به نام خدا

(جلسه دهم آموزش AVR)

(Assembly 5)

مقدمه:

در جلسه قبل به نحوه عملکرد تایمرها و پورت سریال پرداختیم. در این جلسه با مبدل آنالوگ به دیجیتال داخلی میکرو و نیز حافظه دائمی E^2Prom آشنا می شویم. مبدل آنالوگ به دیجیتال به اختصار A2D نیز نامیده می شود و معمولا برای ساخت وسایلی نظیر دماسنج، ولتمتر، رطوبت سنج و غیره به کار می رود. ای تو پرام نیز بخشی از حافظه دائمی میکرو می باشد که بر خلاف SRAM با قطع برق اطلاعات آن پاک نمی شود و معمولاً برای ذخیره Setting های سیستم (مثلاً حداکثر دمای قابل تنظیم برای عملکرد یک رله در یک پروژه دماسنج دیجیتالی و یا Pin Code امنیتی در یک قفل دیجیتالی) به کار می رود. همانطوریکه می دانید این جلسه آخرین جلسه اسمبلی می باشد. مباحث اسمبلی واقعاً گسترده تر از مطالب ذکر شده در این جلسات هستند ولی هدف اصلی این مقالات فقط معرفی زبان ها به صورت فهرست وار است.

مبدل آنالوگ به دیجیتال:

در داخل هر میکروکنترلر AVR یک مبدل آنالوگ به دیجیتال مرکزی با دقت ۱۰ بیت قرار گرفته است که با کمک مالتی پلکسر معمولاً ۸ ورودی به آن اختصاص داده می شود. مبدل آنالوگ به دیجیتال میکرو چند رجیستر مختلف برای تنظیم و ذخیره مقدار خروجی دارد که رجیستر ADMUX برای تنظیم ولتاژ مرجع، تعیین کانال مورد استفاده، تقسیم بر ۴ (شیفت به چپ) و ... و رجیستر ADCSRA برای فعال سازی ADC و شروع به کار آن و نیز تعیین فرکانس نمونه برداری به کار می رود. حاصل نمونه برداری در دو رجیستر ADCL و ADCH ذخیره می شود.

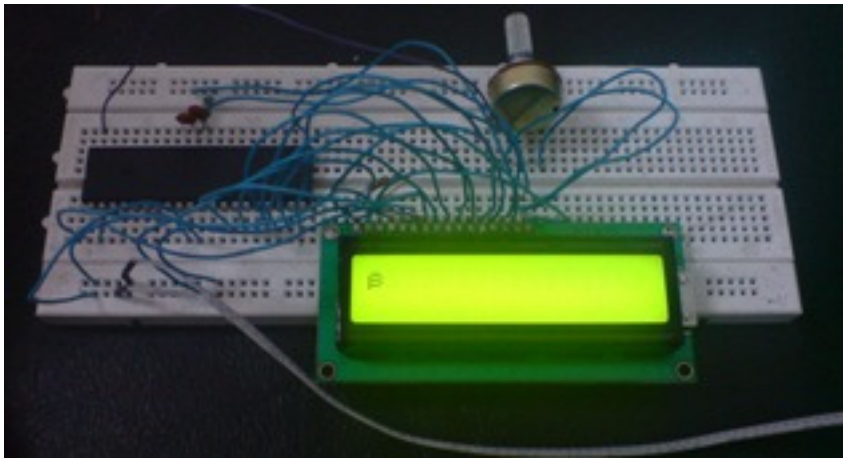
در نمونه برنامه زیر پایه ADC0 میکرو (کانال ۰) به سر وسط یک ولوم ۵ کیلو متصل شده و دو سر این ولوم به قطب های + و – تغذیه وصل می باشند. مقدار ADCH خوانده شده و کد اسکی مربوطه بر روی LCD به صورت کاراکتر نمایش داده می شود. (برای نمایش عدد ۳ یا ۴ رقمی متناظر بر روی نمایشگر بایستی مقدار هر دو رجیستر ACD را خوانده و با تقسیم مقدار یکان و دهگان و صدگان و... را به دست آوریم و متناسب با هر عدد کاراکتر اسکی عدد مربوطه را بر روی نمایشگر نشان دهیم. مثلاً کاراکتر اسکی ۰ عدد ۴۸ می باشد که بایستی عدد حاصله را با ۴۸ جمع نموده و سپس بر روی نمایشگر بیاوریم؛ عملیات مذبور با توجه به وجود کامپایلرهای قوی که تمام این چند صد خط را با تک دستور **LCD Getadc(0)** انجام می دهند کمی خسته کننده به نظر می رسد!).

```
;ADC prog by Behnam Zakizadeh @ 05.Aug.2011 (AVR64.com)

.include "Appnotes/m16def.inc"
.cseg
.org 0000
rjmp start

.org 0x30
start:
ldi             r20,byte1(RAMEND)
out             SPL,r20
ldi             r20,byte2(RAMEND)
out             SPH,r20


call setadc


call delay ;wait for lcd init

ldi r20,0xff
out ddrd, r20
sbi ddrb,0
sbi ddrb,1
```

```
cbi portb,0 ;Instruction

sbi portb,1

ldi r20, 0x38
out portd, r20
cbi portb,1
sbi portb,1
call delay

ldi r20, 0x0c
out portd, r20
cbi portb,1
sbi portb,1
call delay

ldi r20, 0x01
out portd, r20
cbi portb,1
sbi portb,1
call delay

;ldi r20, 0x06 ; Auto move cursor right
ldi r20, 0x02 ; Stay cursor in Home
out portd, r20
cbi portb,1
sbi portb,1
call delay

sbi portb,0 ;Data

ldi r22,'T'
call show
ldi r22,'e'
call show
ldi r22,'s'
call show
ldi r22,'t'
call show

getadc:
sbic ADCSRA, ADSC
rjmp getadc
in r22,ADCH
ldi r21, (1<<ADEN) ; switch ADC off
out ADCSRA, r21
call Clear
call show
call delay
call delay
call delay
call delay
call delay
call setadc
rjmp getadc

end:
rjmp end

delay:
ldi r20,0x40
```

```
loop2: ldi r21,0xff
loop3: dec r21
  brne loop3
  dec r20
  brne loop2
ret

show:
out portd, r22
cbi portb,1
sbi portb,1
call delay
ret

Clear:
cbi portb,0 ;Instruction
ldi r20, 0x01
out portd, r20
cbi portb,1
sbi portb,1
call delay

;ldi r20, 0x06 ; Auto move cursor right
ldi r20, 0x02 ; Stay cursor in Home
out portd, r20
cbi portb,1
sbi portb,1
call delay

sbi portb,0 ;Data

ret

setadc:

ldi r20, 0b11100000
out ADMUX, r20 ; Vref = Internal 2.56 v, Result/4 (1024/4=Max:256) , ADC0 Selected

ldi r20, 0b11000000
out ADCSRA, r20 ; Enable ADC, Start ADC, Prescaler= CPU/2

ret
```

تصویر خروجی: (اجرا شده روی سخت افزار جلسات قبلی اسمبلی)



حافظه دائمی EEPROM:

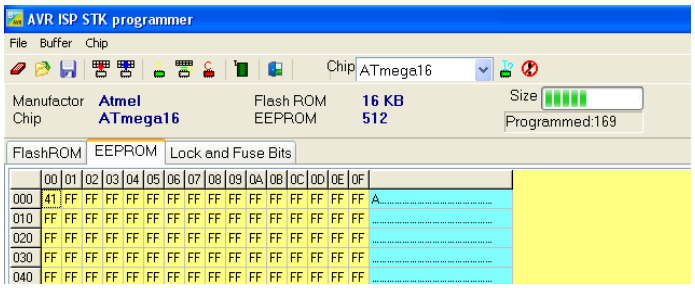
یکی از امکانات مفید AVR حافظه دائمی E²Prom می باشد که قابلیت ذخیره سازی اطلاعات و نگهداری آنها با قطع برق را دارا می باشد. سرعت حافظه EEPROM پایین بوده و با نوشتن و خواندن زیاد به سرعت خراب می شود، برای همین منظور بایستی برنامه را طوری بنویسیم که فقط موقع روشن شدن میکرو متغیر های EEPROM در متغیر های معمولی و یا رجیستر ها ذخیره شوند و فقط در مواقع لازم آپدیت گردند. EEPROM برای میکرو همانند Database برای یک برنامه کامپیوتری و یا یک وب سایت می باشد که فقط در مواقع لازم بایستی با آن ارتباط برقرار کرد و به دلیل سرعت پایین و خراب شدن سریع (در مورد EEPROM) به هیچ وجه نباید از متغیرهای EEPROM به طور مستقیم در برنامه و حلقه ها و شرط ها استفاده کرد و حتی گاهی این استفاده مستقیم باعث ایجاد خطا و یا پایین آوردن سرعت برنامه خواهد شد و اگر این وضعیت در حلقه ای ۱۰ میلیون بار تکرار شود، سلول EEPROM به کلی نابود می شود. EEPROM از یک منظر شبیه هارد کامپیوتر می باشد که اطلاعات برای اجرا بایستی از آن به RAM منتقل شود.

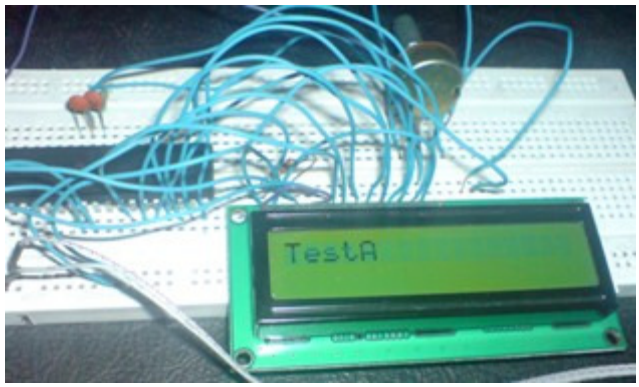
در نمونه برنامه زیر با نحوه کار با متغیرهای EEPROM آشنا می شویم، در این برنامه در ابتدا مقدار آدرس ۰ حافظه ایپرام خوانده و در نمایشگر نشان داده می شود. سپس کاراکتر A در همان آدرس ذخیره می شود. با توجه به اینکه در اولین راه اندازی مقدار این خانه برابر با ۲۵۵ است فقط در نمایشگر عبارت Test را می بینیم، ولی در راه اندازی های بعدی عبارت TestA دیده می شود. شایان ذکر است با فلش کردن میکرو به طور پیشفرض حافظه EEPROM نیز پاک می شود ولی می توان فیوزبیت مخصوص این کار را طوری تنظیم کرد که با فلش کردن میکرو اطلاعات EEPROM پاک نشود.

```
;EEPROM prog by Behnam Zakizadeh @ 05.Aug.2011 (AVR64.com)

.include "Appnotes/m16def.inc"
.cseg
.org 0000
rjmp start

.org 0x30
start:
ldi             r20,byte1(RAMEND)
out             SPL,r20
ldi             r20,byte2(RAMEND)
out             SPH,r20


call delay ;wait for lcd init

ldi r20,0xff
```

```
out ddrd, r20
sbi ddrb,0
sbi ddrb,1

cbi portb,0 ;Instruction

sbi portb,1

ldi r20, 0x38
out portd, r20
cbi portb,1
sbi portb,1
call delay

ldi r20, 0x0c
out portd, r20
cbi portb,1
sbi portb,1
call delay

ldi r20, 0x01
out portd, r20
cbi portb,1
sbi portb,1
call delay

ldi r20, 0x06 ; Auto move cursor right
;ldi r20, 0x02 ; Stay cursor in Home
out portd, r20
cbi portb,1
sbi portb,1
call delay

sbi portb,0 ;Data

ldi r22,'T'
call show
ldi r22,'e'
call show
ldi r22,'s'
call show
ldi r22,'t'
call show

;Address of EEPROM Variable (0)
ldi R20, 0
ldi R21, 0

read_mem:
sbic EECR, EEWE
rjmp read_mem
out EEARH, R20
out EEARL, R21

SBI EECR, EERE
in R22, EEDR
call show

write_mem:
sbic EECR, EEWE
rjmp write_mem
out EEARH, R20
```

```
out EEARL, R21
ldi R22, 'A'
out EEDR, R22
CLI
SBI EECR, EEMWE
SBI EECR, EEWE
SEI

end:
rjmp end

delay:
ldi r20,0x40
loop2: ldi r21,0xff
loop3: dec r21
  brne loop3
  dec r20
  brne loop2
ret

show:
out portd, r22
cbi portb,1
sbi portb,1
call delay
ret

Clear:
cbi portb,0 ;Instruction
ldi r20, 0x01
out portd, r20
cbi portb,1
sbi portb,1
call delay

;ldi r20, 0x06 ; Auto move cursor right
ldi r20, 0x02 ; Stay cursor in Home
out portd, r20
cbi portb,1
sbi portb,1
call delay

sbi portb,0 ;Data

ret
```





در این برنامه موقع خواندن اطلاعات بیت EEWE از رجیستر EECR بررسی می شود و در صورتی که هنوز 1 باشد صبر می کنیم تا عملیات نوشتن قبلی تمام شود (فرض می کنیم که در حال نوشتن آرایه ای از سلول های حافظه هستیم). سپس آدرس محل ذخیره سازی به رجیسترهای EEARH و EEARL داده شده و با 1 کردن بیت EERE از رجیستر EECR آدرس در ثبات های مذکور نوشته می شود. سپس مقدار اطلاعات از رجیستر EEDR خوانده و در R22 نوشته می شود. موقع نوشتن نیز عملیات به همین منوال بوده و فقط قبل از عملیات اصلی نوشتن، تمام وقفه های سراسری و عملیات boot Loader غیر فعال می شود.

با اتمام این جلسه مباحث اسمبلی نیز تقریباً تمام می شود. البته زبان C در واقع کنترلی بر دستورات اسمبلی است و فقط تعدادی ساختار شرطی و تصمیم گیری برای کد نویسی راحت تر ارائه می دهد. C استاندارد فاقد توابع مخصوص AVR است. برخی از زبان های گسترش یافته تجارتی از قبیل کدویژن توابعی را به صورت پیش ساخته ارائه می دهند که برخی از کارها را راحت تر می کند. توابعی مثل printf برای نمایش اطلاعات در LCD و یا پورت سریال. C استاندارد فقط از ساختارهای if، Switch، While و از این قبیل تشکیل شده و برای نوشتن هر تابعی بایستی برنامه مورد نظر را به زبان اسمبلی و با کمک ساختارهای C به صورت هدرفایلهایی با پسوند .h تهیه کرد و سپس آنها را به برنامه اصلی پیوند زد. جلسه بعدی به کدویژن اختصاص دارد و از آنجاییکه این برنامه یک کامپایلر تجاری می باشد فقط یک جلسه را به آن اختصاص می دهیم و ۹ جلسه آخر را به وین ای وی آر می پردازیم. فرض ما اینست که خواننده با زبان C آشنایی قبلی داشته باشد. در صورتی که با زبان C آشنایی نداشته باشید می توانید کامپایلر رایگان Dev C++ را که نسخه گرافیکی کامپایلر خط فرمانی GCC می باشد از اینترنت بارگیری کرده و با سرچ مقالات آموزشی C به مطالعه و تمرین با این زبان و نوشتن برنامه های تحت ویندوز بپردازید. با یک ماه تمرین و مطالعه، تسلط کافی برای ادامه جلسات آموزشی AVR پیدا خواهید کرد.

منابع مورد استفاده در تدوین این مقاله:

[1] دیتا شیت میکروکنترلر ATmega16

[2] شهریاری شیرزاد، مرجع علمی کاربردی میکروکنترلرهای AVR, PIC, MCS-51 انتشارات پرتونگار، ۱۳۸۵

[3] مکنزی اسکات، میکروکنترلر 8051، انتشارات ناقوس – انتشارات میرزایی، ۱۳۸۴

[4] http://www.rcgroups.com/forums/showthread.php?t=864307

ادامه دارد...

(آخرین جلسه اسمبلی)

آموزش C از جلسه بعد...

پایان جلسه پنجم اسمبلی (جلسه دهم آموزش AVR)

مولف: بهنام زکی زاده

www.avr64.com

۱۴ مرداد ۱۳۹۰

آخرین ویرایش ۱۴ مهر ۱۳۹۲ ✓