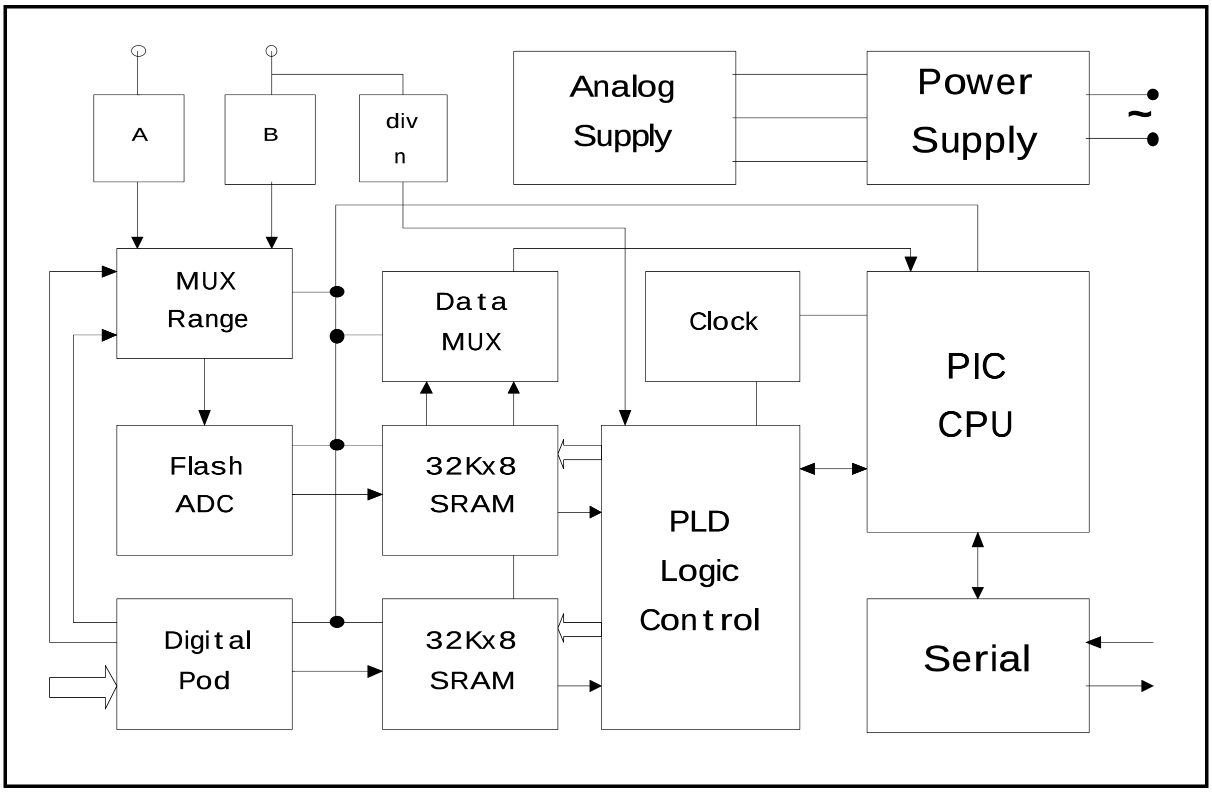
BitScope

Mixed Signal Capture Engine

Features

- Simultaneous Analog/Digital capture
- 100MHz Analog Channel Bandwidth
- 4 Analog In channels multiplexed
  - 2 Analog In via BNC buffered 1Meg
  - 2 Analog In signals via DB25 POD
- 8 Digital Levels via DB25 POD
- 50MS/s Capture rate Maximum
- Two 32K x 8 deep capture buffers
- Complex trigger on Analog or Digital
- 1GHz Prescaler for Frequency Count
- Low cost PIC 16F84 design
- Logic functions in single PLD
- Full Duplex serial interface
- "Thin Server, Fat Client" model
- ASCII character Command Set
- Output as CSV or Byte-Stream
- POD has hooks for external device
- ADC socket allows SMD devices
- Scalable design to >100MS/s @ 3.3V
- Chipset suitable for embedded DSO



General

Any engineer who has developed an electronic circuit - from a single chip micro to a multiprocessor megaproject - will have desired a Digital Sampling Oscilloscope and a Logic Analyzer (preferably in one box). These devices have been the domain of the big companies like HP and Tektronix and have had blood curdling prices. This usually places them out of the grasp of people who could really use them - an increasing number of designers who are driving the ubiquitous microcontroller into every facet of our lives.

Furthermore, the design philosophy of test equipment has tended to favor the portable, self-contained unit. This means that the "Test Equipment" has a computer, an operating system, a display, a disk, and a power supply. And after all that - the test electronics itself! This does suggest a few re-invented wheels, especially if you look at what is available these days from Pentiums to Palm Pilots. Consider also that the functionality of a 200MHz PC with an SVGA display is hard to duplicate in any embedded system. Engineers like the smooth green trace of a good Analogue CRO - as yet unmatched by any DSO that I know of. Considering what you see in games software these days, a 200MHz CPU with 16 bit colour should be able to draw a reasonable simulation of a CRT in real time - with a few MIPS to spare!

In the last year or so, the makings for a decent sampling device have become readily available. Not 1GS/s - still the domain of HP/Tektronix -, but 50-100MS/s at least. A number of designs have appeared that use the Printer Port or plug into the ISA bus. Not many use a serial interface, or include a Logic Analyzer function. As such, the BitScope is unique in packaging many useful functions in an economical design without being limited to any specific hardware platform.

Serial link - a good design decision

While a serial interface may seem a bottleneck for a capture engine that can potentially store 64K bytes of data, this is not necessarily a significant problem. Thanks to the Internet and 56K modems, most PCs now have fast buffered UARTS, so the transmission speed should be able to be scaled to 115K Baud with a fast PIC. At this rate (~11.5K Bytes/S) enough samples to draw a 640x480 screen - at most 640 bytes - can be transferred in about 55 mS (18 times/sec). For lower frequency data or simple sine waves it is possible to only send a handful of samples to the Host and have the Host software do some curve fitting. Small bursts of sample data may be transferred to the Host to let it draw a trace showing high frequency noise etc.

Logic Analyzers do not need to rapidly update their display. After a trigger event the data may stay in the Sample RAM and only be downloaded when the Host needs it. At 115K baud, the total contents of a 16K buffer can be downloaded in less than 2 seconds. The PC can then draw logic state or timing diagrams with zoom, scroll, search etc.

Net Design

I have long been an admirer of the Linux development philosophy. An open design available for general access on the net encourages a collaborative development effort. Many projects these days are too complex and difficult for a single person or even an organization to take on as a whole. The Internet has changed the way the engineering community works.

The BitsScope design follows this new net philosophy in a number of ways.

- The schematics and design description are available encouraging the understanding of the principles of operation of the device.
- The design does not attempt to enclose all functionality in a single microcontroller. The micro implements an open architecture interpretive virtual machine allowing anyone to develop an interface or application for it.
- The generic Logic POD connector is documented allowing anyone to design a specific hardware application for use with the BitScope.

It is my hope that other engineers and students of electronics will find this design of some benefit. It is also another fond hope that if someone can develop a DOS based user interface for the beast, we might just have a retirement plan for some of those old 386 machines cluttering up our lives!

BitScope

Overview

The BitScope Mixed Signal Capture Engine is an RS-232 peripheral device intended for data acquisition applications where a Digital Sampling Oscilloscope or Logic analyzer would be used.

The device is not stand alone. To operate it must be programmed by scripts from a Host attached to the serial port. The scripts are virtual machine instructions which synthesize the required functionality from the BitScope specialized operations.

BitScope is capable of complex triggering and simultaneous recording of analog data and digital logic states, as well as Time / Frequency measurements.

Virtual Machine Architecture

A virtual machine is an architecture which is hosted on an an unrelated substrate machine. In this case I have chosen the PIC microcontroller as a substrate to implement a custom BitScope Machine, and the PIC instructions are effectively microcode. This means that the BitScope Virtual Machine has instructions and registers - but they are unrelated to the PIC native instruction set. The BitScope Machine has no use for XOR or DECFSZ type instructions. Instead it has instructions for manipulating registers, starting sample RAM, and dumping captured data.

A Virtual Machine design has a number of advantages.

- Efficiency - Each instruction may be highly optimized for performance. A general interpreter like BASIC can do anything - but in a very inefficient way. A Virtual Machine instruction is compact like assembly code, but may perform an extremely complex task.
- Modularity - Once a register set and basic command set are devised, extensions may be made by adding new instructions to enhance the machine. The original instructions remain the same.
- Portability - changes to the substrate machine have less of an impact.

Direct Execution from Serial Port

Most interpreters run from a program stored in memory. BitScope is slightly different, in that it executes directly from the serial port. As you will see later on, the BitScope Machine instruction set is designed to have no syntax. This means that there can only 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. Anyone who has studied network protocol will appreciate the complexity of reliable transmission of packets over a serial link. In a PIC with 1024 words of program it is advisable to be economical with code.

Virtual Registers

Within the virtual machine are virtual registers. The BitScope machine is designed to look like a complex peripheral interface. The virtual registers are hosted by real PIC memory registers, but have meaning only to the BitScope. An example is the Trigger Mask register. This register is loaded with the mask bits for the TRIG byte and will be used by the virtual machine whenever it reloads the PLD.

Other BitScope registers may be option bits, Timer constants, Sample Address and so on. Instructions are defined to read and write all these registers.

Writing a Script

To make it easy for humans to understand, the BitScope command set has been chosen to use common printable ASCII commands. Since the assignment of bytecodes is arbitrary, we could use any value to mean “enter hex nibble 3” but obviously ‘3’ is a good choice for this one.

An example script for loading R6 with 0x5a is:-

```
[6]@[5a]s
```

This may seem a little obscure, but if you study the operation of each bytecode it should make sense. It is envisioned that ultimately a user interface would use debugged scripts to drive the BitScope and writing scripts would only be necessary if a user was developing a new mode of operation - or driving it directly from a terminal.

User Interface

The virtual machine architecture makes a clear separation between the Host computer and the peripheral. This allows more appropriate software methods to be used for implementing the user interface.

A typical user interface could be constructed using Visual Basic or Delphi. This would allow the programmer access to sophisticated graphics tools and off the shelf buttons, sliders and menus. A VB program would only have to deal with a comms port to interface with the BitScope.

A mechanical engineer who needs to collect transducer data may write a nongraphical interface to collect sampled data periodically in csv format.

Post capture signal processing is also a task more suited to a PC than a micro. Subsampling the H.F. RF output of a transmitter should be possible with the BitScope - allowing a fast PC to be a spectrum analyzer.

A Palm Pilot or Win CE developer may write a dinky interface for the BitScope so you can debug something on your palm.

A SPICE program could include a BitScope interface so you could see *real* and *simulated* waveforms from a circuit simultaneously on your PC.

BitScope Register Set

The BitScope virtual machine has a set of 20 Registers R0..R19. The operation of the machine and all its instructions refer to these registers. All other resources in the PIC are outside the scope of the virtual machine.

The function of the Register Set is detailed below. In some cases related commands are shown as examples of how these registers are used.

R0

Byte Input Register	assemble input data here
[	clear R0
0..f	increment R0 by 0..f then nibble swap
]	nibble swap R0

R1

Register Pointer	pointer to R(0..ff)
@	copy R0 to R1
s	move contents of R0 to R(R1) - s(tore)

R2

Register Source	pointer to R(0..ff)
#	copy R0 to R2
l	move contents of R(R2) to R0 - l(oad)

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

R7:0	Digital TRIG / Analog TRIG
R7:1	Select source of TRIG7 Dig MUX
R7:2	Select source of TRIG7 Dig MUX
R7:3	Set Bit PG1 to RAM and Analog Source

R8

Trace Register	Trace Option controls Sample operation of BitScope Machine
R8(0..3)	4 bit vector to different Trigger and Chop modes.
R8(4..7)	4 bit pointer to select one of the Data bus MUX device channels

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 - H / L nibbles contain alternate values for RA(0..3)

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

BitScope Command Set

The command set for the BitScope Virtual machine is a subset of the bytes values between 0 and 255. Initially, active commands are confined to the ASCII range from 0 to 128. Where ever possible, command values correspond to a character with a meaning related to the command function. This makes the byte-codes human readable - with some practice.

The general scheme for allocating byte-code values and their ASCII symbol is as follows:-

numerals	used for entering data
operators	manipulation of register values
lower case	general machine operation
upper case	major machine functions
non-printables	reserved for future commands

Unused bytecodes will be echoed if printable, else ignored.

00 - € Reset

Reset the machine. Print the machine ID string

23 - # Load Source Register

Store R0 into R2. Set up R2 which is a source Register. A Register to Register move may be done by pointing to a source (R2) and destination (R1).

2b - + Inc REG

Increment the register pointed to by R1

2d - - Dec REG

Decrement the register pointed to by R1

30 - 0 Enter nibble 0

Increment R0 by 0 and nibble swap R0

31 - 1 Enter nibble 1

Increment R0 by 1 and nibble swap R0

32 - 2 Enter nibble 2

Increment R0 by 2 and nibble swap R0

33 - 3 Enter nibble 3

Increment R0 by 3 and nibble swap R0

34 - 4 Enter nibble 4

Increment R0 by 4 and nibble swap R0

35 - 5 Enter nibble 5

Increment R0 by 5 and nibble swap R0

36 - 6 Enter nibble 6

Increment R0 by 6 and nibble swap R0

37 - 7 Enter nibble 7

Increment R0 by 7 and nibble swap R0

38 - 8 Enter nibble 8

Increment R0 by 8 and nibble swap R0

39 - 9 Enter nibble 9

Increment R0 by 9 and nibble swap R0

3c - < Get counter value from Spock

Shift the current 16 bit counter value from Spock into R9, R10

3e - > Program Spock from Registers

Load 5 bytes of data from R3..R7 into Spock. Previous contents of counter are destroyed

3f - ? Print Machine ID

Print <CR>Char8..CHAR1<CR> where CHARn is part of a string which identifies the type and revision of this device.

40 - @ Load Address Register

Store R0 into R1. Use to set up register pointer

53 - S Dump Sample RAM (CSV)

Dump lines of 16 Sample Ram values - digital and analog.

```
<CR>ddaa,ddaa,ddaa,....................ddaa,ddaa<CR>
    ddaa,ddaa,ddaa,....................ddaa,ddaa<CR>
```

54 - T Trace with TRIG stop

Begin Sample with OPTION mode, until TRIG then DELAY, Halt Sample Clk, and print Sample Address.

5b - [Clear R0

Register R0 is cleared. This usually precedes a nibble load

5d -] Nibble swap R0

R0:(0..3) is swapped with R0:(4..7). This operation follows a nibble entry and puts the entered nibbles in the correct order.

61 - a Enter nibble ‘a’ hex

Increment R0 by 10 and nibble swap R0

62 - b Enter nibble ‘b’ hex

Increment R0 by 11 and nibble swap R0

63 - c Enter nibble ‘c’ hex

Increment R0 by 12 and nibble swap R0

64 - d Enter nibble ‘d’ hex

Increment R0 by 13 and nibble swap R0

65 - e Enter nibble ‘e’ hex

Increment R0 by 14 and nibble swap R0

66 - f Enter nibble ‘f’ hex

Increment R0 by 15 and nibble swap R0

6c - l Load R0 from @R2

Copy contents of register pointed to by R2 to R0

6e - n Next Address

Increment address register R1

70 - p Print REG value @R1

Print <CR>ASCII,ASCII<CR>

73 - s Store R0 to @R1

Copy contents of R0 to register pointed to by R1

75 - u Update RAM pointers

Copy contents of R3,4 to R9,10. Updates Sample Address value from Sample Preload registers.

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 in POD_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.

Trigger and Chop Operation

The core functionality of the BitScope is contained in the **T** (**T**race until TRIG) function. After setting up all the registers correctly, this is effectively the GO button.

Spock Preloads

Before initiating operation, it will be necessary to set Spock into a known state. R7..R3 contain the preload values which are loaded with the ">" command.

Example - Load Spock with 0, 0f, aa, 00, 00

```
[3]@[]sns[aa]ns[f]ns[]ns>
```

[3]@	enter 3 into R0, then copy it to R1
[]s	enter 0 into R0, then store it at (R1)
ns	increment R1 to 4, then store R0 at (R1)
[aa]ns	enter aa in R0, inc R1, then store R0 at (R1)
[f]ns	enter f in R0, inc R1, then store R0 at (R1)
[]ns	enter 0 in Ro, inc R1, then store R0 at (R1)
>	load Spock with R3..R7

Once these inital preloads have been set up, a single ">" command will reset Spock to a known condition for a retrace.

Channel Register

R14 controls the Channel A / B and range selection during sampling. This register contains 2 nibbles - normal (low) and alternate (hi) which are put out on PortA of the PIC (see Hardware description).

R14[0]	RNG0 to attenuator	
R14[1]	RNG1 to attenuator	
R14[2]	CH-A/B	selects A or B as analog source
R14[3]	zz-clk	value to freeze zz-clk to

The high nibble of R14 has an alt version of these same bits. Depending on the *Trace Mode*, these 2 nibbles may be swapped repeatedly while waiting for TRIG event, effecting a CHOP function

Trace Option

When the machine begins a trace, it will fall to a loop that polls for a trigger event. The Trace Option register R8 contains a vector to a number of different loop algorithms. These will implement different modes of behavior. There are 16 possible modes, not all currently defined.

Trace Mode 0	Single channel mode simple
Trace Mode 1	Single channel TimeBase expansion
Trace Mode 2	CHOP mode simple
Trace Mode 3	CHOP mode TimeBase expansion

TimeBase expansion

R13 holds a byte that can expand the machine Timebase. In the Trace loop (above) the sample clock zz-clk may be frozen for a count of R13, then turned on for a burst (1uS). This will effectively expand the period over which we may sample a waveform.

Post TRIG Delay

These two registers contain a 16 bit counter that must be counted down after a trigger is hit. this is like delayed sweep. The sample clock (zz-clk) is not frozen until the delay is executed. For each iteration of the DELAY, the TimeBase expansion delay value is executed. This magnifies the TRIG Delay by 1..255 - giving delays of up to several minutes. Depending on the Trace mode, TimeBase Expansion may not apply to the sample mode, but will always apply to the Post TRIG Delay.

Trace until TRIG

The "T" command will cause the machine to start sampling and go into a tight loop waiting for a TRIG event. Upon seeing the TRIG condition, the Post Trig delay is initiated before halting the machine. The contents of Spocks address counters will be read and this value printed to the serial port.

Note that the operation of the Sample Loop will be set by the Trace option register. Up to 16 different algorithms may be selected to achieve different results including frequency and period measurements.

Example

>T	reload spock, then sample
73a0	hit the TRIG, delay, print frozen address
S	dump samples from SRAM 13a0
00a2,00a3,00a7,019e,009f.........0176	Data!
>T	arm the TRIG then go again
7395	hit the TRIG again (nearly same place)
....etc	

Note that the Spock counter is a full 16 bits, but the buffer regions it accesses are only 16K. Since we have 32Kx8 SRAM chips, this is explained by the PG1 bit (set in Spock) bank selecting the upper or lower 16K. The full counting range of Spock may be used for timing resolution if required, but the real SRAM will be aliased in all 4 16K regions of the address space.

Hardware

The circuits for the BitScope capture engine are included at the end of this document. there are 5 sheets as follows:-

1 **BitScope CPU and Storage Engine** - covers the PIC, PLD logic, Clock, Data MUXs and Sample RAM.

2 **BitScope Power Supply and Comms** - Filtering, Rectification and Regulation. RS-232 level shifting and indicators. Reset circuit (for debug).

3 **BitsScope Digital Capture** - Logic POD circuit with latching buffer and POD I / O switches.

4 **BitScope Analog Capture** - Vertical channel MUXs, Attenuation switch, ADC Buffer and ADC.

5 **BitScope Input Channel Buffers** - High Impedance voltage followers and op-amp buffers. 1GHz prescaler circuit.

Picard is in command

The PIC 16F84 is an extremely versatile chip, highly suited to control applications where interface to logic circuits is required. The fast RISC instruction set and the 3-state I / O pins are a clear advantage for machine control.

At first glance you might think that an 18 pin micro has insufficient pins for this application. This is not so. With careful design and planning it is possible to use most pins for more than one function. The allocation of pins is described as follows.

RA0 - Range 0

RA1 - Range 1

Output pins in COUNT mode that control the Analog gain of the Y amplifiers (sheet 4).

0	Gain =	4.583
1	Gain =	1.000
2	Gain =	0.500
3	Gain =	0.190

Note that the gain of the ADC Buffer is 1.667

RA0,RA1 are I / O pins in SHIFT mode that are available at the LOGIC POD for external Smart PODs. In COUNT mode, analog switches isolate these signals from the LOGIC POD.

RA2 - Channel A / B

Output pin that swiches the Analog source between CHA BNC and CHB BNC, or in POD mode between PODA or PODB analog signals brought from the LOGIC POD.

RA3 - zz-clk: The Clock That Sleeps

Three state I / O pin. State machines in PLD devices should not be exposed to glitches on the clock. Some registers may see the clock, others may may not, resulting in non-deterministic behavior. Jamming a 50MHz clock signal is not polite. U6A (74AC74) is double clocked by frequency doubler U3D (74AC86) (on the clean edge) and allows the PIC to have 3 state synchronous control via RA3. High, Low, Freerunning - no glitches. This control is important as we must use zz-clk to shift in control words to the PLD as well as modulate the sampling for lower frequency timebase measurements.

RA4 - Digital Data

Input connects to U5 8:1 MUX to read the 8 data bits of the Logic Analyzer RAM bus. Bit7 is routed through Spock to be optionally the TRIGGER MATCH or FREQUENCY / EVENT signals. RA4 may be connected to the PIC prescaler in the OPTION register.

RB0 - Analog Data

COUNT mode - Input connects to U4 8:1 MUX to read the 8 data bits of the ADC RAM Bus. As RB0 is the PIC INT source, during SAMPLE, if the MUX points to Bit7 then INT may be used as a zero crossing detector for Analog TRIGGER (in additional to the Complex trigger implemented in the PLD).

SHIFT mode - Output signal is Shift-In data to feed Spock which needs 5 bytes to set up counters, trigger bytes and options.

RB1 - SEL 0

Count mode - Output A0 for data MUXs U4, U5.

Shift mode - Input Shift-Out data from Spock. Current 16 bit counter state in Spock is shifted out as new data is shifted in.

RB2 - SEL1

RB3 - SEL2

Output pins A1, A2 for data MUXs U4, U5.

RB4 - SHIFT / !COUNT

Output controls the function of Spock.

1	causes Spock to shift new data into its registers
0	causes Spock to count and trigger

RB5 - Serial Out

Serial data output

RB6 - Serial In

Serial data input

RB7 - STORE / !READ

Output control for RAM and ADC / Buffer control. This signal selects either read or write for the RAM, allowing SAMPLE data written to RAM, or later read back.

CLKI, CLKO

The PIC is clocked by a crystal - which depending on the PIC device may be 4MHz, 10 MHz or 20MHz.

Spock is Logical

The PLD soaks up a large amount of logic and makes the circuit look much simpler than it is. The functionality implemented in the PLSI1016 is about 16 - 20 medium density TTL devices.

The two main functions of Spock are address generation for the SRAM, and 8 bit comparator for trigger generation.

All functions of this device are driven by zz-clk in either of two modes:-

a Count Mode - shift / count low The counter will be counting and the comparator will be monitoring the byte stream looking for trigger matches to pass on to RA4 (TOCI).

b Shift Mode - shift / count high The counter will be shifting in bits from A-DATA (RB0), and shifting out bits to SEL0 (RB1). This means we can reload the counter and find out where it was up to when we stopped it. Bits from the top of the counter also get shifted up to the comparator circuits to load trigger words.

Spock contains 5 bytes of loadable registers:

R0	-	16 bit counter low
R1	-	16 bit counter high
R2	-	8 bit TRIGGER PATTERN
R3	-	8 bit TRIGGER MASK
R4	-	4 bit option

R4:0	Digital / Analog Bus as TRIG Source		
R4:1,2	TRIG7:	00	DD7
		01	TRIG MATCH
		10	EVENT 1
		11	EVENT 2
R4:3	PG1 - selects BNC / POD for ADC input and hi / lo 16K of RAM		

TRIGGER OPERATION

The TRIG comparator may be set to monitor either the Logic Bus or the Analog Bus. Bit values will be compared to the PATTERN register, or ignored where the MASK is a ‘1’.

	Pattern	01100100
+	Mask	00001111
->	TRIG	0110XXXX

The meaning of this triggering is obvious for Logic Analyzer functions, but not so clear for Analog trigger. Obviously, setting the mask to 0111 1111 will select the MSB of the ADC output - meaning trigger on zero crossing.

By masking off the lowest 4 bits of the ADC byte we can select 1 of 16 “Bands” of ADC output and trigger as the ADC crosses the edge of this band.

By masking off some MSBs and some LSBs, we can make a sort of AC signal detector trigger. And of course, many other of the 2^16 combinations will be absolutely useless!

Trigger Examples

With this set of logic functions it is possible to make Spock and Picard do some nice sampling tricks.

Pre and Post Trig Display

- Set up TRIG event - DIG / ANALOG
- Start Spock counting, wait for TRIG
- Delay 1 / 2 Sample RAM Buffer
- Freeze zz-clk
- Shift out frozen 16 bit counter
- Dump to host Pre / Post data

Delayed TRIG

- Set up TRIG event - DIG / ANALOG
- Set up Post TRIG delay parameter
- Start Spock counting, wait for TRIG
- Delay Post TRIG delay parameter
- Delay 1 / 2 Sample RAM Buffer
- Freeze zz-clk
- Shift out frozen 16 bit counter
- Dump to host Pre / Post data

Slow TimeBase Capture

- Set up TRIG event - DIG / ANALOG
- Set up SAMPLE / FREEZE mark / space
- Set Spock RAM counters to 0000
- Enable zz-clk for ~ 5uS
- HALT zz-clk high (so logic buffer is transparent)
- Delay space time
- repeat previous 3 steps N times
- STOP and dump Spock state to Host

Chop and Alt

Chop is accomplished as above, but the PIC flips the CHA / B signal at ~ 200KHz. At the end of the SAMPLE, after N flips, zz-clk is frozen and the counter in Spock is dumped to the Host. The chop positions may be calculated as :-

$S_n = n * S_{final} / N$ (within the grasp of a Pentium)

Note that if the Host displays the contents of the Sample RAM after a CHOP capture, what you will see is similar to an Analog Oscilloscope display. The CRO is doing the same thing!

Alt is accomplished by reducing the buffer length and flipping CHA / B each Analog TRIG event

Frequency Period Measurement

The TRIG7 output from Spock may be driven from a clock derived from either the 1GHz prescaler, or the signal present at the input of the ADC (allowing gain to be set for varying signal magnitudes). Frequency can be derived by using the TOCI (DigitalData) on the PIC - using normal counter techniques.

Alternatively, for lower frequencies, a full buffer of samples may be taken. The trigger is then set for zero crossing and Spock is single stepped through from address 0000. The trigger points (TRIG still works in Read Mode!) are dumped to the host, defining the period over N cycles. This would mean, for a sample RAM of 16K, the period (hence frequency) could be resolved to up to 1 part in 16,000 - 0.006%, or ~ 65ppm. This technique could be extended to very low frequencies that exceed the 16 bit counter wrap around, where Spock is used to resolve the end point of the waveform.

The above techniques use the logic in Spock, but do not require large amounts of data to be transferred to the Host.

Serial Interface

Transistors Q1, Q2 level shift the serial In / Out signals. LD1, LD2 LEDs monitor the serial data. Inductor L2 isolates the BitScope circuit from high frequency noise on the serial cable.

Power Supply

Socket P1 connects to a nominal 10Vac PlugPak. D1,D2 form two 1 / 2 wave rectifiers to give us a split rail supply (Vraw- and Vraw+) of ± 10Vdc. U11, U20 regulate +5, -5 for digital circuits.

Analog supplies +5, -5 are regulated by an isolated pair of 3 terminal regulators U9, U10. Note inductor L1 isolates the Analog ground from the Digital ground at high frequencies. In mixed signal designs it is essential to guard against digital noise - like ground bounce - getting into the analog circuits.

Logic Pod

The Logic Pod circuit represents a significant feature of this design. U12 (74AC573) is a transparent buffer which latches the 8 logic levels for writing into the RAM chip. This buffer has input pulldown resistors (nominally 1Mohm).

Analog switch (U22) connects RNG0, RNG1 signals from the PIC to the Logic POD. These signals are available during Spock Shift operations.

Vraw+, Vraw- are available through 200 mA poly fuses at the Logic POD. This allows external POD devices to be powered from the BitScope. +5 regulated is also available.

Note also that 2 analog signals are brought in via the Logic POD. These are the 3rd and 4th Analog channels (selected via PG1 option in Spock). These signals have Analog grounds which should not be connected to Digital grounds.

Applications for Logic POD

The following ideas suggest possible uses for the Logic POD connector. Obviously, the minimum requirement is a 25 way ribbon cable terminated by a pin header for connecting probes.

By providing I / O control signals it is possible to have POD devices that can be configured or provide extra functionality. Note that the ByteCode protocols and scripts to implement these functions are currently not defined, and in some cases may require a PIC with more Program memory than a 16F84. Fortunately Microchip have roadmap of 18 pin enhanced PICs to take us up to 4K of program words!

- Logic Analyzer POD with active TTL level Buffers with signal conditioning.
- Logic Lab POD - a breadboard POD including supplies and Digital / Analog monitoring for circuit development. I / O pins used for configuration.
- Serial protocol analyzer - serial stream fed through a shift register which is monitored by Spock TRIG function to detect sequential patterns.
- 10 Bit Logic Analyzer - 8 level + 2 Analog logic
- Differential Voltage probe for analog signals
- High voltage, or millivolt active probes for analog channels.
- Data Acquisition POD - Current, Voltage, Temperature etc resolved and stored as bytes to Logic RAM, handshake via I / O pins
- PIC Programmer POD
- Spectrum Analyzer POD
- Video Capture POD - capture 8 full video lines for analysis,
- Component Tester
- SONAR POD

Analog

The analog circuitry of BitScope includes the Flash ADC converter, ADC buffer with offset adjust, Range MUX, Analog Source MUX and Vertical Input buffers.

ADC

The Flash A / D Converter is the core component of the Analog part of BitScope. Flash converters have been around for some time - mostly for digitizing video. As such a 25MHz sample rate has been a typical spec for these devices, although recently very fast low cost devices have appeared from Analog Devices and TI that sample up to 80MHz and have bandwidths exceeding 100MHz.

The interface is simple. The data bus goes to the Analog RAM, Spock and MUX. Sample clock is just zz-clk - same as Spock. !STORE drive the !OE pin.

The nominal ADC in the BitScope is the trusty MC10319 from Motorola. This device uses a comparator tree and gray scale decoder, and may be clocked from 25MHz down to DC. This chip is quite old now and is easily outperformed by new CMOS devices. Why design in an outdated chip? It is the footprint for an ADC Module!

ADC Module

The Motorola MC10319 comes in a 24pin 600mil package. It requires all the analog and digital power supplies plus data bus and control signals necessary for any ADC. Rather than have to design a new PCB for every different ADC chip (which are all surface mount now) it seemed a better bet to design for the Motorola part, then make a 24pin 600mil “carrier” PCB for each new Flash ADC of interest. If you care to examine the spec sheets on a few Flash ADCs, you will see they work in much the same way. The only real difference is the input offset and span details. This module scheme is quite flexible. It is even conceivable that a 16 bit ADC module could be devised that outputs odd / even byte pairs - to be descrambled from RAM.

MAXIM to the rescue

You may have noticed that MAXIM make great analog and linear devices. A few years ago, wide bandwidth OP Amps were rocket science. Then came the MAX477. This device has a 300MHz GBP, uses voltage feedback and is happy to drive 50 ohm capacitive loads. As an extra bonus, it has 1 Meg input impedance and negligible input capacitance. This is just the ticket for getting a wide bandwidth signal to an ADC. In addition, MAXIM make superior versions of the venerable 4052 / 4053 analog switch - which have exceptional performance characteristics.

The analog path to our ADC is a series of Wide bandwidth OP Amps forcing the signal through 4:1 analog switches. This combination avoids RC filters which could degrade the high frequency components of our signal.

Vertical Input Buffers

The main source of Analog signals is through CH A and CH B BNC connectors (sheet 5). These input circuits are intended to be compatible with generic 10:1 probes.

S1, S2 provide AC / DC coupling via 100nF caps C32 C34. 1M resistors to GND provide the nominal 1M impedance while R25 / 27, C31 / 33 , D8-11 give input protection.

The high impedance voltage follower circuits are JFETs Q6 / 7 and Q8 / 9. Operating point and offset adjust is set by Q3 / RV3 and Q10 / RV4. JFETs have quite a high output impedance, so MAXIM to the rescue again. U23, U24 unity gain followers buffer the analog signal to the next stage.

Logic POD Analog In

The 3rd and 4th channels of analog input come from the Logic POD. These signals are attenuated by 20K networks R22,23 and R19,20. To compensate for the input capacitance of the 4052 device speedup capacitors C56, C57 are provided. These may need to be trimmed and should have a nominal value of Cin / 4 (about 0-10pF).

The maximum input voltage of U14 is ±3V, so given an attenuation factor of 4.830, this means a maximum analog voltage at the POD of ±15V - suitable for most solid state designs. Higher voltage ranges will need extra circuitry in the POD.

Analog Source

BitScope has 4 Input Channels - 2 from the BNCs and 2 from the Logic POD. These signal may range over ±3V. U17 (4052) selects one signal based on PIC output pin CH-A / B and Spock output pin PG1.

CH-A / B is under dynamic control of the PIC and simply chops the source between BNC A<->BNC B or POD A <-> POD B.

PG1 is set up as an option bit in Spock and may only be changed when Spock is reloaded. PG1 selects either the BNC channels OR the POD channels as the analog source.

Note that PG1 also selects the hi / lo half of both RAMs. This means that we have a completely separate 16K buffer for both Analog and Digital samples for POD source and BNC source!

As a bonus, there is a spare 4:1 MUX in U17. This activates 1 of 4 LEDs during sampling to give Channel Sample indicators at the front panel. As mentioned previously, flashing LEDs are an important feature of any electronic device.

Range Buffer

U14 is a straight 300 MHz unity gain follower that buffers the output of the MUX. The low impedance output of this buffer drives the attenuator section.

3 Resistor networks give attenuation of 1, 2 and 5.273. The other option is a x 4.583 gain stage to boost low level signals. The 4 range options are switched by MUX U18 (4052). This MUX is addressed by RNG0, RNG1 outputs from the PIC.

Depending on the 4052 (MAXIM is best) you may need to add a small speedup cap C69 to the 5:1 range for ideal frequency response. A poor mans trimmer can be made from some adhesive copper foil (stained glass supplier) and a bit of paper.

Gain Buffer

U15 is configured as a (x 4.583) non-inverting amp to boost small signals to feed the ADC. It is possible to use the MAXIM 477 device here, but for a gain of 5 it has only a bandwidth of 25 MHz. The slightly better choice is the Analog devices AD8048 which is optimized for gains of +2 or more. At a gain of 5, it has a bandwidth of better than 50 MHz. For even better performance you may be able to reduce the gain of U15 slightly. The bandwidth improvement will follow 1 / [1+Rf / Rg]. The AD8048 has a unity gain mate - the AD8047 - which is virtually a drop in replacement for the MAXIM 477.

If you use the Analog Devices AD9057 ADC (the pick of the litter) which has a 1 V span, you may halve the gain of this stage - preserving the 100MHz bandwidth across all ranges. This is useful - even though the AD9057 samples at 40 / 60 / 80 MSPS it has an analog bandwidth of 120MHz! This may seem confusing, but will be explained below. There is a bit more to sampling than you might think.

By the way, because this is a 8 bit sample engine, we don't care too much for whole numbers in the gains. The Host display software can sort this out with high precision arithmetic. All the Host needs to know are the constants for each range for a device. There lies a use for some of the 64 bytes of EEPROM in the PIC16F84!

ADC Buffer

After the Range selector, the analog signal is buffered and amplified by U16 - another trusty MAXIM chip. This stage has a gain of 1.667 which takes an input clamped at ±0.6v and outputs a ±1.0v signal. The offset of the output is set by emitter follower Q5 and RV1. Adjusting RV1 will shift the signal span to match the input of the ADC.

The Motorola ADC has a 2V span centered at 1V.

The AD9057 has a 1V span centered at 2.5V

The TI TLC5540 has a 2V span centered at 1.62V

Diode clamps on the input to U16 ensure that the input to the ADC never exceeds 2V p-p. Some ADC chips will not tolerate overvoltage on the input signal. Current limiting resistors R38, R29 ensure that the mighty MAXIM drivers do not exceed the current limits of the Range Select MUX.

The ADC Buffer provides a low impedance drive for the ADC - which may be a few hundred ohms and 30 pF or so. Note that C27 takes off the ac component of the input to the ADC for edge counting in Spock.

RV2 is connected to pin 24 of the ADC to adjust the span. This allows the span to be calibrated if possible for the particular ADC in use.

1GHz Prescaler (Enough is never enough)

A design is finished when either:-

a there is nothing left to add

b there is nothing left to throw out

c both of the above

1GHz prescaler chips (MC12073) are very common and cheap. They are used for PLLs in TV set etc. These chips take a high frequency signal and output a f / 64 square wave. By including a switch (S3) and a 50 ohm resistor it is possible to switch in a prescaler device that drives a spare clock-in on Spock. This can then be routed through to the PIC TOCI pin for counting - giving us a way of frequency measurement up to 1 GHz.

Switch S3 switches in the prescaler with a 50 ohm terminator to BNC ChB. This has other benefits. It means that we can use Channel B as a 50 ohm terminated input.

Idea - Channel B with S3 on could be used as a Ethernet cable terminator with the ability to sample the network traffic for display on a PC.

Sampling

In data acquisition applications there is often some confusion about the relationship between bandwidth and sample rate. The Nyquist rate of Fs / 2 is held to be the maximum frequency that can be captured by periodic sampling at Fs. If so, why would we want an instrument that has a bandwidth of 100MHz and yet samples at a maximum rate only 50 MSPS?

The Nyquist rate applies to continuous time varying signals. In this case, the highest frequency component should be less than Fs / 2 (25MHz@50MSPS) to avoid aliasing.

Repetitive waveforms are a different matter. These are the only high frequency waveforms you will ever see on an analog CRO. The same waveform is re-drawn each sweep and the eye sees a solid trace. Sub-sampling is similar - we use multiple samples and overlay them to build an image.

Providing that the ADC we are using has a wide band-width (does not attenuate the signal) and a small aperture (time window required to freeze the signal level), 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 is known as sub-sampling and is really just another example of the RF Mixer in the digital world.

Sub-sampling has a few limitations. It is not possible to sub-sample a waveform that is harmonically related to the sampling frequency. Practically this means that if the waveform of interest is related to the sample frequency, the sample points will always fall at the same position and the parts between will remain a mystery.

Another problem is resolving the ambiguity of the period of the sub-sampled waveform. Lets say we have a signal of about 28MHz and we are sampling at 40MSPS. What we will see in the sample buffer is a sequence of values with components at 12MHz and 68 MHz. How can these be plotted to build up a profile of the original 28 MHz signal? Well, if we can somehow measure the fundamental frequency of the sampled wave, that will imply period. Since we know the sample rate accurately, we can chop the sample buffer up into segments of N wave periods and then plot them overlaid.

It would be safe to infer from the above discussion that a wide bandwidth DSO should have a facility for frequency measurement on board. Either that or work out how to sample at 1GHz :-) By a stroke of good fortune it happens that the BitScope has provision to measure the frequency of a signal presented to the ADC!

Note that for resolving single event, high frequency pulses (like logic glitches) there is only one solution - *over-sample* by at least a factor of 10. This is the domain of the test equipment companies who apply state of the art circuit techniques to resolve sample to 1nS or better.

Operation

Some operational details of the BitScope are detailed below

Serial Interface

In the 4MHz PIC16F84 version of Picard the baud rate is set at 19200. Faster chips may be scaled up by simply replacing the XTAL. A host program will need to know this - to compensate for different CHOP and timebase calculations.

8MHz - 38,400 12MHz - 57,600

To get to 115,200 Baud will require retuned serial routines, since the fastest PIC currently available is 20MHz.

There is no handshaking for dumped data. The PC must be able to handle the data flow or break it into small chunks with a separate request for each. Fast UARTS have a 16 deep FIFO, so they should be able to handle 16 character chunks at least.

Regaining Control

All BitScope operations - including wait on trigger - may be interrupted by any serial command. The first part of the software UART makes sure that the sample clock is halted. As soon as a serial byte is assembled and echoed, the UART is turned back on and once activated will abort all previous operation. In this sense the BitScope command protocol is truly atomic. Each command is designed to ultimately end in a Halt, if not prematurely aborted.

In addition ASCII code 00 is the reset vector, so it should be possible to get the PICs attention with a <break>.

Voltage Ranges

The BitScope includes 4 internal attenuation ranges, and 4 Channel inputs. Channel A and B are BNC connectors which may have x1 or x10 probes connected. Channel C and D (POD Inputs) have a fixed attenuator - and possibly extra circuitry in the POD. A table of range sensitivities is detailed below. The ranges are not nearly as comprehensive as a Bench CRO, but cover the range most useful to digital and analog circuits. Furthermore, it is intended that the POD connector should deal with unusual or high voltage signals by way of an Active or Smart POD adapter.

It should be possible to alter the gain of some ranges if desired. Since the ADC output is an 8 bit number which 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 will reveal that for a Digital Oscilloscope volts / div and uS / div 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 arbitrary size and grid spacing.

Similarly, the notion of Y offset becomes a display function - nothing to do with the sample engine.

RANGE	BNC x1	BNCx10	POD
00	±130mV	±1.30V	±632mV
01	±600mV	±6.00V	±2.90V
10	±1.20V	±12.00V	±5.80V
11	±3.16V	±31.60V	±15.28V

Table of Input Ranges

ADC Span = 2V

Resistor Attenuators as per circuit

Talking to the POD

If you examine the POD connector, you will see that there are 2 I/O signals that are available when the BitScope is not Sampling. It is intended that these signals form a bi-directional link to another device attached to the POD connector. By establishing a link to the POD, it is possible to make the POD device do some clever things that the BitScope can't. This extends the functionality of the instrument.

I/O Signals

IO-1	Input pulled high by R18	
IO-2	Output	data to active POD

Protocol Pass-Through

This method of POD talking makes the BitScope transparent to the Host machine. A byte to send to the POD is setup in POD_TX register (R18).

Bytecode "|" is executed which echoes the | char, then the contents of POD_TX is sent serially to IO-2. The active POD receives this and may respond via IO-1. Once the byte is transmitted, the PIC connects IO-1 to Serial Out, and will stay that way until aborted by a new command.

This scheme allows a bytecode pass-through! It would be possible to actually connect another BitScope to the POD of the first one. There is a protocol penalty for bytes *from* the Host, but none in the other direction. Subject to serial jitter there is no limit to this recursion.

Example send 'p' followed by 'T' to the POD

```
[12]@[70]s|    set up R1, then 'p'
4d             the response will be piped through
[54]s|         send the POD a 'T'
506f           the response will be piped through
>              we are back talking to BitScope
```

Slow Byte Exchange

A second method of POD control for devices that may not be able to handle high speed serial is a byte exchange protocol.

R18 is POD_Tx, R19 is POD_Rx.

A byte to send is put in POD_Tx as above. Upon execution of Byte command 'x' a software UART (9600) transmits the byte in POD_Tx to IO-2. The UART then monitors IO-1 for a serial byte returned. This byte is put in POD_Rx and then sent to the Host at full speed.

Some POD devices may just need a pulse, so the byte transmitted could be FF, FE, FC, F8, F0, E0, C0, 80, 00.

If no byte is returned by the POD, the next command from the Host will simply abort the UART and proceed with the new command.