Embedded System Design

Chapter 5: C Peripheral Interfaces

(Part 3)
References


- PIC 16F877A datasheet
- PIC C Help
Outline

1. PIC16 UART Serial Link
2. PIC16 SPI Serial Bus
3. I2C Serial Bus
Outline

1. PIC16 UART Serial Link
2. PIC16 SPI Serial Bus
3. I2C Serial Bus
The universal synchronous/asynchronous receive transmit (USART) device is typically used in asynchronous mode to implement off-board, one-to-one connections.

The term asynchronous means no separate clock signal is needed to time the data reception, so only a data send, data receive, and ground wires are needed.

It is quick and simple to implement if a limited data bandwidth is acceptable.

**USART Operation**

**USART RS232 Signal**
• Any pair of pins can be used for this interface, as data rate is quite low, allowing the signals to be generated in software.

• However, a dedicated hardware port is provided, which must be used if an interrupt is needed.
## CCS RS232 Serial Port Functions

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS232 SET BAUD RATE</td>
<td>Set hardware RS232 port baud rate</td>
<td>setup_uart(19200);</td>
</tr>
<tr>
<td>RS232 SEND BYTE</td>
<td>Write a character to the default port</td>
<td>putc(65)</td>
</tr>
<tr>
<td>RS232 SEND SELECTED</td>
<td>Write a character to selected port</td>
<td>s=fputc(&quot;A&quot;,01);</td>
</tr>
<tr>
<td>RS232 PRINT SERIAL</td>
<td>Write a mixed message</td>
<td>printf(&quot;Answer:%4.3d&quot;,n);</td>
</tr>
<tr>
<td>RS232 PRINT SELECTED</td>
<td>Write string to selected serial port</td>
<td>fprintf(01,&quot;Message&quot;);</td>
</tr>
<tr>
<td>RS232 PRINT STRING</td>
<td>Print a string and write it to array</td>
<td>sprintf(astr,&quot;Ans=%d&quot;,n);</td>
</tr>
<tr>
<td>RS232 RECEIVE BYTE</td>
<td>Read a character to an integer</td>
<td>n=getc();</td>
</tr>
<tr>
<td>RS232 RECEIVE STRING</td>
<td>Read an input string to character array</td>
<td>gets(spoint);</td>
</tr>
<tr>
<td>RS232 RECEIVE SELECTED</td>
<td>Read an input string to character array</td>
<td>astring=fgets(spoint,01);</td>
</tr>
<tr>
<td>RS232 CHECK SERIAL</td>
<td>Check for serial input activity</td>
<td>s=kbhit();</td>
</tr>
<tr>
<td>RS232 PRINT ERROR</td>
<td>Write programmed error message</td>
<td>assert(a&lt;3);</td>
</tr>
</tbody>
</table>
RS232 Periperal Simulation
The program

```c
// Serial I/O using hardware RS232 port

#include "16F877A.h"
#include "16F8771.h"
#define delay(clock=8000000) // Delay function needed for RS232
#define rs232(UART1) // Select hardware UART

void main() {
  int incode;
  setup_uart(9600); // Set baud rate

  while(1)
  {
    incode = getc(); // Read character from UART
    printf(" ASCII = %d ", incode); // Display it on
    putc(13); // New line on display
  }
}
```
Outline

1. PIC16 UART Serial Link
2. PIC16 SPI Serial Bus
3. I2C Serial Bus
SPI Bus

• The serial peripheral interface (SPI) bus provides high-speed synchronous data exchange over relatively short distances (typically within a set of connected boards), using a master/slave system with hardware slave selection.

• One processor must act as a master, generating the clock. Others act as slaves, using the master clock for timing the data send and receive.

• The slaves can be other microcontrollers or peripherals with an SPI interface.

• The SPI signals are:
  – Serial Clock (SCK)
  – Serial Data In (SDI)
  – Serial Data Out (SDO)
  – Slave Select (!SS)
SPI Connection

SPI Signals
SPI Operation

• To transfer data, the master selects a slave device to talk to, by taking its SS line low.
• Eight data bits are then clocked in or out of the slave SPI shift register to or from the master. No start and stop bits are necessary, and it is much faster than RS232.
• The clock signal runs at the same speed as the master instruction clock, that is, 5MHz when the chip is running at the maximum 20MHz (16 series MCUs).
# SPI Driver Functions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI SETUP</td>
<td>Initializes SPI serial port</td>
<td>setup_spi(spi_master);</td>
</tr>
<tr>
<td>SPI READ</td>
<td>Receives data byte from SPI port</td>
<td>inbyte=spi_read();</td>
</tr>
<tr>
<td>SPI WRITE</td>
<td>Sends data byte via SPI port</td>
<td>spi_write(outbyte);</td>
</tr>
<tr>
<td>SPI TRANSFER</td>
<td>Sends and receives via SPI</td>
<td>inbyte=spi_xfer(outbyte);</td>
</tr>
<tr>
<td>SPI RECEIVED</td>
<td>Checks if SPI data received</td>
<td>done=spi_data_is_in();</td>
</tr>
</tbody>
</table>
- SPI Test System Schematic

SPI.C.DSN
Demonstrates SPI read from slave transmitter and write to slave receiver via master controller.
// Master program U1

#include "16F877A.h"
void main()
{
    int number;
    setup_spi(spi_master); // SPI master mode

    while(1)
    {
        number = spi_read(); // Read SPI input BCD
        spi_write(number);   // Resend BCD code to slave
    }
}

// Transmitter program U2

#include "16F877A.h"
void main()
{
    int sendnum;
    setup_spi(spi_slave);    // SPI slave mode

    while(1)
    {
        sendnum = input_D();   // Get BCD input
        spi_write(sendnum);   // Send BCD code to master
    }
}

// Receiver program U3

#include "16F877A.h"
void main()
{
    int recnum;
    setup_spi(spi_slave);    // SPI slave mode

    while(1)
    {
        recnum = spi_read(); // Read BCD code at SPI port
        output_D(recnum);    // Display it
    }
}
Outline

1. PIC16 UART Serial Link
2. PIC16 SPI Serial Bus
3. I2C Serial Bus
I²C Bus

• The interintegrated circuit (I²C) bus is designed for short-range communication between chips in the same system using a software addressing system.
• It requires only two signal wires and operates like a simplified local area network.
I²C Bus

• The I²C slave chips are attached to a two-wire bus, which is pulled up to logic 1 when idle. Passive slave devices have their register or location addresses determined by a combination of external input address code pins and fixed internal decoding.

• As for SPI, the clock is derived from the instruction clock, up to 5MHz at the maximum clock rate of 20MHz.
I²C Bus

- To send a data byte, the master first sends a control code to set up the transfer, then the 8-bit or 10-bit address code, and finally the data. Each byte has a start and acknowledge bit, and each byte must be acknowledged before the next is sent, to improve reliability.

- The sequence to read a single byte requires a total of 5 bytes to complete the process, 3 to set the address, and 2 to return the data.
## I2C Functions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C WRITE</td>
<td>Send a single byte</td>
<td>i2c_write(outbyte);</td>
</tr>
<tr>
<td>I2C READ</td>
<td>Read a received byte</td>
<td>inbyte=i2c_read();</td>
</tr>
<tr>
<td>I2C STOP</td>
<td>Issue a stop command in master mode</td>
<td>i2c_stop();</td>
</tr>
<tr>
<td>I2C POLL</td>
<td>Check to see if byte received</td>
<td>sbit=i2c_poll();</td>
</tr>
</tbody>
</table>
I²C Test System

U1

IC16F877A

R1 4k7
R2 4k7

DSW1

PIC16F877A

U2

24FC256

SCK A0
SDA A1
WP A2

OFF
ON

+5V

MCLR/Vpp/THV
OSC1/CLKIN
OSC2/CLKOUT
RA0/AN0
RA1/AN1
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
RA5/AN4/SS
RE0/AN5/RD
RE1/AN6/WR
RE2/AN7/CS
RC0/T1OS0/T1CKI
RC1/T1OS1/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5
RD6/PSP6
RD7/PSP7
// Serial I/O using I2C synchronous link

#include "16F877A.h"
#include "i2c.h"

void main()
{
    int sendbyte, lowadd;

    lowadd=0;
    port_b_pullups(1);
    sendbyte=(input_B());

    while(1)
    {
        i2c_start(); // start write cycle
        i2c_write(0xA0); // send control byte
        i2c_write(0x00); // send high address
        i2c_write(lowadd); // send low address
        i2c_write(sendbyte); // send data
        i2c_stop();

        delay_ms(5); // wait for write
        lowadd++; // inc address
    }
}