پایان نامه دوره کارشناسی مهندسی الکترونیک

موضوع :

طراحی و ساخت دستگاه های اسیلوسکوپ و سیگنال ژنراتور

دانشجویان :

فرشاد طاهونی

حمید کاظم نژاد

پائیز ۹۱

پیشگفتار

در این پروژه به طراحی و ساخت دستگاه های سیگنال ژنراتور و اسیلوسکوپ که قابلیت اندازه گیری و نمایش تمامی شکل موج ها را دارد پرداخته شده است.در هر دوی این دستگاه ها از میکروکنترلرهای خانواده AVR به عنوان پردازشگر اصلی استفاده شده است.

این پروژه در چهار فصل تنظیم شده که در فصل اول به توضیح مسائل کلی و بلاک ودیاگرام پروژه , امکانات و همچنین خلاصه ای از آنچه که در فصل های بعد تشریح می شود پرداخته شده است.در فصل دوم سخت افزارهای استفاده شده در این دو پروژه و مشخصات فنی هر سخت افزار و چگونگی کارکرد هر کدام و تشریح نقشه فنی سیستم مورد بررسی قرار گرفته است.در فصل سوم نیز برنامه نوشته شده در قسمت میکروکنترلر سیستم که مرکز فرماندهی سیستم را تشکیل می دهد مورد بررسی قرار گرفته.در فصل آخر نیز به طور مختصر پروژه را تشریح کرده و نتایج و برخی کاربرد های آن را ارائه کرده ایم.

دستگاه اسیلوسکوپ ساخته شده قابلیت نشان دادن تمامی شکل موج ها از قبیل شکل موج سینوسی , مربعی , مثلثی و همچنین دندانه اره ای را دارد.علاوه بر نشان دادن این شکل موج ها این دستگاه قابلیت اندازه گیری دامنه و همچنین فرکانس این سیگنال ها را دارد.تمامی این قابلیت ها بر روی صفحه LCD گرافیکی قابل مشاهده می باشند.

دستگاه سیگنال ژنراتور ساخته شده نیز توانایی تولید سیگنال سینوسی با قابلیت تغییر دامنه , فرکانس و فاز را دارد.

برای ساخت و به اتمام رساندن این دو پروژه با سختی ها و مشکلات فراوانی روبرو بودیم.اما با وجود همه این سختی ها خدا را شاکریم که توانستیم به طرز مطلوبی این دو پروژه را به اتمام رسانیم.تلاش و پشتکار هر چند بسیار زیاد ما در به ثمر رسانیدن این دو پروژه بدون راهنمایی ها و کمک های تئوری استادان گرانقدری که از شروع این دوره ما را راهنمایی کردند میسر نبود.

در ادامه این پایان نامه ما به دنبال گفتن مطالبی هستیم که شما را در امر ساخت و طراحی و آموزش های مربوطه این دومدار یاری می کند.

امیدواریم که توانسته باشیم شما را در امر آموزش گوشه ای از این علم پر رمز و راز الکترونیک که شما وارد آن شده اید یاری کرده باشیم

در ضمن این پایان نامه را به تمام کسانی که

ما را در راستای کسب علم و دانش یاری کردند تقدیم می کنیم

و از آنها تقدیر و تشکر کرده و از خدای منان برای آنها آرزوی موفقیت و سربلندی داریم.

با تشکر: فرشاد طاهونی – حمید کاظم نژاد

پائیز ۹۱

فصل اول

عنوان پروژه و تحقیق

فصل دوم

تشریح نقشه فنی پروژه و سخت افزار مربوط

فصل سوم

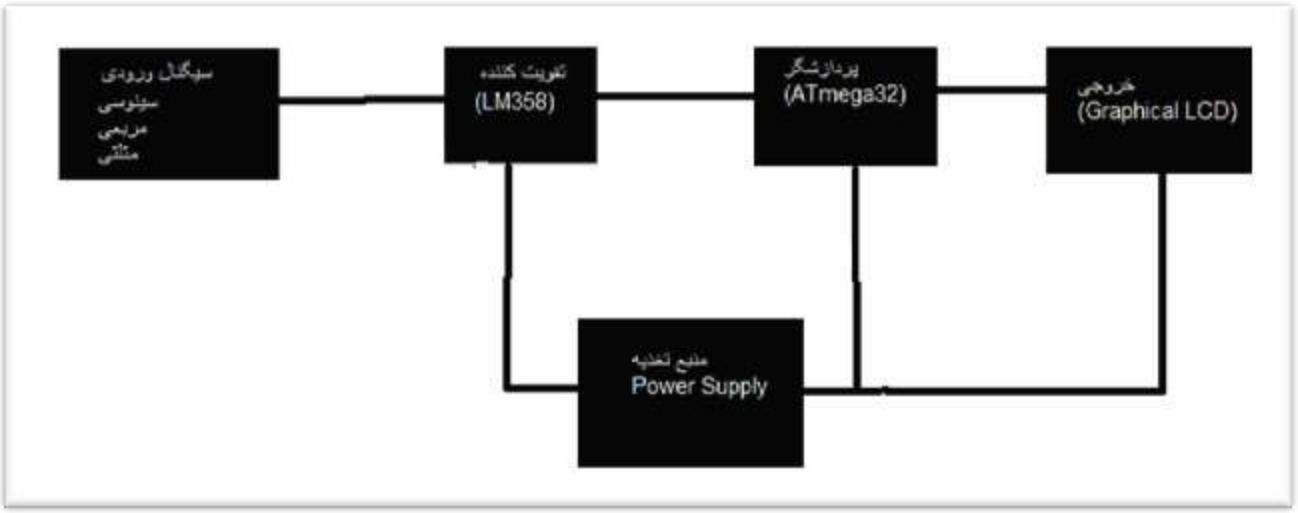
تشریح نرم افزار و برنامه های مربوط به آن

فصل چهارم

خلاصه پروژه

فصل اول

بلوک دیاگرام مدار اسیلوسکوپ :



شکل ۱-۱ بلوک دیاگرام مدار

اجزای مدار اسیلوسکوپ

مدار اسیلوسکوپ ذکر شده از قطعات اصلی زیر تشکیل شده است:

LCD گرافیکی : که برای نمایش داده ها و خروجی ها در مدار به کار رفته است.

آی سی Atmega32 : که برای تجزیه و تحلیل مقادیر و ارسال آن به خروجی به کار رفته است.

آی سی LM358: که برای دریافت مقادیر و تقویت آن در مدار و اعمال آن به آی سی میکروکنترلر به کار رفته است.

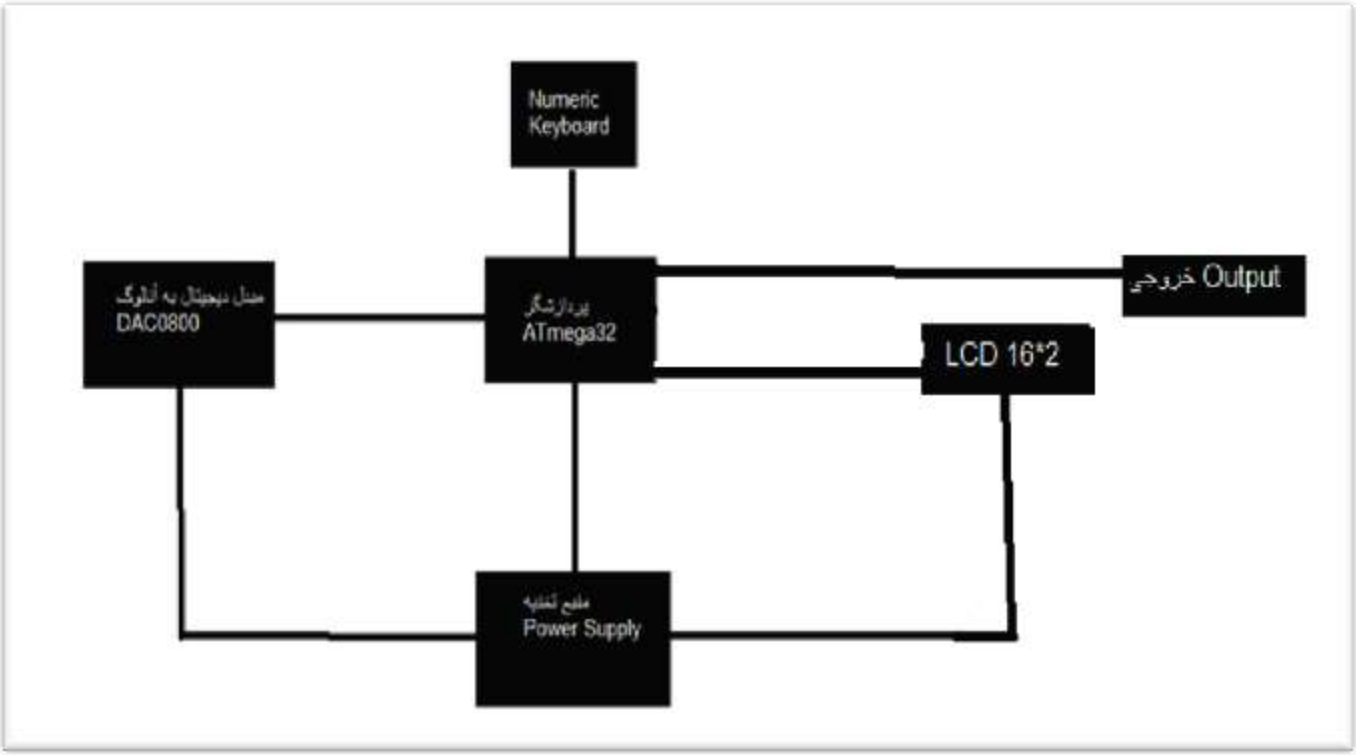
کلید ها و مقاومت متغیرها : که برای تغییر مقادیر و کالیبره کردن مدار و تغییر روشنایی صفحه LCD به کار گرفته شده است .

آی سی رگولاتور: برای تامین ولتاژ تغذیه مدار استفاده شده است .

کریستال 16 MHz: در قسمت اسیلاتور مدار و تامین کلاک پالس مورد نیاز مدار به کار می رود .

خازن های MKT : به دلیل دقت بالا و ثابت بودن مقدار آن ها و نویز پذیری کمتر و بسیاری از موارد دیگر به کار گرفته شده اند.

بلوک دیاگرام مدارسیگنال ژنراتور:



شکل ۲-۱ بلوک دیاگرام مدار

اجزای مدار سیگنال ژنراتور

مدار سیگنال ژنراتور ذکر شده از قطعات اصلی زیر تشکیل شده است :

آی سی Atmega32 : که برای تجزیه و تحلیل مقادیر و ارسال آن به خروجی به کار رفته است .

LCD 16 * 2 : برای نمایش مقادیر از جمله فرکانس و دامنه به کار رفته است .

Keyboard : برای تنظیم مقدار فرکانس و دامنه به کار رفته است .

آی سی DAC0800 : این آی سی مقادیر باینری را با توجه به ولتاژی که به ورودی آن داده می شود می سازد .

فرایند ساخت دستگاه اسیلوسکوپ

پس از تلاش های بسیار در ساخت این مدار به مشکلات سخت افزاری و نرم افزاری زیادی بر خوردیم . از جمله این مشکل ها می توان به نشان ندادن بعضی از شکل موج ها و دقیق نبودن مقادیر اندازه گیری و غیره می توان اشاره کرد . پس از آزمایشات بسیار سر انجام بعد از جواب گرفتن کامل از مدار مربوطه شروع به طراحی PCB مربوط به مدار کردیم . پس از طراحی آن و آماده کردن بورد PCB شروع به بستن قطعات مدار کردیم .

از نکاتی جالب که می توان به آنها اشاره کرد استفاده از مقاومت های ۱ درصد خطا و خازن های MKT و قطعات با درصد خطای کم می باشد .زیرا این قطعات درصد خطای سیستم را پائین آورده و در عین حال سرعت سیستم را بالا برده و در نهایت موجب عملکرد صحیح مدار می شود .

فرایند ساخت دستگاه سیگنال ژنراتور

در ساخت مدار سیگنال ژنراتور با مشکلات بسیار و پیچیده ای روبرو شدیم . از جمله این مشکلات مشکل در راه اندازی آی سی DAC بود . چون با این آی سی تا به حال کار نکرده بودیم بنابر این تجربه نو و تازه ای برای ما به شمار می رفت در آخر بعد از تحقیقات و بررسی های زیاد در مورد این قطعه و نحوه راه اندازی آن تقریبا از مدار مورد نظر جواب گرفتیم . مشکل دیگری که پیش آمده بود مشکل در مقدار دامنه ولتاژ این مدار بود . که توانستیم با انتخاب ولتاژ مرجع مناسب این مشکل را حل کنیم . در آخر بعد از رفت و آمد های زیاد به اتاق پروژه و تست های تکمیلی مدار شروع به ساخت این مدار روی بورد الکترنیکی کردیم.

کارهای صورت گرفته پایانی در انجام پروژه

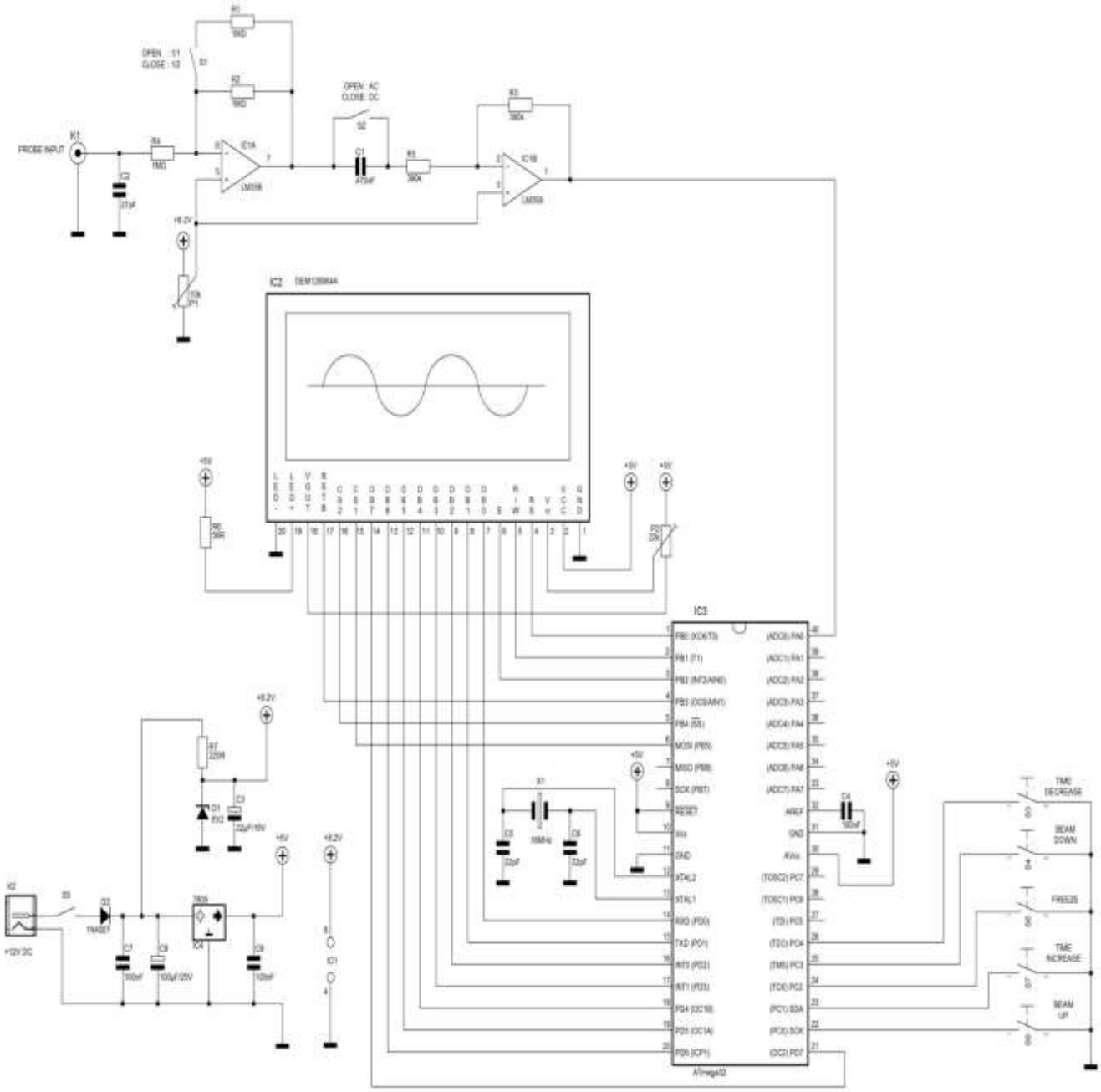
برای تمامی LCDها و keypad و دکمه های موجود در این دو مدار از سوکت و فلت برای نصب استفاده کردیم تا هم مدار با کیفیت و قشنگی ساخته شود و هم اینکه در جداسازی و نصب دوباره آنها مشکلی پیش نیاید . در آخر پس از تهیه جعبه

تمامی بوردها را داخل جعبه پیچ نمودیم و همانطور که گفته شد تمامی LCD و چیز های دیگر را توسط فلت به خارج جعبه متصل کردیم .

فصل دوم

شماتیک و نقشه فنی مدار اسیلوسکوپ

در این قسمت به توضیح مدار عملی دستگاه اسیلوسکوپ و اجزای مختلف آن به طور کامل می پردازیم.



شکل ۱-۲ نقشه فنی مدار اسیلوسکوپ

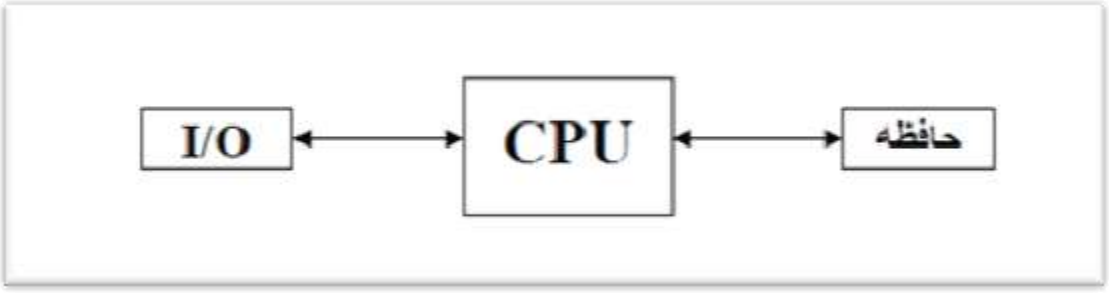
در ابتدا به توضیح قطعاتی که در مدار نقش اصلی داشتند می پردازیم و سپس در انتها قطعات دیگری که ما در ساخت این مدار از آنها استفاده کرده ایم را توضیح می دهیم.

آی سی **ATmega32**

این میکروکنترلر از نوع میکروکنترلهای خانواده AVR می باشد.ابتدا باید بدانیم که میکرو کنترلر چیست و چه کاربردهایی دارد.

میکروکنترلرهای AVR

برای درک بهتر مفهوم میکروکنترلر بهتر است ابتدا روندی که منجر به تولید این قطعات گردید بررسی گردد. کامپیوترها امروزه نقشی غیر قابل انکار در زندگی بشر دارند .گسترش کاربرد کامپیوتر در علوم مختلف در چند دهه اخیر به حدی زیاد بوده که امروزه تصور زندگی بدون آن بسیار سخت است. کامپیوترها با تمام پیچیدگیهایشان، همگی بلوک دیاگرام ساده ای مانند شکل زیر دارند.



شکل ۲-۲ بلوک دیاگرام میکروکنترلر

سه بلوک اصلی نشان داده شده در این شکل، واحد مرکزی پردازش CPU حافظه و ادوات ورودی-خروجی I/O می باشد . واحد مرکزی پردازش در واقع واحد اصلی پردازشگر در هر سیستم کامپیوتری می باشد که از واحدهای متعددی مانند ALU واحد کنترل ، رجیستر و دکودر دستورالعمل و مدارات تولید پالس داخلی تشکیل شده است . در آغاز روند تولید کامپیوترها واحدهای مختلف CPU به صورت ماژولهای جداگانه طراحی می شد و از کنار هم قرار گرفتن آنها و ارتباط واحدها با یکدیگر CPU تشکیل می گردید .

حافظه های ROM و حافظه های RAM که هریک در سیستم میکروپروسسوری وظیفه خاصی برعهده دارند. حافظه ROM همانطورکه از نامش پیداست حافظه ای تنها خواندنیاست . علت استفاده از این نوع حافظه در سیستم ها ، غیر فرار بودن اطلاعات آن است . به این ترتیب داده ای که در یک چنین حافظه ای قرار گرفته باشد، حتی پس از قطع تغذیه سیستم نیز دست نخورده باقی می ماند. بنابراین اینگونه حافظه ها کاندیدای مناسبی برای ذخیره سازی اطلاعات دائمی در سیستم هستند .در یک سیستم میکروپروسسوری عملکرد سیستم توسط برنامه ای که در حافظه ذخیره شده و به ترتیب از خانه های متوالی

حافظه توسط CPU خوانده و اجرا می شود ، تعیین می گردد . تمامی یا بخشی از این برنامه در هر سیستم میکروپروسسوری در داخل ROM قرار می گیرد .

در هر سیستم میکروپروسسوری در ضمن اجرای برنامه بنا به دلایل مختلف نیاز به ذخیره سازی اطلاعات بصورت داده ها و یا حتی برنامه های موقت می باشد، لذا استفاده از حافظه قابل نوشتن در سیستمه میکروپروسسوری ضروری است. برای این منظور از حافظه های RAM استفاده می گردد. برای ساخت حافظه ها ROMو RAM از تکنولوژی های مختلف با مزایا و معایب خاص خود استفاده می گردد که از جمله آنها می توان به EEPROM، PROM،EPROM برای ساخت ROM و تکنولوژی استاتیکی و دینامیکی برای ساخت RAM اشاره کرد.

هر سیستم میکروپروسسوری محلی برای ذخیره دائمی اطلاعات نرم افزار سیستم احتیاج دارد . این واحد حافظه علاوه بر اینکه باید قابلیت ذخیره اطلاعات را داشته باشد ، باید قابلیت بازنویسی و تغییر اطلاعات و نرم افزار سیستم را نیز داشته باشد . برای این منظور از حافظه جانبی استفاده می گردد . در سیستمهای میکروپروسسوری قدیم یتر برای ساخت حافظه های جانبی از ادوات نیمه هادی استفاده نمی شد و به جای آنها از وسایل دارای قسمتهای مکانیکی استفاده می گردید که سبب بروز مشکلاتی مانند سرعت پایین ، حجم ذخیره سازی کم و ... می گردید . امروزه با پیشرفت تکنولوژی ساخت ادوات نیمه هادی از تراشه های نیمه هادی برای حافظه جانبی استفاده می گردد . در اینجا اشاره به این نکته نیز لازم است که به علت سرعت بالای میکروپروسسور و سرعت پایین حافظه ها ، ارتباط بین حافظه جانبی و حافظه اصلی با سایر قسمت ها در سیستم میکروپروسسوری از طریق یک مدار واسط انجام می گیرد .

میکروکنترلرهای خانواده **AVR** به **5** دسته اصلی تقسیم بندی می شود **:**

- میکروکنترلرهای ATiny AVR
- میکروکنترلرهای AT90s AVR
- میکروکنترلرهای CAN AVR
- میکروکنترلرهای LCD AVR
- میکروکنترلرهای ATmega AVR

از آنجا که در این پایان نامه مطالب کامل و تکمیلی در مورد میکروکنترلر ATmega32 ارائه می گردد , در این قسمت تنها به توضیح مختصری درباره قابلیتهای میکروکنترلرهای ATmega AVR بسنده می گردد .

میکروکنترلرهای **ATmega AVR**

این دسته از میکروکنترلرها پرکاربردترین خانواده AVR هستند که امکانات وسیعتر و دستورالعمل های قویتری را پشتیبانی می کنند.ظرفیت حافظه FLASH و فرکانس کاری آنها نسبت به دسته پرکاربرد دیگر یعنی سری ATiny AVR به طور قابل توجهی افزایش یافته است .وجود تایمرها و کانترها، قابلیت کنترل PWM , توانایی برقراری ارتباط سریال و موازی، وجود مدارات تقسیم کلاک داخلی و تنوع فراوان در بست هبندی های متفاوت از بزرگترین مزایای این خانواده AVR هستند . از پرکاربردترین میکروکنترلرهای این خانواده می توان به Atmega8و ATmega16 ،Atmega32 اشاره کرد .

مشخصات آی سی **ATmega32**

* توان مصرفی پایین آنها برای استفاده بهینه از باتری و همچنین کاربرد میکرو در وسایل سیار و سفری طراحی شده که میکروهای جدید AVR با توان مصرفی کم از شش روش اضافی در مقدار توان مصرفی ، برای انجام عملیات بهره می برند.

* این میکروها تا مقدار 1.8 ولت قابل تغذیه هستند که این امر باعث طولانی تر شدن عمر باتری می شود.

*در میکروهای با توان پایین ، عملیات شبیه حالت Standby است یعنی میکرو می تواند تمام اعمال داخلی و جنبی را متوقف کند و کریستال خارجی را به همان وضعیت شش کلاک در هر چرخه رها کند!

*قابلیت دوباره برنامه ریزی کردن بدون احتیاج به اجزای خارجی

* 128 بایت کوچک که به صورت فلش سکتور بندی شده اند

*داشتن مقدار متغیر در سایز بلوکه ی بوت (Boot Block)

*خواندن به هنگام نوشتن

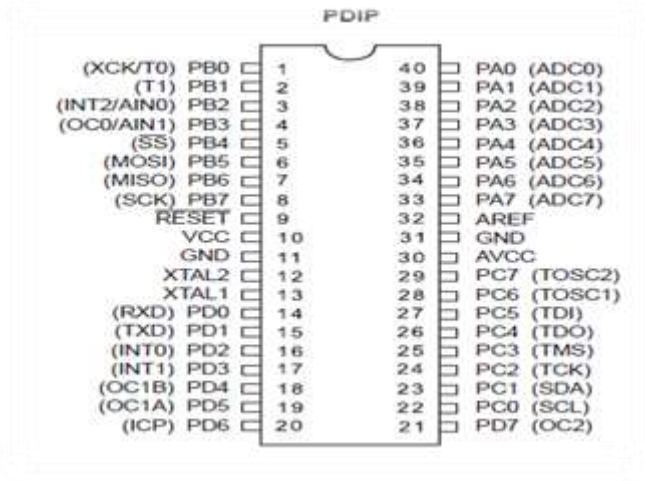
*بسیار آسان برای استفاده

*کاهش یافتن زمان برنامه ریزی

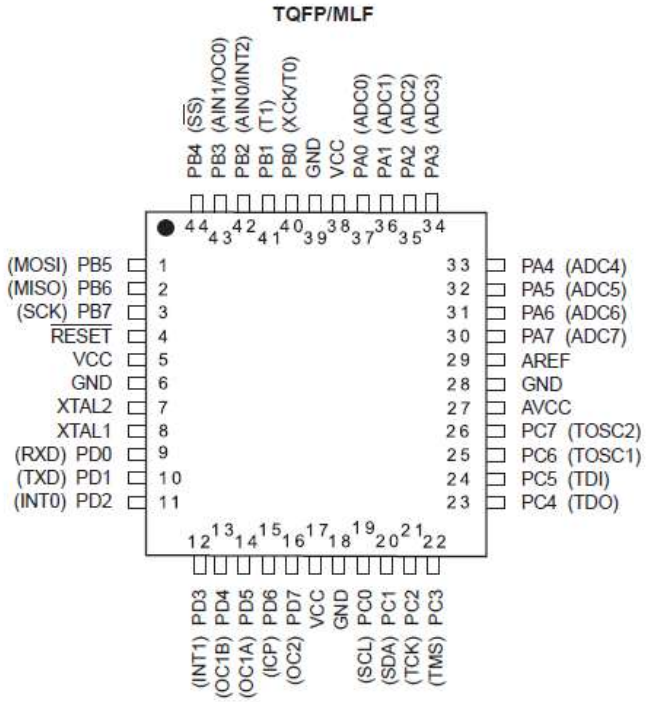
*کنترل کردن برنامه ریزی به صورت سخت افزاری

شکل ظاهری و پایه ها:

ATMEGA32 در سه نوع بسته بندیPDIP با 40 پایه و TQFP با 44 پایه و MLF با 44 پایه ساخته میشود که در بازار ایران بیشتر نوع PDIP موجود می باشد.



شکل ۳-۲ پایه های Atmega32



شکل ۴-۲ مدل TQFP آی سی Atmega32

ATMRGA32 دارای چهار پورت8 بیتی (1بایتی) می باشد که علاوه بر اینکه بعنوان یک پورت معمولی میتوانند باشند کارهای دیگری نیز انجام میدهند . بطور مثال PORTA می تواند بعنوان ورودی ADC (تبدیل ولتاژ آنالوگ به کد دیجیتال) استفاده شود که این خاصیت های مختلف پورت در برنامه ای که نوشته میشود تعیین خواهد شد. ولتاژ مصرفی این آی سی از 4.5ولت تا 5.5ولت می تواند باشد .

فرکانس کار هم تا 16 مگاهرتز میتواند انتخاب شود که تا 8 مگاهرتز نیازی به کریستال خارجی نیست و در داخل خود آی سی می تواند تامین شود . فرکانس کار از جمله مواردی است که باید در برنامه تعیین شود . لازم به ذکر است که این فرکانس بدون هیچ تقسیمی به CPU داده میشود . بنابراین این خانواده از میکروکنترلرها سرعت بیشتری نسبت خانواده های دیگر دارند . پایه ی شماره 9 نیز ریست سخت افزاری میباشد و برای عملکرد عادی آی سی نباید به جایی وصل شود و برای ریست کردن نیز باید به زمین وصل شود.

پایه های 12و 13 نیز برای استفاده از کریستال خارجی تعبیه شده است.

ساختار داخلی **Atmega32**

برنامه ای که برای میکروکنترلر در کامپیوتر نوشته میشود وقتی که برای استفاده در آی سی ریخته میشود (توسط پروگرامر مخصوص آن خانواده) در مکانی از آن آی سی ذخیره خواهد شد بنام ROM. در این آی سی مکانی برای ذخیره موقت اطلاعات یا همان RAM هم وجود دارد که مقدارش 2کیلو بایت است. در RAM اطلاعات فقط تا زمانی که انرژی الکتریکی موجود باشد خواهد ماند و با قطع باتری اطلاعات از دست خواهند رفت . به همین منظور در Atmega32 مکانی برای ذخیره اطلاعات وجود دارد که با قطع انرژی از دست نخواهند رفت . به این نوع حافظه ها EEPROM گفته میشود که در این آی سی مقدارش 1KB است و تا 100,000 بار میتواند پر و خالی شود .

LCDگرافیکی

مقدمه

LCD ها صفحه هایی از جنس **کریستال مایع** هستند نسبتا ارزان و بسیار پرکاربرد. برای اینکه بتونیم اطلاعاتی رو نمایش بدیم میتونیم از این LCD ها استفاده کنیم حتما تا حالا توی مدارهای الکترونیکی از اینها دیدید :

شکل ۵-۲ شمای LCDگرافیکی

در تصویر بالا دو تا از این LCD ها رو میبینید که کوچیکه رو LCD دو در شانزده میگن یعنی ۲ تا سطر و ۱۶ تا ستون داره اون یکی هم که اندازه اش چند برابر اونه یه GLCD یا همون LCD گرافیکی هست اندازه اون هم ۲۴۰ در ۱۲۸ هست LCD های معمولی (کاراکتری) فقط متن رو نشون میدن اما نوع گرافیکی اونها میتونن عکس رو هم نشان دهند. این ال سی دی ها تک رنگ هستن در دو رنگ سبز و آبی موجود اند در تصویر بالا LCD از نوع سبز رنگ و GLCD از نوع آبی رنگ هست (آبی نسبت به سبز کمی گرونتر هست ولی زیباتر هم هستند) نوع رنگی اونها هم هستن که تو موبایلها میبینید .

اندازه های LCD

LCD ها در چند سایز مختلف وجود دارند 2*20 , 4*16 , 4*20 , 2*16 16*1a(نوع خاصی از LCD هست که به صورت 2*8 استفاده میشه) وجود داره .

یک LCD گرافیکی از تعدادی ماتریس نقطه ای به هم پیوسته تشکیل شده اند و توسط یک تراشه راه انداز کنترل می شوند .

به طور کلی این نوع LCDها بر اساس تراشه راه انداز داخلی خود دسته بندی می شوند که دو نوع از معروف ترین آن ها عبارت اند از :

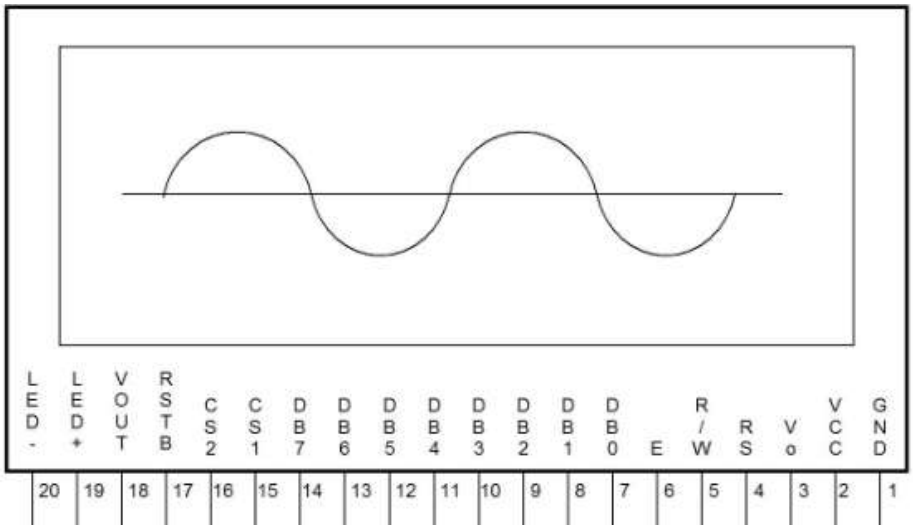
۱- مدل GDM با تراشه راه انداز KS0108

۲- مدل Toshiba با تراشه راه انداز t6963C

در اینجا ما از **LCD** نوع اول استفاده کرده ایم و به توضیح آن می پردازیم :

GDM128*64A پرکاربرد ترین مدل GDM می باشد.این LCD ماتریسی نقطه ای بر اساس تکنولوژی CMOS ساختهشده است و می تواند ماتریس نقطه ای به اندازه ۶۴ سطر و ۱۲۸ ستون (۸۱۹۲ نقطه) را توسط یک RAM نمایش (DDRAM) به ظرفیت ۱۲۸*۶۴ بیت نمایش دهد. این مدل توسط تراشه ks0108 کنترل می شود.این تراشه یک راه انداز با ۶۴ کانال خروجی و به منظور کنترل LCD های ماتریسی گرافیکی طراحی شده است.

شرح پایه ها



شکل ۶-۲ نمای پایه های LCD

پایه GND

پایه زمین

پایه VCC

پایه تغذیه با ولتاژ بین ۴.۵ تا ۵.۵ ولت

پایه VO

پایه ای جهت کنترل روشنایی صفحه که معمولا توسط یک پتانسیومتر و از پایه VEE تغذیه می شود.

پایه RS

اگر RS=0 باشد رجیستر دستورالعمل انتخاب شده و اطلاعات روی پایه های DB0 تا DB7 به عنوان دستور شناخته می شود.

و اگر برابر یک شود رجیستر داده برای نمایش اطلاعات روی LCD انتخاب می شود.

پایه RW

اگر RW=0 باشد اطلاعات از روی پایه های DB0 تا DB7 خوانده می شود و اگر برابر یک شود اطلاعات بر روی پایه ها

قرار خواهد گرفت که توسط میکروکنترلر جانبی خوانده خواهد شد.

پایه E

LCD از این پایه برای قفل کردن اطلاعات روی پایه های DB0 تا DB7 استفاده می کند.هنگامی که اطلاعات روی پایه های

DB0 تا DB1 قرار بگیرد با یک لبه پایین رونده روی این پایه اطلاعات درون تراشه LCD قفل می شود.حداقل پالس اعمالی به

این پایه باید ۴۵۰ نانو ثانیه طول داشته باشد.

پایه های **DB0** تا **DB7**

از این پایه ها برای انتقال اطلاعات به LCD و یا خواندن از آن استفاده می شود.

پایه های **CS1** و **CS2**

همانطور که در بخش قبلی گفته شد این تراشه دارای ۶۴ کانال خروجی است بنابراین برای کنترل یک LCD از نوع ۱۲۸*۶۴ نیاز به دو تراشه است.توسط پایه های CS1 و CS2 می توان هرکدام از این تراشه ها را که یکی مربوط به کنترل نیمه راست و دیگری نیمه چپ صفحه است انتخاب نمود.

اگر CS1=0 وCS2=1 باشد نیمه راست صفحه انتخاب می شود.

اگر CS1=1 وCS2=0 باشد نیمه چپ صفحه انتخاب می شود.

پایه **RST**

با فعال کردن (low) این پایه LCD در وضعیت Reset قرار می گیرد.

پایه **VOUT**

این LCD ها توسط مدار های داخلی خود یک ولتاژ منفی در حدود ۹.۵ ولت روی ای ن پایه ایجاد می کند.به کمک این پایه و VO می توان میزان روشنایی صفحه را توسط یک پتانسیومتر کنترل کرد.

پایه های **LED+** و **LED –**

این دو پایه مربوط به آند و کاتد روشن کننده صفحه LCD است و به ولتاژی در حدود ۵ ولت نیاز دارد.

تراشه **LM358N**

امروزه هر مداری که مشاهده می کنید در بطن ان یک یا چند تا ایسی به شماره LM358 را می بینید این ایسی چیست ؟ چکاره هست که تقریبا تمام مدارات حتی مدارات پویایی مثل PIC ها را در بر گرفته است .

معرفی ای سی: **LM358**

این ایسی یکی از مشهورترین اپ امپ هایی است که با توجه به ورودی های موسفیت ان تقریبا در اکثر مدارات کاربرد دارد و در شکل های فیزیکی ۸ پایه معمولی و SMD و.... تولید شده است میتوان انرا به صورت دو ولتاژه سه سیم _+ با خط صفر و تک ولتاژ دو خطی منفی و مثبت تا ۳۲ ولت و بستگی به نیاز در مدارات با امکانات مهم و بسیاری که دارد بکار بست.

تمام این ایسی ها از یک خانواده و از گروه اپ امپ های موسفیت ۱۲۴ و همگی مشابه هم هستند فرقی که اینها با همدیگر دارند در ولتاژ کار بالاتر از 15ولت و کمتر از ۵ ولت دارند مثلا خود ایسی lm358 تحمل ولتاژ بیش از ۳۲ ولت را ندارد و در کمتر از ۵ ولت پاسخ مناسبی نخواهد داد و این در حالیست که ایسی مشابه ان به شماره 2904حداکثر تا ۲۴ ولت و حداقل واتاژ کاری ان ۳/۳ ولت میباشد فرق دوم در تمپراتور (حرارت تحملی) انها می باشد. این ای سی یک کودر و دکودر فرکانس تا محدوده 500کیلوهرتزمی باشد (نوع Typical ان تا فرکانسهای ۱۰ مگاهرتز) کاربرد دارد .

تشریح کاربرد: **LM358**

قبل از تشریح اجازه دهید اشنایی کاملی از پایه های این ایسی را داشته باشیم تا بموقع در درک چگونگی انجام بکارگیری ان مشکلی نداشته باشیم.

با توجه به وجود دو عدد مقایسه گر مستقل در داخل این ایسی پایه های ان به شرح ذیل میباشد.

۱- خروجی مقایسه گر A

۲ - ورودی اول (منفی) A از مقایسه گر . A

۳- ورودی دوم (مثبت) A از مقایسه گر A

پایه چهارم (۴)خط تغذیه منفی دو ولتاژه یا گراند تک ولتاژه

و پایه های ۵ الی ۸ به ترتیب ۵ ورودی مثبت مقایسه گر. B و ۶ ورودی منفی مقایسه گر. B و پایه هفت (۷) خروجی مقایسه گرB و بالاخره پایه هشتم ۸خط ولتاژ تغذیه مثبت ایسی میباشد .

مثل سایر اپ امپ ها این ایسی نیز کاربرد های مختلف و فراوانی دارد که بستگی تمام به نوع مدار و نیاز طراح ان دارد ولی عمدتا همون مقایسه گر و با ارایشهایی در اشمیت تریگر و اوسیلاتور و نوعی فیلپ فلاپ و صا فی هایی با گذرهای بالا و پایین و میان و ..... می توان نام برد .برای حصول به نتیجه کار کامل توسط این ایسی شرکت های سازنده بسته به نوع کاربرد ان در مدارات فرمولهایی را ارایه کرده اند که هنگام طراحی و بکارگیری این ایسی رعایت انها مهم و ضروری می باشد .

یکی دیگر از محاسن این ایسی در شکل (اندازه) ولتاژهای ورودی و خروجی و انطباق راحت انها با مدارات انالوگ و دیجیتال می باشد که این ایسی را در گروه کاربردی ترین ایسی های بکاررفته در مدارات انالوگ و دیجیتال قرار داده است تمام تابع های اپ امپ ها در این ایسی نیز به راحتی قابل اجرا می باشد .

میتوانید فرکانس کاری ای سی را تعیین کرده و اوسیلاتور داخلی ای سی را در فرکانس محاسبه شده از طریق مقاومت و خازن خارجی در صورت نیاز با کریستال کوارتز (روی پایه های شماره ۶ و ۵ راه اندازی کنید) . این ایسی در داخل خود یک VCOداره که ورودی انرا پایه ۳ تشکیل میده و خروجی همون VCO روی پایه ۸ ای سی قرار دارد پایه هفت ایسی ولتاژ خط منفی و پایه ۴ ان ولتاژ تغذیه خط مثبت ای سی می باشد .

فرض کنید شما یک اوسیلاتوری دارید که روی فرکانس مثلا ۸ کیلو هرتز در نوسان است اگر خروجی این اوسیلاتور به پایه ۳ ورودی این ایسی اعمال شود در صورتیکه اوسیلاتور داخلی این ایسی نیز روی فرکانس ۸ کیلو هرتز تنظیم و در حال نوسان باشد با رسیدن اولین سیکل فازی اوسیلاتور جانبی شما به VCO این ای سی روی فاز ورودی سیگنال ۸ کیلوهرتز اعمال شده قفل شده و از خروجی خود روی پایه ۸ یک ولتاژ دی سی معادل منبع تغذیه را خواهد داد این ولتاژ ثابت و پایدار خواهد ماند تا زمانیکه فرکانس یکی از اوسیلاتور ها از محدوده قفل خارج شود و یا سیگنال ورودی بکلی قطع شود در این صورت خروجی نیز به خط منفی خواهد رفت .

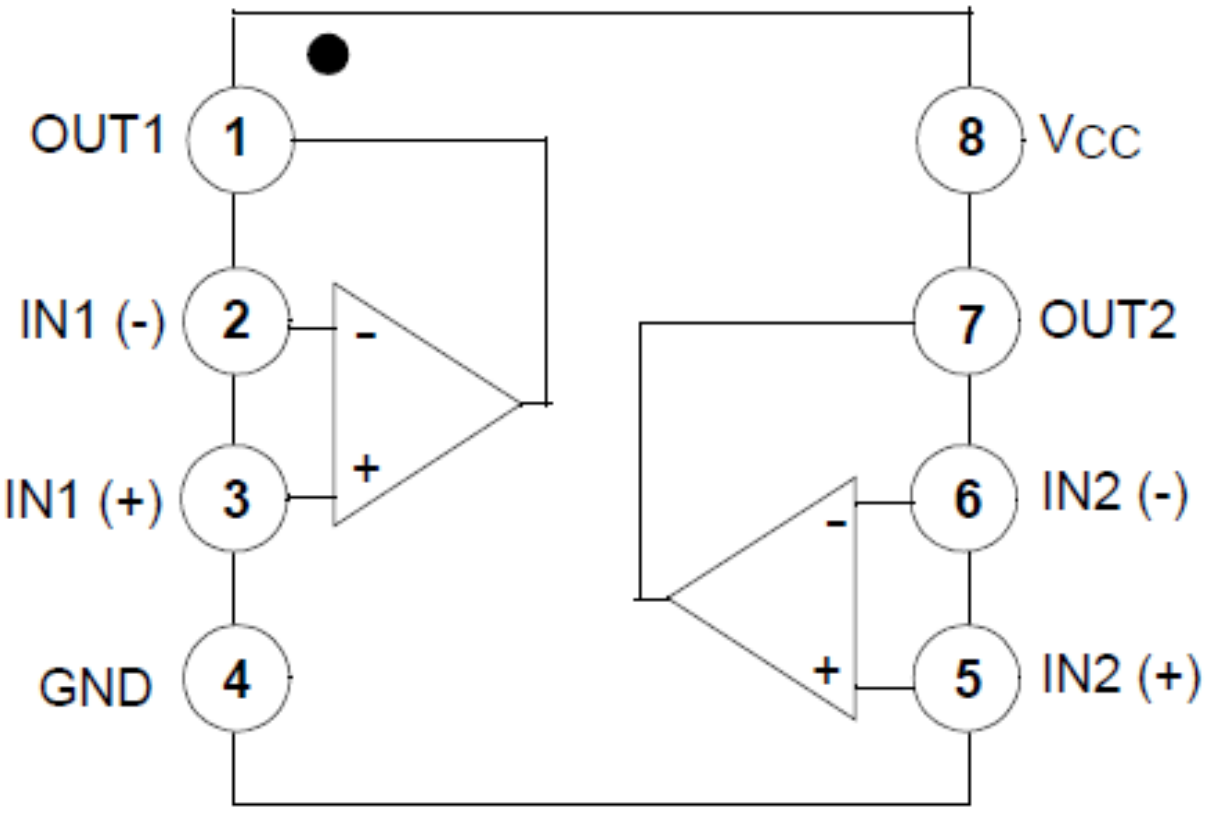
موارد کار برد این ای سی خیلی گسترده تر از ان است که بتوان در اینجا شرح داد اما چند موردی از انها را مثال میزنم :

الف : فرض کنید شما در ساختمان مسکونی خود می خواهید تمام لامپ های روشنایی خود را (مثلا ۵ لامپ در مکانهای مختلف ساختمان) دارای کنترل از راه دور کنید خب در این صورت لازم است به تعداد لامپ های روشنایی تان گیرنده و فرستنده کنترل از راه دور را داشته باشید تا هر کدام با فرکانس کاری خود موجب تداخل در دیگری نداشته باشد اینجاست که این ایسی خود را نشون میده یعنی شما میتوانید به تعداد لامپ روشنایی تان در هر لامپ یک گیرنده ساده ای با فرکانس مخصوص همون لامپ کار بگذارید و با یک فرستنده از راه دور تمام انها را کنترل کنید . چطور ؟؟؟ خیلی ساده است شما توسط این ایسی یک فرستنده میسازید که در قسمت اوسیلاتور ان یک مقاومت محاسبه شده را به تعداد مورد نیاز مثلا همون ۵ عدد تقسیم کرده و مقاومتها را بطور سری بهم وصل میکنید و توسط یک کلید چند حالته (مثلا سلکتور ۵ گام) نقطه اتصال بین راهی مقاومت ها را توسط همون سلکتور برای روشن کردن لامپ مورد نظر انتخاب میکنید . طبیعی است که با تغییر ظرفیت مقاومت در اوسیلاتور فرکانس کاری انهم تغییر کرده و ان فرکانسی را تولید خواهد کرد که در گیرنده مورد تطبیق داده شده تعریف شده است بنابرین مشاهده میکنید که بدون تداخل هر تک تک لامپ ها توسط یک کنترل دستی بطور مجزا قابل قطع و وصل خواهد بود .

ب : شما میتونید با مدوله کردن مثلا چندین کانال صوتی روی یک فرکانس حامل (کاریر) و بکار گیری چندین گیرنده در محل دیگر که به فرکانس های حامل مدوله شده تنظیم گشته اند سیگنال های صوتی مدوله شده را بطور مجزا وبدون تداخل در محل پیاده کرده واز تک تک انها به راحتی استفاده کنید .

پ : دیکود کردن فرکانسهای شماره گیر dtmf مدوله شده توسط شماره گیر تلفن و تبدیل انها به اعداد ده دهی از صفر الی نه و ...

بلوک دیاگرام آی سی **LM358**

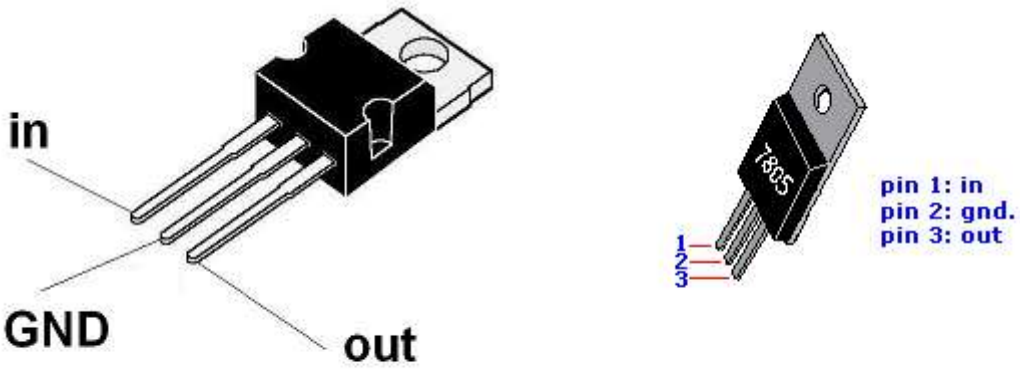


شکل ۷-۲ بلوک دیاگرام آی سی تقویت کننده LM358

حال به تشریح قطعات دیگری که در ساخت این مدار نقش داشتند می پردازیم .

آی سی رگولاتور ۷۸۰۵

از این قطعه برای ساخت ولتاژ یکسوشده و صاف برای قسمت های تغذیه میکروکنترلر و LCD استفاده می کنیم .



شکل ۸-۲ رگولاتور ۷۸۰۵

دیود زنر ۸.۲ ولت

از این دیود برای ولتاژ تغذیه آی سی LM358 استفاده می کنیم .

شکل ۹-۲ زنر ۸.۲ ولتی

کریستال ۱۶ مگاهرتز

در قسمت اسیلاتور مدار و تامین کلاک پالس مورد نیاز میکروکنترلر استفاده شده است .

نکته :

دو خازن C5 و C6 نیز باید حتما مثل شکل به کریستال وصل شوند.برای کارایی و بالا بردن سرعت و فرکانس باید این کار را انجام دهیم.اما دلیل اصلی آن گرفتن نویز از مدار می باشد که در واقع با این کار تمام مطالب گفته شده بالا را انجام داده ایم .

کلید **open** و **close** **S2 AC,DC**

این کلید اگر در حالت open باشد از اسیلوسکوپ در حالت AC استفاده کرده ایم.یعنی مدار فقط شکل موج های AC را اندازه گیری و به ما نشان می دهد.و بالعکس اگر در حالت close یا DC باشد فقط مقدار DC را به ما نشان می دهد .

کلید **S1**

از این کلید در مواقعی که ولتاژ ورودی زیاد می باشد باید از آن استفاده کرد.در واقع ورودی اسیلوسکوپ تا دامنه 2* برابر را می تواند در این حالت اندازه گیری کند .

پتانسیومتر **P1**

از این قطعه برای کالیبره کردن اسیلوسکوپ استفاده می کنیم بدین صورت که باید آنقدر آن را بچرخانیم تا خط صاف نمایش داده شده در LCD به سمت وسط آن بیاید.ما با این کار در واقع ولتاژ آفست تقویت کننده عملیاتی در آی سی را صفر کرده ایم و با این کار اسیلوسکوپ ما با نهایت دقت تنظیم شده و آماده اندازه گیری سیگنال می شود .

پتانسیومتر **P2**

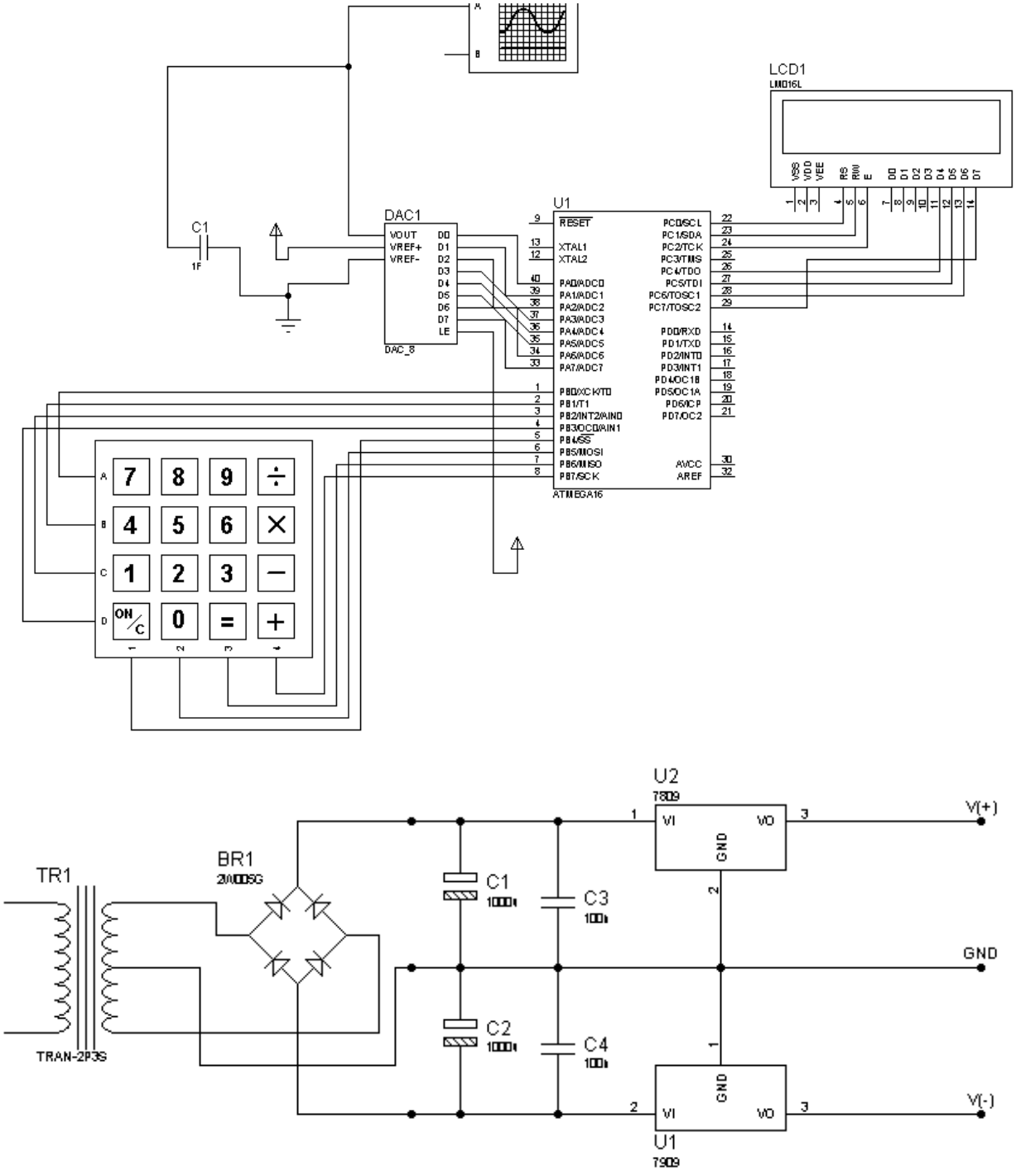
از این ولوم برای تنظیم روشنایی نور صفحه LCD استفتده می کنیم .

کلید های **S3** تا **S8**

از این کلید ها برای بالا بردن یا پایین آوردن و یا برای جمع کردن و یا باز کردن شکل موج استفاده میکنیم .

از کلید S6 نیز برای توقف وایستادن شکل موج هایی که تکان زیاد دارند و قابل مشاهده نیستند استفاده می شود.بدین صورت که با فشردن آن شکل موج در همان جایی که هست متوقف شده و قابل اندازه گیری می باشد .

شماتیک و نقشه فنی مدار سیگنال ژنراتور



شکل ۱۰-۲ نمای شماتیک سیگنال ژنراتور به همراه منبع تغذیه دوبل

مشخصات میکرو کنترلر **Atmega16**

این میکروکنترلرهای هشت بیتی دارای توان مصرفی پایینی بوده و در معماری آنها از ساختار پیشرفته RISK بهره گرفته شده است .

به عبارت دیگر این میکروکنترلرها دارای صد و سی و یک دستورالعمل ساده هستند که اغلب آنها در یک پالس ساعت اجرا می شوند اجرا شدن دستورالعملها در یک سیکل باعث افزایش سرعت این میکروکنترلرها گردیده است همچنین Atmega 16دارای سی و دو رجیستر همه منظوره هشت بیتی است و قابلیت اجرای حداکثر شانزده میلیون دستورالعمل در ثانیه را دارد این قابلیت یکی دیگر از دلایل افزایش سرعت این میکروکنترلرهاست .

Atmega 16دارای ۱۶ کیلو بایت حافظه فلش با قابلیت خواندن و نوشتن تا ده هزار مرتبه ،۵۱۲ بایت حافظه EEpro با قابلیت خواندن و نوشتن تا صد هزار بار و ۱ کیلوبایت حافظه داخلی SRAM می باشد.برای برنامه ریزی میکروکنترلرهای AVRمی توان از استاندارد JTAG استفاده نمود. این استاندارد برای برنامه ریزیFLASH ، EEpromفیوزها و Lockbitها از طریق رابط JTAG به کاربرده می شود .

یکی دیگر از مزیتهای میکروکنترلرهای AVR دارا بودن تجهیزات جانبی مختلف مورد نیازاست ، این تجهیزات که دارای کاربردهای متنوعی هستند، به شرح زیر می باشند .

۱- دارای دو شمارنده هشت بیتی و یک شمارنده شانزده بیتی است ، فرکانس کار این شمارنده ها به طور جداگانه تنظیم می شود. این شمارنده ها دارای واحد مقایسه هستند که برای ایجاد شکل موجهای PWM در مدهای مختلف به کار برده می شود .

۲- این میکروکنترلر دارای یک مبدل ADC با هشت کانال ده بیتی است هشت ورودی مبدل ADC با استفاده از مالتی پلکس داخلی انتخاب و به این مبدل اعمال می شوند انتخاب ورودیهای مختلف و ولتاژ مرجع با برنامه نویسی انجام می شود از طرف دیگر اگر ورودیهای Single Ended به این پایه ها اعمال شود، می توان هر هشت کانال را به طور جداگانه به کار گرفت حالت Single Ended زمانی است که ورودیها دارای زمین مشترک باشند در حالت دیفرانسیلی که ورودیها دارای پلاریته هستند (به عنوان مثال ولتاژ دو سر یک مقاومت در داخل یک مدار) نوع TQFP ، هفت کانال ورودی برای مبدل دارد و نوع PDIPآن که دارای چهل پایه است ، دو کانال ورودی ADC در اختیار قرار می دهد. همچنین در حالت PDIPمی توان بهره را به مقدارهای ۱۰×۲۰×۱× نیز تنظیم نمود .

۳- دارای رابط سریال TWI است که اتصال چندین میکروکنترلر را توسط دو باس دیتا و پالس فراهم می کند .

۴- قابلیت ارتباط سریال USART از دیگر مشخصات این میکروکنترلرهاست توضیح اینکه ارتباط با استفاده از پورت سریال USARTبه دو صورت سنکرون و آسنکرون صورت می گیرد .در حالت سنکرون از یک پالس ساعت برای همزمانی استفاده می شود. در حالت آسنکرون میکروکنترلر ورود و خروج اطلاعات را کنترل کرده و برنامه ریزی در این حالت ساده تر است .

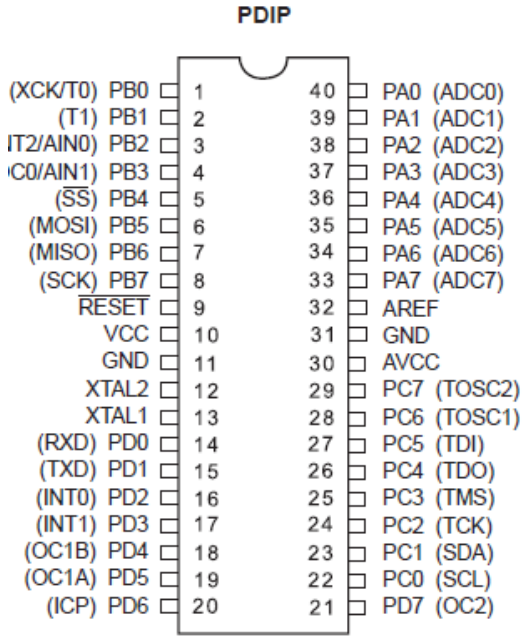
۵- دارای رابط سریال SPI است که در دو مد Master/Slave به کار گرفته می شود، نحوه استفاده از این رابط برای برنامه ریزی میکروکنترلرهای AVR بیان می شود .

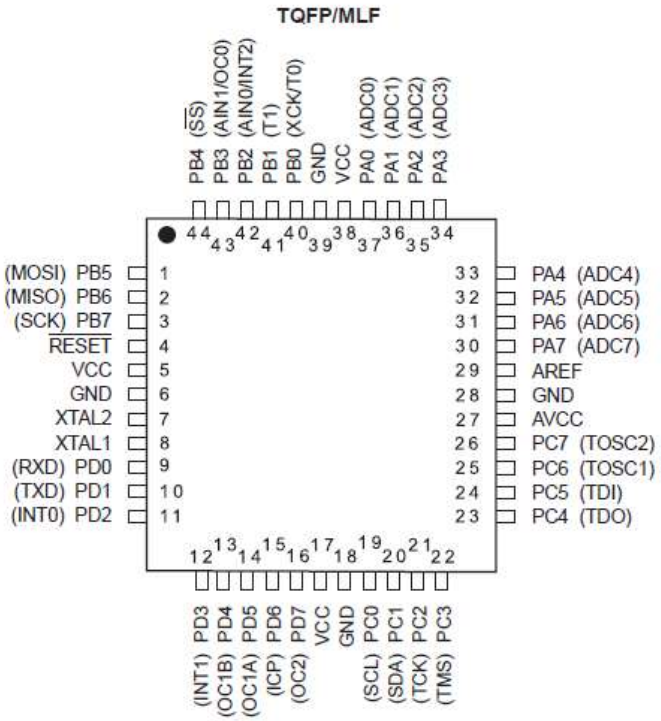
۶- شمارنده Watchdoge با اسیلاتور جداگانه ، که برای جلوگیری از هنگ کردن میکروکنترلر به کار می رود ، یکی دیگر از قسمت های جانبی این میکروکنترلرهاست در صورتی که تنظیمات لازم برای فعال شدن این شمارنده انجام شده باشد، با شروع به کار میکروکنترلر ، این شمارنده شروع به کار می کند. برنامه نویس با توجه به مدت زمان اجرای دستورالعملها ، در زمان مشخصی قبل از رسیدن شمارنده به انتهای سیکل کاری خود ، با استفاده از دستور WDR شمارنده را ریست می کند. حال اگر میکروکنترلر به دلایلی از کار افتاده باشد، دستور WDR اجرا نشده، شمارنده ریست نمی شود. در نتیجه Watchdogتا انتهای سیکل کاری خود شمارش کرده و میکروکنترلر را ریست نموده ، خود از ابتدا شروع به شمارش می کند .

۷- مقایسه کننده آنالوگ داخلی یکی دیگر از تجهیزات جانبی این میکروکنترلرهاست ورودیهای این مقایسه کننده از پورت Bتامین می شود .

حداکثر کریستال مورد استفاده در این آی سی تا ۱۶ مگاهرتز برای Atmega16L و ۸ مگاهرتز برای Atmega16 می باشد. ولتاژ کاری آن هم بین ۲.۷تا ۵.۵ ولت برای Atmega16L می باشد.و برای Atmega16 از ۴.۵ تا ۵.۵ ولت می باشد .

شکل ظاهری و نمای پایه های **Atmega16**





شکل ۱۱-۲ نمای ظاهری Atmega 16

LCD کاراکتری ۱۶*۲

این نوع lcd دارای دو سطر میباشد که در هر سطر آن تعداد ۱۶ کاراکتر یا حرف می شود نوشت دارای ۱۶ پایه می باشد این lcdبه راحتی به آیسی میکرو وصل میشود و ما میتوانیم هر متغییر یا حرفی را در آن نمایش دهیم مثلا می توانیم میزان دمای هوا را بصورت دیجیتالی در آن نمایش دهیم یا پیغام مربوط به سیستمی را در آن نمایش دهیم.شکل ظاهری این lcd در عکس زیر ببینید نمای پشت , lcd 16*2 و نمای روبرو :

شکل ۱۲-۲ نمای ظاهری LCD

شکل ۱۳-۲ نمای شماتیک LCD

همونطور که گفتم این lcd دارای ۱۶ پایه هست که اسماشون به ترتیب زیر هست :

تغذیه این lcd بین ۴.۵ تا ۵ ولت میباشد .

1- Vss (زمین)

2 - Vdd(مثبت)

3- Vee یا vo (برای تنظیم کنتراست lcd)

4- RS (به میکرو وصل میشه)

5- R/W (این پایه باید به زمین وصل شود تا lcd کار کند)

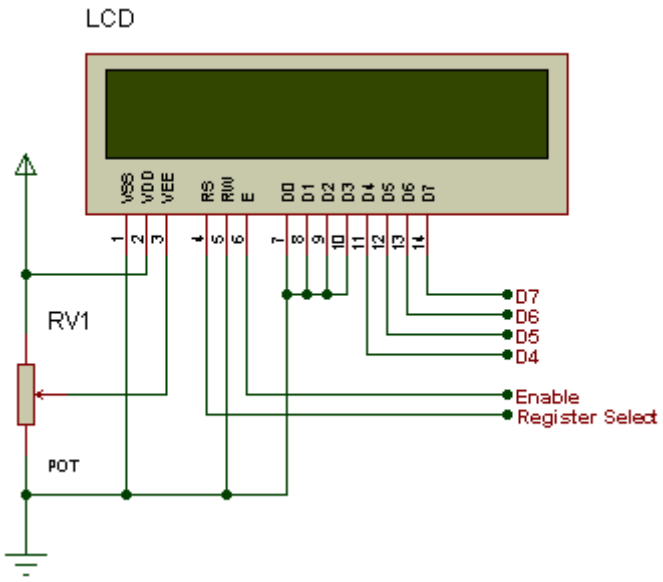
6- E (به میکرو وصل میشود)

D0,D1,D2,D3 این پایه های دیتا میباشد که به جایی وصل نمیشود (آزاد میمانند)

D4,D5,D6,D7 این پایه ها به پایه های میکرو وصل میشوند به کدوم پایه ها فرق نمی کند فقط باید برنامه میکرو را تنظیم کنیم

15- A(آند led که برای روشنایی صفحه lcd میباشد که باید به مثبت وصل شود)

16- K(کاتد led که باید به منفی (زمین) وصل شود)



شکل ۱۴-۲ نحوه اتصال پایه ها

در نقشه بالا میبینید که یک پناسیومتر به پایه ۳ وصل شده است که برای تغییر کنتراست lcd میباشد این پناسیومتر را میتوانید ۱۰کیلو یا ۱کیلو انتخاب کنید وهمچنین در این نقشه پایه های ۷و۸و۹و۱۰ را به زمین وصل کرده ولی اگر هم وصل نکنید مشکلی نیست ۶ پایه ای که اینور هستند باید به میکرو وصل کنیم و همچنین پایه های ۱۶و۱۵ در این نقشه نشان داده نشده که آند وکاتد led میباشند که همون طور که در بالا گفتم باید وصل شوند .

آی سی **DAC0800**

مدار های میکروپروسسوری با سطوح منطقی صفر و یک عمل می کنند. اما مواقعی پیش می آید که ما با مقادیر پیوسته (آنالوگ) سرو کار داریم , بنابراین اگر بخواهیم از طریق یک سیستم میکروپروسسوری سیستمی با مقادیر پیوسته (آنالوگ) کنترل کنیم لازم است از یک آیسی واسطه برای تبدیل اعداد باینری به مقدار آنالوگ آن استفاده کنیم. به این آی سی ها مبدل دیجیتال به آنالوگ و یا DAC گفته می شود.

این آی سی ها مقادیر باینری را با توجه به ولتاژی که به پایه ورودی آن داده می شود می سازد.

آی سی که ما در این پروژه استفاده کردیم DAC0800 می باشد که یک مبدل دیجیتال به انالوگ ۸ بیتی می باشد.

هدف ما در استفاده از این آی سی در این پروژه این است که توسط یک عدد از این آی سی ها یک موج سینوسی تولید کنیم.

برای ایجاد موج سینوسی ما به مقادیر آنالوگ احتیاج داریم که با کمک این آی سی مقادیر میکرو را آنالوگ خواهیم کرد.

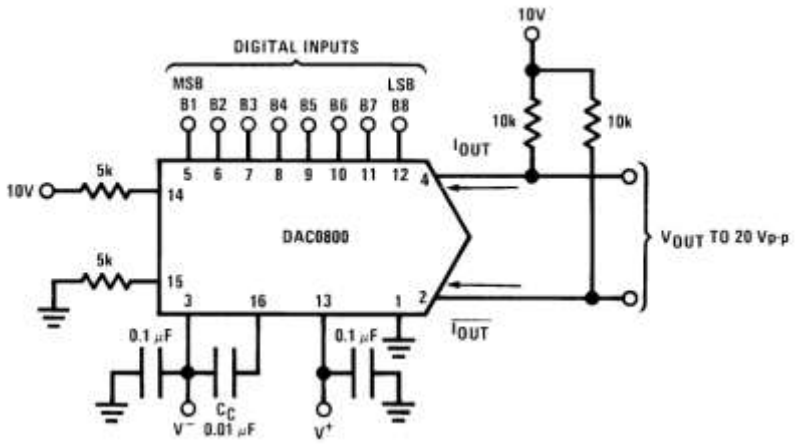
پایه های ۵ تا ۱۲ این آی سی ورودی های این ۸ بیت می باشد.

که پایه شماره ۵ با ارزش ترین بیت و پایه ۱۲ کم ارزش ترین بیت می باشد.

پایه ۱۴ ورودی منفی و پایه ۱۵ ورودی مثبت برای ایجاد خروجی مناسب می باشند.

سایر اطلاعات مربوط به این آی سی و نقشه راه انداز آن را می توانید از داخل فایل شماتیک این آی سی مشاهده کنید.

نحوه اتصال و راه اندازی آی سی DAC0800



شکل ۱۵-۲ راه اندازی آی سی

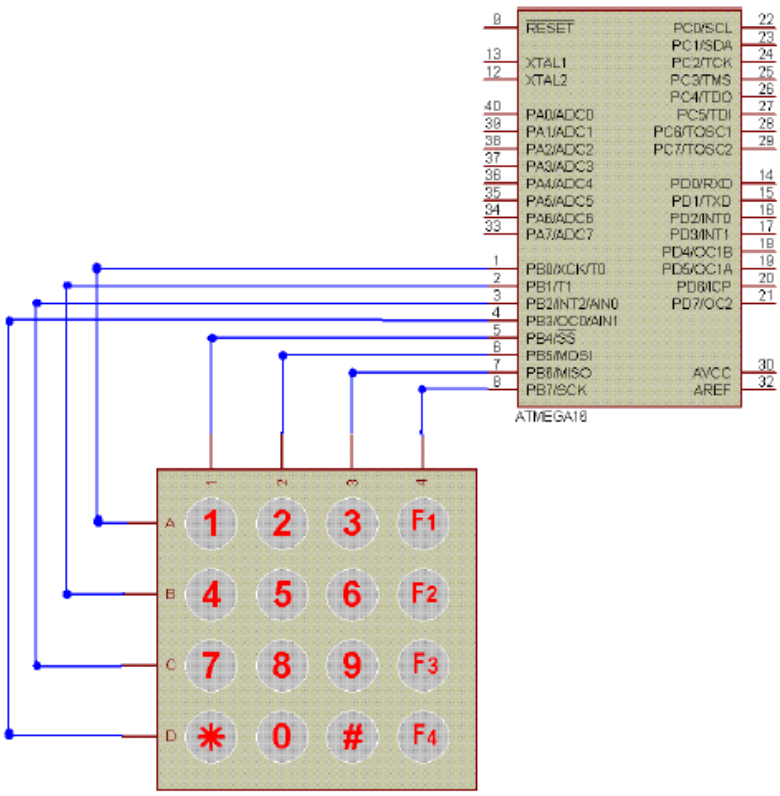
جدول اتصال پایه های آی سی **DAC0800**

Pin Number	Description
1	Vlc - Threshold Control
2	Iout - Current Out (Inverse)
3	V- - Negative Supply
4	Iout - Current Out
5	B1 - Digital Input Bit 1
6	B2 - Digital Input Bit 2
7	B3 - Digital Input Bit 3
8	B4 - Digital Input Bit 4
9	B5 - Digital Input Bit 5
10	B6 - Digital Input Bit 6
11	B7 - Digital Input Bit 7
12	B8 - Digital Input Bit 8
13	V+ - Positive Supply
14	Vref+ - Voltage Reference +
15	Vref- - Voltage Reference -
16	COMP - Compensation

جدول ۱-۲ اتصال پایه های آی سی

KEYBOARD

نحوه اتصال کیبورد به میکروکنترلر



شکل ۱۶-۲ اتصال کیبورد به میکروکنترلر

نحوه کار به این شکل است که ابتدا چهار پین از میکرو که به سطرهای صفحه کلید متصل است به صورت خروجی تعریف می شود و ۴ پین دیگر به عنوان ورودی به ستون ها متصل می شود.

همین طور مقدار اولیه ورودی ها نیز ست می شود تا هنگام خواندن از پورت مقدار ورودی صفر که نشان دهنده اتصال یا همان

فشرده شدن کلیدی از کی پد است مشخص می شود.

در صورتی که یکی از پین های ورودی برابر صفر باشد , یعنی کلیدی فشرده شده است.

ما با خواندن از پین های ورودی می توانیم سطری که کلید فشرده شده در آن قرار دارد را مشخص کنیم.

برای مشخص شدن ستون و در نهایت کلید فشرده شده این بار چهار پینی که به سطرها متصل است را ورودی تعریف می کنیم

و با مقداردهی و خواندن از پورت می توانیم سطر مورد نظر و در نهایت کلید فشرده شده را پیدا کنیم.

به این روش روش سرکشی می گویند.

فصل سوم

در این فصل ما به توضیح نرم افزاری و معرفی نرم افزارهای به کار گرفته شده در ساخت این پروژه می پردازیم . ابتدا به معرفی نرم افزار استفاده شده در ساخت سیگنال ژنراتور و همچنین به نحوه پروگرام کردن میکروکنترلر می پردازیم .

برنامه **Codevision AVR**

Codevision AVR یک کامپایلر C , محیط توسعه یافته یکپارچه (IDE) و تولید کننده خودکار کد های برنامه می باشد که برای کار با میکروکنترلرهای AVR ساخت شرکت ATMEL طراحی شده است.

برنامه طوری طراحی شده که در ویندوزهای XP , 2000 , NT , ME , 98 , 95 قابل اجرا باشد.

این برنامه یک پروگرامر ISP را هم شامل می شود که امکان انتقال کد های برنامه به میکروکنترلرها را بعد از انجام موفق عمل کامپایل فراهم می کند.

این نرم افزار علاوه بر کتابخانه های C استاندارد , دارای کتابخانه های دقیقی برای کار با LCD های کاراکتری , تولید وقفه , تنظیم انرژی مصرفی تراشه , قابلیت SPI , قابلیت I2C , ارتباط یک سیمه , کار با سنسورهای دمای LM75 , DS1820 و EEprom های سریال را دارد.

شکل ۳-۱ نرم افزار code vision AVR

ایجاد یک پروژه جدید

اولین کار ایجاد پروژه جدیدی برای برنامه مورد نظر است.می توانید این کار را با استفاده از منوی File / New و یا فشار دادن دکمه Create New File در نوار ابزار انجام دهید.

با انجام این کار پنجره ای باز می شود (شکل ۳-۲).شما باید File type /Project را انتخاب نموده و دکمه OK را فشار دهید.

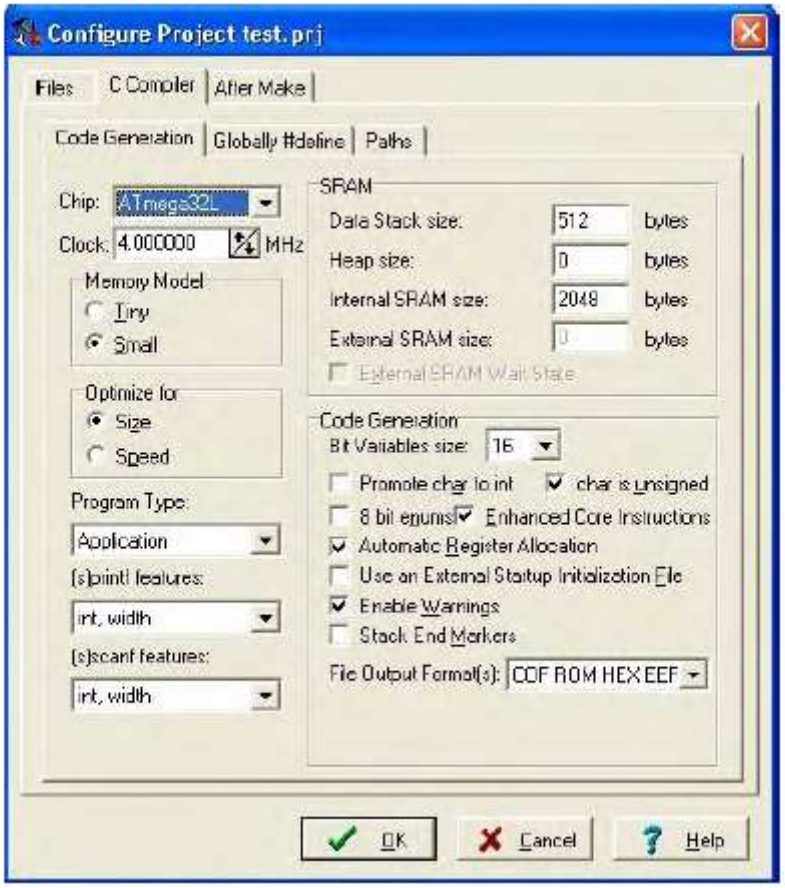
شکل ۳- ۲ پنجره Create New File

حالا پنجره دیگری باز می شود (شکل ۳-۳) و از شما سوال می کند که آیا می خواهید پروژه جدید را به کمک Code Wizard AVR انجام دهید.

شکل ۳- ۳ پنجره Confirm

در این قسمت دکمه NO را انتخاب کنید با این کار پنجره Create New Project باز خواهد شد (شکل ۳-۴) که باید در آن نام فایل پروژه و محل آن را مشخص نمایید.فایل پروژه با پسوند prj. ذخیره خواهد شد.

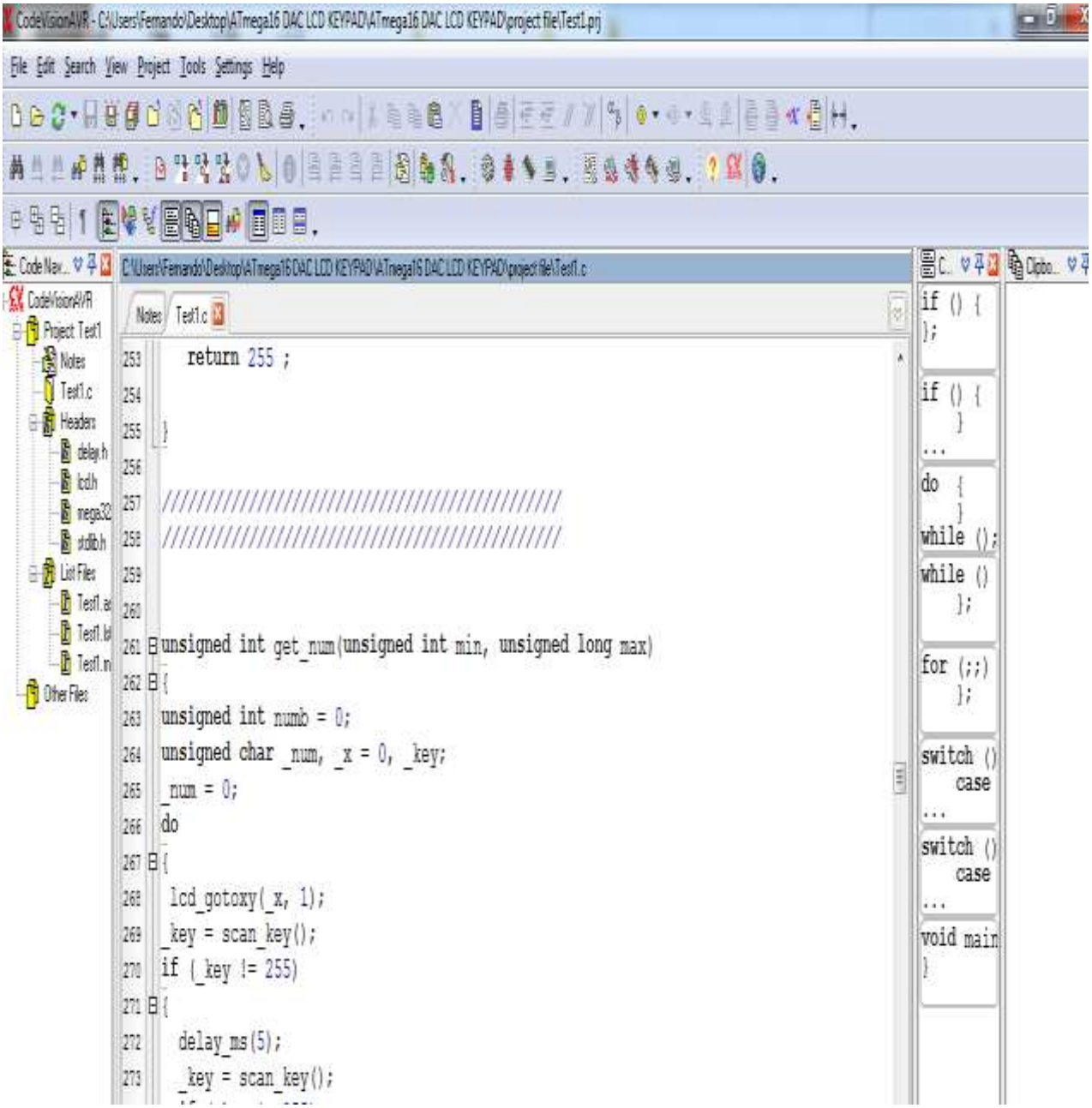
در پنجره جدید باز شده در قسمت chip نوع میکروی مورد نظر که در اینجا Atmega16 می باشد را انتخاب و همچنین در قسمت clock کلاک مورد نظر را انتخاب می کنیم.چون در این مدار از کریستال خارجی ۱۶ مگاهرتز استفاده کرده ایم آن را بر روی ۱۶ مگاهرتز می گذاریم.



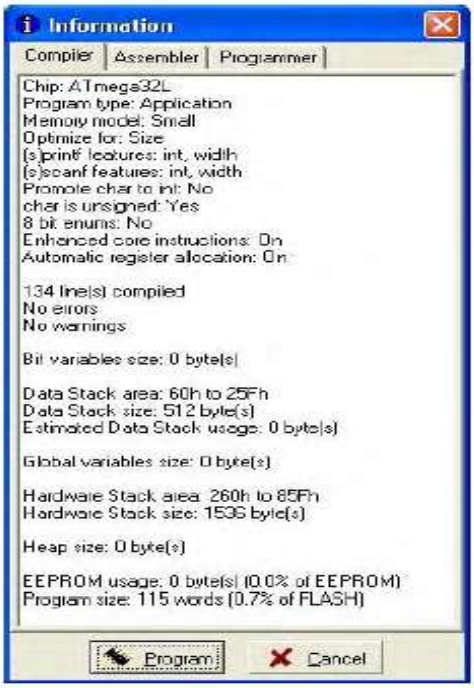
شکل ۴-۳ محیط قسمتی از نرم افزار

پس از نوشتن برنامه در محیط نوشتاری code vision و پس از make کردن فایل به صورت زیر برنامه را داخل چیپ مورد نظر می ریزیم . باید توجه داشته باشید که دستگاه پروگرامر را از طریق پورت USB به کامپیوتر وصل کرده باشید و میکروی مورد نظر را نیز داخل سوکت پروگرامر قرار داده باشید.

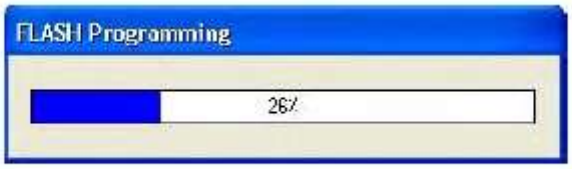
پس از انجام این مراحل به شکل زیر برنامه را داخل آی سی پروگرام می نمائیم .



شکل ۵-۳ تصویری از محیط برنامه نویسی نرم افزار



شکل ۶-۳ گزینه پروگرام برنامه



شکل ۷-۳ برنامه در حال ریختن فایل HEX روی آی سی

توضیحات مربوط به برنامه مدار و نحوه کارکرد سیگنال ژنراتور

در مورد برنامه:

برای تولید موج سینوسی توسط آی سی های مبدل ما باید مقادیر متناظر با ولتاژ لحظه ای این موج را توسط میکروکنترلر ایجاد کرده و به آی سی مبدل بدهیم تا در خروجی موج سینوسی داشته باشیم.

البته به دلیل اینکه حداکثر این آی سی می تواند ۲۵۶ حالت داشته باشد موج ما سینوسی کامل نیست اما بسیار شبیه است.

مقدار لحظه ای یک موج سینوسی در واحد باینری با فرمول زیر محاسبه می شود.

((sin ((x/256)*(2*PI))+1)/2)*255

به دلیل اینکه محاسبه این فرمول در هنگام اجرای برنامه میکروکنترلر فرایندی زمان بر است و ما برای ایجاد یک موج دقیق تر احتیاج به سرعت بالا داریم, به همین منظور این مقادیر را از قبل حساب کرده و داخل میکروکنترلر قرار می دهیم

به این گونه اطلاعات جداول لوک آپ گفته می شود.

به دلیل اینکه اعداد باینری از صفر شروع می شوند لذا ما مجبوریم از عدد ۱۲۸ که نصف ۲۵۶ می باشد برای حالت صفر موج استفاده کنیم.تا اعداد پایین تر برای سیکل منفی و اعداد بالاتر برای سیکل مثبت این موج مورد استفاده قرار گیرد.

این فرمولی که در بالا ذکر شد نیز اعداد را طبق خواسته ما بدست می آورد.

در این برنامه چون ما باید دامنه را نیز کنترل کنیم لذا از ۵ جدول لوک آپ با تفاوت ۲۰ درصد استفاده کردیم.

در هنگام بازخوانی از این جداول با توجه به دامنه تنظیم شده توسط کاربر از جدول مناسب استفاده می شود.

همانطور که گفته شد هدف ما از این پروژه ساخت موج سینوسی با قابلیت های تنظیم فرکانس و دامنه موج می باشد.

برای تنظیم فرکانس موج مورد نظر باید مقدار تایمر را تغییر دهیم تا وقفه سر ریزی تایمر طبق زمان بندی ما رخ دهد.

کدی که داخل روتین وقفه تایمر نوشته ایم کد ساده ای می باشد که در زیر می بینیم:

```
TCNT1=T1;

Dac1 = source[Psec][ph1];

Dac2 = source[Psec][ph2];


Ph1+=2;

Ph2+=2;
```

TCNT1

که در خط اول مقدار دهی شده است , رجیستر مقدار ۱۶ بیتی تایمر یک می باشد.

با مقدار دهی این رجیستر می توان فرکانس موج را تنظیم کرد.

متغیر T1 نیز مقدار مناسب برای فرکانس تعیین شده را در خود نگهداری می کند.

دو مولفه ای که در خط دوم و سوم این دستورات مقدار دهی شده اند دو پورت از میکروکنترلر هستند که برای سادگی توسط

دو دستور زیر با این اسامی نام گذاری شده اند:

```
#define dac1 PORTA

#define dac2 PORTD
```

یعنی پایه های دیتای مبدل اول به PORTA از میکرو متصل شده و پایه های دیتای مبدل دوم نیز به PORTD میکروی

ما متصل شده است.

کار با LCD با کامپایلر Codevision بسیار یاده است.

برای راحتی کار می توانیم توسط ابزار Codewizard به تب LCD رفته و پورت مورد نظر خود را برای کنترل LCD معرفی کنیم.

در برنامه ما از پورت C برای LCD استفاده کرده ایم.

توسط دستور هایی از قبیل

Lcd_putsf

Lcd_puts

اطلاعات خود را روی صفحه LCD نمایش داده ایم.

در این برنامه از یک صفحه کلید 4*4 استفاده شده که طریقه خواندن کلید فشرده شده در آن به اصطلاح روش سرکشی گفته

می شود.

ما برای این کار کتابخانه ای طراحی کرده ایم تا بتوانیم در برنامه های دیگر نیز به راحتی از صفحه کلید استفاده کنیم.

حال به شرح مختصری راجع به این کتابخانه می پردازیم:

این کتابخانه کلا دو تابع دارد:

```
Void keypad_init(void);
Unsigned char scan_key(void);
```

تابع اول برای مقدار دهی اولیه به پورت مورد نظر و تعین ورودی و خروجی بودن پین های پورت مربوطه به کار رفته است.

```
Void keypad_init(void)
{
Keypad_DDR=0x0f;
Keypad_PORT=0xF0;
}
```

نکته مهمی که باید به آن توجه کنیم این است که برای استفاده از این کتابخانه باید در برنامه خود سه ماکروی زیر را به این

شکل تعریف کنیم:

```
#define keypad_DDR DDRB
```

#define keypad_PORT PORTB

#define keypad_PIN PINB

این کار برای این است که برنامه نویس هر پورتی را که لازم داشت برای استفاده از Keypad مورد استفاده قرار دهد.

تابع دوم تابعی است که برای دریافت کد کلید فشرده شده از آن استفاده می کنیم.

توابعی که در برنامه اصلی هستند به شرح زیر می باشند:

Unsigned int get_num(unsigned int min,unsigned int max)

که برای گرفتن اعداد از keypad می باشد.

برنامه به شکلی است که عدد خوانده شده از صفحه کلید بین مینیمم و ماکزیمم تعیین شده در ورودی های این تابع است.

Void lcd_panel1(void)

که برای نمایش اطلاعات فرکانس و دامنه و ... روی صفحه LCD و فعال شدن سرویس وقفه ها به کار می رود.

Void panel2(void) //change freq

این تابع را هنگامی فراخوانی می کنیم که کلید فانکشن اول از کی پد فشرده شده باشد.کار این تابع این است که فرکانس موج ما را طبق درخواست کاربر و توسط صفحه کلید تامین می کند.

Void panel3(void) //change scale

این تابع را هنگامی فراخوانی می کنیم که کلید فانکشن سوم از کی پد فشرده شده باشد.کار این تابع این است که دامنه فرکانس را توسط صفحه کلید از کاربر دریافت کند.

سورس برنامه به زبان **C**

```
Chip type           : ATmega16
Program type        : Application
Clock frequency     : 16.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <stdlib.h>
#include <delay.h>

#define keypad_DDR DDRB
#define keypad_PORT PORTB
#define keypad_PIN PINB
#define key_ctr 5
#define key_delay 1


#define D20  0
#define D40  1
#define D60  2
#define D80  3
#define D100 4
#define F1hz 125000
#define dac1 PORTA
```

```
#define dac2 PORTD

#define F1 14

#define F2 24

#define F3 34

#define _FF_ 0.71112


unsigned char const table[4][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}};

// Alphanumeric LCD Module functions

#asm

   .equ __lcd_port=0x15 ;PORTC

#endasm

#include <lcd.h>


// Declare your global variables here

unsigned char Psec = D100;  //perSec Damane Freq

unsigned char ph1 = 0;      //zavie phase 1

unsigned char ph2 = 0;      //zavie phase 2

unsigned long T1;           //Timer Numeric

unsigned int frequnce = 100;

unsigned int radu = 0;

unsigned char scale = 100;

// Timer 1 overflow interrupt service routine


interrupt [TIM1_OVF] void timer1_ovf_isr(void)

{

// Reinitialize Timer 1 value

TCNT1=T1;

dac1 = source[Psec][ph1];

dac2 = source[Psec][ph2];

ph1+=2;
```

```
ph2+=2;

//TCNT1L=0xAA;
// Place your code here

}

// Declare your global variables here
////////////////////////////////////////////////
////////////////////////////////////////////////

unsigned char scan_key(void)
{
  unsigned char i=0,j=0,key_ctr_temp=0,key_temp=0,scan_code[4]={0xFE,0xFD,0xFB,0xF7};

  for (i=0;i<4;i++)
  {
      keypad_PORT|=0x0F;
      keypad_PORT&=scan_code[i];
      delay_ms(5);
      //key_temp=keypad_PIN;
      if (keypad_PIN<0xF0)
      {
        key_temp=keypad_PIN;
        for(j=0;j<key_ctr;j++)
        {
          delay_ms(key_delay);
          if(keypad_PIN==key_temp)
            key_ctr_temp++;
        }
```

```
//          if(key_ctr_temp==key_ctr)
        {    key_ctr_temp = keypad_PIN;
             switch(key_ctr_temp)
             {
                  case 238:
                       return 1;
                       break;
                  case 237:
                       return 4;
                       break;
                  case 235:
                       return 7;
                       break;
                  case 231:
                       return 41;
                       break;
                  case 222:
                       return 2;
                       break;
                  case 221:
                       return 5;
                       break;
                  case 219:
                       return 8;
                       break;
                  case 215:
                       return 0;
                       break;
                  case 190:
                       return 3;
```

```
            break;
        case 189:
            return 6;
            break;
        case 187:
            return 9;
            break;
        case 183:
            return 43;
            break;
        case 126:
            return 14;
            break;
        case 125:
            return 24;
            break;
        case 123:
            return 34;
            break;
        case 119:
            return 44;
            break;
        default:
            return 255;
            break;
    }
  }

}
```

```
  }



  return 255 ;


}


//////////////////////////////////////////////

//////////////////////////////////////////////



unsigned int get_num(unsigned int min, unsigned int max)

{

unsigned int numb = 0;

unsigned char _num, _x = 0, _key;

_num = 0;

do

{

 lcd_gotoxy(_x, 1);

_key = scan_key();

if (_key != 255)

{

  delay_ms(5);

  _key = scan_key();

  if (_key != 255)

  {

    switch (_key)

    {

     case 1  : if (_x < 4) {_num = 1; numb = numb *10; numb += _num; lcd_putsf("1"); _x++;
delay_ms(150);} break;
```

```
    case 2  : if (_x < 4) {_num = 2; numb = numb *10; numb += _num; lcd_putsf("2"); _x++;
delay_ms(150);} break;

    case 3  : if (_x < 4) {_num = 3; numb = numb *10; numb += _num; lcd_putsf("3"); _x++;
delay_ms(150);} break;

    case 4  : if (_x < 4) {_num = 4; numb = numb *10; numb += _num; lcd_putsf("4"); _x++;
delay_ms(150);} break;

    case 5  : if (_x < 4) {_num = 5; numb = numb *10; numb += _num; lcd_putsf("5"); _x++;
delay_ms(150);} break;

    case 6  : if (_x < 4) {_num = 6; numb = numb *10; numb += _num; lcd_putsf("6"); _x++;
delay_ms(150);} break;

    case 7  : if (_x < 4) {_num = 7; numb = numb *10; numb += _num; lcd_putsf("7"); _x++;
delay_ms(150);} break;

    case 8  : if (_x < 4) {_num = 8; numb = numb *10; numb += _num; lcd_putsf("8"); _x++;
delay_ms(150);} break;

    case 9  : if (_x < 4) {_num = 9; numb = numb *10; numb += _num; lcd_putsf("9"); _x++;
delay_ms(150);} break;

    case 0  : if (_x < 4) {_num = 0; numb = numb *10; numb += _num; lcd_putsf("0"); _x++;
delay_ms(150);} break;

    case 43 : if (_x > 0) {_x--; lcd_gotoxy(_x, 1); lcd_putsf(" "); numb = numb / 10; delay_ms(150);};
break;

    case 44 : if (numb < min) {numb = min; }; if (numb > max) {numb = max; }; break;

   }

   lcd_gotoxy(_x, 1);

 }

}

} while(_key != 44);

//numb = max-min;

return numb;

}


void lcd_panel1(void)

{

char s_s[5];

#asm("cli")
```

```
lcd_clear();
lcd_gotoxy(0,0);
lcd_putsf("Freq :      hz. ");
lcd_gotoxy(0, 1);
lcd_putsf("R :   . Y-G:%   ");
itoa(frequnce, s_s);
lcd_gotoxy(7, 0);
lcd_puts(s_s);
itoa(radu, s_s);
lcd_gotoxy(3, 1);
lcd_puts(s_s);
itoa(scale, s_s);
lcd_gotoxy(13, 1);
lcd_puts(s_s);
#asm("sei")
}

void panel2(void)          // change freq
{
#asm("cli")
lcd_clear();
lcd_gotoxy(0, 0);
lcd_putsf("2000 > Freq > 9");
frequnce = get_num(10, 1300);
ph1 = 0;
ph2 = radu * _FF_;
T1 = ((F1hz / frequnce) - 34)*-1;
TCNT1 = T1;
lcd_panel1();
}
```

```
void panel3(void)          // change phase
{
#asm("cli")
lcd_clear();
lcd_gotoxy(0, 0);
lcd_putsf("Phase R  (0~359)");
ph1 = 0;
radu = get_num(0, 359);
ph2 = radu * _FF_;
T1 = ((F1hz / frequnce) )*-1;
TCNT1 = T1;
lcd_panel1();
}

void panel4(void)          // change scale
{
#asm("cli")
lcd_clear();
lcd_gotoxy(0, 0);
lcd_putsf("Scale 1~5 * %20");
Psec = get_num(1, 5) - 1;
switch(Psec)
{
case 0 : scale = 20; break;
case 1 : scale = 40; break;
case 2 : scale = 60; break;
case 3 : scale = 80; break;
case 4 : scale = 100; break;
}
```

```
ph1 = 0;
ph2 = radu * _FF_;
T1 = ((F1hz / frequnce) - 34)*-1;
TCNT1 = T1;
lcd_panel1();
}

void main(void)
{

unsigned char keybd;
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// Port B initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTB=0xF0;
DDRB=0x0F;

// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
```

```
PORTC=0x00;
DDRC=0xFF;

// Port D 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=0 State2=0 State1=0 State0=0
PORTD=0x00;
DDRD=0xFF;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 16000.000 kHz
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: On
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
```

```
TCCR1A=0x00;
TCCR1B=0x01;
TCNT1H=0xAA;
TCNT1L=0xAA;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x04;
```

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

// LCD module initialization
lcd_init(16);
// Global enable interrupts
frequnce = 100;
T1 = ((F1hz / frequnce) -33)*-1;
lcd_panel1();
while (1)
      {
      // Place your code here
//      keybd = get_kbd();
//      if (keybd != 255)
//      {
//      delay_ms(5);
      keybd = scan_key();
      if (keybd != 255)
      {
        switch (keybd)
        {
           case F1 : panel2(); break;
           case F2 : panel3(); break;
           case F3 : panel4(); break;
        }
      }
```

```
//    }
    };
```

نرم افزار مربوط به برنامه نویسی و پروگرام مدار اسیلوسکوپ

برنامه **AVR Studio 4**

شکل ۸-۳ نرم افزار AVR Studio 4

نرم افزار ارائه شده توسط شرکت ATMELبه نام AVR Studio 4

این نرم افزار به صورت رایگان در سایت شرکت ATMEL قرار دارد می توانید با رجوع به آدرس http://www.atmel.com آن را دانلود کنید.
این نرم افزار در حقیقت یک اسمبلر برای محصولات AVR اتمل است و به صورت کاملا ویژوالی است. می تواند با انواع دستگاههای برنامه نویس میکرو ارتباط برقرار کند و کدها را در میکرو دانلود کند. و قابلیت ترجمه کدها به زبانهای C و Assembly را دارد و ...

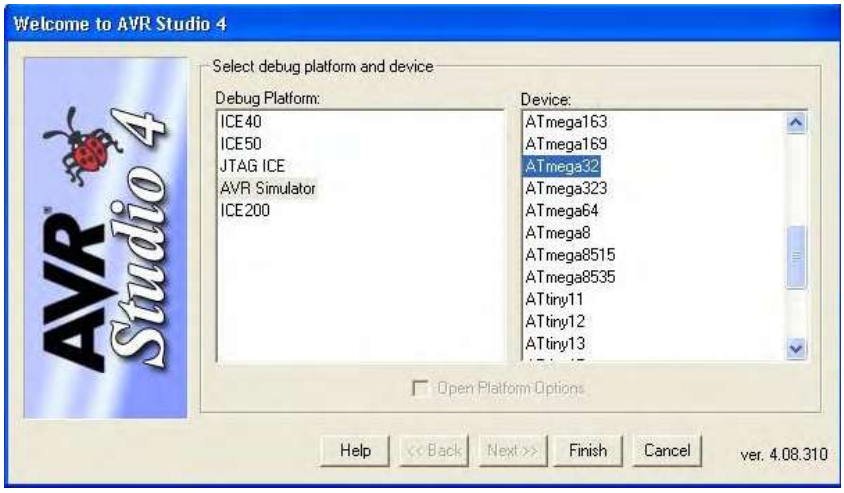
انواع برنامه نویسها که AVR Studio 4 با آنها سازگار است :

در این قسمت خصوصیات پروگرامرهایی را که با این نرم افزار سازگاری دارند را بر روی جدول برای شما آورده ایم .

Starter Kits	In System Programmers	Emulators Platforms
STK500	AVRISP	ICE 40/50
STK501	JTAGICE	JTAGICE
STK502		

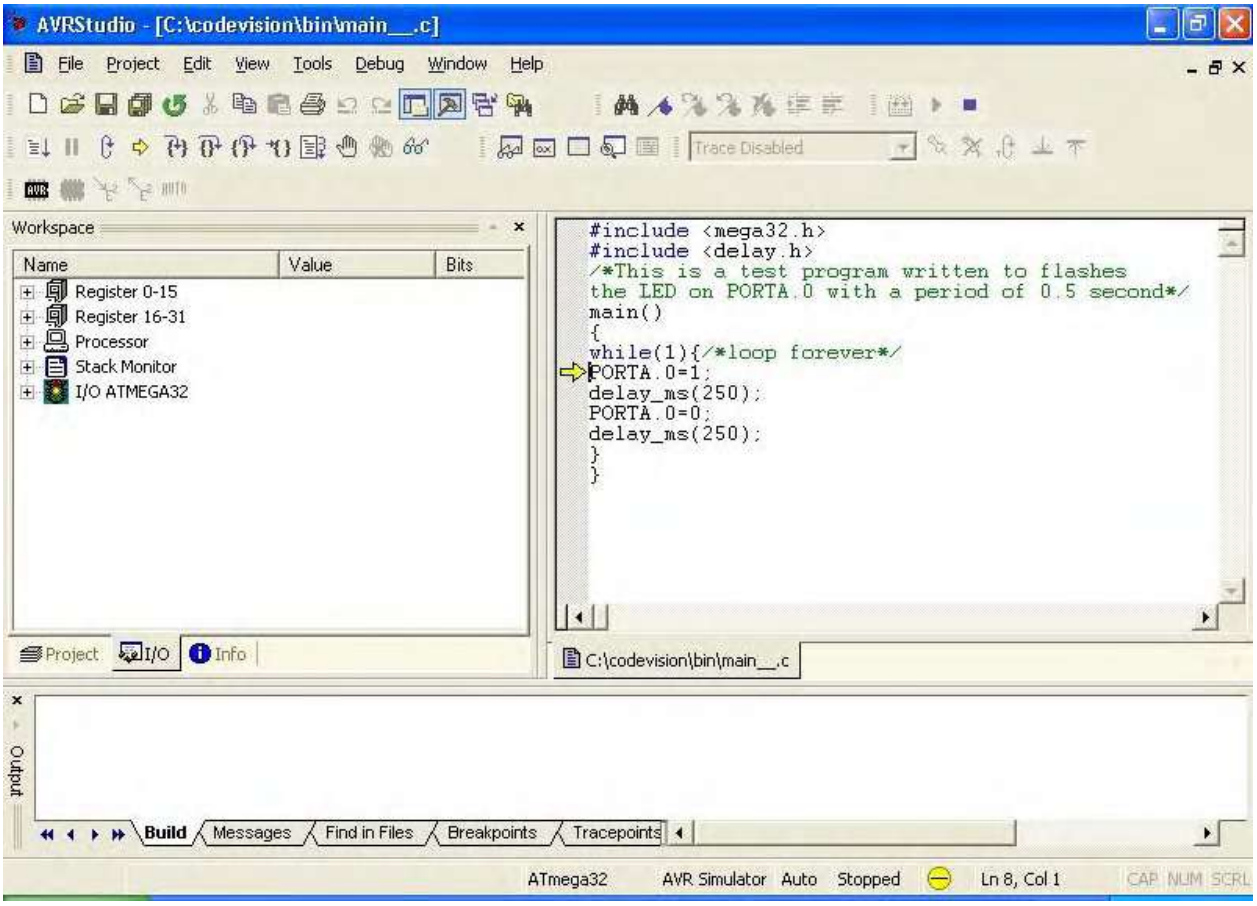
جدول ۱-۳ انواع پروگرامرهای سازگار با AVR Studio

پس از باز کردن نرم افزار گزینه create new project را انتخاب نموده و در صفحه بعدی باز شده نوع پروگرامر مورد نظرکه در اینجا از نوع STK500 می باشد را انتخاب و سپس همانند شکل زیر نوع چیپ مورد نظر را انتخاب و گزینه ok را می زنیم .



شکل ۹-۳ محیط انتخاب میکرو در AVR Studio

در پنجره جدید باز شده برنامه مورد نظر را به زبان C نوشته و آن را داخل میکرو می ریزیم.



شکل ۱۰-۳ محیط برنامه نویسی AVR Studio

نحوه کار مدار اسیلوسکوپ

چگونه شکل موج ها بر روی **LCD** محاسبه و سپس نمایش داده می شوند؟

تمامی این کار ها به صورت نرم افزاری در مدار اجرا می شوند و مراحل آن به صورت زیر می باشد .

۱- میکرو کنترلر AVR قابلیت دریافت و محاسبه ۱۵۰۰۰ نمونه آنالوگ به دیجیتال را دارد.میکروی AVR این کار را با محاسبه مقدار متوسط سیگنال ورودی انجام می دهد . تعداد این مقدار عدد بستگی به زمان پایان یک سیکل موج دارد .

۲- پس از اندازه گیری و دریافت نمونه جدید شروع به مقایسه آن با نمونه قبلی می کند .

۳- اگر نمونه بعدی بزرگتر از نمونه قبلی باشد , نتیجه می گیریم که شکل سیگنال ما در حال افزایش می باشد.پس بنابراین میکروی ما به مرحله ۴ یا مرحله بعدی وارد می شود.اگر نمونه بعدی کوچکتر از نمونه قبلی باشد , سپس میکروی ما مرحله ۲ را انجام می دهد .

۴- نمونه بعدی را اندازه گیری کرده و سپس آن را با مقدار متوسط سیگنالی که در مرحله یک گرفت مقایسه می کند .

۵- میکرو چک می کند که آیا این سیگنال دریافتی بزرگتر از مقدار متوسط می باشد یا نه؟

۶- اگر این مقدار بزرگتر باشد پس اول شکل موج برای میکرو پیدا شده است.میکرو شروع به ریختن ۱۰۰ نمونه بعدی داخل RAM می کند.برای این مقدار ۱۰۰ نمونه به کار گرفته شد چون برای نشان دادن شکل موج روی LCD از ۱۰۰ پیکسل استفاده شده است.

۷- میکرو این ۱۰۰ نمونه را داخل LCD پرینت می گیرد.

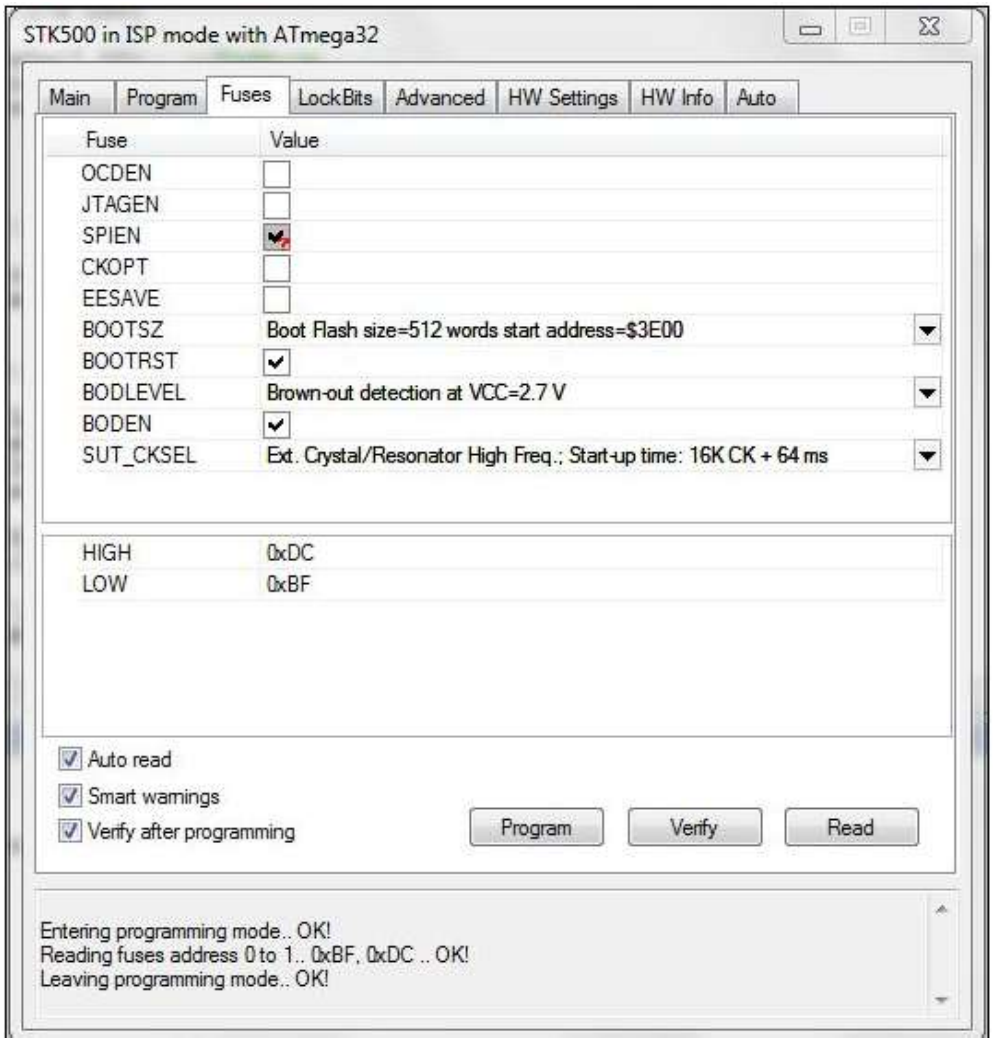
۸- میکرو دوباره از مرحله اول شروع به کار می کند.

چگونه فرکانس ورودی اندازه گیری و سپس بر روی LCD نمایان می شود؟

این محاسبه از طریق شمارش زمان شروع یک موج (یعنی مرحله ۴ صفحه قبل) تا زمان شروع سیکل بعدی انجام می پذیرد. سپس این شمارش در برنامه موجود در میکرو از طریق فرمول محاسبه شده و در LCD ظاهر می شود.

نحوه پروگرام کردن و تنظیمات **fusebit**

برای این کار در محیط نرم افزار باید تمامی تنظیمات را به صورت شکل زیر انجام دهیم .



شکل ۱۱-۳ تنظیمات فیوزبیت

فیوز بیت **JTAGEN**

را باید غیر فعال کرد.زیرا اگر این فیوز بیت فعال بماند باعث می شود که هنگام اجرای صفحه نخست LCD , آن را نشان داده و سپس سیستم ریست شده و دوباره به صفحه اول باز گشته و تا ابد روی آن صفحه می ماند و باعث می شود که صفحه مربوط به نمایش شکل موج در اسیلوسکوپ هیچ گاه نمایان نشود .

فیوز بیت **SPIEN**

این فیوز بیت برای فعال کردن قابلیت برنامه ریزی از طریق رابط SPI قابل استفاده و در حالت پیش فرض برنامه ریزی شده .

فیوزبیت **BOOTRST**

این فیوز بیت برای انتخاب بردار RESET استفاده می شود و در حالت پیش فرض برنامه ریزی نشده .

فیوزبیت **BODEN**

این فیوزبیت برای فعال کردن واحد BROWN-OUT استفاده می شود. در قسمت SUT _ CKSEL مقدار کریستال را انتخاب و روی گزینه Ext.crystal Freq 16k+64ms می گذاریم . بعد از تمامی این کارها گزینه پروگرام را زده و در این مرحله برنامه و همچنین تمامی فیوزبیت ها بر روی آی سی پروگرام خواهد شد .

سورس قسمت **welcome screen** به زبان C

```
/* As graphic LCD used the DEM128064A model (128x64 pixels).

*/
```

```
#include <avr/io.h>
#include <avr/pgmspace.h>

#define lcdrs     0                 // LCD's RS pin is connected to Pin0 of AVR
#define lcdrw     1                 // LCD's r/w pin is connected to Pin1 of AVR
#define lcde      2       // LCD's e pin is connected to Pin2 of AVR
#define lcdcs1    5       // LCD's CS1 pin is connected to Pin5 of AVR
#define lcdcs2    4       // LCD's CS2 pin is connected to Pin4 of AVR
#define lcdrst    3       // LCD's RST pin is connected to Pin3 of AVR

#define ctrl_port                 PORTB
#define ctrl_port_ddr     DDRB
#define ctrl_port_pins    PINB

#define data_port                 PORTD
#define data_port_ddr     DDRD
#define data_port_pins    PIND


void glcdInit (void);
void createWelcomeScreen (void);
void createRaster (void);
void delayTime (unsigned long counter);
void eStrobe (void);
void enable_cs1 (void);
void enable_cs2 (void);
void sendDataOnLCD (unsigned char data);
unsigned char readDataFromLCD (void);
void gLCDgotoXY (unsigned char lineData, unsigned char columnData);
void createWave (void);
```

```
void restoreRaster (void);
void fillDataLcdBuffer (unsigned char address, unsigned char data);
void GLCD_WriteChar(char charToWrite);

unsigned char column = 0;
unsigned char line = 0;
unsigned int  lcdAddress = 0;
unsigned int  flashAddress = 0;
static unsigned char dataLcdBuffer[100];
static unsigned char dataLcdBackupBuffer[200];
unsigned int  backupLcdAddress = 0;

const char LcdRaster[] __attribute__ ((progmem)) = {
255,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1
,1,1,1,3,1,1,1,1,1,1,1,1,1,3,1,1
,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1
,1,171,1,1,1,1,1,1,1,1,1,3,1,1,1,1
,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,3
,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1
,1,1,1,1,255,0,0,240,248,140,140,248,240,0,0,252
,252,0,0,252,252,0,0,4,252,252,68,196,252,56,0,0
,255,8,0,0,0,0,0,0,0,8,0,0,0,0,0,0
,0,0,0,8,0,0,0,0,0,0,0,0,0,8,0,0
,0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0
,8,170,8,0,0,0,0,0,0,0,8,0,0,0,0
,0,0,0,0,0,8,0,0,0,0,0,0,0,0,8
,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0,0
,0,0,0,8,255,0,0,199,39,32,192,7,7,0,0,1
,3,6,6,3,1,0,0,4,7,7,0,0,7,7,0,0
,255,32,0,0,0,0,0,0,0,32,0,0,0,0,0,0
```

,0,0,0,32,0,0,0,0,0,0,0,0,0,32,0,0

,0,0,0,0,0,0,0,32,0,0,0,0,0,0,0,0

,32,170,32,0,0,0,0,0,0,0,0,32,0,0,0,0

,0,0,0,0,0,32,0,0,0,0,0,0,0,0,0,32

,0,0,0,0,0,0,0,0,0,32,0,0,0,0,0,0

,0,0,0,32,255,0,0,25,34,34,28,0,30,33,33,33

,0,30,33,33,30,0,255,33,33,30,0,30,41,41,46,0

,255,128,0,128,0,128,0,128,0,192,0,128,0,128,0,128

,0,128,0,192,0,128,0,128,0,128,0,128,0,192,0,128

,0,128,0,128,0,128,0,192,0,128,0,128,0,128,0,128

,0,170,0,128,0,128,0,128,0,128,0,192,0,128,0,128

,0,128,0,128,0,192,0,128,0,128,0,128,0,128,0,192

,0,128,0,128,0,128,0,128,0,192,0,128,0,128,0,128

,0,128,0,128,255,0,40,32,40,32,40,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,255,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0

,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0

,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0

,0,170,0,0,0,0,0,0,0,0,0,1,0,0,0,0

,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1

,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0

,0,0,0,0,255,0,16,8,16,32,16,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,255,2,0,0,0,0,0,0,0,2,0,0,0,0,0,0

,0,0,0,2,0,0,0,0,0,0,0,0,0,2,0,0

,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0

,2,170,2,0,0,0,0,0,0,0,0,2,0,0,0,0

,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,2

,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0

,0,0,0,2,255,0,0,0,0,0,0,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,255,8,0,0,0,0,0,0,0,8,0,0,0,0,0,0

,0,0,0,8,0,0,0,0,0,0,0,0,0,8,0,0

,0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0

,8,170,8,0,0,0,0,0,0,0,0,8,0,0,0,0

,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,8

,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0,0

,0,0,0,8,255,0,0,124,20,4,0,124,20,104,0,124

,84,68,0,56,68,68,120,0,0,0,0,0,0,0,0,0

,255,128,128,128,128,128,128,128,128,128,160,128,128,128,128,128,128,128

,128,128,128,160,128,128,128,128,128,128,128,128,128,128,160,128,128,128

,128,128,128,128,128,128,128,128,160,128,128,128,128,128,128,128,128,128

,128,170,128,128,128,128,128,128,128,128,128,128,160,128,128,128,128,128

,128,128,128,128,128,160,128,128,128,128,128,128,128,128,128,128,160,128

,128,128,128,128,128,128,128,128,128,128,160,128,128,128,128,128,128,128

,128,128,128,128,255,0,0,0,0,0,0,0,0,0,0,0

,0,0,0,0,0,0,0,124,16,124,0,72,104,88,0,0

};

const char welcomeScreen[] __attribute__ ((progmem)) = {

255,1,1,1,1,1,1,1,129,129,65,33,33,17,17,17

, 17,17,17,17,17,33,33,65,65,129,129,1,1,1,1,1

, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

, 1,1,129,193,97,33,161,241,241,241,241,49,1,1,1,17

, 25,9,13,5,189,253,113,1,1,1,145,241,113,17,145,193

, 225,33,49,145,241,241,49,17,145,241,113,1,1,1,1,255

, 255,0,16,0,16,24,4,18,1,16,0,16,0,16,0,16

, 0,16,0,16,0,16,0,16,0,16,0,17,2,20,4,24

, 16,16,32,80,128,144,0,16,0,16,0,16,0,16,0,16

, 0,16,0,16,0,16,0,16,0,16,128,80,32,16,0,16

, 0,0,0,0,0,0,0,0,0,0,192,192,128,128,192

, 96,48,56,12,14,11,137,248,255,255,195,64,0,0,0,0

, 0,0,128,248,127,55,24,12,6,3,1,0,128,128,192,192

, 240,188,31,7,3,62,254,255,193,64,64,0,0,0,0,255

, 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,128,128,0,0,0,1,130,4,4,8,8,8,16,16,16

, 144,16,16,16,16,8,8,4,2,1,128,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,128,128,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255

, 255,0,0,0,0,0,0,240,248,60,14,7,14,252,240,0

, 0,0,112,248,156,140,6,7,3,2,0,0,224,248,124,30

, 15,7,3,1,0,224,126,15,1,0,0,0,0,0,224,254

, 31,7,0,0,0,0,0,0,224,254,31,7,0,0,0,0

, 0,0,240,248,60,14,7,14,252,240,0,0,0,112,248,156

, 140,6,7,3,2,0,0,224,248,124,30,15,7,3,1,0

, 240,248,60,14,7,14,252,240,0,0,0,244,254,254,34,18

, 14,0,0,0,240,254,63,19,19,19,1,0,0,0,0,255

, 255,0,0,0,0,0,0,1,3,2,6,6,3,3,1,0

, 0,0,8,8,8,8,9,6,0,0,0,0,1,3,2,2

, 2,0,0,0,0,15,0,0,0,0,0,0,0,0,3,2

, 6,6,6,2,2,2,2,0,3,2,6,6,6,2,2,2

, 2,0,1,3,2,6,6,3,3,1,0,0,0,8,8,8

, 8,9,6,0,0,0,0,1,3,2,2,2,0,0,0,0

, 1,3,2,6,6,3,3,1,0,0,0,7,3,0,0,0

, 0,0,0,0,1,3,2,2,3,1,1,0,0,0,0,255

, 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,140

, 66,34,28,0,252,2,2,254,0,8,4,254,0,0,140,66

, 34,28,0,0,0,0,0,0,254,16,16,224,0,60,64,64

, 188,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255

, 255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

, 0,0,240,16,16,16,48,0,224,16,16,16,224,0,240,16

, 16,16,224,0,112,144,16,48,0,240,0,0,0,240,0,227

, 18,18,18,224,3,242,18,17,32,194,2,3,2,0,115,18

, 18,242,16,16,112,0,224,16,19,18,226,1,240,4,4,4

, 243,0,224,16,16,16,16,224,0,224,16,16,16,16,224,0

, 240,32,192,0,0,0,240,0,0,16,240,16,0,0,0,0

, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255

, 255,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128

, 128,128,159,129,129,128,128,128,159,129,129,129,159,128,159,129

, 131,133,152,128,152,144,145,158,128,159,129,129,129,159,128,159

, 129,129,129,159,128,159,144,144,136,135,128,129,129,129,128,128

, 128,159,128,128,128,128,159,129,129,129,159,128,159,129,129,129

, 159,128,143,144,144,144,144,143,128,143,144,144,144,144,143,128

, 159,128,128,129,134,136,159,128,128,144,159,144,128,128,128,128

, 128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,255

};

سورس برنامه اسیلوسکوپ به زبان C

```
#include <avr/io.h>                    // include I/O definitions (port names, pin names, etc)
#include <avr/interrupt.h>
#include "glcd.c"
```

```
/* define CPU frequency in Mhz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#define msUp            1
#define msDwn           4
#define YposUp          0
#define YposDwn             3
#define freeze          2

#define AC              0
#define DC              1
#define SQUARE          2
#define TRUE            0
#define FALSE           1

unsigned int  dataCounter = 0;
unsigned int  timeDiv = 0;
unsigned char trigger = 0;
unsigned char findZero = 0;
unsigned char upLimit = 0;
unsigned char lowLimit = 255;
unsigned char prevADCvalue = 0;
unsigned char voltageType = AC;
unsigned char complete = TRUE;
unsigned int  voltage;
unsigned char frequency;
unsigned int  ADCvalue;
```

```
unsigned char hex2asciiBuffer[4];


signed char Ypos   = 0;
signed char Ypos2   = 0;
signed char position  = 0;

void ADC_init (void);
void hex2Ascii(unsigned int data, unsigned char Buffer[4]);



int main (void)
{

        unsigned int  i,k;
        unsigned long endOfPeriod=0;
        unsigned char freqComplete=0;
        unsigned int  maxVoltage=0;

        DDRC = 0b00000000;
        PORTC = 0b11111111;

        DDRA = 0b00000000;

        glcdInit();
        ADC_init();
        createWelcomeScreen();
        _delay_ms(4000);
        createRaster();
        createWave();
```

```
for(;;)
{

        if (~PINC & (1<<msUp) && (timeDiv <= 120))
        timeDiv += 1;

        if (~PINC & (1<<msDwn) && (timeDiv >= 1))
                timeDiv -= 1;



        if (~PINC & (1<<YposUp) && (Ypos2 <= 60))
        Ypos2++;

        if (~PINC & (1<<YposDwn) && (Ypos2 >= -60))
                Ypos2--;

        if (~PINC & (1<<freeze))
                while (~PINC & (1<<freeze)); // It freezes the display to watch the wave.



//--------v
// Read 100 samples from analog input.
// From these 100 samples we can find the midle of the signal to be used as trigger...
// ... the Volts peak-to-peak and the frequency of the measured signal.

        findZero = 0;
        upLimit = 0;
        lowLimit = 255;
        endOfPeriod = 0;
```

```
freqComplete = 0;
complete = FALSE;

for (i=0; i<15000; i++)
{
    ADCSRA |= (1 << ADSC); // Enable ADC
    loop_until_bit_is_set(ADCSRA, ADIF); // wait until conversion complete.
    ADCvalue = ADCH;

    //-------------------------------------------------------v
    //  Find the end of the first period that is appeared on LCD.
    //  The begining of the period is always the first pixel on the left of LCD.

//Find the start of the period of the measured waveform.
    if((ADCvalue > trigger) && (prevADCvalue < ADCvalue) && (freqComplete ==
0))
        freqComplete = 1;

    //If you have found the start of the period, find the rise of the waveform.
    if((ADCvalue < trigger) && (prevADCvalue < ADCvalue) && (freqComplete ==
1))
        freqComplete = 2;

    //The next step is to find the start of the next period...
    if((ADCvalue > trigger) && (prevADCvalue < ADCvalue) && (freqComplete ==
2))
    {
        freqComplete = 3; //Wow! we found the end of the first period.
        endOfPeriod = ((23000)/(i/2)); //Calculate the frequency that will be
displayed on LCD.
    }
    //-------------------------------------------------------^
```

```
prevADCvalue = ADCvalue; // Get a backup of the current ADC value.

for(k=timeDiv;k>0;k--)   // Make a delay to see on LCD low-frequency waveforms. Normally, the minimum waveform that can be entirely displayed on LCD is 310 Hz.
{
        ADCSRA |= (1 << ADSC);   // Enable ADC
        loop_until_bit_is_set(ADCSRA, ADIF); // wait until conversion complete.
        ADCvalue = ADCH;
}

if (upLimit < ADCvalue)  // Find the higher voltage level of the input waveform.
        upLimit = ADCvalue;

if (lowLimit > ADCvalue) // Find the lower voltage level of the input waveform.
        lowLimit = ADCvalue;

maxVoltage = (upLimit * 2);  // Maximum voltage on ADC pin is calculated from upLimit register (255*2=510 = 5.10V).

if (ADCvalue > 0)
{
        voltage = ((upLimit-lowLimit)*2); //Get the Vpp and store it to "voltage" (Volts Peak-to-peak of inputed waveform).
        ADCvalue += 5;
        ADCvalue /= 5;
        ADCvalue += 2;
}else
        ADCvalue = 2;
```

```
                position = ADCvalue + Ypos2 +5;

                if ((position <= 63) && (position >= 0) && (i<100))  // Adjust Up-Down the wave on LCD.

                        fillDataLcdBuffer(i,position);

                else

                if(i<100)

                        fillDataLcdBuffer(i,0);


                if((i>100)&&(freqComplete==3))   //If i>100 and freqComplete=3 that means that our waveform is outside of LCD displaying area.

                        break;  //Do not wait until i=15000. Terminate this loop ("for (i=0; i<15000; i++)")


        }


        if(upLimit != lowLimit)

                trigger = (((upLimit - lowLimit)/2)+ lowLimit); // Find the middle of the wave to be used as trigger.

        else

                trigger = upLimit;


//--------^


//--------- Print Volts peak-to-peak and frequency on display ----------


        restoreRaster();

        createWave();


        line=3;                 //Show the DC voltage

        column=109;

        gLCDgotoXY(line,column);

        hex2Ascii(maxVoltage,hex2asciiBuffer);
```

```
                GLCD_WriteChar(hex2asciiBuffer[2]);
                sendDataOnLCD(0b01000000); //Print one dot character on LCD (only 1 column
length).
                sendDataOnLCD(0);
                GLCD_WriteChar(hex2asciiBuffer[1]);
                GLCD_WriteChar(hex2asciiBuffer[0]);
                sendDataOnLCD(0x00);
                GLCD_WriteChar('V');

                line=4;                 //Show the Vpp (Volts peak-to-peak).
                column=109;
                gLCDgotoXY(line,column);
                hex2Ascii(voltage,hex2asciiBuffer);
                GLCD_WriteChar(hex2asciiBuffer[2]);
                sendDataOnLCD(0b01000000); //Print one dot character on LCD (only 1 column
length).
                sendDataOnLCD(0);
                GLCD_WriteChar(hex2asciiBuffer[1]);
                GLCD_WriteChar(hex2asciiBuffer[0]);
                sendDataOnLCD(0x00);
                GLCD_WriteChar('V');

                line=6;                 // Go to 6th line on LCD.
                column=122;
                gLCDgotoXY(line,column);

                if(timeDiv == 0)        // If 'timeDiv' = 0 then the frequency in to the 'endOfPeriod'
variable is real.
                        GLCD_WriteChar(0x5c); // So, print on LCD the "Play" character. '0x5c' is the
'\' character ("Play" symbol).
                else
```

```
                GLCD_WriteChar(']');  // If 'timeDiv' is not zero that means we have shrink the waveform. So, print the "Pause" symbol on LCD.


                if(timeDiv == 0)              // If 'timeDiv' = 0 then the frequency in to the 'endOfPeriod' variable is real.
                {
                        line=7;             // Go to 7th line and print on LCD the waveform's frequency
                        column=102;
                        gLCDgotoXY(line,column);
                        if(endOfPeriod <10000) // The maximum frequency that can be displayed on LCD is 9999 Hz.
                        {
                                gLCDgotoXY(line,column);
                                hex2Ascii(endOfPeriod,hex2asciiBuffer);
                                GLCD_WriteChar(hex2asciiBuffer[3]);
                                GLCD_WriteChar(hex2asciiBuffer[2]);
                                GLCD_WriteChar(hex2asciiBuffer[1]);
                                GLCD_WriteChar(hex2asciiBuffer[0]);
                        }

                }               // if 'timeDiv' > 0 do not update the frequency. Keep the previous frequency value instead.

//-------------------------------------------------------


                dataCounter = 0;
                complete = FALSE;
                freqComplete = 0;

                do  //Wait in this loop until you find again the start of the measured waveform.
                {
                        prevADCvalue = ADCvalue;
```

```
                ADCSRA |= (1 << ADSC);    // Enable ADC
                loop_until_bit_is_set(ADCSRA, ADIF);
                ADCvalue = ADCH;

                //Find the start of the period of the measured waveform.
                if((ADCvalue > trigger) && (prevADCvalue < ADCvalue) && (freqComplete ==
0))
                        freqComplete = 1;

                //If you have found the start of the period, find the rise of the waveform.
                if((ADCvalue < trigger) && (prevADCvalue < ADCvalue) && (freqComplete ==
1))
                        freqComplete = 2;

                //The next step is to find the start of the next period...
                if((ADCvalue > trigger) && (prevADCvalue < ADCvalue) && (freqComplete ==
2))
                {
                        freqComplete = 3;
                        complete = TRUE;
                }
                if(dataCounter > 3000)
                        complete = TRUE;
                dataCounter++;

        }while(complete == FALSE);

    }
}
```

```
//=============================================================================
===

//  Initiallizing the Analog to Digital Converter (ADC).

//=============================================================================
===


void ADC_init (void)

{

        ADMUX = 0b01100000;    // PA0 -> ADC0, ADLAR=1 (8-bit)

        ADCSRA |= ((1<<ADEN) | (1<<ADSC) | (1<<ADPS1)); // ADC prescaler at 4

}




//=============================================================================
===

//  Function that converts an integer into 4 ASCII-character bytes.

//  For examplee if 'data' contains the number "6543" then

//    Buffer[3] = '6'  Buffer[2]='5'  Buffer[1]='4'   Buffer[0]='3'.

//  I know, I could have used the 'itoa' function to do the conversion from integer into ASCII

//    but my function is easier and more convenient to be handled.

//=============================================================================
===

void hex2Ascii(unsigned int data, unsigned char Buffer[4])

{

        Buffer[3] = ((data/1000)+0x30);

        data %= 1000;

        Buffer[2]  = ((data/100)+0x30);

        data %= 100;

        Buffer[1]  = ((data/10)+0x30);

        Buffer[0]  = ((data%10)+0x30);

}
```

فصل چهارم

لیست قطعات موجود در اسیلوسکوپ :

میکرو کنترلر **Atmega32**

آی سی **LM358N**

LCDگرافیکی **128*64**

کریستال **16** مگا هرتز

رگولاتور **7805**

دیود **1n4001**

دیود زنر **8.2** ولتی

مقاومت های **220 ohm , 390 k, 1 M ohm , 56 ohm**

ولوم های **10K , 22K**

خازن های **27PF, 470nf , 100nf, 22PF , 22uf , 100uf**

انواع کلید های کشوئی و الاکلنگی

لیست قطعات موجود درسیگنال ژنراتور :

میکرو کنترلر **Atmega16**

LCD کاراکتری **16*2**

Keyboard عددی(**numerical keyboard**)

آی سی **DAC0800**

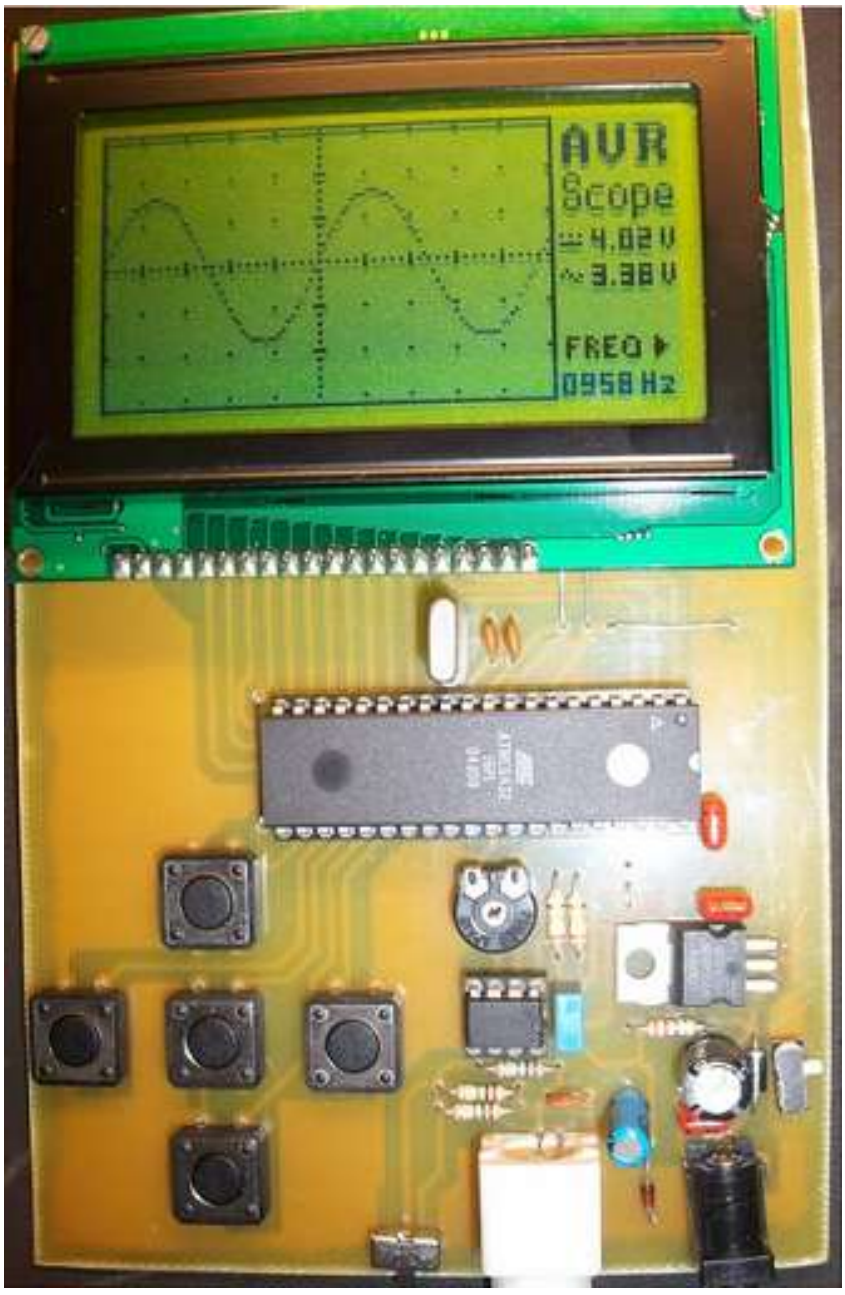
پل دیود

رگولاتور **7809**و **7909**

خازن های **100nf , 1000uf**

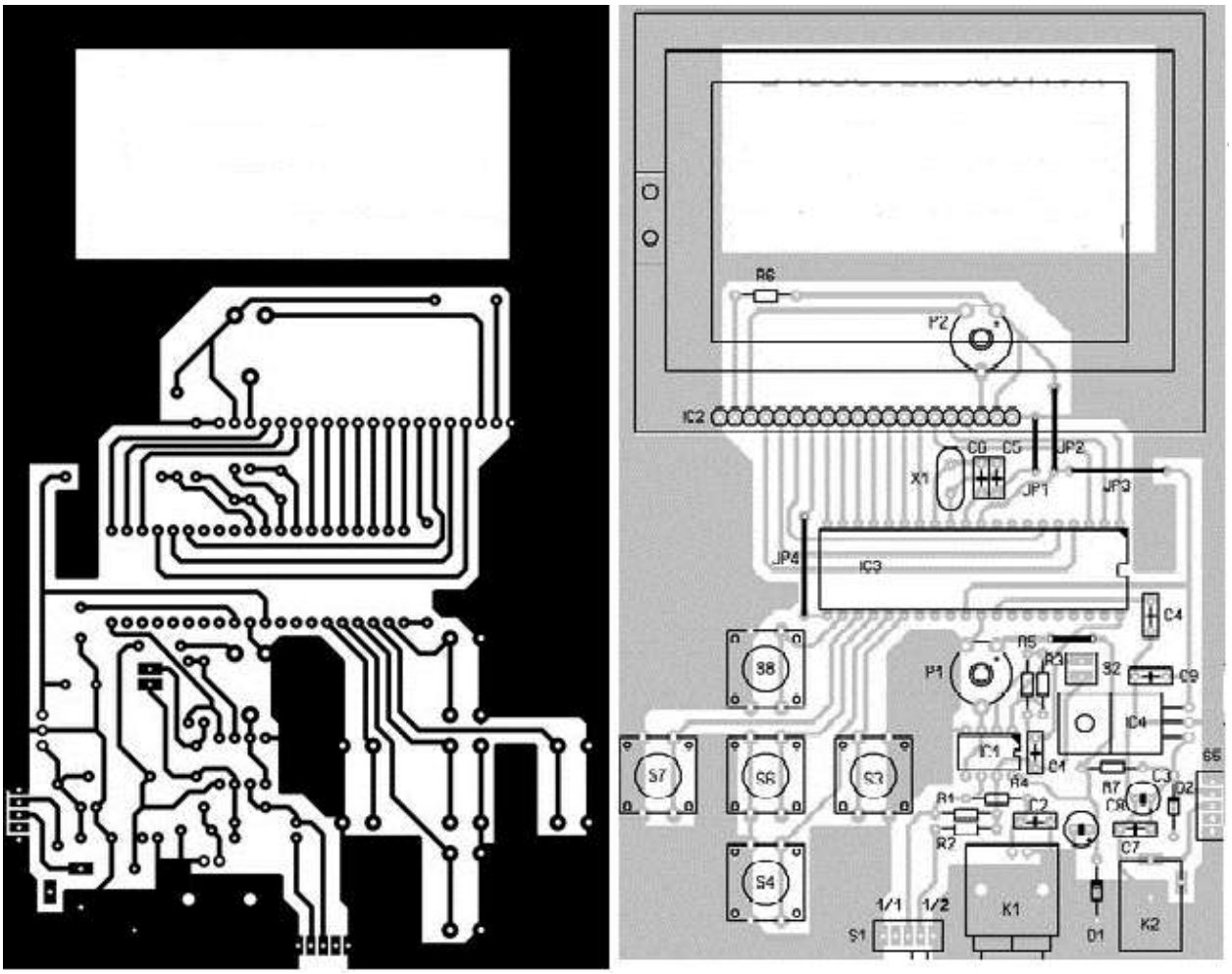
ترانس **9** ولتی

شکل ظاهری مدار اسیلوسکوپ



شکل ۱-۴ شمای اسیلوسکوپ

بورد **PCB** مدار اسیلوسکوپ



نمای پشت PCB

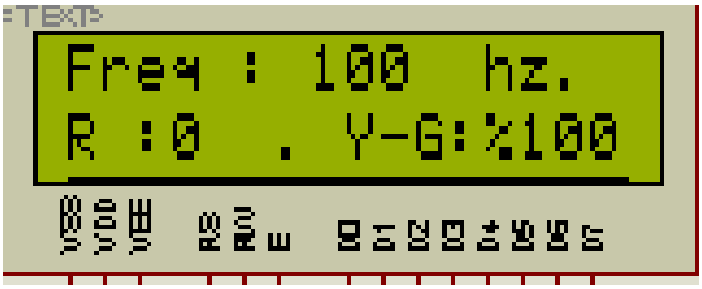
نمای روی PCB

شکل ۲-۴ بورد PCB اسیلوسکوپ

نحوه کار با دستگاه سیگنال ژنراتور

در آغاز پس از استارت مدار نمایشگر آن در خط اول مقدار فرکانس بر حسب هرتز را نمایش میدهد.

.

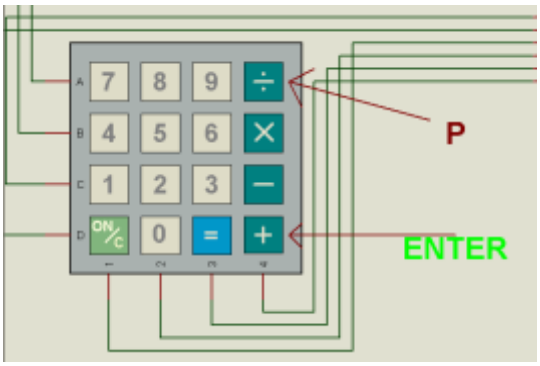


شکل ۳-۴ نمایشگر LCD

منوی تنظیم فرکانس

با فشار دادن دکمه P وارد منوی فرکانس شده و مقدار فرکانس مورد نظر را وارد و سپس گزینه Enter را می زنیم .

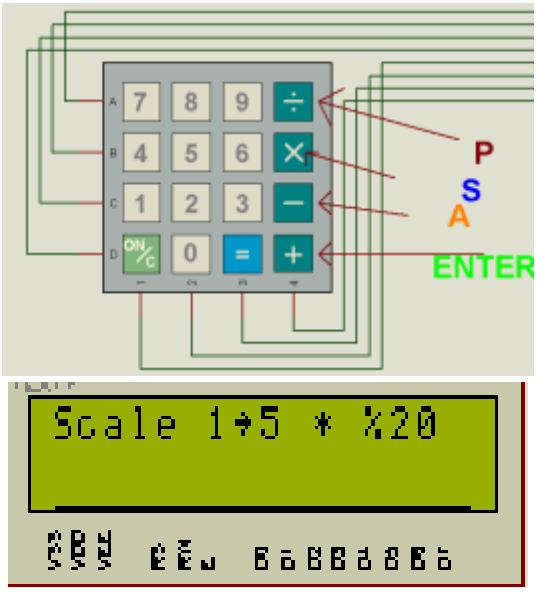
به صورت شکل زیر:



شکل ۴-۴ تنظیم فرکانس

منوی تنظیم دامنه

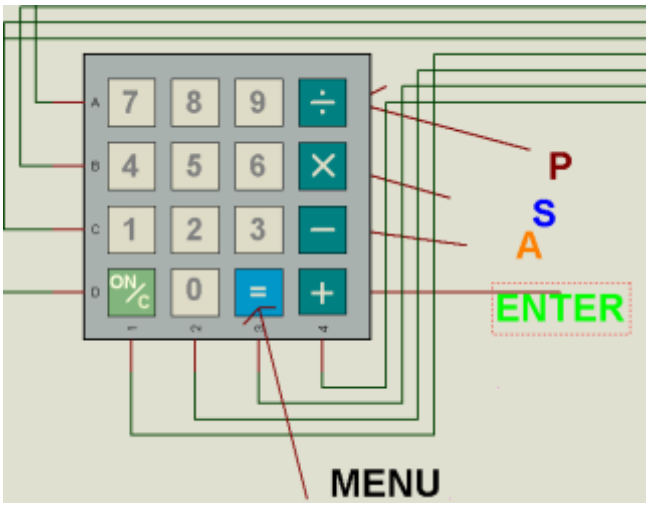
با زدن این گزینه به صورت شکل زیر می توانیم دامنه سیگنال خروجی را از ۱ تا ۵ ولت بسته به ولتاژ مرجع ورودی تغییر دهیم .



شکل ۵-۴ تنظیم دامنه

دکمه **Delete** یا **Clear**

این برنامه دارای یک دکمه تصحیح خطای نوشتاری کاربر است که می توان با آن اشتباهات تایپی خود را درست کرد . دکمه MENU این عمل را بر عهده دارد.به صورت زیر :



شکل۶-۴ پاک کردن خطا

پروژه های دیگری که به عنوان **proposal** پیشنهاد داده شده بود عبارت اند از :

*پارکینگ هوشمند با کارت خوان RFID

* کنترل وسایل خانگی از طریق تلفن

*مانیتورینگ دما توسط کامپیوتر به صورت بی سیم

* کنترل وسایل منزل از طریق خط تلفن

* شبیه سازی سیستم نوبت دهی بانکی با قابلیت پخش صوت

پروژه های ساخته شده

*روبات مسیریاب

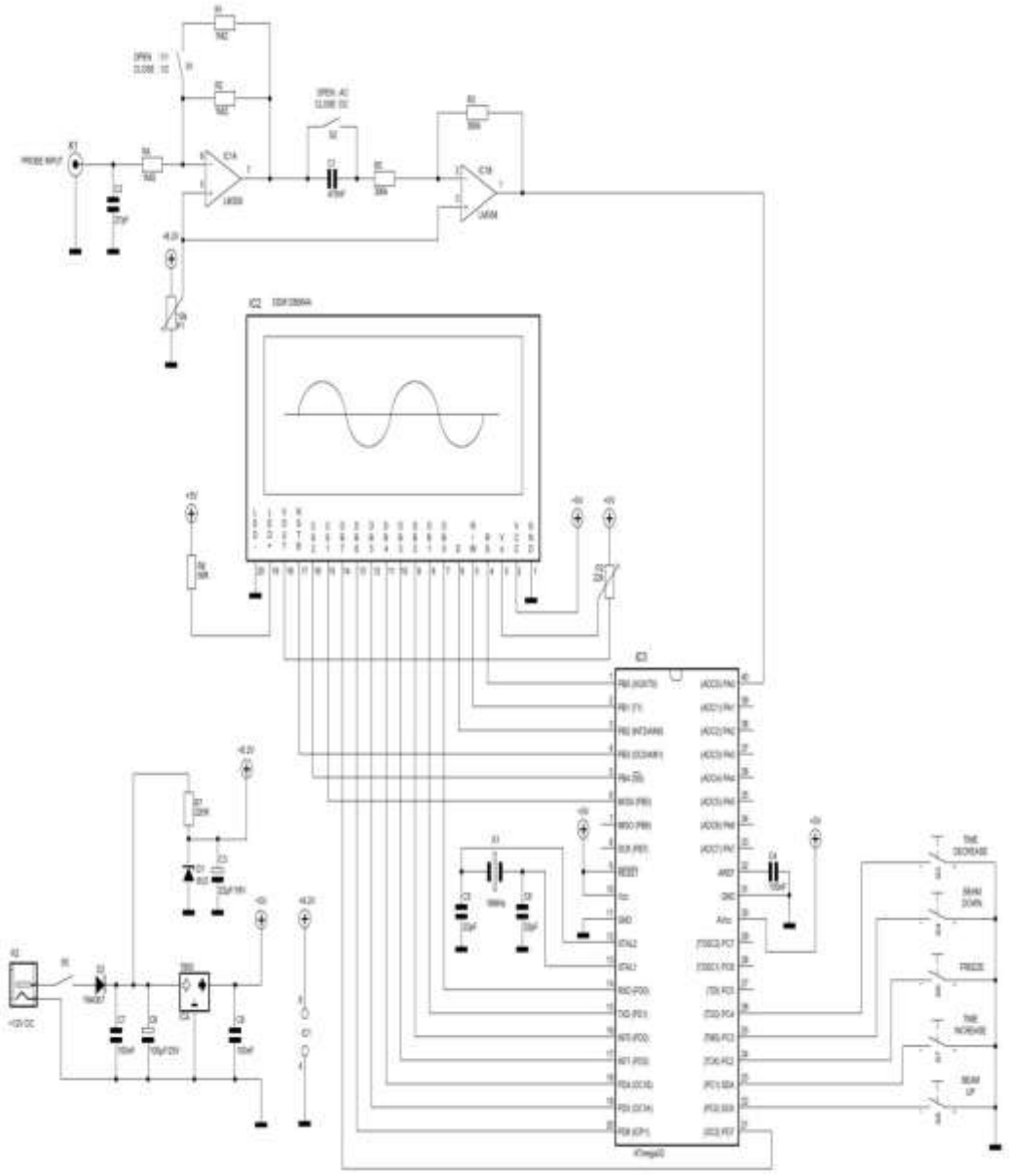
*منبع تغذیه صفر تا ۳۰ ولت دیجیتال هوشمند

*مصافت سنج اولتراسونیک تا ۴ متر با خطای ۵ ٪

فصل دوم

شماتیک و نقشه فنی مدار اسیلوسکوپ

در این قسمت به توضیح مدار عملی دستگاه اسیلوسکوپ و اجزای مختلف آن به طور کامل می پردازیم.



شکل ۱-۲ نقشه فنی مدار اسیلوسکوپ

فهرست منابع و ماخذ

کتاب مرجع کامل میکرو کنترلرهای AVR تالیف پرتوی فر و مظاهریان انتشارات نص تابستان ۸۶

کتاب میکرو کنترلرهای AVR تالیف علی کاهه انتشارات نص زمستان ۸۶

جزوه میکروکنترلرهای AVR استاد جعفری

جزوه آزمایشگاه میکروکامپیوتر استاد فتاحی

سایت الکترونیکی تکنو الکترونیک به آدرس www.techno-electro.com

سایت دیتا شیت به آدرس www.alldatasheet.com

و بسیاری از منابع اینترنتی دیگر.....