

Altera FPGA 全速漂移 开发指南

基于最小 NIOS II 系统的 System ID 打印实例

欢迎加入 FPGA/CPLD 助学小组一同学习交流：

EDN:

http://group.ednchina.com/GROUP_GRO_14596_1375.HTM

ChinaAET: <http://group.chinaaet.com/273>

淘宝店链接: <http://myfpga.taobao.com/>

技术咨询: orand_support@sina.com

特权 HSC 最新资料例程下载地址:

<http://pan.baidu.com/s/1pLmZaFx>

版本信息		
时间	版本	状态
2016-07-09	V1.00	创建。



目录

Altera FPGA 全速漂移 开发指南	1
基于最小 NIOS II 系统的 System ID 打印实例	1
1 Qsys 系统概述	4
2 Qsys 工具基本使用	5
2.1 进入 Qsys 系统	5
2.2 Qsys 界面简介	6
2.3 新建 Qsys 系统	7
2.4 保存 Qsys 系统	8
2.5 加载 Qsys 系统	9
3 Qsys 组件添加与互联	11
3.1 时钟组件添加与设置	11
3.2 NIOS II 处理器添加与设置	14
3.3 RAM 组件添加与配置	16
3.4 NIOS II 处理器复位向量与异常向量地址设置	19
3.5 System ID 组件添加与配置	20
3.6 JTAG UART 组件添加与配置	22
3.7 Timer 组件添加与配置	25
4 Qsys 系统生成	28
4.1 中断连接	28
4.2 地址分配	30
4.3 系统生成	31
4.4 Qsys 系统例化模板	33
5 Quartus II 工程设计实现	34
5.1 Verilog 顶层文件设计	34
5.2 语法检查	36
5.3 引脚分配	36
5.4 系统编译	37
6 软件开发工具 EDS	37
6.1 EDS 软件开启	37
6.2 BSP 工程创建	39
6.3 开启 BSP Editor	42
6.4 BSP Editor 设置	43
6.5 BSP 工程编译	46
6.6 C 代码源文件创建	46
6.7 软件应用工程编译	48

6.8 移除当前工程.....	49
6.9 加载工程.....	49
6.10 移植工程.....	51
7 System ID 与 Timestamp 读取板级调试.....	53
7.1 软件功能概述.....	53
7.2 软件代码解析.....	54
7.3 板级调试.....	56

1 Qsys 系统概述

在我们这个 Qsys 嵌入式系统平台上，除了“万众瞩目”的 32 位处理器 NIOS II 外，还有一些最基本的常用标准外设（已经出现在 Qsys 的组件库中供直接加载使用），如 Clock 组件、片上 RAM、JTAG UART 外设、Timer 外设和 System ID 外设，该系统架构如图所示。

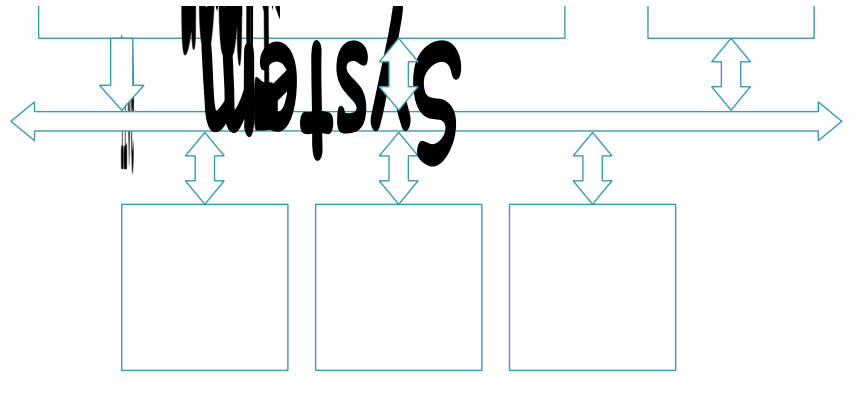


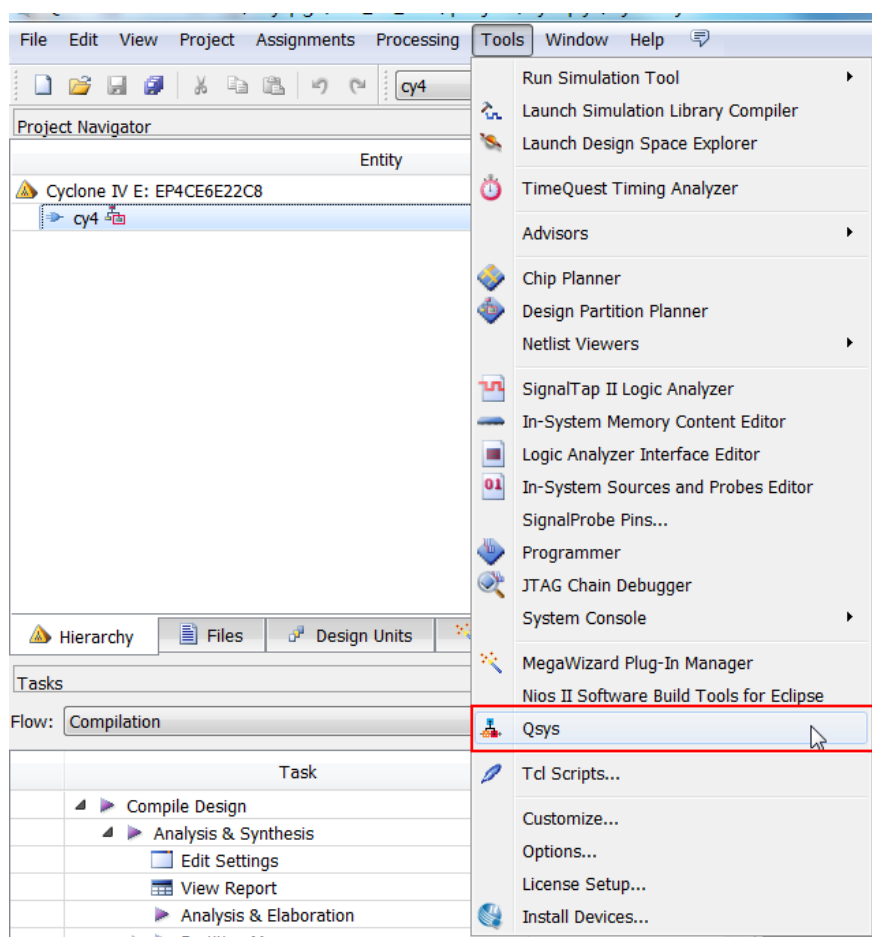
图 Qsys 系统框图

有了这个最小的 NIOS II 运行系统，就可以通过 C 语言代码，实现 NIOS II 处理器的编程控制。

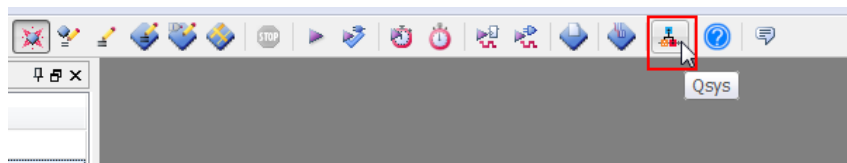
2 Qsys 工具基本使用

2.1 进入 Qsys 系统

如图所示，点击 Quartus II 工程的“Tools → Qsys”，可以打开进入 Qsys 编辑界面。

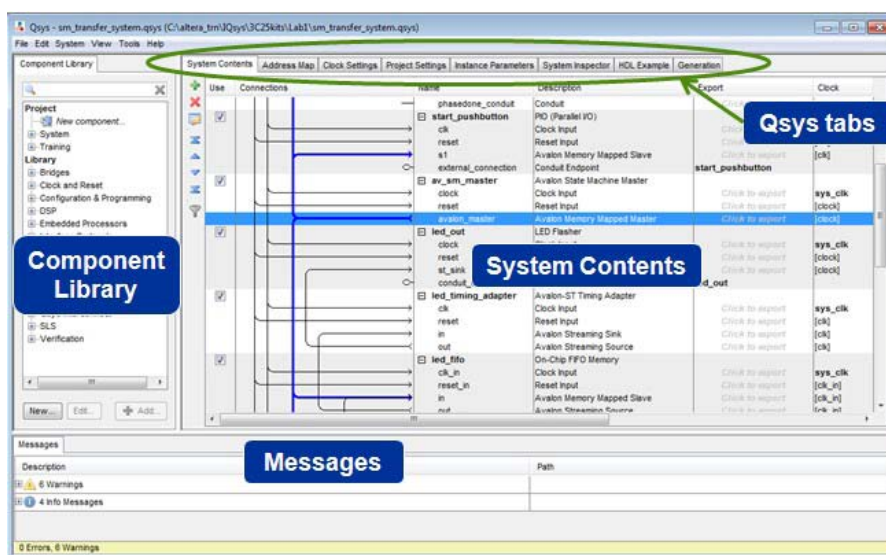


如图所示，我们也可以直接找到 Quartus II 的快捷按钮 Qsys，点击它进入 Qsys 编辑界面。

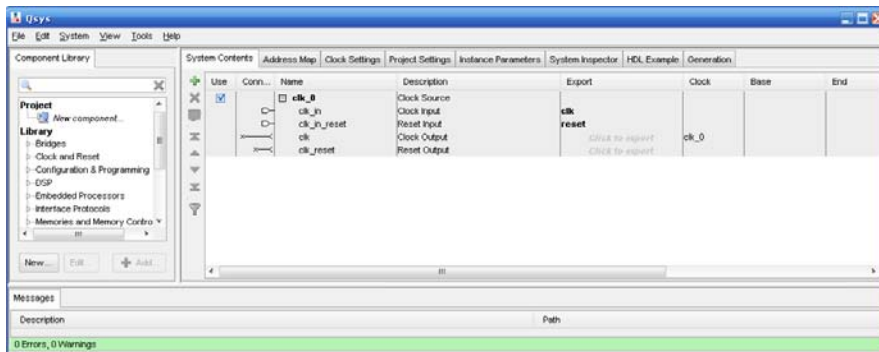


2.2 Qsys 界面简介

进入 Qsys 后，界面窗口的布局还是很清爽的，左侧的组件库（Component Library）中列出所有可供使用的 IP 核；右侧的工作栏（Qsys tabs）中有很多的选项，Qsys 系统的个性化编辑和设置都是在此进行。最下面的信息栏（Messages）实时显示当前系统编辑状态和一些错误或警告信息。

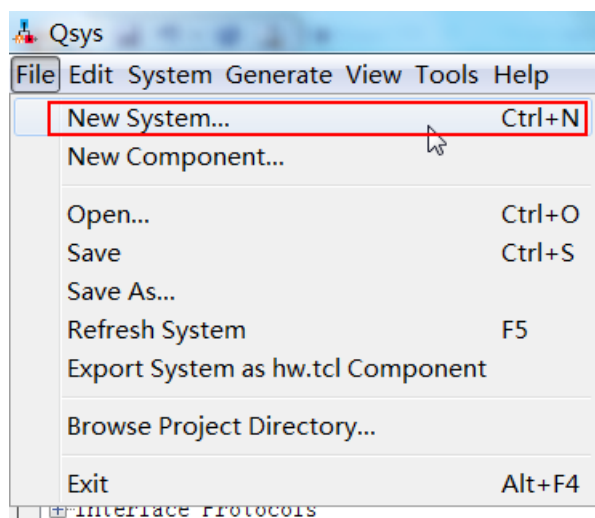


初次打开 Qsys，System Content 中默认已经添加了一个孤零零的 CLOCK 组件，其他啥也没有，光杆司令只是个摆设，啥活干不了。于是乎，我们需要在 Component Library 中各种查找，添加各种组件外设搭建出我们所期望的嵌入式系统。



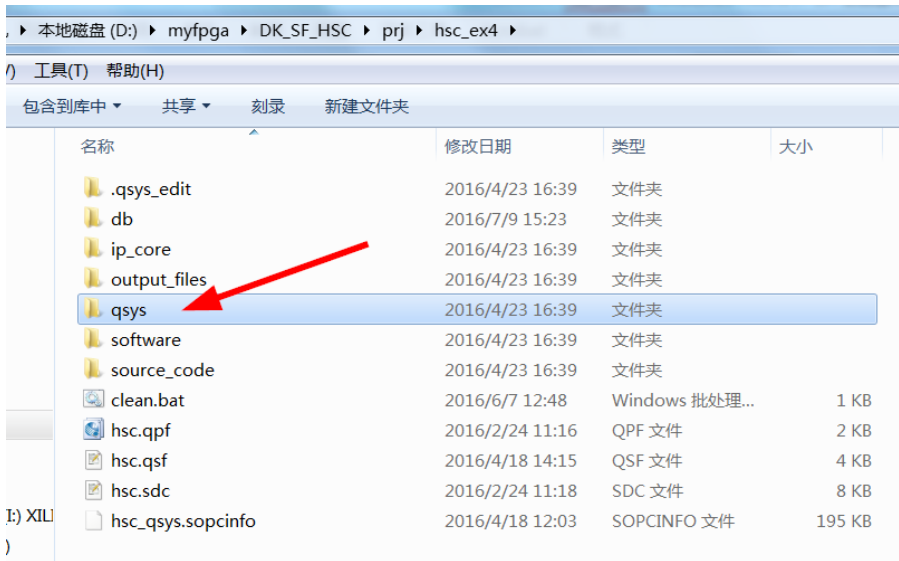
2.3 新建 Qsys 系统

第一次进入 Qsys 界面，其实我们所看到只有 CLOCK 单个组件的系统就是一个新建系统。当然了，我们也可以找到菜单栏点击“File → New System ...”来新建一个系统。

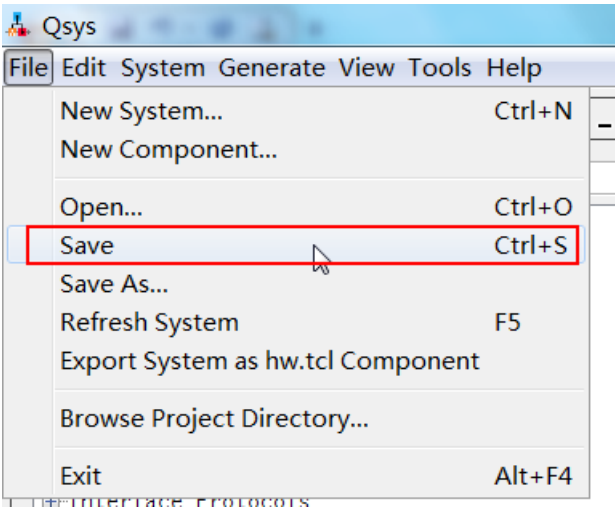


2.4 保存 Qsys 系统

首先，为了工程文件夹的管理更有层次感，便于将来查找和维护，我们极力建议大家在工程文件夹下创建一个名为 **qsys** 的文件夹，用于保存当前的 Qsys 系统生成的所有文件，如图所示。

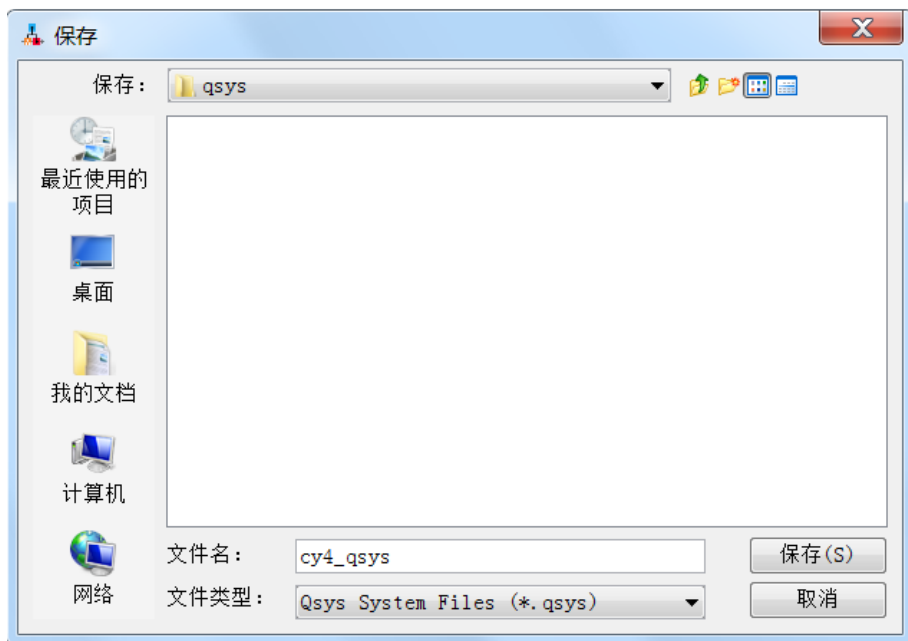


如图所示，在 Qsys 界面中点击菜单 “File → Save”。



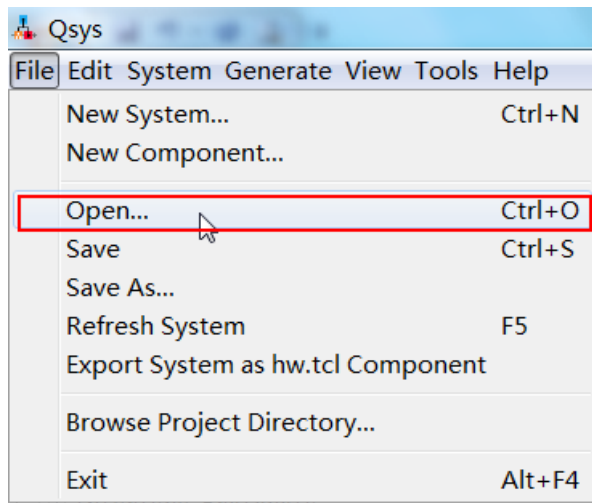
随后弹出了保存路径窗口，如图所示，我们找到刚刚创建的 **qsys** 文件

夹，将当前新建的 Qsys 系统命名为 hsc_qsys，然后保存。将来所有和这个新建 Qsys 系统有关的文件，都会生成到该文件夹下。

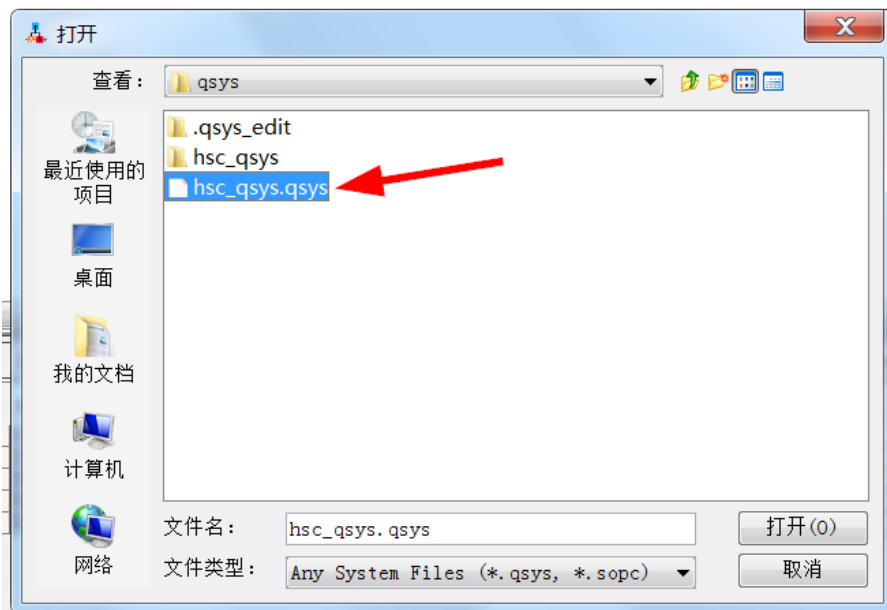


2.5 加载 Qsys 系统

完成当前系统的保存后，我们可以关闭当前的 Qsys 界面，回到 Quartus II 中重新进入 Qsys。每次重新进入 Qsys 后，一般不会自动加载最后一次编辑时的 Qsys 系统，而是和我们第一次进入 Qsys 界面时一样的一个新建系统状态。如图所示，我们必须点击菜单“Files → Open...”来加载当前可用的 Qsys 系统。



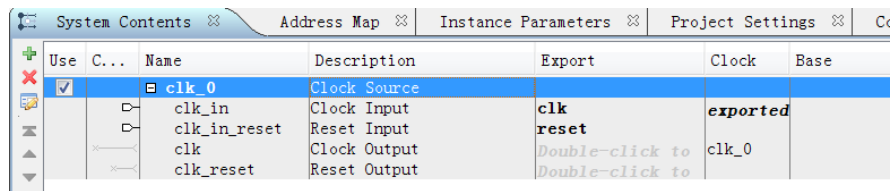
如图所示，弹出的“打开”窗口中，找到我们存放 cy4_qsys.qsys 的 qsys 文件夹，加载这个 Qsys 系统。



3 Qsys 组件添加与互联

3.1 时钟组件添加与设置

如图所示，当前系统中已经有了 CLOCK 组件（在 System Contents 选项卡中），我们双击它进入设置页面。



System Contents						
Address Map						
Instance Parameters						
Project Settings						
Clock						
Use	C...	Name	Description	Export	Clock	Base
<input checked="" type="checkbox"/>		clk_0	Clock Source			
	<input type="checkbox"/>	clk_in	Clock Input	clk	exported	
	<input type="checkbox"/>	clk_in_reset	Reset Input	reset		
	<input type="checkbox"/>	clk	Clock Output	Double-click to	clk_0	
	<input type="checkbox"/>	clk_reset	Reset Output	Double-click to		

图 CLOCK 组件

如图所示，设置时钟频率(Clock frequency)为 50000000Hz，即 50MHz，这个时钟频率设置不是随意的，我们这里设定好 50MHz，那么它在 Qsys 系统的顶层会引出一个 clock 输入信号，我们在 FPGA 中就必须输入一个 50MHz 的时钟信号连接到它。随后的“Clock frequency is known”勾选后，可以设置“Reset synchronous edges: ”，有“None”、“Both”和“Deassert”三个选项，这些设置主要是与复位和时钟的同步或异步控制机制有关。若没有特殊要求，可以不用设置。完成设置后，点击“Finish”退出。

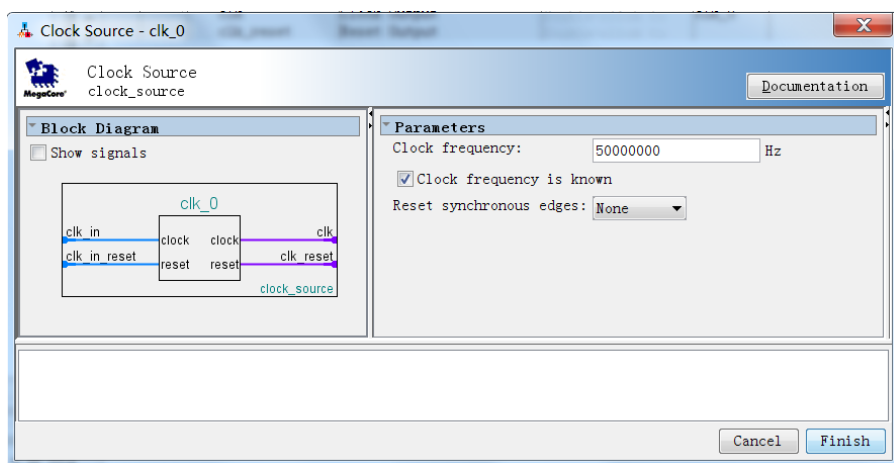


图 CLOCK 组件参数配置

如图所示，再 clk_0 的位置，单击右键，选择“Rename”，然后“Name”下面的“clk_0”的名称将处于可编辑状态。

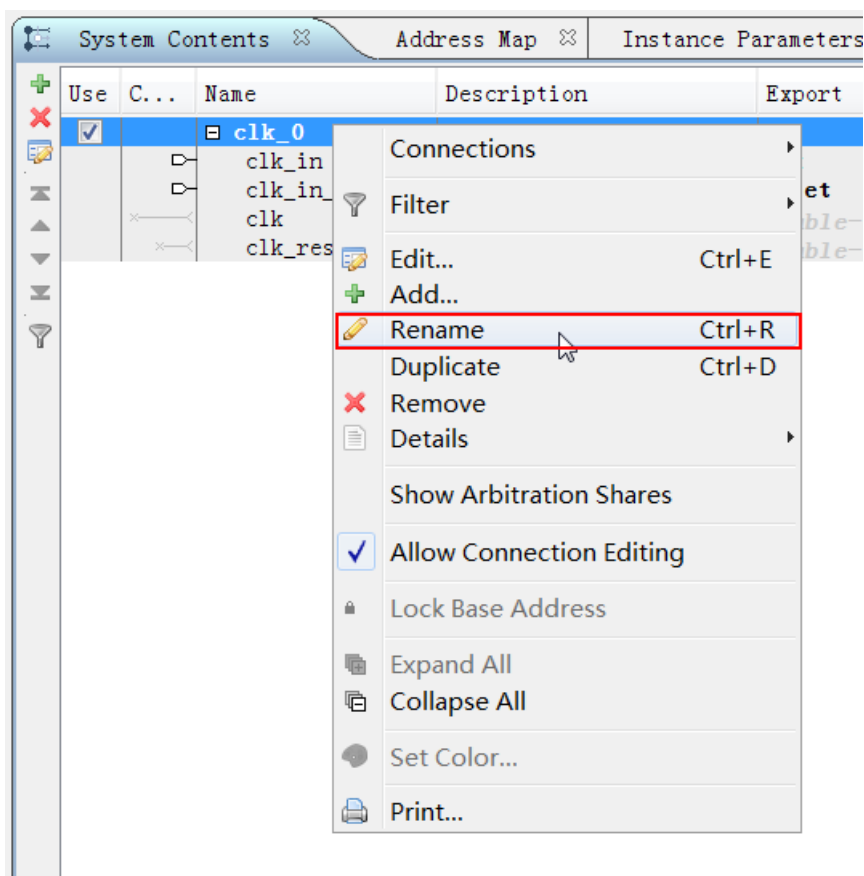


图 重命名 CLOCK 组件菜单

如图所示，我们将 CLOCK 组件重命名为“clk_50m”。

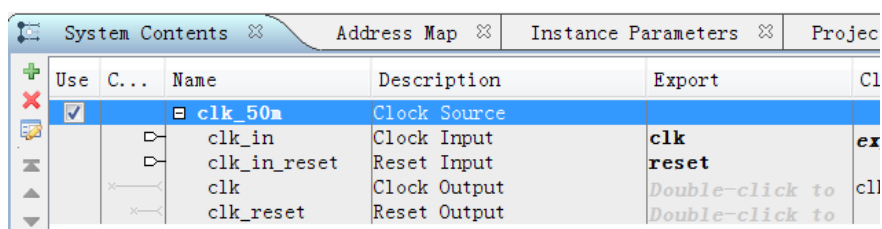


图 CLOCK 组件重命名

3.2 NIOS II 处理器添加与设置

如图所示，在 Library 面板中，选择“Library → Embedded Processors → Nios II Processor”，双击即可添加该组件。

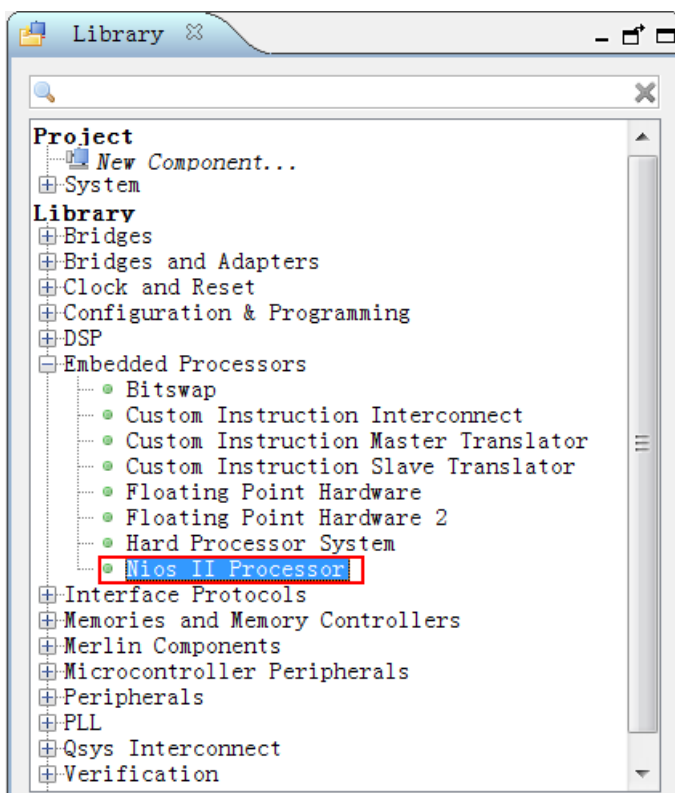


图 添加组件库中的 NIOS II 处理器

弹出 NIOS II 组件的设置页面，如图所示，这里有 3 个可选的 NIOS II 类型，即 Nios II/e（经济型，消耗资源最少，性能也最低）、Nios II/s（标准型，“性价比”最好）和 Nios II/f（快速型，性能最强，消耗资源也最多）。我们选择 Nios II/s。其它选项都使用默认状态，暂时不做任何设置，点击“Finish”完成配置。

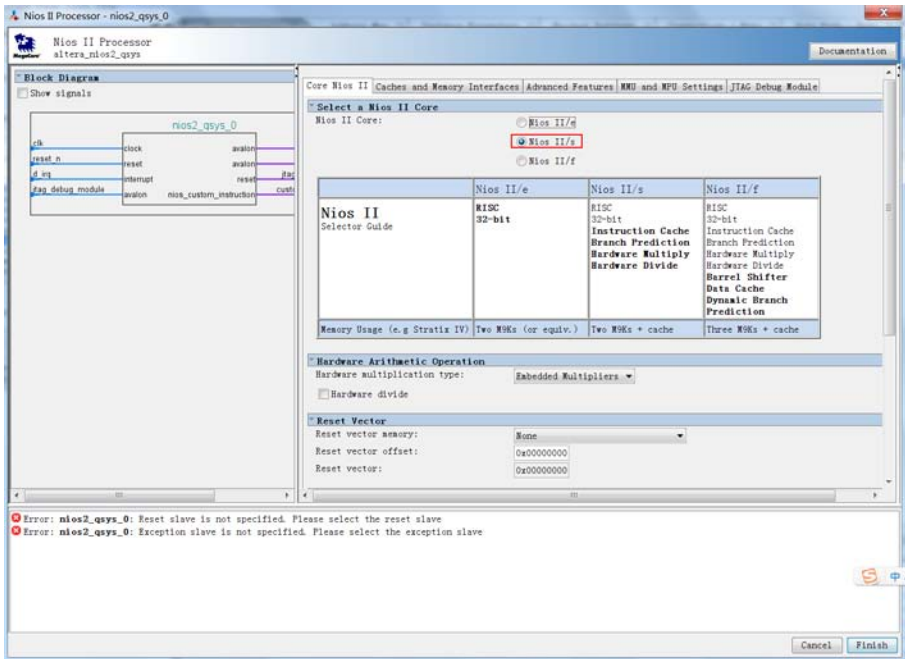


图 NIOS II 处理器参数配置

如图所示，修改刚刚添加的 NIOS II 组件的名称为“nios2”。

Use	Connections	Name	Description
<input checked="" type="checkbox"/>		clk_50m	Clock Source
<input checked="" type="checkbox"/>		nios2	Nios II Processor
		clk	Clock Input
		reset_n	Reset Input
		data_master	Avalon Memory Mapped ...
		instruction_master	Avalon Memory Mapped ...
		jtag_debug_module_reset	Reset Output
		jtag_debug_module	Avalon Memory Mapped ...
		custom_instruction_master	Custom Instruction Ma...

图 NIOS II 处理器重命名

如图所示，在 Connections 一列中，需要将 nios2 组件的时钟、复位信号分别连接到 CLOCK 组件的相应信号上。这意味着 NIOS II 处理器工作的时钟和复位源来自 CLOCK 组件。后面我们会看到，整个 Qsys 系统的时钟和复位信号都会连接到 CLOCK 组件输出的 clk 和 clk_reset 信号上，CLOCK 组

件可谓 Qsys 系统的“大心脏”。

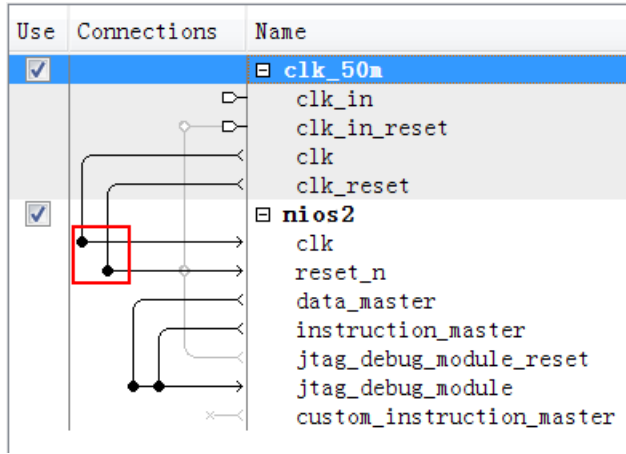


图 NIOS II 处理器与 CLOCK 组件互联

3.3 RAM 组件添加与配置

如图所示，在 Library 面板中，选择“Library → Memories and Memory Controllers → On-Chip → On-Chip Memory (RAM or ROM)”，双击即可添加该组件。

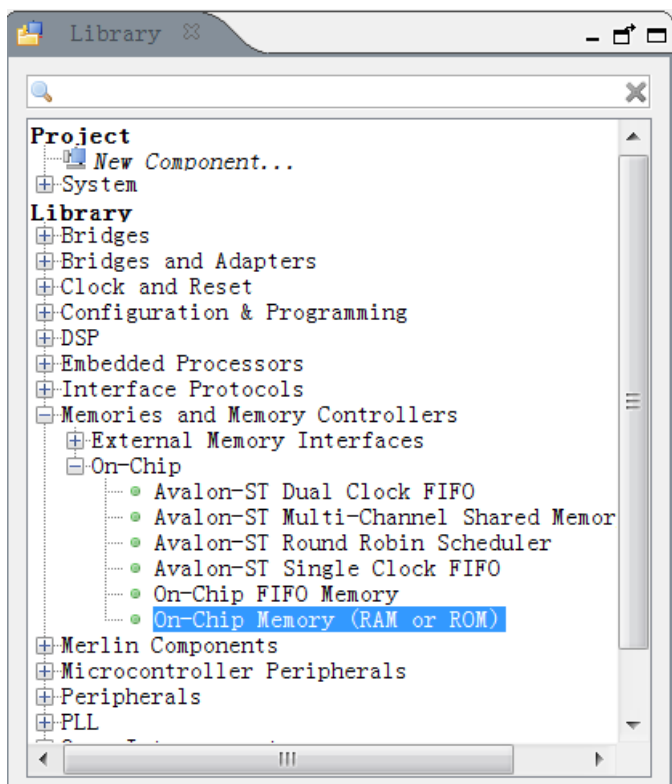


图 添加组件库中的片内 RAM 组件

弹出 On-Chip Memory 组件的设置页面，如图所示。选择存储器类型（Memory type）为“RAM (Writable)”，这个 RAM 要扮演 NIOS II 处理器程序存储器和数据存储器的重要角色。接着在 Size 中设定存储器数据位宽（Data width）为 32bit，存储量大小（Total memory size）为 22528 Bytes（即 22KBytes）。其它设置使用系统默认设置，点击 Finish 完成设置。

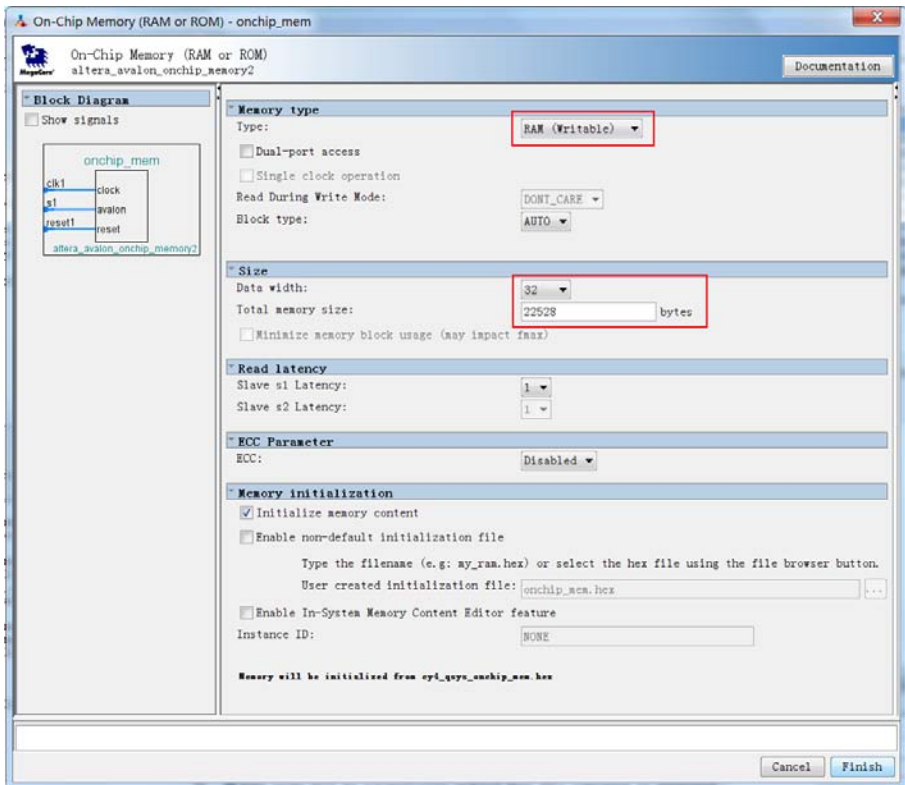


图 片内 RAM 组件参数配置

如图所示，修改刚刚添加的 On-Chip memory 组件的名称为“onchip_mem”。

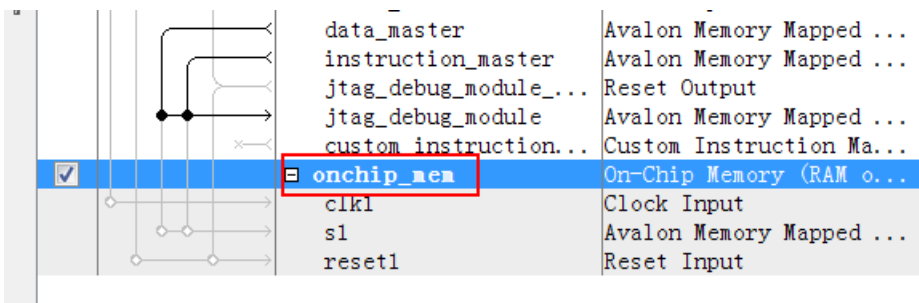


图 片内 RAM 组件重命名

如图所示，在 Connections 一列中，需要将 onchip_mem 组件的时钟、复位信号分别连接到 CLOCK 组件的相应信号上（图示红色方框内的左边两个实心点）。此外，因为我们要用这个 Onchip_mem 作为 NIOS II 处理器的

程序存储和程序运行的存储器，所以必须把 NIOS II 的指令总线（instruction_master）和数据总线（data_master）连接到 Onchip_mem（即 s1）上，即图示的红色方框内的右边两个实心点。

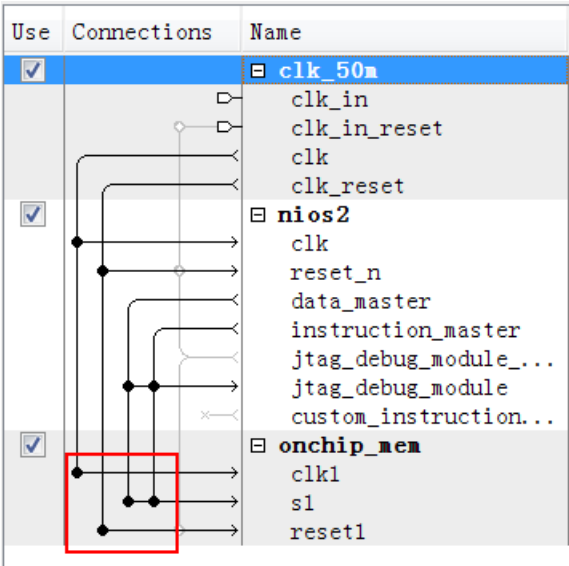


图 片内 RAM 组件与 CLOCK 组件、NIOS II 处理器互联

3.4 NIOS II 处理器复位向量与异常向量地址设置

接着我们还要双击打开 nios2 组件进行设置，如图所示，将其 Reset vector memory 和 Exception vector memory 均设为 onchip_mem.s1。

Reset Vector	
Reset vector memory:	onchip_mem.s1
Reset vector offset:	0x00000000
Reset vector:	0x00000000
Exception Vector	
Exception vector memory:	onchip_mem.s1
Exception vector offset:	0x00000020
Exception vector:	0x00000020

图 设置 NIOS II 处理器的复位向量和异常向量地址

如此设置后，当系统上电后，NIOS II 就从 Onchip_mem 存储器开始运行程序，并且数据读写也是在 Onchip_mem 存储器中。

3.5 System ID 组件添加与配置

如图所示，在 Library 面板中，选择“Library → Peripherals → Debug and Performance → System ID Peripheral”，双击即可添加该组件。

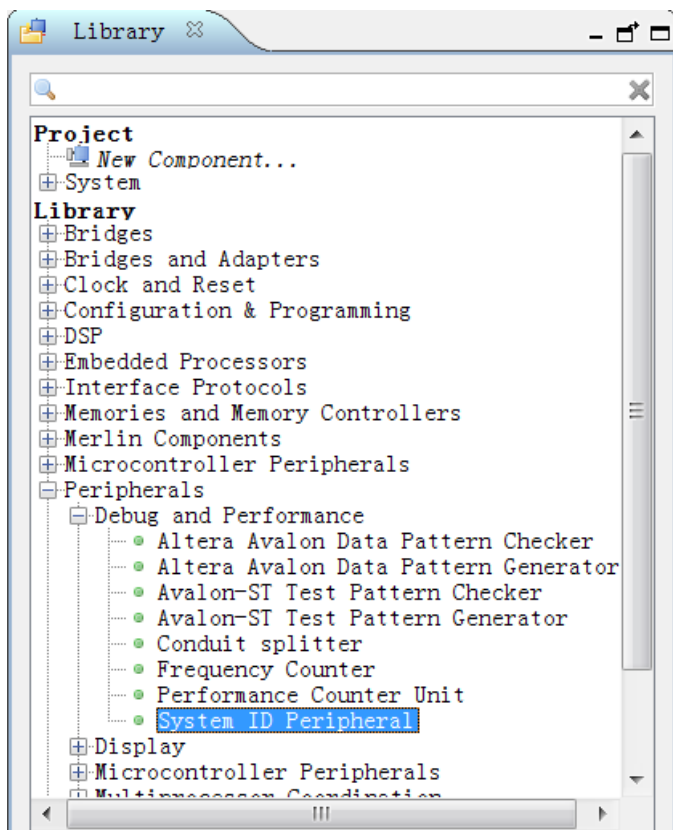


图 添加组件库中的 System ID 外设

System ID 是 NIOS II 处理器的唯一识别号，用于确认当前运行的程序和 FPGA 中内嵌的 NIOS II 处理器相匹配。System ID 的值为 32bit，由 Qsys 中

添加该组件时设置。

在弹出的 System ID 组件设置页面中，我们可以设置这个 32 位 ID 值为 0x11223344，如图所示。

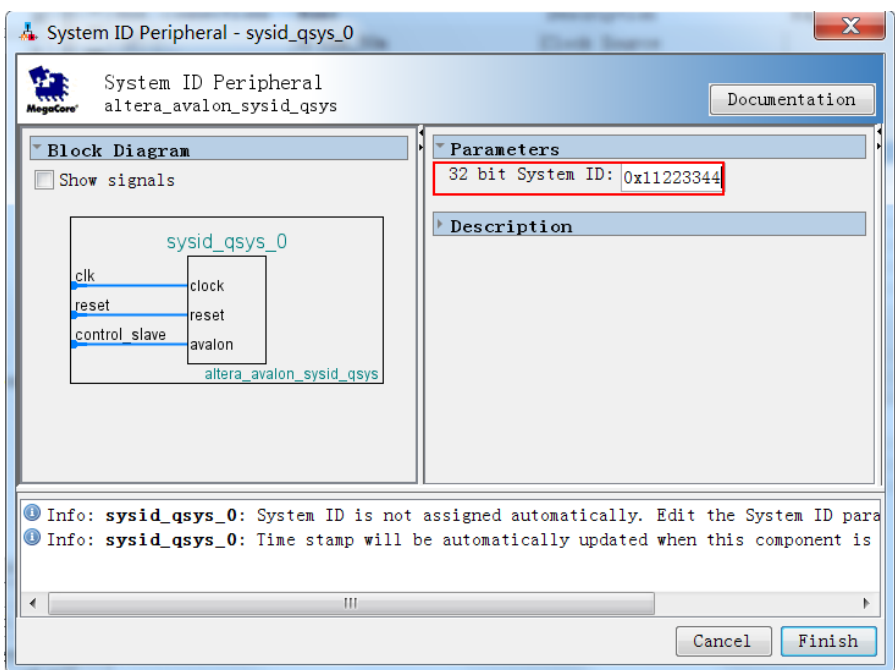


图 System ID 组件参数配置

如图所示，修改刚刚添加的 System ID 组件的名称为“sysid”。

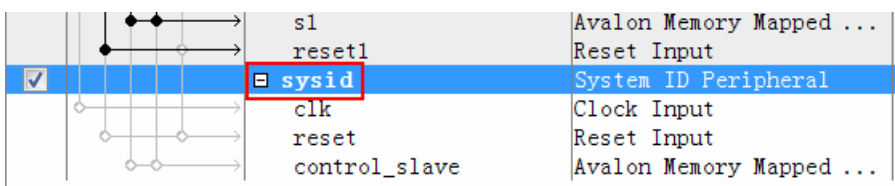


图 System ID 组件重命名

如图所示，在 Connections 一列中，需要将 System ID 组件的时钟、复位信号分别连接到 CLOCK 组件的相应信号上（图示红色方框内的左边两个实心点）。此外，因为 NIOS II 处理器需要能够访问到这个 System ID 组件，读取这个外设的 ID 数据，所以我们必须把 NIOS II 的数据总线(data_master) 连接到 System ID 组件（即 control_slave）上，即图示的红色方框内的最右

边一个实心点。

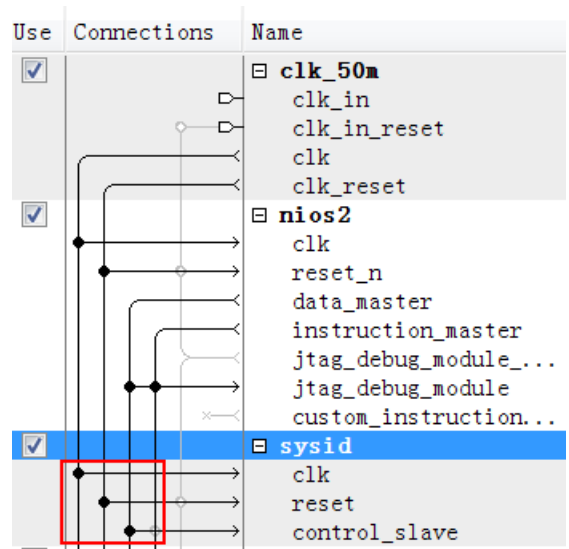
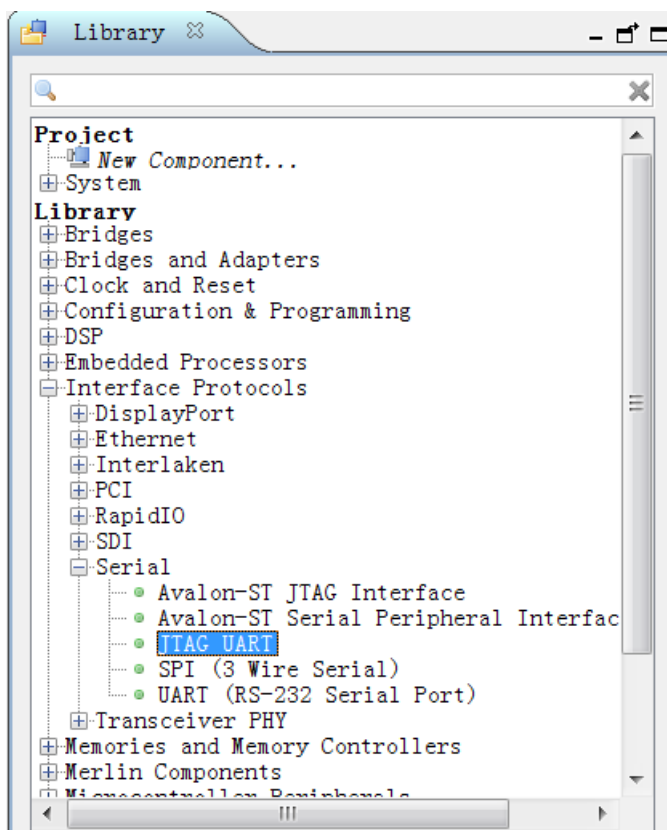


图 System ID 组件与 CLOCK 组件、NIO S II 处理器互联

3.6 JTAG UART 组件添加与配置

如图所示，在 Library 面板中，选择“Library → Interface Protocols → Serial → JTAG UART”，双击即可添加该组件。



图添加组件库中的 JTAG UART 组件

JTAG UART 使用 FPGA 既有的 JTAG 接口协议实现 PC 和 FPGA 内部的 NIOS II 处理器之间的串行字符串的传输，这如同很多嵌入式处理器调试中需要用到的 RS232 UART。

如图所示，在弹出的 JTAG UART 组件设置页面中，为了最小化这个外设需要用到的缓存 FIFO 对存储器的需求，我们设置 Write 和 Read FIFO 的存储量为 16Byte，并且勾选上 “Construct using registers instead of memory blocks”，表示不适用 FPGA 片内存储器，而是使用 FPGA 的逻辑来实现 FIFO。

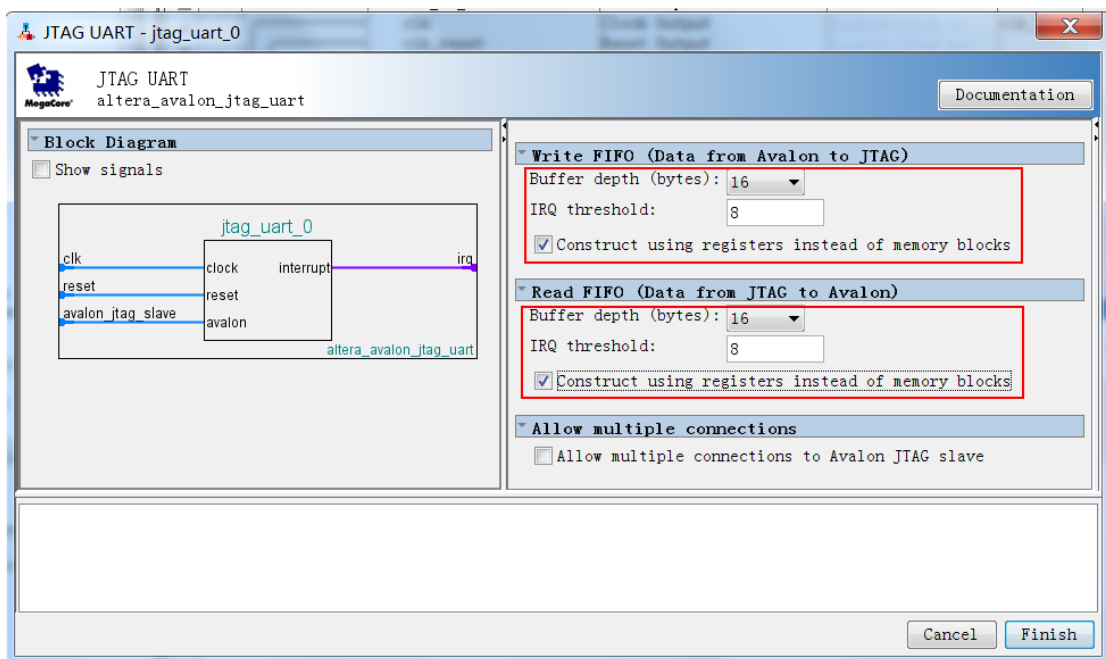


图 JTAG UART 组件参数配置

如图所示，修改刚刚添加的 JTAG UART 组件的名称为“jtag_uart”。

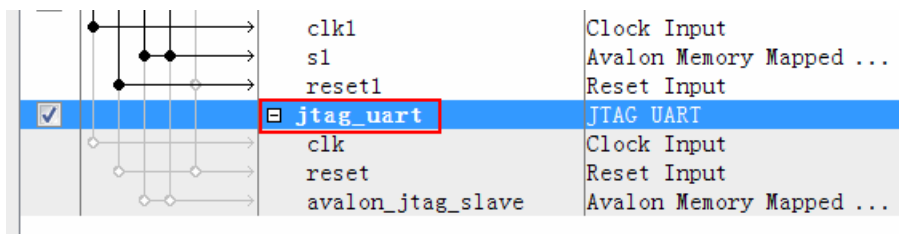


图 JTAG UART 组件重命名

如图所示，在 Connections 一列中，需要将 JTAG UART 组件的时钟、复位信号分别连接到 CLOCK 组件的相应信号上（图示红色方框内的左边两个实心点）。此外，因为 NIOS II 处理器需要能够访问到这个 JTAG UART 组件，实现 PC 和 NIOS II 处理器之间的数据传输，我们必须把 NIOS II 的数据总线（data_master）连接到 JTAG UART 组件（即 avalon_jtag_slave）上，即图示的红色方框内的最右边一个实心点。

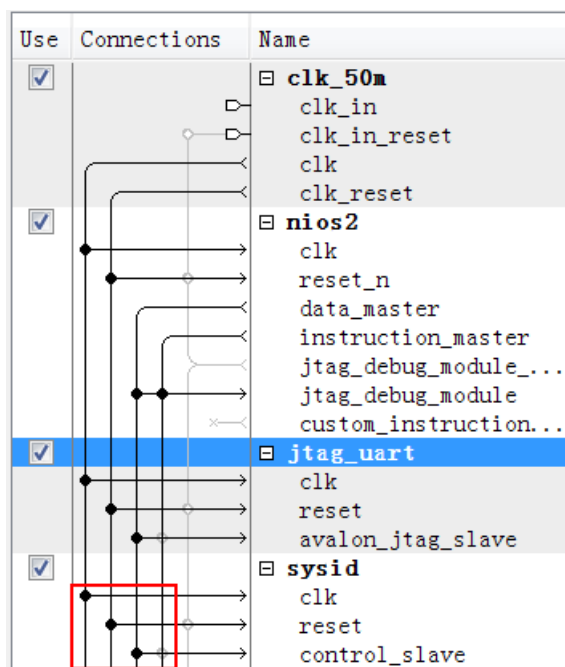
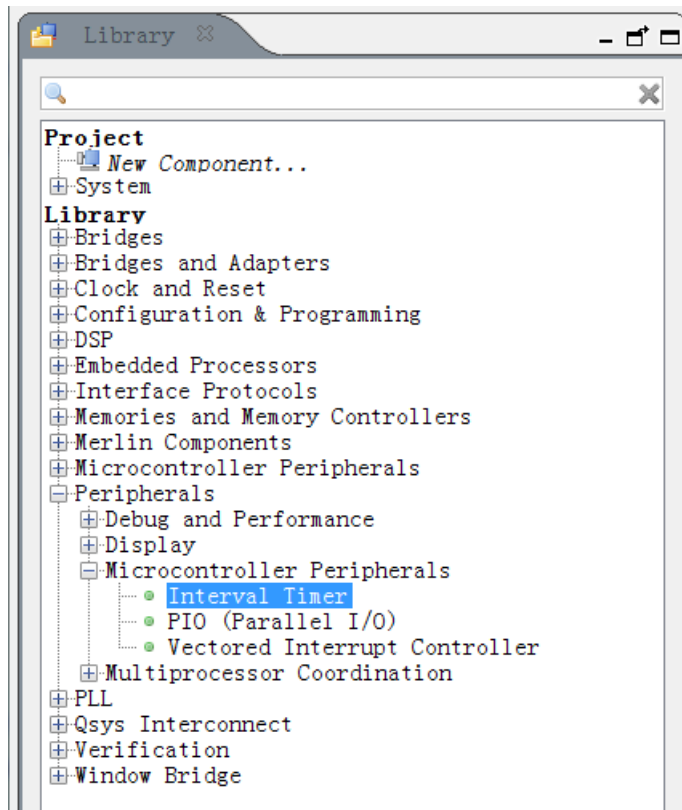


图 JTAG UART 组件与 CLOCK 组件、NIOS II 处理器互联

3.7 Timer 组件添加与配置

如图所示，在 Library 面板中，选择“Library → Peripherals → Microcontroller Peripherals → Interval Timer”，双击即可添加该组件。



图添加组件库中的 Timer 组件

如图所示，在弹出的 Timer 组件设置页面中，点击加载右侧 Presets 面板的“Full-featured”选项，接着设定定时器的默认时间为 1s（Period 值为 1，Units 为 s）。其他设置默认即可。

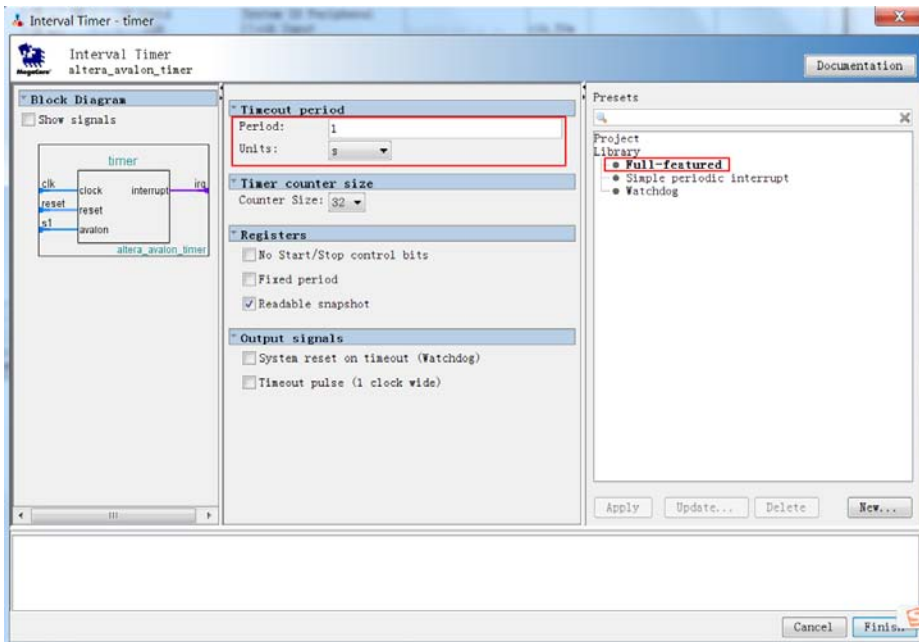


图 Timer 组件参数配置

如图所示，修改刚刚添加的 Timer 组件的名称为“timer”。

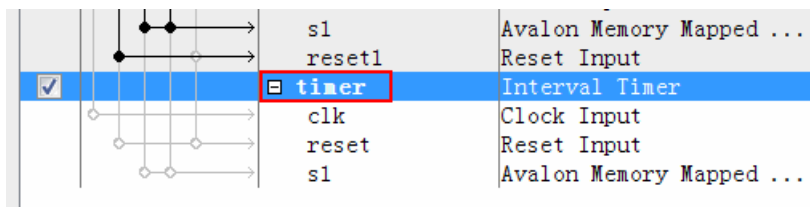


图 4.24 Timer 组件重命名

如图所示，在 Connections 一列中，需要将 Timer 组件的时钟、复位信号分别连接到 CLOCK 组件的相应信号上（图示红色方框内的左边两个实心点）。此外，因为 NIOS II 处理器需要能够访问到这个 Timer 组件，实现定时器中断功能，因此我们必须把 NIOS II 的数据总线（data_master）连接到 Timer 组件（即 s1）上，即图示的红色方框内的最右边一个实心点。

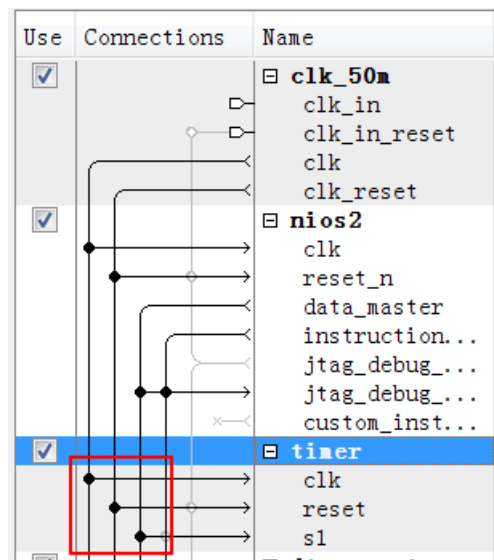


图 Timer 组件与 CLOCK 组件、NIOS II 处理器互联

4 Qsys 系统生成

4.1 中断连接

前面我们已经完成了整个 Qsys 系统所有组件的配置和添加。但是接下来我们还需要对一些组件的中断信号进行连接。如图所示，在 IRQ 一列中，与 NIOS II 处理器相连接的中断信号有 timer 组件和 jtag_uart 组件。我们注意这些 IRQ 信号和 NIOS II 处理器的连接点是一个空心圆，说明还未连接上。

Use	Conn...	Name	Description	Export	Clock	Base	End	IRQ	Tags
<input checked="" type="checkbox"/>		clk_50m	Clock Source		exported				
<input checked="" type="checkbox"/>		nios2	Nios II Processor						
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m				
<input checked="" type="checkbox"/>		reset.n	Reset Input	Double-click to	[clk]				
<input checked="" type="checkbox"/>		data_master	Avalon Memory Mapped ...	Double-click to	[clk]			IRQ 0	IRQ 31
<input checked="" type="checkbox"/>		instruction...	Avalon Memory Mapped ...	Double-click to	[clk]				
<input checked="" type="checkbox"/>		jtag_debug...	Reset Output	Double-click to	[clk]				
<input checked="" type="checkbox"/>		custom_inst...	Custom Instruction Ma...	Double-click to	[clk]	0x0001_0800	0x0001_0fff		
<input checked="" type="checkbox"/>		onchip_mem	On-Chip Memory (RAM o...						
<input checked="" type="checkbox"/>		clk1	Clock Input	Double-click to	clk_50m				
<input checked="" type="checkbox"/>		sl	Avalon Memory Mapped ...	Double-click to	[clk1]	0x0000_8000	0x0000_d7ff		
<input checked="" type="checkbox"/>		reset1	Reset Input	Double-click to	[clk1]				
<input checked="" type="checkbox"/>		sysid	System ID Peripheral						
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]				
<input checked="" type="checkbox"/>		control_slave	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1028	0x0001_102f		
<input checked="" type="checkbox"/>		jtag_uart	JTAG UART						
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]				
<input checked="" type="checkbox"/>		avalon_jtag...	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1020	0x0001_1027		
<input checked="" type="checkbox"/>		timer	Interval Timer						
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]				
<input checked="" type="checkbox"/>		sl	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1000	0x0001_101f		

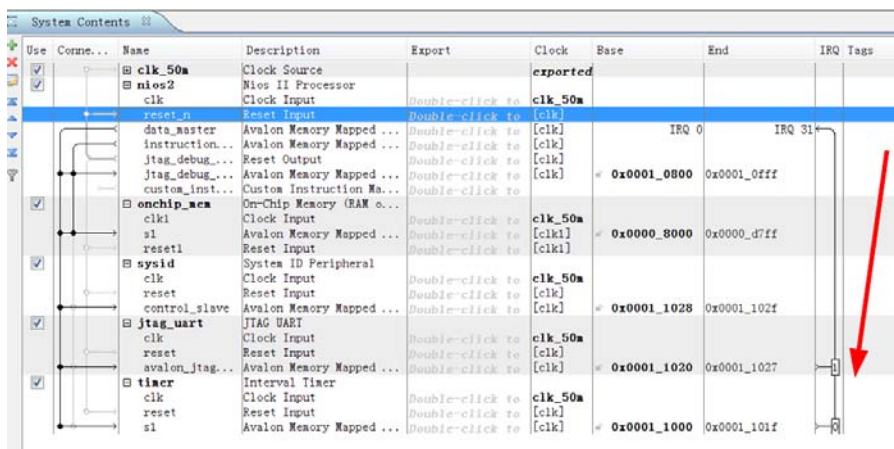
图 尚未连接的 Qsys 系统中断

我们点击 timer 组件所在行对应的 IRQ 列的空心圆后，如图所示，出现了数字“0”，表示它们已经连接上，在 NIOS II 处理器编程时就可以接收到 timer 组件发出的中断请求，并且数字“0”代表 timer 组件的中断号，这个号码越低优先级越高，“0”即最高优先级。

<input checked="" type="checkbox"/>		avalon_jtag...	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1020	0x0001_1027		
<input checked="" type="checkbox"/>		timer	Interval Timer						
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]				
<input checked="" type="checkbox"/>		sl	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1000	0x0001_101f	0	

图 timer 组件的中断连接

接着点击 jtag_uart 组件在 IRQ 一列的空心圆，如图所示，相应的连接了这些中断，并且分配了中断优先级。



Use	Corne...	Name	Description	Export	Clock	Base	End	IRQ Tags
<input checked="" type="checkbox"/>		clk_50m	Clock Source	exported				
<input checked="" type="checkbox"/>		nios2	Nios II Processor					
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m			
<input checked="" type="checkbox"/>		reset_n	Reset Input	Double-click to	[clk]			
<input checked="" type="checkbox"/>		data_master	Avalon Memory Mapped ...	Double-click to	[clk]			
<input checked="" type="checkbox"/>		instruction...	Avalon Memory Mapped ...	Double-click to	[clk]			
<input checked="" type="checkbox"/>		jtag_debug...	Reset Output	Double-click to	[clk]			
<input checked="" type="checkbox"/>		jtag_debug...	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_0800	0x0001_0fff	
<input checked="" type="checkbox"/>		custom_inst...	Custom Instruction Ma...	Double-click to	[clk]			
<input checked="" type="checkbox"/>		onchip_mcm	On-Chip Memory (RAM o...					IRQ 0
<input checked="" type="checkbox"/>		clk1	Clock Input	Double-click to	clk_50m			
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped ...	Double-click to	[clk1]	0x0000_8000	0x0000_d7ff	
<input checked="" type="checkbox"/>		reset1	Reset Input	Double-click to	[clk1]			
<input checked="" type="checkbox"/>		sysid	System ID Peripheral					
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m			
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]			
<input checked="" type="checkbox"/>		control_slave	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1028	0x0001_102f	
<input checked="" type="checkbox"/>		jtag_uart	JTAG UART					
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m			
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]			
<input checked="" type="checkbox"/>		avalon_jtag...	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1020	0x0001_1027	
<input checked="" type="checkbox"/>		tlcr	Interval Timer					
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to	clk_50m			
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to	[clk]			
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped ...	Double-click to	[clk]	0x0001_1000	0x0001_101f	

图 连接好的 Qsys 系统中断

4.2 地址分配

在菜单栏中，如图所示，我们可以点击“System → Assign Base Address”让工具自动进行地址分配。

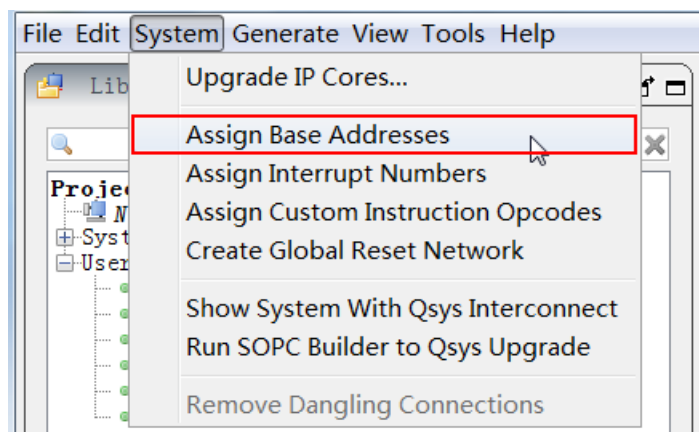


图 自动地址分配菜单

分配好地址后如图所示。

Address Map	Instance Parameters	Project Settings	Connections - Beta	Data Path - Beta
	nios2.data_master		nios2.instruction_master	
sysid.control_slave	0x0001_1028 - 0x0001_102f			
jtag_uart.avalon_jtag...	0x0001_1020 - 0x0001_1027			
onchip_mem.sl	0x0000_8000 - 0x0000_d7ff		0x0000_8000 - 0x0000_d7ff	
timer.sl	0x0001_1000 - 0x0001_101f			
nios2.jtag_debug_module	0x0001_0800 - 0x0001_0fff		0x0001_0800 - 0x0001_0fff	

图 做好地址映射的列表

4.3 系统生成

最后，如图所示，我们要点击菜单栏“Generate → Generate ...”进行 Qsys 系统生成。

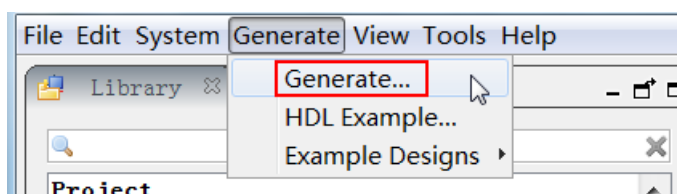


图 系统生成菜单

如图所示，对弹出的 Generation 对话框做设置后，点击右下角的“Generate”按钮。

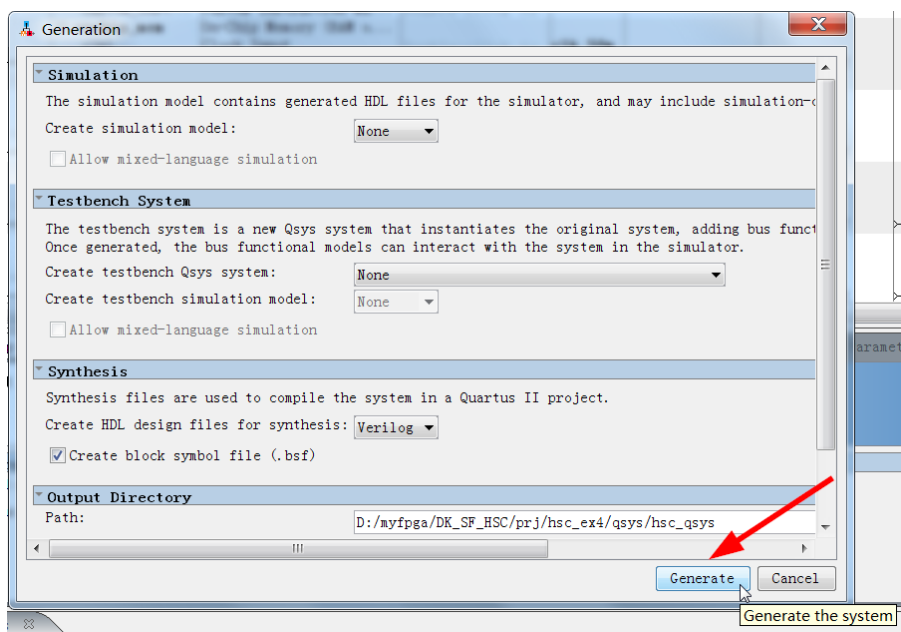


图 Generation 对话框

若之前未作保存，则会弹出如图所示的保存对话框，点击“Save”。

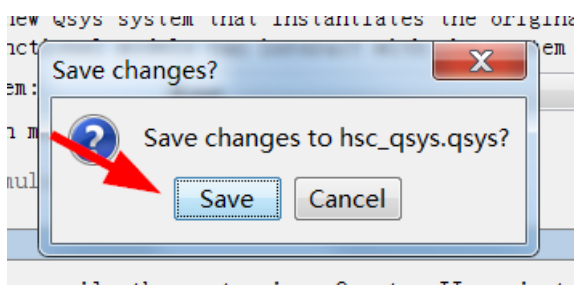


图 保存窗口

完成系统生成后，如图所示。

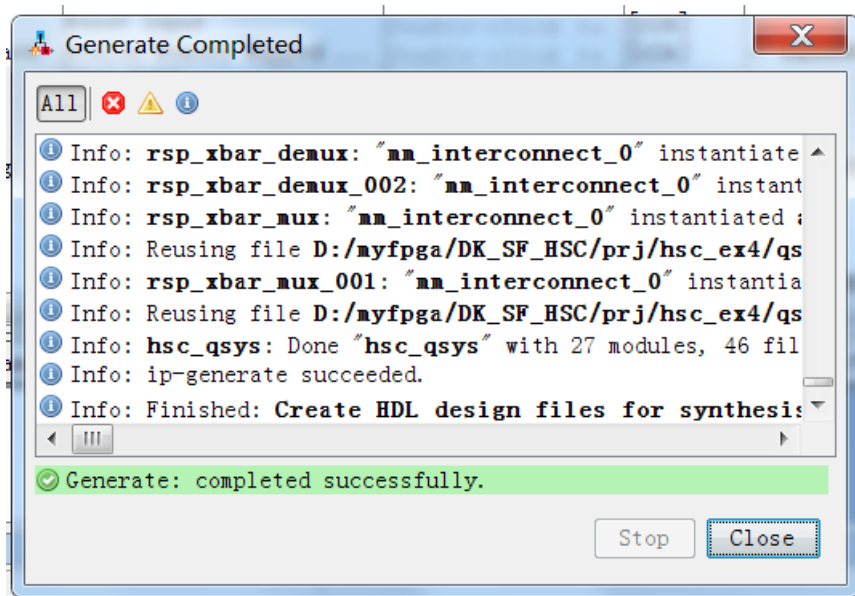


图 系统生成完毕

4.4 Qsys 系统例化模板

如图所示，点击 Qsys 菜单栏 “Generate → HDL Example ...”。

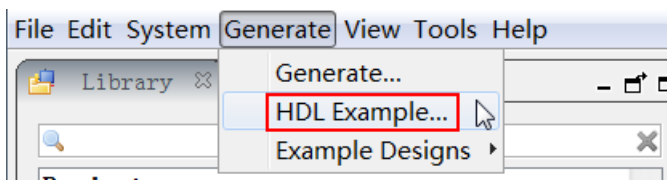


图 HDL 例化模板菜单

弹出来 HDL 例化模板如图所示，我们可以选择模板为 VHDL 或 Verilog 代码语言，我们这个实例使用 Verilog 模板，可以复制 Verilog 代码模板，随后粘贴到 Quartus II 工程源码中进行映射编辑。

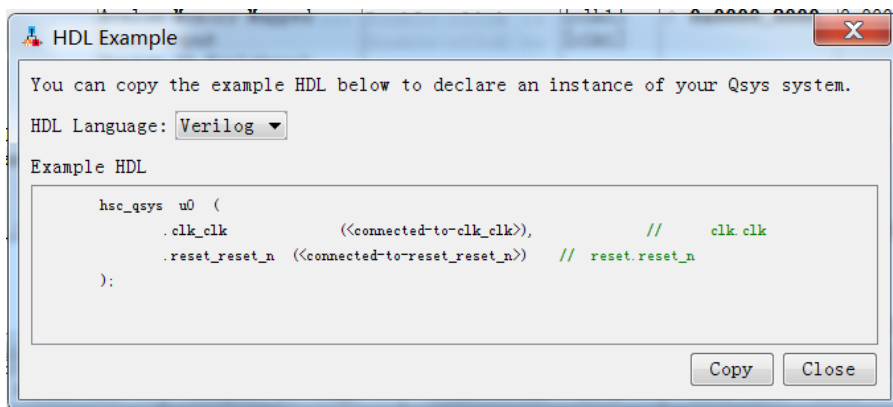


图 HDL 例化模板

5 Quartus II 工程设计实现

5.1 Verilog 顶层文件设计

回到 Quartus II 中，我们需要先创建一个 Quartus II 工程。并创建的 Verilog 代码源文件，命名为 hsc.v，作为工程的顶层源文件，保存在工程目录下的 source_code 文件夹下。源码如下，这里例化了一个 PLL，产生 Qsys 系统所需的 50MHz 时钟；也例化了 Qsys 系统。

```
module hsc(  
    //外部输入时钟和复位接口  
    input ext_clk,        //外部 25MHz 输入时钟  
    input ext_rst_n,      //外部低电平复位信号输入  
    //LED 指示灯接口  
    output led //用于测试的 LED 指示灯  
);
```

```
////////////////////////////////////
```

```
//系统内部时钟和复位产生模块例化
```

```
    //PLL 输出复位和时钟，用于FPGA 内部系统
```

```
wire sys_rst_n; //系统复位信号，低电平有效
```

```
wire clk_25m;      //PLL 输出 25MHz
```

```
wire clk_33m;      //PLL 输出 33MHz
```

```
wire clk_50m;      //PLL 输出 50MHz
```

```
wire clk_65m;      //PLL 输出 65MHz
```

```
wire clk_100m;     //PLL 输出 100MHz
```

```
sys_ctrl    u1_sys_ctrl(  
    .ext_clk(ext_clk),  
    .ext_rst_n(ext_rst_n),  
    .sys_rst_n(sys_rst_n),  
    .clk_25m(clk_25m),  
    .clk_33m(clk_33m),  
    .clk_50m(clk_50m),  
    .clk_65m(clk_65m),  
    .clk_100m(clk_100m)  
);
```

```
////////////////////////////////////
```

```
//LED 闪烁逻辑产生模块例化
```

```
led_controller    u2_led_controller(  
    .clk(clk_25m),  
    .rst_n(sys_rst_n),  
    .led(led)  
);
```

```
////////////////////////////////////
```

```
//Qsys 系统例化
hsc_qsys      u3_hsc_qsys(
                .clk_clk      (clk_50m),           //
clk.clk
                .reset_reset_n (sys_rst_n) // reset.reset_n
            );

endmodule
```

5.2 语法检查

如图 8.1 所示，点击“Analysis & Elaboration”进行语法检查，通过后前面出现一个绿色的勾号。

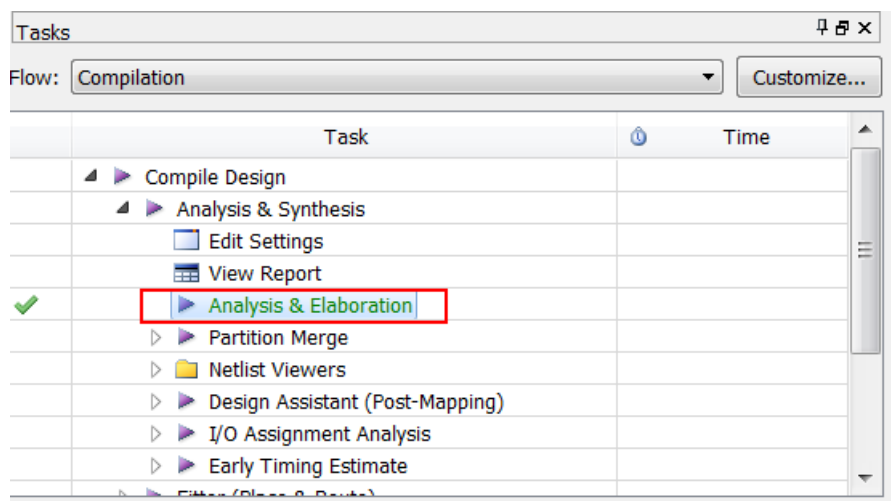


图 8.1 语法检查编译

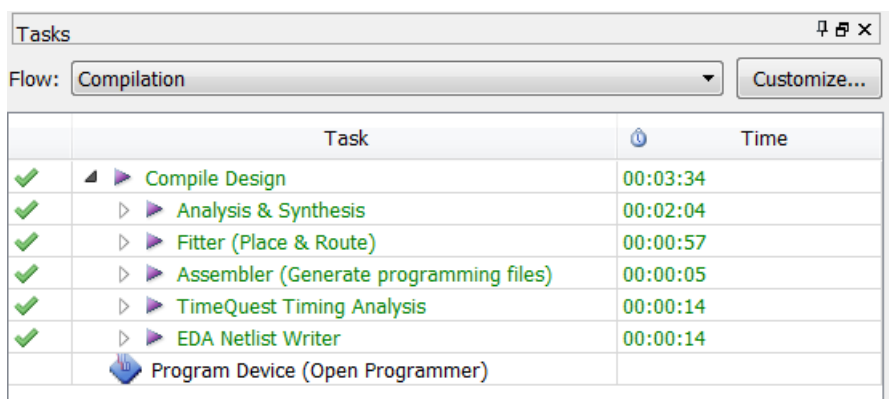
5.3 引脚分配

如图 8.2 所示，点击菜单栏“Assignments → Pin Planner”进入引脚分

配界面进行时钟和复位信号的引脚分配。

5.4 系统编译

点击菜单栏“Processing → Start Compilation”对整个 Quartus II 工程进行编译。完成编译后，如图所示，Compilation 窗口清一色的绿色勾勾表示通过编译。




Tasks			
Flow: Compilation		Customize...	
	Task		Time
✓	▶ Compile Design		00:03:34
✓	▶ Analysis & Synthesis		00:02:04
✓	▶ Fitter (Place & Route)		00:00:57
✓	▶ Assembler (Generate programming files)		00:00:05
✓	▶ TimeQuest Timing Analysis		00:00:14
✓	▶ EDA Netlist Writer		00:00:14
	 Program Device (Open Programmer)		

图 编译通过

至此，恭喜你，基于 Qsys 的 Quartus II 工程已经完成设计工作。后面我们就要迈入 NIOS II 处理器软件开发的大门了。

6 软件开发工具 EDS

6.1 EDS 软件开启

这节开始，我们就要专注于“高大上”的 NIOS II 处理器的嵌入式软件

开发工作了。首先我们要点击开始菜单，打开如图所示的“Nios II 13.1 Software Build Tools for Eclipse”（简称 EDS）。

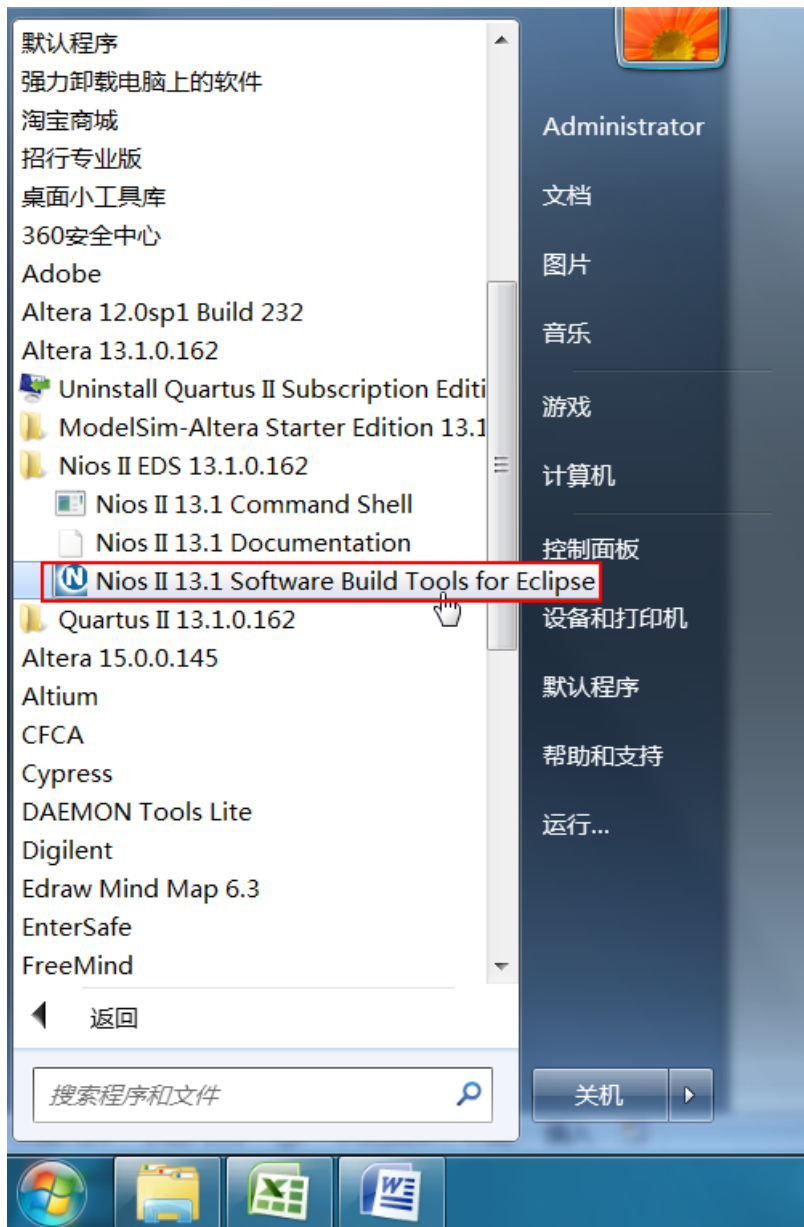


图 EDS 程序菜单

开启的 EDS 界面如图所示。

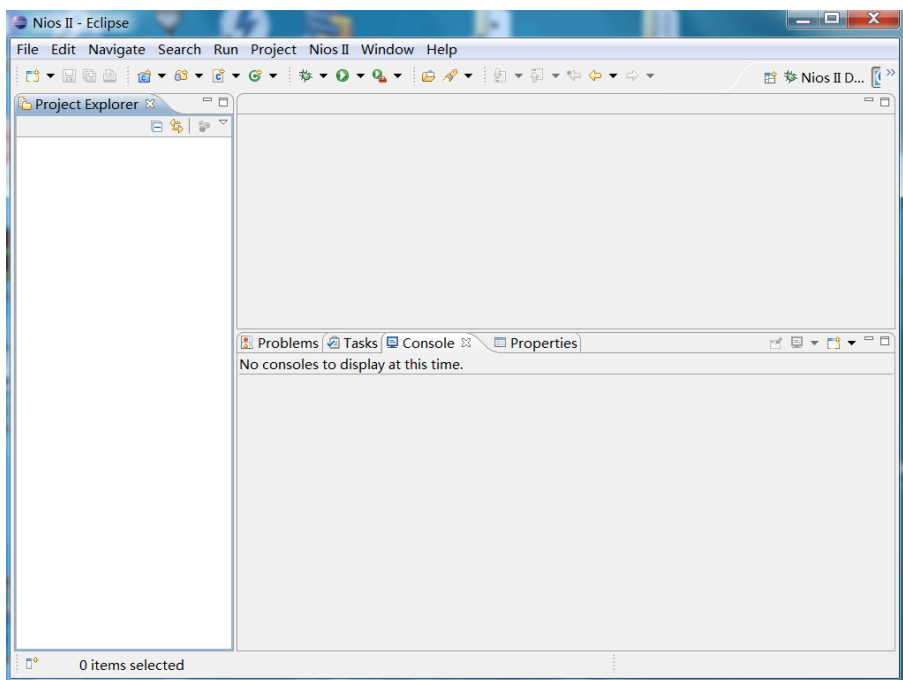


图 EDS 主界面

6.2 BSP 工程创建

如图所示，点击菜单栏“File → New → Nios II Application and BSP from Template”创建一个包含应用程序和 BSP 的工程。BSP 工程将会包含所有 Qsys 系统中的 NIOS II 处理器和外设组件的硬件信息，包括地址、中断优先级等硬件参数。

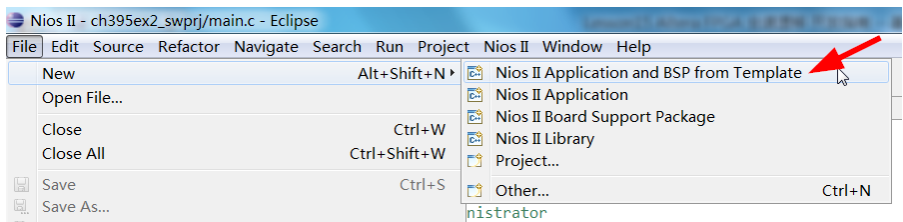


图 新建工程菜单

如图所示，我们首先选择 SOPC Information File name 为前面创建的 hsc_qsys.SOPCinfo 文件。cy4_qsys.SOPCinfo 文件是 Qsys 系统生成时一同产生的，它包含了所有 Qsys 系统的硬件信息，通过它导入到 BSP 工程，就使得 BSP 工程中也能获得所有 Qsys 系统的硬件信息。接着输入 Project name 为 hsc_swprj。Templates 选择 Blank Project。

其他设置默认即可，点击 Finish 完成 BSP 工程的创建。

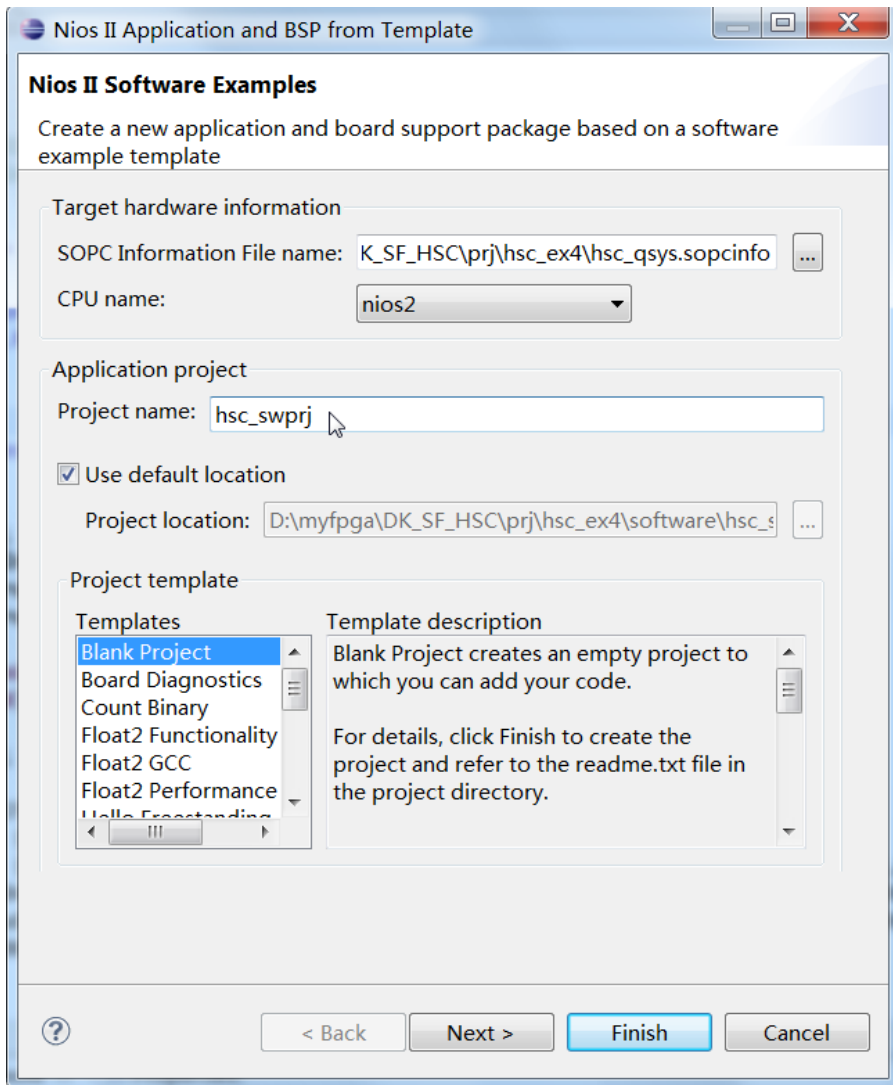


图 9.4 BSP 工程创建

这时我们可以看到在 Project Explorer 下出现了新建的 hsc_swprj 和 hsc_swprj_bsp 工程目录，展开后如图所示，可以看到 hsc_swprj_bsp 这个文件夹下包含了各种和当前 Qsys 系统相关的板级驱动源文件和头文件，供应用软件调用。尤其图中打开的头文件 system.h，它将 Qsys 系统中的 NIOS II 处理器和所有外设的名称、基地址、中断有无以及优先级号码等相关硬件信息做了定义。当然，它不是平白无故生成的，是我们在图加载的 hsc_qsys.SOPCinfo 文件导入的。hsc_swprj 文件夹则用于创建用户的应用程序。

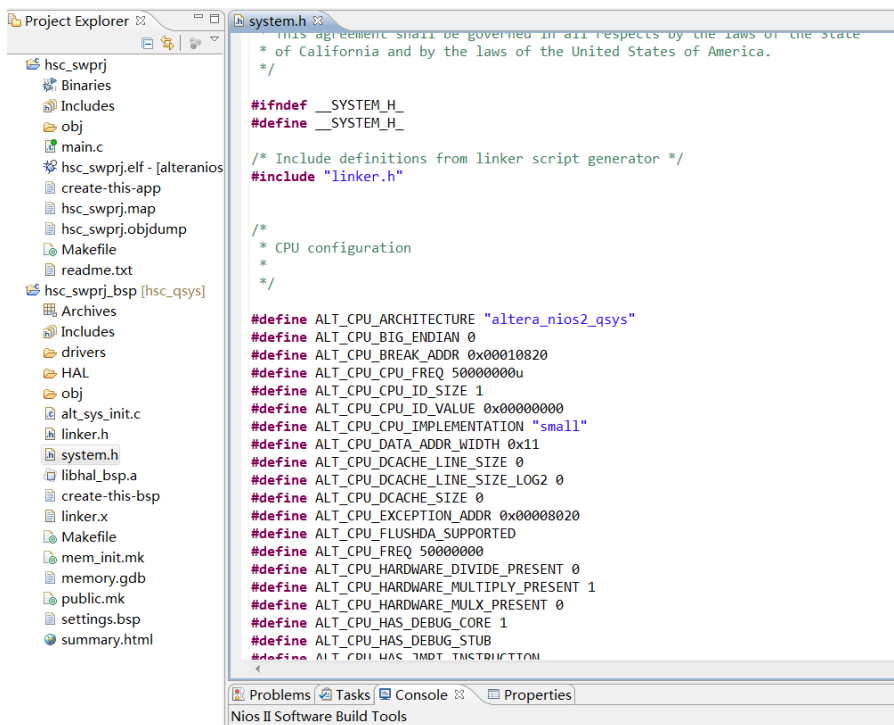


图 9.5 nios2bsp 工程目录

6.3 开启 BSP Editor

BSP Editor 顾名思义，BSP 工程的“编辑器”，功能如同一般的“属性”窗口。在 BSP Editor 中，我们可以对板级驱动层做一些定制化的配置，比如代码裁剪、标准输入输出外设和定时器外设的设置等。

如图所示，在新创建的 hsc_swprj_bsp 工程中单击鼠标右键，选择菜单“Nios II → BSP Editor ...”。

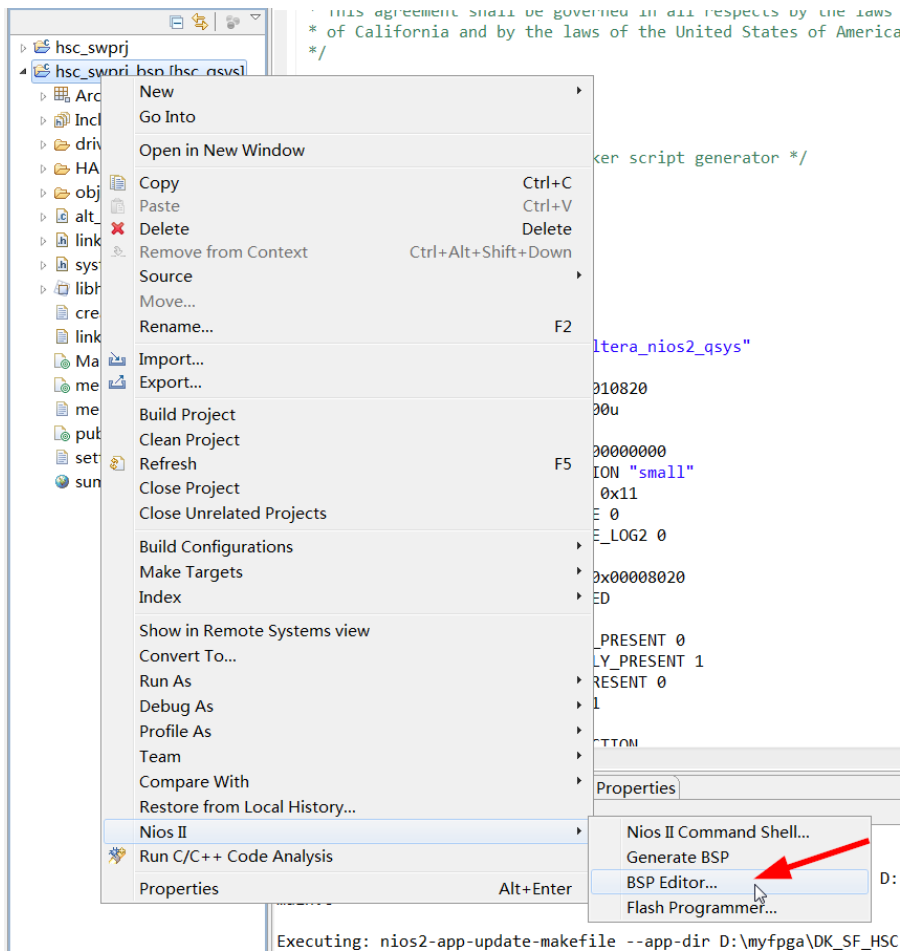


图 开启 BSP Editor 菜单

BSP Editor 界面如图所示，我们主要在 Main 选项卡中对 BSP 的驱动设

置做更改。

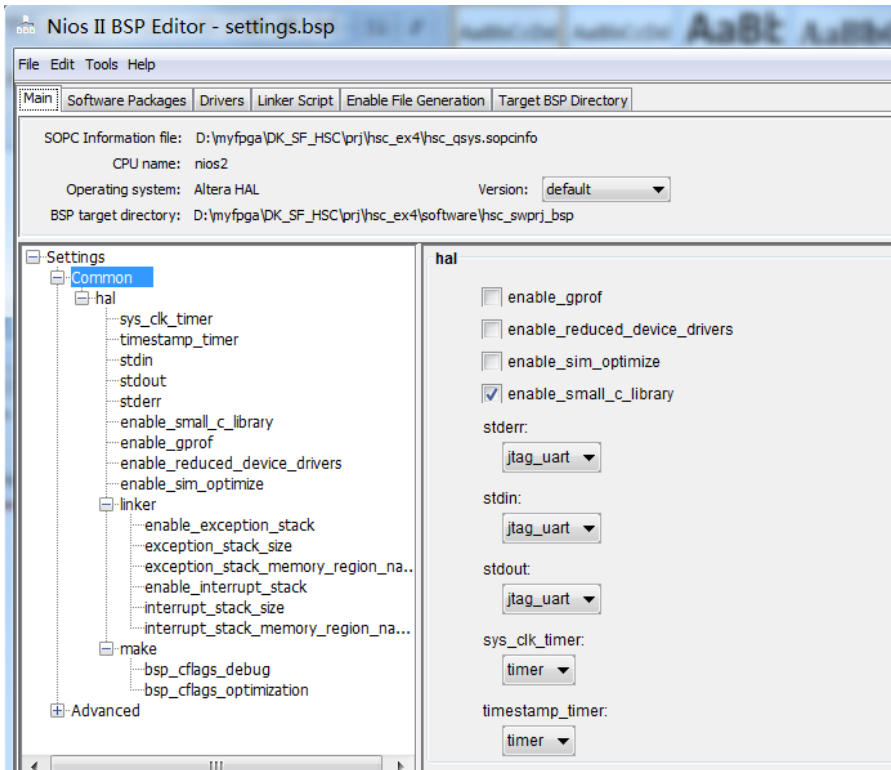


图 BSP Editor 窗口

6.4 BSP Editor 设置

由于我们使用了存储量有限的 FPGA 片内 RAM 作为 NIOS II 处理器数据和程序存储器,因此需要对 BSP 做裁剪,以减少不需要的一些驱动层代码。

在 BSP Editor 中,选中左侧的 Settings, 右侧相应设置如图所示(未出现在图中的使用默认设置即可)。

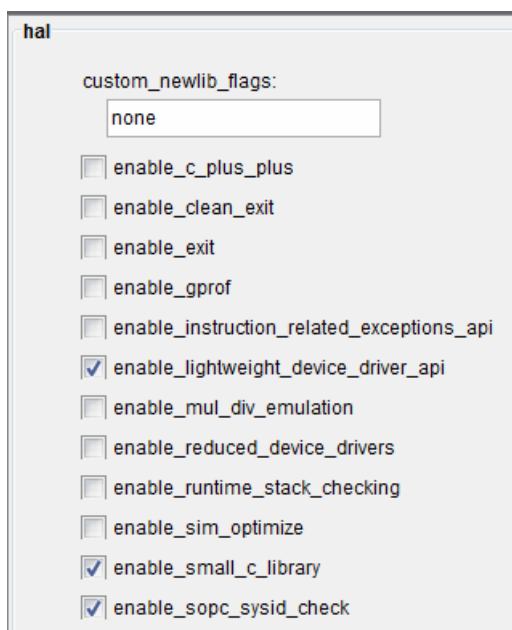


图 Settings 设置 1

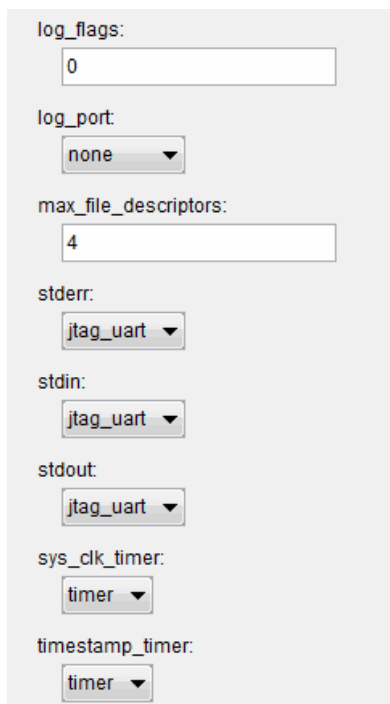


图 Seetings 设置 2

选择左侧的 **Advanced**，右侧设置如图所示（未出现在图中的使用默认

设置即可)。

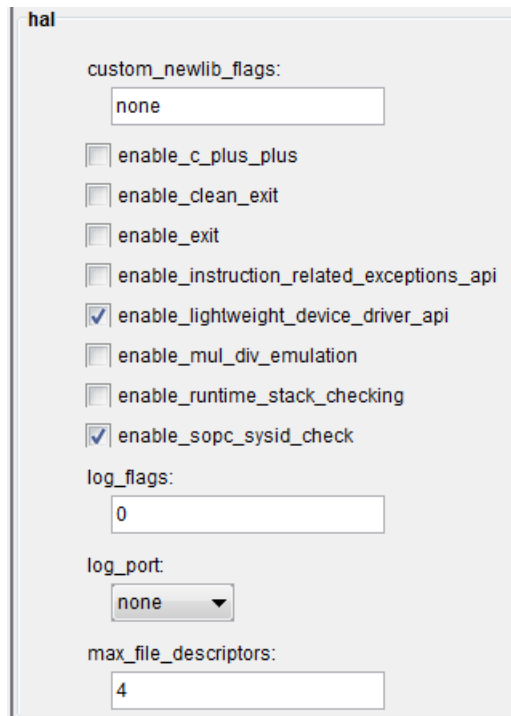


图 Advanced 设置

完成设置后，如图所示，点击 BSP Editor 右下角的 Generate 按钮完成设置，最后点击 Exit 退出。

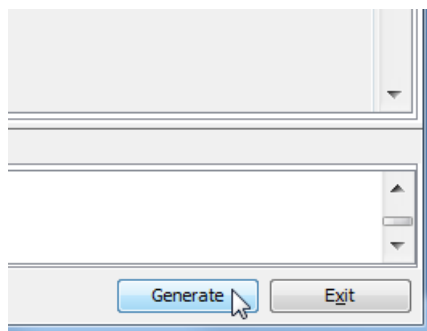


图 Generate 按钮

6.5 BSP 工程编译

回到 Project Explorer 中，鼠标右键点击 hsc_swprj_bsp 工程文件夹，弹出菜单中选择“Builde Project”进行工程编译。

编译完成后，如图所示，Console 中有“Build Finished”的提示。

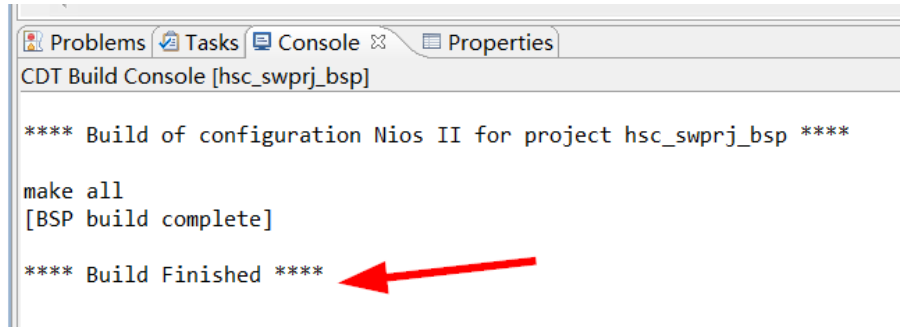


图 Console 面板打印编译信息

6.6 C 代码源文件创建

如图所示，在 Project Explorer 的软件应用工程 hsc_swprj 上海，鼠标右键单击它，在弹出菜单中选择“New → Source File”。

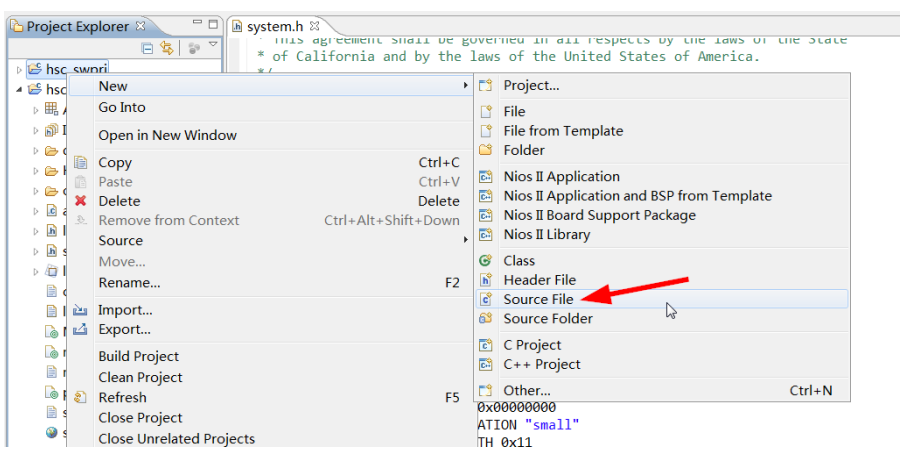


图 新建 C 源文件菜单

如图所示，创建一个 main.c 源文件。

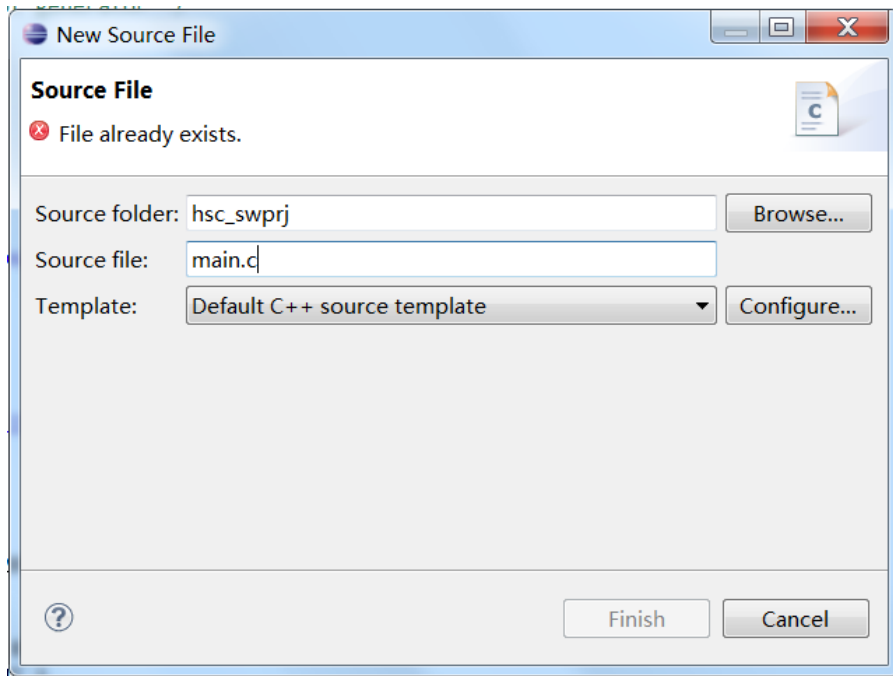


图 新建 main.c 源文件

此时，我们可以看到 hsc_swprj 文件夹下出现了新建的 main.c 文件，并且处于打开可编译的状态。我们可以在 main.c 中编写一个简单的例程。该例程实现 JTAG UART 的 ID 信息读取操作。

```
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "system.h"
#include <stdio.h>
#include <unistd.h>
#include "altera_avalon_sysid_qsys.h"
#include "altera_avalon_sysid_qsys_regs.h"

////////////////////////////////////

//函数名:   main
//功 能:   主函数，读取 System ID 的 ID 值和 timestamp 值，通过 JTAG
```

```

UART 打印
//参 数： 无
//返 回： int
//备 注：
////////////////////////////////////
int main(void)
{

    /* Read the hardware-tag, aka value0, from the hardware. */
    alt_u32                hardware_id                =
IORD_ALTERA_AVALON_SYSID_QSYS_ID(SYSID_BASE);

    /* Read the time-of-generation, aka value1, from the hardware
register. */
    alt_u32                hardware_timestamp          =
IORD_ALTERA_AVALON_SYSID_QSYS_TIMESTAMP(SYSID_BASE);

    printf("System ID is 0x%8x\n", hardware_id);
    printf("System timestamp is 0x%8x\n", hardware_timestamp);

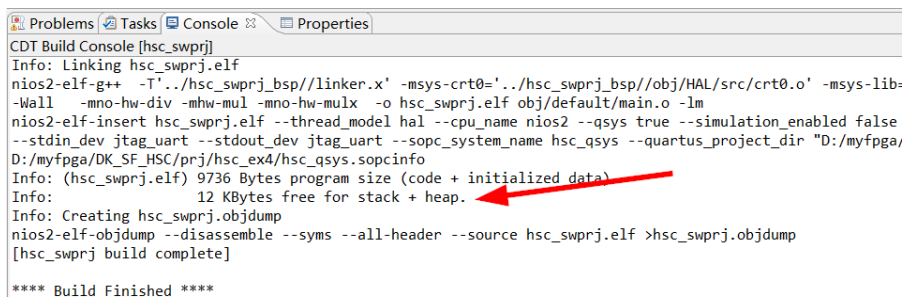
    while(1);
    return 0;
}

```

6.7 软件应用工程编译

在软件应用工程 `nios2ex1` 上右键单击鼠标，选择菜单“Build Project”对工程进行编译。

编译完成后，如图所示，这里有软件应用工程的总代码量和余下可用的代码空间。



```
CDT Build Console [hsc_swprj]
Info: Linking hsc_swprj.elf
nios2-elf-g++ -T'../hsc_swprj_bsp/linker.x' -msys-crt0='../hsc_swprj_bsp/obj/HAL/src/crt0.o' -msys-lib=
-Wall -mno-hw-div -mhw-mul -mno-hw-mulx -o hsc_swprj.elf obj/default/main.o -lm
nios2-elf-insert hsc_swprj.elf --thread_model hal --cpu_name nios2 --qsys true --simulation_enabled false
--stdin_dev jtag_uart --stdout_dev jtag_uart --sopc_system_name hsc_qsys --quartus_project_dir "D:/myfpga/
D:/myfpga/DK_SF_HSC/prj/hsc_ex4/hsc_qsys.sopcinfo
Info: (hsc_swprj.elf) 9736 Bytes program size (code + initialized data)
Info: 12 KBytes free for stack + heap.
Info: Creating hsc_swprj.objdump
nios2-elf-objdump --disassemble --syms --all-header --source hsc_swprj.elf >hsc_swprj.objdump
[hsc_swprj build complete]

**** Build Finished ****
```

图 Console 面板打印编译信息

6.8 移除当前工程

若希望在 EDS 移除当前工程（注意不是删除，只是将当前工程从 EDS 中移除，相当于关闭工程），鼠标右键点击工程，弹出菜单中选择“Delete”。

6.9 加载工程

在 Project Explorer 的空白处，单击鼠标右键，选择菜单“Import ...”。然后如图所示，选择“General → Existing Projects into Workspace”，再点击“Next”。

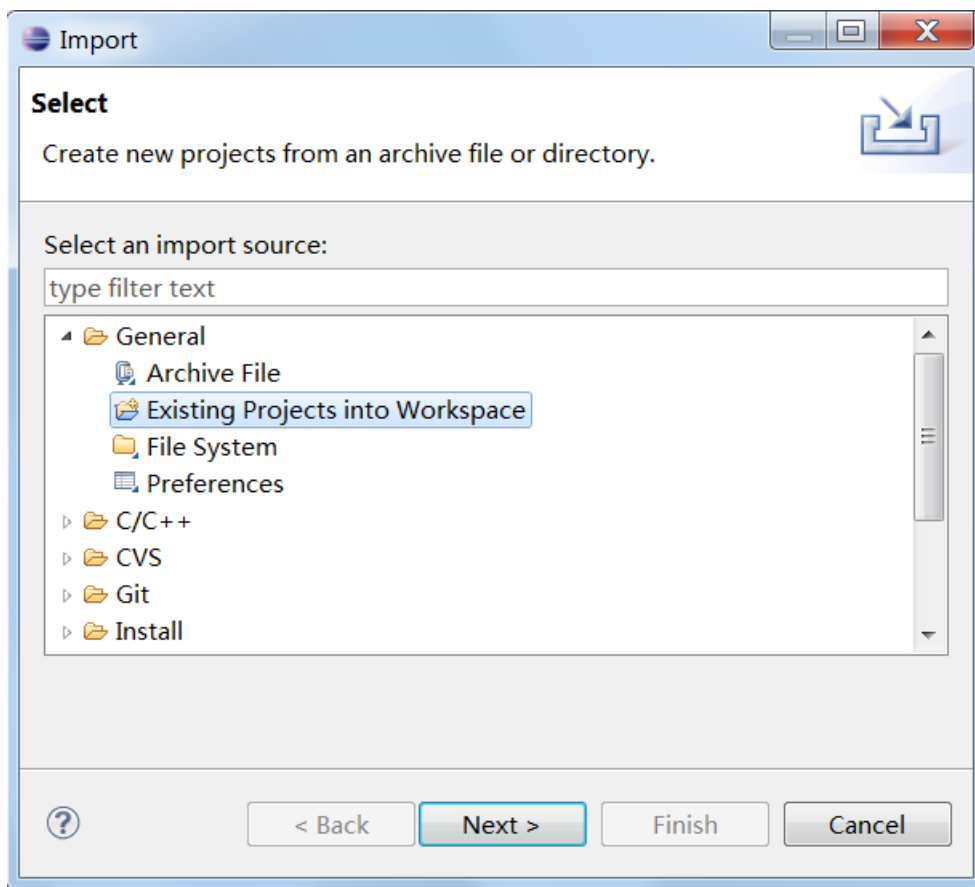


图 9.27 选择导入已有工程

如图所示，将 **Select root directory** 定位到希望加载的工程所在路径，EDS 会自动识别是否有软件工程供加载，罗列在 **Projects** 下。我们可以勾选希望加载的工程，然后点击“**Finish**”完成加载。

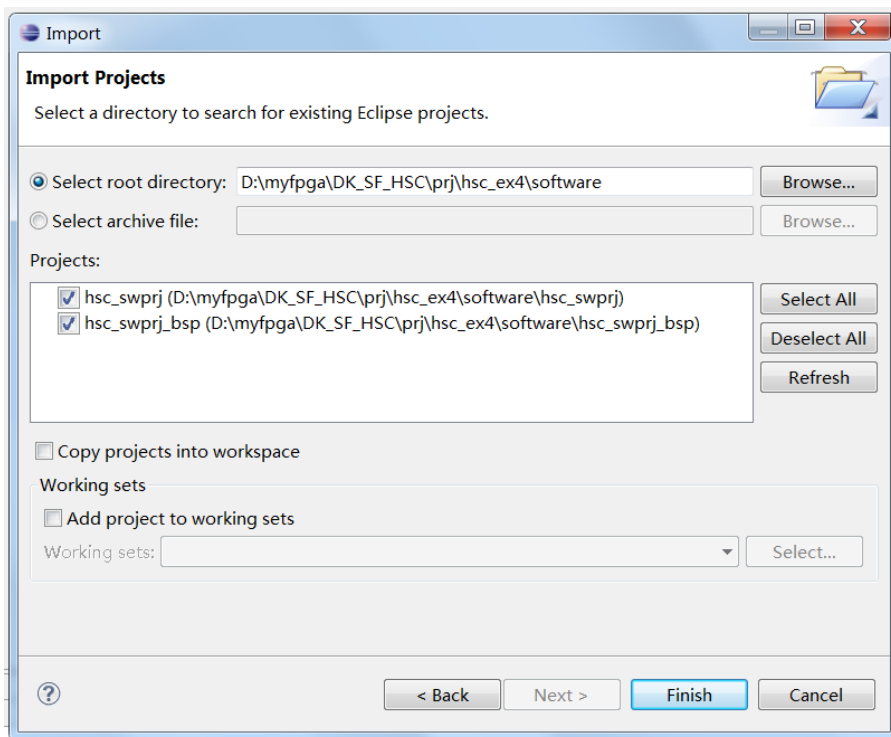


图 选择导入工程路径

6.10 移植工程

如果将一个工程（注意这个工程必须是完整的包括 Quartus II 工程和软件 BSP 工程和应用工程）的从原有的路径拷贝到另一个不同的路径下，那么需要做一些路径上的修改，才能够正常编译。

如图所示，打开 BSP 工程文件夹，双击 `settings.bsp` 文件。

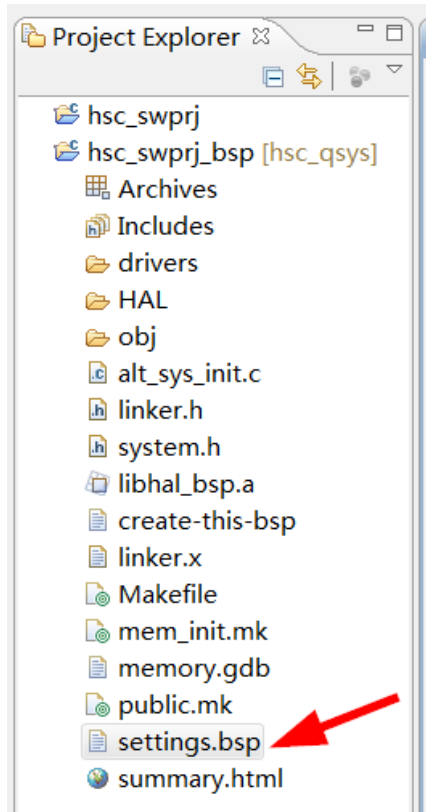


图 打开 settings.bsp 文件

如图所示,将高亮3行中的 BspGeneratedLocation 路径和 SopcDesignFile 路径修改为新的工程文件夹路径即可完成移植。

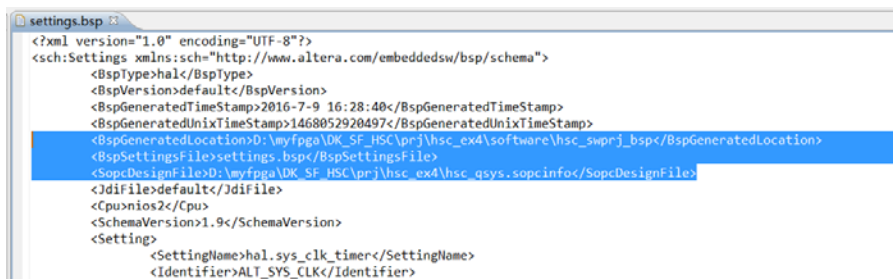


图 修改 setting.bsp 中的路径

7 System ID 与 Timestamp 读取板级调试

7.1 软件功能概述

System ID 外设有两个寄存器，其描述如表所示。

表 System ID 外设寄存器定义

地址偏移	寄存器名称	读/写	功能描述
0	id	读	基于 Qsys 系统定义的唯一的 32 位数值。该 id 值类似于校验和；不同组件、不同选项配置的 Qsys 系统则产生不同的 id 值。
1	timestamp	读	基于系统生成时间的唯一 32 位值。该值等效于从 1970 年 1 月 1 日以来所经过的总秒数。

简单说，System ID 组件的 id 值为 32 位，Qsys 中添加该组件时我们就设置好了。System ID 组件 timestamp 寄存器取值为自 1970 年 1 月 1 日以来到该外设生成时的总秒数。

本实例读取 System ID 外设的两个寄存器值，一个是 id 值，另一个是 Timestamp 值。软件流程如图所示。

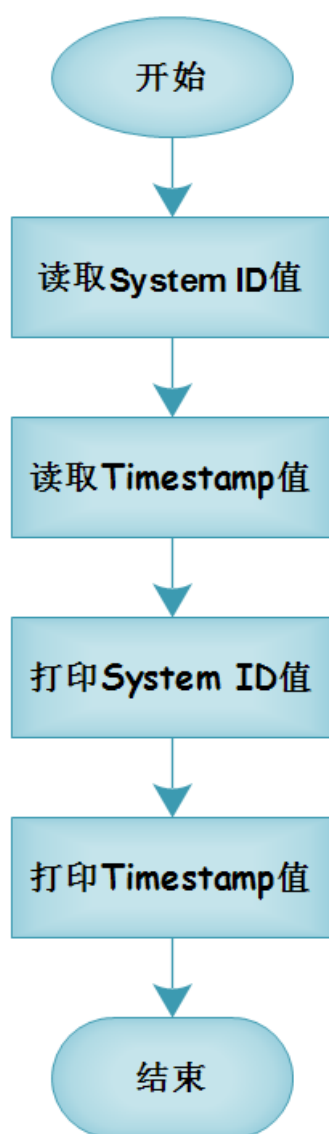


图 System ID 与 Timestamp 实例软件流程图

7.2 软件代码解析

本实例的软件代码如下。

```
#include "alt_types.h"
```

```

#include "sys/alt_irq.h"
#include "system.h"
#include <stdio.h>
#include <unistd.h>
#include "altera_avalon_sysid_qsys.h"
#include "altera_avalon_sysid_qsys_regs.h"

////////////////////////////////////

//函数名:   main
//功 能:   主函数, 读取 System ID 的 ID 值和 timestamp 值, 通过 JTAG
UART 打印
//参 数:   无
//返 回:   int
//备 注:

////////////////////////////////////

int main(void)
{

    /* Read the hardware-tag, aka value0, from the hardware. */
    alt_u32          hardware_id          =
IORD_ALTERA_AVALON_SYSID_QSYS_ID(SYSID_BASE);

    /* Read the time-of-generation, aka value1, from the hardware
register. */
    alt_u32          hardware_timestamp    =
IORD_ALTERA_AVALON_SYSID_QSYS_TIMESTAMP(SYSID_BASE);

    printf("System ID is 0x%8x\n", hardware_id);
    printf("System timestamp is 0x%8x\n", hardware_timestamp);

    while(1);
    return 0;
}

```

}

- 这里有 2 个和上一个实例不同的头文件,我们来看看它们的主要用处。
 - `alt_types.h` 中对 `altera` 定义的数据类型进行宏定义和申明。例如 `alt_u32` 表示 32bit 的无符号整型。
 - `altera_avalon_sysid_qsys_regs.h` 中定义了 System ID 硬件寄存器访问的接口函数。如

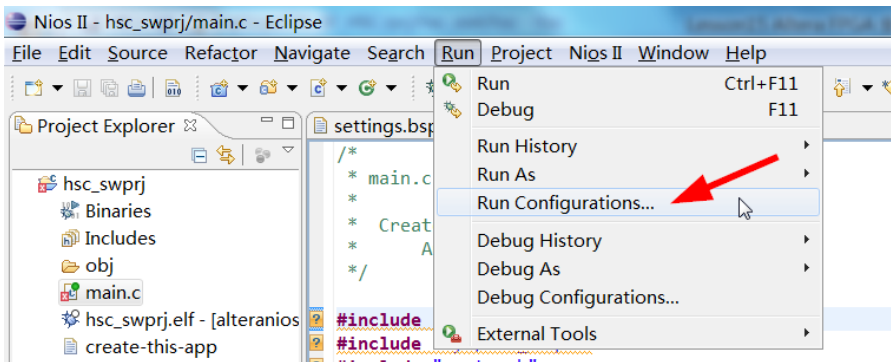
`IOR_D_ALTERA_AVALON_SYSID_QSYS_ID(SYSID_BASE)`函数和

`IOR_D_ALTERA_AVALON_SYSID_QSYS_TIMESTAMP(SYSID_BASE)` 函数。
- `IOR_D_ALTERA_AVALON_SYSID_QSYS_ID(SYSID_BASE)`函数即读取我们定义的 System ID 外设的 id 值, `SYSID_BASE` 是我们定义的 System ID 外设的基址。
- `IOR_D_ALTERA_AVALON_SYSID_QSYS_TIMESTAMP(SYSID_BASE)` 函数即读取我们定义的 System ID 外设的 Timestamp 值。

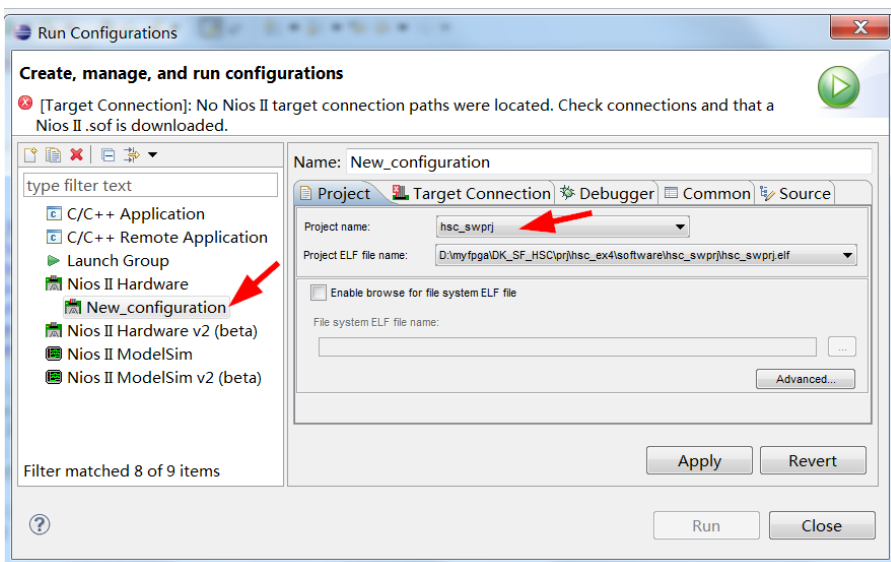
7.3 板级调试

首先,我们需要将 Quartus II 工程中产生的 `hsc.sof` 文件烧录到 SF-HSC 开发板的 FPGA 中(在这之前, SF-HSC 开发板和 PC 之间连接好下载器,给 SF-HSC 开发板上电)。

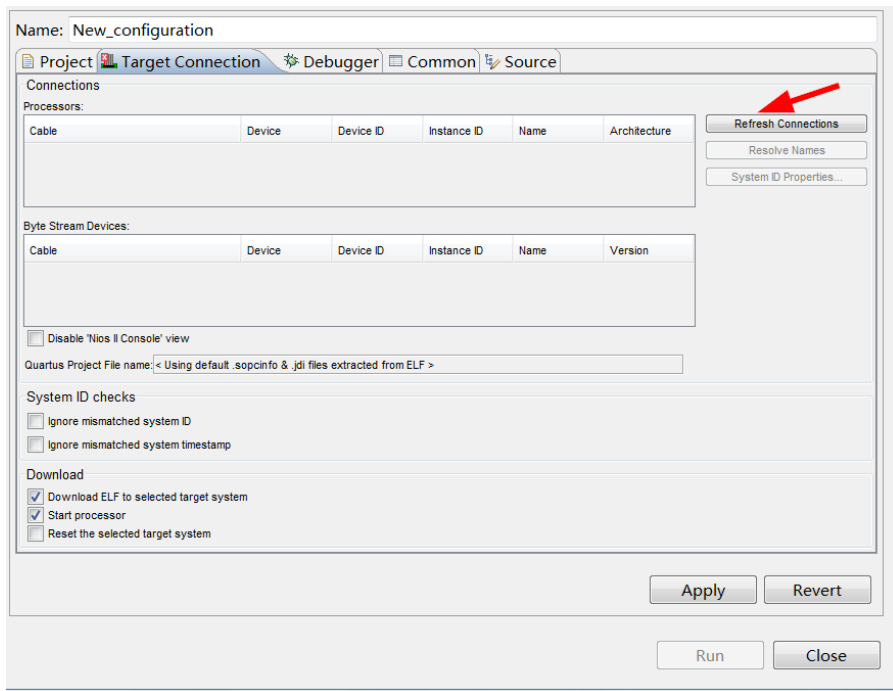
接着,在 EDS 下,将程序 Run 起来。点击菜单栏的“Run→Run Configurations...”,如图所示。



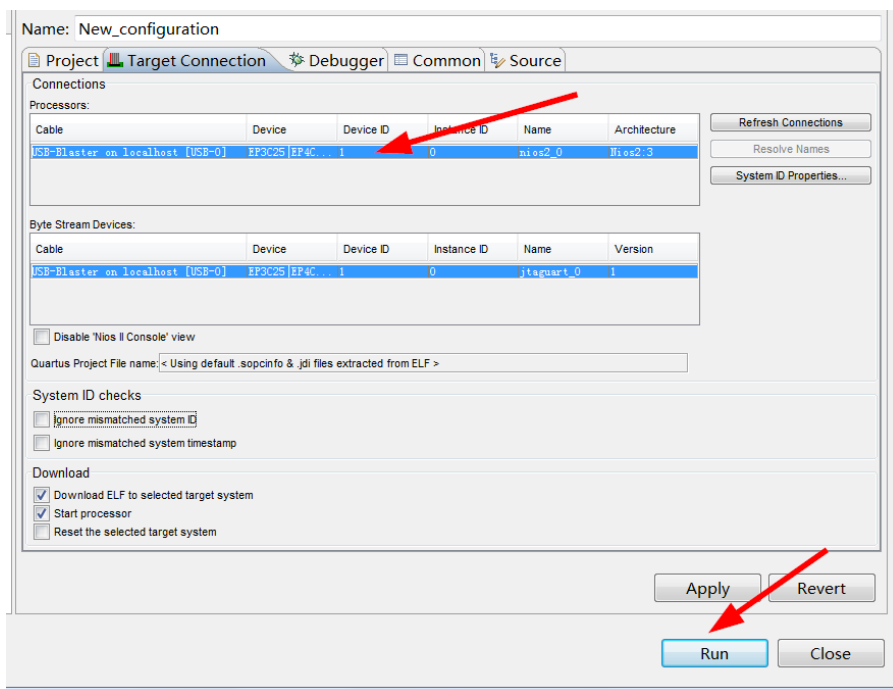
接着如图所示，选中“New_configuration”，Project name 选择“hsc_swprj”。



切换到“Target Connection”选项卡。如图所示，点击右侧的“Refresh Connection”。



直到 Cable 识别到了，并且 Run 按钮高亮，可以被点击。点击 Run。



片刻后，我们可以在 EDS 的 Nios Console 中看到如图所示打印出来的

字符串。

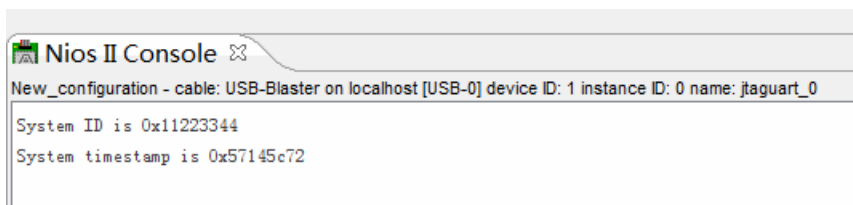


图 System ID 与 Timestamp 打印信息

System ID 的值正是我们在 Qsys 所设定的, 而 timestamp 的值大家可以自己换算一下, 是不是从 1970 年 1 月 1 日到今天所经过的秒数 (这个值大家根据自己实验得到的为准)。