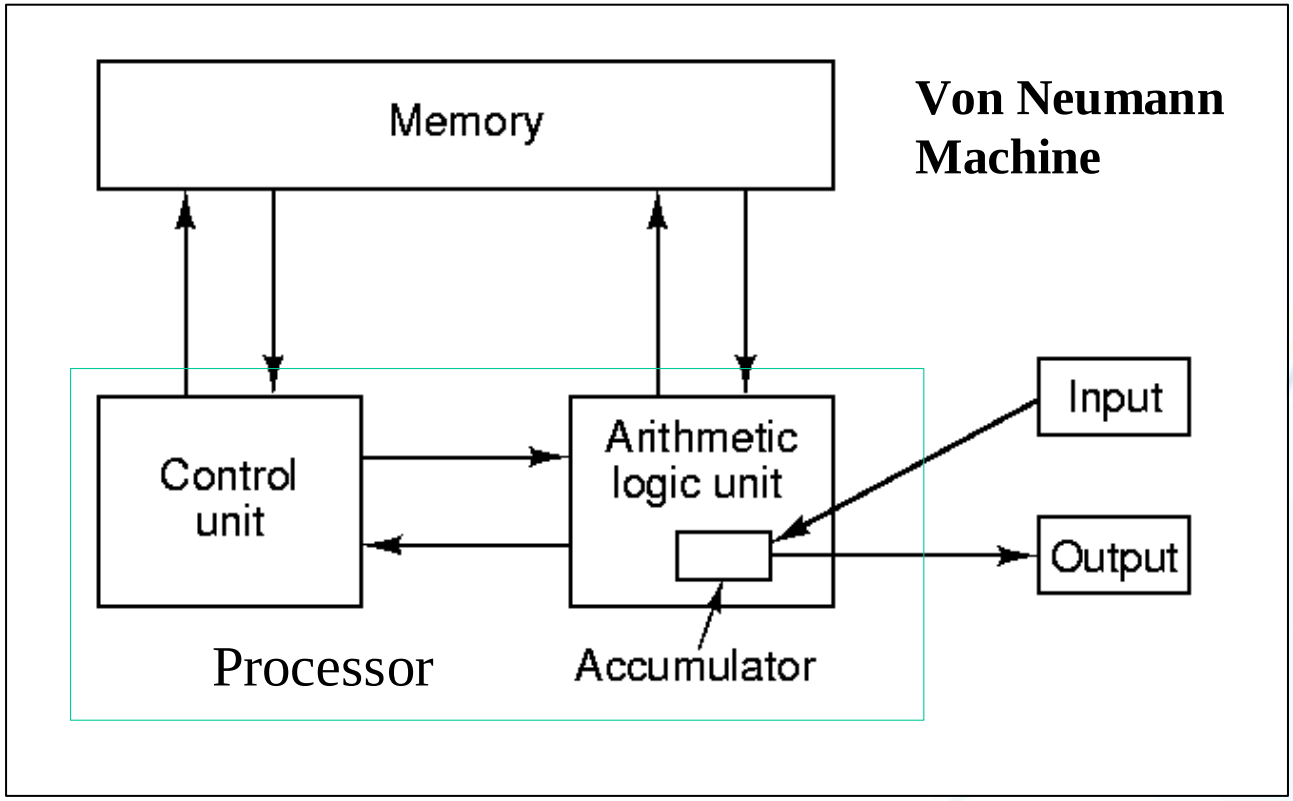
内容

- Nios II® 简介
- NiosII硬件构建
- Nios II软件开发

Nios II简介

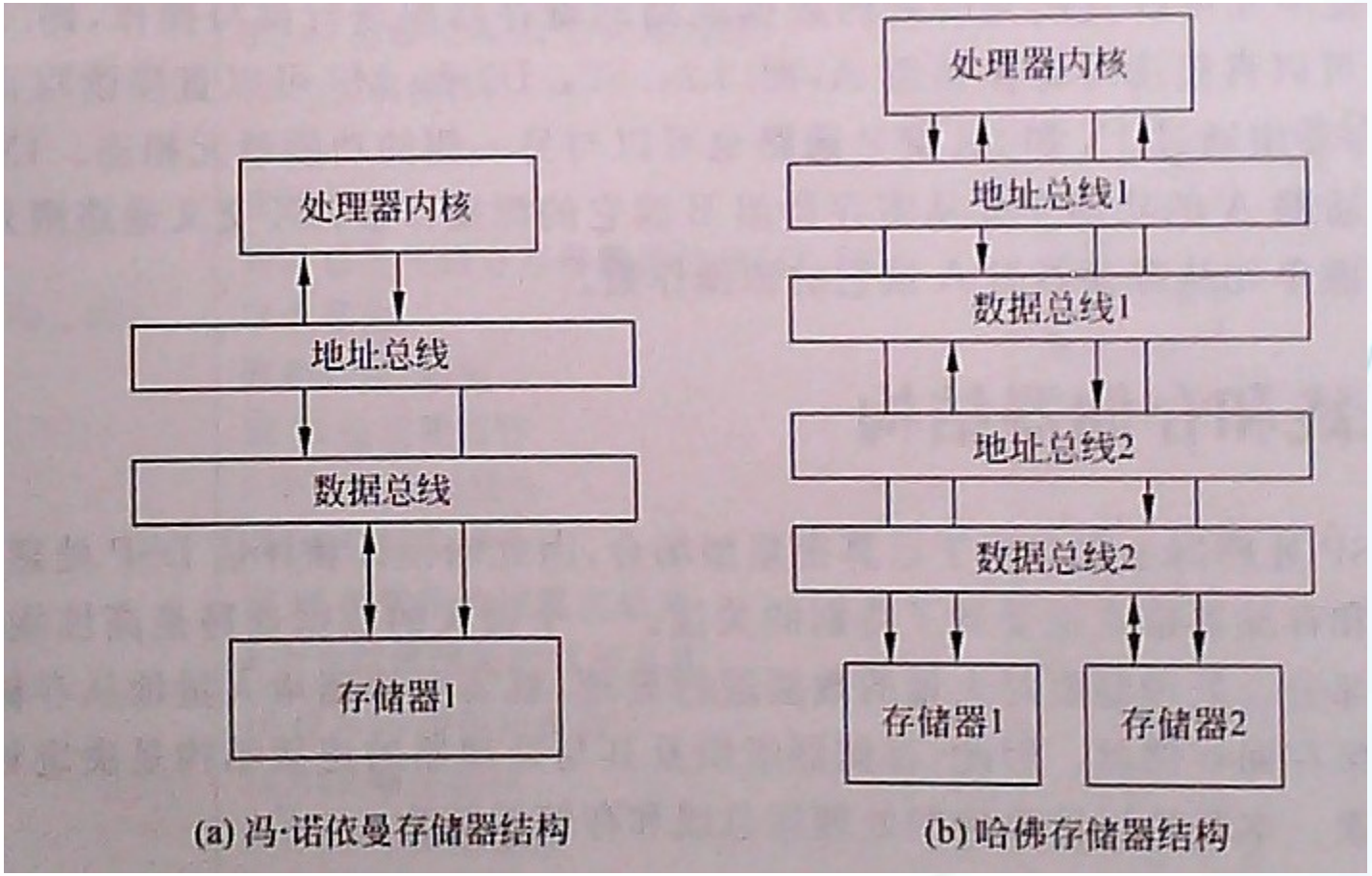
回顾

- 冯.诺依曼机


Memory
Von Neumann Machine
Control unit
Arithmetic logic unit
Input
Output
Processor
Accumulator

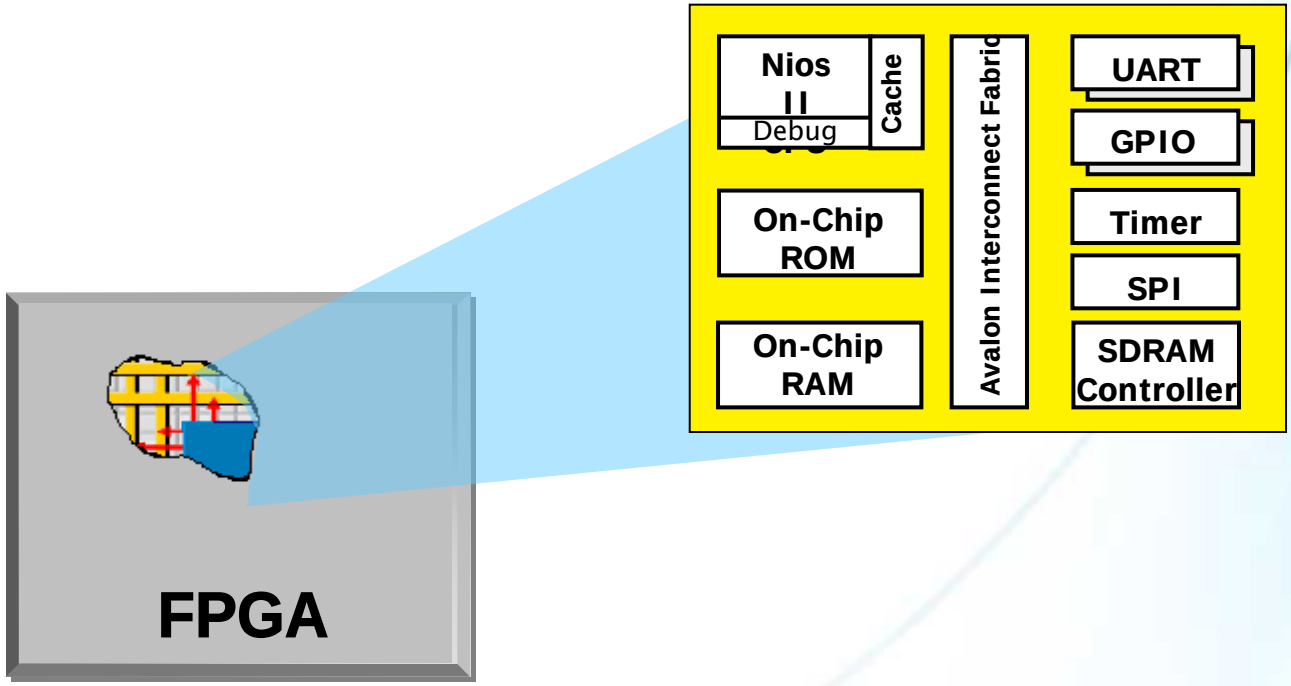
回顾

- 冯.诺依曼与哈佛结构



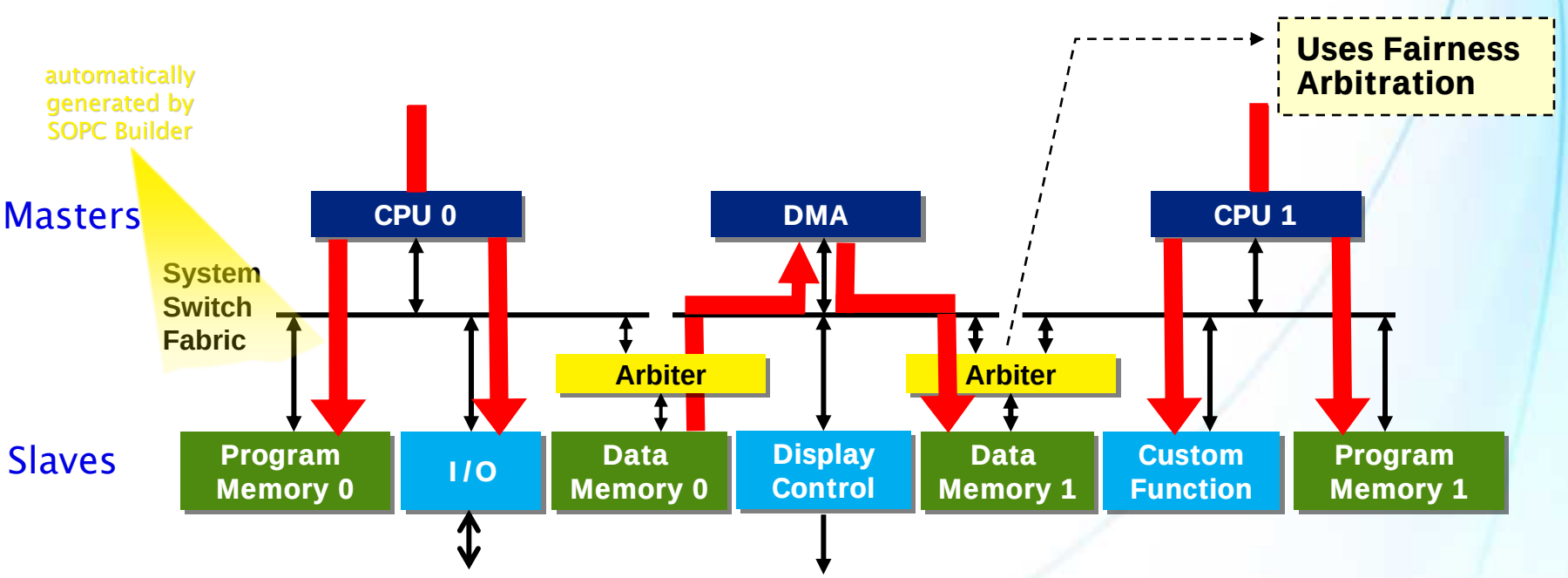
什么是Nios II?

- Altera的第二代32位软核RISC微处理器
  - Nios II以及所有外设以HDL源代码的方式提供
  - 可用于所有的Altera FPGA
  - 使用Quartus II集成综合工具进行综合

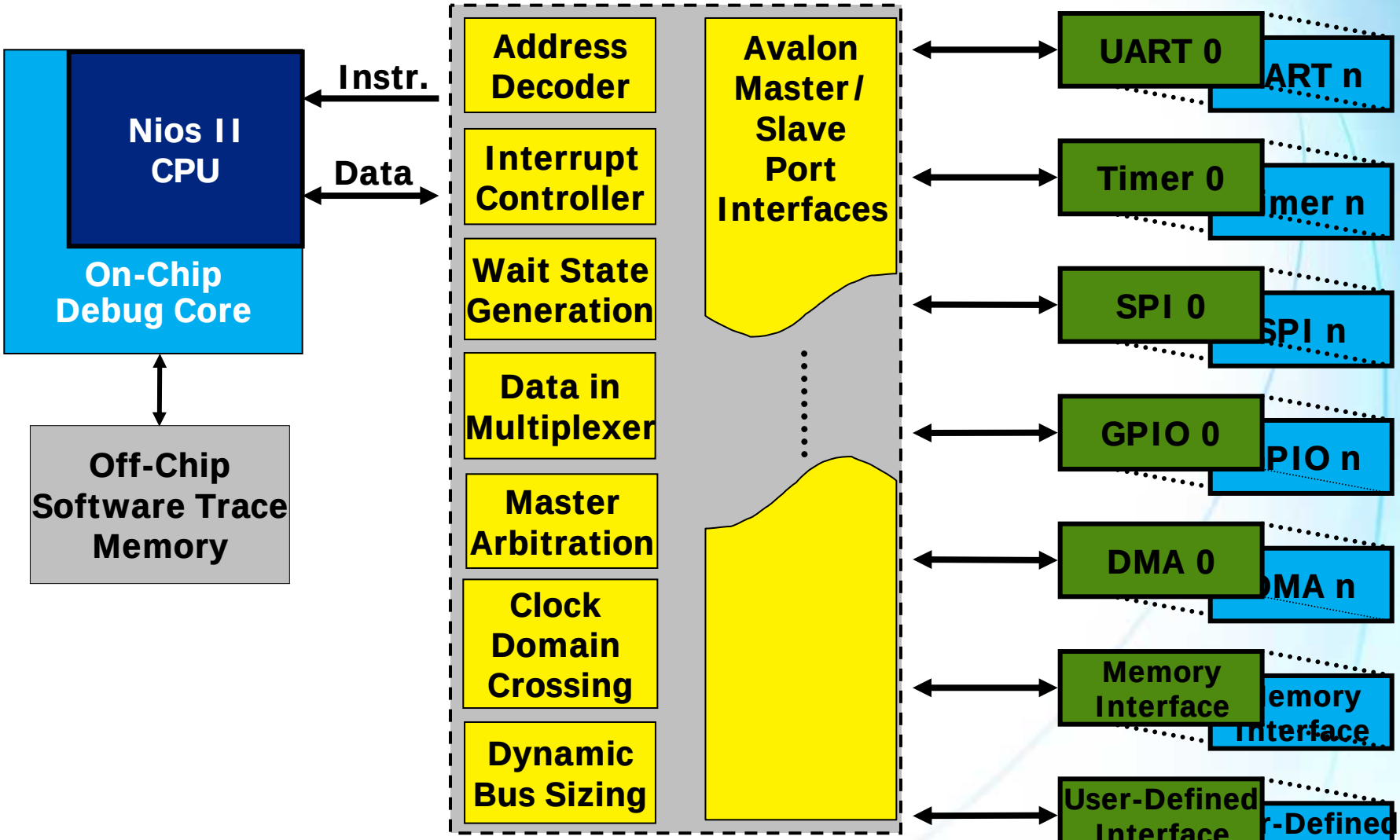


NiosII的系统互连架构

- 主从设备之间实现点到点互联，多主设备同时工作（不在同一时钟周期访问同一设备）；
- 多主设备在同一时钟周期访问同一设备时，由从端仲裁器协调；
- 突破传统总线的局限



典型的Nios II系统体系结构：任意添加剪裁外设

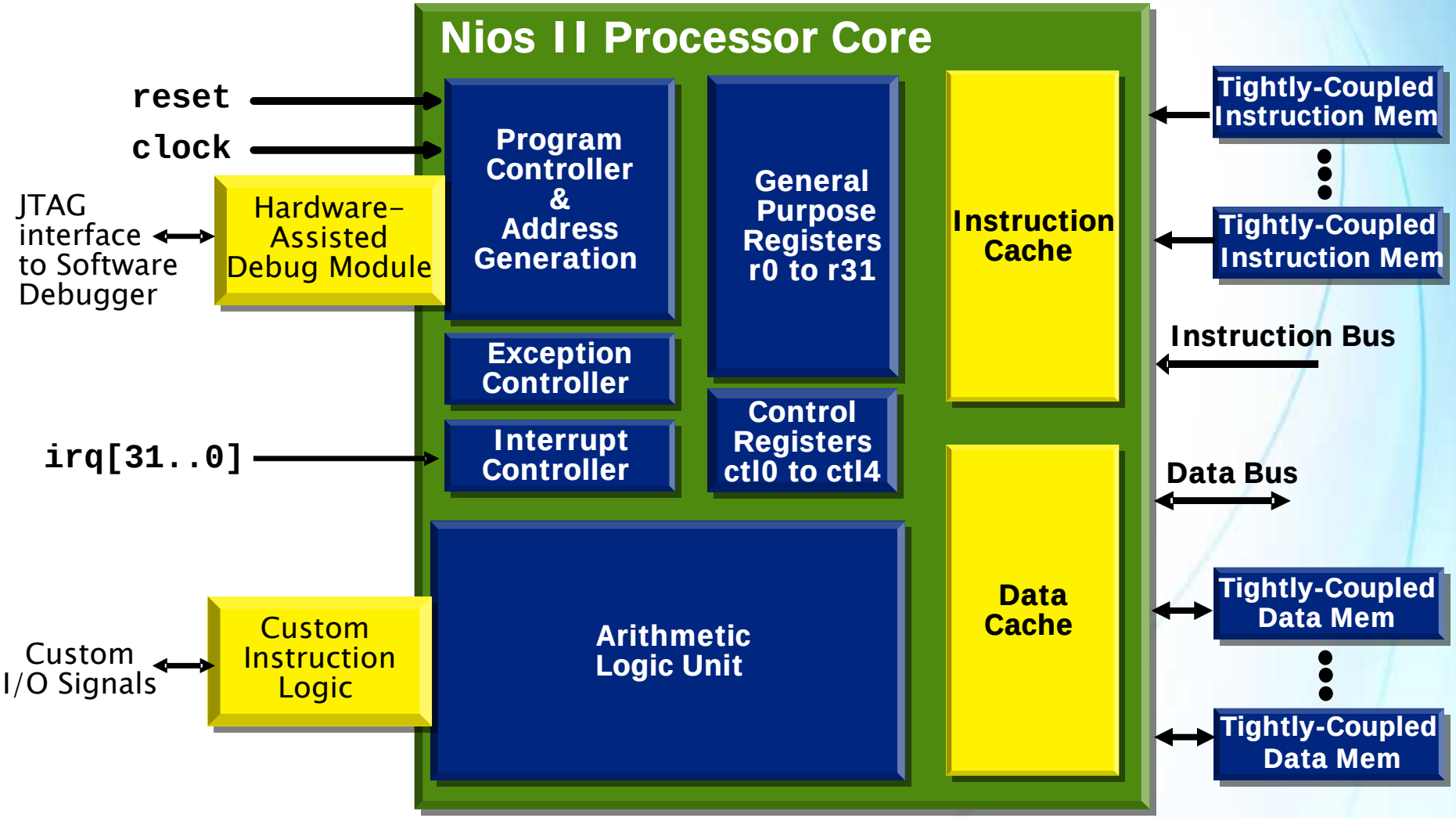


Avalon互联架构

Nios II处理器体系结构

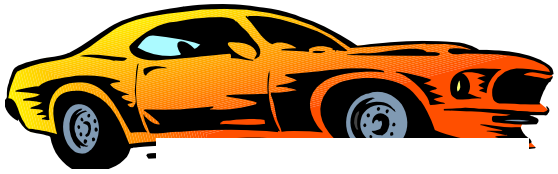
- 标准流水线RISC机
  - 32个通用寄存器
  - 3种指令格式
  - 32位指令通路
  - 32位数据通路
  - 平展的寄存器文件
  - 指令和数据高速缓冲分开(可配置容量)
  - 紧耦合存储器选项
  - 分支预测
  - 32个不同优先级的外部中断
  - 片内硬件(乘法、移位和循环)
  - 定制指令
  - 基于JTAG的硬件调试单元

Nios II处理器结构图



Nios II版本

- **Nios II处理器有三种ISA兼容版本**



– Fast：针对速度进行优化

– Standard：速度和面积之间平衡

– Economic：针对面积进行优化

- 软件
  - 代码兼容
    - CPU变化时，无需改变

二元兼容性 / 灵活的性能

	Nios II /f 快速	Nios II /s 标准	Nios II /e 经济
流水线	6级	5级	无
H/W乘法器和桶形移位寄存器	1周期	3周期	在软件中仿真
分支预测	动态	静态	无
指令高速缓冲	可配置	可配置	无
数据高速缓冲	可配置	无	无
TCM (Instr / Data)	达到： 4 / 4	达到： 4 / 0	0 / 0
逻辑占用（逻辑单元）	1400 - 1800	1200 - 1400	600 - 700
定制指令	达到256		

硬件乘法器加速

- Nios II经济型 – 没有乘法硬件
  - 使用GNUPro数学库来实现乘法运算
- Nios II标准型 – 完全的硬件乘法器
  - 32 x 32 ➔ 如果有DSP块，3个时钟周期后可以得到中32位的乘法结果，否则使用软件乘法
- Nios II快速型 – 完全的硬件乘法器
  - 32 x 32 ➔ 如果有DSP块， 1个时钟周期后可以得到中32位的乘法结果，否则使用软件乘法

加速硬件	时钟周期 (32 x 32 ➔ 32)
无	**250**
Stratix中的标准MUL	**3**
Stratix中的快速MUL	**1**

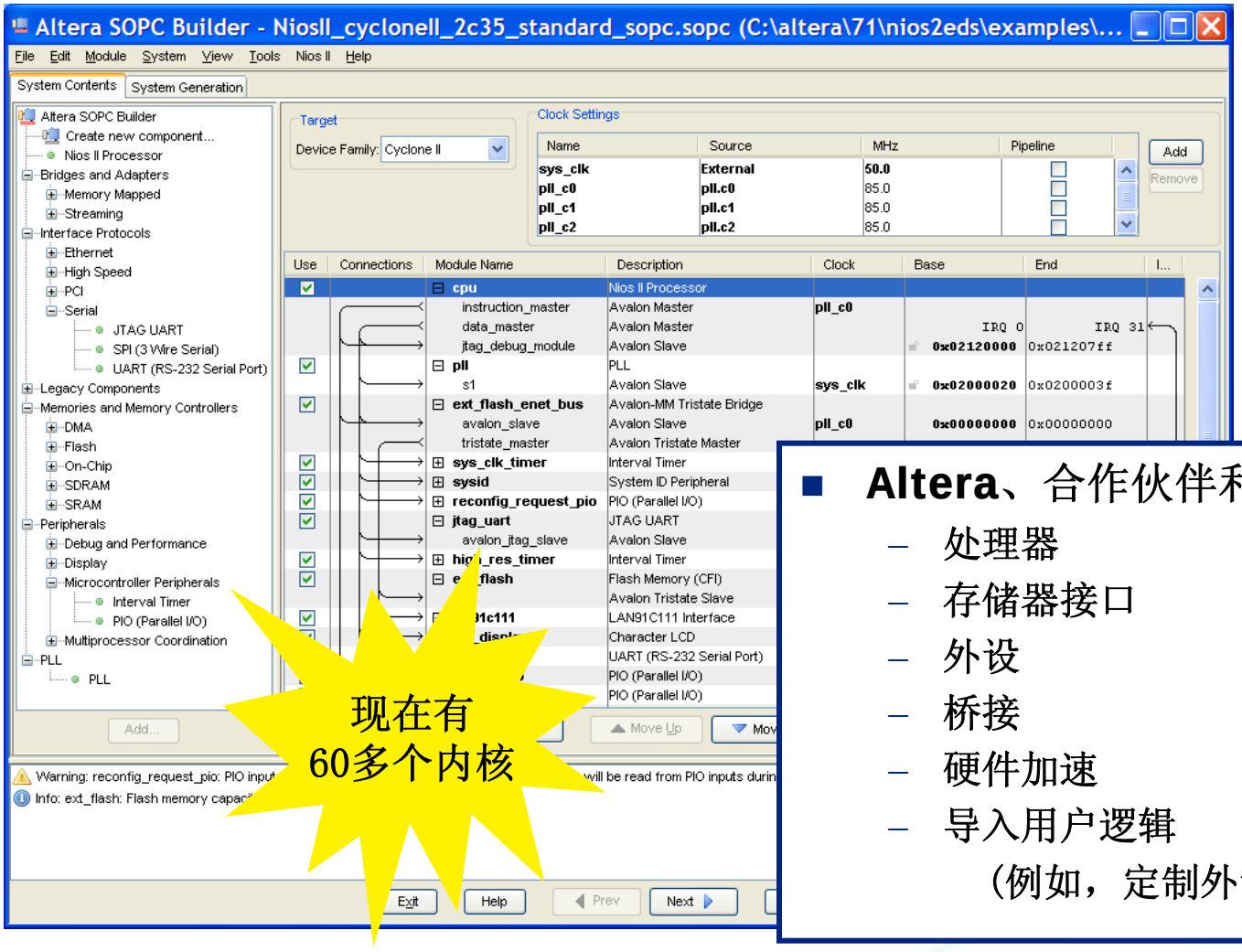
Nios II: 性能指标

	Nios II/f	Nios II/s	Nios II/e
Stratix III	**300 DMIPS@266 MHz** EP3SL70F4840C2	**128 DMIPS@200 MHz** EP3SL70F4840C2	**50 DMIPS@322 MHz** EP3SL70F4840C2
Stratix II	**251 DMIPS@222 MHz** EP2S60F1020C3	**110 DMIPS@171 MHz** EP2S60F1020C3	**44 DMIPS@285 MHz** EP2S60F1020C3
HardCopy Stratix II	**228 DMIPS@202 MHz** HC2300F1020C5	**129 DMIPS@202** HC2300F1020C5	**49 DMIPS@321 MHz** HC2300F1020C5
HardCopy Stratix	**166 DMIPS@147 MHz** EP1S80F1020C5_HC	**84 DMIPS@131** EP1S80F1020C5_HC	**27 DMIPS@176 MHz** EP1S80F1020C5_HC
Stratix	**168 DMIPS@148 MHz** EP1S80F1020C5	**82 DMIPS@128 MHz** EP1S80F1020C5	**27 DMIPS@172 MHz** EP1S80F1020C5
Cyclone III	**165 DMIPS@163MHz** EP2C20F484C6	**68 MHz@136 MHz** EP2C20F484C6	**17 DMIPS@190 MHz** EP2C20F484C6
Cyclone II	**144 DMIPS@142MHz** EP3C40F324C6	**55 MHz@111 MHz** EP3C40F324C6	**18 DMIPS@193 MHz** EP3C40F324C6
Cyclone	**130 DMIPS@134 MHz** EP1C20F400C6	**53 DMIPS@121 MHz** EP1C20F400C6	**17 DMIPS@173 MHz** EP1C20F400C6

* **FMax**是基于在片内存储器上运行得到的结果

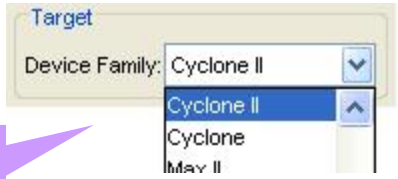
SOPC Builder 建立硬件系统过程

SOPC Builder – 系统内容页



- **Altera**、合作伙伴和用户内核
  - 处理器
  - 存储器接口
  - 外设
  - 桥接
  - 硬件加速
  - 导入用户逻辑
    （例如，定制外设）

系统内容页特性



目标系列

时钟域

地址映射

Use	Connections	Module Name	Description	Clock	Base	End	I...
☑		⊞ pll	PLL	sys_clk	0x01000020	0x0 03f	
☑		⊟ cpu	Nios II Processor				
		instruction_master	Avalon Master	pll_c0			
		tightly_coupled_instruction_master_0	Avalon Master				
		data_master	Avalon Master		IRQ 0	IRQ 31	
		tightly_coupled_data_master_0	Avalon Master				
		jtag_debug_module	Avalon Slave		0x02120000	0x021207ff	
☑		⊟ ext_flash_enet_bus	Avalon-MM Tristate Bridge				
		avalon_slave	Avalon Slave	pll_c0	0x00000000	0x00000000	
		tristate_master	Avalon Tristate Master				
☑		⊞ sys_clk_timer	Interval Timer	pll_c0	0x02120800	0x0212081f	0
☑		⊞ sysid	System ID Peripheral	pll_c0	0x021208b8	0x021208bf	
☑		⊞ reconfig_request_pio	PIO (Parallel I/O)	pll_c0	0x021208a0	0x021208af	
☑		⊞ jtag_uart	JTAG UART	pll_c0	0x021208b0	0x021208b7	1
☑		⊞ high_res_timer	Interval Timer	pll_c0	0x02120820	0x0212083f	3
☑		⊞ ext_flash	Flash Memory (CFI)	pll_c0	0x00000000	0x00ffffff	
☑		⊞ lan91c111	LAN91C111 Interface	pll_c0	0x02110000	0x0211ffff	6
		⊞ lcd_display	Character LCD	pll_c0	0x02120880	0x0212088f	
☑		⊞ uart1	UART (RS-232 Serial Port)	pll_c0	0x02120840	0x0212085f	4
☑		⊞ button_pio	PIO (Parallel I/O)	pll_c0	0x02120860	0x0212086f	2
☑		⊞ led_pio	PIO (Parallel I/O)	pll_c0	0x02120870	0x0212087f	
☑		⊞ seven_seg_pio	PIO (Parallel I/O)	pll_c0	0x02120890	0x0212089f	
☑		⊞ tightly_coupled_instruction_memory	On-Chip Memory (RAM or ROM)	*multiple*	*multiple*	*multiple*	
☑		⊞ tightly_coupled_data_memory	On-Chip Memory (RAM or ROM)	*multiple*	*multiple*	*multiple*	
☑		⊞ performance_counter	Performance Counter Unit	pll_c0	0x02120900	0x0212093f	
☑		⊞ ext_ssram_bus	Avalon-MM Tristate Bridge	pll_c0	0x00000000	0x00000000	
☑		⊞ ext_ssram	Cypress CY7C1380C SSRAM	pll_c0	0x20000000	0x201fffff	
☑		⊞ epcs_controller	EPCS Serial Flash Controller	pll_c0	0x03200000	0x032007ff	5
☑		⊞ ddr_sdram_0	DDR SDRAM Controller MegaCore Fun...	pll_c0	0x30000000	0x3fffffff	
☑		⊞ dma	DMA Controller	pll_c0	0x02120a00	0x02120a1f	7
☑		⊞ pllsysx2	PLL	pllsysx2_cl...	0x01000040	0x0100005f	

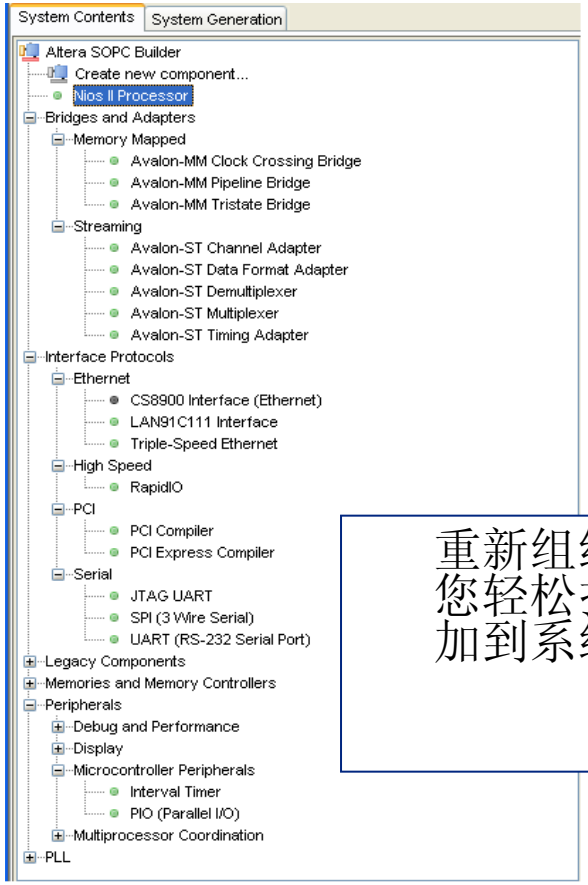
组件

连接面板

IRQ优先级

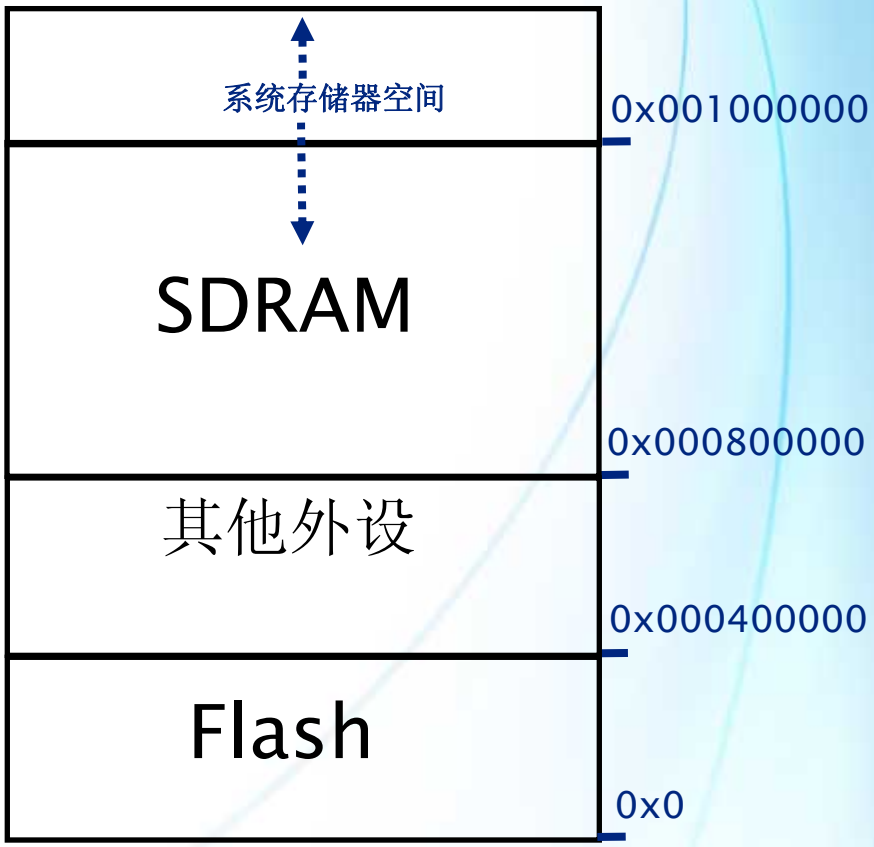
添加外设，包括Nios II处理器

- 双击外设，或者按下Add...
  - 为您的嵌入式系统构建存储器映射

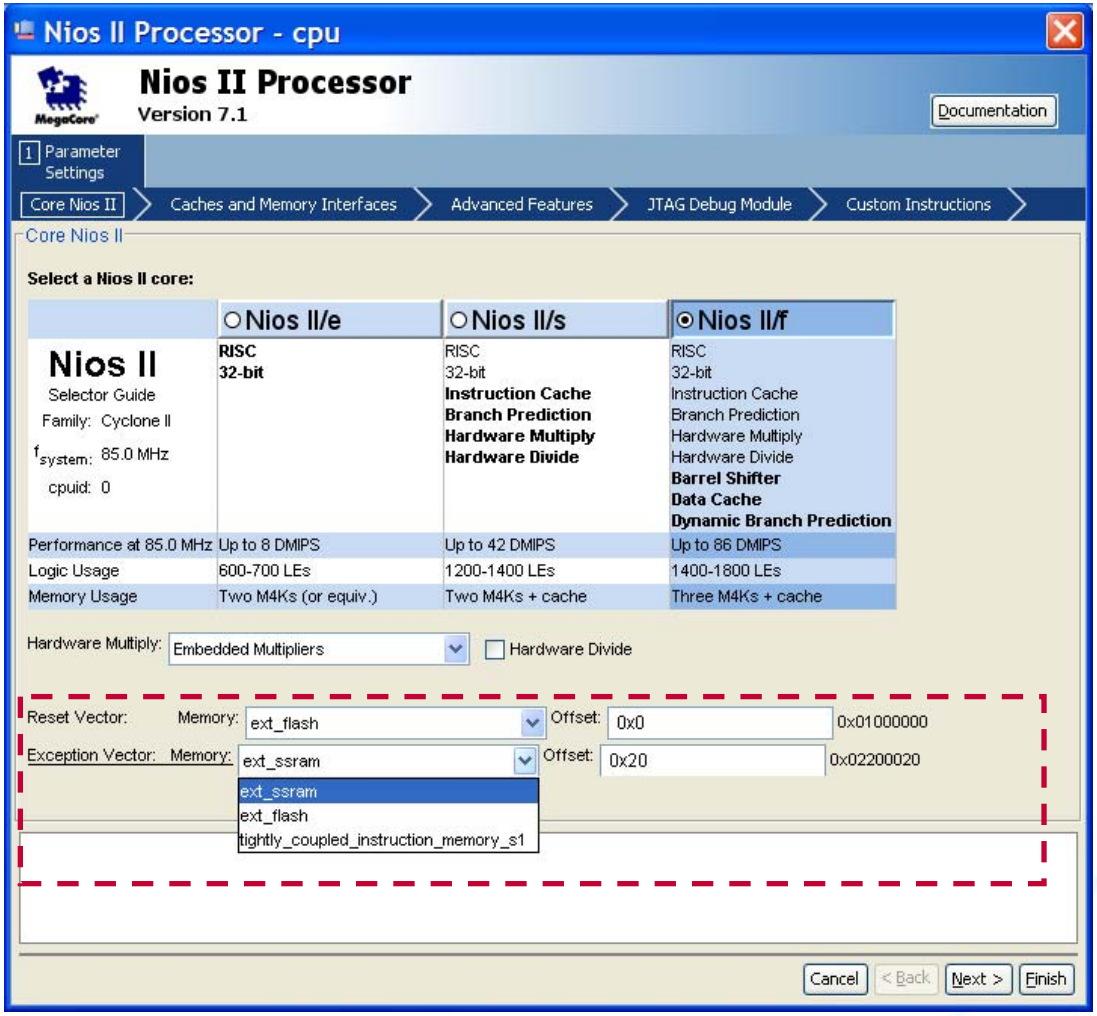


重新组织，以帮助您轻松找到组件，加到系统中

例如



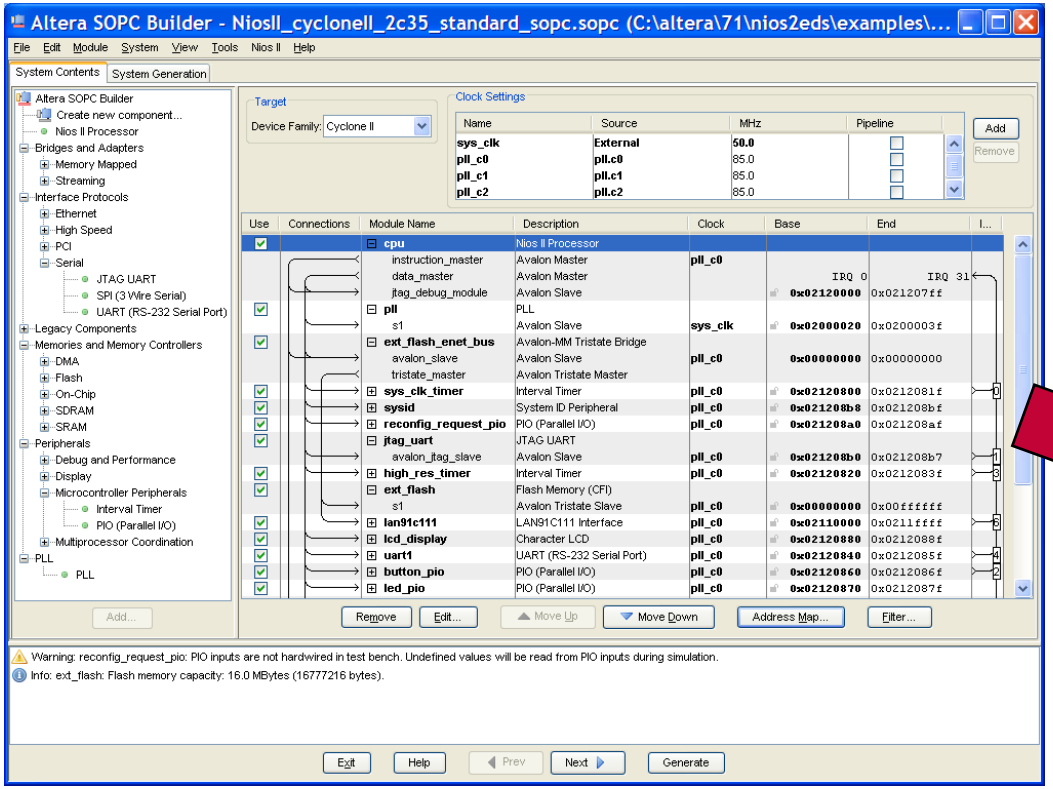
设置复位和异常地址

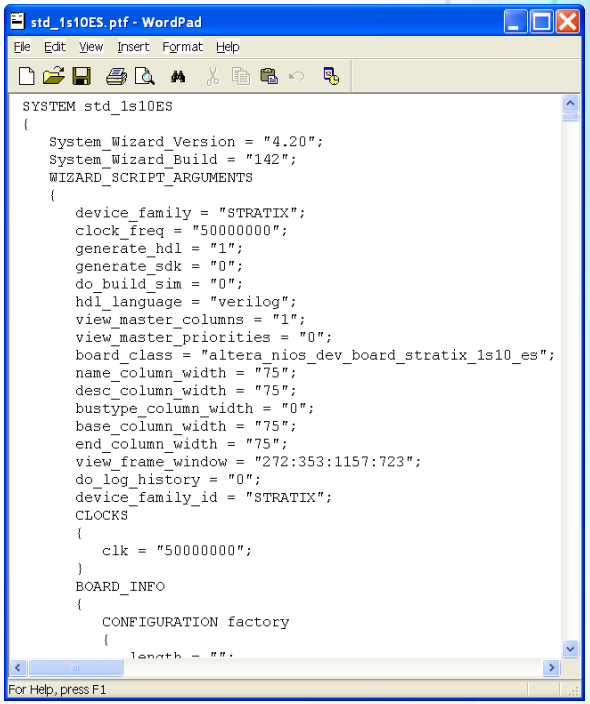


为了选择**复位**或者异常地址，系统中必须添加了相应的存储器。

SOPC Builder生成其他文本文件

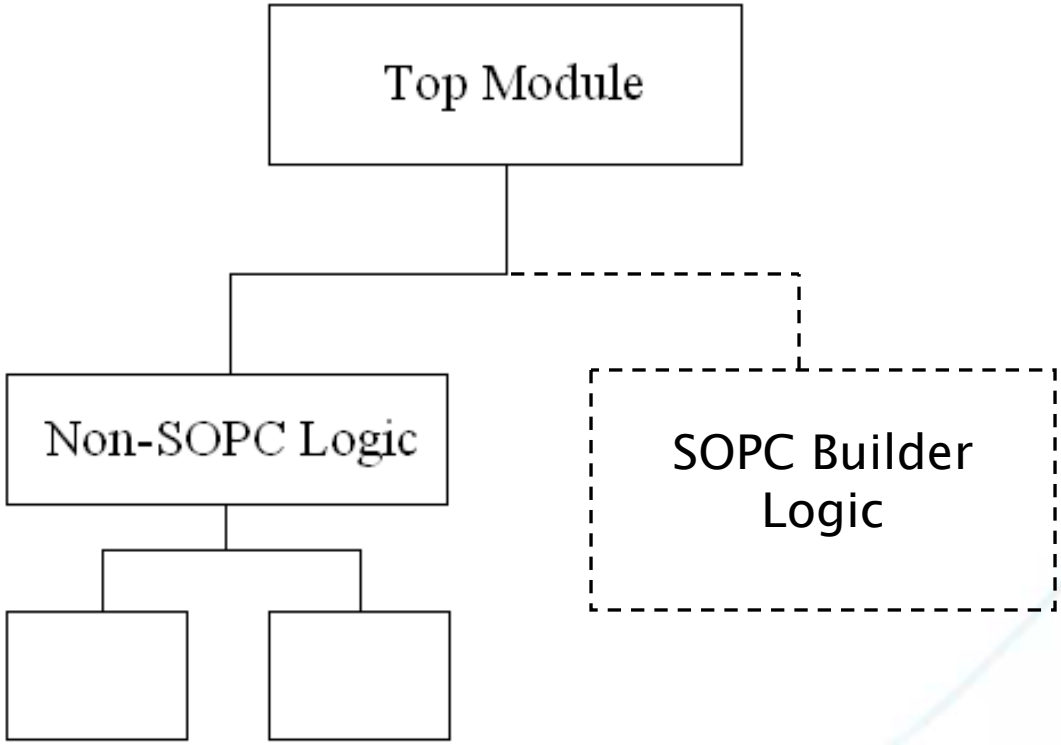
- .SOPC文件
  - 文本文件，记录SOPC Builder编辑并描述Nios II系统
- .PTF文件
  - 供SW开发工具使用





集成SOPC Builder子系统

- 使用HDL代码或者原理图输入工具，进入Quartus II顶层设计



Verilog例化

- 在“system”HDL文件中 →

```
// Adapted from low-cost reference design:

module  top_level (
                    // inputs:
                      in_port_to_the_button_pio,
                      reset_n,
                      sys_clk,

                      // outputs:
                      clk_to_sdram,
                      clk_to_sdram_n,
                      ddr_a,
                              ⋮
                    ddr_ras_n,
                      ddr_we_n,
                      out_port_from_the_led_pio,
                      out_port_from_the_seven_seg_pio,
                      pll_c0_out,
                      pll_c1_out
);


// Port Declarations …
                              ⋮

// Wire Declarations …
                              ⋮
```

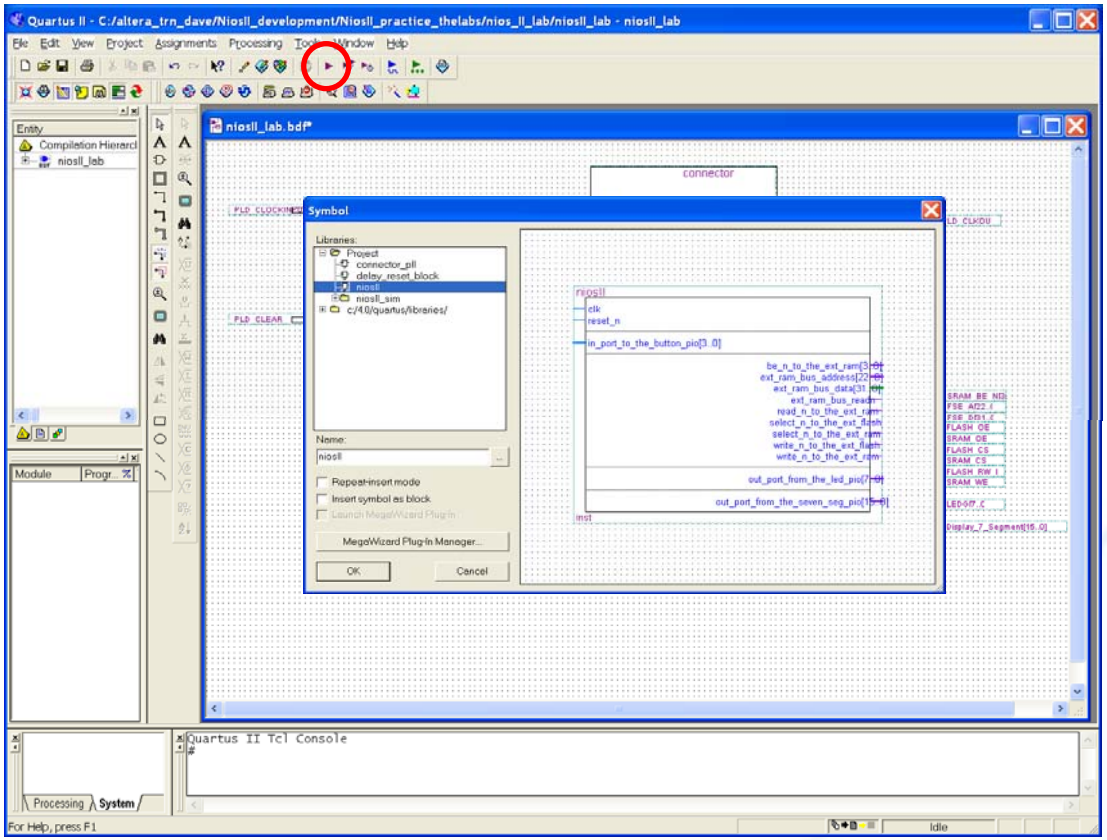
```
SOPC_system SOPC_system_inst
  (
    .clk_to_sdram_from_the_ddr_sdram_0 (single_bit_clk_to_sdram),
    .clk_to_sdram_n_from_the_ddr_sdram_0
      (single_bit_clk_to_sdram_n),
    .ddr_a_from_the_ddr_sdram_0 (ddr_a),
    .ddr_ba_from_the_ddr_sdram_0 (ddr_ba),
    .ddr_cas_n_from_the_ddr_sdram_0 (ddr_cas_n),
    .ddr_cke_from_the_ddr_sdram_0 (single_bit_ddr_cke),
    .ddr_cs_n_from_the_ddr_sdram_0 (single_bit_ddr_cs_n),
    .ddr_dm_from_the_ddr_sdram_0 (ddr_dm),
    .ddr_dq_to_and_from_the_ddr_sdram_0 (ddr_dq),
    .ddr_dqs_to_and_from_the_ddr_sdram_0 (ddr_dqs),
    .ddr_ras_n_from_the_ddr_sdram_0 (ddr_ras_n),
    .ddr_we_n_from_the_ddr_sdram_0 (ddr_we_n),
    .in_port_to_the_button_pio (in_port_to_the_button_pio),
    .out_port_from_the_led_pio (out_port_from_the_led_pio),
    .out_port_from_the_seven_seg_pio
      (out_port_from_the_seven_seg_pio),
    .pll_c0_out (pll_c0_out),
    .pll_c1_out (pll_c1_out),
    .pll_c2_out (pll_c2_out),
    .reset_n (reset_n),
    .sys_clk (sys_clk),
    .write_clk_to_the_ddr_sdram_0 (write_clk_to_the_ddr_sdram_0)
  );

// Local assignments:
                              ⋮

endmodule
```

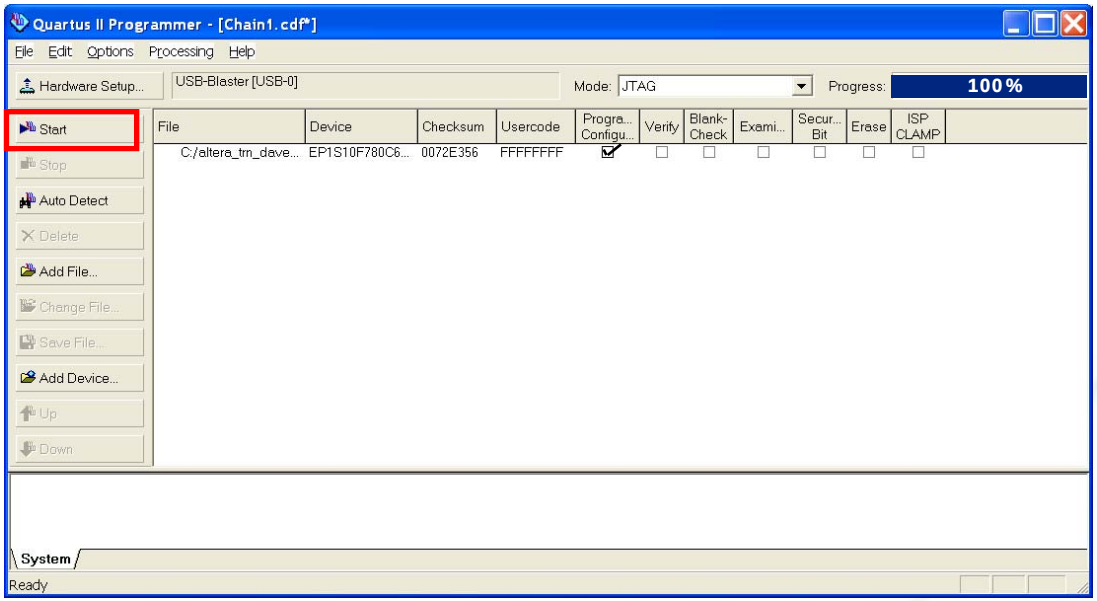
在结构图设计文件中例化

- 如下所示，选择组件
- 然后在Quartus II中编译设计 



对目标硬件编程

- 配置FPGA
  - 含有Nios II和外设逻辑，以及FPGA中的其他顶层逻辑

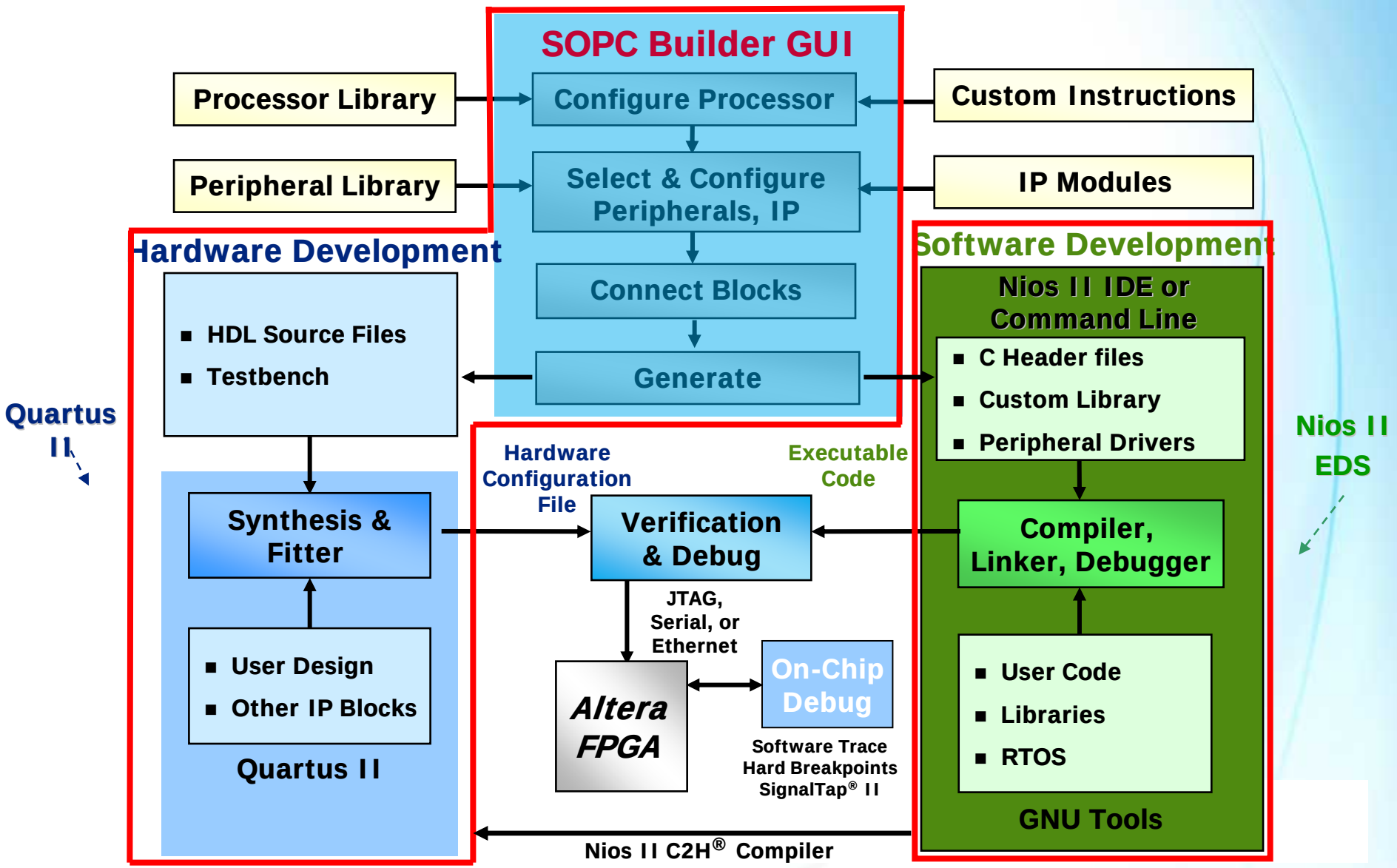


Quartus II演示

- 查看文件
- 集成顶层模块
- 编译设计
- 对开发板编程
- 链接

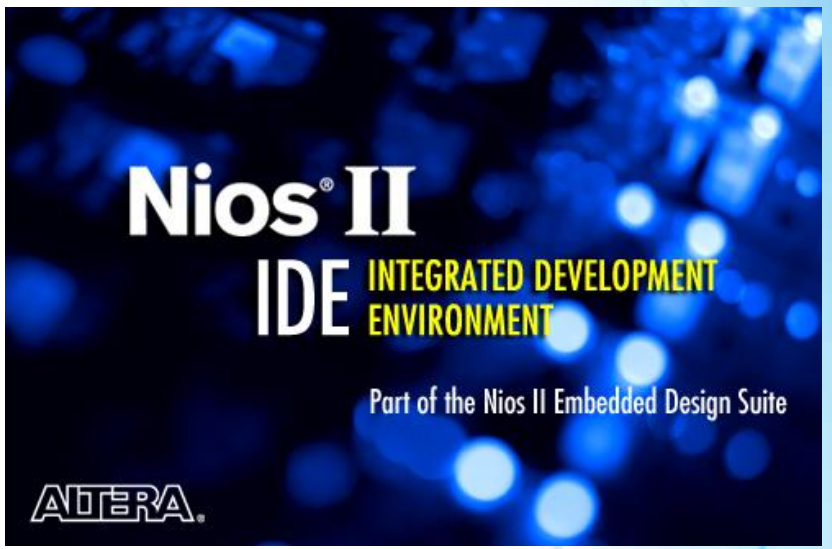
Nios II软件开发

Nios II系统设计流程



Nios II IDE（集成开发环境）*

- Nios II EDS中的前沿软件开发工具
- 目标连接
  - 硬件（JTAG）
  - 指令集仿真器
  - ModelSim®-Altera软件
- 高级调试特性
  - 软件断点
  - 硬件断点
  - 数据触发
  - 指令流记录
- Flash和Quartus II编程支持



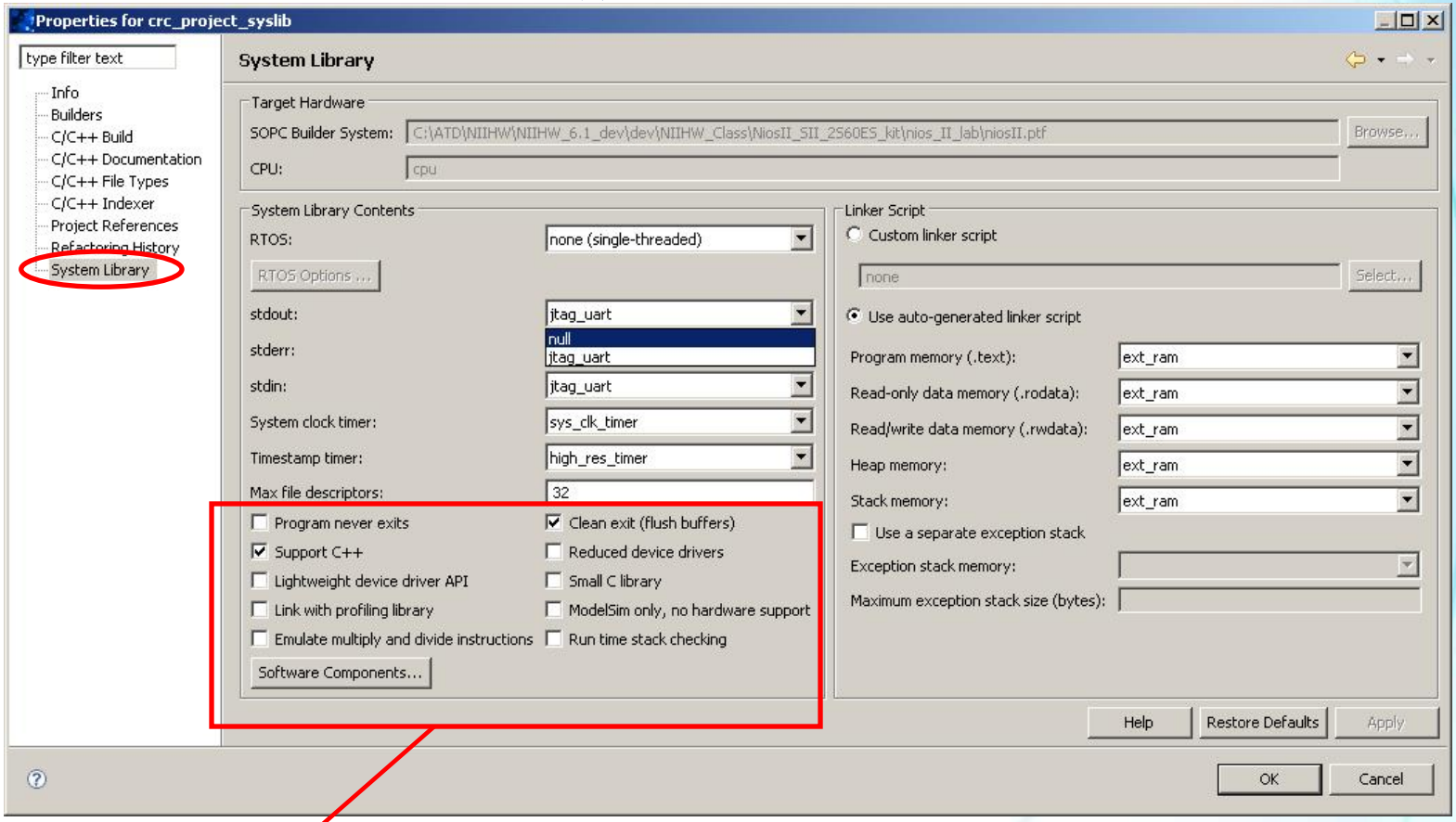
* 基于Eclipse 3.2/CDT 3.1

Nios II IDE 演示1

- 启动Nios II IDE
- 建立新的软件工程
- 设置工程属性
- 编译软件工程

系统库工程特性

指定**stdio**器件



其他选项，包括减少代码

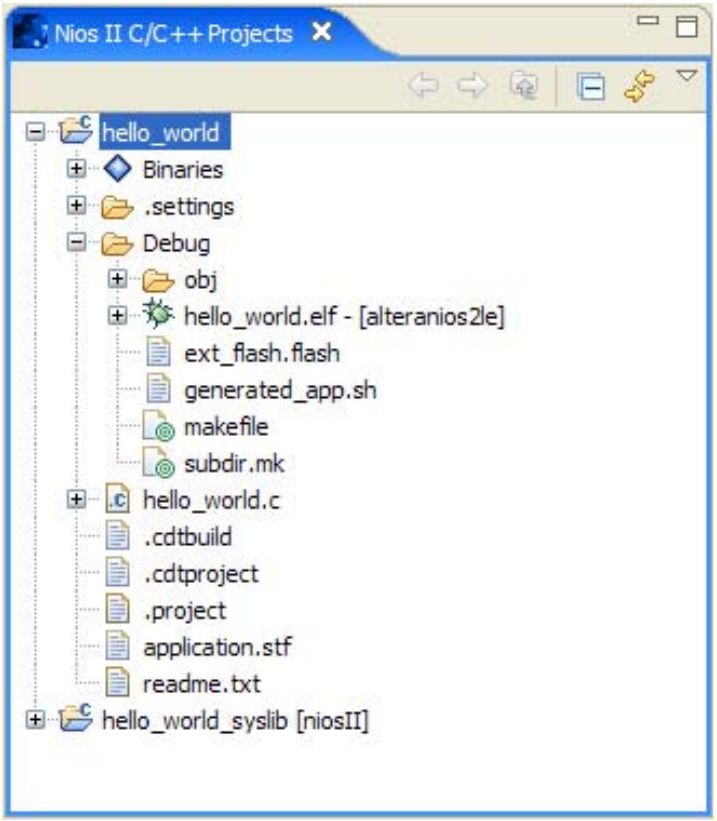
划分存储器

这就是重要的系统库属性页，建议用户在第一次编译软件之前，都要检查一下该页的内容。

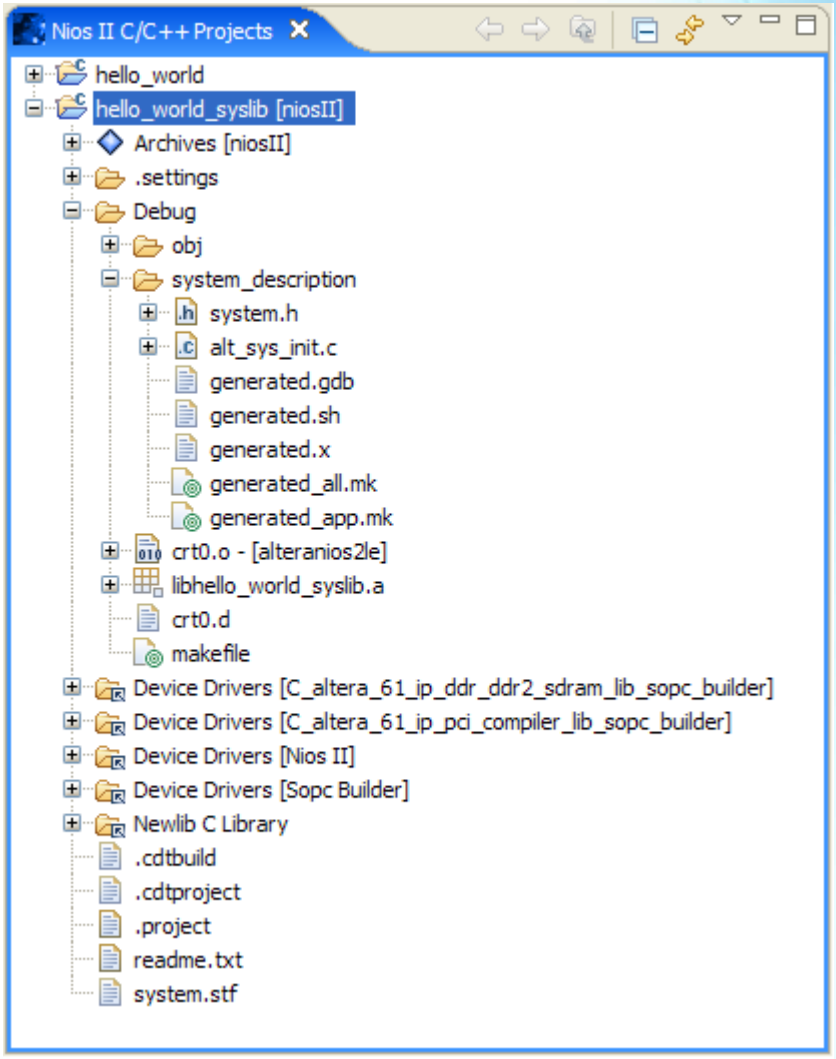
- 该页的主要工作，一是为软件映象的不同段，如text，rodata，rwdata，堆和栈等等，划分真正的物理存储器，也就是指定软件的加载地址。
- 二是为系统指定标准输入输出或系统时钟等设备，
- 其他选项，如small C library， reduced device driver等等，用于减少代码尺寸。

编译后的目录结构

- 应用工程



- 系统库工程

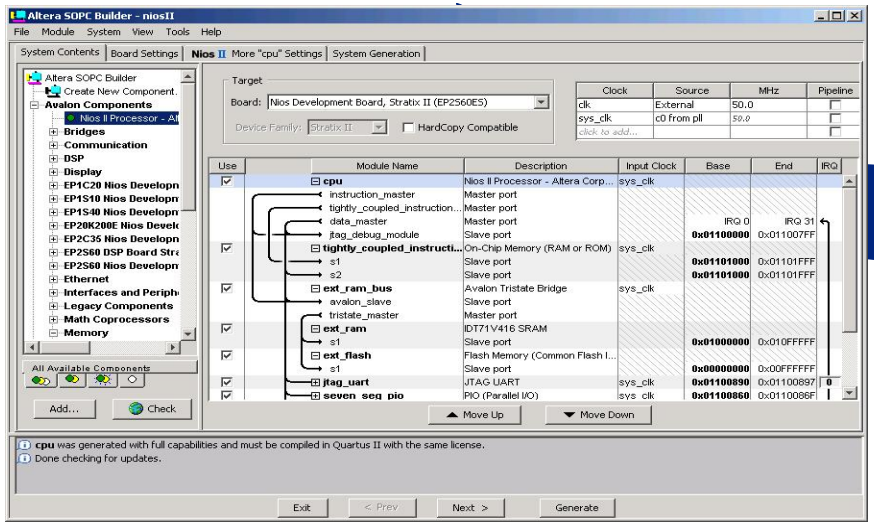


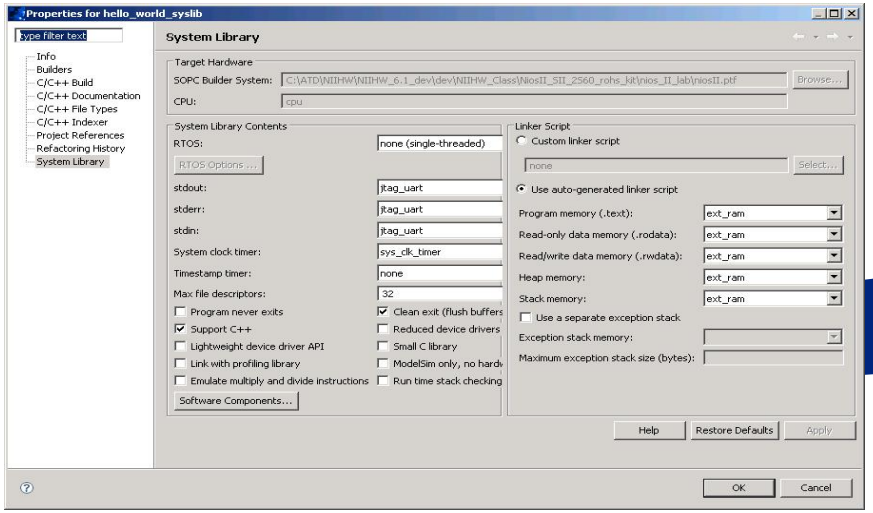
在编译结束之后，请关注一下编译后生成的主要文件。

- 一是应用工程目录下，生成的elf文件，它是软件工程的目标执行文件；flash文件，用于编程flash器件的文件；makefile，生成的软件编译make脚本
- 二是在系统库工程目录的system_descripiton子目录下，system.h文件，它是系统头文件；alt_sys_init.c文件，是生成的设备初始化文件；generated.x文件，软件的链接脚本。

系统头文件

SOPC Builder 系统内





system.h

系统库设置

- 系统头文件system.h是一个重要文件，它主要由两个部分的内容来生成.
- 一是SOPCBuilder中的所有硬件信息，例如，都有哪些外设，外设的配置如何，外设名称，外设的存储器映象等等。
- 二是在NiosII IDE中，系统库属性页中的所有配置信息，例如是否指定了标准输入输出设备，是否支持MicroOS操作系统，是否使用了软件构件等。

system.h

- 含有系统参数的宏定义，包括外设配置，例如：
  - 外设硬件配置
  - 基地址
  - IRQ优先级（如果需要）
  - 外设符号名称
- 位于syslib工程目录中
- 当以名称调用外设时，只需要将其包含在您的应用代码中。

system.h – 例子

- 定义系统设置和外设配置

```
/*
 * system configuration
 *
 */
#define ALT_SYSTEM_NAME "niosII"
#define ALT_CPU_NAME "cpu"
#define ALT_CPU_ARCHITECTURE "altera_nios2"
#define ALT_DEVICE_FAMILY "STRATIXII"
#define ALTERA_NIOS_DEV_BOARD_STRATIX_2S60_ES
#define ALT_STDIN "/dev/jtag_uart"
#define ALT_STDIN_TYPE "altera_avalon_jtag_uart"
#define ALT_STDIN_BASE 0x01100890
#define ALT_STDIN_DEV jtag_uart
#define ALT_STDIN_PRESENT
#define ALT_STDOUT "/dev/jtag_uart"
#define ALT_STDOUT_TYPE "altera_avalon_jtag_uart"
#define ALT_STDOUT_BASE 0x01100890
#define ALT_STDOUT_DEV jtag_uart
#define ALT_STDOUT_PRESENT
#define ALT_STDERR "/dev/jtag_uart"
#define ALT_STDERR_TYPE "altera_avalon_jtag_uart"
#define ALT_STDERR_BASE 0x01100890
#define ALT_STDERR_DEV jtag_uart
#define ALT_STDERR_PRESENT
#define ALT_CPU_FREQ 50000000
#define ALT_IRQ_BASE NULL
                    .
                    .
                    .
```

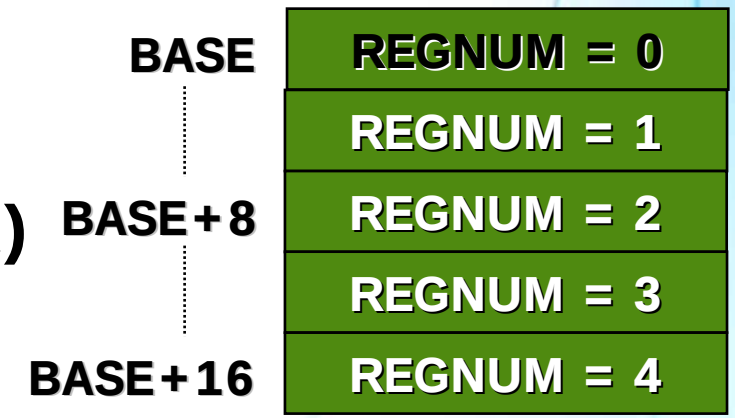
```
                    .
                    .
                    .
/*
 * button_pio configuration
 *
 */
#define BUTTON_PIO_NAME "/dev/button_pio"
#define BUTTON_PIO_TYPE "altera_avalon_pio"
#define BUTTON_PIO_BASE 0x01100880
#define BUTTON_PIO_SPAN 16
#define BUTTON_PIO_DO_TEST_BENCH_WIRING 0
#define BUTTON_PIO_DRIVEN_SIM_VALUE 0x0000
#define BUTTON_PIO_HAS_TRI 0
#define BUTTON_PIO_HAS_OUT 0
#define BUTTON_PIO_HAS_IN 1
#define BUTTON_PIO_CAPTURE 0
#define BUTTON_PIO_EDGE_TYPE "NONE"
#define BUTTON_PIO_IRQ_TYPE "NONE"
#define BUTTON_PIO_FREQ 50000000
#define ALT_MODULE_CLASS_button_pio altera_avalon_pio
                    .
                    .
                    .
```

在Nios II中读/写硬件

- 用于访问硬件的I/O宏
  - 为进行硬件访问，I/O宏旁路高速缓冲
  - 它们使用STxIO或者LDxIO指令

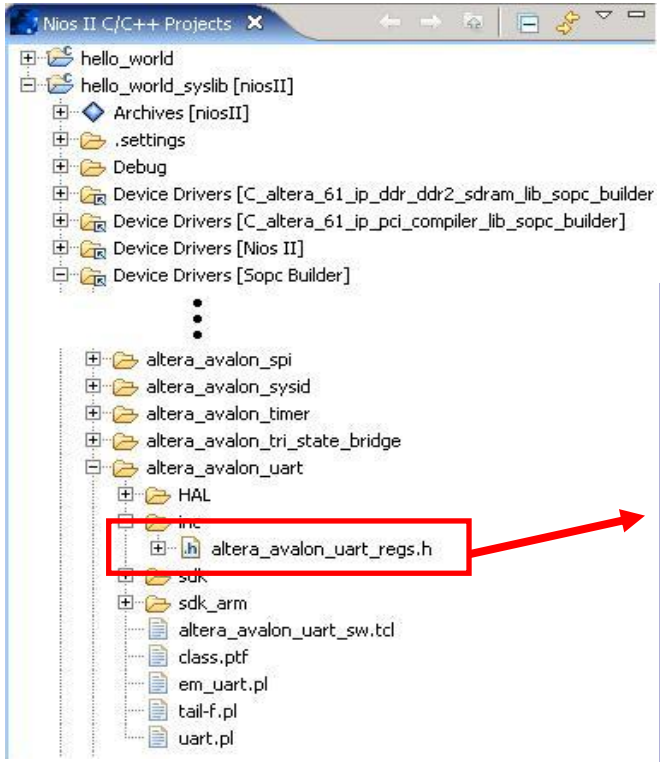
  - **IORD(BASE, REGNUM)**
    - 读取从基本地址BASE偏移REGNUM的寄存器值

  - **IOWR(BASE,REGNUM,DATA)**
    - 向从基本地址BASE偏移REGNUM的寄存器写入数据



Nios II外设头文件

- 每个Nios II外设针对每一寄存器提供特定读/写宏
  - 例子： UART（“altera_avalon_uart_regs.h”）




```
#define IORD_ALTERA_AVALON_UART_RXDATA(base)       IORD(base, 0)
#define IOWR_ALTERA_AVALON_UART_RXDATA(base, data) IOWR(base, 0, data)

#define IORD_ALTERA_AVALON_UART_TXDATA(base)       IORD(base, 1)
#define IOWR_ALTERA_AVALON_UART_TXDATA(base, data) IOWR(base, 1, data)

#define IORD_ALTERA_AVALON_UART_STATUS(base)       IORD(base, 2)
#define IOWR_ALTERA_AVALON_UART_STATUS(base, data) IOWR(base, 2, data)
```

软件运行和调试