10 جلسه دهم
ارتباط سریال دو سیمه (I^2C یا TWI)

10.1 هدف

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

10.2 مقدمه

ارتباط سریال دو سیمه Inter-Integrated Circuit یا به اختصار I^2C ، 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^2C می‌تواند تا 128 وسیله مختلف را از طریق یک ارتباط اتصال‌گرا به همراه مکانیزم تصدیق به یکدیگر متصل کند. هرکدام از این تجهیزات متصل‌شده 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^2C می‌تواند حداکثر از 1008 دستگاه Slave در مد آدرس‌دهی ده بیتی 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^2C برای هر 8 بیت داده، یک بیت اضافه به نام بیت ACK/NACK اختصاص می دهد.

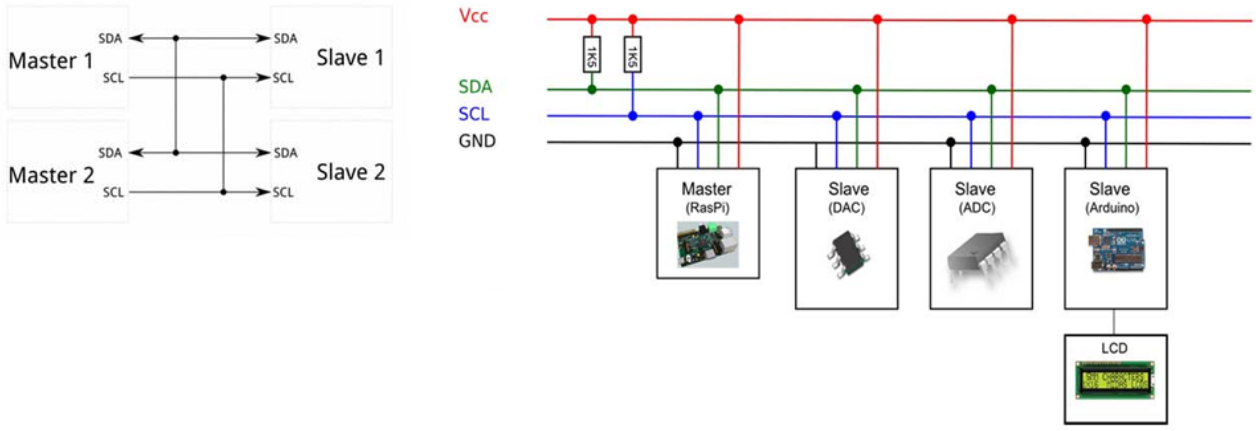
گذرگاه I^2C از دو سیگنال SCL برای انتقال پالس ساعت و SDA برای تبادل داده تشکیل گردیده که هر کدام به از آن‌ها به یک مقاومت بالاکش متصل شده است. پالس ساعت گذرگاه نیز همواره توسط دستگاه Master تولید می‌شود.

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 هنگامی‌که 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^2C وجود دارد. به I^2C که فرآیندهای آن به صورت سخت‌افزاری و بر روی پایه‌های اختصاصی انجام می‌شود، TWI نیز می‌گویند.

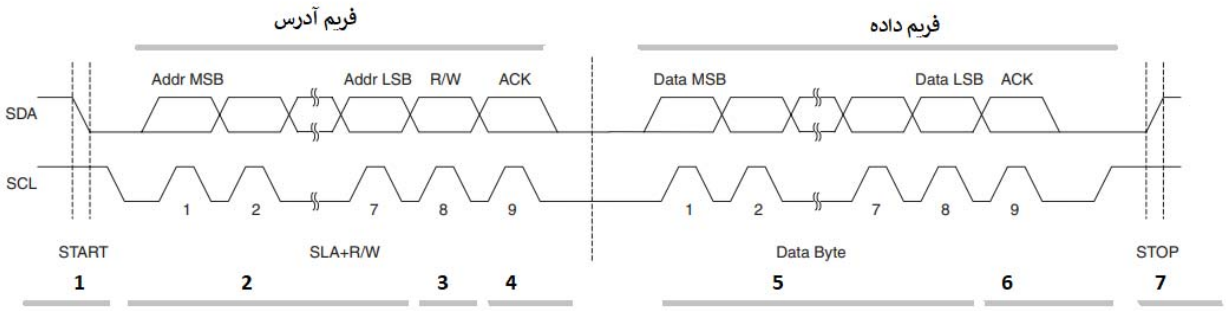
10.3 پروتکل I^2C

در حالت کلی مطابق شکل 10-1، تعدادی دستگاه Master و Slave به گذرگاه متصل می‌شوند که هر یک از Slaveها دارای آدرس مشخصی است. هنگامی‌که 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 غیر فعال می‌شوند. به این صورت ارتباط Master و Slave برقرار می‌شود.



شکل 10-1: نمایی از ارتباط I2C

جزییات 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^2C

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): Master آدرس Slave 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^2C 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.

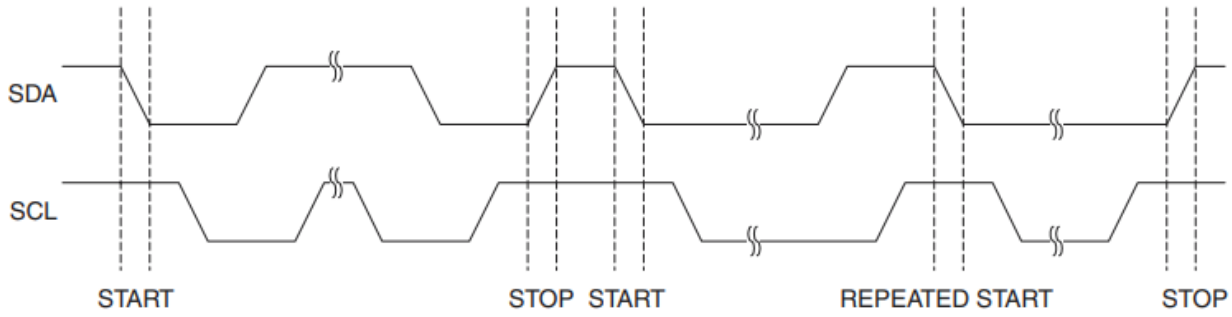
6- بسته‌های داده نیز مشابه بسته آدرس دارای یک بیت ACK هستند که اگر داده‌ی ارسالی از Master با موفقیت در Slave دریافت شد، ACK توسط slave ارسال می‌گردد. اگر Slave نتواند داده را به درستی دریافت نماید، خط SDA را در حالت بالا رها می‌کند (NACK). همچنین هنگامی که master آخرین بایت را از Slave دریافت می‌کند یا به هر دلیلی نتواند بایت دیگری را دریافت کند، یک NACK ارسال می نماید.

7- هنگامی‌که تمامی بسته‌های داده از جانب Master ارسال شدند و یا Master تمام داده های مورد نظر از جانب Slave را دریافت نمود، Master یک وضعیت توقف ایجاد می‌کند که نشان‌دهنده پایان ارتباط با Slave است و در مقابل Slave 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، هنگامی‌که خط SCL بالاست، نباید مقدار SDA تغیر کند تا از بروز وضعیت توقف اشتباهی جلوگیری شود.

در I^2C همگی دستگاه‌های Master و Slave توانایی ارسال و دریافت داده را دارند.. هر دستگاه متصل به گذرگاه قابلیت قرار گرفتن در وضعیت Master/Salve را دارد. و 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 Master Transmitter، Master Receiver، Slave Transmitter و Slave Receiver.

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 سایرMaster ها 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 هنگامی‌که 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^2C هر تعداد شروع مکرر مجاز است و فرایند ارسال داده تا زمانی که Master با ارسال وضعیت توقف کنترل گذرگاه را رها نماید می‌تواند ادامه یابد.

10.5 ثبات‌های I^2C

ثبات TWBR (TWI Bit Rate register): 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 SCL را تولید می‌کند.

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

شکل 10-4: ساختار ثبات TWBR

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

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

ثبات TWCR (TWI Control Register):

این ثبات مانند شکل 10-5 برای کنترل TWI یا I^2C به کار می‌رود.

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	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 تبادل 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 TWIرا 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 مادامی که پرچم TWINT یک باشد فعال می‌گردد.

ثبات TWSR (TWI Status Register):

این ثبات مانند شکل 6-10 برای نمایش وضعیت TWI یا I^2C به کار می‌رود.

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

شکل 6-10: 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^2C کدی در این بخش قرار می‌گیرد که به شرایط خاصی از پروسه ارسال و دریافت اشاره می‌نماید. جزییات بیشتر در راهنمای تراشه آمده است.

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

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

ثبات TWDR (TWI Data Register):

در حالت ارسال، ثبات TWDR بایت بعدی که باید ارسال شود را در خود نگه می‌دارد و در حالت دریافت آخرین 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	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

شکل 10-7: ساختار ثبات TWDR

ثبات TWAR (TWI (Slave) Address Register):

این ثبات مطابق ساختار نشان داده شده 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	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

شکل 10-8: ساختار ثبات TWAR

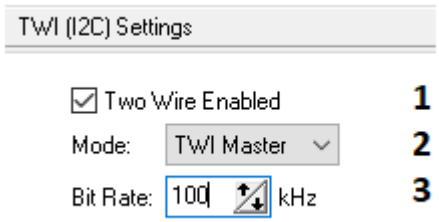
بیت‌های 7 تا 1-TWA (TWI Slave Address Register): این 7 بیت آدرس SLAVE را درخود نگاه می‌دارند.

بیت صفر-TWGCE (TWI General Call Recognition Enable Bit): 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.

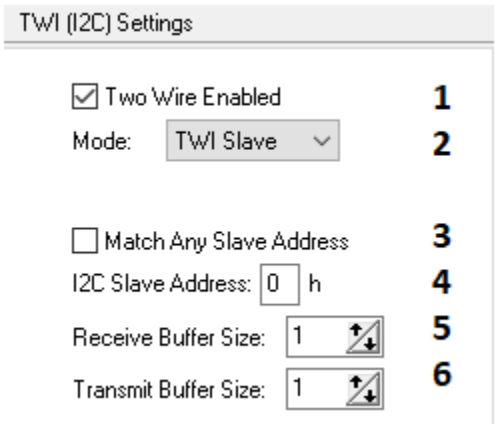
10.6 فعال‌سازی I^2C از طریق Codevision

برای فعال سازی I^2C در محیط نرم‌افزار CodeVision می‌توان از تنظیمات نشان داده شده درشکل 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 10-9: تنظیمات I^2C در محیط CodeWizard

10.7 توابع I^2C یا TWI

در Codevision دو کتابخانه برای کار با توابع I^2C فراهم شده است:

- کتابخانه twi.h که برای راه‌اندازی I^2C سخت‌افزاری یا TWI به کار می‌رود.
- کتابخانه i2c.h که برای راه اندازی I^2C 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 هرکدام از پین‌های ریزپردازنده تنظیم نمود.

10.7.1 فایل سرآیند twi.h

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^2C در حالت Master به کار می‌رود.

twi_master_trans که برای تبادل داده با Slave استفاده می‌شود.

twi_slave_initکه برای پیکربندی I^2C در حالت Slave استفاده می‌شود و تبادل داده هم با همین دستور انجام می گردد.

نکته : در صورت استفاده از این کتابخانه حتماً باید بیت وقفه عمومی را فعال کرد.

10.7.2 فایل سرآیند i2c.h

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^2C استفاده نمود. توابع این فایل سرآیند عبارتند از:

تابع i2c_init که تنظیمات اولیه I^2C 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.

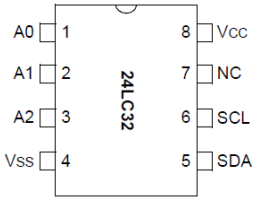
تابع i2c_write یک پارامتر ورودی از نوع char دارد که مقدار آن را بر روی رابط I^2C ارسال می‌کند.

تابع i2c_read یک بایت را از رابط I^2C 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.

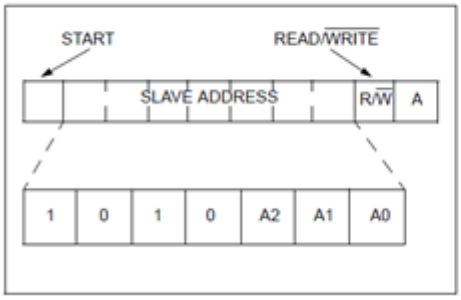
10.8 ارتباط ریزپردازنده با حافظه EEPROM

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 I^2C،SPI، Micro wire 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^2C استفاده می‌کند بررسی می‌شود.



شکل 10-10: پایه‌های تراشه حافظه 24LC32

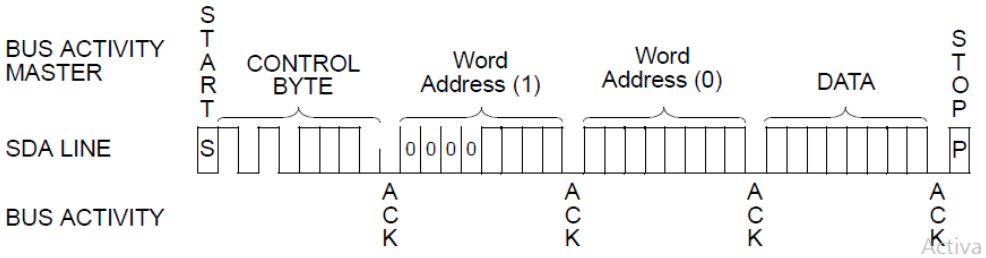
آدرس Slave برای این تراشه به صورت سخت‌افزاری با اتصال پایه‌های A0 تا A2 به ولتاژ زمین یا Vcc مطابق شکل 10-11 تنظیم می‌گردد. به عنوان مثال اگر همه این پایه‌ها به زمین متصل شوند، آدرس Slave برابر با 0x50 خواهد بود. بر اساس این نحوه آدرس‌دهی حداکثر 8 تراشه‌ی حافظه از این مدل را می‌توان به گذرگاه I^2C متصل نمود.



Operation	Control Code	Device Select	R/W
Read	1010	Device Address	1
Write	1010	Device Address	0

شکل 10-11: آدرس Slave برای حافظه EEPROM

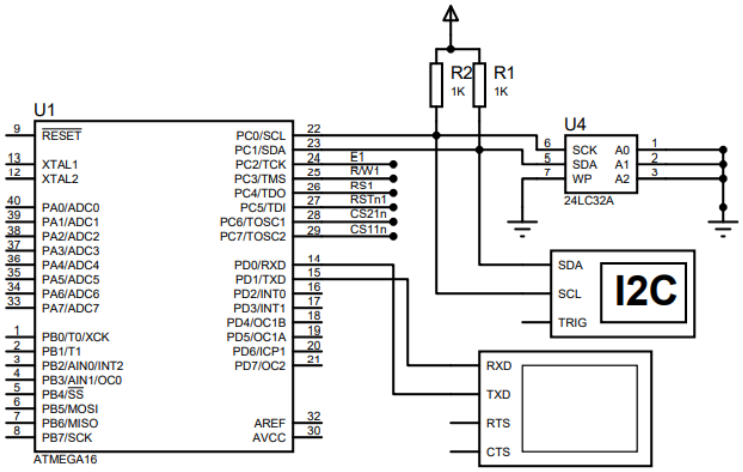
در حالت کلی مانند شکل 10-12، پردازنده به عنوان Master پس از ارسال آدرس حافظه Slave مورد نظر (Control byte)، آدرس فضایی که قرار است عملیات خواندن یا نوشتن در آن انجام شود را ارسال نموده و سپس داده‌ها مبادله می‌شوند. حجم این حافظه 32KB است و فضای آدرس آن در محدوده‌ی 0x0000 تا 0x03FF قرار دارد. بنابراین آدرس داده درون حافظه با دو بایت نشان داده می‌شود.



شکل 10-12: بسته ارتباطی حافظه EEPROM با پروتکل I^2C

10.8.1ارتباط ریزپردازنده با حافظه EEPROM از طریق توابع twi.h

برای ارتباط ریزپردازنده با حافظه، سخت‌افزار شکل 10-13 را در نظر بگیرید. در این سخت افزار بلوکی به نام "I2C" است که اصطلاحا I^2C Debugger نامیده می‌شود و در هنگام اجرای برنامه تمام اطلاعاتی که روی گذرگاه I^2C ارسال می شود را نشان می‌دهد و راهنمای بسیار خوبی برای درک پوتکل I^2C است.



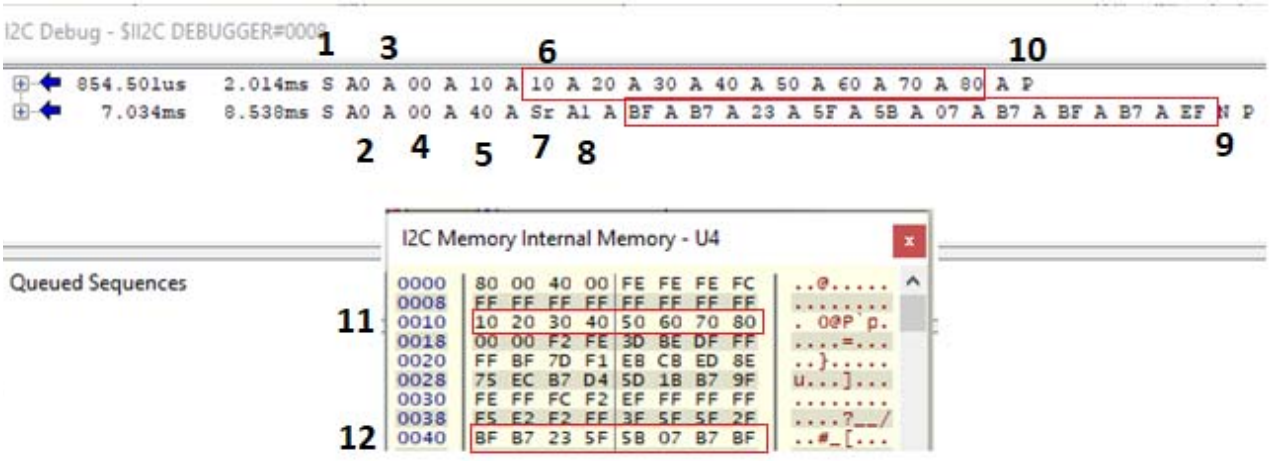
شکل 10-13: نمایی از سخت‌افزار مبحث I^2C برای ارتباط ریزپردازنده و حافظه EEPROM

برای ارتباط با حافظه از طریق I^2C و توابع موجود در فایل سرآیند twi.h می‌توان از برنامه 10-1 استفاده نمود. این برنامه در حافظه EEPROM با آدرسSalve برابر با 0x50، داده‌هایی را در آدرس0x0010 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.

برنامه 1-10

```
#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 *rxp=&rx_data[0];

twi_master_init(100);
#asm("sei")

 twi_master_trans(0x50, txp ,10,rxp,0);   // for wite to eeprom
 delay_ms(5);
 tx_data[0]=0x00;
 tx_data[1]=0x40;
 twi_master_trans(0x50, txp ,2,rxp,10);   //for read from eeprom
```

```
        while (1);
        }
```

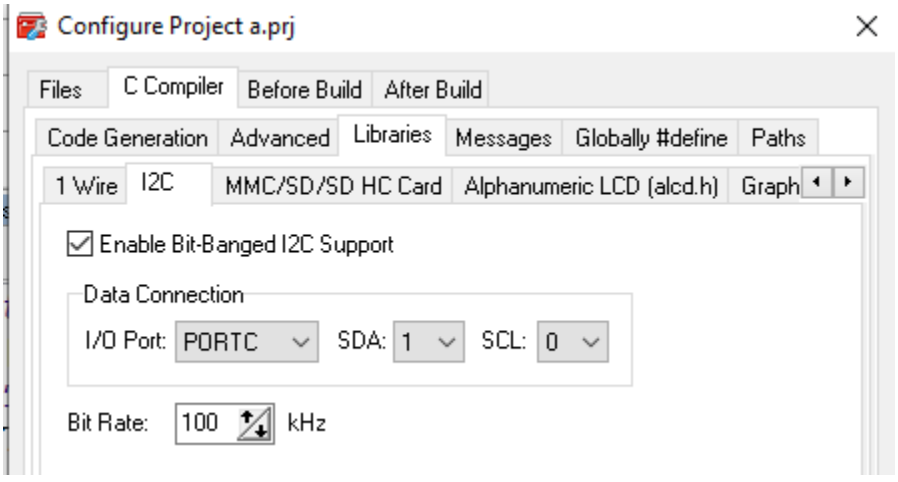


1: شروع
2: آدرس Slave برای نوشتن - 0 |(1<<0x50)=0xA0
3: پیام ACK
4: بایت با ارزش آدرس مکانی درون حافظه
5: بایت کم ارزش آدرس مکانی درون حافظه
6: داده‌ها
7: شروع مجدد
8: آدرس Slave برای خواندن - 1 |(1<<0x50)=0xA1
9. not ACK
10: پایان
11: فضای حافظه از آدرس 0x0010
12: فضای حافظه از آدرس 0x0040

شکل 10-14: خروجی اجرای برنامه در I2C Debugger

10.8.2ارتباط ریزپردازنده با حافظه EEPROM از طریق توابع i2c.h

برای ارتباط با حافظه، با I^2C 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.



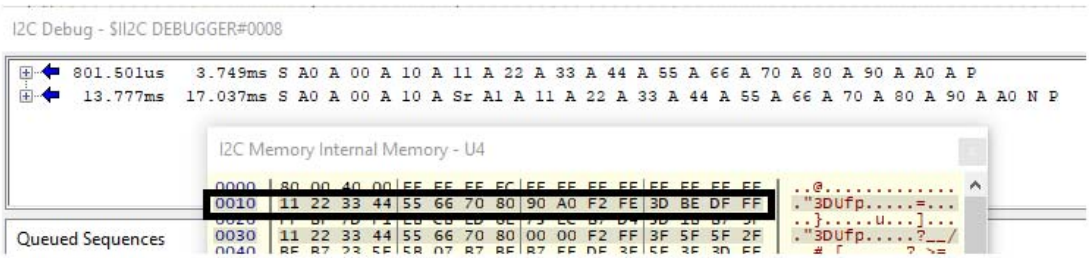
شکل 10-15 : تنظیمات لازم برای استفاده از i2c.h در پروژه

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.

برنامه 10-2

```
#include <mega16.h>
#include <delay.h>
#include <i2c.h>

void main(void)
{

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 ریزپردازنده‌ی دیگر

در ارتباط دو ریزپردازنده از طریق I^2C، امکان استفاده از توابع i2c.h در مواقعی 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 ریزپردازنده‌ها فایل اجرایی مخصوص آن بارگذاری گردد و در تنظیم درگاه‌های ورودی و خروجی دقت کافی انجام شود تا به طور اشتباهی گذرگاه I2C را در وضعیت صفر قرار ندهد.

برنامه 3-10

```
/*
 * I2C_Master_C_file.c
 *
 */


#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 */
```

```
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? */

return 0;             /* If not then return 0 to indicate start condition fail */
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 3 bits */
if (status == 0x18)   /*Check weather SLA+W transmitted & ack received or not? */

return 1;
/*If yes then return 1 to indicate ack received i.e. ready to accept data byte */

if (status == 0x20) /* Check weather SLA+W transmitted & nack received or not? */
return 2;/* If yes then return 2 to indicate nack received i.e. device is busy */
else
return 3;                           /* Else return 3 to indicate SLA+W failed */
}
/************************************************************************/


char I2C_Repeated_Start(char read_address)
{
char status;
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 3 bits */

if (status != 0x10)
    /* Check weather repeated start condition transmitted successfully or not? */

return 0;     /* If no then return 0 to indicate repeated start condition fail */
TWDR = read_address;           /* If yes then write SLA+R 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 == 0x40) /* Check weather SLA+R transmitted & ack received or not? */
return 1;
                              /* If yes then return 1 to indicate ack received */
if (status == 0x20)  /*Check weather SLA+R transmitted & nack received or not? */
return 2; /*If yes then return 2 to indicate nack received i.e. device is busy */
else
return 3;                           /* Else return 3 to indicate SLA+R failed */
}
/************************************************************************/


void I2C_Stop()
{
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 دربرنامه 10-4 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 10-5 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 */
}
```

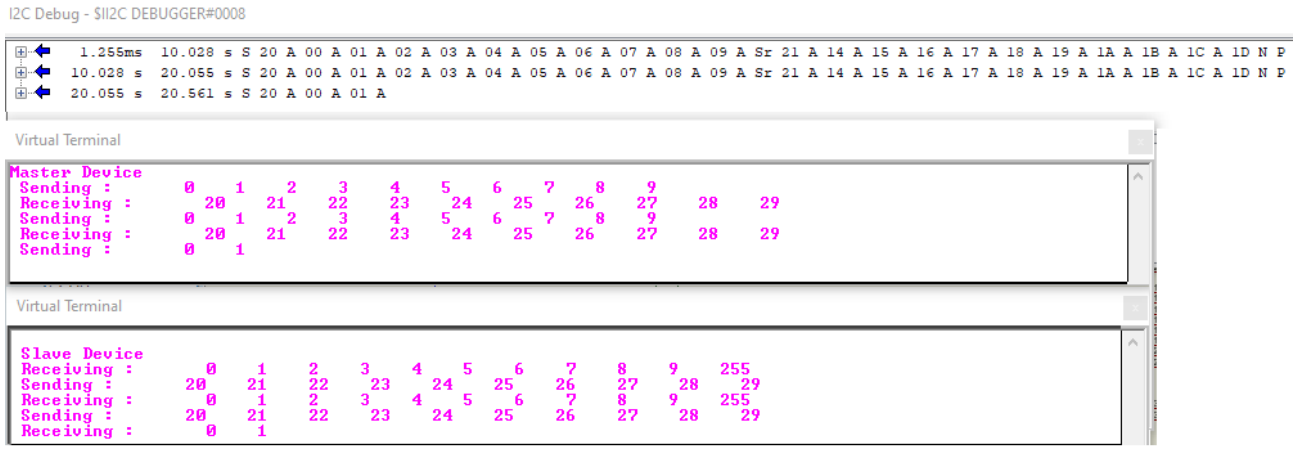
کدهای فایل سرآیند Slave هم در برنامه 10-6 نشان داده شده است.

برنامه 10-6

```
#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 می‌نماید. ریزپردازنده‌ی Slave نیز همواره باید نسبت به گذرگاه هوشیار باشد تا اگر آدرس خود را

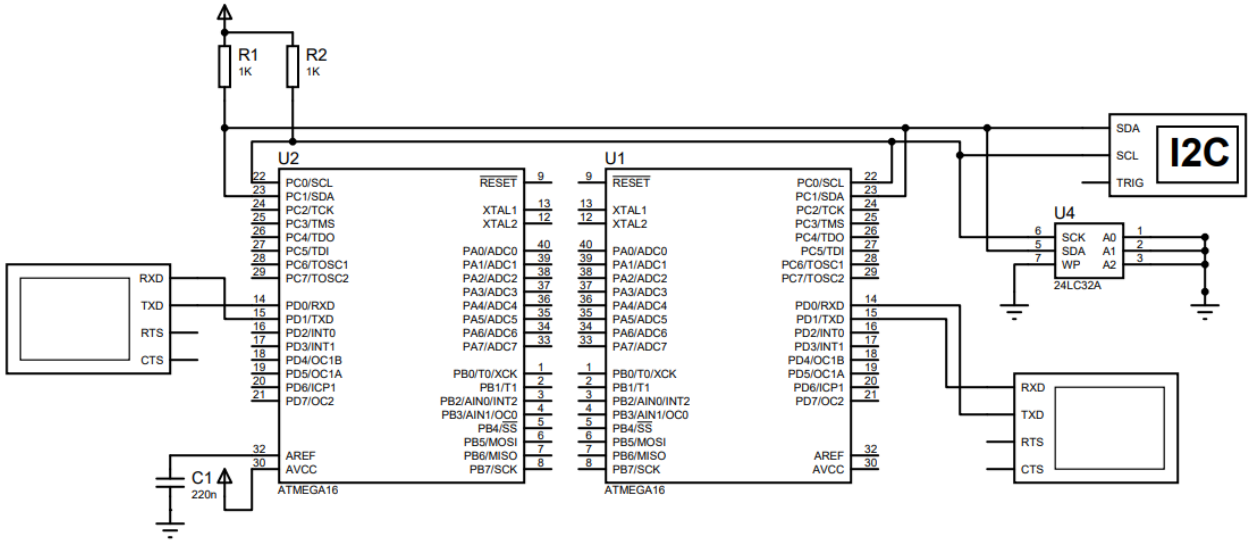
دریافت کرد، طبق خواسته‌ی Master روند ارسال و دریافت داده را انجام دهد. داده های ارسالی روی UART هم ارسال می گردد که در شکل 10-17نشان داده شده است. Master تعدادی داده شامل اعداد 0 تا 9 را برایSlave می‌نویسد و در مقابل اعداد 20تا 29 را از Slave دریافت می‌نماید.



شکل 10-17: اجرای برنامه Master/Slave

10.10 برنامه‌های اجرایی مبحث I^2C

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^2C

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 twi.h و i2c.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.