# iTE SDK

# UART/SWUART/SPI/I2C 介面開發指南

**V0.9**

## ITE TECH. INC.

修訂記錄

| 修訂日期 | 修訂說明 | 頁次 |
|---|---|---|
| 2014/10/01 | 初建版本 V0.9 | |
| 2016/05/10 | 修正I2C測試 | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

目錄

# 1. 前言

## 1.1 編寫目的

介紹UART/SWUART/SPI/I2C 模組之功能, 說明UART/SWUART/SPI/I2C模組相關的API之操作及使用.

## 1.2 適用範圍

UART通常用來列印信息或是用來控制MCU。SPI有較快的存取速度，目前IT9850有用於NOR的讀/寫介面，或是當作與PC溝通的傳輸介面。I2C速度比UART稍快，使用最少的控制腳位，通常用作兩個晶片之間的溝通或控制的介面，例如AUDIO CODEC，I/O Expander，TOUCH KEY/PANEL 等。

## 1.3 適用人員

軟體應用程式，驅動程式開發者(如touch panel或是touch key等)

# 2. UART/SWUART/SPI/IIC模組介紹

## 2.1 Description

Embedded system領域中經常用到UART/SWUART/SPI/IIC等serial bus。以下將介紹在IT9850的SDK中，UART/SWUART/SPI/I2C相關應用的API與使用方式。

## 2.2 UART 模組

ITP UART driver相關程式放在”\sdk\driver\itp\itp_uart.c”
ITP DRIVER是依據POSIX規範實作的API，可以使用OPEN/READ/WRITE/IOCTL等函式對I/O DEVICE進行如讀寫檔案般的操作。以下是基於這種規範所實做的UART ITP API。

### 2.2.1 IOCTL (UART初始化與重置)

DEVICE ID : ITP_DEVICE_UART1 (參考“sdk\include\ite\itp.h”)

UART device 註冊

```
itpRegisterDevice(ITP_DEVICE_UART1, &itpDeviceUart1);
```

使用此函式可以將UART device註冊到ITP的driver中，使UART可以透過ioctl/read/write等函式來操作UART的功能。本函式已經含在ITP driver的初始化過程當中(參考 “sdk\driver\itp_init_openrtos.c”)，所以一般不必再執行此註冊函式。

UART device的初始化

```
/**
 * Device ioctl method (controls device operating parameters).
 *
 * @param file File descriptor referring to an open device.
 * @param request Selects the control function to be performed.
 * @param ptr Additional information that is needed by this specific device to perform the requested function.
 * @param info Device custom data.
 * @return Upon successful completion, it shall return a value other than -1 that depends upon the device control function. Otherwise, it shall return -1 and set errno to indicate the error.
 */
int (*ioctl)(int file, unsigned long request, void* ptr, void* info);

ioctl(ITP_DEVICE_UART1, ITP_IOCTL_INIT, (void*)baud_rate);
```

baud_rate: 輸入參數為baud rate值。

- UART 模組的初始化已經含在ITP driver的初始化過程當中，所以一般不必再執行此註冊函式(參考 “sdk\driver\itp_init_openrtos.c”)。

UART device的重置

```
ioctl(ITP_DEVICE_UART1, ITP_IOCTL_RESET, (void*)baud_rate);
```

此函式是重置UART DEVICE，其中輸入參數為baud rate值。

## 2.3 SWUART模組

ITP SWUART driver相關程式放在”\sdk\driver\itp\itp_swuart_codec.c”
ITP DRIVER是依據POSIX規範實作的API，可以使用OPEN/READ/WRITE/IOCTL等函式對I/O DEVICE進行如讀寫檔案般的操作。以下是基於這種規範所實做的SWUART ITP API。

### 2.3.1 IOCTL (SWUART初始化)

itpRegisterDevice(ITP_DEVICE_SWUART, & itpDeviceSwUartCodec);

使用此函式可以將Software UART device註冊到ITP的driver中，使Software UART可以透過ioctl/read/write等函式來操作Software UART的功能。本函式已經含在ITP driver的初始化過程當中(參考“sdk\driver\itp_init_openrtos.c”)，所以一般不必再執行此註冊函式。

Software UART device的初始化

```
/**
 * Device ioctl method (controls device operating parameters).
 *
 * @param file File descriptor referring to an open device.
 * @param request Selects the control function to be performed.
 * @param ptr Additional information that is needed by this specific device to perform the requested function.
 * @param info Device custom data.
 * @return Upon successful completion, it shall return a value other than -1 that depends upon the device control function. Otherwise, it shall return -1 and set errno to indicate the error.
 */
int (*ioctl)(int file, unsigned long request, void* ptr, void* info);
```

```
ioctl(ITP_DEVICE_SWUART, ITP_IOCTL_SET_BAUDRATE, (void*)baud_rate);
ioctl(ITP_DEVICE_SWUART, ITP_IOCTL_SET_GPIO_PIN, (void*)gpio);
ioctl(ITP_DEVICE_SWUART, ITP_IOCTL_INIT, NULL);
```

baud_rate: 輸入參數為baud rate值。
gpio : 輸入參數為gpio值

## 2.4 SPI模組

Device ID : ITP_DEVICE_SPI (參考“sdk\include\ite\itp.h”)
Structure of SPI device (參考“sdk\include\ite\itp.h”)

```
/** @defgroup itp_spi SPI
 *  @{
 */
typedef enum
{
    ITP_SPI_PIO_READ,
    ITP_SPI_DMA_READ,
    ITP_SPI_PIO_WRITE,
    ITP_SPI_DMA_WRITE,
}ITPSpiReadWriteFunc;
```

SPI device執行read/write時，所需使用的I/O操作設定(使用DMA mode或是PIO mode，參考2.3.5 & 2.3.6)

```
typedef struct
{
    ITPSpiReadWriteFunc readWriteFunc;
    void*               cmdBuffer;
    uint32_t            cmdBufferSize;
    void*               dataBuffer;
    uint32_t            dataBufferSize;
    uint8_t             dataLength;
} ITPSpiInfo;

/** @} */ // end of itp_spi
```

SPI device執行read/write時，所需使用的SPI相關資料(參考2.3.5 & 2.3.6)

### 2.4.1 IOCTL (SPI device的註冊)

```
itpRegisterDevice(ITP_DEVICE_SPI, &itpDeviceSpi0);
```

- 使用此函式可以將SPI device註冊到ITP的driver中，使SPI可以透過open/close/ioctl/read/write等函式來操作SPI的功能。本函式已經含在ITP driver的初始化過程當中(參考“sdk\driver\itp_init_openrtos.c”)，所以一般不必再執行此註冊函式。

### 2.4.2 IOCTL (SPI的ioctl與初始化)

SPI device是透過ioctl函式來執行初始化的操作(參考“sdk\include\ite\itp.h”)。

```
    /**
     * Device ioctl method (controls device operating parameters).
     *
     * @param file File descriptor referring to an open device.
     * @param request Selects the control function to be performed.
```

```
    * @param ptr Additional information that is needed by this specific device to perform the requested function.
    * @param info Device custom data.
    * @return Upon successful completion, it shall return a value other than -1 that depends upon the device control function. Otherwise, it shall return -1 and set errno to indicate the error.
    */
   int (*ioctl)(int file, unsigned long request, void* ptr, void* info);
```

以下是SPI device透過ioctl函式來執行初始化的code。

```
ioctl(ITP_DEVICE_SPI, ITP_IOCTL_INIT, NULL);
```

## 2.4.3 open SPI device

使用 “open” 來open SPI device 。操作方式如下所示：

```
{
Int fd;
fd = open(":spi:0", O_RDWR);
if(fd == -1)        printf(“open SPI fail!!\n”);
else          printf(“open SPI success!!\n”);
}
```

當OPEN成功時會回傳一個整數值(file descriptor)，若OPEN失敗，則會回傳-1。輸入參數”O_RDWR”表示以可讀寫方式開啟device(可參考“fcntl.h”)。

## 2.4.4 close SPI device

使用 “close”來close SPI device 。操作方式如下所示：

```
{
Int fd;
fd = close(":spi:0", O_RDWR);
if(fd != -1) close(fd);
else                printf(“close SPI device fail!!\n”);

}
```

## 2.4.5 read SPI device

當完成SPI的初始化以及OPEN SPI DEVICE等操作之後，便可開始對SPI DEVICE進行讀寫操作。讀取SPI DEVICE的流程如下所示：

```
{
        ITPSpiInfo evt;                     //用ITPSpiInfo宣告一個 evt的結構變數
        unsigned char spiCmd = 0x82;        //宣告spiCmd變數
        unsigned char spiData[2];           //宣告一個1*2的array

        evt.cmdBuffer      = &spiCmd;       //設定command buffer
        evt.cmdBufferSize  = 1;             //設定command長度(以dataLength為單位)
        evt.dataBuffer     = &spiData;      //設定SPI data buffer
        evt.dataBufferSize = 2;             //設定SPI data buffer長度(以dataLength為單位)
        evt.dataLength     = 8;             // for FIFO length 8/16/32 bits

        evt.readWriteFunc  = ITP_SPI_DMA_READ;   //設定SPI使用DMA MODE讀取資料
```

```
        result = read(fd, evt, 1);              //執行”read”函式
}
```

## 2.4.6 write SPI device

當完成SPI的初始化以及open SPI device等操作之後，便可開始對SPI device進行讀寫操作。資料寫入SPI device的流程如下所示：

```
{
        ITPSpiInfo evt;                         //用ITPSpiInfo宣告一個 evt的結構變數
        unsigned char spiCmd = 0x98;            //宣告spiCmd變數
        unsigned char spiData[2]={0x1,0x2} //宣告一個1*2的array

        evt.cmdBuffer      = &spiCmd;           //設定command buffer
        evt.cmdBufferSize  = 1;                 //設定command長度(以dataLength為單位)
        evt.dataBuffer     = &spiData;          //設定SPI data buffer
        evt.dataBufferSize = 2;                 //設定SPI data buffer長度(以dataLength為單位)
        evt.dataLength     = 8;                 // for FIFO length 8/16/32 bits

        evt.readWriteFunc  = ITP_SPI_DMA_WRITE;  //設定SPI使用DMA MODE寫入資料

        result = write(fd, evt, 1);             //執行”write”函式
}
```

## 2.5 I2C模組

Device ID : ITP_DEVICE_I2C (參考“sdk\include\ite\itp.h”)

Structure of I2C device (參考“sdk\include\ite\itp.h”)

```
/** @defgroup itp_i2c I2C
 *  @{
 */
typedef struct
{
    uint8_t     slaveAddress;
    uint8_t*    cmdBuffer;
    uint32_t    cmdBufferSize;
    uint8_t*    dataBuffer;
    uint32_t    dataBufferSize;
} ITPI2cInfo;

/** @} */ // end of itp_i2c
```

I2C device執行read/write時，所需使用的I2C相關資料(參考2.3.5 & 2.3.6)

### 2.5.1 IOCTL (I2C device的註冊)

```
itpRegisterDevice(ITP_DEVICE_I2C, &itpDeviceI2c);
```

- 使用此函式可以將I2C device註冊到ITP的driver中，使I2C可以透過open/close/ioctl/read/write等函式來操作SPI的功能。本函式已經含在ITP driver的初始化過程當中(參考“sdk\driver\itp_init_openrtos.c”)，所以一般不必再執行此註冊函式。

### 2.5.2 IOCTL (I2C的ioctl與初始化)

I2C device是透過ioctl函式來執行初始化的操作(參考“sdk\include\ite\itp.h”)。

```
    /**
     * Device ioctl method (controls device operating parameters).
     *
     * @param file File descriptor referring to an open device.
     * @param request Selects the control function to be performed.
     * @param ptr Additional information that is needed by this specific device to perform the requested function.
     * @param info Device custom data.
     * @return Upon successful completion, it shall return a value other than -1 that depends upon the device control function. Otherwise, it shall return -1 and set errno to indicate the error.
     */
    int (*ioctl)(int file, unsigned long request, void* ptr, void* info);
```

I2C device透過ioctl函式來執行初始化的code。

```
ioctl(ITP_DEVICE_I2C, ITP_IOCTL_INIT, NULL);
```

透過ioctl函式來執行關閉I2C功能的code。
ioctl(ITP_DEVICE_I2C, ITP_IOCTL_EXIT, NULL);

透過ioctl函式來執行重置I2C device的code。
ioctl(ITP_DEVICE_I2C, ITP_IOCTL_RESET, NULL);

### 2.5.3 open I2C device

使用 “open”來open SPI device 。操作方式如下所示：

```
{
Int fd;
fd = open(":i2c", O_RDWR);
if(fd == -1)        printf(“open I2C fail!!\n”);
else          printf(“open I2C success!!\n”);
}
```

當open成功時會回傳一個整數值(file descriptor)，若open失敗，則會回傳-1。輸入參數”O_RDWR”表示以可讀寫方式開啟device(可參考“fcntl.h”)。

### 2.5.4 close I2C device

使用 “close”來close I2C device 。操作方式如下所示：

```
{
Int fd;
fd = close(":i2c", O_RDWR);
if(fd != -1) close(fd);
else                printf(“close I2C device fail!!\n”);

}
```

### 2.5.5 read I2C device

當完成I2C的初始化以及open I2C device等操作之後，便可開始對I2C device進行讀寫操作。讀取I2C device的流程如下所示：

```
{
            ITPI2cInfo evt;                         //用ITPI2cInfo宣告一個 evt的結構變數
            unsigned char i2cCmd = 0x82;        //宣告I2CiCmd變數
            unsigned char i2cData[14];          //宣告一個1*2的array

evt.slaveAddress   = 0x8C;              //設定I2C slave address
            evt.cmdBuffer      = &i2cCmd;       //設定command buffer
            evt.cmdBufferSize  = 1;             //設定command長度(以1byte為單位)
            evt.dataBuffer     = &i2cData;      //設定I2C data buffer
            evt.dataBufferSize = 14;            //設定I2C data buffer長度(以1byte為單位)

            result = read(fd, evt, 1);          //執行”read”函式
}
```

## 2.5.6 write I2C device

當完成I2C的初始化以及open I2C device等操作之後，便可開始對I2C device進行讀寫操作。資料寫入I2C device的流程如下所示：

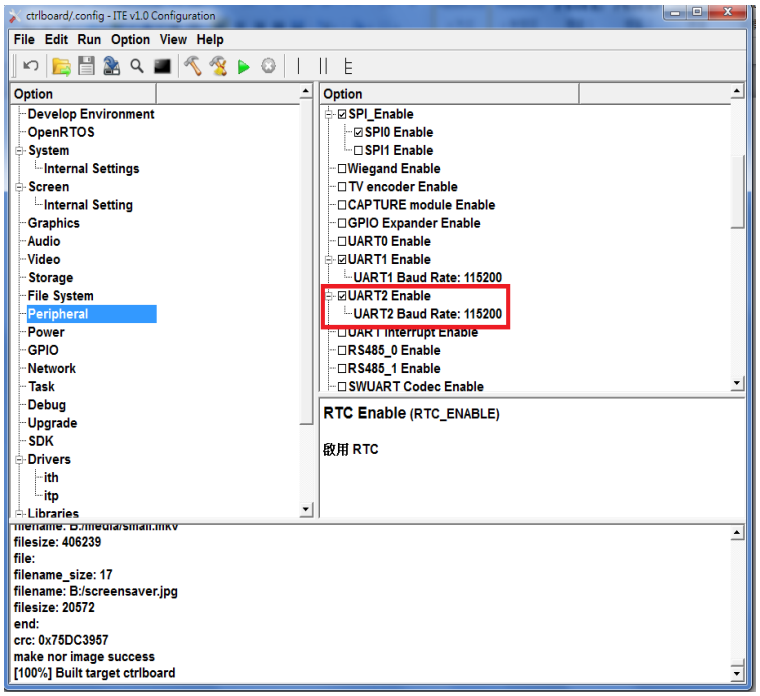
```
{
        ITPI2cInfo evt;                 //用ITPI2cInfo宣告一個 evt的結構變數
        unsigned char i2cCmd = 0x82;    //宣告I2CCmd變數
        unsigned char i2cData[2]={10,20}; //宣告一個1*2的array

evt.slaveAddress   = 0x8C;          //設定I2C slave address
        evt.cmdBuffer     = &i2cCmd;    //設定command buffer
        evt.cmdBufferSize  = 1;         //設定command長度(以1byte為單位)
        evt.dataBuffer     = &i2cData;  //設定I2C data buffer
        evt.dataBufferSize = 2;         //設定I2C data buffer長度(以1byte為單位)

        result = write(fd, evt, 1);     //執行”write”函式
}
```

# 3. 軟件配置說明

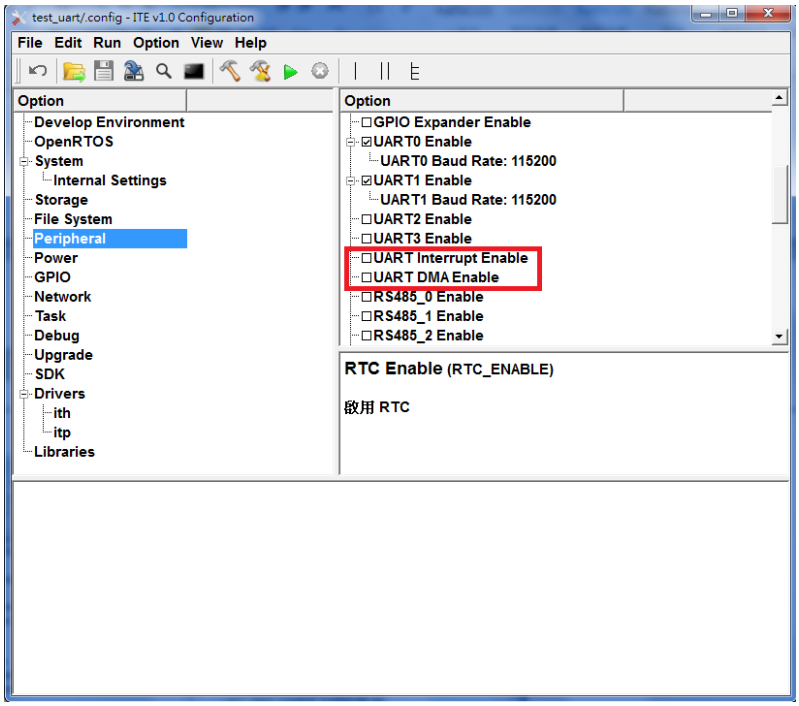
## 3.1 KConfig

To set Kconfig Depend on necessary of project application

### 3.1.1 UART Setting of KConfig

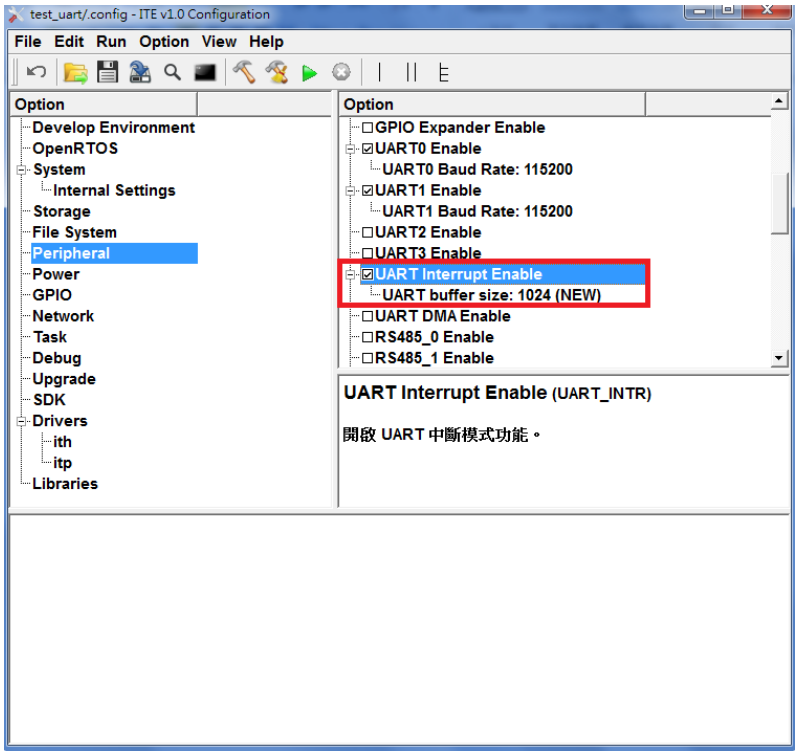


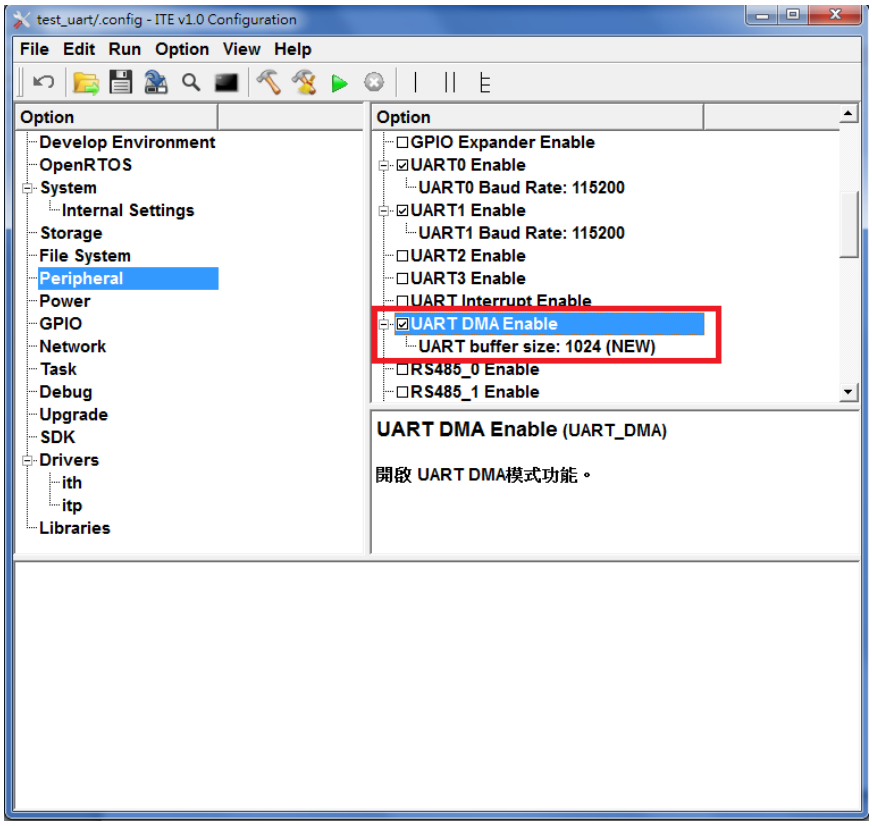

UART2 Enable: To enable UART2 module (建議避免使用 ➔:"UART0 只能使用GPIO0~3，而GPIO0~3通常拿來當SPI或其他功能使用，所以常常跟UART0相衝，UART2可以設定成任何GPIO PIN，所以UART功能通常是選擇UART2而不選UART0").
UART2 Enable: To enable UART2 module.
UART2 Baud Rate :To set the Baud Rate of UART2 (up to 115.2 kbps)。



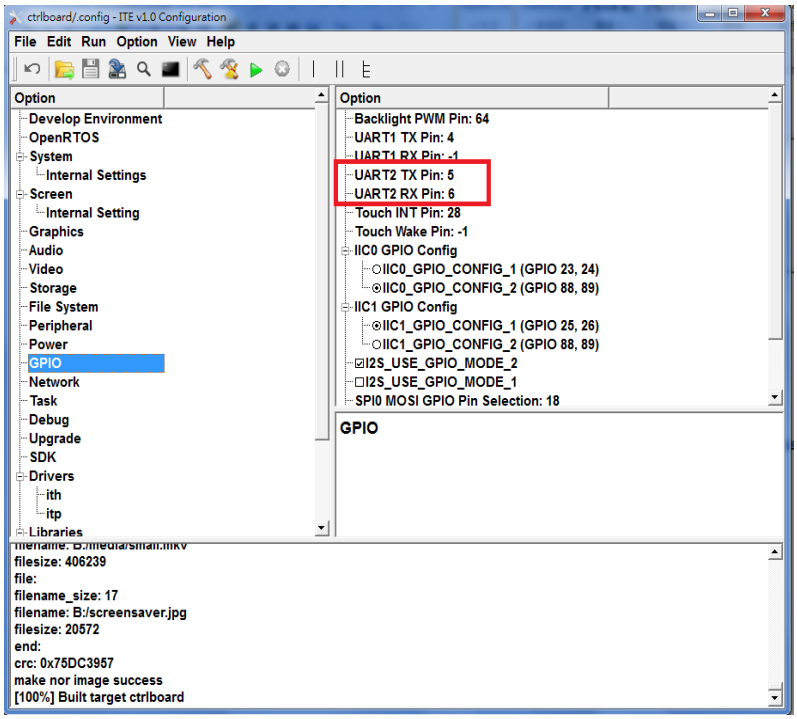

UART TX/RX使用Interrupt or DMA mode or Default

a. Default兩個都不勾的話, UART TX/RX 是用polling mode
b. Interrupt/DMA 只能二擇一, 不能同時
c. DMA mode只有UART0, UART1有, UART2, UART3沒有


test_uart/.config - ITE v1.0 Configuration
File Edit Run Option View Help
Option
Develop Environment
OpenRTOS
System
Internal Settings
Storage
File System
Peripheral
Power
GPIO
Network
Task
Debug
Upgrade
SDK
Drivers
ith
itp
Libraries
Option
GPIO Expander Enable
UART0 Enable
UART0 Baud Rate: 115200
UART1 Enable
UART1 Baud Rate: 115200
UART2 Enable
UART3 Enable
UART Interrupt Enable
UART buffer size: 1024 (NEW)
UART DMA Enable
RS485_0 Enable
RS485_1 Enable
UART Interrupt Enable (UART_INTR)
開啟 UART 中斷模式功能。

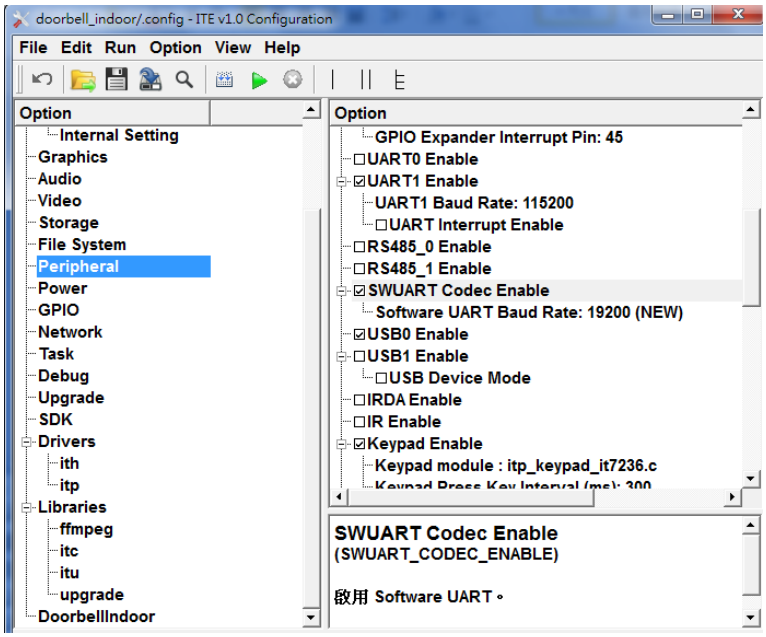



選擇UART Interrupt mode/ DMA mode 需要設定 UART buffer size

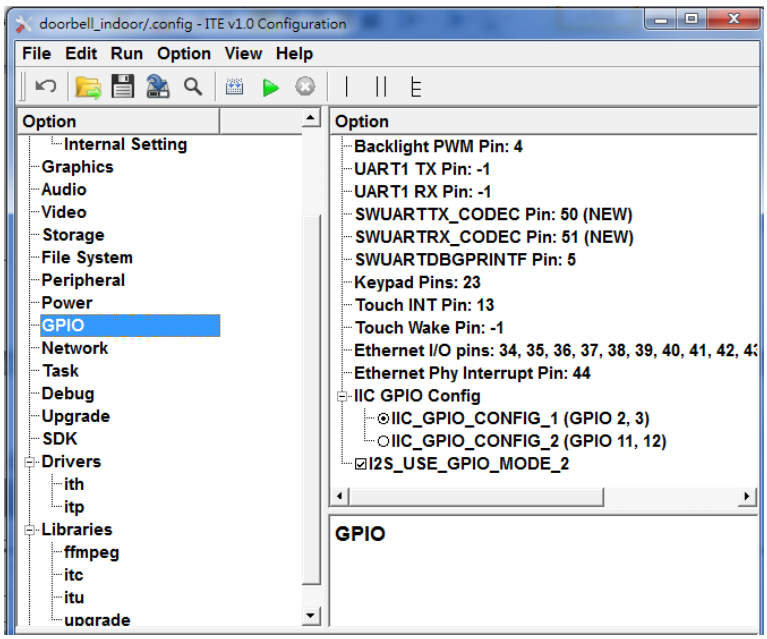



設定UART2 TX/RX的GPIO PIN
UART2 TX PIN: UART2 TX GPIO PIN
UART2 RX PIN: UART2 RX GPIO PIN

## 3.1.2 SWUART Setting of KConfig

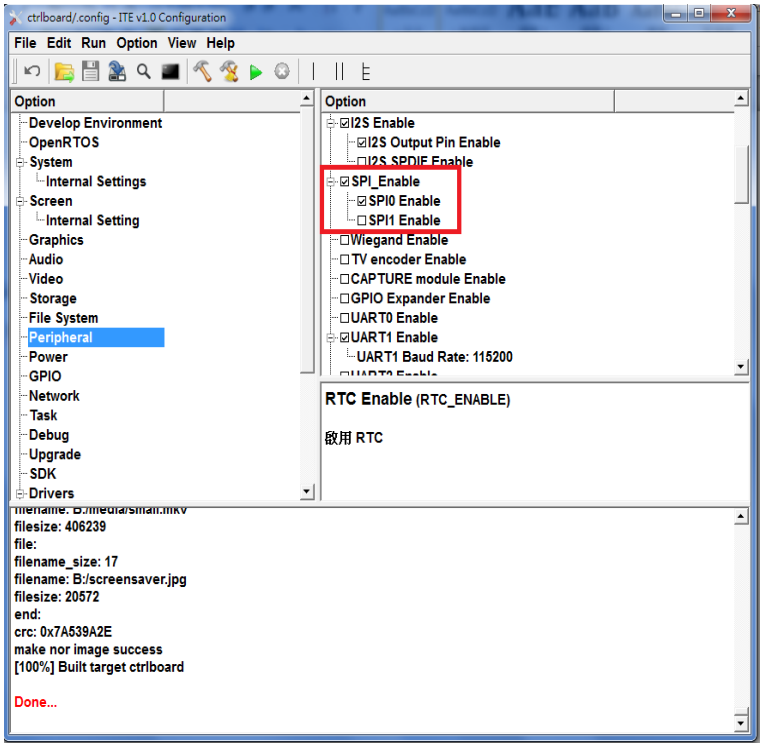



SWUART Enable: To enable SWUART module
Software UART Baud Rate :To set the Baud Rate of SWUART (up to 19.2 kbps)。

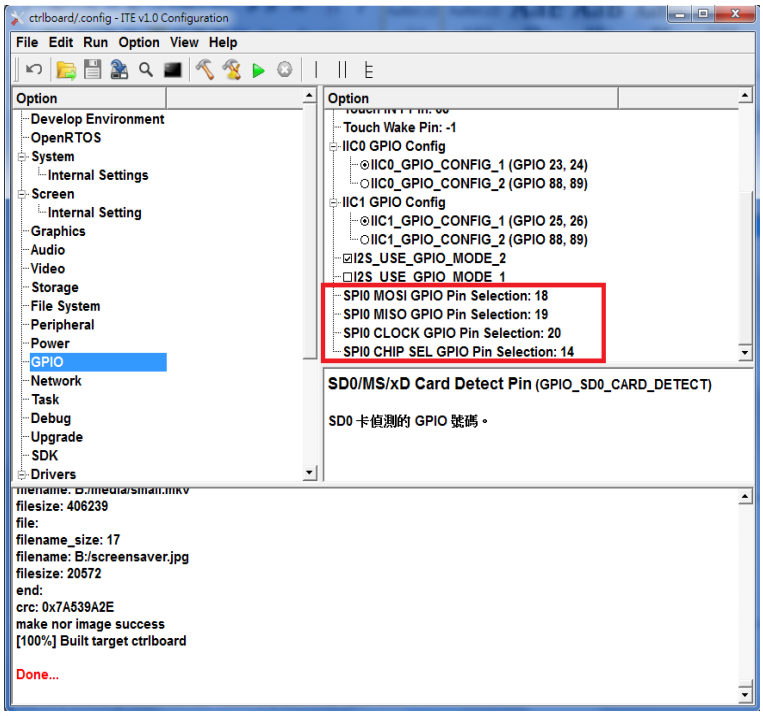



SWUARTTX_CODEC pin : SWUART Tx GPIO Pin
SWUARTRX_CODEC pin : SWUART Rx GPIO Pin

## 3.1.3 SPI Setting of KConfig

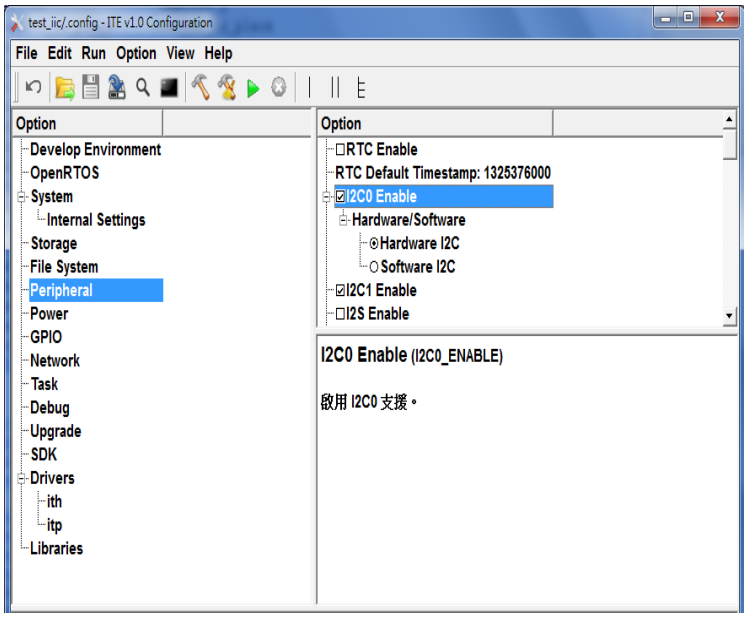



SPI Enable: To enable SPI module.

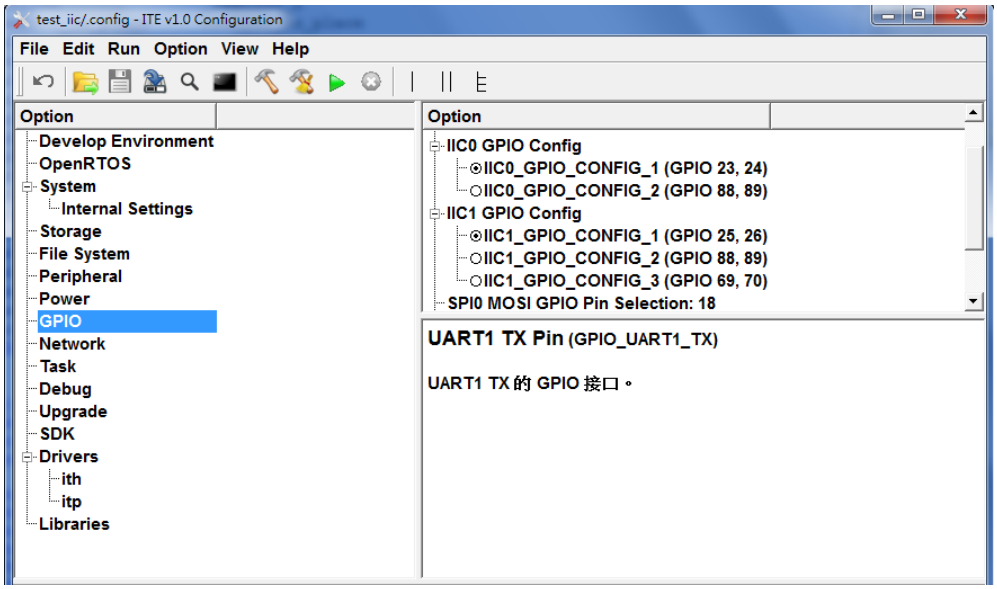



選擇SPI0 GPIO PIN

## 3.1.4 I2C Setting of KConfig




I2C Enable: To enable I2C module.




選擇IIC0/IIC1所在的GPIO PIN

## 3.2 MACRO參數定義

Refer to the “sdk\include\ite\itp.h”

```
/**
 * Device types.
 */
typedef enum
{
    // Standard IO devices
    ITP_DEVICE_STD          = (0 << ITP_DEVICE_BIT),    ///< Standard IO
    ITP_DEVICE_SOCKET       = (1 << ITP_DEVICE_BIT),    ///< LWIP socket

    // File system devices
    ITP_DEVICE_FS           = (2 << ITP_DEVICE_BIT),    ///< File systems
    ITP_DEVICE_FAT          = (2 << ITP_DEVICE_BIT),    ///< FAT file system
    ITP_DEVICE_NTFS         = (3 << ITP_DEVICE_BIT),    ///< NTFS file system

    // Custom devices
    ITP_DEVICE_CUSTOM       = (4 << ITP_DEVICE_BIT),    ///< Custom devices
    ITP_DEVICE_PRINTBUF     = (5 << ITP_DEVICE_BIT),    ///< Print buffer
    ITP_DEVICE_SWUART       = (6 << ITP_DEVICE_BIT),    ///< Software UART
    ITP_DEVICE_UART0        = (7 << ITP_DEVICE_BIT),    ///< UART0
    ITP_DEVICE_UART1        = (8 << ITP_DEVICE_BIT),    ///< UART1
    ITP_DEVICE_LCDCONSOLE   = (9 << ITP_DEVICE_BIT),    ///< LCD console
    ITP_DEVICE_OSDCONSOLE   = (10 << ITP_DEVICE_BIT),   ///< OSD console
```

```
    ITP_DEVICE_SCREEN      = (11 << ITP_DEVICE_BIT),   ///< Screen
    ITP_DEVICE_I2C0        = (12 << ITP_DEVICE_BIT),   ///< I2C
    ITP_DEVICE_I2C1        = (13 << ITP_DEVICE_BIT),   ///< I2C1
    ITP_DEVICE_SPI         = (14 << ITP_DEVICE_BIT),   ///< SPI
    ITP_DEVICE_IR          = (15 << ITP_DEVICE_BIT),   ///< IR
    ITP_DEVICE_NAND        = (16 << ITP_DEVICE_BIT),   ///< NAND
    ITP_DEVICE_NOR         = (17 << ITP_DEVICE_BIT),   ///< NOR
    ITP_DEVICE_SD0         = (18 << ITP_DEVICE_BIT),   ///< SD0
    ITP_DEVICE_SD1         = (19 << ITP_DEVICE_BIT),   ///< SD1
    ITP_DEVICE_USBDFSG     = (20 << ITP_DEVICE_BIT),   ///< USB acts as a USB Mass Storage device
    ITP_DEVICE_CARD        = (21 << ITP_DEVICE_BIT),   ///< Card
    ITP_DEVICE_DRIVE       = (22 << ITP_DEVICE_BIT),   ///< Drive
    ITP_DEVICE_KEYPAD      = (23 << ITP_DEVICE_BIT),   ///< Keypad
    ITP_DEVICE_POWER       = (24 << ITP_DEVICE_BIT),   ///< Power
    ITP_DEVICE_GSENSOR     = (25 << ITP_DEVICE_BIT),   ///< G-Sensor
    ITP_DEVICE_HEADSET     = (26 << ITP_DEVICE_BIT),   ///< Headset
    ITP_DEVICE_AMPLIFIER   = (27 << ITP_DEVICE_BIT),   ///< Audio amplifier
    ITP_DEVICE_STC         = (28 << ITP_DEVICE_BIT),   ///< STC
    ITP_DEVICE_DECOMPRESS  = (29 << ITP_DEVICE_BIT),   ///< Decompress
    ITP_DEVICE_CODEC       = (30 << ITP_DEVICE_BIT),   ///< Audio codec
    ITP_DEVICE_ETHERNET    = (31 << ITP_DEVICE_BIT),   ///< Ethernet
    ITP_DEVICE_WIFI        = (32 << ITP_DEVICE_BIT),   ///< WiFi
    ITP_DEVICE_FILE        = (33 << ITP_DEVICE_BIT),   ///< File
    ITP_DEVICE_DEMOD       = (34 << ITP_DEVICE_BIT),   ///< Demod
    ITP_DEVICE_WATCHDOG    = (35 << ITP_DEVICE_BIT),   ///< Watch Dog
    ITP_DEVICE_NETCONSOLE  = (36 << ITP_DEVICE_BIT),   ///< Network console
    ITP_DEVICE_USB         = (37 << ITP_DEVICE_BIT),   ///< USB
    ITP_DEVICE_DPU         = (38 << ITP_DEVICE_BIT),   ///< DPU (encryption/decryption)
    ITP_DEVICE_XD          = (39 << ITP_DEVICE_BIT),   ///< XD
    ITP_DEVICE_LED         = (40 << ITP_DEVICE_BIT),   ///< LED
    ITP_DEVICE_SWITCH      = (41 << ITP_DEVICE_BIT),   ///< Switch
    ITP_DEVICE_TUNER       = (42 << ITP_DEVICE_BIT),   ///< Tuner
    ITP_DEVICE_STNLCD      = (43 << ITP_DEVICE_BIT),   ///< STN LCD
    ITP_DEVICE_USBMOUSE    = (44 << ITP_DEVICE_BIT),   ///< USB Mouse
    ITP_DEVICE_USBKBD      = (45 << ITP_DEVICE_BIT),   ///< USB Keyboard
    ITP_DEVICE_RTC         = (46 << ITP_DEVICE_BIT),   ///< RTC
    ITP_DEVICE_BACKLIGHT   = (47 << ITP_DEVICE_BIT),   ///< Backlight
    ITP_DEVICE_GPIO_EXPANDER = (48 << ITP_DEVICE_BIT), ///< GPIO Extender

    ITP_DEVICE_LAST
} ITPDeviceType;
```

# 4. 範例

## 4.1 UART範例

```
#include "ite/itp.h"

int main(void)
{
	int i;
char getstr[256];
char sendtr[256];
int len = 0;

	printf("Start uart test!\n");

while(1)
{
len = read(ITP_DEVICE_UART1,getstr,256);

if(len > 0)
{
printf("uart read: %s\n",getstr);
sprintf(sendtr,"uart write: read str = %s",getstr);
write(ITP_DEVICE_UART1,sendtr,256);
}
	}
}
```

## 4.2 SWUART範例

```
unsigned short val;
unsigned char strbuf[256];
int len;
int fd;

fd = _open(":swuartcodec", O_RDONLY);
while(1)
{
	if (fd < 0)
		printf("open device fail\n");
	else
	{
		memset(strbuf, 0, sizeof(strbuf));
		len = _read(fd, strbuf, sizeof(strbuf));
		printf("IO read string = %s\n", strbuf);

		memset(strbuf, 0, sizeof(strbuf));
```

```
            len = sprintf(strbuf, "This a very very very very long string!\n");
            _write(fd ,strbuf, len);
        }
}
```

## 4.3 SPI範例

```
#include <sys/fcntl.h>       //O_RDWR
#include "ite/itp.h"

int main(void)
{
        int fd;
        ITPSpiInfo evt;
        int result = 0;
        unsigned char spiCmd = 0x82;
        unsigned char spiData[2];

        printf("Start SPI test!\n");

    fd = open(":spi:0", O_RDWR);

    while(1)
    {
            evt.cmdBuffer      = &spiCmd;
            evt.cmdBufferSize  = 1;
            evt.dataBuffer     = &spiData;
            evt.dataBufferSize = 2;
            evt.dataLength     = 8;       // for FIFO length 8/16/32 bits

            result = read(fd, evt, 1);

            if(result != evt.dataLength)
            {
                printf("SPI read fail, result=%d, len=%d\n",result,evt.dataLength);
            }
            else
            {
                printf("SPI read data=[%x,%x]\n",spiData[0],spiData[1]);
            }
            usleep(1000*1000);  //sleep 1 second
        }

        return NULL;
}
```

## 4.4 I2C範例

```
/*This test function only for ITE EVB board with I2C0, I2C1 linked together*/
void* TestFunc(void* arg)
{
    /*set master or slave mode*/
    IIC_OP_MODE iic_port0_mode = IIC_MASTER_MODE;
     IIC_OP_MODE iic_port1_mode = IIC_SLAVE_MODE;
    /*init I2C device*/
#ifdef CFG_I2C0_ENABLE
    itpRegisterDevice(ITP_DEVICE_I2C0, &itpDeviceI2c0);
        gMasterDev = open(":i2c0", 0);
    ioctl(ITP_DEVICE_I2C0, ITP_IOCTL_INIT, (void*)iic_port0_mode);
#endif

    // init i2c1 device
#ifdef CFG_I2C1_ENABLE
    itpRegisterDevice(ITP_DEVICE_I2C1, &itpDeviceI2c1);
        gSlaveDev = open(":i2c1", 0);
    ioctl(ITP_DEVICE_I2C1, ITP_IOCTL_INIT, (void*)iic_port1_mode);
#endif
    /*test I2C0 master to read I2C1 slave*/
   TestMasterReadSlave();

        return NULL;
}

void
TestMasterReadSlave()
{
        bool iicResult = false;
        ITPI2cInfo evt;

        /*set I2C1 act as slave mode*/
        iicResult = mmpIicSetSlaveModeCallback(IIC_PORT_1, 0x77,
        _TestMasterReadSlave_IicReceiveCallback, _TestMasterReadSlave_IicWriteCallback);
        mmpIicSetSlaveMode(IIC_PORT_1, 0x77);
        assert(iicResult == true);

        while(1)
        {
                bool        result = false;
                uint8_t             cmd = 0xFE;
                uint8_t             recvBuffer[256] = {0};
                uint32_t    i;

                usleep(500000);
                evt.slaveAddress   = 0x77;
                evt.cmdBuffer      = &cmd;
                evt.cmdBufferSize  = 1;
```

```
            evt.dataBuffer     = recvBuffer;
            evt.dataBufferSize = 256;
            /*I2C0 act master mode to receive  I2C1 slave*/
            result = read(gMasterDev, &evt, 1);

            printf("Master received:\n");
            for (i = 0; i < 256; i++)
            {
                printf("0x%02X ", recvBuffer[i]);
                if(buf[i] != recvBuffer[i])
                {
                    printf("data compare error(0x%x!=0x%x)!!!\n", buf[i], recvBuffer[i]);
                    break;
                }
                else if(i == 255)
                    printf("IIC master read slave data compare ok, test success\n");
            }
            break;
        }
}
```