Moving window deconvolution module

Introduction:

The moving window deconvolution (MWD) module despite implementing simple calculations has relatively high complexity due to the use of fixed point arithmetic and need for time synchronisation between parallel calculation pipes. Furthermore the moving window deconvolution result energies are normally read back using ‘fast’ readout mode which is an addition to the base FEBEX firmware. This document explains the calculation flow, number formats and delays through the MWD hardware. For more details on setting up fast readout mode see ‘FEBEX4A Trigger Matrix V3’.

Brief notes on Q format:

Q format is used to denote numbers with a ‘virtual’ binary point in fixed point arithmetic systems. E.G the value “0101” in a conventional binary representation would be decimal 5, if this where in Q2.2 format this would be 1.25. From the left the value of each binary place is 2^{1}, 2^{0}, 2^{-1}, 2^{-2} . By keeping track of the Q format we are using for values in our digital signals processing, padding and selecting appropriate bits we can use binary multipliers to perform division. Rounding (floor) can be performed by truncating fraction bits. When numbers are represented as signed twos complement I have elected to explicitly denote they are signed and count the sign bit as a Q value. For example direct conversion of an unsigned Q16 value without additional positive range would be to a Q17 (signed) value as the first bit is used by the sign bit.

Notes on custom float16 format used to export waveforms:

The FEBEX waveform memory is only 16 bits wide, therefore unless multiple data words are used there must be some loss of precision to store numbers greater than 2^{16}-1. The ‘T’ waveform that is viewed to adjust the moving window deconvolution parameters is internally a 35 bit signed number. This cannot be sent to the trace memory directly as the trace memory is 16 bit wide and runs at 100MHz. Therefore a custom 16 bit floating point format has been designed to store the ‘T’ waveform in the trace memory. This format has the following specification (from MSB to LSB):

- 1 sign bit (‘1’ indicates negative number, ‘0’ positive
- 5 bit power of 2 exponent with no pre-bias (fractional numbers cannot be represented)
- 11 bit significand (implicit 1 allows extra bit)

- 35 bit input word is truncated to remove 3 least significant bits before conversion into the 16 bit float

Example conversions:

1000 => 0x63D0
-1000 => 0xE3D0
0 => 0x0000

VHDL code (simulation) to convert custom floating point back to 35bit signed:

```
221 ¦ --Convert output float to real number again
222 Pregen : PROCESS(export_T)
223 ¦ variable sign_bit : std_logic;
224 ¦ variable signif : unsigned(9 DOWNTO 0);
225 ¦ variable exp : unsigned(4 DOWNTO 0);
226 ¦ variable bit_store : std_logic_vector(34 DOWNTO 0);
227 ¦ variable bit_store2 : std_logic_vector(34 DOWNTO 0);
228 ¦     BEGIN
229 ¦         sign_bit := export_T(15);
230 ¦         exp := unsigned(export_T(14 DOWNTO 10));
231 ¦         signif := unsigned(export_T(9 DOWNTO 0));
232 ¦         bit_store(34) := '0'; --we will encode sign later
233 ¦         bit_store(33) := '1';
234         if ((signif = to_unsigned(0,signif'length)) and (exp = to_unsigned(0,exp'length))) then
235 ¦           bit_store(33) := '0'; --special zero case
236         end if;
237 ¦       bit_store(32 DOWNTO 23) := export_T(9 DOWNTO 0); --fraction
238 ¦       bit_store(22 DOWNTO 0) := (OTHERS=>'0'); --set all remaining bits to zero
239         if (sign_bit = '1') then
240 ¦           --need to take into account sign  (invert all bits and add 1 to find complment)
241 ¦           bit_store2 := std_logic_vector(shift_right(unsigned(bit_store),to_integer(exp))); --shifts done by exponent
242 ¦           regen <= std_logic_vector(unsigned(not(bit_store2))+to_unsigned(1,bit_store2'length));
243 ¦       else
244 ¦           --twos complement of unsigned is the same as signed
245 ¦           regen <= std_logic_vector(shift_right(unsigned(bit_store),to_integer(exp))); --shifts done by exponent
246         end if;
247 END PROCESS Pregen;
```

Illustration 1: VHDL code to convert custom 16 bit floating point format to signed number, note expanded version that takes into account special values in the appendix.

the input 16 bit word is export_T. Firstly all the bit fields are read. Secondly the special zero case is handled, this is detected by a zero exponent and a zero significand. The exponent then applies a bit shift to the the significand which has its bits flipped and 1 added if its negative. A special value that no number will encode can be encoded with -0 (0x8000). Certain special values can be encoded to display events like triggers on waveforms, details can be found in the section ‘Visualisation processing of T waveform’ of this document.

The maximum and minimum value that can be encoded is +/- 2^{31} * 1.999023437 = 4292870144. The maximum value of a 32bit signed number (truncated 35 bit) is 2147483647 and minimum -2147483648 so there is a large range of values that will never be used by the waveform available for special signal encoding (E.G triggers, sample points).

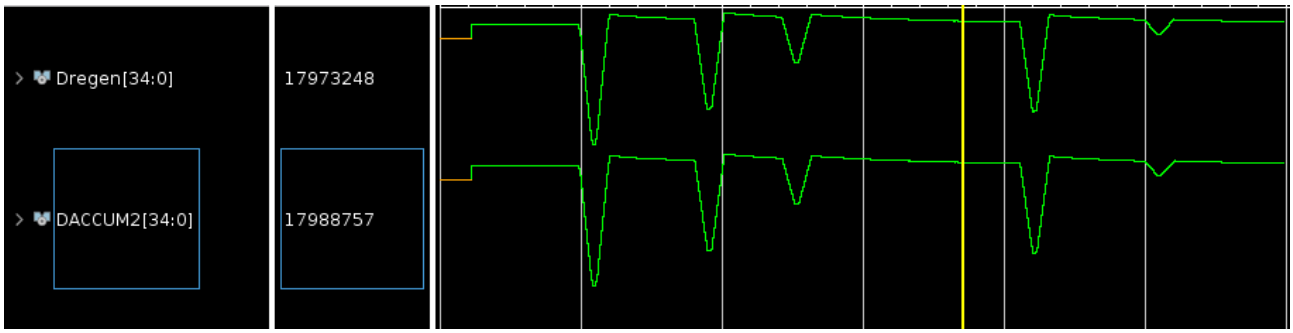

Illustration 2: Comparison of regenerated from 16 bit float waveform (Dregen) and original 35 bit signed number

Example of conversions:

To aid debugging listed are example conversions. All values represented as hexadecimal.

35 bit signed number (internal)	**16 bit float**	**Regenerated 35 bit signed number**
0000003e8	63d0	0000003e8
7fffffc18	e3d0	7fffffc18
000000000	0000	000000000
3ffffffff	03ff	3ff800000
400000008	83ff	400800000
4005b8d88	83ff	400800000

Notes on CRC calculation:

The final 16bit word of the data packet is a 16bit CRC value of the preceding 96bits (0xA5A5 header not included). This CRC conforms to the CRC16-CCITT specification:

- Width = 16 bits
- Truncated polynomial = 0x1021
- Initial value = 0xFFFF
- Input data is NOT reflected
- Output CRC is NOT reflected
- No XOR is performed on the output CRC

more details on this can be found at: https://srecord.sourceforge.net/crc16-ccitt.html

In the appendix is a MATLAB/OCTAVE implementation of the CRC using a lookup table. The hardware implementation generates identical results to this implementation.

Data format and delivery:

The results of the moving window deconvolution are placed in the ‘dummy’ trace buffer. The MWD peripheral constantly generates a trigger signal for readout. In fast read out mode only this trigger signal is enabled to read out the ‘dummy’ trace buffer only (see trigger matrix V3 document). The fast trigger mode replaces the normal triggering mode whereby CFDs or global triggers cause acquisition of trace data. When more then 64 words are in the trace buffer data is returned for a read request, otherwise zero length data is returned. Up to 8190 words can be sent.

The MWD units for each channel are addressed in the ‘round robin’ fashion whereby each unit is checked in turn and serviced if a result is ready. This constrains the maximum time to service a MWD module. If a result is ready its recorded in a buffer as ‘packet’ of 8, 16 bit words. . If readout does not occur before buffer full the MWD modules become blocked and will miss events. New events are rejected while the old event is retained. The data is stored in the trace buffer (8192 16 bit words long) in the following format:

Word	**Format (16 bit words)**
W0	0xA5A5
W1	[channel]&“000”&[pile up flag]& Timestamp(55 DOWNTO 48)
W2	Timestamp(47 DOWNTO 32)
W3	Timestamp(31 DOWNTO 16)
W4	Timestamp(15 DOWNTO 0)
W5	Uenergy(31 DOWNTO 16)
W6	Uenergy(15 DOWNTO 0)
W7	CRC-16

Channel is stored as a 4 bit unsigned number giving the channel of the result (the CFD of the channel that has triggered to generate the result). Pile up flag is a bit that is set if pile up is detected during the MWD process. The time stamp is the FEBEX time stamp for the event and is given as a 56 bit unsigned number. The energy is a 32 bit unsigned value generated by an absolute function and truncation (see Uenergy_shift register parameter).

CRC-16 is the 16bit CRC of W1-W6 inclusive (96 bits). This is an example computation of the CRC of “a87827a02469addc61a97d5a” :

loop: 1, data = 0, rom_addr = FF, crc_reg = E1F0
loop: 2, data = 0, rom_addr = E1, crc_reg = 1D0F
loop: 3, data = A8, rom_addr = B5, crc_reg = F87E
loop: 4, data = 78, rom_addr = 80, crc_reg = EF88
loop: 5, data = 27, rom_addr = C8, crc_reg = D044
loop: 6, data = A0, rom_addr = 70, crc_reg = 3A97
loop: 7, data = 24, rom_addr = 1E, crc_reg = 64FF
loop: 8, data = 69, rom_addr = D, crc_reg = 2EAD
loop: 9, data = AD, rom_addr = 83, crc_reg = CEB

loop: 10, data = DC, rom_addr = D0, crc_reg = 207D
loop: 11, data = 61, rom_addr = 41, crc_reg = 25E5
loop: 12, data = A9, rom_addr = 8C, crc_reg = B504
loop: 13, data = 7D, rom_addr = C8, crc_reg = 5C44
loop: 14, data = 5A, rom_addr = 6, crc_reg = 24C6
The test word CRC is: 24C6

W0 is used as a header word to ensure alignment. If data corruption is detected this can be used to skip ahead in the data to the next valid packet.

RC1 global trigger time stamp packets:

Using the options register it is possible to enable a special packet to be generated for each global trigger received on the EXPLODER trigger input 1 (NIM type). This feature can be used to check that the FEBEX boards in the system have the same times tamp counter value at the same time instant. If using to check time stamp offset a very slow clock or user controlled pulse should be used to drive the trigger input 1 to reduce the probability of an edge arriving before all boards have been enabled. For these packets there is a small change in format:

Word	**Format (16 bit words)**
W0	0xA5A5
W1	“0000”&“001”&“0”&Timestamp(55 DOWNTO 48)
W2	Timestamp(47 DOWNTO 32)
W3	Timestamp(31 DOWNTO 16)
W4	Timestamp(15 DOWNTO 0)
W5	0xFFFF
W6	0xFFFF
W7	CRC-16

The CRC-16 is of 96bits W1-W6 inclusive.

Padding facility:

Sometimes the DMA data received from a FEBEX module is left or right shifted by one 32bit word, this can corrupt the start and end packets. To mitigate this a padding facility has been introduced. When turned on (using the options register) this adds two 16bit words of 0x0000 to the start and end of data sent over GOSSIP such that the received DMA data has a 32bit 0x00000000 at the start and end of each modules data. This is an example of padded data with no inadvertent left or right shift:

Raw Data:
0xff001934 0x00000088 0x00000000 0x0000a5a5 0x9be4000d 0x36136d63 0xb3b7192e
0x0000a5a5 0xb922000d 0x360f5ef8 0x530c9c78 0x0000a5a5 0xb923000d 0x3610e598
0x934fd23d 0x0000a5a5 0xb925000d 0x360c6c38 0x4645ac47 0x0000a5a5 0xb926000d

0x3611f2d7 0xa6122b18 0x0000a5a5 0xb928000d 0x360f7977 0xc9cfd298 0x0000a5a5 0xb92a000d 0x36110017 0x0963e0e7 0x0000a5a5 0xb92b000d 0x360f86b7 0x00008cb3 0x00000000

The padding words have been highlighted. Note the padding words have been included in the 0x00000088 data length parameter as they are GOSSIP data.

Design:

Moving average calculation (Delay 7 clocks):

This hardware calculates a moving average of trace values (achan), implementing the MATLAB code:

MA(loop) = sum(Waveform((loop-M):loop-1))/torr; %Moving average

where loop is the loop index incrementing over the waveform length. Note that Wavefom(loop-1) is the first sample acted on by this calculation and so there is a sample delay required relative to the derivative calculation.

Firstly the trace data is accumulated, this has a delay of one clock cycle due to the final unit delay in the accumulator been used as a pipeline register.

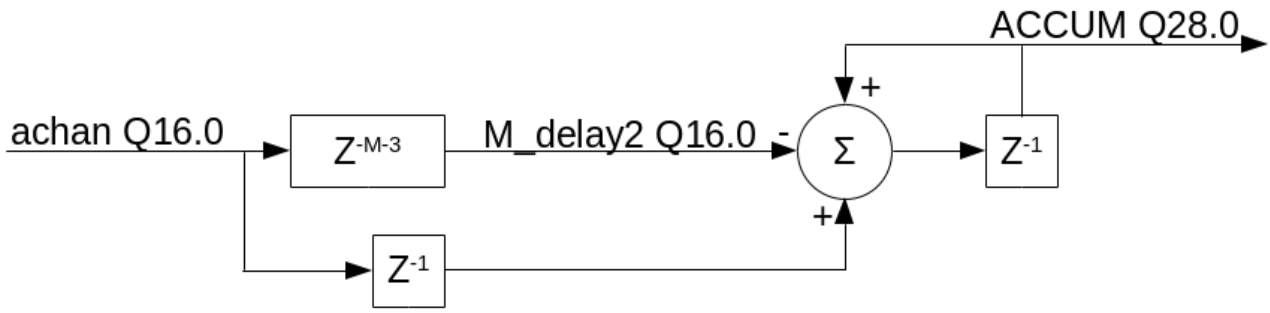

Illustration 3: Calculation of sample accumulator

next the division by torr is accomplished using a multiplier which has a delay of 6 clock cycles. By padding achan and torr with differing numbers of zeros leading zeros pre-devision by 4096 is accomplished on ACCUM and torr represents only the factional part of the Q16.16 input words to the divider.

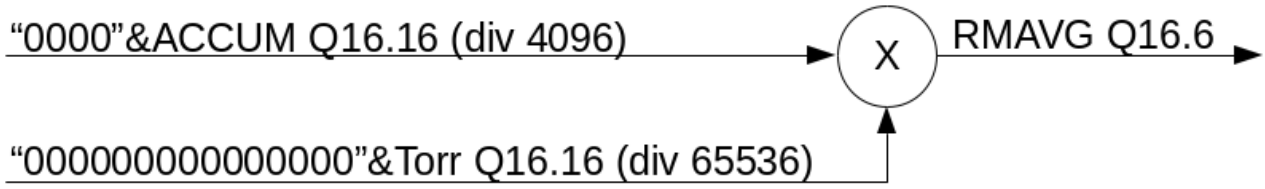

Illustration 4: Calculation of moving average

The value torr is the preamp time constant in clock cycles to divide by. FEBEX uses a 100MHz sample clock. For example if the preamp had a time constant of 200µS this would be a divide by 20,000 clock cycles, we have an existing pre-divider of 4096 so we actualy need to divide by 4.883 clock cycles. This is equivalent to multiplication by 0.2048, in Q16.16 format this is 0000000000000000.0011010001101110 (torr_calcs.ods shows how to perform these calculations).

Numerical differentiation calculation (Delay 7 clocks):

This hardware calculates a numerical derivative implementing the MATLAB code:

D(loop) = Waveform(loop)-Waveform(loop-M); %Numerical differentiation

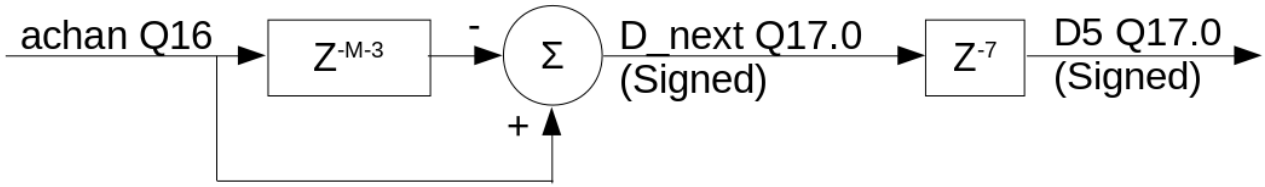

Illustration 5: Numerical derivative calculation

A 7 cycle delay is used to align the result with the RMAVG waveform for which should have synchronized results with this module for the MWD waveform calculation.

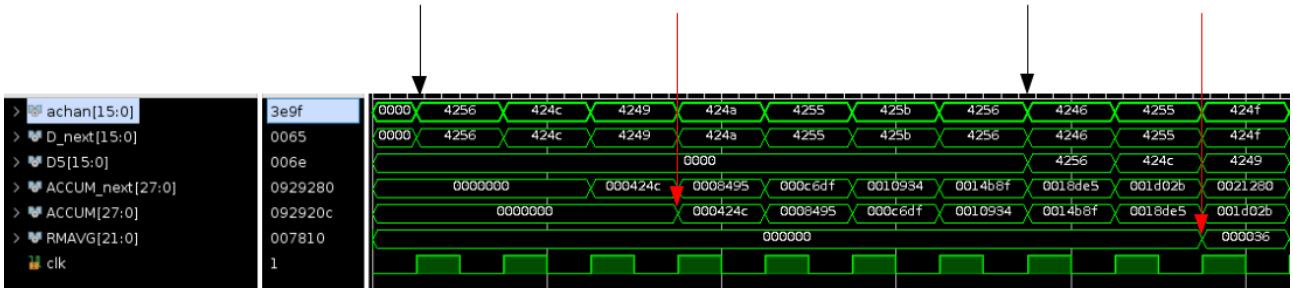

Illustration 6: Delay verification at startup for differentiation calculation alignment (D5) with moving average (RMAVG). Black lines show time to first data (7 clocks) for derivative and red for MAVG (6 clocks) however ACCUM has an extra cycle delay from ACCUM_next balancing the delays at 7 clocks. The startup of ACCUM is delayed W.R.T RMAVG to avoid corrupt data entering the accumulator where it cannot be cleared (mainly simulation issue).

MWD waveform calculation (1 clock delay):

This hardware calculates:

MWD(loop) = D(loop)+MA(loop);

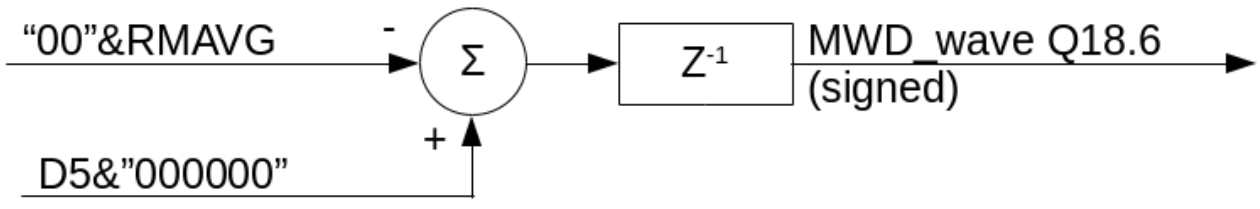

Illustration 7: MWD waveform calculation

The delay to this point is now 8 clock cycles relative to the input waveform (achan).

Calculation of ‘T’ wave (1 clock delay)

This calculates:

T(loop) = sum(MWD(loop-L:loop-1))/L; %Moving average

however the final division by L has been omitted as this is a common factor in the energy calculation and so only acts as a scaling factor on the final energy value that can be removed in software.

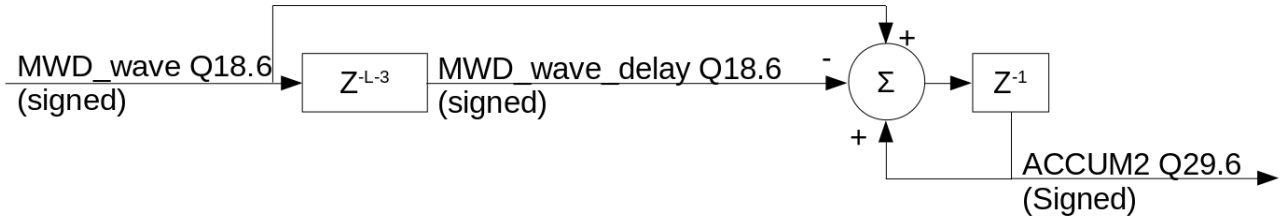

Illustration 8: T' wave calculation

ACCUM2 is is sampled to determine the energy along with the baseline. This waveform displayed by the end user as a trace to adjust the MWD parameters, in this process triggers are (optionally) overlayed on the ‘T’ waveform. This waveform is 9 clock cycles behind the trigger signal and so a delay is required.

Visualisation processing of T waveform (2 clock delay):

To adjust the moving window deconvolution parameters it is required to display the ‘T’ waveform with triggers and energy sampling points overlay. A barrier to this is that the waveform is Q29.6 (signed) requiring 35 bits to display it, however the FEBEX only has 16 bit trace memory buffers. A solution to this problem is to use a custom 16 bit floating point format which has been defined in ‘Notes on custom float16 format used to export waveforms:’ in this document. Generating this floating point number takes 2 clock cycles using the module ‘S35_to_float16’.

The module ‘S35_to_float16’ performs all possible bit shifts to generate the significand in parallel and then selects the smallest bit shift that results in the implicit ‘1’ of the significand been set. This would only take a single clock cycle but a pipe line register has been used on the output of the significand selector to increase timing flexibility.

For visualisation of trigger points the options register is set to 1010xxxx. This causes the internal ‘T’ waveform of the channels moving MWD unit to be stored in that channels trace buffer with special floating point values outside of the normally used range:

- Trigger special value: 0xEFFF
- Energy sample point special value: 0xFFFF

Used to show trigger points and energy sample points.

A 11 clock cycle delay line is used to delay the trigger signal for visualisation and a 2 cycle delay line for the energy sampling such that the sample they overwrite on the waveform is the sample they are acting upon. There is no waveform value available when a special value is used and so its recommended that the previous waveform value is displayed.

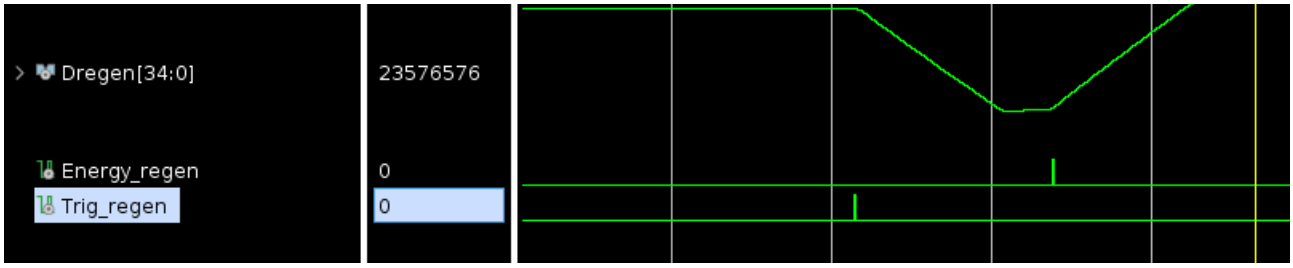

Illustration 9: Regenerated to signed number floating point waveform with extraction of trigger and energy sampling points. Also using previous value hold technique to disguise missing samples

Calculation of baseline (no clock delay):

The baseline must be subtracted from the peak values to determine the energy. The baseline is calculated by blanking ACCUM2 during events; during blanking the pre-blanking value of ACCUM2 us retained until the blanking period is completed. The blanking period is M+L+6+extra_blank where extra_blank is a user set value 0 to 4095 clock cycles long (upto 40.95µS) that can be used to extend blanking to eliminate tail effects. The +6 is to take into account that the M and L values have +3 added to them by the operation of the delay lines in earlier calculations.

To ensure that the blanking is aligned with the ACCUM2 waveform the blanking is started by a 9 clock delayed trigger signal.

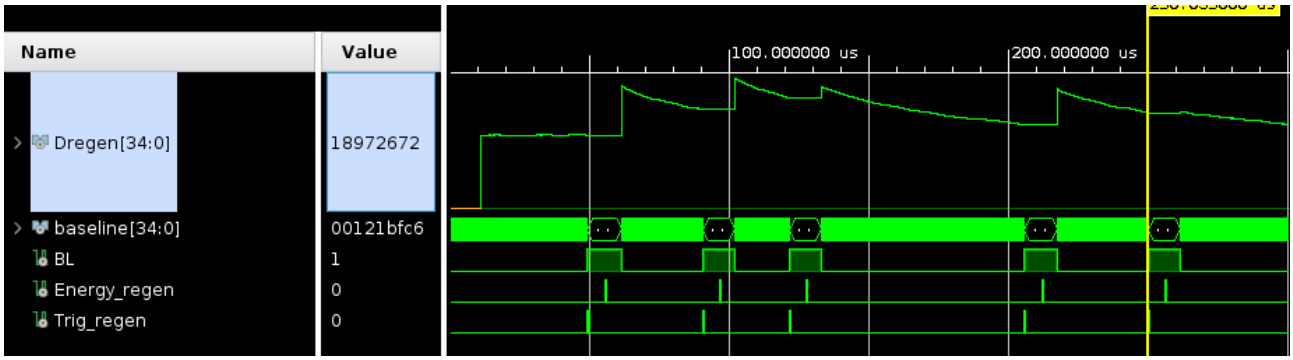

Illustration 10: Illustration 10: Baseline waveform selected as trace showing blanking (when BL='1')

Operation:

Addressing:

Only one register is used to communicate with the trigger peripheral with the GOSIP address 0x200030. The writes to this register are 32 bits with the most significant byte used to distinguish

which sub register the write is for and the nibble below this used to select the channel (labelled [chan]). The least significant bits are used for the data payload.

If settings are needed to be read back the same address is written to but with the most significant bit set (E.G M becomes 1000 0001). To read a write should be undertaken and then a subsequent read of the register 0x200030 will have the correct data payload. (similar to how the SPI read/write is performed (see memory map)).

The channel is selected using the 3rd most significant nibble. Channels are numbered 0 through to 15 (16 total).

Parameter	**Data word (32bit, x= don't care)**	**Format**
M	0000 0001 [chan] xxxx xxxx [data] [data] [data]	12 bit unsigned
L	0000 0010 [chan] xxxx xxxx [data] [data] [data]	12 bit unsigned
Torr	0000 0011 [chan] xxxx [data] [data] [data] [data]	16 bit unsigned
Extra blank	0000 0100 [chan] xxxx xxxx [data] [data] [data]	12 bit unsigned
Options	0000 0101 [chan] xxxx xxxx xx[data] [data] [data]	11 bit logic and unsigned
cfd_trig_delay	0000 0110 [chan] xxxx xxxx [data] [data] [data]	12 bit unsigned
uenergy_shift	0000 1010 [chan] xxxx xxxx xxxx xxxx xx[data]	2 bit unsigned
Test mode	0000 1011 xxxx xxxx xxxx xxxx xxxx xx[data]	2 bit
Cross trigger	0000 1100 [chan] xxxx [data] [data] [data] [data]	16 bit logic
data_len	1000 1101 xxxx xxxx xxxx xxxx xxxx xxxx	Write to read only
MCNT_REG	0000 1110 [data] [data] [data] [data] [data] [data]	24 bit unsigned
GPON	0000 1111 xxxx xxxx xxxx xxxx xxxx xxx1	1 bit

Write example:

In this example the M value is set to 500 on channel 15, this involves a data payload of 497 as M values have 3 added to them internally.

goc -w -x 1 0 [address] 0x01F001F1

Read example:

In this example the cfd_trig_delay is read from channel 1

goc -w -x 1 0 [address] 0x86100000

goc -r -x 1 0 [address]

0x00000111

Detailed description of input parameters:

M:

length of step signal from exponential tail (samples), note actual value in module is +3 on input value. I.E input of 0 results in 3. This is due to the delay line logic. Measured in clock cycles at 100MHz, I.E input value of 497 results in M value of 500 which is 5µS. This value is unsigned and the maximum is 4095 (4098 effective).

L:

Length of moving average (L < M to get trapizoid) (samples), note actual value in module is +3 on input value. I.E input of 0 results in 3. This is due to the delay line logic. Measured in clock cycles at 100MHz, I.E input value of 497 results in L value of 500 which is 5µS. This value is unsigned and the maximum is 4095 (4098 effective).

Torr:

The value torr is the preamp time constant in clock cycles to divide by. FEBEX uses a 100MHz sample clock (10nS granularity). For example if the preamp had a time constant of 200µS this would be a divide by 20,000 clock cycles, we have an existing pre-divider of 4096 so we actualy need to divide by 4.883 clock cycles. This is equivalent to multiplication by 0.2048, in Q16.16 format this is 0000000000000000.0011010001101110 (torr_calcs.ods shows how to perform these calculations).

1. Calculate number of clock cycles to divide by: α=round(100e6 * torr[s])
2. Calculate value to encode right of decimal point taking into account pre-divider: β= 1/(α*4096)
3. Calculate the decimal Torr value: Torr = round(2^16*β) this is the value (in hex) that is sent to set the register on FEBEX.

Extra_blank:

This is the extra number of clock cycles to blank the baseline after a trigger. If this is set to 0 the blanking time is M+L+6 (to take into account the extra 3 added to each value). This can be used to eliminate tail effects from the baseline estimate.

Options:

This register is used to configure multiple parameters related to waveform display from the moving window deconvolution blocks.

D10: ‘1’ turns on RC1 global trigger time stamp packets in response to trigger RC1 (input 1 of EXPLODER)

D9: (pad_reg) ‘1’ turns on the data padding facility.

D8-D7: (WaveSel) “10” makes trace data be a test pattern to diagnose data loss in the trace data-path , “01” makes the trace data be moving window deconvolution data (16 bit custom floating point format), “00’ makes the trace data be normal ADC data (16 bit uinsigned raw ADC values).

D6: (TorB) ‘0’ sets the floating point waveform export to the ‘T’ waveform while ‘1’ sets it to the baseline waveform. This setting is overwritten by Read_mwd.

D5: (mark_sp) When this option is set ‘1’ and read_mwd =’0’ the exported floating point waveforms will have trigger and energy sampling points encoded onto them which allows tuning of the MWD parameters. See ‘Visualisation processing of T waveform’.

D4: (Read_MWD) When set to ‘1’ the trace exported from the MWD module is set to an internal 16bit signed value that contains the MWD waveform amplified by the magnification factor. This setting has priority over TorB.

D0 – D3: The magnification factor of the readout of the internal MWD waveform if this is selected using Read_MWD=’1’. The magnification factor is in binary places and is used to increase the resolution of the visualisation of the MWD waveform at the expense of reducing the clipping point of this waveform. It is not anticipated that users will want to view this waveform.

E.G “10100000” is: WaveSel = ‘1’, TorB = ‘0’, mark_sp = ‘1’, Read_MWD = ‘0’, MAG = “0000” this will results in a 16bit floating point ‘T’ waveform been put into the trace buffers with the trigger and sample points marked.

cfd_trig_delay:

cfd_trig_delay : number of clocks after the trigger signal to sample the energy. Note in the case of pileup the 1st event causes the energy readout (subsequent events do not restart countdown) this is to avoid very high rates causing no data to be written. Using the option mark_sp and a pulsar this value can easily be tuned to the correct point on the trapezoids. (See illustration 9)

Push_thresh:

number of words that when reached by recording energy events causes a trigger and pushing from the energy ring buffer to the dummy register. This should be set to greater than 26 and less than or equal to 8190 words . The default value is 4095 words in order for the trigger to be generated before the buffer is full (8190 words) avoiding dead-time during the time that the readout computer services the trigger.

timeout:

32 bit Timer which will cause a push to the dummy register and trigger if insufficient events have occoured not reaching the push_thresh before timeout. Counts in number of clock cycles, max is ~42s, can be disabled by setting to all bits to ‘1’ any lower value than this causes timeout timer functionality with one bit worth 1/100E6 seconds. The value is loaded with two words an upper word and a lower word.

uenergy_shift:

This value is default zero and is unlikely to need to be changed. The unsigned energy value is internally 35 bits long, however it's read out as a 32bit value. The default behaviour is that the 32 bits are the lower bits with the upper 3 bits discarded as its unlikely energy values will fill even the full 32 bits. If energy values are 'clipped' uenergy_shift provides a facility to left shift the energy value by up to 3 places such that the lower bits are instead truncated. E.G a value of 1 will result in bits 32 DOWNTO 1 to be selected.

Test mode:

Test mode has four valid settings:

"00": off

"01": Causes the moving window deconvolution modules to be disconnected from the logic that drives and dummy trace buffer. Instead of MWD data packets packets are generated at a rate of default rate 1kHz, variable using MCNT_REG with the following format:

Data Word:	Contents:
W0	0xA5A5
W1	0xDEAD
W2	0xBEAF
W3	Packet Count
W4	0xDEAD
W5	0xBEAF
W6	0xAAAA
W7	0x5555

The MWD modules are still operating but will become blocked as the data readout is no longer servicing them. CFD setting does not effect data readout.

Packet count is an incrementing auto reset to 0, 16 bit counter that increments every time a data packet is stored. Using test mode it should be possible to diagnose any data loss or corruption events.

"10": Causes the moving window deconvolution modules to be disconnected from the logic that drives and dummy trace buffer. Instead of MWD data packets packets are generated at a rate of default rate 1kHz, variable using MCNT_REG with the following format:

Data Word:	Contents:
W0	0xA5A5
W1	"000000000000000" LFSR33(32)
W2	LFSR33(31 DOWNTO 16)
W3	LFSR33(15 DOWNTO 0)

W4	0xDEAD
W5	0xBEAF
W6	0xAAAA
W7	0x5555

The MWD modules are still operating but will become blocked as the data readout is no longer servicing them. CFD setting does not effect data readout.

LFSR33 is a 33 bit linear feedback shift register constructed according to Xilinx XAPP201 with a repetition period of 8589934591 cycles. A new value is generated for each data packet when test mode is “10”. The reset and first value is 0x000000000.

“11”: resets the LFSR33 to 0x000000000. Data is still generated at the rate set by MCNT_REG as per the “10” setting but with an LFSR33 value of 0x000000000. Switching between “11” and “10” can be used to reset the LFSR for synchronisation with a software verification routine.

Cross trigger:

This allows the CFD of one channel to start the MWD process on that channel and any other channel. For example if the core of a detector fires it may be desired to read out all segments even if their CFDs have not fired. To perform this there is a register for each channel forming a matrix:

	T15	**T14**	**T13**	**T12**	**T11**	**T10**	**T9**	**T8**	**T7**	**T6**	**T5**	**T4**	**T3**	**T2**	**T1**	**T0**
C15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C14	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C13	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
C12	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
C11	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C10	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
C9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
C8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
C7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
C6	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
C5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
C4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
C3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
C2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
C1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
C0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Each channel has a register whereby setting a bit will enable the respective cross trigger. This table shows the default values whereby each channel only triggers itself. Channels will always trigger

themselves even if the relevant bit is clear. If it is desired to stop a channels own trigger its CFD should be disabled.

Example: if C15 was the core and we wanted to always readout C1, C2, C3, C4 MWD energy on it firing we would set C15 to “1000 0000 0001 1110”, this would be accomplished by writing 0x0CF0801E to the MWD register (0x200030). Note that these registers only effect MWD operation and will have no effect if traces are been captured.

data_len

This is a write-to-read only register that returns the last data_len parameter of how many bytes where written to the GOSSIP DPM memory by dummy_memsend.vhd. This is for debug purposes.

MCNT_REG

Register that sets the frequency of test mode data packet generation. The value sets the reset point for the internal counter in clock cycles. Each reset generates a test packet. The clock frequency is 100 MHz. The default value of 100,000 results in data generation at 1kHz. The maximum value is 167,772,15 resulting in data generation of 5.96 Hz. This has no effect if test mode is disabled.

GPON

Set this bit to enable padding mode whereby all GOSSIP reads will be padded with 0xFFFF words to a length of 8184 16bit words (largest number of words with complete 8 word packets).

Input parameter default values:

The registers are loaded with these default values when power on reset is performed in order to aid with configuration. Note MIDAS may/will overwrite these values. These values are intended as starting points for a correct configuration of the MWD peripheral. These values are applied to all channels.

Parameter	**Default value**
M	597
L	447
Torr	“0011010001101110”
extra_blank	110
pad_reg (options)	‘0’
Mag (options)	2
read_MWD (options)	‘1’
mark_SP (options)	‘1’
TorB (options)	‘0’
Pad (options)	‘0’
cfd_trig_delay	1050

push_thresh	4095
uenergy_shift	0
Timeout	0xFFFFFFFF (disabled)
wave_sel	‘0’ (options) (ADC data)
Test Mode	‘0’
Cross trigger	0x0000
MCNT_REG	0x186a0

Appendix 1: float to real signed number conversion

```
--Convert output float to real number again
Pregen : PROCESS(export_T)
variable sign_bit : std_logic;
variable signif : unsigned(9 DOWNTO 0);
variable exp : unsigned(4 DOWNTO 0);
variable bit_store : std_logic_vector(34 DOWNTO 0);
variable bit_store2 : std_logic_vector(34 DOWNTO 0);
   BEGIN
      sign_bit := export_T(15);
      exp := unsigned(export_T(14 DOWNTO 10));
      signif := unsigned(export_T(9 DOWNTO 0));
      bit_store(34) := '0'; --we will encode sign later
      bit_store(33) := '1';
      if ((signif = to_unsigned(0,signif'length)) and (exp = to_unsigned(0,exp'length))) then
         bit_store(33) := '0'; --special zero case
      end if;
      bit_store(32 DOWNTO 23) := export_T(9 DOWNTO 0); --fraction
      bit_store(22 DOWNTO 0) := (OTHERS=>'0'); --set all remaining bits to zero
      if (sign_bit = '1') then
         --need to take into account sign  (invert all bits and add 1 to find complment)
         bit_store2 := std_logic_vector(shift_right(unsigned(bit_store),to_integer(exp))); --shifts
done by exponent
         regen <= std_logic_vector(unsigned(not(bit_store2))+to_unsigned(1,bit_store2'length));
      else
         --twos complement of unsigned is the same as signed
         regen <= std_logic_vector(shift_right(unsigned(bit_store),to_integer(exp))); --shifts done by
exponent
      end if;


      --Handle special signaling cases
      if ((export_T = CTrigSpecial) or (export_T = CEnergySpecial)) then
         regen <= regen_old; --blank as no trace data
      end if;
      if (export_T = CTrigSpecial) then
         Trig_regen <= '1';
      else
         Trig_regen <= '0';
      end if;
      if (export_T = CEnergySpecial) then
         Energy_regen <= '1';
      else
         Energy_regen <= '0';
      end if;

END PROCESS Pregen;
```

Internal data packet format:

This is how data is sent to the dummy memory (as 16bit words)

Data word:	Contents
W0	0xA5A5
W1	[Chan]&“000”&[PF]&Timestamp(55 DOWNTO 48)
W2	Timestamp(47 DOWNTO 32)
W3	Timestamp(31 DOWNTO 16)
W4	Timestamp(15 DOWNTO 0)
W5	Uenergy(31 DOWNTO 16)
W6	Uenergy(15 DOWNTO 0)
W7	CRC-16

Key:

Chan: 4bit unsigned channel number

PF: 1 bit, ‘1’ indicates pile up (triggering during MWD calculation)

Timestamp: 56 bit unsigned timestamp

Uenergy: 32 bit unsigned energy value from MWD calculation

MATLAB code to calculate CRC-16:

function ui16RetCRC16 = CRC16_CCIT_CORRECT (data, debug)

Crc_ui16LookupTable=[0,4129,8258,12387,16516,20645,24774,28903,33032,37161,41290,45419,49548,...

53677,57806,61935,4657,528,12915,8786,21173,17044,29431,25302,37689,33560,45947,41818,54205,...

50076,62463,58334,9314,13379,1056,5121,25830,29895,17572,21637,42346,46411,34088,38153,58862,...

62927,50604,54669,13907,9842,5649,1584,30423,26358,22165,18100,46939,42874,38681,34616,63455,...

59390,55197,51132,18628,22757,26758,30887,2112,6241,10242,14371,51660,55789,59790,63919,35144,...

39273,43274,47403,23285,19156,31415,27286,6769,2640,14899,10770,56317,52188,64447,60318,39801,...

35672,47931,43802,27814,31879,19684,23749,11298,15363,3168,7233,60846,64911,52716,56781,44330,...

48395,36200,40265,32407,28342,24277,20212,15891,11826,7761,3696,65439,61374,57309,53244,48923,...

44858,40793,36728,37256,33193,45514,41451,53516,49453,61774,57711,4224,161,12482,8419,20484,...

16421,28742,24679,33721,37784,41979,46042,49981,54044,58239,62302,689,4752,8947,13010,16949,...

21012,25207,29270,46570,42443,38312,34185,62830,58703,54572,50445,13538,9411,5280,1153,29798,...

25671,21540,17413,42971,47098,34713,38840,59231,63358,50973,55100,9939,14066,1681,5808,26199,...

30326,17941,22068,55628,51565,63758,59695,39368,35305,47498,43435,22596,18533,30726,26663,6336,...

2273,14466,10403,52093,56156,60223,64286,35833,39896,43963,48026,19061,23124,27191,31254,2801,6864,...

10931,14994,64814,60687,56684,52557,48554,44427,40424,36297,31782,27655,23652,19525,15522,11395,...

```
7392,3265,61215,65342,53085,57212,44955,49082,36825,40952,28183,32310,20053,24180,11923
,16050,3793,7920];

data = [0 0 data]; %zero pad to get correct results

ui16RetCRC16 = hex2dec('FFFF');
for I=1:length(data)
   ui8LookupTableIndex = bitxor(data(I),uint8(bitshift(ui16RetCRC16,-8)));
   ui16RetCRC16 = bitxor(Crc_ui16LookupTable(double(ui8LookupTableIndex)
+1),mod(bitshift(ui16RetCRC16,8),65536));
   if debug == 1
     printf('\n loop: %d, data = %s,  rom_addr = %s, crc_reg = %s', I, dec2hex(data(I)),
dec2hex(ui8LookupTableIndex), dec2hex(ui16RetCRC16));
   end
end
if debug == 1
  printf('\n');
end;


endfunction



%Tests of CRC algorithm

%Blank mesage
data = [];
ui16RetCRC16 = CRC16_CCIT_CORRECT(data,0 );
printf('Blank message CRC is: %s, should be 0x1D0F\n', dec2hex(ui16RetCRC16)); %0x1D0F

%AScii character 'A'
data = [65];
ui16RetCRC16 = CRC16_CCIT_CORRECT(data, 0);
printf('A CRC is: %s, should be 0x9479\n', dec2hex(ui16RetCRC16)); %0x9479

%AScii character message '123456789'
data = [49 50 51 52 53 54 55 56 57];
ui16RetCRC16 = CRC16_CCIT_CORRECT(data, 0);
printf('123456789 CRC is: %s, should be 0xE5CC \n', dec2hex(ui16RetCRC16)); %0xE5CC

%AScii character 'A' 256 times
data = [65].*ones(1,256);5
ui16RetCRC16 = CRC16_CCIT_CORRECT(data, 0);
printf('256*A CRC is: %s, should be 0xE938 \n', dec2hex(ui16RetCRC16)); %0xE938
```