三级存储
C语言交叉编译实现
Bootloader

19年计算机体系结构 高级实验答辩

1751740 刘鲲

2019/12

主要内容

- 第一部分：三级存储（纯硬件）
- **第二部分：C语言交叉编译实现Bootloader**

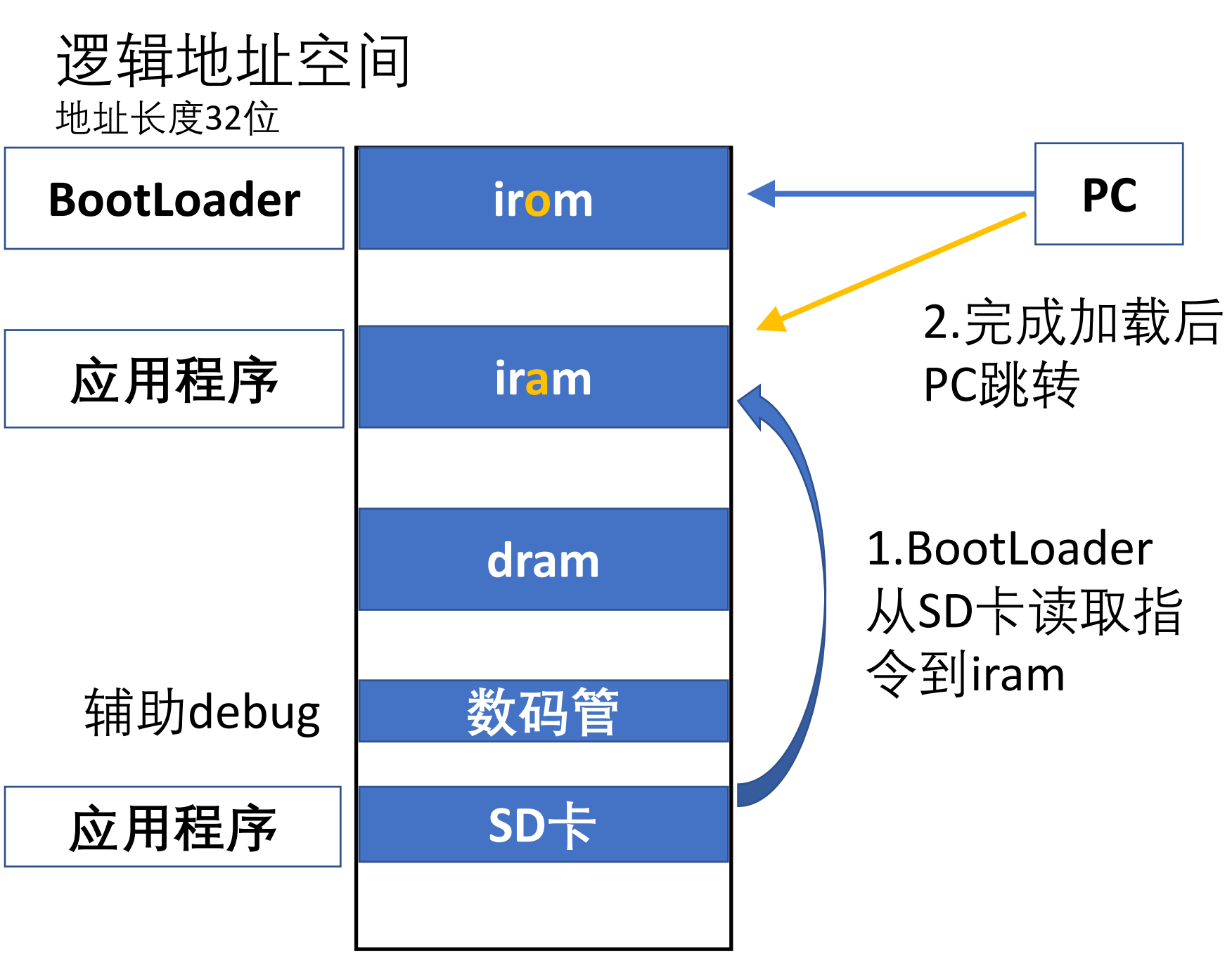
第二部分:

C语言交叉编译实现Bootloader
(暂无ddr2)

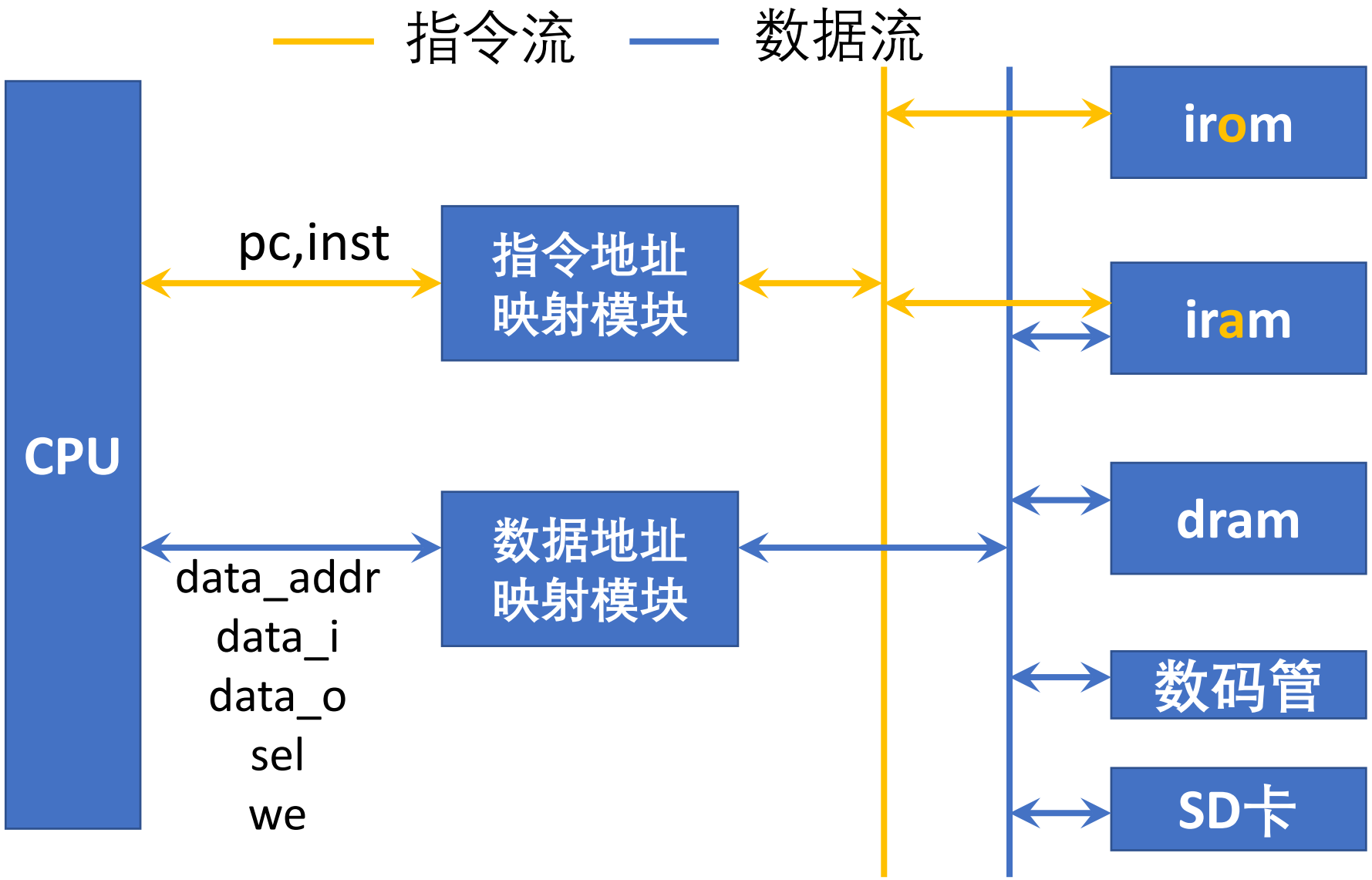
逻辑地址空间

地址长度32位


```
// 统一编址，逻辑地址，单位：Byte
`define InstRom_ADDR_BEG    32'h0
`define InstRom_ADDR_LEN    32'd2048

`define InstRam_ADDR_BEG    32'h0040_0000
`define InstRam_ADDR_LEN    32'd2048    //

`define DataRam_ADDR_BEG  32'h1001_0000 /
//`define DataRam_ADDR_LEN  `Seg7_ADDR_BE

`define Seg7_ADDR_BEG    32'h20000000
`define Seg7_ADDR_LEN    32'd8

`define SD_ADDR_BEG      32'h2200_0000
`define SD_ADDR_LEN      32'd4
```


逻辑地址空间
地址长度32位
BootLoader
irom
PC
应用程序
iram
2.完成加载后
PC跳转
dram
1.BootLoader
从SD卡读取指
令到iram
辅助debug
数码管
应用程序
SD卡

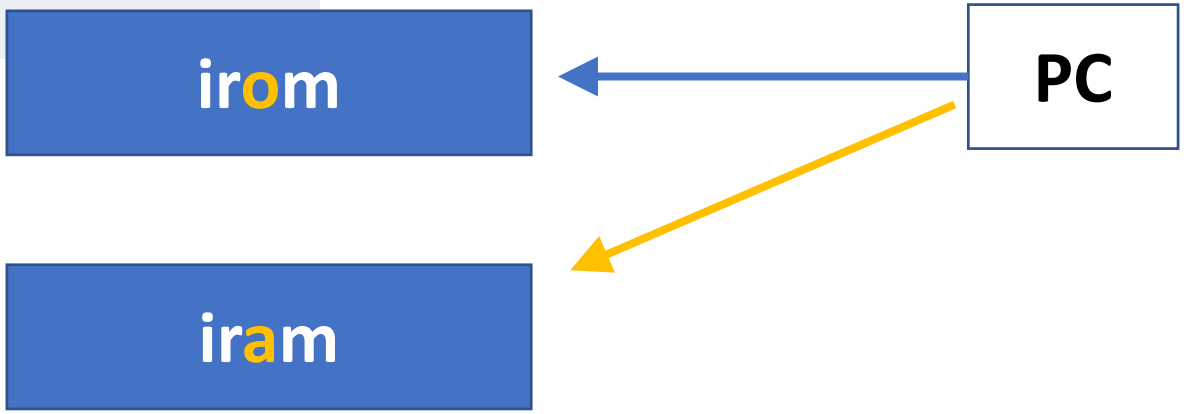
系统设计整体模块图


指令流
数据流
CPU
pc,inst
指令地址
映射模块
data_addr
data_i
data_o
sel
we
数据地址
映射模块
irom
iram
dram
数码管
SD卡

如何实现PC跳转？

```
.org 0x00000000
.global _start
.set noat
_start:
    lui $at,0x1000
    ori $at,$at,0x1F00
    add $sp,$zero,$at
    jal main
    nop
    lui $at,0x40    # 0040 0000
    jr $at
    nop
```

- Irom逻辑地址从0x0开始
- 在开头加一个跳转。跳转到到Bootloader的main函数
- main函数返回后，指令跳转到0x0040_0000，开始执行应用程序



SD卡管脚如何读写？(SPI模式)

- 管脚电平高/低=
对该管脚逻辑地址写1/0
- 需要**接口**

```
#define SD_CS 0x22000000
#define SD_CLK 0x22000001
#define SD_DATAIN 0x22000002
#define SD_DATAOUT 0x22000003
```

```
#define SD_HALF_CLK_LEN 0 //延时，这里是0，否则跟不上频率
// 宏定义
#define SD_CLK_UP()  *((uchar *)SD_CLK) = 1;

#define SD_CLK_DOWN()  *((uchar *)SD_CLK) = 0;

#define SD_DATAIN_UP()  *((uchar *)SD_DATAIN) = 1;

#define SD_DATAIN_DOWN()  *((uchar *)SD_DATAOUT) = 0;

#define SD_CS_UP()  *((uchar *)SD_CS) = 1;

#define SD_CS_DOWN() *((uchar *)SD_CS) = 0;
```

SD卡接口


```
module SD_soft(
    input clk,  //写入时钟
    input rst,
    input we,   //写使能
    input [3:0] sel_i, // 位选信号
    input [31:0] data_i, //写入数据
    output [31:0] data_o, // 将它视作一个4字节的有

    // sd相关
    output reg SD_cs, //片选, addr = 0
    output reg SD_clk, //时钟, addr = 1
    output reg SD_datain, //数据输入, addr = 2
    input SD_dataout //数据输出, addr = 3
);
```

- 上(左)部分：一个存储器该有的端口
- 下(右)部分：与板上SD卡管脚相连
- **统一编址目的：视同存储器，可用lw/sw指令"读写管脚"**

SD卡接口

```
assign data_o = {7'b0, SD_cs, 7'b0, SD_clk, 7'b0, SD_datain, 7'b0, SD_dataout};
always @ (posedge clk or posedge rst) begin
    if(rst) begin
        SD_cs = 1'b0;
        SD_clk = 1'b0;
        SD_datain = 1'b0;
    end
    else if(we) begin
        if (sel_i[3] == 1'b1) begin
            SD_cs <= (data_i[31:24] != 8'b0);
        end
        if (sel_i[2] == 1'b1) begin
            SD_clk <= (data_i[23:16] != 8'b0);
        end
        if (sel_i[1] == 1'b1) begin
            SD_datain <= (data_i[15:8] != 8'b0);
        end
    end
end
```

- 除了名字以外，与一个32位（4字节）存储单元没有区别
- 换言之，**对CPU来说，接口是存储单元，对FPGA来说，是管脚信号。**

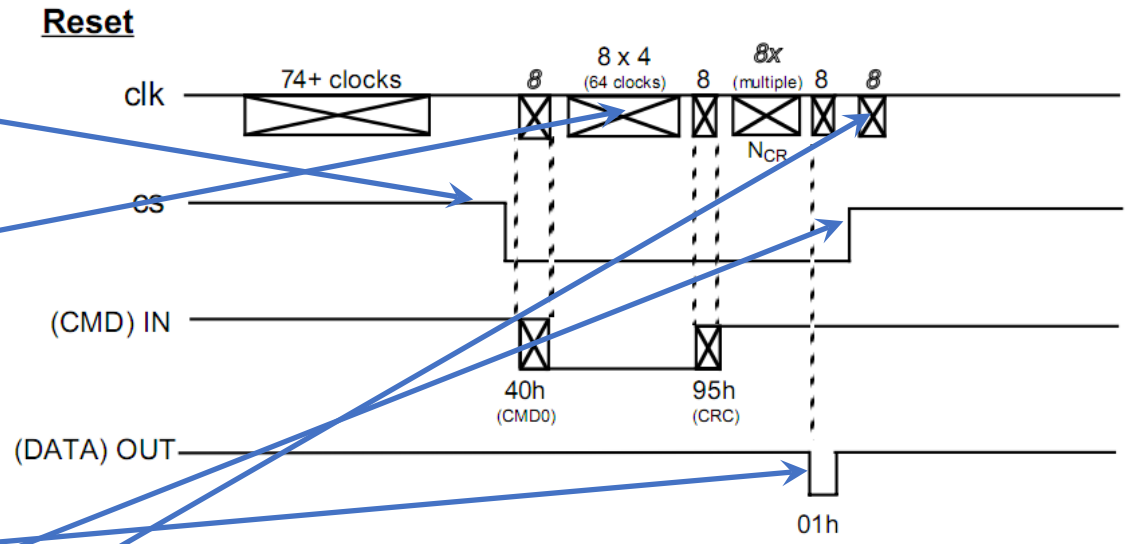
如何用软件发送时钟?

```
void SD_send_clk()
{
    SD_CLK_DOWN();
    DELAY_HALF_CLK();
    SD_CLK_UP();
    DELAY_HALF_CLK();
}
```

```
for (int i = 0; i < 80; i++)
    SD_send_clk();
```

如何撰写协议？

```
void write_CMD(ullong cmd, uchar *res, int n)
{
    SD_CS_DOWN();
    // 为什么宏定义的单元编译后是大端模式？
    // 高位在低地址？不懂，测试死我了
    uchar *p = (uchar *)(&cmd) + 2;
    for (int i = 0; i < 6; i++){
        SD_send_byte(*(p + i));
    }
    SD_DATAIN_UP();

    for (int i = 0; i < 8; i++) {
        SD_send_clk();
    }

    for (int i = 0; i < n; i++){
        res[i] = SD_get_byte();
    }

    if (*p != 0x51 && *p != 0x58) {
        SD_CS_UP();
        for (int i = 0; i < 8; i++)
            SD_send_clk();
    }
    //*((int *)SEG_DA) = *p; // debug
}
```



- 一一对应即可！

如何将读到的SD卡指令写进iram？

```
uint buffer[512];
uint *p_text = (uint *)0x00400000;
```

```
uint code_sector_cur = 8272;
SD_read_sector(code_sector_cur, buffer);  // 32'h0040_A000
```

```
for (int j = 0; j < 128; j++) //一个扇区128个字(512/4)，一个字4 Byte
{
    //*p_text = (buffer[idx + 3] << 24) | (buffer[idx + 2] << 16) | (buffer[idx + 1] << 8) | buffer[idx]
    *p_text = (buffer[idx] << 24) | (buffer[idx + 1] << 16) | (buffer[idx + 2] << 8) | buffer[idx + 3];
    *((uint *)SEG_DA) = *p_text;
    for(int k=0;k<1000000;k++) ///debug 可以看到指针
        ;
    p_text++;
    idx = idx + 4;
}
```

- 将p_text指向0040 0000
- 然后读扇区，数据存在buffer里
- 然后拼装，写入iram

撰写完毕后，make！

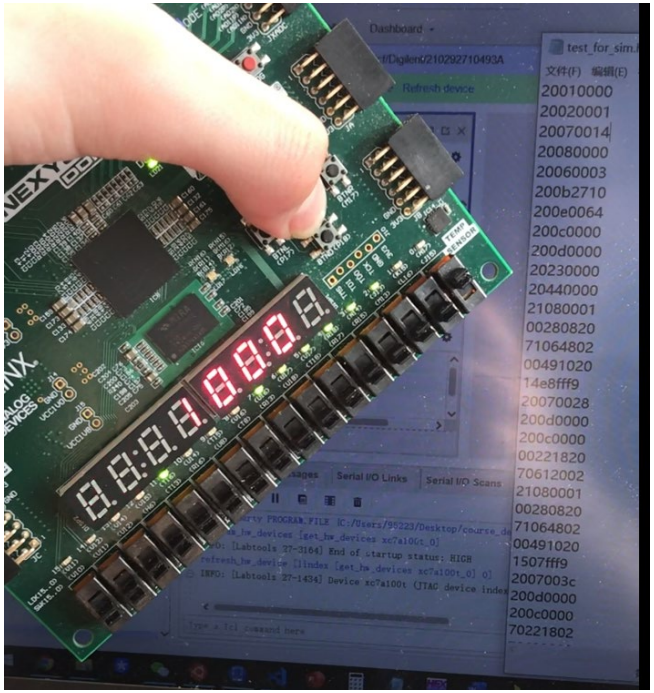
```
lk@DESKTOP-K4JAJDO:/mnt/c/Users/95223/Desktop/test_cross_compile$ make
```

```
.PHONY:clean
CROSS_COMPILE = mips-mti-elf
CC = ${CROSS_COMPILE}-gcc
AS = ${CROSS_COMPILE}-as
LD = ${CROSS_COMPILE}-ld
OBJCOPY = ${CROSS_COMPILE}-objcopy
OBJDUMP =  ${CROSS_COMPILE}-objdump
RM = del
COE = BOOT.coe

all: BOOT.c init.s
    $(CC) -c -std=c99 -mips32 BOOT.c
    $(AS) init.s -o init.o
    $(LD) -T ram.ld init.o BOOT.o -o BOOT
    $(OBJCOPY) -j ".text" -O binary BOOT BOOT.bin
    $(OBJDUMP) -b binary -m mips -D BOOT.bin -EB

    echo 'memory_initialization_radix = 16;' > ${COE}
    echo 'memory_initialization_vector =' >> ${COE}
    hexdump -v -e '4/1 "%02x" "\n"' BOOT.bin >> ${COE

clean:
    -$(RM) BOOT.o init.o BOOT BOOT.bin
```

```
8e4:    afc20014        sw      v0,20(s8)
8e8:    afc00020        sw      zero,32(s8)
8ec:    10000004        b       0x900
8f0:    00000000        nop
8f4:    8fc20020        lw      v0,32(s8)
8f8:    24420001        addiu   v0,v0,1
8fc:    afc20020        sw      v0,32(s8)
900:    8fc20020        lw      v0,32(s8)
904:    28422710        slti    v0,v0,10000
908:    1440fffa        bnez    v0,0x8f4
90c:    00000000        nop
910:    8fc2001c        lw      v0,28(s8)
914:    24420001        addiu   v0,v0,1
918:    afc2001c        sw      v0,28(s8)
91c:    8fc2001c        lw      v0,28(s8)
920:    28420080        slti    v0,v0,128
924:    1440ffdb        bnez    v0,0x894
928:    00000000        nop
92c:    00001025        move    v0,zero
930:    03c0e825        move    sp,s8
934:    8fbf022c        lw      ra,556(sp)
938:    8fbe0228        lw      s8,552(sp)
93c:    27bd0230        addiu   sp,sp,560
940:    03e00008        jr      ra
944:    00000000        nop
#./Bin2Mem.exe -f BOOT.bin -o BOOT.coe
echo 'memory_initialization_radix = 16;' > BOOT.coe
echo 'memory_initialization_vector =' >> BOOT.coe
hexdump -v -e '4/1 "%02x" "\n"' BOOT.bin >> BOOT.coe
lk@DESKTOP-K4JAJDO:/mnt/c/Users/95223/Desktop/test_cross_compile$
```

测试结果

- 上板后，七段数码管会显示读到的指令



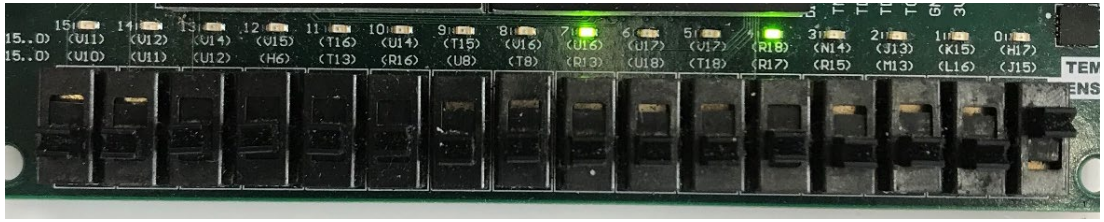
- 上图从录像中截取，由于手机录像和数码管频率的关系，闪烁较为厉害，直接下板肉眼观察更清晰
- 录像已提交

测试结果

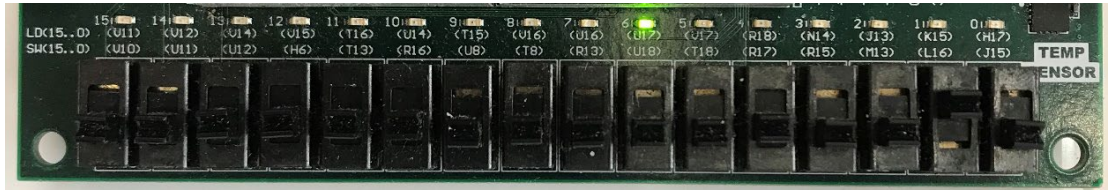
- 设置开关控制led
- Sw[1:0]==2时显示pc低16位；==1时显示pc高16位

```
assign led = sw[0] ?
             (sw[1] ? {debug_once_was, debug_i_data[14:0]} : inst_addr[15:0]): inst_addr[31:16];
```

- Sw[1:0]==2,pc[15:0]==0b0000_0000_1001_0000=0x0090;



- Sw[1:0]==1,pc[31:16]==0b0000_0000_0100_0000=0x0040;



测试结果

- 将结果拼凑起来:
- Pc=0x0040_0090

- 测试程序末尾无条件跳转:

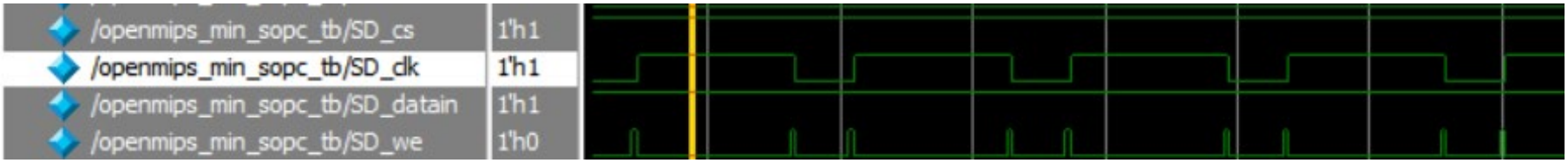
	0x00400088	0x00491020	add $2, $2, $9	82: add $2, $2, $9
	0x0040008c	0x14e8fff9	bne $7, $8, 0xfffffff9	83: bne $7, $8, Label3
	0x00400090	0x08100024	j 0x00400090	85: j correct

问题：SD卡时钟频率不足

```
void SD_send_clk() //
{
    Down_CLK();
    Delay_halfclk();
    Up_CLK();
    Delay_halfclk();
}
```

```
for (int i = 0; i < 8; i++)
    SD_send_clk();
```

- 波形图并不工整，因为for中有对i的操作，上升沿长



- 访存操作慢

```
58:   27bdfff0          addiu   sp, sp, -16
5c:   afbe000c          sw      s8, 12(sp)
60:   03a0f025          move    s8, sp
64:   3c022200          lui     v0, 0x2200
68:   34420001          ori     v0, v0, 0x1
6c:   a0400000          sb      zero, 0(v0)
70:   afc00000          sw      zero, 0(s8)
74:   10000004          b       0x88
```

问题：SD卡时钟频率不足

- 更换SD卡：有时行，有时不行

- 网上博客：**各个商家的同版本SD卡规则格式可能不同！**

问题：SD卡时钟频率不足

- **更换CPU**，再次升频，稳定成功

```
#define SD_HALF_CLK_LEN 0 //延时，这里是0，否则跟不上频率
```

```
clk_wiz_1 U_clk_div (
    // Clock out ports
    .clk_100m(clk_100m),     // output clk_100m
    .clk_10m(clk_cpu),     // output clk_10m
    // Status and control signals
    .reset(rst), // input reset
    // Clock in ports
    .clk_board(clk)
);      // input clk_board
```

1.要求的54条cpu不完整
2.要求的54条cpu和mips标准有差异！
jalr/jal，返回地址+4还是8
3.自己写的cpu频率太低

问题：大小端

```
.PHONY:clean
CROSS_COMPILE = mips-mti-elf
CC = ${CROSS_COMPILE}-gcc
AS = ${CROSS_COMPILE}-as
LD = ${CROSS_COMPILE}-ld
OBJCOPY = ${CROSS_COMPILE}-objcopy
OBJDUMP =  ${CROSS_COMPILE}-objdump
```

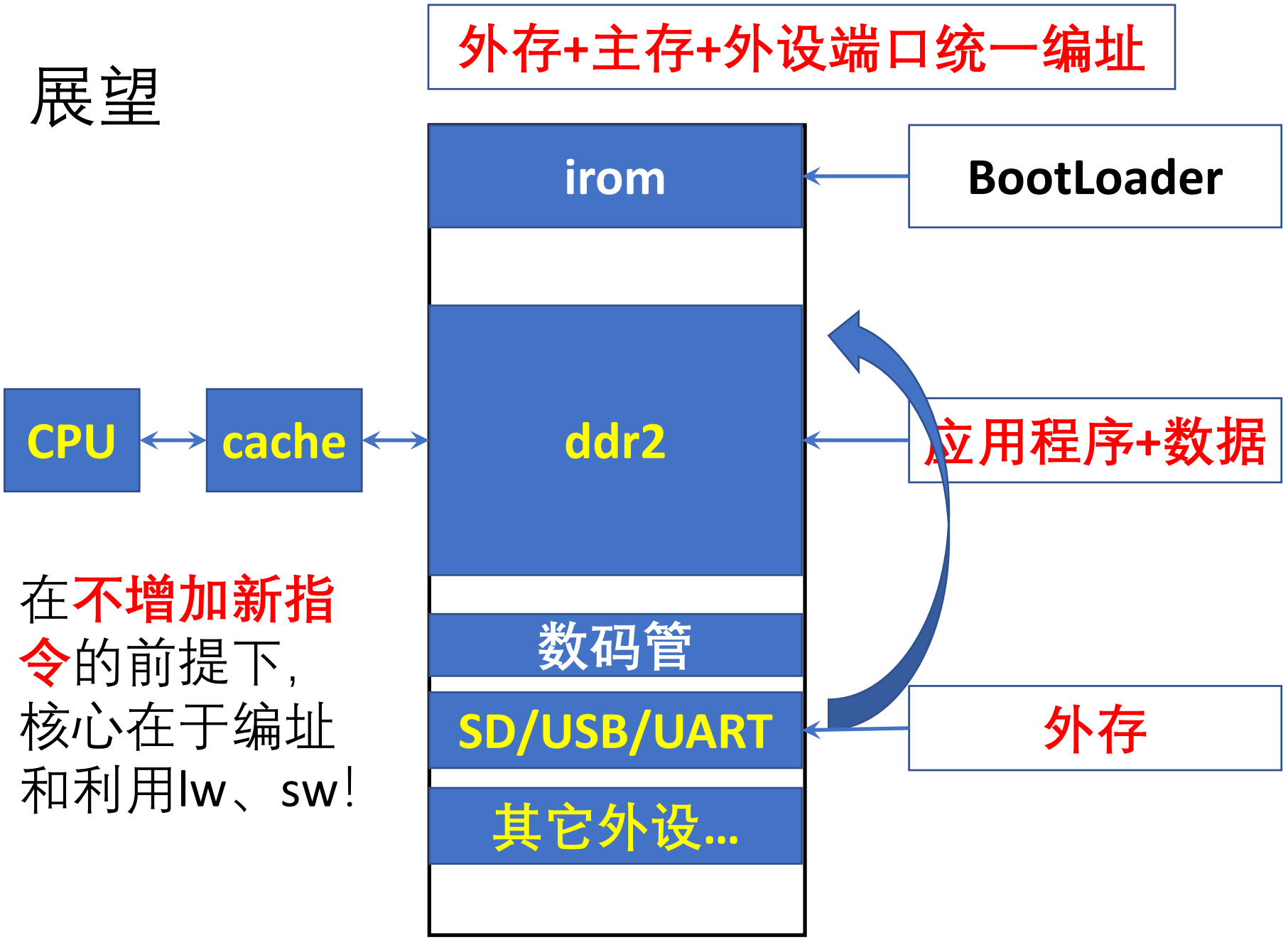
```
all: BOOT.c init.s
    $(CC) -c -std=c99 -mips32 BOOT.c
    $(AS) init.s -o init.o
    $(LD) -T ram.ld init.o BOOT.o -o BOOT
    $(OBJCOPY) -j ".text" -O binary BOOT BOOT.bin
    $(OBJDUMP) -b binary -m mips -D BOOT.bin -EB
```

```
// cmd
#define CMD0 0x400000000095ULL  // 6个Byte
```

```
uchar *p = (uchar *)(&cmd) + 2; //无解！
for (int i = 0; i < 6; i++){
    SD_send_byte(*(p + i));
}
```

- 分明都是小端编译，但是为什么反编译要大端？
- Modelsim波形图debug
- SD串口发送

展望



在**不增加新指令**的前提下，核心在于编址和利用lw、sw!

展望

- 外设协议尽可能用软件实现，开发速度快，**比Verilog写得舒服**，方便debug（如一句话显示数码管）

```
uint idx = 0;
for (int j = 0; j < 128; j++) //一个扇区128个字(512/4)，一个字4 Byte
{
    //*p_text = (buffer[idx + 3] << 24) | (buffer[idx + 2] << 16) | (buffer[idx + 1] << 8) | buffer[idx];
    *p_text = (buffer[idx] << 24) | (buffer[idx + 1] << 16) | (buffer[idx + 2] << 8) | buffer[idx + 3];
    *((uint *)SEG_DA) = *p_text;
    for(int k=0;k<1000000;k++) ///debug 可以看到具体指令
        ;
    p_text++;
    idx = idx + 4;
}
```

- FAT32文件系统（实验书有代码）

总结

- 对于有较**严格时钟频率**要求的**高速**设备
  - 内嵌汇编指令
  - 接口+硬件驱动，软件控制接口(如本实验数码管实现)
- **根本瓶颈：CPU频率是外设频率的上限！**
- **汇编结果难以预测！**(如本实验SD卡时钟)

第一部分

硬件实现三级存储

什么样的封装才是合理的封装?

- **我的结论：**
- **对于一个在一个边沿只能做读或者写的单口存储器，需要读、写数据线、写使能、地址线四根基本线；**
- **对于一次读写需要多个周期的，需要额外添加ack信号。**

Cache

```
// 第一级 cache
cache U_cache(
    .clk(clk100mhz),
    .a(cache_addr_toIMEM[8:0]),
    .d(cache_data_toIMEM),
    .we(cache_read_write),
    .spo(cache_data_fromIMEM)
);
```

无论是什么样的存储器都至少有的4个信号

信号名	含义
cache_addr_toIMEM	读写地址线
cache_data_toIMEM	写数据线
cache_data_fromIMEM	读数据线
cache_read_write	读写使能

SD(参考实验书)

```
module SD (
    input clk100mhz,
    input rst,

    // sd卡的读写信号
    input sd_mem_read_write,//useless 但
    input [31:0] sd_mem_addr,
    input [31:0] sd_mem_data_toSDMEM, //
    output reg sd_mem_ready,
    output [31:0] sd_mem_data_fromSDMEM,
```

信号名	含义
sd_mem_addr	读写地址线
sd_mem_data_toSDMEM	写数据线
sd_mem_data_fromSDMEM	读数据线
sd_mem_read_write	读写使能

+ sd_mem_ready 指令加载完成的标志。

DDR2(参考实验书)

```
module sealedDDR (
    input clk100mhz,
    input rst,
    input [31:0] addr_to_DDR,
    input [31:0] data_to_DDR,
    input read_write,
    output [31:0] data_from_DDR,
    output reg busy,
    output reg done,
    output ddr_start_ready,
```

根据之前得出的结论，由于ddr2不是一个周期就能读写的，所以一定要有req、ack信号，尤其是ack。在书上的代码的设计中，加上了如下信号：

信号	含义
busy	忙
done	上个操作已完成

基本信号：

信号名	含义
addr_to_DDR	读写地址线
data_to_DDR	写数据线
data_from_DDR	读数据线
read_write	读写使能

DataBus (参考实验书)

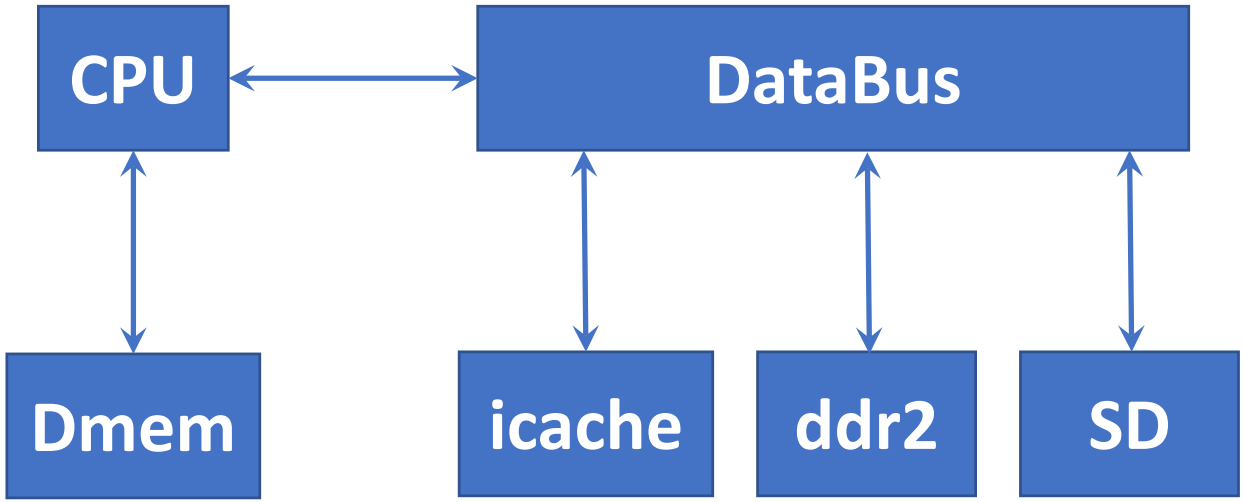
```
module DataBus (
    input clk100mhz,
    input rst,
    input [31:0] cpu_addr,
    input cpu_read_write,
    output reg bootdone,

    // 与CPU的交互信号
    output [31:0] cpu_data_toCPU,
    output reg DataBus_busy,
    output reg DataBus_done,

    // 与cache的交互信号
    input [31:0]cache_data_fromIMEM,
    output  [31:0]cache_addr_toIMEM,
    output  [31:0]cache_data_toIMEM,
    output reg cache_read_write,
```

```
    // 与DDR的交互信号
    input [31:0]ddr_data_fromDDR,
    input ddr_busy,
    input ddr_done,
    output [31:0]ddr_addr_toDDR,
    output [31:0]ddr_data_toDDR,
    output reg ddr_read_write,
    input ddr_start_ready,

    // 与SD卡的交互信号
    output reg sd_mem_read_write,
    output  [31:0]sd_mem_addr,
    input  [31:0]sd_mem_data_fromSDMEM,
    output reg [31:0]sd_mem_data_toSDMEM,
    input sd_mem_ready,
```

- 上一小结封装的模块的所有端口都和DataBus相连
- 其中有一组端口与CPU相连

从模块封装的角度出发，能够对总线布局有清晰的认识。

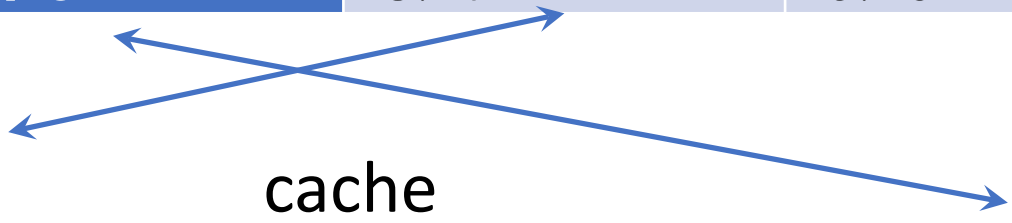
系统模块



Cache映射

按照参考书的指导，4块cache，直接映射，主存地址：

标识位	索引位	块内位移
31：9	8：7	6：0



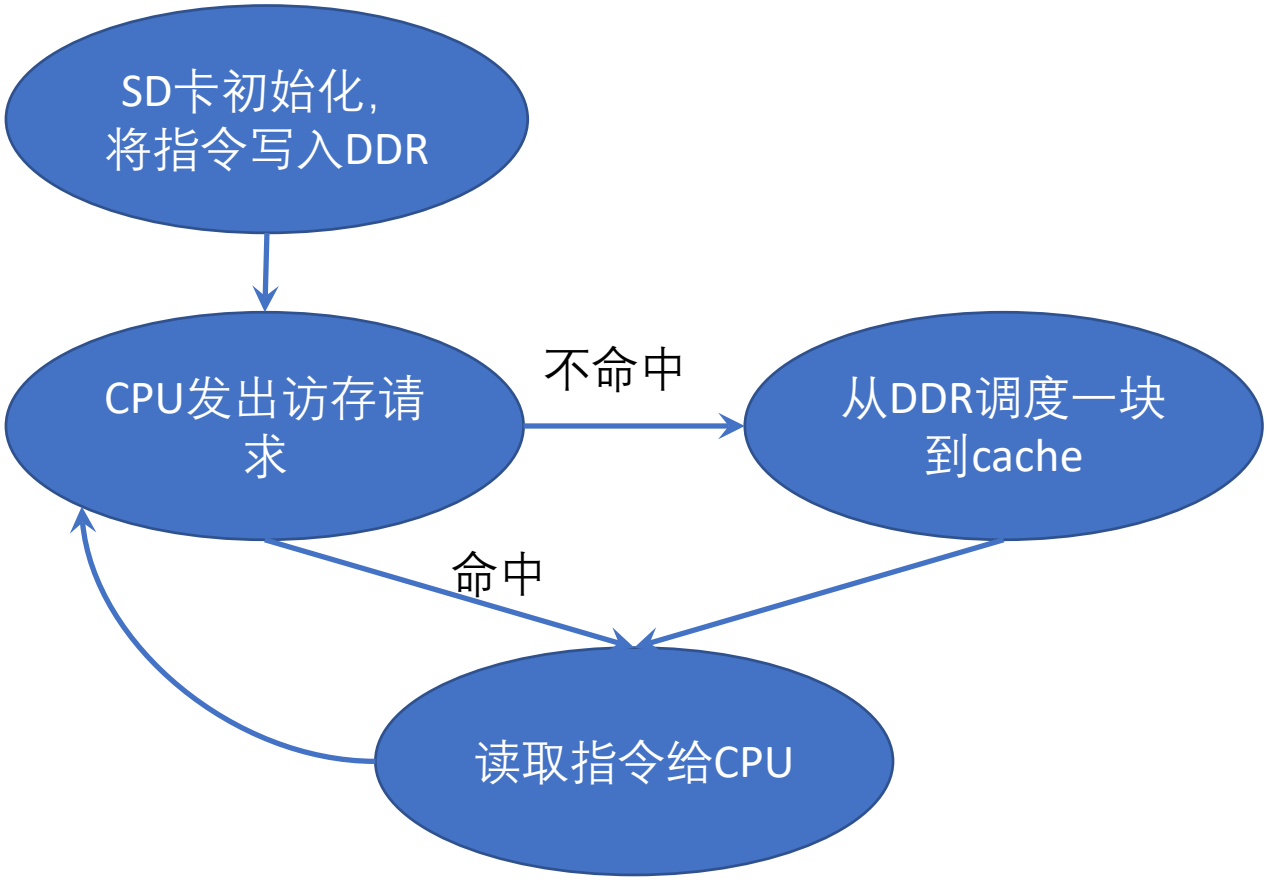
主存

8:7
0
1
2
3

cache

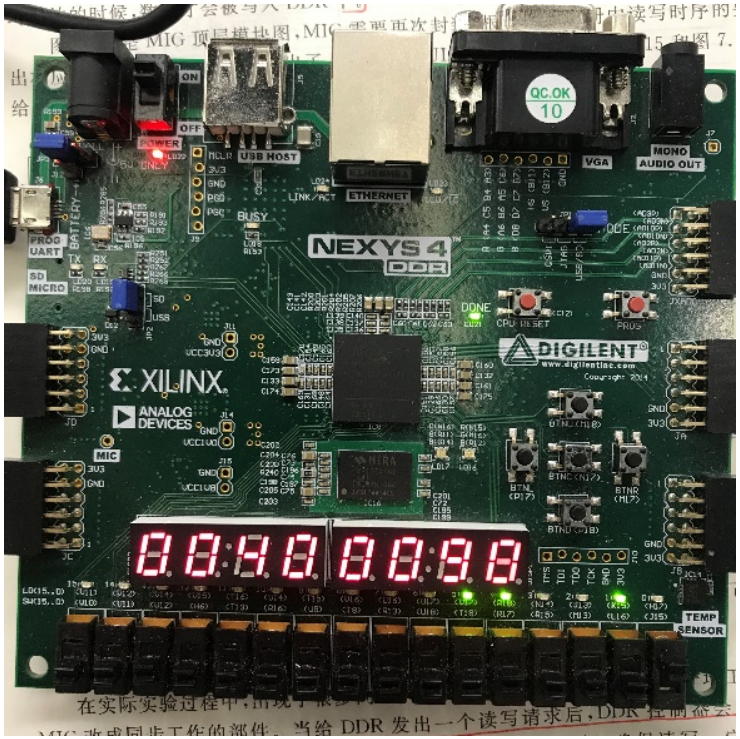
index	23	22:0
0	是否有数据	标识位
1		
2		
3		

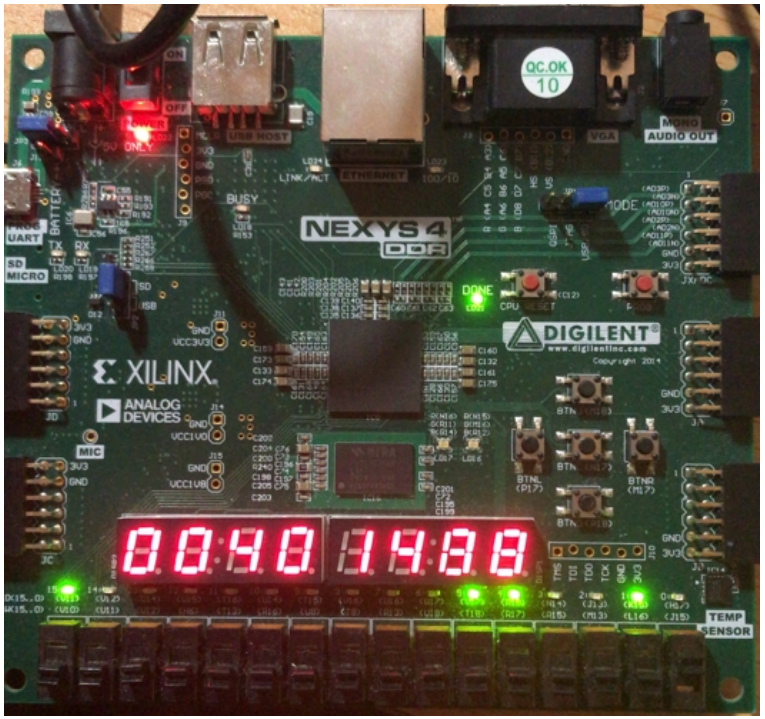
状态机


SD卡初始化，
将指令写入DDR
CPU发出访存请
求
不命中
从DDR调度一块
到cache
命中
读取指令给CPU

结果展示

- 更换卡前后，七段数码管显示pc值，led辅助debug





- 0x1488 > 0x200，说明读到了多个扇区，验证完毕!

主要参考资料

- **WinHex软件使用教程:**
- https://blog.csdn.net/weixin_39282491/article/details/80881468
- **SD卡原理相关：**
- https://blog.csdn.net/ming1006/article/details/7281597
- https://wenku.baidu.com/view/ce5042052af90242a895e582.html
- https://blog.csdn.net/zengaliang/article/details/76944664
- **交叉编译：**
- 简单的ld链接脚本学习https://www.jianshu.com/p/42823b3b7c8e
- Mips GNU工具链简介https://blog.csdn.net/alex_xhl/article/details/8137844
- hexdump命令https://man.linuxde.net/hexdump
- 【write a toy cpu】环境搭建 https://aojueliuyun.github.io/2018/04/30/2018.4.30--write-a-toy-cpu--environment/
- **DDR2：**
- Vivado中MIG核中DDR的读写控制 https://blog.csdn.net/wordwarwordwar/article/details/79539049

感谢聆听！