

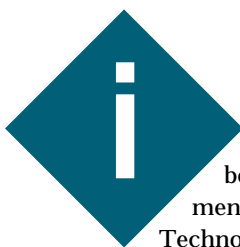
# FEATURE ARTICLE

Duane Mattern

## Synchronous Serial Communication

A PC Link to a SPI/Microwire Device

Duane had an application that required microvolt resolution, so he requested Linear Technology's LTC-2400 eval board. Not only did he like what he got, but he thought you might like it too. Pay attention as he puts the board to use and comes up with a synchronous serial communications link using Win32 subsystem calls and C.



I noticed a number of advertisements from Linear Technology about their new 24-bit ADC, the LTC2400. I have an application that requires microvolt resolution, so I followed the Internet links to the NetSeminar web site to get the lowdown on the LTC2400.

Mike Mayes's online presentation entitled "High-Resolution Data Conversion Application and Techniques" provided an impressive overview of the LTC2400. So I signed up to receive the LTC2400 evaluation board, the DDC228.

The 8-pin LTC2400 has a three-wire synchronous serial communication link that is compatible with SPI and Microwire. The evaluation board interfaces this three-wire serial link to a DE-9 connector for connection to a PC and it uses the PC serial connection to derive power for the board.

The evaluation kit includes a Windows program that appeared to poll the serial port directly to bit-bang the synchronous bitstream. The evaluation program uses the LTC2400 internal clock to drive a communication rate of 19.2 kbps.

Polling the PC serial port to monitor bit updates every 52  $\mu$ s has an

impact on the mouse and keyboard response. Also, I typically use the Windows NT and the evaluation software doesn't run under NT because NT doesn't allow direct access to the serial port.

Because I wanted to have full control of the data anyway, I decided to write my own version of their interface software. I could use an NT port I/O driver to access the serial port under NT, but I didn't want to poll the serial port because of the impact to the system response.

I decided to configure the LTC2400 to power up as a slave device. This setup allows the PC to generate the clock signal, instead of having the PC poll the serial port, thus greatly reducing the impact on the PC response.

In this article, I discuss the pieces of information that I assembled to implement this serial port synchronous link using Win32 subsystem calls and C.

### ASYNCHRONOUS

We're all familiar with asynchronous serial communications on the PC. Hook up a DE-9 or DB-25 connector from the PC comm port to a modem or some other device, configure the port, and you're ready to go. If you are using one of the Microsoft OSs, you might use a dialog box to configure the serial port, like the one shown in Photo 1.

This port configuration sets up the PC's universal asynchronous receiver-transmitter (UART) to perform the required signal timing. The UART adheres to the RS-232 recommended standard. PCs typically use a dual UART, like the 16450 or 16552 from National Semiconductor.

When combined with an appropriate crystal, the UART is capable of rates from 75 to 115 kbps; with 5-8

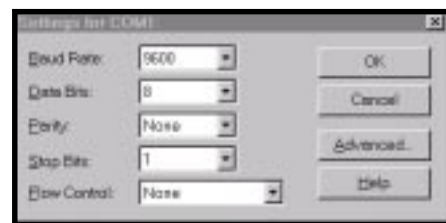


Photo 1—need caption

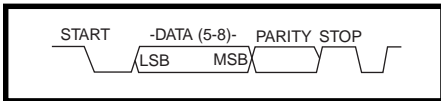


Figure 1—(need caption)

data bits; a parity bit that can be configured as even, odd, none, mark, or space; 1, 1.5, or 2 stop bits; and configurable flow control. The 1.5 stop bit setting only pertains to 5 data bit situations.

The UART performs the bit timing in hardware and provides a FIFO buffer to ease the software timing requirements. RS-232 uses a start and stop bit to bracket the transmission of the 5–8 data bits as shown in Figure 2. The UART provides the proper timing of the bits transmitted between the start and stop bits.

RS-232 uses the nine signal lines listed in Table 1. RX, TX, and GND are the only three signals required to transmit and receive data. Without the other signal lines, byte codes are needed so that each device knows what the other is doing.

In the past, I haven't paid much attention to these electrical signal levels. I found logical 1 as  $-10\text{ V}$  and logical 0 as  $+10\text{ V}$ . The voltage levels for RS-232 signals can range from  $-3$  to  $-25\text{ V}$  for logical 1 and  $+3$  to  $+25\text{ V}$  for logical 0.

The RS-232 standard is a point-to-point protocol. There are other asynchronous schemes—RS-422 and RS-485, for example. The RS-422 standard extends RS-232 to longer distances using a differential voltage measurement instead of measuring the voltage relative to ground and allows for multiple slave devices.

The RS-485 standard extends RS-422 by allowing multiple receivers and transmitters [1]. You can also get more information on RS-485 by checking out Jan Axelson's "Designing RS-485 Circuits" (*Circuit Cellar* 107).

## SYNCHRONOUS

Synchronous serial communications require a separate signal line to carry a clock pulse that triggers the arrival of a new data bit. Thus, rather than using a UART to handle the bit

timing, a separate line, shown as the clock signal in Figure 1, provides the timing information.

Figure 1 presents a periodic signal for the clock pulse, but the signal does not have to be periodic. A fast clock would require appropriate hardware, fast interrupt handling, or fast polling by a listening device to capture the data bits when triggered by the clock signal.

The packing of the data in the bitstream provides one distinction between the various protocols. Three common synchronous communication methods are I<sup>2</sup>C, SPI, and Microwire.

## I<sup>2</sup>C

Philips developed the inter-integrated circuit (I<sup>2</sup>C) in the 1970s [2,3]. If you're curious about I<sup>2</sup>C specifics, see Stuart Ball's article, "Multiprocessor Communications, Part 2: Serial Communication Methods" (*Circuit Cellar* 103).

This synchronous bidirectional serial bus was originally developed for speeds to 100 kbps and recently extended to support speeds of up to 3.4 Mbps. As you see in Figure 2, I<sup>2</sup>C is a two-wire design with a serial data line (SDA) and a serial clock line (SCL). Philips extended the original 7-bit addressing to 10 bits, allowing up to 1024 addresses.

The total bus capacitance limits the number of possible devices on the bus. Because there is no chip select signal line, all devices must monitor the bus all the time, just in case the transmission is addressed to them.

I<sup>2</sup>C also incorporates a level-shifting capability, enabling the interconnection of devices operating from different supply voltages.

To define the transmission start and stop conditions, unique patterns are used.

A high-to-low transition on the SDA line with SCL high indicates a start condition. A low-to-high transition on the SDA line while SCL is high defines a stop condition.

The bus permits multiple masters and includes an arbitration scheme to handle conflicts

between masters. Data transfer must be eight bits and one acknowledge bit, but the number of bytes transmitted per transfer is unlimited.

## SPI

The serial peripheral interface (SPI) is a four-wire interface advanced by Motorola [4]. It consists of two data lines and two control line—master out, slave in (MOSI), master in, slave out (MISO), serial clock (SCK), and slave select (SS). If SPI is used for point-to-point communication between two devices, then you can drop the SS line and SPI becomes a three-wire interface.

SPI was designed to communicate synchronously over short distances at speeds up to 4 Mbps and is oriented for 8-bit transfers. When an SPI transfer occurs, an 8-bit character is shifted out one data pin while a different 8-bit character is simultaneously shifted in a second data pin.

You can view this transfer as a circular 16-bit shift register, composed of two interconnected 8-bit shift registers in the master and the slave. When a transfer occurs, this distributed shift register shifts eight bit positions. Thus, the 8-bit characters in the master and slave are effectively exchanged.

Motorola has other serial interfaces as well. SCI is an asynchronous scheme and SCI+ is an asynchronous and synchronous method that includes SPI. SSPI is a simplified version of SPI.

## MICROWIRE

Microwire was around before 1992 [5]. It also is a four-wire serial inter-

Pin	Abbreviation	Description
1	DCD	Data carrier detect
2	RX	Received signal in, SIN
3	TX	Transmit signal out, SOUT
4	DTR	Data terminal ready
5	GND	Signal electrical ground
6	DSR	Data set ready
7	RTS	Request to send
8	CTS	Clear to send
9	RI	Ring indicator

Table 1—PC serial port DB9 Pin-out(need caption)

face and it includes a serial clock (SK), data-in (DI), data-out (DO), and chip-select (CS) lines.

Microwire is similar to SPI. Its peripherals accommodate digital words of arbitrary bit length, although they typically operate on 16-bit words.

Data into the device should be valid on the rising edge of the clock, and data out of the device is synchronized with the falling edge of the clock. The clock rate depends on the peripheral timing but typically ranges from 250 to 625 kHz, with a minimum logic-high interval of 1  $\mu$ s.

The chip-select pins have a non-standard, active-high polarity. National's Microwire Plus protocol reverses the clock phase for data in and data out and also speeds up the interface timing. Table 2 summarizes the three synchronous communication methods: I<sup>2</sup>C, SPI, and Microwire.

## THE LTC2400

The LTC2400 is a single-ended, 24-bit delta-sigma ADC in a SO-8. For under \$10 you get offset errors of less than 1 ppm and full-scale errors of 4 ppm [6]. The integral nonlinearity is just  $\pm 2$  ppm and the converter's noise characteristic is only 0.3 ppm.

The LTC2400 also provides 120 dB of 50- or 60-Hz noise rejection (selectable), and it does not require an external clock or crystal, (but can use one). The eight-pin package is easy to use with only one power supply line (2.7–5.5 V) and only one ground.

The separation of analog and digital power is handled internally. The three-wire interface is compatible with SPI and Microwire.

In the 60-Hz filter configuration, the LTC2400 has a 24-bit data conversion time of 133 ms for an update rate of 7.5 Hz. Thus, this 24-bit ADC is appropriate for DC and low-frequency measurements.

Figure 3 shows you the evaluation board that Linear Technology provides for the LTC2400. This

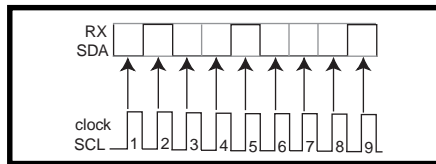


Figure 2—(need caption)

board provides a DE-9 connector for easy connection to a PC. Five pins on the serial port are used: pins 3, 4, 5, 7, 8 for TX, DTR, GND, RTS, and CTS.

The evaluation board receives power from pin 3 of the serial port (TX). This is achieved by setting the UART break control to logic 1, which forces the serial output to the spacing state, which is +10 V on the PC com port.

The LTC2400 only draws 200  $\mu$ A at 5 V when active and 20  $\mu$ A in sleep mode. The CTS line senses the data bits transmitted from the LTC2400's serial data out (SDO) line. The DTR line provides the serial clock (SCK) signal to the LTC2400 from the PC. The RTS line provides the chip select signal. Schmitt triggers buffer the PC serial port from the three-wire serial interface on the LTC2400.

## ASYNCHRONOUS PC INTERFACE

While looking for a solution to the problem of synchronous communication with the PC, I ran across the MAX3100 from Maxim. The MAX3100 is a UART that supports Microwire/SPI. Unfortunately, you have to configure the MAX3100 through the SPI interface using some smart device, like a microcontroller.

Since I could not configure the MAX3100 from the RS-232 side, I ruled out this device for my particular application. Instead, I decided to use the evaluation hardware as received, without any additional hardware.

To solve the communication problem, I placed the LTC2400 into external serial clock mode. This setup avoids the timing complications caused by polling the serial port. The clock signal is provided from the user mode program on the PC, without the use of a device driver.

The LTC2400 has several different operating modes and some of the configuration occurs at powerup or power reset. By configuring the LTC2400 to start up in external serial clock mode, it acts like a slave device. As a slave, the LTC2400 signals the completion of a data conversion and then goes to sleep until communication is requested.

The transmission of data normally consists of 32 bits—two flag bits, one sign bit, one extended mode flag, 24 data bits, and the last four bits are don't use. The LTC2400 is capable of input conversion range from  $-12.5\% V_{REF}$  to  $112.5\% V_{REF}$ , and the extended mode flag indicates when the result is outside of the  $0-V_{REF}$  range.

To save time, rather than transmitting the last four unused bits, a new data conversion cycle can be forced by toggling the chip select line. This change causes the "reduced data output length" mode and tells the LTC2400 to abort the data transfer and start a new conversion immediately. The flowchart in Figure 4 details the command sequence necessary to configure the LTC2400 for external clock and reduced data output length modes.

With the communication link established, the only remaining issue is how to go about obtaining a fixed sampling rate on the PC under Windows. The ADC conver-

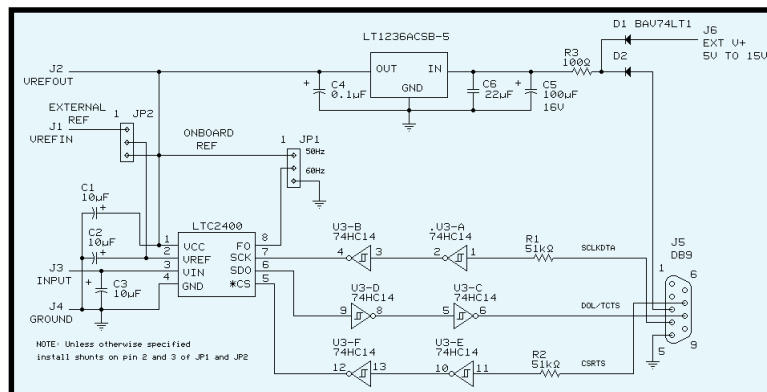


Figure 3—(need caption)

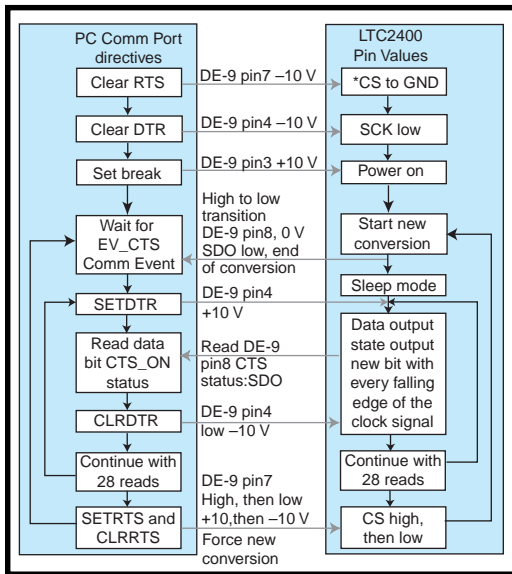


Figure 4—(need caption)

sion times takes approximately 133 ms, and then flags the PC that the conversion is complete.

Initially, I just recorded the conversion completion time using the Win32 high-performance counter. Running the serial communication in a continuous loop with the LTC2400 in external clock mode provides a stable sampling period of about 0.155 s on a 133-MHz PC.

This rate is slower than the fastest rate possible, which is 135 ms (133.33 ms for the conversion and 1.67 ms for the 32-bit serial transmission at 19.2 kbps). However, it does not suffer the performance degradation caused by polling the serial port.

If a consistent sampling period is required, you can use the multimedia timer. Although the multimedia timer does not guarantee a periodic sampling period, it does decrease the average variability [7] when compared to a pure software loop.

Listing 1 shows the Win32 calls used to implement the steps in the flowchart shown in Figure 4. Given the overhead of the Win32 commands, you can call them directly without worrying about sending the signal too fast, given the clock speeds on the current venue of PCs.

Note that the LTC2400 serial interface does not generate a -10-V signal for the PC's RS-232 interface, (normally used for logic 1). In order to sense a logic 1, you have to detect the absence of a +10-V signal. If it ain't false, then in this case it's true.

Table 3 shows the data output from the C program. The program displays the time with a millisecond resolution, the hex version of the masked data, an integer version of the masked data, and the data converted to engineering units of volts.

The data shown in Table 3 is taken with the LTC2400 voltage input shorted to ground with the reference voltage set to the voltage supply  $V_{CC}$ . A longer record of this data is plotted in Figure 5, showing the low internal noise of the LTC2400.

### NOW WE'RE TALKING

This software method for synchronous serial communication is slow, but it enables you to communicate with SPI and Microwire devices from a PC using only user mode code. Certainly using a microcontroller to convert the synchronous stream to an asynchronous would greatly speed up the communication rate, allowing you to use the UART and serial port directly. For the LTC2400, though, that

kind of setup is unnecessary because 133 ms are required for the 24-bit data conversions.

If you need a low-voltage measurement, check out the LTC2400. It can also be used with a multiplexer, so you can obtain multiple high-resolution measurements using a single chip. ■

*Duane Mattern is a instrumentation and controls engineer with 10 years of experience in the areas of modeling, simulation, control system design, and implementation. You may reach Duane at d.mattern@ieee.org.*

## SOFTWARE

The C and Visual Basic project files are available via the *Circuit Cellar* web site.

## REFERENCES

- [1] G. Sakmar, "The right bus for your data highway," *Electronics Tech Briefs* within *NASA Tech Briefs*, p. 1a, February, 1999.
- [2] Philips I<sup>2</sup>C web site, [www.us.semiconductors.com/i2c](http://www.us.semiconductors.com/i2c)
- [3] I<sup>2</sup>C specification, [www.us.semiconductors.com/acrobat/various/I2C\\_BUS\\_SPECIFICATION\\_2.pdf](http://www.us.semiconductors.com/acrobat/various/I2C_BUS_SPECIFICATION_2.pdf)
- [4] SPI Specification, USAR Systems, [www.usar.com/indact/standards/spi.htm](http://www.usar.com/indact/standards/spi.htm)
- [5] Motorola SPI, [www.mot.com/pub/SPS/DSP/LIBRARY/56L811/UM\\_REV0/7.PDF](http://www.mot.com/pub/SPS/DSP/LIBRARY/56L811/UM_REV0/7.PDF)
- [6] Linear Technology, Demo manual DC228, 24-bit A/D demo board, LTC 2400 24-bit high-performance A/D converter.
- [7] D. Mattern, "Soft" Real-Time

Manufacturer	Philips I <sup>2</sup> C	Motorola SPI	NationalMicrowire
Number of wires for bus (without ground)	2—SDA (serial data) and SCL (serial clock)	4—two data and two control lines	4—two data and two control lines
Maximum transmission rates and modes	Normal 100Kbps fast 400Kbps fastest 3.4 Mbps	Up to 4Mbps selected fraction of the master clock	200–625 kHz faster for Microwire Plus
Number of bits transmitted per data byte	9—8 data + 1 ACK bit every byte is obliged to acknowledge	8 data bits shifted before refilling buffer	16 bits is typical, but continuous stream is possible
Addressable # devices	1024	Chip select logic	Chip select logic

Table 2—(need caption)

**Listing 1**—Here's the Key portion of Console Mode Program *Sync\_Comm.c* (The VB program contains the same Win 32api calls).

```
//----- LTC2400 Power-up Configuration for External Serial Clock Mode
if( EscapeCommFunction(h_port, CLRRTS ) == 0) // INIT CS_bar LOW
    printf("CLRRTS failed\n");
if( EscapeCommFunction(h_port, CLRDTR ) == 0) // INIT SCK
    Low (-10volts)
    printf("SETDTR failed\n");
if( EscapeCommFunction(h_port, SETBREAK) == 0) // POWER ON
    (+10volts)
    printf("SETBREAK failed\n");

//----- Wait Loop for end of conversion from LTC2400 (CTS Event)
printf("  Sec      Voltage      Int      Hex      Raw\n");
QueryPerformanceFrequency(&lFreq);
lFreq = lFreq.QuadPart;
dClockFreq = (double)lFreq; // Get the counter frequency
QueryPerformanceCounter(&lHPCount0);
lCount0 = lHPCount0.QuadPart; // Get Initial Starting Count

while(j){
    WaitCommEvent( h_port, &EvtMask, NULL) == 0;
    GetCommModemStatus( h_port, &ModemStatus) == 0; // Read Status
        // High-to-Low transition ends LOW
    if( (EvtMask == EV_CTS) & (!(ModemStatus & MS_CTS_ON)) )
        {
        QueryPerformanceCounter(&lHPCount); // Get Time
        SetCommMask( h_port, 0); // Turn Events Off
        dwData = 0;
        for(i=0; i<28; i++) // Begin synchronous transfer
            {
            dwData = dwData << 1; // shift 1 left
            EscapeCommFunction(h_port, SETDTR ); // SCK HIGH 10v
            GetCommModemStatus( h_port, &ModemStatus); // Read Status
            if(ModemStatus & MS_CTS_ON) dwData |= 0x01; // fill bit0
            EscapeCommFunction(h_port, CLRDTR); // SCK LOW -10v
            }
            // Reduced Data Length Mode
        EscapeCommFunction(h_port, SETRTS); // CS_bar HI, then low
        EscapeCommFunction(h_port, CLRRTS); // Starts new conversion

        lDiff = lHPCount.QuadPart - lCount0;
        dTime = (double)lDiff/dClockFreq; // calc deltaTime

        if(dwData > 33554432)
            dwMaskedData = dwData & 0X01FFFFFF; // positive
        else if(dwData == 0 | dwData == 33554432)
            dwMaskedData = 0; // zero
        else
            dwMaskedData = dwData | 0XFE000000; // negative

        intMaskedData = (int)dwMaskedData; // 32 bit signed integer
        printf("%6.3f %13.4e %6d %8X %8X\n",dTime,
            intMaskedData*EngUnits,dwMaskedData,dwMaskedData,dwData);
        j--; // decrement conversion counter
        }
    SetCommMask( h_port, EV_CTS); // Turn EV_CTS Event back on
}
```

Using Windows NT, a timing study of the Win32 multimedia timer under Windows 95/NT, <http://home.columbus.rr.com/dmattern/realtime/>.

## SOURCES

### LTC2400

Linear Technologies, Inc.  
(408) 432-1900  
Fax: (408) 434-0507  
[www.linear-tech.com](http://www.linear-tech.com)

### MAX3100

Maxim Integrated Products  
(408) 737-7600  
Fax: (408) 737-7194  
[www.maxim-ic.com](http://www.maxim-ic.com)

### 16450, 16552

National Semiconductor  
(408) 721-5000  
Fax: (408) 739-9803  
[www.national.com](http://www.national.com)

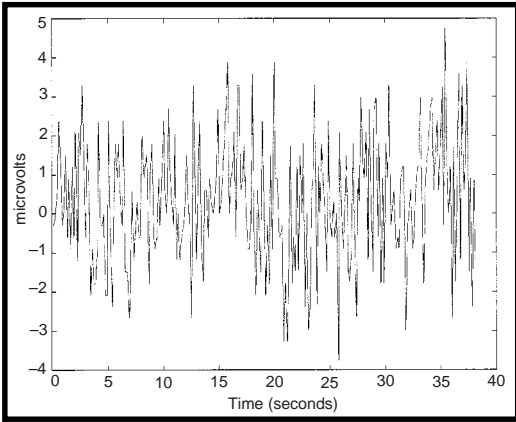


Figure 5—(need caption)

Sec	Voltage	Int	Hex	Raw
0.179	-2.9802e-007	-1	FFFFFFFF	1FFFFFFF
0.336	-1.7881e-006	-6	FFFFFFFA	1FFFFFFA
0.484	1.7881e-006	6	6	2000006
0.632	-2.9802e-007	-1	FFFFFFFF	1FFFFFFF
0.781	2.9802e-007	1	1	2000001
0.929	-2.9802e-007	-1	FFFFFFFF	1FFFFFFF
1.078	8.9407e-007	3	3	2000003
1.227	1.1921e-006	4	4	2000004
1.377	-1.1921e-006	-4	FFFFFFFC	1FFFFFFC
1.525	-2.9802e-007	-1	FFFFFFFF	1FFFFFFF
1.674	4.1723e-006	14	E	200000E
1.822	-5.9605e-007	-2	FFFFFFFE	1FFFFFFE
1.970	8.9407e-007	3	3	2000003
2.118	2.9802e-007	1	1	2000001
2.267	-2.3842e-006	-8	FFFFFFF8	1FFFFFF8
2.415	2.9802e-007	1	1	2000001
2.563	-2.9802e-007	-1	FFFFFFFF	1FFFFFFF
2.711	0.0000e+000	0	0	2000000
2.859	-2.0862e-006	-7	FFFFFFF9	1FFFFFF9
3.008	2.9802e-007	1	1	2000001

Table 3—(need caption)