Validity Store and Processor

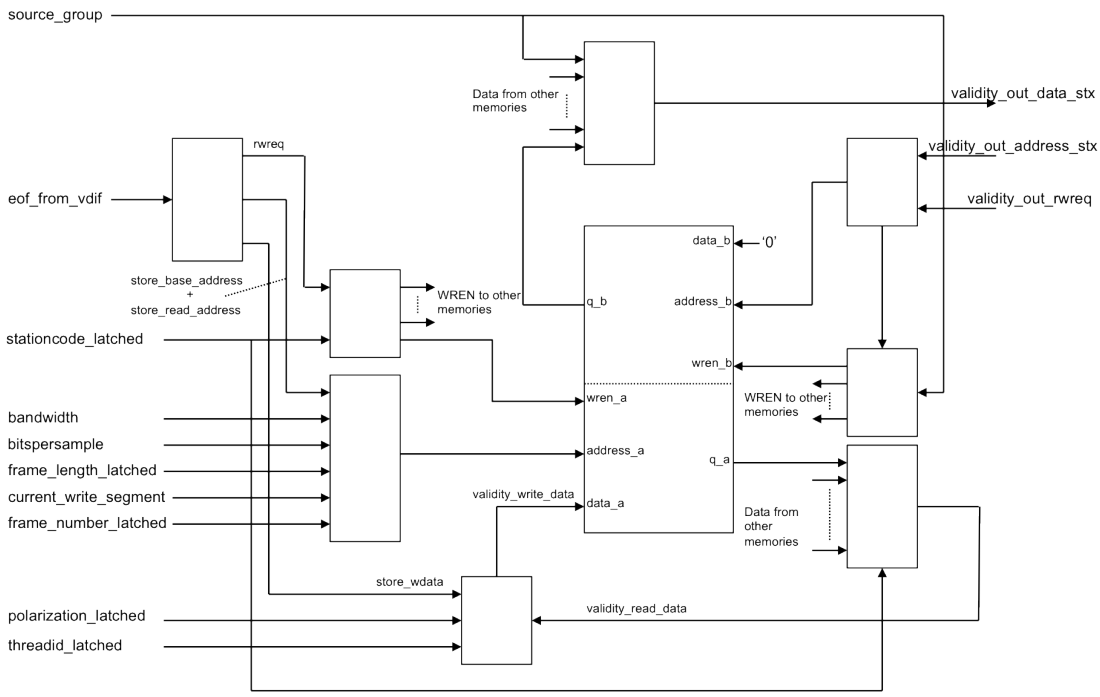
When frames are written into DDR, a corresponding validity bit (1 per frame) is written in a separate, onboard memory. This is called the Validity Store. When data is read from the DDR, the corresponding validity is read from the Validity Store. The validity info indicates to the framer module whether the data that has been read is valid. This is done by the Validity Processor.

Both Validity Store and Validity Processor are explained in detail below.

Validity Store

The 'eof_from_vdif' signal starts the process of writing a frame into DDR. The same signal is used to initiate writing of the validity belonging to that frame. To control this process, a state machine is used.

Below is a simplified block diagram of the Validity Store.



Validity Store block diagram

Structure of the Validity Store

Input signals 'bandwidth' and 'bitspersample' are used to calculate the size of the data segment. This is defined in number of 64 bit units. Only a limited number of valid combinations is possible and therefore a lookup table is used. The implicit calculation that is being done by addressing the table is:

$$\frac{bandwidth \times 2\ (for\ Nyquist) \times bitspersample}{64}$$

The default numbers for 'bandwidth' and 'bitspersample' are 16 MHz and 2 respectively, so that the size of the data segment (signal 'datasegmentsize') would be 10^6 units of 64 bits. This is the amount of data for 1 second.

With the combination of 'datasegmentsize' and 'frame_length_latched', the size of a validity segment can be calculated. A validity segment is actually equal to the number of frames that fit in a data segment. The calculation is done as:

$$\frac{datasegmentsize}{frame\_length\_latched}$$

The result of this calculation, 'validitysegmentsize', must be an integer. Valid and supported frame lengths are: 1000, 1024, 1280, 1600, 2000, 2560, 3200, 4000, 5000, 5120, 6400 and 8000 bytes. In case of the above used numbers for 'bandwidth' (16 MHz) and 'bitspersample' (2), the largest number for validity bits that need to be stored (equal to the number of frames) is 8000. So, to store 4 seconds worth of data, 4 x 8000 bits = 32000 bits are needed. This is per station, so for 8 stations we need 8

x 32kbit per subband and per polarization. The design uses 8 RAMs of 32k x 8 bits, divided over 2 memory banks: 1 for station 0, 2, 4, 6 and 1 for station 1, 3, 5, 7.

The data word of 8 bits is defined as follows:

P0.d	P0.c	P0.b	P0.a	P1.d	P1.c	P1.b	P1.a

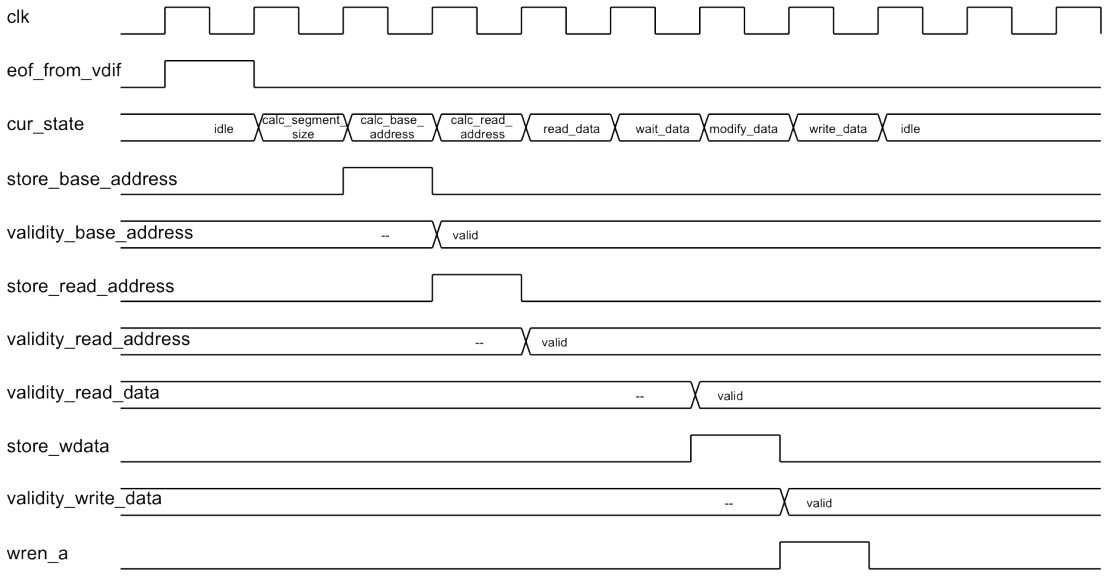
Writing data to the store

Writing to the Validity Store is done by reading the data of the desired location, modifying the bit of interest and writing the data back to the same location. The 'validity_base_address' is calculated and clocked into a register. It is obtained by: 'validitysegmentsize' x 'current_write_segment'. This defines to which of the 4 segments the validity data belongs. Within that segment (or second) the exact location is addressed by adding 'frame_number_latched' to 'validity_base_address'. The result is called 'validity_read_address'.

The 'validity_read_address' is applied to all 8 memories. This results in 8 data words of 8 bits each. Every data word belongs to one of the 8 stations. Based on the station currently being processed, the appropriate data word is selected. The selected data word is called 'validity_read_data'. The individual bits of the data word cannot be changed inside the memory. Therefore, as described above, a read-modify-write procedure is used. Signals 'polarization_latched' and 'threadid_latched' address the bit position within the data word. That particular location will be set (to '1'), while the other 7 bits remain unchanged. The now obtained new data word is called 'validity_write_data'.

The 'validity_write_data' must be written back into the store to the same location as it was read from. This is done by applying it to all 8 memories and toggle the appropriate write enable. This 'wren_a' is based on 'stationcode_latched'.

Below is a diagram of the write timing.



Validity Store write timing

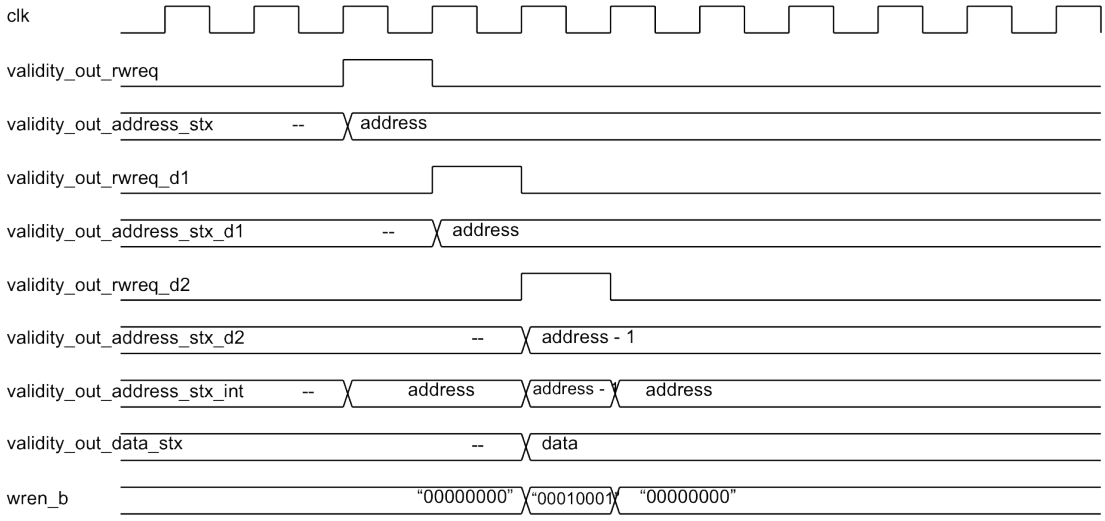
Reading data from the store

The 2 memory banks (1 for station 0, 2, 4, 6 and 1 for station 1, 3, 5, 7) are read by 2 address signals coming from the Validity Processor. These signals, 'validity_out_address_st0' and 'validity_out_address_st1' are used to read the validity data from the memories and to clear the previous location. Both addresses are registered twice, but the second time the current location minus 1 is stored. That will serve to clear the previous location. The signal 'validity_out_rwreq' is registered twice as well, because it serves as a write enable for the location to be cleared.

The addresses that are actually applied to the memories are either the raw input signals or the "current location minus 1" signals. This depends on whether a read ('validity_out_rwreq_d2' = '0') or a write ('validity_out_rwreq_d2' = '1') has to be performed. The name of the signals to the memories are 'validity_out_address_st0_int' and 'validity_out_address_st1_int'. Out of the 8 data words that become available 2 clock cycles after the read address has been applied, the data words for the appropriate station pair is selected. For this, a registered version of 'source_group' is used. The 2 data

words, 1 for each station, is then made available for further processing by the Validity Processor. They are called ‘validity_out_data_st0’ and ‘validity_out_data_st1’.

At the same time that data is available on the output of the memories, the previous location for the current station pair is cleared by writing a ‘0’ to the appropriate locations of both banks. Asserting the right write enable signals, ‘wren_b’, the registered version of ‘source_group’ is used. This can be seen in the timing diagram below.

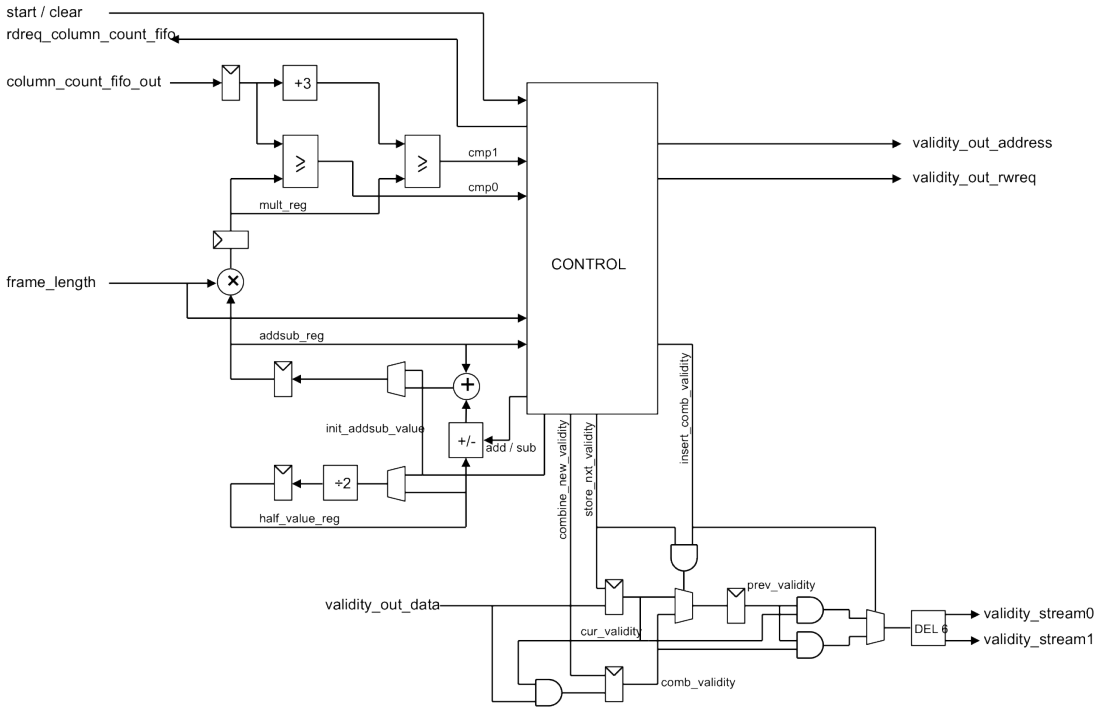


Validity Store read timing

Validity Processor

The task of the Validity Processor is to calculate to which frame the incoming address (‘column_count_fifo_out’) belongs, to read the validity for that frame from the Validity Store and to process this validity so that it can be used by the framer. This is done at the end of an FFT-period, just before the data is applied to the framer.

Below is the block diagram of the Validity Processor.



Validity Processor block diagram

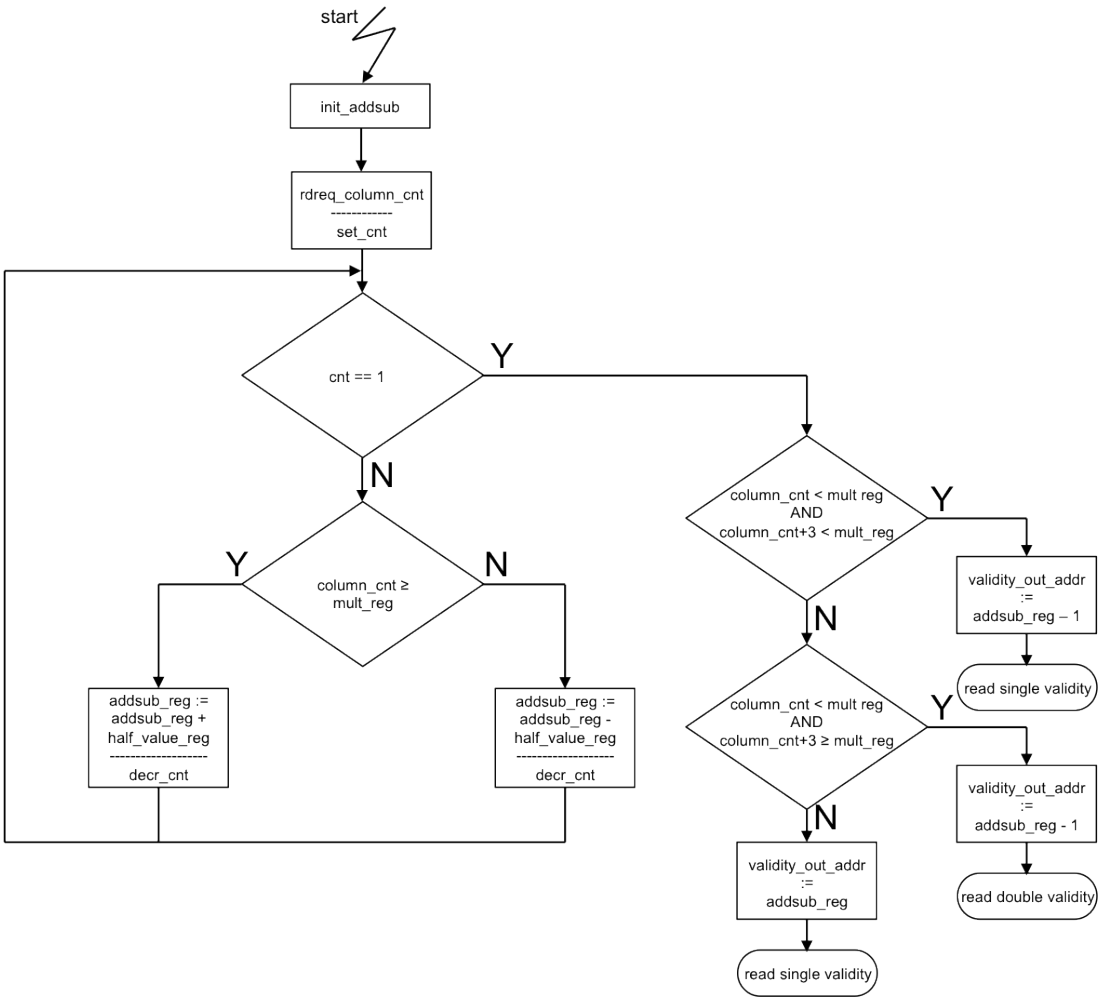
Calculation of the frame number

The start address that is used to start reading from the DDR marks the start of an FFT. The same address is used to calculate which location must be read from the validity store. Basically, the number of the frame in which the FFT starts is calculated. A simple way of doing this, is to divide the incoming address 'column_count_fifo_out' by the predefined 'frame_length'. However, a division by a number, other than a power of 2, is rather complicated in hardware and takes a lot of resources. Therefore, a method based on a successive approximation algorithm is used, which is explained below.

Upon a 'start' signal, registers 'addsub_reg' and 'half_value_reg' are preloaded with a value that is based on 'frame_length' and the actual configuration of subbands and bandwidth. The value that is preloaded in 'addsub_reg' represents the center address of the validity store, rounded-up to the nearest power of 2. The initial value for 'half_value_reg' is half the value of 'addsub_reg'. Then, data is requested from the column count fifo, which will become available after a few clock cycles. Signal 'column_count_fifo_out' is the address that is used to form the DDR read address and is in words of 256 bits. Here, the signal is multiplied by 4, so that it represents words of 64 bits. This value, which is stable during the remainder of this procedure, is compared with 'mult_reg', which is the result of 'addsub_reg' x 'frame_length'. Based on whether it's greater/equal or smaller than 'mult_reg', the number in 'half_value_reg' is added to or subtracted from the value in 'addsub_reg'. After a predefined number of clock cycles, the final result is available in 'addsub_reg'. This value is then transported to output 'validity_out_address', together with signal 'validity_out_rwreq'.

Address 'column_count_fifo_out' can be part of 2 frames (it is reworked to represent a 64 bit value, because frames end on 64 bit boundaries). In the case that it is indeed part of 2 frames, the validity for both these frames must be read. This is taken care of by the control state machine. To detect whether the start of an FFT is indeed in both frames, 'column_count_fifo_out' as well as 'column_count_fifo_out_plus_3' is used. If the first one is smaller than mult_reg, while the plus_3 signal is bigger than or equal to mult_reg, then the start of the FFT is in both frames.

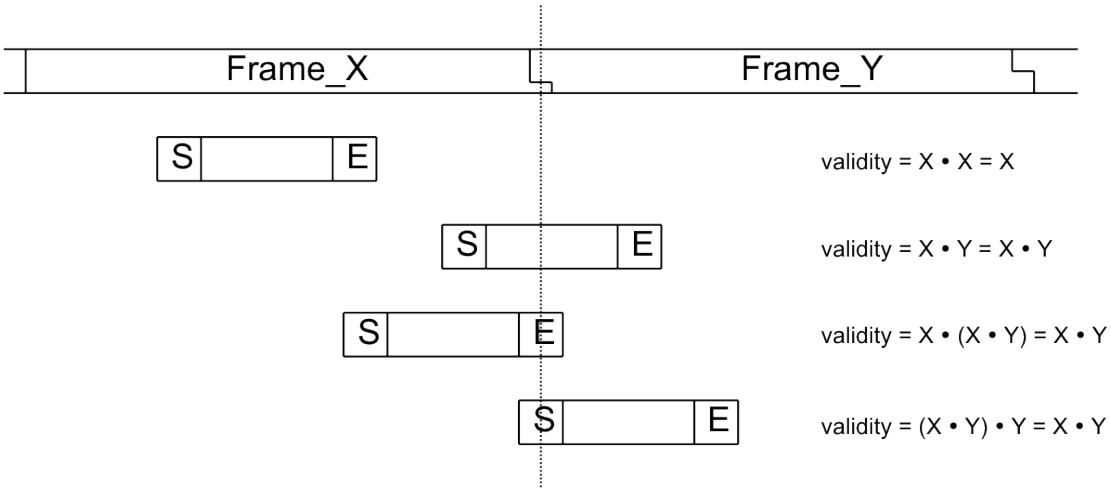
The flow chart of the implemented successive approximation algorithm can be viewed below.



Successive approximation implementation

Obtaining the correct validity figures

As stated earlier, 'column_count_fifo_out' can be in 2 frames. To be more precise: An FFT can either start in a certain frame and end in the same frame, it can start in a certain frame and end in the next frame, it can start in a certain frame and end on the boundary of that and the next frame or it can start on the boundary of 2 frames and end in the second frame. All 4 scenarios are covered by the Validity Processor. The next figure shows all 4 possibilities, together with the actions to be taken by the Validity Processor to obtain the correct validity figures.



Validity Processor FFT positions within frame