9 جلسه نهم
ارتباط سریال SPI

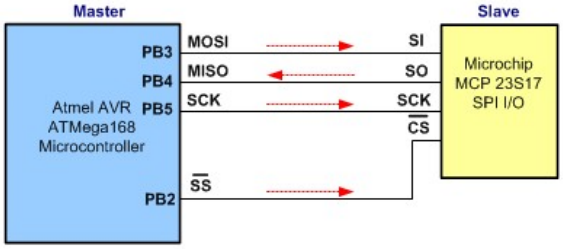
9.1 هدف

در این جلسه نحوه برقراری ارتباط سریال SPI بررسی می‌گردد.

9.2 مقدمه

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

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



Pin Name	Description
MOSI	Master Out Slave In
MISO	Master In Slave Out
SCK	Synchronous Clock
SS	Slave Select (active Low)
CS	Chip Select (active low)
SI	Slave In
SO	Slave Out

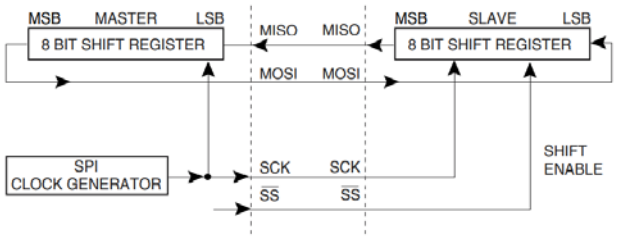
Typical SPI Master and SPI Slave Device Connection

شکل 9-1 : نمایی از ارتباط SPI

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

اساس کارSPI مانند شکل 9-2 بر پایه دو ثبات می‌باشد که پس از برقراری اتصال بین دو دستگاه، با هر پالس ساعت، یک بیت از ثبات فرستنده خارج شده و به ثبات گیرنده وارد می‌شود. لذا دو ثبات ۸ بیتی در MASTER و SLAVE را می‌توان به عنوان یک ثبات چرخشی ۱۶ بیتی در نظر گرفت. زمانی که داده‌ای از MASTER به SLAVE

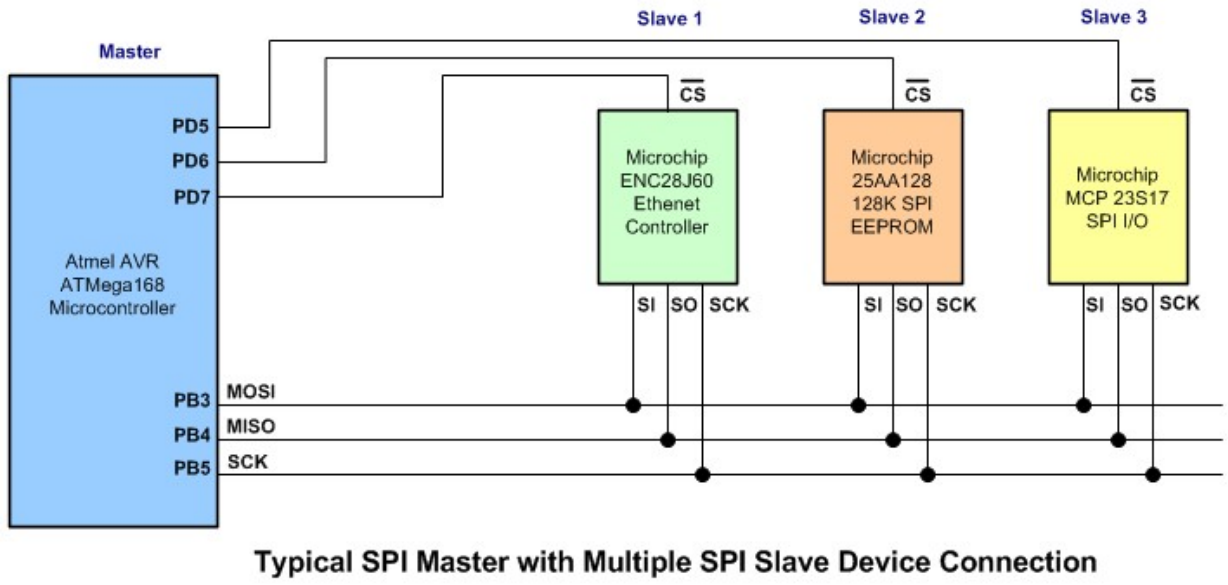
ارسال می‌شود، در همان حال و در جهت مخالف، داده‌ای از SLAVE به MASTER انتقال می‌یابد. به این صورت در طول هشت پالس ساعت SPI، داده‌های MASTER و SLAVE به صورت کامل با یکدیگر عوض می‌شوند.



شکل 9-2: نحوه انتقال اطلاعات در ارتباط SPI

بر همین اساس ارتباط SPI یک ارتباط Full Duplex محسوب می‌گردد که به صورت همزمان توانایی ارسال و دریافت داده را دارد. زمانی که MASTER بخواهد از SLAVE داده دریافت کند، SLAVE باید یک بایت داده بر روی ثبات قرار دهد و بعد از 8 پالس ساعت، MASTER ضمن ارسال یک داده‌ی 8 بیتی از سمت خودش، داده SLAVE را دریافت خواهد کرد.

همچنین مانند شکل 9-3 امکان اتصال چندین ماژول در یک ارتباط SPI به طور همزمان وجود دارد که برای هر کدام می‌توان از یک خط SSn جداگانه استفاده کرد.



شکل 9-3: اتصال چند ماژول به SPI

9.3 ثبات‌های SPI

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

9.3.1 ثبات SPDR (SPI Data Register)

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

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

شکل 9-4: ساختار ثبات SPDR

9.3.2 ثبات SPSR (SPI Status Register)

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

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

شکل 9-5: ساختار ثبات SPSR

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

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

بیت صفر SPI2X (Double SPI Speed Bit): با نوشتن یک در این بیت، فرکانس پالس ساعت در حالت Master دو برابر می‌شود.

9.3.3 ثبات SPCR (SPI Control Register)

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

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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 9-6: ساختار ثبات SPCR

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

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

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

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

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

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

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

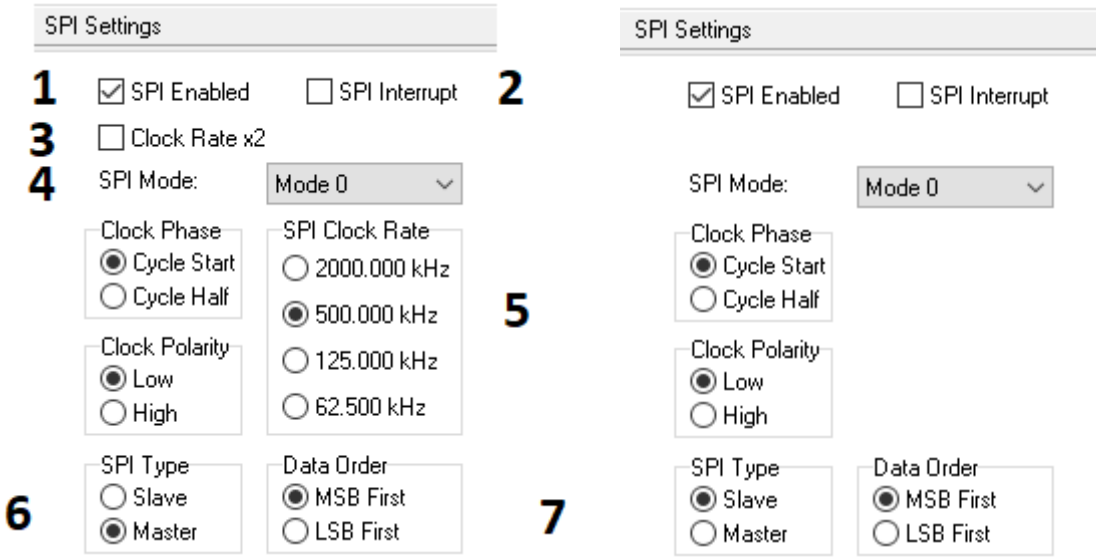
9.4 پیکربندی SPI در Codevision

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

File> new project>

via code wizard

select: Serial Peripheral Interface



1: فعال شدن SPI
2: فعال شدن وقفه
3: دو برابر شدن پالس ساعت
4: حالت SPI انتخاب می‌شود. با انتخاب لبه نمونه برداری و تقدم سطح پالس ساعت، وضعیت SPI نیز تنظیم می‌شود.
5: انتخاب پالس ساعت در حالت Master
6: انتخاب Master/Slave
7: انتخاب ترتیب داده در ارسال

شکل 9-7: تنظیمات SPI

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

جدول 9-1 : پیکربندی پایه‌های SPI

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

9.5 سناریوهای مختلف ارتباط SPI

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

در حالت Master

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

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

در حالت Slave

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

عملکرد پایه‌ی SSn در حالت Master

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

عملکرد پایه‌ی SSn در حالت Slave

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

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

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

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

9.6 برنامه‌های کاربردی SPI

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

برنامه 9-1

```
// Master Program

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

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


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


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

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



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

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

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

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

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

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

        delay_ms(500);
      }
}
/*********************************************************************/
```

برنامه 9-2

```
//Slave Program

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

void main(void)
{
```

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

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


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


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


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

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

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

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

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

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

```
            delay_ms(500);
        }
    }
```

در برنامه‌های فوق با فراخوانی تابع ()spi، می‌توان یک بایت را از طریق سرکشی بر روی گذرگاه SPI ارسال نمود و به صورت همزمان بایت دیگری را دریافت کرد. ولی نکته مهم در ارسال داده، تشخیص ابتدا و انتهای بسته‌ی داده است که چگونه آن‌ها را بفرستد و پردازش نماید. یک روش مطمئن برای انجام این کار، تبدیل داده‌ها به کاراکترهای معادل و ارسال رشته‌های کاراکتری روی گذرگاه SPI می‌باشد. برای ارسال و دریافت کاراکتر، دو تابع ()putchar و ()getchar مطابق برنامه 3-9 مجدد تعریف شده‌اند. با این کار، از آن جایی که توابع printf و puts و putsf از این توابع استفاده می‌کنند، نحوه عملکرد آن‌ها نیز تغییر می‌یابد.

برنامه 9-3

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

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

در برنامه 9-4 و برنامه 9-5 نمونه برنامه‌ای برای بخشSPI با استفاده از تعریف مجدد دستورات Getchar و putchar آمده است.

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

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

برنامه 4-9

```
    }
#pragma used-

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

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

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

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

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

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

      }
}
```

برنامه 5-9

```
/* slave program*/
#include <mega16.h>
#include <spi.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
#include <string.h>

#define _ALTERNATE_PUTCHAR_
#pragma used+
```

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

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

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

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

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

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

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

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

lcd_init(16);

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

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

برنامه 9-6

```
/* Master program */

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

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

void main(void)
{

    char count=0;
    char buffer[5];
```

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


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


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


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

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

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

    lcd_init(16);

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



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

while (1)
      {
       SPDR=count;
```

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

برنامه 7-9

```
/* Slave Program*/

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

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

data=SPDR;
SPDR=count;

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



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

void main(void)
{


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


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

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


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


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

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

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

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

while (1);

}
```

درصورتی که وقفه فعال نشده باشد، می‌توان بدون استفاده از Code Wizard و فایل سرآیند spi.h و به کمک توابعی که در قالب فایل‌های جانبی به پروژه اضافه می‌گردند، برای پیکربندی و تبادل داده SPI اقدام نمود. تنظیمات LCD را هم می‌توان از طریق Project/Configure/C compiler/libraries/ Alphanumeric LCD(alcd.h) ویرایش نمود. در برنامه 9-8 نمونه فایل سرآیند آمده است.

برنامه 9-8

```
#ifndef SPI_H_files_H_
#define SPI_H_files_H_

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

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

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

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

#endif
```

دربرنامه 9-9 پیاده‌سازی مربوط به توابع فوق قابل مشاهده است.

برنامه 9-9

```
#include "SPI_H_files.h"
void SPI_Slave_Init()                          /* SPI Initialize function */
{
    DDRB &= ~((1<<MOSI)|(1<<SCK)|(1<<SS));
                          /* Make MOSI, SCK, SS pin direction as input pins */
    DDRB |= (1<<MISO);                        /* Make MISO pin as output pin */
    SPCR = (1<<SPE);                         /* Enable SPI in slave mode */
}
/************************************************************************/
char SPI_Slave_Transmit(char data)           /* SPI transmit data function */
{
    SPDR = data;                          /* Write data to SPI data register */
    while(!(SPSR & (1<<SPIF)));           /* Wait till transmission complete */
    return(SPDR);                                 /* return received data */
}
/************************************************************************/
char SPI_Slave_Receive()                     /* SPI Receive data function */
{
    while(!(SPSR & (1<<SPIF)));           /* Wait till reception complete */
    return(SPDR);                                 /* return received data */
}

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

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

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

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

برنامه 9-10

```
/*Master Program*/

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

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

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

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

```
    }


/* Slave program*/


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

برنامه 9-11

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

       lcd_init(16);
       SPI_Slave_Init();


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

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

}
```

9.7 مقایسه ارتباط سریال UART و SPI

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

9.7.1 UART

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

همچنین درگاه‌های سریال آسنکرون نیاز به سخت‌افزار جانبی مانند Max232 دارند. همچنین حداقل نیاز به یک بیت شروع و یک بیت پایان بخش درون هر بسته داده است، به این معنی که برای انتقال هر 8 بیت داده به زمانی معادل با انتقال 10 بیت داده نیاز است و این به شدت نرخ انتقال داده را پایین می‌آورد.

یک ایراد بنیادی دیگر در ارتباط آسنکرون این است که این نوع ارتباط صرفاً برای ارتباط بین دو دستگاه طراحی شده است.

نرخ تبادل داده نیز یک مشکل است. با این که از نظر تئوری محدودیتی در ارتباطات سریال آسنکرون وجود ندارد، بیشتر دستگاهای USART تنها از مجموعه‌ای از نرخ‌های ثابت از پیش تعیین شده استفاده می‌کنند. بالاترین این نرخ‌ها معمولاً حدود 230400 بیت بر ثانیه است.

9.7.2 SPI

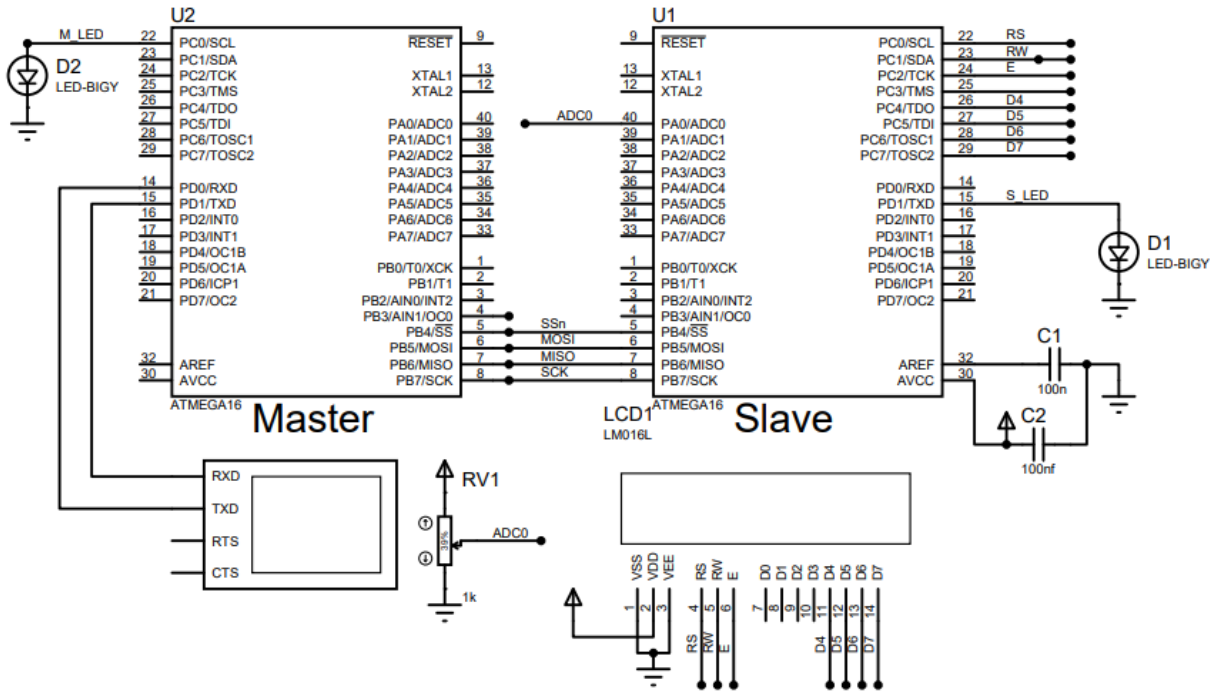
بزرگ‌ترین ایراد SPI تعداد پایه‌های مورد نیاز است. برقراری ارتباط SPI تنها بین یک زوج دستگاه Master و Slave نیاز به 4 سیم دارد. هر دستگاه Slave دیگری هم به ارتباط اضافه شود، یک پایه SSn بین آن و دستگاه Masterاضافه خواهد شد. افزایش سریع تعداد اتصالات پایه‌ها، این پروتکل را در شرایطی که تعداد زیادی دستگاه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

ارتباط SPI فقط می‌تواند از یک Master و تعداد زیادی Slave پشتیبانی نماید و تعداد Slaveها به ظرفیت دستگاه‌های متصل به گذرگاه و تعداد پایه‌های SSn بستگی دارد.

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

9.7.3 برنامه‌های اجرایی مبحث SPI

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



شکل 9-8: نمایی از سخت‌افزار مبحث SPI

برنامه‌های Master

1- ریزپردازنده Master از طریق درگاه سریال، اطلاعاتی را از کاربر دریافت می‌نماید. این اطلاعات شامل نام و نام خانوادگی و شماره دانشجویی است. در ادامه این داده‌ها را از طریق SPI به Slave می‌فرستد. لازم است در زمان دریافت هر داده از کاربر، از پیام‌های راهنمای مناسب نیز استفاده نمایید.

2- Master تغییرات پتانسیومتر را که از SPI دریافت نموده است همراه با پیام مناسب، روی درگاه سریال ارسال می‌نماید.

برنامه‌های Slave

1- ریزپردازنده Slave داده‌هایی که از طریق ارتباط SPI با Master دریافت کرده است را پس از تشخیص ابتدا و انتهای بسته، بر روی LCD نمایش می‌دهد.

2- ریزپردازنده Slave، تغییرات پتانسیومتر را از طریق ارتباط SPI برای Master ارسال می‌کند.

برنامه کد ویژن (شامل تمام فایل‌ها را) برای هریک از بندها توسعه داده و از فایل‌های کمکی نیز استفاده نمایید. سپس این برنامه‌ها را در محیط پروتئوس شبیه‌سازی نموده و پروژه کامل را ارسال نمایید.