
بسم الله الرحمن الرحیم

مرکز آموزش علمی کاربردی شرکت صنعتی کوشا (واحد تهران)

رشته :

مکانیک خودرو

عنوان :

ساخت ولت متر دیجیتال برای جریانهای مستقیم

استادراهنما :

مهندس رضا اصفهانی

تهیه کننده :

محمد جهانیان

تابستان ۹۳

گزارش عملکرد مدار ولتمتر DC

چکیده :

در این گزارش به بررسی اجزا و عملکرد مدار ولتمتر DC با استفاده از میکروکنترلر AVR atmega8 پرداخته شده است.

عملکرد این مدار به این صورت است که مقدار ولتاژ موجود بین دو ترمینال خود را با دقت صدم ولت اندازه گیری کرده و بر روی سون سگمنت نمایش می دهد. برای این منظور از مدارات ADC داخلی میکروکنترلر استفاده شده است.

از آنجا که ولتمتر به صورت موازی در مدار قرار می گیرد مقدار مقاومت داخلی آن باید بسیار زیاد باشد به گونه ای که تغییری در جریان های مدار ایجاد نکند، به همین دلیل مقدار مقاومت داخلی آن برابر 1100Kohm یا تقریبا 1Mohm می باشد که برای ولتمتر مقدار قابل قبولی است.

بازه ی ولتاژ قابل اندازه گیری توسط این ولتمتر ۱۰ میلی ولت تا ۳۰ ولت با دقت ۱۰ میلی ولت می باشد.

فهرست

معرّفی

معرّفی :

هسته ی اصلی این مدار در واقع مدار مبدّل آنالوگ به دیجیتال یا ADC موجود در میکروکنترلر می باشد. با توجه به اینکه مقدار ولتاژ ورودی تا ۳۰ ولت می باشد و مقدار تغذیه ی دیجیتال و آنالوگ میکروکنترلر برابر 3.3v می باشد، در مرحله ی اول سیگنال ورودی از یک تقسیم مقاومتی عبور داده می شود تا مقدار آن 1/10 برابر شود. مقاومتی که در این تقسیم مقاومتی استفاده می شود تقریبا همان مقاومت داخلی ولتمتر را تشکیل می دهد، در نتیجه به منظور افزایش مقدار مقاومت داخلی ولتمتر از دو مقاومت سری شده ی 1Mohm و 100Kohm به منظور تقسیم مقاومتی استفاده شده است. پس از عبور ولتاژ ورودی و تضعیف آن با استفاده از تقسیم مقاومتی، این ولتاژ به ورودی مدار ADC میکروکنترلر اعمال می شود.

قبل از ادامه ی صحبت در مورد عملکرد مدار، ابتدا کمی در مورد مبدل های آنالوگ به دیجیتال یا همان مدارات ADC صحبت می کنیم.

یک ADC یا مبدل آنالوگ به دیجیتال مقادیر آنالوگ را به کدهای دیجیتال تبدیل می کند تا امکان پردازش دیجیتال سیگنال های ورودی وجود داشته باشد.سیگنالهای آنالوگ دارای طیف پیوسته ای هستند.کار یک مبدل آنالوگ به دیجیتال ، تبدیل یک بازه به چند زیر بازه است که به هر کدام از این زیر بازه ها معمولا یک پله گفته می شود که به هر پله یک کد دیجیتال نسبت داده می شود. یک مبدل آنالوگ به دیجیتال که به اختصار A/D،ADC هم نامیده می شود یک مدار الکترونیکی داخلی است که سیگنال های پیوسته را به کد های دیجیتالی گسسته تبدیل می کند. کد دیجیتال خروجی می تواند از رویه کدهای مختلفی مانند سیتم دودویی یا مکمل دوم باینری استفاده کند.

متداول ترین انواع adc به قرار زیر است

۱- مبدل توع شمارشی (counting analog digital convertor)

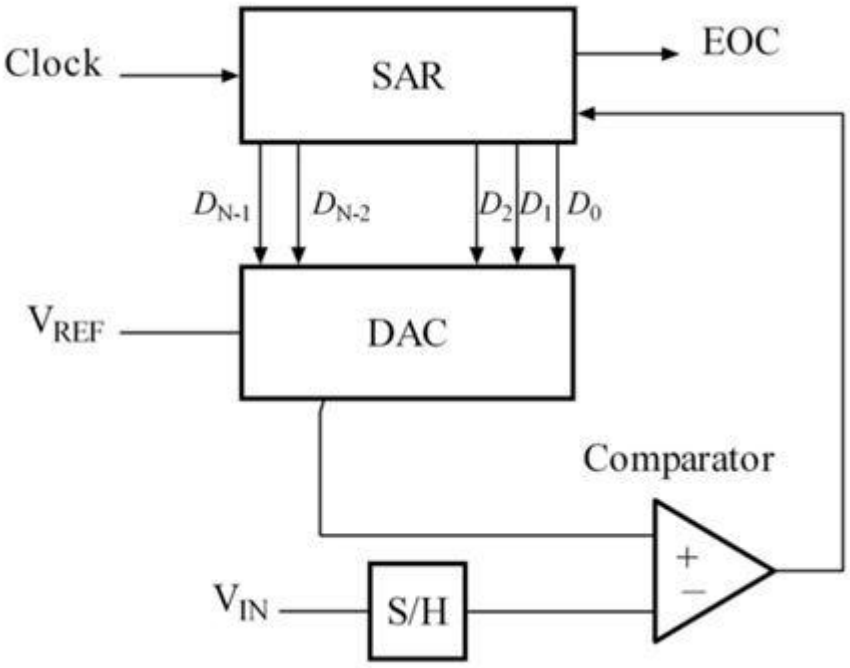
۲- مبدل نوع تقریب های متوالی (successive-approximation convertor)

۳- مبدل با مقایسه موازی (parallel-comparator adc)

۴- مبدل دو شیبه (dual-slop or ratiometric adc)

مدار ADC داخلی میکروکنترلر های AVR از نوع تقریب متوالی می باشد.

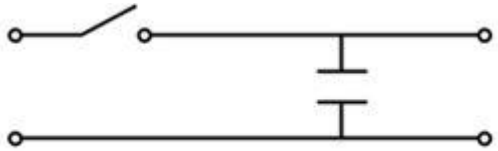
بلوک دیاگرام ساده شده ی این مبدل به صورت شکل زیر است:



شکل بلوک دیاگرام تبدیل با روش تقریب های متوالی

:SampleandHold

مبدل انالوگ به دیجیتال برای تبدیل یک نمونه برداری انالوگ به مقدار باینری متناظر با ان نیاز به یک ورودی Stableدارد که از این طریق مدار Sample and Holdایجاد می شود. در شکل زیر یک نمونه ی بسیار ساده از ان را مشاهده می کنید، سیگنال ورودی را با هر نمونه ی برداشته شده به خازن وصل می کند و خازن نیز مقدار ولتاژ را تا نمونه ی بعدی ثابت نگه می دارد.



شماتیک Sample and Hold

Successive Approximation Register:

این رجیستر مقدار تقریب زده شده ی دیجیتال را برای مقایسه به ADCمی دهد.

الگوریتم تبدیل:

ابتدا رجیستر SARبا عدد باینری 10000000بارگذاری می شود و این عدد توسط DACبا مقدار انالوگ ورودی مقایسه می شود. در صورتی که عدد مقایسه شده بزرگتر باشد و خروجی مقایسه کننده باعث می شود، SARبیت MSBرا پاک کرده وبیت قبل از ان را پاک می کند و در نتیجه

عدد 01000000در DACبارگذاری می شود. در صورتی عدد مقایسه شده کوچکتر باشد خروجی مفایسه کننده باعث می شود بیت کوچکتر نیز یک شده و در نتیجه عدد 11000000در ورودی DACبارگذاری شود. این عمل تا پیدا شدن مقدار انالوگ ادامه داشته و در این زمان بیتEndofConversionبه نشانه پایان تبدیل یک می شود.

در مورد ADC ها آنچه بسیار مهم است دو پارامتر از آنها می باشد :

۱-تعداد بیت کد دیجیتال خروجی

۲-فرکانس نمونه برداری

۱- تعداد بیت کد دیجیتال خروجی : مدارات ADC سیگنال آنالوگ ورودی را به کد دیجیتال تبدیل می کنند. هر قدر تعداد این بیت ها بیشتر باشد، دقت ADC بیشتر می باشد. به عنوان نمونه اگر در یک ADC هشت بیتی ولتاژ ورودی بین ۰ تا ۵ ولت تغییر کند، مدار ADC ای بازه را به ۲۵۶ قسمت تقسیم نموده و به هر یک، یک کد از ۰ تا ۲۵۵ اختصاص می دهد. اما در یک ADC ده بیتی این باز به ۱۰۲۴ قسمت تقسیم شده و به هر یک یک کد از ۰ تا ۱۰۲۳ اختصاص می دهد که این امر باعث کوچکتر شدن بازه های تقسیم بندی و دقیق تر شدن عملکرد ADC می شود.

۲- فرکانس نمونه برداری : این پارامتر در واقع مشخص می کند که ADC با چه سرعتی از سیگنال آنالوگ ورودی نمونه برداری می کند و در واقع در هر ثانیه چند کد دیجیتال از سیگنال آنالوگ ورودی تهیه می کند. هر قدر فرکانس نمونه برداری بیشتر باشد، عملکرد ADC دقیق تر خواهد بود زیرا در هر ثانیه نمونه های بیشتری را از سیگنال ورودی برداشت می کند که این امر باعث می شود هنگام تولید مجدد سیگنال آنالوگ از مقادیر دیجیتال سیگنال خروجی به سیگنال آنالوگ اصلی اوّلیه نزدیکتر باشد.

نکته ی دیگری که در مورد سنسور های مدار وجود دارد، این است که صفحه ی لمسی به کار رفته در این مدار نیز در واقع به نوعی یک سنسور است که در بخش مربوطه در مورد آن صحبت خواهیم کرد.

در این پروژه از ADC داخلی با رزولوشن ۱۰ بیتی و با فرکانس کاری 125KHz (بهترین فرکانس کاری مدار ADC داخلی میکروکنترلر های AVR طبق گفته ی دیتاشیت آنها) استفاده شده است و ولتاژ رفرنس آن نیز 3.3V می باشد. به این ترتیب هر پله از خروجی ADC برابر 3.3/1024=3.22mv می باشد. به عبارتی هر ۳.۲۲ میلی ولت افزایش ولتاژ ورودی باعث می شود که کد دیجیتال ولتاژ خروجی یک واحد افزایش یابد.

ولتاژ ورودی پس از تضعیف (1/10 برابر شدن) وارد مدار ADC شده و تبدیل به کد دیجیتال می شود. با ضرب عدد تولید شده در 3.22*10 مقدار ولتاژ واقعی ورودی بر حسب mV بدست می آید. این کار درون

برنامه ی درون میکرو کنترلر انجام می شود. سپس عدد تولید شده به واحد ولت برگردانده می شود و بر روی سون سگمنت ها با استفاده از روش جاروب کردن نمایش داده می شود. سون سگمنت ها در روتین وقفه ی تایمر Refresh می شوند.

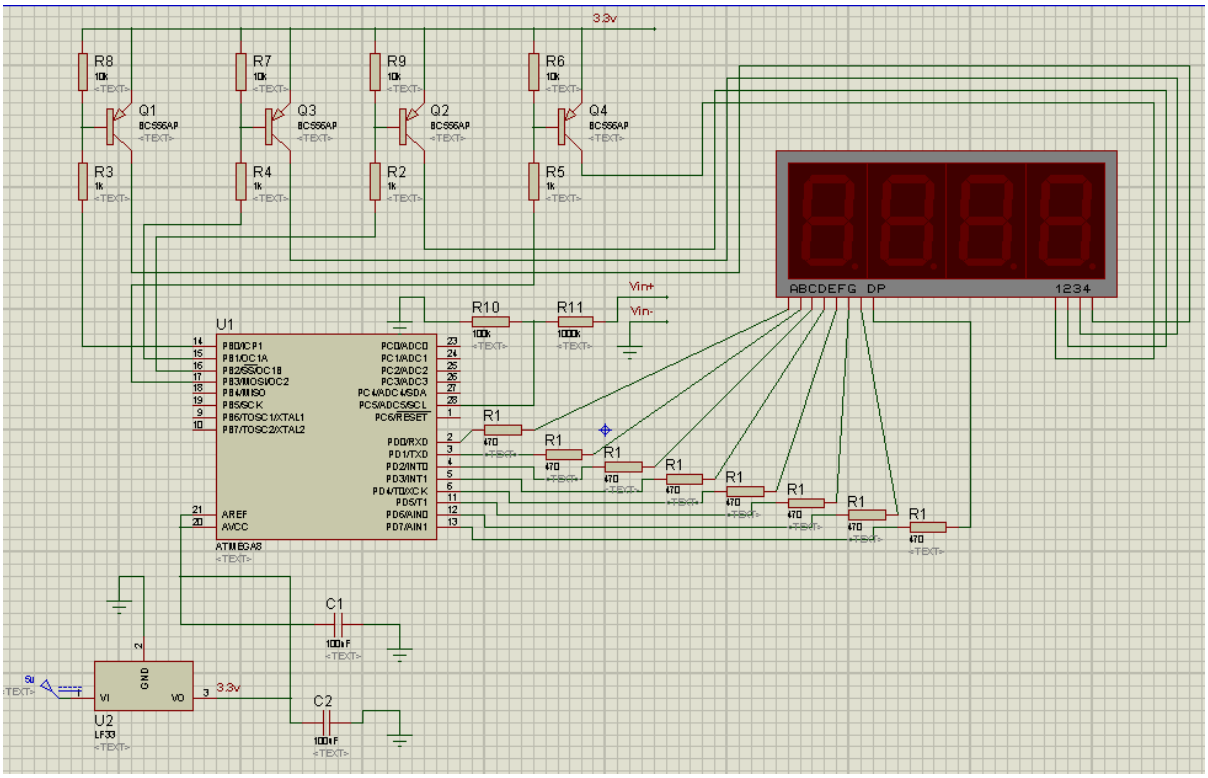
کار خواندن ADC و تبدیل آن به ولت و نمایش آن بر روی سون سگمنت ها در یک حلقه ی بی پایان در برناه انجام می پذیرد.

سخت افزار کلّی

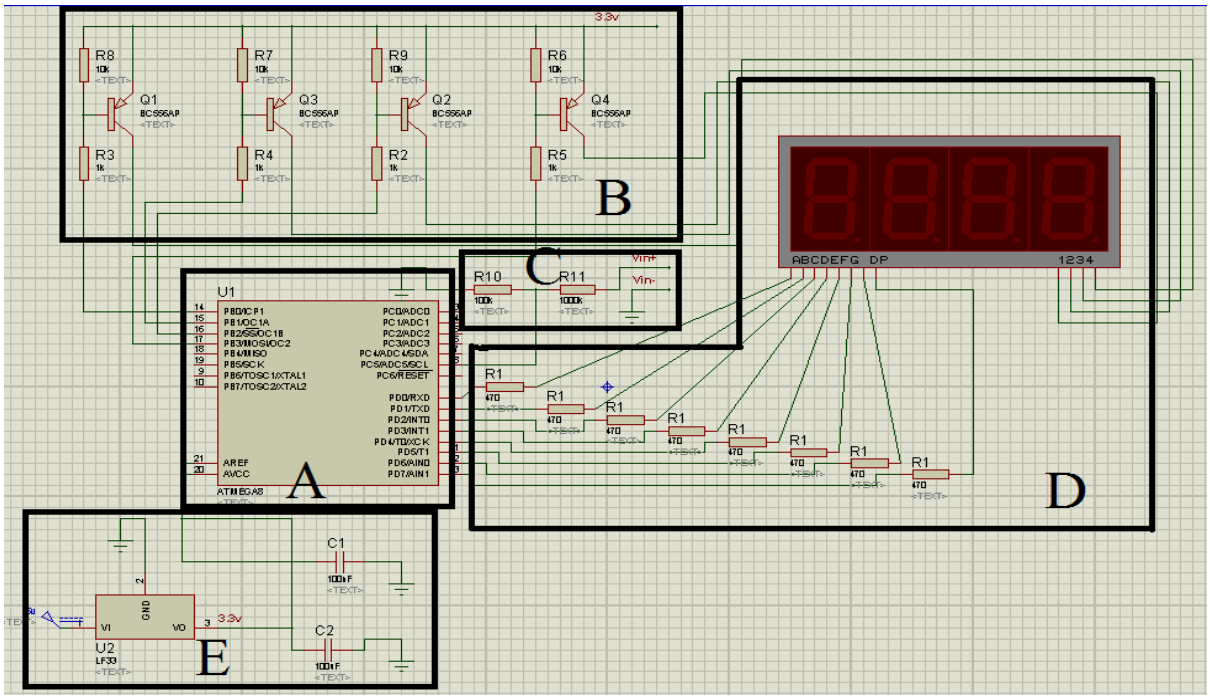
مدار

سخت افزار کلّی مدار :

شماتیک کلّی مدار این پروژه به صورت شکل زیر می باشد.



این مدار را می توان به صورت زیر به پنج قسمت اصلی تقسیم نمود :



با توجه به شکل فوق قسمت A در واقع میکروکنترلر AVR atmega8 می باشد که کنترل همه ی اجزای مدار را به عهده دارد و نرم افزار موجود در آن در بخش ۴ ارائه می شود. قسمت B در واقع چهار عدد ترانزیستور PNP هستند که جریان پین های مشترک چهار سون سگمنت را تامین می کنند. قسمت C همان تقسیم ولتاژی است که برای تضعیف در مورد آن صحبت کردیم. قسمت D سون سگمنت چهارتایی است که برای نمایش نتیجه به کار می رود. قسمت E نیز رگولاتور LF33 است که ولتاژ ورودی را به 3.3v تقسیم می کند.

از بین تمام قسمت های فوق در بخش بعد فقط در مورد قسمت A صحبت می کنیم زیرا سایر قسمت ها ساده می باشند و نیازی به توضیح ندارند.

توصیف

میکروکنترلر

توصیف میکروکنترلر atmega8 :

میکروکنترلر به کار رفته در مدار، AVR Atmega8 می باشد.

در این مدار میکرو کنترلر با فرکانس 8MHz و تغذیه 3.3v راه اندازی شده است. در ادامه کمی در مورد خانواده ی میکروکنترلر های AVR و میکروکنترلر Atmega8 صحبت خواهیم کرد.

خانواده ی میکروکنترلرهای AVR:

وقتی که عبارت "میکروکنترلرAVR" را به کار می بریم در واقع درحال اشاره کردن به خانواده ی بزرگی از میکروکنترلرها هستیم. به طور کلی خانواده ی میکروکنترلرهای AVR را می توان بـه پـنج دسـته ی کلـی تقسیم کرد:

۱.میکروکنترلرهای ATtiny AVR

۲.میکروکنترلرهای ATmega AVR

۳.میکروکنترلرهای AT90S AVR

۴.میکروکنترلرهای CAN AVR

۵.میکروکنترلرهای LCD AVR

.میکروکنترلرهای ATtiny AVR :

این دسته از میکروکنترلرهای AVR دارای تعداد پین و حجم و اندازه ی کـم هسـتند و بیشـتر در مـواردی استفاده می شوند که به میکروکنترلرهای کوچک نیاز است, هر یک از میکروکنترلرهای این دسته دارای یک سری قابلیت هستند که در مقایسه با دسته ی ATmega تعداد آنها نسبتا کم است.

۲.میکروکنترلرهای ATmega AVR :

این دسته از میکروکنترلرهای AVR نسبت به دسته ی ATtiny , AT90S امکانات بیشتری دارند به نـوعی قدرتمندترین دسته از خانواده ی AVR هستند. ظرفیت حافظه ی فِلش و فرکـانس کـاری آنهـا نسـبت بـه دسته ی ATtiny فوق العاده بالاتر است و قابلیت خود برنامه ریزی نیز دارند. ایـن دسـته پروتکـل ارتبـاطی JTAG, TWI, SPI, USART را پشتیبانی می کنند.

۳.میکروکنترلررهای AT90S AVR:

این دسته از میکروکنترلرها نسبتا قدیمی هستند و اگر چه قابلیت هایی بالاتر نسبت بـه دسـته ی ATtiny دارند اما کم کم در مدارات, جای خود را به دسته ی ATmega مـی دهنـد بـه طـوری کـه کمپـانی Atmel در Data sheet این دسته از میکروکنترلرها نحوه ی جایگزین کردن تراشه های جدید به جای ایـن دسته راتوضیح داده است.

۴.میکروکنترلرهای CAN AVR :

این دسته از میکروکنترلرها پروتکل CAN(Controlled Area Network) را پشتیبانی مـی کننـد. پروتکـل CAN از پروتکل های مهم و صنعتی است و فاصله های نسبتا طولانی را پشتیبانی کرده و نسـبت بـه نـویز مقاوم است. یکی از مصارف این پروتکل در صنعت خودرو سازی است.

۵.میکروکنترلرهای LCD AVR :

این دسته از میکروکنترلرها برای راه اندازی LCD ها در نظر گرفته شده اند و دارای درایـو LCD و کنتـرل خودکار وضوح تصویر هستند. از ویژگی های ایندسته مصرف توان بسیار پائین بـرای راه انـدازی LCDهـای عددی و همچنین ولتاژ پائین کاری می باشد.

حال به بررسی میکروکنترلر32 ATmega که یک میکروکنترلر PDIP با چهل پین است می پردازیم.

-برای ورود به میکرو کنترلر32 ATmega از Datasheet این میکروکنترلر و از قسمت ویژگی های آن شروع می کنیم.

۱.میکروکنترلر AVR عملکردی بسیار خوب و توان معرفی کمی دارد و در عین حال یک پردازنده ی هشت پینی است به عبارت دیگر میکروکنترلر های AVR هشت پینی هستند یعنی اطلاعات را به صـورت هشـت پینی دریافت می کنند, به صورت هشت پین پردازش می کنند و به صورت هشت بیتی به خروجی تحویـل می دهند. در واقع اطلاعاتی که قرار است در اختیار AVR قرار بگیرد باید هشت بیتی باشد.

۲.میکروکنترلرAVR دارای معماری RISC پیشرفته (بهبود یافته) است. همـانطور کـه قـبلا گفتـیم RISC نوعی معماری CPU است که در مقابل CISC قراردارد.

مجموعه ی دستورالعمل های RISC شامل دستوراتی ساده هستند که غالبا در یک سیکل (هـر یـک پـالس ساعت CPU) اجرا می شوند. این باعث می شود نوشتن برنامه مشـکل تـر شـود امـا سـرعت اجـرای برنامـه افزایش یابد. اما مجموعه ی دستورالعمل های CISC شامل دستوراتی طولانی و پیچیده هستند که غالبـا در بیش از یک سیکل اجرا می شوند و این باعث می شود نوشتن برنامه ساده تـر و سـرعت اجـرای آن کنـدتر شود.

یکی از ویژگی های AVR با معماری RISC این است که دستورالعمل های آن دارای سایز ثابـت ۱۶ یـا ۳۲ بیت می باشد در صورتی که در CPU با معماری CISC دارای دستورالعمل های بـا سـایز متغیـر اسـت و از ویژگی های آن ثابت بودن سایز و سادگی دیکد کردن آن بوسیله ی CPU است.

سومین ویژگی RISC تعداد کم دستورالعمل و سریع بودن آن است تعداد این دستورات با توجه بـه سـری AVR بین ۸۹ تا ۱۳۵ دستورالعمل متغیر است.

چهارمین ویژگی که معماری RISC برای میکروکنترلر AVR ایجاد می کند این است که نوشتن برنامه با زمان اسمبلی مشکل تر می شود و به همین دلیل د غالب موارد از کامپایلرها ی سطح بالا برای برنامه نویسی برای AVR استفاده می شود.

۳.حافظه ی میکروکنترلر ATmega 8 :

دارای ۸ کیلوبایت حافظه ی فِلش داخلی است.

حافظه ی فلش این قابلیت را دارد تا به تعداد ۱۰۰۰۰ مرتبه برنامه ریزی شود یعنی تا ۱۰۰۰۰ مرتبه می توان بر روی آن برنامه نوشت و یا برنامه را از روی آن پاک کرد.

نکته: در میکروکنترلرهای AVR از معماری "هاروارد"در مورد حافظه استفاده شده است به طوریکه حافظه ی میکروکنترلر AVRبه دو قسمت "حافظه ی برنامه" و "حافظه ی داده ها" تقسیم می شود و برای ایجاد ارتباط با هریک از این قسمت ها از گذرگاهها ی مجزا استفاده می شود. حافظه ی برنامه را حافظه ی Flash و قسمتی از حافظه ی داده را حافظه ی داده را حافظه ی SRAM می نامند.

حافظه ی برنامه (Flash) : این حافظه برمبنای تکنولوژی حافظه ی ROM ساخته شده است و جهت ذخیره ی برنامه مورد استفاده قرار می گیرد.

حافظه ی داده: این حافظه به چندین قسمت تقسیم می شود: ۱. ۳۲ رجیستر عمومی ۲. ۶۴ رجیستر I/O ۳. حافظه ی داده داخلی SRAM ۴. حافظه ی داده ی خارجی SRAM . رجیسترهای عمومی مستقیما با ALU در ارتباط هستند. نکته ی قابل ذکر این است که استفاده از حافظه ی SRAM جهت انجام عملیات زمان بیشتری نسبت به حالتی که عملیات فقط روی رجیسترهای عمومی انجام شود نیاز دارد.

علاوه بر حافظه های فوق میکروکنترلر ATmega 8 دارای ۵۱۲ بایت حافظه ی EEPROM است. با قطع برق میکروکنترلر محتویات موجود در حافظه های SRAM و Flash از بین می روند اما اطلاعات موجود در EEPROM میکروکنترلر یک حافظه ی غیر فعال است.

حافظه ی EEPROM میکروکنترلر فقط تا ۱۰۰۰۰ با قابلیت برنامه ریزی شدن و نوشته شدن اطلاعات روی آن را دارد و در نتیجه می توان با توجه به موارد فوق گفت که حجم حافظه ی EEPROM و تعداد دفعاتی که می توان بر روی آن نوشت محدود و کم است به همین خاطر در عمل و در مواردی که لازم است حجم بالایی از اطلاعات به دفعات برروی EEPROM نوشته و پاک شوند از EEPROM های خارجی استفاده می کنند که از راههای مختلف آن ها را به میکرو کنترلرمتصل می کنند.

نکته: ظرفیت حافظه ی SRAM برای میکروکنترلر ATmega8 برابر ۱ کیلو بایت است . نکته ی مهم این است حافظه ی SRAM یکی از محل های ذخیره ی داده بوده و یک حافظه بر مبنای تکنولوژی RAM است. حافظه ی SRAM بطور مستقیم در اختیار CPU نمی باشد و برای دستیابی به SRAM معمولا از یک رجیستر واسط که یکی از رجییسترهای عمومی است استفاده می شود. علاوه بر حافظه ی SRAM داخلی

میکروکنترلر , برخی از میکروکنترلر ها مانند ATmega128 می توانند حافظه ی SRAM خارجی هم داشته باشند , میکروکنترلر ATmega 8 قابلیت را ندارد.

واحد (Master Control Unit) :

واحد MCU واحدی در میکروکنترلراست که مدیریت تمام فعالیت های میکروکنترلر , انجام عملیات لازم بر روی داده ها , ارتباط با حافظه ها و کنترل تجهیزات جانبی را به عهده دارد.

ماژول های جانبی مختلفی نظیر Timer/Counter از طریق گذرگاه داده (Data Bus) اطلاعات خود را برای پردازش در اختیار CPU میکروکنترلر قرار میدهند و CPU هم از طریق همین گذرگاه حاصل های پردازش را به مقصد مورد نظر (که می تواند یک یا چند پین ورودی / خروجی (I/O) و یا یک ماژول ارتباطی مثل SPI باشد) می فرستد.

بخش بعدی معرفی ویژگی های ATmega8 درDatasheet مربوط به ماژول هایی است که به صورت مستقیم با CPU میکروکنترلر در ارتباط هستند:

۱.Timer/Counter :

همانطور که از نام این ماژول پیداست , این ماژول , کارزمان سنجی و شمارش را انجام می دهد.

میکروکنترلر ATmega8 دارای دو Timer/counter هشت بیتی است که دارای تقسیم کننده ی های جداگانه و مد مقایسه گر هستند.

هردو عمل زمان سنجی و شمارش برای ما مهم هستند چرا که گاهی در یک پروژه بحث زمان مطرح است و گاهی بحث شمارش مطرح است .

میکروکنترلر ATmega8 دارای مُد RTC (Real Time counter) است که به کمک آن می توان به صورت دقیق مدت زمان "یک ثانیه" را تولید کرده.

توجه: در مورد ماژول Timer/counter و نحوه ی کار با آن در مباحث بعد به صورت مفصل صحبت خواهیم کرد.

۲.میکروکنترلر ATmega8 دارای چهار کانال تولید مُد PWM (pulse Width Modulation)

۳.میکروکنترلر ATmega8 هشت کانال ADC (Analog to Digital Converter) ده بیتی دارد که امواج آنالوگ ورودی را به مقادیر دیجیتال تبدیل می کند.

۴.میکروکنترلرATmega8 که به صورت بسته بندی TQFP باشد دارای هشت کانال دیفرانسیلی (تفاضلی) است . این ویژگی در بسته بندی PDIP وجود ندارد و در نتیجه ما آن را مورد بررسی قرار نمی دهیم.

۵.بسته بندی TQFP علاوه بر هشت کانال دیفرانسیلی مذکور در بالا دارای دو کانال دیفرانسیلی با قابلیت برنامه ریزی نیز است.

ارتباط های سریال:

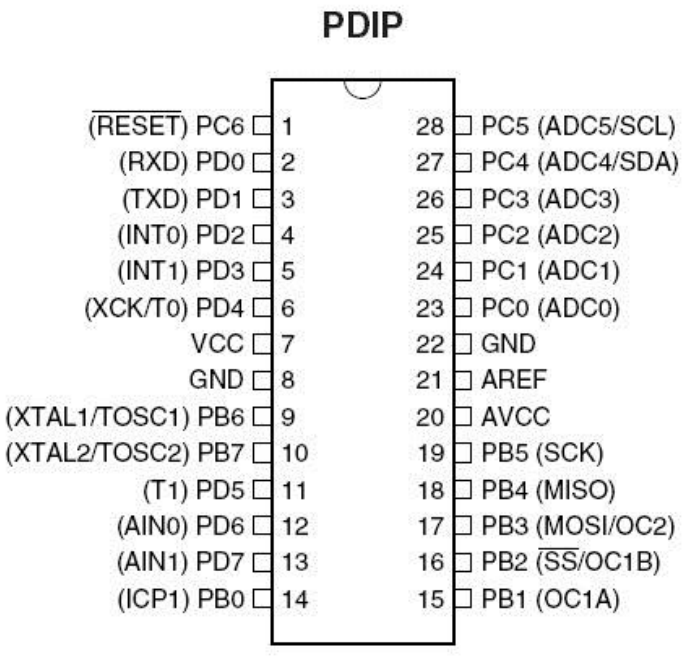
۶.پروتکل سریال دو سیمی (TWI) : این پروتکل با استفاده از دو سیم ویک ارتباط دو طرفه از میکروکنترلربرای یک المان دیگر داده می فرستد و یا از آن المان برای میکروکنترلر داده می گیرد. میکروکنترلر با استفاده از این قابلیت می تواند المان هایی نظیر EEPROM های خارجی و یا برخی از سنسورها را کنترل کند.

۷.پروتکل سریال USART : این پروتکل سریال, یک پروتکل سریال قابل برنامه ریزی است که استاندارد های RS 232 و RS 485 را پشتیبانی می کند . با استفاده از این دو قابلیت و این دو استاندارد می توان یک ارتباط سریال برقرار کرد. دلیل ایجاد استاندارد RS232 در میکروکنترلر ایجاد توانایی برقراری ارتباط با کامپیوتر است چرا که پورت سریال کامپیوتر هم استاندارد RS232 را پشتیبانی می کند.

۸.پروتکل سریال SPI : با استفاده از این پروتکل میکروکنترلر می تواند با چندین المان یا میکروکنترلر ارتباط برقرار کند و در این حالت Mater یا Slave باشد.

۹.میکروکنترلرهای AVR دارای تایمر نگهبان (Watch Dog) قابل برنامه ریزی هستند. به طور خلاصه وظیفه ی تایمر نگهبان این است که در صورتی که میکروکنترلر در طول اجرای برنامه هنگ کرد و اجرای برنامه برای مدت زمانی خاص متوقف شد, میکروکنترلر را Reset کرده و اجازه ندهد که میکروکنترلر برای مدت زمان طولانی در حالت هنگ بماند وسیستم از کار بیفتد.

۱۰.میکروکنترلر ATmega8 دارای مقایسه کننده ی آنالوگ بر روی چیپ است. مقایسه کننده ی آنالوگ (OPAMP) دو سیگنال (که ممکن است تفاوت اندکی باهم داشته باشند) را باهم مقایسه کرده و نتیجه ای را در اختیار ما قرار می دهد. از این مقایسه کننده می توان مصارف تقزنت کنندگی کرد که در این حالت بیشترین مقداربهره ۲۰۰ خواهد بود.



نرم افزار موجودر میکرو کنترلر

نرم افزار موجودر میکرو کنترلر :

در این قسمت قصد داریم برنامه ی قرار گرفته در میکروکنترلر را مورد بررسی قرار دهیم. با توجه به توصیفاتی که از قسمت های مختلف مدار ارائه شد، میکرو کنترلر در این مدار پنج وظیفه دارد :

۱- خواندن مقدار ADC و تبدیل آن به ولتاژ

۲- نمایش مقدار بدست آمده بر روی سون سگمنت با استفاده از روش جاروب کردن با وقفه ی تایمر

این موارد با استفاده از پین های I/O و ماژول ADC و با استفاده از کامپایلر CodeVisionAVR قابل اجرا می باشد. قبل از اینکه کد برنامه را ارائه دهیم، در مورد زبان برنامه نویسی و کامپایلر مورد استفاده توضیح می دهیم.

نرم افزار میکروکنترلر در این پروژه با استفاده از زبان C و در کامپایلر CodeVisionAVR نوشته شده است. زبان C یک زبان قدرتمند و جامع برای برنامه نویسی انواع میکروکنترلر ها می باشد و در اکثر پروژه ها از آن استفاده می شود. همانطورکه می دانیم، برای نوشتن برنامه نیازمند یک کامپایلر و یک زبان برنامه نویسی هستیم که آن کامپایلر آن را پشتیبانی می کند. کامپایلر CodeVisionAVR دارای ویژگی های مناسبی از قبیل توابع آماده ی زیاد و code wizard است که کاربرنامه نویسی را تا حدی آسان می کند. این کامپایلر، libraryهای آماده جهت تنظیمات رجیسترهای مختلف بسیاری از میکروکنترلرهای خانواده ی AVR از جمله AT mega8 را دارد که در آنها آدرس ها تبدیل به نامهای رجیستری شده است که این عمل کاربرنامه نویسی را ساده می کند. در کنار این ویژگی code wizard باعث می شود تا بتوان بدون نیاز به حفظ بودن حالات رجیسترها برای تنظیمات مورد نظر و با استفاده از برنامه نویسی گرافیکی رجیسترها را مقدار دهی کرد.

کد برنامه ی درون میکرو کنترلر به صورت زیر می باشد :

```
/*****************************************************
Chip type               : ATmega8L
Program type            : Application
AVR Core Clock frequency: 8.000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/
#include <mega8.h>
#include <delay.h>

#define A  PORTD.7
#define B  PORTD.6
#define C  PORTD.5
#define D  PORTD.4
#define E  PORTD.3
#define F  PORTD.2
#define G  PORTD.1
#define DP PORTD.0
#define COM1 PORTB.3
#define COM2 PORTB.2
#define COM3 PORTB.1
#define COM4 PORTB.0


unsigned char temp1,temp2,temp3,temp4,comcounter=1;
unsigned int SevSegDigit=23456,SSDtemp,voltage;
float vbuffer;
//-----------------------------------------------------------------
--
void Digit (unsigned char dig);

//-----------------------------------------------------------------
--
#define ADC_VREF_TYPE 0x40

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}


// Timer2 output compare interrupt service routine
interrupt [TIM2_COMP] void timer2_comp_isr(void)
{
```

```
//Seven Segment-----------------------------------------------------

SSDtemp=SevSegDigit/10;
temp4=SSDtemp%10;
temp3=SSDtemp/10;
temp3=temp3%10;
temp2=SSDtemp/100;
temp2=temp2%10;
temp1=SSDtemp/1000;
temp1=temp1%10;


if(comcounter==1){Digit(temp1);DP=1;COM1=0;COM2=1;COM3=1;COM4=1;}
if(comcounter==2){Digit(temp2);DP=0;COM1=1;COM2=0;COM3=1;COM4=1;}
if(comcounter==3){Digit(temp3);DP=1;COM1=1;COM2=1;COM3=0;COM4=1;}
if(comcounter==4){Digit(temp4);DP=1;COM1=1;COM2=1;COM3=1;COM4=0;}


comcounter++;
if(comcounter==5){comcounter=1;}
}




void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=Out Func1=Out
Func0=Out
// State7=T State6=T State5=T State4=T State3=1 State2=1 State1=1
State0=1
PORTB=0x0F;
DDRB=0x0F;

// Port C initialization
// Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out
Func1=Out Func0=Out
// State7=1 State6=1 State5=1 State4=1 State3=1 State2=1 State1=1
State0=1
PORTD=0xFF;
DDRD=0xFF;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
```

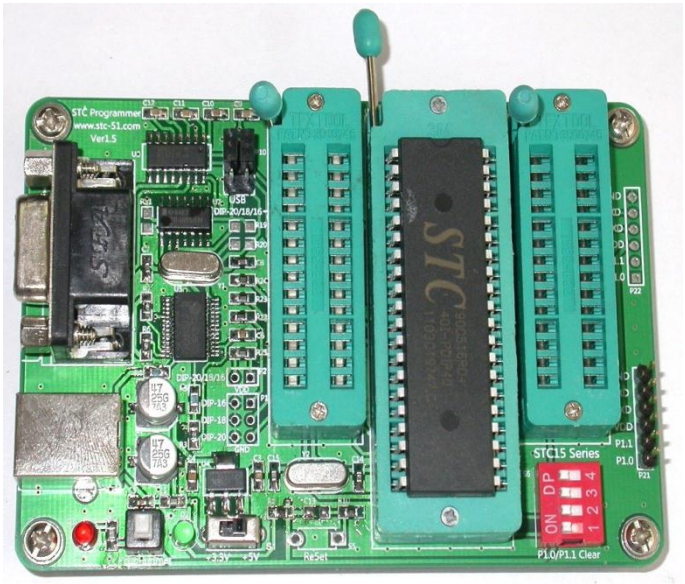
```
TCNT0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
// Mode: CTC top=OCR2
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x0C;
TCNT2=0x00;
OCR2=0xFA;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x80;

// USART initialization
// USART disabled
UCSRB=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: AVCC pin
```

```
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x86;

// SPI initialization
// SPI disabled
SPCR=0x00;

// TWI initialization
// TWI disabled
TWCR=0x00;

// Global enable interrupts
#asm("sei")

SevSegDigit=12560;
while (1)
      {
      voltage=read_adc(5);
      vbuffer=voltage*3.222656;
      vbuffer=vbuffer*11.2;
      voltage=vbuffer;
      SevSegDigit=voltage;
      delay_ms(20);


      }
}
//-----------------------------------------------------------------
---

//-------------------------------------------------------
void Digit (unsigned char dig)
{
if(dig==0){A=0;B=0;C=0;D=0;E=0;F=0;G=1;}
if(dig==1){A=1;B=0;C=0;D=1;E=1;F=1;G=1;}
if(dig==2){A=0;B=0;C=1;D=0;E=0;F=1;G=0;}
if(dig==3){A=0;B=0;C=0;D=0;E=1;F=1;G=0;}
if(dig==4){A=1;B=0;C=0;D=1;E=1;F=0;G=0;}
if(dig==5){A=0;B=1;C=0;D=0;E=1;F=0;G=0;}
if(dig==6){A=0;B=1;C=0;D=0;E=0;F=0;G=0;}
if(dig==7){A=0;B=0;C=0;D=1;E=1;F=1;G=1;}
if(dig==8){A=0;B=0;C=0;D=0;E=0;F=0;G=0;}
if(dig==9){A=0;B=0;C=0;D=0;E=1;F=0;G=0;}

}
//-------------------------------------------------------
```

همانطور که دیده می شود، در ابتدا کتابخانه هایی که از توابع موجود در آنها استفاده شده است معرفی شده اند.

پس از معرفی کتابخانه ها، ماکرو ها و متغیر های Global موجود در برنامه معرفی شده اند.سپس تابع محاسبه ی کد دیجیتال توسط ماژول ADC داخلی آورده شده است. در بدنه ی Main ابتدا پیکربندی های مورد نیاز برای راه اندازی ماژول های داخلی آورده شده است.

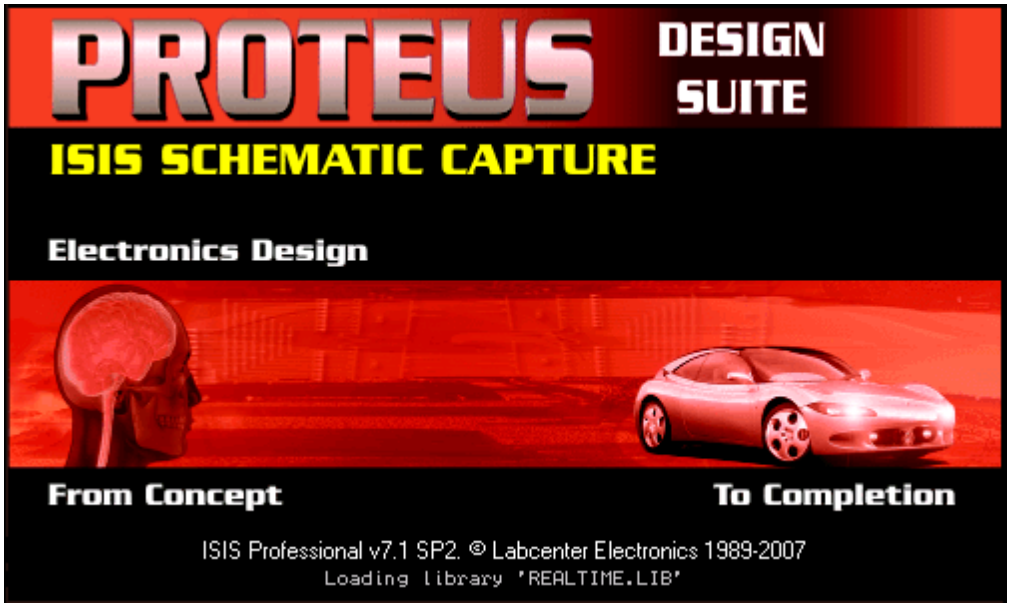
پس از کامپایل کردن بی خطای پروژه فایل hex. ساخته شده توسط کامپایلر را با استفاده از پروگرمر به میکروکنترلر منتقل می نماییم.



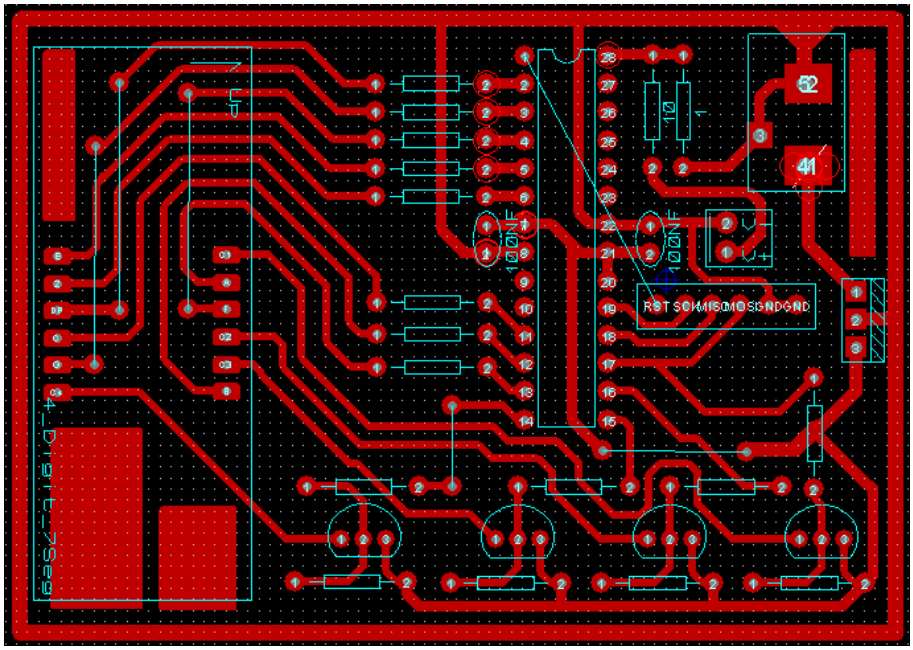
ساخت برد مدار چاپی

ساخت برد مدار چاپی :

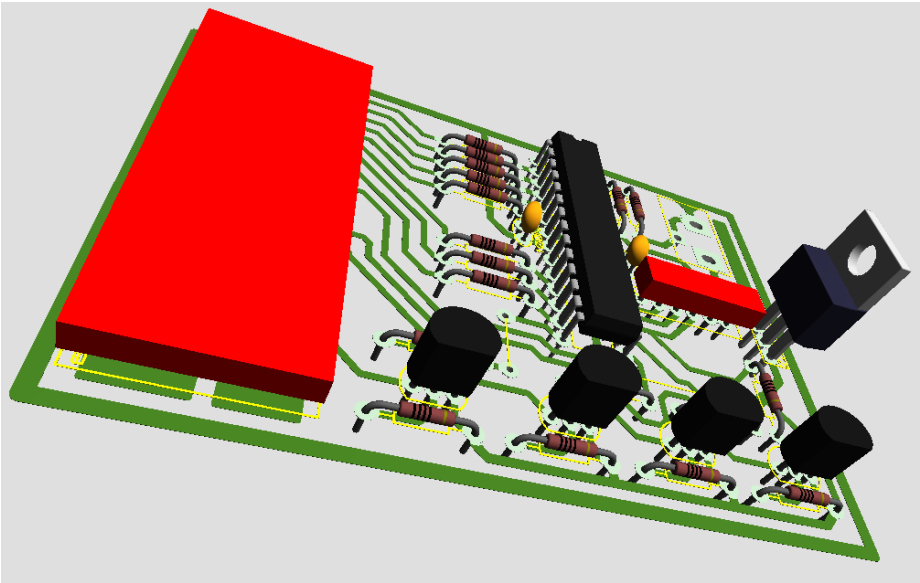
پس از اطمینان از عملکرد صحیح مدار و مشخص شدن همه ی المان های موجود در آن، نوبت به پیاده سازی مدار بر روی برد مدار چاپی می رسد. برای این کار ابتدا بر اساس شماتیک ارائه شده برای مدار و با استفاده از نرم افزار ARES از نرم افزار های موجود در مجموعه ی Proteus نقشه ی PCB مدار را درون این نرم افزار رسم می نماییم.



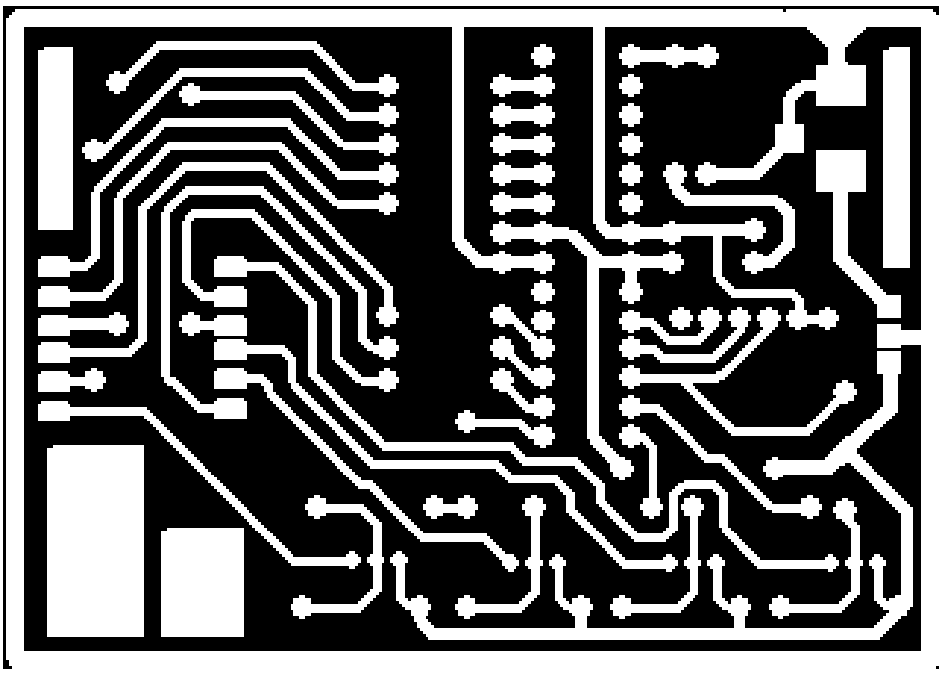
شکل PCB رسم شده برای مدار این پروژه به صورت زیر می باشد :



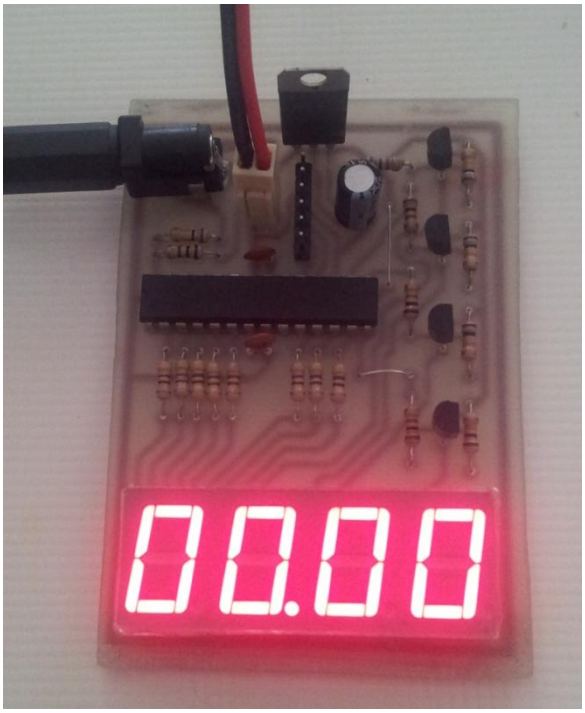
برای اطمینان از صحیح بودن چیدمان قطعات قبل از چاپ کردن برد، شکل سه بعدی شبیه سازی شده در نرم افزار را مشاهده می کنیم. برای این مدار، این شکل به صورت زیر حاصل شد که نشان دهنده ی انتخاب ابعاد صحیح برای مدار می باشد :



پس از حصول اطمینان، Negative نقشه ی PCB چاپ شد و با استفاده از تکنیک های چاپ PCB با استفاده از لامینت، مدار بر روی فیبر یک رو چاپ گردید. شکل Negative مدار به صورت زیر می باشد:



پس از چاپ برد، قطعات مدار بر روی آن مونتاژ شدند و شکل نهایی مدار به صورت شکل زیر شد. سپس تغذیه ی مدار متصل شد و از صحت عملکرد آن اطمینان حاصل شد.



فهرست منابع

فهرست منابع :

۱- WikiPedia /Analog To Digital Converter

۲- Atmega8 MCU Datasheet

۳- CodevisionAVR Compiler Help

۴- AvrFreaks Web/ADC Application note

پایان