**1.SOPC的设计技术是现代计算机辅助设计技术，EDA技术和大规模化成电路高度发展的产物。**

**2.SOPC技术的目标就是试图将尽可能大而完整的电子系统，包括嵌入式处理器系统，接口系统，硬件协处理或加速器系统，DSP系统，数字通信系统，存储电路以及普通数字系统，在单一FPGA中实现。**

**3.使得所涉及的电路系统在其规模，可靠性，体积，功耗，性能指标，上市周期，开发成本，产品维护及其硬件升级多方面最优化**

**SOPC在电子设计技术上给出了一种以人的基本能力为依据的软硬件综合解决方案**

* **由于同时涉及底层的硬件系统设计和相应的软件设计，在系统优化方面有了前所未有的自由度。**
* **SOPC技术使开发者更能动地在软件系统的综合与构建两个方面有了充分发挥自己创造性和想象力的的巨大空间，从而使得从多角度，多因素和多结构层面上大幅度优化自己的设计成为可能。**
* **事实上，诸如单片机，DSP或ARM系统等基于传统开发技术的设计流程而言，不存在严格意义上的硬件设计，而只有软件设计。**
* **这是因为，一旦方案确定，硬件系统的核心器件是现成的，功能是确定的，结构是固定的，指令系统是不可更改的，从而导致硬件组织方案和连接方案是限定的，用户只能被动地遵循和适应，这时的硬件“设计”只能流于拼接和连接。**
* **系统构成后的唯一性任务是依据既定的指令系统来编程，除了系统功能和算法可以通过软件改变外，系统的性能和指标已无从改变，设计者的创新能力，想象力和优化设计能力的发挥空间已被选定的硬件性能所界定。**
* **没有了创造，更谈不上超越了，进而导致系统的综合性能基本取决于器件原有的性能和指标。**
* **换言之，传统技术是以既定的硬件及其性能而非以人的能力为主轴的技术。**
* **这样不难明白，在这个以硬件决定“创造”的世界里，为什么优秀的8031单片机出现后，仍然层出不穷地涌现出其他功能和性能的单片机。**
* **而同样是DSP处理器，G5X系列与G6X系列器件则把开发者带到完全不同的技术领域和功能范畴。**
* **显然，硬件的可设计性和用户目标的适应性在系统性能指标上占有更大份额，而SOPC技术的优势正是设计者本身而非器件设计商去更有效地占据这一份额。**

**SOPC在应用和知识构成上，达成了一种有机融合**

* **基于SOPC的结构特点，SOPC系统的开发对于设计者的知识范围有了更高的要求。**
* **除了必须了解基本的EDA软件，硬件描述语言的FPGA器件的相关知识外，还必须熟悉计算机组成与接口，汇编语言和C语言，DSP算法，数字通信，嵌入式系统开发，嵌入式操作系统，片上系统构建与测试等知识。**
* **这样，SOPC技术的普及必将有力地推动电子信息及工程类各学科分支与相应的课程类别的融合。**
* **这样融合在SOPC技术中已不是简单的合并与算术相加，而是有机的融汇和充满设计者创新意识的升华。**
* **例如可以利用Matlab/DSPBuilder/Quartus II完成纯硬件的DSP算法模型以及实现，进而构成嵌入式系统外围接口的协处理模块，再进一步构成软件程序的精简指令，从而实现完全自主，正在意义上的硬件设计和系统优化**

1. **Altrea公司于21世纪初推出SOPC技术，其SOPC称为\_\_\_可编程片上系统\_\_**
2. **Altera公司的SOPC集成了\_硬核\_或软核CPU、DSP、锁相环（PLL）、\_存储器 \_、I/O接口及可编程逻辑，SOPC开发工具将软、\_\_硬件\_\_的设计结合起来，为用户提 供很好的开发环境**
3. **SOPC的设计包括，设计准备、设计输入、设计编译、设计仿真、器件编程，\_硬件\_ 测试等设计流程。**
4. **用FPGA实现DSP并不是指用FPGA来构造一个\_DSP\_芯片，是直接用\_\_FPGA\_\_硬件来实现DSP功能。**
5. **Altera公司的DSP Builder可以帮助开发者完成基于FPGA的DSP设计，自动完成大部分的设计过程和仿真，直至把设计文件下载至\_\_\_FPGA \_\_中**
6. **DSP Builder将用户设计的DSP模块转换成\_\_硬件\_\_\_描述语言（HDL）,最终在FPGA上实现。**

**简答：**

**1、Avalon总线具有哪两种接口类型？说明其特点**

**答：Avalon总线接口可分为两类Slave和Master。**

**Slave是一个从接口，而Master是主控接口，区别在与对总线的控制，Master接口具有对总线的控制权，而Slave是被动的。**

**2、说出可以帮助用户完成基于FPGA的DSP系统的设计工具。使用该设计工具进行DSP系统开发必须要安装哪两种软件？**

**答：用户首先利用Matlab进行DSP模块设计，然后用DSP Builder将用户设计的DSP模块转换成硬件描述语言（HDL），最终在FPGA上实现。使用Matlab/DSP Builder进行DSP系统的开发必须要安装Matlab和DSP Builder软件。**

**3、在SOPC Builder的组件库中，提供的Nios II系列微处理器有几个成员？通常选用的成员是哪种类型？**

**答：提供了三个成员供选择：**

**（1）NiosⅡ/e(经济型)成员，具有占用最小逻辑的优化，占用600～700逻辑元件。**

**（2） NiosⅡ/f(快速型)成员，具有高性能的优化，占用1400～1800逻辑元件。**

**（3）NiosⅡ/s(标准型)成员，在占用逻辑和高性能优化方面的性能居中，占用1200～1400逻辑元件。**

**4、Avalon总线是由哪个公司提出的？在基于FPGA片上系统中有哪些用途？**

**答：Avalon总线是ALTERA公司提出的，用于在基于FPGA片上系统中连接片内处理器和片内外设的总线结构。连接到Avalon总线的设备分为主从设备，并各有其工作方式。**

**5、简要说明硬件抽象层（HAL）的用途及HAL支持的外围设备**

**答：用途：作为支持Nios II处理器系统的软件包，为用户的嵌入式系统上的外围设备提供了与之相匹配的接口程序。用户不用自己建立或复制HAL文件，也不需要编辑HAL中的任何源代码，使用时只需要在C/C++源程序中指明代表这些接口程序的库函数包含的头文件即可**

**支持的外围设备：①字符模式设备，包括UART核、JTAG核和LCD16207显示控制器。**

**②Flash存储器设备，包括通用Flash接口芯片和Altera EPCS串行配置设备控制器。**

**③文件子系统，包括只读文档系统。**

**④定时器设备，包括定时器核。**

**⑤DMA设备，包括DMA控制器核。**

**⑥以太网设备，包括LABN91C111以太网MAC/CHY控制器。**

**6、Nios II IDE调试分为几个步骤？分别是哪些步骤？**

**答：Nios II IDE调试分为四个步骤，分别是新建软件工程、编译工程、调试工程和运行工程。**

**7、锁相环电路SDRAM\_PLL在NiosⅡ系统中的作用是什么？**

**答：给外部时钟倍频给cpu。**

**8、在DE2开发板上，有哪些常用的支持Nios II系统的外部设备？**

**答：在DE2开发板上，支持NiosII系统的外部设备包括SDRAM、SRAM、FLASH、LCD、七段数码管、开光二极管、按钮开关、电平开关、VGA、ISP1362、I²C器件等。**

**9、简述HAL系统库的构成。**

**答：HAL可以为与硬件进行通信的程序提供简单的设备驱动接口。HAL系统库为用户提供下列支持：**

**①与ANSIC合成标准库，提供类似的C语言的标准函数库。**

**②提供访问系统中每个设备的驱动程序。**

**③提供标准的接口程序。④完成对处理器的初始化。 ⑤完成对系统外围设备的初始化。**

**10、试简要说明Nios II的IDE与SDK调试方法的区别。**

**答：IDE调试方式面向对象，通过直观的软件界面上的按钮来完成调试的各种操作，但编译程序和进入运行程序的时间较长；SDK调试方式是面向过程，通过各种命令的执行来完成程序的调试和运行，但编译程序和进入运行程序的时间较短。**

**11、简要说明DSP Builde的用途及特点。**

**答：用途：帮助开发者完成基于FPGA的DSP设计，自动完成大部分的设计工程和仿真，直至把设计文件下载至FPGA中。**

**特点：应用灵活，成本低，但在速度上比用硬件实现的DSP要慢很多。**

**12、说出生成Nios II硬件系统要完成的主要工作。**

**答：新建工程→添加CPU→添加外部设备→锁定引脚→编辑顶层文件→硬件调试。**

**13、试简述Quartus II原理图输入法的设计流程。**

**答：图形编辑输入设计文件→编译设计文件→仿真设计文件→编程下载设计文件。**

**14、请你以文字或图示方法说明Nios Ⅱ嵌入式系统软、硬件开发流程。**

**答：硬件开发：新建SOPC设计项目→加入Nios II系统组件→调整SDRAM地址→编辑 Nios II顶层文件→引脚锁定→再次编译顶层设计文件。**

**软件开发：新建软件系统→编辑工程→调试工程→运行工程。**

**15、什么是SOC？什么是SOPC？两者有何区别？**

**SOC: System on Chip的缩写，指在单片上集成系统级多元化的大规模功能块，从而构成一个能够处理各种信息的集成系统。**

**SOPC: System-on-a-Programmable-Chip，即可编程片上系统。用可编程逻辑技术把整个系统放到一块硅片上，称作SOPC。**

**16、SOPC和ARM、DSP各有什么特点？**

**答：ARM处理器是目前应用较广的典型32位RISC处理器，和另外两种处理器相比，其特点在于集成功能模块多、在系统中的控制能力强，产品线丰富、开发工具成熟。**

**DSP处理器转为数字信号处理而设计，在芯片的硬件设计中已经加入了对数字信号常用算法的支持和优化，相对ARM处理器，其控制功能较差。**

**SOPC是在FPGA基础上构建的微控制器，各种性能和功能和ARM处理器基本类似，但由于构建于FPGA，因此还可以通过定制指令的方式为处理器配置专用的硬件加速器，从而在不同时间内针对不同的处理目标，向FPGA配置相应的硬件模块。**

**17、简述Nios II 软核CPU与普通CPU的比较优势？**

**答：Nios II系列支持使用专用指令。专用指令是用户增加的硬件模块，它增加了算术逻辑单元（ALU）。用户能为系统中使用的每个Nios II处理器创建多达256个专用指令，这使得设计者能够细致地调整系统硬件以满足性能目标。**

**同时，Nios II系列支持60多个外设选项，开发者能够选择合适的外设，获得最合适的处理器、外设和接口组合，而不必支付根本不使用的芯片功能。**

**18、Avalon总线与普通总线相比有更高的数据吞吐量，特别在多CPU系统中。为什么Avalon总线会有更好的性能？**

**答：Avalon总线支持多个总线主外设，允许单个总线事务中在外设之间传输多个数据单元。Avalon主从之间的交换是建立端口仲裁技术上的:当多个主外设同时要求访问一个端口时，从端口仲裁决定哪一个主外设取得访问权。**

**从端口仲裁具有以下两个优点:**

**（1）仲裁的细节被封装在Avalon总线内部。因而，主从外设的接口是一致的，与总线的主从外设的数量无关。每个主外设到总线的接口都与总线上是否还有其他主外设无关；**

**（2）多个主外设只要不是在同一总线周期访问同一个从端口，便可同时执行多个总线传输；**

**19、什么是片内外设？什么是片外外设？这两类外设的主要区别是什么？（**

**答：片内、外设是两个概念，片内指做成芯片的集成电路内部，简称片内；外设是外部设备的简称，是指集成电路芯片外部的设备。集成电路芯片与外部设备的连接一般需要专门的接口电路和总线的连接（包括控制总线路、地址总线和数据总线等）。**

**由于大规模集成电路的技术发展得很快，现在许多芯片在制造时已经能够将部分接口电路和总线集成到芯片内部。对于这部分电路与传统的接口电路和总线是有区别的，为了加于区别可以称之为片内外设，其含义是在集成电路芯片内部集成的用于与外部设备连接的接口电路和总线。**

**简单的说，“片内外设”是芯片内部用于与外部设备连接的接口电路和总线。**

**片外外设则被设计在芯片之外，作为单独的一个接口设备，通过总线方式或I/O口与处理器进行连接。**

**20、简述操作系统实现任务调度的两种典型方法及实现原理，uC/OS使用哪种方法进行任务调度？**

**答：第一种为优先级调度，首先为每个任务或进程指定优先级，操作系统根据优先级的高低，来决定处理各程序的先后次序。如果即将被运行的进程的优先级比正在运行的进程的优先级高，则系统可以强行剥夺正在运行的进程的CPU，让优先级高的进程先运行。**

**第二种为时间片轮转，每个进程被分配一个时间段，称作它的时间片，即该进程允许运行的时间。如果在时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。**

**20、32位RISC处理器具有哪些典型特点？**

**答：（1）格式统一，功能简单，数量有限的指令系统；**

**（2）采用大量的寄存器，使大部分指令操作都在寄存器之间进行，提高了处理速度**

**（3）通过优化指令流水线提高性能**

**21、简述在NIOS中进行中断编程的三个步骤？**

**一般分为一下3个步骤**

**（1）定义中断服务子程序**

**（2）安装中断服务子程序，使用系统函数将编写好的中断服务子程序入口地址**

**安装到系统的中断向量表中**

**（3） 使能中断，设置相关的中断控制寄存器，中断屏蔽寄存器等，使能中断。**

**22、举出3种常见的嵌入式操作系统，说明其特点。**

**（1）Windows CE ,为桌面WINDOWS的嵌入式版本，其特点为开发工具完善，平台兼容性强，软件资源丰富，开发速度较快，缺点在于内核需要付费使用，对硬件资源消耗也较大**

**（2）嵌入式LINUX，其中包含了RT-LINUX uClinux等不同版本，具有免费、开源、完整的网络协议栈支持等优点，但开发工具还不完善，配套软件支持相对较差，兼容性较差**

**（3）uC/OS相对来说是一个轻量级的操作系统，仅具有基本的任务调度等功能，无文件系统、无网络协议等配套软件，对硬件要求低，同时也具有开源、免费等优点。**

**23、简述uC/OS操作系统任务调度的基本原理？任务优先级分配的基本原则是什么？**

**uC/OS采用基于优先级的任务调度方式，首先为每个任务或进程指定优先级，操作系统根据优先级的高低，来决定处理各程序的先后次序。如果即将被运行的进程的优先级比正在运行的进程的优先级高，则系统可以强行剥夺正在运行的进程的CPU，让优先级高的进程先运行。**

**优先级分配时，通常将占用CPU时间最长的任务分配较低的优先级，将需要及时处理，并且占用时间不长的任务分配较高的优先级。 21. 什么是PWM，简述下图PWM模块的实现原理**

#include "sys/alt\_stdio.h"

//#include "altera\_avalon\_timer\_regs.h"

#include "system.h"

#include <io.h>

#define f 1000000

#define rhythm 5000000

#define ff f\*50

/\*\*\*\*\*节拍宏定义\*\*\*\*\*/

#define \_1 rhythm\*4

#define \_1d rhythm\*6

#define \_2 rhythm\*2

#define \_2d rhythm\*3

#define \_4 rhythm\*1

#define \_4d (rhythm\*3)/2

#define \_8 (rhythm\*1)/2

#define \_8d (rhythm\*3)/4

#define \_16 rhythm\*1/4

#define \_16d (rhythm\*3)/8

#define \_32 (rhythm\*1)/8

/\*\*\*\*\*低音宏定义\*\*\*\*\*/

#define \_1do (ff/131)/2

#define \_1dor (ff/139)/2

#define \_1re (ff/147)/2

#define \_1rer (ff/155)/2

#define \_1mi (ff/165)/2

#define \_1fa (ff/175)/2

#define \_1far (ff/185)/2

#define \_1sol (ff/196)/2

#define \_1solr (ff/207)/2

#define \_1la (ff/220)/2

#define \_1lar (ff/233)/2

#define \_1si (ff/247)/2

/\*\*\*\*\*中音宏定义\*\*\*\*\*/

#define \_do (ff/262)/2

#define \_dor (ff/277)/2

#define \_re (ff/294)/2

#define \_rer (ff/311)/2

#define \_mi (ff/330)/2

#define \_fa (ff/349)/2

#define \_far (ff/370)/2

#define \_sol (ff/392)/2

#define \_solr (ff/416)/2

#define \_la (ff/440)/2

#define \_lar (ff/466)/2

#define \_si (ff/492)/2

/\*\*\*\*\*高音宏定义\*\*\*\*\*/

#define \_do1 (ff/523)/2

#define \_do1r (ff/554)/2

#define \_re1 (ff/579)/2

#define \_re1r (ff/740)/2

#define \_mi1 (ff/651)/2

#define \_fa1 (ff/695)/2

#define \_fa1r (ff/740)/2

#define \_sol1 (ff/784)/2

#define \_sol1r (ff/830)/2

#define \_la1 (ff/880)/2

#define \_la1r (ff/932)/2

#define \_si1 (ff/983)/2

/\*\*\*\*\*音强宏定义\*\*\*\*\*/

#define song\_size 119

#define mute(tone) (tone)>>3

#define loud(tone) (tone)>>2

/\*\*\*\*\*乐曲简朴程序表示\*\*\*\*\*/

int huanlesong[song\_size][3]={

{\_mi,\_4,loud(\_mi)},

{\_mi,\_4,mute(\_mi)},

{\_fa,\_4,mute(\_fa)},

{\_sol,\_4,mute(\_sol)},

{\_sol,\_4,loud(\_sol)},

{\_fa,\_4,mute(\_fa)},

{\_mi,\_4,mute(\_mi)},

{\_re,\_4,mute(\_re)},

{\_do,\_4,loud(\_do)},

{\_do,\_4,mute(\_do)},

{\_re,\_4,mute(\_re)},

{\_mi,\_4,mute(\_mi)},

{\_mi,\_4d,loud(\_mi)},

{\_re,\_8,mute(\_re)},

{\_re,\_2,mute(\_re)},

{\_mi,\_4,loud(\_mi)},

{\_mi,\_4,mute(\_mi)},

{\_fa,\_4,mute(\_fa)},

{\_sol,\_4,mute(\_sol)},

{\_sol,\_4,loud(\_sol)},

{\_fa,\_4,mute(\_fa)},

{\_mi,\_4,mute(\_mi)},

{\_re,\_4,mute(\_re)},

{\_do,\_4,loud(\_do)},

{\_do,\_4,mute(\_do)},

{\_re,\_4,mute(\_re)},

{\_mi,\_4,mute(\_mi)},

{\_re,\_4d,loud(\_re)},

{\_do,\_8,mute(\_do)},

{\_do,\_2,mute(\_do)},

{\_re,\_4,loud(\_re)},

{\_re,\_4,mute(\_re)},

{\_mi,\_4,mute(\_mi)},

{\_do,\_4,mute(\_do)},

{\_re,\_4,loud(\_re)},

{\_mi,\_8,mute(\_mi)},

{\_fa,\_8,mute(\_fa)},

{\_mi,\_4,mute(\_mi)},

{\_do,\_4,mute(\_do)},

{\_re,\_4,loud(\_re)},

{\_mi,\_8,mute(\_mi)},

{\_fa,\_8,mute(\_fa)},

{\_mi,\_4,mute(\_mi)},

{\_re,\_4,mute(\_re)},

{\_do,\_4,loud(\_do)},

{\_re,\_4,mute(\_re)},

{\_1sol,\_2,mute(\_sol)},

{\_mi,\_4,loud(\_mi)},

{\_mi,\_4,mute(\_mi)},

{\_fa,\_4,mute(\_fa)},

{\_sol,\_4,mute(\_sol)},

{\_sol,\_4,loud(\_sol)},

{\_fa,\_4,mute(\_fa)},

{\_mi,\_4,mute(\_mi)},

{\_re,\_4,mute(\_re)},

{\_do,\_4,loud(\_do)},

{\_do,\_4,mute(\_do)},

{\_re,\_4,mute(\_re)},

{\_mi,\_4,mute(\_mi)},

{\_re,\_4d,loud(\_re)},

{\_do,\_8,mute(\_do)},

{\_do,\_2,mute(\_do)},

};

//延时函数（延时时间为(2+2\*i)us）

void delay(alt\_u32 cnt)

{

alt\_u32 i =0;

while(i < cnt)

{

i++;

}

}

int main()

{

int i;

while(1)

{

for(i=0;i<song\_size;i++)

{

IOWR(MYPWM\_0\_BASE,2,1);

//PWM\_1\_WritePeriod(huanlesong[i][0]);

IOWR(MYPWM\_0\_BASE,0,huanlesong[i][0]);

//PWM\_1\_WriteCompare(huanlesong[i][2]);

IOWR(MYPWM\_0\_BASE,1,huanlesong[i][2]);

delay(huanlesong[i][1]);

IOWR(MYPWM\_0\_BASE,2,0);

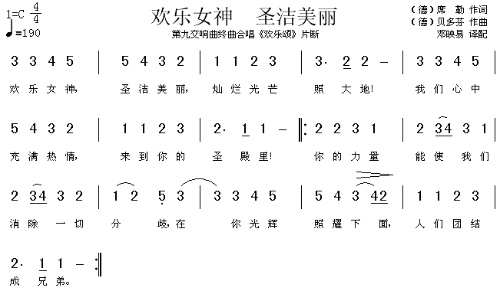
delay(500000);//100ms

}

}

return 0;

}

****