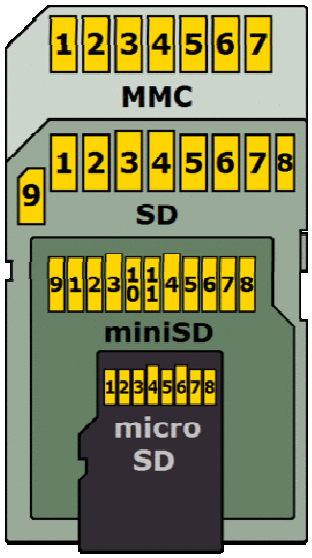
به نام خدا

آموزش میکروکنترلر AVR به زبان C (WinAVR)

(جلسه هفتم)

مقدمه:

در جلسه قبل به معرفی تایمرها و شمارنده ها و نحوه کار با آن پرداختیم. در این جلسه تصمیم داریم به معرفی مموری کارت های SD/MMC که در اصطلاح عوام به آنها رم موبایل یا رم دوربین و یا کارت حافظه گفته می شود بپردازیم و نحوه ارتباط دهی آنها به میکروکنترلر AVR را شرح دهیم. مموری کارت های موبایل ها که امروزه بسیار متداول هستند معمولاً از نوع SD بوده و مموری های دوربین ها و موبایل های قدیمی نیز از نوع MMC می باشند. نحوه تشخیص آنها بدین صورت است که MMC ها معمولاً بزرگتر بوده و برخی از انواع آنها از وسط نصف می شوند و می توان دسته آنرا جدا کرد. همچنین MMC ها هفت پایه برای اتصال به دنیای خارج دارند. MMC مخفف Multi Media Card می باشد. مموری SD ها که معمولاً در ۳ نوع معمولی، مینی SD و میکرو SD ساخته می شوند بیش از ۷ پایه برای ارتباط با بیرون داشته و متداول تر از MMC ها هستند. SD مخفف Secure Digital بوده و بدلیل اینکه میتوان اطلاعات آن را قفل کرد محبوبیت بیشتری یافته است.



آنچه برای ما اهمیت دارد این است که مموری کارت ها حداقل چندین مگابایت و حداکثر چندین گیگابایت حافظه در اختیار ما قرار می دهند و جالب اینجاست که با پروتکل SPI که یک پروتکل استاندارد برای میکروکنترلرهای AVR است براحتی می توان با این حافظه های ارزان قیمت ارتباط برقرار کرد. در این مقاله با یک مثال نحوه خواندن و نوشتن در یک مموری کارت MMC با ظرفیت 1GB را شرح می دهیم. البته فعلاً با مموری به عنوان یک آی سی حافظه برخورد می کنیم و نمی توان اطلاعات نوشته شده را در کامپیوتر دید. برای نوشتن اطلاعات بصورت فایل بایستی از روش متفاوتی استفاده کرد که در مقاله بعدی شرح داده خواهد شد. البته در این مقاله با ترفندی یک فایل متنی را ویرایش می کنیم که مسلماً تغییرات ایجاد شده توسط کامپیوتر قابل مشاهده است ولی این کار آزمایشی است و به هیچ وجه توصیه نمی شود.

نحوه کار با مموری کارت:

یکی از نکاتی که قبل از شروع کار با مموری کارت ها بایستی در نظر داشت اینست که این آی سی های حافظه برای کامپیوترها و سیستم های دیجیتال کامپیوتری مانند تلفن های هوشمند، دوربین های دیجیتال و سایر ابزارهایی طراحی شده اند که با مفهوم فایل سر و کار دارند. فایل به معنای مجموعه ای از اطلاعات می باشد که همراه با هم یک مفهوم واحد را تشکیل می دهند؛ مانند یک عکس یا یک فایل mp3. فایلها بر دو نوع اند. باینری و متنی. فایل های باینری معمولاً بصورت یکجا معنا می دهند. مثلاً اگر مجموعه بایتهایی را که موبوط به یک فایل عکس می باشد نصف کنیم نمی توانیم براحتی نصف عکس را ببینیم. و یا اگر نصف اطلاعات یک برنامه را داشته باشیم قادر به نصب آن نیستیم. اما فایل های متنی مجموعه از از کاراکترها هستند که می توان هر مقدار از فایل را براحتی مشاهده و استفاده کرد. وقتی با یک دستگاه مثلاً کامپیوتر یا یک موبایل مموری دار یک فایل متنی ایجاد می کنید این مجموعه از کاراکترها بایستی طبق یک استاندارد خاص در حافظه قرار بگیرد تا سایر دستگاه ها نیز قادر به خواندن آن باشند. برای این منظور در آدرس های ابتدای مموری کارت ها جدولی به نام FAT و ROOT Directory قرار دارد که نام فایل و آدرس شروع آن و یکسری مشخصات دیگر در آنجا قرار داده می شود و محتوای فایل نیز در آدرس خاصی از حافظه به ترتیب قرار می گیرد. وقتی که مموری در یک دستگاه دیجیتال قرار داده می شود برنامه های سیستم عامل دستگاه و بخش مدیریت حافظه ابتدا BOOT SECTOR را بررسی کرده و از آنجا آدرس ROOT DIRECTORY و جدول FAT را بدست آورده و نام فایل ها را استخراج و به کاربر نشان می دهند. حتی وقتی فایلی پاک می شود فقط نام آن پاک شده و اطلاعات اصلی در حافظه باقی می ماند (به همین دلیل است که میتوان با نرم افزارهای ریکاوری فایل های پاک شده را بازگرداند).

نکته ای که در این مقاله باید مورد توجه قرار دهیم اینست که ما قرار نیست فعلاً فایلهایی بنویسیم که توسط سایر دستگاه ها قابل خواندن باشد. پس نیازی به استفاده از جدول FAT و BOOT Sector و ROOT Directory نداریم، اما با این حال نباید اطلاعات این بخشها را خراب کنیم چون در این صورت مموری کارت بلااستفاده شده و معمولاً براحتی نمی توان آن را به حالت اولیه باز گرداند. به همین منظور بایستی از آدرس شروع دیتا شروع به نوشتن اطلاعات و خواندن نماییم. در این مقاله با کتابخانه MMC از avrlib کار می کنیم که به ما اجازه می دهد اطلاعات مختلف را به صورت بایت به بایت در آدرس های مختلف مموری کارت بنویسیم و به همین صورت اطلاعات را از حافظه بخوانیم. با استفاده از این برنامه می توان دستگاه های مختلفی مثل ضبط صوت دیجیتال، ذخیره اطلاعات دما، ذخیره یادداشت، دفترچه تلفن و هزاران وسیله کاربردی که به حافظه زیاد نیازمند است را طراحی و تولید کرد.

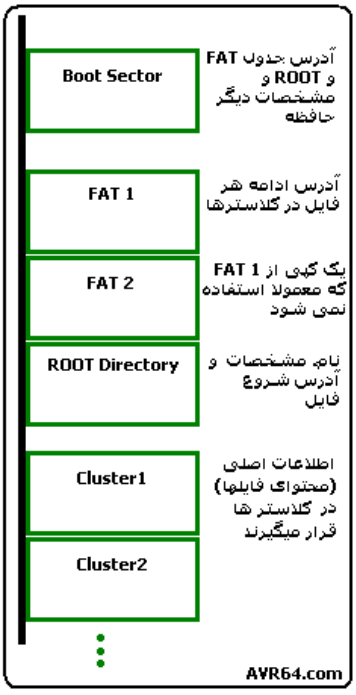
در مقاله بعدی به طور کامل در خصوص مفهوم فایل توضیح می دهیم و برنامه ای می نویسیم که بتوان بوسیله آن فایل های مختلف را ایجاد کرد که توسط کامپیوتر و سایر دستگا هها قبل خواندن باشد و نیز فایل های باینری و متنی ایجاد شده توسط کامپیوتر سایر دستگاه ها را بوسیله میکروکنترلر خواند.

ساختار داخلی مموری کارت:

در این بخش در مورد ساختار داخلی حافظه ها و جداول FAT و Boot Sector و Root Directory بیشتر توضیح می دهیم. قبل از هر چیز توجه داشته باشید که ما در این مقاله از یک مموری کارت MMC که ظرفیت آن 1GB بوده و با FAT32 فرمت شده است کار می کنیم. ضمناً فرض میکنیم که یک فایل متنی به نام salam.txt داخل مموری کپی شده است که از نوع ANSI بوده و داخل آن رشته inhellofile نوشته شده است.

هر حافظه عموماً از مجموعه ای از بایت ها ایجاد شده است اما در حافظه های متداولی مثل مموری کارت ها به دلیل اینکه آدرس دهی یک میلیارد بایت و یا بیشتر قدری مشکل به نظر می رسد نمی توان حافظه را به طور مستقیم و به صورت بایت به بایت آدرس دهی کرد. در این حافظه ها از مفهومی به نام سکتور (Sector) استفاده می شود که معمولاً شامل 512 بایت بوده و قابل آدرس دهی و خواندن و نوشتن است. با این اوصاف برای نوشتن و خواندن اطلاعات مموری کارت بایستی یک سکتور و یا به عبارتی 512 بایت را به صورت یکجا بخوانیم و بنویسیم. اکثر کتابخانه های موجود توابعی برای خواندن و نوشتن سکتورها دارند که کار را بسیار آسان می کند. اما نکته مهم بعدی اینست که مموری ها مانند آی سی های حافظه معمولی از سکتور 0 تا آخر قابل دسترسی نیستند. البته منظور از قابل دسترسی نبودن این نیست که نتوان این آدرس های ابتدایی را خواند و یا نوشت بلکه منظور اینست که نباید محتوای آدرس های ابتدایی مموری را تغییر داد چرا که مموری کارت بلااستفاده می شود و توسط کامپیوتر قابل شناسایی نخواهد بود.

یک مموری کارت فرمت شده با FAT32 مطابق شکل زیر تقسیم بندی شده است. بخش اول که بوت سکتور نامیده می شود در آدرس های ابتدایی مموری قرار گرفته است. BOOT SECTOR لزوماً در آدرس صفر نیست و در مموری کارت مورد آزمایش ما در سکتور ۶۳ قرار گرفته بود. در این قسمت تمام اطلاعات مورد نیاز در خصوص نوع فرمت کارت، حجم کل حافظه، آدرس بخش های دیگر، حجم کلاسترها و اطلاعات مفید دیگری قرار گرفته است که در بخش بعدی به طور کامل توضیح داده می شود.



بخش دوم و سوم جداول FAT نامیده می شوند. جدول شماره ۲ یکی کپی از جدول شماره ۱ بوده و برای مواقعی که جدول یک آسیب دیده باشد استفاده می شود. در این جدول ها تعدادی مدخل به اندازه تعداد کلاسترها قرار دارد که در هر مدخل شماره کلاستر بعدی ادامه یک فایل قرار گرفته است. این بخش نیز به طور مفصل توضیح داده می شود.

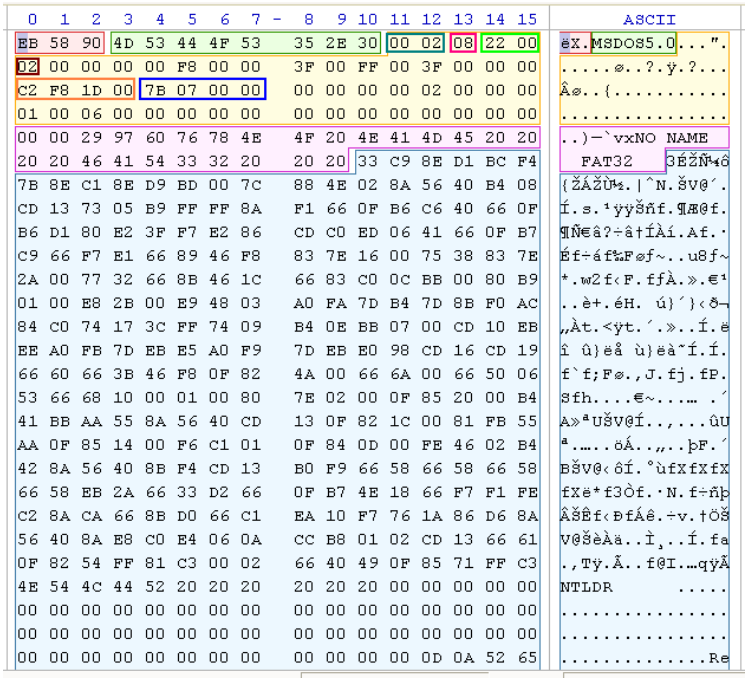
بخش چهارم که ROOT Directory نامیده می شود حاوی اسم، آدرس و مشخصات هر فایل بوده و خصوصیات، حجم و تارخ و زمان فایلها را در خود نگهداری میکند که در جای خود به طور کامل تشریح می شود.

در بخش آخر نیز کلاسترها قرار گرفته اند. هر کلاستر مجموعه ای از چند سکتور بوده (در مموری کارت مورد آزمایش ما هر کلاستر شامل ۸ سکتور بود) و محتوای فایلها در داخل این کلاسترها قرار می گیرد. کلاسترها نیز در جای خود به طور کامل توضیح داده می شوند.

تمام این قسمت ها در ابتدای یک حافظه فرمت شده با تکنولوژی FAT و خصوصاً FAT32 قرار داشته و در تمام سیستم ها مشترک می باشد. با استفاده از این سیستم هر دستگاهی می تواند فایلهای موجود در حافظه را بخواند و بنویسد. در ادامه هر یک از این قسمت ها را به طور مفصل شرح می دهیم.

بوت سکتور (BOOT SECTOR)

بوت سکتور شامل اطلاعات مختلفی از قبیل نوع سیستم عاملی که حافظه با آن فرمت شده است و سایر آدرس های لازم و مشخصات حافظه می باشد که در ادامه به طور جز به جز توضیح می دهیم. شایان ذکر است که بوت سکتور باید از سکتور صفر شروع شود ولی همیشه اینطور نیست و در مموری مورد تست در سکتور ۶۳ قرار گرفته بود. اندازه بوت سکتور تقریباً یک سکتور است ولی اندازه دقیق آن در داخل بوت سکتور نوشته شده است. در بایت های صفر تا ۱۰ مطابق شکل زیر نام سیستم یا کمپانی تولید کننده نرم افزاری که مموری را فرمت کرده است قرار می گیرد. در این شکل عبارت MSDOS5.0 قرار گرفته است که مربوط به سیستم عامل داس شرکت مایکروسافت و نسخه ۵ آن می باشد. البته سه بایت اول بصورت رمز بوده و در همه مموری ها یکسان نیست. بایت ۱۱ و ۱۲ روی هم تعداد بایتهای هر سکتور را نمایش می دهند. توجه داشته باشید که بایت هایی که با هم تشکیل یک عدد را می دهند بصورت معکوس نوشته شده اند و باید دومی را پرارزش تر و اولی را بایت کم ارزش تر به حساب بیاورید. به طور مثال بایت های ۱۱ و ۱۲ اعداد **0002** را در خود جای داده اند که بایستی به صورت **0200** هگز خوانده شود یعنی 512.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ASCII
	EB	58	90	4D	53	44	4F	53	35	2E	30	00	02	08	22	00	ëX.MSDOS5.0...".
	02	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00	.....ø..?.ÿ.?...
	C2	F8	1D	00	7B	07	00	00	00	00	00	00	02	00	00	00	Âø..{...........
	01	00	06	00	00	00	00	00	00	00	00	00	00	00	00	00	....
	00	00	29	97	60	76	78	4E	4F	20	4E	41	4D	45	20	20	..)—`vxNO NAME
	20	20	46	41	54	33	32	20	20	20	33	C9	8E	D1	BC	F4	FAT32 3ÉŽÑ¼ô
	7B	8E	C1	8E	D9	BD	00	7C	88	4E	02	8A	56	40	B4	08	{ŽÁŽÙ½.|^N.ŠV@´.
	CD	13	73	05	B9	FF	FF	8A	F1	66	0F	B6	C6	40	66	0F	Í.s.¹ÿÿŠñf.¶Æ@f.
	B6	D1	80	E2	3F	F7	E2	86	CD	C0	ED	06	41	66	0F	B7	¶Ñ€â?÷â†ÍÀí.Af.·
	C9	66	F7	E1	66	89	46	F8	83	7E	16	00	75	38	83	7E	Éf÷áf‰Føf~..u8f~
	2A	00	77	32	66	8B	46	1C	66	83	C0	0C	BB	00	80	B9	*.w2f‹F.ffÀ.».€¹
	01	00	E8	2B	00	E9	48	03	A0	FA	7D	B4	7D	8B	F0	AC	..è+.éH. ú}´}‹ð¬
	84	C0	74	17	3C	FF	74	09	B4	0E	BB	07	00	CD	10	EB	„Àt.<ÿt.´.».Í.ë
	EE	A0	FB	7D	EB	E5	A0	F9	7D	EB	E0	98	CD	16	CD	19	î û}ëå ù}ëà˜Í.Í.
	66	60	66	3B	46	F8	0F	82	4A	00	66	6A	00	66	50	06	f`f;Fø.‚J.fj.fP.
	53	66	68	10	00	01	00	80	7E	02	00	0F	85	20	00	B4	Sfh....€~.... .´
	41	BB	AA	55	8A	56	40	CD	13	0F	82	1C	00	81	FB	55	A»ªUŠV@Í..‚...ûU
	AA	0F	85	14	00	F6	C1	01	0F	84	0D	00	FE	46	02	B4	ª.…..öÁ..„..þF.´
	42	8A	56	40	8B	F4	CD	13	B0	F9	66	58	66	58	66	58	BŠV@‹ôÍ.°ùfXfXfX
	66	58	EB	2A	66	33	D2	66	0F	B7	4E	18	66	F7	F1	FE	fXë*f3Òf.·N.f÷ñþ
	C2	8A	CA	66	8B	D0	66	C1	EA	10	F7	76	1A	86	D6	8A	ÂŠÊf‹ÐfÁê.÷v.†ÖŠ
	56	40	8A	E8	C0	E4	06	0A	CC	B8	01	02	CD	13	66	61	V@ŠèÀä..Ì¸..Í.fa
	0F	82	54	FF	81	C3	00	02	66	40	49	0F	85	71	FF	C3	.‚Tÿ.Ã..f@I.…qÿÃ
	4E	54	4C	44	52	20	20	20	20	20	20	00	00	00	00	00	NTLDR .....
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
	00	00	00	00	00	00	00	00	00	00	00	00	0D	0A	52	65	..............Re

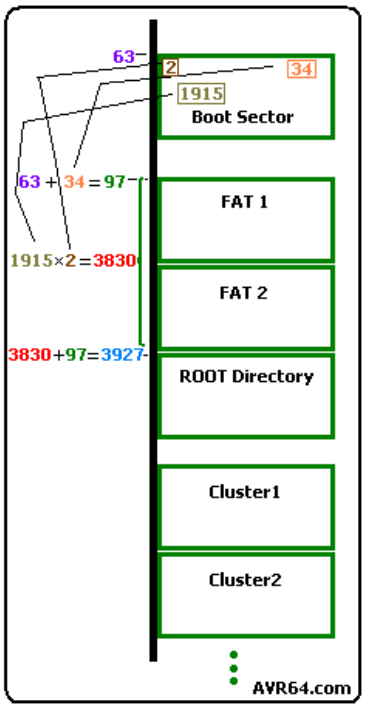
بایت ۱۳ به تنهایی تعداد سکتورهای موجود در هر کلاستر را نمایش می دهد. در اینجا 08 هگز همان 8 دسیمال است. در این صورت در این مموری هر کلاستر (که مکانی برای قرار گیری محتوای فایلهاست) از ۸ سکتور تشکیل شده است و نظر به اینکه هر سکتور ۵۱۲ بایت است داریم: ۸*۵۱۲=۴۰۹۶ یعنی هر کلاستر ۴کیلوبایت می باشد.

بایت های ۱۴ و ۱۵ روی هم آدرس شروع جدول FAT1 را نشان می دهند. البته این عدد به تنهایی نمایانگر آدرس شروع جدول فت نمی باشد. بلکه بایستی آن را با آدرسی که بوت سکتور در آن قرار گرفته است جمع کنیم تا آدرس شروع جدول

FAT بدست آید. ما به آدرسی که Boot Sector در آن قرار گرفته آفست (Offset) یا آدرس نسبی می گوییم. با توجه به اینکه بوت سکتور در این مموری در آدرس ۶۳ قرار گرفته است و عدد دسیمال موجود در دو بایت ۱۴ و ۱۵ برابر با ۳۴ (22هگز) است پس مجموع آنها می شود: ۹۷=۳۴+۶۳. با این حساب جدول FAT شماره یک در سکتور ۹۷ مموری قرار گرفته است. ضمناً با استفاده از این عدد می توانیم حجم تقریبی و رزرو شده بوت سکتور را نیز بدست آوریم.

بایت شماره ۱۶ در بوت سکتور تعداد FAT ها را نشان می دهد. از این عدد در محاسبات بعدی برای پیدا کردن جدول روت استفاده می شود. در اینجا حاوی عدد ۲ می باشد که نمایانگر وجود دو جدول FAT در حافظه است.

بایت های ۳۶ و ۳۷ و ۳۸ و ۳۹ که در جدول BOOT صفحه قبل با کادر مستطیل آبی ضخیم و پررنگ مشخص شده است بصورت معکوس عدد 0000077B را شامل می شود که اندازه هر FAT بر حسب سکتور است. در این مموری عدد فوق برابر با ۱۹۱۵ بوده و از آنجاییکه دو جدول FAT بصورت پشت سر هم قرار گرفته اند این عدد ضربدر ۲ می شود ۳۸۳۰ و اندازه هر دو FAT را نشان می دهد. البته بجای استفاده از عدد ۲ باید از عددی که در بایت ۱۶ بوت سکتور قرار گرفته و تعداد FATها را نشان می دهد استفاده کرد. در صورتی که عدد حاصل یعنی ۳۸۳۰ را با آدرس شروع FAT1 جمع کنیم آدرس انتهای FAT2 بدست می آید که همان ابتدای ROOT Directory می باشد.



پس تا اینجا اطلاعات مفیدی را از بوت سکتور استخراج کردیم که به وسیله آنها توانستیم آدرس شروع Root را پیدا کنیم. Root جدولی مفیدی است که در آن نام فایلها و مشخصات آنها و نیز آدرس شروع هر فایل قرار گرفته است. با استفاده از آدرس شروع فایل می توانیم تشخیص دهیم که چه نقطه ای از مموری به کلاسترها و در واقع بخش دیتای مموری اختصاص یافته است و از آن نقطه به بعد به راحتی در مموری بنویسیم و اطلاعات را بخوانیم و هیچ گونه نگرانی از بابت خراب شدن جداول ابتدای مموری نداشته باشیم.

جدول FAT (File Allocation Table)

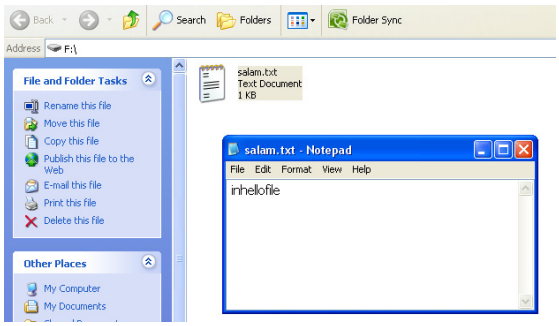
جدول FAT شامل تعدادی مدخل به اندازه تعداد کلاسترها می باشد و برای آدرس دهی کلاسترهای ادامه فایل به کار می رود. نظر به اینکه در این مقاله فقط به آدرس شروع فایل نیاز داریم توضیح بیشتر در خصوص این جدول را به مقاله بعدی که صرفاً در خصوص FAT می باشد موکول می کنیم.

جدول Root Directory

جدول Root مهمترین بخش این مقاله می باشد. در این جدول نام فایلهای موجود در مموری بعلاوه مشخصات و آدرس نسبی شروع فایل درج شده است. نمایی از دایرکتوری ریشه مربوط به مموری کارت مورد آزمایش را در تصویر زیر مشاهده می کنید. در این مموری فایل متنی به نام salam.txt در داخل مموری کپی شده بود.

0	1	2	3	4	5	6	7	-	8	9	A	B	C	D	E	F	ASCII
53	41	4C	41	4D	20	20	20		54	58	54	20	18	C0	5C	53	SALAM TXT .À\S
6E	45	6E	45	00	00	5A	53		6E	45	03	00	0B	00	00	00	nEnE..ZSnE.....
00	00	00	00	00	00	00	00		00	00	00	00	00	00	00	00	.....
00	00	00	00	00	00	00	00		00	00	00	00	00	00	00	00	.....

در این جدول به ازای هر فایل ۳۲ بایت اشغال می شود. ۱۱ بایت اول نام و پسوند فایل، بایت شماره ۱۲ بصورت بیت به بیت شامل مشخصات فایل از قبیل Hidden و ... که در جلسه بعدی توضیح می دهیم و بایت های بعدی تاریخ و زمان ساخت و دسترسی و ویرایش می باشد که این بخش نیز در جلسه FAT به طور مفصل توضیح داده می شود. چهار بایت آخر نیز حجم فایل را نشان می دهد. اما آنچه در این جلسه برای ما مهم است پیدا کردن آدرس اولین فایل می باشد تا به کمک آن آدرس شروع کلاسترها (یعنی بخش داده ای) در مموری کارت را پیدا کنیم. بایت های ۲۶ و۲۷ برای این منظور تدارک دیده شده اند. این دو بایت روی هم عددی را نشان می دهند که به کمک آن و فرمول زیر می توان آدرس دقیق اولین سکتور حاوی محتوای اولین فایل را بدست آورد.



همانطوریکه از جدول بالا مشخص است عدد 0300 در این دوبایت قرار گرفته است که بصورت معکوس بوده و صحیح آن 0003 هگز می باشد و مقدار دسیمال آن 3 است. نام این عدد را آدرس نسبی می گذاریم. طبق فرمول زیر می توان آدرس سکتور شروع محتوای فایل یا همان آدرس شروع اولین کلاستر را بدست آورد:

آدرس شروع اولین کلاستر = ((آدرس نسبی منهای ۲) ضربدر (تعداد سکتور در کلاستر)) + آدرس شروع دایرکتوری ریشه

آدرس شروع اولین کلاستر = ((۲-۳) ضربدر (۸)) + ۳۹۲۷

آدرس شروع اولین کلاستر = ((۱) ضربدر (۸)) + ۳۹۲۷

آدرس شروع اولین کلاستر = ۸ + ۳۹۲۷

آدرس شروع اولین کلاستر = ۳۹۳۵

در تصویر زیر محتوای سکتور ۳۹۳۵ را مشاهده می کنیم که محتوای فایل در آن قرار گرفته است.

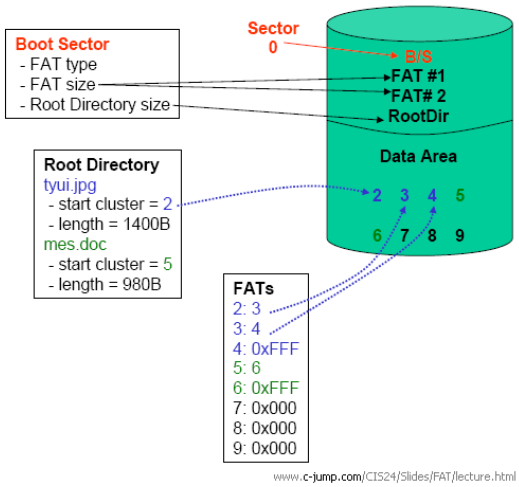
0	1	2	3	4	5	6	7	-	8	9	A	B	C	D	E	F	ASCII
69	6E	68	65	6C	6C	6F	66		69	6C	65	00	00	00	00	00	inhellofile.....
00	00	00	00	00	00	00	00		00	00	00	00	00	00	00	00	.....

کلاسترها Clusters

کلاسترها مجموعه ای از چند سکتور می باشند. در مموری مورد آزمایش هر کلاستر از ۸ سکتور تشکیل شده بود که با یک محاسبه ساده و بر اساس اینکه هر سکتور ۵۱۲ بایت می باشد می توان نتیجه گرفت که حجم هر کلاستر ۴ کیلوبایت است. در بخش FAT گفتیم که به ازای هر کلاستر یک مدخل در جدول FAT قرار گرفته است، با توجه به اینکه حجم جدول فت بسیار کم و در این مموری فقط ۱۹۱۵ سکتور است تعداد خانه های کمی قابل آدرس دهی می باشد و اگر قرار بود به ازای هر سکتور یک مدخل در جدول وجود داشته باشد حجم جدول FAT بیش از اندازه بزرگ می شد.

هر کلاستر ممکن است بخشی از یک فایل را در خود جای دهد. البته در صورتی که حجم فایل از حجم کلاستر کوچکتر باشد کل کلاستر به فایل اختصاص می یابد و بقیه فضای آن هدر می رود. اگر هم حجم فایل بیشتر از کلاستر باشد بقیه فایل در کلاستر دیگر قرار می گیرد و در اینجا با مراجعه به جدول FAT می توان ادامه فایل را پیدا کرد (همیشه یک فایل در کلاسترهای پشت سر هم قرار نمی گیرد، مثلا وقتی فایلی پاک می شود و دوباره یک فایل دیگر در مموری نوشته می شود فایل جدید در حفره های ایجاد شده یا همان کلاسترهای خالی ذخیره شده و با ابزاری مثل Disk Defragment می توان فایل ها را مرتب کرد).

تصویر زیر از یک وبسایت خارجی کپی شده و گویای مطالب بالاست. اگر ۹۰ درصد این مقاله را بر مبنای مقاله مهندس علی تروشه نوشته باشیم، به جرات می توان گفت ۱۰درصد بقیه این مقاله به کمک تصویر زیر تالیف شده است.



مبحث کلاستر ها و نحوه ارتباط با جدول FAT به جلسه بعدی اختصاص دارد. به همین دلیل بحث بیشتر در خصوص مموری کارت را در همین جا خاتمه می دهیم و در بخش بعدی این مقاله به نحوه ارتباط مموری کارت با میکروکنترلر ATmega8 از خانواده AVR می پردازیم و برنامه ای به زبان C و با استعانت از کتابخانه avrlib برای این پروژه می نویسیم که تمام کارها را به صورت خودکار انجام داده و تمام آدرس های لازم، نام فایل متنی و محتوای آن را بر روی نمایشگر کریستال مایع 2*16 نشان می دهد و در انتها فایل را ویرایش کرده و کاراکتر اول محتوای آن را تغییر می دهد.

برنامه:

برنامه این جلسه به طور کامل در پوشه همراه مقاله آورده شده است. در صورتی که همراه با این مقاله هیچ کدی قرار نگرفته باشد توصیه می شود نسخه اصلی این مقاله را از بخش آموزش وب سایت avr64.com دانلود نمایید.

```
// mmc and AVR By Behnam Zakizadeh with WinAVR gcc compiler @ 31.Oct.2014 [1393/08/09]
// website: AVR64.com
// license: freeware

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "global.h"
#include "rprintf.h"
#include "timer.h"
#include "lcd.h"
#include "uart.h"
#include "debug.h"
#include "spi.h"
#include "mmc.h"

int main(void)
{
	u32 sector=0;
	u08 buffer[0x200];
	u08 reset_status = 0;
	u08 read_status = 0;
	u32 offset=0;
	u32 rootDirectory=0;
	u32 reservedSector=0;
	u32 sectorPerFat=0;
	u32 numberOfFat=0;
	u32 sectorPerCluster=0;
	u32 addrOfFirstFileinFat=0;
	u32 realAddrOfFirstFile=0;
	u08 wrstate=0;

	_delay_ms(1000); // very important! (wait for mmc startup)
	lcdInit();
	rprintfInit(lcdDataWrite);
	rprintf("Reset MMC");
	_delay_ms(1000);
	lcdClear();

	mmcInit();
	reset_status = mmcReset();

	//uartInit();
	//uartSetBaudRate(115200);
	//rprintfInit(uartSendByte);
	//rprintf("\r\nWelcome to AVRlib!\r\n");

	lcdGotoXY(0,0);
	rprintf("reset_status=%d", reset_status);
	_delay_ms(1000);
	lcdClear();

	for(u32 i=0;i<1000;i++)
	{
		lcdClear();
		read_status = mmcRead(i, buffer);
		lcdGotoXY(0,0);
		rprintf("Sector=%d-rd=%d", i, read_status);
		lcdGotoXY(0,1);
		rprintf("data=%c%c%c%c%c%c%c%c",
buffer[3],buffer[4],buffer[5],buffer[6],buffer[7],buffer[8],buffer[9],buffer[10]);

		if(buffer[3]=='M' && buffer[4]=='S' && buffer[5]=='D' && buffer[6]=='O' &&
buffer[7]=='S' && buffer[8]=='5'  && buffer[9]=='.'  && buffer[10]=='0')
		{
```

```
                        sector = i;
                        break;
                }
                //if(buffer[3]=='a' && buffer[4]=='n' && buffer[5]=='d' && buffer[6]=='r' &&
buffer[7]=='o' && buffer[8]=='i'  && buffer[9]=='d'  && buffer[10]==' ')
                //      break;

                _delay_ms(50);
        }

        _delay_ms(10000);

        // claculate start address of file
        offset = sector;
        reservedSector = (reservedSector << 8) + buffer[15];
        reservedSector = (reservedSector << 8) + buffer[14];

        clr();
        rprintf("rsv=%d", reservedSector);

        clr2();
        rprintf("offset=%d", offset);

        _delay_ms(1000);

        numberOfFat = buffer[16];
        sectorPerFat = (sectorPerFat << 8) + buffer[39];
        sectorPerFat = (sectorPerFat << 8) + buffer[38];
        sectorPerFat = (sectorPerFat << 8) + buffer[37];
        sectorPerFat = (sectorPerFat << 8) + buffer[36];

        clr();
        rprintf("numFAT=%d", numberOfFat);

        clr2();
        rprintf("SecPrFAT=%d", sectorPerFat);

        _delay_ms(1000);

        rootDirectory = reservedSector+offset+(sectorPerFat*numberOfFat);
        sectorPerCluster = buffer[13];

        clr();
        rprintf("RootDir=%d", rootDirectory);

        clr2();
        rprintf("SecPrClustr=%d", sectorPerCluster);

        _delay_ms(1000);

        mmcRead(rootDirectory, buffer);
        clr();
        rprintf("Fnme=%c%c%c%c%c%c%c%c%c%c%c",
buffer[0],buffer[1],buffer[2],buffer[3],buffer[4],buffer[5],buffer[6],buffer[7],buffer[8],buffer[
9],buffer[10],buffer[11]);

        addrOfFirstFileinFat = (addrOfFirstFileinFat << 8) + buffer[27];
        addrOfFirstFileinFat = (addrOfFirstFileinFat << 8) + buffer[26];

        clr2();
        rprintf("filAdOfst=%d", addrOfFirstFileinFat);

        _delay_ms(1000);

        realAddrOfFirstFile = rootDirectory + ( (addrOfFirstFileinFat-2)*sectorPerCluster );

        clr();
        rprintf("fileAddr=%d", realAddrOfFirstFile);

        mmcRead(realAddrOfFirstFile, buffer);

        clr2();
```

```
        rprintf("t=%c%c%c%c%c%c%c%c%c%c%c",
buffer[0],buffer[1],buffer[2],buffer[3],buffer[4],buffer[5],buffer[6],buffer[7],buffer[8],buffer[
9],buffer[10],buffer[11]);

        buffer[0] = 'w';// replace first char of file with w

        wrstate = mmcWrite(realAddrOfFirstFile, buffer);// write changes to file

        clr();
        rprintf("wrstate=%d", wrstate);

        return 0;
}

void clr(void)
{
        lcdGotoXY(0,0);
        rprintf("                ");
        lcdGotoXY(0,0);
}

void clr2(void)
{
        lcdGotoXY(0,1);
        rprintf("                ");
        lcdGotoXY(0,1);
}
```

در این برنامه ابتدا یک تاخیر یک ثانیه ای ایجاد شده است تا پس از اتصال تغذیه میکرو، مموری فرصت پیدا کند که به کار بیفتد و آماده ارتباط شود. اگر این تاخیر در برنامه قرار داده نشود و بلافاصله با میکرو ارتباط برقرار کنیم منجر به شکست شده و برنامه به هیچ وجه کار نخواهد کرد. (این مشکل باعث تاخیر زیادی در نوشتن این مقاله شد).

در بخش بعدی با دستور mmcInit(); مموری را آماده کرده و با دستور mmcReset(); آن را ریست می نماییم. بهتر است خروجی دستور ریست را در متغیر قرار داده وآن را بر روی نمایشگر نشان دهیم چرا که در صورت موفقیت عدد 0 و در صورت بروز خطا عدد دیگری را بر می گرداند که به کمک آن می توان مدار را عیب یابی نمود.

در بخش بعدی با ایجاد یک حلقه از سکتور 0 تا 1000 به دنبال بوت سکتور می گردیم. نیازی به جستجوی ۱۰۰۰ سکتور اول نیست و فقط برای اطمینان خاطر نوشته شده است. در داخل این حلقه با دستور mmcRead(i, buffer); سکتور متناسب با متغیر i خوانده شده و کل محتوای آن در متغیر آرایه ای ۵۱۲ عنصری از نوع بایت قرار داده می شود. شایان ذکر است در WinAVR متغیر های u08 به معنی unsigned هشت بیتی یعنی همان Byte می باشد که در هدر فایل avrlibtypes.h در این خصوص توضیح داده شده است. سپس محتوای بایت های سوم تا دهم با عبارت MSDOS5.0 مقایسه می شود، اگر این عبارت در سکتور پیدا شد به معنای بوت سکتور می باشد. البته این کار کاملاً ابداعی بوده و ممکن است با حافظه هایی که با سیستم عامل دیگری مثل لینوکس یا اندروید فرمت شده باشند جواب ندهد. به طور مثال اگر یک مموری کارت با استفاده از تبلت یا موبایل اندرویدی فرمت شده باشد عبارت android را بجای MSDOS مشاهده می کنید. به هر حال می توان رشته های دیگری را در این سکتور بررسی کرد و یا تمام رشته های ممکن را جستجو نمود. در صورتی که رشته فوق یافت شود شماره سکتور جاری که همان مقدار i می باشد در متغیر sector قرار گرفته و از حلقه خارج می شود.

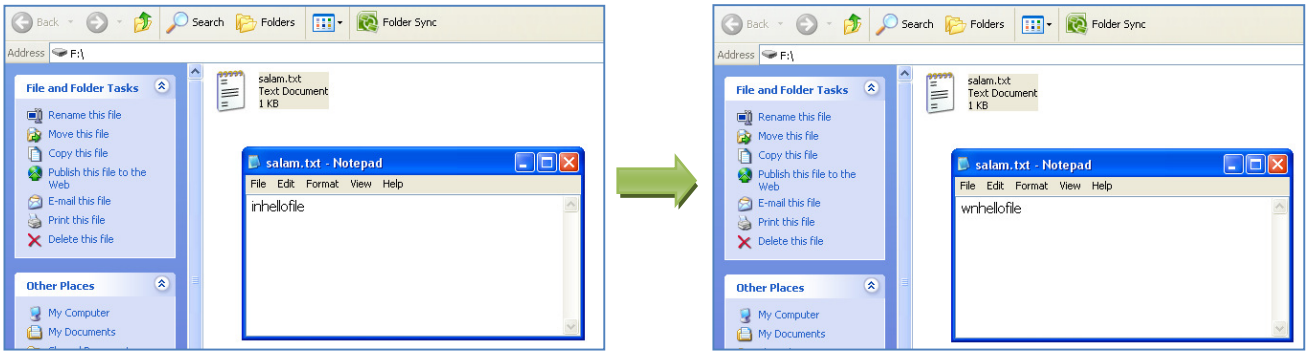
در خارج از حلقه مقدار Sector در متغیری به نام offset قرار می گیرد که نمایانگر آفست شروع بوت سکتور بوده و در محاسبات بعدی برای پیدا کردن سایر آدرس ها از آن استفاده می شود. بقیه برنامه مطابق با توضیحات ابتدای این مقاله برای پیدا کردن آدرس ها نوشته شده است و چیزی جز جمع و تفریق و ضرب آدرس ها نیست. تنها بخش مهم که ممکن است نیاز به توضیح داشته باشد قطعه کد های شبیه زیر است:

```
sectorPerFat = (sectorPerFat << 8) + buffer[39];
sectorPerFat = (sectorPerFat << 8) + buffer[38];
sectorPerFat = (sectorPerFat << 8) + buffer[37];
sectorPerFat = (sectorPerFat << 8) + buffer[36];
```

در چنین بخش هایی با توجه به اینکه محتوای خانه های بافر از نوع بایت می باشد برای تبدیل چند بایت به یک عدد بزرگتر (در اینجا تبدیل ۴ عدد ۸ بیتی به یک عدد ۳۲ بیتی) بایستی مطابق با دستور بالا عمل کنیم. در این سلسل دستورات اعداد به ترتیب پرارزش به کم ارزش با عدد مقصد جمع شده و هر بار محتوای متغیر مقصد ۸ بیت به چپ شیفت داده می شود تا در نهایت تمام ۴ بایت در متغیر ۳۲ بیتی قرار بگیرند.

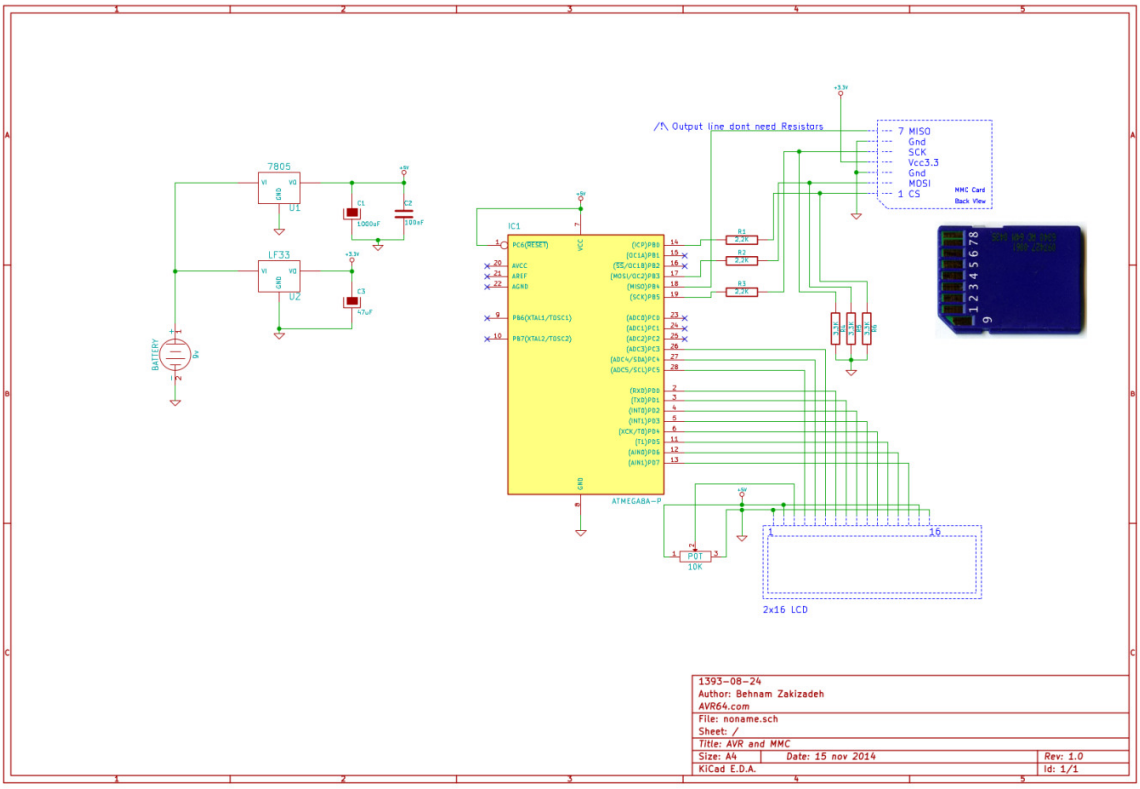
توابع clr(); و clr2(); نیز به ترتیب برای پاک کردن خط اول و دوم نمایشگر بکار می روند تا بتوان آدرس های بدست آمده را در این خطوط نشان داد.

در بخش آخر این برنامه محتوای فایل موجود در مموری به نمایش در آمده و سپس کاراکتر اول فایل با حرف w جایگزین شده و به کمک دستور mmcWrite این تغییر در مموری نوشته می شود. در صورتی که مموری به کمک کامپیوتر بررسی شود می توان این تغییر را مشاهده کرد. همانطوریکه در ابتدای این مقاله ذکر شده بود فرض بر این است که مموری با FAT32 فرمت شده و فایل متنی به نام salam.txt با محتوای inhellofile در داخل مموری کپی شده باشد.



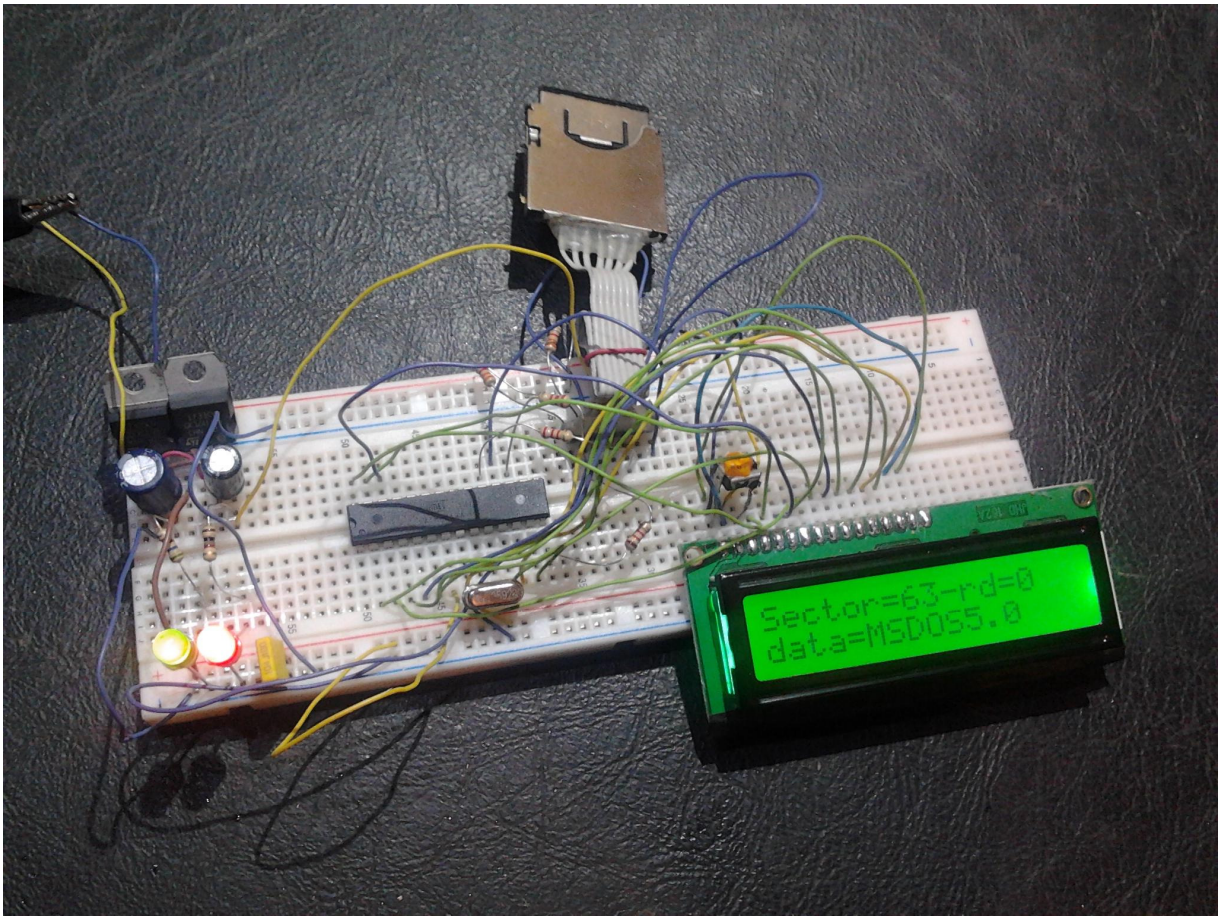
شماتیک:

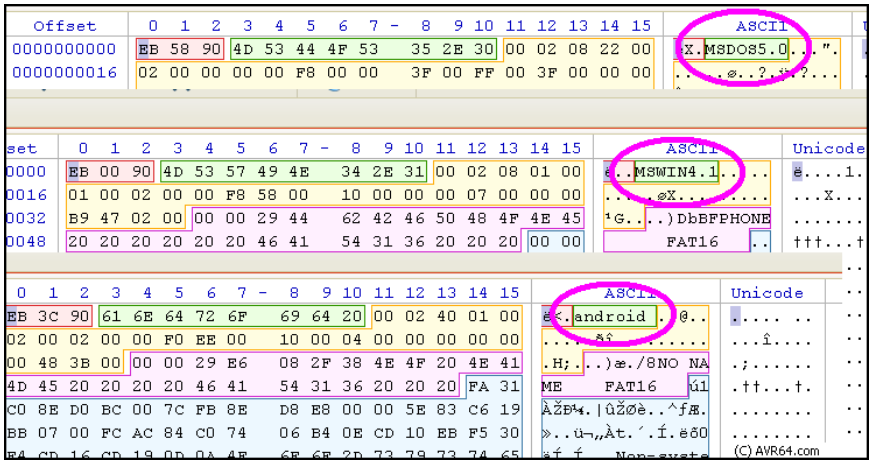
شماتیک پروژه ارتباط با مموری کارت را در شکل زیر ملاحظه می فرمایید که با استفاده از نرم افزار قدرتمند و رایگان کیکد (KiCad) ترسیم شده است.



سخت افزار:

تصویری از پروژه در حال تست را در شکل زیر مشاهده می فرمایید.





در این مقاله یادگرفتیم که چگونه موری کارت ها را به میکروکنترلر متصل کنیم و اطلاعات را از آدرسهای خاص حافظه بخوانیم و در آن بنویسیم. در جلسه بعد با مفهوم فایل آشنا شده و با نوشتن یک File Explorer کوچک با AVR نحوه کار با فایل ها و پوشه ها را بررسی می کنیم.

ادامه دارد...

منابع:

http://aminelec.persianblog.ir/post/4/

http://electrorc.blogfa.com/post-28.aspx

http://www.c-jump.com/CIS24/Slides/FAT/lecture.html

http://en.wikipedia.org/wiki/Secure_Digital

(توجه: این آموزش بر اساس مقاله dominoی مهندس علی تروشه نوشته شده که بدینوسیله از زحمات ایشان قدردانی می شود)
http://electrorc.blogfa.com

این مقاله به همراه سایر مقالات منابع بالا در پوشه این آموزش قرار گرفته است.

پایان جلسه هفتم آموزش AVR

مولف: بهنام زکی زاده
www.avr64.com

۲۴ آبان ۱۳۹۳

آخرین ویرایش: ۱۰ اسفند ۱۳۹۴ ✓