版本说明

时间	版本	更新
2007.02	V1.0	51board

深圳市武耀博德信息技术有限公司

产品总代理

亿道电子技术有限公司
深圳市福田区上梅林梅林路 10 号
申汇基大厦 5 层　　518049

Tel ： 86-755-83142770
Fax ： 86-755-83142771
www.emdoor.com

第一部分

汇编指令以及接口实验篇

实验一 系统引导实验

[实验目的]

√ 了解 PXA270 处理器功能结构
√ 了解系统的基本硬件组成
√ 了解 ARM 指令集
√ 掌握嵌入式系统的一般引导规律
√ 掌握常见 ARM 开发工具软件的使用

[实验原理]

1．程序介绍

本章主要通过一个简短的 Boot 程序向读者揭示如何编写开发板的启动程序，同时本程序也可以用来引导其他章节的示例程序。本程序主要为了让读者能够清晰了解系统复位后如何从 0x0 开始引导，特意省略了中断向量表，有兴趣的读者可以在本程序上扩建其他功能。本引导程序驱动底板上的八盏 LED 就会向右点亮，不断循环下去。

2．系统复位

对于 PXA270 处理器来说，系统复位后的 PC 指针总是为 0x0，以本开发板来说，片选信号 nCS0 所连接的为 FLASH 芯片，boot 程序应该被烧写到该 FLASH 芯片上，且第一条指令应该放在 0x0 的地址（注意并不是所有的处理器都从地址 0x0 开始运行，有些处理器是从 0xFFFF0 开始运行的）。事实上，地址 0x0－0x20 之间为中断向量表，地址 0x0 为复位中断例程的入口点，即通过在 0x0 放一条无条件跳转语句，在系统加电或复位时，在地址 0x0 开始跳转，从复位中断例程开始运行下去。

但系统复位后，大部分寄存器都被清空，要想存储器上的映像能够运行起来（映像中的可读写段在运行时需要被装载到 SDRAM），这就需要对 Memory Controller registers 进行配置，简单的说，这里配置的目的就是要告诉处理器关于开发板上存储器的相关信息。当然，这只是举例，用户应该根据需要，对处理器和周边设备进行初始化。

映像文件的结构

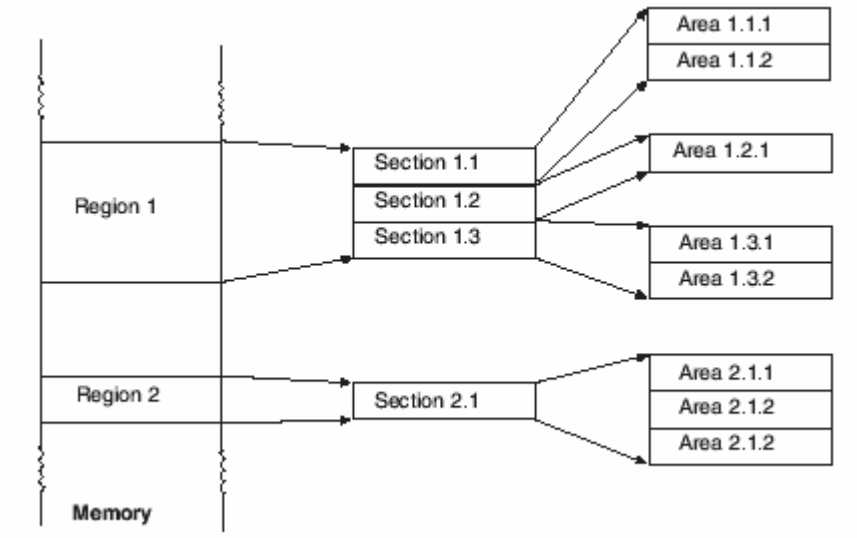


图 1.1 映像文件结构图

Boot 程序在 ARM ADS 软件里被编译和链接后会产生 ELF（Executable And Linking Format）格式的映像文件。上图中的“Area”是指在汇编程序中使用伪操作定义的段,每个 Area 都有相应的属性，可以是只读（RO），可读写（RW）以及初始化为 0（ZI）。除了 ZI 属性的段可以在映像被执行前自动生成外，其他的段都必须通过用户自己定义。在链接过程中，相同属性的段会被重组成一个“Section”，故 Section 的属性是由所包含的 Area 属性决定，在同一个 Section 里的 Area 是紧密相连的。然后多个 Section 再组织成一个或多个 Region，通过对 Region 在存储器中定位，就形成链接后的映像。

Boot 源代码中，有多个 Area，如何保证包含复位中断向量的 Area 被定位在存储器的开始位置。可以通过两个链接参数来决定：－ro-base ， －first.

可用参数－ro-base address1 来设置 Region 的装载位置，该 Region 必须是包含只读的 Section。这里的装载位置 address1 是指映像还没有执行时在 FLASH 空间里被保存的位置。事实上－ro-base 是用作设置 Region 的执行时位置，执行时位置是指映像执行时在存储空间的位置，因为具有可读写属性的 Area 在执行前会从 FLASH 存储空间拷贝到 RAM 存储空间，所以映像里各 Region 装载时和执行时位置可能不同，但对于包含只读 Section 的 Region，装载时位置和执行时位置是一致的，即它不用被拷贝到 RAM。为了让包含复位中断向量的 Area 能够定位在 0x0，这里需要指定－ro-base 0x0，那么，包含代码段的 Region

就被定位在 0x0。

虽然 Region 的位置决定了，但 Region 是由多个 Section 组成，Section 的排列次序又直接影响各 Area 的位置，通常情况下，只读属性的 Section 是放置在 Region 的开始位置，通常代码段是只读的（也可以是读写的），即代码段会放在 Region 的开始位置。所以这里可以推断只读属性的 Section 的开始位置也为 0x0。

最后，排列只读属性 Section 里的 Area，由于包含复位中断向量的 Area 必定在其中，所以需要对该 Area 指定 0x0 位置。默认情况下，同一 Section 里的 Area 是通过名称来排列的。由于本例子里包含复位中断向量的 Area 名字是 boot，所以该 Area 的位置也定位在 0x0。如果需要显示指定，可以使用参数－first，这个参数具体形式是 －first object(area)。读者可以通过产生一个映像的 Memory Map 文件来观察各 Area 的地址分配。

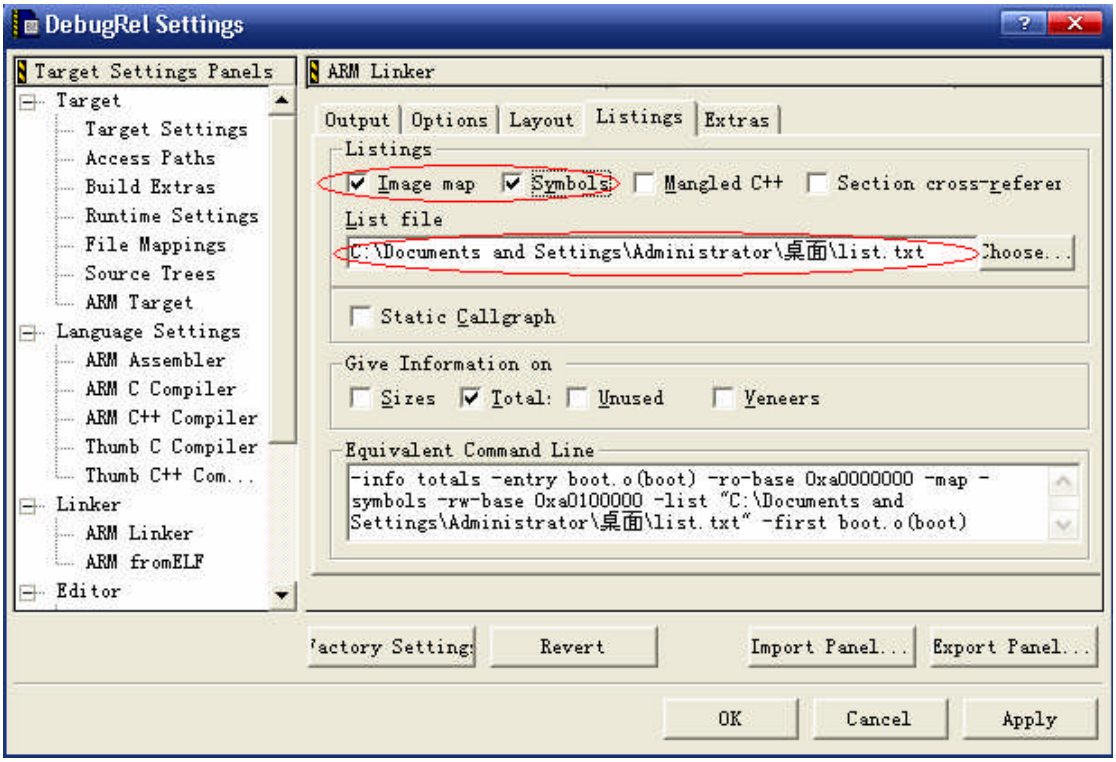


图 1.2 生成映像的 Memory Map

这里还有一点需要补充的，就是链接参数－rw-base 的使用，该参数用来指定可读写 Section 的执行时地址。因为可读写 Section 是由一个或多个可读写的 Area 所组成，所以该参数会影响代码中被定义为可读写的 Area 的位置。因为可读写属性具有写入的原因，如果 Area 被定义为读写的，通常需要指定－rw-base，且地址应该是在 RAM 区域，因为如果指定的区域在 FLASH 区域，则写入是无效的（FLASH 芯片写入一个字节是有规定的写入时序，

它不像 RAM 那样直接可以修改）。如果不定义－rw-base，则可读写的 Section 在执行时的地址就会紧紧跟着只读 Section 的后面，然而，只读 Section 通常是被指定在 ROM 区的，所以，这样可能导致可读写的 Section 也被自动安排在 ROM 区。读者可以通过生成映象的 Memory Map 来查看他们之间的变化。

3．程序进入点

因为引导程序是自举的的程序，无需操作系统加载来执行，所以即使不设置初始入口点也可以执行，但这里是有必要对程序进入点进行描述。

我们可以将 Boot 程序看成是普通的映像文件，假设我们现在已经生成了映像文件，当 Boot 映像被操作系统加载时，如何决定映像被执行的第一条指令呢？这里引入初始入口点和普通入口点。初始化入口点定义了映像的第一条被执行的指令，在编译程序时可以添加参数－entry address（或－entry offset+object(area) ）来标示初始入口点，如果没有添加该参数，只要源程序中有唯一的伪操作 ENTRY，则程序就被默认成初始入口点。即当镜像被烧入 FLASH 后，以 ro_base 属性决定的映像位置的第一条指令就是被定义在 ENTRY 标示的段的第一条指令，读者可能对此有些模糊，我们这里以给定的程序来分析。在 BootLoader 代码里的 boot.s 汇编文件里，可以发现以下程序：

```
AREA boot ,CODE ,READONLY
ENTRY
B Reset_Handler
B Undefined_Handler
B SWI_Handler
B Prefetch_Handler
B DataAbort_Handler
NOP
B IRQ_Handler
B FIQ_Handler
```

这小段代码标示了一个名叫 boot 的代码段，属性为只读，而 ENTRY 本来只表示为一个普通入口点，但在 Boot 代码中，因为只使用一次 ENTRY，所以 ENTRY 就被定义为初始化入口点，这里并不是要求所有的源程序中只能使用一次 ENTRY，相反可以多次使用 ENTRY 来标示普通入口点，但多次使用 ENTRY 后，就无法让系统知道镜像的第一条执行的指令在哪里，就必须在编译时增加－entry address 参数。读者可以尝试一下代码中都不使用 ENTRY 或多次使用 ENTRY 会发生什么现象。回到上面的代码中，我们发现伪操作 ENTRY 下有一条无条件跳转指令 B，由于 AREA 和 ENTRY 都是伪操作，在不分配成实质的指令，所以，程序的第一条会被执行的指令就是 B 指令，作为 Boot 代码，初始入口点是不起作用的，因为它无需被加载而运行，初始入口点保存在 ELF 头文件中，该值可被操作系统读取而跳转到初始入口点执行。

```
Reset_Handler
    ;*************************
    ;Check if run in the SDRAM
    ;*************************
    MOV  R0,PC
    CMP R0,#0x00000040
    BNE Stack

    ;*************************
    ;Init Memory
    ;*************************
    mov r14,pc
    ldr pc,=post_initMem

    ;*************************
    ;Init Stack
    ;*************************
Stack
    mov r14,pc
    ldr pc,=init_Stack

    ;*************************
    ;Init Gpio
    ;*************************
    mov r14,pc
    ldr pc,=post_initGpio

    ;*****************************
    ;Init Variant
    ;*****************************
    mov r14,pc
    ldr pc,=post_initVariant
```

4. 系统初始化

在 Boot 代码中，通过 B 指令跳转到程序的系统初始化部分。以下是程序的初始化代码

指令“mov r14, PC”的目的是保存返回地址，因为各部分的初始化代码被组织在不同的段中，将段地址赋值给 PC 指针，便可实现跳转。初始化部分应该根据需要，就以本程序为例，段 post_initMen 则对开发板的存储器进行初始化工作。段 post_initStack 主要完成初始化堆栈。段 post_initGpio 主要完成对 MCPU 的 Pin GPIO 的功能的定义。段 post_initVariant 主要对程序使用的几个变量进行初始化，它们的地址都定位在 SDRAM 上。当完成基本的初始化工作后，Boot 程序便可以跳转到用户程序的开始地址处执行。故 Boot 程序的编写可以算已经完成。

5. Led 程序分析

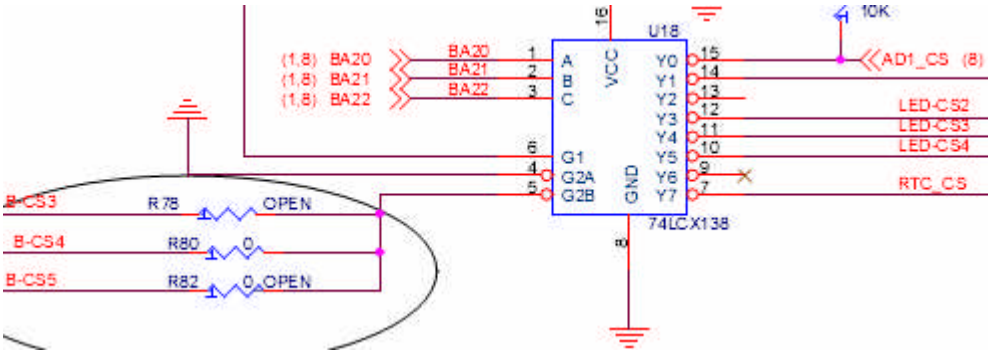


图 1.3 LED_CS4 片选地址

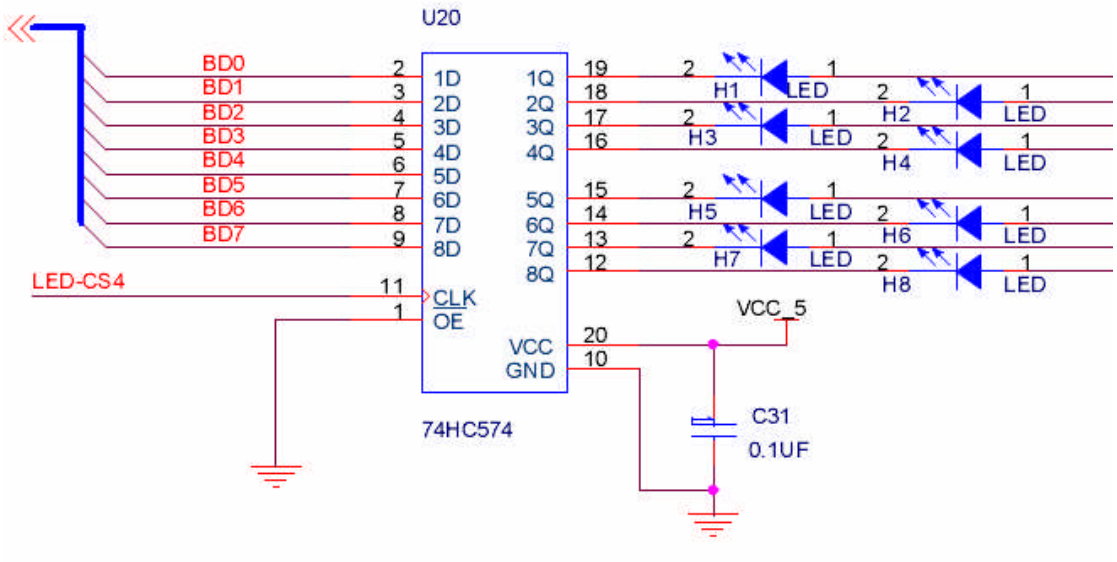


图 1.4 LED 电路图

由图 1.3 与 1.4 可知，LED_CS4 为八个 LED 的片选，由于 B_CS4 的地址为 0x10000000，加上 BA20，BA21 和 BA22 组成的值，LED_CS4 的地址为 0x10500000。在 74HC574 中，当 BDx 为 0 时，相应的 LED 的就会点亮。

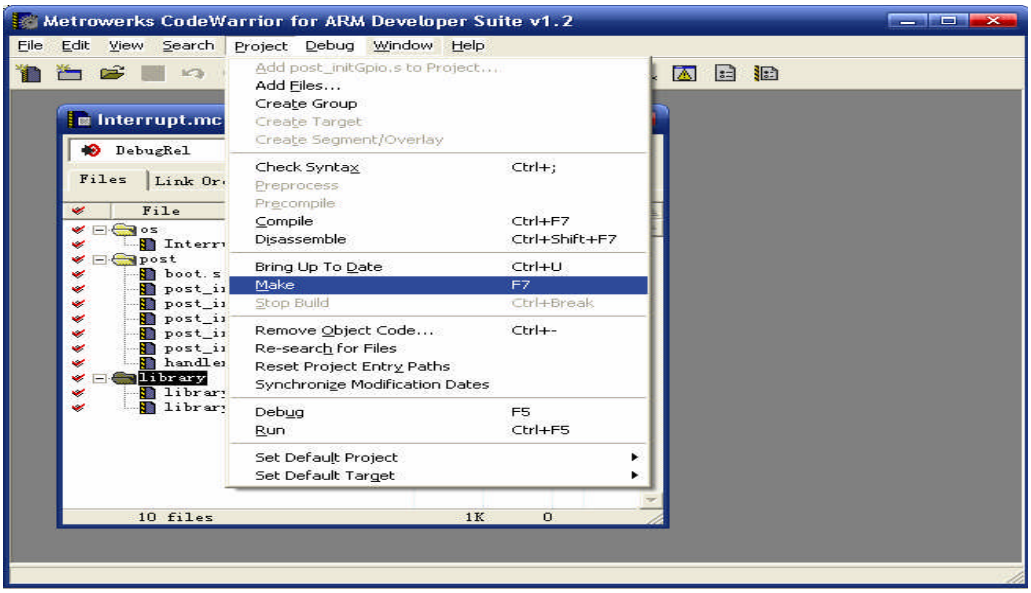
[实验内容]

1)分析代码

结合以上说明，对本实验所提供的汇编源代码进行分析，深入理解针对具体的硬件实现，软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图所示：



编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1. 简述 ELF 文件的内部层次结构。
2. 简述连接器的 4 个参数－ro_base，－rw_base，－first，－entry 的意义。
3. 简述初始化入口点和普通入口点的区别，分别用在什么场合。

实验二 8段码实验

[实验目的]

√ 在实验一的基础上进一步了解ARM体系结构和编程方法
√ 了解8段码的知识

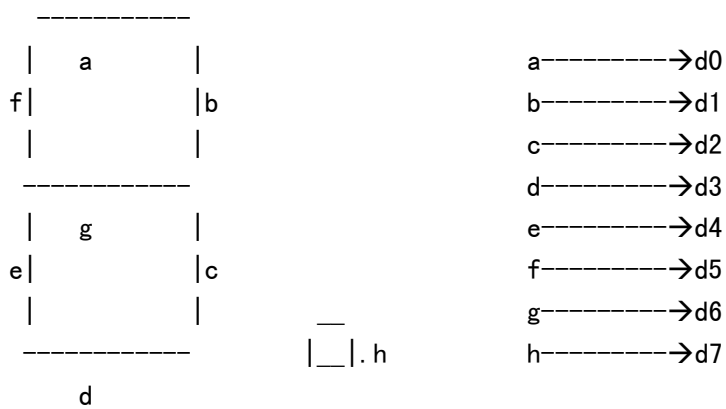
[实验原理]

1．程序介绍

本程序在实验一的基础上增加了对8段码显示的支持，在系统开机后，会在4个八段管上分别显示1到4的数字，每隔一定的时间，数字递增一次。每个LED上升到9时回到0。

2．8段管原理

8段管是由8个LED灯（每个灯我们称为一个段）组合成的形状为数字8带小数点的图形，通过控制每个段的开启和关闭来形成数字图形0到9。每个段通过一个缓冲器和数据总线相连，在本系统中如果数据为0则对应的该段点亮。段码和数据线的对应关系如下：



例如：要显示数字1，我们只需要打开b,c两个段，对应的数据为0x79;

3 硬件原理图

图 2.1 表示四个 8 段数码管的片选地址，图 2.2 表示数码管。

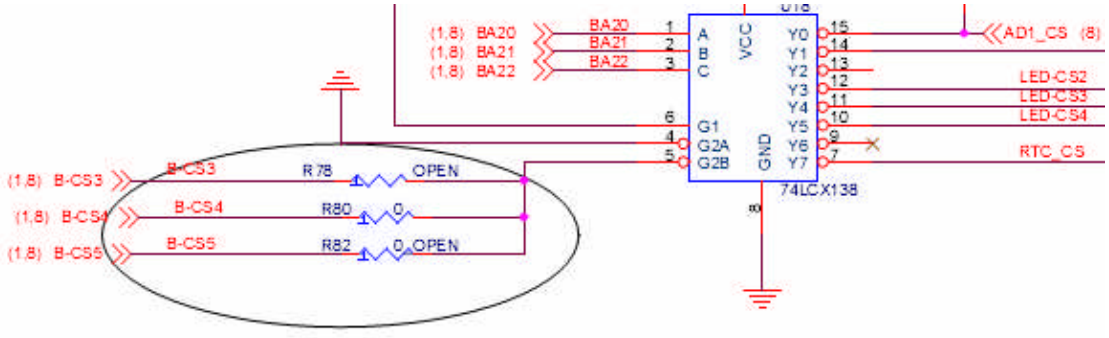


图 2.1 LED_CS2 与 LED_CS3

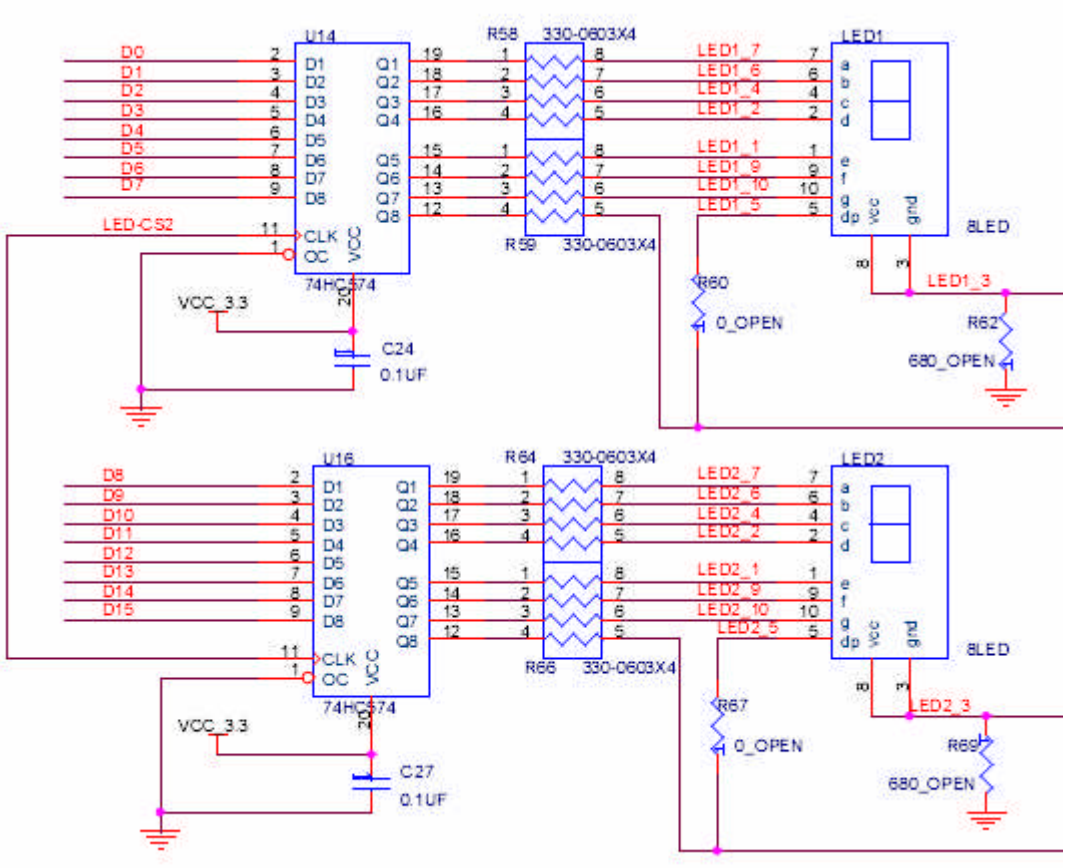


图 2.2 LED1 与 LED2

由图 2.1 与图 2.2 可知，LED_CS2 为 LED1 与 LED2 的片选，LED_CS3 为 LED3 与 LED4 的片选。由于 B_CS4 的地址为 0x10000000，加上 BA20，BA21 和 BA22 组成的值，LED_CS2 的地址为 0x10300000，LED_CS3 的地址为 0x10400000。

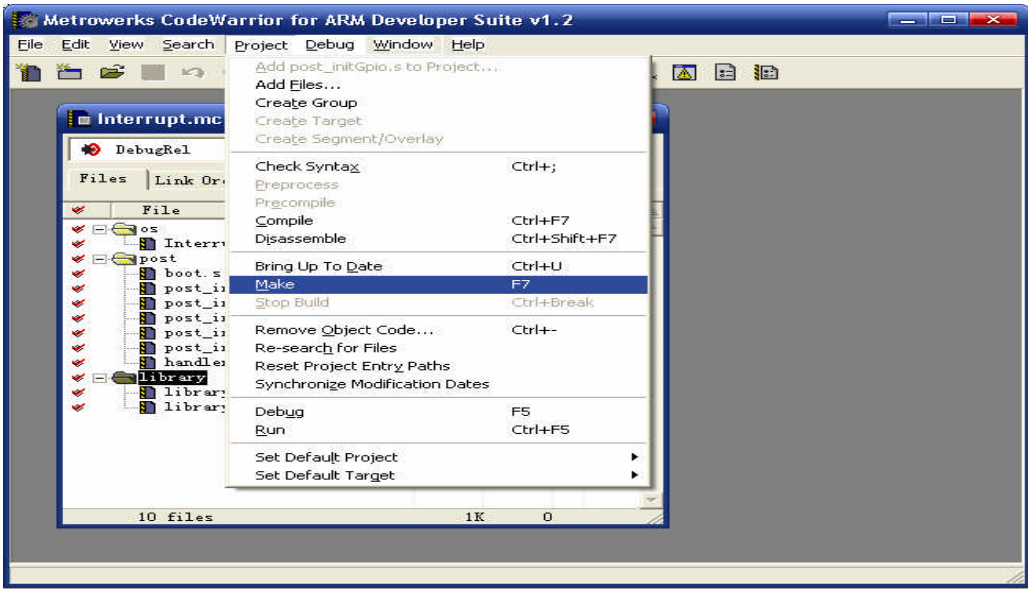
[实验内容]

1)分析代码

结合以上说明，对本实验所提供的汇编源代码进行分析，深入理解针对具体的硬件实现，软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图所示：



编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1、在调试程序时，当停止后，数码管为什么能够显示原来的数字？
2、怎样显示大于 9 的数字时。

实验三 键盘实验

[实验目的]

√ 了解直入键盘与矩阵键盘的原理
√ 了解键盘寄存器的功能

[实验原理]

1、程序介绍

本章例子结合实验二的八段数码管，通过对键盘的操作，实现对八段数码管控制。当按 1 键的时候，LED1 数码管就会亮，同理，按相应的键，相应的数码管就会亮。

2、概念理解

在 EELiod 中，使用了直入键盘与矩阵键盘结合的方式。其中 SW1-SW4 使用的是直入键盘，SW5-SW16 使用的是矩阵键盘。图 3.1 显示了直入键盘与矩阵键盘的结构图。

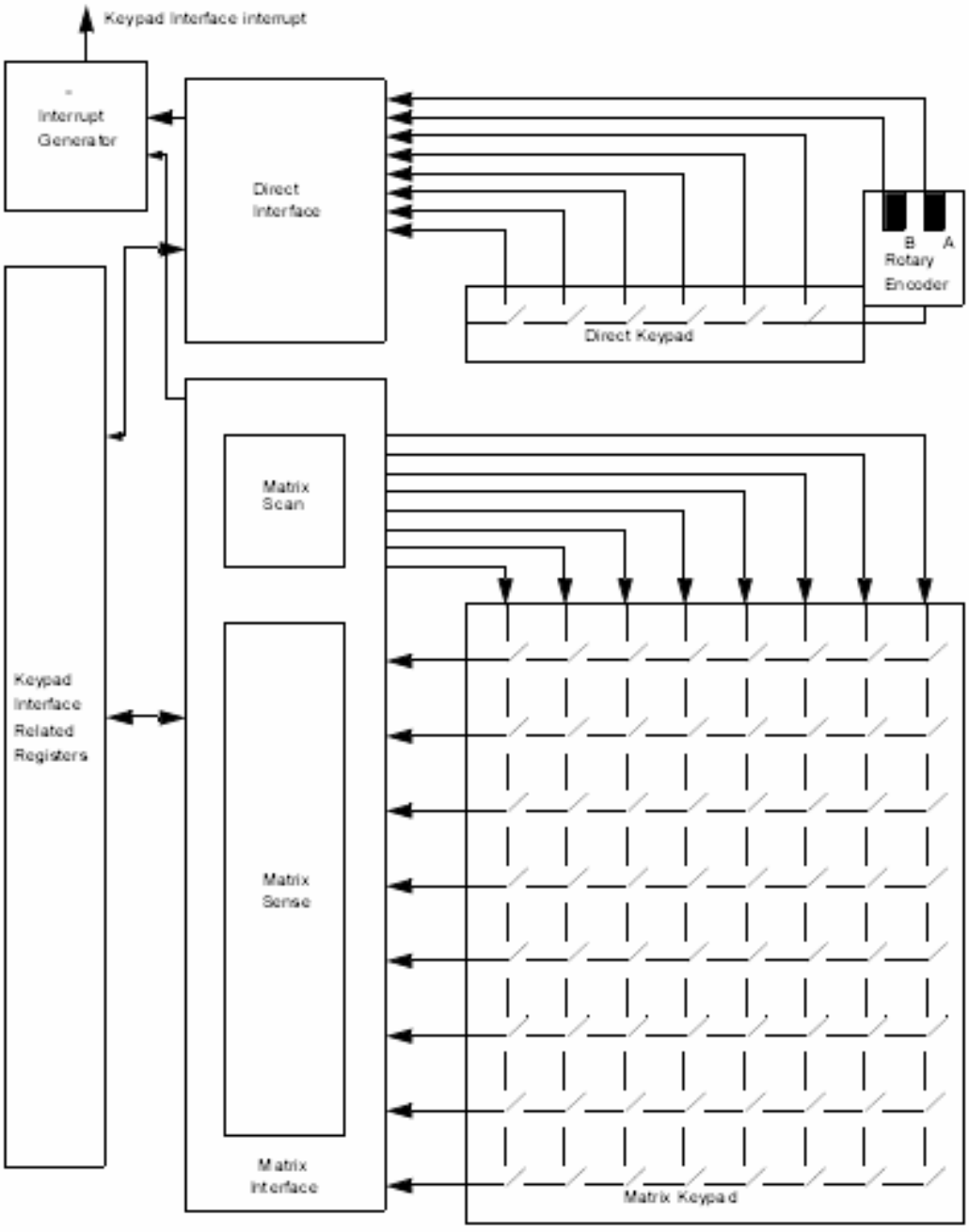


图 3.1 键盘接口结构图

直入键盘接口

在 EELiod 中，系统可以支持八个直入键盘，或者六个直入键盘和一个旋转译码器（两

个管脚组成一个旋转译码器），或者四个直入键盘和二个旋转译码器。如图 3.2 所示，显示直入键盘的电路图。

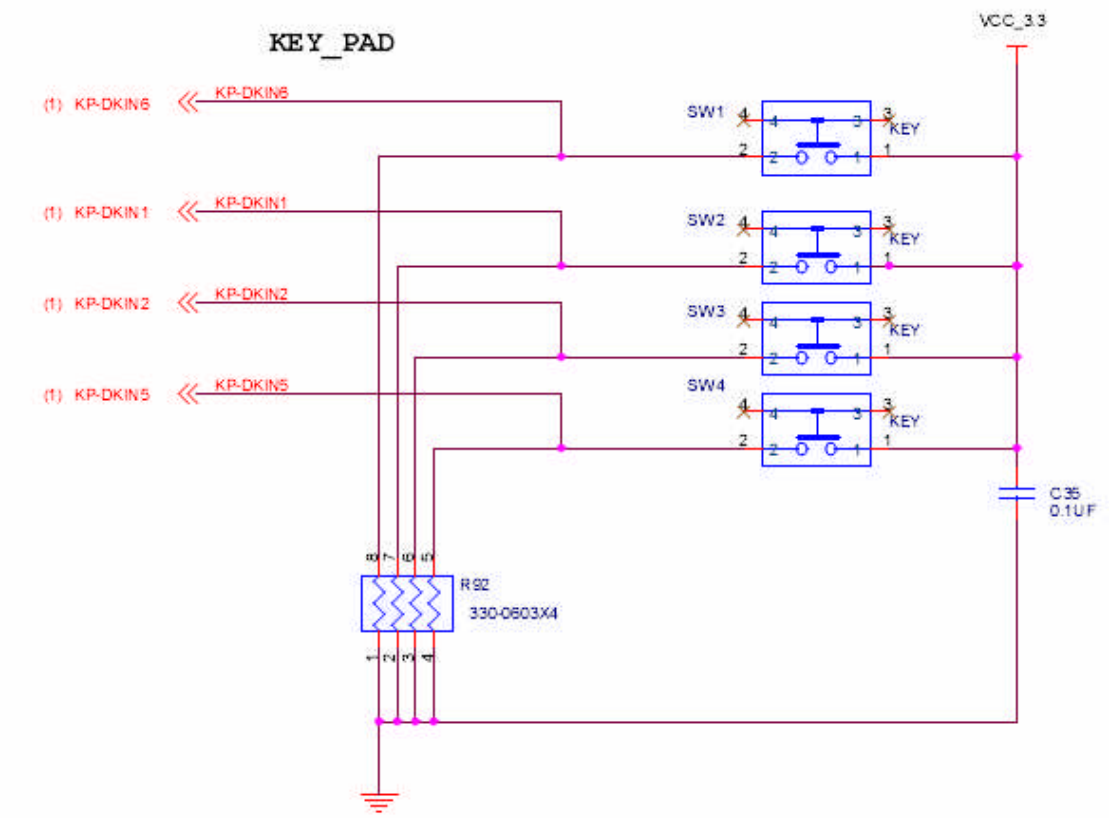


图 3.2 直入键盘电路图

其中 SW1 中的 KP-DKIN6 的 GPIO 口是 99，SW2 中的 KP-DKIN1 的 GPIO 口是 94，SW3 中的 GPIO 口是 95，SW4 中的 KP-DKIN5 的 GPIO 口是 98。

当按下某一个键盘时，键盘就会导通，就会向所在 GPIO 口发送一个高电平，触发按键事件。CPU 通过检测按键位置，从而在键盘寄存器显示相应的值。

矩阵键盘

矩阵键盘接口支持自动扫描与手动扫描的方式进行对矩阵键盘按键的检测。最多可以支持八个输入/输出，总共 64 个矩阵键盘。由图 3.3 所示，组成了 3X4 的矩阵键盘。CPU 通过 KP-MKINx 自动或者手动发出信号检测矩阵键盘的的按键情况。加上四个直入键盘，就组成了 4X4 键盘。

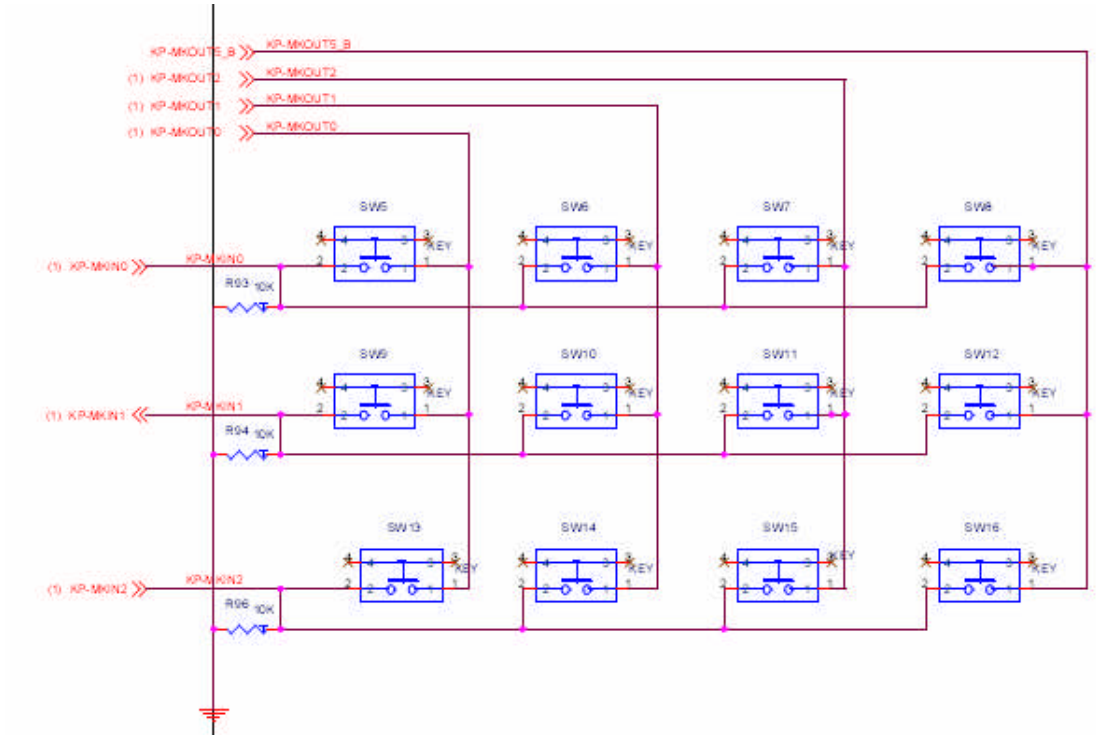


图 3.3 矩阵键盘接口电路图

3、键盘寄存器

KPC（Keypad Interface Control register）

键盘接口控制寄存器是一个可读写的寄存器，它可实现对键盘的全局控制操作。包括对直入、矩阵键盘的使能、中断、自动扫描等。

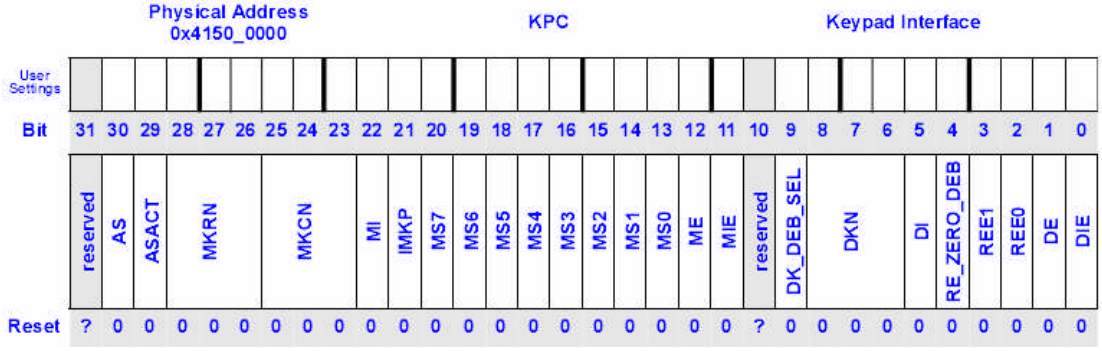


图 3.4 KPC 寄存器

举例说明一下：

```
LDR R1,=0x41500000
;KPC
LDR R0,=0x2FAFF1C2    ;(0x2FAFF9C3:interrupt)
STR r0,[r1,#0x0]
```

这样就可以对键盘初始化操作了。

KPDK（Keypad Interface Direct Key register）

直入键盘寄存器是一个只读寄存器，它显示了直入键盘的按键值。如图 3.5 所示，表示各个直入键盘的状态。

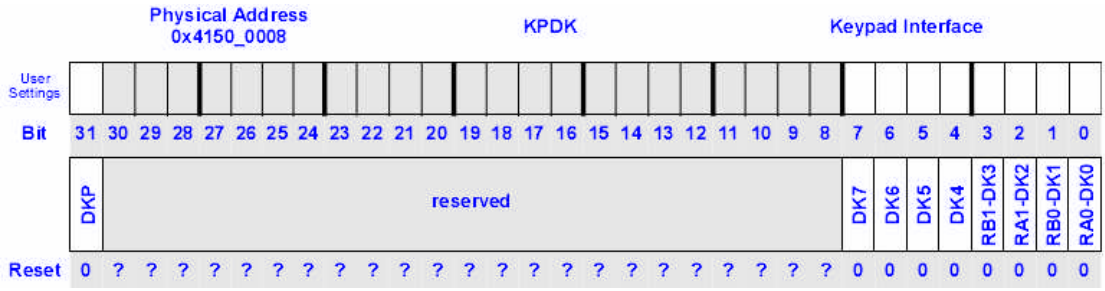


图 3.5 直入键盘寄存器

举例说明一下：

```
#define KPDK_VALUE     (*((volatile unsigned char *)(0x41500008)))
Int temp;
Temp = KPDK_VALUE;
```

KPAS（Keypad Interface Automatic Scan register）

键盘接口自动扫描寄存器是一个只读寄存器，主要功能是自动扫描的矩阵键盘的值，如图 3.6 所示。

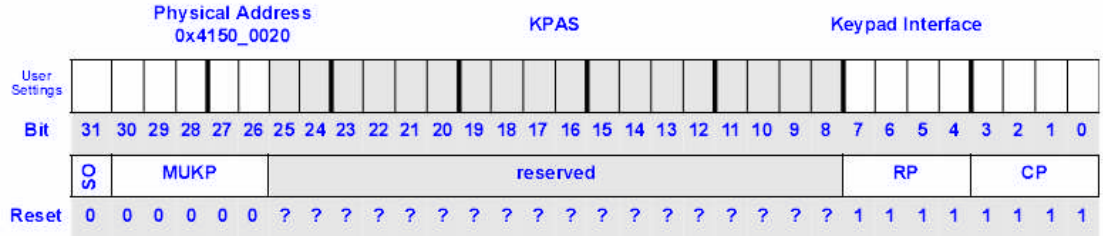


图 3.6 键盘自动扫描寄存器

4、程序分析

本程序是通过键盘的按键来驱动 8 段数码管的点亮，图 3.7 为程序的流程图。

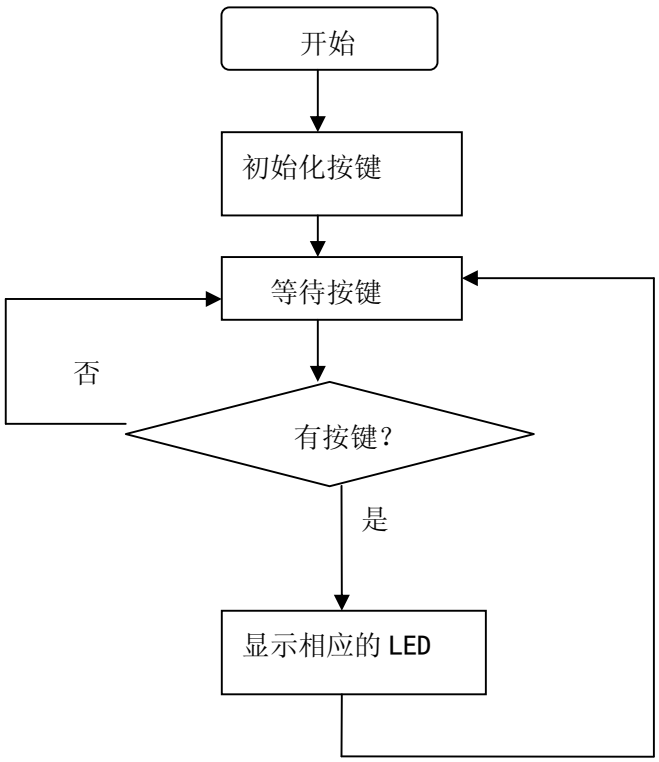


图 3.7 流程图

[实验内容]

1)分析代码

结合以上说明，对本实验所提供的汇编源代码进行分析，深入理解针对具体的硬件实现，软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文

件。如图 3.8 所示：

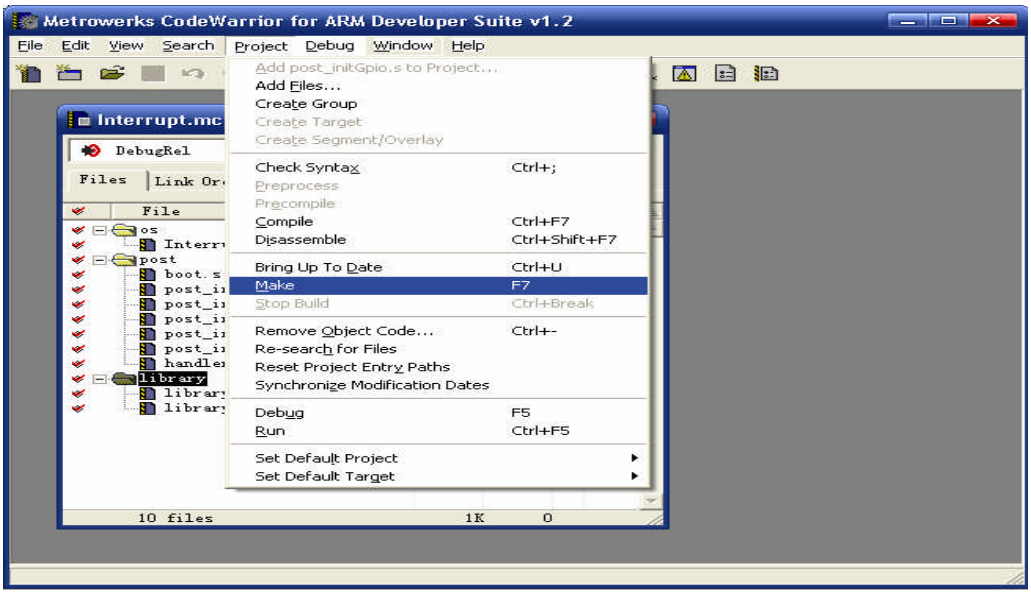


图 3.8 编译、连接及生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1、简述直入键盘接口与矩阵键盘接口的区别。

2、程序中没有用到矩阵键盘，现在要变为矩阵键盘，如何修改程序？

实验四 IRQ 中断处理

[实验目的]

√ 学习编写中断处理程序

[实验原理]

1．程序介绍

本章例子主要使用按键实现外部中断，执行中断服务子程序。系统启动后，按 12345678 任意键，相应的八段数码管就会点亮。

2．中断向量表

当异常中断发生时，系统执行完当前指令后，将跳转到相应的异常中断处理程序处执行。处理器能够准确无误地响应中断，是因为 ARM 体系结构里有一个中断向量表，该中断向量表将系统能够响应的 7 种异常中断类型的“入口地址”登记在一块连续的字节空间内，每种异常中断的“入口地址”占据 4 个字节，这里“入口地址”实质是一些跳转指令或者是让 PC 指针赋值的指令，通常使用 B 或 ldr 指令。简单地说，异常中断发生首先会跳转到中断向量表，此时跳转的位置会由系统根据中断类型来判断，由于中断向量表实质也是跳转指令所组成的指令序列，所以系统会再进行一次跳转，这次跳转便跳到中断处理程序(中断服务例程)的入口。从第一章开始，读者在学习 Boot 程序的编写时便开始接触中断向量表，Boot 程序的第一条指令 b post 就是中断向量表里的第一个单元空间（4 个字节），这个单元空间是用作处理复位异常中断。以下给出中断向量表的相关信息：

表 4.1 ARM 中断向量表

中断向量地址	中断类型	触发原因
0x0	复位	系统加电或复位
0x4	未定义指令	ARM 处理器或协处理器认为当前指令未定义
0x8	软件中断（SWI）	执行软中断指令 SWI
0x0c	指令预取中止	预取的指令地址不存在
0x10	数据访问中止	访问的目标地址不存在
0x14	保留	不使用
0x18	外部中断请求（IRQ）	处理器的外部中断请求引脚有效，并且 CPSR 寄存器的 I 控制位被清除。
0x1c	快速外部中断（FIQ）	处理器的外部快速中断请求引脚有效，并且 CPSR 寄存器的 F 控制位被清除。

由于一条 B 和 Ldr 指令就是 32 位的指令，即占 4 个字节，所以整个中断向量表可以是如下形式：

```
IMPORT   Reset_Handler
IMPORT   Undef_instrution_Handler
IMPORT   SWI_Handler
IMPORT   Prefetch_Handler
IMPORT   Abort_Handler
IMPORT   IRQ_Handler
IMPORT   FIQ_Handler

AREA boot ,CODE ,READONLY
B    Reset_Handler
B    Undef_instrution_Handler
B    SWI_Handler
B    Prefetch_Handler
B    Abort_Handler
NOP
B    IRQ_Handler
B    FIQ_Handler
```

或者使用 Ldr 指令来实现：

```
AREA boot ,CODE ,READONLY
LDR PC, =Reset_Handler
LDR PC, =Undef_instrution_Handler
LDR PC, =SWI_Handler
LDR PC, =Prefetch_Handler
```

```
LDR PC, =Abort_Handler
NOP
LDR PC, =IRQ_Handler
LDR PC, =FIQ_Handler
```

这里使用 B 和 LDR 指令的主要区别是指令跳转范围，对于 B 指令，可以跳转的范围为－32MB ～ ＋32MB，而 LDR 则可以将一个 32 位常数和地址值读取到 PC 寄存器中，适合于大范围的跳转。

从表 4.1 可见，中断向量表是定位在 0x0 到 0x20 之间，从编程的角度来看，用户需要对中断向量表在可执行映像中的位置进行定位，即必须将包含中断向量表的 AREA 定位在映像中的最开始位置，而且，该映像必须烧写在 ROM 空间 0x0 的开始位置上，读者可以参考第一章的内容来实现。

3．IRQ 和 FIQ 中断开关

系统复位后，IRQ 和 FIQ 中断都是被禁止的，所以即使建立了中断向量表，当有中断请求也是不会响应的，因此在系统复位后，必须通过程序控制来打开 IRQ 和 FIQ 中断。IRQ 和 FIQ 的控制位分别是当前程序状态寄存器 CPSR 的第 7 和第 6 位，对这两个控制位的修改是不能直接地通过对 CPSR 进行操作，而是首先通过读取 CPSR 到通用寄存器中，然后修改，再写入到 CPSR 里。

读取状态寄存器到通用寄存器可以通过 MRS 指令，将通用寄存器的值写入到状态寄存器可以通过 MSR。这里假设开启 IRQ 和 FIQ 中断，程序如下：

```
MRS  R0, CPSR
AND  R0, R0, #0x3F
MSR  CPSR_C, R0
```

CPSR_C 表示 CPSR[7:0]控制位域，将 CPSR[7]和 CPSR[6]清零便可以开启 IRQ 和 FIQ 中断处理。通过这一步设置，只是打开了 IRQ 和 FIQ 中断的总开关，而且，对于 ARM 体系结构的 CPU 都是必须的。IRQ 和 FIQ 对应于中断向量表的不同位置，同时，FIQ 叫快速中断模式，它的优先级比 IRQ 高，执行 FIQ 中断处理可以无需跳转到中断服务例程，因为 FIQ 位于中断向量表的最末位置，故 FIQ 中断服务例程可以直接安排在 0x1c 开始的位置，所以 FIQ 可以对中断请求实行快速的响应。 对中断源的具体操作还需对集成在处理器内部的中断控制器进行配置，不同型号的处理器里的中断控制器略有不同，这里只针对 Intel PXA270 进行讨论。

4．中断控制器

Intel PXA270 内部集成了中断处理器，该处理器能对 23 个中断源进行操作，这种操作

包括：

通过设置寄存器 ICMR 屏蔽中断源。

通过设置寄存器 ICLR 对中断源分类，即可以让中断源发出的中断请求以 IRQ 中断方式或以 FIQ 中断方式被处理。

可以查询寄存器 ICPR 得知 23 个中断源当前是否有中断请求，在 ICPR 寄存器上显示发出中断请求的中断源不受 ICMR 影响，即 ICMR 即使屏蔽某个中断源，只要中断源发出中断请求，ICPR 仍然会在相应的位上显示“1”。

可以查询寄存器 ICIP 得知以 IRQ 方式被处理的中断源是否发出中断请求，该寄存器受到 ICMP 影响。即 ICIP ＝ ICPR & ICMR & （～ICLR）。

可以查询寄存器 ICFP 得知以 FIQ 方式被处理的中断源是否发出中断请求，该寄存器受到 ICMP 影响。即 ICFP ＝ICPR & ICMR & ICLR。

图 4.1 显示某个中断请求进入中断控制器后，成功被响应为 IRQ 和 FIQ 的流程图，读者可以先浏览第 5 段“中断控制寄存器”。这里先假设处理器的时钟和电力管理模式为运行模式。首先中断源发出的中断请求“Interrupt Source Bit”会送到寄存器 ICPR 上，相应的位会被设置 1，该中断请求会受到 ICMR 影响，如果在 ICMR 上与中断请求相对应的位被设置 1，则该中断请求就会继续发送出去，否则会被屏蔽。继续发送出去的中断请求通过 ICLR 上的设置被分为 IRQ 或 FIQ 中断，同时该中断请求会发送到寄存器 ICIP 或 ICFR 上。最后，中断请求就会以 IRQ 或 FIQ 异常中断方式被处理，即中断控制器会根据中断向量表，跳转到适合的中断服务例程处理该中断请求。

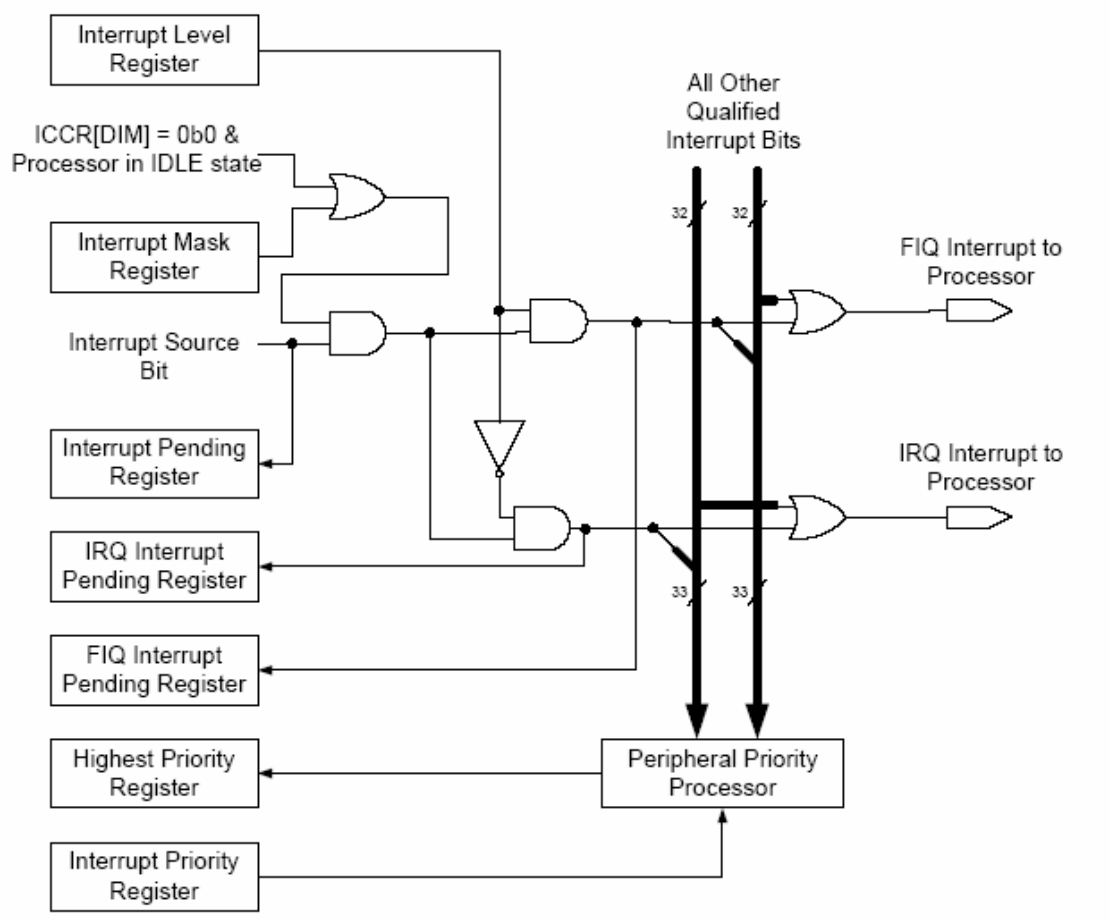


图 4.1 中断控制器操作流程

这里有个问题需要注意，中断控制器只支持简单的优先级——IRQ 和 FIQ，FIQ 的优先级比 IRQ 高。但是同一级别的中断请求如何被处理呢？究竟哪个中断源会被响应呢？通常，可以在 IRQ 或 FIQ 中断服务例程通过读取 ICIP 或 ICFP 上的各位顺序来决定优先级。另外，当中断发生时，系统并不知道是由哪个中断源引起中断的，可以在 IRQ 中断服务例程里逐位的轮询寄存器 ICIP 或在 FIQ 中断服务例程里逐位轮询寄存器 ICFP。

5. 中断控制寄存器

中断控制寄存器包括 6 个寄存器，它们分别是：

Interrupt Controller Mask register (ICMR)

该寄存器能够屏蔽中断源，寄存器上每 1 位对应一个中断源，只要在相应的位上设置 0，就能屏蔽响应的中断。系统复位后，该寄存器全为 0，即所有的中断源都被屏蔽。

Interrupt Controller Pending register (ICPR)

该寄存器是一个只读的 32 位寄存器，它显示了所有发出中断请求的中断源。ICPR 不受寄存器 ICMP 影响，即只要中断源发出中断请求，即使 ICMP 屏蔽了中断源，都会在 ICPR 相应的位上显示。

Interrupt Controller Level register (ICLR)

该寄存器能够决定中断源以 IRQ 或 FIQ 中断方式被处理。只要中断源没有被屏蔽，设置该寄存器后，如果是以 IRQ 中断方式被处理，则 ICIR 相应位会置 1，如果是以 FIQ 中断方式被处理，则 ICFR 相应位会置 1。系统复位后该寄存器全为 0，则表示所有的中断源都以 IRQ 方式被处理。

Interrupt Controller IRQ Pending register (ICIP)

该寄存器是只读的 32 位寄存器，它显示所有没有被屏蔽，且以 IRQ 方式请求的中断源。

Interrupt Controller FIQ Pending register (ICFP)

该寄存器是只读的 32 位寄存器，它显示所有没有被屏蔽，且以 FIQ 方式请求的中断源。

Interrupt Controller Control register (ICCR)

该寄存器只包括一个简单的控制位——ICCR[DIM]。当处理器处于“IDLE”模式时，如果该控制位被设为 1，只有非屏蔽的可用中断源才能将处理器从“IDLE”模式中唤醒。否则，即使屏蔽了中断源，所有的中断源仍可以将处理器唤醒。

6. 中断控制器的使用

在系统初始化的时候，除了要完成打开 IRQ 和 FIQ 中断开关，还要对中断控制器进行设置。假设要开启键盘中断，并以 IRQ 的方式被处理，设置以下符号：

```
;Interrupt Controller
ICMR            EQU     0x40d00004
init_ICMR       EQU     0x00000010
```

开启中断源，设置屏蔽寄存器 ICMP

```
    ldr  r1, =ICMR
    ldr  r2, =init_ICMR
    str  r2,[r1]
```

在 ICMR 里，KEYPAD 代表键盘中断，键盘的中断的位是第 5 位。在这位上设 1，则允许这些中断源请求。

在中断服务例程中查询中断源请求状态

```
    ldr     r11,   = REG_ICIR
    ldr     r4,   [r11]
    mov    r5,   #0x40000000
    mov    r7,   #5

    handleIdentifyLoop
              and    r6,   r4,   r5
              cmp    r6,#0x0
          bne    interruptIdentify

              mov    r5,   r5,   LSR #0x1
    sub     r7,   r7,   #0x1
              cmp    r7,   #0x0
               bne    handleIdentifyLoop
```

以上程序通过对 ICIR 的 5 个状态位进行轮询，当某一位为 1 时，则表示中断源已经发出了中断请求，然后则跳转到 interruptIdentify 运行。如果该 5 个中断源没有被屏蔽，且都以 IRQ 方式处理，则轮询寄存器 ICPR 的效果与 ICIP 是一样的。

7. IRQ 异常中断的响应过程

处理器响应 IRQ 异常中断时，在执行中断服务例程的第一条指令前，处理器会自动完成以下操作：

保存返回地址：R14_IRQ = 下一条将要执行的指令的地址＋4 个字节

保存当前状态：SPSR_IRQ = CPSR

改变处理器模式：CPSR[4:0] = 0b10010

4) 切换到 ARM 状态：CPSR[5]=0

5) 禁止 IRQ 异常中断：CPSR[1]=1

6) 跳转到中断向量表的 IRQ 向量地址执行：PC ＝ 0x018

在执行完以上操作后，由于在地址 0x18 处是一条跳转指令，故程序会跳转到 IRQ 中断服务例程。这里需要强调，在 IRQ 中断服务例程里，处理器的运行模式已经改变为外部中断模式，因此对 R13 和 R14，SPSR 的操作实质是外部中断模式（IRQ）下的 R13 和 R14，SPSR，例如，mov pc，r14，这里操作结果是将 R14_IRQ 的值赋予 PC 指针寄存器。

读者在编写服务例程时，唯一需要注意的是保存现场，包括保存 CPSR，服务例程里所使用的寄存器，返回地址，由于 CPSR 和返回地址在进入中断服务例程前都已经由处理器自动保存。所以，服务例程的一般形式可以如下

```
AREA  IRQ_Handler，CODE，READONLY
SUB  LR，LR，＃4
STMFD  SP！，{寄存器列表，LR}
….
程序主体
….
LDMFD  SP！，{寄存器列表，PC}^
END
```

以上程序将返回地址减 4 个字节的原因是因为执行完当前指令后，准备执行异常中断处理时，处理器已经预取了 2 个指令，而正确的返回地址应该是预取后的 PC 地址减 4 个字节。再次将 LR（R14_IRQ，返回地址）保存在栈中，是为了防止在程序的主体使用 BL 再次跳转时将保存在 LR 寄存器的中断返回地址冲失。最后使用的指令 LDMFD 恢复寄存器原来的数值，标识符^表示将 SPSR_IRQ 寄存器内容复制到当前程序状态寄存器 CPSR 中，那么 CPU 的处理器模式恢复为原来的模式，并且也打开了 IRQ 中断开关。

8．例子程序分析

程序的最开始部分为中断向量表，这里只为复位中断（0x0）和 IRQ 中断（0x10）指定了中断服务例程。当系统加电后便执行 0x0 位置上的指令，这个位置同时是复位异常中断在中断向量表里的位置，程序一开始便跳转到复位异常中断例程，然后完成 Memory Controller，设置各种中断的堆栈，GPIO 方面的初始化，另外程序开启了 IRQ 中断。

程序中建立了两个有效的中断向量表，第一个中断向量表的建立是必须的，是由处理器的体系结构规定，第二个中断向量表的建立只是程序中的策略，该向量表每四个连续的字节为一个单元，该单元内存放的是个中断处理程序的入口地址，这样做的目的是因为当程序响应 IRQ 中断时，处理器并不知道是由哪一个中断源发出的请求，所以在内存中建立一个中断向量表，当 IRQ 中断发生时，通过查询寄存器 ICPR 或 ICIR 读取内存中 IRQ 中断向量表里的地址，然后再跳转到该地址执行。

当发生按键事件时，键盘中断发生，发出中断请求，CPU 响应，进入 IRQ 异常中断服务例程，然后程序根据按键的具体情况来处理，完成中断事件后，从中断服务例程返回。IRQ 异常中断服务例程流程图如下：

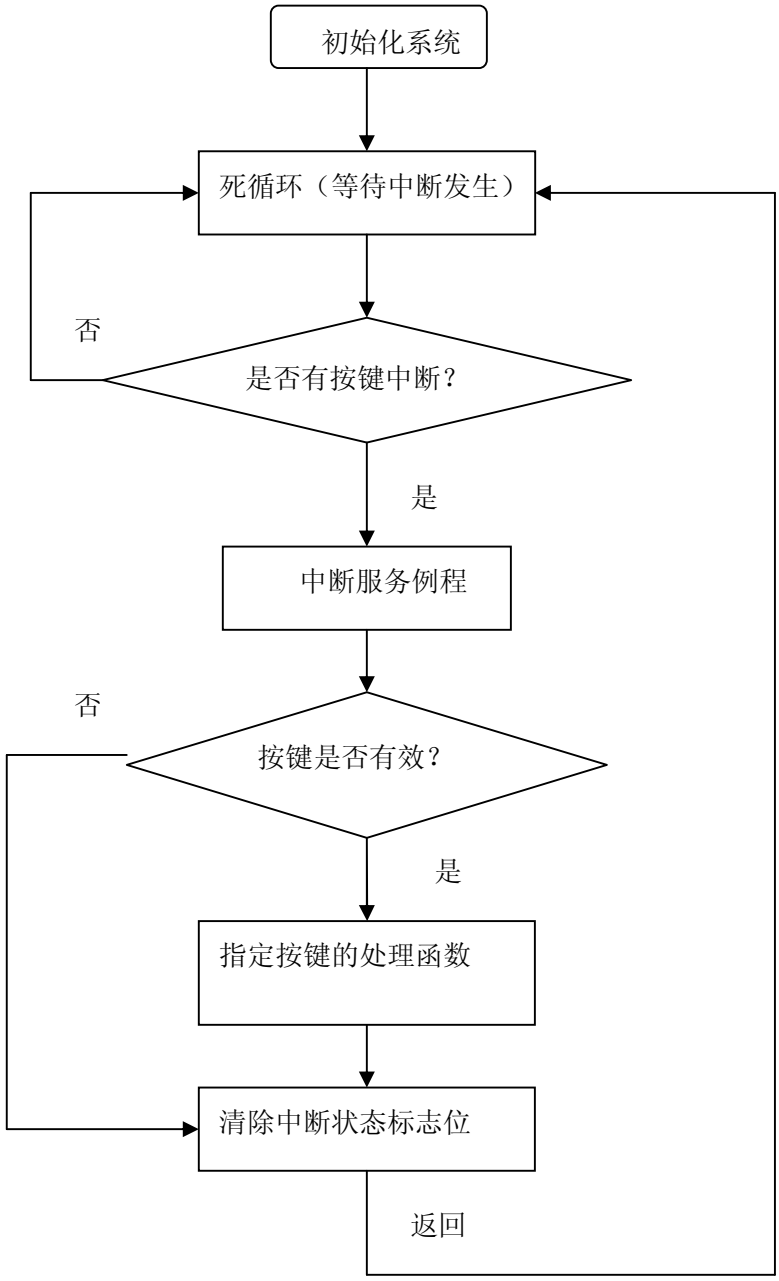


图 4.2 中断服务例程流程图

[实验内容]

1)分析代码

结合以上说明,对本实验所提供的汇编源代码进行分析,深入理解针对具体的硬件实现,软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图 4.3 所示：

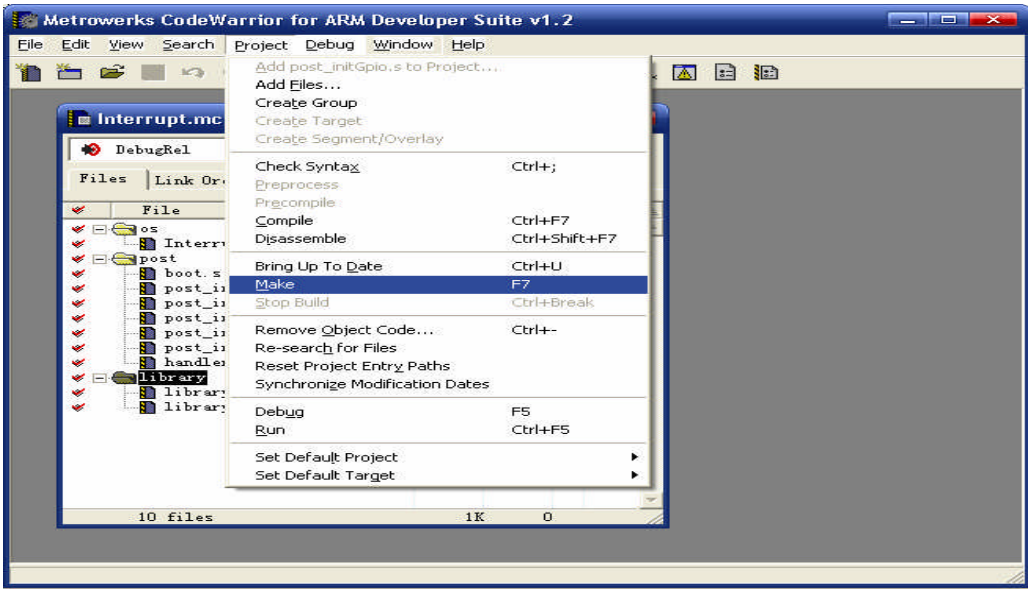


图 4.3 编译、连接生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1．ARM 体系结构中有多少种异常中断，它们分别是在什么情况下发生，并且它们是如何被组织的。

2．查看 ARM 体系结构的资料，结合本章例子程序，完善中断向量表，为软中断和数据访问中止异常中断编写服务例程，并在程序中适当加入语句令程序产生这两种中断。

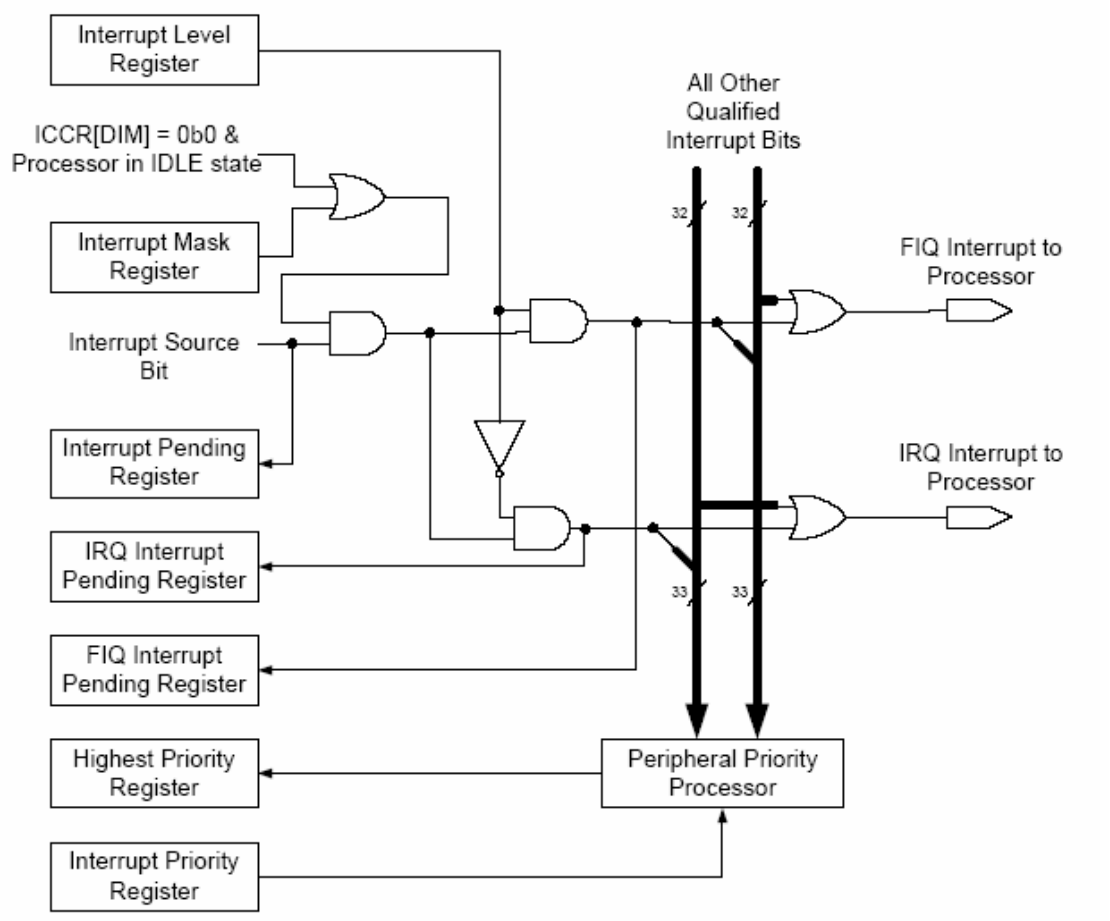
3．结合下图，简要阐述中断控制器的工作原理，并分析当以下情况发生时，各状态寄存器（ICPR，ICIP，ICFP）的数值以及中断如何被响应。

A．ICMR＝1；ICLR＝0；

B．ICMR＝1；ICLR＝1；

C．ICMR＝0；ICLR＝1；

以上设置寄存器的方式仅表示在寄存器中对应中断源的那一位被设置 1 或 0；


Interrupt Level Register
ICCR[DIM] = 0b0 & Processor in IDLE state
Interrupt Mask Register
Interrupt Source Bit
Interrupt Pending Register
IRQ Interrupt Pending Register
FIQ Interrupt Pending Register
Highest Priority Register
Interrupt Priority Register
All Other Qualified Interrupt Bits
32
32
33
33
FIQ Interrupt to Processor
IRQ Interrupt to Processor
Peripheral Priority Processor

实验五 定时器

[实验目的]

√ 学习 Operating system timer (OS timer)的使用
√ 学习如何通过配置相关的寄存器来产生定时中断。

[实验原理]

1. 程序介绍

本程序以第一章的 Boot 程序作为引导程序，然后调用关于 OS Timer 提供的 4 个定时器中断程序。将程序烧入开发板后，八段数码管就会显示 1234，当按 1234 中某一键时，就会产生定时中断，执行定时中断服务子程序，只显示相应键的数码管，执行完后又恢复原来的状态。

2. 概念理解

OS TIMER

OS Timer 提供了四个定时器，每个定时器对应一个匹配寄存器，并且为 4 个匹配寄存器提供一个以 3.68MHZ 计时的参考计数器。这个参考计数器是自动递增的，参考计数器的初始值是在寄存器 OSCR 中设置,它是一个 32 位的计数器,然后通过在 4 个匹配寄存器(OSMR0，OSMR1,OSMR2,OSMR3）中设值，当计数器的数值和任何一个匹配寄存器中数值相同时，便可以产生相应的中断。产生中断的前提还包括在寄存器 OIER 相应的位设 1 以使得相应的定时器可用，当参考计数器递增到匹配寄存器相同时，便可自动在寄存器 OSSR 相应的位上设 1 标记相应的匹配已发生，然后 OSSR 中被标示 1 的位会被发送到中断控制器从而引发中断。

这里需要强调一点，通过 OS Timer 可以提供 4 个计时器中断，就算以上前提都发生了，并不代表一定会执行中断，中断是否会被执行，首先需要在 CPSR 里开启中断（IRQ 或 FIQ），还要看中断控制器里的有没有屏蔽由 OS Timer 产生的中断，若屏蔽了，就算参考计数器递增到 OSMR 的值也不会执行中断,但由于中断计数器里寄存器 Interrupt Controller Pending Register(ICPR)保存了当前激活的中断事件，它不受中断屏蔽寄 存器（ICMP）的影响，所

以可以通过查询 ICPR 得知发生了什么中断事件。

3. 寄存器分析与配置

OS TIMER 寄存器

OS Timer Count Register (OSCR)

这是一个 32 位的寄存器，在 OSCR 里设置的数值会被传送的参考计数器中，计数器会在每个 3.6864MHZ 时钟信号的上升沿递增。

OS Timer Match Register 0-3 (OSMRx)

OS Timer 包括 4 个 Match Register，每个这样的寄存器都是 32 位的，在该寄存器设置的数值，会在每个 3.6864MHZ 时钟信号的上升沿与 OSCR 进行比较。

OS Timer Interrupt Enable Register（OIER）

这个寄存器只包含 4 个位，每个位对应一个匹配事件发生后，是否在 OSSR 寄存器中设置相应的状态位。

OS Timer Status Register (OSSR)

这个寄存器只包含 4 个位,每个位被读出为 1 时代表相应的 OSMR 与 OSCR 匹配事件发生，这个位会发送到中断控制器中，从而引发中断事件。当写入 1 到相应的位上则会清除该状态位。

OS Timer Watchdog Match Enable Register (OWER)

这个寄存器只包含一个位，用这个位来代表是否将 OSMR3 来作为 Watch Dog 来使用。

定时器使用

由于 OS Timer 有 4 个定时器，所以这里假设只使用 4 号定时器，定义以下符号。

```
osTimer_OSMR3 EQU  0x40a0000c
osTimer_OSCR  EQU  0x40a00010
osTimer_OSSR  EQU  0x40a00014
osTimer_OIER  EQU  0x40a0001C
int_ICPR      EQU  0x40d00010
```

置定时时间，配置寄存器 OSCR，OSMR3

```
ldr r1,=osTimer_OSCR
```

```
ldr r2,=osTimer_OSMR3
ldr r0,[r1]
add r0,r0,#0x100000
str r0,[r2]
```

以上代码通过读取 OSCR 寄存器里的值，并且加上相应的定时时间（0x100000），并将相加后的结果写入寄存器 OSMR3 里。

2） 开启 4 号定时器中断功能，配置寄存器 OIER，OSSR

```
ldr r1,=osTimer_OIER
mov r0,#0x8
str r0,[r1]

ldr r10,=osTimer_OSSR
ldr r0,[r10]
str r0,[r10]
```

寄存器 OIER[3]代表 4 号定时器中断允许位，在这一位设 1 则表示允许由 4 号定时器发出中断请求。若 OSSR[3]上出现 1，则表示 OSCR 已经增加到与 OSMR3 一致，并向中断控制器发出 4 号定时器的中断请求。向 OSSR 回写的目的是清除上一次由 OSMR3 发出的中断请求，重新开始计时。

3）4 号定时器中断事件是否发生。

```
ldr r11,=int_ICPR
ldr r4,[r11]
mov r5,#0x20000000
tst r4,r5
```

激活的中断事件可以在 ICPR 中查询得到，为 1 表示事件激活，ICPR[29]代表 4 号定时器的中断请求状态。

关闭 4 号定时中断

```
ldr r11,=osTimer_OIER
ldr r4,[r11]
and r4,r4,#07
str r4,[r11]
```

```
    ldr r11,=osTimer_OSSR
    ldr r4,[r11]
    orr r4,r4,#0x80
    str r4,[r11]
```

当中断发生后便执行相应的处理操作，所以处理完后，必须清除上一次的定时中断请求状态，即清除 ICPR[29]，由于 ICPR 是只读的寄存器，所以不能直接对 ICPR 设置，只需向 OSSR[3]设 1，便能清除 ICPR[29]。

5．例子程序分析

在 timer.c 文件中，程序进入定时器中断等待状态，在这之前，系统已经进行了相关的初始化和定时器中断的开启。我们利用 1234 这四个按键来确定使用不同的定时器，从而引发不同的中断事件。按键的处理有五种情况：

情况 1：当没有按键时，系统处于等待定时中断的状态。

情况 2：当按下 1 键时，OIER 使能定时器 0，并且设置定时为 0x800000，当到达定时时间后就会产生定时中断，跳到 IRQ_Handler 处执行中断服务例程。

情况 3：当按下 2 键时，OIER 使能定时器 1，并且设置定时为 0x800000，当到达定时时间后就会产生定时中断，跳到 IRQ_Handler 处执行中断服务例程。

情况 4：当按下 3 键时，OIER 使能定时器 2，并且设置定时为 0x800000，当到达定时时间后就会产生定时中断，跳到 IRQ_Handler 处执行中断服务例程。

情况 5：当按下 4 键时，OIER 使能定时器 3，并且设置定时为 0x800000，当到达定时时间后就会产生定时中断，跳到 IRQ_Handler 处执行中断服务例程。

当每次执行中断服务例程，从中断服务例程返回之前，都要清除定时器的中断标志位。并且重新开启定时器中断。

```
    ;clean the timer flag
    LDR R3,=OSSR
    MOV R0,#0xf
    STR R0,[R3]

    ;enable the timer interrupt
    LDR R0,=ICMR
    MOV R1,#0x3c000000
    STR R1,[R0]
```

程序流程图：

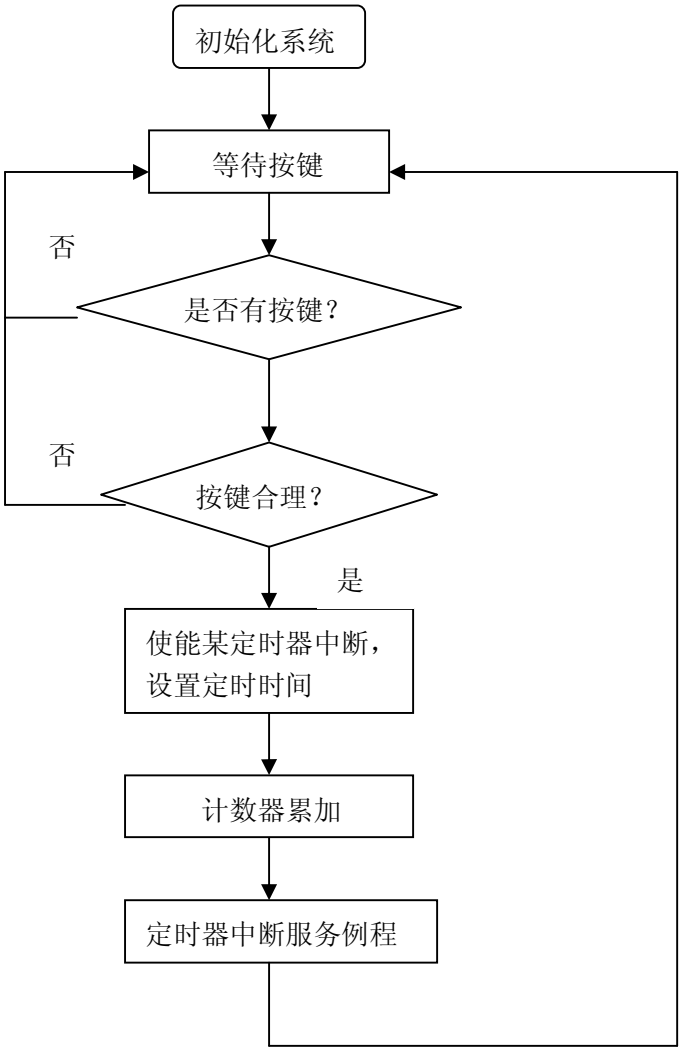


图 5.1 整体流程图

[实验内容]

1)分析代码

结合以上说明,对本实验所提供的汇编源代码进行分析,深入理解针对具体的硬件实现,软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图图 5.2 所示：

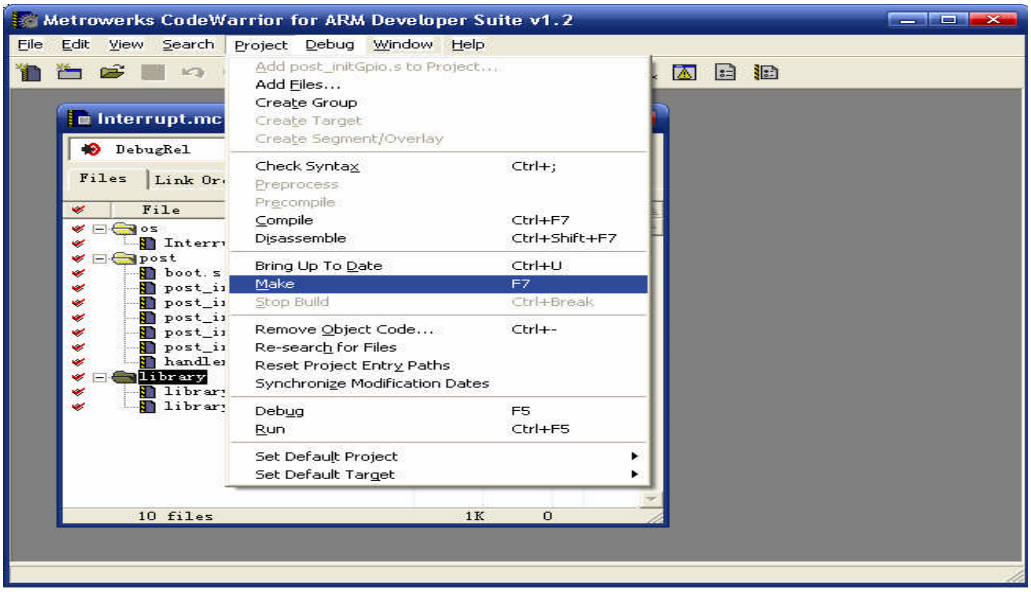


图 5.2 编译、连接生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1. PXA270 提供多少个定时器，哪个定时器可作为看门狗使用。
2. 简要阐述 OS Timer 的 5 个寄存器的作用，并简述如何操作 OS Timer。

实验六 串口传输

[实验目的]

√ 了解普通串口的工作机制
√ 学习对 PXA270GPIO 的控制
√ 学习为串口写驱动程序
√ 学习使用串口，对数据进行发送和接收

[实验原理]

1．程序介绍

本章例子是 PXA270 的串口传输程序，并以两种方式实现：程序查询状态寄存器和中断处理。本章例子根据 PXA270 开发板的串口硬件连接，不加入 Moden 传输协议，紧紧利用 UART 引脚 TXD 和 RXD 进行数据接收和发送。

2．实现方法

本章实验目标：通过配置 FFUART（串口 1），将字符显示在 PC 机上超级终端上，并且将键盘输入内容回显在超级终端上。实现步骤：

配置 GPIO 寄存器，实现 CPU 引脚 34，39 接收发送功能。
配置寄存器 POWER Manager Sleep Status (PSSR)。
配置全功能 UART 的寄存器。
编写接收字符和输出字符函数。
编写 FFUART 的中断服务例程。

3．原理概述

1）UART 操作原理

PXA270 处理器有四个 UART，分别是：全功能 UART（FFUART），蓝牙 UART（BTUART），

标准 UART（STUART），硬件 UART（HWUART）。UART 即 universal asynchronous receiver/transmitter 的简写。每个 UART 能将从 RXD 端接收的串行数据转变为并行的数据，并且能够将来自处理器的并行数据转化串行数据，然后通过 TXD 端发送出去。根据 UART 是否在 FIFO 模式下执行，发送和接收的数据会有选择的锁存在发送/接收 FIFO。例如，当 UART 在接收数据时，来自 RXD 端的数据首先会经过接收移位寄存器，然后组织成一个字节的数据，如果运行在 FIFO 模式，数据会首先锁存在接收 FIFO 里，同时接收缓冲寄存器 RBR 会保存 FIFO 第一字节单元数据，FIFO 的内容可以通过连续读取 RBR 获得，每读写完一次后，FIFO 第一字节单元数据会被移出。当 UART 收到来自总线的并行数据时，数据首先进入发送缓冲寄存器 THR，如果工作在 FIFO 模式，数据会再被锁存在发送 FIFO，最后才被送入发送移位寄存器，将并行数据以逐位方式在 TXD 端发送出去，每次向 THR 写入的数据（有效数据最长为 8 位)会被送入 FIFO，只有 FIFO 的第一字节单元会被送入发送移位寄存器里，并且在 FIFO 里还未发送的数据会逐渐上移到第一字节单元。无论是接收还是发送，当运行在 non—FIFO 方式时，数据不会被锁存在 FIFO，而只会被锁存在寄存器 RBR 或 THR，可以简单认为在 non—FIFO 时，RBR 和 THR 分别与接收移位寄存器和发送移位寄存器直接相连。

当需要对数据接收或发送时，应该首先根据 UART 的状态标志来决定是否写入 RBR 或从 THR 读出，每个 UART 都有一个 **Line Status Register**（LSR），它提供了传输状态信息，通过读取响应的位便可以得知当前情况是否适合发送。这里有两种方式实现控制 UART 的发送和接收：

⑴ 通过程序不断的轮询 UART 的状态寄存器 LSR 来决定是否发送和接收数据。

⑵ 以中断方式来实现发送和接收数据，此时可以通过 UART 的当前状态来触发中断。

即利用接收或发送事件请求令中断发生，然后在中断服务例程里实现发送和接收。

对于第二种方式，还需要通过读取 UART 的 **Interrupt Identification Register**（IIR），这是因为对于每个 UART 来说，可以引发中断发生的中断源有五种类型：

表 6.1 UART 中断源

优先级	中断源
1（最高）	Receiver Line Status:：接收到的数据帧出现错误时发出中断请求，该数据帧的出错情况会反映在寄存器 LSR 里。
2	Received Data is available：在 FIFO 模式，如果在 FIFO 里接收到的数据数量到达设定的接收触发水平（在 FCR[ITL] 里设定），则会发出中断请求；如果在 non-FIFO 模式，只要 RBR 里接收到数据时，就会发出中断请求。
3	Character Timeout Indication occurred：该中断源只发生在 FIFO 模式，当接收 FIFO 锁存了最近接收到的数据，但由于数据量还没有到达接收触发水平，Received Data is available 中断没有被请求，所以数据一直锁存在 FIFO，当 4 个连续的字符周期没有发送数据时就会发出中断请求
4	Transmitter requests data：该中断事件可以在以下情况

	发生，在 FIFO 模式，发送 FIFO 至少有一半空间为空，或在 non-FIFO 模式，THR 的数据已经发送出去，此时为空。
5（最低）	Modem Status：一个或多个 MODEN 输入信号（nCTS，nDSR，nDCD，nRI）状态改变时发生

对于上表任何一种中断请求，可通过 UART 的 **Interrupt Enable Register**（IER）屏蔽中断请求。假设 FFUART 其中一种中断请求发生，该中断请求会发送到处理器的中断控制器，中断控制器的 **Interrupt Controller Pending Register** (ICPR)的 ICPR[22]就会标示 1，表示 FFUART 发出中断请求，只要 FFUART 中断请求被容许，程序执行完当前指令后就会跳转到响应的中断服务例程，在中断服务例程里需要再次读取 FFUART 的 IIR[0:3]确认发出中断的请求是上表的哪一种中断请求，然后执行响应的处理。

2）波特率产生器

每个 UART 包含一个可编程的波特率发生器，它采用 14.7456MHz 作为固定的输入时钟，并且可以对它以 1 至（2^{16}-1）分频，波特率可以通过以下公式计算：

$$BaudRate = \frac{14.7456\ \text{MHz}}{(16 x Divisor)}$$

Divisor 的取值可以是 1 至（2^{16}-1），该值是通过在寄存器 **Divisor Latch Registers** (DLL and DLH)中设置，DLL 和 DLH 都是 32 位的寄存器，但只有低 8 位可以使用，所以 DLH[0:7]和 DLL[0:7]就组成了一个 16 位的分频器，DLH[0:7]为分频器的高 8 位，DLL[0:7]为分频器的低 8 位。以下给出部分波特率与分频器对应表：

表 6.2　波特率与分频器对应表

BaudRate(bps)	Divisor	DLH	DLL
9600	96	0x0	0x60
38400	24	0x0	0x18
57600	16	0x0	0x10
115200	8	0x0	0x8

3）数据帧格式

每一个数据帧有 7 到 12 位字长，字长是可以程控的，这依赖于程序对数据帧的格式控制。一个完整的数据帧包含四个部分：开始位，有效数据位，奇偶校验位，停止位，各部分的顺序是固定的。如下图所示：

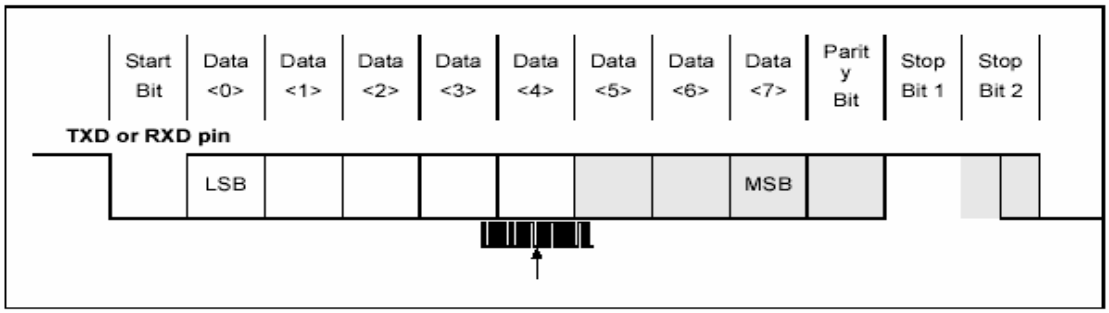


图 6.1 数据帧格式

实质上，一个完整的数据帧可以不包括奇偶校验位，所以通过 UART 发送出去的数据帧至少包含开始位，有效数据位，停止位。

表 6.3 数据帧格式描述表

数据帧的各部分	位长（bit）	描述
开始位	1	代表一个数据帧的开始，它以电平从高到低的形式出现。
数据位	5－8	它是写入输出缓冲寄存器 THR 里的值或从接收缓冲寄存器 RBR 里读出的值。LSB 代表数据位最低位，MSB 代表数据位的高位，如果接收的数据位是被配置成少于 8 位长，则在 RBR 里，该数据位是右对齐的，不足 8 位的地方会被填充 0
奇偶校验位	1 或 0	如果是偶校验，这一位会被设置 1，并且数据位有奇数个 1，如果是奇校验，数据位则有偶数个 1
停止位	1 或 1．5 或 2	表示一个数据帧的结束，以高电平显示

对每个 UART，可以通过在寄存器 Line Control Register（LCR）里对数据帧进行格式设置。作为发送和接收的双方，波特率和数据帧格式必须一致。

EDR 平台串口原理

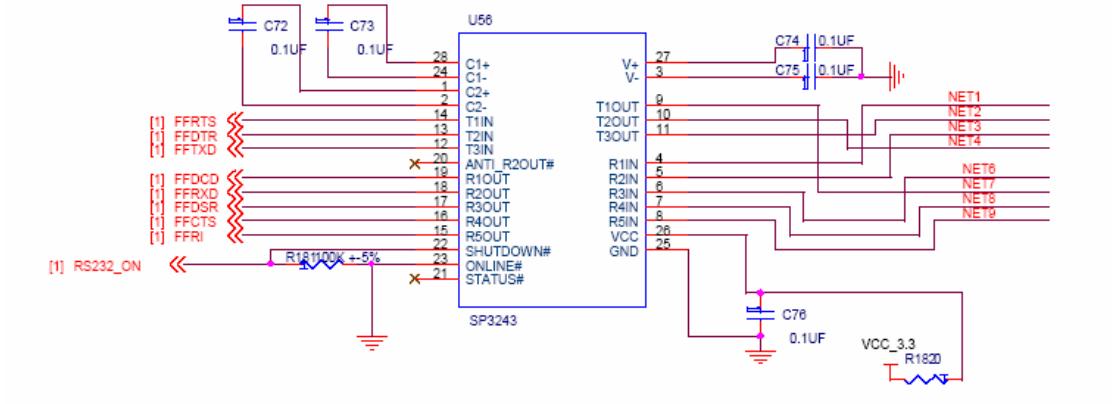


图 6.2 串口原理图 1

4. 寄存器使用与配置

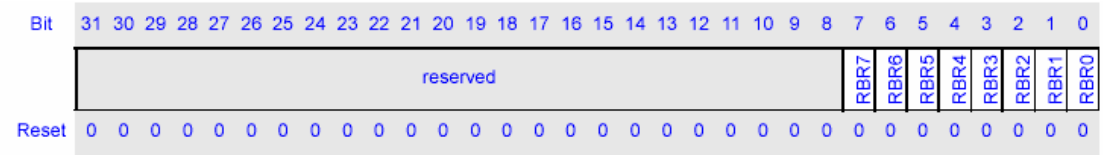
每个 UART 各自有一组寄存器,共 13 个，这些寄存器都是 32 位长，但都只使用低 8 位，并且，存在不同的寄存器使用相同地址，这就需要使用 DLAB 来区分不同的寄存器，如寄存器 FFRBR，FFTHR，FFDLL 地址同为 0x4010_0000，当 DLAB 为 0 时，访问 0x4010_0000 则表示访问 FFRBR，FFTHR，为 1 则表示访问 FFDLL，另外通过读写方式来区别地址相同的寄存器，如 FFRBR，FFTHR，当执行读操作时，访问的是 FFRBR，而执行写操作时，访问的是 FFTHR。下表以 FFUART 为例子，只罗列在实验中遇到的寄存器。

表 6.4 FFUART 寄存器表

地址	名称	DLAB	描述
0x4010_0000	FFRBR	0	Receive Buffer register (read only)
0x4010_0000	FFTHR	0	Transmit Holding register (write only)
0x4010_0004	FFIER	0	Interrupt Enable Register (read/write)
0x4010_0008	FFIIR	X	Interrupt Identification register (read only)
0x4010_0008	FFFCR	X	FIFo Control Register(write only)
0x4010_000C	FFLCR	X	Line Control Register(read/write)
0x4010_0010	FFMCR	X	Moden Control Register(read/write)
0x4010_0014	FFLSR	X	Line Status Register(read)
0x4010_0000	FFDLL	1	Divisor Latch Low register (read/write)
0x4010_0004	FFDLH	1	Divisor Latch High register (read/write)

上表的 DLAB 指的是 LCR[DLAB]，通过设置该位值可以访问地址相同的不同寄存器，打 X 的代表访问该寄存器不受到 DLAB 影响。以下对这些寄存器详细描述。

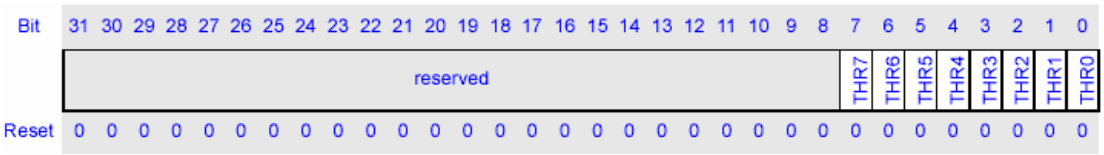
Receive Buffer register（RBR）



在 non-FIFO 模式，该寄存器锁存了来自接收移位寄存器的数据，接收移位寄存器是不能访问的，通过读取 FFRBR 可以得到刚接收到的数据，并且该数据会一直保持在 FFRBR 直到 FFRBR 被读取。当 FFRBR 接收到数据时，就会将 LSR[DR]设置 1，读取 FFRBR 后则会将 LSR[DR]清空。

在 FIFO 模式，FFRBR 锁存的是 FIFO 里最前字节单元的数据，每读取 FFRBR 一次，FIFO 里的最前字节单元的数据都会消失，并且被紧跟着的字节单元数据代替。只要 FIFO 里有数据还未读出，LSR[DR]则一直保持 1，直到 FIFO 里全部数据被读出后，LSR[DR]才被清空。

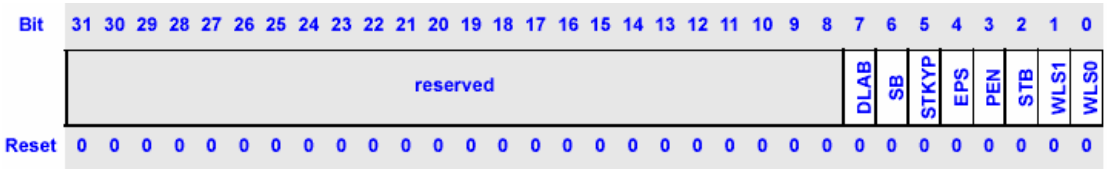
Transmit Holding register（THR）



在 non-FIFO 模式，FFTHR 保存了下个将要发送的字节数据，该数据会根据设定的数据帧格式组织成完整的数据帧，然后送入发送移位寄存器中。如果 FFTHR 当前为空（即没有数据需要发送），LSR[TDRQ]会被设置 1，当正在装载数据到 FFTHR 时，LSR[TDRQ]会被清空。

在 FIFO 模式，写入 FFTHR 的数据会被送入 FIFO 最高为空的位置。当 FIFO 至少一半空间为空时，LSR[TDRQ]会被设置 1，当 FIFO 有多于一半空间存有数据时，则 LSR[TDRQ]会被清空。

Line Control Register(LCR)

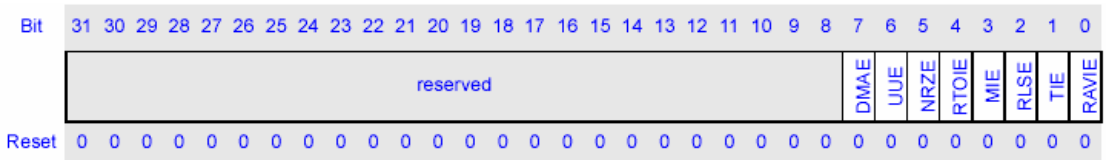


本寄存器用来设定数据帧格式，各位表示信息如表 6.5 所示：

化　　表 6.5 数据帧格式

Bits	Name	Description
7	DLAB	用来选择地址相同的不同寄存器 0—选择访问 FFRBR，FFTHR，FFIER 1—选择访问 FFDLL，FFDLH
6	SB	引发停止条件输出到接收方 0－对 TXD 端没有影响 1－强迫 TXD 端输出 0
5	STKYP	在奇偶位设置 EPS 的相反值。如果 PEN 为 0，STKYP 无效 0－对奇偶位没有影响 1－在奇偶位设定 EPS 的相反值
4	EPS	奇或偶校验选择，当 PEN 为 0 时，EPS 无效： 0—发送并检查奇校验位 1—发送并检查偶检验位
3	PEN	奇偶可用： 0－奇偶校验 1－没有奇偶校验
2	STB	停止位长度： 0—1 个停止位 1—2 个停止位，如果数据长度为 5 位，则停止位为 1-1.5
1:0	WLS[1:0]	数据长度： 00—5 位字长 01—6 位字长 10—7 位字长 11—8 位字长

Interrupt Enable Register（IER）



本寄存器主要用来设定 UART 的中断源是否可用

表 6.6 中断使能寄存器

Bits	Name	Description
7	DMAE	DMA 请求可用 0—DMA 请求不可用 1—DMA 请求可用
6	UUE	UART 可用 0—UART 不可用 1—UART 可用
5	NRZE	NRZ 编码可用 0—NRZ 编码不可用 1—NRZ 编码可用
4	RTOIE	Character Timeout Indication Interrupt 可用 0—Character Timeout Indication Interrupt 不可用 1—Character Timeout Indication Interrupt 可用
3	MIE	Modem Interrupt 可用 0—Modem Interrupt 不可用 1—Modem Interrupt 可用
2	RLSE	Receiver Line Status Interrupt 0—Receiver Line Status Interrupt 不可用 1—Receiver Line Status Interrupt 可用
1	TIE	Transmit Data request Interrupt 可用 0—Transmit Data request Interrupt 不可用 1—Transmit Data request Interrupt 可用
0	RAVIE	Receiver Data Available Interrupt Enable. 可用 0—Receiver Data Available Interrupt Enable 不可用 1—Receiver Data Available Interrupt Enable 可用

Moden Control Register（MCR）

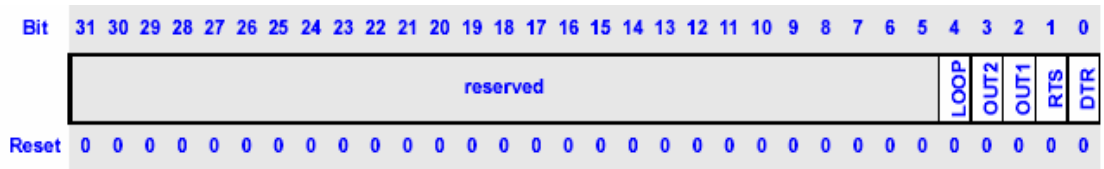
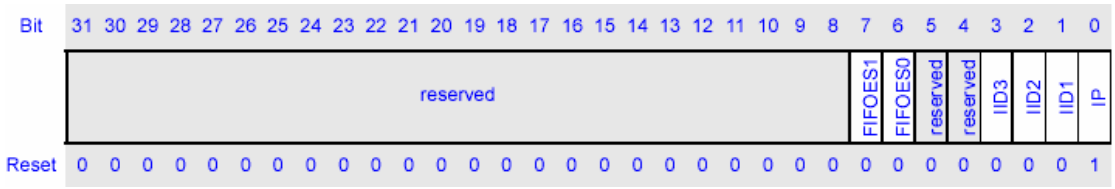


表 6.7 UART 总开关

Bits	Name	Description
3	OUT2	该位是 UART 中断的总开关，如果想使用 UART 的中断，此位必须设 1，这样 UART 的中断请求便可以发送中断控制器。

Interrupt Identification Register (IIR)



该寄存器是用来报告引发 UART 发生中断请求的中断源。

表 6.8 中断标志寄存器

Bits	Name	Description
7:6	FIFOES[1:0]	显示 UART 当前是否使用 FIFO 模式，若 FCR[TRFIFOE] 设置 1，则 FIFOES[1:0]为 11 00－使用了 non-FIFO 模式 11－使用了 FIFO 模式
3	IID3	Character Timeout Indication 中断是否发生 0－该中断没有发生 1－该中断发生
2:1	IID[2:1]	00 - Modem Status 中断发生 01 - Transmit FIFO request data 中断发生 10 - Received Data Available 中断发生 11 - Receive error 中断发生
0	IP	中断是否发生，当 UART 任何一种中断发生时，该位都会被置 1 0－没有中断发生 1－中断发生

FIFO Control Register (FCR)

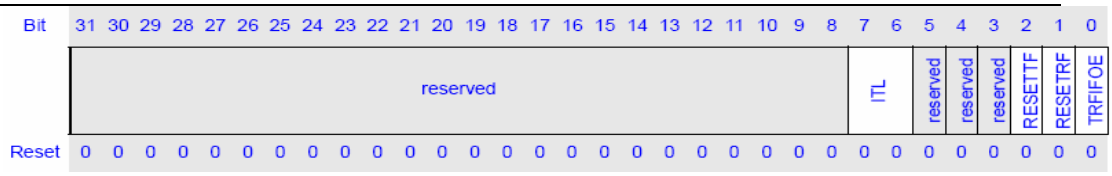
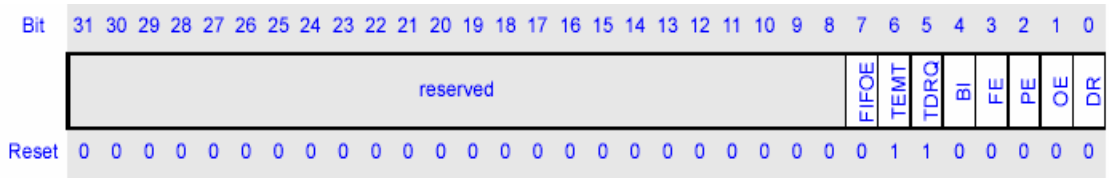


表 6.9 FIFO 控制寄存器

Bits	Name	Description
7	ITL	该位只在 FIFO 模式下有效，是 FIFO 里的接收字节数的触发水平，影响 Received Data Available 中断的发生，当 Received Data Available 中断可用时，如果锁存在 FIFO 里的字节数大于等于 ITL，则 Received Data Available 中断就会发出中断请求。 00—1 个或一个以上的字节会引发中断 01—8 个或 8 个以上的字节会引发中断 10—16 个或 16 个以上的字节会引发中断 11—32 个或 32 个以上的字节会引发中断
2	RESETTF	该位设 1 表示清除发送 FIFO 里的内容
1	RESETRF	该位设 1 表示清除接收 FIFO 里的内容
0	TRFIFOE	该位用来控制发送/接收 FIFO 是否可用 0—FIFO 不可用 1—FIFO 可用

Line Status Register（LSR）



该寄存器提供当前 UART 的状态信息，下面只对程序中使用的状态信息进行描述

表 6.10 线性状态寄存器

Bits	Name	Description
5	TDRQ	该位表示当前 UART 的发送请求状态，在 non-FIFO 模式，如果在 THR 里的数据已经被发送到发送移位寄存器，则该位会被设置 1，表示允许新的发送请求。在 FIFO 模式，如果 FIFO 里的锁存的字节数不够 FIFO 的一半，则发出中断请求。同时，该位会产生 Transmit Data Request 请求。但字节数被装载到 THR 或 FIFO 锁存的字节数超过 FIFO 的一半时，该位便会清空

0	DR	该位表示是否有新的数据被接收，并且还没有被读出。在 non-FIFO 模式，新接收的完整数据被锁存在 RBR，此时该位被设置 1，但读取 RBR 后，RBR 的内容和 DR 都会被清空。在 FIFO 模式，如果 FIFO 内锁存了数据，DR 会一直保持高电平，直到 FIFO 的内容全部被读出。

Divisor Latch Register(DLL,DLH)

DLL

Bit	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7	6	5	4	3	2	1	0
	reserved	DLL7	DLL6	DLL5	DLL4	DLL3	DLL2	DLL1	DLL0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0

DLH

Bit	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7	6	5	4	3	2	1	0
	reserved	DLH15	DLH14	DLH13	DLH12	DLH11	DLH10	DLH9	DLH8
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0

这两个寄存器组成 16 位分频器，DLH 为高 8 位，DLL 为低 8 位。通过该分频器可以产生不同频率的波特率，详细可以看第三段的“**原理概述**”关于波特率发生器的内容。

5．串口使用程序设计

根据以上讲解，现在开始设计串口 1 的基本发送/接收功能。由于串口的接收和发送有两种实现方式：程序查询状态寄存器和中断处理。现在先对第一种方式进行设计。

实验一：程序查询状态寄存器

第一步，配置 GPIO，目的是使处理器的引脚 GP34，GP39 分别作为 FFUART 的 RXD 和 TXD 端。

GPDR1(0x40E0_0010)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PD63	PD62	PD61	PD60	PD59	PD58	PD57	PD56	PD55	PD54	PD53	PD52	PD51	PD50	PD49	PD48	PD47	PD46	PD45	PD44	PD43	PD42	PD41	PD40	PD39	PD38	PD37	PD36	PD35	PD34	PD33	PD32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

由于 GP34 要作为输入端，GP39 作为输出端，所以，GPDR1[PD34]设为 0，GPDR1[PD39]设为 1。

GAFR1_L(0x40E0_005c)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AF47		AF46		AF45		AF44		AF43		AF42		AF41		AF40		AF39		AF38		AF37		AF36		AF35		AF34		AF33		AF32	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

由于 GP34 要作为 FFRXD 功能引脚来使用，GP39 要作为 FFTXD 功能引脚来使用，所以，需要将 AF34 设为 01，AF39 设为 10。

可得出 GPDR1 和 GAFR1_L 配置：

```
GPDR1 |= 0x80;
GAFR1_L |= 0x8010;
```

第二步，配置寄存器 Power Manager Sleep Status Register (PSSR)，目的是将 CPU 的 GPIO 输入引脚可用。以下可用汇编语言来实现：

定义以下符号：

```
PSSR    EQU    0x40F00004
```

执行以下汇编指令：

```
ldr     r1, =PSSR
ldr     r2, [r1]
orr     r2, r2,#0x10
        str     r2, [r1]
```

第三步，配置 FFUART 寄存器，实现 FFUART 的发送接收功能。

定义以下类型和宏

```
typedef unsigned long  ulong;
#define FFUART_BASE        0x40100000
```

设置数据帧格式：8 位有效数据长度，无奇偶校验位，1 个停止位

```
#define FFLCR     (*((volatile ulong *)(FFUART_BASE+0x0C)))
          FFLCR = 0x00000003;
```

设置接收字节触发数，接收/发送 FIFO 可用，并且在使用 UART 前清空接收/发送 FIFO。

```
#define FFFCR     (*((volatile ulong *)(FFUART_BASE+0x08)))
    FFFCR = 0x00000007;
```

设置分频器，将发送/接收的频率设为 115200bps。

访问 FFDLL 和 FFDLH 时需要将 FFLCR[DLAB]设为 1，并且将设置完分频器后需要将 FFLCR[DLAB]设为 0，使得程序可以访问 FFRBR，FFTHR，FFIER。

```
FFLCR |= 0x00000080;  //将 DLAB 设为 0
          FFDLL = 0x8;
          FFLCR &= 0xFFFFFF7F;  //将 DLAB 设为 1
```

4) 关闭 FFUART 的所有中断源，并且将 FFUART 设为可用。

```
    FFIER = 0x00000040;
```

第四步 设计接收/发送函数

接收函数

通过查询寄存器 FFLSR[DR]的状态判断是否需要访问寄存器 FFRBR。

```
   int SerialInputByte(char *c)
{
                if((FFLSR & 0x00000001)==0)
{
                return 0;
                }
                else
{
                *c = FFRBR;
                return 1;
                }
}
```

发送函数

通过查询寄存器 FFLSR[TDRQ]的状态来判断是否适合发送数据。

```
void SerialOutputByte(const char c)
{
    while ((FFLSR & 0x00000020) == 0 );
    FFTHR = ((ulong)c & 0xFF);

    if (c=='\n') SerialOutputByte('\r');
}
```

实验二：中断处理

本实验采用 Receiver Data Available 中断来接收数据。

第一步 打开 IRQ 中断控制

```
mrs  r1, CPSR
            and  r1, r1,#0x3f
            msr  CPSR_c, r1
```

第二步 设置中断控制器的中断屏蔽寄存器 ICMR（0x40D0_0004）。

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	IM31	IM30	IM29	IM28	IM27	IM26	IM25	IM24	IM23	IM22	IM21	IM20	IM19	IM18	IM17	reserved	reserved	IM14	IM13	IM12	IM11	IM10	IM9	IM8	reserved							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	?	?	?	?	?	?	?	?

ICMR[IM22]为 FFUART 的中断屏蔽位，将该位设 1，中断控制器可以接收 FFUART 发出的中断请求。

```
ICMR          EQU         0x40d00004
init_ICMR     EQU         0x00400000
          ldr  r1, =ICMR
          ldr  r2, =init_ICMR
                    str  r2,[r1]
```

第三步 设置 UART 寄存器

大部分的寄存器与实验一是相同，这里只修改 FFIER，FFMCR

```
FFIER = 0x00000041;    //开启 Received Data Available 中断
```

```
FFMCR  = 0x08;        //开启 FFUART 中断开关。
```

第四步 设置 IRQ 中断服务例程

由于中断屏蔽寄存器 ICMP 只开启 FFUART 中断，另外，FFUART 只开启 Received Data Available 中断，所以，当中断发生时，必然是由 FFUART 的 Received Data Available 中断引起的。

第五步 设计中断服务例程

在中断向量表的 0x18 的位置写上跳转到 IRQ 中断服务例程的指令。

```
B IRQHandler
```

2）中断服务例程

```
__irq void IRQHandler(void)
{
                        char newchar;
                        newchar = FFRBR;
                        FFTHR  = newchar;
}
```

当接收数据时，处理器会跳转到中断服务例程执行，所以程序无需查询 FFUART 的状态来判断是否需要接收数据。

[实验内容]

1）读者可以根据前几章所讲述的办法对程序进行编译与下载，由于本章例子程序采用汇编与 C 语言的混合编程方式，所以需要设置映像的 first 属性将中断向量表的所在 AREA 的开始位置固定在 0x0：－first boot.o(boot)，设置方法可以参照前几章所述。

2）设置超级终端

将程序下载到 FLASH 后，通过串口线将开发板上串口 1 与 PC 的串口进行连接，跟据刚才对 FFUART 的设置，现在对 PC 上的超级终端进行设置，接发双方设置内容必须一致。

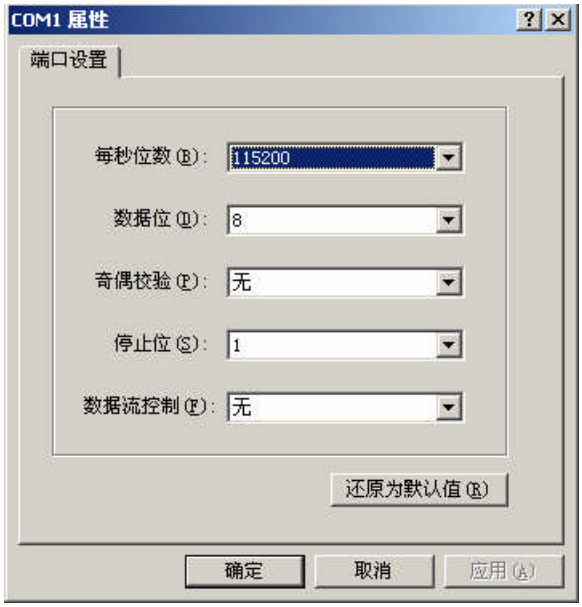


图 6.3 超级终端的端口设置

3）观察代码执行情况

[习题与思考题]

1. PXA270 内部提供多少个 UART，功能有什么区别。

2. 若需要设置 UART 的波特率为 19200，应如何设置分频器。

3. 实现简单 printf 函数，实现字符格式（%c），整数格式（%d），字符串格式（%s）的输出功能。

实验七 实时时钟

[实验目的]

√ 了解 RTC 应用及相关概念
√ 实际控制使用 Real time clock（RTC），显示系统时间

[实验原理]

1. 程序介绍

本章例子是一个使用 HZ 时钟来显示系统时间的程序，还利用实验六的串口传输实验的知识，通过串口打印系统的时间。首先设置超级终端的波特率为 38400，数据流控制设为无。打开超级终端，根据选择项进行操作，按 1 时，显示当前的系统时间。按 2 时，会对系统时间重设。

2. 概念理解

Real-timer clock （RTC）

RTC 是一个配置时钟的机制，通过从外部的晶振送入时钟信号到 MCPU，利用倍频或分频产生所需的时钟信号。可以通过配置 RTC 相关的寄存器，让 RTC 提供一个持续不断的频率，用来反映现实世界使用的时、分、秒时间。通常，RTC 还可以设计成产生一个 1HZ 输出（HZ 时钟名称的由来）。它的闹钟功能体现在当 RTC 增量到预定时间后便产生中断或唤醒事件。

为了能够产生系统时间，RTC 提供了一个 32 位的计数器 RTC Counter Register（RCNR），该计数器在系统复位后为 0，并在外部时钟源的信号上升沿到来时加 1，可 RDCR、RYCR 寄存器写入期望值，然后该计数器便开始递增。另外，通过在另一寄存器 RTC Alarm Register（RTAR）设置数值（也可以说是时间），当 RCNR 增加到 RTAR 时，便可产生中断。具体来说，当 RCNR 与 RTAR 匹配时，还需要以下条件满足时才能产生中断。首先在寄存器 RTC Status Register（RTSR）对中断的允许位必须设为 1，RTC 提供了两种可以产生中断的事件：HZ 中断和 RTC Alarm 中断。当 HZ 时钟的上升沿被检测到或 RCNR 和 RTAR 匹配相等时，RTSR 上相应的状态位就会标示 1，该位会被发送到中断控制器，在中断控制器的 Interrupt Controller

Pending Register (ICPR)上标示该中断事件已经激活，只要 CPSR 里的中断位被清除和 Interrupt Controller Mask Register(ICMR)没有对该中断事件屏蔽，中断便会产生，并且执行 IRQ 或 FIQ 中断处理。究竟该中断是以 IRQ 还是 FIQ 来处理，要看 Interrupt Controller Level Register(ICLR)是怎样对其归类。这里暂且不对中断控制器进行深入研究。

刚才所提到的 HZ 时钟就是通过外部晶振输入，分频后产生时钟源。它可以通过分频两个时钟源中的其中一个来产生，一个是将外部的 3.6864MHZ 晶体振荡器输入，然后再分频 112 产生 32.914kHz 信号。另一个将 32.768kHz 晶体振荡器输入，直接产生 32.768kHz 的信号。

3.寄存器分析与配置

RTC 寄存器

RTC Counter Register (RCNR)

RCNR 是一个 32 位的可读写寄存器，该寄存器在系统复位后为 0，并且是一个自动增长的计数器。

RTC Alarm Register (RTAR)

RTAR 是一个 32 位的可读写寄存器，对该寄存器设置值的目的是让寄存器 RCNR 和 RTAR 在 HZ 时钟的上升沿时比较，在相同时产生中断。

RTC Status Register (RTSR)

RTSR 包括了两个中断允许位和两个状态位。当 RTSR[HZE]为 1 时并且检测到 HZ 时钟信号上升沿到来时，RTSR[HZ]便被自动设为 1。当 RTSR[ALE]为 1 时并检测到 RCNR 和 RTAR 相同时，RTSR[AL]便被自动设为 1。RTSR[HZ]或 RTSR[AL]为 1 时都会发送到中断控制器。

RTC Trim Register (RTTR)

通过在该寄存器设置数值，可以调整 HZ 时钟的频率。复位后，该寄存器的数值为 0x7FFFF 可让 HZ 时钟准确输出 1HZ 时钟。

RTC Day Counter Register (RDCR)

RDCR 是一个计算系统时间的寄存器，除了包含小时、分钟、秒之外，还包括星期以及星期在每个月的位置。秒在每个 HZ 时钟的计时下增 1，增加到 59 后就会向分钟进位。分钟增加到 59 的同时也向小时进位。

RTC Year Counter Register (RYCR)

通过对寄存器赋值，可以设置系统的年、月、日时间。复位后，显示的时间是 2000 年 1 月 1 日。寄存器的值变化跟另外一个寄存器相关的，这就是 RDCR，当 RDCR 产生进位时，就会在 RYCR 上增 1，实现系统计时。

RTC 的使用

RTC 提供两种机制来产生中断：HZ 中断和 RTC Alarm 中断。Alarm 中断的使用方法与 OS Timer 的定时中断相同，所以这里紧紧以 HZ 中断的使用来举例。定义以下符号：

```
rtc_RTSR        EQU 0x40900008
int_ICPR        EQU 0x40d00010
```

设置 HZ 时钟的频率

RTC 在系统复位后被用作 HZ 时钟使用，该 HZ 时钟是通过分频外部时钟信号而准确的产生 1HZ 时钟输出，HZ 时钟的频率可以通过寄存器在 RTC Trim Register（RTTR）而调整。

开启 HZ 时钟中断

```
ldr r1,=rtc_RTSR
mov r0,#0xa
str r0,[r1]
```

程序开启了 HZ 时钟中断，没输出 1 个时钟信号（周期为 1HZ），就会产生一个中断请求。同时程序在 RTSR[1]设 1，目的是清除上一次时钟信号输出时所引起的中断请求。

3．查询中断是否发生

```
ldr r1,=int_ICPR
    ldr r0,[r1]
    mov r2,#0x40000000
    tst r0,r2
```

ICPR[30]代表是否有 HZ 时钟中断请求，为 1 则表示有请求。

4. 清除 HZ 时钟中断请求

```
ldr r11,=rtc_RTSR
    ldr r0,[r11]
    str r0,[r11]
```

通过回写 RTSR 来完成（实质是在 RTSR[1]写 1）清除上一次 HZ 时钟中断请求。这样 RTSR[30]就会被清除。

5．例子程序分析

系统运行到 dummyOs 函数时，就对 RTC 进行初始化，设置的起始日期为 2007 年 1 月 1 日，时间为 0 时 0 分 0 秒。

```
    RTSR = 0x0;                 //reset the status registers
    RCNR = 0x0;                 //reset the clock counter
     RYCR = 0xfae21;            //set the data 2007.1.1
    RDCR = 0x1e0000;            //set the time 00:00:00
```

接下来，运行 options()函数，显示了菜单选择项。我们根据选择项进行不同的函数处理。具体的流程请看图 7.1 的流程图。

6. 程序流程图

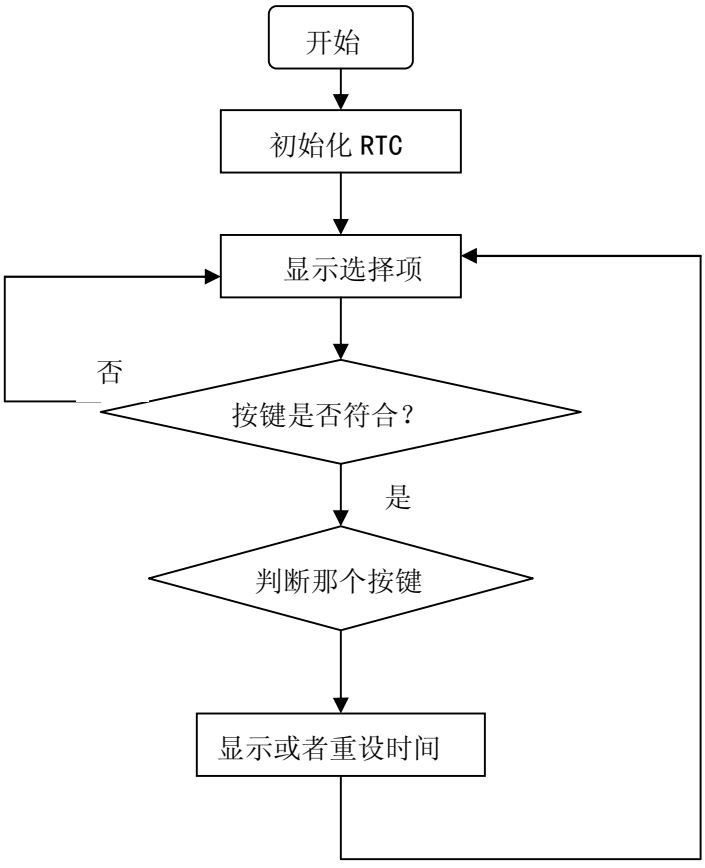


图 7.1 总体流程图

[实验内容]

1)分析代码

结合以上说明，对本实验所提供的汇编源代码进行分析，深入理解针对具体的硬件实现，软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图 7.2 所示：

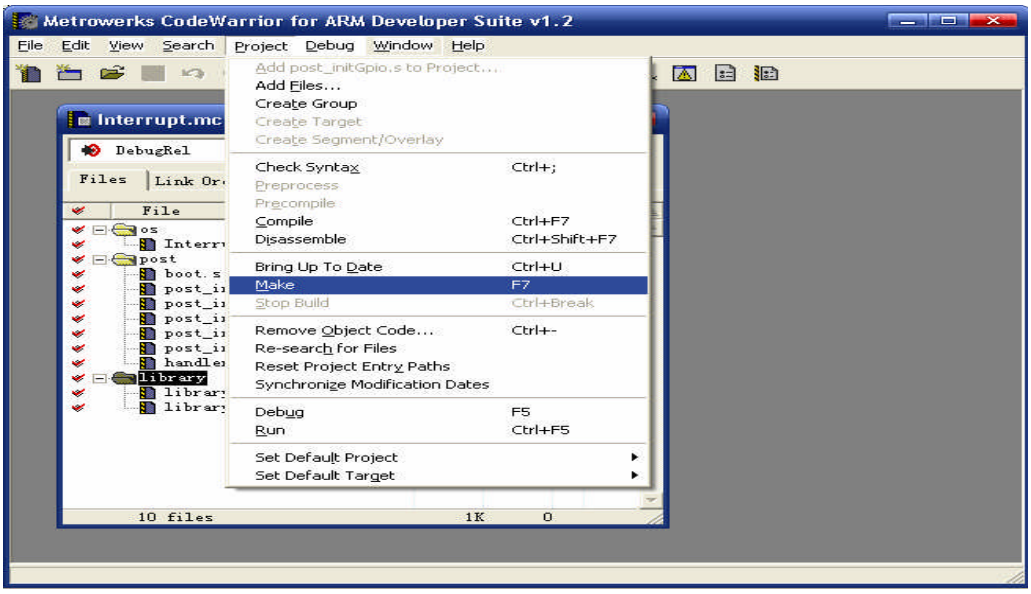


图 7.2 编译、连接生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1. 系统是怎样通过寄存器的进位来计算系统时间的？

2．简要阐述 RTC 的 RTSR、RDCR、RYCR 寄存器的作用。

3．如何实现产生定时中断一样的功能？

实验八 LCD 控制器

[实验目的]

通过配置 LCD 控制器，将字符和图像显示在显示屏上。

[实验原理]

1. 程序介绍

本章例子是一个使用 LCD 控制器的演示程序，结合了 LCD 控制器和 DMA 控制器的原理，能够将字符集里的字符显示在屏幕上。

2. 实验步骤

本章实验目标：学习配置 LCD 控制器，掌握 LCD 控制器的使用原理，掌握 Frame Buffer 与显示屏的映射关系，最终在屏幕上实现图像和字符的显示。

具体实验步骤如下：

配置 GPIO 寄存器，将与 LCD 连接的引脚定义为所需的功能引脚。

将帧描述符定义在 SDRAM 里，在 DMAC 被初始化后，供 DMAC 提取。

配置 LCD 控制器的各寄存器。

建立 LCD 屏幕上的每一象素与 FRAME BUFFER 对应位置的映射关系。将字符位图转换成字符矩阵数据，并且写入到 FRAME BUFFER 里。

原理概述

1) Frame Buffer

在还没涉及 LCD 控制器原理之前，我们先研究 Frame Buffer 与显示屏的对应关系。显示屏的整个显示区域，在系统内会有一段存储空间与之对应，通过改变该存储空间的内容，从而改变显示屏的内容，该存储空间被称为 Frame Buffer，或显存，显示屏上的每一点都必然与 Frame Buffer 里的某一位置对应，所以解决显示屏的显示问题，首先需要解决的是

Frame Buffer 的大小以及屏上的每一象素与 Frame Buffer 的映射关系。按照显示屏的性能或显示模式区分，显示屏可以以单色或彩色显示，单色用 1 位来表示（单色并不等于黑与白两种颜色，而只是说只能以两种颜色来表示，通常取允许范围内颜色对比度最大的两种颜色），彩色又分为 2 位色（4 种颜色），4 位色（16 种颜色），8 位色（256 种颜色），16 位色（65536 种颜色），24 为色（16777216 种颜色）这些色调代表整个屏幕所有象素的颜色取值范围，如：采用 8 位色/象素的显示模式，显示屏上能够出现的颜色种类最多只能有 2^8 种。究竟应该采取什么显示模式，首先必须根据显示屏的性能，然后再由显示的需要来决定。这些因素会影响 Frame buffer 空间的大小，因为 Frame buffer 空间的计算大小是以屏幕的大小和显示模式来决定的，另外还有另一影响因素，就是显示屏的单/双屏幕模式。如下图显示：

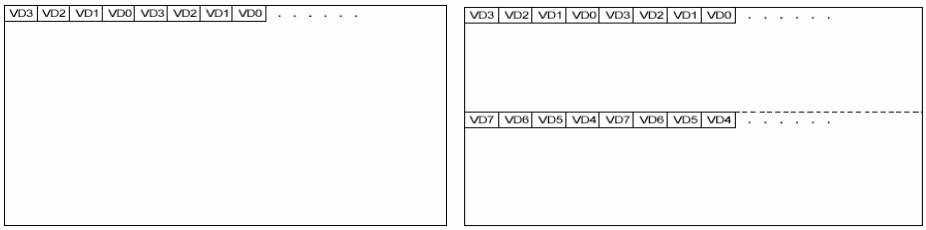


图 8.1 单屏幕与双屏幕模式

单屏幕模式代表屏幕的显示范围是整个屏幕，这种显示模式只需一个 Fame Buffer 来存储整个屏幕的显示内容，并且只需一个通道来将 Fame Buffer 内容传输到显示屏上（Frame Buffer 的内容可能需要被处理后再传输到显示屏）。双屏幕模式则将整个屏幕划分成两部分，上半部和下半部，它有别于将两个独立的显示屏组织成一个显示屏，单看上半部或下半部，它们的显示方式是与单屏幕的方式一致，并且上半部与下半部都是同时扫描，工作方式是独立的，同时这两部分都各自有 Frame Buffer，且他们的地址无需连续（这里指的是下半部的 Frame Buffer 的首地址无需紧跟在上半部 Frame Buffer 的地址末端），并且同时具有独立的两个通道将 Frame Buffer 的数据传输到显示屏。

有了上面整体的了解，现在再具体论述屏幕单个象素是如何在 Frame Buffer 里表示。由于 Frame Buffer 通常就是从内存空间分配所得，并且它是由连续的字节空间组成，而屏幕的显示操作总是从左到右逐点象素扫描，从上到下逐行扫描，直到扫描到右下角，然后再折返到左上角，而 Frame Buffer 里的数据则是按地址递增的顺序被提取，当 Frame Buffer 里的最后一个字节被提取后，再返回到 Frame Buffer 的首地址，所以屏幕同一行上相邻的两象素被映射到 Frame Buffer 里是连续的，某一行的最末象素与它下一行的首象素反映在 Frame Buffer 里也是连续的，并且屏幕上最左上角的象素对应 Frame Buffer 的第一单元空间，而屏幕上最右下角的象素对应 Frame Buffer 的最后一个单元空间。

计算机反映自然界的颜色是通过 RGB 值来表示的，如果要在屏幕某一点显示某种颜色，

则必须给出响应的RGB值,Frame Buffer为屏幕提供显示的内容,就必须能够从Frame Buffer里得到每一个象素的RGB值，方式有两种，可以是直接从Frame Buffer里得到或是间接得到。直接得到RGB值是指Frame Buffer里存放的象素颜色对应的RGB值，通过将该RGB值传输到显示屏上而令屏幕显示。间接得到RGB值是指Frame Buffer存放的并不是RGB值，而是调色板的索引值，而调色板里放的才是RGB值，通过Frame Buffer得到的索引值来提取调色板的RGB值，然后再发送到显示屏上。调色板的大小代表了显示屏最多能够显示的颜色范围。直接得到RGB的方式比间接得到RGB值的方式相对简单,因为在直接得到的方式下，Frame Buffer是由所有象素的RGB值或RGB值的部分位(RGR由Red，Green，Blue各8位组成，共24位，由于某些显示屏的数据线有限，只有16条数据线或更少，这时只能取Red，Green，Blue部分位与数据线对应)所组成。例如，16位/象素的模式下，Frame Buffer里的每个单元16位，每个单元代表一个象素的RGB值，如下图显示。

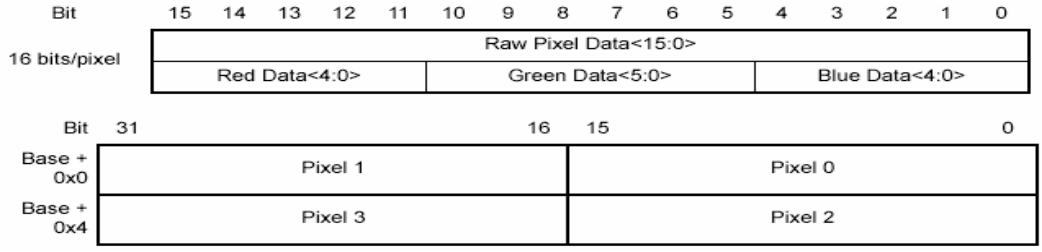


图 8.2　16位/象素Frame Buffer结构

对于间接得到RGB的方式，调色板每个单元空间可以是16位，这里假设调色板的总大小为512个字节，那么调色板就能存放256种颜色的RGB值，对于1位/象素，2位/象素，4位/象素，8位/象素的情况如下：

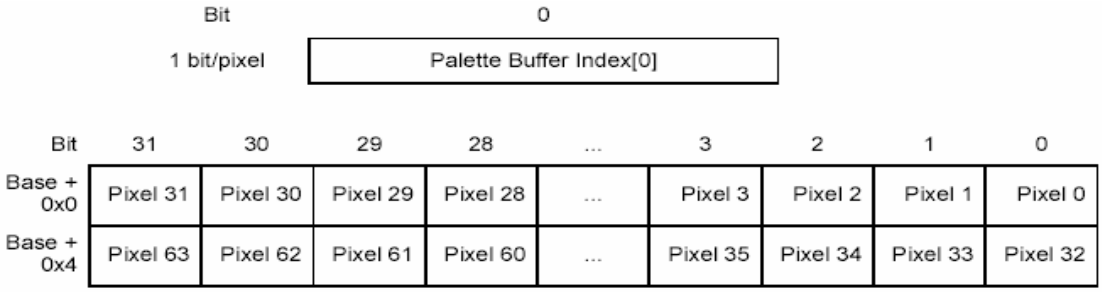


图 8.3　1位/象素的Frame Buffer结构

上图是1位/象素显示模式的Frame Buffer内部组织形式，对应于显示屏的每一点，Frame Buffer只需用1位来表示，

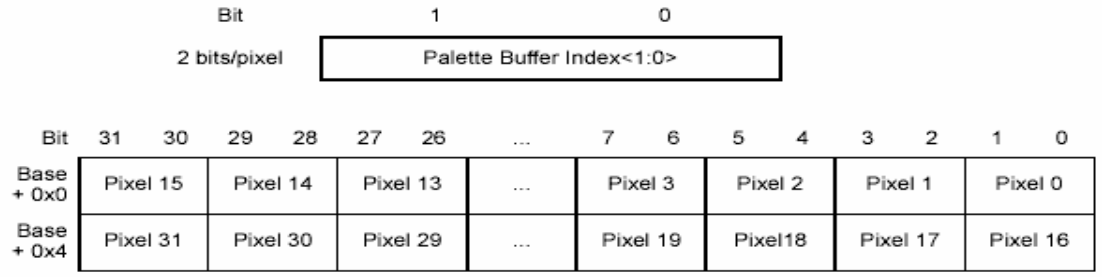


图 8.4　2 位/象素的 Frame Buffer 结构

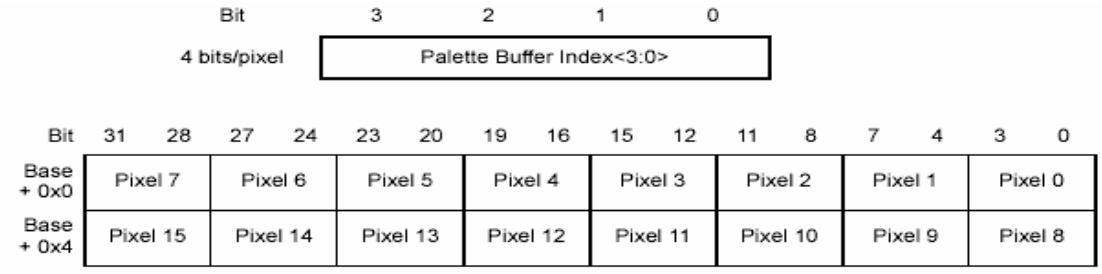


图 8.5　4 位/象素的 Frame Buffer 结构

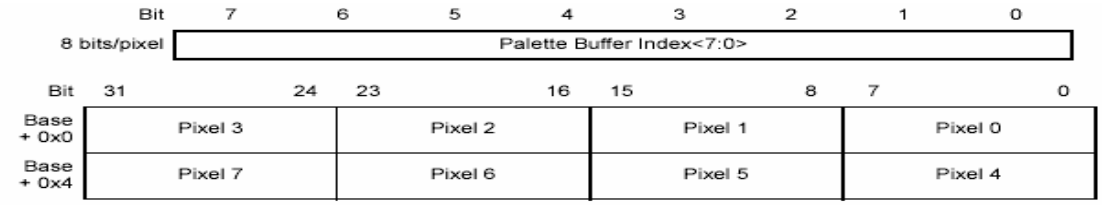


图 8.6　8 位/象素的 Frame Buffer 结构

有了以上的分析，可以得出以下的计算公式：

$$\text{FrameBufferSize} = \frac{(\text{显示屏长} \times \text{显示屏宽}) \times \text{象素位长}}{8}$$

显示屏长代表显示屏每一行的象素个数，显示屏宽代表显示屏的行数，象素位长代表在 Frame Buffer 里每个象素需要用多少位来表示。计算出来的 Frame Buffer 大小以字节为单位。如 640×480，16 位/象素，单屏幕模式，Frame buffer 大小为 614400 个字节。

2) LCD 控制器

在 Frame Buffer 与显示屏之间还需要一个中间件，该中间件负责从 Frame Buffer 里提

取数据，进行处理，并传输到显示屏上。PXA270 处理器内部集成 LCD 控制器（以下简称 LCDC），它提供了一个从 PXA270 处理器到 Passive（DTSN）或 Active（TFT）显示屏的接口，LCDC 的作用是将 Frame Buffer（可以理解为显存）里的数据传输到 LCDC 的内部，然后经过处理，输出数据到 LCD 的输入引脚上。

从图 6.7 可知，LCD 控制器由以下部分组成：LCD DMAC（本章提出的 DMAC 都是指集成在 LCDC 内部的 DMAC），输入/输出 FIFO，内部调色板，TMED 抖动引擎，寄存器组。LCDC 会因应所接的 LCD 类型，内部操作方式有所不同：

当接 Passive 显示屏（STN）时，并且显示模式为单色（1 位/象素）或彩色（2 位/象素，4 位/象素，8 位/象素）时，LCDC 必须首先初始化内部调色板（LCDC 内部的 DMAC 会将外部调色板的数据填充内部调色板，外部调色板是预先写于内存空间的调色板，大小与内部调色板一致）。然后 DMAC 将存储在 Frame Buffer 内的编码象素值传输到输入 FIFO 中，输入 FIFO 的数据会再次被提取出来，作为索引指针来提取内部调色板的数据（16 位），从内部调色板得到的数据会被传送到帧速率控制逻辑单元，该帧速率控制逻辑单元使用非持久调节能量发送算法来产生发送到 LCD 的象素数据，该象素数据会被锁存到输出 FIFO 里，然后再发送的 LCD 上。如果显示模式为 16 位/象素，则无需填充内部调色板，事实上内部调色板由于只能存放 256 种颜色的 RGB 值，明显不能满足 16 位/象素（因为 16 位/象素能够表示的颜色范围达 64K），因此从 Frame Buffer 里提取的每个象素值则直接为 RGB 值（Red：5bits，Green：6bits，Blue：5bits），16 位/象素的显示模式会与其他的显示模式的唯一区别是不使用内部调色板，所以数据从输入 FIFO 出来后就直接进入到帧速率控制逻辑单元。

当接 Active 显示屏（TFT）时，LCDC 内部的工作方式相对简单，此时，LCDC 无需加载数据到内部调色板，并且数据无需经过帧速率控制单元的处理，Frame Buffer 内的数据是 16 位的象素数据，通过 DMAC 传输到输入 FIFO 后，数据又立刻被传送到输出 FIFO。

下图是 LCDC 内部结构框架：

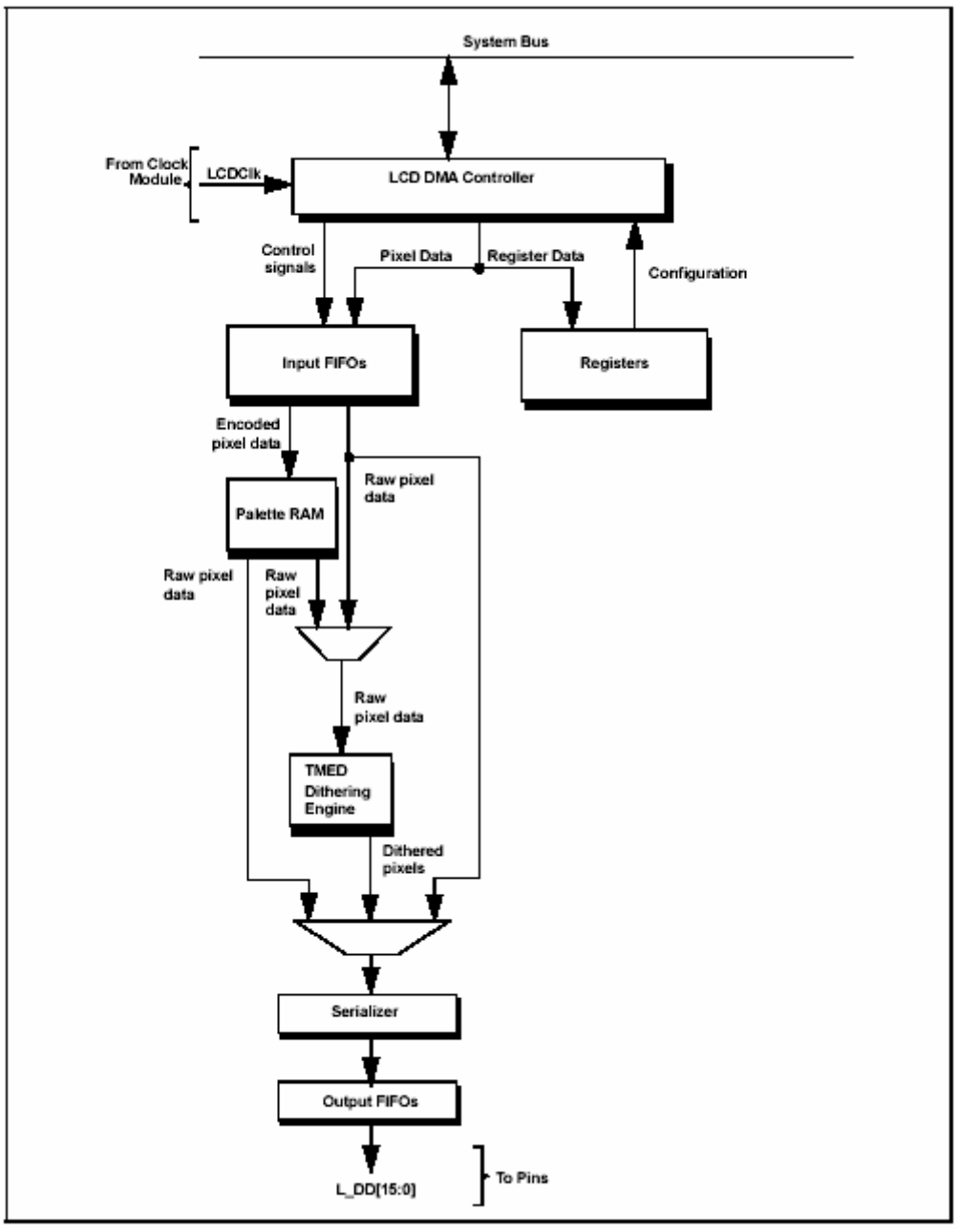


图 8.7 LCDC 内部结构框架

2. 1) DMAC

LCDC 内部有一个 DMAC（Direct Memory Access Controller），这个 DMAC 包括两个通

道，通道一用于传输外部调色板的数据到 LCDC 的内部调色板以及将 Frame Buffer 的数据传输到输入 FIFO 内。通道二则用在双行扫描模式，屏幕的上下两部分分别使用通道一和通道二来传送其对应的 Frame Buffer 数据，而外部调色板数据则只使用通道一来传输。在使用 DMAC 前，必须进行对其初始化，DMAC 的初始化方式有别于其他控制器，对它的初始化工作会留在稍后介绍。在 DMAC 被初始化后，它就会自动的从 Frame Buffer 里提取数据，每当输入 FIFO 有 32 字节的空间为空时，输入 FIFO 就会向 DMAC 发出服务请求，DMAC 就会从 Frame Buffer 里提取 32 字节的数据到输入 FIFO。

2．2）输入 FIFO

LCDC 内有两个输入 FIFO，每个 FIFO 由 128 个字节组成，也可以看成是由 16 个单元组成，每个单元 8 个字节。DMAC 的每次操作都会从 Frame Buffer 内将 32 字节数据传输到 FIFO，并且每单元（8 字节）的数据会按照在寄存器里（LCCR3[BPP]）设定的模式，以 1 位，2 位，4 位，8 位或 16 位进行分解，分解后的数据会独立地作为调色板的索引值。这两个 FIFO 分别与 DMAC 的两个通道相连接，所以，当运行在单屏幕模式下时，只有与 DMAC 的一号通道相连接的 FIFO 才会工作，令一个 FIFO 仅会在双屏幕模式下被使用。

2．3）调色板

LCDC 的内部调色板是由 512 个字节组成，每两个字节为 1 个单元，能够保存 256 种 16 位的颜色值，对于彩色，取红色的 5 位，绿色的 6 位，蓝色的 5 位组成；对于单色，虽然调色板仍然使用 16 位来存储 RGB 值，但仅用低 8 位来存储。

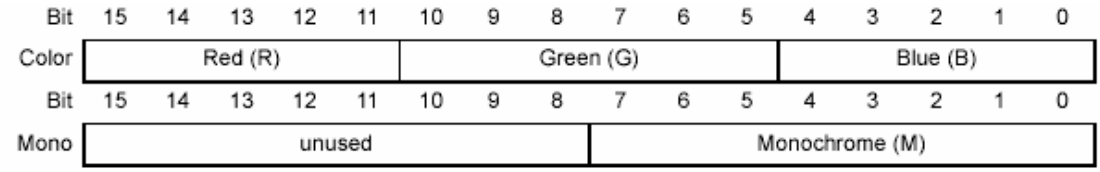


图 8.8　调色板的 16 位单元数据

LCDC 的内部调色板在使用前为空，所以如果 LCDC 在工作后，会用到内部调色板，则必须在 LCDC 使用前对内部调色板初始化，这里的初始化是指将 RGB 值写入到内部调色板内。用户并不需要将数据直接写入到内部调色板，写入内部调色板是交由 DMAC 来完成，但用户必须在内存里建立一个外部调色板，这个外部调色板的空间大小应该与内部调色板大小一致，但传输的数量并不是将完整的外部调色板传输到内部调色板，传输数量的依据是按照显示模式来决定的。

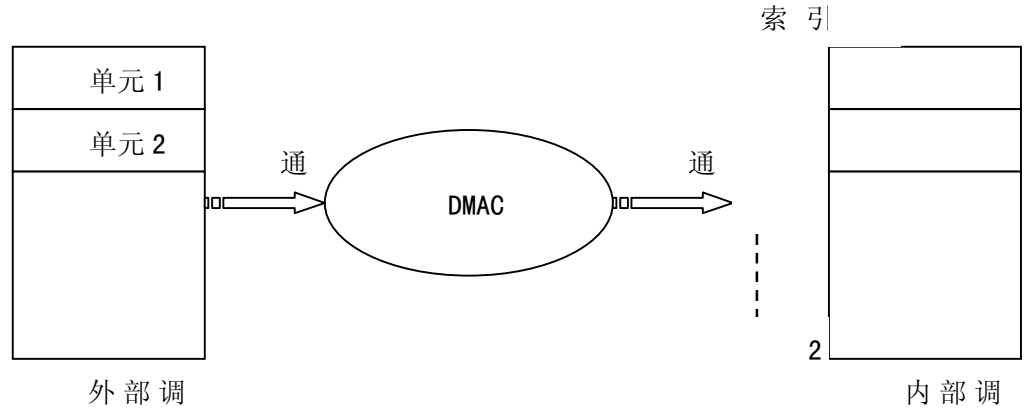


图 8.9　外部调色板/内部调色板

由于在 1 位/象素的显示模式下，仅能访问调色板的头 两个单元（每单元 16 位），调色板的其他空间都不能被访问，所以只需将外部调色板的两单元数据传输到内部调色板即可，如此类推，2 位/象素的显示模式下，则能访问内部调色板的头 4 个单元；4 位/象素的显示模式下，则能访问 16 个单元；8 位/象素的显示模式下，则能完全访问整个内部调色板，只有在这个模式下，才需要将整个外部调色板填充到内部调色板里。但由于 Frame Buffer 和外部调色板都是使用通道 1，DMAC 又是如何区分这两种数据以及决定传输的数量呢，这是通过在 LDCMD0[PAL]和 LDCMD0[LEN]设置的，对 LDCMD0 的描述会在稍后进行。

2．4）输出 FIFO

象素数据被发送到输出引脚之前会被锁存到输出 FIFO，同样，LCDC 内有两个输出 FIFO 应用与单屏幕或双屏幕显示模式。

2．5）引脚描述

LCDC 与 LCD 的通信引脚：数据线引脚 L_DD<15:0>；时钟信号引脚 L-PCLK，L-LCLK，L-FCLK，L-BIAS。

L_DD<15:0>用于传输象素数据，根据不同的显示模式 L_DD 会有所不同。

单/彩色显示	/双屏幕模式	Passive(STN)/Active(TFT)显示屏	屏幕位置	L_DD
单色		Passive	全屏	L_DD<3:0>
单色	单	Passive	全屏	L_DD<7:0>
单色	双	Passive	上半部	L_DD<3:0>
			下半部	L_DD<7:4>
彩色	单	Passive	全屏	L_DD<7:0>
彩色	双	Passive	上半部	L_DD<7:0>

			下半部	L_DD<15:8>
彩色	单	Active	全屏	L_DD<15:0>

表 8.1 各模式的数据线引脚分配

综合来说，决定 L_DD 的使用是通过设置 LCCR0[CMS]（决定单/彩色显示），LCCR0[SDS]（决定单/双屏幕模式），LCCR0[PAS]（决定 Passive/Active），LCCR0[DPD](决定单色显示时是使用 4/8 条数据线)。

时钟信号引脚

Pin	Description
L_PCLK	在 Passive 模式,只有当数据在数据线引脚（L_DD）上可用时，即象素数据正被写入到显示屏时被触发；在 Active 模式，象素时钟会不停的触发，作为其他时钟信号频率的基准时钟频率，数据的发送频率，水平同步信号频率，垂直同步信号频率都是以象素时钟频率为基准。
L_LCLK	在 Passive 模式，当 LCD 的某一行完整的象素数据被送到显示屏后，该信号就会确认。由于该信号的产生，显示屏的扫描点会水平折回，并且开始下一新行的扫描。在 Active 模式，它作为水平同步信号（HSYNC）。
L_FCLK	在 Passive 模式，当某一帧的数据被完整地发送到显示屏后，该信号就会确认。该信号确认后，显示屏的扫描点就会垂直折回，由于一帧的最后一行的行时钟信号和帧时钟信号是同时发生的，扫描点会折回到屏幕最左上角。在 Active 模式，它会作为垂直同步信号（VSYNC）。
L_BIAS	在 Passive 模式，L_BIAS 可用于周期性的切换电源与地，如果显示屏不能自动的切换，则需要使用 L_BIAS 提供一个周期信号，L_BAS 周期的长度可以通过 LCCR3[ACB] 决定。在 Active 模式，L_BIAS 的作用相当于 Passive 模式下 L_PCLK，当数据发送到 L_DD 时发出信号。

表 8.2 时钟信号引脚

4. 寄存器

LCD 控制器包括 4 个控制寄存器，10 个 DMA 寄存器，1 个状态寄存器和 256×2 字节的内部调色板。

Address	Name	Description
0x4400_0000	LCCR0	LCD controller control register 0
0x4400_0004	LCCR1	LCD controller control register 1
0x4400_0008	LCCR2	LCD controller control register 2
0x4400_000c	LCCR3	LCD controller control register 3
0x4400_0020	FBR0	DMA channel 0 frame branch register
0x4400_0024	FBR1	DMA channel 1 frame branch register
0x4400_0038	LCSR	LCD controller status register
0x4400_003C	LIIDR	LCD controller interrupt ID register
0x4400_0040	TRGBR	TMED RGB Seed Register
0x4400_0044	TCR	TMED Control Register

0x4400_0200	FDADR0	DMA channel 0 frame descriptor address register
0x4400_0204	FSADR0	DMA channel 0 frame source address register
0x4400_0208	FIDR0	DMA channel 0 frame ID register
0x4400_020C	LDCMD0	DMA channel 0 command register
0x4400_0210	FDADR1	DMA channel 1 frame descriptor address register
0x4400_0214	FSADR1	DMA channel 1 frame source address register
0x4400_0218	FIDR1	DMA channel 1 frame ID register
0x4400_021C	LDCMD1	DMA channel 1 command register

表 8.3 LCDC 寄存器

本实验仅使用以下寄存器：LCCR0，LCCR1，LCCR2，LCCR3，FDADR0，FSADR0，FIDR0，LDCMD0。这些寄存器会在稍后讲述。

5．LCD 显示程序设计

1）实验基本配置

实验使用的显示屏是 6.4 寸，分辨率是 640×480，TFT 类型。

显示配置：16 位/象素，单屏幕，不使用任何 LCDC 内部中断

2）设置 LCDC 与 LCD 接口引脚

由于使用的 LCD 是 TFT 显示屏，所以数据引脚为 L_DD<15:0>。

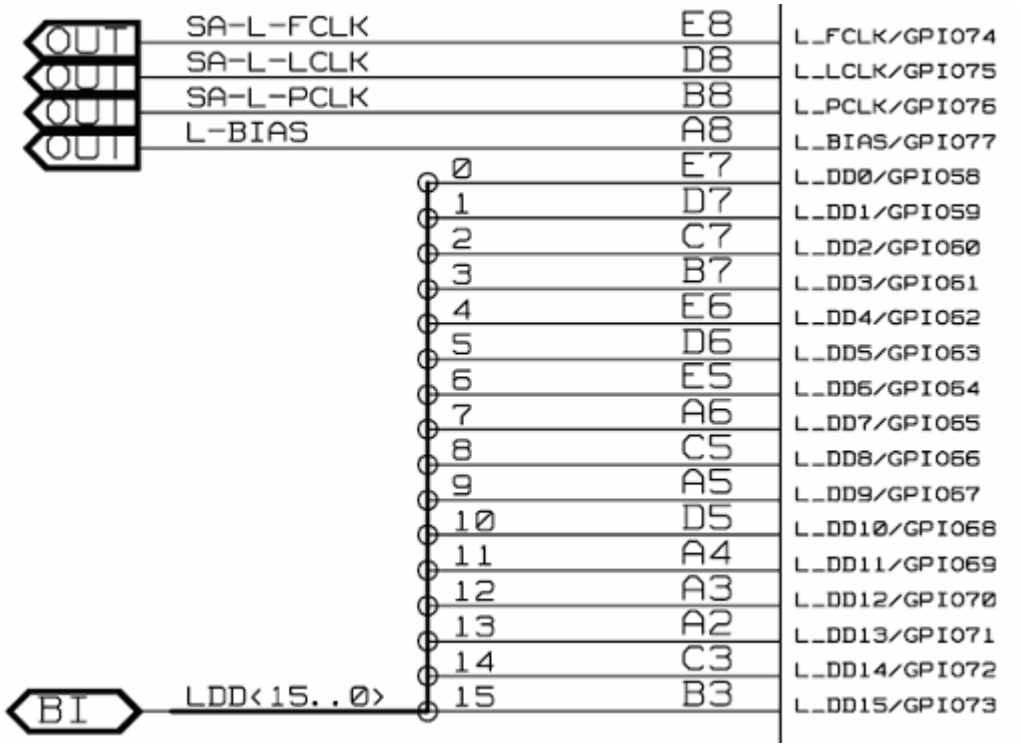


图 8.10　LCDC 引脚与 GPIO 引脚连接图

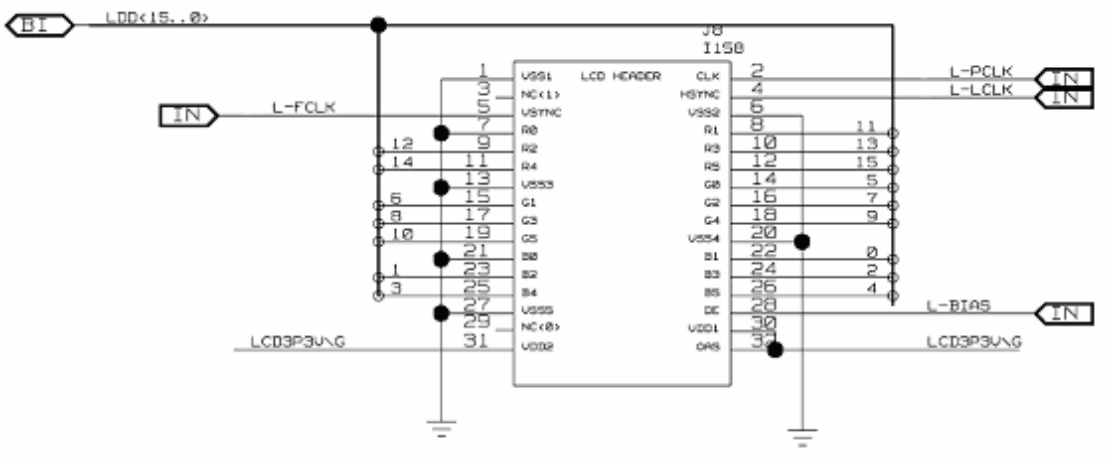


图 8.11　LCDC 接插件

从图 6.11 中，可以得知 16 跟数据线引脚的 RED，Green，Blue 的分配如下

R5	R4	R3	R2	R1	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1

R0 和 B0 接地。

以下各 LCDC 引脚与 GPIO 引脚的对照表

LCDC 引脚	引脚序号	引脚方向设置		引脚功能设置	
		Input/Output	GPDR 设置位置	功能值	GAFR 设置位置
L_DD<0>	GP58	Out	GPDR1[PD58]	10	GAFR1_U[AF58]
L_DD<1>	GP59	Out	GPDR1[PD59]	10	GAFR1_U[AF59]
L_DD<2>	GP60	Out	GPDR1[PD60]	10	GAFR1_U[AF60]
L_DD<3>	GP61	Out	GPDR1[PD61]	10	GAFR1_U[AF61]
L_DD<4>	GP62	Out	GPDR1[PD62]	10	GAFR1_U[AF62]
L_DD<5>	GP63	Out	GPDR1[PD63]	10	GAFR1_U[AF63]
L_DD<6>	GP64	Out	GPDR2[PD64]	10	GAFR2_L[AF64]
L_DD<7>	GP65	Out	GPDR2[PD65]	10	GAFR2_L[AF65]
L_DD<8>	GP66	Out	GPDR2[PD66]	10	GAFR2_L[AF66]
L_DD<9>	GP67	Out	GPDR2[PD67]	10	GAFR2_L[AF67]
L_DD<10>	GP68	Out	GPDR2[PD68]	10	GAFR2_L[AF68]
L_DD<11>	GP69	Out	GPDR2[PD69]	10	GAFR2_L[AF69]
L_DD<12>	GP70	Out	GPDR2[PD70]	10	GAFR2_L[AF70]
L_DD<13>	GP71	Out	GPDR2[PD71]	10	GAFR2_L[AF71]
L_DD<14>	GP72	Out	GPDR2[PD72]	10	GAFR2_L[AF72]
L_DD<15>	GP73	Out	GPDR2[PD73]	10	GAFR2_L[AF73]
L_FCLK	GP74	Out	GPDR2[PD74]	10	GAFR2_L[AF74]
L_LCL	GP75	Out	GPDR2[PD7	10	GAFR2_L[AF75]

K			5]		
L_PCLK	GP76	Out	GPDR2[PD76]	10	GAFR2_L[AF76]
L_BIAS	GP77	Out	GPDR2[PD77]	10	GAFR2_L[AF77]

表 8.4　LCDC 引脚与 GPIO 引脚的对照

对于 LCDC 每个引脚的方向和功能的设置都是类似的，以 L_DD<0>为例，由于该 LCD 数据线对应的引脚是 GP58，方向为输出，所以需要在 GPDR1[PD58]设置 1，并且需要将 GP58 的功能设置为 LCD 的 0 号数据线引脚，故需要在 GAFR1_U[AF58]里设为 10。下图是寄存器 GPDR1，GPDR2，GAFR1_U，GAFR2_L。

Physical Address 0x40E0_0010　GPDR1　System Integration Unit

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PD63	PD62	PD61	PD60	PD59	PD58	PD57	PD56	PD55	PD54	PD53	PD52	PD51	PD50	PD49	PD48	PD47	PD46	PD45	PD44	PD43	PD42	PD41	PD40	PD39	PD38	PD37	PD36	PD35	PD34	PD33	PD32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Physical Address 0x40E0_0014　GPDR2　System Integration Unit

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved											PD84	PD83	PD82	PD81	PD80	PD79	PD78	PD77	PD76	PD75	PD74	PD73	PD72	PD71	PD70	PD69	PD68	PD67	PD66	PD65	PD64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Physical Address 0x40E0_0060　GAFR1_U　System Integration Unit

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AF63		AF62		AF61		AF60		AF59		AF58		AF57		AF56		AF55		FA54		AF53		AF52		AF51		AF50		AF49		AF48	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Physical Address 0x40E0_0064　GAFR2_L　System Integration Unit

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AF79		AF78		AF77		AF76		AF75		AF74		AF73		AF72		AF71		AF70		AF69		AF68		AF67		AF66		AF65		AF64	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

各 GPIO 寄存器设置如下：

寄存器	地址	数值	备注
GPDR1	0x40E0_0010	0xFC00_0000	
GPDR2	0x40E0_0014	0x0000_7FFF	GPDR2[PD78]为 nCS2，需设置为 1
GAFR1_U	0x40E0_0060	0xAAA0_0000	

GAFR2_L	0x40E0_0064	0x2AAA_AAAA	GAFR2_L[AF78]为 nCS2，需设置为 0b10

表 8.5　GPIO 寄存器设置表

打开 LCD 背光

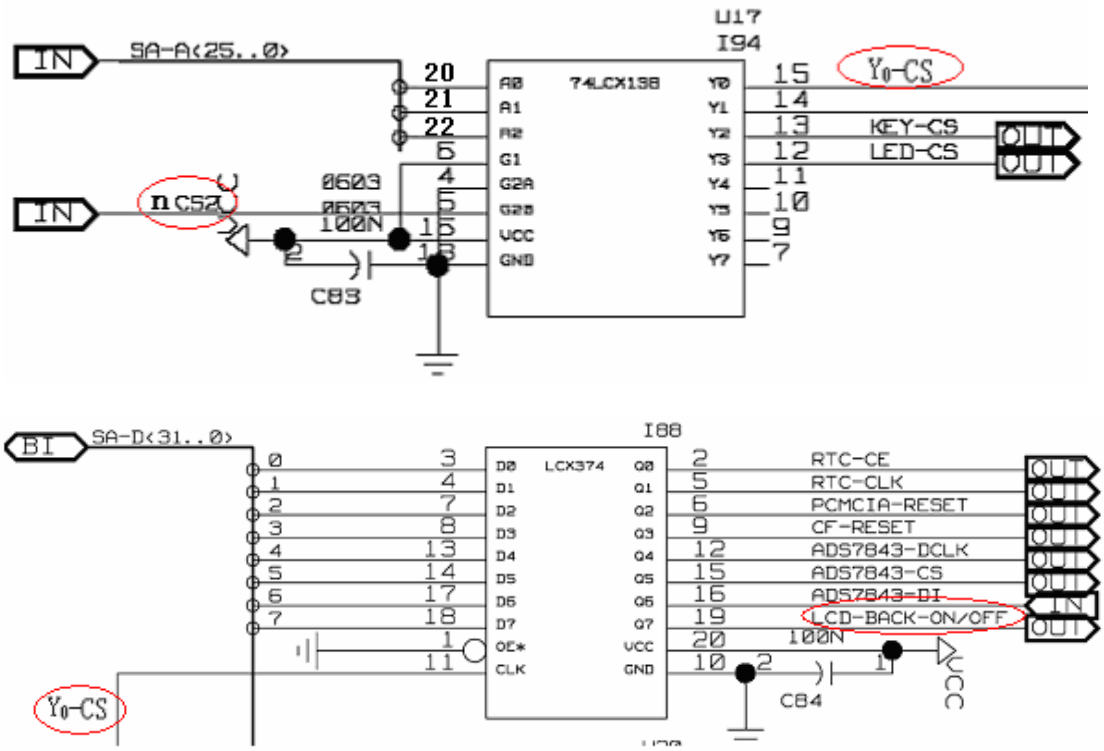


图 8.12　LCD 背光电路图

LCD 的背光是通过上图的 LCD-BACK-ON/OFF 控制的，只要这一引脚输出高电平，LCD 背光就可打开。首先，以片选信号 nCS2 选中 74LCX138 译码器，并且以地址线 20,21,22 作为译码输入，让 74LCX138 的输出引脚 Y_0 为高电平，该信号可让 LCX374 锁存来自数据总线的低 8 位数据，只要在数据总线上输出 0x80，则可以使 LCD-BACK-ON/OFF 输出高电平。

以下是 74LCX138 的真值表：

输入信号			输出信号							
A_{22}	A_{21}	A_{20}	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0

1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

表 8.6　74LCX138 的真值表

nCS2 对应的引脚为 GP78，需将此引脚设置为输出状态，即需要在 GPDR2[PD78]设置为 1，

另外要将该 GP78 引脚的功能设置为 nCS2，即需要在 GAFR2_L[AF78]设置为 0b10，由于 nCS2 的片选基地址为 0x0800_0000，并且译码输入信号为 000（即 $A_{22}A_{21}A_{20}$=000），所以可以只需访问地址 0x0800_0000，并且输出 0x80 便可以打开 LCD 背光。程序如下

```
mov  r1, #0x80
mov  r0, #0x08000000
strh  r1, [r0, #0]
```

4) 设置帧描述符（Frame Descriptors）

LCDC 内的 DMAC 需要初始化才可以工作，因为在 DMAC 工作之前需要提供 DMAC 一些相关信息：

下一个帧描述符的位置。

Frame Buffer 的位置。

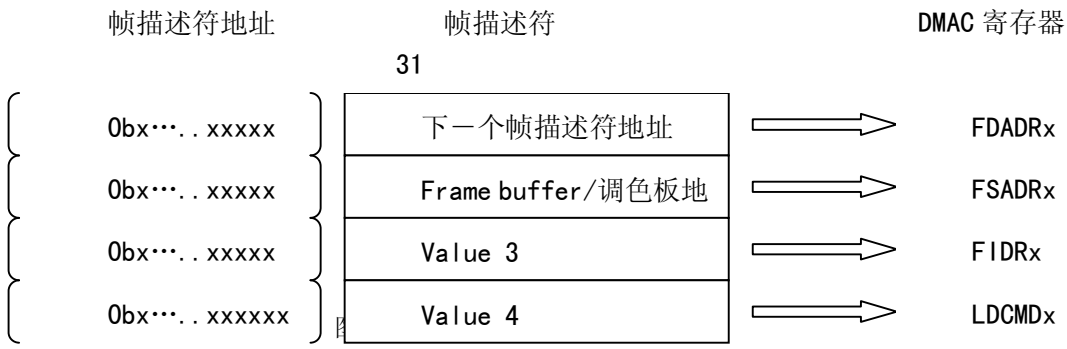
现在传送的是调色板数据还是图像数据。

在一个帧开始/结束时是否产生中断，

以及所传送的数据的总容量。

LCDC 内关于帧描述符的寄存器有四个，它们分别是：FDADRx，FSADRx，FIDRx，LDCMDx（x 取 0 或 1，与 DMAC 的 0 号通道或 1 号通道相对应），这四个寄存器的初始化方式并不是通过程控赋值，而是通过 DMAC 自动提取预先在内存里写好的数值，然后自动为这些寄存器赋值。为了让 DMAC 找到这些预先写好的数值，第一次使用 DMAC 或停用 LCDC 后又重新使用时，仍然需要通过软件来初始化 FDADRx，因为 FDADRx 的数值就是表示下一个帧描述符的位置。当 DMAC 获得帧描述符的位置后，它从该位置获取相应的数值，然后将另外三个寄存器初始化，再读取写在内存的为 FDADRx 预先准备好的数值，根据该数值跳转到指定的位置读取其他的帧描述符。

首先，需要在启动 LCDC 前定义帧描述符，该帧描述符必须定义在内存空间，而且必须是连续的 4×4 字节的空间，这个空间划分为 4 个单元，每个单元都是 32 位，并且帧描述符的开始地址能够被 8 整除，即二进制表示的开始地址的最低 3 位必须为 0。如下所示：



这里需要提出一点：FDADRx 的值是指下一个帧描述符的地址，并不是当前的帧描述符的地址。读者可能会提出为什么需要这么多帧描述符，这里主要有两个原因：

如果需要将外部的调色板装载到内部调色板时，就需要另外一个帧描述符来描述该外部调色板的空间，这样，当 LCDC 启动后，DMAC 就可以知道从哪里装入外部调色板数据。

如果使用多个 Frame Buffer，也需要使用帧描述符来为 DMAC 提供相关信息。LCDC 启动后至少需要一个 Frame Buffer 来存储当前显示的图像数据，但可以多开辟几个 Frame Buffer 来保存数据，这样便能够快速切换显示屏的画面，另外还可以将需要进行处理的数据在另一 Frame Buffer 内存储，处理完再显示出来。

通过程序为 FDADRx 赋值：

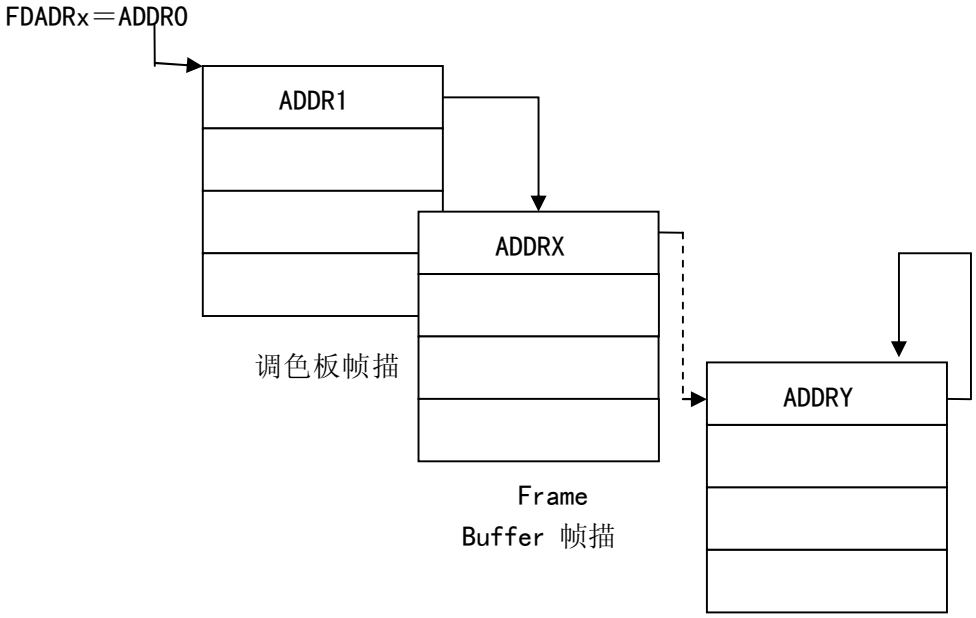
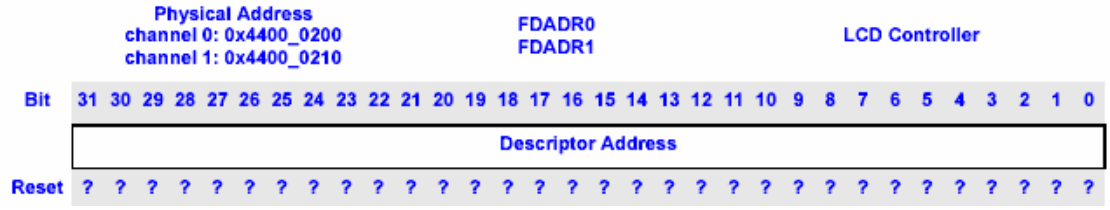


图 8.14　帧描述符的访问流程

如果只使用一个帧描述符，则必须将帧描述符的第一单元设为当前帧描述符的地址，如上图，若只装入调色板描述符，需将 ADDR1 设为 ADDR0。

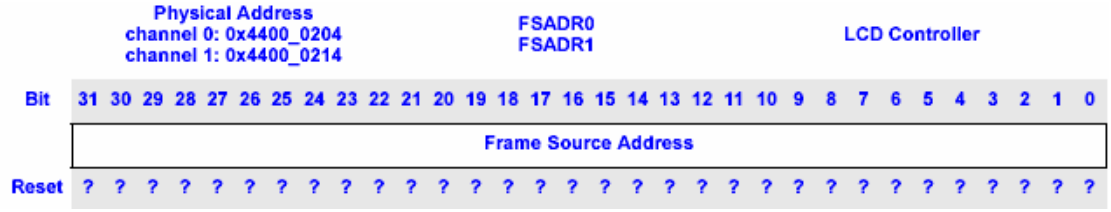
由于帧描述符的 4 个单元实质就是寄存器 FDADR0，FSADR0，FIDR0，LDCMD0 的数值，所以在帧描述符里设置的数值必须依据这些寄存器进行配置。

寄存器 FDADR0：



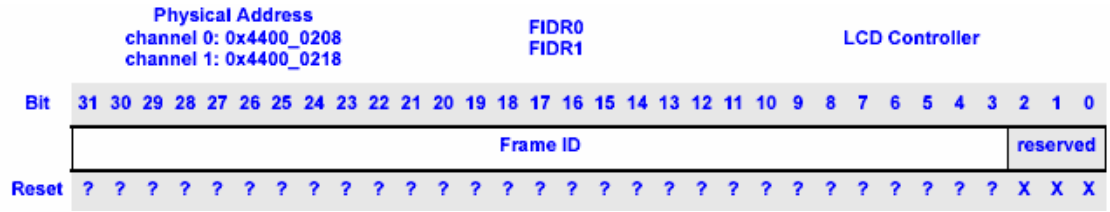
寄存器 FDADRx 存储的是下一个帧描述符地址，该地址必须能够被 8 整除，即 FDADRx[2:0]为 0。

寄存器 FSADRx：



寄存器 FSADRx 存储的是 Frame Buffer 的地址，该地址必须能够被 8 整除，即 FDADRx[2:0]为 0。

寄存器 FIDRx：



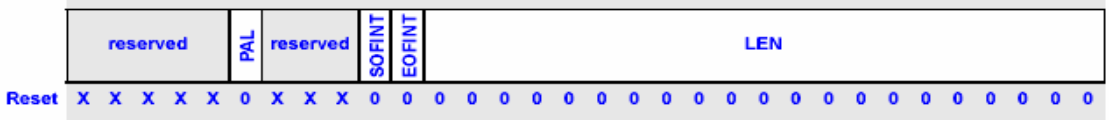
寄存器 FSADRx 存储的是当前帧的 ID，这个 ID 会在中断发生时传到 LCD Controller Interrupt ID 寄存器。本实验不使用 LCDC 提供的中断，故实验中不使用该寄存器，设 0 即可。

寄存器 LDCMDx：

Physical Address
channel 0: 0x4400_020C
channel 1: 0x4400_021C

LDCMD0
LDCMD1

LCD Controller



Bits	Name	Description
26	PAL	这个位是用来告知当前帧描述符是外部调色板的描述符还是 Frame Buffer 描述符。0 代表是 Frame Buffer 描述符；1 代表是外部调色板描述符
22	SOFINT	该位决定当一个新帧开始时是否发出一个中断信号。0 代表不发出
21	EOFINT	该位决定当一个帧传送完毕后是否发出一个中断信号。0 代表不发出
20:0	LEN	该位指定了传送数据的字节数。如果本描述符是 Frame Buffer 帧描述符，则该位数值应该是整个 Frame Buffer 的大小。如果是调色板描述符，1 位/象素和 2 位/象素显示模式的情况，LEN 应为 8；4 位/象素显示模式的情况，LEN 为 32；8 位/象素的显示模式的情况下，LEN 应为 512

表 8.7　LDCMDx 各位描述表

由于本实验使用的是 TFT 显示屏，故无需初始化内部调色板，只需一个帧描述符来描述 Frame Buffer 空间便可以了。根据以上讲解，在内存地址 0xA030_0000 开始的地方开辟一个帧描述符。如下所示：

```
[0xA030_0000] = 0xA030_0000
[0xA030_0004] = 0xA050_0000
[0xA030_0008] = 0x0
[0xA030_000C] = 0x0009_6000
```

5）设置寄存器 LCCR1，LCCR2，LCCR3

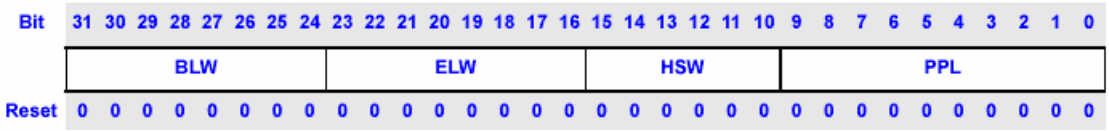
本实验对 LCCR1 设置如下：

LCCR1 = 0x530F_EE7F

Physical Address
0x4400_0004

LCD Controller Control Register 1

LCD Controller



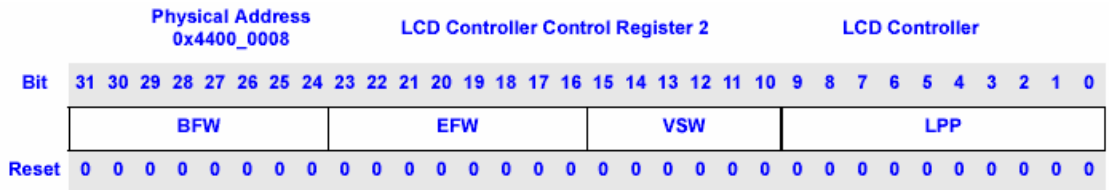
Bits	Name	Value	Description
31:24	BLW	0d147	在发送每一行象素前，插入（BLW+1）个象素时钟周期
23:16	ELW	0d15	在发送每一行象素后，插入（ELW+1）个象素时钟周期
15:10	HSW	0d59	在 Active 模式下，具体指定了水平同步脉冲的宽度，实际值=

			HSW＋1；
9:0	PPL	0d639	具体指定了 LCD 每一行的象素个数，实际象素个数＝（PPL＋1）

BLW，ELW，HSW 的设置没有具体的要求，应根据所使用的 LCD 适当调节，读者可以修这些值，达到改善显示效果的作用。但 PPL 必须与所使用的 LCD 一致。

本实验对 LCCR2 设置如下：

LCCR2 = 0x210A_05DF

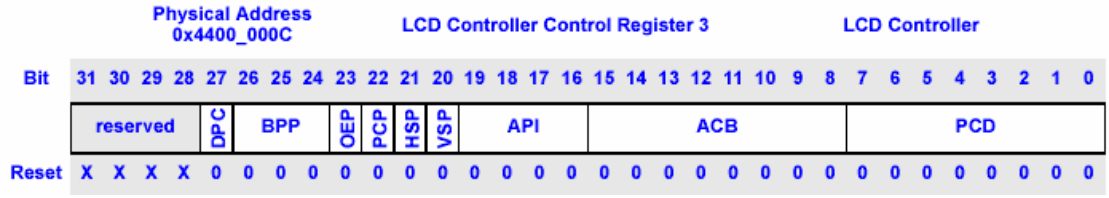


Bits	Name	Value	Description
31:24	BFW	0d33	在 Active 模式，具体指定了在发送每一帧前插入（BFW＋1）个行时钟周期
23:16	EFW	0d10	在 Active 模式，具体指定了在发送完每一帧后插入（EFW＋1）行时钟周期
15:10	VSW	0d2	在 Active 模式下，具体指定了垂直同步脉冲的宽度为（EFW＋1）
9:0	LPP	0d479	具体指定了 LCD 的行数，实际行数＝（LPP＋1）

BFW，EFW，VSW 的设置没有具体的要求，应根据所使用的 LCD 适当调节，读者可以修这些值，达到改善显示效果的作用。但 LPP 必须与所使用的 LCD 一致。

本实验对 LCCR3 设置如下：

LCCR3 = 0x0440_FF01



Bits	Name	Value	Description
27	DPC	0b0	象素时钟的频率是由（LCLK/PCD）决定，该位能够决定 L_PCLK 输出的频率是否两倍该数值，0 表示 L_PCLK 的输出频率直接为该数值；1 表示两倍该数值。
26:24	BPP	0b100	表示每个象素所需的位数 000－1bits/pixel；001－2bits/pixel；010－4bits/pixel；011－8bits/pixel；100－16bits/pixel

23	OEP	0b0	该位决定 L_BIAS 激活时的性。当 OEP 为 0 时，L_BIAS 输出高电平时为激活；取 1，则输出低电平时为激活。
22	PCP	0b1	象素时钟极性。当 PCP 取值 0 时，对数据的取样发生在 L_PCLK 的上升沿，取 1，则发生在 L_PCLK 的下降沿
21	HSP	0b0	该位决定水平同步信号的极性，当 HSP 取 0 时，L_LCLK 输出高电平为激活状态，取 1，则输出低电平为激活状态
20	VSP	0b0	该位决定垂直同步信号的极性，当 VSP 取 0 时，L_FCLK 输出高电平为激活状态，取 1，则输出低电平为激活状态。
19:16	API	0b0	设置引发 API 中断的 L_BIAS 输出个数，本实验不使用 API 中断，故该位设为 0
15:8	ACB	0d255	AC Bias 引脚频率，在 Passive 模式下，它会影响 L_Bias 的触发频率，但在 Active 模式，ACB 不影响 L_BIAS，是因为象素时钟不断的触发，而 L_BIAS 被用作可用输出信号，当象素数据发送到 LCD 数据引脚时，该信号就会被触发。
7:0	PCD	0b1	设置象素时钟的频率，取值范围是 0－255，象素时钟频率是基于 LCD/Memory 控制器时钟频率（LCLK），即产生的象素时钟频率的取值范围从 LCLK/2 到 LCLK/255，实际所产生的象素时钟的频率还会受到 DPC 的影响。

6）设置 FDADR0，LCCR0

FDADR0 的数值为帧描述符地址，设置该寄存器是为了让 DMAC 自动到 FDADR0 所指的位置读取帧描述符，并且自动为 FSADR0，FIDR0，LDCMD0 赋值。

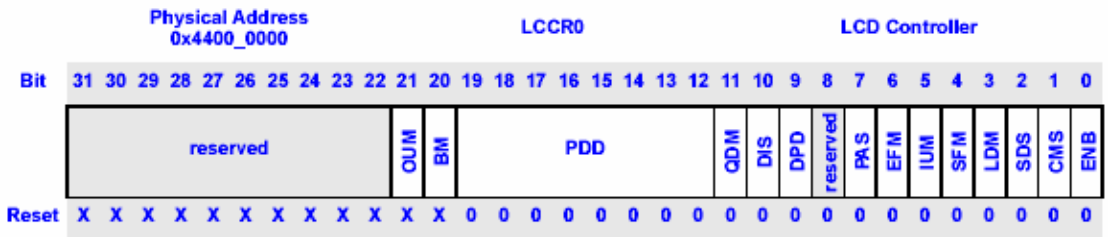
对 FDADR0 设置如下：

FDADR0 ＝ 0xA030_0000

地址 0xA030_0000 为本实验在内存开辟的帧描述符的首地址。本寄存器必须在 LCDC 启动之前设置。

LCCR0 设置如下：

LCCR0 ＝ 0x0030_0CF8



Bits	Name	Value	Description
21	OUM	1	该位决定是否屏蔽 Output FIFO Underrun 中断。0 表示允许产生该中断；1 则表示屏蔽该中断
20	BM	1	当 DMAC 从一个帧描述符切换到另一个帧描述符里提取数据时，可以产生一个中断，设 1 则表示屏蔽
19:12	PDD	0	该位可在 DMA 每次装载内部调色板时插入等待时钟。
11	QDM	1	该位决定是否屏蔽 LCD Quick Disalbe 中断。0 表示允许；1 表示屏蔽
10	DIS	1	LCD 控制器当前不可用
9	DPD	0	在 Passive，单色，单屏幕模式下 0 代表使用 L_DD[3:0]来传送数据 1 代表使用 L_DD[7:0]来传送数据
7	PAS	1	Passive(STN)/Active(TFT)显示屏选择 0 表示 Passive 显示操作；1 表示 Active 显示操作。这个位由所接 LCD 的类型来决定，如果是 STN，则选 0；如果是 TFT，则选 1
6	EFM	1	屏蔽当一个帧发送完毕后所产生的中断
5	IUM	1	屏蔽 Input FIFO Underrun 中断
4	SFM	1	屏蔽当发送一个新的帧时产生的中断
3	LDM	1	屏蔽 LCD Disable Done 中断
2	SDS	0	单/双屏幕显示 0 表示单屏幕；1 表示双屏幕
1	CMS	0	彩色/单色显示 0 表示彩色；1 表示单色
0	ENB	0	LCD 控制器是否可用 0 表示不可用；1 表示可用

7）字符显示

设置完以上的寄存器后，现在将需要显示的图像“画”到 Frame Buffer。下图是 32×32 的字模。

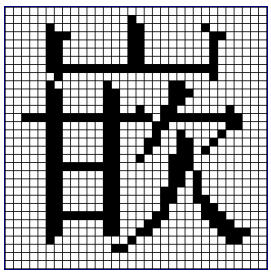


图 8.15 字模

字模的每一行有 32 个小格，计算机只需用 1 位来表示 1 个小格是否有填色，如果有填色，则用 1 表示，没有则用 0 表示，所以字模的 1 行需要用 32 位来表示，以 16 进制表示如下：

0x00，0x00，0x00，0x00，
0x00，0x01，0x00，0x00，
0x04，0x01，0x80，
⋮
0x00，0x00，0x00，

（32）

本实验采用的显示模式是 16 位/象素，Frame Buffer 里表示每个象素的象素值需要 2 个字节，LCD 上任何一个象素对应与 Frame Buffer 的计算公式如下：

$$Position(x, y) = Address\_base + y \times 640 \times 2 + x \times 2$$

Position(x,y)为 LCD 上的（x，y）坐标在 Frame Buffer 的对应地址。Address_base 为 Frame Buffer 的基地址。只要对上表的每一位进行扫描，当扫描到 1，则在 Frame Buffer 的相应位置设置象素值就能将该字符打印出来。

8）启动 LCD 控制器

LCCR0[ENB] = 1;

当 LCDC 启动后，DMAC 便根据 FDADR0 的值找到帧描述符，然后将帧描述符的第二个单元传递到寄存器 FSADR0，DMAC 便到 FSADR0 所指的地址提取象素数据。

LCD 控制器可以在使用时关闭，然后在需要使用时再打开。通过在 LCCR0[DIS]设“1”

可以关闭 LCD；当关闭后，可以通过重新为 LCCRO 赋值，并且在 LCCRO[ENB]设“1”来重新启动 LCDC。这里值得注意的是如果内部调色板已经被初始化，重启 LCDC 就无需再次载入外部调色板，仅传送 Frame buffer 数据即可，所以可能需要在重启 LCDC 前为 FDADRO 再次赋值。

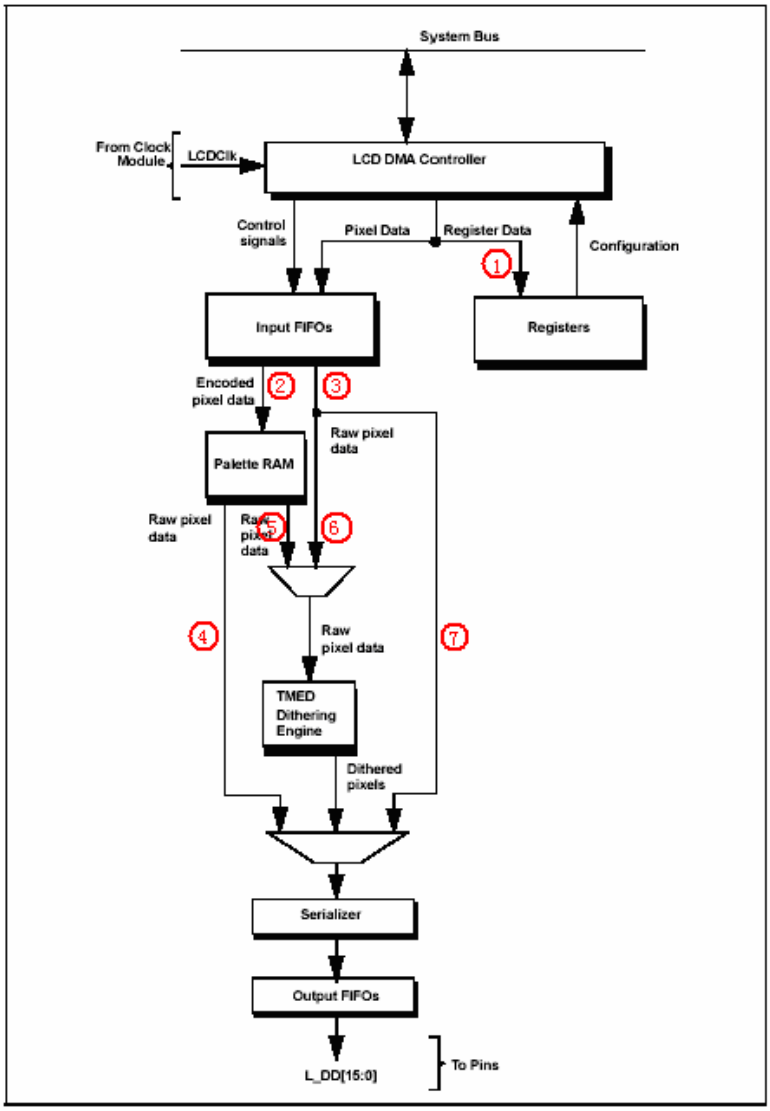
[实验内容]

1）读者可以根据前几章所讲述的办法对程序进行编译与下载，由于本章例子程序采用汇编与 C 语言的混合编程方式，所以需要设置映像的 first 属性将中断向量表的所在 AREA 的开始位置固定在 0x0：－first boot.o(boot)，设置方法可以参照前几章所述。

2）观察代码执行情况

[习题与思考题]

1. 结合下图，简要阐述 LCDC 工作原理，并分析图中标注的 7 点分别在什么情况下发生。



2. 若使用 1024×768 的显示屏，并以 8bits/pix 显示，则 Frame Buffer 需要多少字节。

3. 若使用双屏幕模式显示，应如何初始化 LCDC 内部的第二个 DMAC 通道。

4. 若需要使用内部调色板，应如何操作 LCDC，此时 DMAC 如何区别是 Frame Buffer 还是调色板的的内容。

5. 结合本实验所掌握的知识要点，实现一个包含图片，文字等要素的互动式图形界面。

实验九 触摸屏

[实验目的]

√ 理解触摸屏与触摸屏控制器芯片 UCB1400BE 的工作原理。

[实验原理]

程序介绍

本章例子是一个驱动触摸屏控制器芯片 UCB1400BE 的程序，将来自触摸屏输出端的电压值转换为数字量，并将点击的位置通过串口打印出来。

实验步骤

本章实验目标：理解触摸屏的工作原理，并且学习如何驱动芯片 UCB1400BE，结合触摸屏与 UCB1400BE，编写驱动程序，将点击的目标位置打印出来。

原理概述

1) 触摸屏

触摸屏是一套透明的绝对定位系统，从技术原理来区别触摸屏，可分为四个基本种类：电阻技术触摸屏、电容技术触摸屏、红外线技术触摸屏、表面声波技术触摸屏。本实验使用的触摸屏属于电阻式触摸屏。

a. 电阻触摸屏

电阻触摸屏的屏体部分由多层复合薄膜构成，按结构和实现原理来划分可以分为四线、五线、七线和八线式，由于本实验所使用的设备为四线电阻触摸屏，故以下内容仅适合 4 线电阻触摸屏。电阻触摸屏通常由五层薄膜叠合在一起，如图 9.1 所示。最外面两层分别为基层和塑料层，主要是为保护屏体而设计，触摸屏的工作主要是靠在中间两层互相绝缘的导电层来完成。基层是一层玻璃或有机玻璃，塑料层则是一层外表面硬化处理、光滑防刮的塑

料，塑料层必须具备一定的弹性，当受到外来挤压时可以出现一定程度的凹陷。而在基层的上面紧贴两层互相绝缘透明导电层（ITO，氧化铟），这两层导电层是通过中间的透明隔离点隔开。当外力施加在塑料层上时，会造成外表面的凹陷，导致两层导电层接触，当触摸屏控制器测得输入端（四线触摸屏的其中一条输出线）的有电压输入时，便知道触摸屏被点击。

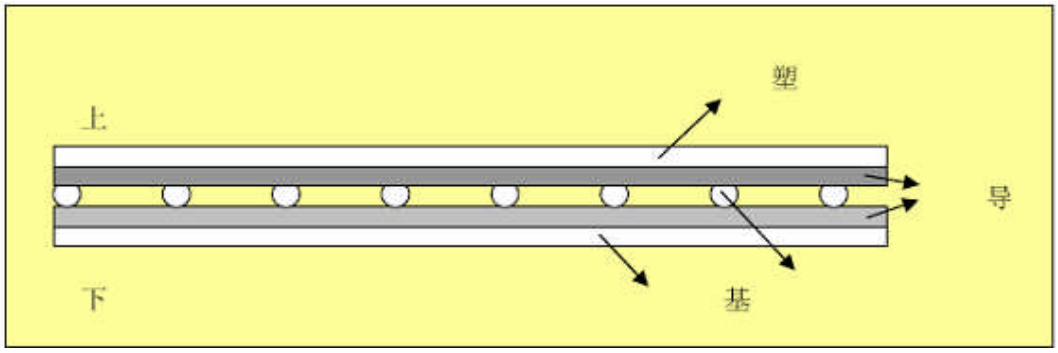


图 9.1：触摸屏结构图

现在我们来分析一下触摸屏内的两层导电层的工作原理。以四线电阻触摸屏为例，这两层导电层其实是阻性层，层内按照垂直或水平方向平均分布电阻，以图 9.2 所示。

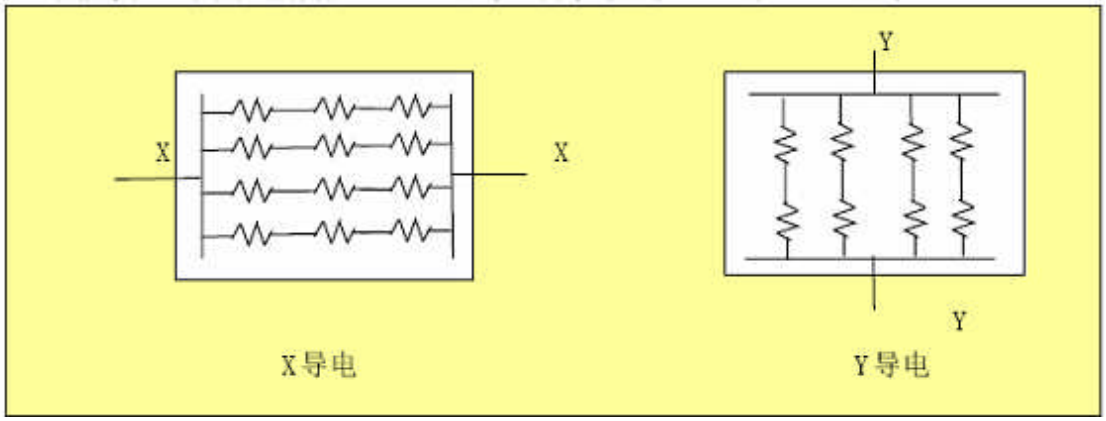


图 9.2：导电层结构图

在 X 导电层中，X＋和 X－分别接在导电层的左右两端，在这两端间按水平方向平均分布电阻，水平分布的电阻互不干扰；同理，Y 导电层是按垂直方向分布电阻，垂直分布的电阻互不干扰。在没有通电且没有挤压触摸屏时，我们通过万用表可以测得 X＋和 X－之间以及 Y＋和 Y－之间的电阻，另外测量得知 X 端与 Y 端之间的电阻为无限大，可以知道此时两导电层互隔；通过挤压触摸屏，X 端与 Y 端之间可以测得电阻，此时表明两导电层出现短路。

b.红外线触摸屏

红外线触摸屏安装简单，只需在显示器上加上光点距架框，无需在屏幕表面加上涂层或接驳控制器。光点距架框的四边排列了红外线发射管及接收管，在屏幕表面形成一个红外线网。用户以手指触摸屏幕某一点，便会挡住经过该位置的横竖两条红外线，电脑便可即时算出触摸点的位置。任何触摸物体都可改变触点上的红外线而实现触摸屏操作。

c.电容式触摸屏

电容式触摸屏的构造主要是在玻璃屏幕上镀一层透明的薄膜体层，再在导体层外上一块保护玻璃，双玻璃设计能彻底保护导体层及感应器。此外，在附加的触摸屏四边均镀上狭长的电极，在导电体内形成一个低电压交流电场。用户触摸屏幕时，由于人体电场、手指与导体层间会形成一个耦合电容，四边电极发出的电流会流向触点，而其强弱与手指及电极的距离成正比，位于触摸屏幕后的控制器便会计算电流的比例及强弱，准确算出触摸点的位置。电容触摸屏的双玻璃不但能保护导体及感应器，更有效地防止外在环境因素给触摸屏造成影响，就算屏幕沾有污秽、尘埃或油渍，电容式触摸屏依然能准确算出触摸位置。

d.表面声波触摸屏

表面声波触摸屏的触摸屏部分可以是一块平面、球面或是柱面的玻璃平板，安装在 CRT、LED、LCD 或是等离子显示器屏幕的前面。这块玻璃平板只是一块纯粹的强化玻璃，区别于别类触摸屏技术是没有任何贴膜和覆盖层。玻璃屏的左上角和右下角各固定了竖直和水平方向的超声波发射换能器，右上角则固定了两个相应的超声波接收换能器。玻璃屏的四个周边则刻有 45°角由疏到密间隔非常精密的反射条纹。

2) 触摸屏控制器

本实验用到的触摸屏控制器选用 UCB1400BE，它是一个立体声音频多媒体数字信号编解码器并整合触摸屏与电源管理接口。立体声音频多媒体数字信号编解码器以线性的方式输出信号，直接驱动耳机。触摸屏接口直接连接到四线式的触摸屏，内置一个 10 位模拟信号到数字信号转换器来读出触摸屏与电源管理的参数值，提供 10 个通用 I/O 可编程针脚作为系统的输出/输入。

a.特点

采用 48 针脚 LQFP 封装。

整合 AC’97 Rev.2.1 接口。

20 针脚立体声音频多媒体数字信号编解码器提供可编程抽样速率，并增加输出/输入控制。

4 线式电阻触摸屏接口电路提供位置、压力、金属板反抗测量。

带有 10 位连续的近似值整合轨迹跟踪 A/D 转换控制器，类似多路复用器的触摸屏输出，扩展电压供应（7.5V）电源的管理。

10 个通用 I/O 口。

3.3V 外部电源供电，内置便携式的电池的支持应用。

b.引脚定义与电路图

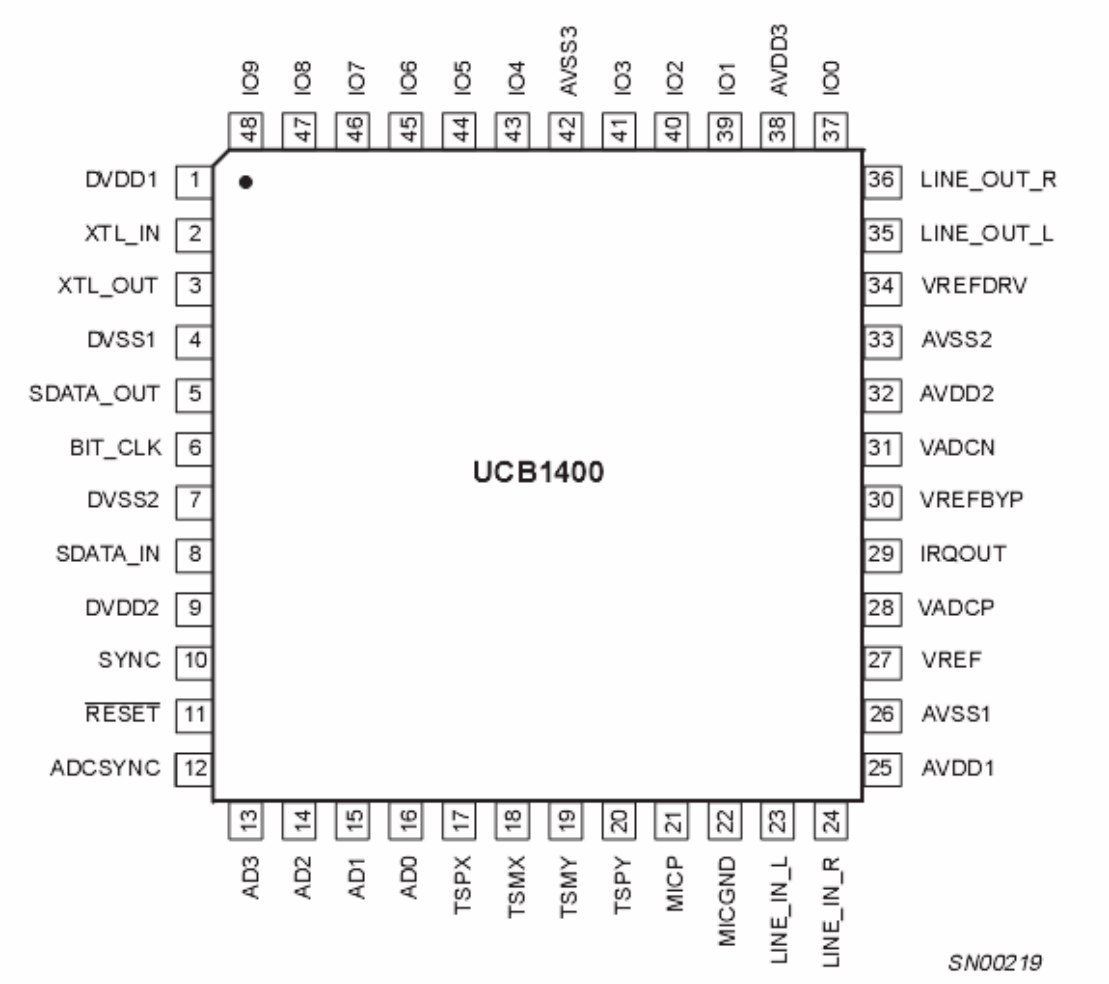


图 9.3：LQFP 封装图

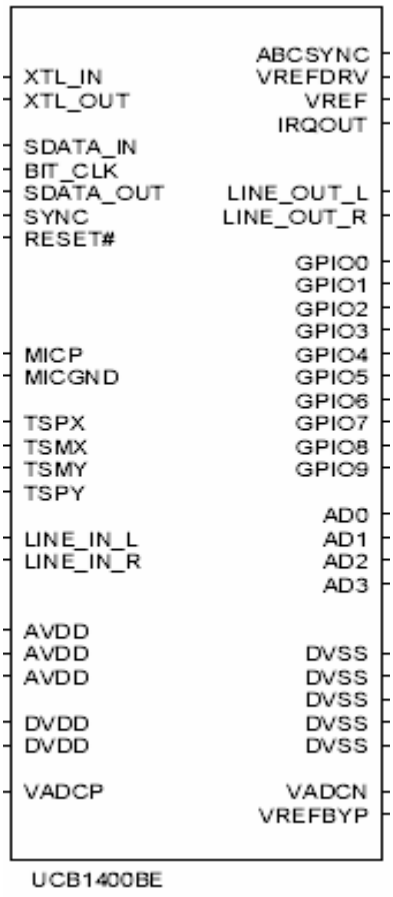


图 9.4：UCB1400BE 功能引脚

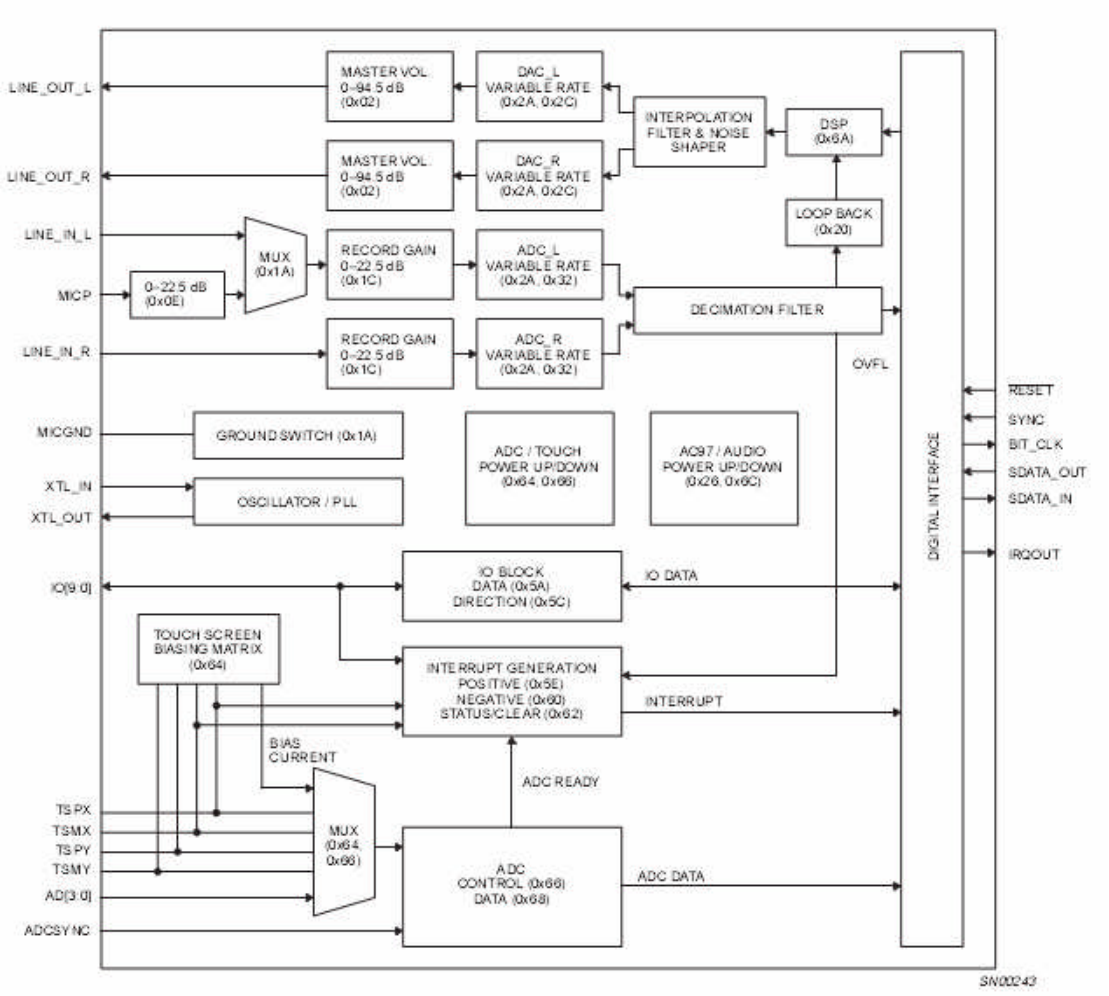


图 9.5：UCB1400BE 结构图

由图 9.3、图 9.4 和图 9.5 可知，我们可以看到各个引脚的结构与功能。在图 9.3、图 9.4 和图 9.5 中，TSPX、TSMX、TSMY、TSPY 这四个引脚是触摸屏的接口。表 9.1 是 ADC 与触摸屏接口的定义。

表 9.1：ADC 与触摸屏接口的定义

符号	针脚	类型	描述
AD[3：0]	13、14、15、16	输入	电压输出
TSPX	17	输入/输出	触摸屏 X+端
TSMX	18	输入/输出	触摸屏 X-端
TSMY	19	输入/输出	触摸屏 Y-端

TSPY	20	输入/输出	触摸屏 Y+端

根据图 9.4 的 UCB1400BE 功能引脚中，从 17、18、19、20 四个引脚引出 X+、X-、Y+、Y-，如图 9.6、图 9.7 所示。

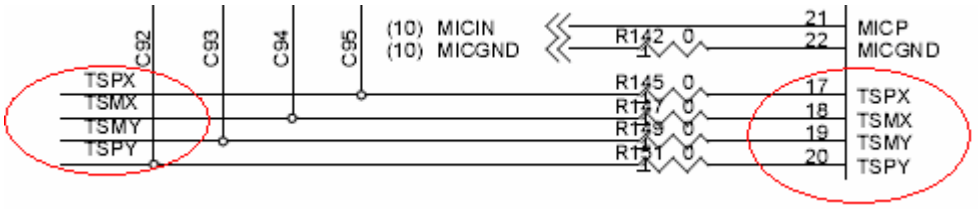


图 9.6：触摸屏接口电路图

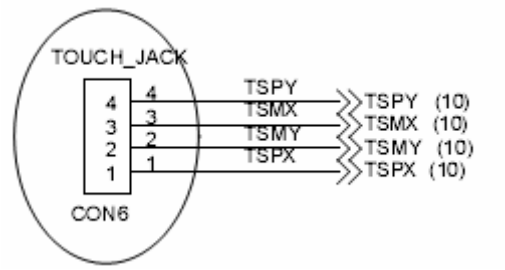


图 9.7：触摸屏接线插槽

c. UCB1400BE 中断

UCB1400BE 包含一个可编程的中断控制模块，可以由一个或者多个通用 I/O[9：0]产生一个 0 到 1 或者 1 到 0 信号转换，包括音频加载侦察、ADC 准备信号和 TSPX 和 TSMX 信号。图 8 为 UCB1400BE 的中断电路图。

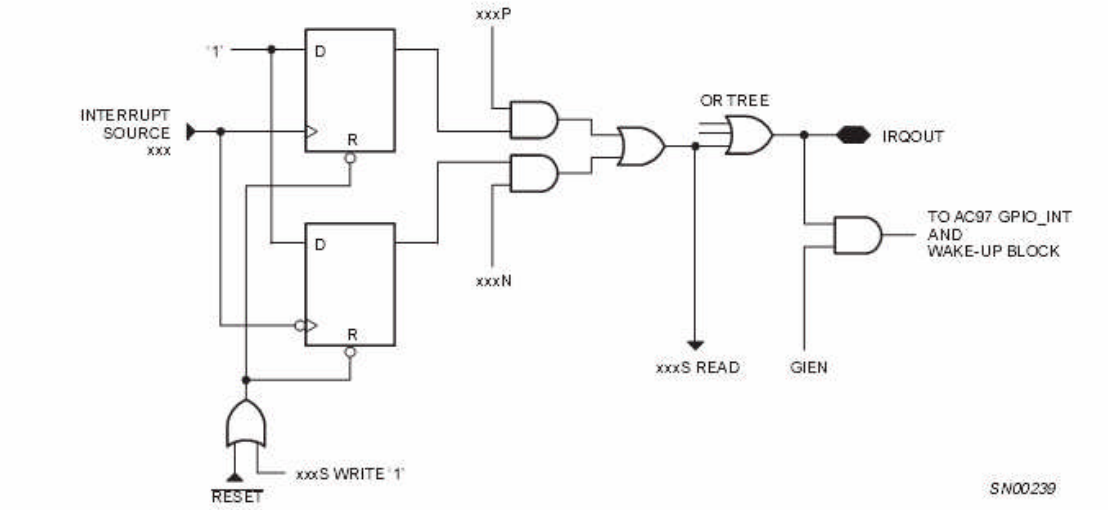


图 9.8：UCB1400BE 的中断电路图

中断的模式由 Positive INT Enable Register (0x5E) and Negative INT Enable Register (0x60)设置。每个信号的真实状态可由 INT Clear/Status Register (0x62)读出。只要 INT Clear/Status Register(0x62)的相应位置 1，中断状态就会被清除。

中断控制器是一个异步工具，当 BIT_CLK 位停止时就可能产生中断。例如，一个中断就有可能发生在电源挂起模式下，当按下触摸屏或者 I/O 管脚的状态改变时。

3) 触摸屏接口

UCB1400BE 是一个通用的四线电阻式触摸屏，它能够测量位置、压力、平板电阻值。除此之外，当触摸屏受压力时，可产生可编程中断。触摸屏接口以四线的方式连接到触摸屏，它们分别是：TSPX、TSMX、TSPY、TSMY。每个引脚都是由可编程触摸屏寄存器的 PXP、MXP、PYP、MYP 和 PXG、MYG、MYG 位组成。图 9.9 就是触摸屏接口内部电路图。触摸屏每个引脚的信号都可以作为 10 位的内置 ADC 的输入，ADC 通常用作对触摸屏选定针脚的位置测量模式进行电压检测。

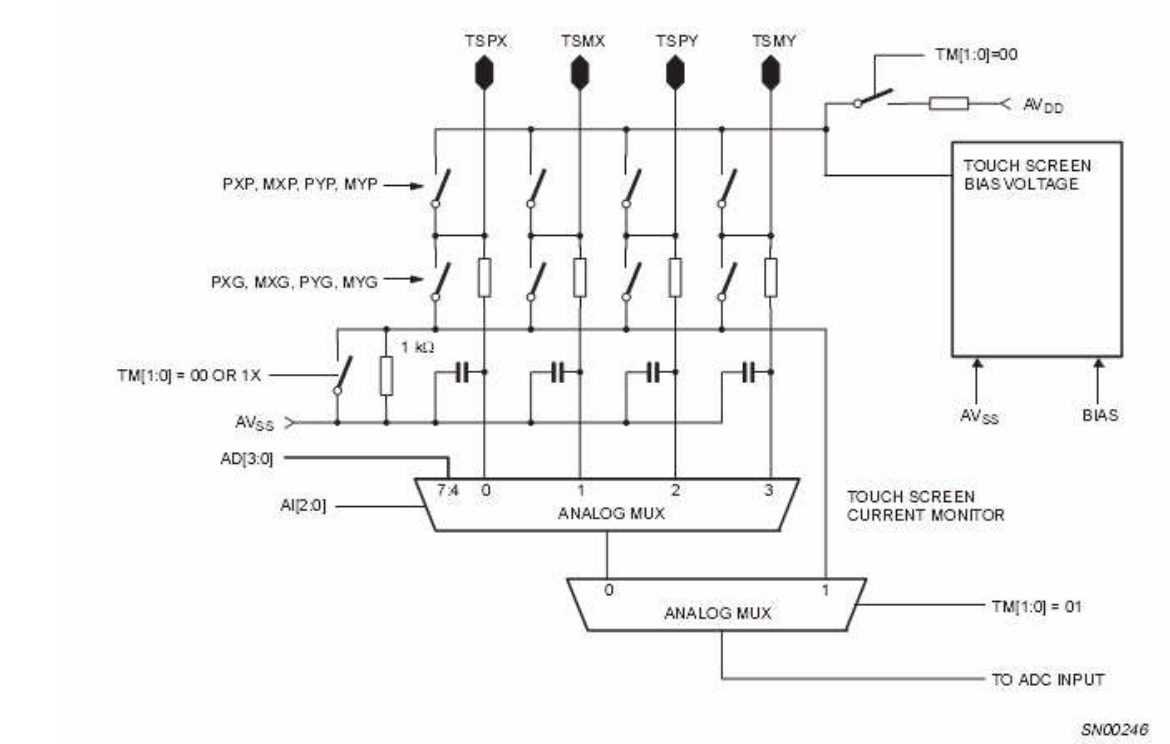


图 9.9：触摸屏接口内部电路图

由图 9.9 触摸屏的内部电路图可知，触摸屏接口连接以四根线：TSPX，TSMX，TSPY 和 TSMY 连接到触摸屏。在触摸屏矩阵的每个针脚都可以被编程，供电，或者接地。每个针脚都在可以被触摸屏控制寄存器设置为 PXP、MXP、PYP、MYP 和 PXG、MYG、PYG、MYG 位。UCB1400BE 可以检测可能存在的冲突设置（同时设置触摸屏的针脚为接地和供电），如果是这样的话，UCB1400BE 会设置针脚为接地。

这四个触摸屏信号都可以作为内置 10 位 ADC 的输入端，ADC 在测量模式下通常检测触摸屏针脚位置的电压。除此之外，UCB1400BE 可以通过内部的 1KΩ电阻控制触摸屏，这个 1KΩ电阻在压力或者平板电阻测量模式下作为内置 10 位 ADC 的输入。交换矩阵和复用定义的触摸屏偏置电压电路使能 UCB1400BE 的配置。

UCB1400BE 内部电压 V_{ref} 作为触摸屏偏置电压控制电路的参考电压。由于 LCD 与触摸屏紧密联结起来，并且 LCD 是一个典型的产生噪音的器件，触摸屏终端尽可能过滤来自从 LCD 到触摸屏的噪音信号。

触摸屏的偏置电路和 ADC 多路复用器的设置由触摸屏控制寄存器的 TM[1：0]决定。如表 9.2 所示，显示了触摸屏的选择模式。

表 9.2：触摸屏的选择模式

TM[1：0]	选择模式	触摸屏偏置来源	ADC 多路复用设置

00	中断	电阻到 AV_{DD}	定义 AI[2：0]
01	压力	触摸屏偏置电路	触摸屏当前控制
10	位置	触摸屏偏置电路	定义 AI[2：0]
11	位置	触摸屏偏置电路	定义 AI[2：0]

UCB1400BE 提供三种触摸屏测量模式：位置、压力和平板电阻。另外，中断模式用作触摸屏提供了触摸事件。

图 9.11 显示了位置测量的模式，列出了 X 位置测量。

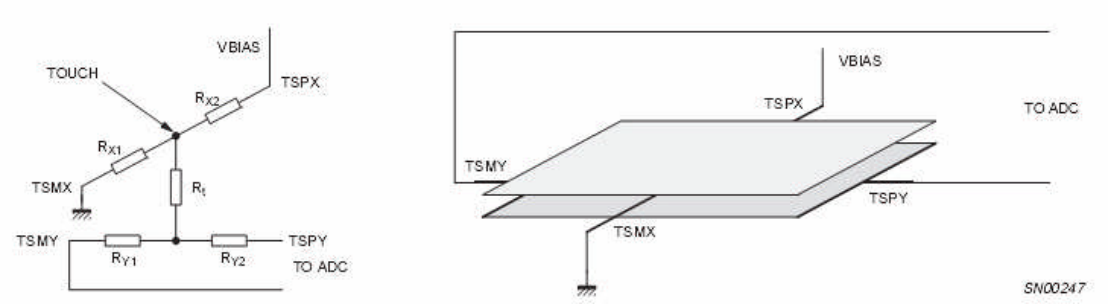


图 9.11：位置测量模式

两个位置(X,Y)测量需要检测挤压的位置。当 X 位置测量和一个或者二个 Y 终端(TSPY、TSMY）受挤压的时候，X 平板会被偏置。电路会被分压，并且 TSPY 和/或者 TSMY 电极会被“擦除”。在 TSPY/TSMY 终端的标准电压与触摸屏挤压 X 位置相对应。

在 Y 位置模式下，X 平板和 Y 平板终端互换，因此 Y 平板被偏置，TSPX 和/或者 TSMX 终端的电压就可以被测量出来了。

图 9.12 显示了压力测量模式。

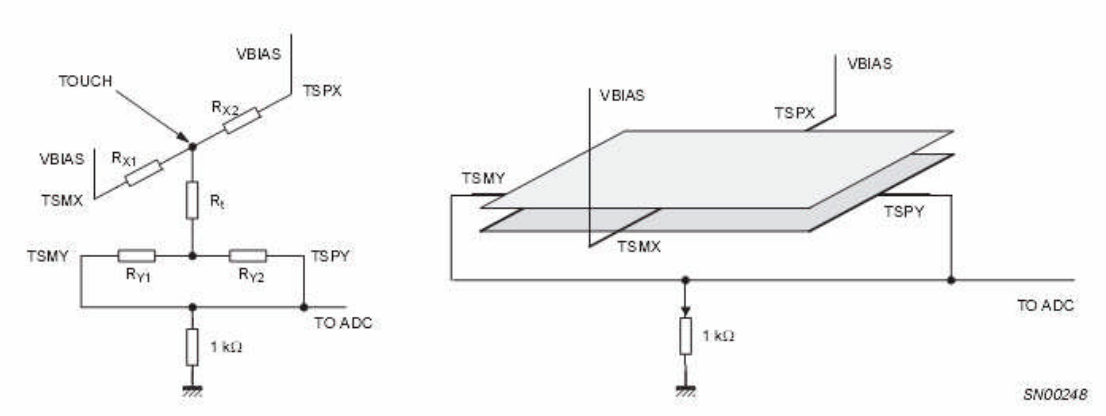


图 9.12：压力测量模式

当有压力挤压时，触摸屏可以检测出来。实际上，在 X 与 Y 平板之间的接点电阻就会被测量，显示挤压现场和应用压力的位置尺寸。触摸屏铁笔、手指都会导致一个相当大的区域的挤压，尖铁笔、笔会导致一个小区域挤压。

在挤压模式时，一个或者二个终端的平板会被偏置，然而其它平板会被接地，与偏置的平板相反。当前变动的电压直接经由触摸屏通过，直接显示出两平板之间的电阻。串行电阻会的补偿会被触摸屏平板和内部 1KΩ 电阻转换为改进准确度的测量值。

图 9.13 显示了平板电阻测量模式。

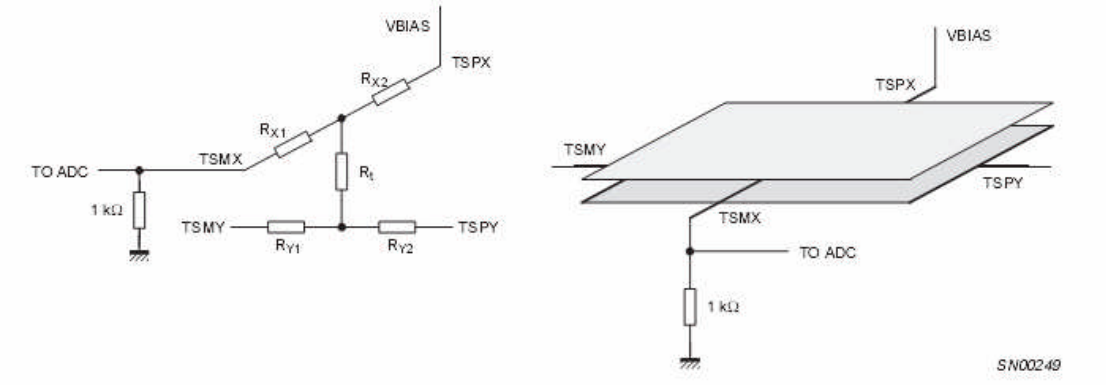


图 9.13：显示了平板电阻测量模式

触摸屏将平板电阻分成很多的小部分。得知的实际平板电阻产生平板电阻补偿，实现平板电阻测量。其次，当两个或者更多接触点挤压触摸屏时，平板的电阻值会慢慢减少。如果是这样的话，一段平板的一部分，例如 X 平板，会被另一部分平板缩短，来减少实际的平板电阻。

平板电阻测量模式的执行方式与压力电阻测量模式相同。既然这样，两个平板电阻的电极只会有一个被偏置，其它平板保持不确定状态。

图 9.14 显示了触摸屏的中断模式

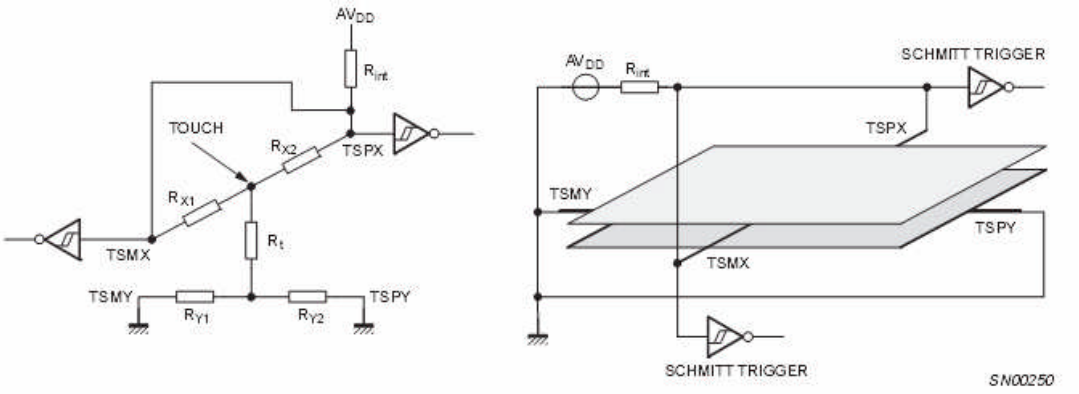


图 9.14：触摸屏的中断模式

除了以上介绍的几种模式之外，触摸屏还可以看成一个中断源。在这种模式下，触摸屏的 X 平板接入电源，而 Y 平板接地。既然这样，触摸屏不会通过偏置电路进行电压偏置，但是通过一个到 AV_{DD} 的电阻。这种配置方式只是简单的偏置，UCB1400BE 不会消耗电源，除非触摸屏受到挤压。当触摸屏受到挤压时，X 平板终端的电压会下降。施密特触发电路会检测到电压输出的下降，并连接到中断控制模块。当触摸屏受挤压（下降沿使能）或者触摸屏松开（上升沿使能）时，就会产生触摸屏中断。这会被用来激活 UCB1400BE 从触摸屏读出中断的次序。当内置的低电平过滤打开之后，内部的施密特触发电路信号就连接到 TSPX 和 TSMX 。将 LCD 屏与触摸屏传感器连结在一起，这样可以减少一些假的中断产生。

[实验内容]

1)分析代码

结合以上说明，对本实验所提供的汇编源代码进行分析，深入理解针对具体的硬件实现，软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图 9.15 所示：

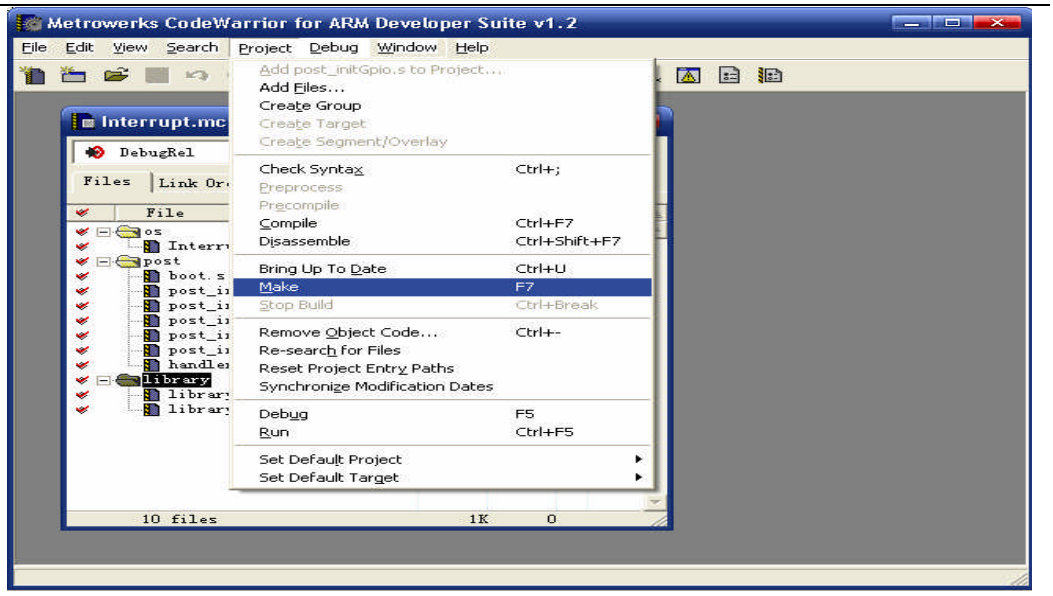


图 9.15 编译、连接生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)设置超级终端的波特率为 38400，数据流控制位为无。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1、简述 4 线电阻式触摸屏的工作原理。

2、UCB1400BE 触摸屏的参考模式有哪几种，各应用在什么场合，如何连接硬件以及软件如何设置。

实验十 MMU

[实验目的]

√ 了解存储器管理单元的原理；
√ 了解与 MMU 相关的协处理器寄存器功能；
√ 掌握 MMU 中地址变换过程。

[实验原理]

程序介绍

本实验重点讲述 ARM 的存储器管理单元 MMU 管理机制，运行该程序后会在串口上打印出“Run in virtual memory mode!”一行字符串，该试验代码启动了 MMU 后，在虚拟地址空间中完成了对串口的初始化，并且打印了以上字符串，串口波特率是 38400。

存储器管理单元 MMU 概述

在 ARM 系统中，存储器管理单元 MMU 主要完成以下工作：

虚拟存储空间到物理存储空间的映射。在 ARM 中采用了页式虚拟存储管理。它把虚拟地址空间分成一个个固定大小的块，每一块称为一页，把物理内存的地址空间也分成同样大小的页。页的大小分为粗粒度和细粒度两种。MMU 就是从虚拟地址到物理地址的转换。

存储器访问权限的控制。

设置虚拟存储空间的缓冲的特性。

在 ARM 存储系统中，使用 MMU 实现虚拟地址到实际物理地址的映射。为何要实现这种映射？首先就要从一个嵌入式系统的基本构成和运行方式着手。系统上电时，处理器的程序指针从 0x0（或者是由 0Xffff_0000 处高端启动）处启动，顺序执行程序，在程序指针（PC）启动地址，属于非易失性存储器空间范围，如 ROM、FLASH 等。然而与上百兆的嵌入式处理器相比，FLASH、ROM 等存储器响应速度慢，已成为提高系统性能的一个瓶颈。而 SDRAM 具

有很高的响应速度，为何不使用 SDRAM 来执行程序呢？为了提高系统整体速度，可以这样设想，利用 FLASH、ROM 对系统进行配置，把真正的应用程序下载到 SDRAM 中运行，这样就可以提高系统的性能。

然而这种想法又遇到了另外一个问题，当 ARM 处理器响应异常事件时，程序指针将要跳转到一个确定的位置，假设发生了 IRQ 中断，PC 将指向 0x18(如果为高端启动，则相应指向 0xffff_0018 处)，而此时 0x18 处仍为非易失性存储器所占据的位置，则程序的执行还是有一部分要在 FLASH 或者 ROM 中来执行的。那么我们可不可以使程序完全都 SDRAM 中运行那？答案是肯定的，这就引入了 MMU，利用 MMU，可把 SDRAM 的地址完全映射到 0x0 起始的一片连续地址空间，而把原来占据这片空间的 FLASH 或者 ROM 映射到其它不相冲突的存储空间位置。例如，FLASH 的地址从 0x0000_0000－0x00ff_ffff,而 SDRAM 的地址范围是 0x3000_0000－0x31ff_ffff，则可把 SDRAM 地址映射为 0x0000_0000－0x1fff_ffff 而 FLASH 的地址可以映射到 0x9000_0000－0x90ff_ffff（此处地址空间为空闲，未被占用）。映射完成后，如果处理器发生异常，假设依然为 IRQ 中断，PC 指针指向 0x18 处的地址，而这个时候 PC 实际上是从位于物理地址的 0x3000_0018 处读取指令。通过 MMU 的映射，则可实现程序完全运行在 SDRAM 之中。

3、页表

页表（Translate Table）是实现上述这些功能的重要手段，它是一个位于内存中的表。表的每一行对应于虚拟存储空间的一个页，该行包含了该虚拟内存页（称为虚页）对应的物理内存页（称为实页）的地址、该页的方位权限和该页的缓冲特性等。这里将页表中这样的一行称为一个地址变换条目（entry）。

页表存放在内存中，系统通常有一个寄存器来保存页表的基地址。在 ARM 系统中，协处理器 CP15 的寄存器 C2 用来保存页表的基地址。

从虚拟地址到物理地址的变换过程其实就是查询页表的过程，由于页表存放在内存中，这个查询过程通常代价很大。而程序在执行过程中具有局部性，因此，对页表的访问只是局限在少数几个单元中。根据这一特点，采用一个容量更少（通常为 8~16 个字）、访问速度和 CPU 中通用寄存器相当的存储器件来存放当前访问需要的地址变换条目。这个小容量的页表称为快表。快表在英文资料中被称为 TLB（Translation Lookaside Buffer）。

当 CPU 需要访问内存时，先在 TLB 中查找需要的地址变换条目。如果该条目不存在，CPU 从位于内存中的页表查询，并把相应的结果添加到 TLB 中。这样，当 CPU 下一次又需要该地址变换条目时，可以从 TLB 中直接得到，从而使地址变换的速度大大加快。

当内存中的页表内容改变，或者通过修改协处理器 CP15 的寄存器 C2 使用新的页表时，TLB 中的内容需要全部清除。MMU 提供了相关的硬件支持这种系统。协处理器 CP15 的寄存器 C8 用来控制清除 TLB 内容的相关操作。

MMU 可以将整个存储空间分为最多 16 个域（Domain）。每个域对应一定的内存区域，

该区域具有相同的访问控制属性。MMU 中寄存器 C3 用于控制与域相关属性的配置。当存储访问失效时，MMU 提供了相应的机制用于处理这种情况。在 MMU 中寄存器 C5 和寄存器 C6 用于支持这些机制。

ARM 协处理器 CP15

常用指令

ARM 的协处理器指令主要用于 ARM 处理器初始化 ARM 协处理器的数据处理操作，以及在 ARM 处理器的寄存器和协处理器的寄存器之间传送数据，和在 ARM 协处理器的寄存器和存储器之间传送数据。ARM 协处理器指令包括以下 5 条：

— CDP 协处理器数操作指令
— LDC 协处理器数据加载指令
— STC 协处理器数据存储指令
— MCR ARM 处理器寄存器到协处理器寄存器的数据传送指令
— MRC 协处理器寄存器到 ARM 处理器寄存器的数据传送指令

1) CDP 指令

CDP 指令的格式为：

CDP{条件} 协处理器编码，协处理器操作码 1，目的寄存器，源寄存器 1，源寄存器 2，协处理器操作码 2。

CDP 指令用于 ARM 处理器通知 ARM 协处理器执行特定的操作,若协处理器不能成功完成特定的操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，目的寄存器和源寄存器均为协处理器的寄存器，指令不涉及 ARM 处理器的寄存器和存储器。

指令示例：

```
CDP      P3，2，C12，C10，C3，4        ；该指令完成协处理器 P3 的初始化
```

2) LDC 指令

LDC 指令的格式为：

LDC{条件}{L} 协处理器编码,目的寄存器，[源寄存器]

LDC 指令用于将源寄存器所指向的存储器中的字数据传送到目的寄存器中，若协处理器不能成功完成传送操作，则产生未定义指令异常。其中，{L}选项表示指令为长读取操作，如用于双精度数据的传输。

指令示例：

```
LDC      P3，C4，[R0]                ；将 ARM 处理器的寄存器 R0 所指向的存储
器中的字数据传送到协处理器 P3 的寄存器 C4 中。
```

3) STC 指令

STC 指令的格式为：

STC{条件}{L} 协处理器编码,源寄存器，[目的寄存器]

STC 指令用于将源寄存器中的字数据传送到目的寄存器所指向的存储器中，若协处理器不能成功完成传送操作，则产生未定义指令异常。其中，{L}选项表示指令为长读取操作，如用于双精度数据的传输。

指令示例：

```
STC     P3，C4，[R0]                ；将协处理器 P3 的寄存器 C4 中的字数据传送到 ARM 处理器的寄存器 R0 所指向的存储器中。
```

4) MCR 指令

MCR 指令的格式为：

MCR{条件} 协处理器编码，协处理器操作码 1，源寄存器，目的寄存器 1，目的寄存器 2，协处理器操作码 2。

MCR 指令用于将 ARM 处理器寄存器中的数据传送到协处理器寄存器中,若协处理器不能成功完成操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，源寄存器为 ARM 处理器的寄存器，目的寄存器 1 和目的寄存器 2 均为协处理器的寄存器。

指令示例：

```
MCR     P3，3，R0，C4，C5，6     ；该指令将 ARM 处理器寄存器 R0 中的数据传送到协处理器 P3 的寄存器 C4 和 C5 中。
```

5) MRC 指令

MRC 指令的格式为：

MRC{条件} 协处理器编码，协处理器操作码 1，目的寄存器，源寄存器 1，源寄存器 2，协处理器操作码 2。

MRC 指令用于将协处理器寄存器中的数据传送到 ARM 处理器寄存器中,若协处理器不能成功完成操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，目的寄存器为 ARM 处理器的寄存器，源寄存器 1 和源寄存器 2 均为协处理器的寄存器。

指令示例：

```
MRC     P3，3，R0，C4，C5，6     ；该指令将协处理器 P3 的寄存器中的数据传送到 ARM 处理器寄存器中。
```

协处理器寄存器

MMU 由系统控制寄存器的 2、3、4、5、6、8、10 号寄存器和 1 号寄存器的一些位控制。表 10.1 与 MMU 操作相关的寄存器。

表 10.1：与 MMU 操作相关的寄存器

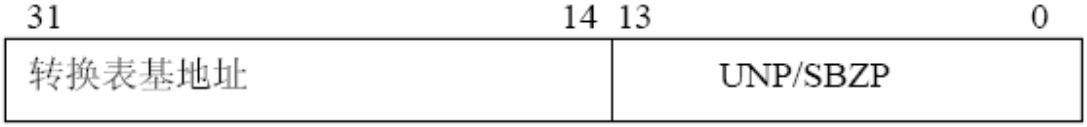
协处理器寄存器	作用
C1 中的某些位	用于配置 MMU 中的一些操作
C2	保存内存中页表的基地址
C3	设置域(Domain)的访问控制属性
C4	保留
C5	内存访问失效状态指示
C6	内存访问失效时失效的地址
C8	控制与清除 TLB 内容相关的操作
C10	控制与锁定 TLB 内容相关的操作

下面分别介绍一下各个寄存器功能：

1) C1：某些位用来控制 MMU

M(bit[0])	使能 MMU。 0 = 禁止 MMU 1 = 允许 MMU 在没有 MMU 和保护单元的系统上，这个位应该读出为 0，并忽略写。
A(bit[1])	使能对齐错检查 0 = 禁止 1 = 允许
S(bit[8])	这是系统保护位。
R(bit[9])	这是 ROM 保护位。

2) C2：转换表基地址



读 CP15 寄存器 2 时，在 bits[31:14]返回当前活动的第一级转换表的物理地址，bits[13:0]不确定。

读 CP15 寄存器 2 时，CRm 和操作数 2 被忽略，并应该是 0。

写 CP15 寄存器 2 时，在 bits[31:14]更新当前活动的第一级转换表的物理地址，bits[13:0]应该写 0 或先前读回的值。

写 CP15 寄存器 2 时，CRm 和操作数 2 被忽略，并应该是 0。

3）C3：域访问控制

读 CP15 寄存器 3 时，返回域访问控制寄存器的值。CRm 和操作数 2 被忽略，并应该是 0。

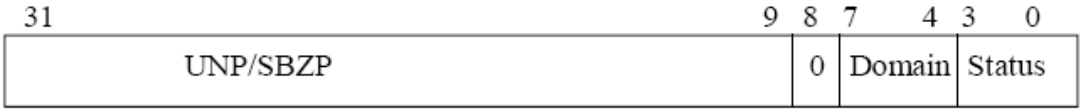
写 CP15 寄存器 3 时，更新域访问控制寄存器的值。CRm 和操作数 2 被忽略，并应该是 0。

域访问控制寄存器包含 16 个 2 位的字段，它定义了对应域的访问权限。

4）C4：保留

读写 CP15 寄存器 4 不可预料结果。

5）C5：错误状态 FSR



读 CP15 寄存器 5 时，返回 FSR 寄存器的值。FSR 包含最近一次数据错的信息。只有低 9 位有效，高 23 位不确定。FSR 指出异常发生时的域和试图访问的类型。

bit[8]　　返回 0

bit[7:4]　　指出错位发生时访问的域

bit[3:0]　　试图访问的类型，这些位的编码见表 10.2：

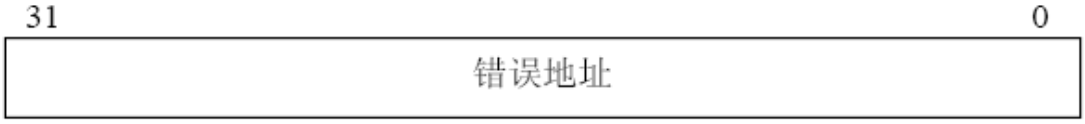
表 10.2：域的访问控制字段编码及含义

控制位编码	访问类型	含义
00	没有访问权限	这时访问该域将产生访问失效
01	客户类型	根据页表中地址变换条目中的访问权限控制位决定是否允许特定的存储访问
10	保留	使用该值会产生不可预知的结果
11	管理者权限	不考虑页表中地址变换条目中的访问权限控制位这种情况下不会产生访问失效

FSR 在数据错时更新，由具体实现确定取指令异常是否更新 FSR。错误地址寄存器(FAR)和错误状态寄存器(FSR)， CRm 和操作数 2 被忽略，并应该是 0。

写 FSR 将把 FSR 设定成写的值。这对于程序调试器非常有用，可以用来恢复 FSR 的值。高 24 位应该写 0 或上次读到的值。CRm 和操作数 2 被忽略，并应该是 0。

6) C6：错误地址 FAR



读 CP15 寄存器 6 返回 FAR 的值。FAR 保存着错误产生时访问的虚拟地址。在数据错时更新 FAR。由具体实现确定取指令异常是否更新 FSR。错误地址寄存器(FAR)和错误状态寄存器(FSR)， CRm 和操作数 2 被忽略，并应该是 0。

写 FSR 将把 FAR 设定成写的值。这对于程序调试器非常有用，可以用来恢复 FAR 的值。高 24 位应该写 0 或上次读到的值。CRm 和操作数 2 被忽略，并应该是 0。

7) C8：TLB 功能

当 CP15 的寄存器 8 用来控制 TLB 时是只读寄存器。表 3 显示了定义的 TLB 功能和在 MCR 指令中用的 CRm 和第二个朝操作数<opcode2>的值。使用没有在表中的 CRm 和 opcode2 的组合将导致不可预料的结果。

如果下面的任何操作被用在单一 TLB 的实现中，则在单一 TLB 中实现相同的功能：

无效的指令 TLB（Invalidate instruction TLB）
无效的指令单一入口（Invalidate instruction single entry）
无效的整个数据 TLB（Invalidate entire data TLB）
无效的数据单一入口（Invalidate data single entry）

否则，如果执行一个与特定实现不相关的功能，会导致不确定的结果。

表 10.3：TLB 功能

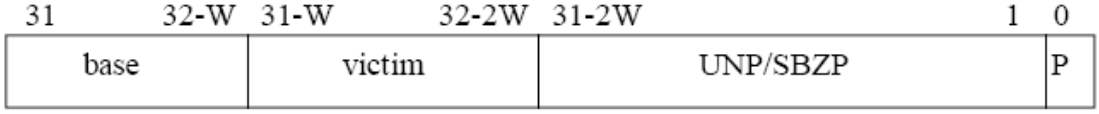
功能	Opcode2	CRm	Data	指令
无效整个唯一的 TLB 或指令和数据 TLB	0b000	0b0111	SBZ	MCR p15,0,Rd,c8,c7,0
无效唯一的单一入口	0b001	0b0111	Virtual address	MCR p15,0,Rd,c8,c7,1
无效整个指令 TLB	0b000	0b0101	SBZ	MCR p15,0,Rd,c8,c5,0
无效指令单一入口	0b001	0b0101	Virtual address	MCR p15,0,Rd,c8,c5,1
无效整个数据 TLB	0b000	0b0110	SBZ	MCR p15,0,Rd,c8,c6,0
无效数据单一入口	0b001	0b0110	Virtual address	MCR p15,0,Rd,c8,c6,1

8）C10：TLB 锁定

转换表遍历的执行需要一定的时间，特别当访问慢速的主存储器时。在实时中断处理程序中，当 TLB 不包含中断处理程序的转换和/或要访问的数据时，中断延迟回大量加长。

TLB 锁定是一些 ARM 存储器系统的特性，它允许把特定的转换表遍历的结果装载到 TLB 中。这种方式不会被后来的转换表遍历的结果覆盖。由 CP15 寄存器 10 设定。

设 W=LOG2(TLB 入口数)，如果需要的话取整(round-up)，则 CP15 寄存器 10 的格式为：



如果具体的实现有分开的指令和数据 TLB，那么有 2 个不同的寄存器，由访问寄存器 10 的 MCR 或 MRC 指令中的 opcode2 字段选择：

opcode2 == 0	选择数据 TLB 锁定寄存器
opcode2 == 1	选择指令 TLB 锁定寄存器

如果具体的实现只有唯一的 TLB，那么只有 1 个寄存器，opcode2 字段应该为 0。访问寄存器 10 的 MCR 或 MRC 指令中的 CRm 总应该为 0。

写寄存器 10 有如下结果：

victim　　字段表示下次 TLB 失败(miss)时，转换表遍历的结果替代哪个 TLB 入口。

Base　　字段包含 TLB 替换的策略，只使用从(base)到(TLB 入口-1)的 TLB 入口，victim 应该在这个区间。

转换表遍历的结果在写到 TLB 入口时，若 P==1 则它被保护起来，不能被寄存器 8 的使整个 TLB 失效操作影响；若 P==0 则会被那些操作给失效掉。

TLB 锁定过程：

通常锁定 N 个 TLB 入口的过程如下：

<1> 禁止中断等，来保证当这个过程执行时不会产生异常

<2> 如果一个指令 TLB 或唯一 TLB 被锁定，用 base==N、index==N 和 P==0 写到适当版本的寄存器 10。如果可能，把另指令预取很难理解的分枝预测功能关掉。

<3> 使要被锁定的整个 TLB 失效。

<4> 如果是指令 TLB 锁定，要确保剩下的锁定过程所要预取的指令相关的 TLB 入口都被装载。(要注意锁定是从哪里开始的，通常可能一个 TLB 入口包含所有这些。这时 TLB 被失效后的第一条指令能完成这个功能)如果是数据 TLB 锁定，要确保剩下的锁定过程所要访问数据的相关的 TLB 入口都被装载。这包含被代码用到的嵌入文字(inline literals)（通常最好避免在锁定过程中使用嵌入文字，并把所有的数据放在由一个 TLB 入口所包含的区域，然后从那里加载一个数据）如果一个唯一 TLB 被锁定，执行以上所有的过程。

<5> i 从 0 到 N-1 循环

a. 用 base==i、index==i 和 P==1 写到寄存器 10。

b. 强迫被锁定到 TLB 入口 i 处的转换表遍历结果的存储器区域发生转换表遍历:

* 如果是数据 TLB 或唯一 TLB 被锁定，从那个区域加载一个数据

* 如果是指令 TLB 被锁定，用 B5-15 页所描述的指令预取高速缓冲寄存器 7 来在那个区域产生指令预取。

<6> 用 base==N、index==N 和 P==0 写到寄存器 10。

TLB 解锁过程：

用上面的过程解锁被锁定的 TLB 部分：

<1> 用寄存器 8 的操作使每个被锁定的单一入口失效

<2> 用 base==0、index==0 和 P==0 写到寄存器 10。

禁止/使能 MMU

CP15 的寄存器 C1 的位[0]用于禁止/使能 MMU。当 CP15 的寄存 C1 的位[0]设置成 0 时，禁止 MMU；当 CP15 的寄存器 C1 的位[0]设置 1 时，使能 MMU。下面的指令使能 MMU。

```
MRC P15, 0, R0, C1, C0, 0
ORR R0, #01
MCR P15, 0, R0, C1, C0, 0
```

5、MMU 中地址变换过程

虚拟存储空间到物理存储空间的映射是以内存为单位进行的。即虚拟存储空间中一块连续的存储空间被映射成物理存储空间中同样大小的一块连续存储空间。在页表中（TLB 中也是一样的），每一个地址变换条目实际上记录了一个虚拟存储空间的存储块的基地址与物理存储空间相应的一个存储块的基地址的对应关系。根据存储块大小，可以有多种地址变换。

ARM 支持的存储块大小有以下几种：

段（section）：是大小为 1M 的存储模块。

大页（Large Pages）：是大小为 64KB 的存储模块。

小页（Small Pages）：是大小为 4KB 的存储模块。

极小页（Tiny Pages）：是大小为 1KB 的存储模块。

通过采用另外的访问控制机制，还可以将大页分成大小为 16KB 的子页；将小页分成大小为 1KB 的子页；极小页不能再细分，只能以 1KB 大小的整页为单位。

在 MMU 中采用下面两级页表实现上述的地址映射。

一级页表中包含有以段为单位的地址变换条目以及指向二级的指针。一级页表实现的地址映射粒度较大。

二级页表中包含以大页和小页为单位的地址变换条目。其中，一种类型的二级页表还包含有以极小页为单位的地址变换条目。

通常，以段为单位的地址变换过程只需要一级页表。而以页为单位的地址变换过程还需要二级页表。下面介绍这些地址变换过程。

(1) 基于一级页表的地址变换过程

一级页表的地址变换过程称为一级地址变换过程。一级地址变换过程如图 10.1 所示。

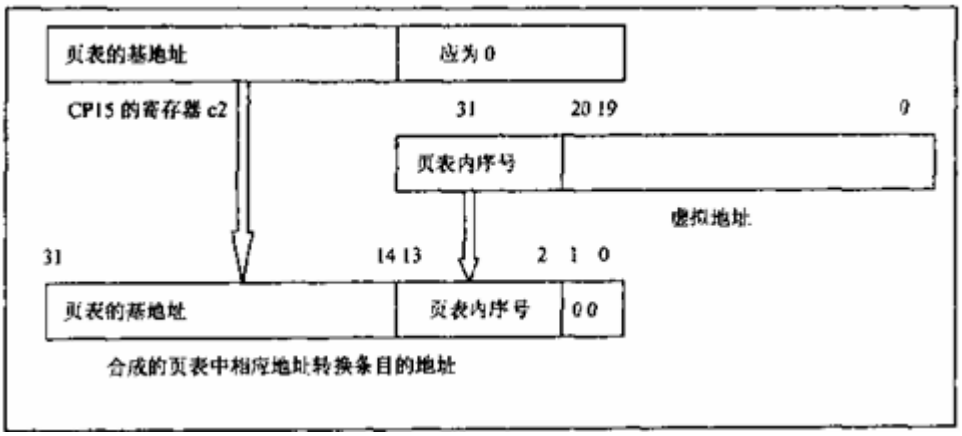


图 10.1：一级地址变换过程

CP15 的寄存器 C2 中存放的是内存中页表的基地址。其中位[31：14]为内存中页表的基地址，位[13：0]为 0。因此一级页表的基地址必须是 16KB 对齐的。CP15 的寄存器 C2 的位[31：14]和虚拟地址的位[31：14]和虚拟地址的位[31：20]结合作为一个 32 位数的高 30 位，在将该 32 位数的低两位值为 00，从而形成一个 32 位的索引值。使用该 32 位的索引值从页表中可以查到一个 4 字节的地址变换条目。该条目中或者包含了一个一级描述符(first-level describtor)，或者包含了一个指向二级页表的指针。

段描述符及其地址变换过程

当一级描述符的位[1：0]为 10 时，该一级描述符为段描述符。基于段的地址变换的过程如图 10.2 所示：

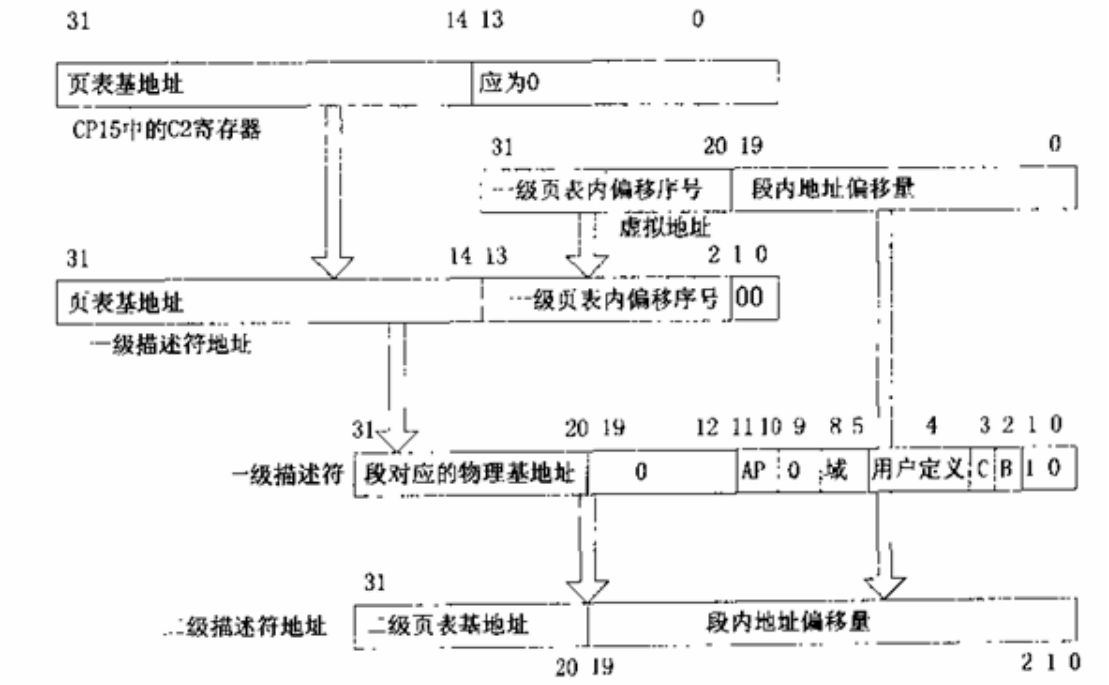


图 10.2：基于段的地址变换的过程

粗粒度页表描述符

当一级描述符的位[1：0]为 01 时，该一级描述符中包含了粗粒度的二级页表的物理地址，这种一级描述符称为粗度页表描述符。由粗粒度页表描述符获取二级描述符的过程如图 10.3 所示。

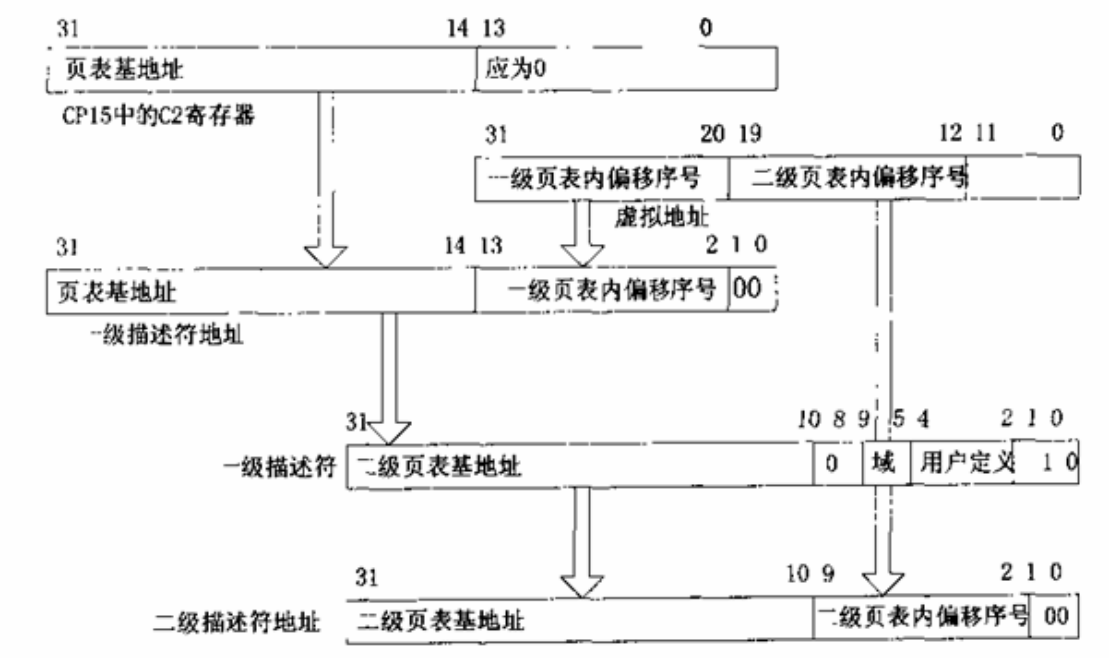


图 10.3：由粗粒度页表描述符获取二级描述符的过程

细粒度页表描述符

当一级描述符的位[1：0]为 11 时，该一级描述符中包含了细粒度的二级页表的物理地址，称为细度粒度页表描述符。由细粒度页表描述符获取二级描述符的过程如图 10.4 所示。

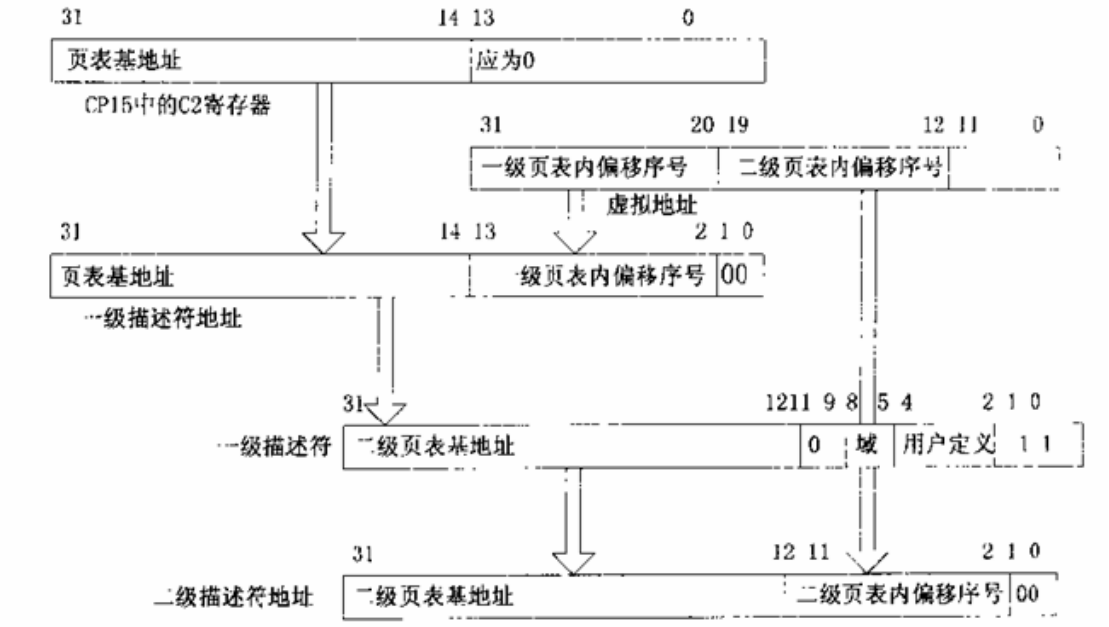


图 10.4：由细粒度页表描述符获取二级描述符的过程

(2) 基于二级页表的地址变换过程

二级页表有两种：粗粒度的二级页表和细粒度的二级页表。

粗粒度的二级页表以 4KB 为单位进行地址映射，即粗粒度二级页表中每个地址变换条目定义了如何将一个 4KB 大小的虚拟空间映射到同样大小的物理空间，同时定义了该空间的访问权限以及域控制属性等。由于每个粗度的二级页表定义了 1MB 大小的虚拟空间的映射关系，而每个条目定义了 4KB 大小的虚拟空间映射关系，每个条目的大小为 4 字节，因而每个粗粒度的二级页表的大小为 1KB。

细粒度的二级页表以 1KB 为单位进行地址映射，即细粒度的二级页表中每个地址变换条目定义了如何将一个 1KB 大小的虚拟空间映射到同样大小的物理空间，同时定义了该空间的访问权限一级域控制属性等。由于每个细粒度的二级页表定义了 1MB 大小的虚拟空间的映射关系，每个条目的大小为 4 字节，因而每个细粒度的二级页表的大小为 4KB。

大页描述符以及相关的地址变换

当页描述符的位[1：0]为 01 时，该二级描述符为大页描述符，大页地址变换过程如图 10.5 所示。

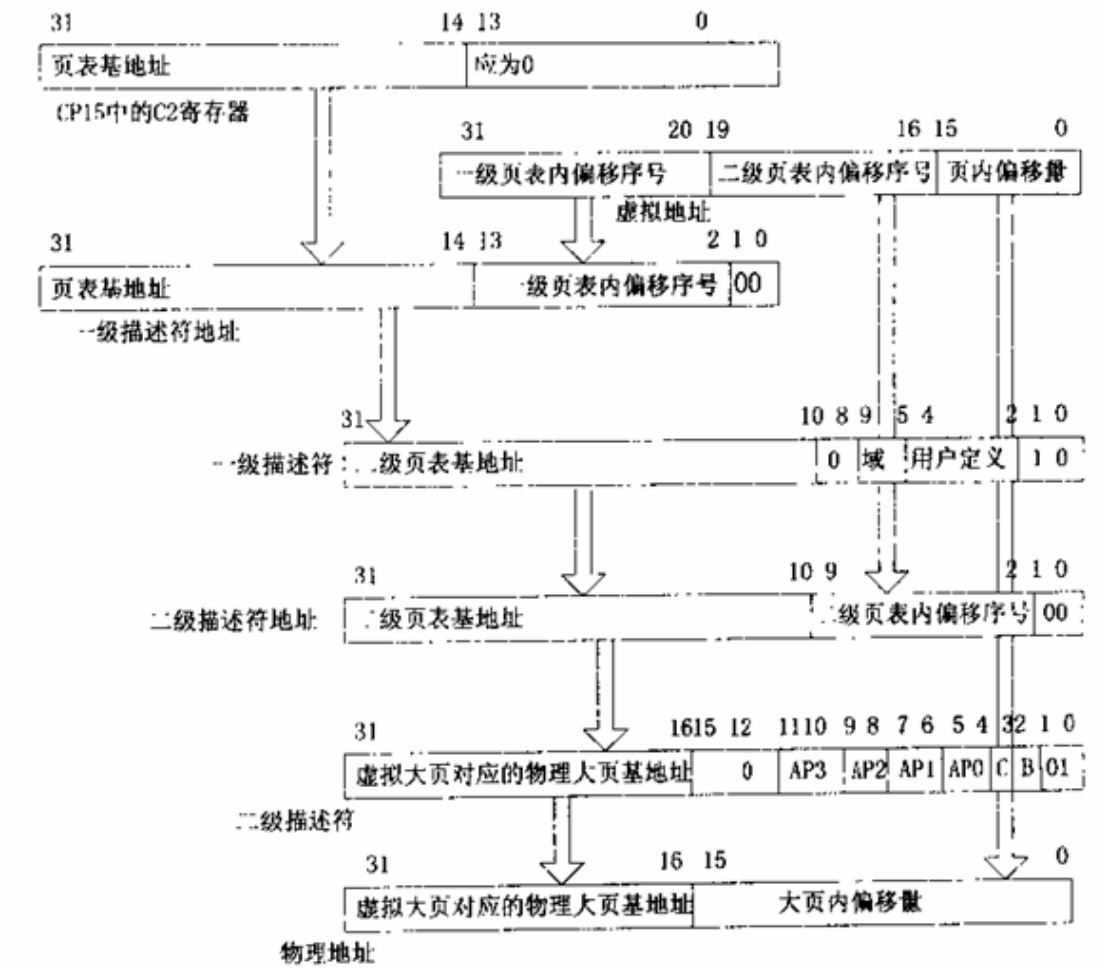


图 10.5：大页地址变换过程

小页描述符以及相关的地址变换

当页描述符的位[1：0]为 10 时，该二级描述符为小页描述符。小页地址变换过程如图 10.6 所示。

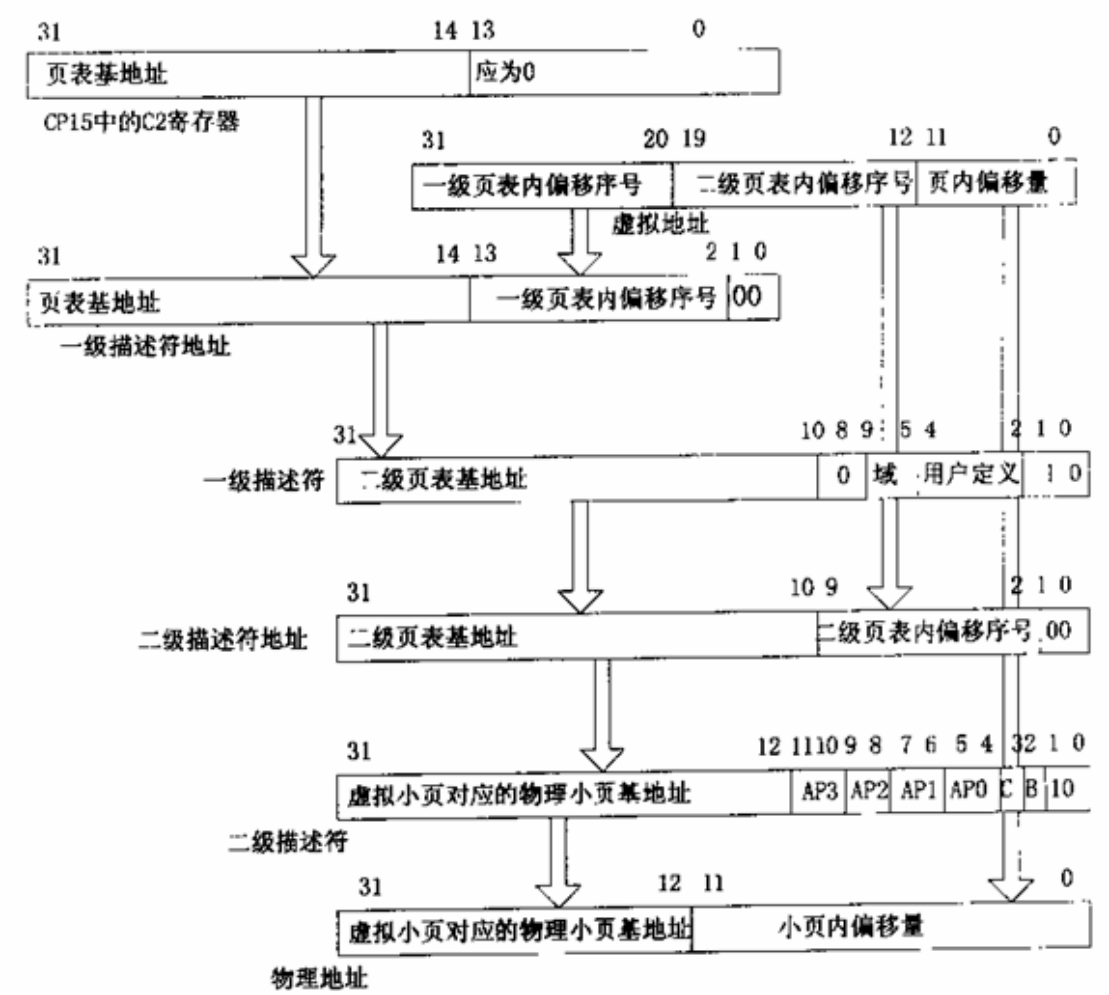


图 10.6：小页地址变换过程

极小页描述符以及相关的地址变换

当页描述符的位[1：0]为 11 时，该二级描述符为极小页描述符，极小页地址变换过程如图 10.7 所示。

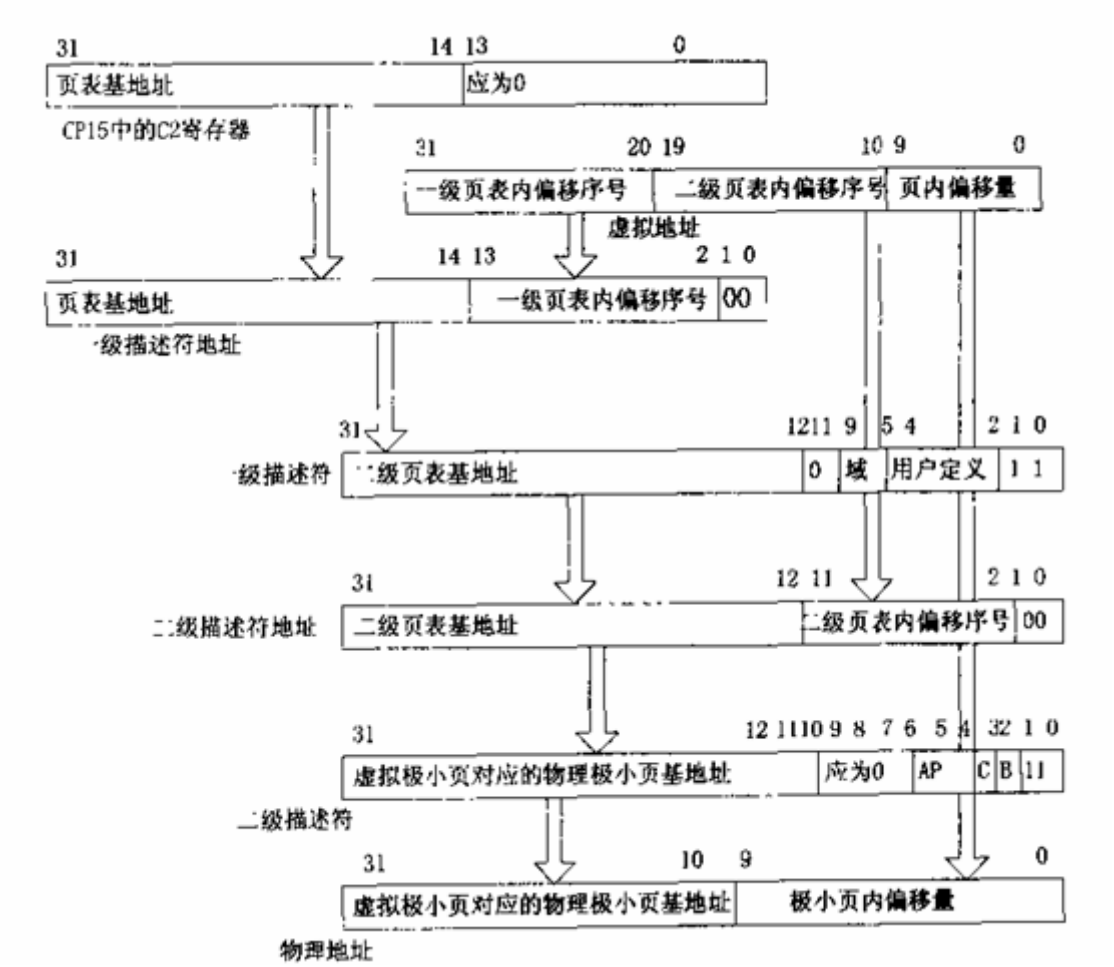


图 10.7：极小页地址变换过程

[实验内容]

1、通过 ADS 打开工程文件 MMU.mcp，进入 Target Settings 页面设置链接属性，－ro-base，－rw-base，－entry，－first，编译链接后产生 ELF 格式映像文件。

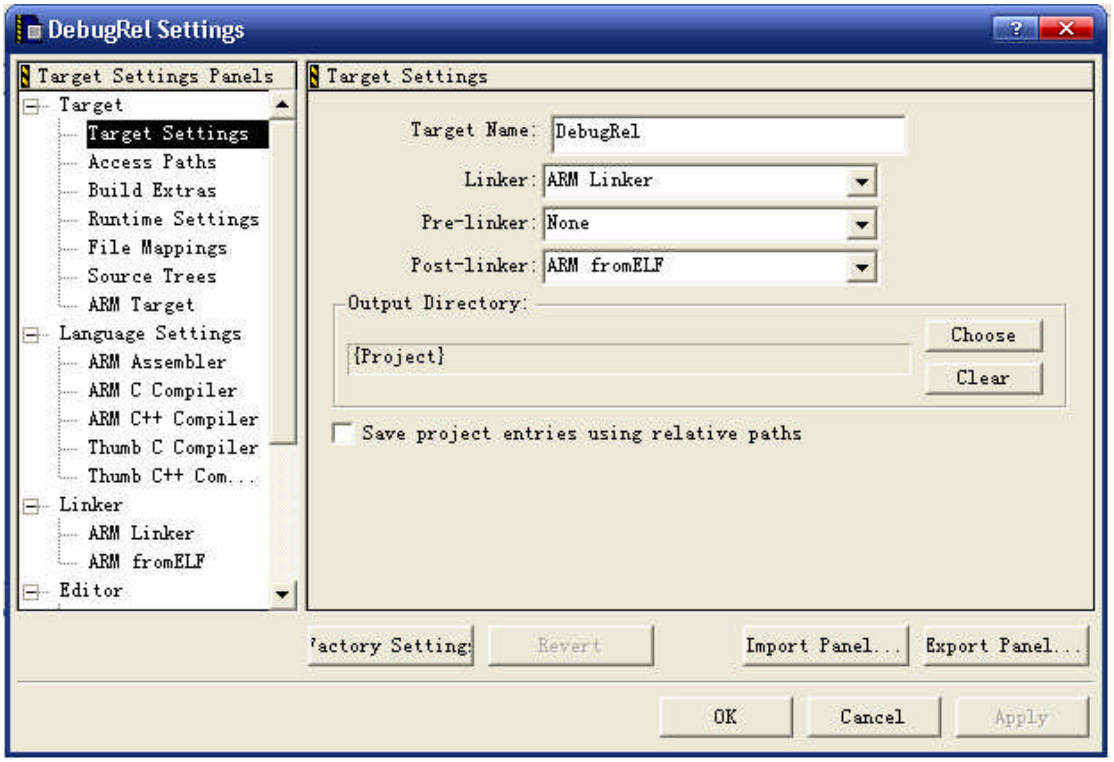


图 10.8：Target Settings

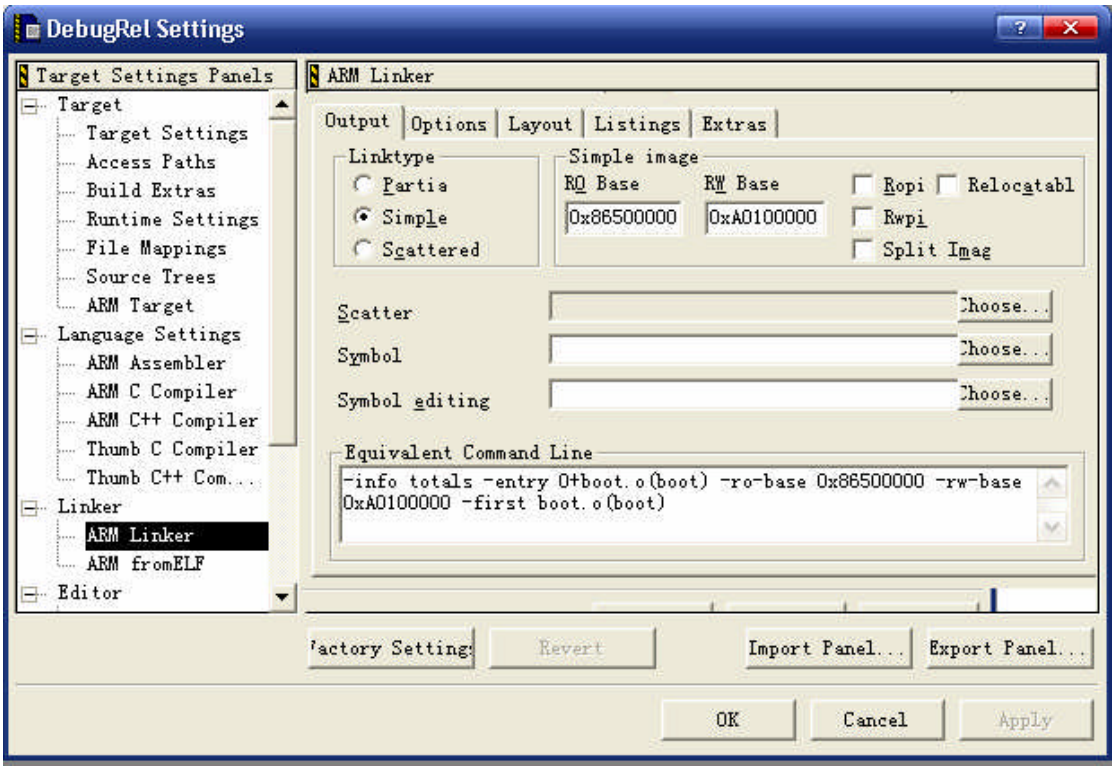


图 10.9：－ro-base，－rw-base

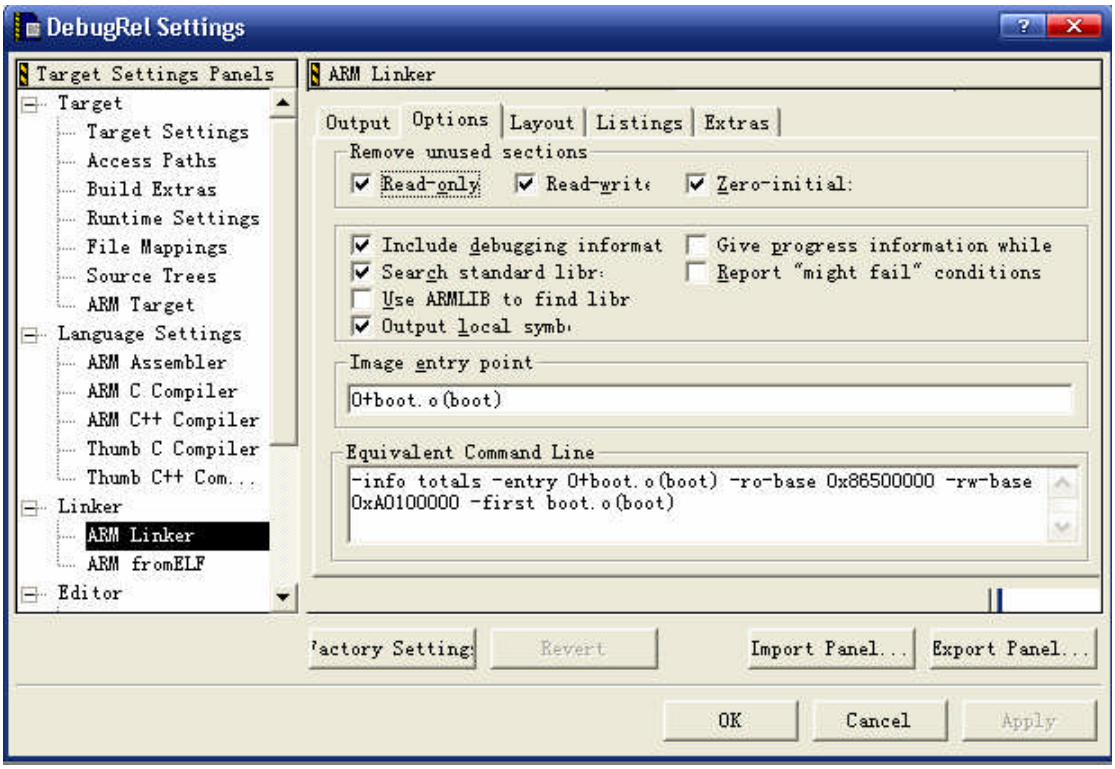

DebugRel Settings
Target Settings Panels
Target
Target Settings
Access Paths
Build Extras
Runtime Settings
File Mappings
Source Trees
ARM Target
Language Settings
ARM Assembler
ARM C Compiler
ARM C++ Compiler
Thumb C Compiler
Thumb C++ Com...
Linker
ARM Linker
ARM fromELF
Editor
ARM Linker
Output
Options
Layout
Listings
Extras
Remove unused sections
Read-only
Read-writ
Zero-initial
Include debugging informat
Search standard libr
Use ARMLIB to find libr
Output local symb
Give progress information while
Report "might fail" conditions
Image entry point
0+boot.o(boot)
Equivalent Command Line
-info totals -entry 0+boot.o(boot) -ro-base 0x86500000 -rw-base 0xA0100000 -first boot.o(boot)
actory Setting
Revert
Import Panel...
Export Panel...
OK
Cancel
Apply

图 10.10：—entry

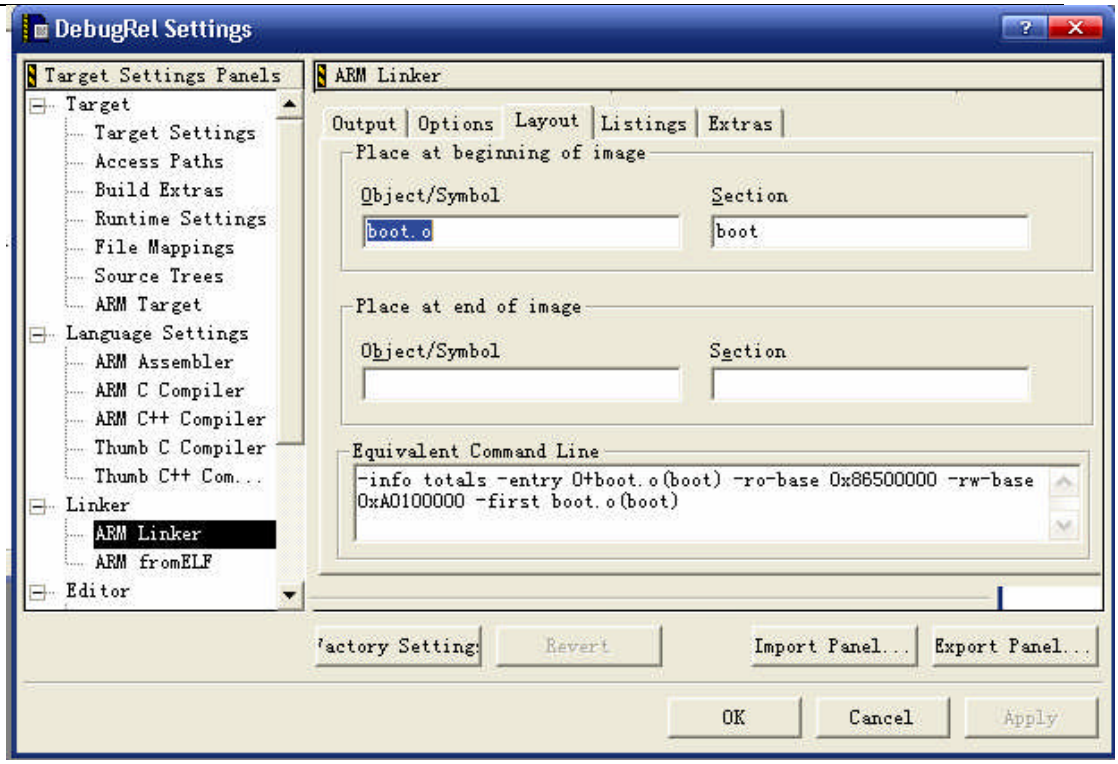


图 10.11：－first

2、编译链接后会产生 MMU.axf。

3、使用 ADS 的工具 fromelf 将 ELF 文件转化 BIN 格式的文件。该工具位于 ADS 的安装目录下，通常为 X:\Program Files\ARM\ADSv1_2\Bin，在 Dos 环境下执行如下命令：

fromelf -bin -output MMU.bin MMU.axf

或者使用 ADS 的 project→make 进行编译、连接生成映像。

4、用 JTAG 电缆连接主机和目标板，通过 JTAG 下载 BIN 格式的目标程序，执行如下命令： Jflashmm.exe pxa270 MMU.bin ，或者通过仿真器下载到目标板中。

5、将程序下载到 FLASH 后，通过串口线将开发板上调试串口 与 PC 的串口进行连接，对 PC 上的超级终端进行设置如图 10.12 所示。

图 10.12：超级终端设置

6、如果程序运行成功，则可以在串口上看到“Run in virtual memory mode!”一行字。

[习题与思考题]

1、本实验是介绍 MMU 的，那么如何禁止 MMU 功能？
2、当 CPU 访问内存地址空间时，先查找页表还是快表？为什么？
3、MMU 是如何实现虚拟地址到物理地址的映射的？
4、如何将 FLASH 的虚拟地址由 0x86500000 改为 0xb0000000？
5、可不可以将 FLASH 的虚拟地址改为 0xa0000000？为什么？

实验十一 Can Bus

[实验目的]

√ 了解 CAN BUS 总线规范；
√ 了解带有 SPI 接口的独立 CAN BUS 控制器 MCP2515 的结构与原理；
√ 掌握 CAN BUS 报文的发送原理。

[实验原理]

程序介绍

本章主要通过 CAN BUS 程序发送特定信息，利用 CAN232MB 智能协议转换器，把发送的信息转换到串口输出。本程序主要为了让读者能够清晰了解 CAN BUS 是如何初始化，发送信息的过程。

本程序得到的结果是 CAN BUS 不断向发送数据“11111111”出去，在发送过程中，同时注意到 CAN232MB 智能协议转换器的显示灯是不断闪亮的。

原理概述

(1) 什么是 CAN BUS 总线？

CAN 是 Controller Area Network 的缩写（以下称为 CAN），是 ISO 国际标准化的串行通信协议。 在当前的汽车产业中，出于对安全性、舒适性、方便性、低公害、低成本的要求，各种各样的电子控制系统被开发了出来。由于这些系统之间通信所用的数据类型及对可靠性的要求不尽相同，由多条总线构成的情况很多，线束的数量也随之增加。为适应“减少线束的数量”、“通过多个 LAN，进行大量数据的高速通信”的需要，1986 年德国电气商博世公司开发出面向汽车的 CAN 通信协议。此后，CAN 通过 ISO11898 及 ISO11519 进行了标准化，现在在欧洲已是汽车网络的标准协议。现在，CAN 的高性能和可靠性已被认同，并被广泛地应用于工业自动化、船舶、医疗设备、工业设备等方面。

（2）CAN BUS 总线协议规范

通信是通过以下 5 种类型的帧进行的。

- 数据帧
- 遥控帧
- 错误帧
- 过载帧
- 帧间隔

另外，数据帧和遥控帧有标准格式和扩展格式两种格式。标准格式有 11 个位的标识符（Identifier：以下称 ID），扩展格式有 29 个位的 ID。各种帧的用途如表 11.1 所示：

表 11.1：帧的种类及用途

帧	帧用途
数据帧	用于发送单元向接收单元传送数据的帧。
遥控帧	用于接收单元向具有相同 ID 的发送单元请求数据的帧。
错误帧	用于当检测出错误时向其它单元通知错误的帧。
过载帧	用于接收单元通知其尚未做好接收准备的帧。
帧间隔	用于将数据帧及遥控帧与前面的帧分离开来的帧。

(3) MCP2515 概述

MCP2515 是一款独立 CAN 控制器，可简化需要与 CAN 总线连接的应用。图 11.1 简要显示了 MCP2515 的结构框图。该器件主要由三个部分组成：

1) CAN 模块，包括 CAN 协议引擎、验收滤波寄存器、验收屏蔽寄存器、发送和接收缓冲器。

2) 用于配置该器件及其运行的控制逻辑和寄存器。

3) SPI 协议模块。

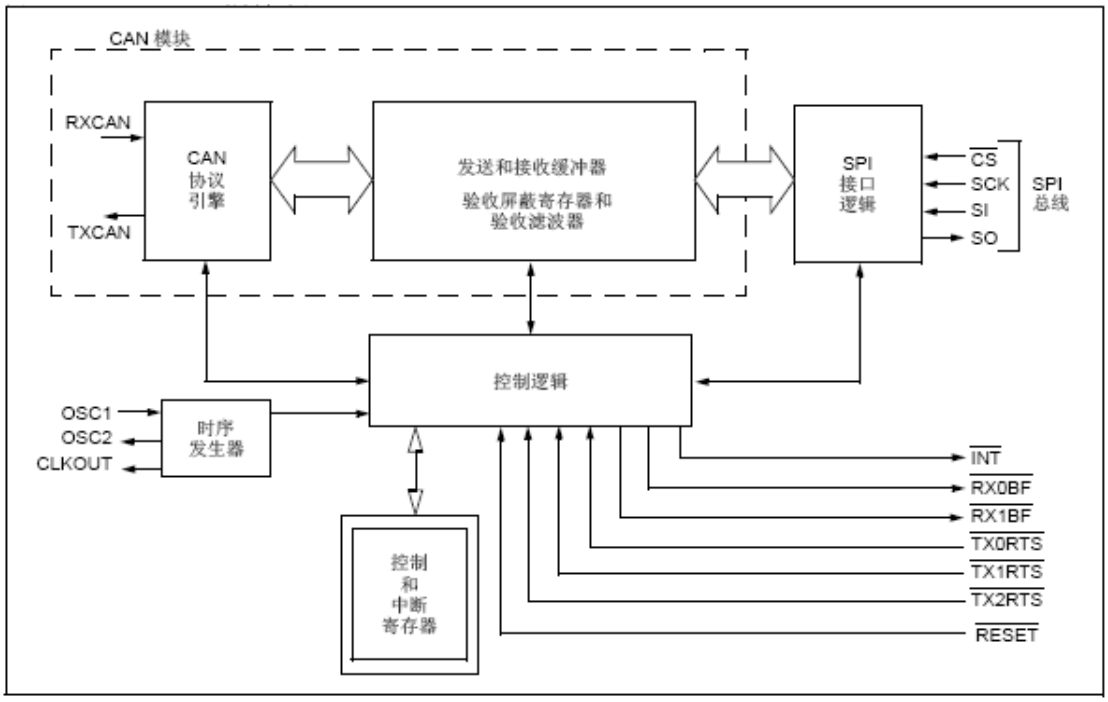


图 11.1：结构框图

CAN 模块:CAN 模块的功能是处理所有 CAN 总线上的报文接收和发送。报文发送时，首先将报文装载到正确的报文缓冲器和控制寄存器中。通过 SPI 接口设置控制寄存器中的相应位或使用发送使能引脚均可启动发送操作。通过读取相应的寄存器可以检查通讯状态和错误。会对在 CAN 总线上检测到的任何报文进行错误检查，然后与用户定义的滤波器进行匹配，以确定是否将报文移到两个接收缓冲器中的一个。

控制逻辑:通过与其他模块连接，控制逻辑模块控制 MCP2515 的设置和运行，以便传输信息与控制。所提供的中断引脚提高了系统的灵活性。器件上有一个多用途中断引脚及各接收缓冲器的专用中断引脚，用于指示有效报文是否被接收并载入接收缓冲器。可选择使用专用中断引脚。通用中断引脚和状态寄存器（通过 SPI 口访问）也可用来确定何时接收了有效报文。器件还有三个引脚，用来启动将装载在三个发送缓冲器之一中的报文立即发送出去。是否使用这些引脚由用户决定；若不使用，也可利用控制寄存器（通过 SPI 接口访问）来启动报文发送。

SPI 协议模块：MCU 通过 SPI 接口与该器件连接。使用标准的 SPI 读/写指令以及专门的 SPI 命令来读/ 写所有的寄存器。

(4) CAN 协议引擎

CAN 协议引擎包含数个功能模块，见图 11.2，下面将对这些模块及其功能进行介绍。

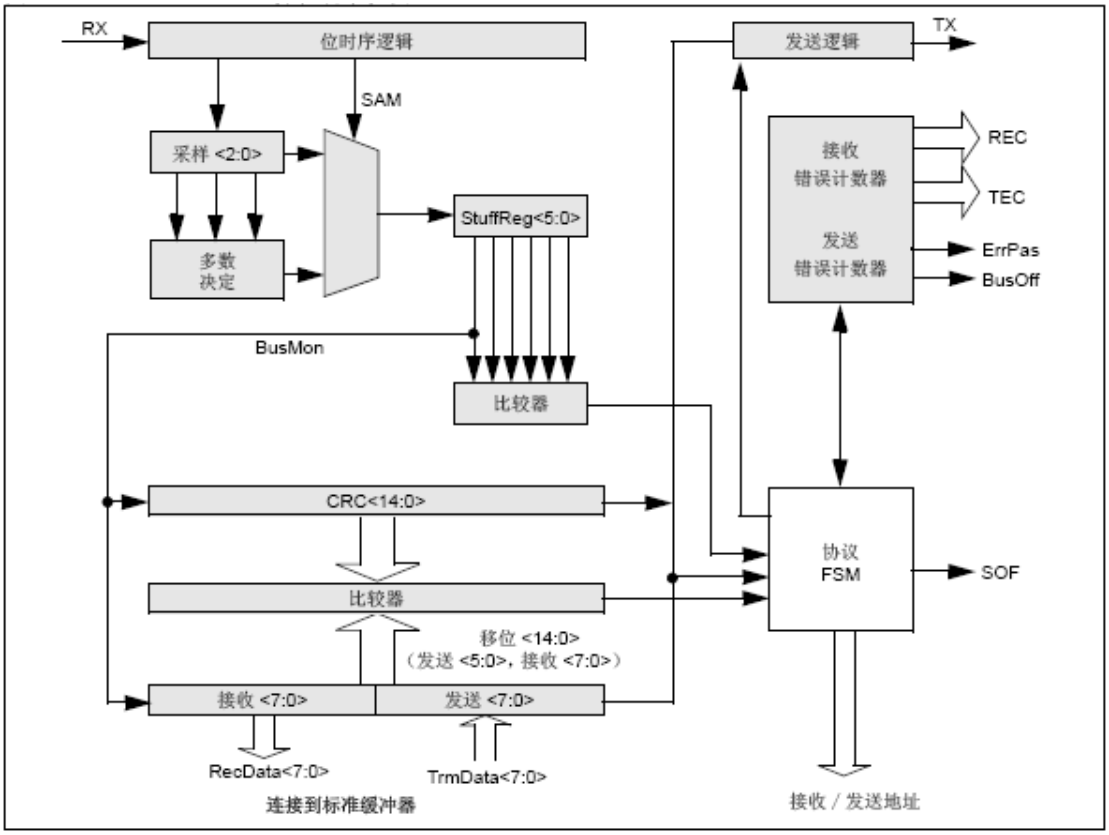


图 11.2：CAN 协议引擎框图

1）协议引擎的核心是有限状态机（Finite State Machine，FSM）。FSM 是一个定序器，对 TX/RX 移位寄存器、循环冗余校验（Cyclic Redundancy Check，CRC）寄存器和总线之间的顺序数据流进行控制。FSM 还对错误管理逻辑（Error Management Logic，EML）及 TX/RX 移位寄存器和缓冲器之间的并行数据流进行控制。FSM 确保了依据 CAN 协议，进行报文接收、总线仲裁、报文发送以及错误信号发生等操作过程。FSM 还对总线上报文的自动重发进行处理。

2）循环冗余校验寄存器产生循环冗余校验（CRC）代码。该代码在控制字段（数据字

节数为 0 的报文）或数据字段之后被发送，并用来检查进入报文的 CRC 字段。

3）错误管理逻辑（EML）负责将 CAN 器件的故障进行隔离。该逻辑有两个计数器，即接收错误计数器（Receive Error Counter， REC）和发送错误计数器（Transmit Error Counter， TEC）。这两个计数器根据来自位流处理器的命令进行增减计数。根据错误计数器的计数值，CAN 控制器将被设定为错误主动、错误被动或总线关闭。三种状态。

4）位时序逻辑（Bit Timing Logic， BTL）可监控总线输入，并根据 CAN 协议处理与总线相关的位时序操作。BTL 在起始帧时，对从隐性状态到显性状态的总线过渡进行同步操作（称为硬同步）。如果 CAN 控制器本身不发送显性位，则在以后的隐性状态到显性状态总线过渡时会再进行同步操作（称为再同步）。BTL 还提供可编程时间段以补偿传播延迟时间和相位位移，并对位时段内的采样点位置进行定义。对 BTL 的编程取决于波特率和外部物理延迟时间。

(4) MCP2515 的封装类型

其封装类型有以下两种：(如图 11.3 所示)

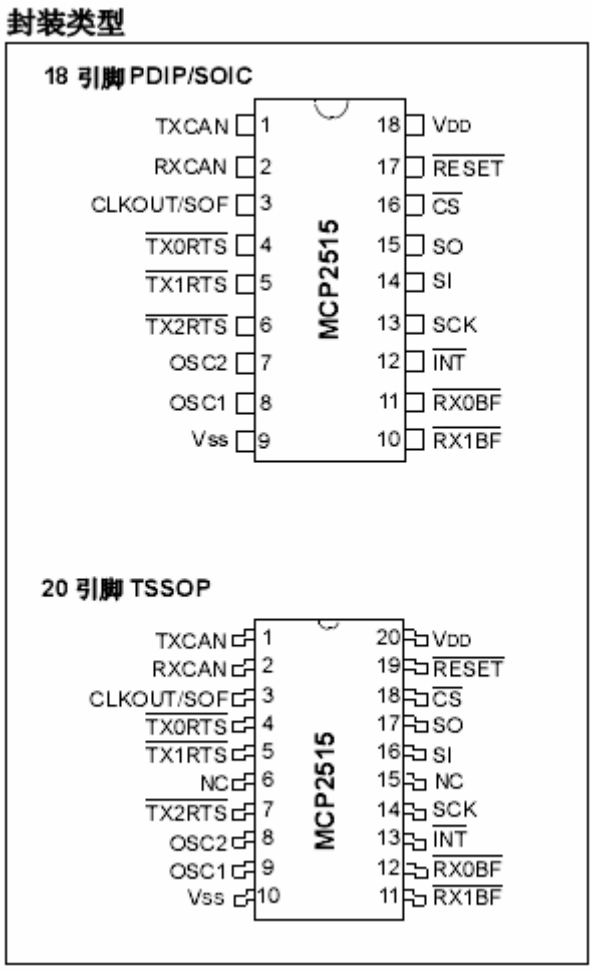


图 11.3：封装类型

(5) MCP2515 内部寄存器和工作原理

MCP2515 在 CAN 总线上的数据接收是通过两个接收缓冲器、两个接收屏蔽器、六个接收过滤器的组合来实现的。CAN 总线上的帧只有同时满足至少任意一个接收屏蔽器和一个接收过滤器的条件才可以进入接收缓冲器。

MCP2515 具有灵活的中断管理功能，它有 8 个中断源，包括发送、接收中断，各种错误中断以及总线唤醒中断等。单片机可以通过对 MCP2515 的中断允许控制寄存器 CANINTE 的设置来设定和屏蔽各种中断的发生条件，并可以通过读取 MCP2515 的中断标志位寄存器 CANINTF 或者通过 MCP2515 的 Read Status(读状态寄存器)命令读取 CANSTAT 寄存器中的

ICOD 部分来判断当前中断的中断源。

表 11.2 为 MCP2515 各个寄存器的地址与工作模式：

表 11.2：MCP2515 寄存器

CAN 地址	正常模式		配置模式	
	读	写	读	写
0x00~0x03	接收过滤寄存器 0		接收过滤寄存器 0	接收过滤寄存器 0
0x04~0x07	接收过滤寄存器 1		接收过滤寄存器 1	接收过滤寄存器 1
0x08~0x0B	接收过滤寄存器 2		接收过滤寄存器 2	接收过滤寄存器 2
0x0C	BF 引脚配置	BF 引脚配置	BF 引脚配置	BF 引脚配置
0x0D	发送请求控制		发送请求控制	发送请求控制
0xXE ①	状态寄存器	状态寄存器	状态寄存器	状态寄存器
0xXF ②	控制寄存器	控制寄存器	控制寄存器	控制寄存器
0x10~0x13	接收过滤寄存器 3		接收过滤寄存器 3	接收过滤寄存器 3
0x14~0x17	接收过滤寄存器 4		接收过滤寄存器 4	接收过滤寄存器 4
0x18~0x1B	接收过滤寄存器 5		接收过滤寄存器 5	接收过滤寄存器 5
0x20~0x23	接收屏蔽寄存器 0		接收屏蔽寄存器 0	接收屏蔽寄存器 0
0x24~0x27	接收屏蔽寄存器 1		接收屏蔽寄存器 1	接收屏蔽寄存器 1
0x28	位定时 3		位定时 3	位定时 3
0x29	位定时 2		位定时 2	位定时 2
0x2A	位定时 1		位定时 1	位定时 1
0x2B	中断屏蔽	中断屏蔽	中断屏蔽	中断屏蔽
0x2C	中断标志	中断标志	中断标志	中断标志
0x2D	错误标志	错误标志	错误标志	错误标志
0x30~0x3D	发送缓冲器 0	发送缓冲器 0	发送缓冲器 0	发送缓冲器 0
0x40~0x4D	发送缓冲器 1	发送缓冲器 1	发送缓冲器 1	发送缓冲器 1
0x50~0x5D	发送缓冲器 2	发送缓冲器 2	发送缓冲器 2	发送缓冲器 2
0x60~0x6D	接收缓冲器 0	接收缓冲器 0	接收缓冲器 0	接收缓冲器 0
0x70~0x7D	接收缓冲器 1	接收缓冲器 1	接收缓冲器 1	接收缓冲器 1

注：① 0xXE (0x0E, 0x1E, …, 0x7E) ② 0xXF (0x0F, 0x1F, …, 0x7F)

下面对几个重要的寄存器进行分析：

TXBnCTRL——发送缓冲器 n 控制寄存器 （地址：30h， 40h， 50h）

U-0	R-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0
—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0
bit 7							bit 0

bit 7　　未用：读为 0
bit 6　　ABTF：报文发送中止标志位
　　1 = 报文中止
　　0 = 报文发送成功完成
bit 5　　MLOA：报文仲裁失败位
　　1 = 报文发送期间仲裁失败
　　0 = 报文发送期间仲裁未失败
bit 4　　TXERR：检测到发送错误位
　　1 = 报文发送期间发生总线错误
　　0 = 报文发送期间未发生总线错误
bit 3　　TXREQ：报文发送请求位
　　1 =缓冲器等待报文发送
　　（MCU 将此位置 1 以请求报文发送－报文发送后该位自动清零）
　　0 =缓冲器无等待发送报文
　　（MCU 将此位清零以请求中止报文发送）
bit 2　　未用：读为 0
bit 1-0　　TXP：发送缓冲器优先级<1:0>位
　　11 = 最高的报文发送优先级
　　10 = 中偏高的报文发送优先级
　　01 = 中偏低的报文发送优先级
　　00 = 最低的报文发送优先级

图注：			
R = 可读位	W = 可写位	U = 未用位，读为 0	
-n = 上电复位时的值	1 = 置 1	0 = 清零	x = 未知值

TXRTSCTRL——TXnRTS 引脚控制和状态寄存器 （地址：0Dh）

U-0	U-0	R-x	R-x	R-x	R/W-0	R/W-0	R/W-0
—	—	B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM
bit 7							bit 0

bit 7　　未用：读为 0
bit 6　　未用：读为 0
bit 5　　B2RTS： TX2RTS 引脚状态位
- TX2RTS 为数字输入模式时，读出值为该引脚的电平
- TX2RTS 为请求发送模式时，读为 0
bit 4　　B1RTS：TX1RTX 引脚状态位
- TX1RTX 为数字输入模式时，读出值为该引脚的电平
- TX1RTX 为请求发送模式时，读为 0
bit 3　　B0RTS：TX0RTS 引脚状态位
-TX0RTS 为数字输入模式时，读出值为该引脚的电平
- TX0RTS 为请求发送模式时，读为 0
bit 2　　B2RTSM：TX2RTS 引脚模式位
1 = 该引脚用来请求 TXB2 缓冲器发送报文（在下降沿）
0 = 数字输入
bit 1　　B1RTSM：TX1RTS 引脚模式位
1 = 该引脚用来请求 TXB1 缓冲器发送报文（在下降沿）
0 = 数字输入
bit 0　　B0RTSM：TX0RTS 引脚模式位
1 = 该引脚用来请求 TXB0 缓冲器发送报文（在下降沿）
0 = 数字输入

图注：

R = 可读位	W = 可写位	U = 未用位，读为 0	
-n = 上电复位时的值	1 = 置 1	0 = 清零	x = 未知值

TXBnSIDL——发送缓冲器 n 标准标识符低位 （地址：32h， 42h， 52h）

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDE	—	EID17	EID16
bit 7							bit 0

bit 7-5 SID：标准标识符位 <2:0>
bit 4 未用：读为 0
bit 3 EXIDE：扩展标识符使能位
1 = 报文将发送扩展标识符
0 = 报文将发送标准标识符
bit 2 未用：读为 0
bit 1-0 EID：扩展标识符位 <17:16>

图注：

R = 可读位	W = 可写位	U = 未用位，读为 0	
-n = 上电复位时的值	1 = 置 1	0 = 清零	x = 未知值

TXBnDLC——发送缓冲器 n 数据长度码 （地址：35h， 45h， 55h）

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	RTR	—	—	DLC3	DLC2	DLC1	DLC0
bit 7							bit 0

bit 7 未用：读为 0
bit 6 RTR：远程发送请求位
1 = 发送的报文为远程发送请求
0 = 发送的报文为数据帧
bit 5-4 未用：读为 0
bit 3-0 DLC：数据长度码位 <3:0>
设定要发送的数据长度 （0 到 8 字节）
注： 可以将 DLC 设定为大于 8 的值，但只发送 8 个字节。

图注：

R = 可读位	W = 可写位	U = 未用位，读为 0	
-n = 上电复位时的值	1 = 置 1	0 = 清零	x = 未知值

TXBnDm——发送缓冲器 n 数据字节 m
（地址：36h - 3Dh， 46h - 4Dh， 56h - 5Dh）

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
TXBnDm 7	TXBnDm 6	TXBnDm 5	TXBnDm 4	TXBnDm 3	TXBnDm 2	TXBnDm 1	TXBnDm 0
bit 7							bit 0

bit 7-0　TXBnDM7:TXBnDM0：发送缓冲器 n 数据字段字节 m

图注：

R = 可读位	W = 可写位	U = 未用位，读为 0	
-n = 上电复位时的值	1 = 置 1	0 = 清零	x = 未知值

MCP2515 共有 5 种工作模式：配置模式、正常模式、睡眠模式、监听模式、和自检模式。工作模式的改变主要是通过控制器(0xXE)的 REQOP 位选择。当工作模式改变的时候，要等到所有的数据传输完毕后才能生效，因此在运行另一种工作模式之前，可通过查询状态寄存器(0xXF)的 OPMODE 位来确认已经进入该模式。

MCP2515 共有 128 个寄存器，地址由高三位和低四位确定，有效寻址范围在 0～0x7F 之间，某些专用的控制寄存器和状态寄存器可以通过 SPI 接口的 Bit Modify 命令进行位修改。

(6) MCP2515 与单片机的 SPI 接口与 SPI 指令

MCP2515 可与任何带有 SPI 接口的单片机直接相连，并且支持 SPI 1,1 和 0,0 模式。单片机通过 SPI 接口可以读取接收缓冲器数据。MCP2510 对 CAN 总线的数据发送则没有限制，只要用单片机通过 SPI 接口将待发送的数据写入 MCP2515 的发送缓存器,然后再调用 RTS(发送请求)命令即可将数据发送到 CAN 总线上。

在时钟 SCK 的上升沿，命令和数据通过 SI 引脚送入 MCP2515。在时钟 SCK 的下降沿，通过 SO 引脚把数据送出。操作中片选引脚 CS 保持低电平。

MCP2515 的 SPI 指令如表 11.3 所示：

表 11.3：MCP2515 的 SPI 指令表

指令名称	指令格式	指令功能
RESET	1100 0000	将内部寄存器复位成默认状态，工作模式进入配置模式
READ	0000 0011	从指定地址开始的寄存器中读取数据
Read Rx Buffer	1001 0nm0	从'nm'组合指定的接收缓冲器中读取数据
WRITE	0000 0010	从指定地址开始的寄存器中写入数据
Load Tx Buffer	0100 0abc	往'abc'组合指定的发送缓冲器中写数据
RTS	1000 0nnn	请求发送指令
Read Status	1010 0000	读取状态，包括发送接收中断标志和个请求发送位
RX Status	1011 0000	确定与接收到的报文和报文类型相匹配的过滤寄存器
Bit Modify	0000 0101	对指定的寄存器进行位修改

(7) MCP2515 报文发送

发送缓冲器

MCP2515 采用三个发送缓冲器。每个发送缓冲器占用 14 字节的 SRAM，并映射到器件存储器中。其中第一个字节 TXBnCTRL 是与报文缓冲器相关的控制寄存器，该寄存器中的信息决定了报文在何种条件下发送，并在报文发送时指示其状态。用 5 个字节来装载标准和扩展标识符以及其他报文仲裁信息，最后 8 个字节用于装载等待发送报文的 8 个可能的数据字节。

至少须将 TXBnSIDH、TXBnSIDL 和 TXBnDLC 寄存器装载数据。如果报文包含数据字节，还需要 TXBnDm 寄存器进行装载。若报文采用扩展标识符，应对 TXBnEIDm 寄存器进行装载，并将 TXBnSIDL.EXIDE 位置 1。在报文发送之前，MCU 应对 CANINTE.TXInE 位进行初始化，以便在报文发送时使能或禁止中断的产生。

注： 在写入发送缓冲器之前，必须将 TXBnCTRL.TXREQ 位清零（表明发送缓冲器无等待发送的报文）。

2）启动发送

通过将 TXBnCTRL.TXREQ 位置 1，可以启动相应缓冲器的报文发送。它可以按如下方式设定：

- 利用 SPI 写命令写寄存器
- 发送 SPI RTS 命令

• 将要发送报文的发送缓冲器的 TXnRTS 引脚置为低电平

通过 SPI 接口启动报文发送后，可以同时将 TXREQ 位和 TXP 优先级控制位置 1。当 TXBnCTRL.TXREQ 位置 1 后， TXBnCTRL.ABTF、TXBnCTRL.MLOA 和 TXBnCTRL.TXERR 位都将被自动清零。

注： 将 TXBnCTRL.TXREQ 位置 1 不会启动报文发送。仅会将报文缓冲器标记为准备发送。当器件检测到总线空闲时，才会启动报文发送。

报文发送成功后，TXBnCTRL.TXREQ 位将被清零，CANINTF.TXnIF 位置 1，若 CANINTE.TXnIE 位被置 1，将产生中断。如果报文发送失败， TXBnCTRL.TXREQ 将保持置 1，表明该报文仍在等待发送。此时以下条件标志之一将被置 1：

• 如果报文已开始发送但发生错误，TXBnCTRL.TXERR 和 CANINTF.MERRF 位将被置 1，此时在 CANINTE.MERRE 位置 1 后，器件将在 INT 引脚产生中断。

• 若发送报文总线仲裁失败，TXBnCTRL.MLOA 位将被置 1。

[实验内容]

1)分析代码

结合以上说明,对本实验所提供的汇编源代码进行分析,深入理解针对具体的硬件实现,软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图 11.4 所示：

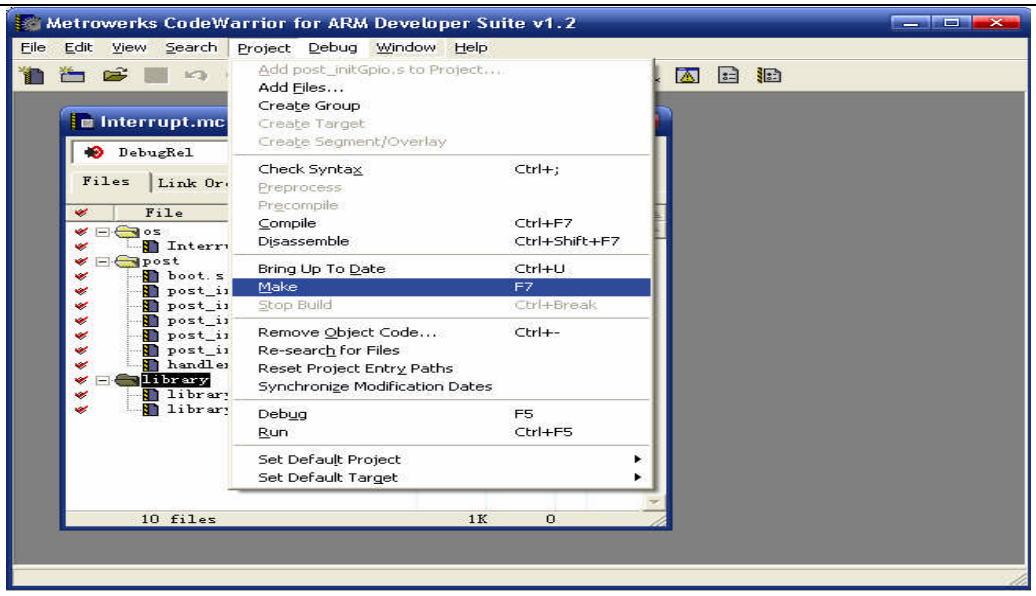


图 11.4 编译、连接生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)根据 CAN232MB 智能协议转换器手册的说明,连接好各个器件。设置超级终端如图 11.5 所示：

图 11.5：超级终端设置

4)如果成功，则可以在与 CANBUS 连接的串口中不断接收到“11111111”字符串，如果需要，也可以在程序中修改成其他字符串。

[习题与思考题]

1、在本程序中，CAN BUS 是不断向发送报文的，如何实现对每次发送信息后进行一个延时？

2、如何改用发送缓冲器 1 发送信息？

3、设置过滤掩码有什么作用？

实验十二 步进电机

[实验目的]

√ 理解步进电机的应用及相关概念
√ 掌握步进电机的原理及 GPIO 模拟脉冲的方法

[实验原理]

1、程序介绍

本章例子主要使用 GPIO 口进行模拟脉冲信号，来驱动步进电机转动。当系统启动后就会正转起来，一直运行下去。

2、步进电机（STEPPER-MOTOR）

步进电机是将电脉冲信号转变为角位移或线位移的开环控制元件。在非超载的情况下，电机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响，即给电机加一个脉冲信号，电机则转过一个步距角。这一线性关系的存在，加上步进电机只有周期性的误差而无累积误差等特点。使得在速度、位置等控制领域用步进电机来控制变的非常的简单。

虽然步进电机已被广泛地应用，但步进电机并不能像普通的直流电机，交流电机在常规下使用。它必须由双环形脉冲信号、功率驱动电路等组成控制系统方可使用。因此用好步进电机却非易事，它涉及到机械、电机、电子及计算机等许多专业知识。

EELIOD 系统的步进电机使用的是四相步进电机，采用的电机控制芯片是 Allegro 公司的UCN4202A,它包含低功率 CMOS 逻辑控制部分和达林顿管输出驱动极,最大输出电流为 1.5A,使用单相或双相，半步激励方式，内设续流二极管和过热保护电路。

UCN4202A 的控制功能包括 PWM 波输入，电机转动方向，输出使能和复位功能。OE 端使用 GPIO53 控制，为高时，电机没有输出；为低时 UCN4202A 开始工作。DIC 端为方向端，为低时为正向，为高时为反向。 UCN4202A 的逻辑控制有 ABCD 四个相位，在正向时，单相激励的顺序是 A-B-C-D，两相激励的顺序是 AB-BC-CD-DA，而半步激励的顺序是

A-AB-B-BC-C-CD-D-DA。当为反向时，同理就是从 D 相开始。

当内部结温接近 165 度时，过热保护电路起作用，将全部输出关断，当结温冷却到 145 度时候，重新使输出恢复正常。PWM 波输入由 GPIO83 脚完成，电机的转动电压为+12V，当电机转动的时候，发光二极管 D4 会发光。

3、电路分析

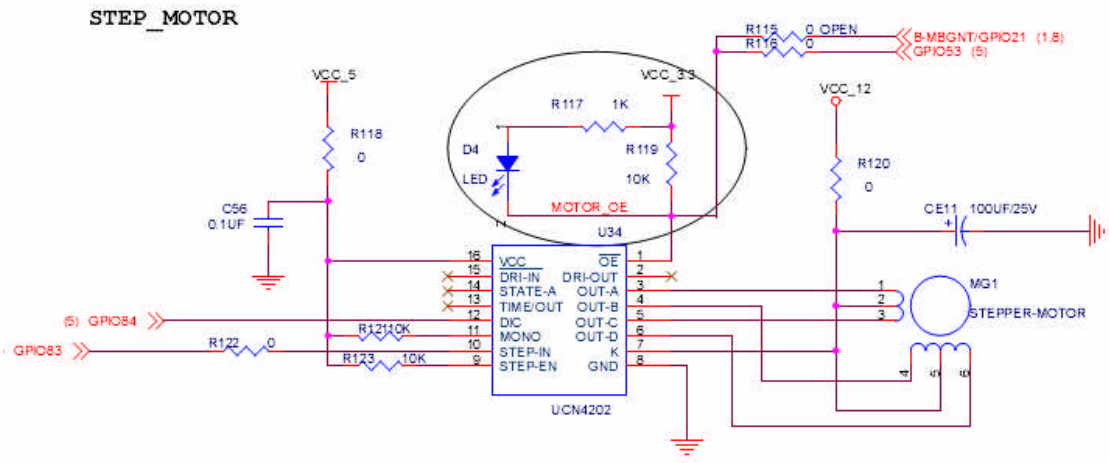


图 12.1 步进电机电路图

OE：这个管脚控制整个 UCN4202 的使用，如果 OE 为高电平，整个 UCN4202 就不能工作了，这个管脚为由 GPIO 的 53 控制，当输入为低电平是 LED 就会点亮。

STEP-EN:管脚 9 必须保持为高电平才能使能步进脉冲来驱动传输逻辑时钟电路，如果为低电平就会抑制传输逻辑。

STEP-IN：管脚 10 通常为高电平。当来了一个低电平后，逻辑就会向正方向移动一个位置。通常为方波，最少的周期为 2 微秒，占空比为 1。

DIC：旋转方向控制是由管脚 12 决定的，连接到 GPIO84。如果输入为高电平，旋转的方向为 A-D-C-B，如果输入为低电平，旋转方向为 A-B-C-D。

OUT-A，OUT-B，OUT-C，OUT-D：为输出的脉冲。具体如图 12.2 所示：

下面是电路的波形图：

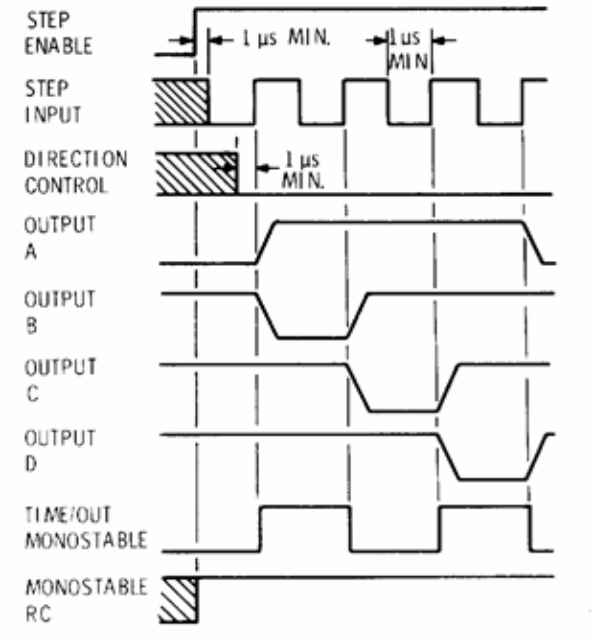
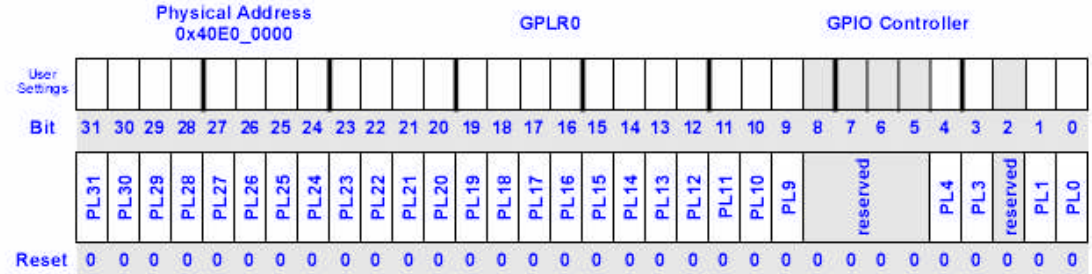


图 12.2 波形图

4、GPIO 口操作

在 EELiod 中共有 121 个 GPIO 口，我们可以通过 GPIO 寄存器设置 GPIO 的功能。下面介绍几个常用的寄存器：

GPLRx(0,1,2,3)：GPIO 电平寄存器是一个只读的寄存器，显示 GPIO 口的电平情况，0 时表示为低电平，1 时表示为高电平。

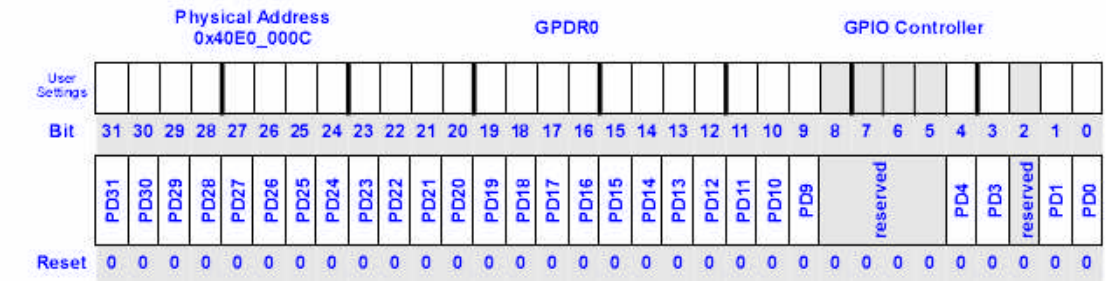


举例说明一下如何操作：

```
GPLR0    EQU    0x40e00000
LDR    R0, =GPLR0
```

这样，R0 就保存了 GPIO 从 0 到 31 的每个管脚的电平情况。

GPDRx(0,1,2,3)：GPIO 方向寄存器是一个可读写的寄存器，用来设置 GPIO 管脚的输入/输出情况，当某一位为 0 时，相应的 GPIO 作为输入，当某一位为 1 时，相应的 GPIO 作为输出。在默认的情况下，全部为 0，作为输入。

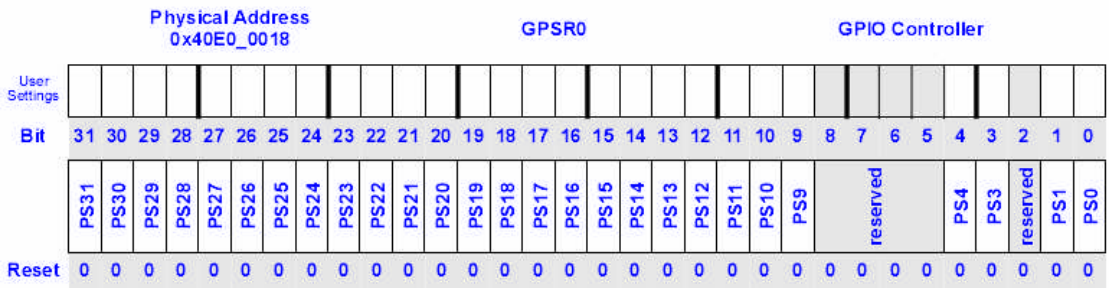


举例说明一下如何操作：

```
GPDR0    EQU    0x40e0000c
LDR   R1,=GPDR0
MOV   R0,#0x4758e004
```

这样就把 R0 的值 0x4758e004 写到了 GPDR0 寄存器，也就决定了每个管脚的输入/输出情况。

GPSRx(0,1,2,3):GPIO 管脚输出设置寄存器，这是一个只写的寄存器，用来设置定义管脚的高低电平。当为 1 时，设置为高电平输出，当为 0 时，设置为低电平输出。

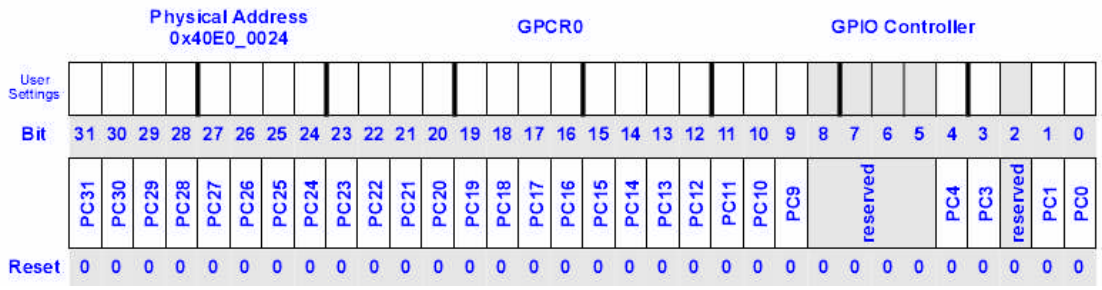


举例说明一下如何操作：

```
GPSR0     EQU    0x40e00018
MOV    R0, #0x4758e004
LDR    R1, =GPSR0
STR    R0, [R1]
```

这样就把 R0 的值写入到寄存器 GPSR0 中，也就决定了每个输出管脚的高低电平。

GPCRx(0,1,2,3):GPIO 输出清除寄存器，这是一个只写的寄存器，用来清除输出设置寄存器 GPSRx(0,1,2,3)的输出电平状态，恢复为低电平。写 1 时，清除输出，写 0 时，不改变状态。



举例说明一下如何操作：

```
GPCR0    EQU   0x40e00024
MOV   R0,#0x10000000
LDR   R1,=GPCR0
STR   R0,[R1]
```

这样就可以清除管脚 28 的输出高电平状态了，其他位不改变。

[实验内容]

1)分析代码

结合以上说明，对本实验所提供的汇编源代码进行分析，深入理解针对具体的硬件实现，软件是如何配合工作的。

2)程序的编译和下载

打开 ADS，执行 Project→Make ，也可以直接用快捷键 F7 进行编译、连接生成映像文件。如图 12.3 所示：

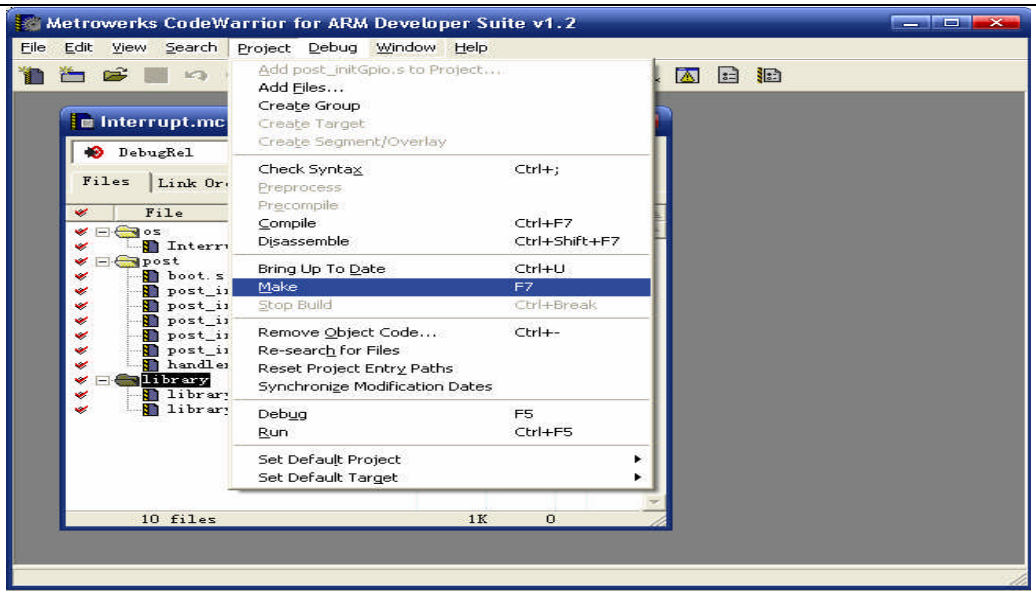


图 12.3 编译、连接生成映像

编译、连接后就生成映像文件，我们可以把它下载到 FLASH 或者 SDRAM 运行和调试。具体办法请查看文档---ADS 实验调试方法。

3)观察系统运行情况，对系统进行源码调试。

[习题与思考题]

1、简述步进电机的原理与分类。UCN4202 属于那种类型？
2、简述 GPIO 寄存器的功能。
3、如何改变步进电机的旋转方向？
4、在步进电机程序中，改变 Interval()函数参数的大小有什么结果？

附录一 嵌入式系统教学,科研开发平台－EELiod

快速掌握高端嵌入式系统开发,尽(仅)在 EELiod

随着芯片技术和电子产品智能化应用的飞速进展，嵌入式技术越来越受到人们的关注，应用领域遍及几乎所有的电子产品领域：智能机器人，网络通信，军用设备，汽车导航，环境保护，智能仪器,多媒体处理等等。 中国在计算机基础工业上落后于西方国家,在嵌入式处理器上也是如此。但是嵌入式系统面向应用的特点决定了处理器应用开发的产值要占有整个嵌入式工业的大部分,而且嵌入式系统的应用的开发只能由精通应用系统的用户来完成。因此中国在嵌入式系统方面存在着相当大的发展机会。我国在大专院校中针对工科院校学生开展单片机开发方面的教学工作已有 10 多年历史了,为国家培养了大量的单片机开发人员,创造了巨大的社会价值。但是由于嵌入式系统中软硬件技术的迅速更新,新技术在新产品中的大量采用（比如 32bit SOC，先进的开发方法及嵌入式操作系统的应用），使我们原有的教学内容与社会需求之间出现了脱节的情况。针对这些问题深圳市亿道电子技术有限公司为大专院校推出 XSBase255-ERD（目前嵌入式系统开发领域功能最强，提供配套资源最完整）教学科研开发平台，将有利推动院校嵌入式教学水平的提升。

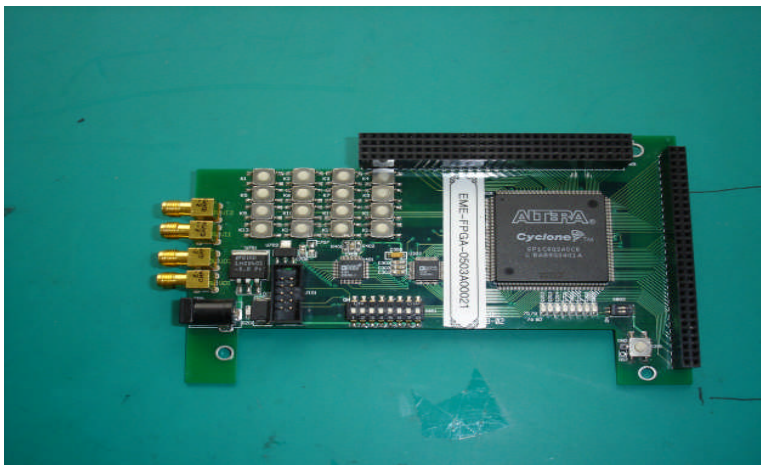
完整的嵌入式系统软硬件方案

亿道电子推出的 EELiod 基于 Xscale 体系结构的 PXA270 提供了一拥有 32 位数据总线,高达 520M 主频的功能强大的高端嵌入式开发系统。

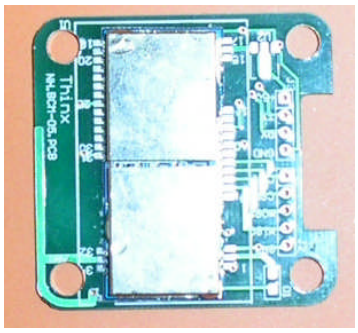
丰富的外设接口，满足多种应用领域的需求

Seria l Port *3	以 太 网口	IRDA	JTA G Port	Audio IN/OUT	PCMC IA	CF Port
MMC	6.4” TFT LCD	USB Host/Dev	RTC	Keypad	LED	120pin 扩展板

另外我们还提供 FPAG 拓展、AD/DA 拓展、VGA 支持、485 支持、蓝牙支持、PC104 支持等扩展模块。提供详尽原理图。



FPGA 子板



蓝牙模块

GSM/GPS/GPRS(三合

双操作系统解决方案，给您更多选择

提供业界使用最广的 Linux，WinCE 双操作系统解决方案，全部源代码开放不仅给您更多选择，而且同过对比研究，将使您更加深入，全面地了解嵌入式系统。提供 BSP 完整稳定。

内容详尽，深入浅出的配套教材及课件

为配合教学工作，亿道电子技术工程师精心打造 4 本配套资料，篇幅超过 800 页：《EELiod（WinCE）使用手册》、《EELiod(Linux)使用手册》、《实验理论指导书》和《实验上机指导书》。通过这些资料，教师将可以以理论讲学为基础，配合近 30 个具体的上机实

验指导，由浅入深让学生掌握嵌入式开发手段与流程，深入了解嵌入式开发过程中常见技巧，掌握目前嵌入式开发领域最先进的软硬件技术并可迅速转入实际的项目与产品开发。

实验教学，科研，实际产品开发三合一，迅速学以致用

EELiod 不仅仅是一个非常好的嵌入式教学和实验平台，由于技术的先进性，使我们现有的很多前沿的科研工作可以在此平台上开发测试，如：媒体交换，移动计算等。此外，EELiod 很容易产品化。近三年来选用 EELiod 做参考设计的企业客户已成功完成了近百个产品的开发工作，其中绝大部分已经批量上市。这无疑很好的验证了 EELiod 软硬件平台优异的稳定性。

全面的配套服务，确保教学效果

为确保 EELiod 平台在教学中的实际使用效果。亿道电子建立了强大的技术支持队伍，为师生提供全方位的技术服务：

1. 现场安装与师资技术培训
2. 24 小时问题响应制度
3. 不断丰富和优化的软硬件及手册文档，升级服务
4. 多样的支持手段：邮件，电话，BBS 论坛（www.xsbase.com），用户现场
5. 不定期专题培训与回访制度
6. 一年系统维护服务

附录二 亿道电子技术有限公司简介

公司简介

亿道电子技术有限公司创建于改革开放前沿的经济特区——深圳，是一家正在高速成长的高科技企业，它富有朝气、勇于创新，有着深厚的技术积累和行业经验。

亿道成立于 2002 年，公司自创建始，一直以专注、专业、协作、创新、关爱为企业理念，以团结、勤奋、务实、进取为企业精神；以人为本，给每个员工以广阔的发展空间，充分发挥每个员工的智慧和才能，形成了公司良好的企业文化氛围；使公司在日益激烈的市场竞争中立于不败之地，在嵌入式行业这个朝阳产业中脱颖而出，并飞速发展壮大。公司员工多是新时代的技术菁英，亿道融合了年轻人的激情和创新精神，也有着专注（Focus）、专业(Expert)、协作(Cooperation)、关爱（Care）的优良传统。在社会公益事业上亿道更是不遗余力的奉献着一颗颗真挚的心，多次去到贫困地区资助失学的儿童，尽己绵薄之力去关心爱护社会上需要帮助的群体，关爱之风在亿道已形成风尚。

公司业务

亿道电子深刻分析了中国嵌入式行业的实际现状，结合自身优势，不断探索和创新，为客户提供嵌入式实时多任务操作系统、集成开发（编译）环境、开发板、在线仿真器、ROM 仿真器、编程器以及各种技术方案和技术服务。亿道为嵌入式领域的开发人员提供各种开发方案，提供最为完善的售前、售中、售后服务，客户群涉及通讯、工业控制及自动化、医疗仪器仪表、消费类电子产品、军方、航空航天等众多领域。提供的方案如高性能的软件开发平台、嵌入式软件产品、代码自动生成工具、代码仿真环境、实时在线仿真器、调试器、RTOS（实时多任务操作系统）等支持 Motorola、Intel、AMD、IDT、MIPS、ARM 不同系列的处理器。

亿道和 Intel、AMD、Philips、Samsung、Lynuxworks、Sophia、Ashling、Paradigm 等十几家国外知名厂商有着良好的合作关系，在嵌入式领域与国外领先技术的公司保持着高度的技术同步。

2004 年 9 月 1 日－3 日在北京月亮河度假村举行了英特尔®嵌入式系统高校研讨会，此次研讨会标志着英特尔在国内高校设立基于 XScale、EIA 技术嵌入式课程计划的全面启动，

也标志着亿道电子全面获得英特尔的认可和支持，其 XScale 系列大学教育产品全面进军高校市场。对国内高校嵌入式教育平台的丰富，推动基于 XScale 嵌入式课程的发展有着非常积极、深远的意义。亿道电子作为国内最大的英特尔 XScale 嵌入式方案第三方供应商，长期致力于英特尔 XScale 架构底层方案的提供和服务。

亿道拥有勇于创新、技术积累深厚的研发队伍，经验丰富的技术支持队伍和专业干练、富有朝气的销售队伍，并建立起研发、产品推广和技术支持的全方位的整合营销体系。亿道在国内外已经建立了良好销售网络，拥有众多的高校、研究所及知名企业客户，在嵌入式系统创下许多经典的成功案例，在业界享有很高的声誉。

亿道的追求是在嵌入式领域实现顾客的梦想，并依靠点点滴滴、锲而不舍的不懈追求，使亿道的业务蒸蒸日上，规模不断壮大。亿道必将成为嵌入式技术领域中最好的也是最全面的系统方案提供商。

企业文化

把企业做强做大，是任何一家有远大抱负企业的追求，更是一种企业发展的奋斗目标；同时，也是企业用户和产品供应商发展的一个共同目标。亿道电子在短短几年时间里，无论从公司规模、产值、客户群还是产品数量方面都有着飞速的发展，这一切都与公司的紧贴市场、定位准确的经营理念息息相关。

专注（Focus）

亿道将长期专注于嵌入式领域，致力于整合先进的嵌入式硬件、软件平台，系统方案，为客户提供优质高效的解决方案。亿道认为只有专注于自己的行业，并沿着目标坚持不懈的努力下去，才有可能在行业里面取得更大的成功。

专业（Expert）

亿道对于“专业”的理解是：对于嵌入式行业的技术和市场发展要有深刻、透彻的理解和认识；对于目标客户的需求，能够以敏锐的眼光捕捉到并把握住；对于瞬息变化的市场，要有比竞争对手更快捷的反应速度和更准确的第一反应；对于自己的用户，要能够提供更加专业化的服务；在企业内部，要运用专业化的管理，比竞争对手更高效。

亿道立志成为所在领域举足轻重的公司，以专业的精神和态度去理解客户的需求，所有的员工以专业的精神和态度为客户提供专业化的服务，并深入了解和剖析行业未来发展趋势，以最高端、最专业的姿态屹立于嵌入式行业。

协作(Cooperation)

亿道坚持开放与合作的原则，认为只有团队的同心协力才能创造更大的价值，良好的团队合作意识和沟通协调能力是亿道对员工的基本要求，也是亿道高速发展的一个重要因素。在亿道没有个人英雄，因为我们深深知道：没有团队的英雄是孤独和不堪一击的，英雄群集的团队才能所向披靡。亿道一直坚持与客户共赢的政策，持续关注合作伙伴的利益，建立长

期合作、共同成长的合作关系，与业界精诚合作、互惠互利，在利益平衡的基础上为客户创造更大的价值。

创新（Innovation）

在信息技术时代，业务的复杂性要求我们必须创新，任何一个公司只有具备敏锐的市场洞察力和创新精神才能走在时代的前沿，才能领先或者紧跟市场潮流进而获取更大的发展。

我们相信，作为商业环境中的一分子，要追求卓越、超越平凡就必须创新。未来竞争就是持续创造与把握不断出现的商机的竞争，也即重划新的竞争空间的竞争。创造未来比拼命赶超别人更富有挑战性。

亿道是一个年轻的团队，是一个具备创新精神的团队，拥有创新的思维能力，创新的技术与应用方案，创新的市场策略，创新的管理模式，才会创造性地满足市场多样的需求。

关爱（Care）

在亿道这个大家庭里，关爱不仅仅体现在同事之间的关心和爱护，更深深地体现在对社会的责任上。

附录三 高校嵌入式实验室建设方面的优势

1、技术优势

1）技术深度广度

亿道电子的嵌入式实验室方案采用 INTEL 公司的 XScale 520Mhz 微处理器（属于 ARM10 的架构），具备 LINUX 和 WINCE 两种操作系统。此嵌入式实验室方案的原型是源自亿道电子的针对企业的 XScale 平台，一般来说企业客户较学校客户面临更为迫切的技术深度和研发时间压力，所以亿道电子的 XScale 嵌入式实验室已经具备研发产品的复杂的高要求，这是某些纯粹的教学平台厂家无法企及的。

除了对于 XScale 技术本身的积累以外，亿道的研发工程师们还在系统周边等很多领域都取得了非常积极的成果，比如对于 FPGA 拓展、ADDA 拓展、NAND Flash 支持、IDE 支持，VGA 支持、485 支持、蓝牙支持、PC104 支持、大屏显示、无线通讯、MPEG4 等等。

2）技术成熟

亿道电子从 StrongARM、250 走到现在的 270 在英特尔的平台研发上历经了 5 年的时间，在这三年中亿道完善了产品、丰富了资源、锻炼了队伍。通过合作，客户可以一下子就获得亿道三年以来的成熟产品的技术积累。

国内众多知名高校已经购买我们的 EELiod 实验平台建立了嵌入式实验室，还有许多的高校客户、研究院所以及企业客户购买了我们的 EELiod 软硬件参考设计平台作为科研开发的参考设计平台。“不经历风雨，怎么见彩虹”，亿道实验平台的广泛使用说明亿道的产品经过了市场的考验，是一款技术成熟稳定的产品。

3）技术延续性

“专注”、“创新” 是亿道人遵循的理念中重要的两条，亿道作为国内知名的嵌入式方案提供商，将持续的专注于嵌入式研发和教学平台开发领域，为高校源源不断的提供新的技术、新的产品。

亿道成功地参加第六届中国国际高新技术成果交易会（链接如下），引起多家创投机构的关注，相信初具规模的亿道电子将通过创投机构的协助进一步加强自己的领先优势。

http://u0316057.f2.13939.org/news2/testnews/05090533/20041013160417.htm

2、针对教学做了众多工作

为配合教学工作，亿道技术工程师精心编写了 4 本配套资料，篇幅超过 600 页：EELiod（WinCE）使用手册，EELiod (Linux)使用手册，实验理论指导书，实验上机指导书。通过这些资料，教师将可以以理论讲学为基础，配合近 30 个具体的上机实验指导，由浅入深让学生掌握嵌入式开发手段与流程，深入了解嵌入式开发过程中常见技巧，掌握目前嵌入式开发领域最先进的软硬件技术并可迅速转入实际的项目与产品开发。

亿道的嵌入式实验平台提供开放式的程序接口、具有很好的拓展性，为学生预留了丰富的开发、扩展空间，通过 120pin 的拓展接口，学生可以做丰富的小电路与实验平台相连，这对于有步骤的锻炼学生动手能力非常重要。

提供两种完善稳定的操作系统支持，WinCE 代表当今最流行的首页操作系统，开发尤其迅速。而 Linux 代表当今最流行的开源操作系统，对学生研究操作系统底层的教学非常适合。这两种操作系统教学相辅相成，使学习过的学生很容易满足当前社会对嵌入式人才的需求条件。

3、得到英特尔的技术认可

亿道电子是目前国内最大的 XScale 方案提供商和开发工具的供应商厂商，得到英特尔的认可，并作为成功方案提供商在 Intel 网站上推介：

http://www.intel.com/cn/gb/wireless/case/index_3.htm

应邀参加了 2004、2003 年度英特尔在大陆的 IDF 大展。

http://www.emdoor.com/news2/testnews/05090533/200459234234.htm

http://www.emdoor.com/news2/testnews/05090533/20031211215115.htm

以上几点都表明了英特尔对亿道技术的认可，并且亿道和英特尔大学合作部、英特尔软件大学都有密切的合作关系。对于有相关技术需求和密切合作的高校，亿道还将赠送英特尔的 XScale 优化软件，可进一步加深学生对嵌入式优化的认识。

4、服务优势

为客户精心服务是亿道奉行不渝的宗旨。任何时候，不管是提供设备给客户，还是探索一项新的技术、开发一项新的产品；不管是与客户交流、沟通，还是优化内部工作流程，亿道公司总是不断地回到最根本的问题－－客户的需求是什么。

关注客户需求，是亿道服务的起点，满足客户需求，是亿道服务的目标。对亿道来说，通过服务为客户创造价值，永远是第一位。

1）本地的服务

针对嵌入式教学内容改革大、师资要求高、服务量大的特点，亿道电子专门成立了大学合作部，在北京、上海、深圳三地都有大学合作部员工专门负责提供相关本地服务，直接辐射了华北、华东、华南等高校聚集的地区，西部办事处的筹集工作也在紧锣密鼓之中。

2）专业的技术支持队伍、严格的公司管理

亿道电子有一支高素质的专业技术支持队伍，成员均来自研发第一线。对于技术支持团队，公司有严格、规范的制度进行管理，从制度上保证了服务的及时性和可靠性。

现场安装与技术培训；

24 小时问题响应制度

不断丰富和优化的软硬件及手册文档，升级服务

周期性的回访制度

多样的支持手段：邮件、电话、BBS 论坛（www.xsbase.com）

用户现场

5、可以通过以下途径获得技术支持：

亿道 XScale 技术论坛：www.XSBase.com

专用 E-MAIL 技术支持：info@emdoor.com

提供电话、传真咨询的技术支持

3）研讨会、培训服务

亿道作为英特尔的合作伙伴，和英特尔大学合作部一起参加了众多的市场活动以及为众多知名高校的老师召开了 XScale 技术研讨会、培训班：

http://www.emdoor.com/news2/testnews/05090533/200491082405.htm

http://www.emdoor.com/news2/testnews/05090533/200498153343.htm

http://u0316057.f2.13939.org/news2/testnews/05090533/20041129214048.htm

针对不同的学校客户情况，亿道为高校授课老师提供不定期的专题培训，并积极地和各高校一起组织针对学生的兴趣讲座，深得广大师生欢迎。

4）服务的延续性 持续的合作

亿道作为一家有规模、业界知名的专业公司，服务的延续性、可靠性是有保证的。

亿道针对高校并不是一种单纯的先销售后售后支持的传统模式，而是积极地与老师互动，不仅仅在教学方面，在科研、著书等众多领域以及推动其与英特尔合作等方面都贡献自己的力量。

亿道还将持续推动合作高校在吸收英特尔 XScale 嵌入式软件领域所做的优秀成果及优化经验。

此外，每年寒、暑假合作高校还可以派老师（研究生）来亿道实习，和亿道工程师共同分享嵌入式系统开发的经验。

5）专业论坛

亿道创立了全国第一个专业的 XScale 技术 BBS 论坛（www.xsbase.com），并有专人负责巡视，及时回答相关问题。

6）共享课件联盟

亿道积极的倡导一个自由交流的共享课件联盟，“众人拾柴火焰高”，目前已有几个进行 XScale 教学高校积极参与此联盟。

附录四 高校嵌入式实验室建设方面的成功案例

1、浙江大学计算机学院

浙大计算机学院是全国首批进行 XScale 本科教学的院校，2002 年起就开始以选修课形式开课。2004 年初在英特尔的推荐下，浙大计算机学院进行了仔细的评估，最终亿道竞标成功，30 套亿道的 XScale 实验平台进入浙大计算机学院 2004 年的本科生教学。

2、西安电子科技大学电工实验室

西安电子科技大学电工实验室是全国级电工电子重点实验室，积极地进行优秀生选拔、差异性教育。选择了 23 套亿道的 XScale 实验平台率先在国内开始 XScale 课程，这种差异性教育使得其在 2004 年的全国大学生电子竞赛中把得头筹。

亿道上海多次和指导老师、优秀学生沟通，使得他们的课外设计丰富多彩，在亿道电子技术工程师的支持下，自行设计出 FPGA 和 AD/DA、键盘等等模块，对学生的创新研发能力得到了很大锻炼。

3、同济大学电信学院

同济大学电信学院是亿道电子的老客户，先采购了四套 XScale 参考设计平台，之后在建设嵌入式实验室的过程中又与亿道合作，选择了 15 套亿道的 XScale 实验平台率先在国内开始 XScale 课程。

除了教学合作以外，亿道给同济的年青老师在视频传输的科研项目上提供了力所能及的帮助，深得老师好评。

4、华南理工大学软件学院

华南理工大学软件学院是全国首批示范性软件学院之一，选择了 15 套亿道的 XScale 实验平台率先在国内开始 XScale 课程。

为了普及嵌入式基本知识、引导本科同学积极参加创新实验室活动，在华南理工大学老

师积极地组织下，亿道电子多次派工程师到华南理工大学进行兴趣讲座，老师和同学都受益匪浅。

更多信息，欢迎访问亿道电子网站 http://www.emdoor.com