## 摘 要

当今社会，嵌入式系统的应用越来越广，需求越来越大。传统的嵌入式系统开发方法不利于修改设计，开发成本高，设计周期长，不能满足激烈的市场竞争的需要。一种新型的嵌入式系统设计方法——快速样机生成技术应运而生。使用SOC 技术可以进行快速样机设计。在 SOC 基础上，Altera 公司推出基于 NIOS 的 SOPC 设计。使用基于 NIOS 的 SOPC 技术，可以快速地进行嵌入式系统的设计，并且开发的系统易于修改，开发成本低，开发周期短。

SOPC 是 Altera 公司提出的一种灵活、高效的片上系统设计方案。它将处理器、存储器、I/O 口等系统设计需要的组件集成到一个 PLD 器件上，构建成一个可编程的片上系统。NIOS 是 Altera 公司开发的可进行 SOPC 设计的 RISC 型处理器软核。进行 SOPC 设计的环境是 SOPC Builder，集成在 Altera 公司开发的 EDA 工具 Quartus II 里。进行 SOPC 设计的关键是使用 IP 核。Altera 公司除提供自己开发的 IP 核外，也提供了使用第三方 IP 核的简单接口。

本文首先对嵌入式系统设计做了简要介绍。第二章详细介绍了 Altera 公司推出的基于 NIOS 的 SOPC 技术，包括对 NIOS 软核的介绍和对 SOPC 技术的介绍。第三章介绍了复旦大学 CAT 实验室研制的快速样机平台，介绍了样机平台的主要组成部分及样机平台的特性。第四章介绍了本人对样机平台进行的 A/D、D/A 功能扩展的工作；第五章介绍了在样机平台上进行的应用开发，汽车运行信息记录仪。文章最后对本文进行了一下总结，并对实验室今后的工作做了展望。



关键词：嵌入式系统，SOC，NIOS，SOPC

**Abstract**

Nowadays the embedded system is becoming more and more popular and its applications can be found in many aspects of industry. In the heated competition the traditional developing method for embedded system is being driven out because of its drawbacks: hard to debug, relatively higher cost and long developing cycle and so on. Meanwhile, a new developing method is coming into being—Rapid Prototyping Technology. By using SOC technology a rapid prototyping design can be made. NIOS based SOPC design is developed by Altera Corporation. By using this technology, embedded system can be designed in a rapid way and can be easily modified, the cost can be reduced and the developing circle can be shortened.

SOPC is a flexible、efficient SOC design methodology, which integrates processor、memory、I/Os and other peripherals into a PLD. NIOS is a soft core microprocessor, which can be used to build a SOPC design. The SOPC design environment is SOPC Builder, which is integrated in the EDA tool Quartus II. The key issue of SOPC design is IP cores. Altera provides not only IP cores but also the interface to make possible to use IP cores developed by the third party.

In chapter one, the embedded system design is introduced briefly. In chapter two, NIOS based SOPC design is introduced in detail, including the introduction of NIOS soft core and SOPC technology. The rapid prototyping platform, which is developed by CAT lab of Fudan University, is introduced in chapter three. In chapter four, the function of A/D、D/A conversion expanded to the platform is introduced in detail. In chapter five, the application— Vehicle Information System that is developed based the platform of CAT lab is presented. At the end of this article, a summing up of this article will be given and the expectation of the future research work of CAT lab is put forward.

Key words: embedded system, SOC, NIOS, SOPC

# 第一章 引言

## 1.1 嵌入式系统概述

数字系统（digital system）可以分为两种：通用系统（general-system）和专用系统（special-purpose system）。通用系统就是传统的计算机，范围从 PC 机，手提电脑，工作站，一直到超级计算机。这些计算机的特点是用户可以在其上进行编程，并且通过可执行的软件，可以支持很多不同的应用。在 80 年代及 90 年代早期，通用系统是设计方法革新的主要推动力。然而，随着几种新型应用的产生，比如 MPEG4，通用计算机不能高效地实现这些应用，上述情况有了改变，专用系统变得越来越重要，因为它们是专门为实现特定应用而开发的。嵌入式系统就属于这类系统。

一般来说，嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可以被剪裁以适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。在嵌入式系统中，操作系统和应用软件集成于计算机硬件系统中，即系统的应用软件与系统的硬件一体化。嵌入式系统具有软件代码少、高度自动化、响应速度快等特点。

如今，嵌入式系统的广泛应用，已经渗入到我们日常生活的各个方面。在手机、电视机、数码相机、洗衣机、电冰箱、空调，甚至电饭锅、手表里，都有嵌入式系统的身影。嵌入式系统小到一个芯片，大到一台标准的 PC 板或者一台独立的设备，种类繁多，让人顿生目不暇接之感。微型计算机虽然占据了全球计算机工业的 90%的市场，但事实上，嵌入式系统在数量上远远超过了各种通用计算机。PC 机的各种输入输出和外部设备均是由嵌入式系统控制的。每台 PC 的外部设备中包含了 5～10 个嵌入式处理器。当今工业界的自动化、信息化和网络化已经发展成为一种不可逆转的趋势。在工业自动化控制、通信、仪器仪表、汽车、船舶、航空航天、军事装备、消费类电子产品等领域更是嵌入式系统的天下。

一般来说，嵌入式系统具有如下一些特征[1]：

1. 通常嵌入在其他产品中。
2. 独立工作。
3. 通常不被重新编程，它们的功能很固定。
4. 通常在反应模式下工作，对于外部的输入要频繁地做出反应。
5. 由大量并发的工作进程实现，这些进程之间需要通信。
6. 有严格的时间要求，比如实时限制。
7. 常常是 I/O 密集的。
8. 对于成本、功耗、体积、重量等性能指标有一定的约束。
9. 有严格的可靠性、正确性限制，比如 ABS（Anti-lock Brake System）必须在

任何情况下都可以无错工作。

上面的特征只是嵌入式系统的一般特征，并不是所有嵌入式系统都要具有上面的特征。实际上，不同的嵌入式系统在设计要求上是有一定差异的。例如，某些系统对实时性要求十分严格，如果系统在规定的时间内不能对输入做出适当的反应，则会导致严重的错误甚至灾难性的后果，比如跟踪导弹的雷达系统。这类系统称为强实时或者硬实时嵌入式系统。相对的，有些系统虽然对实时性有一定要求，但并不十分严格，即使在规定时间内对于输入没能及时做出反应，也不会造成重大损失，比如 ATM 提款机。这类系统称为软实时嵌入式系统。

嵌入式系统又可以被分成两个子类：嵌入式控制器和嵌入式数据处理系统。

嵌入式控制器是专用于控制功能的。它们是面向控制流的（data flow dominated），对于外部事件起反应。因此，嵌入式控制器常被称作反应系统（reactive system）。反应系统对于环境的刺激所做出的反应是改变其内部状态和产生输出结果。通常地，它们支持一组模式和设置，并且它们的实时限制一般在毫秒范围内。因此，性能需求通常是低等到中等。由于这个原因，微控制器对于实现嵌入式控制器就足够了。嵌入式控制器典型的应用在以下方面：

1．控制器（例：电梯，电子窗帘）

2．家用电器（例：微波炉，洗衣机）

3．汽车工业应用（例：发动机控制单元，燃料喷射器，防锁刹车（anti-locking brakes））

4．工业机器人

嵌入式数据处理系统专用于数据通信和处理。因此，它们常被称为转换系统（transformational systems）。这些系统是面向数据流的，通常是实时系统，要在一个预先定义的时间窗内执行一个特定的功能。与嵌入式控制器相比，它们需要更高的性能。因此，微控制器已经不能满足要求，需要更强大的微处理器（microprocessors）和 ASICs。数字信号处理和高层综合是它典型的应用领域。以下应用领域包含了典型的嵌入式数字处理系统：

1．多媒体（multi-media）

2．消费电器（consumer electronics）

3．无线通讯（wireless communication）

4．通用电讯（general telecommunication）

## 1．2 嵌入式系统设计现状与前景

传统的嵌入式系统的开发过程是一种“瀑布”式[2]（Waterfall Mode）的设计过程。这种设计方法的最大特点是从系统设计的开始就将系统所要实现的功能划分到用硬件实现或者用软件实现，然后分别进行硬件设计和软件设计，最后进行

硬件和软件的集成。这种设计方法不利于修改设计，而且成本相对较高，设计周期较长，不能满足激烈的市场竞争的要求。

为了解决上面的问题，近年来，一些基于硬/软件协同设计的嵌入式系统自动化设计技术，即所谓的快速样机系统（Rapid Prototyping System），成为嵌入式系统领域的研究热点。这种嵌入式系统的设计方法不同于传统的嵌入式系统的设计方法，最主要的有两点：一是硬件和软件的设计不是各自独立进行设计的，而是从系统级设计到最后的综合均采用硬/软件协同设计的方法。二是系统设计是从与具体实现无关的系统级开始的。采用系统级设计的好处是一方面可以通过对所设计系统的功能和性能进行验证，及时发现和纠正设计中的错误和问题，否则到对系统进行综合实现时再发现错误就将对整个设计付出很大的代价；另一方面，由于系统级设计与具体实现无关，所以即使今后的硬/软件划分方法、协同综合方法、快速样机平台发生变动，都不会对系统的设计造成任何影响。

除设计方法外，嵌入式系统设计领域还有其他一些研究热点：

1．处理器

为了满足不同的应用，半导体厂商提供一些具有不同处理能力及寻址能力的处理器，如 8 位、16 位、32 位处理器。为了专门进行数字信号处理，数字信号处理器 DSP（Digital Signal Processor）也应运而生。另外，还有基于知识产权核 IP Core（Intellectual Property Core）的、用户可以进行修改的处理器——软核（Soft Core）处理器。本文后面将要讨论到的 NIOS 就是一种软核处理器。

2．存储器

随着 ROM 和 RAM 存储设备存储容量的不断提高及价格的不断下降，在嵌入式系统中可以存储大量的数据及程序。另外，随着闪存的技术和容量的提高，很多嵌入式系统使用闪存作为存储设备，这使得嵌入式系统可以包含“巨大”的操作系统。

3．操作系统

现在，很多嵌入式系统使用操作系统。这种操作系统与桌面操作系统相比，比如 Windows 系列操作系统，它们占用的存储空间更小。但这些操作系统也提供应用程序接口 API（Application Programming Interface），这使得嵌入式系统开发人员可以在操作系统层面进行开发，减少了开发时间与付出的努力。对嵌入式系统的操作系统进行相应的更改与裁减，使其适合嵌入式系统的要求，这也是现在嵌入式系统设计领域中的一个研究热点。

4．编程语言

以前人们使用汇编语言来编写嵌入式系统软件，但现在人们更多的使用的是高级语言来进行嵌入式系统的开发，比如 C 语言。另外，随着面向对象技术应

用越来越广，一些面向对象的语言也被用来进行嵌入式系统的开发，如C++语言。另一个重要的发展是Java的使用。由于Java的平台无关性，它在嵌入式系统软件开发中变得非常流行。由Java编译器生成的字节代码能够被移植到任何平台上，只要该平台上运行了Java虚拟机JVM（Java Virtual Machine）。

5．开发工具

众多电子设计自动化EDA（Electronic Design Automatic）开发工具的发展正在加速嵌入式系统设计的开发。这些开发工具一般包括交叉编译器、调试器、仿真器等。利用这些工具，开发人员可以在主机上编写程序，测试软件，最后移植到目标硬件上。

## 1．3 后续章节的安排

本文主要介绍一个快速样机生成技术，基于NIOS的SOPC技术。在后续的章节里，内容安排如下：

第二章：具体介绍基于NIOS的SOPC技术。2.1节对SOC技术进行简要介绍；2.2节对基于NIOS的SOPC技术进行详细介绍。2.3节介绍SOPC的开发过程；2.4节介绍本人对SOPC的高级特性的研究。

第三章：介绍复旦大学CAT实验室研制的快速样机平台。其中3.1节对复旦大学CAT实验室研制的快速样机平台做概要介绍；3.2节介绍这个样机平台的特性。

第四章：详细介绍了本人对样机平台进行的功能扩展——A/D、D/A转换板的设计。

第五章：介绍在复旦大学CAT实验室开发的快速样机平台上开发的一个应用实例——汽车运行信息记录仪。其中，5.1节对汽车运行信息记录仪做了简要介绍；5.2节介绍了汽车运行信息记录仪的具体实现；5.3节介绍了系统模拟运行的情况。

第六章：对本文进行总结，同时对复旦大学CAT实验室在快速样机生成技术研究方面的今后工作做一下展望。

# 第二章 基于NIOS的SOPC设计

## 2.1 SOC概述

随着大规模集成电路设计技术的进步和制造工艺水平的提高，单个芯片上的逻辑门数的增加，嵌入式系统设计变得日益复杂。与此同时，上市时间的压力也越来越大，传统的嵌入式系统设计方法已不能适应当前嵌入式系统设计的需要。单个芯片上的逻辑门数的增加在使设计任务复杂的同时，也为设计人员的开发设计开辟了新的天地——可以把整个系统集成到一个芯片上，这就是所谓的SOC技术，即片上系统（System On Chip）。使用SOC技术可以快速地进行嵌入式系统设计，从这点上来讲，SOC技术属于快速样机生成技术。

虽然SOC一词多年前已经出现，但到底什么是SOC则有不同的说法。在经过了多年的争论之后，专家们就SOC的定义达成了一致意见。这个定义虽然不是非常严格，但明确地表明了SOC的特征[3]：

1. 采用复杂系统功能的VLSI；
2. 采用超深亚微米工艺技术；
3. 使用一个以上嵌入式CPU/数字信息处理器（DSP）；
4. 外部可以对芯片进行编程；
5. 主要采用第三方IP进行设计。

从上述SOC的特征来看，SOC中包含了微处理器/微控制器、存储器以及其他专用功能逻辑，但并不是包含了微处理器、存储器以及其他专用功能逻辑的芯片就是SOC。SOC技术被广泛认同的根本原因，并不在于SOC可以集成多少个晶体管，而在于SOC可以用较短时间被设计出来。这是SOC的主要价值所在——缩短产品上市周期。因此，SOC更为合理的定义为：SOC是在一个芯片上由广泛使用预定制模块IP而得以快速开发的集成电路。从设计上来说，SOC就是一个通过设计复用达到高生产率的硬/软件协同设计过程。从方法学的角度看，SOC是一套极大规模集成电路的设计方法学，包括IP核可复用设计/测试方法及接口规范、系统芯片总线式集成设计方法学、系统芯片验证和测试方法学。

为了快速设计生产出SOC产品，设计人员必须利用预先定义并验证好的IP核。IP 核是SOC设计的关键技术之一。虽然IP核一词在众多场合被使用，但它并没有一个统一的定义。从概念上可以这样理解它：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（Finite Impusle Response）滤波器，SDRAM控制器，PCI接口等等设计成可修改参数的模块，让其他用户可以直接调用这些模块，这样就大大减轻了工程师的负担，避免重复劳动。IP 核设计一般可以有三个来源，一是EDA厂商提供，二是用户自己定义，三是第三方提供。

IP 核一般可以分为三类[4]：

1．软核（soft-core）

软核用硬件描述语言写成，可以是对设计的算法级描述，或功能级描述，也可以是仅仅用于功能仿真的行为模拟。

2．固核（firm-core）

固核在软核基础上开发，是一种可综合的、并带时序信息及布局布线规划的设计，用硬件描述语言写成。

3．硬核（hard-core）

指和特定工艺相联系的物理版图，设计的正确性已经投片验证，可以在新设计中作为特定的功能模块直接调用。

## 2．2 基于 NIOS 的 SOPC 设计

### 2．2．1 NIOS 软核

#### 2．2．1．1 NIOS 软核概述

Altera 是世界知名的可编程逻辑器件公司。过去，Altera 作为可编程逻辑器件供应商，他提供 PLD 器件。如今，Altera 同时又是系统方案的供应商。他现在为客户提供的服务是，如何把一个系统所需要的嵌入式功能块，也就是 IP 核，放到可编程逻辑器件上组成系统，这就是 SOPC（System On Programmable Chip）技术。

在进行 SOPC 设计时，最重要的一个 IP 核是一个 CPU 的软核，这就是 Altera 公司自行研制的 NIOS CPU 软核[5]。NIOS CPU 软核是一种采用流水线技术、单指令流的 RISC 处理器，如图 2.1 所示，其大部分指令可以在一个时钟周期内完成。NIOS 软核处理器家族包括 32 位和 16 位两种体系结构的版本，见表 2.1。这里主要介绍 32 位的 NIOS CPU，并规定字节宽度为 8 位，半字宽度为 16 位，字宽为 32 位。

表 2.1

| NIOS CPU 规格 | 32 位 NIOS CPU | 16 位 NIOS CPU |
|---|---|---|
| 数据总线宽度（位） | 32 | 16 |
| ALU 宽度（位） | 32 | 16 |
| 内部寄存器宽度（位） | 32 | 16 |
| 地址总线宽度（位） | 32 | 16 |
| 指令宽度（位） | 16 | 16 |

NIOS CPU 采用 16 位指令系统，其指令集有以下特点：

- 拥有较大的窗口化的寄存器文件

NIOS CPU 包含 512 个内部通用寄存器，编译器使用这些寄存器来加速子程序调用和本地变量的访问。

- 简单完整的指令集

32 位和 16 位的 NIOS CPU 都使用 16 位宽的指令，减小了代码文件的大小和指令存储器的带宽。

- 强大的寻址模式

NIOS 指令集包含加载（Load）和存储（Store）指令，编译器可用来加速对结构和本地变量（栈）的访问。

- 可扩展性

用户可以直接将自己的逻辑单元（作为用户定制指令）加入 NIOS 算术逻辑单元（ALU）中。在软件开发包（SDK）中，系统会相应生成访问该定制指令的宏（用 C 语言或汇编语言编写）。

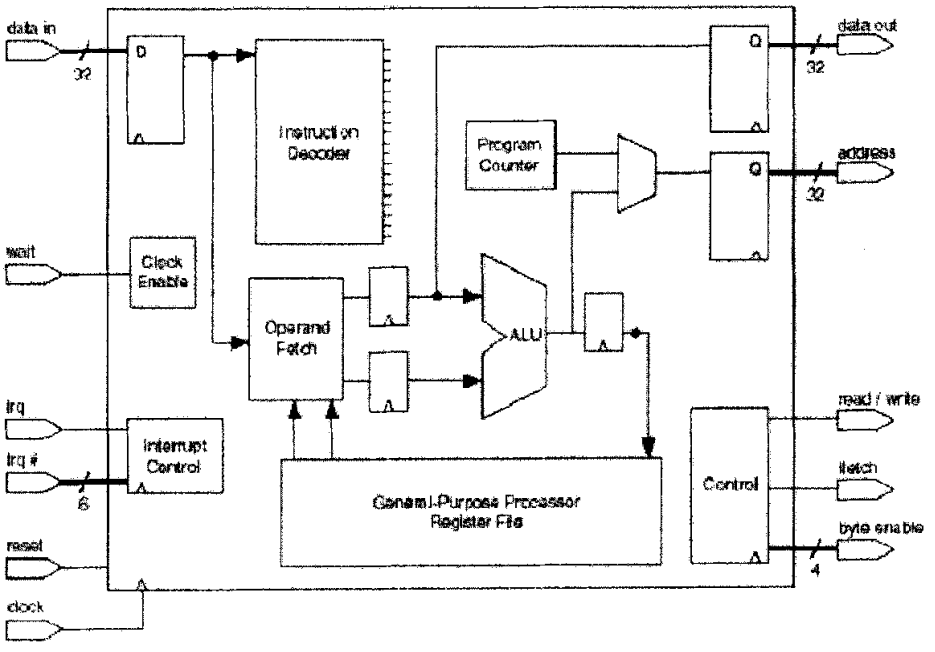



图 2.1－NIOS CPU 框架

### 2．2．1．2 内部寄存器

NIOS CPU 中的内部寄存器包括：一个通用寄存器文件，多个内部控制寄存器，一个程序计数器以及一个用于前缀指令的 K 寄存器。

1．通用寄存器

在 32 位 NIOS CPU 中的通用寄存器是 32 位宽的。寄存器文件的大小可以配置为 128 个、256 个或者 512 个寄存器。软件通过滑动窗口（32 个寄存器宽度）来访问内部寄存器，每次窗口的滑动步长为 16 个寄存器。滑动窗口能够遍历整个寄存器文件并且可对其子集进行访问。

寄存器窗口被分为 4 个相连的部分，如表 2.2 所示。最低 8 个寄存器（%r0

－%r7）是全局寄存器，也被称作%g0－%g7。全局寄存器并不跟随窗口位置的移动而变化，而是一直作为%g0－%g7 被访问。寄存器文件中上面的 24 个寄存器（%r8－r31）可以通过当前的窗口访问。

表 2.2－寄存器组

| | |
|---|---|
| 输入寄存器 | %r24－%r31 或 %i0－%i7 |
| 局部寄存器 | %r16－%r23 或 %L0－%L7 |
| 输出寄存器 | %r8－%r15 或 %o0－%o7 |
| 全局寄存器 | %r0－%r7 或 %g0－%g7 |

2．K 寄存器

K 寄存器是一个 11 位的寄存器，除 PFX 及 PFXIO 指令外，其他指令会把它置为 0。PFX 及 PFXIO 指令可以将一个 11 位的立即数置入 K 寄存器，然后紧接在 PFX（或 PFXIO）后的指令才可以使用 K 寄存器中的非零值。

PFX 和 PFXIO 使中断失效一个周期，因此 PFX（或 PFXIO）和其后的指令构成原子操作。它们也会被 SKP 类型的条件指令跳过。

K 寄存器不能通过软件直接访问，只能间接访问。例如，MOVI 指令把 K 寄存器的所有 11 位的内容传送到目标寄存器的位 15..5 中去。只有当前面指令是带有非零参数的 PFX 指令时，K 寄存器的读操作才能产生非零的结果。

3．%r0（%g0）寄存器

该寄存器被显式地用作以下指令的参数或者结果：STS16S、STS8S、ST8S、ST16S、ST8D、 ST16D、FILL8、FILL16、MSTEP、和 USR1－USR4。

4．程序计数器

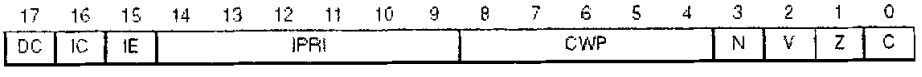
程序计数器（PC）寄存器包含当前正在执行的指令的字节地址。因为所有指令必须是半字对齐的，所以 PC 的最低有效位一直为 0。

除非跳转，否则每条指令执行后 PC 的值加 2（PC <— PC+2）。以下指令可以直接改变 PC 的值：BR、BSR、CALL、JMP、LRET、RET 和 TRET。

5．控制寄存器

NIOS 中含有 5 个独立寻址的控制寄存器，RDCTL 和 WRCTL 指令是唯一可以读写这些控制寄存器的指令（这意味着%ctl0 与%g0 是无关的）。

- STATUS 寄存器（%ctl0）




%ctl0 为状态寄存器。该寄存器内部各字段的作用如下：

DC：数据缓存使能位。

IC：指令缓存使能位。

IE：中断使能位。

IPRI：中断优先级。

CWP：当前寄存器窗口指针。

N、V、Z、C：状态标志位。各标志位含义如表2.3所示。

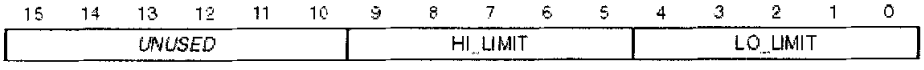
表2.3－状态标志位意义

| 标志 | 位 | 结果 |
|---|---|---|
| N | 3 | 结果的符号，即最高有效位 |
| V | 2 | 算术溢出，即结果的符号位与按照无限精度计算时的符号不一致 |
| Z | 1 | 结果为零 |
| C | 0 | 加法的进位，减法的借位 |

- ISTATUS寄存器（%ctl1）

ISTATUS是STATUS寄存器的拷贝。当处理异常时，STATUS寄存器的内容被拷贝到ISTATUTS寄存器内。这种机制使得在中断返回时能够恢复STATUS寄存器中的内容。

- WVALID寄存器（%ctl2）




WVALID寄存器包含两个值：HI_LIMIT和LOW_LIMIT。当SAVE指令使CWP由LOW_LIMIT减小到LOW_LIMIT－1时，寄存器窗口产生下溢异常（异常#1）。当RESTORE指令使CWP由HI_LIMIT增大到HI_LIMIT＋1时，寄存器窗口产生上溢异常（异常#2）。WVALID是可配置的，既可以只读也可以进行读/写。当CPU被复位时，LO_LIMIT被置为1，HI_LIMIT被置为最大的有效窗口指针（（寄存器文件大小/16）－2）。

- ICACHE寄存器（%ctl5）

ICACHE是指令缓存的行失效（line-invalidate）寄存器。向ICACHE写入一个地址后，会使得包含该地址的缓存行失效。ICACHE是一个只写寄存器。要注意在写ICACHE之前一定要禁止指令缓存（STATUS寄存器的IC位要置0）。

- CPU_ID寄存器（%ctl6）

CPU_ID包含一个16位的常量，用于标识NIOS处理器的版本。

- DCACHE寄存器（%ctl7）

DCACHE是数据缓存的行失效（line-invalidate）寄存器。向DCACHE写入一个地址后，会使得包含该地址的缓存行失效。DCACHE是一个只写控制寄存器。要注意在写DCACHE之前一定要禁止数据缓存（STATUS寄存器的DC位要置0）。

- CLR_IE寄存器（%ctl8）

对CLR_IE寄存器进行WRCTL操作都会将STATUS寄存器的IE位置0

（WRCTL 的值会被忽略）。

- SET_IE 寄存器（%ctl9）

对 SET_IE 寄存器进行 WRCTL 操作都会将 STATUS 寄存器的 IE 位置 1（WRCTL 的值会被忽略）。

**2．2．1．3 流水线**

NIOS CPU 是采用了流水线技术的 RISC 体系结构。除延迟槽和当 CWP 被 WRCTL 改变外，流水线的实现对于软件来说是透明的。

- 直接 CWP 操作

每一条改变 STATUS 寄存器（%ctl0）的 WRCTL 指令必须紧跟一条 NOP 指令。

- 分支延迟槽

分支延迟槽被定义为紧跟在 BR、BSR、CALL 和 JMP 指令后的指令。分支延迟槽在分支指令之后、分支目标指令之前被执行。

**2．2．1．4 存储器组织**

NIOS 处理器的数据访问采用小端对齐方式（little-endian）。数据存储器的地址空间必须是连续的且字对齐。如果存储器的数据宽度小于字宽，则数据总线可采用动态地址对齐方式（dynamic-bus sizing）来模拟 NIOS CPU 的全宽数据。如果外设的寄存器宽度小于字宽，则在高位填 0，这样 CPU 看来仍是一个字宽的数据。

- 读存储器（外设）

NIOS CPU 只能进字边界对齐的存储器访问。32 位的读操作只能从 4 的倍数的字节地址开始读取 1 个全字。对存储器进行读操作指令总是认为地址的低 2 位为 0。为了读取字节和半字，NIOS 提供从字中提取特定字节和半字的指令。

从存储器中读取数据的最简单的指令是 LD 指令。这个指令的典型例子是 LD %g3,[%o4]。该指令把%o4 作为间接寻址寄存器，把它寻址的数据装入%g3。地址的最低 2 位被当作 0。

然而，编写软件时经常遇到读取的数据小于 32 位的情况。NIOS CPU 提供指令来从字中提取字节和半字。EXT8D 指令用来提取 1 个字节，EXT16D 指令用来提取半字。EXT8D 指令的典型例子是 EXT8D %g3,%o4。EXT8D 指令使用%o4 的最低 2 位从%g3 中提取 1 个字节，然后把提取结果存入%g3。

- 写存储器（外设）

NIOS CPU 可以对存储器进行宽度为字节、半字或者字的对齐写操作。一个字可以用一条指令写到任意一个是 4 的倍数的地址内。一个半字可以用两条指令写到任意一个是 2 的倍数的地址内。一个字节可以用两条指令写到任意地址内。

将寄存器内容写入存储器时，最低字节写入的地址为 4 的倍数；第二低位的字节写入的地址是 4 的倍数加 1；其他字节依此类推。而低位半字的写入地址是 4 的倍数；高位半字的写入地址则为 4 的倍数加 2。

ST 指令可以把寄存器内容写到字对齐的地址内；ST8D 和 ST16D 可以把寄存器的字节和半字按照上面描述的方式写到相应地址内。

通常软件需要将特定的字节或者半字写入任意地址。该字节或半字在寄存器内的位置可能没有按照上面的要求与要写入的地址对应，为此 NIOS 提供了 FILL8 和 FILL16 指令。FILL8 指令和 FILL16 指令可以分别复制寄存器的最低字节和半字并占满整个%r0 寄存器。

- 缓存

NIOS CPU 可以通过配置决定是否含有指令缓存和数据缓存。数据缓存影响 NIOS 的存储器访问。数据缓存存储最近被访问过的数据字，只要可能就使用缓存中的数据值，从而减少了对存储器的访问。NIOS CPU 使用了最简单的直接映象方式，即地址的低位被用来选择缓存的行。在直接映象方式中，如果数据地址的索引（index）部分相同，这些数据字将被映射到缓存中相同的行中。为了检测在缓存同一行中到底存储的是哪个字，字地址的最高位作为标识（tag）。为了检测缓存中的数据是否有效，在缓存中每行还对应一个有效位。

当执行装载指令时（LD、LDP 和 LDS），NIOS CPU 比较装载地址的高位和选中的缓存行的标识。如果高位与标识匹配并且该行有效，则处理器将使用缓存中的数据而不再读取存储器，因此加速了处理性能。当处理器使用缓存数据时，称作“命中”（hit）。当缓存不包含所需的数据时，称作“失效”（miss）。

NIOS CPU 的写策略采用了最简单的写直达法（write-through）。即数据不但写入缓存，同时也写入存储器。向缓存写数据时，缓存行的地址由写入地址的索引部分决定，这样随后对于相同地址的读操作就会命中。此外，地址的高位作为标识被写入，并置有效位。

当缓存失效时，处理器执行一次存储器读传输，找到所需的数据字，把这个字写到目的寄存器中，同时写入缓存。这样，下次从相同地址读数据时就会命中。

### 2．2．1．5 寻址方式

- 5/16 位立即数寻址

很多算术和逻辑指令采用 5 位的立即数作为操作数。例如，ADDI 指令带有两个操作数：一个寄存器操作数和一个 5 位的立即数操作数。5 位的立即数代表了范围为 0～31 的常数。为了指定 6～16 位（数字 32～65535）的常量，需要用 PFX 指令来设置 11 位的 K 寄存器。它的值与 5 位立即数的值连接起来。PFX 指令必须直接放在它所改变的指令之前。

为了支持把 16 位的立即数常量拆分为 11 位（装入 PFX 寄存器）和 5 位立即数，汇编语言提供了%hi()和%lo()操作。%hi（x）把常量 x 的高 11 位（5～15 位）提取出来，%low（x）把常量 x 的低 5 位（0～4 位）提取出来。

- 全宽（full width）寄存器间接寻址

LD 指令和 ST 指令可以从存储器中装载一个字到寄存器中，或者从寄存器中存储一个字到存储器中，存储器地址由另一个寄存器给出。地址首先要向下与字地址对齐。K 寄存器被看作带符号的地址偏移量，以字为单位。偏移范围是（－4096～4092）字节。有效地址是：K×4＋（间接寻址寄存器的值&0xFFFFFFFC）。

如果 NIOS 处理器包括数据缓存，读取外设时在 LD 指令前需要 PFXIO 指令。

- 部分宽（patial width）寄存器间接寻址

32 位的指令不能读取一个字的部分内容。为了读取一个字的部分内容，需要把全宽寄存器间接读指令与提取指令（EXT8D、EXT8S、EXT16D 或者 EXT16S）结合在一起。

有些指令可以写一个字的部分内容。这些指令分为静态和动态两种类型。在动态情况下，源寄存器的内部位置与存储器中的字的内部位置由地址寄存器的低位来决定。在静态情况下，内部位置由指令的 1 或 2 位立即数来决定。和全宽寄存器间接寻址一样，K 寄存器被看作带符号的偏移量。

部分宽寄存器间接寻址的指令都使用%r0 作为源操作数。这些指令可以很容易地与 FILL8 和 FILL16 指令一起使用。

- 带有偏移的全宽寄存器间接寻址

LDP、LDS、STP 和 STS 指令可以从存储器中装载一个字到寄存器中，或者从寄存器中存储一个字到存储器中，存储器的地址由寻址寄存器指出的地址加偏移量形成。

不像 LD 和 ST 指令可以使用任意一个寄存器来指明存储器地址，这些指令使用各自专有的寄存器来寻址。LDP 和 STP 指令仅仅使用%L0、%L1、%L2 和%L3 寄存器来寻址。LDS 和 STS 指令仅仅使用堆栈指针，即%sp 寄存器（相当于%o6 寄存器），作为寻址寄存器。这些指令每一个都带有 1 个有符号的立即索引值，这个索引值指出了一个字大小的偏移量。

- 带有偏移的部分宽寄存器间接寻址

使用 STS8S 和 STS16S 指令可以通过立即数来指定相对于堆栈指针的一个字节或者半字的偏移量，写入源寄存器%r0 中相应的部分字。这些指令只能使用堆栈指针，即%sp（相当于%o6 寄存器）作为地址寄存器，并且只能使用%r0 寄存器（相当于%g0 寄存器，但在汇编指令里必须称作%r0 寄存器）作为数据寄存器。这些指令可以很容易地与 FILL8 和 FILL16 指令一起使用。

### 2．2．1．6 程序流程控制

- 相对分支（relative-branch）指令（BR 和 BSR）

NIOS 指令集中有两条相对分支指令：BR 和 BSR。分支的目标地址由当前程序计数器（也就是 BR 指令本身的地址）和 IMM11 指令计算得出。BR 指令和 BSR 指令都是无条件转移指令。条件转移可以在 BR 或 BSR 指令前设置 SKP 类型的指令来实现。

- 绝对跳转（absolute-jump）指令（JMP 和 CALL）

NIOS 指令集中有两条绝对（计算的）跳转指令：JMP 和 CALL。跳转的目标地址由一个通用寄存器给出。寄存器的内容被左移 1 位然后送到 PC 中。CALL 指令除返回地址保存在%o7 寄存器外，其他与 JMP 指令相同。条件跳转可在 JMP 或者 CALL 指令前设置 SKP 类型的指令来实现。

- 陷阱（trap）指令（TRET 和 TRAP）

NIOS 处理器为软件异常的处理提供了两条指令：TRAP 和 TRET。不像 JMP 和 CALL 指令，TRAP 和 TRET 指令都没有延迟槽：跟在 TRAP 后的指令直到异常处理返回后才被执行。紧跟在 TRET 指令后的指令根本不被执行。

- 条件执行（condition）指令（SKP、SKP0、SKP1、SKPRZ 和 SKPRNZ）

NIOS 指令集中有五条条件执行指令（SKP、SKP0、SKP1、SKPRZ 和 SKPRNZ）。这些指令都有相应的伪汇编指令（分别是 IFS、IF0、IF1、IFRZ 和 IFRNZ）。这些指令都要测试 CPU 的条件码，然后依据测试结果决定是否执行下一条指令。除了不同的测试对象外，五条 SKP 类型的指令（包括它们的汇编伪指令）的操作是相同的。无论测试的结果怎样，紧接着的指令都会被从存储器中取出。根据测试结果，下一条指令或者被执行，或者被取消。

### 2．2．1．7 Avalon 总线

NIOS 体系结构中的总线采用的是 Altera 公司开发的 Avalon 总线结构[6]。Avalon 总线通过端口把连接到它上面的主、从部件联系起来，通过指定时序，部件之间就可以进行通信了。Avalon 总线的显著特点是把连在它上面的部件分为主、从部件，并且通过总线自身提供的仲裁部件来协调部件间的通信。主部件是指那些含有主端口（master port）的部件，它们可以启动总线的传输过程；而从部件只能在总线上进行接收，不能启动总线传输。例如，连接在 Avalon 总线上的 NIOS 处理器是主部件，而连接在 Avalon 总线上的 SDRAM 是从部件。使用 Avalon 总线的一个例子系统如图 2.2 所示。

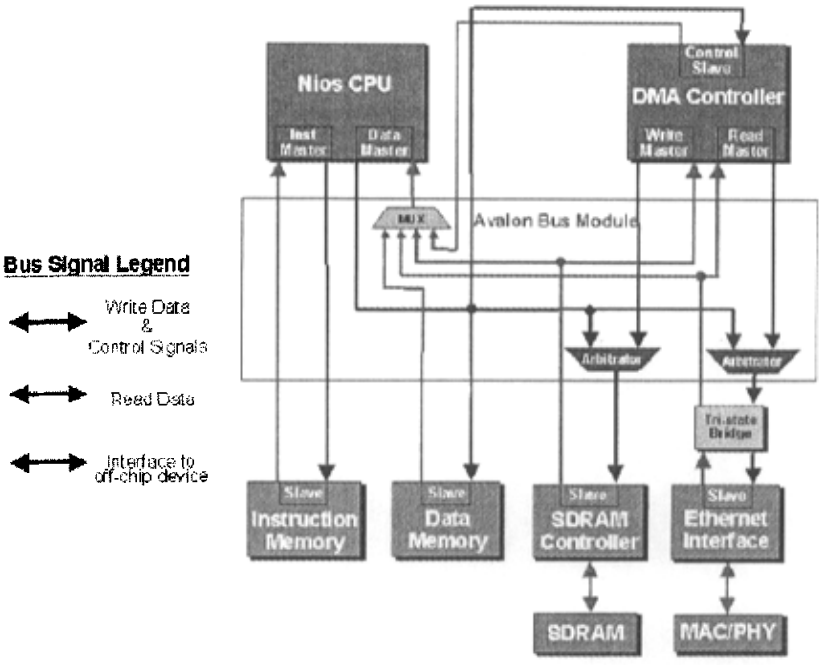



图 2.2—使用 Avalon 总线的例子系统

## 2．2．2 SOPC 设计

SOPC[7]是 Altera 公司提出的一种灵活、高效的片上系统设计方案。它的实质是 SOC 设计技术，它利用 FPGA 或 CPLD 器件的可编程性来进行 SOC 设计。

减轻设计者负担的最佳途径是把所有和处理器子系统相关的底层详细资料集中到单个工具中。这个工具就是 SOPC Builder，它需要有两方面的考虑：第一，它必须具有直观的图形用户接口 GUI（Graphic User Interface），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。第二，它必须自动完成系统集成工作，这样设计者不必拘泥于定义存储器映射，中断控制和总线控制这样的“制造商工作”。

通过 GUI，用户可以从 Altera 提供的 IP 库中选取一些组件，如处理器、SDRAM、Flash、各种 I/O 口等，并可通过选择配置相应的参数。如果用户有特殊功能要求，但 IP 库中没有，则用户可以加入自定义的逻辑来实现。在选择并配置好系统所需的各种 IP 后，点击 GUI 中的“Generate”按钮就可以自动生成系统了。当用户点击“Generate”按钮时，SOPC Builder 会生成每个硬件部件以及连接部件的片内总线结构、仲裁和中断逻辑。SOPC Builder 也会产生系统可仿真的 RTL 描述，以及为特定硬件配置设计的测试平台，能够（可选）把硬件系统综合到单个网表中。

拥有了这些合适的部件，硬件设计人员对自动硬件生成过程基本满意，但

是还需要满足软件设计者的要求。利用设计过程中采集的信息，SOPC Builder 能够生成 C 和汇编头文件，这些头文件定义了存储器映射、中断优先级和每个外设寄存器空间的数据结构。这样的自动生成过程帮助软件设计者处理硬件潜在的变化性。如果硬件改变了，SOPC Builder 会自动更新这些头文件。SOPC Builder 也会为系统中现有的每个外设生成定制的 C 和汇编函数库 。例如，如果系统包括一个 UART，然后 SOPC Builder 就会为访问 UART 的寄存器定义一个 C 结构，生成通过 UART 发送和接收数据的 C 和汇编例程。

SOPC Builder 运行时的 GUI 如图 2.3 所示。在 GUI 界面的左侧显示出的是各种 IP，右侧是系统中选中并配置好的各种 IP。

在 SOPC Builder 的 GUI 中，通过 Component Wizards 来选择系统所需的组件，然后生成组件，再由组件组成系统。其中，组件的信息来自 Class PTF 文件，Class PTF 包含 SOPC Builder 配置和生成部件所需的详细信息。在整个设计过程中，每一步都会与 System PTF 文件进行交互来交换组件及系统的信息，该文件是系统的配方，它定义了 SOPC Builder 生成完整系统必需的详细信息 。在组件生成及系统生成过程中，会产生输出文件及相关的库，其中包括 HDL 文件、软件文件、模拟文件及用户自定义文件。SOPC Builder 架构如图 2.4 所示。

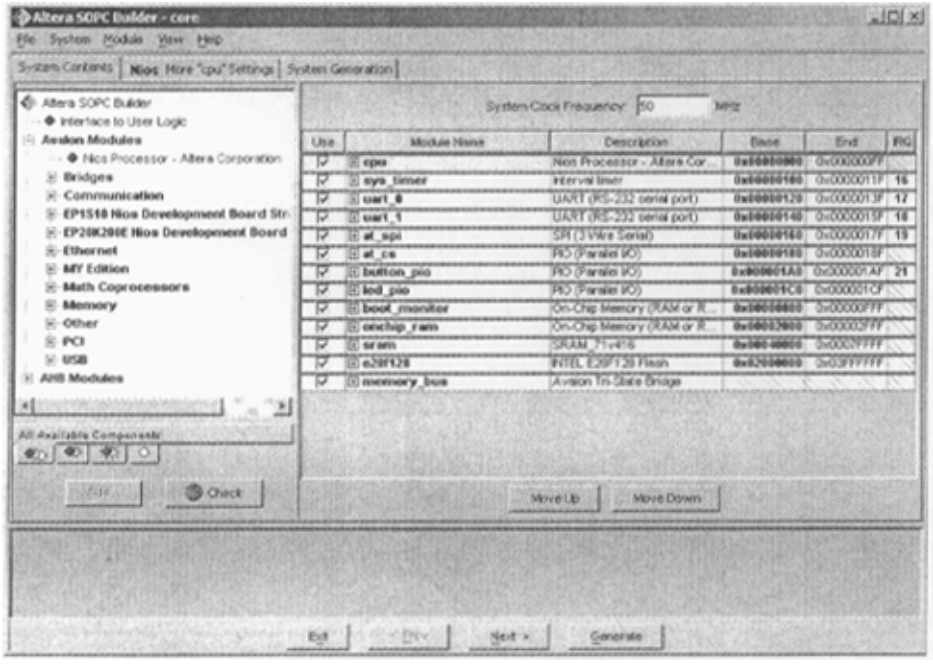



图 2.3－SOPC GUI 界面

在进行 SOPC 设计时，用户需要先通过 SOPC Builder 选择 IP 组件，然后用 SOPC Builder 产生所选择的系统组件的 VHDL 或者 Verilog 源文件。之后，用 Quartus II 编译整个系统，编译成功后，把编译文件下载到开发板上进行验证。

用 SOPC 进行开发的一般流程如图 2.5 所示。

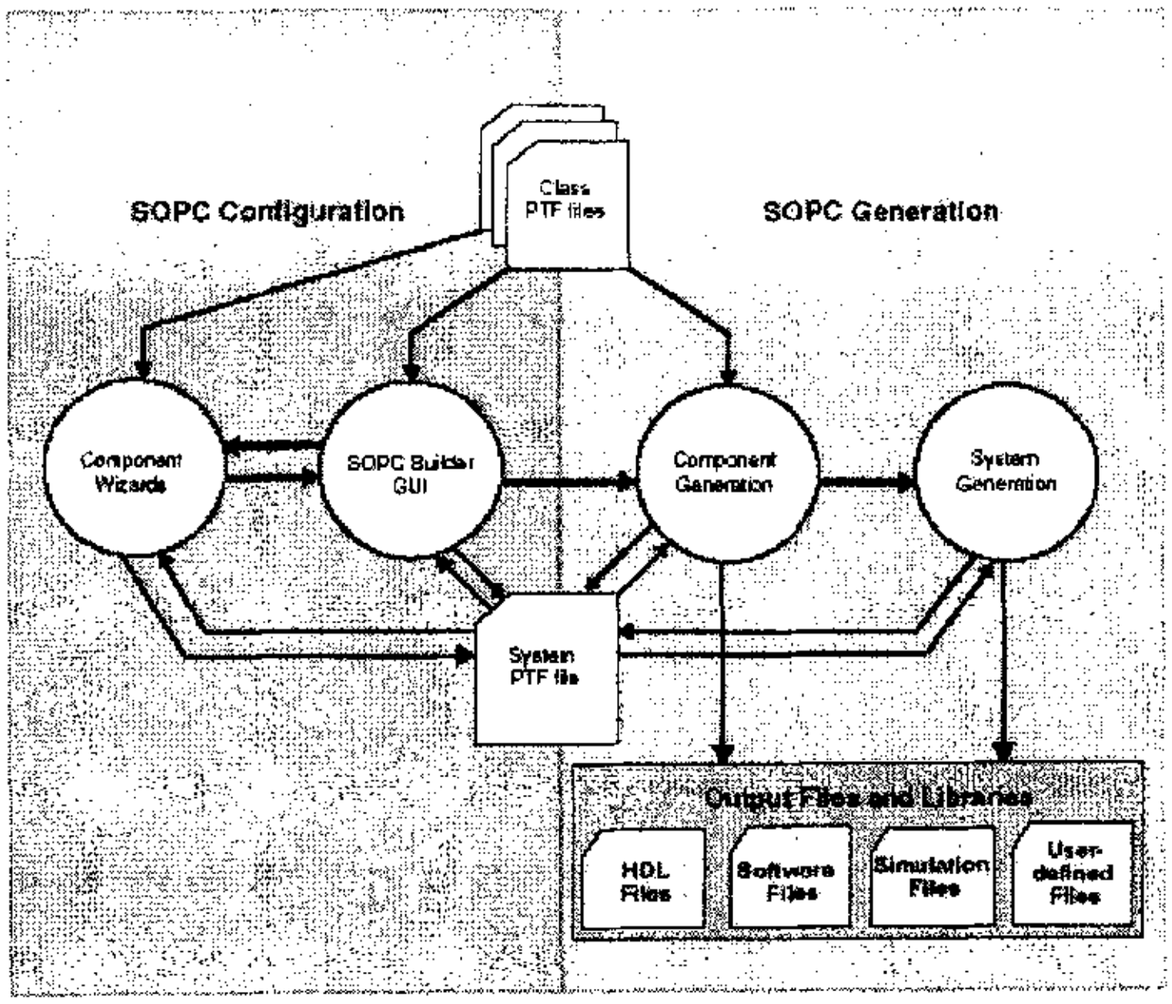



图 2.4－SOPC Builder 架构

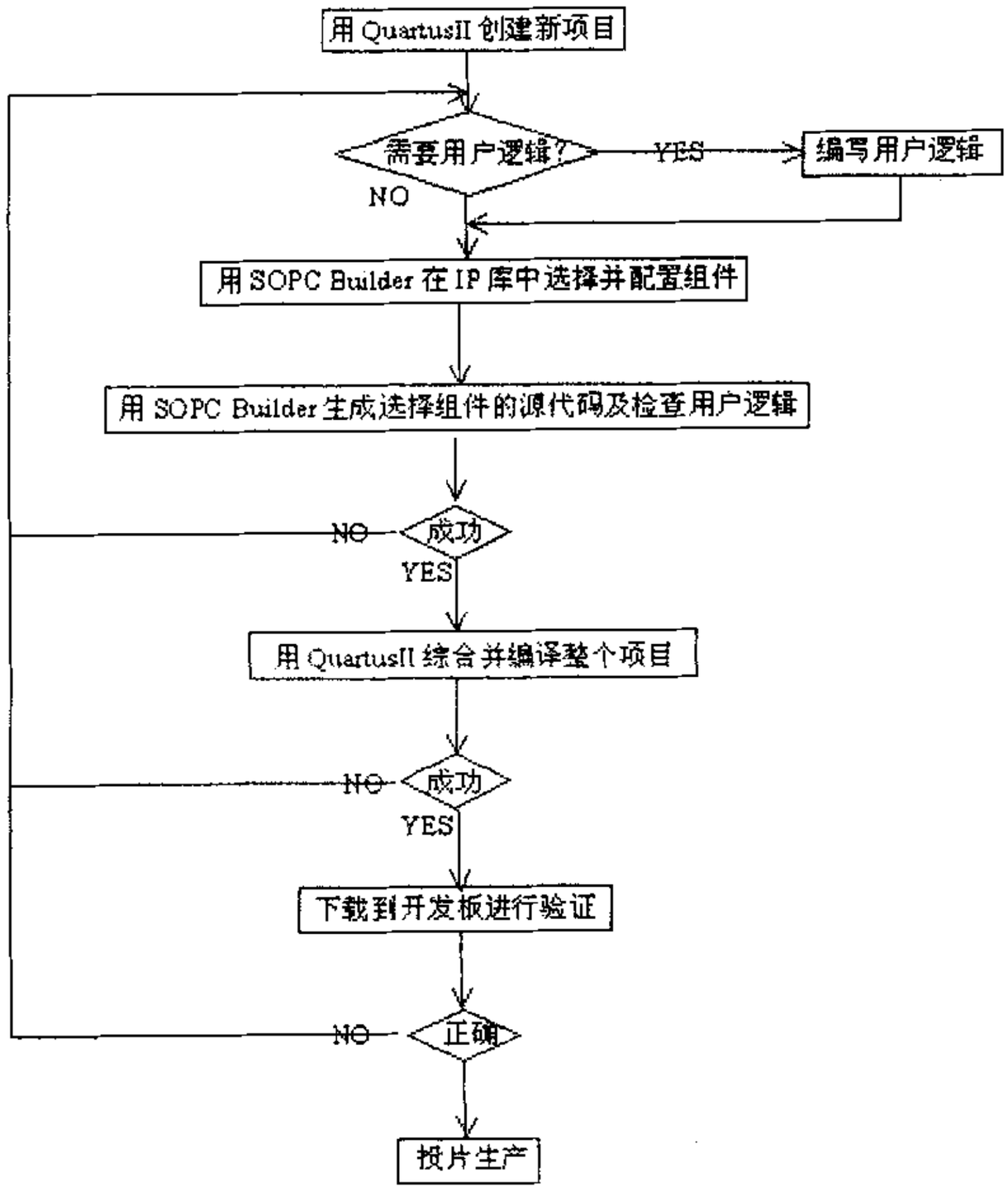



图 2.5－SOPC 开发流程

## 2．3 开发过程

### 2．3．1 概述

本小节介绍基于 NIOS 的 SOPC 开发过程[8]。它介绍使用 SOPC Builder 和 Quartus II 创建和处理 NIOS 系统模块设计，这个模块设计与 NIOS 开发板上提供的组件之间有接口联系。

本小节介绍如何一步步地创建并编译一个 32 位 NIOS 系统模块设计，这个模块设计叫做 nios_system_module，然后把它下载到 NIOS 开发板上。这个系统模块有一个 NIOS 嵌入式系统处理器，还有与系统相连的外设和它们之间的互联。

在创建完 nios_system_module 设计并把它与外部引脚相连后，把它下载到 NIOS 开发板上的 Altera APEC 设备上。之后，APEC 设备与 NIOS 开发板上的其他设备相互连接，这样，NIOS 嵌入式处理器就可以与 RAM、Flash、LED、LCD、开关和按钮相互通信。

### 2．3．2 设计入口

下面的部分按照所需步骤创建 nios_system_module 工程（project），然后解释如何创建包含 NIOS 系统模块的顶层（top-level）BDF。通过 SOPC Builder，可以创建并实例化 NIOS 系统模块。

1．创建 Quartus II 工程

运行 Quartus II，使用 **New Project Wizard** 创建工程，为工程和顶层设计入口指定名称，这里使用 nios_system_module。

2．创建 NIOS 系统模块

创建包含 NIOS 系统模块的顶层块设计文件（Block Design File — .bdf）。创建了设计文件之后，使用 SOPC Builder 创建 NIOS 嵌入式处理器，配置系统外设，连接这些组件组成 NIOS 系统模块。之后，把 NIOS 系统的端口连到 APEC 设备的引脚，这些引脚连接了 NIOS 开发板上的硬件组件。

我们创建的 NIOS 系统模块包括 NIOS CPU、boot_rom、RAM、Flash、Avalon_Bus、UART、LED、7 段显示码、Button_PIO 等组件。

### 2．3．3 编译并下载文件

创建完 NIOS 系统模块后，我们把它加入到系统顶层文件 bdf 中，并为模块加入输入/输入引脚，把这些引脚分配到 APEC 设备上。完成的顶层文件 bdf 如图 2.6 所示。在指定完器件并分配完引脚后，我们可以编译整个工程。编译的结果会生成一个.sof 文件。这就是要下载到 APEC 器件上的文件。我们使用 Quartus II 提供的工具把生成的 nios_system_module.sof 文件下载到开发板上的 APEC 器件上。

### 2．3．4 编程

使用 NIOS SDK 环境来编译软件程序，然后把编译完的软件程序下载到

APEC 器件上运行。这里，使用 Altera 提供的例子程序 hello_nios.c 来进行演示。

1．编译软件

使用如下命令来编译软件：

nb hello_nios.c

如果编译成功，会生成一个可供下载的文件，hello_nios.srec。

2．下载文件

使用如下命令把编译的结果下载到开发板上的 APEC 器件中去：

nr hello_nios.srec

程序在 NIOS SDK 中的运行情况如图 2.7 所示。

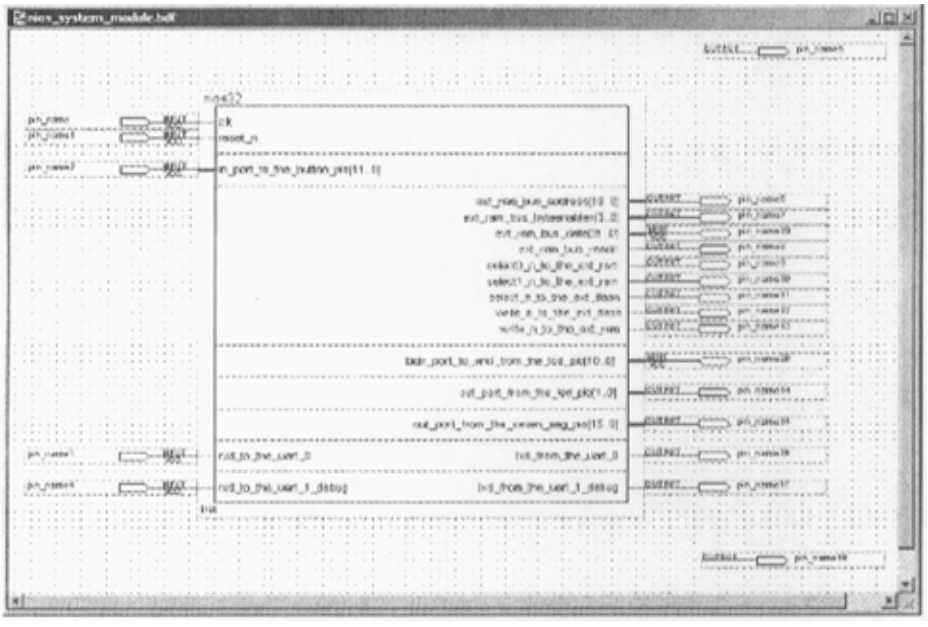


图 2.6－系统顶层文件图


图 2.7－NIOS SDK 中软件运行情况

## 2．4 SOPC 高级特性的研究

进行 SOPC 设计的关键是使用 IP 核。使用现存的大量的 IP 核可以减少重复劳动，进而可以加快进行系统的设计，这也是 SOPC 或者说是 SOC 的魅力所在。Altera 公司推出 SOPC 设计技术的同时也为用户提供了一些可以选用 IP 核，其中除最重要的 NIOS 处理器软核外，还有诸如 SRAM、Flash、PIO、Timer 等 IP 核。

然而，在实际开发中，由于系统的需求是多样的，这就导致系统所需的功能是多样的，从而进一步要求有实现这些功能的各种 IP 核，而 Altera 公司提供的这些 IP 核是远远不能满足人们设计各种系统所需的各种 IP 核的要求的。针对这种情况，Altera 提供了使用第三方 IP 核的简单接口，只要符合这些接口规范，就可以使用各种 IP 核进行 SOPC 设计，这就给广大的开发者提供了一个广阔的开发空间。

在使用第三方 IP 核方面，SOPC 中提供了两种方法：用户自定义逻辑和定制指令。由于这两部分功能很重要，而在 Altera 提供的文档里介绍的又不是十分详细，所以本人专门对这两部分功能进行了研究。下面对这两部分功能进行详细介绍。

### 2．4．1 SOPC 设计中的用户自定义逻辑

#### 2．4．1．1 概述

在进行 SOPC 设计时，有些系统功能可以由 IP 库中提供的组件来实现。虽然 IP 库中提供的组件可以完成系统所需的功能，但有时显得不够灵活。比如要并行处理几个输入源的输入数据，用 PIO 作为输入源实现时，每个输入源每次输入都要产生一个中断。当输入源较多时，对这些中断源的处理，增加了处理器的负担。为解决这个问题，可由用户自定义逻辑[9]来实现这个功能。

#### 2．4．1．2 实例简介

下面的实例是一个小型的嵌入式系统的实现，并不涉及到从系统说明到最后实现的所有步骤，主要介绍了其中的硬/软件划分及最后的实现部分。这个系统主要包括以下功能部件：

1．一个处理器，用于运算及控制功能。

2．一个用户自定义逻辑，对 Excalibur 开发板上的三个按钮动作的并行计数。

3．用户程序，读取用户逻辑对按钮的计数并显示其结果。

其中第 1、2 项由硬件实现，第 3 项由软件实现。

#### 2．4．1．3 Excalibur 开发平台

本实例是在 Excalibur 开发平台上实现的。Excalibur 开发平台是 Altera 公司的一个样机平台，上面除集成有一块 APEX EP20K200 EF C484-2X 可编程芯片外，还有 SDRAM、Flash、Controller、PIO、JTAG 接口等外设。结合 Altera 的

Quartus II 应用软件及 SOPC Builder，可以把设计的嵌入式系统下载到 Excalibur 开发板上进行验证。如果各项标准符合设计要求，就可以投片生产。

#### 2．4．1．4 实现相关

1．硬/软件划分

首先，我们要考虑一个硬/软件划分问题。硬/软件划分是嵌入式系统硬/软件协同设计方法中很重要的一个步骤。目前在硬/软件协同设计领域中已经有一些成熟的硬/软件划分算法，比如 GCLP(Global Criticality/Local Phase)算法、遗传算法(GA：Genetic Algorithms)、混合整数线性规划(MILP：Mixed Integer Linear Programming)算法，等等。当然，在某些设计中，设计人员的经验也大量应用于硬/软件划分过程。在这个实例中，由于系统小，实现的功能少，所以没有应用复杂的硬/软件划分算法，而是设计人员的经验在硬/软件划分中起了主要作用。这个实例中硬/软件划分的依据是实现的复杂度与代价。这个实例中需要一个处理器来执行计算、调动等任务。这个处理器当然要由硬件来实现。同时需要实现的一个功能是三个按钮并行计数，考虑到硬件的固有特性，很容易实现并行功能，并且占用的硬件资源也不多；相反，如果这个功能用软件来实现，其实现复杂度相对硬件实现来说要高得多。这样，经过复杂度与代价的综合考虑，系统的这个按钮计数部分也由硬件来实现。并行计数的结果存储在系统中的暂存器中，等待后续处理。后续处理只是读出计数结果并显示，虽然用硬件也可实现，但软件实现比较简单，而且不用占用硬件资源，所以这个功能可以用软件来实现。这样，考虑到系统所要完成的功能以及硬/软件的各自特性，完成了该实例的硬/软件的划分。

2．硬件部分实现

接下来，用集成在 Quartus II 中的 SOPC Builder 生成一个 NIOS 处理器，还包括一些外设，如 SDRAM、Flash 等。这里所涉及的一个问题就是选择什么样的 NIOS 处理器。假定我们所要实现的系统需要的是 32 位的处理器，而且需要大容量的寄存器文件，那么在配置 NIOS 处理器时，可以选取 32 位的 NIOS 处理器，512 个寄存器来满足要求。

最后，用 VHDL 编写用户逻辑，实现 3 个按钮并行计数的功能。考虑到用户逻辑要与 NIOS 处理器进行通信，所以要增加 2 个信号：chipselect 和 address 信号。其中 chipselect 信号是片选信号，当其值为“1”时 NIOS 处理器选中用户逻辑。address 信号是地址信号，NIOS 处理器用地址信号来寻址用户逻辑。而且，只有通过 address 信号，用户逻辑才能正确的连到总线上与 NIOS 处理器进行通信。

由于片选信号(chipselect)和地址信号(address)在用户逻辑与 NIOS 的连接中尤为重要，下面再进行一下详细说明：

对于独立的用户逻辑，如果不需要与 NIOS 进行通信，则片选信号和地址信号是不需要的。比如，在这个实例中，如果只是实现一个并行计数和显示的功能，那么片选信号和地址信号都不需要。当然，在这种情况下，NIOS 也是不知道这个用户逻辑的存在的，二者是并行对等的关系，即 NIOS 在工作的同时，用户逻辑可以并行的独立工作。

当需要用户逻辑与 NIOS 进行通信时，则必须加入片选信号和地址信号。只有这样，用户逻辑才能正确地连接到 NIOS 总线上与 NIOS 进行通信。

前面说过，当片选信号为“1”时 NIOS 选中用户逻辑。那么片选信号什么时候被设置为“1”呢？这就要与地址信号一同来说明。当在用户逻辑里加入片选信号和地址信号后，SOPC Builder 会自动给用户逻辑分配一个基地址。SOPC Builder 对于系统中包括用户逻辑在内的所有组件统一进行编址。由于这个实例中要处理 3 个按钮并行计数，则地址信号需要 2 位。2 位地址信号可以处理 4 个地址，这里只用了 3 个：“00”，“01”和“10”。这个地址信号在 VHDL 中所形成的地址，比如“00”，“01”，是相对地址。这个相对地址与基地址结合在一起，形成绝对地址。比如，第一个按钮的计数值分配到相对地址“00”，那么它的绝对地址为 0x4A0+0*4=0x4A0，第三个按钮的计数值分配到相对地址“10”，则它的绝对地址为 0x4A0+2*4=0x4A8，其中 0x4A0 是系统分配的基地址。在计算绝对地址时，为什么要给相对地址乘 4 呢？因为这个实例实现的是 32 位 NIOS 处理器，所以每个寄存器占据 32 位，而地址是按字节分配的，所以 32÷8=4，即每个寄存器使用 4 个字节地址。当应用程序访问这些地址时，无论读还是写，首先将片选信号设置为“1”。比如在应用程序中有如下代码：

```
asic1=(int *)0x4a0; //指定要访问的地址
read_data1=*asic1; //读取地址中的内容
```

则在读操作时片选信号 chipselect 置为“1”。

对于地址信号的选择，则要在应用程序中明确指定，比如上面代码中指定访问的地址是 “0x4A0”。

下面附上实现用户逻辑的 VHDL 代码。由于篇幅限制，详细代码略去，这里只附上一些重要部分：

```
--以下是实体说明部分
ENTITY test IS
port(
        clock           :   IN  STD_LOGIC;
        pb_gen_count1   :   IN  STD_LOGIC;
        pb_gen_count2   :   IN  STD_LOGIC;
```

```
        pb_gen_count3   :   IN   STD_LOGIC;
        pb_gen_clear    :   IN   STD_LOGIC;
        chipselect      :   IN   STD_LOGIC;
        address         :   IN   STD_LOGIC_VECTOR(1 downto 0);
        count_out       :   OUT STD_LOGIC_VECTOR(7 downto 0)
    );
END test;
--以下是实体的结构部分中的输出部分
--其中的temp1、temp2、temp3是内部信号
--用以保存三个按钮的计数值
--这三个信号并不出现在实体说明部分中
--使用的是并行语句，保证三个按钮可并行计数
count_out<=temp1 when (chipselect='1' and address="00") else
           temp2 when (chipselect='1' and address="01") else
           temp3 when (chipselect='1' and address="10") else
           "00000000";
```

接下来要做的是把用户逻辑加入到NIOS处理器中去。在SOPC Builder的图形用户界面中，有一项选项用来添加用户逻辑。选择该选项后，加入用户编写的VHDL文件，同时指定实体说明中各个信号的类型。clock被指定为“clk”类型，pb_gen_count1等被指定为“export”类型，chipselect被指定为“chipselect”类型，address被指定为“address”类型，count_out被指定为“readdata”类型。之后，生成一个NIOS处理器软核及相关外设。

最后工作是编译整个项目，并把生成的后缀为sof的文件下载到Excalibur开发板上的可编程芯片中。

3．软件部分实现

软件部分工作是编写一个应用程序，其功能是读出各个按钮的计数值，并把它们显示出来。这个部分主要要考虑的问题是访问连在NIOS总线上的用户逻辑。前面介绍过，通过定义chipselset和address信号，SOPC Builder会自动分配给用户逻辑一个地址，那么对于用户逻辑的访问，其实就是对分配给用户逻辑的地址的访问。

下面附上应用程序的部分代码来说明如何与用户逻辑进行通信：

```
……
//三个按钮的地址
int asic1=(int *)0x4a0;
int asic2=(int *)0x4c0;
```

```
int asic3=(int *)0x4d0;
……
//读取计数值
int read_data1=*asic1;
int read_data2=*asic2;
int read_data3=*asic3;
……
//显示结果
printf("pb1:%d\tpb2:%d\tpb3:%d\n",read_data1,read_data2,read_data3);
……
```

在生成的 NIOS 处理器上编译并运行该应用程序。我们可以随意按动按钮，然后检查显示的计数值，验证结果的正确性。

### 2．4．2 SOPC 设计中的定制指令

#### 2．4．2．1 概述

在进行 SOPC 设计时，系统设计者可以将费时、复杂的软件算法作为定制指令[10]加入到 NIOS 的指令系统集中。使用定制指令这个特性可以大大提高系统的性能，它的用途也是十分广泛的，比如可以用来优化软件的内部循环、计算耗时的应用等。

定制指令包括两个核心部分：

1．定制逻辑块：实现定制指令的硬件，通过 HDL 来实现。NIOS 微处理器最多可包括 5 条定制指令。

2．软件宏：允许系统设计者通过软件来访问定制指令。

实现定制指令的 HDL 源文件一般按照表 2.4 来定义端口。HDL 源文件中的端口名称必须与表 2.4 中的名称相同。

在进行 SOPC 设计加入定制指令时要为加入的指令进行命名，例如加入了一条定制指令，为其命名为“cust_inst”，则在软件中按如下格式使用软件宏来调用定制指令：

```
nm_<name>_pfx(prefix,dataa,datab),
或者
nm_<name>(dataa,datab);
```

其中的“name”就是定制指令的名称，比如上例可以用如下方法调用软件宏：

```
nm_<cust_inst>_pfx(prefix,dataa,datab),
或者
nm_<cust_inst>(dataa,datab)。
```

表 2.4一定制指令中的端口

| 端口名 | 宽度（位） | 方向 | 说明 |
|---|---|---|---|
| dataa | cpu 宽度 | 输入 | 操作数 |
| datab | cpu 宽度 | 输入 | 操作数（可选） |
| result | cpu 宽度 | 输出 | 结果 |
| clk | 1 | 输入 | 时钟信号 |
| reset | 1 | 输入 | 时钟重置信号 |
| clk_en | 1 | 输入 | 时钟使能信号 |
| start | 1 | 输入 | 操作开始信号 |
| prefix | 11 | 输入 | 参数（可选） |

下面对于表 2.4 中的端口再进一步进行一下说明：

定制指令有两个操作数：dataa、datab，其中第一操作数 dataa 是必须的，第二操作数 datab 是可选的。这两个操作数的位宽和 NIOS 处理器的位宽相同。

result 是定制指令的运行结果。

clk、reset、clk_en、start 这 4 个端口是起时序控制作用的。

prefix 这个端口的位宽是 11 位，它可以作为额外的信息传递到定制指令中，也即通过 prefix 可以实现最多 256 种功能。例如可能有这样的定义：prefix 为 0 时执行 dataa 与 datab 相乘的功能；prefix 为 1 时执行 dataa 与 datab 相除的功能。对于上述假设，在软件中可以这样调用软件宏：

```
//dataa=50,datab=25;
//计算 dataa 与 datab 相乘的结果
tmp1=<nm_cust>(0,50,25);
//计算 dataa 与 datab 相除的结果
tmp2=<nm_cust>(1,50,25);
```

调用软件宏执行的结果为：

```
tmp1=1250;
tmp2=2;
```

另外，在加入定制指令时要指定执行定制指令所需的时钟周期数。这可以通过单独模拟实现定制指令的 HDL 文件来获得。

### 2．4．2．2 开发实例

下面通过一个具体的实例来介绍如何在 SOPC 设计时实现定制指令。

在 NIOS 上浮点数的乘法一般是用软件方法实现的，效率不高。下面的实例分别用软件方法和定制指令方法实现了浮点数的“乘 4”运算。这个实例的重点在于介绍如何实现定制指令，而不在于它的实用性。这个实例在说明如何实现定

制指令的同时，通过软件实现方法和定制指令实现方法的比较，可以看到定制指令提高了软件的运行速度，进而提高了系统的效率。

首先要实现定制指令的 HDL 源文件，这里使用语言是的是 VHDL。由于篇幅有限，这里仅给出 VHDL 源文件中的一些重要部分并予以解释。这个实例只用到第一个操作数 dataa。

```
--下面是实体说明部分
--可以看到声明的端口名称与表 2.4 相同
entity cust is
port
(
   dataa      :    in   std_logic_vector(31 downto 0);
   result     :    out  std_logic_vector(31 downto 0);
   clk        :    in   std_logic;
   reset      :    in   std_logic;
   clk_en     :    in   std_logic;
   start      :    in   std_logic
);
end cust;
--下面是结构体部分
--具体实现定制指令的功能
--本实例实现的就是“乘 4”功能
--“乘 4”可以用左移 2 位来实现
architecture a of cust is
begin
     process(clk)
     begin
          if clk'event and clk='1' then
               if reset='1' then
                    result<="00000000000000000000000000000000";
          else
               if clk_en='1' then
                    result<=dataa(29 downto 0) & "00";
               end if;
          end if;
```

```
        end if;
    end process;
end a;
```

用 VHDL 实现了定制指令的功能后通过 SOPC Builder 的图形用户界面把这个源文件作为定制指令加入到 NIOS 指令集中，并为其命名为“cust”。

在加入定制指令时要指定执行定制指令所需的时钟周期数，为此单独把定制指令的 VHDL 文件编译并进行模拟，其模拟波形图如图 2.6 所示。从其模拟波形中看到，定制指令的结果在第 3 个时钟周期开始后才开始稳定、正确地出现，故可以确定该定制指令需要 3 个时钟周期。

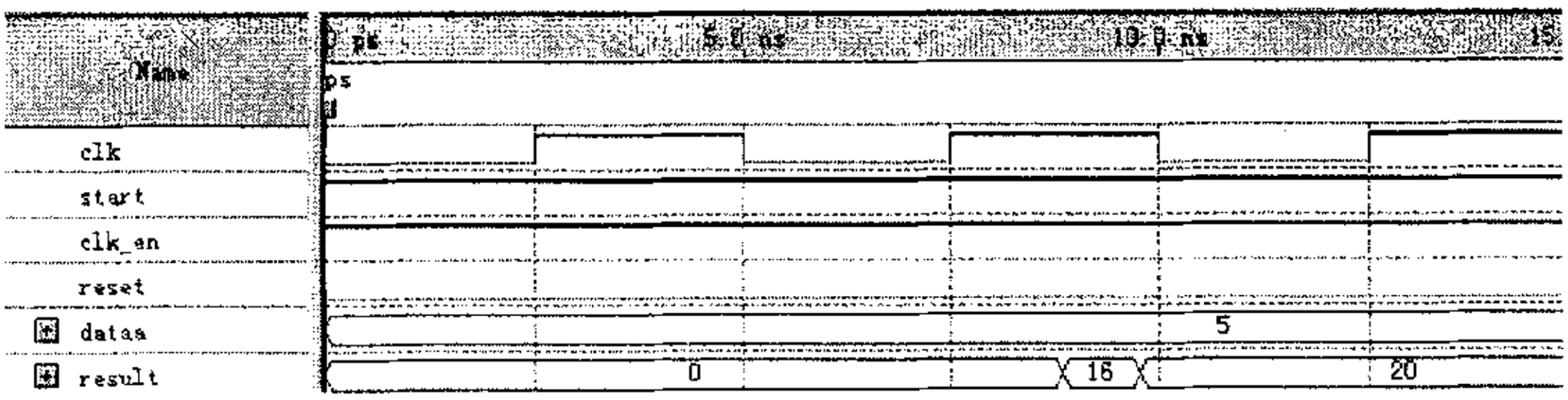



图 2.6－定制指令的模拟波形图

在应用程序中通过软件宏来调用定制指令。下面给出应用程序的部分代码。

```
void main()
{
    volatile float a=5.0;
    volatile float res1,res2;
    volatile DWORD dwStartTick;
    volatile DWORD lTicksUsed;
    volatile DWORD our_dwStartTick;
    volatile DWORD our_lTicksUsed;
    ……
    //软件实现浮点“乘 4”
    //记录开始时间
    dwStartTick=GetTickCount();
    // 软件执行结果
    res1=a*4;
    //记录结束时间
    lTicksUsed=GetTickCount();
    //计算实现功能所需的时间
    CheckTimeStamp (dwStartTick, lTicksUsed);
```

```
……
//定制指令实现浮点“乘 4”
//记录开始时间
our_dwStartTick=GetTickCount();
//定制指令执行结果
res2=nm_cust(a);
//记录结束时间
our_lTicksUsed=GetTickCount();
//计算实现功能所需的时间
CheckTimeStamp(our_dwStartTick, our_lTicksUsed);
……
}
```

运行该程序的结果显示，软件完成这个功能用了“2048”个时钟周期，而定制指令完成这个功能只用了“758”个时钟周期。通过指令完成功能所需时间的比较，可以看到定制指令的实现方法要比软件的实现方法效率高很多。

# 第三章 复旦大学 CAT 实验室快速样机平台设计

## 3.1 复旦大学 CAT 实验室快速样机平台概述

### 3.1.1 样机平台主要组成部分

本小节对复旦大学 CAT 实验室研制的快速样机平台做一下简要介绍。在后面的章节里，如无特殊说明，则样机平台均指复旦大学 CAT 实验室研制的这台快速样机平台。

复旦大学 CAT 实验室研制的样机平台实物如图 3.1 及图 3.2 所示。这个样机平台分三层：底板、主控板及配置板，分别如图 3.3、图 3.4 及图 3.5 所示。

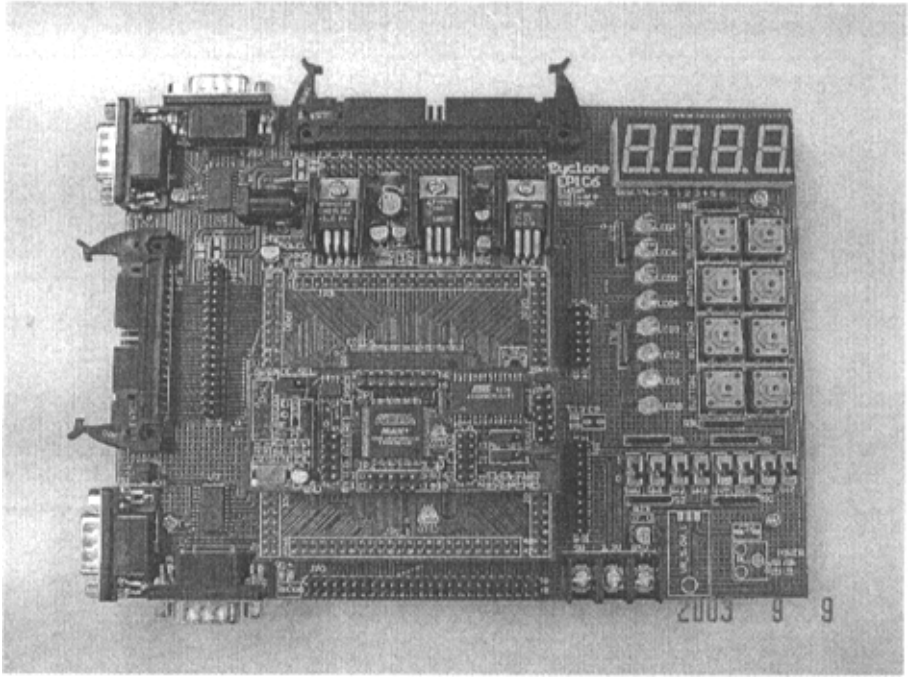



图 3.1－样机平台俯视图

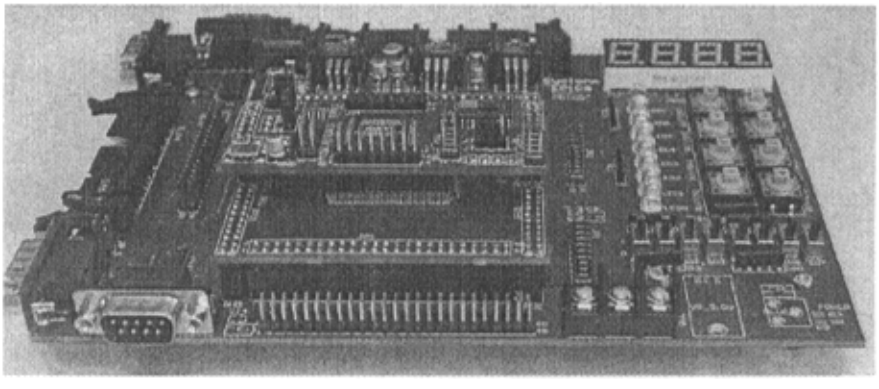


图 3.2－样机平台正视图

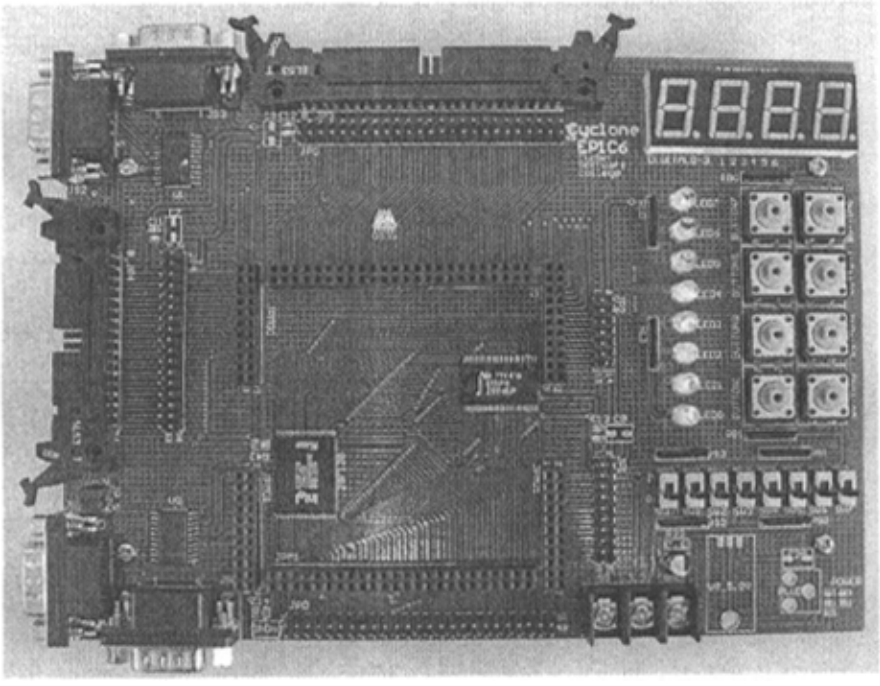



图 3.3－样机平台的底板

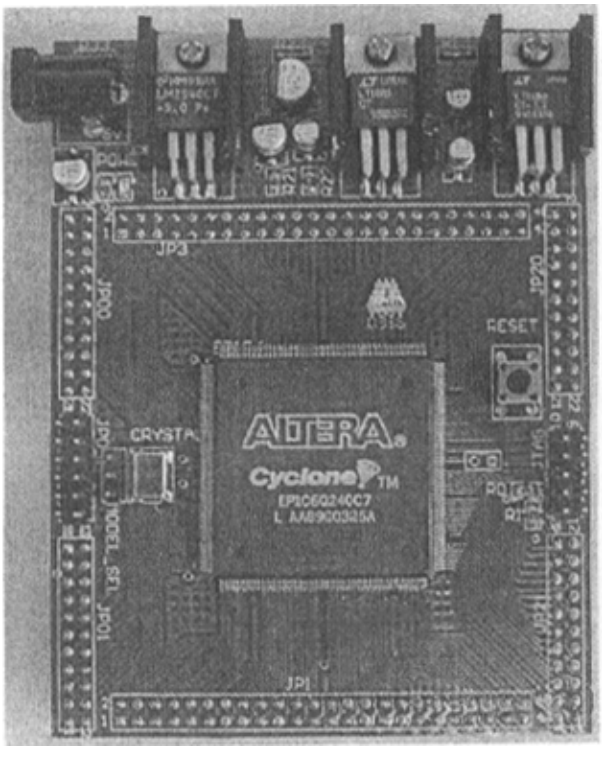



图 3.4－样机平台主控板

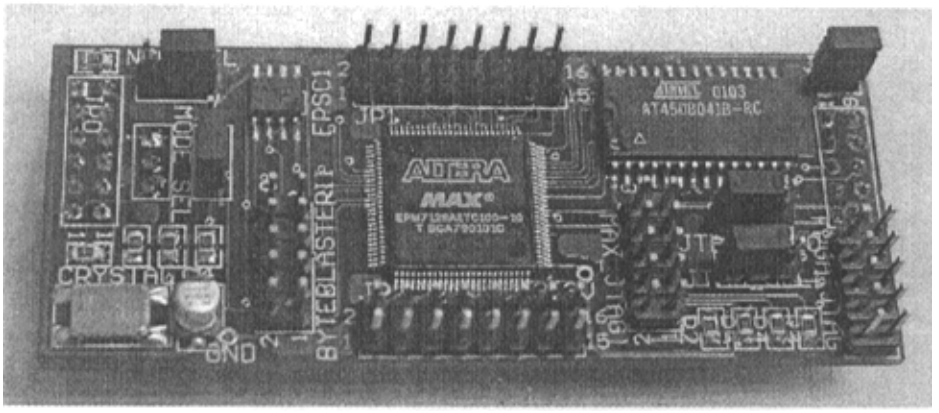



图 3.5－样机平台配置板

样机平台主要由以下几部分组成：

1．底板部分

底板部分主要有存储模块，输入/输出模块、串口通信模块组成。存储模块包括两部分：SRAM 及 Flash。SRAM 采用的是 IDT（Integrated Device Technology）公司的 IDT71V416 芯片；Flash 采用的是 Intel 公司的 StrataFlash 芯片。

2．控制板部分

控制板部分主要由 FPGA 和电源组成。其中的 FPGA 是 Altera 公司最新推出的 Cyclone 芯片族的 EP1C6 芯片。样机平台的输入电压是 9V 的，经过转换，可以为样机平台上的器件提供 3.3V 或者 5V 的电源。

3．配置板部分

配置板的作用主要是对主控板上的 FPGA 进行配置。为了对 FPGA 进行灵活配置，增加了两种配置方式：主动配置与被动配置。为此，进行主动配置采用的芯片是 Altera 公司的 EPCS1；进行被动配置采用的芯片是 Altera 公司的 Max7000A 芯片族的 EPM7128AE 芯片及 Atmel 公司的 AT45DB041B 芯片。对于主动配置与被动配置在后面会进行更详细的介绍。

下面对样机平台的组要器件做一些详细的介绍。

### 3．1．2 样机平台主要器件特性

1．Cyclone 芯片族及 EP1C6 芯片

Cyclone 芯片族是一款低价格，中等密度的 FPGA，采用 0.13um，全铜 SRAM 工艺，容量从 2,910 个逻辑单元到 20,010 个逻辑单元，1.5v 内核。 Cyclone 在设计初期，针对成本做了认真的优化，根据网上（www.altera.com）公布的价格，Cyclone 仅为 Altera 现有主流器件价格的 30%－50%。Cyclone 采用和 Stratix 器件相似的结构，有着和 Startix 相似的性能，但去掉 DSP 块、MegaRAM，降低 LVDS 接口速率等指标，以适应大多数设计的要求，同时分担用户所面临的成本

压力。

我们样机平台使用的是 Cyclone 芯片族中的 EP1C6 芯片。该款芯片具有 5,980 个逻辑单元 LEs（Logic Elements）、1 个锁相环、20 个 M4K RAM 块。

2．IDT71V416 芯片

Cyclone 芯片虽然可以使用片内 SRAM，但容量有限。所以我们的样机平台增加了片外 SRAM，使用的是 IDT 公司的 IDT71V416 芯片。该 SRAM 具有 256K*16-bit 的存储容量。

3．StrataFlash 芯片

为了长久保存数据、程序等资料，我们在样机平台上加入了 Flash 存储器，采用的是 Intel 公司的 StrataFlash 芯片。该芯片的存储容量为 128Mbit，也即 16Mbyte 的容量。该芯片被组织成 128 个 128-Kbyte 模块，也即共含有 128 个模块，每个模块的容量为 128Kbyte。

4．EPCS1 芯片

EPCS1 是一款基于 Flash 技术的 FPGA，这样，存储在它里面的数据会长期保存。它的容量是 1,048,576bits。由于 Cyclone 芯片族使用了新的数据压缩技术，这使得设计者可以使用容量较小的串行配置器件，如 EPCS1，来配置容量较大的 FPGA，如 Cyclone 的 EP1C6。

5．EPM7128AE 芯片及 AT45DB041B 芯片

我们样机平台上采用 EPM7128AE 芯片及 AT45DB041B 芯片是为了对 Cyclone 的 EP1C6 芯片进行被动配置。EPM7128AE 芯片是一款基于 Flash 技术的 CPLD，它属于 Altera 公司 Max7000A 芯片族。EPM7128AE 具有 2,500 个可用门，128 个宏单元，8 个逻辑阵列块及 100 个用户 I/O 引脚。EPM7128AE 主要由各种逻辑门组成，不能存放数据，所以我们的样机平台配置了 AT45DB041B 芯片来存放数据，以便配置 Cyclone 的 EP1C6 芯片。AT45DB041B 是一个 Flash 存储器，它的容量是 4Mbit。

## 3．2 样机平台特性

我们的样机平台具体有如下一些特性：

1．可扩展性

通过主板上扩展用连接件（双排插针）可以对样机平台进行功能扩展。Cyclone EP1C6 一共有 185 个用户管脚，除了样机平台上使用的用户管脚外，还有剩余的用户管脚。我们把这些剩余的用户管脚作为功能扩充用管脚，把这些管脚做成了双排插针 JP3 和 JP4。JP3 为 50 脚扩展用双排插针，JP4 为 34 脚扩展用双排插针。利用这些扩展用的双排插针，可以方便地对样机系统进行功能扩充，加入各种新的功能模块。例如，在样机平台上开发的应用实例——汽车运行信息记录仪就是

利用了 JP3 扩展了 A/D、D/A 功能，使系统有了数/模、模/数转换功能。

2. 易维护性

如前所述，我们的样机平台由三层组成：底板层、主控板层和配置板层。底板层主要由输入/输出模块、存储器模块、串口通信模块组成；主控层主要由 Cyclone EP1C6 FPGA 组成；配置层主要由用于配置 Cyclone EP1C6 FPGA 的器件组成。这样设计的目的是为了今后易于维护考虑的。分层的结果是不同功能的部件位于不同的层上，这样如果今后希望更改什么功能部件的话，不用改动其他的层。

3. 多种配置方式

样机平台对于 Cyclone EP1C6 FPGA 的配置方式有三种：

- 直接配置

在主控板和配置板上都有一个 JTAG 接口，这两个 JTAG 接口是直接相连的。使用这个 JTAG 接口，可以把配置文件直接下载到 FPGA 中。在平时的实验中多数使用的就是这种配置方式。

- 主动配置

使用配置板上的 EPCS1 可以对 FPGA 进行主动配置。使用这种配置方式时，当第一次把配置文件配置到 EPCS1 中后，下次上电时 FPGA 会主动到 EPCS1 中寻找配置文件，找到的话就用 EPCS1 中的配置文件配置 FPGA。从 FPGA 角度来讲，是 FPGA 主动去 EPCS1 寻找配置文件，所以称作主动配置。使用这种配置方式的好处是对 EPCS1 配置一次后就可以不再另行配置 FPGA 了，因为 FPGA 可以主动的从 EPCS1 中找到配置文件配置自己。

- 被动配置

使用配置板上的 EPM7128AE 芯片及 AT45DB041B 芯片可以对 FPGA 进行被动配置。使用这种配置方式时，FPGA 不能主动的到 EPM7128AE 芯片及 AT45DB041B 芯片中去寻找配置文件，而是由 EPM7128AE 芯片“通知”FPGA 来找配置文件。从 FPGA 角度来讲，是 FPGA 被动去 EPM7128AE 芯片及 AT45DB041B 芯片寻找配置文件，所以称作被动配置。在没有主动配置方式之前使用这种被动配置方式来对 FPGA 芯片进行自动配置。

# 第四章 样机平台功能扩展——A/D、D/A 转换板

如第三章所述，我们研制的样机平台具有易扩展的特性。通过样机平台提供的功能扩展接口，本人对样机平台进行了功能扩展，通过增加 A/D、D/A 转换板，为样机平台加入了 A/D、D/A 转换功能，即使样机平台具有了模/数、数/模转换功能。本章详细对这部分实现及功能做详细介绍。

## 4．1 A/D、D/A 转换板

### 4．1．1 A/D、D/A 转换板组成部分

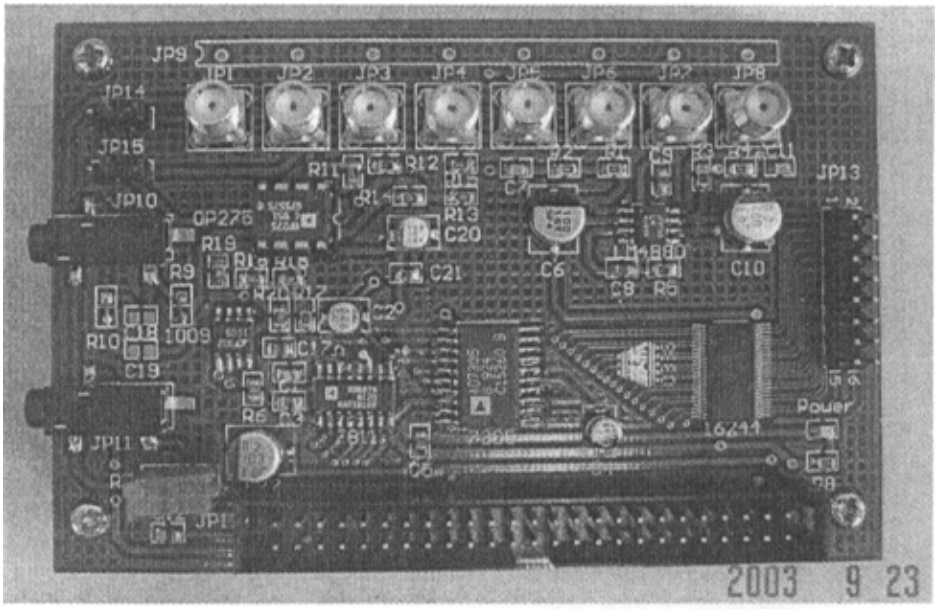



图 4.1－A/D、D/A 转换板

A/D、D/A 转换板的实物如图 4.1 所示。转换板主要由以下几部分组成：

1．输入、输出部分

这部分主要包括模拟信号输入、输出接口，数字信号输入接口。图片上部的 8 个金色的器件（JP1～JP8）是 SMA 模拟输入、输出端子，通过它们可以进行 4 路模拟信号的输入和 4 路模拟信号的输出。左侧的 2 个插孔（JP10、JP11）也是模拟输入、输出端口，通过它们可以进行 2 路模拟信号的输入和 2 路模拟信号的输出。其中一个作为模拟信号输入的插孔（JP10）与 SMA 端子中的 2 路（JP1、JP2）通过跳子共享 A/D 转换芯片 AD7811 的 2 个输入引脚。右侧的 16 引脚插排是数字信号输入接口。

2．功能转换部分

功能转换部分主要由 A/D 转换芯片 AD7811 和 D/A 转换芯片 AD7305 组成，同时还有几个辅助芯片，它们是为 AD7305 提供参考 2.5V 电压的芯片 1009；将

模拟输入信号进行运算放大的芯片 AD822；将模拟输出信号进行放大的芯片 LM4880。

3．与样机平台接口部分

进行 A/D、D/A 转换的芯片 AD7811 和 AD7305 的控制信号通过转换板下方的 50 芯插排与样机平台的 Cyclone 芯片连接，通过这个插排，Cyclone 芯片可以对转换芯片进行控制。

**4．1．2 A/D、D/A 转换板功能实现**

**1．A/D、D/A 转换器指标中，精度与分辨率**

在 A/D、D/A 转换时，将最低位增 1 所引起的增量和最大输入量的比称为分辨率。而转换精度可分为绝对转换精度和相对转换精度。所谓绝对转换精度是指每个输出电压接近理想值的程度。相对转换精度是更加常用的描述输出电压接近理想值程度的物理量。在 A/D 转换时，转换精度反映了 A/D 转换器的实际输出接近理想输出的精确程度。通常用数字量的最低有效位（LSB）来表示。而 A/D 转换器的分辨率表明了能够分辨的最小的量化信号能力，通常用位数来表示。

**2．A/D 转换**

对理想的 A/D 转换器，其模拟输入值和数值量输出的关系是：

N=[VA/VR]

式中：N：二进制数字；VA 输入模拟信号幅值；VR 模拟基准参考电压。

例：一个 A/D 转换器，VR＝5V，分辨率为 10 位，则：

VA＝0V，　N＝0000000000；

VA＝5V，　N＝1111111111；

VA＝2.5V，N＝1000000000；

VA＝1.25V, N＝100000000；

VA＝0.3V，　N＝111101；

**3．D/A 转换**

D/A 转换与 A/D 转换的原理正好相反，它把离散的数字输入量转换为连续的模拟输出量。例：一个 D/A 转换器，VR＝5V，分辨率为 8 位，则：

N＝00000000，　VA＝0V；

N＝11111111，　VA＝5V；

N＝10000000，　VA＝2.5V；

**4．A/D 转换器 AD7811**

AD7811 是 Analog 公司的一款 A/D 转换器，它具有如下一些关键特性：

- 4 个模拟输入通道
- 1 个数字输出通道
- 分辨率为 10 位
- 串行接口
- 2.5V 内部参考电压
- 外部参考电压范围为 1.2V～VDD
- 模拟输入范围为 0～Vref
- 可通过控制寄存器进行软件控制
- 电源供电范围为 2.7～5V

AD7811 它的引脚如图 4.2 所示。

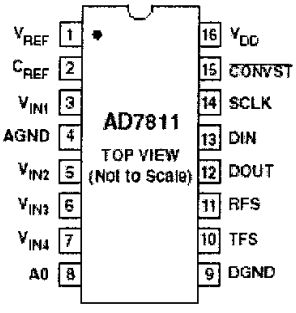



图 4.2－7811 引脚图

AD7811 通过它的控制寄存器来进行 A/D 转换。控制寄存器是 10 位的只写寄存器。当 AD7811 的 TFS 引脚收到一个下降沿信号时可以开始往控制寄存器里写内容。控制寄存器内的内容一直保持到下一次写入新的内容。

AD7811 的转换时序及其串行接口如图 4.3 所示

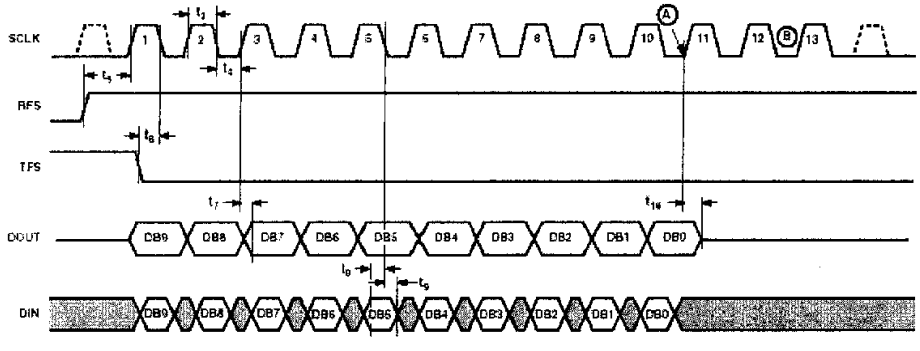



图 4.3－AD7811 时序

AD7811 进行模/数转换的关键就是控制好 nCONVST、SCLK、RFS、TFS、DIN、DOUT 这几个信号之间的时序关系。

**5. D/A 转换器 AD7305**

AD7305 是 Analog 公司的一款 D/A 转换器，它具有如下一些关键特性：

- 4 个模拟输出通道
- 8 个数字输入通道
- 分辨率为 8 位
- 并行接口
- 外部参考电压范围为 VSS～VDD
- 模拟输出范围为 VSS～VDD
- 电源供电范围为 2.7～5V

AD7305 引脚图如图 4.4 所示：

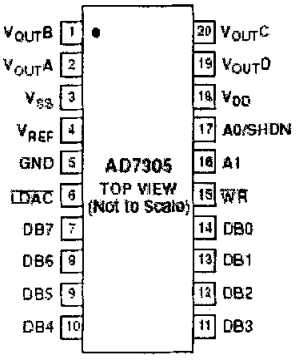



图 4.4－AD7305 引脚图

图 4.5 是 AD7305 的转换时序图。

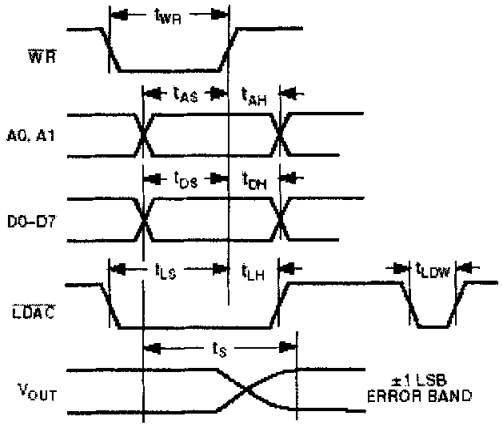



图 4.5－AD7305 的转换时序图

AD7305 进行数/模转换的关键就是控制好 nWR、A1、A0、nLDAC 这几个信号之间的时序关系。

**6．LM4880：音频放大器**

LM4880 把由 AD7305 输出的 VoutA、VoutB 模拟信号进行放大，然后把放大后的模拟信号送到模拟输出——耳机（转换板上的 JP11）。

**7．AD822：运算放大器**

由麦克风（转换板上的 JP10）输入的模拟信号有负电压，而 AD7811 的模拟输入端只能接受正电压，故对于麦克风输入的模拟信号要通过 AD822 进行放大，把输入信号进行线性“加法”处理：把输入信号“加”2.5V 的电压（输入信号电压范围：－2.5V ～2.5V）。故经 AD822 放大后的模拟输入的电压范围为：0～5V。

经 AD822 放大后的模拟信号作为 AD7811 的模拟输入，AD7811 的模拟输入的电压范围为：0～Vref，而这个 A/D、D/A 转换板上的 AD7811 的 Vref 接的是 VDD（5V），故可以满足其输入范围的要求。

**4．2 A/D、D/A 转换板功能验证**

对设计完成的转换板需要进行功能验证。本人通过如下实验对转换板进行了功能验证：由 JP10 输入 2 路模拟信号，经过 AD7811 转换成数字信号，之后，将转换后的数字信号经由 AD7305 再转换回模拟信号，由 JP11 输出。比较输入模拟信号和输出模拟信号，二者相同，说明转换板工作正常。具体实现是通过麦克风（JP10 插孔），将由计算机声卡输出的音频信号输入到转换板，经过 A/D、D/A 转换后由耳机（JP11 插孔）输出音频。

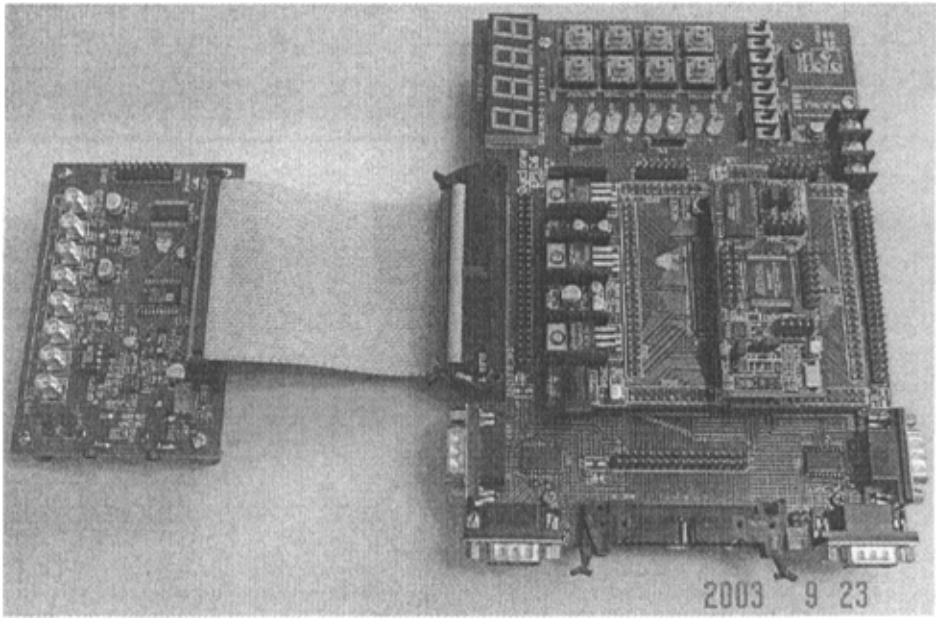



图 4.6－转换板与样机平台的连接

如前节所述，控制转换的关键是控制好转换芯片的控制信号。通过转换板上的 50 芯插排，由 Cyclone 中的逻辑来控制转换芯片。转换板与样机平台的连接实物如图 4.6 所示。

通过编写 VHDL 代码来实现控制转换芯片的逻辑。实现的 VHLD 代码如下所示：

```
LIBRARY IEEE;
USE   IEEE.STD_LOGIC_1164.all;
USE   IEEE.STD_LOGIC_ARITH.all;
USE   IEEE.STD_LOGIC_UNSIGNED.all;
--************************************************************
ENTITY AD IS

    PORT
    (
        clk_20M    :      in      std_logic;
        SCLK       :      out     std_logic;
        nCONVST    :      out     std_logic;
        DIN        :      out     std_logic;
        RFS        :      out     std_logic;
        TFS        :      out     std_logic;
        DOUT       :      in      std_logic;
        DB0_7      :      out     std_logic_vector(7 downto 0);
        A0         :      out     std_logic;
        A1         :      out     std_logic;
        nLDAC      :      out     std_logic;
        nWR        :      out     std_logic
    );
END AD;
--************************************************************
architecture a of AD is

signal   clk_count      :   std_logic_vector(4 downto 0);
signal   DIN_TMP        :   std_logic_vector(9 downto 0);
signal   DB_TMP         :   std_logic_vector(9 downto 0);
```

```
begin
    --count clock

    process(clk_20M)
    begin
        if clk_20M'event and clk_20M='1' then
            clk_count<=clk_count+1;
            if clk_count=29 then
                clk_count<="00000";
            end if;
        end if;
    end process;

    --7811

    SCLK<=clk_20M;

    process(clk_20M)
    begin
        if clk_20M'event and clk_20M='1' then
            if clk_count>=1 and clk_count<=13 then
                DIN_TMP<="0011000011";
            elsif clk_count>=15 and clk_count<=27 then
                DIN_TMP<="0011000111";
            end if;
        end if;
    end process;

    process(clk_20M)
    begin
        if clk_20M'event and clk_20M='1' then
            if (clk_count>=1 and clk_count<=10) or (clk_count>=15 and
clk_count<=24) then
                for i in 9 downto 0 loop
```

```
                DIN<=DIN_TMP(i);
            end loop;
        end if;
    end if;
end process;

nCONVST<='1';

RFS<='0' when clk_count=14 else
      '1';

TFS<='1' when clk_count=0   or (clk_count>=11 and clk_count<=14) else
      '0';

--7305

process(clk_20M)
begin
    if clk_20M'event and clk_20M='1' then
        if (clk_count>=1 and clk_count<=13)   then
            nWR<='0';
            nLDAC<='1';
            A0<='0';
            A1<='0';
        elsif clk_count>=15 and clk_count<=27 then
            nWR<='0';
            nLDAC<='1';
            A0<='1';
            A1<='0';
        else
            nWR<='1';
            nLDAC<='0';
        end if;
    end if;
```

```
    end process;

    process(clk_20M)
    begin
        if clk_20M'event and clk_20M='1' then
            if (clk_count>=1 and clk_count<=10) then
                for i in 9 downto 0 loop
                    DB_TMP(i)<=DOUT;
                end loop;
            end if;
        end if;
    end process;

    DB0_7<=DB_TMP(9 downto 2);

end a;
```

VHDL 代码编写完成后，模拟其波形，可以看到，模拟出的波形符合转换芯片的控制要求。把 VHDL 代码编译后生成的配置文件下载到 Cyclone 芯片，由计算机放出音乐，通过麦克风进入转换板，经过 A/D、D/A 转换，由转换板上的耳机可以听到计算机中放出的音乐，说明转换板可以正常工作。

# 第五章 汽车运行信息记录仪

## 5.1 汽车运行信息记录仪概述

汽车是当今世界上使用最为广泛的交通工具，同时，汽车交通事故也是对人类危害最为严重的事情之一。全世界每年会发生2400万起交通事故，2.4亿人会因此受伤。据估算，每年车祸中死亡的人数相当于每天坠毁一架波音737。

当发生交通事故后就要进行事故鉴定来分析事故原因、主要责任人等情况。然而，在我国，甚至在世界范围内，鉴定汽车交通事故的方法还很落后。比如当一辆汽车超越另一辆汽车时，在后车超过前车之后，一定要给被超车辆留足安全停车距离，如果车距太短，从而引起追尾事故，则主要责任应由超车的车来承担。但实际情况恰恰相反，这样的追尾事故的责任通常由被超车的那辆车来承担，原因就是鉴定交通事故方法的落后性，因为还没有办法来鉴定超车的那辆车是否在并道时保持了足够的车距。显然这样的事故鉴定是不合理的。

这种情况使人们联想到飞机的“黑匣子”。飞机发生事故后，人们做的第一件事是寻找“黑匣子”，因为飞机的“黑匣子”记录了飞机运行中的一些重要数据，从而当飞机发生事故时人们可以通过飞机的“黑匣子”记录的信息来分析事故原因。为什么汽车不能有“黑匣子”呢？至少它可以让警察和汽车制造商明白车祸前一刹那到底发生了什么。而传统的还原事故的方式是观察刹车痕迹及车身撞毁的程度，这其实很不可靠、很不科学。在飞机“黑匣子”的启发下，一种用于记录汽车信息的系统应运而生，这就是汽车信息系统。

汽车信息系统，亦名汽车黑匣子，是用于监测、记录、储存汽车在行驶中各种状态和数据的智能装置。它将微机应用的先进性、实用性与车辆运行状态统一起来，可以完整、准确地记录汽车行驶状态下的有关情况，将汽车行驶轨迹完整记录，并通过专用软件在电脑上再现，为分析、判断汽车驾驶状态和处理交通事故提供可靠准确的科学依据。汽车黑匣子在记录汽车运行数据的同时，还可以约束汽车驾驶员的行为，使汽车驾驶员在开车时会比平时更加小心，从而可以更多的避免交通事故。

汽车黑匣子的问世，为交通管理部门准确了解界定交通事故发生原因和公正处理提供了科学、权威的依据，如行驶速度、车辆所处的地理位置、电器系统、制动系统状况以及发动机系统的温度、油压等进行连续较长时间的记录。为汽车的科研、生产、日常维护以及对于及时查找突发性交通事故的原因，减少车辆故障提供有效的测试手段，为产品设计与故障分析提供依据。同时，汽车黑匣子也规范了汽车驾驶员的行为，可以有效的避免交通事故。汽车黑匣子对深入研究各类车辆的完善设计、故障分析、降低成本和交通管理、避免交通事故具有重要的实际作用及意义。

复旦大学 CAT 实验室在自行研制的样机平台的基础上，对样机平台的功能进行了扩展，设计了“汽车运行信息记录仪”系统。我们设计这个系统的目的是力图使得 NIOS 的特点和优势在汽车信息系统中得到最大的体现和发挥。

## 5．2 系统设计与实现

### 5．2．1 系统组成

整个系统可以分为两个部分：黑匣子和模拟器。

模拟器是为了在实验室中模拟实际中可采集的各种汽车信号。我们系统设计的模拟环境是使用一个连到计算机上的游戏用的方向盘，用这个游戏方向盘来模拟实际汽车的一些信号输入，如速度、方向、油门、转向灯等。

黑匣子部分是汽车信息系统的中心框架，主要是完成对各种信号的采集存储，有条件的判断事故，与主控中心通信等任务。

1．模拟器：

主要由主控板与 A_D 转换板组成，如图 5.1 所示。主控板通过串口接收上位机汽车方向盘（游戏手柄）的行驶信号，然后通过 A_D 转换板向黑匣子系统发送模拟信号（模拟汽车上的模拟量信号，如速度、方向等）和数字信号（模拟汽车上的数字量信号，如各种灯的状况，在模拟器中，由指示灯的明灭来显示）。

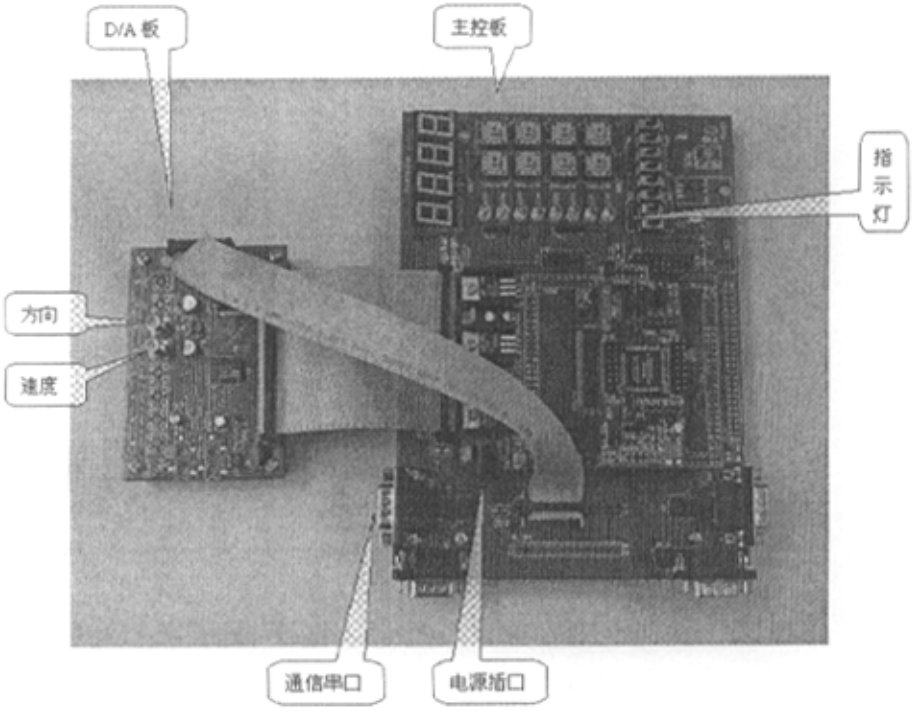



图 5.1—模拟器

2．黑匣子：

整个系统由主控板、配置板、A_D 转换板和状态板组成，如图 5.2 所示。通过 A_D 转换板的采集线与模拟器相关 A_D 转换板连接，完成信号的接收与存储。通过 GPS 卫星定位，给全系统对钟和定位信息。通过 GPRS 无线接入，发送事故报告短消息。通过串口将信息数据上传做进一步地分析处理。主控板通过缆线完成配置板中 FLASH 的固件数据更新。

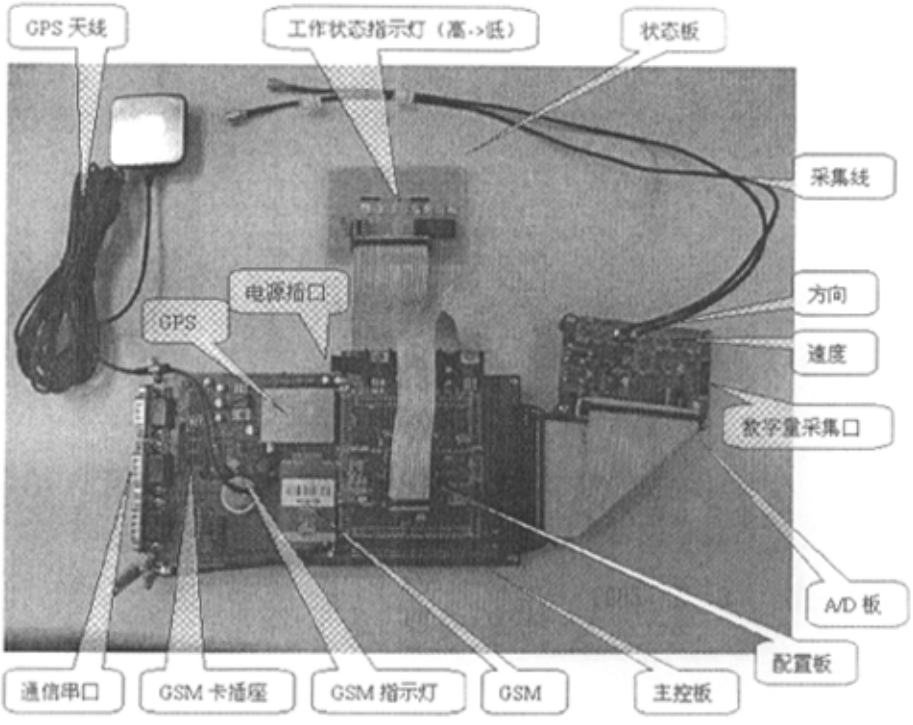



图 5.2－黑匣子

**5．2．2 具体实现**

1．完整、独立的 NIOS 系统

我们利用片内 ROM 保存事先编好的 Bootloader 程序，并把 Reset 地址指向该 ROM，这样在系统上电时，由 Bootloader 程序负责把用户程序从片外的 Flash 装入 SRAM 中，并跳到指定地址运行。

除了该系统能独立工作外，整个系统还是一个比较完整的嵌入式系统，包括了 SRAM，FLASH，显示，按键，串口，GPRS 无线接入，GPS 卫星定位等等，这样整个系统就证明了 NIOS 处理器是完全能够适应嵌入式系统开发的。

2．利用上位机软件可以实现系统软硬件更新

系统升级可分为固件升级和软件升级，软件升级在很多产品设计中已经得到应用，主要是采用通信链路把程序代码写入非易失存储器的方法来完成的。类似的方法也可以借鉴到固件升级当中来。Quartus II 软件可以生成多种描述硬件系

统的文件，如 POF、SOF、RBF 文件等，其中 RBF 文件内保存的是配置 FPGA 时输入到其配置接口的原始比特流。如果把该文件保存到 FLASH 中，升级时只需更新该文件就可以完成硬件升级。

ALTERA 的试验开发板上采用的 EPM7128＋AM29LV065 的方法，其中 EPM7128 中事先固化了一个状态机，用于上电时来配置 FPGA。但是 AM29LV065 是一款并行的 FLASH，不可避免的体积要大一些（相对串行 FLASH 来讲），由于汽车信息系统中对体积的要求是越小越好，所以我们决定采用串行 FLASH，这样我们就不能照搬原有状态机的设计，为此我们自己设计了一个用于配置 FPGA 的状态机，上电时从串行 FLASH 中读出数据送到 FPGA 中，配置完成后即释放对 FLASH 的控制权，这时 FLASH 可以由 FPGA 中的 NIOS 来控制，实际上，汽车信息系统中的汽车标识数据和其他一些信息就是保存在该 FLASH 中的。

3．定制用户 IP 并已集成到 SOPC 中

如前所述，NIOS 系统的优点表现在它的灵活性和可裁剪上，不仅仅是指它可以对系统提供的 IP 核进行取舍，而且在低层也提供了开放的 IP 接口，使得高级用户可以把自己的 IP 集成到 SOPC BUILDER 中去，有效地解决了系统设计中的复用问题。

在我们的系统中，SRAM 和 FLASH 的接口 SOPC 都没有提供相应的 IP，但是提供了类似的 IP，在对已有的 PTF 文件（SOPC 的配置文件）考察后，我们写出了自己的 IP，并集成到 SOPC 中。

4．模块化的硬件结构，方便用户定制

我们在进行系统设计时，一个主导思想就是模块化，以尽可能方便用户定制。在硬件设计上体现在系统分成了配置板，底板，控制板，A_D 转换板四块。

目前市场上的类似产品一般都是功能完全固定，这样就会造成一些用户不需要的功能也被强加其上，增加了产品成本同时也加重了用户负担，例如对于有些应用是完全没必要使用 GPS 和 GPRS 这种较昂贵设备的，但产品一旦定型就不便修改了，所以要解决这个问题，从产品设计阶段就应开始着手考虑，这样不可避免要增加设计复杂度，但如果因此而获得用户的认可，也还是值得的。

5．利用 GPS 对采样数据校正进行定位

在方案设计中我们加入了 GPS 全球定位设备，主要是用于给全系统对钟，另外也参考它给出的方位和速度，和采集到的信号做比较，进行取舍。此外也可采取信息融合的算法来加强定位的精确度和可靠性。

由于整个系统是分布的，终端在汽车上，如果采用分离时钟，再经过一段时间后，车辆之间就会产生计时偏差，这对于事故鉴定和模拟是不允许的。所以只

有采用全球定位系统的报时，才有可能将各个离散点的时钟统一起来。不过，由于 GPS 信号不能保证每时每刻都被接收到，所以实用的系统应该自备一个高精度独立时钟，这样就能保证系统在没有 GPS 的情况下也能够正常工作。

6．可调整的数据采样率

目前市场上的类似产品都采用固定采样率进行信息收集（每秒一次），这样在出现意外或者紧急情况时，会显得数据量过小，对事故状态的细节描述能力较差，在事后进行分析和鉴定时，显得比较苍白。但过快的采样率会造成数据存储量的成倍增长，提高了成本。

考虑到以上问题，我们在设计时进行了折衷，在正常工作时，系统按照每秒一次的采样率开始采样，一旦出现事故（按照预设条件判定），系统会自动加快采样率，一旦判定条件失效，仍然恢复到默认采样率。这样就较好的解决了数据存储量和采样率间的矛盾。

**5．3 系统模拟运行**

系统硬件连接成功后，运行控制黑匣子的计算机上的软件，对“GSM 通信”进行设置，如图 5.3 所示。一旦出现事故，黑匣子就会向该号码发送短消息，并报告经度和纬度。


图 5.3－设置“GSM 通信”

执行模拟器部分软件，软件下方会给出速度和方向的历史曲线，用于和车载信息系统描绘的曲线进行对比，如图 5.4 所示。

启动控制黑匣子的软件，选择菜单“联机通信 / 采集数据操作”，出现如图 5.5 所示界面。选择读取全部数据，程序会从下位机读出采集数据并显示出来，点击保存文件按钮，把数据存入文件。选择菜单“事故管理 / 数据分析”，出现如图 5.6 所示界面。也可以选择菜单“事故管理 / 现场模拟”菜单，界面如图 5.7 所示。

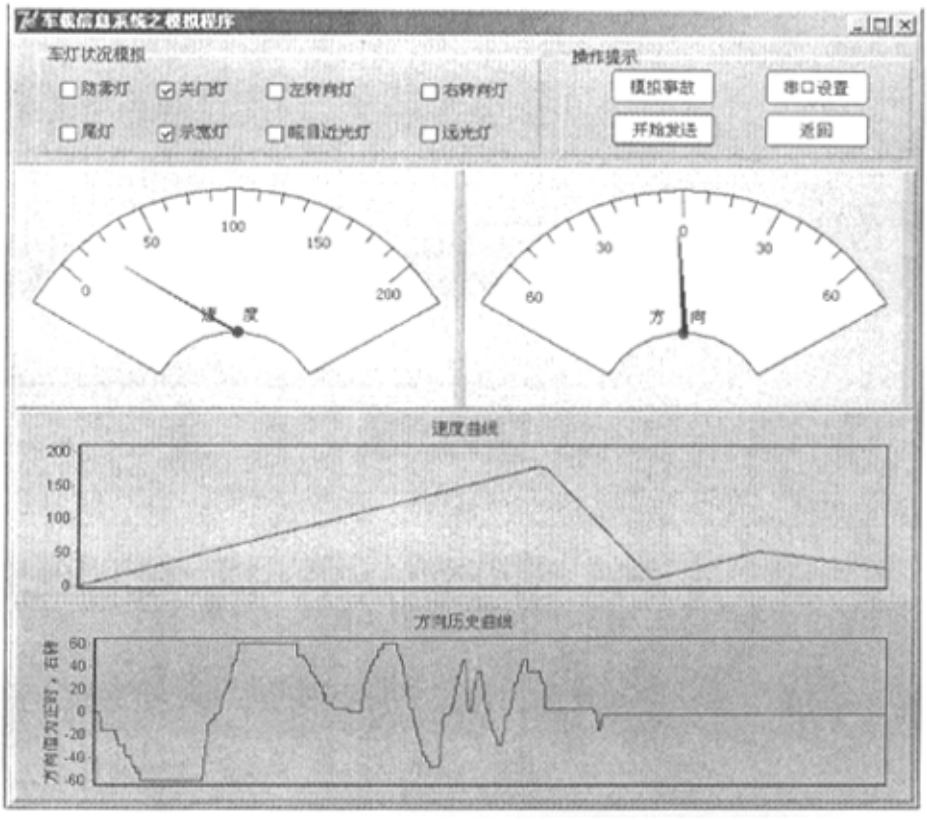



图 5.4－模拟器界面


图 5.5－控制黑匣子软件界面

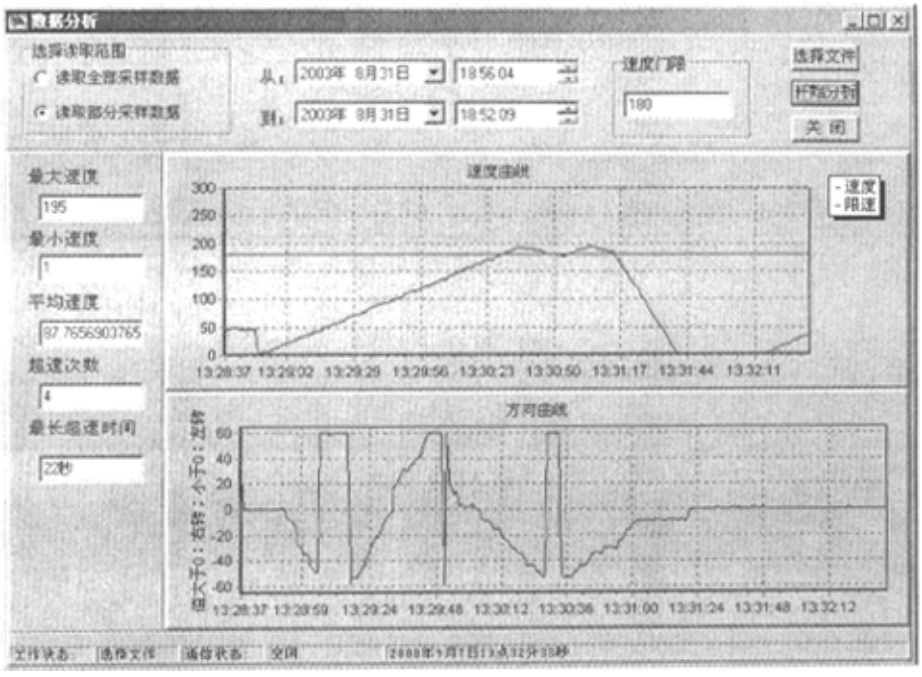



图 5.6—“事故管理 / 数据分析”界面

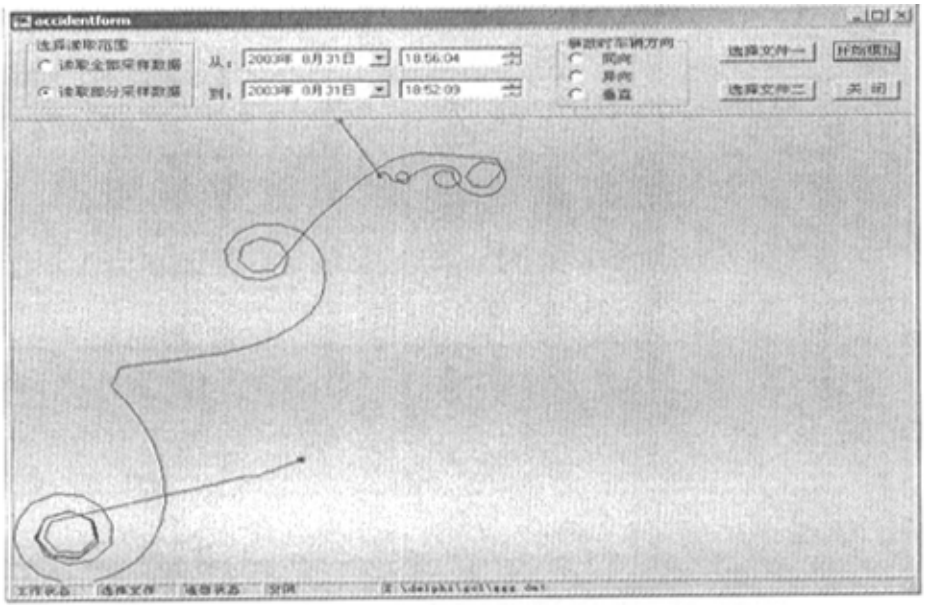



图 5.7—“事故管理 / 现场模拟”界面

# 第六章 结语及展望

本文首先对嵌入式系统设计做了一下概要地介绍，接下来对 Altera 公司提出的基于 NIOS 的 SOPC 技术做了比较详细地介绍，其中详细介绍了 NIOS 软核及 SOPC 技术。之后介绍了复旦大学 CAT 实验研制的一台快速样机平台及对样机平台进行的 A/D、D/A 功能扩展。最后介绍了在这个样机平台上开发的一个基于 NIOS 的 SOPC 技术的实例——汽车运行信息记录仪。

复旦大学 CAT 实验室目前主要研究快速样机生成技术，具体的说是 SOPC 技术，这在嵌入式系统设计方面属于新的领域。实验室希望在 SOPC 技术的基础上进行扩展，完整地开发出一套快速样机生成系统。

目前，实验室已经成功地开发出一款样机平台，并且在样机平台上成功地开发出具有实际应用意义的系统——汽车运行信息记录仪。在这个快速样机生成系统的其他方面，实验室的科研工作也取得了部分可喜的成绩，比如在系统建模方面，实现了 UML 部分功能向 SystemC 的转换工作；在处理器软核方面，对 RISC IK 软核向 Altera FPGA 的移植工作也取得了较大进展。

在今后的科研工作中，实验室的工作重点会放在系统建模、IP 核接口技术及 Linux 操作系统的移植方面。

在系统建模方面，现已有一些建模语言，如 UML、SpecC、VHDL、SystemC 等。这里考虑用 SystemC 进行系统建模，是基于如下考虑：SystemC 就是在 C++ 中加上硬件类库和仿真核，这使得 SystemC 既可进行软件建模，又可进行硬件建模，使其成为一个统一建模语言，而且任何一个熟悉 C++语言的用户只要了解类库中各种类引入的语义就可以用 SystemC 编程。这充分利用了现有的条件，如知识的积累、各种工具的支持等，使得用 SystemC 进行系统建模成为一个很好的选择。

IP 技术是快速样机生成技术中的一个核心技术，这是因为现存大量可用的 IP 核，使用这些现存的 IP 核，可以避免重复开发，这样可以缩短系统的开发时间，进而达到快速生成系统的目的。但是，现存的各种 IP 核并不都是来自同一开发商，所以在开发系统时，如何把来自不同开发商的 IP 核集成到一起是个关键问题。实验室今后的一个工作重点就是研究 IP 核之间的接口技术，在这种技术的基础上，可以方便地集成来自各方的 IP 核，达到高效、迅速地开发系统的目的。

嵌入式系统如果有了操作系统的支持，则可以在其上开发出更多的应用，使嵌入式系统更具有生命力。目前，嵌入式 Linux 已经成功地被移植到一些嵌入式系统中，例如基于 ARM 的嵌入式系统。但在基于 NIOS 的嵌入式系统中，嵌入式 Linux 的移植工作还是一个崭新的任务。所以实验室也把嵌入式 Linux 向 NIOS 的移植作为今后科研的一个工作重点。

我们坚信，在实验室师生的一致努力下，实验室一定能研制出一套出色的快速样机生成系统，为国家计算机领域的科研贡献一份力量！

# 参考文献

[1] Ralf Niemann. Hardware-Software Co-design for Data Flow Dominated Embedded Systems, Boston: Kluwer Academic Publishers, 1998.

[2] Carolyn Kuttner. Hardware-Software Co-design Using Processor Synthesis. IEEE Design & Test of Computer. Vol.13, No.3, Pages:43-52. Fall 1996.

[3] Rochit Rajsuman 著，于敦山，盛世敏，田泽译著。SOC 设计与测试。北京航空航天大学出版社，2003 年。

[4] Rincon, A.M., Cherichetti, G., Monzel, J.A., Stauffer, D.R. and Trick, M.T. Core design and system-on-a-chip integration. Design & Test of Computers, IEEE, Volume: 14, Issue: 4, Oct.-Dec. 1997. Pages:26-35.

[5] Altera Corporation. NIOS Embedded Processor 32-Bit Programmer's Reference Manual. http://www.altera.com/literature/lit-nio.html.

[6] Altera Corporation. Avalon Bus Specification Reference Manual. http://www.altera.com/literature/lit-nio.html.

[7] Altera Corporation. SOPC Builder. http://www.altera.com/literature/lit-nio.html.

[8] Altera Corporation. NIOS Tutorial. http://www.altera.com/literature/lit-nio.html.

[9] 方茁，彭澄廉，陈泽文。基于 NIOS 的 SOPC 设计。计算机工程与设计。2004 年，第 25 卷，第 4 期，P504-P507。

[10] 方茁，邱卫东，彭澄廉。SOPC 设计中的定制指令。Altera 2003 年全国大学教师会议。

## 致谢

首先感谢我的导师彭澄廉教授。在学业上，彭老师对我们谆谆教诲、悉心指导，我们在学业上每前进一步都倾注着彭老师的心血。彭老师严谨的治学态度、认真的工作作风将是我们今后工作、学习的典范。在生活上，彭老师也是时时刻刻关心我们，尽量帮助我们解决各种生活中的困难。可以说，在我三年的研究生学习、生活期间，彭老师无论是从学业上还是从生活上都给予了我巨大的帮助。

感谢吴百锋教授。吴老师治学严谨，对待工作严肃认真，学术造诣深厚，这些都给我们留下深刻的印象。在做吴老师的助教期间，吴老师更是对我进行了悉心地指导，使我顺利完成助教工作。

感谢陈泽文高级工程师。陈老师对待工作认真负责，在计算机硬件方面的造诣深厚。陈老师在工作上给予我很大的指导和帮助，使我在实践工作中学到很多知识。

感谢孙晓光副教授。孙老师严谨的治学态度和平易近人的作风给我们很深的印象。孙老师在讨论班上给我们讲述的知识使我们受益匪浅。

感谢周博博士。周博博士对待学术严肃认真，对待工作热情积极，对待同学热心帮助。周博博士在实验室工作中起着重要的带头作用，在他地带动下，实验室的工作成绩卓著，大家从中都受益匪浅。

感谢周湘赟、邱卫东、陈燕三位博士。三位师兄、师姐在学习、工作中都给予了我很大的帮助。

感谢梅宏亮、黄新生同学。我们三人是实验室中同一年级的同学。在学习、生活中三人相互关心、相互帮助，结下了深厚的友谊。

感谢周学功、薛志军、朱琦同学。在学习、工作中多次和三位同学进行讨论，使我深受启发。

特别要感谢我的父亲、母亲和姐姐。家人给我的无穷的支持和关怀是我工作、学习、生活的动力、保障。他们是我永远的精神支柱和前进动力。

最后感谢所有关心和帮助过我的人。

方茁

2004 年 5 月于复旦园

# 论文独创性声明

本论文是我个人在导师指导下进行的研究工作及取得的研究成果．论文中除了特别加以标注和致谢的地方外，不包含其他人或其它机构已经发表或撰写过的研究成果．其他同志对本研究的启发和所做的贡献均已在论文中作了明确的声明并表示了谢意．

作者签名：[signature] 日期：2004.6.2

# 论文使用授权声明

本人完全了解复旦大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。保密的论文在解密后遵守此规定．

作者签名：[signature] 导师签名：[signature] 日期：2004.6.2