第5章 SOPC技术

5.1 SOPC 硬件开发环境及硬件开发流程

自 20 世纪 90 年代末以来，电子系统的设计方式发生了巨大改变。其中，基于模块的芯片设计成为电子系统设计的主流。Altera 公司也提出了基于 FPGA 的片上系统解决方案——SOPC 技术。该技术利用了计算机辅助设计技术，以嵌入式技术为核心，集软、硬件为一体，最大限度地优化系统，符合电子技术的发展趋势。本章主要介绍 SOPC 的相关技术。

Altera 公司的 SOPC 简单地说就是在可编程逻辑器件的基础上实现一个以 CPU 为核心的智能控制系统，是 Altera 公司提出的一种灵活、高效的 SOC 解决方案。在一个 SOPC 设计中，将所用到的微处理器、DSP 芯片、存储器件、I/O、控制逻辑、混合信号模块等集成到 FPGA 器件上，构建成一个可编程片上系统。可编程系统具有灵活的设计方式，可裁剪、可扩充、可升级，且具备软、硬件在系统可编程的功能。

FPGA 内部含有小容量的高速 RAM 资源。利用可供灵活选择的 IP 核资源，用户可以构成各种系统，如单处理器、多处理器系统。除了系统使用的资源外，还可以利用足够的可编程逻辑资源实现其他的附加逻辑。

SOPC 是 PLD 和 ASIC 技术融合的结果。它是一种特殊的嵌入式系统，具备软、硬件系统可编程的功能。近年来，FPGA 无论在逻辑门密度还是在运行频率等方面都取得了长足进步，已经可以把处理器软核、ASIC 硬核、数字信号处理器件及网络控制等各种数字逻辑控制器，以 IP 核的形式集成到 FPGA 芯片中，构成嵌入式系统。采用这种设计方式，能够使系统具有开发周期短和系统可修改等优点。因此，基于 FPGA 的嵌入式系统成为 SOPC 的热点。

2000 年，Altera 公司发布了 Nios 处理器，这是业界第一款可编程逻辑优化的可配置的软核处理器。它基于 RISC 技术，具有 16 位指令集、16 位/32 位数据通道和 5 级流水线，在一个时钟周期内完成一条指令的处理。Altera 公司把可编程逻辑的优势集成到嵌入处理器的开发流程中，设计者定义了处理器之后，再把 CPU 周边的专用硬件逻辑集成进去，构成可定制的 SOPC，随后就可以开展软件原型设计。

在 Nios Ⅱ系统设计的每个阶段，软件都能够进行测试，解决遇到的问题。另外，软件组可以对结构方面提出一些建议，改善代码效率和处理器性能，软件和硬件权衡可以在硬件设计过程中间完成。

在 Altera 公司的 Nios 嵌入式处理器中，设计者能够在 Nios 指令系统中增加自定义指令，以增强对实时软件算法的处理能力。用户自定义指令可以完成复杂的处理任务。另外，增加的用户自定义指令也可以访问存储器或 Nios 系统外的逻辑。设计者可以在 Avalon 互连架构中加入定制外设，这一特性可以用于数字信号处理、数据包处理及计算密集的地方。

传统的 SOC 设计方法需要用户把处理器以及外设手动连接起来，还需要用户手动去分配地址空间资源，这样既耗时又容易出错。针对这种情况，Altera 公司开发了一种

智能的工具，帮助用户方便快捷地产生一个 SOPC，这个工具就是 SOPC Builder。SOPC Builder 是内嵌在 Quartus Ⅱ设计工具中的，用户可以非常方便地用 SOPC Builder 产生完一个系统后，在 Quartus Ⅱ中对其进行编译，并实现在目标器件中。SOPC Builder 开发工具具有直观的图形用户接口，便于设计者准确地添加和配置系统所需的外设，包括存储器，定制外设和 IP 模块。

Altera 公司还提供了软件开发工具，该开发工具支持 C/C++语言，并提供了常用的功能类库。开发者可以直接使用 C/C++语言进行系统软件开发，然后在线调试自行设计的 Nios 处理器和软件。当软件达到设计要求时，可通过该工具将执行代码下载到 Flash 或 FPGA 中，使所设计的系统独立运行。

SOPC Builder 最大的好处是使系统设计过程自动化，这也是 EDA 业界追求的目标。在 SOPC Builder 中，设计者只需要选择自己需要的处理器和外设类型，工具将自动根据 Avalon 总线的标准产生一些互连逻辑，将各个模块连接起来。这些互连逻辑功能包括数据通道复用、等待状态产生、中断控制和数据宽度匹配。同时，工具也可以自动分配外设的地址空间。用户也可以根据需要对连接关系进行调整，或者手动指定外设地址空间。

SOPC Builder 的输出文件包括定义所有模块的 HDL 描述，还有一个顶层的 HDL 描述文件，用来把所有的模块集成在一起。另外，在使用 SOPC Builder 定制系统的同时，所有的设置都放在扩展名为“.ptf”的文件中，这个文件可以作为归档文件。用户也可以直接修改这个文件，不过一般不建议用户这样做。

1. 定制界面

启动 SOPC Builder 即可在 Quartus Ⅱ的菜单栏中选择 Tools|SOPC Builder 命令，然后指定需要产生的系统名称和 HDL 原语类型，就可以进入到系统定制界面中，如图 5-1 所示。

图 5-1 的左边是可供用户选择的模块资源池，包括处理器（Nios 和 Nios Ⅱ）和各种外设。这些模块的数量一直在增加，有 Altera 公司自己开发的模块，也包括第三方开发的模块。用户逻辑也可以加到资源池中，用户逻辑在第一次被定义之后，就可以将其放到模块资源池中，以后就将其当作普通模块一样使用。

图 5-1 右边的上方，需要用户选择目标单板、目标器件和系统时钟频率，右下方是用户自己增加的系统模块列表。图中的行列线以及交叉点指明了各个模块之间的连接关系，用户可以修改其连接关系。SOPC Builder 工具自动给这些模块分配了地址空间，确保不会冲突。用户也可以手动分配地址空间。

用户可以在图 5-1 的 More "cpu_0" Settings 中设置 CPU 的复位初始地址和异常处理地址。

需要注意的是，SOPC Builder 不仅可以作 Nios 处理器系统，同样可以生成没有处理器的系统，Avalon 总线只在各个模块之间起到互连的作用。

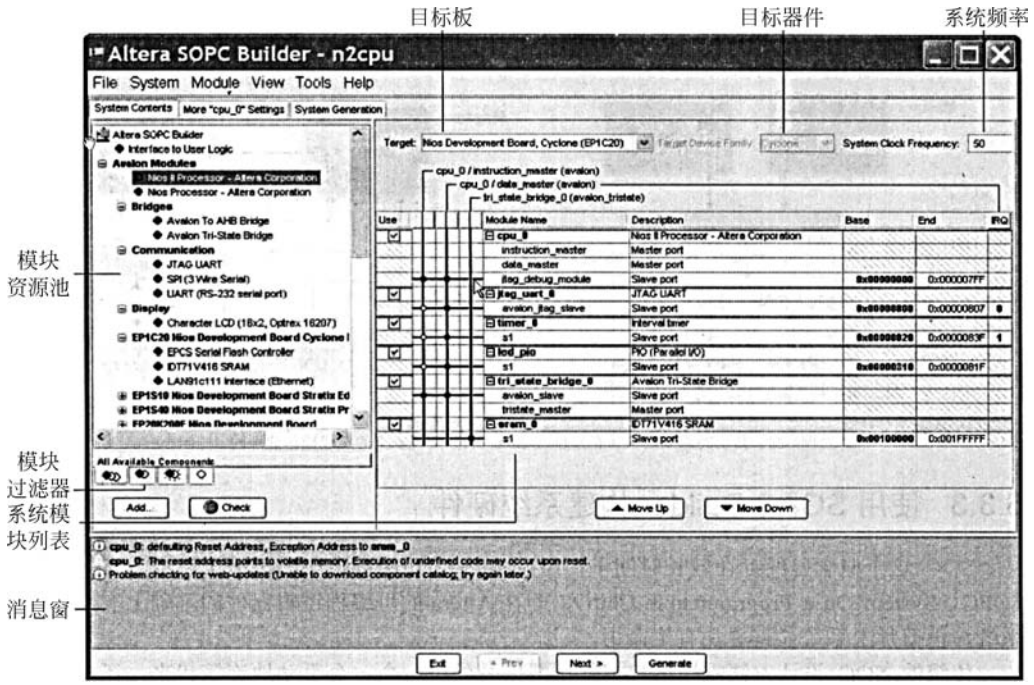


图 5-1　SOPC Builder 定制界面

2. 生成界面

用户定制完系统后，需要将其编译形成模块文件，用作功能仿真，系统生成界面如图 5-2 所示。

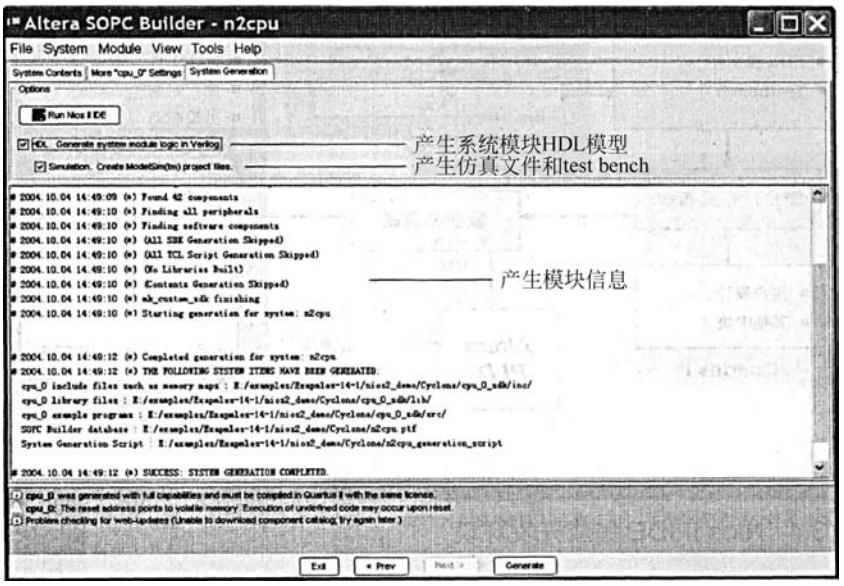


图 5-2　SOPC Builder 生成编译文件界面

单击 Generate 按钮，SOPC Builder 自动产生所需要的文件，并放在工程的根目录下面。

3. 系统开发流程

前面已经讨论了如何在 SOPC 中定制用户系统，并如何生成系统。那么 SOPC Builder 如何融合到整个系统的开发流程中？

图 5-3 为 SOPC Builder 开发全流程。

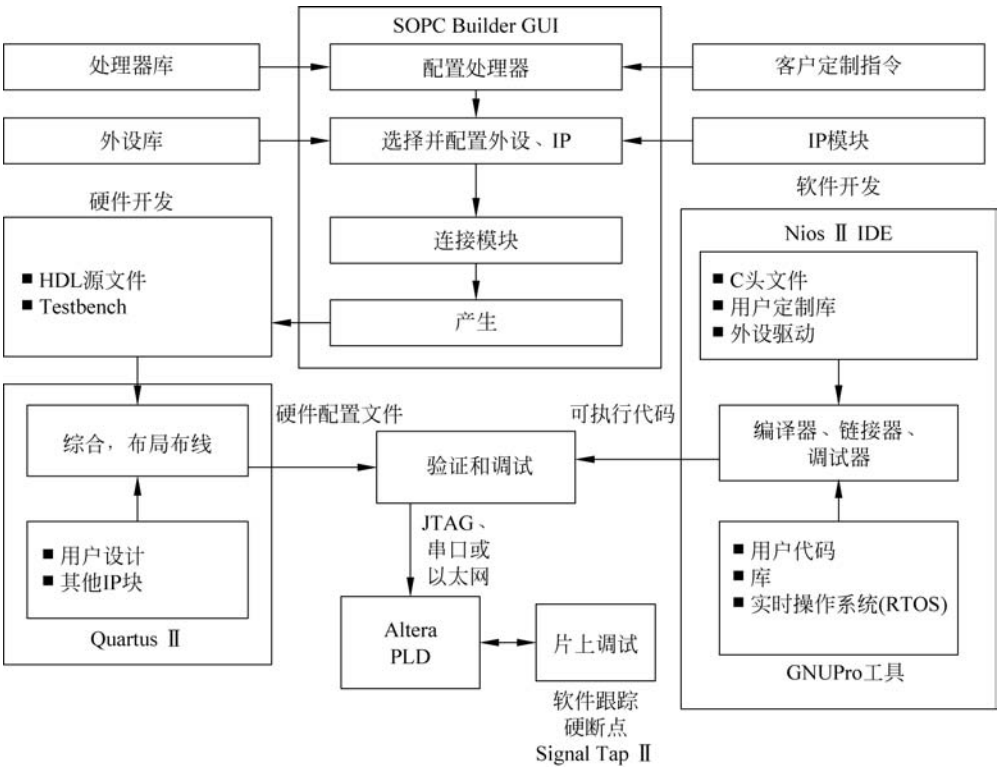


图 5-3 SOPC Builder 开发流程

用户首先利用 SOPC Builder 的图形界面定制系统，产生输出文件，然后进入传统的硬件开发流程，在 Quartus Ⅱ 中进行逻辑综合、布局布线。在软件开发流程中，用户可以利用 Nios Ⅱ IDE 环境，建立工程、编译设计和调试等。

5.2 Nios Ⅱ IDE 集成开发环境

前面简单介绍了使用 SOPC Builder 构建 Nios Ⅱ 系统的基本流程，而要开发基于 Nios Ⅱ 系统的应用程序，可以使用 Altera 公司为 Nios Ⅱ 系统定制的 Nios Ⅱ IDE 系统。

1. 集成开发环境

Nios Ⅱ IDE是基于Eclipse IDE的集成开发环境，已经被许多软件工程师熟悉。用户可以在Nios Ⅱ IDE中为Nios Ⅱ系统开发模块驱动程序、板级支持包(BSP)以及用户应用程序，使用非常方便。

用户打开Nios Ⅱ IDE后，要新建应用程序，选择File|New|C/C++ Application命令，如图5-4所示。

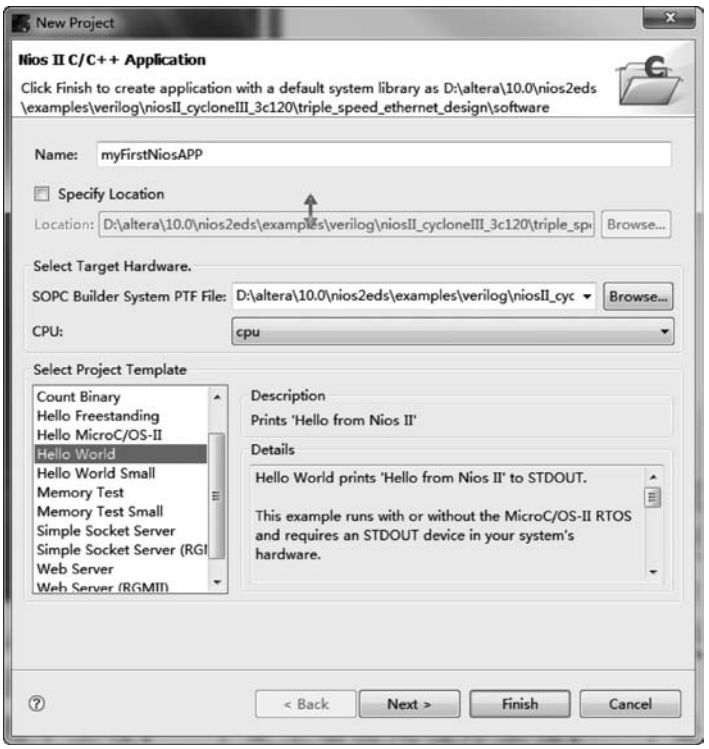


图5-4 Nios Ⅱ IDE新建工程

在新建工程窗口下，用户输入工程名，同时选择应用程序的目标硬件，也就是SOPC Builder产生的Nios Ⅱ系统*.ptf描述文件。Nios Ⅱ IDE还提供了一些应用程序模板，用户也可以选择空模板或自己建立的模板。

如图5-5所示，在下一个配置页面中，用户需要选择新建系统库工程，还是利用已有的系统库工程。如果选择新建，该系统库工程就是基于目标硬件的*.ptf文件建立的，这个工程会自动命名，它实际上是一个硬件抽象层(Hardware Abstract Layer，HAL)，它能够使上层应用程序像访问C程序库一样访问系统硬件和文件。在本节后面部分将介绍HAL。

这样在IDE中就产生了上层应用程序工程和系统库工程。用户应用程序源文件应该加到上层应用工程中。

图 5-5　Nios Ⅱ IDE 工程库建立选项

在上层应用程序工程上右击，在弹出的菜单中选择 Build Project 命令，就可以编译整个工程。

类似地，在 C/C++工程上右击，用户可以选择 Run As 或 Debug As(图 5-6)，Run As 是在硬件或者指令集仿真器(ISS)运行程序，Debug As 是在硬件或者指令集仿真器(ISS)调试程序。

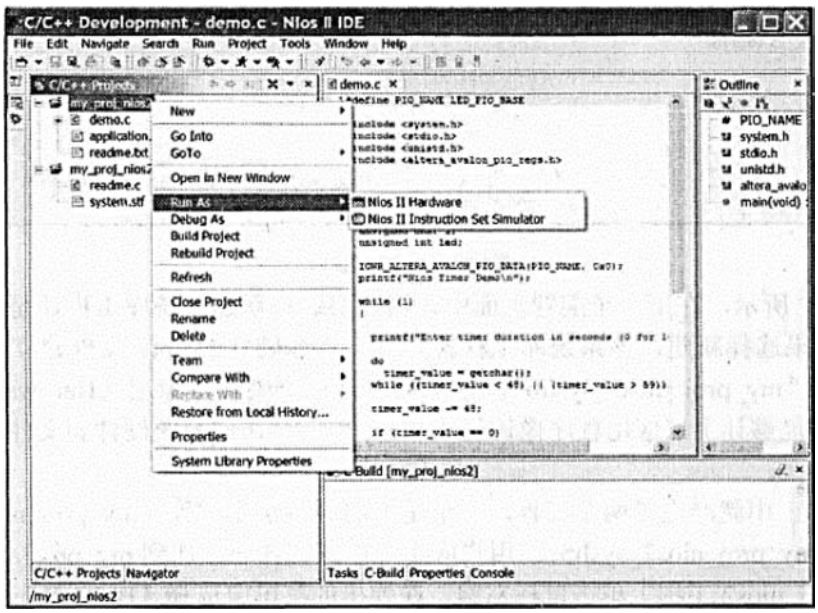


图 5-6　Nios Ⅱ工程运行及调试

对通用 IDE 中的软件运行和调试方法，软件工程师应该是非常熟悉的，这里不再介绍，感兴趣的读者可以参考 Nios Ⅱ的相关资料文档。

2. 硬件抽象层

硬件抽象层是指在应用程序和系统硬件之间的一个系统库。软件工程师可以非常方便地使用这些系统库来与底层硬件通信，而无须关心底层硬件实现细节。这样，在上层应用程序和底层硬件之间就构成了一个明显的界限，底层驱动程序的修改不会对应用程序造成任何影响。

HAL API 集成了 ANSI C 的标准库，它允许应用程序使用类似 C 库函数的方式访问硬件和文件。实际上，HAL 的目的也是要使得软件工程师可以像以前的方式一样开发基于 Nios Ⅱ的程序，使用类似的系统库。Nios Ⅱ硬件抽象层如图 5-7 所示。

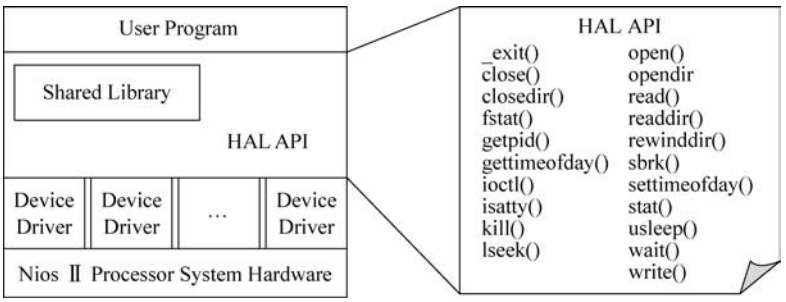


图 5-7　Nios Ⅱ 硬件抽象层

HAL 是 Nios Ⅱ IDE 根据系统具体的硬件配置来产生的，它包括硬件驱动、初始化软件、文件系统、stdio 和 stderr。

基于 HAL 的程序在启动时，首先运行一段启动程序_start()，用来初始化 Cache，建立堆栈等工作；然后调用 alt_main()，初始化操作系统，中断控制器，而且将调用 alt_sys_init()函数来初始化硬件驱动程序等；最后调用应用程序中的 main()函数，进入应用程序运行，如图 5-8 所示。

用户也可以通过自己定义 alt_main()来定制系统初始化过程，这样做比较麻烦。

用户可以采用 HAL 来初始化系统，也可以用独立的程序来做这个工作，不过 Altera 公司并不建议这样做，因为这样完全抛弃了 HAL 的好处。自己负责所有模块的初始化工作，是一项烦琐且容易出错的工作，若用户这样做是为了减小程序的空间，则可以采用 IDE 中的优化方法来优化程序空间。

有了 HAL，在 Nios Ⅱ的系统开发过程中，软件工程师可以不用关心硬件的具体实现细节，而是按照以前习惯的方式工作，适应在 Nios Ⅱ系统上开发应用程序；硬件工程师可以把主要精力放在实现系统结构和设计外设的驱动程序上。

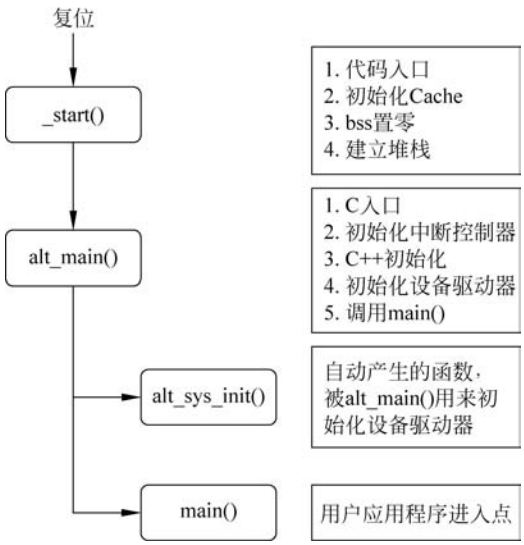


图 5-8　HAL 启动过程

3. RTOS 支持

实时操作系统(RTOS)在复杂的嵌入式软件设计中经常用到。Nios Ⅱ系统支持的RTOS 包括 MicroC/OS-Ⅱ、Nucleus PLus、uCLinux 和 KROS，用户可以根据自己的需要选择。

MicroC/OS-Ⅱ是比较常用的一种，它的主要功能如下：

(1) 任务(线程)管理；

(2) 事件标记；

(3) 消息传递；

(4) 内存管理；

(5) 标志位；

(6) 时间管理。

MicroC/OS-Ⅱ内核工作在 HAL 的顶部。有了 HAL 这一层，基于 MicroC/OS-Ⅱ的程序具有更好的可移植性，而且不受底层硬件改变的影响，如图 5-9 所示。

4. Flash 编程器

许多用户设定的系统，采用 Flash 来存储数据，包括程序代码、程序数据、FPGA 配置文件或者其他数据。

Nios Ⅱ IDE 提供了一种叫作 Flash 编程器的工具，可以帮助用户在线把数据内容烧制到 Flash 中。支持的 Flash 必须是 CFI 接口，或是 Altera 公司的串行配置器件 EPCS 系列。

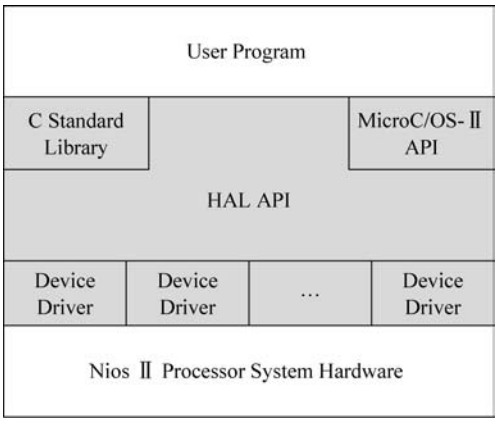


图 5-9　基于 MicroC/OS-Ⅱ的程序结构

使用 Flash 编程器必须完成两步：

(1) 要产生一个 Flash 编程器的设计；

(2) 由主机通过 IDE 中的 Flash 编程器把 Flash 的内容发送给板上运行的 FPGA，由 FPGA 中专用于烧制 Flash 的设计(Flash 编程器的设计)把内容烧制到 Flash 中。

Flash 编程器工作示意如图 5-10 所示。

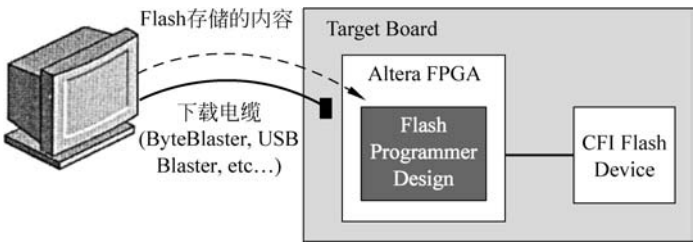


图 5-10　Flash 编程器工作示意图

Flash 编程器的设计包括以下内容：

(1) Nios Ⅱ CPU(一个 Nios Ⅱ处理器)；

(2) JTAG UART (JTAG 串口)；

(3) Active serial memory interface(主动串行存储器接口)；

(4) Tri-state bridge(三态桥)；

(5) CFI—compatible flash interface(CFI 兼容的闪存接口)；

(6) System ID peripheral on-chip memory for formware and buffers(一个作为系统 ID 外设的片上存储器，用在固件和缓冲区上)。

用户需要在 SOPC Builder 中生成一个 Flash 编程器的设计，然后编译生成配置文件对板上的 FPGA 进行配置。

用户必须在主机上运行 IDE 中的 Flash 编程器，指定 Flash 的内容和地址空间，然后对 Flash 进行编程，如图 5-11 所示。

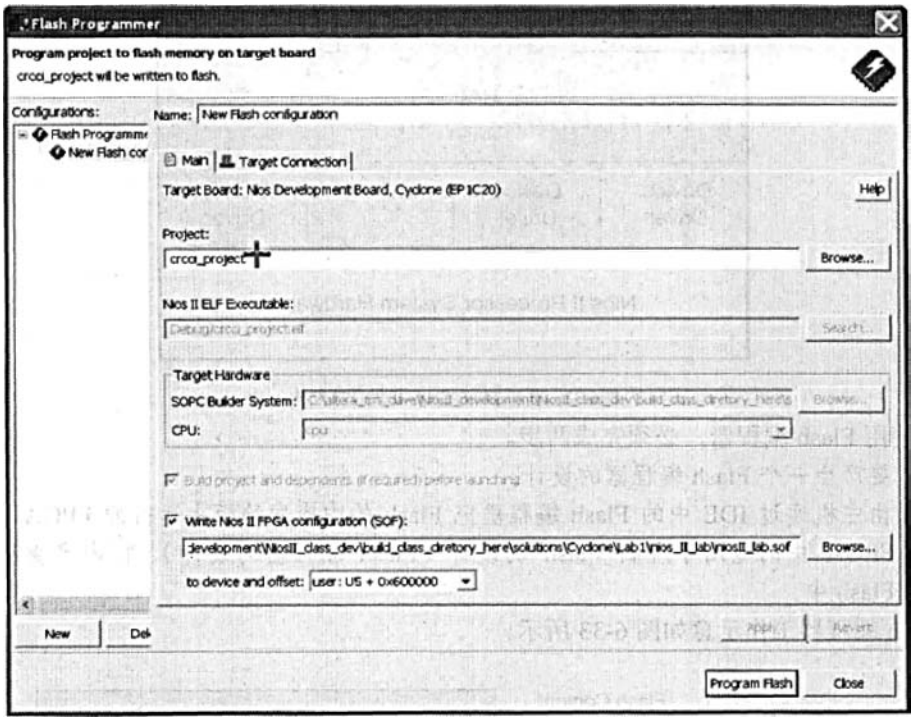


图 5-11　Flash 编程器界面

5.3　SOPC Builder 设计流程

采用 Nios 处理器设计 SOPC 嵌入式系统时，开发流程(图 5-12)如下：

(1) 定义 Nios Ⅱ嵌入式处理器系统。使用 SOPC Builder 系统综合软件从处理器库中选取合适的 CPU。从外设库中选取合适的存储器及外围器件，选择定制指令，最后形成系统模块。

(2) 指定目标器件、连接各个模块，分配引脚、编译硬件。使用 Quartus Ⅱ选取 Altera 器件系列，选择需要用的 IP 模块并设定参数，对 Nios Ⅱ系统的各种 I/O 口分配引脚。由 SOPC Builder 编译后生成系统，也生成配置文件。

(3) 在上一步会生成网表文件、HDL 源文件和 Testbench 测试文件。用 SOPC Builder 生成的 HDL 设计文件进行综合和布局布线，进行硬件编译选项或时序约束的设置，生成网表文件。

(4) 硬件下载。使用 Quartus Ⅱ软件和下载电缆，将配置文件下载到开发板上的 FPGA 中。当校验完当前硬件设计后，还可再次将新的配置文件下载到开发板上的非易失存储器中。

(5) 在使用 SOPC Builder 进行硬件设计的同时，就可以开始编码独立于器件的 C/C++软件，如算法或控制程序。用户可以使用现成的软件库和开放的操作系统内核来

加快开发过程。

(6) 在 Nios Ⅱ IDE 中建立新的软件工程,IDE 会根据 SOPC Builder 对系统的硬件配置自动生成一个定制 HAL 系统库。这个库能为程序和底层硬件的通信提供接口驱动程序。

(7) 使用 Nios Ⅱ IDE 对软件工程进行编译、调试。

(8) 将硬件设计下载到开发板后,就可以将软件下载到开发板上并在硬件上运行。

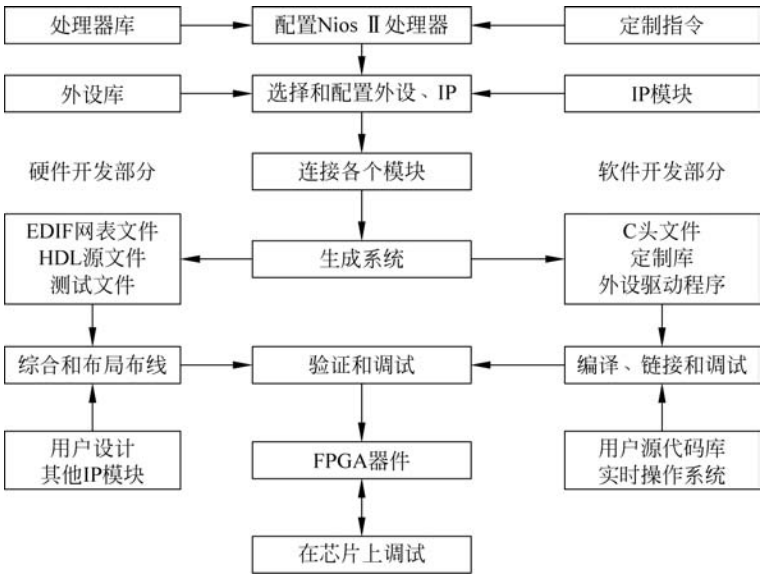


图 5-12 SOPC 系统开发流程

5.4 Nios Ⅱ体系结构

Nios Ⅱ是 Altera 公司特有的基于通用 FPGA 构架的 CPU 软内核。随着用户对系统可定制性和灵活性需求的逐步增加,Nios Ⅱ被越来越多的用户接受。本节简单介绍基于 Nios Ⅱ的嵌入式系统及其应用。

5.4.1 Nios Ⅱ处理器系统

传统的设计模式是把各个芯片厂家的产品集成到一个 PCB 上完成相应的功能,这样产品的灵活性和性能都受到影响。而随着业界 IP 核的兴起,工程师逐渐把以前的由分立芯片实现的系统放在一个芯片中(片上系统),用户可以根据自己的具体需求来定制芯片的功能模块和规格。

当 SOC 的概念盛行时,许多专用芯片公司纷纷把嵌入式处理器内核放到自己的 ASIC 中,构建自己的片上系统,其中用户数较多的是 ARM 内核。ARM 不仅提供了嵌

入式 CPU，也提供了 SOC 的解决方案，包括内部总线、外设等。

两大 PLD 供应商 Altera 公司和 Xilinx 公司也分别把 ARM 和 PowerPC 硬核放到了自己的 FPGA 中，然而这种看似性能强大的 FPGA 内嵌处理器并没有取得较大成功，而 Altera 公司的应用在中低端的 CPU 软内核 Nios 在客户中取得了不错的口碑。随着 Nios 的成功，Altera 公司的 SOPC 概念也被许多用户所接受。

2004 年 6 月，Altera 公司继在全球推出 Cyclone Ⅱ 和 Stratrix Ⅱ 器件系列后，又推出支持这些新款芯片的 Nios Ⅱ 嵌入式处理器。它是目前 SOPC 设计的主流产品，是 Altera 公司推出的第二代 32 位软核微处理器。它采用哈佛体系结构，除了需要购买 license 许可来开发 Nios Ⅱ 系统外，用户可以把 Nios Ⅱ 用在自己的产品中，不需要缴纳其他的版权费。

Nios Ⅱ 及所有外设都是以 HDL 代码的形式提供的，能够使用 Quartus Ⅱ 集成综合工具进行综合，并用于 Altera 公司所有的 FPGA 芯片。通过 Altera 公司提供的系统集成工具，能够快速生成包括 Nios Ⅱ 处理器、各种嵌入式外设及系统互连的所有 HDL 源代码。设计生成的 SOPC 系统设计文件，在 Quartus Ⅱ 软件中完成综合和布局布线操作后，在 FPGA 逻辑资源中实现，最终生成 FPGA 的编程文件。

在可编程逻辑器件中，用户使用 CPU 绝大部分并不是为了追求性能，而是为了 PLD 特有的灵活性和可定制性，同时也可以提高系统的集成度，这些正是 Nios 系统的内存特性，也是 Nios 受欢迎的原因。

Nios 是 Altera 公司开发的嵌入式 CPU 软内核，几乎可以用在 Altera 公司所有的 FPGA 内部。Nios 处理器及其外设都是用 HDL 编写的，在 FPGA 内部利用通用的逻辑资源实现，所以在 Altera 公司的 FPGA 内部实现嵌入式系统具有极大的灵活性。凭借不错的性能和非常灵活的配置，Nios 已被许多客户所接受。Nios 常应用在一些集成度较高、对成本敏感以及功耗要求低的场合，如远程测量和医疗诊断设备。在光传输和存储网络等对性能和灵活性都有要求的领域，也有 Nios 应用的例子。

Altera 公司在 Nios 的基础上推出了第二代嵌入式 CPU 软核 Nios Ⅱ。与前一代相比，其用户的配置和使用更加灵活方便，同时在占用的逻辑资源和性能上都有明显改善。

Nios Ⅱ 处理器是一个通用的 32 位 RISC 处理器内核。它的主要特点如下：

(1) 完全的 32 位指令集、数据通道和地址空间；

(2) 可配置的指令和数据 Cache；

(3) 32 个通用寄存器；

(4) 32 个有优先级的外部中断源；

(5) 单指令的 32×32 乘除法，产生 32 位结果；

(6) 专用指令用来计算 64 位或 128 位乘积；

(7) 单指令 Barrel Shifter(桶形移位器)；

(8) 可以访问多种片上外设，可以与片外存储器和外设接口；

(9) 具有硬件协助的调试模块，可以使处理器在 IDE 中执行开始、停止、单步和跟踪等调试功能；

(10) 在不同的 Nios Ⅱ系统中,指令集结构(ISA)完全兼容;

(11) 性能达到 150DMIPS(150×10^2 万条指令/s)以上。

Nios Ⅱ处理器内核有 3 种类型,分别是快速型、经济型和标准型,用来满足不同设计的要求。快速型 Nios Ⅱ内核具有最高的性能,经济型 Nios Ⅱ内核具有最低的资源占用,标准型 Nios Ⅱ在性能和面积之间做了一个平衡。三种 CPU 的性能比较见表 5-1。

表 5-1 Nios Ⅱ处理器的三种处理器内核

性能	快速型(Nios Ⅱ/f)	标准型(Nios Ⅱ/s)	经济型(Nios Ⅱ/e)
用途	用于最佳性能优化	比第一代 Nios CPU 的速度快,体积更小	用于最小逻辑资源占用优化
流水线	6 级	5 级	无
乘法器	1 周期	3 周期	软件仿真实现
支路预测	动态	静态	无
指令缓冲	可设置	可设置	无
数据缓冲	可设置	无	无
定制指令	256	256	256

我们所说的 Nios Ⅱ处理器系统,包括一个可配置的 CPU 软内核、FPGA 片内的存储器和外设、片外的存储器和外设接口。Nios Ⅱ处理器系统的典型架构如图 5-13 所示。

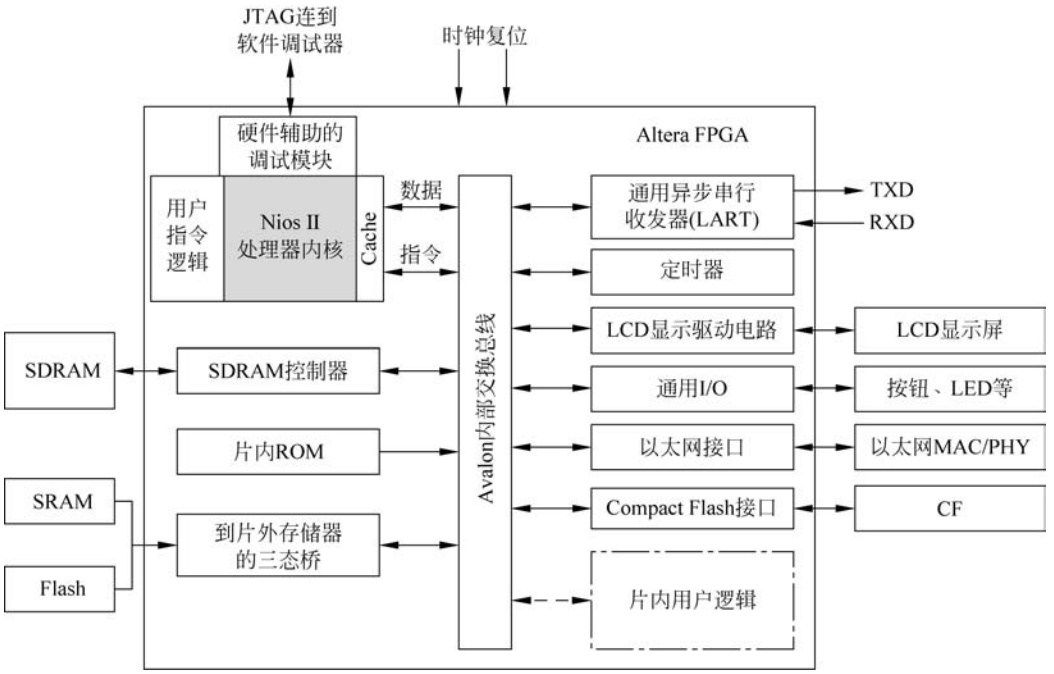


图 5-13 Nios Ⅱ处理器系统的典型架构

在图 5-13 中,Nios Ⅱ处理器系统由 Nios Ⅱ处理器内核(包括调试模块)、Avalon 交换总线、系统外设和片内用户逻辑组成。

系统中的外设如 SDRAM 控制器、片内 ROM、三态桥、UART、定时器、LCD 显示驱动电路、通用 I/O、以太网接口和 Compact Flash 等，都是由 FPGA 内部的逻辑和 RAM 资源实现的。

Nios Ⅱ开发包含一套通用外设和接口库，MegaCore 或者 Altera Megafunction Partners Program(AMPP)也提供一些外设。

Nios Ⅱ开发包含的通用外设和接口有定时器/计数器、外部三态桥接、外部 SRAM 接口、UART、LCD 接口、用户逻辑接口、JTAG UARTC、并行 I/O、S8900 10Base-T 接口、系统 ID、EPCS 串行闪存控制器片内 ROM、直接存储器通道(DMA)、紧凑闪存接口(CFI)、串行外设接口(SPI)、SDR SDRAM、片内 RAM、LAN91CXX10/100 网络控制器、有源串行存储器接口、PCI 等。

MegaCore 或者 AMPP 提供的外设有 PCI、DDR SDRAM、CAN、RNG、USB、DDR2 SDRAM、DES、16550 UART、RSA、SHA-1、I2C、10/100/1000 Ethernet MAC、浮点单元。

利用 SOPC Builder 软件工具可以生成用户定制外设，并将其集成在 Nios Ⅱ处理器系统中。在 Altera FPGA 中，组合实现现有处理器无法达到的嵌入式处理器配置。

不同的用户，其设计类型的差别很大。在有些用户的设计中，CPU 为主要部件，需要较强大的性能，除了实现 Nios Ⅱ处理器系统外，少数 FPGA 中剩余的资源可以用作粘合逻辑。而在另外一些设计中，Nios Ⅱ处理器系统只占用了 FPGA 的一小部分功能，性能要求也不高，剩下的逻辑资源是为了实现主要的逻辑功能。这就要求用户根据自己的系统需求，选择合适的 FPGA 规模。在这些系统中，如果用户逻辑需要和 Nios Ⅱ处理器系统间相互通信，用户逻辑可以非常方便地直接挂在片内的 Avalon 交换总线上，而且访问时序可以由用户自己定义。

Nios Ⅱ是一个可灵活配置的软内核处理器。可灵活配置是指 Altera 公司提供的处理器并不是固定的微控制器，用户可以根据自己设计的性能或成本要求，灵活地增加或裁减一些系统特性和外设，甚至可以在系统中放置多个 Nios Ⅱ处理器内核，以满足应用要求。Nios Ⅱ处理器定制指令扩展了 CPU 指令集，可以提高对时间要求严格的软件运行速度，从而使开发人员能够提高系统性能。采用定制指令，可以实现传统处理器无法达到的系统性能。

Nios Ⅱ处理器可支持 256 条定制指令，加速通常由软件实现的逻辑和复杂数学算法。例如，在 64KB 缓冲中，执行循环冗余编码计算的逻辑模块，其定制指令速度比软件快 27 倍。Nios Ⅱ处理器支持固定和可变周期操作，其向导功能将用户逻辑作为定制指令输入系统，自动生成便于在开发人员代码中使用的软件宏功能。

软内核是指 Nios Ⅱ是以一种“软”(加密网表)的设计形式交给客户使用的，它可以在 Altera 公司的 FPGA 内部实现。用户根据自己的需要定制 Nios Ⅱ处理器的数量、类型(3 种类型)，也可以自己定义需要的外设种类和数量，还可以自由分配外设的地址空间，甚至可以自己定制 Nios Ⅱ的指令，使得一些耗时耗资源的操作在用户指令中实现。由 FPGA 内部的其他资源(如 LE、RAM、DSP 块)来实现这些特殊的用户定制指令功能

块，可以提高某些特殊操作的性能，而且对软件设计人员来说用户自定义的指令和系统自带的指令没有区别。

Altera 公司的 SOPC Builder 工具使得用户产生 Nios Ⅱ处理器系统的过程非常简单。在 SOPC Builder 中，用户可以建立自己的系统，包括 Nios Ⅱ处理器、片内和片外的 RAM、外设(如以太网等)。SOPC Builder 自动使用 Avalon 交换结构将它们互连起来，而不需要进行任何的原理图或 HDL 代码的输入。在 SOPC Builder 中可以自动为这些外设指定地址空间，增加仲裁机构，也可由用户设置访问优先级等。

在 SOPC Builder 中也可以输入一个用户自己设计的模块，使得集成用户逻辑到 Nios Ⅱ系统中变得非常方便。将用户逻辑加到 Nios Ⅱ系统中，在 SOPC Builder 中可以采用两种方法：一种是将用户逻辑的代码引入 Nios Ⅱ的系统中，系统可以一起仿真；另一种是在 SOPC Builder 中，仅将用户逻辑接口留出来，需要在设计的顶层将用户逻辑和 Nios Ⅱ系统实例化并连在一起。

在 Nios Ⅱ系统的开发过程中，可以认为硬件细节对软件开发人员来说是透明的。Nios Ⅱ的软件开发环境称作 Nios Ⅱ集成开发环境(Nios Ⅱ IDE)。Nios Ⅱ是基于 Eclipse IDE 和 GNU C/C++编译器的，它提供给软件开发人员一个熟悉的开发环境，可以用来对 Nios Ⅱ系统的软件进行编译、仿真和调试。Nios Ⅱ也提供了 Flash Programmer 功能，在软件调试完成以后，可以通过 Flash Programmer 把应用程序烧到 Flash 中，使得设计在上电配置完成以后，自动从 Flash 中开始运行程序。

5.4.2 Avalon-MM 总线架构

Nios Ⅱ处理器采用 Avalon-MM 总线架构。Avalon 交换架构能够同时处理多路数据，实现巨大的系统吞吐量。SOPC Builder 自动生成的 Avalon 交换架构针对系统处理器和外设的专用互连需求进行优化。传统总线结构中，单个总线仲裁器控制总线主机和从机之间的通信。

每个总线主机发起总线控制请求，由总线仲裁器对某个主机授权接入总线。如果多个主机试图同时接入总线，总线仲裁器就会根据一套固定的总裁规则分配总线资源给某个主机。这样，每次只有一个主机能够接入总线使用总线资源，因而会导致带宽瓶颈。

采用 Avalon 交换架构，由于 FPGA 内部有丰富的互连资源，各个主、从设备之间实际上是点到点的互连。每个总线主机均有自己的专用互连，总线主机只需抢占共享从机而不是总线本身。Avalon 交换架构的同时多主机体系结构提高了系统带宽，消除了带宽瓶颈。

每当系统加入模块或者外设接入优先权改变时，SOPC Builder 利用最少的 FPGA 资源产生新的最佳 Avalon 交换架构。

Avalon 交换架构支持多种系统体系结构，如单主机/多主机系统，能够实现数据在外设与性能最佳数据通道之间的无缝传输，Avalon 交换架构同样支持设计的片外处理器和外设。

Nios Ⅱ处理器有三种运行模式，分别为用户模式、超级用户模式和调试模式。系统程序代码通常运行在超级用户模式。V6.0 版本以前的 Nios Ⅱ处理器都不支持用户模式，永远运行在超级用户模式。用户模式是超级用户模式功能访问的一个子集，它不能访问控制寄存器和一些通用寄存器；超级用户模式除了不能访问与调试有关的寄存器(btstatus、bastatus 和 bstatus)外，无其他访问限制；调试模式拥有最大的访问权限，可以无限制地访问所有的功能模块。

5.5 Nios Ⅱ系统典型应用

Nios 的出现改变了人们使用 CPU 的传统概念，用户可以在任何有 Altera FPGA 的系统中使用 CPU。有些 CPU 甚至只是在用户做系统调试或者测试时用到，而将在正式发布的产品中去掉。人们可以用 Nios 实现传统 CPU 无法实现的功能。

为了启发读者使用 Nios Ⅱ系统，这里给出 5 种应用实例。

1. 定制处理器系统

设计传统的嵌入式系统时，CPU 和外设均采用分立器件，在 PCB 上实现互连，如图 5-14 所示。大量分立器件占用了不少 PCB 的空间，因此集成度较差。

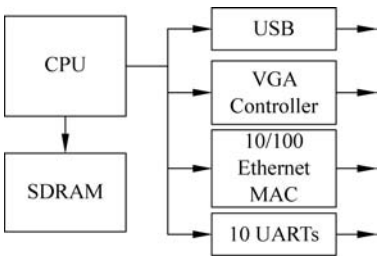


图 5-14 分立器件实现嵌入式系统

如果采用 Nios 系统，单板上只需要一个 Altera FPGA 即可实现整个系统，这样大大提高了系统的集成度和可靠性，如图 5-15 所示。

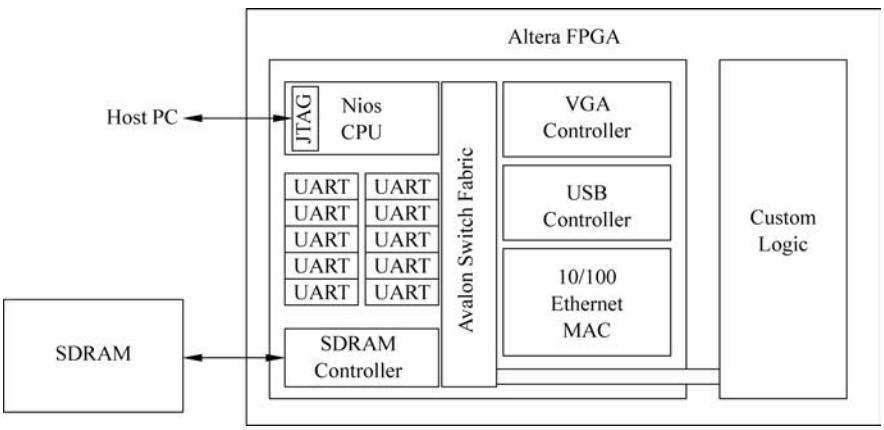


图 5-15 Nios 实现嵌入式系统

2. 作为协处理器

如果在一个系统中已有性能较高的 PowerPC，但是出于其需要处理大量的 I/O 工作

而严重影响了它的性能。如果单板上也同时有一个 Altera FPGA，用户就可以考虑使用 Nios 来做一些 I/O 处理方面的工作，以减轻 PowerPC 的工作负担，如图 5-16 所示。

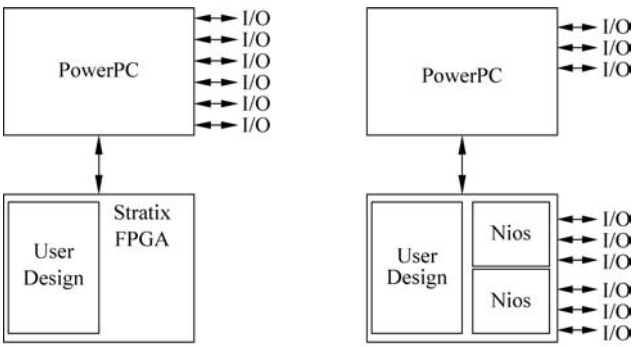


图 5-16　Nios 用作协处理器

3. I/O 处理

在一些高速的 I/O 处理中，纯粹用传统的逻辑电路去实现，占用的资源较多，而且实现控制不太灵活。如果在 I/O 模块中增加一个 Nios 处理器，就可以有效地实现 I/O 中比较复杂的控制功能。如图 5-17 所示的 MAC 模块，使用 Nios 来实现一些状态机的控制功能，而只用逻辑去实现高速的数据通道，这样占用的资源较少，同时兼具了控制的灵活性。

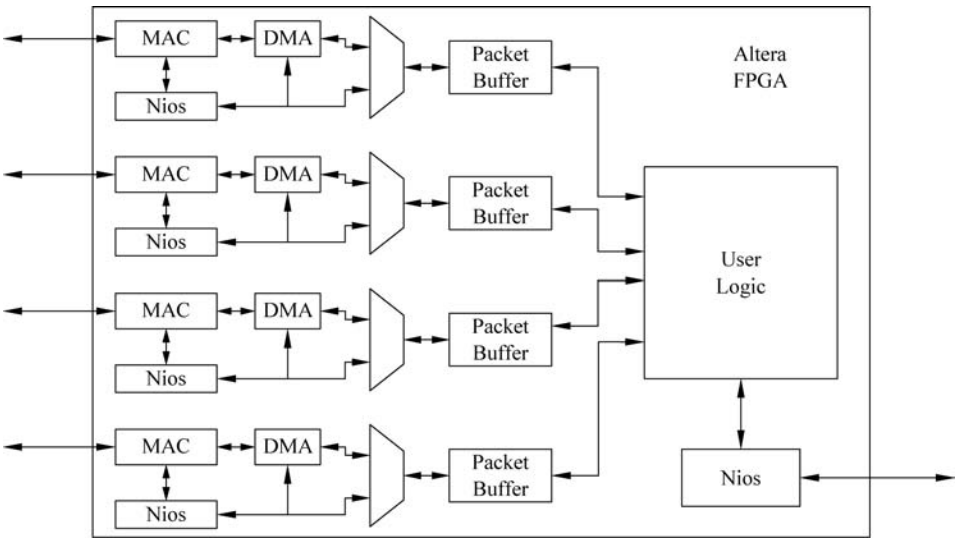


图 5-17　Nios 实现 I/O 处理

4. 替代状态机电路

在传统的设计中，状态机用独立的 HDL 来实现，这样的状态机实现起来非常复杂，占用的资源也较多，修改起来也比较麻烦。如果采用 Nios 来实现状态机，问题就简单得多，状态机成了软件的工作，这样既节省逻辑资源，也缩短开发的周期，将来的维护也更方便，如图 5-18 所示。

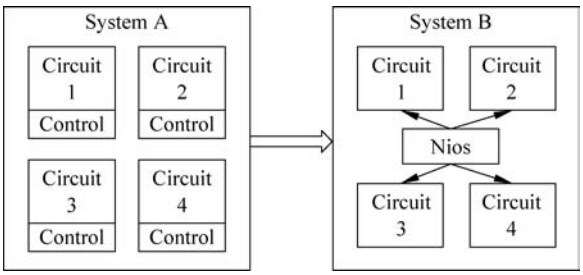


图 5-18　状态机替换

5. 实现测试功能

在一个复杂的设计中，即使没有用到 Nios 处理器，在系统调试和测试阶段也可以在系统中增加一个 Nios，以方便调试。毕竟 Nios 实现一些测试功能（如统计分析等）更灵活方便。

5.6　DSP Builder 工具

DSP Builder 是 Altera 公司提供的一种 DSP 系统设计工具。它是 MathWorks 公司的 MATLAB/Simulink 设计工具和 Altera 公司的 Quartus Ⅱ设计工具之间的一座桥梁，把 MATLAB/Simulink 的 DSP 系统设计转化为 HDL 文件，在 Quartus Ⅱ开发平台中实现到具体的器件中。

5.6.1　DSP Builder 设计流程

1. 在 Simulink 中构建系统并仿真

在 Simulink 环境中装入 Altera 的 DSP 库，其中包括一些基本的算术单元和 DSP 类的 IP 核，如图 5-19 所示。

直接将这些 DSP 模块增加到 Simulink 的设计框图中，用户需要参数化其中的模块，包括一些 IP 核，如图 5-20 所示。

系统中的功能模块在 Simulink 中建立完毕后，可以在其中对系统功能进行仿真，以保证设计功能正确。仿真波形实例如图 5-21 所示。

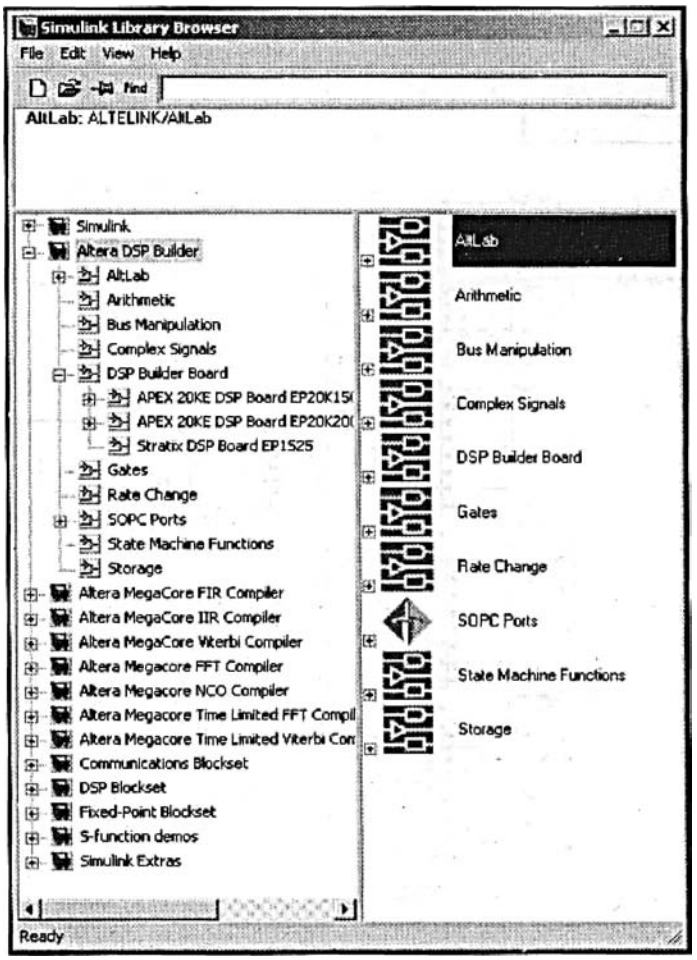
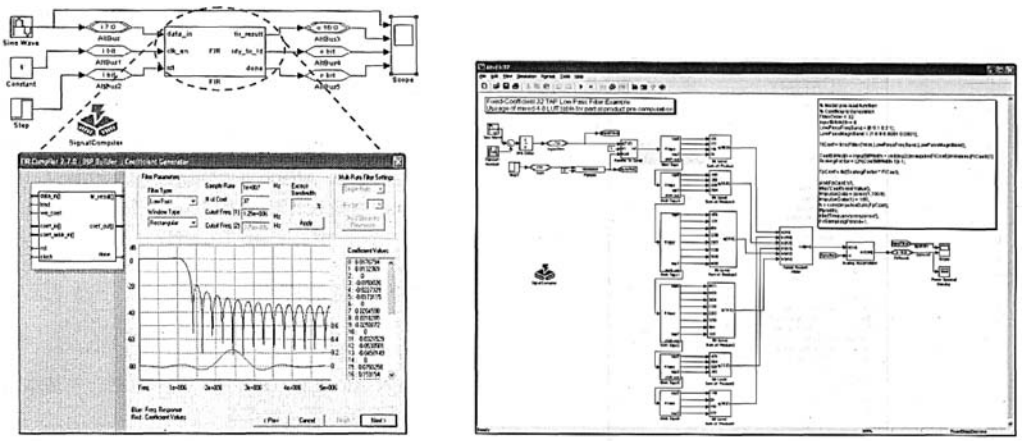


图 5-19　Simulink 中的 Altera DSP 库



(a)　　(b)

图 5-20　Simulink 中构建 DSP 系统

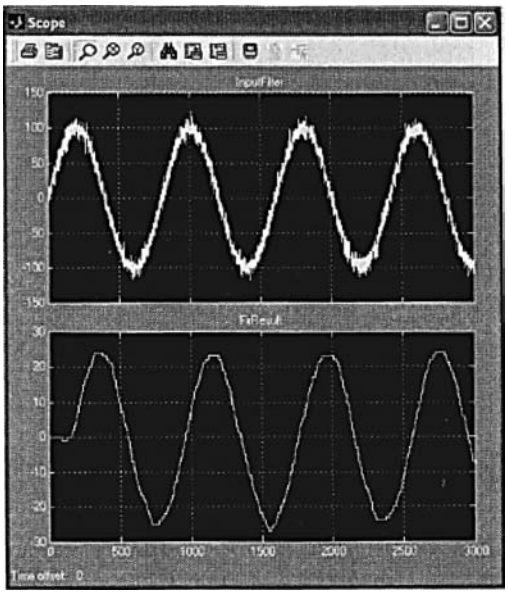


图 5-21　Simulink 中的模型仿真波形

2. 使用 Signal Compiler 产生 HDL 文件和 testbench

如果已在 Simulink 中完成仿真，就使用 Signal Compiler 将系统模型转成 HDL 文件，同时输出 testbench，供 HDL 仿真器仿真设计功能，如图 5-22 所示。

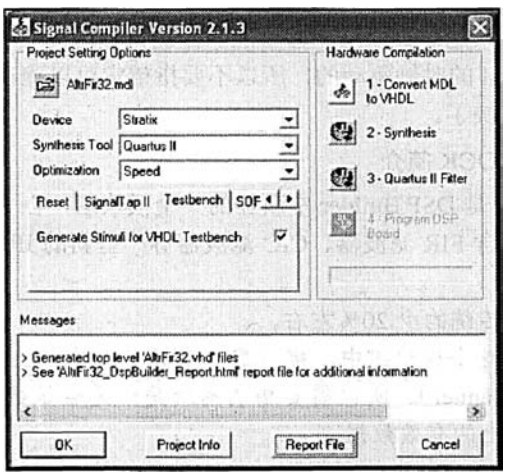


图 5-22　生成 HDL 文件和 testbench 输出

3. 使用 HDL 仿真工具仿真并用 Quartus Ⅱ 工具实现

用户可以利用 Signal Compiler 输出的 HDL 代码和 testbench 来仿真设计的功能。

如果完成仿真，同样可以在 Signal Compiler 调用 Quartus Ⅱ工具的综合和布局布线功能来实现设计，如图 5-23 所示。

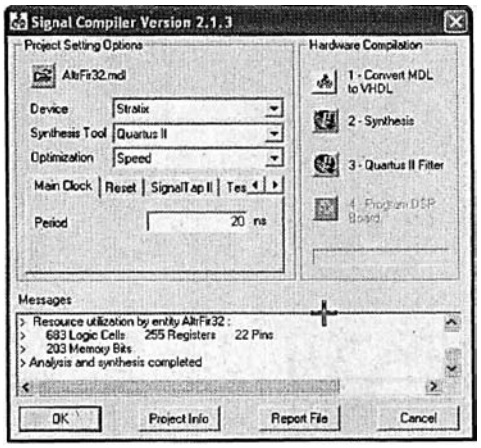


图 5-23　实现设计

最后，用户可以把设计下载到实际器件中验证其功能。

4. HIL

DSP Builder 中的硬件在回路(Hardware in the Loop，HIL)是一个非常有创新性的概念。目前的 FPGA 平台，其芯片是可以无限次写入的。在进行系统设计时经常无法确定设计本身是否能正常工作，往往需要首先进行功能仿真；具体在实施仿真时也存在很多问题，例如，速度会太慢，最重要的是无法确定仿真模型是否能完整描述实际系统。所以可以尝试用 HIL 来加速仿真。HIL 把设计包裹在一套接口中进行编译，然后下载到板子的 FPGA 中。Simulink 通过下载电缆把测试数据不断地输入，然后在输出端不断地获得硬件“跑”出来的结果。通过这样一个过程，能保证仿真达到多快好省的效果。但是，这也只是一种仿真模式，并不能叫作测试，因为它不是在真实的时钟频率下操作的。时钟是通过 JTAG 的时钟驱动的，一般工作频率低于 100MHz，速度能够满足需要。

5. Advanced Block 简介

Advanced Block 是 DSP Builder 中新加的一个模块组件，在 Quartus Ⅱ 8.0 以后的版本中才有这个新特性。它包含 FIR 滤波器、级联积分器梳状(CIC)滤波器等新的 IP 模块，这些 IP 与以往的相比有很大改进。例如：

(1) 所消耗的资源比传统的少 20%左右；

(2) 多通道支持。在这个模块组中，接口都非常简单，基本上就是 V(Valid)、D(Data)、C(Channel)。需要指出的是：是否是数据，是否是有效数据，是哪个通道上的有效数据。

(3) 自动插流水寄存器。这是一个比较高级的功能，就是在设计中加入寄存器。在

设计中无须预先放置任何寄存器，只告诉工具用户想要的时钟频率、目标器件，工具可以自动在电路中插入流水寄存器。这样可以保证设计完全使用器件的最大资源，同时不会出现时序问题。缺点是无法预知时延。一般情况下，如果一个设计是流水线模式的，其时延是多少并不重要。例如：用 Advanced Block 生成的 FIR 滤波器在 Cyclone Ⅲ FPGA 中可以轻松“跑”到 220MHz，而用传统的 IP 在同样器件下只能“跑”到 180MHz 左右。

(4) 系统层面的设计。这也是一个比较高级的功能，所有设计中的寄存器都会被编入一个系统地址查找表，如 FIR 的系数，一些控制寄存器都会有不同的地址。可以通过一个系统接口对这些寄存器进行操作，使整个设计更具有系统化概念。在编译的同时会生成一个寄存器列表(网页格式)，里面包含了寄存器名字、地址和初始值。

通过这个高级模块的增强功能使算法方面的实现与设计变得更加容易，也可以很容易地实现非常复杂的系统。

在 MATLAB Simulink 中调用 Advanced Block，如图 5-24 所示。

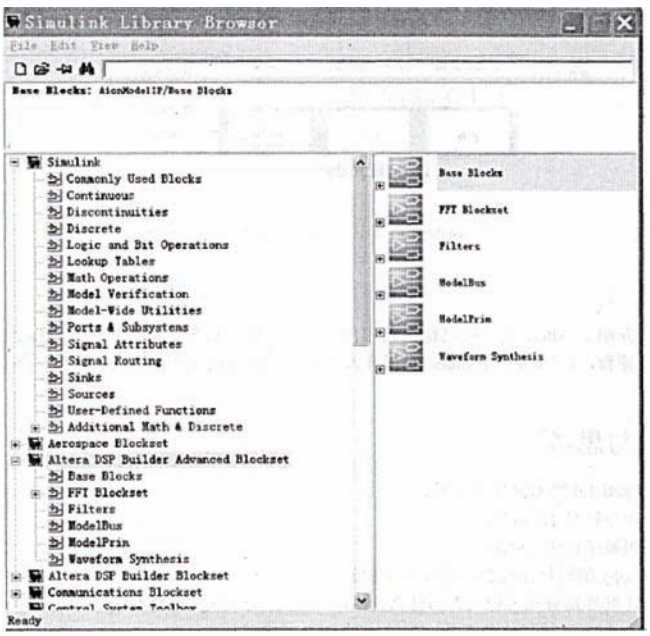


图 5-24　在 Simulink 中调用 Advanced Block

5.6.2　DSP Builder 与 SOPC Builder 一起构建系统

在一些设计中，最好的选择不是用纯软件或纯数字逻辑，而是同时需要处理器的灵活性和硬件电路的高性能，如图 5-25 所示。

在构建系统时，用户可以利用 SOPC Builder 来构建处理器系统，同时用 DSP Builder 来构建硬件加速系统，然后把它们实现到一个 FPGA 中，如图 5-26 所示。

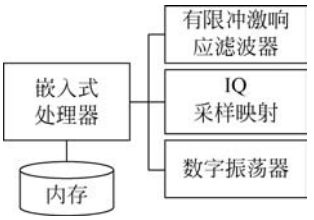


图 5-25　处理器＋硬件加速

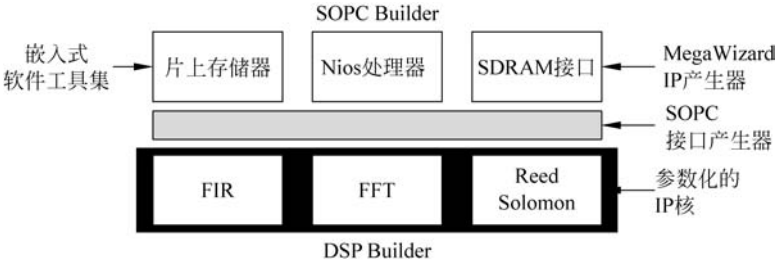


图 5-26　SOPC Builder 和 DSP Builder 设计模块的接口

5.7　Avalon 总线规范

在 SOC 的设计过程中，最具特色的是 IP 复用技术，即用户选择所需功能的 IP 核，然后集成到可编程芯片中。由于 IP 核的设计千差万别，IP 核的接口就成为构造 SOC 的关键。片上总线(OCB)是实现 SOC 中 IP 核连接常见的技术手段，它以总线方式实现 IP 核之间的数据通信。与板上总线不同，片上总线不需要驱动底板上的信号和连接器，使用更简单，速度更快。片上总线规范一般需要定义各个模块之间初始化、仲裁、请求传输、响应、发送接收等过程中的驱动、时序、策略等关系。

片上总线与板上总线应用范围不同，存在着较大的差异，其主要特点如下：

(1) 片上总线尽可能简单。一是结构简单，这样可以占用较少的逻辑单元；二是时序简单，以利于提高总线的速度，三是接口简单，可减少与 IP 核连接的复杂度。

(2) 片上总线有较大的灵活性。由于片上系统应用广泛，不同应用对总线的要求各异，因此片上总线需要具有较大的灵活性。其主要表现为：一是多数片上总线的数据和地址宽度都可变，如 AMBA AHB 支持 32～128 位数据总线宽度；二是部分片上总线的互连结构可变，如 Wishbone 总线支持点到点、数据流、共享总线和交叉开关四种互连方式；三是部分片上总线的仲裁机制灵活可变，如 Wishbone 总线的仲裁机制可以完全由用户定制。

(3) 片上总线对功耗的要求严格。在实际应用时，总线上各种信号尽量保持稳定，就能降低动态功耗。另外，要求多采用单向信号线，从而进一步降低功耗，同时也简化了时序。片上总线的输入数据线和输出数据线是分开的，也没有板上总线常有的地址线与数

据线复用的现象。

片上总线一般分为高性能的系统总线与低功耗的外围总线两个部分。系统总线用来连接微处理器、DMA 控制器、片上存储器和其他具有高带宽通信要求的设备。系统总线可以连接多个主设备,因而需要总线仲裁来控制各个主设备对总线的访问请求。系统总线的特点是高速、高带宽。外围总线用来连接对速度要求不高的各类外围设备。在外围总线上通常只有一个主设备,因此其协议比系统总线协议简单。外围总线的特点是低速、低带宽,但是它要满足低功耗、重用性等方面的要求。SOC 设计中利用总线的分层技术可以使各种不同特性的模块与总线更好地连接,提高总线的运行效率。

片上总线尚处于发展阶段,不像微机总线那样成熟,目前还没有统一的标准。各大厂商和组织纷纷推出自己的标准,以便在未来的 SOC 片上总线标准上占有一席之地,其中比较有影响的片上总线有 AMBA 系列总线、IBM 的 CoreConnect 总线、Wishbone 总线、Avalon 总线等。

5.7.1 Avalon 总线

Avalon 总线架构是 Altera 公司开发的片上总线架构。总的来说,Avalon 是一种相对简单的总线架构,主要用来将处理器和外围设备集成到片上可编程系统中,并规定了主设备和从设备的端口连接方式以及时序关系。Avalon 总线架构的基本设计目标如下:

(1) 简洁性,提供一种易于理解的协议;

(2) 低成本,为总线逻辑提供优化的资源,从而节约可编程逻辑资源;

(3) 同步性,基于同步操作,易于与片上的其他用户逻辑集成,避免了复杂的时序约束和分析过程。

Avalon 总线拥有多种传输模式,以适应不同设备的要求。Avalon 总线的基本传输模式是在主设备和从设备之间传输一个字。一次传输过后,总线可以立刻进行下一次传输,而且与上一次传输的目的设备和源设备无关。Avalon 总线还支持流水线传输、突发传输等模式。这些传输模式使得在一次总线传输中,在设备之间能够完成多个数据单位的交换。

Avalon 总线支持多个总线主设备,允许单个总线事务在设备之间传输多个数据单元。这一多主设备结构为构建 SOPC 系统提供了极大的灵活性,并且能适应高带宽的设备。例如,一个主设备可以进行直接存储器访问(DMA)传输,从设备到存储器传输数据时不需要处理器干预。

Avalon 总线主设备和从设备的交互采用了“从端仲裁”技术,在多个主设备试图访问同一个从设备时,用于决定哪个主设备获得访问权。这使其具有以下两个优点:

(1) 仲裁的细节被封装到 Avalon 总线内,主设备和从设备的接口与总线上设备数目无关。

(2) 多个主设备能够同时执行总线传输,只要它们不在同一时钟周期访问同一个从设备。

另外，Avalon 总线是为 SOPC 环境而设计的，整个总线的互连电路都由 FPGA 内部的逻辑单元实现。Avalon 总线具有以下基本特点：

(1) 所有设备的接口与 Avalon 总线时钟同步，不需要复杂的握手初应答机制，简化了 Avalon 总线的时序行为，而且便于集成高速设备。Avalon 总线以及整个系统的性能可以采用标准的同步时序分析技术来评估。

(2) 所有的信号都是高电平或低电平有效，便于信号在总线中高速传输。在 Avalon 总线中，由数据选择器代替三态缓冲器来决定哪个信号驱动哪个设备。因此，外设即使未被选中，也不需要将输出置为高阻态。

(3) 为了方便外设的设计，地址、数据和控制信号使用分离的、专用的端口。外设不需要识别地址总线周期和数据总线周期，也不需要在未被选中时使输出无效。分离的地址、数据和控制通道还简化了与片上用户自定义逻辑的连接。

常用 Avalon 从设备接口信号见表 5-2。

表 5-2　常用 Avalon 从设备接口信号

信号类型	宽度/位	方向	说明
clk	1	In	时钟
reset	11	In	复位
chipselect	11	In	片选
address	1～32	In	地址
read	1	In	读请求
readdata	1～32	Out	读数据
write	1	In	写请求
writedata	1～32	In	写数据
irq	1	Out	中断请求

5.7.2　Avalon 交换结构

1. 交换结构

Avalon 交换结构是一种在片上可编程系统中连接片上处理器和各种外设的互连机构。它定义了主、从节点之间通信的信号类型和时序关系，使得用户可非常方便地把自己选定或设计的外设模块通过 Avalon 总线连接到 Nios Ⅱ 系统上。Avalon 总线是构成 SOPC 的重要技术。在 SOPC Builder 添加外设时，Avalon 总线会自动生成，还会随着外设的增加和删减而自动调整。初学者可以不必关心 Avalon 总线的细节，但对于计划开发外设的用户来说，需要了解 Avalon 总线互连规范，以便设计的外设能够与 Avalon 总线协调工作。

Avalon 总线的设备分为主设备和从设备，并各有其工作模式。Avalon 总线本身是一个数字逻辑系统，在实现“信号线汇接”这一传统总线功能的同时，增加了许多内部功能模块，如从端仲裁模式、多主端工作方式和延迟数据传输等。

连接多个 Avalon 外设的 Avalon 总线系统如图 5-27 所示。

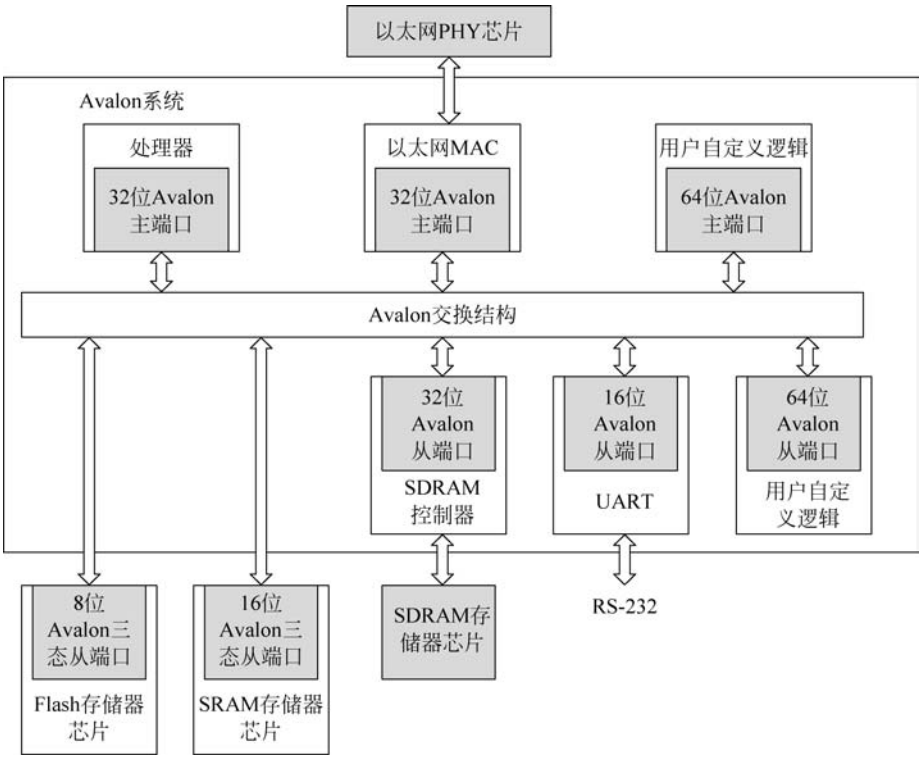


图 5-27　Avalon 总线系统

Avalon 交换结构将各个外设连接起来。Avalon 总线架构采用交换式架构，各个主机均有独立的总线，总线主机只需抢占共享从机而不是抢占总线，某一时刻多个主机可以与多个从机交换数据。

Avalon 总线设计的原则是操作简单，占用的逻辑资源也经过了优化，其接口信号全部和 Avalon 总线时钟同步。

Avalon 交换总线的特性包括：

(1) 简单的图形界面配置方式；

(2) 简化了片上系统的互连规则，提供一种易用、简单的接口规范；

(3) 在总线逻辑优化方面节省系统资源；

(4) 同时多个主设备的操作；

(5) 最大支持 4GB 的寻址空间；

(6) 同步接口；

(7) 内嵌地址译码功能；

(8) 延迟读写操作；

(9) 流传输；

(10) 动态外设总线宽度调整。

2. 图形界面配置

在使用 SOPC 构建系统的过程中，用户是看不到 Avalon 总线的具体实现的，只需要在 SOPC Builder 的库中选择系统需要的模块，包括 Nios Ⅱ处理器、各种外设以及用户自定义的逻辑模块，SOPC Builder 就会在图示中显示出 Avalon 总线的各种可能的连接关系，可以根据设计的实际情况选择连接。

如图 5-28 所示，在行列的连接线中，交叉点处表示可以创建连接关系，实心的点表示已连接上，空心的点表示尚未连接。

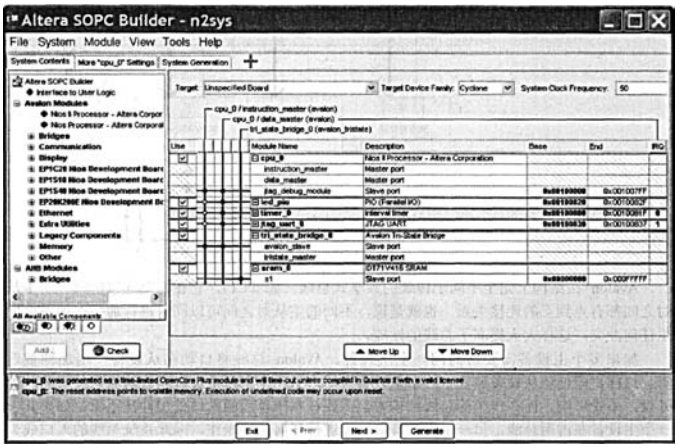


图 5-28　Avalon 总线连接

3. Avalon 总线功能

由于受 PCB 布线的限制，传统的 CPU 外部总线需要尽量控制互连线的数量，如采用三态数据总线方式。所有的外设和 CPU 共用这条总线，同时有许多信号线是复用信号，这将大大降低系统总线的吞吐性能。

Avalon 总线不同于传统的 CPU 外部总线。在 FPGA 内部，信号走线数量不再是设计的瓶颈，因此 Avalon 采用一种全交换功能的内嵌总线形式，如图 5-29 所示。

Avalon 在结构上完全不同于传统的共享式总线(如 PCI)，它在需要连接的每一个主、从对之间都有点到点的连接关系。也就是说，不同的主、从对之间可以同时进行通信，而不会发生任何冲突，这样大大提高了总线的性能。

如果多个主设备需要访问同一个从设备，Avalon 总线将自动在从设备一端加仲裁逻辑。这样，即使该从设备被其中一个主设备占用，另一个主设备可以同时访问系统中的其他外设，不受影响。在传统的总线结构中，需要在系统总线的入口处加仲裁逻辑，如果其中一个主设备在占用总线，另一个主设备就不能进行任何总线操作，因此系统总线的入口成为限制系统性能的一个严重的瓶颈(参考图 5-30 进行分析)。

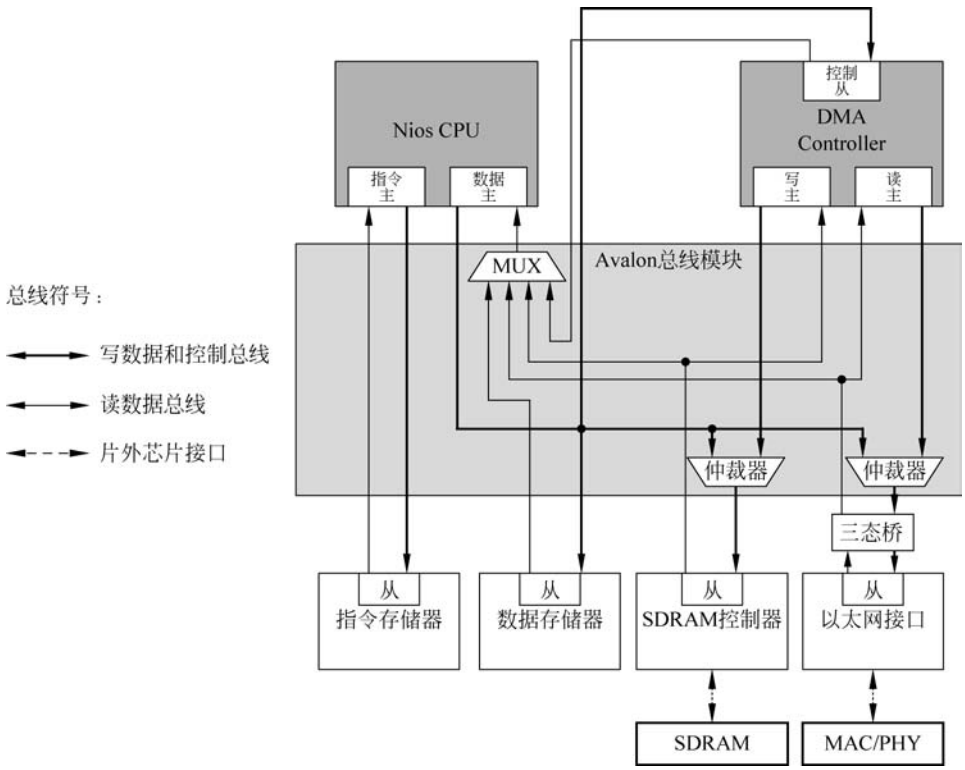


图 5-29　Avalon 总线结构

Avalon 总线为挂在其上面的外设提供如下服务：

(1) 数据通道复用：Avalon 结构中的多路器把所选中的外设数据传送到正确的主设备上。

(2) 地址译码：Avalon 中的地址译码逻辑为每一个外设产生一个片选信号，独立的外设内部就不需要对地址译码产生片选逻辑，简化了外设接口的设计。

(3) 流水线传送能力：挂在 Avalon 总线上的外设，如果知道具体访问的时延，可以发起连续的读操作，而不用等待第一个操作完成。

(4) 产生等待状态：为不能在一个时钟周期内响应的目标外设提供等待状态。

(5) 总线宽度动态调整：Avalon 总线会动态地适应不同接口数据或地址宽度的外设。

(6) 分配中断：多个不同从设备的中断源可以由 Avalon 总线传递到主设备中，通过中断信号来控制不同的中断优先级。

(7) 延迟传输模式：主、从设备之间的延迟传送模式使用的逻辑功能在 Avalon 总线内部已经包含。

(8) 流传输模式：主、从设备之间的流传输模式所使用的逻辑功能在 Avalon 总线内部已经包含。

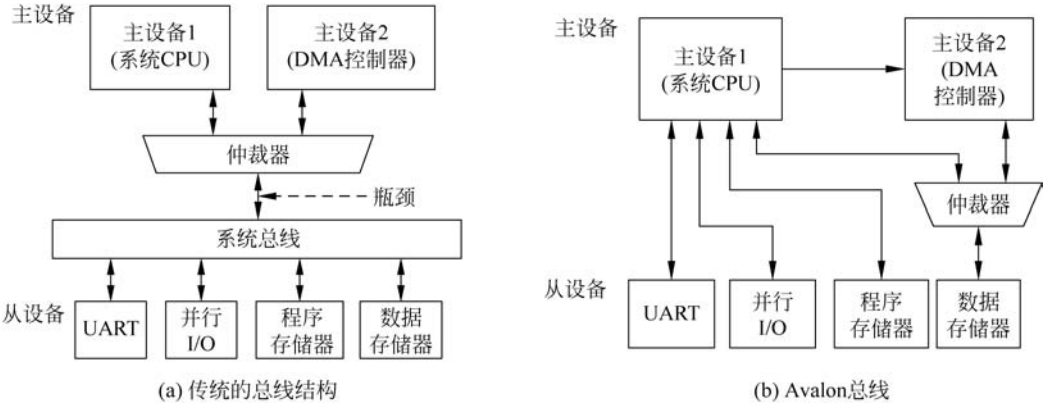


图 5-30　传统的总线结构和 Avalon 实现多主的总线操作的比较

5.7.3　Avalon 互连规范基本概念

Avalon 总线与传统的总线有显著的不同，它用的许多术语和概念是全新的，构成了 Avalon 总线规范的概念框架。为了更好地理解 Avalon 总线规范，有必要说明相关术语和概念，以免混淆。

1. Avalon 信号

Avalon 接口定义了一组信号类型（片选、读允许、写允许、地址、数据等），用于描述主、从外设上基于地址的读写接口。Avalon 外设只使用和其内核逻辑进行接口必需的信号，而省去其他会增加不必要开销的信号。

Avalon 信号的可配置特性是 Avalon 接口与传统总线接口的主要区别之一。Avalon 外设可以使用一小组信号来实现简单的数据传输，或者使用更多的信号来实现复杂的传输类型。例如，ROM 接口只需要地址、数据和片选信号，而高速的存储控制器可能需要更多的信号来支持流水线的突发传输。

Avalon 的信号类型为其他总线接口提供了一个超集，使大多数标准芯片的引脚都能映射成 Avalon 信号类型，从而使 Avalon 系统直接与这些芯片连接。例如，大多数分离的 SRAM、ROM 和 Flash 芯片上的引脚都能映射成 Avalon 信号类型。

2. Avalon 外设

Avalon 外设是 Avalon 存储器映射外设的简称，Avalon 外设包括存储器、处理器、UART、PIO、定时器和总线桥、用户自定义 Avalon 外设等。主外设能够在 Avalon 总线上发起总线传输，至少拥有一个 Avalon 主端口，从端口可选。从外设只能响应 Avalon 总线传输，不能发起总线传输，至少拥有一个 Avalon 从端口并且只能拥有 Avalon 从端口。

用户自定义 Avalon 外设，必须具有符合 Avalon 总线规范的 Avalon 信号。

3. 主端口和从端口

Avalon 端口分为主端口和从端口，主端口在 Avalon 总线上发起数据传输，从端口在 Avalon 总线上响应主端口发起的数据传输。一个 Avalon 外设可能有一个或多个主端口，一个或多个从端口，也可能既有多个主端口又有多个从端口。

Avalon 的主端口和从端口之间不是直接连接的，主、从端口都连接到 Avalon 交换架构上，由交换架构来完成信号的传递。在传输过程中，主端口和交换架构之间传递的信号与交换架构和从端口之间传递的信号可能有很大的不同，在讨论 Avalon 传输时必须区分主、从端口。

4. 总线传输

Avalon 总线传输是指对数据的一次读或写操作，发生在 Avalon 端口和系统互连结构之间。Avalon 端口一次可以在一个或多个时钟周期内传输 1024 位数据，传输完成后，在下一个时钟周期可以重新传输新数据。

Avalon 传输分为主传输和从传输。Avalon 主端口发起对交换架构的主传输，主端口只能执行主传输；Avalon 从端口响应来自交换架构的传输请求。传输是与端口相关的，主端口只能执行主传输，从端口只能执行从传输。

5. 主、从端口对

主、从端口对是指在数据传输过程中，通过 Avalon 交换架构相连接起来的主端口和从端口。在传输过程中，主端口的控制和数据信号通过 Avalon 交换架构和从端口进行交互。

6. 总线周期

总线周期是总线传输中的基本时间单元，其定义为从 Avalon 总线主时钟的一个上升沿到下一个上升沿之间的时间。总线信号的时序以总线周期为基准来确定。

7. 流传输模式

流传输模式是指在流模式主外设和从外设之间建立一个开放的通道，以提供连续的数据传输。只要存在有效数据，便能通过该通道在主、从端口对之间流动，主外设不必为了确定从外设是否能够发送或接收数据而不断地访问从外设的状态寄存器。流传输模式使得主、从端口对之间的数据吞吐量达到最大，同时避免了从外设的数据上溢或下溢，这对于 DMA 传输特别重要。

8. 延迟读传输模式

有些同步外设在第一次访问时需要几个时钟周期的时延，此后每个总线周期都能返

回数据。这样的延迟读传输模式可以提高带宽利用率。延迟传输使得主外设可以发起一次读传输，转而执行一个不相关的任务，等外设准备好数据后再接收数据。这个不相关的任务可以是发起另一次读传输，即使上一次读传输的数据还没有返回。

在取指令操作(经常访问连续地址)和 DMA 传输中，延迟传输是非常有用的。CPU 或 DMA 主外设会预取期望的数据，从而使同步存储器处于激活状态，并减少平均访问时间。

9. Avalon 总线模块

Avalon 总线模块是系统模块的主干，是 SOPC 设计中外设之间通信的主要通道。Avalon 总线模块由各类控制、数据和地址信号及仲裁逻辑组成，它将构成系统模块的外设连接起来。Avalon 总线模块是一种可配置的总线结构，它会随着用户的不同互连需求而改变。

从 Avalon 总线模块是由 SOPC Builder 自动生成的。因此，系统用户不需要关心总线与外设的具体连接。Avalon 总线模块很少作为分离的单元使用，因为用户几乎总是使用 SOPC Builder 自动将处理器和其他 Avalon 总线外设集成到系统模块中。对于用户来说，Avalon 总线模块通常可以看作是连接外设的途径。

5.7.4 Avalon 总线信号

由于 Avalon 总线是由一个 HDL 文件综合而来的，因此在连接 Avalon 总线模块和 Avalon 外设时需要考虑一些特别的问题。SOPC Builder 必须准确地了解每个外设提供了哪些 Avalon 端口，以便连接外设与 Avalon 总线模块；同时还需要了解每个端口的名称和类型，这些信息部在系统 ptf 文件中定义。

Avalon 总线规范不要求 Avalon 外设必须包含哪些具体信号，只定义了外设可以包含的各种信号类型(地址、数据、时钟等)。外设的每一个信号都要指定一个有效的 Avalon 信号类型，以确定这个信号的作用。信号也可以是用户自定义的。在这种情况下，SOPC Builder 不将该端口与 Avalon 总线模块连接。

Avalon 信号分为主端口信号和从端口信号。外设使用的信号类型首先由端口的主、从角色来决定。每个单独的主端口或从端口使用的信号类型由外设的设计人员决定。只有输出的 16 位 PIO 从外设如图 5-31 所示。

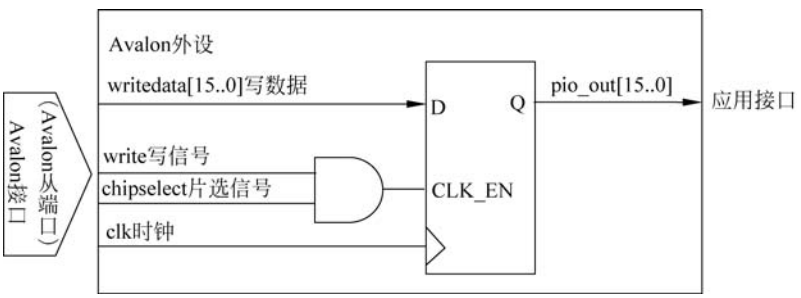


图 5-31　只有输出的 16 位 PIO 从外设

用户只需定义用于写传输(输出方向)的信号,而不需定义用于读传输的信号。尽管中断请求(IRQ)输出是从端口允许的信号类型,但也不一定是必须使用的。

1. Avalon 从端口信号

表 5-3 列出了 Avalon 从端口信号,其中信号方向以外设为参考。

表 5-3 Avalon 从端口信号

信号类型		宽度/位	方向	必需	说明
基本信号	clk	1	In	No	Avalon 从端口的同步时钟,所有的信号必须与 clk 同步,异步外设可以忽略 clk 信号
	chipselect	1	In	No	Avalon 从端口的片选信号
	address	1～32	In	No	连接 Avalon 交换架构和从端口的地址线,指定了从外设地址空间的一个字的地址偏移
	read	1	In	No	读从端口的请求信号。当从端口不输出数据时,不使用该信号。若使用了该信号,则必须使用 readdata 或 data 信号
	readdata(注)	1～1024	Out	No	读传输时,输出到 Avalon 交换架构的数据线。若使用了该信号,则 data 信号不能使用
	write	1	In	No	写从端口的请求信号。当从端口不从 Avalon 交换架构接收数据时,不需要该信号。若使用了该信号,必须使用 writedata 或 data 信号,writebyteenable 信号不能使用
	writedata(注)	1～1024	In	No	写传输时,来自 Avalon 交换架构的数据线。若使用了该信号,data 信号不能使用
	byteenable	2,4,6,8,16,32,64,128	In	No	字节使能信号。在对宽度大于 8 位的存储器进行写传输时,该信号用于选择特定的字节段。若使用了该信号,writedata 信号必须使用,writebyteenable 信号不能使用
	writebyteenable	2,4,6,8,16,32,64,128	In	No	相当于 byteenable 信号和 write 信号的逻辑与操作。若使用了该信号,writedata 信号必须使用,write 和 byteenable 信号不能使用
	begintransfer	1	In	No	在每次传输的第一个周期内有效,用法取决于具体的外设
等待周期信号	waitrequest	1	Out	No	若从端口不能立即响应 Avalon 交换架构,则用该信号来暂停 Avalon 交换架构
流水线信号	readdatavalid	1	Out	No	用于具有可变读延迟的流线读传输。该信号用于标记从端口发出的有效 readdata 时的时钟上升沿

续表

信号类型		宽度/位	方向	必需	说　明
突发信号	burstcount	2～32	In	No	用于突发传输。用来指示每一次突发传输中数据传输的次数。当使用 burstcount 信号时，waitrequest 信号必须一并使用
	beginbursttransfer	1	In	No	在突发数据传输的第一个时钟周期有效，标志突发数据传输开始。其用法取决于外设
流控制信号	readyfordata	1	Out	No	用于具有流控制的传输。表示外设准备好一次写传输
	dataavailable	1	Out	No	用于具有流控制的传输。表示外设准备好一次读传输
	endofpacket	1	Out	No	用于具有流控制的传输。向 Avalon 交换架构指示包结束的状态。实现取决于外设
三态信号	data	1～1024	In Out	No	三态从端口的双向数据读/写。若使用了该信号，则 readdata 和 writedata 不能使用
	outputenable	1	In	No	data 信号的输出使用能信号。若该信号无效，三态从端口不能驱动自身的 data 信号。若使用了该信号，则 data 信号必须使用
其他信号	irq	1	Out	No	中断请求信号。从外设的中断请求信号
	reset	1	In	No	外设复位信号。该信号有效时，从外设进入确定的复位状态
	resetrequest	1	Out	No	允许外设将整个 Avalon 系统复位。复位操作立即执行。

注：若从端口使用动态地址对齐，则信号宽度必须是 2 的整数次幂；若从端口同时使用 readdata 和 writedata 信号，则这两个信号宽度必须相等。

Avalon 总线规范不规定 Avalon 外设信号的命名规则。不同信号类型的作用是预先定义的，信号的名称由外设决定。信号可以按照它的信号类型来命名，也可以遵照系统级的命名规范。

表 5-3 中列举的信号类型都是高电平有效。Avalon 总线还提供了各个信号类型的反向形式。在 ptf 声明中，在信号类型名称后面添加“_n”，便可将对应的端口声明为低电平有效。这方便了许多使用低电平有效逻辑的片外外设。不论外设实现是在系统模块的内部还是在外部，Avalon 总线信号及操作都是一样的。在内部实现的情况下，SOPC Builder 自动将外设的主端口或从端口连接到 Avalon 总线模块。在外部实现的情况下，用户必须手工地将主端口或从端口连接到系统模块。在任何情况下，Avalon 总线信号的行为都是相同的。

2. Avalon 主端口信号

Avalon 主端口信号类型见表 5-4。

表 5-4 Avalon 主端口信号类型

信号类型		宽度/位	方向	必需	说明
基本信号	clk	1	In	Yes	Avalon 主端口的同步时钟，所有的信号必须与 clk 同步
	waitrequest	1	In	Yes	迫使主端口等待，直到 Avalon 交换架构准备好处理传输
	address	1～32	Out	Yes	从 Avalon 主端口到 Avalon 交换架构的地址线。该信号表示的是一个字节的地址，但主端口只发出字边界的地址
	read	1	Out	No	主端口的读请求信号。主端口不执行读传输时不需要该信号。若使用了该信号，则 readdata 或 data 信号线必须使用
	readdata	8，16，32，64，128，256，512，1024	In	No	读传输时，来自 Avalon 交换架构的数据线。当主端口不执行读传输时，不需要该信号。若使用了该信号，则 read 信号必须使用，data 信号不能使用
	write		Out	No	主端口的写请求信号。不执行写传输时不需要该信号。若使用该信号，则 writedata 或 data 信号必须使用
	writedata		Out	No	写传输时，到 Avalon 交换架构的数据线。当主端口不执行写传输时，不需要该信号。若使用了该信号，write 信号必须使用，data 信号不能使用
	byteenable		Out	No	启用字节信号。在对宽度大于 8 位的存储器进行写传输时，该信号用于选择特定的字节段。读传输时，主端口必须置所有的 byteenable 信号线有效
	writebyteenable		In	No	相当于 byteenable 信号和 write 信号的逻辑与操作。若使用了该信号，writedata 信号必须使用，write 和 byteenable 信号不能使用
	begintransfer		In	No	在每次传输的第一个周期内有效，用法取决于具体的外设
流水线信号	readdatavalid	1	In	No	用于具有延迟的流水线读传输。该信号表示来自 Avalon 交换架构的有效数据出现在 readdata 数据线上。若主端口采用流水线传输，则要求使用该信号
	flush	1	Out	No	用于流水线读传输操作。主端口置 flush 信号有效，以清除所有挂起的传输操作
突发	burstcount	2～32	Out	No	用于突发传输。用来指示每一次突发传输中数据传输的次数

续表

信号类型		宽度/位	方向	必需	说　明
流控制信号	endofpacket	1	In	No	用于控制流模式的数据传输。标志一个数据包的结束状态。实现取决于外设
三态信号	data	8,16,32,64,128,256,512,1024	In Out	No	三态主端口的双向数据读/写信号。若使用了该信号，则 readdata 和 writedata 不能使用
其他信号	irq	1,32	In	No	不断请求信号。若 irq 信号是一个 32 位的适量信号，则它的每一位直接对应一个从端口上的中断信号，它与中断优先级没有任何的联系；若 irq 是一个单比特信号，则它是所有从外设的 irq 信号的逻辑或，中断优先级由 irqnumber 确定
	irqnumber	6	In	No	只有在 irq 信号为单比特信号时，才使用 irqnumber 信号来确定外设的中断优先级。irqnumber 的值越小，代表的中断优先级越高
	reset	1	In	No	全局复位信号。实现与外设相关
	resetrequest	1	Out	No	允许外设将整个 Avalon 系统复位。复位操作立即执行

注意：若主端口同时使用 readdata 和 writedata 信号，则两个信号的宽度必须相等。

在这里，Avalon 从端口没有任何信号是必需的。而 Avalon 主端口必须有 clk、address、waitrequest 三个信号。Avalon 接口是一个同步协议，Avalon 主端口和从端口都与 Avalon 交换架构提供的时钟 clk 同步，同步不意味着所有的信号都是时序信号。Avalon 外设只对 clk 的边沿敏感，对其他信号的边沿不敏感。Avalon 接口没有固定的或最高的性能。

Avalon 接口是同步的，可以被交换架构提供的任意频率的时钟驱动。最高性能取决于外设的设计和系统实现。不同于传统的共享总线实现规范，Avalon 接口没有指定任何的物理和电气特性。所有的传输都与 Avalon 交换架构的时钟 clk 同步，并在时钟 clk 上升沿启动。

5.7.5　Avalon 的中断与复位信号

Avalon 接口提供系统级功能的控制信号，如中断请求信号和复位信号请求信号，这些信号与单个数据传输不直接相关。

1. 中断请求信号

Avalon 中断请求信号 irq 允许从端口设置 irq 有效，标志它需要主端口的服务。Avalon 交换架构在从端口和主端口之间传输 irq 信号。

(1) 从端口中断信号 irq：从端口可以包括 irq 输出信号，作为一个标志来指示外设逻辑需要主端口的服务。从端口能在任何时间设置 irq 有效，irq 的时序和传输没有任何关系。外设逻辑须保持 irq 一直有效，直到主端口复位中断请求。

(2) 主端口中断信号 irq 和 irqnumber：主端口包括 irq 和 irqnumber 信号，主端口能检测和响应系统中从端口的 irq 的状态。Avalon 接口支持两种方法来计算最高优先级的 irq。

① 软件优先计算：主端口包含 32 位的 irq 信号，不包含 irqnumber 信号；Avalon 交换架构将来自 32 个从端口的 irq 直接传递给主端口；在有多个位被同时置为有效的情况下，主端口(在软件控制下)决定哪个 irq 有最高的优先级。

② 硬件优先级计算：主端口包含 1 位的 irq 信号和 6 位的 irqnumber 信号；Avalon 交换架构将 irq 信号直接传递给主端口，同时将最高优先级 irq 的 irqnumber 信号发给主端口；在有多个从端口 irq 同时有效的情况下，Avalon 交换架构(硬件逻辑)识别最高优先级的 irq。

2. 复位控制信号

利用 Avalon 接口提供的信号，可以使交换架构复位外设，也可以使外设复位系统。

(1) reset 信号：Avalon 主端口和从端口可以使用 reset 输入信号。只要 Avalon 交换架构发出 reset 信号，外设逻辑必须复位自己到一个已定义的初始状态。Avalon 交换架构可以在任何时刻发出 reset，不管一个传输是否正在进行。reset 脉冲的宽度大于一个时钟周期。

(2) resetquest 信号：Avalon 主端口和从端口可以使用 resetquest 信号复位整个 Avalon 系统。发出 resetquest 导致 Avalon 交换架构对系统中的其他外设发出 reset。

5.8 SOPC 软件设计流程和方法

5.8.1 SOPC Builder 简介

SOPC Builder 是 Altera 公司提供的一个灵活、方便的系统设计工具，它利用一个组件库搭建基于总线的系统。用户从组件库中挑选所需的组件，配置组件参数，然后 SOPC Builder 自动生成总线互连逻辑，并将参数化后的组件实例连接起来，形成一个完整的可编程片上系统模块。SOPC Builder 可以快速地开发定制的方案，重建已经存在的方案，并为其添加新的功能，提高系统的性能。通过自动集成系统组件，SOPC Builder 允许用户将工作的重点集中到系统级的设计开发，而不是烦琐的组件装配工作。

SOPC Builder 提供了一个强大的平台，用于组建一个在模块级和组件级定义的系统。SOPC Builder 的组件库包含微处理器、内存接口、总线桥以及一些常用的外设接口等一系列的组件，用户还可简单地创建定制的 SOPC Builder 组件。

图 5-32 为 SOPC Builder 主界面。

从用户的角度来看，SOPC Builder 是一个能够生成复杂硬件系统的工具。从内部结构来看，SOPC Builder 包含图形用户界面和系统生成程序两个主要部分，如图 5-33 所示。

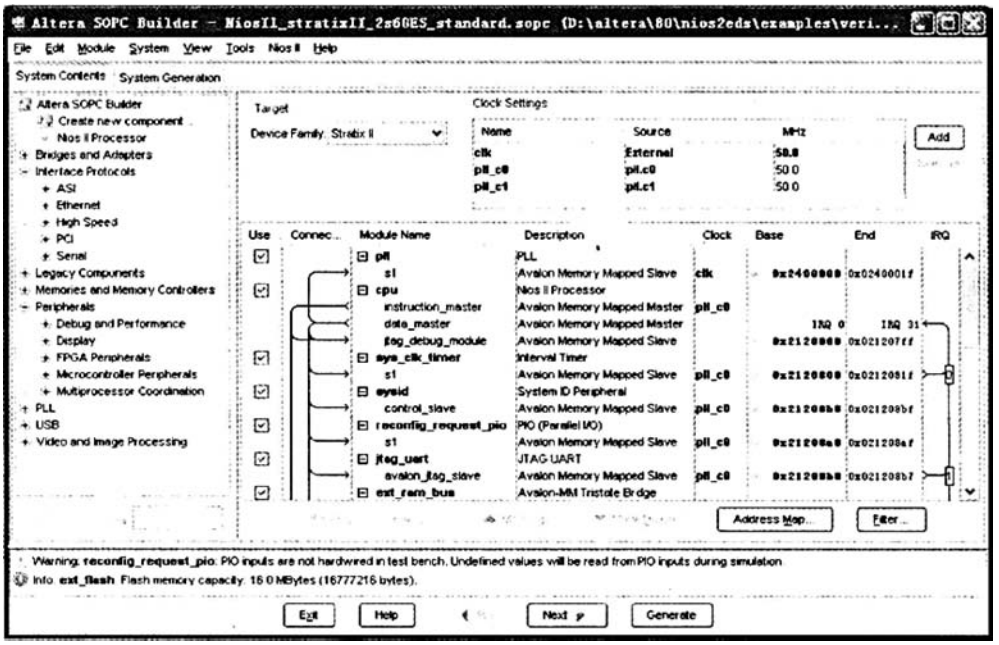


图 5-32　SOPC Builder 主界面

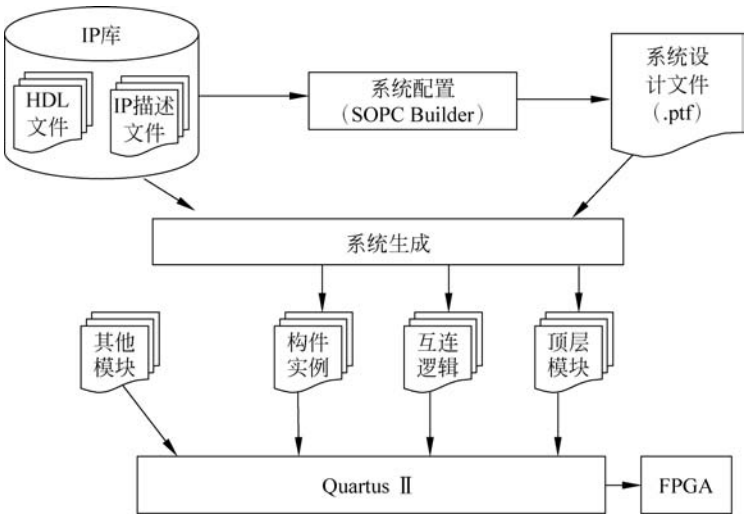


图 5-33　基于 Altera FPGA 的嵌入式系统硬件设计

SOPC Builder 用户图形界面提供管理 IP 模块、配置系统和报告错误等功能。用户通过图形界面设计系统时，所有的设置都保存在一个系统描述文件中。

用户通过 SOPC Builder 用户图形界面完成设计之后，单击图 5-32 中的 Generae 按钮，启动系统生成程序。系统生成程序执行了大量的功能，创建了几乎所有的 SOPC Builder 输出文件（HDL 代码、C 程序的头文件、库文件和仿真文件等）。

SOPC Builder 集成在 Quartus Ⅱ软件中，可以通过 Quartus Ⅱ的界面启动。SOPC Builder 生成的系统模块输出给 Quartus Ⅱ，然后用户利用 Quartus Ⅱ把所创建的 SOPC 系统与其他模块，如 PLL 等集成，经 Quartus Ⅱ编译后产生 FPGA 的配置文件。

5.8.2 SOPC Builder 设计流程

SOPC Builder 可看作是以 IP 库为输入、集成后的 SOPC 系统为输出的工具。SOPC Builder 设计过程主要包含三个步骤。

1. 组件开发

SOPC Builder 的 IP 模块包含由 IP 开发人员提供的硬件描述（如 RTL 代码、原理图或 EDIF）及其软件（如 C 源代码、头文件等）。高级 IP 模块可能还会包含一个相关的用户图形界面、生成程序和其他支持系统参数化生成的程序。在完成 IP 模块的硬件和软件实现后，需要利用 SOPC Builder 中的组件编辑器建立该 IP 模块的描述文件，从而把这个 IP 模块添加到 SOPC Builder 的 IP 模块库中。图 5-34 示出了组件编辑器界面。

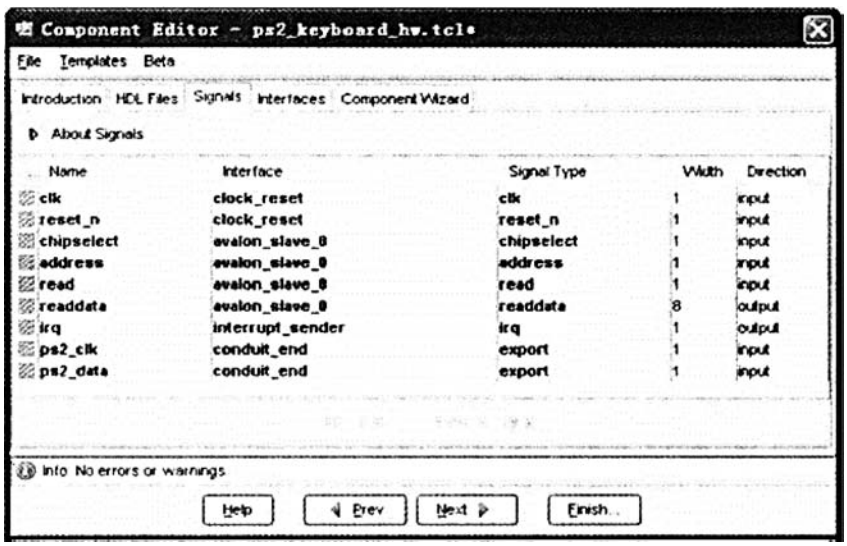


图 5-34 组件编辑器界面

2. 系统集成

用户创建和编辑一个新的系统时，首先要从库中选择一些 IP 模块，并逐个地配置这些 IP 模块，然后设置整个系统的配置。例如，指定地址映射和主/从端口连接等，如

图 5-35 所示。

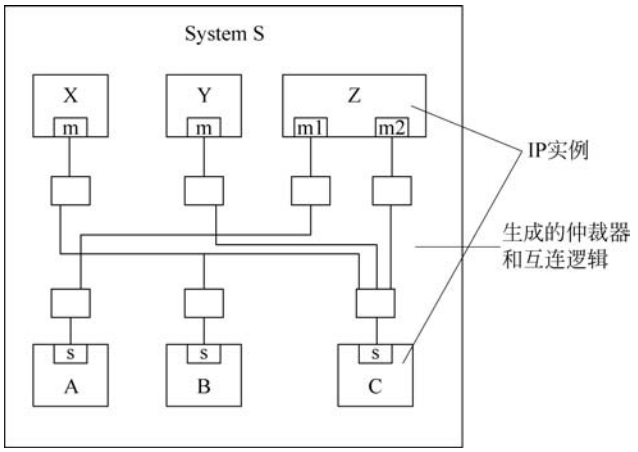


图 5-35 生成后的系统主模块实例

在这个过程中,用户的设置都会保存在系统描述文件中。

3. 系统生成

当用户在完成了 SOPC Builder 中的设计和配置之后,单击 Generate 按钮,或从命令行执行系统生成程序时,系统生成程序就开始运行。系统生成的结果是一系列设计文件,包括各模块的 HDL 代码实例、总线互连逻辑、系统顶层模块和仿真工程文件等。生成的系统模块可以在 Quartus Ⅱ 编译,并下载到 FPGA 中。

5.9 IP 核

IP 是设计中不可或缺的组成部分。随着数字系统越来越复杂,将系统中的每个模块都从头开始设计是一件十分困难的事,而且会大大延长设计周期,甚至增加系统的不稳定因素。IP 的出现使得设计过程变得十分简单,用户甚至只需要将不同的模块连接起来,就可以实现一个完整的系统。

可复用的 IP 核是 SOC 设计的基础。IP 核是预先设计、经过验证和优化的硬件模块,SOC 由多个 IP 核互相连接而成,因而 SOC 设计包括 IP 核设计与系统集成两个阶段。在嵌入式系统中,通常将直接或间接地通过总线与微处理器相连,并提供存储、输入或输出等功能的硬件模块或设备称为外围设备,简称外设。在 SOPC 设计中,大多数 IP 核都属于外设。

各类 SOPC 设计工具的组件库已提供了微处理器、内存接口、总线桥以及常用外设等 IP 组件。然而在实际的系统设计中,这些组件往往不能满足特定需要,还应加入用户自定义的外设。

在 SOPC 设计工具中一般提供了通用输入输出(GPIO)组件,通过一个或多个 GPIO

组件可以连接任意的硬件模块。在传统的嵌入式系统设计中，某个外设的接口和系统的总线接口不一致，只能通过 GPIO 组件间接地连接到总线上。然而，在系统设计中过多地采用这样的连接外设，会给软件开发带来很大负担。这种设计方法需要较多的软件代码与用户通信逻辑，软件的可读性也比较差。在 SOPC 设计中，利用 FPGA 的可编程特性，能够使用户在自定义外设中实现总线接口，直接与总线相连。它与采用 GPIO 的方式相比，减少了软件的工作量，增加了系统的可理解性与可维护性。更进一步来说，对于一个经常使用的外设，还可以将它加入组件库中，提高了外设的重用性。

为提高自定义外设的可重用性，在结构上通常将自定义外设分为两个功能模块：提供了外设基本功能的任务逻辑电路，以及为外设的数据输入、输出和对外设的控制提供标准的接口电路，它通常直接与总线相连。任务逻辑的设计取决于自定义外设要实现的功能。下面主要介绍接口电路的设计。

外设通过总线与处理器或其他外设通信。硬件模块间的直接相连可以自由定义数据线与控制线等接口信号，但总线连接与此不同。总线的接口信号是由总线协议规定的，特别是数据线的宽度是事先固定的，然而外设对外通信可能需要更多的接口信号，解决这一问题的常规手段是使用设备寄存器。

外设通常需要定义多个寄存器，如数据输入寄存器、数据输出寄存器、状态寄存器、控制寄存器等，每个寄存器的位数不超过数据总线的宽度。处理器可通过地址线的选择读写不同的寄存器，实现对外设的数据通信与控制。

外设往往还需要主动通知处理器某些事件的发生，这一般是通过中断信号实现的。中断信号分为电平中断和边沿中断两种方式。电平中断通过将中断信号置为高电平或低电平来通知处理器，处理器响应中断后需主动清除中断。边沿中断通过中断信号的上升沿或下降沿来通知处理器，外设通常产生一个脉冲信号，不需要处理器清除中断。外设具体使用哪一种中断方式取决于系统中使用的处理器类型，许多微处理器同时支持两种中断方式。

除了上述功能外，外设接口电路还可以包含输入/输出 FIFO 模块，用于实现处理器和外设间的速度匹配，以及 DMA 电路实现外设间的直接数据传输。图 5-36 是外设接口电路结构框图，显示了完整的接口功能。

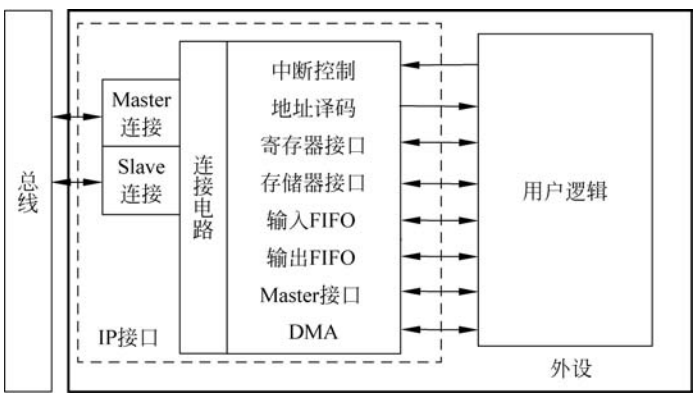


图 5-36　外设接口电路结构框图

5.9.1 IP的概念

美国Dataquest咨询公司将半导体产业的IP定义为用于ASIC、ASSP、PLD等芯片中的，并且是预先设计好的电路功能模块。

在可编程逻辑器件领域，IP核是指将一些在数字电路中常用但比较复杂的功能块（如FIR滤波器、SDRAM控制器、PCI接口等）设计成参数可修改的模块，让其他用户可以直接调用这些模块。

随着CPLD/FPGA的规模越来越大，设计越来越复杂，使用IP核是一个发展趋势。用户可以在自己的FPGA设计中使用这些经过严格测试和优化过的模块，减少设计和调试时间，降低开发成本，提高开发效率。

根据实现的不同，IP可以分为软IP、固IP和硬IP。

软IP用硬件描述语言的形式描述功能块的行为，但是并不涉及用什么电路和电路元件实现这些行为。软IP的最终产品基本上与通常的应用软件类似，开发过程与应用软件的开发过程类似，只是所需要的开发软、硬件环境要求较高，尤其是EDA工具软件很昂贵。软IP的设计周期短、设计投入少，由于不涉及物理实现，为后续设计留有很大的发挥空间，增大了IP的灵活性和适应性。软IP的缺点是设计中会有一定比例的后续工序无法适应软IP设计，从而造成一定程度的软IP修正。

固IP是完成了综合的功能模块，设计深度接近实际应用，以网表的形式提交客户使用。如果客户与固IP使用同一个生产线的单元库，IP的成功率就会比较高。

硬IP提供设计的最终阶段产品——掩模。随着设计深度的提高，后续工序所需要做的事情就越少，当然也伴随着灵活性下降。

Altera公司以及第三方IP合作伙伴给用户提供了许多可用的功能模块，它们基本可以分为免费的LPM宏功能模块（Megafunctions/LPM）和需要授权使用的IP知识产权（MegaCore）两类，二者的使用方法基本相同。

Altera LPM宏功能模块是一些复杂或高级的构建模块，可以在Quartus Ⅱ设计文件中和门、触发器等基本单元一直使用，这些模块的功能一般都是通用的，如Counter、FIFO、RAM等。Altera提供的可参数化LPM宏功能模块和LPM函数均为Altera器件结构做了优化，而且必须使用宏功能模块才可以使用一些Altera公司特定器件的功能，如存储器、DSP块、LVDS驱动器、PLL以及SERDES和DDIO电路。

知识产权模块是某一领域内的实现某一算法或功能的参数化模块（简称IP核）。专门针对Altera的可编程逻辑器件进行过优化和测试，一般需要用户付费购买才能使用。这些模块可以从Altera公司的网站（www.altera.com）上下载，安装后就可以在Quartus Ⅱ软件以及实际系统中进行使用和评估。用户对需用的IP核满意后，可以联系Altera公司购买使用授权许可。

Altera的IP核都是以加密网表的形式交给客户使用的，这就是前面所提到的固IP，同时配合一定的约束文件，如逻辑位置、引脚以及I/O电平的约束。

5.9.2 Altera IP 核

1. 基本宏功能

在 Altera 公司的开发工具 Quartus Ⅱ中，有一些内带的基本功能可供用户选用，如乘法器、多路选择器、移位寄存器等。这些基本的逻辑功能也可以由通用的硬件描述语言实现，但 Altera 公司提供的这些基本宏功能都是针对其实现的目标器件进行优化过的模块，它们应用在具体 Altera 公司器件的设计中，往往可以使用户的设计性能更高，使用的资源更少。此外，还有一些 Altera 公司器件特有的资源，如片内 RAM 块、DSP 块、LVDS 驱动器、PLL、DDIO 和高速收发电路等，同样是通过基本宏功能方式提供给用户使用的，用户只需要通过图形界面简单设置一些参数即可，而且不易出错。Altera 公司可以提供的基本宏功能见表 5-5。

表 5-5 Altera 公司提供的基本宏功能

类 型	说 明
算术组件	包括累加器、加法器、乘法器和 LPM 算术函数
门	包括多路复用器和 LPM 门函数
I/O 组件	包括时钟数据恢复(CDR)、PLL、双数据速率(DDR)控制器、千兆位收发器功能模块(GXB)、LVDS 接收器和发送器、PLL 重新配置和远程更改宏功能模块
存储器编译器	包括 FIFO 分配器、RAM 和 ROM 宏功能模块
存储组件	存储器、移位寄存器宏模块和 LPM 存储器函数

在实际使用过程中，一些简单的功能模块，如加/减、简单的多路器等，使用通用的 HDL 来描述。这样的逻辑功能用 HDL 描述非常简洁，而且综合工具可以把这些基本功能放在整个设计中进行优化，使得系统达到最优。如果使用 Altera 公司的基本宏功能，由于综合工具的算法无法对该模块进行基本逻辑的优化操作，反而会影响设计的结构。而对一些相对比较复杂的设计，如一个同步可载入的计数器，使用 Altera 公司的基本宏功能会得到较好的结果。

另外，在设计代码中过多地使用基本宏功能，也会降低代码的可移植性，这些都需要在实践中体会和积累。与设计工具自动从源代码中推断出逻辑功能块的方法相比，使用基本宏功能这个设计方法是否总能给设计者带来显著的性能提升和芯片面积节省，有时候还需要用户自己去实践和总结，不能一概而论。

2. Altera IP 核与 AMPP IP 核

Altera 公司除了提供一些基本宏功能以外，提供了一些比较复杂、相对比较通用的功能模块，如 PCI 接口、DDR、SDRAM 控制器等。这些就是 Altera 可以提供的 IP 库，也称为 MegaCore。主要可以分为四类，见表 5-6。

表 5-6　Altera 提供的复杂 IP 核

数字信号处理类	通信类	接口和外设类	微处理器类
FIR	UTOPIA2	PCI MT32	Nios&Nios Ⅱ
FFT	POS－PHY2	PCI T32	SRAMInterface
Reed Solomon	POS－PHY3	PCI MT64	SDR DRAM Interface
Virterbi	SPI4.2	PCI64	Flash Interface
Turbo Encoder/Decoder	SONET Framer	PCI32 Nios Target	UART
NCO	Rapid IO	DDR Memory I/F	SPI
Color Space Converter	8B10B	HyperTransport	Programmable IO
DSP Builder			SMSC MAC/PHY I/F

另外，Altera 公司的合作伙伴 AMPP 也向 Altera 公司的客户提供基于 Altera 器件优化的 IP 核。

Altera 公司或 AMPP 的 IP 核具有统一的 IP Toolbench 界面，用来定制和生成 IP 文件。所有的 IP 核可以支持功能仿真模型，绝大部分 IP 核支持 OpenCorePlus。也就是说，用户可以免费在实际器件中验证所用的 IP 核（用户必须把所用器件通过 JTAG 电缆连接到 PC 上，否则 IP 核电路不会工作），直到用户觉得没有问题，再购买 IP 的使用授权许可。

在使用 Altera 公司或 AMPP 的 IP 核时，一般的开发步骤如下：

（1）下载所要 MegaCore 的安装程序并安装；

（2）通过 MegaWizard 的界面打开 IP 核的统一界面 IP Toolbench；

（3）根据用户的需要定制要生成 IP 的参数；

（4）生成 IP 的封装和网表文件，以及功能仿真模型；

（5）用户对 IP 的 RTL 仿真模型进行功能仿真；

（6）用户把 IP 的封装文件和网表文件放在设计工程中，并实现设计；

（7）如果 IP 支持 OpenCorePlus，用户就可以把设计下载到器件中进行验证和调试；

（8）如果确认 IP 使用没有问题，就可以向 Altera 或第三方 IP 供应商购买使用授权许可。

3. MegaWizard 管理器

为了方便用户使用宏功能模块，Quartus Ⅱ 软件为用户提供了 MegaWizard Plug-In Manager，即 MegaWizard 管理器。它可以帮助用户建立或修改包含自定义宏功能模块变量的设计文件，然后可以在用户自己的设计文件中对这些 IP 模块文件进行实例化。这些自义宏功能模块变量基于 Altera 公司提供的宏功能模块，包括基本宏功能、MegaCore 和 AMPP 函数。MegaWizard 管理器运行一个向导，帮助用户轻松地为自定义宏功能模块变量指定选项，生成所需要的功能。

5.9.3 Altera IP 核在设计中的作用

为了提高设计速度，Altera 公司建议尽量使用 IP 模块，而不是对用户自己所有逻辑模块从头开始编码。与传统的 ASIC 器件或者用户自己设计的模块相比，使用 Altera 公司的 IP 来设计项目具有以下优势。

1. 提高设计性能

IP 模块可以提供更有效的逻辑综合和器件实现，所有的 IP 模块都经过严格的测试和优化，从而使得 IP 模块可以在 Altera 公司的可编程逻辑器件中达到最好的性能和最低的逻辑资源使用率，用户只需要通过设置参数即可方便地按需定制自己的宏功能模块。

2. 降低产品开发成本

Altera 公司 IP 模块销售量大，采用了世界领先的封装技术和加密技术，因此，Altera 公司的大部分 IP 模块的价格大约只有市场上相同功能的 ASIC 器件的 1/5，极大地降低了基于 IP 的 FPGA 产品的开发成本。

3. 缩短设计周期

IP 模块经过供应商的严格测试和验证，并且已经封装完毕，用户只需要在自己的设计中实例化 IP 模块即可。因此，用户使用 IP 模块可以避免重复设计标准化的功能模块，缩短产品的研发周期。同时，也可以将精力集中于系统顶层与关键功能模块的设计上，致力于提高产品整体性能和个性化特性。

4. 设计灵活性强

IP 模块的参数是可变的，这样用户可以按照设计的需要定制自己的 IP 模块。在使用 Altera 公司的 IP 时，用户可以通过 Quartus Ⅱ 提供的 MegaWizard 管理器启动 IP Toolbench，通过一个直观的图形用户界面来为他们的设计输入 IP 模块的各种参数，包括不同的器件类型、不同的性能和面积的均衡等。

5. 便于仿真

与 ASIC 器件不同，使用 IP 可以在 Altera 公司的 FPGA 开发工具 Quartus Ⅱ 中对设计进行功能和时序仿真，并且仿真相当可靠。

IP Toolbench 可以为任何参数化的 IP 模块生成行为仿真模型文件(testbench)，每一个行为仿真模型是 Quartus Ⅱ 软件生成的 VHDL 和 Verilog HDL 文件。通过工业标准的 VHDL 或 Verilog HDL 仿真工具可以对这些 IP 的模型进行快速的功能仿真。用户还可以使用 Quartus Ⅱ 软件内置的简单易用的仿真环境，或者第三方 HDL 仿真工具，

对 IP 模块进行布局布线后的门级时序仿真。

6. OpenCorePlus 支持无风险应用

使用 ASIC 来完成用户的设计时，用户必须首先购买 ASIC 器件，然后在单板上进行硬件调试。Altera 公司针对自己的 IP 提供了 OpenCore Plus Evaluation 功能，此功能允许用户在购买 IP 模块之前，首先仿真和验证集成了 IP 模块的设计的正确性、评估设计的资源使用率以及时钟速度。甚至无须授权许可，用户就可以在 Quartus Ⅱ 软件中为集成了 IP 模块的设计生成有限制的下载文件，从而可以使得用户在购买 IP 许可前将设计下载到 FPGA 器件中(需要将 JTAG 下载电缆一直连接在芯片上)，在硬件系统中充分验证，并测试 IP 模块的性能。当用户对 IP 模块的功能与性能完全满意，并准备将设计投入生产时，用户再买 IP 模块的许可使用权，从而可以将设计下载到器件中无限制地使用。

5.9.4 使用 Altera IP 核

IP Toolbench 是 Altera 公司的 IP 定制工具，可以帮助用户快速简单地查看 IP 模块信息、输入模块的参数、设置第三方 EDA 工具、生成输出文件，供仿真和实现工具使用。

1. 使用方法

Altera 公司的所有核的安装文件都可以在 www.altera.com 网站上免费下载。下载的文件为扩展名为.exe 的安装文件，安装完成后，在 MegaWizard 管理器的宏功能函数选择窗口中就会出现可供选择的选项。填写基本的需求参数后，下一步就进入 IP Toolbench 窗口。执行“About this core”功能，会出现一个显示该 IP 核基本信息的页面，包括版本信息、支持的器件等。接着根据需要设置 IP 核的模式参数、生成的仿真模型等参数，即可生成 IP 核文件。生成的 IP 核文件包括封装文件、加密网表文件、仿真模型和仿真向量文件等。

2. 实现 IP 核

1) 功能仿真

使用 IP Toolbench 为所用 IP 核生成一个仿真激励文件(testbench)，扩展名为.v，也可以自己编写一个 testbench 文件。对相应的扩展名为.vo(Verilog)或扩展名为.vho(VHDL)的 IP 核仿真模型文件，仿真激励文件产生各种各样的激励，送给仿真模型文件，然后观察仿真模型文件的输出是否正确。

.vo 和.vho 文件是一个只能用于仿真的模型，不能用来综合实现，可以使用第三方的 HDL 仿真工具来仿真这个模块。

2) 综合实现

如果用户使用第三方综合工具综合该 IP 模块，首先需要在设计中实例化该 IP 核，然后把仿真模型文件加入综合工程中(作为一个黑盒)。综合生成整个工程的网表文件后，

在 Quartus Ⅱ 工具中实现时，需要把该网表加入工程中。同时，必须把 IP 核的封装文件和其中实例化的加密网表文件加入工程中。如果用户在 Quartus Ⅱ 中综合与实现，直接把 IP 核的封装文件和加密的网表文件加入工程中即可。

3）仿真

用户可以在布局布线之后，在 Quartus Ⅱ 中做门级的时序或功能仿真，也可以由 Quartus Ⅱ 工具在布局布线完成之后，输出一个可以用在第三仿真工具中的仿真模型和时延文件。

4）验证与调试

如果该 IP 核支持 OpenCorePlus，那么用户可以免费将其下载到芯片中去验证，只要主机上的加载电缆连在芯片上，该 IP 就可以持续工作，直到用户拔去电缆，IP 核随即停止工作。

Altera 公司的 IP 核生成器称为 MegaWizard，Xilinx 公司的 IP 核生成器称为 Core Generator，Lattice 的 IP 核生成器称为 Module/IP Manager。另外，可以通过在综合、实现步骤的约束文件中约束属性来完成时钟模块的约束。Altera 公司的 Stratix、Stratix GX、Stratix Ⅱ 等器件族内部集成了 DSP 核，配合通用逻辑资源，还可以实现 ARM、MIPS、NIOS 等嵌入式处理器系统。