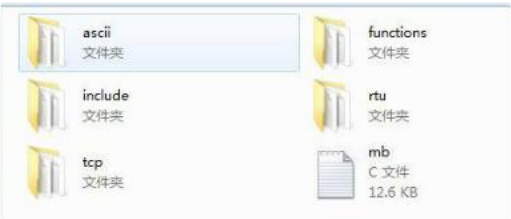
FreeModbus 的 MSP430 版本的 demo 是在 GCC 平台，现移植到 IAR 上，版本 5.2。

一. 新建 IAR 工程

FreeModbus V1.5 源码层次非常明显，**demo** 中存放的是 BSP 硬件层，其中 **port** 中存放定时器和串口驱动文件；**system** 中存放的 DCO 也就是 CPU 时钟初始化驱动，这部分和编译器紧密相关，有 GCC 和 ROWLEY 两个版本，我用 IAR 是无法完成 DCO 初始化的，所以我写了一个 DCO-IAR，后面会详细介绍。

Modbus 文件存放的是 modbus 的实现函数，modbus 有不同的功能因此有个文件 **functions**，所有的功能实现均在这个文件中。根据 modbus 报文格式和链路层的不同又有：ASCII、RTU、TCP 三种类型，因此有三个文件夹 **ASCII**、**RTU**、**TCP**，文件中存放对应的实现程序。用户可以根据自己的需要加载这些文件。相关的头文件存放在 **Include** 文件中。



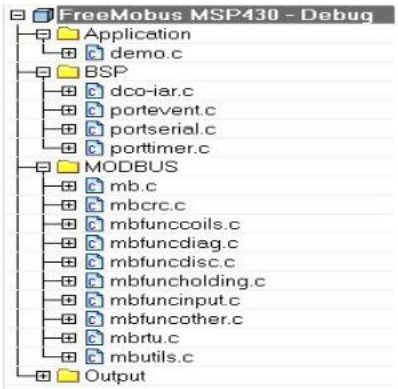
理清楚各个文件夹中存放的文件类型之后，接下来就是建立一个 IAR 工程，建立 3 个 Group 分别是 Application、BSP、MODBUS。

Group MODBUS 加载文件前需要搞清楚自己需要使用哪种类型的 modbus。我这个例程使用的是 RTU,mb.c 负责初始化和调度整个 MODBUS 功能模块；mbcrc.c 实现 CRC 校验；mbfunxxx.c 负责实现各种功能码；murtu.c 属于被 mb.c 调用的，实现 MB RTU 的初始化和调度；mbutil.c 是 funcation 实现的公共函数。调用层次为

mb.c---->murtu.c---->mbfunxxx.c--->mbutil.c

Group BSP 存放硬件定时器、串口、DCO 时钟的相关驱动，在 demo 文件夹找到 MSP430，打开就有 port 和 system，打开 port 文件夹将 portevent.c、portserial.c、porttimer.c、portevent.c 加载到 Group BSP，system 中是 DCO 相关驱动，自己要编写 dco-IAR.c

Group Application 存放主函数 demo.c，里面有主函数 mian；



二. 功能实现

dco-iar.c 的编写：主要是实现 *cTISetDCO* 函数，该函数就是上电之后设置系统时钟，delta 是设置时钟的频率，在 dco.h 中有时钟频率的代号，*cTISetDCO* 函数的声明也在 dco.h 中。

cTISetDCO 函数并没有遵循原作者的意图可以灵活的配置时钟，而是不管 delta 参数值，均将 CPU 主时钟设置为 8MHz。

新建文件：dco-iar.c

```
/* ---------------------- Platform includes ------------------------------*/
#include "port.h"
#include "dco.h"
char
cTISetDCO( int delta )
{
    char z,result;
    /*------选择系统主时钟为 8MHz-------*/
    BCSCTL1 &= ~XT2OFF;                         //打开 XT2 高频晶体振荡器
    do
    {
        IFG1 &= ~OFIFG;                         //清除晶振失败标志
        for (z = 0xFF; z > 0; z--);        //等待 8MHz 晶体起振
    }
    while ((IFG1 & OFIFG));                   //晶振失效标志仍然存在？
    BCSCTL2 |= SELM_2 + SELS;                  //MCLK 和 SMCLK 选择高频晶振
    result = TI_DCO_NO_ERROR;
    return(result);
}
```

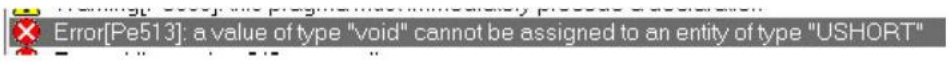
官方 demo 使用的 MCU 为 MSP430F169，我用的开发板是 F149 因此要更改相应的库函数。

文件 port.h.

更改为：

```
/* ---------------------- Platform includes ------------------------------*/
#include <msp430x14x.h>
#if defined (__GNUC__)
#include <signal.h>
#endif
#undef CHAR
```

Portserial.c 中 usOldSR = _DINT( )程序，_DINT()是没有返回值的所以编译会报错，



将该部分程序更改为这样

文件：Portserial.c---EnterCriticalSection(void)

更改为：

```
void
EnterCriticalSection( void )
{
    USHORT usOldSR;
    if( ucCriticalNesting == 0 )
    {
```

```
#if defined (__GNUC__)
            usOldSR = READ_SR;
            _DINT( );
#else
//            usOldSR = _DINT( );
#endif
//            ucGIEWasEnabled = usOldSR & GIE ? TRUE : FALSE;
            _DINT( );
            ucGIEWasEnabled = TRUE;
      }
      ucCriticalNesting++;
}
```

文件：Portserial.c--- ExitCriticalSection(void)

更改为：

```
void
ExitCriticalSection( void )
{
      ucCriticalNesting--;
      if( ucCriticalNesting == 0 )
      {
            if( ucGIEWasEnabled )
            {
                  ucGIEWasEnabled = FALSE;
                  _EINT(    );
            }
      }
}
```

更改中断服务函数，原程序的中断服务函数格式不能在 IAR 中使用，因此要更改为 IAR 格式

文件：portserial.c---prvvMBSerialRXIRQHandler(void)

更改为：

```
#pragma vector = USART0RX_VECTOR
__interrupt
void
 prvvMBSerialRXIRQHandler( void ) //__interrupt[USART0RX_VECTOR]
{
      DEBUG_TOGGLE_RX( );
      pxMBFrameCBByteReceived(    );
}
```

文件：portserial.c--- prvvMBSerialTXIRQHandler (void)

更改为：

```
#pragma vector = USART0TX_VECTOR
__interrupt
void
```

```
prvvMBSerialTXIRQHandler( void )    // __interrupt[USART0TX_VECTOR]
{
    DEBUG_TOGGLE_TX( );
    pxMBFrameCBTransmitterEmpty(    );
}
```

文件：porttimer.c--- prvvMBTimerIRQHandler (void)

更改为：

```
#pragma vector = TIMERA0_VECTOR
__interrupt
void
 prvvMBTimerIRQHandler( void )
{
    ( void )pxMBPortCBTimerExpired(    );
}
```

Demo.c 中有 4 个配置宏，REG_INPUT_START 定义输入寄存器的开始地址，这里我配置成 0；REG_INPUT_NREGS 定义输入寄存器的数目，我配置为 4 个；REG_HOLDING_START 定义保持寄存器的开始地址，这里我配置为 0；REG_HOLDING_NREGS 定义保持寄存器的个数，这里我定义为 125。

文件：demo.c

更改为：

```
/* ----------------------- Defines ------------------------------------------*/
#define REG_INPUT_START     0
#define REG_INPUT_NREGS     4
#define REG_HOLDING_START 0
#define REG_HOLDING_NREGS 125
```

这里配置 modbus 支持的工作模式，有 RTU、ASCII、TCP 三种模式。0 表示失能，1 表示使能。这里我只开启 RTU 模式。

文件：mbconfig.h

更改为：

```
/*! \brief If Modbus ASCII support is enabled. */
#define MB_ASCII_ENABLED                    (  0 )
/*! \brief If Modbus RTU support is enabled. */
#define MB_RTU_ENABLED                      (  1 )
/*! \brief If Modbus TCP support is enabled. */
#define MB_TCP_ENABLED                      (  0 )
```

到这一步新建的工程可以编译成功，接下来将测试功能是否正常，在测试之前先对 modbus 模块参数进行配置。

文件：demo.c--- main(void)

更改为：

```
/* ----------------------- Start implementation -----------------------------*/
int
main( void )
{
    eMBErrorCode       eStatus;
```

```
    volatile USHORT usACLKCnt;

    /* Stop Watchdog Timer. */
    WDTCTL = WDTPW + WDTHOLD;

    /* Delay for ACLK startup. */
    for( usACLKCnt = 0xFFFF; usACLKCnt != 0; usACLKCnt-- );
    if( cTISetDCO( TI_DCO_8MHZ ) == TI_DCO_NO_ERROR )
    {
        _EINT(  );
        /* Initialize Protocol Stack. */
        if( ( eStatus = eMBInit( MB_RTU, 0x01, 0, 115200, MB_PAR_EVEN ) ) != MB_ENOERR )
        {
        }
        /* Enable the Modbus Protocol Stack. */
        else if( ( eStatus = eMBEnable(  ) ) != MB_ENOERR )
        {
        }
        else
        {
            for( ;; )
            {
                ( void )eMBPoll(  );
                /* Here we simply count the number of poll cycles. */
                usRegInputBuf[0]++;
            }
        }
    }
    for( ;; );
}
```

这里是对 modbus 的串口参数以及地址的配置，这里我配置为 MB_RTU, 0x01, 0, 115200, MB_PAR_EVEN，这时 modbus 工作在 RTU 模式，节点地址为 01，使用的串行端口为 0 也就是 USART0，波特率 115200，偶校验。

文件：port.h

更改为：

```
#define SMCLK                           ( 8000000UL )
#define ACLK                            ( 32768UL )
```

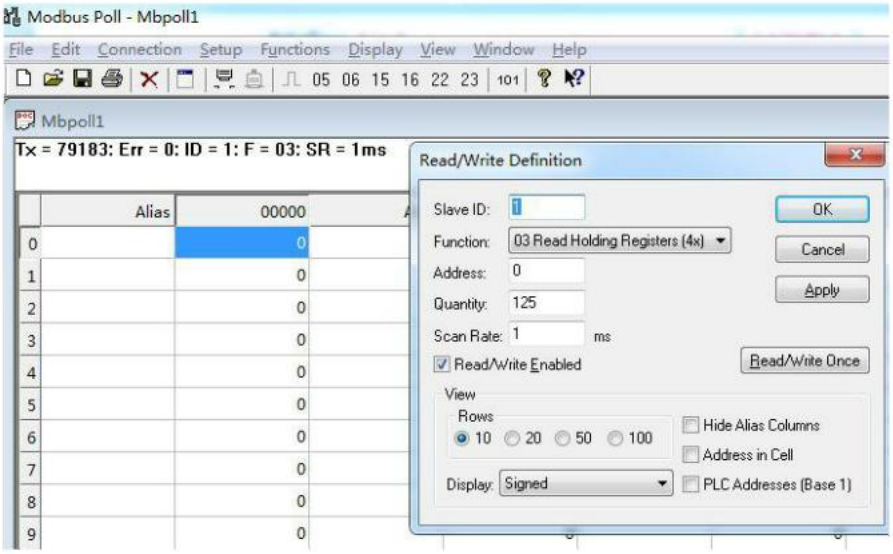
这里定义系统时钟频率，我的开发板使用外部 8MHz 晶振，在 dco-IAR.c 中的 cTISetDCO 函数中我将 MCU 初始化为 8MHz，因此 SMCLK = 8000000 ，这个参数非常重要因为串口波特率和以及定时器的计算均参考 SMCLK 的设置。

三.　　功能测试

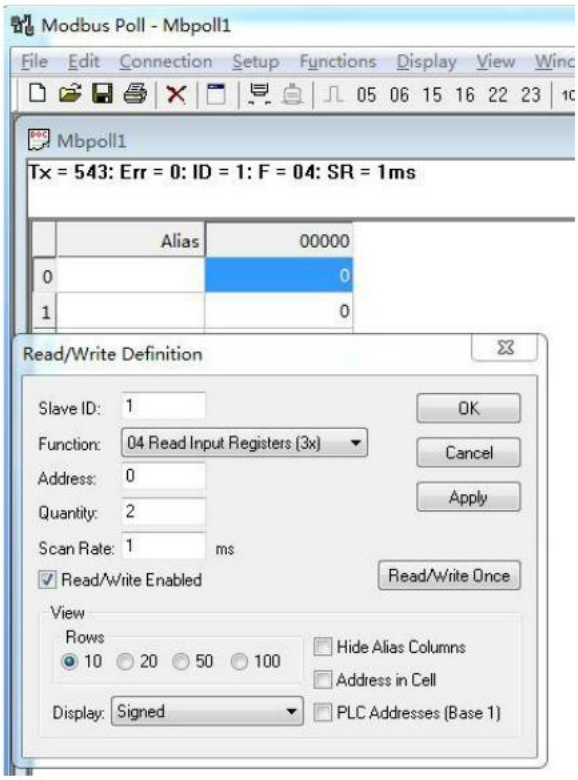
完成以上两个步骤之后，编译程序，烧录到开发板，连接好硬件开始功能测试。

FreeModbus 是一个从机的协议栈，因此还需要一个主机来完成功能测试，这里我使用 Modbus Poll 软件版本 4.34 来做主机。

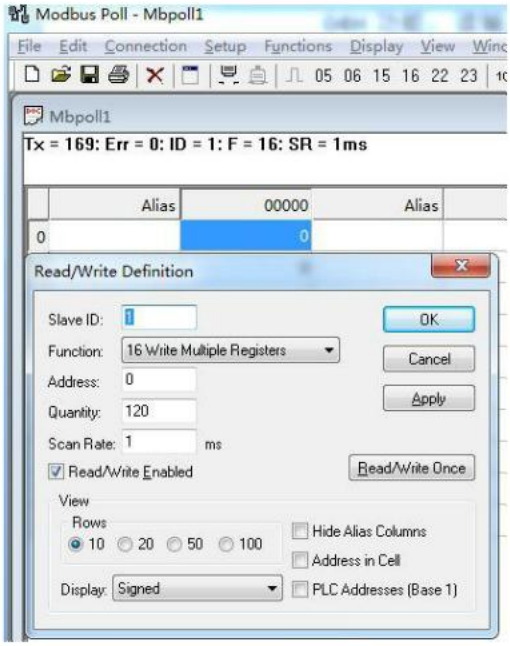
03H 功能，读保持寄存器



04H 功能，读输入寄存器



16H 功能，写多个寄存器



用户可以在 demo.c 中的 eMBRegInputCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNRegs)，eMBRegHoldingCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNRegs, eMBRegisterMode eMode)回调函数中完成对相应寄存器值的刷新。

另附：完整工程（@IAR FOR 430 V5.2）和 MODBUS Poll 调试工具

链接：http://pan.baidu.com/s/1hqBu04k　百度网盘