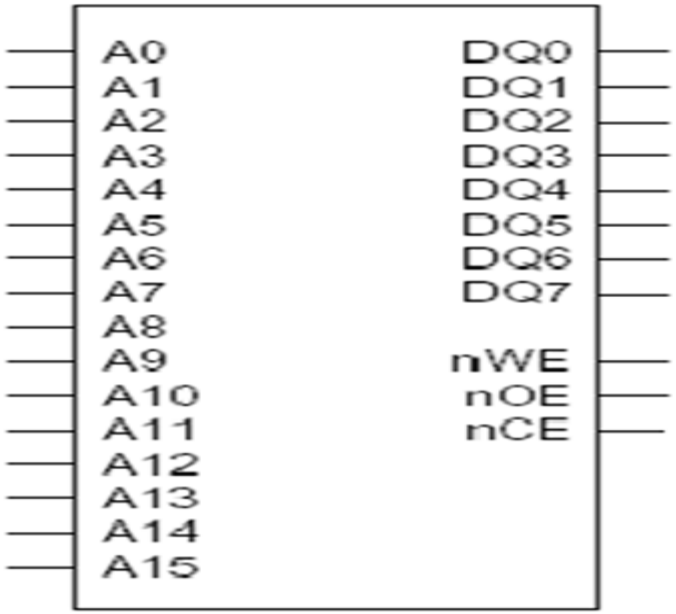
ARM与1x8bit RAM接口电路示例

已知：使用1个8 bit的RAM存储芯片搭建8位存储接口电路。

地址信号线(64K):A0---A15

数据信号线: 8位 DQ0---DQ7　　一次只能传送一个字节数据

读写控制信号线:　nWE(写信号) nOE(读信号)　nCE(片选信号)



ARM与1x8bit RAM接口电路示例

1.S3C2440 三总线接口

2.8bits RAM芯片引脚图

3.存储器接口电路以及工作原理

4.配置总线宽度与引脚时序

5.可用存储器指令

6.寻址范围

7.读写指令执行时，引脚时序关系

ARM与1x8bit RAM接口电路示例

1.S3C2440 三总线接口

(1)地址总线

A[31:30]=0b000　0x00000000--0x3FFFFFFF　1G　外部存储存储控件

A[29:27]=0b000　0x00000000—0x07FFFFFF　128M　BANK0　GCS0

=0b001　0x08000000—0x0FFFFFFF　128M　BANK1　GCS1

=0b010　0x10000000—0x17FFFFFF　128M　BANK2　GCS2

=0b011　0x18000000—0x1FFFFFFF　128M　BANK3　GCS3

=0b100　0x20000000—0x27FFFFFF　128M　BANK4　GCS4

=0b101　0x28000000—0x2FFFFFFF　128M　BANK5　GCS5

=0b110　0x30000000—0x37FFFFFF　128M　BANK6　GCS6

=0b111　0x38000000—0x3FFFFFFF　128M　BANK7　GCS7

A[26:0]用于与存储芯片地址引脚连接

ARM与1x8bit RAM接口电路示例

(2) 数据总线

D[31:0]用于与存储芯片数据引脚连接

(3)控制总线

nGCS[7:0]用于连接存储器片选信号

nWE:用于连接存储器写信号

nOE: 用于连接存储器读信号

ARM与1x8bit RAM接口电路示例

2.8bits RAM存储芯片引脚图

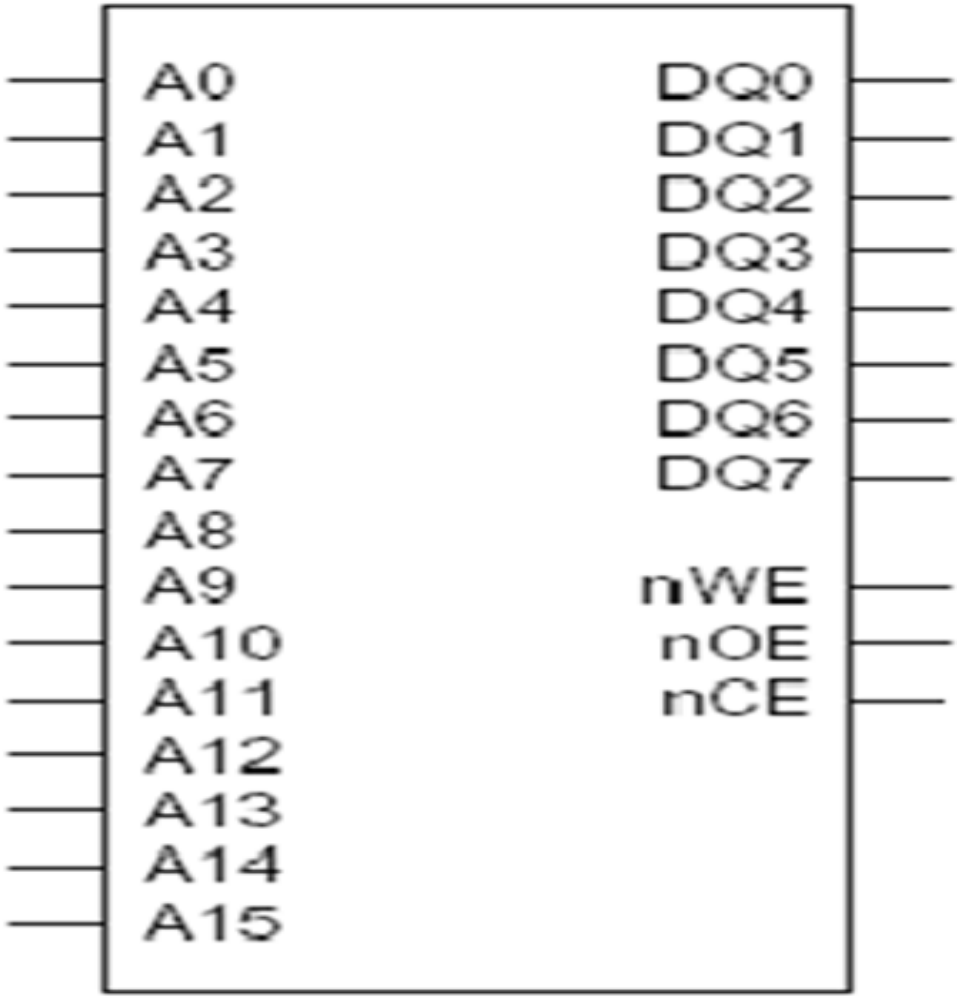
A[15:0]:16根地址线，用于为存储单元编址。
每个单元存储一个字节。
与处理器地址信号连接。

DQ[7:0]:通过此引脚读写锁定单元；
与处理器数据信号连接。

nCE:片选信号，低电平有效。此信号有效时，芯片处于工作状态；
与处理器片选信号连接。

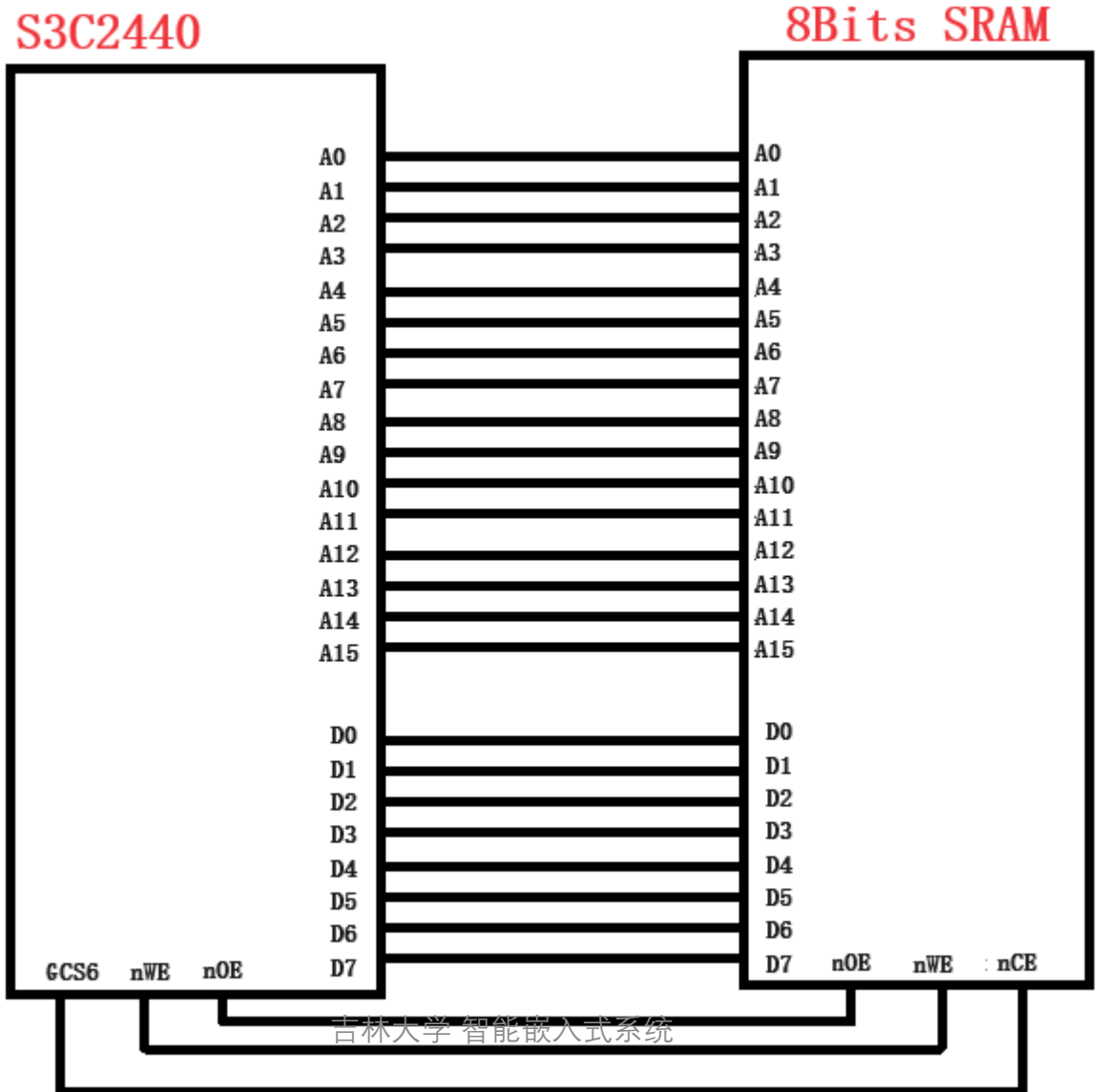
nWE:写信号，此信号有效时，处理器向存储器写数据；
与处理器写信号连接。

nOE:读使能信号，此信号有效时，处理器从存储器读数据；
与处理器读信号连接。



ARM与1x8bit RAM接口电路示例

3.存储器接口电路以及工作原理



ARM与1x8bit RAM接口电路示例

4.配置总线宽度与引脚时序

	BWSCON	位	描述	初始值
Bank7	ST7	[31]	SRAM对7号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank7	WS7	[30]	7号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank7	DW7	[29:28]	7号存储块的数据总线宽度控制位。 00 = 8bit; 01= 16bit; 10 = 32bit; 11 =保留。	0
Bank6	ST6	[27]	SRAM对6号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank6	WS6	[26]	6号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank6	DW6	[25:24]	决定对于6号存储块的数据总线宽度。 00 = 8bit; 01 = 16bit; 10 = 32bit; 11 =保留。	0
	ST5	[23]	SRAM对5号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0

(1).BWSCON(0x48000000):

设定Bank6的数据宽度及WAIT使能。

位[25:24]=0b00　8位数据总线[7:0]

位[26]=0b1　WAIT使能

位[27]=0b0　不使用UB/LB

ARM与1x8bit RAM接口电路示例

(2)BANKCON6配置

[16:15]=0b00 SRAM

依据SRAM参数配置

- 确定nGCSn信号有效前的地址建立时间
- 确定nOE信号有效前的片选建立时间
- 确定访问周期
- 确定nOE信号失效后片选信号保持的时间
- 确定nGCSn信号失效后有效地址保持的时间

BANKCON6	位	描述	初始
MT	[16:15]	确定6号存储块和7号存储块的存储器类型 00 = ROM or SRAM 01 = FP DRAM 10 =EDO DRAM 11 = SDRAM	11
当存储器类型为SRAM或ROM时(MT = 00)，需用下面15位			
Tacs	[14:13]	确定nGCSn信号有效前的地址建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcos	[12:11]	确定nOE信号有效前的片选建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacc	[10:8]	确定访问周期，注意，当nWAIT信号有效时，访问周期>=4时钟周期 000 = 1时钟 001 = 2时钟 010 = 3时钟 011 = 4时钟 100 = 6时钟 101 = 8时钟 110 = 10时钟 111 = 14时钟	111
Tcoh	[7:6]	确定nOE信号失效后片选信号保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcah	[5:4]	确定nGCSn信号失效后有效地址保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacp	[3:2]	确定页模式访问周期 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	00
PMC	[1:0]	确定页模式 00 = 常规（1data） 01 = 4 data 10 = 8 data 11 = 16 data	00
当存储器类型为SDRAM(MT = 11)时，需用下面4位			
Trcd	[3:2]	确定RAS对CAS的延时 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	10
SCAN	[1:0]	确定列地址位数 00 = 8位 01 = 9位 10 = 10位	00

ARM与1x8bit RAM接口电路示例

5.可用存储器指令

由于数据传送使用数据总线【7：0】，索引可用存储指令为,

LDRB

STRB

SWPB

ARM与1x8bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

(1)A[31:30]=0b00

(2)A[29:27]=0b110　BANK6　nGCS6有效

(3)A[26:16]=0bXXX XXXX XXXX　　x取值0或1

(4)A[15:0]连接SRAM的A[15:0]

- 存储芯片第一个字节单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxxx 0000 0000 0000 0000

- 存储芯片最后一个字节单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxxx 1111 1111 1111 1111

SRAM芯片寻址范围:

0b 0011 0xxx xxxx xxxx 0000 0000 0000 0000 到

0b 0011 0xxx xxxx xxxx 1111 1111 1111 1111

ARM与1x8bit RAM接口电路示例

7.读写指令执行时，引脚时序关系(以读指令LDRB为例)

(1)流水线译码部件对指令译码，解析出地址数据，依据此数据控制地址引脚A[31:0]产生相应的高低电平信号,

首先是A[26：0]引脚线产生电信号，用于锁定存储芯片的字节单元;

其次是A[29:27]=0b110, A[26：0]信号稳定后，nGCS6产生电信号，控制SRAM片选信号有效，从而SRAM工作;

然后是nOE有效，将锁定的字节单元数据放入数据总线D[7:0]上;

(2)流水线执行部件，从数据总线D[7:0]上将数据读入寄存器.

(3)nGCS6失效、地址信号失效，一次读数据过程结束。

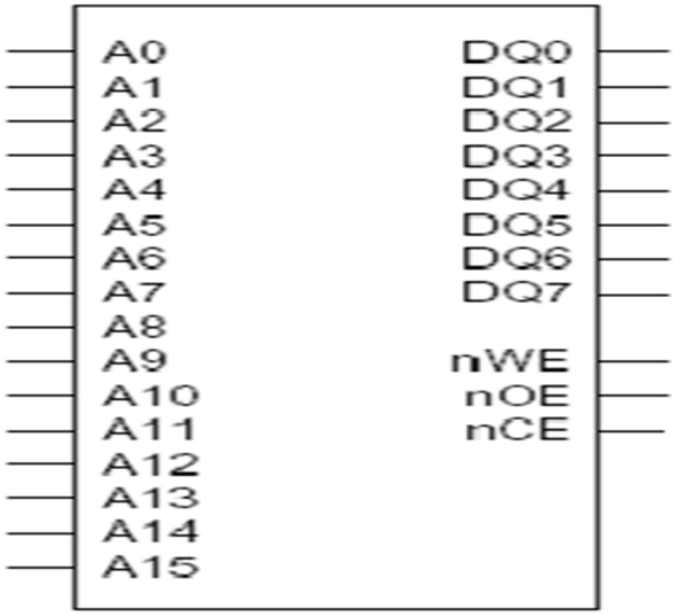
ARM与2x8bit RAM接口电路示例

已知：使用2个8 bit的RAM存储芯片搭建16位存储接口电路。

地址信号线(64K):A0---A15

数据信号线: 8位 DQ0---DQ7　　一次只能传送一个字节数据

读写控制信号线:　nWE(写信号) nOE(读信号)　nCE(片选信号)



ARM与2x8bit RAM接口电路示例

1.S3C2440 三总线接口

2.16bits RAM芯片引脚图

3.存储器接口电路以及工作原理

4.配置总线宽度与引脚时序

5.可用存储器指令

6.寻址范围

7.读写指令执行时，引脚时序关系

ARM与2x8bit RAM接口电路示例

1.S3C2440 三总线接口

(1)地址总线

A[31:30]=0b000　0x00000000--0x3FFFFFFF　1G　外部存储存储控件

A[29:27]=0b000　0x00000000—0x07FFFFFF　128M　BANK0　GCS0

=0b001　0x08000000—0x0FFFFFFF　128M　BANK1　GCS1

=0b010　0x10000000—0x17FFFFFF　128M　BANK2　GCS2

=0b011　0x18000000—0x1FFFFFFF　128M　BANK3　GCS3

=0b100　0x20000000—0x27FFFFFF　128M　BANK4　GCS4

=0b101　0x28000000—0x2FFFFFFF　128M　BANK5　GCS5

=0b110　0x30000000—0x37FFFFFF　128M　BANK6　GCS6

=0b111　0x38000000—0x3FFFFFFF　128M　BANK7　GCS7

A[26:0]用于与存储芯片地址引脚连接

ARM与2x8bit RAM接口电路示例

(2) 数据总线

D[31:0]用于与存储芯片数据引脚连接

(3)控制总线

nGCS[7:0]用于连接存储器片选信号

nWBE0、nWBE1:用于连接存储器写信号

nOE：用于连接存储器读信号

ARM与2x8bit RAM接口电路示例

2.8bits RAM存储芯片引脚图

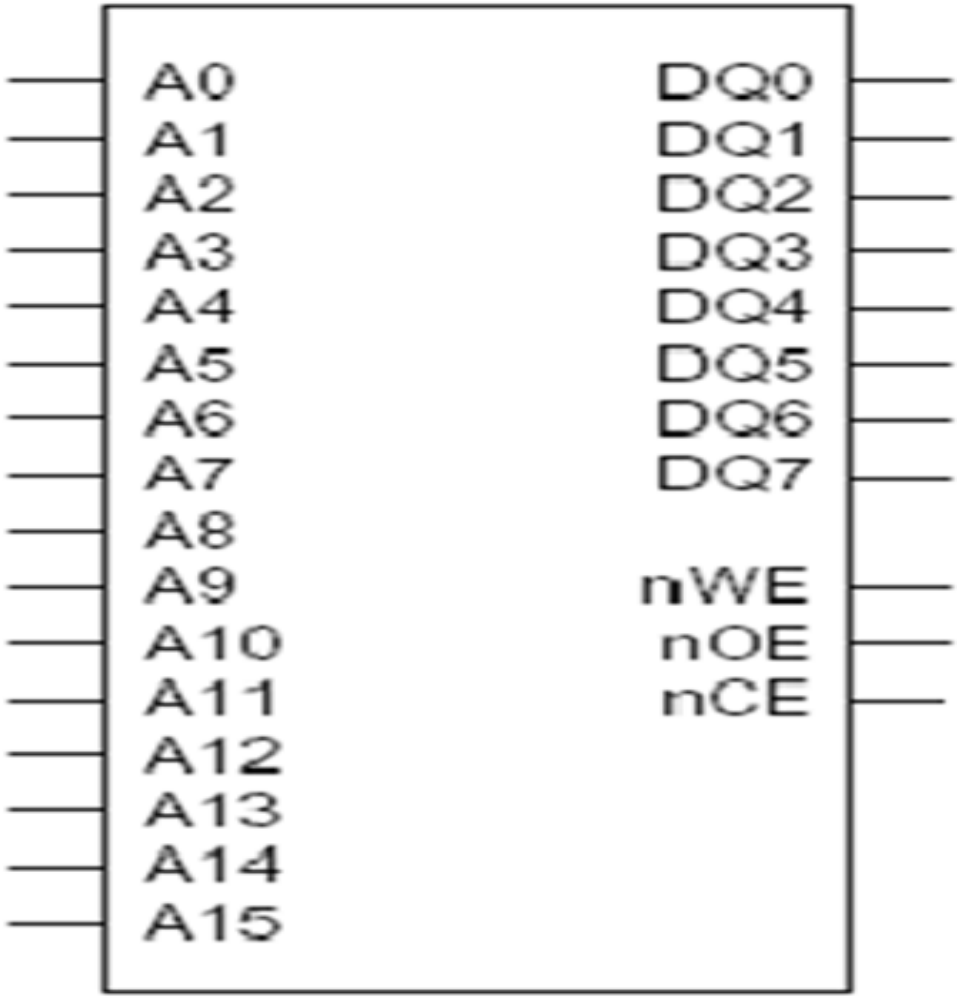
A[15:0]:16根地址线，用于为存储单元编址。
每个单元存储一个字节。
与处理器地址信号连接。

DQ[7:0]:通过此引脚读写锁定单元；
与处理器数据信号连接。

nCE:片选信号，低电平有效。此信号有效时，芯片处于工作状态；
与处理器片选信号连接。

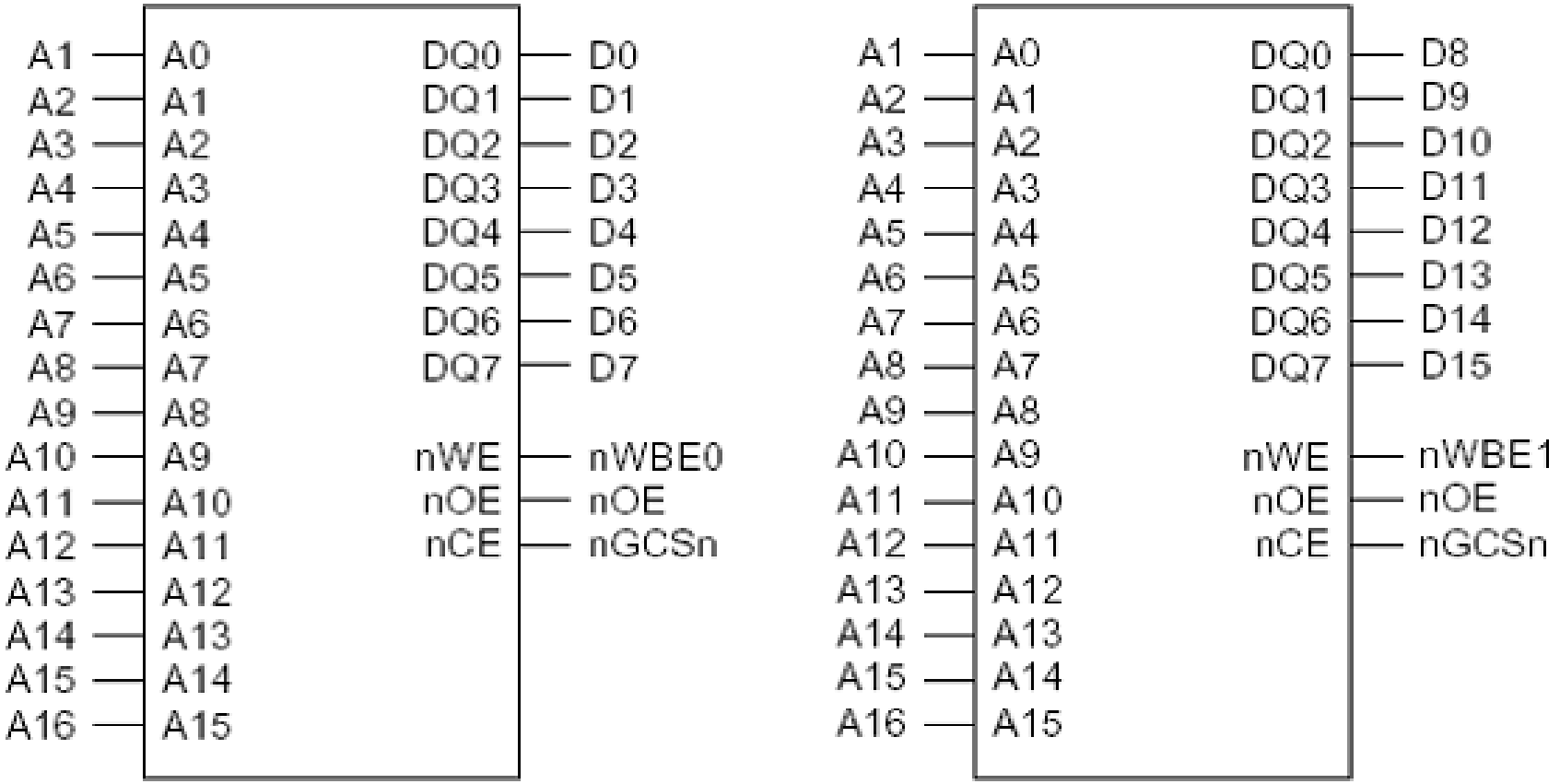
nWE:写信号，此信号有效时，处理器向存储器写数据；
与处理器写信号连接。

nOE:读使能信号，此信号有效时，处理器从存储器读数据；
与处理器读信号连接。



ARM与2x8bit RAM接口电路示例

3.存储器接口电路以及工作原理



ARM与2x8bit RAM接口电路示例

4.配置总线宽度与引脚时序

	BWSCON	位	描述	初始值
Bank7	ST7	[31]	SRAM对7号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank7	WS7	[30]	7号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank7	DW7	[29:28]	7号存储块的数据总线宽度控制位。 00 = 8bit; 01= 16bit; 10 = 32bit; 11 =保留。	0
Bank6	ST6	[27]	SRAM对6号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank6	WS6	[26]	6号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank6	DW6	[25:24]	决定对于6号存储块的数据总线宽度。 00 = 8bit; 01 = 16bit; 10 = 32bit; 11 =保留。	0
	ST5	[23]	SRAM对5号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0

(1).BWSCON(0x48000000):

设定Bank6的数据宽度及WAIT使能。

位[25:24]=0b01　16位数据总线[15:0]

位[26]=0b1　WAIT使能

位[27]=0b0　不使用UB/LB

ARM与2x8bit RAM接口电路示例

(2)BANKCON6配置

[16:15]=0b00 SRAM

依据SRAM参数配置

- 确定nGCSn信号有效前的地址建立时间
- 确定nOE信号有效前的片选建立时间
- 确定访问周期
- 确定nOE信号失效后片选信号保持的时间
- 确定nGCSn信号失效后有效地址保持的时间

BANKCON6	位	描述	初始
MT	[16:15]	确定6号存储块和7号存储块的存储器类型 00 = ROM or SRAM 01 = FP DRAM 10 =EDO DRAM 11 = SDRAM	11
当存储器类型为SRAM或ROM时(MT = 00)，需用下面15位			
Tacs	[14:13]	确定nGCSn信号有效前的地址建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcos	[12:11]	确定nOE信号有效前的片选建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacc	[10:8]	确定访问周期，注意，当nWAIT信号有效时，访问周期>=4时钟周期 000 = 1时钟 001 = 2时钟 010 = 3时钟 011 = 4时钟 100 = 6时钟 101 = 8时钟 110 = 10时钟 111 = 14时钟	111
Tcoh	[7:6]	确定nOE信号失效后片选信号保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcah	[5:4]	确定nGCSn信号失效后有效地址保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacp	[3:2]	确定页模式访问周期 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	00
PMC	[1:0]	确定页模式 00 = 常规（1data） 01 = 4 data 10 = 8 data 11 = 16 data	00
当存储器类型为SDRAM(MT = 11)时，需用下面4位			
Trcd	[3:2]	确定RAS对CAS的延时 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	10
SCAN	[1:0]	确定列地址位数 00 = 8位 01 = 9位 10 = 10位	00

ARM与2x8bit RAM接口电路示例

5.可用存储器指令

由于数据传送使用数据总线【15：0】，索引可用存储指令为,

LDRH

STRH

LDRB

STRB

SWPB

ARM与2x8bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

(1)A[31:30]=0b00

(2)A[29:27]=0b110　BANK6　nGCS6有效

(3)A[26:17]=0bXXX XXXX XXX　　x取值0或1

(4)A[16:1]连接SRAM的A[15:0]

(5)A[0]空闲

- 存储芯片第一个字节单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxx0 0000 0000 0000 0000

- 存储芯片最后一个字节单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxx1 1111 1111 1111 1111

SRAM芯片寻址范围(字节空间):

0b 0011 0xxx xxxx xxx0 0000 0000 0000 0000 到

0b 0011 0xxx xxxx xxx1 1111 1111 1111 1111

ARM与2x8bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

- 存储芯片第一个半字单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxx0 0000 0000 0000 000x

- 存储芯片最后一个半字单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxx1 1111 1111 1111 111x

SRAM芯片寻址范围(半字空间):

0b 0011 0xxx xxxx xxx0 0000 0000 0000 000x 到

0b 0011 0xxx xxxx xxx1 1111 1111 1111 111x

ARM与2x8bit RAM接口电路示例

7.读写指令执行时，引脚时序关系(以读指令LDRB为例)

(1)流水线译码部件对指令译码，解析出地址数据，依据此数据控制地址引脚A[31:0]产生相应的高低电平信号,

首先是A[26：0]引脚线产生电信号，用于锁定存储芯片的字节单元;

其次是A[29:27]=0b110, A[26：0]信号稳定后，nGCS6产生电信号，

A0=0,控制SRAM1片选信号有效，从而SRAM1工作;

然后是nOE有效，将锁定的字节单元数据放入数据总线D[7:0]上;

流水线执行部件，从数据总线D[7:0]上将数据读入寄存器[7:0].

A0=1,控制SRAM2片选信号有效，从而SRAM2工作;

然后是nOE有效，将锁定的字节单元数据放入数据总线D[15:8]上;

流水线执行部件，从数据总线D[15:8]上将数据读入寄存器[7:0].

(3)nGCS6失效、地址信号失效，一次读数据过程结束。

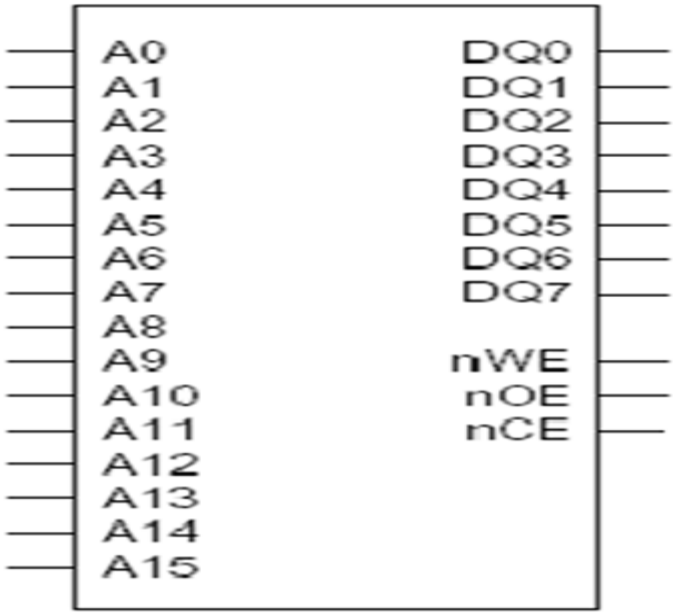
ARM与4x8bit RAM接口电路示例

已知：使用4个8 bit的RAM存储芯片搭建32位存储接口电路。

地址信号线(64K):A0---A15

数据信号线: 8位 DQ0---DQ7　　一次只能传送一个字节数据

读写控制信号线:　nWE(写信号) nOE(读信号)　nCE(片选信号)



ARM与4x8bit RAM接口电路示例

1.S3C2440 三总线接口

2.32bits RAM芯片引脚图

3.存储器接口电路以及工作原理

4.配置总线宽度与引脚时序

5.可用存储器指令

6.寻址范围

7.读写指令执行时，引脚时序关系

ARM与4x8bit RAM接口电路示例

1.S3C2440 三总线接口

(1)地址总线

A[31:30]=0b000　0x00000000--0x3FFFFFFF　1G　外部存储存储控件

A[29:27]=0b000　0x00000000—0x07FFFFFF　128M　BANK0　GCS0

=0b001　0x08000000—0x0FFFFFFF　128M　BANK1　GCS1

=0b010　0x10000000—0x17FFFFFF　128M　BANK2　GCS2

=0b011　0x18000000—0x1FFFFFFF　128M　BANK3　GCS3

=0b100　0x20000000—0x27FFFFFF　128M　BANK4　GCS4

=0b101　0x28000000—0x2FFFFFFF　128M　BANK5　GCS5

=0b110　0x30000000—0x37FFFFFF　128M　BANK6　GCS6

=0b111　0x38000000—0x3FFFFFFF　128M　BANK7　GCS7

A[26:0]用于与存储芯片地址引脚连接

ARM与4x8bit RAM接口电路示例

(2) 数据总线

D[31:0]用于与存储芯片数据引脚连接

(3)控制总线

nGCS[7:0]用于连接存储器片选信号

nWBE[3:0]:用于连接存储器写信号

nOE：用于连接存储器读信号

ARM与4x8bit RAM接口电路示例

2.8bits RAM存储芯片引脚图

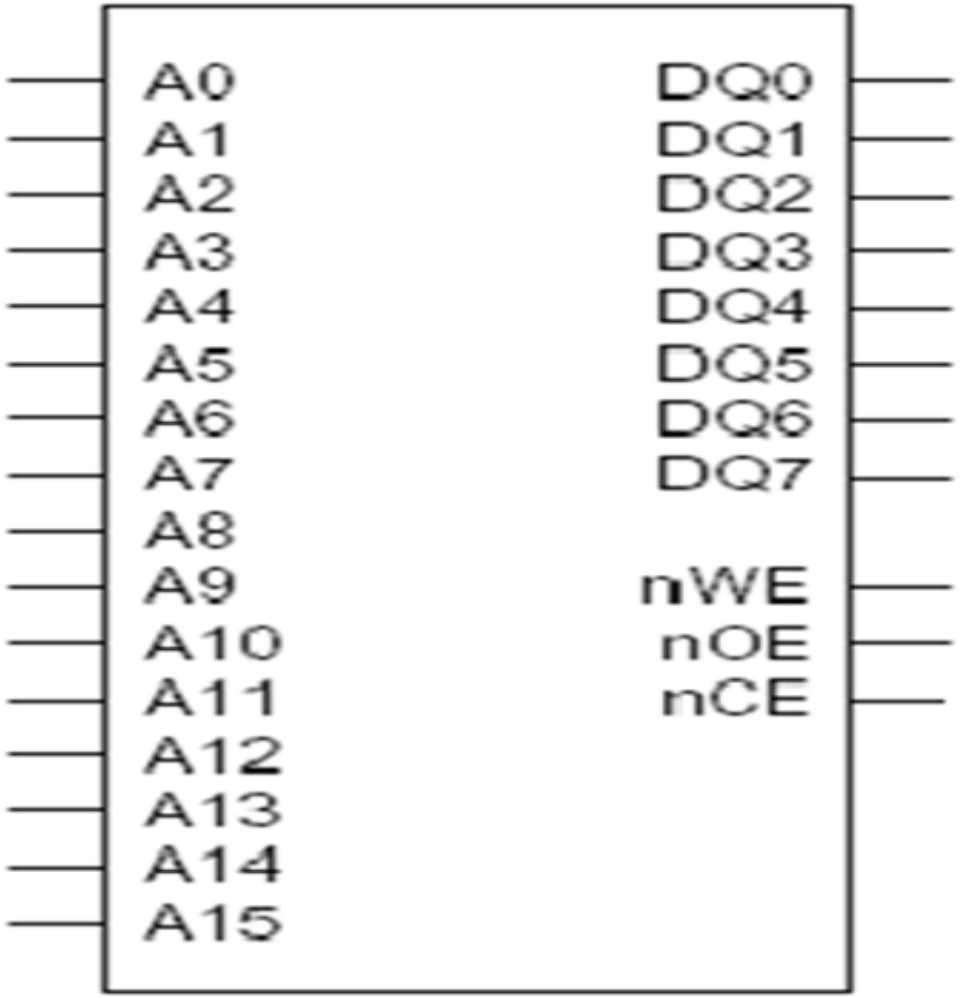
A[15:0]:16根地址线，用于为存储单元编址。
每个单元存储一个字节。
与处理器地址信号连接。

DQ[7:0]:通过此引脚读写锁定单元；
与处理器数据信号连接。

nCE:片选信号，低电平有效。此信号有效时，芯片处于工作状态；
与处理器片选信号连接。

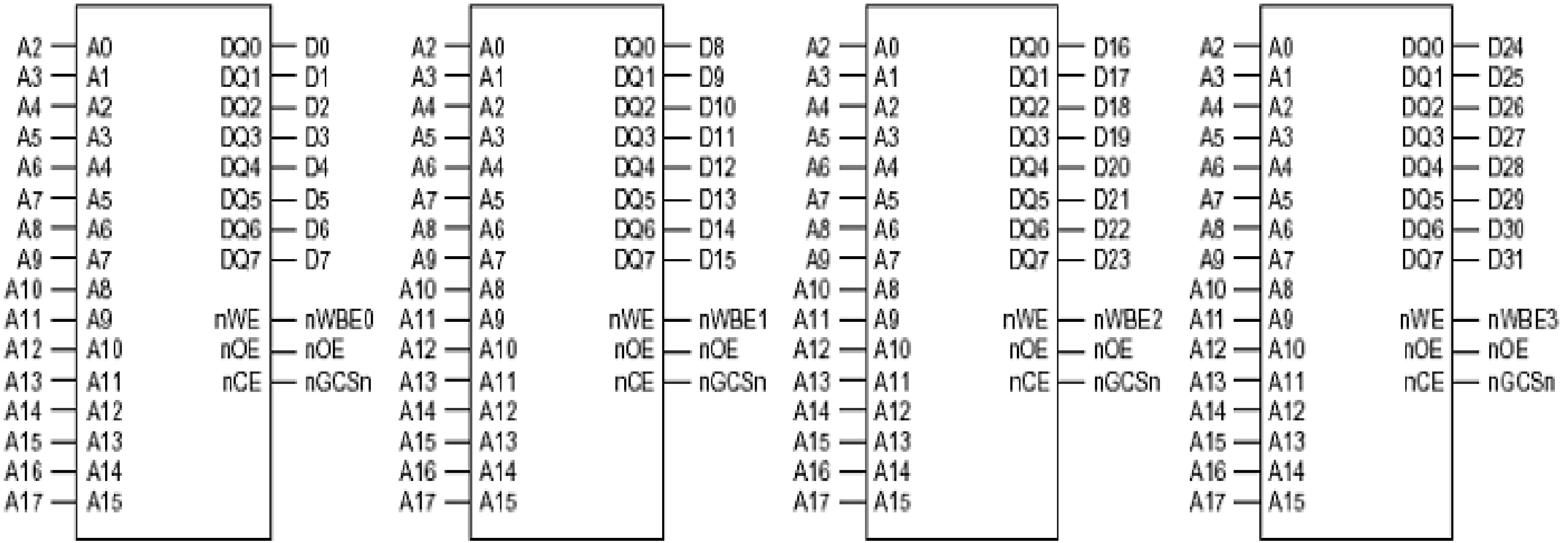
nWE:写信号，此信号有效时，处理器向存储器写数据；
与处理器写信号连接。

nOE:读使能信号，此信号有效时，处理器从存储器读数据；
与处理器读信号连接。



ARM与4x8bit RAM接口电路示例

3.存储器接口电路以及工作原理



ARM与4x8bit RAM接口电路示例

4.配置总线宽度与引脚时序

	BWSCON	位	描述	初始值
Bank7	ST7	[31]	SRAM对7号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank7	WS7	[30]	7号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank7	DW7	[29:28]	7号存储块的数据总线宽度控制位。 00 = 8bit; 01= 16bit; 10 = 32bit; 11 =保留。	0
Bank6	ST6	[27]	SRAM对6号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank6	WS6	[26]	6号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank6	DW6	[25:24]	决定对于6号存储块的数据总线宽度。 00 = 8bit; 01 = 16bit; 10 = 32bit; 11 =保留。	0
	ST5	[23]	SRAM对5号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0

(1).BWSCON(0x48000000):

设定Bank6的数据宽度及WAIT使能。

位[25:24]=0b10　32位数据总线[31:0]

位[26]=0b1　WAIT使能

位[27]=0b0　不使用UB/LB

ARM与4x8bit RAM接口电路示例

(2)BANKCON6配置

[16:15]=0b00 SRAM

依据SRAM参数配置

- 确定nGCSn信号有效前的地址建立时间
- 确定nOE信号有效前的片选建立时间
- 确定访问周期
- 确定nOE信号失效后片选信号保持的时间
- 确定nGCSn信号失效后有效地址保持的时间

BANKCON6	位	描述	初始
MT	[16:15]	确定6号存储块和7号存储块的存储器类型 00 = ROM or SRAM 01 = FP DRAM 10 =EDO DRAM 11 = SDRAM	11
当存储器类型为SRAM或ROM时(MT = 00)，需用下面15位			
Tacs	[14:13]	确定nGCSn信号有效前的地址建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcos	[12:11]	确定nOE信号有效前的片选建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacc	[10:8]	确定访问周期，注意，当nWAIT信号有效时，访问周期>=4时钟周期 000 = 1时钟 001 = 2时钟 010 = 3时钟 011 = 4时钟 100 = 6时钟 101 = 8时钟 110 = 10时钟 111 = 14时钟	111
Tcoh	[7:6]	确定nOE信号失效后片选信号保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcah	[5:4]	确定nGCSn信号失效后有效地址保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacp	[3:2]	确定页模式访问周期 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	00
PMC	[1:0]	确定页模式 00 = 常规（1data） 01 = 4 data 10 = 8 data 11 = 16 data	00
当存储器类型为SDRAM(MT = 11)时，需用下面4位			
Trcd	[3:2]	确定RAS对CAS的延时 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	10
SCAN	[1:0]	确定列地址位数 00 = 8位 01 = 9位 10 = 10位	00

ARM与4x8bit RAM接口电路示例

5.可用存储器指令

由于数据传送使用数据总线【31：0】， 索引可用存储指令为,

LDM

STM

LDR

STR

LDRH

STRH

LDRB

STRB

SWP

SWPB

ARM与4x8bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

(1)A[31:30]=0b00

(2)A[29:27]=0b110　BANK6　nGCS6有效

(3)A[26:17]=0bXXX XXXX XXX　　x取值0或1

(4)A[17:2]连接SRAM1\SRAM2\SRAM3\SRAM4的A[15:0]

(5)A[1:0]空闲

- 存储芯片第一个字节单元地址:

  A[31:0]=0b 0011 0xxx xxxx xx00 0000 0000 0000 0000

- 存储芯片最后一个字节单元地址:

  A[31:0]=0b 0011 0xxx xxxx xx11 1111 1111 1111 1111

SRAM芯片寻址范围(字节空间):

0b 0011 0xxx xxxx xx00 0000 0000 0000 0000 到

0b 0011 0xxx xxxx xx11 1111 1111 1111 1111

ARM与4x8bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

- 存储芯片第一个半字单元地址:

A[31:0]=0b 0011 0xxx xxxx xx00 0000 0000 0000 000x

- 存储芯片最后一个半字单元地址:

A[31:0]=0b 0011 0xxx xxxx xx11 1111 1111 1111 111x

SRAM芯片寻址范围(半字空间):

0b 0011 0xxx xxxx xx00 0000 0000 0000 000x 到

0b 0011 0xxx xxxx xx11 1111 1111 1111 111x

- 存储芯片第一个字单元地址:

A[31:0]=0b 0011 0xxx xxxx xx00 0000 0000 0000 00xx

- 存储芯片最后一个字单元地址:

A[31:0]=0b 0011 0xxx xxxx xx11 1111 1111 1111 11xx

SRAM芯片寻址范围(半字空间):

0b 0011 0xxx xxxx xx00 0000 0000 0000 00xx 到

0b 0011 0xxx xxxx xx11 1111 1111 1111 11xx

ARM与4x8bit RAM接口电路示例

7.读写指令执行时，引脚时序关系(以读指令LDRB为例)

(1)流水线译码部件对指令译码，解析出地址数据，依据此数据控制地址引脚A[31:0]产生相应的高低电平信号，

首先是A[26：0]引脚线产生电信号，用于锁定存储芯片的字节单元；

其次是A[29:27]=0b110，A[26：0]信号稳定后，nGCS6产生电信号，

A[1:0]=00 控制SRAM1片选信号有效，从而SRAM1工作；

然后是nOE有效，将锁定的字节单元数据放入数据总线D[7:0]上；

流水线执行部件，从数据总线D[7:0]上将数据读入寄存器[7:0].

A[1:0]=01 控制SRAM2片选信号有效，从而SRAM2工作；

然后是nOE有效，将锁定的字节单元数据放入数据总线D[15:8]上；

流水线执行部件，从数据总线D[15:8]上将数据读入寄存器[7:0].

A[1:0]=10 控制SRAM3片选信号有效，从而SRAM3工作；

然后是nOE有效，将锁定的字节单元数据放入数据总线D[23:16]上；

流水线执行部件，从数据总线D[23:16]上将数据读入寄存器[7:0].

A[1:0]=11 控制SRAM4片选信号有效，从而SRAM4工作；

然后是nOE有效，将锁定的字节单元数据放入数据总线D[31:24]上；

流水线执行部件，从数据总线D[31:24]上将数据读入寄存器[7:0].

(2)nGCS6失效、地址信号失效，一次读数据过程结束。

ARM与1x16bit RAM接口电路示例

已知：使用1个16 bit的RAM存储芯片搭建16位存储接口电路。

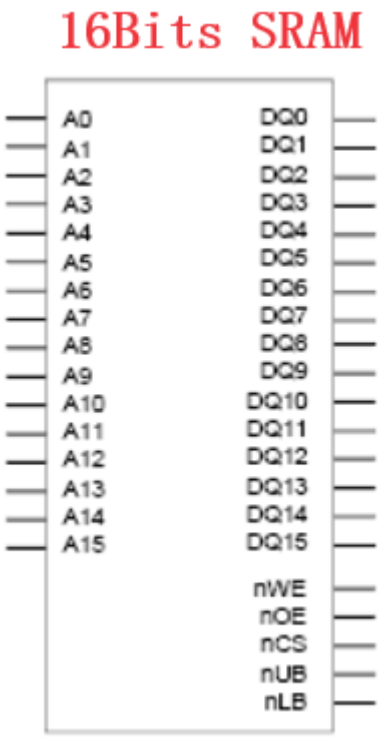
地址信号线(64K 半字单元):A0---A15

数据信号线: 8位 DQ0---DQ15　　一次只能传送一个半字数据

读写控制信号线:　nWE(写信号) nOE(读信号)　nCE(片选信号)

nUB(半字单元的高8位读写)

nLB(半字单元的低8位读写)



ARM与1x16bit SRAM接口电路示例

1.S3C2440 三总线接口

2.16bits RAM芯片引脚图

3.存储器接口电路以及工作原理

4.配置总线宽度与引脚时序

5.可用存储器指令

6.寻址范围

7.读写指令执行时，引脚时序关系

ARM与1x16bit RAM接口电路示例

1.S3C2440 三总线接口

(1)地址总线

A[31:30]=0b00　0x00000000--0x3FFFFFFF　1G　外部存储存储控件

A[29:27]=0b000　0x00000000—0x07FFFFFF　128M　BANK0　GCS0

=0b001　0x08000000—0x0FFFFFFF　128M　BANK1　GCS1

=0b010　0x10000000—0x17FFFFFF　128M　BANK2　GCS2

=0b011　0x18000000—0x1FFFFFFF　128M　BANK3　GCS3

=0b100　0x20000000—0x27FFFFFF　128M　BANK4　GCS4

=0b101　0x28000000—0x2FFFFFFF　128M　BANK5　GCS5

=0b110　0x30000000—0x37FFFFFF　128M　BANK6　GCS6

=0b111　0x38000000—0x3FFFFFFF　128M　BANK7　GCS7

A[26:0]用于与存储芯片地址引脚连接

ARM与1x16bit RAM接口电路示例

(2) 数据总线

D[31:0]用于与存储芯片数据引脚连接

(3)控制总线

nGCS[7:0]用于连接存储器片选信号

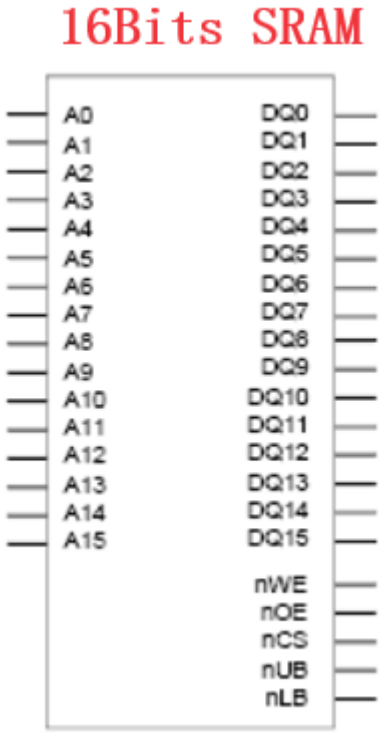
nWE:用于连接存储器写信号

nOE：用于连接存储器读信号

nBE[3:0]:用于连接存储区nUB/nLB，控制读取半字中的高/低8位数据

ARM与1x16bit RAM接口电路示例

2.16bits RAM存储芯片引脚图



A[15:0]:16根地址线，用于为存储单元编址。
每个单元存储一个半字（16位）。
与处理器地址信号连接。

DQ[15:0]:通过此引脚读写锁定单元；
与处理器数据信号连接。

nCS:片选信号，低电平有效。此信号有效时，
芯片处于工作状态；
与处理器片选信号连接。

nWE:写信号，此信号有效时，处理器向存储器
写数据；
与处理器写信号连接。

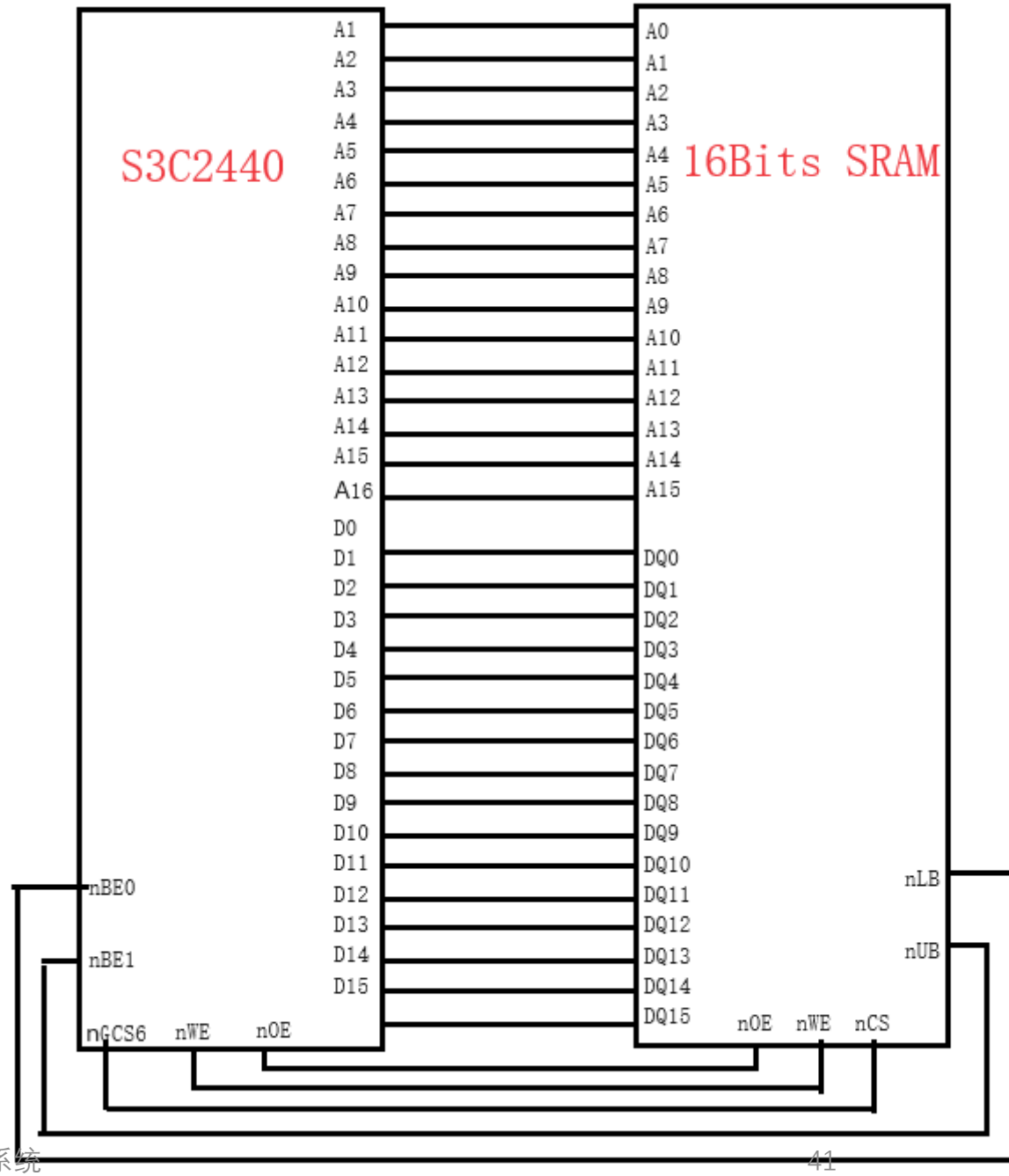
nOE:读使能信号，此信号有效时，处理器从存储器
读数据；
与处理器读信号连接。

nUB:半字单元的高8位读写信号，与nWE、nOE配合，读写取半字单元的高8位；

nLB:半字单元的低8位读写信号，与nWE、nOE配合，读写取半字单元的低8位。

ARM与1x16bit RAM接口电路示例

3.存储器接口电路以及工作原理



ARM与1x16bit RAM接口电路示例

4.配置总线宽度与引脚时序

	BWSCON	位	描述	初始值
Bank7	ST7	[31]	SRAM对7号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank7	WS7	[30]	7号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank7	DW7	[29:28]	7号存储块的数据总线宽度控制位。 00 = 8bit; 01= 16bit; 10 = 32bit; 11 =保留。	0
Bank6	ST6	[27]	SRAM对6号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank6	WS6	[26]	6号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank6	DW6	[25:24]	决定对于6号存储块的数据总线宽度。 00 = 8bit; 01 = 16bit; 10 = 32bit; 11 =保留。	0
	ST5	[23]	SRAM对5号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0

(1).BWSCON(0x48000000):

设定Bank6的数据宽度及WAIT使能。

位[25:24]=0b01 16位数据总线[15:0]

位[26]=0b1 WAIT使能

位[27]=0b1 使用UB/LB

ARM与1x16bit RAM接口电路示例

(2)BANKCON6配置

[16:15]=0b00 SRAM

依据SRAM参数配置

- 确定nGCSn信号有效前的地址建立时间
- 确定nOE信号有效前的片选建立时间
- 确定访问周期
- 确定nOE信号失效后片选信号保持的时间
- 确定nGCSn信号失效后有效地址保持的时间

BANKCON6	位	描述	初始
MT	[16:15]	确定6号存储块和7号存储块的存储器类型 00 = ROM or SRAM 01 = FP DRAM 10 =EDO DRAM 11 = SDRAM	11
当存储器类型为SRAM或ROM时(MT = 00)，需用下面15位			
Tacs	[14:13]	确定nGCSn信号有效前的地址建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcos	[12:11]	确定nOE信号有效前的片选建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacc	[10:8]	确定访问周期，注意，当nWAIT信号有效时，访问周期>=4时钟周期 000 = 1时钟 001 = 2时钟 010 = 3时钟 011 = 4时钟 100 = 6时钟 101 = 8时钟 110 = 10时钟 111 = 14时钟	111
Tcoh	[7:6]	确定nOE信号失效后片选信号保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcah	[5:4]	确定nGCSn信号失效后有效地址保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacp	[3:2]	确定页模式访问周期 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	00
PMC	[1:0]	确定页模式 00 = 常规（1data） 01 = 4 data 10 = 8 data 11 = 16 data	00
当存储器类型为SDRAM(MT = 11)时，需用下面4位			
Trcd	[3:2]	确定RAS对CAS的延时 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	10
SCAN	[1:0]	确定列地址位数 00 = 8位 01 = 9位 10 = 10位	00

ARM与1x16bit RAM接口电路示例

5.可用存储器指令

由于数据传送使用数据总线【15：0】，索引可用存储指令为,

LDRH、STRH 半字读写

LDRB、STRB　字节读写

SWPB　字节交换

ARM与1x16bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

(1)A[31:30]=0b00

(2)A[29:27]=0b110　BANK6　GCS6

(3)A[26:17]=0bXXX XXXX XXX　　x取值0或1

(4)A[16:1]连接SRAM的A[15:0]

(5)A0=0b0，nBE0有效，读写半字单元的低8位； A0=0b1，nBE1有效，读写半字单元的高8位.

- 存储芯片第一个半字单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxx0 0000 0000 0000 000x

- 存储芯片最后一个半字单元地址:

  A[31:0]=0b 0011 0xxx xxxx xxx0 1111 1111 1111 111x

SRAM芯片寻址范围(半字):

0b 0011 0xxx xxxx xxx0 0000 0000 0000 000x 到

0b 0011 0xxx xxxx xxx0 1111 1111 1111 111x

ARM与1x16bit RAM接口电路示例

7.读写指令执行时，引脚时序关系(以读指令LDRH为例)

(1)流水线译码部件对指令译码，解析出地址数据，依据此数据控制地址引脚A[31:0]产生相应的高低电平信号,

首先是A[26：0]引脚线产生电信号，用于锁定存储芯片的半字单元;

其次是A[29:27]=0b110, A[26：0]信号稳定后，nGCS6产生电信号，控制SRAM片选信号有效，从而SRAM工作;

然后是nOE有效，将锁定的半字单元数据放入数据总线D[15:0]上;

(2)流水线执行部件，从数据总线D[15:0]上将数据读入寄存器[15:0].

(3)nGCS6失效、地址信号失效哦啊，一次读数据过程结束。

ARM与2x16bit RAM接口电路示例

已知：使用2个16 bit的RAM存储芯片搭建32位存储接口电路。

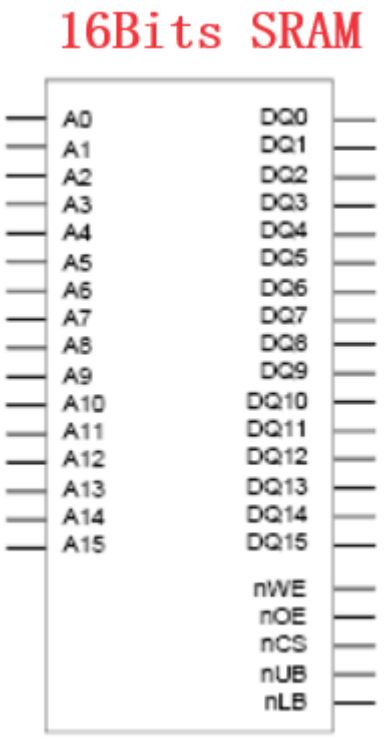
地址信号线(64K 半字单元):A0---A15

数据信号线: 8位 DQ0---DQ15　　一次只能传送一个半字数据

读写控制信号线:　nWE(写信号) nOE(读信号)　nCE(片选信号)

nUB(半字单元的高8位读写)

nLB(半字单元的低8位读写)



ARM与2x16bit SRAM接口电路示例

1.S3C2440 三总线接口

2.32bits RAM芯片引脚图

3.存储器接口电路以及工作原理

4.配置总线宽度与引脚时序

5.可用存储器指令

6.寻址范围

7.读写指令执行时，引脚时序关系

ARM与2x16bit RAM接口电路示例

1.S3C2440 三总线接口

(1)地址总线

A[31:30]=0b00　0x00000000--0x3FFFFFFF　1G　外部存储存储控件

A[29:27]=0b000　0x00000000—0x07FFFFFF　128M　BANK0　GCS0

=0b001　0x08000000—0x0FFFFFFF　128M　BANK1　GCS1

=0b010　0x10000000—0x17FFFFFF　128M　BANK2　GCS2

=0b011　0x18000000—0x1FFFFFFF　128M　BANK3　GCS3

=0b100　0x20000000—0x27FFFFFF　128M　BANK4　GCS4

=0b101　0x28000000—0x2FFFFFFF　128M　BANK5　GCS5

=0b110　0x30000000—0x37FFFFFF　128M　BANK6　GCS6

=0b111　0x38000000—0x3FFFFFFF　128M　BANK7　GCS7

A[26:0]用于与存储芯片地址引脚连接

ARM与2x16bit RAM接口电路示例

(2) 数据总线

D[31:0]用于与存储芯片数据引脚连接

(3)控制总线

nGCS[7:0]用于连接存储器片选信号

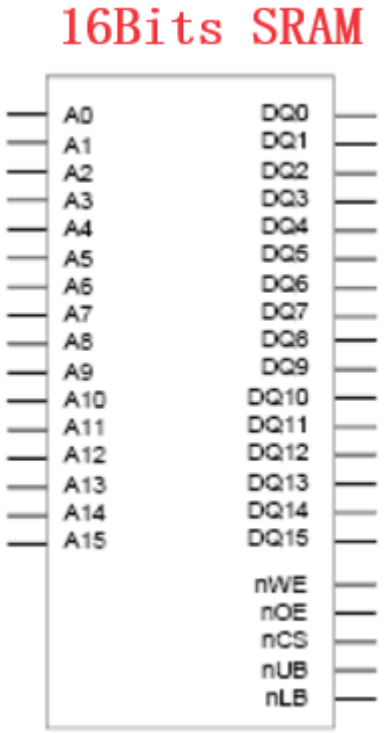
nWE:用于连接存储器写信号

nOE：用于连接存储器读信号

nBE[3:0]:用于连接存储区nUB/nLB，控制读取半字中的高/低8位数据

ARM与2x16bit RAM接口电路示例

2.16bits RAM存储芯片引脚图



A[15:0]:16根地址线，用于为存储单元编址。
每个单元存储一个半字（16位）。
与处理器地址信号连接。

DQ[15:0]:通过此引脚读写锁定单元；
与处理器数据信号连接。

nCS:片选信号，低电平有效。此信号有效时，
芯片处于工作状态；
与处理器片选信号连接。

nWE:写信号，此信号有效时，处理器向存储器
写数据；
与处理器写信号连接。

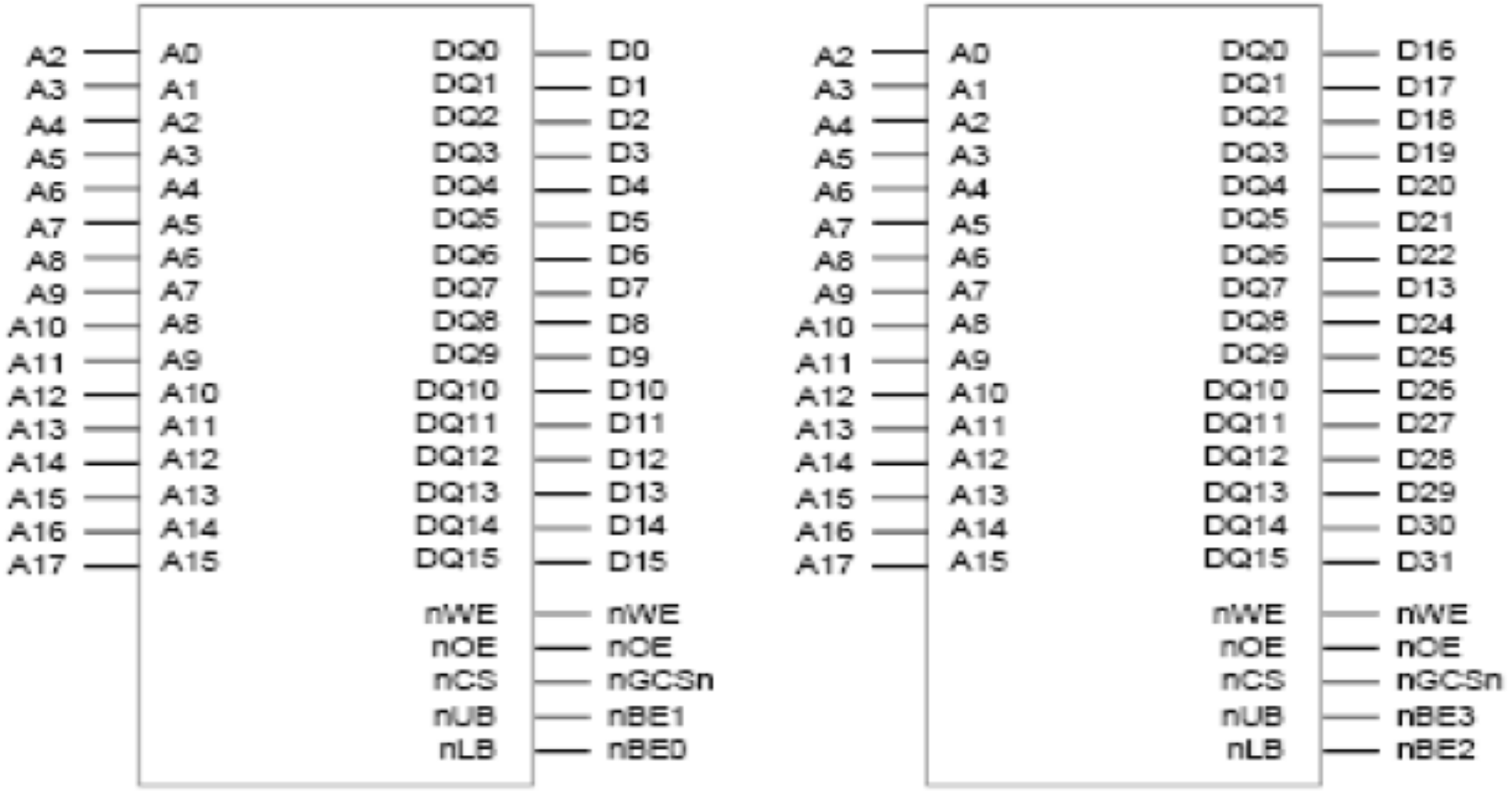
nOE:读使能信号，此信号有效时，处理器从存储器
读数据；
与处理器读信号连接。

nUB:半字单元的高8位读写信号，与nWE、nOE配合，读写取半字单元的高8位；

nLB:半字单元的低8位读写信号，与nWE、nOE配合，读写取半字单元的低8位。

ARM与2x16bit RAM接口电路示例

3.存储器接口电路以及工作原理



ARM与2x16bit RAM接口电路示例

4.配置总线宽度与引脚时序

	BWSCON	位	描述	初始值
Bank7	ST7	[31]	SRAM对7号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank7	WS7	[30]	7号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank7	DW7	[29:28]	7号存储块的数据总线宽度控制位。 00 = 8bit; 01= 16bit; 10 = 32bit; 11 =保留。	0
Bank6	ST6	[27]	SRAM对6号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0
Bank6	WS6	[26]	6号存储块的等待状态控制位。 0 = WAIT不使能; 1 = WAIT使能。	0
Bank6	DW6	[25:24]	决定对于6号存储块的数据总线宽度。 00 = 8bit; 01 = 16bit; 10 = 32bit; 11 =保留。	0
	ST5	[23]	SRAM对5号存储块是否使用UB/LB控制位。 0 = 不使用UB/LB (引脚对应nWBE[3:0]); 1 = 使用UB/LB (引脚对应nBE[3:0])。	0

(1).BWSCON(0x48000000):

设定Bank6的数据宽度及WAIT使能。

位[25:24]=0b10 32位数据总线[15:0]

位[26]=0b1 WAIT使能

位[27]=0b1 使用UB/LB

ARM与2x16bit RAM接口电路示例

(2)BANKCON6配置

[16:15]=0b00 SRAM

依据SRAM参数配置

- 确定nGCSn信号有效前的地址建立时间
- 确定nOE信号有效前的片选建立时间
- 确定访问周期
- 确定nOE信号失效后片选信号保持的时间
- 确定nGCSn信号失效后有效地址保持的时间

BANKCON6	位	描述	初始
MT	[16:15]	确定6号存储块和7号存储块的存储器类型 00 = ROM or SRAM 01 = FP DRAM 10 =EDO DRAM 11 = SDRAM	11
当存储器类型为SRAM或ROM时(MT = 00)，需用下面15位			
Tacs	[14:13]	确定nGCSn信号有效前的地址建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcos	[12:11]	确定nOE信号有效前的片选建立时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacc	[10:8]	确定访问周期，注意，当nWAIT信号有效时，访问周期>=4时钟周期 000 = 1时钟 001 = 2时钟 010 = 3时钟 011 = 4时钟 100 = 6时钟 101 = 8时钟 110 = 10时钟 111 = 14时钟	111
Tcoh	[7:6]	确定nOE信号失效后片选信号保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tcah	[5:4]	确定nGCSn信号失效后有效地址保持的时间 00 = 0时钟 01 = 1时钟 10 = 2时钟 11 = 4时钟	00
Tacp	[3:2]	确定页模式访问周期 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	00
PMC	[1:0]	确定页模式 00 = 常规（1data） 01 = 4 data 10 = 8 data 11 = 16 data	00
当存储器类型为SDRAM(MT = 11)时，需用下面4位			
Trcd	[3:2]	确定RAS对CAS的延时 00 = 2时钟 01 = 3时钟 10 = 4时钟 11 = 6时钟	10
SCAN	[1:0]	确定列地址位数 00 = 8位 01 = 9位 10 = 10位	00

ARM与2x16bit RAM接口电路示例

5.可用存储器指令

由于数据传送使用数据总线【31：0】，索引可用存储指令为,

LDM、STM多字读写

LDR、STR字读写

LDRH、STRH 半字读写

LDRB、STRB　字节读写

SWPB　字节交换

SWP　字交换

ARM与2x16bit RAM接口电路示例

6.寻址范围:BANK6 0x30000000—0x37FFFFFF

(1)A[31:30]=0b00

(2)A[29:27]=0b110　BANK6　GCS6

(3)A[26:17]=0bXXX XXXX XX　　x取值0或1

(4)A[17:2]连接SRAM的A[15:0]

(5)A1A0=0b00，nBE0有效，　读写芯片1半字单元的低8位；

A1A0=0b01，nBE1有效，　读写芯片1半字单元的高8位；

A1A0=0b10，nBE2有效，　读写芯片2半字单元的低8位；

A1A0=0b11，nBE3有效，　读写芯片2半字单元的高8位；

- 存储芯片第一个字单元地址:

A[31:0]=0b 0011 0xxx xxxx xx00 0000 0000 0000 00xx

- 存储芯片最后一个半字单元地址:

A[31:0]=0b 0011 0xxx xxxx xx11 1111 1111 1111 11xx

SRAM芯片寻址范围(半字):

0b 0011 0xxx xxxx xx00 0000 0000 0000 00xx 到

0b 0011 0xxx xxxx xx11 1111 1111 1111 11xx

ARM与2x16bit RAM接口电路示例

7.读写指令执行时，引脚时序关系(以读指令LDR为例)

(1)流水线译码部件对指令译码，解析出地址数据，依据此数据控制地址引脚A[31:0]产生相应的高低电平信号,

首先是A[26：0]引脚线产生电信号，用于锁定存储芯片1、存储芯片2的半字单元;

其次是A[29:27]=0b110, A[26：0]信号稳定后，nGCS6产生电信号，控制SRAM1、SRAM2片选信号有效，从而SRAM1、SRAM2工作;

然后是nOE有效，将锁定的芯片1半字单元数据放入数据总线D[15:0]上;

将锁定的芯片2半字单元数据放入数据总线D[31:16]上;

(2)流水线执行部件，从数据总线D[31:0]上将数据读入寄存器[31:0].

(3)nGCS6失效、地址信号失效哦啊，一次读数据过程结束。

Nand Falsh接口电路—1 K9F1208U0C芯片参数

K9F1208U0C 芯片是典型的NAND flash，韩国三星公司生产，主要特点如下：

（1）容量：64M×8 bit。

（2）工作电压：2.7～3.6V。

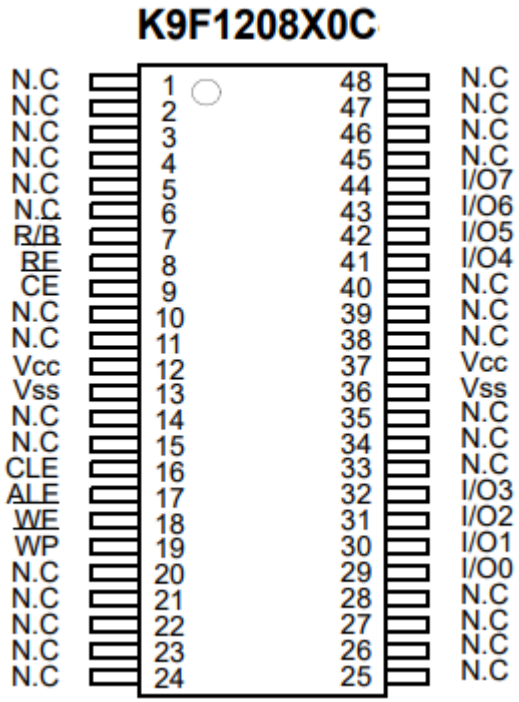
（3）内部存储结构：528字节×32页×4096块。

（4）页大小：512+16字节。

（5）块大小：16K+512字节。

（6）可实现擦除、读/写操作。

（7）内部有命令寄存器。



Nand Falsh接口电路—2 接口电路--1

1.nFCE首先有效，

使K9F1208UC工作

2.如果S3C2440要

读\擦除\写数据的话，

首先查询K9F1208U0C

是否忙，即RnB

3.信号线

IO[7:0]用于传送命令、地址、

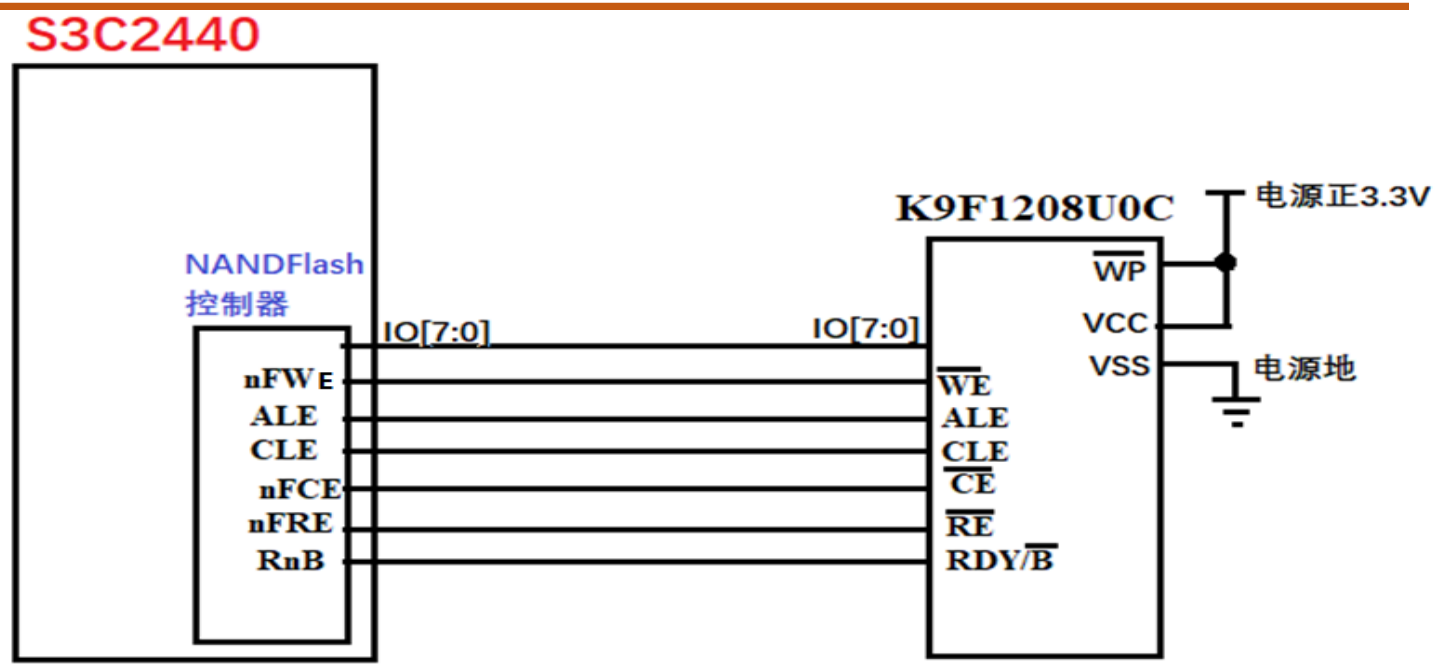
数据

CLE有效时传送命令

ALE有效时，传送地址

nFRE有效时，读操作，Nand Flash控制器从IO[7:0]读数据

nFWE有效时，写操作，Nand Flash控制器向IO[7:0]写数据



Nand Falsh接口电路—2 接口电路--2

S3C2440中Nand flash 引脚状态定义

NCON0	GPG13	GPG14	GPG15
0: Normal NAND	0: 256Words	0: 3-Addr	0: 8-bit bus width
	1: 512Bytes	1: 4-Addr	
1: Advance NAND	0: 1Kwords	0: 4-Addr	1: 16-bit bus width
	1: 2Kbytes	1: 5-Addr	

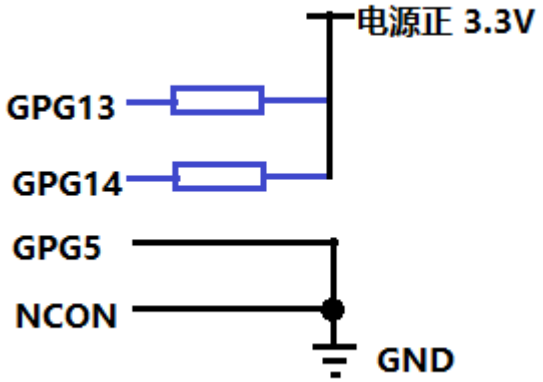
- K9F1208 一页:512byte，

  NCON 接低电平，

  GPG13 接高电平。
- K9F1208 需要4 个寻址命令

  GPG14 接高电平。
- K9F1208 的数据位宽是8

  GPG15 接低电平。



Nand Falsh接口电路—3 从F9K1208U0C读取一页-1

- NAND Flash读数据一次读一页数据。
- 读过程:

  在I/O[7:0]发送读数据命令

  K9F1208有2个读命令：0x00：使A[8]=0，选上半页。0x01：使A[8]=1，选下半页

  在I/O[7:0]发送要读取页的地址:

  4个地址序列（cycle）：A[7:0]、A[16:9]、A[24:17]、A[25]， A[8]用命令控制。

  在I/O[7:0]读取一页的数据

Nand Falsh接口电路—3 从F9K1208U0C读取一页-2

读操作过程：

（1）发送读命令。

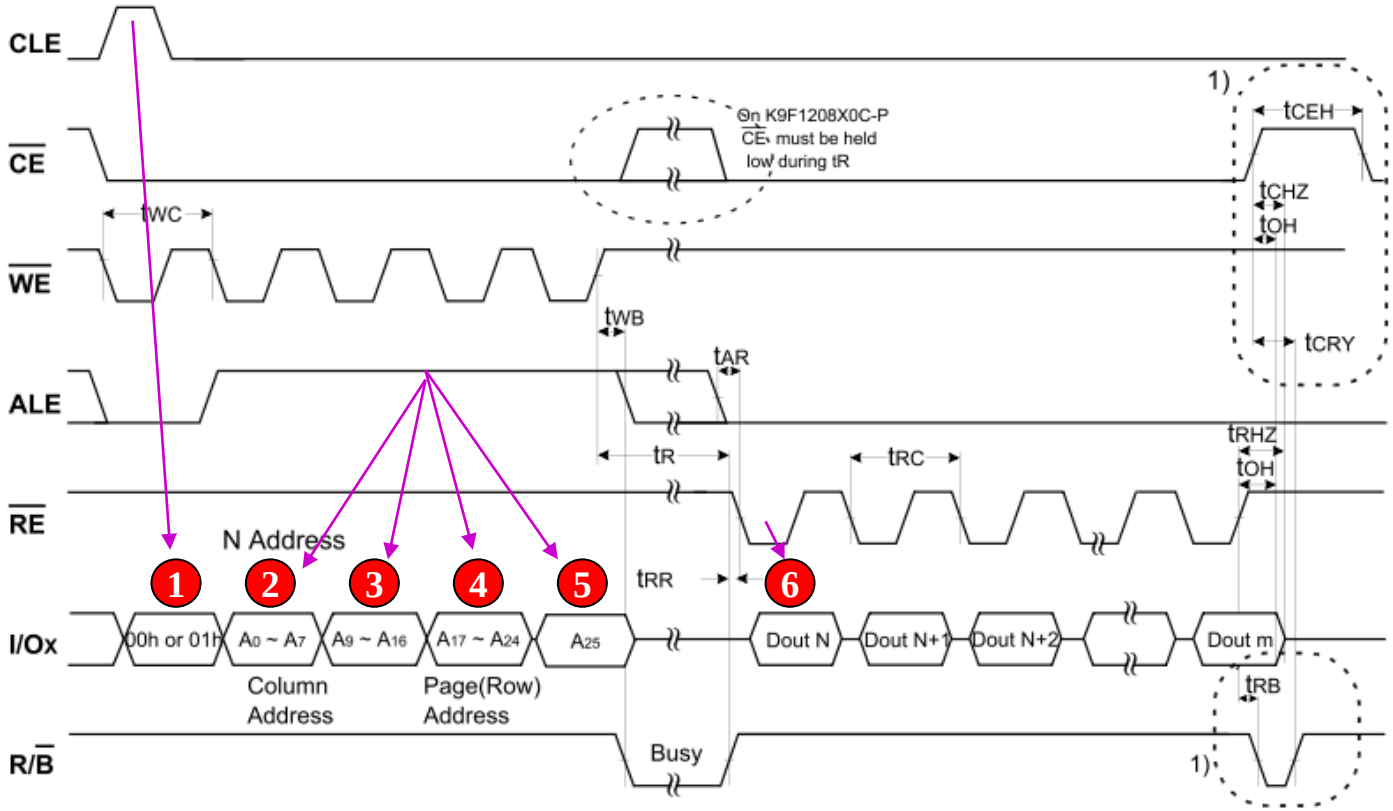
（2）发送第1个cycle地址A[7:0]。

（3）发送第2个cycle地址A[16:9]。

（4）发送第3个cycle地址A[24:17]。

（5）发送第4个cycle地址A[25]。

（6）读数据至页末。



Nand Falsh接口电路—3 从F9K1208U0C读取一页函数-3

```
#define  READCMD0   0                          // 读命令0，512字节的1st half部分
static void ReadPage (U32 addr, U8 *buf)       // 整页读取，addr：flash中的页地址，  buf:存放读取数据
                                               // 即 flash地址>>9（右移9位）
{     U16 i;
      rNFCONT&=~0x02;      // NFCONT[1]=0  nFCE低电平；
                           //F9K1208U0C芯片CE低电平,芯片处于工作状态
      rNFCMMD= READCMD0;           // 发送读指令0x00，整页读取   NFCMMD[7:0]
      rNFADDR=0;                   // 写第1个地址，即A[7:0]，取0 ,页地址A[9:25]   NFADDR[7:0]
      rNFADDR=addr;                // 写第2个地址，即A[9:16]
      rNFADDR=addr>>8;             // 写第3个地址，即A[17:24]
      rNFADDR=addr>>16;            // 写第4个地址，即A[25]
      InitEcc( );                  // 启动ECC校验
     while(!(rNFSTAT&0x1));        //rNFSTAT[0]=0   F9K1208U0C忙则等待
    for(i=0; i<512; i++) buf[i] = rNFDATA;        // 循环读出一页数据
      rNFCONT|=0x02;               // NFCONT[1]=1  nFCE高电平；
                                   // F9K1208U0C芯片片选信号无效
}
```

Nand Falsh接口电路—4 F9K1208U0C擦除1块-1

- F9K1208U0C 一次擦除一块。
- F9K1208U0C写一页以前必须先擦除，因为F9K1208U0C中的位只能从1变成0，不能从0变成1，所以要想写进去必须通过擦除命令把相关块擦除为全1。
- ECC：Error Checking and Correcting，错误检查与校验。ECC能纠正单比特错误和检测双比特错误，而且计算速度很快，但对1比特以上的错误无法纠正，对2比特以上的错误不保证能检测。

Nand Falsh接口电路—4 F9K1208U0C擦除1块-2

擦除操作过程：

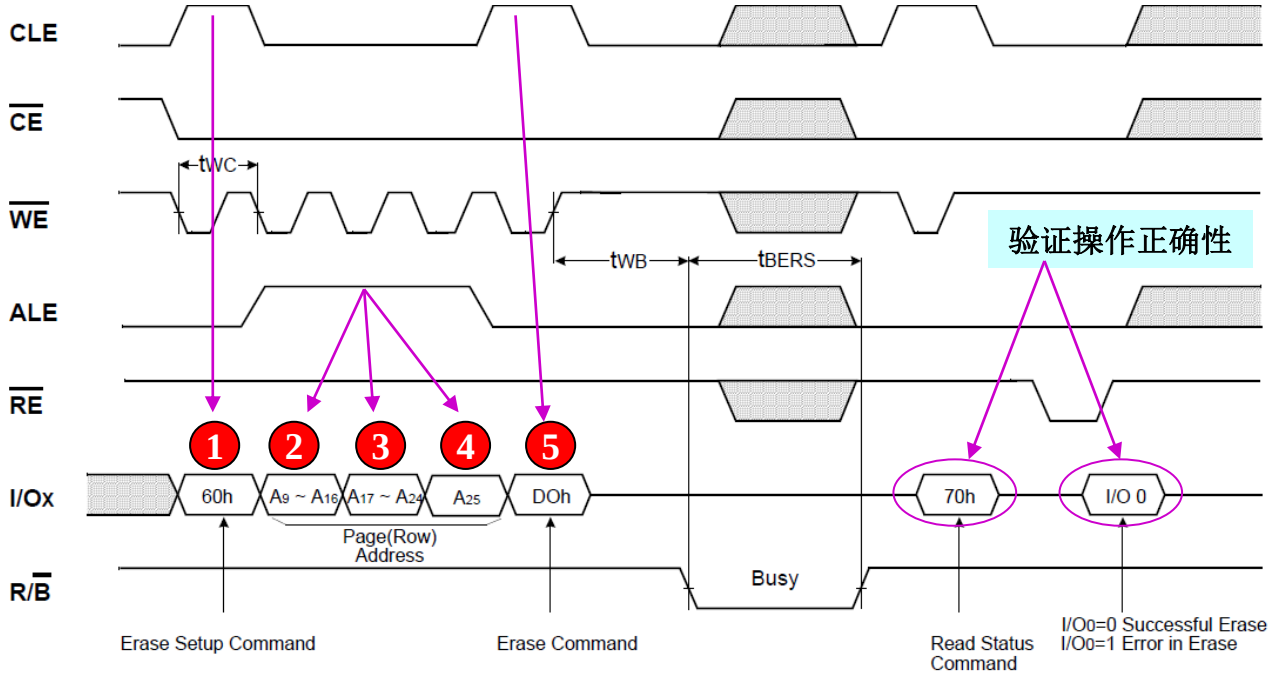
（1）发送擦除开始命令。

（2）发送第2个cycle地址A[16:9]。

（3）发送第3个cycle地址A[24:17]。

（4）发送第4个cycle地址A[25]。

（5）发送擦除结束命令。



Nand Falsh接口电路—4 F9K1208U0C擦除1块函数-3

```
static U32 EraseBlock(U32 addr)     // 整块擦除（即将整块的数据位全变为1）
{
    addr &= ~0x1f;                  // 因为是按块擦除，所以page地址清0
     rNFCONT&=~0x02;       // NFCONT[1]=0  nFCE低电平；
                           //NandFlash芯片CE低电平,芯片处于工作状态
    rNFCMMD= ERASECMD0;          // 发送擦除命令0x60
     rNFADDR=addr;               // 写第2个地址，即A[9:16]
    rNFADDR=addr>>8;             // 写第3个地址，即A[17:24]
    rNFADDR=addr>>16;            // 写第4个地址，即A[25]
     rNFCMMD= ERASECMD1;                 // 发送擦除命令0xD0
   while(!(rNFSTAT&0x1));        //rNFSTAT[0]=0 忙则等待
    rNFCONT|=0x02;                       // 片选禁止，释放NandFlash
}
```

Nand Falsh接口电路—5 F9K1208U0C写数据-1

写操作过程：

（1）发送写开始指令。

（2）发送第1个cycle地址A[7:0] 。

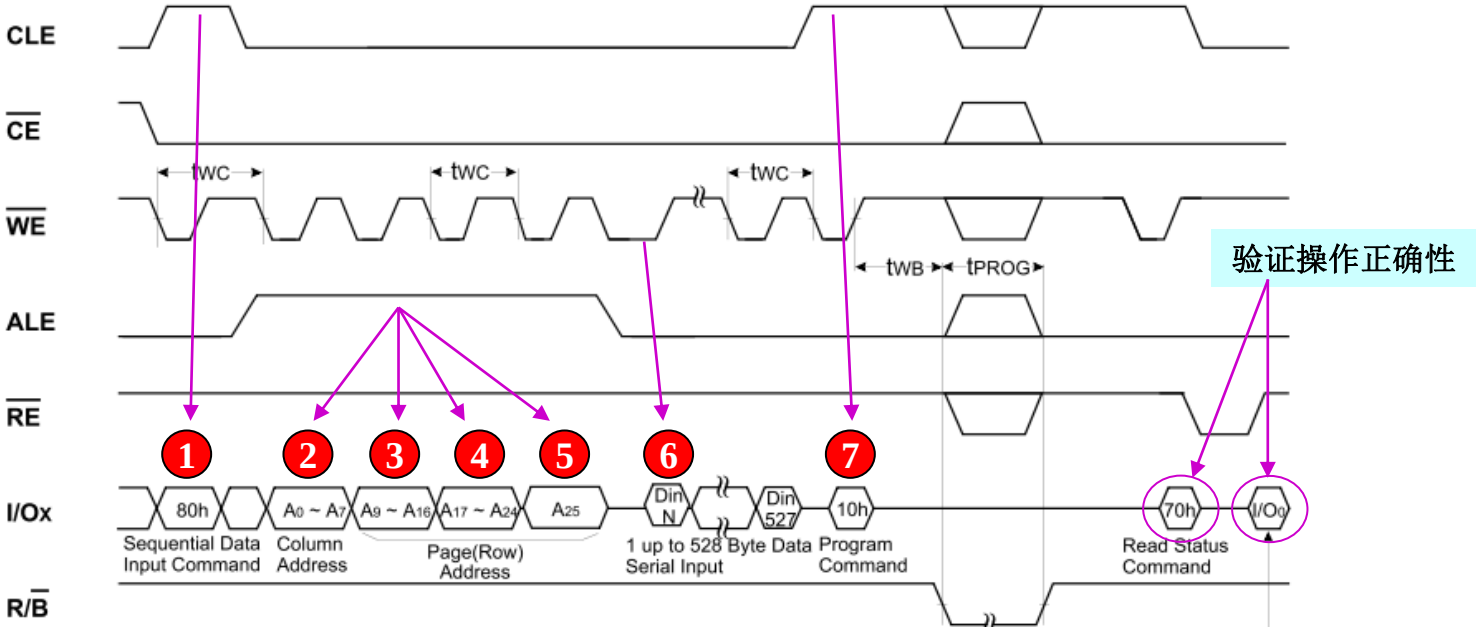
（3）发送第2个cycle地址A[16:9] 。

（4）发送第3个cycle地址A[24:17] 。

（5）发送第4个cycle地址A[25] 。

（6）写入数据至页末。

（7）发送写结束指令。



Nand Falsh接口电路—4 F9K1208U0C写数据函数-2

```
static U32 WritePage(U32 addr, U8 *buf)    // 写整页，addr：flash中的第几页，
                                                    // 即 flash地址>>9（右移9位）
{
    rNFCONT&=~0x02;      // NFCONT[1]=0  nFCE低电平；
                         //NandFlash芯片CE低电平,芯片处于工作状态
    rNFCMMD= PROGCMD0;          // 发送写开始指令0x80
    rNFADDR=0;                  // 写第1个地址，即A[7:0]
    rNFADDR=addr;                       // 写第2个地址，即A[9:16]
    rNFADDR=addr>>8;            // 写第3个地址，即A[17:24]
    rNFADDR=addr>>16;           // 写第4个地址，即A[25]
    InitEcc( );                 // 启动ECC校验
    while(!(rNFSTAT&0x1));      //rNFSTAT[0]=0 忙则等待
    for(i=0; i<512; i++)
        rNFDATA buf[i];         // 循环写一页数据
    rNFCMMD= PROGCMD1;          // 发送写结束指令0x10
    rNFCONT|=0x02;              // 片选禁止，释放NandFlash
}
```

S3C2440 GPIO示例-流水灯　01-题目

设计LED接口电路，其中LED为8个.编写程序实现流水灯效果

S3C2440 GPIO示例-流水灯　02-接口电路

8个LED接口电路，以LED1为例

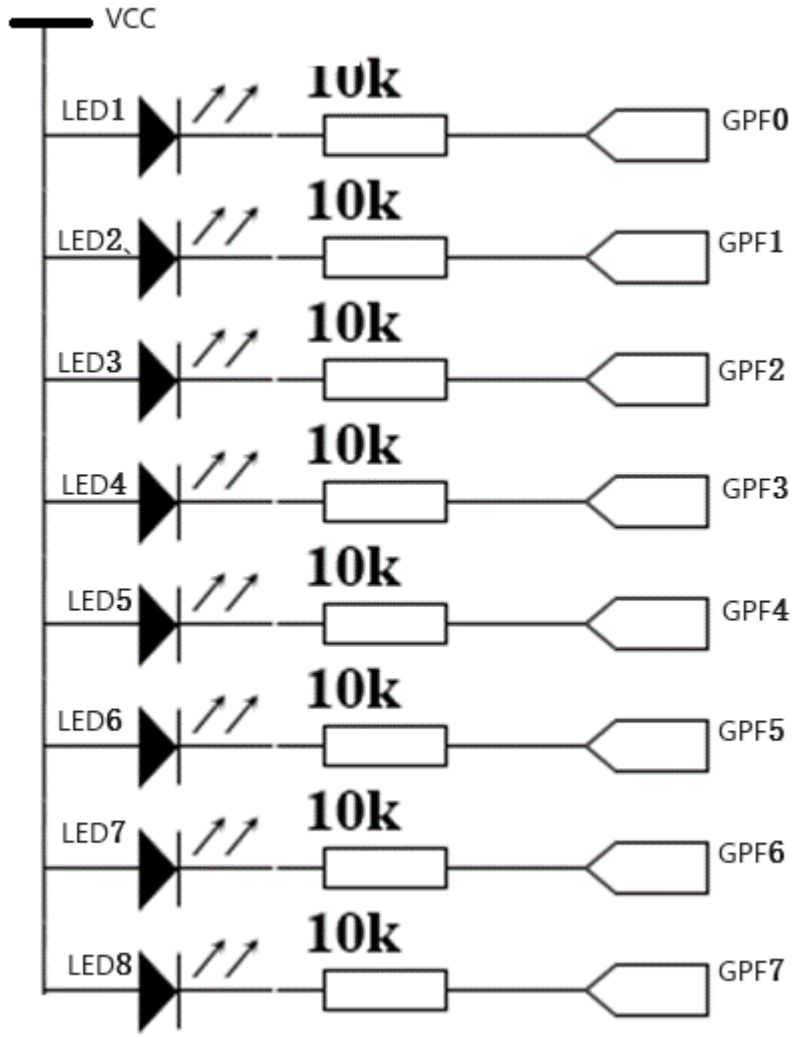
1. 电路连接：端口F的引脚GPF0通过电阻与LED1的输出连接；
　LED1的输入连到电源正VCC

2. 工作原理

GPF0输出高电平时，LED1输入高电平、输出高电平，LED1不亮；

GPF0输出低电平时，LED1输入高电平、输出低电平，LED1亮；

所以控制GPF0引脚输出高/低电平，从而控制LED1不亮/亮



S3C2440 GPIO示例-流水灯　03-分析

1.配置GPF[7:0]为GPIO输出，并且内接上拉电阻

- GPFCON配置数据=2_0101 0101 0101 0101

=0x5555

GPF[7:0]引脚都为GPIO输出

- GPFUP配置数据=2_0000 0000

=0x00

GPF[7:0]引脚连接内部上拉电阻

GPFCON	位	描述			初始值
GPF7	[15:14]	00=输入；	01=输出；	10= EINT[7]； 11=保留。	00
GPF6	[13:12]	00=输入；	01=输出；	10= EINT[6]； 11=保留。	00
GPF5	[11:10]	00=输入；	01=输出；	10= EINT[5]； 11=保留。	00
GPF4	[9:8]	00=输入；	01=输出；	10= EINT[4]； 11=保留。	00
GPF3	[7:6]	00=输入；	01=输出；	10= EINT[3]； 11=保留。	00
GPF2	[5:4]	00=输入；	01=输出；	10= EINT[2]； 11=保留。	00
GPF1	[3:2]	00=输入；	01=输出；	10= EINT[1]； 11=保留。	00
GPF0	[1:0]	00=输入；	01=输出；	10= EINT[0]； 11=保留。	00

GPFUP	位	描述	初始值
GPF 7:0	[7:0]	1=对应的GPIO引脚上拉电阻不使能； 0=对应的GPIO引脚上拉电阻使能。	0x000

S3C2440 GPIO示例-流水灯　03-分析

2.寄存器变量定义

```
#define rGPFCON (*((volatile unsigned char *) 0x56000050))
#define rGPFDAT (*((volatile unsigned char *) 0x56000054))
#define rGPFUP (*((volatile unsigned char *) 0x56000058))
```

3.引脚初始化函数

```
void  PortF_Init(void)
{
 rGPFCON=0x5555;
 rGPFUP=0x00;
}
```

S3C2440 GPIO示例-流水灯　03-分析

4.LED灯控制数据

- 仅LED1亮：　GPF7-GPF0 = 1111 1110=0xFE,　GPFDAT=0xFE
- 仅LED2亮：　GPF7-GPF0 = 1111 1101=0xFD,　GPFDAT=0xFD
- 仅LED3亮：　GPF7-GPF0 = 1111 1011=0xFB,　GPFDAT=0xFB
- 仅LED4亮：　GPF7-GPF0 = 1111 0111=0xF7,　GPFDAT=0xF7
- 仅LED5亮：　GPF7-GPF0 = 1110 1111=0xEF,　GPFDAT=0xEF
- 仅LED6亮：　GPF7-GPF0 = 1101 1111=0xDF,　GPFDAT=0xDF
- 仅LED7亮：　GPF7-GPF0 = 1011 1111=0xBF,　GPFDAT=0xBF
- 仅LED8亮：　GPF7-GPF0 = 0111 1111=0x7F,　GPFDAT=0x7F
- 全灭：　　GPF7-GPF0 = 1111 1111=0xFF,　GPFDAT=0xFF
- 全亮:　　GPF7-GPF0 = 0000 0000=0x00,　GPFDAT=0x00

编码表:

```
LED_CODE[]={0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF, 0x7F,0xFF, 0x00};
```

S3C2440 GPIO示例-流水灯　03-分析

5. LED流水灯显示函数

```
int Ledlamp( )                          //流水灯控制程序
{
   unsigned char ledtab[]={0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF， 0x7F, 0xFF， 0x00};//流水灯点亮数据,编码表
   int i;
   while (1)
   {
            for ( i=0; i<8; i++ )          //循环点亮8个发光二极管
            {     rGPFDAT=ledtab[i];       //点亮发光二极管LEDi
                  Delay(10);               //延时
            }
   } return(0);
}
```

S3C2440 GPIO示例-流水灯　04-完整程序

```
#define rGPFCON (*(volatile unsigned *)0x56000050)          //定义GPFCON地址
#define rGPFDAT (*(volatile unsigned *)0x56000054)          //定义GPFDAT地址
#define rGPFUP (*(volatile unsigned *)0x56000058)           //定义GPFUP地址
void Delay(unsigned int);                  //无返回结果， LED点亮的延时程序
int Ledlamp( )                             //流水灯显示函数
{
   unsigned char ledtab[]={0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF， 0x7F, 0xFF， 0x00};//流水灯点亮数据,编码表
   int i;
   while (1)
   {
            for ( i=0; i<8; i++ )          //循环点亮8个发光二极管
            {    rGPFDAT=ledtab[i];        //点亮发光二极管LEDi
                 Delay(10);                //延时
            }
   }
 return(0);
}
```

S3C2440 GPIO示例-流水灯　04-完整程序

```
void Delay ( unsigned int x )                    //延时程序
{   unsigned int i,j,k;
    for(i=0;i<=x;i++)
            for(j=0;j<=0xff;j++)
                    for(k=0;k<=0xff;k++);
}
```

//主程序

```
#include <stdio.h>
int main(void)
{  Ledlamp( );
   return 0;
}
```

S3C2440 GPIO示例-流水灯　04-完整程序

```
void Delay ( unsigned int x )                    //延时程序
{   unsigned int i,j,k;
    for(i=0;i<=x;i++)
            for(j=0;j<=0xff;j++)
                    for(k=0;k<=0xff;k++);
}
```

//主程序

```
#include <stdio.h>
int main(void)
{  Ledlamp( );
   return 0;
}
```

S3C2440中断　14-中断示例　01-题目

接口电路图如下，

使用F端口引脚作为LED的控制端；

使用GPG0引脚获取K1按键操作信息；

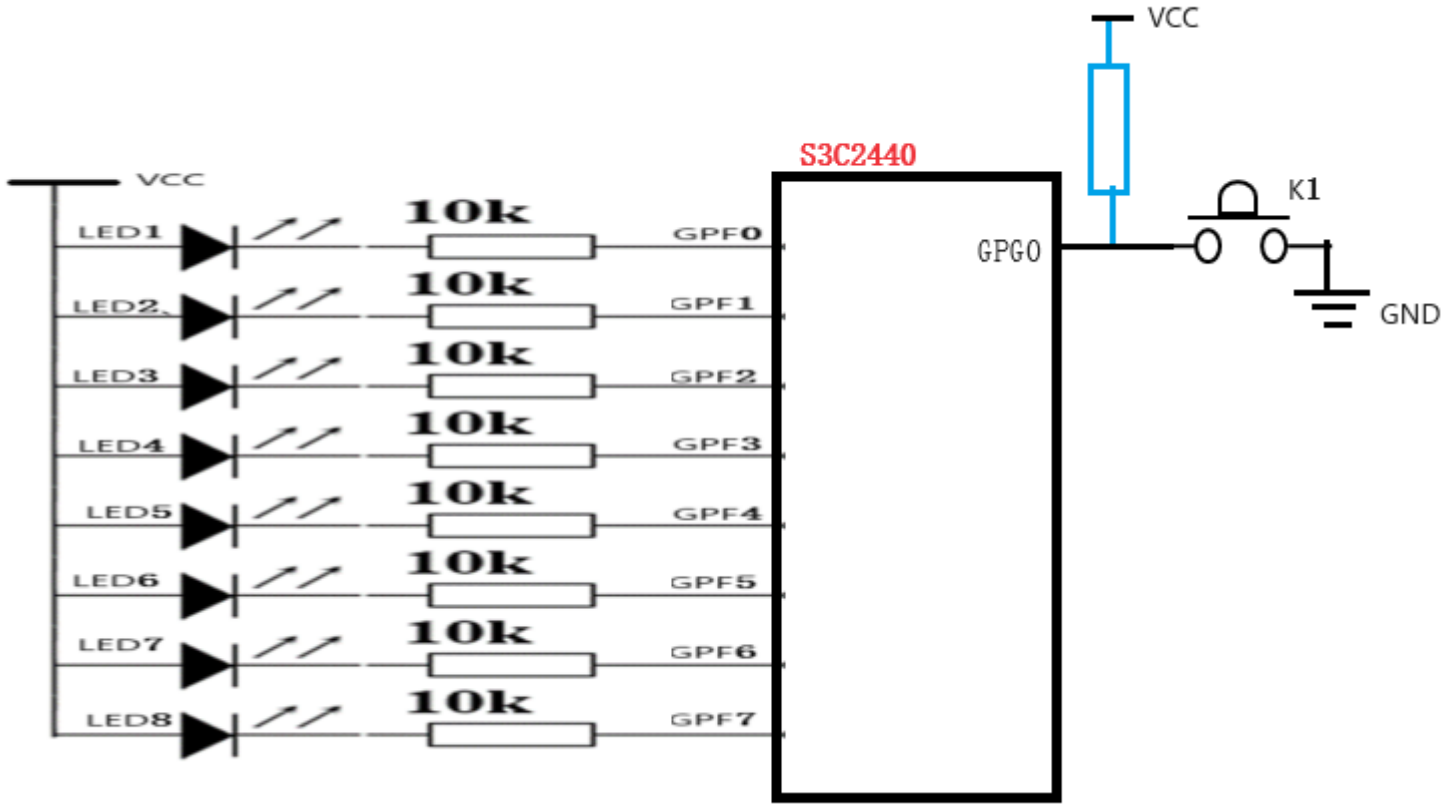
K1键初始状态，LED从左向右流水显示；

单击K1键，LED流水显示变向，即原来从左向右流水显示变为从右向左流水显示原来从右向左流水显示变为从左向右流水显示；

回答问题:

(1)说明接口电路工作原理?

(2)编写程序实现以上功能?



S3C2440中断　14-中断示例　02-接口电路工作原理

(1)LED1为例

当控制端GPF0输出高电平时，LED1不亮;

当控制端GPF0输出低电平时，LED1亮;

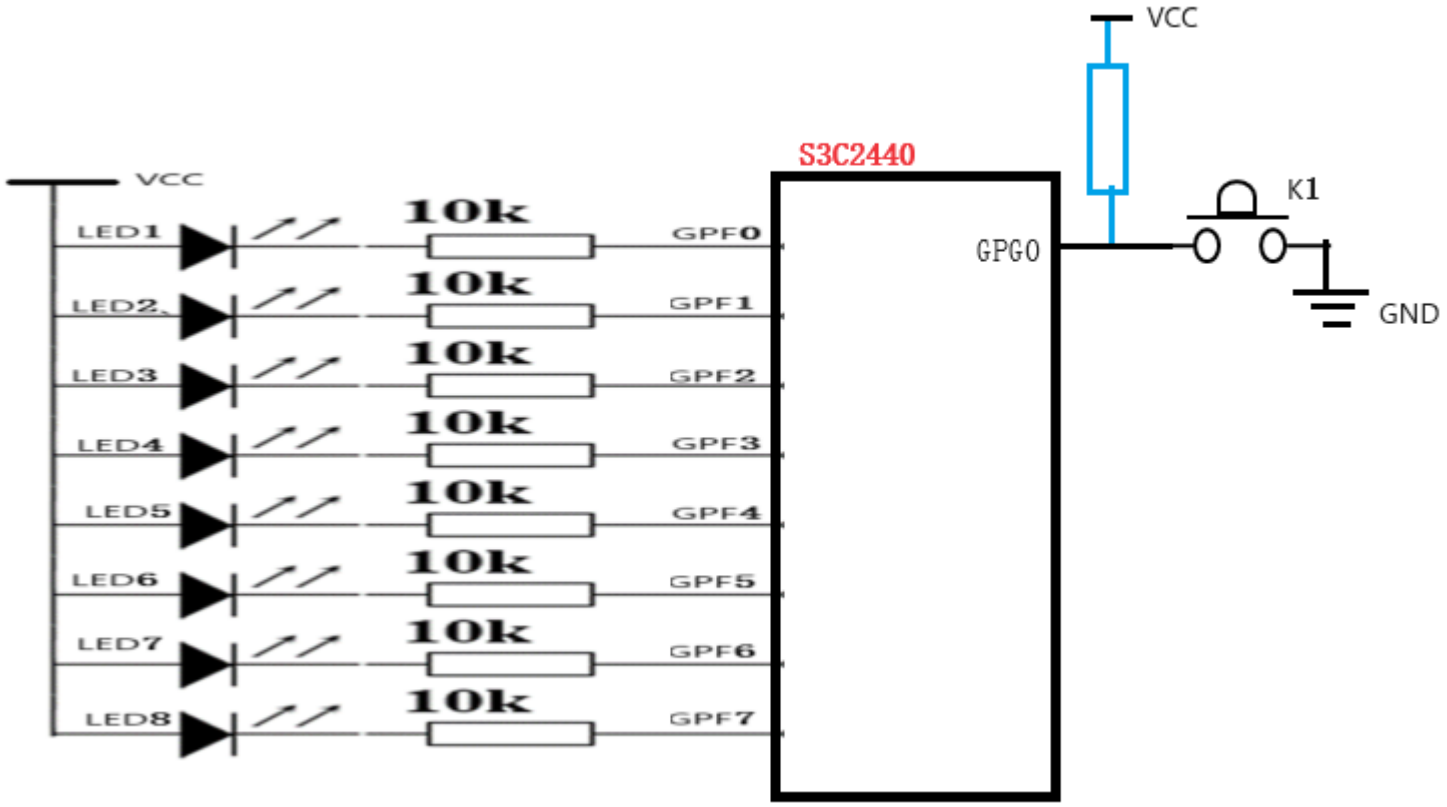
(2)K1按键

K1按钮处于未按下状态，GPG0高电平;

K1按钮处于按下状态，GPG0低电平;

一次单击K1,在GPG0输入一个负脉冲信号，

正好可以作为中断请求信号。



S3C2440中断 14-中断示例 03 引脚初始化

构建端口F引脚配置数据

(1)GPF[7:0]用于控制LED【7：0】亮与灭，

所以都要配置为输出

GPFCON=2_0101 0101 0101 0101

=0x5555

(2) GPF[7:0]引脚内接上拉电阻

GPFUP=2_0000 0000=0x00

GPFCON	Bit	Description
GPF7	[15:14]	00 = Input 01 = Output 10 = EINT[7] 11 = Reserved
GPF6	[13:12]	00 = Input 01 = Output 10 = EINT[6] 11 = Reserved
GPF5	[11:10]	00 = Input 01 = Output 10 = EINT[5] 11 = Reserved
GPF4	[9:8]	00 = Input 01 = Output 10 = EINT[4] 11 = Reserved
GPF3	[7:6]	00 = Input 01 = Output 10 = EINT[3] 11 = Reserved
GPF2	[5:4]	00 = Input 01 = Output 10 = EINT2] 11 = Reserved
GPF1	[3:2]	00 = Input 01 = Output 10 = EINT[1] 11 = Reserved
GPF0	[1:0]	00 = Input 01 = Output 10 = EINT[0] 11 = Reserved

GPFUP	Bit	Description
GPF[7:0]	[7:0]	0: The pull up function attached to the corresponding port pin is enabled. 1: The pull up function is disabled.

S3C2440中断 14-中断示例 03-引脚初始化

构建GPG0引脚配置数据

(1)单击K1在GPG0引脚输入一个负脉冲，可以作为中断请求信号.

需要将GPG0配置为EINT8功能

GPGCON=2_0000 0000 0000 0010

=0x0002

(2)GPG0作为EINT8引脚，所以不适用内部上拉电阻

GPGUP=2_0000 0000 0000 0001

=0x01

GPGCON	Bit	Description	
GPG7	[15:14]	00 = Input 10 = EINT[15]	01 = Output 11 = SPICLK1
GPG6	[13:12]	00 = Input 10 = EINT[14]	01 = Output 11 = SPIMOSI1
GPG5	[11:10]	00 = Input 10 = EINT[13]	01 = Output 11 = SPIMISO1
GPG4	[9:8]	00 = Input 10 = EINT[12]	01 = Output 11 = LCD_PWRDN
GPG3	[7:6]	00 = Input 10 = EINT[11]	01 = Output 11 = nSS1
GPG2	[5:4]	00 = Input 10 = EINT[10]	01 = Output 11 = nSS0
GPG1	[3:2]	00 = Input 10 = EINT[9]	01 = Output 11 = Reserved
GPG0	[1:0]	00 = Input 10 = EINT[8]	01 = Output 11 = Reserved

GPGUP	Bit	Description
GPG[15:0]	[15:0]	0: The pull up function attached to the corresponding port pin is enabled. 1: The pull up function is disabled.

S3C2440中断　14-中断示例　03-引脚初始化

引脚初始化函数

(1)寄存器变量定义

```
#define rGPFCON (*((volatile unsigned char *) 0x56000050))
#define rGPFDAT (*((volatile unsigned char *) 0x56000054))
#define rGPFUP (*((volatile unsigned char *) 0x56000058))
#define rGPGCON (*((volatile unsigned char *) 0x56000060))
#define rGPGDAT (*((volatile unsigned char *) 0x56000064))
#define rGPGUP (*((volatile unsigned char *) 0x56000068))
```

(2)引脚初始化函数

```
Void PortPin_Init()
{
rGPFCON=0x5555;
rGPFUP=0x00;
rGPGCON=0x0002;
rGPGUP=0x01;
}
```

S3C2440中断　14-中断示例　04-构建LED显示数据

仅LED1亮 GPFDAT=2_1111 1110=0xFE

仅LED2亮 GPFDAT=2_1111 1101=0xDF

仅LED3亮 GPFDAT=2_1111 1011=0xBF

仅LED4亮 GPFDAT=2_1111 0111=0x7F

仅LED5亮 GPFDAT=2_1110 1111=0xEF

仅LED6亮 GPFDAT=2_1101 1111=0xDF

仅LED7亮 GPFDAT=2_1011 1111=0xBF

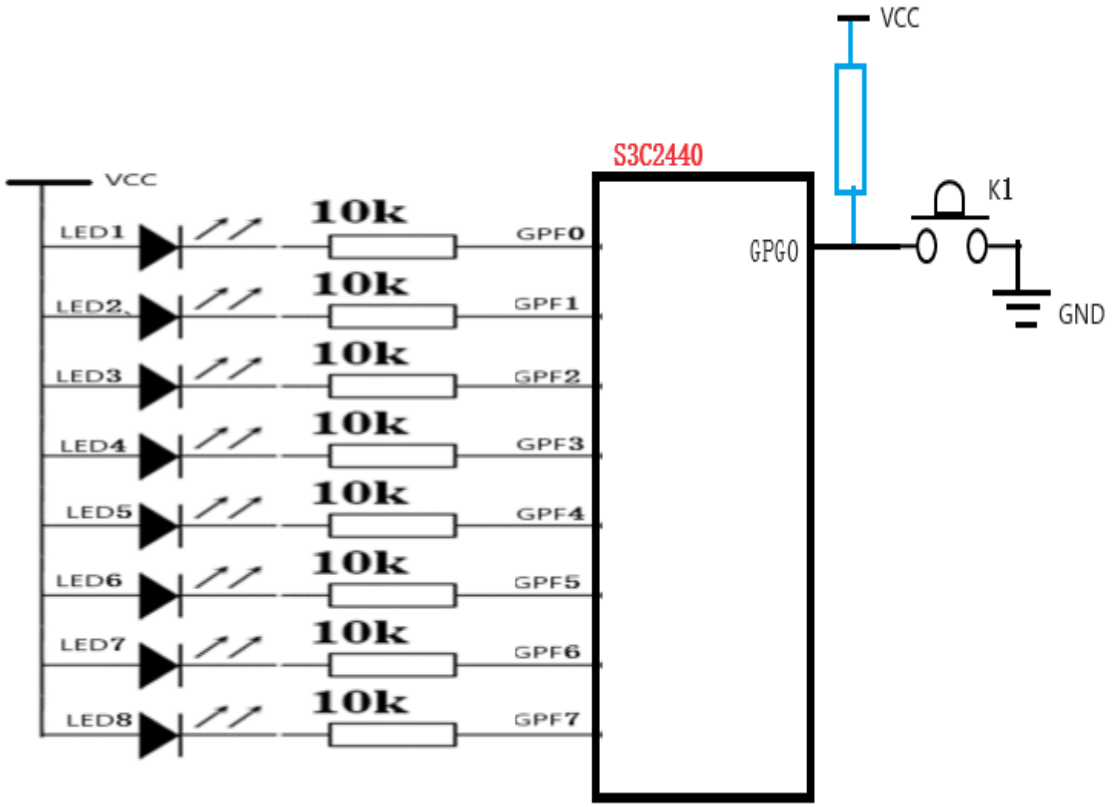
仅LED8亮 GPFDAT=2_0111 1111=0x7F

全亮 GPFDAT=2_0000 0000=0x00

全灭 GPFDAT=2_1111 1111=0xFF

编码表:

LED_Encode[]={0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,0x00,0xFF}



S3C2440中断　14-中断示例　05-延时子程序

1.作用：

LED从亮到灭或从灭到亮有物理作用时间，程序中需要有个延时程序与之匹配,所以定义延时子程序.

2.子程序

```
void Delay ( unsigned int x )                    //延时程序
{
    unsigned int i,j,k;
    for(i=0;i<=x;i++)
            for(j=0;j<=0xff;j++)
                    for(k=0;k<=0xff;k++);
}
```

S3C2440中断　14-中断示例　06-中断初始化

1.EINT8触发方式:下降沿触发

rEXTINT1=2_0000 0000 0000 0000 0000 0000 0000 0010

=0x02

EXTINT1	Bit	Description
FLTEN12	[19]	Filter enable for EINT12 0 = Filter Disable　　1 = Filter Enable
EINT12	[18:16]	Setting the signaling method of the EINT12. 000 = Low level　001 = High level　01x = Falling edge triggered 10x = Rising edge triggered　11x = Both edge triggered
FLTEN11	[15]	Filter enable for EINT11 0 = Filter Disable　　1 = Filter Enable
EINT11	[14:12]	Setting the signaling method of the EINT11. 000 = Low level　001 = High level　01x = Falling edge triggered 10x = Rising edge triggered　11x = Both edge triggered
FLTEN10	[11]	Filter enable for EINT10 0 = Filter Disable　　1 = Filter Enable
EINT10	[10:8]	Setting the signaling method of the EINT10. 000 = Low level　001 = High level　01x = Falling edge triggered 10x = Rising edge triggered　11x = Both edge triggered
FLTEN9	[7]	Filter enable for EINT9 0 = Filter Disable　　1 = Filter Enable
EINT9	[6:4]	Setting the signaling method of the EINT9. 000 = Low level　001 = High level　01x = Falling edge triggered 10x = Rising edge triggered　11x = Both edge triggered
FLTEN8	[3]	Filter enable for EINT8 0 = Filter Disable　　1 = Filter Enable
EINT8	[2:0]	Setting the signaling method of the EINT8. 000 = Low level　001 = High level　01x = Falling edge triggered 10x = Rising edge triggered　11x = Both edge triggered

S3C2440中断　14-中断示例　06-中断初始化

2.EINTMASK:

rEINTMASK=2_0000 0000 0000 0000 0000 0000

=0x000000

EINTMASK	Bit	Description
EINT23	[23]	0 = enable interrupt 1= masked
EINT22	[22]	0 = enable interrupt 1= masked
EINT21	[21]	0 = enable interrupt 1= masked
EINT20	[20]	0 = enable interrupt 1= masked
EINT19	[19]	0 = enable interrupt 1= masked
EINT18	[18]	0 = enable interrupt 1= masked
EINT17	[17]	0 = enable interrupt 1= masked
EINT16	[16]	0 = enable interrupt 1= masked
EINT15	[15]	0 = enable interrupt 1= masked
EINT14	[14]	0 = enable interrupt 1= masked
EINT13	[13]	0 = enable interrupt 1= masked
EINT12	[12]	0 = enable interrupt 1= masked
EINT11	[11]	0 = enable interrupt 1= masked
EINT10	[10]	0 = enable interrupt 1= masked
EINT9	[9]	0 = enable interrupt 1= masked
EINT8	[8]	0 = enable interrupt 1= masked
EINT7	[7]	0 = enable interrupt 1= masked
EINT6	[6]	0 = enable interrupt 1= masked
EINT5	[5]	0 = enable interrupt 1= masked
EINT4	[4]	0 = enable interrupt 1= masked
Reserved	[3:0]	Reserved

S3C2440中断　14-中断示例　06-中断初始化

3.EINT8配置为IRQ模式

EINT8属于模块EINT8_23，复合中断源EINT8_23,INTMOD配置位5=0，IRQ模式

rINTMOD=2_0000 0000 0000 0000 0000 0000 0000 0000

=0x00000000

INT_UART2	[15]
INT_TIMER4	[14]
INT_TIMER3	[13]
INT_TIMER2	[12]
INT_TIMER1	[11]
INT_TIMER0	[10]
INT_WDT_AC97	[9]
INT_TICK	[8]
nBATT_FLT	[7]
INT_CAM	[6]
EINT8_23	[5]
EINT4_7	[4]
EINT3	[3]
EINT2	[2]
EINT1	[1]
EINT0	[0]

Register	Address	R/W	Description	Reset Value
INTMOD	0X4A000004	R/W	Interrupt mode register. 0 = IRQ mode　　1 = FIQ mode	0x00000000

S3C2440中断　14-中断示例　06-中断初始化

4.EINT8配置为开中断

EINT8属于模块EINT8_23，复合中断源EINT8_23,INTMSK配置位5=0，开中断

rINTMSK=2_0000 0000 0000 0000 0000 0000 0000 0000

=0x00000000

INT_UART2	[15]
INT_TIMER4	[14]
INT_TIMER3	[13]
INT_TIMER2	[12]
INT_TIMER1	[11]
INT_TIMER0	[10]
INT_WDT_AC97	[9]
INT_TICK	[8]
nBATT_FLT	[7]
INT_CAM	[6]
EINT8_23	[5]
EINT4_7	[4]
EINT3	[3]
EINT2	[2]
EINT1	[1]
EINT0	[0]

Register	Address	R/W	Description	Reset Value
INTMSK	0X4A000008	R/W	Determine which interrupt source is masked. The masked interrupt source will not be serviced. 0 = Interrupt service is available. 1 = Interrupt service is masked.	0xFFFFFFFF

S3C2440中断　14-中断示例　06-中断初始化

5.清理记录标志

```
rEINTPEND=1<<8;          //EINTPEND[8],写入1，清除EINT8中断请求记录
rSRCPND=1<<5;            //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
rINTPND=1<<5;            //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
```

S3C2440中断 14-中断示例 06-中断初始化

6.EINT8_23中断服务程序

EINT8为二级中断，归模块EINT8_23,在EINT_23中断服务程序内判断EINTPEND[8]位来确定EINT8是否申请中断。

全局变量 LED_Flag=0时，LED流水显示，从左向右；

LED_Flag=1时，LED流水显示，从右向左；

每单击K1，执行一次中断服务程序，中断程序切换LED_Flag的值.

```
static void_irq Eint8_23_ISR(void)              //声明一个Eint8_23_ISR中断处理函数
{
If(rEINTPEND&(1<<8))
{//EINT8中断
  if(LED_Flag==0) LED_Flag=1;  //切换流水灯显示方向
     else              LED_Flag=0;
}//EINT8中断
 rEINTPEND=1<<8;                 //EINTPEND[8],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;                   //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;                  //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

S3C2440中断　14-中断示例　06-中断初始化

7.中断向量地址变量定义

中断向量表首地址=_ISR_STARTADDRESS+0x20　　//EINT0的中断向量地址

EINT8_23中断向量地址=中断向量表首地址+5*4

#define pISR_EINT8_23 (*(unsigned *)(_ISR_STARTADDRESS+0x20+5*4))

INTOFFSET	偏移量	INTOFFSET	偏移量	INTOFFSET	偏移量	INTOFFSET	偏移量
INT_ADC	31	INT_UART1	23	INT_UART2	15	nBATT_FLT	7
INT_RTC	30	INT_SPI0	22	INT_TIMER4	14	INT_CAM	6
INT_SPI1	29	INT_SDI	21	INT_TIMER3	13	EINT8_23	5
INT_UART0	28	INT_DMA3	20	INT_TIMER2	12	EINT4_7	4
INT_IIC	27	INT_DMA2	19	INT_TIMER1	11	EINT3	3
INT_USBH	26	INT_DMA1	18	INT_TIMER0	10	EINT2	2
INT_USBD	25	INT_DMA0	17	INT_WDT_AC97	9	EINT1	1
INT_NFCON	24	INT_LCD	16	INT_TICK	8	EINT0	0

S3C2440中断　14-中断示例　06-中断初始化

8. EINT8_23中断初始化函数

```
#define pISR_EINT8_23 (*(unsigned *)(_ISR_STARTADDRESS+0x20+5*4))
#define rEXTINT1 (*((volatile unsigned char *) 0x5600008C))
#define rEINTPEND (*((volatile unsigned char *) 0x560000A8))
#define rEINTMASK (*((volatile unsigned char *) 0x560000A4))
#define rSRCPND (*((volatile unsigned char *) 0X4A000000))
#define rINTMOD (*((volatile unsigned char *) 0X4A000004))
#define rINTMSK (*((volatile unsigned char *) 0X4A000008))
#define rPRIORITY (*((volatile unsigned char *) 0x4A00000C ))
#define rINTPND (*((volatile unsigned char *) 0X4A000010))
Void EINT8_23_Init()
{rEINTMASK&=~(1<<8);        //EINT8开中断
 rPRIORITY = 0x0000 007F;   // 使用默认的循环优先级
 rINTMOD = 0x0000 0000;   // 所有中断均为默认的IRQ中断
 rINTMSK&=~(1<<5);              //EINT8_23开中断
 pISR_EINT8_23= (unsigned)Eint8_23_ISR;
 rEINTPEND=1<<8;              //EINTPEND[8],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;                //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;                //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
```

S3C2440中断 14-中断示例 07-应用程序 01

```
#define rGPFCON (*((volatile unsigned char *) 0x56000050))
#define rGPFDAT (*((volatile unsigned char *) 0x56000054))
#define rGPFUP (*((volatile unsigned char *) 0x56000058))
#define rGPGCON (*((volatile unsigned char *) 0x56000060))
#define rGPGDAT (*((volatile unsigned char *) 0x56000064))
#define rGPGUP (*((volatile unsigned char *) 0x56000068))
#define pISR_EINT8_23 (*(unsigned *)(_ISR_STARTADDRESS+0x20+5*4))
#define rEXTINT1 (*((volatile unsigned char *) 0x5600008C))
#define rEINTPEND (*((volatile unsigned char *) 0x560000A8))
#define rEINTMASK (*((volatile unsigned char *) 0x560000A4))
#define rSRCPND (*((volatile unsigned char *) 0X4A000000))
#define rINTMOD (*((volatile unsigned char *) 0X4A000004))
#define rINTMSK (*((volatile unsigned char *) 0X4A000008))
#define rPRIORITY (*((volatile unsigned char *) 0x4A00000C ))
#define rINTPND (*((volatile unsigned char *) 0X4A000010))
```

S3C2440中断　14-中断示例　07-应用程序 02

```
Void PortPin_Init()
{
rGPFCON=0x5555;
rGPFUP=0x00;
rGPGCON=0x0002;
rGPGUP=0x01;
}
static void_irq Eint8_23_ISR(void)  //声明一个Eint8_23_ISR中断处理函数
{
If(rEINTPEND&(1<<8))
{//EINT8中断
  if(LED_Flag==0) LED_Flag=1;  //切换流水灯显示方向
     else              LED_Flag=0;
}//EINT8中断
 rEINTPEND=1<<8;              //EINTPEND[8],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;                //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;                //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

S3C2440中断 14-中断示例 07-应用程序 03

```
void EINT8_23_Init()
{rEINTMASK&=~(1<<8);        //EINT8开中断
 rPRIORITY = 0x0000 007F;   // 使用默认的循环优先级
 rINTMOD = 0x0000 0000;     // 所有中断均为默认的IRQ中断
 rINTMSK&=~(1<<5);          //EINT8_23开中断
 pISR_EINT8_23= (unsigned)Eint8_23_ISR;
 rEINTPEND=1<<8;            //EINTPEND[8],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;              //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;              //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

S3C2440中断　14-中断示例　07-应用程序 04

```
void Delay ( unsigned int x )                    //延时程序
{
    unsigned int i,j,k;
    for(i=0;i<=x;i++)
            for(j=0;j<=0xff;j++)
                    for(k=0;k<=0xff;k++);
}
```

S3C2440中断　14-中断示例　07-应用程序 05

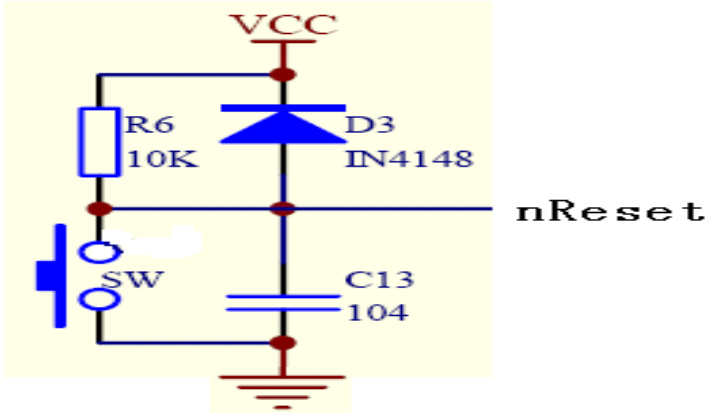
```
static LED_Flag=0;

static Int  main()

{ PortPin_Init();
  EINT8_23_Init();
  while(1)
{unsigned char LED_Encode[]={0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,0x00,0xFF};
  int index1=0;
  int index2=7;
```

S3C2440中断 14-中断示例 07-应用程序 05

```
switch(LED_Flag)
{
  case 0:// 从左向右
   {   rGPFDAT= LED_Encode[index1];  Delay(10);
       index1++;
      if(index1>=8)     index1=0;
    }  break;
  case 1: //从右向左
  {
       rGPFDAT= LED_Encode[index2];   Delay(10);
       index2--;
      if(index2<0)     index2=7;
   }  break;
}
 Delay(1);
}
Return 0;
}
```

简单应用题　01

nReset为复位引脚,说明下边电路的工作原理？



解答：

1. 电路元器件　SW:按钮;　C13 ：无极性电容　;D3：二极管　　;R6:电阻　　; nReset：S3C2440 ARM9嵌入式处理器复位引脚（输入:低电平有效）

2. 工作原理

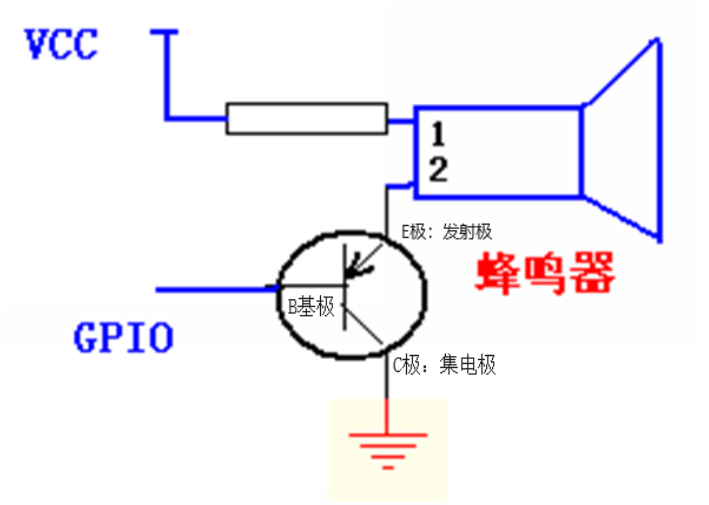
（1）上电复位（加电复位）:当给系统加电时，VCC(电源正)通过电阻R6为C13充电；利用C13的交流特性，　随着C13表面电荷增加，nReset引脚电压值从0V开始升高，直至升到VCC，这个过程中，nReset引脚电平由低电平变化到高电平；

低电平保持时间>=4个时钟周期，复位信号有效，系统复位，从头开始运行.

(2)手动复位：当点击SW开关时，按下SW，nReset引脚通过SW接地，为低电平；释放SW，　nReset引脚通过R6接Vcc,为高电平。手动按下的时间>4时钟周期，nReset有效　，系统复位，从头开始运行.

简单应用题　02

已知蜂鸣器电路图如下，说明电路的工作原理?



解答：

(1)元器件：

蜂鸣器：1引脚为输入端，2引脚为输出端，当1引脚加高电平，2引脚加低电平，蜂鸣器响，否则，不响。

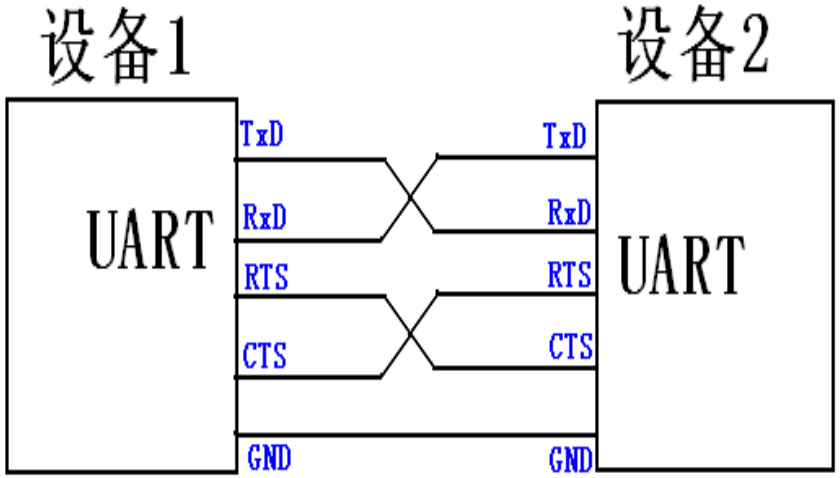
三极管：PNP型三极管, E极加高电平,B极加低电平时，EC导通。

GPIO:GPIO引脚

(2)工作原理：当GPIO输出低电平时，PNP三极管EC导通，VCC通过电阻作用与蜂鸣器1引脚，蜂鸣器2引脚通过PNP EC连到GND,则蜂鸣器响；否则，不响。

简单应用题　03

设备1与设备2 采用UART技术时接口电路如下图：



说明采用自动流方式(AFC)设备1向设备2传送数据的过程。

解答：

（1）设备2，检查RxFIFO缓冲区是否有空闲，如果无空闲，重复（1）；如果有空闲，置RTS信号有效；

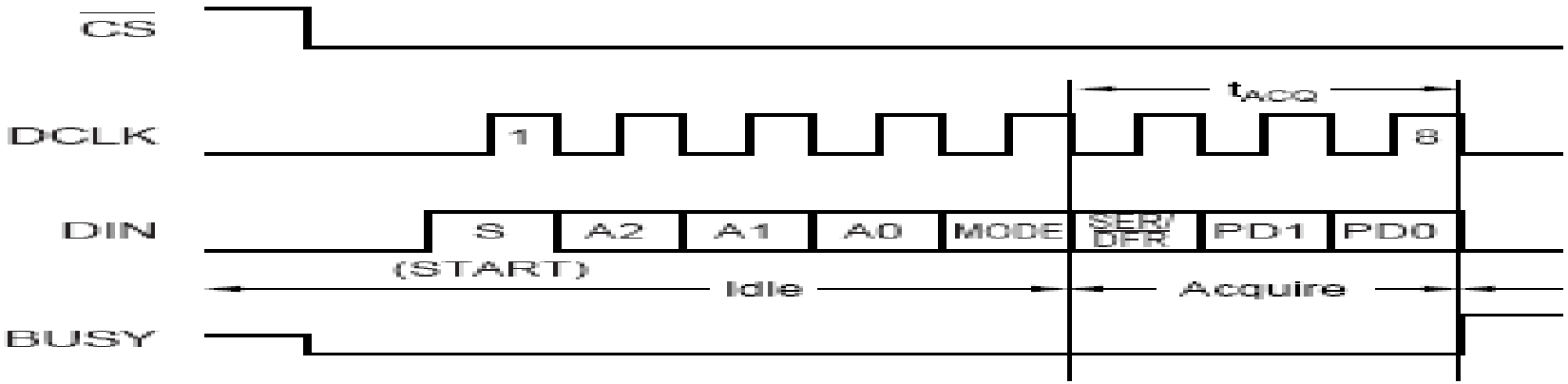
（2）设备1，设备1的CTS与设备2的RTS连接，CTS有效，触发设备1工作；

（3）设备1，检查TxFIFO缓冲区是否有数据，如果没有，重复（3），如果有，则从TxFIFO取数据，经TxD引脚发送出去。

（4）设备2，设备1 TxD与设备2 RxD连接，从RxD接收数据，同时将RTS信号设置无效；接收的数据存入RxFIFO中，转（1）。

简单应用题　04

ARM处理器向ADS7843传送读取触摸屏X(或Y坐标)命令的时序图如下，依据时序图说明命令传送过程。



解答：

（1）准备传送: CS有效，ADS7843处于工作状态；BUSY低电平，ADS7843空闲;

ARM处理器通过GPIO引脚控制DCLK低电平,准备传送数据

（2）传送一位数据：

ARM处理器通过GPIO引脚，向DIN写入一位数据;

延迟一段时间;等待DIN线上数据稳定;

ARM处理器通过GPIO引脚,控制DCLK从低电平变化到高电平，通知ADS7843可以从DIN上读取一位数据;

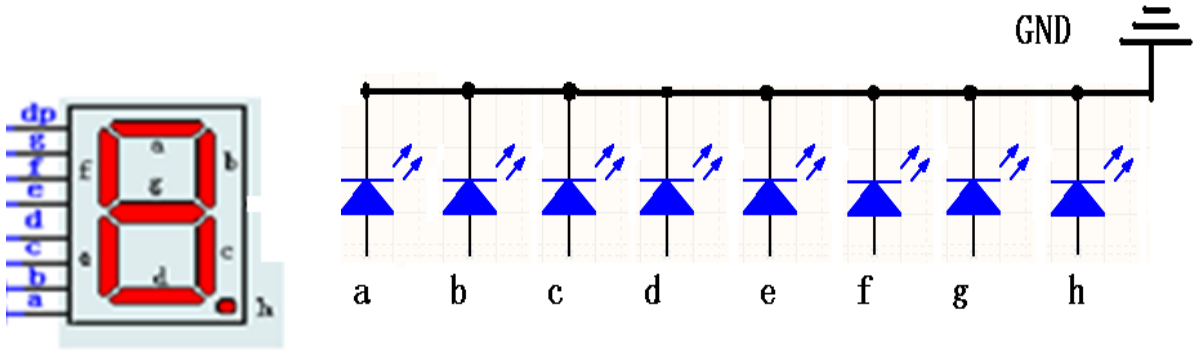
延迟一段时间，等待ADS7843读取数据;

ARM处理器通过GPIO引脚,控制DCLK从高电平变化到低电平，准备下一位传送

(3)（2）的操作一共8次,一个命令传送结束

简单应用题　05

数码管逻辑图示如下：



回答下列问题：

（1）、图中的数码管是什么类型的？

（2）、给出编码表

解答：

（1）LED发光管输出极连在一起，为共阴极数码管

（2）LED输入极加高电平(1)，LED亮；加低电平(0)，LED不亮；一般情况下，

a段连接端口GPIO引脚0

b段连接端口GPIO引脚1

c段连接端口GPIO引脚2

d段连接端口GPIO引脚3

e段连接端口GPIO引脚4

f段连接端口GPIO引脚5

g段连接端口GPIO引脚6

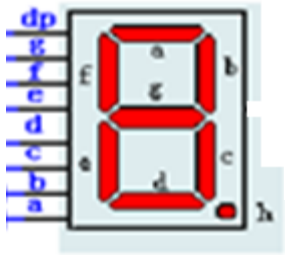
h段连接端口GPIO引脚7

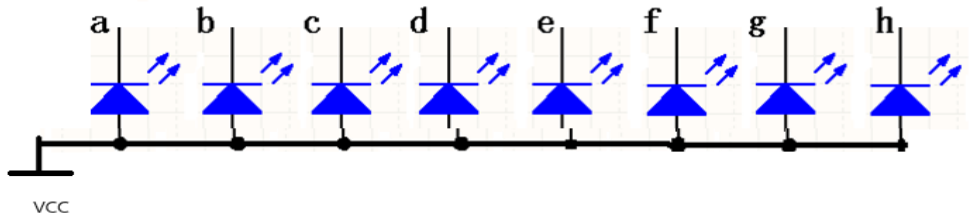
LED 数码管显示字型码表

显示字型	h	g	f	e	d	c	b	a	共阴极字形码
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F
A	0	1	1	1	0	1	1	1	0x77
b	0	1	1	1	1	1	0	0	0x7C
C	0	0	1	1	1	0	0	1	0x39
d	0	1	0	1	1	1	1	0	0x5E
E	0	1	1	1	1	0	0	1	0x79
F	0	1	1	1	0	0	0	1	0x71

简单应用题　06

数码管逻辑图示如下：





回答下列问题：

（1）、图中的数码管是什么类型的？

（2）、给出编码表

解答：

（1）LED发光管输入极连在一起，为共阳极数码管

（2）LED输出极加低电平(0)，LED亮；加高电平(1)，LED不亮；一般情况下，

a段连接端口GPIO引脚0

b段连接端口GPIO引脚1

c段连接端口GPIO引脚2

d段连接端口GPIO引脚3

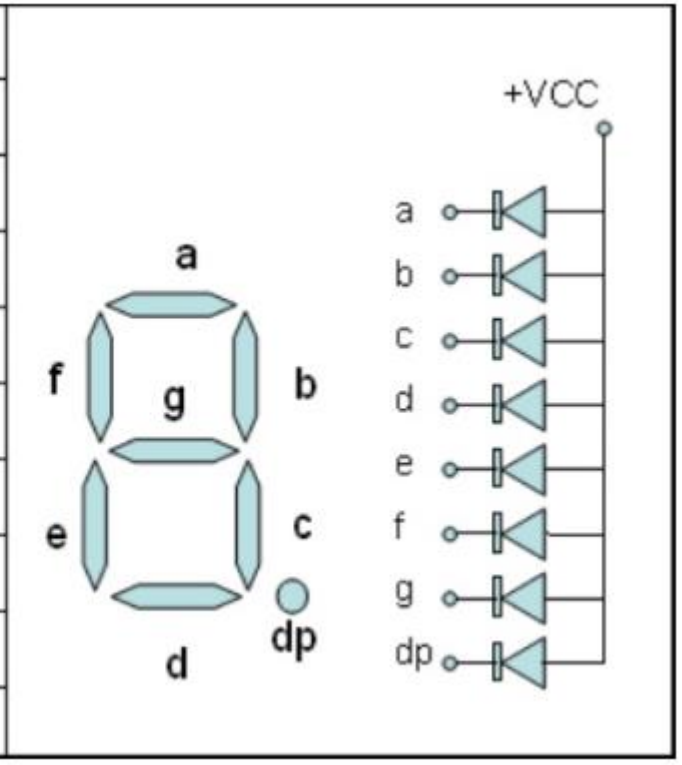
e段连接端口GPIO引脚4

f段连接端口GPIO引脚5

g段连接端口GPIO引脚6

h段连接端口GPIO引脚7

显示字型	dp ,g,f,e,d,c,b, a	字符码
0	1 1 0 0 0 0 0 0	C0H
1	1 1 1 1 1 0 0 1	F9H
2	1 0 1 0 0 1 0 0	A4H
3	1 0 1 1 0 0 0 0	B0H
4	1 0 0 1 1 0 0 1	99H
5	1 0 0 1 0 0 1 0	92H
6	1 0 0 0 0 0 1 0	82H
7	1 1 1 1 1 0 0 0	F8H
8	1 0 0 0 0 0 0 0	80H



简单应用题　07

1. 已知LCD分辨率320x240，颜色为256色，视频缓冲区的起始地址为0x40000000，回答下列问题：

（1）、说明显示数据在视频缓冲区是如何存储的？

（2）、计算点(x,y)显示数据在视频缓冲区的存储地址？

解答：

（1）256色=2的8次方，一个像素显示数据是8位，正好1个字节。

显示数据在视频缓冲区存储规则：

行：从上到下次序存储；行的序号0…239

每行：从左向右依次存储；列的序号0…319

（2）显示屏左上角第一个像素（0，0）显示数据在视频缓冲区存储地址为视频缓冲区首地址；

Y为行值，取值范围是（0..239）；X为列值，取值范围(0..319)

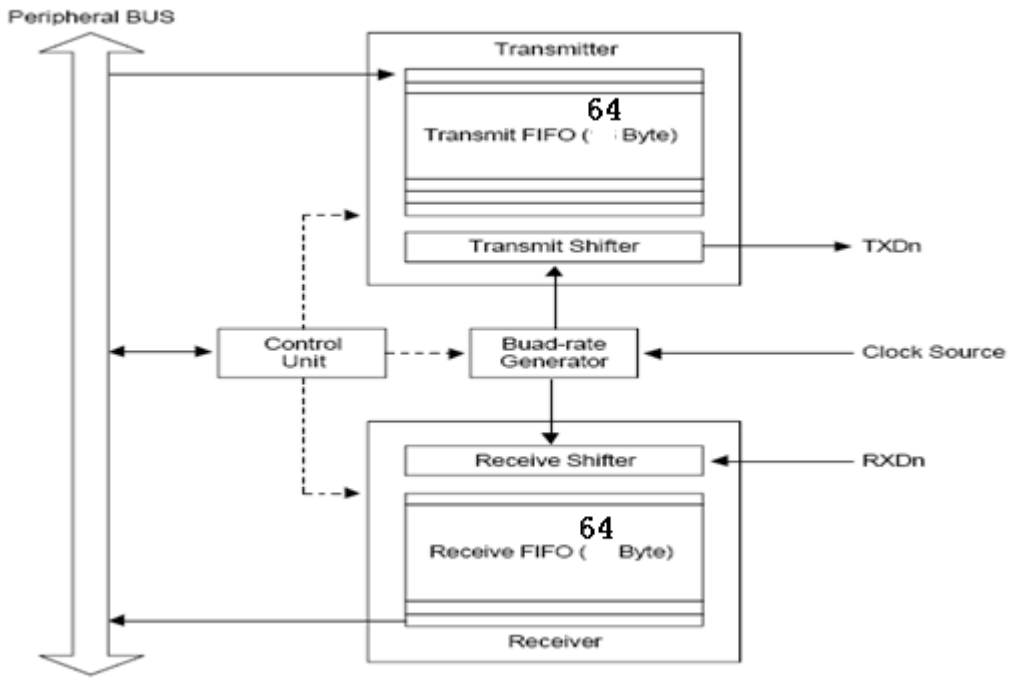
点 (x,y)的显示数据存储地址=视频缓冲区首地址+x+y*320

简单应用题　08

1.　Uart逻辑图如下，回答以下问题：

（1）、Uart有那几部分组成，都有什么功能？

（2）、阐述Uart的工作原理?



解答：

（1）组成:

发送器：将缓冲区中的数据从TxD引脚发送出去；

接收器：从RxD引脚接收数据，并存入缓冲区；

波特率发生器：产生控制发送、接收数据的速率；

控制单元：管理UART的工作模式，控制数据发送，记录接收、发送数据中错误，根据需要向

中断请求

（2）工作原理

数据发送原理：

ARM处理器将发送数据写入发送缓冲寄存器（UTXHn）；

如果使用TxFIFO缓冲区，则存入TxFIFO缓冲区，然后从TxFIFO取1字节数据，按帧格式打包，存取发送一位寄存器；

如果不适用TxFIFO缓冲区，则直接按帧格式打包，存取发送一位寄存器；

发送移位寄存器按波特率发生器规定传输速率，将数据通过TxD引脚发送出去。

接收数据原理:

接收移位寄存器按波特率发生器的速度从RxD引脚按帧格式接收一帧数据，然后从一帧数据中提取有效数据；

如果使用RxFIFO,则将数据存入RxFIFO缓冲区;否则存入接收缓冲寄存器（URXHn）

ARM处理器从接收缓冲寄存器（URXHn）读取接收数据

简单应用题　09

```
#define  rGPHCON        (*((volatile unsigned char *) 0x56000070))
         rGPHCON= (GPHCON&(~(0xF<<4)))|(0x0A<<4);
```

分析说明以上语句的功能。其中，0x56000070为H口配置寄存器地址。GPH寄存器中

引脚 GPH2　[5:4]=00 input　　引脚GPH3　[7:6]=00　input

01 output　　01　output

10 TxD0　　10　RxD0

11 保留　　11　保留

解答：

(1)定义端口H控制寄存器变量

```
#define  rGPHCON        (*((volatile unsigned char *) 0x56000070))
```

(2) ~(0xF<<4)=~(0xF0)=0x0F

GPHCON&(~(0xF<<4))= GPHCON&0x0F　　清除GPHCON的位[7:4]

(0x0A<<4)=0xA0

rGPHCON= (GPHCON&(~(0xF<<4)))|(0x0A<<4)　GPHCON的位[7:4]=0xA=1010　　配置GPH2为TxD0 GPH3为RxD0功能

简单应用题　10

K9F1208U0C读数据的时序图如下，说明其读操作过程？并编写读一页数据函数？

解答：

1.读操作过程：

（1）发送读命令。

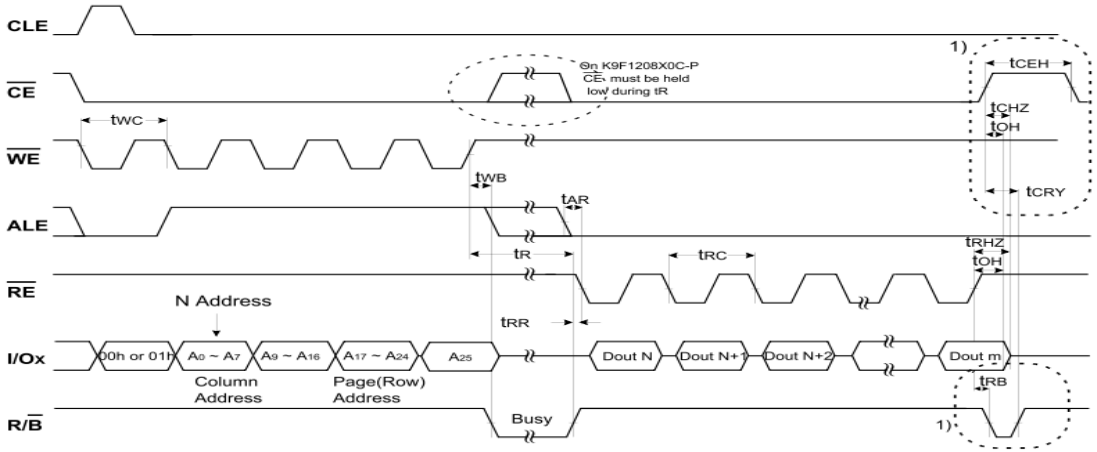
（2）发送第1个cycle地址A[7:0]。

（3）发送第2个cycle地址A[16:9]。

（4）发送第3个cycle地址A[24:17]。

（5）发送第4个cycle地址A[25]。

（6）读数据至页末。



2.读函数

```
#define  READCMD0   0
static void ReadPage (U32 addr, U8 *buf)

{     U16 i;
      rNFCONT&=~0x02;

      rNFCMMD= READCMD0;
      rNFADDR=0;
      rNFADDR=addr;
      rNFADDR=addr>>8;
      rNFADDR=addr>>16;
            InitEcc( );
     while(!(rNFSTAT&0x1));
           for(i=0; i<512; i++) buf[i] = rNFDATA;
      rNFCONT|=0x02;
}
```

简单应用题　11

K9F1208U0 擦除的时序图如下，说明其擦除操作过程？并编写擦除函数？

解答：

1.擦除操作过程：

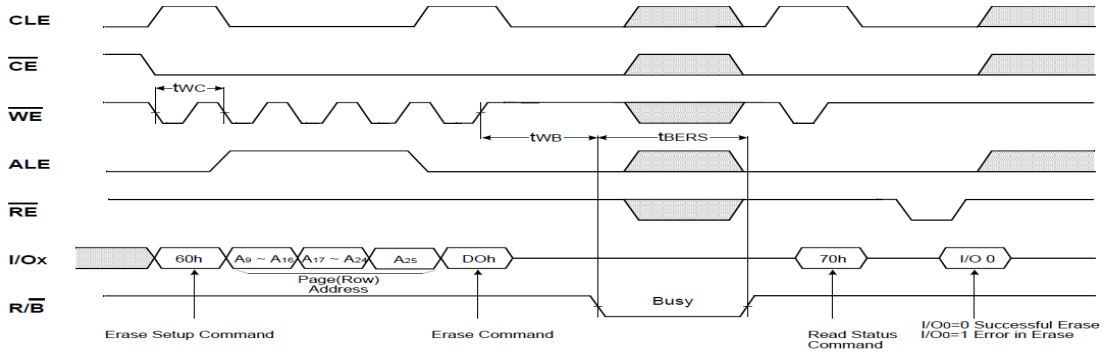
（1）发送擦除开始命令。

（2）发送第2个cycle地址A[16:9]。

（3）发送第3个cycle地址A[24:17]。

（4）发送第4个cycle地址A[25]。

（5）发送擦除结束命令。



2.**整块擦除函数**

```
static U32 EraseBlock(U32 addr)
// 整块擦除（即将整块的数据位全变为1）
{
        addr &= ~0x1f;
        rNFCONT&=~0x02;
        rNFCMMD= ERASECMD0;
    rNFADDR=addr;
        rNFADDR=addr>>8;
        rNFADDR=addr>>16;
    rNFCMMD= ERASECMD1;
        while(!(rNFSTAT&0x1));
        rNFCONT|=0x02;
}
```

简单应用题　12

K9F1208U0 写数据的时序图如下，说明其写操作过程？并编写写一页数据函数？

解答：

1.写操作过程：

（1）发送写开始指令。

（2）发送第1个cycle地址A[7:0] 。

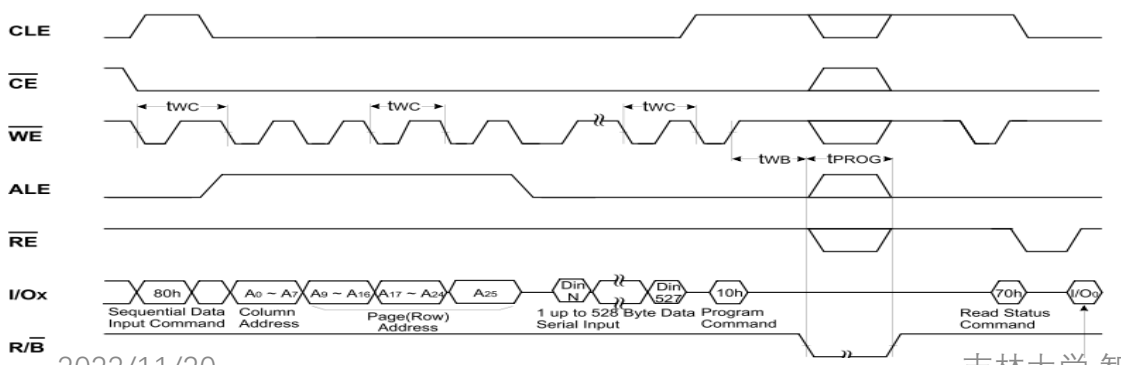
（3）发送第2个cycle地址A[16:9] 。

（4）发送第3个cycle地址A[24:17] 。

（5）发送第4个cycle地址A[25] 。

（6）写入数据至页末。

（7）发送写结束指令。



2. 写一页函数

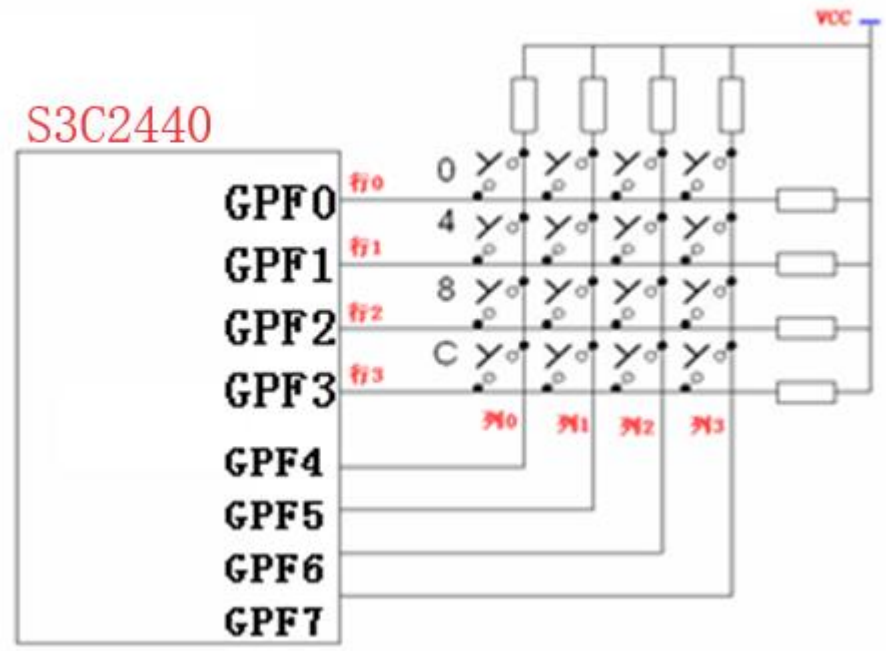
```
static U32 WritePage(U32 addr, U8 *buf)
{
    rNFCONT&=~0x02;
    rNFCMMD= PROGCMD0;
    rNFADDR=0;
    rNFADDR=addr;
    rNFADDR=addr>>8;
    rNFADDR=addr>>16;
        InitEcc( );
        while(!(rNFSTAT&0x1));
        for(i=0; i<512; i++)
      rNFDATA buf[i];
        rNFCMMD= PROGCMD1;
        rNFCONT|=0x02;
}
```

简单应用题　13-01

并行键盘接口电路如下,回答下列问题：

（1）、说明接口电路的组成以及工作原理？

（2）、编写使用扫描法识别按键的按键识别函数？



解答：

(1)组成

矩阵键盘，4行x4列，16个按钮，

行0包括0，1，2，3按钮

行1包括4，5，6，7按钮

行2包括8，9，A，B按钮

行3包括C，D，E，F按钮

每个按钮一引脚与一行线连接，另一个引脚与列线连接，当按钮按下时，对应的行线与列线线短接；

每个行线通过电阻与VCC连接；每个列线线通过电阻与VCC连接

4条行线分别与端口F的引脚GPF0、GFP1、GPF2、GPF3连接；

4条列线分别与端口F的引脚GPF4、GFP5、GPF6、GPF7连接；

简单应用题　13-02

解答：

(1)

- 扫描法按键识别工作原理

➢ 行线逐行输出低电平，

当选择一行输出低电平，读取列线值，读取的列线值不全为1，

➢ 则当前行有键按下；

➢ 根据行号与列线值查表，获取按键值。

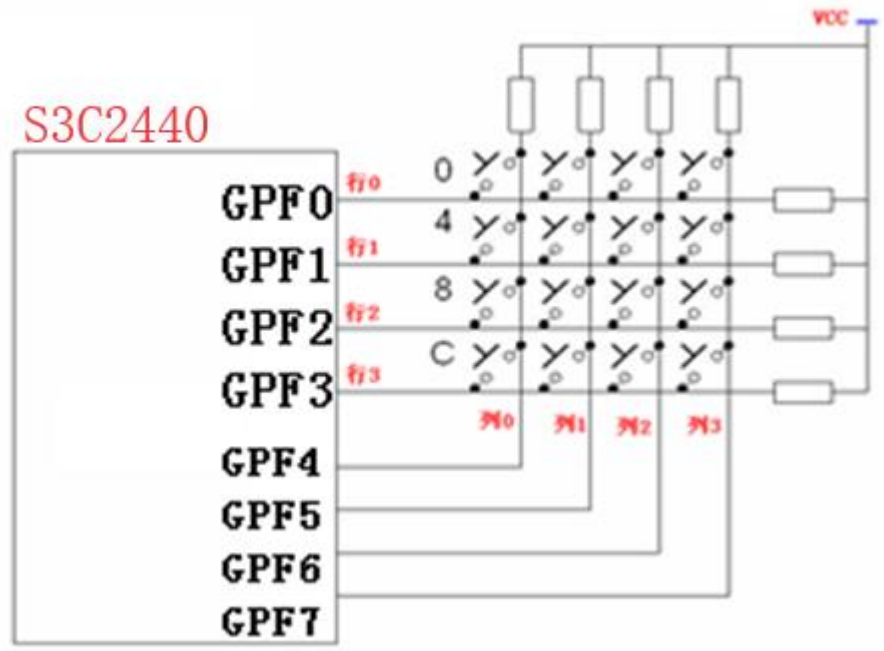
- 反转法按键识别工作原理

首先，所有行线输出低电平，读取所有列线值

然后，所有列线输出低电平，读取所有行线值

读取的行线值、列线值不全为1，则有键按下；

根据行线值、列线值查表获取按键值。



简单应用题　13-03

解答：

(2)编写使用扫描法识别按键的按键识别函数?

配置GPF[3:0]为GPIO输出

配置GPF[7:4]为GPIO输入

GPFCON=2_0000 0000 0101 0101

=0x0055

GPF[7:0]使用内部上拉电阻

GPFUP=2_0000 0000

=0x00

GPFCON	Bit	Description	
GPF7	[15:14]	00 = Input 10 = EINT[7]	01 = Output 11 = Reserved
GPF6	[13:12]	00 = Input 10 = EINT[6]	01 = Output 11 = Reserved
GPF5	[11:10]	00 = Input 10 = EINT[5]	01 = Output 11 = Reserved
GPF4	[9:8]	00 = Input 10 = EINT[4]	01 = Output 11 = Reserved
GPF3	[7:6]	00 = Input 10 = EINT[3]	01 = Output 11 = Reserved
GPF2	[5:4]	00 = Input 10 = EINT2]	01 = Output 11 = Reserved
GPF1	[3:2]	00 = Input 10 = EINT[1]	01 = Output 11 = Reserved
GPF0	[1:0]	00 = Input 10 = EINT[0]	01 = Output 11 = Reserved

GPFDAT	Bit	Description
GPF[7:0]	[7:0]	When the port is configured as an input port, the corresponding bit is the pin state. When the port is configured as an output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

GPFUP	Bit	Description
GPF[7:0]	[7:0]	0: The pull up function attached to the corresponding port pin is enabled. 1: The pull up function is disabled.

简单应用题　13-03

- 引脚初始化函数

```
#define rGPFCON (*((volatile unsigned char *) 0x56000050))
#define rGPFDAT (*((volatile unsigned char *) 0x56000054))
#define rGPFUP (*((volatile unsigned char *) 0x56000058))
void   Fport_Init()
{
rGPFCON=0x0055;
rGPFUP=0x00;
}
```

GPFCON	Bit	Description	
GPF7	[15:14]	00 = Input 10 = EINT[7]	01 = Output 11 = Reserved
GPF6	[13:12]	00 = Input 10 = EINT[6]	01 = Output 11 = Reserved
GPF5	[11:10]	00 = Input 10 = EINT[5]	01 = Output 11 = Reserved
GPF4	[9:8]	00 = Input 10 = EINT[4]	01 = Output 11 = Reserved
GPF3	[7:6]	00 = Input 10 = EINT[3]	01 = Output 11 = Reserved
GPF2	[5:4]	00 = Input 10 = EINT2]	01 = Output 11 = Reserved
GPF1	[3:2]	00 = Input 10 = EINT[1]	01 = Output 11 = Reserved
GPF0	[1:0]	00 = Input 10 = EINT[0]	01 = Output 11 = Reserved

GPFDAT	Bit	Description
GPF[7:0]	[7:0]	When the port is configured as an input port, the corresponding bit is the pin state. When the port is configured as an output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

GPFUP	Bit	Description
GPF[7:0]	[7:0]	0: The pull up function attached to the corresponding port pin is enabled. 1: The pull up function is disabled.

简单应用题　13-04

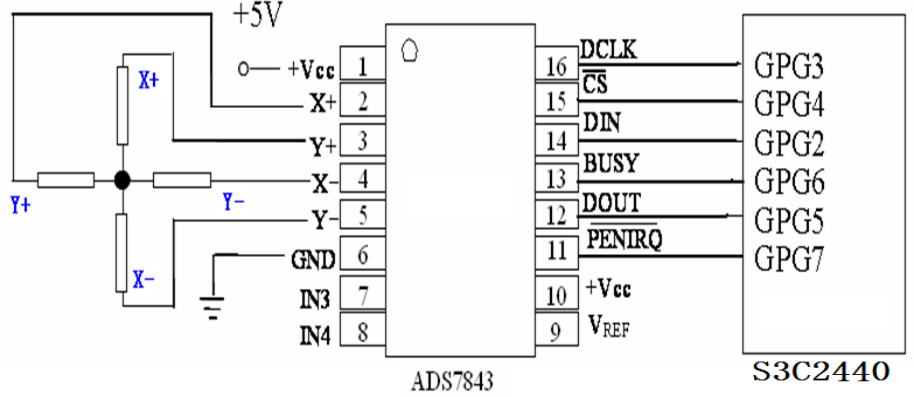
- 按键识别函数

```
char  readKeyVal(void)
{//1
  unsigned char I,j,H_Val,L_Val;
  char keyval=-1;
  rGPFDAT=0xf0;     // GPF[3:0]输出低电平
  H_Val=0xfe;
 for (i=0;i<4;i++)
{ //2
     rGPFDAT=H_Val;  //控制某一行输出低电平，其他行高电平
     for(j=0;j<100;j++);  //延时
     if(L_Val=(rPDATF&0xF0)!=0xF0)
          {   L_Val=(L_VAL>>4)|0xf0;
              keyval=get_val(H_Val)*4+get_val(L_Val);      //获取按键键值
             break;
           }
     else H_Val=(H_Val<<1)|0x01;      //下一次扫描
    }
  }//2
return keyval;
}//1
```

简单应用题　13-05

```
char get_val(unsigned char val)
{
  unsigned char i,x=0;
  for(i=0;i<4;i++)
  {
    if(~val==1)  return x;
    val=(val>>1)|0x80;
    x=x+1
   }
  return x;
}
```

简单应用题　14



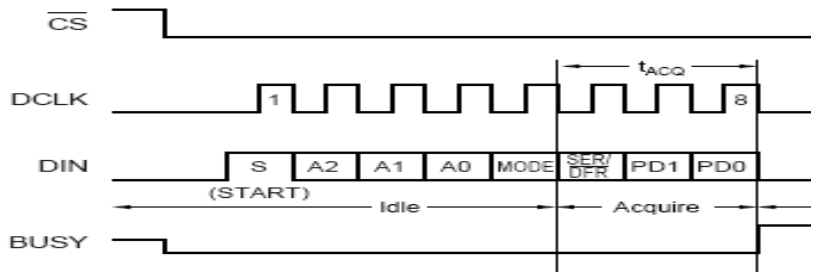
基于S3C2440 ARM处理器的触摸屏接口电路如下，回答下列问题：

1) 说明接口电路的工作原理。

2) 说明触摸屏坐标与显示器坐标以及之间的关系，说明如何将触摸屏坐标转换为显示器坐标。

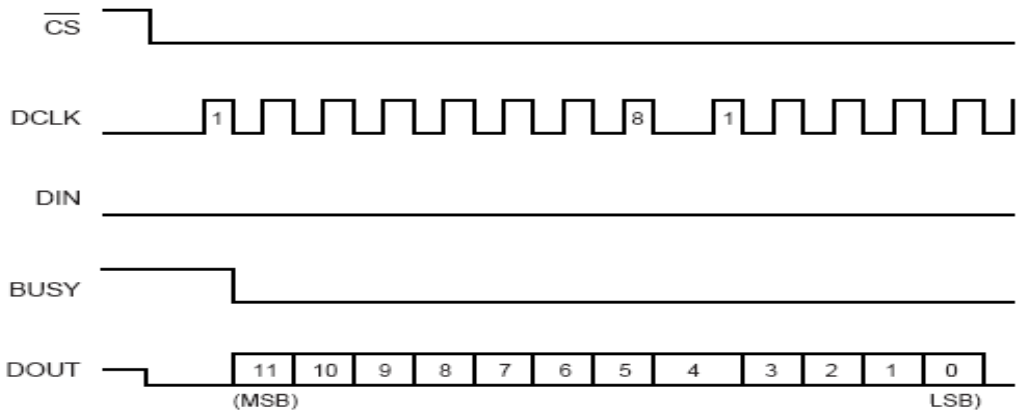
3) 给出G口初始化程序段。

4) 假设读取触摸屏X方向坐标的命令TOUCH_MSR_X为0xDC，由S3C2440向ADS7843传送数据的时序图如下，编写C函数，实现将命令TOUCH_MSR_X由S3C2440传送至ADS7843功能。



5）由ADS7843向S3C2440传送A/D转换结果的时序图如下，

编写C函数，实现S3C2440从ADS7843接收A/D转换结果



简单应用题　14-01

1.说明接口电路的工作原理。

（1）电路连接

- 触摸屏与触摸屏控制器ADS7843连接

触摸屏横向（X方向）的Y+与触摸屏控制器X+连接;

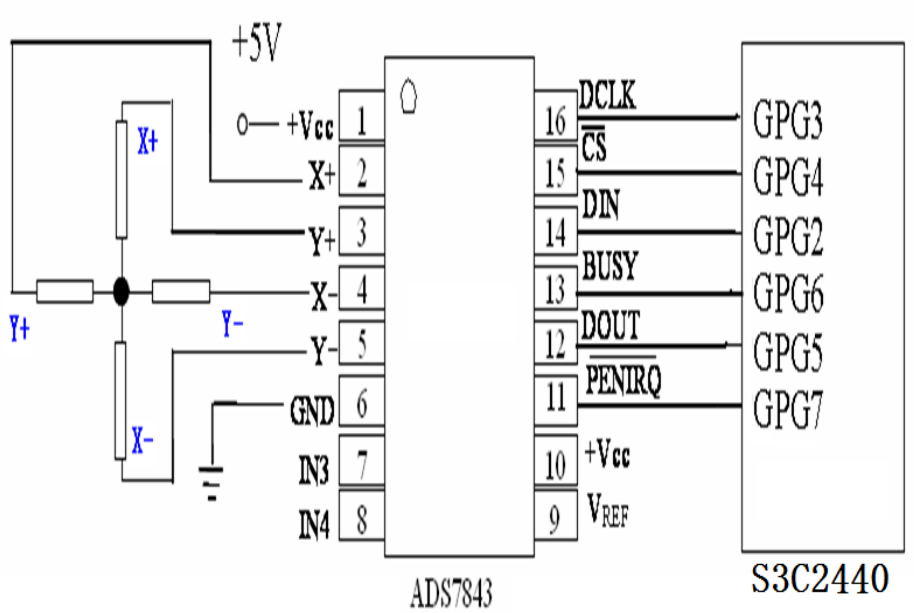
触摸屏横向（X方向）的Y-与触摸屏控制器X-连接;

采集触摸屏X方向坐标。

触摸屏纵向（Y方向）的X+与触摸屏控制器Y+连接;

触摸屏纵向（Y方向）的X-与触摸屏控制器Y-连接;

采集触摸屏Y方向坐标



- 触摸屏控制器与S3C2440连接

ADS7843 CS与S3C2440 GPG4连接，用于片选信号;

ADS7843 DCLK与S3C2440 GPG3连接，用于时钟信号;

ADS7843 DIN与S3C2440 GPG2连接，用于接收命令;

ADS7843 BUSY与S3C2440 GPG6连接，用于示忙;

ADS7843 DOUT与S3C2440 GPG5连接，用于向S3C2440传送采集的触摸屏坐标;

ADS7843 PENIRQ与S3C2440 GPG7连接，用于表示触摸屏有触摸操作;

简单应用题　14-01

（2）工作原理

S3C2440 通过GPG7获取ADS7843 PENIRQ引脚状态， PENIRQ =1,触摸屏没有触摸操作，返回;

PENIRQ =0,触摸屏有触摸操作，下一步，

S3C2440 GPG4输出低电平，控制ADS7843 片选有效,使ADS7843处于工作状态;

S3C2440 通过GPG6获取ADS7843 busy引脚状态，BUSY=1,ADS7843忙，重复执行查询操作，直至空闲;

BUSY=0,空闲，下一步，

S3C2440 通过GPG3、GPG2控制DCLK、DIN,从而向ADS7843传送命令(读取X或Y方向命令);

S3C2440 通过GPG6获取ADS7843 busy引脚状态，BUSY=1,ADS7843忙，重复执行查询操作，直至空闲;

BUSY=0,空闲，下一步，

S3C2440 通过GPG3、GPG5控制DCLK、DOUT,从ADS7843接收触摸屏坐标采集结果;

简单应用题　14-02

2.说明触摸屏坐标与显示器坐标以及之间的关系

触摸屏是一个物理定位系统，是放置在LCD显示器上与显示器高度吻合的透明坐标定位系统，通过触摸屏可为LCD显示屏定位。

触摸屏与LCD显示屏是两个设备，都有自己的坐标系统，当点击触摸屏时，有个触摸点，而在LCD显示器上总有一个点在垂直方向与之吻合，在垂直方向可认为是一个点。这样，就可以找到一种转换关系，将触摸屏坐标转换成显示器坐标。

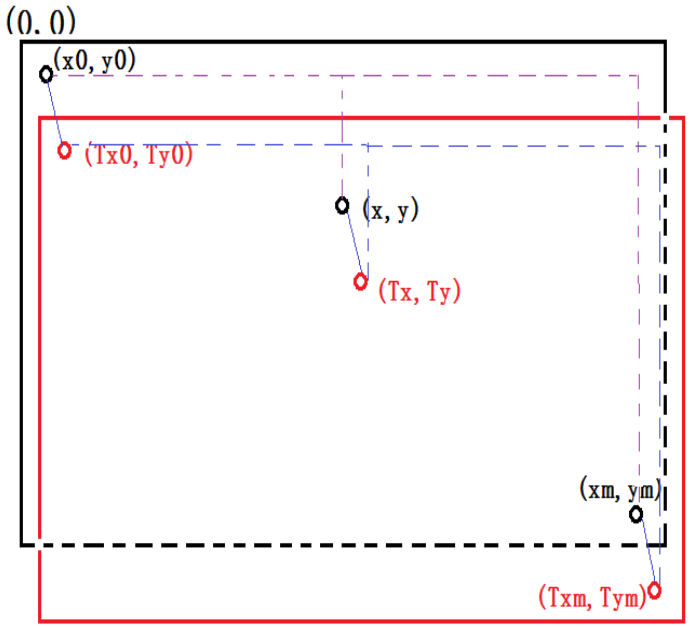
触摸屏坐标系为一个用二进制表示的坐标系；而LCD显示器是以像素来表示的坐标系,转换关系如下,

x=x0+(xm-x0)*(Tx-Tx0)/(Txm-Tx0)

y=y0+(ym-y0)*(Ty-Ty0)/(Tym-Ty0)

Tx,Ty,Txm,Tym,Tx0,Ty0为触摸屏坐标

X0,y0,xm,ym,x,y为显示器坐标



简单应用题　14-03

3.写出G端口引脚初始化程序

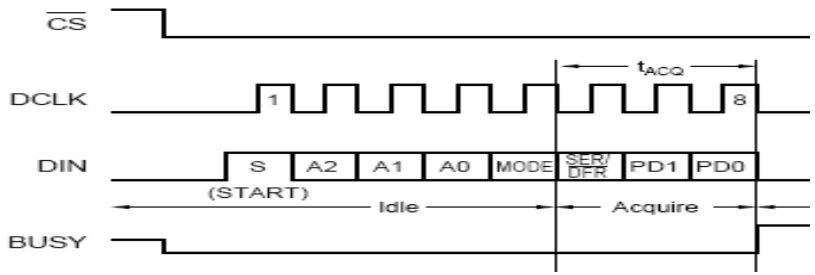
由接口电路图可知，GPG[4:2]为GPIO输出,GPG[6:4]为GPIO输入

```
rGPGCON=2_0000 0000 0000 0000 0000 0001 0101 0000
        =0x00000150
rGPGUP=2_0000 0000 0000 0000 0000
      =0x00
#define rGPGCON (*((volatile unsigned char *) 0x56000060))
#define rGPGDAT (*((volatile unsigned char *) 0x56000064))
#define rGPGUP (*((volatile unsigned char *) 0x56000068))
Void Gport_Init()
{
rGPGCON=0x00000150;
rGPGUP=0x00;
}
```

GPGCON	Bit	Description	
GPG7	[15:14]	00 = Input 10 = EINT[15]	01 = Output 11 = SPICLK1
GPG6	[13:12]	00 = Input 10 = EINT[14]	01 = Output 11 = SPIMOSI1
GPG5	[11:10]	00 = Input 10 = EINT[13]	01 = Output 11 = SPIMISO1
GPG4	[9:8]	00 = Input 10 = EINT[12]	01 = Output 11 = LCD_PWRDN
GPG3	[7:6]	00 = Input 10 = EINT[11]	01 = Output 11 = nSS1
GPG2	[5:4]	00 = Input 10 = EINT[10]	01 = Output 11 = nSS0
GPG1	[3:2]	00 = Input 10 = EINT[9]	01 = Output 11 = Reserved
GPG0	[1:0]	00 = Input 10 = EINT[8]	01 = Output 11 = Reserved

简单应用题　14-03

4.假设读取触摸屏X方向坐标的命令TOUCH_MSR_X为0xDC，由S3C2440向ADS7843传送数据的时序图如下，编写C函数，实现将命令TOUCH_MSR_X由S3C2440传送至ADS7843功能。



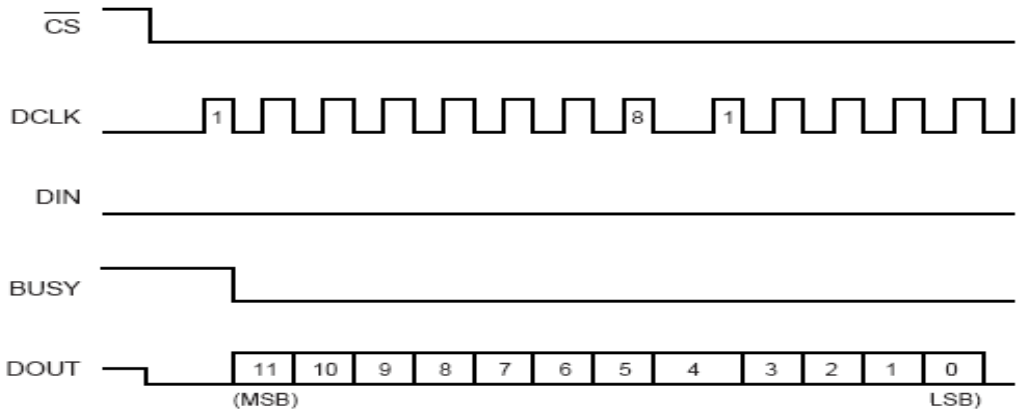
解答：

```
void  TouchSend_CMD(unsigned char CMD)
{
rGPGDAT &= Oxe7;  //CS=0;DCLK=0，GPG4
temp =0x80;      //设置要传送的位
for(i = 0; i<8; i++)   //发送命令字节
  { if(CMD & temp))  rGPGDAT| = 0x04;//DIN=1
    else   rGPGDAT & = 0xfb; //DIN=0，  GPG2
    rGPGDAT |= 0x08；    //DCLK置高上升沿，GPG3
    delay(2)；
    rGPGDAT & = 0xf7;   //清除DCLK,1位送出，GPG3
    delay(2)；
   temp =temp>>1；                    //右移1位
  }
}
```

简单应用题　14-03

5.由ADS7843向S3C2440传送A/D转换结果的时序图如下，编写C函数，实现S3C2440从ADS7843接收A/D转换结果

```
WORD TOUCHGet_Value()
{
 WORD ack=0;
rGPGDAT|=0x08;     //DCLK 置高平，GPG3=1
 delay(2);
 rGPGDAT& =0xf7; //DCLK 低电平 ，下降沿 GPG3=0
 delay(2);      //延时子程序
 for(i=0;i<8;i++)  //取得8位坐标数据
{   rGPGDAT| =0x08;    //DCLK置高，GPG3
    ack=ack<<1;
    if (rGPGDAT & 0x20)  ack+=1;
    delay(2);
    rGPGDAT& =0xf7;//DCLK 低电平 ，GPG3=0
    delay(2);
}
    rGPGDAT|=0x10;     //CS=1

}
```



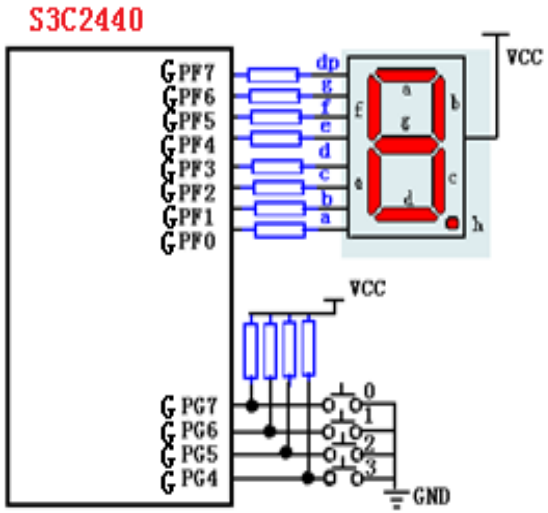
简单应用题　15

接口电路图如下,

(1)阐述接口电路工作原理。

(2)编写程序实现当按键0按下时，数码管从0到F显示一次，当按键1按下时，数码管从F到0显示一次。要求利用中断方式实现按键识别。

已知条件: VCC为电源正，GND为电源地;数码管为共阳极数码管。



简单应用题　15　01

1.阐述接口电路工作原理。

（1）数码管

数码管：共阳极数码管,可显示0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F、小数点

GPF[7:0]为控制端，在控制端加字形编码，数码管即可显示对应数字。

（2）按钮0、1、2、3

按钮一端接地，一端连接引脚GPGn (n取值4、5、6、7)

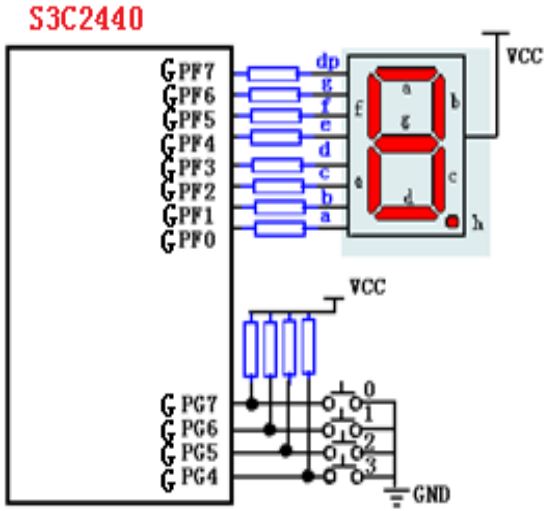
GPGn引脚通过电阻接电源正VCC.

当按钮未按下时，GPGn引脚输入高电平;

当按钮按下时，GPGn引脚接地，输入低电平;

一次点击按钮，会在GPGn引脚输入一个负脉冲信号

按钮识别方法可采用中断方式或GPIO查询方式.



简单应用题　15

2.编写程序实现当按键0按下时，数码管从0到F显示一次，当按键1按下时，数码管从F到0显示一次。要求利用中断方式实现按键识别。

已知条件: VCC为电源正，GND为电源地;数码管为共阳极数码管。

(1)F端口引脚初始化

GPF[7:0]控制数码管显示，所以为GPIO输出

rGPFCON=0x5555

rGPFUP=0x00;　//GPF【7：0】接内部上拉电阻

```
#define rGPFCON (*((volatile unsigned char *) 0x56000050))
#define rGPFDAT (*((volatile unsigned char *) 0x56000054))
#define rGPFUP (*((volatile unsigned char *) 0x56000058))
void PortF_init()
{
rGPFCON=0x5555;
rGPFUP=0x00;　//GPF【7：0】接内部上拉电阻
}
```

GPFCON	Bit	Description
GPF7	[15:14]	00 = Input 01 = Output 10 = EINT[7] 11 = Reserved
GPF6	[13:12]	00 = Input 01 = Output 10 = EINT[6] 11 = Reserved
GPF5	[11:10]	00 = Input 01 = Output 10 = EINT[5] 11 = Reserved
GPF4	[9:8]	00 = Input 01 = Output 10 = EINT[4] 11 = Reserved
GPF3	[7:6]	00 = Input 01 = Output 10 = EINT[3] 11 = Reserved
GPF2	[5:4]	00 = Input 01 = Output 10 = EINT2] 11 = Reserved
GPF1	[3:2]	00 = Input 01 = Output 10 = EINT[1] 11 = Reserved
GPF0	[1:0]	00 = Input 01 = Output 10 = EINT[0] 11 = Reserved

GPFDAT	Bit	Description
GPF[7:0]	[7:0]	When the port is configured as an input port, the corresponding bit is the pin state. When the port is configured as an output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

GPFUP	Bit	Description
GPF[7:0]	[7:0]	0: The pull up function attached to the corresponding port pin is enabled. 1: The pull up function is disabled.

简单应用题　15

(2)G端口引脚初始化

单击按钮0、1，会在GPG7\GPG6输入一个负脉冲，索引GPG7\GPG6为中断输入

rGPGCON=2_0000 0000 0000 0000 1010 1010 0000 0000

=0x0000AA00; //GPG[7:4]配置为EINT[15:12]

rGPGUP=2_1111 1111 1111 1111

=0xFFFF;　/GPG[7:4]不接内部上来电阻

```
#define rGPGCON (*((volatile unsigned char *) 0x56000060))
#define rGPGDAT (*((volatile unsigned char *) 0x56000064))
#define rGPGUP (*((volatile unsigned char *) 0x56000068))
void PortG_init()
{
rGPGCON=0x0000AA00;
rGPGUP=0xFFFF;
}
```

GPGCON	Bit	Description	
GPG15*	[31:30]	00 = Input 10 = EINT[23]	01 = Output 11 = Reserved
GPG14*	[29:28]	00 = Input 10 = EINT[22]	01 = Output 11 = Reserved
GPG13*	[27:26]	00 = Input 10 = EINT[21]	01 = Output 11 = Reserved
GPG12	[25:24]	00 = Input 10 = EINT[20]	01 = Output 11 = Reserved
GPG11	[23:22]	00 = Input 10 = EINT[19]	01 = Output 11 = TCLK[1]
GPG10	[21:20]	00 = Input 10 = EINT[18]	01 = Output 11 = nCTS1
GPG9	[19:18]	00 = Input 10 = EINT[17]	01 = Output 11 = nRTS1
GPG8	[17:16]	00 = Input 10 = EINT[16]	01 = Output 11 = Reserved
GPG7	[15:14]	00 = Input 10 = EINT[15]	01 = Output 11 = SPICLK1
GPG6	[13:12]	00 = Input 10 = EINT[14]	01 = Output 11 = SPIMOSI1
GPG5	[11:10]	00 = Input 10 = EINT[13]	01 = Output 11 = SPIMISO1
GPG4	[9:8]	00 = Input 10 = EINT[12]	01 = Output 11 = LCD_PWRDN
GPG3	[7:6]	00 = Input 10 = EINT[11]	01 = Output 11 = nSS1
GPG2	[5:4]	00 = Input 10 = EINT[10]	01 = Output 11 = nSS0
GPG1	[3:2]	00 = Input 10 = EINT[9]	01 = Output 11 = Reserved
GPG0	[1:0]	00 = Input 10 = EINT[8]	01 = Output 11 = Reserved

简单应用题　15

(3)编码表: 数码管为共阳极数码管

数码管a段连接GPF0

数码管b段连接GPF1

数码管c段连接GPF2

数码管d段连接GPF3

数码管e段连接GPF4

数码管f段连接GPF5

数码管g段连接GPF6

数码管h段连接GPF7

当将编码数据写入GPFDAT寄存器，则数码管显示对应数字

LED 数码管显示字型码表

显示字型	h	g	f	e	d	c	b	a	共阴极字形码	共阳极字形码
0	0	0	1	1	1	1	1	1	0x3F	0xC0
1	0	0	0	0	0	1	1	0	0x06	0xF9
2	0	1	0	1	1	0	1	1	0x5B	0xA4
3	0	1	0	0	1	1	1	1	0x4F	0xB0
4	0	1	1	0	0	1	1	0	0x66	0x99
5	0	1	1	0	1	1	0	1	0x6D	0x92
6	0	1	1	1	1	1	0	1	0x7D	0x82
7	0	0	0	0	0	1	1	1	0x07	0xF8
8	0	1	1	1	1	1	1	1	0x7F	0x80
9	0	1	1	0	1	1	1	1	0x6F	0x90
A	0	1	1	1	0	1	1	1	0x77	0x88
b	0	1	1	1	1	1	0	0	0x7C	0x83
C	0	0	1	1	1	0	0	1	0x39	0xC6
d	0	1	0	1	1	1	1	0	0x5E	0xA1
E	0	1	1	1	1	0	0	1	0x79	0x86
F	0	1	1	1	0	0	0	1	0x71	0x8E

```
Unsigned char
LED_Encode[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
```

简单应用题　15

(4)延时子程序

```
void Delay ( unsigned int x )                    //延时程序
{  unsigned int i,j,k;
   for(i=0;i<=x;i++)
          for(j=0;j<=0xff;j++)
                 for(k=0;k<=0xff;k++);
}
```

(5)EINT8_23向量地址变量

中断向量表首地址=_ISR_STARTADDRESS+0x20　　//EINT0的中断向量地址

EINT8_23中断向量地址=中断向量表首地址+5*4

```
#define pISR_EINT8_23 (*(unsigned *)(_ISR_STARTADDRESS+0x20+5*4))
```

INTOFFSET	偏移量	INTOFFSET	偏移量	INTOFFSET	偏移量	INTOFFSET	偏移量
INT_ADC	31	INT_UART1	23	INT_UART2	15	nBATT_FLT	7
INT_RTC	30	INT_SPI0	22	INT_TIMER4	14	INT_CAM	6
INT_SPI1	29	INT_SDI	21	INT_TIMER3	13	EINT8_23	5
INT_UART0	28	INT_DMA3	20	INT_TIMER2	12	EINT4_7	4
INT_IIC	27	INT_DMA2	19	INT_TIMER1	11	EINT3	3
INT_USBH	26	INT_DMA1	18	INT_TIMER0	10	EINT2	2
INT_USBD	25	INT_DMA0	17	INT_WDT_AC97	9	EINT1	1
INT_NFCON	24	INT_LCD	16	INT_TICK	8	EINT0	0

简单应用题　15

(5)中断服务程序

EINT8_23中断服务程序

EINT8为二级中断，归模块EINT8_23,在EINT_23中断服务程序内判断EINTPEND[8]位来确定EINT8是否申请中断。

全局变量 LED_Flag=0时，数码管显示0到F

LED_Flag=1时，数码管显示F到0

每单击0、1按钮，执行一次中断服务程序，中断程序切换LED_Flag的值.

```
static void_irq Eint8_23_ISR(void)              //声明一个Eint8_23_ISR中断处理函数
{
If(rEINTPEND&(1<<15))
{LED_Flag=0; }//EINT15中断
If(rEINTPEND&(1<<14))
{LED_Flag=1;}//EINT14中断
 rEINTPEND=1<<12;              //EINTPEND[12],写入1，清除EINT12中断请求记录
 rEINTPEND=1<<13;              //EINTPEND[13],写入1，清除EINT13中断请求记录
 rEINTPEND=1<<14;              //EINTPEND[14],写入1，清除EINT14中断请求记录
 rEINTPEND=1<<15;              //EINTPEND[15],写入1，清除EINT15中断请求记录
 rSRCPND=1<<5;                //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;                //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

简单应用题　15

(6)中断初始化

```
#define pISR_EINT8_23 (*(unsigned *)(_ISR_STARTADDRESS+0x20+5*4))
#define rEXTINT1 (*((volatile unsigned char *) 0x5600008C))
#define rEINTPEND (*((volatile unsigned char *) 0x560000A8))
#define rEINTMASK (*((volatile unsigned char *) 0x560000A4))
#define rSRCPND (*((volatile unsigned char *) 0X4A000000))
#define rINTMOD (*((volatile unsigned char *) 0X4A000004))
#define rINTMSK (*((volatile unsigned char *) 0X4A000008))
#define rPRIORITY (*((volatile unsigned char *) 0x4A00000C ))
#define rINTPND (*((volatile unsigned char *) 0X4A000010))
```

简单应用题　15

```
Void EINT8_23_Init()
{rEINTMASK&=~(1<<15);        //EINT15开中断
 rEINTMASK&=~(1<<14);        //EINT14开中断
 rPRIORITY = 0x0000 007F;    // 使用默认的循环优先级
 rINTMOD = 0x0000 0000;      // 所有中断均为默认的IRQ中断
 rINTMSK&=~(1<<5);           //EINT8_23开中断
 pISR_EINT8_23= (unsigned)Eint8_23_ISR;
 rEINTPEND=1<<15;            //EINTPEND[15],写入1，清除EINT8中断请求记录
 rEINTPEND=1<<14;            //EINTPEND[14],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;               //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;               //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

简单应用题　15

```
Void EINT8_23_Init()
{rEINTMASK&=~(1<<15);        //EINT15开中断
 rEINTMASK&=~(1<<14);        //EINT14开中断
 rPRIORITY = 0x0000 007F;    // 使用默认的循环优先级
 rINTMOD = 0x0000 0000;      // 所有中断均为默认的IRQ中断
 rINTMSK&=~(1<<5);           //EINT8_23开中断
 pISR_EINT8_23= (unsigned)Eint8_23_ISR;
 rEINTPEND=1<<15;            //EINTPEND[15],写入1，清除EINT8中断请求记录
 rEINTPEND=1<<14;            //EINTPEND[14],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;               //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;               //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

简单应用题　15

(7)程序

```
#define rGPFCON (*((volatile unsigned char *) 0x56000050))
#define rGPFDAT (*((volatile unsigned char *) 0x56000054))
#define rGPFUP (*((volatile unsigned char *) 0x56000058))
#define rGPGCON (*((volatile unsigned char *) 0x56000060))
#define rGPGDAT (*((volatile unsigned char *) 0x56000064))
#define rGPGUP (*((volatile unsigned char *) 0x56000068))
#define pISR_EINT8_23 (*(unsigned *)(_ISR_STARTADDRESS+0x20+5*4))
#define rEXTINT1 (*((volatile unsigned char *) 0x5600008C))
#define rEINTPEND (*((volatile unsigned char *) 0x560000A8))
#define rEINTMASK (*((volatile unsigned char *) 0x560000A4))
#define rSRCPND (*((volatile unsigned char *) 0X4A000000))
#define rINTMOD (*((volatile unsigned char *) 0X4A000004))
#define rINTMSK (*((volatile unsigned char *) 0X4A000008))
#define rPRIORITY (*((volatile unsigned char *) 0x4A00000C ))
#define rINTPND (*((volatile unsigned char *) 0X4A000010))
```

简单应用题　15

```
void PortF_init()
{
rGPFCON=0x5555;
rGPFUP=0x00;   //GPF【7：0】接内部上拉电阻
}

void PortG_init()
{
rGPGCON=0x0000AA00;
rGPGUP=0xFFFF;
}
```

简单应用题　15

```
static void_irq Eint8_23_ISR(void)    //声明一个Eint8_23_ISR中断处理函数
{
If(rEINTPEND&(1<<15))
{LED_Flag=0; }//EINT15中断
If(rEINTPEND&(1<<14))
{LED_Flag=1;}//EINT14中断
 rEINTPEND=1<<12;                  //EINTPEND[12],写入1，清除EINT12中断请求记录
 rEINTPEND=1<<13;                  //EINTPEND[13],写入1，清除EINT13中断请求记录
 rEINTPEND=1<<14;                  //EINTPEND[14],写入1，清除EINT14中断请求记录
 rEINTPEND=1<<15;                  //EINTPEND[15],写入1，清除EINT15中断请求记录
 rSRCPND=1<<5;                    //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;                   //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

简单应用题　15

```
Void EINT8_23_Init()
{rEINTMASK&=~(1<<15);        //EINT15开中断
 rEINTMASK&=~(1<<14);        //EINT14开中断
 rPRIORITY = 0x0000 007F;    // 使用默认的循环优先级
 rINTMOD = 0x0000 0000;      // 所有中断均为默认的IRQ中断
 rINTMSK&=~(1<<5);           //EINT8_23开中断
 pISR_EINT8_23= (unsigned)Eint8_23_ISR;
 rEINTPEND=1<<15;            //EINTPEND[15],写入1，清除EINT8中断请求记录
 rEINTPEND=1<<14;            //EINTPEND[14],写入1，清除EINT8中断请求记录
 rSRCPND=1<<5;               //SRCPND[5]=EINT8_23,写入1，清除EINT8_23中断请求记录
 rINTPND=1<<5;               //INTPND[5]=EINT8_23,写入1，清除EINT8_23中断服务记录
}
```

简单应用题　15

```
void Delay ( unsigned int x )                    //延时程序
{  unsigned int i,j,k;
   for(i=0;i<=x;i++)
          for(j=0;j<=0xff;j++)
                for(k=0;k<=0xff;k++);
}
```

简单应用题　15

```
static LED_Flag=0;
static Int  main()
{ PortPin_Init();
  EINT8_23_Init();
  while(1)
 {unsigned char
    LED_Encode[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
  int index1=0;
  int index2=0xF;
```

简单应用题　15

```
switch(LED_Flag)
{
  case 0:// 从0向F
   {   rGPFDAT= LED_Encode[index1];  Delay(10);
       index1++;
      if(index1>=15)     index1=0;
    }  break;
  case 1: //从F向0
  {
       rGPFDAT= LED_Encode[index2];   Delay(10);
       index2--;
      if(index2<0)     index2=15;
   }  break;
}
 Delay(1);
}
Return 0;
}
```