



دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

## دستور کار آزمایشگاه ریز پردازنده دستور کار آزمایشگاه طراحی سیستم های دیجیتال 2

(مبتنی بر ریز پردازنده ATMEGA16/32)

تهیه کننده:

زهرا محمدزاده

بررسی کننده:

دکتر امیر خورسندی

1401 دیماه

## فهرست

1 .....	مقدمه
3 .....	معرفی بورد آزمایشگاه
5 .....	1 جلسه اول
5 .....	1.1 هدف
5 .....	1.2 معرفی درگاههای ورودی و خروجی Atmega16/32
7 .....	1.3 معرفی فایل سرآیند (Header File)
8 .....	1.3.1 فایل سرآیند mega16.h
8 .....	1.3.2 معرفی فایل سرآیند delay.h
9 .....	1.4 معرفی مقاومت داخلی درگاهها
9 .....	1.5 معرفی واحدهای ورودی/خروجی اولیه پکیج آموزشی
9 .....	1.5.1 معرفی کلیدهای کشویی
10 .....	1.5.2 معرفی Push Button
10 .....	1.5.3 معرفی نمایشگر 7-Segment
11 .....	1.6 ایجاد پروژه در محیط codevision
11 .....	1.6.1 ایجاد پروژه بدون استفاده از CodeWizard
13 .....	1.6.2 ایجاد پروژه با استفاده از CodeWizard
14 .....	1.7 نحوه پروگرام نمودن ریزپردازنده
17 .....	1.8 برنامههای کاربردی
19 .....	1.9 زیربرنامه نویسی و فراخوانی آن در تابع اصلی
20 .....	1.10 سوالات تشریحی درگاههای ورودی و خروجی

21 .....	1.1 برنامه‌های اجرایی در گاههای ورودی و خروجی:	1.1
23 .....	2 جلسه دوم	
23 .....	2.1 هدف	
23 .....	2.2 مقدمه	
23 .....	2.3 ایجاد زیربرنامه با آرگومان ورودی و خروجی	
27 .....	2.4 ایجاد فایل جانبی با قالب .c	
27 .....	2.5 ایجاد فایل جانبی با قالب .h	
28 .....	2.6 ایجاد فایل سرآیند برای کل پروژه	
29 .....	2.7 تعریف متغیر سراسری	
29 .....	2.8 برنامه‌های اجرایی مربوط به زیربرنامه‌نویسی و فایل‌های جانبی	
32 .....	3 جلسه سوم	
32 .....	3.1 هدف	
32 .....	3.2 معرفی LCD کارکتری	
32 .....	3.2.1 آشنایی با مدار راهانداز، پایه‌ها و حالت‌های کاری	
33 .....	3.2.2 ارتباط 4 سیمه با ریزپردازنده	
36 .....	3.2.3 آشنایی با فایل سرآیند «alcd.h»	
36 .....	3.3 آشنایی با صفحه کلید ماتریسی	
39 .....	3.4 وقفه‌ها در ریزپردازنده	
39 .....	3.4.1 راه اندازی وقفه در AVR	
42 .....	3.5 پردازش صفحه کلید با وقفه	
43 .....	3.6 برنامه‌های اجرایی مبحث LCD و صفحه کلید	
45 .....	4 جلسه چهارم	
45 .....	4.1 هدف	

45	.....	4.2 مقدمه
46	.....	4.3 ثبات‌های تایمیر
50	.....	4.4 تنظیمات CodeWizard برای تایمیر
51	.....	4.5 حالت‌های کاری تایمیر
51	.....	4.5.1 حالت Normal
52	.....	4.5.2 حالت CTC
53	.....	4.5.3 حالت Fast PWM
54	.....	4.5.4 حالت Phase Correct PWM
55	.....	4.6 مثال کاربردی - طراحی ثانیه شمار
57	.....	4.7 محاسبه خطای
60	.....	4.8 برنامه‌های اجرایی مبحث تایمیرها
62	.....	5 جلسه پنجم
62	.....	5.1 هدف
62	.....	5.2 مقدمه
62	.....	5.3 آشنایی با رله
64	.....	5.4 آشنایی با موتور DC
65	.....	5.5 آشنایی با انکودر
66	.....	5.6 اندازه گیری دور موتور DC
67	.....	5.7 موتور پله‌ای
69	.....	5.8 برنامه‌های اجرایی مبحث موتورها
72	.....	6 جلسه ششم
72	.....	6.1 هدف
72	.....	6.2 مقدمه

73 .....	6.3 حسگرها
74 .....	6.4 مبدل ADC
75 .....	6.5 ثبات‌های مبدل آنالوگ به دیجیتال
75 .....	6.5.1 ثبات کنترلی ADMUX
77 .....	6.5.2 ثبات ADCSRA
78 .....	6.5.3 ثبات داده (ADCH, ADCL) ADCW
78 .....	6.5.4 ثبات SFIOR
79 .....	6.5.5 معرفی و قله ADC
79 .....	6.6 مراحل برنامه‌نویسی ADC
81 .....	6.6.1 برنامه خواندن داده‌های مبدل دیجیتال
82 .....	6.7 اندازه‌گیری دما
85 .....	6.8 آشنایی با تبدیل سیگنال دیجیتال به آنالوگ
87 .....	6.9 نمونه برنامه مبدل آنالوگ به دیجیتال
88 .....	6.10 برنامه‌های اجرایی مبدل‌های آنالوگ به دیجیتال
90 .....	7 جلسه هفتم
90 .....	7.1 هدف
90 .....	7.2 مقدمه
91 .....	7.3 معرفی ارتباط USART (RS232)
92 .....	7.4 ارتباط Atmega16/32 و کامپیوتر
93 .....	7.5 ثبات‌های ارتباط USART
93 .....	7.5.1 ثبات UDR
94 .....	7.5.2 ثبات UCSRA
95 .....	7.5.3 ثبات UCSRB

95 .....	UCSRC ثبات ..... 7.5.4
96 .....	UBRRH و UBRRL ثبات‌های (USART Baud rate Register) ..... 7.5.5
98 .....	stdio.h کتابخانه ..... 7.6
99 .....	برنامه نویسی در محیط Codevision برای USART ..... 7.7
99 .....	وقدنهای بخش UART ..... 7.7.1
104 .....	وقنهی RX ..... 7.7.2
104 .....	وقنهی TX ..... 7.7.3
105 .....	تغییر ماهیت توابع putchar و getchar ..... 7.7.4
105 .....	نرم افزار Teraterm ..... 7.8
107 .....	برقراری ارتباط سریال در نرمافزار MATLAB ..... 7.9
110 .....	برنامه‌های اجرایی ارتباط سریال ..... 7.10
112 .....	جلسه هشتم ..... 8
112 .....	هدف ..... 8.1
112 .....	نمایشگر Dot-Matrix ..... 8.2
113 .....	ایجاد قلم برای نمایش متون روی Dot-Matrix ..... 8.2.1
115 .....	اصول کلی نمایش متن بر روی DotMatrix ..... 8.2.2
116 .....	استفاده از چند ماژول DotMatrix ..... 8.2.3
117 .....	LCD گرافیکی ..... 8.2.4
119 .....	آشنایی با پایه‌های GLCD ..... 8.2.5
120 .....	برنامه نویسی GLCD در محیط کد ویژن ..... 8.3
120 .....	glcd.h فایل سرآیند ..... 8.4
121 .....	نمونه برنامه GLCD ..... 8.5
121 .....	قلم‌های نمایش کاراکتر ..... 8.6

122.....	8.7 نمایش تصویر روی GLCD
122.....	8.7.1 تهیه کدهای عکس با استفاده از نرمافزار LCD Vision
123.....	8.7.2 برنامه نمایش عکس در محیط Codevision
123.....	8.8 برنامه‌های اجرایی مبحث GLCD
125.....	9 جلسه نهم
125.....	9.1 هدف
125.....	9.2 مقدمه
126.....	9.3 ثبات‌های SPI
127.....	9.3.1 SPDR (SPI Data Register)
127.....	9.3.2 SPSR (SPI Status Register)
127.....	9.3.3 SPCR (SPI Control Register)
128.....	9.4 پیکربندی SPI در Codevision
129.....	9.5 سناریوهای مختلف ارتباط SPI
131.....	9.6 برنامه‌های کاربردی SPI
143.....	9.7 مقایسه ارتباط سریال SPI و UART
143.....	9.7.1 UART
144.....	9.7.2 SPI
144.....	9.7.3 برنامه‌های اجرایی مبحث SPI
146.....	9 جلسه دهم
146.....	10.1 هدف
146.....	10.2 مقدمه
147.....	10.3 پروتکل I <sup>2</sup> C
149.....	10.4 وضعیت شروع مکرر

150 .....	10.5 شبتهای I <sup>2</sup> C .....
152 .....	10.6 فعالسازی I <sup>2</sup> C از طریق Codevision .....
153 .....	10.7 توابع I <sup>2</sup> C یا TWI .....
153 .....	10.7.1 فایل سرآیند twi.h .....
154 .....	10.7.2 فایل سرآیند i2c.h .....
154 .....	10.8 ارتباط ریزپردازنده با حافظه EEPROM .....
156 .....	10.8.1 ارتباط ریزپردازنده با حافظه EEPROM از طریق توابع twi.h .....
157 .....	10.8.2 ارتباط ریزپردازنده با حافظه EEPROM از طریق توابع i2c.h .....
159 .....	10.9 ارتباط ریزپردازنده با یک ریزپردازنده‌ی دیگر .....
165 .....	10.10 برنامه‌های اجرایی مبحث I <sup>2</sup> C .....

## مقدمه

این دستور کار برای دانشجویان محترم که واحد آزمایشگاه ریزپردازنده یا آزمایشگاه طراحی سیستم‌های دیجیتال ۲ را اخذ می‌نمایند تهیه شده است. آزمایشگاه ریزپردازنده و طراحی سیستم‌های دیجیتال ۲ فرصتی برای کسب مهارت در زمینه طراحی و اجرای پروژه‌های عملیاتی با استفاده از ریزپردازنده‌های خانواده AVR می‌باشد. این هدف از طریق آموزش مهارت برنامه‌نویسی و به کارگیری واحدهای سخت‌افزاری مختلف در کنار دستگاه‌های جانبی دنبال خواهد شد. برای این منظور استفاده از مهارت‌های شخصی در برنامه نویسی و به کارگیری خلاقیت‌های فردی، حضور فعال در آزمایشگاه و فعالیت عملی به صورت گروهی ضروری خواهد بود. در ادامه روش کار به همراه معیارهای ارزشیابی مورد نظر در آزمایشگاه به صورت مختصر شرح داده خواهند شد.

در هر جلسه از آزمایشگاه تعدادی از واحدهای سخت‌افزاری ریزپردازنده AVR معرفی شده و برنامه‌های اجرایی مرتبط با آن مورد بحث قرار خواهد گرفت. واحدهای سخت‌افزاری که در این دستور کار مورد توجه قرار گرفته‌اند شامل درگاه‌های ورودی و خروجی، LCD، صفحه کلید، تایمرها، موتورها، مبدل آنالوگ به دیجیتال، ارتباط سریال UART، نمایشگرهای Dot-Matrix و LCD گرافیکی، ارتباط سریال SPI و I<sup>2</sup>C می‌باشد. همچنین با توجه به اهمیت وقفه‌ها، به فراخور نیاز این ویژگی نیز در برخی پروژه‌ها استفاده گردیده است.

در این آزمایشگاه برای نوشتن برنامه از زبان C و نرم‌افزار CodeVision استفاده می‌شود. به منظور افزایش قابلیت انعطاف برنامه‌ها، سهولت عیب‌یابی و نیز ایجاد قابلیت استفاده مجدد از کدهای آماده شده در یک جلسه برای جلسات بعدی بهتر است که برنامه اصلی به صورت ماژولار و با استفاده از چندین زیربرنامه با آرگومان‌های ورودی و خروجی توسعه داده شود.

هم چنین از آن جا که پیش‌طراحی و شبیه‌سازی نرم‌افزاری از ضروریات دستیابی به طرح سخت‌افزاری مناسب است، قسمت‌هایی تحت عنوان شبیه‌سازی با نرم‌افزار Proteus در دستور کار قرار داده شده است.

بنابراین لازم است هر دانشجو پیش از آغاز جلسه عملی در آزمایشگاه، موارد مشخص شده را در محیط نرم افزار codevision آماده و سپس در محیط نرم‌افزار شبیه‌ساز پروتوتوس تست نماید. طی جلسه عملی آزمایشگاه این برنامه‌ها توسط اعضای گروه روی بورد آزمایشگاهی پیاده‌سازی خواهد شد. لازم به ذکر است که مدل شبیه‌سازی شده صرفاً برای تست اولیه صحت برنامه‌ها مورد استفاده قرار می‌گیرد و بنابراین به هنگام پیاده‌سازی روی پکیج آموزشی باید ملاحظات و تمهیدات لازم از جمله تامین جریان المانهای مختلف و همچنین درایورهای مورد نظر توسط دانشجویان لحاظ گردد.

نسخه نهایی فایل مربوط به هر جلسه نیز باید در پایان جلسه در سامانه یکتا و در بخش تکلیفی که برای همان جلسه تعریف شده است بارگذاری نمایند. قالب ارسال این تکالیف شامل یک فایل فشرده با محتوای مشخص شده در شکل 1 می‌باشد.



نمودار 1: قالب ارسال تکالیف

حضور به موقع در تمامی جلسات آزمایشگاه الزامی بوده و هر غیبت غیر مجاز باعث کسر نمره می‌گردد. به علاوه هر یک از اعضای گروه باید آمادگی لازم را برای پاسخگویی به سوالات احتمالی و توضیح دادن برنامه‌های ایجاد شده داشته باشند. هم چنین برگزاری کوئیز در جلسات آزمایشگاه یکی از بخش‌های ارزشیابی در آزمایشگاه خواهد بود.

دانشجویان موظفند از کلیه‌ی وسایلی که در اختیار آن‌ها قرار داده می‌شود مراقبت کرده و در پایان هر جلسه آن‌ها را در جای مربوط به خود قرار دهنند تا نظم آزمایشگاه حفظ گردد.

امتحان پایان‌ترم به صورت کتبی نیز بخش دیگری از ارزشیابی آزمایشگاه خواهد بود. هم چنین بنا به شرایط و با اطلاع قبلی پروژه عملی مجزا نیز در نظر گفته خواهد شد.

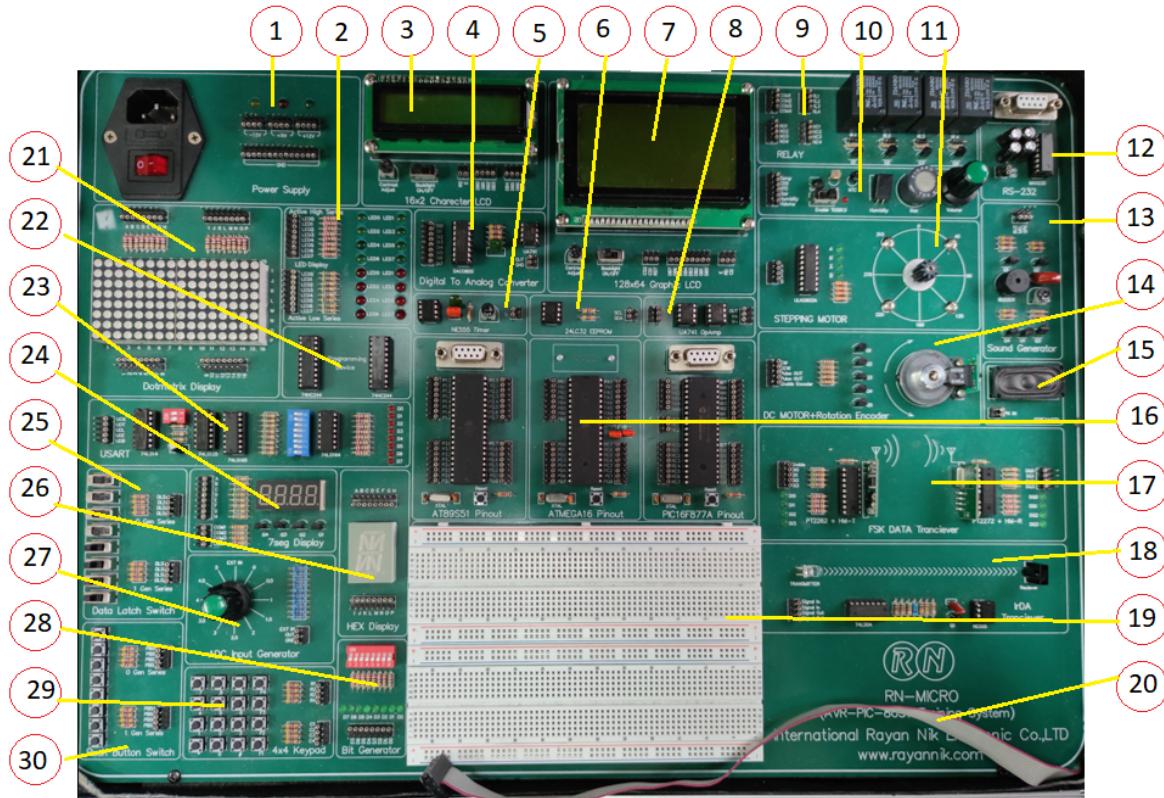
در نهایت با توجه به محدودیت زمان آزمایشگاه توصیه می‌گردد که دانشجویان علاوه بر آزمایش‌های موجود در این دستور کار محتوای موجود در منابع اینترنتی و نیز کتاب‌های مرجع را دنبال نمایند. برای دسترسی آسانتر به محتوای آزمایشگاه، هر بخش در قالب فایل جداگانه‌ای آماده شده که در سامانه الکترونیکی دروس قابل مشاهده است.

با تشکر

مسئول آزمایشگاه ریزپردازنده / طراحی سیستمهای دیجیتال 2

## معرفی بورد آزمایشگاه

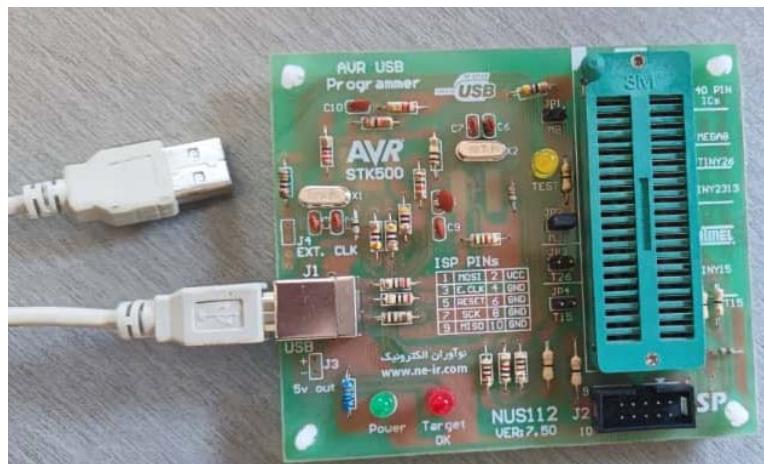
همان گونه که در پیش گفتار توضیح داده شد، هدف از این آزمایشگاه آشنایی با قابلیت‌های ریزپردازنده‌ای خانواده AVR است که با توجه به بورد آزمایشگاهی موجود، به صورت خاص ریزپردازنده‌های Atmega16/32 مورد بررسی قرار خواهد گرفت. نمایی از سخت‌افزار مورد استفاده در آزمایشگاه به همراه واحدهای سخت‌افزاری و دستگاه‌های جانبی تعییه شده بر روی آن در شکل 1 نشان داده شده‌اند. هر یک از این واحدها متناسب با موضوع هر جلسه در محتوای مربوط به همان جلسه تشریح خواهد شد.



1	Power Supply	11	Stepper motor	21	Dot-Matrix
2	LEDs	12	RS-232	22	Programming device
3	Alphanumeric LCD	13	Sound Generator	23	USART
4	Digital to Analog converter	14	Dc motor	24	7-Segment
5	NE555 timer	15	Speaker	25	switch
6	24LC32 EEPROM	16	Atmega32/16	26	Hex Display
7	Graphical LCD	17	FSK Data Tranciever	27	ADC input Generator
8	UA741 Op-Amp	18	IR data Tranciever	28	Bit Generator
9	Relay	19	Breadboard	29	4*4 keypad
10	Sensores	20	ISP Programmer	30	Push button Switch

شکل 1: نمایی از سخت‌افزار مورد استفاده در آزمایشگاه

هم چنین برای پروگرام نمودن ریزپردازنده موجود بر روی بورد از پروگرامر STK500 استفاده می‌گردد که نمایی از آن در شکل 2 نشان داده شده است.



شکل 2: نمایی از پروگرامر STK500 با کابل USB

# 1 جلسه اول

## درگاه‌های ورودی و خروجی

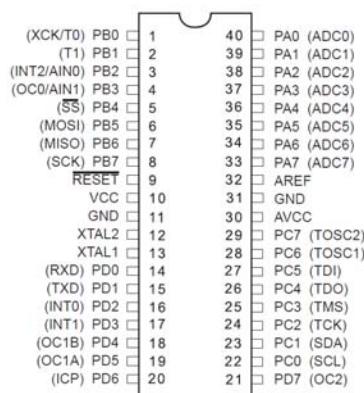
### 1.1 هدف

در این جلسه موارد ذیل بررسی می‌گردد.

- آشنایی با درگاه‌های ریزپردازنده و ثبات‌های مربوط به آن
- آشنایی با فایل‌های سرآیند
- آشنایی با واحدهای ورودی/خروجی اولیه: کلیدهای فشاری و کشویی، LED و نمایشگر 7-Segment
- ایجاد پروژه و پروگرام کردن ریزپردازنده
- زیربرنامه نویسی

### 1.2 معرفی درگاه‌های ورودی و خروجی Atmega16/32

نمای ریزپردازنده Atmega16/32 با بسته‌بندی PDIP در شکل 1-1 نشان داده شده است. این ریزپردازنده دارای 40 پایه است که 32 پایه‌ی آن مربوط به 4 درگاه 8 بیتی ورودی و خروجی می‌باشد.



شکل 1-1: پایه‌های ریزپردازنده Atmega16/32 با بسته‌بندی PDIP

به ازای هر درگاه سه ثبات<sup>۱</sup> به شرح زیر وجود دارد:

<sup>۲</sup>PORTx ثبات داده خروجی

<sup>1</sup> Register

<sup>2</sup> PORTX data register

<sup>۱</sup> ثبات جهت داده DDRx

<sup>۲</sup> ثبات داده ورودی PINx

در شکل 2-1 ساختار این ثبات ها برای درگاه A نشان داده شده است و سایر درگاهها نیز ساختاری مشابه دارند.

#### PORATA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORATA
Initial Value	R/W								

#### DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Initial Value	R/W								

#### PINA – Port A Input Pins Address

Bit	7	6	5	4	3	2	1	0	
Read/Write	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Initial Value	R	R	R	R	R	R	R	R	

شکل 2-1: ثبات‌های درگاه A

در مورد ثبات‌های درگاه‌های ورودی و خروجی نکات زیر حائز اهمیت است:

- ثبات PINx تنها قابل خواندن است ولی ثبات‌های PORTx DDRx و هم قابل خواندن و هم قابل نوشتן هستند.

- اگر بیت <sup>۱</sup>ام از ثبات DDRx یک درگاه برابر صفر باشد پایه <sup>۱</sup>ام از آن درگاه به صورت ورودی و اگر آن بیت برابر ۱ باشد، پایه متناظر به صورت خروجی خواهد بود.

- برای خواندن وضعیت پایه‌ای که به صورت ورودی تعریف شده است، بیت متناظر از ثبات PINx خوانده می‌شود و برای تعیین وضعیت پایه‌هایی که به صورت خروجی تعریف شده‌اند، مقدار آن‌ها در بیت متناظر از ثبات PORTx نوشته می‌شود.

<sup>۱</sup> PORTX data direction register

<sup>۲</sup> PORTX input pins address

به عنوان نمونه در برنامه 1-1، چهار بیت کم ارزش در گاه B به عنوان خروجی و چهار بیت پر ارزش آن به عنوان ورودی تعریف شده است.

برنامه 1-1

DDRB = 0x0F;

همان گونه که در برنامه 1-2 دیده می‌شود، دسترسی به بیت‌های هر کدام از ثبات‌ها به تنها‌یی نیز امکان پذیر است.

برنامه 2-1

```
DDRA.3 = 1; //set bit 3 of port A as output
PORTA.3 = 1; //make bit 3 of port A high
```

ذکر این نکته لازم است که با توجه به محدودیت تعداد پایه‌های ریزپردازنده و به منظور ایجاد امکان استفاده از سایر قابلیت‌های Atmega16/32، برخی از پایه‌های در گاهها به کمک یک مالتی پلکسر به سایر واحدهای عملکردی ریزپردازنده نیز متصل شده است. لذا این پایه‌ها علاوه بر عملکرد یک ورودی/خروجی ساده، کاربردهای دیگری هم مانند مبدل آنالوگ به دیجیتال، تایмер، ارتباط سریال و ... دارند که در سایر جلسات بررسی خواهند شد.

### 1.3 معرفی فایل سرآیند (Header File)

فایل‌های سرآیند، شامل برخی زیربرنامه‌ها یا تعاریف اولیه هستند که جهت سهولت در برنامه نویسی به کار می‌روند و در پوشه INC در محل نصب CodeVision قابل دسترسی هستند. برای فراخوانی هر یک از این فایل‌ها، از دستور #include در ابتدای برنامه و قبل از تابع main استفاده می‌شود. در برنامه 1-3 دو روش برای این منظور معرفی شده است.

برنامه 3-1

```
#include <file_name.h>
#include "file_name.h"
```

در روش اول، کامپایلر فایل سرآیند را در زیر شاخه inc در محل نصب برنامه جست‌وجو می‌کند. اما در حالت دوم، ابتدا پوشه جاری برنامه را جست‌وجو می‌کند و سپس به مسیر /inc/ رجوع می‌کند.

### 1.3.1 فایل سرآیند mega16.h

ریزپردازنده‌های AVR دارای 32 پایه ورودی خروجی هستند که ارسال و دریافت داده، پیکربندی ریزپردازنده و تنظیم امکانات داخلی آن از طریق این ثبات‌ها انجام می‌شود. کار با این ثبات‌ها از طریق دسترسی به حافظه ریزپردازنده میسر خواهد بود که آدرس‌ها و اشاره‌گرهای مربوطه را می‌توان با مطالعه Datasheet مربوطه به دست آورد.

در برنامه 1-4 تنظیم درگاه D به صورت خروجی با استفاده از آدرس انجام شده است.

```
برنامه 4-1
void main(void)
{
    *(unsigned char *) 0x11=0b11111111;
    while(1) { };
}
```

اما برای سهولت کار، هر ریزپردازنده فایل سرآیند مخصوص به خود را دارد که در آن برای تمامی ثبات‌های ریزپردازنده، یک اشاره‌گر براساس نام و آدرس آن‌ها در حافظه تعریف شده است. با include کردن این فایل سرآیند می‌توان به جای آدرس ثبات، از اسم آن ثبات استفاده کرد.

در برنامه 1-5 تنظیم درگاه D به صورت خروجی با استفاده از نام DDRD که در سرآیند فایل ریزپردازنده تعریف شده، انجام شده است.

```
برنامه 5-1
#include <mega16.h>
void main(void)
{
    DDRD=0xFF;
    while(1) { };
}
```

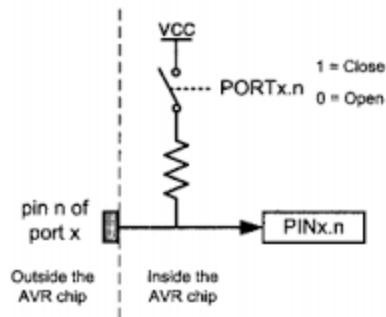
### 1.3.2 معرفی فایل سرآیند delay.h

این فایل سرآیند شامل دو تابع void delay\_ms(unsigned int n) و void delay\_us(unsigned int n) است که به کمک آن‌ها می‌توان در برنامه تأخیر ایجاد کرد. لازم به ذکر است در هنگام استفاده از این توابع اگر وقفه‌ای فعال باشد ممکن است تأخیر بیش از اندازه طولانی گردد. لذا در مواردی که میزان تاخیر اهمیت داشته باشد استفاده از این توابع توصیه نمی‌گردد.

## 1.4 معرفی مقاومت داخلی در گاهها

هنگامی که یک پایه را به صورت ورودی تنظیم می‌کنیم، وظیفه تعیین مقدار آن بر عهده یک مدار راهانداز خارجی خواهد بود. به جهت ساده شدن این مدار و نیز کاهش هزینه و انرژی مصرفی عموماً این مدار صرفاً در زمانی که پایه باید مقدار صفر بگیر آن را به ولتاژ زمین متصل می‌نماید و در غیر این صورت مقداری را به پایه اعمال نمی‌کند. در این حالت ممکن است در اثر تغییرات ولتاژ، تغییرات جریان، نویز و... حالت آن پایه به صورت ناخواسته تغییر کند که با توجه به سرعت بالای ریزپردازنده این تغییر ممکن است به اشتباه برای ریزپردازنده به معنی یک یا صفر شدن پایه تلقی شود و در اجرای برنامه اخلال ایجاد کند. لذا برای از بین این تأثیرات ناخواسته باید با استفاده از مقاومت‌های بالاکش و یا پایین کش، پایه‌ها از حالت شناور خارج شده و به یکی از خطوط تغذیه متصل شوند. اگرچه برای این کار می‌توان از مقاومت خارجی استفاده نمود، اما در داخل تراشه‌های AVR نیز یک سری مقاومت‌های بالاکش جهت استفاده در درگاه‌های ورودی تعییه شده‌اند.

برای فعل نمودن مقاومت‌های داخلی کافیست در رجیستر PORTx بیت مورد نظر برابر یک گردد. با این کار مدار مقاومت بالاکش مطابق شکل 3-1 فعال می‌گردد.



شکل 3-1: نمایی از مقاومت داخلی در ریزپردازنده

## 1.5 معرفی واحدهای ورودی/خروجی اولیه پکیج آموزشی

### 1.5.1 معرفی کلیدهای کشویی

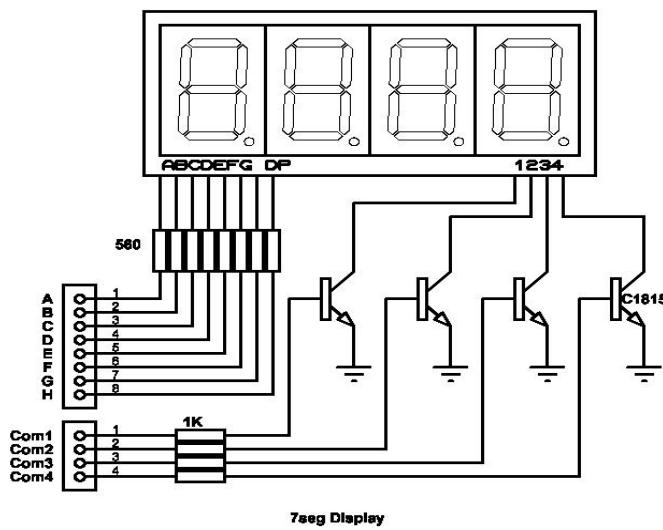
در پکیج آموزشی 8 عدد کلید کشویی به منظور تولید صفر و یک دائمی در بلوکی تحت عنوان Data Latch Switch قرار داده شده است. این کلیدها وظیفه تولید صفر و یک منطقی را بر عهده دارند. در ساختار این بورد آموزشی، از مقاومت‌های بالاکش و پایین کش تعییه شده روی بورد برای جلوگیری از به وجود آمدن حالت شناور بر روی پایه‌های ریزپردازنده استفاده شده است.

### Push Button 1.5.2 [برگشت]

از کلیدهای Push Button به منظور تولید صفر و یک لحظه‌ای استفاده می‌شود. در بورد آموزشی، 8 عدد Push Button قرار دارد که از آن‌ها برای تولید صفر و یک منطقی استفاده می‌شود. با فشار دادن این کلیدها، یک سیگنال صفر یا یک در پایه متناظر ایجاد می‌کند و بلافاصله پس از رها کردن کلید، خروجی به سطح یک یا صفر باز می‌گردد.

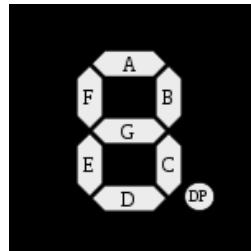
### 7-Segment 1.5.3 [برگشت]

این نمایشگر، برای نمایش اعداد و بعضی از حروف لاتین به کار می‌رود. مطابق شکل 1-4 بر روی بورد آموزشی یک نمایشگر چهار رقمی وجود دارد که در آن برای هر 7-Segment یک پایه‌ی Enable تعییه شده است. علاوه بر آن 8 پایه‌ی داده به نام‌های A تا H نیز وجود دارد که برای هر چهار 7-Segment مشترک هستند.



شکل 1-4 نحوه اتصال پایه‌های 7-Segment

یک کردن هر کدام از پایه‌های داده، موجب روشن شدن Segment متناظر با آن خواهد شد. بنابراین برای نمایش یک عدد خاص، بر روی هر 7-Segment باید داده‌ی مناسب را بر روی پایه‌های داده ارسال کرده و پایه Enable آن را یک کرد. چنان‌چه این کار به صورت متناوب و با فرکانس مناسب انجام شود، می‌توان به صورت همزمان اعداد چهار رقمی دلخواه را روی چهار 7-Segment رؤیت کرد. با توجه به شکل 1-5 برای نمایش عدد '1' باید به B و C مقدار 1 منطقی اعمال کرد.

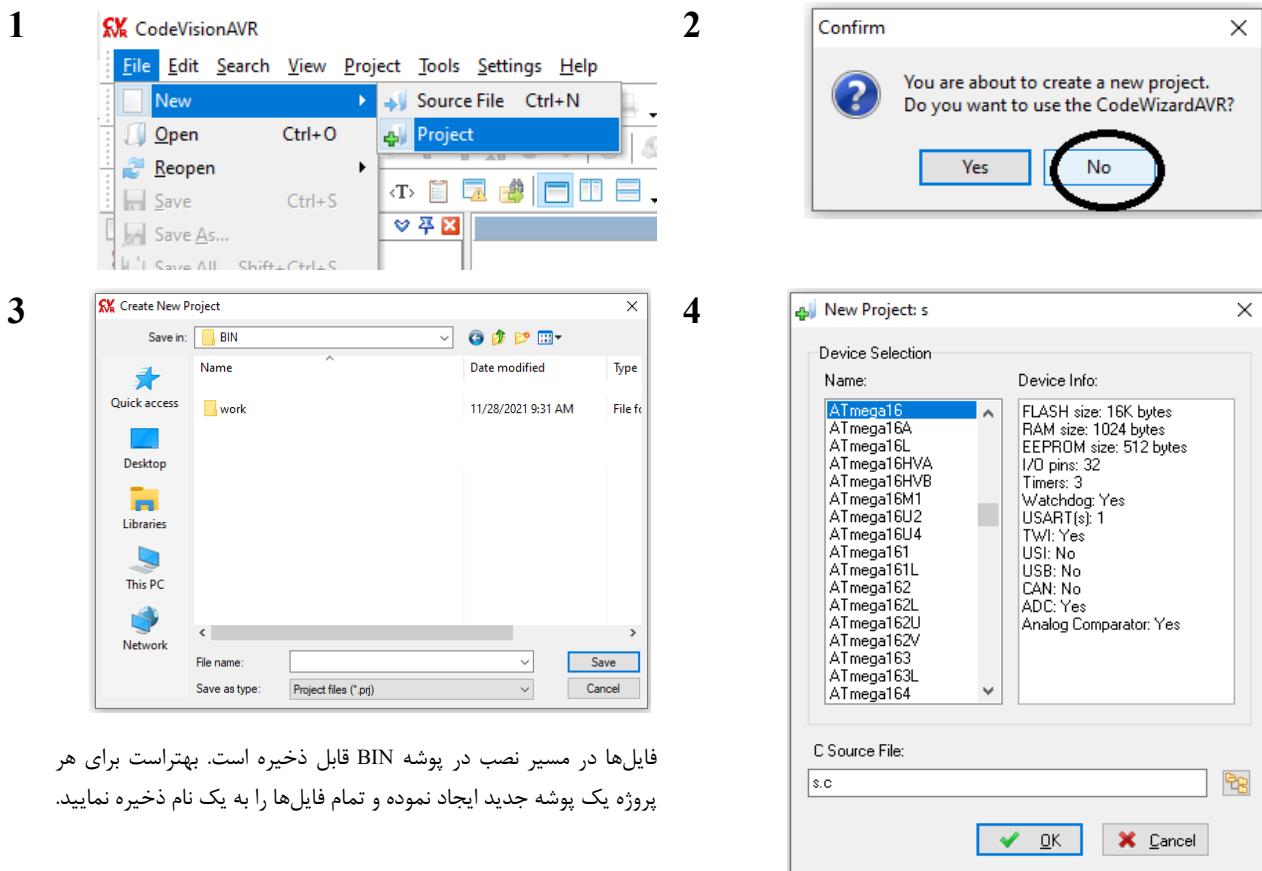


شکل ۵-۱ نامگذاری بخش‌های 7-Segment

## 1.6 ایجاد پروژه در محیط codevision

## 1.6.1 ایجاد پروژه بدون استفاده از CodeWizard

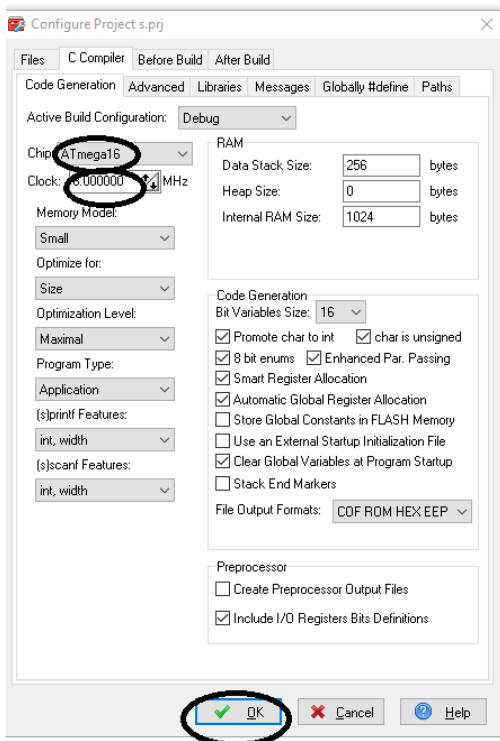
برای ایجاد پروژه بدون استفاده از CodeWizard باید مراحل نشان داده شده در شکل ۶-۱ اجرا گردد.



فایل‌ها در مسیر نصب در پوشه BIN قابل ذخیره است. بهتر است برای هر پروژه یک پوشه جدید ایجاد نموده و تمام فایل‌ها را به یک نام ذخیره نمایید.

ریزپردازنده مورد نظر را از لیست انتخاب نمایید.

5

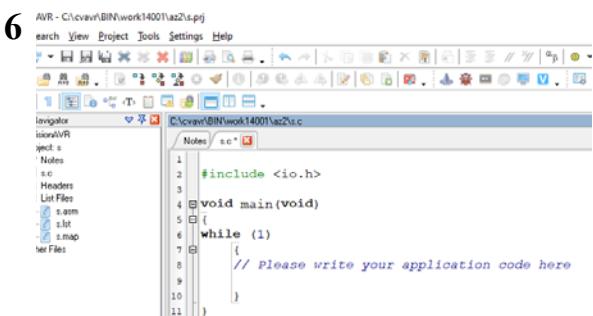


فرکانس کاری ریزپردازنده را تعیین نمایید.

7 leVisionAVR - C:\cvavr\BIN\work14001\az2\s.prj

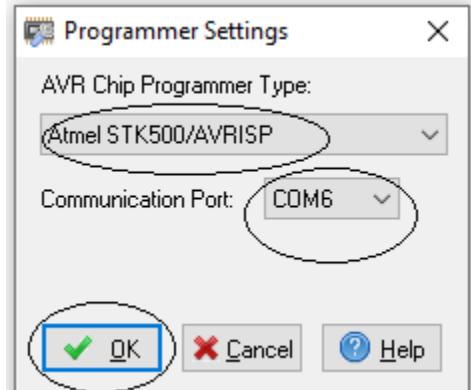


6



کدهای فوق به صورت پیشفرض تولید می‌شوند. فایل io.h حاوی فایل‌های سرآیند مربوط به تمام ریزپردازنده‌های معرفی شده در است. می‌توان آن را با mega16.h جایگزین نمود.  
در ادامه کدهای مورد نظر به برنامه اضافه می‌شود.

8



در گاهی که پروگرامر به آن متصل شده است را انتخاب نمایید.

9

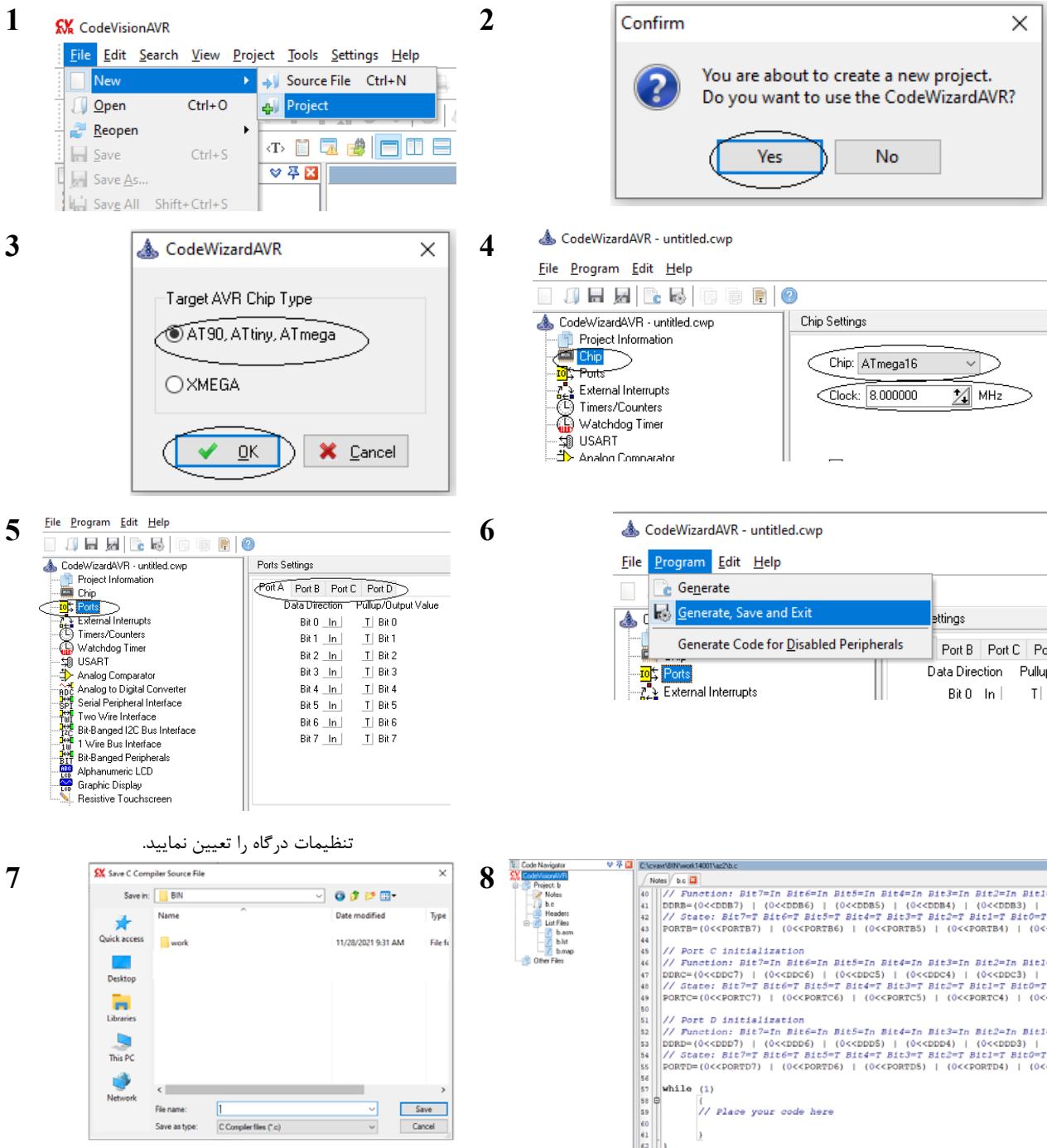


پروژه را کامپایل نمایید. بدین وسیله فایل اجرایی با قالب hex که قابل بارگذاری در ریزپردازنده است ایجاد می‌گردد.

شکل 1-6: مراحل ایجاد پروژه بدون استفاده از codewizard

### 1.6.2 ایجاد پروژه با استفاده از CodeWizard

مراحل ایجاد پروژه با استفاده از قابلیت‌های codwizard در شکل 7-1 نشان داده شده است.



فایل‌های هر پروژه را داخل یک پوشه و با نام یکسان برای تمام فایل‌ها ذخیره نمایید.

کدهای فوق به صورت پیش‌فرض تهیه می‌شود. در ادامه کدهای مورد نظر به برنامه اضافه می‌شود.

شکل 7-1: مراحل ایجاد پروژه با استفاده از CodeWizard

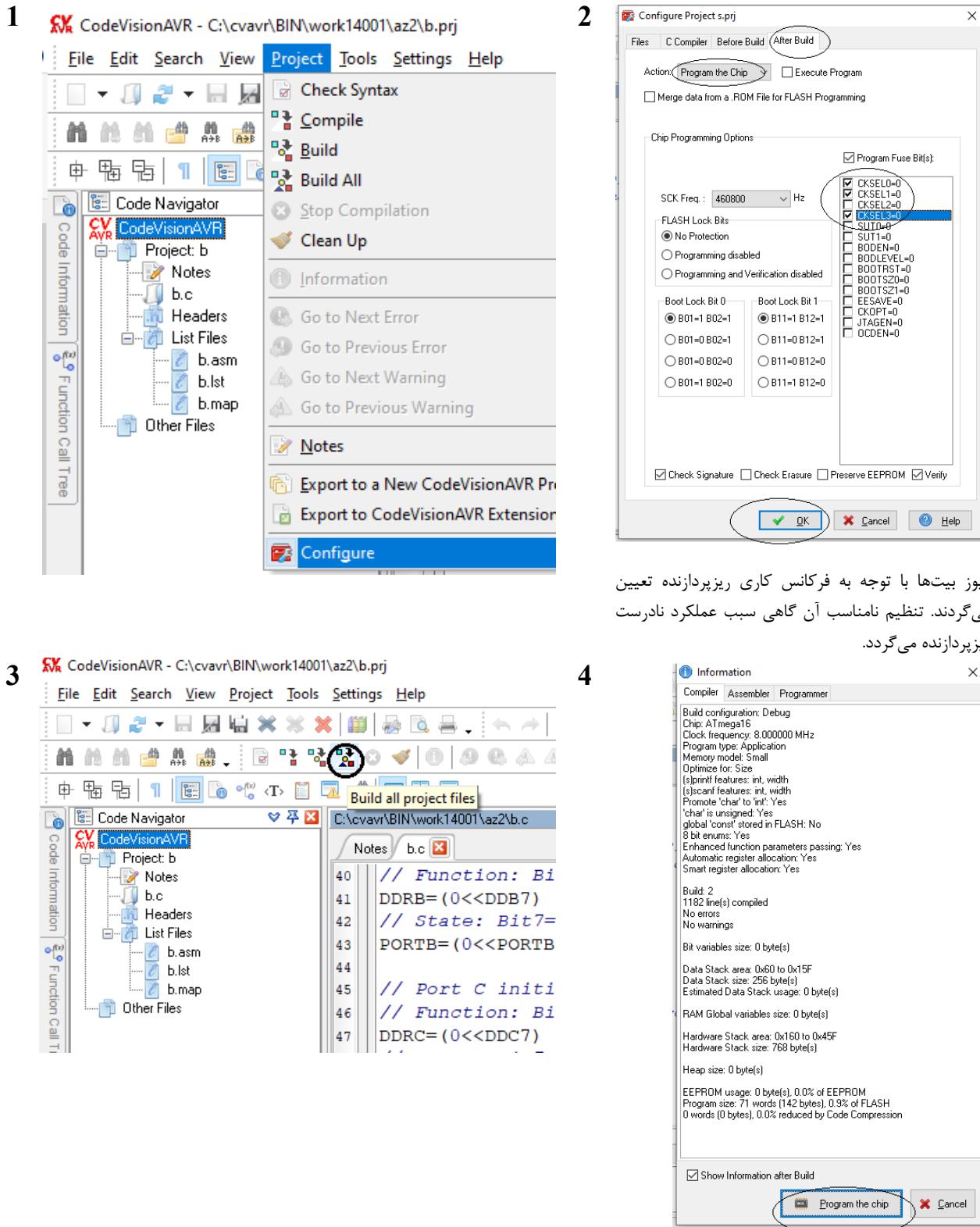
### 1.7 نحوه پروگرام نمودن ریزپردازنده

برای پروگرام ریزپردازنده ابتدا مطابق مراحل قبل، مدل پروگرامر انتخاب می‌شود و یکی از روش‌های زیر اجرا می‌گردد.

در روش نخست که مراحل آن در شکل 1-8 نشان داده شده، پس از کامپایل پروژه گزینه‌ی پروگرام تراشه فعال می‌شود و بدین وسیله روند پروگرام نمودن ساده‌تر می‌گردد.

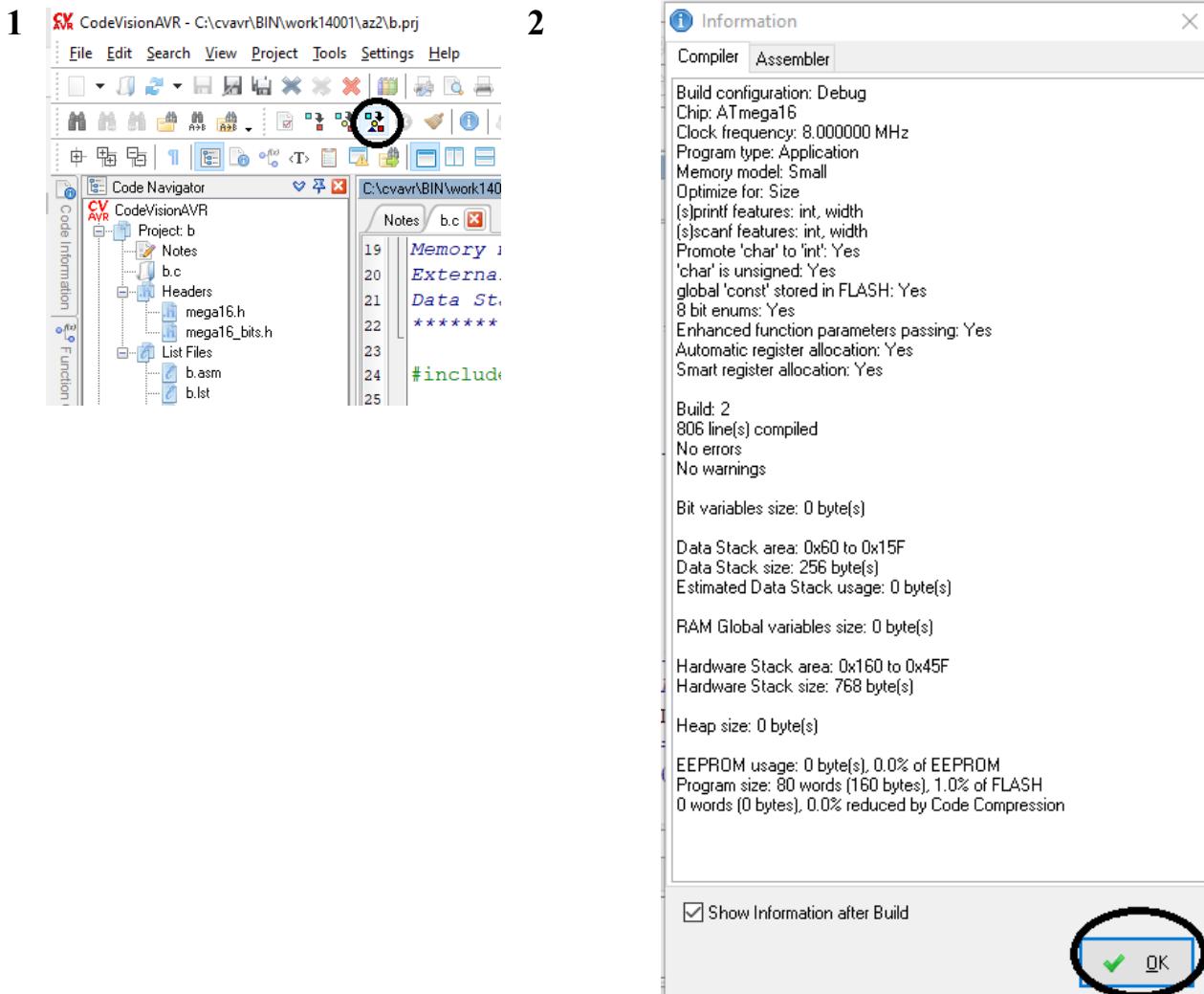
در پردازنده Atmega16/32 تعدادی فیوز بیت (Fuse bits) وجود دارد که تنظیمات خاصی را به صورت کاملاً سخت‌افزاری جهت پیکربندی عملکرد کلی تراشه درون خود ذخیره می‌کنند. به عنوان مثال فعال کردن قفل برنامه که از خوانده شدن اطلاعات ریزپردازنده به صورت خارجی جلوگیری می‌کند، انتخاب نوع اسیلاتور ریزپردازنده با توجه به محدوده فرکانس تولید شده‌ی آن و تعیین نحوه پروگرام کردن ریزپردازنده از جمله تنظیماتی است که توسط فیوز بیت‌ها قابل انجام است.

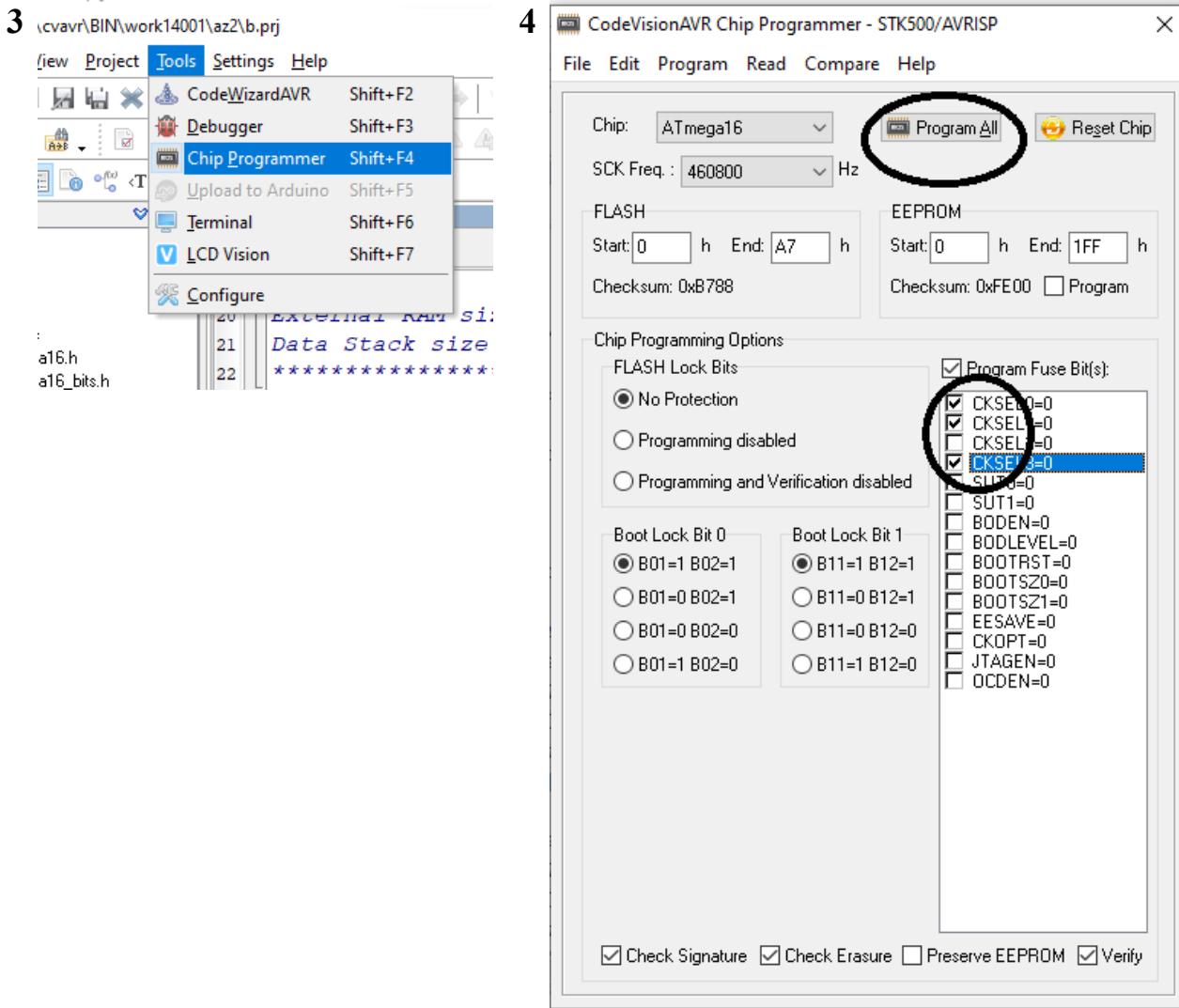
این فیوز بیت‌ها در یک بخش رزرو شده از حافظه دائمی تراشه قرار گرفته‌اند که مقادیر آن‌ها تا زمان برنامه‌ریزی مجدد تغییر نخواهد کرد و حتی با قطع برق نیز این مقادیر حفظ می‌گردند. لذا برای تغییر پیکربندی این فیوز بیت‌ها باید در زمان برنامه‌ریزی، بخش مربوط به آن‌ها به صورت مجزا و از طریق نرم‌افزار مربوطه تنظیم گردد.



شکل 1-8: روش اول بروگرام کردن تراشه

مراحل روش دوم در شکل 1-9 نشان داده شده است. با استفاده از این روش می‌توان هنگامی که کدهای ایجاد شده در Codevision موجود نیست و فقط کد Hex وجود دارد، محتوای این فایل را بر روی ریزپردازنده بارگذاری نمود. بدین منظور از منوی file می‌توان فایل hex را انتخاب نمود.





شکل 4-1 : روش دوم پروگرام نمودن ریزپردازنده

### 1.8 برنامه‌های کاربردی

در ادامه چندین برنامه کاربردی برای درگاه‌های ورودی و خروجی قابل مشاهده است. در برنامه ۱-۶ یکی از درگاه‌ها بدون استفاده از CodeWizard مقدار دهی شده است.

برنامه ۱-۶

```
#include <mega16.h>
void main(void)
{
    DDRB=0xFF;
    PORTB=0b01010101;
    PORTB.5=1;
    while(1);
}
```

در برنامه 1-7 داده از کلیدهای کشویی پکیج که به درگاه B متصل خوانده شده و بر روی هشت LED که به درگاه D متصل هستند نمایش داده می‌شود.

برنامه 7-1

```
#include <mega16.h>
char number;
void main(void)
{
    DDRB = 0x00;
    DDRD = 0xFF;
    while(1)
    {
        number = PINB;
        PORTD = number;
    }
}
```

در برنامه 1-8 یک شمارنده طراحی شده است که در صورت یک بودن PINB.0 که به یک کلید کشویی متصل است به مقدار آن اضافه می‌شود. با استفاده از دستور char count=0x00; یک متغیر 8 بیتی تعریف شده است که مقدار اولیه آن صفر بوده و در ادامه مقدار شمارنده را در هر لحظه در خود نگه می‌دارد. هم چنین مقدار آن همواره بر روی LEDهای متصل به درگاه D نمایش داده می‌شود.

برنامه 8-1

```
#include <mega16.h>
void main(void)
{
    char count = 0X00;
    DDRD = 0xFF;
    PORTD = 0X00;
    DDRB.0 = 0;
    while(1)
    {
        if (PINB.0)
        {
            count = count + 1;
            PORTD = count;
        }
    }
}
```

با ریختن این برنامه بر روی ریزپردازنده به نظر می‌رسد LEDها همگی با نور کمی روشن هستند و شمارنده به درستی عمل نمی‌کند. علت آن است که سرعت کار ریزپردازنده بسیار زیاد است و شمارنده با سرعت بالای LEDها روشن و خاموش می‌کند. برای حل این مشکل باید شمارش با فاصله زمانی مناسبی انجام شود. برای ایجاد تأخیر می‌توان از توابع وجود در delay.h استفاده کرد.

در برنامه 1-9، مقدار متغیر number 7-Segment روی داده D که Enable آنها از طریق تغذیه تامین گردیده نمایش داده می‌شود.

برنامه 9-1

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

```

flash unsigned char digit[]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,
    0x07, 0x7F, 0x6F};
int number = 0;
void main(void)
{
    DDRD = 0xFF;
    number = 5;
    while(1)
    {
        PORTD = digit[number];
        delay_ms(1000);
    }
}

```

### 1.9 زیربرنامه نویسی و فراخوانی آن در تابع اصلی

در برنامه 10-1 نمونه‌ای از زیربرنامه نویسی نشان داده شده است.

برنامه 10-1

```

#include <mega16.h>
Void sub_routine(void)
{
    DDRB=0xFF;
    PORTB=0b01010101;
    PORTB.5=1;
}
void main(void)
{
    sub_routine();
    while(1);
}

```

### 1.10 سوالات تشریحی در گاههای ورودی و خروجی

(1) به سوالات زیر پاسخ دهید:

- الف - ساختار مقاومت بالاکش (pull-up) و کاربرد آن را مختصراً شرح دهید.
- ب - انواع روش‌های اتصال کلید به ریزپردازنده را نام ببرید.
- ج - انواع روش‌های برنامه‌ریزی ریزپردازنده‌های AVR را نام ببرید.
- د - ریزپردازنده AVR چند نوع حافظه دارد؟ نحوه استفاده از آن‌ها چگونه است؟ تفاوت آن‌ها در چیست؟

(2) نحوهی عملکرد ثبات‌های DDRx, PORTx و PINx را توضیح دهید و پیکربندی لازم برای موارد زیر را

بنویسید:

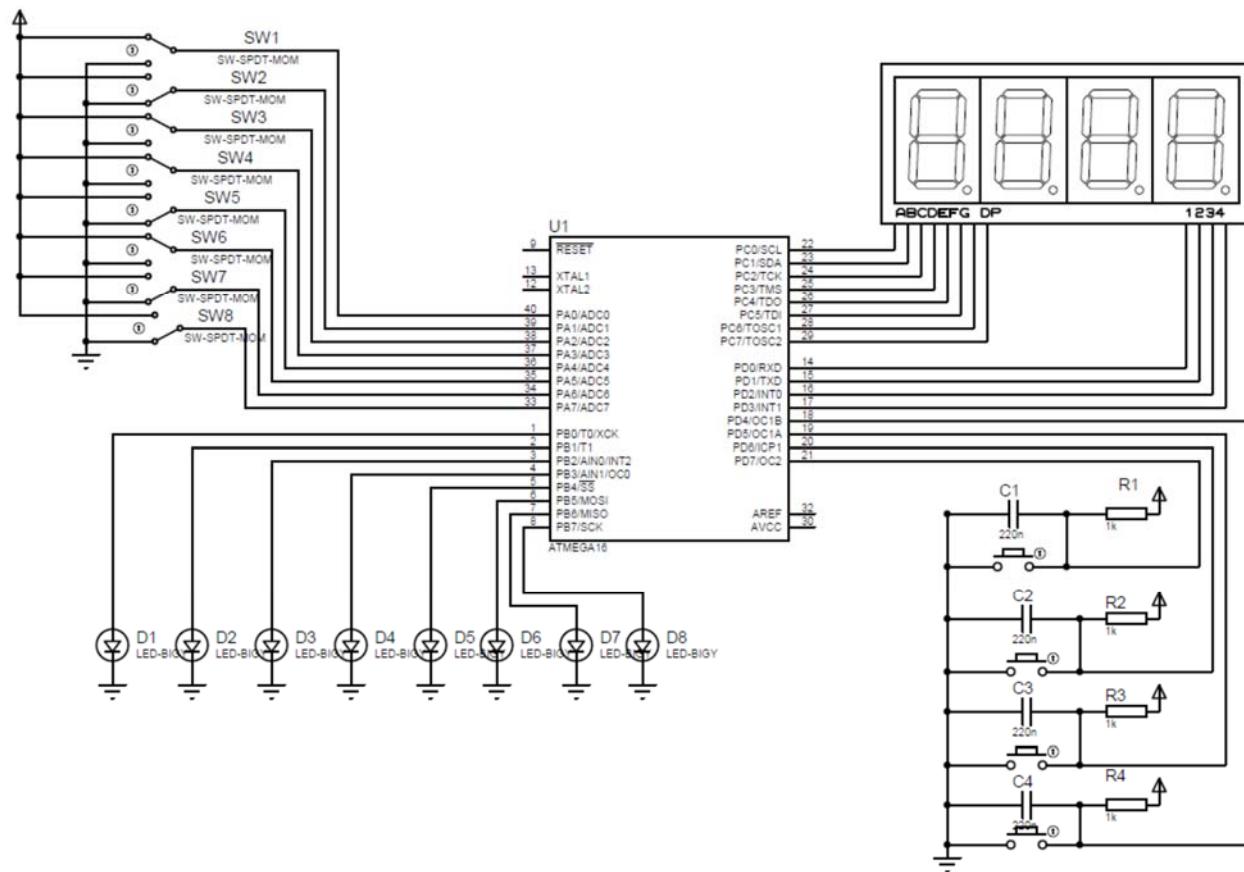
الف) درگاه A را طوری تعریف کنید که بیت‌های آن به صورت یک در میان ورودی و خروجی باشند.

ب) درگاه B را به صورت خروجی تعریف کنید به طوری که مقاومت بالاکش آن حذف نشود.

۳) برای نمایش کاراکترهای زیر، چه مقادیری را باید به ورودی خط داده‌ی 7-Segment اعمال کرد؟

d  
A  
H  
F

## 1.11 برنامه‌های اجرایی درگاه‌های ورودی و خروجی:



شکل 1-10: سختافزار نمادین برای تست برنامه‌های درگاه‌های ورودی-خروجی

با توجه به سختافزار نشان داده شده در شکل 1-10 زیربرنامه های زیر را بنویسید.

(1) همه‌ی led 4 مرتبه خاموش و روشن شوند (در فاصله زمانی 0.5 ثانیه).

(2) موقعیت LED روی درگاه B، به مدت 3 ثانیه جایه جا گردد (یکی از LED ها را با قرار دادن عدد

1 روی پورت روشن نموده و در فاصله زمانی مشخصی عدد یک را شیفت دهید).

(3) تغییرات درگاه A روی led ها نمایش داده شود.

(4) اعداد 0 تا 9 به صورت شمارش معکوس روی هر یک از 7segment نشان داده شود.

(5) عدد سه رقمی خوانده شده از درگاه A روی 7Segment نشان داده شود و با دقت 0.2 و با فاصله زمانی 200 ms کاهش یابد.

6) فرض کنید که هر یک از کلیدهای فشاری برای ریست نمودن رقم‌های 7segment در نظر گرفته شده است. با استفاده از بند 5، برنامه‌ای بنویسید که ضمن کاهش عدد به اندازه‌ی 0.2، در صورت فشردن کلیدها رقم مربوطه صفر گردد و روند کاهشی همچنان ادامه یابد تا به صفر برسد.

7) زیربرنامه‌های بندهای 1 تا 6 را در یک پروژه تلفیق نمایید به صورتی که همه بندها به ترتیب اجرا گردند.

پروژه کامل کد ویژن (شامل تمام فایل‌ها) برای هریک از بندها را بنویسید و از فایل‌های کمکی نیز استفاده نمایید. سپس در محیط پروتئوس برنامه را شبیه سازی نموده و ارسال نمایید.

## 2 جلسه دوم

### زیربرنامه‌نویسی و ایجاد فایل‌های جانبی

#### 2.1 هدف

در این جلسه، نحوه زیربرنامه‌نویسی به همراه آرگومان‌های ورودی/خروجی و همچنین ایجاد فایل‌های جانبی شامل فایل‌های پیاده‌سازی زیربرنامه با قالب C و فایل‌های سرآیند زیربرنامه‌ها با قالب h بررسی می‌گردد.

#### 2.2 مقدمه

امروزه در اکثر پروژه‌های عملیاتی به منظور افزایش قابلیت انعطاف سیستم و رفع نیازهای متنوع کاربران، برنامه‌های پیچیده و با حجم زیادی از کد به کار گرفته می‌شوند که عموماً به روش کار گروهی توسعه می‌یابند. لذا منطقی به نظر می‌رسد که جهت توسعه این برنامه‌ها راهکاری دنبال شود تا ضمن تسهیل انجام کار گروهی، با ایجاد ساختار و نظم مناسب در کدهای نوشته شده، از یک سو خوانایی (Readability) و قابلیت ردیابی (Traceability) این کدها افزایش یابد و از سوی دیگر امکان استفاده بخش‌هایی از برنامه‌های قبلی در پروژه‌های جدید فراهم گردد (Reusability).

راه حل مناسب برای این منظور، کدنویسی ساختار یافته و استفاده از زیربرنامه‌های مجزا برای پیاده‌سازی بخش‌های مختلف کد می‌باشد. به عبارت دقیق‌تر بخش‌هایی از کد که هدف مشخصی دارند و احتمالاً در جاهای مختلف، استفاده از آن‌ها تکرار می‌گردد، در قالب یک زیربرنامه مجزا توسعه داده شده و هر جا که نیاز به آن‌ها هست این زیربرنامه‌ها فراخوانی می‌شوند. هم چنین می‌توان برای هر زیربرنامه تعدادی آرگومان ورودی و خروجی در نظر گرفت تا پویایی و قابلیت انعطاف لازم برای به کار گیری آن در شرایط گوناگون فراهم گردد.

برای جلوگیری از افزایش حجم کد در برنامه اصلی و نیز ایجاد قابلیت استفاده مجدد از کد زیربرنامه‌ها در پروژه‌های مختلف، عموماً زیربرنامه‌های توسعه داده شده در فایل‌های جانبی ذخیره می‌گردد. در هر مجموعه فایل جانبی دسته‌ای از زیربرنامه‌ها برای یک کاربرد خاص جمع‌آوری و به صورت یک کتابخانه قابل استفاده خواهند بود. به صورت کلی هر مجموعه فایل جانبی، شامل دو فایل با قالب‌های c و h است. فایل c حاوی بدنی اصلی زیربرنامه‌ها می‌باشد و فایل h سرآیند زیربرنامه‌های تعریف شده در فایل c و برخی تعاریف اولیه را در بر می‌گیرد.

#### 2.3 ایجاد زیربرنامه با آرگومان ورودی و خروجی

برنامه‌ای ساده را در نظر بگیرید که در آن ریزپردازنده باید داده‌ها را از یکی از درگاه‌های ورودی بخواند و همان داده را روی یک درگاه خروجی نمایش دهد. Error! Reference source not found. داده ورودی را از درگاه A خوانده و بر روی درگاه B نمایش می‌دهد.

برنامه 1-2

```
#include <mega16.h>
void main(void)
{
    DDRA=0x00;      // as input
    DDRB=0xFF; // as output

    while (1)
    {
        PORTB=PINA;
    }
}
```

**Error! Reference source** اکنون می‌خواهیم همین کار را با استفاده از زیربرنامه انجام دهیم. برای این منظور **not found.** را به صورت برنامه 2-2 اصلاح می‌کنیم.

برنامه 2-2

```
#include <mega16.h>
/*********************************************************/
void portfun(void)
{
    DDRA=0x00;      // as input
    DDRB=0xFF; // as output
    PORTB=PINA;
}
/*********************************************************/
void main(void)
{
    while (1){
        Portfun();
    }
}
```

در مرحله‌ی بعدی، برای این که در برنامه انعطاف بیشتری ایجاد شود و بتوان داده را از هر درگاه دلخواهی بخواند و بر روی یک درگاه دلخواه دیگر نمایش دهد، لازم است درگاه‌های مورد نظر به صورت آرگومان‌های ورودی به تابع **معرفی گردد.**

```
#include <mega16.h>
#define port_A 1
#define port_B 2
#define port_C 3
#define port_D 4
/*********************************************************/
void portfun(unsigned int port_in,unsigned int port_out)
{
    char data_in;

    switch( port_in)
```

برنامه 3-2

```

{
    case port_A:
        DDRA=0x00;      // as input
        data_in=PINA;
        break;
    case port_B:
        DDRB=0x00;      // as input
        data_in=PINB;
        break;
    case port_C:
        DDRC=0x00;      // as input
        data_in=PINC;
        break;
    case port_D:
        DDRD=0x00;      // as input
        data_in=PIND;
        break;
}
switch( port_out )
{
    case port_A:
        DDRA=0xFF;      // as output
        PORTA=data_in;
        break;
    case port_B:
        DDRB=0xFF;      // as output
        PORTB=data_in;
        break;
    case port_C:
        DDRC=0xFF;      // as output
        PORTC=data_in;
        break;
    case port_D:
        DDRD=0xFF;      // as output
        PORTD=data_in;
        break;
}
/*
 *****
void main(void)
{
while(1){
portfun(port_A,port_B);
}

}

```

برای ایجاد آرگومان خروجی نیز می‌توان تغییراتی را مانند برنامه 4-2 اعمال کرد.

```

#include <mega16.h>
#define port_A 1
#define port_B 2
#define port_C 3
#define port_D 4
*****

```

برنامه 4-2

```

unsigned int portfun(unsigned int port_in,unsigned int
port_out)
{
char data_in;

switch( port_in)
{
case port_A:
    DDRA=0x00;      // as input
    data_in=PINA;
break;
case port_B:
    DDRB=0x00;      // as input
    data_in=PINB;
break;
case port_C:
    DDRC=0x00;      // as input
    data_in=PINC;
break;
case port_D:
    DDRD=0x00;      // as input
    data_in=PIND;
break;
}
switch( port_out)
{
case port_A:
    DDRA=0xFF;      // as output
    PORTA=data_in;
break;
case port_B:
    DDRB=0xFF;      // as output
    PORTB=data_in;
break;
case port_C:
    DDRC=0xFF;      // as output
    PORTC=data_in;
break;
case port_D:
    DDRD=0xFF;      // as output
    PORTD=data_in;
break;
}
return 1; // can be any data with unsigned int format
}
/*
 ****
void main(void)
{
char data_out;
while(1){
data_out=portfun(port_A,port_B);
}
}

```

## 2.4 ایجاد فایل جانبی با قالب .c

برای ایجاد یک فایل جانبی با قالب .c. می‌توان از طریق منوی File/New/ Source File اقدام کرد. با انجام این کار فایل جدیدی ایجاد گردیده که می‌توان بدنه اصلی زیربرنامه‌های مورد نظر را درون آن پیاده‌سازی نمود. در این فایل مطابق برنامه 5-2 کلیه‌ی تعاریف و فایل‌های سرآیند مورد نیاز در همان ابتدای فایل معرفی می‌گردد.

```
#include <mega16.h>

#define port_A 1
#define port_B 2
#define port_C 3
#define port_D 4

برنامه 5-2
unsigned int portfun(unsigned int port_in,unsigned int port_out)
{
    .....
    .....
}
```

اکنون فایل جدید را با استفاده از منوی پیکربندی پروژه (Project/Configure)، پنجره‌ی Configuration Files/input files و دکمه add به پروژه اضافه می‌کنیم. سربرگ Project<project\_name>.prj

## 2.5 ایجاد فایل جانبی با قالب .h

این کار از طریق منوی File/New/ Source File قابل انجام است که در نتیجه آن فایل جدیدی ایجاد گردیده و با قالب .h. ذخیره می‌شود. بهتر است فایل‌های ایجاد شده دارای نام یکسانی با فایل نظرشان در قالب .c. بوده و هر دو در مسیر پروژه ذخیره شده باشند.

در این فایل مطابق برنامه 6-2 سرآیند زیربرنامه‌های پیاده‌سازی شده در فایل جانبی با قالب .c. آورده می‌شود.

```
#ifndef _portio_INCLUDED_
#define _portio_INCLUDED_

برنامه 6-2
unsigned int portfun(unsigned int port_in,unsigned int port_out);

#endif
```

بعد از ایجاد فایل‌های کمکی، همانند برنامه 7-2 در ابتدای فایل اصلی و پیش از تعریف تابع main، فایل سرآیند ایجاد شده فراخوانی می‌گردد و در ادامه می‌توان از زیربرنامه‌های تعریف شده در آن استفاده نمود. همان گونه که مشاهده می‌شود، با این کار حجم فایل اصلی به صورت قابل توجهی کاهش پیدا کرده است.

```
#include <mega16.h>
#include <portio.h>
```

```
#define port_A 1
#define port_B 2
#define port_C 3
#define port_D 4
```

برنامه 7-2

```
void main(void)
{
char data_out;
while(1) {
data_out=portfun(port_A,port_B);
}
}
```

## 2.6 ایجاد فایل سرآیند برای کل پروژه

تا اینجا نحوه ایجاد فایل‌های سرآیند برای کاهاش حجم برنامه اصلی و ایجاد قابلیت استفاده از کد نشان داده شد. اما برخی تعاریف اولیه مورد نیاز زیربرنامه‌ها هستند که به صورت همزمان در فایل‌های حابنی مختلف و نیز فایل اصلی پروژه مورد نیاز هستند. مثلاً تعاریف انجام شده در ابتدای فایل c. مربوط به برنامه 5-2 هم در فایل حابنی مورد استفاده هستند و هم برای فراخوانی زیربرنامه در فایل اصلی مورد نیاز می‌باشند. پس ناگزیر به اضافه کردن مجدد آن‌ها به فایل اصلی هستیم. بدیهی است که با افزایش حجم این تعاریف، میزان نظم و بهینه بودن کدهای پروژه کاهاش می‌یابد. لذا برای حل این مشکل از فایل جداگانه‌ای با قالب h. که حاوی تمام فراخوانی‌ها و تعاریف اولیه است استفاده می‌شود.

در برنامه 2-8 محتوای فایل سرآیند مذکور مشاهده می‌شود. این فایل در ابتدای فایل اصلی و تمام فایل‌های جانبی که به آن نیاز دارند، فراخوانی می‌شود.

```
#ifndef _header_INCLUDED_
#define _header_INCLUDED_
```

```
#include <mega16.h>
#include <portio.h>
```

برنامه 8-2

```
#define port_A 1
#define port_B 2
#define port_C 3
#define port_D 4
```

#endif

در برنامه 2-9 تغییرات مربوط به فایل‌های اصلی و جانبی آورده شده است.

فایل جانبی

#include &lt;header.h&gt;

برنامه 9-2

```
unsigned int portfun(unsigned int port_in,unsigned int
port_out)
{
..
..
```

```
}
```

فایل اصلی

```
#include <header.h>

void main(void)
{
...
}
```

## 2.7 تعریف متغیر سراسری

در یک پروژه گاهی به تعدادی متغیر سراسری نیاز است که بتوان از آن در تعدادی از زیربرنامه‌ها استفاده نمود. بدین منظور متغیر عمومی مورد نظر در فایل اصلی و قبل از تابع main برنامه 2-10 تعریف می‌گردد.<sup>۱</sup>

برنامه 10-2

```
#include <header.h>

int data=120; //global variable

void main(void)
{
...
}
```

همچنین این متغیر data را می‌توان در فایل سرآیند بدون مقداردهی اولیه و با افزودن عبارت «extern» مانند برنامه 2-11 تعریف نمود.

برنامه 11-2

```
#ifndef _header_INCLUDED_
#define _header_INCLUDED_

#include <mega16.h>
#include <portio.h>

#define port_A 1
#define port_B 2
#define port_C 3
#define port_D 4

extern int data;

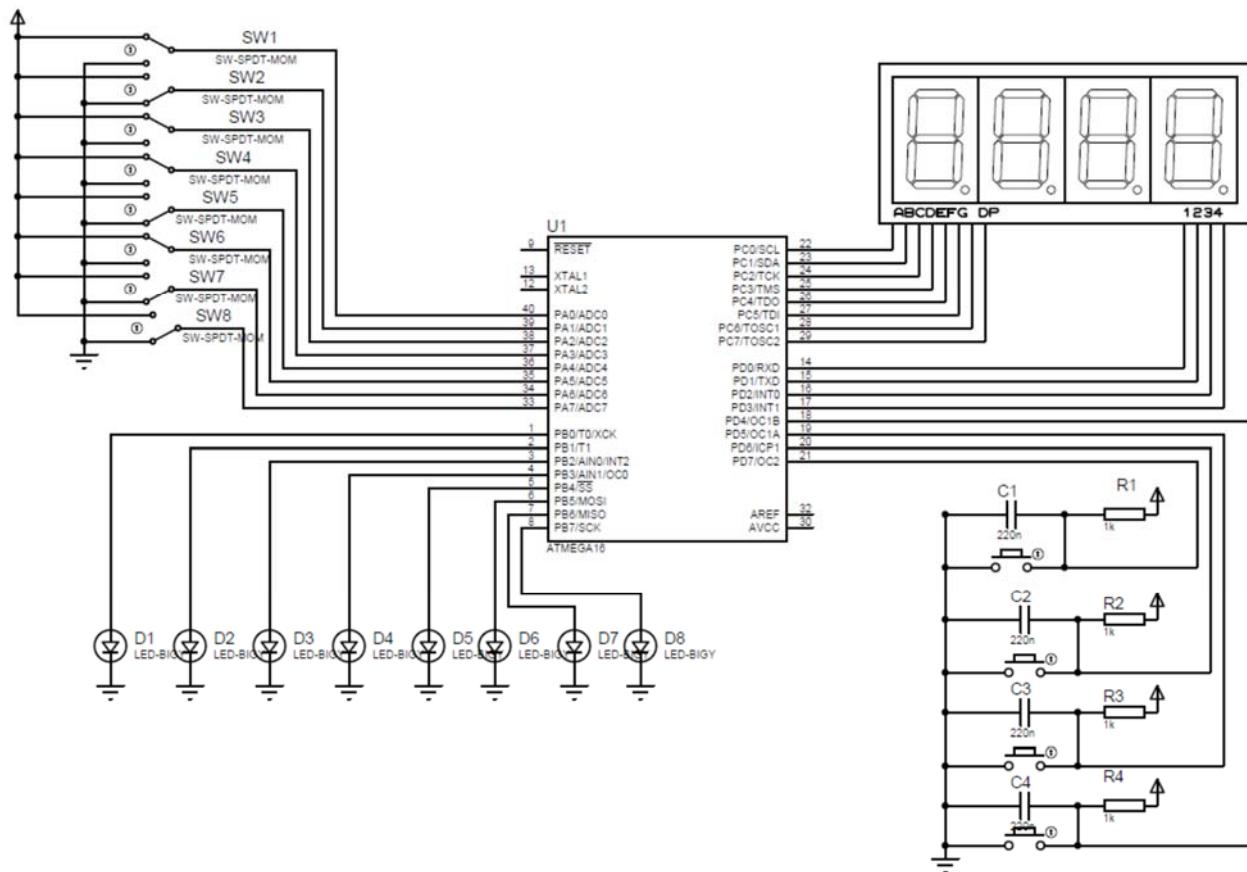
#endif
```

## 2.8 برنامه‌های اجرایی مربوط به زیربرنامه‌نویسی و فایل‌های جانبی

برای سختافزار نشان داده شده در شکل 2-1 برنامه‌های زیر را بنویسید.

---

<sup>۱</sup> استفاده از متغیرهای سراسری در برنامه‌نویسی کامپیوتر چندان توصیه نمی‌شود. اما در ریزپردازنده با توجه به محدود بودن حجم و ساده‌تر بودن کد و همچنین محدودیت‌های سختافزاری این کار با حساسیت کمتری صورت می‌پذیرد.



شکل 2-1

(1) زیربرنامه‌ای بنویسید که داده‌ای را از یکی از درگاه‌ها بخواند. در این زیربرنامه باید درگاه مورد نظر به عنوان آرگومان ورودی و داده خوانده شده به عنوان آرگومان خروجی در نظر گرفته شود.

Char part1(char port)

(2) زیربرنامه‌ای بنویسید که داده‌ای را بر روی یکی از درگاه‌ها بنویسد. در این زیربرنامه درگاه مورد نظر و داده، هر دو به عنوان آرگومان ورودی در نظر گرفته می‌شود.

(3) با استفاده از نتایج بند یک و دو، زیربرنامه‌ای بنویسید که با استفاده از آن همه‌ی LED‌ها خاموش و روشن شوند. دفعات و فاصله زمانی بین خاموش روشن شدن به عنوان آرگومان ورودی در نظر گرفته شود.

(4) با استفاده از نتایج بند یک و دو زیربرنامه‌ای بنویسید که مقدار تعیین شده توسط سوییچ‌ها را بر روی LED‌ها نمایش دهد.

(5) با استفاده از نتایج بند یک و دو، زیربرنامه‌ای بنویسید که عددی 4 رقمی را روی 7-segment نشان دهد. عدد مورد نظر و درگاه متصل به خطوط Enable و داده‌ی 7-segment به صورت آرگومان ورودی دریافت شوند.

(فرض نمایید همواره خطوط Enable به چهار بیت پایین در گاه مورد نظر متصل است). الگوی این زیربرنامه در زیر نشان داده شده است.

```
void part5(int data, char enable_port,char data_port)
```

(6) تمام زیر برنامه های توسعه داده شده را در یک فایل جداگانه به پروژه اضافه نمایید و همهی بندها به ترتیب اجرا شوند.

### 3 جلسه سوم

## آشنایی با LCD کاراکتری و صفحه کلید ماتریسی

### 3.1 هدف

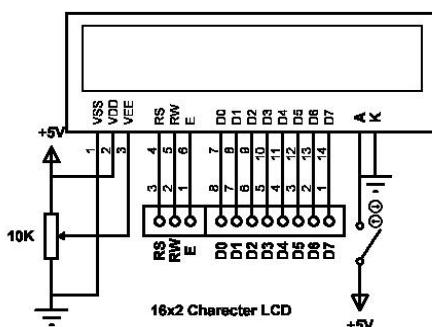
در این جلسه نمایشگر LCD و صفحه کلید ماتریسی به عنوان ادوات ورودی/خروجی متداول برای تعامل با کاربر بررسی می‌شوند. همچنین کار با صفحه کلید به دو روش سرکشی و نیز استفاده از وقفه‌ها صورت خواهد پذیرفت.

### 3.2 معرفی LCD کاراکتری

به طور کلی LCD کاراکتری به منظور نمایش جملات و اطلاعات نوشتاری دلخواه به کاربر مورد استفاده قرار می‌گیرد. در پکیج آموزشی یک عدد LCD کاراکتری از نوع 16x2 (دارای 16 ستون و 2 ردیف) با نور پس زمینه‌ی به رنگ سبز یا آبی تحت بلوکی با عنوان 16x2 Character LCD 16x2 تعبیه شده است.

#### 3.2.1 آشنایی با مدار راهانداز، پایه‌ها و حالت‌های کاری

مدار راهانداز این LCD در شکل 1-3 مشاهده می‌شود. پایه‌های K و A برای تنظیم نور زمینه تعبیه شده‌اند و با استفاده از کلید کشویی تعبیه شده می‌توان این نور پس زمینه‌ی را وصل یا قطع نمود.



شکل 3-1: مدار راهانداز LCD کاراکتری

پایه‌های D0 تا D7 پایه‌های انتقال داده هستند. پایه‌ی E فعال‌ساز لج داخلي است که برای ذخیره اطلاعات در LCD، لازم است یک سیگنال با لبه‌ی پایین رونده به آن اعمال شود. پایه‌ی R/W مشخص می‌کند که باید اطلاعات از LCD خوانده یا روی آن نوشته شود. پایه‌ی RS برای مشخص کردن این است که مقدار قرار گرفته روی D0 تا D7 باشد: در صورتی که RS=0 باشد این مقدار به عنوان دستور و در صورتی که RS=1 باشد به عنوان داده دستور است یا داده: در صورتی که RS=0 باشد این مقدار به عنوان دستور و در صورتی که RS=1 باشد به عنوان داده (کد اسکی) در نظر گرفته می‌شود. پایه‌های VEE، VDD و VSS هم به ترتیب تنظیم کننده‌ی وضوح، ولتاژ تغذیه و زمین هستند. در این مدار، در سر راه VEE یک پتانسیومتر قرار دارد تا کاربر بتواند وضوح نمایشگر را در حد مطلوب تنظیم نماید.

ارتباط ریزپردازنده و LCD را می‌توان به دو صورت برقرار کرد:

ارتباط 4 سیمه : سریال

ارتباط 8 سیمه : موازی

ارتباط 4 سیمه از تعداد پایه کمتری از ریزپردازنده را اشغال می‌کند و از این جهت بهتر بوده و معمولاً از آن استفاده می‌شود. در برنامه Codevision نیز چنان‌چه از CodeWizard برای آماده‌سازی LCD استفاده شود، همین روش 4 سیمه به کار گرفته می‌شود.

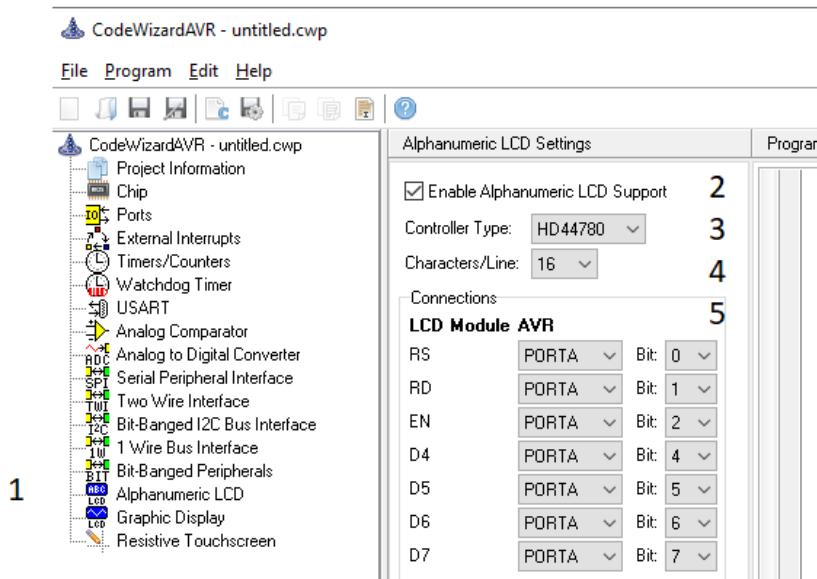
### 3.2.2 ارتباط 4 سیمه با ریزپردازنده

نحوه اتصال پایه‌های LCD به درگاه A ریزپردازنده بر روی پکیج آزمایشگاه در جدول 3-1 نشان داده شده است.

LCD	پایه‌ی ریزپردازنده
RS	PA.0
E	PA.1
(R/W)DB0*	PA.3
DB4-DB7	PA.4-PA.7

جدول 3-1: نحوه اتصال پایه‌های LCD به درگاه A ریزپردازنده (بر روی پکیج آموزشی پایه‌ی R/W از LCD به کانکتور با برچسب DB0 متصل شده است.)

بر اساس جدول بالا تنظیمات اولیه برای ارتباط 4 سیمه از طریق CodeWizard مانند شکل 3-2 انجام می‌شود.



1: انتخاب Alphanumeric LCD

2: فعال نمودن LCD

3: انتخاب کنترلر LCD

4: انتخاب تعداد کاراکتر در هر خط

5: تنظیمات اتصال درگاه میکرو به پایه‌های LCD (دقت شود که در تنظیمات کدوبین RD همان R/W است)

شکل 2-3: نمایی از تنظیمات کدویزر برای LCD

پس از انجام مراحل مذکور، کدهای ایجاد شده برای LCD مطابق برنامه 1-3 ایجاد می‌گردند.

برنامه 1-3

```
#include <mega16.h>
#include <alcd.h>
void main(void)
{
    DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2) |
(0<<DDA1) | (0<<DDA0);
    PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) |
(0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

    DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) |
(0<<DDB1) | (0<<DDB0);
    PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) |
(0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

    DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) |
(0<<DDC1) | (0<<DDC0);
    PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) |
(0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);
```

```

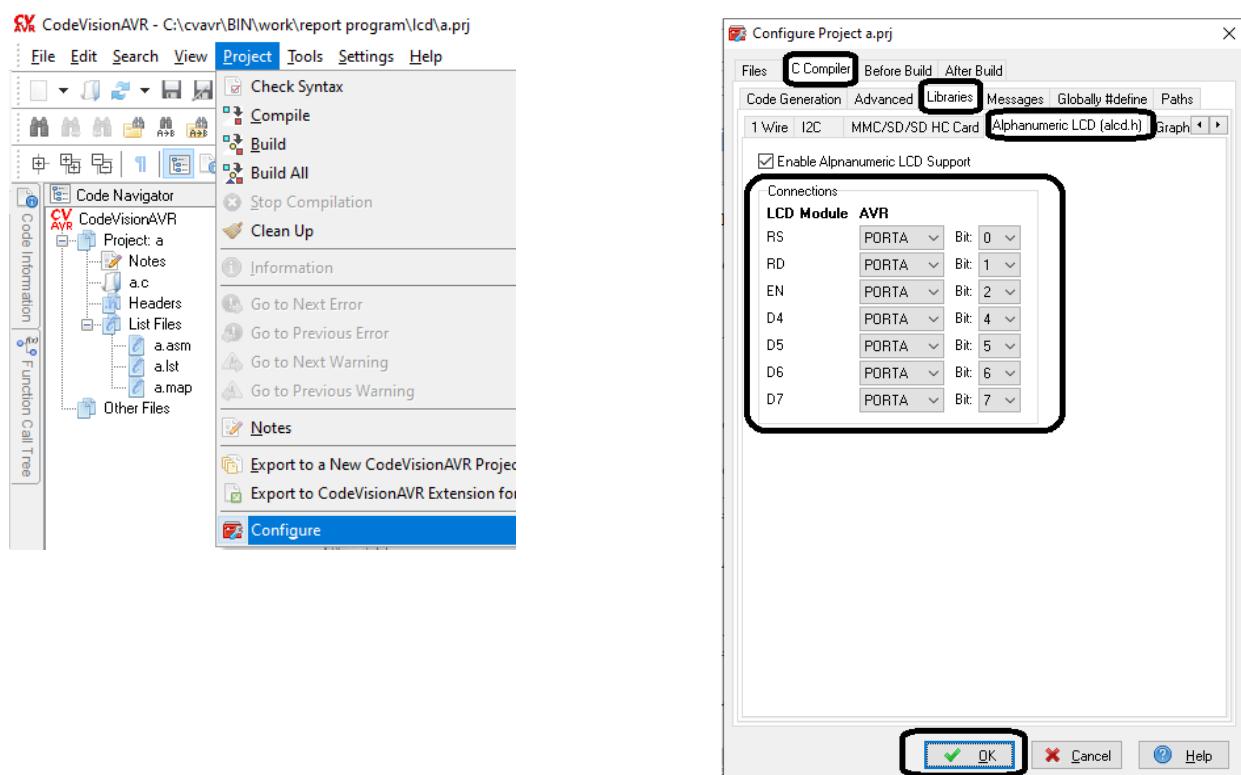
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) |
(0<<DDD1) | (0<<DDD0);
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) |
(0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init;(16)

while (1)
{
}

```

همچنین لازم به ذکر است که اگر در حین انجام پروژه سخت‌افزار تغییر نمود می‌توان مانند شکل 3-3 تغییرات را لحاظ نمود.



شکل 3-3: تغییر تنظیمات LCD

### 3.2.3 آشنایی با فایل سرآیند «alcd.h»

این فایل سرآیند حاوی دستورات آماده برای کار با LCD در محیط نرم افزار CodeVision است. بنا به نسخه مورد استفاده Codevision، ممکن است تفاوت‌های جزئی در دستورات و آرگومان‌های این فایل وجود داشته باشد. لذا برای اطلاع از این تغییرات به راهنمای کاربری نرم افزار مراجعه نمایید. تعدادی از دستورات متداول در جدول 3-2 شرح داده شده‌اند.

---

```
void lcd_init(unsigned char lcd_columns)
```

برای تنظیمات و پیکربندی اولیه LCD به کار می‌رود که در آن تعداد کاراکتر قابل نمایش در هر خط به عنوان ورودی دریافت می‌شود.

---

```
void lcd_clear(void)
```

محتوای نمایش داده شده بر روی LCD را پاک می‌کند.

---

```
void _lcd_ready(void)
```

اجرای ادامه کرد را منتظر آماده شدن LCD نگه می‌دارد.

---

```
void lcd_putchar(char c)
```

کاراکتری که به عنوان ورودی دریافت شده را در موقعیت جاری مکان‌نما بر روی LCD نمایش می‌دهد.

---

```
void lcd_puts(char *str)
```

رشته ورودی را در موقعیت جاری مکان‌نما بر روی LCD نمایش می‌نماید.

---

```
void lcd_putsf(char flash *str)
```

این تابع نیز رشته ورودی را بر روی LCD نمایش می‌دهد. اما توجه کنید که وقتی از این تابع استفاده می‌کنیم که رشته مورد نظر درون flash قرار داشته باشد.

---

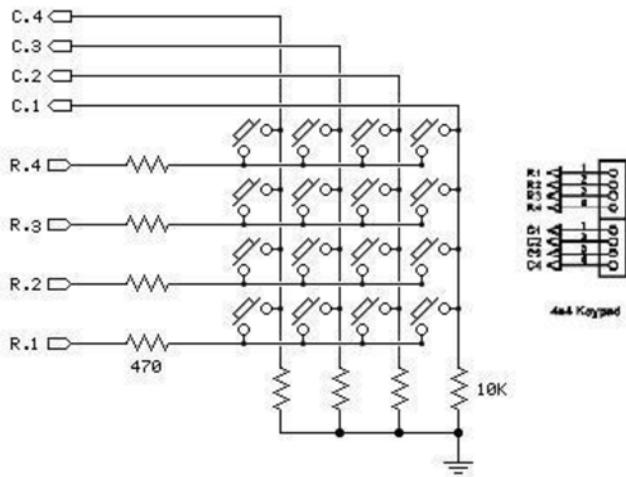
```
void lcd_gotoxy(unsigned char x, unsigned char y)
```

مکان‌نما را به مختصات تعیین شده با x و y منتقل می‌کند.

جدول 3-2. شرح برخی دستورات موجود در کتابخانه alcd.h

### 3.3 آشنایی با صفحه کلید ماتریسی

صفحه کلید یکی از متداول‌ترین ادوات ورودی برای دریافت داده‌ها از سوی کاربر می‌باشد. در پکیج آموزشی یک عدد صفحه کلید ماتریسی شامل 16 عدد کلید فشاری که چیدمان آن‌ها به صورت 4x4 (4 ردیف و 4 ستون) می‌باشد قرار داده شده است. شماتیک مربوط به مدار keypad در شکل 3-4 مشاهده می‌شود.



شکل ۴-۳: مدار صفحه کلید ماتریسی استفاده شده در بورد آموزشی

برای کار با پایه‌های آن به یکی از درگاه‌های ریزپردازنده متصل می‌شود به این صورت که سطرهای R1 تا R4 به صورت خروجی و ستون‌های C4 تا C1 به صورت ورودی تعریف می‌گردند. سپس مقدار متناظر هر کلید به صورت یک آرایه در ریزپردازنده ذخیره می‌گردد. به عنوان مثال، در برنامه ۳-۲، ماتریس data\_key به ازای هر یک از کلیدهای keypad یک مقدار متناظر را در برگرفته است.

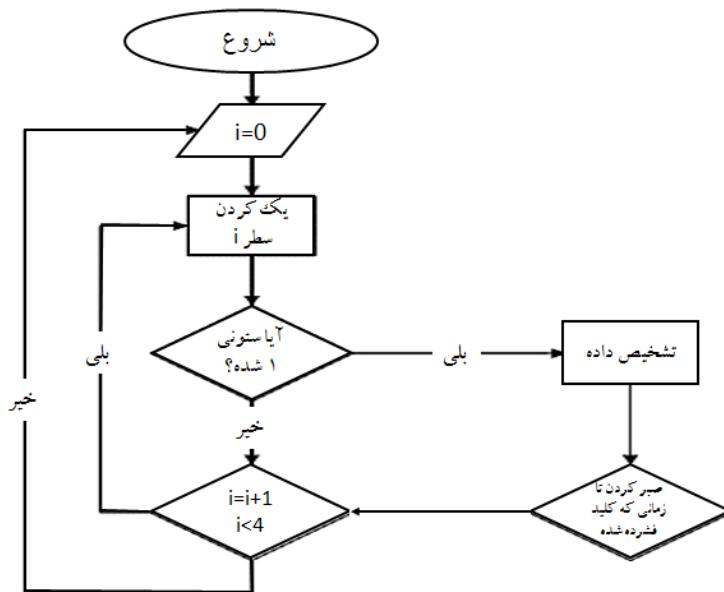
برنامه ۳-۲

```
char data_key [] = {
    '1', '2', '3', '4',
    '5', '6', '7', '8',
    '9', '#', '0', 'a',
    'b', 'c', 'd' };
```

در ادامه مانند شکل ۵-۳ ۵ سطرهای ۱ گردیده و مقدار ستون‌ها خوانده می‌شود. هر جا که مقدار ستونی ۱ شده باشد، یعنی کلید مربوط به آن سطر و ستون فشرده شده است. فرض کنید کلید ۶ فشرده شده باشد، در این صورت با یک شدن پایه نظیر سطر دوم، مقدار متناظر با ستون سوم نیز مقدار ۱ و سایر ستون‌ها مقدار ۰ را برمی‌گردانند. با پیدا شدن سطر و ستون، می‌توان از طریق ماتریس ذخیره شده در ریزپردازنده، به محتوای کلید دست پیدا کرد.

`data_key [(2 - 1) * 4 + (3 - 1)] = '6'`

این روش کار با صفحه کلید را روش سرکشی می‌نامند که یک نمونه کد برای تعیین کلید فشرده شده به این طریق در برنامه ۳-۳ نشان داده شده است.



شکل 3-5: الگوریتم کار با صفحه کلید ماتریسی به روش سرکشی

برنامه 3-3

```

char keypad2(void)
{
    char key=100;
    for (r=0;r<4;r++)
    {
        PORTB=row[r]; //row= 0x10,0x20,0x40,0x80

        c=20;
        delay_ms(10);
        if (PINB.0==1) c=0;
        if (PINB.1==1) c=1;
        if (PINB.2==1) c=2;
        if (PINB.3==1) c=3;

        if (! (c==20))
        {
            key=(r*4)+c;
            PORTB=0xf0;
            while (PINB.0==1) {}
            while (PINB.1==1) {}
            while (PINB.2==1) {}
            while (PINB.3==1) {}

            PORTB=0xf0;
        }
        return key;
    }
}

```

### 3.4 وقفه‌ها در ریزپردازنده

رسیدگی به رویدادهای مربوط به ادوات جانبی و نیز رویدادهای خارجی در ریزپردازنده‌ها به یکی از دو روش زیر قابل انجام است:

- روش سرکشی (Polling): در این روش که در بخش قبل نیز مورد استفاده قرار گرفت، برنامه‌نویس پردازنده را طوری برنامه‌ریزی می‌کند که موقع رویداد مورد نظر با فواصل زمانی مشخص و پی‌درپی مورد بررسی قرار گیرد و در صورت وقوع رویداد، به آن پاسخ می‌دهد. در این روش زمان پردازنده در خیلی از موارد برای بررسی وقوع اتفاق احتمالی، به هدر می‌رود.
- روش استفاده از وقفه (Interrupt): وقفه امکانی در ریزپردازنده است که باعث می‌شود هسته پردازشی در قبال ایجاد یک رویداد لحظه‌ای (که معمولاً زمان آن قابل پیش‌بینی نیست) عملیات خاصی را در قالب یک زیربرنامه مشخص، انجام دهد. در روش استفاده از وقفه، برنامه اصلی در حالت عادی خود اجرا می‌شود، اما به محض وقوع رویداد مورد نظر، پردازنده دستور جاری برنامه را به انتها رسانده و اجرای بقیه‌ی برنامه را متوقف می‌نماید. سپس به صورت سختافزاری به ابتدای زیربرنامه وقفه<sup>۱</sup> پرس کرده و پس از اجرای آن، ادامه‌ی برنامه اصلی اجرا می‌گردد.

در استفاده از وقفه می‌توان درخواست وقفه از وسایل جانبی مختلف را اولویت بندی نمود و بر اساس این اولویت‌ها و باحتی به صورت کلی وقوع برخی وقفه‌ها را نادیده گرفت. در ادامه به تعدادی از مزایای استفاده از وقفه اشاره شده است.

1. رسیدگی بی‌درنگ به درخواست وقفه (در صورت فعل نبودن یک درخواست وقفه با اولویت بالاتر)
2. عدم اتلاف زمان پردازنده در زمان‌هایی که درخواستی برای پردازش موجود نیست.

#### 3.4.1 راه اندازی وقفه در AVR

برای فعال‌سازی وقفه‌ها در ریزپردازنده‌های AVR باید بیت هفتم از ثبات SREG (فعال ساز عمومی) را فعال کرد. پس از فعال کردن این بیت می‌توان هر کدام از وقفه‌های موجود در AVR را از طریق ثبات‌های مربوطه فعال کرد. در زبان C با نوشتن عبارت (#asm("sei")) این بیت فعل می‌شود.

جدول 3-3 وقفه‌های ریزپردازنده Atmega16/32 لیست گردیده است.

<sup>1</sup> Interrupt subroutine

جدول 3-3 : وقفه‌های ریزپردازنده Atmega16/32

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	\$000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$002	INT0	External Interrupt Request 0
3	\$004	INT1	External Interrupt Request 1
4	\$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	\$008	TIMER2 OVF	Timer/Counter2 Overflow
6	\$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	\$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	\$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	\$010	TIMER1 OVF	Timer/Counter1 Overflow
10	\$012	TIMER0 OVF	Timer/Counter0 Overflow
11	\$014	SPI, STC	Serial Transfer Complete
12	\$016	USART, RXC	USART, Rx Complete
13	\$018	USART, UDRE	USART Data Register Empty
14	\$01A	USART, TXC	USART, Tx Complete
15	\$01C	ADC	ADC Conversion Complete
16	\$01E	EE_RDY	EEPROM Ready
17	\$020	ANA_COMP	Analog Comparator
18	\$022	TWI	Two-wire Serial Interface
19	\$024	INT2	External Interrupt Request 2
20	\$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	\$028	SPM_RDY	Store Program Memory Ready

همان‌طور که در جدول فوق مشاهده می‌شود سه مورد از وقفه‌ها، تحت عنوان External interrupt بوده (وقفه‌های INT0, INT1, INT2) و به آن‌ها وقفه‌های خارجی گفته می‌شود. از وقفه‌های خارجی معمولاً برای تشخیص پالس بر روی پایه‌های تراشه استفاده می‌شود. این پالس می‌تواند حاوی اطلاعات خاصی مانند اطلاعات دریافتی از یک حسگر یا IC باشد که به یکی از پایه‌های PB2 ، PD2 و یا PD3 متصل است. غیر از این سه وقفه و البته وقفه Timer/Counter capture interrupt، سایر وقفه‌ها داخلی هستند. در این جلسه وقفه‌های خارجی مورد بررسی قرار می‌گیرند و سایر وقفه‌ها در جلسات بعدی و همزمان با سخت‌افزار جانبی مربوطه تشریح خواهند شد.

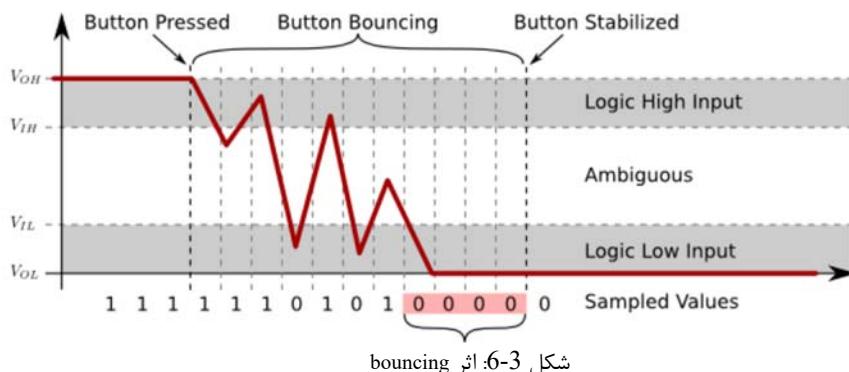
برای زیربرنامه‌ی هر وقفه، فضای مشخصی در نظر گرفته شده است که کدهای مورد نظر باید در این فضا نوشته شوند. این فضاهای تحت عنوان شماره بردار وقفه در جدول 3-3 مشخص شده‌اند. البته در فایل سرایند.h، برای هر کدام از این شماره‌ها نامهای معادلی در نظر گرفته شده که استفاده از وقفه‌ها را آسان‌تر می‌نماید. مثلاً برای استفاده از وقفه خارجی 1 به جای نوشتمن شماره 3 می‌توان عبارت EXT\_INT1 را به کار برد. این نحوه استفاده در برنامه 3-4 نشان داده شده است. در این مثال زیربرنامه‌ی ext\_int1\_isr در فضای مربوط به وقفه EXT\_INT1 ذخیره می‌شود.

## برنامه 4-3

```
Interrupt void void() { }  
interrupt [EXT_INT1] void ext_int1_isr(void){...}
```

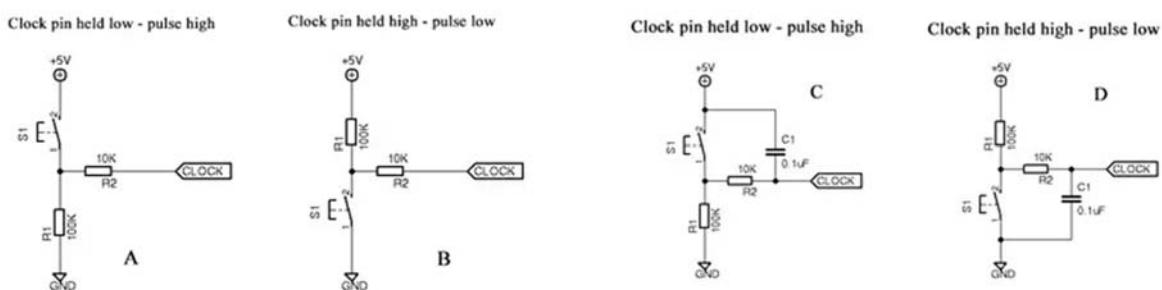
روی بورد موجود در آزمایشگاه تعدادی کلید Push-Botton وجود دارد که می‌توان از آن به عنوان منبع ایجاد وقفه خارجی استفاده نمود. کلیدی است که در حالت عادی قطع است. مادامی که توسط کاربر فشار داده شود، وصل خواهد بود و به محض رها شدن، دوباره قطع خواهد شد.

یکی از مشکلات رایج در Push-Button‌ها پدیده Bouncing است. این پدیده در اثر لرزش اتصالات مکانیکی کلیدها در هنگام قطع و وصل شدن، رخ می‌دهد. چنین لرزش‌هایی موجب می‌شود پیش از رسیدن کلید به حالت دائمی، چند بار صفر و یک شود. به عنوان مثال در شکل زیر بعد از فشرده شدن Push-Button تا زمانی که به حالت ثابت صفر برسد چند صفر و یک دیگر تولید شده و این باعث ایجاد خطأ در خروجی می‌شود.



شکل ۶-۳: اثر bouncing

یکی از روش‌های حل این مشکل ایجاد تاخیر زمانی و صبر کردن تا زمان پایدار شدن وضعیت کلید است. زمان رسیدن کلید به حالت دائمی معمولاً بین 10 تا 20 میلیثانیه می‌باشد. این کار را می‌توان با استفاده از مدارهای RC مانند شکل ۷-۳ انجام داد تا نویز به وجود آمده را فیلتر نماید.

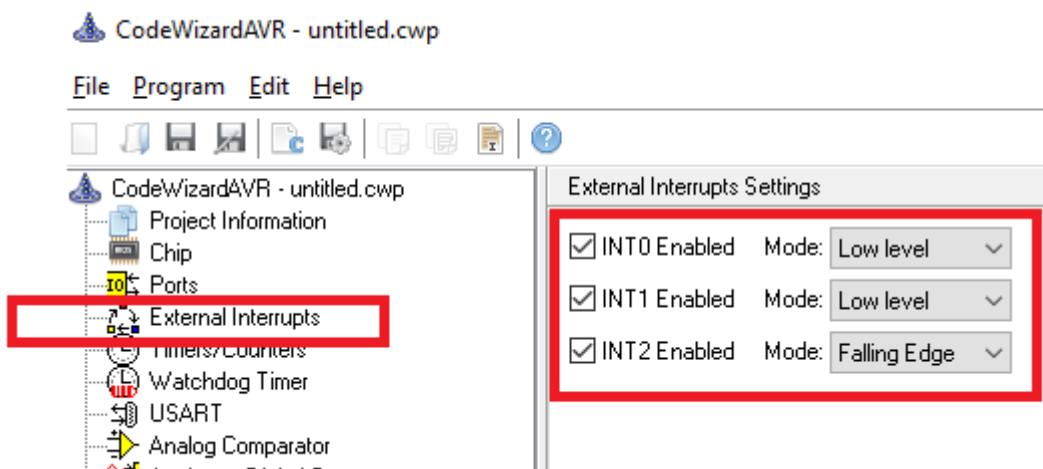


شکل ۷-۳: رفع پدیده bouncing با استفاده از مدار RC

### 3.5 پردازش صفحه کلید با وقفه

همان طور که در بخش قبلی به اهمیت وقفه‌ها در پیشگیری از اتلاف بیهوده وقت پردازنده اشاره شد، برای پردازش صفحه کلید نیز می‌توان از وقفه‌های خارجی استفاده کرد. بدین منظور لازم است از سخت‌افزار جانبی شامل دروازه‌های منطقی استفاده کرد تا با فشردن هر کلید، وقفه‌ای خارجی نیز رخ دهد و پروسه پردازش صفحه کلید در روتین وقفه مربوطه ادامه یابد. در این حالت لازم است مقدار مناسب در درگاه متصل به صفحه کلید نوشته شود و این کار بعد از اتمام وقفه نیز تکرار گردد. این مقدار بنا به سخت‌افزار طراحی شده برابر 0xF0 خواهد بود.

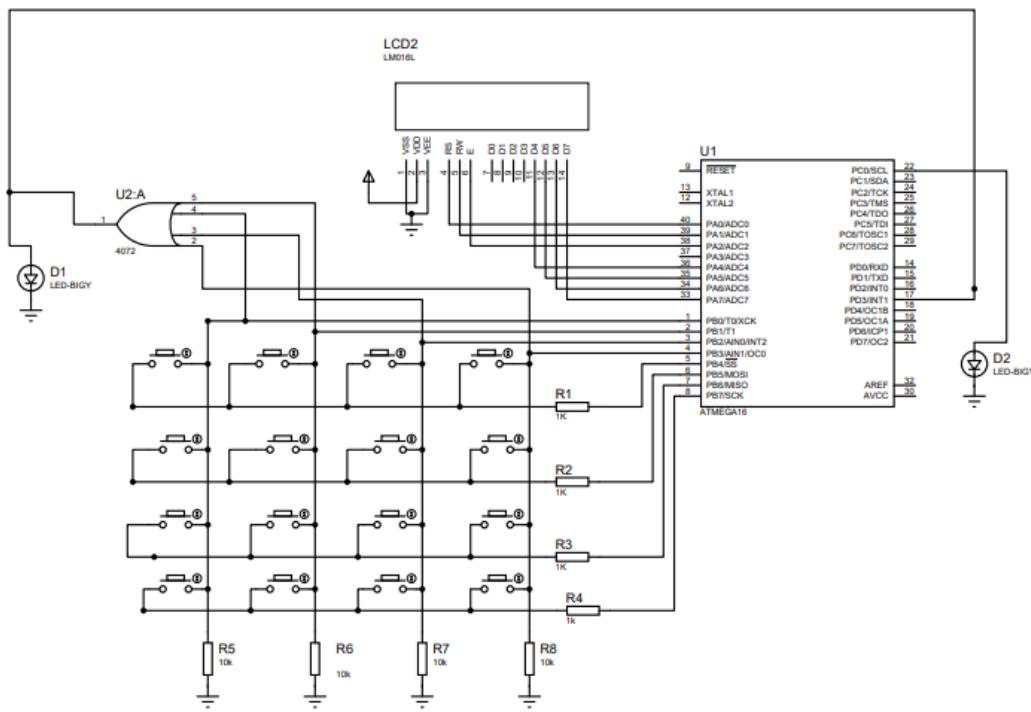
برای فعال کردن وقفه‌های خارجی می‌توان از قابلیت‌های CodeWizard در کد ویژن مانند شکل 8-3 استفاده کرد و حساسیت به لبه یا سطح را برای رخداد وقفه را تعیین نمود.



شکل 8-3: تنظیمات وقفه‌های خارجی در CodeWizard

### 3.6 برنامه‌های اجرایی مبحث LCD و صفحه کلید

برای سخت‌افزار نشان داده شده در شکل زیر، برنامه‌های خواسته شده را بنویسید.



1. نام خانوادگی در خط اول و شماره دانشجویی در خط دوم LCD کاراکتری نمایش داده شود.
2. عبارت زیر به صورت روان روی lcd نمایش داده شود. سرعت حرکت باید به گونه‌ای باشد که قابل دیدن و خواندن باشد.

“Welcome to the Microprocessor Laboratory at Isfahan University of Technology.”

3. زیر برنامه‌ای بنویسید که صفحه کلید زیر را به روش سرکشی برای تشخیص کلید فشرده شده اسکن و مقدار نظیر کلید فشرده شده را روی lcd نمایش دهید.

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

4. صفحه کلید فوق را با استفاده از وقفه خارجی اسکن و مقدار نظیر کلید فشرده شده را روی lcd نمایش دهید. (در این بند، زیر برنامه نوشته شده در بند قبلی، در زیر برنامه وقفه خارجی فراخوانی می‌گردد.)
5. داده‌های اولیه یک سیستم شامل سرعت، زمان، وزن و دما از طریق صفحه کلید دریافت می‌شود. بدین منظور ابتدا هر یک از پیام‌های ذیل روی LCD نمایش داده شده و مقدار اولیه متناظر از طریق صفحه

کلید دریافت و به جای عبارت ؟؟ نمایش داده می شود. اگر عدد دریافتی خارج از محدوده باشد، در این محل عبارت EE نمایش داده می شود و منتظر اصلاح عدد می ماند. اما چنانچه عدد در بازه مورد نظر باشد پیام بعدی نمایش داده خواهد شد و نهایتاً پیام پایان فراخوانی چاپ خواهد شد.

Speed:??(0-50r)

Time:??(0-99s)

W:??(0-99Kg)

Temp:??(20-80C)

End

## 4 جلسه چهارم

### آشنایی با وقفه‌ها و تایمرها

#### 4.1 هدف

معمولًاً تعداد زیادی از کارهای کنترلی، نیازمند اندازه‌گیری زمان یا شمارش یک اتفاق هستند. برای پیاده‌سازی چنین عملیات‌هایی، در ریزپردازندۀ‌ها یک یا چند تایمر<sup>1</sup> تعبیه شده است. در این جلسه نحوه کار با تایمرها، حالت‌های کاری مختلف آن‌ها و وقفه‌های مرتبط مورد بحث قرار خواهد گرفت.

#### 4.2 مقدمه

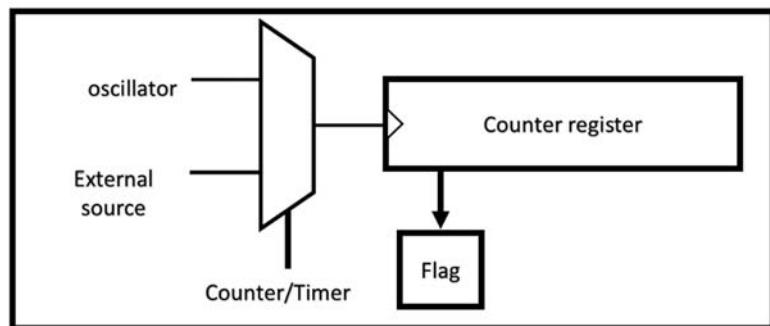
تایمرها در حقیقت شمارنده‌های سختافزاری مجازی هستند که در صورت فعال بودن به طور موازی با پردازش‌های CPU عمل شمارش را انجام داده و مقدار آن‌ها افزایش یا کاهش می‌یابد. از این رو تایمرها مهم‌ترین ابزار برای سنجش زمان در ریزپردازندۀ‌ها می‌باشند.

واحد تایمر مطابق شکل 4-1 با هر پالس ساعت یک واحد می‌شمارد. به طور مثال یک ریزپردازنده با فرکانس پالس ساعت 1MHz را در نظر بگیرید. اگر همین پالس ساعت ریزپردازنده را مستقیماً به تایмер وصل کنیم، محتوای ثبات Counter در هر یک میکروثانیه یک عدد افزایش می‌یابد. لذا برای تاخیر به اندازه 100 میکروثانیه بایستی منتظر ماند تا مقدار شمارنده تایمر از صفر به 100 افزایش یابد. لازم به ذکر است که منبع شمارش تایمر می‌تواند پالس ساعت داخلی CPU یا یک پالس خارجی باشد. چنان‌چه منبع پالس خارجی به کار رود، می‌توان از تایمرها به عنوان شمارنده پالس‌های خارجی ریزپردازنده نیز استفاده نمود.

ریزپردازنده Atmega16/32 سه تایمر دارد که تایمرهای صفر و 2، هشت بیتی و تایمر 1 شانزده بیتی است. تعداد بیت‌های ذکر شده مشخص کننده بازه قابل شمارش و یا در واقع حد بالای شمارنده تایمر هستند.

<sup>1</sup> Timer

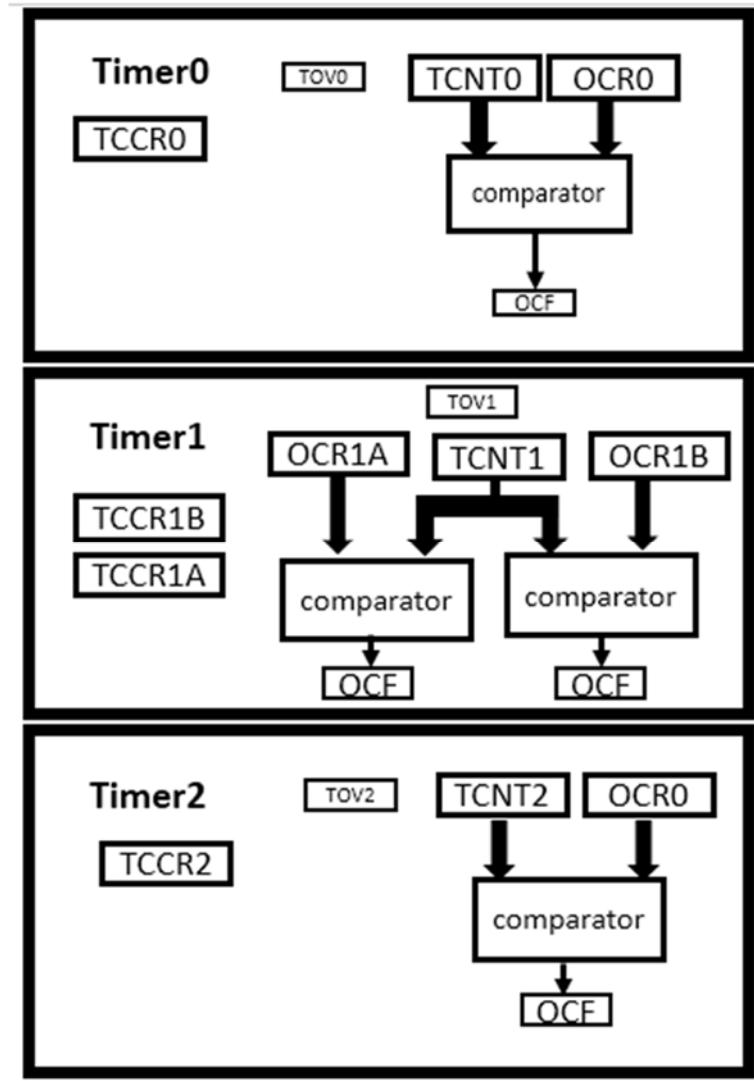
<sup>2</sup> Counter



شکل 4-1: نمایی از ساختار کلی تایمر

#### 4.3 ثبات‌های تایمر

ثبتات‌های هر یک از تایمرها در شکل 4-2 نشان داده شده است.



شکل 4-2. ثبات‌های هر یک از تایمروها

تنظیم تایمروها و استفاده از آن‌ها به وسیلهٔ ثبات‌های مربوطه انجام می‌شود. این ثبات‌ها در زیر معرفی شده‌اند. حرف n شمارهٔ تایمر را مشخص می‌کند و در ریزپردازنده Atmega16/32 می‌تواند صفر، یک یا دو باشد.

ثبات TCCR<sup>1</sup><sub>n</sub>: در این ثبات پیکربندی تایمر انجام می‌شود. بیت‌های کنترلی تایمر صفر در شکل 4-3 نشان داده شده است.

7	6	5	4	3	2	1	0	TCCR0
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

شکل 4-3: معرفی بیت‌های ثبات کنترلی Timer 0

<sup>1</sup> Timer/Counter Control Register

نحوهی کار با بیت‌های این ثبات در جدول‌های زیر آمده است (برای جزئیات بیشتر به Datasheet مراجعه نمایید).

جدول 4-1: تنظیم حالت کاری تایمر صفر با استفاده از بیت‌های WGM01 و WGM00

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

جدول 4-2: تنظیم نحوهی فعال شدن پایهی OC0 در حالت غیر PWM با استفاده از بیت‌های COM01 و COM00

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

جدول 4-3: تنظیم نحوهی فعال شدن پایهی OC0 در حالت Fast PWM با استفاده از بیت‌های COM01 و COM00

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (non-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

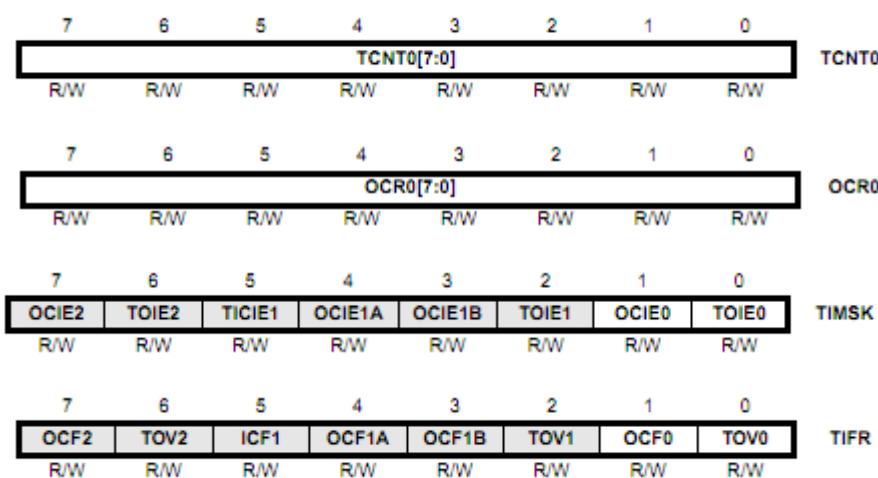
جدول 4-4: تنظیم نحوهی فعال شدن پایهی OC0 در حالت Phase Correct PWM با استفاده از بیت‌های COM01 و COM00

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

جدول 5-4: تنظیم فرکانس شمارنده با استفاده از بیت‌های CS02 و CS01 و CS00

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk <sub>I/O</sub> /(No prescaling)
0	1	0	clk <sub>I/O</sub> /8 (From prescaler)
0	1	1	clk <sub>I/O</sub> /64 (From prescaler)
1	0	0	clk <sub>I/O</sub> /256 (From prescaler)
1	0	1	clk <sub>I/O</sub> /1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

سایر ثبات‌های مربوط به تایمر صفر در شکل 4-4 نشان داده شده است.



شکل 4-4: ساختار ثبات‌های TIFR, TIMSK, OCR0, TCNT0

**ثبات TCNT<sub>n</sub>:** این ثبات مقدار اصلی شمارنده یا تایمر را در هر لحظه درون خود نگه می‌دارد. به محض راهاندازی مجدد، محتوای آن صفر می‌شود و از آن پس با هر پالسی که به آن وارد می‌شود یکی می‌شمارد. همچنان می‌توان مقداری را در آن ذخیره کرده یا از روی آن خواند.

**ثبات OCR<sub>n</sub>:** محتوای این ثبات با محتوای TCNT<sub>n</sub> مقایسه می‌شود. اگر با هم برابر باشند پرچم OCF<sub>n</sub> یک خواهد شد.

<sup>1</sup> Timer/Counter Register

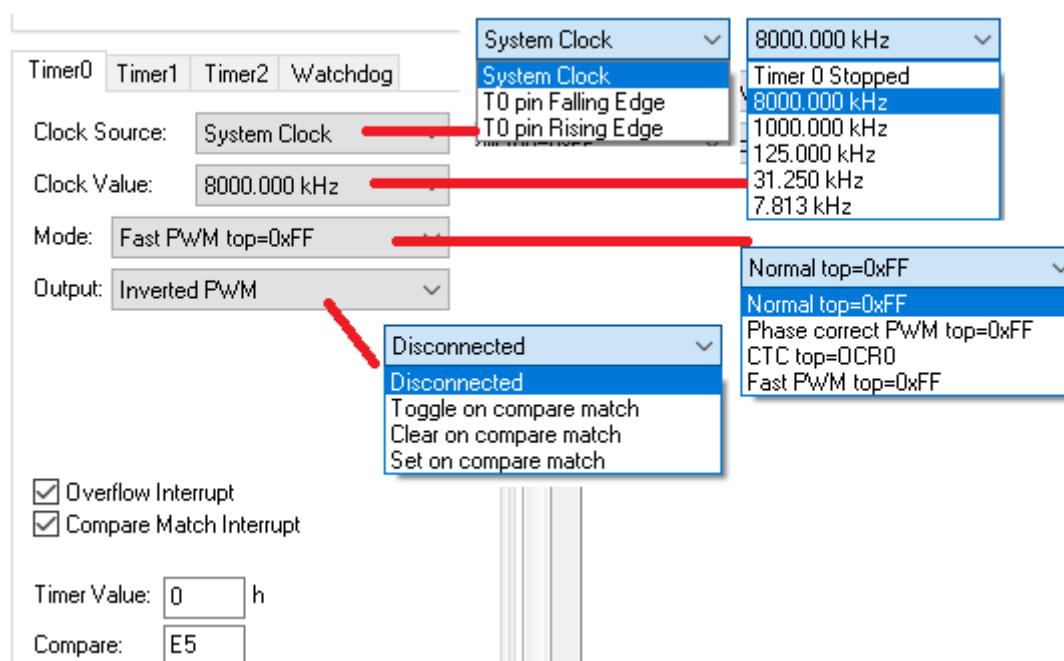
<sup>2</sup> Output Compare Register

ثبات **TIMSK**<sup>۱</sup>: وقفه‌های تایمر صفر، یک و دو را در شرایط سرریز کردن و یا برابر شدن محتوای ثبات‌های **TCNTn** و **OCRn**، به ترتیب با بیت‌های **TOIE<sub>n</sub>** و **OCIE<sub>n</sub>** فعال می‌کند.

ثبات **TIFR**<sup>۲</sup>: هنگامی که وقفه‌های مورد نظر از تایمر فعال باشد، رخداد این وقفه‌ها در ثبات **TIFR** قابل مشاهده است. هرگاه سرریز اتفاق بیافتد پرچم سرریز (**TOV<sub>n</sub>**) برابر با یک می‌گردد و با برابر شدن ثبات‌های **TCNn** و **OCRn** پرچم مقایسه‌ی خروجی (**OCFn**) یک خواهد شد.

#### 4.4 تنظیمات **CodeWizard** برای تایمر

پارامترهایی که در تنظیم تایمر در محیط **Codevision** قابل پیکربندی هستند در شکل ۴-۵ نشان داده شده است.



شکل ۴-۵: تنظیمات تایمر در **CodeWizard**

<sup>۱</sup> Timer/Counter Interrupt Mask Register

<sup>۲</sup> Timer/Counter Interrupt Flag Register

-1 - **Clock Source**: همان گونه که گفته شد، منبع پالس ساعت تایمر می‌تواند پالس ساعت داخلی ریزپردازنده باشد یا این یکی از پایه‌های ریزپردازنده برای شمارش استفاده شود و تایمر به یک شمارنده تبدیل گردد (در این حالت هر زمان یک پالس به پایه‌ی مشخصی از ریزپردازنده اعمال شود، مقدار شمارنده یک واحد زیاد می‌شود و این زمان می‌تواند دوره تناوب مشخصی نداشته باشد).

-2 - **Clock Value**: در صورتی که برای تایمروها، System Clock از نوع Clock Source انتخاب شود، فرکانس تایمر توسط Clock Value تعیین می‌شود. با این کار فرکانس تایمر، مضربی از فرکانس ریزپردازنده خواهد بود (با ضریب کمتر از 1). این فرآیند Prescale نامیده می‌شود. در حالتی که در تنظیمات Clock Source تایmer را به Counter تبدیل کنیم، تنها Timer2 قابلیت Prescale کردن را خواهد داشت. برای به دست آوردن فرکانس تایمر در حالت Prescale (با مضرب N) از رابطه زیر استفاده می‌شود:

$$\text{Prescale: } N=1,8,64,256,1024 \rightarrow F_{\text{Timer-Clock}} = \frac{F_{osc}}{N}$$

-3 - **حالت کاری**: تایمروها می‌توانند شمارش را به انواع مختلف و تا مقادیر متفاوتی انجام دهند. چهار حالت قبل استفاده شامل Phase Correct PWM و Fast PWM، Normal و CTC می‌باشند که هر کدام از این مدها در بخش‌های بعدی توضیح داده خواهد شد.

-4 - **Output**: تایمروها می‌توانند بر روی پایه‌های مشخصی از تراشه با عنوانین OCn (که n در آن شماره‌ی تایمر است) پالس‌هایی را به طور خودکار ایجاد کنند. این قابلیت برای زمانی که پالس‌های دقیق و منظم مورد نیاز است استفاده می‌شود.

-5 - **Input Capt**: امکان ثبت زمان رخدادهای خارجی با استفاده از تایمر 16 بیتی شماره یک فراهم می‌شود.

-6 - **Interrupt on**: نوع وقفه مورد نظر از تایمر انتخاب می‌شود.

-7 - **Value**: در این بخش مقدار اولیه برخی ثبات‌ها ثبت می‌شود.

#### 4.5 حالت‌های کاری تایمر

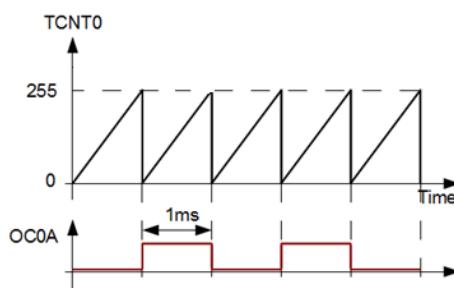
همان گونه که توضیح داده شد چهار حالت کاری مختلف برای تایمروها قابل انتخاب می‌باشد که در ادامه هر کدام از این حالت‌ها توضیح داده شده‌اند.

##### 4.5.1 Normal حالت

در حالت Normal شمارش تایمر تا بالاترین مقداری که تایمر می‌تواند بشمارد ادامه می‌یابد (برای تایمر هشت بیتی تا 255 و برای تایمر شانزده بیتی تا 65535) و بعد از آن دوباره تایمر از صفر شروع به شمارش می‌کند. هر بار

که این سرریز اتفاق می‌افتد، خروجی بر روی پایه OCn تغییر وضعیت داده و یک پالس مربعی ایجاد می‌کند که فرکانس آن برای شمارنده m بیتی از فرمول زیر به دست می‌آید:

$$F_{Generatedwave} = \frac{F_{TimerClock}}{2^{m+1}}$$



شکل 4-6- نحوه شمارش و فعال شدن بیت OC0A در حالت شمارش Normal

همان طور که در شکل 4-6 مشاهده می‌شود مقدار OCR0A با هر سرریز تایмер 8 بیتی شماره 1، تغییر وضعیت می‌دهد.

در شکل فوق چرخه کار (Duty Cycle) برابر با 50 درصد است، چون مدت زمان یک و صفر بودن خروجی در یک دوره تناوب یکسان است. در حالت کلی چرخه کار طبق رابطه زیر محاسبه می‌شود:

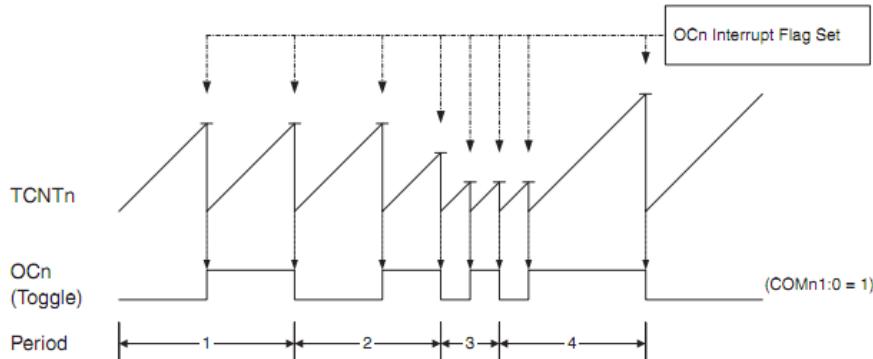
$$\text{چرخه کار} = \frac{\text{زمان یک بودن پالس}}{\text{زمان کل یک دوره تناوب}}$$

#### 4.5.2 حالت CTC<sup>1</sup>

در این حالت محتوای شمارنده تایмер (محتوای ثبات TCNTn) با محتوای ثبات OCRnx (n صفر، یک یا دو و x برابر A یا B) مقایسه می‌شود. در صورتی که این دو برابر باشند مقدار شمارنده صفر قرار داده می‌شود. در حقیقت در این حالت مقدار OCRnx به عنوان حد ماکزیمم شمارش در نظر گرفته می‌شود. لذا پس از برابری TCNTn با OCRnx (OC1A) PD5/(OC1B) PD4 (OC0/AIN1) PB3 (OC2) PD7 (OC0/AIN1) با به تنظیمات سرریز اتفاق می‌افتد و پایه خروجی (OC1A) PD5/(OC1B) PD4 (OC0/AIN1) PB3 (OC2) PD7 (OC0/AIN1) در حین برنامه انجام شده در کدوییزارد مانند شکل 4-7 تغییر می‌کند. مزیت این حالت قابلیت تغییر ثبات OCRnx در حین برنامه است که می‌توان پالس‌هایی با دوره تناوب متغیر ایجاد نمود. فرکانس موج PWM در خروجی OCn برابر است با:

<sup>1</sup> Clear Timer on Compare Match

$$f_{OCnPWM} = \frac{f_{osc}}{2 * N * (1 + OCRnx)} \quad N = 1,8,64,256,1024$$



شکل 7-4: تعیین حد بالای شمارش با استفاده از ثبات OCRnx، در این شکل بعد از هر وقفه Compare Match وضعیت پایه OCn تغییر می‌نماید.

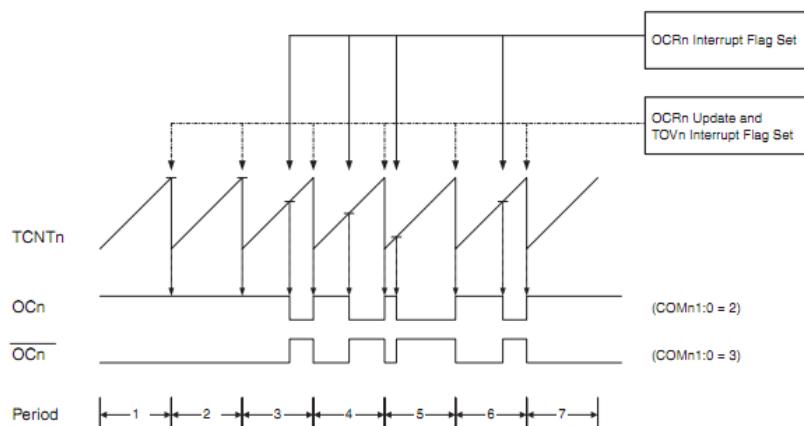
### Fast PWM حالت 4.5.3

این حالت به عنوان ترکیبی از حالت‌های Normal و CTC در نظر گرفته می‌شود. در این حالت از یک سو شمارش مانند حالت Normal از صفر تا حد ماکزیمم (0xffff و یا 0xff) صورت می‌پذیرد و از سوی دیگر مانند حالت CTC مقدار ثبات TCNTn همواره با ثبات OCRn مقایسه می‌شود. پایه خروجی OCn در هنگام برابری این ثبات‌ها و همچنین در زمان سرریز شدن تایمر مانند شکل 4-8 تغییر وضعیت می‌دهد. در این حالت امکان استفاده از دو وقفه تایمر شامل وقفه سرریز و وقفه Compare Match وجود دارد.

بدین ترتیب با تغییر فرکانس تایمر می‌توان دوره تناوب و با تغییر مقدار رजیستر OCRn چرخه‌ی کار را تغییر داد. لازم به ذکر است که مقدار OCRn در روتین وقفه سرریز تایمر به روز می‌گردد. فرکانس و چرخه کار موج PWM تولید شده بر روی پایه خروجی OCn با فرض شمارنده 8 بیتی برابر است با:

$$f_{OCnPWM} = \frac{f_{osc}}{N * 256} \quad N = 1,8,64,256,1024$$

$$Duty\ Cycle(\%) = \frac{OCRnx}{256} * 100$$



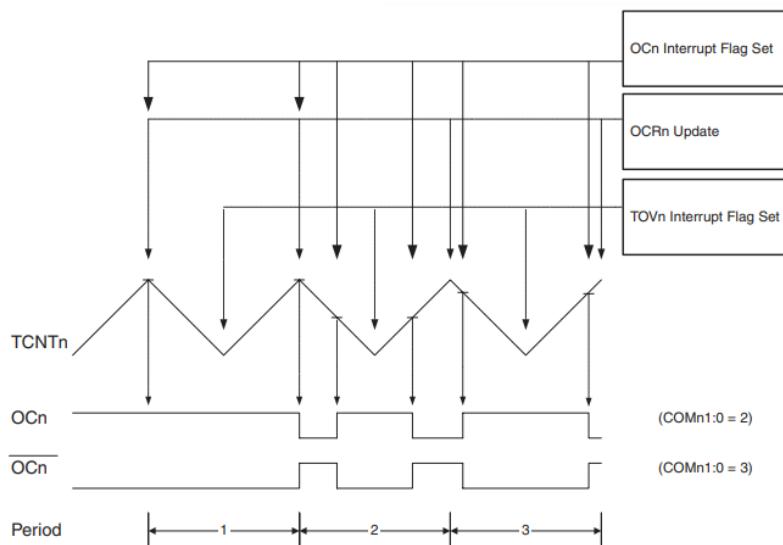
شکل ۸-۴: شکل موج تولیده شده بر روی پایه OCx در حالت Fast PWM

#### Phase Correct PWM 4.5.4

در حالت Phase Correct PWM قابلیت ایجاد موج PWM با تصحیح فاز وجود خواهد داشت. در این حالت تایمیر ابتدا از مقدار اولیه تا مقدار ماکزیمم به صورت افزایشی می‌شمارد. سپس از مقدار ماکزیمم تا مقدار اولیه به صورت کاهشی به شمارش ادامه می‌دهد. در حین شمارش مقدار TCNTn با OCRn مقایسه می‌شود و در صورتی که برابر باشند خروجی OCn تغییر وضعیت می‌دهد.

در این حالت امکان استفاده از دو وقفه تایمیر شامل وقفه سرریز و وقفه Compare Match وجود دارد.

وقفه سرریز تایمیر بر خلاف حالت‌های قبلی با صفر شدن TCNTn رخ می‌دهد در حالی که وقفه CompareMach در صورت برابری مقدار TCNTn و OCRn اتفاق می‌افتد. در هریک از وقفه‌ها می‌توان مقدار OCRn را تغییر داد، ولی فقط هنگامی که TCNTn برابر با حد ماکزیمم می‌باشد مقدار OCRn به روز می‌گردد.



شکل 4-9. نحوه فعال شدن پایه‌ی OCn در حالت Phase Correct PWM

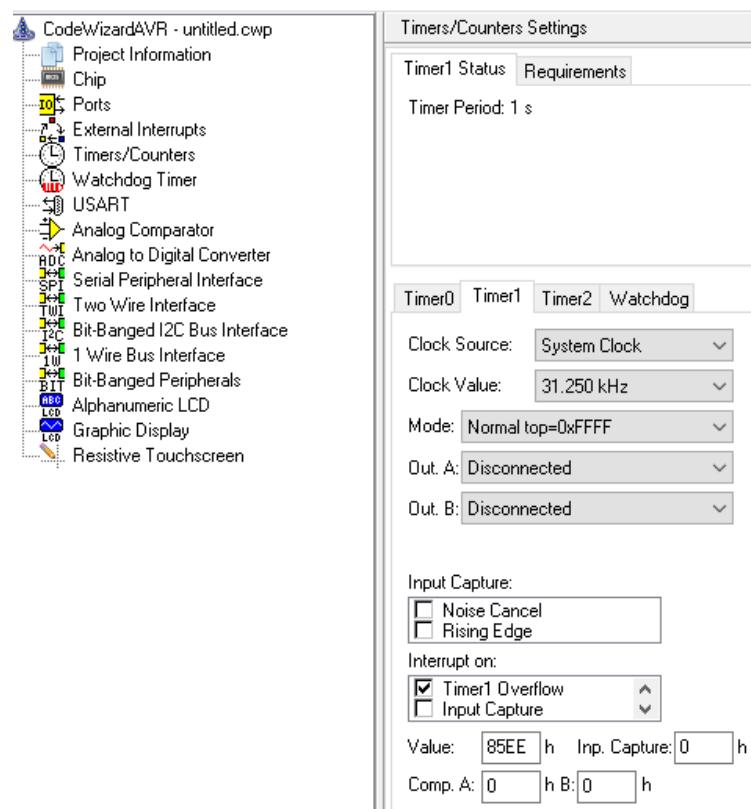
در این حالت، فرکانس موج PWM در پایه خروجی OCn از رابطه‌ی زیر محاسبه می‌شود:

$$F_{\text{PWM}} = \frac{F_{\text{osc}}}{N * 510} \quad N = 1,8,64,256,1024$$

عدد 510 بر اساس شمارش از 0 تا 255 و از 255 تا 0 به دست آمده است.

#### 4.6 مثال کاربردی- طراحی ثانیه شمار

ثانیه شماری را در نظر بگیرید که قابلیت شمارش تا 60 ثانیه را دارد و در هر لحظه زمان را روی LCD نیز نشان می‌دهد. در شکل 4-10 نمایی از تنظیمات تایмер یک برای این منظور نشان داده شده است.



شکل 4-10: نمایی از تنظیمات تایمر یک برای طراحی ثانیه شمار

کد ایجاد شده توسط CodeWizard به همراه تغییرات لازم در برنامه 4-1 نشان داده شده است. برای مختصر شدن حجم برنامه، توضیحات حذف شده است.

```
#include <mega16.h>
#include <alcd.h>
#include <stdio.h>

int i=0;
char str[10];

interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
    TCNT1H=0x85EE >> 8;
    TCNT1L=0x85EE & 0xff;
    i++;
    if( i==60 )
    {
        i=0;
    }

    sprintf(str,"%d ",i);
    lcd_clear();
    lcd_puts(str);
}
```

```

1-4 برنامه void main(void)
{
    //NOTE: define port init for your application

    TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) |
    (0<<WGM10);
    TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (1<<CS12) |
    (0<<CS11) | (0<<CS10);
    TCNT1H=0x85;
    TCNT1L=0xEE;
    ICR1H=0x00;
    ICR1L=0x00;
    OCR1AH=0x00;
    OCR1AL=0x00;
    OCR1BH=0x00;
    OCR1BL=0x00;

    TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) | (0<<OCIE1B) |
    (1<<TOIE1) | (0<<OCIE0) | (0<<TOIE0);

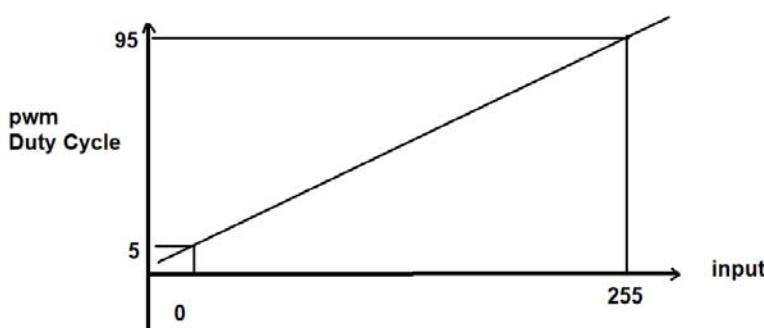
    // Global enable interrupts
    #asm("sei")
    lcd_init(16);
    lcd_clear();

    while(1);
}

```

#### 4.7 محاسبه خطای

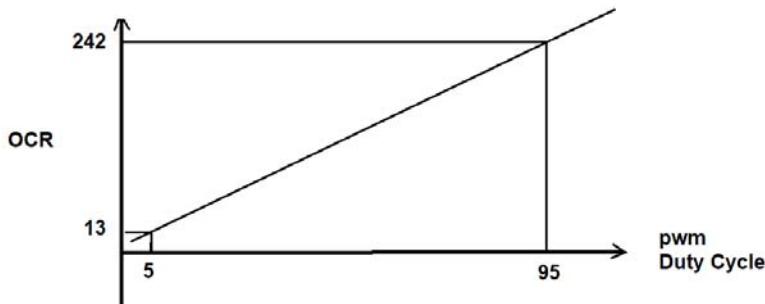
در برخی سیستم‌ها گاهی لازم خواهد شد که رفتار سیستم بنا به ورودی‌های دریافت شده تغییر کند و با شرایط جدید تطبیق یابد. به عنوان مثال فرض کنید طراحی یک پالس PWM با دوره تناوب 8.192 ms و با چرخه‌ی کار متغیر (بر اساس داده دریافتی از ورودی‌های متصل به یکی از درگاه‌ها) مد نظر باشد، در این شرایط بنا بر شکل 11-4 رابطه‌ی ذیل برقرار است.



شکل 4-11: رابطه خطی بین ورودی و Duty\_Cycle

$$DutyCycle_{PWM} = \frac{95 - 5}{255 - 0} * input + 5$$

بعد از تعیین DutyCycle بر اساس ورودی، با فرض حالت کاری Fast PWM و وضعیت پایه خروجی به صورت مقادیر مختلف OCR محاسبه می‌شود.



$$OCR = \frac{242 - 13}{95 - 5} * (DutyCycle_{PWM} - 5) + 13$$

از تلفیق روابط بالا، رابطه‌ی زیر به دست می‌آید.

$$OCR = \left( \frac{229}{255} * input \right) + 13$$

بدین ترتیب مقدار ثبات OCR برای ایجاد موج PWM محاسبه می‌گردد. حال دستورات زیر را برای پیاده سازی در نظر بگیرید:

```

1      OCR_reg=(229/255)*input+13;
2      OCR_reg=((229*input)/255)+13 ;
3      OCR_reg=(((229*input)+13*255)/255);

```

با توجه به این که ریزپردازنده ۳۲/۱۶ Atmega است و نوع داده floating point را پشتیبانی نمی‌کند، مقدار خروجی رابطه‌ی یک همواره برابر با ۱۳ خواهد بود. ولی روابط ۲ و ۳ به درستی اجرا می‌شوند. با استفاده از برنامه ۴-۲ می‌توان حاصل عملیات روابط ۱ تا ۳ را با یکدیگر مقایسه نمود.

```

for(input=0;input<255;input++)
{
    ans1_8bit=(229/255)*input+13;
    ans2_8bit=((229*input)/255)+13;
    ans3_8bit=(((229*input)+13*255)/255);

    sprintf(scr,"%2d,%2d,%2d,%2d\n\r",input,ans1_8bit,ans2_8bit,ans3_8bit);
    puts(scr);
    delay_ms(50);
}

```

برنامه ۴-۲

input	ans1	ans2	ans3	real
0	13	13	13	13
1	13	13	13	13.8
2	13	14	14	14.7
3	13	15	15	15.7
28	13	38	38	38.1
188	13	181	181	181.8
254	13	241	241	241.1

لذا می‌توان درصد خطای دوره تناوب‌های مورد نظر را با استفاده از رابطه زیر محاسبه نمود.

$$\text{Error}(\%) = \frac{\text{Real value (floating point)} - \text{Calculated value(integer)}}{\text{Real value (floating point)}} * 100$$

البته برای محاسبه رقم‌های اعشاری می‌توان از تکنیک‌های دیگری نیز استفاده کرد. در یکی از تکنیک‌ها، می‌توان مقدار متغیر را 100 برابر نمود و بعد از پایان محاسبات، نتیجه نهایی را با بر 100 تقسیم نمود تا قسمت حقیقی محاسبه گردد و با محاسبه باقیمانده تقسیم متغیر بر 100 قسمت اعشاری نیز محاسبه می‌شود. در این حالت بهتر است متغیر از نوع unsigned long int انتخاب شود. با استفاده از این روش می‌توان مانند برنامه 3-4 بسیاری از پارامترهای اعشاری را محاسبه و روی صفحه نمایش مانند شکل 4-12 نشان داد.

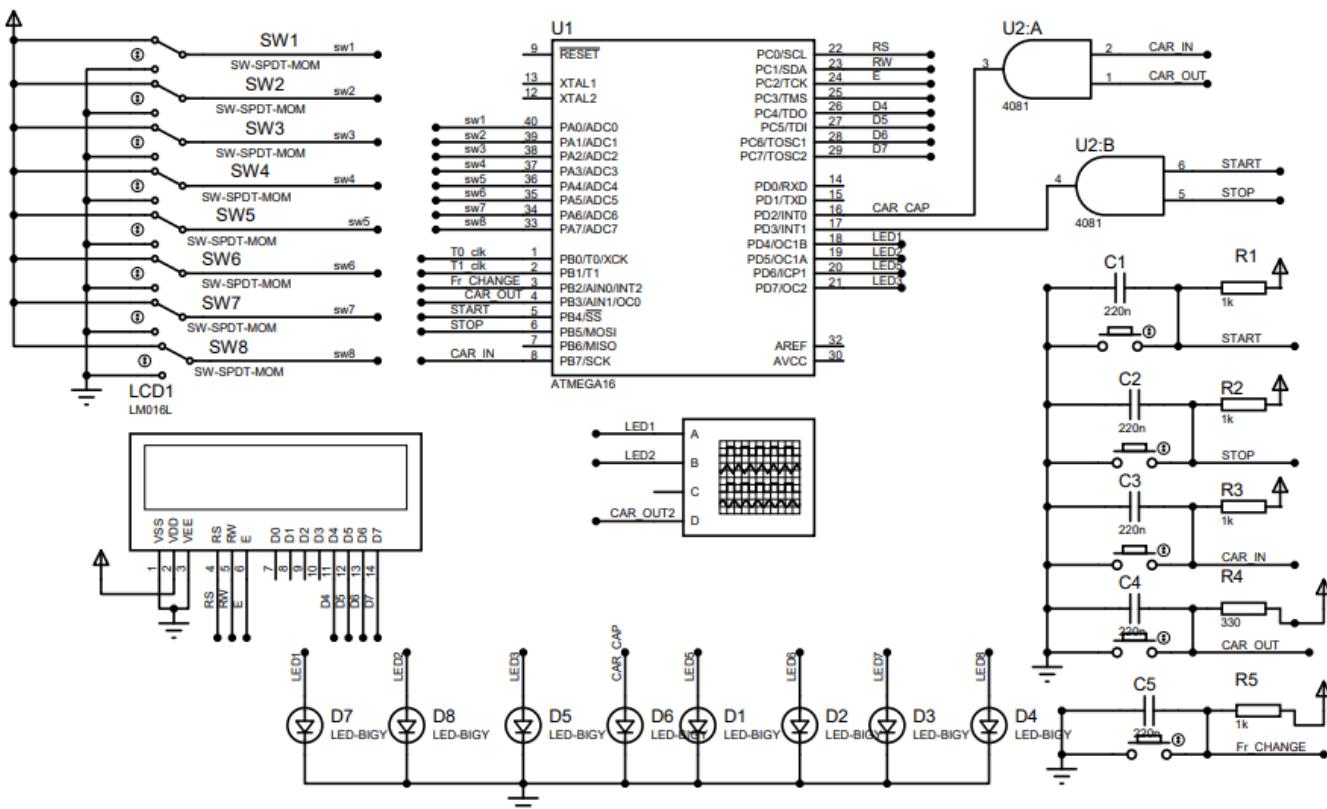
```
3-4 برنامه
data3=10;
data2= ((229*data3)+13*255)/255;
data=(100*((229*data3)+13*255)/255);
sprintf(scr,"float:%d.%d \n\r",data/100,data%100);
lcd_puts(scr);
sprintf(scr,"integer: %d \n\r",data2);
lcd_puts(scr);
```



شکل 4-12: قالب ایجاد اعداد ممیز شناور در ریزپردازنده هشت بیتی

#### 4.8 برنامه‌های اجرایی مبحث تایمرها

با در نظر گرفتن سخت‌افزار نشان داده شده در شکل 4-13، برنامه‌های زیر را در ارتباط با مبحث تایمر و وقه‌های آن‌ها بنویسید.



شکل 4-13: نمای سخت‌افزاری مبحث تایمر

- یک تایمر(کرنومتر) با دقیقیت 0.01 ثانیه طراحی و پیاده‌سازی کنید که با فشردن کلید start شروع به شمارش نماید و زمان آن همواره روی خط اول LCD نمایش داده شود و این تایمر باید با زدن کلید stop متوقف شود و زمان به صورت ثابت روی LCD باقی بماند. اگر بعد از متوقف شدن، مجدداً کلید stop زده شود، زمان نشان داده شده روی LCD صفر گردد. (راهنما: ابتدا یکی از تایمرها را در بازه زمانی مناسب فعال نمایید و

زیربرنامه محاسبه زمان را در وقفه تایمر فراخوانی نمایید. سپس در روتین وقفه خارجی یک، وضعیت پایه‌های stop و start را بررسی کرده و بنا به شرایط، تصمیم لازم اتخاذ گردد.)

2- پارکینگی را در نظر بگیرید که به اندازه 1000 ماشین ظرفیت دارد. در ورودی پارکینگ، سنسوری وجود دارد که به محض ورود ماشین، سیگنال CAR\_IN و در زمان خروج ماشین، سیگنال CAR\_OUT را فعال می‌نماید (در فایل شبیه‌سازی این دو سیگنال را با استفاده از دو کلید مجزا پیاده‌سازی شده است). در هر لحظه ظرفیت فضای خالی پارکینگ روی LCD نمایش داده می‌شود و بعد از پر شدن عبارت FULL نشان داده می‌شود. اگر کلیدهای مربوط به سنسور ورود/خروج به پایه‌های وقفه‌ی خارجی متصل باشند، زیربرنامه‌ای بنویسید که با استفاده از آن ظرفیت خالی پارکینگ روی LCD نمایش داده شود. (راهنمایی در زیربرنامه وقفه خارجی، وضعیت کلیدهای مورد نظر را بررسی نمایید و از آن جا که ساعت در خط اول نمایش داده می‌شود، برای نمایش ظرفیت پارکینگ از خط دوم استفاده نمایید.)

3- زیربرنامه‌ای بنویسید که یک شکل موج مربعی را بر روی پایه‌های خروجی تایمرها (PD4/OC1B و PD5/OC1A) ایجاد نماید. فرکانس ورودی باید از سوییچ‌های متصل به ریزپردازنده دریافت شده و دوره تناوب روی LCD نمایش داده شود. (دوره تناوب را در محدوده 1 میکروثانیه تا 10 میلی ثانیه در نظر بگیرید و با توجه به محدودیت شبیه‌ساز پروتئوس، فقط از فرکانس‌های 8 مگاهرتز و 1 مگاهرتز استفاده نمایید. دوره تناوب شکل موج را می‌توانید در خط دوم و در کنار ظرفیت خالی پارکینگ نمایش دهید.)

## 5 جلسه پنجم

### آشنایی با راه اندازی موتور DC و استپر موتور

#### 5.1 هدف

در این جلسه نحوه راهاندازی برخی از موتورها از جمله موتورهای DC و موتور پله‌ای بررسی می‌شود. هم چنین ادوات جانبی مورد نیاز برای کار با موتورها از جمله رله‌ها و انکوادر نیز معرفی می‌گردد.

#### 5.2 مقدمه

برای ارتباط با محیط پیرامون لازم است که سیستم دیجیتال مبتنی بر ریزپردازنده بتواند با سیستم‌های غیردیجیتال تعامل داشته باشد. موتورها با دریافت فرمان‌های الکتریکی و تبدیل آن‌ها به نیروی مکانیکی و ایجاد حرکت یکی از مهم‌ترین المان‌هایی هستند که در سیستم‌های مختلف توسط ریزپردازنده کنترل می‌گردند. اما با توجه به این که جریان خروجی ریزپردازنده بسیار کم هست و از سوی دیگر برای راهاندازی موتورها نیاز به جریان زیادی داریم، نمی‌توان به صورت مستقیم موتور را به خروجی ریزپردازنده متصل نمود. بنابراین می‌توان از رله‌ها استفاده کرد به این صورت که با تعابیه یک منبع جریان مناسب برای راه اندازی موتور و قطع یا وصل این منبع توسط رله و بر اساس فرمان ریزپردازنده، موتور کنترل می‌گردد. همچنین برای راه اندازی موتورها می‌توان از مدارهای راهانداز از پیش طراحی شده استفاده نمود.

#### 5.3 آشنایی با رله

رله یک سوییچ کنترلی الکتریکی است که با هدایت یک مدار الکتریکی قطع و وصل می‌شود و امکان ایزوله‌ی دو بخش مجزا از یک سیستم با دو منبع ولتاژ متفاوت را فراهم می‌نماید. این سوییچ در کنترل صنعتی، سیستم‌های الکتریکی و الکترونیکی خودرو و بسیاری از وسایل دیگر به طور گسترده استفاده می‌شود. یک مدل از انواع رله مدل الکترومغناطیسی (EMR)<sup>1</sup> است که در شکل 5-1 نشان داده شده است.

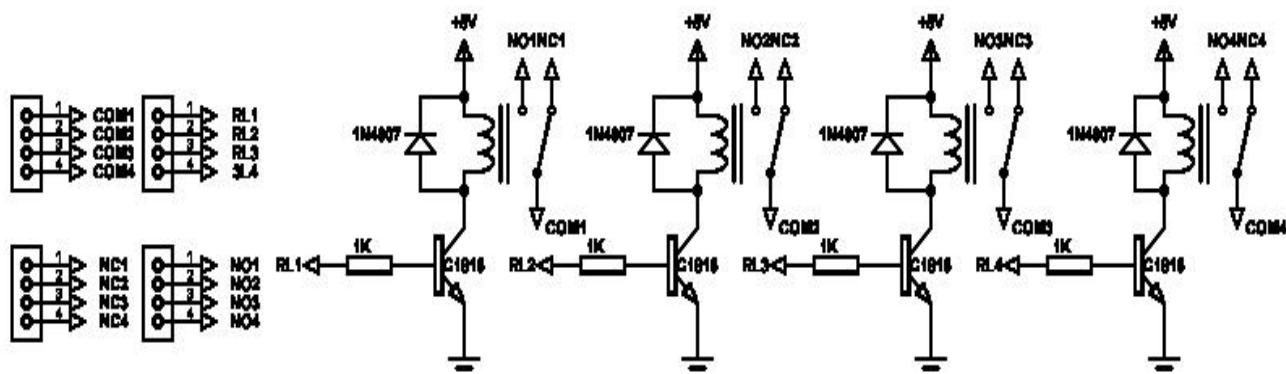
---

<sup>1</sup> ElectroMagnetic Relay



شکل 5-1: رله

در مجموعه بورد آموزشی چهار عدد رله‌ی تک‌کنتاکت به منظور قطع و وصل کردن خطوط مختلف در بلوکی با عنوان Relay مطابق با شماتیک شکل 5-2 قرار داده شده است. هر بلوک رله متشکل از ترانزیستور، رله و دیود می‌باشد. با اعمال ولتاژ 5 ولت به  $RL_x$ ، سیم پیچ درون رله فعال شده و مسیر عبور جریان را به سمت  $NO_x$  تغییر می‌دهد. به عنوان مثال برای کنترل موتور می‌توان پایه‌ی  $COM_x$  را به منبع ولتاژ 5 ولت و یکی از پایه‌های  $NO_x$  یا  $NC_x$  را به پایه  $CW$  موتور وصل کرد. در نهایت با متصل کردن پایه‌ی  $RL_x$  به یکی از پایه‌های ریزپردازنده می‌توان موتور را روشن یا خاموش نمود. نحوه رفتار هر یک از رله‌ها در جدول 5-1 نمایش داده شده است.



شکل 5-2: مدار 4 عدد رله تعبیه شده بر روی بورد آموزشی

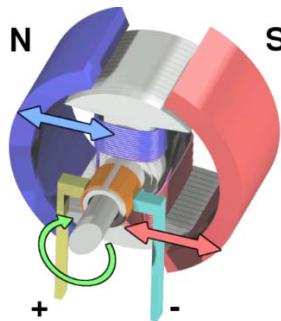
جدول 5-1: وضعیت کنتاکت‌های رله

RL	COM	NO (Normal Open)	NC (Normal Close)
LOW	H	مدار باز	H
	L	مدار باز	L
HIGH	H	H	مدار باز
	L	L	مدار باز

#### 5.4 آشنایی با موتور DC

موتور DC با ساختاری مطابق شکل 5-3 قادر است ولتاژ الکتریکی را به حرکت مکانیکی تبدیل نماید. یک موتور الکتریکی DC از یک جاروبک، روتور دو قطبی و استاتور با آهنربای دائمی تشکیل شده است. حروف N و S نشان دهنده جهت میدان نیروی مغناطیسی حاصل از آهنرباهای دائمی است. علائم + و - نشانگر جریانی است که به کموتاتور اعمال می‌شود و وظیفه تغذیه سیم‌پیچ‌های روتور را بر عهده دارد. فلاش سبز رنگ هم جهت چرخش روتور را نمایش می‌دهد.

با اتصال سرهای + و - به یک منبع ولتاژ DC، موتور در یک جهت و با معکوس کردن اتصال سیم‌ها، موتور در جهت مخالف خواهد چرخید.



شکل 5-3: نمایی از موتور DC

موتورهای DC دو ویژگی بسیار مهم دارند:

1- سرعت موتور به وسیله ولتاژ اعمالی به دو سر آن تعیین می‌شود. حداکثر سرعت یک موتور DC در datasheet مربوط به آن بر حسب rpm (دور بر دقیقه) نشان داده شده است و در محدوده ولتاژ کاری موتور، هر چه ولتاژ اعمال شده به موتور را افزایش دهیم rpm نیز بیشتر می‌شود.

2- گشتاور موتور به وسیله جریانی که می‌کشد تعیین می‌شود و مقدار جریان عبوری به میزان بار بستگی دارد. وجود بار باعث کاهش سرعت موتور می‌شود. با یک ولتاژ ثابت، هنگامی که بار افزایش می‌یابد، جریان مصرفی موتور افزایش خواهد یافت. اگر به موتور بیش از حد بار اعمال کنیم، موتور متوقف شده و ممکن است به علت گرمای تولید شده به دلیل مصرف جریان بالا، به موتور آسیب وارد شود.

لذا برای این که دور موتور DC را با دقت بیشتری کنترل کنیم می‌توان از سیگنال PWM استفاده کرد. در این حالت فرکانس موج مورد نظر با توجه به محدودیت‌های مشخص شده در دیتا شیت تعیین می‌شود و با تغییر عرض پالس می‌توان دور موتور را کنترل کرد.

## 5.5 آشنایی با انکودر

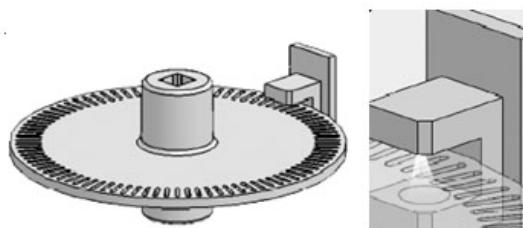
برای اندازه گیری میزان چرخش موتور DC از انکودر استفاده می شود. انکودر حسگری است که به محور چرخ، چرخ دنده یا موتور وصل می شود و می تواند میزان چرخش را اندازه گیری کند. با اندازه گیری میزان چرخش، می توان میزان جابه جایی، سرعت، شتاب یا زاویه چرخش را تعیین نمود.

انکودرهای عموماً از نوع نوری یا لیزری می باشند که در آن ها یک فرستنده و یک گیرنده امواج در دو سمت یک جسم مکانیکی چرخنده (دیسک شیاردار) قرار می گیرند. اگر نور ارسالی توسط فرستنده از شیارهای چرخنده عبور کند توسط گیرنده دریافت می شود و مقدار ولتاژ خروجی یک می شود و زمانی که نور ارسالی به پرهای برخورد کند توسط گیرنده دریافت نمی شود و مقدار ولتاژ خروجی از گیرنده صفر می گردد. به این ترتیب پالس های الکتریکی تولید می شود.

یک عدد موتور DC به همراه یک عدد انکودر لیزری در بلوکی با عنوان DC Motor + Encoder Rotation در مجموعه آموزشی قرار داده شده است. کاربر با فعال کردن پایه CW می تواند موتور را در جهت عقربه های ساعت و با فعال سازی پایه CCW، آن را در جهت خلاف عقربه های ساعت به گردش در آورد.

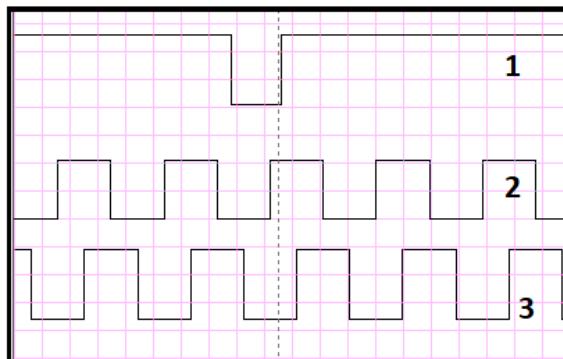
اگرچه با اعمال ولتاژ ۵ ولت به پایه CW موتور شروع به چرخش می کند، اما برای کنترل موتور با استفاده از ریز پردازنده، به دلیل عدم تامین جریان کافی، نیاز به یک سیستم تقویت جریان خواهیم داشت. برای تقویت این جریان از رله های تعییه شده در این بورد آموزشی که در بخش قبلی توضیح داده شد استفاده خواهیم نمود.

به منظور موقعیت سنجی شفت موتور مانند شکل 4-5، یک انکودر لیزری 200 پالسی روی شفت موتور بسته شده است. در واقع روی صفحه لغزان 200 شیار ایجاد شده است. برای کار با این انکودر ابتدا پایه Enable مربوط به Encoder با اعمال یک منطقی فعال می گردد تا فرستنده روشن و گیرنده آماده دریافت پالس های دریافتی شود. در ادامه با چرخش موتور و در اثر عبور نور لیزر از شیارها، پالس هایی متناظر با موقعیت شفت بر روی پین Pulse Out تولید می گردد.



شکل 4-5. نمایی از انکودر متصل به موتور DC (صفحه لغزان قرار داده شده روی شفت به دقت تنظیم گردیده است. از این روز از تغییر مکان صفحه روی شفت اکیداً خودداری نمایید).

در خروجی انکودر دو سری پالس با دوره تناوب مشابه و اختلاف فاز این سیگنال‌ها می‌توان جهت چرخش را مشخص کرد. همچنین بر اساس دوره تناوب هر یک از این پالس‌ها و گام موتور، دور موتور محاسبه می‌شود.



1: شکل موج PWM، 2: خروجی اول انکودر و 3: خروجی دوم انکودر

شکل 5-5. خروجی انکودر متصل به موتور DC

## 5.6 اندازه گیری دور موتور DC

همان‌طور که در بخش قبل اشاره شد، انکودر دو پالس تولید می‌کند که با یکدیگر دارای اختلاف فاز هستند. اگر فرض کنید که سیگنال اول زودتر از سیگنال دوم دریافت شود، به این معنی است که موتور به حالت ساعتگرد حرکت می‌کند و چنان‌چه سیگنال دوم پیش از سیگنال اول دریافت شد، یعنی موتور به صورت پاد ساعتگرد حرکت می‌کند. با اتصال این سیگنال‌ها به وقفه‌های خارجی ریزپردازنده می‌توان جهت چرخش موتور DC را تعیین نمود.

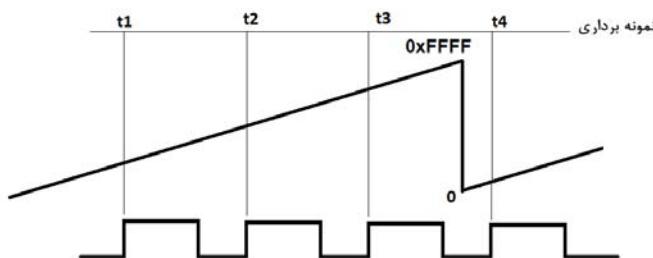
برای تعیین میزان چرخش موتور DC می‌توان از وقفه‌های تایмер یک استفاده کرد. لذا برای محاسبه دوره تناوب پالس‌های دریافتی انکودر، بایستی وقفه INPUT Capture فعال گردیده و حساسیت آن بر روی لبه‌ی بالا رونده یا پایین رونده تنظیم شود تا هنگامی که سطح منطقی رویداد در پایه ورودی (ICP1) تغییر کرد، فرمان ضبط صادر و محتوای شمارنده TCNT1 در ثبات ICR ذخیره شود. در روال زیر برنامه وقفه نیز بلافاصله داده‌ها در یک متغیر ذخیره می‌شوند تا مشکل نوشتن مجدد رخ ندهد. سپس با توجه به اختلاف زمانی بین نمونه‌های ثبت شده می‌توان دوره تناوب را محاسبه کرد.

در هنگام نمونه‌برداری وضعیت سرریز شدن تایمر هم بررسی می‌گردد. اگر فاصله بین دو نمونه‌برداری با سرریز شدن تایمر همراه باشد، داده‌های به دست آمده برای زمان رویداد معتبر نیستند. لذا بایستی فرکانس کار تایmer به گونه‌ای انتخاب شود که در فاصله شمارش از صفر تا  $\text{max}$  بتوان حداقل دو بار عملیات نمونه‌برداری را انجام داد. به عبارت دیگر دوره تناوب پدیده‌های رخ داده باید از دوره تناوب تایمر کمتر باشد.

فرکانس تایمیر می‌تواند با توجه به اطلاع قبلی از محدوده دوره تنابوب رویداد و یا به طور خودکار تنظیم گردد تا بتوان چندین نمونه‌برداری را در یک بازه شمارش از صفر تا مقدار بیشینه تایمیر انجام داد. بر اساس شکل 5-6 می‌توان دوره تنابوب لحظه‌ای را از طریق رابطه زیر به دست آمده امکان محاسبه میانگین دوره تنابوب هم وجود خواهد داشت.

$$T_i = t_{i+1} - t_i \quad i = 1, 2, \dots, n$$

در شکل 5-6 زمان  $t_4$  در لحظه بعد از سرریز تایمیر ثبت شده است و نمی‌توان از اختلاف آن با  $t_3$  دوره تنابوب را اندازه گیری نمود.



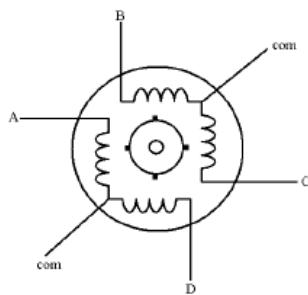
شکل 5-6: نمایی از نمونه‌برداری از رخدادهای متناوب خارجی با استفاده از تایمیر یک

به همین صورت می‌توان عرض پالس را هم اندازه گرفت. برای این منظور بایستی پس از نمونه‌برداری در لبه بالارونده، حساسیت input capture به لبه پایین‌روندۀ تغییر نماید و مجدداً پس از نمونه‌برداری، حساسیت به لبه بالا روندۀ برگردد. این روندۀ ادامه می‌یابد تا تمام لبه‌های پالس دارای برچسب زمانی شده و امکان محاسبه‌ی عرض پالس فراهم گردد.

## 5.7 موتور پله‌ای

موتور پله‌ای موتوری است که به ازای پالس‌های الکتریکی حرکت دورانی ایجاد می‌کند. در واقع یک موتور پله‌ای ترکیبی از یک موتور الکتریکی DC و یک سیم‌پیچ است که حرکت دورانی آن دارای زاویه چرخش معینی است. از آن جایی که این موتورها می‌توانند در یک زاویه خاص قفل شوند، کاربردهای گوناگونی برای آن‌ها وجود دارد. از موتورهای پله‌ای می‌توان برای کاربردهایی که در آن‌ها کنترل دقیق موقعیت یک محور، اهرم و .. مورد نیاز باشد استفاده کرد.

هر موتور پله‌ای دارای یک هسته متحرک مغناطیسی دائمی است که روتور یا شفت نام دارد و به وسیله یک بخش ثابت به نام استاتور احاطه شده است. در شکل زیر ساختار یکی از متداول‌ترین انواع موتور پله‌ای را مشاهده می‌کنید.



شکل 5-7: ساختار موتور پله‌ای با چهار سیم‌پیچ

این نوع موتورها دارای 5 یا 6 سیم می‌باشند که 4 سیم برای استاتور و 2 سیم آن پایه مشترک بوده و باید به VCC وصل شوند (در اکثر موتورها این دو سر وسط از داخل به هم وصل می‌شوند در نتیجه موتور دارای 5 سیم می‌شود). نحوه عملکرد یک موتور پله‌ای تفاوت زیادی با یک موتور DC ندارد و تنها تفاوت در نحوه حرکت موتور است.

برای حرکت دادن موتورهای پله‌ای، پالس‌هایی با فواصل زمانی مختلف به پایه‌های چهارگانه سیم‌پیچ‌ها اعمال می‌گردد. ترتیب اعمال این پالس‌ها از نظم مشخصی پیروی می‌نماید. به عنوان مثال اگر پالس‌هایی مانند جدول 5-2 با فاصله زمانی مشخص به هریک از سیم‌پیچ‌ها اعمال شود، موتور در جهت ساعتگرد یا پادساعتگرد می‌چرخد.

جدول 5-2: ورودی سیم‌پیچ‌های موتور پله‌ای برای چرخش ساعتگرد و پادساعتگرد

A	B	C	D	جهت موتور	A	B	C	D	جهت موتور
1	0	0	0	در جهت عقربه‌های ساعت	0	0	0	1	خلاف جهت عقربه‌های ساعت
0	1	0	0		0	0	1	0	
0	0	1	0		0	1	0	0	
0	0	0	1		1	0	0	0	

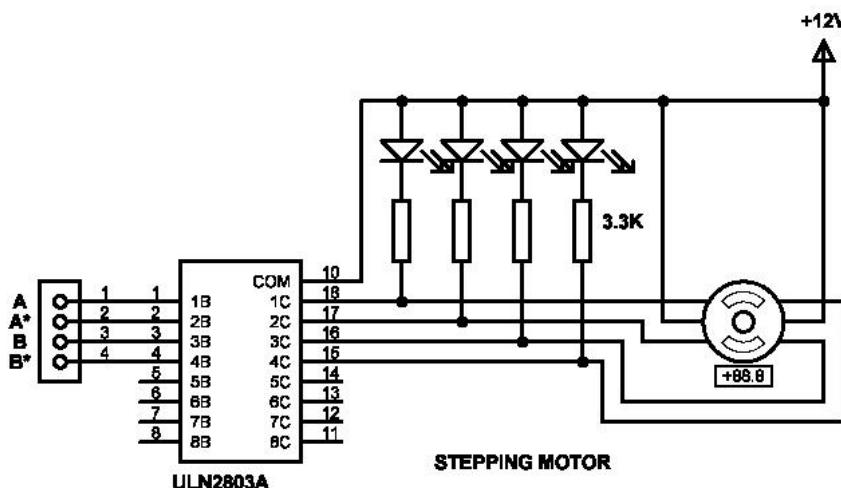
هنگامی که یک پالس به یکی از سیم‌پیچ‌ها اعمال شود، موتور به اندازه یک پله حرکت می‌کند. زاویه پله حداقل زاویه چرخش موتور می‌باشد و گام موتور نیز نامیده می‌شود. این زاویه در موتورهای مختلف بین محدوده 0.72 تا 90 درجه متفاوت می‌باشد که متداول‌ترین میزان زاویه پله، 1.8 درجه است. براساس زاویه پله باید تعداد پالس‌های مشخصی به موتور داده شود تا یک دور کامل بچرخد. تعداد پالس‌های مورد نیاز برای یک چرخش کامل در گام‌های مختلف، در جدول 5-3 مشاهده می‌شود.

جدول 5-3: تعداد پالس‌ها برای یک چرخش کامل در گام‌های متفاوت

زاویه پله	تعداد پله در یک دور
0.72	500
1.8	200
7.5	48
15	24
90	4

برای راهاندازی موتور پله‌ای توسط ریزپردازنده هم به جریان‌دهی مناسب نیاز داریم. پس باید از ترانزیستور یا IC‌های مخصوص استفاده کنیم. یک عدد موتور پله‌ای 6 سیمه با زاویه چرخش پله 1.8 درجه به همراه یک عدد درایور ULN2803A در بلوکی با عنوان Stepping Motor در مجموعه‌ی آموزشی قرار داده شده است. این موتور با ولتاژ 12 ولت تغذیه شده است.

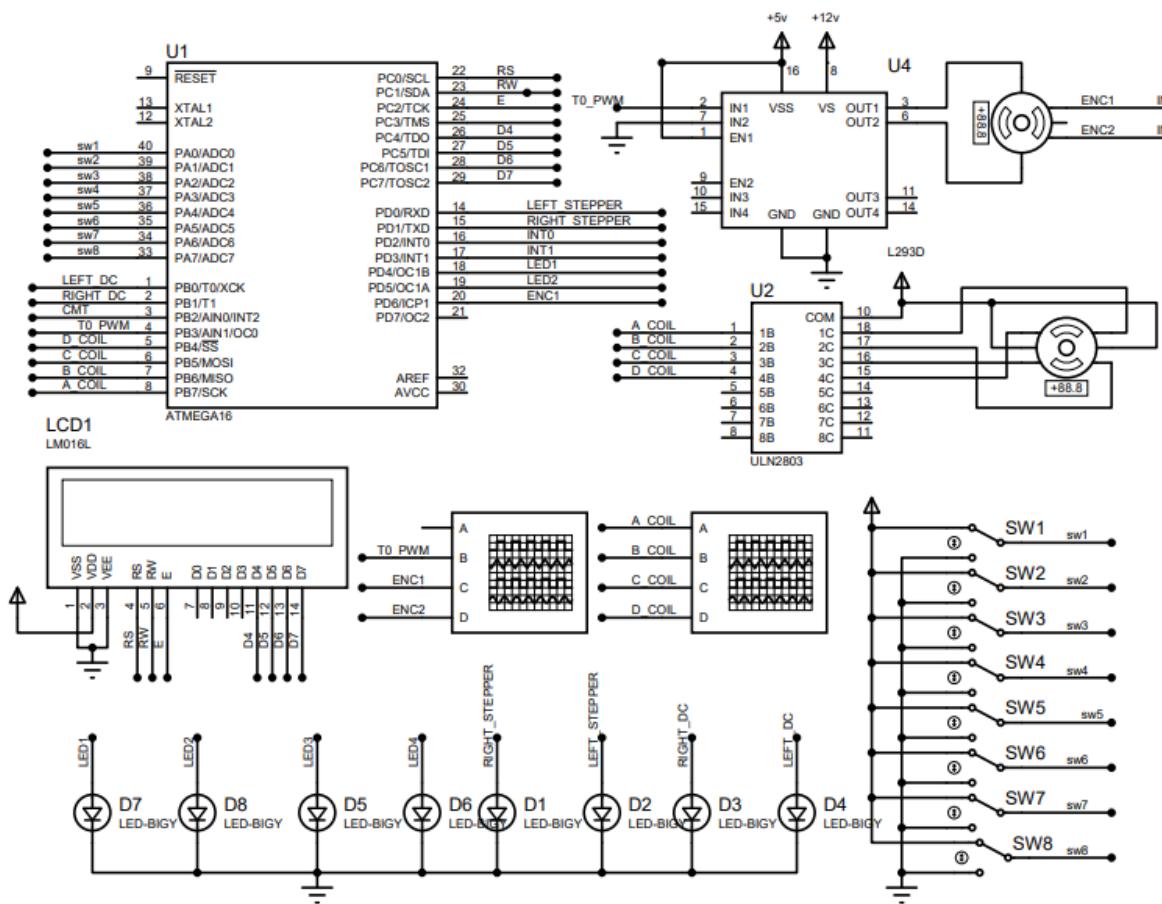
پالس‌های تولید شده توسط ریزپردازنده در ابتدا وارد درایور شده و پس از تقویت جریان به اندازه مطلوب، خروجی به سیم‌پیچ‌های استاتور موتور اعمال می‌شود. 4 عدد LED به منظور نمایش اطلاعات تولید شده توسط ریزپردازنده و همچنین درک بهتر توالی سیگنال‌ها، در این بلوک تعییه شده است. کاربر با معکوس نمودن اطلاعات ارسالی توسط ریزپردازنده می‌تواند جهت چرخش موتور را نیز کنترل نماید. شماتیک مربوط به این بلوک در شکل 5-8 نشان داده شده است.



شکل 5-8: شماتیک بخش موتور پله‌ای

### 5.8 برنامه‌های اجرایی مبحث موتورها

در سختافزار نشان داده شده در شکل 5-9 از برخی المان‌ها از جمله مقاومت‌های سری با LED، به دلیل محدودیت کیفیت تصویر صرفه نظر شده است که البته ایرادی به اجرای برنامه‌ها وارد نمی‌کند. همچنین برای موتور DC از درایور L293D و برای موتور پله‌ای از ULN2803 استفاده شده است.



شکل ۵-۹. نمایی از ساخت افزار بخش موتورها

1. در مورد ساختار و نحوه عملکرد سرو موتورها تحقیق نمایید و گزارش آن را در حد یک صفحه بنویسید.
2. زیربرنامه‌ای بنویسید که با استفاده از تایмер صفر، یک موج PWM با دوره تناب دلخواه و با چرخه‌های کار نشان داده شده در جدول، در خروجی PB3 ایجاد نماید. پس از اجرای برنامه، جدول زیر را کامل نمایید. (در هر بار اجرا دور موتور DC را با استفاده از نمایشگر نشان داده شده در محیط پروتئوس بخوانید. مقدار OCR را هم می‌توان از تنظیمات قابل دسترس در محیط CodeWizard استخراج نمود. سپس مجدداً چرخه کار را تنظیم نموده و روال قبل را تکرار نمایید).

PWM duty cycle%	10	30	50	70	90
Speed(rpm)					
Compare register(OCR0)					

3. در بند 2، رابطه خطی بین PWM\_duty\_cycle و OCR0 را به دست آورید.

4. با استفاده از کدهای توسعه داده شده در بندهای 2 و 3، زیربرنامه‌ای بنویسید که با استفاده از تایمیر صفر، یک موج PWM با دوره تناوب دلخواه را در خروجی PB3 ایجاد نماید. چرخه کار این موج از طریق سوئیچ‌های متصل به درگاه A به عنوان آرگومان ورودی و در محدوده 0 تا 100 درصد، دریافت شود. (صفر ورودی را به عنوان 0 و 255 ورودی را به عنوان 100 درصد در نظر بگیرید).
5. زیربرنامه‌ای بنویسید که با استفاده از تایمیر دو، موتور پله‌ای را با سرعت دلخواه به حرکت درآورد و سرعت آن نیز روی LCD نمایش داده شود (سرعت موتور پله‌ای را با توجه به تنظیمات تایمیر و گام‌های موتور محاسبه نموده و روی LCD نمایش دهید). در این حرکت موتور باید به ترتیب به مدت چند ثانیه راست‌گرد و سپس چند ثانیه چپ‌گرد چرخش نماید. در زمان تغییر جهت چرخش نیز به موتور پله‌ای چند ثانیه استراحت دهید.
- نکته: چنان‌چه سرعت از حد اکثر سرعت مجاز موتور پله‌ای که در برگه‌های راهنمای و بر اساس پارامترهای مکانیکی مشخص می‌شود، بیشتر باشد صدای زنگ مانندی به گوش می‌رسد که بیانگر سرعت بیش از حد توان موتور است. در این حالت موتور به دلیل اینرسی، اصطکاک و ... نمی‌تواند تغییر ایجاد شده روی سیم‌پیچ‌ها را دنبال و در نتیجه حرکت نماید.
6. یکی از خروجی‌های انکودر موتور DC به پایه PD6 متصل شده است. تایمیر یک را در حالت input capture اندازی نمایید و دور موتور DC را محاسبه نمایید. سپس دور موتور را با دقیقه ۵ دور در دقیقه، روی LCD نمایش دهید.
7. بندهای 4، 5 و 6 را در قالب یک پروژه با فایل‌های جانبی در آورید.

## 6 جلسه ششم

### آشنایی با مبدل‌های آنالوگ به دیجیتال و دیجیتال به آنالوگ

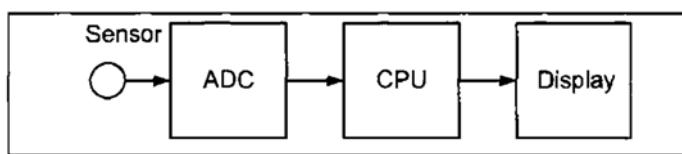
#### 6.1 هدف

در این جلسه، نحوه‌ی تبدیل سیگنال آنالوگ به دیجیتال، استفاده از مبدل ADC داخلی ریزپردازنده، خواندن اطلاعات از حسگرهای آنالوگ و کار با مبدل دیجیتال به آنالوگ بررسی می‌شود.

#### 6.2 مقدمه

در بسیاری از سیستم‌ها، ریزپردازنده لازم است با محیط پیرامون ارتباط برقرار کرده به این صورت که داده‌هایی را از این محیط دریافت نموده و پس از پردازش لازم خروجی را به محیط برگرداند. نکته نخست در این رابطه این است که محیط پیرامون صرفاً شامل سامانه‌های الکتریکی نیست و ساختارهای مکانیکی، نوری و ... نیز در آن وجود دارند. از سوی دیگر در این ارتباط سیگنال‌ها و کمیت‌های محیط پیرامون از جنس آنالوگ بوده، در حالی که در ساختار ریزپردازنده ما با کمیت‌های دیجیتال سر و کار داریم. لذا نیاز به واحدهای سختافزاری برای اندازه گیری کمیت‌های فیزیکی و تبدیل آن‌ها به سیگنال‌های الکتریکی دیجیتال وجود دارد.

به عنوان مثال برای پایش دما یا شدت نور یا هر کمیت دیگر و نمایش آن بر روی LCD کاراکتری، باید مطابق شکل 6-1 ابتدا کمیت فیزیکی مورد نظر با استفاده از یک حسگر مناسب به یک سیگنال الکتریکی (ولتاژ یا جریان) آنالوگ تبدیل شود. سپس این سیگنال توسط مبدل آنالوگ به دیجیتال به داده‌های دیجیتال تبدیل شده که پردازنده می‌تواند آن‌ها را مورد پردازش قرار داده و مقادیرشان را روی نمایشگر نمایش دهد.



شکل 6-1: اتصال یک حسگر به ریزپردازنده با استفاده از مبدل ADC

معکوس روال فوق برای انتقال یک داده دیجیتال از درون ریزپردازنده به محیط پیرامون با استفاده از مبدل دیجیتال به آنالوگ صورت می‌پذیرد. در ادامه اجزای مختلف مورد نیاز در ارتباط ریزپردازنده با محیط پیرامون معرفی خواهند شد.

### 6.3 حسگرها

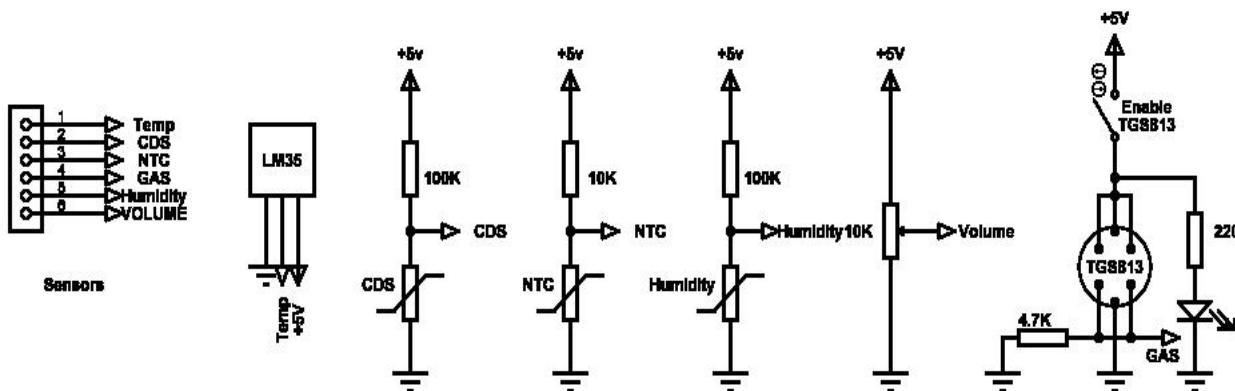
همان گونه که در بخش قبل بیان شد، برای تبدیل کمیت‌های فیزیکی مختلف به کمیت‌های الکتریکی از حسگرها مختلف استفاده می‌شود. بر روی بورد آموزشی حسگرهای گوناگونی به شرح زیر در بلوکی با عنوان Sensors قرار داده شده است.

- دو عدد حسگر دما (NTC و LM35)
- یک عدد حسگر نور (CDS)
- یک عدد حسگر رطوبت (HIS06)
- یک عدد حسگر گاز شهری (TGS813)
- یک ولوم 10 کیلو اهم جهت شبیه‌سازی حسگر مولفه‌های محیطی دلخواه

حسگرهای NTC، CDS و HIS06 با تغییر پارامترهای محیطی (به ترتیب نور، دما و رطوبت) تغییر مقاومت می‌دهند. از این رو به منظور ارتباط با ریزپردازنده طبق مدارهای Error! Reference source not found. تغییرات مقاومت در آن‌ها به تغییرات ولتاژ تبدیل شده است.

حسگر TGS813 هم یک حسگر آشکارساز گاز شهری با خروجی ولتاژ می‌باشد که مستقیماً به ریزپردازنده متصل می‌شود. به دلیل افزایش حرارت بدن حسگر و همچنین جریان کشی بالای این حسگر از منبع تغذیه مطابق شماتیک شکل 6-2 یک کلید کشویی برای قطع و وصل نمودن این حسگر در مسیر تغذیه حسگر تعبیه شده و LED موجود در این بلوک نشانگر وصل یا قطع بودن این حسگر در مدار است.

همچنین یک عدد ولوم 10 کیلو اهم به منظور تولید ولتاژ از سطح صفر ولت تا 5 ولت برای شبیه‌سازی حسگرهای پارامترهای محیطی در این بلوک قرار داده شده است.



شکل 6-2- شماتیک مربوط به حسگرهای موجود بر روی بورد آموزشی

در مورد حسگر LM35 در بخش‌های بعدی به تفصیل صحبت خواهد شد.

## ADC مبدل 6.4

این مبدل وظیفه تبدیل سیگنال آنالوگ ورودی به یک سیگنال دیجیتال را بر عهده دارد. در ادامه با ویژگی‌های اصلی و ساختار این مبدل آشنا خواهیم شد.

یکی از اصلی‌ترین ویژگی‌های مبدل A/D دقت یا وضوح می‌باشد که بر حسب تعداد بیت بیان می‌شود. در واقع این ویژگی نشان‌دهنده تعداد سطوح قابل تفکیک در بازه تغییرات هر نمونه از سیگنال آنالوگ ورودی است. به عنوان مثال دقت مبدل‌های ADC در ریزپردازنده Atmega16/32، 8 یا 10 بیت می‌باشد که نشان می‌دهد هر نمونه سیگنال ورودی با 256 یا 1024 سطح مختلف قابل تبدیل خواهد بود.

بر این اساس فاصله بین مقدار معادل دو مقدار دیجیتال متوالی را step size می‌نامند. هرچه تعداد بیت‌های مبدل دیجیتال بیشتر باشد، step size کمتر است.

$$\text{step size} = \frac{V_{ref}}{2^n}$$

یک ولتاژ مرجع است که A/D را قادر می‌سازد تا سیگنال‌های آنالوگ در محدوده‌ی بین 0 تا  $V_{ref}$  را مانند شکل 6-3 اندازه بگیرد.  $V_{ref}$  می‌تواند یکی از مقادیر AVCC (5 ولت)، ولتاژ مرجع داخلی (2.56 ولت) و یا ولتاژ تعیین شده توسط پایه‌ی خارجی AREF را اتخاذ نماید.

لازم به ذکر است که برای کاهش اثرات نویز در ریزپردازنده از خط تغذیه (AVCC) و زمین (AGND) جداگانه‌ای برای بخش مرتبط با سیگنال‌های آنالوگی استفاده می‌شود که AVCC نباید بیش از  $\pm 0.3V$  نسبت به VCC اختلاف داشته باشد.

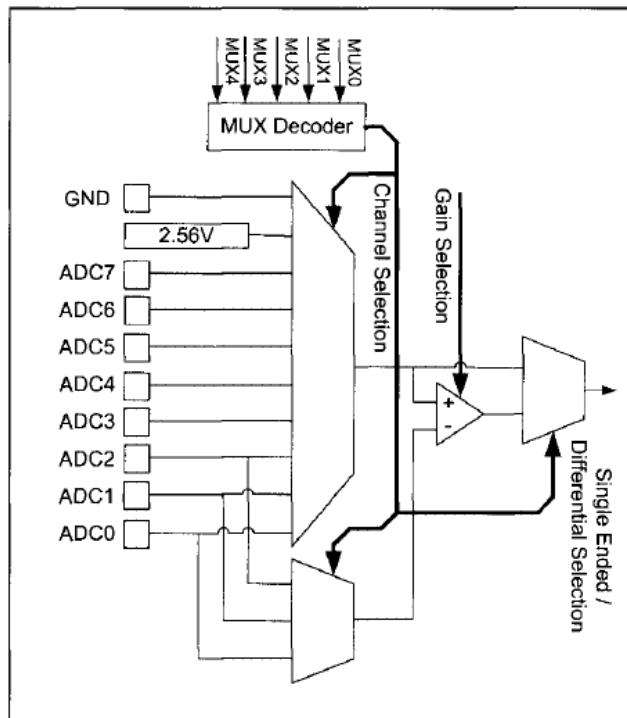
$V_{ref}$ (V)	$V_{in}$ Range (V)	Step Size (mV)
5.00	0 to 5	$5/256 = 19.53$
4.0	0 to 4	$4/256 = 15.62$
3.0	0 to 3	$3/256 = 11.71$
2.56	0 to 2.56	$2.56/256 = 10$
2.0	0 to 2	$2/256 = 7.81$
1.28	0 to 1.28	$1.28/256 = 5$
1	0 to 1	$1/256 = 3.90$

شکل 6-3 ارتباط میان ولتاژ مرجع، ولتاژ ورودی و step

دقت شود که حداقل ولتاژ قابل اندازه‌گیری برابر با  $V_{ref}$  (=VCC) است و در صورت اعمال ولتاژ بیشتر، مبدل آنالوگ به دیجیتال آسیب می‌بیند. کمترین ولتاژ اعمالی نیز برابر با GND است. بنابراین ADC به ازای ولتاژ 5 ولت در

حالت 10 بیتی عدد 1023 (و در حالت 8 بیتی عدد 255) و به ازای صفر ولت عدد صفر را به عنوان خروجی محاسبه و در ثبات مربوطه قرار می‌دهد.

در 4-6 Atmega16/32 مطابق شکل می‌توان 8 سیگنال آنالوگ ورودی را به دیجیتال تبدیل کرد. در هر لحظه یک یا دو کanal ورودی از طریق مالتی پلکسor انتخاب می‌شوند و پس از اعمال ضریب بهره مناسب، نمونه برداری انجام شده و خروجی به صورت داده‌های دیجیتال 8 بیتی و یا 10 بیتی در ثبات مربوطه قرار می‌گیرد.



شکل 6-4: کانال‌های ورودی ADC (تعییه شده بر روی درگاه A)

## 6.5 ثبات‌های مبدل آنالوگ به دیجیتال

برای کار با ADC یک سری ثبات در دسترس است که در ادامه به معرفی آن‌ها می‌پردازیم.

### 6.5.1 ثبات کنترلی ADMUX

تنظیمات اولیه مبدل آنالوگ به دیجیتال در ثبات کنترلی ADMUX انجام می‌شود. این تنظیمات شامل انتخاب ولتاژ مرجع، 8 یا 10 بیتی بودن مبدل، انتخاب کanal ورودی و تنظیم حالت‌های عملکردی می‌باشد.

7	6	5	4	3	2	1	0	ADMUX
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

شکل 6-5: ساختار ثبات کنترلی

از این دو بیت برای انتخاب ولتاژ مرجع ADC استفاده می‌کنیم که دارای چهار حالت زیر می‌باشد:

جدول 6-1: تعیین ولتاژهای مرجع مبدل آنالوگ به دیجیتال

REFS1	REFS0	Vref
0	0	AREF
0	1	AVCC
1	0	-
1	1	2.56

تامین ولتاژ مرجع دقیق در فرایند تبدیل آنالوگ به دیجیتال نقش بسیار مهمی دارد. دقیق‌ترین ولتاژ مرجع همان ولتاژ 2.56 داخلی می‌باشد. البته می‌توان از تثبیت کننده‌های ولتاژ خارجی برای تولید ولتاژ مرجع دلخواه نیز استفاده نمود که در این حالت باید آن را به پایه‌ی AREF متصل نمود.

ADLAR: از این بیت برای تعیین نحوه پر شدن ثبات ADC به صورت از چپ به راست و یا بالعکس استفاده می‌شود. اگر مقدار این بیت صفر باشد ثبات ADC از سمت چپ پرشده و در غیر این صورت از راست پر می‌شود.

MUX0-4: مقدار این بیت‌ها، مانند شکل 6-6 ترکیب ورودی‌های آنالوگ را برای مبدل تعیین می‌نماید. همچنین در ترکیب‌های تفاضلی مقدار ضریب بهره را نیز مشخص می‌کند. به صورت کلی خروجی از رابطه زیر محاسبه خواهد شد:

$$D_{out} = A * \left( \frac{V_i^+ - V_i^-}{V_{ref}} \right) * 2^n$$

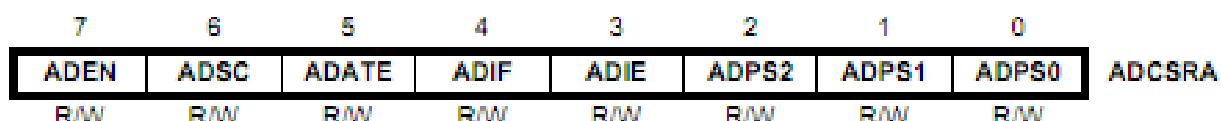
دقت شود که ترکیب تفاضلی فقط در پکیج‌های TQFP و MLF وجود دارد و برای استفاده از آن باید پایه Positive Differential Input را به ولتاژ ورودی آنالوگ و پایه Negative Differential Input را به زمین وصل کنید.

شکل 6-6: تنظیمات قابل اعمال برای ADC با تغییر پارامترهای MUX4-0 در ثبات ADMUX

MUX4_0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22V ( $V_{AO}$ )	N/A		
11111	0V (GND)			

### ADCSRA ثبات 6.5.2

این ثبات نحوه فعال شدن مبدل را تنظیم می نماید و همچنین وضعیت پایان یافتن عملیات تبدیل را درون خود نگه می دارد.



شکل 6-7: ساختار ثبات ADCSRA

: با یک کردن این بیت مبدل ADC فعال می شود.

: با نوشتن یک در این بیت، تبدیل شروع می شود.

ADATE: با یک کردن این بیت، مبدل ADC می‌تواند به صورت خودکار با لبه بالا رونده منبع تحریک کننده که توسط بیت‌های ADTS از ثبات SFIOR انتخاب می‌شود شروع به کار کند.

ADIF: این بیت بعد از اتمام تبدیل و به روز شدن مقدار درون ثبات داده ADC، برابر با یک می‌شود. در حقیقت یک شدن این بیت نشانه‌ی معتبر بودن داده‌های ثبات ADC برای خوانده شدن است.

ADIE: با یک کردن این بیت، پس از اتمام تبدیل وقفه‌ای صادر می‌شود و در زیر روال وقفه، داده ثبات ADC خوانده می‌شود.

ADPS0-3: از این بیت‌ها مطابق جدول 6-2 برای تعیین ضریب تقسیم پالس ساعت ریزپردازنده برای تولید پالس ساعت بخش مبدل آنالوگ به دیجیتال استفاده می‌شود.

جدول 6-2: جدول تنظیمات تقسیم پالس ساعت

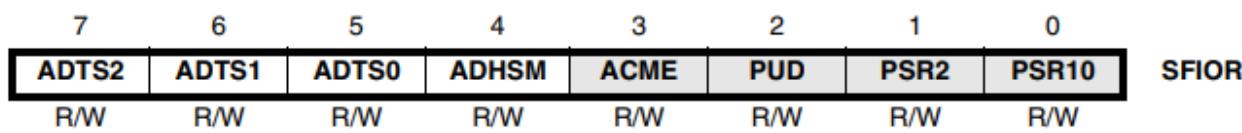
ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

### 6.5.3 ثبات داده (ADCH, ADCL) ADCW

این ثبات شانزده بیتی، حاوی داده‌ی خروجی مبدل است و بنا به مقدار ADLAR، از چپ یا راست پر می‌گردد. در حالت عملکرد 8 بیتی، صرفاً داده‌ی دیجیتال صرفاً ثبات ADCH برگردانده می‌شود. اما در حالت ده بیتی، داده‌ی دیجیتال در رجیستر ADCW برگردانده می‌شود.

### 6.5.4 ثبات <sup>1</sup>SFIOR

با استفاده از این ثبات که ساختار آن در شکل 6-8 نشان داده شده است، منبع تحریک مبدل و نیز حالت عملکرد سرعت بالا تنظیم می‌شود.



شکل 6-8: ساختار ثبات

<sup>1</sup>Special Function IO Register

3-6: از طریق بیت‌های ADTS0-2 می‌توان منبع تحریک مبدل برای شروع تبدیل را مانند جدول 6-3 تنظیم نمود.

جدول 6-3: تعیین منابع تحریک ADC

منبع تحریک ADC	ADTS0	ADTS1	ADTS2
مددعملکردآزاد	0	0	0
مقایسه کننده آنالوگ	1	0	0
وقفه خارجی صفر	0	1	0
وقفه‌ی Compare Match (Compare Match) تایمر صفر	1	1	0
سرریز تایمر صفر	0	0	1
Compare Match B (وقفه‌ی Compare Match B)	1	0	1
سرریز تایمر یک	0	1	1
ضبط رخداد تایمر یک	1	1	1

بیت ADHSM: با فعال شدن این بیت، فرایند نمونه‌برداری در مبدل با سرعت بیشتر و البته با مصرف انرژی بیشتر انجام می‌شود.

### 6.5.5 معرفی وقفه ADC

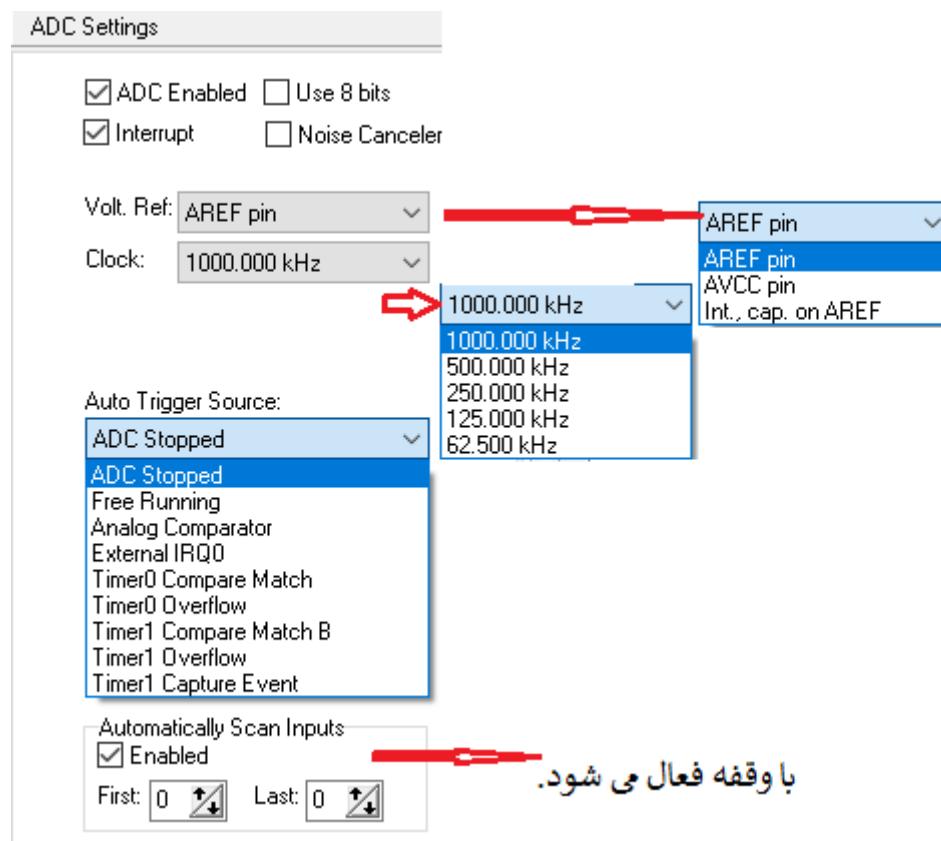
برای جلوگیری از اتلاف زمان ریزپردازنده، به جای بررسی مکرر بیت ADSC در ثبات ADCSRA بهتر است از وقفه استفاده شود (برای این منظور لازم است بیت ADIE از ثبات ADCSRA برابر با یک باشد). در این صورت به محض کامل شدن تبدیل، پرچم ADIF یک می‌شود و CPU به محل اجرای وقفه پرش کرده که در آن جا نتیجه ADC قابل استفاده خواهد بود.

### 6.6 مراحل برنامه‌نویسی ADC

مراحل برنامه‌نویسی مبدل آنالوگ به دیجیتال در شکل 6-9 نشان داده شده است. بدین منظور لازم است پایه ورودی سیگنال آنالوگ تعریف شود و از منوی CodeWizard سایر پارامترها نیز تنظیم شود که به ترتیب شامل موارد ذیل می‌باشد:

1. فعال کردن مازول ADC

2. انتخاب 8 یا 10 بیتی بودن تبدیل: در حالت کلی مبدل Atmega16/32 به صورت ده بیتی نمونهبرداری را انجام می‌دهد، ولی با انتخاب هشت بیتی تنها 8 بیت با ارزش‌تر در نظر گرفته می‌شوند.
3. حذف نویز: عملکرد عادی بخش‌های مختلف ریزپردازنده که با فرکانس بالا صورت می‌پذیرد می‌توانند به عنوان یک عامل ایجاد نویز، بر روی عملکرد واحد ADC تاثیر نامطلوب بگذارند. لذا در برخی موارد، سعی می‌شود این عملکرد به صورت مقطعی متوقف شده تا بتوان با کیفیت بهتری نمونهبرداری از داده‌ی آنالوگ را انجام داد. برای این کار باید گزینه Noise Canceler انتخاب شود. البته قبل از انجام این کار بایستی حتماً وقف واحد ADC فعال شده باشد. با فعال شده این گزینه، نمونهبرداری فقط در زمانی که ریزپردازنده حالت بیکار داشته باشد انجام می‌شود. لذا قبل از هر بارخواندن مقدار آنالوگ ورودی بایستی یک پیام Sleep به ریزپردازنده ارسال گردد.
4. تعیین کردن سرعت تبدیل: انتخاب سرعت تبدیل به فرکانس سیگنال آنالوگ ورودی بستگی دارد. اگر فرکانس سیگنال آنالوگ بالا باشد بهتر است از نرخ‌های بالاتر برای تبدیل استفاده نمود و اگر تغییرات ورودی به کندی انجام می‌شود می‌توان از نرخ‌های پایین‌تر تبدیل استفاده نمود تا هم مصرف انرژی ریزپردازنده کمتر شود و هم نویز کمتری از طریق ADC به کل مدار وارد شود. در حقیقت در این حالت، برعکس بند قبلی، عامل نویز خود واحد ADC است که می‌تواند سایر بخش‌های حساس و با فرکانس کاری بالا بر روی برد را تحت تاثیر قرار دهد.
5. انتخاب ولتاژ مرجع: همان گونه که پیش‌تر بیان شد، در مبدل‌های ADC برای بهبود دقت اندازه‌گیری از ولتاژ مرجع دقیق استفاده می‌شود که می‌توان در صورت نیاز توسط منبع داخلی و یا با استفاده از تراشه‌های خارجی مناسب این ولتاژ را تامین نمود.
6. انتخاب منبع تحریک مبدل: در حالت عادی می‌توان از حالت free running استفاده کرد. همچنین امکان فعال نمودن وقفه در بازه زمانی مشخص هم مانند استفاده از وقفه‌های تایмер وجود دارد.
7. اگر وقفه فعال شده باشد می‌توان به طور خودکار تعدادی از کانال‌های ADC را پردازش نمود.



شکل 6-9: برنامه نویسی مبدل آنالوگ به دیجیتال با

### 6.6.1 برنامه خواندن داده‌های مبدل دیجیتال

برای خواندن داده‌های مبدل ADC در حالت بدون وقفه از زیر برنامه‌ی `read_adc` مانند برنامه 6-1 می‌توان استفاده کرد. هم‌چنین اگر وقفه فعال شده باشد، با استفاده از روتین وقفه‌ی برنامه 6-2، مقدار هر یک از ورودی‌ها خوانده شده و در یک متغیر ذخیره می‌گردد. همان‌گونه که پیش‌تر توضیح داده شد، در برنامه 6-1 و برنامه 6-2 مبدل به صورت ده بیتی فعال بوده و بنابراین کل ثبات ADCW خوانده می‌شود. در صورتی که در حالت هشت بیتی کافی است فقط ADCH خوانده شود.

```

unsigned int read_adc(unsigned char adc_input)
{
    برنامه 6-1 ADMUX=adc_input | ADC_VREF_TYPE;
    //Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    //Start the AD conversion
    ADCSRA|=(1<<ADSC);
    //Wait for the AD conversion to complete
    while ((ADCSRA & (1<<ADIF)) == 0);
    ADCSRA|=(1<<ADIF);
    return ADCW;
}

```

}

```

interrupt [ADC_INT] void adc_isr(void)
{
2-6 برنامه static unsigned char input_index=0;
// Read the AD conversion result
adc_data[input_index]=ADCW;
// Select next ADC input
if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
    input_index=0;
ADMUX=(FIRST_ADC_INPUT | ADC_VREF_TYPE)+input_index;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=(1<<ADSC);
}

```

## 6.7 اندازه‌گیری دما

برای اندازه‌گیری دما می‌توان از حسگر LM35 که در شکل 6-10 نشان داده شده استفاده نمود. این حسگر دمای بین 55- تا 150 درجه‌ی سانتیگراد را اندازه‌می‌گیرد و به ازای هر  $1 \pm 10$  mV سانتی‌گراد، ولتاژ خروجی را تغییر می‌دهد. یعنی به ازای دمای 1 درجه، ولتاژ خروجی حسگر 10mV و به ازای دمای 100 درجه، خروجی حسگر 1V خواهد بود.



شکل 6-10: حسگر اندازه‌گیری دما

مطابق شکل فوق، این حسگر دارای 3 پایه می‌باشد. در صورتی که حسگر را روپروری خود بگیرید به گونه‌ای که بتوانید نوشه‌هایش را ببینید، پایه سمت چپ تغذیه حسگر (5 ولت)، پایه وسط ولتاژ خروجی (که به ریزپردازنده متصل می‌شود) و پایه سمت راست پایه زمین خواهد بود. خروجی آنالوگ این حسگر را می‌توان توسط مبدل‌های

آنالوگ به دیجیتال موجود در تراشه Atmega16/32 به دیجیتال تبدیل کرده و به عنوان مثال ان را روی LCD نمایش داد.

مثال: با استفاده از حسگر LM35، دما را در بازه‌ی 0 تا 50 درجه سانتیگراد اندازه‌گیری و بر روی LCD نمایش دهید.

برای اندازه‌گیری دما در بازه‌ی 0 تا 50 درجه روابط ذیل برقرار است:

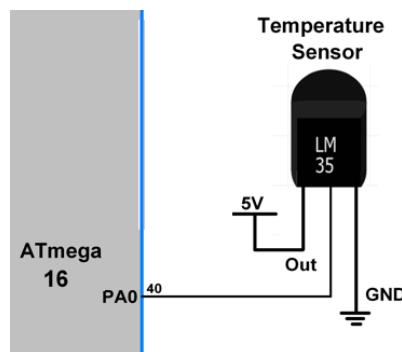
$$D_{OUT} = \frac{V_{LM35}}{V_{REF}} \times 2^{10}$$

$$V_{LM35} = 10mV \times T$$

اگر ولتاژ مرجع برابر با 5 ولت در نظر گرفته شود و مبدل آنالوگ به دیجیتال در حالت 10 بیتی کار کند، دما برابر خواهد بود با:

$$T = \frac{D_{OUT} \times 500}{2^{10}} \cong \frac{D_{OUT}}{2}$$

در ادامه مانند شکل 11-6 خروجی LM35 را به پایه ADC0 متصل می‌کنیم. LCD نیز به درگاه B وصل است.



شکل 11-6 نحوه اتصال LM35 به ریزپردازنده

سپس در قسمت CodeWizard تنظیمات مربوط به LCD و ADC انجام و قطعه کد زیر به برنامه اضافه می‌گردد.

```
#include <mega16.h>

#include <delay.h>
#include <stdio.h>

// Alphanumeric LCD functions
#include <alcd.h>

// Declare your global variables here
```

```

// Voltage Reference: AVCC pin
#define ADC_VREF_TYPE ((0<<REFS1) | (1<<REFS0) | (0<<ADLAR))

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | ADC_VREF_TYPE;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=(1<<ADSC);
// Wait for the AD conversion to complete
while ((ADCSRA & (1<<ADIF))==0);
ADCSRA|=(1<<ADIF);
return ADCW;
}

void main(void)
{
// Declare your local variables here
char str[20];
// Input/Output Ports initialization
// Port A initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) |
(0<<DDA2) | (0<<DDA1) | (0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) |
(0<<PORTA3) | (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) |
(0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) |
(0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
(0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) |
(0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In

```

```

DDRD= (0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) |
(0<<DDD2) | (0<<DDD1) | (0<<DDD0) ;
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
(0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0) ;

// ADC initialization
// ADC Clock frequency: 1000.000 kHz
// ADC Voltage Reference: AVCC pin
// ADC Auto Trigger Source: Free Running
ADMUX=ADC_VREF_TYPE;
ADCSRA=(1<<ADEN) | (0<<ADSC) | (1<<ADATE) | (0<<ADIF) | (0<<ADIE) |
(0<<ADPS2) | (1<<ADPS1) | (1<<ADPS0) ;
SFIOR=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0) ;

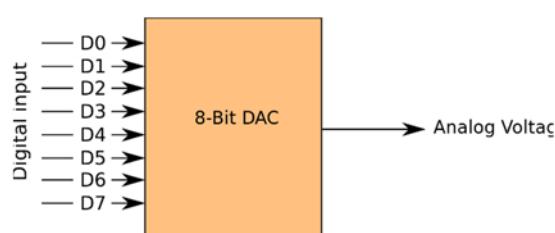
// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTB Bit 0
// RD - PORTB Bit 1
// EN - PORTB Bit 2
// D4 - PORTB Bit 4
// D5 - PORTB Bit 5
// D6 - PORTB Bit 6
// D7 - PORTB Bit 7
// Characters/line: 16
lcd_init(16);
while (1)
{
    // Place your code here
    read_adc(0);
    sprintf(str,"%d.%d", (ADCH*500)/1023, ((ADCH*500)/1023)%10);
    lcd_puts(str);
    delay_ms(1000);

}
}

```

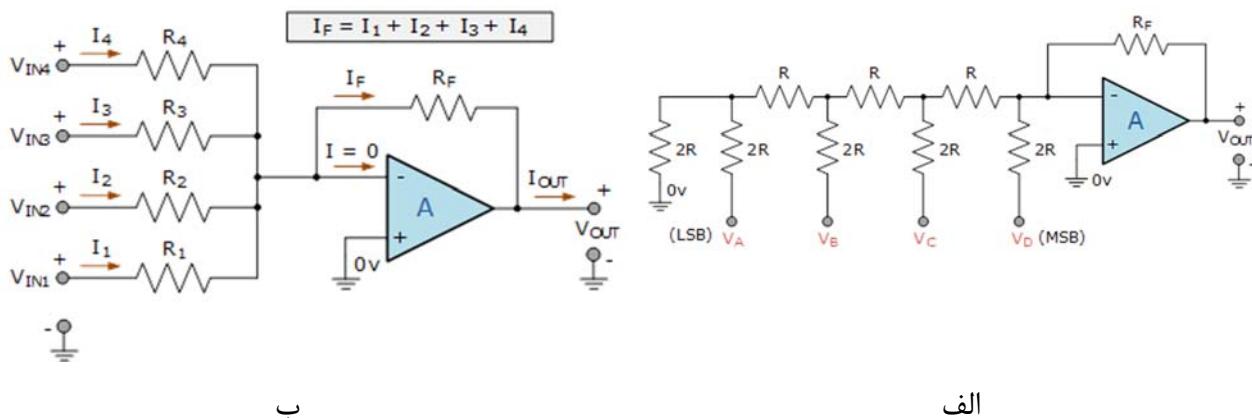
### 6.8 آشنایی با تبدیل سیگنال دیجیتال به آنالوگ

در تبدیل کننده‌های دیجیتال به آنالوگ مانند شکل 6-12، به ازای داده‌های دیجیتال ورودی، خروجی معادل آنالوگ به صورت ولتاژ یا جریان تولید می‌شود.



شکل 6-12: پلوک دیاگرام مبدل دیجیتال به آنalog

دقت یا وضوح خروجی بر حسب تعداد بیت‌های ورودی تعیین می‌شود به این صورت که اگر تعداد ورودی‌های DAC به تعداد  $n$  باشد، تعداد سطوح خروجی  $2^n$  خواهد بود. پروسه‌ی تبدیل سیگنال دیجیتال به آنalog مانند شکل 6-13 به دو روش نردبانی<sup>1</sup> و باینری وزن‌دار قابل انجام است.



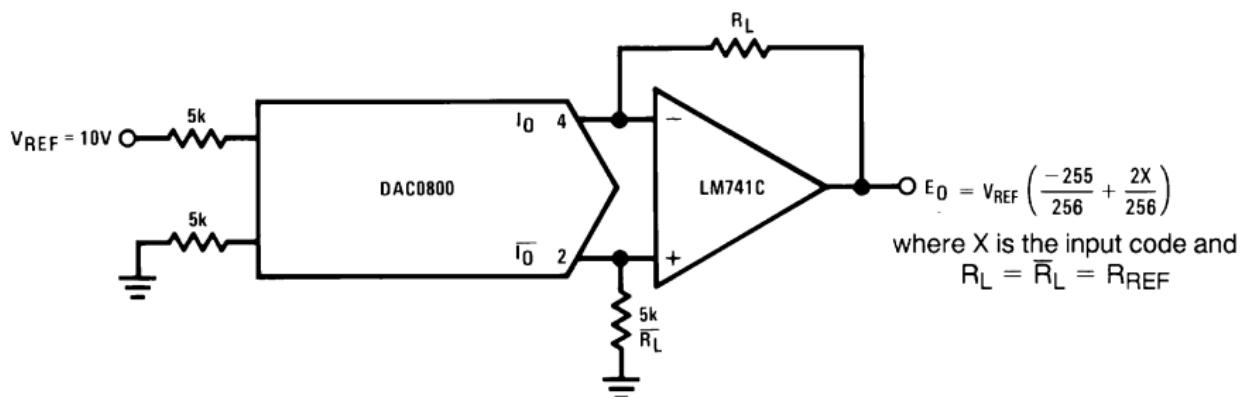
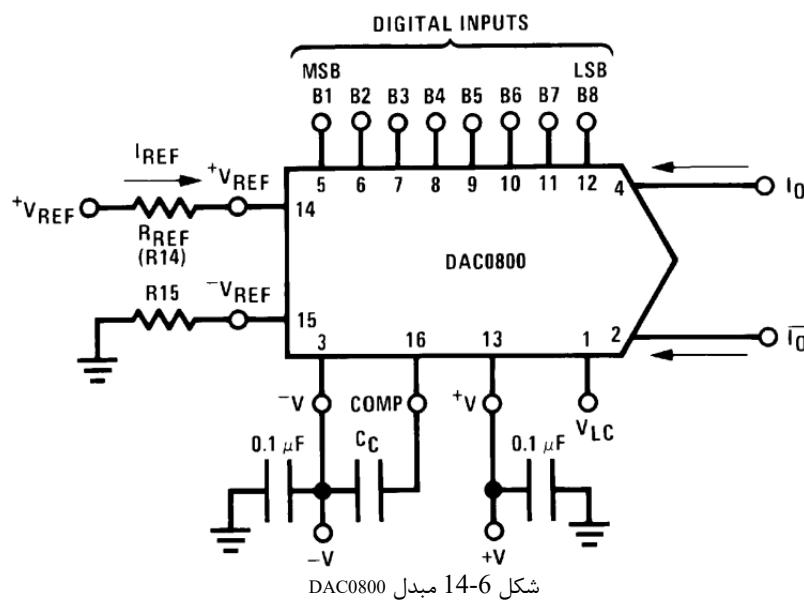
ب

الف

شکل 6-13: پروسه‌ی تبدیل سیگنال دیجیتال به آنalog، الف: نردبانی، ب: باینری وزن‌دار

در بورد آموزشی آزمایشگاه از مژول DAC0800 به عنوان یک تبدیل‌کننده دیجیتال به آنalog هشت بیتی به روش باینری وزن‌دار استفاده شده است که دارای 256 سطح در خروجی می‌باشد. در شکل 6-14 نمایی از آن نشان داده شده است. ورودی‌های دیجیتال از LSB تا MSB به ترتیب به پایه‌های B1 تا B8 متصل هستند. ولتاژ مرجع برای تبدیل به آنalog را هم می‌توان از تغذیه تراشه و یا ولتاژی که به پایه‌های Vref(+) و Vref(-) (به ترتیب در پایه‌های 14 و 15) اعمال می‌شود، تامین نمود. همچنین برای تقویت خروجی می‌توان مانند شکل 6-15 از یک تقویت کننده عملیاتی استفاده نمود.

<sup>1</sup> ladder



شکل 6-15: تقویت کننده در خروجی مبدل دیجیتال به آنالوگ

### 6.9 نمونه برنامه مبدل آنالوگ به دیجیتال

به عنوان نمونه با استفاده از برنامه 6-4 و با به کارگیری مبدل دیجیتال به آنالوگ می‌توان یک موج دندانه ارهاهی مانند شکل 6-16 طراحی نمود.

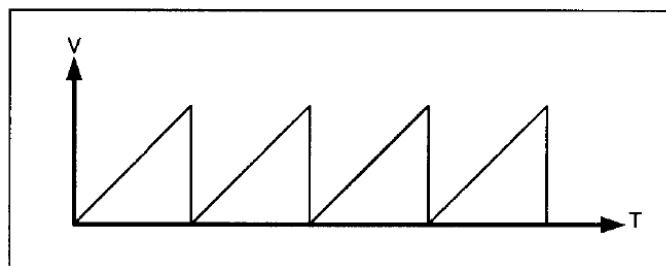
```
#include <mega16.h>
unsigned char i; //define a counter
int main (void)
{
    DDRB = 0xFF;
    while (1)
    {
        PORTB = i;
```

برنامه 6-4

```

    i++;
}
return 0;
}

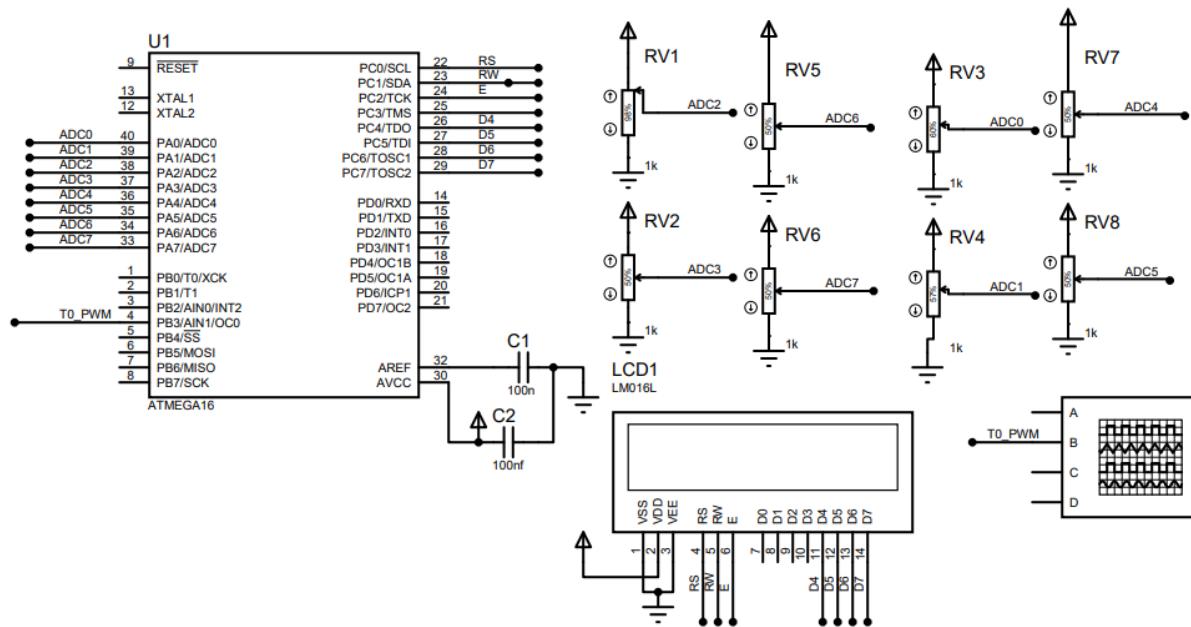
```



شکل 6-16 Step Ramp Output

## 6.10 برنامه‌های اجرایی مبدل‌های آنالوگ به دیجیتال

سخت‌افزار زیر را در نظر بگیرید:



1- زیربرنامه‌ای بنویسید که ولتاژ هر یک از ورودی‌های آنالوگ را بخواند و بر حسب میلی‌ولت روی LCD نشان دهد. (راهنمایی: تنظیمات CodeWizard ADC را برای بدون وقفه فعال نمایید. در کدهای ایجاد شده برای خواندن مقدار دیجیتال هر یک از کانال‌ها، از تابع `read_adc` استفاده می‌شود. پس از اجرای صحیح این بند، پیکربندی بخش ADC را در قالب یک زیر برنامه به همراه تابع `read_adc` به فایل جانبی منتقل نمایید تا بتوان از آن در بخش نهایی استفاده نمود.)

2- در یک پروژه جدید، زیر برنامه ای بنویسید که از طریق وقفه مقدار هر یک از ورودی های آنالوگ را بخواند و در صورتی که تغییرات بیش از 5 درصد باشد، آن را روی LCD نشان دهد. (آخرین مقدار نشان داده شده روی LCD را با مقدار جدید هر ورودی آنالوگ مقایسه نمایید و چنانچه بیش از 5٪ تغییر داشت، جهت نمایش بر روی LCD اقدام نمایید. بدین وسیله بار پردازشی ریزپردازنده کمتر شده و نوشته های روی LCD پرش خواهد داشت.

پس از اجرای صحیح این بند، پیکربندی بخش ADC را در قالب یک زیر برنامه به همراه روال سرویس دهی وقفه و تعاریف اولیه به فایل جانبی منتقل نمایید. این فایل در بخش نهایی استفاده خواهد شد).

3- زیر برنامه ای بنویسید که با استفاده از تایمر صفر، یک پالس PWM تولید نماید به صورتی که چرخه‌ی کار این پالس بر اساس سیگنال آنالوگ متصل به ADC0 تنظیم گردد. (راهنمایی: در این بخش مانند بند قبلی بایستی وقفه‌ی ADC فعال باشد).

4- بندهای فوق را در قالب یک پروژه در بیاورید. (راهنمایی: از فایل های جانبی تهیه شده در بندهای 1 و 2 استفاده نمایید. می‌توان ابتدا ADC را بدون وقفه فعال کرد و پس از نمایش ورودی های آنالوگ، مجددًا ADC را پیکربندی نمود و وقفه‌ی ADC را فعال کرد. سپس بند 2 و 3 را به طور همزمان اجرا نموده، به گونه‌ای که در حین اجرای برنامه تغییرات هر یک از ورودی ها نیز لحظه گردد).

## 7 جلسه هفتم

### آشنایی با ارتباط سریال

#### 7.1 هدف

در این جلسه نحوه برقراری ارتباط و تبادل داده بین ریزپردازنده و دستگاه‌های جنبی را از طریق ارتباط سریال USART بررسی خواهیم نمود.

#### 7.2 مقدمه

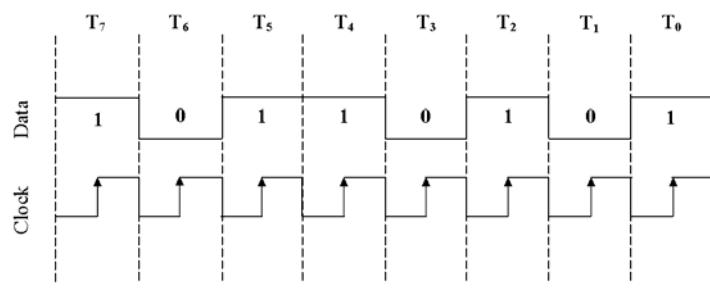
در سیستم‌های بزرگ و پیچیده دیجیتال که از اجزای متعدد تشکیل شده‌اند، لازم است به منظور ایجاد هماهنگی بین این بخش‌ها و اشتراک داده‌ها انتقال اطلاعات، یک بستر ارتباطی مناسب فراهم گردد. به صورت کلی انتقال اطلاعات در سیستم‌های دیجیتالی، به دو روش سریال و موازی صورت می‌گیرد. در روش موازی  $n$  بیت اطلاعات از طریق  $n$  خط داده به صورت همزمان منتقل می‌شود، اما در روش سریال بیت‌های داده از طریق یک خط و به صورت پست سر هم منتقل می‌شوند. در این جلسه به تشریح ارتباط سریال خواهیم پرداخت.

ارتباط سریال انواع مختلف دارد که از نظر حداکثر طول کابل ارتباطی، نرخ ارسال و دریافت داده، تعداد سیم ارتباطی، یک طرفه یا دو طرفه بودن، قالب ارسال و دریافت داده‌ها، سنکرون یا آسنکرون بودن و نوع مدولاسیون با یکدیگر تفاوت دارند. از انواع ارتباط سریال می‌توان به UART، USART، Ethernet، CAN، USB،<sup>1</sup> LonWorks، اشاره نمود.

ریزپردازنده Atmega16/32 به صورت سخت‌افزاری قابلیت برقراری ارتباط USART، SPI و I2C را برای ارتباط با وسایل جانبی از قبیل SD card، EEPROM و مبدل‌ها خارجی ADC یا DAC دارا می‌باشد. در این آزمایش به بررسی ارتباط USART پرداخته می‌شود.

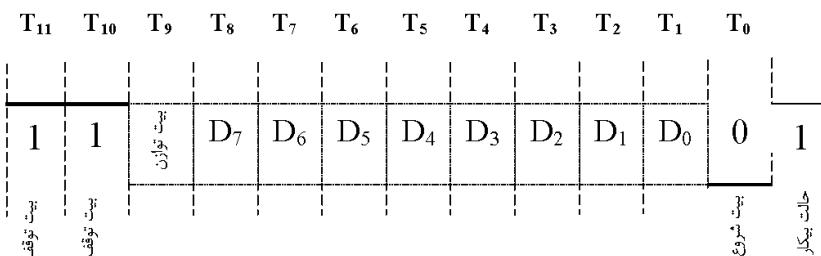
به طور کلی ارتباط سریال به دو صورت سنکرون و آسنکرون برقرار می‌شود. در ارتباط سنکرون مانند شکل 1-7، داده‌ها بر روی یک خط ارسال می‌شوند و یک خط پالس ساعت همزمان کننده نیز وجود دارد که به همراه داده‌ها برای سنکرون‌سازی مبدأ و مقصد، توسط فرستنده ارسال می‌شود.

<sup>1</sup> LonWorks or Local Operating Network is an open standard (ISO/IEC 14908) for networking platforms specifically created to address the needs of control applications.



شکل 7-1: ارتباط به صورت سنکرون

در ارتباط آسنکرون، پالس ساعت برای سنکرون‌سازی ارسال نمی‌شود. در چنین روشی باید داده‌ها تحت قالب‌بندی خاص و به صورت بیت به بیت با فواصل زمانی تعریف شده برای فرستنده و گیرنده منتقل شوند. تعداد این فواصل زمانی در واحد زمان، نرخ انتقال داده یا Baud Rate را تعیین می‌کند. در شکل 7-2 یک قالب داده با یک بیت توازن<sup>۱</sup> و دو بیت توقف در ارتباط آسنکرون مشاهده می‌شود.



شکل 7-2: ارتباط به صورت آسنکرون

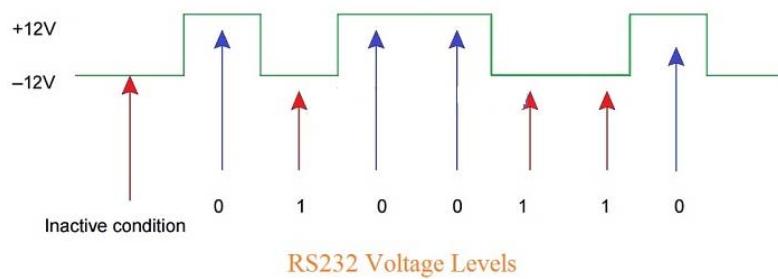
### 7.3 معرفی ارتباط (RS232) USART

یکی از پروتکل‌هایی که ریزپردازنده Atmega16/32 برای ارتباط سریال پشتیبانی می‌کند، پروتکل USART<sup>۲</sup> است. این پروتکل قابلیت برقراری ارتباط با هر دو حالت سنکرون و آسنکرون را دارد و برای افزایش قابلیت اطمینان تحت استانداردهای مختلفی کار می‌کند. یکی از این استانداردها RS232-C است که در سال 1969 توسط موسسه EIA تعریف شد. اگرچه نام این استاندارد RS232 است اما معمولاً به نام RS232 شناخته می‌شود و مخفف Recommended Serial فرستنده/گیرنده RF، GPS و GSM جهت ایجاد امکان ارتباط با ریزپردازنده استفاده می‌شود.

در استاندارد RS232 سطح ولتاژ +3.3V تا +12V نمایانگر وضعیت صفر منطقی و بازه‌ی -3V تا -12V ولت نمایشگر وضعیت یک منطقی می‌باشد. این در حالی است که تجهیزات مبتنی بر استاندارد TTL مثل ریزپردازنده Atmega16/32 در سطوح بین 0V و 5V کار می‌کنند.

<sup>1</sup> Parity

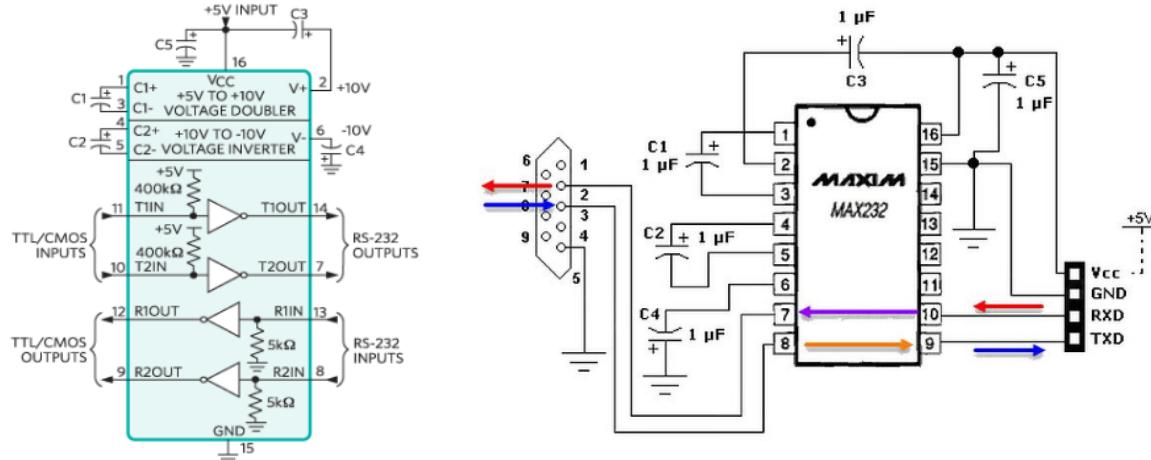
<sup>2</sup> Universal Synchronous Asynchronous serial Receiver/Transmitter



RS232 Voltage Levels

شکل 7-3: سطوح ولتاژ در RS232

در نتیجه برای برقراری ارتباط بین وسایل با سطوح ولتاژ متفاوت TTL و RS232، از یک تراشه درایور مانند MAX232 استفاده می‌شود تا سطح ولتاژ آن‌ها را به یکدیگر تبدیل کند. MAX232 یک تراشه‌ی 16 پایه شامل 2 فرستنده و 2 گیرنده است (شکل 4-7).



شکل 4-7: مدار داخلی MAX232 و نحوه اتصال تراشه به کانکتور مادگی DB9

#### 7.4 ارتباط Atmega16/32 و کامپیوتر

Atmega16/32 دارای 2 پایه با نام‌های TX و RX واقع در PD0 و PD1 برای دریافت و انتقال داده به صورت سریال مطابق با استاندارد RS232 است. البته برای ایجاد ارتباط با یک دستگاه دیگر مانند کامپیوتر نمی‌توان این پایه‌ها را مستقیماً به کار برد، زیرا سطح ولتاژ خروجی‌های ریزپردازنده بین صفر تا 5V+ است، در صورتی که سطح ولتاژ واحد سریال کامپیوتر بین 12V- و 12V+ می‌باشد. لذا از تراشه‌های مبدل مانند MAX232 برای این تبدیل استفاده می‌شود. همان طور که در شکل 4-7 نشان داده شده پایه TX به T2IN و پایه RX به R2OUT از تراشه MAX232 متصل می‌گردد. بعد از برقراری اتصال ریزپردازنده به MAX232، می‌توان این تراشه را به درگاه سریال کامپیوتر متصل نمود تا بستر ارتباط کامل گردد.

در گاه سریال در کامپیوترهای شخصی به نام‌های RS-232 Port یا COM Port شناخته می‌شود و دارای کانکتور DB9 مانند شکل 7-5 است که برای برقراری ارتباط سریال با دستگاه‌های خارجی مانند مودم، ماوس‌های سریال، قفل‌های دیجیتال و ... به کار می‌رود. این درگاه به تراشه‌ی UART تعبیه شده بر روی بورد اصلی کامپیوتر متصل شده و با استفاده از ثبات‌های این تراشه می‌توان اموری مانند ارسال و دریافت داده، خواندن وضعیت خط، کنترل سیگنال‌های handshaking و تنظیم Baud Rate وغیره را انجام داد.

1	Data carrier detect
2	Receive Data (RXD)
3	Transmit Data (TXD)
4	Data terminal ready (DTR)
5	GND
6	Data set ready
7	Request to send
8	Clear to send
9	Ring indicator



شکل 7-5. سمت راست کانکتور مادگی DB9 - سمت چپ کانکتور نری DB9 و پایه‌های کانکتور

البته باید توجه داشت که امروزه اکثر کامپیوترها فاقد کانکتور DB9 هستند و به جای آن از مبدل‌های usb to uart استفاده می‌گردد. مازول CP2102 یکی از این گونه مبدل‌ها است.

## 7.5 ثبات‌های ارتباط USART

در ادامه به معرفی ثبات‌های مربوط به ارتباط USART در AVR شامل UDR، UCSRA، UCSRB، UCSRC و UBRR خواهیم پرداخت.

### 7.5.1 ثبات UDR

این ثبات اصطلاحاً ثبات بافر داده نامیده می‌شود و داده‌های ارسالی و دریافتی در آن ریخته می‌شوند. مطابق شکل 7-6 این ثبات 16 بیتی از دو بخش RXB[7:0] و TXB[7:0] تشکیل شده است. با استفاده از ثبات UDR می‌توان انتقال Full Duplex را انجام داد. به عبارت دیگر می‌توان به طور همزمان در یک جهت اطلاعات را بر روی پایه TX ارسال و در جهت دیگر داده‌ها را از روی پایه RX دریافت نمود.

Bit	7	6	5	4	3	2	1	0	UDR (Read)	UDR (Write)
RXB[7:0]										
TXB[7:0]										
Read/Write	R/W									
Initial Value	0	0	0	0	0	0	0	0		

شکل 7-6. ساختار ثبات UDR

## 7.5.2 ثبات UCSRA<sup>1</sup>

این ثبات مانند شکل 7-7 برای کنترل و نمایش وضعیت ارتباط به کار می‌رود.

Bit	7	6	5	4	3	2	1	0	UCSRA
Read/Write	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	
Initial Value	0	0	1	0	0	0	0	0	

شکل 7-7: ساختار ثبات UCSRA

بیت 7 - (USART Receive Complete) RXC: در صورتی که داده‌های موجود در ثبات گیرنده خوانده نشده باشند، این پرچم برابر یک و در غیر این صورت صفر خواهد بود.

بیت 6 - (USART Transmit Complete) TXC: زمانی که آخرین بسته فرستاده شده باشد و داده‌ای در ثبات فرستنده وجود نداشته باشد، این بیت برابر یک و در غیر این صورت صفر خواهد بود.

بیت 5 - (USART Data Register Empty) UDRE: در صورت یک بودن، ثبات فرستنده آماده دریافت داده‌ی جدید می‌باشد.

بیت 4 - (Frame Error) FE: اگر کاراکتر بعدی در زمان دریافت در بافر گیرنده، خطای بسته داشته باشد این بیت یک می‌شود.

بیت 3 - (Data Over Run) DOR: زمانی که بافر گیرنده پر باشد و کاراکتر جدیدی هم در ثبات گیرنده منتظر باشد و بیت شروع جدیدی هم تشخیص داده شود، این بیت یک می‌گردد.

بیت 2 - (Parity Error) PE: اگر در کاراکتر بعدی موجود در بافر گیرنده خطای parity وجود داشته باشد، این بیت یک می‌شود.

بیت 1 - (Double the USART transmission Speed) UDX: با نوشتن یک در این بیت، نرخ ارسال به جای 16 بر 8 تقسیم می‌گردد و به این ترتیب در ارتباط آسنکرون سرعت انتقال را 2 برابر می‌کند. البته این بیت تنها در حالت آسنکرون کاربرد دارد و در حالت سنکرون باید صفر گردد.

بیت صفر - (Multi-Processor communication Mode) MPCM: این بیت امکان ارتباط چند ریزپردازنده با یکدیگر را فراهم می‌کند.

<sup>1</sup> USART Control and Status Register

### UCSRB ثبات 7.5.3

این ثبات نیز مانند شکل 7-8 برای کنترل و نمایش وضعیت ارتباط به کار می‌رود.

Bit	7	6	5	4	3	2	1	0	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

شکل 7-8: ساختار ثبات UCSRБ

بیت 7 - RXCIE (RX Complete Interrupt Enable): با یک کردن این بیت، وقفه اتمام دریافت فعال می‌شود.

بیت 6 - TXCIE (TX Complete Interrupt Enable): با یک کردن این بیت، وقفه مربوط به اتمام ارسال فعال می‌گردد.

بیت 5 - UDRIE (USART Data Register Empty Interrupt Enable): با یک کردن این بیت، وقفه مربوط به خالی شدن بافر فعال می‌گردد.

بیت 4 - RXEN (Receiver Enable): با یک کردن این بیت، USART به صورت گیرنده فعال می‌گردد.

بیت 3 - TXEN (Transmitter Enable): با یک کردن این بیت، USART به صورت فرستنده فعال می‌گردد.

بیت 2 - UCSZ (Character Size): این بیت در کنار بیت‌های USCZ 1:0، تعداد بیت‌های داده در یک بسته را مشخص می‌کند.

بیت 1 - RXB8 (Receive Data Bit 8): زمانی که بسته 9 بیت داده داشته باشد، بیت نهم مربوط به داده دریافتی را در خود جای می‌دهد.

بیت صفر - TXB8 (Transmit Data bit 8): زمانی که بسته 9 بیت داده داشته باشد، بیت نهم مربوط به داده ارسالی را در خود جای می‌دهد.

### UCSRC ثبات 7.5.4

این ثبات نیز مانند شکل 7-9 برای کنترل و نمایش وضعیت ارتباط به کار می‌رود که از فضای آدرس مشترک با رجیستر UBRRH استفاده می‌نماید.

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	UCSRC							
Initial Value	1	0	0	0	0	1	1	0	

شکل 7-9: ساختار ثبات UCSRC

بیت 7 - URSEL (register select): با توجه به مشترک بودن فضای آدرس، این بیت دسترسی به ثبات UBRRH را مشخص می‌نماید. هنگام خواندن یا نوشتan UCSRC این بیت باید برابر با یک باشد.

بیت 6 - UMSEL (USART Mode Select): این بیت، حالت عملکرد سنکرون و آسنکرون را تعیین می‌کند.

بیت‌های 5 و 4 - UPM1:0 (Parity Mode): این بیت‌ها وضعیت parity را تنظیم می‌نمایند.

بیت 3 - USBS (STOP Bit Select): تعداد بیت‌های STOP که فرستنده بایستی اضافه نماید را مشخص می‌کند. این بیت برای گیرنده کابردی ندارد.

بیت‌های 2 و 1 - UCSZ (Character size): تعداد بیت‌های داده را در بسته ارسالی و دریافتی تعیین می‌کنند.

بیت صفر - این بیت فقط برای مد سنکرون استفاده می‌شود. اگر مقدار آن 1 باشد، فرستنده/گیرنده به ترتیب در لبه پایین‌رونده بالا‌رونده و اگر 0 باشد فرستنده/گیرنده به ترتیب در بالا‌رونده/لبه پایین‌رونده پالس ساعت سنکرون، نمونه برداری می‌نماید.

### 7.5.5 ثبات‌های UBRRH و UBRRRL (USART Baud rate Register)

این ثبات مانند شکل 7-10 برای تنظیم نرخ ارسال داده استفاده می‌شود. همان‌گونه که پیش‌تر گفته شد، ثبات UBRRH و ثبات UCSRC از یک آدرس مشترک در حافظه I/O استفاده می‌کنند.

Bit	15	14	13	12	11	10	9	8	UBRRH
Read/Write	URSEL	-	-	-	UBRR[11:8]				UBRRL
Initial Value	0	0	0	0	0	0	0	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	UBRRH
Initial Value	0	0	0	0	0	0	0	0	UBRRL

شکل 7-10: ساختار ثبات UBRR

بیت 15-URSEL (Register Select): این بیت برای انتخاب دسترسی به ثبات‌های UBRRH و UCSRC به کار می‌رود و برای دسترسی به ثبات UBRRH باید صفر شود.

بیت‌های 14 تا 12-Reserved Bits: این بیت‌ها برای استفاده‌های بعدی ذخیره شده‌اند.

بیت‌های 11 تا 0-UBRR: این یک ثبات 12 بیتی است که نرخ ارسال USART را تنظیم می‌کند. البته باید توجه داشت که برای ارسال داده با نرخ‌های متفاوت لازم است که فرکانس واحد USART به صورت ضریبی از فرکانس اصلی ریزپردازنده ایجاد گردد، که این تبدیل فرکانسی با درصد خطای مشخصی قابل انجام خواهد بود. برای دستیابی به کیفیت مطلوب، نرخ خطای کمتر از ۰.۵٪ قابل قبول خواهد بود.

جدول 7-1: تنظیمات نرخ ارسال داده

Baud Rate (bps)	$f_{osc} = 8.0000 \text{ MHz}$			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%
4800	103	0.2%	207	0.2%
9600	51	0.2%	103	0.2%
14.4k	34	-0.8%	68	0.6%
19.2k	25	0.2%	51	0.2%
28.8k	16	2.1%	34	-0.8%
38.4k	12	0.2%	25	0.2%
57.6k	8	-3.5%	16	2.1%
76.8k	6	-7.0%	12	0.2%
115.2k	3	8.5%	8	-3.5%
230.4k	1	8.5%	3	8.5%
250k	1	0.0%	3	0.0%
0.5M	0	0.0%	1	0.0%
1M	-	-	0	0.0%
Max <sup>(1)</sup>	0.5 Mbps		1 Mbps	

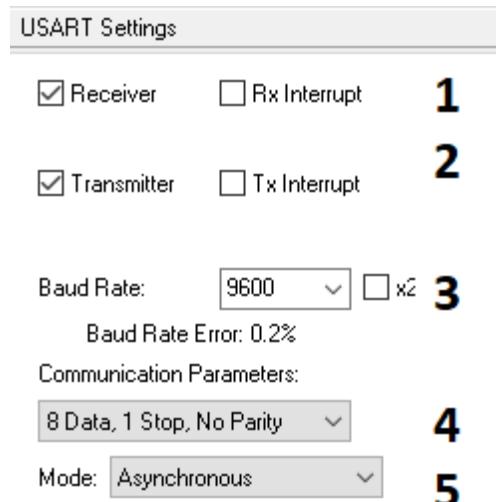
1. UBRR = 0, Error = 0.0%

## 7.6 stdio.h کتابخانه

برای برقراری ارتباط با پایه‌های USART ریزپردازinde از فایل سرآیند stdio.h استفاده می‌شود. برای اطلاع از آخرین تغییرات این فایل کتابخانه ای به راهنمای نرمافزار Codevision مراجعه نمایید. این فایل سرآیند شامل توابعی برای ارسال و دریافت کاراکتر و رشته بر روی خط سریال مانند printf، scanf، puts و gets می‌باشد. همه‌ی این توابع بر پایه‌ی دستورات putchar(char c) و getchar() تعریف و پیاده‌سازی شده‌اند که این دو به ترتیب کاراکتری را از روی خط سریال دریافت و بر روی آن ارسال می‌نمایند.

### 7.7 برنامه نویسی در محیط **Codevision** برای **USART**

با استفاده از CodeWizard، تنظیمات ثبات‌ها به صورت خودکار انجام می‌شود. لذا پارامترهای اولیه مانند شکل 11-7 تنظیم می‌گردد.



1: فرستنده فعال می‌شود.

2: گیرنده فعال می‌شود.

3: نرخ ارسال مورد نظر تنظیم می‌گردد.

4: قالب بسته ارسالی مشخص می‌گردد.

5: حالت سنکرون و آسنکرون انتخاب می‌گردد.

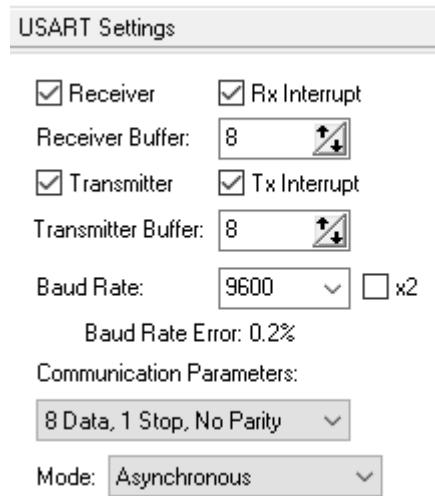
شكل 11-7: تنظیمات **UART** در **CodeWizard**

در تنظیم نرخ ارسال بایستی توجه داشت که برای کارکرد مناسب باید نرخ احتمال خطأ کمتر از ۰.۵٪ باشد. بنابراین نرخ ارسال‌های بالاتر که ممکن است خطای بیشتری داشته باشند مناسب نخواهند بود.

پس از انجام تنظیمات مورد نیاز می‌توان کد را ذخیره نمود و در ادامه توابع موجود در فایل سرآیند را به کار گرفت. در این روش با ارسال داده، اطلاعات در قالب بسته مورد نظر روی TX ارسال می‌گردد و برای دریافت اطلاعات هم منتظر می‌ماند تا داده‌ای از RX دریافت نماید.

### 7.7.1 **UART** وقفه‌های بخش

سرکشی مداوم به ارسال و دریافت در بخش **UART**، وقت زیادی از ریزپردازنده را صرف می‌نماید. لذا بهتر است از روش مبتنی وقفه در ارسال و دریافت داده استفاده گردد که تنظیمات آن در شکل 12-7 نشان داده شده است.



- 1: وقفه گیرنده فعال می‌شود.
- 2: اندازه بافر گیرنده تنظیم می‌شود.
- 3: وقفه فرستنده فعال می‌گردد.
- 4: اندازه بافر فرستنده تنظیم می‌شود.

شکل 7-12: تنظیمات UART در حالت مبتنی بر وقفه

در حالت استفاده از وقفه، اندازه بافر فرستنده و گیرنده با توجه به شرایط پروژه، حجم کاری ریزپردازنده و نحوه پردازش داده‌ها تعیین می‌گردد. برنامه 7-1 نحوه تبادل داده بر روی خط سریال بر مبنای وقفه را نشان می‌دهد. بدیهی است در این حالت با توجه به این که تابع اصلی مستقل از روال عملکرد سخت‌افزار UART کار خود را به پیش می‌برد، ماهیت دستورات `putchar` و `getchar` نسبت به حالت سرکشی تغییر اساسی پیدا کرده و از یک سری بافر جهت نگهداری و مدیریت داده‌های دریافتی و ارسالی خط UART استفاده می‌گردد. این تفاوت در انتهای همین بخش بیشتر بررسی خواهد شد.

```
1-7 برنامه #include <mega16.h>

// Declare your global variables here

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

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

#if RX_BUFFER_SIZE <= 256
unsigned char rx_wr_index=0, rx_rd_index=0;
```

```

#else
unsigned int rx_wr_index=0,rx_rd_index=0;
#endif

#if RX_BUFFER_SIZE < 256
unsigned char rx_counter=0;
#else
unsigned int rx_counter=0;
#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)
{
    rx_buffer[rx_wr_index++]=data;
#endif RX_BUFFER_SIZE == 256
    // special case for receiver buffer size=256
    if (++rx_counter == 0) rx_buffer_overflow=1;
#else
    if (rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
    if (++rx_counter == RX_BUFFER_SIZE)
    {
        rx_counter=0;
        rx_buffer_overflow=1;
    }
#endif
}
}

#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++];
#endif RX_BUFFER_SIZE != 256
if (rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#endif
#asm("cli")
--rx_counter;
#asm("sei")
return data;

```

```

}

#pragma used-
#endif

// USART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE <= 256
unsigned char tx_wr_index=0,tx_rd_index=0;
#else
unsigned int tx_wr_index=0,tx_rd_index=0;
#endif

#if TX_BUFFER_SIZE < 256
unsigned char tx_counter=0;
#else
unsigned int tx_counter=0;
#endif

// USART Transmitter interrupt service routine
interrupt [USART_TxC] void usart_tx_isr(void)
{
if (tx_counter)
{
    --
    tx_counter;
    UDR=tx_buffer[tx_rd_index++];
#if TX_BUFFER_SIZE != 256
    if (tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
#endif
}
}

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY)==0))
{
    tx_buffer[tx_wr_index++]=c;
#if TX_BUFFER_SIZE != 256
    if (tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
#endif
    ++tx_counter;
}
else
    UDR=c;
#asm("sei")
}

```

```

#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

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

    // Input/Output Ports initialization
    // Port A initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
    Bit0=In
    DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) |
    (0<<DDA2) | (0<<DDA1) | (0<<DDA0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
    PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) |
    (0<<PORTA3) | (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

    // Port B initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
    Bit0=In
    DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) |
    (0<<DDB2) | (0<<DDB1) | (0<<DDB0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
    PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) |
    (0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

    // Port C initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
    Bit0=In
    DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
    (0<<DDC2) | (0<<DDC1) | (0<<DDC0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
    PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) |
    (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

    // Port D initialization
    // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
    Bit0=In
    DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) |
    (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
    // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
    PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) |
    (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

    // USART initialization
    // Communication Parameters: 8 Data, 1 Stop, No Parity
    // USART Receiver: On
    // USART Transmitter: On
    // USART Mode: Asynchronous
    // USART Baud Rate: 9600

```

```

UCSRA= ( 0<<RXC) | ( 0<<TXC) | ( 0<<UDRE) | ( 0<<FE) | ( 0<<DOR) | ( 0<<UPE)
| ( 0<<U2X) | ( 0<<MPCM) ;
UCSRB= (1<<RXCIE) | (1<<TXCIE) | (0<<UDRIE) | (1<<RXEN) | (1<<TXEN) |
(0<<UCSZ2) | (0<<RXB8) | (0<<TXB8) ;
UCSRC= (1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) |
(1<<UCSZ1) | (1<<UCSZ0) | (0<<UCPOL) ;
UBRRH=0x00;
UBRRL=0x33;

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

while (1)
{
    // Place your code here
}

}

```

### RX وقفه‌ی 7.7.2

وقتی که داده جدیدی در بخش RX از ثبات UDR قرار گرفت، وقفه گیرنده رخ می‌دهد و در زیربرنامه مربوطه، این داده در فضایی از بافر گیرنده که با شاخص rx\_wr\_index اشاره شده ذخیره می‌گردد و متغیر rx\_counter یک واحد افزایش می‌یابد. در حقیقت متغیر rx\_counter تعداد داده‌های موجود در بافر گیرنده که هنوز توسط ریزپردازنده بررسی نشده‌اند را نگه می‌دارد.

در زمان دریافت داده، ریزپردازنده با استفاده از توابع فایل‌های سرآیند که مبتنی بر تابع getchar هستند به فضای بافر گیرنده دسترسی پیدا خواهد کرد. تابع getchar داده‌ای از بافر گیرنده که توسط شاخص rx\_rd\_index مشخص شده را دریافت و متغیر rx\_counter را یک واحد کاهش می‌دهد.

یک متغیر تک بیتی هم برای تشخیص سرریز بافر گیرنده وجود دارد که توسط کاربر می‌تواند مورد استفاده قرار گیرد.

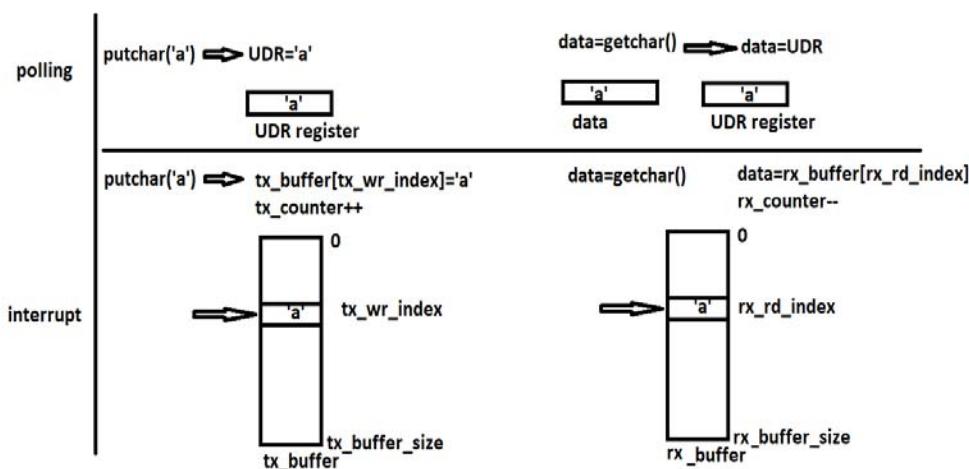
### TX وقفه‌ی 7.7.3

وقتی که داده قبلی بر روی خط سریال ارسال گردد، وقفه فرستنده رخ می‌دهد و در زیربرنامه مربوطه، داده‌ی جدیدی از فضای بافر فرستنده که با شاخص tx\_rd\_index اشاره شده به بخش TX از ثبات UDR منتقل می‌شود و متغیر tx\_counter یک واحد کاهش می‌یابد. متغیر tx\_counter تعداد داده‌هایی که در بافر فرستنده وجود دارند و هنوز توسط UART ارسال نشده‌اند را درون خود نگه می‌دارد.

در زمان ارسال داده، ریزپردازنده با استفاده از دستورات فایل‌های سرآیند که مبتنی بر putchar هستند به فضای بافر فرستنده دسترسی دارد. دستور putchar داده را در بافر با شاخص tx\_wr\_index ذخیره می‌نماید و متغیر tx\_counter را یک واحد افزایش می‌دهد.

#### 7.7.4 تغییر ماهیت توابع putchar و getchar

فرایندهای متفاوتی که برای استفاده از توابع putchar و getchar در حالت سرکشی و وقفه رخ می‌دهد در شکل 7-13 نشان داده شده است.

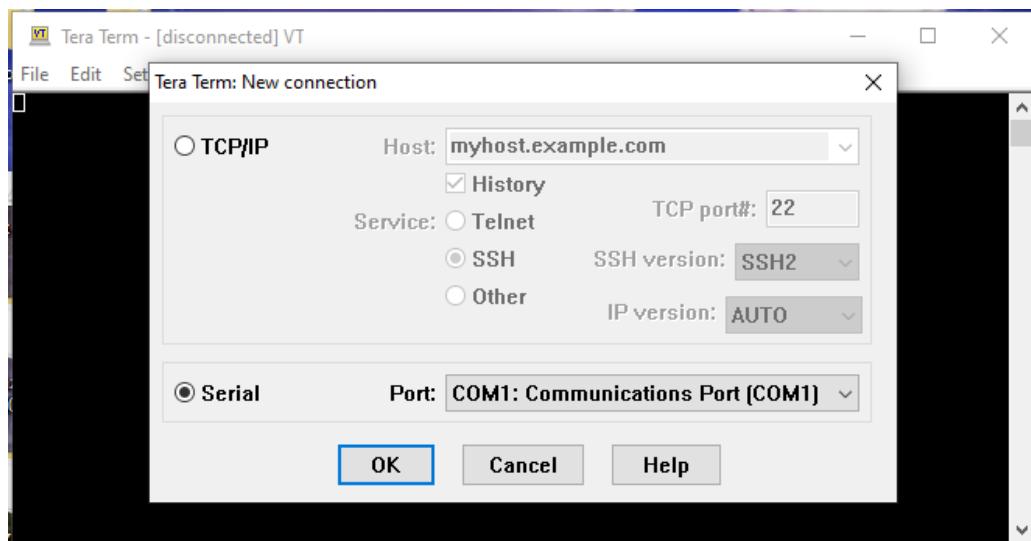


شکل 7-13: تفاوت ماهیت توابع putchar و getchar در حالت سرکشی و وقفه ارتباط UART

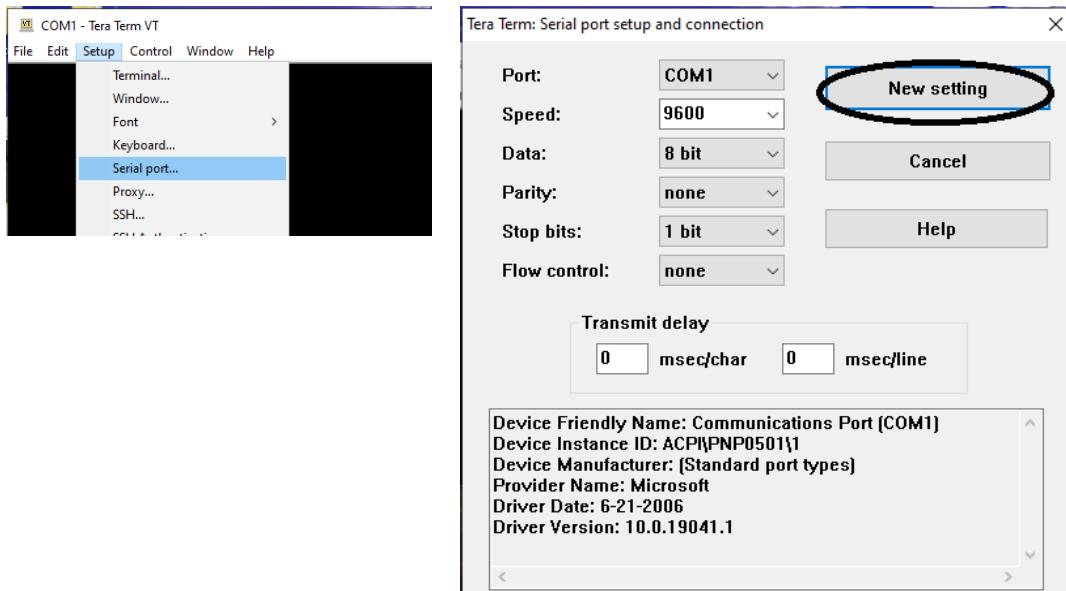
#### 7.8 نرم افزار Teraterm

پس از برقراری ارتباط بین پایه‌های RX و TX ریزپردازنده و درگاه ارتباط سریال کامپیوتر مطابق آن‌چه در بخش‌های پیشین توضیح داده شد، اکنون می‌توان از نرم افزارهای کاربردی در کامپیوتر مانند Teraterm برای نمایش داده‌های دریافتی از سمت ریزپردازنده و هم‌چنین ارسال داده برای آن استفاده نمود. مراحل کار با این نرم‌افزار در شکل 7-14 نشان داده شده است.

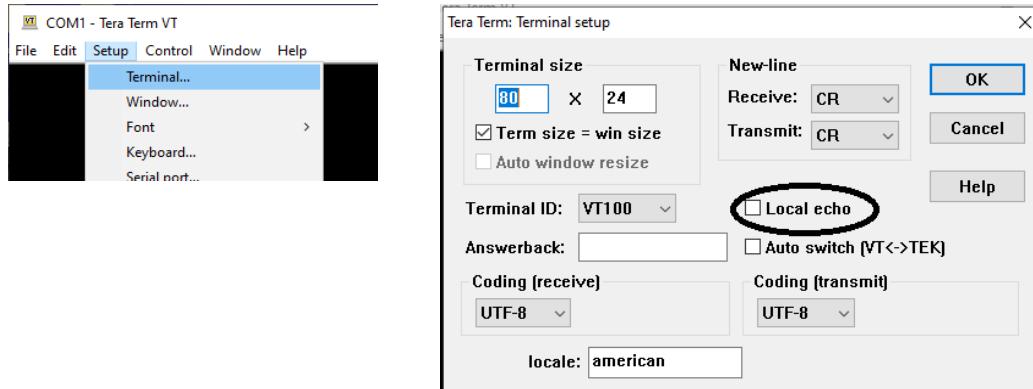
الف



ب



ج

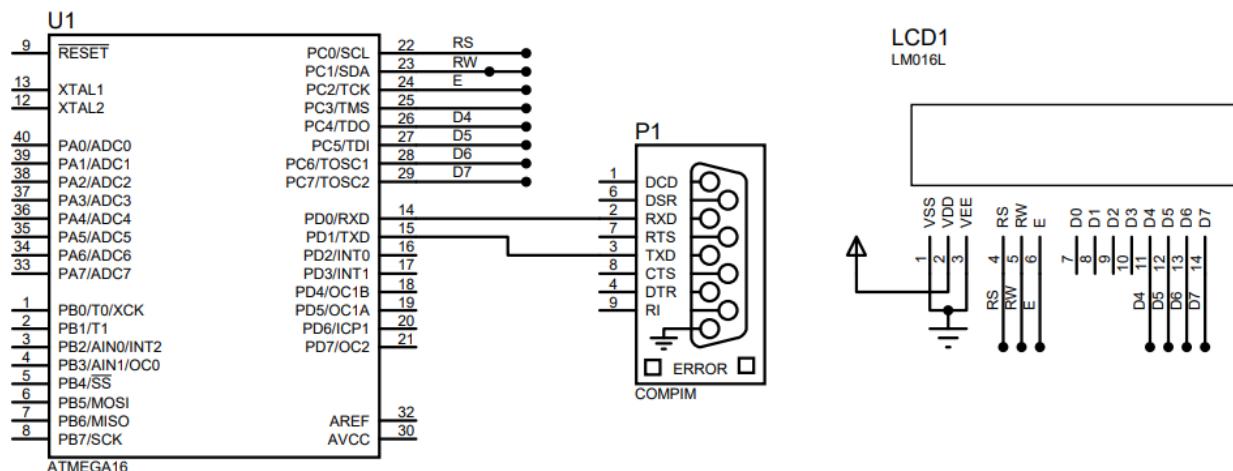


شکل 14-7: نمایی از نحوه کار با محیط Teraterm

## 7.9 برقراری ارتباط سریال در نرمافزار MATLAB

با استفاده از ارتباط UART می‌توان سخت‌افزار را به نرم افزارهای توانمندی مانند Matlab و Labview نیز متصل نمود. لذا در این بخش نحوه اتصال به نرمافزار Matlab از دو طریق محیط شبیه‌سازی پروتئوس و سخت‌افزار واقعی شرح داده خواهد شد.

برای برقراری ارتباط نرمافزار پروتئوس به Matlab R2021b از طریق UART می‌توان از سخت‌افزار شکل 7-15 استفاده نمود. این طرح با استفاده از کانکتور COMPIM به درگاه مجازی متصل می‌گردد. برای تعریف و ایجاد درگاه‌های مجازی هم نرمافزار Virtual Serial Port Driver Pro مورد استفاده قرار می‌گیرد. کدهای نوشته شده در محیط‌های Matlab و CodeVision به ترتیب در برنامه 7-3 و برنامه 7-2 آمده است. لازم به ذکر است که در برنامه وقفه‌های UART فعال شده است.



شکل 7-15: سخت‌افزار مورد نظر جه ایجاد اتصال به Matlab در محیط پروتئوس

```

s = serialport("COM2",9600);
a=[];
while(1)
    write(s,1,"uint8");
    b= read(s,10,"uint8");
    for x=1:length(b)
        if isnumeric(b(x))
            a=[a,b(x)];
        end
        if(length(a)<400 || length(a)==400)
            plot(a);
            axis([1 400 0 15]);
        else
            plot(a(end-399:end));
            axis([1 400 0 15]);
        end
    end
end

```

برنامه 2-7

```

        end
    end
b=[];
grid on;
drawnow;
end

```

برنامه 3-7

```

while (1)
{
    lcd_putchar(getchar () +0x30);
    for (i=0;i<10; i++) putchar(i);
    delay_ms(100);
}

```

برای اجرای برنامه ابتدا درگاه مجازی مانند شکل 7-16 تعریف گردیده و سپس به ترتیب برنامه‌های پروتئوس و Matlab اجرا می‌شود. دقت شود در صورتی که نیاز به تغییر برنامه باشد باید درگاه مجازی پاک و دوباره ایجاد گردد و سپس روند مذکور تکرار شود.

Open “Virtual Serial Port Driver Pro”

Select “Pair”

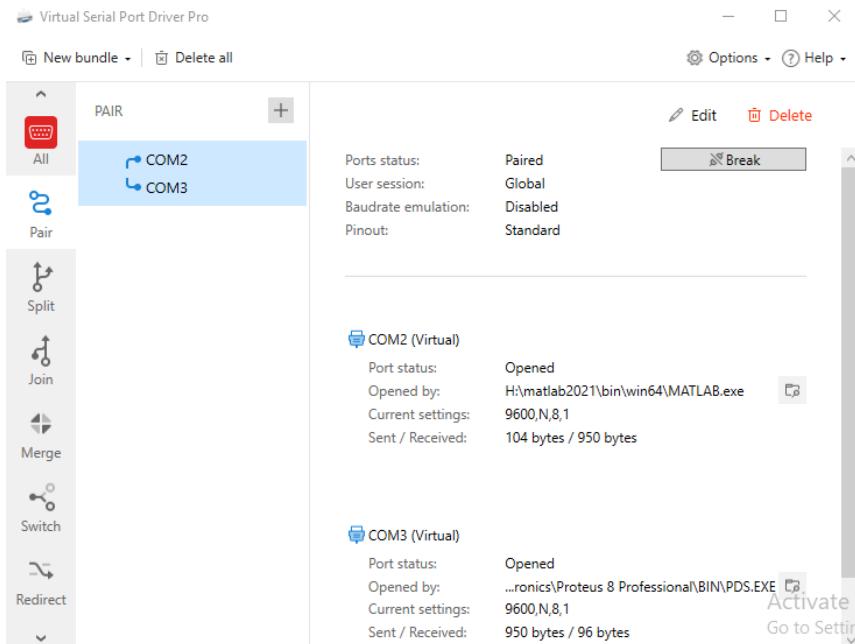
Select “add a new pair”

Select “Create”

.....

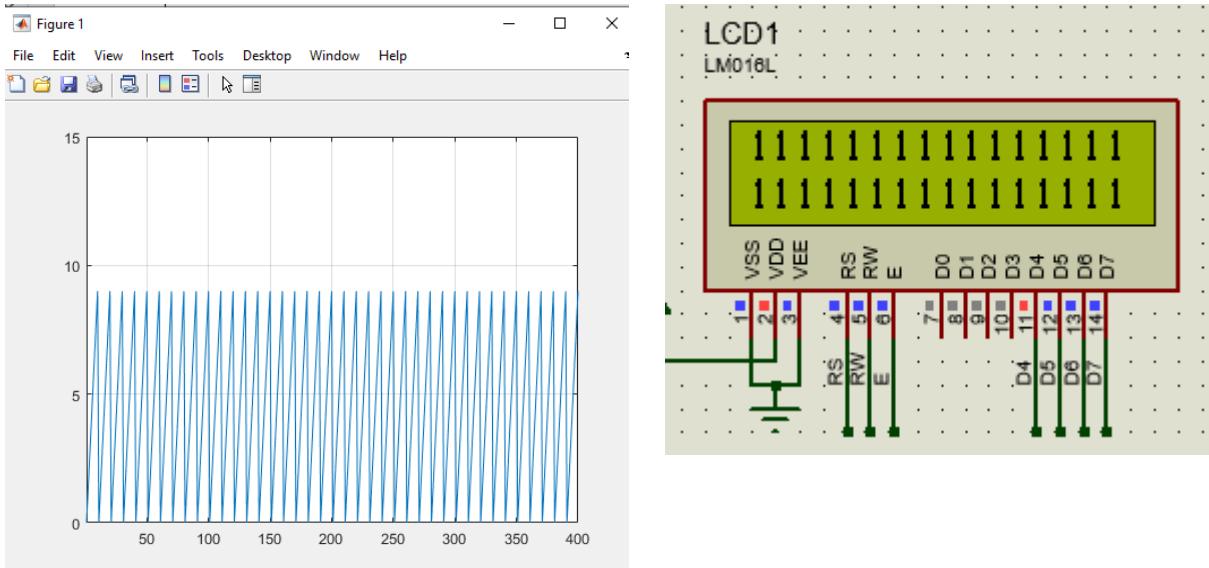
For end of connection :  
select “Delete”

On pair section



شکل 7-16: فعال نمودن درگاه مجازی

رونده اجرای این برنامه در شکل 7-17 نشان داده شده است.

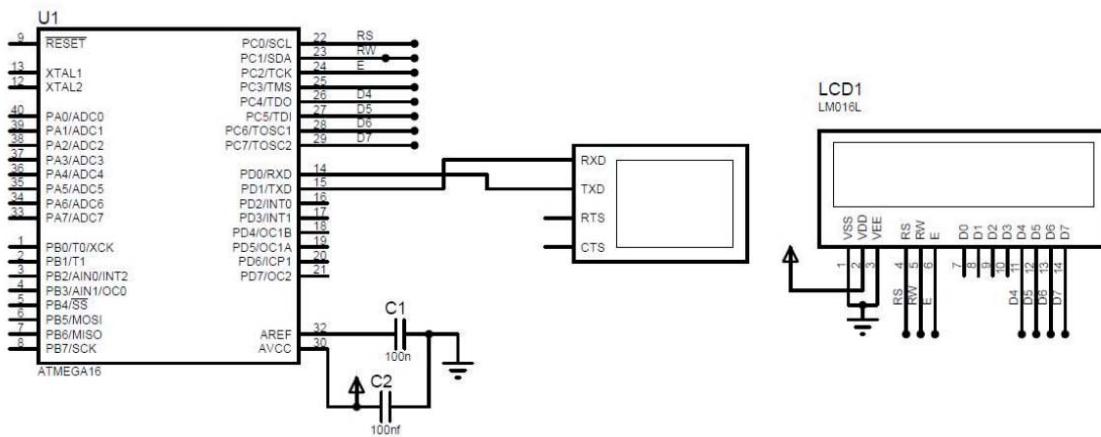


شکل ۱-۷: اجرای برنامه در محیط پروتوبوس و Matlab

برای اتصال سختافزار واقعی به نرمافزار Matlab فقط کافی است در گاه مورد نظر به درستی و به جای در گاه مجازی انتخاب شود و پس از آن ابتدا سختافزار و سپس برنامه Matlab اجرا گردد.

## 7.10 برنامه‌های اجرایی ارتباط سریال

سیستم طراحی شده در شکل 7-18 را در نظر بگیرید.



4. برنامه‌های فوق را در قالب دو پروژه مستقل ارائه دهید. اولین پروژه شامل بند یک و پروژه‌ی دیگر شامل بندهای 2 و 3 باشد. (چنان‌چه هر سه بند در یک پروژه باشد با توجه به تغییر ماهیت دستورات `getchar` و `putchar` با فعال شدن وقفه، بند یک به درستی اجرا نمی‌شود). پیشنهاد می‌گردد در ابتدای اجرای هر بند پیام‌هایی را مطابق زیر نمایش دهید.

Part 2 is running!

اجرای بند دوم

Part 2 is ending!

Part 3 is running!

اجرای بند سوم

Part 3 is ending!

## 8 جلسه هشتم

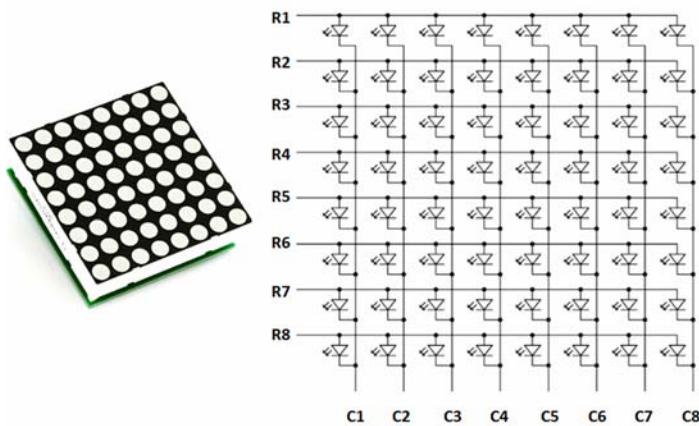
### آشنایی با نمایشگرهای LCD و Dot-matrix گرافیکی

#### هدف 8.1

در این جلسه به بررسی نمایشگرهای LCD و Dot-Matrix گرافیکی می‌پردازیم و ویژگی‌ها و نحوه کار هریک از آن‌ها را تشریح خواهیم نمود.

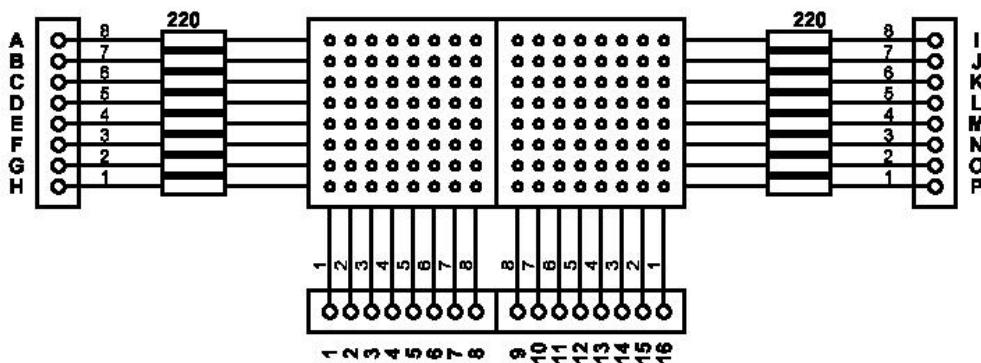
#### Dot-Matrix 8.2 نمایشگر

این نمایشگرها مانند شکل 8-1 از تعدادی LED تشکیل شده‌اند که در هر ردیف دارای آندهای مشترک و در هر ستون با کاتدهای مشترک می‌باشد. برای روشن شدن هر یک از LED‌ها بایستی سطر و ستون مورد نظر به نحو مناسب فعال گردد.



شکل 8-1: نمایی از ساختار Dot-Matrix

در مجموعه آموزشی آزمایشگاه، دو عدد نمایشگر Dot-Matrix از نوع 8x8 (8 ستون و 8 ردیف) به رنگ سبز در بلوکی تحت عنوان Dot-Matrix Display قرار داده شده است. ستون‌ها توسط اعداد 1 تا 16 و ردیف‌ها توسط حروف A تا P نام‌گذاری شده‌اند. لذا برای روشن نمودن هر کدام از LED‌ها می‌توان به ستون مربوط به همان LED صفر منطقی و به ردیف متناظر یک منطقی اعمال نمود. شماتیک مربوط به این بلوک در شکل 8-2 مشاهده می‌شود.

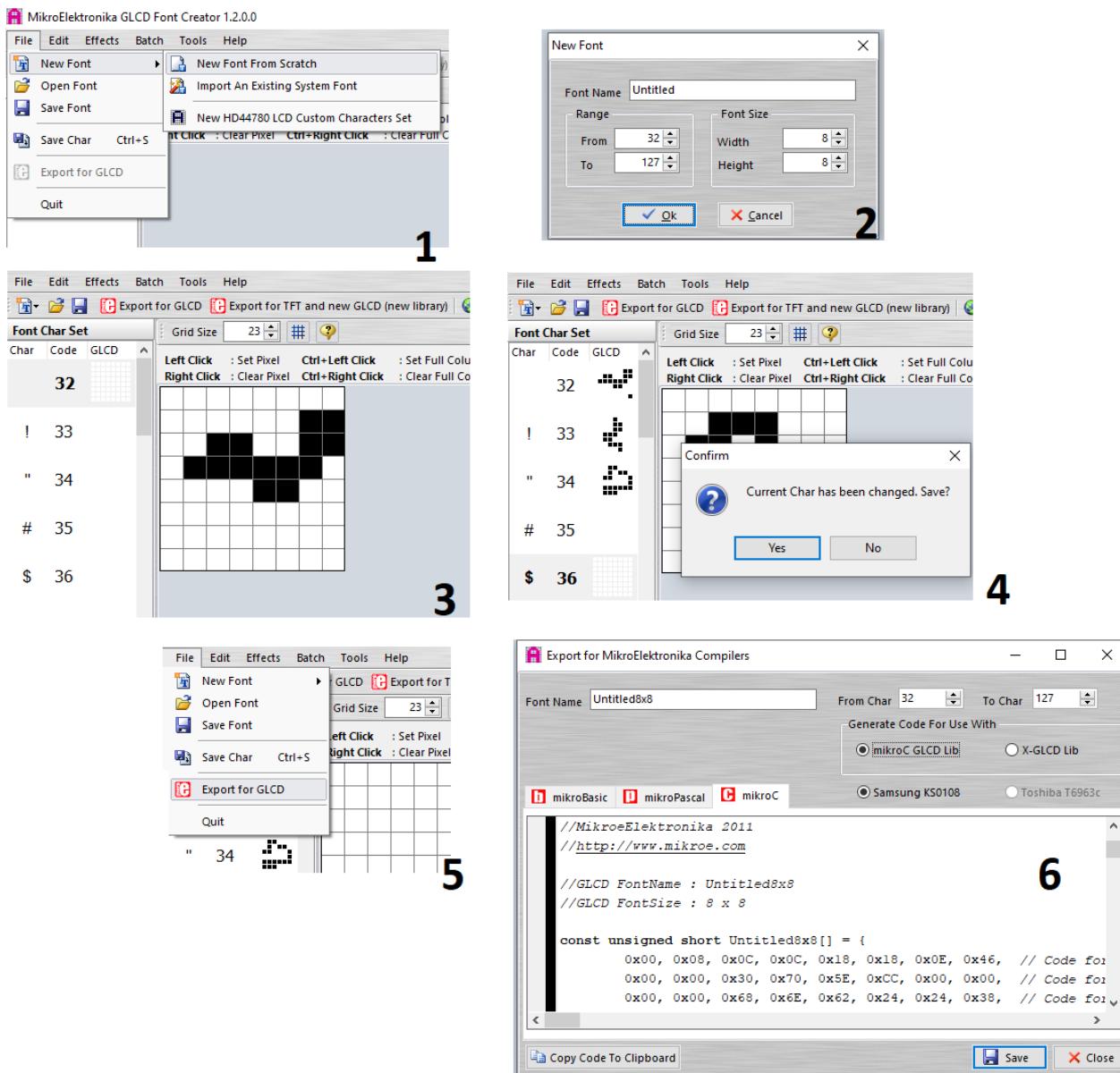


شکل 8-2 - ساختار نمایشگر DotMatrix تعبیه شده بر روی پکیج آزمایشگاه

برای نمایش حروف و علایم روی 64 عدد LED، از روش جاروب کردن سطر و ستون استفاده می‌گردد، به این صورت که ابتدا سطر اول برابر با یک و مابقی سطرهای صفر می‌گردد. سپس مقادیر ستون‌ها با توجه به وضعیت خاموش یا روشن بودن هر LED در این سطر تنظیم می‌شود. بعد از تاخیر زمانی مشخص، سطر دوم برابر با یک و مابقی سطرهای صفر شده و مقادیر مربوط به ردیف دوم بر روی 8 پایه متناظر با ستون‌ها قرار می‌گیرد. این روند تا سطر هشتم ادامه می‌یابد. (عمل جاروب کردن را می‌توان با شیفت دادن صفر منطقی روی ستون‌ها و قرار دادن مقادیر روی ردیف‌ها نیز انجام داد). برای این که طرح مورد نظر بر روی Dot-Matrix روشن بماند، پروسه ذکر شده با تاخیرهای در حد چند میلی ثانیه به دفعات اجرا می‌گردد و با توجه به خطای دید در چشم انسان، طرح مورد نظر بر روی کل 64 عدد LED در هر لحظه مشاهده می‌گردد. مشابه این روش در LCD‌های کاراکتری و برای ایجاد کاراکترهای خاص (مانند علایم و حروف فارسی) نیز استفاده می‌گردد.

### 8.2.1 ایجاد قلم برای نمایش متون روی Dot-Matrix

برای ایجاد قلم جهت نمایش نشانه‌های نوشتاری مختلف می‌توان از نرمافزارهای جانبی مانند GLCD Font Creator استفاده نمود. مراحل مختلف کار با این نرمافزار در شکل 8-3 نشان داده شده است.



- 1: قلم جدید ایجاد می‌گردد.  
 2: می‌توان تعداد کاراکترها را محدود کرد.  
 3: می‌توان قلم مورد نظر را طراحی نمود.  
 4: با کلیک روی آرایه بعدی (مانند کد 33)، پنجره ذخیره نشان داده می‌شود.  
 5: کدهای منتظر با قلم‌ها ایجاد می‌گردد.  
 6: می‌توان تمام کدها را انتخاب و پس از copy کردن در یک فایل txt ذخیره نمود و یا از save استفاده کرد. (در برخی از موارد به سبب اشکالات برنامه، گزینه Save فایلی را ذخیره نمی‌کند).

شکل 3-8: مراحل کار با GLCD Font Creator

## 8.2.2 اصول کلی نمایش متن بر روی DotMatrix

داده‌های خروجی حاصل از نرم‌افزار ایجاد قلم برای هر نشانه را به عنوان مقادیر مربوط به ستون‌های Dot-Matrix مطابق برنامه 8-1 و مقادیر مربوط به روند جاروب کردن سطرها را مطابق برنامه 8-2، در ساختارهای آرایه‌ای تعریف و ذخیره می‌نماییم.

```
1-8 برنامه char alphabet [] [8] = {
    { 0x06, 0x29, 0x29, 0x29, 0x1F, 0x00, 0x00, 0x00 }, // a Locate in array = 0
    { 0xFF, 0x09, 0x11, 0x11, 0x0E, 0x00, 0x00, 0x00 }, // b Locate in array = 1
    { 0x1E, 0x21, 0x21, 0x21, 0x12, 0x00, 0x00, 0x00 }, // c Locate in array = 2
    { 0x7F, 0x88, 0x88, 0x88, 0x7F, 0x00, 0x00, 0x00 }, // A Locate in array = 3
    { 0xFF, 0x91, 0x91, 0x91, 0x6E, 0x00, 0x00, 0x00 }, // B Locate in array = 4
    { 0x7E, 0x81, 0x81, 0x81, 0x42, 0x00, 0x00, 0x00 }, // C Locate in array = 5
};

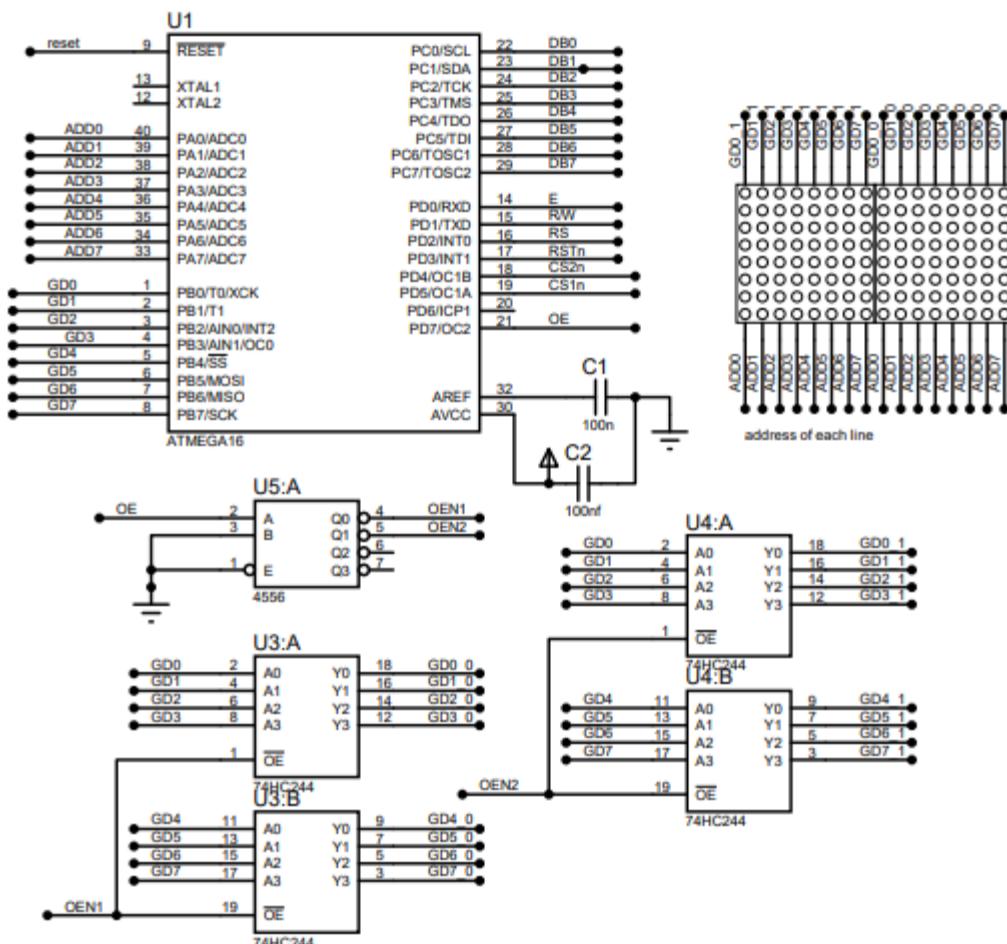
2-8 برنامه unsigned char R_data[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
```

در ادامه یک ساختار switch/case مطابق برنامه 3-8 پیاده‌سازی می‌گردد که در آن به ازای هر نشانه، یک case در نظر گرفته می‌شود. با اجرای این ساختار، موقعیت داده متناظر با حرف مورد نظر در متغیر locat ذخیره می‌شود. به عنوان مثال با در نظر گرفتن برنامه 8-1، برای حرف اول (a) مقدار صفر را برمی‌گرداند، برای حرف دوم (b) مقدار 1 و همین طور برای مقادیر دیگر موقعیت را برمی‌گرداند.

```
3-8 برنامه void lookup(char input)
{
    switch (input)
    {
        case 'a' :
            locat=0;
            break;
        case 'b' :
            locat=1;
            break;
        case 'c' :
            locat=2;
            break;
        case 'A' :
            locat=3;
            break;
        case 'B' :
            locat=4;
            break;
        case 'C' :
            locat=5;
            break;
    }
}
```

### 8.2.3 استفاده از چند مازول DotMatrix

با توجه به این که هر مازول به دو درگاه جهت ساماندهی ستون‌ها و ردیف‌ها نیاز دارد، در صورت استفاده بیش از یک مازول، می‌توان سطرهای همه مازول‌ها را به یکی از درگاه‌ها متصل کرد و برای ردیف‌ها از تعدادی-D-Flip flop استفاده نمود. نمایی از این سخت‌افزار در شکل ۸-۴ نشان داده شده است. در این سخت‌افزار از درگاه A برای سطرهای و از درگاه B به صورت مشترک برای ستون‌ها استفاده شده است. در این سخت‌افزار با فعال نمودن پایه OE، داده‌های ارسالی از سوی ریزپردازنده در حافظه خارجی ثبت می‌شود و با غیرفعال شدن این پایه، این مقادیر ثابت باقی خواهند ماند. بدین ترتیب در بازه‌های زمانی مناسب می‌توان داده‌های هر ستون مازول DotMatrix را به صورت مجزا مقدار دهی کرد. برای مدیریت سیگنال‌های OE، می‌توان از یک انکدر استفاده نمود تا بتوان با استفاده از تعداد پایه‌های کمتری از ریزپردازنده، سیگنال‌های OE بیشتری را کنترل نمود.



شکل ۸-۴: راهاندازی چند مازول DotMatrix به صورت همزمان

## LCD گرافیکی 8.2.4

اساس کار GLCD<sup>۱</sup> مشابه DotMatrix است ولی واحدهای نمایشگر نورانی در آن بسیار کوچکتر می‌باشد. بنابراین برای نمایش داده‌ها روی آن نیز از روش جاروب کردن سطر استفاده می‌شود، بدین صورت که در هر مرحله تنها یکی از سطرهای یک و مابقی صفر می‌شوند و در این لحظه، داده‌های مربوط به آن سطر نوشته می‌شوند. با توجه به تراکم بالای نقاط نورانی در GLCD، برای تعیین زمان‌بندی روشن و خاموش شدن هر سطر و ستون نیاز به یک راهانداز<sup>۲</sup> می‌باشد.

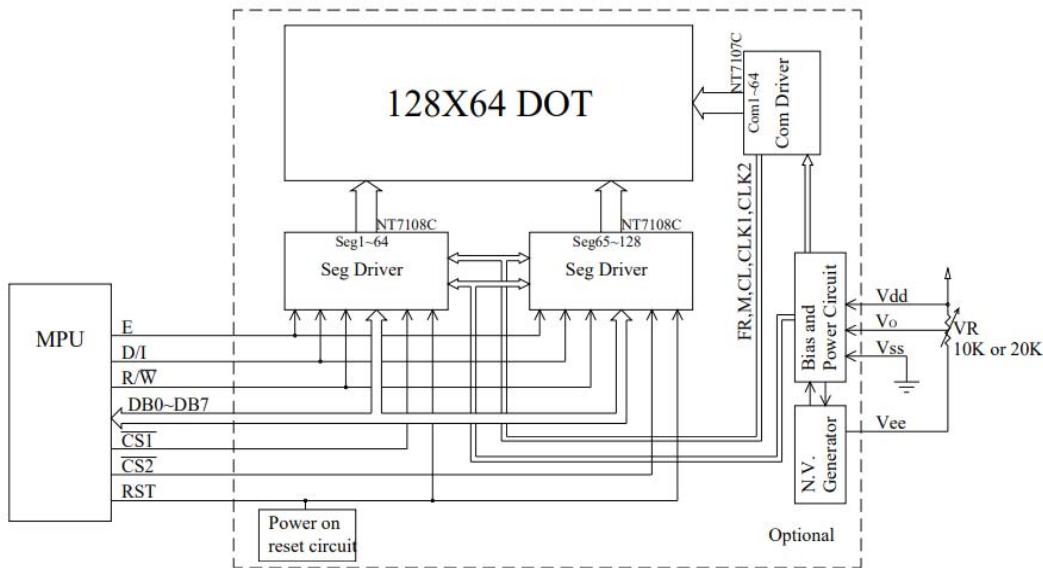
برای راهاندازی ستون‌های GLCD، از تراشه‌ی NT7108C استفاده می‌شود. این تراشه، شامل یک RAM، یک لج 64 بیتی و یک دیکدر می‌باشد. حافظه RAM برای ذخیره داده‌هایی که از طریق ریزپردازنده منتقل می‌شود، به کار می‌رود.

تراشه NT7107C، نیز یک راهانداز 64 کاناله است که در فاصله‌های زمانی مشخص، هر یک از سطرهای GLCD را فعال می‌نماید تا داده‌های ذخیره شده درون NT7108C بر روی GLCD نمایش داده شود. در حقیقت، این تراشه عملیات جاروب کردن سطرهای را انجام می‌دهد.

GLCD مورد استفاده در بورد آزمایشگاه دارای ابعاد 64\*128 پیکسل می‌باشد و به همین دلیل مطابق شکل 5-8 از دو تراشه NT7108C و یک تراشه NT7107C برای راهاندازی نمودن 128 ستون و 64 سطر آن استفاده می‌شود. این تراشه‌ها با تراشه‌های KS0107B و KS0108B از شرکت سامسونگ سازگار هستند و می‌توان در تنظیمات آن‌ها را انتخاب نمود. CodeVision

<sup>1</sup> Graphical LCD

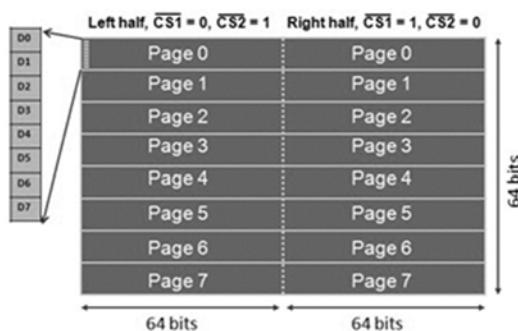
<sup>2</sup> Drive



شکل ۵-۸: مدار راهانداز GLCD در پکیج آزمایشگاهی

تراشه‌ی NT7107C ، 64 سطرنمایشگر را (COM1 – COM64) را راهاندازی می‌کند. در هر لحظه از زمان، تنها یکی از خطوط COM فعال می‌باشد. اولین تراشه NT7108C (IC1) سگمنت‌های نیمه سمت چپ را فعال می‌کند و دومین تراشه NT7108C (IC2) سگمنت‌های نیمه سمت راست را فعال می‌نماید. با استفاده از پین‌های CS11 و CS22 می‌توان به طور جداگانه به هر یک از نیمه‌های نمایشگر دست یافت. هر کدام از نیمه‌ها مانند شکل ۶-۸ شامل 8 صفحه افقی با شماره‌های ۰-۷ می‌باشد که هر کدام از 64 بایت تشکیل شده است.

مقداردهی از صفحه صفر شروع می‌شود. اگر یک بایت داده به GLCD انتقال داده شود، در اولین ستون صفحه صفر نمایش داده می‌شود (شکل ۶-۸). اگر این کار برای 63 بایت داده تکرار شود و سپس با فعال‌سازی نیمه دوم، برای 64 داده دیگر ادامه یابد، در واقع 8x128 پیکسل از GLCD که همگی در صفحه صفر قرار دارند، مقداردهی خواهند شد. این کار برای سایر صفحات نیز به همین صورت تکرار می‌گردد.



شکل ۶-۸

<sup>1</sup> Chip Select 1<sup>2</sup> Chip Select 2

## 8.2.5 آشنایی با پایه‌های GLCD

پایه‌های GLCD مانند جدول 8-1 به دو دسته تقسیم می‌شوند. دسته نخست پایه‌های داده (DB0-DB7) که وظیفه انتقال داده را برعهده دارند و دسته دیگر پایه‌های کنترلی هستند. پایه‌های کنترلی به شرح زیر می‌باشند: E=R/W زمانی که این پایه مقدار یک داشته باشد، داده‌های موجود در [7:0] DB را نمایش می‌دهد و زمانی که E=0 باشد، مقدار آن توسط پردازنده می‌تواند خوانده شود.

CS1=0: زمانی که CS1=1 و CS2=0 باشد، تراشه اول (IC1) NT7108C انتخاب می‌شود و زمانی که CS1=0 و CS2=1 باشد تراشه دوم (IC2) NT7108C انتخاب می‌شود.

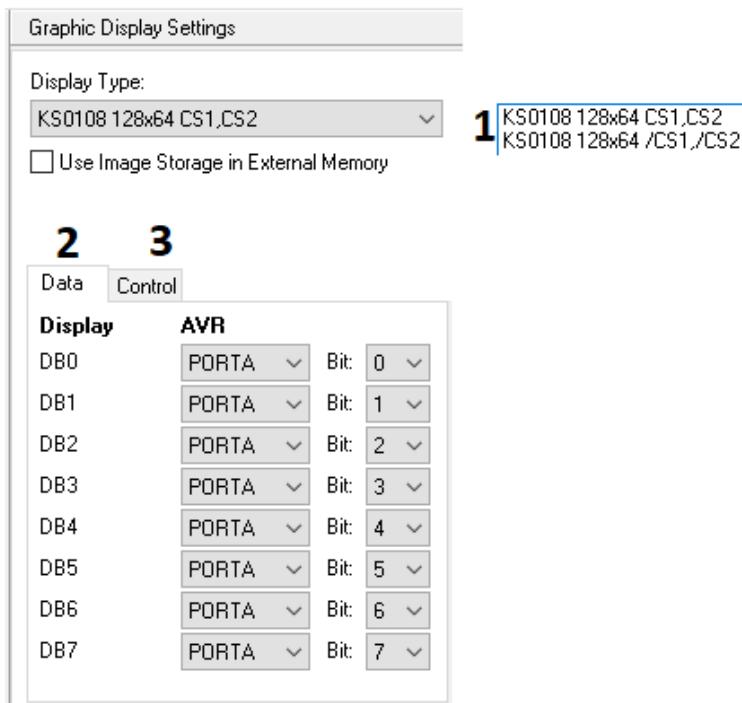
RS=0: اگر RS=0 باشد، GLCD دستورالعمل‌های دریافتی را اجرا می‌کند، در غیر این صورت مقدار خوانده شده از پایه‌های داده را در Data Register قرار می‌دهد.

جدول 8-1: مشخصات پایه‌های GLCD

	Name	Level	Function
1	CS1	L	Chip Select 1(segment 1-64)
2	CS2	L	Chip Select 2(segment 65-128)
3	Vss	0V	Ground
4	Vcc	5.0V	5v Supply in+
5	V0	variable	Contrast Adjust
6	RS or D/I or CD	H/L	H: Data , L: Instruction
7	R/W	H/L	H: Read(MPU← Module) , L :Write(MPU→ Module)
8	E	H	ENABLE SIGNAL
9	DB0	H/L	Data bus line
10	DB1	H/L	Data bus line
11	DB2	H/L	Data bus line
12	DB3	H/L	Data bus line
13	DB4	H/L	Data bus line
14	DB5	H/L	Data bus line
15	DB6	H/L	Data bus line
16	DB7	H/L	Data bus line
17	RST	L	Reset the LCM
18	VEE		Negative Voltage Output
19	A		Power supply for B/L(+)LED BACKLIGHT
20	K		Power supply for B/L(-)LED BACKLIGHT

### 8.3 برنامه نویسی GLCD در محیط کد ویژن

تنظیمات LCD گرافیکی از طریق محیط CodeWizard مانند شکل 8-7 انجام می‌شود. در تنظیمات GLCD لازم است که تراشه کنترل کننده آن مشخص شود. لذا تراشه‌ی KS0108B موجود در شبیه‌ساز که سازگار با NT7108C به کار رفته روی GLCD است، انتخاب می‌شود.



1: انتخاب کنترل کننده مورد نظر (یکی از دو مورد نشان داده شده انتخاب می‌شود).

2: انتخاب سیگنال‌های داده متصل شده به GLCD

3: انتخاب سیگنال‌های کنترلی متصل شده به GLCD

شکل 8-7: تنظیمات کد ویژن برای GLCD

بعد از تنظیمات انجام شده، کدها ذخیره می‌گردند. البته بعد از ذخیره کدها، در صورت نیاز می‌توان از طریق مسیر زیر پایه‌های متصل به GLCD را تغییر داد.

Project>configure>C compiler>Libraries>Graphic display

### 8.4 معرفی فایل سرآیند glcd.h

در کدهای ایجاد شده، فایل سرآیند جدیدی به نام glcd.h فراخوانی می‌شود که حاوی توابع لازم برای کار کردن با GLCD می‌باشد. این توابع شامل تنظیمات اولیه، پاک کردن GLCD، نوشتن متن، انتقال مکان‌نما و یا رسم شکل‌های هندسی و غیره می‌باشد. برای اطلاع از آخرین تغییرات دستورالعمل‌ها بایستی به راهنمای Codevision مراجعه گردد.

## 8.5 نمونه برنامه GLCD

برای آشنایی بیشتر با GLCD تنظیمات اولیه در محیط CodeWizard انجام شده و برخی دستورات به برنامه اضافه می‌گردد تا متن مورد نظر روی GLCD نشان داده شود. همچنین چند خط به عنوان یک طرح گرافیکی ساده نیز روی GLCD رسم می‌گردد. قطعه کد شکل ۸-۸ چگونگی نمایش یک متن و شکل را بر روی GLCD نشان می‌دهد که البته برای کاهش حجم برنامه، توضیحات هر دستور حذف شده است.

```

1 #include <mega16.h>
2 #include <glcd.h>
3 #include <font5x7.h>
4 #include <delay.h>
5
6 void main(void)
7 {
8     GLCDINIT_t glcd_init_data;          //note 1
9     glcd_init_data.font=font5x7;        //note 2
10    glcd_init_data.readxmem=NULL;       //note 3
11    glcd_init_data.writexmem=NULL;      //note 4
12    glcd_init(&glcd_init_data);         //note 5
13
14    glcd_outtextf("MICROPROCESSOR LAB"); //note 6
15    delay_ms(800);
16    glcd_clear();                      //note 7
17    glcd_outtextf("Some line styles:");
18
19    glcd_setlinestyle(1,GLCD_LINE_DOT_SMALL); //note 8
20    glcd_line(0,10,127,10);                //note 9
21    glcd_setlinestyle(1,GLCD_LINE_DOT_LARGE); //note 10
22    glcd_line(0,20,127,20);
23    while(1);
24 }

```

شکل ۸-۸: نمونه کد برای GLCD

۵: پیکربندی GLCD با تنظیمات مشخص شده در ساختار

نکات:

۱: تعریف ساختار

۶: نمایش متن روی GLCD

۲: تعیین نوع قلم در ساختار

۷: پاک کردن GLCD

۳: تعیین قابلیت خواندن از حافظه خارجی

۸: تنظیم استایل خط قبل از شروع ترسیم

۴: تعیین قابلیت نوشتمن در حافظه خارجی

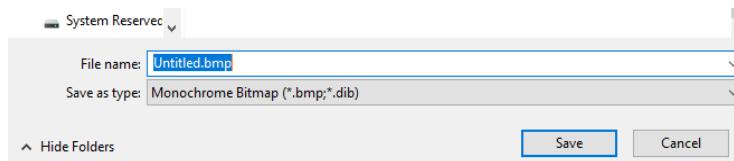
۹: رسم خط

## 8.6 قلم‌های نمایش کاراکتر

برای نمایش حروف بر روی GLCD معمولاً از دو قلم متفاوت با ابعاد ۵x8 و ۸x8 استفاده می‌شود. قلم‌های فارسی معمولاً از نوع ۸x8 و قلم‌های انگلیسی از نوع ۵x8 می‌باشند. فایل سرآیند font5x7.h برای قلم انگلیسی قابل استفاده است. برای ایجاد قلم‌های جدید هم می‌توان از نرم‌افزارهای طراحی قلم مشابه قسمت DotMatrix استفاده نمود.

## 8.7 نمایش تصویر روی GLCD

با توجه به این که GLCD تعییه شده در پکیج آزمایشگاه، سیاه و سفید است، صرفاً تصاویر محدودی را می‌توان بر روی آن نشان داد. این تصاویر باید دارای قالب monochrome باشند که می‌توان عکس مورد نظر را در محیط paint و مطابق شکل 8-9 در این قالب ذخیره نمود. همچنین برای جلوگیری از افت کیفیت بهتر است اندازه عکس  $128 \times 64$  پیکسل باشد.

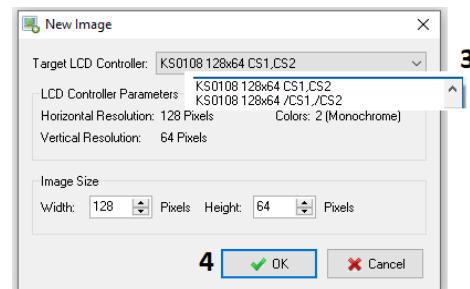


شکل 8-9: ذخیره تصویر در محیط Paint

### 8.7.1 تهیه کدهای عکس با استفاده از نرمافزار LCD Vision

برای ذخیره تصاویر در حافظه پردازنده و به منظور نمایش بر روی GLCD باید آن‌ها را به صورت آرایه دو بعدی از داده‌های باینری در آورد. البته جهت فشرده‌سازی از کد مبنای 16 به جای باینری استفاده می‌شود. به این منظور پس از آماده‌سازی اولیه عکس مورد نظر، می‌توان در محیط LCD Vision کدهای مبنای 16 تصاویر را تهیه نمود که مراحل آن در شکل 8-10 قابل مشاهده است.

- 1 نرم افزار LCD Vision را باز نمایید.
- 2 از منوی file گزینه new Image را بزنید.
- 3 در پنجره باز شده کنترلر مربوطه را انتخاب نمایید.
- 4 گزینه ok را بزنید.



- 5 در پنجره باز شده، تصویر را با استفاده از موس رسم نمایید یا از منوی file گزینه Import File انتخاب نمایید.
- 6 در صورتی که Import File انتخاب شده باشد، در پنجره باز شده تصویر مورد نظر را انتخاب و open نمایید.
- 7 در پنجره باز شده گزینه ok را انتخاب نمایید.



- 8 بخشی از تصویر که قصد دارید روی GLCD نمایش دهید را با موس انتخاب نمایید.
- 9 از منوی File گزینه Export را انتخاب نمایید.
- 10 در پنجره باز شده گزینه ok را انتخاب نمایید.
- 11 کدهای تولید شده، در پنجره Export Preview نشان داده می‌شود.
- 12 از منوی file گزینه Save Export را انتخاب نمایید.
- 13 در پنجره باز شده گزینه ok را انتخاب نمایید.
- 14 مسیر مورد نظر را انتخاب و ذخیره نمایید.
- 15 در پنجره باز شده اعلام می‌نماید که دو عدد فایل با فرمت .h و .c ایجاد شده است.



شکل 8-10: مراحل تهیه کدهای مبنای 16 تصویر در محیط LCD Vision

### 8.7.2 برنامه نمایش عکس در محیط Codevision

بهتر است فایل‌های تهیه شده توسط LCD Vision در مسیر پروژه قرار گیرند و سپس در محیط Code vision با استفاده از منوی زیر به فایل‌های پروژه اضافه گردند.

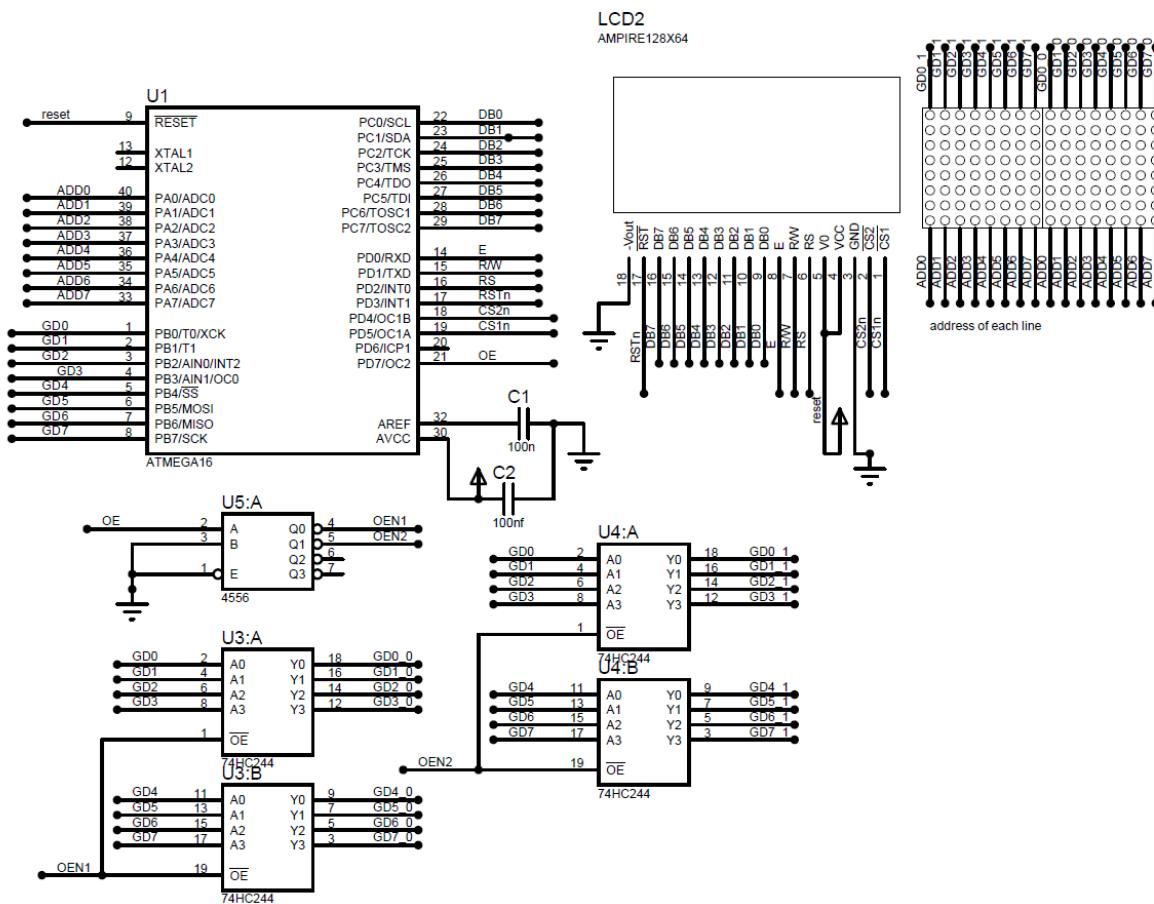
Project > Configure >Files >Input Files >Add

همچنین فایل سرآیند ایجاد شده نیز باید در ابتدای برنامه فراخوانی گردد (#include "picture\_name.h"). لازم به ذکر است که نام آرایه‌ی شامل کدهای تصویر نبایستی با عدد شروع شود. در غیر این صورت خطای کامپایلر را به دنبال خواهد داشت. برای رسم تصویر بر روی GLCD از یکی از توابع موجود در graphic.h به نام glcd\_putimage به صورت زیر استفاده می‌گردد. چنان‌چه کدها در حافظه flash ذخیره شده باشد، از تابع جایگزین glcd\_putimagef استفاده می‌گردد.

```
glcd_putimage (0, 0, picture_name, GLCD_PUTCOPY);
glcd_putimagef (0, 0, picture_name, GLCD_PUTCOPY);
```

### 8.8 برنامه‌های اجرایی مبحث GLCD

سیستم طراحی شده شکل 8-11 را در نظر بگیرید که حاوی دو مجموعه DotMatrix می‌باشد و هر کدام برای فعال شدن به دو درگاه (جستا 32 پایه) نیاز دارد. اما از آن جا که در این سختافزار از GLCD نیز استفاده گردید باقیستی تعداد درگاه‌های مورد نیاز برای راه اندازی DotMatrix بهینه گردد. لذا از یک درگاه به طور مشترک برای آدرس‌دهی هر ردیف استفاده شده است و با استفاده از بافرهای 74HC244 می‌توان داده‌های هر DotMatrix را ذخیره نمود و در فاصله زمانی مشخصی آن را به روز نمود. برای تهیه سیگنال OEn بافر نیز می‌توان از یک دیکدر استفاده کرد تا بتوان چندین DotMatrix را درایو نمود. بدین ترتیب فقط 17 پایه از ریزپردازنده برای DotMatrix اشغال می‌شود.



شکل ۸-۱۱: نمایی از سختافزار مبحث DotMatrix و GLCD

۱. زیربرنامه‌ای بنویسید که حرف اول نام و نام خانوادگی شما را به صورت متن روان روی DotMatrix نشان

دهد.

۲. زیربرنامه‌ای بنویسید که یک تصویر دلخواه را روی LCD گرافیکی نمایش دهد.

۳. یک ساعت آنالوگ دارای عقربه‌های ساعت شمار، دقیقه شمار و ثانیه شمار طراحی و روی LCD گرافیکی نمایش دهید. (راهنمایی: برای رسم ساعت آنالوگ می‌توان با استفاده از روابط مثلثاتی، مختصات هریک از عقربه‌ها را محاسبه کرد و با تغییر هر کدام، خط جدیدی به مبدا ساعت و نقطه مورد نظر رسم نمود).

۴. بندهای فوق را در قالب یک پروژه در بیاورید به گونه‌ای که ابتدا تصویر به مدت سه ثانیه نمایش داده شود، سپس متن روان روی DotMatrix و نهایتاً ساعت روان روی LCD گرافیکی نمایش داده شود.

## 9 جلسه نهم

### SPI ارتباط سریال

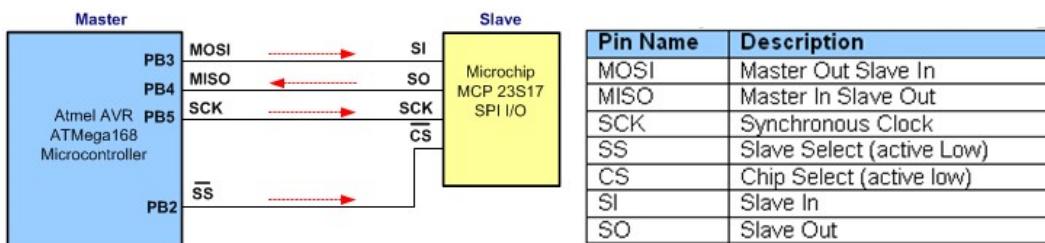
#### 9.1 هدف

در این جلسه نحوه برقراری ارتباط سریال SPI بررسی می‌گردد.

#### 9.2 مقدمه

ارتباط سریال SPI مانند شکل 9-1 یک پروتکل ارتباط سریال سنکرون با سرعت بالا بوده که می‌تواند برای برقراری ارتباط بین ریزپردازندۀ‌های AVR و وسایل جانبی متفاوت به کار رود. ویژگی‌های اصلی این ارتباط به صورت زیر می‌باشد:

- ارسال داده به صورت سنکرون دو طرفه (Full-Duplex)
- حالت‌های کاری master و slave
- پرچم پایان ارسال و فعال شدن وقفه
- امکان دو برابر کردن سرعت ارسال



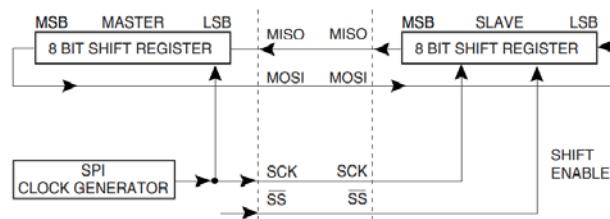
Typical SPI Master and SPI Slave Device Connection

شکل 9-1: نمایی از ارتباط SPI

ارتباط SPI از سمت دستگاه Master شروع و به سمت دستگاه Slave ختم می‌گردد. دستگاه Master وظیفه تولید پالس ساعت (SCK) و انتخاب دستگاه مورد نظر Slave را با استفاده از پایه SS بر عهده دارد. دو پایه MOSI و MISO هم برای انتقال داده در دو جهت مختلف استفاده می‌گردد. بعد از انتقال کامل داده توسط MASTER، پالس ساعت SPI قطع، پرچم وقفه پایان ارسال داده (SPIF) برابر با یک شده و برنامه وقفه اجرا می‌گردد.

اساس کار SPI مانند شکل 9-2 بر پایه دو ثبات می‌باشد که پس از برقراری اتصال بین دو دستگاه، با هر پالس ساعت، یک بیت از ثبات فرستنده خارج شده و به ثبات گیرنده وارد می‌شود. لذا دو ثبات 8 بیتی در MASTER و SLAVE را می‌توان به عنوان یک ثبات چرخشی 16 بیتی در نظر گرفت. زمانی که داده‌ای از MASTER به

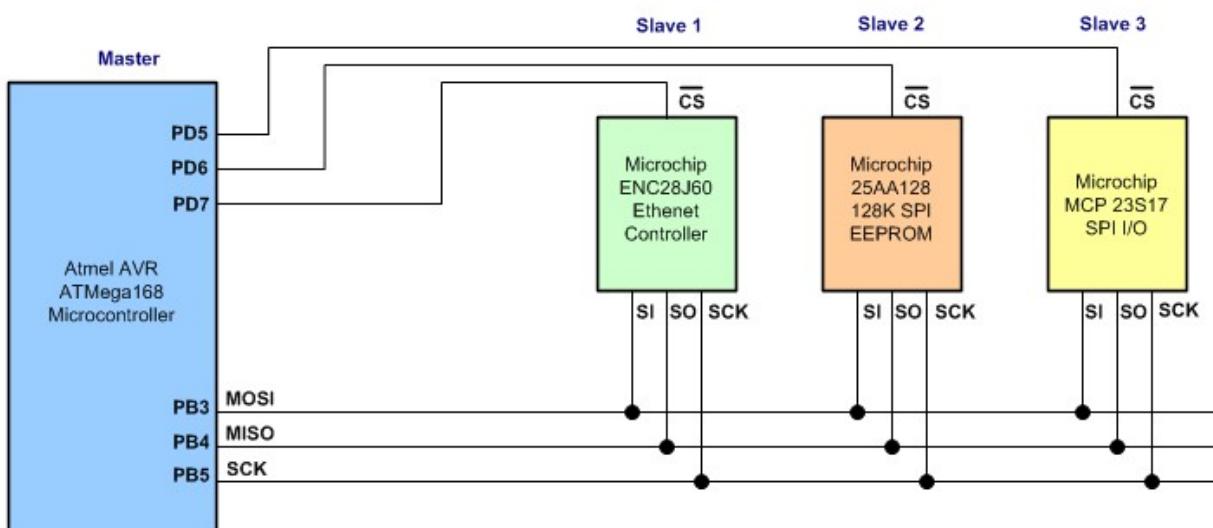
ارسال می‌شود، در همان حال و در جهت مخالف، داده‌ای از MASTER به SLAVE انتقال می‌یابد. به این صورت در طول هشت پالس ساعت SPI، داده‌های MASTER و SLAVE به صورت کامل با یکدیگر عوض می‌شوند.



شکل ۹-۲: نحوه انتقال اطلاعات در ارتباط SPI

بر همین اساس ارتباط SPI یک ارتباط Full Duplex محسوب می‌گردد که به صورت همزمان توانایی ارسال و دریافت داده را دارد. زمانی که بخواهد از SLAVE داده دریافت کند، SLAVE باید یک داده بر روی SCK و SS را دریافت خواهد کرد.

همچنین مانند شکل ۹-۳ امکان اتصال چندین ماژول در یک ارتباط SPI به طور همزمان وجود دارد که برای هر کدام می‌توان از یک خط SS<sub>n</sub> جداگانه استفاده کرد.



Typical SPI Master with Multiple SPI Slave Device Connection

شکل ۹-۳: اتصال چند ماژول به SPI

### SPI ثبات‌های 9.3

واحد SPI دارای سه ثبات می‌باشد که در ادامه شرح داده شده‌اند.

### SPDR (SPI Data Register) 9.3.1 ثبات

نوشتن در این ثبات پروسه‌ی ارسال داده را فعال می‌نماید. همچنین بعد از اتمام ارسال، محتوای این ثبات در برگیرنده محتوای ثبات گیرنده طی فرآیند شیفت داده‌ها خواهد بود.

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	SPDR							
Initial Value	X	X	X	X	X	X	X	X	Undefined

شکل 9-4: ساختار ثبات SPDR

### SPSR (SPI Status Register) 9.3.2 ثبات

این ثبات که در شکل 9-5 نشان داده شده است، وضعیت ارتباط SPI را نشان می‌دهد.

Bit	7	6	5	4	3	2	1	0	
Read/Write	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Initial Value	R	R	R	R	R	R	R	R/W	0

شکل 9-5: ساختار ثبات SPSR

بیت 7: SPIF (SPI Interrupt Flag): وقتی پروسه‌ی تبادل داده تمام شد، در صورتی که وقفه‌ی SPI فعال شده باشد، این بیت برابر با یک می‌شود و پس از اجرای زیربرنامه وقفه یا خواندن ثبات داده، مقدار آن صفر می‌گردد.

بیت 6: WCOL (Write COLision flag): اگر در طول پروسه تبادل داده، در ثبات داده مقدار جدیدی نوشته شود این بیت یک می‌گردد. پس از خواندن ثبات داده یا ثبات SPSR هم مقدار آن صفر می‌گردد.

بیت صفر (Double SPI Speed Bit): با نوشتن یک در این بیت، فرکانس پالس ساعت در حالت Master دو برابر می‌شود.

### SPCR (SPI Control Register) 9.3.3 ثبات

این ثبات که در شکل 9-6 نشان داده شده است، کنترل ارتباط SPI را بر عهده دارد.

Bit	7	6	5	4	3	2	1	0	SPCR
ReadWrite	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	
Initial Value	0	0	0	0	0	0	0	0	

شکل 9-6: ساختار ثبات SPCR

بیت 7: SPIE (SPI Interrupt Enable) و قوهی SPI را فعال می‌کند.

بیت 6: SPE (SPI Enable) واحد SPI را فعال می‌نماید.

بیت 5: DORD (Data Order) اگر یک باشد بیت LSB و اگر صفر باشد بیت MSB از ثبات داده ارسال می‌گردد.

بیت 4: MSTR (Master/Slave Select) نوشتمن یک در این بیت حالت عملکرد Master و نوشتمن صفر در آن حالت Slave را فعال می‌کند. اگر پایه SSn در حالت Master به ورودی تبدیل شود و سطح منطقی صفر به آن اعمال گردد، سیستم از حالت Master خارج شده و SPIF در SPSR برابر با یک می‌شود.

بیت 3: CPOL (Clock Polarity) اگر یک باشد پالس ساعت در حالت بیکاری سطح یک را دارد و در غیر این صورت دارای سطح صفر است.

بیت 2: CPHA (Clock Phase) اگر صفر باشد نمونه‌برداری در لبه‌ی بالارونده و در غیر این صورت نمونه‌برداری در لبه‌ی پایین رونده پالس ساعت رخ می‌دهد.

بیتهای 1 و صفر SPR1, SPR0 (SPI Clock Rate Select 1 and 0): پالس ساعت Master را به عنوان ضریبی از پالس ساعت ریزپردازنده انتخاب می‌نماید.

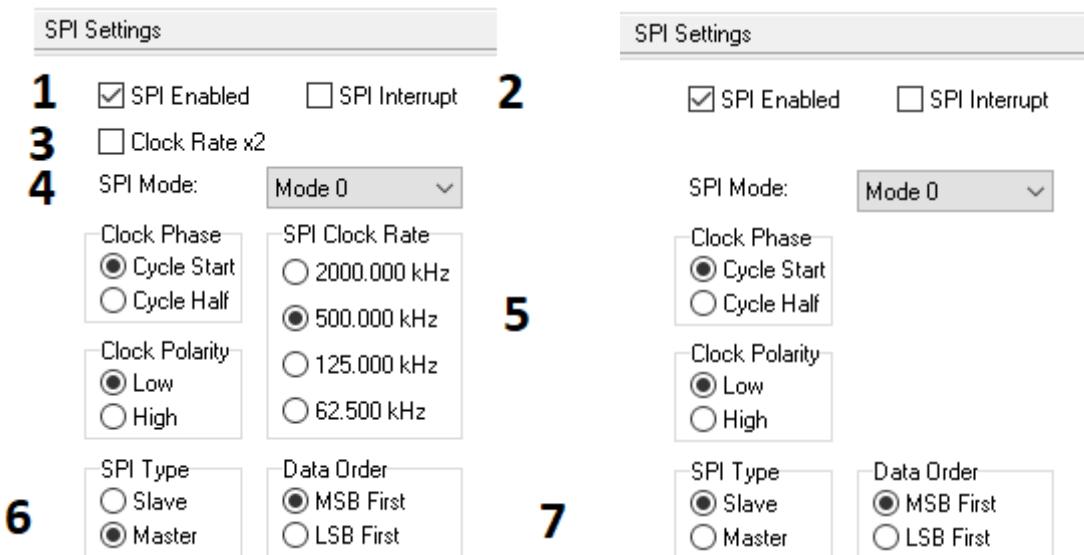
#### 9.4 پیکربندی SPI در Codevision

برای پیکربندی واحد SPI مطابق شکل 9-7 از CodeWizard استفاده می‌شود. دو دستگاه فرستنده و گیرنده در دو حالت کاری متفاوت Master و Slave پیکربندی می‌گردد. اما سایر پارامترها شامل مد کاری، تعیین نمونه‌برداری در لبه بالا رونده و یا پایین رونده پالس ساعت، تقدم سطح صفر یا یک در پالس ساعت و ترتیب داده‌ها یکسان در نظر گرفته می‌شوند.

File> new project>

via code wizard

select: Serial Peripheral Interface



2: فعال شدن وقفه

4: حالت SPI انتخاب می‌شود. با انتخاب لبه نمونه برداری و تقدم سطح پالس ساعت، وضعیت

SPI نیز تنظیم می‌شود.

5: انتخاب ترتیب داده در ارسال

SPI

1: فعال شدن SPI

3: دو برابر شدن پالس ساعت

5: انتخاب پالس ساعت در حالت Master

6: انتخاب Master/Slave

### همچنین باستی وضعیت پایه‌های ورودی و خروجی نیز طبق جدول 9-1 تعیین گردد.

جدول 9-1 : پیکربندی پایه‌های

Pin	Master SPI(I/O)	Slave SPI(I/O)
MOSI	Output	Input
MISO	Input	Output
SCK	Output	Input
SS	Output	Input

### 9.5 SPI سناریوهای مختلف ارتباط

پیش از معرفی سناریوهای متفاوت، ابتدا عملکرد ریزپردازنده در یک ارتباط SPI شرح داده می‌شود.

در حالت Master

داده را در ثبات SPDR می‌نویسد و بلافاصله ارسال داده شروع می‌شود. •

- طی 8 پالس ساعت و با تکرار فرایند شیفت، داده برای Slave ارسال می‌گردد. پس از آن SCK متوقف و پرچم SPIF یک می‌گردد.

#### در حالت Slave

- تا وقتی که سیگنال SS<sub>n</sub> دارای سطح یک است، Slave در حالت IDLE می‌باشد.
- با تغییر SS<sub>n</sub> به مقدار صفر، Slave فعال می‌شود و داده‌های موجود در ثبات SPDR با هر پالس ساعت دریافتی از Master شیفت پیدا می‌کند.
- وقتی یک بایت کامل شیفت داده شد، پرچم SPIF یک می‌شود.

#### عملکرد پایه‌ی SS<sub>n</sub> در حالت Master

- در این حالت، پایه‌ی SS<sub>n</sub> به عنوان پایه I/O در نظر گرفته می‌شود.
- هنگامی که Master بخواهد Slave را فعال کند این پایه به عنوان خروجی در نظر گرفته می‌شود.
- اگر در حالت Master این پایه ورودی باشد، بایستی در سطح یک منطقی قرار گیرد.
- اگر در حالت Master این پایه ورودی باشد و توسط مدار خارجی به سطح صفر تغییر کند، SPI متوجه می‌شود که یک Master دیگر گذرگاه SPI را در اختیار گرفته و قصد دارد با این ریزپردازنده ارتباط برقرار نماید.
- در این حالت بیت MSTR در ثبات SPCR صفر می‌گردد و ریزپردازنده به حالت Slave تغییر می‌یابد.

#### عملکرد پایه‌ی SS<sub>n</sub> در حالت Slave

- در حالت Slave، این پایه همواره به عنوان ورودی در نظر گرفته می‌شود.
- وقتی به سطح صفر تغییر می‌کند SPI فعال می‌شود.
- وقتی به سطح یک تغییر می‌کند SPI بازنشانی (Reset) می‌گردد و دیگر پیامی دریافت نمی‌کند.

با توجه به نوع مژول‌های شرکت یافته در ارتباط، سناریوهای مختلفی را می‌توان برای یک ارتباط SPI در نظر گرفت. مهم‌ترین این سناریوها شامل ارتباط دو ریزپردازنده و یا ارتباط ریزپردازنده با مژول‌های دیگر مانند حافظه خارجی، مدل‌های A/D و غیره می‌باشد.

در ارتباط دو ریزپردازنده از طریق SPI که عموماً حالت ساده‌تری می‌باشد، می‌توان برنامه‌های دو سمت ارتباط را به دلخواه توسعه داد و تنها کافی است زمان‌بندی دو سمت با یکدیگر سازگاری داشته باشند. یعنی زمانی که یک طرف قصد ارسال داده دارد، طرف دیگر آمادگی پذیرش آن را داشته باشد.

اما در ارتباط ریزپردازنده با ماژول‌های دیگر که در حالت Slave فعال خواهد شد، لازم است برگه‌های راهنمای قطعه مورد نظر به دقت بررسی گردد، زیرا معمولاً هر ماژول ترتیب خاصی را برای ارسال و دریافت در نظر می‌گیرد و در جزئیاتی مانند فضای آدرس دهی، تعداد بایت‌ها و غیره منحصر به فرد خواهد بود. به عنوان مثال در ارتباط با حافظه M950x، قبل از هر گونه نوشتن، غیر فعال کردن نوشتن، خواندن ثبات وضعیت حافظه، خواندن ثبات وضعیت نوشتن، خواندن و نوشتن در حافظه، خواندن و نوشتن در صفحه‌ای خاص از حافظه لازم است که همه این موارد باید در قالب خاصی صورت پذیرد. بناراین لازمه‌ی برقراری ارتباط صحیح، بررسی دقیق برگه‌های راهنمای تراشه‌ی حافظه می‌باشد.

## 9.6 برنامه‌های کاربردی SPI

اگر وقفه فعال نباشد می‌توان از توابع موجود در فایل سرآیند spi.h استفاده نمود که نمونه آن برای دستگاه Master در برنامه 9-1 و برای دستگاه Slave در برنامه 9-2 نشان داده شده است.

```
// Master Program

#include <mega16.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
#include <spi.h>
void main(void)
{
    char count=0,data=0;
    char buffer[5];

DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2) |
(0<<DDA1) | (0<<DDA0) ;
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) |
(0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0) ;

DDRB=(1<<DDB7) | (0<<DDB6) | (1<<DDB5) | (1<<DDB4) | (0<<DDB3) | (0<<DDB2) |
(0<<DDB1) | (0<<DDB0) ;
// State: Bit7=0 Bit6=T Bit5=0 Bit4=0 Bit3=T Bit2=T Bit1=T Bit0=0
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) |
(0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0) ;

DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) |
(0<<DDC1) | (0<<DDC0) ;
```

برنامه 9-1

```

// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) |
(0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) |
(0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) |
(0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 2000.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=(0<<SPIE) | (1<<SPE) | (0<<DORD) | (1<<MSTR) | (0<<CPOL) | (0<<CPHA) |
(0<<SPR1) | (0<<SPR0);
SPSR=(0<<SPI2X);

lcd_init(16);
lcd_gotoxy(0,0);
lcd_puts("M Send: ");
lcd_gotoxy(0,1);
lcd_puts("M recieve: ");

while (1)
{
    data=spi(count); //count: sending data: recieve

    sprintf(buffer, "%d    ", count);
    lcd_gotoxy(10,0);
    lcd_puts(buffer);
    count++;

    sprintf(buffer, "%d    ", data);
    lcd_gotoxy(10,1);
    lcd_puts(buffer);

    delay_ms(500);
}
} ****
//Slave Program

#include <mega16.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
#include <spi.h>

void main(void)
{

```

```

char count=0,data=0;
char buffer[5];

DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2) |
(0<<DDA1) | (0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) |
(0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

DDRB=(0<<DDB7) | (1<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) |
(0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=0 Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) |
(0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) |
(0<<DDC1) | (0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) |
(0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) |
(0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) |
(0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// SPI initialization
// SPI Type: Slave
// SPI Clock Rate: 2000.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=(0<<SPIE) | (1<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA) |
(0<<SPR1) | (0<<SPR0);
SPSR=(0<<SPI2X);

lcd_init(16);
lcd_gotoxy(0,0);
lcd_puts("S Send: ");
lcd_gotoxy(0,1);
lcd_puts("S recieve: ");

while (1)
{
    data=spi(count); //count: sending data: recieve

    sprintf(buffer, "%d    ", count);
    lcd_gotoxy(10,0);
    lcd_puts(buffer);
    count=count+2;

    sprintf(buffer, "%d    ", data);
    lcd_gotoxy(10,1);
    lcd_puts(buffer);
}

```

```

        delay_ms(500) ;
    }
}

```

در برنامه‌های فوق با فراخوانی تابع `(spi, می‌توان یک بایت را از طریق سرکشی بر روی گذرگاه SPI ارسال نمود و به صورت همزمان بایت دیگری را دریافت کرد. ولی نکته مهم در ارسال داده، تشخیص ابتدا و انتهای بسته‌ی داده است که چگونه آن‌ها را بفرستد و پردازش نماید. یک روش مطمئن برای انجام این کار، تبدیل داده‌ها به کاراکترهای معادل و ارسال رشته‌های کاراکتری روی گذرگاه SPI می‌باشد. برای ارسال و دریافت کاراکتر، دو تابع (putchar() و getchar() مطابق برنامه ۳-۹ مجدد تعریف شده‌اند. با این کار، از آن جایی که توابع printf و puts و putchar از این توابع استفاده می‌کنند، نحوه عملکرد آن‌ها نیز تغییر می‌یابد.`

```

#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
    spi(c);
}

#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
    return spi(0);
}

#endif

```

در برنامه ۴-۹ و برنامه ۵-۹ نمونه برنامه‌هایی برای بخش SPI با استفاده از تعریف مجدد دستورات `getchar` و `putchar` آمده است.

```

/*master program*/
#include <mega16.h>
#include <spi.h>
#include <delay.h>
#include <stdio.h>
#include <string.h>

#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
    spi(c);
}

```

---

```

        }
#pragma used-

#define _ALTERNATE_GETCHAR_
#pragma used+
    char getchar(void)
{
    return spi(0);
}
#pragma used-

void main(void)
{
char count=0;
char str[20];

DDRC=0x00; //as input
PORTC=0x00;
DDRB=0xB0; // SSn, SCK, MOSI as output
PORTB=0x00;

// SPI initialization: Master
// sck : 500.000 kHz
// SPI Clock Phase: Cycle Start;
//SPI Clock Polarity: Low ;
//SPI Data Order: MSB First

SPCR=(0<<SPIE) | (1<<SPE) | (0<<DORD) | (1<<MSTR) | (0<<CPOL) | (0<<CPHA)
| (0<<SPR1) | (1<<SPR0);
SPSR=(0<<SPI2X);
memset(str, '\0', sizeof str);

while (1)
{
    delay_ms(500);
    sprintf(str,"count=%3d \n",count);
    count=count+5;
    if(count>127)count=0;
    puts(str);

}
}

برنامه 4-9 /* slave program*/
5-9 #include <mega16.h>
#include <spi.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
#include <string.h>

#define _ALTERNATE_PUTCHAR_
#pragma used+

```

```

void putchar(char c)
{
    spi(c);
}
#pragma used-

#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
    return spi(0);
}
#pragma used-

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

    char scr[30];
// Input/Output Ports initialization
// Port A initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2)
| (0<<DDA1) | (0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3)
| (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

// Port B initialization
// Function: Bit7=In Bit6=Out Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRB=(0<<DDB7) | (1<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2)
| (0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=0 Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3)
| (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2)
| (0<<DDC1) | (0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3)
| (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In
Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2)
| (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3)
| (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

```

```

// SPI initialization
// SPI Type: Slave
// SPI Clock Rate: 2000.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=(0<<SPIE) | (1<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA)
| (0<<SPR1) | (0<<SPR0);
SPSR=(0<<SPI2X);

lcd_init(16);

while (1)
{
    lcd_gotoxy(0, 0);
    memset(scr, '\0', sizeof scr);
    gets(scr,30);
    lcd_puts(scr);
}
}

```

در حالتی که از وقفه استفاده می‌گردد می‌توان با نوشتن و یا خواندن ثبات SPDR، داده‌ی مورد نظر را ارسال و یا دریافت نمود. در برنامه 6-9 و برنامه 7-9 به ترتیب نمونه کدهای توسعه داده شده برای دستگاه‌های Master و Slave نشان داده شده است.

```

/* Master program */

#include <mega16.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>

6-9 برنامه

// SPI interrupt service routine
interrupt [SPI_STC] void spi_isr(void)
{
    unsigned char data;
    char buffer2[5];
    data=SPDR;
    sprintf(buffer2, "%3d", data);
    lcd_gotoxy(12,1);
    lcd_puts(buffer2);
}

void main(void)
{
    char count=0;
    char buffer[5];

```

```

DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2)
| (0<<DDA1) | (0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3)
| (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

DDRB=(1<<DDB7) | (0<<DDB6) | (1<<DDB5) | (1<<DDB4) | (0<<DDB3) | (0<<DDB2)
| (0<<DDB1) | (0<<DDB0);
// State: Bit7=0 Bit6=T Bit5=0 Bit4=0 Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3)
| (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2)
| (0<<DDC1) | (0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3)
| (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2)
| (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3)
| (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 2000.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=(1<<SPIE) | (1<<SPE) | (0<<DORD) | (1<<MSTR) | (0<<CPOL) | (0<<CPHA)
| (0<<SPR1) | (0<<SPR0);
SPSR=(0<<SPI2X);

// Clear the SPI interrupt flag
#asm
    in    r30,spsr
    in    r30,spdr
#endasm

lcd_init(16);

lcd_gotoxy(0,0);
lcd_puts("M Send: ");
lcd_gotoxy(0,1);
lcd_puts("M recieve: ");

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

while (1)
{
    SPDR=count;
}

```

```

        sprintf(buffer, "%3d", count);
        lcd_gotoxy(10,0);
        lcd_puts(buffer);
        count++;
        if(count>127)count=0;
        delay_ms(1000);
    }
}

7-9 برنامه /* Slave Program*/
#include <mega16.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>

// SPI interrupt service routine
interrupt [SPI_STC] void spi_isr(void)
{
unsigned char data;
static char count=0;
char buffer[5];

data=SPDR;
SPDR=count;

sprintf(buffer, "%3d",data);
lcd_gotoxy(12,1);
lcd_puts(buffer);

sprintf(buffer, "%3d", count);
lcd_gotoxy(10,0);
lcd_puts(buffer);
count=count+3;
if(count>127)count=0;
}

void main(void)
{

DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2)
| (0<<DDA1) | (0<<DDA0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3)
| (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);

DDRB=(0<<DDB7) | (1<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2)
| (0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=0 Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T

```

```

PORTB= (0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3)
| (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0) ;

DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2)
| (0<<DDC1) | (0<<DDC0) ;
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3)
| (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0) ;

DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2)
| (0<<DDD1) | (0<<DDD0) ;
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3)
| (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0) ;

// SPI initialization
// SPI Type: Slave
// SPI Clock Rate: 2000.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=(1<<SPIE) | (1<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA)
| (0<<SPR1) | (0<<SPR0) ;
SPSR=(0<<SPI2X) ;
SPDR=0;
// Clear the SPI interrupt flag
#asm
    in    r30,spsr
    in    r30,spdr
#endifasm
    lcd_init(16);

    lcd_gotoxy(0,0);
    lcd_puts("S Send: ");
    lcd_gotoxy(0,1);
    lcd_puts("S recieve: ");

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

while (1);

}

```

در صورتی که وقفه فعال نشده باشد، می‌توان بدون استفاده از Code Wizard و فایل سرآیند spi.h و به کمک توابعی که در قالب فایل‌های جانبی به پروژه اضافه می‌گردند، برای پیکربندی و تبادل داده SPI اقدام نمود. تنظیمات LCD را هم می‌توان از طریق Project/Configure/C compiler/libraries/ Alphanumeric LCD (alcd.h) ویرایش نمود. در برنامه ۹-۸ نمونه فایل سرآیند آمده است.

```

#ifndef SPI_H_files_H_
#define SPI_H_files_H_

#include <io.h>                                /* Include AVR std. library file */

#define MOSI 5                                     /* Define SPI bus pins */
#define MISO 6
#define SCK 7
#define SS 4
#define SS_Enable PORTB &= ~(1<<SS)           /* Define Slave enable */
#define SS_Disable PORTB |= (1<<SS)             /* Define Slave disable */

8-9 برنامه

void SPI_Slave_Init();                         /* SPI Initialize function */
char SPI_Slave_Transmit(char data);            /* SPI transmit data function */
char SPI_Slave_Receive();                      /* SPI Receive data function */

void SPI_Master_Init();                        /* SPI initialize function */
void SPI_Master_Write(char);                  /* SPI write data function */
char SPI_Master_Read();                       /* SPI read data function */

#endif

```

در برنامه 9-9 پیاده‌سازی مربوط به توابع فوق قابل مشاهده است.

```

#include "SPI_H_files.h"                           /* SPI Initialize function */
void SPI_Slave_Init()                            /* Make MOSI, SCK, SS pin direction as input pins */
{
    DDRB &= ~((1<<MOSI) | (1<<SCK) | (1<<SS));
    DDRB |= (1<<MISO);                          /* Make MISO pin as output pin */
    SPCR = (1<<SPE);                           /* Enable SPI in slave mode */
}

9-9 برنامه

char SPI_Slave_Transmit(char data)               /* SPI transmit data function */
{
    SPDR = data;                                /* Write data to SPI data register */
    while(!(SPSR & (1<<SPIF)));              /* Wait till transmission complete */
    return(SPDR);                               /* return received data */
}

char SPI_Slave_Receive()                         /* SPI Receive data function */
{
    while(!(SPSR & (1<<SPIF)));              /* Wait till reception complete */
    return(SPDR);                               /* return received data */
}

void SPI_Master_Init()                          /* SPI Initialize function */
{
    DDRB |= (1<<MOSI) | (1<<SCK) | (1<<SS); /* Make MOSI, SCK, 0th pin direction as output pins */
    DDRB &= ~(1<<MISO);                      /* Make MISO pin as input pin */
    PORTB |= (1<<SS);                         /* Disable slave initially by making high on SS pin*/
    SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0);
    /* Enable SPI, Enable in master mode, with Fosc/16 SCK frequency */
    SPSR &= ~(1<<SPI2X);                      /* Disable speed doubler */
}

```

```

}

/*****************************************/
void SPI_Master_Write(char data)           /* SPI write data function */
{
    char flush_buffer;
    SPDR = data;                      /* Write data to SPI data register */
    while(!(SPSR & (1<<SPIF)));      /* Wait till transmission complete */
    flush_buffer = SPDR;                /* Flush received data */

    /* Note: SPIF flag is cleared by first reading SPSR (with SPIF set) and then
       accessing SPDR hence flush buffer used here to access SPDR after SPSR read */
}
/*****************************************/
char SPI_Master_Read()                   /* SPI read data function */
{
    SPDR = 0xFF;
    while(!(SPSR & (1<<SPIF)));      /* Wait till reception complete */
    return(SPDR);                     /* return received data */
}
/*****************************************/

```

نمونه برنامه Master که در آن از توابع پیاده‌سازی در بالا استفاده شده، در برنامه 9-10 آمده است. نمونه برنامه Slave هم در برنامه 9-10 نشان داده شده است.

```

/*Master Program*/

#include <mega16.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
#include "SPI_H_files.h"

10-9 برنامه

void main(void)
{
    char count;
    char buffer[5];

    lcd_init(16);
    SPI_Master_Init();
    lcd_gotoxy(0,0);
    lcd_puts("Master Device");
    lcd_gotoxy(0,1);
    lcd_puts("Sending:  ");

    SS_Enable;
    count = 0;
    while (1)
    {
        SPI_Master_Write(count);
        sprintf(buffer, "%d    ", count);
        lcd_gotoxy(10,1);
        lcd_puts(buffer);
        count++;
        delay_ms(500);
    }
}

```

```

}

/* Slave program*/



#include <mega16.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
#include "SPI_H_files.h"
11-9 برنامه

void main(void)
{
    char count;
    char buffer[5];

    lcd_init(16);
    SPI_Slave_Init();

    lcd_gotoxy(0,0);
    lcd_puts("Slave Device");
    lcd_gotoxy(0,1);
    lcd_puts("Receive:");

    while (1)
    {
        count = SPI_Slave_Receive();
        sprintf(buffer, "%d", count);
        lcd_gotoxy(13,1);
        lcd_puts(buffer);
    }

}

```

## 9.7 مقایسه ارتباط سریال و SPI

در این بخش روش‌های ارتباط سریال UART و SPI با یکدیگر مقایسه شده اند و نقاط قوت و ضعف هر یک از آن‌ها در کاربردهای مختلف تشریح گردیده است.

### 9.7.1 UART

در ارتباط UART، به دلیل این که ارتباط از نوع آسنکرون است، طرفین ارتباط باید از قبل بر سر یک نرخ انتقال داده‌ی مشخص توافق کنند. همچنین هر دو دستگاه باید پالس ساعت‌هایی نزدیک به همان نرخ انتقال داشته باشند. اختلاف زیاد بین نرخ‌های پالس ساعت در هر یک از دو سمت منجر به از دست رفتن داده می‌شود.

همان‌طور که در مبحث UART گفته شد، درگاه‌های سریال آسنکرون نیاز به سخت‌افزار جانبی مانند MAX232 دارند. همچنین حداقل نیاز به یک بیت شروع و یک بیت پایان بخش درون هر بسته داده است، به این معنی که برای انتقال هر 8 بیت داده به زمانی معادل با انتقال 10 بیت داده نیاز است و این به شدت نرخ انتقال داده را پایین می‌آورد. یک ایراد بنیادی دیگر در ارتباط آسنکرون این است که این نوع ارتباط صرفاً برای ارتباط بین دو دستگاه طراحی شده است. نرخ تبادل داده نیز یک مشکل است. با این که از نظر تئوری محدودیتی در ارتباطات سریال آسنکرون وجود ندارد، بیشتر دستگاه‌های USART تها از مجموعه‌ای از نرخ‌های ثابت از پیش تعیین شده استفاده می‌کنند. بالاترین این نرخ‌ها معمولاً حدود 230400 بیت بر ثانیه است.

### SPI 9.7.2

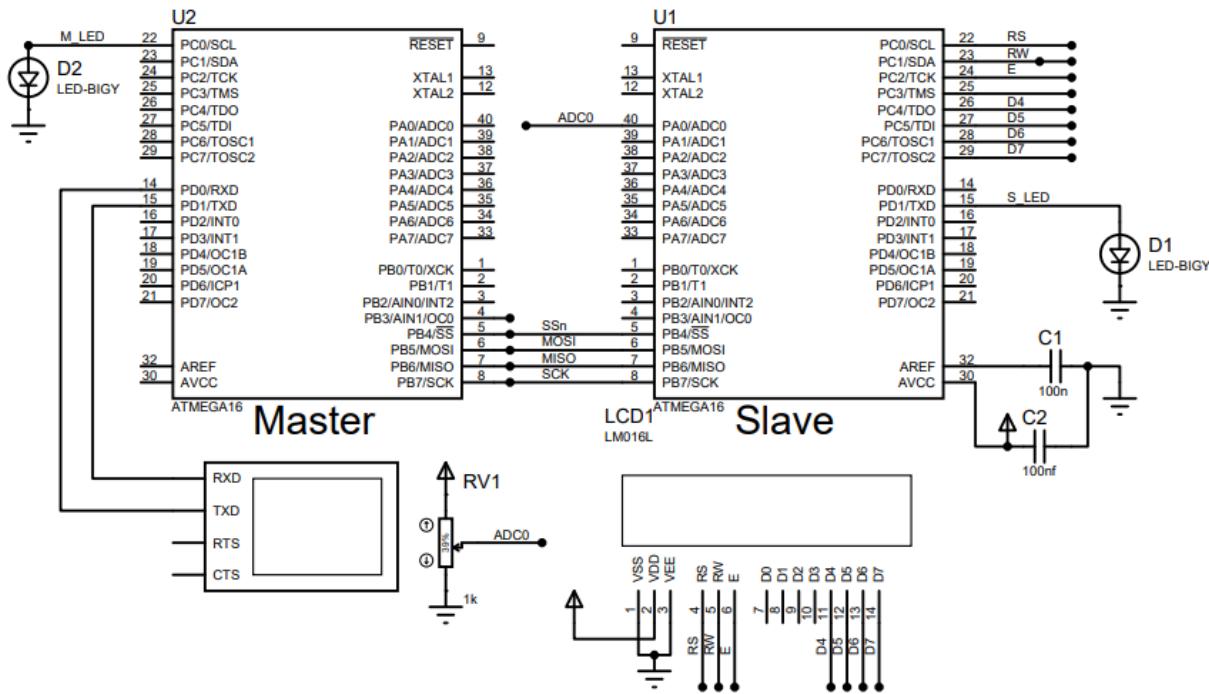
بزرگ‌ترین ایراد SPI تعداد پایه‌های مورد نیاز است. برقراری ارتباط SPI تنها بین یک زوج دستگاه Master و Slave نیاز به 4 سیم دارد. هر دستگاه Slave دیگری هم به ارتباط اضافه شود، یک پایه SS<sub>n</sub> بین آن و دستگاه Master اضافه خواهد شد. افزایش سریع تعداد اتصالات پایه‌ها، این پروتکل را در شرایطی که تعداد زیادی دستگاه Slave باید به یک Master متصل شوند، پیاده‌سازی را غیر ممکن می‌کند. همچنین، تعداد زیاد اتصالات برای هر دستگاه، طراحی PCB را با چالش مواجه می‌کند.

ارتباط SPI فقط می‌تواند از یک Master و تعداد زیادی Slave پشتیبانی نماید و تعداد Slave‌ها به ظرفیت دستگاه‌های متصل به گذرگاه و تعداد پایه‌های SS<sub>n</sub> بستگی دارد.

برای ارتباطات SPI Full-Duplex (ارسال و دریافت همزمان داده) با نرخ انتقال بالا مناسب است، زیرا از سرعت‌هایی بیشتر از 10MHz (نرخ داده 10 میلیون بیت بر ثانیه) در بعضی شرایط پشتیبانی می‌کند. سخت‌افزار مورد استفاده در هر طرف هم معمولاً یک ثبات چرخشی ساده است، و عمدۀ پروتکل به صورت نرم افزاری و با هزینه کم قابل پیاده‌سازی خواهد بود.

### SPI 9.7.3 برنامه‌های اجرایی مبحث

سخت‌افزار شکل 9-8 را در نظر بگیرید و برنامه‌های زیر را برای هر یک از دستگاه‌های Master و Slave بنویسید.



## 10 جلسه دهم

### ارتباط سریال دو سیمه (TWI یا I<sup>2</sup>C)

#### 10.1 هدف

در این جلسه نحوه برقراری ارتباط با استفاده از پروتکل سریال دو سیمه مورد بررسی قرار خواهد گرفت.

#### 10.2 مقدمه

ارتباط سریال دو سیمه Inter-Integrated Circuit یا به اختصار I<sup>2</sup>C، یک پروتکل ارتباطی سریال است که برای نخستین بار توسط شرکت Philips ارایه شد و در سال‌های اخیر به طور گسترده توسط بسیاری از کمپانی‌ها استفاده می‌شود. این پروتکل برای اتصال لوازم جانبی کم سرعت به بورد اصلی کامپیوتر یا ریزپردازنده در سیستم‌های تعبیه شده مورد استفاده قرار می‌گیرد.

I<sup>2</sup>C می‌تواند تا 128 وسیله مختلف را از طریق یک ارتباط اتصال‌گرا به همراه مکانیزم تصدیق به یکدیگر متصل کند. هر کدام از این تجهیزات متصل شده یک گره یا node خوانده می‌شوند که می‌تواند Master یا Slave باشد. وظیفه تولید پالس ساعت سیستم بر عهده‌ی Master می‌باشد و Slave گره‌ای است که پالس ساعت و آدرس را توسط Master دریافت می‌کند.

پروتکل I<sup>2</sup>C در مد آدرس دهی ده بیتی می‌تواند حداقل از 1008 دستگاه Slave پشتیبانی کند. بر خلاف پروتکل SPI، امکان اتصال چند دستگاه Master نیز وجود دارد تا در نوبت‌های تعیین شده با دستگاه‌های Slave در ارتباط باشند. نرخ انتقال داده در این پروتکل در محدوده‌ی 100 تا 400 کیلوهرتز قرار دارد. همچنین پروتکل I<sup>2</sup>C برای هر 8 بیت داده، یک بیت اضافه به نام بیت ACK/NACK اختصاص می‌دهد.

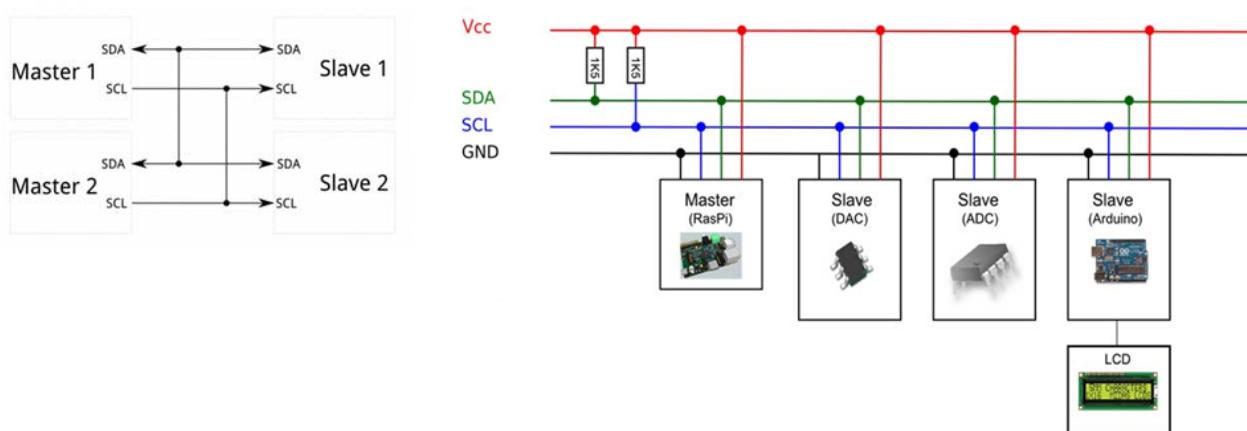
گذرگاه I<sup>2</sup>C از دو سیگنال SCL برای انتقال پالس ساعت و SDA برای تبادل داده تشکیل گردیده که هر کدام به از آن‌ها به یک مقاومت بالاکش متصل شده است. پالس ساعت گذرگاه نیز همواره توسط دستگاه Master تولید می‌شود.

در تمامی وسایلی که از TWI حمایت می‌کنند، درایورهای گذرگاه به صورت open drain یا open collector عمل می‌کنند. این ویژگی موجب می‌شود تا آن‌ها به صورت wire-AND عمل کنند که برای راه اندازی گذرگاه ضروری است. بنابراین یک سطح پایین در گذرگاه TWI، زمانی تولید می‌شود که خروجی یک یا چند وسیله صفر باشد و سطح بالای آن نیز تنها زمانی که تمام وسایل TWI در حالت امپدانس بالا باشند، حاصل می‌گردد ( مقاومت بالاکش هنگامی که هیچ دستگاهی آن را پایین نمی‌کشد گذرگاه را در سطح بالا نگه می‌دارد).

در ریزپردازنده AVR ضمن وجود پایه‌های اختصاصی برای این پروتکل، امکان استفاده از تمام درگاه‌های موجود برای I<sup>2</sup>C وجود دارد. به فرآیندهای آن به صورت سختافزاری و بر روی پایه‌های اختصاصی انجام می‌شود، TWI نیز می‌گویند.

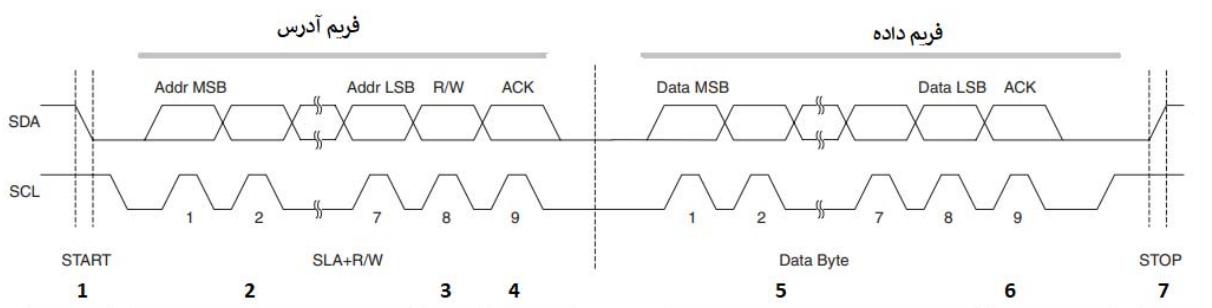
### I<sup>2</sup>C پروتکل 10.3

در حالت کلی مطابق شکل 1-10، تعدادی دستگاه Master و Slave به گذرگاه متصل می‌شوند که هر یک از آنها دارای آدرس مشخصی است. هنگامی که Master قصد ارسال داده دارد، ابتدا آدرس Slave را روی گذرگاه قرار می‌دهد. دستگاه Slave هم با دریافت آدرس خود، گذرگاه را در اختیار می‌گیرد و سایر Slave‌ها غیرفعال می‌شوند. به این صورت ارتباط Master و Slave برقرار می‌شود.



شکل 10-1: نمایی از ارتباط I<sup>2</sup>C

جزئیات فرایند اشاره شده در بالا در شکل 10-2 نشان داده شده است.



.1 شروع بسته (Start)

2.	ارسال آدرس Slave (SLA)
3.	ارسال R/W
4.	ارسال ACK
5.	ارسال داده
6.	ارسال ACK
7.	پایان بسته (Stop)

شکل 10-2: نمایی از تبادل داده در پروتکل I<sup>2</sup>C

1 - شروع بسته (Start): قبل از ارسال آدرس، دستگاه Master خط SCL را بالا نگه داشته و SDA را پایین می‌کشد. این کار به تمامی دستگاه‌های Slave متصل به گذرگاه اعلام می‌کند که آماده‌ی دریافت آدرس باشند. اگر دو دستگاه Master در یک زمان قصد ارسال داده را داشته باشند، دستگاهی که زودتر خط SDA را پایین بکشد برنده خواهد بود و کنترل گذرگاه را در اختیار می‌گیرد.

2 - ارسال آدرس Slave (SLA): آدرس Master مورد نظر که قرار است با آن ارتباط برقرار کند را ارسال می‌کند. این آدرس 7 بیتی است و ابتدا MSB آن ارسال می‌شود. در این وضعیت تمام Slave‌ها آدرس را دریافت می‌نمایند و با آدرس خود مقایسه می‌کنند. دستگاه Slave که در آن تطابق صورت پذیرد گذرگاه را در اختیار گرفته و سایر Slave‌ها گذرگاه را رها می‌کنند.

البته در پروتکل I<sup>2</sup>C امکان ارسال آدرس به صورت ده بیتی نیز وجود دارد که تعداد Slave‌ها را افزایش می‌دهد. در این حالت از دو بسته آدرس استفاده می‌شود که می‌تواند همزمان با Slave‌های دارای آدرس 7 بیتی و 10 بیتی کار نماید.

3 - ارسال R/W: در ادامه Master بیتی را مبنی بر این که آیا از Slave داده‌ای را می‌خواهد دریافت کند (مقدار یک) و یا این که قصد دارد داده‌ای را برای Slave ارسال نماید (مقدار صفر)، می‌فرستد.

4 - نهمین بیت این بسته، بیت NACK/ACK است. در این مرحله اگر Slave در مدار بوده و آماده به کار باشد، این بیت را صفر می‌کند. به این طریق Master متوجه می‌شود که Slave پیام او را دریافت کرده و می‌تواند ارتباط را ادامه دهد. ولی اگر Slave صفر را ارسال نکند یا به عبارتی گذرگاه را به سطح پایین تغییر ندهد، Master متوجه می‌شود که مشکلی در ارتباز پیش آمده یا Slave اصلاً به گذرگاه متصل نیست. پس انتقال داده متوقف می‌شود و Master بنا به شرایط تصمیم می‌گیرد که چه فرایندی را دنبال نماید.

5 - بعد از این که بسته آدرس ارسال شد و Slave با موفقیت پاسخ داد، Master به تولید پالس ساعت ادامه می‌دهد. در این زمان اگر R/W=1 باشد Slave داده را می‌فرستد و در غیر این صورت Master داده را ارسال می‌نماید.

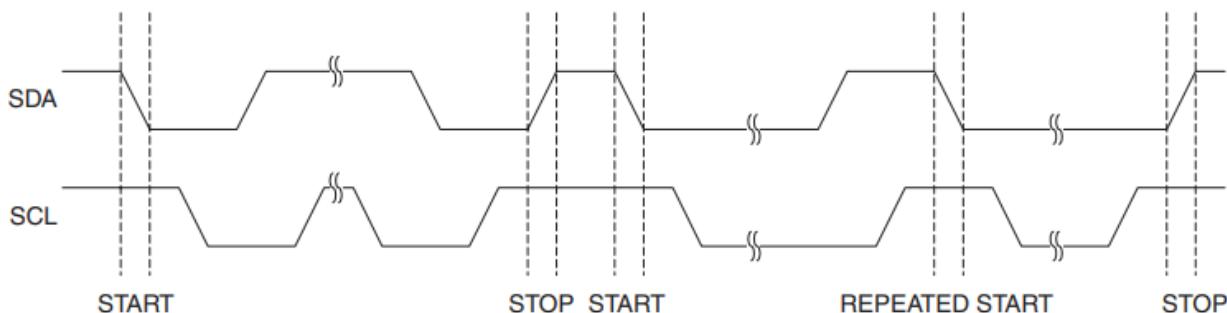
6- بسته‌های داده نیز مشابه بسته آدرس، دارای یک بیت ACK/NACK هستند که اگر داده‌ی ارسالی از Master در Slave دریافت شد، ACK توسط Slave با مقدار صفر ارسال می‌گردد. در صورتی هم که Slave نتواند داده را دریافت نماید، خط SDA را در حالت بالا رها می‌کند (NACK). از سوی دیگر، هنگامی که Master بایت را از Slave دریافت می‌کند ACK را با مقدار صفر برای Slave می‌فرستد. چنان‌چه به آخرین بایت دریافتی از Slave برسد و یا به هر دلیلی نتواند بایت دیگری را دریافت کند، با رها کردن خط در حالت بالا، یک NACK ارسال می‌نماید.

7- هنگامی که تمامی بسته‌های داده از جانب Master ارسال شدند و یا Master تمام داده‌های مورد نظر از Slave جانب Slave را دریافت نمود، Master یک وضعیت توقف ایجاد می‌کند که نشان‌دهنده پایان ارتباط با Slave است. پس از آن، هم گذرگاه را رها می‌کند. این وضعیت با یک تغییر سطح از صفر به یک بر روی خط SDA، در حالی که خط SCL در سطح یک قرار دارد ایجاد می‌گردد. بنابراین در طی فرایند نوشتن معمولی، هنگامی که خط SCL بالاست، نباید مقدار SDA تغییر کند تا از بروز وضعیت توقف اشتباهی جلوگیری شود.

در I<sup>2</sup>C همگی دستگاه‌های Master و Slave توانایی ارسال و دریافت داده را دارند. همچنین در این پروتکل هر دستگاه متصل به گذرگاه، بنا به شرایط قابلیت قرار گرفتن در وضعیت Master یا Slave را دارد. البته همواره در هر لحظه از زمان فقط یک Master کنترل گذرگاه را در اختیار دارد. پس در کل 4 حالت برای ارسال و دریافت وجود دارد که عبارتند از Slave Receiver، Slave Transmitter، Master Receiver و Master Transmitter.

#### 10.4 وضعیت شروع مکرر

بعضی اوقات، لازم است به یک دستگاه Master اجازه داده شود تا چندین پیام را پشت سرهم و بدون اجازه دادن به سایر Master‌ها روی گذرگاه ارسال کند. برای این کار، وضعیت شروع مکرر مانند شکل 10-3 تعریف شده است.



شکل 10-3: وضعیت شروع مکرر

برای انجام شروع‌های مکرر، در حالی که خط SCL پایین است، خط SDA بالا می‌رود و سپس SCL بالا می‌رود. در ادامه هنگامی که SCL بالاست SDA پایین کشیده می‌شود. در این حالت Master بدون این که کنترل گذرگاه را رها کند، تبادل داده‌ی جدیدی را شروع می‌نماید. پیام جدید نیز مانند هر پیام دیگری شامل یک بسته آدرس و سپس بسته‌های داده ارسال می‌گردد. در پروتکل I<sup>2</sup>C هر تعداد شروع مکرر مجاز است و فرایند ارسال داده تا زمانی که با ارسال وضعیت توقف کنترل گذرگاه را رها نماید می‌تواند ادامه یابد.

## I<sup>2</sup>C ثبات‌های 10.5

ثبات (TWI Bit Rate register) TWBR: این ثبات مانند شکل 10-4 فاکتور تقسیم مربوط به تولیدکننده نرخ بیت را انتخاب می‌کند. تولیدکننده نرخ بیت یک تقسیم‌کننده فرکانسی است که در دستگاه Master پالس ساعت را تولید می‌کند. SCL

Bit	7	6	5	4	3	2	1	0	TWBR
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

شکل 10-4: ساختار ثبات TWBR

$$SCL \ frequency = \frac{CPU \ clock \ frequency}{16 + 2(TWBR). 4^{TWPSS}}$$

در این رابطه TWSR به محتوای ثبات (0:1) اشاره دارد.

## ثبات (TWI Control Register) TWCR 10-5

این ثبات مانند شکل 10-5 برای کنترل TWI یا I<sup>2</sup>C به کار می‌رود.

Bit	7	6	5	4	3	2	1	0	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

شکل 10-5: ساختار ثبات TWCR

بیت 7-TWINT (TWI Interrupt Flag): این بیت پس از اتمام عملیات تبدل داده TWI توسط سخت‌افزار یک می‌شود و می‌تواند اجرای روتین وقفه را آغاز نماید. صفر نمودن آن توسط نرم افزار انجام می‌شود.

بیت 6-TWEA (TWI Enable Acknowledge Bit): این بیت تولید پالس ACK را کنترل می‌کند به این صورت که اگر مقدار آن 1 باشد، پالس ACK روی گذرگاه TWI تولید می‌شود.

بیت 5-TWSTA (TWI START Condition Bit): زمانی که یک وسیله در حالت Master باشد، کاربر می‌تواند با نوشتن مقدار 1 در بیت TWSTA، حالت شروع را ایجاد کند.

بیت 4-TWSTO (TWI STOP Condition Bit): در حالت Master با نوشتن مقدار 1 در بیت TWSTO، یک حالت توقف روی گذرگاه TWI ایجاد می‌شود.

بیت 3-TWWC (TWI Write Collision Flag): زمانی که سعی شود تا با وجود صفر بودن TWINT، داده‌ای در داخل ثبات داده نوشته شود این بیت یک می‌گردد.

بیت 2-TWEN (TWI Enable Bit): واحد سخت‌افزاری TWI را فعال می‌کند.

بیت 1-Res (Reserved Bit): این بیت رزرو شده است.

بیت صفر-TWIE (TWI Interrupt Enable): زمانی که این بیت و بیت وقفه عمومی در ثبات SREG یک باشند، درخواست وقفه TWI مدامی که پرچم TWINT یک باشد فعال می‌گردد.

**ثبات TWSR:**

این ثبات مانند شکل 10-6 برای نمایش وضعیت TWI یا I<sup>2</sup>C به کار می‌رود.

Bit	7	6	5	4	3	2	1	0	TWSR
Read/Write	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	
Initial Value	R	R	R	R	R	R	R/W	R/W	

شکل 10-6: ساختار ثبات TWSR

بیت‌های 3 تا 7-TWS (TW Status): این پنج بیت، وضعیت TWI و گذرگاه سریال را نشان می‌دهند. در هر مرحله از ارسال فریم I<sup>2</sup>C، کدی در این بخش قرار می‌گیرد که به شرایط خاصی از پروسه ارسال و دریافت اشاره می‌نماید. جزئیات بیشتر در راهنمای تراشه آمده است.

بیت 2-Res (Reserved Bit): این بیت رزرو شده است.

بیت‌های یک تا صفر-TWI Prescaler Bits (TWPS): این بیت‌ها می‌توانند خوانده یا نوشته شوند و تقسیم‌کننده نرخ بیت را مشخص می‌نمایند که در فرکانس پالس ساعت I<sup>2</sup>C تاثیرگذار است.

#### ثبات (TWI Data Register) TWDR

در حالت ارسال، ثبات TWDR بایت بعدی که باید ارسال شود را در خود نگه می‌دارد و در حالت دریافت آخرين بایت دریافت‌شده را در خود جای می‌دهد. نمایی از ثبات TWDR در شکل 10-7 نشان داده شده است.

Bit	7	6	5	4	3	2	1	0	TWDR
Read/Write	R/W								
Initial Value	1	1	1	1	1	1	1	1	

شکل 10-7: ساختار ثبات TWDR

#### ثبات (TWI (Slave) Address Register) TWAR

این ثبات مطابق ساختار نشان داده در شکل 10-8 با یک آدرس 7 بیتی که نشان‌دهنده آدرس وسیله است پر می‌شود.

Bit	7	6	5	4	3	2	1	0	TWAR
Read/Write	R/W								
Initial Value	1	1	1	1	1	1	1	0	

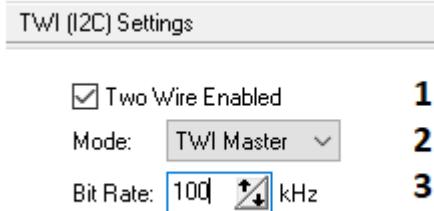
شکل 10-8: ساختار ثبات TWAR

بیت‌های 7 تا 1 (TWI Slave Address Register) TWA-1: این 7 بیت آدرس SLAVE را در خود نگاه می‌دارند. بیت صفر-TWI General Call Recognition Enable Bit (TWGCE): اگر مقدار این بیت برابر 1 باشد تشخیص ارسال چندپخشی بر روی گذرگاه را فعال می‌نماید. در این نوع از ارسال دستگاه Master می‌تواند داده‌ای را برای تمام Slave‌ها ارسال کند.

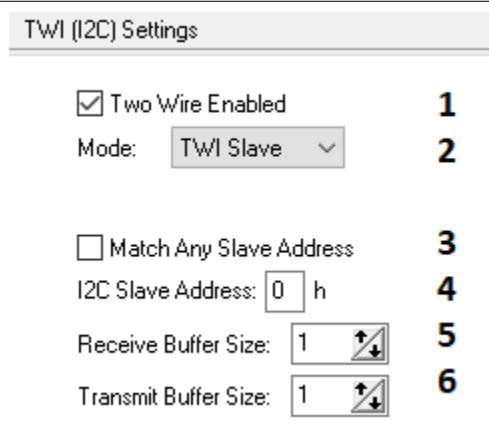
### 10.6 از طریق I<sup>2</sup>C فعال سازی Codevision

برای فعال سازی I<sup>2</sup>C در محیط نرم‌افزار CodeVision می‌توان از تنظیمات نشان داده شده در شکل 10-9 استفاده نمود. تنظیم نرخ بیت بر عهده Msater می‌باشد. برای Slave می‌توان آدرس مورد نظر را تنظیم نمود. همچنین می‌توان دو بافر مجزا با اندازه دلخواه برای ارسال و دریافت داده‌ها در نظر گرفت.

Master



Slave

شکل 9-10: تنظیمات I<sup>2</sup>C در محیط CodeWizard

## TWI یا I<sup>2</sup>C 10.7

در Codevision دو کتابخانه برای کار با توابع I<sup>2</sup>C فراهم شده است:

- کتابخانه twi.h که برای راهاندازی I<sup>2</sup>C سختافزاری یا TWI به کار می‌رود.

- کتابخانه i2c.h که برای راه اندازی I<sup>2</sup>C به صورت نرمافزاری آماده شده است.

کتابخانه twi.h از ثبات‌های اختصاصی ریزپردازنده استفاده می‌کند و از همین رو سرعت بیشتری بهره می‌برد. در مقابل کتابخانه i2c.h به صورت کاملاً نرمافزاری بوده و تمام سیگنال‌ها توسط CPU تولید می‌شود و به همین خاطر سرعت کمتری دارد. مزیت i2c.h این است که پایه‌های SDA و SCL را به دلخواه می‌توان بر روی هر کدام از پین‌های ریزپردازنده تنظیم نمود.

## فایل سرآیند twi.h 10.7.1

می‌توان از توابع معرفی شده در فایل سرآیند twi.h برای هر یک از حالت‌های Master یا Slave استفاده کرد. این توابع عبارتنداز:

تابع `twi_master_init` که برای پیکربندی رابط I<sup>2</sup>C در حالت Master به کار می‌رود.

تابع `twi_master_trans` که برای تبادل داده با Slave استفاده می‌شود.

تابع `twi_slave_init` که برای پیکربندی I<sup>2</sup>C در حالت Slave استفاده می‌شود و تبادل داده هم با همین دستور انجام

می‌گردد.

**نکته:** در صورت استفاده از این کتابخانه حتماً باید بیت وقفه عمومی را فعال کرد.

### i2c.h فایل سرآیند 10.7.2

اگر ریزپردازنده فقط قرار است به عنوان Master فعال باشد، می‌توان از فایل سرآیند `i2c.h` با قابلیت انتخاب هر یک از پایه‌های ریزپردازنده به عنوان گذرگاه I<sup>2</sup>C استفاده نمود. توابع این فایل سرآیند عبارتند از:

تابع `i2c_init` که تنظیمات اولیه I<sup>2</sup>C را انجام می‌دهد و به صورت پیش فرض، نرخ ارسال را بر روی بیشترین مقدار تنظیم می‌کند.

تابع `i2c_start` یک بیت آغاز ایجاد می‌کند که برای شروع یک انتقال داده الزامی است. این تابع پارامتر ورودی و خروجی ندارد.

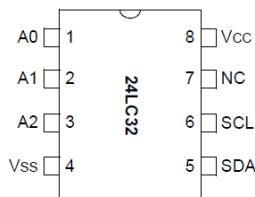
تابع `i2c_write` یک پارامتر ورودی از نوع `char` دارد که مقدار آن را بر روی رابط I<sup>2</sup>C ارسال می‌کند.

تابع `i2c_read` یک بایت را از رابط I<sup>2</sup>C دریافت می‌کند و به عنوان آرگومان خروجی برمی‌گرداند. همچنین این تابع یک آرگومان ورودی دارد که وضعیت بیت ACK را معلوم می‌کند. اگر ورودی 1 باشد، پس از خواندن بایت، بیت ACK و در غیر این صورت، بیت NACK برای Slave فرستاده خواهد شد.

تابع `i2c_stop` یک بیت پایان به نشانه اتمام تبادل داده با Slave ایجاد می‌کند.

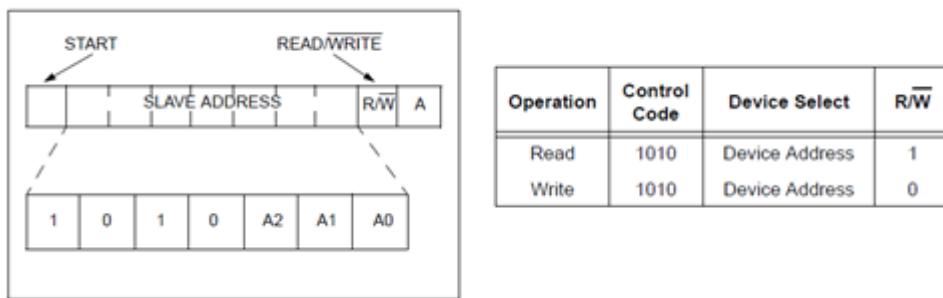
### EEPROM ارتباط ریزپردازنده با حافظه 10.8

حافظه‌های EEPROM به طور کلی به دو دسته موازی و سریال تقسیم می‌گردند. از EEPROM‌های موازی می‌توان به خانواده حافظه‌های 28CXX اشاره کرد که سرعت زیادی در دسترسی به داده‌ها دارند و در مقابل حافظه‌های سریال با سرعت پایین‌تر و تعداد پایه‌ها و حجم کمتر مدار هستند. دسته دوم برای برقراری ارتباط سریال از پروتکل‌های Micro wire, I<sup>2</sup>C, SPI و یک سیمه استفاده می‌نمایند. در ادامه تراشه حافظه سریال 24LC32 که در شکل 10-10 نشان داده شده است و از پروتکل ارتباطی I<sup>2</sup>C استفاده می‌کند بررسی می‌شود.



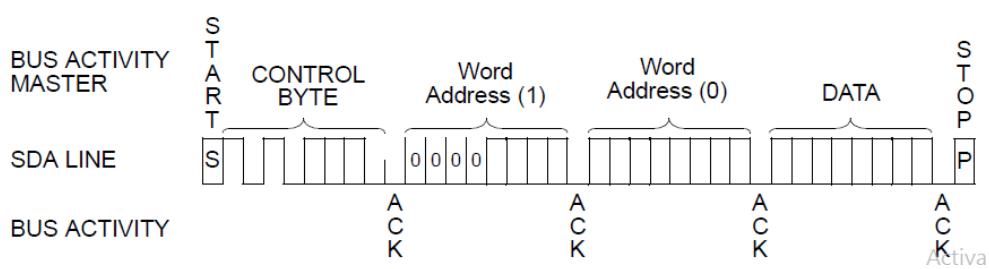
شکل 10-10: پایه‌های تراشه حافظه 24LC32

آدرس Slave برای این تراشه به صورت سخت‌افزاری با اتصال پایه‌های A0 تا A2 به ولتاژ زمین یا Vcc مطابق شکل 11-10 تنظیم می‌گردد. به عنوان مثال اگر همه این پایه‌ها به زمین متصل شوند، آدرس Slave برابر با 0x50 خواهد بود. بر اساس این نحوه آدرس‌دهی حداکثر 8 تراشه‌ی حافظه از این مدل را می‌توان به گذرگاه I<sup>2</sup>C متصل نمود.



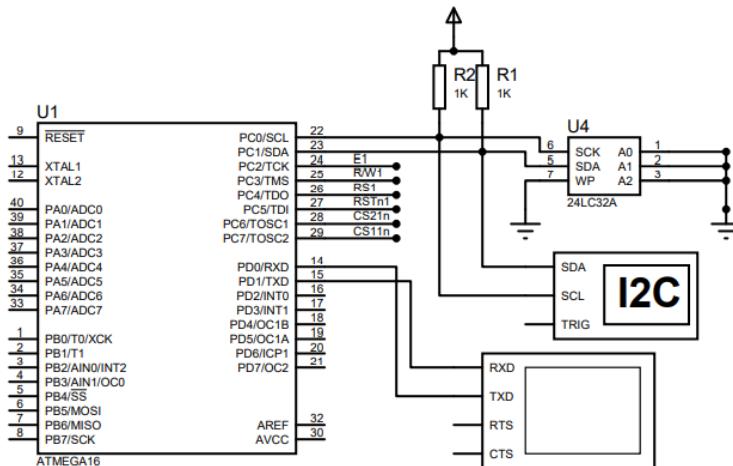
شکل 11-10: آدرس Slave برای حافظه EEPROM

در حالت کلی مانند شکل 12-10، پردازندۀ به عنوان Master پس از ارسال آدرس Slave مورد نظر (Address), آدرس فضایی که قرار است عملیات خواندن یا نوشتمن در آن انجام شود را ارسال نموده و سپس داده‌ها مبادله می‌شوند. حجم این حافظه 32KB است و فضای آدرس آن در محدوده 0x0000 تا 0x03FF قرار دارد. بنابراین آدرس داده درون حافظه با دو بایت نشان داده می‌شود.

شکل 12-10: بسته ارتباطی حافظه EEPROM با پروتکل I<sup>2</sup>C

#### 10.8.1 ارتباط ریزپردازنده با حافظه EEPROM از طریق توابع twi.h

برای ارتباط ریزپردازندۀ با حافظه، سخت‌افزار شکل 10-13 را در نظر بگیرید. در این سخت‌افزار بلوکی به نام "I<sup>2</sup>C" است که اصطلاحا Debugger I<sup>2</sup>C نامیده می‌شود و در هنگام اجرای برنامه، تمام اطلاعاتی که روی گذرگاه I<sup>2</sup>C ارسال می‌شود را نشان می‌دهد. به همین سبب راهنمای بسیار خوبی برای درک پروتکل I<sup>2</sup>C است.



شکل ۱۰-۱۳: نمایی از سخت‌افزار مبحث C<sup>۲</sup> برای ارتباط ریز پردازنده و حافظه EEPROM

برای ارتباط با حافظه از طریق I<sup>2</sup>C و توابع موجود در فایل سرآیند twi.h می‌توان از برنامه 10-1 استفاده نمود. این برنامه در حافظه EEPROM با آدرس Salve برابر با 0x50، داده‌هایی را در آدرس 0x0010 به بعد می‌نویسد و تعدادی داده را از آدرس 0x0040 به بعد می‌خواند. روند اجرای این برنامه در محیط نرمافزار Proteus نیز در شکل 10-14 نشان داده شده است.

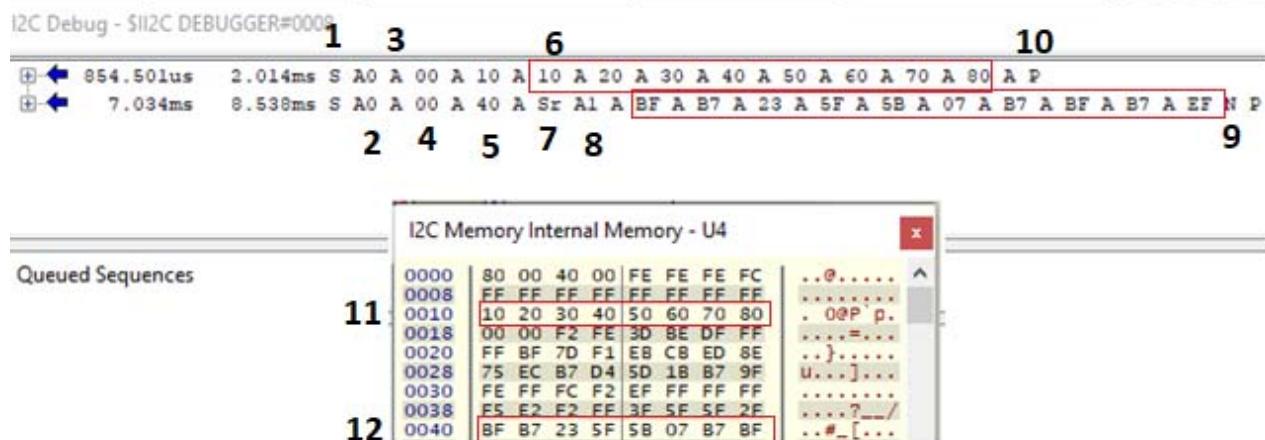
```
#include <mega16.h>
#include <twi.h>
#include <delay.h>

void main(void)
{
    unsigned char tx_data[10]={0x00,0x10,0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80};
    unsigned char rx_data[10];
    unsigned char *txp=&tx_data[0];
    unsigned char *rpx=&rx_data[0];

    twi_master_init(100);
    #asm("sei")

    twi_master_trans(0x50, txp ,10,rpx,0);    // for write to eeprom
    delay_ms(5);
    tx_data[0]=0x00;
    tx_data[1]=0x40;
    twi_master_trans(0x50, txp ,2,rpx,10);    //for read from eeprom
```

```
while (1);
}
```

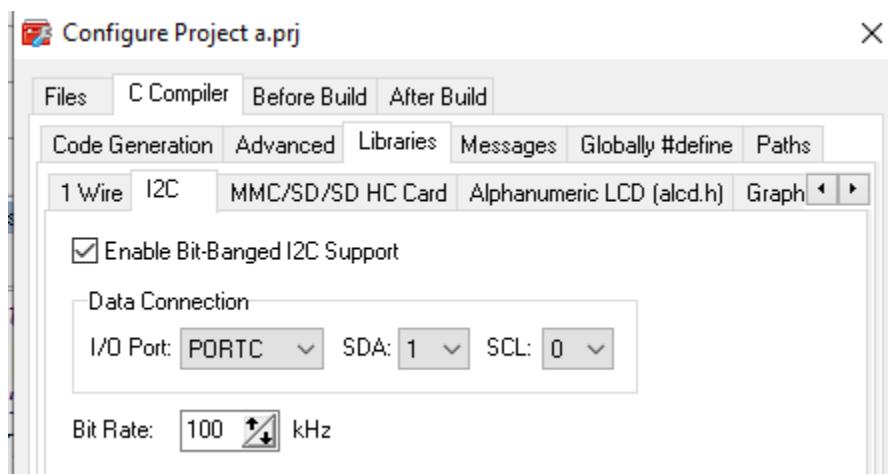


- 1: شروع مجدد
- 2: آدرس Slave برای نوشت - 0xA0=(l<<0x50)| 0
- 3: پیام ACK
- 4: بایت با ارزش آدرس مکانی درون حافظه
- 5: بایت کم ارزش آدرس مکانی درون حافظه
- 6: داده‌ها
- 7: آدرس Slave برای خواندن - 1
- 8: not ACK
- 9: پایان
- 10: فضای حافظه از آدرس 0x0010
- 11: فضای حافظه از آدرس 0x0040
- 12: فضای حافظه از آدرس 0x0040

شکل 10-14: خروجی اجرای برنامه در I2C Debugger

### 10.8.2 ارتباط ریزپردازنده با حافظه EEPROM از طریق توابع i2c.h

برای ارتباط با حافظه، با I<sup>2</sup>C و توابع موجود در فایل سرآیند i2c.h ابتدا بایستی تنظیمات شکل 10-15 را انجام داد. سپس این فایل سرآیند را در ابتدای برنامه فراخوانی نمود.



شکل 10-15 : تنظیمات لازم برای استفاده از i2c.h در پروژه

در ادامه می‌توان از برنامه 10-2 استفاده نمود. این برنامه در حافظه EEPROM که دارای آدرس Salve برابر با 0x50 است داده‌هایی را از آدرس 0x0010 دورن فضای حافظه می‌نویسد و تعدادی داده را از آدرس 0x0010 حافظه می‌خواند. روند اجرای آن در محیط نرم‌افزار Proteus نیز در شکل 10-16 نشان داده شده است.

```
#include <mega16.h>
#include <delay.h>
#include <i2c.h>

void main(void)
{
    2-10    char i;
    unsigned char tx_data[12]={0x00 ,0x10 ,0x11 ,0x22 ,0x33 ,0x44 ,0x55 ,0x66
    ,0x70 ,0x80 ,0x90,0xA0};
    unsigned char rx_data[12];

    // TWI initialization
    // Mode: TWI Master
    // Bit Rate: 100 kHz

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

    // write a fram to eeprom via new code
    {
        i2c_start();
        i2c_write(0xA0);
        i2c_write(0x00); // tx_data[0]
        i2c_write(0x10); // tx_data[1]
        for(i=0;i<10;i++)
        {
            i2c_write(tx_data[i+2]);
        }
        i2c_stop();

        delay_ms(10);
    }

    // write a fram to eeprom via new code
    {
        i2c_start();
        i2c_write(0xA0);
        i2c_write(0x00); // tx_data[0]
        i2c_write(0x10); // tx_data[1]

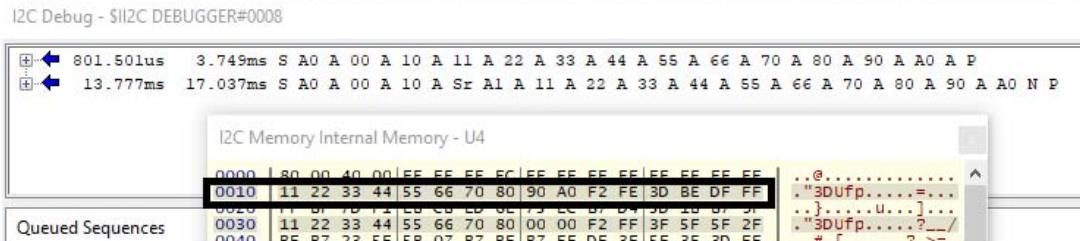
        i2c_start();
        i2c_write(0xA1);
        for(i=0;i<10;i++)
        {
            if(i==9)rx_data[i]=i2c_read(0); //unsigned char i2c_read(unsigned char
            ack);
            else rx_data[i]=i2c_read(1);
    }
}
```

```

        }
        i2c_stop();
    }

    while (1);
}

```



شکل 10-16: خروجی اجرای برنامه در I2C Debugger

## 10.9 ارتباط ریزپردازنده با یک ریزپردازنده دیگر

در ارتباط دو ریزپردازنده از طریق I<sup>2</sup>C، امکان استفاده از توابع i2c.h در مواقعي که Slave یک ریزپردازنده است وجود ندارد. از طرفی برنامه‌ای که توسط توابع twi.h برای Slave توسعه داده شده، گاهی اوقات سبب از دست رفتن ارتباط می‌شود. لذا جهت رفع مشکلات فوق، می‌توان برای Master و Slave فایل‌های سرآیند و کمکی جدید تعریف نمود. کدهای فایل کمکی برای Master در برنامه 10-3 نشان داده شده است.

لازم به ذکر است که اگر در سختافزاری دو ریزپردازنده وجود داشته باشد، بایستی به ازای هر یک از ریزپردازندها، فایل اجرایی مخصوص آن بارگذاری گردد و در تنظیم درگاه‌های ورودی و خروجی دقت کافی انجام شود تا به طور اشتباهی گذرگاه I<sup>2</sup>C در وضعیت صفر قرار نگیرد.

```

/*
 * I2C_Master_C_file.c
 *
 */

```

**برنامه 3-10**

```

#include "I2C_Master_H_file.h"

void I2C_Init()
{
    TWBR = BITRATE(TWSR = 0x00); /* Get bit rate register value by formula */
}

/*****************************************/
char I2C_Start(char write_address)
{
    char status;
    TWCR = (1<<TWSTA) | (1<<TWEN) | (1<<TWINT);
    /* Enable TWI, generate start condition and clear interrupt flag */
}

```



```

TWCR= (1<<TWSTO) | (1<<TWINT) | (1<<TWEN) ;
        /* Enable TWI, generate stop condition and clear interrupt flag */
while(TWCR & (1<<TWSTO));           /* Wait until stop condition execution */
}
/*********************************************/

void I2C_Start_Wait(char write_address)
{
char status;
while(1)
{
TWCR = (1<<TWSTA) | (1<<TWEN) | (1<<TWINT) ;
        /* Enable TWI, generate start condition and clear interrupt flag */
while (!(TWCR & (1<<TWINT)));
        /* Wait until TWI finish its current job (start condition) */
status = TWSR & 0xF8; /*Read TWI status register with masking lower three bits*/
if (status != 0x08)
        /*Check weather start condition transmitted successfully or not? */
continue;                /* If no then continue with start loop again */
TWDR = write_address;    /* If yes then write SLA+W in TWI data register */
TWCR = (1<<TWEN) | (1<<TWINT);      /* Enable TWI and clear interrupt flag */
while (!(TWCR & (1<<TWINT)));
        /* Wait until TWI finish its current job (Write operation) */
status = TWSR & 0xF8; /*Read TWI status register with masking lower three bits*/
if(status != 0x18 ) /* Check weather SLA+W transmitted & ack received or not? */
{
I2C_Stop();            /* If not then generate stop condition */
continue;              /* continue with start loop again */
}
break;                  /* If yes then break loop */
}
}
/*********************************************/

char I2C_Write(char data)
{
char status;                      /* Declare variable */
TWDR = data;                     /* Copy data in TWI data register */
TWCR = (1<<TWEN) | (1<<TWINT); /* Enable TWI and clear interrupt flag */
while (!(TWCR & (1<<TWINT)));
        /* Wait until TWI finish its current job (Write operation) */
status = TWSR & 0xF8; /*Read TWI status register with masking lower 3 bits */
if (status == 0x28) /* Check weather data transmitted & ack received or not?
*/
return 0;                         /* If yes then return 0 to indicate ack received */
if (status == 0x30) /*Check weather data transmitted & nack received or not?
*/
return 1;                         /* If yes then return 1 to indicate nack received */
else
return 2;                         /* Else return 2 to indicate data transmission failed */
}
/*********************************************/

char I2C_Read_Ack()
{
TWCR= (1<<TWEN) | (1<<TWINT) | (1<<TWEA);
        /* Enable TWI, generation of ack and clear interrupt flag */
while (!(TWCR & (1<<TWINT)));
        /* Wait until TWI finish its current job (read operation) */
return TWDR;                      /* Return received data */
}
/*********************************************/

```

```

char I2C_Read_Nack()
{
    TWCR= (1<<TWEN) | (1<<TWINT);           /* Enable TWI and clear interrupt flag */
    while (! (TWCR & (1<<TWINT)));
                                /* Wait until TWI finish its current job (read operation) */
    return TWDR;                      /* Return received data */
}

```

کدهای فایل سرآیند Master نیز در برنامه 4-10 نشان داده شده است.

**برنامه 4-10**

```

/*
 * I2C_Master_H_file.h
 *
 */

#ifndef I2C_MASTER_H_FILE_H_
#define I2C_MASTER_H_FILE_H_


#define F_CPU 8000000UL
#include <mega16.h>
#include <delay.h>
#include <math.h>
#define SCL_CLK 100000L          /* Define SCL clock frequency */
#define BITRATE (TWSR) ((F_CPU/SCL_CLK)-
16)/(2*pow(4, (TWSR&((1<<TWPS0) | (1<<TWPS1)))))           /* Define bit rate */



void I2C_Init();
char I2C_Start(char write_address);
char I2C_Repeated_Start(char read_address);
void I2C_Stop();
void I2C_Start_Wait(char write_address);
char I2C_Write(char data);
char I2C_Read_Ack();
char I2C_Read_Nack();


#endif

```

کدهای فایل کمکی Slave در برنامه 5-10 نشان داده شده است.

**برنامه 5-10**

```

/*
 * I2C_Slave_C_File.c
 *
 */

#include "I2C_Slave_H_File.h"
/****************************************/
void I2C_Slave_Init(int slave_address)
{
    TWAR = slave_address;      /* Assign address in TWI address register */
    TWCR = (1<<TWEN) | (1<<TWEA) | (1<<TWINT);
                                /* Enable TWI, Enable ack generation, clear TWI interrupt */
}
/****************************************/

```

```

int I2C_Slave_Listen()
{
while(1)
{
int status;
while (!(TWCR & (1<<TWINT))); /* Wait to be addressed */
status = TWSR & 0xF8; /* Read TWI status register with masking lower three bits
*/
if (status == 0x60 || status == 0x68) /* Check weather own SLA+W received & ack returned (TWEA = 1) */
return 0; /* If yes then return 0 to indicate ack returned */

if (status == 0xA8 || status == 0xB0) /* Check weather own SLA+R received & ack returned (TWEA = 1)
*/
return 1; /* If yes then return 1 to indicate ack returned */
if (status == 0x70 || status == 0x78) /* Check weather general call received & ack returned (TWEA = 1) */

return 2; /* If yes then return 2 to indicate ack returned */
else
continue; /* Else continue */
}
} ****
*****



int I2C_Slave_Transmit(char data)
{
int status;
TWDR = data; /* Write data to TWDR to be transmitted */
TWCR = (1<<TWEN) | (1<<TWINT) | (1<<TWEA); /* Enable TWI and clear interrupt flag */
while (!(TWCR & (1<<TWINT))); /* Wait until TWI finish its current job (Write operation) */
status = TWSR & 0xF8; /* Read TWI status register with masking lower three bits
*/
if (status == 0xA0) /* Check weather STOP/REPEATED START received */
{
TWCR |= (1<<TWINT); /* If yes then clear interrupt flag & return -1 */
return -1;
}
if (status == 0xB8) /* Check weather data transmitted & ack received */
return 0; /* If yes then return 0 */
if (status == 0xC0) /* Check weather data transmitted & nack received */
{
TWCR |= (1<<TWINT); /* If yes then clear interrupt flag & return -2 */
}
return -2;
}
if (status == 0xC8) /*If last data byte transmitted with ack received TWEA = 0 */
return -3; /* If yes then return -3 */
else /* else return -4 */
return -4;
} ****
*****



char I2C_Slave_Receive()
{

```

```

int status;
TWCR= (1<<TWEN) | (1<<TWEA) | (1<<TWINT);
/* Enable TWI, generation of ack and clear interrupt flag */

while (! (TWCR & (1<<TWINT)));
/* Wait until TWI finish its current job (read operation)
 */

status = TWSR & 0xF8;
/* Read TWI status register with masking lower three bits */

if (status == 0x80 || status == 0x90)
/* Check weather data received & ack returned (TWEA = 1) */

return TWDR; /* If yes then return received data */

if (status == 0x88 || status == 0x98)
/* Check weather data received, nack returned and switched to not addressed slave
mode */

return TWDR; /* If yes then return received data */
if (status == 0xA0) /* Check weather STOP/REPEATED START received */
{
TWCR |= (1<<TWINT); /* If yes then clear interrupt flag & return 0
*/
return -1;
}
else
return -2; /* Else return 1 */
}

```

کدهای فایل سرآیند Slave هم در برنامه 6-10 نشان داده شده است.

6-10 برنامه #ifndef I2C\_SLAVE\_H\_FILE\_H\_
#define I2C\_SLAVE\_H\_FILE\_H\_

#include <mega16.h>

void I2C\_Slave\_Init(int slave\_address);
int I2C\_Slave\_Listen();
int I2C\_Slave\_Transmit(char data);
char I2C\_Slave\_Receive();

#endif

در برنامه نوشته شده یکی از ریزپردازنده ها Master و دیگری Slave است که Master فرآیند ارسال و دریافت داده را شروع می نماید. ریزپردازندهی Slave نیز همواره باید نسبت به گذرگاه هوشیار باشد تا اگر آدرس خود را دریافت کرد، طبق خواستهی Master روند ارسال و دریافت داده را انجام دهد. داده های ارسالی و دریافتی، به صورت همزمان روی UART نیز ارسال می گردند که در شکل 10-17 نشان داده شده است. Master تعدادی داده شامل اعداد 0 تا 9 را برای Slave می نویسد و در مقابل اعداد 20 تا 29 را از Slave دریافت می نماید.

```
I2C Debug - $I2C DEBUGGER#0008

[+] 1.255ms 10.028 s S 20 A 00 A 01 A 02 A 03 A 04 A 05 A 06 A 07 A 08 A 09 A Sr 21 A 14 A 15 A 16 A 17 A 18 A 19 A 1A A 1B A 1C A 1D N P
[+] 10.028 s 20.055 s S 20 A 00 A 01 A 02 A 03 A 04 A 05 A 06 A 07 A 08 A 09 A Sr 21 A 14 A 15 A 16 A 17 A 18 A 19 A 1A A 1B A 1C A 1D N P
[+] 20.055 s 20.561 s S 20 A 00 A 01 A

Virtual Terminal

Master Device
Sending : 0 1 2 3 4 5 6 7 8 9
Receiving : 20 21 22 23 24 25 26 27 28 29
Sending : 0 1 2 3 4 5 6 7 8 9
Receiving : 20 21 22 23 24 25 26 27 28 29
Sending : 0 1

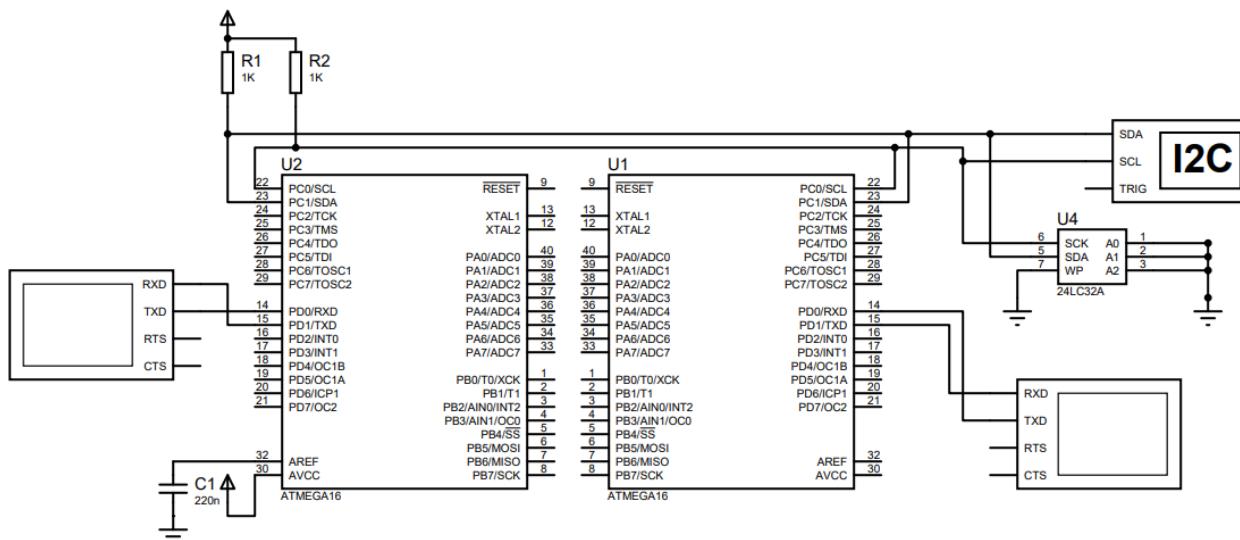
Virtual Terminal

Slave Device
Receiving : 0 1 2 3 4 5 6 7 8 9 255
Sending : 20 21 22 23 24 25 26 27 28 29
Receiving : 0 1 2 3 4 5 6 7 8 9 255
Sending : 20 21 22 23 24 25 26 27 28 29
Receiving : 0 1
```

شکل 10-17: اجرای برنامه Master/Slave

### 10.10 برنامه‌های اجرایی مبحث I<sup>2</sup>C

سخت‌افزار شکل 10-18 را که از دو ریزپردازنده، یکی برای Master و دیگری برای Slave تشکیل شده است در نظر بگیرید.

شکل 10-18: نمایی از سخت‌افزار مبحث I<sup>2</sup>C

- 1 - تعدادی داده را برای EEPROM ارسال نموده و مجدد همان داده ها را بخوانید. اگر داده‌های ارسالی و دریافتی مطابقت داشتند، پیغامی روی UART نمایش داده شود. (راهنمایی: از هریک از فایل‌های سرآیند i2c.h و twi.h می‌توانید استفاده نمایید).

-2 Master پیامی را برای Slave می‌فرستد و پیام‌های مبادله شده روی UART نمایش داده می‌شود.

(راهنمایی: از فایل‌های جانبی ارائه شده در دستورکار برای Master و Slave استفاده نمایید.)

پروژه‌ی کد ویژن شامل تمام فایل‌ها برای ریزپردازنده‌های Slave و Master را برای هریک از بندها بنویسید و از فایل‌های کمکی نیز استفاده نمایید. سپس در محیط پروتئوس برنامه را شبیه‌سازی نموده و پروژه نهایی را ارسال نمایید.