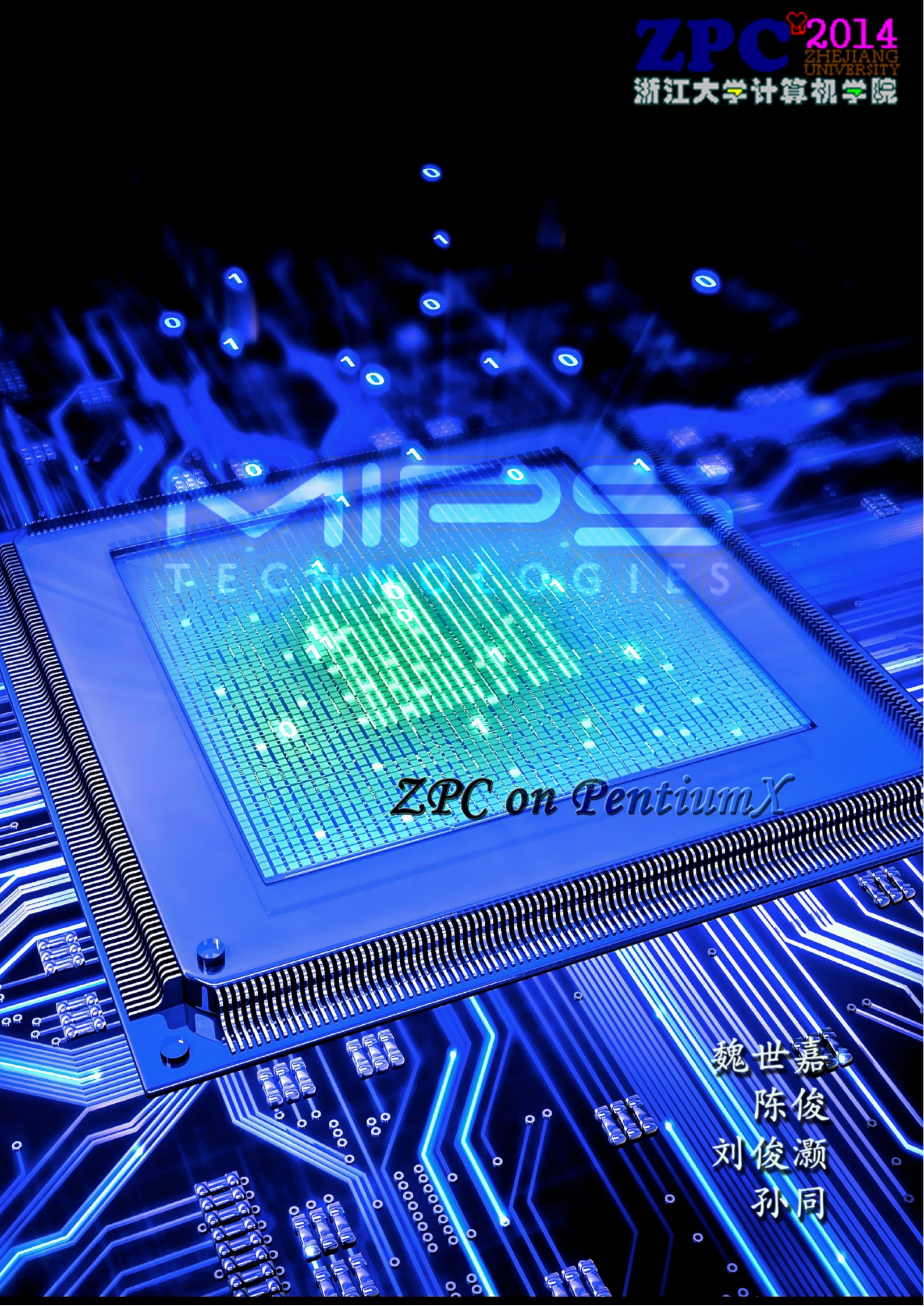
汇编与接口、接口实验

ZPC on PentiumX

魏世嘉
陈俊
刘俊灏
孙同

浙江大学

二零一四年十二月


ZPC 2014
ZHEJIANG UNIVERSITY
浙江大学计算机学院
ZPC on PentiumX
魏世嘉
陈俊
刘俊灏
孙同

目录

第 1 章 综述

1.1 物理地址分配

整个 SOC 系统的物理地址空间大致分为 RAM、ROM 和外设 I/O 三部分。外设 I/O 部分虽然较为复杂，但是所需要的地址空间很少，而 ROM 区域则用于存放 Bootloader，也不需要太大的空间。因此，整个物理空间的绝大部分将用于 RAM 空间。

- 0x0000_0000～0x0000_FFFF：RAM 空间，16KB，用作系统内存。
  - 其中栈起始地址 0x0000_6000
  - 代码段 0x0000_3000
- 0x0001_0000～0xFFFF_FFFF：外设 I/O 空间,实际上并没有用到如此多的外设接口，也无需那么多 I/O 空间

外设部分，目前只实现了一部分，具体可以参考第 5 章的相关内容。

1.2 MIPS 多周期 CPU 设计

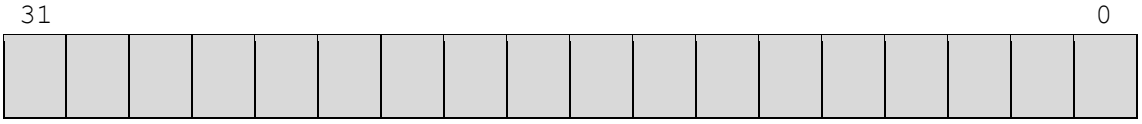
本 CPU 在上学期计算机组成与软硬件接口课程的基础上，继续使用多周期设计作为 CPU 结构的基础。增加了指令集，详见《MIPS 汇编语言规范--PentiumX》，另外对外部中断以及软件自陷进行了一定程度上的支持。

第 2 章 CP0 设计

CP0 全称为协处理器 0（co-processor zero），本 CPU 的 CP0 实现并不完全与 MIPS 兼容，仅仅实现了中断控制器的部分功能。

2.1 寄存器设计

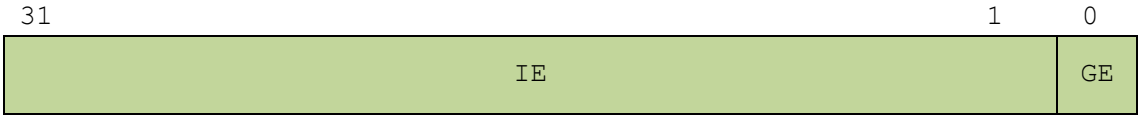
在本段中，寄存器各个位及其功能使用如下表格表示：



已实现的 CP0 寄存器按照 MIPS 标准编号，具体如下：

2.1.1 中断使能寄存器（Interrupt Enable Register，IER）

编号：11



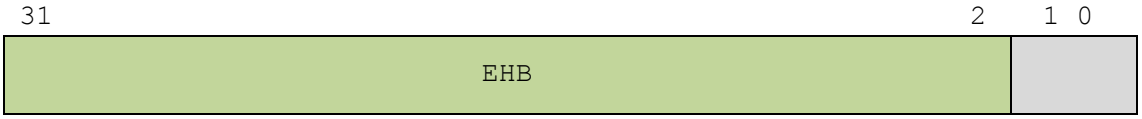
IER 保存对可屏蔽中断的使能控制信息。该寄存器可读写。

GE：全局中断使能位，置 1 表示开启全局中断使能。

IE[31:1]：单个中断使能位，置 1 表示开启相应位的中断使能。共有 31 个中断使能位，分别控制相应中断号对应的可屏蔽中断。 实际实现中用到的只有四个使能位，详见第四章。

2.1.2 异常处理基址寄存器（Exception Handler Base Register，EHBR）

编号：12

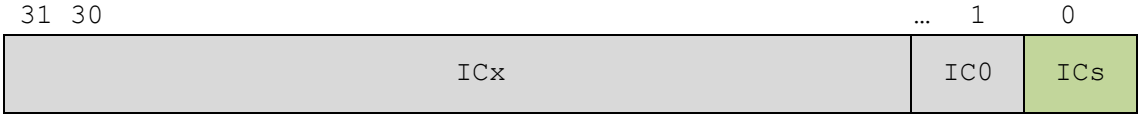


EHBR 保存当发生异常时 CPU 需要跳转到的目标地址。该寄存器可读写。

EHB：异常处理程序地址的高 30 位（其最低 2 位永远为 0）。

2.1.3 中断来源寄存器（Interrupt Cause Register，ICR）

编号：13



ICR 保存当前检测到的可屏蔽中断的信息。该寄存器可读写（特殊处理）。

IC[30:0]：中断产生信号，置 1 表示相应中断号的中断已产生，需要进行处理。共有 31 个中断产生信号，与 IER 中的 31 个中断使能位相对应。当全局中断使能为 0 或者对应的中断使能为 0 时，相应中断号的中断产生信号会置 1 但

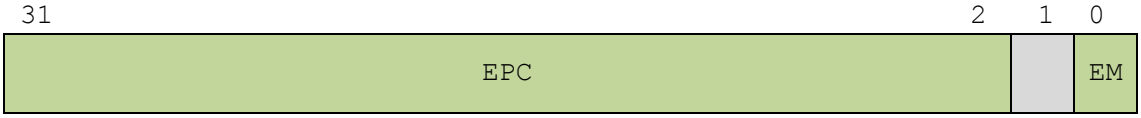
不会引发中断过程。中断处理程序在完成相关的中断处理后，中断源应根据中断响应情况清零中断标志。

从 0 位开始代表不同的中断源，最低位为 syscall

其余各位对应关系与中断使能寄存器（Interrupt Enable Register，IER）相同。

2.1.4 异常返回地址寄存器（Exception Program Counter Register，EPCR）

编号：14



EPCR 保存当前异常的返回地址。该寄存器可读写。

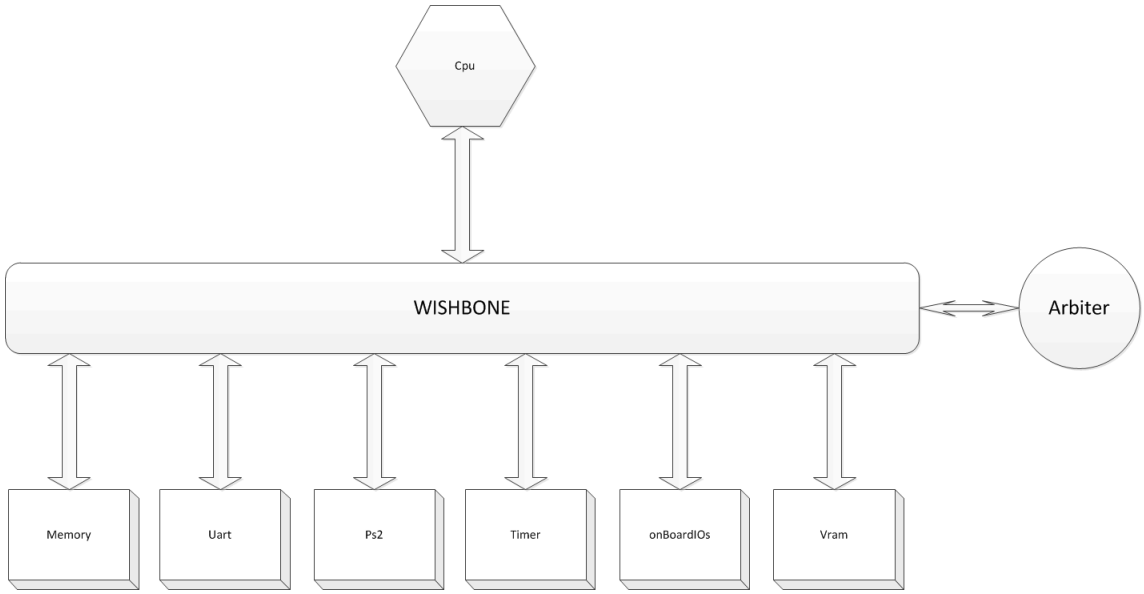
EPC：异常返回地址的高 30 位（其最低 2 位永远为 0）。

第 3 章 总线设计

总线在片上系统的设计过程中是提高系统运转稳定性的不可或缺的重要部件之一。同时，总线的设计另一方面简化了系统在连接外设时的复杂性。

本系统在设计时采用了通用的片上总线 WISHBONE 总线，将 CPU 作为总线上的主设备，外设视作从设备，主设备在发起从设备访问时采用同步握手的方式，进行访问，另外，设计了总线 arbiter，支持总线扩展，支持最多八个主设备八个从设备在总线上工作。但实际应用时只采用了 CPU 单一主设备设计。一定程度上简化了 WISHBONE 总线的线路设计，使得其更加适合本系统。

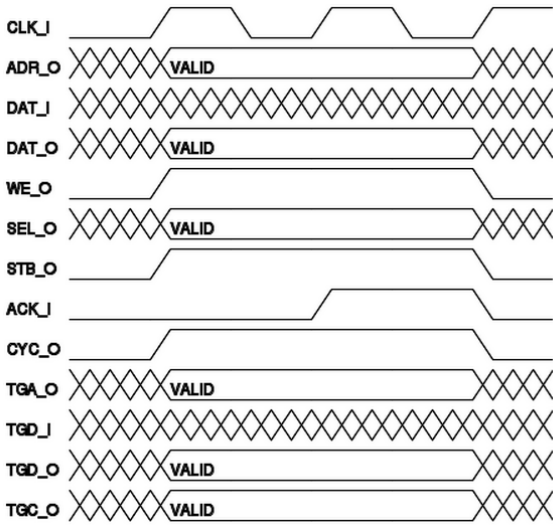
设计逻辑如下：



图表 1 WISHBONE 总线

系统设计支持主设备单次读写操作，但由于指令的限制，在总线块读写方面还存在欠缺。

典型地，一次 CPU 主设备的写存储器总线访问图如下：



图表 2 单次总线写操作

第 4 章 中断处理

中断处理是一个完整的 CPU 非常重要的功能，同时这部分也是 CPU 设计的难点。本 CPU 通过最简单的硬件和软件的配合实现了理论上无限的中断嵌套功能。

4.1 中断分类

CPU 的中断总共可以分为 2 个大类，按优先级从高到底排列为：外部中断，软件自陷

4.1.1 外部中断

外部中断是外设请求 CPU 进行信息处理的主要手段，外设由 CPU 进行控制和设定，在指定的条件满足后再通过中断的方式告知 CPU。与轮询的方式相比，使用中断可以大大提升 CPU 与外设进行数据通信的效率。外部中断受中断使能寄存器（IER）进行使能控制，其中断号从 0 开始，按中断号顺序排列如下：

4.1.1.0 计时器中断

当计时器所计时间达到 TIR 所指定的时间时发生。

4.1.1.1 板级输入中断

当使用开发板上的 SWITCH 或者 BUTTON 时发生。

4.1.1.2 键盘中断

当使用键盘进行输入时发生

4.1.1.3 UART 中断

当通过 UART 串口（作为系统硬盘）接收到数据需要 CPU 读取时发生。

4.1.2 软件自陷

软件自陷是用户态程序调用内核态的主要方式，所有资源均由操作系统进行管理，用户态程序通过软件自陷的方式“申请”资源，这样也大大提升了整个系统的可用性和安全性。

软件自陷由无条件自陷指令 SYSCALL 或者有条件自陷指令 TEQ、TEQI、TGE、TGEI、TGEIU、TGEU、TLT、TLTI、TLTIU、TLTU、TNE、TNEI 产生。

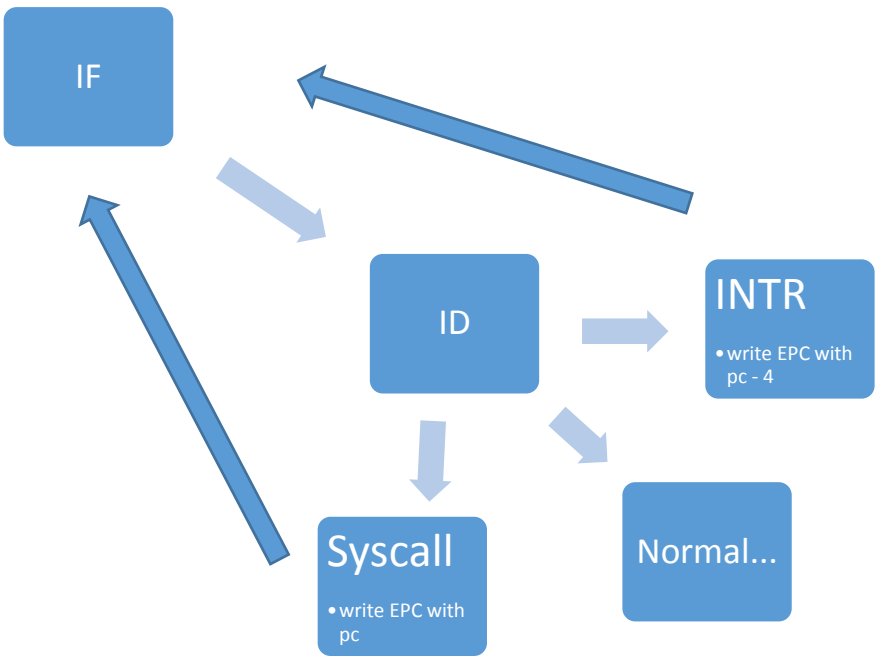
值得注意的是，内核态程序不允许使用软件自陷指令，在设计时也省略了对 Trap 类指令的支持。

4.2 中断细节

中断可以分为 2 大类，系统设计硬件完成对中断源进行优先级的分类，并实际单独的中断控制器

在中断检测时，为了将两大类中断处理尽量统一，采用在指令运行周期 ID 级检测中断的方式，同时检测外部中断和软件自陷情况，根据中断控制器优先级情况来确定当前 CPU 下一步的中断响应 方式。需要注意的是，当中断发生时（不论是外部中断还是软件自陷）CPU 需要将两者进行区别对待，即 CP0 中 **EPC** 保存的当前 PC 的值应分别为 PC 和 PC – 4 即当响应软件自陷中断时，当前指令无需再次执行，而当响应外部中断时，被中断的指令需要在中断相应程序返回后被再次执行。在 ID 级设计中断检测与响应决策一定程度上也是为了兼容 MIPS 经典五级流水的设计方式。

中断响应流畅如下：



另一方面，在中断处理过程中，为了实现中断嵌套，操作系统需要和硬件完成如下约定：

- 刚进入中断处理过程时，IER.GE 由硬件关闭，软件需尽快保存上下文。
- 软件保存好相关寄存器和 EPC 后，可以利用 *mtc0* 指令打开 IER.GE 允许中断嵌套。
- 软件完成中断相关的处理后，关闭 IER.GE，之后进行寄存器和 EPC 的恢复。
- 调用 eret 指令，硬件打开 IER.GE，同时当前中断返回。

中断处理程序处理完中断之后，需要外设根据相应情况将 ICR 中的相应位置为 0。

各种中断和异常使用统一的入口点，具体地址由软件通过 EHBR 寄存器设置，默认为系统启动的地址（即零地址加四），中断优先级全权交给软件处理，在进入中断入口后由系统软件进行外部中断和软件自陷的判断，在决定跳转至外部中断入口还是系统调用入口。详细中断入口及系统调用表见附录。

第 5 章 外设接口

外设 I/O 端口统一安排在地址 0x0000_FFFF 往上区域。内部又细分成 256 份，每份 256 字节，提供给每个外设接口，用于相关控制寄存器地址的映射。

这些设备都通过 wishbone 总线与 cpu 进行数据交换

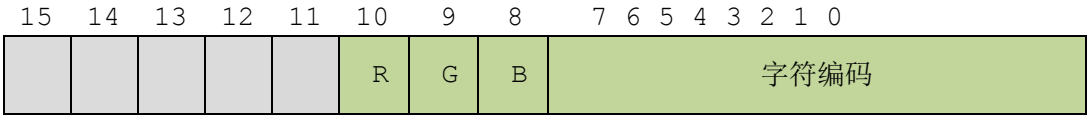
外设列表按外设编号排列如下：

5.1 VGA

VGA 只支持文本模式。该文本模式支持 8 色 16*16 点阵 ASCII 码（扩展了 128 位中文编码）显示，且支持硬件光标。

●VGA 的 IO 端口起始地址为 0x000C_0000，读写共用同一地址，显存大小为 1200x16 位

在文本模式下，每个字符使用两个字节表示，如下所示。



R：背景红色分量。

B：背景蓝色分量。

G：前景绿色分量。

我们默认背景色为黑色

字符编码：待显示的字符编码，我们在 ASCII 编码的基础上向上扩展了 128 位用于中文字，所以字符编码总共是 8 位，支持标准 ASCII 及 128 个汉字。

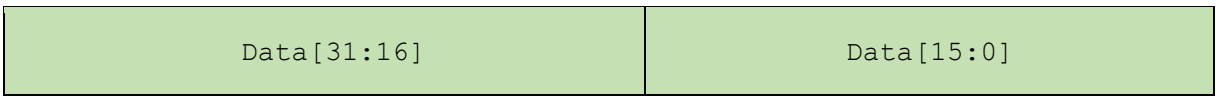
5.2 板级 IO

板级 IO 包括开发板上的基本输入输出，包括按钮，开关，LED 灯，七段数码管。按钮和开关输入包含基本的防抖功能，七段数码管则可直接显示 16 位的数据。

我们将板级 IO 统一通过另一条小总线 MIO_BUS 挂到我们总的 wishbone 总线上。

MIO_BUS 的起始地址为 0xFFFF_F000

● 0xFFFF_FE00 为七段码的写。



Data:送到七段码的数据 ， 有 32 位，但是七段码只能显示 16 位，由板上 switch 控制高低位切换。

● 0xFFFF_FF00 为 Led 和硬件光标的读写。

读：

Led: 板上 led 灯的状态

BTN: 板上 button 的状态

SW：板上 switch 的状态

写：

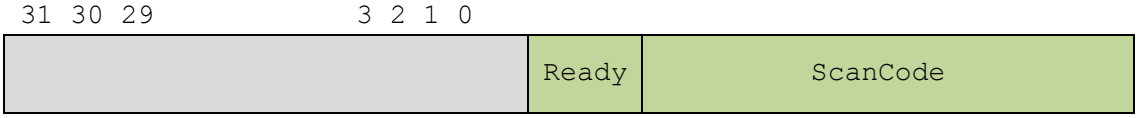
Cursor:硬件光标位置，高 6 位为行数，低六位为列数
LED：板上 led 的状态
Counter_set：计数器的选择（共有三个计数器和一个计数器控制位）

● 0xFFFF_FE00 为对计数器的读写。
读：
根据当前选择的计数器，读出其中的值。
写：
根据 Counter_set 选择写入哪个计数器或者计数器控制位。

5.3 PS2 键盘

当按下按键时键盘会发送相应按键的扫描码，然后我们的软件会将其转换成 Ascii 码。为了保证键盘发送的每个数据都能被获取到，我们在键盘模块中设置了 8 个字节的缓冲区。键盘控制器在接收到数据的时候产生中断。

●键盘的 IO 端口地址为 0xFFFF_D000，只读



Ready：数据有效标志，当键盘输入一个扫描码时，该位被置高，当CPU读走扫描码时，该位被置低。

ScanCode：数据位，8 位的扫描码

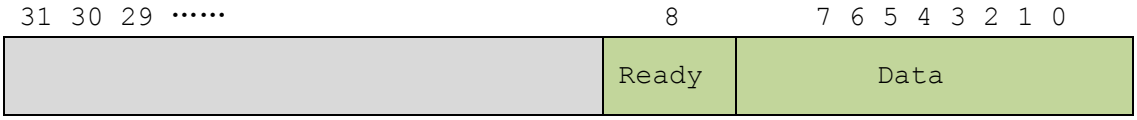
5.4 UART

因为我们通过 UART 来实现对磁盘的读写，因此对数据的收发速率要求都很高，所以我们在 UART 控制器内设置了相互独立的发送缓冲区和接收缓冲区。其中接收缓冲区需要固定为 512 字节的大小，因为我们规定每次只从磁盘读取一个大小为 512 字节的扇区，当 UART 接收到 512 字节的数据时便会产生一个中断通知 CPU，然后我们在中断响应程序中将这 512 字节的数据搬到我们内存中的磁盘缓冲区。

另外虽然有了缓冲区，但是发送数据仍显得频繁，缓冲区依旧十分容易满，对此我们主要有软硬件上两种策略。软件上我们在每次发送（这个发送指的是写入发送缓冲区）完一个数据块（每个数据块总是 512 字节）之后执行 Sleep()函数（重复的空操作）以等待数据发送完成。硬件上，当发送缓冲区满时，如果此时 CPU 想要向发送缓冲区中写入数据则 CPU 会一直处于等待状态直到缓冲区不再是满的，此时它才能将数据写入。

●UART 的 IO 端口地址为 0xFE00_D000，读写共用同一地址

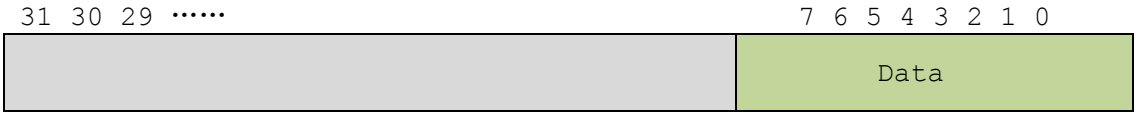
读 UART：



Ready：数据有效标志位。当 CPU 缓冲区有数据时，该位被拉高。

Data：读到的数据。

写 UART：



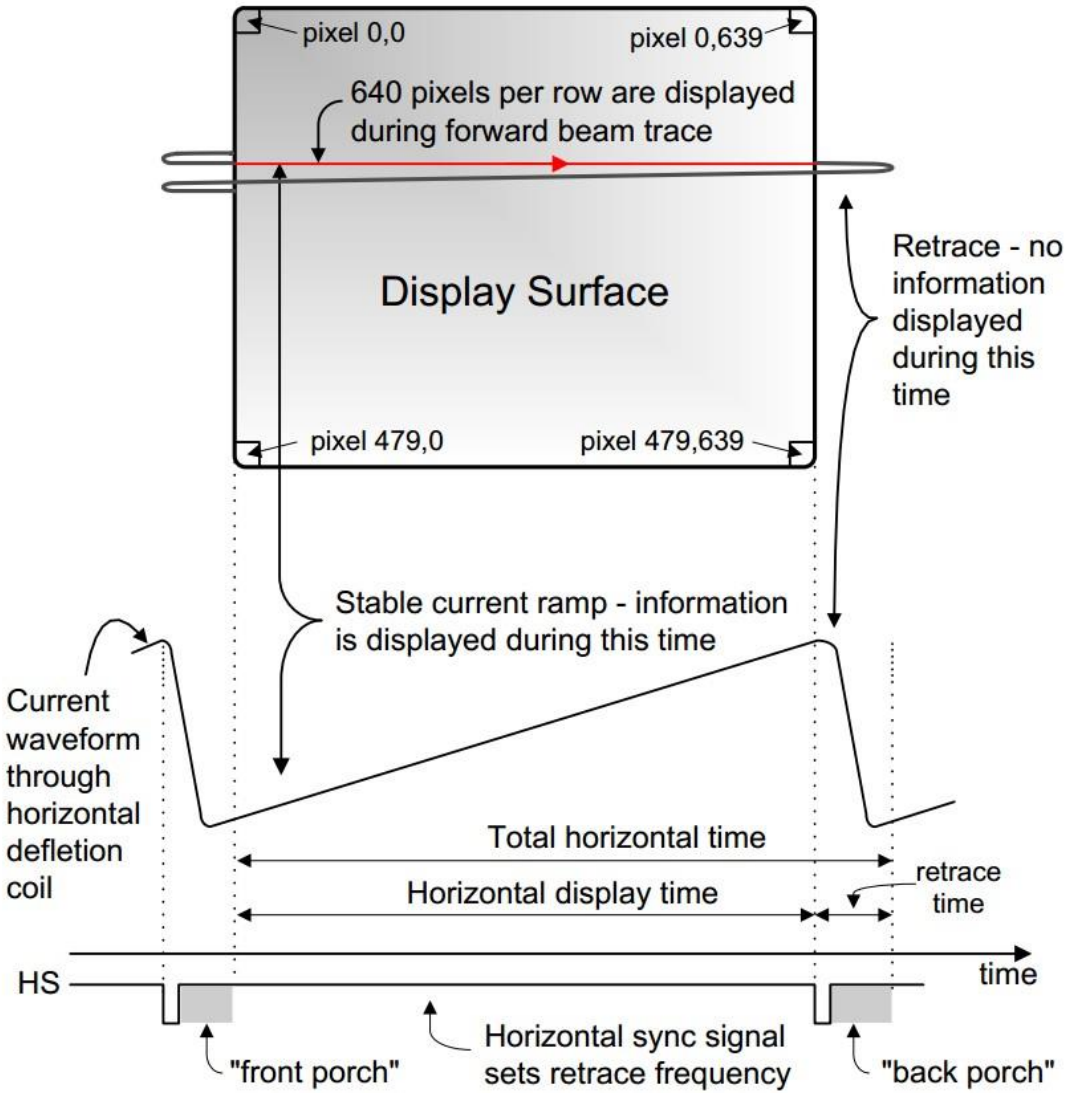
Data：写入缓冲区的数据。

第 6 章 外设实现

6.1 VGA

VGA（Video Graphics Array）是广泛使用的一种视频传输标准，最早是指640x480 分辨率的显示模式，后来基于此发展出了很多不同的分辨率，由于使用的是相同的接口，也统称为 VGA。

VGA 的时序主要由行同步信号（HS）和场同步信号（VS）控制。最早是为了 CRT 显示器逐行扫描的显示原理而设计的，现在 LCD 显示器的显示原理已经完全不同了，但也都兼容这种 VGA 的信号格式。标准 VGA 信号的时序图如下：



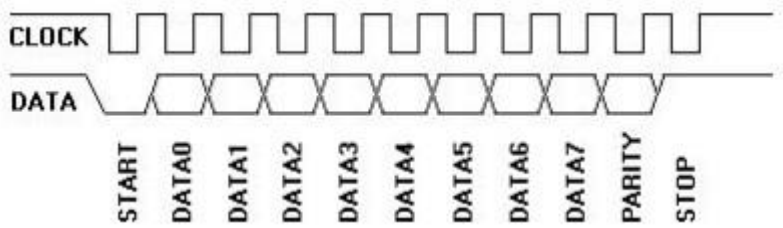
VGA 信号在每个行周期或场周期中，同步脉冲前后各有一段空余时间，分别叫做前沿（front porch）和后沿（back porch）。VGA 在不同分辨率下，其同步时间，显示时间，前沿和后沿时间都是不同的。

具体实现详见 VGA_IO.v

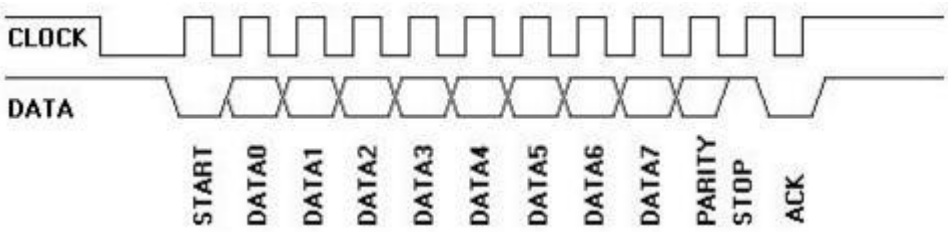
6.2 PS2

PS2 接口主要用于老式的键盘和鼠标，时序比较简单且速度较慢。

PS2 主要由 1 根时钟线和 1 根数据线组成。在没有数据传输时，两者均为高电平。在数据传输过程中，时钟线的频率一般在 15kHz 左右，由设备提供，主机在时钟线为低电平时采样或者输出数据。



上图是设备发送数据到主机的时序图。设备先拉低数据线再拉低时钟线表示一个数据帧的开始，之后发送 8 位数据位，1 位奇校验位和 1 位停止位。主机需在时钟线为低电平时采集数据，在设备发送数据到主机的过程中，主机可随时拉低时钟线来中断数据传输。



上图是主机发送数据到设备的时序图。主机首先拉低时钟线至少 100 微秒表示需要发送数据，然后拉低数据线，之后放开时钟线。之后设备会产生时钟信号，主机在时钟线为低电平时改变数据线，依次发送 8 位数据位，1 位奇校验位和 1 位停止位。之后设备会返回一个应答位，若主机在发送完停止位后 100 微秒内没有接收到应答位，则会产生一个错误。

在 Spartan-6 上编写 PS2 模块的时候，特别注意需要在 UCF 引脚约束文件中给 PS2 的两个引脚加上“PULLUP”约束，表明需要连接上拉电阻。

6.3 串口

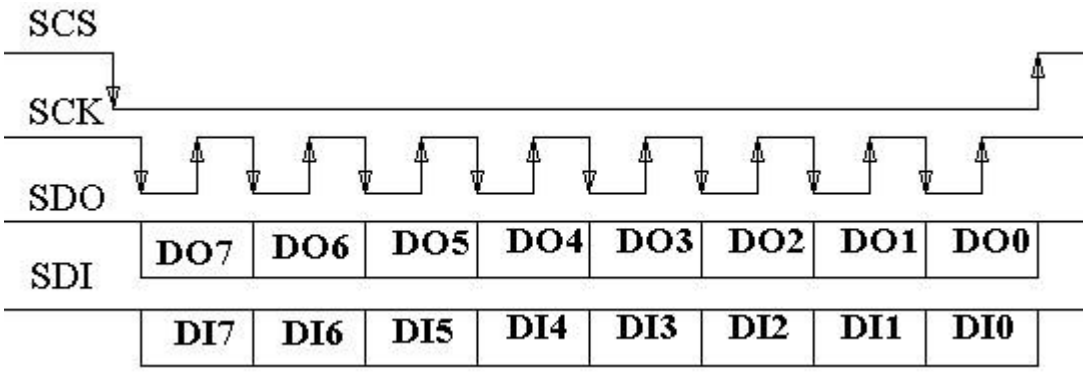
串行接口（Serial Interface）是最简单的一种数据通信方式。串行通信最少只需要一根传输线即可完成，传输距离长，成本低，但是速度较慢。

串口可以分为同步串口 SPI（Serial Peripheral interface）和异步串口 UART（Universal Asynchronous Receiver/Transmitter）两种。一般情况下，两者均使用两根数据线，一根负责数据发送，另一根负责数据接收，来实现全双工模式。不同

的是，同步串口需要一个同步时钟来控制所有的串口设备，异步串口则需要各个串口设备自行产生时钟，并相互约定串口的比特率。

6.3.1 同步串口

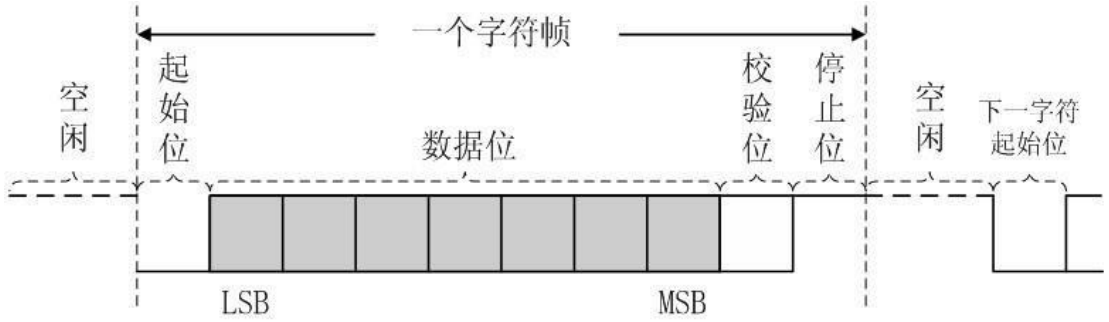
同步串口一般包括片选信号线，时钟信号线，数据发送线，数据接收线这 4 根线。



同步串口严格区分主设备和从设备，时钟信号完全由主设备控制，因此主设备可以随时控制整个数据传输的速率。

6.3.2 异步串口

异步串口一般包括数据发送线，数据接收线这 2 根线。



异步串口不区分主设备和从设备，通信双方处于完全相同的地位，需要事先约定好数据传输的波特率，数据位数，校验位类型，停止位长度等参数才可以开始数据传输。最常用的波特率是 9600 和 115200。

由于异步数据传输过程中没有时钟信号进行相位同步，串口控制器在处理数据接收时，应对同一个 bit 进行多次采样，将出现次数较高的电平值作为此 bit 的数据值。

异步串口的停止位可能被设定为 1.5 位。如果设备无法产生半个位长的数据，可以在发送时使用 2 位停止位而在接收时使用 1 位停止位。

在 Spartan-6 上编写 PS2 模块时发现，当串口线未连接时，FPGA 刚刚编程完成后可能会在引脚上收到噪声信号，因此建议在确定不使用串口时关闭其使能。具体实现详见 uart.v

第 7 章 系统软件

系统软件包括控制台与文件系统两部分。控制台负责与用户的交互，并根据用户的命令操作文件系统将结果返回给用户。文件系统部分则实现了一个 FAT32 的文件系统，可以执行 dir, ren, touch, del, type 等操作。

7.1 控制台

控制台实则为一个死循环，它不停地从键盘获得输入，并对其进行解析以及执行。主要由以下函数组成

```
void ReadLine(char* line);
void GetParameter(char* command, int* argc, char* argv[]);
void Execute(unsigned int argc, char* argv[]);
```

控制台通过 *ReadLine* 函数从键盘获得输入，当获得回车键时，表示一行结束，并返回所获得的字符串到参数 line 中。

控制台通过 *GetParameter()*函数从 *command* 中获得参数，存入 *argv[]* 中，并统计其个数，以便系统执行。*command* 参数即为 *ReadLine()*函数所获得的输入，其中参数用空格分隔。

控制台通过 *Execute()*函数执行命令，控制台将从 GetParameter()函数中获得的参数传入 *Execute()*，然后 Execute()对其进行解释执行。*Execute()*可以执行的指令有 *dir*, *type*, *ren*, *del*, *touch*，*exec*, *exit*, *lou*, *lougb*。当命令错误或者参数格式错误时，输出 *error* 并返回。

7.2 文件系统

文件系统实现了一个微型 FAT32 文件系统，它支持 dir, type, ren, del, touch 等指令。其磁盘内容存储在笔记本上，CPU 通过串口与笔记本通信，并对该磁盘进行操作。文件系统的基础函数有如下两个：

```
void SectionRead(unsigned int section_number)
```

`void SectionWrite(unsigned int section_number)`

其中，SectionRead()函数读取磁盘指定扇区，并将其加载到文件缓冲区中，SectionWrite()函数将文件缓冲区的内容写入到指定扇区中。我们通过这两个函数实现了对 FAT32 文件系统的解析与读写。

在上面两个函数的基础上，文件系统提供了如下函数以实现文件系统的功能。

`void OpenFIle(char* file_name, unsigned int flag, unsigned int mode)`

`void ReadFile(char* buffer, unsigned int length)`

`void Dir()`

`void Type(char* argv[])`

`void Rename(char* argv[])`

`void Touch(char* argv[])`

`void Del(char* argv[])`

其中，OpenFile()函数打开文件，并将其第一个扇区的内容放在文件缓冲区中，若文件不存在，则输出 error 并返回。

ReadFile()函数将文件缓冲区中指定长度的内容读入到参数 buffer 中。文件缓冲区维护了一个读写头，当该读写头读到文件末尾时，则直接返回，忽略掉多余指定的长度。若读写头读完了缓冲区，则将缓冲区的内容写回，重新读取下一块文件内容。

Dir()函数将磁盘现有的文件列出来。该函数遍历 FAT32 文件系统的目录项，每读到一个文件，刚输出其文件名。

Type()函数输出文件内容。该函数首先查找目标文件，若文件不存在，则输出 error 并返回，若存在，则遍历其内容，并将内容作为文本输出。

Rename()函数重命名文件。将源文件名修改为目标文件名，若源文件不存在，则输出 error 并返回。

Touch()函数创建空文件。若指定文件名长度太长，则文件系统只取其前 8 个字节的内容，同样，若扩展名太长，则文件系统只取其前 3 个字节的内容。

Del()函数删除文件。该函数首先查找目标文件，若文件不存在，则输出 error 并返回，若存在，则删除其 FAT 表的数据，文件内容数据不作删除。

Appendix

表格 1 重要硬件模块列表(I/O 地址表、内存分配表)

主存	0x0000_0000 - 0x0000_ffff
显存	0x000c_0000 - 0x000c_ffff
串口	0xfe00_0000
PS2 键盘	0xffff_d000
板载 IO(MIO_BUS)	0xffff_f000

表格 2 系统功能调用表(Syscall)

Service	SystemCallCode($v0)	Arguments	Results
sys_PrintInt	1	$a0=int	--
sys_PrintString	4	$a0=string	--
sys_PrintChar	11	$a0=char	--
sys_ReadChar	12	--	$a0=char
sys_Read	14	$a0=blockNum	--
sys_Write	15	$a0=blockNum	--

表格 3 中断响应程序

ISR	Function
Timer	Raise a timer flag
Ps2	read char from ps2 IO address, move it to keyboard buffer in memory
Uart	read block(512Bytes) from uart IO address, move it to file buffer in memory

表格 4 应用程序表

Application	Function
Lou	Print “楼 sir， 你好 魏世嘉， 陈俊， 刘俊灏， 孙同”
Lougb	Print “楼 sir， goodbye”
Strcmp	Compare if two strings are equal
Filesystems operations	dir, type, touch, ren, del, exec, etc.