

www.circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

#114 JANUARY 2000

COMMUNICATIONS

Synchronous Serial
Communication

Resistive Touchscreens

Designing Neural Networks

Embedded CAN



CIRCUIT CELLAR ONLINE

Double your technical pleasure each month. After you read *Circuit Cellar* magazine, get a second shot of engineering adrenaline with *Circuit Cellar Online*, hosted by ChipCenter.

— FEATURES —

Voice-Recognition Controlled Sailboat

Mike Smith, Todd Turner, and Steve Alvey

For his engineering design project at the University of Calgary, Todd and his team created a prototype for a voice-recognition system that enables quadriplegic sailors to independently control a full size Martin 16 sailboat. Batten down the hatches for a storm of information about this practical and resourceful project.

Building a Embedded Timing System

Jamie Pollack

Although development systems are robust and have a great diversity of hardware and software, a few applications and some wire wrap can go a long way. Jamie used a MC68HC11 series microcontroller to design and develop a system that will fit into even the tightest of budgets.

RC Servo Control via TPU

Jeff Loeliger

If you've ever wanted to control RC servos without any additional hardware, then pay attention to this project because that's just what Jeff has done. By designing a time processor unit (TPU) function, he provides an easy-to-use interface and puts some of the fun back into servo control.

Resource Links

- [H-Bridges and Class-D](#)
- [Peltier Thermoelectric Coolers](#)

Bob Paddock

- [Flash Memory](#)

Ben Day

Test Your EQ

8 Additional Questions

— COLUMNS —

Considering the Details

The Basics of Thermocouples

Bob Perrin

This month, Bob sets out to shed some light on the mysteries of the thermocouple by explaining hot junctions, cold junctions, and dissimilar metals. As usual, he provides enough circuits and information to get you ready to design a thermocouple into your next project.

Lessons from the Trenches

Defects For Sale

George Martin

Although it might not be the best slogan to put on your next batch of business cards, if you design custom hardware and software, defects (bugs, failures, etc.) are a reality of every project. According to George, the key to staying in business is understanding your ability to design, find, and repair defects.

Silicon Update Online

The Captain is Back

Tom Cantrell

Captain ZiLOG is back! Armed with the new eZ80 and Z80S183, he returns to wipe out 20 years of somewhat lackluster performance by ZiLOG. As Tom explains, these new chips certainly have the potential to boost ZiLOG. Stay tuned to find out more about the Captain's return.

WWW.CIRCUITCELLAR.COM/ONLINE
Table of Contents for December 1999

GRAND PRIZE

FOR THE BEST DESIGN THAT INCLUDES BOTH
INTERNET CONNECTIVITY AND INTERNET APPLICATION:

\$5000 CASH Plus

- MPLAB-ICE 2000 EMULATOR
- MPLAB-C COMPILER
- PRO MATE II PROGRAMMER

Each CATEGORY:

1st PLACE: \$4000

CASH plus PICSTART Plus AND MPLAB-C

2nd PLACE: \$3000

CASH plus PICSTART Plus AND MPLAB-C

3rd PLACE: \$2000

CASH plus MPLAB-ICD

INTERNET
PIC[®] 2000
CONTEST

www.circuitcellar.com/pic2000

Deadline is May 1, 2000


12 Synchronous Serial Communication
A PC Link to a SPI/Microwire Device
Duane Mattern


20 Reach Out and Touch
Designing a Resistive Touchscreen
Tom Dahlin

26 Neural Stamp
A Low-Cost Neural Network Processor
Robert Lacoste

36 Embedded CAN Can
Michael Howard

40 Embedded Living
Tuning Up the HCS-II
Mike Baptiste

62  **MicroSeries**
High-Definition TV
Part 3: DTV System Architecture
Mark Balch

70  **From the Bench**
Speed Racer
Virtual Speed with the SX
Jeff Bachiochi

78  **Silicon Update**
Atmel Gets Tiny
Tom Cantrell

Task Manager 6
Random Thoughts
Steve Meyst

New Product News 8
edited by Harv Weiner

Test Your EQ 83

Advertiser's Index 95
February Preview

Priority Interrupt 96
Steve Ciarcia
It Starts with an Idea

48 Nouveau PC
edited by Harv Weiner

50 RPC **Real-Time PC**
A Matter of Time
Part 1: Accurate Timing and Frequency
Ingo Cyliax

55 APC **Applied PCs**
The Mockingbird Trial
PIC vs. 80188
Fred Eady

EMBEDDED PC

INSIDE ISSUE 114

Random Thoughts



finally broke down and bought a DVD player this week. The last straw was a trip to the video store during which I wandered up and down the aisles, seeing only blank spaces where all the good videos were supposed to be, and alongside these spaces, shiny new DVD packages of the same movies. \$249 later, I have added to the eight different formats of recordable media in the house. I was against getting a DVD player, in part for that reason, but also because I didn't think it would be much of a boost to my video-watching experience. Though the format is digital, the signal is still converted back to analog. I was perhaps a little more pleased than I thought I'd be with the image quality, an onscreen GUI adds a touch of familiarity, and I don't miss rewinding tape. I admit there are enhancements here, but it is the additional features I find intriguing. That there are additional features speaks to there being room in the MCU to implement more of them. I did not find the feature set to be uniform across all brands and titles, however. Some offer a zoom capability. Most offer some kind of "alternate camera" view and all of them have something like a random play function. Let's look at this last feature.

You probably know that Hollywood productions are assembled into finished products from raw stock. The raw stock is created asynchronously in terms of the plot line; they shoot scenes in a sequence that makes economic and logistical sense. Depending on the budget, they will film sequences many times, from many angles. Then everything is stitched together for the theatrical release.

Random play seems like a feature cribbed from an audio CD controller. What other reason could there be? In the audio realm, it makes sense to shuffle the tracks on a particular release. Even though the artist, producer, production engineers, and others put plenty of thought into the organization of the whole CD, random play can expand the listening experience by confounding the expectations of the listener. This succeeds because there is nothing inherently linear in most music CDs.

The same can not be said for the medium of film on a DVD. The data on DVD is organized into chapters and these tend to be 3-4 minute segments of a plot element. They are numbered, and next to each numbered chapter is a title taken from the dialogue of the film. So, what happens when you enable random play on a typical movie? Well, when I tried it, an asteroid smashed into the earth killing most everyone, then I watched characters I didn't know anything about doing things that made no sense. That is, I saw this happen on the screen, not in real life.

I have to believe that random play is not a feature that is suited very well to video. It's there on the DVD platform because there was room for the routine and it was easy to implement. Applied in a broader scope, its presence there signifies that there is a good amount of unused or misused capability in all of the MCUs of like devices throughout the world. This bodes well for inventive articles at Circuit Cellar.

Until the DVD feature set gets straightened out, feel free to enjoy this issue however you please. Skip ahead to Robert Lacoste's article on embedded neural networks, start with Tom's discussion of the new Atmel chip and work your way backwards, or read it straight through from cover to cover. Circuit Cellar is completely random play-friendly.

steve.meyst@circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

EDITORIAL DIRECTOR/PUBLISHER

Steve Ciarcia

ASSOCIATE PUBLISHER

Sue Skolnick

MANAGING EDITOR

Steven Meyst

CIRCULATION MANAGER

Rose Mansella

TECHNICAL EDITORS

Michael Palumbo Rob Walker

CHIEF FINANCIAL OFFICER

Jeannette Ciarcia

WEST COAST EDITOR

Tom Cantrell

CUSTOMER SERVICE

Elaine Johnston

CONTRIBUTING EDITORS

Mike Baptiste Ingo Cyliax Fred Eady

ART DIRECTOR

KC Zienka

George Martin Bob Perrin

GRAPHIC DESIGNER

Jessica Nutt

NEW PRODUCTS EDITOR

Harv Weiner

STAFF ENGINEERS

Jeff Bachiochi

PROJECT EDITORS

Steve Bedford Janice Hughes
Elizabeth Laurençot David Tweed

John Gorsky

EDITORIAL ADVISORY BOARD

Ingo Cyliax Norman Jackson
David Prutchi

QUIZ MASTERS

Tak Auyeung
Benjamin Day
Bob Perrin

Cover photograph Ron Meadows—Meadows Marketing

PRINTED IN THE UNITED STATES

ADVERTISING

ADVERTISING SALES MANAGER

Bobbi Yush
(860) 872-3064

Fax: (860) 871-0411
E-mail: bobbi.yush@circuitcellar.com

ADVERTISING COORDINATOR

Valerie Luster
(860) 875-2199

Fax: (860) 871-0411
E-mail: val.luster@circuitcellar.com

ADVERTISING CLERK

Sally Collins

CONTACTING CIRCUIT CELLAR

SUBSCRIPTIONS:

INFORMATION: www.circuitcellar.com or subscribe@circuitcellar.com
TO SUBSCRIBE: (800) 269-6301 or via our editorial offices: (860) 875-2199

GENERAL INFORMATION:

TELEPHONE: (860) 875-2199 FAX: (860) 871-0411
INTERNET: info@circuitcellar.com, editor@circuitcellar.com, or www.circuitcellar.com
EDITORIAL OFFICES: Editor, Circuit Cellar, 4 Park St., Vernon, CT 06066

AUTHOR CONTACT:

E-MAIL: Author addresses (when available) included at the end of each article.
ARTICLE FILES: ftp.circuitcellar.com

For information on authorized reprints of articles,
contact Jeannette Ciarcia (860) 875-2199 or e-mail jciarcia@circuitcellar.com.

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 0896-8985) and Circuit Cellar Online are published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate USA and possessions \$21.95, Canada/Mexico \$31.95, all other countries \$49.95. Two-year (24 issues) subscription rate USA and possessions \$39, Canada/Mexico \$55, all other countries \$85. All subscription orders payable in U.S. funds only via VISA, MasterCard, international postal money order, or check drawn on U.S. bank.

Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301.

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

Circuit Cellar® makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar® disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar®.

Entire contents copyright © 1999 by Circuit Cellar Incorporated. All rights reserved. Circuit Cellar is a registered trademark of Circuit Cellar Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

NEW PRODUCT NEWS

Edited by Harv Weiner

STEP-DOWN SOLAR BATTERY CHARGERS

The **Power Advantage 302** and **304** chargers are step-down chargers that decrease incoming solar-generated voltage to optimize charging currents for battery banks. They deliver up to 25% more DC power to the solar electrical system's batteries from photovoltaic (PV) panels. By increasing current to the batteries, the system maximizes usable power from solar panels to increase overall solar energy system efficiency.

The chargers are ideal for retrofit applications because they are configured the same way as buildings with existing charge controlling systems.

Both models are able to optimize the battery charging process by finding the maximum power point of the solar panels and constantly read-

justing for changes in sunlight, temperature, and battery voltage. An LCD gives the user real-time system status such as state-of-charge and time left on battery based on current usage. The LCD also

offers the user digital precision to adjust set points such as charging voltage. The software runs on any PC equipped with Microsoft Windows 95/98 and a serial port.

The chargers are able to monitor energy production, energy consumption, battery use, battery temperature, and illumination of the solar cells.

The Power Advantage 302 lists for **\$449** and the Power Advantage 304 lists for **\$549**.

Fire, Wind, & Rain Technologies LLC
(520) 526-1133
Fax: (520) 5274664
www.firewindandrain.com



NEW PRODUCT NEWS

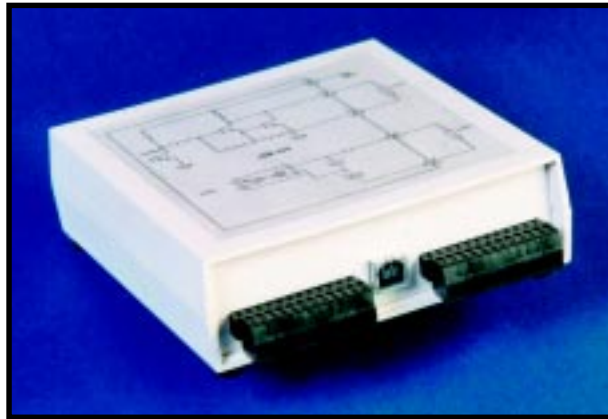
USB OPTO I/O MODULE

The **JSB-320 USB Opto I/O** module allows a user to interface to opto-isolated inputs and outputs from any universal serial bus (USB). Several configurations of output and input up to 32 points are available and up to 127 modules per USB channel can be addressed. Programming the module is done with simple commands from the host computer. Any programming language that supports USB communications can be used to control the input and output. The module replaces internal PC-based plug-in cards in various test, control, and measurement applications.

Output is rated at 1 A @7-30 VDC and the input range is 10-30 V. The inputs include reserve protection diodes and are isolated in groups of two to 2500 VDC. The unit sup-

ports a USB data rate of 1.5 mbps and operates over a voltage range from 4.1 to 5.25 V (self-powered from the host). Plug-in style terminal block connectors allow quick hook up.

Single unit pricing for the JSB-320 ranges from **\$205 to \$299** and includes sample interface source code, Windows 98 driver software, and a USB cable.



J-Works, Inc.
(818) 361-0787
Fax: (818) 270-2413
www.j-works.com

NEW PRODUCT NEWS

SMART BATTERY ICS

PowerSmart's **PS331** is a smart battery IC that supports packet error checking per v.1.1 of the system management bus specification. Its integrating, self-calibrating 14-bit A/D ensures precise measurements of current, voltage, and temperature to enable remaining run-time predictions to within 1%. Patented, self-learning algorithms compensate for self-discharge, temperature, charge/discharge efficiencies and other factors, and an auto-zero offset correction feature automatically compensates for accuracy drift during usage. An advanced 8-bit RISC microprocessor enables exceptionally fast data computations.

The PS331 enables a host device, which appends a packet error code (PEC) at the end of each message transfer, to check the integrity of the data communication. For each read or write bus transaction, an 8-bit

cyclic redundancy check (CRC-8) is used to calculate a frame check sequence. Only 1 ms is required for the CRC-8 calculation of a single data byte. The PS331 is also capable of communicating with devices that do not implement PEC error correction protocols.

The PS331 is packaged in a standard 28-pin 209-MIL SSOP and is priced at under \$4 in large quantities. Evaluation kits are also available for purchase at \$199. Kits include all necessary hardware and software for programming of battery control algorithm parameters and for customizing preprogrammed cell data models.



PowerSmart, Inc.
(203) 925-1340
Fax: (203) 925-1714
www.powersmart.com

NEW PRODUCT NEWS

8051 IN-CIRCUIT EMULATOR

Signum Systems has released a new POD for support of TEMIC Semiconductor's TS80C32X2. The **POD32X2-60** in the company's USP-51A emulator fully emulates the device in targets at the maximum clock speed of 64 MHz. The POD uses the standard TS80C32X2 device for emulation and can switch between internal and external clock and power sources via software control.

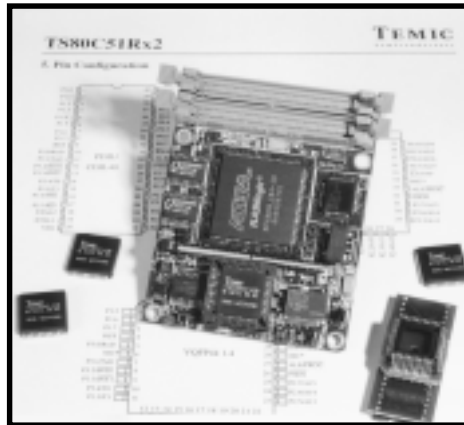
The TS80C32X2 microprocessor uses the X2 feature to double the internal operating frequency, allowing up to 60 MHz of equivalent speed with a 30-MHz external quartz at 5 V. The X2 feature shortens the typical instruction execution time from 400 to 200 ns

To allow for this high-speed in-target emulation, POD32X2-60 plugs directly into the user's 40-pin processor socket. The POD supports 44-PLCC foot-

prints with an optional socket adapter. It also supports the ONCE (on-chip emulation) mode of emulation which facilitates testing and debugging of the microcontroller without the device having to be removed from the circuit. This mode requires an optional ONCE adapter that clips over the microcontroller and allows the POD32X2-60 to be plugged into it.

POD32X2-60 fully supports debugging of target power-up and power-down sequences, and will not take any extra power from the target board, nor will it supply any current to the target board when the target is off.

Prices for POD32X2 are **\$795**, **\$895** and **\$1,095** for up to 30, 40, and 60 MHz, respectively. The 44-PLCC adapter is priced at **\$200** and the 44-PLCC ONCE clip-over adapter is **\$250**.



Signum Systems Corp.
(805) 523-9774
Fax: (805) 523-9776
www.signum.com

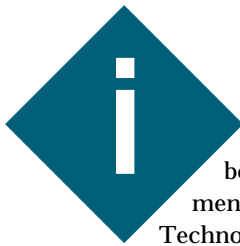
FEATURE ARTICLE

Duane Mattern

Synchronous Serial Communication

A PC Link to a SPI/Microwire Device

Duane had a project that needed microvolt resolution, so he requested Linear Technology's LTC-2400 eval board. Pay close attention as he 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 μ 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 like the one shown in Photo 1 to configure the serial port.

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 data bits; a parity bit that can be configured as even, odd, none, mark, or



Photo 1—An example of a typical Windows dialog box used to configure serial port communications.

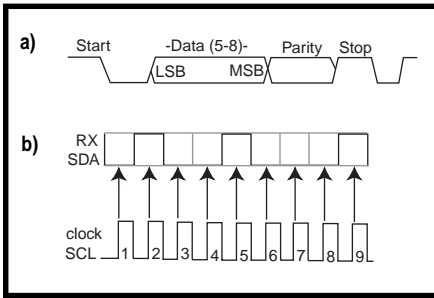


Figure 1a—In asynchronous communication data sequences, a byte of data is bracketed by a start and stop bit. **b**—Note that synchronous communication data is valid on the rising edge of the clock signal. A periodic clock is not required.

space; 1, 1.5, or 2 stop bits; and configurable flow control. The 1.5 stop bit setting only pertains to five 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 1b. 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 paid little attention to these electrical signal levels. I found logical 1 as -10 V and logical 0 as $+10\text{ V}$. The voltage levels for RS-232 signals can range from -3 to -25 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 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 2, provides the timing information.

Figure 2 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 I²C, SPI, and Microwire.

I²C

Philips developed the inter-integrated circuit (I²C) bus in the 1970s [2,3]. If you’re curious about I²C specifics, see Stuart Ball’s article, “Multi-processor 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, I²C 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

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—Of these PC serial port DE9 Pin-out assignments, pins 2, 3, and 5 are the minimum subset for bi-directional communication.

line, all devices must monitor the bus all the time, just in case the current transmission signal is addressed to them.

I²C 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.

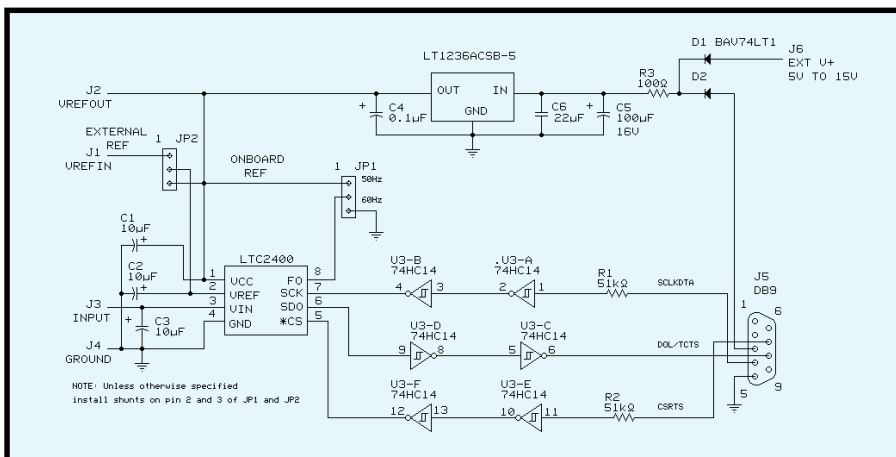


Figure 2—The Linear Technology DC228 evaluation board draws power from pin 3 of the serial connector.

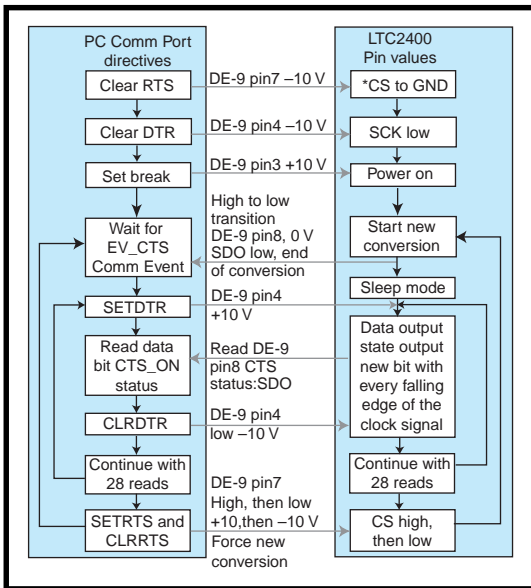


Figure 3—Running the LTC2400 in external clock mode allows the PC to control the clock.

SPI

The serial peripheral interface (SPI) is a four-wire interface advanced by Motorola [4]. It consists of two data lines and two control lines—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 on 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 variations of serial interfaces as well. For instance the SCI is an asynchronous scheme while the SCI+ is an asynchronous and synchronous method that includes SPI. SSPI is just a simplified version of SPI.

MICROWIRE

Microwire was around before 1992. It also is a four-wire serial interface 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 μ 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: I²C, 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 ± 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 2 shows you the evaluation board that Linear Technology provides for the LTC2400. This 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) by setting the UART break control to logic 1. This forces the serial output to the spacing state, which is +10 V on the PC comm port.

The LTC2400 only draws 200 μ A at 5 V when active which drops to only 20 μ 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.

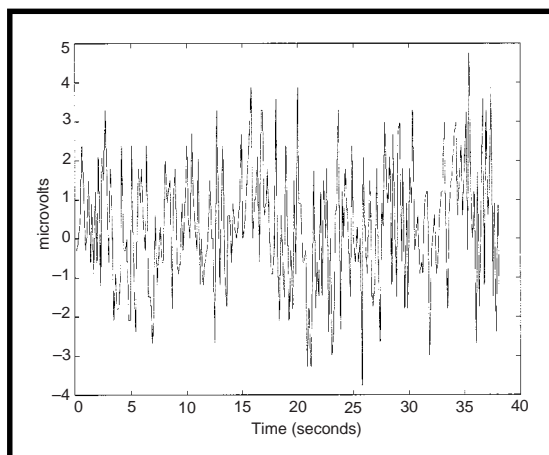


Figure 4—This 40-s time slice of output data demonstrates the low internal noise of the LTC2400.

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 (-10 volts)
    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);
l1Freq = lFreq.QuadPart;
dClockFreq = (double)l1Freq; // Get the counter frequency
QueryPerformanceCounter(&lHPCount0);
l1Count0 = lHPCount0.QuadPart; // Get Initial Starting Count

while(j){
    WaitCommEvent( h_port, &EvtMask, NULL) == 0;
    GetCommModemStatus( h_port, &ModemStatus) == 0;
    // Read Status
    // High-to-Low transistion 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
        l1Diff = lHPCount.QuadPart - l1Count0;
        dDtime = (double)l1Diff/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",dDtime,
intMaskedData*EngUnits,dwMaskedData,dwMaskedData,dwData);
        j--; // decrement conversion counter
    }
    SetCommMask( h_port, EV_CTS); // Turn EV_CTS Event back on
}
}
```

Feature	Philips I ² 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—Each asynchronous serial communications protocol has a weakness.

Because 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 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 that are configured at powerup or power reset. When

configured to start up in external serial clock mode, the LTC2400 acts like a slave device. As a slave, the LTC2400 signals the completion of a data conversion, 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 not used. 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. Figure 3 shows the command sequence needed 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 establish a fixed sampling rate on the PC under Windows. The ADC conversion times take approximately 133 ms, and then flags the PC that the conversion is complete.

Duane Mattern is an 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 I²C web site, www-us.semiconductors.com/i2c
- [3] I²C specification, www-us.semiconductors.com/acrobat/various/I2C_BUS_SPECIFICATION_2.pdf
- [4] SPI Specification, USAR Systems, www.usar.com/indact/standards/spi.htm
- [5] Motorola SPI, 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 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

MAX3100

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

16450, 16552

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

Sec	Voltage	Int	Hex	Raw
0.179	-2.9802e-007	-1	FFFFFFF	1FFFFFF
0.336	-1.7881e-006	-6	FFFFFFFA	1FFFFFFA
0.484	1.7881e-006	6	6	2000006
0.632	-2.9802e-007	-1	FFFFFFF	1FFFFFF
0.781	2.9802e-007	1	1	2000001
0.929	-2.9802e-007	-1	FFFFFFF	1FFFFFF
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	FFFFFFF	1FFFFFF
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	FFFFFFF	1FFFFFF
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—The data output from the C program indicates that the conversions are processed approximately every 150 ms.

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 3. The overhead of the Win32 commands combined with the clock speeds on the current crop of PCs means you can call them directly without worrying about sending the signal too fast.


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.

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 4, 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 one 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 an interval of 133 ms is 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. 

FEATURE ARTICLE

Tom Dahlin

Reach Out and Touch Designing a Resistive Touchscreen

While working on a design assignment that involved an expensive off-the-shelf touchscreen interface, Tom began to consider rolling his own. Watch as he points out some of the tricks of working with resistive touchscreens.



The use of touchscreens as computer interface devices has become widespread. I don't think there are many readers out there who have not used an ATM machine with a touchscreen or at least played with one of the Apple Newton-type personal digital assistants. The popularity of touchscreens is in a large part due to the simplicity they bring to user interfaces. They eliminate the need for a large keyboard and offer the benefit of context specific menu choices.

I was recently involved in the design of a machine controller for an industrial application. We liked the advantages of a touchscreen interface, and prototyped the system using an industrial embedded PC, and a commercially available motion-control board. We purchased an off-the-shelf industrial LCD display/touchscreen combination and connected it to the PC's VGA and COM1 interfaces. We then developed the application software in VB and got the machine up and running in short order. Life was good.

The euphoria was short-lived however. Economic reality set in, and we realized we couldn't justify spending over \$2k on

the off-the-shelf LCD/touchscreen for the commercial product. Besides that, the mechanical form factor was wrong. With a short development cycle ahead of us, life was looking not so good.

We quickly realized that we would either have to roll our own LCD display/touchscreen, or drop back to a character-oriented LCD/16 key keypad for the user interface. While it is not practical for a small company to consider making its own touchscreen (the glass part that is), it is possible to save both money and package size by purchasing only the raw touchscreen, using a standard LCD or CRT, and designing your own controller and package. This article will show you a few of the ways of designing the touchscreen-to-computer interface.

TYPES OF TOUCHSCREENS

Not all touchscreens are created equally. There are two primary technologies used today—resistive and capacitive sense. There are others, such as IR scanning, acoustic wave, and electromagnetic technologies. Although they all have their merits, resistive- and capacitive-sense technologies have emerged as favorites thanks to their low cost and high resolution.

Resistive-sense technology has the added advantage of being able to detect a touch from a rubber-gloved finger, something that is a problem for capacitive type touchscreens. The other technologies mentioned are used when they offer a unique advantage, such as the sense-before-touch feature that IR scanning provides.

For a discussion of the relative pros and cons of each technology, you might want to read the application notes written by David Blass of Sharp

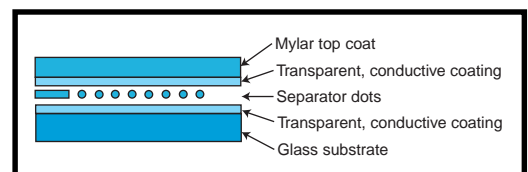


Figure 1—Most resistive touchscreens have a construction similar to the one shown here. Two conductor layers are separated by a layer of tiny dots. The dots allow the two planes to make contact when force is applied to the top layer.

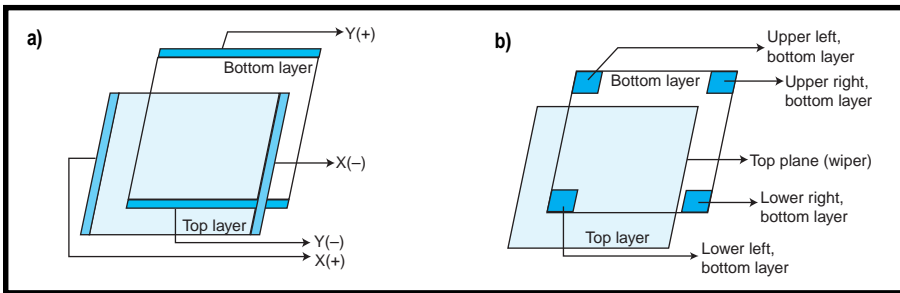


Figure 2a—A 4-wire touchscreen has bus bars on the right and left edges of one layer, and on the top and bottom edges of the other layer. **b**—A 5-wire version has four leads, each connected to the corner of the bottom layer, and one lead connected to the top layer that works like a potentiometer wiper.

[1], and another by VJ Kuroodi of Tritech [2].

AN INSIDE LOOK

Resistive touchscreens almost all start with a glass or hard plastic substrate, onto which a thin, transparent conductive layer (usually ITO) has been applied. Figure 1 illustrates a typical touchscreen cross-section.

A fine grid of spacer micro dots is then applied and another layer of a conductive-coated flexible plastic (usually Mylar) is laid on top. You end up with two transparent conductive planes of material separated by a few thousandths of an inch. Pressure from a stylus or finger causes the two planes to make electrical contact and forms the means of sensing the touch.

There are two commonly used types of resistive touchscreens. These are called 4-wire and 5-wire. In a 4-wire resistive touchscreen, the two planes each have two wires connected to opposite ends.

For example, the *x* plane would have wires connected to the left and right edges, and the *y* plane would have wires connected to the top and bottom edges (see Figure 2). In operation, a

controller must first apply a voltage across the *x* plane thereby forming a gradient because of the resistive coating.

A touch is sensed by using the *y* plane as an input to an ADC and detecting a voltage when the two planes are forced together. The ADC reading will vary as a

function of the *x* (right/left) position of the touch.

The *y* position is then calculated by removing the voltage from the *x* plane and applying it to the *y* plane, from top to bottom. The *x* plane is then used as a pickoff and its output is routed to the ADC.

In a 5-wire resistive touchscreen like the one shown in Figure 2b, the operation is similar, but the alternating *x* and *y* fields are applied across only one plane, and the other plane is used solely as a pickoff. Thus, one wire goes to each corner of the bottom plane, and a fifth wire is connected to the top plane.

To read an *x* position, the controller applies a voltage to the left two corners and ground to the right two. The fifth wire would go to the ADC. To read a *y* position, the controller grounds the bottom two corners and applies a voltage to the top two.

SMORGASBORD OF OPTIONS

There are several different ways of interfacing the glass touchscreen to your system. I'm going to show you four methods I've used with success. Two of the methods use a combin-

ation of a PIC and a touchscreen controller chip, one uses only a PIC, and the last uses no processor at all. Which method you use depends on your system requirements for scan rate, accuracy, and cost.

Interfacing to either a 4- or 5-wire touchscreen is easy thanks to a pair of chips from Burr-Brown. The ADS7843 is designed to interface to a 4-wire touchscreen, and the ADS-7845 to a 5-wire. The devices have identical hardware and control interfaces, differing only in the type of touchscreen they interface to.

Figures 3 and 4 show examples of circuits using a PIC and the devices to interface to both 4- and 5-wire touchscreens. Let's take a look at each circuit and chip individually, starting with the 4-wire device.

The ADS7843 is a single-chip interface to a 4-wire touchscreen. At its core is a 12-bit successive approximation analog-to-digital converter (ADC). It performs all the front-end analog multiplexing necessary to generate the required voltage gradients across the touchscreen planes and switch the pickoff into the ADC.

As shown in Figure 3, a PIC and an RS-232-level shifter are all that's required to build a 4-wire interface to a PC com port. The PIC-to-ADS7843 interface is simple, needing only three lines—clock, data, and chip select.

In the example circuit, the data input and output lines from the ADS7843 are tied together via a 10-k Ω resistor. This arrangement allows the use of a single PIC I/O line to handle both. Also shown is a PIRQ, or pen interrupt signal that can alert the PIC to the presence of a touch (or pen in a PDA application), and a BUSY signal that enables the PIC to monitor the status of the ADC conversion. The latter two signals are not used in my application, but are brought into the PIC for future use, if needed.

Figure 5 shows the logic diagram and

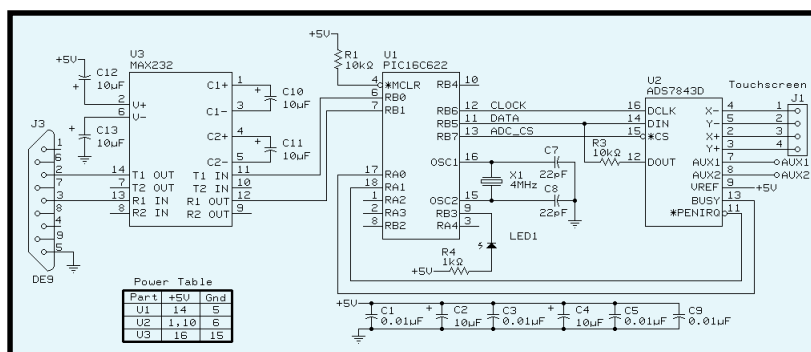


Figure 3—3 chips are all you need for this circuit. The ADS7845 handles the analog functions and the PIC performs the sequencing, scaling, and messaging formatting. The MAX232 handles the RS-232 level shifting.

timing of the PIC interface. This timing is called 24-clock mode, referring to the single byte of control info sent to the device and the two bytes returned. Other modes are also available and overlap the shifting of data in and out to save transit time, providing an ability to get more samples per second.

To read the touchscreen, the PIC must send an 8-bit control byte to the ADS7843. This byte always has its most significant bit or start bit set. The next three bits (A2, A1 and A0) specify whether we want to read x, y, or one of the two Auxiliary ADC inputs.

The auxiliary inputs, with proper signal conditioning, can be connected to any system analog value you might want to read (e.g., a battery voltage). The next bit is called MODE and is set to zero if you want a 12-bit conversion, or a one if you want an 8-bit conversion. The following bit is called SER/DFR and is set to zero if you want to use a differential voltage reference (normally preferred) or is set to one for a single-ended type.

Lastly, the final two bits in the control byte are called PD1 and PD0. These are the power down mode select bits. If both are low, the device is powered down between conversions to save power for portable applications. If both are high, then the device is always enabled. An important fact is that the PIRQ, or pen interrupt is

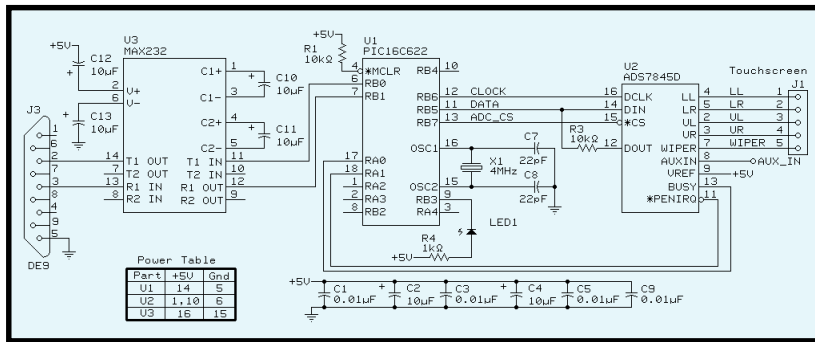


Figure 4—By swapping an ADS7845 into the circuit shown in Figure 3, we can create a 5-wire controller. The ADS7843 and ADS7845 are 12-bit devices that provide resolution capability up to 1/4096 of the touchscreen width.

disabled when either of the two bits is set high. Thus, you can't use the PIRQ if you leave the device always powered.

So, if the PIC wants to read the x channel, it sends a \$93 to the ADS7843. This selects not only that channel, but also a 12-bit conversion, differential referencing, and non-powerdown operation. After clocking out these eight bits to the ADS7843, in return, the PIC clocks in 16 bits that contain the 12-bit result and four zero-filled trailer bits. To read the y channel, the PIC performs the same operation, only sending a \$D3 for the control byte.

TAKING THE 5-WIRE ROUTE

Burr-Brown has the ADS7845 device for a 5-wire touchscreen interface. Like its cousin the ADS7843, it connects to your microprocessor via a simple serial interface. It too uses a 12-bit ADC. The pinouts of the chips are nearly identical. The 5-wire device uses one of the two spare analog inputs available on the 4-wire device to accommodate the fifth wire input.

Another way to interface to a 5-wire touchscreen is to do it all with a PIC. As shown in Figure 6, this method results in a low parts count. The thing that makes this design easy is the fact that we can control the four corners of the bottom plane with the PIC's digital drivers and run the sense-

plane wire directly into the PIC's on-chip ADC. Thus eliminating the need for fancy analog multiplexing using external FETs.

As you can see, we used four PIC I/O lines (RB2-5) to connect to the four corners, labeled UL (upper left), LL (lower left), UR (upper right), and LR (lower right). To generate a left-to-right voltage gradient, the PIC sets UL and LL to a low (0 V) and sets UR and LR high (5 V). It then performs an ADC conversion, reading AN0.

The presence of a voltage greater than a few counts indicates a touch. The bleed resistor R5 in Figure 6 pulls the ADC input low, so we have no problem knowing that a touch has occurred.

To generate a top-to-bottom voltage gradient, the PIC simply sets UL and UR to high and LR and LL to low. Note that LL is always low and UR is always high. Although we could hardwire them to ground and +5 V respectively, it's better to allow the PIC to do this to preserve balanced levels on all four corners.

Once the PIC has secured readings for the x and y directions, it can adjust for offset and scale factors, and determine the x and y positions. The PIC I used was a PIC16C71 with an 8-bit ADC, which works for applications where positional accuracy is not important. Newer members of the PIC family have better accuracy and would improve this design.

LOOK MA, NO PROCESSOR

If you want a simple interface to a 4-wire glass and you can live with a predefined output format, the TriTech TR88L811 chip makes it possible to

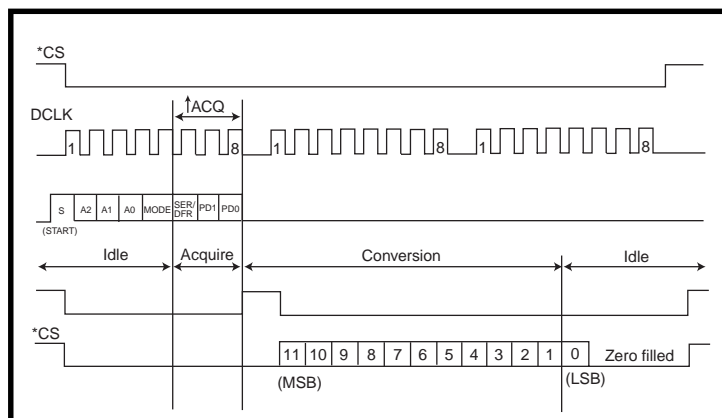


Figure 5—eight clocks are used to shift out a control byte from the PIC to the device and another 16 clocks are used to retrieve the result. Since the data input and output lines are not simultaneously active, it's possible for them to share the same microcontroller I/O pin.

go from the glass to a serial bitstream with only one chip. If you have an extra serial channel available and only need 10 bits of positional accuracy, then this may be the way to go.

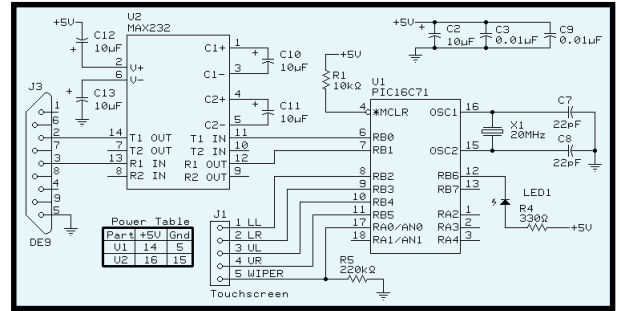
The TR88L811 is designed for standalone applications and requires only a 1.8432-MHz crystal and an RS-232-level shifter to form a complete interface that you can attach to a spare PC COM port.

Figure 7 shows an example circuit that steals its power from the PC's COM port. The TriTech device scans the touchscreen continuously and sends a serial data packet out of its TxD pin when a touch is detected. The data packet, sent at 19,200 bps, contains five bytes—a header and two bytes each of *x* and *y* position. The position is resolved to 10 bits, which is adequate for most applications.

If a touch is maintained, the chip will send data out at a rate of approximately 200 coordinate pairs per second. The main advantage of this device is that it requires no firmware. As long as you can live with the 10-bit resolution and can work with its output format, then it's a turnkey solution.

You can buy the raw touchscreen glass from several manufacturers. I've

Figure 6—Four of the PIC's digital lines are used to provide *x* and *y* voltage gradients. A single analog input is used to determine the contact point by measuring the pick-off voltage in both the *x* and the *y* planes.



had a good experience dealing with the Bergquist Company and I've also been successful interfacing to glass made by Elo and Microtouch.

DEVELOPMENT NOTES

I began this project by first locating the touchscreen-interface chips. I then needed a quick and dirty means of testing their functionality, which I did using a Basic Stamp II device with a serial LCD attached. It was a simple matter to interface the chips to the Stamp, and then use the interactive Stamp-development environment to debug the chip interfaces.

Of course, a \$50 Basic Stamp is not a good solution for a production product. The production hardware was designed around a lower cost PIC, the 16C622. I intended to discard the Stamp code and write the production

code in C. However, I had heard about a compiler for the Stamp BASIC, and decided to give it a try. The results were good and enabled me to use most of the stamp code with little modification. I've since used this approach (successfully) on other projects.

The compiler is the PIC BASIC Pro (PBP). It is available from microEngineering Labs for about \$250.

The PIC sends a five-byte data packet to communicate the *x/y* touch coordinates to the host processor (see Figure 8). The first byte in the five-byte header always has its most significant bit set to one. The other four bytes always have their most significant bits set to zero to allow the receiver to synchronize to the packet. The first byte is either a \$C0 or \$80, depending on whether the touch is active (\$C0) or a touch-up event has occurred (\$80).

The four bytes that follow hold the *x* and *y* coordinate values, with 14 bits allowed. Because we digitize to 12-bit resolution, bits 12 and 13 are zero filled.

The firmware was written as a state machine, as shown in Figure 9. On powerup, the code enters State 0, where it initializes the data direction registers and blinks the LED three times. A transition to State 1 follows, where the code continuously scans the touchscreen, looking for a touch.

When a touch is detected, a transition to State 2 occurs, where the controller sends out a five-byte data packet, and turns on the LED. While in State 2, the controller continues to scan the touchscreen and sends out a new data packet each as long as a touch is detected. If a touch is not detected, a transition to State 3 occurs, where a final data packet is sent with the header byte set to \$80 indicating a touch-up event, the LED is turned off, and a transition back to State 1 is initiated.

A low-level subroutine called `Convert` is used to talk directly to

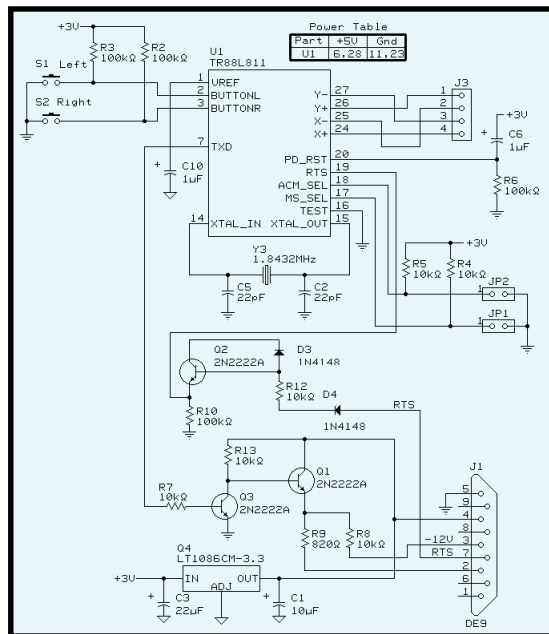


Figure 7—The processor in this interface is actually the Tritec TR88L811, a dedicated 4-wire touchscreen interface device. It handles all of the touchscreen scanning, touch detection, and message formatting chores. The 3-VOLT device was originally developed for the PDA industry.

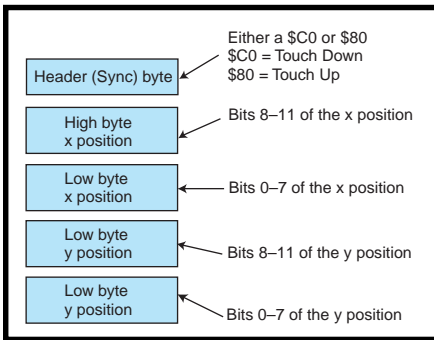


Figure 8—This is the message format used for circuits in Figures 3, 4, and 6. A simple five-byte data packet is used to transmit the touch coordinates from the touchscreen controller to the host system.

the touchscreen controller chip. The routine is similar for both 4- and 5-wire chips. Convert passes a variable called channel, which contains a 0 or 1. This variable controls whether we read the x or y channel of the device. The 12-bit result comes back in a 16-bit word named ADC.

Convert is called by a higher level subroutine named Read_Glass. This routine determines if a touch has occurred and sets a flag called touch to indicate such. A touch is determined to have occurred if the result of a Convert operation is above a certain noise threshold. When Convert is called and no touch has occurred, a value near zero is returned.

WRAPPING IT UP

Well, there you have it. You've seen four different means of interfacing a resistive touchscreen to your system. The Burr-Brown chips offer high accuracy, off-the-shelf solutions to both 4- and 5-wire glass types. My roll-your-own PIC interface is a

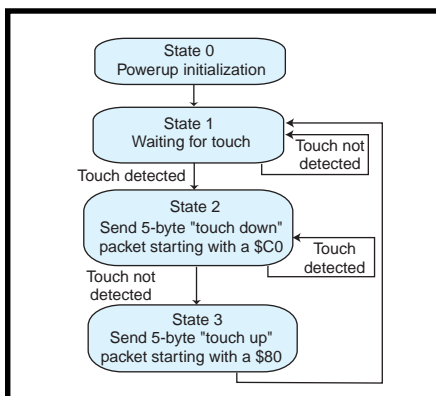


Figure 9—In this flow diagram for Figures 3, 4, and 6, the controller continuously scans the touchscreen, detects the touch, and sends out a five-byte data packet.

minimal parts count design and is best suited to low-accuracy applications. The TriTech chip offers a unique solution in that it involves no firmware. Those of you whose favorite programming language is solder will appreciate that. ☒

Tom Dahlin is a published author who runs a consulting business where he has developed the electronics for products such as a Talking Trash Compactor used in the fast food industry, an automated french fry dispenser, and (don't laugh) a pig sperm heater/controller for the animal-breeding industry. Tom has 20 years of design engineering

REFERENCES

D. Blass, *Touch Screens for Flat Panel Applications*, Sharp application note, January 18, 1996.

V. Kuroodi, *Pen Input Capability for Portable Devices*, TriTech Microelectronics application note, 1996.

SOURCES

TR88801/802 4-wire controllers
TriTech Microelectronics Int'l Inc.
(408) 894-1900
Fax: (408) 894-1919

ADS-7843/-7845
Burr-Brown Corp.
(520) 746-1111
Fax: (520) 889-1510
www.burr-brown.com

Touchscreen glass
Bergquist Touch Products
(800) 796-6824 • (612) 835-2322
Fax: (612) 835-4156
www.bergquistcompany.com

Microtouch Systems Inc.
(800) 642-7686 • (978) 659-9000
Fax: (978) 659-9105
www.microtouch.com

PIC16C71
Microchip Technology, Inc.
(888) 628-6247 • (480) 786-7200
Fax: (480) 899-9210
www.microchip.com

Basic Stamp
Parallax, Inc.
(916) 624-8333 • Fax: (916) 624-8003
www.parallaxinc.com

Neural Stamp

FEATURE ARTICLE



Robert Lacoste

A Low-Cost Neural Network Processor

Neural networks are no longer just an ideal research experiment for academia. In his Design99 award-winning entry, Robert clears up some of the mystery of implementing neural network technology in everyday applications.



Considered a research topic for years, neural networks are now a mature technology, with applications ranging from image recognition to data forecasting and from real-time process control to statistical analysis. But the widespread use of this technology is still quite limited, especially in embedded systems, probably owing to three main factors:

- embedded-system designers seem afraid of the academic style predominant in neural network publications
- ready-made hardware implementations of neural networks exist, but they are targeted to high-speed or high-complexity systems and are not adequate for low-cost applications [1]
- time-to-market is usually a predominant constraint, so neural-network technology is often considered a risky alternative because no dedicated embedded development tools are available

Why not develop a universal canned neural network module, reusable from design to design, using a simple microcontroller and a minimal number of external components to

keep the bill as low as possible? I'd had this idea for years, but Design99 was the trigger to start transforming the idea into a working prototype... and Neural Stamp was born (see Photo 1)!

I wanted Neural Stamp to be a purely analog macrochip. Its eight inputs are 0–5-V analog inputs; its eight outputs are 0–5-V analog outputs. With this approach, a Neural Stamp can easily be integrated into an existing analog design, replacing discrete analog regulation circuitry by neural network technology while minimizing system impact.

In this article, I first give a brief overview of the artificial neural network model. I then describe the Neural Stamp hardware and embedded software, as well as the associated PC-based development tool, NS-Drive.

NEURAL NETWORK MODEL

A lot of books describe artificial neural networks and how to use them [2, 3]. I'm not a specialist, so here is my very simplified, engineering-oriented personal summary.

A neural network is a set of computing elements (neurons), connected through weighted links. Each neuron calculates the weighted sum of its inputs (plus some bias), applies a non-linear function to the result (like the classical sigmoid function illustrated in Figure 1), and puts the resulting value on its output (see Figure 2).

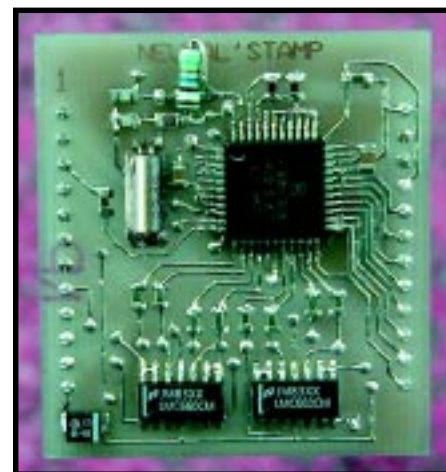


Photo 1—The Neural Stamp macro-chip is a SMT PCB with an onboard 68HC908GP20 microcontroller and circuitry for eight analog inputs and eight analog outputs.

Basically, a neural network is “only” a computing gizmo that takes n inputs, calculates a nonlinear mathematical function of these inputs, and puts the resulting values on the outputs.

The main difference between classical programming and neural network technology is that neural networks are not “programmed” but “trained,” using a large set of training samples (input values and desired output values). The goal is not to manually define the network weights but to let neural network training software do it for you.

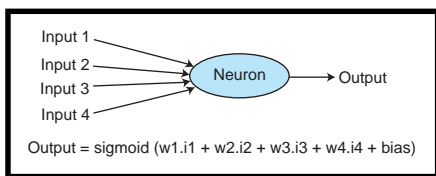


Figure 2—Each artificial neuron calculates a weighted sum of its inputs and passes this through a non-linear function.

Many training algorithms exist, from the classical back-propagation technique to complex multimode algorithms. And, the debate within the scientific community regarding how to size a neural network for a given task and which learning algorithm to use is extremely active!

For common problems, however, a feed-forward two-layer network (inputs, one hidden layer, and one output layer; see Figure 3) is usually a good compromise between performance and training convergence issues. The choice of a good set of training samples is far more important than the learning algorithm.

Moreover, a lot of good simulation and learning software is available (often for free on the web), so there’s no need to reinvent the wheel. Now you have the basics, so let’s take a look at the Neural Stamp concept.

NEURAL STAMP TOPOLOGY

To keep costs as low as possible (i.e., single-chip implementation), Neural Stamp has a fixed connectivity of eight inputs and eight outputs. Internally, the Neural Stamp is structured as a two-layer feed-forward neural network, as in Figure 3.

It has a hidden layer of 16 neurons, each connected to the eight inputs and to a constant bias source. It also has an output layer of eight neurons, each connected to the 16 hidden neurons and to a constant bias source.

Each link has an associated weight, so this network is characterized by 280 parameters ($16 \cdot 9 + 8 \cdot 17$). As a coarse estimation, this kind of network needs about 150–500 training samples to be correctly defined.

If the connectivity or computing power of the Neural Stamp is insufficient for a given application, several devices can be chained (with analog links) to build larger networks. For example, two Neural Stamps can be used to build a full-connected 8 inputs/16 hidden/16 outputs network.

WHY THE 68HC908GP20?

To decide on a microcontroller for the Neural Stamp macrochip, I listed the major constraints for this project:

- low-cost microcontroller
- eight analog inputs
- speed, especially quick arithmetic operations
- large amount of flash memory to store the weight associated to each neural connection, and enough RAM for neuron states storage
- in-circuit programming to download weight files
- small surface-mount package

And the 68HC908GP20 fit these requirements almost exactly! I won’t present the details of the chip here, because Tom Cantrell covered it well in “Flash Forward” (*Circuit Cellar* 104).

The main positive points for this application are that it’s cheap, and it’s even cheaper if you take into account that a simple 32-kHz crystal can be used, thanks to the onboard PLL. Also, its 8-MHz bus frequency, but more importantly its fast $8 \cdot 8$ multiply instruction, is interesting for this application because it permits a calculation cycle under 50 ms for the 24 neurons.

The 20-KB flash in-circuit programmable memory stores the application

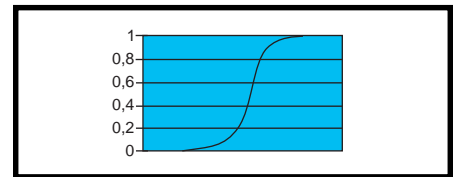


Figure 1—Any non-linear function may be used, but a sigmoid function, as shown here, is the most common.

software and up to 12 full sets of weights. And, the integrated low-voltage detection and watchdog allow a safe “macrochip” approach, helping to hide the microcontroller’s presence from the end user.

NEURAL STAMP DETAILS

Figure 5 shows the schematic of the Neural Stamp macrochip. The eight analog inputs AD0–AD7 connect directly to the Neural Stamp input pins.

The eight Neural Stamp analog outputs come from PD0–PD7 digital outputs through a 20-Hz simple low-pass filter and a quarter of an LMC660 used as an analog buffer [4]. This simple circuit is associated with an interrupt-based routine to implement a low-cost eight-channel sigma-delta DAC.

For those interested in bit-banging, the difference between a classical PWM-based design and a sigma-delta DAC lies in the shape of the digital

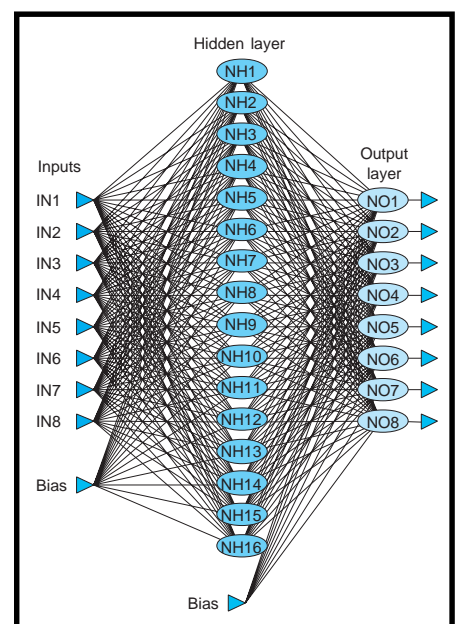


Figure 3—The Neural Stamp implements a fully connected neural network with its 8 inputs, 16 hidden neurons, and 8 outputs.

signals. With a sigma-delta converter, the ones and zeros are (on average) more spread out, giving far fewer low-frequency harmonics with the same sampling frequency, as Figure 4 illustrates.

And because the harmonics are at higher frequencies, a simple low-pass filter gives surprisingly good results, but at the expense of more computations on the microcontroller. For more details about sigma-delta algorithms, see David Tweed's article, "Digital Processing in the Analog World, III," (*Circuit Cellar* 101) and Listing 1.

The clock generator circuit is built around a standard 32.768-Hz clock crystal (upconverted to 32 MHz by the internal PLL, giving an 8-MHz bus frequency). Finally, the TXD and RXD SCI connections are routed to some Neural Stamp pins for a PC connection, as well as the reset line and the pins needed for the monitor-based in-circuit programming (PA0, IRQ, and OSC1).

The Neural Stamp components fit on a 39 × 44 mm PCB (1.52 × 1.82) with two 12-pin headers. All components are surface-mount, except the crystal and the headers (and, well, a 10-Mw resistor, because I couldn't find it in a surface-mount package in time).

The Neural Stamp prototype was in fact my first surface-mount home-built project. I'm pleased to confirm that the soldering of 0603 size resis-

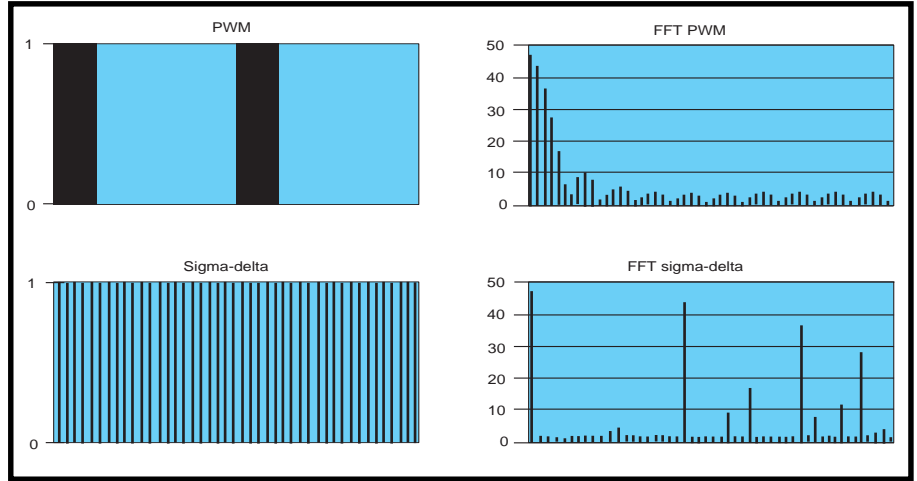


Figure 4—PWM and sigma-delta signals and frequency spectrums for an 18% output ratio with the same sampling frequency (23 ones and 105 zeros each 128 cycles). The sigma delta version has far fewer low frequency harmonics and is easier to filter.

tors and even QFP packages is possible without huge equipment—just a delicately handled fine iron, and lots of patience!

NEURAL STAMP TEST BOARD

To test the Neural Stamp macrochip, I wired a small test board to provide input signal (potentiometers) and output signal monitoring (LEDs). This board, shown in Photo 2, also includes all the circuitry needed for the in-circuit programming of the flash memory. Three switches on the test board let you switch the PC serial port between SCI and flash download modes (see Figure 6).

I should point out that this in-circuit programming is possible using the 'GP20 onboard monitor ROM, but

it needs an external 4.9152-MHz oscillator because the PLL is not enabled if the flash memory isn't empty [5]. This factor complicated the project because I wanted to have the full 8-MHz power for the main program.

Of course, it's possible to bypass the monitor ROM by including flash-memory management routines in the application software. But, I didn't implement this feature on the current version of the Neural Stamp firmware (deadlines).

Instead, I used an external 4.9152-MHz oscillator for downloading weight files and application software updates. A future firmware evolution will solve this issue, but I think the Motorola guys should have included a more flexible monitor ROM.

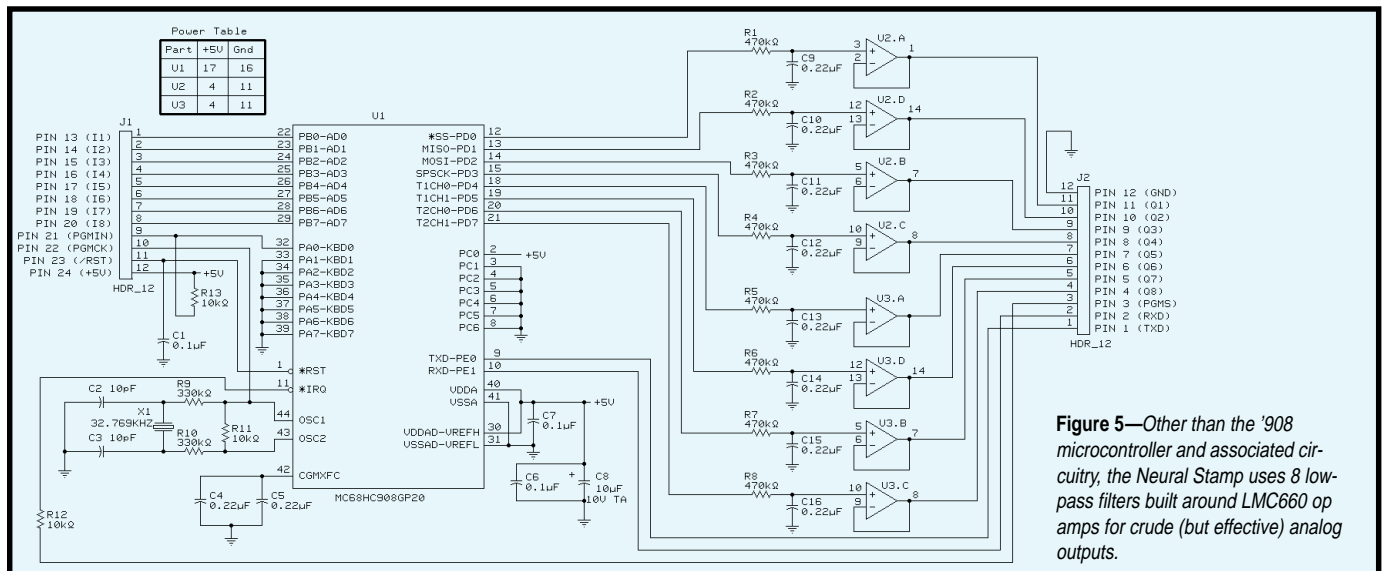


Figure 5—Other than the '908 microcontroller and associated circuitry, the Neural Stamp uses 8 low-pass filters built around LMC660 op amps for crude (but effective) analog outputs.

Listing 1—The sigma-delta algorithm needs 375 cycles to update the eight analog outputs.

```
Variables:  
S = sigma-delta accumulator (-128 to +127)  
Vo = desired output mean value (0 to 127)  
X(N)=previous output binary value (0/1)  
X(N+1)=next output binary value (0/1)
```

```
Algorithm:  
For each time step  
  For each output  
    S = S + Vo  
    if X(N)=1 then S=S-128  
    if S>0 then X(N+1)=1  
    else X(N+1)=0
```

NEURAL STAMP EMBEDDED SOFTWARE

The Neural Stamp embedded software, written in assembler, resides in the 'GP20 flash memory. As Figure 7 illustrates, it is classically structured into two tasks—a main program, which is in charge of the initializations and the main processing loop (analog acquisitions, neural network calculations and host serial link management); and an interrupt routine,

called at a 4-kHz rate, which is in charge of the sigma-delta output pulse generation.

These two tasks communicate through eight RAM variables that store the desired output values. The software uses about 1.5 KB of flash memory, starting at SE000.

The remaining part of the flash memory is divided into 12 memory blocks. Each block can store a complete set of neural network param-

eters (weights associated to each of the 280 connections). During initialization, the software looks for the last nonempty block and uses the values it contains for all neural network computations.

This arrangement allows multiplying by 12 the erase/write-cycle specification of the chip. Up to 12 sets of parameters can be downloaded and tested before the flash memory has to be erased!

The software itself currently uses exactly 67 bytes of RAM, using an additional 12 bytes for the stack. So, a majority of the initial 512 available bytes are still free, but they'll soon be used by the software extensions I have in mind.

MAIN PROGRAM

The main program (`nstamp09.asm`) is easy to read and generously commented. The trickier parts are the numerical computations. To save both memory and execution time, all calculations are done on the minimum argument size:

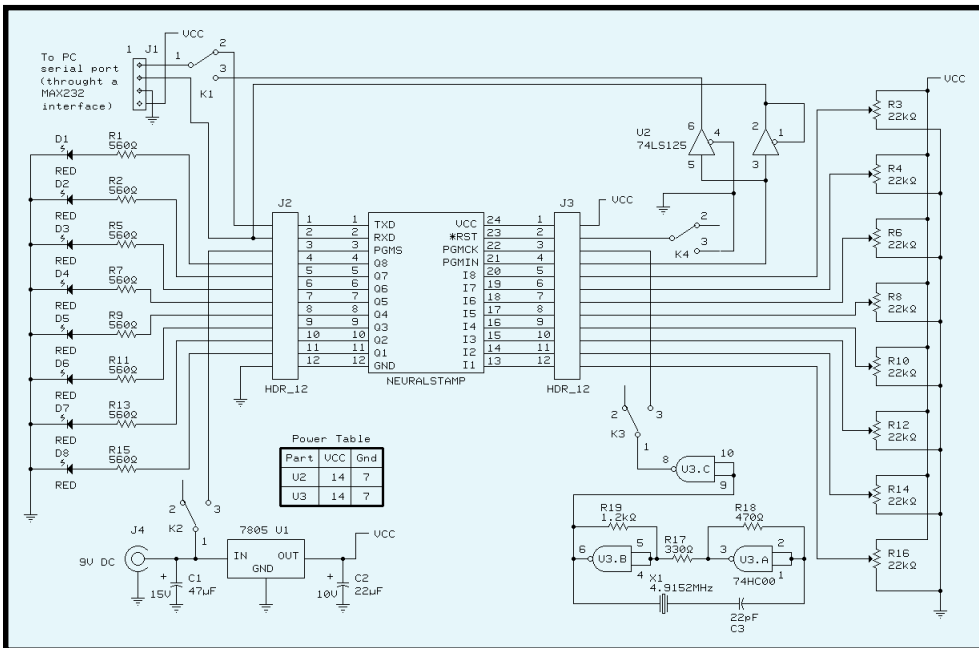


Figure 6—The test board uses potentiometers for input signals and LEDs to monitor the output levels. It also contains the necessary logic and clock for the monitor mode flash programming.

mands from the host, needs about 128,000 cycles for complete execution (acquisition and neural network calculation), or 32-ms CPU time. Taking the interrupt CPU time into account, this gives a 50-ms global cycle time, or a cycle frequency of 20 Hz. Not bad for a low-cost neural-network implementation!

PC-HOSTED DEVELOPMENT SOFTWARE

A system designer needs a good development and debugging environment, so I had to leave the soldering iron for the keyboard and develop the dedicated PC-based software, which I call NS Drive.

With this tool, a PC can be linked to the Neural Stamp

- the input and output of each neuron is stored in one unsigned byte (0–255)
- the weight of each connection is stored in one signed word (–32,768 to +32,767)
- all the intermediate calculations are done on three signed bytes

This appears to be a good compromise between rounding and truncating errors and complexity. A normalization factor is used to optimize the available numerical range.

A dedicated multiplication routine is included to get the maximum performance possible. For division, performance is less critical, so the generic $32 \div 32$ division routine provided in Motorola's application note is called directly [6]. The sigmoid estimation is done via a precalculated table.

Lastly, a simple serial protocol is implemented for host-to-target dialog during development steps. One-byte-long commands allow the host to identify the target, read input and output values, start and stop the neural network computations, or write specific values to the target's outputs in stop mode.

INTERRUPT ROUTINE

The interrupt routine is called at a

frequency of 4 kHz, thanks to the integrated time-base generator. Each time it is called, a new binary value is calculated for each of the eight analog outputs.

The desired output value is first divided by 2 (resolution 0–127) and then processed through a sigma-delta emulation. In worst-case conditions (output value 1 or 126), the output bit is updated at a frequency of:

$$\frac{4000}{128} = 31 \text{ Hz}$$

which is adequate, given the Neural Stamp 50-ms global cycle time.

The sigma-delta algorithm is given in Listing 1. This interrupt routine needs 375 cycles to update the eight analog outputs. With the 8-MHz bus frequency, this translates into a 93- μ s execution time. Because the interrupt frequency is 4 kHz, the interrupt routine uses:

$$4000 \times \frac{93 \mu\text{s}}{1 \text{ s}} = 37\%$$

of the CPU time, as shown in Figure 8.

The main program, while not receiving com-

through a serial link and the designer can get real-time training samples from the real-world application, train and simulate the neural network, and download a neural-network parameter set into flash memory. After this learning phase, the Neural Stamp is autonomous and no longer needs the PC connection.

NS Drive can also be used during design optimization or maintenance operations. The designer can reconnect the serial link and monitor the neural network behavior in real time, without any impact on the process control itself.

Because a lot of good neural network development software is already

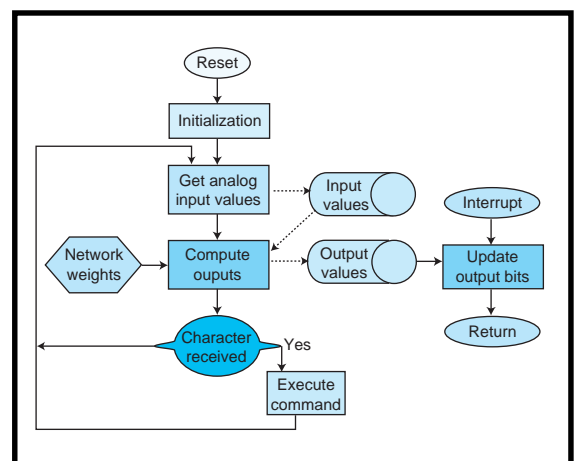


Figure 7—The embedded software is structured into a main program (managing the calculations and serial port) and an interrupt routine.

available (and enhanced every day by an active research community), I decided not to develop something from scratch. Rather, I use a general-purpose neural network software for training, optimization, and verification.

NS Drive serves as a gateway between the Neural Stamp device and this external software (through ASCII files) as well as integrating real-time monitoring functions. For the training software, I used the simple but effective Visible Neural Network shareware. However, virtually any other package can be used, with some postprocessing.

NS Drive supports five different modes, as shown in Figure 9. In the training-set acquisition mode, real-time input values are read from Neural Stamp inputs and stored in a file with the corresponding output values entered by the user. This training set can then be fed to the general-purpose neural network training software.

The simulation mode consists of offline verification of the weight file generated by the training software, with exactly the same algorithm as the one embedded on the Neural Stamp (integer calculations) or with a floating-point version of the same algorithm (useful to detect rounding issues).

Emulation mode is online verification of the weight file. Inputs are read from the target inputs, the neural network is simulated on the PC, and the outputs are written back to the target's outputs. This mode lets you quickly test a weight file on a real application before burning it into flash memory.

The download mode lets you generate a binary file that can be down-

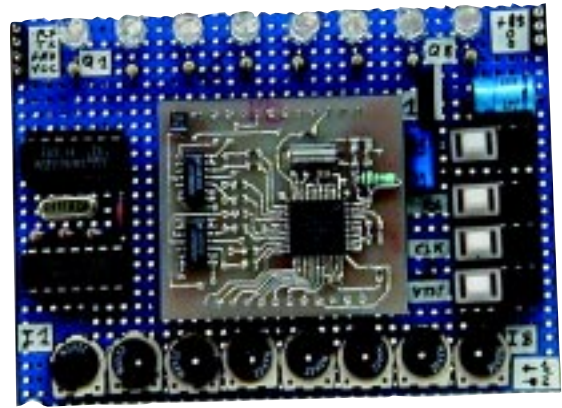


Photo 2—The switches on the right of this Neural Stamp test board are used to switch from flash download mode to execution mode.

loaded on the Neural Stamp flash memory through Motorola's supplied tools [5].

Finally, there's real-time monitoring of Neural Stamp inputs and outputs, with the neural network calculation still done locally on the Neural Stamp. This mode can be used without disturbing the real-time control process, which aids on-site problem solving.

In a nutshell, NS Drive fills the gap between the powerful academic neural network development tools and the needs of the embedded system engineer who's more used to in-circuit emulators and cross compilers than mathematics! It provides a set of sliders to monitor and/or control the inputs and outputs of the Neural Stamp (see Photo 3). It also simulates the neural network in emulation or simulation modes.

Technically, NS Drive is Win95-based software, developed with Bill's Visual C++ under the MFC framework. I designed it as a multithreaded software, with one thread dedicated to the user interface and a second thread managing the serial link.

NEURAL STAMP AT WORK

To test the Neural Stamp, and to demonstrate what can be done with it, I trained my Neural Stamp to recognize specific patterns on the analog inputs. As illustrated in Figure 10, each output is trained to be full on when a specific pattern is defined on the inputs.

For example, output 2 is trained to be full on (and all other outputs full

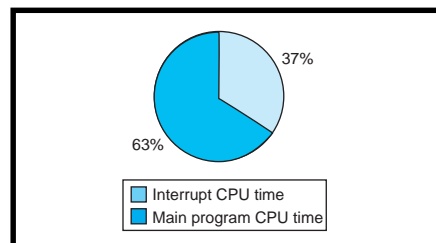


Figure 8—37% of CPU time sharing is dedicated to managing the sigma-delta emulation for the 8 outputs.

off) when input 1 is at 5 V, input 2 at 4.2 V, and so on, down to input 8 at 0 V.

To train the network, I used about 30 different training sets. In addition to the eight basic sets that are shown in Figure 10, I also entered several sets of pseudo-random input signals and what I, as a human, can see as a general trend created on these input values.

Using the Visual Neural Network toolset with the standard back-propagation algorithm, I got a good convergence of the learning and was able to download the resulting weight file to the Neural Stamp.

Photo 4 shows that an input pattern may not be exactly identical to the training set C4, but “similar” is correctly identified by the network. Frankly, the output C4 is “on.”

A major advantage of the neural network technology is that even if the

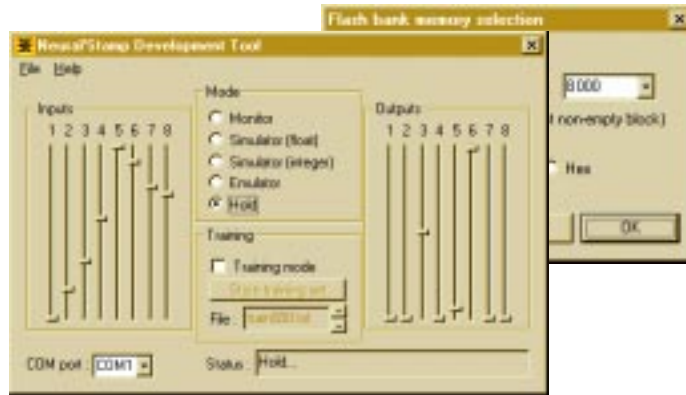


Photo 3—NS-Drive is windows-based software capable of real-time display of input and output values.

input pattern isn't like any of the training sets, the outputs are still reasonable:

- output 6 is doubtless “on” because the inputs 1–4 are increasing as in test case C6
- output 3 is moderately “on” because the general shape is similar to test case C3
- output 5 is just a little “on” even if nobody knows why!

As you see from the example I have given, neural networks can generalize or interpolate from the training cases. Because the number of training sets is usually small, sometimes the network's behavior isn't what we want (as with output 5 above). In a situation like this, you simply need to add training sets, making the network learn what you want it to do, and relaunch the learning algorithm!

DEVELOPMENT PROCESS

Developing an embedded application like Neural Stamp may be a nightmare without the appropriate tools and method. On the tool side, the 'GP20 development environment has two big advantages—it's freely available via the Internet, and it includes a powerful simulator that can work either offline or with I/O redirected to the target.

I like to develop an embedded application in an iterative, step-by-step approach. I try to find bugs as soon as possible, often before going on the target or even writing assembler code! Here are the different steps I used for this project.

I first developed the NS Drive PC-based software, with its floating-point mode neural-network emulation. This software was fully tested by comparing its outputs with the Visible Neural Network outputs. This helped me get a sense of how a neuron works. All the serial communication to the target was also emulated using a palmtop computer (a PSION 5) connected to the PC serial port and macros with the terminal emulator application [7].

Still on the PC, I then developed the NS Drive integer-mode neural-network emulation, coding it with the target architecture in mind. This step let me debug rounding and truncating problems, comparing the outputs in integer and floating-point modes.

Then I wrote the target software in assembler. First, the basic arithmetic functions (multiply, divide, sigmoid function, etc.) were written and debugged with the simulator, and then the rest of the application software.

I tested nearly 100% of the embedded software with the simulator. The same neural network simulation algorithm was coded both in NS Drive source and in the target code, so I could single-step through both sets of code in parallel on the same PC (NS Drive with Visual C++, target code with the 'GP20 simulator). This way, I could flag any differences in the intermediate results.

Finally, the real thing—download

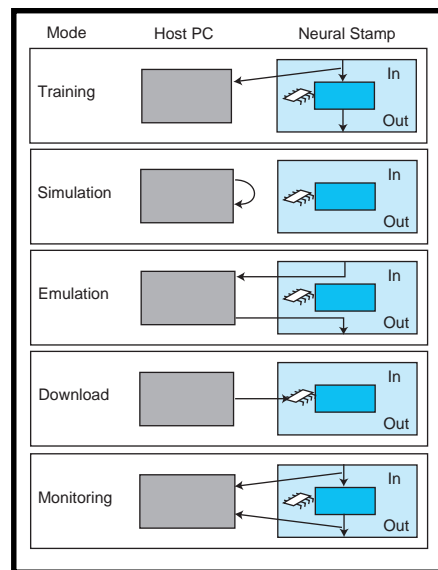


Figure 9—Support for five different operating modes allows the quick development and debugging of neural network-based applications

the software into the target and test it. Thanks to all the previous testing, this phase was quick. In fact, the only major remaining bug was related to the PLL initialization, which is quite difficult to simulate.

GOING FURTHER

Because a lot of parameters are predefined, Neural Stamp can be a quick and cost-effective solution for medium-complexity embedded applications. Thanks to the speed of the 'GP20, the Neural Stamp's quick response time (under 50 ms) should be adequate for most process control applications. Moreover, several Neural Stamp macrochips can be chained to build more complex networks.

Of course, a project like this one is never really finished. New ideas always crop up before the previous version is even finished!



Photo 4—The neural network recognizes an input pattern close to the test case in C4 and indicates this with an active level on output 4.

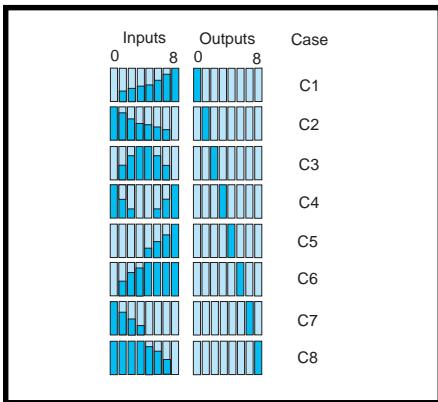


Figure 10—In this basic set of training cases, each output is trained to be fully on when presented with a specific input pattern.

The first improvement I'd make would be to integrate flash-memory writing algorithms in the application software. That way, the external 4.9152-MHz clock wouldn't be needed to download the weight files. Moreover, the software could automatically look for the next free block and erase the flash memory if all blocks are already used, hiding flash-memory management from the user.

On the PC side, a better integration of the NS Drive software with a good window-based neural network simulation package should also be planned because the two tools are quite distinct. This necessitates selecting a preferred tool from the extensive available software and writing some integration code.

And finally, why not include some learning algorithms directly on the target itself? This change could dramatically increase the number of applications of the Neural Stamp, allowing adaptive in-field training. However, I'm still a little leery of this approach. Learning algorithms sometimes give strange results, and are definitely memory hungry!

To be honest, another goal of mine is to find a partner to industrialize and commercialize the Neural Stamp concept, since this an after-hours project not linked at all with my current job. So, dear reader, if you're interested, don't hesitate to contact me! ☺

Robert Lacoste lives in France, near Paris. He has 10 years of experience in real-time software and embedded-

system developments but still loves building innovative microcontroller-based devices after hours. He is currently a senior project manager in the wireless industry. You may reach him at robert_lacoste@yahoo.fr.

SOFTWARE

The main program (Nstamp09L.asm) and executable files are available via the *Circuit Cellar* web site.

REFERENCES

- [1] C. S. Lindsey and T. Lindblad, Survey of Neural Network Hardware, Swedish Royal Institute of Technology, <http://msia02.msi.se/~lindsey/pub/spie.html>
- [2] Neural networks FAQ, <ftp://ftp.sas.com/pub/neural/FAQ.html>
- [3] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA, 1997.
- [4] National Semiconductor, LMC660 CMOS Quad Operational Amplifier, Datasheet DS008767, 1999.
- [5] G. Whitacre, In-circuit Programming of Flash Memory in the MC68HC908GP20, Motorola, Application note AN1770, 1998.
- [6] M. Johnson, M68HC08 Integer Math Routines, Motorola, Application note AN1219, 1996.
- [7] A. Denver, "Serial Communications in Win32," Microsoft Windows Developer Support, www.microsoft.com/win32dev/base/serial.htm.

SOURCES

68HC908GP20

Motorola
(512) 328-2268
Fax: (512) 891-4465
www.sps.mot.com/

Visible Neural Network shareware

The Visible Neural Network, Inc.
ftp://industry.net/pub/sharewar/ai_xprt/visnn20.exe

FEATURE ARTICLE

Michael Howard

Embedded CAN Can

If you are looking for a networking solution and RS-485 and Ethernet aren't going to cut it, you might consider CAN, using Microchip's MCP2510. Follow Michael as he shows us some of the benefits of this standalone controller.



A few weeks ago, I needed to get an EEPROM datasheet from Microchip's web site and happened to glance at the flashing features section. The one about the new standalone CAN controller caught my eye, so I checked it out.

The web page describes a chip that matched my needs so well that I immediately downloaded the 74-page datasheet and ran off to the coffee shop. So, here I am a few weeks later, after having some first-hand experience with the MCP2510 part. And I must say, if you have any plans to network embedded devices, check this one out. The MCP2510 is a good solution when RS-485 and Ethernet are not.

As a software architect at emWare, I spend many hours trying to match customers' requirements for device networking with communications technologies appropriate for their product. This activity continually reminds me that there is no single right answer when it comes to establishing remote connectivity. Sometimes this means that there are many reasonable ways to solve a connection problem, but it often seems there's no good way to make it happen at all.

The customers who desire to add or enhance device networking in their products have a notion of how valuable this connection is in their particular product. This value is measured against the cost of implementing network connectivity and it governs the decision whether to go ahead with the project. In other words, they're seeking a high level of capability for a low overall system cost increase.

It's common to have customers consider using RS-485 networks to achieve low-cost communication, especially if the controller in an existing design has an unused serial port. For some applications, a simple RS-485 network is appropriate.

Other implementations must consider the disadvantages. The CPU load can be pretty high for this type of network, even if 9-bit characters are used to flag node addresses.

As well, the maximum speed is often constrained to 115 kbps. This limitation is due to the load imposed on the CPU for address filtering and capabilities of the UARTs in the controllers on the network.

The lack of reliable collision detection generally leads to a master/slave media access strategy. This means that when a single point of failure exists on the network, the communications bandwidth may be lost due to bottlenecks in the network master.

One networking alternative is to use Ethernet. There are many low-cost, but reasonable, Ethernet controllers that include significant RAM on-chip and can be used by small controllers like a PIC16xxx or similar 8-bit device.

The Realtek 8019AS chip is a good example of an NE2000-compatible part with 16-KB SRAM and a modest price of less than \$5 in 1000-piece quantities. Using this type of part is easy on the software side because of widely available Linux drivers for NE2000 chips.

The code size can be significant, though, because you need to manage the memory in the device driver. emWare customers who adopt Ethernet are generally in need of a high-speed connection, a hook into an

existing network, and available I/O pins to dedicate to the interface.

Control area networks (CAN) offer an alternative that falls somewhere between these two in terms of system cost and data rates. Also, its features make it particularly well suited for distributed, embedded systems. These features include priority encoding on the physical network, which guarantees that the highest priority message gets through.

Priority, coupled with a small packet size, minimizes latency and makes that latency predictable. When there are collisions on an Ethernet network, there's no guarantee that the important messages will arrive in a timely fashion. Even though a 100-Mbps Ethernet network that's lightly loaded can deliver the data faster, the speed is often not as important as the determinism offered by CAN.

Another characteristic of CAN is that a node does not send messages to a specific target. It places the message on the bus, and any node that is interested in the message may use it.

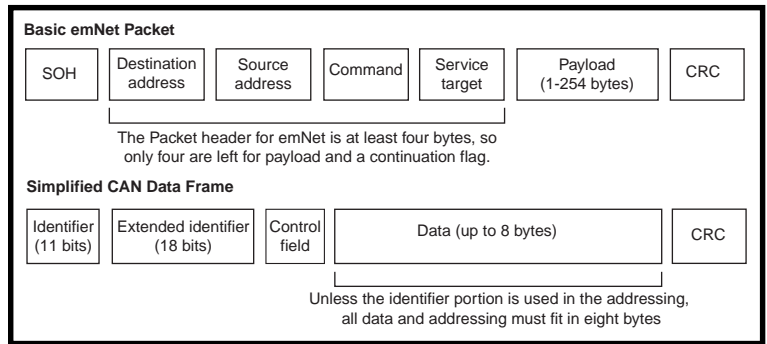
The reception of messages is also based on the identifier of the data, not the sender. Unless you build a layer on top of the message delivery offered by the controller, the nodes have no way of telling what other nodes are sending or receiving messages.

This feature, similar to using Ethernet in multicast mode, lets the systems engineer add new nodes that participate in the network without modifying code in the original nodes. A typical implementation has a node sending messages when it is certain of its variable's change state. Other network nodes can then receive these updates and react accordingly.

Consider a simple HVAC control, for example. The temperature sensor may continually send messages with the current temperature reading. The furnace and air conditioner can then watch for messages to alter the temperature.

The master control compares the desired

Figure 1—Note the packet size differences in this diagram of serial protocol packet information for basic emNet and Simplified CAN.



value input by a user-interface node and generates the temperature-change messages. If a new device is added to the network, it can monitor the temperature and commands to the other equipment and supply another method to input desired temperature.

A CAN node may request data by issuing a remote frame. This message is a request that some node places the specified message on the bus. If each node is the sole supplier of one specific message type, you can use the remote frame to determine whether a specific node is present.

So, as usual, the best choice for any one customer depends on their particular project needs. My various recommendations are shown in Table 1.

For example, I recommend RS-485 if a bit rate of 115 kbps or lower will work and if the latency inherent in a polled master/slave bus is not too high for appropriate reaction to events on the device. Also, the load imposed on all the nodes needed to accomplish address filtering can't be too high, and sporadic interrupts caused by incoming characters should not adversely affect the application.

On the other hand, Ethernet is a good choice if the required bit rate exceeds 1 Mbps. Another factor is

whether the timing jitter in packet arrival will disrupt the control algorithm. You also need to have enough I/O and program space to connect the controller to the embedded system. The controller and associated support circuitry shouldn't be too expensive.

The MCP2510 CAN controller meets your needs when a bit rate of 1 Mbps will work and the network latency must be predictable, especially for critical messages. This kind of network requires only a small number of I/O pins and small amount of program space, and it's less expensive than the Ethernet solution. It's also a good choice if a peer-to-peer or multi-master system is required.

THE MCP2510

The MCP2510 is a CAN controller that comes in an 18-pin PDIP/SOIC or a 20-pin TSSOP package. The interface to the controller is accomplished through an SPI port, so adding CAN with this part requires as little as four control lines from the CPU. Additional pins can signal interrupts or be used as generic I/O lines (replacing the ones used by the SPI).

This part is a controller that supports Full CAN 2.0A, and 2.0B. The term "Full CAN" means that it's

better than BASIC CAN because the part performs address filtering in hardware.

The address filters screen the messages placing them into one of the two receive buffers. The BASIC CAN parts require the CPU to filter all the messages to see if that CPU is interested, similar to address filtering in a RS-485 network.

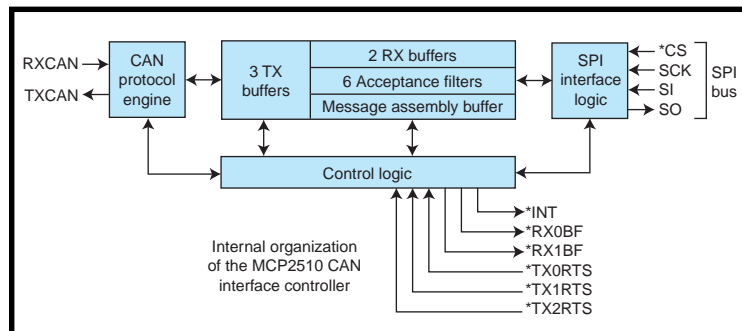


Figure 2—Internal organization of the MCP2510 CAN interface controller. All access to CAN controller functions can be accomplished through the SPI bus, but additional handshaking is available through discrete status and control lines.

The 2.0A and 2.0B compatibility refers to the size of the message identifier. The 2.0A-compatible CAN controllers use an 11-bit identifier. The extended message format of CAN 2.0B allows a 29-bit message identifier.

The SPI port accessing the controller uses six types of instructions: RESET, READ, WRITE, RTS (Request to Send), READ STATUS, and BIT MODIFY. The READ and WRITE instructions internally increment an address counter, so it's easy and efficient to set up and transfer messages to and from the part.

One of my goals in using a part like this is to try to use as little program space in the MCU as possible. Because our customers often employ a fairly large EEPROM, I like to have as much of the communications initialization as possible placed into a data table. This enables me to loop through the table at startup and efficiently initialize a complex part.

The MCP2510's WRITE instruction increments to the next address, so I just have to move a large group of configuration data from the EEPROM onto the SPI bus. The READ STATUS command returns, in one byte, the status of the two receive buffers and the three transmit buffers. The same buffer-status information can be output to discrete pins on the controller and read via a port pin on the MCU.

A typical application of this capability is to set up one of the receive

buffer's filters to accept only high-priority messages. The buffer-full output can then be signaled by a pin on the MCP2510 and routed to an external interrupt line of the MCU.

Using this, the MCU can poll the SPI port for lower priority messages but be alerted to the presence of a high-priority message at any time. The transmit buffers can take similar roles.

The controller can be set up by the MCU at startup with a particular emergency message and the REQUEST TO SEND for that buffer can be tied to an input pin. An emergency condition can be attached in hardware to the pin, so the message is quickly dispatched. Because the high-priority messages will take precedence and the longest message will be 136 bits, the latency is extremely low.

The BIT MODIFY instruction on the SPI bus permits the MCU to modify specific bits in many of the MCP2510's 128 registers. This command eliminates the need to read, modify, and then write over the bus when the MCU needs to simply set one bit.

DEVELOPMENT SYSTEM

The development system for this part includes two of the CAN controllers on a local CAN bus that also has a connector for tying into other nodes. One of the controllers is wired to a parallel port connector linked to a PC.

Included software provides functions to manipulate the registers, view CAN traffic, and send messages over the network.

The second controller is wired to a set of three sockets that are wired for the various types of PIC microcontrollers. You can remove the PIC and wire a different controller to the second MCP2510 via SPI.

When I first started to work with the kit, I hooked a Phytec C164 development board to the external CAN port. I created an emWare interface to the C164 board, which has a set of variables that correspond to the message buffers and a few status variables.

I used this second board as a tool to see if the Siemens controller would talk to my Microchip controller, because it seemed likely that I'd miss problems if I used the same two types of chips for development.

Using the software provided by Microchip, I managed to interactively initialize the controller hooked to the parallel port so I could send and receive messages to the C164 board. I guess I was just a little paranoid about the compatibility. I was glad to have this interactive session because the 128 registers in the MCP2510 are a little daunting at first.

There are three configuration registers that control the bit timing for CAN transactions that especially had me worried, but they proved to be no problem. Having taken these steps, I felt ready to create a driver for a PIC controller hooked directly to the MCP2510.

CAN JUMP HURDLES

After becoming familiar with the part, my next goal was to link emWare SDK boards to a network server using this part. The SDK boards in question come in versions that use controllers from Analog Devices, Hitachi, Microchip, Motorola, and Philips.

I replaced the RS-485 network drivers on these controllers with an SPI link to the MCP2510. The result is a faster network that lets the controllers asynchronously signal the network device server when events occur and variables change.

RS-485

bit rate of 115 kbps or lower
latency inherent in a polled master/slave bus is not too high for appropriate reaction to events
load imposed on all nodes needed to accomplish address filtering is not too high
sporadic interrupts caused by incoming characters do not adversely affect the application

Ethernet

bit rate exceeding 1 Mbps is required
timing jitter in packet arrival will not disrupt the control algorithm
enough I/O and program space exists to connect the controller to the embedded system controller and associated support circuitry is not too expensive

MCP2510 CAN controller

bit rate of 1 Mbps
network latency must be predictable, especially for critical messages
a small number of I/O pins and small amount of program space are available for network use
Ethernet solution is too expensive
peer-to-peer or multimaster system is required

Table 1—When considering factors such as data rate, cost, and I/O pin requirements, the gap filled by MCP2510 becomes apparent.

Of course, I really didn't want to change a lot of higher-level code in the devices. The ideal integration would simply be to send our packets as messages to the devices. My serial protocol looked something like the series shown in Figure 1.

Because the CAN controller handles the header/trailer and CRC portions, I needed to handle the address, command, and payload portions. I first addressed the fact that the minimum size of our packets exceeds the maximum size of a CAN packet.

Our product encapsulates the capabilities of an embedded application into a table of function, variables, and events, and creates a network object representation at our server. Because we do not dictate the variable types, or the function input and output types, I can't use a single CAN message for a transaction between the Internet client and the device.

This problem isn't new, as shown by the many choices of application layers available to CAN users. Some examples are CAN Application Layer (CAL), CANOpen, and DeviceNet.

If I was trying to implement a network from the ground up, I'd design the whole set of controllers to work with sections of distributed shared memory. This way, the 8-byte limit would never be a problem.

If I wanted the easiest path to a CAN network hooked up to our emGateway server, I would pillage the 29-bit extended message identifiers and encode the command, ID, and offset into the identifier.

Our customers are generally interested in adding Internet connectivity to existing networks. At the least, they want the final result to be compatible with other systems.

So, even though the hacker in me is eager just to get it working, I am off on a tour through the available standards to find out how to fit emNet, a simple protocol that provides remote device access, inside one of these application layers.

The MCP2510, diagrammed in Figure 2, is a capable, low-cost communications controller that provides a simple way for any microcontroller lacking integrated CAN support to

participate in a CAN system. A CAN system meets the needs of embedded developers who need a faster link than a RS-485 connection and those who need to build a reliable distributed real-time control system.

Using SPI to access this controller enables it to be used in situations where a full microprocessor bus is not available. The hardware send and receive control lines on the device provide an additional mechanism to insure system reliability. I believe these features will result in great market success for this component. 📄

Mike Howard serves as chief architect for emWare, Inc. His responsibilities include managing development of the company's Embedded Micro Internetworking Technology (EMIT) device-networking software. Mike has 16 years of experience in working with distributed embedded systems in the areas of robotics, ATE, and CAM. He can be reached at mhoward@emware.com

REFERENCES

W. Lawrenz, *CAN System Engineering: From Theory to Practical Applications*, Springer-Verlag, New York, NY, 1997.

Microchip Technology, MCP2510 datasheet, 1999

Siemens, C164 datasheet, 2/98

SOURCES

MCP2510

Microchip Technology, Inc.
(888) 628-6247 • (480) 786-7200
Fax: (480) 899-9210
www.microchip.com

8019AS

RealTek Innovation Systems
(510) 351-5411
www.realtek.com

Siemens C164 development board

Phytec America LLC
(800) 278-9913 • (206) 780-9047
Fax: (206) 780-9135
www.phytec.com

SDK boards, emGateway, emNet

emWare, Inc.
(877) 4-emWare • (801) 453-9300
Fax: (801) 453-0150
www.emware.com

Mike Baptiste

Tuning up the HCS-II

A problem with most home automation systems is that they often take years to develop, so some parts may be out of date. Mike looks at the components of an HA system to make sure that old and new work together peacefully.



Home automation is a great hobby (or in some cases an obsession). You get to play with lots of neat stuff and (usually) make your home a better place to live. Most systems are the result of months and often years of work.

It's not that home automation is that difficult. However, home automation is often a generic term covering just about anything that performs an automated function in your home. Serious home automation systems handle things such as lighting control, infrared control and distribution, audio/video distribution, home security, hard-wired control, heating and air conditioning control, and sometimes even speech synthesis and recognition. These are often combined into a system that controls most aspects of a home.

The problem is, most systems are developed over long periods of time. Very few people sit down and plan their home-automation system from start to finish. Functionality is often added as money and free time allow. The result can be a system prone to strange behavior as new additions intermittently conflict with old parts.

Although I'm a firm believer in "if it ain't broke, don't fix it," there are changes you can make to improve the

stability and robustness of your home-automation system. Many of my suggestions are tailored for the HCS-II, but others are more generic and can be used in just about any HA setup.

POWER, THE GOOD AND THE BAD

The first step in any home-automation system is ensuring that it has a clean and stable power supply. Most HCS-II systems require a single 12-VDC power source that can supply 1–4 A. A cheap wall wart power supply will get the system running, but I don't recommend it because they usually don't have regulated DC outputs so their DC voltage varies depending on the load and the line voltage. To ensure that they maintain their rated output voltage at full load, they often put out higher voltages at low loads. I've seen 12-VDC supplies that put out 16–18 V at no load. Even at 25–50% of their rated load, some will still pump out 15 VDC or so.

"But, my HCS-II boards use on-board regulators, so it shouldn't matter," you say. True, however, linear regulators generate much more heat when supplied with higher voltages. If you have a couple add-on boards stacked on your HCS-II controller board, your regulator will be hot to the touch when fed with 12 VDC. Bump that up a few volts and the 7805 voltage regulator heat sink will probably burn your finger. Although the 7805 protects itself with a thermal shutdown, any EE will tell you that hot parts fail faster.

Another problem with wall warts is that they are little more than transformers with a diode bridge and maybe a filter capacitor. Noise and transients on the power line usually come out the other side. The 7805 does a great job holding the voltage steady, but boards like the PL-Link are sensitive to power-supply noise.

For most systems, a linear-regulated 12-VDC supply that can handle 2 A will do fine. However, if you have a large system, you might consider powering the SC and boards located near it with 5 V and using the 12-V supply to feed remote network boards.

Consult your HCS-II board manuals about the placement of a jumper to

bypass the 7805 so the boards can be powered with 5 VDC directly. Be careful! After you bypass the 7805 on a board, applying anything other than 5 V will damage the board. Switching supplies can also be used (I use one), although some folks report problems resulting from switching noise.

When you run power to your HCS-II controller, make sure you use a heavy-gauge wire (18 AWG), 22 or 24 AWG wire is too small where currents can reach an amp or so because of the expansion boards. Most remote network boards don't draw that much power, so running 12 VDC on another twisted pair is generally no problem.

If you have a large system, a good thing to monitor is its actual current draw. Although this seems odd because the current draw shouldn't change much, monitoring the current draw helps you spot potential trouble in your system. Any unexplained change in current needs to be investigated. Flip through most electronics catalogs these days and you'll see 3.5-digit LCD displays with voltmeter circuitry for around \$15. All Electronics, Mouser, Digi-Key, and Jameco can provide these LCD voltmeters.

On my system, I simply use a 0.01-W power resistor and use the LCD voltmeter to monitor the small voltage drop, which is proportional to the current passing through it. Because $V = IR$ and $R = 0.01 \text{ } \Omega$, I can display the current by placing the decimal point in the right place on the LCD display.



Photo 1—Here's an LCD voltmeter monitoring the current draw of my remote nodes (and HCS-II boards on my test bench—thus the high reading). The boards in the cabinet are powered by a separate 5-V supply. The power resistor is on the lower left. A 3-W power resistor is more than adequate at 0.01 W.

These LCD voltmeter displays enable you to place the decimal point anywhere. Photo 1 shows my setup. You'll notice a 9-V battery powering the display. Most of these LCD voltmeter displays cannot monitor their own power source. However, new versions that enable you to power the current monitor from your main power supply have recently shown up on the market. Because my 9 V dies every month or two, I plan to upgrade to the newer type LCD voltmeter.

If you have an X-10 board in your HCS-II system (PL-Link or HCS-PLIX), your HCS-II can monitor for power failures. The X-10 boards monitor the 60-Hz signal sent from the TW-523 and if that signal disappears, the boards signal a power failure.

Many people like their HA system to take various actions when the power fails (e.g., turn on battery-powered emergency lights) and also when the power comes back on. However your HCS-II needs a backup power source to do this.

A computer UPS can power an HCS-II for quite a while. After hurricane Fran, we lost power for a week, but my HCS-II stayed up for two days with a 400-VA UPS. The price of a low-power (280–500 VA) UPS is now between \$100 and \$200. Adding one of these to your system can ensure that it runs during a power outage and will usually provide surge protection to boot. For better protection, buy a UPS that also has modem protection and plug your HCS-DTMF or Caller ID modem into it.

Some systems use 12-VDC batteries to power the HCS-II. They provide clean and stable power. One HCS user has even designed an HCS-II UPS using lead acid batteries. I highly recommend this project, so check the reference section for a link to the web site. If a UPS isn't for you, I recommend plugging your system into a good surge suppressor.

TAMING THE RS-485 NETWORK

One of the most powerful features of the HCS-II system is the ability to locate devices far away from the controller. However, the RS-485 network can be a source of much frustration to

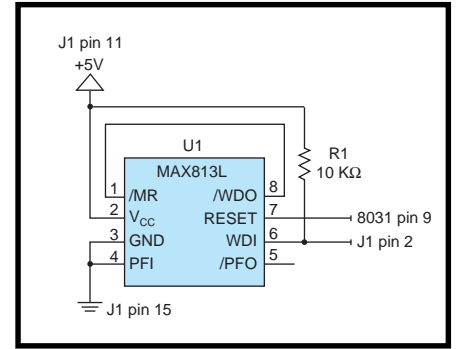


Figure 1—To modify your PL-Link, clip off C1, R2, and D1. Install the MAX813L and 10-k Ω resistor in the prototype area and connect it as shown.

some users. Before you go any further, I suggest reading “The Art and Science of RS-485” by Bob Perrin (*Circuit Cellar Online*, July 1999). Bob's article provides a great overview of RS-485 and the many pitfalls of using an RS-485 network.

One of the more common mistakes in HCS-II networks is not running a common ground wire along with the RS-485 pair. Not only does this cause data-integrity issues, it can also make your network more susceptible to damage from voltage transients. Because most people run CAT3 or CAT5 wire for their networks, there are usually extra wires you can use for carrying the ground.

A related error deals with any unused wires in your cable. Instead of grounding them (as most people do), Bob recommends using termination resistors at each end of an unused wire. An easy way to implement this idea is with bussed resistor networks. You can buy them in SIP or DIP form. Simply tie the common pin to ground and connect the other side of each resistor to an unused conductor.

Anyone who designs RS-485 networks will tell you the only topology to use is a daisy chain. Star topologies are to be avoided. However, most HCS-II users (myself included) use them, and the rest are probably fibbing. At short distances and low data speeds, signal reflections due to improper topologies are seldom a problem.

Nevertheless, if you have network nodes that sometimes fall off the network and you are using a star (or stars), get ready for some fun. First, try moving your terminators. Nor-

mally, the extreme ends of an RS-485 network should be terminated. However, star topologies make this difficult and figuring out the right place to terminate is an art, or just plain luck. Remember, you should only have two terminators on your RS-485 network.

If playing with the terminators doesn't help, try a slew-rate limited RS-485 driver. These chips replace your 75176 and reduce reflections caused by improper topologies and termination. The tradeoff is a lower top-end network speed, but the HCS-II runs at 9600 bps so this isn't an issue. Maxim's MAX483 works well, but before rushing out to buy one, read on.

The number one repair I perform on HCS-II controller boards is blown RS-232 and RS-485 transceiver ICs. Stringing hundreds of feet of wire in your house is asking for trouble. Any EE knows that the longer the wire, the bigger the induced transients should lightning strike nearby. Bob's article covers this subject in great detail and he concludes with a recommendation for Maxim's new ESD protected transceiver ICs. He couldn't zap one with 16 kV after 50 tries.

The Maxim MAX483E (the E is for "ESD") is a drop-in replacement for the 75176 with +15 kV of ESD protection and slew-rate limiting in one package. For almost a year now I've used these ICs in every HCS-II network node that I have without any failures. Before, I'd replace three to five 75176s each year, thanks to severe thunderstorms. North Carolina is a thunderstorm magnet. This summer, a lightning bolt hit a tree near the house. Not a single IC failed and the worst casualty was one module that hung and had to be power cycled.

ESD protection doesn't stop with the RS-485 network. Most HCS users have long cables running between their PC and HCS-II SC (mine is over 30'). To protect the HCS-II, I use a Maxim MAX232E and haven't had an RS-232 failure since.

GREMLINS AND X-10

X-10 is probably the most widely used home-automation technology and refers to both a company and the power-line transmission protocol they



Photo 2—An X-10 signal bridge looks like a dimmer with no knob. You will need an extra deep, single-gang electrical box to install it in due to its size. Use 14/3 cable wired to a 15-A 220-V circuit breaker.

developed to control devices over existing electrical wiring. X-10 has been around for more than 20 years and continues to dominate the power line carrier market with low prices and a wide range of products.

Some X-10 installations work from day one and never have any problems. Others can work great for months and then suddenly start turning things on at random. Most of this is due to X-10 using the power line to transmit data.

Electrical power lines are not ideal for transmitting data of any kind. X-10 controls remote devices by sending control codes in short 120-kHz bursts synched to the zero-cross of the AC power signal. For more technical details on how X-10 works, check out the X-10 transmission theory paper on X-10's web site.

However, power lines are notoriously noisy and this noise can cause all sorts of strange behavior in X-10 devices. After a while, you start to wonder about the existence of gremlins. Nevertheless, electrical noise isn't the only issue.

Before I dive into improving X-10 installations, a word about the HCS-II PL-Link. The PL-Link is the first-generation X-10 gateway for the HCS-II. Thousands of them are in service and most work flawlessly. However, a number of users (myself included) have had problems with them locking up in transmit mode. Once they lock up, X-10 commands wouldn't function. Thanks to the efforts of a number of HCS-II users who posted their findings on the HCS-II newsgroup, we

discovered that the PL-Link's were spontaneously resetting as a result of power-supply noise or transients.

I used a latching logic probe to monitor my PL-Link's reset line. Every time my PL-Link locked up, the latch light was flashing, meaning a reset pulse was generated for some reason. The reset pulse was not a clean pulse (it never reached 5 V) so the 8031 CPU would hang up after the substandard reset pulse came in from the simple RC reset circuit.

The exact cause was never found, but it seemed to relate to noisy power supplies or voltage fluctuations. Battery-powered HCS-II systems rarely had these problems. Systems with switching power supplies saw it the most. Other COMM-Link boards like the MCIR-Link never had these problems so it was specific to X-10.

To fix the lock-up problems, I replaced the PL-Link reset circuit with a Maxim 813L watchdog chip. See Figure 1 for the simple circuit. This chip performs two functions. It monitors the supply voltage and issues a clean reset to the CPU if the voltage drops below 4.65 V. The MAX813L also has a watchdog feature, which can issue a reset pulse if the PL-Link stops running. The PL-Link has a heartbeat output that toggles every second. If this heartbeat stops, the MAX813L resets the PL-Link.

CROSSING THE X-10 BRIDGE

Before I go on, here's a note about safety. Many of the X-10 improvements I'm about to discuss require you to connect devices to your electrical power lines via your home's circuit-breaker panel. If you decide to perform any of these improvements, please use extreme caution and turn off the main breaker. If you have any doubts or are unfamiliar with residential electrical wiring, have these devices installed by a professional electrician. Don't take any chances!

The most common X-10 problem in home automation is X-10 modules that won't turn on, or seem to only respond at random times. Thankfully, there is an easy fix to this problem.

Homes in the United States are fed with 240 VAC. The 240-V feed is split

at the circuit breaker panel into two 120-V legs. X-10 signals from an HCS-II are injected into your power line using a TW-523 interface. This interface injects signals on the hot side of the 120-V outlet it is plugged into. Thus, the X-10 signals are only injected into one 120-V leg. However, if the module you are trying to control is connected to the other 120-V leg, the X-10 signals must travel all the way out to the power transformer and then all the way back to your house. If the power transformer is far from your home, this can cause modules on the other hot leg to not respond.

You can easily see if a bridge will help. Turning on your dryer or range will “bridge” the two 120-V legs via the heating element. If your X-10 modules start working, you need to provide a path between the legs for the X-10 signals.

Leviton and other X-10 manufacturers make a device called an X-10 signal bridge. This aptly-named device creates a bridge for X-10 signals to cross between the legs in your circuit panel without shorting them together. This arrangement can significantly reduce the distance X-10 signals have to travel between legs.

Photo 2 shows a Leviton signal bridge. It can be mounted in a normal electrical box right next to your circuit breaker box. X-10 signal bridges must be wired into their own circuit breakers to meet code. Follow the instructions carefully.

X-10 signal bridges solve many problems related to X-10 modules not responding. However, just as long distances can affect X-10 signals between the feeder legs, it can also be a factor if you live in a large home.

If your home is 3000 sq. ft. or more and you have many electrical circuits, you may need an X-10 amplifier. X-10 amplifiers like the LV6201 from Leviton take received X-10 signals and amplify them to 5 V or so. Amplifiers also act as a signal bridge so you don't need a bridge and an amplifier. However, amplifiers are not a magic bullet. Before you install one, make sure you need one.

Professional X-10 installers use X-10 signal meters to measure the ampli-



Photo 3—Leviton makes a wide variety of X-10 noise blocks and filters to improve reliability. Take care when installing filters into circuit panels and electrical fixtures/appliances. When in doubt—hire a professional electrician.

tude of X-10 signals throughout a home. If they notice that the amplitude is too low (below 100 mV), an amplifier is usually necessary.

Until recently, X-10 signal meters were too expensive for the homeowner, with the top models costing hundreds of dollars. However, a new device called the ESM1 is available for less than \$70 and indicates signal strength and X-10 integrity. It's worth the cost if it prevents you wasting \$200 on an X-10 amplifier you don't need.

If you have an X-10 signal meter, program your HCS-II to send X-10 commands on a regular basis. Plug your meter into the various circuits in your home with X-10 modules that act strangely. Most X-10 modules will not work with signals less than 100 mV although anything under 1 V is weak. X-10 transmitters inject signals with amplitudes between 3 and 5 V.

If your X-10 signals are weak and you do install an amplifier, read the instructions carefully. The circuit breakers you connect an amplifier to must be turned on in a specific order.

NOISE YOU CAN'T HEAR

Electrical noise can wreak havoc on X-10 devices. All sorts of household devices can generate noise including microwaves, fluorescent lights, and televisions. This noise can sometimes interfere with an X-10 signal so the receiver doesn't respond or worse, it can fool an X-10 module into turning on or off when it isn't supposed to.

Leviton makes a number of products designed to reduce or eliminate noise from appliances and other electrical devices. Photo 3 shows a few of

these devices. As you can see, some can simply be plugged in and others are directly wired into things like fluorescent light fixtures.

To see if you have noisy appliances causing problems, try to note what is turned on when your X-10 modules start acting funny. Fluorescent, mercury vapor, and sodium vapor lights with internal ballast can create noise when turned off. If you have an X-10 signal meter, use it to see where the noise is the worst. Track down which electrical circuits the suspect X-10 modules are on and see what other devices are on the same circuit. Once you notice a pattern of problems when specific devices are on, try turning them off one by one to see if things improve. In the case of ballast fixtures, see if X-10 devices turn on or off when you turn the light fixtures off. Note there may be multiple devices causing problems.

For devices you don't need to control with X-10, the best solution is an X-10 noise block. For appliances like televisions, the Leviton LV6288 enables you to simply plug the noisy device into it and it will block all noise (and X-10 signals) in between the device and the power line. If you have a ballast fixture that's noisy, use a LV6287 noise block, which you wire inside the light fixture.

Leviton's LV6289 can filter noise without impacting X-10 signals. To prevent noise from crossing between circuits and from coming in from neighbor's homes, use two LV6289s in your main circuit breaker panel. Connect one between neutral and each incoming hot leg via a 220V 15A breaker.

DON'T PULL YOUR HAIR OUT, YET

If you've tried everything else and you still have X-10 modules with a mind of their own, the X-10 modules may be working fine. No, I'm not crazy. Remember when I mentioned how X-10 signals can travel between legs without a signal bridge? If those X-10 signals go all the way to the power transformer, they will also go to any other home that is nearby and connected to your transformer. Therefore, your X-10 modules may be responding to neighbors who use X-10 and the same house codes you do. Use the HCS-II HOST display to monitor various house codes to see if they are changing states on their own.

If a module activates at a given time each day and your system isn't on, you can bet a neighbor has caught the home automation bug. If you only have a few X-10 modules, try a different house code. If you have a lot of modules and can't seem to find enough available codes, you need to install an X-10 whole-house signal block in your circuit panel.

The LV6284 block is installed over the neutral feeder coming from your electric meter. This active filter will cancel out any signals that enter your home via your power lines. It also serves as a signal bridge so signals can cross between legs since they can't travel to the power transformer. This device must be installed by a qualified electrician and will often require that your electricity be shut off (unless you have external disconnect breakers).

SURGE SUPPRESSION TROUBLE

People with large home automation systems often have hundreds of dollars of X-10 equipment installed in their homes. To protect these devices, some folks plug X-10 transmitters and receivers into surge suppressors. Believe it or not, surge suppressors can actually interfere with X-10 signals. Actually, it is the RFI/Noise Filter circuits in surge suppressors that cause the trouble.

I had my TW-523 plugged into a single outlet surge suppressor because I figured it couldn't hurt to protect it. Recently, I was having trouble with a couple of lights not turning on when

they should have. When I used my manual transmitters, the lights came on without any problems.

I had read about the possible impact of surge suppressors so I removed it from the TW-523. Voila! The troublesome modules started responding to X-10 commands again.

I'm not saying surge suppression is a bad idea for X-10 devices. Just don't plug them into surge suppressing power strips or modules, which can interfere with the signals. The best way to protect your X-10 investment (and the other gadgets in your home) is to install a whole-house surge suppressor. Most circuit panel manufacturers sell surge suppressors that snap into the panel like a 220-V breaker. Note that these devices generally have mid-level surge suppression capabilities (~400 Joules).

If you live in an area with frequent or severe thunderstorms, I recommend the Leviton LV51120 surge suppressor. It handles 950 Joules and a maximum 50,000 A. If you use a whole-house surge suppressor, you should still keep your surge-protected power strips for non-X-10 equipment. A whole-house surge suppressor will absorb a lot, but some of the surge will still travel into your home and the localized suppressors will reduce a surge even more.

I hope that some of these tips will help improve the stability of your home automation system and you make installing new parts easier. As you can see, there are many areas you can tweak to improve an existing installation. If you have some tips of your own, post them to the HCS newsgroup. There are thousands of HCS-II users out there who will benefit. ☑

Mike Baptiste works for Nortel Network's R&D Facility in North Carolina's Research Triangle Park where he manages the Desktop and Intranet Services Support Groups. You may reach him at baptiste@cc-concepts.com.

RESOURCE

HCS newsgroup,
[news://bbs.circuitcellar.com/
local.cci.hcs2](http://news://bbs.circuitcellar.com/local.cci.hcs2)

REFERENCES

- Michel Clavette's HCS Website (HCS-II UPS)
<http://www.dsUPER.net/~clavettm/hcs.html>
- "The Art and Science of RS-485," Bob Perrin
Circuit Cellar Online – July 1999
<http://www.chipcenter.com/circuitcellar/july99/c79TOC.htm>
- "X-10 Technology Transmission Theory," X-10 Inc.
<http://www.x10.com/support/technology1.htm>
- "PL-Link Reset Fix," Creative Control Concepts
<http://www.cc-concepts.com/products/plfix/>

SOURCES

- All Electronics Corp.
(800) 826-5432
Fax: (818) 826-5432
www.allcorp.com
- Jameco
(800) 536-4316 • (415) 592-8097
Fax: (415) 592-2503
www.jameco.com
- Digi-Key Corp.
(218) 681-6674
Fax: (218) 681-3380
www.digikey.com
- Mouser Electronics
(800) 346-6873
Fax: (619) 449-6041
www.mouser.com
- ESD RS-485 and RS-232 ICs**
Maxim Integrated Products
(408) 737-7600
Fax: (408) 737-7194
www.maxim-ic.com
- ESM1 X-10 signal strength meter**
Home Controls, Inc.
(858) 693-8887
Fax: (858) 693-8892
www.homecontrols.com
- X-10 home-automation products**
X10 USA
(800) 675-3044 • (201) 784-9700
Fax: (201) 784-9464
www.x10.com
- Leviton Manufacturing Co., Inc.
(800) 824-3005
www.leviton.com

MULTIMEDIA EMBEDDED PC

The **SBC-MediaGX** is an EBX-compliant board based on the National Semiconductor MMX-enhanced 233-MHz MediaGX processor. It comes fully loaded with all the standard PC interfaces plus a complete range of multimedia features. Included on the board are video (Chips & Technologies 69000 HiQvideo graphics accelerator); support for STN, TFT and EL flat-panel displays; 10/100Base-TX Ethernet (PCI 2.1 compatible); flash-memory disk (solid state, maximum capacity 16 MB); and an analog resistive touchscreen interface. Also included are dual USB and Soundblaster interfaces, four 16550 fast serial ports (RS-232/422/485), and fanless operation to maximize MTBF. The board is capable of driving high-resolution modes on CRT and flat-panel displays simultaneously. Expansion is provided via PC/104 and PC/104-plus interfaces.

The SBC-MediaGX will run all popular operating systems including, DOS, Windows 95/98/NT/CE, QNX, and Linux. Development kits are available for the SBC-MediaGX, bringing hardware and software together to enable rapid development for new applications.

The SBC-MediaGX sells for **\$450** in quantities of 50 (memory excluded).

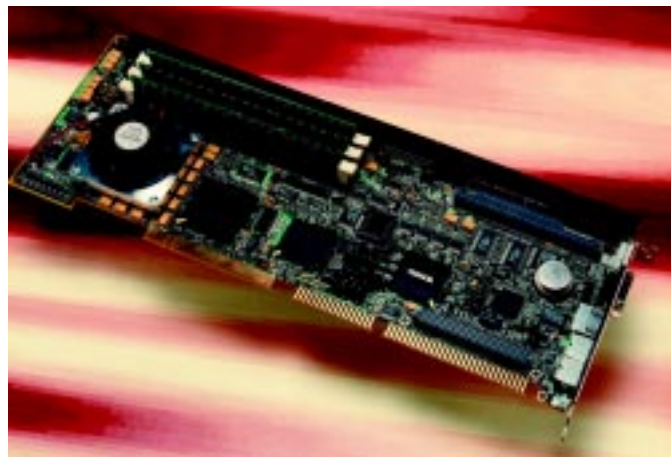


Arcom Control Systems
(888) 941-2224
Fax: (816) 941-7807
www.arcomcontrols.com

INDUSTRIAL SINGLE-BOARD COMPUTER

The **PCI-944 SBC** is an industrial single-board computer that features a 64-bit 333-MHz mobile Pentium II processor with a closely coupled 256-KB L2 cache and the 440BX Chipset. Included on board are a PCI 64-bit flat panel/CRT controller with 2 MB of SDRAM memory and Panellink interface, high-performance PCI Ultra DMA/33 IDE, Ultra Fast/Wide SCSI 3, and 10Base-T/100BaseTX Ethernet controllers.

The PCI-944 SBC includes the same features of the SEC cartridge Pentium II processor, namely, dual independent bus (DIB) architecture, dynamic execution, Intel MMX technology, and an on-die 256-KB 64-bit Level 2 cache running at full core speed. In addition to support for error checking and correction (ECC) memory using standard 72-bit 66-MHz SDRAM, the processor contains built-in power-management features that help manage power consumption and improve reliability. Universal serial bus



(USB) ports, serial/parallel ports, a floppy interface, CompactFlash disk support and a PC/104-Plus expansion connector are also onboard.

The Award Hi-flex BIOS, in bootable flash memory, supports serial/parallel port remapping/disable, console redirection, and advanced configuration and power management interface (ACPI). In addition to ISA and PCI bus mastering, the CPU supports PCI-to-PCI bridging, and has high ISA bus drive capability. Other

features include a CPU temperature monitor, two-stage watchdog timer, power-fail circuit, and a two-year warranty.

The PCI-944 with an Intel 333-MHz mobile Pentium II processor with video controller, but no SDRAM memory sells for **\$1,850**.

Teknor Industrial Computers, Inc.
(450) 437-5682
Fax: (450) 437-8053
www.teknor.com

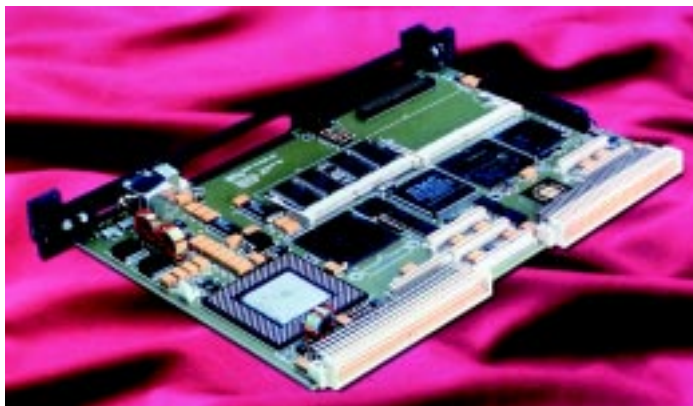
Nouveau PC

CPU BOARD

The **Mariner C157** is a CompactPCI board based on the embedded roadmap version of Intel's Pentium III (Coppermine) processor. The board occupies a single CPCI slot and provides a 600-MHz entry-level clock speed that is almost twice that of top-of-the-line mobile modules. The Mariner can also be equipped with a Celeron processor for applications requiring high performance at a reduced cost.

Featuring 128/256 KB of on-die L2 cache and 512 MB of field-upgradeable SODIMM main memory, Mariner is equipped with a 10/100 Base-Tx Ethernet interface and dual Ultra DMA-33 IDE hard driver interfaces. The Mariner also provides 32 MB of flash memory, a pair of USB ports, two PMC expansion slots, two comm ports, a printer port, a floppy port, and interfaces for both a mouse and keyboard.

Two PMC sites facilitate the addition of plug-and-play



communications, graphics, mass storage, and industrial I/O options. The Mariner is also available with an optional mass storage card that lets OEMs add a 2.5", 6.4-GB IDE hard drive (or 244-MB flash memory disk). When equipped with this mass storage card, the Mariner occupies just one CompactPCI slot.

Software support for the Mariner C157 includes Windows NT 4.0, VxWorks, Linux, Solaris, and QNX. The Mariner also provides 100% power-on-self-test diagnostics, which includes monitoring of all onboard voltages, temperature, fan speed, and other critical parameters.

The Mariner C157 starts at **\$1995** in single-piece quantities.

General Micro Systems
(909) 980-4863
Fax: (909) 987-4863
www.gms4vme.com

EMBEDDED BIOS ON-LINE ADAPTATION KIT

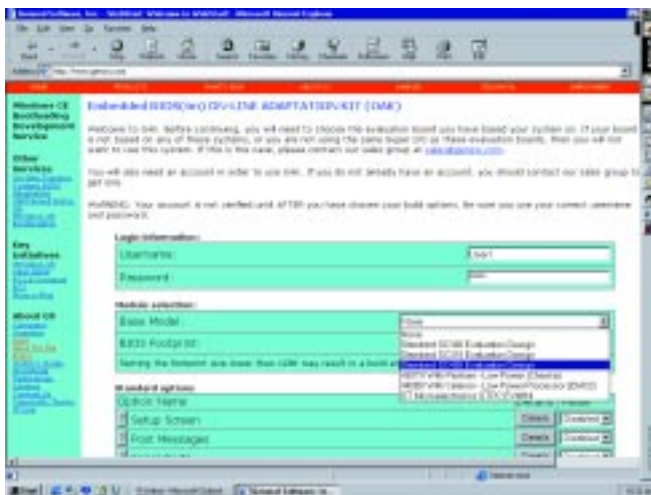
The **Embedded BIOS On-Line Adaptation Kit (OAK)** enables manufacturers to build custom BIOS firmware on the Internet. It provides a fast and easy way to configure a customized BIOS without actually using the source code.

A user subscribes to OAK during the hardware development phase of a new project, or when modifications to an existing platform are to be made.

After providing a valid subscriber ID and password on the OAK web page, the user selects a standard reference platform as the base model for a new BIOS. Then, by navigating the OAK menu, the user sets more than 200 options to define a unique configuration of the embedded BIOS firmware functionality. Once the BIOS options are set, the user submits a request for a custom BIOS binary file simply by clicking a button. The OAK

server accepts and processes BIOS build requests 24 hours a day and sends the resulting BIOS binary files, map files, and build logs to the subscriber's e-mail address in minutes.

The Embedded BIOS firmware provides a standard PC platform for 'x86 embedded applications. Features include built-in ROM, RAM, and flash memory disk drivers; a remote-access manufacturing mode for re-flashing the BIOS and updating the NT embedded OS and application software; a burn-in hardware diagnostics suite; and a comprehensive BIOS-level debugger used during board power-up. OEM engineers receive full source code and the BIOSStart rule-based expert system with the Embedded BIOS Adaptation Kit or can build customized BIOS files using the new On-Line Adaptation Kit.



General Software, Inc.
(800) 850-5755
Fax: (425) 454-5744
www.embeddedbios.com

Nouveau PC

A Matter of Time

Part 1: Accurate Timing and Frequency

All the talk about timing issues and the Y2K bug have Ingo thinking about the importance of keeping accurate time. Have a good time as Ingo kicks off this series with a look at crystal oscillators and some ways of synchronizing them.

OK, so we've made it to the year 2000. If everything went well, you should have this copy of *Circuit Cellar* and the Y2K bug didn't collapse the power grid, and many generators will soon be for sale. But, I guess the worst is yet to come.

This year is leap year, isn't it? Let's see, every four years, except for centenary years not divisible by 400.... Yup, this is a leap year. The irony here lies in the fact that really naive software (i.e., software that assumes every year divisible by four is a leap year) will get this right. But, this is not an article about glitches.

Starting out this year with a series about time seems fitting. I'm going to write about keeping time and methods we can use to make sure the time is correct and accurate. Having accurate time is important in many areas. File servers need to be in-sync with the clients that

use services, perhaps on the other side of the Pacific. Telephone exchange equipment needs to be perfectly synchronized to extract time-multiplexed channels from high-speed serial data. Astronomers need accurate time to synchronize observation in different parts of the world for long-base interferometry.

KEEPING TIME

Keeping accurate time is actually pretty hard, but there are several solutions,

depending on the accuracy required. Essential to keeping good time is some kind of oscillator or frequency standard.

To describe the "goodness" or quality of an oscillator, we use the terms accuracy and stability. The accuracy is pretty obvious—it's a measurement of how close the actual frequency is to the desired frequency. For example, a 5.0-MHz oscillator might actually run at 5.000001 MHz (i.e., 1 high in 5 MHz accuracy [2×10^{-7}]).

Stability describes frequency variations caused by changes in temperature, voltage, or time. A typical crystal oscillator might have a temperature stability of 100 ppm over its operating temperature range.

For any kind of reasonable frequency stability, we need at least a quartz crystal oscillator. Quartz crystals are devices that use piezoelectric effects to convert mechanical vibrations to elec-

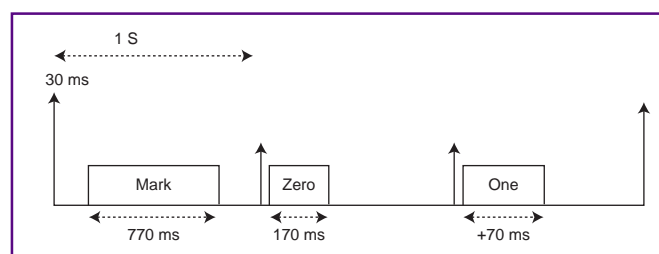


Figure 1—These are timings for the 100-Hz time-code subcarrier on WWW/WWVH. Each pulse represents the 100-Hz tone is on. The actual reference mark is 30 ms before the start edge of the tone and represents the time when there is a tick in the audio.

Bit	Function
0	position 0
1-9	zero bits
10	position 1
11-14	minutes (BCD)
15	zero
16-18	tens of minutes (BCD)
19	zero
20	position 2
21-24	hours (BCD)
25	zero
26-27	tens of hours (BCD)
28-29	zero
30	position 3
31-34	days (BCD)
35	zero
36-39	tens of days (BCD)
40	position 4
41-42	hundreds of days (BCD)
43-49	zero
50	position 5
51-55	zero
56	daylight savings
57-59	offset in ms to UT1

A typical stability value is 100 ppm (the crystal will be within 100 Hz/MHz over the temperature). For a 10-MHz crystal, the actual frequency may be anywhere between 9.999500 and 10.999499 MHz, which is adequate for general-purpose uses like a processor clock, or the dot-rate clock on a video card.

You may have noticed that when you leave your computer on for a long time, the system time will drift. This is because the processor clock is used to time the interrupt that the operating system uses to keep track of the time. Remember, a typical crystal-based oscillator can be off by 100 ppm. This works out to:

$$24\text{hr/day} \times 3600 \text{ s/hr} \times 30\text{days} = 2,592,000 \text{ s/month}$$

$$2592000 \text{ s/month} \times 100 \text{ ppm} = 259 \text{ s/month (4.32 min/month)}$$

You can increase the stability (and accuracy) by a factor of ten by synchronizing the system time to the BIOS time. The BIOS time is kept in a battery-powered real-time clock that's controlled with a more accurate tuning fork-type low frequency (32.768 kHz) crystal that can be as good as 10 ppm. When you reboot your system, the system time gets updated from the BIOS time.

Of course, these estimates are optimum, in reality they can be much worse. Many home PCs are not used in time-critical applications where the clock fre-

quency accuracy of the processor is really an issue. It's one place a PC manufacturer can cut costs by using even cheaper crystals and crystal oscillators, which have worse stability.

One way you can increase the stability of a crystal oscillator is to use a high-quality temperature-compensated quartz crystal. Although they cost more, these crystals can have temperature stabilities of 1 ppm over their operating temperature range (e.g., 0–50°C).

You can also fix the temperature of the crystal by using a temperature-controlled oven. With an oven-controlled crystal oscillator you can achieve impressive stabilities (1 part/billion). A frequency source of this type is likely to be an external component, such as the Hewlett-Packard 105B, which can be bought used for \$2000. However, crystals will drift in frequency over the lifetime of the crystal and need to be occasionally recalibrated.

For more stable oscillators, we will use the nuclear resonances in materials like Rubidium and Cesium gases. Cesium sources are the most stable, with long-term stabilities of three parts in 10^{12} and they cost the most (\$15,000 and up). A Rubidium source, on the other hand, has reasonable stability (one part in 10^{11}) and is cheaper (under \$10,000), but they age at a rate of one part in 10^{11} per month.

Because high-stability frequency standards are large, we can't really include them in a PC so they have to be external. Frequency standards can be used to provide a highly stable signal (100 kHz–10 MHz) that a computer can then use as an interrupt source. You can also compare a frequency standard to the clock frequency of your PC and then compute a correction factor that the software can use to correct the PC clock. This correction factor will work reasonably well for short times, if the temperature doesn't change much.

trical signals. Crystals resonate at some fundamental frequency that depends on the physical size of the material and the way it has been cut. One can also resonate a crystal at a multiple of the fundamental frequency (overtone) to get higher frequencies, but this mode of operation has more harmonics and is less accurate.

To build a crystal oscillator, we incorporate the crystal as a resonant filter in an amplifier that has a feedback path. There are several different crystal oscillator designs. Furthermore, for computer applications, you can buy complete crystal oscillators in a module that outputs a TTL- or CMOS-level square wave. Every PC has several quartz oscillators.

Quartz crystals suffer from temperature stability problems and aging. Manufacturing tolerance and in-circuit capacitance affect the initial accuracy. Stability is measured in ppm, and the accuracy in the number of decimal places shown in the frequency label. So, a 10.0-MHz crystal is not as accurate as a 10.000-MHz crystal.

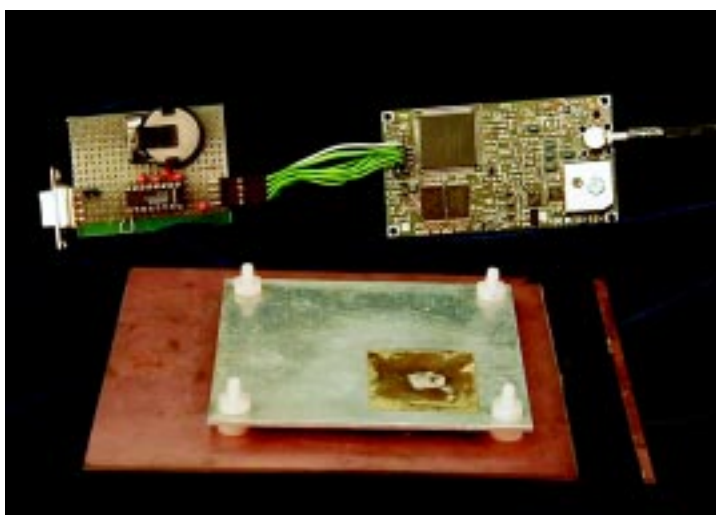


Photo 1—A GPS module can be used for timing applications. This one has a serial port and a pulse per second (PPS) signal. The serial port transmits the date/time, while the leading edge of the PPS signal identifies the exact moment the second starts. The protoboard has a backup battery for the GPS module and RS-232-level converters. The flat metal object is a home-built patch antenna.

SYNCHRONIZING COMPUTER TIME

Synchronizing a computer to a frequency standard is usually done with some kind

of time code. In the laboratory, if you had a frequency standard available, you would feed the output of the standard into a time-code generator. A time-code generator is a real-time clock that will encode the current time using a bit serial protocol on its output. The protocol varies, but is usually one of the IRIG standards developed by the Range Commanders Council, or based on one of the IRIG codes.

A single time-code signal thus encodes data and because the data rate is derived from the frequency standard, the data transitions can be used to synchronize to the frequency standard. Time codes can be recorded on tape along with data, distributed with distribution amplifiers, and sent long distances over coax or RF. I'll describe IRIG code(s) in detail next month.

Not everyone can afford a time or frequency reference, so I'll discuss the many time synchronization sources that are freely available. Most of these are radio based and thus aren't as accurate as having your own frequency standard. The inaccuracies are due to variable propagation delays and this introduces uncertainties. However, because these timing sources are themselves based on Cesium sources, they do have excellent long-term stability.

Probably the best-known radio time signal is WWV/WWVH. This signal is transmitted by NIST (National Institute of Standards and Technology) on several shortwave frequencies (2.5, 5, 10, 15, 20, and 25 MHz). The WWV/WWVH timing signal has data components and voice announcements. For computer timing, there is a pulse-per-second clock tick and tones that can be used to find the beginning of the minute (1 kHz) and hour (1.5 kHz).

More interestingly, WWV/WWVH also transmits a 100-Hz subcarrier that is used to encode 60 bits of information each minute. Each bit is encoded as either a 0.170-s or 0.470-s 100-Hz tone for zero and one marks and 0.770 s for a positional mark. Figure 1 shows the timing of the pulses involved.

The reference for each pulse is actually 30 ms in front of the pulse start. The data-bit position encoding for WWV/WWVH is shown in Table 1. It's fairly easy to build a shortwave radio receiver and decode this tone. Also, several commercial receiver models exist.

Bit	Function
0	position 0
1	position reference
2-4	40,20,10 minutes (BCD)
5	zero
5-9	8,4,2,1 minutes (BCD)
10	position 1
11-12	zero
13-14	20,10 hours (BCD)
15	zero
16-18	8,4,2,1 hours (BCD)
19	zero
20	position 2
21-22	zero
23-24	200,100 days (BCD)
25	zero
26-29	80,40,20,10 days (BCD)
30	position 3
31-34	8,4,2,1 days (BCD)
35-36	zero
37	plus
38	minus
39	plus
40	position 4
41-44	0.800,0.400,0.200,0.100 ms UT1 offset (BCD)
45-49	zero
50	position 5
51-55	zero
56	leap year
58	zero
59	daylight saving

Table 2—Here are the bit assignments for WWVB timecodes. In this timecode, we find the beginning of the frame, but notice that there are two position identifiers in the beginning. The second position identifier is a position reference, and its leading edge defines the exact time the minute starts.

Because WWV/WWVH uses short-wave frequency, the arrival time of the signal is not very accurate. Depending on conditions, shortwave signals bounce around in our ionosphere and it would be impossible to know how far the radio signal travelled to get to the receiver. This uncertainty can be up to tens of milliseconds. But WWV/WWVH is simple to use and accurate enough to synchronize the time on a PC for many applications.

Canada also has a time station (CHU) that broadcasts time at 7.335 MHz. CHU transmits pulse-per-second ticks and time data using a 1200 bps FSK at several specific time intervals each minute. Because CHU is also shortwave-based, it has about the same performance as WWV/WWVH.

NIST also operates a VLF (very low frequency) time station beacon at 60 kHz called WWVB. The propagation is more predictable than at shortwave frequencies, and once you've measured the propa-

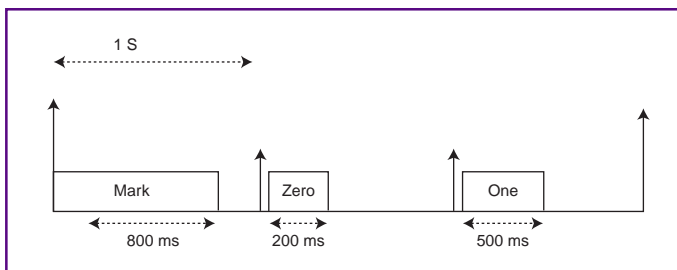


Figure 2—For WWVB, the 100-Hz timing is similar to an IRIG-H signal. Each pulse represents a 100-Hz pulse. So, a 200-ms pulse represents two cycles, a 500-ms pulse—five cycles, and so on. The reference is the leading edge of the pulse, which is the zero crossing time of the first cycle in the 100-Hz tone.

gation time from the station to your location, this method is accurate down to 1–10 μ s. Similar to those in the shortwave band, this station transmits a 100-Hz time-code signal. Here the timing is 0.2 s for zero, 0.5 s for one, and 0.8 s for a mark. The timing is shown in Figure 2 and the reference is the leading edge of the timing pulse. The encoding differs from WWV/WWVH as seen in Table 2. WWVB also has no voice announcements.

Again, commercial receivers exist for this time station. Next month, we will look at an economical receiver and interface it

to a PC running RT-Linux. A high-end WWVB receiver can cost over \$1500.

There are several other beacons in the VLF band. There are two time stations in Europe. The one in England transmits time code on 60 kHz and the one in Germany is

DCF77, which broadcasts at 77.5 kHz.

In the US, we can also use Loran-C beacons. These navigational beacons work roughly as follows. There is a master station and several slave stations. The master station sends a start pulse and the slave stations respond with their own pulses. To measure your position, you would measure the relative delay of the stations in their time slots and by knowing the location of the station, you can triangulate your position.

Even though Loran-C beacon stations are used for navigation, the station's

broadcast frequencies and pulses are all synchronized using Cesium frequency standards. We can use the frequency and pulse timing to synchronize our clock rate down to ± 500 -ns accuracy, if you know the range to the nearest beacons. Loran-C stations do not transmit absolute timing information as WWVB does, so you'll need some way to initially set the clock on your PC.

VLF beacons have one nice feature. The frequency is low and will penetrate buildings and structures easily. Often only a simple loop-stick antenna is needed to pick up VLF stations.

If your PC connects to a digital telecommunication network like ISDN or T1, you can use the recovered data clock as a frequency standard. The national digital telecommunication network is synchronized using frequency and time standards.

Essentially, the phone system works because it uses time-division multiplexing and you need accurate time to find out the time slot that data needs to be inserted. If you have a synchronous communication card attached to a T1 or ISDN network,

you can use the interrupts from this to synchronize and calibrate your PC.

Finally, there is GPS. Each GPS satellite has an atomic clock that is monitored and calibrated. Each satellite broadcasts its current time, which is received by a GPS receiver. By knowing where the satellite is located, the receiver computes the range to each satellite and thus uses trilateration to find its location. Once you know where you are, you can then compute the time.

Accuracy of ± 300 ns can be achieved with commercial GPS time receivers, and up to ± 200 ns if you average your location over several days to eliminate the diather that the GPS system introduces. With a consumer-grade receiver module, you can achieve accuracy down to microseconds. Photo 1 shows a GPS module from Trimble. This module will provide a pps (pulse per second) signal and a serial data stream for the time/date data.

However, one of the problems with GPS is that it's a line-of-sight system. The antenna must have an unobstructed view

of the sky. Also, because the system uses microwave frequencies, the antenna can't be too far from the receiver, unless high-quality coaxial cable is used.

Having the high accuracy of GPS available even with an inexpensive GPS receiver makes the other time stations mostly obsolete. As I mentioned before, WWVB (and other VLF stations outside the US) have an advantage if you can't erect a GPS antenna with a clear sky view.

TIME OUT

Now, you know what timing and frequency sources are available to synchronize the time on your PC. Next month, I'll show you how you can interface an economical WWVB receiver to a PC and generate an IRIG-B signal. This will illustrate some common software techniques you can use. OK, so maybe you're not as much of a time junky as I am. Perhaps, my obsession with time and time signals stems from my German heritage. [RPC.EPC](#)

Ingo Cyliax has written for Circuit Cellar on topics such as embedded systems, FPGA design, and robotics. He is a research engineer at Derivation Systems Inc., a San Diego-based formal synthesis company, where he works on formal-method design tools for high-assurance systems and develops embedded-system products. You may reach him at cyliax@derivation.com.

SOURCES

WWVB Receiver

Parallax, Inc.
(916) 624-8003/8333
Fax: (916) 624-8003
www.parallaxinc.com

Ultralink, Inc.
1547 Anthony Ct.
Gardnerville, NV 89410
(775) 782-9758
Fax: (775) 782-2128
www.ulio.com

Timecode and Frequency Standards

Products

Datum, Inc.
(512) 721-4000
Fax: (512) 251-9685
www.datum.com

GPS Time Receiver Products

Trimble Navigation Ltd.
(408) 481-8000
Fax: (408) 481-6885
www.trimble.com

REFERENCES

P. Horowitz, *The Art Of Electronics*, Winfield Hill, Cambridge University Press, Cambridge, UK.
Range Commanders Council, *IRIG Standard 200-98*, <http://tecnnet0.jcte.jcs.mil:9000/RCC/oldoc.htm>
Datum, Inc., *Timing & Time Code Reference*, Datum Inc., http://www.datum.com/res_technical.html

Applied PCs

Fred Eady

The Mockingbird Trial

PIC vs. 80188

Not the sequel to the Harper Lee classic, but Fred's account of fitting a PIC peg into an 80188 hole. The PIC17C42A can sing a variety of songs, Fred shows us how to reason when it's a viable replacement for the 80188.

As a teen, I was into music. There were bands that rocked and bands that, frankly, did not rock. (Some of them are still attempting to rock today.)

There were "hip" songs and there were Gregorian chants. Good or bad, it was all music. What I called "good" was dog-awful to some of my friends, and some of the stuff my friends drooled over I couldn't swallow with a bucket of water.

Embedding a processor complex into a product is akin to music appreciation. In each discipline, there differing beats and messages wrapped in a multitude of formats.

The ear and emotions of the listener determine the mood the music extends. In a similar fashion, ease of product development and end-user functionality is music to an embedded developer's ears.

Been to a music store lately? There are literally thousands of artists to choose from.

Shopped for an embedded hardware platform lately? Yep, there are hundreds of vendors and thousands of products. There are even some vendors sporting exclusive catalogs of nothing but embedded hardware.

Where am I going here? Think about this. Dolly Parton's "I Will Always Love You" was done later by Whitney Houston. Same melody, same words, two very different artists. The result? A hit song both times.

So, with that, instead of introducing a fancy new Intel-processor-based SBC, I'm going to step off the normal embedded PC trail into the embedded PC twilight zone and let you listen to an old 80188 song as performed by a PIC.

SINGIN' LIKE A MOCKINGBIRD

The 80188 and the Microchip embedded way of doing things aren't really so different on the result side of the equation. But, getting them to that final result is a very different process.

I'm not here to take sides. There are instances where a PIC-to-80188 reverse process just won't work.

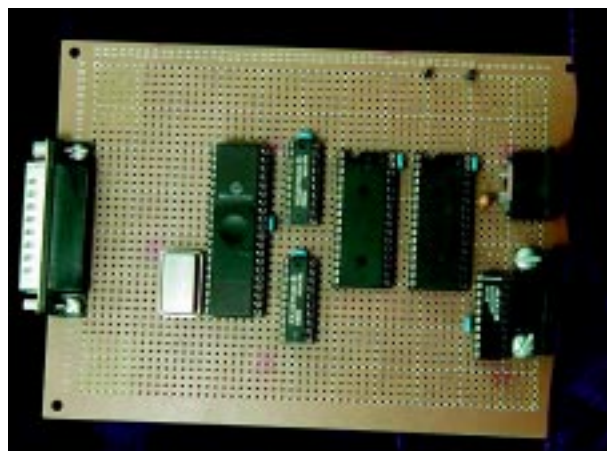


Photo 1—Note the two single binding posts near the power connector. I tend to smoke-test stuff often (especially when I use point-to-point construction). So, I added a couple of logic-probe power pins. Note also the open pins on the SRAM sockets that can take on higher density SRAM, EPROM, or flash memory.

There are just as many instances where it will.

My purpose is to illustrate what it takes to stuff a PIC where an 80188 or equivalent would or could normally go. To that end, I assembled a typical PIC17C42A embedded complex that I call the Mockingbird. A birdwatcher's-eye view of the Mockingbird can be seen in Photo 1.

Just like the 80188 (and the real-life feathered flying chirper), the Mockingbird can assume many differing roles depending on its firmware load and the type of electronics externally attached to its I/O port pins.

Programming devices like flash memory, EEPROMs, and PICs are "in." So today, let's make device programming the Mockingbird's example application.

HARDWARE IS HARDWARE IS HARDWARE

The PIC17C42A firmware is assembled using a 58-word command set, with each command consisting of 16 bits. The upside to this is, if you've programmed with other parts of the PIC family, you can

program this part with a minimal learning curve. And if you don't know how to pronounce "PIC," you're not at a great disadvantage either. All of the '17C42A instructions execute in a single cycle with the exception of program branch and table read/write instructions, which take two cycles.

Mockingbird's instruction cycle is 250 ns with a 16-MHz clock. The maximum clock frequency for the Mockingbird with a '17C42A is 33 MHz, which yields a 121-ns cycle time.

I said I wouldn't take sides, but in the real world, the PIC is quite a bit faster than the 80188 on instruction cycle times. Time-based or compute-intensive applications may force you to count the nanoseconds, but instruction-cycle times are of no concern in our device-programmer application.

About 100 commands make up the 80188 command set. For programmers who know 8088, 8086, 80386, and 80486 assembly language, programming the 80188 is not much different because the 80188 is actually a core 808x with commonly used 808x peripherals built on

the chip.

A typical 80188 variant like AMD's can operate with a maximum clock frequency of around 40 MHz. At that speed, a 70-ns memory device can be accommodated. The 16-MHz memory access calculations for Mockingbird's '17C42A also specified a 70-ns SRAM.

As you see in Figure 1, the standard '17C42A bears a total of 33 I/O lines. Mockingbird uses many of them to implement the external SRAM. The remaining I/O pins are programmable and thus multipurpose.

By setting bits within internal '17C42A registers, the Mockingbird's I/O lines are capable of interrupting the processor on input change, capturing pulse widths and PWM output, and timing functions.

This PIC is a full-function microcontroller that includes interrupt capability, a 16-level stack, direct and indirect addressing modes, and 64K × 16 addressable memory space. And, a synchronous/asynchronous USART is standard equipment.

The 80188 has 32 programmable I/O pins and 64 KB of I/O address space. The PIC17C42A can be configured to imitate

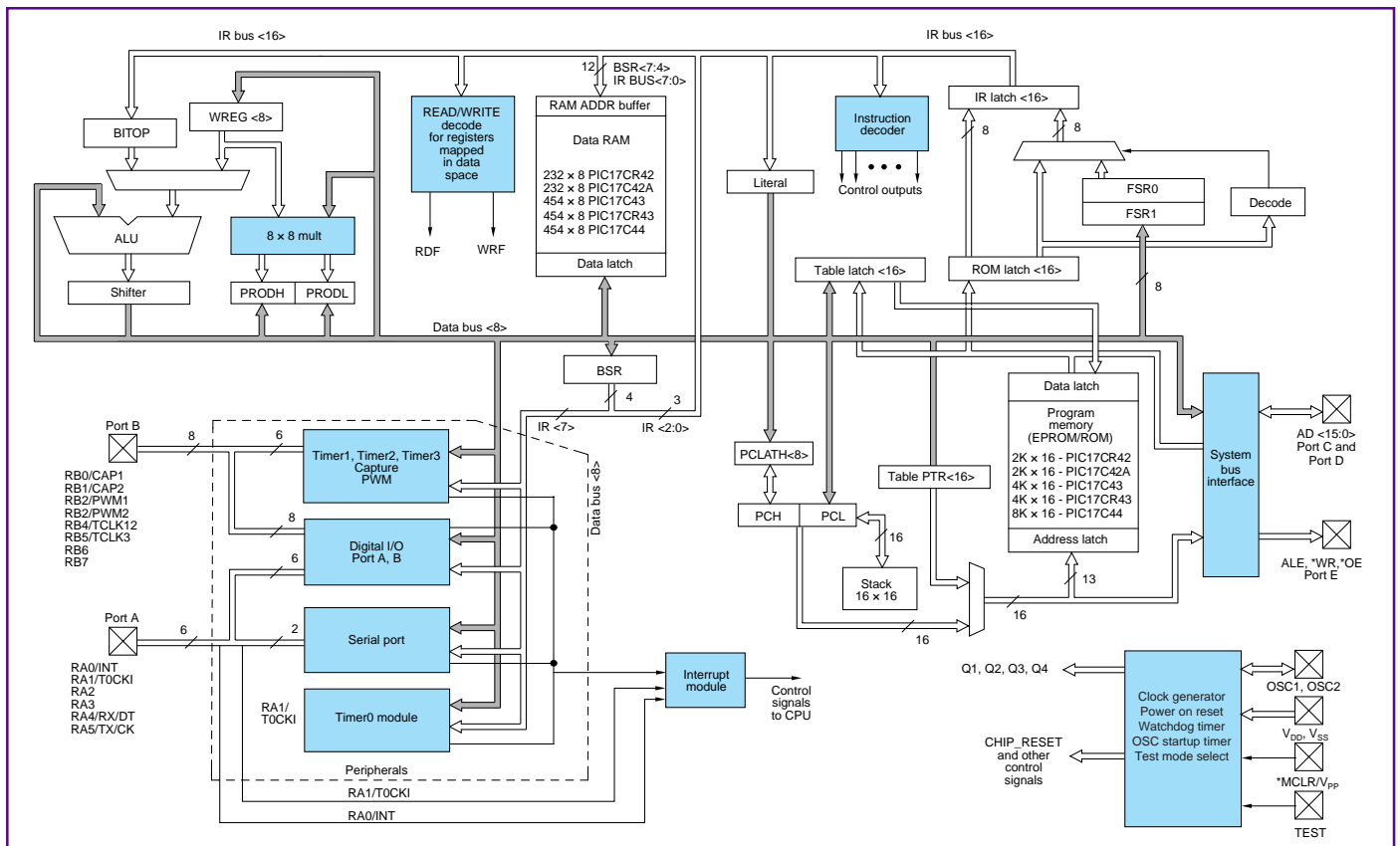


Figure 1—The PIC17C42A is designed to be a microcontroller in the standard sense and a microprocessor in the embedded sense. Note that there are many internal RAM sizes available in the PIC17C4x family.

the 64 KB of I/O space, but that comes with the negatives of added address decoding hardware and support code overhead for the decode hardware.

The 80188 can address 1 MB of memory and includes all of the interrupt and addressing modes to match or surpass the '17C42A. Again, 1 MB of memory can be accessed by the PIC, but with the same expense as adding "virtual" I/O address space.

Two full-featured serial ports are included with most 80188 variants, and using them as such will sacrifice some of the 32 dedicated I/O lines. The AMD part even includes a multidrop 9-bit serial port protocol and a PWM input.

Although it's supported on the 80188 parts, DMA capability is nonexistent on this manufacturer's microcontroller. A logical block diagram of AMD's 8018x part is shown in Figure 2.

The PIC17C42A can access external static RAM, EPROM, and flash memory as well as its internal EPROM and RAM (see Figure 3). Program execution between internal ('17C42A EPROM) and external memory (SRAM in the Mockingbird) is automatically taken care of by the PIC's microcode.

A 16-MHz oscillator drives the micro's 16-bit processor core, which is complemented by 2 KB of 16-bit on-chip EPROM. And, there's plenty of 8-bit on-chip RAM (232 bytes) that is augmented by 30 KB of fast external 16-bit SRAM.

Two industry-standard 74LS573 latches tie the external SRAM to the PIC's 16-bit memory interface. In addition to the micro's in-house features, the Mockingbird's 7-IC configuration includes 10 bidirectional I/O lines, two input-only data lines, and a serial port.

Two of the 10 bidirectional lines (RA2 and RA3) are high voltage, high current (+12 V at 60 mA) open-collector I/Os and are pulled up by resistors R1 and R2. Most of the Mockingbird I/O is accessible via a female DB-25 connector.

With the exception of the high-voltage pins and a few of the memory statistics, I could almost write exactly the same paragraph to describe the 80188. The similarity also shows up on the drawing board.

The 80188 configuration in Figure 4 looks much like the '17C42A lashup you see in the schematic in Figure 3. Both the

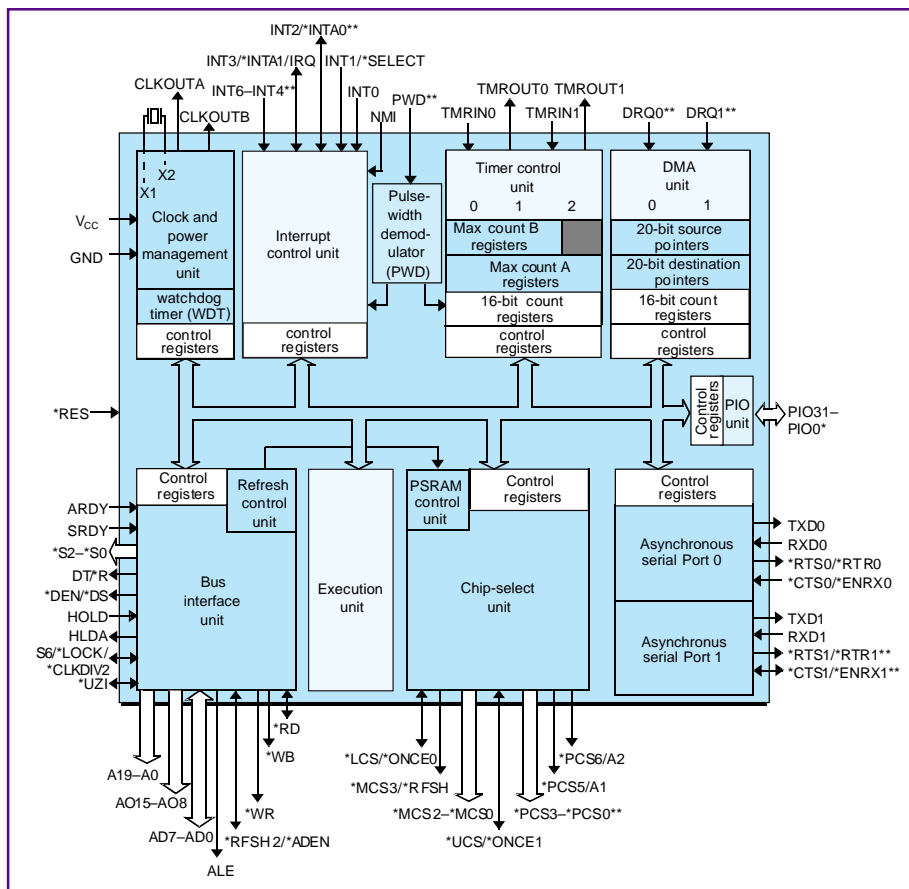


Figure 2—Remember the old 8088 in the first personal computers? Well, this is it with the addition of powerful on-chip peripherals that eliminate most of the support ICs necessary to run an 8088. Here you see AMD's 80188 variant.

80188 and the PIC were designed to minimize external parts count.

Even with all this power, the Mockingbird or an 80188 system is useless if data can't be passed between the user and the application. A nine-pin female shell connector is the portal to the Mockingbird's Maxim MAX233 RS-232 IC.

The Maxim IC interfaces a PC's serial port to the microcontroller serial port and is used here in its standard configuration. The Maxim or any other similar RS-232 or RS-485 part would be implemented identically in a similar 80188 embedded system.

I've seen designs that cleverly substitute discrete components that emulate an RS-232 driver/receiver. They work just fine. The advantage of using "real" bilevel RS-232 signals from the MAX233 is that noisy, fluttering, or marginal RS-232 voltage levels won't fool the host and embedded serial ports.

The Mockingbird firmware communicates with the host PC control program at 19.2 kbps. The 80188 serial ports can chirp this fast, too.

MOCKINGBIRD'S MONITOR

Although the 80188 and PIC17C42A hardware is proven, the monitor and application code executing in external SRAM form the foundation that enables the universal nature of both devices. The monitor firmware that resides within the micro's on-chip EPROM is written to take advantage of a powerful feature of the PIC17C42A—table read/write.

In a word, table read/write instructions let the user/programmer read and write the PIC17C42A's 30 KB of external SRAM in 8- and 16-bit word lengths. Data is transferred in 16-bit chunks between its internal registers and the bank of SRAM.

Once inside the PIC's holding register, the data can be manipulated as two 8-bit words or a single 16-bit word. Once the desired data is transferred to the external read/write memory, that data, if it is an application program, can be addressed and executed. If the downloaded data is not an application, then it can be manipulated just as you'd manipulate data in any other RAM storage area.

Taking this concept a step further, this implies that Mockingbird programs and data can be downloaded from the host PC and stored in Mockingbird's 30 KB external RAM area using the PIC's table read/write capability. On the other side, data can be uploaded to the host PC for processing.

Unless you build your 80188 system from scratch, some type of monitor will most likely be included with the 80188 SBC you purchase. The 80188 is structured in a logical fashion, and hand-coding a monitor similar to that of the PIC17C42A's shouldn't present any major problems. All of the 80188-based SBCs I've encountered and described in past columns have included good monitors.

In this application, the '17C42A monitor code is responsible for receiving and decoding commands in the form of ASCII characters from the host-PC control program via the serial port and placing downloaded data and applications into designated areas of the 30-KB external RAM space.

The monitor code has the ability to test the 30 KB of SRAM, clear the SRAM, and

dump the SRAM to a standard Intel hex file on the host PC. This code resides in the first 2 KB of program space, which is the on-chip '17C42A EPROM. This PIC17C42A-resident 2 KB is part of the 32-KB total memory area of the Mockingbird.

If you choose not to implement code within the micro, you may install a blank '17C42A on the Mockingbird and install EPROM where the SRAM resides. This approach assumes you have access to an EPROM programmer, as the monitor code would reside there with your application code.

Of course, the drawback to this modification is that the Mockingbird would be dedicated to the EPROM application and SRAM would be nonexistent. Another approach would be to use battery-backed SRAM or flash-memory form "instantly reprogrammable" EPROM.

Mockingbird relies on some preprogramming of vital components to implement a "programmable" PIC SBC. In the 80188 environment, the same preprogramming would be necessary, and depending on the complexity of the off-chip peripheral set, that may be as little as a

single PAL or a couple of EPROMs or flash ICs.

Although you can write the host-PC control code in any language, my device programmer is written with Bill's Visual Basic 6.0. All of the command codes and data passed between the Mockingbird and the host PC are ASCII characters or Intel hex files. Any language that can address the PC serial port and send and receive ASCII files can be used to deploy an application on the Mockingbird.

If you choose to program a PIC device, the Mockingbird device programmer can be used with the Microchip MPLAB 4.0 environment, which can be downloaded free from Microchip's web site. MPLAB enables the user/programmer to write, test, and simulate PIC code for the entire PIC line.

MPLAB includes an assembler that produces an Intel hex file that can be downloaded to the Mockingbird via our VB6 PC host program. MPLAB can be used to write, test, simulate, and assemble both the Mockingbird device-programmer application code and the code that will be programmed into the target PIC device.

I haven't attempted to find it, but I'm sure there's an abundance of 80188 freebie development code floating around on the Internet. Although the "free" development environment offered by Microchip is an advantage here, you can spend just as much money on serious PIC development tools (i.e., ICE boxes and in-depth development/debug software) as you would on 80188 building blocks.

Either way, on the PC side, the way you approach an application depends on what compilers and development tools you already own or can find on the web and whether you desire to run under DOS or Windows. There is no reason that you can't write a host-PC control program using any version of DOS BASIC or DOS-based C.

At a minimum, if you own a PC, most likely you own a

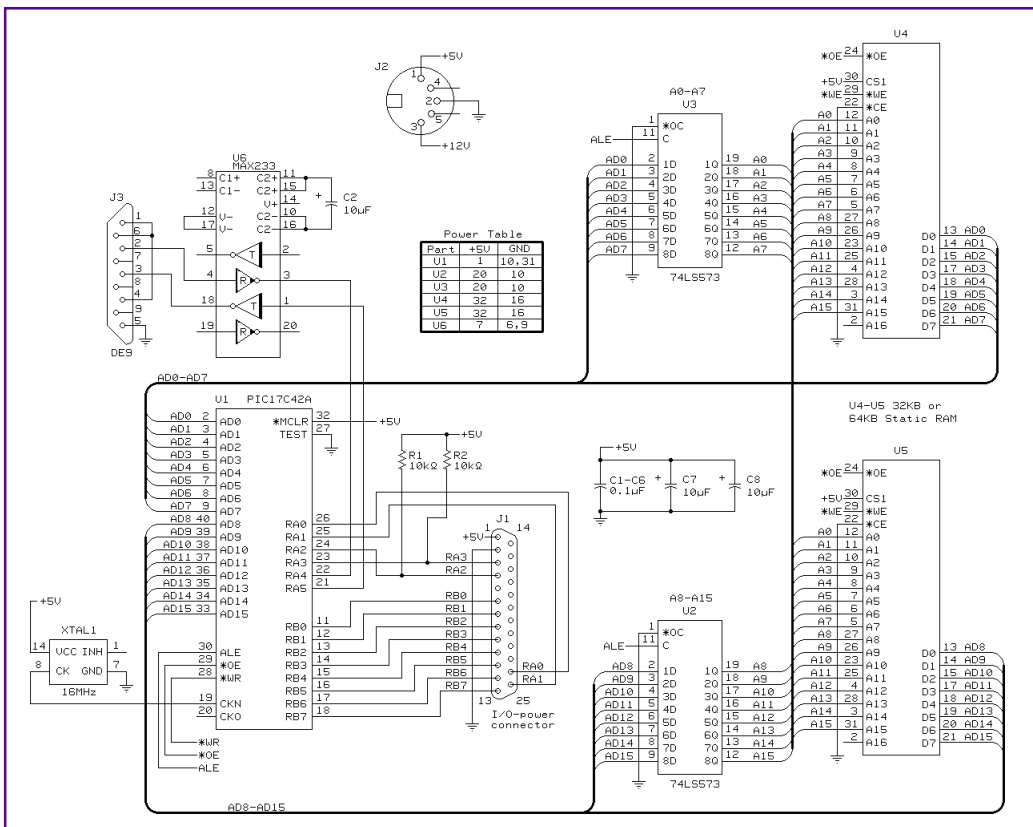


Figure 3—The Microchip PIC17C42A didn't spend years gestating inside a personal computer, but it can access all kinds of memory devices, both internal and external, and has adequate capabilities to keep part counts down.

copy of DOS. QBASIC included with DOS 6.22 in this case is virtually free.

With Windows in the forefront these days, and the beauty of using a GUI, I wrote this app with VB6 to run under Windows 95/98/NT. Visual Basic code tends to be self-documenting if you use real words. (I did.) So porting the basic program theme to other languages like Bill Zales's PB5 should be relatively painless.

The executable program area defined by the PIC17C42A monitor begins at 0x800, which is SRAM just above the EPROM program area of the PIC. The program space defined for the device programmer application including the micro's EPROM area is 16KB x 16 and ends at 0x3FFF.

The data area begins at 0x4000 and extends the remaining 16KB to end at 0x7FFF. This memory configuration allows the programmer to write a '17C42A application that could be a maximum of 30 KB.

Of course, this assumes that no external RAM needs to be accessed. For this device programmer, the memory is allocated so that the actual program size is held down to 14B x 16 and the maximum data space size is 16 KB of 16-bit memory.

This memory-mapping scheme equates to 16 KB of PIC instructions that can be downloaded and transferred to the Mockingbird using a 14-KB '17C42A application. If this isn't enough space, by simply adding larger SRAM ICs, Mockingbird can be upgraded to address 64 KB of 16-bit memory.

A command-line interface can be used to talk to the Mockingbird, but a GUI is

more user-friendly. Photo 2 is a screenshot of the device-programmer GUI. From the command button captions, you can see that the Mockingbird's monitor recognizes the Load Program, Load Data, and Run commands.

Earlier, I pointed out that the monitor also had the ability to test, clear, and dump the SRAM. As you see, there are also command buttons for those diagnostic tasks.

By using MPLAB to build the application code that is ultimately programmed into a target, the user/programmer can incorporate all of the PIC configuration information into the source file. This configuration fuse information is inserted into the assembler's Intel hex output file and used by our device programmer's application code.

Embedding the configuration fuse information into the assembler's output Intel hex file eliminates the need to display and manipulate configuration settings with your host-PC control code. Because I don't have to manipulate the configuration fuse information, it simplifies the device-programmer GUI.

The location of the fuse information for Mockingbird is consistent with MPLAB's placement of that same data. For example, `__CONFIG _WDT_OFF & _CP_OFF & _XT_OSC` sets the watchdog timer off (WDT), code protection off (CP), and instructs the part to use the XT oscillator option.

For the PIC12C508, this information would be placed at 0xFF inside the PIC and 0x1FFE in the Intel hex file. This location (0xFF) is where the configura-

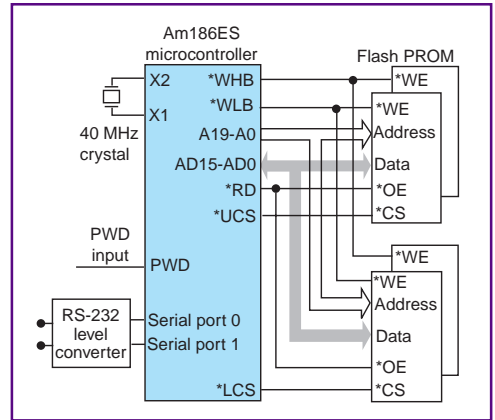


Figure 4—The song remains the same. Only the names of the control lines are different between the PIC and the 80188.

tion word is found on the '12C508. Obviously, this is just a SMOP (simple matter of programming) for both the 80188 and '17C42A.

I chose to incorporate the PIC programmer command buttons into the same window as the monitor commands. The PIC programmer buttons become active when the PIC programmer application code is loaded and executed.

The Load Data command button loads the code that gets programmed into the target PIC. The Run command button does double duty as it changes its caption to an exit message while the application is executing.

A COM port status area and a communications status area round out the host control GUI. The host-PC program keeps in constant communication with the Mockingbird and this is displayed in the Heartbeat area. By "reusing" command buttons, the GUI design is simple and highly functional because it is programmable and can be configured for multiple devices.

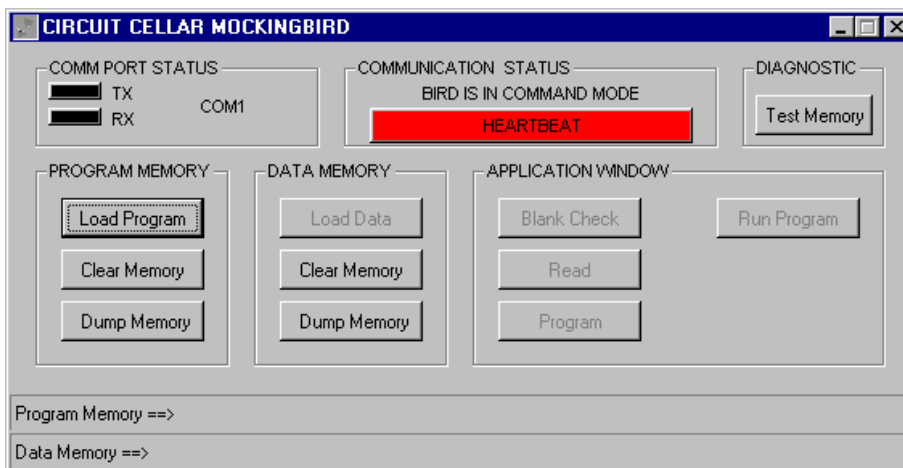


Photo 2—Say what you will about Bill G., but it's hard to beat his GUI-based compilers for good looks and ease of assembly.

TELLING STORIES

Comparing Figures 1 and 2, you'd conclude that the mechanics of assembling an 80188 versus a PIC17C42A embedded system would be worlds apart. It's obvious that the 80188 is much more bus-oriented with tons of sense and status lines popping out of its silicon, whereas the PIC17C42A is, well, a big, fat microcontroller.

Underneath it all, both processors are capable and programmable in terms of I/O. Many of the advantages the 80188 has in on-chip hardware can be emulated by the '17C42A with some tricky coding.

Conversely, hardware goodies on this micro like PWM and pulse width capture

can be SMOPed on the 80188. The high-voltage I/O lines on the '17C42A become a couple of transistors and some resistors if such a feature is needed on an 80188 embedded system.

As I mentioned earlier, there are circumstances and applications where a PIC or any other similar micro simply can't do the job an 80188 would. The whole discussion boils down to what is easiest to implement, what has to be done and how quickly, and most importantly, how much it costs to produce.

If the programmer application required DOS to run at the embedded end, the '17C42A would have been quickly eliminated as a processor choice. If your shop were full of PIC coders, the 80188 wouldn't stand a chance. A C programmer assigned to this project wouldn't care which processor was chosen because his or her C code would look pretty much the same either way.

For simple dedicated projects like this one, it may cost less to use a PIC instead of a more full-featured microprocessor complex like the 80188. That reminds me of a story....

While attending a Microchip seminar, I overheard a heated conversation between a couple of seasoned embedded programmers as to why they should be forced to move the projects they develop from their familiar 68k platforms to the "lowly" PIC.

"I can do anything that PIC can do and just as fast!" one of them said.

"OK, what do you do with all of those leftover cycles and hardware on the small projects?" the other retorted.

As you can imagine, this went on all morning and into the afternoon breaks. I learned a great deal about the 68k platform just listening to these guys wrangle. Neither party yielded, yet both were "sent" to this seminar by their supervisors.

It became obvious to me that neither of these guys had either read nor had anything to do with their company's financial statement. The 68k and its spin-offs were and are a viable embedded solution for a host of applications, but anyone else at that seminar would have told them that you don't put a \$1 part in a \$0.59 product.

Those two guys were wearing blinders and couldn't see that they were being

offered an alternative method to be heroes. Having the 68k and PIC join forces in their company's products was their reason for sitting there beside me.

I couldn't imagine designing everything with a 68k! I sincerely hope these folks read *Circuit Cellar* now because at that time they obviously didn't know that it doesn't have to be complicated to be embedded. [APC.EPC](#)

Fred Eady has over 20 years' experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

SOURCES

PIC17C42A, PIC12C508

Microchip Technology, Inc.
(888) 628-6247
(480) 786-7200
Fax: (480) 899-9210
www.microchip.com

MAX233

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

MICRO SERIES

Mark Balch

High-Definition TV

DTV System Architecture

Part
3
of
3

Now that Mark's given us some information on HDTV signal technology, it's time to look at the distribution and production aspects of this versatile new medium and just how it will benefit viewers.



igital television (DTV) is bringing completely new equipment and paradigms to the U.S. terrestrial broadcast industry. Not only do content producers have their choice of multiple digital video formats, but broadcasters are being given powerful flexibility in the range of services they can offer.

These changes come at a cost—all new studio, production, and transmission infrastructures. However, DTV's many enhancements stand to benefit both consumers and the broadcast industry once the initially choppy waters of the DTV transition period become calm.

The FCC's current plan calls for termination of the familiar analog NTSC TV broadcasts at the end of 2006. From May 1, 1999 through 2006, the U.S. is embarking on the largest single move in television technology since the beginning of broadcast TV earlier in the twentieth century.

We're leaving the realm of purely analog video and entering the domain of digital pipelines carrying bits and bytes in myriad groupings. Whereas a TV used to be as simple as extracting the NTSC signal from a modulated RF carrier and displaying it on a CRT, DTVs incorporate MPEG-2 decoders

and RTOSs running on embedded microprocessors.

Parts 1 and 2 discussed the nature of the digital video formats and the data structures that enable the proper reception and viewing of DTV programs. However, you can't gain a complete grasp of DTV production and broadcast without understanding how the many new pieces of equipment fit together to form a usable system. Let's get started on that journey and finish up this series.

BROADCASTING BITS

Unlike a conventional TV broadcast signal, where the analog video signal directly modulates an RF carrier, DTV's RF channel carries a general bitstream. Therefore, DTV is a flexible digital medium whose bits can be allocated according to the business model of a particular broadcaster. This contrasts with analog TV, where all broadcasters have to transmit the same basic signal.

The FCC chose 8-VSB (vestigial side band) as the mandatory modulation scheme for terrestrial DTV broadcast in the U.S. Conventional TV transmission is performed with an analog VSB modulation scheme. Given the 6-MHz channel bandwidth allocated to individual broadcasters, 8 VSB yields a digital bandwidth of ~19.39 Mbps.

Aside from the technical controversy surrounding the FCC's selection of 8 VSB versus COFDM (coded orthogonal frequency division multiplex; a popular modulation scheme used in Europe), there are some inherent signal-reception differences between DTV and conventional TV. In short, there is the digital "cliff effect" versus a more gradual degradation with analog.

People have become used to the idea that their rabbit-ear antennas might not provide clear reception—especially when the TV is located outside the broadcaster's primary coverage area. However, these people are still able to watch TV despite the degradation in quality.

Digital communications systems tend to exhibit a flatter quality versus signal-strength curve until a certain minimum signal threshold is reached.

At this threshold, the bit error rate of the channel increases dramatically and the integrity of the data link falls apart.

Therefore, it's realistic to expect that some people will experience higher quality TV reception while some are unable to receive a usable DTV signal with their current antenna configuration. The exact distribution of these differences in a given area depends on many variables, including local terrain, tall buildings, transmission power, and individual antenna size and placement.

TRANSMITTING BITS

Recall that the DTV bitstream is an MPEG-2 transport stream and that MPEG-2 transport streams are formed by the combination of multiple elementary streams in a multiplexer. The Society of Motion Picture and Television Engineers (SMPTE) has defined an interface between an MPEG-2 transport multiplexer and an 8-VSB DTV modulator.

Dubbed SMPTE 310M, this synchronous serial interface (SSI) runs at exactly the bit rate of the DTV channel—19,392,658.46 Hz. The master clock in the multiplexer that drives the SSI must be within 2.8 ppm in its accuracy because this clock figures directly into the 8-VSB modulation process.

Although a DTV station may contain multiple multiplexers, only the final multiplexer, called the emission multiplexer, drives the modulator. Therefore, only the emission multiplexer generally contains a 310M output port because 310M isn't used as a generic data link.

310M was developed to address a narrow communications application—conveying the final broadcast MPEG-2 transport stream to the modulator. As such, it is a relatively expensive data link because of the tight tolerances of its transmission clock.

The 310M data link consists of a single coaxial cable with data that is biphasic mark encoded to enable recovery of the bit clock at the modulator. Biphasic mark encoding is a scheme whereby the state of a bit is determined by the number of master clock transitions on the link.

In Figure 1, observe that the 310M master clock is twice the bit rate—38,785,316.92 Hz. Whenever a binary 1 is transmitted, the encoded output toggles at half the master clock rate. Whenever a binary 0 is transmitted, the encoded output toggles at one quarter the master clock rate. (In contrast, biphasic space encoding has the opposite toggling relationship.)

In such a system, the downstream receiver (modulator, in this context) is able to recover the transmit clock with a PLL because the data link is guaranteed to contain regular state transitions, regardless of the binary data being sent across.

QUALITY VS. BANDWIDTH

Each broadcaster is given the same 19.39 Mbps of bandwidth. So, how much of that should be used for transmitting a single program? What if the broadcaster wishes to take advantage of the much-discussed multichannel capabilities of DTV (i.e., multicasting)?

The question of MPEG-2 bit rate versus quality is as religious as it is technical. Video quality is a highly subjective metric and you could get five different opinions if you asked four people!

The quality of an MPEG-2 encoded program depends on three factors—the complexity of the video, the compressed bandwidth allocated to the MPEG-2 encoder, and the capabilities of the specific MPEG-2 encoder.

MPEG-2 exploits the inherent temporal (frame to frame) and spatial (pixel to pixel) redundancy in a video signal to gain the tremendous compression ratios it provides. Generally, successive frames of video contain similar content, and within frames, neighboring pixels follow similar shading.

Consider a scene of a person walking down a street. Successive frames have a gradually shifting background, a person moving, and perhaps a car passing by. Between each frame, the unique content doesn't change much.

In this same scene, the asphalt

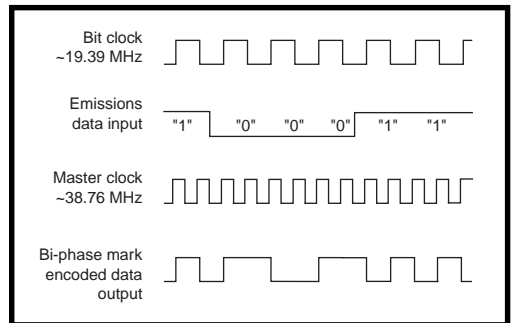


Figure 1—Bi-phase encoding allows both clock and data information to be sent over the same piece of wire. A receiver is able to recover the clock using a PLL because the bi-phase encoded signal contains regular transitions that serve as a PLL reference. Data is extracted by observing the number of transitions on the wire over a period of two master clock cycles.

street is a large patch of similar colors, as are the person's flesh tones. All of this redundant information is compressed according to the MPEG-2 algorithm.

Clearly, video content that contains a lot of detail reduces the effectiveness of spatial redundancy. Likewise, content with rapid motion reduces the temporal redundancy.

That's why it's easier to compress the average talk show, compared to the average action-packed basketball game. Talk shows contain soft detail and low motion, whereas sporting events contain higher detail (crowds, wide shots of many players, etc.) and fast motion.

An MPEG-2 encoder does not compress the incoming video into an arbitrarily small bandwidth. It is assigned an output bandwidth and works to achieve this target regardless of the video complexity. So, encode quality can be varied according to how many bits are assigned to the encoder.

The final contributing factor to encode quality is the ability of the encoder itself. I guarantee you, each manufacturer of MPEG-2 encoders will have their own opinion of who provides the best quality! Here's where head-to-head competition and subjective viewing comes into play.

A savvy potential MPEG-2 encoder buyer may carry a videotape with tough scenes when visiting different manufacturers. It's then up to the subjective opinion of the buyer to look at each vendor's quality at a common bit rate to determine the winner.

BANDWIDTH RULES OF THUMB

Invariably, customers want to know at what bit rate a given digital video format will compress with good quality. They want a set of numbers, not a lot of shuffling around with vagaries.

There are rules of thumb, but they differ depending on whom you ask. And, once again, no number is carved in stone. With that disclaimer aside, we can discuss the relative demands of the various digital video formats and estimate what each would realistically require with low- and high-complexity content.

The standard definition (SD) digital formats, 480I and 480P, each have the least raw information and therefore compress into the least bandwidth. These SD formats each contain nearly 350,000 pixels per frame. A single SD program encoded at a constant bit rate requires about 1.5 Mbps for low- and perhaps 4 Mbps for high-complexity content.

High definition (HD) formats contain much greater spatial resolution and require higher compression bandwidths. 720P contains more than 900,000 pixels per frame. As such, a 720P program requires approximately 8 Mbps for low- and perhaps 15 Mbps for high-complexity content.

1080I and 1080P each contain more than 2,000,000 pixels per frame! These two detailed formats require anywhere from 10 Mbps for low- to as much as 22 Mbps for high-complexity content.

Although the HD bandwidth numbers, especially those for 1080I/P, may look high, realize that they're in the context of today's capabilities. HD compression is still a relatively new technology. Yet even with today's bit rates, there are additional tricks that increase the number of programs a broadcaster can transmit.

STATISTICAL MULTIPLEXING

When compressing a video signal at a constant bit rate, the instantaneous complexity often doesn't match well with the selected rate. At times, the content is too static and bits are wasted.

Or, if there is fast motion, quality suffers because too few bits are available. When only one program is being transmitted over a constant bit-rate interface, not much can be done to alleviate these problems.

However, imagine being able to connect multiple encoders that share a common bandwidth pool. In a scenario like this, each encoder figures out the incoming signal's complexity

and arbitrates for varying amounts of bandwidth in real time. This practice, known as statistical multiplexing, has become common in the SD encoding world over the last few years.

The idea is to exploit the statistically low probability that all members of a pool will require maximum bandwidth at any given time. For example, it is possible that during an explosion in an action drama, another program is showing a scene where two people are talking.

As is the case with most statistically based concepts, the benefit improves as the size of the pool increases. The probability of ten programs all requiring maximum bandwidth at the same time is less than that of two programs.

Digital broadcast satellite providers reap significant benefits from statistical multiplexing because they typically transmit numerous compressed SD programs over a transponder with 30+ Mbps of bandwidth. This provides them with a large pool of encoders to maximize the statistical payoff.

Figure 2 illustrates a typical statistical multiplexing architecture. Observe that each encoder contains two processing elements—an MPEG-2 compression engine and a complexity look-ahead engine. This measure is converted into a desired bit rate according to the quality targets set by the operator.

Each encoder then communicates this desired bit rate to a central bandwidth arbiter. The arbiter divides the available bandwidth pool among the encoders according to their individual requests and operator-defined priority attributes.

Meanwhile, each encoder feeds the video through an internal delay path so that it can wait for a bandwidth response from the arbiter. The bandwidth setting for each encoder arrives and is fed to the compression engines along with the delayed video.

Statistical multiplexing provides benefits for pools with as few as two or three encoders. So, a broadcaster who is trying to evaluate the number of programs that can be transmitted doesn't have to simply add up all of the maximum rule-of-thumb bit rates. Instead, depending on the material's complexity and video format, anywhere from two to four (and possibly more) encoders can be configured as a statistical multiplexing pool with favorable results.

In this manner, it's quite realistic for a TV station to offer an HD main event along with one or more SD sideshows. In the absence of a show that truly merits the resolution of HD, perhaps eight SD programs could be multicast in the 19.39-Mbps DTV channel.

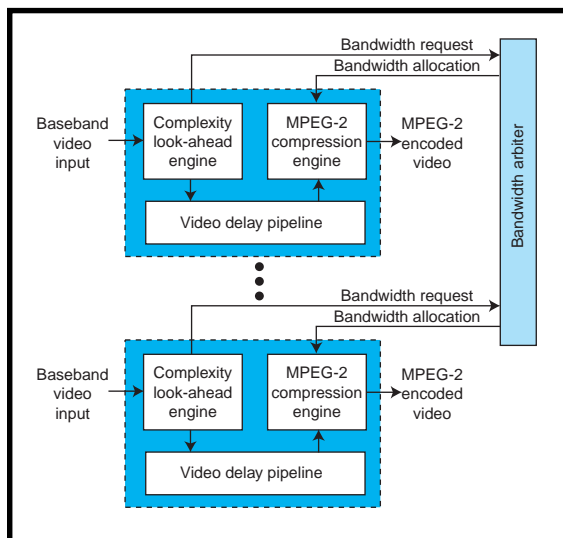


Figure 2—Multiple MPEG-2 video encoders can share a fixed pool of bandwidth through a central arbiter. Each encoder assesses the complexity of its incoming video signal and requests a certain bandwidth from the arbiter. Based on the collective requests, the arbiter divides the available bandwidth among the requestors. The encoders then adjust their compression rates to comply with the bandwidth grant. A video delay buffers the video during the time it takes the arbiter to receive, process, and distribute the bandwidth requests.

BEST VIDEO FORMAT?

As with video quality, the video-format debate often gets religious. Although some have a single preferred format, the question should be posed in terms of the type of content and picture quality you want.

The 480P SD format is an improvement over 480I, the digital version of what we've been watching all these years. At its highest frame rates, 59.94 and 60 Hz, motion is smooth and flicker is nonexistent. It compresses well into relatively low bandwidth because the spatial resolution is no greater than that of 480I.

480P is a great general-purpose format for daytime TV and news interviews, where you want a crisp picture. But, the greater spatial resolution of HD wouldn't significantly add to the viewing experience.

Although the SD formats are specified in both conventional (4:3) and wide-screen (16:9) aspect ratios, most SD material is shot in 4:3. Note that at 16:9, the effective horizontal resolution decreases because the image is

"stretched" over the same number of horizontal pixels.

720P is an interesting format because of its high frame rates (59.94 and 60 Hz), its improved spatial resolution, and its relatively modest bandwidth requirements. With nearly three times the pixels of SD and a fixed 16:9 aspect ratio, the 720P format is great for sports programming—especially when the bandwidth savings is used for multicasting alternative programming.

1080I/P has the highest spatial resolution of all DTV formats. It is also the most bandwidth-intensive format. 1080 line formats bring out the maximum detail in movies by providing more than twice the spatial resolution of 720P. Because movies are shot at 24 frames per second, the 29.97-/30-Hz frame rate does not limit the quality of motion.

The video-format debate may go on for a long time. In the end, consumers and the broadcast industry together will decide which formats are best suited to various programming.

CLOSED CAPTIONING

As with analog TV, closed captioning (CC) is an integral part of the DTV system. CC places text subtitles on screen so hearing-impaired viewers are able to follow spoken dialog. Like many other parts of the TV world, CC has gotten a facelift with the coming of DTV.

In the analog world, CC is encoded two characters at a time into line 21 of each NTSC video frame, according to the EIA-608 standard. Televisions decode the CC information and display the characters if the viewer so requests.

For several years, the FCC has mandated that all TVs sold in the U.S. must have CC support. Likewise, CC is a required part of DTV.

The vertical blanking interval of a video signal is not transmitted under DTV, so CC information is now placed into reserved data structures. A new standard, EIA-708, defines many similar CC concepts as well as more advanced screen controls. Under EIA-708, CC authors can more accurately

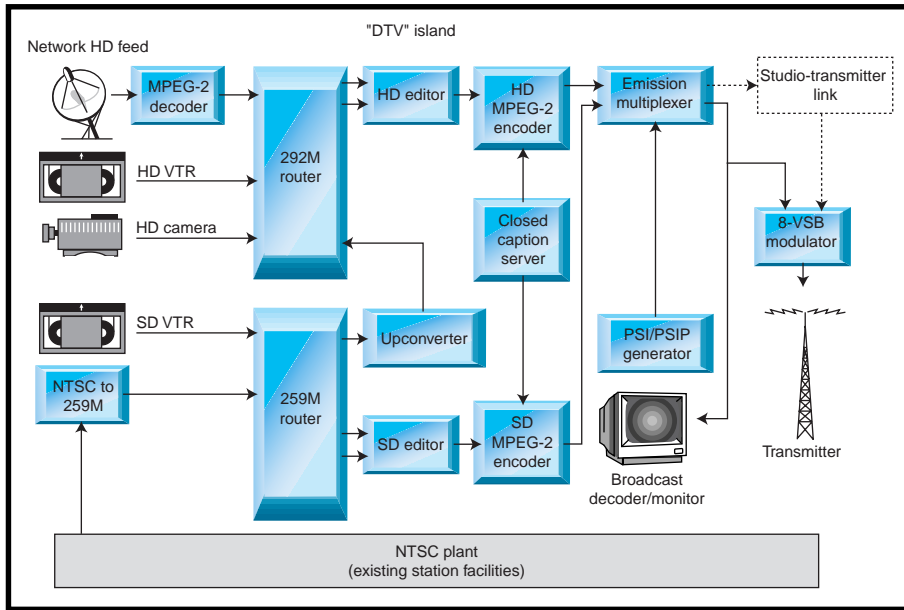


Figure 3—Many early DTV stations will contain the minimum equipment necessary to get on the air and will function as islands off to the side of the main NTSC facility. Sources range from digitized NTSC, to SD and HD videotapes, to digital network feeds. Early network feeds are compressed at high bit rates (greater than 30 Mbps), sent via satellite to local affiliates, decoded to baseband at the local station, and then re-encoded at the desired emission rate.

define the placement and appearance of CC text on the viewer's DTV.

CONDITIONAL ACCESS

Multicast DTV brings with it some interesting possibilities. Among these is new possibilities the idea of subscription-based programming.

With inherent support for conditional access (CA), DTV enables broadcasters to send some of their programs scrambled and to only distribute viewing rights to those customers who have paid a fee. The FCC has mandated that broadcasters must transmit their primary programming in the clear (i.e., without scrambling). However, once that requirement is met, the law allows for new revenue from subscription programming. Will this happen?

At this point, it is purely a business decision because the technical problems are resolved. Digital broadcast satellite and cable providers have successfully used complex CA systems to protect their services for quite some time.

In order for a broadcaster to implement CA into a new or existing DTV system, several components are necessary. First, a CA management system is needed to manage the individual viewing rights, generate

the CA datastreams, and encrypt descrambling keys.

Second, an MPEG-2 transport stream scrambler is required to apply the scrambling process to selected programs. Third, potential customers need access to a receiver with built-in security and authentication hardware (e.g., a smartcard).

CA management systems and smartcards are proprietary components, and MPEG-2 scramblers are often integrated into multiplexers and are therefore obtained from the multiplexer manufacturers. Receivers or set-top boxes are made by a variety of consumer electronics manufacturers.

Appropriate receivers already exist for satellite customers, but these boxes are unsuitable for terrestrial broadcast reception. It may take a while before security features like smartcards make their way into terrestrial receivers.

DATACASTING

DTV is all about data. Most of the data that will be present in early broadcast installations will be MPEG-2-compressed programs and the MPEG-2 and Advanced Television Systems Committee data structures that enable reception and decoding of the transport stream.

Also, some portion of the 19.39-Mbps transmission bandwidth may be allocated for generic data, which may or may not be related to a given program. Datacasting is the term used to describe the broadcast of generic data. Data is grouped into three classifications—unsynchronized, opportunistic, and synchronized.

Unsynchronized data is the easiest to implement. It can be constant bit-rate data where a fixed bandwidth is allocated and data is simply placed into MPEG-2 transport stream packets and transmitted. It can also be variable bit-rate data where the data source is incorporated into a statistical multiplexing pool.

The only unique requirement placed on the system is for a data encapsulation engine that takes generic data from an external source and inserts it into the payload of MPEG-2 transport stream packets. There are many multiplexer products that can do this today.

Opportunistic data seeks to occupy unused bandwidth in an otherwise fully allocated transport stream. A real-world encoding system periodically emits null packets, wasting valuable bandwidth. An opportunistic data inserter looks for these packets and replaces them with useful data packets. Such systems require multiplexers smart enough to perform the packet-substitution process and that can either cache data locally or communicate with an external data server.

Delivery is not guaranteed with opportunistic data. Transmission is based solely on the presence of null packets. Also, given the continuing efficiency improvements of MPEG-2 encoders, the quantity of null packets emitted continues to drop as more sophisticated algorithms are employed to maximize video quality.

However, if a broadcaster decides it's worth the expense of implementing an opportunistic data system to gain a small bandwidth advantage, the pieces are falling into place to allow it. Some multiplexer vendors already support internal opportunistic data insertion whereby the multiplexer also serves as the data server.

For implementing opportunistic data with an external data server, SMPTE and the ATSC collaborated on a standard dubbed SMPTE 325M. This standard, an outgrowth of the ATSC's Data Implementation Working Group, defines an opportunistic data flow control protocol that allows the multiplexer and data server to properly coordinate their activities.

The protocol consists of simple data request commands embedded within MPEG-2 transport stream packets. Carriage of this protocol and opportunistic data is defined for three data links: Ethernet, DVB-ASI, and SMPTE 305M.

DVB-ASI is a 270-Mbps serial interface over a coaxial cable and is the common medium for carrying MPEG-2 transport streams within studio facilities. SMPTE 305M, also referred to as SDTI (serial data transmission interface), is a 270/360-Mbps serial interface that encapsulates data within an SD video structure over a coaxial cable.

The most complex data type—synchronized data—has interesting potential applications. Basically, generic data objects (or continuous datastreams) are sent along with and synchronized to a program's video stream.

A given data object is marked with a timestamp that tells the receiver when to display that object relative to the video. The data object might be a graphical overlay that pops onscreen exactly when a corresponding event occurs in the video. Or, it could be an audible sound effect or an action such as spawning a local application on the intelligent set-top box.

Synchronized data affects almost every element in DTV production and distribution. The data itself must be created and then closely timed with the video program at a postproduction (authoring) console.

Then, the video, audio, and synchronized datastreams that compose the program must be recorded in a way that preserves the timing relationships. When handed over to a broadcaster, the streams may need to be routed around the studio, perhaps separately, without losing timing.

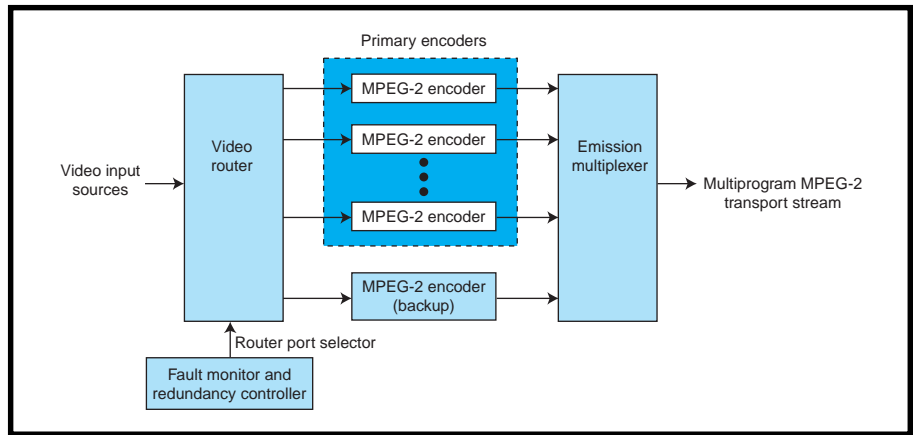


Figure 4—Fault tolerance across certain DTV equipment can be practically achieved without installing spares for every item. Encoders, for example, can be spared “1 for N” whereby a single backup encoder sits ready to handle the video signal for any other encoder that fails. In such a circumstance, the fault-recovery procedure would require routing the particular video signal to the backup encoder and changing the multiplexer’s input configuration from the failed encoder to the backup.

Near the time of transmission, these streams are passed through encoders for compression, and the relative timing of the streams must be preserved here as well.

Finally, the receiver needs a standard way to handle the incoming generic data objects. This handling includes not only decoding and displaying the object at the correct time but also buffer management to ensure that the object is ready in memory at the appointed time.

Standard interfaces, equipment, and practices to implement end-to-end synchronized data implementations do not yet exist, but groups within ATSC and SMPTE are working to fill in these gaps. Once the end-to-end timelines, data interface formats, and time-triggering methods are fully developed, manufacturers can add synchronized data support to their products.

STATION ARCHITECTURE

All of the equipment and datastreams I’ve discussed so far can be arrayed in myriad combinations to produce a DTV broadcast with unique offerings. In the early stages of the transition period from NTSC to DTV, broadcasters will deploy relatively limited infrastructures as they learn the day-to-day operational realities of DTV.

These small-scale installations will feature a mix of mostly upconverted SD content with nuggets of native HD

programming to highlight special events and popular shows. As the population accepts DTV and more equipment becomes available, the mix will begin to take advantage of DTV’s power and flexibility.

LEARNING THE ROPES

In an environment of limited native HD content, most broadcasters realistically have very little exciting material to transmit. A lot of the material will come from their national network as a live HD feed, with the balance being composed mostly of prerecorded HD material.

The more aggressive and motivated stations will purchase HD cameras and shoot some of their own programming in HD. This is no small undertaking because such HD equipment is fairly expensive in the beginning.

For the remainder of the day and in the absence of additional true HD material, the station will take SD programming—very often their main NTSC signal—and upconvert it to an HD format.

Unlike with NTSC, broadcasters now have several format choices. It’s important for each broadcaster to choose a “house format” because of the expense incurred when converting back and forth between different formats at this early time in the industry. Most broadcasters will choose the house format of their affiliated network for consistency. As of press time, CBS and NBC have chosen

1080I, ABC has chosen 720P, and Fox has chosen 480P.

Figure 3 illustrates a typical early DTV station architecture. Note that the layout consists of a DTV island alongside the main bread-winning NTSC plant.

On the left side of the DTV island are the video feeds: MPEG-2 compressed HD network feed (usually via digital satellite link), local HD video tape recorder (VTR), local SD digital VTR, and local NTSC analog feed. Intrepid broadcasters will have one or more HD cameras as well.

The NTSC feed is passed through an A/D formatter where it is converted to 480I over the SMPTE 259M serial digital interface (SDI). Remember, in a DTV plant, video is distributed digitally, not as analog.

The network's HD source is fed into a studio-quality MPEG-2 decoder where it is uncompressed to baseband—SMPTE 292M SDI. This step is performed for a number of reasons.

First, for the best quality, the network distributes their HD programming at a high bit rate. Second, baseband video can be more easily edited, stored, and manipulated with today's technology. Third, the local station can visually monitor the network feed.

The baseband network feed and the HD VTR's output is fed into a 292M router for easy signal distribution to other pieces of equipment. HD cameras, if present, would also feed into the 292M router.

A 259M router distributes the digital SD sources—the VTR and the converted NTSC feed. One of the SD router's outputs drives an upconverter that converts the 480I digital signal into that particular station's HD house format.

The upconverter's 292M output feeds back into the 292M router. Two of the 259M router's outputs feed a simple real-time digital SD editing platform that enables a director to perform baseband effects (e.g., splices and fades). The editor's output drives an SD MPEG-2 encoder, providing the compressed SD signal source as part of the DTV multicast.

The output side of the 292M router is configured similarly. Two outputs feed a simple real-time HD editing platform for baseband effects. An HD MPEG-2 encoder then compresses this broadcast-ready HD signal that comprises the real "wow" of the DTV broadcast.

CC servers provide data to both HD and SD encoders. However, many SD sources already have caption data embedded in the vertical blanking interval, so this may not be necessary for the SD signal. SMPTE is still working to embed captions within the HD signal, so an external caption server for HD is required early on.

The last and vital component of the DTV multiplex is the PSI and PSIP data structures that are necessary to adequately describe the separate digital datastreams. A dedicated PSI/PSIP generator takes configuration information from station personnel and emits the properly formatted streams to the emission multiplexer.

All of the MPEG-2 transport streams to be broadcast (in this case, the output of the HD and SD encoders and PSI/PSIP packets) are fed into the emission multiplexer. The emission multiplexer combines these separate streams and produces a single broadcast-ready transport stream that is 19.39 Mbps in bandwidth.

Depending on the station, the 8-VSB modulator and transmitter may or may not be in the same general vicinity as the other equipment. If the modulator is nearby, it is driven directly by the emission multiplexer via the SMPTE 310M SSI.

If the modulator is at a different location, a studio-transmitter link is driven by the multiplexer, perhaps 310M, which then drives the remotely located modulator. Lastly, the modulator excites a transmitter that broadcasts the DTV multiplex.

GETTING FANCY

That's what a simple DTV station might look like, but much, much more can be done with the other pieces of equipment. The broadcaster may try to attract viewers by offering more programs in the multicast. As I mentioned, this could be done by

statistically multiplexing additional SD encoders with those already in service.

More flexible editing and MPEG stream manipulation hardware will soon become available. One immediate benefit of a so-called transcoder would be to translate the network HD feed directly into a lower bit rate suitable for broadcast without the need for a full decode and re-encode step. Such a process would result in a higher quality picture.

MPEG-domain splicing offers a quality improvement effect similar to transcoding by allowing programming to be stored and edited in a compressed form. It also makes ad insertion easier. Format conversion enables easier sharing of material between stations that use different house formats.

Easy MPEG-domain editing will allow the proliferation of MPEG video servers. Instead of libraries of tapes storing uncompressed video, video servers with huge hard-drive arrays will store large selections of programming compressed at high enough bit rates to maintain archival quality yet still save space.

Data-enhanced programming is another way to attract viewers. Educational programming could provide supporting interactive material, and sporting events could be supplemented by player and team statistics. Data servers would be loaded with program-related data that would be emitted at the proper times into the emission multiplexer.

With all of these complicated boxes sitting around the studio and all the detailed parameters needed to configure them, an automation system becomes more of a necessity than a luxury. The industry is grappling with this problem, and hopefully, such equipment will become available sooner rather than later. When it does, a broadcaster will be able to easily manage a complex DTV system in a truly reliable fashion.

As DTV gains popularity, broadcasters will want to sell ad space. When this happens, the station will be on the hook to guarantee a reliable broadcast. If the football game is

dropped, the ads drop and the station loses money.

So, the high reliability that broadcasters have implemented in their NTSC plants will become necessary in the DTV plants. A certain amount of redundancy with failure detection and automatic recovery systems is required to ensure that any single failure will not significantly disrupt the real-time end product for the viewer.

Redundancy is costly, but not to the point of providing complete duplicates of an entire station. Some items (e.g., MPEG-2 encoders) can be backed up on a 1-for-*n* basis whereby one encoder sits ready to provide single-fault recovery for an array of encoders.

As Figure 4 shows, the backup encoder is fed by a completely separate router output port and has its own connection into the MPEG-2 transport multiplexer. If the redundancy controller detects a failure on a primary encoder, it selects the proper video source for the backup and modifies the multiplexer's configuration to

switch in the backup and switch out the failed unit.

Video servers might also be backed up inexpensively by preloading the program onto a smaller backup server. In this manner, if the main video server experienced an on-air fault, the backup could be switched in.

ONWARD TO THE DIGITAL FUTURE

After many years of innovation and preparation, the U.S. television industry is launching into a new age. Technology has matured to the point where these high-definition programs can be economically produced and broadcast.

No one knows exactly what path the industry will take, but it is guaranteed that significant change is happening at this very moment. Whether some people begin watching DTV on their computers and use interactive data enhancements to gain new information, or others begin to enjoy theater-like programming in their homes on a daily basis, this transformation

will affect everyone differently and at different points in the future.

And from a technical point of view, DTV opens up new business opportunities across the board. Who knows? You may find yourself drawn behind the scenes, too! ☒

Mark Balch is a senior hardware design engineer and has participated in a variety of MPEG-2 product designs, including an HDTV MPEG-2 encoder. Mark actively attends meetings of the ATSC and SMPTE industry standards groups. You may reach him at mark_balch@hotmail.com.

REFERENCES

- SMPTE 310M, *Synchronous Serial Interface for MPEG-2 Digital Transport Stream*, 1998.
- SMPTE 325M, *Opportunistic Data Broadcast Flow Control*, 1999.
- www.atsc.org
- www.fcc.gov
- www.mpeg.org
- www.smpte.org

Speed Racer

FROM THE BENCH

Jeff Bachiochi

Virtual Speed with the SX



In the hustle of modern life, speed is essential. Jeff knows this. He thinks there is no time like the present to cover the inner workings of today's fastest microcontrollers.



We are driven by speed. We expect to get what we want right now. Give me food now—instant breakfast, fast food lunch, and pizza delivery for supper. Give me weather now—plan the whole week based on a 10-s forecast. Give me phone service now—call from anywhere, anytime, but don't put me on hold. Give me Internet now—bring up this site now (what good is a 500-MHz processor if it still takes thirty seconds to download site data). Get me there now—raise the speed limit, use the drive-through, and pay at the pump. Road rage is spreading because we hate to wait. If we can't keep pace, it all starts to crumble.

Maybe we should just let it crumble a bit. Just enough to let you feel like you're in charge. When you do, the food will taste better, the weather will seem more predictable, and you might even figure out what day of the week it is before Friday comes!

If you work with processors, you know that execution speeds continue to increase. Some of the latest technologies are now using internal PLLs to create execution clocks faster than the crystals that run them. One advantage to this internal speed, beyond the obvious, is less EMI,

which is a serious threat to any product's acceptance. A disadvantage of PLL clocks is jitter (edge accuracy). Whatever the potential disadvantage, programmers usually agree, more speed is better.

TALKING 'BOUT MY GENERATION

Do today's speeds lend themselves to the generation of waveforms? Certainly, the creation of a TTL output square wave isn't a big deal. Merely setting and clearing an output bit over and over again doesn't take a lot of overhead. Let's use 1 μ s as a typical instruction cycle. This setting and clearing would result in a 500-kHz square wave. There will be a glitch in the timing as you try to jump back to the loop unless you filled the micro's code memory with set, clear, set, clear allowing the eventual wrap to automatically bring you back to the beginning.

All of this assumes that the processor did not require any kind of initialization code. So as it stands, the shortest loop would be a bit slower than just set, clear, set, clear. It is necessary to introduce a slight delay between the set and the clear. The delay should have the same number of cycles as the jump instruction needs after the clear.

- Start—set bit, 1 instruction cycle
- Nops—2 instruction cycles
- Clear bit—1 instruction cycle
- Jump start—2 instruction cycles

The number of nops depends on the number of instructions cycles in the jump command to keep the square wave symmetrical (50% duty cycle). This number can vary from two to many, especially if the instructions are pipelined. The best case is six instruction cycles per loop (166-kHz square wave).

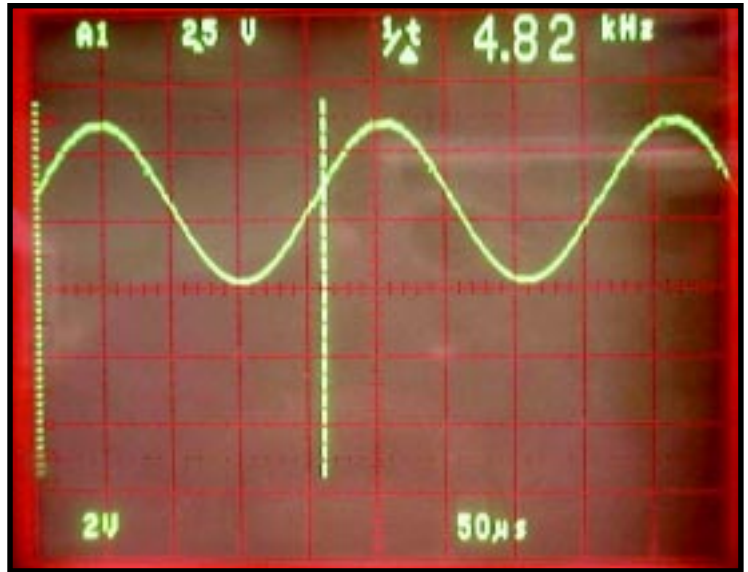
Now, let's suppose that a single frequency output isn't good enough. You want to have the ability to change the frequency. You might choose to use a number of input bits to select the frequency. Reading a byte value from an input port certainly is quick, and it gives you 256 different frequencies.

To place variable delays between the set and clear commands you might choose to place 256 nops between them and use a jump+offset command to jump to specific points within the nop delays. The byte read from the input port could provide the offset of the jump. So, you add the port read command, which adds another instruction cycle to the loop (actually it adds double because you need to balance the added instructions on both half cycles of the waveform). The loop is now a minimum of eight instructions long, for a maximum frequency of 125 kHz.

TIME IS ON MY SIDE

Using the processor's timer function is a much better idea. This method will also allow for longer delays if it has a 16-bit timer available. However, an 8-bit timer can also be used. Timers are normally incremented by the instruction clock. The timer generates an interrupt when the time increments (or decrements) past its maximum or minimum count and overflow (or underflow) of the counter. If the timer is allowed to count without the user altering the count value, it will interrupt again in 256 counts (or 65536 counts for 16-bit counters).

Photo 1—This is the output produced by the SX processor. On close inspection, you can see the actual output steps produced by the R2R ladder. This DAC was constructed with 5% resistors.



The time between interrupts can be adjusted by altering the counter value anytime after an interrupt yet before the next interrupt. In fact, careful attention must be paid to updating the count, especially with 16-bit counters, as the count may be incorrectly incremented if the low byte of the count overflows into the high byte before the high byte is updated.

Of course, the timer can be used without interrupts. Some processors don't have interrupts. Without interrupts you are required to stay in a loop waiting for the overflow to occur.

Although using the timer can be easier than jump+offset, without interrupts it's a pain.

The advantage of using interrupts is that the set and reset commands can be part of the interrupt background routine allowing other things to take place in the foreground, but interruptable (at a lesser priority). An important pitfall to avoid when using the timer is that any count placed into the timer must not overflow before the interrupt routine exits. Overflows, which take place during a timer interrupt, will cause another interrupt immediately upon exiting. In this case not only is the timing wrong (late), but no other code will ever be executed except for the interrupt routine.

There are several ways of preventing this from happening. If a prescaler is available, you can choose a prescale divisor, which will only send the timer an increment every other instruction cycle (or other multiple thereof). This enables a few more instructions to execute before incrementing the timer and essentially slows down the timer, creating a longer tick. The longer the tick, the greater the difference in minimum frequency change.

Alternately, the timer can be turned off while the count is loaded and reenabled just before the reti. This method adds a few instruction cycles to the delay, but it keeps the timing accurate. Still, the minimum timing (maximum frequency) is the

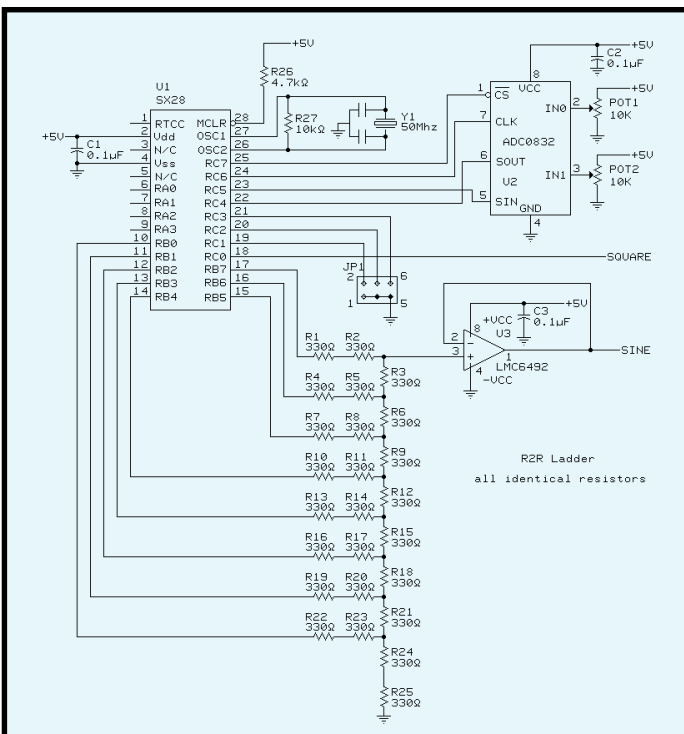


Figure 1—The R2R ladder is a programmable current source into R. V_{out} can vary from 0 volts to $V_{cc}/2N$ where N is the number of output bits.

maximum code path through the interrupt. In this case, save registers turn off the timer, complement the output bit, reload the timer, turn on the timer, restore registers, and exit. This might be as little as 17 instructions/half cycle (~30 kHz).

You might be getting the picture about now. It doesn't take long before the steps necessary to perform a function begin to detract from the original purpose (to program a variable high-frequency square wave oscillator).

Imagine now that we also want other things like PWM on the square wave output or alternately, a sine or triangle output. If we choose to create a sine wave using 256 discrete steps, that would (at the absolute minimum) limit the output frequency to 111 Hz. If we could find a micro with a faster instruction cycle certainly that would help things, right?

BIT-BALL WIZARD

I received my first samples of Scenix Semiconductor's first 50-MHz microcontroller in early 1998. Their first flash memory-based micros were poised to pounce on Microchip's low-end micros. Flash memory makes for quick development. Their idea at Scenix is to keep the price lower by eliminating all the on-chip hardware peripherals.

Scenix figures if they give you enough speed, you can create virtual peripherals on an as-needed basis.

Listing 1—In the timer overflow interrupt routine, the value of A/D channel #1 determines how many interrupts must occur before the square wave output bit is flipped.

```

TMROVF      org      $000
TMRO_SQR    mov      RTCC,#$E0      ; (2)
            test     POT1           ; (1)
            jz       TMRO_SQR0     ; (2,4 skip)
            dec      POT1           ; (1)
            jmp      TMRO_SQRX     ; (3)
TMRO_SQR0   mov      POT1,CNT1     ; (2)
            inc      CNT            ; (1)
            movb    CNT.0,RC.0     ; (4)
TMRO_SQRX   reti                    ; (3)
    
```

These micros do include an 8-bit timer with an interrupt. Additionally, interrupts have a hardware context save/restore to automatically take care of some necessary housecleaning. There is no timer overflow flag to poll, so if you don't use the interrupt service routine, you must continuously read the timer to determine if it has rolled over.

Because the SX's timer can't be stopped and started, turning off the timer isn't an option. Therefore, in order to prevent reload values from timing out before the routine has exited, the reload value must be greater than the longest path through the routine.

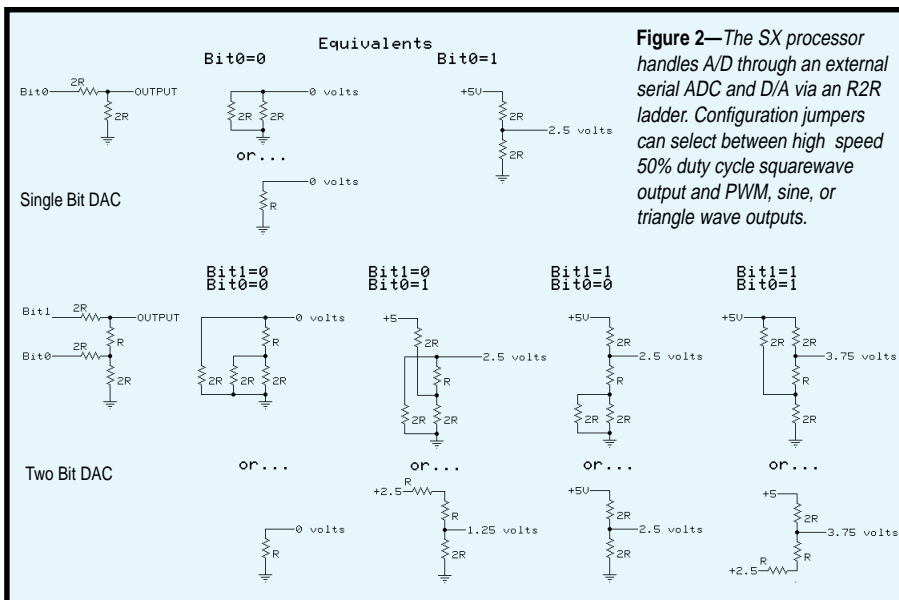
In Listing 1, we are trying to get our square wave output to be as fast as possible, yet still be able to vary its frequency.

The longest path through this interrupt routine requires 20 in-

struction cycles. The timer must be loaded with a value, which will not cause a rollover until after the interrupt exits. If you want any real work done before the next interrupt comes along, you must adjust this value appropriately. Doubling the 20-instruction count to 40 would balance the execution to 50% interrupt, 50% other work. Adjusting the reload value sets the tick time (minimum duty cycle) of the output waveform. This is the highest possible frequency, so it makes sense to keep this tick as fast as possible. Here I use a value of 32 (eight cycles have passed by the time I get to reload the count).

Producing the output waveform is handled by the interrupt routine, so what else is left? If we wish the output frequency to be variable, we need to collect a control value from somewhere. Although an 8-input port allows for 256 possibilities, I wanted to use a potentiometer to make tuning easier than plugging and unplugging jumpers. I decided to add a simple serial 8-bit A/D (see Figure 1), because producing a virtual A/D really requires use of the RTCC (or at least no interrupting) and an external capacitor to charge and discharge.

A small 8-pin dual-channel A/D is available from a number of manufacturers (e.g., National's ADC0832). The nice thing about using these serial devices is that the clocking is asynchronous. The main loop, which reads the A/D, can be interrupted by the RTCC interrupt without causing problems in getting the A/D's converted value (see Listing 2). The 8-bit value read is used as the reload value of the tick counter (POT1).



Listing 2—The second A/D channel adds a “fine” adjustment to extend the half cycle count to a 16-bit value.

```

org          $000
TMROVF      mov     RTCC,#$E0      ; (2)
TMRO_SQR    test    POT1           ; (1)
            jz     TMRO_SQR       ; (2,4 skip)
            dec    POT1           ; (1)
            jmp    TMRO_SQRX      ; (3)
TMRO_SQR    test    POT2           ; (1)
            jz     TMRO_SQR0      ; (2,4 skip)
            dec    POT1           ; (1)
            dec    POT2           ; (1)
            jmp    TMRO_SQRX      ; (3)
TMRO_SQR0   mov     POT1,CNT1      ; (2)
            mov     POT1,CNT2      ; (2)
            inc     CNT            ; (1)
            movb   RC.0,CNT.0     ; (4)
TMRO_SQRX   reti
            ; (3)

```

I can add a second pot because I have two A/D inputs. This little change adds a second A/D channel to increase the tick counter value from an 8-bit to a 16-bit value. It adds eight execution cycles to the interrupt loop, but because I didn't readjust the reload value, all the timing remains the same. This 16-bit count creates a 50% square wave variable from 625 kHz down to 10 Hz. Ultimately this maximum speed is reached, thanks to the 50-MHz clock and the 20-ns execution cycle of the SX micro.

The next step brings us to PWM square waves. We've already got a couple of 8-bit inputs to the processor, so let's set up one as frequency and

one as duty cycle. Right off the bat, the maximum frequency is going to go down by a factor of 256 (the control range of the duty cycle parameter). So the most we could hope for is about 2500 Hz. The math necessary to produce a value for the duty cycle on or off time based on the total cycle time is eliminated with this scheme.

The 8-bit frequency value (POT1) is used as a loop-count value. Every time the interrupt loop is entered this value is decremented until it reaches zero. This produces the variable-frequency portion of the output. However, once it reaches zero, an 8-bit counter (CNT) is incremented. Whenever this counter reaches zero, the square wave

output bit is cleared. Although this reduces the frequency by 256, it does divide the selected frequency set by POT1 into 256 equal pieces.

Because the duty cycle value (POT2) is also 0–255, this value can be used to directly control duty cycle without any math. The off time is this value in relation to the counter (CNT). Remember that the output bit is cleared when $CNT = 0$? In the same way, the output bit is set when the duty cycle decrements to zero. So, the output bit goes low once every 256 counts (CNT) and goes high at one of those counts, when POT2 decrements to zero. Again, most of the real work is done in the interrupt routine (see Listing 3).

The maximum number of execution cycles through the interrupt loop is now 29. We still don't have to increase the minimum interrupt time (RTCC reload value). Because our operation is performed on each half cycle of 625-kHz/256 frequency, the maximum output frequency is ~5 kHz and not 2.5 kHz as suggested previously. Duty cycle is frequency independent and adjustable in <0.5% increments.

You might wish to trade off duty cycle increment size for maximum frequency. Reducing the duty cycle resolution to 128 allows the maximum frequency to go up by a factor of two resulting in ~10 kHz. I chose to use 256 because it greatly simplifies the programming, and it leads into the next section quite nicely.

SURFIN' USA

Certainly there must be life beyond square waves. In fact, most signal generators provide sine and triangle outputs in addition to square waves. One of the easiest ways of producing sinewaves is to provide a PWM output from which you could filter out most of the harmonics, leaving a clean (as clean as they get) sinewave. I don't want to use this method, because it doesn't lend itself well to a varying frequency output.

The more direct approach is to use a DAC. The cycles necessary for writing (rapidly enough) to a serial DAC would greatly reduce the

Listing 3—In this PWM implementation, the A/D channel #2 value becomes a count of how long the output will remain low for each cycle.

```

org          $000
TMROVF      mov     RTCC,#$E0      ; (2)
            jnb    PC.CFG,TMRO_SQR ; (2,4 jump)
TMRO_PWM    test    POT1           ; (1)
            jz     TMRO_PWM0      ; (2,4 jump)
            dec    POT1           ; (1)
            jmp    TMRO_PWMX      ; (3)
TMRO_PWM0   inc     CNT            ; (1)
            test   POT2           ; (1)
            jz     TMRO_PWM00     ; (2,4 jump)
            dec    POT2           ; (1)
            jmp    TMRO_PWM000    ; (3)
TMRO_PWM00  mov     POT2,CNT2      ; (2)
            setb   RC.0           ; (1)
TMRO_PWM000 test    CNT            ; (1)
            jnz   TMRO_PWM0000    ; (2,4 jump)
            clrb  RC.0           ; (1)
            mov   POT2,CNT2      ; (2)
TMRO_PWM000 mov     POT1,CNT1      ; (2)
TMRO_PWMX   reti
            ; (3)

```

maximum frequency available here even more than the 5 kHz we presently have. So, I don't want to use this method either.

A third possibility uses an R2R ladder as a parallel DAC by writing 8-bit values directly to an output port. Using either DAC method requires some computations or a lookup table for determining the appropriate output values. As if you haven't figured it out already, notice that I've been paving the way for a 256-byte lookup table by the way I've designed the PWM square wave interrupt routine.

The CNT variable which was used to divide the square wave into its 256 possible duty cycle points can now become the offset into a lookup table. Instead of getting the off-time from POT2 for the PWM square wave output, in this method POT2 is not needed, since the table will supply all of the necessary wave shape data. The value returned from the table is simply placed into Port B's output register. Where did this data come from?

Attempting to calculate the necessary sine data on the fly at each new degree point within the 360° of a cycle would again be too time consuming. I wrote a few QBASIC program lines on my PC to calculate the hex values for each table entry. Although I could have had this program create a file properly formatted for direct copying into my source code, I opted for just a simple program to print a paper list of the table offset and entry data.

I entered this data by hand into the source code. However, I did need to make a small adjustment. Using Parallax's SK-KEY development system I noticed that the `jmp PC+W` command is actually `jmp PC+W+1` and it can jump to the first 256 bytes of each 512-byte page. Because the actual jump command must reside in the same 256 bytes, when `W = $FF`, the jump will actually end up back on top of itself (because it can't roll into the upper 256 bytes). To prevent this from playing havoc with program execution, I test for `W=$FF` and force it to `W=$00`. This means that the offset

data at \$00 will in fact be used twice for each cycle.

The data presented to Port B is the same data that would be passed to an 8-bit hardware DAC. I create a DAC by using an R2R ladder on these output pins and following the ladder with an OP-AMP to buffer the output signal. For more detail, see the analysis of how the R2R ladder works in Figure 2. Each bit of the R2R ladder creates a voltage divider capable of controlling the voltage difference between its output pin and the next lower bit's output.

The higher the number of bits in the resistor DAC, the closer to V_{CC} the output can reach. With a single bit the R2R output switches between $\frac{1}{2} V_{CC}$ and ground. With two bits, the R2R output can reach $\frac{3}{4} V_{CC}$ with all output bits high. At 8-bits the R2R output will reach 99.6% of V_{CC} with an LSB of 19.5 mV (sound familiar?) The actual output impedance will always equal R. Take a look at the actual output waveform produced by the SX chip in Photo 1.

From this you can see that adding other special waveforms is no more difficult than pointing to a different table. The number of possible waveforms is limited only by the amount of space available for the tables. The total space needed for the code is less than 256 bytes (not counting any tables). You might find that you could develop algorithms, which require less table space.

You can do sine waves of the same resolution with a table of only 64 bytes. However, figuring out which quadrant you're in and in which direction you would need to read through the table, would seriously reduce the maximum frequency. Often simpler is better, since you'd be trading fanciness for crucial time. Note that many compilers can be optimized for either maximum speed or minimum code size.

SLOW RIDE

When I look back at what I started with (a 50-MHz processor with a 20-ns execution time), I can't help but feel that there is something basically wrong here. Although I could get

some fairly fast square wave outputs (>600 kHz), when it comes to generating some of the other useful basic waveforms, speed deteriorates quickly. There is a 10,000:1 difference between the execution cycle time and the maximum sine wave frequency output (based on my resolution criteria).

It's true I haven't yet delved too deeply into cutting every corner and tightening up the code to its absolute minimum execution times. I thought it would be better to work through this based on clarity as opposed to absolute maximum attainable speed. Besides, no matter what I come up with, I'm sure there are many of you out there who will program cycles around me. You don't see many waveform generators based on a microcontroller, and I guess this is for several good reasons. Achieving complex variable high-frequency waveforms takes precious cycle time.

It's not very often that the ideas I investigate actually end up as a piece of useful equipment. But I seem to always have the need for a simple waveform generator. After all, 10,000:1 sounds pretty good when compared to the odds of winning the lottery. ☒

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar's engineering staff. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.

SOURCES

SX-28

Scenix Semiconductor, Inc.
(408) 327-8888
Fax: (408) 327-8880
www.scenix.com

SX-KEY

Parallax, Inc.
(916) 624-8003/8333
Fax: (916) 624-8003
www.parallaxinc.com

ADC0832, LMC6492

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

SILICON UPDATE

Tom Cantrell

Atmel Gets Tiny



Today's econo-MCUs are stripped

down to the smallest size possible and yet loaded with enough features to satisfy plenty of applications. Not to be left out, Atmel introduces the ATtiny15 to the mix.



Small is beautiful. That's the

name of the game in the 8-bit microcontroller biz these days. You need look no further than Microchip—they've ridden the quaint little '70s-era PIC from obscurity to number two in the 8-bit MCU market, behind only perennial leader Motorola who move even more massive quantities of minimalist '05s.

What's the big deal with these small and cheap micros anyway? It's the silicon version of Econ101: cut the price of a chip and more (many more, as in hundreds of millions of units) will get sold.

Some of the volume boost is simply a result of pass-through price savings as existing gadgets are redesigned to cut their BOM. More profound, it seems there are certain inflection points (e.g., under a buck) at which a

bunch of completely new applications emerge, spiking demand upward.

The latest generation of econo-MCUs, by their stripped-down nature, have much in common (small memory arrays, tiny packages, miserly power consumption, and low-fat peripherals) to cut application costs to the bone. Dig a bit deeper and you'll find small, but perhaps critical for your application, differences.

Atmel has hopped on the downsizing bandwagon by introducing 8-pin versions of their AVR microcontroller. Those of you not familiar with the AVR should check out my column in *Circuit Cellar* 81, "Not Your AVRage MCU." In short, AVR is a modern 8-bit architecture (i.e., pipelined, load/store, 32 × 8 general-purpose registers) that counts on youth to give it an advantage over the existing, and somewhat long-in-the-tooth, favorites like PIC, '68, '51, Z8, and so on.

Atmel's Tiny lineup consists of '1x and '2x variants. The basic difference between the two (as shown in Figure 1) is the '1x core is limited to 1-KB program memory and no RAM (i.e., the only working storage is the 32-byte register set) and gets by with a smaller instruction set (90 versus 118 instructions) than the '2x.

The latest chip, so new it's not even on the chart in Figure 1, is the ATtiny15 shown in Figure 2. Let's take a closer look.

8-PINS, WILL TRAVEL

The ATtiny15 includes 1 KB of flash memory for code and 32 bytes of RAM, quite on par for the entry-level course. However, it also includes 64 bytes of EEPROM, with 100,000 write/erase cycle endurance. The

	Flash memory (KB)	EEPROM (Bytes)	RAM (Bytes)	Instructions	I/O pins	Interrupts	Ext. interrupts	SPI	UART	8-bit timer	16-bit timer	PWM	Watchdog timer	RTC timer	Analog comp	10-bit AD channels	On-chip oscillator	Brown-out detector	In-system programming	Vcc (V)	Clock speed	Packages
ATtiny11L	1	-	90	6	3	1 ₁	-	-	1	-	-	Y	-	Y	-	Y	-	Y ₃	2.7-5.5	0-2	8-pin DIP 8-pin SOIC	
ATtiny11	1	-	90	6	3	1 ₁	-	-	1	-	-	Y	-	Y	-	Y	-	Y ₃	4.0-5.5	0-6	8-pin DIP 8-pin SOIC	
ATtiny12V	1	64	90	6	3	1 ₁	-	-	1	-	-	Y	-	Y	-	Y ₂	Y	Y	1.8-5.5	0-1	8-pin DIP 8-pin SOIC	
ATtiny12L	1	64	90	6	3	1 ₁	-	-	1	-	-	Y	-	Y	-	Y ₂	Y	Y	2.7-5.5	0-4	8-pin DIP 8-pin SOIC	
ATtiny12	1	64	90	6	3	1 ₁	-	-	1	-	-	Y	-	Y	-	Y ₂	Y	Y	4.0-5.5	0-8	8-pin DIP 8-pin SOIC	
ATtiny22L	1	128	128	90	5	2	1	-	1	-	-	Y	-	-	-	Y	-	Y	2.7-6.0	0-4	8-pin DIP 8-pin SOIC	
ATtiny22	1	128	128	90	5	2	1	-	1	-	-	Y	-	-	-	Y	-	Y	4.0-6.0	0-8	8-pin DIP 8-pin SOIC	

1) One external interrupt + interrupt and wake-up on pin change (all I/O pins) 2) High accuracy (5%) internal RC oscillator with programmable speed 3) Requires 12V signal on RESET pin during programming

Figure 1—The line of "Tiny" 8-pin AVR micros makes Atmel a player in the small-is-beautiful MCU biz.

addition of EEPROM to the on-chip memory mix is a major advantage for applications that would otherwise require an external EEPROM chip or battery back-up scheme.

I/O-wise, with 8-pin packages all the rage, doing it right demands careful attention to precise pin assignment and peripheral integration. The idea is to serve the broadest range of applications while imposing no cost penalty for the luxury of such flexibility. Simple in theory, but quite difficult in practice.

The Tiny15 does a good job of making do with only six pins (you still need power and ground after all), as you can see in Figure 3. First, notice there are no external clock inputs. Like many recent entrants, the Tiny15 incorporates an on-chip RC oscillator, a feature that's becoming a must-have to reduce system cost, power, noise, and size. Unlike most other MCUs with on-chip oscillators, there's absolutely no option or provision for an external clock, so it's take what the Tiny15 offers (1.6 MHz) or leave it.

Now, 1.6 MHz doesn't sound like much in these days of GHz-or-bust high-end machines. Just remember the AVR pipelined RISC design delivers close to 1 MIPS per megahertz and 1-3 MIPS is exactly where the high-volume nuts-and-bolts apps are.

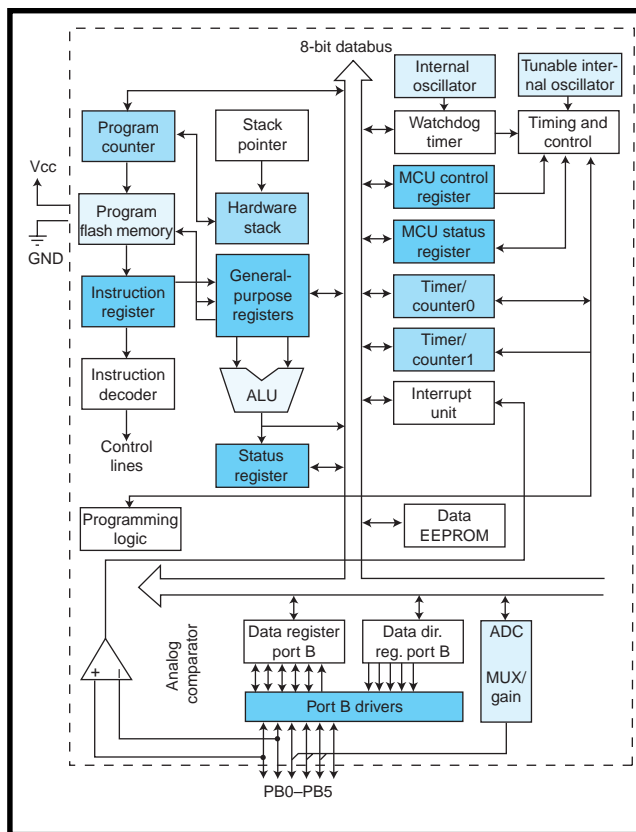


Figure 2—The ATtiny15 is the newest addition to the Tiny line, notably the first to incorporate a 10-bit A/D.

Of course, the accuracy of RC oscillators is such that voltage and temperature drift are inevitable. However, the Tiny15 has an 8-bit calibration register allowing dynamic clock adjustment in applications where the MCU can access an external timebase. Software could retune the oscillator, just as you would periodically adjust a clock running a bit fast or slow.

POWER IS EVERYTHING

“How long will the battery last?” (along with, “How big is it?” “How

much does it weigh?” and “How much does it cost?”) is what econo-MCUs are all about.

Power management and flexibility are reflected in many aspects of the Tiny15 design, starting with integration of all the functionality of an external supervisor IC including power-on-**RESET*, brown-out detection, and watchdog timer (see Figure 4). This arrangement not only cuts chip count and so on, but saves up to three pins that would otherwise go to waste. Every pin counts when there's so few to play with.

At the same time, the Tiny15 does offer **RESET* and INT pin options. Generally speaking, the integration of power-on and brown-out **RESET* on-chip would seem to eliminate the need for an external **RESET* pin. In essence, V_{CC} becomes the **RESET* pin so you could just cycle the power.

However, if buried in an otherwise non-stop application, cycling power to the MCU may be cumbersome, so the **RESET* pin option is available. In addition to **RESET*, there's plenty of external interrupt capability in the form of a specific pin option (INT0) as well as an interrupt-on-pin-change feature.

Even the INT0 pin can be part of the interrupt-on-pin-change scheme, in which case, two interrupts (INT0 and pin change) could be caused by activity on the INT0 pin. I'm not sure what such a feature might be good for, but it is interesting.

Interrupt response is four cycles minimum, which includes stacking of the PC. However, multicycle instructions (the longest being four clocks) are allowed to complete before the interrupt is taken and using an interrupt to wakeup from power down mode incurs a four-cycle delay. Typically, another couple of cycles are

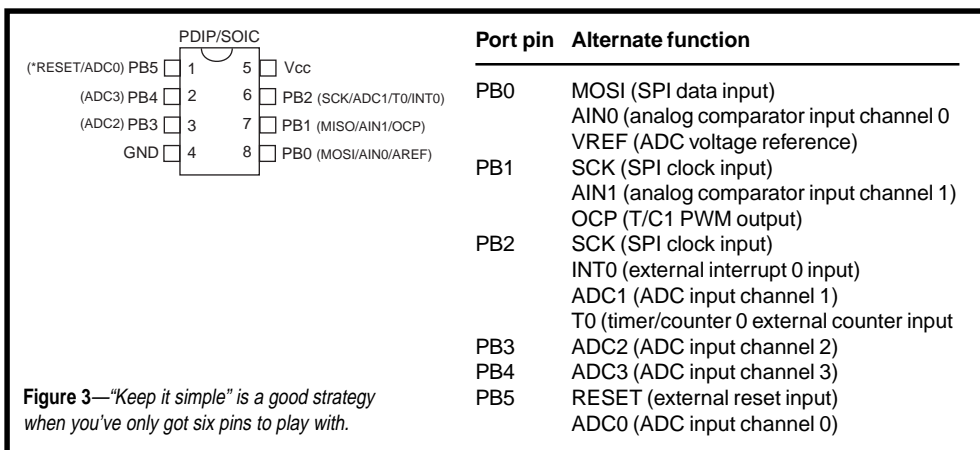


Figure 3—“Keep it simple” is a good strategy when you've only got six pins to play with.

required to execute a jump to the handler.

Tiny15 supports both high- (12 V) and low- (i.e., single supply, V_{CC} only) voltage serial programming. The latter is ideal for programming the chip *in situ* for designs that don't have 12 V built-in. If you do have 12 V handy, you're only using it to tell the chip to enter programming mode, rather than delivering programming power. Note that the *RESET pin is used to select between high- and low-voltage programming modes. If you set the fuse that disables the *RESET pin function, you won't be able to use low-voltage programming mode.

In either programming mode, communication with the chip is accomplished via a clocked serial port taking advantage of an extensive sequence of built-in commands to program and verify the flash memory, EEPROM, and various configuration fuses. Unlike some other flash-memory MCU chips, all the complicated timing is handled with features like auto-erase, self-timed write, and data polling. They make incorporating in-system programmability (ISP) into your design that much easier so you won't have to spend a lot of design time, software, or silicon to take advantage of the ISP concept.

Software access to the on-chip EEPROM is just as easy, involving little more than setting up address and data registers and setting a write-enable bit. Some 2.5 ms later, a status bit (or optional interrupt) signals the completion of the write cycle.

Actually, it's a little more complicated by virtue of a built-in EEPROM protection mechanism. There are two write-enable bits, a regular (EWE) and a main (EEMWE) counterpart. First you set EEMWE and then, within four clocks, set EWE. Should the EWE bit not be set within four clocks, the EEMWE bit will automatically time out, preventing write access.

This timeout ensures that a software crash doesn't inadvertently blitz the EEPROM. The datasheet also reminds us that non-volatile data (i.e.,

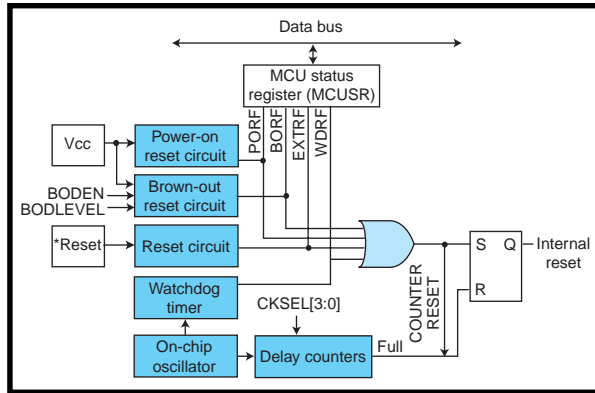


Figure 4—Tiny15 features like built-in oscillator, power-on-*RESET, brown-out detection, and watchdog timer eliminate the need for external chips and extra pins.

constants) should be stored in the program flash memory which, unlike the EEPROM, can only be written to during device programming and cannot be accessed by the CPU.

The Tiny15 features the usual idle (instruction execution stops) and power down (everything turned off) options with wakeup via a variety of internal and external *RESETs and interrupts. Although the final AC/DC characteristics aren't available as I

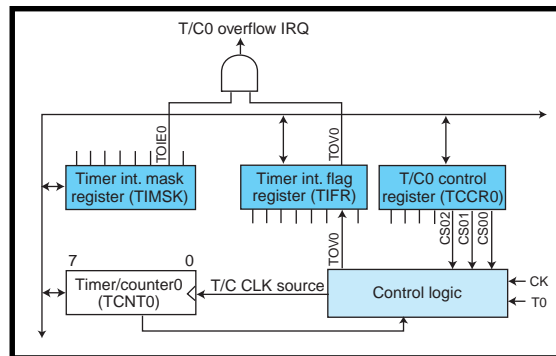
write this, based on the low clock rate and specs for other members of the Tiny family, expect power consumption to be extremely low, on the order of a few mA active and a few μ A in power down. This is especially true for the L version that extends the standard part's 4–5.5 V operating voltage range downward to 2.7 V.

TINY TIMERS

Given the pin constraint, the Tiny15 manages to integrate surprisingly powerful I/O capabilities starting with two 8-bit timers.

The first (T0, shown in Figure 5), is the same as on other members of the Tiny1x family and is a fairly basic unit that runs off a five-stage prescaler (clock/1 to clock/1024) or optional external input (T0 pin). Note that in external clocking mode, the time between transitions on the pin must be greater than 1 CPU clock cycle. Also, activity on T0 will clock the

Figure 5—Timer T0 is a rather basic unit common to many members of the Tiny family. The Tiny15 incorporates a second timer, T1, with more features as well.



counter, even if the pin is configured as an output.

The new timer (T1) is similar to T0, but somewhat more sophisticated by virtue of two major enhancements.

First, an on-chip PLL generates an up to 16× clock (i.e., 25.6 MHz at the nominal 1.6-MHz clock rate) to drive T1. This process expands the list of optional prescale divide ratios to more than a dozen between clock × 16 and clock/1024. The overclock options (×2, ×4, ×8, ×16) deliver multimegahertz resolution (e.g., <100 ns), quite a bit better than that normally expected of an econo-MCU.

Second, T1 has the extra bells and whistles (two output compare registers and an output pin (OCP)) that turn a plain timer into a PWM. Notably, it includes glitch-free operation by only effecting changes to the PWM ratio on cycle boundaries, avoiding the glitches that plague less-sophisticated designs. At 8-bit resolution, the maximum frequency is the PLL clock (25.6 MHz) divided by 256, which works out to a nice, round 100 kHz.

LET'S GET ANALOG

As their role as nerve cells between the real and digital worlds expands, look for MCUs to invest more silicon in analog functions. The Tiny15 is no exception.

Like other members of the Tiny1x family, the Tiny15 incorporates an analog comparator, which compares the voltage on two pins (AIN0 and AIN1). When the voltage on AIN0 is higher than AIN1, the comparator output is set. The comparator output can be monitored directly via status bit or specific states (comparator high, low, or toggle) can be selected to generate an interrupt.

Another comparator option substitutes an on-chip 1.22-V reference (also used by the brownout detection circuits) for one of the external inputs. If brownout detection isn't enabled (by a fuse setting), the reference voltage

must be enabled in software and given time to stabilize before accurate comparisons can be made.

Comparators are useful, and clever designers can make them do many things. A common trick would be to configure T1's PWM output as a DAC (using an R/C for smoothing) and connect it to one of the comparator inputs. Then, the PWM/DAC can be driven by software to create a poor-man's successive approximation A/D.

That's nice, but Tiny15 goes for the gusto by incorporating a dedicated 4-channel 10-bit A/D (see Figure 6). Once again, the unit is surprisingly powerful for an entry-level MCU. Features start with speedy conversion (minimum 65 μs) and both single-ended (0–Vcc) and differential (0–2.56 V) inputs are provided. The differential channel even includes a 20× gain selection that handles low-level signals without sacrificing resolution or having to add an external amp.

Single and free run (continuous) conversion modes are offered. Single conversions can take advantage of a noise-canceling feature which is basically a special variant of power down mode that quiets the rest of the chip while a conversion is in progress.

Conversion completion can be monitored by either status bit or interrupt. There's also special logic that insures that readings from both data registers (ADCL—8 least significant bits, and ADCH—2 most significant bits) belong to the same conversion. Once ADCL is read by software, the A/D converter cannot access either data register until ADCH is read. Don't forget to always read both ADCL and ADCH (in that order), especially before you go calling Atmel swearing their A/D is broken!

PRIORITY INTERRUPT

It Starts with an Idea



b

y now you're probably sick of hearing dire warnings about January 1, 2000. Don't worry, I'm not going to bore you to death with more talk on the subject. I said my piece a few months ago and believe me, I'm just as happy as you are to have this media hype over and done with.

I witnessed some absurd levels of preparedness. A friend of mine was describing the plan in his town. Their readiness got to the point of having hundreds of civil preparedness "block-watchers" throughout the community whose job was to place special colored flags on houses and at road intersections so roving police cars could find people in distress. While remotely plausible I suppose, I wonder if the kind of readiness that assumes a total failure of the infrastructure of things like telephones, cellphones, and the 911 system only served to feed the media hype on all this.

We know the truth, of course. If there were any problems, it was the result of lazy programming, not divine intervention or doomsday conspiracy. And, if systems weren't fixed before January 1, it probably had more to do with some bean-counter determining that it was cheaper to fix things that actually failed rather than performing a lot of expensive preventive maintenance.

So, now that we've made it to January 2000, what's in store at *Circuit Cellar*? Perhaps the best way to answer that is to look at what we started last year. For *Circuit Cellar*, 1999 was a banner year. When many other technical magazines were imploding, consolidating, or restructuring, *Circuit Cellar* was expanding. We've increased our circulation, doubled the editorial content we bring you each month, doubled the number of design contests we sponsor, and greatly expanded our technical coverage.

If past achievement offers a clue about doing things that will be successful in the future, it certainly has to be Internet related. Last year we started *Circuit Cellar Online* and it has become a tremendous success. Unlike many other publications, our online magazine isn't an HTML rehash of the print version. Instead, it contains 100% new editorial with many web-specific features and enhancements (and printable PDFs for the web-phobic types). I appreciate the fact that so many of you frequent it each month (which keeps the sponsors happy).

Last year was a beginning. If I have to admit to having a long-term objective, it would be to make the *Circuit Cellar* brand into a serious "dot com" resource. When I say serious, I mean that wherever you see a *Circuit Cellar*-sponsored activity on the Internet, you know it's unique or just plain better than elsewhere.

Our latest venture may seem a little like déjà vu. If you've been hanging around this technical stuff as long as I have, you might remember "Ask BYTE." It's fifteen years later, but the need for an informative technical Q&A forum is still there.

Starting this month, *Circuit Cellar* brings you the latest in question-and-answer authorities—Ask Us. Managed by Jeff Bachiochi and hosted on ChipCenter.com, Ask Us offers readers a convenient channel directly to experts who are ready to answer your most pressing technical questions. Have a nagging analog interfacing problem? Want to know how acoustically correct MP3 is before you buy those \$8000 B&W speakers? Pose your question to the Ask Us team of researchers.

Of course, all the things you like seeing at *Circuit Cellar* each month began as ideas. Many of them came from reader correspondence or from guys like Bob Perrin (*Circuit Cellar Online*—Considering the Details column). Our future depends on keeping these ideas flowing. If you've got a great inspiration or an idea for something that you'd like to see done at *Circuit Cellar*, tell me about it. I've told you all along that our magazine is a community and it's that community of ideas that keeps it all worthwhile.



steve.ciarcia@circuitcellar.com