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

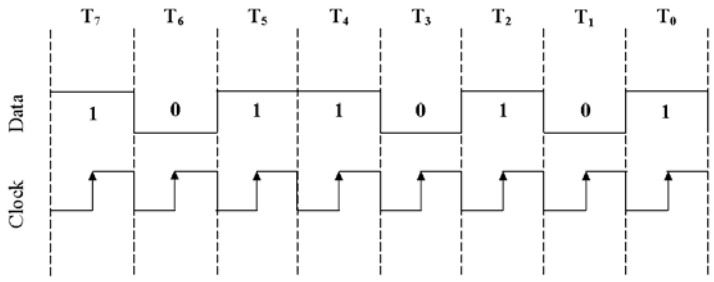
در سیستم‌های بزرگ و پیچیده دیجیتال که از اجزای متعدد تشکیل شده‌اند، لازم است به منظور ایجاد هماهنگی بین این بخش‌ها و اشتراک داده‌ها انتقال اطلاعات، یک بستر ارتباطی مناسب فراهم گردد. به صورت کلی انتقال اطلاعات در سیستم‌های دیجیتالی، به دو روش سریال و موازی صورت می‌گیرد. در روش موازی n بیت اطلاعات از طریق 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، USB، CAN، LonWorks[1]، SATA، I^2C و SPI 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.

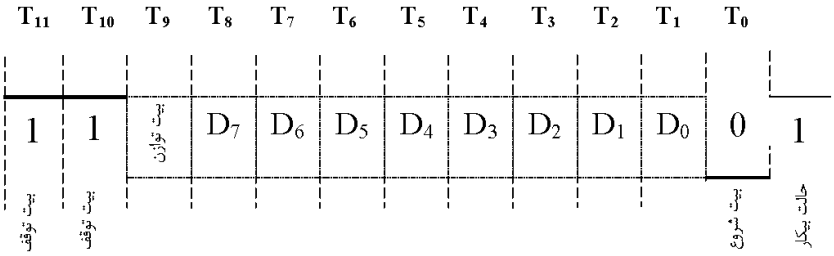
به طور کلی ارتباط سریال به دو صورت سنکرون و آسنکرون برقرار می‌شود. در ارتباط سنکرون مانند شکل 7-1، داده‌ها بر روی یک خط ارسال می‌شوند و یک خط پالس ساعت همزمان کننده نیز وجود دارد که به همراه داده‌ها برای سنکرون‌سازی مبدأ و مقصد، توسط فرستنده ارسال می‌شود.

[1] 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[1] و دو بیت توقف در ارتباط آسنکرون مشاهده می‌شود.



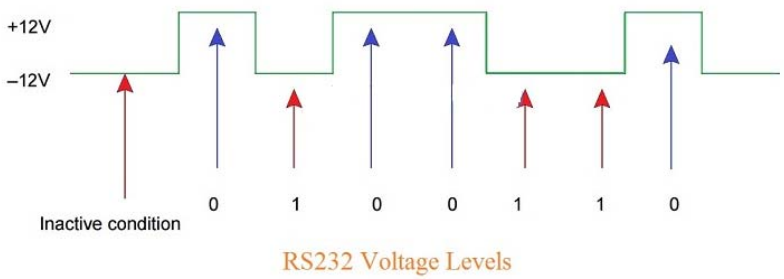
شکل 7-2: ارتباط به صورت آسنکرون

7.3 معرفی ارتباط USART (RS232)

یکی از پروتکل‌هایی که ریزپردازنده Atmega16/32 برای ارتباط سریال پشتیبانی می‌کند، پروتکل USART[2] 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-C است اما معمولاً به نام RS232 شناخته می‌شود و مخفف Recommended Serial می‌باشد. این استاندارد معمولاً در درگاه سریال کامپیوترهای شخصی و ماژول‌هایی مانند فرستنده/گیرنده RF، GPS و GSM جهت ایجاد امکان ارتباط با ریزپردازنده استفاده می‌شود.

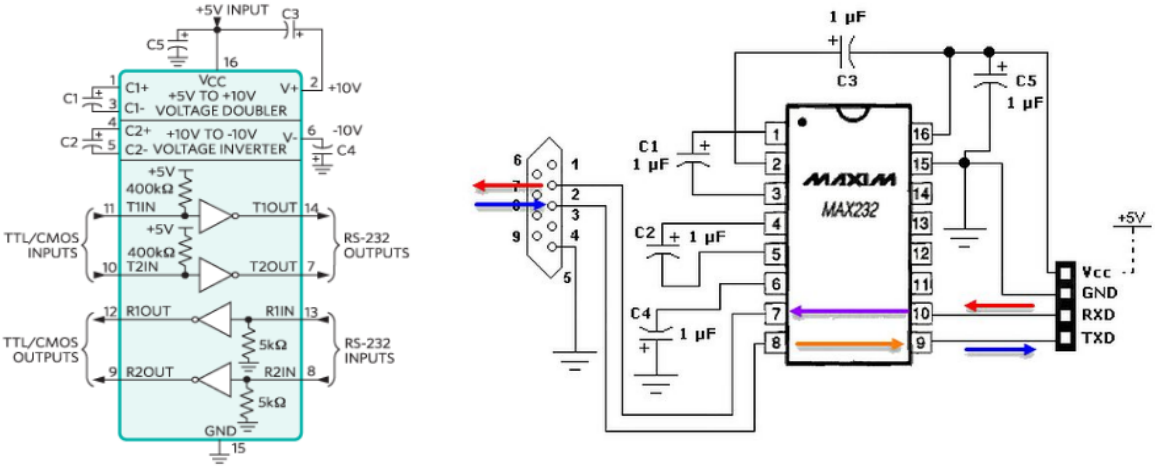
در استاندارد RS232 سطح ولتاژ 3+ تا 12+ نمایانگر وضعیت صفر منطقی و بازه‌ی 3- تا 12- ولت نمایشگر وضعیت یک منطقی می‌باشد. این در حالی است که تجهیزات مبتنی بر استاندارد TTL مثل ریزپردازنده Atmega16/32 در سطوح بین 0 و 5 ولت کار می‌کنند.

[1] parrity
[2] Universal Synchronous Asynchronous serial Receiver/Transmitter



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 7-4).



شکل 7-4: 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 وRX واقع در PD0 و PD1 برای دریافت و انتقال داده به صورت سریال مطابق با استاندارد RS232 است. و با تراشه MAX232 سطوح ولتاژ از 0 تا 5 به 12+ و 12V- به ازای صفر و یک منطقی منتقل می‌شود. همان طور که درشکل 7-4 نشان داده شده پایه TX به T2IN و پایه RX به R2OUT از تراشه MAX232 متصل می‌گردند. بعد از برقراری اتصال ریزپردازنده به MAX232، می‌توان این تراشه را به درگاه سریال کامپیوتر متصل نمود تا بستر ارتباط کامل گردد.

درگاه سریال در کامپیوترهای شخصی به نام‌های COM Port یا RS-232 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، تنظیم Baud Rate و غیره 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 پایه های کانکتور

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

7.5 ثبات‌های ارتباط USART

در ادامه به معرفی ثبات‌های مربوط به ارتباطUSART 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	
	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	

شکل 7-6: ساختار ثبات UDR

7.5.2 ثبات UCSRA[1]

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

[1] USART Control and Status Register

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	

شکل 7-7: ساختار ثبات UCSRA

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

بیت 6 - TXC (USART Transmit Complete): 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 - UDRE (USART Data Register Empty): در صورت یک بودن، ثبات فرستنده آماده دریافت داده‌ی جدید می‌باشد.

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

بیت 3 - DOR (Data Over Run): 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 - PE (Parity Error): اگر در کاراکتر بعدی موجود در بافر گیرنده خطای parity وجود داشته باشد، این بیت یک می‌شود.

بیت 1 - UDX (Double the USART transmission Speed): 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 - MPCM (Multi-Processor communication Mode): این بیت امکان ارتباط چند ریزپردازنده با یکدیگر را فراهم می‌کند.

7.5.3 ثبات UCSRB

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	
	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	

شکل 7-8: ساختار ثبات UCSRB

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.

7.5.4 ثبات UCSRC

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

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	

شکل 7-9: ساختار ثبات UCSRC

بیت7- URSEL (register select): این بیت دسترسی به ثبات UBRRH و UCSRC را مشخص می‌نماید. هنگام خواندن یا نوشتن 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 را 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 می شود. اگر1 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 UBRRL و UBRRH (USART Baud rate Register)

این ثبات مانند شکل 7-10 برای تنظیم نرخ ارسال داده استفاده می شود و ثبات UBRRH و ثبات UCSRC از یک مکان مشترک در حافظه I/O استفاده می‌کنند.

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	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

شکل 7-10: ساختار ثبات UBRR

بیت 15- URSEL (Register Select): این بیت برای انتخاب دسترسی به ثبات‌های UBRRH وUCSRC 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. لازم به ذکر است که در Atmega16/32 نرخ خطای کمتر از 0.5٪ قابل قبول است.

جدول 7-1: تنظیمات نرخ ارسال داده

Baud Rate (bps)	f_{osc} = 8.0000 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 (1)	0.5 Mbps		1 Mbps	

1. UBRR = 0, Error = 0.0%

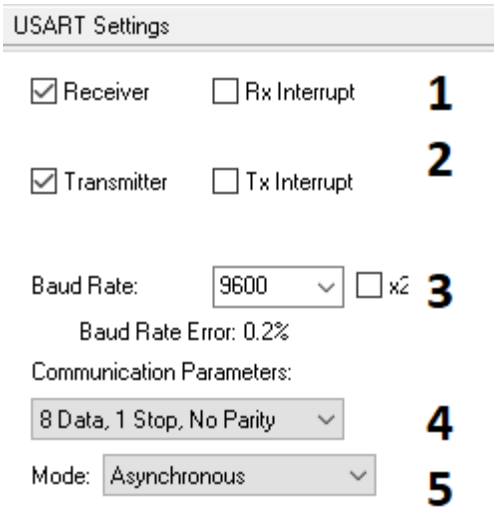
7.6 کتابخانه stdio.h

برای برقراری ارتباط با پایه‌های USART ریزپردازنده 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 getchar() و putchar(char c) 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 7-11 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.

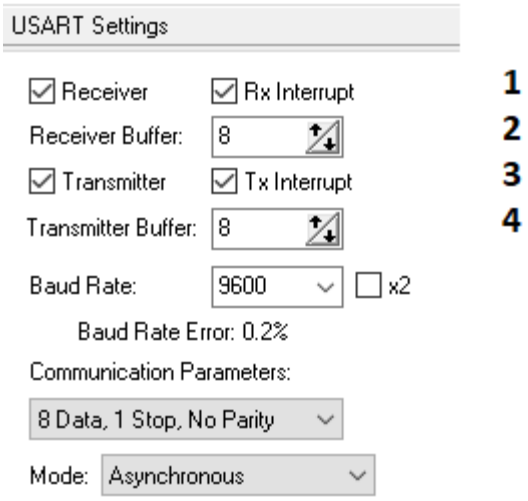
شکل 7-11: تنظیمات UART در CodeWizard

در تنظیم نرخ ارسال بایستی توجه داشت که برای کارکرد مناسب باید نرخ احتمال خطا کمتر از 0.5٪ 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 وقفه‌های بخش UART

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 درشکل 7-12 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 می-برد، ماهیت دستورات putchar و getchar نسبت به حالت سرکشی تغییر اساسی پیدا کرده و از یک سری بافر جهت نگه‌داری و مدیریت داده‌های دریافی و ارسالی خط UART استفاده می‌گردد. این تفاوت در انتهای همین بخش بیشتر بررسی خواهد شد.

برنامه 7-1

```
#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;
#if 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++];
#if 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

      }
}
```

7.7.2 وقفه‌ی RX

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.

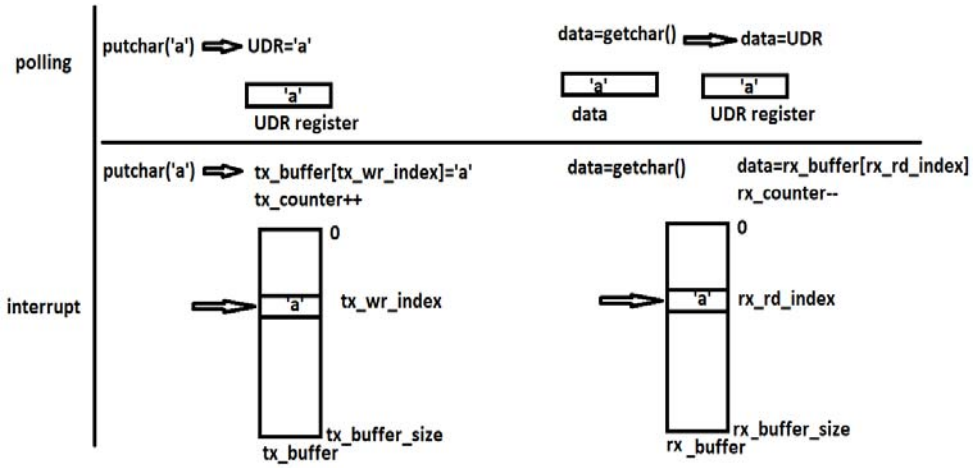
7.7.3 وقفه‌ی TX

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 getchar و putchar در حالت سرکشی و وقفه رخ می‌دهد درشکل 7-13 نشان داده شده است.

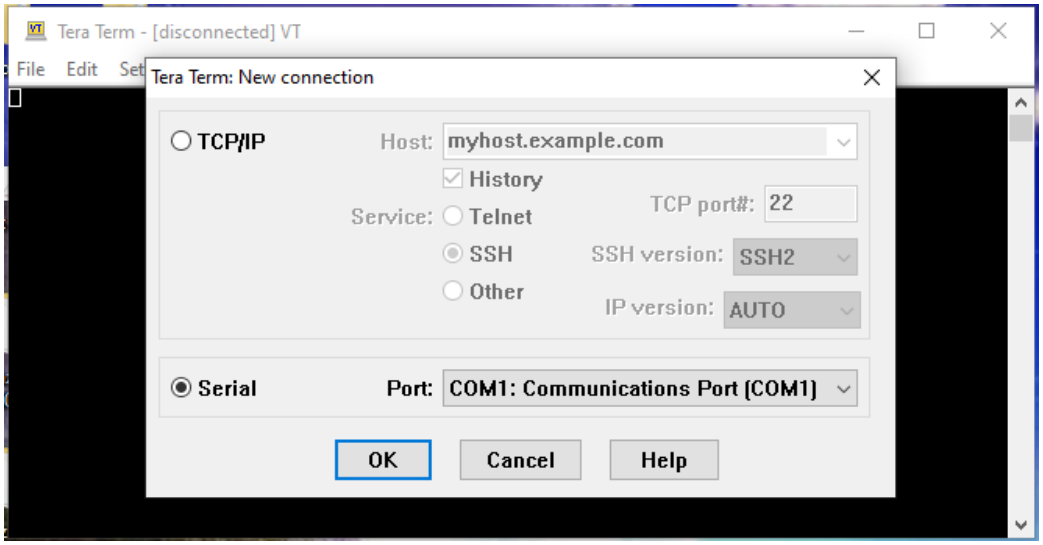


شکل 7-13: تفاوت ماهیت توابع putchar و getchar در حالت سرکشی و وقفه ارتباط UART

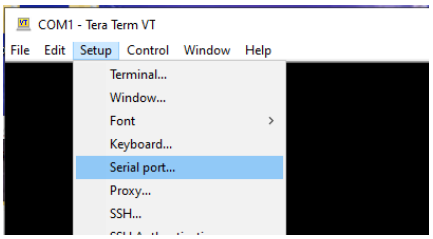
7.8 نرم افزار Teraterm

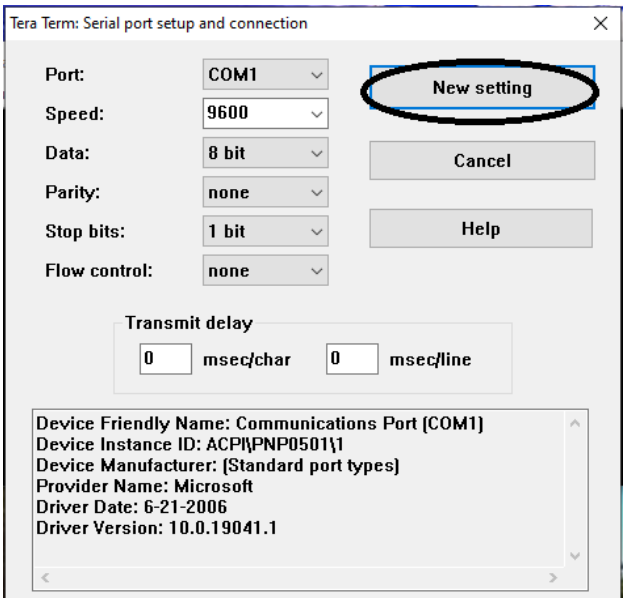
پس از برقراری ارتباط بین پایه‌های RX و TX ریزپردازنده و درگاه ارتباط سریال کامپیوتر مطابق آن‌چه در بخش‌های پیشین توضیح داده شد، اکنون می‌توان از نرم افزارهای کاربردی در کامپیوتر مانند Teraterm برای نمایش داده‌های دریافتی از سمت ریزپردازنده و هم‌چنین ارسال داده برای آن استفاده نمود. مراحل کار با این نرم‌افزار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

الف

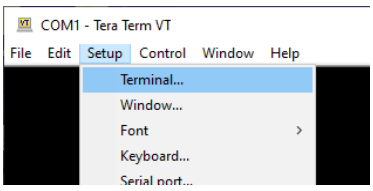


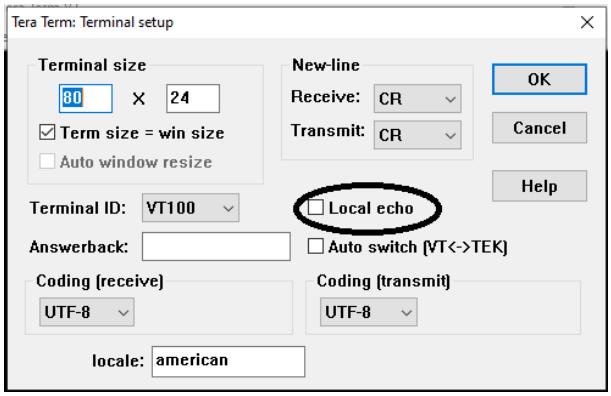
ب





ج



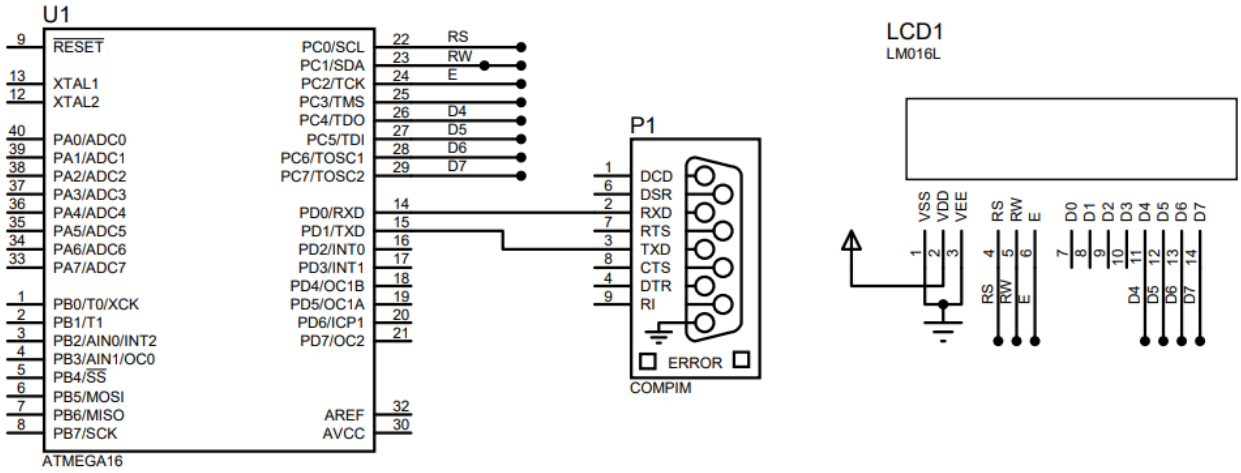


شکل 7-14: نمایی از نحوه کار با محیط Teraterm

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 7-2 و برنامه 7-3 آمده است. لازم به ذکر است که در برنامه CodeVision وقفه‌های UART فعال شده است.



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

برنامه 7-2

```
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
        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”

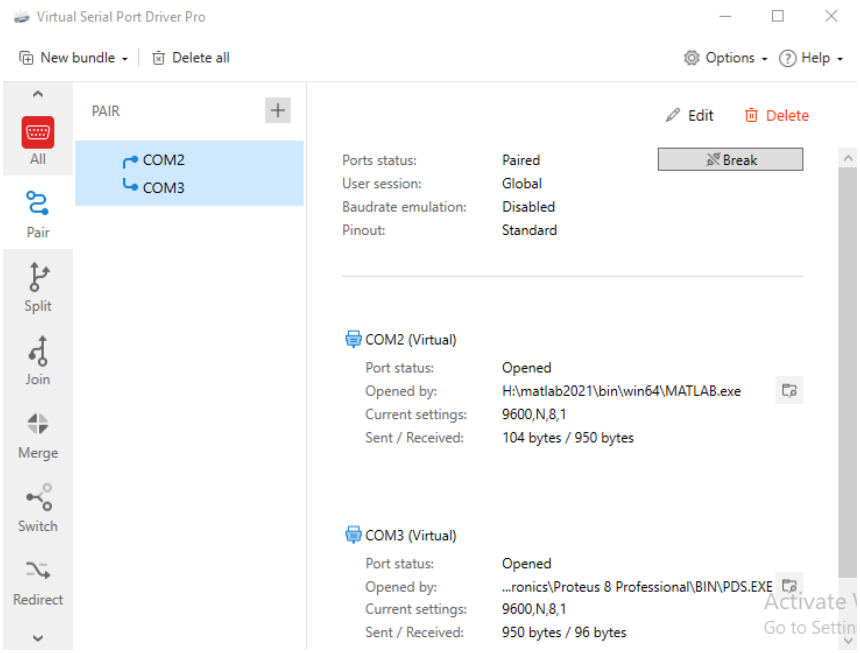
Select “add a new pair”

Select “Create”

……..

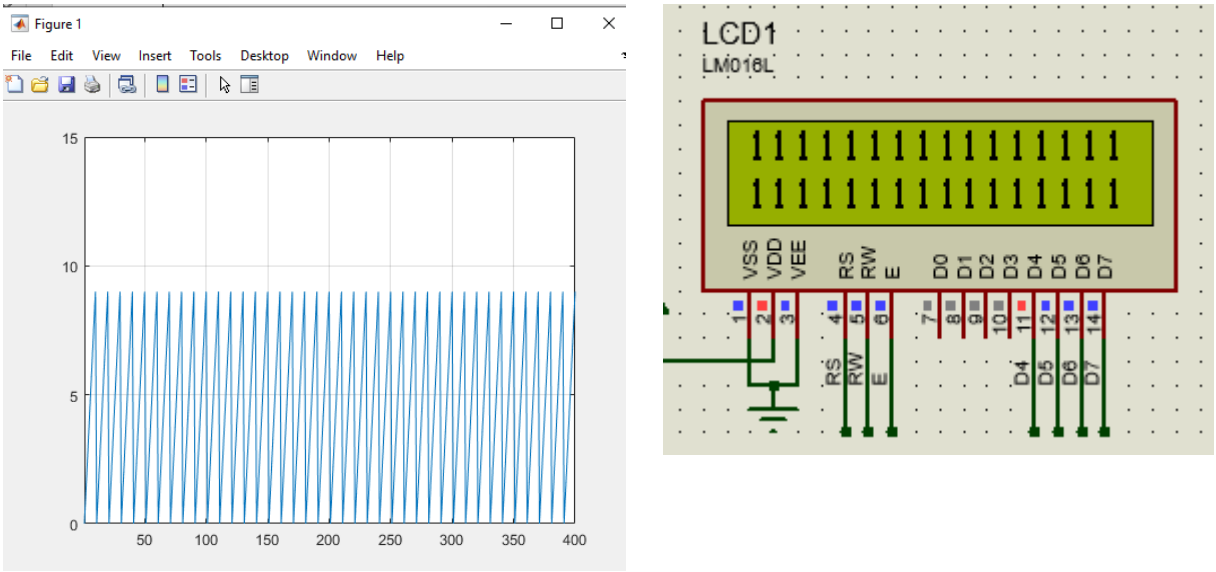
For end of connection : select “Delete”

On pair section



شکل 7-16: فعال نمودن درگاه مجازی

روند اجرای این برنامه درشکل 7-17 نشان داده شده است.

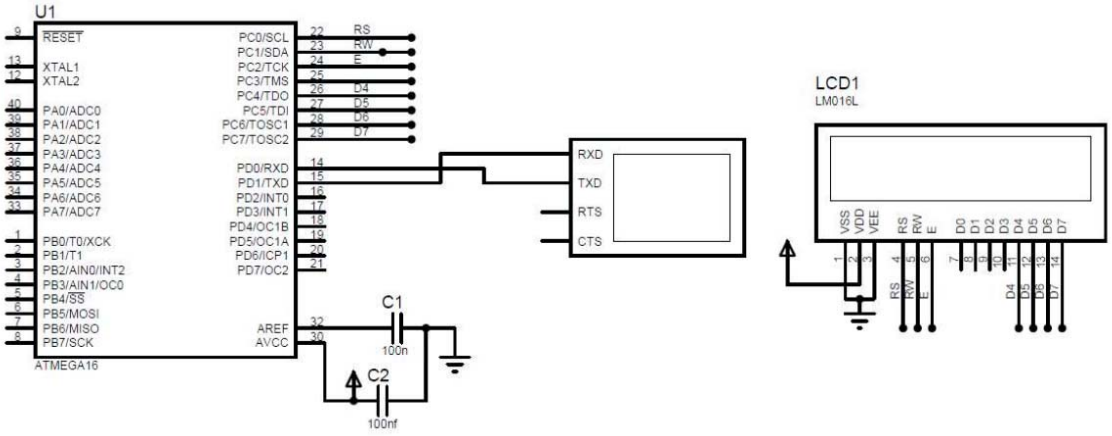


شکل 7-17: اجرای برنامه در محیط پروتئوس و Matlab

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

7.10 برنامه‌های اجرایی ارتباط سریال UART

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



شکل 7-18: نمایی از سخت‌افزار طراحی شده برای مبحث UART

1. زیر برنامه ای بنویسید که در آن ریزپردازنده یک رشته مانند نام و نام خانوادگی را از خط سریال دریافت و سپس با افزودن (<<>>) به ابتدا و انتهای آن، نتیجه را بر روی LCD نمایش دهد.
2. وقفه‌ی فرستنده و گیرنده‌ی UART را فعال نموده و زیربرنامه‌ای بنویسید که به ازای دریافت یک کاراکتر مشخص، عبارت تعیین شده را مطابق جدول زیر بر روی LCD نمایش دهد.

کاراکتر	عبارت مورد نظر	مثال
کاراکتر بین 0 تا 9	ده برابر آن نمایش داده شود	Tx: 5 Rx : data= 5 and 10*data=50
کاراکترD	چاپ شدن عبارت !LCD Deleted روی LCD	
کاراکترH	نمایش توضیحاتی دلخواه	********** Micro processor lab **********
کاراکتر E	پایان اجرای این بند	Rx: END of this part
سایر کاراکترها	نمایش کاراکتر	Tx: p Rx: input letter is “p”

3. زیر برنامه ای بنویسید که یک بسته 5 کاراکتری از ارقام را بین دو پرانتز دریافت نماید (مانند (12345)) و مطابق جدول زیر پیام‌هایی را روی LCD نمایش دهد.

دریافت	عبارت مورد نظر	مثال
(تعداد رقم‌ها برابر با 5 نباشد)	Incorrect frame size ! The frame must be 5 integers	Tx: (156) Rx: Incorrect frame size
(تعداد رقم‌ها برابر با 5 باشد)	The frame is correct و چاپ شدن بسته روی lcd	Tx: (12345) Rx: The frame is correct
بسته شامل حرف باشد	The frame must be 5 integers به همراه پیام دریافتی	Tx:(986a4) Rx: Frame must be 5 integer

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!