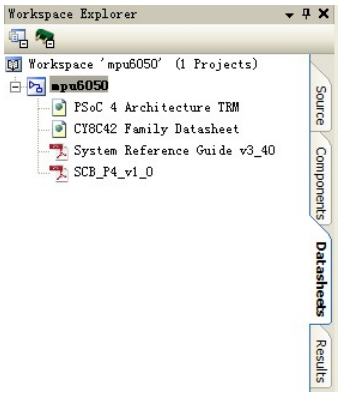
从零开始学PSOC4

1 如何查找API函数说明？



在SCB_P4_v1_0就是啦。看来赛普拉斯想的太周到了，真是站在工程师的角度想问题。

2 如何操作UART的API函数？

以前使用过PSOC3，但是这次还是忘记了把UART要换成UART component的名字了。

即在调用时：**void UART_Start(void)**

改为：**void UART_1_ Start (void)**

第一节 UART中断接收

PSOC4的UART模块特性：

带有硬件地址检测功能的 9 位寻址模式

波特率范围从110 到 921600 bps ，也可任意高达 4 Mbps

RX 和 TX 缓冲区大小范围有4字节 到 65535字节

帧检测、奇偶校验检测和溢出检测

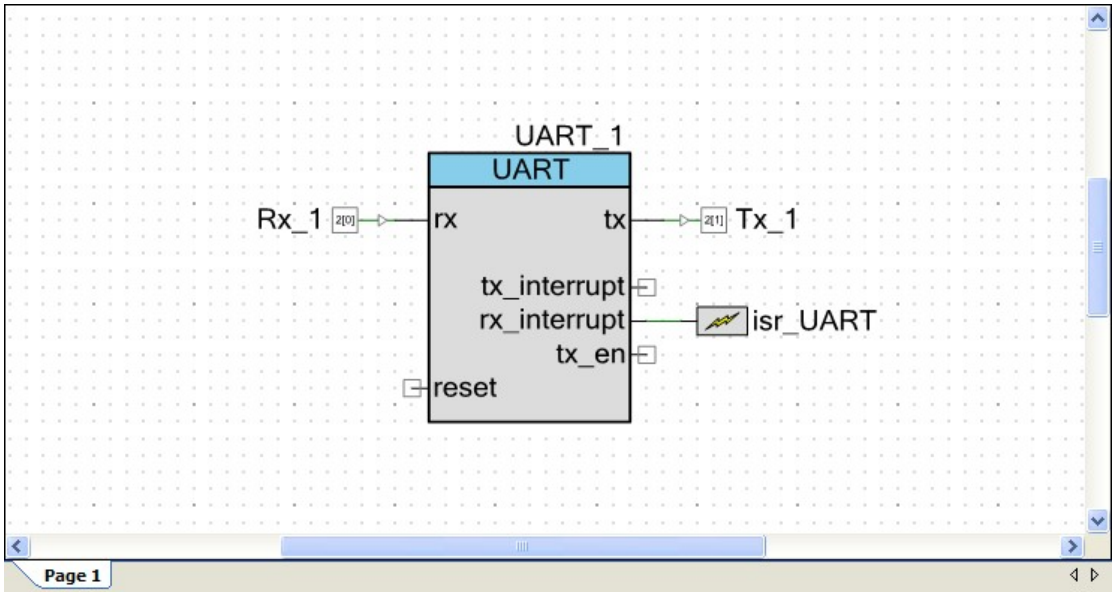
可优化的硬件选择，全双工、半双工、仅发送TX，和仅接收 RX

按位 3 取 2 表决

中断信号产生和检测

8倍（8x） 或 16倍（16x） 过采样

UART原理图：

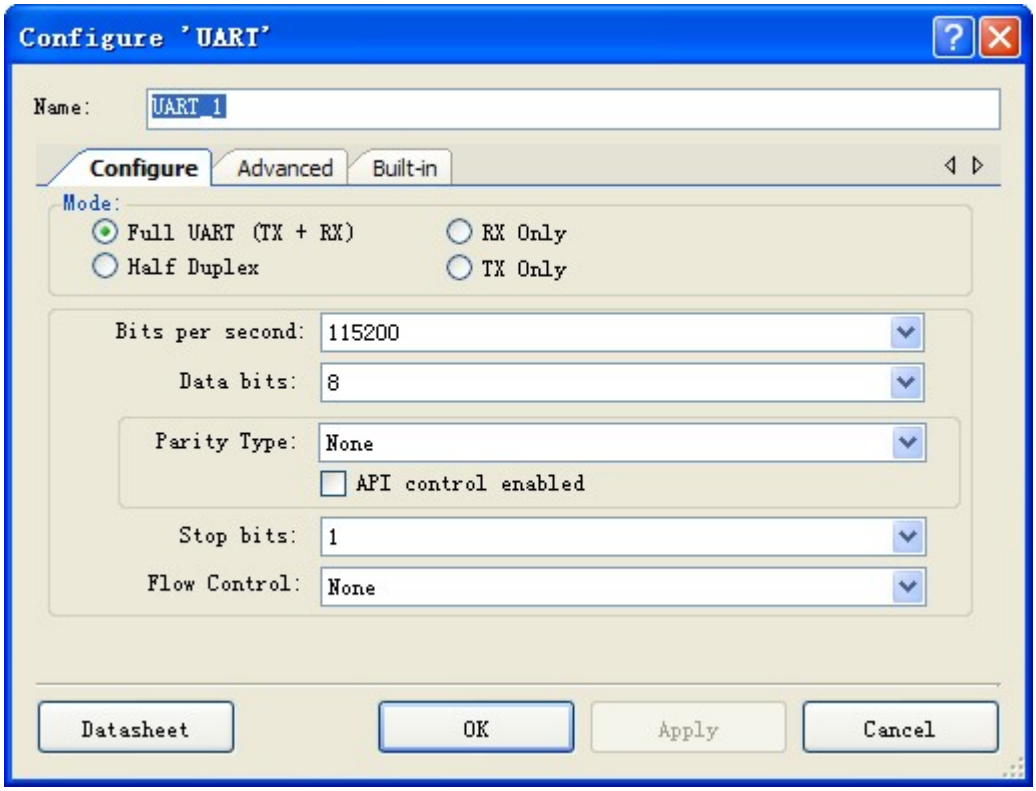


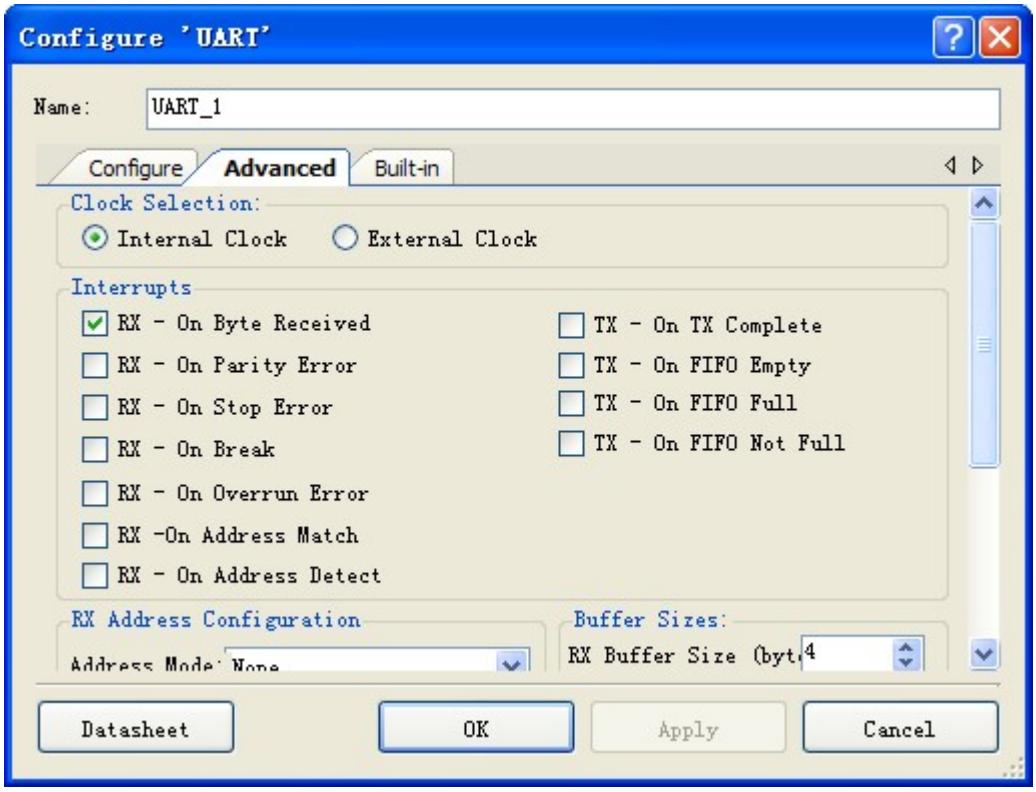
从原理图中可以看出

RX_1引脚：P2[0]

TX_1引脚：P2[1]

配置UART_1模块的方法是





特别说明：RX缓冲区大小（字节）

此参数定义分配给 RX 缓冲区的 RAM 字节数。数据从接收寄存器移至该缓冲区。

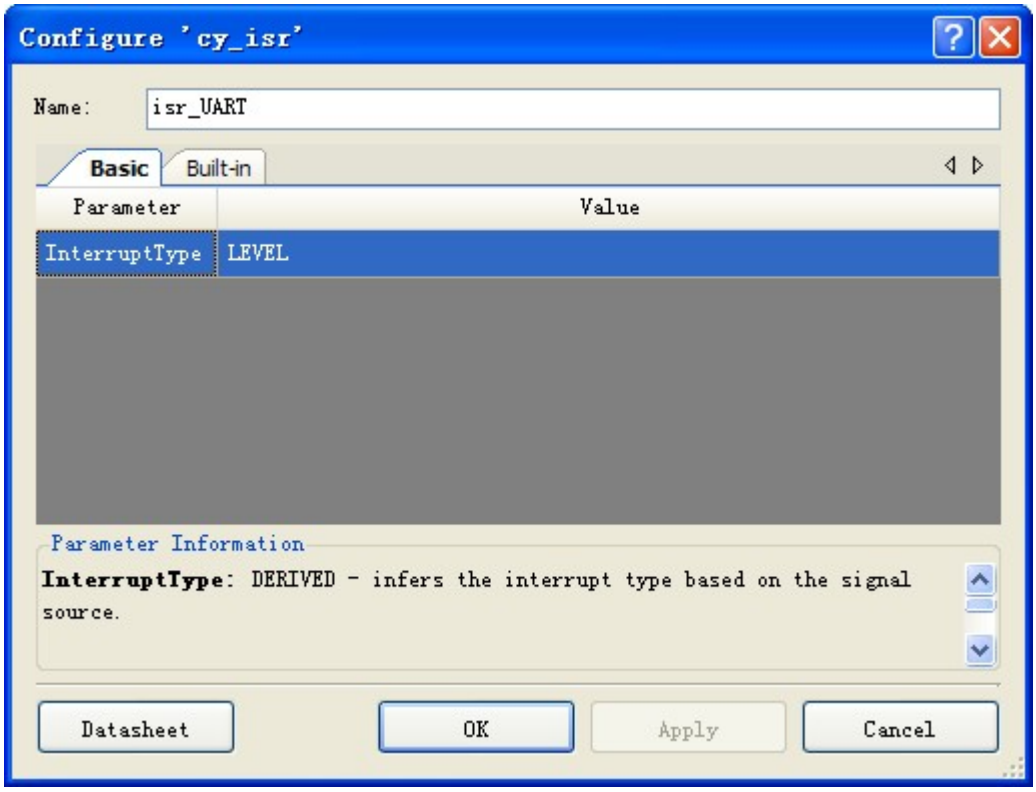
当所选的缓冲区大小是 4 字节时，硬件 FIFO 的四个字节将用作缓冲区。缓冲区大小超过 4 字节时，需要使用中断将数据从接收 FIFO 移动至此缓冲区。UART_GetChar() 或 UART_ReadRXData() API函数在不需要对顶层固件进行任何更改的条件下,可以从正确数据源获取正确数据。

当 RX 缓冲区大小超过 4 字节时，**Internal RX Interrupt ISR**（内部 RX 中断 ISR）将自动启用，并且 **RX - On Byte Received**（RX - 接收字节）中断源被选中并禁用，因为它会导致错误的处理功能。

然后添加isr_UART

方法是在右侧的Component Catalog中System->Interrupt[V1.70]

isr_UART的配置方法如图



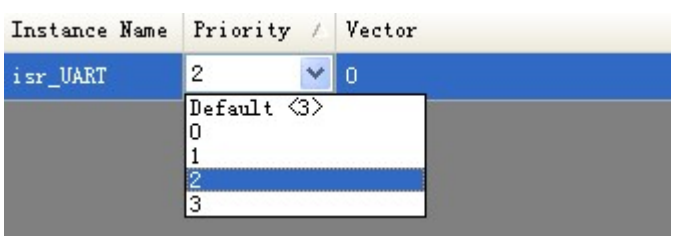
仅为基于UDB 的模块（如UART
和SPI）生成的中断选择LEVEL选项。其原因是这些模块通过生成一个电平中断来表示UD
B FIFO 缓冲区的状态信号。深度为4 个字节的FIFO 缓冲区用于暂时存储传输（Tx
FIFO）或接收的数据（Rx FIFO）。

对于Rx FIFO，只要存在最少一个数据字节可供CPU 读取，FIFO
状态信号便为高电平。如果选中了DERIVED 或RISING_EDGE
选项，即使缓冲区内有多个字节，该中断也只会触发一次。如果编写ISR
代码仅在每次中断时读取一个字节，那么缓冲区内的额外数据将有可能丢失。

接下来看一下DWR，进行引脚配置、中断优先级和时钟配置。

Alias	Name	Port	Pin	Lock
	Rx_1	P2[0]	2	☑
	Tx_1	P2[1]	3	☑

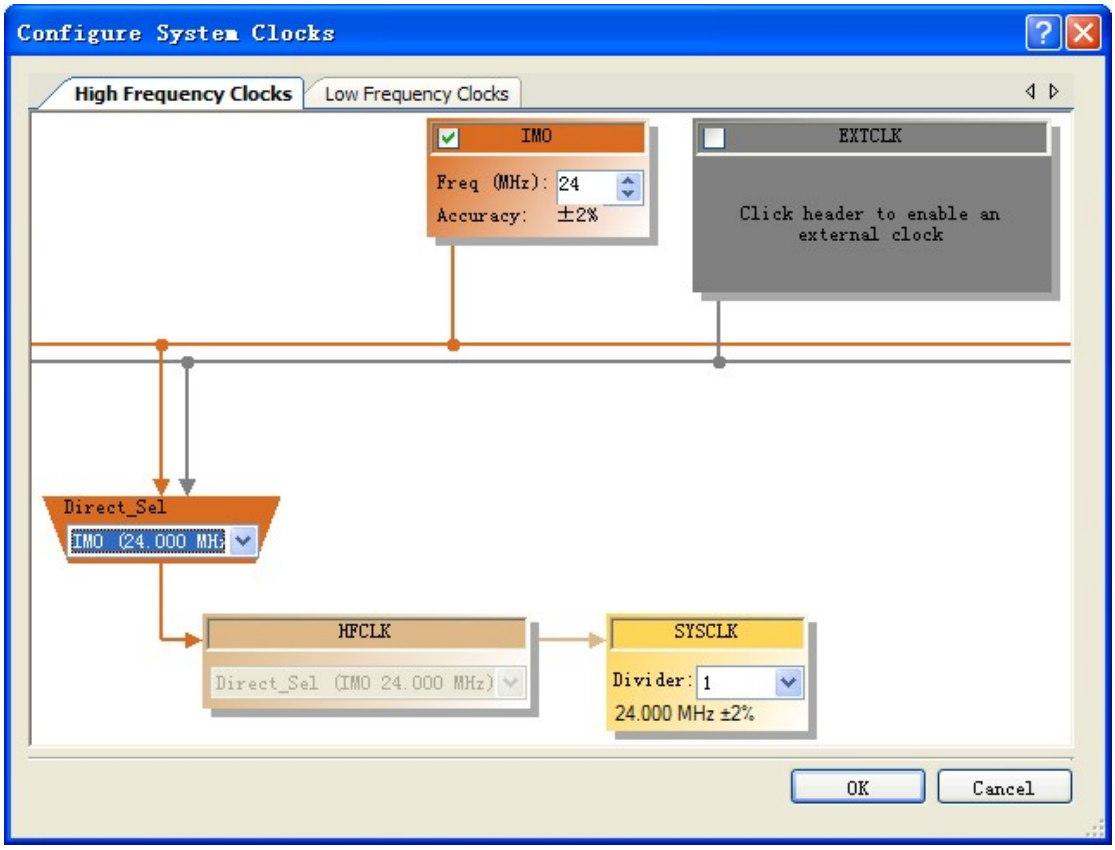
在Port选项中直接选取任意的引脚来作为Rx_1使用，本文选用P2[0]；
同样选取任意的引脚来作为Tx_1使用，本文选用P2[1]；



在优先级配置里，默认是<3>，也可以进入中断。
当然，0的优先级是最高的。

Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	EXTCLK	N/A	24.000 MHz	? MHz	±0	-	0	☐	
System	LFCLK	N/A	? MHz	32.000 kHz	±30	-	0	☑	ILO
System	ILO	N/A	32.000 kHz	32.000 kHz	±30	-	0	☑	
System	SYSCLK	N/A	? MHz	24.000 MHz	±2	-	1	☑	HFCLK
System	IMO	N/A	24.000 MHz	24.000 MHz	±2	-	0	☑	
System	HFCLK	N/A	24.000 MHz	24.000 MHz	±2	-	0	☑	Direct_Sel
Local	UART_1_IntClock	DIGITAL	921.600 kHz	923.077 kHz	±2	±2	26	☑	Auto: HFCLK

时钟配置：这里使用的是内部24M时钟，双击System是可以修改频率的，



Direct_Sel：可以选择使用内部或是外部时钟。
SYSCLK：可以进行1,2,4,8,16,32,64,128分频。
IMO：可以调节4~48MHz之间的任意整数频率值。

源程序：

```
void main()
{
    /* Place your initialization/startup code here (e.g.
MyInst_Start()) */
    UART_1_Start();
    isr_UART_Start();
    UART_1_PutString("UART is OK\r\n");
```

```
    /* Enable global interrupts */
    CyGlobalIntEnable;

    for(;;)
    {

    }
}
```

中断服务程序：位置在Generated Source下面的isr_UART.c第138行。

```
CY_ISR(isr_UART_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START isr_UART_Interrupt` */
    if(UART_1_ReadRxData()=='1')
    {
    UART_1_PutString("RX_Interrupt is Right\r\n");
    }
    /* `#END` */
}
```

演示效果图：