به نام خدا

آموزش میکروکنترلر AVR به زبان C (WinAVR)

(جلسه ششم)

مقدمه:

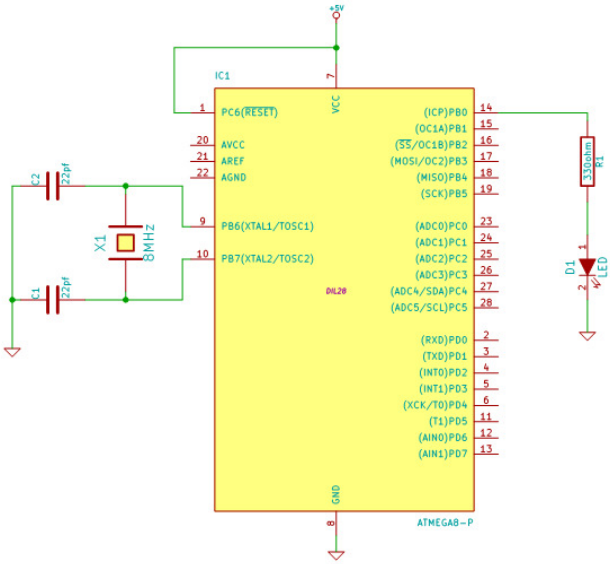
در جلسه قبل به معرفی مبدل آنالوگ به دیجیتال میکرو و نحوه کار با آن پرداختیم. در این جلسه تصمیم داریم به معرفی تایمرها و شمارنده ها (Timer/Counter) در میکروکنترلر AVR و چگونگی تنظیم و کار با آنها بپردازیم. تایمر بخشی در میکروکنترلر AVR است که در ساده ترین حالت برای ایجاد زمان دقیق مورد استفاده قرار می گیرد. هر میکروکنترلر بسته به نوع چندین تایمر داخلی داشته که برای مصارف خاص بکار می رود. از جمله کاربردهای دیگر تایمرها استفاده به عنوان شمارنده با اعمال پالس به پایه خاص میکرو، استفاده به عنوان مبدل موج PWM (برای ساخت ادواتی مانند کنترل دور موتور، کنترل نور لامپ، نوعی مبدل دیجیتال به آنالوگ، خروجی صوت برای پخش صدا مثل پروژه های wave player) و نیز استفاده به عنوان ساعت زمان واقعی (RTC)، کپچر سیگنال روی پایه خاص میکرو برای اندازه گیری زمان وقوع یا عرض پالس، تولید پالس خروجی با فرکانس دقیق، تولید مدولاسیون خاص ترکیبی مثل DTMF برای شمار گیری تلفن و نیز ایجاد مبنای زمان برای طراحی پروتکل های سریال نرم افزاری اشاره کرد. در این مقاله فقط به برخی از کاربردهای ساده و متداول تایمرها خواهیم پرداخت، بقیه موارد بدلیل تخصصی بودن و افزایش حجم مقاله بررسی نخواهد شد. سایر موارد را می توانید با بررسی توابع موجود در هدرفایل درایور تایمر کتابخانه avrlib یا هر درایور دیگر بررسی نمایید. همچنین مطالعه دیتاشیت (برگه اطلاعاتی) میکروکنترلر مربوطه نیز توصیه می شود، چرا که هر میکروکنترلر دارای امکانات متفاوتی بوده و ممکن است بخش هایی در آن حذف شده و یا بخش هایی

به آن اضافه شده باشد. همانطوریکه می دانید در این سری مقالات فقط استارت کار زده می شود، طی ادامه راه در صورت علاقه و یا نیاز بر عهده خوانندگان خواهد بود.

تایمر برای ایجاد پالس:

در میکروکنترلر نمونه ATmega8 سه تایمر با شماره های صفر، یک و دو قرار گرفته است. تایمر صفر و دو هشت بیتی بوده و تایمر یک ۱۶ بیتی می باشد. تایمر در ساده ترین کاربرد همانند شمارنده ای است که با اعمال پالس شمارش را آغاز کرده و پس از پر شدن یک وقفه نرم افزاری ایجاد می کند، با استفاده از این وقفه می توان تابعی را فراخوانی کرده و کاری را در میکرو انجام داد. تایمرها را علاوه بر اعمال پالس خارجی و یا کریستال ساعت می توان با فرکانس اصلی میکرو نیز تغذیه کرد البته این امکان وجود دارد که فرکانس میکرو را بر مقدار خاصی تقسیم کرده و سپس به تایمر وارد نمود. مقادیر تقسیم معمولاً اعداد خاصی می باشد که عموماً عبارتند از ۸، ۶۴، ۱۲۸، ۲۵۶، ۱۰۲۴. بطور مثال اگر میکرو با کریستال ۸ مگاهرتز راه اندازی شده باشد در صورتی که فرکانس را تقسیم نکنیم و مستقیماً تایمر را راه اندازی نماییم پس از شمارش ۲۵۶ پالس (برای تایمر ۸ بیتی صفر) یک وقفه اتفاق می افتد و با محاسبه فرکانس میکرو یعنی ۸۰۰۰۰۰۰ هرتز تقسیم بر ۲۵۶ (تایمر ۸ بیتی) عدد ۳۱۲۵۰ بدست می آید، و یا در صورتی که تایمر شماره ۲ با کریستال ساعت 32.768KHz که برروی پایه های TOSC1 و TOSC2 قرار داده شده راه اندازی شود و مقدار تقسیم برابر ۱۲۸ تنظیم گردد در این صورت دقیقاً هر ثانیه یک وقفه روی می دهد که می توان از آن برای ساخت یک ساعت یا تایمر دیجیتال استفاده کرد. در صورتی که تایمر با فرکانس داخل میکرو (فرکانس CPU) تغذیه می گردد اکیداً توصیه می شود که از کریستال خارجی استفاده شده و خازن های 22PF نیز بین پایه های اسیلاتور و زمین قرار داده شود. (اسیلاتور داخلی RC دقیق نبوده و نسبت به تغییرات دما حساس می باشد و فقط برای پروژه های بی نیاز از زمان مانند راه اندازی LCD و غیره کاربرد دارد).

در این پروژه ابتدا یک چشمکزن با استفاده از تایمر ۱۶ بیتی شماره یک میکروکنترلر ATmega8 طراحی می کنیم که با فرکانس نوسان ساز داخلی میکرو که روی ۸ مگاهرتز قرار داده شده کار میکند و با فرکانس حدود ۱/۹ هرتز LED را روشن و خاموش می نماید. برای شروع شماتیک زیر را بر روی بِرد بُرد ببندید: (نظر به اینکه در این پروژه از نوسان ساز داخلی استفاده کرده ایم می توانید کریستال و خازن ها را از شماتیک زیر حذف کنید).



برنامه:

برنامه این جلسه به طور کامل در پوشه مقاله آورده شده با این حال در این قسمت توضیح داده می شود. برای برنامه نویسی تایمر ابتدا پوشه ای با نام دلخواه ایجاد کرده و پس از ساختن Makefile و تنظیم میکرو به ATmega8 و تنظیم فرکانس CPU بر روی 8000000 فایل مورد نظر را در پوشه برنامه ذخیره می نماییم. در مرحله بعدی دو فایل درایور تایمر یعنی فایلهای timer.c و timer.h را از پوشه avrlib در پوشه برنامه خود کپی می کنیم. همچنین فایل global.h را از پوشه avrlib/conf کپی کرده و در پوشه برنامه paste می نماییم. در آخر فایلهای avrlibdefs.h و avrlibtypes.h را نیز از پوشه avrlib در پوشه برنامه کپی می کنیم.

در مرحله بعد وارد پوشه برنامه خود شده، Makefile را با Notepad++ باز میکنیم و timer.c را به خطی که با SRC شروع شده می افزاییم:

```
81
82  # List C source files here. (C dependencies are automatically generated.)
83  SRC = $(TARGET).c timer.c
84
```

سپس فایل global.h را باز کرده و مانند شکل زیر فقط خط 8000000 را از حالت کامنت خارج می کنیم تا فرکانس CPU را بر روی ۸ مگاهرتز قرار دهیم. (این کار مجدداً تنظیمات Makefile را Over Write می کند:

```
33  //#define F_CPU     14745000              // 14.745MHz processor
34  #define F_CPU       8000000                     // 8MHz processor
35  //#define F_CPU     7372800               // 7.37MHz processor
```

در مرحله بعد فایل timer.c را باز کرده و خط rprintf.h را کامنت یا پاک می کنیم:

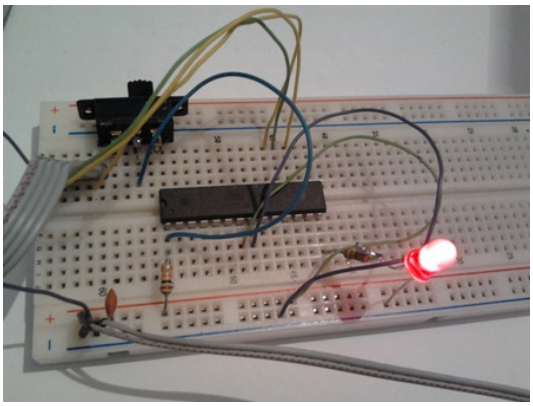
```
25
26  //#include "rprintf.h"
27
```

توجه داشته باشید که تمام فایلهای بالا به غیر از Makefile جزئی از کتابخانه avrlib نوشته Pascal Stang بوده و اساس آموزش های ما قرار گرفته است. در صورتی که بخواهید مستقیماً وارد عمل شده و تایمرها را راه اندازی نمایید می توانید بدون نیاز به درایور و هدر نویسی به طور مستقیم در تابع main فایل اصلی رجیسترهای تایمرها را مقدار دهی کرده و با آنها کار کنید، به هر روی کتابخانه avrlib بسیار زیبا و دقیق نوشته شده و با بررسی تمام هدرفایلها به ریزبینی جناب پروفسور استنگ پی خواهید برد، جای تعجب است که چرا این پروژه نیمه کاره رها شده و پشتیبانی نمی شود. به جرات می توان گفت کامل ترین و تمیز ترین (از نظر کدنویسی) کتابخانه در زمینه avrgcc همین کتابخانه avrlib است.

در مرحله آخر ادیتور WinAVR را باز کرده و کدهای زیر را در آن وارد می کنیم:

main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "timer.h"
#include <util/delay.h>

void myovf(void);

int main(void){

    DDRB = 255;
    timerInit();
    timerAttach(TIMER1OVERFLOW_INT,myovf);

    while(1){

    }

    return 0;
}


void myovf(void){
    PORTB = 255;
    _delay_ms(50);
    PORTB = 0;
}

```



در برنامه بالا کتابخانه های io.h، interrupt.h و delay.h از کتابخانه استاندارد winavr را وارد کرده و سپس کتابخانه timer.h از کتابخانه شخص ثالث پروفسور استنگ را اینکلود نموده ایم. سپس تعریف تابع myovf را در ابتدای برنامه ذکر کرده ایم. در زبان C هرگاه بخواهیم تابعی به غیر از main در برنامه بنویسیم بایستی بدنه تابع را بعد از main نوشته و خط اول آن را قبل از main تکرار کنیم تا کامپایلر برای تابع ما حافظه رزرو کند و در واقع آنرا بشناسد. تابع myovf با هر بار اجرا یک لحظه LED متصل شده به هر یک از پایه های PORTB را روشن می کند و در واقع با اجرای تابع، LED فلش می زند. در داخل تابع main ابتدا تمام پایه های PORTB را به صورت خروجی تعریف کرده ایم و سپس تابع timerInit() از کتابخانه timer را فراخوانی کرده ایم. با بررسی این تابع در کتابخانه timer.c مشاهده میکنیم که کار آن مقداردهی اولیه به سه تایمر صفر، ۱ و ۲، تنظیم مقدار پیش فرض تقسیم کننده فرکانس با توجه به هدرفایل timer.h برای هر سه تایمر و در نهایت فعال سازی وقفه سراسری میکرو است. در واقع می توان گفت با اجرای این تابع هر سه تایمر شروع به کار می کنند. دستور بعدی تابعی است که دو آرگومان می پذیرد. آرگومان اول این تابع شماره وقفه ایجاد شده و آرگومان دوم تابعی است که موقع ایجاد وقفه بایستی اجرا شود. در واقع کار این تابع اتصال تابع کاربر به وقفه مورد نظر میکرو (تایمر) است. در اینجا برای آرگومان اول از ثابت تعریف شده در هدرفایل timer.h که به معنای وقفه سرریزی تایمر ۱ است استفاده کرده ایم و آرگومان دوم را نیز تابع myovf که با هر اجرا LED را وادار به فلش زدن می کند تنظیم کرده ایم. سپس یک حلقه بی انتهای while ایجاد کرده و برنامه را در آن رها نموده ایم. در این صورت با اجرای برنامه طبق تنظیمات پیش فرض در هدرفایل timer.h تایمر شماره یک با پرسکالر ۶۴ فرکانس CPU را دریافت کرده و پس از هر بار پر شدن (۶۵۵۳۶ پله) یک سیگنال سرریزی ایجاد می کند. این سیگنال توسط درایور timer.c دریافت شده و با توجه به تابعی که به این سیگنال اتصال داده ایم تابع مورد نظر ما را فراخوانی می کند که در نتیجه سبب می شود LED به طور تقریبی هر ثانیه ۲ بار چشمک بزند. ۸۰۰۰۰۰۰ تقسیم بر ۶۴ برابر است با ۱۲۵۰۰۰ و این عدد بخش بر ۶۵۵۳۶ تقریباً مساوی ۲ می باشد. (۲هرتز).

درایورهای avrlib معمولاً کمی شلوغ بوده و مملو از تابع و ثابت های بیشمار می باشد. همچنین بیشتر این کتابخانه ها به یکدیگر وابسته بوده و برای اجرای یک پروژه کوچک مجبوریم چندین هدرفایل را در پوشه برنامه خود قرار دهیم. این در صورتی است که راه اندازی یک تایمر برای پروژه ساده ای مثل پروژه فوق نیازی به این همه هدرفایل نداشته و تنها با مقدار دهی چند رجیستر کوچک انجام می پذیرد. با بررسی درایور timer.c مشاهده می کنیم که بطور مثال رجیسترهای TCNT0 و TIMSK برای تایمر شماره صفر مقدار دهی شده اند. این کار را میتوان توسط هدرفایل های شخصی و حتی توسط تابعی در کنار تابع main انجام داد.

در اینجا تقریباً کار ما با بخش های داخلی میکرو پایان می پذیرد، جلسه بعدی به نحوه ارتباط با حافظه SD/MMC اختصاص داشته و یاد میگیریم که چگونه اطلاعات را در مموی کارت بنوسیم و از آن بخوانیم و با مموری کارت فقط به عنوان یک حافظه برخورد می کنیم. در جلسه بعد از آن با مفهوم فایل آشنا شده و یاد میگیریم که چگونه اطلاعات را بر اساس قوانین خاص و جدول بندی FAT در مموری کارت بنویسیم که هم توسط میکرو و هم توسط کامپیوتر و سایر دستگاه های دیجیتال قابل فهم باشد. در جلسه نهم هم دوباره گریزی به میکرو زده و برخی از مباحث بررسی نشده از قبیل وقفه های خارجی، خواندن کلید، کی پد و نیز نحوه استفاده از حافظه دائمی میکرو (EEPROM) و همچنین برخی از ترفندهای C را خاطرنشان خواهیم کرد. جلسه دهم از ابتدا وجود نداشت ولی برای اینکه ۹ جلسه آموزشی کمی ناتمام به نظر می آید جلسه دهم را نیز به مجموعه جلسات افزودیم و قرار است در آن چند پروژه عملی با C بنویسیم و کاربرد آن را نمایش دهیم.

ادامه دارد...

منابع:

پرتوی فر، محمد مهدی. مظاهریان، فرزاد. بیانلو، یوسف. (۱۳۹۱). مرجع کامل میکروکنترلرهای AVR، انتشارات نص.

(از منبع بالا فقط برای نوشتن مقدمه و کاربردهای تایمر استفاده شده است، در هر صورت مرجع کاملی بوده و خواندن آن توصیه می شود)

پایان جلسه ششم آموزش AVR

مولف: بهنام زکی زاده

www.avr64.com

۱۵ فروردین ۱۳۹۳