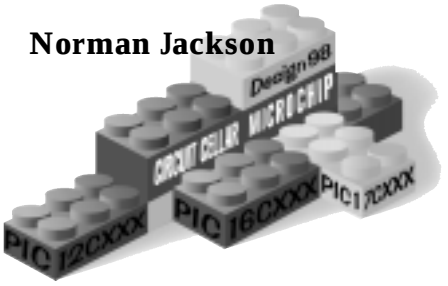
CIRCUIT CELLAR INK®
THE COMPUTER APPLICATIONS JOURNAL

FEATURE ARTICLE

BitScope

A Mixed-Signal Capture Engine

Norman Jackson



Has your office become so cluttered that you can't find your oscilloscope or logic analyzer? No problem, Norman will help you build a low-cost, mixed-signal capture engine that connects to your computer via the serial port.

Some time ago, I had a bad experience with a bus—a logic bus. It had six rampaging DSP cards and a SCSI controller all trying to ride at the same time.

About once an hour, there was a sickening crash. After going through the usual stages of blaming the software, I relented, admitted possible culpability, and borrowed a mixed-mode DSO.

This machine has a digital sampling oscilloscope and a logic analyzer effectively joined at the hip. They share a common trigger module that enables the user to identify a complex event and record the state of the target hardware before and after the trigger—in both the analog and digital domains.

In the case of my erratic bus logic, the culprit turned out to be a delinquent GAL with a ground bounce problem. The offending chip had its duties reassigned and the documentation police were alerted. Engineer triumphs over bug.

By employing a high-tech piece of test equipment, I could trigger on a complex digital event and correlate this event to an oscilloscope trace that showed what was really happening in the analog domain. I was saved in the nick of time, but despite having formed a deep attachment to the trusty 'scope, I had to give it back.

Following this adventure, I started musing about how to roll my own version of that useful electronic gadget. After some mental tinkering and with the added incentive of Design98, I was soon sketching electronic stuff on the grid pad. BitScope began to emerge (see Photo 1).

THE BIG PICTURE

The basic idea behind BitScope is that of a specialized piece of data-capture hardware that doesn't include any user interface other than an RS-232 plug. Most engineers have more computers, mice, and keyboards than they

know what to do with. If I was going to build a cheap 'scope, I certainly didn't want any more of that stuff.

What I needed was an electronic drone that could capture and disgorge data on command. No more, no less.

The commands had to be simple ASCII characters that are intuitive and easy to learn. The PC-based user interface can then synthesize functionality of arbitrary complexity by sending scripts of command characters and receiving the replies.

The answer: a virtual instrument where specialized hardware does the electronic test job and a PC lets the engineer drive it. One big advantage of this setup is that changing the way the virtual instrument works doesn't usually involve reprogramming chips (hard) but may be done by downloading a new program from the 'Net (easy).

As described in the sidebar "Virtual Machine Architecture," the microcontroller firmware is designed as a virtual machine (VM). The BitScope design is novel because it has an unusual arrangement of the VM program code. The instructions are not located in memory on the microcontroller but reside in the user interface and are executed atomically direct from the serial port.

If you study BitScope's virtual instruction set, you see that arranging things in this crazy way has its advantages. All instructions are atomic. In other words, there is no inherent syntax associated with any command byte.

All instruction bytes are echoed to provide a simple handshake mechanism. And, all instructions are preemptive, so you can always abort the previous command and regain control simply by sending a new command.

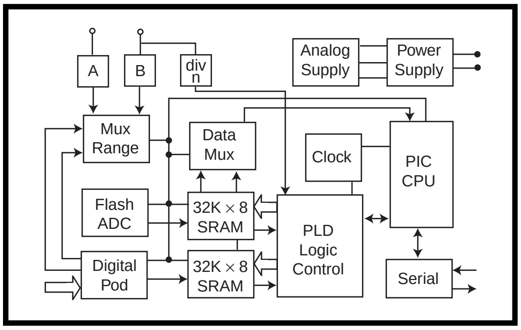


Figure 1—*This block diagram of the mixed-signal capture engine shows basic design architecture.*

SERIAL CONNECTION

While a serial interface may seem a bottleneck for a capture engine that can potentially store 64 KB of data, this is not a problem. Thanks to the Internet and 56k modems, most PCs now have fast, buffered UARTs.

The transmission speed of the BitScope serial link can be scaled to 115 kbps using a fast microcontroller. At this rate, you can transfer enough

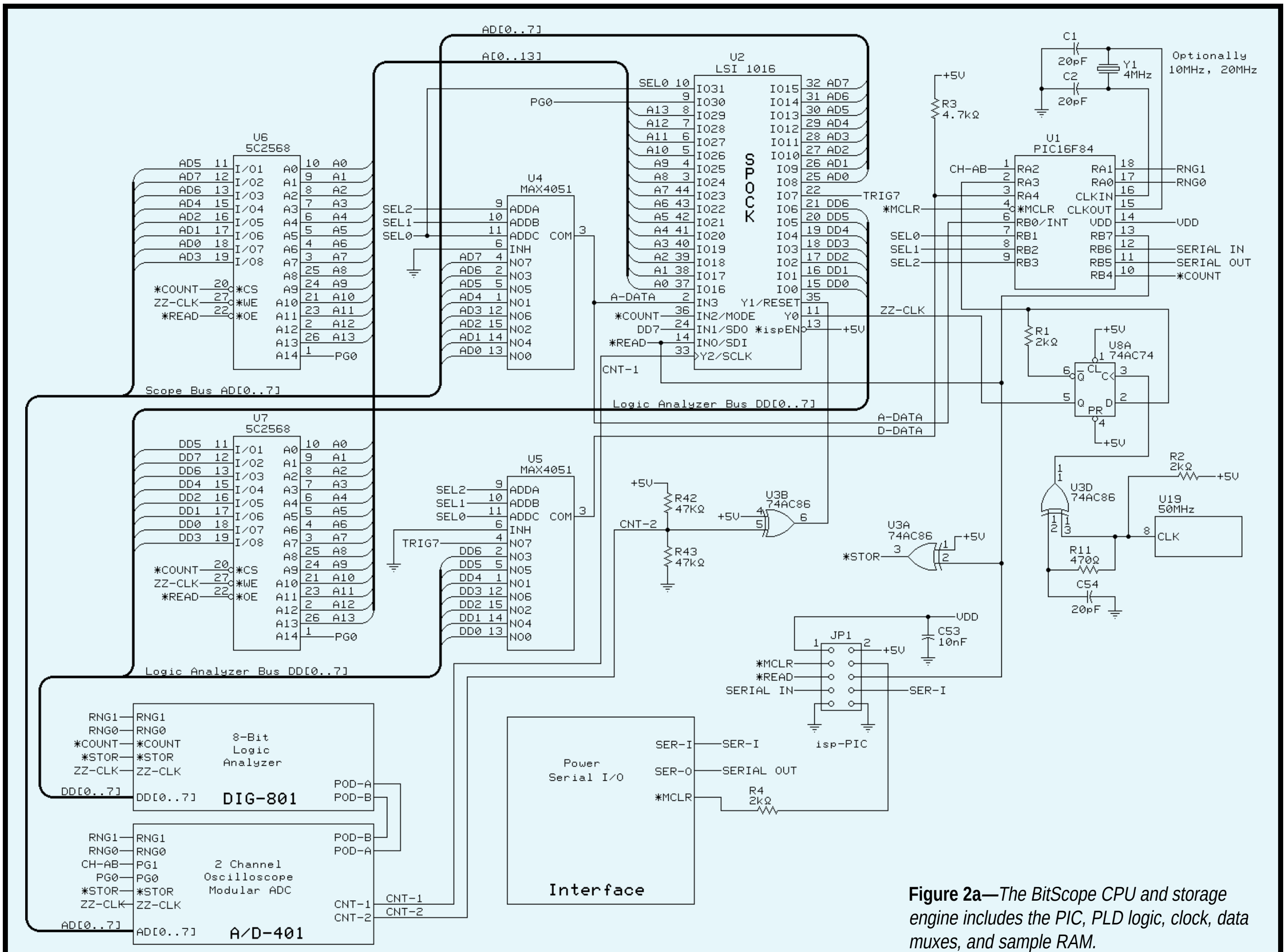


Figure 2a—*The BitScope CPU and storage engine includes the PIC, PLD logic, clock, data muxes, and sample RAM.*

Virtual-Machine Architecture

A virtual machine (VM) consists of a fully functional processor hosted on an unrelated substrate machine. VM design has advantages over conventional coding. Each instruction may be highly optimized for performance—unlike a general-purpose interpreter like BASIC, which can do anything but inefficiently. VM instructions are compact like assembly but perform extremely complex tasks. Once a register and command set are devised, you can add new instructions to enhance the machine. The original instructions remain the same, which promotes modularity. Since the operational definition of the VM is rigid, firmware changes tend to be straightforward, even to the point of hosting the target architecture on a completely new substrate.

In this design, the PIC16F84 is a substrate to implement a custom BitScope machine with its instruction set becoming microcode to implement the VM. So, the BitScope VM has instructions and registers but they're unrelated to the PIC native instruction set. The virtual registers are hosted by PIC memory registers but have meaning only to the BitScope. Similarly, BitScope has no use for XOR- or DECFSZ-type instructions. Instead, it has instructions for manipulating registers, starting sample RAM, and dumping captured data. BitScope registers may be option bits, timer constants, sample address, and so on. The exact function of the register set is detailed in Table i. Table ii shows the current command set.

Most interpreters run from a program stored in memory. BitScope is different because it executes directly from the serial port. BitScope's instruction set is designed to have no syntax, so there can be a maximum of 256 instructions and each is stand alone—just like a RISC instruction set. An atomic protocol means the software at both ends of the serial line is simple and does not have to preserve state information. In a PIC with 1024 words of program, it's advisable to be economical with code, espe-

R0	Byte Input Reg	Assemble input data here
R1	Register Pointer	Pointer to R(0–ff)
R2	Register Source	Pointer to R(0–ff)
R3	Sample Preload L	Low byte of RAM addr to load to Spock
R4	Sample Preload H	High byte of RAM addr to load to Spock
R5	TRIG Logic Byte	Logic levels for Spock to match, loaded during Spock Init
R6	TRIG Mask Byte	Don't Care bits in trigger match, loaded during Spock Init
R7	Spock OPTION byte	TRIG and PG1 setup in Spock
R8	Trace Register	Trace Option controls Sample operation of BitScope
R9	Counter capture Lo	Counter low byte shifted out of Spock
R10	Counter capture Hi	Counter high byte shifted out of Spock
R11	DELAY-L	Post TRIG delay before halting
R12	DELAY-H	Post TRIG delay before halting
R13	TimeBase	TimeBase expander count
R14	Channel-A/B	Channel Range settings for Chop
R15	Dump Length	Counter for number of samples transmitted per request
R16	EEPROM Data	Data register for EEPROM
R17	EEPROM Address	Address register for EEPROM
R18	POD Transmit	Register holds byte for POD
R19	POD Receive	Register gets byte from POD

Table i—*The BitScope virtual machine has a set of 20 registers. The operation of the machine and all its instructions refer to these registers.*

00	•	Reset	Reset the machine and print its ID string
23	#	Load Source Reg	Store R0 into R2. Set up R2 which is a source reg. A reg-to-reg move may be done by pointing to a source (R2) and destination (R1).
2b	+	Inc REG	Incr the reg pointed to by R1
2d	–	Dec REG	Decr the reg pointed to by R1
30	0	Enter nibble 0	Incr R0 by 0 and nibble swap R0
31	1	Enter nibble 1	Incr R0 by 1 and nibble swap R0
32	2	Enter nibble 2	Incr R0 by 2 and nibble swap R0
33	3	Enter nibble 3	Incr R0 by 3 and nibble swap R0
34	4	Enter nibble 4	Incr R0 by 4 and nibble swap R0
35	5	Enter nibble 5	Incr R0 by 5 and nibble swap R0
36	6	Enter nibble 6	Incr R0 by 6 and nibble swap R0
37	7	Enter nibble 7	Incr R0 by 7 and nibble swap R0
38	8	Enter nibble 8	Incr R0 by 8 and nibble swap R0
39	9	Enter nibble 9	Incr R0 by 9 and nibble swap R0
3c	<	Get ctr value from Spock	Shift the current 16 bit ctr value from Spock into R9, R10
3e	>	Program Spock from Registers	Load 5 bytes of data from R3–R7 into Spock. Previous contents of ctr are destroyed
3f	?	Print Machine ID	Print <CR>CHAR8–CHAR1<CR> where CHAR*n* is part of a string identifying the type and revision of this device.
40	@	Load Address Reg	Store R0 into R1. Use to set up reg ptr.
53	S	Dump Sample RAM (CSV)	Dump lines of 16 Sample RAM values (digital and analog)
54	T	Trace with TRIG stop	Begin sample with Opt mode, until Trig then Delay, Halt Sample Clk, and print sample add.
5b	[	Clear R0	Reg R0 is cleared. This usually precedes a nibble load
5d	]	Nibble swap R0	R0:(0–3) is swapped with R0:(4–7). This command puts the entered nibbles in the correct order.
61	a	Enter nibble 'a' hex	Incr R0 by 10 and nibble swap R0
62	b	Enter nibble 'b' hex	Incr R0 by 11 and nibble swap R0
63	c	Enter nibble 'c' hex	Incr R0 by 12 and nibble swap R0
64	d	Enter nibble 'd' hex	Incr R0 by 13 and nibble swap R0
65	e	Enter nibble 'e' hex	Incr R0 by 14 and nibble swap R0
66	f	Enter nibble 'f' hex	Incr R0 by 15 and nibble swap R0
6c	l	Load R0 from @R2	Copy contents of reg pointed to by R2 toR0
6e	n	Next Address	Incr addr reg R1
70	p	Print REG value @R1	Print <CR>ASCII,ASCII<CR>
73	s	Store R0 to @R1	Copy contents of R0 to reg pointed to by R1
75	u	Update RAM pointers	Copy contents of R3,4 to R9,10. Updates sample addr value from sample preload reg.
78	x	Exchange byte with POD	Transmit byte in POD_TX to POD IO-0. Wait for reply byte on IO-1 and put it inPOD_RX then return it to host.
7c	|	Pass Through byte to POD	Transmit byte in POD_TX to POD IO-0. Connect IO-1 to Serial Out for host.

Table ii—*The command set for the BitScope virtual machine is a subset of the byte values between 0 and 255. Active commands are confined to the ASCII range from 0 to 127.*

cially given the importance of reliably transmitting packets over a serial link.

I decided the BitScope command set should use common printable ASCII commands. Since the assignment of byte codes is arbitrary, any value could mean "enter hex nibble 3," but obviously 3 is a good choice. The general scheme for allocating byte-code values and their ASCII symbol is:

- numerals—data entry
- operators—manipulation of register values
- lower case—general machine operation
- upper case—major machine functions
- nonprintables—reserved for future commands

An example script for loading R6 with 0x5a is `[6]@[5a]s`. It may seem obscure, but if you study it, it should make sense. Ultimately, a user interface will debug scripts and writing scripts will only be necessary if a user develops a new mode of operation or drives it directly from a terminal.

All BitScope operations, including wait on trigger, may be interrupted by any serial command. The first part of the software UART ensures that the sample clock is halted. When a serial byte is assembled and echoed, the UART turns on and, once activated, aborts all previous operation. In this sense, BitScope's command protocol is truly atomic. Each command ends in a halt, if not prematurely aborted. ASCII code `00` is the reset vector, so it can get the PIC's attention with a `<break>`.

Inevitably, a VM like this will get enhanced firmware. Microchip has devices that potentially double the number of byte codes implemented. To cope with the potential of other feature sets, ? returns a 32-bit ID code. User-interface software may keep a register of feature sets supported by each byte-code revision.

samples to draw a 640 × 480 screen—at most 640 bytes—in about 55 ms, or 18 screens per second.

For lower frequency data or simple sine waves, it's necessary to only send a handful of samples to the host and have the user interface do some curve fitting. Small bursts of contiguous sample data may be used to enhance a waveform display to show high-frequency noise.

Logic analyzers don't need to rapidly update their display at all. After a trigger event, the data may stay in the sample RAM and be downloaded only when the host needs it. At 115 kbps, the total contents of a 16-KB buffer can download in less than 2 s. The user interface may then draw logic state or timing diagrams and manipulate them as necessary.

USER INTERFACE

Don't think shrink-wrapped monolithic Windows software for this design. Think more about the Linux model where the engineering community builds its own tools and can customize them as needs arise.

Because BitScope uses simple ASCII commands, in a pinch, you can use a terminal program and spreadsheet to display waveforms. For complex applications, you need more advanced software based on C, Delphi, or Visual Basic.

A BitScope user interface can run under many possible environments, including Windows, MAC, Unix, WinCE, Palm Pilot, Psion, DOS, or Amiga. Basically, it can work on any machine with a serial port.

No single person could write all that software. Instead, I made the BitScope design open and documented so you could create what you need.

On *INK*'s Web site, you'll find some user-interface software with source listings to start the ball rolling. Via the Internet, you can also find existing programs that already simulate oscilloscopes, logic circuits, and data displays.

DESIGN PHILOSOPHY

A good place to start designing is with a functional specification. For BitScope, the main issue was sample rate. While it seemed clear that a 200-MHz sample rate was out of reach, I could easily get to about 50 MS/s and still be ahead on the price/performance curve.

For the engineer dealing with microcontroller circuits, it's unlikely that frequencies of interest will exceed 20 MHz—at least for the time being. Later on, when 3-V logic becomes more prevalent, that 50-MS/s rate can probably stretch to 100 MS/s in an SMT version of the design.

To make BitScope as useful as possible, I was determined that it should physically stand alone. It needed to be unconstrained to a particular machine or bus standard, and I wanted it to communicate with any computer using the ubiquitous RS-232 interface.

From my experience, the most commonly required features of this type of test equipment are two analog input channels and eight digital logic inputs. Combine those features with a flexible trigger capability, and you get a pretty useful instrument. I set a design goal of about $100 for the cost of required components, all of which should be readily available.

For the core of BitScope, I selected a PIC16F84 micro tightly coupled with a Lattice 1016 PLD. The PIC controls the serial port and implements a VM architecture. The Lattice counts RAM addresses and waits for a trigger.

These chips are cheap and solid performers. Both are flash-memory based for easy upgrades, and they have excellent entry-level development software. Sample RAM is provided by two 32-KB 15-ns cache memories.

These devices will take the design to 50 MS/s and have the great advantage that about eight of them are perched on every '486 motherboard ever built. That should put their head count at about one billion, so don't tell me you can't find any!

Every DSO must be built around a flash ADC. These chips were exotic until a few years ago when digital manipulation of video became popular.

Now, several companies have devices that can provide 40 MS/s or better for less than $10. Even an older device like Motorola's MC10319P can sample from DC to 25 MS/s and is available in a DIP package.

In fact, I used this device for BitScope. By selecting a 600-mil DIP package, I could accommodate any of the new SMD devices as a plug-in

module and avoid the need for multiple PCB versions.

For vertical amplifiers that process analog signals to the ADC, the video industry again provides a solution. Maxim and Analog Devices both have cheap, stable 300-MHz op-amps that make wide-band amplifier design easy.

Using these devices lets the vertical-amplifier bandwidth get close to 100 MHz, matching the input specs on the new flash ADC chips from Analog Devices and TI. For an insight into why we need such wide-bandwidth vertical amplifiers, see the sidebar "Subsampling—Bending Nyquist."

Photo 1—*BitScope was prototyped on a two-layer PCB. Notice that the components are arranged to separate analog and digital sections of the circuit.*

WALKING THRU SCHEMATICS

Before delving into the schematics, take a look at Figure 1, which overviews the functionality of the BitScope design.

The PIC, the Lattice PLD, and the SRAMs are shown in Figure 2a. These chips are closely coupled to form the sample capture functions at the core of this design.

By using a synchronous tristate clocking circuit, the PIC is able to stop, start, and preload the Lattice PLD using just a handful of signals. Notice that it's necessary to read in data from the RAM chips one bit at a time because there are no spare eight-bit ports available.

One fundamental rule in mixing analog and digital circuits is to avoid contamination of the analog grounds. Figure 2b shows that great care was taken to isolate the analog and digital sections of this circuit at high frequencies. Similarly with the RS-232 port, it's best not to allow PC noise to have any path to a test circuit.

Digital test signals and two spare analog signals are shown on Figure 2c connecting to the DB25M pod connector. Logic levels are latched and conditioned ready for storage in the digital sample RAM.

You might guess from the extra signals on the pod that it's not just eight logic levels in. As well as fused balanced power supplies, there is a digital I/O communication port. Everything you need is there to connect an active, programmable extension module.

Most of the analog conditioning circuits and the flash ADC are shown in Figure 2d. The circuit consists of an amplifier chain driving through a pair of 4:1 analog mux devices.

Modern video op-amps help here. They give you high input impedance, low output impedance, and unity gain stability.

The PIC controls the mux sources that allow implementation of range switching and channel chop functions. To accommodate different ADC chips, there are adjustment pots for both the range and offset voltages as required by the manufacturers.

Figure 2e shows the final part of the analog conditioning circuit. Channels A and B are standard 1-MB input impedance AC/DC BNC connectors. A classic source follower tree driving a unity gain buffer for each channel completes the vertical-amplifier section.

For engineers who like to measure high frequencies, I added a small 1-GHz prescaler circuit, which includes a switchable 50-Ω terminator hanging off the Channel B input circuit. Note that BitScope has a couple of ways to measure the frequencies of applied signals. I explain the motivation behind this in the sidebar "Subsampling—Bending Nyquist."

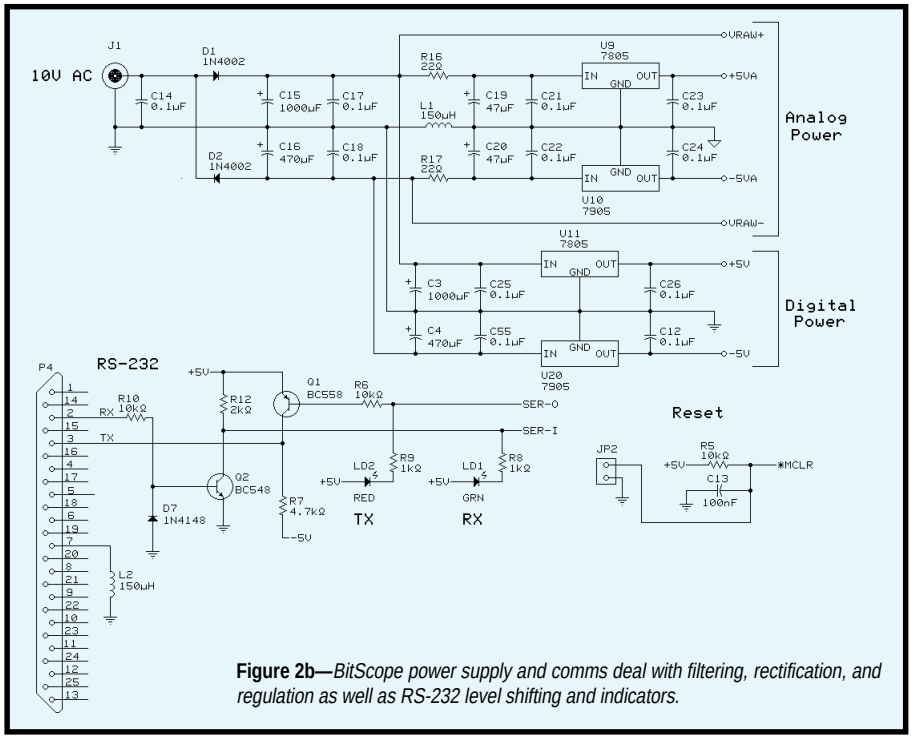


Figure 2b—*BitScope power supply and comms deal with filtering, rectification, and regulation as well as RS-232 level shifting and indicators.*

THAT IS LOGICAL, CAPTAIN

PLDs such as the Lattice 1016 can swallow a whole swag of logic functions. In this case, about 18 medium TTL devices with all their wiring disappear into a 44-pin PLCC device.

Radial PLDs like the Lattice are like eight PALs in a circle surround-

ing a big breadboard. This architecture favors the tight timing requirements of counters and glue logic.

Mostly, this PLD is a 16-bit shift register and counter with a configurable comparator for triggering. The PIC can load a five-byte configuration word that sets the operation of the chip, after which it may be clocked at full speed.

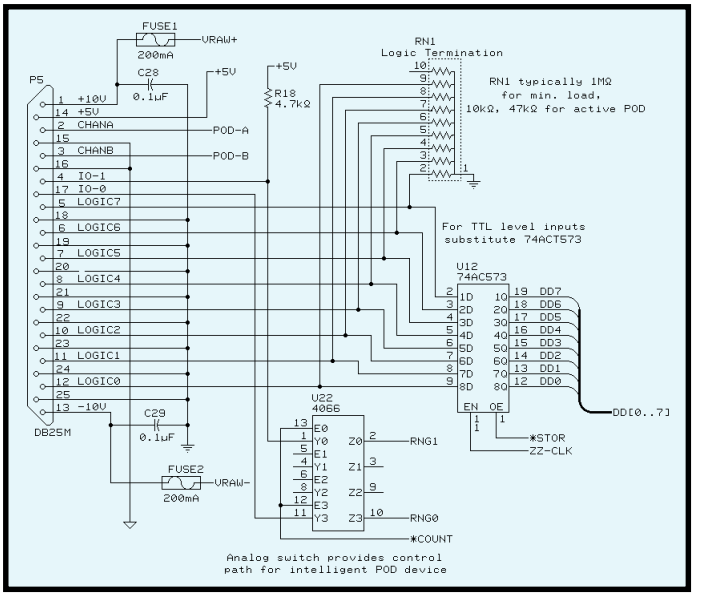


Figure 2c—*The BitScope digital capture unit has a logic pod circuit with latching buffer and pod I/O switches.*

THRU THE LENS MEASURING

The SLR lens-mount system from the photographic world is a great design that has stood the test of time. You start with a camera body with a general-purpose 50-mm lens, and for specialized work, you screw in any of a hundred matching lens types. From fisheye to telescopic, as long as the mounts match, you have a new camera.

I tried to use the same SLR principle in the BitScope design. The device on its own is an extremely useful DSO and logic analyzer, but it is not everything.

The pod connector provides an electronic lens mount for test equipment. Think of the sample RAM in BitScope as a roll of 35-mm film, and the data you store there may come from either built-in connectors or any weird and wonderful "data lens" you care to attach via the data pod. Because the pod architecture and protocol is open and documented, anyone may design a specialized data lens for BitScope.

VOLTAGE RANGES

The BitScope DSO includes four internal attenuation ranges and four channel inputs. Channel A and B are

Subsampling—Bending Nyquist

In data-acquisition applications, there is often some confusion about the relationship between bandwidth and sample rate. The Nyquist rate of half of the sampling frequency (F_s) is well known to be the maximum frequency that can be captured by periodic sampling at F_s. Given that mathematical constraint, why would we want an instrument that has a bandwidth of 100 MHz and yet samples at a maximum rate of only 50 MS/s? The answer lies with subsampling.

The Nyquist rate applies to continuous time varying signals. In that general case, the highest-frequency component should be less than half of F_s (25 MHz at 50 MS/s) to avoid aliasing. Repetitive waveforms are a different matter. They're the only high-frequency waveforms you ever see on an analog CRO. The same waveform is redrawn each sweep, and the eye sees a solid trace. Subsampling is similar. You use multiple samples and overlay them to build an image. Providing that your ADC has a wide bandwidth and a small aperture, it is possible to sample a repetitive waveform over many cycles and build up a snapshot of the exact waveform, limited only by the bandwidth of the signal path. This technique, known as subsampling, is just an example of the RF mixer in the digital world.

Subsampling has a few constraints. It isn't possible to subsample a waveform that's harmonically related to the sampling frequency. Practically, this means that if the waveform of interest is related to the sample frequency, the sample points always fall at the same relative position on the waveform and the regions between will forever remain a mystery.

Another concern has to do with resolving the ambiguous period of the subsampled waveform. Let's say you have a signal of 28 MHz and are sampling at 40 MS/s. In the sample buffer, you'll see a sequence of values with components at 12 and 68 MHz. How can these be plotted to build up a profile of the original 28-MHz signal? Well, if you can measure the fundamental frequency of the sampled wave, that will imply period. Since you know the sample rate accurately, you can fractionally chop the sample buffer up into segments of *n* wave periods and then plot them overlaid. You will have traded the freedom for those *n* waveforms to vary in exchange for *n* different points on the waveform. It may now be apparent why the BitScope design has provision to measure the frequency of any signal presented to the ADC.

Even if you can't measure the frequency of a subsampled waveform directly, all is not lost. DSP engineers have some fancy autocorrelation algorithms that can be let loose on a chunk of acquired data to pull a waveform out of meaningless numbers. It is important to note, however, that for resolving single event (such as high-frequency pulses like logic glitches), there is only one solution: oversample by at least a factor of 10. This performance is exclusively in the domain of specialized test equipment using state-of-the-art circuit techniques to resolve samples to 1 ns or better.

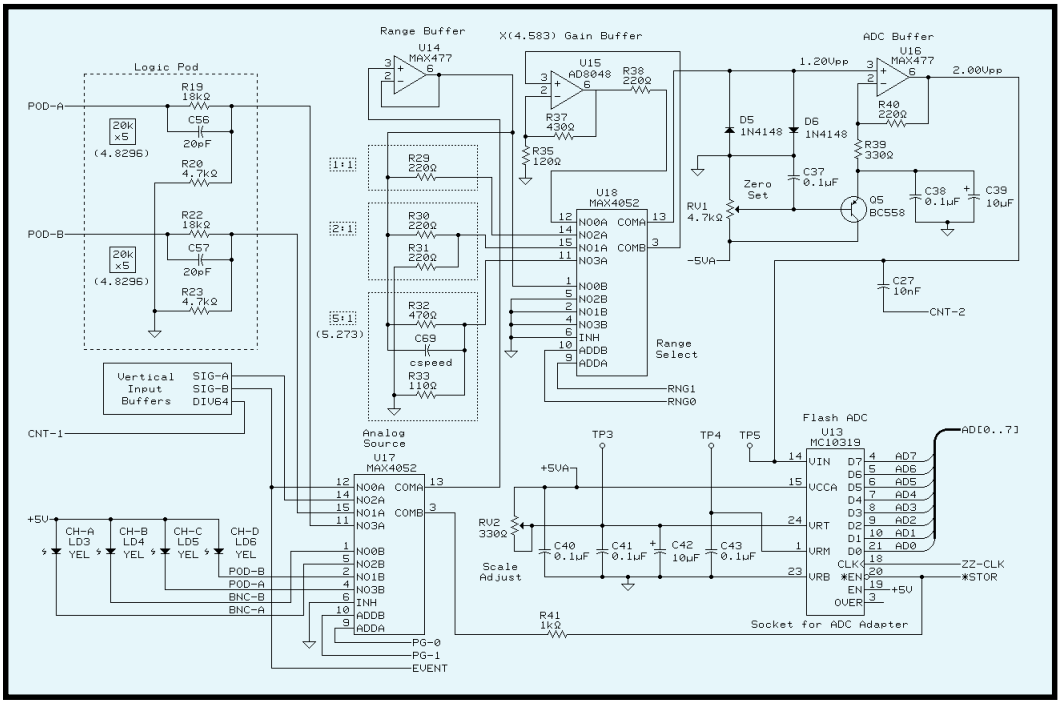
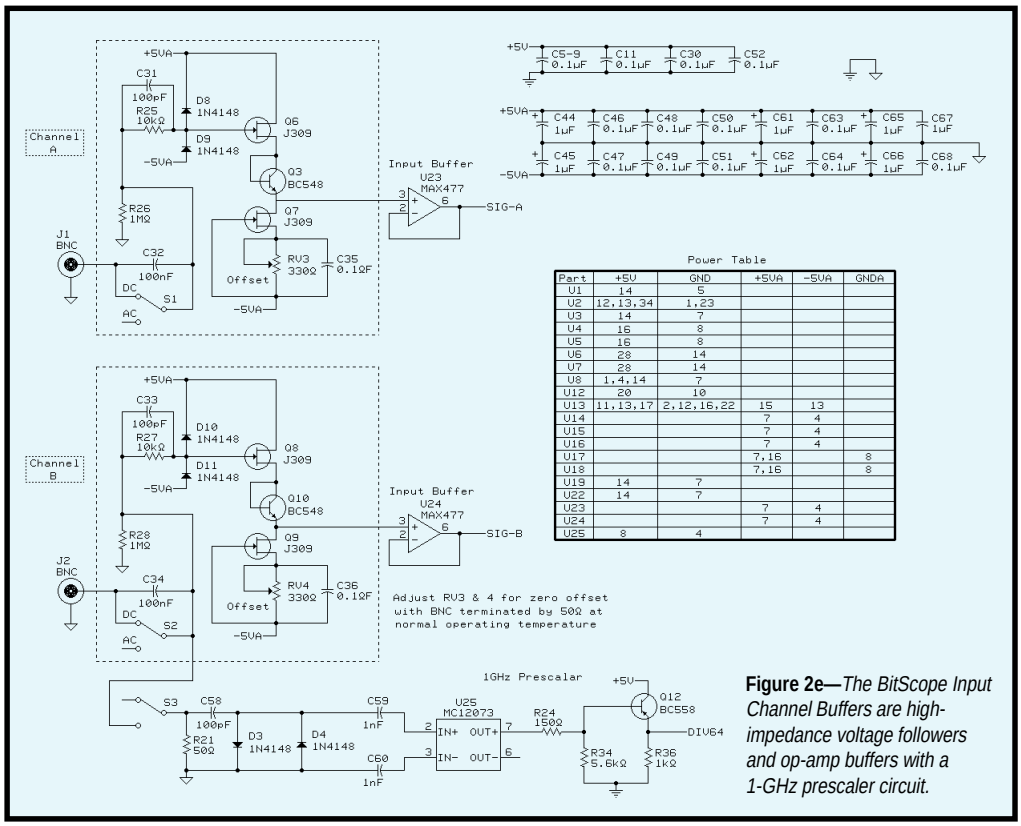


Figure 2d—*The BitScope analog capture features the vertical channel muxes, attenuation switch, ADC buffer, and ADC.*



Part	+5V	GND	+5VA	-5VA	GNDA
U1	14	5			
U2	12,13,34	1,23			
U3	14	7			
U4	16	8			
U5	16	8			
U6	28	14			
U7	28	14			
U8	1,4,14	7			
U12	20	10			
U13	11,13,17	2,12,16,22	15	13	
U14			7	4	
U15			7	4	
U16			7	4	
U17			7,16		8
U18			7,16		8
U19	14	7			
U22	14	7			
U23			7	4	
U24			7	4	
U25	8	4			

Figure 2e—*The BitScope Input Channel Buffers are high-impedance voltage followers and op-amp buffers with a 1-GHz prescaler circuit.*

Range	BNCx1	BNCx10	POD
00	±130 mV	±1.30 V	±632 mV
01	±600 mV	±6.00 V	±2.90 V
10	±1.20 V	±12.00 V	±5.80 V
11	±3.16 V	±31.60 V	±15.28 V

Table 1—*Here are the BitScope input ranges for an ADC span of 2 V. Resistor attenuators can be found in the schematic.*

BNC connectors that may have ×1 or ×10 probes connected. Channel C and D (pod inputs) have a fixed attenuator, and possibly, there's some extra circuitry in the pod.

Table 1 details the range sensitivities. The ranges aren't nearly as comprehensive as a bench CRO, but it covers those most useful to digital and analog circuits. As well, I intended for the pod connector to deal with unusual or high voltage signals by way of an active pod adapter.

It's also possible to alter the gain of some ranges. Since the ADC output is an eight-bit number that ranges from 00 to FF, the final interface just needs to ratiometrically apply this hex value to the voltage range of each stage.

A little thought reveals that for a digital oscilloscope, volts per division and microseconds per division are quite arbitrary notions. Provided that the signal under consideration is within the ADC range and the sample-buffer size, a display can be of any size and grid spacing. Similarly, the notion of *y* offset becomes a display function, which has nothing to do with the sample engine.

IN YOUR HANDS

With this design, I hope to have presented a low-cost solution to the engineer's needs for sophisticated test equipment. I have heeded the call for more open designs and liberation from the single-platform juggernaut.

In the coming months, I look forward to hearing from any of you who can think of applications for this device that I haven't even dreamed of.

Norman Jackson is principal hardware design engineer for Discrete Time Systems P/L in Sydney, Australia. He designs DSP-based digital audio systems for use in film and TV postproduction. You may reach Norman at normj@discrete.net.

SOFTWARE

The Circuit Cellar Web site has downloadable software listings, technical documents, programmable binaries, and PCB overlays. Information about BitScope is available at www.discrete.net or via bitscope@discrete.net.

SOURCES

PIC16F84
Microchip Technology, Inc.
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

1016 PLD
Lattice Semiconductor Corp.
(503) 681-0118
Fax: (503) 681-3037
www.latticesemi.com

MC10319P ADC
Motorola SPS
(800) 521-6274
Fax: (602) 897-5725
www.mot-sps.com

Preprogrammed PIC, 1016 PLD, MC10319P ADC, and PCB
Discrete Time Systems
+612 9212 3469
Fax: +612 9212 3470
bitscope@discrete.net
www.discrete.net