۸ آشنایی با ارتباط سریال UART

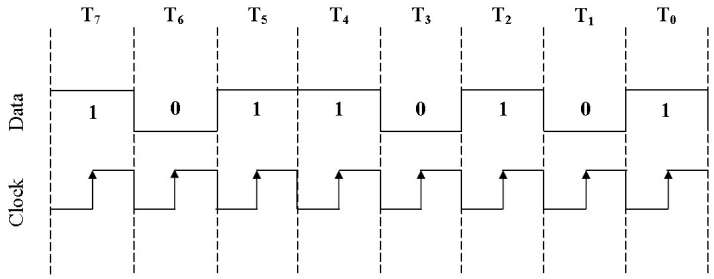
انتقال اطلاعات در سیستم‌های دیجیتالی، به دو روش سریال و موازی صورت می‌گیرد. در روش موازی n بیت اطلاعات از طریق n خط داده منتقل می‌شود اما در روش سریال همه‌ی داده از طریق یک یا دو خط منتقل می‌شود. ارتباط سریال انواع مختلف دارد که از نظر حداکثر طول کابل ارتباطی، نرخ ارسال و دریافت داده و تعداد سیم‌ارتباطی، 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 Ethernet، USB، CAN، Lon Works، SATA، I2C و SPI اشاره نمود.

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

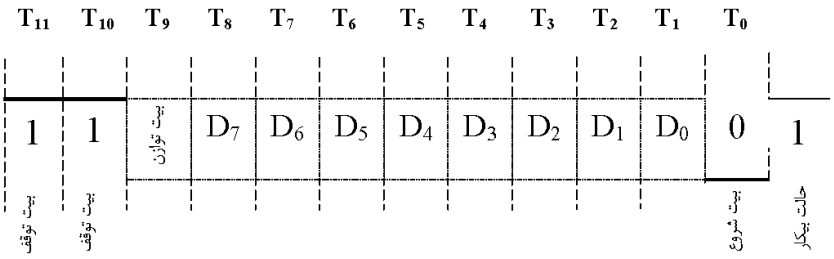
برای انتقال به روش سریال، پروتکل‌های متنوعی وجود دارد. اما به طور کلی ارتباط سریال به دو صورت سنکرون و آسنکرون برقرار می‌شود.

در ارتباط سنکرون مانند شکل ۸-۱، داده‌ها بر روی یک خط ارسال می‌شوند و یک خط پالس ساعت همزمان کننده نیز وجود دارد که به همراه داده‌ها برای سنکرون‌سازی، توسط فرستنده ارسال می‌شود.



شکل ۸-۱: ارتباط به صورت سنکرون

در ارتباط آسنکرون، داده موردنظر از طریق خط TXD ارسال شده و از خط RXD دریافت می‌شود. بنابراین در این ارتباط 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. به این فواصل زمانی، نرخ انتقال داده یا Baud Rate گفته می‌شود. در شکل ۸-۲ یک قالب داده با یک بیت توازن و دو بیت توقف در ارتباط آسنکرون مشاهده می‌شود.



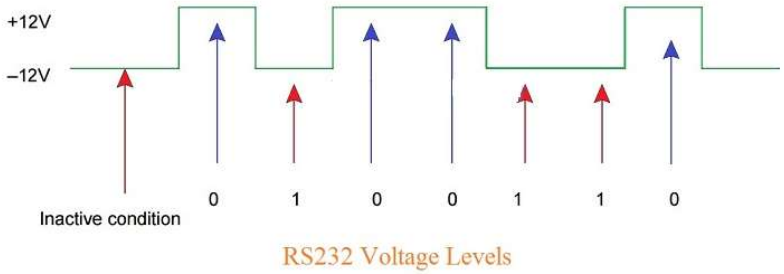
شکل ۸-۲: ارتباط به صورت آسنکرون

۸.۱ معرفی ارتباط USART (RS232)

یکی از پروتکل‌هایی که ریزپردازنده Atmega16 برای ارتباط سریال پشتیبانی می‌کند، پروتکل USART[1] است که قابلیت برقراری ارتباط با هر دو حالت سنکرون و آسنکرون را دارد که برای افزایش قابلیت اطمینان تحت استاندارد کار می‌کند.

یکی از این استانداردها RS232-C است که در سال ۱۹۶۹ 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 برای ارتباط آن‌ها با ریزپردازنده استفاده می‌شود.

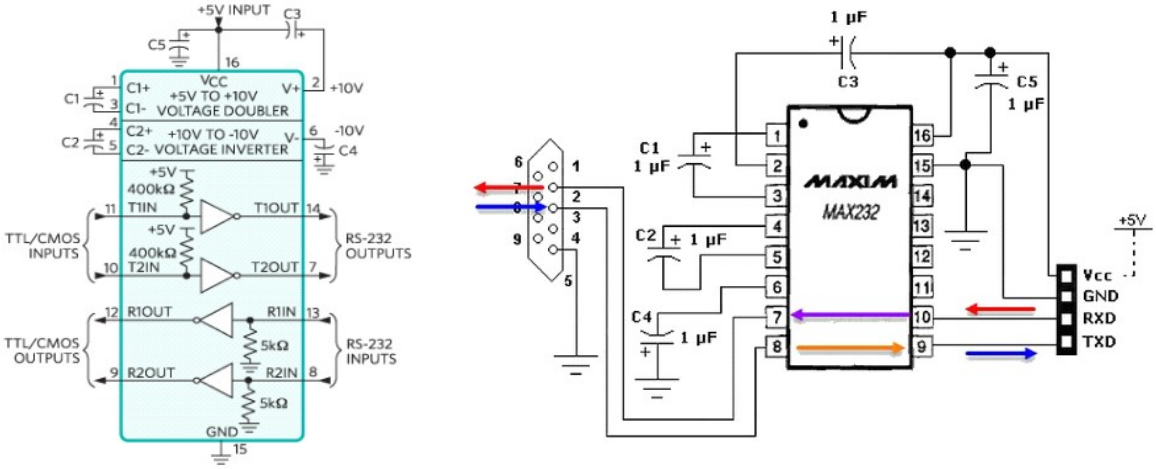
در استاندارد RS232 سطح ولتاژ ۳+ تا ۱۲+ نمایانگر وضعیت صفر منطقی و بازه‌ی ۳- تا ۱۲- ولت نمایشگر وضعیت یک منطقی می‌باشد. این در حالی است که تجهیزات استاندارد TTL مثل ریزپردازنده Atmega16 در سطوح بین ۰ و ۵ ولت کار می‌کنند.



شکل ۸-۳: سطوح ولتاژ در RS232

در نتیجه برای برقراری ارتباط بین وسایل TTL و RS232 از درایوری 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 ۸-۴).

[1] Universal Synchronous Asynchronous serial Receiver/Transmitter



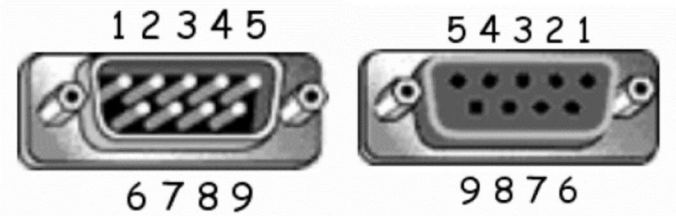
شکل ۸-۴: مدار داخلی max232 و نحوه اتصال تراشه به کانکتور مادگی DB9

۸.۲ ارتباط Atmega16 و کامپیوتر

Atmega16 دارای ۲ پایه برای دریافت و انتقال داده به صورت سریال (مطابق با استاندارد RS232) با نام‌های TX وRX واقع در PD0 و PD1 است که مطابق استاندارد TTL می‌باشند. به همین دلیل باید این پایه‌ها را به MAX232 متصل نمود. همان طور که درشکل ۸-۴ نشان داده شده TX به T2IN و RX به R2OUT متصل می‌گردند. بعد از برقراری اتصال ریزپردازنده به MAX232 سپس باید این تراشه را به درگاه سریال کامپیوتر متصل نمود تا این ارتباط کامل گردد.

درگاه سریال در کامپیوترهای شخصی که به نام‌های COM Port و RS-232 Port هم شناخته می‌شود که دارای کانکتور DB9 مانند شکل ۸-۵است 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 در کامپیوتر متصل است 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 ۸-۵: سمت راست کانکتور مادگی DB9 - سمت چپ کانکتور نری DB9 و پایه های کانکتور

۸.۳ ثبات‌های ارتباط USART

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

۸.۳.۱ ثبات UDR

اصطلاحا ثبات بافر داده خوانده می‌شود و داده‌های ارسالی و دریافتی در آن ریخته می‌شوند. این ثبات ۱۶ بیتی مانند شکل ۸-۶ از دو بخش RXB[0:7] و TXB[0:7] تشکیل شده است. ثبات UDR دارای انتقال Full Duplex است و می‌تواند به طور همزمان اطلاعات را ارسال (با استفاده از پایه TX) و دریافت کند (با استفاده از پایه RX).

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 ۸-۶: ثبات UDR

۸.۳.۲ ثبات UCSRA

این ثبات مانند شکل ۸-۷ برای کنترل و نمایش وضعیت به کار می‌رود.

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 ۸-۷: ثبات UCSRA

بیت ۷ - 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 ۶ - 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 ۵ - UDRE (USART Data Register Empty): در صورت یک بودن، ثبات فرستنده آماده دریافت داده‌ی جدید می‌باشد.

بیت ۴ - FE (Frame Error): اگر کاراکتر بعدی در زمان دریافت در بافر گیرنده، خطای بسته داشته باشد، این بیت یک می‌شود.

بیت ۳ - 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 ۲ - PE (Parity Error): اگر در کاراکتر بعدی موجود در بافر گیرنده خطایparity وجود داشته باشد ،این بیت یک می‌شود.

بیت ۱ - UDX (Double the USART transmission Speed): این بیت تنها در حالت آسنکرون کاربرد دارد و در حالت سنکرون باید صفر گردد. 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.

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

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 ۸-۸: ثبات UCSRB

۸.۳.۳ ثباتUCSRB:

این ثبات مانند شکل ۸-۸شکل ۸-۷ برای کنترل و نمایش وضعیت به کار می‌رود.

بیت ۷ - RXCIE (RX Complete Interrupt Enable): با یک کردن این بیت، وقفه اتمام دریافت فعال می‌شود.

بیت ۶ - TXCIE (TX Complete Interrupt Enable): با یک کردن این بیت، وقفه مربوط به اتمام ارسال فعال می‌گردد.

بیت ۵ - UDRIE (USART Data Register Empty Interrupt Enable): با یک کردن این بیت وقفه مربوط به خالی شدن بافر فعال می‌گردد .

بیت ۴ - RXEN (Receiver Enable): با یک کردن این بیت،USART به صورت گیرنده فعال می‌گردد.

بیت ۳ - TXEN (Transmitter Enable): با یک کردن این بیت،USART به صورت فرستنده فعال می‌گردد.

بیت ۲ - UCSZ (Character Size): این بیت به صورت ترکیب با بیت‌های USCZ 1:0 تعداد بیت‌های داده در یک بسته را مشخص می‌کند.

بیت ۱ - RXB8 (Receive Data Bit 8): زمانی که بسته ۹ بیت داده داشته باشد، بیت نهم مربوط به بیت داده ازکاراکتر دریافتی را در خود جای می‌دهد.

بیت صفر - TXB8 (Transmit Data bit 8): زمانی که بسته ۹ بیت داده داشته باشد، بیت نهم مربوط به بیت داده از کاراکتر ارسالی را در خود جای می‌دهد.

۸.۳.۴ ثبات UCSRC

این ثبات مانند شکل ۸-۷ شکل ۸-۹ برای کنترل و نمایش وضعیت به کار می‌رود.

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 ۸-۹: ثبات UCSRC

بیت۷- URSEL(register select) : این بیت دسترسی به ثبات UBRRH و UCSRC را مشخص می‌نماید. هنگام خواندن UCSRC این بیت برابر با یک است و برای نوشتن در UCSRC نیز مقدار یک در آن بایستی نوشته شود.

بیت ۶-UMSEL(USART Mode Select) : این بیت حالت عملکرد سنکرون و آسنکرون را نشان می‌دهد.

بیت ۵ و ۴ UPM1:0(Parity Mode): این بیت‌ها وضعیت parity را تنظیم می‌نماید.

بیت ۳USBS(STOP Bit Select): تعداد بیت‌های STOP را در بسته ارسالی توسط فرستنده مشخص می‌نماید.

بیت ۲ و ۱UCSZ(Character size): تعداد بیت‌های داده را در بسته ارسال و دریافتی نشان می‌دهد.

۸.۳.۵ ثبات‌های UBRRL و UBRRH (USART Baud rate Register)

ثبات UBRRH و ثبات UCSRC از یک مکان در حافظهI/O استفاده می‌کنند.

بیت ۱۵ - 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 ۱۴-۱۲ Reserved Bits: این بیت‌ها برای استفاده‌های بعدی ذخیره شده‌اند.

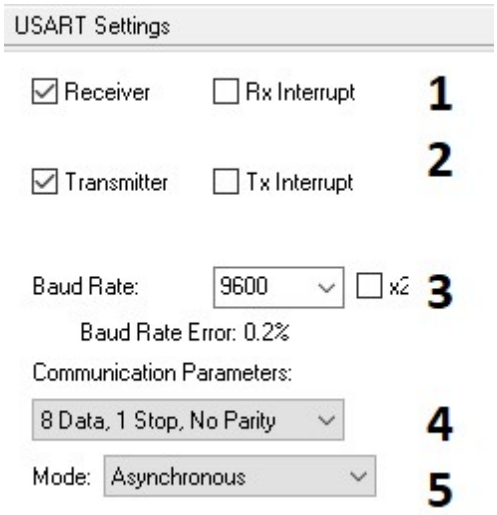
بیت‌های ۱۱-۰ UBRR: این یک ثبات ۱۲ بیتی است که نرخ ارسالUSART را تنظیم می‌کند.

۸.۴ 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 Codevision مراجعه نمایید. این سرآیند فایل شامل دستوراتی برای ارسال و دریافت کاراکتر، رشته شامل printf و scanf puts وgets می‌باشد. همه‌ی این دستورات بر پایه‌ی دستورات getchar() و putchar(char c) می‌باشد که به ترتیب کاراکتری را دریافت و ارسال می‌نماید.

۸.۵ برنامه نویسی در محیط Codevision برای USART

با توجه به وجود CodeWizard، تنظیمات ثبات‌ها به صورت خودکار انجام می‌شود. لذا پارامترهای اولیه مانند شکل ۸-۱۰ تنظیم می‌گردد.



۱: فرستنده فعال می‌شود.
۲: گیرنده فعال می‌شود.
۳: نرخ ارسال مورد نظر تنظیم می‌گردد.
۴: قالب بسته ارسالی مشخص می‌گردد.
۵: حالت سنکرون و آسنکرون انتخاب می‌گردد.

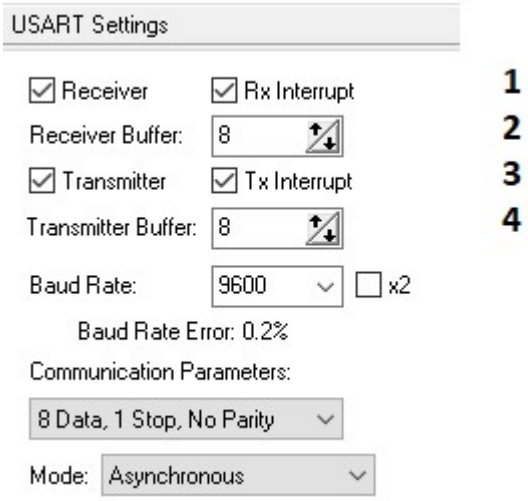
شکل ۸-۱۰: تنظیمات UART در CodeWizard

در تنظیم نرخ ارسال بایستی توجه داشت که خطای به وجود آمده برای کارکرد مناسب باید کمتر از 0.5٪ باشد. نرخ ارسال‌های بالاتر که ممکن است خطای بیشتری داشته باشد برای Atmega16 مناسب نیست.

با توجه تنظیمات انجام شده می‌توان کد را ذخیره نمود و از تمام دستورات موجود در فایل سرآیند استفاده نمود. در این روش با ارسال داده، اطلاعات در قالب بسته مورد نظر روی TX ارسال می‌گردد و برای دریافت اطلاعات منتظر می‌ماند تا داده‌ای از RX دریافت نماید.

۸.۵.۱ وقفه‌های بخش UART

سرکشی مداوم به ارسال و دریافت در بخش UART، وقت زیادی از ریزپردازنده را درگیر نموده و بهتر است از روش وقفه‌ای در ارسال و دریافت داده استفاده گردد که تنظیمات آن درشکل ۸-۱۱ نشان داده می‌شود.



۱: وقفه گیرنده فعال می‌شود.
۲: اندازه بافر گیرنده تنظیم می‌شود.
۳: وقفه فرستنده فعال می‌گردد.
۴: اندازه بافر فرستنده تنظیم می‌شود.

شکل ۸-۱۱: تنظیمات UART در حالت وقفه‌ای

در حالت استفاده از وقفه، اندازه بافر فرستنده و گیرنده با توجه به شرایط پروژه و حجم کاری ریزپردازنده و نحوه پردازش داده‌ها تعیین می‌گردد. مشاهده می‌شود که مانند برنامه ۸-۱ حجم کدهای تولید شده توسط CodeWizard در مقایسه با حالت بدون وقفه کاملا متفاوت است و در این بخش زیربرنامه‌های مربوط به وقفه ارسال و دریافت ایجاد شده است. همچنین ماهیت دستورات putchar و getchar نیز تغییر کرده است و برای هر یک تعریف جدیدی ارائه شده است.

برنامه ۸-۱

```
#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

       }
}
```

۸.۵.۲ وقفه‌ی RX

وقتی که داده جدیدی در ثبات UDR بخش RX 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. متغیر rx_counter به تعداد داده‌های موجود در بافر گیرنده که هنوز توسط ریزپردازنده بررسی نشده اشاره دارد.

برای دریافت داده، ریزپردازنده با استفاده از دستورات فایل‌های سرآیند که مبتنی بر getchar هستند به فضای بافر گیرنده دسترسی دارد. دستور getchar داده ای از بافر گیرنده با شاخص rx_rd_index را دریافت و متغیر rx_counter را یک واحد کاهش می‌دهد.

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

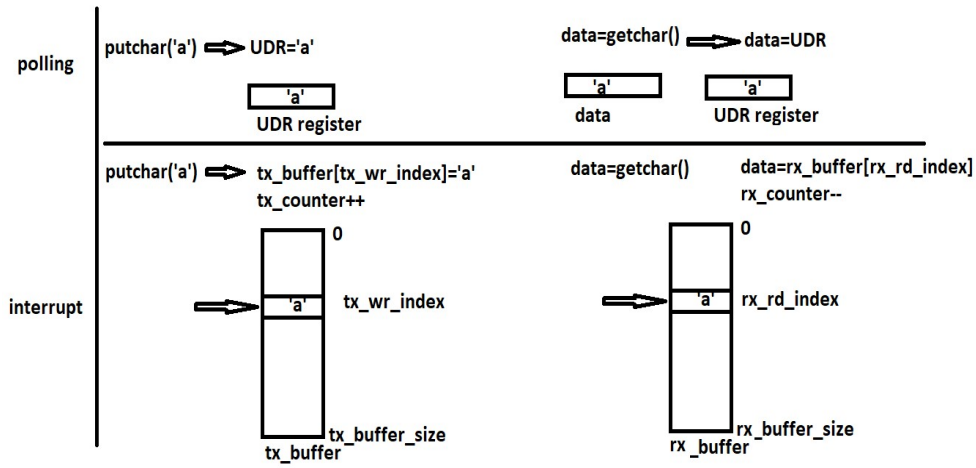
۸.۵.۳ وقفه‌ی TX

وقتی که داده در ثبات UDR بخش TX ارسال گردد، وقفه فرستنده رخ می‌دهد و در زیربرنامه مربوطه، داده‌ی جدیدی از فضای بافر فرستنده با که با شاخص tx_rd_index اشاره شده به UDR منتقل می‌شود و متغیر tx_counter یک واحد کاهش می‌یابد. متغیر tx_counter به تعداد داده‌هایی که در بافر فرستنده، هنوز توسط UART ارسال نشده اشاره دارد.

برای 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.

۸.۵.۴ تغییر ماهیت دستورات putchar و getchar

فرایندی که برای استفاده از دستورات getchar و putchar در حالت سرکشی و وقفه رخ می‌دهد درشکل ۸-۱۲ نشان داده شده است.

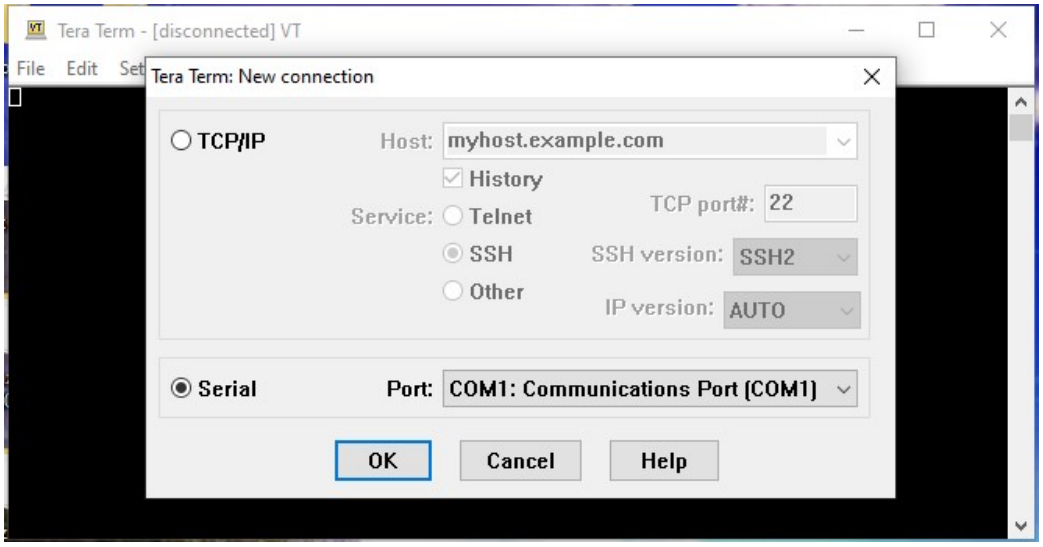


شکل ۸-۱۲: تفاوت ماهیت دستورات putchar و getchar در حالت polling و وقفه در UART

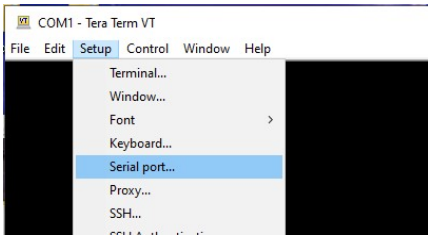
۸.۶ نرم افزار Teraterm

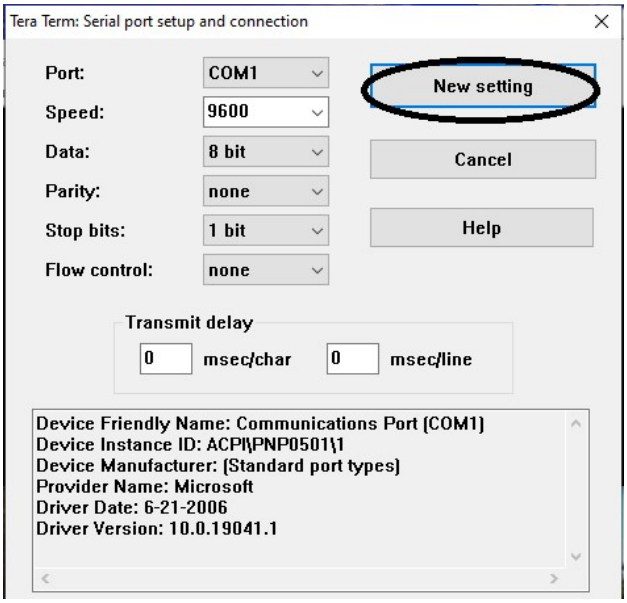
پس از اتصال سیگنال‌های TX و RX به MAX232 می‌توان خروجی آن را به یکی از درگاه‌های کامپیوتر با کانکتور DB9 متصل نمود و از نرم افزارهایی مانند Teraterm برای نمایش داده‌های ارسالی استفاده نمود. مراحل کار با این نرم‌افزار در **Error! Reference source not found.** نشان داده شده است.

الف

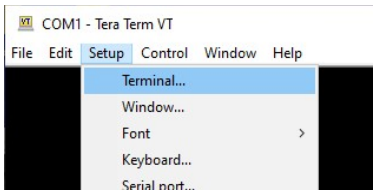


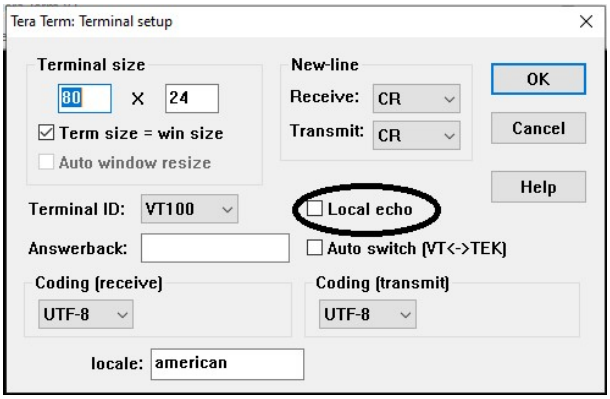
ب





ج



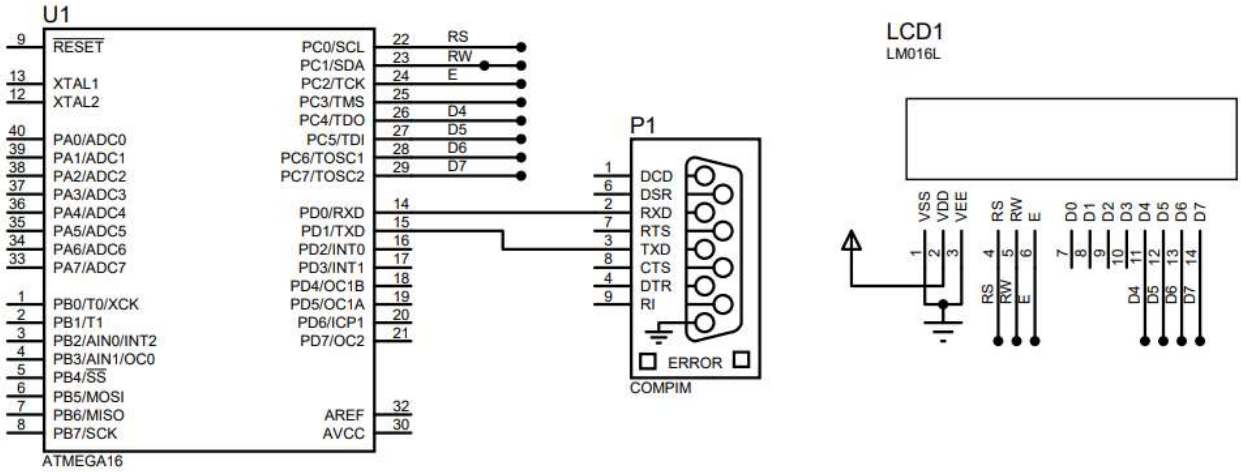


با توجه به اینکه امروزه اکثر کامپیوترها فاقد کانکتور DB9 هستند و بی

۸.۷ برقراری ارتباط سریال در نرم‌افزار MATLAB

وجود UART در طراحی این امکان را فراهم کرده که بتوان سخت‌افزار را به نرم افزارهای توانمندی مانند Matlab و Labview متصل نمود. لذا در این بخش اتصال به Matlab از محیط شبیه‌سازی پروتئوس و سخت‌افزار واقعی شرح داده شده است.

برای ارتباط نرم‌افزار پروتئوس به Matlab R2021b از طریق UART می‌توان از سخت‌افزار شکل ۸-۱۳ استفاده نمود که با کانکتور COMPIM به درگاه مجازی متصل می‌گردد. برای اتصال درگاه‌های مجازی نرم افزار Virtual Serial Port Driver Pro مورد استفاده قرار می‌گیرد. کدهای نوشته شده در محیط Matlab و CodeVision به ترتیب در برنامه ۸-۲ و برنامه ۸-۳ آمده است. در برنامه CodeVision وقفه‌های UART فعال شده است.



شکل ۸-۱۳: سخت افزار اتصال به Matlab در محیط پروتئوس

برنامه ۸-۲

```
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 ۸-۳

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

برای اجرا ابتدا درگاه مجازی مانند شکل ۸-۱۴تعریف گردیده و ابتدا برنامه پروتئوس و سپس برنامه Matlab اجرا می‌شود. اجرای این برنامه درشکل ۸-۱۵ نشان داده شده است.

در صورتی که نیاز به تغییر برنامه باشد باید درگاه مجازی پاک و دوباره ایجاد گردد و روند بالا ادامه یابد.

Open “Virtual Serial Port Driver Pro”

Select “Pair”

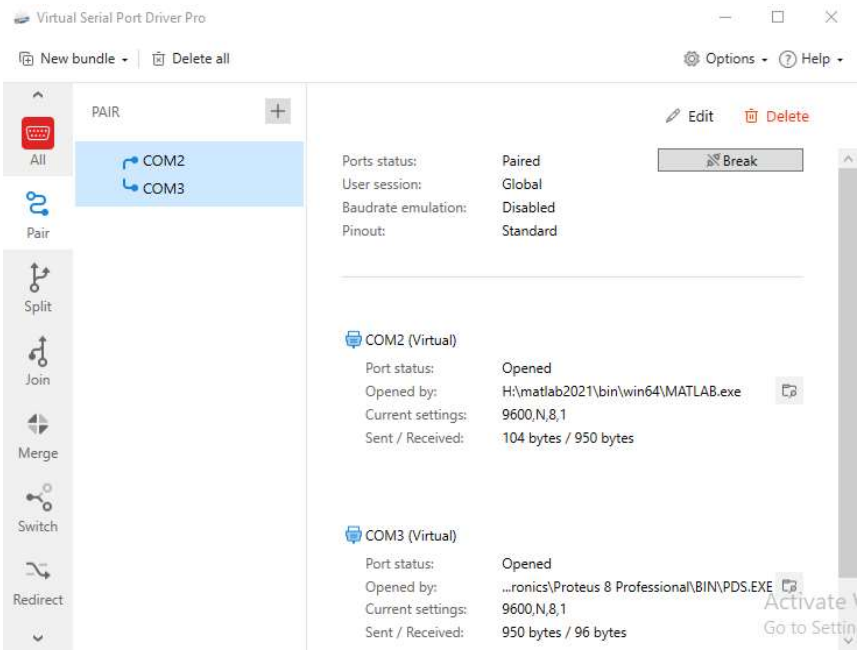
Select “add a new pair”

Select “Create”

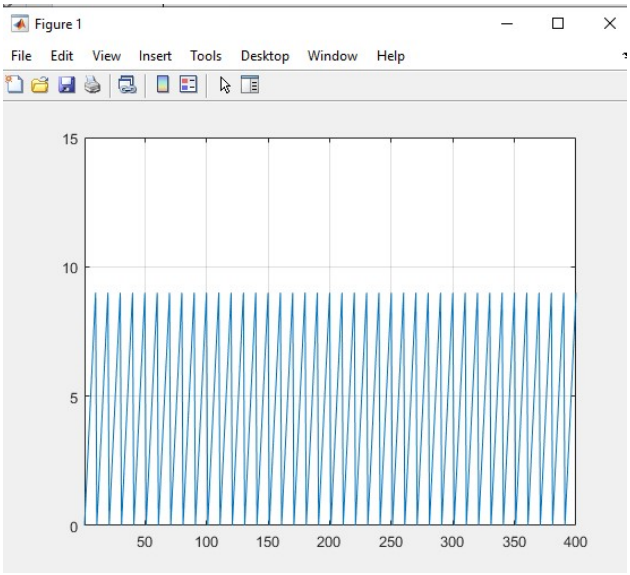
……..

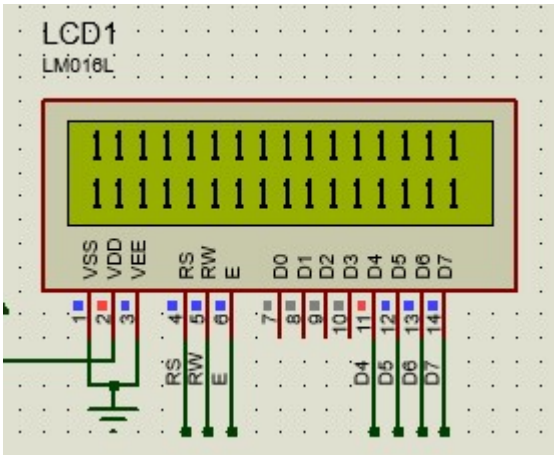
For end of connection : select “Delete”

On pair section



شکل ۸-۱۴: فعال نمودن درگاه مجازی



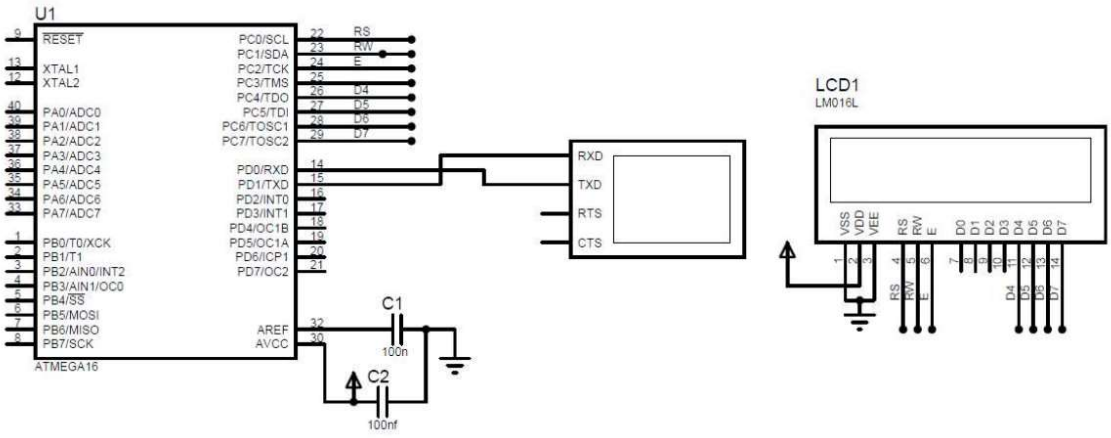


شکل ۸-۱۵: اجرای برنامه در محیط پروتئوس و Matlab

در صورتی که سخت افزار به کامپیوتر متصل شود فقط کافی است درگاه مورد نظر به درستی در Matlab انتخاب شود و ابتدا سخت افزار و سپس برنامه Matlab اجرا گردد.

۸.۸ برنامه‌های اجرایی ارتباط سریال UART

سیستم طراحی شده در شکل ۸-۱۶ را در نظر بگیرید.



شکل ۸-۱۶: نمایی از سخت افزار طراحی شده برای مبحث UART

۱. زیر برنامه ای بنویسید که میکرو یک رشته مانند نام و نام خانوادگی را به روش سرکشی دریافت و سپس با افزودن (<<>>) به ابتدا و انتهای رشته آن را نمایش دهد.

۲. در قالب پروژه مستقل از بند یک، وقفه‌ی فرستنده و گیرنده‌ی UART را فعال نموده و زیربرنامه‌ای بنویسید که به ازای دریافت عبارت‌های ذیل کارکردهای مورد نظر را انجام دهد.

نمونه	عملیات مورد نظر	کاراکتر
Tx: 5 Rx: Data is a integer and 10*data=50	ده برابر آن نمایش داده شود	کاراکتر بین ۰ تا ۹
	چاپ شدن عبارت LCD Deleted! روی LCD	کاراکترD
********** Micro processor lab ********** Rx: END of the part	نمایش توضیحاتی دلخواه	کاراکترH
Tx: p Rx: input letter is “p”	پایان اجرای این بند	کاراکتر E
	نمایش کارکتر	سایر کاراکترها

۳. زیر برنامه ای بنویسید که یک فریم 5 کاراکتری را مشابه (۱۲۳۴۵) دریافت نماید و مطابق جدول زیر پیام‌هایی را رویLCD نمایش دهد. ۵ رقم مورد نظر باید داخل () قرار بگیرد.

نمونه	عملیات مورد نظر	دریافت
Tx: (156) Rx: Incorrect frame size	Incorrect frame size ! The frame must be 5 integers	(تعداد رقم‌ها برابر با ۵ نباشد)
Tx: (12345) Rx: The frame is correct	The frame is correct و چاپ شدن بسته روی lcd	(تعداد رقم‌ها برابر با ۵ باشد)
Tx:(۹۸۶a4) Rx: Frame must be 5 integer	Frame must be 5 integer	بسته شامل حرف باشد

۴. برنامه‌های فوق را در قالب دو پروژه مستقل ارائه دهید. اولین پروژه شامل بند یک و پروژه‌ی دیگر شامل بندهای ۲ و ۳ 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. می‌توانید در ابتدای اجرای هر بند پیام‌هایی را مانند ذیل نمایش دهید.

Part 2 is running!

اجرای بند دوم

Part 2 is ending!