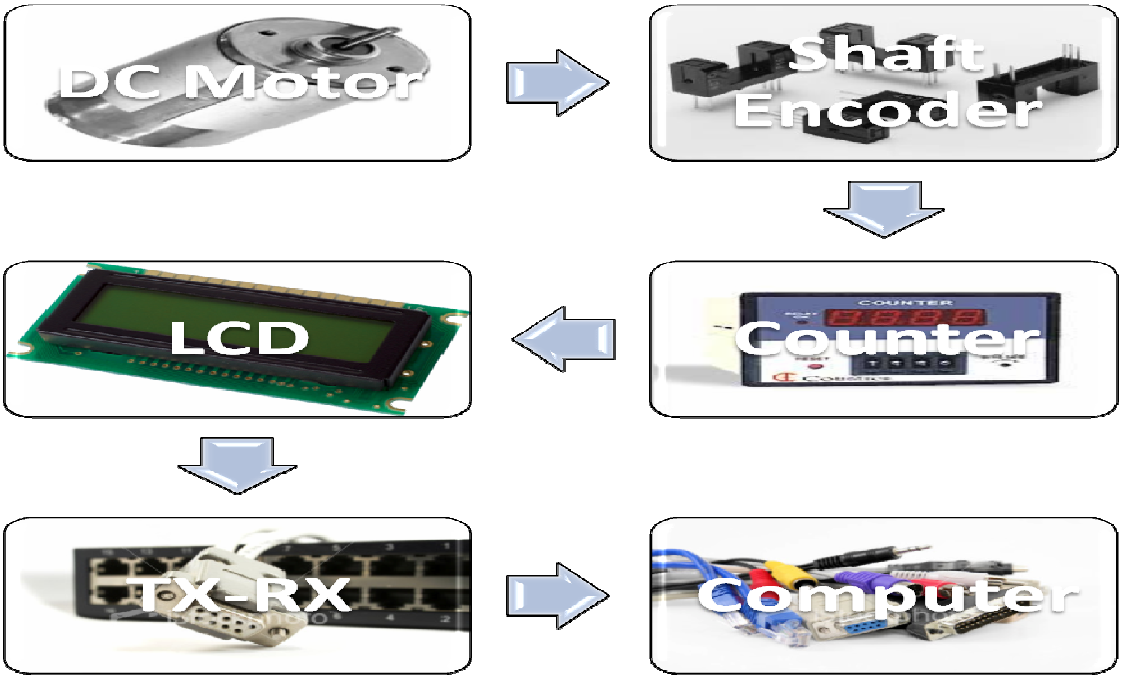
به نام خدا

کنترل دور موتور با استفاده از نرم افزار MATLAB

نگارش : ۱.۰

تهیه و تنظیم:
امیرعلي بلورچيان



فهرست

عنوان **صفحه**

مقدمه	۲
انواع کنترل گرها	۳
ATMEGA 16	۵
بلوک دیاگرام مدار	۹
موتور	۱۰
Shaft Encoder	۱۶
Counter	۲۱
نمایش بر روی LCD	۲۶
مدار واسط	۳۰
نحوه پروگرم کردن	۴۰
Computer	۴۲
سورس برنامه	۴۴
شماتیک مدار	۵۱

مقدمه:

امروزه در صنعت و در بسیاری از وسایل خانگی کنترل دور موتور مورد استفاده میگرد. از جمله می توان به کاربردهای کنترل گرهای دور موتور، به موارد زیر اشاره کرد:

۱) وسایل خانگی:

کنترل گر های دور موتور در وسایل شخصی خانگی، در کار برد های کوچک و بزرگ مورد استفاده قرار میگیرند. به عنوان مثال، پنکه های دیواری یا پنکه تهویه حمام که توسط کلیدی کنترل می شوند.

۲) در وسایل اداری و درمانی:

در این دسته دستگاه های بسیاری را می توان مثال زد. مداد تراش های برقی در ادارات، دستگاه هایفکس، کامپیوتر ها یا دستگاه های کپی و.. . سیستم کاری این کنترل گر ها بسیار پیچیده بوده و حتی در مورد وسایل درمانی پیچیده تر نیز میشود. مثلاً کنترل دور موتور داخل هارد دیسک کامپیوتر را در نظر بگیرید.

۳) در کار برد های تجاری:

ساختمان های تجاری دارای سیستم تهویه بزرگتر و مجهز تری نسبت به موارد مشابه در منازل شخصی دارند. همچنین می توان در این دسته موتور ها برای آسانسور ها، پله های برقی و موارد مشابه را نام برد.

۴) کار برد های صنعتی:

بسیاری از صنایع وابسته به موتور ها و کنترل دور موتور آن ها می باشند. موتور های کوچک DC تا موتور ای بزرگ صنعتی، یا موتور های استفاده شده در خطوط مترو. همچنین در صنعت ممکن است یک کنترل گر عمل کنترل بیش از یک موتور را به طور همزمان بر عهده داشته باشد.

۵) در وسایل نقلیه:

تمام وسایل نقلیه از جمله، خودرو ها، هواپیما ها، دستگاه آلات کشاورزی، همه و همه ممکن است دارای موتور برای انجام کار های گوناگونی باشند.

۶) ابزار قدرت:

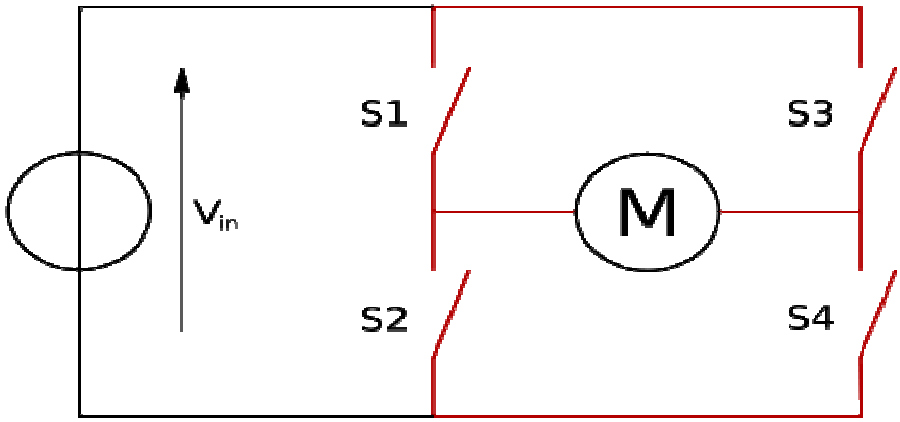
وسایل قدرتی همانند دریل ها، اره ها، چرخ سمباده ها که توسط کاربر خانگی استفاده می شوند. تمام وسایل قدرتی قایل حمل یا ثابت دارای معمولاً همراه با کنترل گر های سرعت این موتور ها نیز می باشند.

انواع کنترل گرها :

کنترل گر های موجود را می توان بر اساس موتوری که باید کنترل کنند، دسته بندی کرد.
SERVO موتور ها، مغناطیسی ثابت، سری، تحریک مجزا و جریان متناوب و مستقیم.
توضیح برخی از کنترل گرهای شناخته شده:

۱) H-Bridge:

موتور های Dc معمولاً توسط ترانزیستور ها با مداری مشهور به مدار H-Bridge کنترل میشوند.
این روش شامل حداقل ۴ سویچ مکانیکی یا الکترونیکی (ساخته شده از نیمه هادی ها) می باشد.



در مدار فوق که نمونه ای از مدار H-Bridge است، با بستن کلید های S2 و S3 ولتاژ V_{in} بطور معکوس بر روی موتور قرار میگرید که موجب گردش موتور در خلاف جهت اولیه (زمانی که کلید های S1 و S4 وصل بودند) می شود. این مدار توسط نیمه هادی با استفاده از دو عنصر با پلاریته های متفاوت ساخته می شوند. برای مثال: با ترانزیستور های BJT، PNP یا MOSFET، P کانال متصل به سطح ولتاژ بالا و ترانزیستور های BJT، NPN یا MOSFET، N کانال.

۲) کنترل گرهای Servo:

بسیاری از موتور های SERVO توسط روشی مشهور به PWM کنترل میشوند که در ادامه در این مورد بیشتر بحث خواهیم کرد.

۳) کنترل گر های STEP:

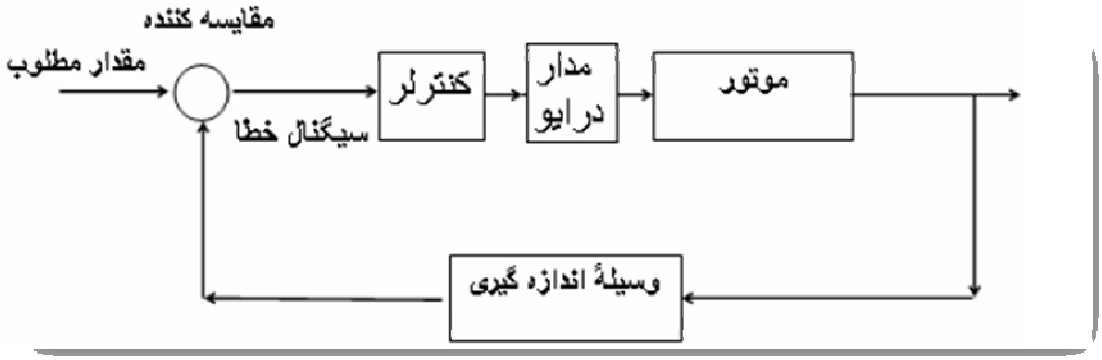
این موتور های توسط مدارات زمان بندی شده کنترل میشوند. این امر موجب میشود تا کنترل سرعت و چرخش موتور با دقت زیادی صورت گیرد. از این کنترل گر ها در چاپگر ها استفاده میشوند.

از میان روش های گفته شده، در این مقاله کنترل دور موتور با استفاده از PWM توضیح داده خواهد شد.

استفاده از PWM برای کنترل دور موتور:

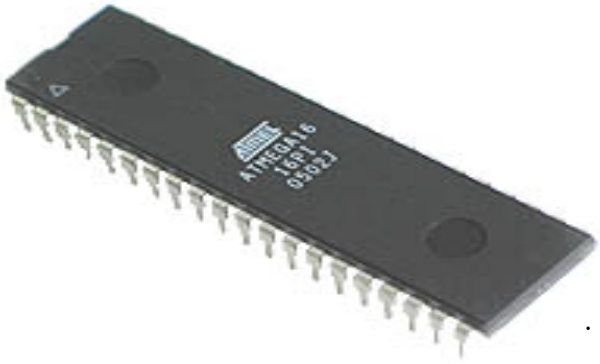
ایده کلی استفاده از PWM برای کنترل دور موتور DC بسیار ساده است. اما ابتدا توضیحی در مورد PWM و روش های گوناگون تولید آن خواهیم داد.

در حالت کلی برای بسیاری از موارد گفته شده و یا راه کار های مشابه می توان بلوک دیاگرام کنترلی زیر را در نظر گرفت:



در این مقاله از میکروپروسسورها به عنوان کنترلر و از IC هایی که در ادامه تشریح خواهد بزای مدار درایور استفاده خواهد شد. قبل از اینکه وارد بحث کنترل دور موتور شویم، باید دور موتور را در ولتاژ های معین شده ای در معین کرده باشیم. در این پروژه یک موتور DC ۱۲ ولت را بررسی و کنترل خواهیم کرد. برای بدست آوردن دور موتور در ولتاژ های مختلف یکی از روش های ساده استفاده از یک Shaft Encoder دیجیتال و شمردن تعداد پالس های این Shaft Encoder در مدت یک ثانیه یا یک دقیقه است. برای اینکه زمان و دقت بیشتری داشته باشیم، در مدت زمان یک ثانیه تعداد پالس های ورودی را خواهیم شمرد. از آنجایی که در ادامه پروژه نیز از میکروکنترلر AVR ATMEGA 16 استفاده خواهیم، پس در این قسمت از شمارنده داخلی همین IC برای شمارش تعداد پالس های Shaft Encoder استفاده خواهیم کرد.
پس تا کنون اشاره شد که باید با شمارش تعداد پالس های حاصل از Shaft Encoder، دور مورتور را در ولتاژ های معینی بدست آوریم. اصولاً به چنین سیستم دور سنج نیز گفته می شود.

خصوصیات ATMEGA 16 و ATMEGA 16L :



- از معماری AVR RISC استفاده می کند .
- کارایی بالا و توان مصرفی کم .
- دارای ۱۳۱ دستورالعمل با کارایی بالا که اکثرا تنها در یک کلاک سیکل اجرا می شوند .
- ۸*۳۲ رجیستر کاربردی .
- سرعتی تا 16 MIPS در فرکانس 16 MHz .
- حافظه برنامه و داده غیرفرار
- 16k بایت حافظه FLASH داخلی قابل برنامه ریزی .
- پایداری حافظه FLASH : قابلیت ۱۰۰۰۰ بار نوشتن و پاک کردن .
- ۱۰۲۴ بایت حافظه داخلی SRAM .
- ۵۱۲ بایت حافظه EEPROM داخلی قابل برنامه ریزی .
- پایداری حافظه EEPROM : قابلیت ۱۰۰۰۰۰ بار نوشتن و پاک کردن .
- قفل برنامه FLASH و EEPROM .

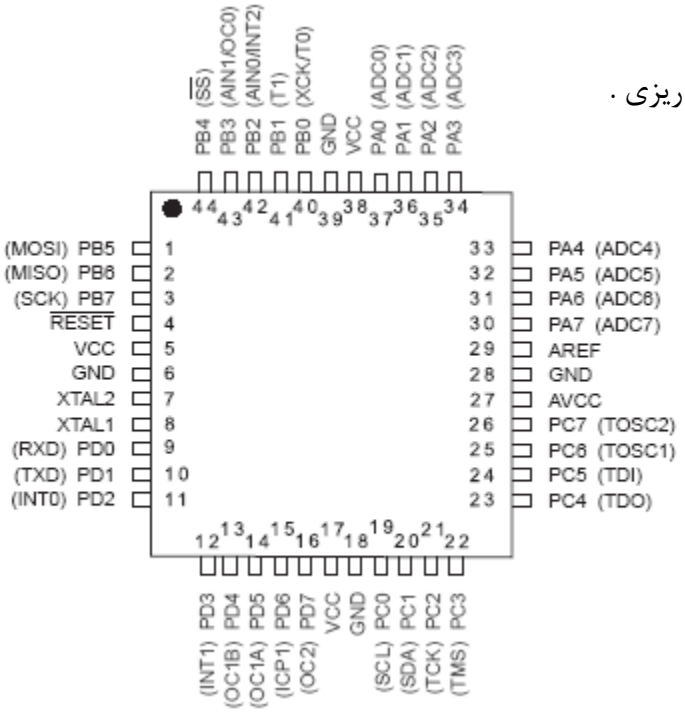
خصوصیات جانبی

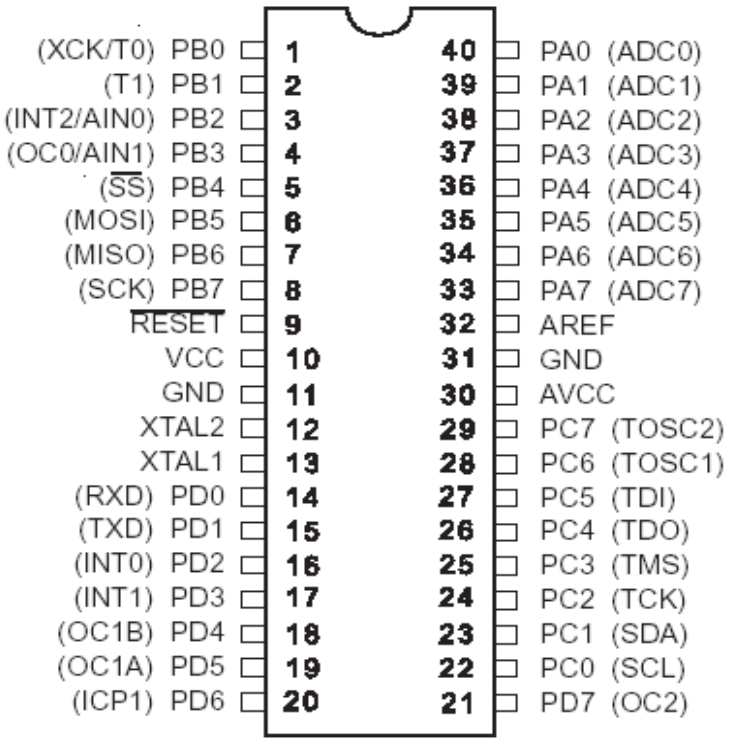
- دو تایمر/کانتر ۸ بیتی با Prescaler مجزا و مود Capture .
- یک تایمر/کانتر ۱۶ بیتی با Prescaler مجزا و دارای مودهای Capture و Compare .
- ۴ کانال PWM .

- ۸ کانال مبدل آنالوگ به دیجیتال ۱۰ بیتی .
- یک مقایسه کننده آنالوگ داخلی .
- WATCHDOG قابل برنامه ریزی با اسیلاتور داخلی .
- قابلیت ارتباط با پروتکل سریال دوسیمه (TWO WIRE) و I2C .
- قابلیت ارتباط سریال SPI (Serial Peripheral Interface) به صورت Master یا Slave .
- USART سریال قابل برنامه ریزی .

خصوصیات ویژه میکرو کنترلر

- ♦ دارای اسیلاتور RC داخلی کالیبره شده .
- ♦ منابع وقفه (Interrupt) داخلی و خارجی .
- ♦ توان مصرفی پایین و سرعت بالا توسط تکنولوژی CMOS .
- ♦ ولتاژ کاری ۲.۷ تا ۵.۵ ولت برای Atmega16L و ۴.۵ تا ۵.۵ وبت برای Atmega16 فرکانس های کاری 0 تا 8MHz برای Atmega16L و 0 تا 16MHz برای Atmega16
- ♦ ۳۲خط ورودی خروجی قابل برنامه ریزی .





همانطور که در شکل دیده می شود ATMEGA 16 دارای ۴ پورت D,C,B,A می باشد که علاوه بر اینکه به عنوان ورودی-خروجی مورد استفاده قرار می گیرند ، کاربردهای جانبی دیگری نیز دارند . در این بخش به شرح این پورت ها می پردازیم :

پورت A

پورت A یک I/O دو طرفه ۸ بیتی است . سه آدرس از مکان حافظه I/O اختصاص به PORTA دارد ؛ یک آدرس برای رجیستر داده PORTA ، دومی برای رجیستر جهت داده DDRA و سومی برای رجیستر ورودی PINA .
پورت A به عنوان ADC هم استفاده می شود . اگر تعدادی از پایه های پورت A خروجی تعریف شوند ، این نکته بسیار مهم است که در زمان نمونه برداری از سیگنال آنالوگ توسط ADC ، سوئیچ نشوند . این کار ممکن است عملیات تبدیل ADC را نامعتبر کند .

پورت B

پورت B یک I/O دو طرفه ۸ بیتی است . سه آدرس از مکان حافظه I/O اختصاص به پورت B دارد ؛ یک آدرس برای رجیستر داده PORTB ، دومی برای رجیستر جهت داده DDRB و سومی برای رجیستر ورودی PINB .

دیگر کاربرد های پورت B

- PORTB.7 (SCK) : کلاک خروجی MASTER و کلاک ورودی SLAVE برای ارتباط SPI است . زمانی که SPI به عنوان SLAVE شکل دهی می شود این پایه با توجه به تنظیم DDRB.7 ورودی و در حالت MASTER خروجی تعریف می شود .
- PORTB.6 (MISO) : ورودی داده MASTER و خروجی داده SLAVE که برای ارتباط SPI استفاده می شود .
- PORTB.5 (MOSI) : ورودی داده SLAVE و خروجی داده MASTER که برای ارتباط SPI استفاده می شود .
- PORTB.4 (SS) : زمانی که SPI به عنوان SLAVE شکل دهی می شود PORTB.4 با توجه به DDRB.4 ورودی تعریف می شود و در SLAVE با LOW شدن این پایه SPI فعال می شود .

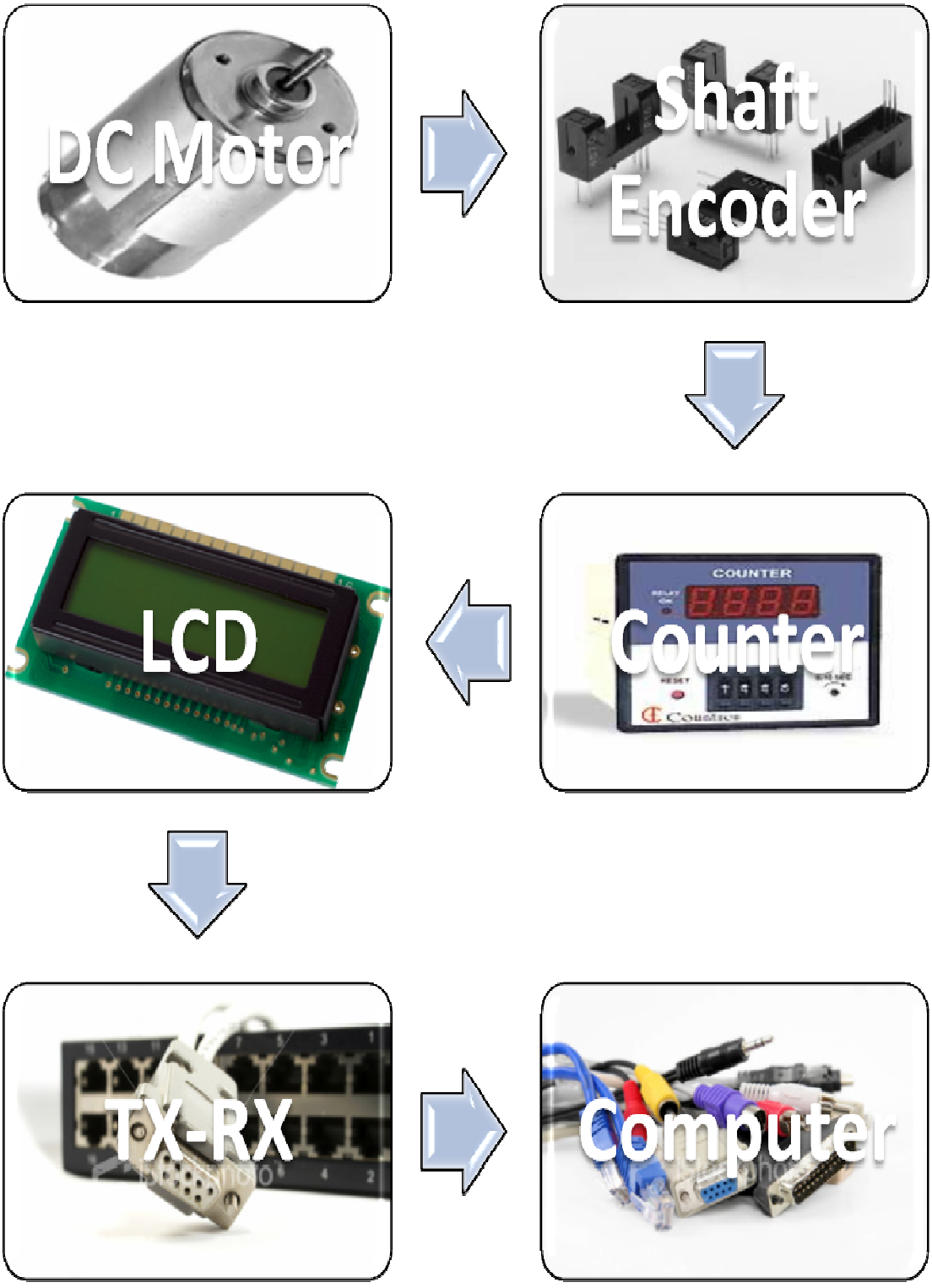
پورت C

پورت C یک I/O دو طرفه ۸ بیتی است . سه آدرس از مکان حافظه I/O اختصاص به PORTC دارد ؛ یک آدرس برای رجیستر داده PORTC ، دومی برای رجیستر جهت داده DDRC و سومی برای رجیستر ورودی PINC .

پورت D

پورت D یک I/O دو طرفه ۸ بیتی است . سه آدرس از مکان حافظه I/O اختصاص به PORTD دارد ؛ یک آدرس برای رجیستر داده PORTD ، دومی برای رجیستر جهت داده DDRD و سومی برای رجیستر ورودی PIND .

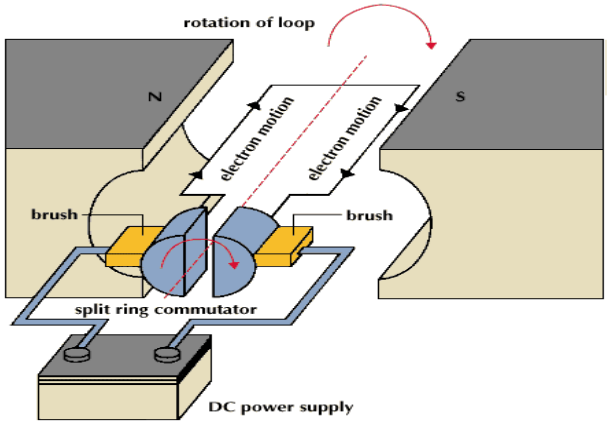
بلوک دیاگرام مدار:



اکنون هر یک از قسمتهای مدار به صورت کامل مورد بررسی قرار می دهیم

● موتور:

یک موتور الکتریکی ، الکتریسیته را به حرکت مکانیکی تبدیل می‌کند. وقتی که یک ماده حامل جریان الکتریسیته تحت اثر یک میدان مغناطیسی قرار می‌گیرد، نیرویی بر روی آن ماده از سوی میدان اعمال می‌شود. در یک موتور استوانه‌ای ، **روتور** به علت گشتاوری که ناشی از نیرویی است که به فاصله‌ای معین از محور روتور به روتور اعمال می‌شود، می‌گردد.



موتور کلاسیک DC دارای آرمیچری از آهنربای الکتریکی است. یک سوییچ گردشی به نام کموتاتور جهت جریان الکتریکی را در هر سیکل دو بار برعکس می کند تا در آرمیچر جریان یابد و آهنرباهای الکتریکی، آهنربای دائمی را در بیرون موتور جذب و دفع کنند. سرعت موتور DC به مجموعه ای از ولتاژ و جریان عبوری از سیم پیچهای موتور و بار موتور یا گشتاور ترمزی ، بستگی دارد.

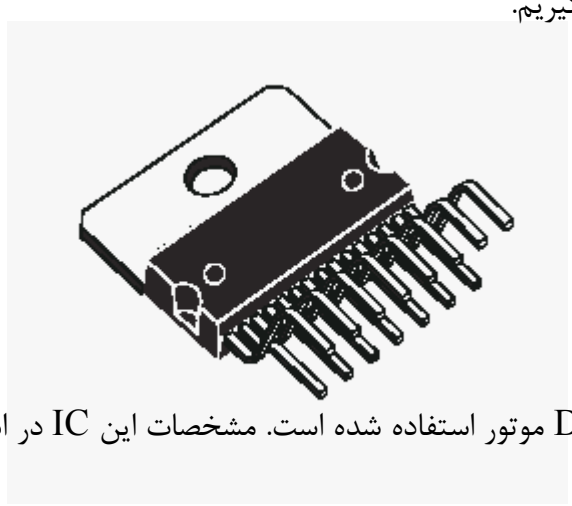
سرعت موتور DC وابسته به ولتاژ و گشتاور آن وابسته به جریان است. معمولاً سرعت توسط ولتاژ متغیر یا عبور جریان و با استفاده از تپها (نوعی کلید تغییر دهنده وضعیت سیم پیچ) در سیم پیچی موتور یا با داشتن یک منبع ولتاژ متغیر ، کنترل می‌شود. بدلیل اینکه این نوع از موتور می‌تواند در سرعتهای پایین گشتاوری زیاد ایجاد کند، معمولاً از آن در کاربردهای ترکشن (کششی) نظیر لکوموتیوها استفاده می‌کنند .

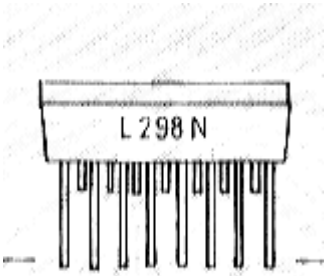
اما به هرحال در طراحی کلاسیک محدودیتهای متعددی وجود دارد که بسیاری از این محدودیتها ناشی از نیاز به جاروبکهایی برای اتصال به کموتاتور است. سایش جاروبکها و کموتاتور ، ایجاد اصطکاک می‌کند و هر چه که سرعت موتور بالاتر باشد، جاروبکها می‌بایست محکمتر فشار داده شوند تا اتصال خوبی را برقرار کنند. نه تنها این اصطکاک منجر به سر و صدای موتور می‌شود بلکه این امر یک محدودیت بالاتری را روی سرعت ایجاد می‌کند و به این معنی است که جاروبکها نهایتاً از بین رفته نیاز به تعویض پیدا می‌کنند .اتصال ناقص الکتریکی

نیز تولید نویز الکتریکی در مدار متصل می‌کند. این مشکلات با جابجا کردن درون موتور با بیرون آن از بین می‌روند، با قرار دادن آهنرباهای دائم در داخل و سیم پیچها در بیرون به یک طراحی بدون جاروبک می‌رسیم.

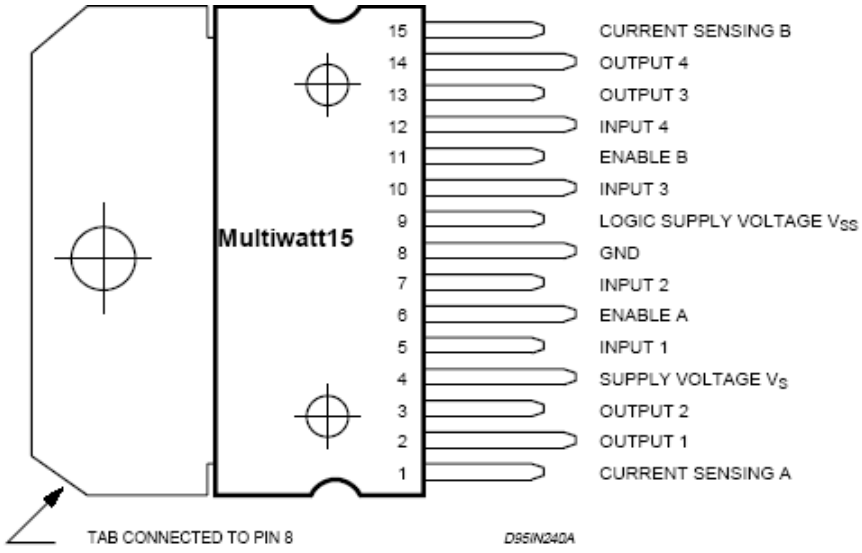
یک موتور ۱۲ ولت ساده را مدل خواهیم کرد. پس حداکثر ولتاژی که بر روی موتور می توا ن قرار داد، برابر ۱۲ ولت در نظر میگیریم.

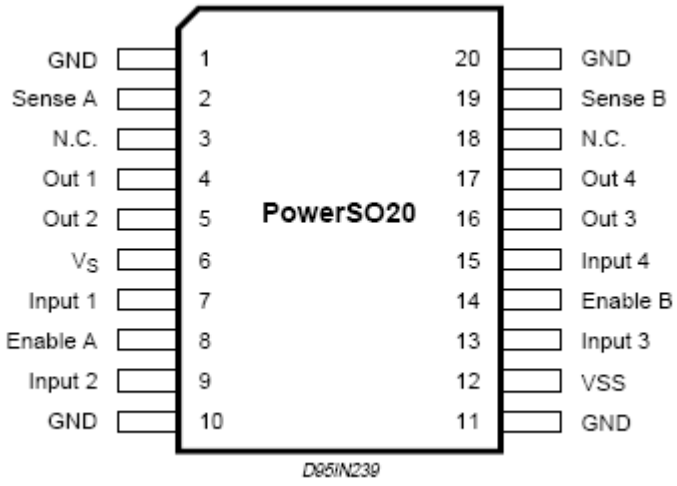
DC Motor Driver:





در این پروژه از L298 برای DRIVE موتور استفاده شده است. مشخصات این IC در ادامه نشان داده شده است.

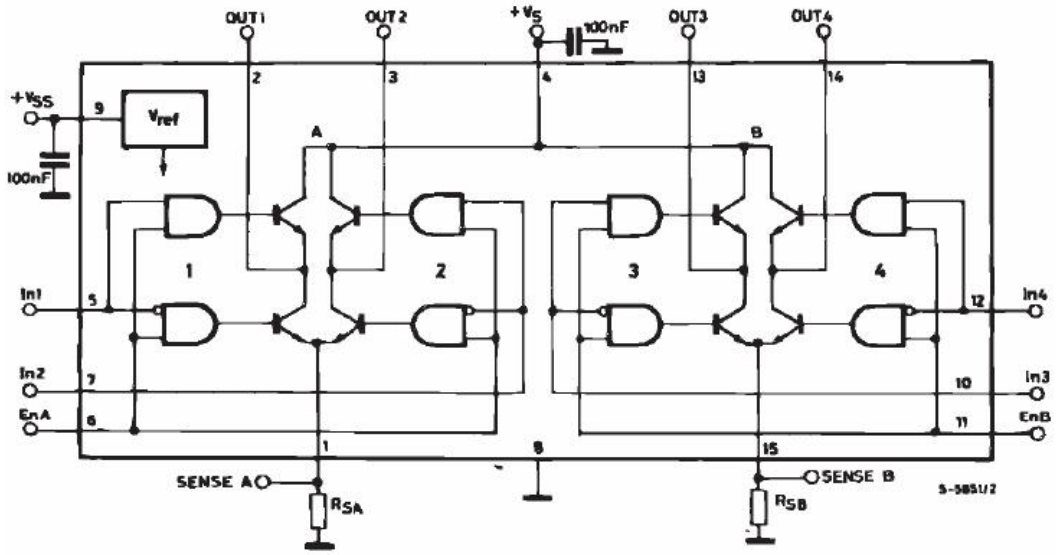




ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I, V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

این IC از دو پل H ساخته شده است. در نتیجه می توان با استفاده از این IC دو موتـور را بطـور همزمان کنترل نمود که ما تنها از یک خروجی آن جهت کنترل موتور خود استفاده خـواهیم کـرد. شماره پایه های ورودی و خروجی متناظر را از شکل زیر برای یک پل H بدسـت آورده و اتصـالات مروبطه را برقرار خواهیم کرد. در ادامه مدار کامل مورد نظر ارائه شده است.

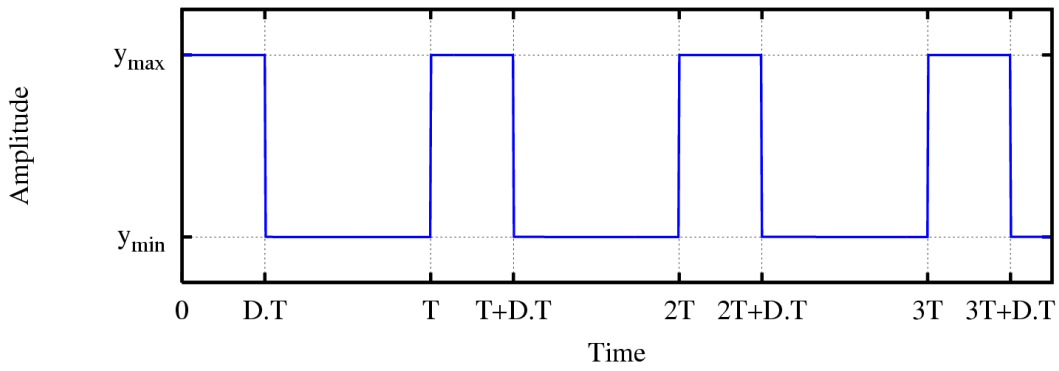


سیگنال PWM خروجی از میکرو را به پایه ENABLE این IC (L298) می دهـیم. بـه ایـن ترتیب این IC متناسب با پهنای پالس مدوله شده، فعال و غیر فعال شده و ولتاژ موتور قطع و وصل می شود و در نتیجه دور موتور بسته به Duty Cycle سیگنال PWM، کـاهش یـا افـزایش مـی یابد.

PWM:

این کلمه مخفف Pulse Width Modulation به معنای مدولاسیون پهنای پالس می باشد. یا به عبارتی دیگر یعنی با تغییراتی در پهنای پالس، توان (قدرت) الکتریکی انتقالی به موتور را کاهش یا افزایش می دهیم. وقتی می گوییم موتور DC با ولتاژ DC دارای دور نامی مشخصی می باشد، یعنی اگر ولتاژی با مقدار معین را در سر موتور قرار دهیم، در این صورت قدرت انتقالی به موتور ثابت بوده و در نتیجه موتور با دور نامی خود، کار خواهد کرد. ولتاژ DC یعنی ولتاژ ثابت. میدانیم که یک ولتاژ ثابت همان پالس با پهنا پالس دلخواه است. یعنی به ازای این مقادیر زمانی ولتاژ دو سر بار مقداری مستقیم است. حال اگر ما به هر نحوی این ولتاژ مستقیم روی دو سر موتور را کاهش دهیم، نتیجتاً قدرت انتقالی به موتور و در نتیجه دور موتور کمتر خواهد بود. با ادامه بحث هر چه بیشتر متوجه منظورم خواهید شد.

اصل و مبنای PWM تغییر (مدوله کردن) پهنای پالس و در نتیجه تغییر مقدار متوسط ولتاژ موج است.در صورتی که یک موج مربعی را در نظر بگیریم، در این صورت خواهیم داشت:



می دانیم:

$$\bar{y} = \frac{1}{T}\int_0^T f(t)\,dt$$

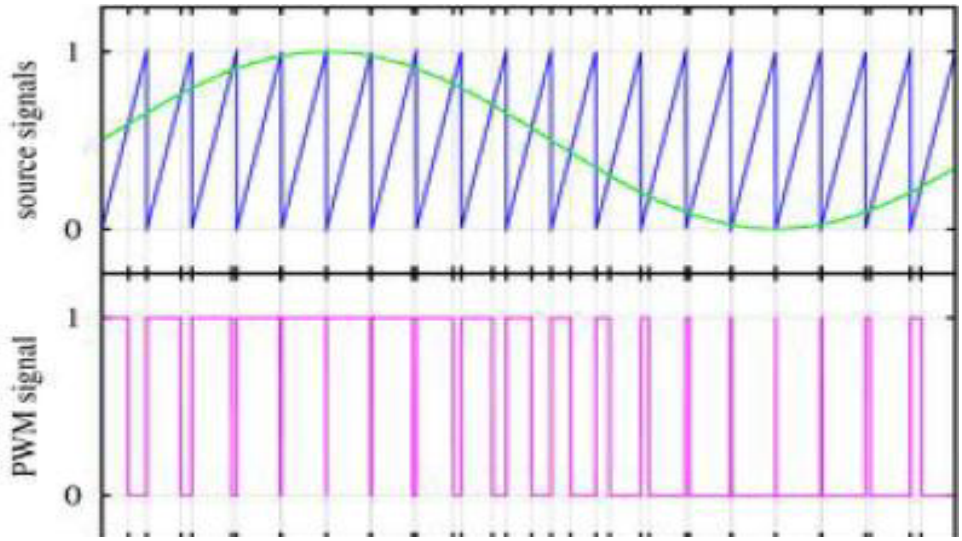
در شکل موج فوق،

$$\begin{aligned}\bar{y} &= \frac{1}{T}\left(\int_0^{DT} y_{max}\,dt + \int_{DT}^{T} y_{min}\,dt\right)\\ &= \frac{D\cdot T\cdot y_{max} + T(1-D)y_{min}}{T}\\ &= D\cdot y_{max} + (1-D)\,y_{min}\end{aligned}$$

که در آن y مقدار ولتاژ یکسو شده میباشد.

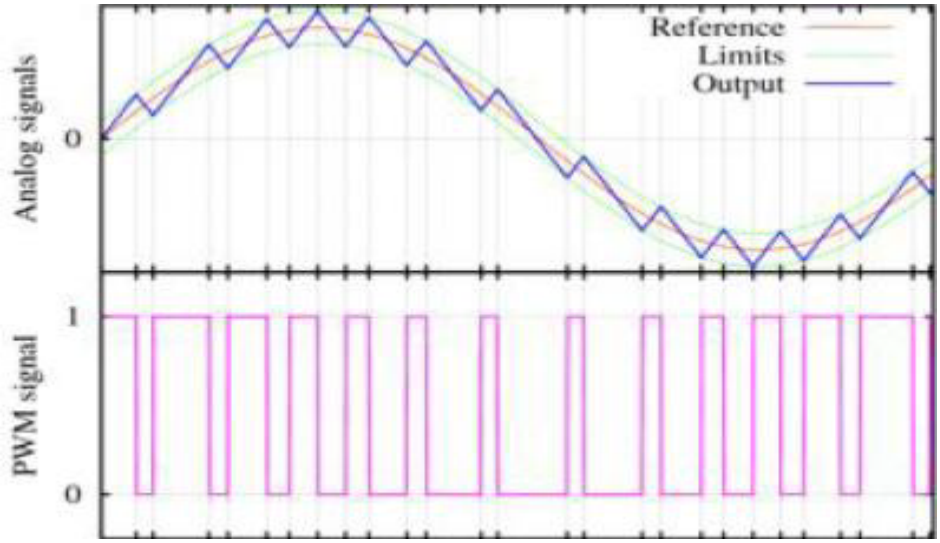
روش های تولید PWM:

۱) ساده ترین راه ایجاد PWM استفاده از یک موج اره ای و یک موج سینوسی ایجاد ده توسط اسیلاتور می باشد. حال کافی است تا با استفاده از یک OpAmp این دو ولتاژ را با یکدیگر مقایسه شوند. در صورتی که ولتاژ سبز رنگ در شکل زیر بیشتر از ولتاژ آبی رنگ باشد، سطح ولتاژ بالا و در صورتی که منحنی سبز رنگ از منحنی آبی رنگ پائین تر باشد، سطح ولتاژ پائین خواهد بود.



۲) روش DELTA:

در این روش ولتاژ خروجی با دو سطح ولتاژ معین که یکی از آنها همان مقدار ولتاژ اول با مقداری Offset می باشد، مقایسه میشود. در صورتی که ولتاژ خروجی از یکی از این دو محدودیت افزایش یا کاهش یابد، در این صورت سطح ولتاژ پالس نیز تغییر خواهد کرد. شکل زیر بیان گر این موضوع می باشد.



روش ۳) در این روش ولتاژ خروجی از یک ولتاژ مرجع کمتر میشود، در صورتی کـه مجمـوع ایـن ولتاژ خطا (تفاضل ولتاژ خروجی از ولتاژ مرجع) از مقدار معینی بیشتر شود، سطح ولتـاژ خروجـی عوض میشود.

روش ۴) بسیاری از مـدارات دیجیتـال مـی تواننـد PWM تولیـد کننـد. بـرای مثـال بسـیاری از میکروپروسسور ها دارای خروجی PWM می باشند. معمولاً این میکروپروسسور ها درای شـمارنده ای می باشند که پس از زمان معینی سطح ولتاژ خروجی را تغییر می دهند.

علاوه بر روش های گفته شده برای تولید PWM، سه حالت کلی برای PWM موجود می باشد:

۱) مرکز پالس در روی محور زمانی ثابت باشد و با افزایش یا کـاهش کنـاره هـای (لبـه هـای کناری) مدولاسیون پالس رو تغییر دهیم.

۲) لبه بالایی را ثابت نگه داریم و لبه پایینی را تغییر دهیم

۳) لبه پایینی را ثابت نگه داریم و لبه بالایی را تغییر دهیم.

از سایر موارد کار برد PWM علاوه بر کنترل دور موتور، می توان کـار بـرد آن در مخـابرات و تنظیم ولتاژ و پخش توان اشاره کرد.

در این پروژه از Timer2 برای ایجاد PWM ، در مد Fast PWM استفاده شده است.

```
//Timer /Counter 2 initialization
//Clock source: System Clock
//Clock value : 11.719 kHz
//Mode: Fast PWM top=FFh
//OC 2 output: Non-Inverted PWM
ASSR=0x00;
TCCR2=0x6F;
TCNT2=0x00;
OCR2=0x00;
```

Shaft Encoder ●

می توان این انکودر را بطور حاضری خریداری کرد و یا با هزینه ای کمتر، آن را ساخت. در آین قسمت به توضیح نحوه ساخت انکودر مکانیکی خواهیم پرداخت یعنی سیستم داخلی انکودر الکترومکانیکی است.

سیستم داخلی این نوع انکورد متشکل از یک سنسور نوری می باشد که موج (سیگنال نور با فرکانس معین) از فرستنده این سنسور صادر شده و در طرف گیرنده دریافت می شود. این سنسور انواع گوناگونی دارد. چند نمونه از انواع آن در شکل های زیر نشان داده شده اند.

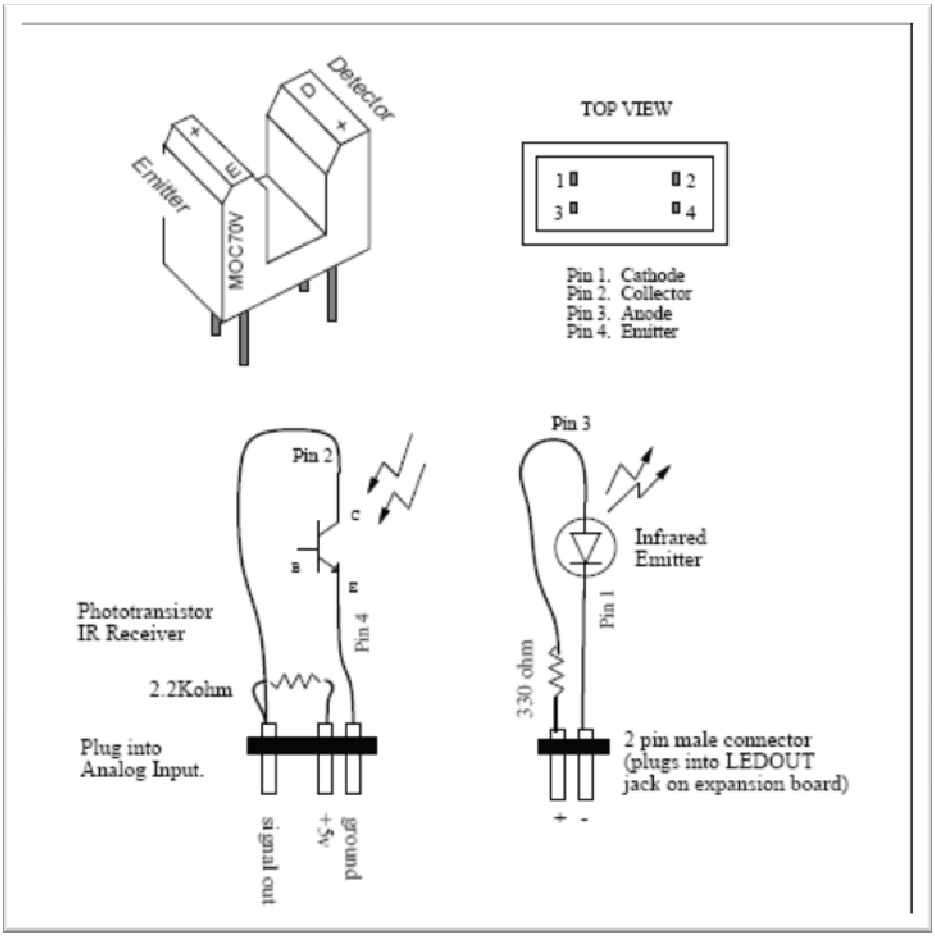
EE-SX1109	EE-SX199	EE-SX398/498	EE-SV3	EE-SX1071	EE-SX1096
Transmissive slot width 3mm - < 5mm					
					
6 x 4 x 5	12.2 x 5 x 10	12.2 x 5 x 10	19 x 15.1 x 10.2	13.6 x 6.2 x 10.2	25 x 6 x 10
Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive
3mm	3mm	3mm	3.4mm	3.4mm	3.4mm
0.5mm	0.5mm	0.5mm	0.2/0.5/1.0mm	0.5mm	0.5mm
940nm	940nm	940nm	940nm	940nm	940nm
Surface Mount	Through-hole	Through-hole	Through-hole	Through-hole	Lead Wires

E-SX1088	EE-SH3	EE-SX3088/4088	EE-SG3/SG3B	EE-SX1057	EE-SX1128	EE-SX1041	EE-SX1042	EE-SX1081	EE-SX1235A-P2	EE-SX3009-P1 /4009-P1	EE-SX4019-P2
Transmissive slot width 3mm - < 5mm							Transmissive slot width 5				
											
25 x 6 x 10	25.4 x 6.2 x 10.2	25 x 6 x 10	25.4 x 6.3 x 11.5	13 x 6.3 x 8.6	13.5 x 5.2 x 9.3	14 x 6 x 10	14 x 5 x 14.5	13.7 x 5 x 10	27 x 8 x 15.9	34 x 11 x 21	38 x 11 x 21
Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive
3.4mm	3.4mm	3.4mm	3.6mm	3.6mm	4.2mm	5mm	5mm	5mm	5mm	5mm	5mm
0.5mm	0.2/0.5/1.0mm	0.5mm	2.0mm	2.0mm	0.5mm	0.5mm	0.5mm	0.5mm	0.5mm	0.5mm	0.5mm
940nm	940nm	940nm	940nm	940nm	940nm	940nm	940nm	940nm	940nm	940nm	940nm
Through-hole	Through-hole	Through-hole	Through-hole	Through-hole	Through-hole	Through-hole	Through-hole	Through-hole	Snap-In	Screw Mounting	Screw Mounting

EE-SX3081 /4081	EE-SX4235A-P2	EE-SX1070	EE-SX3070 /4070	EE-SX1140	EE-SX461-P11
m - 8mm				Transmissive slot width over 12mm	
					
13.7 x 5 x 10	27 x 8 x 15.9	17.7 x 6 x 10	17.7 x 6 x 10	23 x 5 x 16.3	32.5 x 12 x 23.6
Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive
5mm	5mm	8mm	8mm	14mm	15mm
0.5mm	0.5mm	0.5mm	0.5mm	1.5mm	2.0mm
940nm	940nm	940nm	940nm	940nm	940nm
Through-hole	Snap-In	Through-hole	Through-hole	Through-hole	Snap-In

E-SX1107	EE-SX1018	EE-SX1103	EE-SX1105	EE-SX1108	EE-SX1131	EE-SX4134	EE-SX493	EE-SX1055	EE-SX1046	EE-SX1082	EE-SX1136
Transmissive slot width up to 3mm											
											
3.4 x 3 x 3	8 x 4 x 6	5 x 4.2 x 5.2	4.9 x 2.6 x 3.3	5 x 4 x 4	5 x 4 x 4	5 x 4 x 4	11 x 8 x 9.5	8.9 x 4 x 5.4	10 x 6.5 X 5	10 x 6.5 x 5.2	6.4 x 4.2 x 5.4
Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive	Transmissive
1mm	2mm	2mm	2mm	2mm	2mm	2mm	2mm	2.8mm	3mm	3mm	3mm
0.15mm	0.5mm	0.4mm	0.4mm	0.3mm	0.3mm	0.3mm	0.2mm	0.5mm	0.5mm	0.2mm	0.4mm
940nm	940nm	950nm	950nm	940nm	940nm	940nm	940nm	940nm	920nm	920nm	950nm
urface Mount	Through-hole	Through-hole	Through-hole	Surface Mount	Surface Mount	Surface Mount	Through-hole	Through-hole	Through-hole	Through-hole	Through-hole

مدار داخلی این سنسورها تقریباً یکسان و بفرم زیر است:



همان طور که مشاهده می شود، سنسور ها متشکل از دو قسمت فرستنده و گیرنده هستند. در صورتی که مانعی در بین فرستنده و گیرنده موجود نباشد، سیگنال فرستاده شده توسط گیرنده دریافت شده و فوتو ترانزیستور مربوطه شروع به هدایت می کند. در این صورت ولتاژ ۵ ولت ورودی که به پایه این سنسور وصل شده است، در خروجی ظاهر خواهد شد. سنسوری که در این پروژه استفاده خواهیم کرد، H22A1 خواهد بود. در بازار این قطعات با نام Opto Counter شناخته شده هستند ، در واقع شمارنده نوری.

در صورتی که به برگه اطلاعاتی این سنسور مراجعه کنید، اطلاعات بیشتری در مورد جزئیات این IC خواهید یافت. در اینجا به بخشی از این جزئیات شاره می شود:

ELECTRICAL CHARACTERISTICS (T_A = 25°C Unless Otherwise Specified)

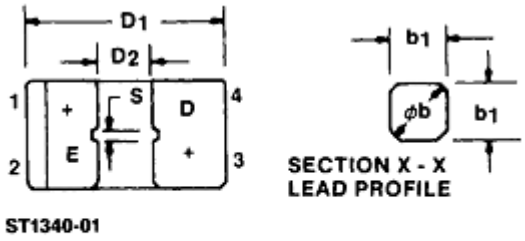
PARAMETER	SYMBOL	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
INPUT DIODE						
Forward Voltage	V_F	—		1.7	V	I_F = 60 mA
Reverse Breakdown Voltage	V_R	6.0		—	V	I_R = 10μA
Reverse Leakage Current	I_R	—		1.0	μA	V_R = 3 V
OUTPUT TRANSISTOR						
Emitter-Collector Breakdown	BV_{ECO}	6.0		—	V	I_E = 100 μA, Ee = 0
Collector-Emitter Breakdown	BV_{CEO}	30		—	V	I_C = 1 mA, Ee = 0
Collector-Emitter Leakage	I_{CEO}	—		100	nA	V_{CE} = 25 V, Ee = 0
COUPLED						
On-State Collector Current	$I_{C(ON)}$		See page 3.		mA	
Saturation Voltage	$V_{CE(SAT)}$		See page 3.		V	
Turn-On Time	t_{on}		See page 3.		μS	
Turn-Off Time	t_{off}		See page 3.		μS	

ABSOLUTE MAXIMUM RATINGS (T_A = 25°C Unless Otherwise Specified)

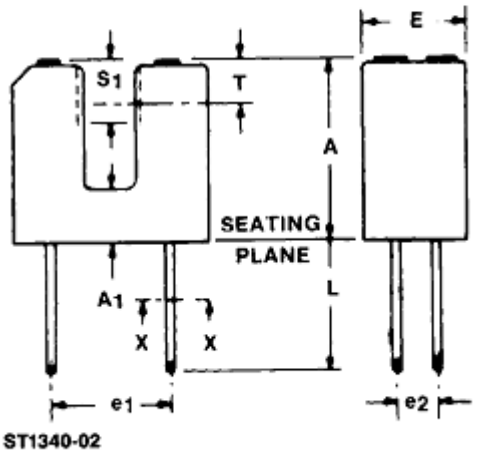
Storage Temperature ..... −55°C to +100°C
Operating Temperature ..... −55°C to +100°C
Soldering:
Lead Temperature (Iron) ..... 240°C for 5 sec.[3,4,5]
Lead Temperature (Flow) ..... 260°C for 10 sec.[3,4]

INPUT DIODE
Continuous Forward Current ..... 60 mA
Reverse Voltage ..... 6.0 Volts
Power Dissipation ..... 100 mW[1]

OUTPUT TRANSISTOR
Collector-Emitter Voltage ..... 30 Volts
Emitter-Collector Voltage ..... 6 Volts
Power Dissipation ..... 150 mW[2]

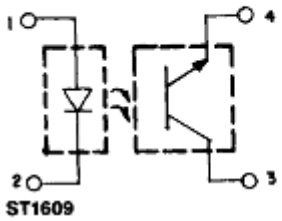


ST1340-01



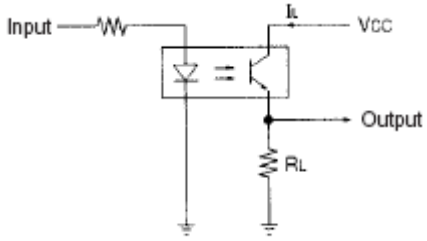
ST1340-02

PACKAGE OUTLINE



آنچه که باید توجه کنید، مقادیر ماکزیمم این IC در دو طرف فرستنده (Emitter) و گیرنده (Detector) می باشد.

مدار معادلی که می توان برای IC در نظر گرفت طبق شکل زیر است:



با توجه به اطلاعات داده شده در قسمت های بالا، مقدار جریان حداکثر در سمت فرستنده (Emitter) 20mA می باشد. در نتیجه باید جریان گذرنده از فرستنده محدود گردد که برای این کار مطابق مدار معادل، مقاومتی در مسیر فرستنده باید قرار گیرد(۳۳۰ اهم).

حال در صورتی که مانعی بین فرستندهو گکیرنده قرار گیرد ولتاژ خروجی برابر ۵ ولت (سطح منطق ۱) و در صورتی که مانعی بین فرستنده و گیرنده موجود نباشد، ولتاژ خروجی برابر صفر ولت (منطق صفر) خواهد بود. حال کافی است تا با اتصال یک چرخ دنده یا شیئ مشابه که با چرخش موتور در ارتباط بین گیرنده و فرستنده؛ قطع و وصل ایجاد کند، سیگنال Digital مورد نظر را بدست آورد.

متناسب با خروجی این Opto Counter میتوان از پایه های intrupt میکرو و یا با استفاده از Timer هاس داخلی میکرو مقدار دور موتور در ثانیه (RPS) یا دور موتور در دقیقه (RPM) را محاسبه کرد که در ادامه توضیح داده خواهد شد.

Counter ●

میکرو کنترلر ATmega16 سه تایمر/کانتر دارد که به طور مختصر آنها را معرفی می کنیم :

- **تایمر/کانتر صفر و رجیستر های مربوطه**

تایمر/کانتر هشت بیتی صفر می تواند کلاک خود را از سیستم ، تقسیمی از کلاک سیستم و یا از پایه خروجی T0 تامین نماید . تایمر/کانتر صفر توسط رجیستر کنترلی TCCR0 می تواند متوقف شود . وقفه های آن توسط رجیستر TIMSK (Timer/Counter Interrupt Mask Enable) می توانند فعال/غیرفعال شوند . پرچم سرریز در رجیستر TIFR موجود است .

تایمر/کانتر صفر ، دو رجیستر بسیار مهم دارد :

TCNT0: این رجیستر ۸ بیتی محتوای تایمر/کانتر را در خود جای می دهد .

TCCR0: این رجیستر ۸ بیتی ، رجیستر کنترلی تایمر/کانتر است

- **تایمر/کانتر یک و رجیستر های مربوطه**

تایمر/کانتر ۱۶ بیتی صفر می تواند کلاک خود را از سیستم ، تقسیمی از کلاک سیستم و یا از پایه خروجی T1 تامین نماید . تایمر/کانتر یک توسط رجیستر کنترلی TCCR1A و TCCR1B می تواند متوقف شود . وقفه های آن توسط رجیستر TIMSK (Timer/Counter Interrupt Mask Enable) می توانند فعال/غیرفعال شوند .

تایمر/کانتر یک ، سه رجیستر بسیار مهم دارد :

TCNT1L: این رجیستر ۸ بیتی محتوای کم ارزش تایمر/کانتر را در خود جای می دهد .

TCNT1H: این رجیستر ۸ بیتی محتوای با ارزش تایمر/کانتر را در خود جای می دهد .

TCCR1B: این رجیستر ۸ بیتی ، رجیستر کنترلی تایمر/کانتر است

- **تایمر/کانتر دو و رجیستر های مربوطه**

تایمر/کانتر ۸بیتی دو قابلیت انتخاب کلاک از کلاک سیستم ، تقسیمی از کلاک سیستم یا از پایه های خروجی به صورت آسنکرون را داراست . تایمر/کانتر دو با توجه به تنظیمات رجیستر کنترلی TCCR2 می تواند متوقف شود . پرچم های سرریز در رجیستر TIFR موجود می باشند .

فعال/غیرفعال کردن وقفه های تایمر/کانتر دو در رجیستر TIMSK قابل تنظیم می باشند .

از تایمر/کانتر دو بیشتر برای سرعت های پایین و ایجاد زمان های دقیق با دقت و وضوح بالا استفاده می شود .

تایمر/کانتر دو ، دو رجیستر بسیار مهم دارد :

TCNT2: این رجیستر ۸ بیتی محتوای تایمر/کانتر را در خود جای می دهد .

TCCR2: این رجیستر ۸ بیتی ، رجیستر کنترلی تایمر/کانتر است.

وقفه و منابع آن:

AVR از چندین منبع وقفه پشتیبانی می کند .این وقفه ها و reset ،بردار وقفه وreset جدا گانه در فضای برنامه نویسی حافظه دارند .هر کدام از این وقفه ها بیت فعال کننده مخصوص به خود را دارند که به همراه بیت I باید فعال شوند.پایین ترین آدرس حافظه برنامه نویسی به بردار های وقفه وreset اختصاص دارد.بردار وقفه ای که دارای پایین ترین آدرس است دارای بالا ترین اولویت بین وقفه ها است.

هنگام وقوع وقفه بیتI پاک شده و وقفه های دیگر غیر فعال می شوند .برنامه نویس می تواند با یک کردن بیتI وقفه بعدی را بپذیرد . ی‌ر این صورت تمامی وقفه های فعال می توانند در هنگام اجرا شدن برنامه سرویس یک وقفه ،فعال شوند . در صورتی که بیتI با برنامه نویسی یک نشود در انتهای برنامه سویس وقفه و با اجرا شدن دستور RETIیک می شود .

وقفه ها به طور کلی دارای دو نوعNon Maskable,Maskable هستندوقفه های Maskable وقفه هایی هستند که با استفاده از یک بیت دیگر می توان از وقوع آن جلوگیری نمود .اکثر وقفه های میکرو کنترلر از این نوع می باشد . وقفه های Non Maskable هایی هستند که با استفاده از یک بیت دیگر نمی توان از وقوع آن جلوگیری نمود.

وقفه ها از لحاظ تریگر شدن به دو دسته تقسیم می شوند .دسته اول وقفه هایی هستند که توسط یک رخداد تریگر شده و پرچم مربوط به وقفه خود را فعال می کنند .دومین نوع وقفه ها ،وقفه های هستند که تا مدت زمان برقرار بودن شرایط وقفه مربوط فعال هستند .این نوع وقفه به پرچمی احتیاج ندارد و با برطرف شدن شرایط وقفه ،برنامه سرویس وقفه اجرا نمی شود.

با توجه به اینکه میکرو های AVR دارای تعداد زیادی رجستر می باشد لذا در ابتدای برنامه فایلی را که رجستر ها و نام بیتها را متناظر با موقعیت بیتها مشخص کرده است با دستور include. ضمیمه نموده و از آن بعد به جای استفاده از شماره بیت از نام بیت و رجستر استفاده می شود .

پاسخ به وقفه حداقل جهار پالس ساعت طول می کشد .به عبارت دیگر در طی چهار پالس ساعت آدرس برنامه وقفه مشخص و مقدار PC در پشته نوشته شده است .

وقفه های خارجی توسط پایه های INT2,INT1,INT0 تریگر می شوند وقفه های INT1,INT0با لبه بالا رونده،پایین رونده یا با سطح پایین تریگر می شوند ولی وقفه INT2 فقط به لبه حساس است .می توان از این وقفه ها برای خارج شدن از مدهای sleep،استفاده کرد چرا که در تمامی مدهایsleep به غیر ازمدIdle کلاک I/O متوقف می شود .

تعدادی از بردار های وقفه خارجی, resetدر جدول صفحه بعد اشاره شده است :

Name	Int Vector Address			triggered by …
	2313	2323	8515	
RESET	0000	0000	0000	Hardware Reset, Power-On Reset, Watchdog Reset
INT0	0001	0001	0001	Level change on the external INT0-Pin
INT1	0002	-	0002	Level change on the external INT1-Pin
TIMER1 CAPT	0003	-	0003	Capture event on Timer 1
TIMER1 COMPA	-	-	0004	Timer1 = Compare A
TIMER1 COMPB	-	-	005	Timer1 = Compare B
TIMER1 COMP1	0004	-	-	Timer1 = Compare 1
TIMER1 OVF	0005	-	0006	Timer1 Overflow
TIMER0 OVF	0006	0002	0007	Timer0 Overflow
SPI STC	-	-	0008	Serial transmit complete
UART RX	0007	-	0009	UART char in receive buffer available
UART UDRE	0008	-	000A	UART transmitter ran empty
UART TX	0009	-	000B	UART All sent
ANA_COMP	-	-	000C	Analog Comparator

رجستر کنترل وقفه عمومی(GICR)

بیت۷-INT1
برای فعال سازی وقفه خارجی یک بکار می رود.
بیت۶-INT0
برای فعال سازی وقفه خارجی صفربکار می رود.
بیت۵-INT2
برای فعال سازی وقفه خارجی دو بکار می رود.

رجستر GIFR(General interrupt flag register)

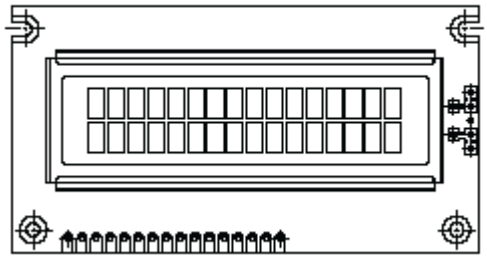
بیت ۷-INTF1
زمانی که یک متغیر منطقی در سطح یا لبه ،در پایهINT1 وقفه مربوط به آن را تریگر کند این بیت یک می شود .
بیت۶-INTF0
زمانی که یک تغییر منطقی در سطح یا لبه ،در پایهINT0 وقفه مربوط به آن را تریگر کند این بیت یک می شود .
بیت ۵-INTF2
زمانی که یک متغیر منطقی در سطح یا لبه ،در پایه INT2 وقفه مربوط به آن را تریگر کند این بیت یک می شود .

در این پروژه از ۲ تایمر و ۱ اینتراپت استفاده شده است.
Timer 1 برای شمارش زمان ۱ ثانیه.
Timer2 برای ایجاد موج PWM.
INT0 برای شمارش تعداد پالس های حاصل از تعداد دور موتور در مدت زمان مشخص.
هدف اصلی شمارش تعداد دور موتور در مدت ۱ ثانیه و سپس ارسال آن به کامپیوتر میباشد. از این رو ابتدا باید تعداد پالس ها در مدت ۱ ثانیه شمرده شوند. برای این منظور خروجی انکودر را به پایه INT0 میکرو وصل کردیم و با استفاده از Timer 1 overflow interrupt service routine زمان ۱ ثانیه محاسبه شده و در نهایت RPS محاسبه میشود.

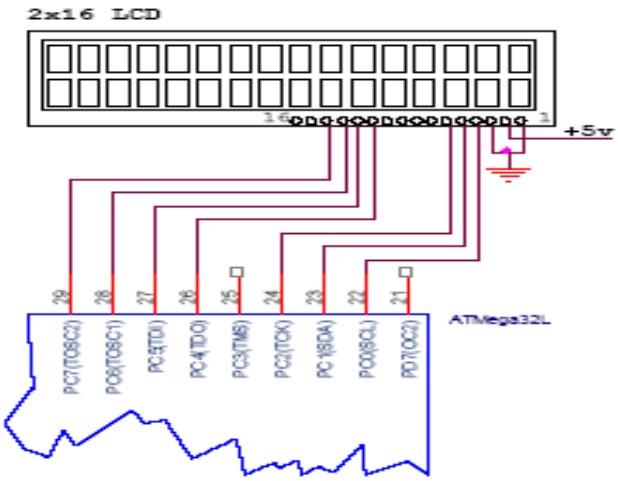
قسمتی از برنامه که مرتبط با بحث فوق میباشد در اینجا نمایش داده میشود :

```
////////////////////////////////////////////////////
//      External Interrupt 0 service routine      //
////////////////////////////////////////////////////
interrupt [EXT_INT0] void ext_int0_isr( void(
{
      speed_ctr++;
}

//**********************************//
//**********************************//

////////////////////////////////////////////////////
//      Timer 1 overflow interrupt service routine      //
////////////////////////////////////////////////////
interrupt [TIM1_OVF]  void timer1_ovf_isr( void)
{
      Ctr++ ;
      if(ctr==4)
      {
            ctr=0 ;
            printf("%d\r",speed_ctr) ;
            speed_ctr=0 ;
       }
      TCNT1=28036 ;
}
//**********************************//
//**********************************//
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

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

● نمایش بر روی LCD:



نحوه اتصال LCD به میکروکنترلر:



پایه های ۱و ۲ به ترتیب به زمین و ولتاژ۵ولت متصل می شوند.پایه ۳میزان شدت نمایش کاراکترها بر روی LCD را مشخص می کند(contrast).پایه های ۴،۵،۶نیز به کنترل LCD مربوط می شود که باید توسط میکرو کنترلر تنظیم شود .از ایه های ۷ تا ۱۴ نیز به عنوان یک باس ۸بیتی برای انتقال اطلاعات مابین LCDو میکروکنترلر استفاده می شود .برای تبادل اطلا عات بین LCDو میکروکنترلر از روش ارتباط باس چهار سیمه استفاده می شود.

پایه های ۷،۸،۹،۱۰در محیطهای پر نویز استفاده می شود که با مقاومتهای ۱۰کیلو اهم به زمین متصل می کنند(pull down)

پایه های ۱۵،۱۶ به ترتیب پلاریته مثبت و منفی برای روشن کردن نور زمینه LCD استفاده می شود.

PIN NUMBER	SYMBOL	FUNCTION
1	Vss	GND
2	Vdd	+ 3V or +5V
3	Vo	Contrast Adjustment
4	RS	H/L Register Select Signal
5	R/$\overline{W}$	H/L Read/Write Signal
6	E	H →L Enable Signal
7	DB0	H/L Data Bus Line
8	DB1	H/L Data Bus Line
9	DB2	H/L Data Bus Line
10	DB3	H/L Data Bus Line
11	DB4	H/L Data Bus Line
12	DB5	H/L Data Bus Line
13	DB6	H/L Data Bus Line
14	DB7	H/L Data Bus Line
15	A/Vee	4.2V for LED/Negative Voltage Output
16	K	Power Supply for B/L (OV)

MECHANICAL DATA		
ITEM	STANDARD VALUE	UNIT
Module Dimension	84.0 x 44.0	mm
Viewing Area	66.0 x 16.0	mm
Dot Size	0.55 x 0.65	mm
Character Size	2.95 x 5.55	mm

ABSOLUTE MAXIMUM RATING					
ITEM	SYMBOL	STANDARD VALUE			UNIT
		MIN.	TYP.	MAX.	
Power Supply for Logic	VDD-VSS	- 0.3	–	7.0	V
Input Voltage	VI	- 0.3	–	VDD	V

NOTE: VSS = 0 Volt, VDD = 5.0 Volt

توابع کتابخانه LCD.h:

قبل از هر چیز لازم است مشخص شود که ماژول lcdبه کدامیک از پورتهای میکـرو کنترلـر وصـل شده است البته با استفاده از Code Vizardنیز می توان پورت را مشخص کرد.

Unsigned char lcd_init(unsigned char lcd-columns)

این تابع ماژولLCD را مقدار دهی اولیه می کند. با فراخوانی این تابع،صفحه نمایشLCD پاک شده،مکان نما نیز حذف می گردد وLCD برای نوشتن کاراکتر در محل سطر و ستون صفر آماده می شود.

Void lcd_clear(void)

این تابع،صفحه نمایشLCD را پاک می کند و برای چاپ کاراکتر در محل سطر و ستون صفر آماده می شود.

Void lcd_gotoxy(unsigned char x,unsigned char y)

این تابع موقعیت فعلی برای چاپ کاراکتر را به محل ستون xوسطرyمنتقل می کند . به طور کلی مبدا مختصات(۰و۰)در lcdهای کاراکتری ،بالا و سمت چپ صفحه نمایش است .

Void lcd_putchar(char c)

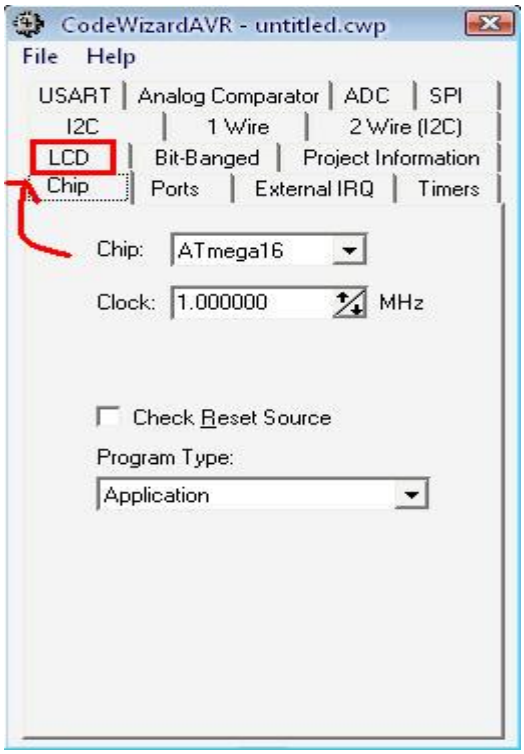
این تابع کاراکترc را بر روی موقعیت فعلی انتخاب شده بر روی lcdمی نویسد.

void lcd_putsf(char flash*str)

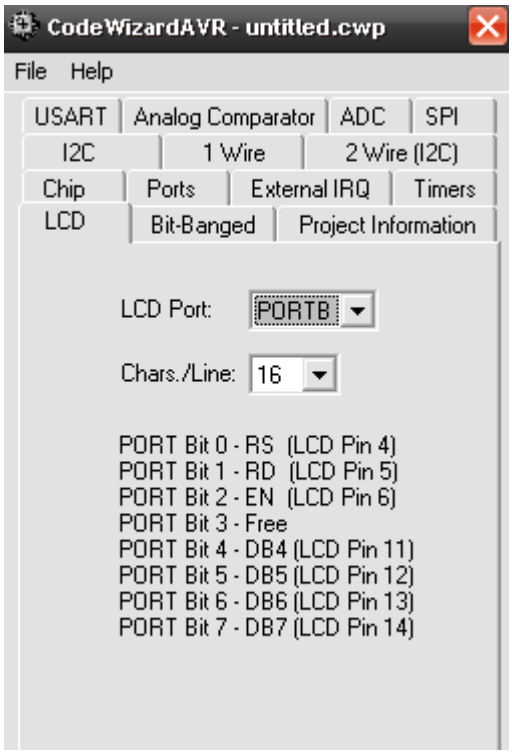
این تابع رشته ی واقع در حافظهflash را با شروع از موقعیت فعلی انتخاب شده ،بر روی lcdمی نویسد.

تنظیمات اولیه LCD در Code Wizard:

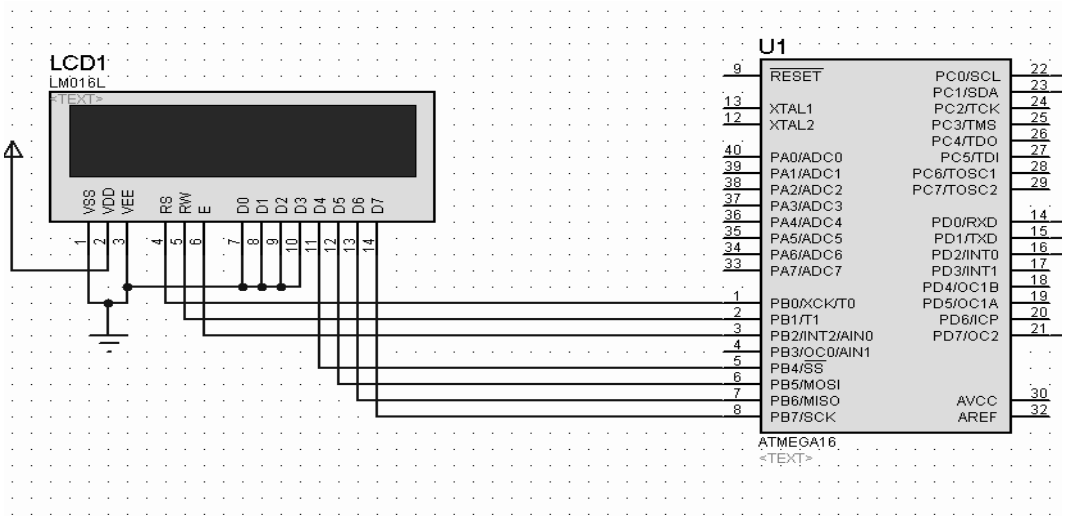
برای این کا کافی است تا پس از ایجاد یک پروژه جدید و استفاده از Wizard (که در ادامه نحوه انجام این کار در قسمت تولید PWM با جزئیات بیشتر بحث شده است) ایجاد کنید. حال در پنجره Wizard به قسمت LCD بروید.



حال در این قسمت تنها کافی است پورت مورد نظر را که قصد دارید LCD به آن را وصل کنید، معین
کنید.



با توجه به شکل بالا با انتخاب پورت B نحوه ی اتصال پایه های lcd به میکرو مشخص شده است.
که در زیر مدار آن رسم شده است.



● مدار واسط

در طرح ارائه شده مدار بعد از شمارش RPS موتور این عدد را به کامپیوتر خواهد فرستاد و منتظر تنظیم شدن سرعت و ارسال عدد دلخواه کاربر و یا عدد تنظیم شده توسط کنترل کننده خواهد بود. برای این منظور ازارتباط سریال UART مابین میکروکنترلر و کامپیوتر استفاده شده است.
از این رو ابتدا توضیحاتی کلی در رابطه با ارتباط سریال خواهیم داد و سپس در مورد نحوه اتصال میکروکنترلر به پورت کامپیوتربحث خواهیم کرد و در نهایت تنظیمات برنامه Codevision در این باره بررسی خواهد شد.

ارتباط سریال USART

بخش ارتباط سریال USART در میکروکنترلرهای AVR قابلیتهای متنوعی دارد که از جمله آنها می توان به موارد زیر اشاره کرد .

- عملکرد Full Duplex
- عملکرد سنکرون و آسنکرون
- عمل به صورت Master و Slave در حالت سنکرون
- تولید کننده نرخ ارسال (Baudrate) دقیق
- حمایت از فریم های سریال ۵ – ۶ – ۷ – ۸ -۹ بیت داده و ۱ یا ۲ بیت توقف
- تولید Parity به صورت زوج یا فرد و امکان چک کردن سخت افزاری آن
- تشخیص خطاهای سرریز و نوع فریم
- فیلتر پایین گذر دیجیتال
- تولید سه وقفه مجزا برای اتمام TX ، خالی شدن رجیستر TX و اتمام RX
- کار در حالت ارتباط چند پردازنده
- امکان دوبرابر کردن سرعت در حالت آسنکرون

سازگاری USART باUART در AVR

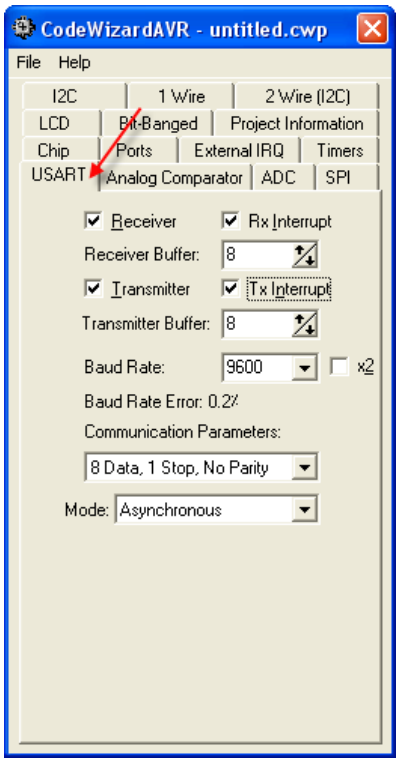
بعضی از انواع AVR تنها از ارتباط سریال UART حمایت میکنند ، به این معنی که ارتباط سریال تنها به صورت آسنکرون قابل انجام میباشد و انواع پیشرفته تر میکروکنترلرهای AVR می توانند به صورت سنکرون و آسنکرون ارتباط برقرار کنند. این دو نوع ارتباط سریال از نظر محل بیت ها در داخل رجیسترها ، نحوه تولید نرخ ارسال ، نحوه ارسال و دریافت اطلاعات و عملکرد بافر مربوط به ارسال اطلاعات کاملا مطابقت دارند و تنها عملکرد بافر مربوط به دریافت اطلاعات در ارتباط سریال USART بهبود یافته است.

نکاتی در مورد تولید نرخ ارسال

واحد تولید نرخ ارسال با استفاده از کلاک میکرو ، نرخ های مختلف را تولید میکند. نکته مهم در ارتباط سریال این است که تولید نرخ های ارسال مختلف به کمک نوسانگرها و کریستالهایی با فرکانس استاندارد میتواند خطایی به همراه داشته باشد و از آنجاییکه در بعضی موارد این خطا بیش از حد بزرگ است باید در هنگام طراحی به این نکته توجه نمود.
بنابر این اگر قصد استفاده از ارتباط سریال با کامپیوتر را داریم باید از کریستال هایی با فرکانس بالا استفاده کنیم که در این پروژه از کریستال 12.0 MHz استفاده شده است تا نرخ ارسال 9600bps بدون خطا ایجاد شود.

انجام تنظیمات اولیه ارتباط سریال در code wizard:

در صورتی که در برنامه code wizard،نوع تراشه را Atmega16 انتخاب کنیم با کلیک بر روی قسمت usart شکل زیر مشاهده می شود:



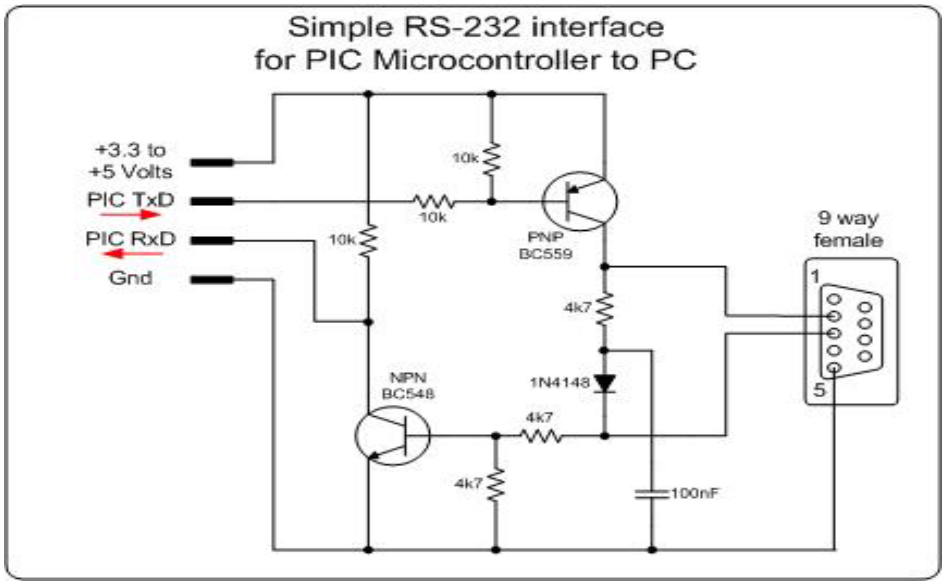
در این صفحه می توانیم در قسمتmodeمی عملکرد usart را به صورت آسنکرون یا سنکرون انتخاب کنیم.

برای فعال سازی ارتباط سریال به صورت گیرنده و یا فرستنده باید به ترتیب گزینه های receiver transmitter,را انتخاب کنیم .با فعال کردن هریک از آنها این امکان فراهم می شود تا بتوانیم وقفه مربوط به آن را نیز فعال کنیم.در صورتی که وقفه ارتباط سریال برای دریافت داده ها فعال گردد ، code visionیک بافر نرم افزاری به نام عمومی rx_bffer را تولید می کند که در آن متغیر عمومی rx_wr_indexهمواره شماره بایتی را که برای آخرین بار داده در آن قرار گرفته است ،در خود نگه می دارد .به همین ترتیب ،برای وقفه سریال مربوط به ارسال داده نیز یک بافربه نام عمومیtx_buffer,متغیر عمومی tx_wr_index برای تغییر اندیس آن تولید می شوند.برای کار با ارتباط سریال می توان به سادگی از توابع ورودی خروجی استانداردprintf,gets,getchar,puts,putchar,scanf استفاده نمود.

در صورتی که وقفه های مربوط به ارتباط سریال فعال شوند ،code wizardبه صورت خودکار توابع getchar,putchar را دوباره تعریف می کند به طوری که برای کار با وقفه ها مناسب باشد.

اتصال AVRبه RS232:

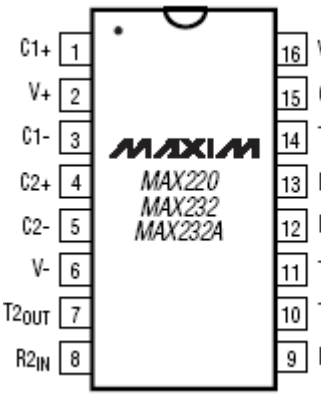
ازنجایی که برای برقراری ارتباط سریال با کامپیوتر ناچاریم از استفاده کنیم ،لازم است تا به نحوی سطوحTTL ایجاد شده توسط میکروو232RS را به یکدیگر تبدیل کنیم.عموما برای تبدیل این سطوح ولتاژ به یکدیگر از از تراشه MAX232 یاMAX233 استفاده می شود .در میکرو کنترل های AVRدو پایه به نام های TXD,RXD وجود دارند که از پایهTXD برای ارسال داده و از پایهRXD برای دریافت آنها استفاده می شود.

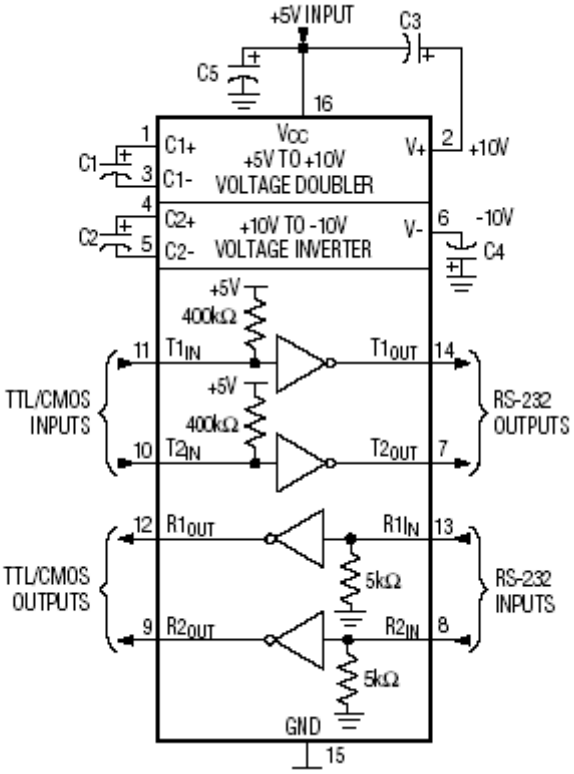


در پروژه فوق از تراشه MAX232 استفاده شده است ولی در زیر مختصرا در مورد هر دو تراشه پرکاربرد MAX233 , MAX232 اشاره میشود .

تراشه MAX232:

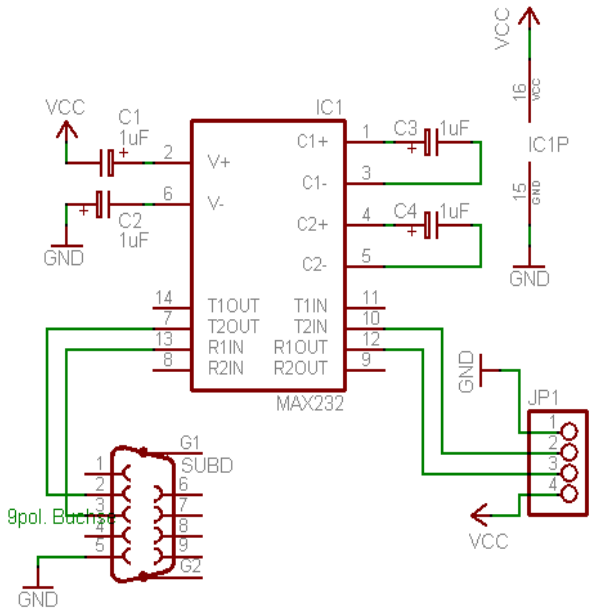
برای تبدیل سطوح ولتاژ TTL وRS232 به یکدیگر ،می توان از MAX232 استفاده نمود ،این تراشه به چهار خازن ۱تا ۲۲میکرو فاراد ،نیاز دارد که عموما از خازن ۲۲میکرو استفادخ می شود

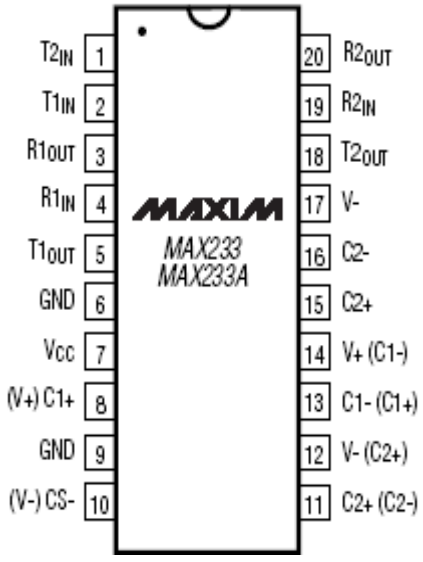




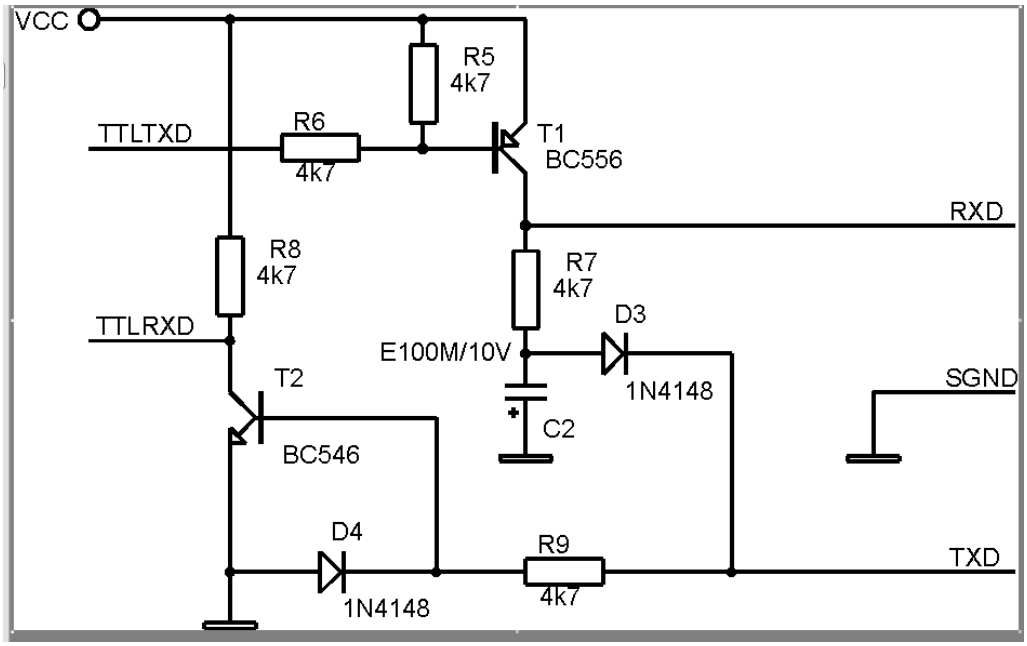
تراشه MAX23:

عملکرد این تراشه دقیقا مشابهMAX232 می باشد با این تفاوت که در اینجا دیگر نیازی به خازن خارجی نیست





مبدل ترانزیستوری منطق TTL وRS232 به یکدیگر:



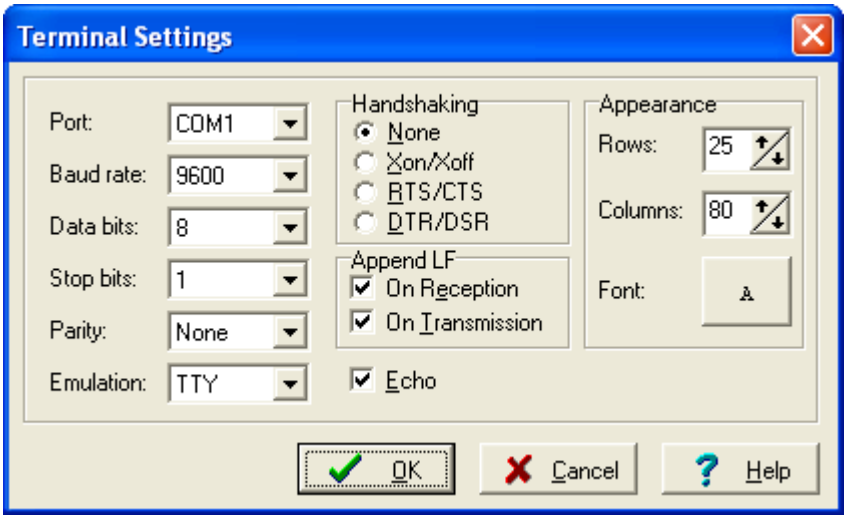
از این مدار می توان برای دریافت وارسال اطلاعات استفاده کرد.

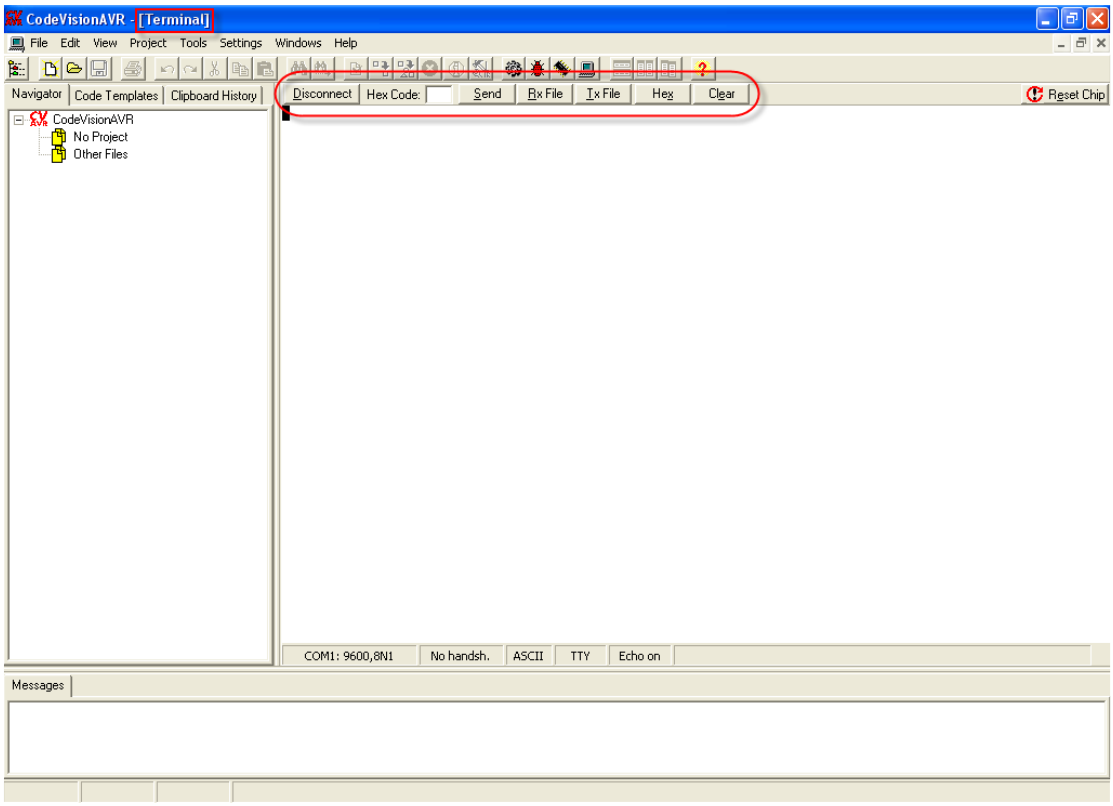
برنامه های وابسته:

در برقراری ارتباط سریال با کامپیوتر لازم است تا برای برای ارسال و دریافت داده ها از نحوه عملکرد برنامه اطمینان حاصل شود لذا می توان از برنامه هایی استفاده نمود که داده های ارسالی از میکرو رانشان دهد و داده های مورد نیاز را برای آن ارسال کند .از جمله این برنامه ها می توان به محیط ترمینال در داخل نر افزار اشاره کرد.

محیط ترمینال:

قبل از باز کردن این محیط لازم است تا تنظیمات مربوط به آن را انجام دهیم،برای این منظور باید در نرم افزار codevisionاز منویsetting،گزینه terminal را انتخاب کنیم.در این صورت پنجره ای باز می شود که میتوان در آن پورت متصل به میکرو مقدار نرخ ارسال و فرمت فریمهای تعریف شده در میکرو کنترلر را تنظیم کرد .پس از انجام تنظیمات می توان این برنامه را از طریق منویtools و یا نوار ابزار اجرا کرد .در این صورت صفحه ای باز می شود که اطلاعات رسیده از میکرو قابل مشاهده است .در صورتی که در این صفحه تایپ کنیم اطلاعات تایپ شده به میکرو فرستاده می شود .همچنین می توان اطلا عات دریافت شده از میکرو را در یک فایل متنی ذخیره کرد و یا محتویات یک فایل متنی را باا باز کردن آن فایل در این برنامه ،برای میکرو ارسال کرد.





همانطور که توضیح داده شد RPS موتور از طریق پایه TX میکروکنترلر به کامپیوتر ارسال میشود و RPS اصلاح شده یا دلخواه کاربر از طریق پایه RX وارد میکروکنترلر شده و در داخل OCR2 قرار میگیرد و PWM جدید از طریق Timer2 ایجاد میشود و به مدار درایور موتور داده میشود و در واقع کار کنترلر تکمیل میشود.

و اما توابعی و روشی که در ارتباط این سیستم مورد استفاده قرار گرفته است :

در این پروژه میکرو به طور متناوب بعد از ارسال عبارت START شروع به ارسال RPS موتور به پورت COM1 میکند. گفتنی است که نرخ ارسال 9600bps در نظر گفته شده است. مدت زمان نمونه گیری از دور موتور ۱ ثانیه است در واقع هر ۱ ثانیه یک عدد که همان تعداد دور موتور در مدت ۱ ثانیه میباشد به موتور ارسال میشود و در جواب منتظر دریافت کاراکتر "#" می ماند تا در صورت دریافت این کاراکتر میکرو آماده دریافت RPS جدید میشود.

پس بعد از ارتباط با کامپیوتر برای مثال در صورتی که در برنامه Codevision در قسمت Terminal عبارت 0040# وارد شود دور موتور در 40(HEX) تنظیم خواهد شد.

در اینجا قسمتی از برنامه را که متناسب با بحث می باشد قرار داده میشود:

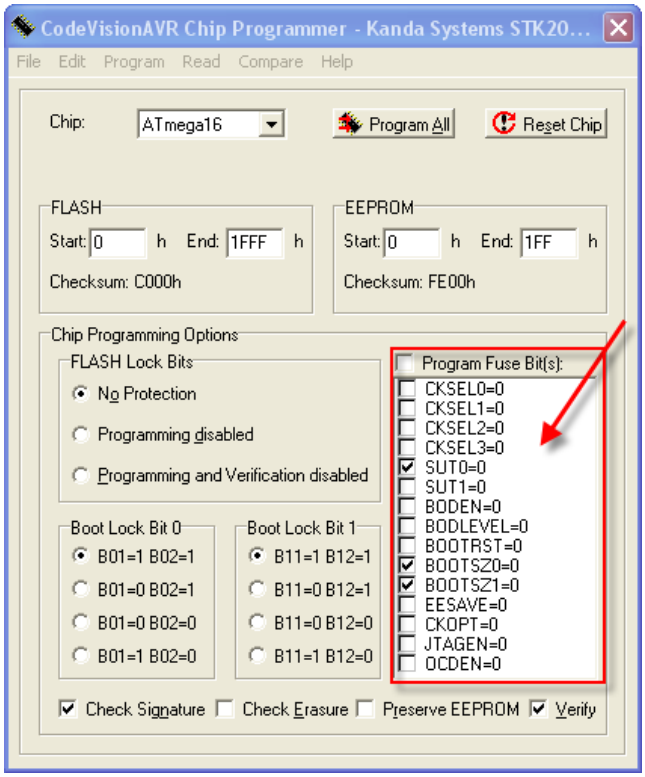
```
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;
```

```
//////////////////////////////////////////////////////
//      USART Receiver interrupt service routine        //
//////////////////////////////////////////////////////
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
  {
      if (data=='#')
      {
            rx_buffer[0]=data;
            rx_wr_index=1;
      }
      else if (rx_wr_index==1 && data=='!')
      {
            ////////////////////////
      }
      else if (rx_wr_index>=1 && rx_wr_index<=5)
      {
            rx_buffer[rx_wr_index]=data;
            rx_wr_index++;
            if (rx_wr_index==5)
            {
                  strcpy(rx_buffer1,rx_buffer);
                  for (i=0;i<4;i++)
                  {
                        #pragma warn-
                        str[i ]=rx_buffer1[i+1];
                        #pragma warn+
                  }
                  pwm=atoi(str);
                  OCR2=abs(pwm) ;
            }
      }else
            rx_wr_index=0;

      if (++rx_counter == RX_BUFFER_SIZE)
      {
            rx_counter=0;
            rx_buffer_overflow=1;
      };
  };
}
```

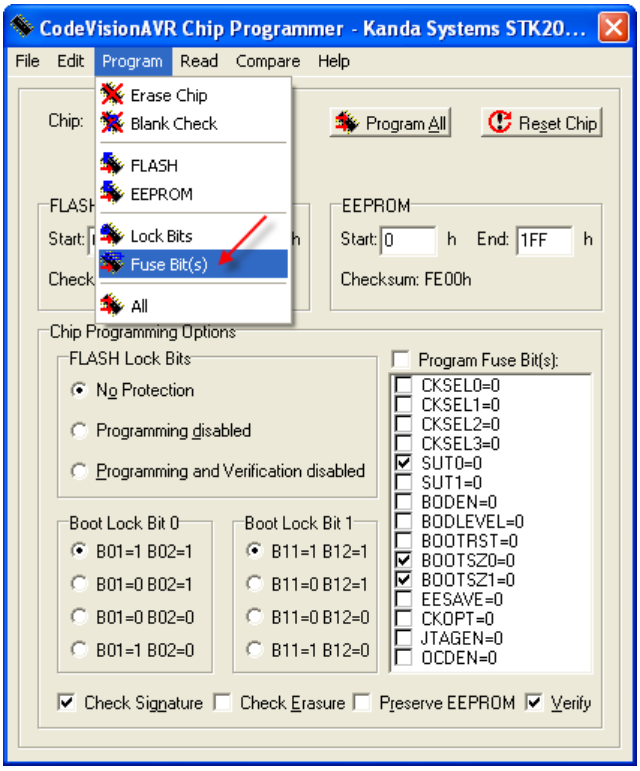
```
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif
```

نحوه پروگرم کردن میکروکنترلر:

بعد از اتمام مراحل برنامه نویسی در برنامه Codevision نوبت به پروگرم کردن میکروکنترلر میرسد.
در انجام این پروژه از یک پروگرمر STK 200/300 استفاده شده است و برای انجام عمل پروگرم از نرم افزار Codevision استفاده شده است.
نکته قابل توجه در این پروژه استفاده از کریستال خارجی میباشد که نیاز به تنظیم نمودن Fusebit های میکرو میباشد.
متناسب با دیتاشیت میکرو کنترلر ATMega16 فیوز بیت های میکرو میبایست مطابق شکل زیر برنامه ریزی شوند.



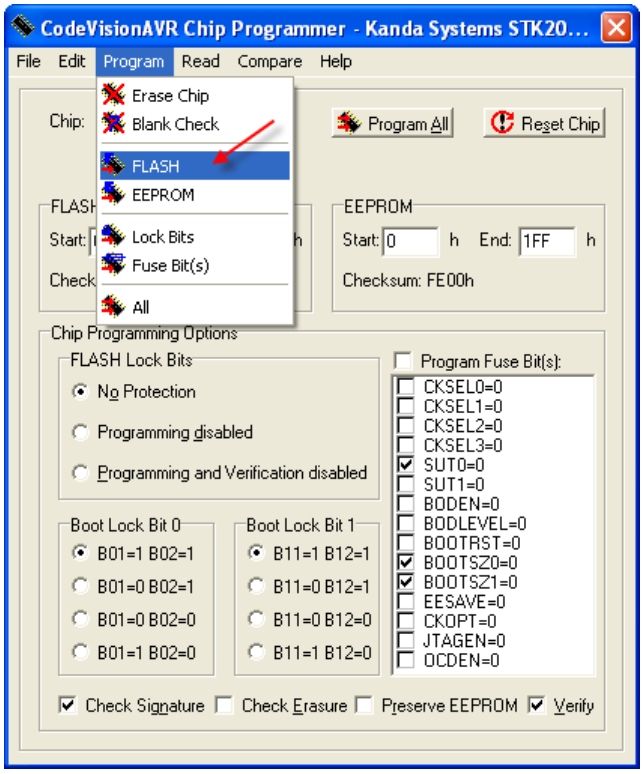
بعد از تنظیم Fuse Bit ها مطابق شکل از منوی Program گزینه Fuse Bit(s) را انتخاب میکنیم تا فیوز بیت های میکرو به طور دلخواه برنامه ریزی شوند.



سپس نوبت به پروگرم کردن Flash میکرو میرسد.

از منوی File گزینه Load Flash را انتخاب کرده و فایل HEX. ایجاد شده را انتخاب میکنیم .

سپس از منوی Program گزینه Flash را انتخاب کرده و منتظر پروگرم شدن میکرو میشویم .



کار پروگرم کردن میکرو در اینجا به پایان میرسد و میکروکنترلر قابلیت اجرای برنامه فوق را دارا می باشد.

طراحی برنامه در نرم افزار MATLAB :

در نسخه ۷ و بعد از آن DAQ Toolbox (Data Acquisition) بیش از پیش توسعه یافته است که به کمک آن میتوان به راحتی با بسیاری از مدارات جانبی ارتباط برقرار کرد که نتیجه آن ارائه مجموعه دستورات شیءگرا (Objective) می باشد و لذا کار با آنها در مقایسه با زبان های برنامه نویسی Basic , C و ... ساده تر است.
در نرم افزار MATLAB برای ارتباط با پورت سریال باید یک شیء تعریف گردد که وظیفه آن برقراری ارتباط است. پس از تعریف شیء و تنظیم مشخصات آن با باز کردن شیء میتوان عمل رد و بدل کردن اطلاعات را آغاز نمود. در پایان لازم است تا شیء مربوطه بسته شد و در صورت نیاز از محیط حذف شود.

با توجه به توضیحات فوق برای کنترل موتور می توان از برنامه زیر استفاده کرد :

```
clear;
new=input('RPS= ');
s=serial('COM1');
set(s,'baudrate',9600,'Terminator',13,'Timeout',1,'InputBufferSize',16,'OutputBufferSi
ze',8);
fopen(s);
while (1)
  a=fgets(s);
  speed=str2 num(a)
  pwm= new;
  fprintf(s,'#0%03 d',pwm);
end
clear;
fclose(s);
delete(s);
```

در برنامه فوق s=serial('COM1') یک شیء را برای برقراری ارتباط سریال با COM1 تعریف میکند و آن را در داخل متغیر S قرار میدهد و سپس به کمک تابع set ، مشخصات آن را تنظیم می کند که در آن baudrate نرخ ارسال، terminator آخرین کاراکتر از هر رشته است که پایان رشته را انتخاب میکند که Enter است. پارامتر Timeout نیز حداکثر زمانی را نشان می دهد که برنامه MATLAB برای دریافت اطلاعات از پورت سریال منتظر می ماند و در صورت عدم دریافت کاراکتر NULL را برمی گرداند. پارامتر های InputBuffer و OutputBuffer نیز

اندازه دو بافر نرم افزاری موجود در برنامه MATLAB را مشخص می کنند که اندازه آنها باید از ماکزیمم طول رشته ها بزرگتر باشد.

تابع fopen(s) شیء سریال را باز می کند که در این صورت امکان دریافت و ارسال اطلاعات از طریق ارتباط سریال فراهم می شود.

تابع fpintf() یک رشته کاراکتری را از طریق ارتباط سریال ارسال میکند.

تابع fget(s) یک رشته کاراکتری را از طریق ارتباط سریال دریافت میکند.

تابع str2 num() یک رشته کاراکتری را به عنوان ورودی می گیرد و آن را به یک عدد تبدیل می کند.

تابع fclose(s) شیء سریال را می بندد.

تابع delete(s) شیء سریال را حذف میکند.

سورس برنامه جهت کنترل دور موتور در AVR چنین خواهد بود:

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.5 Professional
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : DC MOTOR
Version : 1.0
Date    : 7/10/2007
Author  : AMIR
Company : www.ECA.ir
Comments:

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 12.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <delay.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x18 ;PORTB
#endasm
#include <lcd.h>

////////////////////////////////////////////////////////
//                  defines                           //
////////////////////////////////////////////////////////

#define motor_dir_pin0 PORTC.0
#define motor_dir_pin1 PORTC.1

#define motor_dir_ddr0 DDRC.0
#define motor_dir_ddr1 DDRC.1

#define right 0

#define left 1
////////////////////////////////////////////////////////
//                                                    //
////////////////////////////////////////////////////////
```

```
char rx_buffer1[8];
int speed_ctr=0;

char i,str[4];
int pwm;

char lcd_st[20];
unsigned int ctr=0;
//////////////////////////////////////////////////////////
//                  functions                           //
//////////////////////////////////////////////////////////

void motor_dir(unsigned char dir);
void micro_init(void);


//////////////////////////////////////////////////////////
//          External Interrupt 0 service routine       //
//////////////////////////////////////////////////////////

interrupt [EXT_INT0] void ext_int0_isr(void)
{
      speed_ctr++;

}

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

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
```

```
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

//////////////////////////////////////////////////////////
//      USART Receiver interrupt service routine        //
//////////////////////////////////////////////////////////

interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
  {
      if (data=='#')
      {
            rx_buffer[0]=data;
            rx_wr_index=1;
      }
      else if (rx_wr_index==1 && data=='!')
      {
            ////////////////////////
      }
      else if (rx_wr_index>=1 && rx_wr_index<=5)
      {
            rx_buffer[rx_wr_index]=data;
            rx_wr_index++;
            if (rx_wr_index==5)
            {

                  strcpy(rx_buffer1,rx_buffer);
                  for (i=0;i<4;i++)
                  {
                        #pragma warn-
                        str[i ]=rx_buffer1[i+1];
                        #pragma warn+
                  }
                  pwm=atoi(str);
                  OCR2=abs(pwm);
            }
      }else
            rx_wr_index=0;

      if (++rx_counter == RX_BUFFER_SIZE)
      {
            rx_counter=0;
            rx_buffer_overflow=1;
      };
  };
}
```

```
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

////////////////////////////////////////////////////////
//      Timer 1 overflow interrupt service routine      //
////////////////////////////////////////////////////////
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
      ctr++;
      if(ctr==4)
      {
            ctr=0;
            printf("%d\r",speed_ctr);
            speed_ctr=0;
      }
      TCNT1=28036;
}

////////////////////////////////////////////////////////
//                    main                            //
////////////////////////////////////////////////////////

void main(void)
{

micro_init();
motor_dir_ddr0=1;
motor_dir_ddr1=1;
motor_dir(right);

PORTC=0b00000101;

printf("\r\n START \r");
```

```
while (1)
    {
        lcd_clear();
        lcd_gotoxy(0,0);
        sprintf(lcd_str,"SPEED=%d RPS",speed_ctr);
        lcd_puts(lcd_str);
        lcd_gotoxy(0,1);
        lcd_putsf("**DC MOTOR**");
    };
}

//////////////////////////////////////////////////////////
//                  motor_dir                   //
//////////////////////////////////////////////////////////

void motor_dir(unsigned char dir)
{
    if(dir==0)
    {
        motor_dir_pin0=0;
        motor_dir_pin1=1;
    }
    else if(dir==1)
    {
        motor_dir_pin0=1;
        motor_dir_pin1=0;

    }
}
//////////////////////////////////////////////////////////
//                  micro_init                  //
//////////////////////////////////////////////////////////

void micro_init(void)
{

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

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=Out Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x80;

// 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: 187.500 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=0x03;
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: 11.719 kHz
// Mode: Fast PWM top=FFh
// OC2 output: Non-Inverted PWM
ASSR=0x00;
TCCR2=0x6F;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
```

```
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

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

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x4D;

// 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
#asm("sei")

}
```

شماتیک کلی مدار :