

چکیده :

یکی از متدائل ترین روش های نمایش اطلاعات استفاده از LCDها می باشد. LCDها نوعی از نمایشگر ها هستند که از کریستال مایع برای نمایش اطلاعات بهره می جویند . LCDها انواع مختلفی دارند ازجمله کاراکتری ، گرافیکی که هر کدام نیز به دسته های مختلفی دسته بندی می شوند : نمایشگر های مونوکروم ، رنگی و
شیوه ی کار هر کدام از این ال سی دی ها متفاوت بوده و در اینجا فرصتی برای توضیح آن ها نیست.

در این مقاله به توضیح نمایشگر (ELT240320ATP) معروف به N96 LCD می پردازیم . این ال سی دی دارای یکی از کنترل های ili9320 یا ili9325 می باشد که در اینجا به توضیح کنترل ili9325 می پردازیم (البته ناگفته نماند که کد های راه اندازی هر دو آن ها مشابه هم می باشد) در ابتدا به توضیح مختصری درباره اصطلاحاتی که در این مقاله بیشتر با آن ها سروکار داریم و درک مفهوم آن ها حائز اهمیت است میپردازیم.

پیکسل (Pixel) :

پیکسل برگرفته از عبارت Picture element المان های کوچک تک رنگی هستند که با در کنار هم قرار گرفتن آن ها تصویر ساخته میشود. هر چه تعداد پیکسل ها در یک تصویر بیشتر باشد نشان کیفیت بالاتر آن تصویر یا نمایشگر است البته در مورد نمایشگر ها پارامترهای دیگری چون ppi (تعداد پیکسل ها در هر اینچ) نیز مهم اند مثلا ال سی دی مونوکرومیک گرافیکی 128×128 را در نظر بگیرید 128 * 64 نشان میدهد که این ال سی دی 64 سطر و 128 ستون و لذا $128 \times 64 = 8192$ پیکسل دارد .

کنترل :

در داخل ال سی دی ها یک یا دو کنترل (IC) وجود دارد که تمام کارهای ال سی دی اعم از تنظیمات تغذیه ، مقدار دهی پیکسل ها و ... را انجام میدهند .

نحوه نمایش تصویر بر روی ال سی دی های گرافیکی :

در ال سی دی های گرافیکی یک RAM داخلی وجود دارد که اطلاعات مربوط به پیکسل ها در این RAM ذخیره میشود سپس کنترل ال سی دی این اطلاعات را از RAM خوانده و متناسب با آن پیکسل ها را مقدار دهی میکند بعد از این کار تصویر به نمایش در می آید.

نمایشگر : ELT240320ATP

این ال سی دی دارای 320*240 پیکسل می باشد و در دو مد 16 بیتی و 8 بیتی کار می کند این نوع ال سی دی دارای یکی از کنترل های ili9320 یا ili9325 است .(البته کنترل های دیگری نیز وجود دارد .)

Micro SD :

به دلیل پایین بودن حافظه ی میکرو، چه حافظه flash و چه حافظه SRAM برای نمایش تصاویر با کیفیت نیاز به یک حافظه ی خارجی داریم. گزینه های مختلفی برای انتخاب حافظه بیرونی وجود دارد یکی از بهترین انتخاب ها استفاده از حافظه SD است چرا که این حافظه ها در حجم های بالا و با قیمت های بسیار مقرون بصرفه در بازار موجود هستند.
در ادامه درباره راه اندازی این حافظه بحث میکنیم.

پایه های ال سی دی :

پایه های تغذیه : این ال سی دی دارای 5 پایه تغذیه میباشد که پایه های 5 و 34 زمین هستند و پایه های 6932 و 6933 میباشند که باید به ولتاژ تغذیه 3.3 ولت وصل شوند (رنج ولتاژ مجاز در datasheet ال سی دی آورده شده است)

پایه های (DB10 :: 17 :: 8) و (DB1 :: 17 :: 8) :

این پایه ها باس دیتا هستند که کد دستورات و داده ها به ال سی دی ارسال می شوند و یا داده ها از ال سی دی به میکرو فرستاده میشوند.

پایه CS :

پایه انتخاب تراشه یا ChipSelect می باشد که در هنگام خواندن یا نوشتمن دیتا در ال سی دی این پایه باید زمین گردد.

پایه RS :

این پایه وظیفه انتخاب رجیستر دستورالعمل یا رجیستر داده که اگر این پایه 0 باشد رجیستر دستورالعمل و اگر 1 باشد رجیستر داده انتخاب می شود.

پایه WR :

توسط این پایه میتوانیم دستورات یا داده ها را در ال سی دی بنویسیم که این پایه در حالت 0 منطقی فعال می باشد.

پایه RD :

اگر این پایه در سطح منطقی 0 باشد یعنی قصد خواندن داده را داریم.

پایه RST :

اگر این پایه در سطح منطقی 0 باشد ال سی دی را ریست میکند.

پایه IM0 :

توسط این پایه میتوانیم مد کاری ال سی دی را معین بکنیم که اگر 0 باشد مد 16 بیتی و اگر 1 باشد مد 8 بیتی انتخاب می گردد.

توجه بشود که در مد 8بیتی پایه های 17 :: DB10 باس دیتا را تشکیل می دهند.

پایه های X+, X-, Y+، Y- :

پایه های مربوط به تاج اسکرین مقاومتی هستند.

پایه های LED_A, LEDk1::4 :

پایه های مربوط به بک لایت که LED_A پایه آند و بقیه کاتد هستند.

چگونگی تنظیم ال سی دی N96 :

برای تنظیم هر قسمت از ال سی دی باید ابتدا کد دستور یا آدرس رجیستر مربوط به آن را ارسال کنیم و سپس داده را در آن قرار دهیم .

این اطلاعات از مقاله "راه اندازی ال سی دی ili9325" نوشته : گروه M.M.S چاپ شده در ماهنامه تخصصی برق و الکترونیک ECA (نویز) شماره 4 چاپ آذر 1389 و "دیتا شیت ال سی دی" گرفته شده است برای توضیحات کامل درباره قسمت های مختلف ال سی دی می توانید به دو منبع مذکور مراجعه بفرمایید.

در این پژوهه از یک نمونه کتابخانه راه اندازی ili9325 نوشته شده توسط علی ایمانی فر از دانشگاه شاهroud که در اینترنت پخش شده استفاده شده البته نوشتن کتابخانه برای این ال سی دی آنچنان دشوار نیست.

چند نمونه از توابع بسیار مهم این کتابخانه را شرح می دهیم و بقیه آن ها واضح است و نیاز به توضیح ندارد.

همچنان که توضیح دادیم برای تنظیم هر قسمت در ال سی دی باید ابتدا کد دستور را فرستاده و سپس دیتا را در رجیستر مربوط قرار میدهیم.

روال زیر کد دستور مورد نظر را به میکرو ارسال می کند :

```
//-----  
// send a word data to the lcd  
//-----  
void lcd_write_index_register(char ins)  
{  
    CS_LOW;  
    RS_LOW;  
    LCD_DATAPORT_MSB_PORT=0x00;  
    LCD_DATAPORT_LSB_PORT=ins;  
    WR_LOW;  
    WR_HIGH;  
    CS_HIGH;  
}
```

همچنانکه مشخص است برای ارسال کد دستور ابتدا پایه ی CS را فعال (0) و پایه ی RS را 0 کرده تا رجیستر دستور انتخاب شود سپس دستور را در بس قرار داده وبعد پین WR را یک لحظه فعال و بعد غیر فعال میکنیم تا دستور در رجیستر قرار گیرد.

قدم بعدی نوشتن دیتا در رجیستر داده است که روال زیر اینکار را انجام می دهد:

```
//-----  
// write a word data into the WDR LCD register  
//-----  
void lcd_write_wdr(int data)  
{  
    CS_LOW;  
    RS_HIGH;  
    LCD_DATAPORT_MSB_PORT=(data>>8);  
    LCD_DATAPORT_LSB_PORT=data;  
    WR_LOW;  
    WR_HIGH;  
    CS_HIGH;  
}
```

این روال شبیه به روال قبل بوده با این تفاوت که در اینجا پایه RS را 1 میکنیم تا رجیستر داده را انتخاب کند.

همچنین میتوان داده را از ال سی دی خواند که این کار را میتوان با تابع زیر انجام داد .

```
//-----  
// read data from rdr lcd register  
//-----  
int lcd_read_rdr(void)  
{  
    int data_lsb,data_msb,rdr_data;  
    LCD_DATAPORT_CLEAR;  
    LCD_DATAPORT_INPUT;  
    RD_LOW;  
    WR_HIGH;  
    CS_LOW;
```

```

data_lsb=LCD_DATAPORT_LSB_PIN;
data_msb=LCD_DATAPORT_MSB_PIN;
rdr_data=((data_msb<<8)|data_lsb);
CS_HIGH;
RD_HIGH;
WR_HIGH;
LCD_DATAPORT_OUTPUT;
return rdr_data;
}

```

راه اندازی MMC به زبان C در کدویژن

برای آشنایی کامل با مموری SD و تفاوت های آن با دیگر حافظه ها و ساختار درونی آن ها میتوانید به مقاله سفید نوشته شده توسط آقای بسطام بیرامی که در پیوست ضمیمه شده مراجعه کنید و در اینجا تنها به بسط و توضیح در رابطه با توابع و کتابخانه های استفاده شده در این پروژه میپردازیم .

در این پروژه از یک ماژول مموری کارت چینی که شماتیک مداری آن در پیوست ارائه شده استفاده شده است.



همچنانکه میدانید MMC از پروتکل SPI برای انتقال اطلاعات استفاده میکند لذا با اتصال صحیح ماژول به میکرو میتوان آن را با رعایت ضوابط این اینترفیس راه اندازی کرد.

نحوه خواندن و نوشتن:

در کل برای راه اندازی MMC چندین کتابخانه وجود دارد که بهترین آن ها دو کتابخانه ff.h و pff.h میباشند که pff.h جدیدتر بوده و تفاوت این کتابخانه با ff.h در اشغال کردن حافظه های flash و SRAM می باشد که حافظه SRAM در استفاده از این کتابخانه بیشتر مدنظر خواهد بود.

کتابخانه مذکور توسط یک استاد ژاپنی تهیه و بصورت آزاد بر روی اینترنت قرار داده شده است.

فرمت های قابل پشتیبانی کتابخانه شامل fat12/fat16/fat32 میباشد و با توجه به ساپورت fat32 امکان ساپورت کارت های با مقدار فضای بیشتر از 4 گیگ هم امکان پذیر خواهد بود.(برای آشنایی نسبی با این فرمت ها میتوانید به پیوست مراجعه کنید) در این قسمت قبل از هر چیز توضیحاتی مختصر در مورد هدر های ارائه شده در کنار این کتابخانه میپردازیم :

هدر : integer

با استفاده از این هدر نام های جدیدی به انواع داده ها نسبت داده شده یعنی شما بعد از این هرجا UINT را دیدید با مراجعه به این هدر متوجه خواهید شد که منظور Unsigned int میباشد .

هدر : diskio

در این هدر توابع پایه ای کار با کتابخانه fat گنجانده شده است .

هدر : MMC

کتابخانه SPI از pff.h ساخت افزاری استفاده می کند. این هدر شامل تنظیمات SPI و پین های مورد استفاده و همچنین توابع پایه ای کار با MMC میباشد .

هدر : pff

اصلی ترین هدر مورد استفاده ی ما این هدر می باشد و تمامی توابع کار با fat در این هدر گنجانده شده است و یک نکته حائز اهمیت دیگر اینکه در pff.h تنظیماتی برای فعال یا غیر فعال کردن برخی توابع وجود دارد.

برای فعال کردن شناسایی fat32

#define _F_FAT32

1

و حال معرفی توابع :

اولین تابع:

اینیشیالایز کردن کارت حافظه :

شکل کلی تابع :

```
DSTATUS disk_initialize (void);
```

اگر 0 را برگرداند یعنی کارت حافظه با موفقیت شناسایی شده و در غیر اینصورت عملیات ناموفق بوده .

تابع دوم :

باز کردن درایو کارت حافظه :

شکل کلی تابع :

```
FRESULT pf_mount (FATFS*);
```

برای بازکردن یک درایو در fat برای ثبت یا خواندن اطلاعات می باشد.

در ff.h یک شماره برای تابع به عنوان شماره ی درایو ارسال می شود ولی چون pff.h فقط برای یک درایو است لذا بصورت پیش فرض آن آرگومان حذف شده است.

ورودی تابع یک اشاره گر (fs(file system object) به متغیری که از قبل باید به عنوان ساختمان تعریف شده باشد می باشد یعنی باید از قبل متغیری را به این شکل تعریف کنید :

```
FATFS Fs;
```

خروجی های تابع بصورت زیر خواهند بود :

- FR_OK
- FR_NOT_READY
- FR_DISK_ERR
- FR_NO_FILESYSTEM

قبل از ادامه به نمونه کد زیر دقت نمایید :
توجه : در اینجا فرض شده است که کتابخانه ها و هدر ها در پوشه SD_DRIVE ضمیمه پروژه گردیده اند .
از یک نمایشگر کاراکتری برای نمایش اطلاعات استفاده شده است .

```
#include <mega32.h>
#include <alcd.h>
#include <delay.h>
#include "SD_DRIVE\mmc.c"
#include "SD_DRIVE\pff.c"
#define XTAL 8000000
//declare Global variables
#define BUFFER_SIZE 16
FATFS fs;
WORD w_br;
void main (void ){
char buffer[BUFFER_SIZE];
lcd_init(16);
lcd_gotoxy(1,0);
lcd_putsf("Read MicroSD");
lcd_gotoxy(1,1);
lcd_putsf("Mohammad Dehghani");
delay_ms(2000);
lcd_clear();
lcd_putsf("Init Drive");
while(disk_initialize() != FR_OK) delay_ms(100);
lcd_putsf(" --->OK");
delay_ms(1000);
lcd_clear();
lcd_putsf("OPEN DRIVE");
while(pf_mount(&Fs)!=FR_OK) delay_ms(100);
lcd_putsf(" --->OK");
```

```

delay_ms(1000);

lcd_clear();

lcd_putsf("Open File");

while(pf_open("White.bmp")!=FR_OK) delay_ms(100);

lcd_putsf(" --->OK");

delay_ms(1000);

lcd_clear();

lcd_putsf("Read File");

while(pf_read(&buffer,30,&w_br)!=FR_OK) delay_ms(100);

lcd_putsf(" --->OK");

delay_ms(1000);

lcd_clear();

lcd_putsf("Close Drive");

while(pf_mount(0)!=FR_OK ) delay_ms(100);

lcd_putsf(" --->OK");

delay_ms(1000);

lcd_clear();

lcd_puts(buffer);

delay_ms(1000);

while(1);

```

درادامه لازم است تا انواع خروجی توابع را بشناسیم.

در هدر کد زیر نوشته شده : dikio.h

```

typedef enum {

    RES_OK = 0,          /* 0: Function succeeded */

    RES_ERROR,           /* 1: Disk error */

    RES_NOTRDY,          /* 2: Not ready */

    RES_PARERR           /* 3: Invalid parameter */

} DRESULT;

```

این کد یک enum است که به هر عدد یک نام را نسبت داده است.

در فایل pff.h نیز کدی دیگر بصورت زیر وجود دارد:

```
typedef enum {  
    FR_OK = 0,          /* 0 */  
    FR_DISK_ERR,        /* 1 */  
    FR_NOT_READY,       /* 2 */  
    FR_NO_FILE,         /* 3 */  
    FR_NO_PATH,         /* 4 */  
    FR_NOT_OPENED,      /* 5 */  
    FR_NOT_ENABLED,     /* 6 */  
    FR_NO_FILESYSTEM    /* 7 */  
} FRESULT;
```

برای برنامه نویسی MMC در کدویزن ابتدا باید پایه ها را مشخص کنید مثلا :

```
#define SCK_DDR          DDRB
```

```
#define SCK_PRT          PORTB
```

```
#define SCK_BIT           7
```

```
#define MOSI_DDR          DDRB
```

```
#define MOSI_PRT          PORTB
```

```
#define MOSI_BIT           5
```

```
#define MISO_DDR          DDRB
```

```
#define MISO_PRT          PORTB
```

```
#define MISO_BIT           6
```

```
#define SD_CS_DDR          DDRB
```

```
#define SD_CS_PRT          PORTB
```

```
#define SD_CS_BIT           4
```

هنگامی که تابع disk_initialize را فراخوان میکنیم تنظیمات مربوط به SPI انجام می شود.

پس برای شروع کار با MMC باید دو تابع زیر فراخوانی شوند :

```
DSTATUS disk_initialize (void);  
HRESULT pf_mount (FATFS*);
```

حال در ادامه به سراغ تابع pf_open می رویم :

```
FRESULT pf_open (const char*); //open the file
```

این تابع یک ورودی و یک خروجی از نوع FRESULT (همان که در enum بیان شده بود) دارد.
ورودی تابع نام فایل مورد نظر است مثلا "1.bmp" .

تابع : pf_read

```
FRESULT pf_read (  
    Void* buff,  
    WORD btr,  
    WORD* br  
)
```

این تابع سه ورودی و یک خروجی دارد. اولین ورودی بافری است که دیتا در آن قرار می گیرد, دومی تعداد بایتی است که که خوانده میشود و سومین ورودی تعداد بایتی است که سیستم توانسته بخواند.

تابع بعدی pf_write می باشد :

این تابع نیز همچون تابع خواندن سه ورودی و یک خروجی FRESULT دارد :

```
FRESULT pf_write (  
    const void* buff;  
    WORD btw,  
    WORD* bw  
)
```

اولین ورودی بافری است که میخواهیم در فابل مورد نظر بنویسیم, دومی طول بافر و آخرین پارامتر تعداد بایت های نوشته شده است.

برای یادگیری کامل نحوه نوشتن در MMC به پیوست مراجعه کنید.

می توان نقطه‌ی شروع خواندن و نوشتن را توسط تابع زیر مشخص نمود :

```
pf_lseek (ofs);
```

ضمناً توجه شود که باید این تابع را در فایل pff.h فعال کنید :

```
#define _USE_LSEEK     1
```

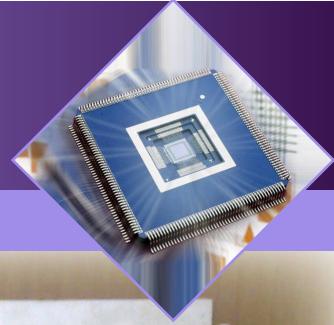
چگونگی نمایش تصویر :

ضمنا لازم است تا توضیحی در رابطه با نحوه نمایش تصویر بر روی ال سی دی بدهم.

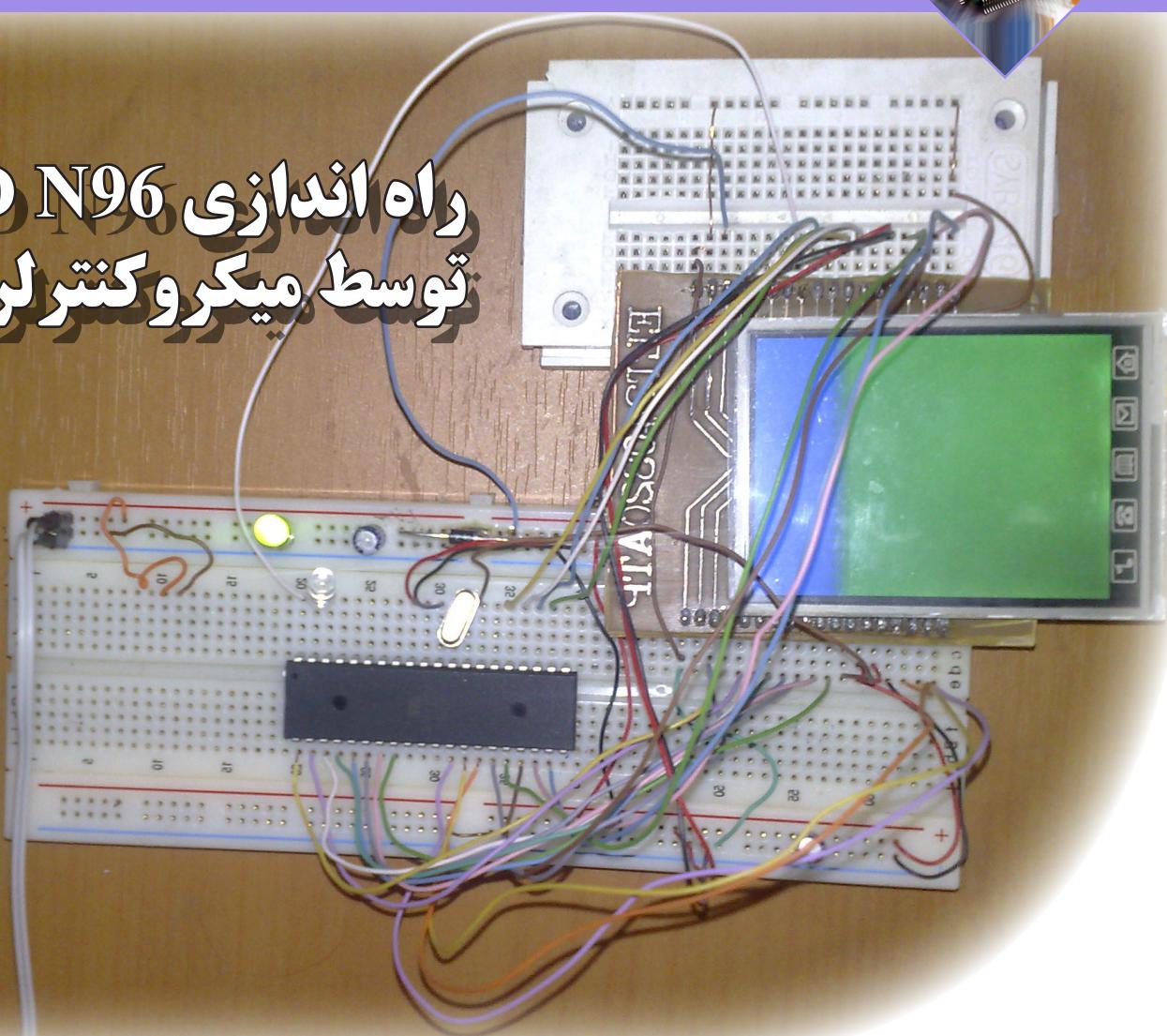
تصاویر با فرمت 24 bmp 24 بیتی در SD CARD ذخیره شده اند و در نتیجه اطلاعات مربوط به تصویر از جمله رنگ پیکسل ها بصورت 24 بیتی در فایل نوشته شده اند و از آنجاکه دیتاباس ما و همچنین GRAM موجود در ال سی دی ساختار 16 بیتی دارند لازم است تا پس از خواندن اطلاعات موجود در فایل آن ها را به فرم 16 بیتی تبدیل کنیم که اینکار در تابع showbmp(void) لازم است فرم 888 در خود ذخیره می کند و می دانیم که ال سی دی فرمت رنگ 565 را ساپورت می کند در نتیجه با انجام چند ماسک و شیفت میتوانیم تبدیل فرمت را انجام دهیم.(در پیوست فرمت 24 بیتی توضیح داده شده است)

از آنجا که RAM داخل میکرو حجمی ندارد مجبوریم اطلاعات تصویر را خط به خط خوانده و در یک آرایه 720x720 بربیزیم. شاید پرسید 720 از کجا آمده است میدانیم که هر خط ال سی دی (در حالت پورتره) دارای 240 پیکسل است و از آنجا که هر پیکسل رنگی است و هر رنگ از ترکیب سه رنگ اصلی RGB تشکیل شده لذا در فایل عکس برای هر پیکسل سه عدد موجود است که میزان قرمزی ، سبزی و آبی بودن پیکسل را مشخص می کند و $720 \times 240 = 3 \times 240^2$ عدد باید خوانده شود تا رنگ کامل یک پیکسل مشخص شود.

پیوست



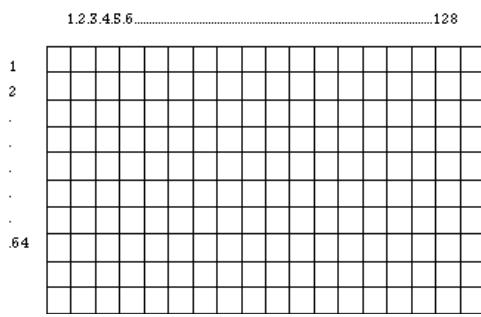
راه اندازی LCD N96 توسط میکروکنترلر AVR



یعنی هر سطر از ۱۲۸ پیکسل تشکیل شده است پس در کل این LCD $128 \times 64 = 8192$ پیکسل دارد و برای نمایش تصویر این پیکسل ها باید مقدار دهی شوند.

هر چه تعداد پیکسل ها در یک مساحت ثابت و مشخص بیشتر باشد، تصویر کیفیت بهتری خواهد داشت.

دو شکل ۲-الف و ۲-ب را در نظر می گیریم مساحت هر دوی آن ها با هم برابر است (هر دوی آن ها ۳*۲ هستند) ولی تعداد پیکسل های شکل (ب) بیشتر از شکل (الف) است در نتیجه فاصله پیکسل ها از هم کم است و پیکسل ها نزدیک به هم هستند در نتیجه تصویری



شکل ۱ : به هر یک از این خانه ها یک پیکسل می گویند

یکی از متداول ترین روش های نمایش اطلاعات استفاده از LCD ها می باشد. LCD ها معمولا در سه نوع کارکتری، گرافیکی و رنگی یافت می شوند. از LCD های کارکتری برای نمایش حروف و اعداد و از LCD های گرافیکی برای نمایش تصاویر گرافیکی سیاه و سفید استفاده می شود. اما برای نمایش تصاویر رنگی باوضوح و کیفیت بالاتر از LCD رنگی استفاده می کنند.

در این مقاله به توضیح LCD N96 چینی (ELT240320ATP) می پردازیم که این LCD دارای یکی از کنترلرهای ILI9325 یا ILI9320 می باشد که در اینجا به توضیح کنترلر ILI9325 می پردازیم (البته ناگفته نماند که کدهای راه اندازی هر دوی آنها مشابه هم می باشد).

در ابتدا به توضیح مختصراً درباره ای اصطلاحاتی که در این مقاله بیشتر با آن ها سرو کار داریم و درک مفهوم آن ها حائز اهمیت است می پردازیم .

پیکسل : LCD های گرافیکی چه از نوع رنگی و یا سیاه و سفید تصویر را به کمک پیکسل ها به نمایش در می آورند مثلاً گرافیکی سیاه و سفید 128×64 را در نظر می گیریم. نشان می دهد که این LCD ۶۴ سطر دارد و در هر سطر 128 ستون

راه اندازی LCD N96 توسط میکروکنترلر AVR

عملکرد پایه های LCD

پایه های تغذیه : این LCD دارای ۵ پایه تغذیه می باشد که پایه های ۵ و ۳۴ GND هستند و پایه های VCC ۶،۳۲،۳۳ هستند که باید به ولتاژ تغذیه ۳،۳ وصل شوند (رنج ولتاژ مجاز در DATA آورده شده است).

پایه های (DB10-DB17) و (DB1-DB8) : این پایه ها DATA BUS (گذرگاه های داده) هستند یعنی توسط این پایه ها کد دستورات و داده ها به LCD ارسال می شوند و میکرو توسط این خطوط داده ها را از LCD می خوانند.

پایه CS : پایه انتخاب تراشه (CHIP SELECT) است که باید در هنگام خواندن و یا نوشت LCD در DATA این پایه فعال باشد (CS=1) حالت فعال CS=0 و حالت غیر فعال (CS=1)

پایه RS : این پایه وظیفه انتخاب رجیستر دستور العمل یا رجیستر داده را بر عهده دارد که اگر این پایه در سطح پایین (0 منطقی) باشد رجیستر دستور العمل انتخاب می شود و اگر در سطح بالا (1 منطقی) باشد رجیستر داده انتخاب می شود.

پایه WR : توسط این پایه می توانیم دستورات یا داده ها را در LCD بنویسیم یعنی اگر این پایه در سطح پایین باشد قصد نوشت دستور در رجیستر دستور العمل یا دیتا در رجیستر داده را داریم.

پایه RD : اگر این پایه در سطح منطقی صفر باشد یعنی قصد خواندن داده را از LCD داریم.

پایه RST : اگر این پایه در سطح منطقی صفر باشد LCD را باز نشانی می کند.

پایه IM0 : توسط این پایه می توانیم مد کاری LCD را تعیین کنیم که اگر این پایه در سطح منطقی پایین باشد (IM0=0) مد ۱۶ بیتی انتخاب می شود و اگر این پایه در سطح منطقی بالا باشد (IM0=1) مد ۸ بیتی انتخاب می شود توجه کنید که اگر مد ۸ بیتی انتخاب شود (IM0=1) در این صورت پایه های (DB10-DB17) را از LCD می بینیم که این گذرگاه های داده مورد استفاده قرار می گیرد. همچنین بر روی فلت LCD دو تا جای مقاومت وجود دارد که اگر مد ۸ بیتی انتخاب شود باید مقاومت لحیم شده در R2 را در بیاوردیم و به R1 لحیم کنیم.

پایه های X+Y-X-Y : مربوط به صفحه لمسی LCD می باشند. **پایه های LEDK1-LEDK4 و LED-A :** این پایه ها مربوط به بک لایت LCD می باشد که LED-A آند دیود های نورانی می باشد و بقیه (LEDK1-LEDK-4) کاتد های دیود های نورانی می باشد.

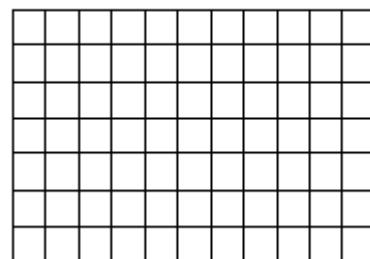
نحوه خواندن و نوشت

در این جا لازم است با مفهوم دو کلمه (دستور و داده) آشنا شویم. **دستور :** برای تنظیم قسمت های مختلف LCD (تنظیمات قسمت های LCD، پانل POWER، تنظیمات شفافیت تصویر و...) ابتدا باید کد هر یک از این قسمت هارا به LCD ارسال کنیم تا کنترلر LCD متوجه شود که ما قصد مقدار دهی بیت های کدام قسمت را داریم. که به این کد ها کد دستوری می گوییم. مثلا برای تنظیم قسمت ENTRY MODE ابتدا باید کد (03H) را ارسال کنیم با ارسال این کد، کنترلر LCD متوجه می شود که ما می خواهیم بیت های

که در شکل (ب) به نمایش در می آید نسبت به شکل (الف) کیفیت بهتری خواهد داشت.

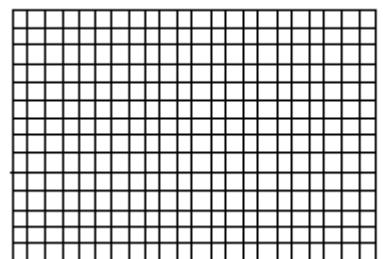
یکی از تفاوت های LCD های رنگی با سیاه و سفید این است که LCD های رنگی تعداد پیکسل بیشتری نسبت به LCD های سیاه و سفید دارند و این باعث می شود که کیفیت تصویر در LCD های رنگی بیشتر از LCD های گرافیکی سیاه و سفید باشد. به علاوه پیکسل های LCD های رنگی توانایی نمایش رنگ های مختلف را دارند ولی پیکسل های LCD های سیاه و سفید از این توانایی برخوردار نیستند.

3 CM



شکل ۲ - الف

3 CM



شکل ۲ - ب

کنترلر : در داخل LCD های گرافیکی ۱ یا ۲ کنترلر (آی سی) وجود دارد که تمام کار های LCD اعم از تنظیمات تغذیه، مقداردهی پیکسل ها و... را انجام می دهد.

نحوه نمایش تصویر بر روی LCD های گرافیکی : در LCD های گرافیکی یک RAM داخلی وجود دارد که اطلاعات مربوط به پیکسل ها در این RAM ذخیره می شوند سپس کنترلر این LCD اطلاعات را از RAM خوانده و متناسب با آن پیکسل ها را مقداردهی می کند بعد از مقدار دهی پیکسل ها تصویر یا چیز دیگری که قرار است روی LCD نشان داده شود به نمایش در می آید.

درباره LCD N96 چینی

این LCD دارای ۲۰۰*۲۰۰ پیکسل می باشد که در دو مد ۱۶ بیتی و ۸ بیتی کار می کند. این نوع LCD دارای یکی از کنترلرهای ILI9320 یا ILI9325 است. البته کنترلرهای دیگری نیز وجود دارند ولی در این مقاله بحث ما فقط در مورد دو کنترلر ذکر شده خواهد بود. این LCD قابلیت اتصال تاچ اسکرین را نیز دارد.

راه اندازی LCD N96 میکروکنترلر AVR

8.2.6. Entry Mode (R03h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	TRI	DFM	0	BGR	0	0	HWM	0	ORG	0	I/D1	I/D0	AM	0	0	0

جدول ۱

مقدار دهی می شود و بایت کم ارزش با مقدار کد دستور مقدار دهی می شود.

برای نوشتندگی داده در رجیستر داده (WRITE TO CONTROL REGISTER OR THE INTERNAL GRAM BY WDR) ابتدا CS را فعال می کنیم (CS=0) بعد RD و RS را SET می کنیم (RS=1, RD=1) سپس مقدار داده را در پایه های DB17 تا DB1 قرار می دهیم و پایه wr را reset می کنیم (wr=0) تا داده ها وارد رجیستر مربوطه شوند سپس CS را set (wr=1, CS=1) برای خواندن داده از lcd طبق جدول (5) می کنیم (wr=1, CS=1). برای خواندن داده از rdr reg (read from the internal gram by rdr reg) ابتدا RS و wr را می کنیم و طبق معمول CS را نیز reset می کنیم بعد هر زمان که بخواهیم داده ها را از رجیستر مربوطه بخوانیم reset RD را RD می کنیم (RD=0) می کنیم در این صورت اطلاعات خوانده شده و در روی پورت های مربوطه قرار می گیرد و سپس بعد از خواندن داده ها پایه i RD را SET می کنیم.

تنظیمات راه اندازی اولیه LCD

برای اینکه LCD بتواند به درستی کار کند نیاز به تنظیماتی از قبیل power, تعیین محدودیت LCD, تعیین مساحت پنجره GRAM و ... است که قبل از استفاده از LCD باید این ها مقدار دهی شوند البته این LCD تنظیمات زیادی دارد که توضیح همه این ها از بحث ما خارج است و ما فقط آن قسمت های را که مهم هستند توضیح خواهیم داد. و بقیه قسمت ها را می توان با استفاده از date sheet مقدار دهی نمود (البته همه این قسمت ها در برنامه بطور خلاصه توضیح داده خواهد شد).

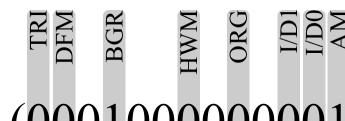
1-۱ start oscillation: طبق جدول کد دستور این قسمت (00h) است که ابتدا باید این کد را به LCD ارسال کنیم و سپس با توجه به جدول داده i (0001h) را به LCD ارسال کنیم.

1-۲ driver output control: در این قسمت باید دو بیت SS و مقدار دهی شوند. بیت SS تا s720 را تعیین sm

این قسمت را مقدار دهی کنیم که به (03H) کد دستور ENTRY MODE می گوییم.

ENTRY MODE در مثال بالا بعد از ارسال کد دستور MODE نوبت به آن می رسد که بیت های آن قسمت (TRI, DFM, BGR, HWM, ORG, I/D1, I/D0, AM) را مقدار دهی می کنیم که به مقدار این بیت ها که توسط میکرو ارسال می شود داده می گوییم. به عنوان مثال این بیت ها را این طوری مقدار دهی می کنیم که به مقدار این بیت ها داده می گوییم (Data=0001000000001000). بعد از کد دستورها به تمامی اطلاعات (کد رنگ ها و تصویر...) نیز داده گفته می شود. بعد از آشنایی با مفهوم داده و دستور به نحوه نوشتندگی در LCD می پردازیم.

در مرد ۱۶ بیتی طبق جدول ۳ برای نوشتندگی در رجیستر دستور العمل (WRITE AN INDEX TO IR REGISTER) ابتدا باید پایه i CS را فعال کنیم (CS=0) بعد طبق جدول پایه i RS را وارد کنیم (RS=1, RD=0) سپس SET RD را وارد کنیم (RD=1, RS=0) و پایه i RESET را وارد کنیم.



(0001000000001000)

شکل ۳

کد دستور را در پایه DB17 تا DB1 قرار می دهیم و سپس WR را RESET می کنیم (WR=0) با WR کردن مقدار WR موجود در پایه های DB17 تا DB1 وارد رجیستر دستور العمل (IR REGISTER) می شود بعد از وارد شدن مقدار در رجیستر (CS=1, WR=1) دستور العمل پایه های CS و WR را SET می کنیم (CS=1, WR=1). توجه: چون تمامی کدهای دستور مقداری یک باشند بقیه پایه ها کد دستور در پایه های DB8 تا DB1 با قرار گیرند و بقیه پایه های DB17 تا DB9 با مقدار صفر مقدار دهی شوند (کد دستور DB1=00H=DB9 و DB8=00H=DB17). یعنی بایت پر ارزش با صفر

8.2.3. Start Oscillation (R00h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	
R	1	1	0	0	1	0	0	1	1	0	0	1	0	0	1	0	1

The device code "9325h" is read out when read this register.

جدول ۲

Registers selection by system interface (8-/9-/16-/18-bit bus width)

I80

Function

Function	RS	nWR	nRD
Write an index to IR register	0	0	1
Read an internal status	0	1	0
Write to control registers or the internal GRAM by WDR register.	1	0	1
Read from the internal GRAM by RDR register.	1	1	0

جدول ۳

راه اندازی LCD N96 توسط میکروکنترلر AVR

8.2.6. Entry Mode (R03h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	TRI	DFM	0	BGR	0	0	HWM	0	ORG	0	I/D1	I/D0	AM	0	0	0

جدول ۵

بیت AM: نحوه استفاده از LCD به صورت افقی یا عمودی را مشخص می کند. اگر AM=0 باشد LCD بصورت افقی و اگر AM=1 باشد LCD بصورت عمودی مورد استفاده قرار می گیرد. بیت های I/D/I/O: مقدار این بیت ها نحوه UPDATE پیکسل ها در GRAM به صورت افزایش یا کاهشی را مشخص می کند. ORG: زمانی که مقدار این بیت صفر باشد مبدأ آدرس شروع (00000H) را در GRAM حرکت می دهد. حالت غیر فعال این بیت صفر است (ORG=0).

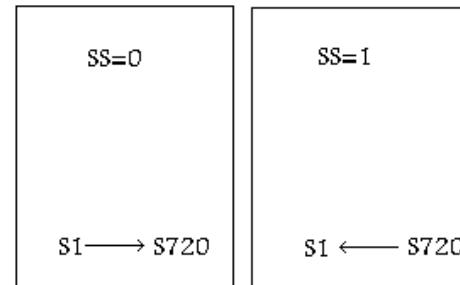
بیت BGR: اگر مقدار این بیت یک باشد (BGR=1) در این صورت جای R و B عوض می شود (RGB) و در حالت غیر فعال مقدار این بیت صفر است (BGR=0). در صورتی که این بیت غیر فعال باشد در آن صورت تغییر در جاهای (RGB) ایجاد نمی شود و در حالت اصلی (RGB) باقی می ماند.

بیت TRI: در حالت غیر فعال مقدار این بیت صفر است اگر مقدار این بیت یک شود (TRI=1) در آن صورت بیت DFM مد انتقال داده را تعیین می کند.

RESIZING CONTROL REGISTER: ۱-۴ کد دستور این قسمت (04H) می باشد که می بایستی ابتدا در رجیستر دستور العمل نوشته شود.

بیت های RSZ0 و RSZ1: مقدار این بیت ها طبق جدول زیر ضریب کوچک شدن تصویر اصلی را نشان می دهد. که از این جدول

می کند. اگر SS=0 باشد جهت شیفت از S1 به S720 می باشد و اگر SS=1 باشد جهت شیفت از S720 به S1 می باشد. (جدول ۶) SCAN CONTROL : عملکرد این بیت را در جدول قسمت SM توضیح خواهیم داد.

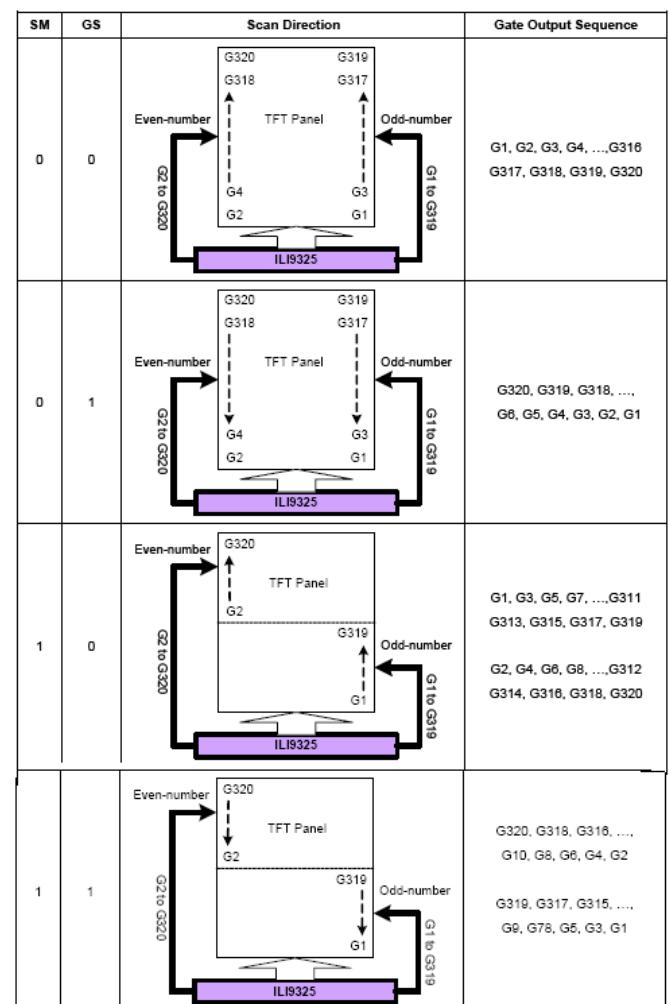


8.2.4. Driver Output Control (R01h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	0	0	0	0	SM	0	SS	0	0	0	0	0	0	0	0

جدول ۶

ENTRY MODE ۱-۳ : طبق جدول ۶ کد دستور این قسمت (03H) می باشد ابتدا باید این کد را در رجیستر دستور العمل بنویسیم و بعد بیت های این قسمت را مقدار دهی کرده و در رجیستر داده می نویسیم.



جدول ۶

LCD رنگی با تاچ (نمایشگر معروف به LCD گوشی N96 چینی)

LCD گوشی N96 چینی با ELT240320 دارای ۷*۵ سانتی متری (با اندازه تصویر ۲۰*۳۲ پیکسل) می باشد که توسط آن میتوانید تصاویر و متون رنگی خود را به نمایش در آورید.

این lcd دارای درایور داخلی به شماره i li9325 می باشد که میتواند از طریق ۱۶ خط داده با انواع میکرو کنترلرهای ۸ و ۱۶ بتی (arm avr یا AVR) ارتباط برقرار کرده و اطلاعات مربوط به تصاویر را از آنها دریافت کند.

این LCD به همراه تاچ اسکرین مخصوص خودش ارائه می شود.

لینک محصول :

<http://eshop.eea.ir/link/511.php>

راه اندازی LCD N96 میکروکنترلر AVR

	I/D[1:0] = 00 Horizontal : decrement Vertical : decrement	I/D[1:0] = 01 Horizontal : increment Vertical : decrement	I/D[1:0] = 10 Horizontal : decrement Vertical : increment	I/D[1:0] = 11 Horizontal : increment Vertical : increment
AM = 0 Horizontal				
AM = 1 Vertical				

جدول ۸

TRI	DFM	16-bit MPU System Interface Data Format
0	*	system 16-bit interface (1 transfers/pixel) 65,536 colors
1	0	80-system 16-bit interface (2 transfers/pixel) 262,144 colors
1	1	80-system 16-bit interface (2 transfers/pixel) 262,144 colors
TRI	DFM	8-bit MPU System Interface Data Format
0	*	system 8-bit interface (2 transfers/pixel) 65,536 colors
1	0	80-system 8-bit interface (3 transfers/pixel) 262,144 colors
1	1	80-system 8-bit interface (3 transfers/pixel) 262,144 colors

جدول ۸

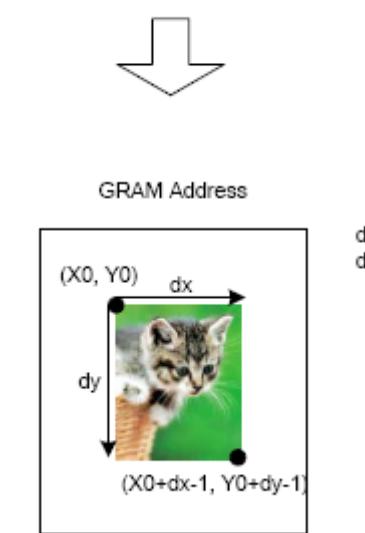
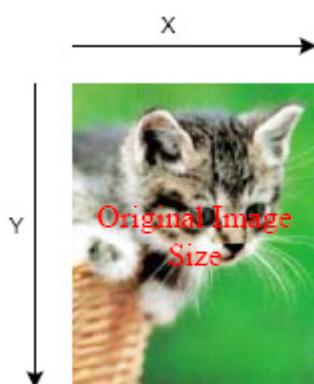
8.2.7. Resizing Control Register (R04h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	0	0	0	0	0	RCV1	RCV0	0	0	RCH1	RCH0	0	0	RSZ1	RSZ0

جدول ۹

مشاهده می شود که تصویر اصلی می تواند ۲ یا ۴ برابر کوچکتر شود.
بیت های (RCH1,RCH0): طبق جدول زیر مقدار این بیت ها باقی مانده پیکسل ها در جهت افقی نشان می دهد.

بیت های (rcv1,rcv0): مقدار این بیت ها باقی مانده پیکسل ها را درجهت عمودی نشان می دهد.



شکل ۳

شروع تصویر در gram می باشد. یعنی می توانیم تصویر کوچک شده را در هر نقطه‌ی دلخواه بر روی پانل lcd به نمایش در بیاوریم (باعظی مقداری $x0 \times y0$ و $(x0+dx-1, y0+dy-1)$).

نیز مختصات نقطه‌ی پایانی تصویر خواهد بود. پارامتر های H و V به ترتیب باقی مانده تعداد پیکسل ها در جهت افقی RCH و باقی مانده پیکسل ها در جهت عمودی RCV است و نیز RESIZING FACTOR یا ضریب کوچک شدن تصویر است.

دستور این قسمت (07H)DISPLAY CONTROL1-5: طبق جدول ۱۴ کد دستور این قسمت (07H) می باشد. که طبق معمول ابتدا باید این کد در رجیستر دستورالعمل نوشته شود.

RSZ[1:0]	Resizing factor
00	No resizing (x1)
01	$\times 1/2$
10	Setting prohibited
11	$\times 1/4$

جدول ۱۰

Original Image Size (X × Y)	Resized Image Resolution	
	1/2 (RSZ=2'h1)	1/4 (RSZ=2'h3)
640 × 480	320 × 240	160 × 120
352 × 288	176 × 144	88 × 72
320 × 240	160 × 120	80 × 60
176 × 144	88 × 72	44 × 36
120 × 160	60 × 80	30 × 40
132 × 132	66 × 66	33 × 33

جدول ۱۱

همان طور که در شکل ۳ دیده می شود. طول تصویر اصلی برابر x و عرض تصویر برابر y فرض شده است. بعد از عمل کوچک کردن

RCH [1:0]	Number of remainder Pixel in Horizontal Direction
00	0 Pixel
01	1 Pixel
10	2 Pixel
11	3 Pixel

جدول ۱۲

تصویر اصلی، تصویری که در پانل lcd به نمایش در خواهد آمد طولی برابر با dx و عرضی برابر dy خواهد داشت. $x0$ و $y0$ آدرس

RCV [1:0]	Number of remainder Pixel in Vertical Direction
00	0 Pixel
01	1 Pixel
10	2 Pixel
11	3 Pixel

جدول ۱۳

راه اندازی LCD N96 میکروکنترلر AVR

8.2.8. Display Control 1 (R07h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	0	PTDE1	PTDE0	0	0	0	BASEE	0	0	GON	DTE	CL	0	D1	D0

جدول ۱۴

در این صورت قسمت یک و دو تصویر غیر فعال می شوند و LCD فقط در صورتی که BASEE=1 باشد یک تصویر را نشان می دهد. و اگر مقدار این بیت ها یک شود (PTDE0=1) و (PTDE1=1) در این صورت می توانیم در پانل LCD دو تصویر مجزا را به نمایش در بیاوریم. یاد آور می شود که در این دراین حالت باید BASEE=0 باشد. (چگونگی ایجاد تصویر با استفاده از بیت های فعال ساز PTDE را در جدول ۳۴ به طور کامل توضیح خواهیم داد).

RGB DISPLAY INTERFACE CONTROL: ۱-۶ جدول ۲۱ کد دستور این قسمت (0CH) می باشد که ابتدا باید این کد را در رجیستر دستور العمل بنویسیم.

بیت های RTM0 و RTM1: طبق جدول زیر مقدار این بیت ها طول بیت های دیتا را در واسط سخت افزاری RGB تعیین می کند. به عنوان مثال اگر هر دو این بیت ها صفر شوند (RTM0=0 و RTM1=0) در این صورت داده ها به صورت ۱۸ بیتی منتقل می شوند (با استفاده از گذر گاه های DB0 تا DB17).

RIM1	RIM0	RGB Interface Mode
0	0	18-bit RGB interface (1 transfer/pixel), DB[17:0]
0	1	16-bit RGB interface (1 transfer/pixel), DB[17:13] and DB[11:1]
1	0	6-bit RGB interface (3 transfers/pixel), DB[17:12]
1	1	Setting disabled

جدول ۱۸

بیت های DM0 و DM1: طبق جدول (c) مقدار این بیت ها نوع واسط سخت افزاری را تعیین می کند. به عنوان مثال اگر مقدار این این بیت ها (01) باشد (dm1=0 و dm0=1) رابط سخت افزاری RGB انتخاب می شود.

DM1	DM0	Display Interface
0	0	Internal system clock
0	1	RGB interface
1	0	VSYNC interface
1	1	Setting disabled

جدول ۱۹

بیت RM: نوع واسط سخت افزاری را برای دستیابی به GRAM تعیین می کند به عنوان مثال اگر مقدار این بیت یک باشد (RM=1) در این صورت واسط سخت افزاری RGB برای نوشتن داده ها در GRAM به کار می رود.

RM	Interface for RAM Access
0	System interface/VSYNC interface
1	RGB interface

جدول ۲۰

8.2.12. RGB Display Interface Control 1 (R0Ch)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	ENC2	ENC1	ENC0	0	0	0	RM	0	0	DM1	DM0	0	0	RIM1	RIM0

جدول ۲۱

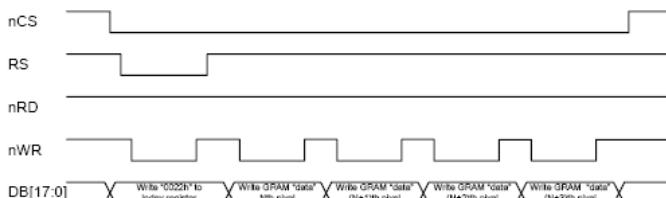
8.2.19. GRAM Horizontal/Vertical Address Set (R20h, R21h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	0	0	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
W	1	0	0	0	0	0	0	0	AD16	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8

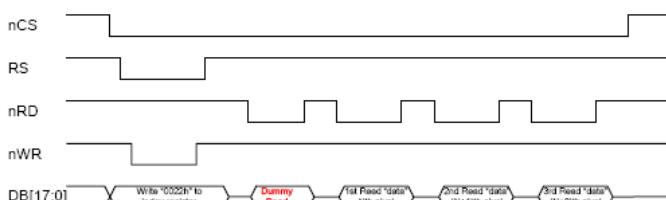
جدول ۲۲

تصویر در GRAM توسط بیت های AD16 تا AD0 می تواند تصویر را در هر نقطه ای دلخواه از پانل LCD به نمایش در آورد. READ DATA FROM GRAM: برای خواندن داده ها از GRAM ابتدا باید کد دستور این قسمت یعنی (22H) را در رجیستر (CS=0) دستور العمل بنویسیم سپس پایه ای CS را (CS=0) می کنیم (جدول ۲۵). موقعی که می خواهیم کد دستور را بنویسیم برای لحظه ای RS و WR را نیز RESET می کنیم (RS=0,WR=0) سپس

(a) Write to GRAM



(b) Read from GRAM



شكل ۴

بعد از نوشتن کد دستور RS و WR را SET می کنیم. بعد با هر بار RESET کردن RD یک پیکسل داده از GRAM خوانده شده و داده ها در روی گذرگاه های DB0 تا DB17 تا AD0 تا AD16 می گیرد.

بعد از نوشتن داده ها در GRAM WRITE DATA FROM GRAM: برای نوشتن داده ها در GRAM ابتدا باید کد دستور این قسمت (22H) را در رجیستر دستور العمل بنویسیم سپس پایه ای CS را RESET می کنیم (طبق جدول ۲۶). یعنی همه ی مراحل نوشتن در GRAM مانند خواندن از آن است. فقط با این تفاوت که در موقع نوشتن هر بار که بخواهیم یک پیکسل داده را در GRAM بنویسیم باید پایه ای WR را RESET و بعد از نوشتن، WR را باید SET کنیم.

8.2.21. Read Data from GRAM (R22h)

R/W	RS	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R	1	RAM Read Data (RD[17:0], the DB[17:0] pin assignment differs for each interface.)																	

جدول ۲۵

8.2.20. Write Data to GRAM (R22h)

R/W	RS	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	RAM write data (WD[17:0], the DB[17:0] pin assignment differs for each interface.)																	

جدول ۲۶

بیت های ENC0، ENC1، ENC2 و ENC3 عملکرد این بیت ها نیز به طور واضح در جدول زیر آورده شده است.

ENC[2:0]	GRAM Write Cycle (Frame periods)
000	1 Frame
001	2 Frames
010	3 Frames
011	4 Frames

جدول ۲۳

GRAM HORIZONTAL/VERTICAL ADDRESS SET: طبق جدول ۲۲ کد دستور این قسمت (20H) و (21H) می باشد که کد دستور (20H) مربوط به قسمت HORIZONTAL و (21H) مربوط به قسمت VERTICAL می باشد.

بیت های AD16 تا AD0 مقدار این بیت ها آدرس شروع در GRAM را مشخص می کند. بیت های AD7 تا AD0 که کد دستور این قسمت (20H) می باشد طول نقطه ای شروع در GRAM و بیت های AD8 تا AD16 که کد دستور این قسمت (21H) می باشد عرض نقطه ای شروع در GRAM را تعیین می کند. بنابراین مقدار بیت های AD0 تا AD7 می تواند از ۰ تا ۲۳۹ باشد و مقدار بیت های AD8 تا AD16 می تواند از ۰ تا ۳۱۹ می تواند (جدول ۲۴).

AD[16:0]	GRAM Data Map
17'h00000 ~ 17'h000EF	1 st line GRAM Data
17'h00100 ~ 17'h001EF	2 nd line GRAM Data
17'h00200 ~ 17'h002EF	3 rd line GRAM Data
17'h00300 ~ 17'h003EF	4 th line GRAM Data
17'h13D00 ~ 17'h13DEF	318 th line GRAM Data
17'h13E00 ~ 17'h13EEF	319 th line GRAM Data
17'h13F00 ~ 17'h13FEF	320 th line GRAM Data

جدول ۲۴

در اینجا لازم که به طور خلاصه با مفهوم GRAM آشنا شویم. GRAM: Hemanstor که از اسمش پیداست مخفف کلمه ای Graphic RAM (رم گرافیکی) است که اطلاعات مربوط به پیکسل ها در این RAM ذخیره شده و کنترلر LCD این اطلاعات را از GRAM خوانده و پیکسل ها را مقدار دهی می کند. هر پیکسل در آدرسی دارد و به همین دلیل با تغییر آدرس نقطه ای شروع

راه اندازی LCD N96 توسط میکروکنترلر AVR

8.2.25. Horizontal and Vertical RAM Address Position (R50h, R51h, R52h, R53h)

	R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R50h	W	1	0	0	0	0	0	0	0	HSA7	HSA8	HSA5	HSA4	HSA3	HSA2	HSA1	HSA0	
R51h	W	1	0	0	0	0	0	0	0	HEA7	HEA8	HEA5	HEA4	HEA3	HEA2	HEA1	HEA0	
R52h	W	1	0	0	0	0	0	0	0	VSA8	VSA7	VSA6	VSA5	VSA4	VSA3	VSA2	VSA1	VSA0
R53h	W	1	0	0	0	0	0	0	0	VEA8	VEA7	VEA6	VEA5	VEA4	VEA3	VEA2	VEA1	VEA0

جدول ۲۷

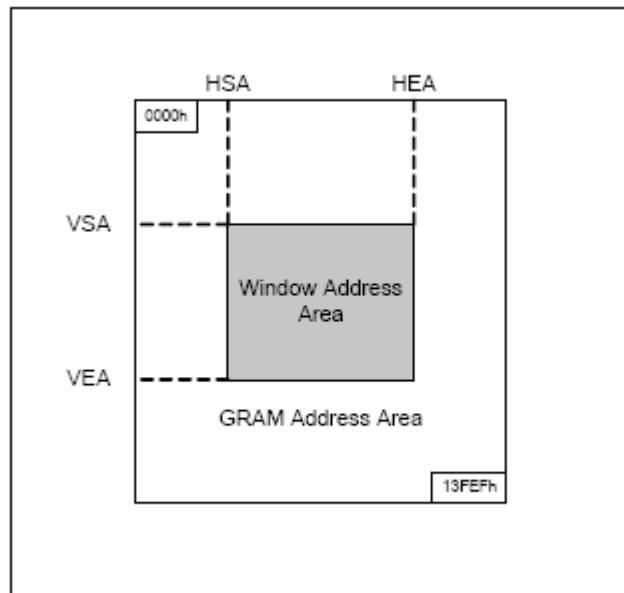
VSA \leq VEA , VEA \leq 319 باشد. هچنین مقدار بیت های VEA کوچکتر از VEA باشد.

: طبق جدول ۲۸ این جدول نیز از GATE SCAN CONTROL سه قسť تشکیل شده است که هر قسť کد دستور مربوط به خود را دارد. در اینجا به علت مهم بودن قسمت اول فقط قسمت اول این جدول را توضیح خواهیم داد.

بیت های SCAN5 تا SCAN0: مقدار این بیتها با توجه به جدول b سطر شروع جاروب را تعیین می کنند به عنوان مثال اگر مقدار این بیت ها (OAH) باشد سطر شروع جاروب از سطر ۸۱ شروع خواهد شد(جدول ۲۹).

بیت های NL5 تا NL0: مقدار این بیتها تعداد سطر LCD را

Horizontal and vertical RAM address position جدول ۲۷ این جدول از چهار قسمت تشکیل شده است که برای تنظیم نمودن هر یک از این قسمت ها ابتدا باید کد دستور مربوط را ارسال کرده و سپس داده ی مربوطه را در رجیستر داده بنویسیم. طبق شکل زیر می توانیم در GRAM قسمت مشخصی را به عنوان مساحت پنجره تعیین کنیم تا تصویر فقط در این قسمت تشکیل شود. برای این منظور باید بیت های جدول ۲۷ را مقدار دهی کنیم. بیت های HSA0 تا HSA7: مقدار این بیتها طول نقطه ی شروع



شکل ۵

را در GRAM تعیین می کنند. بیت های HEA7 تا HEA0: مقدار این بیتها طول نقطه ی پایان را در GRAM تعیین می کنند.

مقدار بیت های HSA و HEA باید در محدوده ی 0 \leq HEA , HEA \leq 239 باشد. هچنین مقدار HSA کوچکتر از HEA باشد.

بیتها VSA8 تا VSA0: مقدار این بیتها عرض نقطه ی شروع را در GRAM مشخص می کند.

بیتها VEA8 تا VEA0: مقدار این بیتها عرض نقطه ی شروع را در GRAM مشخص می کند.

مقدار بیت های VSA و VEA باید در محدوده ی

جدول ۲۹

8.2.26. Gate Scan Control (R60h, R61h, R6Ah)

	R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R60h	W	1	GS	0	NL5	NL4	NL3	NL2	NL1	NL0	0	0	SCN5	SCN4	SCN3	SCN2	SCN1	SCN0
R61h	W	1	0	0	0	0	0	0	0	0	0	0	0	0	0	NDL	VLE	REV
R6Ah	W	1	0	0	0	0	0	0	0	VL8	VL7	VL6	VL5	VL4	VL3	VL2	VL1	VL0

جدول ۲۸

راه اندازی AVR N96 توسط میکروکنترلر LCD

مقاومت ها (VRCP1, VRCP1, VRCN1, VRCN1) نیز از نوع مقاومت های متغیر هستند (مقاومت تنظیم شیب) که مقدار این مقاومت ها می تواند بین ۰ تا ۲۸ اهم متغیر باشد و برای تنظیم مقدار این مقاومت ها بیت های [PRPN2:0] و [PRNN2:0] به کار می روند. با تغییر مقدار این مقاومت ها افت ولتاژ در روی خود این مقاومت ها و مقاومت های دیگر نیز تغییر خواهد کرد و تغییر این ولتاژ باعث تغییر شیب ولتاژ GRAYSCALE خواهد شد.

بعد از تنظیم مقدار مقاومت های متغیر توسط بیتها ذکر شده در روی هر یک از مقاومت های RP0 تا RP47 افت ولتاژ به وجود خواهد آمد که افت ولتاژ این مقاومت ها نسبت به زمین V_{P1} است و به همین نام گذاری شده است مثلا افت ولتاژ R_{P1} , V_{P1} است و به همین ترتیب پیش می رود حال با توجه به شکل به ۸ تا از این مقاومت هایه یک سلکتور وصل شده و کار این سلکتور ها این است که با توجه به مقدار مقدار بیت های [PKN2:0] یکی از این افت ولتاژ را انتخاب می کنند و به خروجی سلکتور (VGPN) می رسانند شکل

تعیین می کند به عنوان مثال اگر مقدار این بیتها (1FH) را داشته باشد، LCD ۲۵۶ سطری خواهد شد و سطرهای بالاتر از ۲۵۶ تا ۳۲۰ بدون استفاده خواهد ماند.

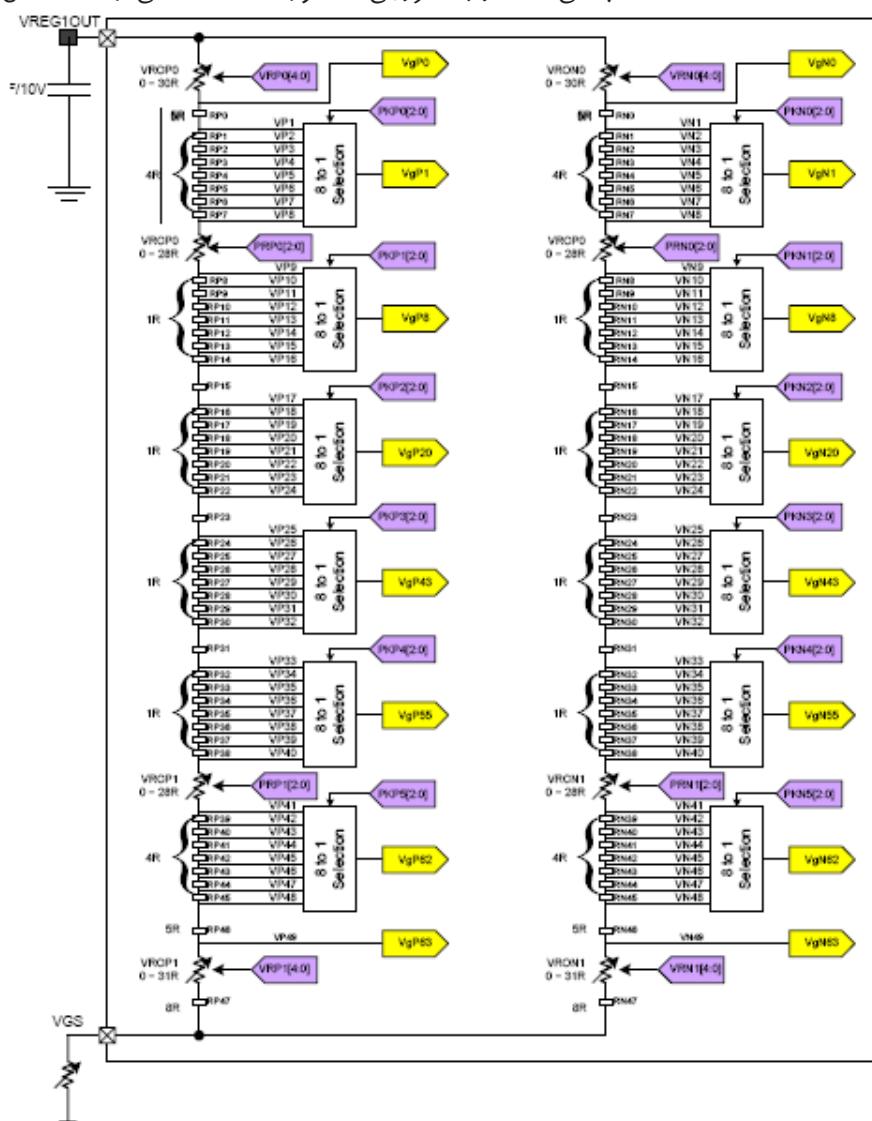
بیت GS: مقدار این بیت جهت شروع جاروب در LCD را تعیین

NL[5:0]	LCD Drive Line
6'h1D	240 lines
6'h1E	248 lines
6'h1F	256 lines
6'h20	264 lines
6'h21	272 lines
6'h22	280 lines
6'h23	288 lines
6'h24	296 lines
6'h25	304 lines
6'h26	312 line
6'h27	320 line
Others	Setting inhibited

جدول ۳۰

می کند اگر مقدار این بیت ۱ شود (GS=1) در اینصورت طبق جدول A جهت جاروب از G320 به طرف G1 می باشد (G1<--G320) و اگر مقدار این بیت صفر شود جهت جاروب از G320 به طرف G1 می باشد اگر جهت جاروب از G1 به طرف G320 باشد (G320<--G1) از G1 بدان معناست که نقطه ی شروع تصویر در این LCD از G1 شروع می شود و اگر جهت جاروب از G320 به طرف G1 باشد در اینصورت عکس اتفاقات بالا رخ می دهد یعنی سطر شروع تصویر از G320 شروع می شود.

۸.۲.۲۳: Gamma control: همانطور که در جدول ۸.۲.۲۳ دیتاپیت مشاهده می شود این جدول از چندین قسمت تشکیل شده که هر قسمت کد دستور مربوط به خود را دارد که برای تنظیم هر قسمت ابتدا باید کد دستور آن قسمت را ارسال کرده و سپس به مقدار دهی بیت های آن قسمت پردازیم و سپس در رजیستر داده را بنویسیم. همانطور که از اسم این قسمت پیداست این قسمت برای تنظیم شیب، شفافیت و دامنه به کار می رود. در ابتدا به توضیح مختصرا در باره بلوك دیاگرام این قسمت می پردازیم با توجه به شکل الف مقاومت های (VRPN0, VRPN1, VRON0, VRON1) از نوع مقاومت متغیر هستند که مقدار آنها می تواند از ۰ تا ۳۰ اهم تغییر یابد و مقدار این مقاومت ها توسط VRPN[2:0] تنظیم می شوند. با تغییر مقاومت های این قسمت افت ولتاژ تمامی مقاومت های دیگر نیز تغییر می کند (مقاومت های تنظیم دامنه).

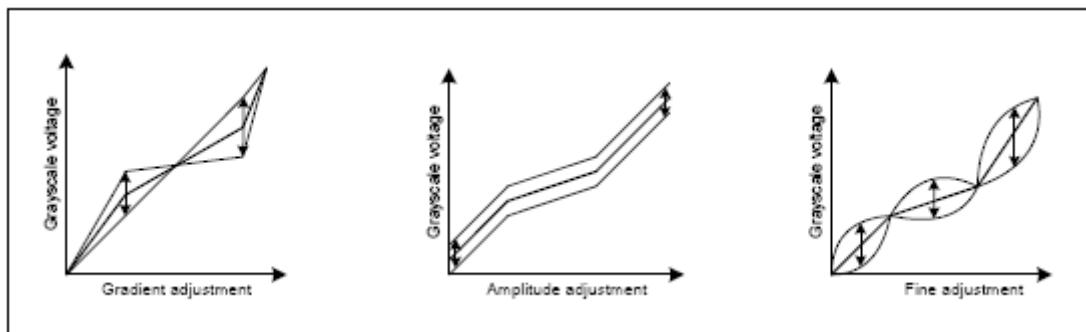


شکل ۶

8.2.33. Panel Interface Control 4 (R95h)																	
R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	0	0	0	0	0	DIVE1	DIVE0	0	0	RTNE5	RTNE4	RTNE3	RTNE2	RTNE1	RTNE0

جدول ۳۱

راه اندازی LCD N96 میکروکنترلر AVR



شکل ۷

:Partial image 1 display positon(R80H)

طبق جدول ۳۴ کد دستور این قسمت (80H) می باشد. ابتدا باید این کد به رجیستر دستورالعمل نوشته شود.

بیت های [PTD P0[0] تا PTD P0[8]: مقدار بیت های این قسمت سطر شروع تصویر را در پانل LCD مشخص می کند. همانطور که قبل از گفته شد می توان در پانل LCD دو تصویر مجزا را به نمایش در آورد.

:Partial image 1 RAM start/end address بیت های [PTS A0[0] تا PTS A0[8]: مقدار این بیت ها سطر شروع تصویر یک را در GRAM مشخص می کند.

PTE A0[0] تا PTE A0[8]: مقدار این بیت ها سطر پایان تصویر یک در GRAM را مشخص می کند.

بیت های [Partial image 2display postion سطر شروع تصویر ۲ را در پانل LCD تعیین می کند.

:Partial image 2 RAM start/end address بیت های [PTS A1[0] تا PTS A1[8]: مقدار بیت های این قسمت سطر شروع تصویر ۲ را در GRAM مشخص می کند.

بیت های [PTE A1[0] تا PTE A1[8]: مقدار بیت های این قسمت سطر پایان تصویر ۲ را در GRAM مشخص می کند. با توجه به توضیحات بالا یک مثال در مورد چگونگی ایجاد دو تصویر در LCD را بررسی می کنیم. طبق جدول ۸ چون می خواهیم دو تصویر را در LCD نمایش دهیم پس باید مقدار بیت BASEE را صفر کنیم و تعداد سطرهای LCD را نیز با کمک بیت های [NL[5:0] تعیین کنیم (۳۲۰ سطر). حال باید بیت های فعال ساز قسمت تصویر

زیر نمودار تغییرات ولتاژ GRAYSCALE به ازای تغییرات شبیه دامنه و شفافیت نشان می دهد . حال به عملکرد بیت های جدول ۳۱ می پردازیم.

بیت های [KPN[2:0]: مقدار این بیت ها برای تنظیم شفافیت ولتاژ GRAYSCALE به کار می روند که طبق جدول ۷ مقدار این بیت ها ولتاژ V_{PN} را در سلکتور انتخاب می کند.

بیت های [RPN[2:0]: مقدار این بیت ها مقدار مقاومت های VRCP(N)۰ را تعیین می کند(تنظیم شبیه ولتاژ GRAYSCALE) بیت های [VRPN[3:0]: مقدار این بیت ها برای تنظیم دامنه ولتاژ

Fine adjustment registers and selected voltage

Register	Selected Voltage						
	KP(N)[2:0]	VgP(N)1	VgP(N)8	VgP(N)20	VgP(N)43	VgP(N)55	VgP(N)62
000	VP(N)1	VP(N)9	VP(N)17	VP(N)25	VP(N)33	VP(N)41	
001	VP(N)2	VP(N)10	VP(N)18	VP(N)26	VP(N)34	VP(N)42	
010	VP(N)3	VP(N)11	VP(N)19	VP(N)27	VP(N)35	VP(N)43	
011	VP(N)4	VP(N)12	VP(N)20	VP(N)28	VP(N)36	VP(N)44	
100	VP(N)5	VP(N)13	VP(N)21	VP(N)29	VP(N)37	VP(N)45	
101	VP(N)6	VP(N)14	VP(N)22	VP(N)30	VP(N)38	VP(N)46	
110	VP(N)7	VP(N)15	VP(N)23	VP(N)31	VP(N)39	VP(N)47	
111	VP(N)8	VP(N)16	VP(N)24	VP(N)32	VP(N)40	VP(N)48	

جدول ۳۲

GRAYSCALE به کار می رود طبق جدول ۷ مقدار این بیت ها مقدار مقاومت های متغیر VROP(N)۰ را تعیین می کنند همهی بیت های گفته شده در بالا برای پلاریته ای مثبت هستند و بقیه ای بیت های جدول ۳۱ برای پلاریته ای منفی هستند و توضیحات بالا در مورد آنها نیز صدق می کند. به عنوان مثال تنظیم شبیه برای پلاریته ای مثبت RP هست و تنظیم شبیه برای RN هست و بقیه نیز به همین شکل هستند.

Gradient adjustment		Amplitude adjustment (1)				Amplitude adjustment (2)			
PRP(N)0/1[2:0]	VRCP(N)0	VRP(N)0[3:0]	VRP(N)0	VRP(N)1[4:0]	VROP(N)1	Register	Resistance	Register	Resistance
000	0R	0000	0R	00000	0R	00000	0R	00001	1R
001	4R	0001	2R	00001	1R	00001	2R	00010	2R
010	8R	0010	4R	00010	2R	00010	4R	00011	4R
011	12R	:	:	00011	4R	00011	4R	00012	8R
100	16R	:	:	00012	8R	00012	8R	00013	12R
101	20R	1101	28R	00013	12R	00013	12R	00014	20R
110	24R	1111	28R	00014	20R	00014	20R	00015	24R
111	28R	1111	30R	00015	24R	00015	24R	00016	28R

جدول ۳۳

8.2.27. Partial Image 1 Display Position (R80h)

R/W	RS	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
W	1	0	0	0	0	0	0	0	PTD P0[8]	PTD P0[7]	PTD P0[6]	PTD P0[5]	PTD P0[4]	PTD P0[3]	PTD P0[2]	PTD P0[1]	PTD P0[0]

جدول ۳۴

راه اندازی LCD N96 توسط میکروکنترلر AVR

می‌توانیم تصویر ۱ را در هر نقطه‌ی دلخواه از پانل LCD نمایش دهیم برای این منظور فقط کافی است سطر شروع تصویر ۱ در پانل LCD را توسط بیت‌های PTD P0[8:0] تعیین کنیم که در این مثال ما سطر شروع تصویر ۱ را در پانل LCD (80H) در نظر گرفتیم (سطر ۱۲۸) همه‌ی این توضیحات در مورد تصویر دو نیز صدق می‌کند شکل ۸ همه‌ی توضیحات بالا را نشان می‌دهد با توجه به همه‌ی توضیحات بالا یک برنامه جهت آموزش راه اندازی این LCD به زبان بیسیک به میکروکنترلر AVR نوشته شده است. این مقاله جهت آموزش دو رنگ LCD روی به نمایش در آورده‌یم و در قسمت بعد به آموزش ایجاد تصویر، فونت، ... خواهیم پرداخت.

```
$regfile = "m32def.dat"
$crystal = 12000000
1
قسمت
Config Porta = Output
Config Portc = Output
Config Portd = Output
Cs Alias Portd.0
Rs Alias Portd.1
Wr Alias Portd.2
Rd Alias Portd.3
Rst Alias Portd.4
Lsb_port Alias Porta
Msb_port Alias Portc
Declare Sub Trigerlcd
Declare Sub Writedata
Declare Sub Writeindex
```

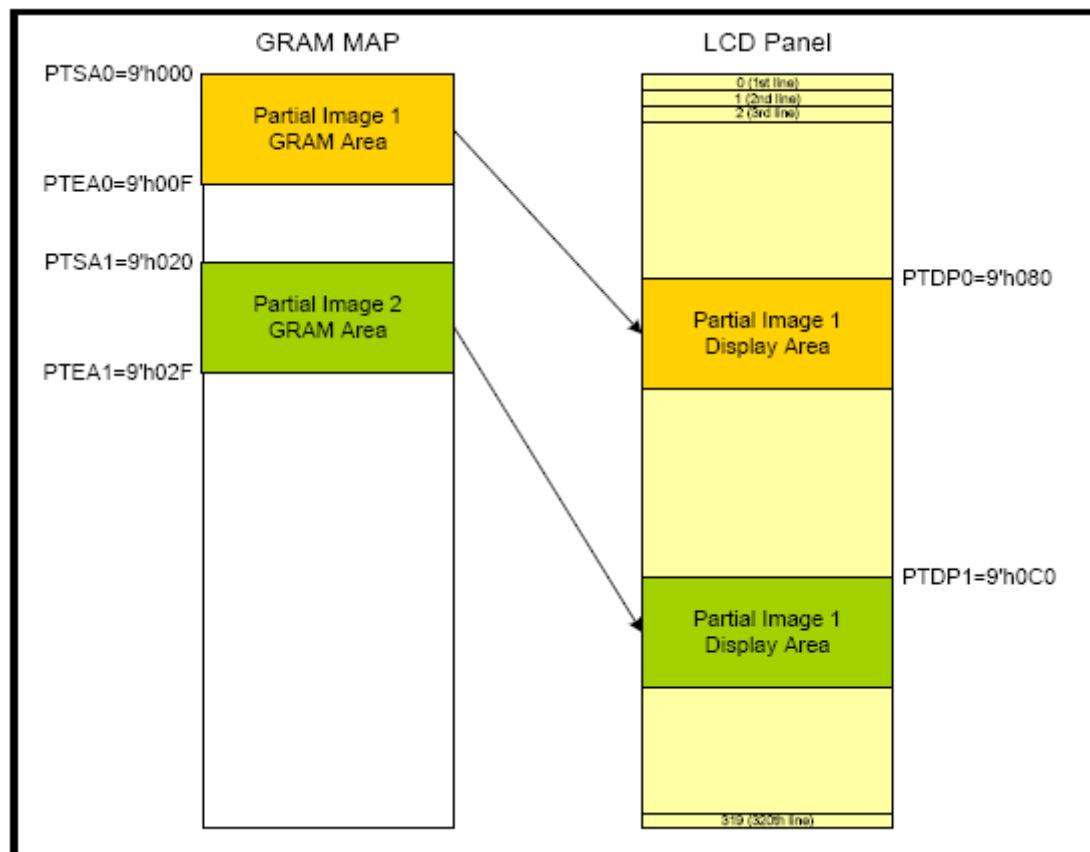
'lsb port
'msb port

۱ و تصویر ۲ را نیز فعال کنیم که این بیت‌ها با مقدار ۱ فعال می‌شوند (PTDE1=1, PTDE0=1). بعد از فعال سازی این بیت‌ها سطر شروع تصویر ۱ را با استفاده از بیت‌های PTS A0[8] تا PTS A0[0] در PTS A0[8] در GRAM تعیین

Base Image Display Setting	
BASEE	0
NL[5:0]	6'h27
Partial Image 1 Display Setting	
PTDE0	1
PTSA0[8:0]	9'h000
PTEA0[8:0]	9'h00F
PTDP0[8:0]	9'h080
Partial Image 2 Display Setting	
PTDE1	1
PTSA1[8:0]	9'h020
PTEA1[8:0]	9'h02F
PTDP1[8:0]	9'h0C0

جدول ۳۵

می‌کنیم که طبق جدول a سطر شروع تصویر در GRAM را برابر (000H) در نظر گرفتیم سپس توسط بیت‌های [0] تا [8] سطر پایانی تصویر ۱ را در GRAM مشخص می‌کنیم که طبق جدول a سطر پایانی تصویر را در GRAM برابر (00FH) در نظر گرفتیم بعد از آدرس دهی تصویر ۱ در GRAM نوبت به نمایش آن در پانل LCD می‌رسد طبق توضیحات قبلی



شکل ۸

راه اندازی LCD N96 توسط میکروکنترلر AVR

```

Declare Sub Colorlcd
Dim Index As Byte , Dat As Word , Reapet As Long
2 قسمت
Const Start_osc = &H00
Const Driver_output_control = &H01
Const Lcd_driving_wave_control = &H02
Const Entry_mode = &H03
Const Resizing_control = &H04
Const Display_control1 = &H07
Const Display_control2 = &H08
Const Display_control3 = &H09
Const Display_control4 = &H0A
Const Rgb_display_interface_control1 = &H0C
Const Frame_marker_position = &H0D
Const Rgb_display_interface_control2 = &H0F
Const Power_control1 = &H10
Const Power_control2 = &H11
Const Power_control3 = &H12
Const Power_control4 = &H13
Const Gram_horizontal_address = &H20
Const Gram_vertical_address = &H21
Const Write_read_data_to_gram = &H22
Const Power_control7 = &H29
Const Frame_rate_and_color_control = &H2B
Const Gamma_control1 = &H30
Const Gamma_control2 = &H31
Const Gamma_control3 = &H32
Const Gamma_control4 = &H35
Const Gamma_control5 = &H36
Const Gamma_control6 = &H37
Const Gamma_control7 = &H38
Const Gamma_control8 = &H39
Const Gamma_control9 = &H3C
Const Gamma_control10 = &H3D
Const Horizontal_start_address = &H50
Const Horizontal_end_address = &H51
Const Vertical_start_address = &H52
Const Vertical_end_address = &H53
Const Gate_scan_control1 = &H60
Const Gate_scan_control2 = &H61
Const Gate_scan_control3 = &H6A
Const Partial_image1_display_position = &H80
Const Partial_image1_ram_start_address = &H81
Const Partial_image1_ram_end_address = &H82
Const Partial_image2_display_position = &H83
Const Partial_image2_ram_start_address = &H84
Const Partial_image2_ram_end_address = &H85
Const Panel_interface_control1 = &H90
Const Panel_interface_control2 = &H92
Const Panel_interface_control4 = &H95
Const Color_red = &H001F
Const Color_green = &H07E0
Const Color_blue = &HF800
Const Color_yellow = &H07FF
Const Color_purple = &HF81F
Const Color_black = &H0000
Const Color_white = &HFFFF
3 قسمت
'-----
Trigerlcd
Colorlcd
End
'-----'end program
'-----4 قسمت
Sub Trigerlcd
Reset Rst
Waitms 60
Set Rst
Waitms 200

```

توضیحات قسمت ۱ :

در این قسمت بعد از خروجی تعریف کردن پورت های مربوطه در خطوط ۳ الی ۵ پین هایی از میکرو را که به پایه های RST,WR,RD,RS,CS)LCD(وصل می شوند، تعیین می کنیم. چون ما در این برنامه از مدار ۱۶ بیتی LCD استفاده کردیم پس باید دو پورت از میکرو را به پین های DB17 تا DB1 وصل کنیم. حال پورتی که به پین های DB8 تا DB1 وصل می شود پورت کم ارزش می نامیم که در برنامه در خط ۱۱ پورت A به عنوان DB8 وصل کنیم. باید پین های DB1 تا DB10 به پین های DB17 وصل شوند و پورتهایی که به پین های DB12 پورت C به عنوان پورت پرازش می نامیم که در برنامه در خط ۱۲ پورت C به عنوان پورت پرازش معرفی شده است که باید پین های DB17 تا DB10 به پورت C وصل شوند. بعد از تعریف کردن پورتهای LSB و MSB در برنامه از خطوط ۱۳ الی ۱۶ ساب روتنی های برنامه تعریف شده است.

توضیحات قسمت ۲ :

در این قسمت کدهای دستور هر قسمت با نام آن قسمت در برناه توسط دستور const از خطوط ۱۸ تا ۶۴ در برنامه تعریف شده است. مثلا کد دستور قسمت ENTRY MODE در ENTRY MODE عدد (03H) می باشد که در برنامه DATA SHEET کد دستور این قسمت یعنی (H03) انتخاب شده است.

توضیحات قسمت ۳ :

در این قسمت از برنامه سابر روتنی های TRIGERLCD و COLORLCD فراخوانی شده است. که سابر روتنی TRIGERLCD برای راه اندازی اولیه LCD و سابر روتنی COLORLCD برای رنگی کردن LCD فراخوانی شده است.

توضیحات قسمت ۴ :

ساب روتنی TRIGERLCD در این سابر روتنین کدهای دستور در متغیر INDEX که یک متغیر یک بایتی است قرار می گیرد و سپس سابر روتنین WRITEINDEX فراخوانی می شود و مقدار متغیر INDEX که همان کدهای دستور هر قسمت می باشد در رجیستر دستور العمل نوشته می شود سپس مقدار داده ری در قسمت در متغیر DAT که یک متغیر دو بایتی است قرار می گیرد و با فراخوانی سابر روتنین WRITEDATA مقدار متغیر DAT در رجیستر داده نوشته می شود. به عنوان مثال می خواهیم بیت های قسمت WRITEDATA را مقدار دهی کنیم و سپس در رجیستر داده بنویسیم در خط ۱۳۳ برنامه ابتدا کد دستور GRAM VERTICAL ADDRESS این قسمت (21H) با نام همان قسمت (GRAM VERTICAL) در متغیر ADDRESS قرار می گیرد و با فراخوانی WRITEINDEX این کد در رجیستر دستور العمل نوشته می شود. بعد از مقدار دهی بیت های این قسمت که در برنامه در خط ۱۴۴ (0000H) مقدار دهی شده است در متغیر DAT قرار گرفته و با فراخوانی سابر روتنین WRITEDATA در رجیستر داده نوشته می شود.

توضیحات قسمت ۵ :

ساب روتنین WRITEDATA این سابر روتنین همان طور که از نامش پیداست داده ها را از رجیستر داده می نویسد، که نحوه

راه اندازی AVR میکروکنترلر LCD N96 توسط

```
Index = &H00E3 : Writeindex  
Dat = &H3008 : Writedata  
Index = &H00EF : Writeindex  
Dat = &H1231 : Writedata  
Index = Start_osc : Writeindex  
Dat = &H0001 : Writedata  
Waitms 50  
Index = Driver_output_control : Writeindex  
Dat = &H0100 : Writedata  
Index = Lcd_driving_wave_control : Writeindex  
Dat = &H0700 : Writedata  
Index = Entry_mode : Writeindex  
Dat = &H0030 : Writedata  
Index = Resizing_control : Writeindex  
Dat = &H0000 : Writedata  
Index = Display_control1 : Writeindex  
Dat = &H0133 : Writedata  
Index = Display_control2 : Writeindex  
Dat = &H0202 : Writedata  
Index = Display_control3 : Writeindex  
Dat = &H000F : Writedata  
Index = Display_control4 : Writeindex  
Dat = &H0000 : Writedata  
Index = Rgb_display_interface_control1 : Writeindex  
Dat = &H0000 : Writedata  
Index = Frame_marker_position : Writeindex  
Dat = &H0000 : Writedata  
Index = Rgb_display_interface_control2 : Writeindex  
Dat = &H0000 : Writedata  
Index = Power_control1 : Writeindex  
Dat = &H0000 : Writedata  
Index = Power_control2 : Writeindex  
Dat = &H0000 : Writedata  
Index = Power_control3 : Writeindex  
Dat = &H0000 : Writedata  
Index = Power_control4 : Writeindex  
Dat = &H0000 : Writedata  
Waitms 200  
Index = Power_control1 : Writeindex  
Dat = &H17A0 : Writedata  
Index = Power_control2 : Writeindex  
Dat = &H0137 : Writedata  
Waitms 70  
Index = Power_control3 : Writeindex  
Dat = &H0018 : Writedata  
Waitms 70  
Index = Power_control4 : Writeindex  
Dat = &H1000 : Writedata  
Index = Power_control7 : Writeindex  
Dat = &H000B : Writedata  
Waitms 70  
Index = Gram_horizontal_address : Writeindex  
Dat = &H0000 : Writedata  
Index = Gram_vertical_address : Writeindex  
Dat = &H0000 : Writedata  
Index = Gamma_control1 : Writeindex  
Dat = &H0101 : Writedata  
Index = Gamma_control2 : Writeindex  
Dat = &H0101 : Writedata  
Index = Gamma_control3 : Writeindex  
Dat = &H0101 : Writedata  
Index = Gamma_control4 : Writeindex  
Dat = &H0101 : Writedata  
Index = Gamma_control5 : Writeindex  
Dat = &H0201 : Writedata  
Index = Gamma_control6 : Writeindex  
Dat = &H0707 : Writedata  
Index = Gamma_control7 : Writeindex  
Dat = &H0707 : Writedata
```

نوشتن داده در رجیستر داده قبل توضیح داده شده است. برای این کار ابتدا CS را فعال می کنیم (خط ۱۷۹) سپس RS و RD را می کنیم (خط ۱۸۰ و ۱۸۱) سپس مقدار داده را که در متغیر دو بایتی قرار دارد، ابتدا بایت پر ارزش متغیر DAT را با استفاده از دستور HIGH در پورت پر ارزش قرار می دهیم و بایت کم ارزش متغیر DAT را با استفاده از دستور LOW در پورت کم ارزش قرار می دهیم . بعد از قرار دادن مقدار داده در روی پورت ها پایهی RESET را می کنیم و در این حالت مقادیر روی پورت ها در رجیستر داده نوشته می شوند بعد از این کار WR را SET می کنیم. در این حالت عملیات نوشتن تمام می شود. بعد از تمام شدن عملیات نوشتن CS را غیر فعال می کنیم (خط ۱۸۶).

توضیحات قسمت ۶:

سابروتین WRITEINDEX: این سابروتین هم کد دستور را در رجیستر دستور العمل می نویسد. برای اینکار ابتدا باید CS را فعال کنیم (خط ۱۸۹) حال چون می خواهیم در رجیستر دستوالعمل بنویسیم پس باید پایهی RS را نیز REST کنیم و RD را SET می کنیم. سپس کد دستور را در متغیر یک بایتی INDEX قرار دارد در روی پورت های مربوطه بارگذاری می کنیم. چون متغیر INDEX یک متغیر یک بایتی است پس باید به جای بایت پر ارزش (00H) را بار می کنیم. (خط ۱۹۲) و مقدار متغیر INDEX را در پورت کم ارزش قرار می دهیم (خط ۱۹۳) سپس با RESET مقدار روی پورت ها وارد رجیستر دستور العمل می شوند بعد از اینکار WR را SET می کنیم و بعد از اتمام شدن عملیات CS را نیز غیر فعال می کنیم (خط ۱۹۶).

توضیحات قسمت ۷:

سابروتین COLORLCD : همانطور که از نام این سابروتین پیداست این سابروتین برای در آوردن رنگ های مختلف در LCD استفاده شده است. در ابتدا مساحت پنجره ای که قرار است تصویر در آن به نمایش در آید تعیین می کنیم که این مورد را قبل توضیحدادیم. در خطوط ۱۹۹ و ۲۰۰ برنامه نقطه ای شروع عرض و در خطوط ۲۰۱ و ۲۰۲ نقطه ای پایان عرض را در پانل LCD تعیین کردیم که نقطه ای شروع (50H) و نقطه ای پایان (A0H) است. در خطوط ۲۰۳ و ۲۰۴ عرض نقطه ای شروع تصویر در GRAM را تعیین می کنیم که (50H) است و در خطوط ۲۰۵ تا ۲۰۸ طول مساحت پنجره را در پانل LCD تعیین می کنیم که طبق برنامه نقطه ای شروع طول (64H) و نقطه ای پایان طول (C8H) می باشد و در خطوط ۲۰۹ و ۲۱۰ طول نقطه ای شروع در GRAM را تعیین می کنیم که (64H) می باشد. حال نوبت به ارسال داده ها به GRAM می رسد پون این داده ها باید در GRAM ابتدای ۲۲H را در پس قبل از نوشتن داده ها در خط ۲۱۱ برنامه اجرا کردیم رجیستر دستوالعمل بنویسیم اینکار را در خط ۲۱۱ برنامه اجرا کردیم. حال تعداد پیکسل ها را در مساحت تعیین شده به دست می آوریم. طبق شکل زیر عرض پنجره ۸۰ پیکسل و طول پنجره ۱۰۰ پیکسل می باشد. پس در مساحت تعیین شده ۸۰۰۰ پیکسل وجود دارد. پس ما باید ۸۰۰۰ پیکسل داده به GRAM ارسال کنیم. در خطوط ۲۱۲ الی ۲۱۵ با هر بار اجرای حلقه FOR NEXT یک پیکسل داده به GRAM ارسال می شود بنابراین در خطوط

راه اندازی LCD N96 میکروکنترلر AVR

```

Index = Gamma_control8 : Writeindex
Dat = &H0707 : Writedata
Index = Gamma_control9 : Writeindex
Dat = &H0101 : Writedata
Index = Gamma_control10 : Writeindex
Dat = &H0201 : Writedata
Index = Horizontal_start_address : Writeindex
Dat = &H0000 : Writedata
Index = Horizontal_end_address : Writeindex
Dat = &H00EF : Writedata
Index = Vertical_start_address : Writeindex
Dat = &H0000 : Writedata
Index = Vertical_end_address : Writeindex
Dat = &H013F : Writedata
Index = Gate_scan_control1 : Writeindex
Dat = &H2700 : Writedata
Index = Gate_scan_control2 : Writeindex
Dat = &H0001 : Writedata
Index = Gate_scan_control3 : Writeindex
Dat = &H0000 : Writedata
Index = Panel_interface_control1 : Writeindex
Dat = &H0010 : Writedata
Index = Panel_interface_control2 : Writeindex
Dat = &H0000 : Writedata
Index = Panel_interface_control4 : Writeindex
Dat = &H0110 : Writedata
Index = Frame_rate_and_color_control : Writeindex
Dat = &H0000 : Writedata
End Sub
5 قسمت
Sub Writedata
Reset Cs
Set Rs
Set Rd
Msb_port = High(dat)
Lsb_port = Low(dat)
Reset Wr
Set Wr

```

```

Set Cs
End Sub
6 قسمت
Sub Writeindex
Reset Cs
Reset Rs
Set Rd
Msb_port = &H00
Lsb_port = Index
Reset Wr
Set Wr
Set Cs
End Sub
7 قسمت
Sub Colorlcd
Index = Horizontal_start_address : Writeindex
Dat = &H0050 : Writedata
Index = Horizontal_end_address : Writeindex
Dat = &H00A0 : Writedata
Index = Gram_horizontal_address : Writeindex
Dat = &H0050 : Writedata
Index = Vertical_start_address : Writeindex
Dat = &H0064 : Writedata
Index = Vertical_end_address : Writeindex
Dat = &H00C8 : Writedata
Index = Gram_vertical_address : Writeindex
Dat = &H0064 : Writedata
Index = Write_read_data_to_gram : Writeindex
For Reapet = 1 To 4000
Dat = Color_yellow
Writedata
Next Repeat
For Reapet = 1 To 4000
Dat = Color_red
Writedata
Next Repeat
End Sub

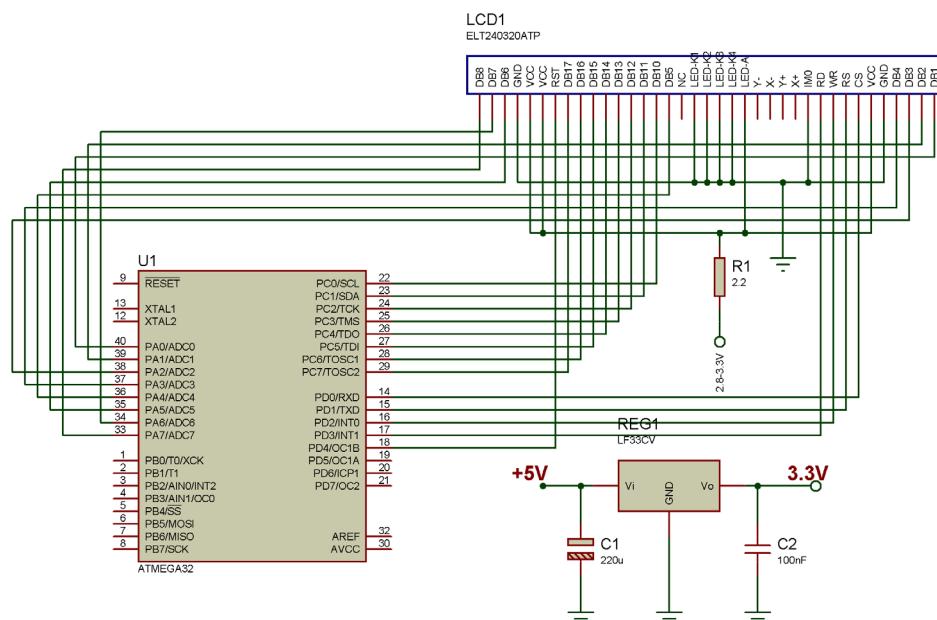
```

پیکسل داده به GRAM ارسال می شود

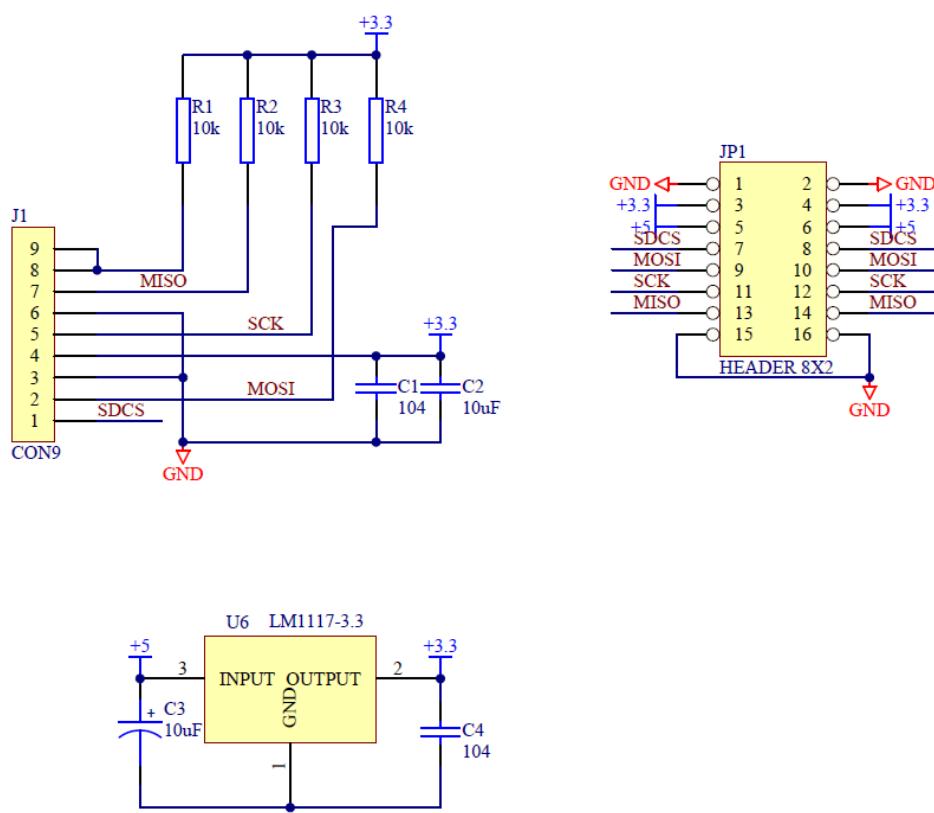
منبع: دیتاشیت ILI9325

<http://miladelectronic.vcp.ir> M.M.S.نویسنده گان: گروه

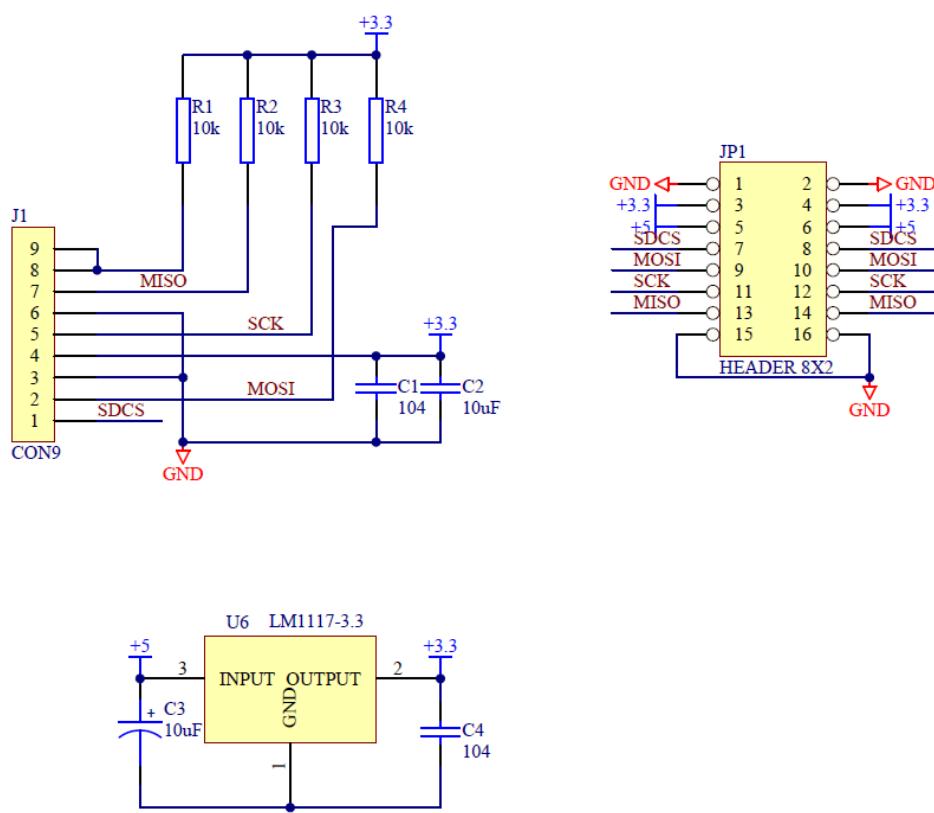
۲۱۵ الی ۴۰۰ پیکسل داده (رنگ زرد) به GRAM ارسال می شود. از خطوط ۲۱۶ الی ۲۱۹ ۴۰۰ پیکسل داده (رنگ قرمز) به GRAM ارسال می شود که در این سایر وظایف در مجموع ۸۰۰

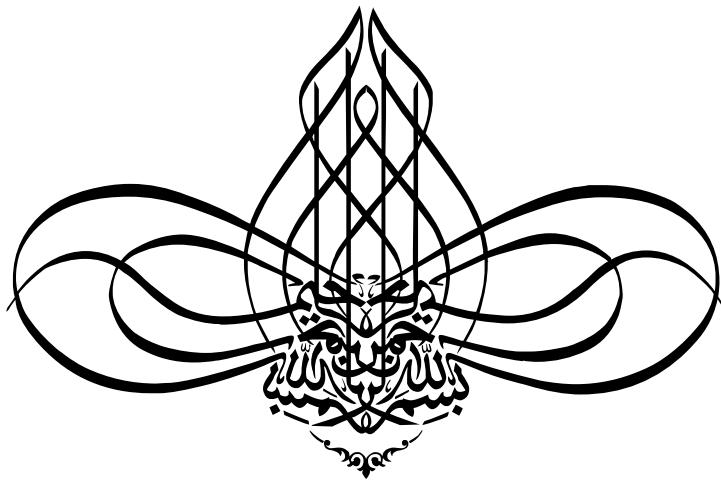


و این هم شماتیک مدار ماثول SD :



و این هم شماتیک مدار ماثول SD :





آموزش راه اندازی Micro SD به زبان C

و

استفاده از کتابخانه pff.h در CodeVision

تھیہ کنندہ : بسطام بیرامی

شرکت مهندسی الکترونیک ایرانیان تراشه اندیش (ITACo.)

با همکاری اعضای انجمن علمی وبسایت ECA.ir

مقدمه :

به دلیل پایین بودن میزان حافظه داخلی میکرو کنترلر ها ، در پروژه هایی که نیاز به فضایی برای ذخیره اطلاعات است از کارت های حافظه استفاده می شود.



کارت چیست ؟ SD

Secure Digital به اختصار SD ، کارت حافظه غیر فرار است که در بسیاری از وسایل قابل حمل مانند موبایل ، دوربین ها دیجیتال و تبلت ها از آن استفاده می شود.

استاندارد Secure Digital در سال 1999 به عنوان تکولوژی مکمل (MMC) Multi Media Cards (MMC) ها معرفی شد.

Secure Digital ها از نظر حجم حافظه به چهار دسته زیر تقسیم می شوند :

.1 SDSC (Secure Digital Standard Capacity) .(Secure Digital کارت هاییست که حجم استاندارد دارند)

SD کارت هایی با حجم 1MB تا 2GB و گاهی تا 4GB در این گروه قرار میگیرند.

.2 SDHC (Secure Digital High Capacity) .(Secure Digital کارت هاییست که حجم بالایی دارند)

SD کارت هایی با حجم GB4 تا 32GB در این گروه قرار میگیرند.

.3 SDXC (Secure Digital eXtended Capacity) .(Secure Digital کارت هاییست که حجم بسیار بالایی دارند)

SD کارت هایی با حجم GB32 تا 2TB در این گروه قرار میگیرند.

.4 SDIO (Secure Digital Input Output) .(Secure Digital کارت هاییست که ترکیبی از توابع ورودی ، خروجی را پشتیبانی می کند .

و با توجه به اندازه SD کارت ها میتوان به سه گروه زیر تقسیم بندی کرد :

- سایز اصلی

Standard: 32.0×24.0×2.1 mm (1.260×0.945×0.083 in)

Standard: ~ 2 g

- مینی

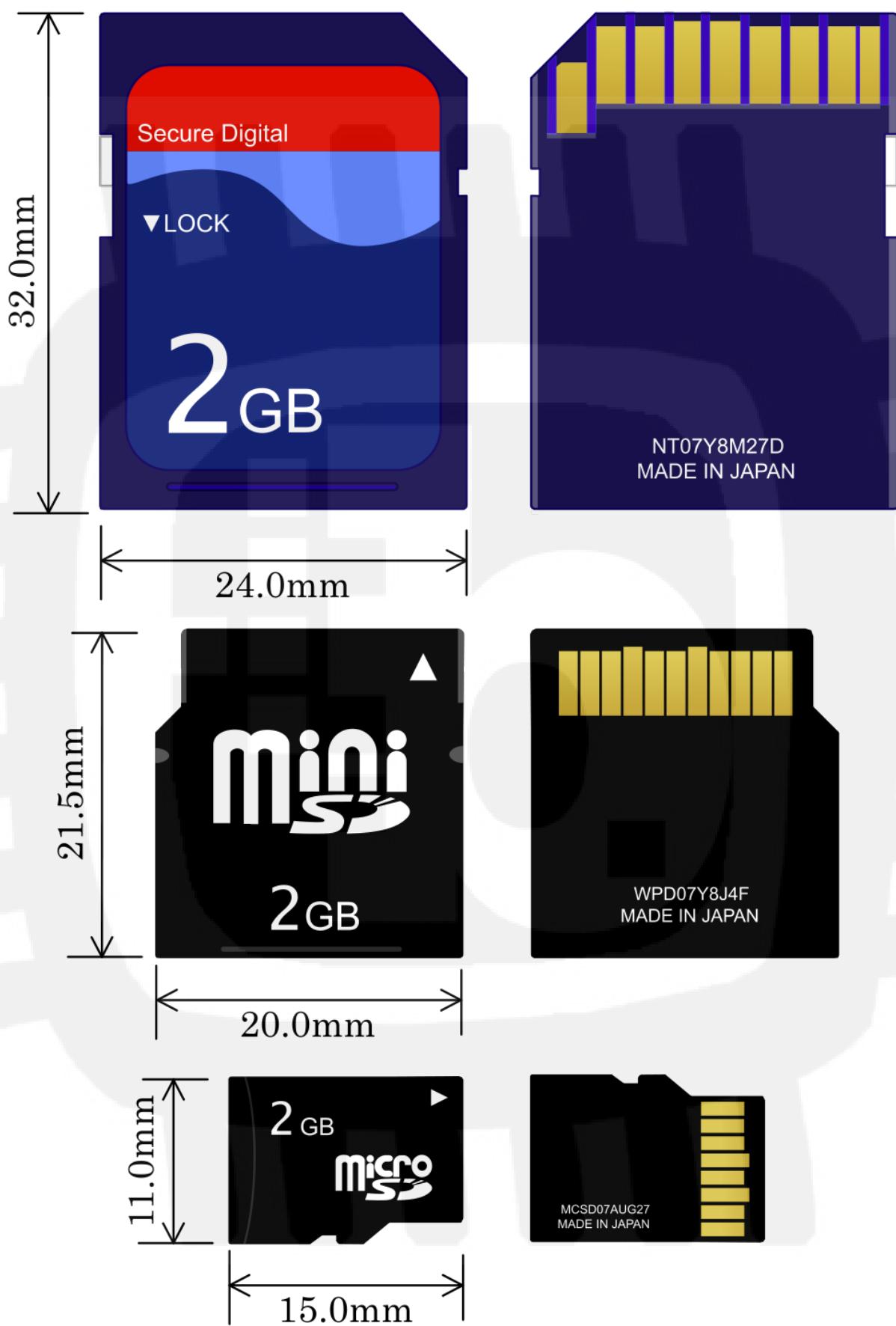
Mini: 21.5×20.0×1.4 mm (0.846×0.787×0.055 in)

Mini: ~ 0.8 g

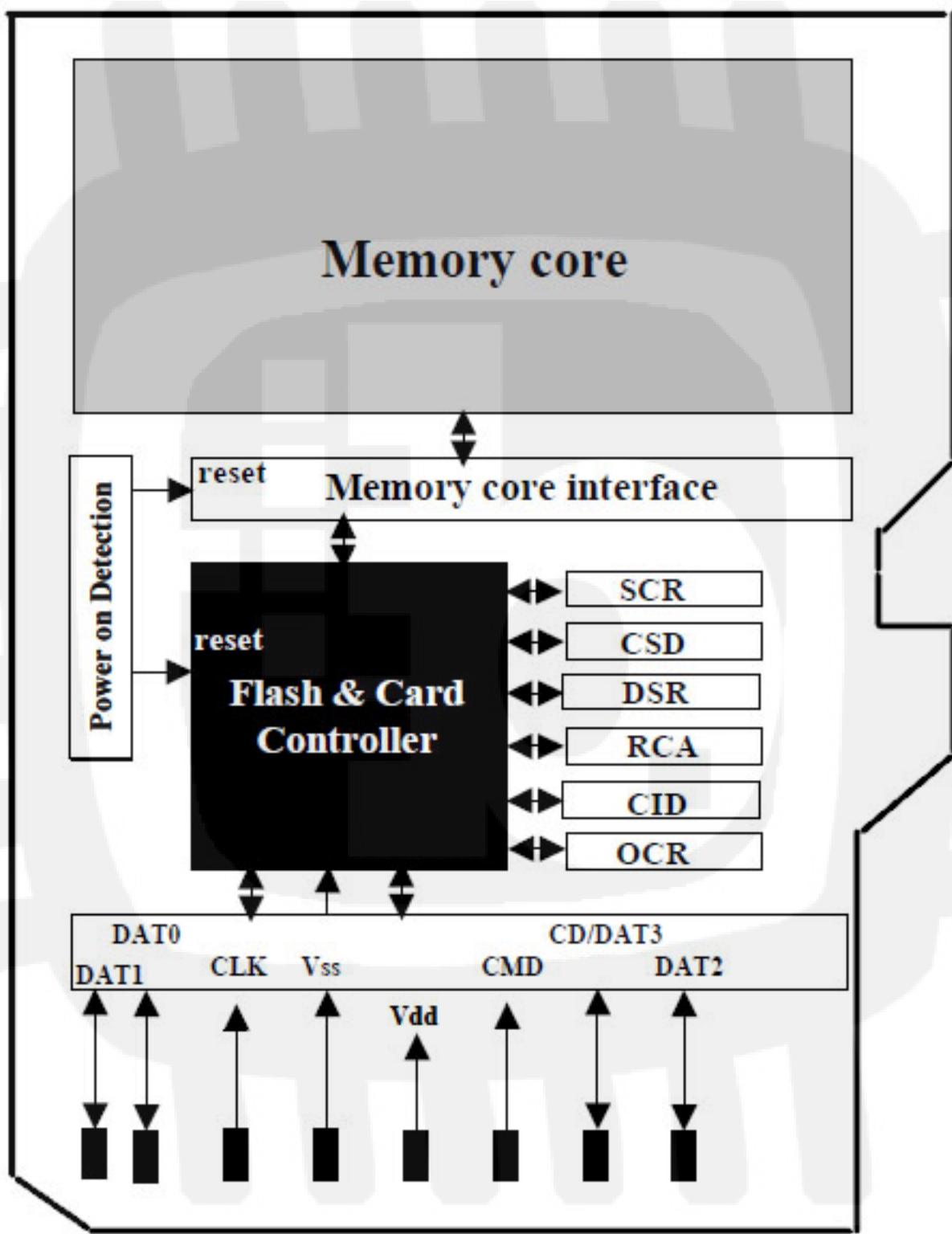
- میکرو

Micro: 15.0×11.0×1.0 mm (0.591×0.433×0.039 in)

Micro: ~ 0.25 g



بلوک دیاگرام مدار داخلی Micro SD



ویژگی های عمومی Micro SD کارت ها :

- SD-protocol compatible
- Supports SPI Mode
- Targeted for portable and stationary applications for secured (content protected) and unsecured data storage
- Voltage range of 2.7 to 3.6V
- Variable clock rate 0-25 MHz (standard), 0-50 MHz (high performance)
- Up to 25 MB/sec data transfer rate (using four parallel data lines)
- Memory field error correction
- Content protection mechanism that complies with highest security of SDMI standard
- Password protection
- Write-protected using mechanical switch
- Built-in write protection features (permanent and temporary)
- Supports card detection (insertion and removal)
- Application-specific commands

Parameter	min	typ	max	unit
Extended Operating Temperature	-25	25	85	°C
Power Supply VCC (3.3V)	2.7	3.3	3.6	V

Current Consumption (typ)	typ	max	Unit
Write	60	70	mA
Read	50	60	
Sleep Mode	0.15	2	

Parameter	Operating	Non Operating
Humidity (non-condensing)	operation: 95% RH @25°C storage: 93% RH @40°C, 500h	
EMC / EMI	Non Contact Pads area: ±8 kV (air discharge) Human body model according to IEC61000-4-2	
	Contact Pads: ±4 kV, Human body model according to IEC61000-4-2	

Parameter	Operating	Non Operating
UV light exposure	UV: 254nm, 15Ws/cm ² according to ISO7816-1	
Durability	10,000 mating cycles	
Drop test	1.5m free fall	
Bending / Torque	10N / 0.10Nm ±2.5° max	

Outer Physical Dimensions	Value	Unit
Length	15.0±0.1	mm
Width	11.0±0.1	
Thickness	0.7 (1.0)±0.1	
Weight (typ.)	0.4	g

Parameter	Value
Data Retention @ 25°C	10 years (JEDEC47G)

پارامترهای DC در Micro SD

Symbol	Parameter	min	typ	max	unit	notes
	Peak Voltage on all Lines	-0.3		VDD+0.3	V	
VIL	Input LOW Voltage	-0.3		0.25*VDD	V	
VIH	Input HIGH Voltage	0.625*VDD		VDD+0.3	V	
VOL	Output LOW Voltage			0.125*VDD	V	at 100µA
VOH	Output HIGH Voltage	0.75*VDD			V	at 100µA
IDD	Operating Current		35	50	mA	
	Pre-initialization Standby Current			3	mA	
	Post-initialization Standby Current		100	200	µA	
ILI	Input Leakage Current	-10		10	µA	without pull up R
ILO	Output Leakage Current	-10		10	µA	

Symbol	Parameter	min	typ	max	unit
VDD	Normal Operating Status	2.7		3.6	V
	Basic Communication (CMD0, CMD15, CMD55, ACMD41)	2.0	3.3	3.6	V
-	Power Up Time (from 0V to VDD min)			250	ms

Timing	Maximum Value
Block Read Access Time	100 ms
Block Write Access Time	250 ms
ACMD1 to ready after power-up	500 ms

7.3 Signal Loading

The total capacitance C_L is the sum of the bus master capacitance C_{HOST} , the bus capacitance C_{BUS} , and the capacitance C_{CARD} of the card connected to the line:

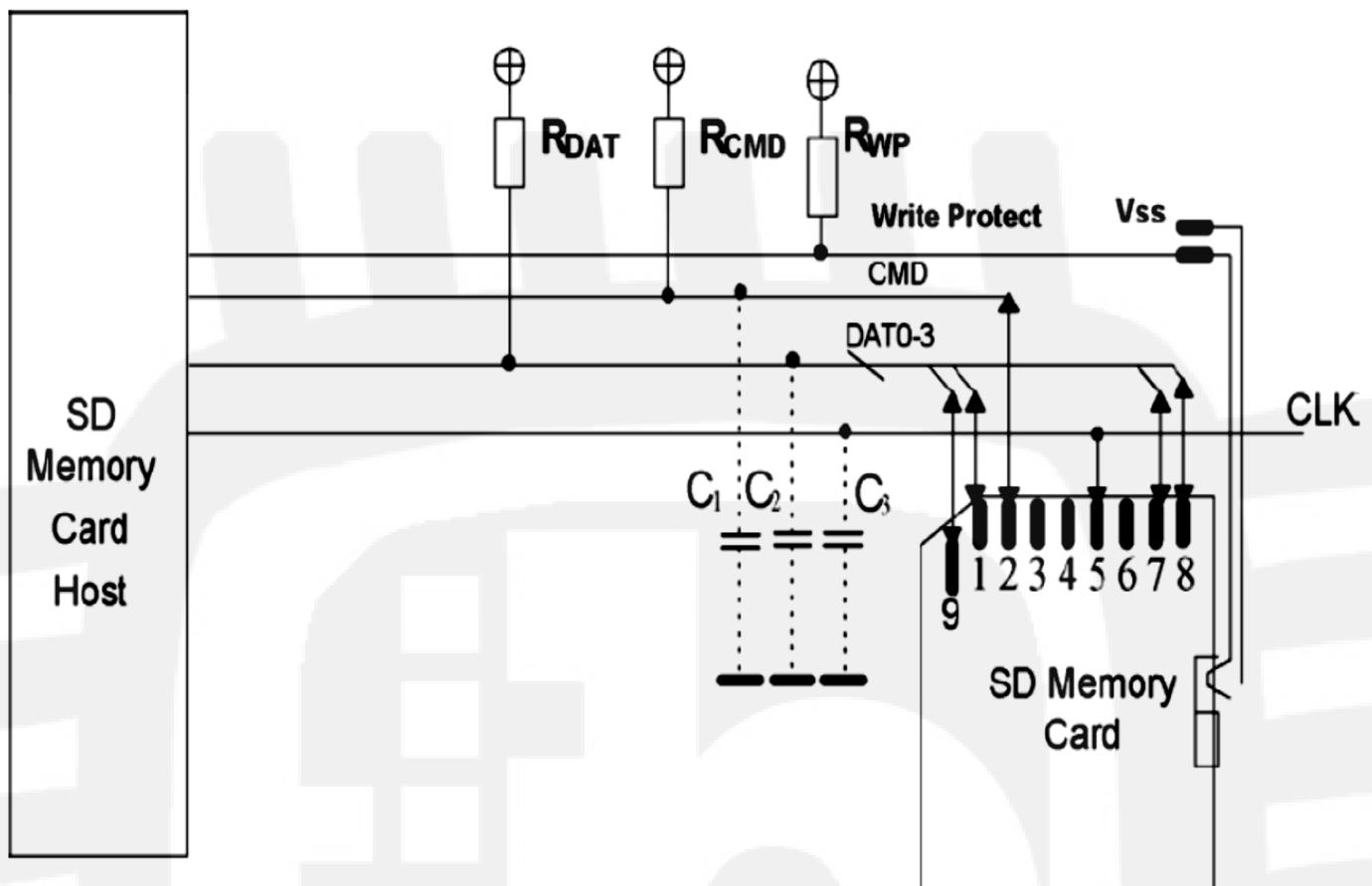
$$C_L = C_{HOST} + C_{BUS} + C_{CARD}$$

To allow the sum of the host and bus capacitances to be up to 20pF for the card, the following conditions in the table below are met by the card.

Table 15: Signal loading

Parameter	Symbol	Min	Max	Unit	Notes
Pull up resistance	R_{CMD}	10	100	kOhm	To prevent bus floating
Pull up resistance	R_{DAT}	10	100	kOhm	To prevent bus floating
Bus signal line capacitance	C_L		40	pF	Single card
Signal card capacitance	C_{card}		10	pF	Single card
Signal line inductance			16	nH	f≤20MHz

Figure 5: Signal Loading (MICRO SD Memory Card has no WP slider)



پارامترهای AC در Micro SD

Parameter	Symbol	Min	Max	Unit	Notes
Clock frequency in data transfer mode	f _{PP}	0	25	MHz	CL≤100pF
Clock frequency in card id mode	f _{OD}	0	400	KHz	CL≤250pF
Clock low time	t _{WL}	10/50		ns	
Clock high time	t _{WH}	10/50		ns	
Clock rise time	t _{TLH}		10/50	ns	CL≤100/250pF
Clock fall time	t _{THL}		10/50	ns	
CMD, DAT input setup time	t _{ISU}	5		ns	
CMD, DAT input hold time	t _{IH}	5		ns	CL≤25pF
CMD, DAT output delay time	t _{ODLY}	0	14	ns	CL≤25pF, data transfer
CMD, DAT output delay time	t _{ODLY}	0	50	ns	CL≤25pF, identification

Notes

1. Rise and fall times are measured from 10% to 90% of voltage level.
2. CLK referenced to VIH min and VIL max.
3. CMD and DAT inputs and outputs referenced to CLK.
4. 0Hz means to stop the clock. The given minimum frequency range is for cases where a continuous clock is required
5. Specified for one card

Figure 6: AC Characteristics Low Speed Mode

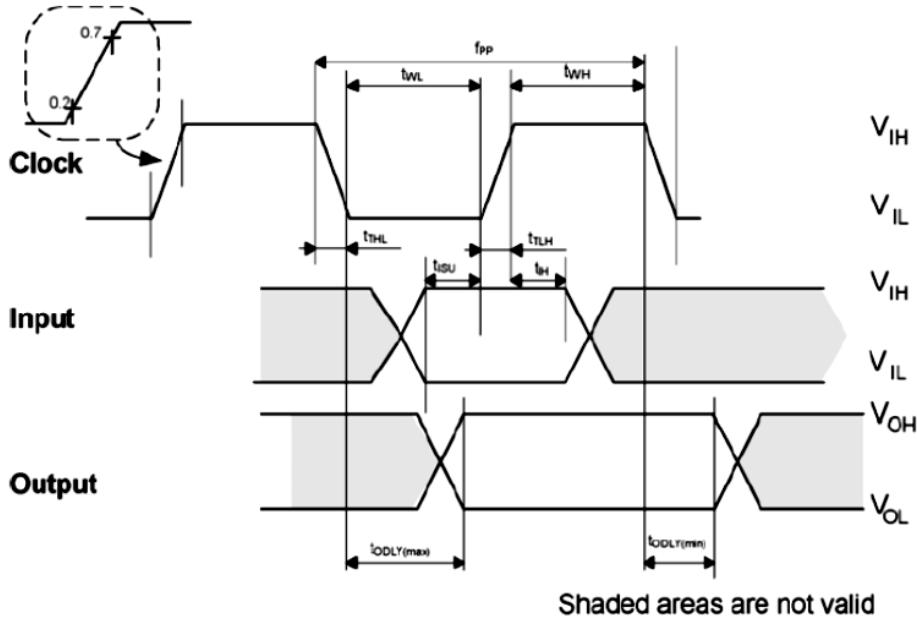


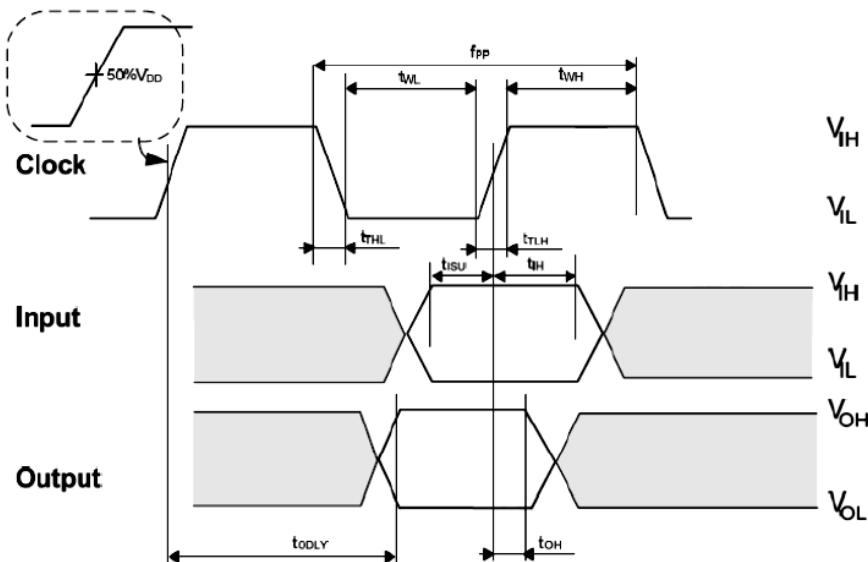
Table 17: AC Characteristics High Speed Mode

Parameter	Symbol	Min	Max	Unit	Notes
Clock frequency in data transfer mode	f_{PP}	0	50	MHz	$CL \leq 10\text{pF}$
Clock low time	t_{WL}	7.0		ns	
Clock high time	t_{WH}	7.0		ns	
Clock rise time	t_{TLH}		3	ns	
Clock fall time	t_{THL}		3	ns	
CMD, DAT input setup time	t_{ISU}	6		ns	
CMD, DAT input hold time	t_{IH}	2		ns	
CMD, DAT output delay time during data transfer mode	t_{ODLY}		14	ns	
CMD, DAT output hold time	t_{OH}	2.5		ns	

Notes

1. Rise and fall times are measured from 10% to 90% of voltage level.
2. CLK referenced to V_{IH} min and V_{IL} max.
3. CMD and DAT inputs and outputs referenced to CLK.
4. In order to satisfy severe timing, the host shall drive only one card with max 40pF total at each line.

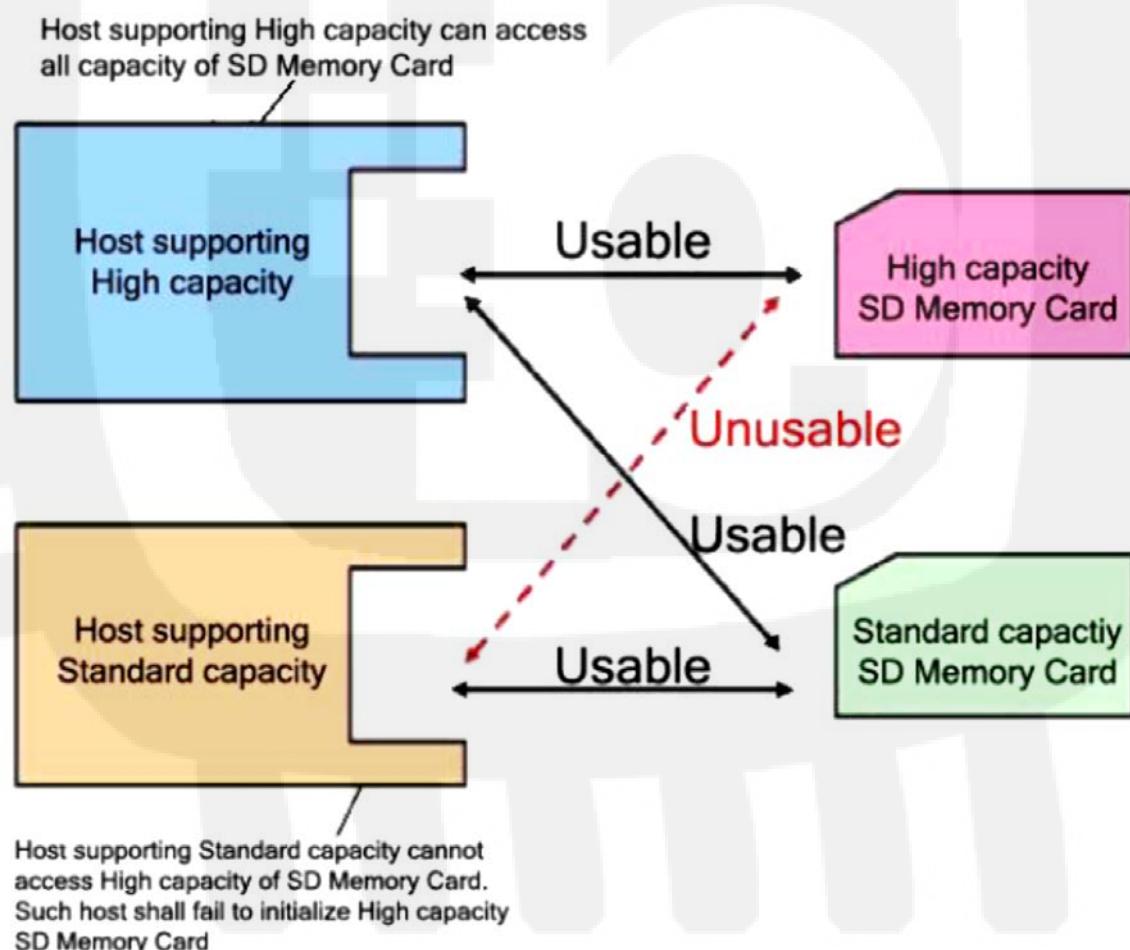
Figure 7: AC Characteristics High Speed Mode



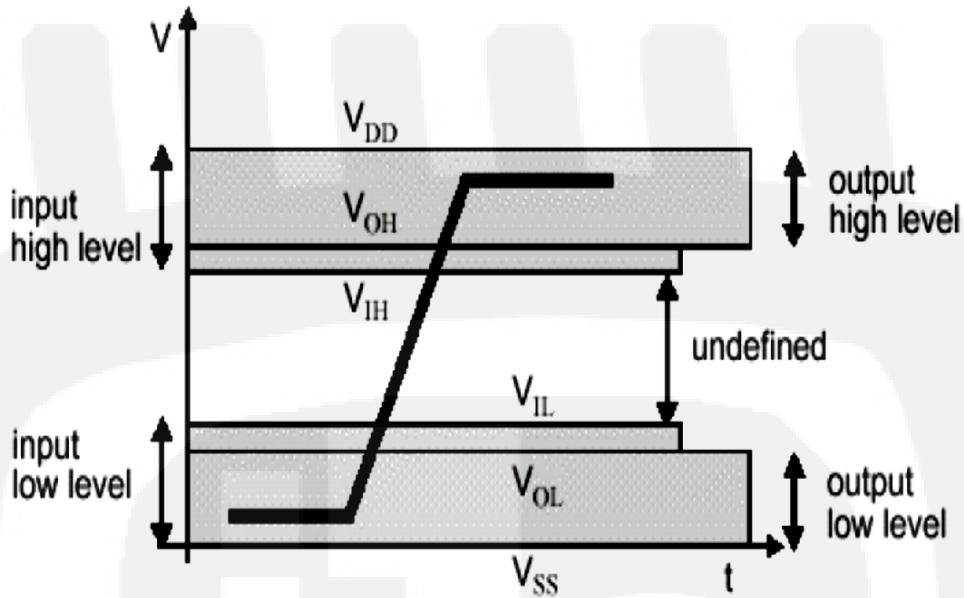
ریجیسترهای Micro SD

Name	Width	Description
CID	128	Card identification number: individual card number for identification.
RCA	16	Relative card address: local system address of a card dynamically suggested by the card and approved by the host during initialization
CSD	128	Card specific data: information about the card operation conditions.
SCR	64	SD Configuration Register: information about the Micro SD Card's special feature capabilities.
OCR	32	Operation Condition Register

نحوه استفاده از خانواده های مختلف کارت های SD در میزبان های متفاوت



سطوح ولتاژ در خطوط انتقال Data



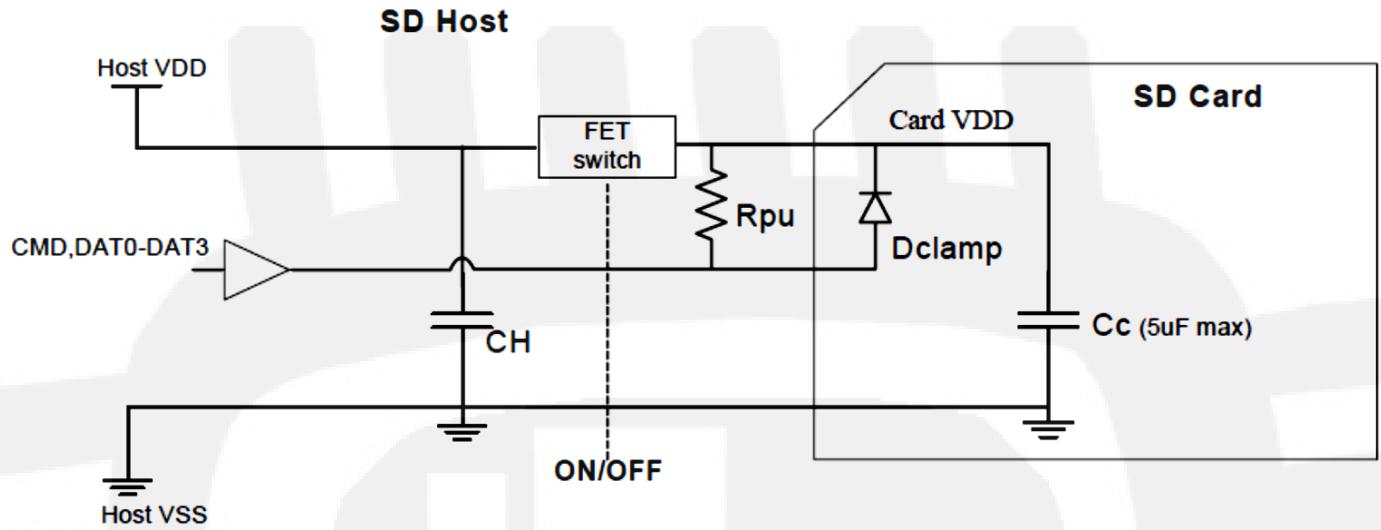
To meet the requirements of the JEDEC specification JESD8-1A, the card input and output voltages shall be within the following specified ranges for any V_{DD} of the allowed voltage range:

Parameter	Symbol	Min.	Max.	Unit	Remark
Output HIGH voltage	V_{OH}	$0.75 * V_{DD}$		V	$I_{OH} = -100 \mu A @ V_{DD} \text{ min}$
Output LOW voltage	V_{OL}		$0.125 * V_{DD}$	V	$I_{OL} = 100 \mu A @ V_{DD} \text{ min}$
Input HIGH voltage	V_{IH}	$0.625 * V_{DD}$	$V_{DD} + 0.3$	V	
Input LOW voltage	V_{IL}	$V_{SS} - 0.3$	$0.25 * V_{DD}$	V	

مدار واسط الکترونیکی

قبل از آن که هر کدام از پایه های انتقال دیتا (I/O Pins) مقدار یک منطقی (High Logic) را به خود بگیرند **باید** ولتاژ V_{DD} به مدار اعمال شود ، به عبارت دیگر ، پایه های 3-Data , CLK , CMD **باید** در زمان اتصال ولتاژ به پایه V_{DD} همگی برابر صفر ولت باشند.

مدار پیشنهادی برای کنترل تغذیه Micro SD



توجه مهم : به دلیل وجود دیود کلمپ (Dclamp) بر روی خطوط CMD , CLK , Data0-3 بسیار مهم است که اطمینان حاصل کنید تا قبل از اتصال ولتاژ V_{DD} پایه های CMD , CLK , Data0-3 حتماً صفر باشند.

چنانچه هر یک از پایه های نام بردہ قبل از زمان اتصال V_{DD} و یا قبل از به پایان رسیدن زمان تاخیر شروع بکار (مقداری بیش از ۰ داشته باشند میتوانند دیود های کلمپ را بایاس مستقیم (Forward Bias) کرده و SD را به وضعیت نامعلومی ببرد.

It is the host's responsibility to make sure power gets to VDD before CMD, CLK, or DAT0-3 go above zero volts.

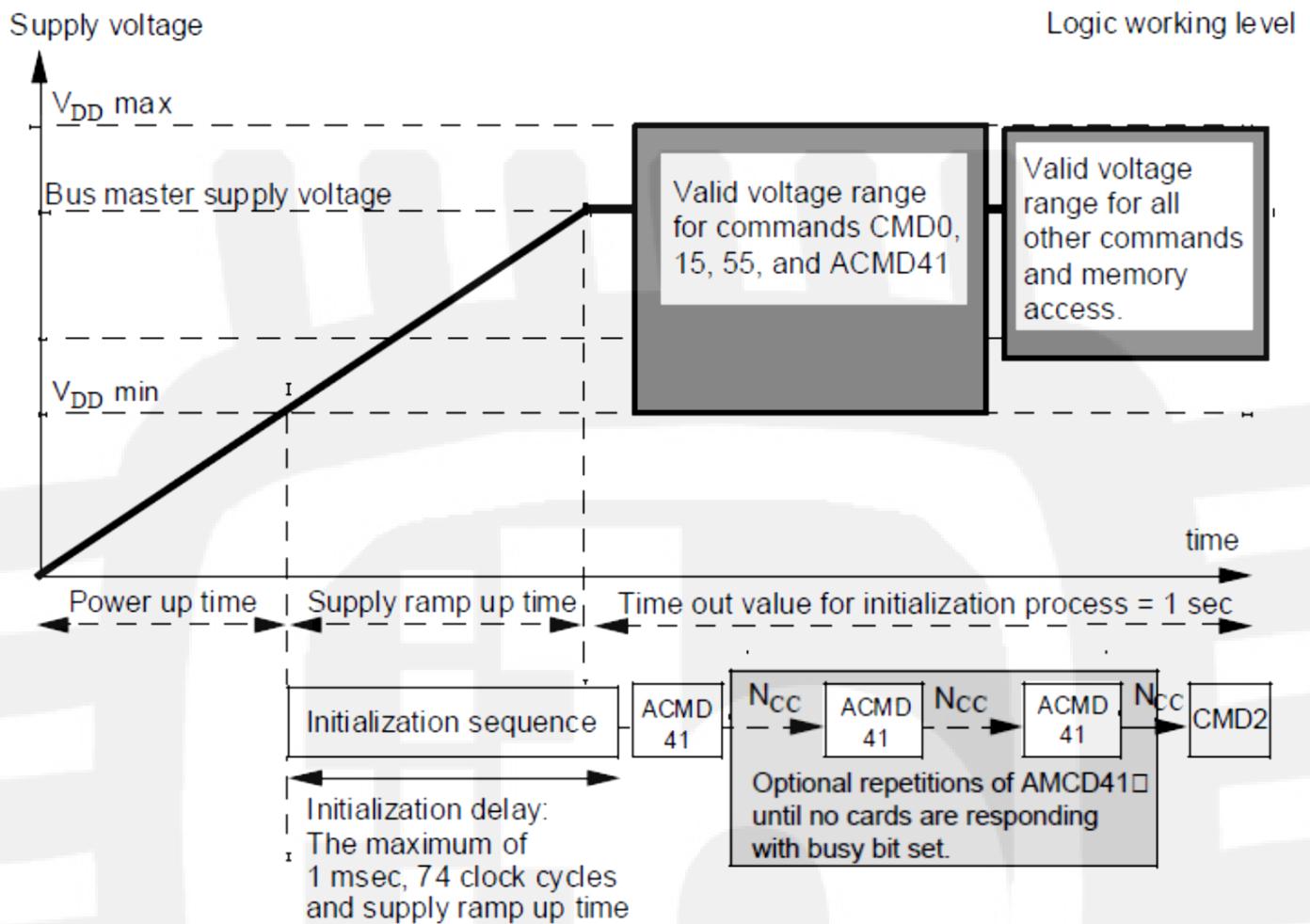


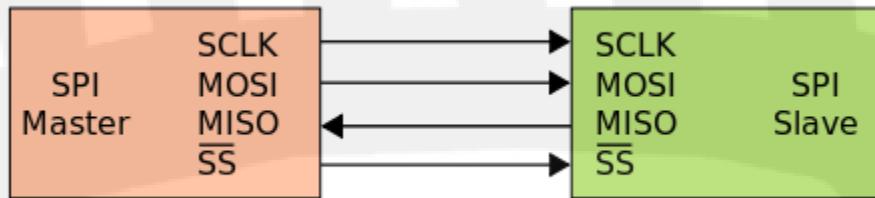
Figure 3-5. Power-up Diagram

Power Supply Voltage					
Parameter	Symbol	Min.	Max.	Unit	Remark
Supply Voltage	V_{DD}	2.0	3.6	V	CMD0, 15, 55, ACMD41 commands
Supply Voltage		2.7	3.6	V	Except CMD0, 15, 55, ACMD41 commands
Supply voltage differentials (V_{SS1} , V_{SS2})		-0.3	0.3	V	
Power up Time			250	μS	From 0V to V_{DD} Min.

مدهای انتقال داده در کارت های SD

کارت ها از مدهای زیر برای انتقال اطلاعات با پردازشگر استفاده می کنند :

- این مد مخفف عبارت Serial Peripheral Interface Bus است که در درجه اول در سیستم های Embedded استفاده می شود ، در این مد SD تنها با ولتاژ 3.3v کار می کند.



همانطور که مشاهده می کنید این ارتباط دارای چهار خط ارتباطی بین (Master (Micro Controller) و (Slave (SD Card) است.

کاربرد و نامهای دیگر این چهار خط ارتباطی به شرح زیر است :

The SPI bus specifies four logic signals:

SCLK : Serial Clock (output from master).

MOSI : Master Output, Slave Input (output from master).

MISO : Master Input, Slave Output (output from slave).

SS : Slave Select (active low, output from master).

Alternative naming conventions are also widely used:

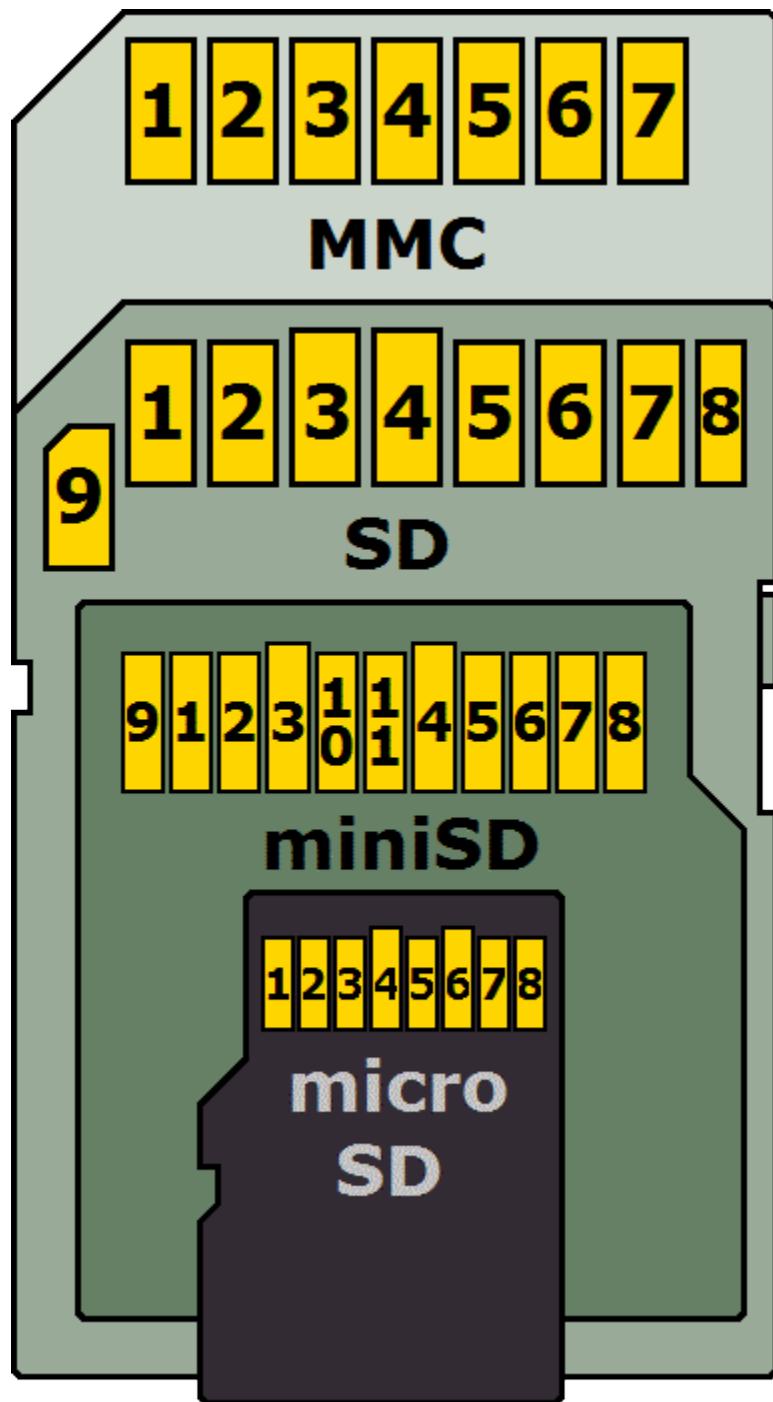
SCLK : SCK, CLK.

MOSI : SIMO, SDO, DO, DOUT, SO, MTSR.

MISO : SOMI, SDI, DI, DIN, SI, MRST.

SS : nCS, CS, CSB, CSN, nSS, STE, SYNC.

تشریح شماره پایه ها و عملکرد آنها در مد SPI Bus



SPI Bus Mode

MMC Pin	SD Pin	miniSD Pin	microSD Pin	Name	I/O	Logic	Description
1	1	1	2	nCS	I	PP	SPI Card Select [CS] (Negative Logic)
2	2	2	3	DI	I	PP	SPI Serial Data In [MOSI]
3	3	3		VSS	S	S	Ground
4	4	4	4	VDD	S	S	Power
5	5	5	5	CLK	I	PP	SPI Serial Clock [SCLK]
6	6	6	6	VSS	S	S	Ground
7	7	7	7	DO	O	PP	SPI Serial Data Out [MISO]
	8	8	8	NC nIRQ	. O	. OD	Unused (memory cards) Interrupt (SDIO cards) (Negative Logic)
	9	9	1	NC	.	.	Unused
		10		NC	.	.	Reserved
		11		NC	.	.	Reserved

به دلیل این که ارتباط با SD کارت ها در پروژه ها فقط با مد SPI Bus Mode انجام می شود تنها به تشریح همین مد بسته می کنیم و مطالعه بیشتر را به عهده خواننده می گذاریم.

- **One-bit SD bus mode:** Separate command and data channels and a proprietary transfer format.
- **Four-bit SD bus mode:** Uses extra pins plus some reassigned pins. UHS-I and UHS-II requires this bus type.

File System

همانند دیگر حافظه های فلش ، SD کارت و تمامی خانواده های ذخیره سازی بلوکی با قابلیت آدرس دهی می باشند که در آنها میزبان (Host Device) می تواند اطلاعات را به صورت بلوک هایی با حجم مشخص (Fixed-Size Blocks) و تعیین شماره بلوک بخواند و بنویسد.

MBR and FAT (Master Boot Record & File Allocation Table)

چیپ اکثر SD کارت ها از قبل با یک یا چند MBR فرمت شده اند که در آن اولین یا نتها پارتیشن SD دارای فایل سیستم است که با SD اجازه می دهد همانند هارد دیسک کامپیوتر عمل کند.

هر SD کارت بر اساس مشخصاتش با یکی از MBR های زیر فرمت می شود :

- For SDSC cards:
 - Capacity of less than 32680 logical sectors (smaller than 16 MB): FAT12 with partition type 01h and BPB 3.0 or EBPB 4.1
 - Capacity of 32680 to 65535 logical sectors (between 16 MB and 32 MB): FAT16 with partition type 04h and BPB 3.0 or EBPB 4.1
 - Capacity of at least 65536 logical sectors (larger than 32 MB): FAT16B with partition type 06h and EBPB 4.1
- For SDHC cards:
 - Capacity of less than 16450560 logical sectors (smaller than 7.8 GB): FAT32 with partition type 0Bh and EBPB 7.1
 - Capacity of at least 16450560 logical sectors (larger than 7.8 GB): FAT32X with partition type 0Ch and EBPB 7.1
- For SDXC cards: exFAT with partition type 07h

نقل قول از دوست عزیزم محمد مهدی صفوی در وب سایت ECA.ir

اطلاعات پایه

مفهوم بیت همان 0 و 1 است که در منطق دیجیتال و بسیاری از جاهای استفاده می شود. اگر 4 بیت را گناه هم بپینیم یک نیبل ساخته ایم

حال اگر دو نیبل را در گناه هم بگذاریم که معادل 8 بیت می شود یک بایت ساخته ایم که میتواند 8 بیت را به فرد اختصاص دهد و هر بیتی میتواند 0 یا 1 باشد.

حال اگر دو بایت را گناه هم بگذاریم که 16 بیت ایجاد کند یک WORD ساخته ایم.

در محفظه های محمولی مثل محافظه 2404 که یک EEPROM سریال است، دارای فانه های محفظه ای هست که هر فانه 8 بیت یعنی یک بایت را در خود جای میدهد و برای فواید این فانه ها نیاز به آدرس دهنده است. طول آدرس نسبت به مجموع محفظه ای آن مشخص می شود.

در محفظه های SD و MMC دو اصطلاح به نامهای سکتور (Sector) و کلاستر (Cluster) وجود دارد.

سکتورها بخش های کوچک تری از کلاسترها هستند و بایت نیز بخش کوچک تری از سکتورها مخصوص می شوند.

هر سکتور معمولاً شامل 512 بایت است و هر کلاستر بسته به نوع مدیریت فایل (نوع فرمت FAT) دارای چندین سکتور داخل فود است و ملاک آدرس دهنده در SD یا MMC همان کلاسترها هستند.

نحوه ذخیره فایل (وی MMC

وقتی فایلی داخل SD یا MMC ذخیره می شود فایل در اولین کلاستر خالی که پیدا شود ذخیره می شود ادامه ای فایل در دومین کلاستر خالی که پیدا شود ذخیره می شود و به همین ترتیب تا آخر فایل ادامه میابد تا فایل کاملاً ذخیره شود و هیچ لزومی ندارد که کلاسترها خالی پشت سر هم باشد در این صورت بهترین استفاده از محفظه صورت گرفته و کلاستر کاملاً خالی وجود ندارد. اینکه تشفیض دهد کدام کلاستر خالی است تا اطلاعات را داخل آن بریزد به عهده ای کامپیوتر یا پردازنده می باشد و خودش تشفیض می دهد.

اگر فایلی از کلاستر کوچک باشد ناچار است که کل کلاستر را به آن اختصاص دهد و مابقی آن بدون استفاده میماند که این مقررین به صرفه و بهینه نیست که باید از جدول تخصیص فضای (FAT) استفاده شود.

حال ما می فواهیم فایل ذخیره شده را بفوانیم اما چون کلاسترها پشت سر هم ذخیره نشده اند ما نمی دانیم برای فایل مورد نظر کدامیک از کلاسترها باید فواید شود، برای این کار باید از جدول تخصیص فضای استفاده شود که نیاز است با مدیریت ذخیره فایل (وی محفظه آشنا شویم).

هر محفظه برای فواید نیاز به یک فایل سیستم دارد. فایل سیستم مسؤول نامگذاری و ذخیره و بازیابی اطلاعات در قالب یک فایل است.

به طور مثال فرض کنیم که هر کلاستر یک مموری 5 سکتور دارد و ما می فواهیم فایلی ذخیره کنیم که 7 سکتور را شامل می شود در کلاستر خالی اول 5 سکتور اول اطلاعات را ذخیره میکند و در کلاستر خالی دوم 2 سکتور باقیمانده را پر میکند و 3 سکتور باقی مانده بی مصرف میماند. برای اینکه از محفظه بهینه استفاده شود باید از 3 سکتور باقیمانده هم استفاده شود و فایل بعدی

که میفواهد ذخیره شود از سکتورهای خالی این کلاستر هم استفاده کند بنابر این از "فایل سیستم" استفاده می شود. فایل سیستم آدرس شروع و پایان کلاستر و تعداد کلاسترها ذخیره شده را در جدول به نام FAT ذخیره می کند.

FAT جدول

FAT مخفف کلمه‌ی File Allocation Table می باشد و به معنی جدول تخصیص فضای فایل است. این جدول در کلاسترها اول حافظه توسط سیستم عامل ایجاد و نگهداری می شود و برای مدیریت ذخیره سازی فایلها مورد استفاده قرار می گیرد.

وظیفه ای این جدول نگهداری فضاهای موجود در حافظه است، فضاهای موجود می توانند فضاهای خراب و اشغال شده و خالی باشند.

وقتی میفواهیم فایلی را در SD یا MMS بریزیم ابتدا سیستم با جدول FAT مشورت کرده که کدام کلاسترها خالی هستند و اطلاعات را در کدام کلاسترها ذخیره کند و در موقع فوایدن هم از جدول FAT می خواهد تا اعلام کند کدام کلاسترها مربوط به فایل است و باید بفوازند.

در این مبحث فقط دو نوع فرمات FAT بروزی می شوند اما فرمات های موجود به اختصار در زیر آورده شده اند.

FAT12 (12-bit version)

FAT16/FAT16B/FAT16X (16-bit versions)

FAT32/FAT32X (32-bit version with 28 bits used)

: FAT16

عددهای جلوی FAT نشانگر تعداد بیت‌های آدرس کلاسترها می باشد. مثلاً FAT16 دارای 16 بیت آدرس برای کلاسترها یعنی $(16^2) = 65536$ کلاستر در این فرمات FAT وجود دارد. حال باید بینیم برای FAT16 اندازه‌ی هر کلاستر په مقداری (په تعداد سکتور) تعیین شده است.

چون 16 نمی‌تواند به اندازه FAT32 اطلاعات ذخیره کند مجبور است اندازه هر کلاستر را بزرگ انتخاب کند. اما هر په اندازه‌ی کلاستر کوچکتر باشد حافظه بهینه تر استفاده می شود زیرا فضای خالی کمتری بین فایلها باقی می‌ماند.

برای فرمت FAT16 اندازه هر کلاستر 64 Kbyte است. یعنی هر کلاستر دارای 128 سکتور میباشد. هر سکتور 0.5 Kbyte است. ($0.5 = 0.5 * 128$)

حال میفواهیم مهاسبه کنیم که FAT16 تا چه مقدار هم بتواند پشتیبانی کند :

$$\text{تعداد کل کلاسترها} = 65536 = 16^2$$

$$\text{مجموع هر کلاستر} = \text{Kbyte 64}$$

$$\text{تعداد سکتورهای هر کلاستر} = 128$$

$$\text{تعداد کل سکتورهای موجود در FAT16} = 65536 * 128 = 8388608$$

$$\text{تعداد بایت هر سکتور} = \text{Byte 512}$$

$$\text{تعداد بایت کل} = 512 * 8388608$$

$$\text{تعداد بایتهای یک گیگ} = 2^{30} = 1073741824$$

$$\text{مجموع حافظه 16 FAT بر مسرب گیگ} = 4\text{GByte}$$

پس حافظه با فرمت FAT16 تا 4GB (4294967296 / 1073741824) بایت است. این تفاوت را آدرس دهی می کند، یعنی دارای 4294967296 کلاستر می باشد و مجموع کلاسترها آن دیگر 64 نیست بلکه برای کارامد شدن مقدار آن کمتر شده است.

FAT32

این فرمت هم یک فایل سیستم مانند FAT16 است با این تفاوت که تا 32 بیت را آدرس دهی می کند، یعنی دارای 4294967296 کلاستر می باشد و مجموع کلاسترها آن دیگر 64 نیست بلکه برای کارامد شدن مقدار آن کمتر شده است.

: NTFS

این سیستم فایل کلاسترها بیشتری نسبت به FAT32 را پشتیبانی میکند بنابراین کلاسترها یعنی گوچکتر بوده و کارآمدتر است.

هر کلاستر در این فرمت فایل سیستم 4 KB هستند.

تقسیم بندی فضای حافظه:

حافظه معمولاً به دو بخش اساسی تقسیم می شود:

1. نامیه سیستم

این نامیه کلاسترهاش شماره صفر ویک را شامل می شود. اولین قسمت حافظه، نامیه سیستم است.

جدول FAT داخل این نامیه است. سیستم عامل با استفاده از این جدول تعیین میکند:

- هر فایل چه کلاسترهايی را اشغال کرده است
- چه کلاسترهايی خالي است
- چه کلاسترهايی خراب است

وظیفه ای جدول FAT این است که هرگاه فایل ایجاد شود یا تغییر کند اطلاعات مربوط به وضعيت آن کلاسترها را درجدولش تغییر دهد تا موقع خواندن کمک کند که کدام کلاسترها مربوط به فایل مورد نظر است و باید خوانده شود.

2. نامیه داده

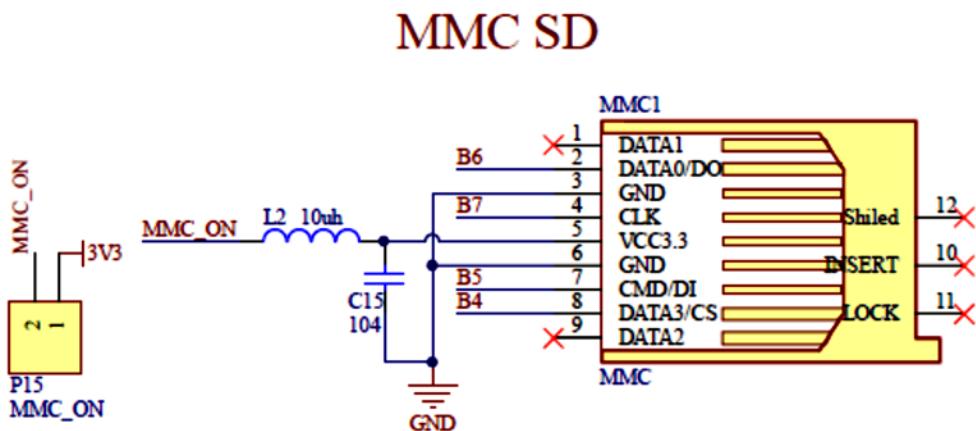
اطلاعاتی که داخل SD یا MMC میبایزیم از کلاستر دوم به بعد قرار می گیرند که با این نامیه، نامیه داده گفته می شود.

پایان نقل قول از دوست عزیزم محمد مهدی صفوی در وب سایت ECA.ir

سخت افزار مورد نیاز برای اتصال Micro SD به میکرو کنترلرها

همانطور که در مطالب بالا به آن اشاره شد ولتاژ کار SD کارت ها $3.6v \sim 2.7v$ است و برای عملیات خواندن و نوشتن در سرعت های متفاوت نیاز به جریان های مختلفی دارد ، در اینجا به دلیل پایین بودن سرعت حداقل جریان مصرفی SD کارت را $100ma$ در نظر می گیریم و بر اساس این دو اصل شماتیکی را برای ارتباط SD به میکرو کنترلر طراحی می نماییم.

شماتیک برای کار با میکرو هایی که ولتاژ کاری و I/O آن ها در رنج $3.6v \sim 2.7v$ است به صورت زیر می باشد :



شماتیک بر گرفته شده از [برید آموزشی حرفه ای میکرو کنترلر های AVR](#)

پایه های B4-B7 مربوط به پورت B در میکرو کنترلر ATMega32L می باشد.

(\overline{SS}) PB4 : این پایه به پایه CS یا همان (Chip Select) از SD متصل می شود ، وضیفه این پایه انتخاب SD کارت مورد نظر است و با 0 شدن این پایه فعال می شود ، حتما نیاز نیست از پایه PB4 برای این کار استفاده شود و هر پایه دیگری از I/O ها قابلیت انتخاب Slave فعال را دارد.

(MOSI) PB5 : این پایه Master Output-Slave Input است ، به این معنی که میکرو به عنوان Master دیتا را با این پایه ارسال می کند و SD به عنوان Slave دیتا را از این پایه دریافت می کند و به پایه DI به مفهوم Data Input از SD متصل می شود.

(MISO) PB6 : این پایه Master Input-Slave Output است ، به این معنی که میکرو به عنوان Master دیتا را از این پایه دریافت می کند و SD به عنوان Slave دیتا را با این پایه ارسال می کند و به پایه DO به مفهوم Data Output از SD متصل می شود.

(SCK) PB7 : این پایه Serial Clock است ، با این معنی که میکرو به عنوان Master پالس ساعت را با این پایه برای همزمان سازی (Synchrono) انتقال اطلاعات به SD می فرستد و به پایه CLK به معنی Clock از SD متصل می شود.

این شماتیک برای MMC طراحی شده است و می توان آن را برای انواع SD کارت ها از قبیل Micro SD استفاده کرد.

در زیر پایه های Micro SD و نحوه اتصال آن به SD Adaptor آورده شده است که می توان بر اساس نیاز در مدار از سوکت دلخواه استفاده نمود.

SD 8 to MicroSD 8 NC

SD 7 to MicroSD 7 DO

SD 6 to MicroSD 6 GND

SD 5 to MicroSD 5 CLK

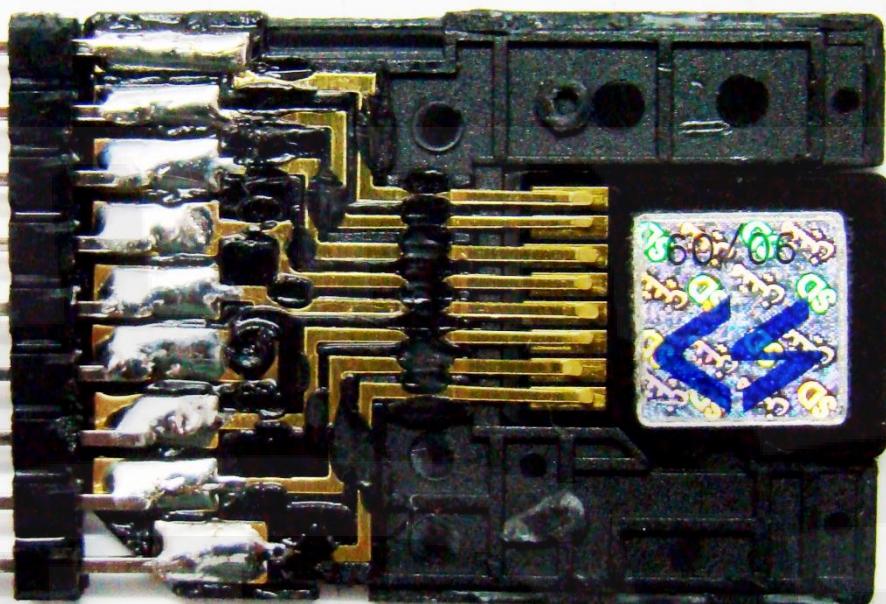
SD 4 to MicroSD 4 VDD

SD 3 not Connect to MicroSD NC

SD 2 to MicroSD 3 DI

SD 1 to MicroSD 8 nCS

SD 9 to MicroSD 1 NC



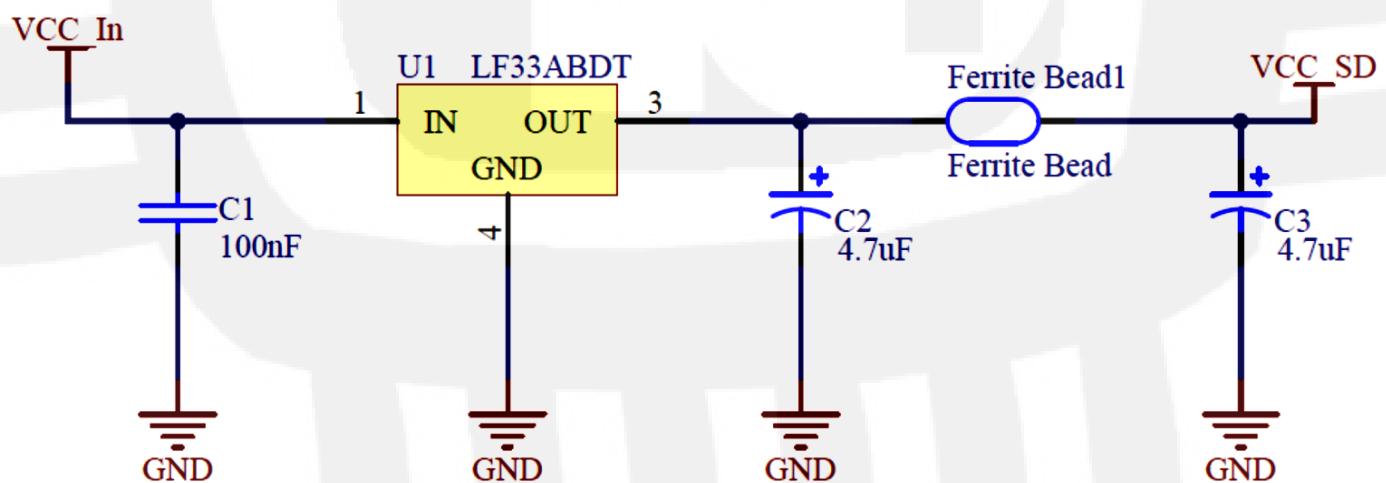
همانطور که ملاحظه فرمودید این شماتیک ساده ترین مداری است که برای کار با SD یا MMC می شود طراحی کرد.

در ادامه این مدار بهبود بخشدید شده و برای ارتباط با میکروکنترلرهایی که ولتاژ کاری آنها بیش از 3.3v است نیز تغییرات لازم داده می شود و امکاناتی از قبیل Chip Detect و Write Protect به آن افزوده می شود.

توجه: استفاده از تقسیم مقاومتی برای کاهش ولتاژ خطوط انتقال اطلاعات توسيه نمی شود و بهتر است از IC های Level Shifter مانند ADG3304 CD4050 و یا ADG3304 استفاده شود. (بهترین انتخاب ADG3304 است)

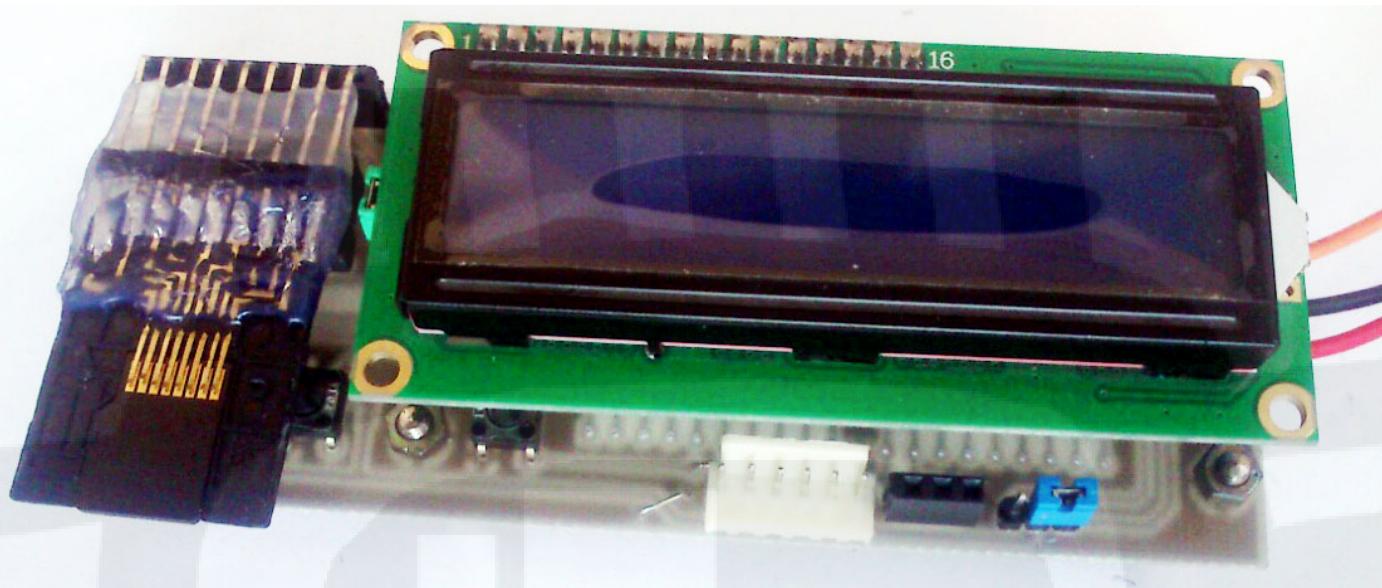
برای تغذیه مدارات SD کارت بهترین ولتاژ 3.3v است که توسيه می شود از ریگولاتور خطی و LDO به شماره LF33 استفاده شود.

مدار پیشنهادی برای این ریگولاتور در زیر آمده است :



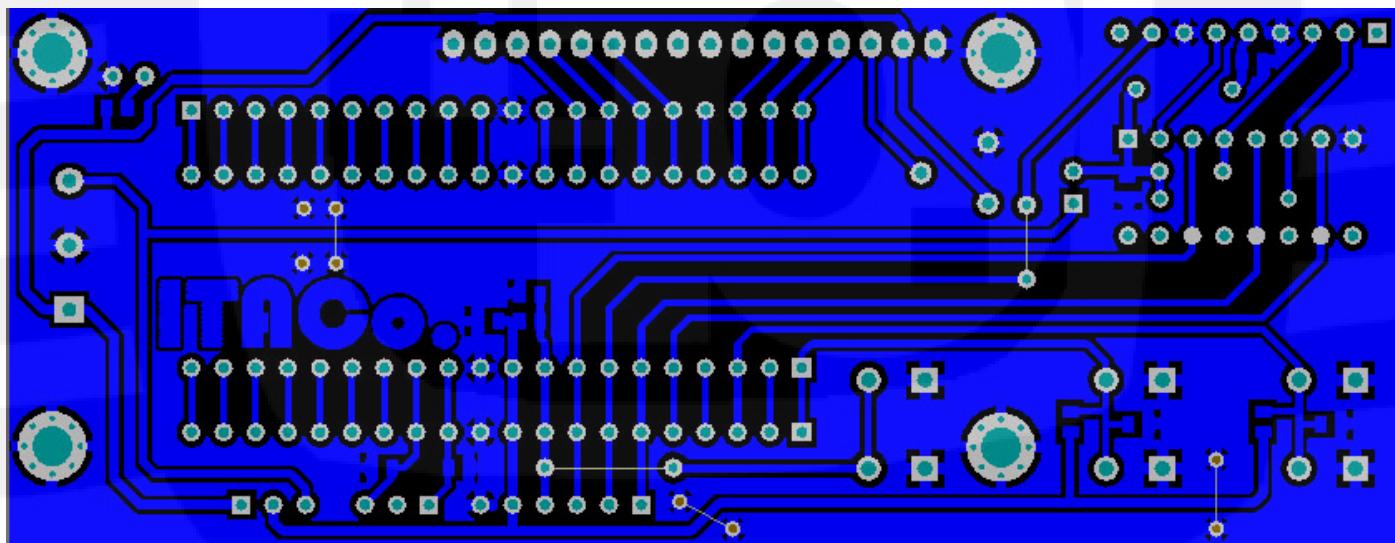
توجه: استفاده از دیود سری در ورودی برای کاهش ولتاژ توسيه نمی شود.

شماتیک نهایی و تست شده برای برد راه انداز Micro SD و Char LCD با ATMega32L

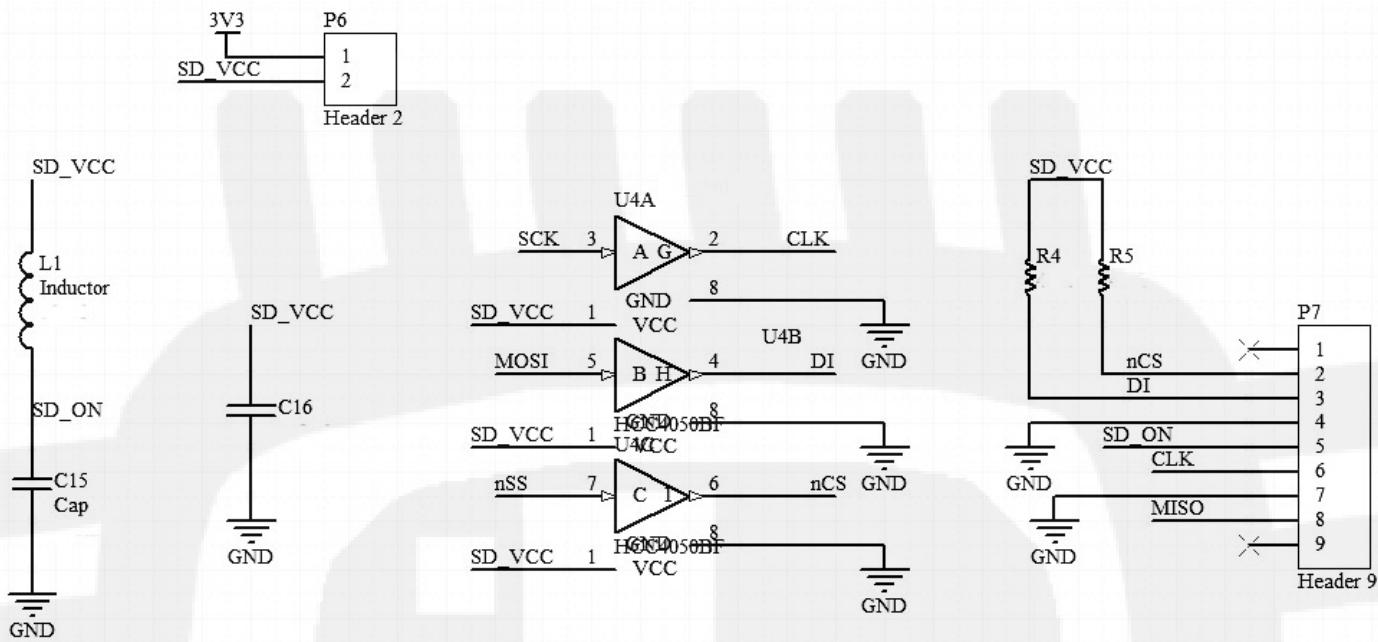


من شماتیک و بردی را برای تست و راه اندازی اولیه Micro SD طراحی کردم که قابلیت کار با ولتاژ 3.3 و 5 ولت را دارد و میتوان به صورت مستقیم و یا با IC HCF4050BE بافتر Micro SD کار کرد.

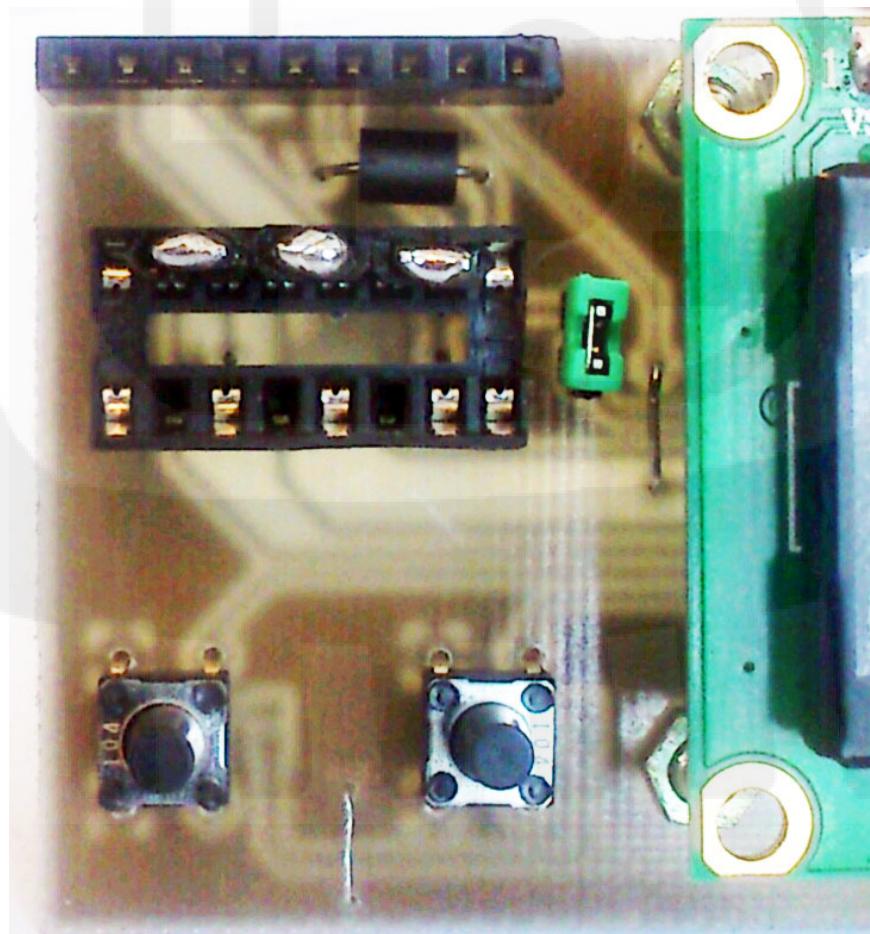
خودم برد رو در هم دو صورت تست کردم و عالی جواب گرفتم ، اطلاعات رو قرار میدم که دوستان هم بتوانن استفاده کنند.



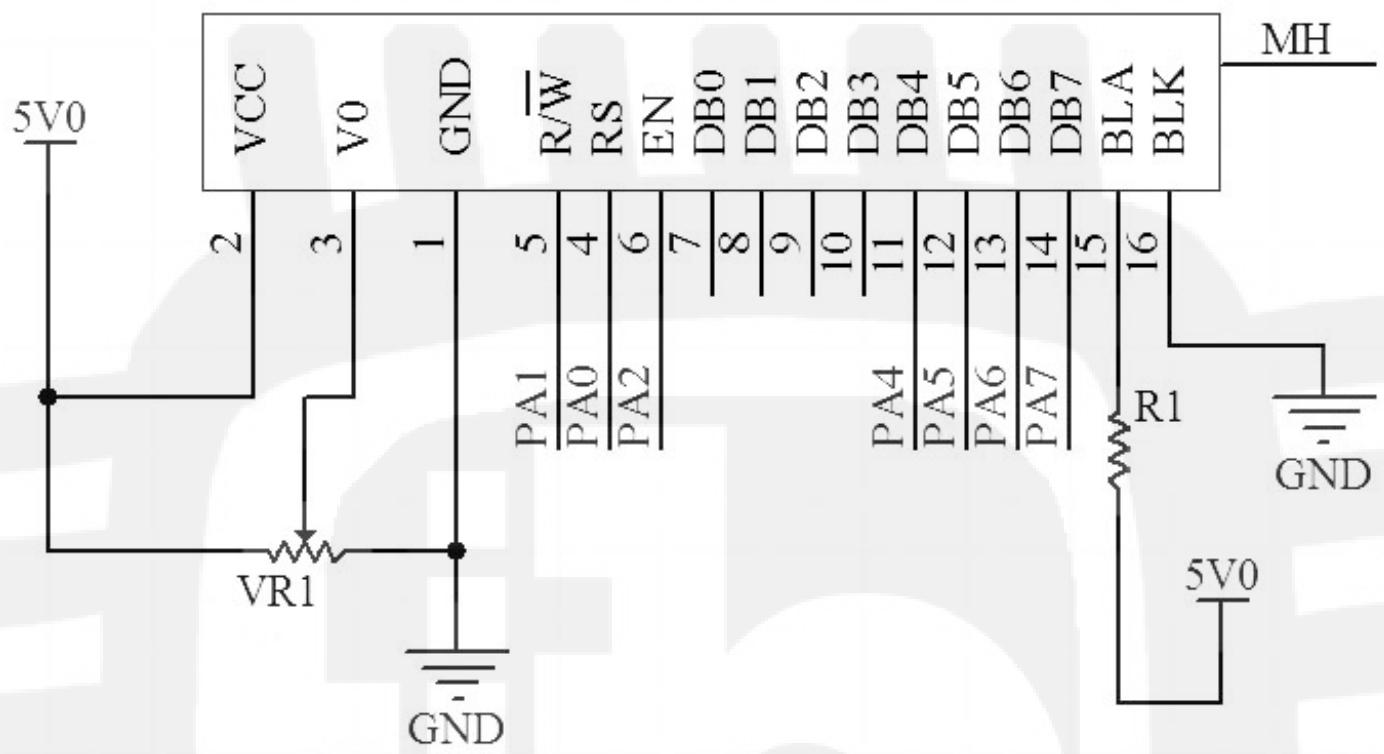
شماتیک مدار Micro SD



برای اتصال مستقیم Micro SD به میکرو کافیست به جای IC بافر HCF4050 پایه های 2 و 3 ، 4 و 5 ، 6 و 7 را با جامپر به هم وصل کنید و تغذیه میکرو کنترلر را 3.3 ولت قرار دهید که بر روی برد جامپری برای تغییر تغذیه تعیین شده است.



LCD1



آموزش کتابخانه pff.h کدویژن

نقل قول از دوست عزیزم سوران آراسته در وب سایت ECA.ir

همونطور که در بالا بیان شد هدف از این تاپیک آموزش استفاده از توابع کتابخانه pff.h می باشد .
تفاوت این کتابخانه با h.ff در اشغال کردن حافظه های flash و Sram می باشد که حافظه Sram در استفاده از این کتابخانه بیشتر مد نظر خواهد بود.

کتابخانه مذکور توسط یک استاد ژاپنی تهیه و به صورت آزاد بر روی اینترنت قرار داده شده :

http://elm-chan.org/fsw/ff/00index_e.html

فرمت های قابل پشتیبانی کتابخانه شامل fat12/fat16/fat32 می باشد و با توجه به ساپورت fat32 امکان ساپورت کارت های با مقدار فضای بیشتر از 4 گیگ هم امکان پذیر خواهد بود.
قبل از ادامه بحث بهتره کتابخانه ذکر شده را با مخففات از لینک زیر دانلود کنید.

soran111.persiangig.com/document/SD_Driver.rar

در این قسمت قبل از هر چیز توضیماتی مفترض در مورد هدرهای ارائه شده در لینک بالا را برآتون قرار میدهیم.

هدر : integer

با استفاده از این هدر نام های جدیدی به انواع داده ای نسبت داده شده یعنی شما بعد از این هرجا uint و دیدین اگه به هدر مراجعه کنید می فهمید که منظورش unsigned int هستش .

هدر : diskio

در این هدر توابع پایه ای کار با کتابخانه fat گنجانده شده است.

هدر : mmc

کتابخانه pff.h از spi سفت افزاری استفاده می کند و لذا شما موقع طراحی سفت افزار باید مواظب این قسمت باشید. هدر mmc شامل تنظیمات spi و پین های مورد استفاده و همچنین توابع پایه ای کار با mmc . فب قبلا که این توابع در diskio بود و شاید بگین چرا تکرار شده و باید بگم تکرار نشده و اگه به هدر diskio نگاه کنید در اون قسمت توابع فقط تعریف شدن و عملکردشون مشخص نیست.

هدر : pff

اصلی ترین هدر مورد استفاده ما همین هستش و تمامی توابع کار با fat در این هدر گنجانده شده است و یک نکته هائز اهمیت دیگه این که در pff.h یک سری تنظیمات برای فعال و غیر فعال کردن برفی توابع هم وجود دارد که در جای خود توضیح خواهم داد.

فب ابتدا بعد از باز کردن کدویزن و سافت یک پروژه فقط نمایشگر کاراکتری و در حالت دیفالت (یعنی اتصال به پورت A) و تعداد سطر 16 فعال کنید.

بعد از تولید کدها تمامی فطوط اضافی (پاک کنید و کتابخانه های mmc.c و pff.c را نیز به برنامه اضافه کنید .

قبل از هر کاری یک بار برنامه رو کامپایل کنید که فطاوی نداشته باشد و به warning هایی که میده زیاد مساحت نشون ندین .
حالا pff.h و باز کنید و همونطور که می بینید در اول فطوط برنامه پند دیفاین برای فعال و غیر فعال کردن قسمت های مختلف کتابخانه را مشاهده می کنید . افرین قسمت مربوط به شناسایی fat32 را یک کنید یعنی فقط زیر :

```
#define _FS_FAT32 1
```

بقیه قسمت ها در جای مناسب فعال یا غیر فعال میشن و در حال حاضر به بقیه کاری نداشته باشین .

بریم سراغ اولین تابع :

این‌شیلیز کردن کارت حافظه :

شکل کلی تابع :

```
DSTATUS disk_initialize (void);
```

کاربردش که مشخصه و به این صورت کار می کنه که وقتی فراخوانیش می کنید در جواب اگر صفر (و برگشت داد یعنی با وفقت کارت حافظه شناسایی شده و در غیر این صورت عملیات ناموفق بوده .

تابع ۵۹

باز کردن درایو کارت حافظه :

شکل کلی تابع :

```
FRESULT pf_mount (FATFS*);
```

نکته ای که در مورد این تابع هست و باید بدونید اینکه در ff.h یک شماره هم برای تابع به عنوان شماره درایو ارسال میشے ولی پون pff.h فقط برای یک درایو هست در نتیجه به صورت پیش فرض اون ارگومان خذف شده.

کارکرد تابع:

باز کردن یک درایو در fat برای ثبت یا فوندن اطلاعات و کلا قبل از هر کاری باید این تابع فراخوانی بشه.

البته تابع فقط یک بار در اول برنامه فراخوانی میشے و تمام.

۹۰۹دی ۶ فروجی های تابع:

۹۰۹دی تابع یک اشاره گر fs= FileSystemObject به متغیری که از قبل باید به عنوان ساختمان تعریف شده باشد می باشد.

یعنی باید از قبل متغیری را به این شکل تعریف کنید:

```
FATFS FS;
```

فروجی های تابع نیز به صورت زیر فواهند بود:

- FR_OK
- FR_NOT_READY
- FR_DISK_ERR
- FR_NO_FILESYSTEM

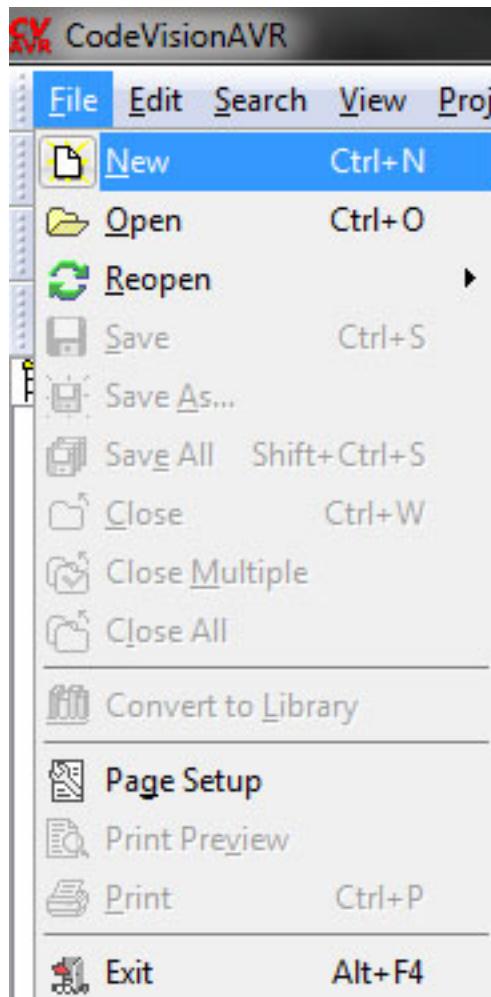
فب قبل از ادامه بهتر شروع کنیم به نوشتن یک سورس اولیه با این دو تابع پایه و تست هایی (۹۱ اوژنه) انجام بدیم.

پایان نقل قول از دوست عزیزم سوران آراسته در وب سایت ECA.ir

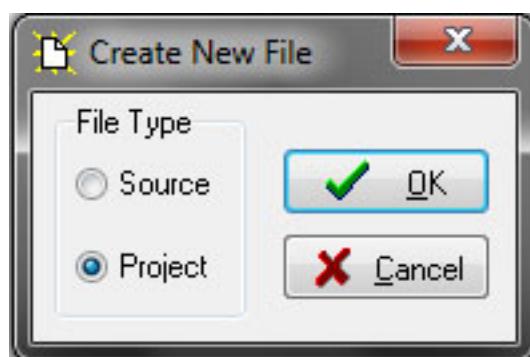
ساخت پروژه جدید برای خواندن از Micro SD در CodeVision

برای ساخت پروژه جدید ابتدا CodeVision را باز کرده و مطابق عکس های زیر کار را پیش می بریم :

1. از منوی فایل گزینه New را انتخاب می کنیم

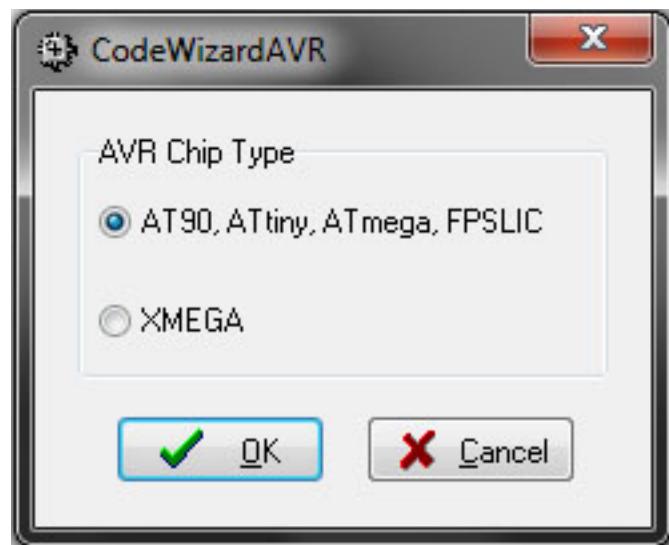


2. در پنجره باز شده گزینه Project را انتخاب و OK می کنیم

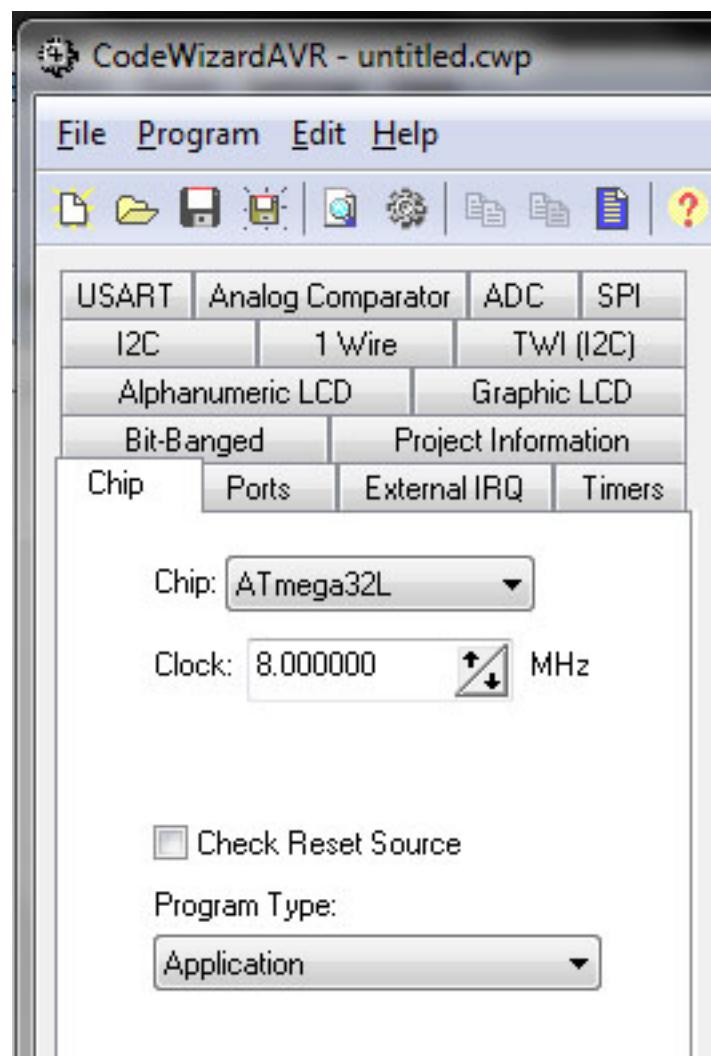


3. در پنجره باز شده Yes را می زنیم

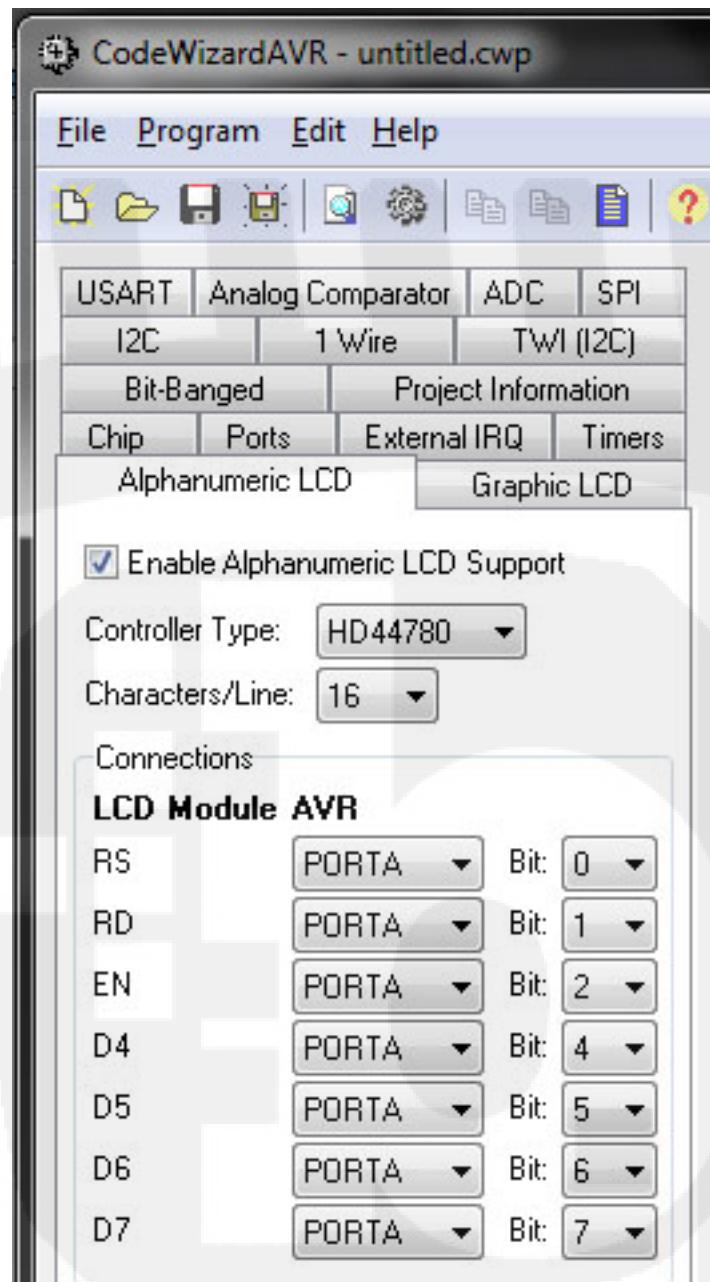
4. در پنجره جدید خانواده میکرو کنترلر خود را انتخاب کرده و OK می کنیم



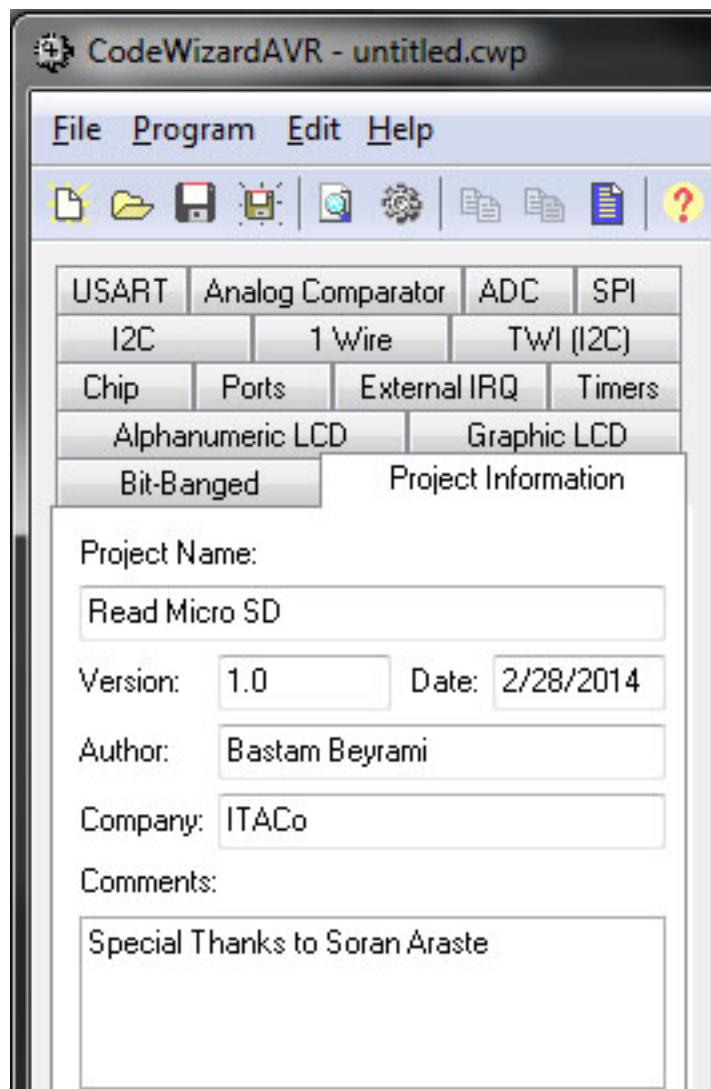
5. در صفحه باز شده برای Chip میکروکنترلر مورد نظر (ATMega32L) را انتخاب و Clock را 8 مگاهرتز قرار می دهیم



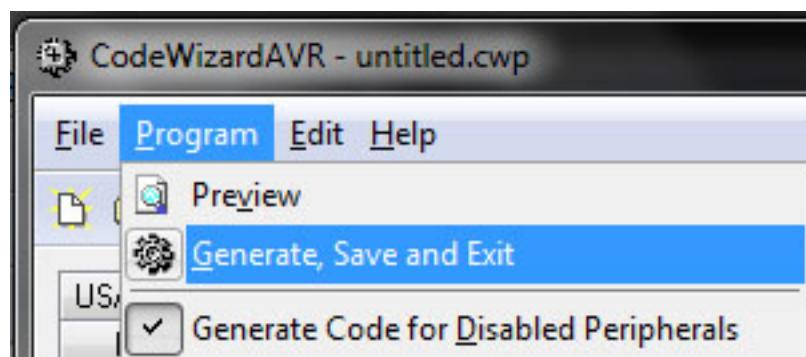
6. از برجسب های بالا Alphanumeric LCD را انتخاب کرده ، تیک Enable را زده و تعداد ستون های LCD و شماره پایه ها را مشخص می نماییم



7. از برچسب های بالا Project Information را انتخاب کرده و اطلاعات پروژه را وارد می کنیم



8. حل از منوی گزینه Program را انتخاب کرده و با دادن نام دلخواه و مسیر مورد نظر پروژه را ذخیره می کنیم



مشاهده می کنید کد مورد نظر تولید شده و در پنجره های باز شده نمایش داده می شود

اضافه کردن فایل های کتابخانه pff.h به پروژه

- برای این کار ابتدا پوشه SD_Driver که ضمیمه این فایل شده است را داخل پوشه پروژه کپی می کنیم
- سپس خطوط زیر را به ابتدای برنامه اضافه می کنیم

```
#include <mega32.h>
#include <alcd.h>
#include <delay.h>
#include "SD_Driver/mmc.c"
#include "SD_Driver/pff.c"

#define xtal 8000000

// Declare your global variables here
#define BUFFER_SIZE 16

FATFS Fs;
WORD w_br;

void main(void)
{
    char buffer[BUFFER_SIZE];
    lcd_init(16);
    lcd_gotoxy(1,0);
    lcd_putsf("Read Micro SD");
    lcd_gotoxy(1,1);
    lcd_putsf("Bastam Beyrami");
    delay_ms(2000);
    lcd_clear();
    lcd_putsf("Init Drive");
    while(disk_initialize()!=FR_OK) delay_ms(100);
```

- و برنامه را برای خواندن یک فایل TXT به اسم "Bastam.txt" از Micro SD که با FAT32 فرمت شده و فایل مذکور در آن کپی شده است این گونه می نویسیم

```
lcd_putsf( "-->OK" );

delay_ms(1000);

lcd_clear();

lcd_putsf( "OPEN Drive" );

while(pf_mount(&Fs)!=FR_OK) delay_ms(100);

lcd_putsf( "-->OK" );

delay_ms(1000);

lcd_clear();

lcd_putsf( "Open File" );

while(pf_open("Bastam.txt")!=FR_OK) delay_ms(100);

lcd_putsf( "-->OK" );

delay_ms(1000);

lcd_clear();

lcd_putsf( "Read File" );

while(pf_read(&buffer,12,&w_br)!=FR_OK) delay_ms(100);

lcd_putsf( "-->OK" );

delay_ms(1000);

lcd_clear();

lcd_putsf( "CLOSE Drive" );

while(pf_mount(0)!=FR_OK) delay_ms(100);

lcd_putsf( "-->OK" );

delay_ms(1000);

lcd_clear();

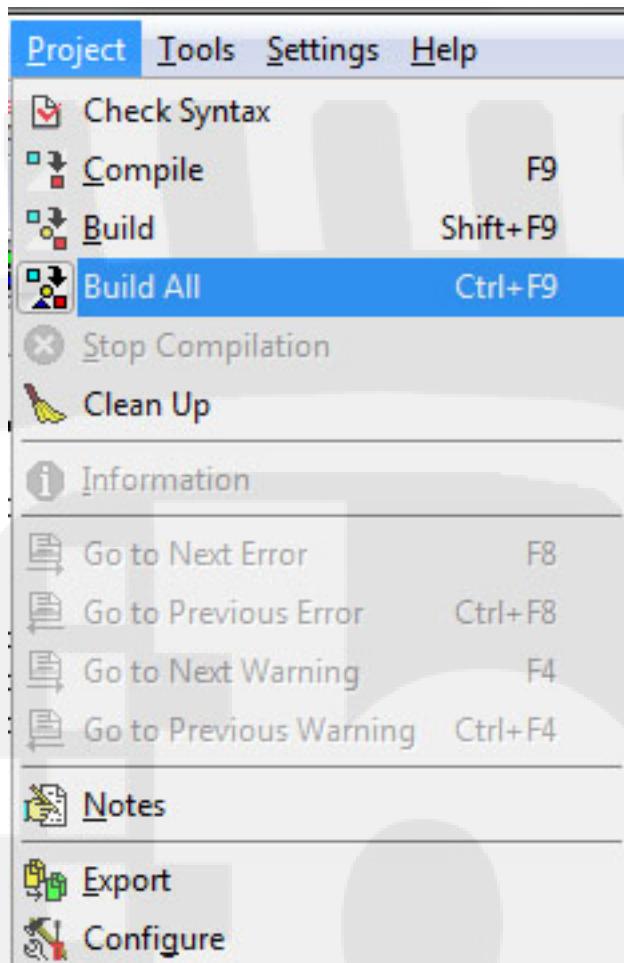
lcd_puts(buffer);

delay_ms(1000);

while (1);

}
```

- در مرحله پایانی پروژه را Save و Compile می کنیم



در زمان کامپایل کردن ممکن است با تعدادی Warning مواجه شوید که اهمیت نداشه و فایل Hex مورد نظر تولید خواهد شد.

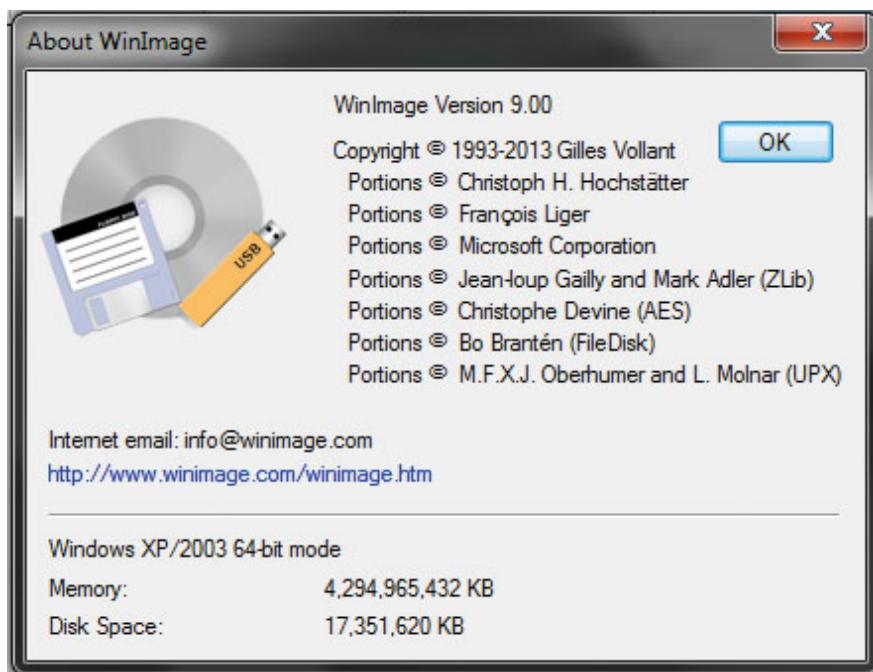
فقط کافیست فایل Hex را بر روی میکروکنترلر Program کنید و Micro SD را داخل سوکت قرار دهید و نتیجه را مشاهده نمایید.

آموزش ساخت WinImage با Proteus برای کار با SD در

برای شبیه سازی برنامه نوشته شده با پروتئوس نیاز به فایل Image به عنوان درایو SD داریم که مراحل ساخت این درایو مجازی در ادامه آورده شده است

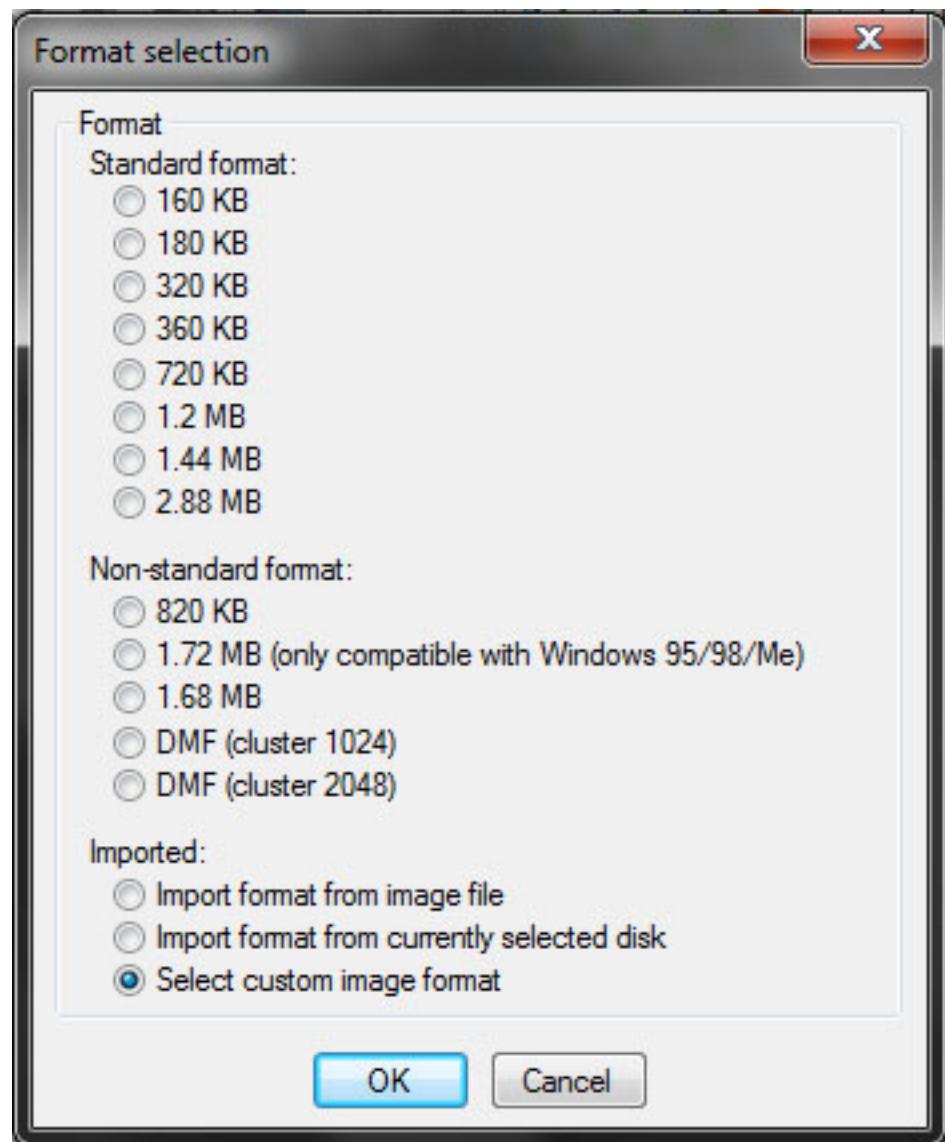
1. ابتدا باید برنامه WinImage را دانلود و نصب کنید (آموزش بر اساس ورژن 9 است)

که میتوانید آخرین ورژن برنامه را از لینک زیر دانلود کنید

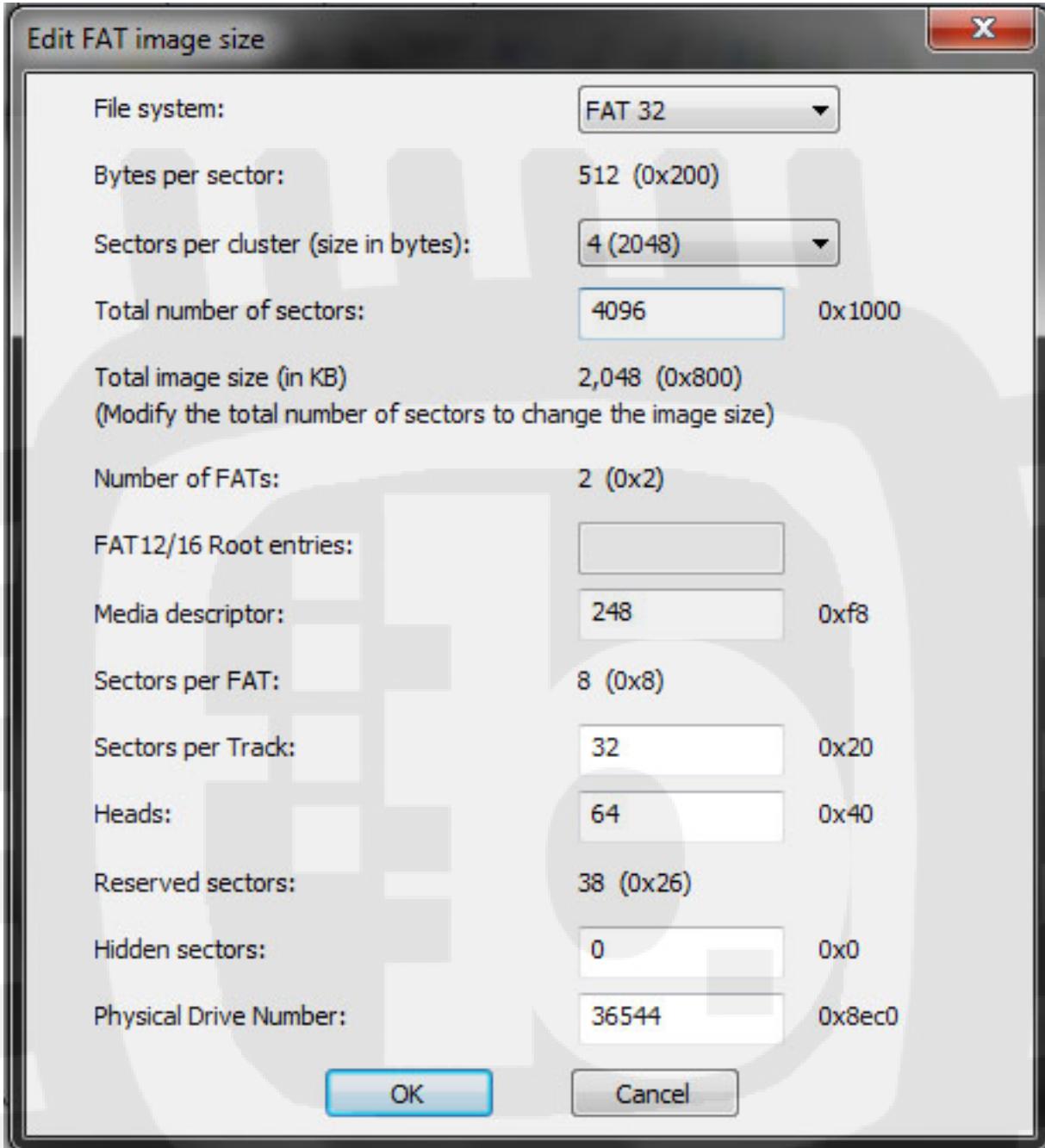


<http://www.winimage.com/download.htm>

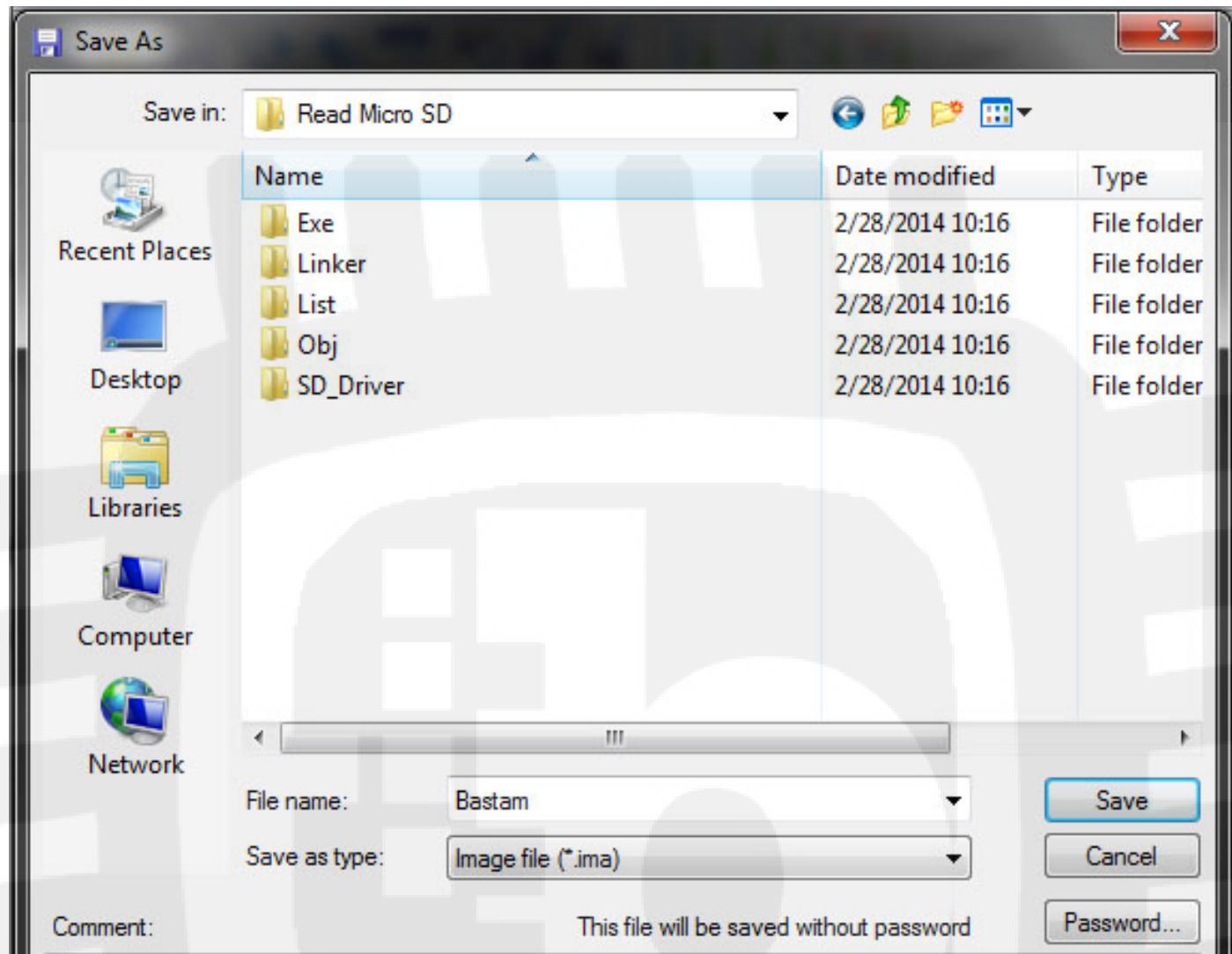
2. از منوی File گزینه New را انتخاب کنید و در پنجره باز شده از زیر روال Imported Image عبارت Format را انتخاب کرده و OK نمایید



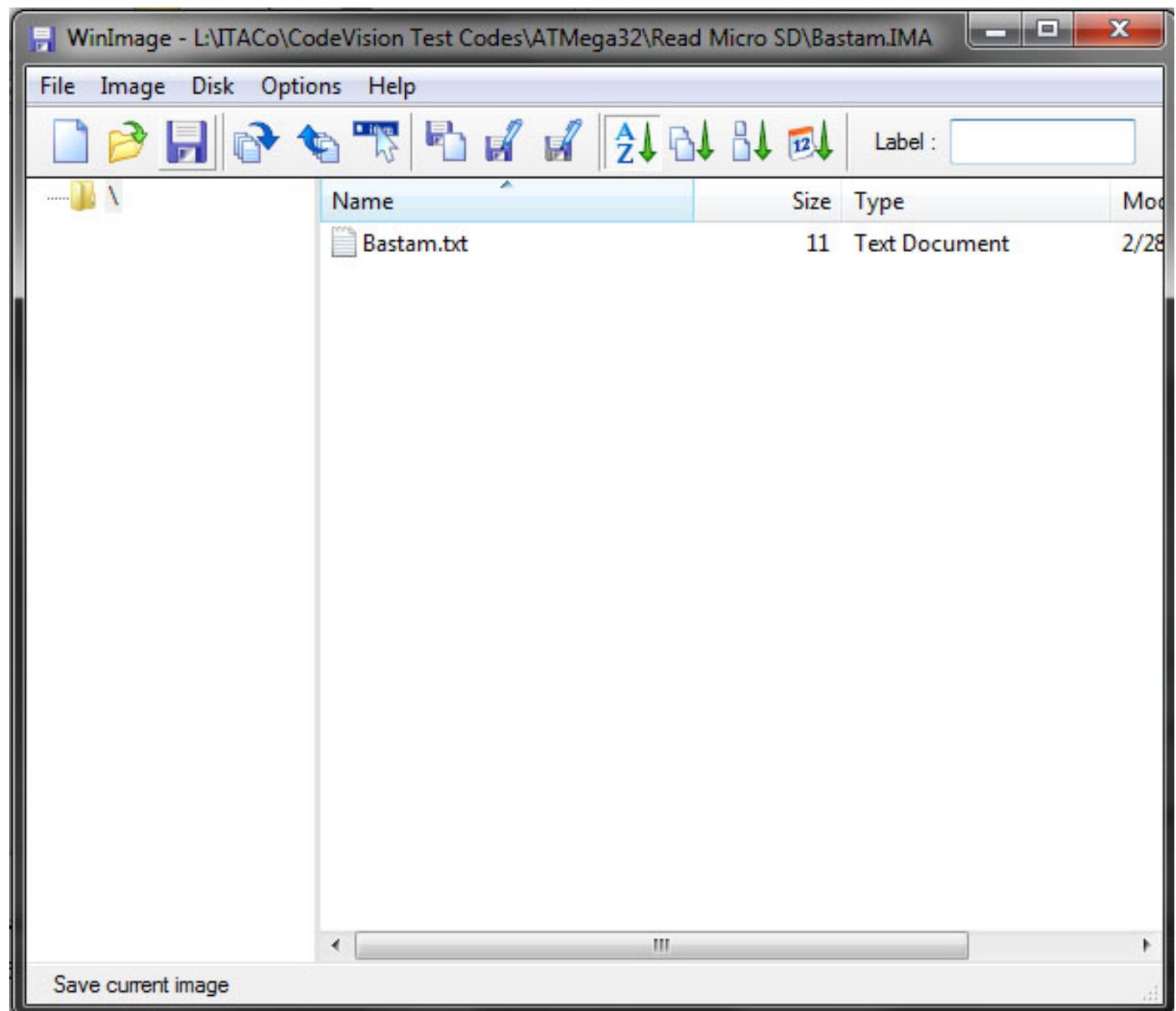
.3 جلوی عبارت File System FAT32 را انتخاب نمایید و مقدار Total Number Of Sectors را عدد 4096 (به معنای حافظه ای با ظرفیت 4 مگا بایت) وارد کنید و دیگر پارامترها را بدون تغییر رها کنید و OK کنید



.4 حل از منوی File بر روی گزینه Save کلیک کنید و مسیر دلخواه و نام مورد نظر را وارد نمایید و نوع فایل را برای Save عبارت (*.ima) انتخاب کرده و فایل را Save As Type کنید



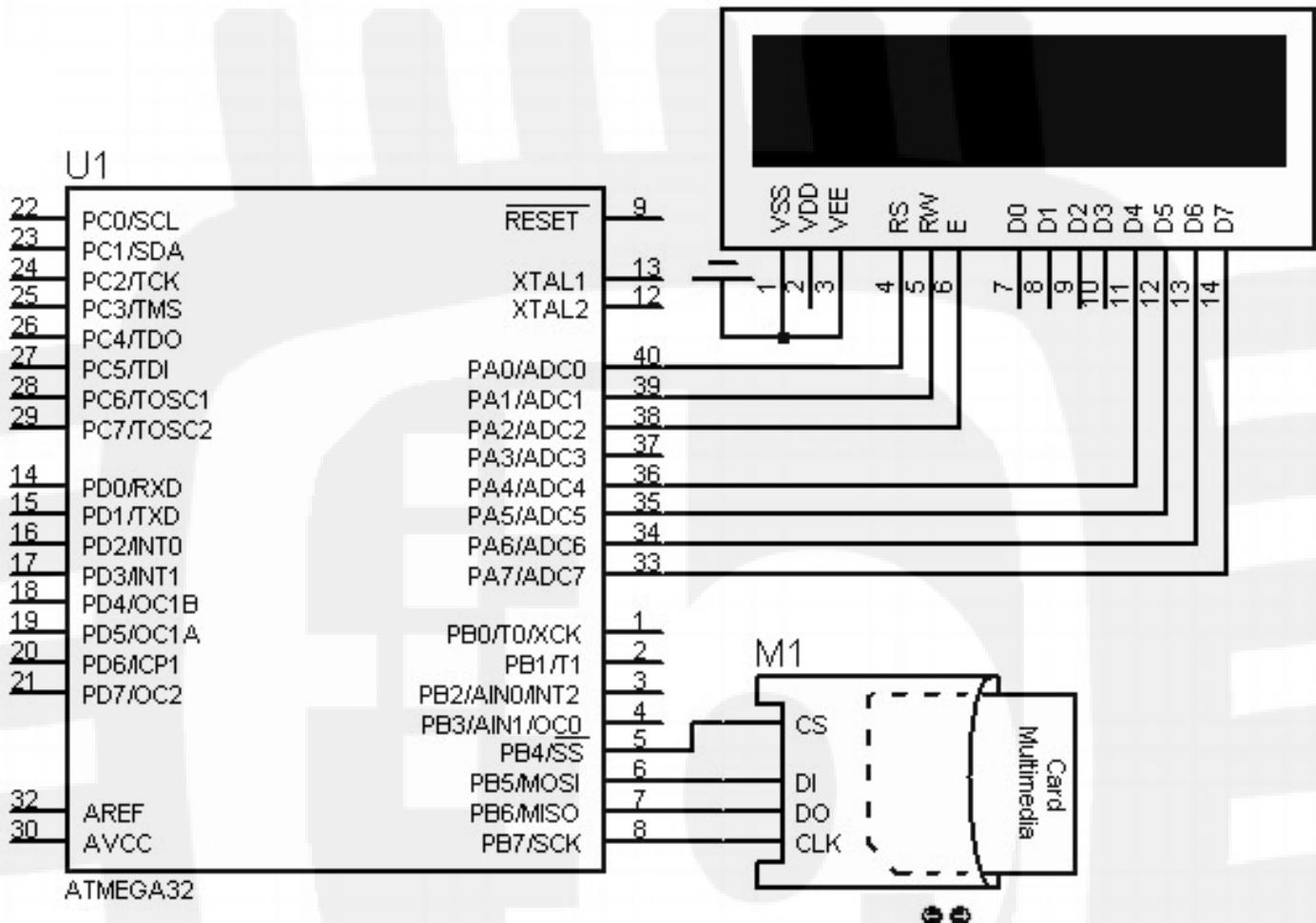
5. حل در صفحه اصلی برنامه فایل مورد نظر (فایل Bastam.txt) که قرار است اطلاعات از روی آن خوانده شود را وارد نمایید (Drag & Drop) و روی Yes کلیک کنید تا فایل داخل درایو مجازی قرار گیرد و مجدد آن را Save نموده و برنامه را بیندید



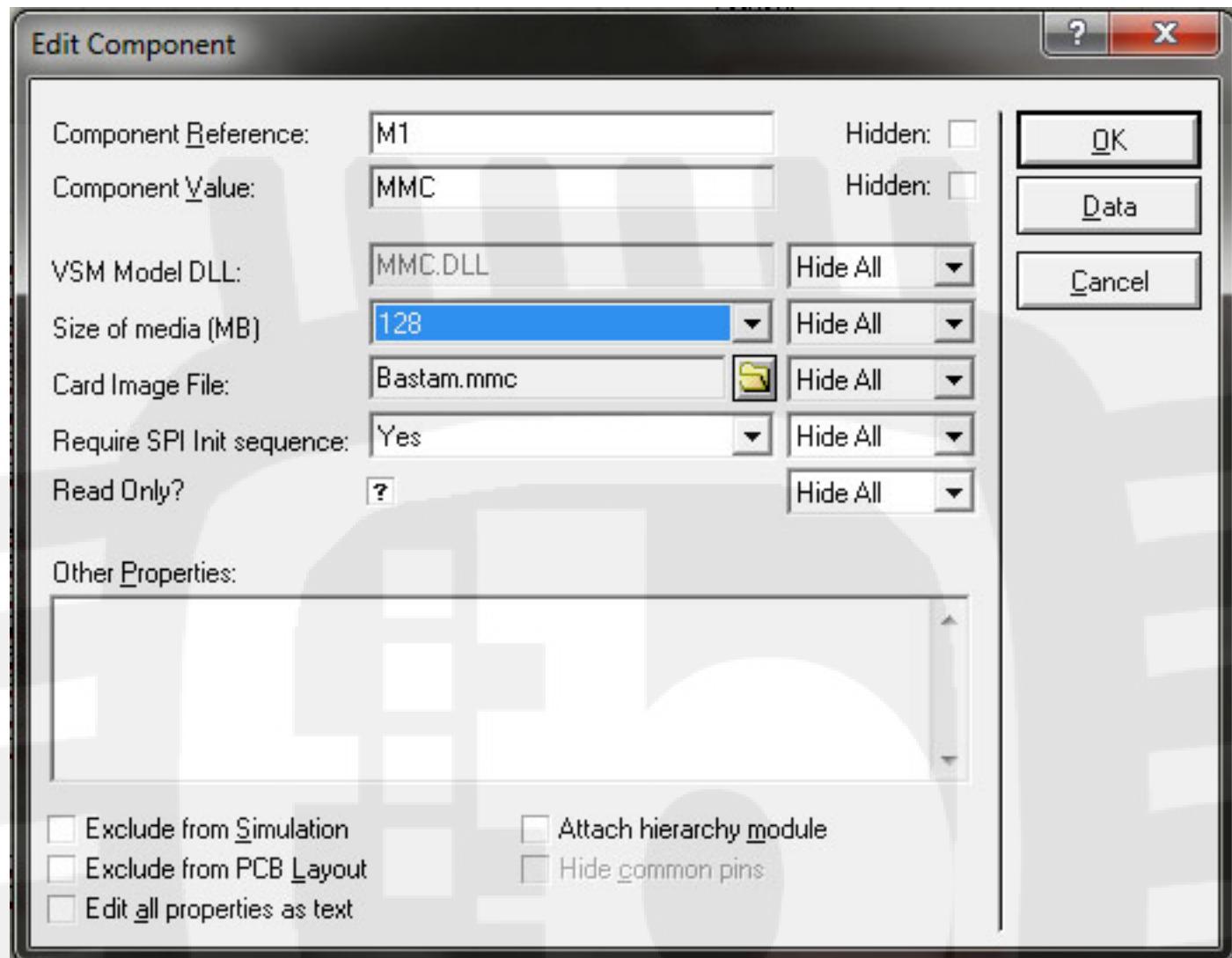
6. در مرحله آخر به مسیر ذخیره فایل Bastam.ima رفته و آن را به Bastam.mmc تغییر نام دهید.

آموزش کار با SD در Proteus

1. در پروتئوس شماتیک مورد نظر را رسم نمایید



2. بر روی شماتیک SD راست کلیک کرده و از منوی باز شده Edit Properties را انتخاب کنید و در صفحه باز شده Size Of Media (MB) را برابر 128 قرار دهید و در قسمت Card Image File (MB) فایل تولید شد توسط WinImage در قسمت قبل که با پسوند *.mmc ذخیره شده (در اینجا Bastam mmc) بود را وارد نموده و OK کنید



3. فایل Hex برنامه را هم به میکروکنترلر در پروتئوس وارد نمایید و پروژه را Save کنید و Play نمایید



Read File-->OK

CLOSE Drive-->OK

V_D V_{EE} R_S E D_D S_D S_A S_B

V_D V_{EE} R_S E D_D S_D S_A S_B

09 5534

V_D V_{EE} R_S E D_D S_D S_A S_B

نکته: دقت نمایید که بسته میکروکنترلر در پروتئوس 8 مگاهرتز داخلی باشد.

ادامه آموزش کتابخانه pff.h کدویژن

نقل قول از دوست عزیزم عباس صرامی در وب سایت ECA.ir

با اجازه دوستان چند فطی هم من بنویسم:

ابتدا بهتره انواع فرمومی توابع را بشناسیم، در فایل diskio.h کد زیر را می بینید:

```
typedef enum {
    RES_OK = 0,
    RES_ERROR,
    RES_NOTRDY,
    RES_PARERR
} DRESULT;
```

/* 0: Function succeeded */
/* 1: Disk error */
/* 2: Not ready */
/* 3: Invalid parameter */

فوب این یک enum هست که به ازاء هر عدد یک نام اختفاب کرده (فهرست اعداد از 0 شروع میشے)
مثلاً اگه یک تابع را فراخوان کردید و تابع به شما 0 برگرداند یعنی همه چیز OK هست (RES_OK یعنی همون
این هم یکی دیگه که توی فایل pff.h هست (مربوط به نتیجه توابع کار با فایل)

```
typedef enum {
    FR_OK = 0,
    FR_DISK_ERR,
    FR_NOT_READY,
    FR_NO_FILE,
    FR_NO_PATH,
    FR_NOT_OPENED,
    FR_NOT_ENABLED,
    FR_NO_FILESYSTEM
} FRESULT;
```

/* 0 */
/* 1 */
/* 2 */
/* 3 */
/* 4 */
/* 5 */
/* 6 */
/* 7 */

هنگامی که تابع disk_initialize را فراخوان میکنید تنظیمات مربوط به SPI انجام میشے ، در ضمن شما باید در فایل mmc.c
بین های مربوطه را معرفی کنید

#define SCK_DDR	DDRB
#define SCK_PRT	PORTB
#define SCK_BIT	7
#define MOSI_DDR	DDRB
#define MOSI_PRT	PORTB
#define MOSI_BIT	5
#define MISO_DDR	DDRB
#define MISO_PRT	PORTB
#define MISO_BIT	6
#define SD_CS_DDR	DDRB
#define SD_CS_PRT	PORTB
#define SD_CS_BIT	4

فوب تامالا پيکر بندی MMC/SD و FAT را ياد گرفتيم(توابع زير)

```
DSTATUS disk_initialize (void);  
FRESULT pf_mount (FATFS*);
```

راي 5) با توابع FAT در ابتدا مى رويم سراغ : pf_open

فرم اين تابع به اين شكل هست (در فایل pff.h) :

```
FRESULT pf_open (const char*); /* Open a file */
```

در اينجا همون طور که مشخصه تابع يك پaramتر ميگيره و فرمي FRESULT داره

ورودي تابع نام فایل هست مثلاً "Bastam.txt" فرمي تابع همون طور که در enum هايي که توضيح داده مشخصه اگه برگردانه يعني فایل باز شده FR_OK

مثال:

```
FRESULT _result;  
_result = pf_open("test.txt");  
if(_result == FR_OK)  
{  
    //other codes , File is open  
}  
else  
{  
    //error  
}
```

پس تا حال:

1. SD را راه انداختيم
2. جدول را پك گرديم
3. فایل مورد نظرمون را باز گرديم

فوب برويم سراغ تابع (در فایل pf_read) :

```
FRESULT pf_read (  
    void* buff,  
    stream)* /  
    WORD btr,  
    WORD* br  
)  
{  
    /* Pointer to the read buffer (NULL:Forward data to the  
    /* Number of bytes to read */  
    /* Pointer to number of bytes read */
```

ساختار فایل به این شکل است که تابع سه پارامتر ورودی و خروجی FRESULT دارد

اولین ورودی بافری هست که قراره دیتا مون داخلش ریخته بشے

`void* buff,` Pointer to the read buffer (NULL:Forward data to the stream

دومیش تعداد بایتی هست که میتوانی از فایل بخونی

WORD btr, Number of bytes to read

و سومیش تعداد بایتی هست که سیستم توانسته بخونه

مثال:

```
unsigned char MyBuffer[50];
HRESULT _result;
WORD ReadIndex;
_result = pf_read(&MyBuffer, 50, &ReadIndex);
if(_result == FR_OK)
{
// read file ok
}
else
{
//error
}
```

در اینجا یک بافر داریم که 50 بایت ظرفیت دارد و به تابع هم گفته‌یم 50 بایت بفروان ، اگه تعداد بایت‌ها در فایل کمتر از 50 بایتی که اعلام کردیم باشه میتوانیم توسط ReadIndex که ایجاد کرده تعداد بایت فوانده شده را بفهمیم

: (pff.c در فایل pf_write هست

```

FRESULT pf_write (
    const void* buff,           /* Pointer to the data to be written */
    WORD btw,                  /* Number of bytes to write (0:Finalize the
                                current write operation) */
    WORD* bw                   /* Pointer to number of bytes written */
)

```

ظاهر کار عین خواندن هست ، تابع `3تا` یا `امتر` وروdi داره و `FRESULT` براي فروجي:

اولین و وودی مربوط به بافری که میفواهیم داخل فایل بنویسیم هست

```
const void* buff,          /* Pointer to the data to be written */  
  
WORD btw,                /* Number of bytes to write (0:Finalize the current write operation) */  
  
WORD* bw                 /* Pointer to number of bytes written */
```

پارامتر بعدی طول بافری که میفواهیم بنویسیم هست

و آخرین پارامتر تعداد بایت های نوشته شده است

مثال:

```
unsigned char MyBuffer[15] = { 'B', 'A', 'S', 'T', 'A', 'M' };  
HRESULT _result;  
WORD WriteIndex;  
_result = pf_write(&MyBuffer, 15, &WriteIndex);  
if(_result == FR_OK)  
{  
// write file ok  
}  
else  
{  
//error  
}
```

یک فایل txt. توسط کامپیوتروی SD ایجاد کنید و توسط تابع pf_open اون را باز کنید و کد بالا تست کنید

فقط یک نکته مهم را توجه کنید: در فایل pff.h مقدار USE_WRITE را برابر یک کنید :

```
#define _USE_WRITE     1      /* pf_write(): 0:Remove ,1:Enable */
```

برایم سراغ یک تابع دیگه امروز تابع pf_lseek را بررسی میکنیم

فرم تابع به این شکل :

```
HRESULT pf_lseek (DWORD);           /* Move file pointer of the open file */
```

فوب مشخصه که تابع 1 ورودی و خروجی FRESULT دارد ، این تابع کمک میکنه که شما اگه آدرس خاصی از فایل را بخواهید بخواهید بتوانید آنرا انتخاب کنید ، مثلًا من میخواهم از بایت 5 به بعد فایل را بخوانم باید قبل از تابع pf_read این طوری بنویسم:

```
pf_lseek(5);
```

در ضمن باید تابع را در فایل pff.h فعال کنید

```
#define _USE_LSEEK 1 /* 1:Enable pf_lseek() */
```

پایان نقل قول از دوست عزیزم عباس صرامی در وب سایت [ECA.ir](#)

و اما نوشتن با کتابخانه pff.h

همانطور که عباس صرامی عزیز توضیح دادند تابع pf_write برای نوشتن روی SD استفاده می شود اما چننا ولی داره :

نقل قول از استاد عزیز هادی اسدی در وب سایت [ECA.ir](#)

- نمی شود فایل ایجاد کرد و فقط باید فایل موجود را ادیت کنید
- نمی توان زمان و تایم فایل را ادیت شده را تغییر داد
- هم فایل محدود است و امکان تغییر آن وجود ندارد
- شروع و پایان نوشتن باید در یک سکتور باشد (به این مورد دقت شود)
- قبل از نوشتن Read-only بودن فایل چک نمی شود
- نوشتن فایل هتما باید پشت سر هم و به این ترتیب انجام بشه

```
• pf_lseek(ofs);
```

برای رفتن به آدرسی که باید از آنها نوشتن شروع شود

```
• pf_write(buff, btw, &bw);
```

برای اینیشیال شروع نوشتن در عمل هیچ کاری نمی کند فقط پوینترها را اینیشیال می کند

```
• pf_write(buff, btw, &bw);
```

نوشتن دیتا "در حال اجرای این تابع امکان دسترسی به فایل دیگری ممکن نیست"

```
• pf_write(0, 0, &bw);
```

مه مه مه مه مه پایان عملیات نوشتن

برای ادامه نوشتن می تونید مورد 3 را قبل از 4 هر چند بار که دوست داشتید استفاده کنید.

یعنی موارد 1 و 2 و 4 فقط یک بار استفاده می شوند ولی مورد 3 می تواند تا پایان نوشتن دیتا پشت سر هم استفاده شود.

در صورتی که شرط $bw < btw$ * برقرار شد یعنی به انتهای فایل رسیدید و دیگه امکان نوشتن و نفواید داشت.

مورد دو^ه و سوم دقیقاً مثل همند پس با اول که از تابع استفاده می‌کنید هیچ کاری نفواید کرد و فقط برای اینیشیال هست که هتما باید به این نکته توجه کنید

و فایل هتما باید قبل از دیتا داشته باشد چون امکان تغییر سایز فایل و ندارید برای این کار می‌توانید فایل (و با هگز ادیتør باز) کرده و تا هر اندازه که دوست دارید مقدار صفر بذید.

نمونه برنامه نهایی و تست شده :

```
#include <io.h>
#include <delay.h>
#include <stdio.h>

#include "pff/pff.c"
#include "pff/mmc.c"
#include "pff/diskio.h"

void main(void)
{
    FATFS    fatfs;
    WORD     bw, i=0;
    char     buff[32];

    do
    {
        delay_ms(200);
    }while(pf_mount(&fatfs));

    if(!pf_open("write.txt"))
    {
        for (i=0; i<=50000; i++)
        {
            sprintf(buff, "%5u=POLESTAR\n", i);
            if (pf_write(buff, 16, &bw) || !bw) break;
        }
    }
    pf_write(0, 0, &bw);
    for (++);
}
```

پایان نقل قول از استاد عزیز هادی اسدی در وب سایت ECA.ir

در این مثال فایل هایی که Include شده اند بینهایت با اهمیت هستند که ضمیمه این فایل آموزشی شده اند.

فایل های پروژه و تمامی سمبل های نوشته شده ضمیمه شده اند.

BMP file format

From Wikipedia, the free encyclopedia

The **BMP file format**, sometimes called **bitmap** or **DIB file format** (for *device-independent bitmap*), is an image file format used to store bitmap digital images, especially on Microsoft Windows and OS/2 operating systems.

Many older graphical user interfaces used bitmaps in their built-in graphics subsystems;^[1] for example, the Microsoft Windows and OS/2 platforms' GDI subsystem, where the specific format used is the *Windows and OS/2 bitmap file format*, usually named with the file extension of .BMP or .DIB.

Contents

- 1 Pixel storage
- 2 Device-independent bitmaps and BMP file format
 - 2.1 DIBs in memory
 - 2.2 BMP File Header
 - 2.3 Bitmap Information (DIB header)
 - 2.4 Color palette
 - 2.5 Bitmap data
 - 2.6 Example of a 2x2 Pixel, 24-Bit Bitmap
- 3 Usage of BMP format
- 4 Related formats
- 5 See also
- 6 References
- 7 External links

Pixel storage

In uncompressed BMP files, and many other bitmap file formats, image pixels are stored with a color depth of 1, 4, 8, 16, 24, or 32 bits per pixel (BPP). Images of 8 bits and fewer can be either grayscale or indexed color. An alpha channel (for transparency) may be stored in a separate file, where it is similar to a grayscale image, or in a fourth channel that converts 24-bit images to 32 bits per pixel.

Uncompressed bitmap files (such as BMP) are typically much larger than compressed (with any of various methods) image file formats for the same image. For example, the 1058×1058 Wikipedia logo, which occupies about 271 KB in the lossless PNG format, takes about 3358 KB as a 24-bit BMP file.

Uncompressed formats are generally unsuitable for transferring images on the Internet or other slow or capacity-limited media.

The bits representing the bitmap pixels are packed within rows. Depending on the color depth, a pixel in the picture will occupy at least $n/8$ bytes (n is the bit depth, since 1 byte equals 8 bits). The approximate size for

Windows Bitmap

Filename extension	.bmp or .dib
Internet media type	image/x-ms-bmp (unofficial) or image/x-bmp (unofficial)
Type code	'BMP' 'BMPf' 'BMPP'
Uniform Type Identifier	com.microsoft.bmp
Type of format	Raster graphics

a n-bit (2^n colors) BMP file in bytes can be calculated, including the effect of starting each word on a 32-bit dword boundary, as:

$$\text{RowSize} = 4 \cdot \left\lceil \frac{(BPP \cdot \text{Width})}{32} \right\rceil,$$

$$\text{FileSize} \approx 54 + 4 \cdot 2^{\text{BPP}} + \text{RowSize} \cdot \text{Height}$$

for $BPP \leq 8$

Height and *Width* are given in pixels.

In the formula above, **54** is the size of the headers in the popular Windows V3 BMP version (14-byte BMP file header plus 40-byte DIB V3 header); some other header versions will be larger or smaller than that, as described in tables below. And $4 \cdot 2^n$ is the size of the color palette; this size is an approximation, as the color palette size will be $3 \cdot 2^n$ bytes in the OS/2 V1 version, and some other versions may optionally define only the number of colors needed by the image, potentially fewer than 2^n .^[2] Only files with 8 or fewer bits per pixel use a palette; for 16-bit (or higher) bitmaps, omit the palette part from the size calculation:

$$\text{FileSize} \approx 54 + \text{RowSize} \cdot \text{Height}$$

for $BPP > 8$.

For detailed information, see the sections on file format below.

Device-independent bitmaps and BMP file format

Microsoft has defined a particular representation of color bitmaps of different color depths, as an aid to exchanging bitmaps between devices and applications with a variety of internal representations. They called these device-independent bitmaps or DIBs, and the file format for them is called DIB file format or BMP file format. According to Microsoft support:^[2]

A device-independent bitmap (DIB) is a format used to define device-independent bitmaps in various color resolutions. The main purpose of DIBs is to allow bitmaps to be moved from one device to another (hence, the device-independent part of the name). A DIB is an external format, in contrast to a device-dependent bitmap, which appears in the system as a bitmap object (created by an application...). A DIB is normally transported in metafiles (usually using the StretchDIBits() function), BMP files, and the Clipboard (CF_DIB data format).

A typical BMP file usually contains the following blocks of data:

BMP File Header	Stores general information about the BMP file.
Bitmap Information (DIB header)	Stores detailed information about the bitmap image.
Color Palette	Stores the definition of the colors being used for indexed color bitmaps.
Bitmap Data	Stores the actual image, pixel by pixel.

The following sections discuss the data stored in the BMP file or DIB in details. This is the standard BMP file format.^[2] Some bitmap images may be stored using a slightly different format, depending on the application that creates it. Also, not all fields are used; a value of 0 will be found in these unused fields.

DIBs in memory

A BMP file is loaded into memory as a DIB data structure, an important component of the Windows GDI API. The DIB data structure is the same as the BMP file format, but without the 14-byte BMP header.

BMP File Header

This block of bytes is at the start of the file and is used to identify the file. A typical application reads this block first to ensure that the file is actually a BMP file and that it is not damaged. The first two bytes of the BMP file format are the character 'B' then the character 'M' in 1-byte ascii encoding. All of the integer values are stored in little-endian format (i.e. least-significant byte first).

Offset#	Size	Purpose
0000h	2 bytes	the magic number used to identify the BMP file: 0x42 0x4D (Hex code points for B and M). The following entries are possible: <ul style="list-style-type: none"> ■ BM - Windows 3.1x, 95, NT, ... etc ■ BA - OS/2 Bitmap Array ■ CI - OS/2 Color Icon ■ CP - OS/2 Color Pointer ■ IC - OS/2 Icon ■ PT - OS/2 Pointer
0002h	4 bytes	the size of the BMP file in bytes
0006h	2 bytes	reserved; actual value depends on the application that creates the image
0008h	2 bytes	reserved; actual value depends on the application that creates the image
000Ah	4 bytes	the offset, i.e. starting address, of the byte where the bitmap data can be found.

Equivalent C-Language Header

```
#include <stdint.h>

/* Note: the magic number has been removed from the bmp_header structure
   since it causes alignment problems
   struct bmpfile_magic should be written/read first
   followed by the
   struct bmpfile_header
   [this avoids compiler-specific alignment pragmas etc.] */

struct bmpfile_magic {
    unsigned char magic[2];
};

struct bmpfile_header {
    uint32_t filesz;
    uint16_t creator1;
    uint16_t creator2;
    uint32_t bmp_offset;
};
```

Bitmap Information (DIB header)

This block of bytes tells the application detailed information about the image, which will be used to display the image on the screen. The block also matches the header used internally by Windows and OS/2 and has several different variants. All of them contain a dword field, specifying their size, so that an application can easily determine which header is used in the image. The reason that there are different headers is that Microsoft extended the DIB format several times. The new extended headers can be used with some GDI functions instead of the older ones, providing more functionality. Since the GDI supports a function for loading bitmap files, typical Windows applications use that functionality. One consequence of this is that for such applications, the BMP formats that they support match the formats supported by the Windows version being run. See the table below for more information.

Size	Header	Identified by	Supported by the GDI of
12	OS/2 V1	BITMAPCOREHEADER	OS/2 and also all Windows versions since Windows 3.0
64	OS/2 V2	BITMAPCOREHEADER2	
40	Windows V3	BITMAPINFOHEADER	all Windows versions since Windows 3.0
108	Windows V4	BITMAPV4HEADER	all Windows versions since Windows 95/NT4
124	Windows V5	BITMAPV5HEADER	Windows 98/2000 and newer

For compatibility reasons, most applications use the older DIB headers for saving files. With OS/2 being obsolete, for now the only common format is the V3 header. See next table for its description. All values are stored as unsigned integers, unless explicitly noted.

Offset #	Size	Purpose
----------	------	---------

Eh	4	the size of this header (40 bytes)
12h	4	the bitmap width in pixels (signed integer).
16h	4	the bitmap height in pixels (signed integer).
1Ah	2	the number of color planes being used. Must be set to 1.
1Ch	2	the number of bits per pixel, which is the color depth of the image. Typical values are 1, 4, 8, 16, 24 and 32.
1Eh	4	the compression method being used. See the next table for a list of possible values.
22h	4	the image size. This is the size of the raw bitmap data (see below), and should not be confused with the file size.
26h	4	the horizontal resolution of the image. (pixel per meter, signed integer)
2Ah	4	the vertical resolution of the image. (pixel per meter, signed integer)
2Eh	4	the number of colors in the color palette, or 0 to default to 2^n .
32h	4	the number of important colors used, or 0 when every color is important; generally ignored.

Note: The image size field can be 0 for BI_RGB bitmaps.

Equivalent C-Language Header (Windows V3 DBI)

```
typedef struct {
    uint32_t header_sz;
    uint32_t width;
    uint32_t height;
    uint16_t nplanes;
    uint16_t bitspp;
    uint32_t compress_type;
    uint32_t bmp_bytesz;
    uint32_t hres;
    uint32_t vres;
    uint32_t ncolors;
    uint32_t nimpcolors;
} bmp_dib_v3_header_t;
```

The compression method field (bytes #30-33) can have the following values:

Value	Identified by	Compression method	Comments
0	BI_RGB	none	Most common
1	BI_RLE8	RLE 8-bit/pixel	Can be used only with 8-bit/pixel bitmaps
2	BI_RLE4	RLE 4-bit/pixel	Can be used only with 4-bit/pixel bitmaps
3	BI_BITFIELDS	Bit field	Can be used only with 16 and 32-bit/pixel bitmaps.
4	BI_JPEG	JPEG	The bitmap contains a JPEG image
5	BI_PNG	PNG	The bitmap contains a PNG image

Note: BI_JPEG and BI_PNG are for printer drivers and are not supported when rendering to the screen.^[3]

Equivalent C-Language Header (Compression Methods)

```
typedef enum {
    BI_RGB = 0,
    BI_RLE8,
    BI_RLE4,
    BI_BITFIELDS,
    BI_JPEG,
    BI_PNG,
} bmp_compression_method_t;
```

The OS/2 V1 header is also popular:

Offset	Size	Purpose
Eh	4	the size of this header (12 bytes)
12h	2	the bitmap width in pixels.
14h	2	the bitmap height in pixels.
16h	2	the number of color planes; 1 is the only legal value
18h	2	the number of bits per pixel. Typical values are 1, 4, 8 and 24.

Note: OS/2 V1 bitmaps cannot be compressed and cannot be 16 or 32 bits/pixel. All values in the OS/2 V1 header are unsigned integers.

A 32-bit version of DIB with integrated alpha channel has been introduced with Windows XP and is used within its logon and theme system; it has yet to gain wide support in image editing software, but has been supported in Adobe Photoshop since version 7 and Adobe Flash since version MX 2004 (then known as Macromedia Flash).

Color palette

The palette occurs in the BMP file directly after the BMP header and the DIB header. Therefore, its offset is the size of the BMP header plus the size of the DIB header.

The palette is a block of bytes (a table) listing the colors available for use in a particular indexed-color image. Each pixel in the image is described by a number of bits (1, 4, or 8) which index a single color in this table. The purpose of the color palette in indexed-color bitmaps is to tell the application the actual color that each of these index values corresponds to.

A DIB always uses the RGB color model. In this model, a color is terms of different intensities (from 0 to 255) of the additive primary colors red (R), green (G), and blue (B). A color is thus defined using the 3 values for R, G and B (though stored in backwards order in each palette entry).

The number of entries in the palette is either 2^n or a smaller number specified in the header (in the OS/2 V1 format, only the full-size palette is supported).^{[2][4]} Each entry contains four bytes, except in the case of the OS/2 V1 versions, in which case there are only three bytes per entry.^[4] The first (and only for OS/2 V1) three bytes store the values for blue, green, and red, respectively,^[2] while the last one is unused and is filled

with 0 by most applications.

As mentioned above, the color palette is not used when the bitmap is 16-bit or higher; there are no palette bytes in those BMP files.

Bitmap data

This block of bytes describes the image, pixel by pixel. Pixels are stored "upside-down" with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image.^[2] Uncompressed Windows bitmaps can also be stored from the top row to the bottom, if the image height value is negative.

In the original DIB, the only four legal numbers of bits per pixel are 1, 4, 8, and 24.^[2] In all cases, each row of pixels is extended to a 32-bit (4-byte) boundary, filling with an unspecified value (not necessarily 0) so that the next row will start on a multiple-of-four byte location in memory or in the file.^[2] The total number of bytes in a row can be calculated as the *image size(bitmap height in pixels)*. Following these rules there are several ways to store the pixel data depending on the color depth and the compression type of the bitmap.

One-bit (two-color, for example, black and white) pixel values are stored in each bit, with the first (left-most) pixel in the most-significant bit of the first byte.^[2] An unset bit will refer to the first color table entry, and a set bit will refer to the last (second) table entry.

Four-bit color (16 colors) is stored with two pixels per byte, the left-most pixel being in the more significant nibble.^[2] Each pixel value is an index into a table of up to 16 colors.

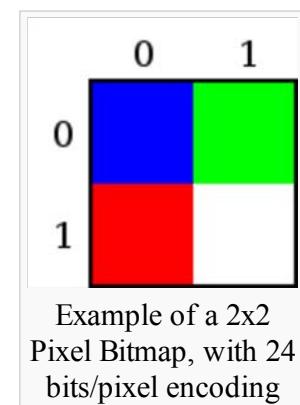
Eight-bit color (256 colors) is stored one pixel value per byte. Each byte is an index into a table of up to 256 colors.

RGB color (24-bit) pixel values are stored with bytes as BGR (blue, green, red).^[2]

Padding bytes are inserted in order to keep the line of data in multiples of four. For example, a 24-bit bitmap with width 1 would have 3 bytes of data per row (blue, green, red) so 1 byte of padding, width 2 would have 2, width 3 would have 3, and width 4 would not have any at all.

Example of a 2x2 Pixel, 24-Bit Bitmap

Offset	Size	Hex Value	Value	Description
0h	2	42 4D	"BM"	Magic Number (unsigned integer 66, 77)
2h	4	46 00 00 00	70 Bytes	Size of the BMP file
6h	2	00 00	Unused	Application Specific
8h	2	00 00	Unused	Application Specific
Ah	4	36 00 00 00	54 bytes	The offset where the bitmap data (pixels) can be found.
Eh	4	28 00 00 00	40 bytes	The number of bytes in the header (from this point).
12h	4	02 00 00 00	2 pixels	The width of the bitmap in pixels



16h	4	02 00 00 00	2 pixels	The height of the bitmap in pixels
1Ah	2	01 00	1 plane	Number of color planes being used.
1Ch	2	18 00	24 bits	The number of bits/pixel.
1Eh	4	00 00 00 00	0	BI_RGB, No compression used
22h	4	10 00 00 00	16 bytes	The size of the raw BMP data (after this header)
26h	4	13 0B 00 00	2,835 pixels/meter	The horizontal resolution of the image
2Ah	4	13 0B 00 00	2,835 pixels/meter	The vertical resolution of the image
2Eh	4	00 00 00 00	0 colors	Number of colors in the palette
32h	4	00 00 00 00	0 important colors	Means all colors are important

Start of Bitmap Data

36h	3	00 00 FF	0 0 255	Red, Pixel (1,0)
39h	3	FF FF FF	255 255 255	White, Pixel (1,1)
3Ch	2	00 00	0 0	Padding for 4 byte alignment (Could be a value other than zero)
3Eh	3	FF 00 00	255 0 0	Blue, Pixel (0,0)
41h	3	00 FF 00	0 255 0	Green, Pixel (0,1)
44h	2	00 00	0 0	Padding for 4 byte alignment (Could be a value other than zero)

Note that the bitmap data starts with the lower left hand corner of the image.

Usage of BMP format

The simplicity of the BMP file format, and its widespread familiarity in Windows and elsewhere, as well as the fact that this format is relatively well documented and free of patents, makes it a very common format that image processing programs from many operating systems can read and write.

While most BMP files have a relatively large file size due to lack of any compression, many BMP files can be considerably compressed with lossless data compression algorithms such as ZIP (in rare cases of non-photographic data, up to 0.1% of original size) because they contain redundant data.

Related formats

Main article: Image file formats

The X Window System uses a similar XBM format for black-and-white images, and XPM (*pixelmap*) for

color images. There are also a variety of "raw" formats, which saves raw data with no other information. The Portable Pixmap (PPM) and Truevision TGA formats also exist, but are less often used – or only for special purposes; for example, TGA can contain transparency information.

Numerous other bitmap file formats are in existence, though most are not widely used.^[5]

See also

- Comparison of graphics file formats

References

1. ^ Julian Smart, Stefan Csomor, and Kevin Hock (2006). *Cross-Platform GUI Programming with WxWidgets* (http://books.google.com/books?id=CyMsvtgnq0QC&pg=PA265&dq=bitmap+pixmap+gui&as_brr=3&ei=4SjwRrTpHYSipgL63NS3BA&sig=4_ev_R-Xs8tXCVONCaiJEnFLtI0) . Prentice Hall. ISBN 0131473816. http://books.google.com/books?id=CyMsvtgnq0QC&pg=PA265&dq=bitmap+pixmap+gui&as_brr=3&ei=4SjwRrTpHYSipgL63NS3BA&sig=4_ev_R-Xs8tXCVONCaiJEnFLtI0.
2. ^ *a b c d e f g h i j k* "DIBs and Their Uses" (<http://support.microsoft.com/kb/q81498/>) . *Microsoft Help and Support*. 2005-02-11. <http://support.microsoft.com/kb/q81498/>.
3. ^ "JPEG and PNG Extensions for Specific Bitmap Functions and Structures" ([http://msdn.microsoft.com/en-us/library/dd145023\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd145023(VS.85).aspx)) . [http://msdn.microsoft.com/en-us/library/dd145023\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd145023(VS.85).aspx).
4. ^ *a b* "GFF Format Summary: OS/2 Bitmap" (<http://netghost.narod.ru/gff/graphics/summary/os2bmp.htm>) . <http://netghost.narod.ru/gff/graphics/summary/os2bmp.htm>.
5. ^ "List of bitmap file types" (<http://www.file-extensions.org/filetype/extensions/name/Bitmap+image/>) . *Search File-Extensions.org*. <http://www.file-extensions.org/filetype/extensions/name/Bitmap+image/>.

External links

- For a table view of the bitmap file format (http://atlc.sourceforge.net/bmp.html#_toc381201084) at sourceforge
- Bitmap File Structure (<http://www.digicamsoft.com/bmp/bmp.html>)
- An introduction to DIBs (Device Independent Bitmaps) (<http://www.herdsoft.com/ti/davincie/imex3j8i.htm>)
- BMP test images (<http://wvnvaxa.wvnet.edu/vmswww/bmp.html>)
- Simple C++ Bitmaploader Class (<http://www.kalytta.com/bitmap.h>)
- Bitmap formats and file extensions (<http://www.file-extensions.org/filetype/extension/name/bitmap-image-files>)

Retrieved from "http://en.wikipedia.org/wiki/BMP_file_format"

Categories: Graphics file formats | Microsoft Windows multimedia technology

-
- This page was last modified on 23 May 2010 at 09:27.
 - Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.

Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

- Privacy policy
- About Wikipedia
- Disclaimers