**计算机系统能力培养实践课程系列**

**《操作系统》**

**课程设计指导手册**

**——基于Minisys平台**

**（草案）**

**版本 1.0**

**杨全胜 谢家骏 编写**

**j0234657**

**东南大学计算机科学与工程学院、软件学院**

**东南大学计算机系统能力培养课题组**

2020年8月

# 目 录

[2020年8月 I](#_Toc49364424)

[目 录 I](#_Toc49364425)

[前言 1](#_Toc49364426)

[第一章 开发之前要掌握的知识 2](#_Toc49364427)

[1.1 MIPSfpga概述 2](#_Toc49364428)

[1.1.1 MIPSfpga核心 2](#_Toc49364429)

[1.1.2 MIPSfpga系统 3](#_Toc49364430)

[1.1.3 MIPSfpga寄存器概述 4](#_Toc49364431)

[1.2 Minisys实验平台简介 8](#_Toc49364432)

[1.2.1 Minisys计算资源 9](#_Toc49364433)

[1.2.2 Minisys存储资源 10](#_Toc49364434)

[1.2.3 Minisys I/O外设资源 11](#_Toc49364435)

[1.2.4 Bus Blaster 11](#_Toc49364436)

[1.3 MIPS32体系结构相关知识 11](#_Toc49364437)

[1.3.1 4G地址空间的划分 11](#_Toc49364438)

[1.3.2 TLB 13](#_Toc49364439)

[1.3.3中断以及异常 15](#_Toc49364440)

[第二章 MIPSfpga-SoC的移植 16](#_Toc49364441)

[2.1工具链的搭建 16](#_Toc49364442)

[2.1.1 OpenOCD的安装 16](#_Toc49364443)

[2.1.2 MIPS工具链的安装 16](#_Toc49364444)

[2.1.3 PuTTY以及驱动程序的安装 17](#_Toc49364445)

[2.2 MIPSfpga-SoC工程的创建 17](#_Toc49364446)

[2.3 MIPSfpga-SoC的搭建 19](#_Toc49364447)

[2.3.1MIPSfpga Core的导入 19](#_Toc49364448)

[2.3.2总线协议的转换 23](#_Toc49364449)

[2.3.3添加AXI总线GPIO设备 25](#_Toc49364450)

[2.3.4存储系统的构建 27](#_Toc49364451)

[2.3.5其他外设的添加 41](#_Toc49364452)

[2.3.6总线地址的分配 46](#_Toc49364453)

[2.4硬件工程的编译与验证 47](#_Toc49364454)

[第三章 Bootloader的设计与实现 50](#_Toc49364455)

[3.1工具链的使用 50](#_Toc49364456)

[3.1.1 MIPS交叉编译器的使用 50](#_Toc49364457)

[3.1.2 GDB调试 50](#_Toc49364458)

[3.2驱动程序的设计与实现 52](#_Toc49364459)

[3.3 Bootloader的实现 55](#_Toc49364460)

[3.3.1初始化TLB 55](#_Toc49364461)

[3.3.2 Load elf文件 56](#_Toc49364462)

# 前言

操作系统，作为计算机系统的基本组成部分，自产生之初就对计算机、通信技术领域产生了重大的影响。从最初的手工系统，到后来的多道程序以及分时系统，再到目前主流的PC端Windows、macOS X、Ubuntu等操作系统，以及移动端iOS、Android操作系统。可以说操作系统的出现使得计算机的发展进入了新的时代，也使得个人计算机进入了千家万户。而操作系统的日新月异，离不开计算机科学的教育。本教程正是为了计算机科学教学提出了一种教学用微内核设计方案，该微内核直接运行在真实的硬件平台上，而不是在虚拟机上运行，学生可以更直观的看到整个系统的运行过程。相比于在虚拟机环境运行的操作系统的学习，对学生的计算机系统综合设计能力的提高更为明显。本教程的方案可供高校的本科阶段操作系统课程以及相关课程设计参考。

# 第一章 开发之前要掌握的知识

## 1.1 MIPSfpga概述

MIPSfpga是一款基于MIPS32 m14k处理器实现的商用软核处理器。该软核由Imagination Technologies公司开发，并且向学术界免费提供许可。这使得它可以被用于教学的目的。MIPSfpga目前已经推出2.0版本，MIPSfpga 2.0提供了三个套件：MIPSfpga GSG（Getting Started Guide 入门指引手册）、MIPSfpga Labs以及MIPSfpga SoC（System-on-Chip 片上系统）。

MIPSfpga GSG套件提供了MIPSfpga的Verilog以及VHDL两个版本的非模糊寄存器传输级（RTL）源代码，给出了MIPSfpga的详细文档，并且介绍了MIPSfpga系统使用、开发方法。MIPSfpga GSG还提供了开发MIPSfpga所需要的一系列工具链，包括交叉编译器以及OpenOCD等。并且MIPSfpga还支持指令集扩展，支持用户根据自己的需求增加MIPSfpga的指令集，来运行自定义的指令。

MIPSfpga Labs套件提供了基于MIPSfpga的一系列实验。这些实验解释了如何在FPGA上创建microAptiv核心，以及如何在其上运行和调试程序，如何扩展MIPSfpga以添加新的外设并通过中断或直接内存访问（DMA）与它们通信，如何使用CorExtend特性和修改core的硬件本身，以及如何使用性能计数器来试验不同的内存层次结构、缓存大小、关联性和内容管理策略。

MIPSfpga SoC套件着重介绍了如何在MIPSfpga上搭建部署SOC，并且移植开源操作系统Linux内核。

### 1.1.1 MIPSfpga核心

MIPSfpga中使用的MIPS软核是通用微芯片PIC32MZ微控制器中使用的microAptiv UP core的一个版本。该软核由一组Verilog HDL文件组成，这些文件在5级流水线中实现MIPS32r3指令集体系结构（ISA）。如图1-1所示，正式发行版的MIPSfpga核心包括具有TLB的MMU、指令和数据cache，以及多个接口（例如EJTAG）。总线接口单元支持高级微控制器总线体系结构（AMBA）、高级高性能总线（AHB）精简协议。

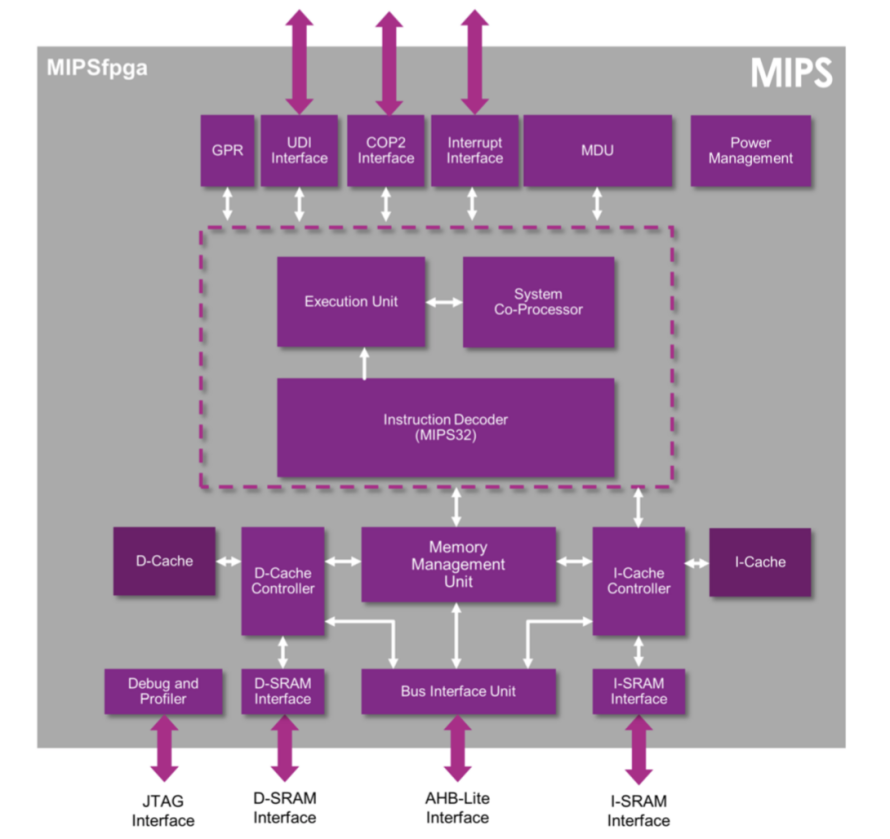


图1-1 MIPSfpga核心结构框图[1]

### 1.1.2 MIPSfpga系统

如图1-2所示，MIPSfpga系统包括MIPSfpga核心以及基于AHB-Lite总线接口的外设。

MIPSfpga系统核心的运行至少需要两个信号，分别是系统输入时钟信号：SI\_ClkIn，系统输入复位信号（低电平触发）：SI\_Reset\_N。该核心还提供了EJTAG接口以便于开发者进行程序下载、调试，可以大大缩短程序开发的周期。

此外，MIPSfpga系统提供AHB-Lite总线接口，以便开发者进行外设的扩展。开发者可以据此添加RAM控制器以便于访问DDR和BRAM等片上存储，也可以添加自定义GPIO来访问片上外设，例如LED、拨码开关等。

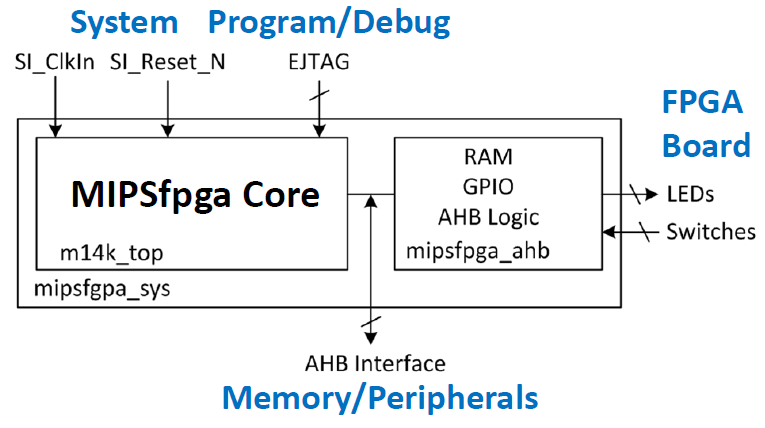


图1-2 MIPSfpga系统

### 1.1.3 MIPSfpga寄存器概述

操作系统，是最接近硬件层的软件。因此操作系统的许多操作依赖于对寄存器的操作，这也是本教程的操作系统采用C语言+汇编语言开发的原因。因此本教程选择性地介绍MIPSfpga中，对于操作系统内核开发十分重要的若干个寄存器。详细的MIPSfpga寄存器介绍请参考官方文档《MicroAptiv UP Software User's Manual MD00942.pdf》。

（1）通用寄存器

MIPS32体系结构共有32个通用寄存器，这些寄存器的长度都是32bit。各个寄存器的编号、助记符以及用途，如表1.1所述。

表1.1 MIPS32通用寄存器

|  |  |  |  |
| --- | --- | --- | --- |
| 寄存器编号 | | 助记符 | 描述 |
| 0  1  2-3  4-7  8-15  24,25  16-23  26,27  28  29  30  31 | zero  at  v0, v1  a0-a3  t0-t7  t8-t9  s0-s7  k0, k1  gp  sp  s8/fp  ra | | 常量0  汇编暂存寄存器，为汇编保留  子程序返回值寄存器  子程序的前几个参数  暂存，子程序使用时不需要恢复  暂存，子程序使用时不需要恢复  子程序必须存储且恢复这些寄存器的值  为异常处理程序保留  全局指针  堆栈指针  子程序用它作为帧指针  子程序返回地址 |

值得注意的是k0，k1两个寄存器的使用。这两个寄存器为操作系统陷入保留，在异常处理程序中会使用到这两个寄存器，理论上不用做其他汇编程序。

sp寄存器被用作堆栈指针。MIPS32体系结构没有相应的堆栈指令，入栈出栈操作都需要显示的对sp寄存器进行修改。通常只在子程序开始和结束时修改sp寄存器的值。MIPS32采取向下生长的栈，入栈时减，出栈时加。

ra寄存器保存子程序的返回地址，在子程序中需要先将ra寄存器入栈，在子程序结束时恢复ra寄存器的值。

（2） CP0寄存器

CP0（Co-Processor 0，协处理器0）是MIPS32体系结构中非常重要的一个部件，CP0是MIPS所有协处理器中唯一一个必须实现的。它起到控制CPU的作用，CP0对CPU的这个控制作用体现在CP0的寄存器上面。本教程给出几个关键CP0寄存器的概述。

1. BadVAddr

BadVAddr寄存器各比特的含义如图1-3所示。该寄存器自动保存最近一次地址相关异常的虚拟地址。在发生TLB相关异常时，需要查寻该寄存器，来获取导致TLB异常的虚拟地址，来进行TLB重填操作。

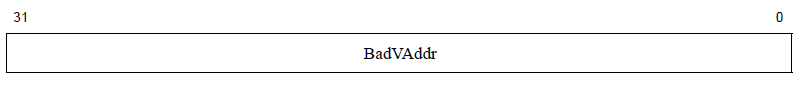


图1-3 BadVAddr寄存器

1. EntryLo0、EntryLo1

EntryLo0、EntryLo1两个寄存器也是与TLB有关的寄存器，EntryLo0存放偶数页的入口物理地址，EntryLo1存放奇数页的入口物理地址。这两个寄存器各比特的含义如图1-4所示。

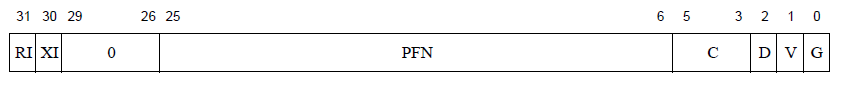


图1-4 EntryLo0和EntryLo1寄存器

RI，XI：读禁止和执行禁止，如果该位被设置位1，那么即使V = 1，对于该页的读取也会导致TLB Invalid异常。

29:26 ：强制写入0

PFN：Page Frame Number，即物理页帧号。对应物理地址的31:12位。

C：描述页面的缓存一致性属性，对于microAptiv UP核来说，2为不可缓存，3为可缓存。

D：脏位，或者叫做写使能位。D = 0时，不允许写入该页，否则会引起TLB修改异常。

V：有效位，表明当前TLB映射是否有效。V = 1，表示允许访问该页，V = 0，对该页面的访问会导致TLB无效异常。

G：全局位，如果G = 1，那么TLB表项匹配时会忽略ASID域。

1. EntryHi

EntryHi是与EntryLo0、EntryLo1功能类似的寄存器，也是用于对TLB读、写操作的寄存器。每当TLB发生异常时，需要将发生异常的虚拟地址的31:13位写入VPN2，该寄存器的各比特含义如图1-5所示。

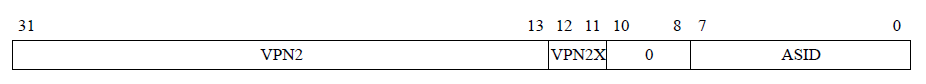


图1-5 EntryHi寄存器

VPN2：发生异常的虚拟地址的31:13bit

VPN2X：本教程中为常量0

ASID：表示地址空间标志，在对TLB进行读操作时，由硬件写入；写操作时，由操作系统软件写入。用于地址空间的隔离。

1. Status

Status是一个用于获取CPU状态的寄存器，其存储的内容可以表示CPU的中断使能、操作状态等信息。该寄存器的各比特含义如图1-6所示。

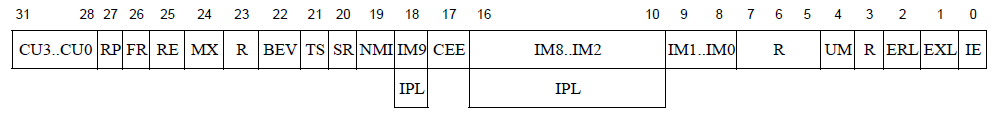


图1-6 Status寄存器

下面给出本教程用到的若干bit的定义：

BEV：是否启动Boot异常向量，用于定位异常向量的入口地址，为1时将异常向量入口定位在kseg1，为0时定位在通常的异常向量入口地址。

TS：TLB是否被关闭，当发生TLB严重错误时，该位被置为1。除非硬件复位，否则无法清除。

NMI：表示是否是不可屏蔽中断，软件只能对其写入0。

IM：是否进行中断屏蔽，0为中断屏蔽，1为不屏蔽。该段与Cause寄存器的IP段配合使用，来进行中断服务。

UM：是否为用户模式。0表示内核模式，1表示用户模式。

ERL：是否处于错误级，该位如果被置1，那么表示处理器处于错误级。eret指令使用ErrorEPC寄存器来返回而不是EPC寄存器。并且kuseg段会被当成一个不经过TLB映射，且不缓存的段。

EXL：是否处于异常级。

IE：是否中断使能，当IE = 1，EXL = 0，ERL = 0时，中断才会被启动。

1. Cause

Cause寄存器记录上一次发生异常的原因，该寄存器被用于定位错误或者异常的原因，其各比特含义如图1-7所示。

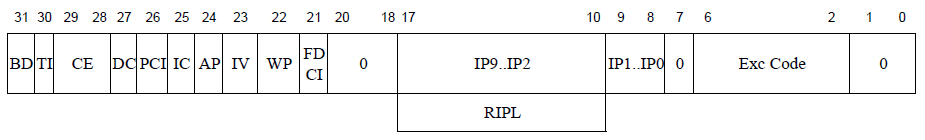


图1-7Cause寄存器

BD：表示最近发生的异常是否位于分支延迟槽，该位如果被置1，那么分支指令将无法进行跳转。在实验中发现，如果在异常处理中使用两个连续的分支或者跳转指令，该位会被置1，进而导致产生保留指令异常（10号异常）

IP：表示当前相应中断是否被挂起，其中IP1:0为软件中断，其他为硬件中断。

Exc Code：异常码（Exception Code），该字段的编码如表1.2所示

表1.2 异常编码

|  |  |  |  |
| --- | --- | --- | --- |
| Exc Code | | 简记 | 描述 |
| 0  1  2  3 | Int  Mod  TLBL  TLBS | | 中断  TLB修改异常  TLB加载或取指令异常  TLB存储异常 |

续表1.2 异常编码

|  |  |  |  |
| --- | --- | --- | --- |
| Exc Code | | 简记 | 描述 |
| 4  5  6  7  8  9  10  11  12  13  14-22  23  24  25-31 | AdEL  AdES  IBE  DBE  Sys  Bp  RI  CpU  Ov  Tr  —  WATCH  MCheck  — | | 加载或取指地址错误异常  存储地址错误异常  总线错误（取指）  总线错误（访问数据）  系统调用异常  断点异常  保留的指令异常  协处理器不可用异常  整数溢出异常  陷阱异常  保留  访问WatchHi/WatchLo地址  机器检查  保留 |

本教程中，操作系统设计需要重点关注的异常有0号异常（中断），1-3号异常（TLB异常），以及8号异常（系统调用异常）。在本教程的异常处理中，保留了其他异常的处理函数，仅实现了上述的异常类型的处理函数。

## 1.2 Minisys实验平台简介

Minisys实验开发板是由东南大学计算机科学与工程学院与依元素科技联合开发的，用于计算机系统综合能力培养的FPGA实验开发板。Minisys实验板是以Xilinx Artix-7TM系列FPGA（XC7A100T FGG484C-1）为主芯片的可用于“数字电路”、“计算机组成原理”、“计算机组成课程设计”、“微机原理与接口技术”、“计算机系统综合课程设计（SoC设计）”等多门课程实验的统一实验平台。

Minisys实验板拥有丰富的计算、存储资源以及I/O外设资源。以供操作系统内核的使用。Minisys实验板如图1-8所示，其资源对照表如表1.3所示。

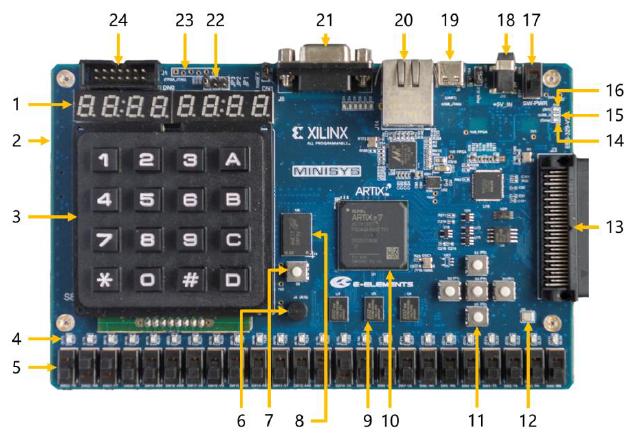


图1-8 Minisys实验板

表1.3 Minisys资源表

|  |  |  |  |
| --- | --- | --- | --- |
| 标注 | 描述 | 标注 | 描述 |
| 1  2  3  4  5  6  7  8  9  10  11  12 | 8个七段数码管  Micro SD卡槽（实验板背面）  4\*4小键盘  LEDs（红、黄、绿各8个）  拨码开关  蜂鸣器  FPGA复位键  DDR3 SDRAM  SRAM  XC7A100T主芯片  五个按键开关  麦克风 | 13  14  15  16  17  18  19  20  21  22  23  24 | 接口板连接器  FPGA烧写完成指示灯  USB\_JTAG指示灯  电源指示灯  电源开关  电源连接口  Type-C接口（编程、串口）  以太网接口  VGA接口  编程跳线  用户扩展IO  JTAG接口 |

### 1.2.1 Minisys计算资源

Minisys的主芯片XC7A100T上有101440个逻辑单元，15850个Slice，每一个Slice中带有4个6输入的查找表（LUT）和8个触发器，片内近12.5%的查找表可以配置为64-bit分布式RAM或者32位的SRL（或两个16位SRL16），使得综合工具能够充分利用这些逻辑和存储资源。

拥有240个DSP48E1数字信号处理单元，每个DSP48E1中包含一个预加器，一个25×18乘法器，一个加法器以及一个累加器。

拥有6个时钟管理模块（CMT），每个包含1个混合模式时钟管理器（MMCM）及一个锁相环（PLL）。MMCM和PLL的中心都有一个可以根据输入电压而调速的晶振，由此能够生成频率范围很宽的时钟信号。同时，这两个部件又都能作为输入时钟信号的抖动滤波器。

XC7A100T的内部时钟最高可达450MHz，Minisys实验板采用100MHz主频。

### 1.2.2 Minisys存储资源

（1）Block RAM

Minisys的Block RAM（或简称BRAM）集成在主芯片XC7A100T的内部，片内集成135个36Kbit的Block RAM，并且每一个可以当作两个独立的18Kbit的Block RAM使用。这些Block RAM资源可以利用Vivado的IP集成器很方便地配置成单端口、双端口等多种类型RAM。

（2）DDR3 SDRAM

Minisys实验板上，将一个容量为256M×16bit的DDR3 SDRAM(芯片型号为MT41J256M16-FBGA96)连接到主芯片上。当主芯片访问SDRAM时，需要传送15位的行地址、3位块地址和10位列地址，其中行地址和列地址分时共用一组地址线。

（3）SRAM

SRAM模块由三块IS61WV51216BLL-10MI芯片并联组成，每块芯片的容量为512K×16bit，并联后的SRAM模块的容量为512K×48bit，通过19根地址线和48根数据线与主芯片连接。

（4）Flash Memory

非易失串行Flash 的容量是128Mbits，使用的是专用的Quad SPI 总线。FPGA 的配置文件可以写入Quad SPI Flash（型号N25Q032A13ESE40F），当短接了JP3后，板子在上电时，FPGA 自动从SPI Flash 中读取配置文件。当编程跳线连接JP3的位置时，可以将编程文件下载到Flash 中。

### 1.2.3 Minisys I/O外设资源

Minisys拥有丰富的I/O外设资源。包括24个拨码开关SW23~0，在实验中可以作为系统的输入。还包括24个LED灯，其中RLD7~0为红色，YLD7~0为黄色，GLD7~0为绿色，可以作为系统的输出。还包括5个方向按键、4\*4按键、一个蜂鸣器以及八个七段数码管。

Minisys丰富的计算以及存储资源使得开发者利用其开发、运行一个自定义的SoC变得非常方便且高效。其丰富的I/O外设资源也使得开发者可以使自己的SoC更有用户交互性。总的来说，选择Minisys作为本教程的操作系统微内核运行的硬件平台是合适的。

### 1.2.4 Bus Blaster

Bus Blaster如图1-9所示。它是一个可以用于FPGA、ARM、flash、CPLD等设备的高速JTAG调试器。在实验中，可以通过Bus Blaster配合OpenOCD进行程序装载、调试、运行等操作。这大大缩短了程序开发的周期。

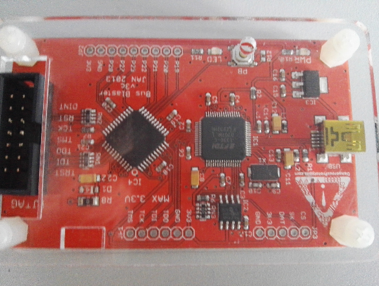


图1-9 Bus Blaster下载调试器

## 1.3 MIPS32体系结构相关知识

在开发操作系统之前，我们需要详细的了解MIPS32体系结构。

### 1.3.1 4G地址空间的划分

在了解MIPS32的地址空间划分之前， 首先需要回顾一下操作系统的概念：用户态和内核态。内核态相对于用户态，拥有更高的权限，可以访问4G全部的虚拟地址；而用户态只能访问其中一半的虚拟地址（0~0x7fff\_ffff）。

MIPS32体系结构的虚拟内存段划分如图1-10所示。

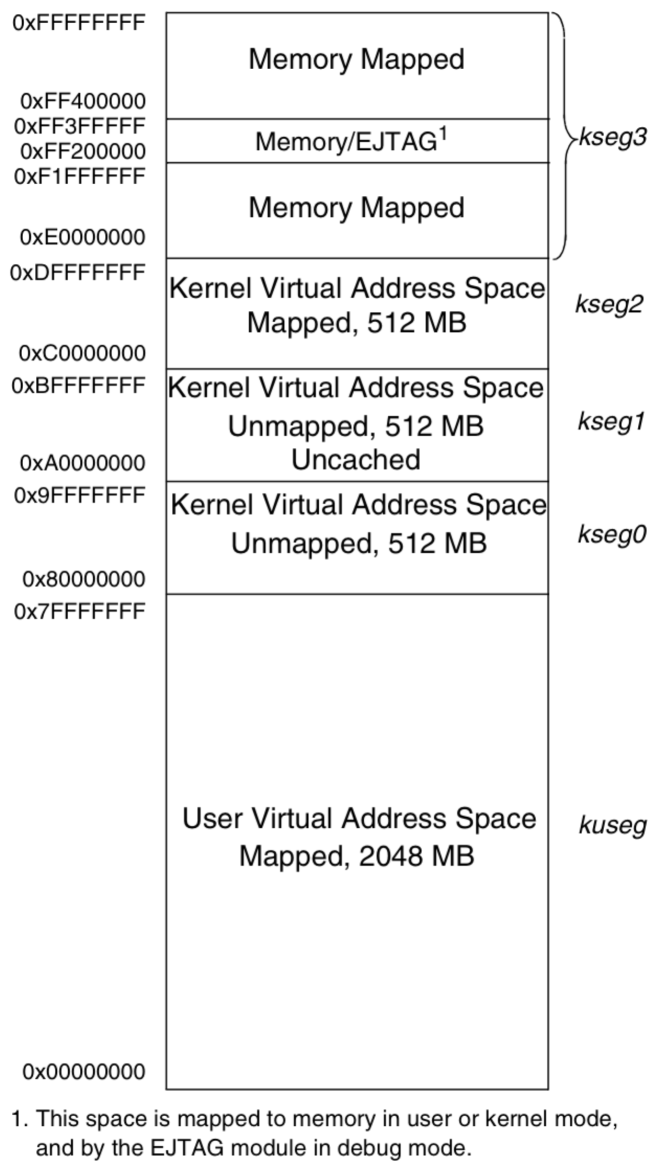


图1-10 MIPS32虚拟内存段划分

* kuseg是用户内核段，该虚拟内存段主要用于用户模式的访问，可以通过TLB映射到任意物理地址。
* kseg0是内核段，该段地址不经过TLB映射，物理地址是虚拟地址直接减去0x8000\_0000，也就是说，虚拟地址0x8000\_0000直接映射物理地址0x0。
* kseg1也是内核段地址，该区域也不经过TLB映射，不经过cache缓存，因此适合把外设（即外设的控制寄存器）映射到这个段地址中去，该段物理地址为虚拟地址减去0xA000\_0000，也就是说0xA000\_0000的虚拟地址映射的物理地址也是0x0。

更详细的地址划分以及MMU映射规则请参考官方文档《MicroAptiv UP Software User's Manual MD00942.pdf》。

### 1.3.2 TLB

TLB（Translation Lookaside Buffer，翻译后备缓冲器，也叫做快表），是MIPS32体系结构中完成映射段的物理地址到虚拟地址的映射的硬件，可以看做是页表的cache。与TLB相关的寄存器有EntryLo0、EntryLo1、EntryHi以及PageMask等寄存器，这些寄存器描述在1.1.3小节中已经给出。MIPS32支持页式存储管理。TLB表项中每一项拥有一个VPN（虚拟页号）和一个PFN（物理页帧号）。当程序访问一个虚拟地址时，会先和TLB中的每一个VPN相比较，同时比较ASID（在EntryHi寄存器中指定），并且对标志位也进行匹配。如果匹配成功，则返回对应的PFN。如果全部匹配失败，那么会出发TLB未命中异常。操作系统内核应查询页表，进行TLB重填。microAptiv UP核拥有16个页表项。TLB的地址转换过程如图1-11所示。

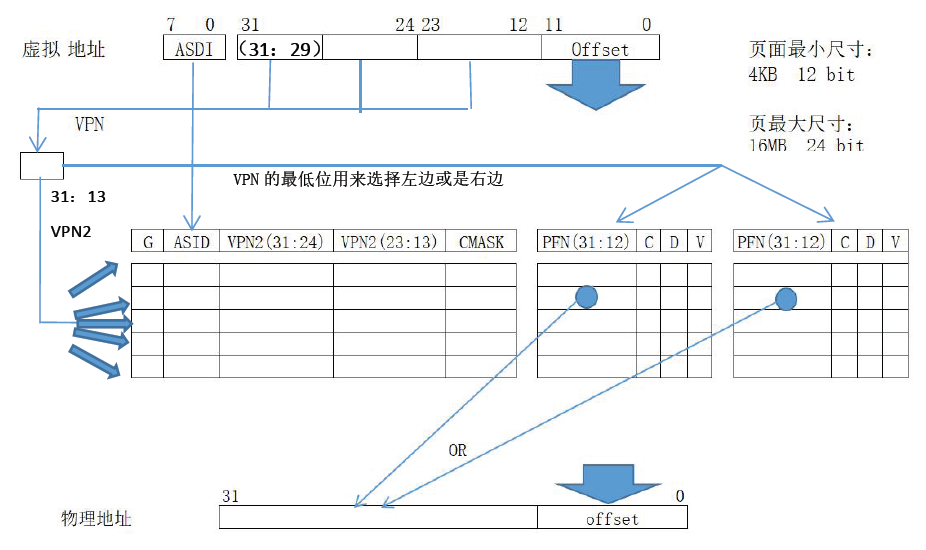


图1-11 TLB地址转换过程

在图1-11中，各个符号的含义如下：

CMASK：页掩码，决定页面大小和用于比较的VPN2的位数。

G：全局位，被置位时忽略ASID。

ASID：地址空间标志，用于地址空间的隔离，在本教程中被定义为进程ID。

VPN：虚拟页号。

PFN：物理帧号。

D：可写位，D = 1时该页可写。

V：有效位，V = 1时该页有效。

C：缓存性质。

（2）TLB异常的处理

TLB异常是操作系统最常见的异常，每次对TLB中缺失的虚拟地址的访问都会导致TLB异常，这也是MIPS提供TLB异常专用入口0x8000\_0000的原因。因此，TLB异常处理是本教程的一个重点。本教程设计的TLB缺失处理流程图如图1-12所示。

首先在MIPS32体系结构中，CPU访问到的虚拟地址导致TLB异常后，CPU会自动进入内核态并且屏蔽中断，然后转到TLB专用异常向量入口0x8000\_0000，因此TLB异常的处理函数应该放在这个入口处。在进入TLB异常处理函数后，内核首先需要查询页目录，找到虚拟地址的页表项，然后根据页表项中的有效位V判断是否发生页缺失。如果没有对应的页表项，或者有效位V = 0，那么就说明没有与虚拟地址对应的物理页存在。在这种情况下，需要进行页插入，当完成了虚拟地址和物理页的映射后，即可根据返回的物理地址进行TLB重填。

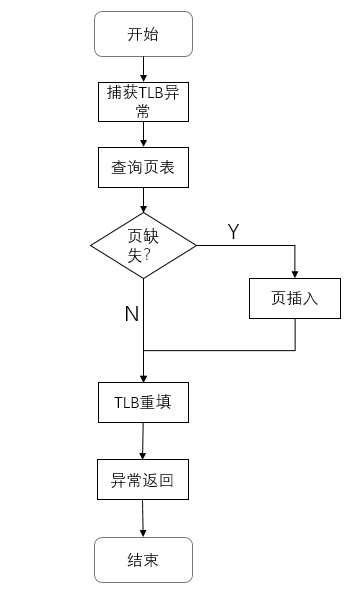


图1-12 TLB异常处理流程

在得到发生错误的虚拟地址BadVAddr所对应的虚拟页号VPN以及物理帧号PFN后，首先要根据VPN来设置EntryHi寄存器。系统需要将虚拟地址的[31:13]位写入EntryHi的VPN2字段，如果TLB异常来自当前正在运行的进程，那么将该进程的ASID也写入EntryHi的[7:0]位。

接着根据页面的奇偶属性，将PFN写入EntryLo0或者EntryLo1的PFN字段，并且设置脏位D = 1，有效位V = 1，表示当前TLB映射项有效且可写。然后设置PageMask为0，表示页大小为4KB。接着执行tlbwr指令，该指令随机选择一个TLB牺牲项，根据上面设置的寄存器，进行TLB写操作，在完成了TLB重填之后，即可进行TLB异常返回。

下面是TLB重填的伪代码设计。

VPN = get\_VPN(BadVAddr) //获取VPN

PFN = get\_PFN(pgdir, va) //页表中查找PFN

set\_EntryHi(VPN, ASID) //设置EntryHi

set\_EntryLo0(PFN, C, V, D) 或者 set\_EntryLo1(PFN, C, V, D) //设置EntryLo

tlbwr //TLB重填

eret //异常返回

### 1.3.3中断以及异常

正如操作系统课程中讲的“操作系统是中断驱动的”。任何操作系统都是离不开中断的。可以说操作系统本身就是一个大型中断和异常处理程序。

MIPS中，中断、陷阱、系统调用和任何可以中断程序正常执行流的情况都称异常。也就是说，中断是异常的一种。各个异常的异常码可以在1.1.3节中的Cause寄存器的介绍中找到，可以看到中断是0号异常。

关于更多microAptiv UP核异常的内容，请读者参考官方文档《MicroAptiv UP Software User's Manual MD00942.pdf》的“Exceptions and Interrupts in the microAptiv™ UP Core”章节。该章节详细的描述了MIPS microAptiv UP核的异常产生条件、异常优先级、中断、异常向量入口、通用异常处理等内容。请读者在开发OS之前，掌握其中的内容，尤其是TLB相关异常的处理、系统调用。

# 第二章 MIPSfpga-SoC的移植

## 2.1工具链的搭建

### 2.1.1 OpenOCD的安装

首先将MIPS\_FPGA压缩包解压至路径D:\

接着通过D:\MIPS\_FPGA\路径找到OpenOCD-0.9.3-Installer.exe，双击打开，安装OpenOCD，当出现如图2-1所示界面时，去掉第二个选项的勾，然后继续安装。

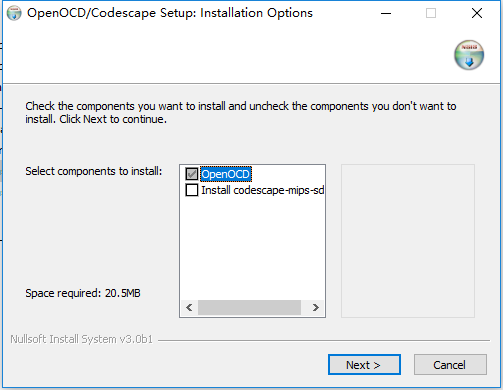


图2-1

### 2.1.2 MIPS工具链的安装

通过D:\MIPS\_FPGA\路径找到mipssdk.v2.0.0k.windows.x64.offline.exe，双击打开，默认安装（安装路径保持默认，并为C盘保证足够的空间）,当遇到如图2-2所示界面时，把空白处都打勾，点击next继续安装。

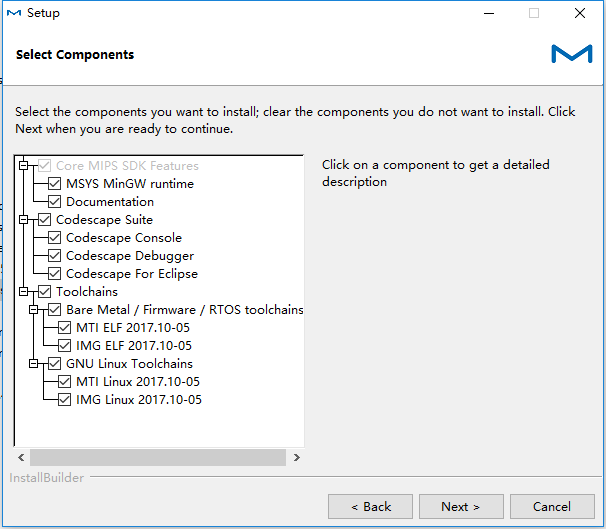


图2-2

等待出现图**2-3**所示界面时，点击**next**。安装完成。

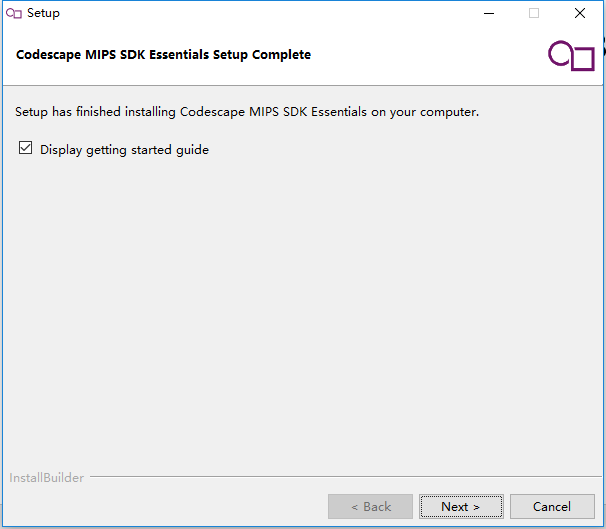


图2-3

### 2.1.3 PuTTY以及驱动程序的安装

通过D:\MIPS\_FPGA找到putty-64bit-0.70-installer，双击打开，一直点击next,直到安装完成。如图2-4所示。点击Finish。PuTTY安装完成。

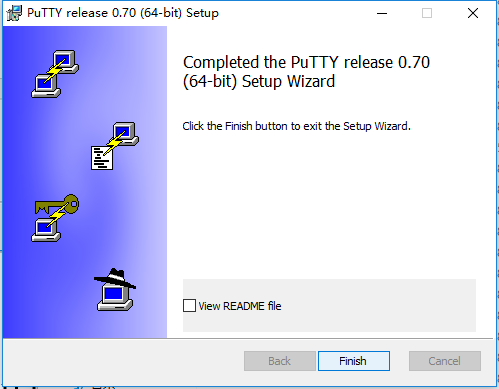


图2-4

最后通过C:\Program Files\Imagination Technologies\OpenOCD找到zadig\_2.1.1.exe，双击打开，把缺少的驱动程序选择安装。

## 2.2 MIPSfpga-SoC工程的创建

首先打开vivado2017.4，点击create project，然后填写工程名称并选择路径。如图2-5所示。

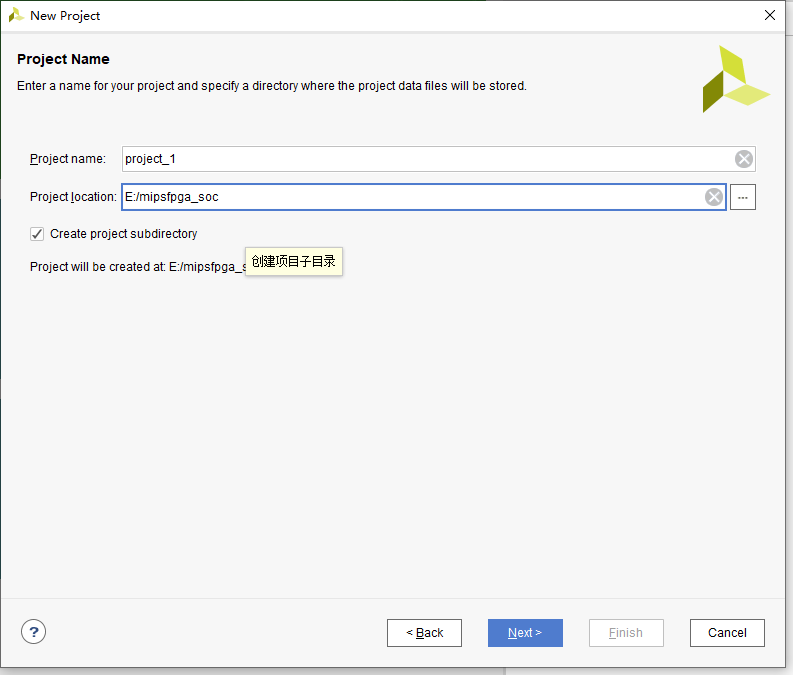


图2-5

点击Next，选择RTL project。

点击Next，这里暂时不导入任何源文件。

点击Next，这里暂时不添加约束文件。

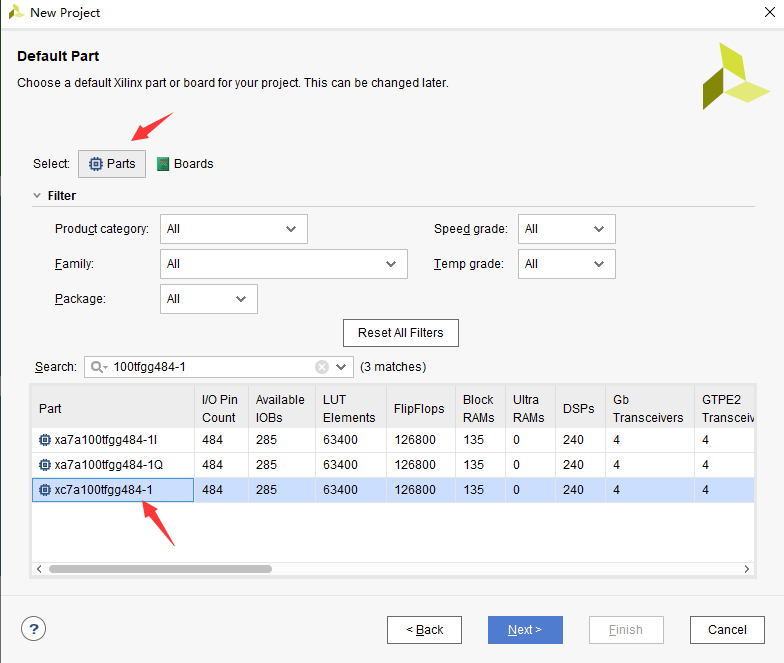


图2-6

点击Next，这一步选择FPGA芯片类型为**xc7a100tfgg484-1.**

点击Next，然后点击Finish完成工程的创建。在创建完成后，在左侧PROJECT MANAGER栏中点击Create Block Design，如图2-7所示，然后点击OK完成Block Design的创建。

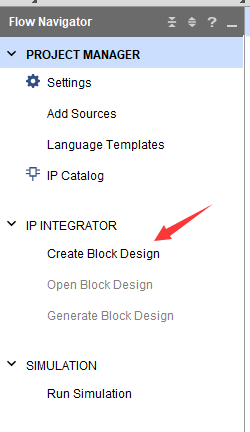


图2-7

## 2.3 MIPSfpga-SoC的搭建

### 2.3.1MIPSfpga Core的导入

首先点击PROJECT MANAGER的Settings进入设置界面。如图2-8所示，展开IP的选项，选择Repository，然后点击加号“+”，添加准备好的MIPSfpga Core IP核（mips\_packaged，该IP核在MIPS\_FPGA目录下可以找到）的路径。点击确定完成添加。如图2-9所示，本教程将该IP的目录复制到工程文件的子目录下，便于移植。

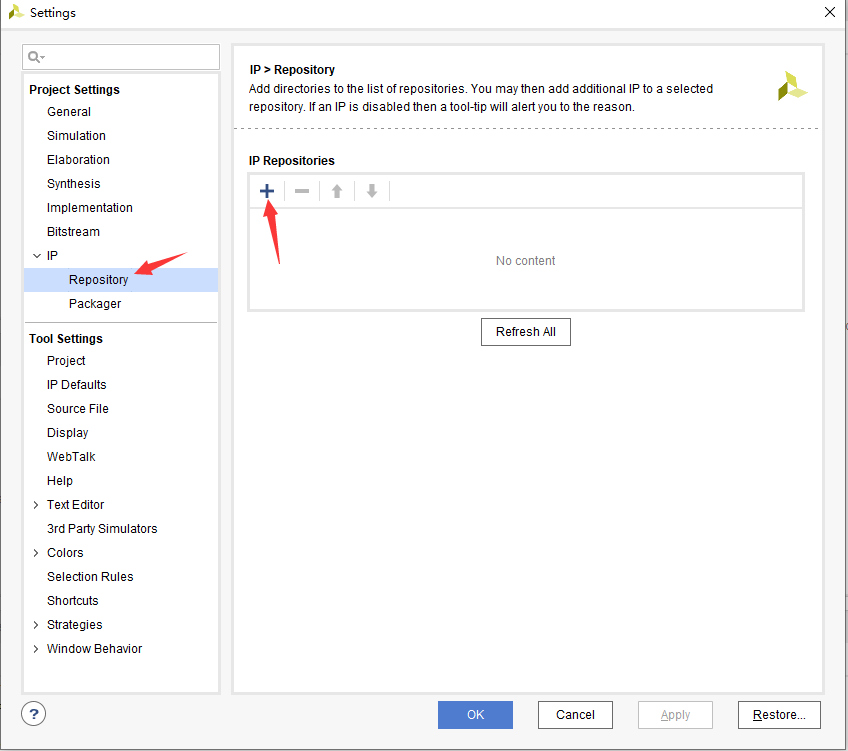


图2-8

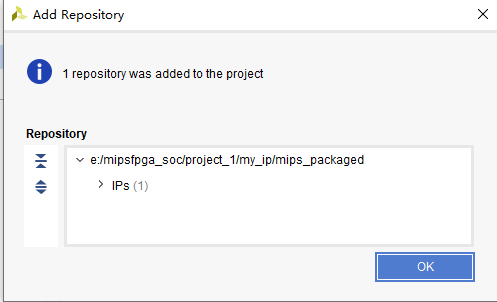


图2-9

接着，回到我们创建的Block Design中，此时它还是空白的。我们选择Add IP，搜素MIPS即可找到刚刚添加的MIPSfpga Core的IP核。点击完成添加，如图2-10所示。

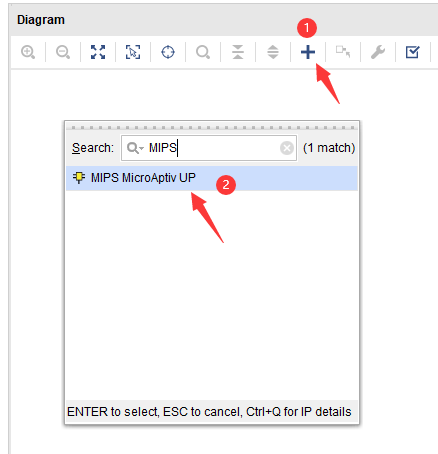


图2-10

这是我们可以在Block Design中看到我们刚才添加的MIPSfpga Core，也就是CPU部分。如图2-11所示。可以看到它上面有许多信号，遵循左进右出原则，这些信号/接口的功能描述如表2.1所示。

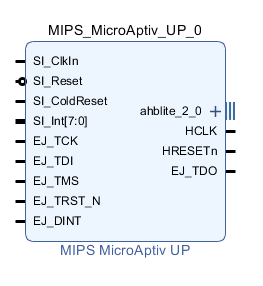


图2-11

表2.1 CPU信号/接口描述

|  |  |  |
| --- | --- | --- |
| 信号/接口名 | 功能描述 | |
| SI\_ClkIn  SI\_Reset  SI\_ColdReset  SI\_Int  EJ\_\*  ahblite\_2\_0  HCLK  HRESETn | | CPU时钟输入  CPU复位  CPU冷复位  8bit中断输入  EJTAG调试相关信号  AHB-Lite总线接口  AHB-Lite总线同步时钟  AHB-Lite总线复位 | |

下面我们将对CPU上的信号进行连接。首先按照如图2-12的方式创建输入输出port进行连接。左侧三个IP核分别为Utility Vector Logic（非门），Utility Buffer（BUFG，CSIZE:1），Constant（位宽8bit，低电平）。这里完成了EJTAG的连接以及CPU复位信号的连接。

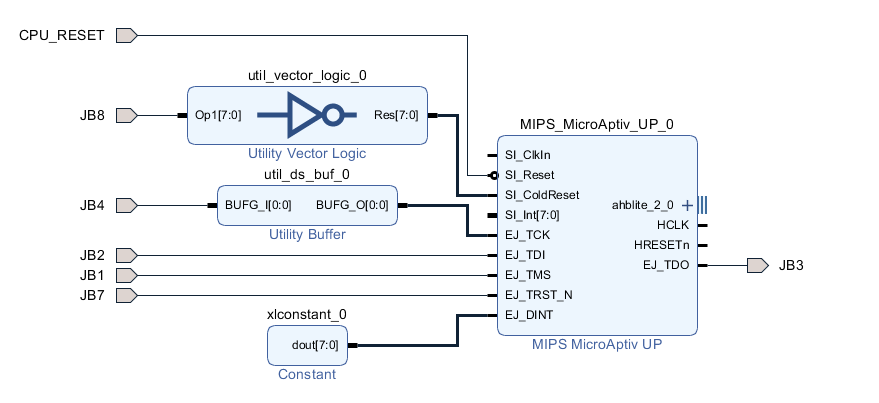


图2-12

在链接CPU时钟前，首先已知Minisys开发板的晶振时钟频率为100MHz。CPU不需要这么高的时钟频率，因此需要创建时钟向导，来创建CPU的输入时钟。

Add IP，找到Clock Wizard，在primitive中选择PLL，如图2-13所示。接着进入Output Option中，进行如图2-14所示的设置。两个时钟输出端口，50MHz的时钟直接连接置CPU的时钟输入，200MHz的后面的教程中会用到。

至此CPU的输入信号，除了中断信号外全部连接完成，下面进行AHB-Lite总线接口的连接。

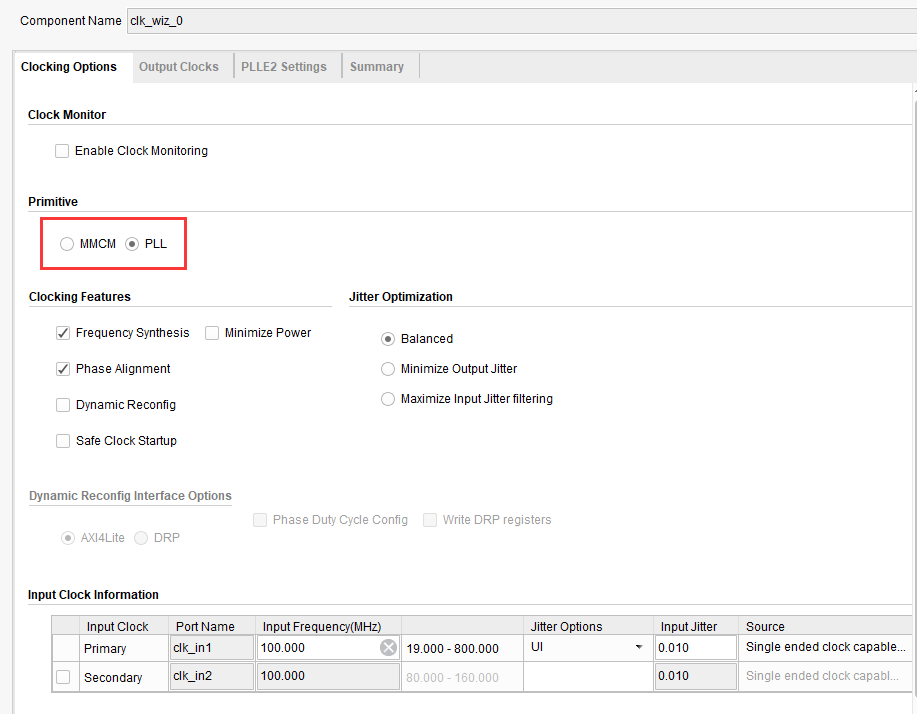


图2-13

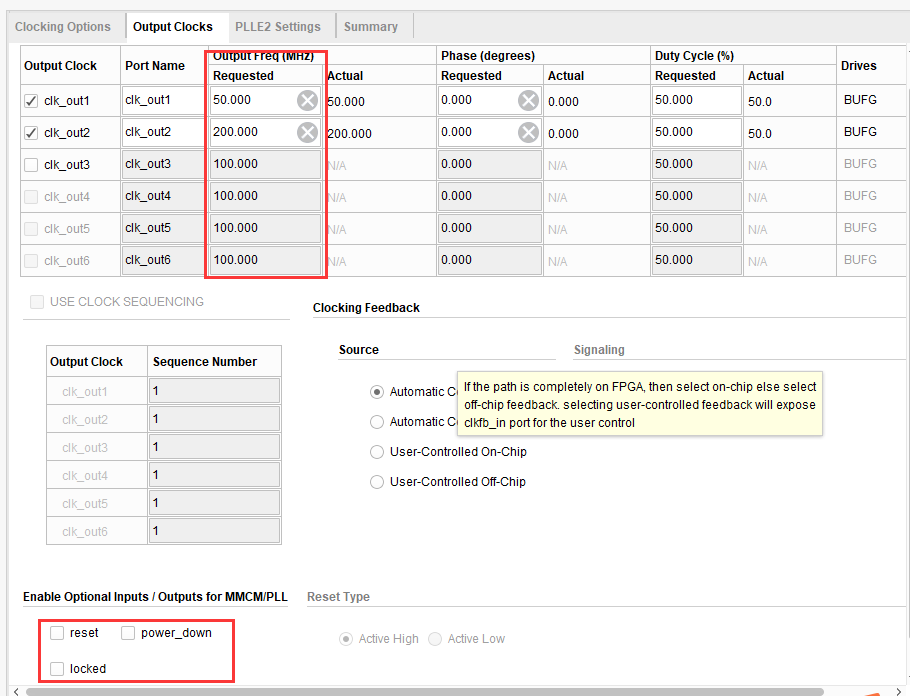


图2-14

### 2.3.2总线协议的转换

由于Xilinx提供的大部分的总线设备控制器使用的总线标准是AXI或AXI-Lite，为了方便硬件设计，这里需要将microAptiv UP核的AHB-Lite总线进行转换，转换成常用的AXI总线标准。Vivado提供了相应的IP核来完成这个转换，找到并添加AHB-Lite to AXI Bridge这个IP核。

将CPU上的AHB-Lite接口展开，找到名为HREADY的信号。再将AHB-Lite to AXI Bridge的AHB-Lite接口展开，找到s\_ahb\_hready\_in和s\_ahb\_hready\_out两个信号，将它们连接在一起，再将两个AHB-Lite接口相连，并且连接时钟和复位信号。这样就完成了总线的桥接，连接好的情况如图2-15所示。

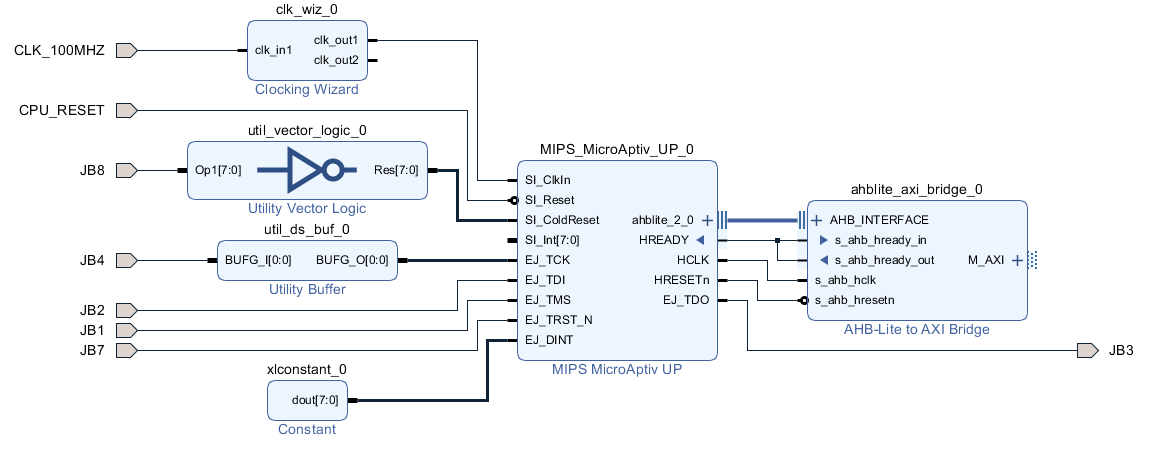


图2-15 CPU与总线桥的连接

此时我们已经将MIPS MicroAptiv核上的AHB总线协议转换成了我们常用的AXI总线协议。下一步是添加AXI总线互联IP核，来实现AXI设备的扩展。

Add IP，找到AXI Interconnect这个IP核并添加，双击它进行自定义，在这里可以自定义Master Interface的数量，可以根据实际需求进行增减。本例程中设置了11个Master Interface（也就是11个AXI Slave设备）。

连接好的情况如图2-16所示，各个AXI从设备的Clk和Reset信号在添加设备的时候进行连接。

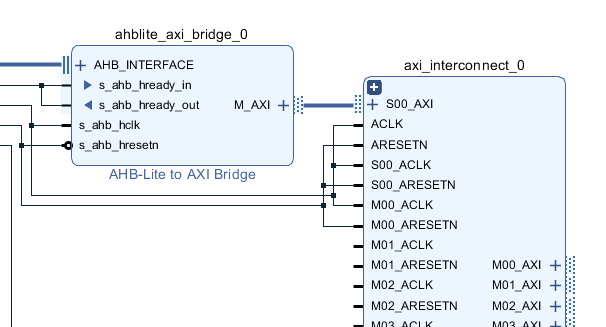


图2-16

### 2.3.3添加AXI总线GPIO设备

LED、蜂鸣器、七段数码管、拨码开关，这些设备的控制都是基于GPIO进行的。以LED为例，添加名为AXI GPIO的IP核，双击进行自定义。按照图2-17的方式，选择All Output模式，并选择位宽为24bit（Minisys实验板共有24个LED）。将该IP核的AXI接口与AXI Interconnect的M00相连，并连接总线时钟和复位信号（这些设备的时钟和复位信号全部连接在一起，即全部连接到CPU的HClk、HRESETn），并且创建LED输出端口，位宽24。连接好的情况如图2-18所示。

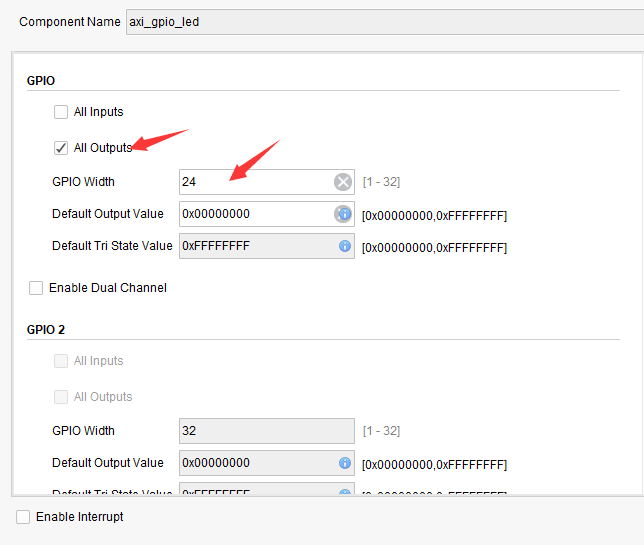


图2-17

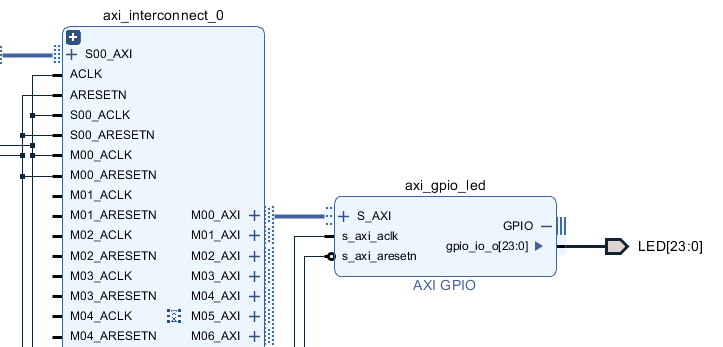


图2-18

类似的，添加蜂鸣器、拨码开关、七段数码管的GPIO。值得注意的是，拨码开关应选择All Input模式。对于蜂鸣器，除了添加GPIO作为控制寄存器外，还应添加控制蜂鸣器的输出波形的控制器，读者可以选择使用本教程提供的IP核（路径MIPS\_FPGA/my\_ip/buzzer），也可以自行设计。七段数码管的GPIO可以分为使能和数据两个，并且也需要添加相应的控制器来点亮数码管，这个数码管实质上就是一个译码器，读者可以根据数字电路的知识自行设计，也可以使用本教程提供的IP核来进行控制（路径MIPS\_FPGA/my\_ip/seven\_seg）。全部添加好后，此时的Block Design如图2-19所示。

添加完外设之后，下一步是创建MIPSfpga-SoC的存储系统。

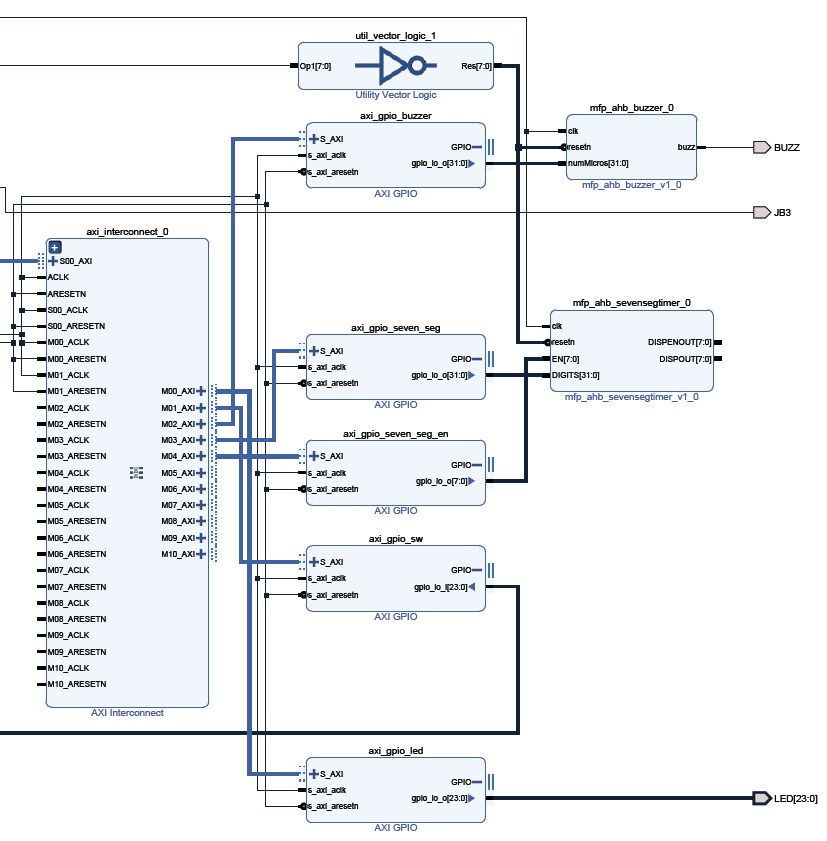


图2-19

### 2.3.4存储系统的构建

MIPSfpga-SoC的存储系统主要由两个部分构成，一部分是Block RAM，该部分容量小速度快，不足以支撑OS运行；另外一部分是由DDR3 SDRAM构成，这一部分容量大，但速度相对较慢，可以支撑OS运行。根据这个特点，我们应当选择DDR3作为系统的主存储器，而将Block RAM映射为存放Bootloader的存储器。

（1）添加Block RAM控制器

Add IP，找到AXI BRAM Controller这个IP核，点击添加。然后双击自定义这个IP，按照图2-20的选择修改端口数为1。点击确定。

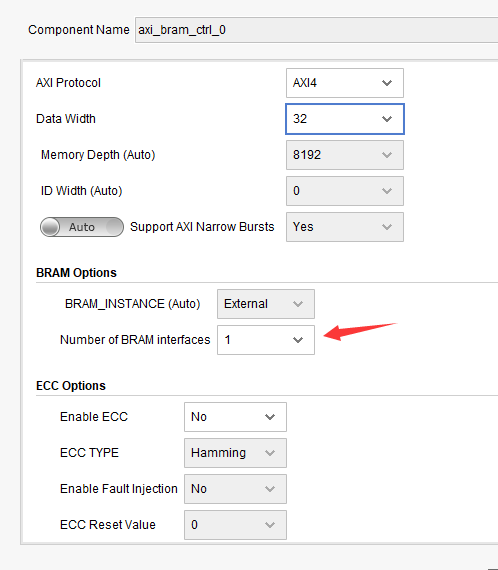


图2-20

接着添加第二个IP，找到Block Memory Generator这个IP，双击并添加。然后自定义这个IP，在basic选择Mode为Stand Alone，其他保持默认，如图2-21所示。

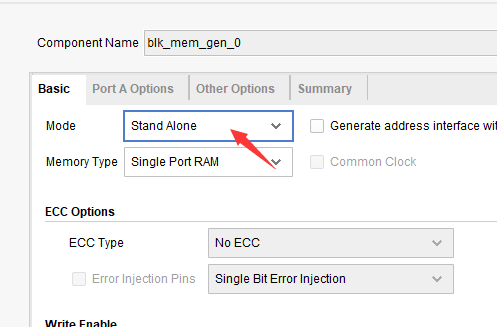


图2-21

接着进入Port A Option，设置数据深度为131072（128k），剩余选项的设置如图2-22所示。在Other Option中，可以设置初始化.coe文件，我们可以将编译好的代码生成.coe文件，将其预读取到BRAM中去。读者可以使用教程中提供好的.coe文件（MIPS\_FPGA/ram\_init/ram\_init.coe，使用此文件时地址分配要和本教程保持一致），也可以自行编译生成.coe文件。

然后连接它们和AXI Interconnect，如图2-23所示。

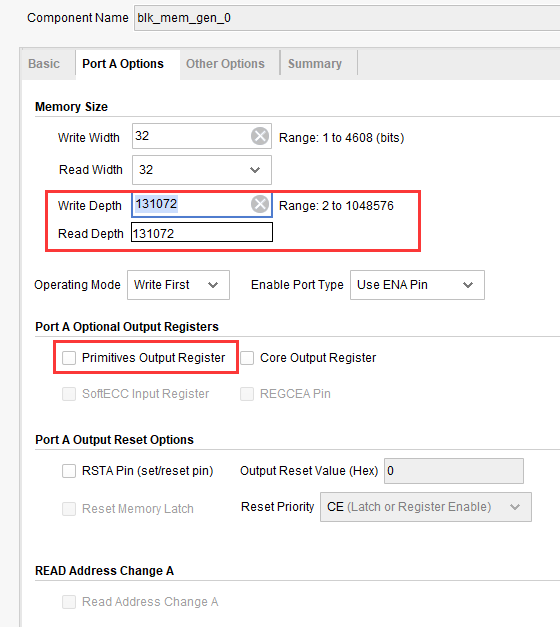


图2-22

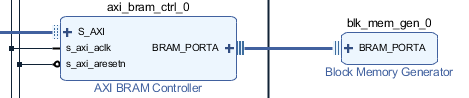


图2-23

在本教程中，共添加了两套上述的BRAM控制器和Block Memory Generator，一个用于Bootloader，映射到物理地址0x1fc0\_0000（请读者思考这是为什么）；另一个是Linux启动所需要的一个8kB内存空间，映射到物理地址0x1000\_0000，如果不需要移植Linux，也可以不添加这组IP。

（2）添加DDR3 SDRAM控制器

DDR3 SDRAM控制器采用Xilinx的Memory Interface Generator（MIG）这个IP核。首先找到这个IP核并添加，接着按照如下步骤自定义该IP。

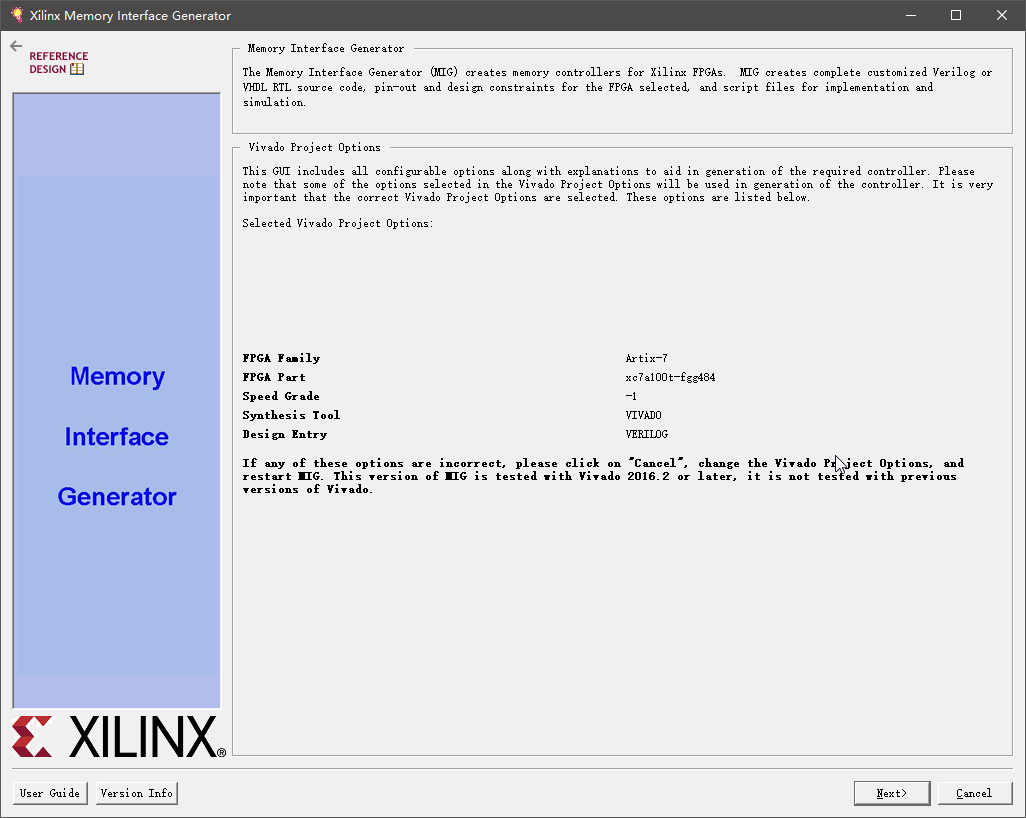


图2-24

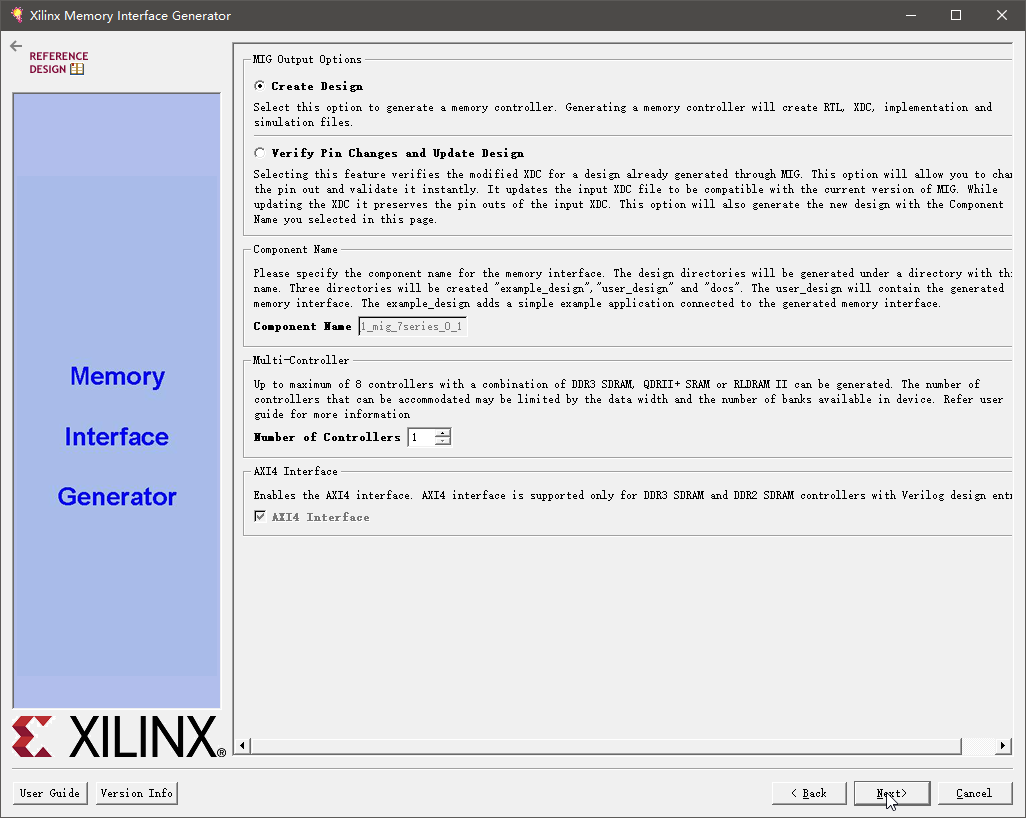


图2-25

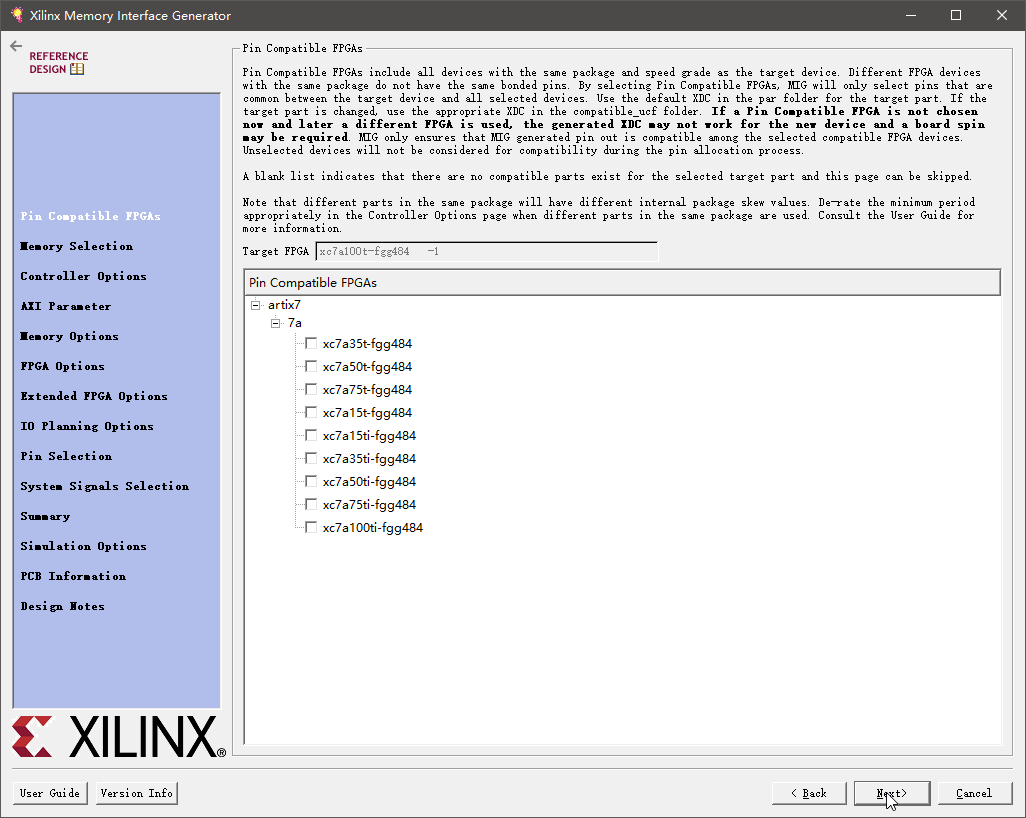


图2-26

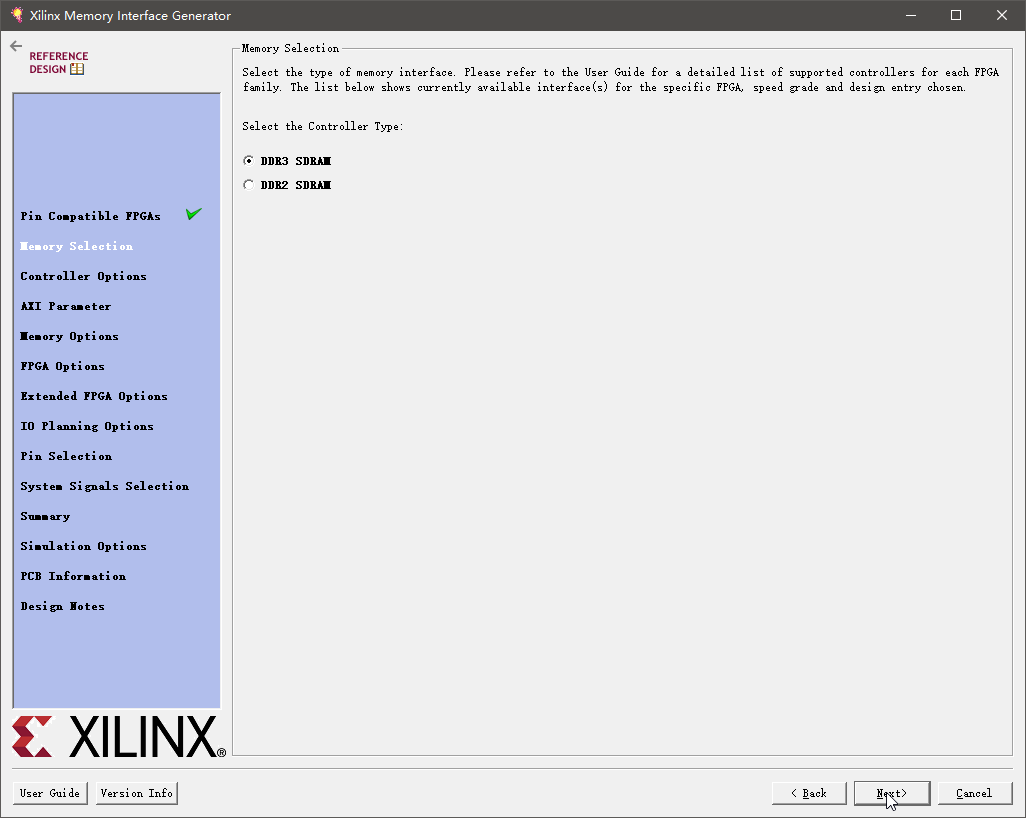


图2-27

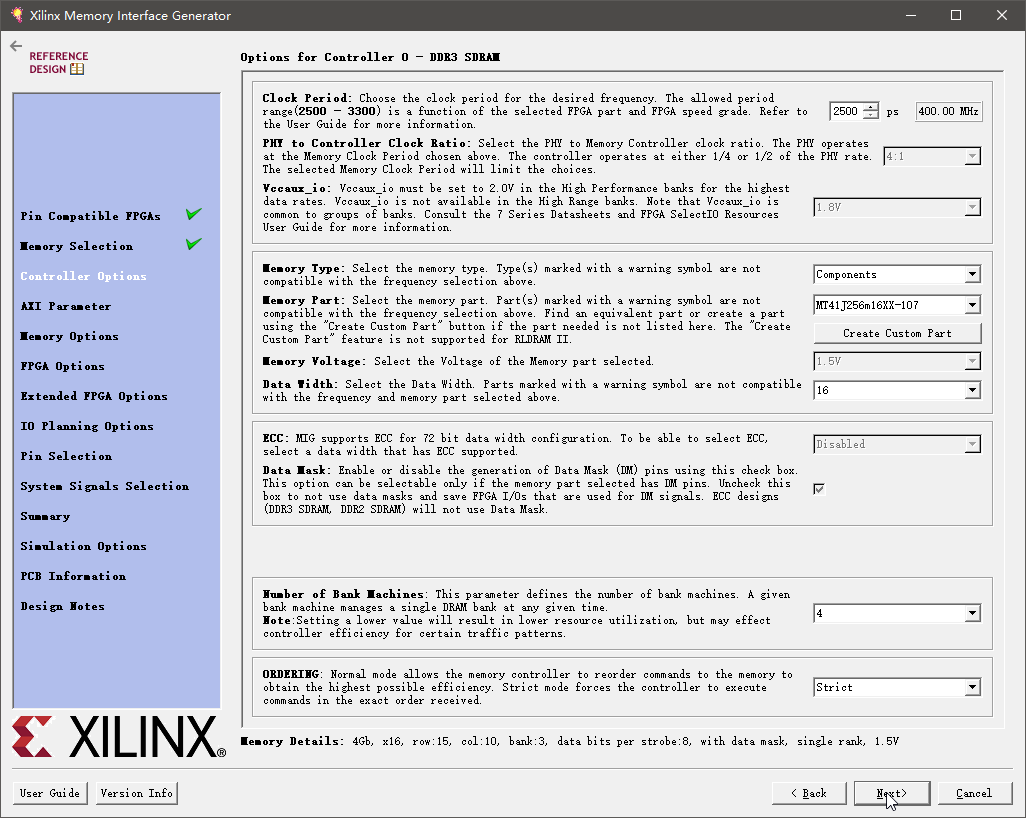


图2-28

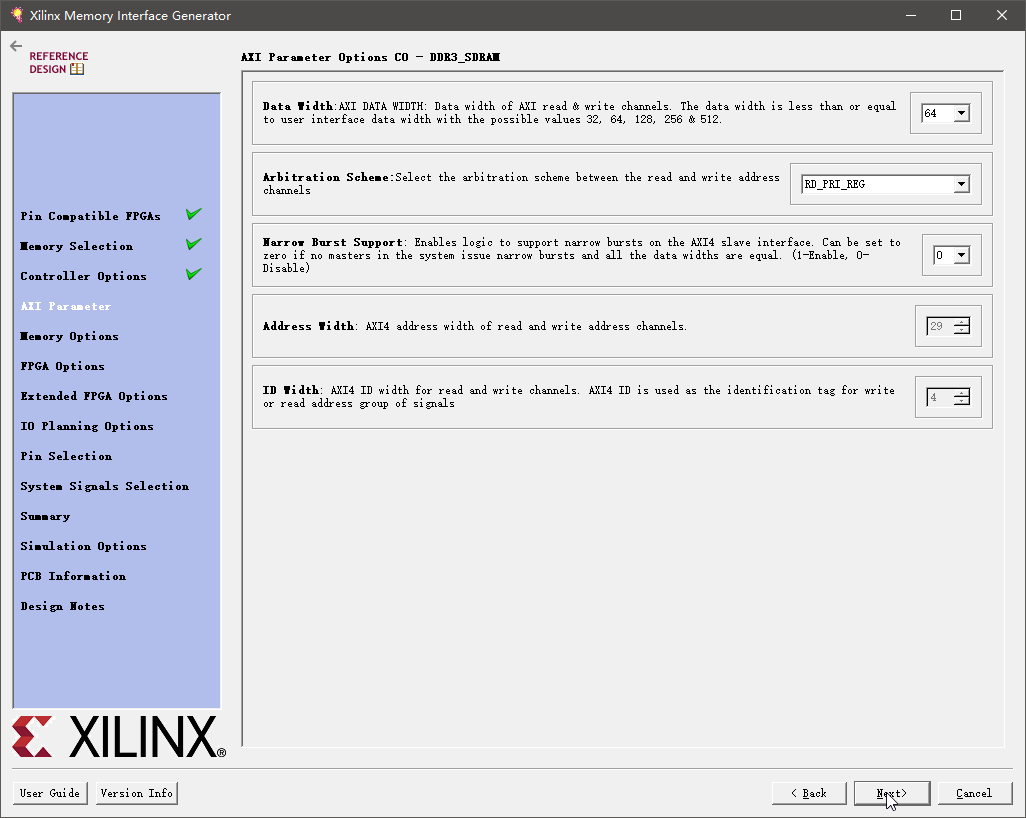


图2-29

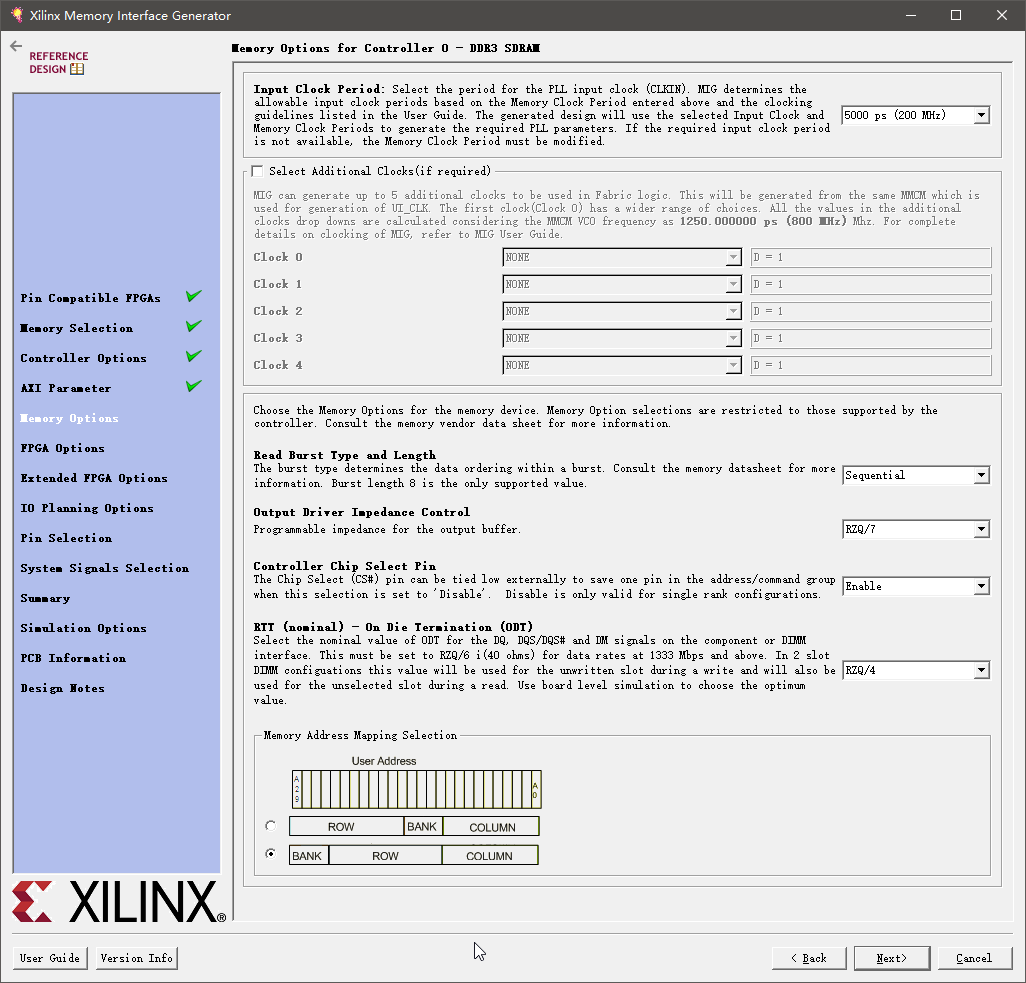


图2-30

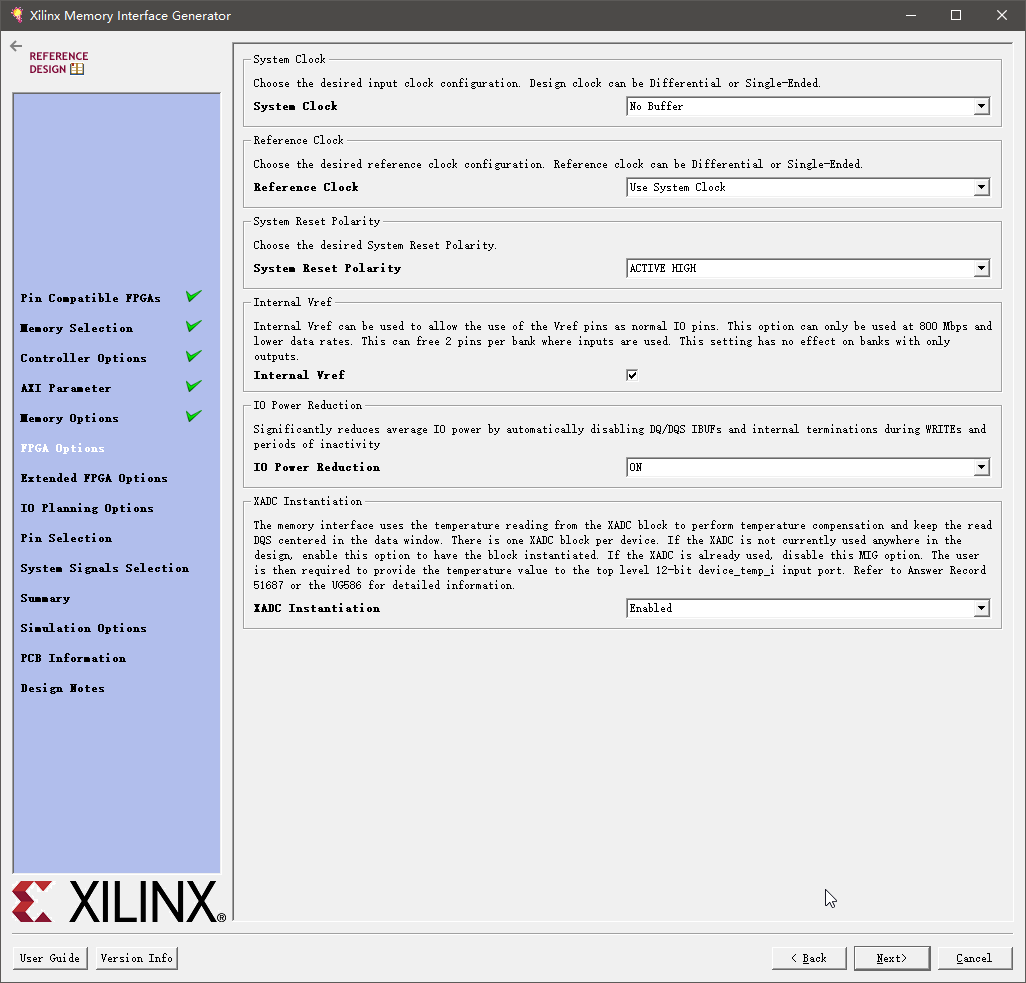


图2-31

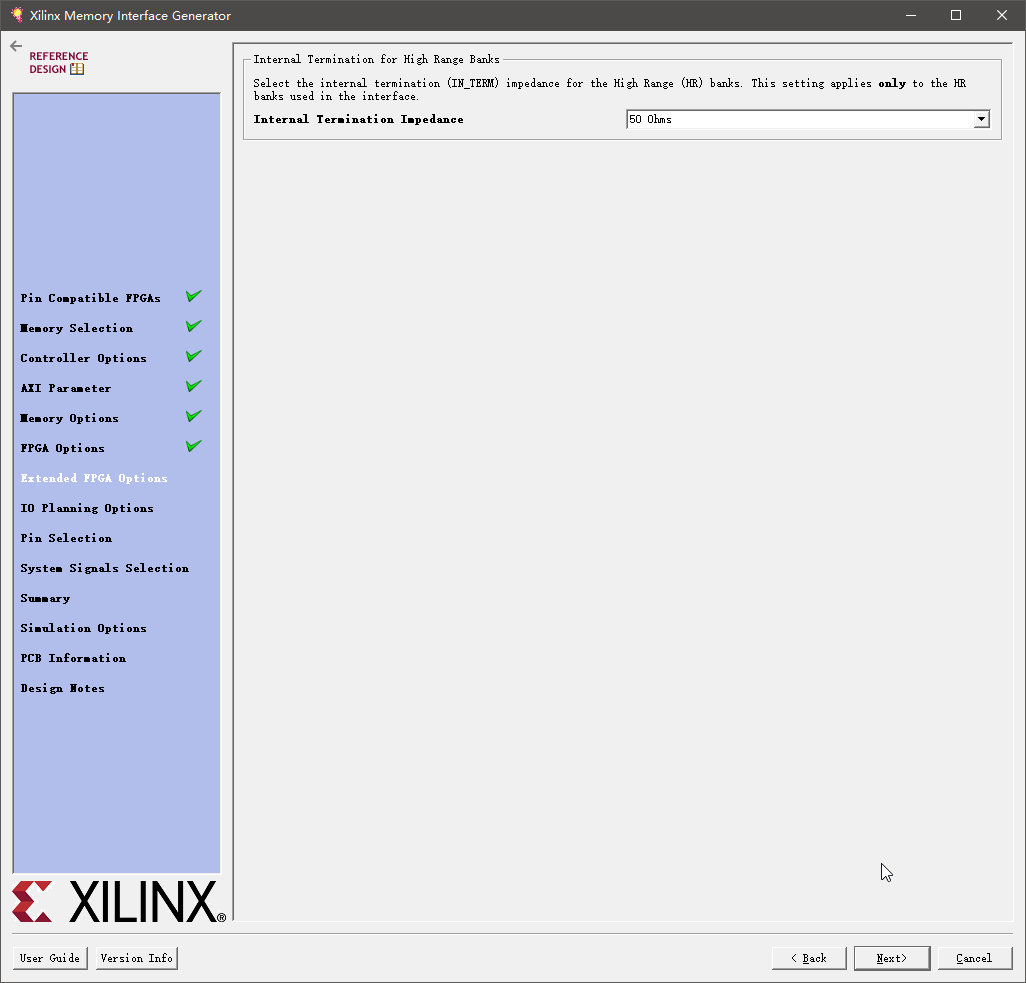


图2-32

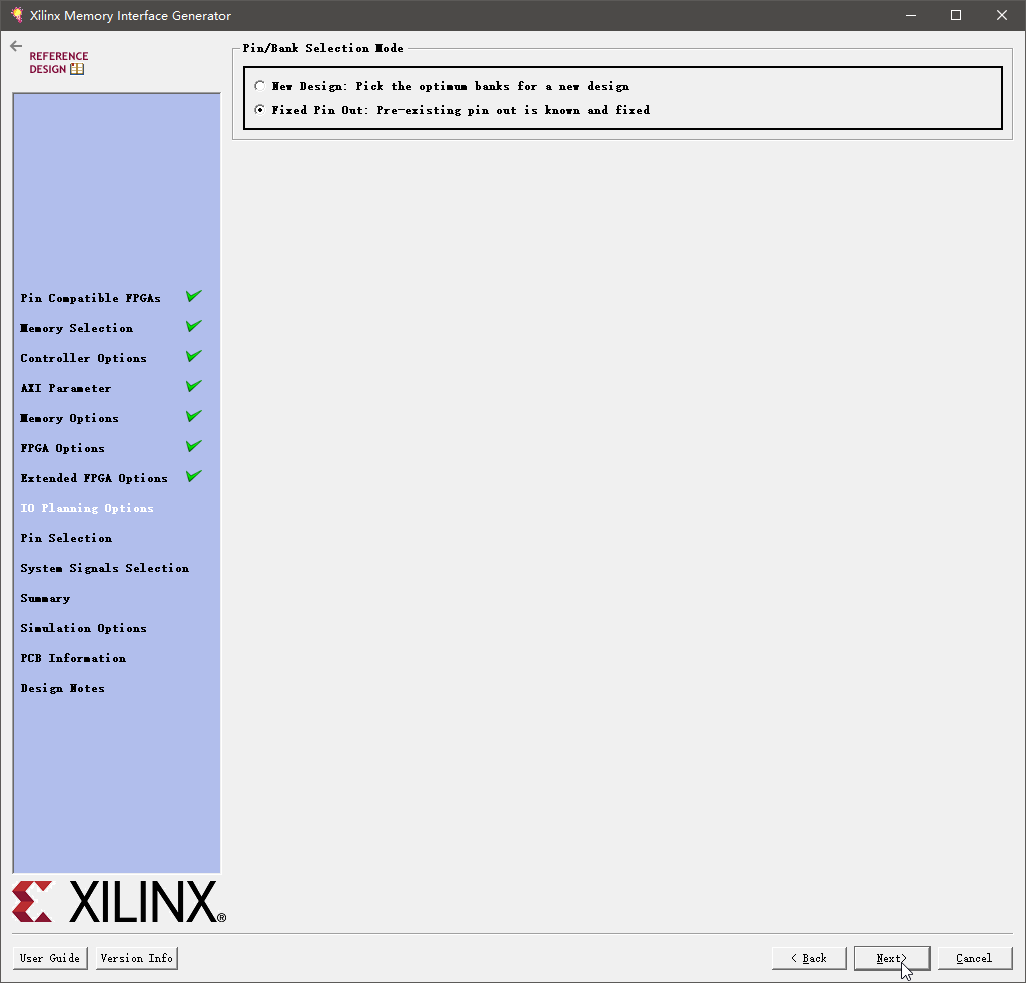


图2-33

在这一步你应该看到的是空白的。这个时候找到这份文件（MIPS\_FPGA/ constraints/ Minisys \_DDR\_Pin\_Map.ucf），用来约束Minisys上的DDR3 SDRAM，安排引脚。

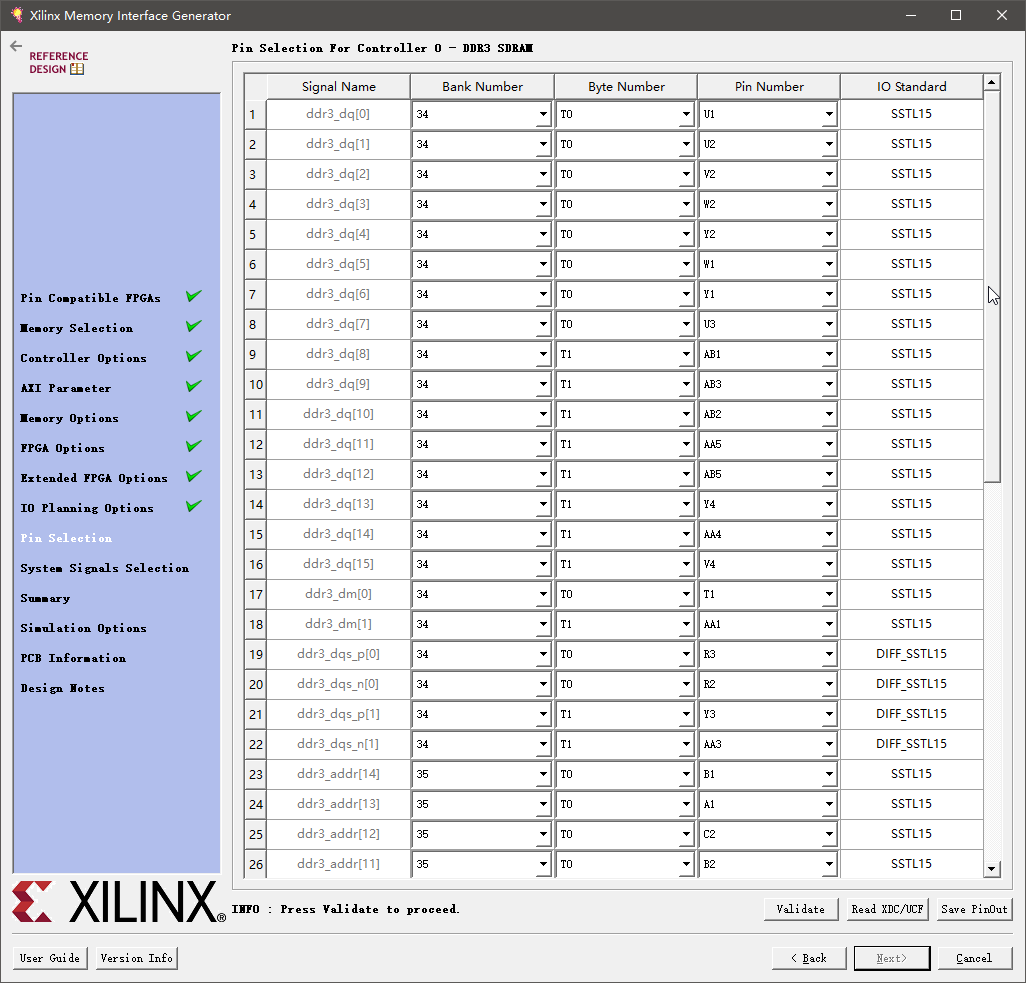


图2-33

根据下列步骤导入ucf。

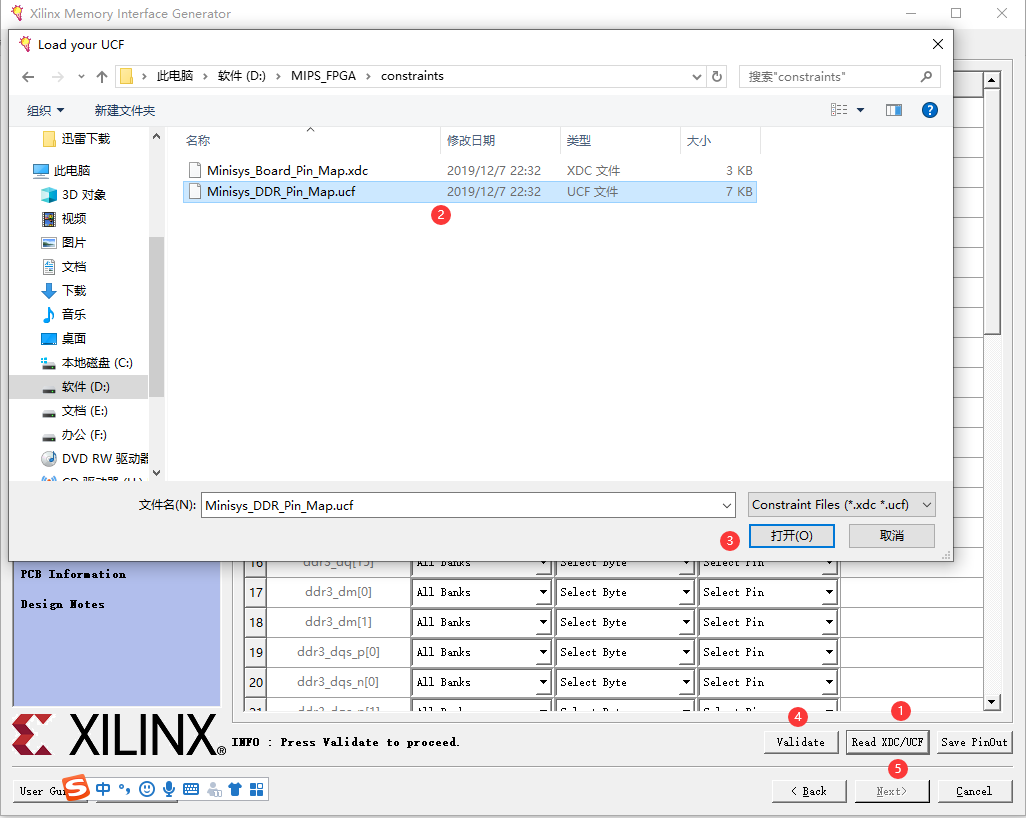


图2-34

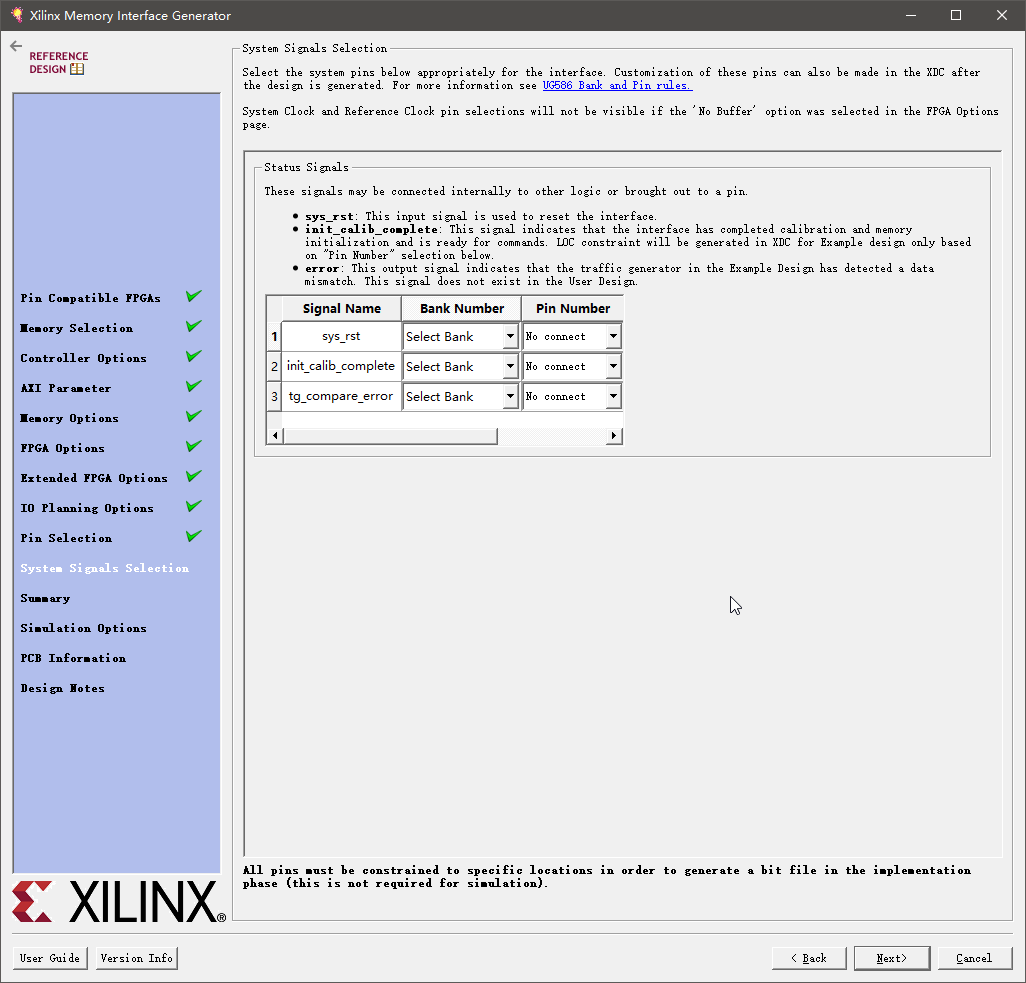


图2-35

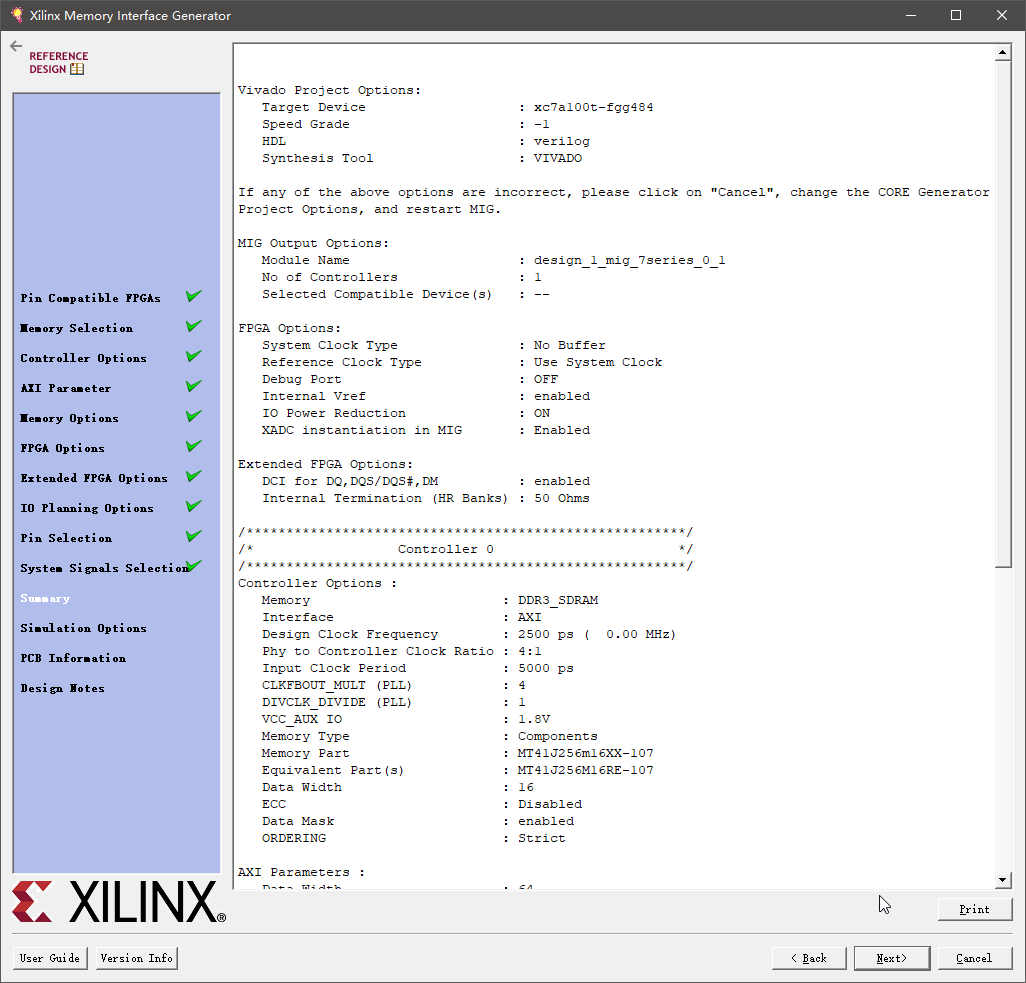


图2-36

之后没有配置项，直接下一步到结束即可。

MIG引脚的连接：

* 添加完成后的MIG IP核如图2-37所示，其中S\_AXI与AXI Interconnect的一个Matser接口相连（本教程是M07\_AXI）。
* sys\_rst直接连在CPU\_RESET上即可。
* sys\_clk\_i与之前创建的时钟向导的clk\_out2（200MHz，MIG需要较高频率的参考时钟）。
* aresetn与其他AXI设备的复位信号直接相连即可。
* 在DDR3接口处创建输出端口，并重名为DDR3\_SDRAM，如图2-38所示。

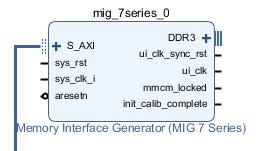


图2-37

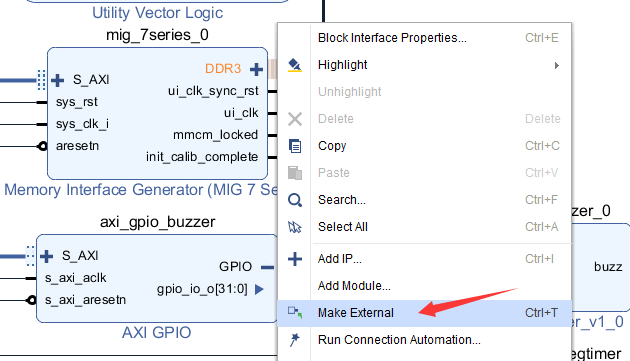


图2-38

* ui\_clk\_sync\_rst通过一个非门连接到AXI Interconnect的M07\_ARESET引脚。
* ui\_clk连接回AXI Interconnect的M07\_ACLK引脚。
* 剩下的两个引脚悬空即可。

至此，存储系统的构建完成。

### 2.3.5其他外设的添加

（1）UART

通用异步收发传输器（Universal Asynchronous Receiver/Transmitter)，通常称作UART。它将要传输的资料在串行通信与并行通信之间加以转换。作为把并行输入信号转成串行输出信号的芯片，用于与上位机的通讯。本教程中使用Xilinx提供的UART16650 IP核进行串口通信的控制。

该IP的详细功能、使用方法以及样例代码可参考该官方文档：

<https://www.xilinx.com/support/documentation/ip_documentation/axi_uart16550/v2_0/pg143-axi-uart16550.pdf>

* Add IP，找到AXI UART16650，双击添加。
* 该IP不需要自定义，这里直接使用默认配置。
* 连接该IP核到AXI Interconnect，如图2-39所示，该IP的freeze引脚置为低电平（使用CONSTANT），将UART接口展开，导出sin和sout引脚。
* 本教程中UART未使用中断模式，因此中断信号悬空。

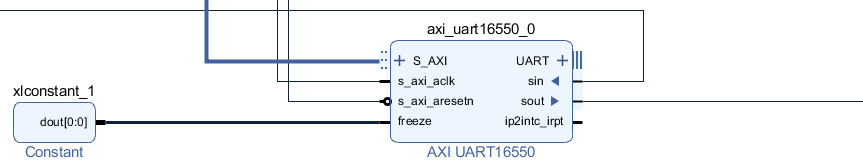


图2-39

（2）定时器

操作系统的运行离不开时钟中断，例如进程调度中的时间片轮转算法。定时器的作用就是产生时钟中断，本教程中使用Xilinx的AXI Timer IP核，该IP核可设置两个独立的定时计数器，读者可根据自己的需要进行配置。

该IP的详细功能、使用方法以及样例代码可参考该官方文档：

<https://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf>

本教程的使用方法仅作为参考。

* Add IP，找到AXI Timer，双击添加。
* 该IP核可根据实际需要决定是否启用第二个计数器。以及计数器的最大值。如图2-40所示。
* 连接IP到AXI Interconnect，将该IP的interrupt信号引出，连接到一个8位的concat的某个引脚（本教程连接到0号引脚，这里最好参考一下MIPSfpga中断信号的引脚定义），这个拼接后的8bit信号，直接送到CPU的中断输入引脚。如图2-41所示。

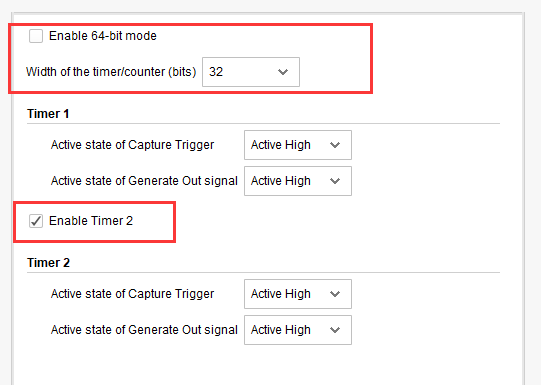


图2-40

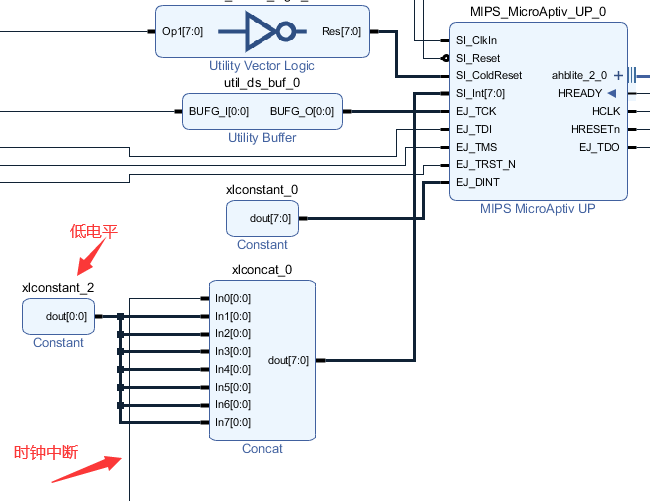


图2-41

（3）SD卡控制器

Minisys支持Micro SD卡的扩展，操作系统的镜像应放在SD卡中，由Bootloader加载到RAM中运行。SD卡的控制有SPI模式和SD模式，本教程选择了控制逻辑较为清晰的SPI模式进行SD的初始化以及控制。下面简单对SD卡的SPI模式进行简单的介绍。

首先介绍Micro SD卡的引脚定义，如图2-42所示。需要我们控制的有CS、MISO、MOSI、CLK。SPI模式通过对这四个信号的控制，我们可以实现对SD卡的初始化、读写操作。

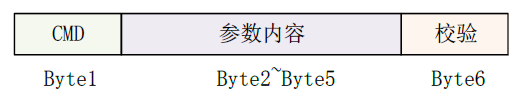


图2-42

SPI模式的命令类型与格式、返回数据格式、初始化时序和过程、读写时序和过程参见附录的内容。请读者阅读后再进行IP核的配置。

~~SPI模式是通过发送串行命令的方式，进行控制的。SD卡在正常读写操作之前，必须先对SD卡进行初始化，SD卡的初始化过程就是向SD中写入命令，使其工作在预期的工作模式。在对SD卡进行读写操作时同样需要先发送写命令和读命令，因此SD卡的命令格式是学习SD卡的重要内容。SD卡的命令格式由6个字节组成，发送数据时高位在前，SD卡的写入命令格式如图2-43所示：~~

* ~~Byte1：命令字的第一个字节为命令号（如CMD0、CMD1等），格式为“0 1 x x x x x x”。命令号的最高位始终为0，是命令号的起始位；次高位始终为1，是命令号的发送位；低6位为具体的命令号（如CMD55，8’d55 = 8’b0011\_0111，命令号为 0 1 1 1 0 1 1 1 = 0x77）。~~
* ~~Byte2~Byte5：命令参数，有些命令参数是保留位，没有定义参数的内容，保留位应设置为0。~~
* ~~Byte6：前7位为CRC（循环冗余校验）校验位，最后一位为停止位0。SD卡在SPI模式下默认不开启CRC校验，在SDIO模式下开启CRC校验。也就是说在SPI模式下，CRC校验位必须要发，但是SD卡会在读到CRC校验位时自动忽略它，所以校验位全部设置为1即可。需要注意的是，SD卡上电默认是SDIO模式，在接收SD卡返回CMD0的响应命令时，拉低片选CS，进入SPI模式。所以在发送CMD0命令的时候，SD卡处于SDIO模式，需要开启CRC校验。另外CMD8的CRC校验是始终启用的，也需要启用CRC校验。除了这两个命令，其它命令的CRC可以不用做校验。~~

~~~~

~~图2-43~~

~~SD卡的命令分为标准命令（如CMD0）和应用相关命令（如ACMD41）。ACMD命令是特殊命令，发送方法同标准命令一样，但是在发送应用相关命令之前，必须先发送CMD55命令，告诉SD卡接下来的命令是应用相关命令，而非标准命令。发送完命令后，SD卡会返回响应命令的信息，不同的CMD命令会有不同类型的返回值，常用的返回值有R1类型、R3类型和R7类型（R7类型是CMD8命令专用）。SD卡的常用命令说明如图~~

Xilinx提供了AXI Quad SPI IP核来实现对SPI设备的控制。该IP的详细功能、使用方法以及样例代码可参考该官方文档：

<https://www.xilinx.com/support/documentation/ip_documentation/axi_quad_spi/v3_2/pg153-axi-quad-spi.pdf>

下面介绍该IP的配置方法。

* Add IP，找到AXI Quad SPI这个IP核并添加。
* 双击该IP进行自定义，按照图2-43进行配置。
* 连接该IP置AXI Interconnect IP核。
* 创建输入端口spi\_miso，输出端口spi\_sclk、spi\_cs、spi\_mosi。展开SPI接口，进行如图2-44所示的连接。
* ext\_spi\_clk为SPI模式的参考时钟，本教程使用50MHz，直接连接到时钟向导的clk\_out1即可。
* 本教程未使用中断模式，中断引脚悬空。

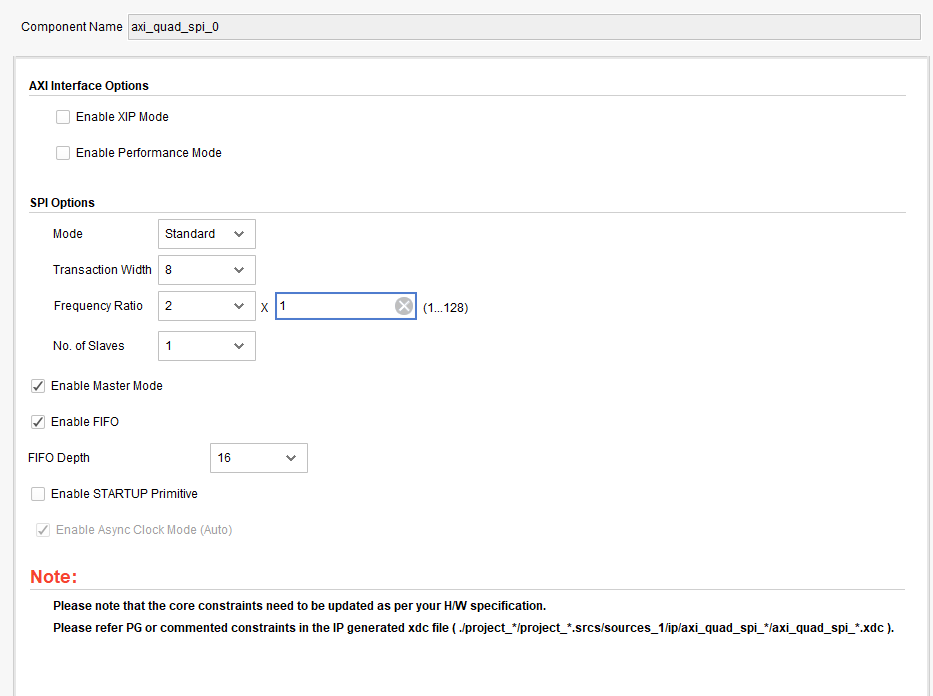


图2-43

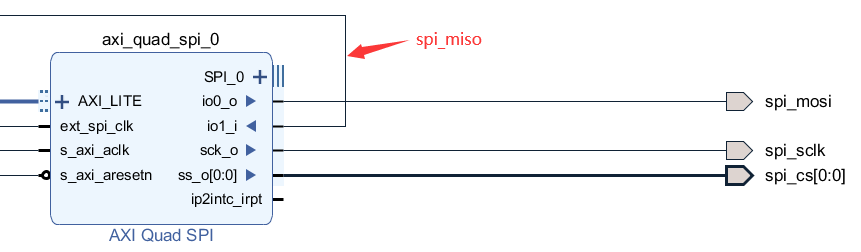


图2-44

### 2.3.6总线地址的分配

在添加完AXI设备之后，需要将这些设备映射到对应的**物理**地址上。回顾一下我们在1.3.1节中介绍的MIPS32 4G虚拟地址的划分规则，以及MMU映射策略。请读者思考一下，BRAM、DDR3、其他外设分别应该映射到4G的虚拟地址空间的哪一段？对应的物理地址是什么？

首先BRAM中的一个是用于存放Bootloader的，MIPS32的复位向量位于虚拟地址0xBFC0\_0000，对应物理地址为0x1FC0\_0000。另一个是Linux启动需要的，映射到虚拟地址0x9000\_0000，对应物理地址为0x1000\_0000。DDR3作为系统的主存储器，操作系统的内核放在其中，内核应起始于虚拟地址0x8000\_0000，对应物理地址为0x0000\_0000。kseg1（0xA000\_0000~0xBFFF\_FFFF）是内核段地址，由于该区域也不经过TLB映射，也不经过cache缓存，因此适合把外设（即外设的控制寄存器）映射到这个段地址中去。因此我们的外设，应该映射到这一段虚拟地址中，也就是物理地址0x0000\_0000 ~ 0x1FFF\_FFFF（注意物理地址应该互相隔离）。

本教程给出的地址分配如图2-45所示，读者可以参考进行分配。

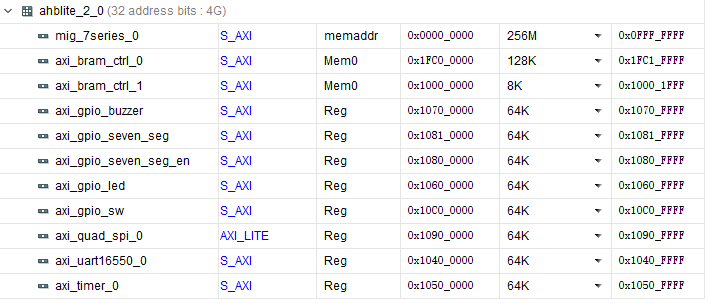


图2-45

## 2.4硬件工程的编译与验证

* 右键点击Block Design文件，选择Create HDL Wrapper，如图2-46所示，点击确定。
* 等待vivado自动生成HDL代码，就可以进行综合了。首先ctrl+s保存，然后点击左侧的Run Synthesis，等待完成。
* 综合完成后，点击Run Implementation，进行实现。

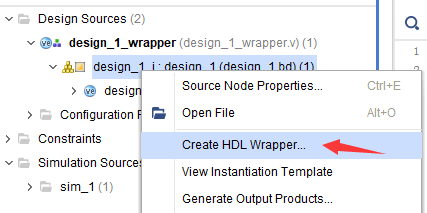


图2-46

* 实现完成后，现在导入约束文件。点击Add Source，选择Add or create constraints，如图2-47所示。然后点击下一步，创建约束文件。将教程中提供的约束文件内容复制进来（MIPS\_FPGA/ constraints/ Minisys\_Board\_Pin\_Map.ucf），然后保存。读者应注意自己的信号名是否与给的例程一致，若不一致应修改约束文件中的信号名。

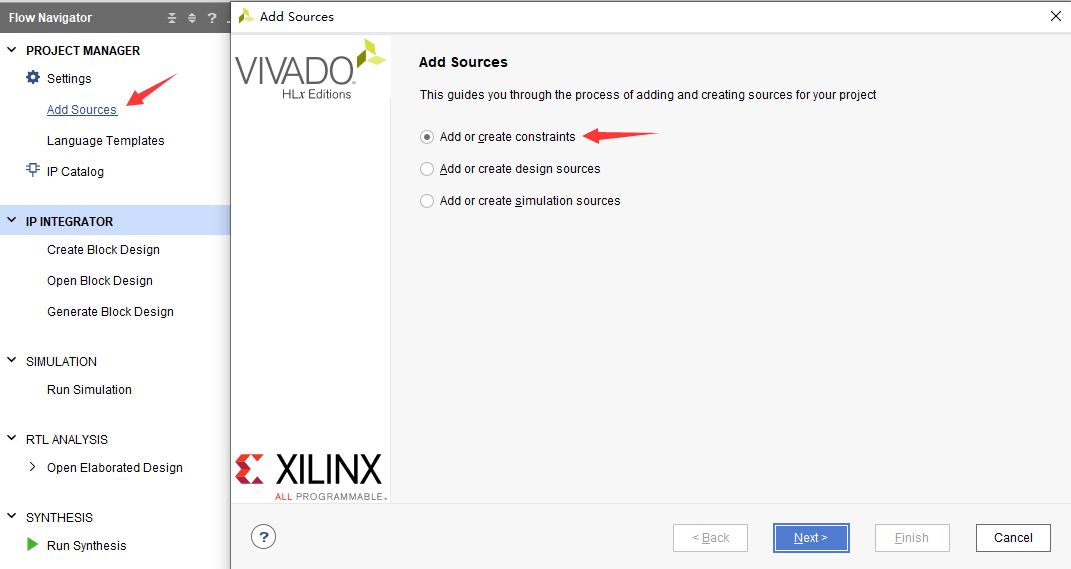


图2-47

* 在添加了约束文件后，就可以生成比特流了。点击Generate Bitstream，等待完成即可。
* 等待比特流生成完毕，点击Open Hardware Manager，选择Auto connect后，就可以进行下板测试了。选择生成好的bit流，点击program device。等待进度条完成后，首先将Minisys实验板的串口用Type-C下载线与PC的USB接口相连，并且打开实验板的电源开关。然后打开PC的设备管理器，找到端口(COM和LPT)，找到Minisys所在的端口，如图2-48所示，可以看出Minisys所在的端口号为COM4。紧接着打开putty，如图2-49所示，选择Serial，输入端口号为COM4，并且设置波特率为115200，然后点击Open即可开始与系统进行通信。
* 打开PuTTY后，按下Minisys开发板的复位键，此时串口循环打印MIPS FPGA，LED灯循环点亮即证明MIPSfpga-SoC移植成功（需要BRAM中加载教程提供的.coe文件）。



图2-48 Minisys的端口号

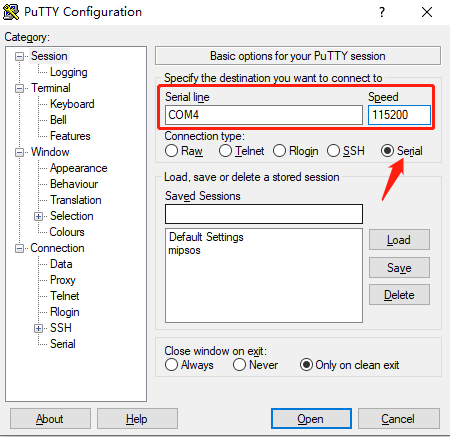


图2-49 putty设置界面

# 第三章 Bootloader的设计与实现

本教程提供的参考代码仓库地址：**https://github.com/flsgavin/MIPSfpga-Boot.git**

## 3.1工具链的使用

### 3.1.1 MIPS交叉编译器的使用

（1） Makefile的设计

Makefile是自动化编译工具，如果读者对Makefile不熟悉，请先自行学习关于Makefile的相关知识。这不是本教程的重点，但对于OS开发十分重要，对程序员来说也是一个基本技能。读者可以自行设计编写Makefile，也可以参考代码仓库中的Makefile的写法来完成。

（2）链接脚本的设计

链接脚本（ld脚本）是将目标代码链接到指定的虚拟地址的工具，需要配合Makefile一起使用。同样的，读者可以自行设计编写链接脚本，也可以参考代码仓库中的链接脚本的写法来完成。

### 3.1.2 GDB调试

GDB（GNU symbolic debugger）简单地说就是一个调试工具。它是一个受通用公共许可证即GPL保护的自由软件。

像所有的调试器一样，GDB可以让你调试一个程序，包括让程序在你希望的地方停下，此时你可以查看变量、寄存器、内存及堆栈。更进一步你可以修改变量及内存值。GDB是一个功能很强大的调试器，它可以调试多种语言。在此我们仅涉及 C语言的调试，而不包括其它语言。还有一点要说明的是，GDB是一个调试器，而不像 VC 是一个集成环境。

下面是我们搭建的Windows环境下，启动GDB并运行、调试程序的过程。

* win+R打开cmd控制台黑窗，输入以下命令，回车

cd C:\Program Files\Imagination Technologies\OpenOCD

openocd-nexys4.bat

* 此时应该弹出图3-1所示的窗口。
* 这时进入MIPS\_FPGA目录，找到Starter\_Tutorial并进入，在图3-2的位置输入cmd并回车，启动一个新的控制台。
* 在新的控制台输入以下命令并回车：

“C:\Program Files\Imagination Technologies\Toolchains\mips-mti-elf\2017.10-05\bin\mips-mti-elf-gdb.exe” -x startup.txt

* 这时应出现图3-3所示的界面，按下Ctrl + C，此时出现“(gdb)”字样，就可以输入GDB命令了。
* 我们提前把要load的编译好的elf文件复制到Starter\_Tutorial目录下，输入GDB命令：load xxx（xxx为elf文件的文件名）并回车，此时elf镜像会被加载到内存中。如图3-4所示。
* 如果需要调试，则需要输入file xxx（xxx为elf文件的文件名）来加载symbols，加载完成后可以输入b命令来设置断点。
* 这时输入continue即可运行加载到内存中的代码到下一个断点处了。
* 常用的GDB命令参考网上的教程，本教程中常用的命令有info reg、b、continue、s、n等等。

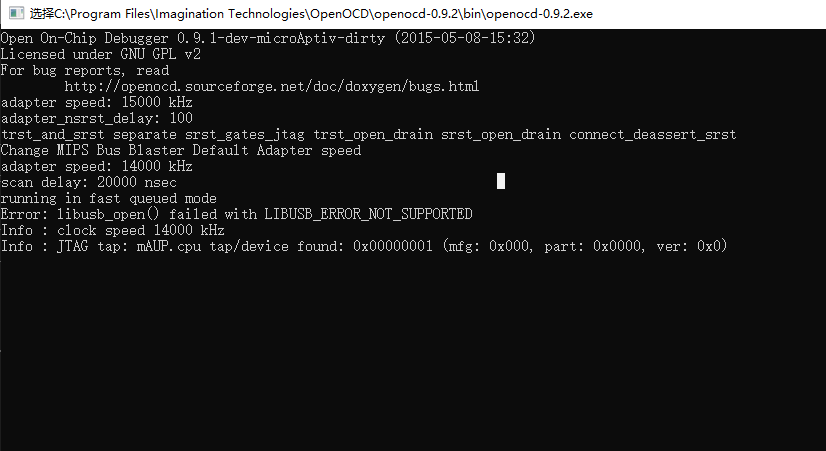


图3-1



图3-2

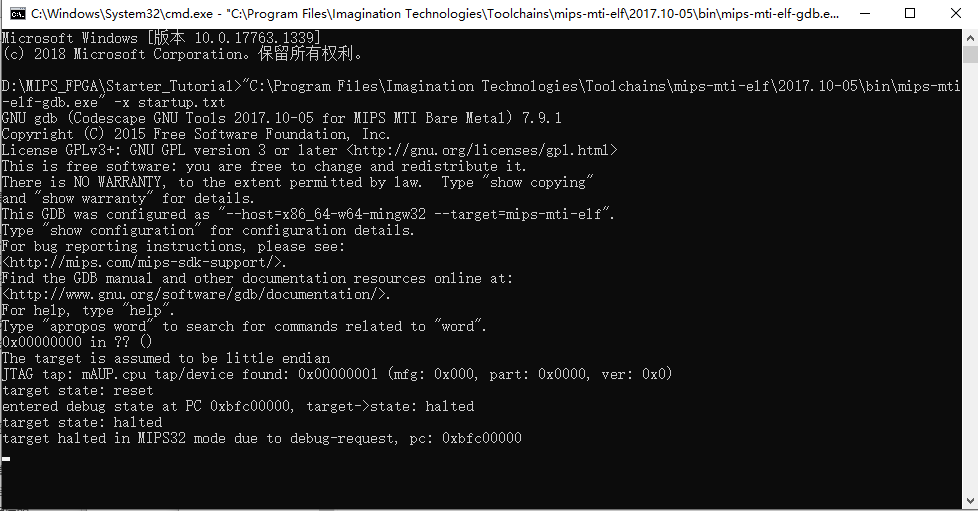


图3-3

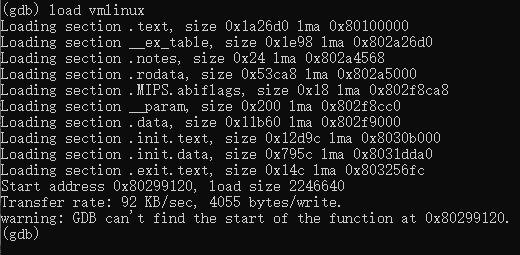


图3-4

## 3.2驱动程序的设计与实现

根据本教程的硬件系统设计，操作系统的设备驱动程序支持LED、拨码开关、蜂鸣器、串口、定时器、七段数码管以及SD卡的驱动。

（1）GPIO设备驱动

硬件中通过GPIO控制的外设有LED、拨码开关、蜂鸣器和七段数码管， LED与拨码开关没有控制逻辑，蜂鸣器与七段数码管得控制逻辑已经在硬件IP核中实现，因此这些外设可以直接通过对GPIO的读写来实现控制与访问。其读写控制函数如表3.1所示。

表3.1 GPIO外设读写控制函数

|  |  |
| --- | --- |
| 函数名 | 功能描述 |
| set\_buzzers  set\_leds  get\_switchs  set\_seven\_seg\_value  get\_seven\_seg\_value  set\_seven\_seg\_en  get\_seven\_seg\_en | 设置蜂鸣器频率  设置24个LED的状态  获取24个拨码开关的状态  设置七段数码管的显示  读取七段数码管的显示  设置七段数码管的使能  获取七段数码管的使能 |

（2）定时器驱动

定时器在Bootloader中是用不到的，但是对于OS的开发来说十分重要。进程调度的时间片算法依赖于设置外部的时钟中断。写在这里供读者参考。

定时器Timer的控制逻辑较为复杂，因为其端口寄存器较多，控制逻辑也更为复杂。Timer IP核提供了两个定时器Timer0以及Timer1，本教程只用到了Timer0。Timer0拥有端口三个寄存器，如表3.2所示。

表3.2 Timer0端口寄存器

|  |  |
| --- | --- |
| 寄存器名 | 功能描述 |
| TCSR0  TLR0  TCR0 | Timer0控制和状态寄存器  Timer0加载寄存器  Timer0计数寄存器 |

其中TCSR0的格式如图3-5所示，

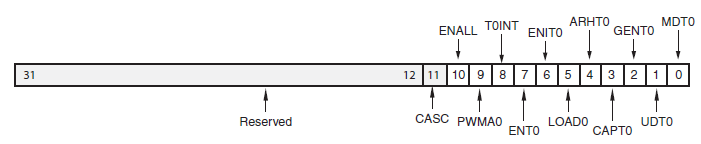


图3-5 TCSR0的格式

本教程的定时器初始化和设置需要用到的字段有ENT0：第7位，置为1表示使能Timer0。ENIT0：第6位，置1为表示使能Timer0的中断信号。ARHT0：第4位，置为1表示Timer0每次计数完成后会自动加载计数值。计数值位于TLR0中。

驱动程序在初始化Timer0时，首先清空TCSR0寄存器，接着设置上述字段为1，目的是使能Timer0、中断和自动加载计数值。然后设置用户输入的计数值到TLR0中，即可完成对Timer的驱动。每当Timer0计数到TLR0中的数时，就会产生中断信号。

（3）串口驱动

Xilinx的UART 16650 IP核中共有13个端口寄存器。本教程对串口的驱动用到了如表3.3所示的五个寄存器。

表3.3 UART16650端口寄存器

|  |  |
| --- | --- |
| 寄存器名 | 功能描述 |
| RBR  THR  LCR  DLL  DLM  IER | 数据接收缓冲寄存器  发送保持寄存器  线路控制寄存器  除数锁存（最低有效字节）寄存器  除数锁存（最高有效字节）寄存器  中断使能寄存器 |

UART的驱动逻辑是，首先对LCR、DLL以及DLM三个寄存器进行初始化，初始化的代码如图3-6所示。查询Xilinx的手册，DLL的作用是设置波特率的除数最低有效字节，其值的计算方法如公式3.1所示：

DLL = Faxi / (16 \* baudrate) （3.1）

Faxi表示AXI总线的时钟频率，baudrate指的是要设置的波特率。本教程的波特率为115200，AXI总线时钟频率为50MHz。因此初始化设置DLL的值为：50 \* 106 / 16 / 115200 = 27

这个值没有超过8bit，因此DLM需要设置为0。

本教程的串口不使用中断方式，因此初始化时设置IER为0。

初始化完成后，就可以通过RBR和THR进行串口数据的收发了。至此，串口的驱动设计就完成了。

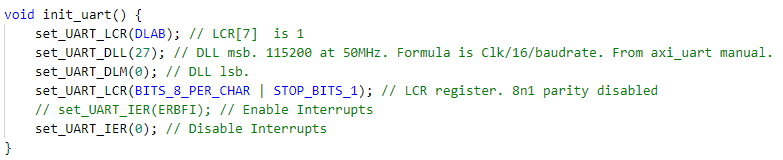


图3-6 串口驱动初始化

**请读者根据给出的串口驱动程序，自行设计一个简单的printf函数，该函数可以实现通过串口打印一个字符串和一个参数到PC的PuTTY中显示出来。**

（4）SD卡驱动

SD卡的驱动程序较为复杂，大致可以分为：

* 对AXI\_QUAD\_SPI IP核的驱动
* 对SD卡SPI模式的初始化以及读写的驱动
* 对SD卡文件系统FAT32的驱动

其中需要读者完成的部分是第二部分，即三个函数：disk\_initialize, disk\_read, disk\_write.请读者仔细阅读附录中对SD卡SPI模式的介绍后，根据已有的代码和提供的函数模板进行相关代码的编写。

## 3.3 Bootloader的实现

Bootloader有两大功能：boot和load。其中，boot的主要功能是将软硬件进行初始化，例如设置通用寄存器值、初始化堆栈、初始化TLB、初始化cache等；load的功能就是要将操作系统的内核代码从SD卡加载到内存。

读者在Bootloader实验中，需要实现的功能有两个：

* 完成初始化TLB代码
* 完成load\_elf代码

### 3.3.1初始化TLB

在初始化TLB之前，首先应读取Config寄存器和Config1寄存器，来读取TLB的size。然后对每一对虚拟地址入口，写入不同的物理页帧号。同时需要设置的寄存器有EntryLo0、EntryLo1、EntryHi、Index、Pagemask。应当注意的是，应保证每个物理页帧号无冲突，建议使用8k作为间隔。

初始化TLB的汇编函数模板如图3-7所示，请读者根据自己的理解，将其补充完整。

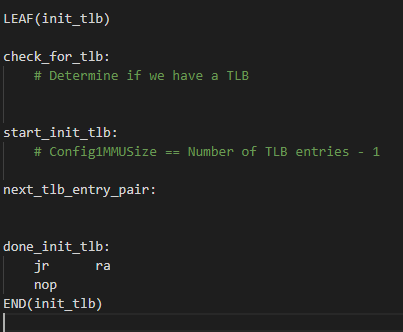


图3-7

### 3.3.2 Load elf文件

在加载elf文件到内存之前，首先需要了解elf文件的结构。我们编译好的elf文件由四个主要部分构成elf文件头Elf32\_Ehdr、程序头表Elf32\_Phdr、段表Elf32\_Shdr、符号表Elf32\_Sym。这些数据结构可以在drivers/elf.h中找到定义。

文件头Elf32\_Ehdr位于elf文件的起始位置，它包含整个文件的静态信息。其结构如图3-8所示。

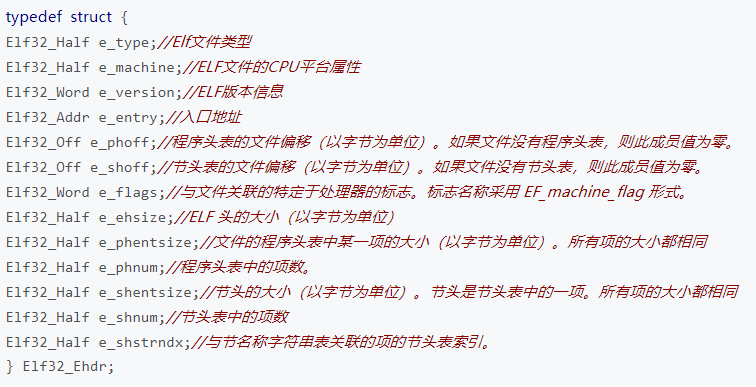


图3-8

程序头表Elf\_Phdr描述与程序执行直接相关的目标文件结构信息。用来在文件中定位各个段的映像。同时包含其他一些用来为程序创建映像所必须的信息。其数据结构如图3-9所示。

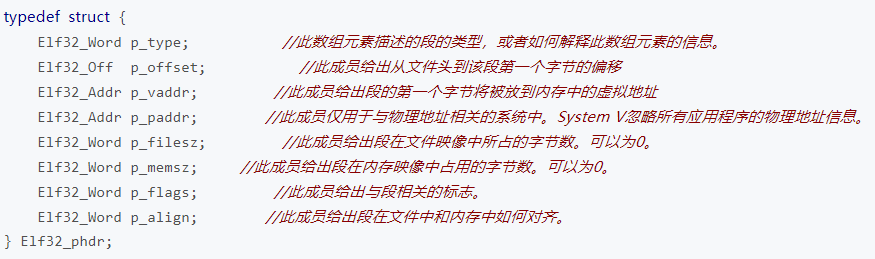


图3-9

本教程中使用上述两个elf文件的结构完成将elf镜像解析成可执行的二进制指令并加载到RAM中。

首先，在加载elf文件之前，elf文件已经被我们从SD卡通过FAT32文件系统的读操作完整的加载到内存的一块缓冲区中。这部分代码已经给出，如图3-10所示。

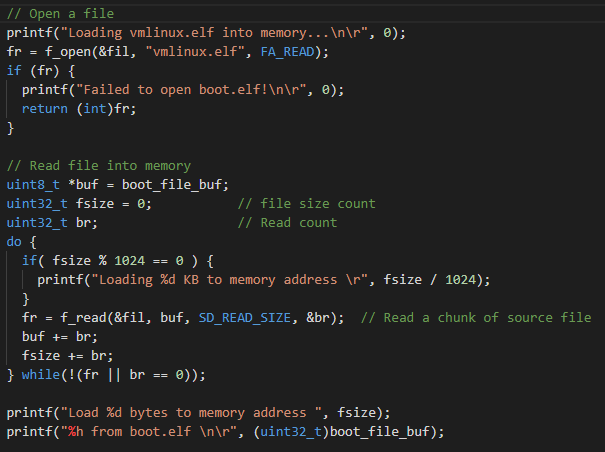


图3-10

那么这个缓冲区的起始地址就是elf文件头Elf32\_Ehdr的地址，根据该地址，我们可以得到e\_phoff，即程序头表的偏移地址（相对于起始地址）；以及e\_phnum，即程序头表的个数。接着我们可以根据这两个参数，顺序读取每个程序头表的信息：需要读取该段的物理地址、偏移和该段的字节数。使用提供的memCpy函数将各个段加载到其物理地址中。值得注意的是，如果内存映像字节数大于文件映像的字节数，那么超出的部分内存应该被置为0。

在学习了elf文件的格式后，读者需要根据如图3-11的函数模板完成load\_elf函数。

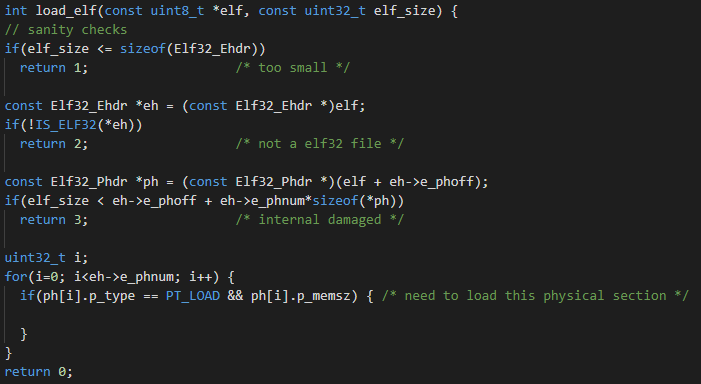


图3-11

在加载elf到内存中后，就可以跳转到入口地址执行了。读者可以自行编写代码，编译生成elf镜像文件，保存到SD卡。然后尝试其是否可以正确的被执行。

需要注意的是，SD卡中的elf的程序虚拟地址应保持在kseg0段，也就是内核所在的虚拟地址。因为该虚拟地址是不需要TLB映射的。如果读者想尝试将代码映射到用户段虚拟地址，则需要建立页表映射机制来进行TLB的填写。这就涉及到更进一步的页式内存管理的操作系统开发，读者可以在后续工作中去探索。