第3章 I/O端口地址译码技术

设备选择功能是接口电路应具备的基本功能之一，因此，进行设备端口选择的I/O端口地址译码电路是每个接口电路中不可缺少的部分。本章在介绍I/O地址空间、I/O端口基本概念和I/O端口译码基本原理与方法的基础上，着重讨论I/O端口地址译码电路的设计，其中包括采用GAL(PAL)器件的I/O端口地址译码电路设计。

3.1 I/O地址空间

如果忽略I/O地址空间的物理特征，仅从软件编程的角度来看，I/O地址空间和内存地址空间一样，也是一片连续的地址单元，专供各种外设与CPU交换信息时存放数据、状态和命令代码。实际上，I/O地址空间中的一个地址单元对应接口电路中的一个寄存器或控制器，所以把它们称为接口中的端口。

I/O地址空间的地址单元可以被任何外设使用，但是，一个I/O地址一经分配给某个外设(通过I/O端口地址译码进行分配)，那么，这个地址就成了该外设固有的端口地址，系统中别的外设就不能同时使用这个端口地址，否则就会发生地址冲突。

I/O端口地址与内存的存储单元一样，都是以数据字节来组织的，因此有8位I/O端口、16位I/O端口、32位I/O端口等。一般情况下，单片机多使用8位或16位I/O端口，而PC和高档嵌入式微机使用32位I/O端口。

3.2 I/O端口

3.2.1 什么是端口

端口(port)是接口(interface)电路中能被CPU访问的寄存器的地址。微机系统给接口电路中的每个寄存器分配一个端口，因此，CPU在访问这些寄存器时，只需指明它们的端口，而不必指明寄存器。这样，在输入输出程序中就只看到端口，而看不到相应的具体寄存器。也就是说，访问端口就是访问接口电路中的寄存器。可见，端口是为了编程从抽象的逻辑概念来定义的，而寄存器是从物理含义来定义的。

CPU通过端口向接口电路中的寄存器发送命令、读取状态和传输数据，因此，一个接口电路中可以有几种不同类型的端口，如命令(端)口、状态(端)口和数据(端)口。并且，CPU的命令只能写到命令口，外设(或接口)的状态只能从状态口读取，数据只能写到数据口或从数据口读出。3种信息与3种端口类型一一对应，不能错位。否则，接口电路就不能正常工作，就会产生误操作。

3.2.2 端口的共用技术

一般情况下，一个端口只用于一种信息（命令、状态或数据）的访问；但有些接口芯片允许同一端口既作为命令口又作为状态口，或允许向同一个命令口写入多个命令字，这就产生端口共用的问题。对这种情况如何处理？

如果命令口和状态口共用一个端口，其处理方法是根据读写操作来区分。向该端口写，就是写命令，该端口作命令口用；从该端口读，就是读状态，该端口作状态口用。当多个命令字写到同一个命令口时，可采用两种办法解决：其一，在命令字中设置特征位（或设置专门的访问位），根据特征位，就可以识别不同的命令，加以执行，例如，在 UART 内部的除数寄存器就采用这种办法；其二，在编写初始化程序段时，按先后顺序向同一个端口写入不同的命令字，命令寄存器就根据这种先后顺序的约定来识别不同的命令。另外，还可以用前面两种方法相结合的手段来解决端口的共用问题，如中断控制器 82C59A。

3.2.3 I/O 端口编址方式

CPU 要访问 I/O 端口，就需要知道端口的编址方式，因为针对不同的编址方式，CPU 会采用不同的指令进行访问。端口有两种编址方式：一种是 I/O 端口与内存地址单元分别独立编址；另一种是 I/O 端口和内存地址单元统一编址。

1. 独立编址方式

独立编址方式是接口中的端口地址单独编址而不和内存地址空间合在一起。大型计算机通常采用这种方式，有些微机，如 Intel 系列微机，也采用这种方式。

独立编址方式的优点是：I/O 端口地址不占用内存地址空间；使用专门的 I/O 指令对端口进行操作，I/O 指令短，执行速度快；对 I/O 端口寻址不需要全地址线译码，地址线少，也就简化了地址译码电路；由于 I/O 端口访问的专门 I/O 指令与内存访问指令有明显的区别，使程序中的 I/O 操作与其他操作界线清楚、层次分明，程序的可读性好；因为 I/O 端口地址和内存地址是分开的，故 I/O 端口地址和内存地址可以重叠，而不会相互混淆。

独立编址方式的缺点是：I/O 指令类型少，只使用 IN 和 OUT 指令，对 I/O 的处理能力不如统一编址方式；由于单独设置 I/O 指令，故需要增加$\overline{\text{IOR}}$和$\overline{\text{IOW}}$的控制信号引脚，这对 CPU 芯片来说是一种负担。

2. 统一编址方式

统一编址方式是从内存地址空间中划出一部分地址空间给 I/O 设备使用，把 I/O 接口中的端口当作内存单元一样进行访问。微控制器、嵌入式微机（例如 MIPS 微机）和单片机采用统一编址方式。

统一编址方式的优点是：由于对 I/O 设备的访问是使用访问内存的指令，I/O 处理能力增强；统一编址可给 I/O 端口带来较大的寻址空间，对大型控制系统和数据通信系统是很有意义的。嵌入式微机系统广泛采用这种方式。

统一编址方式的缺点是：I/O 端口占用了内存的地址空间，使内存容量减小；指令长度比专门的 I/O 指令长，因而执行时间长；地址线多，地址译码电路复杂。

3.2.4 I/O 端口访问

1. 独立编址方式的 I/O 端口访问

在对采用独立编址方式的 I/O 端口进行访问时，需要使用专门的 I/O 指令，并且需要采用 I/O 地址空间的寻址方式进行编程。下面以 Intel x86 微处理器为例讨论 I/O 指令及其寻址方式。

1) I/O 指令

访问 I/O 地址空间的 I/O 指令有两类：累加器 I/O 指令和串 I/O 指令。这里只介绍累加器 I/O 指令。

累加器 I/O 指令 IN 和 OUT 用于在 I/O 端口和 AL、AX、EAX 之间交换数据。其中，8 位端口对应 AL，16 位端口对应 AX，32 位端口对应 EAX。

IN 指令从 8 位(或 16 位、32 位)I/O 端口输入一字节(或一字、一双字)到 AL(或 AX、EAX)。OUT 指令刚好与 IN 指令相反，从 AL(或 AX、EAX)中输出一字节(或一字、一双字)到 8 位(或 16 位、32 位)I/O 端口。例如：

```
IN   AL,0F4H              ;从端口 0F4H 输入 8 位数据到 AL
IN   AX,0F4H              ;将端口 0F4H 和 0F5H 的 16 位数据送 AX
IN   EAX,0F4H             ;将端口 0F4H、0F5H、0F6H 和 0F7H 的 32 位数据送 EAX
IN   EAX,DX               ;从 DX 指出的端口输入 32 位数据到 EAX
OUT  DX,EAX               ;将 EAX 的内容输出到 DX 指出的 32 位数据端口
```

通常所说的 CPU 从端口读数据或向端口写数据，仅仅是指 I/O 端口与 CPU 的累加器之间的数据传输，并未涉及数据是否传输到内存的问题。

若要求将端口的数据传输到内存，在输入时，则除了使用 IN 指令把数据读入累加器之外，还要用 MOV 指令将累加器中的数据再传输到内存。例如：

```
MOV  DX,300H              ;I/O 端口
IN   AL,DX                ;从 I/O 端口读数据到 AL
MOV  [DI],AL              ;将数据从 AL 传输到内存
```

在输出时，数据用 MOV 指令从内存先送到累加器，再用 OUT 指令从累加器传输到 I/O 端口。例如：

```
MOV  DX,301H              ;I/O 端口
MOV  AL,[SI]              ;从内存取数据到 AL
OUT  DX,AL                ;数据从 AL 传输到 I/O 端口
```

2) I/O 端口寻址方式

I/O 端口寻址有直接 I/O 端口寻址和间接 I/O 端口寻址两种方式，其差别表现在 I/O 端口地址是否经过 DX 寄存器传输。若 I/O 端口地址不经过 DX 传输，直接写在指令中，作为指令的一个组成部分，称为直接 I/O 寻址；若 I/O 端口地址经过 DX 传输，称为间接 I/O 寻址。例如，在输入时：

```
IN   AX,0E0H              ;直接寻址，端口号 0E0H 在指令中直接给出
```

```
MOV  DX,300H
IN   AX,DX                    ;间接寻址,端口号 300H 在 DX 中间接给出
```

在输出时：

```
OUT  0E0H,AX                  ;直接寻址
MOV  DX,300H
OUT  DX,AX                    ;间接寻址
```

3）I/O 指令与 I/O 读写控制信号的关系

I/O 指令与 I/O 读写控制信号是完成 I/O 操作这一共同任务的软件(逻辑上的)和硬件(物理上的),是相互依存、缺一不可的两个方面。$\overline{IOR}$和$\overline{IOW}$是 CPU 对 I/O 设备进行读写的硬件上的控制信号,低电平有效。该信号为低,表示对外设进行读写;该信号为高,则不读写。但是,这两个控制信号并不能激活自己,使自己变为有效,以控制读写操作;而必须通过编程,在程序中执行 IN/OUT 指令激活$\overline{IOR}$和$\overline{IOW}$,使之变为有效(低电平),对外设进行读写操作。在程序中,执行 IN 指令使$\overline{IOR}$信号有效,完成读(输入)操作;执行 OUT 指令使$\overline{IOW}$信号有效,完成写(输出)操作。在这里,I/O 指令与读写控制信号$\overline{IOR}$和$\overline{IOW}$的软件与硬件对应关系表现得十分明显。

2. 内存映射的 I/O 端口访问

以 MIPS 微处理器为例,看看两个内存映射 I/O 端口访问的表现形式。代码如下：

```
//把 0x543 写到 LED
addiu  $7,$0,0x543            ;$7 = 0x543
lui    $5,0xbf80              ;$5 = 0xbf800000(LED 地址)
sw     $7,0($5)               ;LED = 0x543
//把开关的值读到 10 号寄存器中
lui    $5,0xbf80              ;$5 = 0xbf800000
lw     $10,8($5)              ;$10 为开关的值
```

3.3 I/O 端口地址分配及选用的原则

I/O 端口地址是微机系统的重要资源,I/O 端口地址的分配对接口设计十分重要。因为要把新的 I/O 设备添加到系统中,就要在 I/O 地址空间给它分配确定的 I/O 端口地址。只有了解了哪些地址被系统占用,哪些地址已分配给了别的设备,哪些地址是计算机厂商申明保留的,哪些地址是空闲的,等等,才能做出合理的地址选择。不同的微机系统,端口地址的分配方案不同。下面以 Intel x86 微机为例介绍 I/O 地址分配情况。

3.3.1 早期微机 I/O 地址的分配

早期微机只使用低 10 位地址线 $A_0 \sim A_9$,故其 I/O 端口地址范围是 0000H～03FFH。同时,I/O 地址空间分成系统级 I/O 接口芯片的端口地址和外设接口卡的端口地址两部分,分别如表 3.1 和表 3.2 所示。

表 3.1 系统级 I/O 接口芯片的端口地址

I/O 接口芯片	端 口 地 址
DMA 控制器 1	000H～01FH
DMA 控制器 2	0C0H～0DFH
DMA 页面寄存器	080H～09FH
中断控制器 1	020H～03FH
中断控制器 2	0A0H～0BFH
定时器	040H～05FH
并行接口芯片	060H～06FH
RT/CMOS RAM	070H～07FH
协处理器	0F8H～0FFH

表 3.2 外设接口卡的端口地址

外设接口卡	端 口 地 址
并行口控制卡 1	378H～37FH
并行口控制卡 2	278H～27FH
串行口控制卡 1	3F8H～3FFH
串行口控制卡 2	2F8H～2FFH
原型插件板(用户可用)	300H～31FH
同步通信卡 1	3A0H～3AFH
同步通信卡 2	380H～38FH
彩显 EGA/VGA	3C0H～3CFH
硬驱控制卡	320H～32FH

随着微机硬件技术的发展,淘汰了许多外设,新增了许多新型外设,但有一部分作为接口上层应用程序的 I/O 设备的地址仍被保留,以便保持逻辑上的兼容。另外,从接口技术分层次的观点来看,现代微机接口分为设备接口和总线接口两个层次,上述 I/O 地址的分配属于接口上层(即设备接口)的资源分配,是整体接口的一部分,因此,这些 I/O 地址对接口上层应用程序可照常使用。

由于早期微机系统没有即插即用的资源配置机制,因此,上述 I/O 端口地址的分配是固定的,不能按照系统资源的使用情况动态分配。

3.3.2 现代微机 I/O 地址的分配

现代微机的操作系统具有即插即用的资源配置机制,因此,作为系统重要资源的 I/O 端口地址的分配是动态变化的。操作系统根据现代微机系统资源的使用情况动态地重新分配用户应用程序要使用的 I/O 地址,即用户程序使用的 I/O 端口地址与操作系统分配的系统端口地址是不一致的,两者之间通过配置地址空间进行映射,即操作系统利用配置地址空间对用户程序要使用的 I/O 端口地址进行重新分配。用户在原创性开发时使用现代微机系统的地址(通过驱动程序使用),而在二次性开发时使用早期微机系统的地址,两种不同系统的地址之间要进行映射。

这种 I/O 地址映射(或者说 I/O 地址重新分配)的工作对用户是透明的,不影响用户对 I/O 端口地址的使用,在用户程序中仍然可以使用早期微机系统的 I/O 地址对端口进行访问。I/O 地址映射已在 2.6.2 节中作了介绍,也可见参考文献[24,25]。

3.3.3 I/O 端口地址选用的原则

用户在使用 PC 系统的 I/O 地址时,为了避免 I/O 端口地址发生冲突,应遵循如下原则:

(1) 凡是由系统配置的外设占用的地址一律不能使用。

(2) 未被占用的地址,用户可以使用,但计算机厂家申明保留的地址不要使用。

(3) 一般情况下,用户可使用 300H～31FH 的地址,这是早期微机留给原型插件板使用的,用户可以使用。

根据上述系统对 I/O 端口地址的分配情况和 I/O 端口地址的选用原则,在本书接口设计举例中使用的端口地址分为两部分:涉及系统配置的接口芯片和接口卡使用表 3.1 和表 3.2 中分配的 I/O 端口地址;用户扩展的接口芯片使用表 3.3 中分配的 I/O 端口地址。

表 3.3 用户扩展的接口芯片 I/O 端口地址

接口芯片	端口地址
82C55A	300H～303H
82C54A	304H～307H
8251A	308H～30BH
82C79A	30CH～30DH

3.4 I/O 端口地址译码

CPU 通过 I/O 地址译码电路把地址总线上的地址信号翻译成要访问的 I/O 端口,这就是 I/O 端口地址译码。

3.4.1 I/O 端口地址译码的方法

微机系统的 I/O 端口地址译码有全译码、部分译码和开关式译码 3 种方法,其中全译码很少使用。

1. 全译码

全译码是指将所有 I/O 地址线全部作为译码电路的输入参加译码。一般在要求产生单个 I/O 端口时采用,在 PC 中很少使用。

2. 部分译码

部分译码的具体做法是把 I/O 地址线分为两部分:一是高位地址线,参加译码,经译码电路产生 I/O 接口芯片的片选 CS 信号,实现接口芯片之间的寻址;二是低位地址线,不参加译码,直接连到接口芯片,进行接口芯片的片内端口寻址,即寄存器寻址。所以,低位地址线(又称接口电路中的寄存器寻址线)由接口芯片内部进行译码。

3. 开关式译码

开关式译码是指在部分译码方法的基础上,加上地址开关来改变 I/O 端口地址,一般在要求 I/O 端口地址需要改变时采用。地址开关不能直接接到 I/O 地址线上,而必须通过某种中介元件将地址开关的状态(ON/OFF)转移到 I/O 地址线上,能够实现这种中介转移作用的有比较器、异或门等。

3.4.2 I/O 端口地址译码电路的输入与输出信号线

在微机系统中,通过 I/O 端口地址译码电路把来自地址总线上的地址信号翻译成要访问的 I/O 端口,因此 I/O 端口地址译码电路的工作原理实际上就是它的输入与输出信号之间的关系。

1. I/O 端口地址译码电路的输入信号

I/O 端口地址译码电路的输入信号首先是地址信号，其次是控制信号。所以，在设计 I/O 端口地址译码电路时，其输入信号除了 I/O 地址线之外，还包括控制线。

2. I/O 端口地址译码电路的输出信号

I/O 端口地址译码电路的输出信号中只有 1 根$\overline{CS}$片选信号，且低电平有效。$\overline{CS}=0$ 时有效，芯片被选中；$\overline{CS}=1$ 时无效，芯片未被选中。

3.4.3 $\overline{CS}$的物理含义

$\overline{CS}$的物理含义是：如果$\overline{CS}$有效，选中一个接口芯片时，这个芯片内部的数据线打开，并与系统的数据总线接通，从而打通了接口电路与系统数据总线的通路；而其他芯片的$\overline{CS}$无效，即未被选中，于是芯片数据线呈高阻抗，自然就与系统的数据总线隔离开，从而关闭了接口电路与系统总线的通路。此时，虽然那些未被选中的芯片的数据线与系统数据总线在外部看起来是连在一起的，但因内部并未接通，呈断路状态，也就不能与 CPU 交换信息。每一个外设接口芯片都需要一个$\overline{CS}$信号去接通/断开其数据线与系统数据总线，从这个意义来讲，$\overline{CS}$是一个起开/关作用的控制信号。

3.5 设计 I/O 端口地址译码电路时应注意的问题

设计 I/O 端口地址译码电路时应注意以下几个问题。

1. 合理选用 I/O 端口地址范围

根据系统对 I/O 地址的分配情况和用户对 I/O 端口地址选用的原则，合理选用 I/O 端口地址范围，即选用用户可用的地址段或未被占用的地址段，以避免地址冲突。

2. 正确选用 I/O 端口地址译码方法

根据用户对 I/O 端口地址的设计要求，正确选用译码方法。一般情况下，单个端口地址译码采用全译码法，多个端口地址译码采用部分译码法。

3. 灵活选用 I/O 端口地址译码电路

I/O 端口地址译码电路设计的灵活性很大，产生同样的 I/O 端口地址的译码电路不是唯一的，可以有多种选择。首先，电路可以采用不同的元器件组成；其次，参加译码的地址信号和控制信号之间的逻辑组合可以不同。因此，在设计 I/O 端口地址译码电路时，对元器件和参加译码的信号之间的逻辑组合可以不拘一格地进行恰当的选择，只要能满足 I/O 端口地址的要求就行。

3.6 I/O 端口地址译码电路举例

I/O 端口地址译码电路设计包括采用不同元器件(IC 电路、译码器、GAL 器件)、不同译码方法以及不同译码电路结构形式(固定式、开关式)的地址译码电路设计。

下面从不同侧面举几个 I/O 端口地址译码电路设计的例子。

例 3.1 固定式单端口地址译码电路的设计。

要求

设计 I/O 端口地址为 2F8H 的只读译码电路。

分析

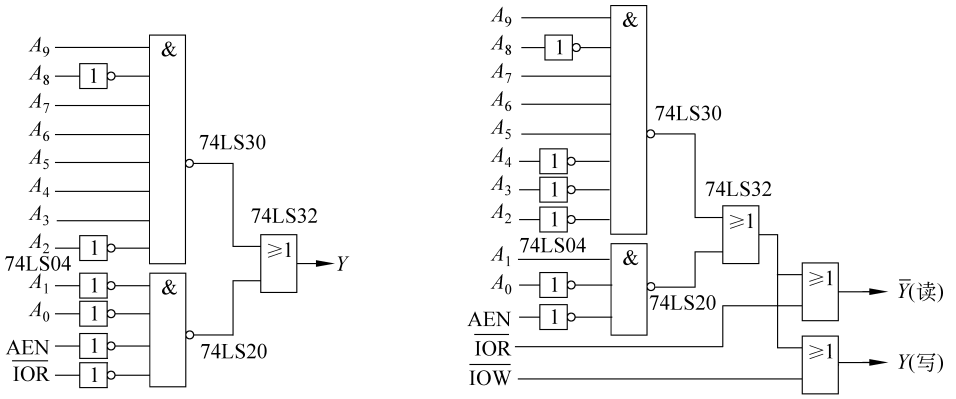
由于是单个端口地址的译码电路，不需要产生片选信号$\overline{CS}$，故采用全译码方法。地址线全部作为译码电路的输入线参加译码。为了满足端口地址是 2F8H 的要求，对于只使用 10 位 I/O 地址线的情况，输入地址线的值必须如表 3.4 所示。

表 3.4 固定式单端口地址 2F8H 输入地址线的值

地址线	A_9 A_8	A_7 A_6 A_5 A_4	A_3 A_2 A_1 A_0
二进制值	1 0	1 1 1 1	1 0 0 0
十六进制值	2	F	8

设计

能够实现上述地址线取值的译码电路有很多种，一般采用 IC 门电路就可以实现，而且很方便。本例采用门电路实现地址译码，译码电路如图 3.1 所示。



（a）只读端口　　（b）可读写端口

图 3.1 固定式单端口地址译码电路

图 3.1 中参加译码的除地址线 $A_0 \sim A_9$ 之外，还有控制线$\overline{IOR}$（读端口）、$\overline{IOW}$（写端口）等，以控制端口可读或可写。例如，图 3.1(a)中，只有$\overline{IOR}$参加译码，故该端口只能读，不能写；而图 3.1(b)中，$\overline{IOR}$和$\overline{IOW}$都参加译码，故该端口既可读又可写。

例 3.2 多个端口的 I/O 地址译码电路的设计。

要求

使用 74LS138 设计一个系统板上的 I/O 端口地址译码电路，要求可选 8 个接口芯片并且让每个接口芯片内部的端口数目为 32 个。

分析

为了让每个被选中的芯片内部拥有 32 个端口，留出 5 根低位地址线不参加译码，其余

的高位地址线作为74LS138的输入线参加译码，或作为74LS138的控制线控制74LS138的译码操作，由此可以得到多个端口的I/O地址译码电路输入地址线的值，如表3.5所示。

表3.5 多个端口的I/O地址译码电路输入地址线的值

地 址 线	$A_9\ A_8$	$A_7\ A_6\ A_5$	$A_4\ A_3\ A_2\ A_1\ A_0$
用途	控制	片选	片内端口寻址
十六进制值	0H	0H～7H	00H～1FH

对于译码器74LS138的分析有两点很重要。一是它的控制信号线G_1、$\overline{G}_{2A}$和$\overline{G}_{2B}$，只有当满足控制信号线$G_1=1$，$\overline{G}_{2A}=\overline{G}_{2B}=0$时，74LS138才能进行译码；二是译码的逻辑关系，即输入(C、B、A)与输出($\overline{Y}_0$～$\overline{Y}_7$)的对应关系，74LS138输入与输出的逻辑关系(即真值表)如表3.6所示。

表3.6 74LS138的真值表

输入						输出							
G_1	$\overline{G}_{2A}$	$\overline{G}_{2B}$	C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
0	×	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	×	1	1	1	1	1	1	1	1
×	×	1	×	×	×	1	1	1	1	1	1	1	1

从表3.6可知，若满足G_1接高电平、$\overline{G}_{2A}$和$\overline{G}_{2B}$接低电平的控制条件，则由输入端C、B、A来决定输出：当$CBA=000$时，输出端$\overline{Y}_0=0$，其他输出端为高电平；当$CBA=001$时，输出端$\overline{Y}_1=0$，其他输出端为高电平……当$CBA=111$时，则输出端$\overline{Y}_7=0$，其他输出端为高电平。由此可分别产生8个译码输出信号(低电平)，可选用8个接口芯片。若控制条件不满足，则输出全为1，不产生译码输出信号，即译码无效。

设计

采用74LS138译码器设计的I/O端口地址译码电路如图3.2所示。

图3.2中的地址线的高5位参加译码，其中A_5～A_7经3-8译码器分别产生$\overline{\text{DMACS}}$(82C37A)、$\overline{\text{INTRCS}}$(82C59A)、$\overline{\text{T/C}}$ $\overline{\text{CS}}$(82C54A)、$\overline{\text{PPICS}}$(82C55A)的片选信号，而地址线

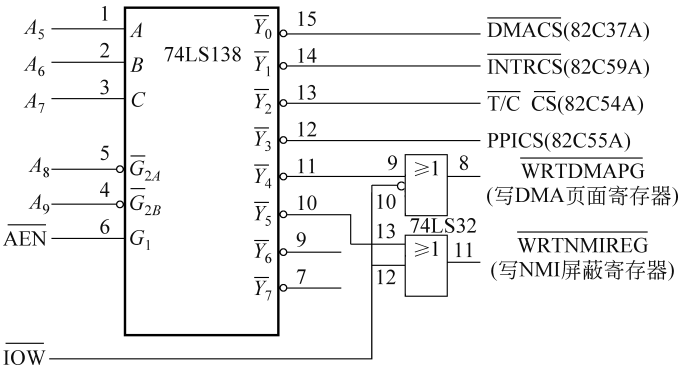


图 3.2　采用 74LS138 译码器设计的 I/O 端口地址译码电路

的低 5 位 A_0～A_4 作为芯片内部寄存器的访问地址线。从表 3.6 给出的 74LS138 译码器的真值表可知，82C37A 的端口地址范围是 000～01FH，82C59A 的端口地址范围是 020～03FH，正好与表 3.1 中所列出的系统级 I/O 接口芯片的端口地址一致。

另外，图 3.2 中的 A_9、A_8 和 $\overline{\text{AEN}}$ 分别连接到 $\overline{G}_{2A}$、$\overline{G}_{2B}$ 和 G_1 端，作为 74LS138 的控制信号，$\overline{\text{IOW}}$ 信号用来控制 DMA 页面寄存器与不可屏蔽中断寄存器只能写不能读，而其他 4 个支持芯片既可写又可读。

例 3.3　开关式 I/O 端口地址译码电路的设计。

要求

设计某微机实验平台板的 I/O 端口地址译码电路，要求实验平台板上每个接口芯片的内部 I/O 端口数目为 4 个，并且 I/O 端口地址可选，其地址选择范围为 300H～31FH。

分析

开关式 I/O 端口地址译码电路可由译码器、DIP 开关、比较器或异或门这几种元器件组成。先分析 DIP 开关、比较器的工作原理，然后根据要求进行电路设计。

DIP 开关有两种状态，即通(ON)和断(OFF)。所以，要对这两种状态进行设置，可以设置 DIP 开关状态为 ON＝0、OFF＝1。

对于比较器有两点要考虑：一是比较的对象；二是比较的结果。本例采用 4 位比较器 74LS85，把它的 A 组 4 根线与地址线连接，B 组 4 根线与 DIP 开关相连，这样就把比较器 A 组与 B 组的比较转换成了地址线的值与 DIP 开关状态的比较。74LS85 比较的结果有 3 种：$A>B$，$A<B$，$A=B$。本例采用 $A=B$ 的结果，并令 $A=B$ 时 74LS85 输出高电平。这意味着，当 4 位地址线的值与 4 个 DIP 开关的状态相等时，74LS85 输出高电平；否则，74LS85 输出低电平。

将 74LS85 的 $A=B$ 输出线连到译码器 74LS138 的控制线 G_1 上，因此，只有当 4 位地址线(A_6～A_9)的值与 4 个 DIP 开关的状态(S_0～S_3)各位均相等时，才能使 74LS138 的控制线 $G_1=1$，译码器工作；否则，译码器不能工作。所以，如果改变 DIP 开关的状态，则迫使地址线的值发生改变，才能使两者相等，从而达到利用 DIP 开关改变地址的目的。

设计

根据上述分析可设计出实验平台板的 I/O 端口地址译码电路，如图 3.3 所示。

从图 3.3 中可看出，在高位地址线中，$A_9A_8A_7A_6$ 的值由 DIP 开关的状态 $S_3S_2S_1S_0$ 决

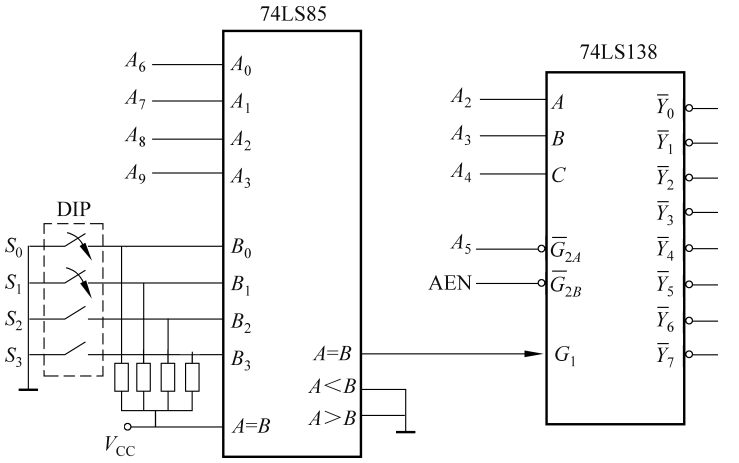


图 3.3　实验平台板的 I/O 端口地址译码电路

定，4 位开关有 16 种不同的组合，也就是可改变 16 种地址。按图 3.3 中开关的状态不难看出，由于 S_3 和 S_2 断开，S_1 和 S_0 合上，故使 $A_9=A_8=1$，$A_7=A_6=0$，而 A_5 连在 74LS138 的 $\overline{G}_{2A}$ 上，故 $A_5=0$。$A_4A_3A_2$ 这 3 根地址线作为 74LS138 的输入线，经译码后可产生 8 个低电平有效的选择信号 $\overline{Y}_0\sim\overline{Y}_7$，作为实验平台板上的接口芯片选择信号。最后剩下两根低位地址线 A_1 和 A_0 未参加译码，作为寄存器选择信号，以实现每个接口芯片内部拥有 4 个 I/O 端口的设计要求。以上设计完全满足 300H～31FH 端口地址范围和每个接口芯片内部具有 4 个 I/O 端口的设计要求，正好与表 3.3 中所列出的 I/O 端口地址一致。

例 3.4　可编程端口地址译码电路设计。

可编程端口地址译码电路采用 FPGA 来实现。主要工作有两点：一是选择 GAL 芯片；二是编写 GAL 编程输入源文件中的逻辑表达式，其中输入与输出信号之间的逻辑表达式的编写是译码电路设计的关键。另外就是借助于编程工具生成 GAL 芯片熔丝状态分布图及编程代码文件，并烧写到 GAL 芯片内部。

要求

利用 GAL 芯片设计 MFID 多功能微机接口实验平台的 I/O 端口地址译码电路，其地址范围为 300H～31FH，包括 8 个接口芯片，每个接口芯片内部拥有 4 个 I/O 端口，每个 I/O 端口可读可写。

分析

首先分析 GAL 芯片的输入线。

根据设计要求，参加译码的有地址线和控制线，从地址范围 300H～31FH 可知，10 根地址线取值如表 3.7 所示。其中，Ix 代表可变输入位，为 0 或 1。

表 3.7　GAL 芯片的地址线取值

地址线	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1　A_0
取值	1	1	0	0	0	I_X	I_X	I_X	

在表 3.7 中，10 位地址线的设置是：高 5 位地址为 $A_9=A_8=1$，$A_7=A_6=A_5=0$，固定不变，保证起始地址 300H；中间 3 位 $A_4\sim A_2$（$I_XI_XI_X$）由 GAL 芯片内部译码，产生 8 个片选信号；最低两位 A_1A_0 不参加译码，由接口芯片内部产生 4 个端口。为了减少送到 GAL 芯片的输入线数目，将参加译码的 8 根地址线做了一些处理，$A_9\sim A_5$ 这 5 根地址线经过与非门之后，其输出线 YM 接到 GAL 芯片，因此，实际上送到 GAL 芯片参加译码的只有 4 根地址线。

控制线有 3 根，除 AEN 外，还有$\overline{IOR}$和$\overline{IOW}$也参加译码，以满足译码产生的 I/O 端口可读可写的设计要求。所以，GAL 芯片的输入线有 4 根地址线和 3 根控制线，共 7 根。

其次分析 GAL 芯片的输出线。

根据设计要求，需要 8 个片选信号 $\overline{Y}_0\sim\overline{Y}_7$，所以，GAL 芯片的输出线有 8 根。

由于需要的输入线、输出线都在 8 根线以内，故选择 GAL16V8 正合适，它有 8 个输入端（2～9）和 8 个输出端（12～19）。

设计

首先进行硬件设计。

根据上述分析，采用 GAL16V8 设计的 MFID 多功能微机接口实验平台的 I/O 端口地址译码电路如图 3.4 所示。

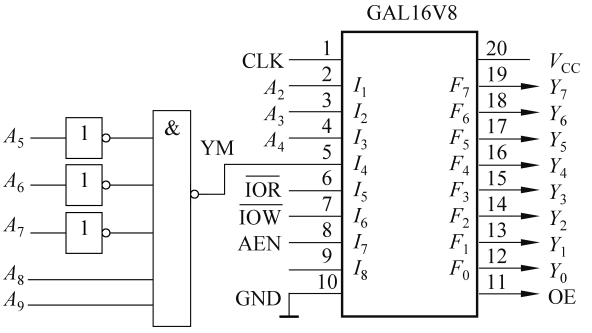


图 3.4　采用 GAL16V8 设计的 I/O 端口地址译码电路

其次进行软件设计。

使用 GAL 芯片进行译码电路设计。与以往采用 SSI、MSI 等 IC 器件的情况不同，本例除了进行硬件设计外，还要根据要实现的逻辑功能和编程工具要求的格式编写 GAL 芯片的编程输入源文件。该文件把逻辑变量之间的函数关系（输入与输出的关系）变换为阵列结构的与或关系（和积式）。再借助于编程工具生成 GAL 芯片熔丝状态分布图及编程代码文件，最后将编程代码烧写到 GAL 芯片内部。

下面是 GAL 芯片的编程输入源文件中产生 8 个输出信号（$\overline{Y}_0\sim\overline{Y}_7$）的逻辑表达式[①]。

$\overline{Y}_0$：A9 * A8 * /A7 * /A6 * /A5 * /A4 * /A3 * /A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * /A4 * /A3 * /A2 * /AEN * /IOW

$\overline{Y}_1$：A9 * A8 * /A7 * /A6 * /A5 * /A4 * /A3 * A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /

① 按照 GAL 芯片的编程输入源文件的格式要求，逻辑表达式中的逻辑符号“非”使用斜线，而不使用上画线。

A6 * /A5 * /A4 * /A3 * A2 * /AEN * /IOW

$\overline{Y}_2$：A9 * A8 * /A7 * /A6 * /A5 * /A4 * A3 * /A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * /A4 * A3 * /A2 * /AEN * /IOW

$\overline{Y}_3$：A9 * A8 * /A7 * /A6 * /A5 * /A4 * A3 * A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * /A4 * A3 * A2 * /AEN * /IOW

$\overline{Y}_4$：A9 * A8 * /A7 * /A6 * /A5 * A4 * /A3 * /A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * A4 * /A3 * /A2 * /AEN * /IOW

$\overline{Y}_5$：A9 * A8 * /A7 * /A6 * /A5 * A4 * /A3 * A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * A4 * /A3 * A2 * /AEN * /IOW

$\overline{Y}_6$：A9 * A8 * /A7 * /A6 * /A5 * A4 * A3 * /A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * A4 * A3 * /A2 * /AEN * /IOW

$\overline{Y}_7$：A9 * A8 * /A7 * /A6 * /A5 * A4 * A3 * A2 * /AEN * /IOR＋ A9 * A8 * /A7 * /A6 * /A5 * A4 * A3 * A2 * /AEN * /IOW

每个逻辑表达式都是两个与式之和，而前后两个与式的不同在于读项和写项的差别，前者是读有效($\overline{\text{IOR}}$=0)，后者是写有效($\overline{\text{IOW}}$=0)，这表示该 I/O 端口既可读又可写。各逻辑表达式前面的 Y 项是逻辑表达式的输出值，即 GAL 芯片译码输出的片选信号。

由 $\overline{Y}_0$～$\overline{Y}_7$ 产生 8 个接口芯片的片选信号，再加上不参加译码的最低两位 00～11 的变化可得

$$\overline{Y}_0 = 300\text{H} \sim 303\text{H}$$
$$\overline{Y}_1 = 304\text{H} \sim 307\text{H}$$
$$\overline{Y}_2 = 308\text{H} \sim 30\text{BH}$$
$$\overline{Y}_3 = 30\text{CH} \sim 30\text{FH}$$
$$\vdots$$
$$\overline{Y}_7 = 31\text{CH} \sim 31\text{FH}$$

习题 3

1. PC 系统的 I/O 地址空间有多少字节？
2. 什么是端口？
3. 什么是端口共用技术？采用哪些方法来识别共用的端口地址？
4. I/O 端口的编址方式有几种？各有何特点？
5. 输入输出指令 IN/OUT 与 I/O 读写控制信号$\overline{\text{IOR}}$/$\overline{\text{IOW}}$是什么关系？
6. 设计 I/O 设备接口卡时，为防止地址冲突，选用 I/O 端口地址的原则是什么？
7. I/O 端口地址译码电路在接口电路中起什么作用？
8. 微机系统的 I/O 端口地址译码有哪几种方法？各用在什么场合？
9. 设计 I/O 端口地址译码电路应注意什么问题？
10. 在 I/O 端口地址译码电路中常常设置 AEN=0，这有何意义？

11. 说明 I/O 端口地址译码电路的输出信号$\overline{CS}$的物理含义。

12. I/O 地址线的高位地址线和低位地址线在 I/O 端口地址译码时分别有什么用途？

13. 可选式 I/O 端口地址译码电路一般由哪几部分组成？

14. 采用 GAL 芯片进行 I/O 端口地址译码电路设计的关键是什么？

15. 若要求 I/O 端口读写地址为 374H，则图 3.1(b)中的输入地址线要做哪些改动？

16. 图 3.2 是 PC 系统板的 I/O 端口地址译码电路，它有何特点？根据图 3.2 中地址线的分配，分别写出 DMA 控制器(DMAC)、中断控制器(INTR)、定时/计数器(T/C)以及并行接口(PPI)等芯片的地址范围。

17. 在图 3.3 所示的 I/O 端口地址译码电路中，若将地址开关的状态改为 S_3 和 S_0 断开，S_2 和 S_1 接通，则此时译码电路译出的地址范围是多少？

18. 要求在例 3.4 中将 I/O 地址范围为从 300H～31FH 改为 340H～34FH，其他不变，此时 GAL 芯片的编程输入源文件中的逻辑表达式应如何改写(可参考例 3.4)？

第 4 章　基于 MIPSfpga 的微处理器系统

4.1　MIPSfpga 处理器

4.1.1　概述

MIPS 是很流行的一种 RISC 处理器。MIPS 的意思是无内部互锁流水级的微处理器(Microprocessor without Interlocked Piped Stages),其机制是尽量利用软件的方法避免流水线中与数据相关的问题。它是在 20 世纪 80 年代初期由美国斯坦福大学 John Hennessy 教授领导的研究小组研制出来的,随后成立 MIPS 计算机公司,完成了 MIPS 处理器的商业化。

MIPS 计算机公司成立于 1984 年,1986 年推出 R2000 处理器,1988 年推出 R3000 处理器,1991 年推出第一款 64 位商用微处理器 R4000。1992 年,SGI 公司收购了 MIPS 计算机公司。1998 年,MIPS 计算机公司脱离 SGI 公司,成为 MIPS 技术公司。随后,MIPS 技术公司的战略发生变化,把重点放在嵌入式系统上。1999 年,MIPS 技术公司发布 MIPS32 和 MIPS64 架构标准,为未来 MIPS 处理器的开发奠定了基础。MIPS 技术公司于 2013 年被 Imagination Technologies 公司收购。

MIPS 处理器在 20 世纪 80 年代和 90 年代是 SGI 高性能图形工作站的核心。MIPS R3000 拥有 5 级流水线,是 MIPS 公司第一个在商业上取得成功的处理器;随后是 R4000,该处理器新增了 64 位指令;R8000 则是超标量处理器;R10000 进而实现了乱序执行流水线。它们均是高性能处理器的代表。

MIPS 处理器架构在高性能处理器领域取得成功的同时,也开始进军消费类产品市场,将其拓展为低功耗、低成本的处理器,应用领域包括消费电子、网络和微控制器。其中 M4K 系列是基于经典的 5 级流水结构的处理器;M14K 则在 M4K 的基础上通过增加 16 位的 microMIPS 指令集来减少应用程序的代码,以便更好地应用于成本敏感的嵌入式系统;microAptiv 则对 M14K 进行了进一步延伸,添加了可选的数字信号处理指令。microAptiv 其实是微控制器(MicroController,μC)和微处理器(MicroProcessor,μP)的变种,通过在微处理器中增加高速缓存和虚拟内存来支持操作系统,以便运行 Linux 或 Android 等操作系统。市场上广泛应用的美国微芯科技(Microchip Technology)公司的 PIC32 微控制器系列就是基于 M4K 架构开发的。

MIPS M4K、M14K 和 microAptiv 是 Imagination Technologies 公司以微体系结构方式(MicroArchitecture,MA)提供的最简单的处理器核。但是,它们与 Imagination Technologies 公司的中档处理器(interAptiv)和高端处理器(proAptiv)系列以及以这些处理器为核心构成的多核处理器在软件方面是兼容的。中档的 MIPS interAptiv 系列定位于 Cortex-A5/A7/A9 的竞争产品,是 32 位处理器核,流水线深度为 9 级,不支持乱序执行;但是,它在硬件上支持多线程以及双发射超标量的 64 位 MIPS I6400 体系结构。高端的 MIPS proAptiv 系列包括 1～6 个处

理器核。每个 proAptiv 核都是一颗超标量并支持乱序执行的处理器，拥有 32 位 MIPS P5600 架构和可扩展 SIMD 等高级功能。

MIPSfpga 是 Imagination Technologies 公司推出的应用于教学目的的处理器，它是在 MIPS32 microAptiv 微处理器架构上通过增加高速缓存和 MMU 而来的，以 Verilog 硬件描述语言源代码方式提供，因此可以在 FPGA 开发板上进行仿真和执行。

4.1.2 MIPSfpga 处理器核

MIPSfpga 处理器核基于 microAptiv 处理器。microAptiv 处理器广泛应用于工业、办公自动化、汽车、消费电子、无线通信等商业领域。MIPSfpga 处理器核用 Verilog 硬件描述语言描述，因此是一个处理器软核，而不是一个处理器芯片。用于描述 MIPSfpga 的 Verilog 程序代码量大约 1 万行。具体来说，MIPSfpga 处理器核有如下特点：

(1) 5 级流水线结构。

(2) 运行 MIPS32 ISA 指令集，性能为 1.5DMIPS/MHz。

(3) 4KB 的两路组相联指令高速缓存和数据高速缓存。

(4) 带 16 个 TLB 表项的 MMU。

(5) AHB-Lite 总线接口。

(6) EJTAG 编程/调试接口，支持两条指令和一个数据断点。

(7) 性能计数器。

(8) 输入信号同步。

(9) CorExtend 接口，用于用户自定义指令。

(10) 不包括数字信号处理(Digital Signal Processing，DSP)扩展、协处理器 2(Co-Processor2 CP2)接口和影子寄存器(Shadow Registers，SR)。

MIPSfpga 仅可用于教学而非商业用途。它可以用来学习微处理器是如何工作的，通过仿真或者在 FPGA 中来观察处理器的工作情况，通过阅读代码了解和学习处理器的微体系结构是如何实现的，用汇编语言或 C 语言编写程序来了解程序在 Verilog 仿真器或 FPGA 开发板上的运行过程。可以通过其总线来连接外部设备，学习接口技术。同样，还可以修改源代码来得到新的指令或者对其微体系结构进行扩展，也可以运行 Linux 操作系统来学习处理器系统是如何从编写 Verilog 设计代码直至在操作系统中运行的整个过程。

MIPSfpga 处理器的结构如图 4.1 所示。处理器的核心是执行单元(Execution Unit，EU)，它负责执行指令的操作，例如进行加法运算或减法运算。执行单元可扩展乘/除单元(Multiply/Divide Unit，MDU)，用于乘法和除法运算。指令译码器(Instruction Decoder，ID)对从指令高速缓存中读取的指令进行译码处理，产生相应的控制信号，对执行单元进行控制。系统协处理器(system co-processor)为协处理器提供系统时钟、复位等系统级的接口信号；通用寄存器(General Purpose Registers，GPR)用于存放指令的操作数。

图 4.1 顶部的其他接口分别是 UDI 接口、CP2 接口和中断接口(interrupt interface)，它们分别用于使处理器能够运行用户自定义指令、与协处理器 2(CP2)互连以及接收外部中断信号。

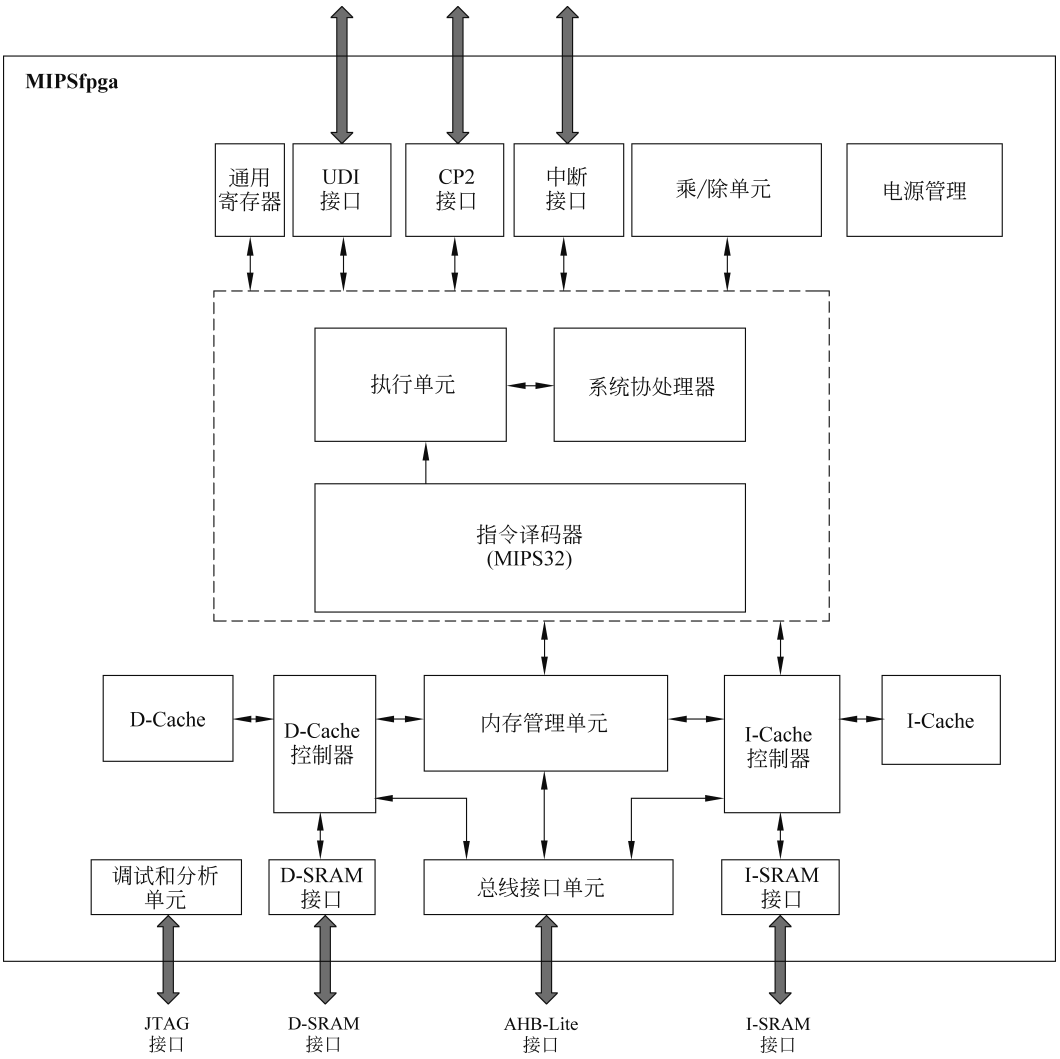


图 4.1　MIPSfpga 处理器核结构

指令高速缓存和数据高速缓存(I-Cache 和 D-Cache)分别通过各自的控制器连接到内存管理单元(Memory Management Unit,MMU)。MMU 负责内存的地址变换以及当指令或数据不在高速缓存中时将其从内存“搬运”到高速缓存中来。总线接口单元(Bus Interface Unit,BIU)使用户可以通过 AHB-Lite 总线协议给处理器外接存储器或者采用内存映射方式的外部设备(详见 4.2 节)。

数据和指令中间结果暂存寄存器(Scratchpad RAM,SRAM)接口使得处理器能够以低延迟访问片上存储器(on-chip memory)。调试和分析器单元(Debug and Profiler Unit)提供 JTAG 接口,用于调试、下载程序代码和对处理器的性能进行监控。

MIPSfpga 处理器采用 5 级流水线,但是与标准的 MIPS 5 级流水线稍有不同。MIPSfpga 处理器 5 级流水线如表 4.1 所示,其时空图如图 4.2 所示。

表 4.1　MIPSfpga 处理器 5 级流水线

序　号	流 水 级	名　称	描　述
1	I	取指	处理器取出指令
2	E	执行	处理器从寄存器取出操作数并进行运算
3	M	访存	根据指令，处理器从内存取出操作数或向内存存入操作数
4	A	对齐	根据指令，处理器将从内存中取出的数进行 32 位边界对齐
5	W	写回	根据指令，处理器将结果写回寄存器

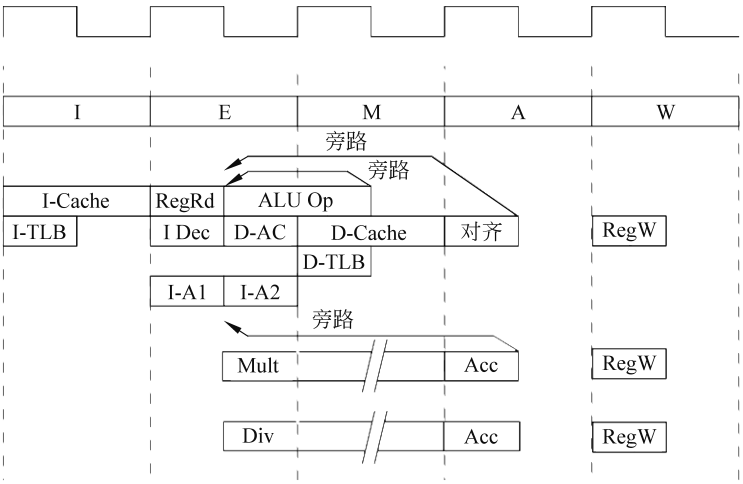


图 4.2　MIPSfpga 处理器 5 级流水线时空图

MIPSfpga 采用 32 位地址空间(虚拟地址和物理地址均是 32 位)。MIPSfpga 有 3 种操作模式，分别为内核模式、用户模式和调试模式。复位后，处理器处于内核模式，并且跳到 0xBFC00000(虚拟地址)复位地址开始运行程序，如图 4.3 所示。0xBFC00000 地址位于 kseg1 段，属于不映射、不缓存地址段。这意味着该段地址中的指令是直接取自内存而不是高速缓存，同时虚拟地址直接映射到物理地址，而不是通过 MMU 进行地址变换。这一点非常重要，因为复位时处理器的 MMU 和高速缓存都还没有完成初始化，不能正常工作。kseg1 段虚拟地址采用直接减 0xA0000000 的方式映射到物理地址，即复位地址 0xBFC00000 将映射到内存的物理地址 0x1FC00000。

图 4.4 为 MIPSfpga 处理器系统的关键部件。该处理器系统的时钟、复位和 EJTAG 编程信号来自 FPGA 开发板或仿真模拟测试平台。在系统最简化情况下，可以仅外接发光二极管(LED)或开关，驱动 AHB 总线接口。在图 4.4 所示的系统中，除 MIPSfpga 处理器核(m14k_top)外，仅包含 mipsfpga_ahb 模块，该模块内含 RAM、GPIO 和 AHB-Lite 总线接口。

mipsfpga_ahb 模块提供的物理地址映射关系如图 4.5 所示。它包括以下几部分：一个起始地址从 0x1FC00000 开始的 128KB RAM，用于存放处理器复位后将执行的程序代码；一个起始地址从 0x00000000 开始的 256KB RAM，用于存放其他的程序代码或数据；另外，还包括 4 个 GPIO 寄存器，用于控制发光二极管和开关的输入和输出(详见 5.3 节)。

图 4.3　MIPSfpga 虚拟地址映射图

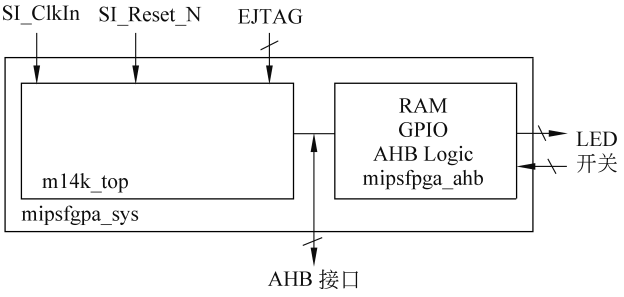


图 4.4　MIPSfpga 处理器系统的关键部件

图 4.5　mipsfpga_ahb 模块提供的物理地址映射关系

4.2　MIPSfpga 处理器的接口

MIPSfpga 处理器系统的接口主要有 3 个：AHB-Lite 总线接口、FPGA 开发板接口和 EJTAG 接口。MIPSfpga 处理器核通过 AHB-Lite 总线与内存和外设连接；FPGA 开发板

接口使得 MIPSfpga 处理器核可以控制 FPGA 开发板上的开关和 LED;EJTAG 接口则用于将程序下载到 MIPSfpga 处理器核,并进行实时调试。下面对这 3 个接口进行详细介绍。

4.2.1 MIPSfpga 处理器接口

MIPSfpga 处理器接口信号如表 4.2 所示。时钟信号(SI_ClkIn)是整个 MIPSfpga 处理器的系统时钟;如果使用 Xilinx 公司的 Nexys4 DDR 开发板,MIPSfpga 可以正常运行的最大时钟频率为 62MHz;建议将开发板上的 100MHz(Nexys4 DDR)时钟分频为 50MHz 输入使用。复位信号(SI_Reset_N)低电平有效(带后缀"_N"),Nexys4 DDR 开发板上的复位按钮(CPU_RESETN)可用于连接该引脚。MIPSfpga 处理器上电后必须先进行复位。

表 4.2 MIPSfpga 处理器接口信号

信号分类	信号名称	备注
系统	SI_Reset_N	处理器复位信号
	SI_ClkIn	时钟信号,最大时钟频率为 62MHz,建议值为 50MHz
AHB-Lite 总线	HADDR[31:0]	总线地址
	HRDATA[31:0]	读的数据
	HWDATA[31:0]	写的数据
	HWRITE	读写控制信号
开发板引脚	IO_Switch[17:0]	接开发板上的滑动开关
	IO_PB[4:0]	接开发板上的按钮
	IO_LEDR[17:0]	接开发板上的 LED
	IO_LEDG[8:0]	接开发板上的 LED
EJTAG	EJ_TRST_N_probe	接开发板上的 EJTAG 接口
	EJ_TDI	
	EJ_TDO	
	EJ_TMS	
	EJ_TCK	
	SI_ColdReset_N	
	EJ_DINT	接地

表 4.2 各个信号名称前缀的意义如下:

- SI 为系统接口信号。
- H 为 AHB-Lite 总线信号。
- IO 为开发板输入输出引脚。
- EJ 为 EJTAG 接口信号。