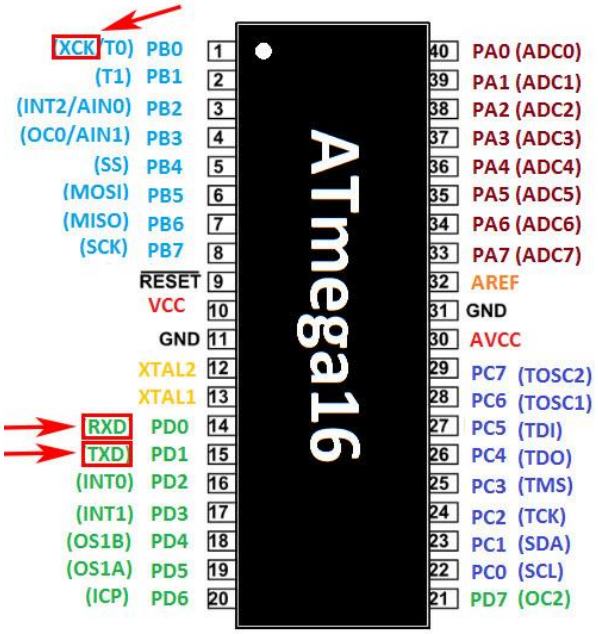
رابط سریال USART

ارتباط سریال به نوعی از ارتباط گفته می‌شود که بیت به بیت اطلاعات به صورت سری از یک رشته سیم منتقل می‌شود. چون ارسال تمام بیت‌ها بصورت پشت سرهم و با فاصله زمانی مشخص است، سرعت این نوع تبادل نسب به ارتباط Parallel یا موازی کم‌تر می‌باشد. همانطور که گفتیم USART نوعی ارتباط سریال است که اول حروف **U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter بوده و به معنای فرستنده گیرنده سنکرون و آسنکرون جهانی است. در میکروکنترلرهای AVR سه پایه برای سریال USART تعبیه شده که این سه پایه مطابق شکل زیر عبارت‌اند از: TXD ، RXD و XCK .



TXD : پایه Transmit یا ارسال داده است و باید به پایه RXD دستگاه مقابل متصل شود.

RXD : پایه Receive یا دریافت داده است و باید به پایه TXD دستگاه مقابل متصل شود.

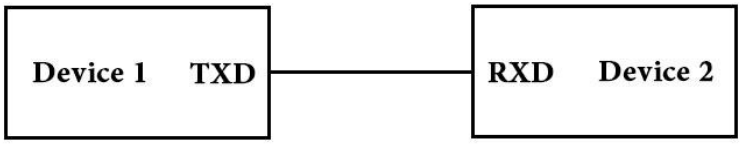
XCK : این پایه مربوط به زمانی است که ارتباط سریال ما سنکرون یا همزمان باشد. در این صورت این پایه به پایه XCK میکروکنترلری که میخواهیم با آن ارتباط برقرار کنیم متصل می‌شود.

نکته : پایه XCK جز استاندارد RS232 استاندارد جهانی ارتباط سریال UART نبوده و تنها در میکروکنترلر AVR وجود دارد.

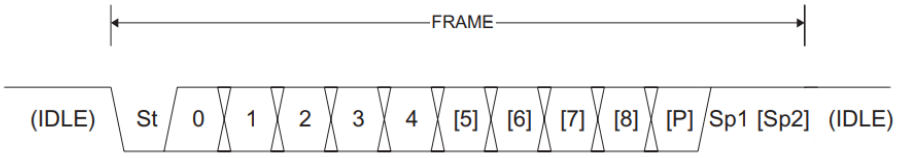
نکته : منظور از "دستگاه مقابل" یک میکروکنترلر، کامپیوتر، سنسور و یا هرچیزی است که دارای ارتباط USART می‌باشد.

شرح عملکردUSART

برای تبادل ارتباط میان دستگاه‌ها باید RXD ها به TXD ها متصل شوند. با توجه به شکل زیر پایه TXD مربوط به Device 1 به پایه RXD مربوط به Device 2 متصل است. باید گفت که در شکل زیر فقط می‌توان اطلاعات را از Device 1 به Device 2 انتقال داد و برای اینکه ارتباط دو طرفه شود باید TXD دستگاه ۲ را به RXDدستگاه ۱ متصل کنیم. اما برای شرح عملکرد سریال، فعلا ما ارتباط را بصورت یک طرفه در نظر می‌گیریم.



اگر Device 1 بخواهد اطلاعاتی را به Device 2 بفرستد، این اطلاعات در قالب Frame ارسال می‌شود که ساختار آن بصورت زیر است.



هر فریم از ۴ قسمت اصلی تشکیل شده است:

۱. بیت **St** (Start) که همواره ۰ منطقی است و در اول شروع فریم می‌آید.

۲. بیت‌های **داده** که حامل داده اصلی هستند و بین ۵ تا ۹ بیت می‌توانند باشند. اینکه تعداد بیت‌های داده چقدر باشد، توسط ما مشخص می‌شود.

۳. بیت **P** (Parity) که به آن بیت توازن هم گفته می‌شود. این بیت می‌تواند وجود داشته باشد یا اینکه اصلا مورد استفاده قرار نگیرد. در ادامه بیت توازن بصورت کامل گفته خواهد شد.

٤. بیت **Sp** (Stop) که همیشه ۱ منطقی است و حداقل ۱ بیت و حداکثر ۲ بیت می‌تواند باشد که این مورد هم توسط ما مشخص می‌شود. بیت Sp بر خلاف بیت St در آخر هر فریم می‌آید.

نکته ۱ :بیتStart ، بیت Parity (درصورت فعال سازی) و بیت Stop بصورت اتوماتیک و سخت افزاری به داده چسبانده می‌شوند و ما تنها باید داده‌هایی را که میخواهیم بفرستیم مشخص کنیم.

نکته ۲ :در حالت عادی (IDLE) اگر اطلاعاتی رد و بدل نشود، وضعیت خط انتقال در حالت ۱ منطقی قرار دارد. به همین خاطر است که بیت استارت بصورت همیشه ۰ ظاهر می‌شود تا آغاز یک فریم را مشخص کند.

نکته ۳ :سرعت انتقال داده یکی از فاکتورهای اصلی ارتباط USART است و به آن نرخ ارسال گویند و واحد آن بیت بر ثانیه (Bps) می‌باشد. مثلا نرخ ارسال ۹۶۰۰ یعنی تعداد ۹۶۰۰ بیت می‌تواند در ۱ ثانیه ارسال شود.

بیت توازن

دلیل وجود این بیت، تشخیص خطا در فریم است. بیت توازن دو نوع است:

۱. توازن زوج (Even Parity) که تعداد ۱ موجود در داده و بیت توازن باید زوج شوند.

۲. توازن فرد (Odd Parity) که تعداد ۱ موجود در داده و بیت توازن باید فرد شوند.

مقدار بیت توازن بستگی به داده‌ای که میخواهیم منتقل کنیم دارد. فرض کنید نوع توازن را زوج انتخاب کرده‌ایم و داده‌ای که می‌خواهیم ارسال شود ۸ بیتی بوده و بصورت ۱۱۰۰۱۰۰۱ باشد. با توجه به این داده، تعداد ۱ ها ۴ می‌باشد. پس چون ۴ عددی زوج است، بیت توازن ۰ منطقی می‌شود.

مثال ۱

سوال :اگر توازن زوج و داده ارسالی ۰۱۱۰۱۱۱۰ باشد، مقدار بیت توازن چقدر است؟

جواب :چون تعداد ۱ های داده برابر ۵ است، بیت توازن ۱ می‌شود تا تعداد ۱ های بیت‌های توازن و داده برابر ۶ شده و زوج شود.

مثال ۲

سوال :اگر توازن فرد و داده ارسالی ما ۱۰۱۰۰۰۰۱ باشد، مقدار بیت توازن چقدر است؟

جواب :چون تعداد ۱ های داده برابر ۳ است، بیت توازن باید ۰ باشد تا تعداد ۱ های بیت‌های توازن و داده، همان ۳ باقی بماند.

مزیت بودن بیت توازن

مثال ۱ را در نظر بگیرید. چون تعداد ۱ برابر ۶ می‌شود و **زوج** است، در طرف گیرنده هم باید همین تعداد ۱ شناسایی شود. اگر در طول مسیر یکی از بیت‌های ارسالی تغییر وضعیت دهد، تعداد ۱ ها فرد شده و گیرنده از این طریق متوجه رخ دادن خطا در داده می‌شود. خطای مربوط به بیت توازن با **PE**نشان داده می‌شود.

تذکر :نوع توازن بیتParity ، در هر دو طرف فرستنده و گیرنده، باید هر دو فرد و یا هر دو زوج باشد.

رجیسترهای USART درAVR

برای دسترسی به ارتباط سریال USART در تمامی میکروکنترلرهای AVR رجیسترهایی تعبیه شده‌اند که تمامی موارد فوق را پیاده‌سازی می‌کنند. مواردی مثل توازن فرد یا زوج، تعداد بیت‌های داده و نرخ ارسال همگی از طریق این رجیسترها تنظیم می‌شوند. ۷ رجیستر برای ارتباط USART وجود دارد که عبارت اند از UCSRB : ، UCSRA، UBRRH، UBRRLو UCSRC و ۲ رجیستر با اسم یکسان.UDR

رجیسترهای UBRRL و UBRRH برای تنظیم نرخ ارسال داده و رجیسترهای UCSRA تا UCSRC وظیفه کنترل و تنظیم نوع عملکرد رابط سریال را بر عهده دارند. همچنین دو رجیستر که نام هر دوی آنها UDR است، یکی برای ارسال داده و دیگری برای دریافت داده خواهد بود.

رجیسترهای UBRRL و (UBRRH (Usart Baud Rate Register

این دو رجیستر برای تنظیم نرخ ارسال هستند که مقدار کم ارزش آن UBRRL و مقدار با ارزش آن UBRRH است.

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	

مقدار UBRR از ۱۲ بیت تشکیل شده که ۸ بیت کم ارزش آن در UBRRL و ۴ بیت با ارزش آن در UBRRH قرار دارد. نرخ ارسال با توجه به یکی از سه فرمول زیر محاسبه می‌شود.

فرمول اول

در اول توضیحات این مطلب گفتیم که USART در دو مد سنکرون و آسنکرون می‌تواند عمل کند. اگر USART در مد آسنکرون باشد و بیت U2X در رجیستر UCSRAبرابر ۰ باشد، از فرمول زیر برای تنظیم Baud Rate استفاده می‌کنیم.

$$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$$

Foscفرکانس کاری میکروکنترلر، UBRRمقدار رجسترهای UBRR که ۱۲ بیت هستند و BAUD که میزان نرخ ارسال یا همان Baud Rate است. اعداد ۱ و ۱۶ هم که ثابت هستند.

سوال :اگر فرکانس کاری میکروکنترلر ۸ MHzباشد و بخواهیم نرخ ارسال داده ۹۶۰۰ بیت بر ثانیه باشد، مقدار UBRRL و UBRRH را به دست آورید؟

جواب :با طرفین وسطین کردن معادله بالا، مقدار UBRR برابر ۵۱.۰۸ به دست می‌آید که با گرد کردن آن باید ۵۱ را درون UBRR بریزیم. پس با تبدیل ۵۱ به مقدار معادل باینری آن یعنی "000000110011" باید مقدار UBRRH برابر "۰۰۰۰" و مقدار UBRRL برابر "۰۰۱۱۰۰۱۱" شود.

فرمول دوم

اگر USART در مد آسنکرون باشد و بیت U2X در رجیستر UCSRA برابر ۱ شود، سرعت انتقال داده ۲ برابر خواهد شد. در این حالت باید از فرمول زیر استفاده کرد.

$$BAUD = \frac{f_{osc}}{8(UBRR + 1)}$$

تنها تفاوت این فرمول نصف شدن عدد ثابت ۱۶ و تبدیل به ۸ است. چون سرعت ۲ برابر شده است.

فرمول سوم

اگر مد کاری بصورت سنکرون انتخاب شود، بیت U2X تاثیری در نرخ ارسال ندارد و باید از رابطه زیر برای تنظیم Baud Rate استفاده کرد.

$$BAUD = \frac{f_{osc}}{2(UBRR + 1)}$$

این فرمول هم مشابه فرمول‌های ذکر شده است و تنها عدد ثابت ۲ در مخرج قرار می‌گیرد.

رجیسترهای UDR (Usart Data Register)

وقتی صحبت از ارسال و یا دریافت اطلاعات می‌شود، باید به این دو رجیستر توجه کرد. این دو رجیستر کاملا همنام هستند و یکی از آنها برای دریافت اطلاعات (Read)و دیگری برای ارسال اطلاعات (Write) است.

نکته :اگر تعداد بیت‌های داده را ۸ بیت انتخاب کنیم که اکثرا مواقع هم همینگونه است، اطلاعات دریافتی در قالب یک بایت رد و بدل می‌شوند.

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
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	

اگر یک بایت دریافت شود) یک بایت از طریق پایه RXD وارد شود(، با خواندنUDR ، در اصل مقدار رجیستری که برای دریافت داده است، خوانده می‌شود. به عبارتی دیگر تمایز بین این دو رجیستر را کامپایلر (CodeVision) انجام می‌دهد.

```
unsigned char A;
A = UDR;
```

اگر بخواهیم یک بایت را ارسال کنیم، باید بر روی UDR بنویسیم. بلافاصله پس از مقدار دادن بهUDR ، ارسال اطلاعات از پایه TXD میکروکنترلر آغاز می‌شود.

```
UDR = 0xAA;
```

به عنوان مثال، یک بایت با مقدار "۱۰۱۰۱۰۱۰ (0xAA) "بر روی UDR نوشتیم که ارسال این بایت بصورت سخت‌افزاری انجام می‌شود.

رجیستر UCSRA (Usart Control Status Register A)

رجیسترهای UCSR جهت کنترل و تعیین وضعیت رابط سریال عمل می‌کنند. اما رجیستر UCSRA بیشتر وضعیت‌ها و اخطارهای جاری رابط سریال را نشان می‌دهد.

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

بیتRXC

اگر یک بایت دریافت شود، این بیت ۱ می‌شود. در غیر این صورت ۰ است.

بیتTXC

پس از اتمام ارسال یک بایت این بیت ۱ می‌شود. در غیر این حالت ۰ است.

بیت UDRE (Usart Data Register Empty)

پس از اتمام ارسال یک بایت، مدت زمانی طول می‌کشد تا رجیستر UDR ارسالی خالی شود. پس از خالی شدنUDR ، این بیت ۱ شده و می‌توان بایت بعدی را برای ارسال روی آن نوشت.

نکته :اگر این بیت ۱ نشده باشد و بایت بعدی بلافاصله روی UDR نوشته شود، خطای سخت افزاری رخ می‌دهد.

بیت FE (Frame Error)

این بیت در گیرنده کاربرد دارد و زمانی ۱ می‌شود که در فریم دریافتی خطاهایی مثل یکسان نبودن نرخ انتقال یا عدم تشخیص بیت‌های Start و Stop رخ دهد. مثلا اگر نرخ ارسال فرستنده ۹۶۰۰ bpsو در گیرنده ۳۸۴۰۰ bpsتنظیم شده باشد، این بیت مداوم در گیرنده ۱ شده و ۱ باقی می‌ماند. از این طریق می‌توان وقوع خطا را متوجه شد.

بیت DOR (Data Over Run)

اگر در گیرنده یک بایت دریافت شود و برنامه‌نویس این بایت را نخواند و سپس یک بایت دیگر دریافت شود، در این صورت مقدار بایت قبلی از بین خواهد رفت و این بیت ۱ می‌شود.

بیت PE (Parity Error)

اگر مشکل بیت توازن رخ دهد که بیت ۱ می‌شود (این مشکل در اواسط آموزش مطرح شد).

بیتU2X

اگر این بیت ۱ شود، نرخ ارسال داده ۲ برابر خواهد شد. در مورد این بیت در قسمت رجیسترهای UBRR توضیح دادیم.

بیت MPCM (Multi Processor Communication Mode)

اگر این بیت را ۱ کنیم، ارتباط سریال وارد مد چند پردازنده‌ای می‌شود که در حوصله این مطلب نمی‌گنجد و سعی می‌کنیم در یک جلسه بصورت جداگانه عملکرد این مد را بررسی کنیم.

رجیستر UCSRB (Usart Control Status Register B)

این رجیستر بصورت شکل زیر است و جهت کنترل سخت افزاری رابط سریال به کار می‌رود.

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	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	

بیت RXCIE (RXC Interrupt Enable)

اگر این بیت ۱ شود، وقفه دریافت USART فعال شده و در صورت گرفتن یک بایت، وقفه رخ می‌دهد .شماره Vector این وقفه عدد ۱۲ است.

بیت TXCIE (TXC Interrupt Enable)

اگر این بیت را ۱ کنیم، در صورت کامل شدن ارسال یک بایت، وقفه رخ می‌دهد. شماره Vector این وقفه عدد ۱۴ است.

بیت UDRIE (Usart Data Register Empty Interrupt Enable)

در صورت ۱ کردن این بیت، اگر پس از ارسال بایت، رجیستر UDR نوشتنی خالی شده و آماده دریافت بایت بعدی شود، وقفه رخ می‌دهد. شماره Vector این وقفه عدد ۱۳ است.

نکته :برای رخ دادن وقفه به غیر از فعال کردن هر کدام از این سه بیت، باید بیت وقفه عمومی هم ۱ شود.

بیت RXEN (RX Enable)

برای حالت گیرندگی باید این بیت را ۱ کرد. در غیر این صورت نمی‌توانیم داده دریافت کنیم.

بیت TXEN (TX Enable)

برای حالت فرستندگی این بیت باید ۱ شود. در غیر این صورت نمی‌توان ارسال داده انجام داد.

بیت UCSZ2 (Usart Character Size 2)

بیت‌های UCSZ از ۳ بیت تشکیل شده که بیت سوم آن یعنی UCSZ2 در این رجیستر قرار دارد و دو UCSZ0 و UCSZ1 در رجیستر UCSRC واقع شده‌اند.

گفتیم که طول داده دریافتی و ارسالی می‌تواند ۵، ۶، ۷، ۸ و یا ۹ بیت باشد. این مقدار توسط این ۳ بیت مطابق شکل زیر تنظیم می‌شود.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

به عنوان مثال اگر بخواهیم طول داده‌های ارسالی یا دریافتی ۸ بیت باشد، وضعیت این ۳ بیت باید "۰۱۱" مقداردهی شود.

بیت RXB8) RX Bit 8)

در جدول بالا ☝ اگر طول داده بصورت ۹ بیتی تنظیم شود، این بیت نهمین بیت دریافتی است و باید قبل از رجیستر UDR خوانده شود. به عنوان یادآوری باید گفت که بیت‌های ۰ تا ۷ در رجیستر UDR فقط خواندنی قرار دارند.

بیت TXB8) TX Bit 8)

اگر طول داده را ۹ بیت تنظیم کنیم، این بیت نهمین بیت ارسالی است و باید قبل از نوشتن بر روی UDR نوشتنی، مقداردهی شود. باز هم یادآور می‌شویم که بیت‌های ۰ تا ۷ در رجیستر UDR نوشتنی قرار دارند.

رجیستر UCSRC) Usart Control Status Register C)

این رجیستر هم قسمت‌های دیگر رابط سریال را کنترل می‌کند.

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

بیت(URSEL (Usart Register Select

اگر به رجیستر UBRRH دقت کرده باشید، آخرین بیت این رجیستر دقیقا بیت URSEL است. چون رجیستر UCSRC و UBRRH در حافظه از آدرس مشترکی استفاده می‌کنند، این بیت تمایز بین این دو رجیستر را انجام می‌دهد. به عبارتی دیگر اگر بخواهیم بر روی رجیستر UCSRC مقداری بنویسیم باید این بیت را بصورت ۱ شده اعمال کنیم و از طرفی اگر بخواهیم مقداری را بر روی UBRRH بنویسیم باید این بیت را ۰ شده در نظر بگیریم.

مثال: با نوشتن کد زیر رجیستر UCSRC مقداردهی می‌شود؛ چون بیت URSEL مقدار ۱ داده شده است. `UCSRC = 0x86;`

اما در کد زیر برای مقداردهی رجیستر UBRRH باید بیت URSEL را ۰ مقداردهی کنیم. `UBRRH = 0x02;`

بیت(UMSEL (Usart Mode Select

رابط USART در دو مد سنکرون و آسنکرون کار می‌کند.

- اگر UMSEL برابر ۰ شود، مد آسنکرون انتخاب می‌شود.
- اگر UMSEL برابر ۱ شود، مد سنکرون فعال خواهد شد.

در اکثر موارد مد آسنکرون استفاده می‌شود. اما اگر مد سنکرون انتخاب شود، کافی است در هر دو میکرو، پایه‌های XCK به یکدیگر وصل شوند. در این صورت پایه XCKیکی از میکروها بصورت خروجی و پایه XCK میکرو دیگر، باید بصورت ورودی تعریف شود.

بیت‌های UPM0 و(UPM1 (Usart Parity Mode

در مورد بیت Parity صحبت کردیم. بیت توازن یا غیر فعال است و یا فعال بوده و در حالت زوج یا فرد قرار دارد.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- اگر این دو بیت "۰۰" شوند، بیت توازن غیرفعال بوده و در Frame ارسالی یا دریافتی وجود ندارد.
- حالت "۰۱" غیر مجاز است.
- حالت "۱۰" بیت توازن فعال شده و بصورت توازن زوج عمل می‌کند.
- حالت "۱۱" بیت توازن فعال شده و بصورت توازن فرد عمل می‌کند.

بیت(Usart Stop Bit Select) USBS

تعداد بیت‌های Stop را معلوم می‌کند.

- اگر ۰ باشد، یک بیت Stop در آخر فریم می‌آید.
- اگر ۱ باشد، دو بیت Stop استفاده می‌شود.

بیت‌های UCSZ0 و(UCSZ1 (Usart Character Size

این بیت‌ها در توضیحات رجیستر UCSRB شرح داده شدند.

بیت(Usart Clock Polarity) UCPOL

این بیت زمانی کاربرد دارد که رابط USART در مد سنکرون عمل کند. توضیحات بیشتر در مورد این بیت فعلا جایز نیست و در مثال‌ها به این بیت می‌پردازیم.

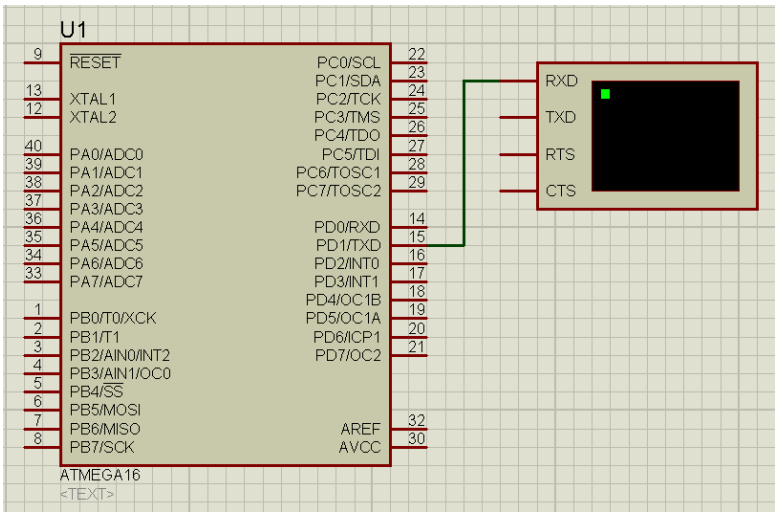
مثال ۱ : برنامه‌ای بنویسید که بصورت مداوم هر ۱ ثانیه حرف A را بر روی خروجی چاپ کند. بیت Parity غیرفعال، تعداد بیت‌های Stop یک و طول داده ۸ بیت است. همچنین نرخ ارسال هم ۹۶۰۰ bps می‌باشد.

تذکر : در تمام مثال‌ها فرکانس کاری میکروکنترلر بصورت ۸ MHz داخلی تنظیم شده است و مد عملکرد هم آسنکرون می‌باشد.

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

void main(void)
 {
//////////////////////USART Initialization
UBRRH = 0;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x08;
UCSRC = 0x86;

   while (1)
{
   //UDR = 65;
   UDR = 'A';
   delay_ms (1000);
   }
}
```



تنظیم UBRRH وUBRRL

اولین کار مشخص کردن نرخ ارسال یا Baud Rate است. چون در مد آسنکرون هستیم و بیت U2X واقع در رجیستر UCSRA صفر است، باید از فرمول اول استفاده کنیم .

$$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$$

با گذاشتن ۸۰۰۰۰۰۰ بجای Fosc و ۹۶۰۰ بجایBAUD ، مقدار UBRR برابر ۵۱ می‌شود. پس UBRRH برابر ۰ و UBRRL برابر ۵۱ خواهد شد.

تنظیمUCSRA

گفتیم که این رجیستر بیشتر وضعیت رابط را نشان می‌دهد. اما دو بیت مهم آن یعنی MPCM و U2X باید ۰ شوند. نوشتن ۰ یا ۱ بر روی U2X روی Baud Rate تاثیر می‌گذارد و اگر در این مثال آن را ۱ کنیم، مقدار Baud Rate دو برابر خواهد شد.

تنظیمUCSRB

از هیچ وقفه‌ای استفاده نمی‌کنیم. پس سه بیت TXCIE ، RXCIEو UDRIE باید ۰ شوند. تنها می‌خواهیم ارسال داده انجام دهیم. پس بیت RXEN برابر ۰ و بیت TXENبرابر ۱ می‌شود. چون طول داده ۸ بیتی است، وضعیت بیت‌های UCSZ باید "۰۱۱" شود. در نتیجه بیت UCSZ2 باید ۰ شود. در آخر هم به دلیل ۸ بیتی بودن طول داده، نیازی به دو بیت RXB8 و TXB8 نیست. بنابراین مقدار این رجیستر ۰ x08خواهد شد.

تنظیمUCSRC

گفتیم که برای نوشتن روی UCSRC باید بیت URSEL را هنگام مقداردهی ۱ کنیم. چون عملکرد در مد آسنکرون است، بیت UMSEL باید ۰ شود. بیت توازن نداریم پس UPM0 و UPM1 هر دو باید ۰ باشند. تعداد بیت Stop هم ۱ بیت است. در نتیجه USBS باید ۰ باشد. اما چون وضعیت بیت‌های UCSZ برابر "۰۱۱" شد، هر دو بیت UCSZ0 و UCSZ1 باید ۱ شوند. در آخر هم UCPOL که در مثال‌های بعدی در مورد آن توضیح می‌دهیم، باید ۰ مقداردهی گردد.

با این تفاسیر مقدار رجیستر UCSRC برابر ۰ x86می‌شود.

بدنهwhile

چون قرار است بصورت مداوم اطلاعات ارسال شود، دستورات را داخل While(1) می‌نویسیم. برای ارسال یک بایت، کافی است به رجیستر UDR مقدار دهیم. حرف Aدر داخل برنامه بصورت 'A' تفسیر می‌شود و این مقدار را به UDR می‌دهیم که بلافاصله پس از مقدارگیری ارسال اطلاعات آغاز شده و CPU به دستور Delay می‌رسد و این چرخه ادامه دارد.

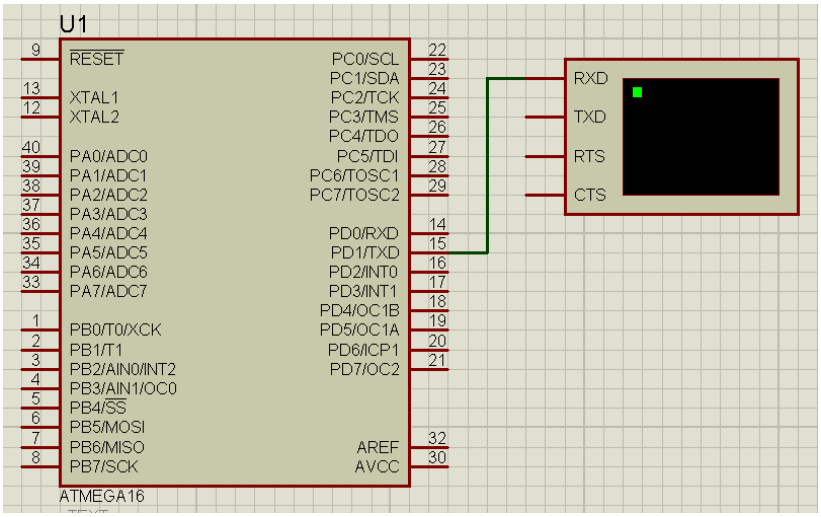
تذکر :به عنوان کامنت عبارت UDR = 65 را نوشته‌ایم. تمام حروف انگلیسی، اعداد و علامت‌ها در جدولی به نام جدول اسکی ذکر شده‌اند که هر حرف یا علامت از یک بایت منحصر به فرد برای نمایش استفاده می‌کند. این مقدار برای حرف A برابر ۶۵ است.

مثال ۲ : برنامه مثال ۱ را طوری ارتقا دهید که بجای حرف A عبارت Hello را چاپ کند. تمام تنظیمات یکسان است.

```
#include <mega16.h>
#include <delay.h>
unsigned char Array[] = {'H', 'e', 'l', 'l', 'o'};
unsigned char i;

void main(void){
//////////////////////USART Initialization
UBRRH = 0;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x08;
UCSRC = 0x86;

   while(1){
      for(i=0;i<5;i++){
      UDR = Array[i];
      while((UCSRA&0x20)==0x00);
      }
   delay_ms(1000);
   }
}
```

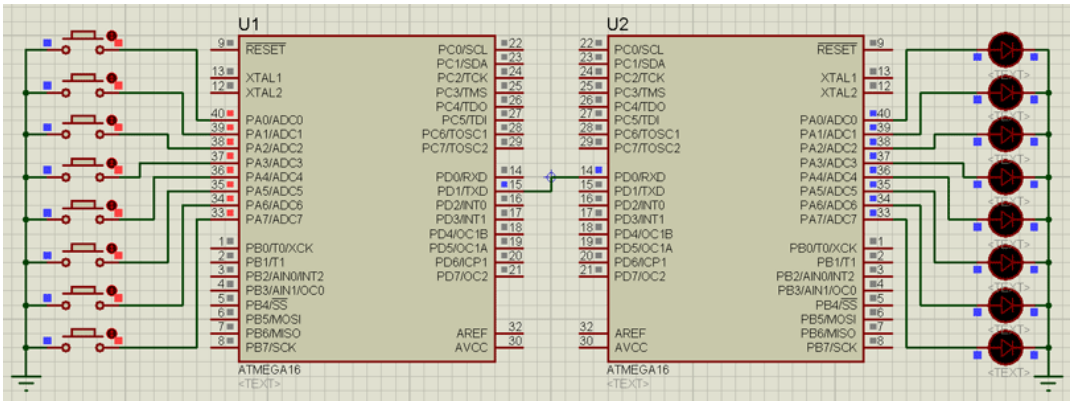


تعریف آرایه : چون عبارت Hello از ۵ حرف تشکیل شده است، بنابراین یک آرایه با طول ۵ بایت تعریف می‌کنیم که هر حرف یک بایت را اشغال کرده است.

حلقهFor : در حلقه For مقدار I با هر بار اجرا یک واحد زیاد می‌شود. مقدار اولیه متغیر I برابر ۰ است و تا عدد ۵ ادامه پیدا می‌کند. اما مقدار رجیستر UDR از عناصر آرایه و به ترتیب از حرف H تا حرف آخر ادامه می‌یابد.

پس از اینکه به UDR مقدار داده شد، باید منتظر بمانیم تا داده ارسال شده و رجیستر UDR خالی و آماده ارسال بایت بعدی شود. اینکار با دستور While((UCSRA&0x20)==0x00)انجام می‌شود. به عبارتی بیت UDRE واقع در رجیستر UCSRA وقتی رجیستر UDR خالی شود ۱ خواهد شد. پس اگر مقدار UCSRA را با ۰ x20بصورت منطقی &کنیم و حاصل را با ۰ x00مقایسه کنیم، در حالت عادی شرط صحیح بوده و CPU در دستور While صبر می‌کند. اما به محض ۱ شدن بیتUDRE ، شرط نادرست شده و CPU از While خارج می‌شود. بنابراین حلقه به انتها رسیده و از اول حلقه، رجیستر UDR مقدار بعدی را گرفته و این روال ادامه خواهد یافت.

مثال ۳ : برنامه‌ای بنویسید که در طرف فرستنده پورت A را هر ۱۰۰ میلی ثانیه بخواند و ارسال کند. همینطور در طرف گیرنده، اطلاعات را دریافت کند و بر روی پورت A نشان دهد. تنظیمات سریال مشابه دو مثال قبلی است



برنامه گیرنده

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

void main(void){
DDRA = 0xFF;
PORTA = 0x00;
//////////////////////USART Initialization
UBRRH = 0;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x10;
UCSRC = 0x86;

  while(1){
  while((UCSRA&0x80)==0x00);
  PORTA = UDR;
  }
}
```

در این برنامه پورت A بصورت خروجی عمل می‌کند. اما چون تنها می‌خواهیم دریافت اطلاعات داشته باشیم، باید بیت RXEN در رجیستر UCSRB را ۱ کنیم. پس مقدار UCSRB برابر 0x10 خواهد شد.

در بدنه این حلقه دستور While((UCSRA&0x80)==0x00) نوشته شده و تا زمانی که بیت RXC در رجیستر UCSRA برابر ۰ باشد، شرط آن صحیح بوده و CPU در این خط می‌ماند. پس از دریافت یک بایت، شرط آن نادرست شده و CPU به خط بعدی می‌رود. در این خط مقدار UDR خوانده شده و درون PORTA ریخته می‌شود.(خواندنی)

برنامه فرستنده

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

void main(void){
DDRA = 0x00;
PORTA = 0xFF;
//////////////////////USART Initialization
UBRRH = 0x00;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x08;
UCSRC = 0x86;

  while(1){
  UDR = ~PINA;
  while((UCSRA&0x20)==0x00);
  delay_ms(100);
  }
}
```

تنها نکته‌ای که این برنامه دارد این است که پورت A را ورودی کرده و مقاومت‌های Pullup آن فعال شده است. در حلقه While هم مقدار PINA خوانده شده و معکوس آن درون UDR ریخته می‌شود. چون در حالت عادی و بدون وصل کردن هیچ کدام از کلیدها، مقدار 0xFF خوانده می‌شود و باید معکوس آن که 0x00 است ارسال شود. بقیه خطوط مشابه مثال ۱ و ۲ است

مثال ۴ برنامه گیرنده مثال ۳ را طوری بازنویسی کنید که به وسیله وقفه اطلاعات را دریافت و بر روی پورت A نشان دهد.

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

void main(void){
DDRA = 0xFF;
PORTA = 0x00;
////////////////////USART Initialization
UBRRH = 0;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x90;
UCSRC = 0x86;
#asm ("sei")

while(1);
}

  interrupt[12]void Receive(void){
  PORTA = UDR;
  }
```

در برنامه فوق مقدار رجیستر UCSRB از 0x10 به 0x90 تغییر کرد. چون میخواهیم از وقفه دریافت استفاده کنیم، باید بیت RXCIE را ۱ کنیم. از طرفی هم باید بیت وقفه عمومی ۱ شود که با دستور #Asm این کار انجام شده است.

شماره Vector وقفه دریافت USART برابر ۱۲ بوده که در Interrupt مربوطه نوشته شده است. در دستورات وقفه، رجیستر UDR خوانده می‌شود و در PORTAقرار می‌گیرد.

کتابخانه stdio

در این کتابخانه توابعی وجود دارد که دریافت و ارسال اطلاعات بر روی رابط USART را انجام می‌دهند. البته در هر صورت باید تنظیمات رجیسترهای USART را انجام داد.

تابع putchar

این تابع یک کاراکتر را بر روی خروجی ارسال می‌کند.

```
putchar('یک کاراکتر');
```

تابع getchar

این تابع منتظر دریافت یک کاراکتر از ورودی می‌ماند و به محض دریافت، آن را در خروجی تابع ارسال می‌کند.

```
unsigned char A;
A = getchar();
```

تابع puts

این تابع یک آرایه را در خروجی ارسال می‌کند.

```
unsigned char Array[] = {'H', 'e', 'l', 'l', 'o'};
puts(Array);
```

در کد بالا آرایه ۵ بایتی است که دستور Putsبصورت پشت سرهم هرکدام را ارسال می‌کند.

تابع putsf

این تابع مانند تابع Puts است با این تفاوت که بجای آرایه یک رشته را در ورودی خود دریافت کرده و آن را ارسال می‌کند.

```
putsf("www.rasdino.ir");
```

نکته: این تابع بصورت اتوماتیک یک بایت با مقدار ۰ به انتهای رشته اضافه می‌کند. پس در طرف گیرنده تعداد بایت دریافتی باید به اندازه ۱ واحد بیشتر باشد.

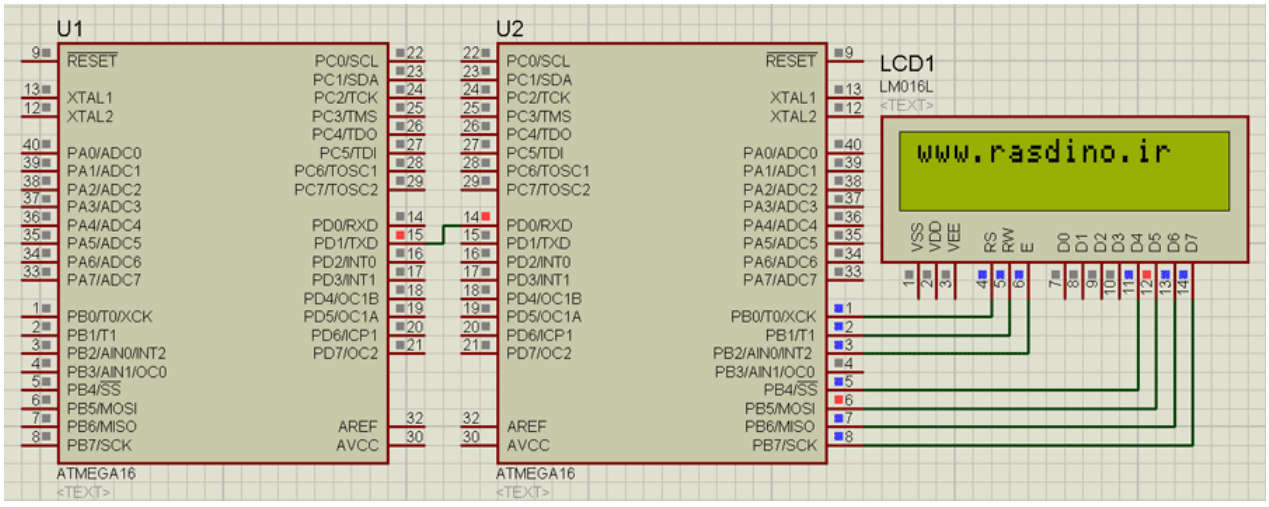
تابع gets

این تابع برای دریافت N بایت از ورودی استفاده می‌شود. دو ورودی می‌گیرد که اولی آرایه‌ای است که باید اطلاعات در آن ذخیره شود و دومی تعداد بایت دریافتی است.

```
putsf("www.rasdino.ir");
```

در کد بالا یک آرایه ۸ بایتی تعریف شده و در خط بعد ۸ بایت دریافت می‌شود و در درون Array ذخیره می‌گردد.

مثال ۵ : برنامه‌ای بنویسید که در فرستنده عبارات Www.Rasdino.Ir و Hello ارسال و در گیرنده پس از دریافت، بر روی Lcd کاراکتری نشان داده شوند



برنامه گیرنده	برنامه فرستنده
#include <mega16.h> #include <delay.h> #include <stdio.h> #include <alcd.h> unsigned char Array[17];	#include <mega16.h> #include <delay.h> #include <stdio.h> void main(void){

```
//////////////USART Initialization
UBRRH = 0x00;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x08;
UCSRC = 0x86;

  while(1){
  delay_ms(1000);
  putsf("www.rasdino.ir");
  delay_ms(1000);
  putsf("Hello");
  }
}
```

```
void main(void){
lcd_init(16);
lcd_clear();
//////////////////////USART Initialization
UBRRH = 0;
UBRRL = 51;
UCSRA = 0x00;
UCSRB = 0x10;
UCSRC = 0x86;

  while(1){
  ////////////////////www.rasdino.ir
  gets(Array,15);
  lcd_clear();
  lcd_gotoxy(0,0);
  lcd_puts(Array);
  ////////////////////Hello
  gets(Array,6);
  lcd_clear();
  lcd_gotoxy(0,0);
  lcd_puts(Array);
  }}
```

در کد بالا درون حلقه While عبارت‌های گفته شده با دستور Putsf با تاخیر ۱ ثانیه‌ای نسبت به هم ارسال می‌شوند

دستورات ابتدایی برنامه مشابه برنامه‌های قبل است. اما در حلقه While با استفاده از تابع Gets تعداد ۱۵ بایت دریافت می‌شود. سپس Lcd پاک شده و در خط و ستون اول، عبارت دریافت شده (Array) بر روی Lcd چاپ می‌شود. در ادامه ۶ بایت دریافت شده و همانند قبل نمایش داده می‌شود.

نکته‌ای که قبلا هم آن را ذکر کردیم این است که چون عبارت Www.Rasdino.Ir از ۱۴ بایت تشکیل شده است و تابع Putsf در فرستنده یک بایت به انتهای این عبارت اضافه می‌کند، باید ۱۵ بایت دریافت کنیم. همینطور برای عبارت Hello هم باید ۶ بایت دریافت شود.