Mark IV Decoder Protocol

RJ Cappallo 2004.11.18

The Mark IV decoder protocol for serial communications is based loosely upon that of the Mark IV formatter. It consists of single word commands followed by 0 or more parameters. Only a few of the initial characters in the command word need to be specified (e.g. one can use the abbreviations **du** or **dump** for the **dump_buffer** command). In order to facilitate control from either a dumb terminal or the Field System's MAT command, commands can be terminated by a carriage return (ASCII 0x0d), a dollar sign (ASCII 0x24), or a percent symbol (ASCII 0x25). The same terminator which appears on a command will be used in the decoder's response to that command. Note also that a leading slash, which characterized Mk4 Formatter commands, is not used.

Arguments: Optionally, any command can be issued with no arguments, in which case a status message will be returned. The status message will consist of the arguments last used for the command, perhaps along with some extra parameters of interest. A few commands (e.g. **dqa**) appear only in the status form – that is, they never have arguments, and don't affect the state of the decoder hardware.

If arguments are present with a command, they must agree in number and in type with what is expected; exceptions will generate an error 02. Numeric arguments can only be given in the format expected: either decimal (dec) or hexadecimal (hex), depending on the command. The parameters in the status message will have the same number base as is expected in the command. String arguments are denoted by (str). If arguments have a well-defined range, it will be specified in the syntax description, along with the type.

Case-folding: All string arguments, including the command itself, are converted internally to lower case before parsing. (hex) numbers are also case-insensitive. Thus, the user can feel free to use upper or lower case, at their discretion.

Errors: If for some reason the decoder is unable to correctly process a command, an error code, along with a brief message, will be returned. A list of error messages can be found at the end of this document.

MAT: Since the MAT bus is a "party line", there will generally be messages directed at many other devices; these messages are ignored by the decoder. The decoder firmware starts listening whenever a "#91" (the decoder's MAT address) is detected, and stops listening at the first "#" followed by something other than the two characters "91".

COMMANDS:

address sets the address to be used for a subsequent **read** or **write** command (see).

Syntax: **address <location>,** where **<location>** (hex) is an accessible address either in RAM, CPU space, or referring to a memory-mapped hardware register.

Example: **address 84206** will cause subsequent **read** or **write** commands to access the error status register, or the LED write register, respectively.

Response: **address <location>**

auxilliary_data returns the most recent auxilliary data from both decoders.

Syntax: **auxilliary_data** (there are no parameters)

Example: **aux**

Response: **auxilliary_data <aaux> <baux>** where **<aaux>** and **<baux>** (str) are in the same format as the front panel display of the decoder: aaaa bbbb rrss ttuu. Note: In the Mark 4 format aaaa is the head1 position in microns (for negative head positions, it is the absolute value + 4000), bbbb is the head2 position, rr is two bits(7-6) of headstack# and 6 bits(5-0) of track#, ss identifies the data stream: 2 fanout sample identity (7-6), S/M bit(bit 5) = 0/1, USB/LSB bit (bit4) = 0/1, and VC# (minus one) in bits 3-0, tt holds formatter status, and uu is the system ID.

bocf_period sets the period of the bocf generator, which in turn drives the correlator chip. The bocf period is the accumulation period for phase cal extraction, state counting, and spectral analysis. The BOCF generator hardware limits the BOCF length to be n x 125K samples, where n can take on integral values from 1..256. Thus the maximum BOCF period is 32 Msamples, which takes one second at the highest data rate of 32 Msamp/s. The new BOCF period won't take effect until a subsequent **pcal** or **samples** command is issued.

Syntax: **bocf_period <n>**, where **<n>** (dec, 1..256) specifies the number of 125K blocks.

Example: **bocf 16** (*sets the bocf period to 2,000,000 samples)*

Response: **bocf_period <n>**

capture defines and initiates the capture of data into the data buffer. There are two different modes of capture. The first has as its source the A decoder (c.f. apar below) channel; parity bits are captured along with every 8 data bits, and the capture starts on the first decoded 10 second boundary. The second capture mode applies to all other sources. Parity bits are either edited out by the capture hardware in the case of anop & bnop, or never exist within the sampler outputs (usbx, etc.). In this mode the capture will occur on the next 1PPS tick after the capture command is received. In the future, the capture software will be upgraded to allow the user to specify the time at which the capture is to commence.

Syntax: **capture <source> <decimator> <# blocks>**, where **<source>** (str) is one of {apar,anop, bnop, usbx, lsbx, usby, lsby}, indicating, respectively, decoder A with parity bits, decoders A or B with parity bits removed, and USB or LSB sampler outputs from the Mk4 Formatter X or Y channel. **<decimator>** (dec; 1..256) is a value specifying that every **<decimator>**th sample be stored. The first sample stored is the first data sample after the end of the BOCF, which is 256 (pre-decimation) samples after the second tick. Note that decimator is not applicable when the source is apar. The size of the capture in 1KB blocks is given by **<# blocks>** (dec | str). The value **all** requests that all available memory (generally 62 MB) be used.

Example: **capture bnop 2 all** causes the subsequent capture of the channel B decoder output, with parity bits stripped, and every other bit discarded, using all available RAM.

Response: **capture <source> <decimator> <# blocks> <status> <time>** The additional arguments are: **<status>** (str) is one of {i, a, c, d} representing {idle, armed, capturing, done} respectively, and **<time>** (str) is the time of the first captured sample, in the same format as the front panel display of the decoder: yddd hhmm ss.sss. Note: the time is captured from the A decoder stream, so for the time to make sense in any mode other than apar, it is necessary

to ensure that a valid decodable stream comes into the A decoder, probably via bypass mode. In version 3.0 of coda the returned time will be that time read from DQA A at the time of receipt of command. The capture time will then be on the next 10 or 1 second (depending on the source) boundary.

dqa controls and reads back the error statistic counters from the data quality analyzer. The parity error counts are accumulated for the specified number of frames, and are displayed on the front panel. The remotely accessible counters run continuously. The dqa's can be configured to accept either Mk4 or VLBA tape frame formats. Since the VLBA mode was added as an after-thought, the hardware is unable to calculate the 16 bit CRC, and the CRC counts should be ignored.

Syntax: **dqa <# frames** or "clear" or **"**mk4" or **"**VLBA"**>** where **<# frames>** (dec) is the number of tape frames over which the parity error count is tabulated, for front panel LCD display purposes only. **dqa clear** causes all counters to be cleared at the end of the next frame received. **dqa mk4** or **dqa VLBA** sets the data quality analyzers to work in the corresponding mode.

Example: **dqa 400**. Since tape frames contain 2500 bytes, a value of 400 would result in the number of parity errors per million bytes.

Response: **dqa <# aframes> <aparity> <anosync> <aresync> <acrc> <# bframes> <bparity> <bnosync> <bresync> <bcrc>** all fields are in (hex). All counters are cumulative: they report the total number of parity errors, or counts of frames with a specific problem, since the last **dqa clear** command was issued (they are not reset after a **dqa** status message).

dump_buffer causes a data buffer dump out of the 115 Kbaud serial port (7 data bits, 2 stop bits, even parity) to commence. The data buffer memory is in RAM above the program space, and currently is a maximum of about 62 MB. After a **capture** command the data will sit in the data buffer until the next **capture** command, and can be fully or partially dumped and re-dumped at will.

Syntax: **dump_buffer <begin_block> <# blocks>** where **<begin_block>** (hex) is the relative block number of the desired data, with 0 referring to the physical beginning of the data buffer. **<# blocks>** (dec) specifies the total number of blocks to be dumped. A block is defined to be 1K (1024) bytes. A request for 0 blocks causes the immediate cessation of any currently active data buffer dump. A request with **<# blocks>** set to the value "all" will request that all currently stored data should be downloaded.

Output format: The ASCII-encoded blocks are 2056 bytes (characters) long, with the following structure:

- 5 byte decimal block number
- colon (":", 0x3a)
- 2048 hex characters representing 1024 bytes
- carriage return (0x0d)
- line feed (0x0a)

Examples: **dump 0 all** requests that all currently stored data be dumped.
dump 0 0 aborts the current dump.

Response: **dump_buffer <begin_block> <# blocks> <current_block> <total_blocks>** The first two arguments are specified above in the syntax section; the additional arguments are: **<current_block>** (dec) is the block # of the block currently being dumped; **<total_blocks>** (dec) is the number of blocks currently in the capture buffer.

pcal sets up the parameters governing phase cal extraction. It can come in four different varieties, depending on the number of signal sources and the number of tones to be extracted. If a formatted signal is being received by either decoder a or b (likely from a tape drive or Mk5 disk), only the first form of the command may be used, and a single tone extracted. The response to the command **pcal**, with no arguments, depends on which of the four pcal modes is currently in effect.

1x1

Syntax: **pcal <source> <freq> <rate>** where **<source>** (str) is one of {a, b, usbx, lsbx, usby, lsby} for decoder a or b, or one of the formatter sampled sideband signals. **<freq>** (dec) is the phase cal tone frequency in Hz (for now, integer values only are accepted). **<rate>** (str) is the sample rate in Ms/s, and must be one of the following values: {.125, .25, .5, 1, 2, 4, 8, 16, 32}.

Example: **pcal a 10000 4** specifies decoder A, 10 KHz tone freq., and 4 Ms/s sample rate.

Response: **pcal <source> <freq> <rate> <amp> <phase>** where **<amp>** (dec) is the phase cal amplitude extracted during the last BOCF in units of Whitneys (1 Whitney = 10^{-3} correlation amplitude in voltage units). **<phase>** (dec) is the corresponding phase in degrees in the interval of (-180..180).

1x8

In order to extract 8 tones simultaneously, it is necessary to use the signals coming directly from the samplers. With one input signal, all 8 tones are specified by:

Syntax: **pcal <source> <f1> <f2> <f3> <f4> <f5> <f6> <f7> <f8>** where **<source>** is one of {a, b, usbx, lsbx, usby, lsby}. The tone frequencies **<fn>** are specified in (integer) Hertz. Note that if the signal source is either decoder channel A or B, the sample rate should be set previously via a 1x1 command, since it isn't specified in this (1x8) command.

Example: **pcal usby 10000 1010000 2010000 3010000 990000 17 1234567 3141592**

Response: **pcal <source> <f1> <a1> <p1> <f2> <a2> <p2> … <f8> <a8> <p8>** where the frequency, amplitude, and phase of each of the 8 tones are given in the same units as specified in the 1x1 response.

2x4

With two input signals, the 8 tones are specified by:

Syntax: **pcal <source1> <f1> <f2> <f3> <f4> <source2> <f5> <f6> <f7> <f8>** where each **<sourcen>** is one of {usbx, lsbx, usby, lsby}. The tone frequencies **<fn>** are specified in (integer) Hertz.

Example: **pcal usby 10000 1010000 2010000 3010000 lsby 990000 1990000 2990000 3990000**

Response: **pcal <source1> <f1> <a1> <p1> … <f4> <a4> <p4> <source2> … <f8> <a8> <p8>** where the frequency, amplitude, and phase of each of the 8 tones are given in the same units as specified in the 1x1 response.

4x2

With four input signals, the 8 tones can only be from 4 different frequencies, and are specified by:

Syntax: **pcal <source1> <source2> <f1> <f2> <source3> <source4> <f3> <f4>** where each **<sourcen>** is one of {usbx, lsbx, usby, lsby}. The tone frequencies **<fn>** are specified in (integer) Hertz. Note that 2 tones are extracted per source, e.g. both **<f1>** and **<f2>** are extracted from **<source1>**, as well as from **<source2>**.

Example: **pcal usbx usby 10000 3010000 lsbx lsby 990000 3990000**

Response: **pcal <source1> <f1> <a1> <p1> <f2> <a2> <p2> <source2> <f1> <a3> <p3> <f2> <a4> <p4> <source3> … <f4> <a8> <p8>** where the frequency, amplitude, and phase of each of the 8 tones are given in the same units as specified in the 1x1 response.

read allows the user to read an arbitrary 16 bit value from the accessible address space. Either RAM or a memory-mapped hardware register can be inspected in this fashion. The address must have previously been set up with an **address** command (see).

Syntax: **read** (no arguments)

Example: **address 8420E** *(refers to a particular location in a Xilinx register)*

read *(reads a hardware status register)*

Response: **read <address> <value>**, where **<value>** is the hex value that was read.

samples is used to measure the sampler statistics of a selected data stream, which can be useful as a diagnostic for detecting problems in the level settings, or AGC function. The selected signal is analyzed for occurrences of each of the four different 2-bit data patterns. Note that decoded streams a and b are intrinsically single bit streams, so only two of the four states will be populated. Also keep in mind that the correlator chip is used to count the samples in each of the four states, so pcal extraction and sample statistics are mutually exclusive.

Syntax: **samples <source>** where **<source>**, which is a string, is one of {“a”, “b”, “usbx”, “lsbx”, “usby”, “lsby”} for decoder a or b, or one of the formatter sampled sideband signals.

Example: **samples usby**

Response: **samples <source> <n00> <n01> <n10> <n11>** where **<source>** is the same as in the samples command, and **<n00>**, etc. are the number of samples (dec) in that particular state.

status returns the current status of the decoder hardware and software.

Syntax: **status** (there are no parameters)

Example: **status**

Response: **status <flags>** where the bits within **<flags>** are defined as follows:

Bit 0: Spurious Interrupt – a spurious interrupt has occurred since status last read
Bit 1: NYI
Bits 2-15: NYI

time returns the most recent time from both decoders.

Syntax: **time** (there are no parameters)

Example: **time**

Response: **time <atime> <btime>** where **<atime>** and **<btime>** (str) are in the same format as the front panel display of the decoder: yddd hhmm ss.sss.

write allows the user to write an arbitrary 16 bit value into the accessible address space. Either RAM or a memory-mapped hardware register can be written to in this fashion.

Syntax: **write <value>** , where **<value>** (hex) is from 1..4 valid hex digits

Example: write B305 writes the 16 bit pattern 1011001100000101 to the current address (see **address**).

Response: **write <location> <value>** where **<location>** (hex) is the address at which the value was written.

ERROR CODES

Syntax: **error <nn> <message>** where **<nn>** (dec) is a 2 digit number, and **<message>** (str) is a brief message describing the error

List of error codes:

error 01 unknown command – command word not recognized

error 02 wrong number of arguments – too many / too few arguments for this command

error 03 illegal argument type – at least one of the argument values is the wrong type (e.g. a string in a field which should be numeric)

error 04 argument out of range – an argument has an impossible value

error 05 illegal address – an address has been specified which is either odd or out of range