小梅哥 AC620 FPGA 开发板进阶设计教程

全功能高性价比 FPGA 开发板 AC620 火热销售中，配套 800 页原创电子书。

购买链接：

https://item.taobao.com/item.htm?spm=a1z10.1-c-s.w4004-13687301132.6.8uPVJ6&id=544830995588

AD7606 SOPC 应用手册

本文所述文件均可通过以下链接下载

https://wiki.analog.com/_media/resources/fpga/altera/ced1z/ad7606_evalboard.zip

AD7606 应用简介

AD7606 是 ADI 公司针对电源设计，工业检测，闭环控制等工业应用推出的一款，16 位，八通道同步并行采样 adc 芯片，采样率最高为每通道 200KSPS，输入电压范围+-5V，最高可达+-10V，该芯片满足工业级应用的温度范围。

芯片对外提供 spi 和并行的数字接口。当 AD7606 的 8 个通道全部以 200KSPS 的最高速率进行转换时，数据输出速率达到了 25.6Mbps，需要使用高性能 MCU 的 SPI 外设才能勉强满足该速率要求。因此可以使用 16 位并口来进行数据的传输，提高数据传输速率。

当 ad7606 应用在 FPGA 系统中的时候，使用 SPI 串行接口和并行接口都能够轻松的满足数据传输的速率需求。当在 FPGA 系统上应用 AD7606 时，可以通过在 FPGA 上设计 ad7606 的转换控制逻辑，将转换结果数据直接存储到片上的存储器如 fifo 或者 RAM 中，也可以存储到 FPGA 片外的存储器如 SRAM 或 SDRAM 中，然后由其他主控芯片如 MCU 或 DSP 读出，或者直接在 FPGA 内部进行数据的运算和处理。当然，由于 FPGA 片上可以设计软核控制器，也可以直接使用软和控制器完成数据的处理和传输工作，如典型的，在 Intel FPGA 器件上使用 NIOS II 软核控制器，将 AD7606 的转换结果通过串口或以太网传输到电脑上，也可以直接将数据显示在液晶显示屏上。

在这里想先说明下使用 NIOS II 软核控制器完成上述应用的优势。

以一个电力系统里面常见的应用——电力故障录波为例，该应用要求能够对 50HZ 的工频交流电的电流和电压波形进行采样，假设每个周波采样 2048 个点，则需要 102.4KSPS 的采样率，8 个通道同步采样，且能够存储 10 个周波的数据，因此需要的存储器空间为 102.4*8*10*16=131072Kbit 的存储空间，即 128Mbit 的存储空间，这对于任何一个 MCU 存储器来说都是很难实现的，而使用 FPGA，可以连接外部的 SDRAM 存储器来完成数据的存储。同时，使用 NIOS II CPU 可以通过简单的 C 语言编程对数据进行预处理，或者直接通过以太网协议传输。如果要使用外部的处理器例如 STM32 或者 DSP 将数据读取之后再进行运算和处理，中间会多出一个数据从 FPGA 传输到 STM32 或 DSP 的过程，会造成一定的延迟，因此，当前很多公司都在使用 NIOS II 软核的方案完成该应用。

基于 NIOS II 的 AD7606 控制器

本节将针对 ADI 公司提供的 AD7606 的控制器 IP 核进行分析介绍，介绍包括该 IP 核的寄存器结构和每个寄存器的功能，以及提供的 NIOS II 程序驱动文件和相关函数介绍，最后，通过三个具体的应用实例展示该 IP 核的使用方法，并提供三个应用的工程文件，方便用户快速移植到自己的系统中进行使用。

AD7606 控制器寄存器介绍

AD7606 控制器共有 5 个寄存器，分别为：

偏移	寄存器名	RW	寄存器功能
0	control_register	RW	AD7606 控制器控制寄存器 ◆ bit[0]：转换启动信号，为 1 时启动转换 ◆ bit[1]：复位控制位，为 1 复位系统 ◆ bit[2]：固定写地址，为 1 时所有数据都将写入同一地址 ◆ bit[3]：AD7606 写控制信号使能位，为 1 时，dut_write_register 中的值将被写入 AD7606 控制逻辑
1	acq_count_register	RW	软件设置的需要转换的次数，通过写该寄存器，设置 AD7606 控制逻辑控制 ADC 执行多少次转换。控制逻辑内部有个计数器，开始转换后每执行完成一次转换，内部计数器减 1，内部计数器计数到 0，则停止转换。
2	base_register	RW	AD7606 IP 转换逻辑写 RAM 起始地址，转换结果使用 Avalon MM Master 总线直接写入 RAM 中，base_register 指定了写入到 RAM 中的地址。
3	status_register	R	AD7606 控制器状态寄存器 ◆ bit[0]：转换完成标志信号，为 1 表明 AD 转换任务完成 ◆ bit[1]：Avalon MM Master 缓冲 fifo 满标志信号，该信号为 1 时表示写 RAM 的 Avalon MM Master 缓冲 fifo 已经满了。通过复位清零该信号。 ◆ bit[2]：控制地址写错，支持写入的地址只有 0、1、2、4，如果控制总线（Avalon MM Slave）产生的写地址超出了这个范围，则表明写地址信号出错，该位置 1，通过复位清零该信号。
4	dut_write_register	RW	AD7606 控制器模式寄存器 ◆ bit[7:5]: channel，注意，这个值是读取的通道个数，如果为 n，则读取第 1~n 通道的数据 ◆ bit[4:2]: os, ◆ bit[1]: standby，注意，该位也同时控制 AD7606 控制器的工作状态，如果一次转换完成后没有将该寄存器对应位清零，AD7606 控制器将继续控制 AD7606 芯片进行数据转换，只是不会将结果写入 RAM 中，这种情况会白白消耗 FPGA 和 AD7606 的功耗。因此建议在不需要采集的时候将该位置 0 ◆ bit[0]: range

◆ AD7606 支持软件和硬件复位，通过写控制寄存器的 bit[1] 为 1，可以软件复位 AD7606 控制器。

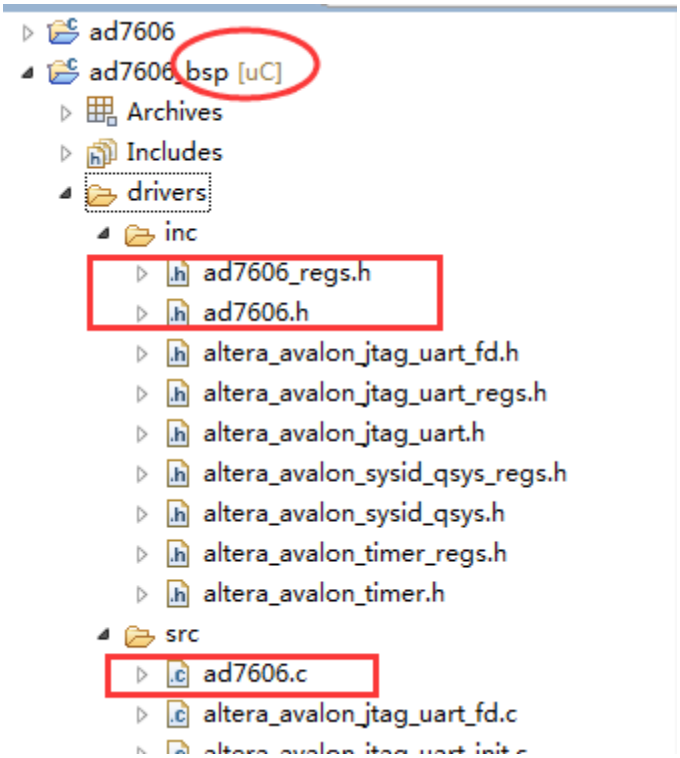
◆ AD7606 控制器可以设置 AD7606 IC 的工作模式，包括输入电压范围，过采样，是否旁

路等，不过这些功能需要模块物理上的支持，即需要 AD7606 芯片的 OSC、standby、range 引脚都与 FPGA 相连了，能够直接被 FPGA 控制，否则无法实现软件控制功能。通过写 dut_write_register 寄存器以支持这些功能。

- AD7606 控制器带有一个 Avalon MM Master 接口，可以直接将转换结果写入 Avalon MM Slave 接口的存储器中，如 onchip ram、FIFO、SRAM、SDRAM、DDR2 中。无需 CPU 去从控制器读取数据，也无需使用 DMA 控制器进行数据传输。或者说，相当于该控制器本身就带有了 DMA 控制器。该控制器会将数据直接写入存储器中。
- AD7606 控制器写入 RAM 中的地址是可以通过软件设置的。base_register 的值即为 AD7606 控制器主动写 RAM 的基地址。
- AD7606 支持可软件编程的数据采集的个数，通过设置 acq_count_register 寄存器实现。软件设置的需要转换的次数，通过写该寄存器，设置 AD7606 控制逻辑控制 ADC 执行多少次转换。控制逻辑内部有个计数器，开始转换后每执行完成一次转换，内部计数器减 1，内部计数器计数到 0，则停止转换。注意，该寄存器指定的是执行多少个数据的转换，例如，设置读取的通道个数为 2，即采集通道 1 和通道 2 的值，设置 acq_count_register 寄存器为 16，则会分别采集通道 1 和通道 2 的值各 8 次，总共 16 个 16 位数据。而设置读取的通道个数为 8，即采集通道 1~8 的值，设置 acq_count_register 寄存器为 16，则会分别采集通道 1~8 的值各 2 次，总共 16 个 16 位数据。

AD7606 软件驱动说明

AD7606 控制器提供了有完整的寄存器映射文件和驱动函数，以及一个可以使用 system console 进行调试的设计实例工程。在 AD7606 控制器 IP 的文件包中，包含了有脚本，可以使 NIOS II EDS 软件在创建 BSP 工程时自动将 AD7606 控制器的驱动文件加载进 BSP 工程，加载进去之后，在 bsp 工程的 drivers/inc 下可以找到 ad7606_regs.h 和 ad7606.h 两个头文件，在 drivers/src 下可以找到 ad7606.c 这个驱动程序源文件。



ad7606_regs.h

该文件为AD7606控制器的寄存器映射文件，文件中提供了底层的直接操作控制器中各个操作寄存器的函数，例如

IOADDR_AD7606_CONTROL(base)	计算AD7606控制器的控制寄存器地址
IORD_AD7606_CONTROL(base)	读取AD7606控制器的控制寄存器地址
IOWR_AD7606_CONTROL(base, data)	将data值写入AD7606控制器的控制寄存器

ad7606.h

该文件为AD7606控制器的C语言驱动程序头文件，包括了各种参数定义以及C源程序中使用到的函数的定义

ad7606.c

该文件为AD7606控制器的C语言驱动程序，包含了4个函数，现就每个函数的功能以及用法介绍如下：

函数	说明
`void AD7606_Reset(void)`	软件复位AD7606控制器

`int AD7606_StartAcquisition(`
`int size,`
`int destination`
`)`
	开始执行转换，调用该函数是需要传递两个参数：
	● size：需要得到的转换的数据个数，N个通道的数据算N个数据，例如选择转换1~4通道，size值设置为16，则1~4通道每个通道会得到4个转换结果。
	● destination：转换结果存储的目的地址起始地址，AD7606控制器会将转换的结果直接写入存储器中以此地址开始的一片连续存储区内。
	● 返回值：1表示传递进来的存储目的地址值超出了指定范围，未执行转换，直接退出。0表示转换正常启动。

`int AD7606_ReadStatus (void)`	读取AD7606控制器状态寄存器，并将读取结果作为返回值返回。

`void AD7606_WriteRegister (`
`int value`
`)`
	写AD7606控制器的模式寄存器，该寄存器控制执行转换时的一些属性，如转换通道个数，详细内容可以参见dut_write_register寄存器的说明

设计实例：基于 NIOS II 的连续数据采集串口发送 PC 显示

特别说明：

在程序发布初期，有网友提出过这样的问题：同样是使用串口传输采样结果数据到电脑端，和 Verilog 直接采样并使用串口实时发送有什么区别呢？

其实，无论是使用 Verilog 直接采样并串口传输数据，还是使用 NIOS II 传输数据，都分为两种情况，低采样速率实时传输和高采样速率批量采样传输。

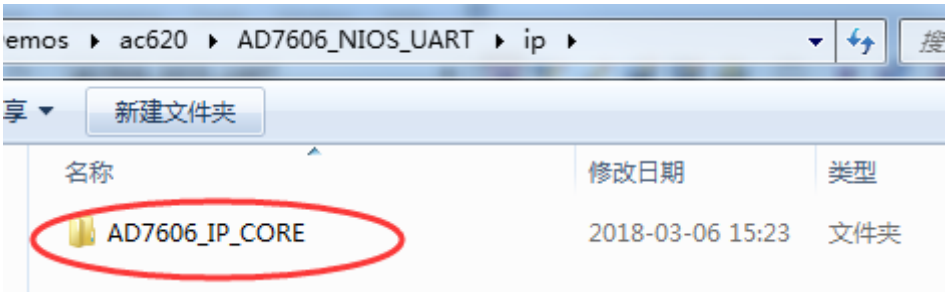
低采样速率实时传输：所谓低采样速率实时传输，是以串口的最大传输速率作为采样速率。因为串口的数据传输速率有限，标准的 RS232 接口，允许的最大波特率为 115200bps，

而一个通道的数据为 16 位，所以 1 秒钟之内最多能够传输 5760（115200/（(8+2）*2））个 16 位采样数据。所以，要想串口能够实时传输最近一次的采样结果，则 ADC 的采样速率最高为 5760sps，两个相邻采样点间的间隔为 173.6us，此种情况适合设计基于串口的实时电压采集显示系统，不适合希望以高采样率采集一段连续时间内的电压数据。

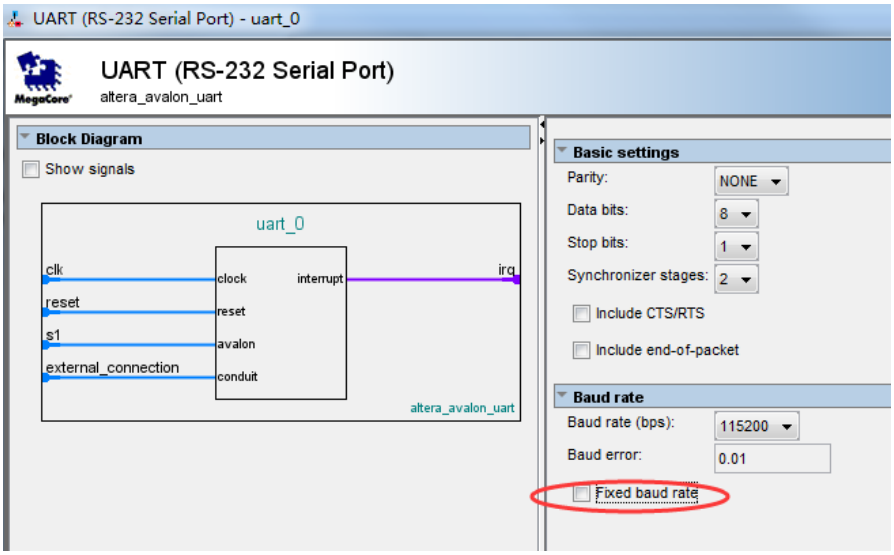
高采样速率批量采样传输：所谓高采样速率批量采样传输，使用的是先存储后传输的思想，例如，以 200KSPS 的采样率对单个通道连续采样 5760 个数据，仅需约 34.72ms，而串口传输 5760 个数据需要 1 秒，因此采用先采样后传输的方式，将数据先全部采集进入存储器中，然后再从存储器中将采样结果逐个使用串口发送。虽然还是 1 秒钟内仅传输了 5760 个采样结果，但是不同于低采样速率实时传输方案中每两个采样点间的间隔为 173.6us，高采样速率批量采样传输中想领两个采样点间的间隔为 5us。此种采样方案适合需要以较高的采样率采集一段数据分析瞬态波形的应用。

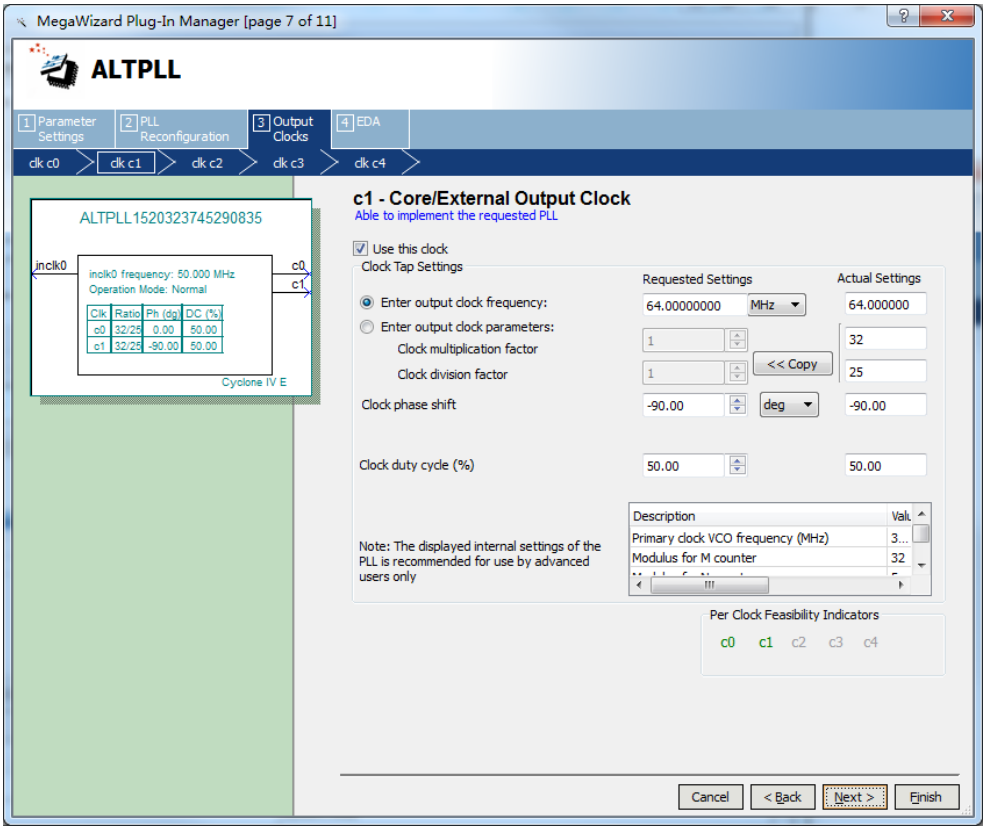
无论是使用Verilog直接编写硬件逻辑代码，还是使用NIOS II软核，都能实现这两种采样方式，本例重点介绍使用NIOS II软核实现这两种采样方案。由于NIOS II操作对于很多网友来说都并不十分熟悉，因此本例以视频的方式展现完整的开发过程。方便网友后期根据本视频举一反三，设计自己的系统。

首先创建一个工程文件夹，如AD7606_NIOS_UART，并将Quartus工程创建在该目录中。接着在该目录下创建一个名为“ip”的空文件夹，然后从我们提供的ACM7606模块的资料包中将AD7606_IP_CORE文件夹拷贝到ip目录下

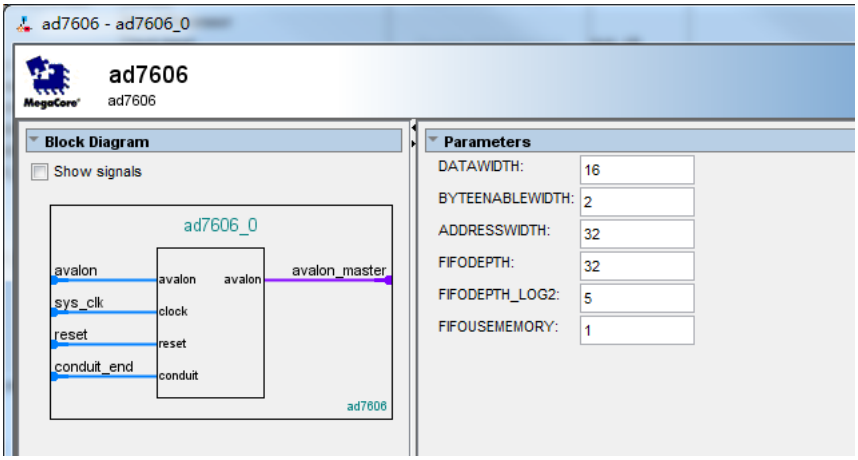


在Quartus软件中打开Qsys工具，依次添加NIOS II CPU、SDRAM控制器、UART控制器和Avalon PLL。使用PLL输出2路64MHz，相位相差90度的时钟信号，一路用作内部逻辑工作时钟，一路提供给外部SDRAM芯片。

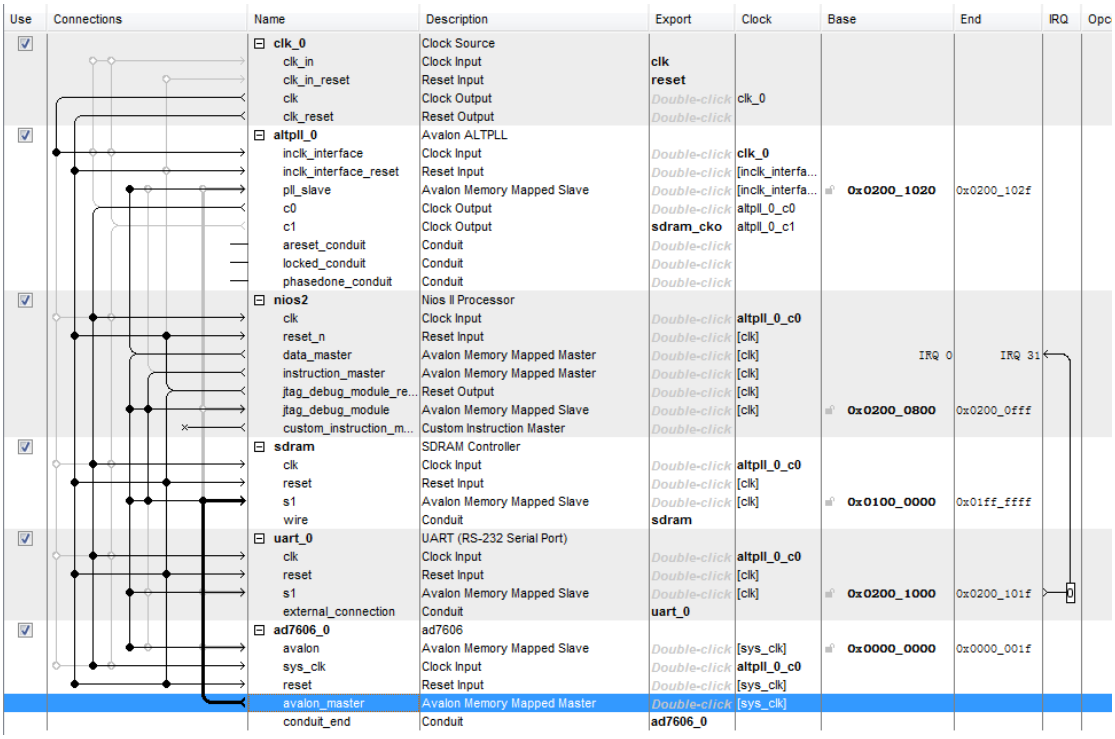




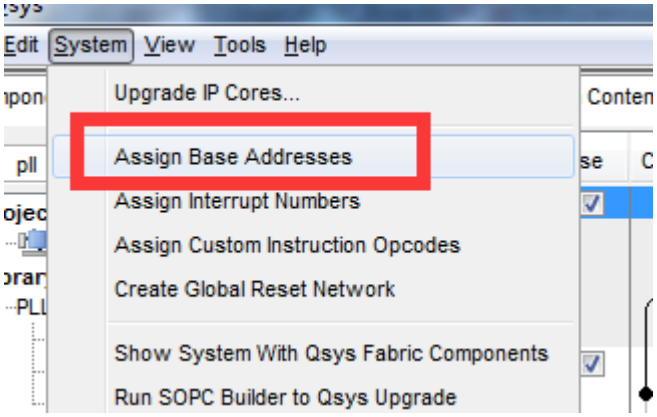
最后添加AD7606控制器。需要注意的是，需要修改控制器的DATAWIDTH值为16，BYTEENABLEWIDTH值为2，当然，这是在AC620开发板上，使用16位的SDRAM存储器作为数据存储器时。如果将采集的数据存储到片上32位总线位宽的存储器或者外部DDR2（如AC6102开发板上使用1片DDR2存储器，则DDR2控制器的数据总线位宽为32位）等32位数据位宽的存储器上时，就需要设置DATAWIDTH值为32、BYTEENABLEWIDTH值为4

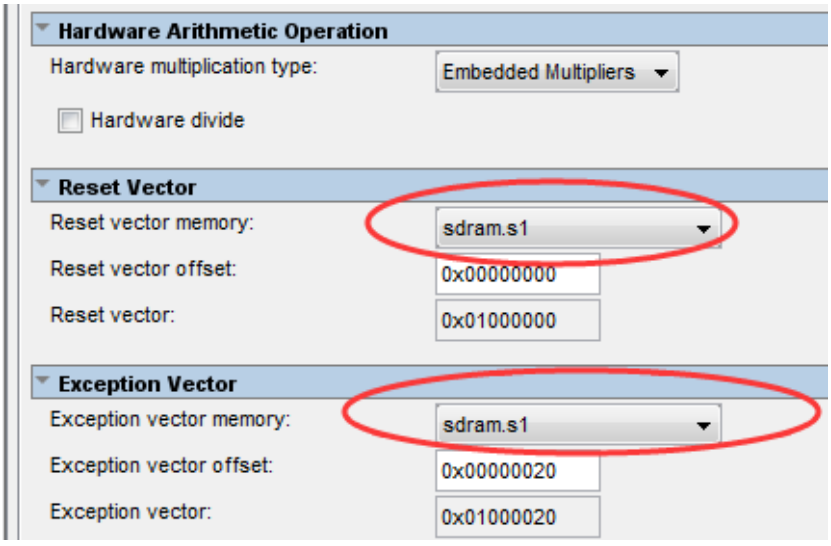


添加完成后，连接时钟、复位和Avalon MM网络。需要注意的是，AD7606控制器除了有一个Avalon MM Slave总线，还有一个Avalon MM Master总线，需要将该总线连接到SDRAM的s1端口上。最后将PLL的C1输出导出为顶层端口，命名为sdram_cko，然后导出SDRAM、UART、AD7606控制器的扩展端口到顶层。



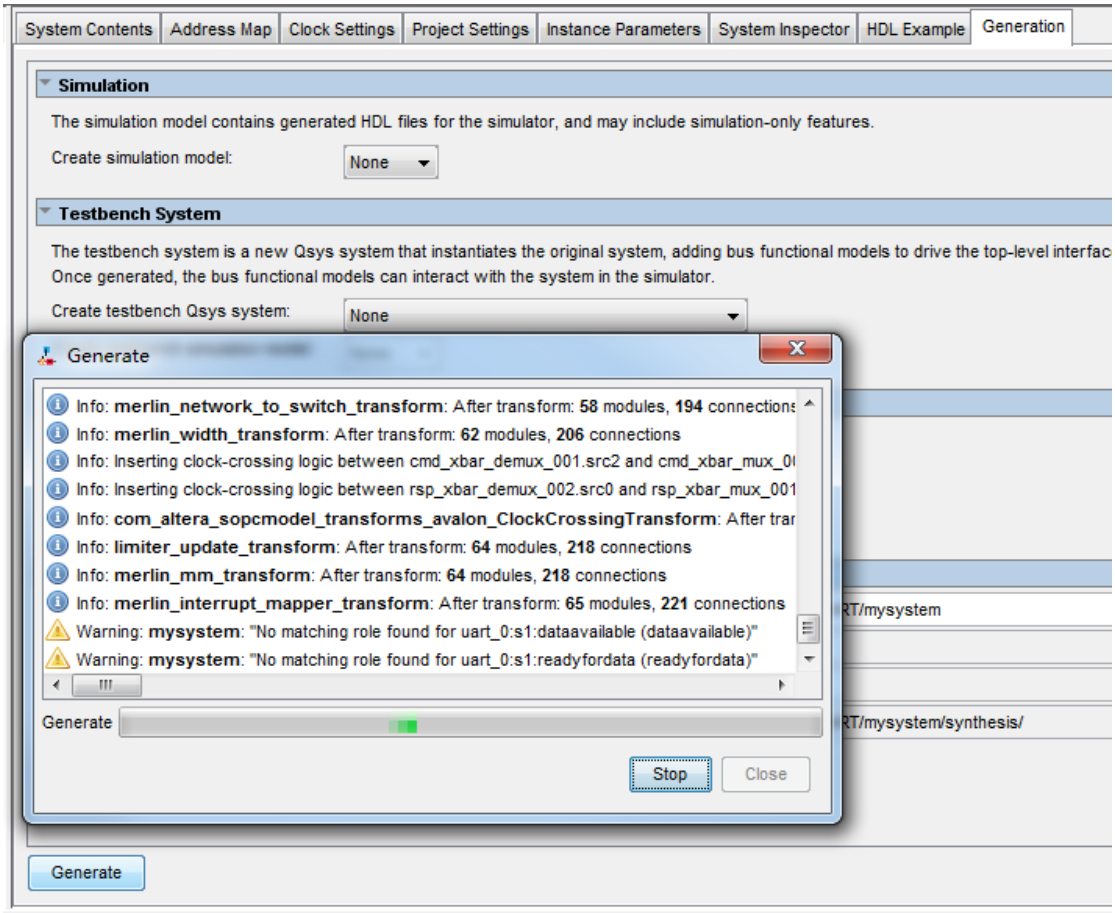
然后执行Assign Base Address命令分配基地址。最后双击NIOS II CPU，修改reset vector和exception vector为SDRAM即可。





至此，整个Qsys系统就搭建完成了，保存设计，命名为mysystem.qsys，接下来切换到HDL Example选项中，点击Copy复制代码到剪切板中。

然后generate生成HDL文件。

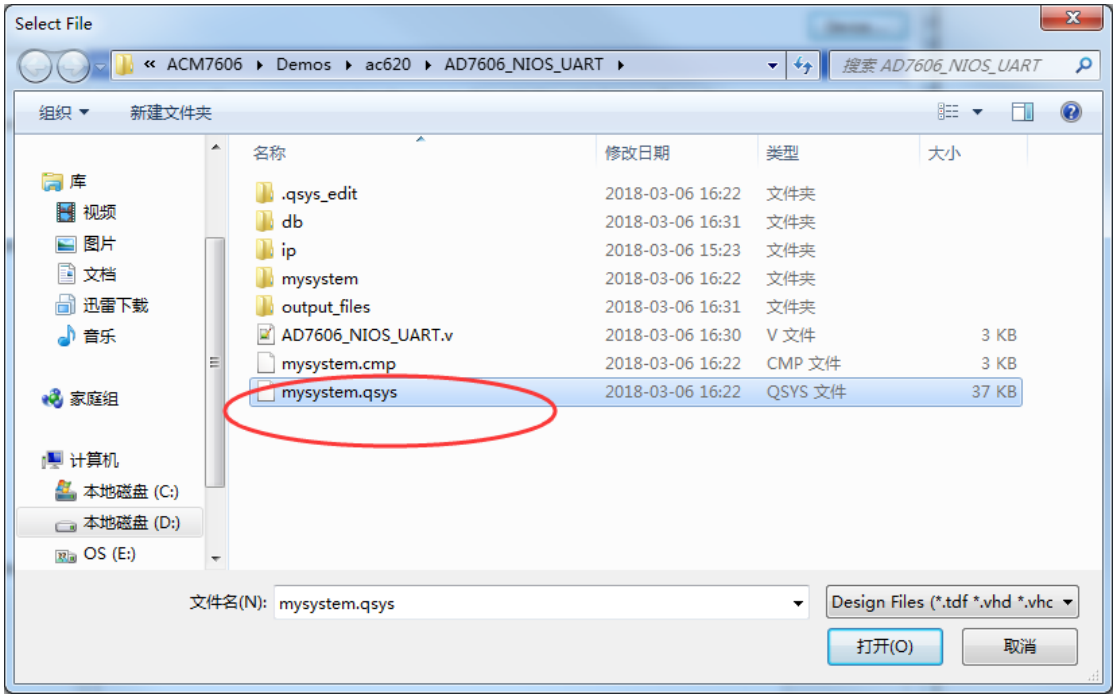


回到Quartus软件中，创建一个名为AD7606_NIOS_UART的Verilog文件，添加下述代码并保存。

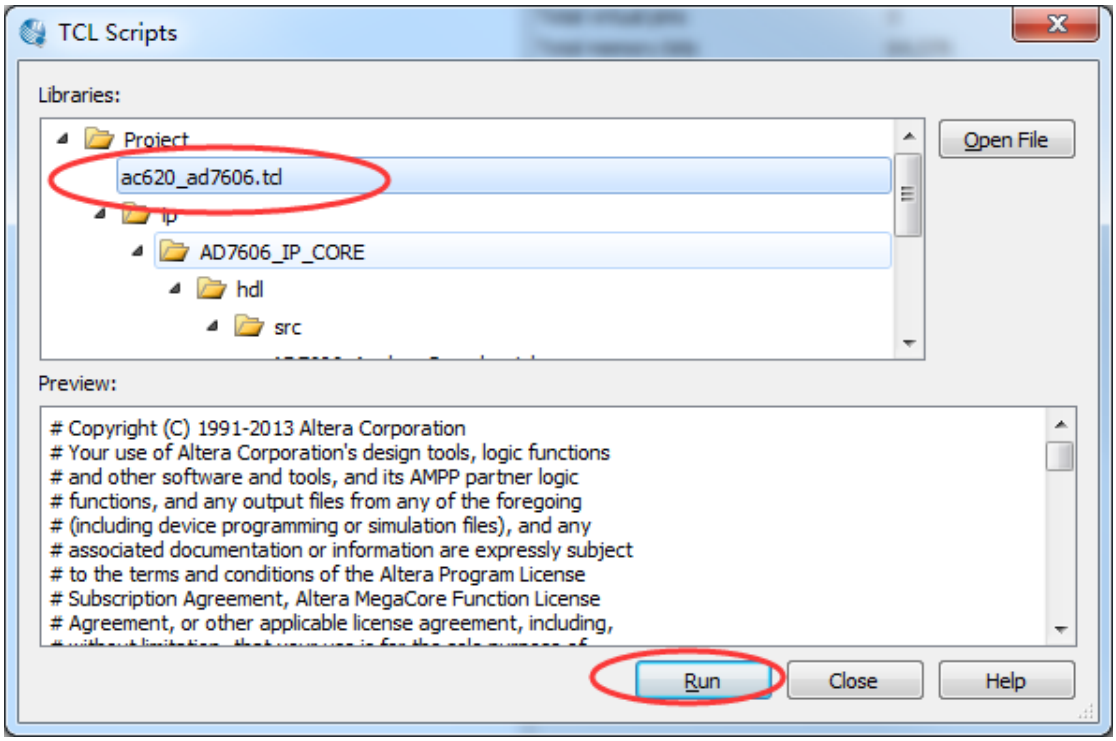
```
module AD7606_NIOS_UART(
    input  wire        reset_n,
    input  wire        clk,
    output wire        ad7606_0_cs_n_o,
    output wire        ad7606_0_rd_n_o,
    input  wire        ad7606_0_busy_i,
    input  wire        ad7606_0_firstdata,
    input  wire [15:0] ad7606_0_db_i,
    output wire [2:0]  ad7606_0_os_o,
    output wire        ad7606_0_range_o,
    output wire        ad7606_0_reset_o,
    output wire        ad7606_0_convst_o,
    output wire        ad7606_0_stdby_o,
    output wire        sdram_clk,
    output wire [11:0] sdram_addr,
    output wire [1:0]  sdram_ba,
    output wire        sdram_cas_n,
    output wire        sdram_cke,
    output wire        sdram_cs_n,
    inout  wire [15:0] sdram_dq,
```

```
    output wire [1:0]  sdram_dqm,
    output wire        sdram_ras_n,
    output wire        sdram_we_n,
    output wire         uart_0_txd,
    input  wire         uart_0_rxd
);

    mysystem u0 (
        .reset_reset_n     (reset_n),
        .clk_clk          (clk),
        .ad7606_0_cs_n_o   (ad7606_0_cs_n_o),
        .ad7606_0_rd_n_o   (ad7606_0_rd_n_o),
        .ad7606_0_busy_i   (ad7606_0_busy_i),
        .ad7606_0_db_i     (ad7606_0_db_i),
        .ad7606_0_os_o     (ad7606_0_os_o),
        .ad7606_0_range_o  (ad7606_0_range_o),
        .ad7606_0_reset_o  (ad7606_0_reset_o),
        .ad7606_0_convst_o (ad7606_0_convst_o),
        .ad7606_0_stdby_o  (ad7606_0_stdby_o),
        .sdram_cko_clk     (sdram_clk),
        .sdram_addr        (sdram_addr),
        .sdram_ba          (sdram_ba),
        .sdram_cas_n       (sdram_cas_n),
        .sdram_cke         (sdram_cke),
        .sdram_cs_n        (sdram_cs_n),
        .sdram_dq          (sdram_dq),
        .sdram_dqm         (sdram_dqm),
        .sdram_ras_n       (sdram_ras_n),
        .sdram_we_n        (sdram_we_n),
        .uart_0_rxd        (uart_0_rxd),
        .uart_0_txd        (uart_0_txd)
    );

endmodule
```

注意，需要添加mysystem.qsys文件到工程中，否则编译会报错的。



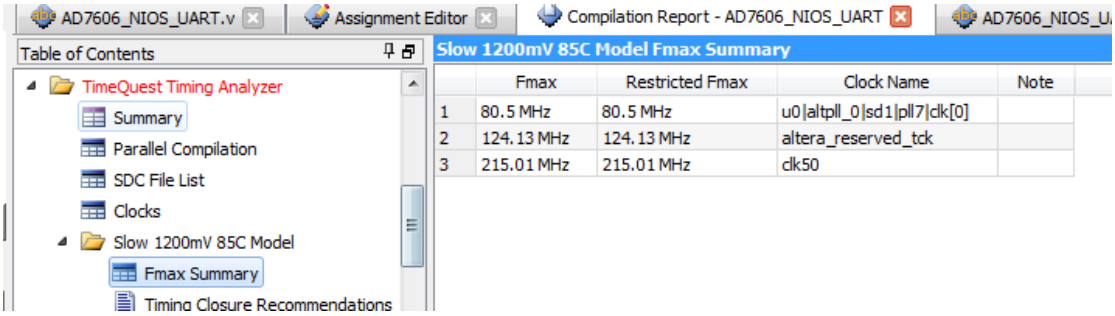
添加完成后，对工程执行一次分析和综合，以检查语法和逻辑错误。并综合出引脚信息。然后就可以对工程分配引脚了，分配引脚时，如果用户在创建工程时端口命令和我们的示例完全一样，就可以直接使用我们提供的tcl脚本文件快速分配引脚。例如针对AC620开发板，从ACM7606模块资料的引脚分配文件夹中复制ac620_ad7606.tcl文件到工程目录下，然后运行该脚本即可。



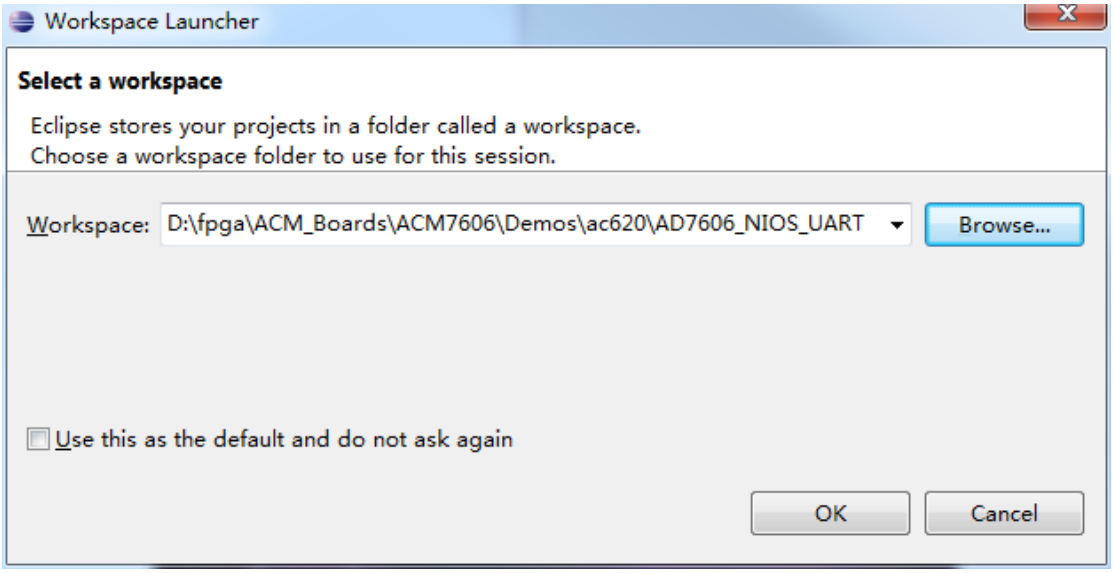
分配完引脚后，建议对工程添加时序约束文件，可以新建一个sdc格式的文件，在文件中写入下面两行命令，保存为AD7606_NIOS_UART.sdc文件，然后对工程执行全编译。

```
create_clock -name {clk50} -period 20.000 -waveform { 0.000 10.000 } [get_ports {clk}]
derive_pll_clocks
```

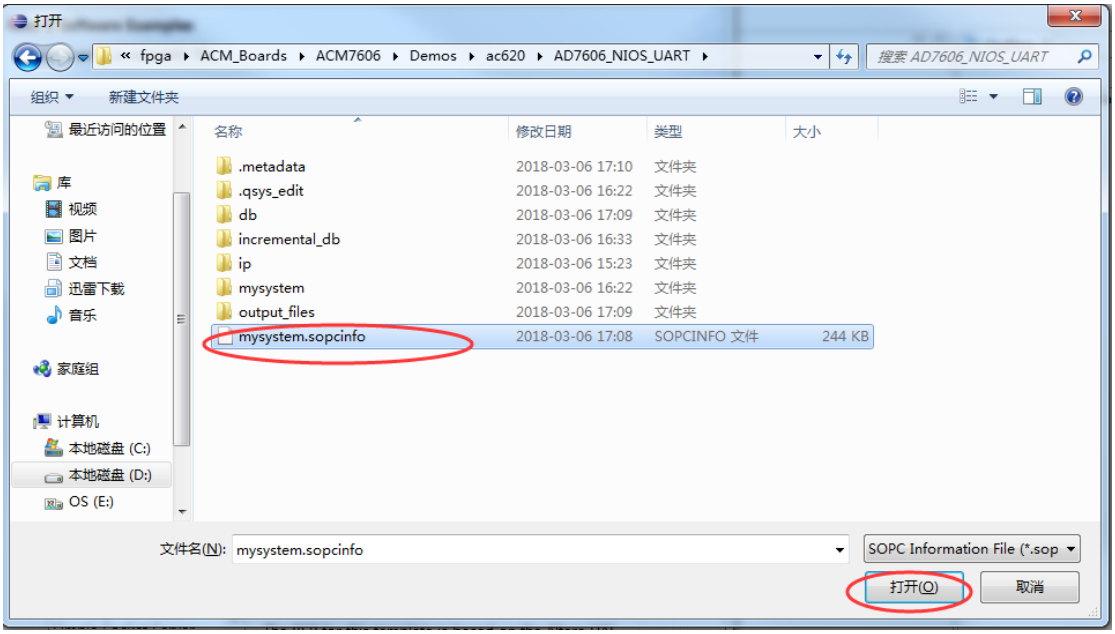
编译结束后，可以在编译报告中简单查看时序分析结果，如下图可以看到，系统最高运行频率为80MHz，大于我们设置的PLL输出时钟65M，因此时序收敛。

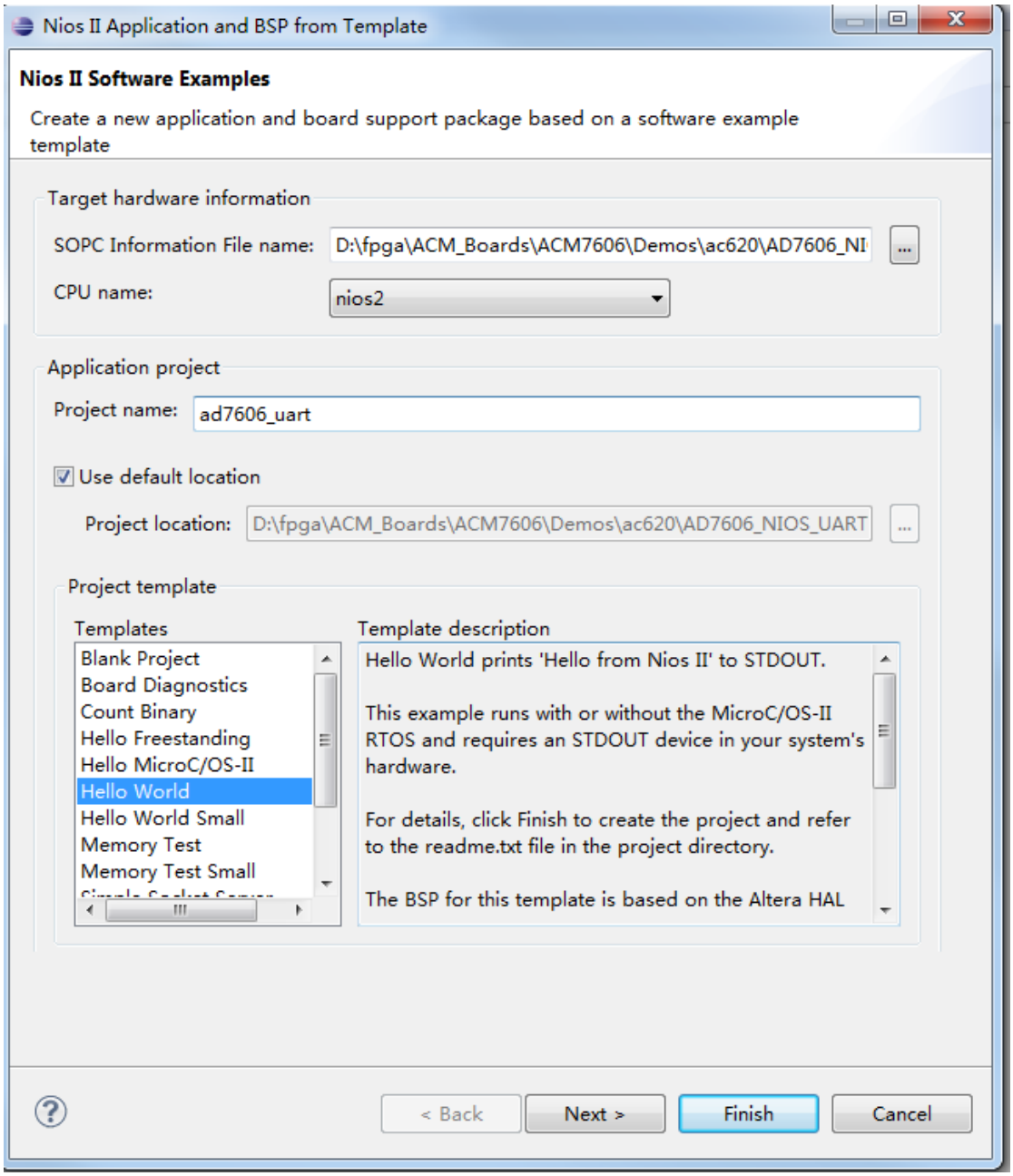


全编译结束后，打开NIOS II EDS软件，切换工作路径到当前工程下，如本示例工程在作者电脑上的路径为D:\fpga\ACM_Boards\ACM7606\Demos\ac620\AD7606_NIOS_UART。

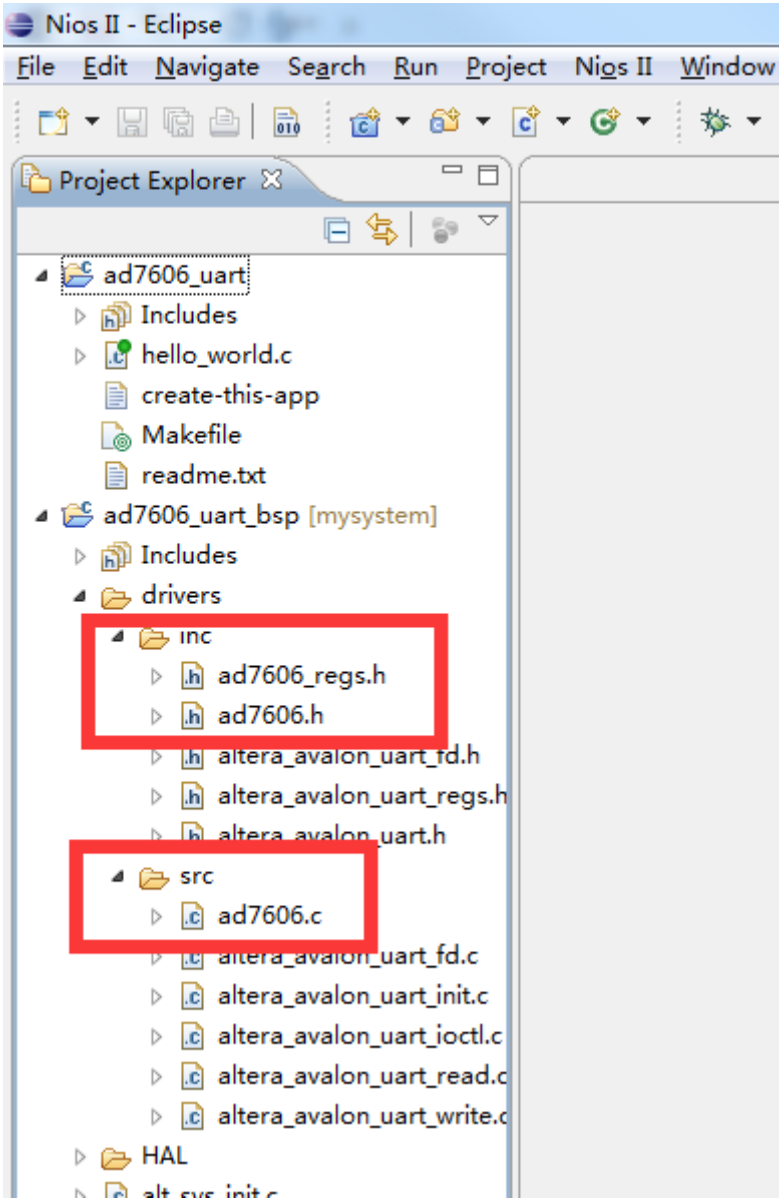


在NIOS II EDS软件中使用本例中Qsys中创建的mysystem.sopcinfo文件创建新的NIOS II软件工程和BSP工程，创建工程时使用Hello World模版以自动完成一些环境变量的设置。

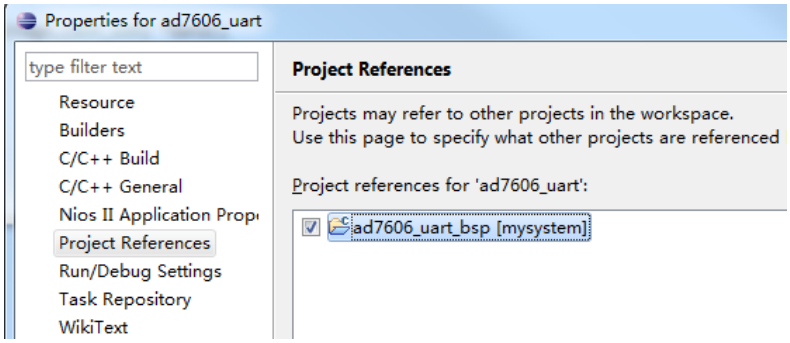




创建完成以后，在bsp工程下可以看到与ad7670相关的3个程序源文件已经自动添加到了工程中。



注意，手动设置用户工程和bsp工程的关联性，以避免软件在编译时报找不到相关文件的错误。



然后将下述代码复制粘贴到hello_world.c文件中

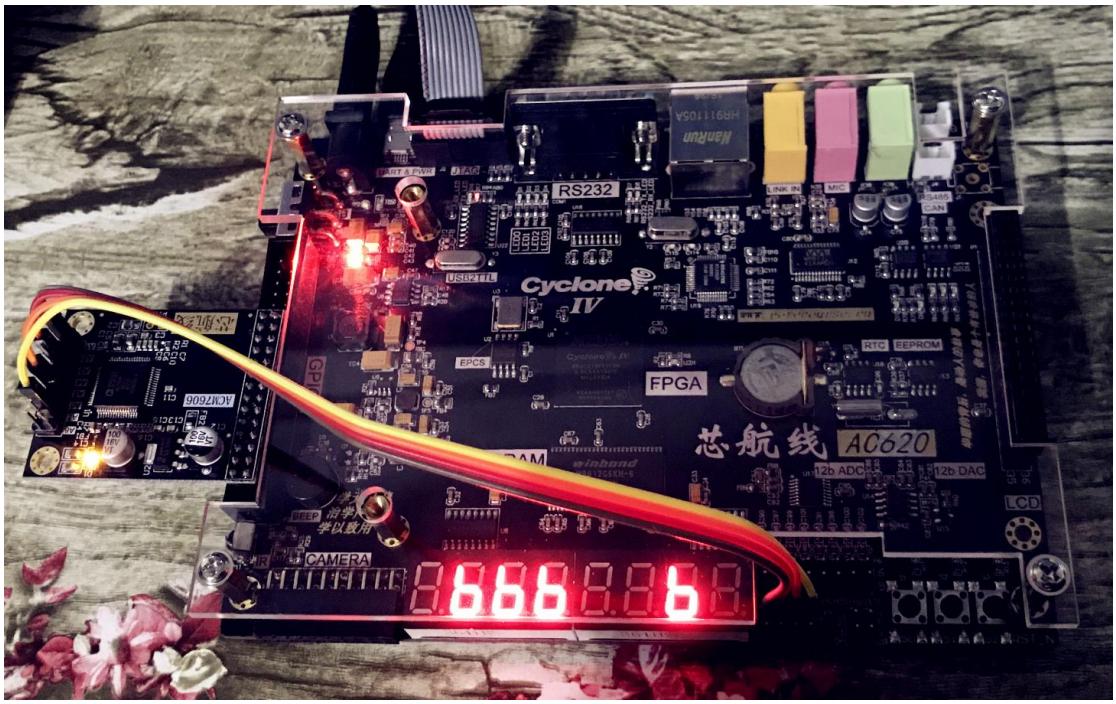
```
#include <stdio.h>
#include <stdlib.h>
#include "unistd.h"
```

```c
#include "system.h"
#include "AD7606.h"
#include "sys/alt_stdio.h"
#include "altera_avalon_uart_regs.h"
#include "alt_types.h"

void uart_0_init() {
    IOWR_ALTERA_AVALON_UART_STATUS(UART_0_BASE, 0x0);
    //清零状态寄存器
    IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);
    //读空接收寄存器中的无用值
    IOWR_ALTERA_AVALON_UART_DIVISOR(UART_0_BASE, UART_0_FREQ/115200 - 1);
    //设置波特率为115200
    IOWR_ALTERA_AVALON_UART_CONTROL(UART_0_BASE, 0x00);
    //关闭所有中断
}

//通过串口发送电压信息
void send_voltage(alt_16 *data, alt_u8 size, alt_u8 channel){
    int i = 0,j=0;
    volatile float Voltage;

    printf("\n\n");
    for(j=0;j<size/channel;j++)
    {
        printf("voltage :");
        for(i=0;i<channel;i++)
        {
            Voltage = *data * (10.0/65536);

            if(Voltage < 0)
                printf("%2.4f,%4x;",Voltage,(alt_u16) *data);
            else
                printf("+%2.4f,%4x;",Voltage,(alt_u16) *data);
            data ++;
        }
        printf("\n");

    }
}

int main() {
    alt_u32 data_base = SDRAM_BASE | 0x80000000 + 65536 * 8;
```

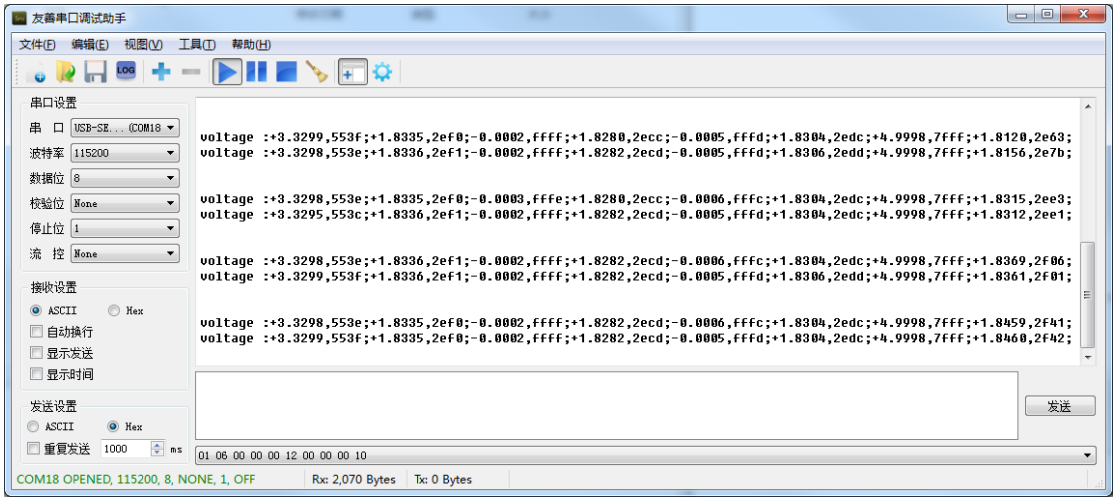
```
    alt_u32 size,size_t;

    alt_16 *data;

    uart_0_init(); //初始化串口
    AD7606_Reset();//复位AD7606控制器
    usleep(13000);
    while (1) {
        //采样8个通道
        AD7606_WriteRegister(CHAN8 | OS16 | NO_STAND_BY);

        //采样size个数据，存储到SDRAM中
        size = 16;
        AD7606_StartAcquisition(size, data_base);
        usleep(1000000);
        //等待采样完成
        while (!(AD7606_ReadStatus() & 0x00000001))
            ;

        data = data_base;

        send_voltage(data,size,8);
    }
    return 0;
}
```

对工程进行编译，先烧写sof文件到FPGA中，然后在NIOS II EDS软件中运行（Run）该软件工程，当程序运行起来后，在串口调试软件上选择波特率115200，打开相应端口，就可以看到系统每秒发送16个采样数据。

测试时将ACM7606模块的A1、A3、A5、A7通道分别连接到3.3V、GND、GND、5V，A2、A4、A6、A8通道悬空，如下图所示。



测量果如下所示：



实际用户在使用时，修改程序中变量size的值，就可以修改一次连续采样的数据量。

需要注意的是，定义数据存储的起始地址时，我们使用了一个变量直接存储地址的值

```
alt_u32 data_base = SDRAM_BASE | 0x80000000 + 65536 * 8;
```

在这里，我们对地址的值最高位置1（| 0x80000000）以屏蔽NIOS II的数据cache，用户自己在编写NIOS II的程序时候也需要注意，如果直接以绝对地址指针的形式进行读写操作时，为了防止data cache的影响，需要将绝对地址的最高位置1。在本实验中，如果不将地址最高位置1，那么在串口发送时，发送的依旧是cache中的旧数据值，即使外部电压值改变，串口发送的数据也不会变化。

另外在实际应用时，我们可能需要仅发送采样结果的原值编码值到PC上，由PC进一步处理，因此可以使用串口直接发送采样结果的16进制数据到PC端，下面以一个简单的例子讲解如何使用串口直接发送16进制原始采样值到PC。

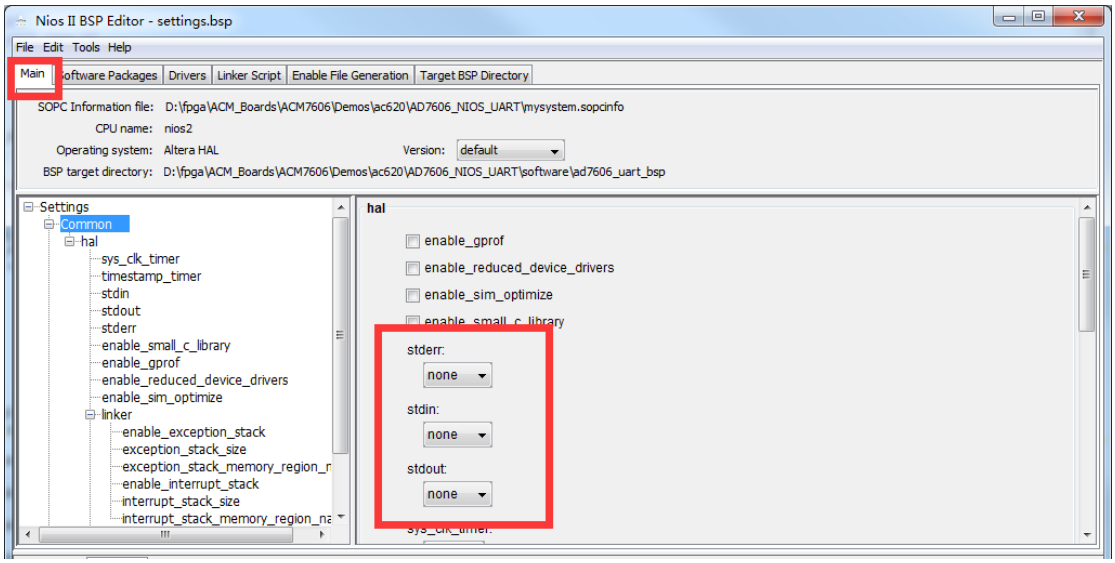
将hello_world.c文件复制并重命名为main.c，删除文件中所有代码，然后将下面的代码复

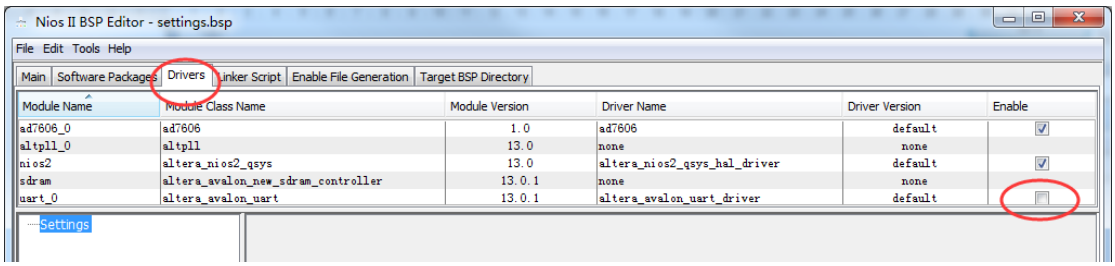
制粘贴到main.c文件中，

```
#include <stdio.h>
#include <stdlib.h>
#include "unistd.h"
#include "system.h"
#include "AD7606.h"
#include "sys/alt_stdio.h"
#include "altera_avalon_uart_regs.h"
#include "alt_types.h"

void uart_0_init() {
    IOWR_ALTERA_AVALON_UART_STATUS(UART_0_BASE, 0x0);
    //清零状态寄存器
    IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);
    //读空接收寄存器中的无用值
    IOWR_ALTERA_AVALON_UART_DIVISOR(UART_0_BASE, UART_0_FREQ/115200 - 1);
    //设置波特率为115200
    IOWR_ALTERA_AVALON_UART_CONTROL(UART_0_BASE, 0x00);
    //关闭所有中断
}

int main() {
    alt_u32 data_base = SDRAM_BASE | 0x80000000 + 65536 * 8;

    alt_u32 size,size_t;

    alt_u8 *data;

    uart_0_init(); //初始化串口
    AD7606_Reset();//复位AD7606控制器
    usleep(13000);
    while (1) {
        //采样8个通道
        AD7606_WriteRegister(CHAN8 | OS16 | NO_STAND_BY);

        //采样size个数据，存储到SDRAM中
        size = 32;
        AD7606_StartAcquisition(size, data_base);
        usleep(1000000);
        //等待采样完成
        while (!(AD7606_ReadStatus() & 0x00000001))
            ;

        //计算总共的字节数
```

```
        size_t = size*2;    //一个数据占2个字节，需要用串口分2次发送
        data = (alt_u8 *) data_base;

        //使用串口将所有数据传输
        while (size_t) {
            while (!(IORD_ALTERA_AVALON_UART_STATUS(UART_0_BASE)
                    & ALTERA_AVALON_UART_STATUS_TRDY_MSK))
                ;
            IOWR_ALTERA_AVALON_UART_TXDATA(UART_0_BASE, *data);
            data++;
            size_t--;
        }
    }
    return 0;
}
```

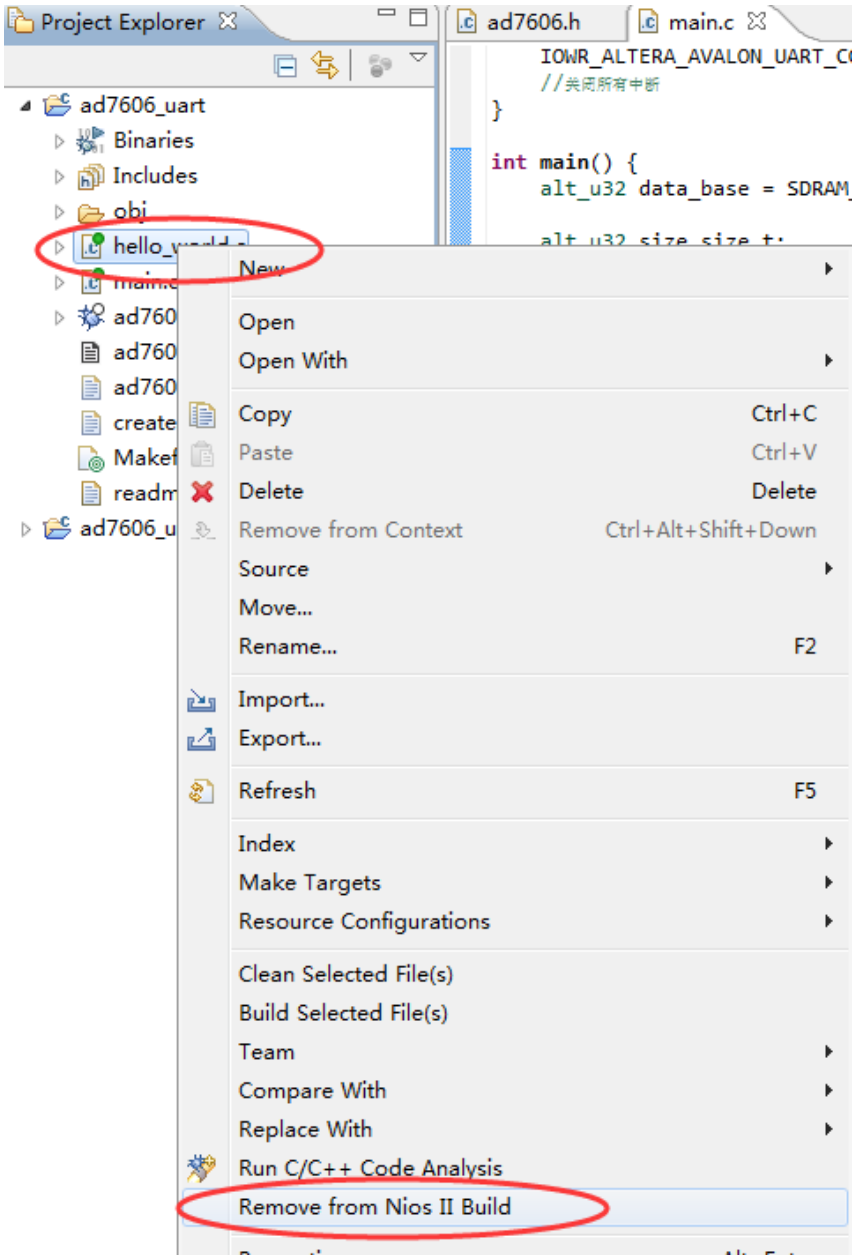
另外需要特别注意的是，使用此种编程方式操作串口时，由于BSP工程中还提供有HAL驱动会在程序运行时自动初始化UART串口，该驱动程序会和我们自己编写的直接操作串口寄存器的程序相冲突，因此我们需要关闭HAL驱动，关闭方式很简单，打开BSP Editor，首先设置stderr、stdin、stdout为none，然后切换到Drivers选项卡中，将UART_0后面的Enable勾选项取消。然后点击Generate，Exit，保存并退出即可。然后重新编译工程，运行就可以正常工作了。

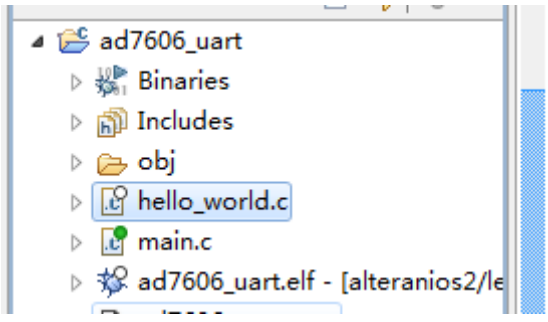




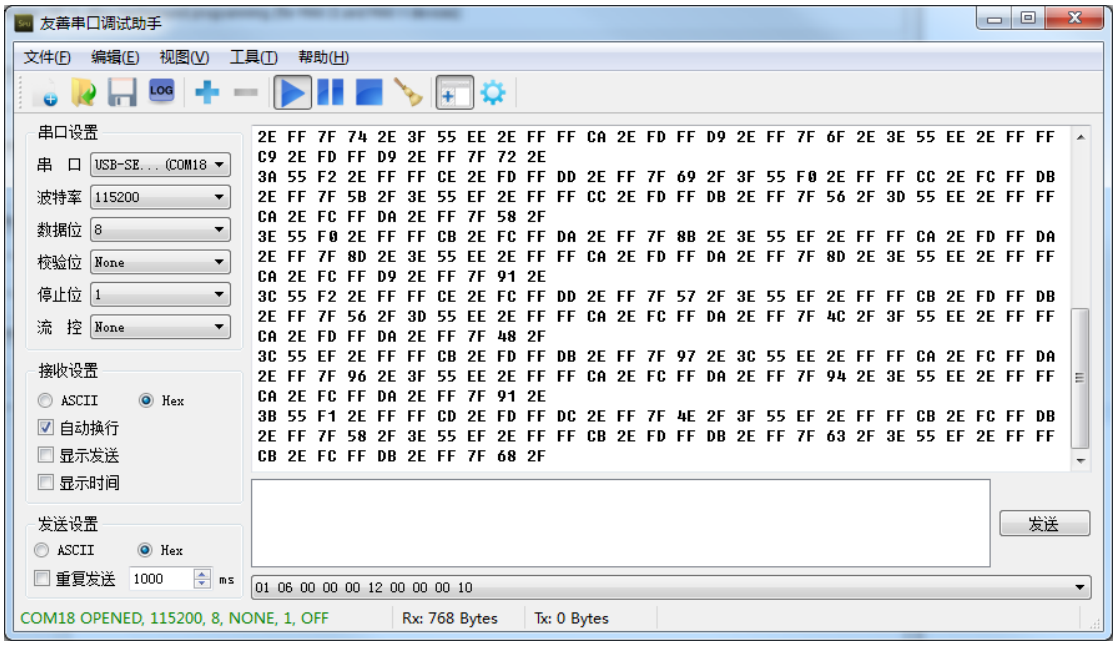
在NIOS II EDS软件的工程浏览器中，选中hello_world.c文件，右键选择Remove from Nios II Build，然后等待软件刷新，刷新完成后可以看到hello_world.c文件前面的绿色小点已经变

为了空心，表示该文件将不参与工程的编译了。如果不进行此设置，hello_world.c和mian.c中都有main函数，编译就会冲突出错。如果后面想切换回该文件作为程序的main文件，只需要重新选中该文件右击选择Add to Nios II Build即可。





编译工程并运行，在串口调试软件上选择16进制显示，可以看到，串口也正常发送了采样结果的16进制数据到电脑上，方便电脑进行进一步分析处理。


友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
串口设置
串 口 USB-SE...(COM18
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
2E FF 7F 74 2E 3F 55 EE 2E FF FF CA 2E FD FF D9 2E FF 7F 6F 2E 3E 55 EE 2E FF FF
C9 2E FD FF D9 2E FF 7F 72 2E
3A 55 F2 2E FF FF CE 2E FD FF DD 2E FF 7F 69 2F 3F 55 F0 2E FF FF CC 2E FC FF DB
2E FF 7F 5B 2F 3E 55 EF 2E FF FF CC 2E FD FF DB 2E FF 7F 56 2F 3D 55 EE 2E FF FF
CA 2E FC FF DA 2E FF 7F 58 2F
3E 55 F0 2E FF FF CB 2E FC FF DA 2E FF 7F 8B 2E 3E 55 EF 2E FF FF CA 2E FD FF DA
2E FF 7F 8D 2E 3E 55 EE 2E FF FF CA 2E FD FF DA 2E FF 7F 8D 2E 3E 55 EE 2E FF FF
CA 2E FC FF D9 2E FF 7F 91 2E
3C 55 F2 2E FF FF CE 2E FC FF DD 2E FF 7F 57 2F 3E 55 EF 2E FF FF CB 2E FD FF DB
2E FF 7F 56 2F 3D 55 EE 2E FF FF CA 2E FC FF DA 2E FF 7F 4C 2F 3F 55 EE 2E FF FF
CA 2E FD FF DA 2E FF 7F 48 2F
3C 55 EF 2E FF FF CB 2E FD FF DB 2E FF 7F 97 2E 3C 55 EE 2E FF FF CA 2E FC FF DA
2E FF 7F 96 2E 3F 55 EE 2E FF FF CA 2E FC FF DA 2E FF 7F 94 2E 3E 55 EE 2E FF FF
CA 2E FC FF DA 2E FF 7F 91 2E
3B 55 F1 2E FF FF CD 2E FD FF DC 2E FF 7F 4E 2F 3F 55 EF 2E FF FF CB 2E FC FF DB
2E FF 7F 58 2F 3E 55 EF 2E FF FF CB 2E FD FF DB 2E FF 7F 63 2F 3E 55 EF 2E FF FF
CB 2E FC FF DB 2E FF 7F 68 2F
发送
01 06 00 00 00 12 00 00 00 10
COM18 OPENED, 115200, 8, NONE, 1, OFF
Rx: 768 Bytes
Tx: 0 Bytes