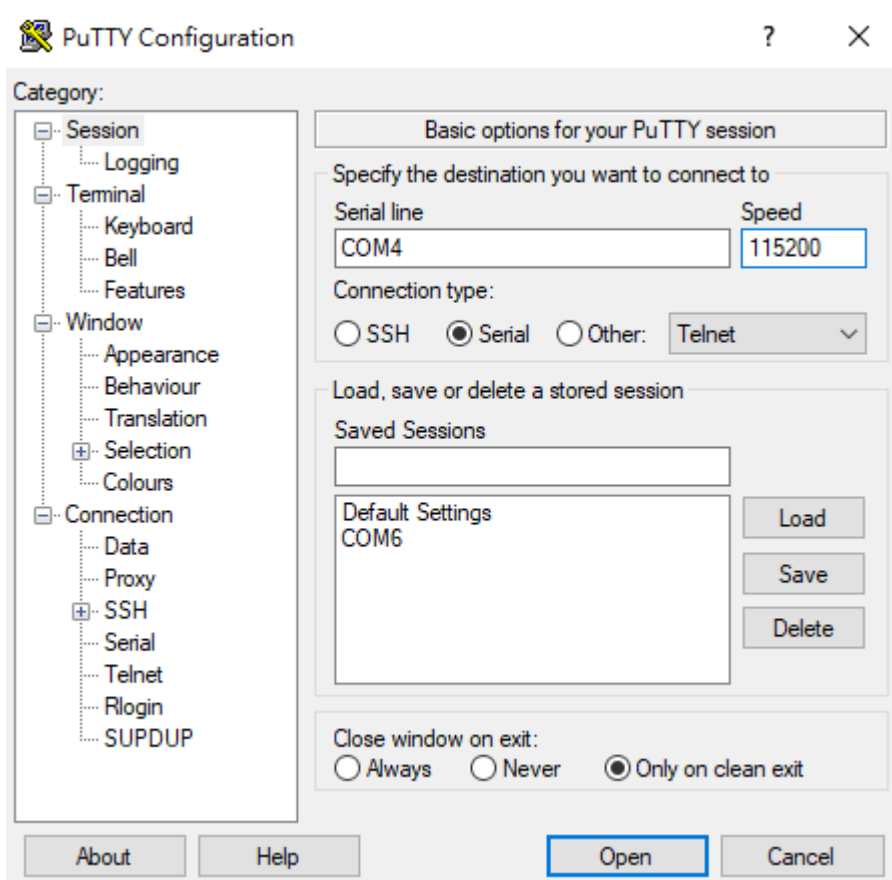
22. 開啟序列通訊埠程式 (如Putty), 並在Connection type中勾選Serial



在Serial line中輸入對應到的COM port(可以在裝置管理員中的連接埠確認, 本例為COM4)



設定好Serial line後, speed輸入 115200, 接著就可以點擊Open



23. 進入到Putty的畫面

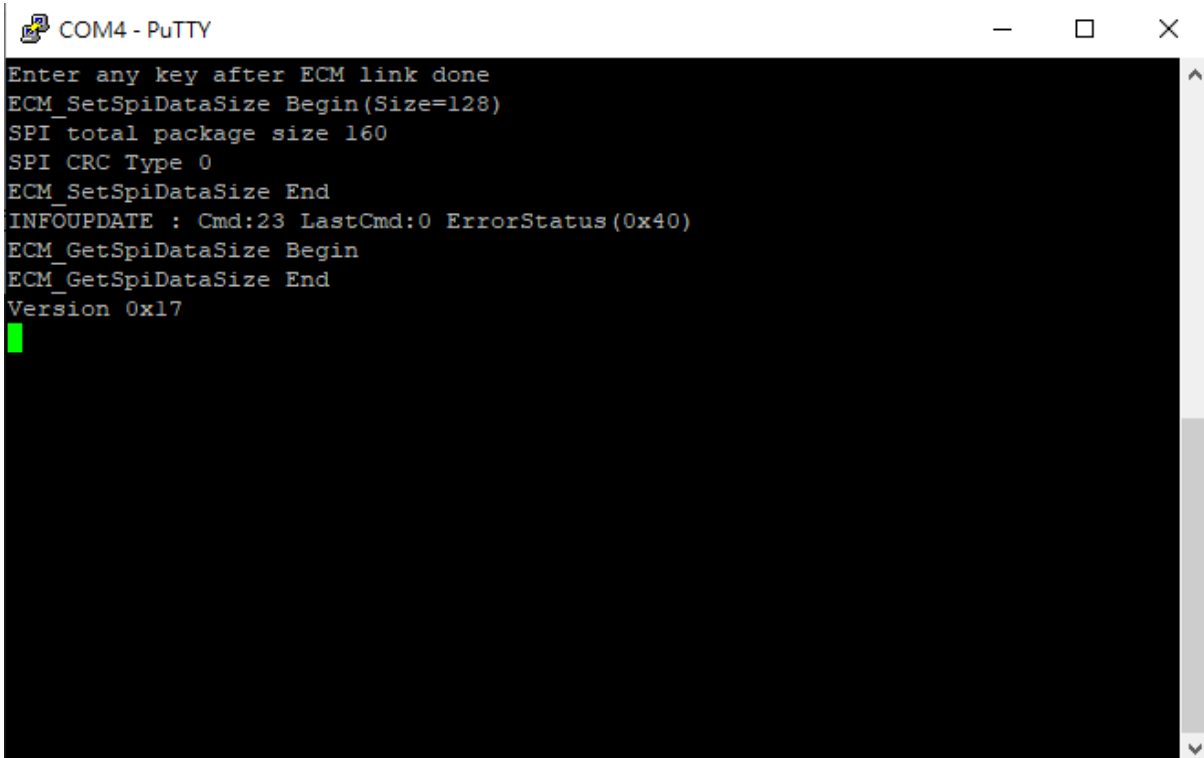


按下STM F401RE版上的reset鍵出現畫面上的字樣(或點選程IDE中執行按鈕)



```
COM4 - PuTTY
Enter any key after ECM link done
█
```

確認ECM-XFU-SK已連結至從站，且網路口燈亮起，按下任意鍵就會開始執行程式



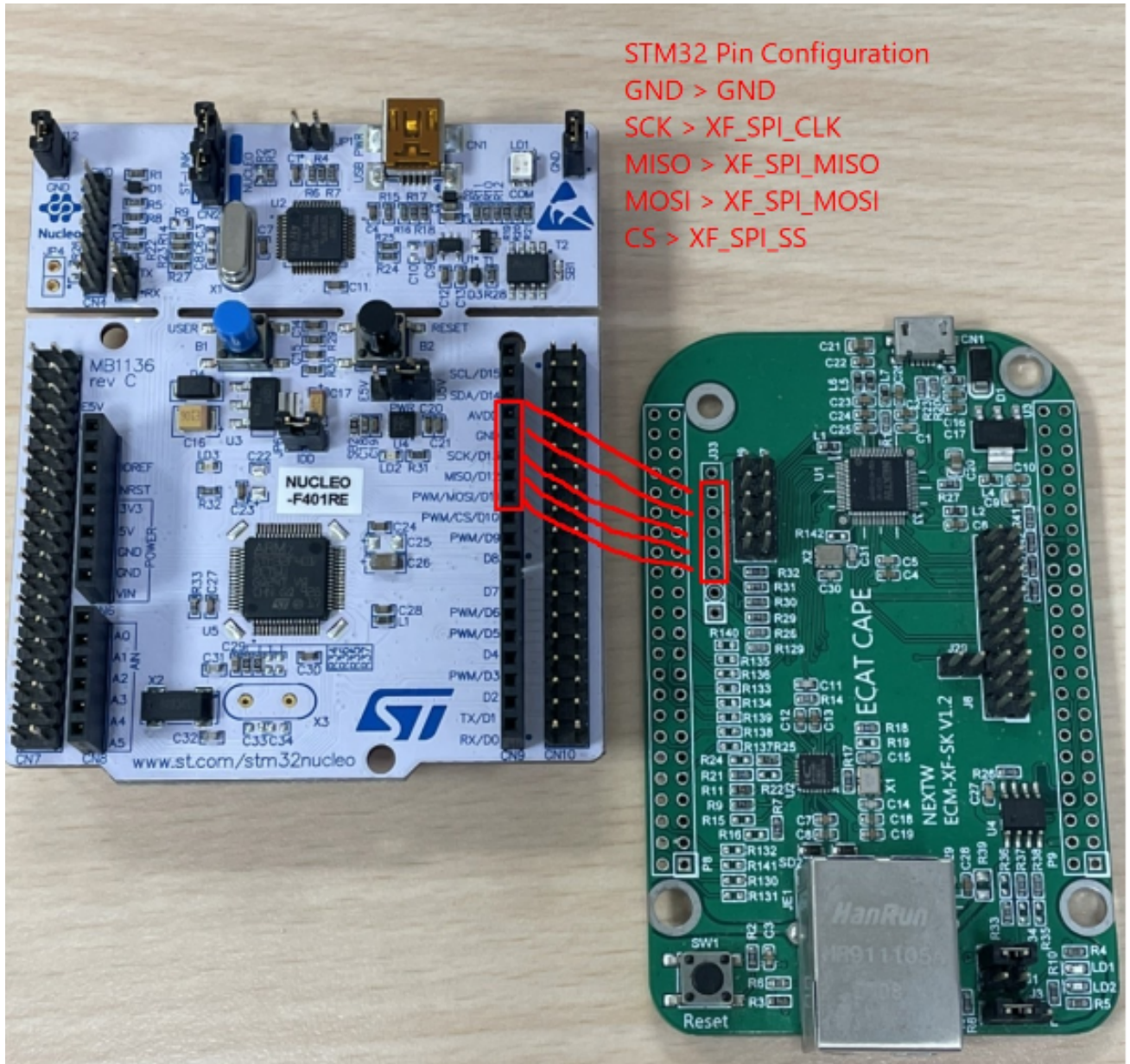
```
COM4 - PuTTY
Enter any key after ECM link done
ECM_SetSpiDataSize Begin(Size=128)
SPI total package size 160
SPI CRC Type 0
ECM_SetSpiDataSize End
INFOUPDATE : Cmd:23 LastCmd:0 ErrorStatus(0x40)
ECM_GetSpiDataSize Begin
ECM_GetSpiDataSize End
Version 0x17
█
```

有出現版本號表示成功代表SPI連線傳輸正常

若出現failed訊息可能為接觸不良，調整版子接合處並按下ECM-XF-SK上的reset鍵重新執行

2.3 實際接線範例

STM32 F401RE 腳位接線

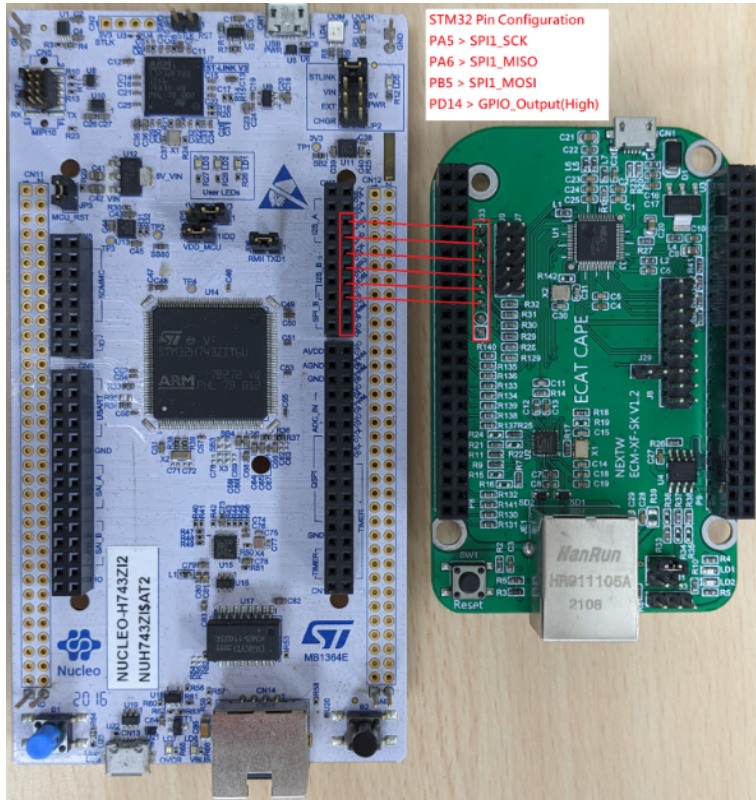


STM32 H7 腳位接線

STM32所生成之main.c與main.h不與F4版本共用，請使用H7範例為範本修改

請使用H7專用platform.h與platform.c

需要F4版本範例可以直接複製main_ini.c覆蓋至H7 main_ini.c



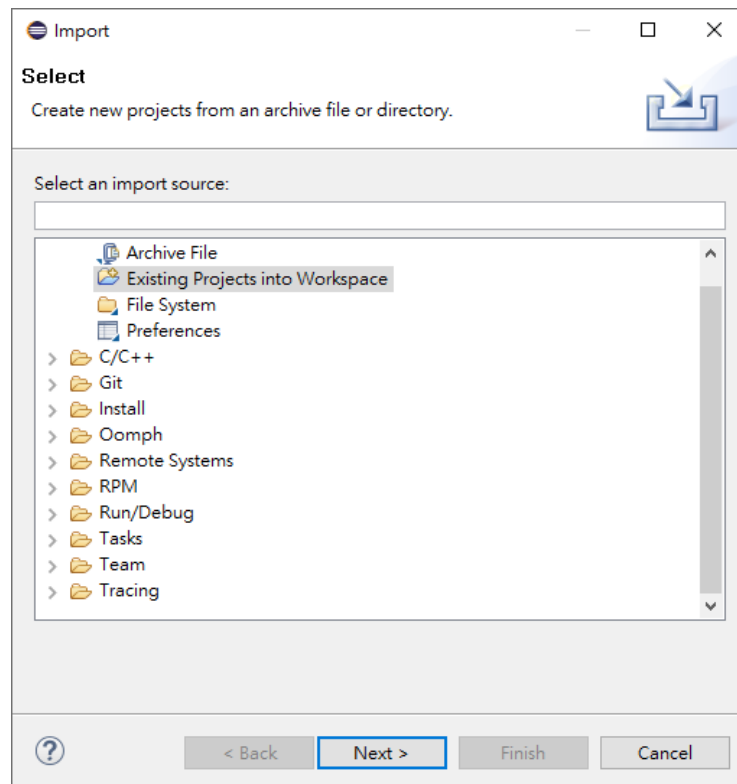
2.4 Nuvoton範例列表

以下範例為使用新唐M487與Nueclipse IDE進行編程

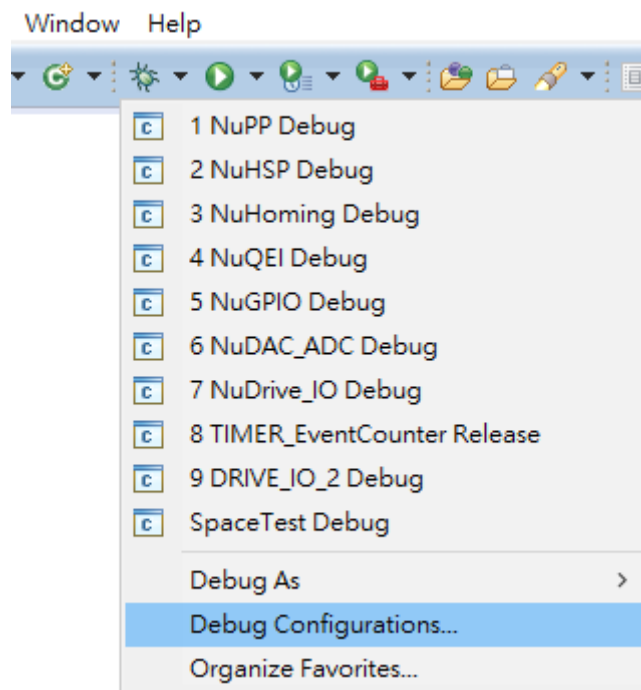
- NuDAC_ADC
ECM-XF晶片上DAC與ADC功能範例
- NuDrive_IO
一個驅動器馬達從站與一個IO從站之範例
- NuDrive
驅動器馬達範例
- NuEEPROM
顯示從站EEPROM範例
- NuGPIO
ECM-XF上GPIO測試範例
- NuHoming
一個驅動器馬達歸零(Homing)範例
- NuHSP
訊成HSP上的兩個馬達範例
- NuHSP_A
訊成HSP上的兩個馬達範例 (自動切換402狀態機Servo on)
- NuHSP_IO
訊成HSP上的兩個馬達加上一個IO從站範例
- NuIO
一個IO從站範例
- NuPP
Profile position mode範例
- NuQEI
讀取ECM-XF晶片上編碼器訊號範例

2.5 Nuvoton新唐範例環境建立導引

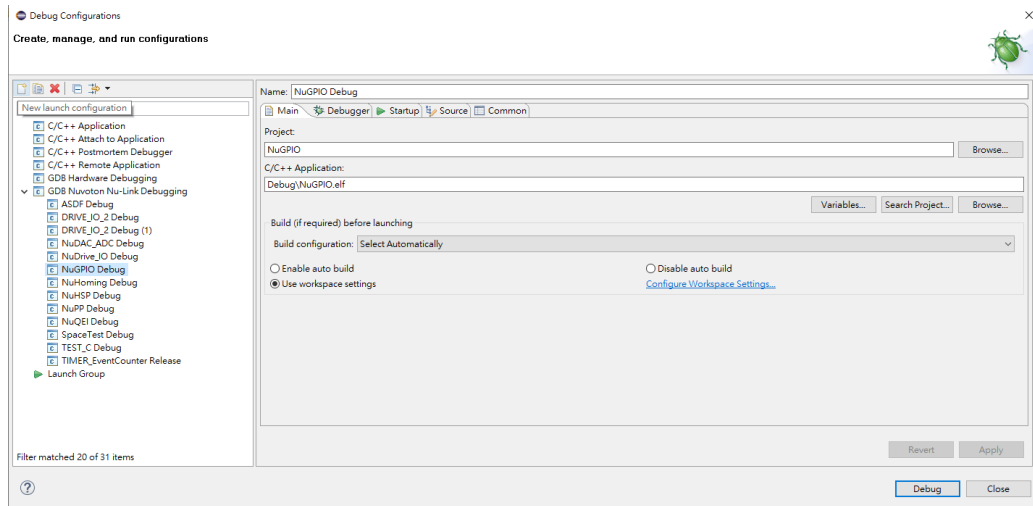
1. 至新唐官網依作業系統下載Nueclipse GCC(開源IDE), 連結為官方下載頁面:
<https://www.nuvoton.com/products/microcontrollers/arm-cortex-m4-mcus/m487-ethernet-series/?group=Software&tab=2>
2. 在同一頁面下載Nu-Link Keil driver
3. 同一頁面下載M480_BSP_CMSIS
4. 安裝IDE與Keil driver
5. 下載範例程式並放入workspace位置
6. 開啟Eclipse並在Project Explorer的位置滑鼠右鍵選擇”Import”
7. 在匯入畫面依以下順序操作: General > existing project into workspace



8. 選擇預計開啟的根目錄，範例程式會在中間的視窗出現並點選”Finish”
9. 在成功匯入專案後，右鍵點選或上方工具列點選建置(build)專案。專案可以允許建置Debug模式與Release模式，建議使用Debug模式。



- 當建置完成後，設定Debug配置與Run配置。點選圖標名稱為”New launch configuration”，接下來應會自動生成剛剛建置完成的專案名稱，確認名稱正確後點擊Debug，系統會自動將程式寫入晶片中



2.6 Nuvoton修改範例程式為實際應用導引

正確開啟範例程式後可依以下步驟順序進行改寫

使用Drive的應用以STFDrive或NuDrive為範本進行修改，中間會有其他參考別範例中的做法

PdoDefine.h設定

- 點選並進入PdoDefine.h中後，在#define中輸入從站數量與單站軸數。目前馬達控制器發展已有單控制器可多軸控制，所以可以是一個控制器帶動兩馬達進行運動，這樣的情況就會是一個從站與兩個軸數。

```
#define TEST_DRV_CNT      1
#define TEST_IO_CNT      0
```

- 完成設定後拉至下方繼續設定控制器資料結構(PDO structure)。資料結構分為RxPDO與TxPDO兩種，不同的從站如果資料結構不相同可以分開設定後最後同整。

```
typedef struct __attribute__((__packed__)) _axis_rxpdo_st_def_t
{
    uint16_t    u16CtlWord;    // 0x6x40 - Control word
    int32_t     n32Target;    // 0x6x7A - Target position
                                // 0x60FF - Target velocity
#ifdef TEST_HSP_DEV
    int32_t     n32Out;        // 0x6xFE
#endif
}AXIS_RXPDO_ST_DEF_T;
typedef struct __attribute__((__packed__)) _axis_txpdo_st_def_t
{
    uint16_t    u16StaWord;    // 0x6x41 - Status word
    int32_t     n32Actual;    // 0x6x64 - Actual position
                                // 0x6063 -
#ifdef TEST_HSP_DEV
    int32_t     n32In;        // 0x6xFD
#endif
}AXIS_TXPDO_ST_DEF_T;
```

STM32系列中main_ini.c或Nuvoton系列中main.c設定

1. 首先要先設定SPI通訊時間，STM32系列中的SPI通訊時間必須另外安裝STM32CubeMX來設定，Nuvoton系則以main.c中的#define TEST_SPI_FREQ進行設定或是尋找UserSys_Init()並填入SPI傳輸值。同時輸入欲測試的週期時間

```
#define TEST_SPI_FREQ          24000000
#define DC_ACTIVE_CODE        0x300
#define BASE_CYCTIME          1000000
```

```
UserSys_Init(TEST_SPI_FREQ);
```

2. 在此範例中有可以進行週期時間測試用的define可以直接倍數測試當前設定的週期時間，並提供RPM與PPR相關的設定、速度相關設定

```
/*
 * TEST_CYCTIME_DIVID
 * 1 : for TEST_CYCTIME_MULTI
 * 2 : 0.5ms
 * 4 : 0.25ms
 * 8 : 0.125ms
 *
 * TEST_CYCTIME_MULTI
 * 1 : for TEST_CYCTIME_DIVID
 * 2 : 2ms
 * 4 : 4ms
 */
#define TEST_CYCTIME_DIVID      1
#define TEST_CYCTIME_MULTI      1
#define TEST_CYCTIME_NS         ((BASE_CYCTIME*TEST_CYCTIME_MULTI)/TEST_CYCTIME_DIVID)
#define ONE_SEC_CYC_CNT         (1000000000/TEST_CYCTIME_NS)
```

3. 設定完成後下方為FIFO設置，FIFO為一暫存空間並依照順序與週期時間將命令由主站送至從站，以部分馬達控制為例週期時間為1ms時如果每次只送一次命令，很有可能會造成從站控制器以將步驟完成並閒置的狀態，導致馬達控制會有不連續運作的情況，此時就應加大每週期時間所傳輸的命令數量，這邊就會修改TEST_PDO_TO_FIFO_ONCE的數值，2為每周期間所傳輸的命令數量為兩筆。此步驟需經過測試才可得出最佳數值

```
#define TEST_PDO_TO_FIFO_ONCE  2
#define TEST_RXFIFO_CNT        40
#define TEST_TXFIFO_CNT        TEST_PDO_TO_FIFO_ONCE
```

4. 設定完成後跳至主程式int main_ini()或int main()中開始進行程式的編寫與說明，一開始會先將記憶體的位置清空以確保接下來運作正常

```

uint64_t u64Data;
int nStartFIFO = 0;
int i = 0, DrvIdx = 0, AxisIdx = 0, YasIdx = 0, nCnt1Sec=0, nRunTimeCnt=0;
int nVel = 0, nSumVel = 0, n32DO=0x5555;
int nret = 0, nServoState = 1;
int nLogStart = 0;
int n32CurPos[TEST_AXIS_CNT];
uint16_t u16LastSW[TEST_AXIS_CNT], u16StalWord[TEST_DRV_CNT][N_AXIS_IN_ONE_DRV];
uint32_t u32CycTimeCnt = 0, u32RunTimeCnt = 0, u32LogFifoCnt=0;
uint8_t u8LEDAxis = 0;
uint8_t u8LEDBit = 0;
uint8_t u8Version = 0, u8FifoCnt = 0, u8FifoCntMax = TEST_RXFIFO_CNT;
uint8_t u8State = 0, u8WkcErrCnt = 0, u8CrcErrCnt = 0, u8IsSlvAlive = 0;
uint8_t u8LastState = 0, u8LastWkcErrCnt = 0, u8LastCrcErrCnt = 0;
uint16_t u16RxPDOSize = 0, u16TxPDOSize = 0, u16SpiSize = 0;
int nDriveRxPDOSize = 0, nDriveTxPDOSize = 0;
int8_t SlaveCnt = 0;
RXPDO_ST_DEF_T *pAllDevRx;
TXPDO_ST_DEF_T *pAllDevTx;
AXIS_RXPDO_ST_DEF_T *pRxPDOAxis;
AXIS_TXPDO_ST_DEF_T *pTxPDOAxis;
memset(RxPDOData, 0, sizeof(RxPDOData));
memset(TxPDOData, 0, sizeof(TxPDOData));
memset(nPos, 0, sizeof(nPos));

```

5. EtherCAT主站的第一個命令為ECM_InitLibrary(&u16SpiSize)。當中會執行SPI資料大小的設定與確認，同時會確認主站IC的韌體版本等功能。u16SpiSize為欲設定新SPI資料大小，如其值為0時則使用預設大小112Bytes。

```
u8Version = ECM_InitLibrary(&u16SpiSize);
```

6. 當確認項目皆正常執行後，開始進行EtherCAT初始化
ECM_EcatInit(DCActCode, CycleTime)，並進入Init state
ECM_EcatInit(DCActCode, CycleTime)中DCActCode值0為關閉DC sync功能、0x300啟動Sync0、0x700同時開啟Sync0與Sync1，CycleTime為週期時間，單位為ns。

※此函數會針對"所有" Slave 下達相同的DCActCode，若對於不同Slave要設定不同的DCActCode，則需使用ECM_CMD_ECAT_DCSYNC（命令代碼50）

```
ECM_EcatInit(DC_ACTIVE_CODE, (BASE_CYCTIME*TEST_CYCTIME_MULTI) / TEST_CYCTIME_DIVID);
```

7. 確認主站與從站連接後透過ECM_StateCheck(Slave, ExpectedState, Timeout)進入Pro-OP開始設定PDO配置，如果從站已有預設配置則只需要給予相對應的Map Index即可，如為特殊配置則需告知，SetPdoConfig提供最多3組PDO各8個Object的配置大小，多數範例配置內容在main_ini.c或main.c中，目前唯Drive範例在另外在Utility.c中。如果使用ConfigDrive進行配置，內部已有Reconfig與ShowPDOConfig的動作，所以無需另外加入Reconfig動作，其他範例則需在完成Configure PDO後加入Reconfigure的動作與ShowPDOConfig。
ECM_StateCheck(Slave, ExpectedState, Timeout)中Slave如為0xFF則表示針對所有從站同時命令，ExpectedState則是欲前往之狀態(Pro-OP、Safe-OP與OP狀態)，Timeout為逾時等待時間，如出現無法狀態切換時可能等待時間過短。

```
ConfigDrive(1, 0, (TEST_DRV_CNT - 1), 1, N_AXIS_IN_ONE_DRV, 0x1602, 0x1A02);
```

或

```

RxPDOConfig[i].SmaIdx = RxPDO_ASSIGN_IDX;
RxPDOConfig[i].PDOCnt = 1;
RxPDOConfig[i].MapIdx[0] = RxPDO_MAP_IDX;
RxPDOConfig[i].ObjsCnt[0] = 2;
SetPdoConfTbl(&RxPDOConfig[i], 0, 0, 0x6040, 0, 16); //control word // 16 bits = 2 bytes for TEST_RXPDO_SIZE
// the 1st parameter is PDO_CONFIG_HEAD
// the 2nd parameter is 0 due to RxPDOConfig.PDOCnt = 1
// the 3rd parameter is 0 for the first object as RxPDOConfig.ObjsCnt[0] = 2
// the 4th parameter is a control word index 0x6040
// the 5th parameter is a sub-index for 0x6040
// the 6th parameter is the bit size for 4th parameter
SetPdoConfTbl(&RxPDOConfig[i], 0, 1, 0x607A, 0, 32); //target position // 32 bits = 4 bytes for TEST_RXPDO_SIZE
TxPDOConfig[i].SmaIdx = TxPDO_ASSIGN_IDX;
TxPDOConfig[i].PDOCnt = 1;
TxPDOConfig[i].MapIdx[0] = TxPDO_MAP_IDX;
TxPDOConfig[i].ObjsCnt[0] = 2;
SetPdoConfTbl(&TxPDOConfig[i], 0, 0, 0x6041, 0, 16); //status word // 16 bits = 2 bytes for TEST_TXPDO_SIZE
SetPdoConfTbl(&TxPDOConfig[i], 0, 1, 0x6064, 0, 32); //actual position // 32 bits = 4 bytes for TEST_TXPDO_SIZE

```

```
-ECM_EcatReconfig();
```

- 完成配置後進行ECM_CheckMEMSpace(TEST_PDO_FIFO_ONCE), 此步驟為確認各部分相關記憶體的大小, 參數需輸入每週期時間塞入多少命令。

```
ECM_CheckMEMSpace(TEST_PDO_TO_FIFO_ONCE);
```

- 檢查確認沒有問題後利用ECM_StateCheck(Slave, ExpectedState, Timeout)進入Safe-OP狀態

```
ECM_StateCheck(0xFF, EC_STATE_SAFE_OP, 1000);
```

- 可進入Safe-OP狀態後繼續執行ECM_StateCheck(Slave, ExpectedState, Timeout)進入OP狀態

- 進入OP狀態後執行ECM_CheckDCStable()確認DC狀態穩定

```
ECM_CheckDCStable();
```

- 當確認完成後, 以ECM_Drv402SM_StateSet(Axes, ServoOn/OffState)或ECM_Drv402SM_Enable(Axes, Slaves)啟用402 state machine至servo on階段, 此階段為自動切換402 state machine的方式, 如果從站無法使用以上兩種切換方式時, 可以參考STF4HSP或NuHSP中的手動切換模式進行單步切換至Servo On狀態

```
ECM_Drv402SM_StateSet((DrvIdx*N_AXIS_IN_ONE_DRV) + AxisIdx, SERVO_ON_STATE);
```

```
PdoExchangeAndGet402State(DrvIdx, AxisIdx, &u8State);
```

或

```
ECM_Drv402SM_Enable(0, 0);
```

- Servo On完成後進行編碼器與控制器的位置對齊AlignmentPosition(), 並使用ECM_InitFIFO()初始化FIFO同時使用ECM_ClearFIFO(Direction)將FIFO內部清空後以ECM_EnableFIFO(Enable)來啟用FIFO功能
ECM_ClearFIFO(0), 0為清除TxPDO與RxPDO
ECM_EnableFIFO(1), 1為Enable、0為Disable

```

ECM_InitFIFO();
ECM_ClearFIFO(0); // 0 for TX and RX both
PRINTF("ClearFIFO\r\n");
ECM_EnableFIFO(1); // Enable FIFO
PRINTF("EnableFIFO\r\n");

```

14. 完成上方步驟後已完成相關相關設定，接下來的資料交換皆會在while迴圈中不斷重複運行，以判斷FIFO的數量為基準，如果FIFO已經快要塞滿時，則選擇暫停塞入命令，此外持續將命令塞入FIFO，速度與位置相關設定在Utility.c中可依需求進行編寫。

2.7 常見錯誤說明

在執行相關設定時如果跳出Error相關問題，請依以下進行排除

階段	顯示	解決方法
ECM_Init_Library	wait ASYNC done timeout	請確認主站與從站連接是否正常，主站與從站網口燈號是否亮起
ECM_Init_Library	u8ErrorStatus 0x40	初次設定會跳出的警告，Novoton系列按Enter繼續
通期	wait ASYNC done timeout	等待時間不足或從站錯誤（上次的非即時命令尚未完成）
通期	CRC Error	SPI傳輸訊號錯誤，請確認SPI主站與EtherCAT主站IC連線

2.8 402自動切換與手動切換方法

- 在STF4Drive範例中用ECM_Drv402SM_StateSet()與PdoExchangeAndGet402State()來進行Servo ON
- 其他範例中以ECM_Drv402SM_Enable()的方式進行Servo On
ECM_Drv402SM_Enable中的參數分別為軸與從站
所以當為一對一控制器時兩站Enable就會變成
ECM_Drv402SM_Enable(0, 0);
ECM_Drv402SM_Enable(1, 1);
如果為一對二控制器且有兩站時命令就會變成
ECM_Drv402SM_Enable(0, 0); //第0站第0個軸

ECM_Drv402SM_Enable(1, 0); //第0站第1個軸
 ECM_Drv402SM_Enable(2, 1); //第1站第2個軸
 ECM_Drv402SM_Enable(3, 1); //第1站第3個軸

- 手動切換的方式請參考STF4HSP或NuHSP中的切換邏輯進行切換402 state machine

```

for(DrvIdx=0;DrvIdx<TEST_SLAVE_CNT;DrvIdx++){
    for(AxisIdx=0;AxisIdx<N_DRV_IN_ONE_SLV;AxisIdx++){
        j = 0;
        while(1){
            nret = ECM_EcatPdoFifoDataExchange(PDO_FIFO_DEFAULT_CNT, RxData, TxData, u16PDOSize, &u8FifoCnt, &u8WkcErrCnt, &u8CrcErrCnt);
            if(nret>0){
                u16LogStatus[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx] = pTxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16StalWord;

                PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx] = pTxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16StalWord & CIA402_SW_STATE_MASK;
                PRINTF("DrvIdx = %d, AxisIdx = %d, state = 0x%x\r\n", DrvIdx, AxisIdx, PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]);

                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_NOTREADYTOSWITCHON){
                    UserDelay(1000);
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_SWITCHEDONDISABLED){
                    pRxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16CtrlWord = 0x6; //Control word: Shutdown = 0x6
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_READYTOSWITCHON){
                    pRxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16CtrlWord = 0x7; //Control word: Switch on = 0x7
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_SWITCHEDON){
                    pRxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16CtrlWord = 0xF; //Control word: Enable operation = 0xF
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_OPERATIONENABLED){
                    j++;
                    if(j==3){
                        break;
                    }
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_QUICKSTOPACTIVE){
                    pRxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16CtrlWord = 0x0; //Control word: Disable voltage = 0x0
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_FAULTREACTIONACTIVE){
                    UserDelay(1000);
                }
                if(PDOstate[(DrvIdx*N_DRV_IN_ONE_SLV)+AxisIdx]==CIA402_SW_FAULT){
                    pRxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16CtrlWord = 0x0; //Control word: Fault reset = 0x0->0x80
                    nret = ECM_EcatPdoFifoDataExchange(PDO_FIFO_DEFAULT_CNT, RxData, TxData, u16PDOSize, &u8FifoCnt, &u8WkcErrCnt, &u8CrcErrCnt);
                    UserDelay(1000);
                    pRxPDOData->DRIVE_GROUP_0[DrvIdx].HSP[AxisIdx].u16CtrlWord = 0x80;
                    nret = ECM_EcatPdoFifoDataExchange(PDO_FIFO_DEFAULT_CNT, RxData, TxData, u16PDOSize, &u8FifoCnt, &u8WkcErrCnt, &u8CrcErrCnt);
                    UserDelay(1000);
                }
            }
        }
    }
}
    
```

第 3 章 Beaglebone透過SPI介面範例說明

3.1 設置Beaglebone系統

STEP1.製作Beaglebone的系統SD卡

製作Beaglebone的系統碟，可至Beaglebone的官方頁面下載後安裝，我們建議使用Debian 9或之後的Debian穩定發行版。若用戶需要用自行編譯的linux內核，我們建議使用4.4以後的LTS版本，開啟MCSPi和SPIDEV功能，且支援U-boot overlay。

STEP2. (開機步驟可參考2.2)開機後連接Beaglebone連接網際網路，安裝device-tree-compiler套件，使用Debian系統的可用以下指令：

```
$ sudo apt-get install device-tree-compiler
```

STEP3. 將系統韌體的dtbo反編譯成dts，以下為指令：

```
$ dtc -I dtb -O dts /lib/firmware/BB-SPIDEV1-00A0.dtbo > BB-SPIDEV1-00A0.dts
```

編輯dts，修改以下兩處：

一，搜尋 pinctrl-single,pins，將

```
pinctrl-single,pins = < 0x190 0x33 0x194 0x33 0x198 0x13 0x19c 0x13 >;
```

改成

```
pinctrl-single,pins = < 0x190 0x33 0x194 0x13 0x198 0x33 0x19c 0x13 >;
```

二，搜尋 ti,pio-mode，在下一行插入以下文字：

```
ti,pindir-d0-out-d1-in;
```

將修改後的dts檔編譯為dtbo，編譯完成後將dtbo檔放在/lib/firmware目錄下。以下為編譯的指令：

```
$ dtc -I dts -O dtb BB-SPIDEV1-00A0.dts > BB-SPIDEV1-00A0.dtbo
```

STEP4. 編輯/boot/uEnv.txt, 加上以下兩行:

```
enable_uboot_cape_universal=1
```

```
uboot_overlay_addr4=/lib/firmware/BB-SPIDEV1-00A0.dtbo
```

STEP5. 重啟Beaglebone, 上述步驟完成後可以看到/dev/spidev2.0、/dev/spidev2.1裝置。若SPIDEV裝置沒有出現, 請特別留意P9.28、P9.29、P9.30、P9.31是否有和其他的overlay有衝突。

3.2 測試範例

以Beaglebone+UART為測試範例, 安裝USB to TTL驅動程式。

STEP1. TTL UART端接Beaglebone的J1, 打開COM PORT程式, 下圖3-1是以PuTTY為範例。

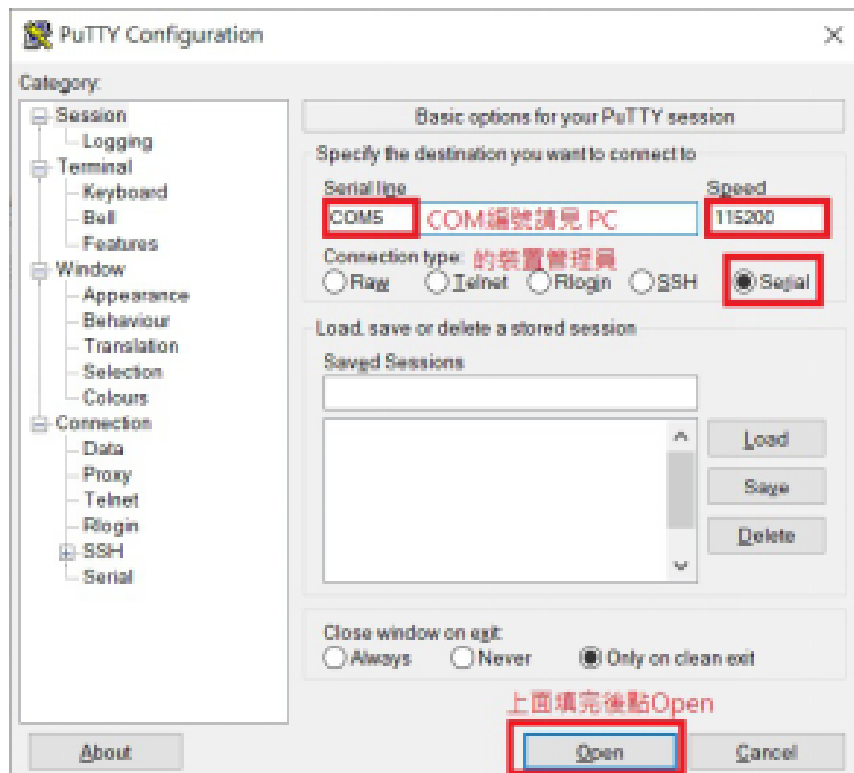


圖3-1

STEP2. 使用跳線帽將ECM-XFU-SK的J2短路(下圖3-2紅框處), J2短路後代表ECM-XFU-SK的電路由外板(即Beaglebone)供電, 插上Beaglebone, EtherCAT網路線接ECM-XF-SK的網口。

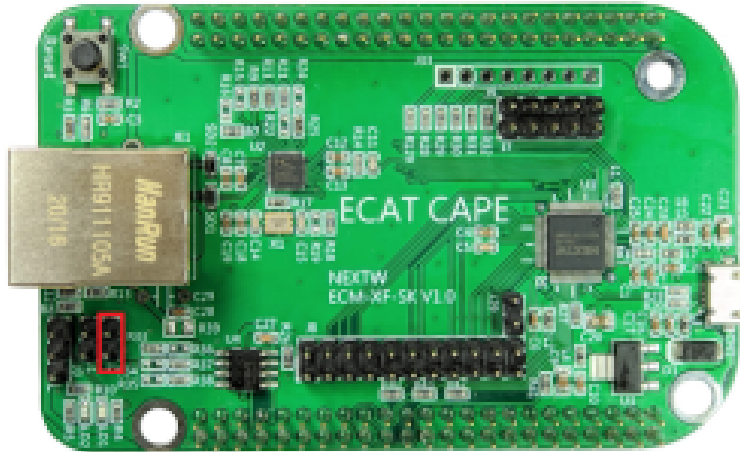


圖3-2

STEP3. 電源接Beaglebone P4(Micro USB), ECM-XFU-SK由Beaglebone 供電。下圖3-3為接線方式, 下圖為ECM-XFU-SK V1.3接法, 圖下板是Beaglebone, 上板是ECM-XFU-SK。

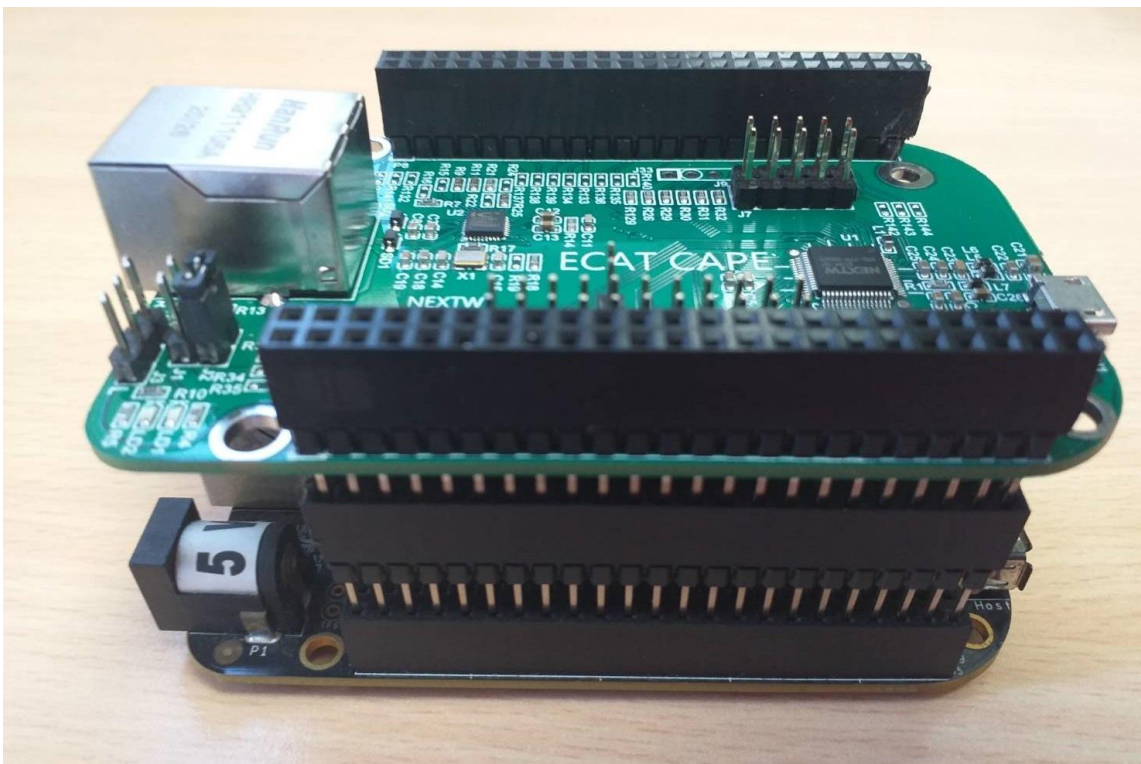
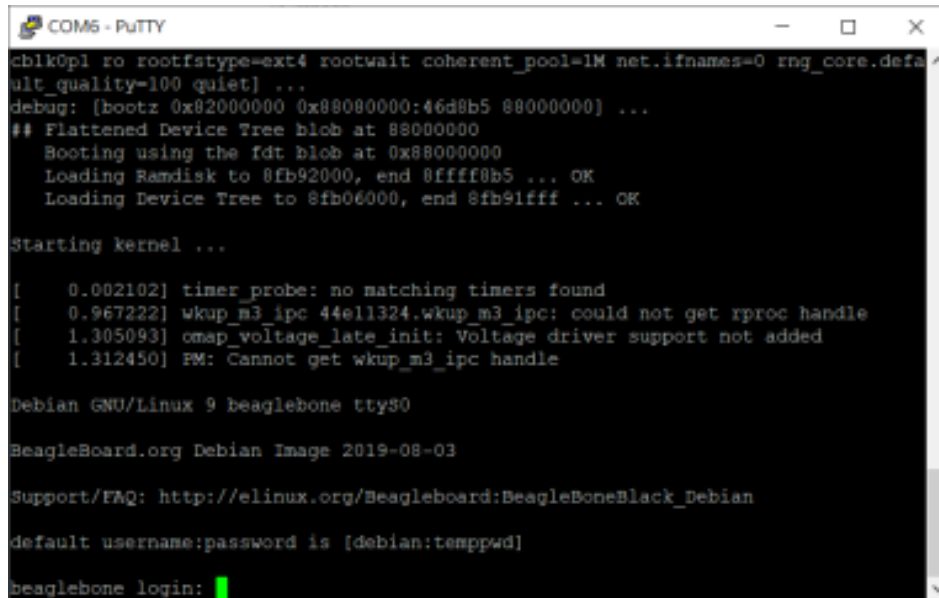


圖3-3

上電後靜待數秒進入登入畫面，輸入帳密登入，下圖2-4為範例登入畫面



```
COM6 - PuTTY
cb1k0pl ro rootfstype=ext4 rootwait coherent_pool=1M net.ifnames=0 rng_core.defa
ult_quality=100 quiet] ...
debug: [bootz 0x82000000 0x88080000:46d8b5 88000000] ...
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Loading Ramdisk to 8fb92000, end 8ffff8b5 ... OK
   Loading Device Tree to 8fb06000, end 8fb91fff ... OK

Starting kernel ...

[   0.002102] timer_probe: no matching timers found
[   0.967222] wkup_m3_ipc 44e11324.wkup_m3_ipc: could not get rproc handle
[   1.305093] omap_voltage_late_init: Voltage driver support not added
[   1.312450] PM: Cannot get wkup_m3_ipc handle

Debian GNU/Linux 9 beaglebone ttyS0
BeagleBoard.org Debian Image 2019-08-03
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
beaglebone login: █
```

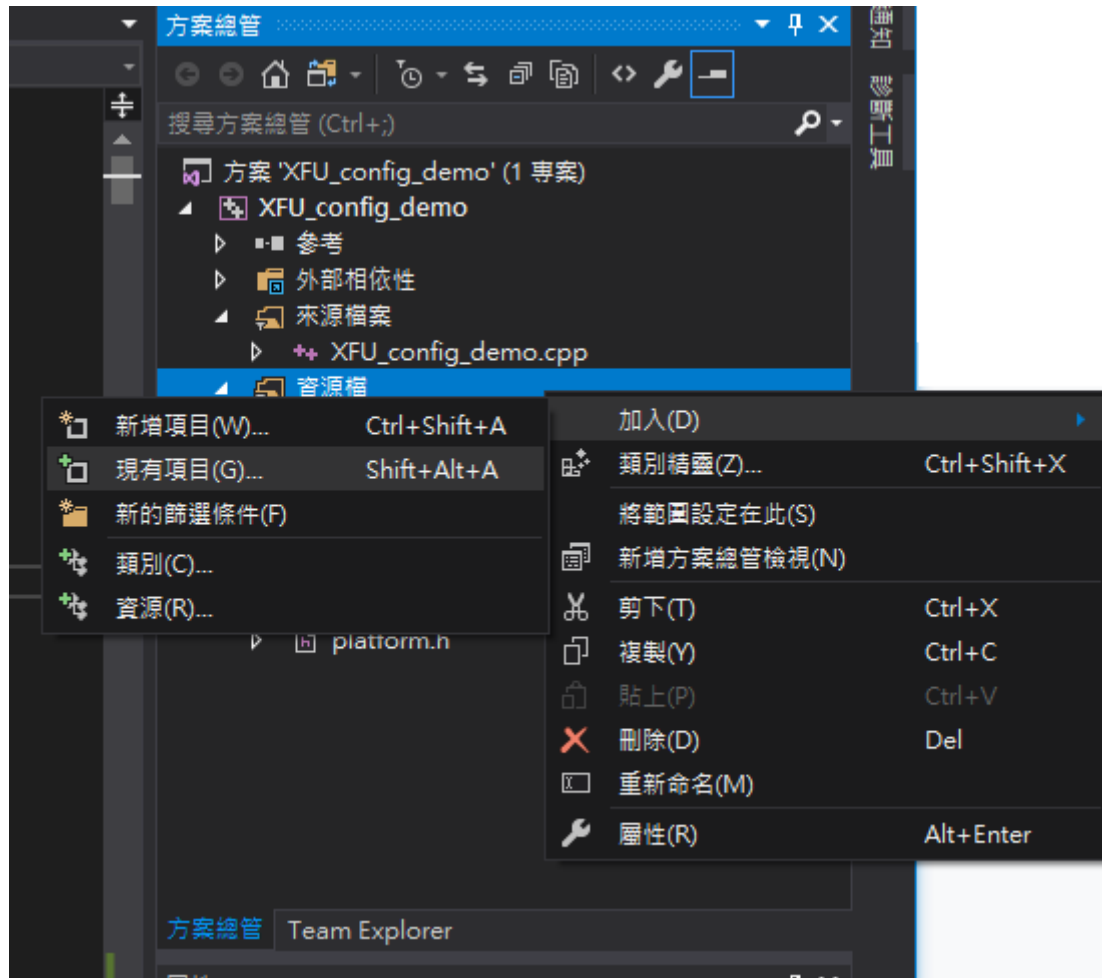
圖2-4

STEP4.下載ECM-XFU-SK驅動和Linux範例，安裝GCC後可以編譯範例程式

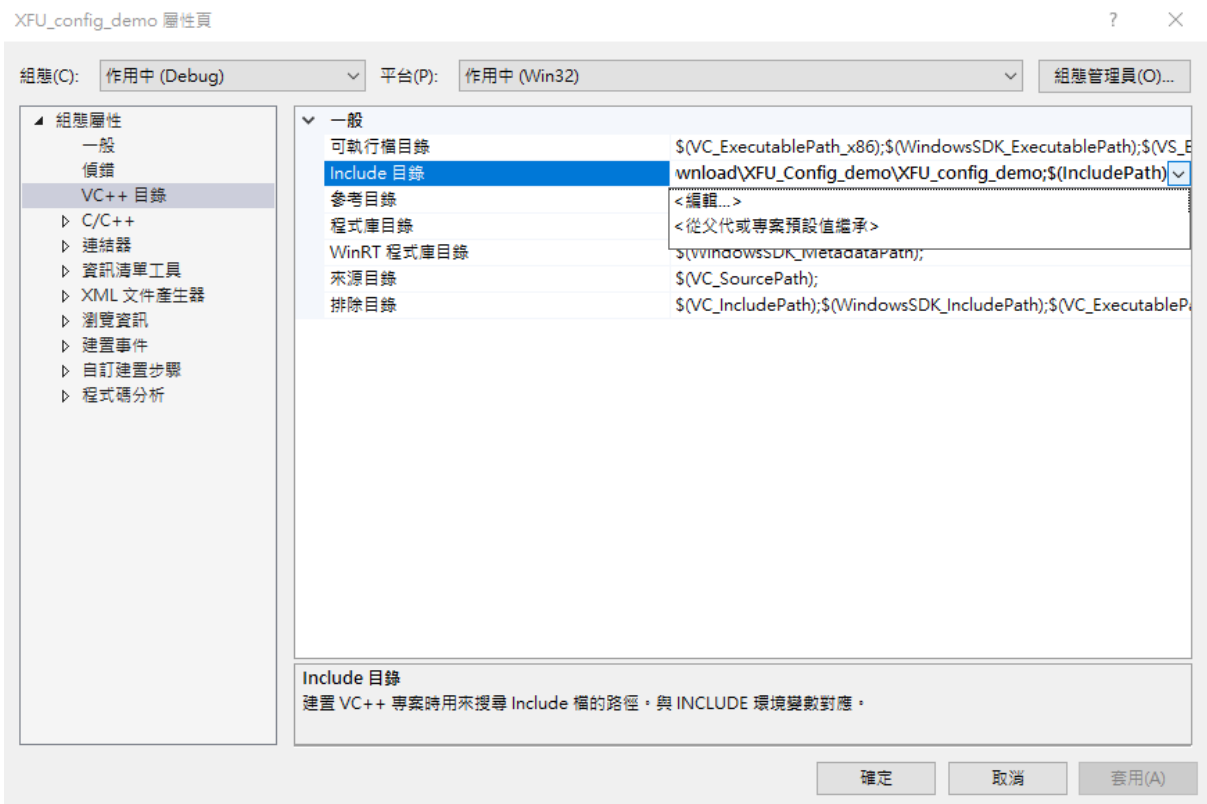
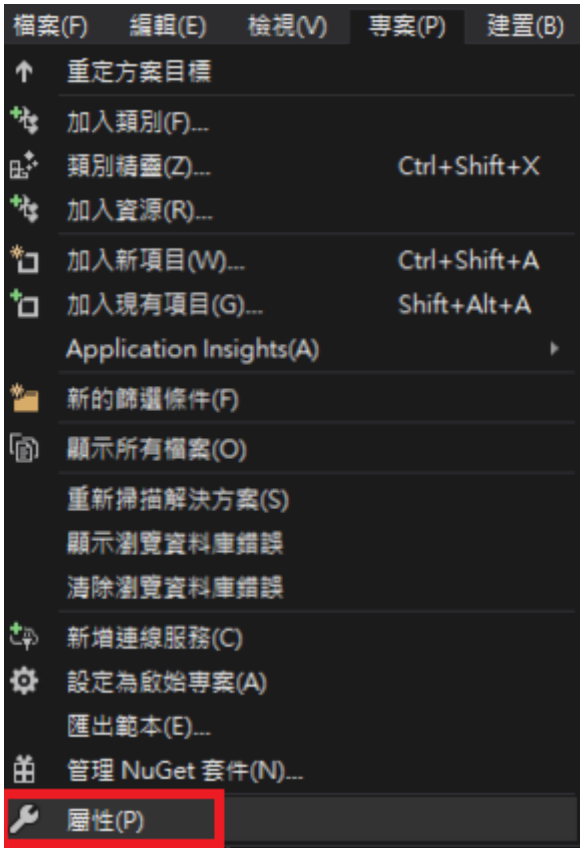
第 4 章 Windows Visual Studio 設置導引

4.1 Visual Studio C++

建立一個新的專案，並於專案中[加入][現有項目]，加入所有.h標頭檔



於專案 [屬性] [VC++ 目錄] 中加入相關函式庫名稱及位置



於[連結器] 加入相關的dll檔

XFU_config_demo 屬性頁

組態(C): 作用中 (Debug) 平台(P): 作用中 (Win32) 組態管理員(O)...

- 組態屬性
 - 一般
 - 偵錯
 - VC++ 目錄
 - C/C++
 - 連結器
 - 一般
 - 輸入
 - 資訊清單檔
 - 偵錯
 - 系統
 - 最佳化
 - 內嵌 IDL
 - Windows 中繼資料
 - 進階
 - 所有選項
 - 命令列
 - 資訊清單工具
 - XML 文件產生器
 - 瀏覽資訊
 - 建置事件
 - 自訂建置步驟
 - 程式碼分析

其他相依性 EcmXfUsbDrv.lib;kernel32.lib;user32.lib;gdi32.lib;winspool.lib

- 忽略所有預設程式庫
- 忽略特定的預設程式庫
- 模組定義檔
- 將模組加入至組件
- 內嵌受控資源檔
- 強制符號參考
- 延遲載入 DLL
- 組件連結資源

其他相依性
指定可加入連結命令列的其他項目 [例如 kernel32.lib]。

確定 取消 套用(A)


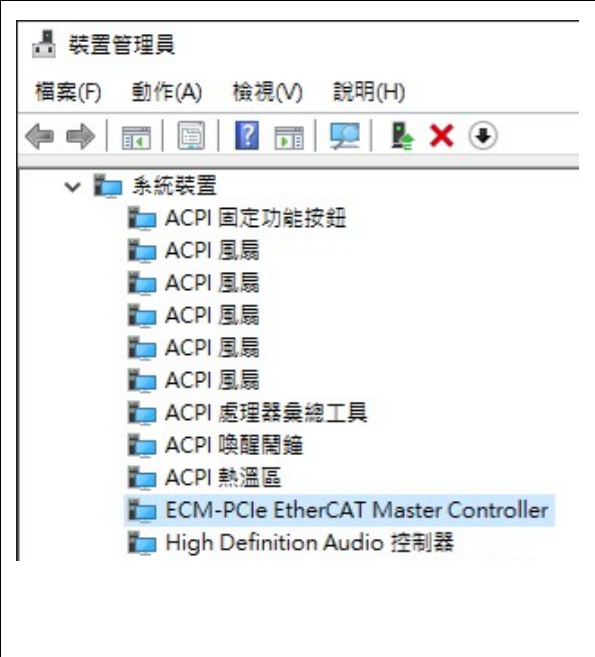
第 5 章 ECM-XFU ITE整合測試環境說明

5.1 ITE簡介

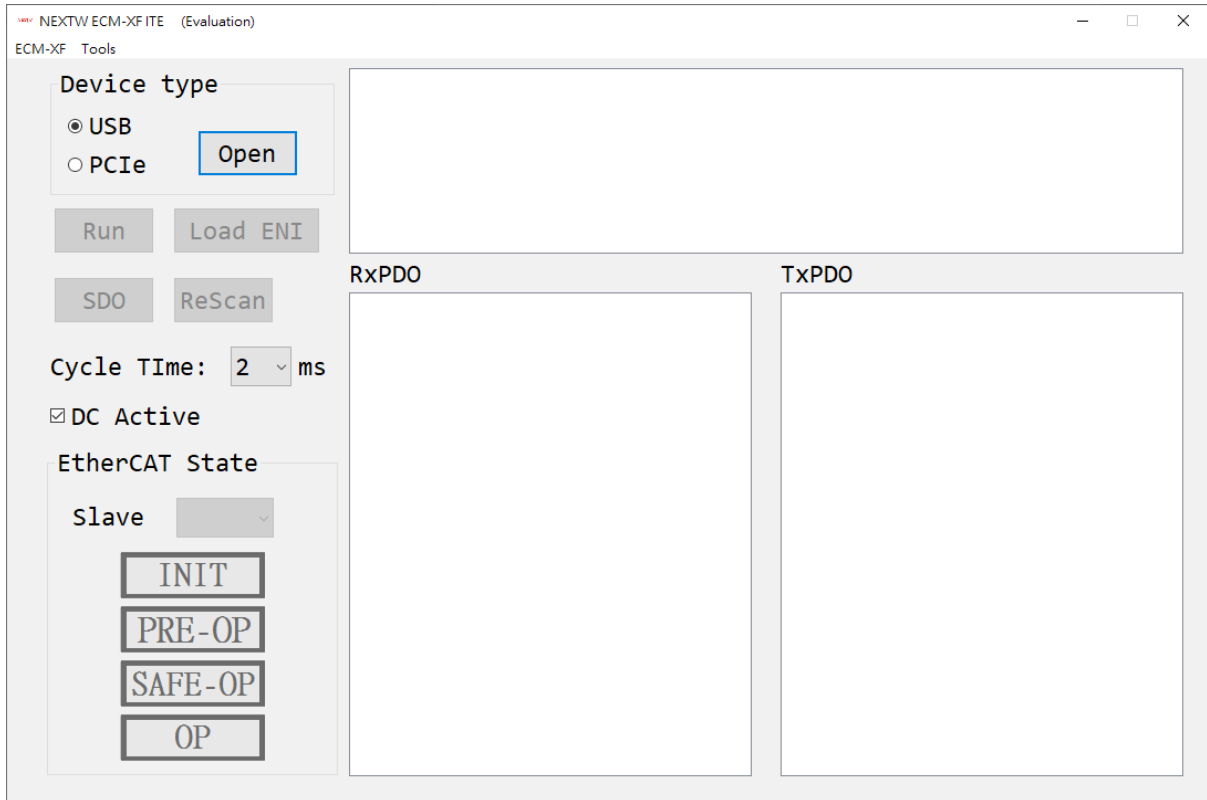
整合測試環境Integrated Test Environment, 簡稱ITE, 是可以透過USB直接讀取/寫入ECM-XFU-SK, 進而透過EtherCAT通訊協定控制從站

5.2 一般操作說明

ITE 支援 使用USB 或 PCIe 連接至電腦的ECM-XFU-SK/ECM-XF-PCIe裝置。
ECM-XFU-SK 透過USB連結到電腦後, 裝置管理員顯示「USB輸入裝置」及「符合HID標準的廠商定義裝置」(HID-compliant device)。PCIe卡裝到電腦後, 裝置管理員則顯示「ECM-PCIe EtherCAT Master Controller」

	
<p>ECM-XFU-SK模組</p>	<p>ECM-XF-PCIe裝置</p>

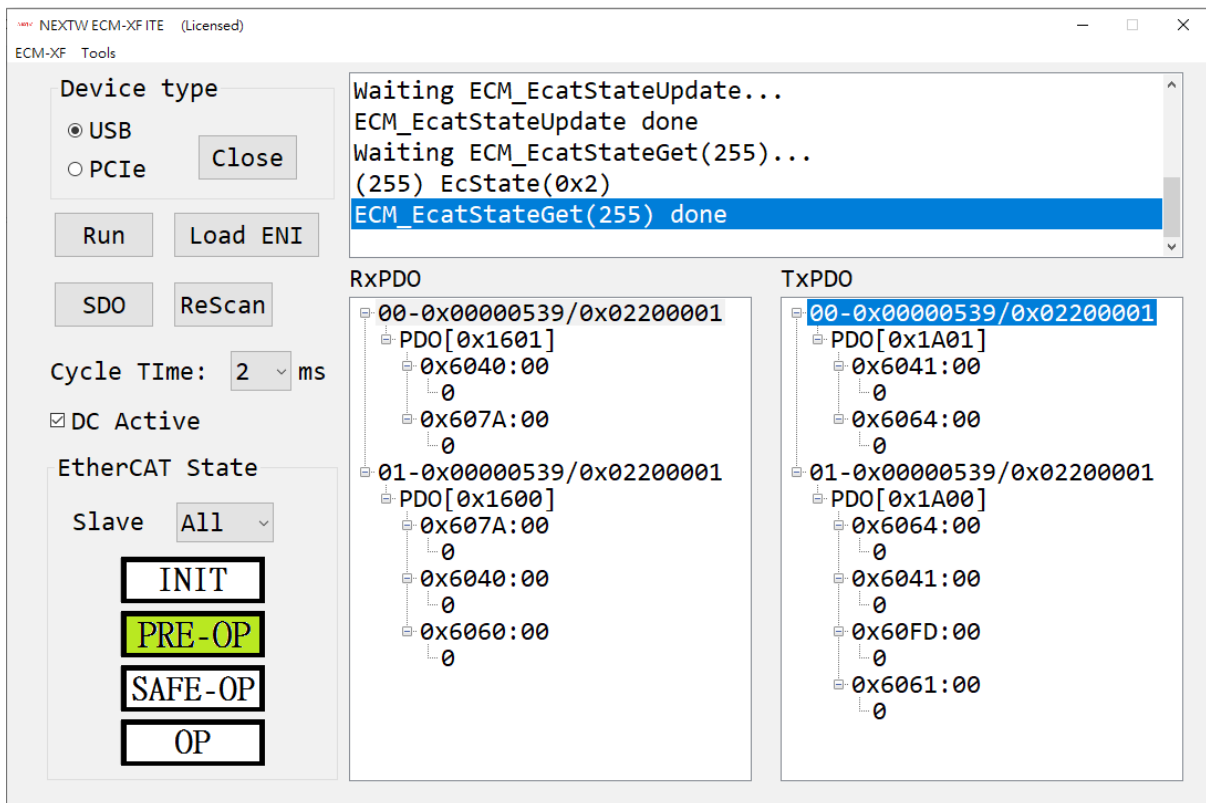
打開ECM-XF ITE軟體，於左上角Device type選擇連線方式（USB或PCIe），確認選擇正確後，按下“Open”按鈕。



打開連接後，左下角狀態應該為Pre-OP狀態。右上角的文字框將顯示 訊息(如 IC 韌體版本、執行過程訊息等)。從站的預設的設置將顯示在 RxPDO 和 TxPDO 文字框中。

前面的00、01代表從站順序編號，後面則代表該從站的製造商編號 (Vendor ID) 及產品編號(Product ID)，透過點擊 RxPDO 和 TxPDO 文字框中內容前面的“+”符號，會顯示該從站的詳細的PDO內容。

下圖為連結2台的 YASKAWA 驅動器的預設配置。

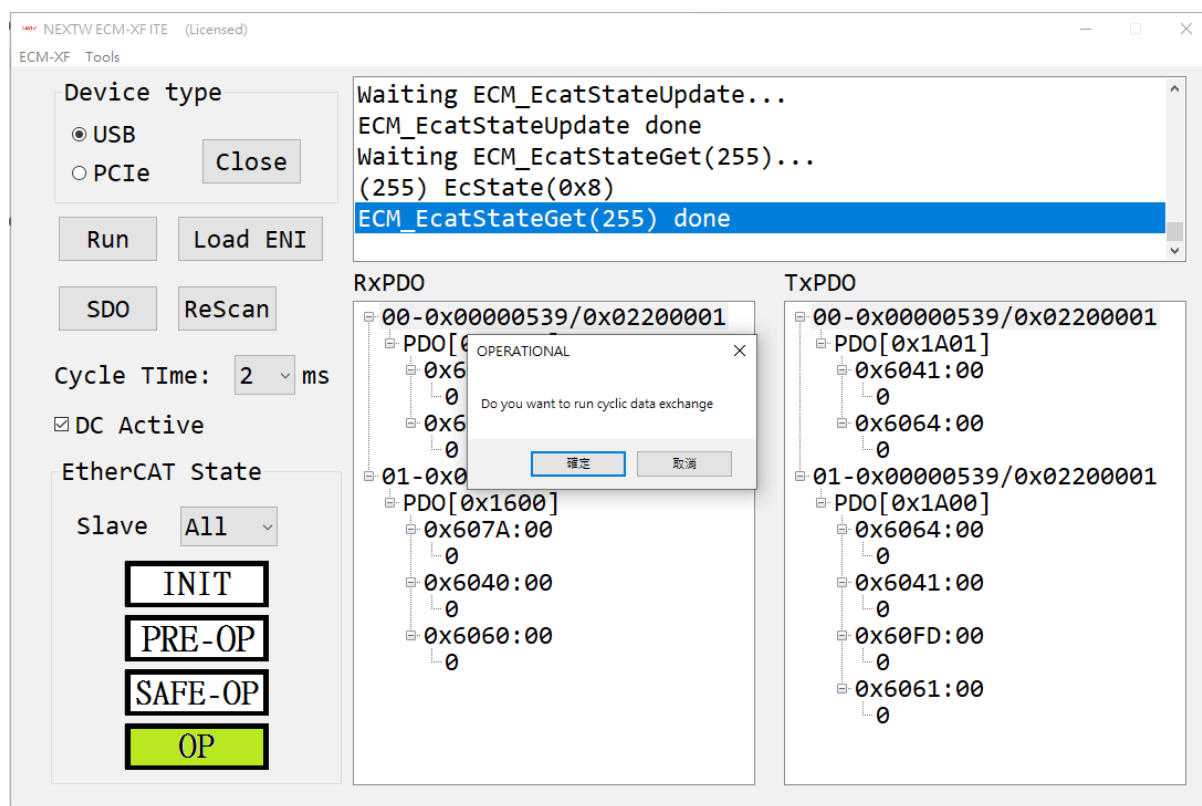


在 Pre-OP 狀態下，可以進行讀/寫 SDO、使用 ENI 中的配置，或是使用 Tools 中的 SII editor 和 code generator。

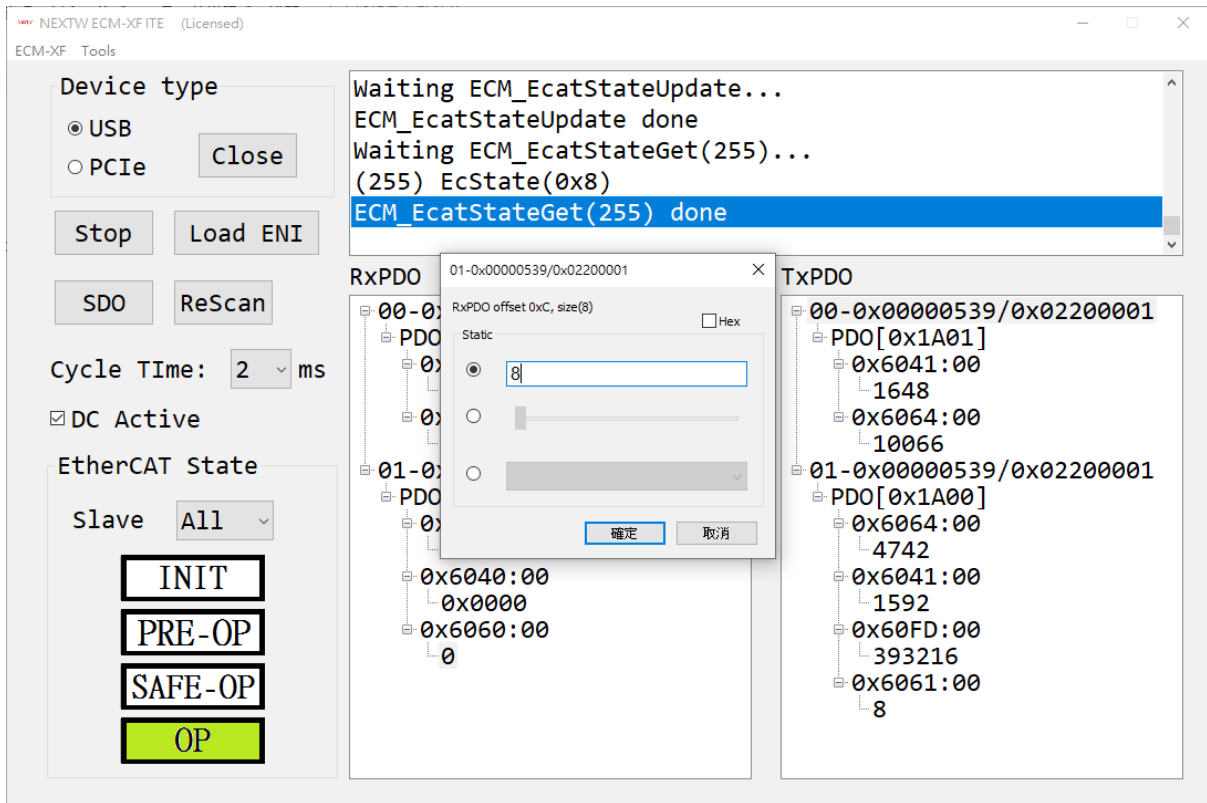
左下角可改變從站的狀態，欲切至 OP 狀態，必須依照 Init->PREOP->SafeOP->OP 依序切換，於 OP 狀態可返回之前任一狀態。需要注意狀態變更的順序(從上到下)，否則切換狀態會失敗。若切換狀態失敗，可參考訊息區的 AL Status Code，以瞭解失敗原因。常見由 PreOP 切換至 SafeOP 失敗的原因在於 (1) Cycle Time 設定不支援、(2) DC 設定不支援、(3) RxPDO 或 TxPDO 設定不支援、(4) Slave EEPROM 設定數值有誤、(5) 其他。常見由 SafeOP 切換至 OP 失敗的原因為 DC 尚未穩定或其他因素。

當進入 OP 狀態時，將會彈出視窗顯示是否要進行 cyclic data exchange 的詢問視窗，選擇 [確定]，即開始與從站進行實際的週期性資料交換，若選擇 [取消]，使用介面上的數

值並不會與從站進行資料交換。按下左邊的[Run]按鈕則會開始進行週期性資料交換，再按下[Stop]則會停止將介面上的數值與從站進行週期性資料交換。

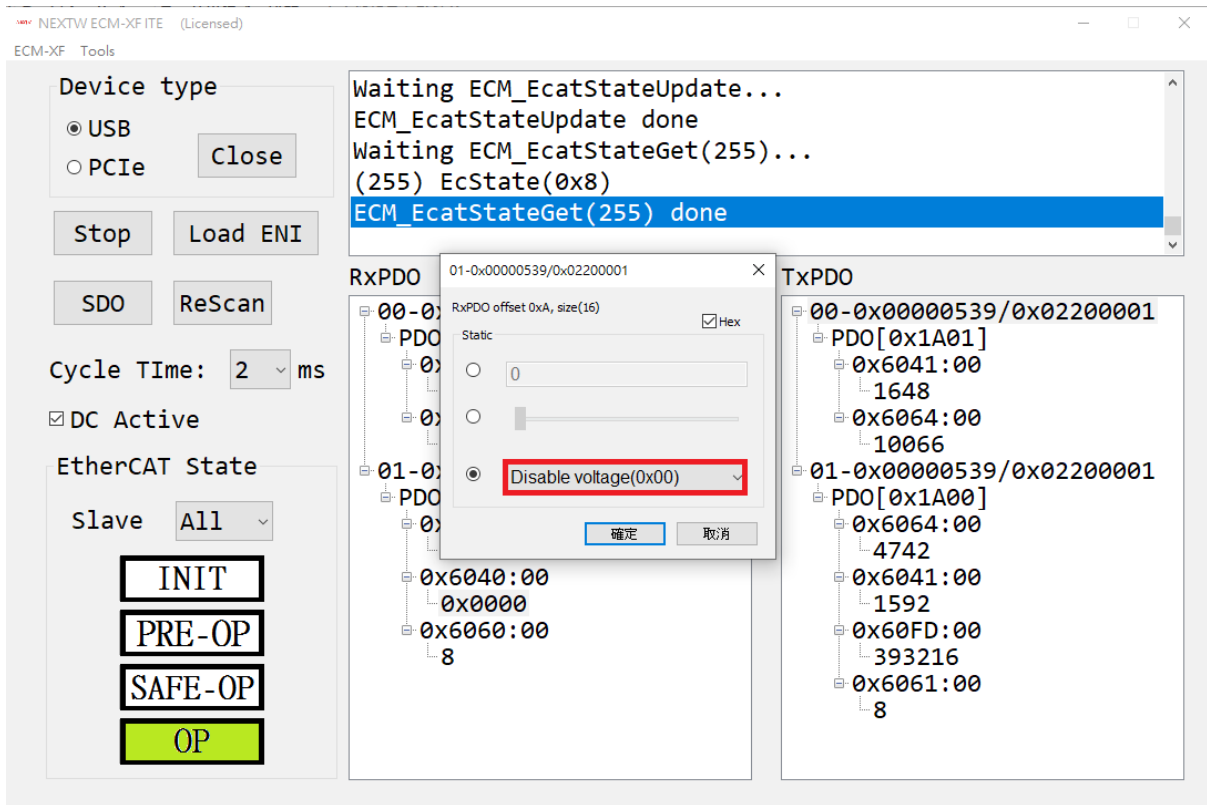


RxPDO為主站送至從站的命令，在OP模式下，可變更RxPDO內各Object的內容值。TxPDO為從站送至主站的狀態，下圖顯示2個從站的當前狀態。如果需要，可以右鍵單擊Hex旁的方框以切換為十六進制。而在RxPDO雙擊值的部分可以變更數值。



雙擊RxPDO中 0x6040 (Control Word)可以改變運行模式下的 402 狀態機。但不需要每次都按 [確定]來改變狀態。可以觀察到TxPDO中0x6041 (Status Word)發生變化。

針對伺服馬達若要激磁 (Servo on)，可透過改變0x6040的數值，依序選擇“Shutdown(0x6)”、“Switch On(0x7)”、“Enable Operation(0xF)”，切至 Enable Operation後馬達即會激磁，詳細的說明可參考從站關於0x6040的說明。

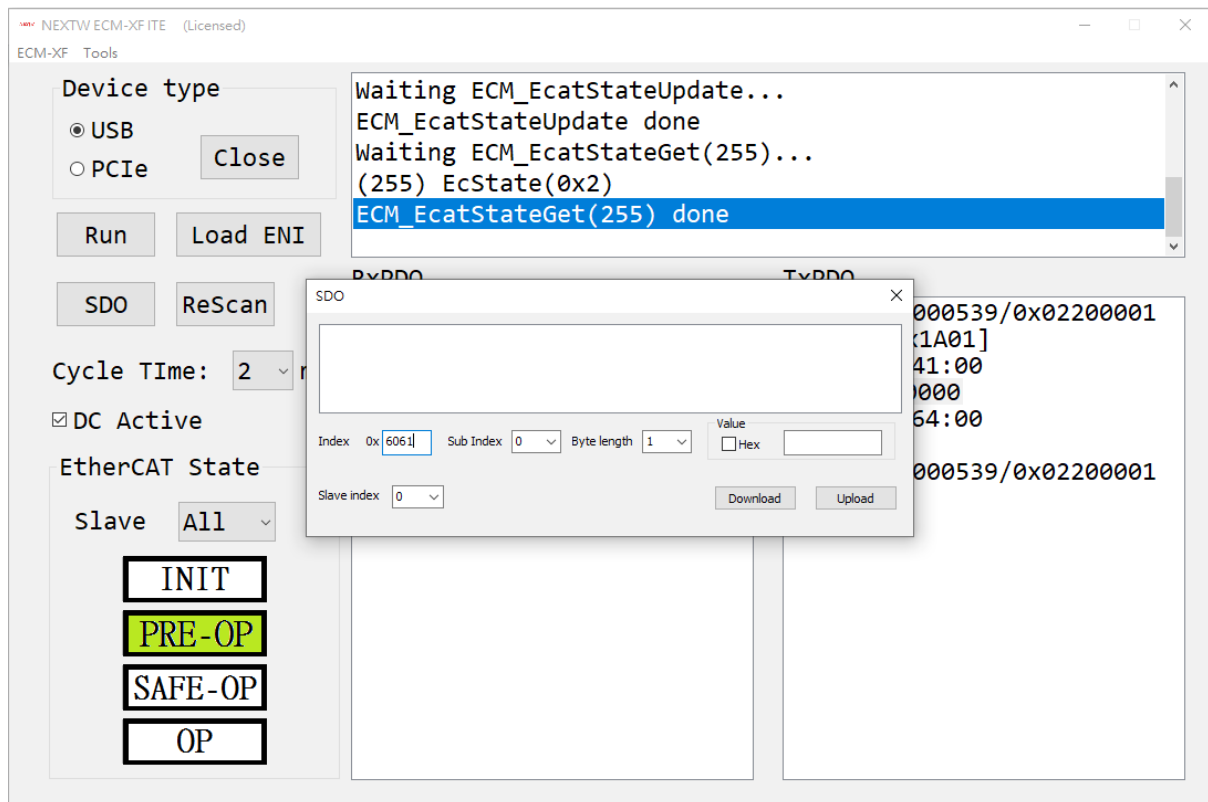


在操作模式為CSP的情況下(即Object 0x6060:0 設定為8)，驅動器會於每個週期嘗試轉動馬達至目標位置(0x607A)，而回傳目前的實際位置(0x6064)。激磁(Servo on)後 RxPDO中的0x6064 (Position actual value)會與TxPDO中的0x607A (Target position) 值相同。接著可以雙擊 0x607A (Target position)設置下一個週期時間的新目標位置。亦可使用拉bar來改變數值。

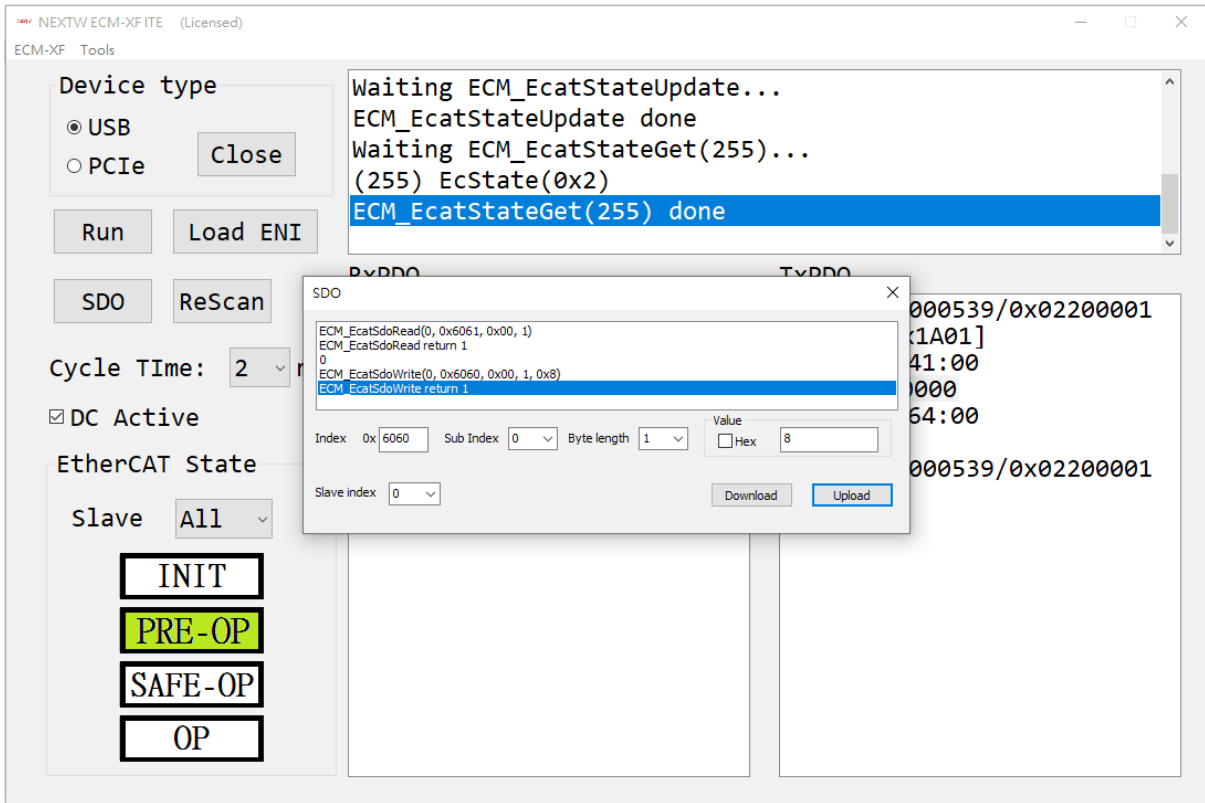
更多的運行模式請參閱從站的使用手冊關於Object 0x6060:0 的說明。

5.3 SDO操作說明

SDO是非週期性資料交換，在PRE-OP、SAFE-OP及OP階段均可進行，按下[SDO]按鈕後會出現SDO視窗。例如，欲讀取運行模式(Modes of Operation Display)，可讀取Object Index為0x6061，Sub Index為0，Byte length(Size) 為1，然後按下[Read]即可於右側Value地方讀到數值。更多關於Object的定義請參閱從站的使用手冊。

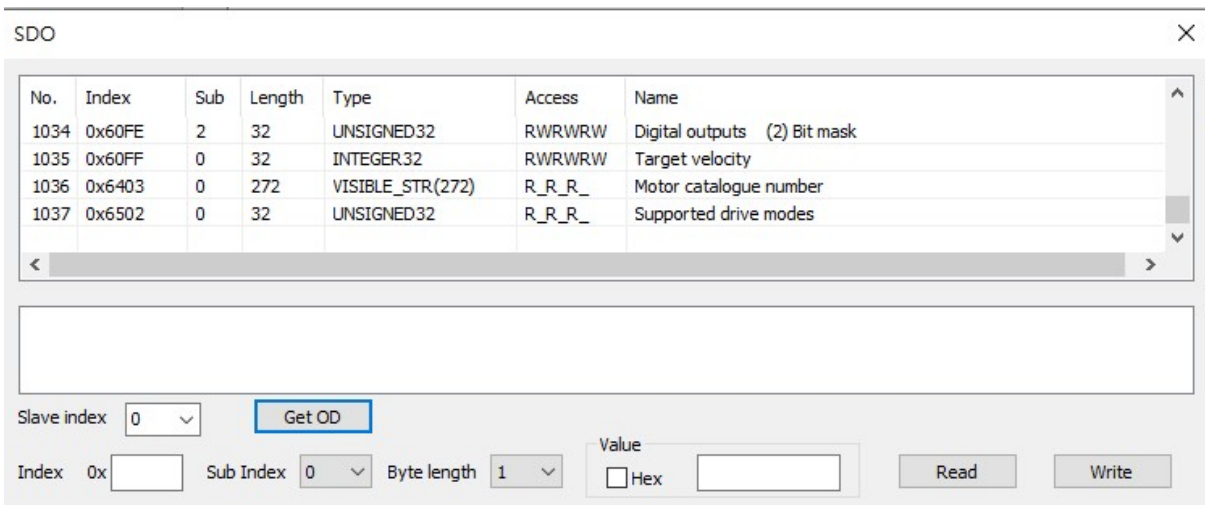


若欲透過SDO機制寫入Object，則除了Index、Sub Index、Byte length均需填寫外，Value亦需填寫。例如欲寫入運行模式(Modes of Operation)，可寫入Object Index為0x6060，Sub Index為0，Byte length(Size) 為1，Value則為欲寫入的值，然後按下[Write]即會寫入。



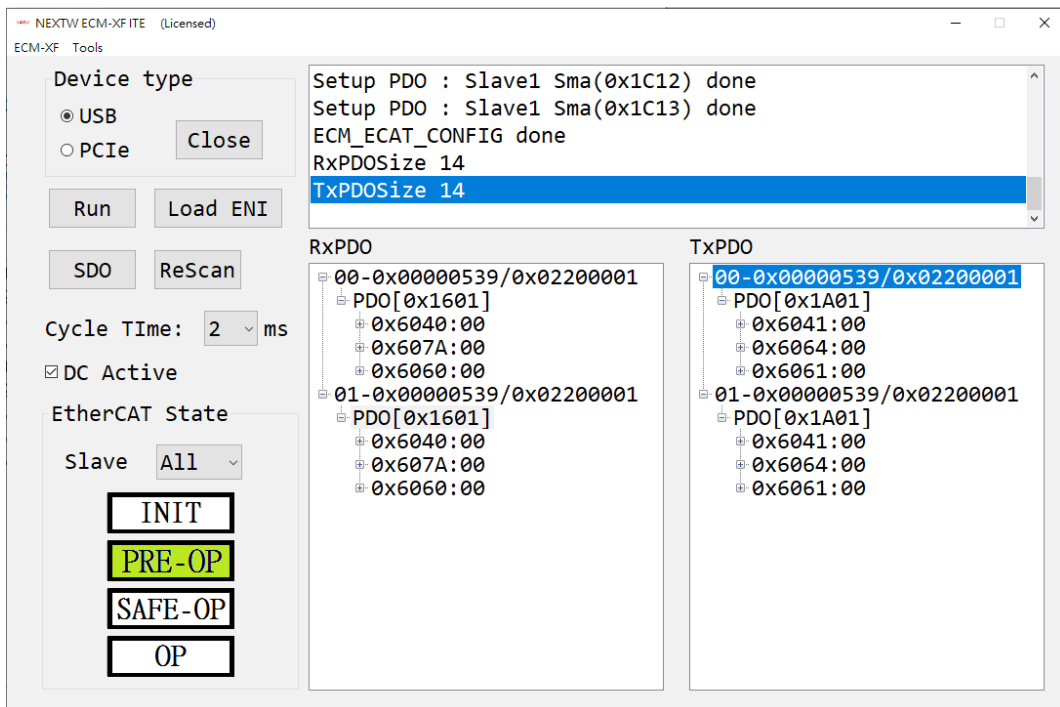
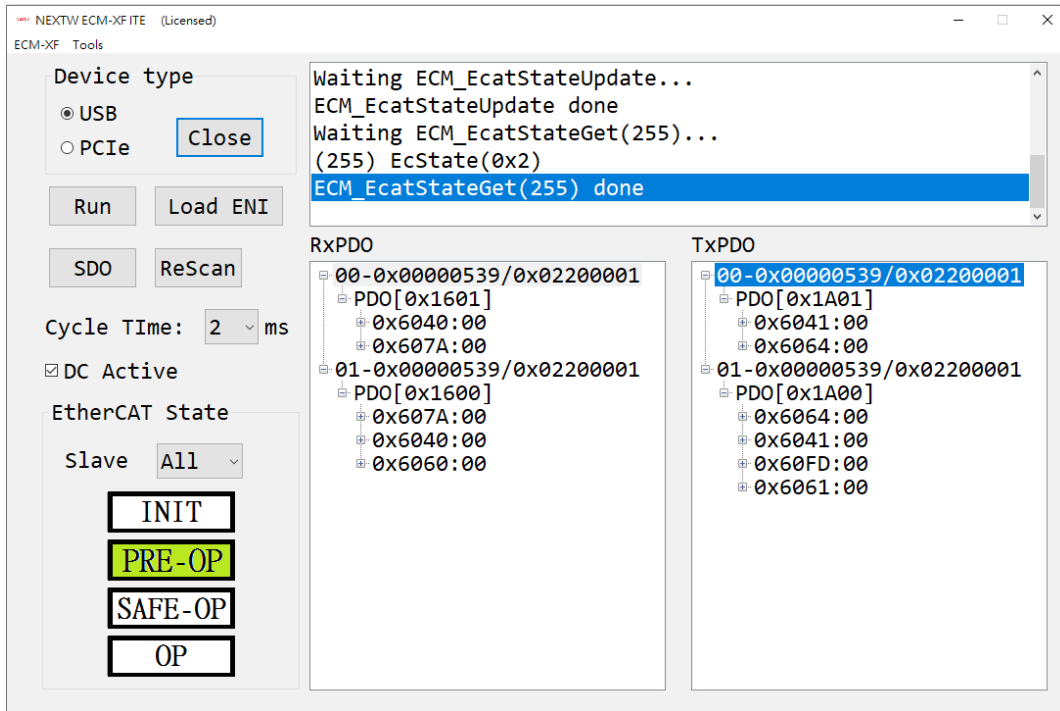
若欲確認是否正確寫入，則可使用SDO Read讀取該數值。

按下[Get OD]按鈕，等待數秒後，可自從站讀回所有的Object, [Access]顯示R代表可讀，W代表可寫，前兩位文字代表於PerOP階段的讀寫狀態，中間兩文字代表於SafeOP階段的讀寫狀態，最後兩文字代表於OP階段的讀寫狀態。請注意，列出之資訊係由從站提供，主站無法驗證其正確性，建議仍應參考從站手冊的說明。可選擇欲讀寫的Object，其資料會自動帶入至下方欄位。



5.4 透過Load ENI修改RxPDO及TxPDO配置說明

ITE支援讀取 ENI (EtherCAT Network Information)內的 PDO 配置。按下[Open]後再按下[Load ENI]按鈕，選擇ENI文件，ITE會根據所選的ENI文件嘗試配置從站PDO結構。下圖顯示 RxPDO 和 TxPDO 經過ENI檔案中設置更改實體配置。



5.5 透過SDO修改RxPDO及TxPDO配置說明

若無ENI檔案亦可在PRE-OP階段自行透過SDO寫入0x1C12、0x1C13的方式來配置PDO，例如欲設定RxPDO的Sync Manager PDO assign objects (0x1C12)設定其PDO mapping objects 為0x1601，詳細的作法及步驟如下：

1. SDO Write 0x1C12:0 (size 1) Value = 0
2. SDO Read 0x1C12:0 (size 1) 確認為 Value = 0
3. SDO Read 0x1C12:1 (size 2) 確認預設值(例如0x1600)
4. SDO Write 0x1C12:1 (size 2) Value為欲設定的PDO值(例如0x1601)
5. SDO Read 0x1C12:1 (size 2) 確認設定是否正確(如0x1601)
6. SDO Write 0x1C12:0 (size 1) Value= 1 (PDO mapping objects的數量)
7. SDO Read 0x1C12:0 (size 1) 確認Value為 1
8. 按下[ReScan]

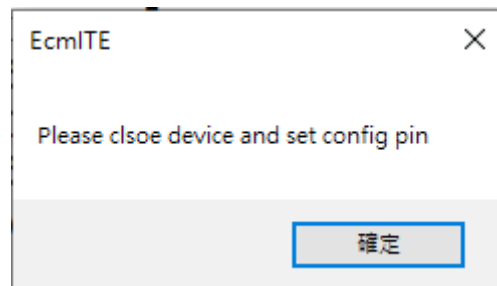
若欲進一步改變PDO mapping objects中的內容mapping entry，則可在上述2和3中間加做以下步驟：(如欲將0x1601設有2個mapping，分別為0x6040:0 size 2 bytes及0x607A:0 size 4 bytes)

- 1 SDO Write 0x1601:0 (size 1) Value = 0
- 2 SDO Read 0x1601:0 (size 1) 確認為 Value = 0
- 3 SDO Write 0x1601:1 (size 4) Value = 0x60400010，其中bit0-7為mapped object的長度(單位bit)，bit8-15為mapped object的sub Index，bit16-31為mapped object的Index。
- 4 SDO Read 0x1601:1 (size 4) 確認Value = 0x60400010。
- 5 SDO Write 0x1601:2 (size 4) Value = 0x607A0020，其中bit0-7為mapped object的長度(單位bit)，bit8-15為mapped object的sub Index，bit16-31為mapped object的Index
- 6 SDO Read 0x1601:2 (size 4) 確認Value = 0x607A0020。
- 7 SDO Write 0x1601:0 (size 1) Value = 2(mapping entry的數量)

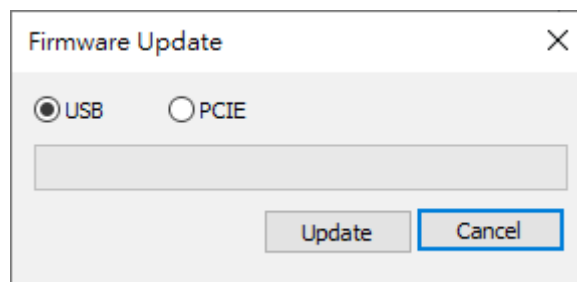
8 SDO Read 0x1601:0 (size 1) 確認為 Value = 2

5.6 更新韌體 (Update firmware)

ITE 支援了 ECM-XF/ECM-XFU 的韌體更新。請先將硬體模式切換到 update mode 並單擊工具欄“ECM-XF”，然後選擇“FW update”。update mode 詳情請參見硬體板的手冊，以 ECM-XF-SK V1.2 為例，須使用跳線帽將 J3 中間針腳與 L 針腳短路即為 update mode。如果您的開發板未處於 update mode，會出現如下圖所示的視窗。

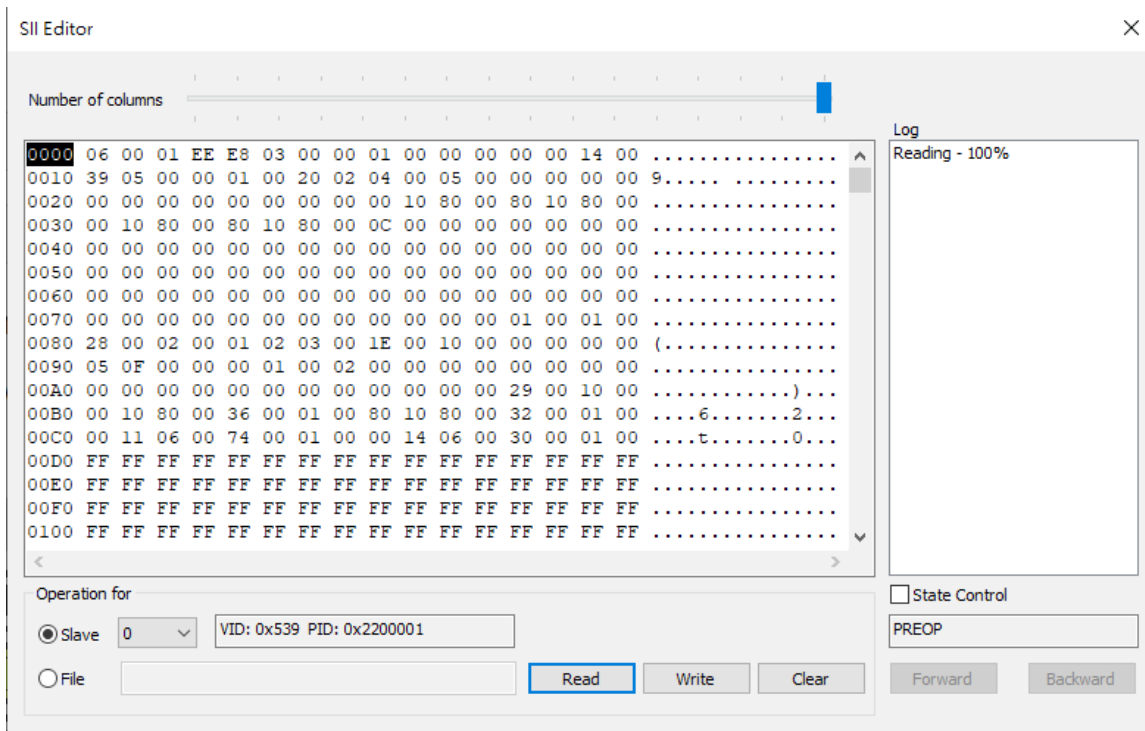
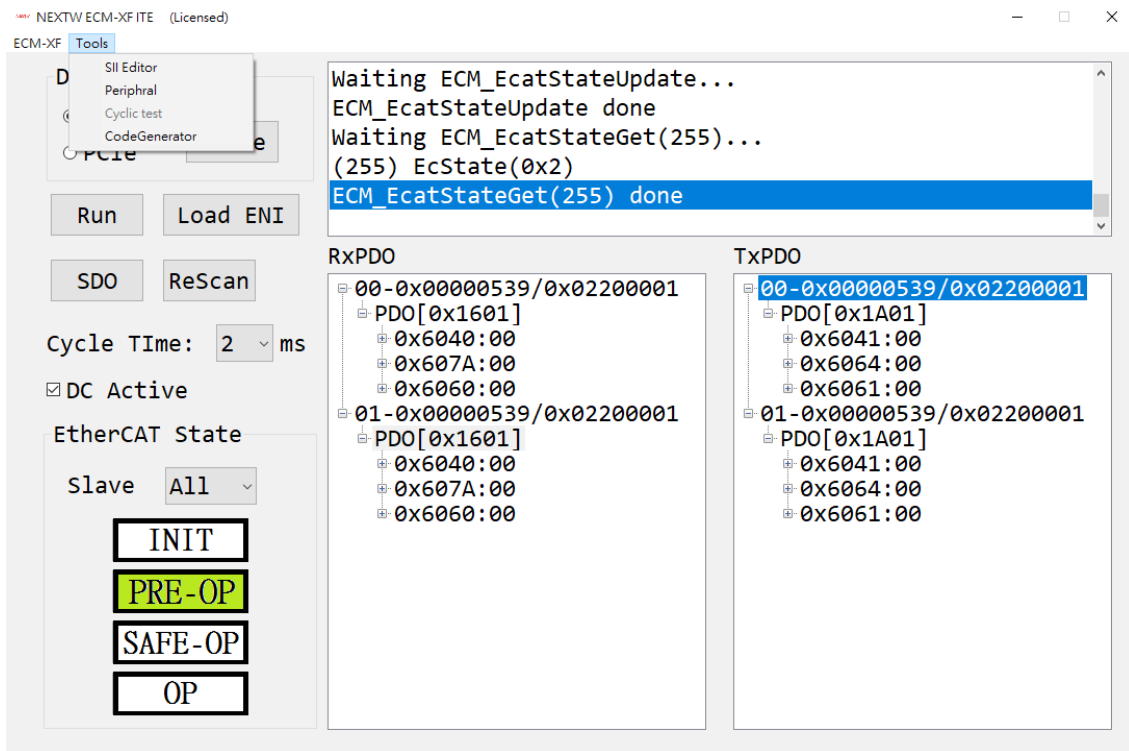


如果您處於 update mode，則在按下 [FW update] 後可選擇連結介面 (USB 或 PCIe)，之後再選擇更新檔案即可。請注意，請務必確認所選的更新檔案是正確的，錯誤的更新檔可能導致無法正常運作，且無法復原。



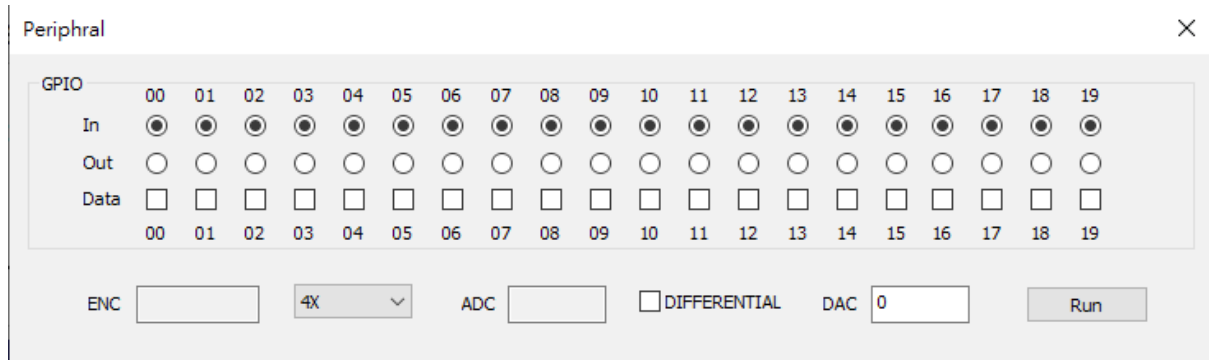
5.7 SII editor

Tools 中的[SII Editor], 可讀取從站 SII文件和並存成檔案, 或讀取檔案並寫入從站。請注意必須 Pre-OP 狀態下使用此editor。



5.8 XFU週邊IO (Peripheral)

Peripheral的視窗可控制或顯示 ECM-XFU / ECM-XF-PCIe GPIO/QEI/ADC 的功能。



5.9 評估版 與 授權版

左上方視窗標題欄中顯示Evaluation為評估版本，評估版本可切換狀態、讀取和寫入從站PDO、與從站進行SDO操作、韌體升級等功能。

正式版本可操作完整的功能，包含Code Generator等。

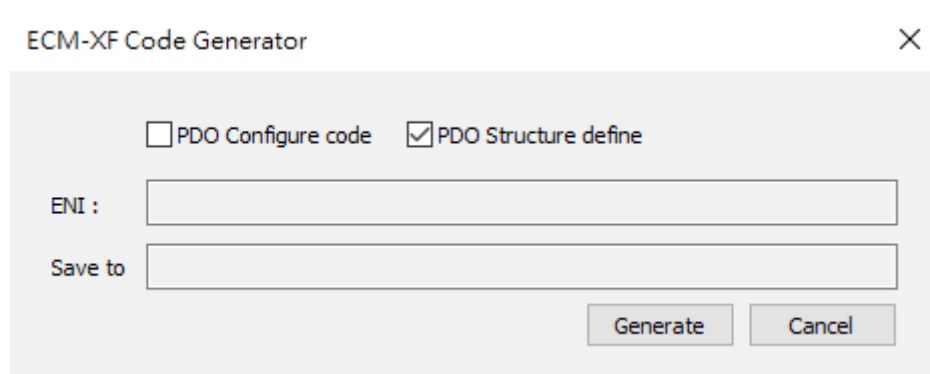
正式版本於完成付費後點擊[ECM-XF]選單內[License]，會有詢問是否已取得License視窗，點選[取消]，會產生“activation.lis”文件，將此文件連同ECM-XFU-SK或ECM-XF-PCIe序號寄至 service@nextw.com.tw，將可取得專屬的授權檔案。

獲得授權後，將授權文件“EcmXF.lis”放入ITE執行檔 的同一文件夾中。然後單擊工具欄上的 [ECM-XF]中[License]，於詢問視窗中選擇[確定]，選擇授權文件並單擊打開。左上方的視窗將顯示“Licensed”表示已升級為正式版。

5.10 代碼產生工具(Code Generator)

代碼產生工具必須是Lincese版本才能使用，可讀取 ENI文件(EtherCAT Network Information File)，並生成配置 c 文件(PDO Configure code)，此文件可作為撰寫程式時的配置參考。

代碼產生工具亦可將當前的 PDO 結構定義(PDO Structure define)另存為 .h 文件，此 PDO結構定義以當下實際的PDO定義為主，您可先使用[Load ENI] 自動配置ENI內的定義，或是使用SDO讀寫0x1C12、0x1C13後再生成 .h 文件之前，此文件可作為撰寫程式時的結構定義參考。



配置參考程式(.c 文件)和 PDO 結構定義的參考文件(.h 文件)可以放入您自行撰寫的應用程式中。

以下為簡單的的導入說明將 c 文件函式 ECM_ConfigPDO_CodeGen(void) 複製到 main.c 中，如下圖所示。

```
#define ECM_TBL_CNT_CC 4

int ECM_ConfigPDO_CodeGen(void)
{
    int nDone = 0;
    int nRet = 0;
    PDO_CONFIG_HEAD sEcmPdoTbl[ECM_TBL_CNT_CC] = {
        {0x539,0x2200001,0,1,0x1C12,{0x1601,0x0,0x0,0x0},{3,0,0,0},{16,0,0x6040},{32,0,0x607A},{8,0,0x6060}},
        {0x539,0x2200001,0,1,0x1C13,{0x1A01,0x0,0x0,0x0},{3,0,0,0},{16,0,0x6041},{32,0,0x6064},{8,0,0x6061}},
        {0x539,0x2200001,1,1,0x1C12,{0x1601,0x0,0x0,0x0},{3,0,0,0},{16,0,0x6040},{32,0,0x607A},{8,0,0x6060}},
        {0x539,0x2200001,1,1,0x1C13,{0x1A01,0x0,0x0,0x0},{3,0,0,0},{16,0,0x6041},{32,0,0x6064},{8,0,0x6061}} };
    for (int i = 0; i < ECM_TBL_CNT_CC; i++) {
        if (sEcmPdoTbl[i].PDOCnt) {
            nRet = ECM_EcatSdoSetPdoConfig(&sEcmPdoTbl[i]);
            if (nRet == 1) {
                nDone++;
            }
        }
    }
    return nDone;
}

int main()
{
    uint16_t u16SpiDataSize = 992; //fixed in USB
    uint8_t u8Version;

    PDO_CONFIG_HEAD RxPDOConfig[2];
    PDO_CONFIG_HEAD TxPDOConfig[2];
    uint8_t RxData[TEST_SPI_DATA_SIZE] = { 0 };
    uint8_t TxData[TEST_SPI_DATA_SIZE] = { 0 };
}
```

在配置部分只需使用函數“ECM_ConfigPDO_CodeGen()”

```
nret = ECM_StateCheck(0xFF, EC_STATE_PRE_OP, 10000); // Set mode must be at PRE-OP state

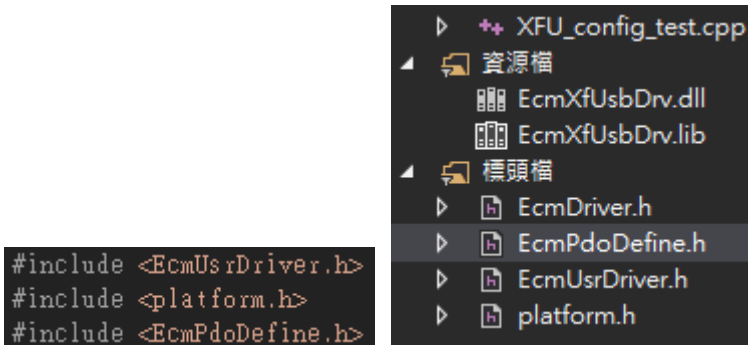
if (nret == 0) {
    return -1;
}

std::cout << "All Slaves are in PRE-OP state\n";
std::cout << "Assign and configure PDO\n";

ECM_ConfigPDO_CodeGen();

// Call ECM_EcatReconfig() after setting PDO configures
nret = ECM_EcatReconfig();
if (nret < 0) {
    std::cout << "ECM_ECANT_CONFIG error : " << nret << "\n";
    return -1;
}
```

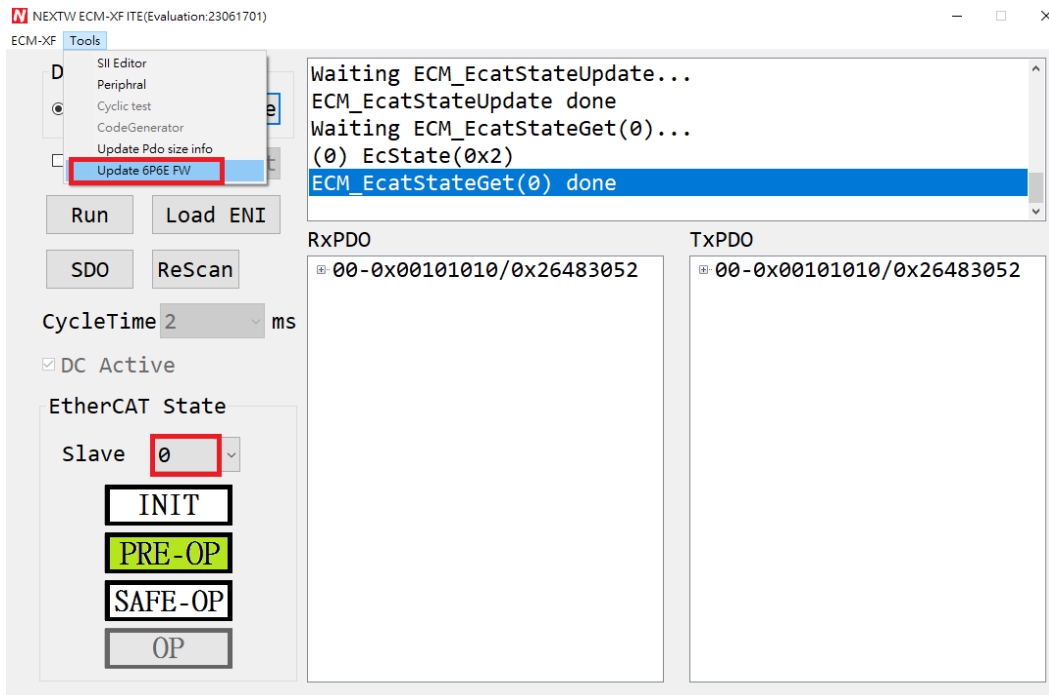
EcmPdoDefine.h 是與 SPI 版本中的 PdoDefine.h 相同的功能文件。在main.c中使用 #include <EcmPdoDefine.h> 即可。



5.11 FoE (File over EtherCAT)

ITE可以利用FoE技術透過EtherCAT傳送檔案至從站，可透過ITE更新ES-6P6E從站的韌體，請注意，從站將辨識檔案的正確性，僅正確的檔案才會用於更新從站的韌體。

請先選定從站編號(一次僅能選定一個從站)，再按下Update 6P6E FW，選擇欲更新的檔案(ECATFW__)，按下確認後即可。傳輸時間依檔案大小而定，一般需數十秒，請耐心等待。



* 若針對其他從站進行FoE傳輸，請注意從站是否支援FoE，且檔案需小於256KB

第 6 章 ECM-XFU 範例函數說明

6.1 ECMXFUDRV簡介

ECMXFUDRV驅動函式庫支援ECM裝置(ECM-XF PCIe / ECM-XFU USB), 使用者可在Windows平台上開發EtherCAT應用, 進而開發PC-base控制器。

ECM裝置皆具有ECM-XF晶片(從端), 提供PCIe或USB介面與PC連接(主端), 為主從通訊的架構, ECM-XFU交換資料封包大小為1024位元組(ECM-XF IC預設為144位元組, 資料長度可變更), 由PC傳送給ECM-XF/XFU的資料封包稱為命令封包, 由ECM-XF/XFU回傳給PC的資料封包稱為回應封包。

函式庫內具有兩個大小為1024位元組的全域變數分別代表命令封包及回應封包, 函式皆透過全域變數pCmd指標存取命令封包, 及全域變數pRet存取回應封包。

ECMXFUDRV的原始程式碼可於以下網址查閱與下載:

<https://bitbucket.org/x-force/nextwecmxfudrv/src/master/>

6.2 ECMXFUDRV API介紹

EcmUsrDriver.cpp		
int32_t ECM_PrintErrStatus(int32_t nCleanFlag)		
印出錯誤名稱, 並根據參數nCleanFlag決定是否清除錯誤		
返回值	-1	有錯誤, 並依據參數nCleanFlag決定是否清除錯誤
	1	無錯誤
參數	int32_t nCleanFlag	0: 不清除錯誤 非零: 清除所有錯誤

EcmUsrDriver.cpp		
int32_t EcmSpiPackSizeCal()		
透過頭碼尾碼計算SPIDataSize		
返回值	0	沒有偵測到頭尾碼, 請檢查SPI傳輸
	>0	SPI Data Size

EcmUsrDriver.cpp		
int32_t SpiDataExchange(uint8_t *RetIdx, uint8_t *RetCmd)		
上位機(PC)透過PCIe To SPI或USB介面與ECMXF進行資料交換, 主端將會送出一筆命令的長度(XFU:1024位元組 / XF 預設144位元組, 資料長度可變更) 資料並取回從端回傳的資料。		
返回值	1	正確
	0	CRC錯誤
	<0	CRC連續錯誤次數達設定值
參數	uint8_t *RetIdx	返回資料的索引號
	uint8_t *RetCmd	返回資料的命令碼

EcmUsrDriver.cpp		
int32_t EcmCmdTransceiver(int32_t nTryCnt)		
發送命令並接收回應, 將會送出命令並取回該命令的回應。 呼叫本函式前應先將pCmd的內容填上必要欄位後, 本函式會透過SpiDataExchange交換資料, 取回來的資料會存放在pRet指標所指的空間。 驅動函式庫內函式大部分都是基於本函式實現, 實現流程如下:		
<ol style="list-style-type: none"> 1. 透過pCmd將命令封包的命令碼及必要欄位填上 2. 呼叫EcmCmdTransceiver給定錯誤重傳次數 3. 判斷返回值是否正確取得回應封包 4. 透過pRet將回應封包資料取回 		
返回值	1	成功
	0	超時
參數	int32_t nTryCnt	錯誤嘗試次數

EcmUsrDriver.cpp		
int32_t EcmCmdTransceiver_WaitAyncDone(int32_t nTryCnt, uint16_t *pu16ReturnDataSize, uint8_t *pu8ReturnData)		
發送命令並接收回應, 將會送出命令並取回該命令的回應後, 等待非同步命令完成後才返回。		

本函式是由EcmCmdTransceiver為基礎，當EcmCmdTransceiver返回後，會輪詢ECM-XF等待非同步命令完成或超時後返回。

返回值	1	成功
	0	超時
	-1	資料成功取回但是有錯誤發生
	-2	資料成功取回但是等待非同步命令完成超時
	-3	參數錯誤
參數	int32_t nTryCnt	錯誤嘗試次數
	uint16_t *pu16ReturnDataSize	返回資料大小
	uint8_t *pu8ReturnData	使用者提供空間及指標傳入，函式將儲存返回資料

EcmUsrDriver.cpp

```
int32_t EcmUniversalCmd( uint8_t u8Cmd, uint8_t u8Param, uint8_t u8Param0, uint8_t u8Param1, uint8_t u8Param2, uint8_t u8Param3, uint8_t *pu8Return, uint8_t *pu8Status, uint8_t *pu8ErrStatus, uint16_t *pu16ReturnDataSize, uint8_t *pu8ReturnData)
```

一般命令，請參考XF命令

返回值	1	成功
	0	超時
	-1	資料成功取回但是有錯誤發生
	-2	資料成功取回但是等待非同步命令完成超時
	-3	參數錯誤
參數	uint8_t u8Cmd	命令代碼
	uint8_t u8Param	命令參數
	uint8_t u8Param0	命令資料0
	uint8_t u8Param1	命令資料1
	uint8_t u8Param2	命令資料2

uint8_t u8Param3	命令資料3
uint8_t *pu8Return	回應的返回值
uint8_t *pu8Status	回應的ECAT狀態
uint8_t *pu8ErrStatus	回應的錯誤狀態
uint16_t *pu16ReturnDataSize	回應的Data大小
uint8_t *pu8ReturnData	回應的Data

EcmUsrDriver.cpp		
int32_t ECM_CheckDCStable()		
回傳DC Sync發生時間間隔是否穩定		
返回值	1	DC Sync發生時間間隔趨近週期時間
	0	DC Sync發生時間間隔差異仍大

EcmUsrDriver.cpp		
int32_t ECM_IsSlaveAlive(uint8_t *pEcmStatus, uint8_t *pRxPDOFifoCnt, uint8_t *CrcErrCnt, uint8_t *WkcErrCnt)		
確認從站狀態		
返回值	1	從站已完成配置且FIFO運作中
	0	非上述情況
	-1	無法取得從站資料
參數	uint8_t *pEcmStatus	返回從站狀態
	uint8_t *pRxPDOFifoCnt	返回RxPDOFIFO中已有的命令數量
	uint8_t *CrcErrCnt	返回CRC錯誤次數計數值
	uint8_t *WkcErrCnt	返回Working Counter錯誤計數值

EcmUsrDriver.cpp		
int32_t ECM_InfoUpdate(uint8_t *pEcmStatus, uint8_t *pRxPDOFifoCnt, uint8_t *CrcErrCnt, uint8_t *WkcErrCnt)		
int32_t ECM_InfoUpdateCri(uint8_t *pEcmStatus, uint8_t *pRxPDOFifoCnt, uint8_t *CrcErrCnt, uint8_t *WkcErrCnt)		
取回從站狀態, ECM_InfoUpdateCri 不使用Critical Section		
返回值	1	CRC正確, 成功取回從站資料
	0	CRC錯誤
參數	uint8_t *pEcmStatus	返回從站狀態
	uint8_t *pRxPDOFifoCnt	返回RxPDOFIFO中已有的命令數量
	uint8_t *CrcErrCnt	返回CRC錯誤次數計數值
	uint8_t *WkcErrCnt	返回Working Counter錯誤計數值

EcmUsrDriver.cpp		
int32_t ECM_SetCrcType(uint8_t u8Type)		
設置CRC的方式		
返回值	1	成功設置CRC Type
	0	設置失敗
參數	uint8_t u8Type	0: 固定值檢查(0x12345678) 3: CRC-32

void ECM_CloseLibrary()		
關閉並釋放驅動函式庫內記憶體空間		

EcmUsrDriver.cpp		
uint8_t ECM_InitLibrary(uint16_t *pu16SpiDataSize)		
初始化驅動函式庫內所需的變數及記憶體空間		
返回值	0	失敗
	Others	成功並返回晶片韌體版本號
參數	uint16_t *pu16SpiDataSize	因為PCIe卡是透過PCIe to SPI與晶片連接, SPI封包大小是可以變更的, 故此參數僅對ECM-XF PCIe裝置有效 0 : 不變更SPI封包資料段大小 32~1408 : 變更SPI封包資料段大小

EcmUsrDriver.cpp		
int32_t ECM_EcatInit(uint16_t DCActCode, uint32_t CycTime)		
初始化EtherCAT網路, 取回從站資訊, 清除錯誤並配置從站。		
返回值	1	成功
	0	超時
	-3	參數錯誤
參數	uint16_t DCActCode	DC AssignActivate code 例如: 0x0300 for DC Sync0 0x0000 for Free run
	uint32_t CycTime	DC週期時間(單位ns)

EcmUsrDriver.cpp		
int8_t ECM_EcatSlvCntGet()		
取回連線中從站數量		
返回值	-1	失敗
	>0	從站數量
	-3	參數錯誤

EcmUsrDriver.cpp	
int32_t ECM_SetTxFIFOCnt(uint8_t u8TxCnt)	
設定ECM-XF內TxPDO的FIFO緩存最大筆數。 ECM-XF內建FIFO緩存空間，可確保PDO週期資料交換不會丟失資料，預設筆數為64筆。	
返回值	參考EcmCmdTransceiver返回值
參數	uint8_t u8TxCnt TxPDO的FIFO緩存最大筆數。

EcmUsrDriver.cpp	
int32_t ECM_SetRxFIFOCnt(uint8_t u8TxCnt)	
設定ECM-XF內RxPDO的緩存筆數。 ECM-XF內建FIFO緩存空間，可確保PDO週期資料交換不會丟失資料，預設筆數為64筆。	
返回值	參考EcmCmdTransceiver返回值
參數	uint8_t u8RxCnt RxPDO的緩存筆數

EcmUsrDriver.cpp	
int32_t ECM_GetTxFIFOCnt(uint8_t *pu8Cnt)	
取回晶片內TxPDO的FIFO緩存最大筆數設定值。	
返回值	參考EcmCmdTransceiver返回值
參數	uint8_t *pu8Cnt 取回TxPDO的FIFO緩存最大筆數

EcmUsrDriver.cpp	
int32_t ECM_GetRxFIFOCnt(uint8_t *pu8Cnt)	
取回晶片內RxPDO的FIFO緩存最大筆數設定值。	

返回值	參考EcmCmdTransceiver返回值	
參數	uint8_t *pu8Cnt	將取回RxPDO的FIFO緩存最大筆數

Utility.c		
int32_t ECM_CheckMEMSpace(int32_t nPdoCntOnce)		
根據TxPDO及RxPDO的大小、設定最大筆數，判斷晶片內FIFO緩存空間是否足夠，也會SPI/USB的資料封包大小是否足夠。		
返回值	1	成功
	-1	讀取封包大小錯誤
	-2	讀取PDO大小錯誤
	-3	讀取FIFO最大筆數錯誤
	-4	FIFO緩存空間不足
	-5	資料封包大小不足
參數	int32_t nPdoCntOnce	每次資料交換會包含的PDO筆數

EcmUsrDriver.cpp		
int ECM_EcatPdoFifoDataExchangeAdv(uint8_t u8Op, uint8_t u8Cnt, uint8_t *pRxData, uint8_t *pTxData, uint16_t u16DataSize, uint8_t *pu8RxPdoFifoCnt, uint8_t *CrcErrCnt, uint8_t *WkcErrCnt, uint8_t *IsSlvAlive)		
發送多筆RxPDO並接收多筆TxPDO		
返回值	-1	CRC錯誤
	4	上次命令無寫入RxPDO到FIFO，也無從FIFO讀取到TxPDO
	5	上次命令 從FIFO讀取到TxPDO成功
	6	上次命令 寫入RxPDO到FIFO成功
	7	上次命令 從FIFO讀取到TxPDO成功，且寫入RxPDO到FIFO成功

參數	uint8_t u8Op	將執行的讀寫操作 bit0：寫入RxPDO到RxFIFO bit1：從TxFIFO讀取到TxPDO
	uint8_t u8Cnt	每次讀寫的PDO筆數
	uint8_t *pRxData	將此指標指向的RxPDO資料複製並送入FIFO
	uint8_t *pTxData	從FIFO讀取讀取TxPDO資料，並複製到此指標空間
	uint16_t u16DataSize	欲送出的RxPDO資料大小
	uint8_t *pu8RxpdoFifoCnt	透過此指標得知預存的RxPDO筆數
	uint8_t *CrcErrCnt	透過此指標取回SPI或USB資料錯誤次數
	uint8_t *WkcErrCnt	透過此指標取回EtherCAT資料錯誤次數
	uint8_t *IsSlvAlive	透過此指標得知從站是否斷線

EcmUsrDriver.cpp	
int32_t ECM_InitFIFO(void)	
初始化FIFO空間，使用FIFO前需要先初始化，晶片會根據設置的FIFO筆數及PDO大小配置適合的空間。	
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義

EcmUsrDriver.cpp	
int32_t ECM_EnableFIFO(uint8_t u8Enable)	
始能FIFO，當晶片進入SAFEOP或OP狀態後，將會週期的從FIFO內取出RxPDO送出，並將收到的TxPDO存入。需先初始化 FIFO 空間。	
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義
參數	uint8_t u8Enable 0: DISABLE 1: ENABLE

EcmUsrDriver.cpp

int32_t ECM_ClearFIFO(uint8_t u8TxRx)	
清除FIFO內的資料。需先初始化 FIFO 空間。	
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義
參數	uint8_t u8TxRx 0 : 清除TxPDO FIFO 及 RxPDO FIFO 1 : 僅清除TxPDO FIFO 2 : 僅清除RxPDO FIFO

EcmUsrDriver.cpp	
int32_t ECM_EcatStateSet(uint8_t u8Slave, uint8_t u8State)	
切換從站EtherCAT狀態	
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義
參數	uint8_t u8Slave 0~127 : 從站位置 255 : 全部從站
	uint8_t u8State 欲切換至的狀態 0x01: INIT 0x02: PRE_OP 0x03: BOOT 0x04: SAFE_OP 0x08: OPERATIONAL 0x10: Clear ERROR

EcmUsrDriver.cpp	
int32_t ECM_EcatStateGet(uint8_t u8Slave, uint8_t *pu8State)	
讀取從站EtherCAT狀態	
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義
參數	uint8_t u8Slave 0~127 : 從站位置 255 : 全部從站

uint8_t *pu8State	會將狀態值複製至此址標空間
-------------------	---------------

EcmUsrDriver.cpp		
int32_t ECM_EcatSdoWrite(uint8_t Slave, uint16_t Index, uint8_t SubIndex, uint16_t size, int32_t Timeout, uint8_t *Data)		
透過SDO寫入從站物件。		
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義	
參數	uint8_t Slave	0~127 : 從站位置
	uint16_t Index	索引號
	uint8_t SubIndex	副索引號
	uint16_t size	資料大小, 單位byte
	int32_t Timeout	超時等待時間, 單位ns
	uint8_t *Data	指標指向欲寫入資料的記憶體位置

EcmUsrDriver.cpp		
int32_t ECM_EcatSdoRead(uint8_t Slave, uint16_t Index, uint8_t SubIndex, uint16_t size, int32_t Timeout, uint8_t *Data)		
透過SDO讀取從站物件。		
返回值	參考EcmCmdTransceiver_WaitAyncDone返回值定義	
參數	uint8_t Slave	0~127 : 從站位置
	uint16_t Index	索引號
	uint8_t SubIndex	副索引號
	uint16_t size	資料大小, 單位byte
	int32_t Timeout	超時等待時間, 單位ns

uint8_t *Data	讀取的資料會複製至指標指向的記憶體位置
---------------	---------------------

附錄 疑難排解

* 如何確認主站IC正常工作？

請依下列步驟確認

STEP1: 將ECM-XFU-SK J3 接至L (即XF IC Pin12拉低, Firmware update模式) 再上電, 綠燈滅, USB插上電腦可以找到裝置 => XF正常工作

STEP2: 上電時pin12拉高, 綠燈亮 => 模式選擇正常

STEP3: 開始例程後, 綠燈滅 => XF 收到正確命令 (SPI MOSI/CLK/CS正常)

STEP4: 讀到版本號 => SPI讀寫正常

STEP5: 讀數slave數量 => PHY / RJ45 正常

* 接USB線後, 電腦找不到裝置？

請確認IC型號為ECM-XFU, 若為ECM-XF則僅在Firmware Update模式支援USB通訊。

請確認USB線材, 部分線材僅供充電, 不具信號傳輸功能, 請更換線材進行測試測試。

* 執行ITE測試程式後, 出現「找不到VCRUNTIME140.dll」信息？

請安裝**Visual studio 2017**可轉發套件, 參考網站

<https://learn.microsoft.com/cpp/windows/latest-supported-vc-redist?view=msvc-170#visual-studio-2015-2017-2019-and-2022>

* 參考電路上SPI接有100R電阻, 是什麼用途？

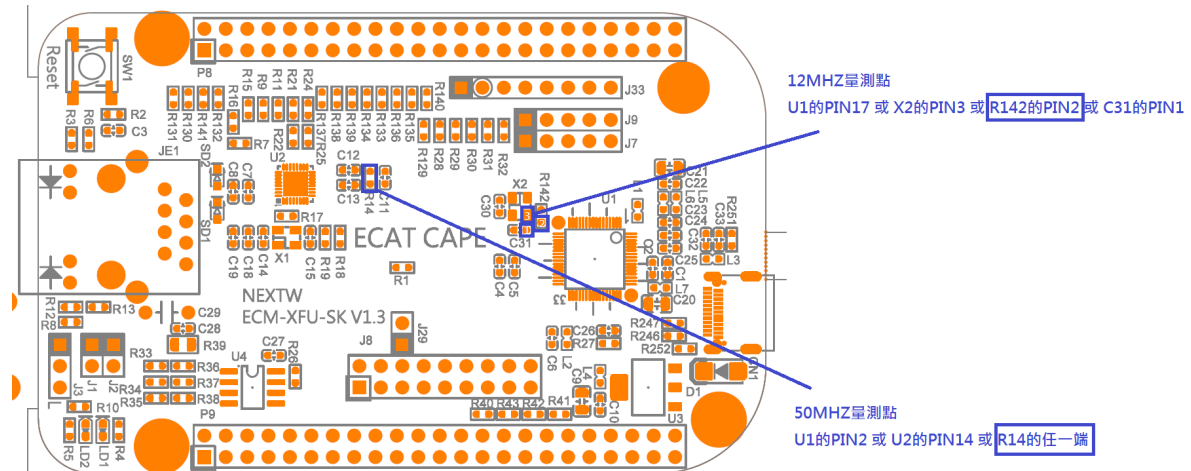
SPI上的100R電阻是阻抗匹配使用的, 一般來說若SPI兩端距離很短且在同一板上, 可以使用0R取代。請注意, 過大的電阻會造成訊號延遲, 而使得SPI速度受限。

* 為何12MHz無源振盪器不起振？

12MHz振盪器為USB功能使用，ECM-XF在一般模式下，所接的12MHz無源振盪器不起振為正常現象。

若IC型號為ECM-XFU或ECM-XF IC在Firmware Update模式時12MHz應起振，若無請檢查電路或確認IC是否正常工作。

25MHz振盪器供給PHY做為網路參考時鐘使用，在所有情況下上電後應起振。

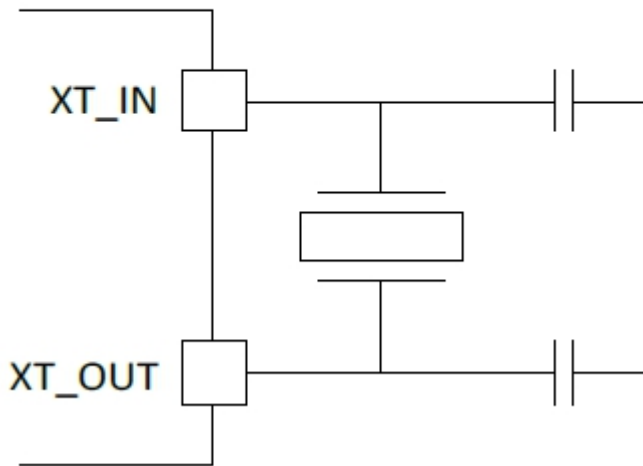


* 可以更換有源晶振嗎？

25M為PHY使用，可以更換為有源振盪器。

12M不可以，XF內部設定12M的晶振來源是無源振盪器，當有使用到USB連線功能時會參考12MHz的振盪器，如下圖

Crystal oscillator



* 可以某些從站使用DC Sync, 某些從站使用Free Run嗎？

可以先使用ECM_EcatInit 來設定大多數的從站DC狀態，在配置完PDO並呼叫ReConfig後，再使用ECM_EcatSetDCSync (參照命令 ECM_CMD_ECAT_DCSYNC)來設定個別從站，並指定特定的DC Code.可參考下圖藍框處。

```

56
57     ECM_SetEcatInitMode((uint8_t)'0'); // 代表使用程式自定義的SM模式
58
59     nRet = ECM_EcatInit(0x300, 1000000); // 第一個參數0x300代表DC Sync, 0代表Free Run (
60     printf("ECM_EcatInit %d\r\n", nRet); //return 1 代表 Init成功
61     ECM_EcatConfigSM(1, 3, 0x11C0, 0, 0x20, 0);
62     ECM_EcatConfigMap();
63     ECM_EcatSetDCSync(1, 0x0, 1000000);

```

* AL Status Code 是什麼意義？

AL Status Code是從站EtherCAT處理的錯誤資訊，通常在轉換狀態失敗後，可以透過此代碼瞭解失敗的原因。若此代碼為0代表沒有錯誤，其他值的意義請參考Beckhoff網站

<https://infosys.beckhoff.com/english.php?content=../content/1033/ethercatsystem/1037010571.html&id=>

* AL Status Code 顯示0x1E是什麼意義？

根據Beckhoff網站所查詢到的AL Status Code 0x1E代表「SM configuration for input process data is invalid」, input process data 代表TxPDO的配置不被從站接受或是SM3(通常SM3為input的SM, 可透過ESI檔確認)的配置錯誤。

1. 若有透過0x1C13來配置PDO, 請注意配置內容是否正確
2. 若使用預設的配置, 代表主站晶片讀取到錯誤的從站配置, 需手動指定, 指定方式如下:

STEP1: 於ECM_EcatInit之前指定強制指定模式 `ECM_SetEcatInitMode((uint8_t)'0');`

STEP2: 於ECM_EcatInit 之後 及 ECM_EcatReconfig之後 強制指定SM的大小

`ECM_DirectAssign(uint8_t u8Slv, uint8_t u8Mode, uint16_t u16Obits, uint16_t u16lbits)`

第一個參數u8Slv為從站的順序站號, 從0開始。

第二個參數u8Mode是否開啟強制指定, 0代表使用從站預設值, 1代表強制指定。

第三個參數u16Obits代表RxPDO的大小, 單位是bit。

第四個參數u16lbits代表TxPDO的大小, 單位是bit。

STEP3: 在指定全部的從站後(若從站能使用預設值則不須指定)呼叫
`ECM_EcatConfigMap();`

若仍無法順利進到OP, 則可將STEP2換成`ECM_EcatConfigSM(uint8_t slave, uint8_t nSM, uint16_t StartAddr, uint16_t Smlength, uint8_t ControlReg, uint8_t Activate)`

第一個參數slave為從站的順序站號, 從0開始。

第二個參數uSM代表欲指定的SyncManager編號, 請參考從站ESI檔案說明, 通常output為2, input為3。

第三個參數StartAddr代表起始位置, 請參考從站ESI檔案說明。

第四個參數Smlength代表SM的大小, 單位是byte, 請填入RxPDO(output)或TxPDO(input)的大小, 此值必須為從站ESI檔中該SM的最大值與最小值之間。

第五個參數ControlReg代表SM的控制碼, 請參考從站ESI檔案說明。

第六個參數Activate代表SM是否啟用，請參考從站ESI檔案中Enable的值。

以下為ESI檔案中關於SM的說明。

```
<Sm MinSize="128" MaxSize="128" DefaultSize="128" StartAddress="#x1000" ControlByte="#x26" Enable="1">MBoxOut</Sm>
<Sm MinSize="128" MaxSize="128" DefaultSize="128" StartAddress="#x1080" ControlByte="#x22" Enable="1">MBoxIn</Sm>
<Sm MinSize="1" MaxSize="64" DefaultSize="8" StartAddress="#x1100" ControlByte="#x24" Enable="1">Outputs</Sm>
<Sm MinSize="1" MaxSize="64" DefaultSize="8" StartAddress="#x11C0" ControlByte="#x20" Enable="0">Inputs</Sm>
```

以下為設定範例

```
u8Version = ECM_InitLibrary(&u16SpiSize, ECM_CRC_TYPE_NONE);
printf("u8Version 0x%X\r\n", u8Version);
if (u8Version == 0)
    return 0;
nRet = ECM_GetNetLinkPin(&bLinkPin);
printf("ECM_GetNetLinkPin linkpin(%d) ret(%d)\r\n", bLinkPin, nRet); //linkpin=0 代表網路接口沒有連結

ECM_SetEcatInitMode((uint8_t)'0'); // 代表使用程式自定義的SM模式

nRet = ECM_EcatInit(0x0, 1000000); // 第一個參數0x300代表DC Sync, 0代表Free Run (依從站ESI說明設置), 第二個參數為週期時間(單位ns)
printf("ECM_EcatInit: %d\r\n", nRet); //return 1 代表 Init成功
ECM_EcatConfigSM(1, 3, 0x11C0, 0, 0x20, 0);
ECM_EcatConfigMap();
```

* 執行時出現「無法載入DLL」「找不到指定的模組」「DllNotFoundException」訊息。

請將「ECMPLATFORM.dll」及「EcmXFUDrv.dll」放入與執行檔相同的資料夾

* 執行時出現「試圖載入格式錯誤的程式」「BadImageFormatException」訊息。

請確認dll為32位元版本或是64位元版本，兩者不相同

* 如何透過SPI傳輸做到高度即時性？

ECM-XF內有TxFIFO用於暫存從站送回主站的資料，亦有RxFIFO用於暫存主站欲送至從站的命令。在某些應用中，希望命令儘快送至從站，且從站的資料能儘速回到上位機中。此種應用上位機必須具備高度即時性，能在中斷發生後立即處理SPI資料交換，具體說明如下：

1. 啟用INT1中斷，中斷來源可為BIT31收到封包時 或 BIT25 RxFIFO低極限 或 BIT24 TxFIFO高極限
2. 若使用RxFIFO低極限或TxFIFO高極限，需先透過ECM_CMD_ECAT_SET_FIFO_TH(命令碼70) 設定高/低極限
2. 當發生中斷時，立刻進行SPI資料交換，此時會收到最新的TxPDO，並將下一週期的RxPDO放入RxFIFO中。

3. SPI資料交換後的UserDelay延時可為0 (此定義在EcmUsrDriver.c中的SpiDataExchange中), 或 TEST_SPI_IDLE_TIME = 0.

上述方式必須透過SPI傳輸及INT1的中斷來進行, 透過USB傳輸受限於Windows作業系統為非即時作業系統(non-real-time OS)無法實現高度即時性。

