VM Specification

The BitScope Next Generation (BSNG) Virtual Machine API Specification

The BitScope virtual machine **firmware** implements the entire functionality of the device (BS10, BS05 etc). It is designed for use with a host side **driver** or for direct VM script level programming. The virtual machine implements a very simple RISC core with a set of *byte code* instructions operating on a set of *byte wide registers*. Registers are arranged in groups for each command and sets of adjacent registers are often used in little-endian order to represent larger 16, 24 or 32 bit values. Modus operandi is to program a (sub)set of registers and issue the command that uses them. *Everything* BitScope does is programmed this way.

Table of Contents

---

1. VM Programming

Programming the BitScope VM is **trivially easy**:

1. Every (ASCII) character sent to the VM is a "executed" as a VM command.
2. Execution is atomic and acknowledged; the VM echoes the same character when execution completes.
3. Some commands reply additional data (after the echo character, see Section 1.1).

All commands execute in the context of a set of **byte wide registers**:

1. Up to 255 registers are available to be used **R0...R255**.
2. The first two registers are special; **R0** is the **Data Register** and **R1** is the **Address Register**.
3. The definition of all the other registers depends on the VM and command being executed.

Registers addresses are expressed in bracketed hexadecimal notation, for example:

R0	**[00]**	Data Register
R1	**[01]**	Address Register
R134	**[86]**	General Purpose Register 134
R201	**[c9]**	General Purpose Register 201

Data is expressed the same way, for example [10] means the number 16 or [da] means the number 218.

1.1 Command Types

Every VM command token is echoed back to the host to acknowledge its execution. Most commands execute and acknowledge within the time it takes to receive the next command (aka the "command window").

This simple protocol means that in most cases no other form of flow control is required because the command echo itself can be used; no "out of band" signalling is required (e.g. no XON/XOFF or RTS/CTS etc). It also means that any octet stream comms link can be used quite easily either in half duplex or full duplex mode. For example, any serial link including RS-232, 1-wire, USB, TCP or UDP can be used from a protocol perspective.

There are some commands that may not execute within the command window (because they have more work to do than can be completed within that time). There are others that produce data after the command echo which must be received and there are others that take an indeterminate period of time to complete (e.g. trace command 'D' which will not complete until a trigger or timeout occurs).

It is therefore important to distinguish between these types of commands and use them appropriately.

Type	Example Commands	Description
Simple	[ ] 0 1 a b c @ s z .	Guaranteed to execute within the command window. May be concatenated into strings without waiting for each command to be acknowledged. For example **[08]@[12]z[34]z[56]z[78]s**
Delayed	X Y Z U >	These commands may take longer than the command window to execute. This means they must appear at the end of the command string. For example **[08]@[12]z[34]z[56]z[78]s>** where the **>** command must not be followed by any other command until its reply is acknowledged.

Complex	T D A S p !	These commands produce additional data following the acknowledge and therefore must only ever appear at the end of a command string *and* their payload data must be received by the host before any further commands are issued to the VM. What the payload data is depends on the command and how it's being used. The host knows this (it programmed the command) and it must act accordingly. So **[08]@[12]z[34]z[56]z[78]s>D** is legal but **[22]@[d2]>DA** is not (because the **D** command must be acknowledged and its payload received before the **A** command can be issued).

1.2 Data Entry

All ASCII characters are commands.

A sub-set of these commands are used exclusively for *data entry*.

Data entry commands "assemble the data" in Data Register R0 as follows:

Entry	**[**	0x5b	Clear R0. Usually commences byte entry (optional)
Entry	**0..9**	0x30..0x39	Left shift R0 4 bits and write the specified digit to the lower nybble.
Entry	**a..f**	0x61..0x66	Left shift R0 4 bits and write the specified hex digit to the lower nybble.
Entry	**]**	0x5d	NOP. Concludes byte entry (optional)

For example, to put the value 193 into the data register R0 simply requires the following command string: **[c1]**

This implicitly results in the [??] notation used for representing data values where ?? are the hex digits of the value.

1.3 Register Programming

Register programming means to assign values to set registers other than R0 and R1.

There are two command defined specifically for this purpose:

RegOps	**@**	0x40	Set Address Register R1.
RegOps	**s**	0x73	Store the value in R0 to register (R1).

When a value is assembled in the data register R0 it may be used to address another register via R1 by:

1. Moving the data register value from R0 to R1 using the **@** command.
2. Assembling the data value in R0 to be programmed to the addressed register R1.
3. Storing this assembled value to the addressed register using the **s** command.

For example to program the value 184 to register R69 simply requires: **[45]@[b8]s**

That is, the modus operandi for register programming is:

[XX]@[YY]s

where XX is the (lower case) hexadecimal register address and YY is the byte value to be stored there.

1.4 Reading Register Values

Most registers are used to control the operation of the device. Once a value is written it does not change. However, other registers may be used to report value which are changed by the VM itself or shared between multiple clients of the VM. In these cases (and for debugging) it is necessary to be able to read register values:

RegOps	**p**	0x70	Print register (R1).

The **p** command prints the value of the register pointed to by R1 as a carriage return bounded hexadecimal value:

<CR>XY<CR>

where XY are two ascii characters representing the register value. The carriage returns are a convenience; it means you can talk to the VM using a standard terminal and read back any register value required, for example:

[45]@p

to read the value of register R69 which would per the earlier example return **<cr>ba<cr>**.

1.5 Multi-byte Registers

Everything needed to program and view register values can be done using commands already described. However, there are a number of commands and shortcuts which can simplify the addressing and programming of VM registers, especially multi-byte registers.

These are documented in the VM Command Set but two of the most useful are:

RegOps	**z**	0x7a	Store the value in R0 to register (R1++).
RegOps	**n**	0x6e	Increment Address Register R1++.

Sets of adjacent registers are used by some VM commands to represent 16, 24 or 32 bit values. They are arranged in little-endian order and the **z** command makes their programming easy. For example, to program the 32 bit value 305,419,896 to the 32 bit register set R80 to R83 can be done as follows:

[50]@[78]z[56]z[34]z[12]s

This can be further simplified in most VM versions because the **[** and **]** commands are optional:

50@78z56z34z12s

The [ ] commands will usually be used throughout the remainder of this document for clarity and portability. They are also often used in driver code to parse command string templates (used to generate command strings).

1.6 Special Commands

There are a few special commands reserved for or used by the VM (in all versions):

Group	Command	Hex	Action
Core	**?**	0x3f	Print a <CR> delimited 8 character string identifying VM revision.
Core	**!**	0x21	Soft VM reset, stop active operations, terminate command sequence.
Core	**.**	0x2e	Terminate a command sequence (with optional context switch).

Use of these commands is optional but **?** is recommended (to confirm the VM version being used) and **!** or **.** to stop the device from active operations and return it to its quiescent (i.e. low power) state when required.

2. BitScope Programming

Chapter 1 describes the universal VM programming techniques required for complete access to everything any BSNG model BitScope can do. The remainder of this document describes the actual register sets, their types and values and the commands that act upon them to program each of BitScope's data acquisition, synthesis and analysis functions.

2.1 Primary Functions

The primary functions of BitScope are:

1. High speed analog and logic waveform capture to internal buffers,
2. Acquisition of previously captured data from those buffers to the host,
3. Lower speed continuous streaming capture to the connected host device,
4. Generation of analog and logic waveforms, voltages and clocks.

Different models offer different performance characteristics but all BSNG devices include these functions.

2.2 Secondary Functions

Most BitScopes offer a range of secondary functions including:

1. Providing power, ground, clocks and control signals to connected circuits,
2. Storing and recalling persistent values from on-board FLASH (e.g. calibration coefficients),
3. Driving various physical indicators including LEDs and control voltages,
4. Communicating and controlling connected Smart POD devices (using VM byte-code),
5. Performing post-capture signal processing such as decimation and filtering.

Not all BitScopes include all of these functions but this is indicated elsewhere in this document.

2.3 Command Groups

Viewed as a generic virtual machine, the register map is simply a set of 256 byte-wide registers where the first two registers (R0 and R1) and a small sub-set of commands have functionality dedicated to the operation of the VM itself.

All the other registers are defined and used in the context of the BitScope commands that perform the waveform capture, generation and acquisition functions of the device. These are managed in *command groups* follows:

Group	Description
Core	Core Virtual Machine functionality (R0, R1, see Chapter 1)
Entry	Data entry commands and registers ([,], 0..9, a..f, see Chapter 1)
RegOps	Register operations to store, print and manage registers (s, p, z, see Chapter 1)
Trace	Triggered and/or frame based high speed waveform and logic capture (D and C)
Stream	Untriggered and/or continuous streaming waveform and logic data capture (T)
Acquire	High speed data acquisition from BitScope to the host (S and A, usually used post Trace)
Generate	Programmable voltage, waveform, clock and logic/protocol pattern generation.
System	System commands that control how the the host communicates with the BitScope.
I/O	Smart POD I/O commands for communicating with connected peripherals.

Flash	Access to BitScope's onboard persistent storage.

2.4 Register Groups

In addition to the command groups, registers may be further classified as follows:

Group	Description
Spock	High speed capture hardware control registers (< and > commands).
Config	Pre-op hardware configuration registers, automatically applied.
Gen	Waveform and logic generator control registers (R, W, X, Y, and Z commands).
Comms	Baud rate control for host communications (only some models).
Update	Pre-op hardware configuration registers requiring the U command.

There are many registers associated with programming each of BitScope's functions so the command and register tables at the end of this document are defined in terms of these command and register groups.

2.5 Simple Example

Using VM10 (BS10) one can control the intensity of the front panel LEDs via their respective registers:

Group	Register	N	A	Description
System	**vrLedLevelRED**	1	[fa]	Red LED Intensity
System	**vrLedLevelGRN**	1	[fb]	Green LED Intensity
System	**vrLedLevelYEL**	1	[fc]	Yellow LED Intensity

To turn the RED LED on to full intensity:

[fa]@[ff]s

To turn it off (note: it will continue to glow very dimly to indicate that BS10 has power applied):

[00]s

and to enable it at low intensity:

[10]s

Note that it's not necessary to reprogram R1 each time, it already has the correct value from the first call. Also, the short-cuts (omitting []) mean turning all three LEDs on can be done as:

fa@ffzffzffs

or even:

fa@ffzzs

and off again as:

fa@00zzs

Note that in this case it is necessary to reprogram the address register because the **z** commands modified it. All register programming is as simple as this. This example simply has results that are immediately visible (the LEDs turning on and off).

3. Waveform Capture (D, K)

The most frequent use for BitScope is the (optionally triggered) capture of analog, logic or mixed signal waveforms.

The process of data capture is referred to as a **trace** and the command to initiate it is **D** (with an optional timeout).

Using the trace command is straightforward. It requires programming a set of registers to establish the sample rate and duration, analog input scaling and offset, trigger condition, pre and post trigger delays, timeout and trace, buffer and (implicit) clock mode. These register settings define exactly what type of data to capture and how to capture it.

After the register values are assigned, the Spock (**>**) and configuration (**U**) commands are normally used to set up the high speed data acquisition hardware and to configure signal ranges and other features for the pending trace.

The **D** command is then issued to commence the trace.

When the trace completes, BitScope returns the buffer sample address and one or more timestamps to report when the trace commenced and when it completed (which depends on specified trigger and/or timeout values).

The whole process typically takes between 2 and 5 ms plus the time it takes to actually capture the signals.

3.1 Trace Setup (D)

The core set of registers used to program a trace are:

Group	Register	N	A	Description
Trace	**vrTraceMode**	1	[21]	Trace Mode (see Trace Mode Table)
Trace	**vrClockTicks**	2	[2e]	Master Sample (clock) period (ticks)
Trace	**vrClockScale**	2	[14]	Clock divide by N (low byte)
Trace	**vrTraceIntro**	2	[26]	Pre-trigger capture count (samples)
Trace	**vrTraceOutro**	2	[2a]	Post-trigger capture count (samples)
Trace	**vrTraceDelay**	4	[22]	Delay period (us)
Trace	**vrTimeout**	2	[2c]	Auto trace timeout (auto-ticks)

The most important register is **vrTraceMode** because this selects the method by which the device will perform the trace. The trace mode automatically selects the clock and buffer modes used which in turn define available sample rates and capture channels. For example the **tmMixedChop** trace mode selects **ckChop** and **bmChopDual** clock and buffer modes. See the trace mode table for the full list of trace modes and associated clock and buffer modes.

When the trace mode is selected the sample rate is programmed via the **vrClockTicks** and **vrClockScale** registers. With reference to the clock mode table, the vrClockTicks register must be programmed with a value within the clock tick limits defined for the selected clock mode. A “tick” is a divisor of the master clock (nominally 40 MHz) to produce the desired sample rate. In some modes this may be further divided via the vrClockScale register.

The sample rate (**Fs**) used for the trace can therefore be calculated as:

Fs = 40 MHz / (vtClockTicks * vtClockScale)

Next the duration of the post trigger capture is programmed via **vrTraceOutro** (expressed in sample units at the programmed sample rate) and (optionally) the pre-trigger capture duration via **vrTraceIntro** (also in samples). The

post trigger duration is the most important as it defines the minimum number of samples the trace will acquire after the trigger. Depending on the trigger event more samples than **vrTraceIntro** samples may be acquired but at least this number is guaranteed to be captured (so long as the sum of both registers does not exceed the buffer size).

If the trace is to start the data capture at *some point in time **after** the trigger event,* the duration of this post-trigger delay may be specified via the **vrTraceDelay** register. This value is expressed in microsecond units (not samples). When a non-zero value is used it means *there will be a discontinuity in the captured data* at the trigger point because capture ceases (for vrTraceDelay microseconds) before resuming (for vrTraceOutro samples).

An optional (auto trace) timeout may also be specified via **vrTimeout** (see Section 3.7 for details).

The best way to understand the steps involved to set up a trace is to work through an example, in this case an analog signal is captured for 1.76mS following a rising edge zero crossing sampled at 1MHz on Channel A:

```
[7b]@[80]s                                  # KitchenSinkA (enable hardware comparators)
[7c]@[80]s                                  # KitchenSinkB (enable analog filter)
[fb]@[00]s                                  # LedLevelGRN (turn CHB LED off, VM10 only)
[fc]@[c0]s                                  # LedLevelYEL (turn CHA LED on, VM10 only)
[37]@[01]s                                  # AnalogAnable (enable CHA input circuits)
[64]@[28]s[65]@[1c]s                        # ConvertorLo (set convertor range, high side)
[66]@[c1]s[67]@[b5]s                        # ConvertorHi (set convertor range, low side)
[14]@[01]s[15]@[00]s                        # ClockScale (set clock prescaler)
[2e]@[28]s[2f]@[00]s                        # ClockTicks (set clock divider)
[31]@[00]s                                  # BufferMode (choose the capture buffer mode)
[21]@[00]s                                  # TraceMode (choose the trace mode)
[22]@[00]s[23]@[00]s[24]@[00]s[25]@[00]s # TraceDelay (set post trigger delay)
[26]@[80]s[27]@[00]s                        # TraceIntro (set pre-trigger capture count)
[2a]@[e0]s[2b]@[06]s                        # TraceOutro (set post-trigger capture count)
[06]@[7f]s                                  # TriggerMask (set the trigger logic mask)
[05]@[80]s                                  # TriggerLogic (program the trigger logic)
[32]@[04]s[33]@[00]s                        # TriggerIntro (set trigger hold-off filter duration)
[34]@[04]s[35]@[00]s                        # TriggerOutro (set trigger hold-on filter duration)
[44]@[00]s[45]@[00]s                        # TriggerValue (set digital trigger level, optional)
[68]@[f5]s[69]@[68]s                        # TriggerLevel (set analog trigger level)
[07]@[21]s                                  # SpockOption (choose edge triggered comparator mode)
[2c]@[00]s[2d]@[00]s                        # Timeout (specify a timeout, “forever” in this case)
[3a]@[00]s[3b]@[00]s                        # Prelude (set the buffer default value; “zero”)
[08]@[00]s[09]@[00]s[0a]@[00]s              # SampleAddress (assign the trace start address)
>                                           # (program capture hardware registers)
U                                           # (programming other hardware registers)
D                                           # commence the trace!
```

3.2 Trigger Setup

Normally the trace will be triggered. The registers used are:

Group	Register	N	A	Description
Spock	**vrSpockOption**	1	[07]	Spock Option Register (see bit definition table for details)
Trace	**vrTriggerIntro**	2	[32]	Edge trigger intro filter counter (samples/2)
Trace	**vrTriggerOutro**	2	[34]	Edge trigger outro filter counter (samples/2)
Trace	**vrTriggerValue**	2	[44]	Digital (comparator) trigger (signed)
Update	**vrTriggerLevel**	2	[68]	Trigger Level (comparator, unsigned)
Trace	**vrTimeout**	2	[2c]	Auto trace timeout (auto-ticks)

First choose the trigger type, source channel, trigger edge and channel swap via **vrSpockOption** register and then the trigger filter parameters via the **vrTriggerIntro** and **vrTriggerOutro** registers. Finally, assign the trigger level via the **vrTriggerValue** (SALT trigger type) or **vrTriggerLevel** (COMP trigger type) registers. If the trace should trigger when no trigger event is seen, put a non-zero value (in auto-ticks, see Section 3.7) in **vrTimeout**.

The values used for the trigger intro and outro define the number of samples the signal must be seen to be FALSE followed by the number of samples it must be seen to be TRUE for a trigger event to be registered. The minimum intro is zero but the minimum outro must be one or more. If the intro is set to zero it means the trigger will occur as soon as it is seen to be true (i.e. it will be a LEVEL trigger not an EDGE) trigger. Recommended default values for both registers is 4. This will ensure that the signal must be in the trigger false condition for at least 4 samples followed by the true condition for another 4 samples which will reject noise and glitches in most cases. For a *hair trigger* use the values 1 and 2. Choose other values as required for longer hold off (intro) and hold on (outro).

3.3 Trace Completion

While tracing and upon completion, the D command returns a series of status packets that look like this:

```
02                                      # stWait (trace started)
41b799c8                                # timestamp
-------------------------------------------------------------------------------
00                                      # stDone (trace complete)
41b8d784                                # timestamp
000007eb                                # address
```

Each packet comprises a status token (e.g. stWait, stDone) followed by one or more payload fields. Each token and all the fields that follow are delimited by carriage return characters.

3.4 Status Packets

The set of all status tokens and their payloads returned by the D (and K) command(s) are:

Name	Token	Payload	Description
stDone	0	timestamp, address	trace complete (normally)
stAuto	1	timestamp, address	trace terminated (timed out)
stWait	2	timestamp	trace in progress (waiting for trigger)
stStop	3	timestamp, address	trace terminated (manually, K command)

The D command always returns **stWait** immediately (to report when the trace *actually* commenced). When the trace completes normally the **stDone** token is returned or if a timeout occurs or the trace is terminated (see Section 3.6) the **stAuto** or **stStop** tokens is returned instead. Every packet reports a timestamp (Section 3.4) and most report a buffer capture address as well (Section 3.5).

3.5 Timestamps

BitScope uses a precision timer when performing all trace operations. The timer is maintained to very high precision (25 ns) regardless of the sample rate used and it is reported (as a timestamp) each time a status packet is returned.

The timer is reported as a 32 bit value in units of 25 ns ticks. It operates continuously and rolls over in epochs of approximately 107 seconds. The host can synchronise its own time of day with the timer via timestamps to reconcile the epoch periodically (once a minute or more often is recommended).

In many test and measurement applications timestamps may be ignored. However, they are very useful if performing multiple high speed episodic waveform captures where all the results need to be precisely time referenced to each other. This is exceptionally useful in data acquisition, remote telemetry and phase coherent monitoring applications.

3.6 Buffer Address (vrSampleAddress)

When performing a trace, acquired signal data is written to a high speed internal capture buffer. The configuration of this buffer (i.e. the number of channels and data types) depends on the selected Buffer Mode (**vrBufferMode**) but in all cases the buffer itself is circular. Acquisition continues "forever" (oldest data is overwritten by the newest) until the specified trigger event, timeout or cancellation event occurs.

The buffer address to which the *next sample* is written is maintained by the 24 bit **vrSampleAddress** register. This register may be manually programmed (normally to start at "zero") but it need not be. It may also be read (via the **vrSampleCounter** register but usually this is usually not necessary either because its value is reported as the address payload in the **stDone**, **stAuto** and **stStop** status packets.

Important: the reported address is the address of the *next sample to be written*. This means upon trace completion this address must be *rewound modulo the buffer size* by the number of post trigger samples requested for capture (via **vrTraceOutro**). This value is then written to **vrSampleAddress** prior to the dump (**A**) command (Section 4).

3.7 Trace Termination (K)

If a trace is initiated which never triggers (either because the trigger condition programmed for the trace is incorrect or the expected event never occurs) the trace will never complete. To prevent this a timeout can be used to force an "auto trace" or the trace can be cancelled manually (and signals captured up until that point may then be acquired).

Trace termination is achieved in one of two ways:

1. A non-zero timeout value is specified via the **vrTimeout** register, and/or
2. the trace is manually terminated using the **K** command.

If a non-zero value is assigned to the **vrTimeout** register a timeout down-counter is activated upon trace start which counts in *"auto ticks"*. When the count reaches zero a timeout event occurs. This is identical to a trigger event in that the trace terminates and the completion data is returned (Section 3.3) but the **stAuto** token is returned instead of **stDone**. A timeout tick period is 6400 nS and **vrTimeout** is 16 bits which supports a maximum timeout of 419 ms.

If a timeout longer than this is required the **K** command must be issued to terminate the trace instead.

4. Data Dumps (A)

Once data has been acquired to the internal buffer (Chapter 3) it may be uploaded to the host. The following sample continues the one started in Section 3.1. In this case we dump 128 samples about the trigger point:

```
[31]@[00]s                          # BufferMode (selected buffer mode)
[08]@[cc]s[09]@[00]s[0a]@[00]s      # SampleAddress (assign the dump start address)
[1e]@[00]s                          # DumpMode (dump mode, raw in this case)
[30]@[00]s                          # DumpChan (channel selected to dump)
[1c]@[80]s[1d]@[00]s                # DumpCount (samples to dump, 128 in this case)
[16]@[01]s[17]@[00]s                # DumpRepeat (number of times to repeat the dump)
[18]@[01]s[19]@[00]s                # DumpSend (points per sample, N/A in this case)
[1a]@[ff]s[1b]@[ff]s                # DumpSkip (points per skip, N/A in this case)
>                                   # (program capture hardware registers)
A                                   # (commence the data dump)
```

4.1 Buffer Mode (vrBufferMode)

The buffer address (**vrSampleAddress**) and the buffer mode (**vrBufferMode**) registers are shared with the trace command (**D**) and perform the same functions (see section 3.5 for details). Once programmed (for a trace) they may not need to be changed for the dump (**A**) when dumping the entire buffer to the host. However, one typically dumps only part of the buffer and so the (start) sample address is normally programmed each time a dump is performed to start at the trigger point or some point before the trigger.

4.2 Dump Mode (vrDumpMode)

BitScope offers a range of dump modes which define how data is uploaded to the host:

Name	Token	Format	Description
dmRaw	0	unsigned 8 bit	raw dump (send/skip ignored)
dmBurst	1	unsigned 8 bit	decimated send burst, skip block (depreciated)
dmSummed	2	signed 16 bit	decimated (skip), summed (send)
dmMinMax	3	unsigned 8+8 bit	decimated (skip), min/max (send)
dmAndOr	4	unsigned 8+8 bit	decimated (skip), and/or (send)
dmNative	5	signed 1.15 bit	raw dump (send/skip ignored)
dmFilter	6	signed 1.15 bit	decimated (skip), decimated & filtered (send)
dmSpan	7	signed 1.15 bit	decimated (skip), min/max (send)

The default and recommended mode is **dmRaw** which is used in the examples. [Other modes documented later].

4.3 Dump Channel (vrDumpChan)

BitScope can trace multiple channels simultaneously but one normally dumps them one at a time.

The dump channel register specifies which of several channels to dump next.

Type	H	Index	Value	Description
Analog	0	0..127	0..127	Analog channel index
Logic	1	0..126	128..254	Logic channel index
Diagnostic	1	127	255	Diagnostic channel

There are only two analog channel indexes (0 and 1) and eight logic (128..135). The others values are reserved for models with different numbers of channels. If used on these models illegal value alias to physical channels.

4.4 Dump Count (vrDumpCount)

The dump count specifies how many (optionally decimated) samples to dump from the device. How many samples are read from the buffer depends on the decimation ratio (see Section 4.6) and may be more than the count but the number of samples returned to the host will always been the count (with optional repeat, Section 4.5)

4.5 Dump Repeat (vrDumpRepeat)

The number of dumps (of size vrDumpCount) may be repeated as specified by **vrDumpRepeat**. Normally this register will be set to one. If more than 2^16 samples are required (i.e. on devices with larger buffers) and/or if a dump needs to be broken down into smaller chunks (e.g. to meet MTU requirements of the host link), this register may be used.

4.6 Dump Send & Skip (vrDumpSend, vrDumpSkip)

The send (vrDumpSend) and skip (vrDumpSkip) registers specify a decimation ratio to apply when dumping data. This is a convenience and bandwidth optimizing feature; it means that only a (useful) data sub-set needs to be

dumped to the host. The data that is actually sent to the host depends on the dump mode (Section 4.2). [To be documented].

5. Data Acquisition (T)

The data acquisition command (T) is the other main mechanism by which BitScope captures data.

In this case data on one or more channels is captured in one of several modes directly to the host without buffering in the BitScope. No trigger or timeouts are required and capture is continuous. The caveat is that in this mode capture is limited to the speed of the link to the host (which is slower than the maximum rates at which BitScope can trace).

6. Waveform Generation (X,Y,Z)

Operating concurrently with waveform capture or data acquisition is the built-in waveform, clock and logic generator.

The modus operandi for waveform generation is to:

1. Y => Synthesize a canonic wavetable (or W => Download a canonic wavetable or logic pattern).
2. X => Translate the canonic wavetable to the replay buffer (or W => download a user waveform).
3. Z => Program the clock and replay parameters and start the generator.

The Y and (sometimes) X steps are optional. Only Z is always required.

All three commands have at least two arguments, **command** (vpCmd) and **mode** (vpMode).

The best way to understand the steps involved is to see an example, in this case a 4kHz sinusoid:

```
[7c]@[40]sU                                 # KitchenSinkB (Enable generator output)

[5a]@[00]s[5b]@[80]s                        # vpSymmetry (Symmetry, 0..100% Y)
[46]@[00]s[47]@[00]sY                       # vpCmd, vpMode, SYNTHESIZE!

[4a]@[e8]s[4b]@[03]s                        # vpSize (Operation size, 1000 samples)
[4c]@[00]s[4d]@[00]s                        # vpIndex (Operation Index, table start)
[4e]@[00]s[4f]@[00]s                        # vpAddress (Destination Address, buffer start)
[54]@[ff]s[55]@[ff]s                        # vpLevel (Output Level, full scale)
[56]@[00]s[57]@[00]s                        # vpOffset (Output Offset, zero)
[5a]@[93]s[5b]@[18]s[5c]@[04]s[5d]@[00]s    # vpRatio (Phase Ratio)
[46]@[00]s[47]@[00]sX                       # vpCmd, vpMode, TRANSLATE!

[48]@[04]s[49]@[80]s                        # vpOption (control flags)
[50]@[28]s[51]@[00]s                        # vpClock (Sample Clock Ticks)
[52]@[e8]s[53]@[03]s                        # vpModulo (Table Modulo Size)
[5e]@[0a]s[5f]@[00]s                        # vpMark (Mark Count/Phase)
[60]@[01]s[61]@[00]s                        # vpSpace (Space Count/Phase)
[78]@[00]s[79]@[7f]s                        # vrDacOutput (DAC Level)
[46]@[02]s[47]@[00]sZ                       # vpCmd, vpMode, GENERATE!
```

6.1 Waveform Generator Registers

The following sub-set of the VM registers are used for waveform, clock and logic pattern generation:

Group	Register	N	A	Description
Gen	**vpCmd**	1	[46]	Command Vector
Gen	**vpMode**	1	[47]	Operation Mode (per command)
Gen	**vpOption**	2	[48]	Command Option (bits fields per command)
Gen	**vpSize**	2	[4a]	Operation (unit/block) size

Gen	**vpIndex**	2	[4c]	Operation index (eg, P Memory Page)
Gen	**vpAddress**	2	[4e]	General purpose address
Gen	**vpClock**	2	[50]	Sample (clock) period (ticks)
Gen	**vpModulo**	2	[52]	Modulo Size (generic)
Gen	**vpLevel**	2	[54]	Output (analog) attenuation (unsigned)
Gen	**vpOffset**	2	[56]	Output (analog) offset (signed)
Gen	**vpMask**	2	[58]	Translate source modulo mask (unused)
Gen	**vpRatio**	4	[5a]	Translate command ratio (phase step S15.16)
Gen	**vpSymmetry**	2	[5a]	Synthesize Symmetry (U0.8)
Gen	**vpRise**	2	[82]	Rising edge clock (channel 1) phase (ticks)
Gen	**vpFall**	2	[84]	Falling edge clock (channel 1) phase (ticks)
Gen	**vpControl**	1	[86]	Clock Control Register (channel 1)
Update	**vrKitchenSinkB**	1	[7c]	Kitchen Sink Register B (Waveform Output Enable, Bit 6)
Update	**vpMap**	1	[99]	Clock Output Enable ([00] => disabled, [12] => enabled)

6.2 Synthesize Command (Y)

Before generating a waveform the wavetable must be created. There are two ways to do this:

1. Use the synthesize command (Y) to create a sine, step, tone or exponential waveform, or
2. Download a user defined wavetable (W) and therefore completely arbitrary waveform.

The waveforms that can be synthesized using the Y command cover all the primary function generator applications; sinusoid, sawtooth triangle and square waves, steps, impulses and exponential decays and ramps. Using the Y command is atomic and instantaneous and is recommended for most waveform and function generation purposes:

vpCmd	vpMode	Function	Size	Bits	Description
0	0	SINE	1024	8	Sinusoidal Waveform
0	1	RAMP	1024	8	Triangle/Sawtooth Waveforms
0	2	EXPN	1024	8	Exponential Waveform
0	3	STEP	1024	8	Square/Impulse Waveforms
0	0	SINE	512	16	Sinusoidal Waveform
0	1	RAMP	512	16	Triangle/Sawtooth Waveforms
0	2	EXPN	512	16	Exponential Waveform
0	3	STEP	512	16	Square/Impulse Waveforms

6.3 Translate Command (X)

6.4 Generate Command (Z) - Waveforms

When the wavetable has been synthesized and translated to the play buffer the generator can started and stopped:

vpCmd	vpMode	Action	Description
1	0	**Stop**	Stop waveform generator and clock.
2	0	**Start**	Start waveform generator and clock.
3	0	**Clock**	Start clock (without waveform generator).

When the waveform generator is running the clock is also running internally. If you wish to generate the clock but not produce any analog waveform, use the **Clock** sub-command instead of **Start** sub-command.

6.5 Generate Command (Z) - Clocks

The clock generator is part of the waveform generator. It may be enabled or not but when the waveform generator is running the clock is used internally to generate the sample clock. When used this way the clock generator is limited to the D/A sample rate, typically 1MHz. When used on its own it can run up to 20MHz (via the **Clock** sub-command).

There are in fact up to three clock channels, all of them running from the same master clock but each of which can be routed to a different output pin and operate at a different phase, e.g. it's easy to produce quadrature phase clocks or generate edge-to-edge logic timing across up to three logic channels.

The modus operandi for clock generation is to:

1. Enable the clock output pin.
2. Set the master clock rate, click rising edge and falling edge phases, and
3. Select a clock source, enable and execute.

The best way to understand the steps involved is to see an example, in this case to generate a 4kHz clock:

```
[99]@[12]sU                                 # vpMap (Enable clock 1 on output L5)
[50]@[10]s[51]@[27]s                        # vpClock (Master clock ticks per period, 20)
[82]@[00]s[83]@[00]s                        # vpRise (Rising Edge at tick 0)
[84]@[88]s[85]@[13]s                        # vpFall (Falling Edge at tick 10)
[86]@[80]s                                  # vpControl (Enable clock source 0)
[46]@[03]s[47]@[00]sZ                       # vmCmd (Command Vector), GENERATE!
```

VM Command Set

Group	Command	Hex	Action
Core	**?**	0x3f	Print the 8 character string identifying the revision.
Entry	**[**	0x5b	Clear R0. Usually commences byte entry (optional)
Entry	**0..9**	0x30..0x39	Increment R0 by the digit specified and nibble swap.
Entry	**a..f**	0x61..0x66	Increment R0 by the hex digit specified and nibble swap.
Entry	**]**	0x5d	Swap nibbles in R0. Usually concludes byte entry (optional)
RegOps	**@**	0x40	Set Address Register R1.
RegOps	**s**	0x73	Store the value in R0 to register (R1).
RegOps	**z**	0x7a	Store the value in R0 to register (R1++).
RegOps	**n**	0x6e	Increment Address Register R1.
RegOps	**p**	0x70	Print register (R1).
Spock	**<**	0x3c	Capture Spock Address to vrSampleCounter
Spock	**>**	0x3e	Program Spock Address from vmSampleAddress
Config	**U**	0x55	Configure device hardware.
Trace	**T**	0x54	Streaming trace/capture mode.
Trace	**D**	0x44	Triggered frame based trace/capture mode.
Trace	**K**	0x4b	Trace cancellation, return sample address and timestamp.
Acquire	**S**	0x53	Sample dump (CSV format, analogue & digital data).
Acquire	**A**	0x41	Analog memory dump (binary format, analog or digital).
Generate	**R**	0x52	Read samples from waveform memory.
Generate	**W**	0x57	Write samples to waveform memory.
Generate	**X**	0x58	Translate buffer inc. amplitude, offset and D/A control bits
Generate	**Y**	0x59	Synthesize waveform (TONE, STEP, RAMP and EXPN) in source buffer
Generate	**Z**	0x5a	Start, Stop or Pause DMA waveform generator output.
Core	**!**	0x21	Soft VM reset, stop active operations, terminate command sequence.
Core	**.**	0x2e	Terminate a command sequence (with optional context switch).
I/O	**~**	0x7e	Set host link baud rate.
I/O	**|**	0x7c	Enable POD I/O receive pipe.
Flash	**r**	0x72	Read from EEPROM.
Flash	**w**	0x77	Write to EEPROM.

VM Register Map

Group	Register	N	A	Description
Core		1	[00]	Input Register
Core		1	[01]	Address Register
Core		1	[02]	Source (Address) Register
Spock	**vrTriggerLogic**	1	[05]	Trigger Logic, one bit per channel (0 => Low, 1 => High)
Spock	**vrTriggerMask**	1	[06]	Trigger Mask, one bit per channel (0 => Don't Care, 1 => Active)
Spock	**vrSpockOption**	1	[07]	Spock Option Register (see bit definition table for details)
Spock	**vrSampleAddress**	3	[08]	Sample address (write) 24 bit
Spock	**vrSampleCounter**	3	[0b]	Sample address (read) 24 bit
Trace	**vrTriggerIntro**	2	[32]	Edge trigger intro filter counter (samples/2)
Trace	**vrTriggerOutro**	2	[34]	Edge trigger outro filter counter (samples/2)
Trace	**vrTriggerValue**	2	[44]	Digital (comparator) trigger (signed)
Trace	**vrTriggerTime**	4	[40]	Stopwatch trigger time (ticks)
Trace	**vrClockTicks**	2	[2e]	Master Sample (clock) period (ticks)
Trace	**vrClockScale**	2	[14]	Clock divide by N (low byte)
Trace	**vrTraceOption**	1	[20]	Trace Mode Option bits
Trace	**vrTraceMode**	1	[21]	Trace Mode (see Trace Mode Table)
Trace	**vrTraceIntro**	2	[26]	Pre-trigger capture count (samples)
Trace	**vrTraceDelay**	4	[22]	Delay period (uS)
Trace	**vrTraceOutro**	2	[2a]	Post-trigger capture count (samples)
Trace	**vrTimeout**	2	[2c]	Auto trace timeout (auto-ticks)
Trace	**vrPrelude**	2	[3a]	Buffer prefill value
Trace	**vrBufferMode**	1	[31]	Buffer mode
Acquire	**vrDumpMode**	1	[1e]	Dump mode
Acquire	**vrDumpChan**	1	[30]	Dump (buffer) Channel (0..127,128..254,255)
Acquire	**vrDumpSend**	2	[18]	Dump send (samples)
Acquire	**vrDumpSkip**	2	[1a]	Dump skip (samples)
Acquire	**vrDumpCount**	2	[1c]	Dump size (samples)
Acquire	**vrDumpRepeat**	2	[16]	Dump repeat (iterations)

Stream	**vrStreamIdent**	1	[36]	Stream data token
Trace	**vrStampIdent**	1	[3c]	Timestamp token
Config	**vrAnalogEnable**	1	[37]	Analog channel enable (bitmap)
Config	**vrDigitalEnable**	1	[38]	Digital channel enable (bitmap)
Config	**vrSnoopEnable**	1	[39]	Frequency (snoop) channel enable (bitmap)
Gen	**vpCmd**	1	[46]	Command Vector
Gen	**vpMode**	1	[47]	Operation Mode (per command)
Gen	**vpOption**	2	[48]	Command Option (bits fields per command)
Gen	**vpSize**	2	[4a]	Operation (unit/block) size
Gen	**vpIndex**	2	[4c]	Operation index (eg, P Memory Page)
Gen	**vpAddress**	2	[4e]	General purpose address
Gen	**vpClock**	2	[50]	Sample (clock) period (ticks)
Gen	**vpModulo**	2	[52]	Modulo Size (generic)
Gen	**vpLevel**	2	[54]	Output (analog) attenuation (unsigned)
Gen	**vpOffset**	2	[56]	Output (analog) offset (signed)
Gen	**vpMask**	2	[58]	Translate source modulo mask
Gen	**vpRatio**	4	[5a]	Translate command ratio (phase step)
Gen	**vpMark**	2	[5e]	Mark count/phase (ticks/step)
Gen	**vpSpace**	2	[60]	Space count/phase (ticks/step)
Gen	**vpRise**	2	[82]	Rising edge clock (channel 1) phase (ticks)
Gen	**vpFall**	2	[84]	Falling edge clock (channel 1) phase (ticks)
Gen	**vpControl**	1	[86]	Clock Control Register (channel 1)
Gen	**vpRise2**	2	[88]	Rising edge clock (channel 2) phase (ticks)
Gen	**vpFall2**	2	[8a]	Falling edge clock (channel 2) phase (ticks)
Gen	**vpControl2**	1	[8c]	Clock Control Register (channel 2)
Gen	**vpRise3**	2	[8e]	Rising edge clock (channel 3) phase (ticks)
Gen	**vpFall3**	2	[90]	Falling edge clock (channel 3) phase (ticks)
Gen	**vpControl3**	1	[92]	Clock Control Register (channel 3)
System		1	[10]	EE Data Register
System		1	[11]	EE Address Register

Comms		1	[3d]	start cell time
Comms		1	[3e]	bit cell time
Comms		1	[3f]	baud rate
Comms		1	[12]	POD Tx Byte
Comms		1	[13]	POD Rx Byte
Update	**vrConverterLo**	2	[64]	VRB ADC Range Bottom (D Trace Mode)
Update	**vrConverterHi**	2	[66]	VRB ADC Range Top (D Trace Mode)
Update	**vrTriggerLevel**	2	[68]	Trigger Level (comparator, unsigned)
Update	**vrLogicControl**	1	[74]	Logic Control
Update	**vrDacOutput**	2	[78]	DAC (rest) level
Update	**vrKitchenSinkA**	1	[7b]	Kitchen Sink Register A
Update	**vrKitchenSinkB**	1	[7c]	Kitchen Sink Register B
Update	**vpMap0**	1	[94]	Peripheral Pin Select Channel 0
Update	**vpMap1**	1	[95]	Peripheral Pin Select Channel 1
Update	**vpMap2**	1	[96]	Peripheral Pin Select Channel 2
Update	**vpMap3**	1	[97]	Peripheral Pin Select Channel 3
Update	**vpMap4**	1	[98]	Peripheral Pin Select Channel 4
Update	**vpMap5**	1	[99]	Peripheral Pin Select Channel 5
Update	**vpMap6**	1	[9a]	Peripheral Pin Select Channel 6
Update	**vpMap7**	1	[9b]	Peripheral Pin Select Channel 7
System	**vrMasterClockN**	1	[f7]	PLL prescale (DIV N)
System	**vrMasterClockM**	2	[f8]	PLL multiplier (MUL M)
System	**vrLedLevelRED**	1	[fa]	Red LED Intensity (VM10 only)
System	**vrLedLevelGRN**	1	[fb]	Green LED Intensity (VM10 only)
System	**vrLedLevelYEL**	1	[fc]	Yellow LED Intensity (VM10 only)
System	**vcBaudHost**	2	[fe]	baud rate (host side)

SpockOption [07]

Bit Field	Definition	Values
6	Trigger Invert	0 => normal, 1 => invert
2	Trigger Source	0 => CHA, 1 => CHB

1	Trigger Swap	0 => normal, 1 => swap upon trigger
0	Trigger Type	0 => sampled analog, 1 => hardware comparator

Range Programming [64] [66]

VM10	**vrConvertorLo**	**vrConvertorHi**
520 mV	[64]@[54]s[65]@[65]s	[66]@[96]s[67]@[6c]s
1.1 V	[64]@[47]s[65]@[61]s	[66]@[a2]s[67]@[70]s
3.5 V	[64]@[86]s[65]@[50]s	[66]@[64]s[67]@[81]s
5.2 V	[64]@[a7]s[65]@[44]s	[66]@[42]s[67]@[8d]s
11 V	[64]@[28]s[65]@[1c]s	[66]@[c1]s[67]@[b5]s

VM05	**vrConvertorLo**	**vrConvertorHi**
1.1 V	[64]@[d6]s[65]@[65]s	[66]@[bc]s[67]@[69]s
3.5 V	[64]@[62]s[65]@[52]s	[66]@[3f]s[67]@[7d]s
5.2 V	[64]@[68]s[65]@[44]s	[66]@[ff]s[67]@[8a]s
11 V	[64]@[6a]s[65]@[12]s	[66]@[8c]s[67]@[ba]s

KitchenSinkA [7b]

Bit Field	**Definition**	**Values**
7	Channel A Comparator	0 => disabled, 1 => enabled
6	Channel B Comparator	0 => disabled, 1 => enabled

KitchenSinkB [7c]

Bit Field	**Definition**	**Values**
7	Analog Filter Enable	0 => disabled, 1 => enabled
6	Waveform Generator Enable	0 => disabled, 1 => enabled

LogicControl [74]

Bit Field	**Definition**	**Values**
7	Logic Buffer Direction	0 => inbound, 1 => outbound

Map [94]..[9b] Peripheral Pin Select

Value	**Token**	**Definition**	**Description**
0	NULL	Logic Input	No peripheral attached to the pin
1	C1OUT		Comparator1 Output (not used)

2	C2OUT		Comparator2 Output (not used)
3	U1TX	UART1 Tx	UART 1 transmit
4	U1RTS	UART1 RTS	UART 1 Ready To Send
5	U2TX	UART2 Tx	UART 2 transmit
6	U2RTS	UART2 RTS	UART 2 Ready To Send
7	SDO1	SPI1 D/O	SPI 1 Data Output
8	SCK1	SPI1 CLK	SPI 1 Clock Output
9	SS1	SPI1 SSO	SPI 1 Slave Select Output
10	SDO2	SPI2 D/O	SPI 2 Data Output
11	SCK2	SPI2 CLK	SPI 2 Clock Output
12	SS2	SPI2 SSO	SPI 2 Slave Select Output
13	CSDO	DCI SDO	I2S/AC97 Serial Data Output
14	CSCK	DCI SCO	I2S/AC97 Serial Clock Output
15	COFS	DCI FSO	I2S/AC97 Frame Sync Output
16	C1TX	CAN1 Tx	CAN1 Transmit
18	OC1	Clock 1	Clock Channel 1 Output
19	OC2	Clock 2	Clock Channel 2 Output
20	OC3	Clock 3	Clock Channel 3 Output
21	OC4	Clock 4	Clock Channel 4 Output (A/D Clock)
128	ASCL1	I2C Clk	I2C Clock Input/Output (D6 Only)
129	ASDA1	I2C Data	I2C Data Input/Output (D5 Only)
130	ANALG	Analog Input	Analog Input Selected (D0..D3 Only)

Trace Modes (D)

Trace Mode	M	Type	Mode	Trig	Clock Mode	Buffer Mode	1/N
tmAnalog	0	analog	single	mask	ckMixed	bmSingle	yes
tmAnalogFast	4	analog	single	mask	ckMixedFast	bmSingle	no
tmAnalogShot	11	analog	single	mask	ckMixedShot	bmSingle	no
tmMixed	1	mixed	single	mask	ckMixed	bmDual	yes
tmMixedFast	5	mixed	single	mask	ckMixedFast	bmDual	no
tmMixedShot	12	mixed	single	mask	ckMixedShot	bmDual	no
tmLogic	14	logic	single	mask	ckLogic	bmSingle	no
tmLogicFast	15	logic	single	bit	ckLogicFast	bmSingle	no
tmLogicShot	13	logic	single	mask	ckLogicShot	bmSingle	no
tmAnalogChop	2	analog	chop	mask	ckChop	bmChop	yes
tmAnalogFastChop	6	analog	chop	mask	ckChopFast	bmChop	no
tmAnalogShotChop	16	analog	chop	mask	ckChopShot	bmChop	no
tmMixedChop	3	mixed	chop	mask	ckChop	bmChopDual	yes
tmMixedFastChop	7	mixed	chop	mask	ckChopFast	bmChopDual	no
tmMixedShotChop	17	mixed	chop	mask	ckChopShot	bmChopDual	no
tmMacro	18	analog	single	mask	ckMacro	bmMacro	no
tmMacroChop	19	analog	chop	mask	ckMacro	bmMacroChop	no

Buffer Modes (D)

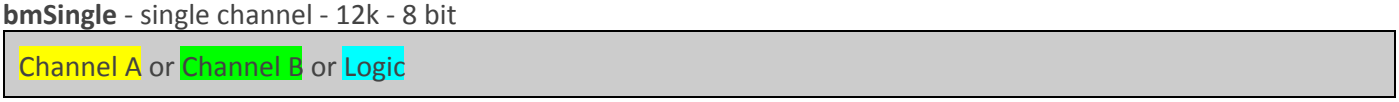
The buffer mode defines the layout and addressing of the buffer memory used for data capture.

The buffer mode is selected automatically (via the chosen trace mode) when performing a trace (see Chapter 3) but it must be explicitly selected (via the **vrBufferMode** register) when performing a dump (see Chapter 4). It's possible (but not recommended) to dump data in a different buffer mode than which it was traced (advanced use only).

For each buffer mode the address range, layout and trace modes that select it are shown in this table:

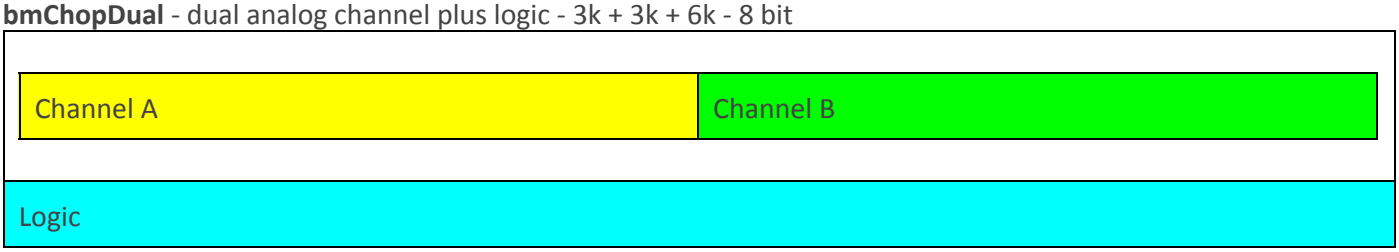
Mode	M	A	Units	Buffer	N	Trace Modes
bmSingle	0	12k	Full	12k	1	tmAnalog, tmAnalogFast, tmLogic, tmLogicFast
bmChop	1	12k	Semi	6k+6k	2	tmAnalogChop, tmAnalogFastChop,
bmDual	2	6k	Full	6k+6K	2	tmMixed, tmMixedFast, tmMixedShot
bmChopDual	3	6k	Both	3k+3K+6k	3	tmMixedChop, tmMixedFastChop, tmMixedShopChop
bmMacro	4	6k	Full	6k	1	tmMacro
bmMacroChop	5	6k	Semi	3k+3k	2	tmMacroChop

bmSingle - single channel - 12k - 8 bit

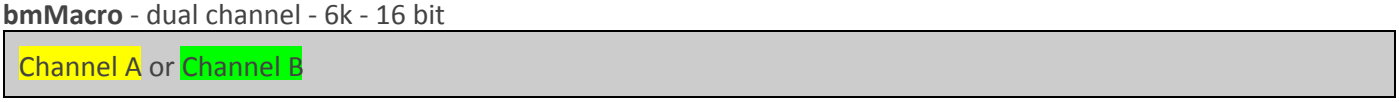


bmChop, **bmDual** - dual channel - 6k - 8 bit

bmChopDual - dual analog channel plus logic - 3k + 3k + 6k - 8 bit



bmMacro - dual channel - 6k - 16 bit



bmMacroChop - dual channel - 3k - 16 bit

Clock Modes (D)

Mode	**M**	**Ticks** (Low)	**Ticks** (High)	**High** (MHz)	**Low** (MHz)	**1/N**	**Trace Modes**
ckMixed	0	15	40	2.66	61 Hz	1 ... 16k	tmAnalog, tmMixed
ckMixedFast	1	8	14	5.00	2.9	1	tmAnalogFast, tmMixedFast
ckMixedShot	2	2	5	20.00	8.0	1	tmMixedShot, tmAnalogShot
ckLogic	3	5	16k	8.00	2.4 kHz	1	tmLogic
ckLogicFast	4	4	4	10.00	10.00	1	tmLogicFast
ckLogicShot	5	1	3	40.00	13.3	1	tmLogicShot
ckChop	6	13	40	3.08	61 Hz	1 ... 16k	tmAnalogChop, tmMixedChop
ckChopFast	7	8	40	5.00	1.0	1	tmAnalogFastChop, tmMixedFastChop
ckChopShot	8	4	5	10.00	8.00	1	tmAnalogShotChop, tmMixedShotChop
ckMacro	9	10	64k	1	600 Hz	1	tmMacro, tmMacroChop

Stream Modes (T)

Trace	M	Type	Channels	Framed	Bits	Comment
tmSteamAny	00	Test	one	Special	8 bit	Link speed test mode
tmStreamAll	01	Stream	all	Yes	8 bit	General purpose mixed signal
tmStreamRaw	02	Stream	one	No	8 bit	Single 8 bit channel, fastest
tmStreamOne	04	Stream	one	Yes	12 bit	Single 12 bit channel (on-board)
tmStreamTwo	03	Stream	two	Yes	12 bit	Dual 12 bit channel (on-board)

Generator Modes (Z)

Cmd	V	Mode	M	Operation
zvStop	1	zmStop	0	Stop Generator (reset phase, mute output)
zvStop	1	zmPause	1	Pause Generator (hold output)
zvPlay	2	zmStart	0	Start Generator
zvPlay	2	zmContinue	1	Continue Generator
zvPlay	2	zmOneShot	2	Play Generator (one shot)
zvClock	3	zmClock	0	Generate clock signal