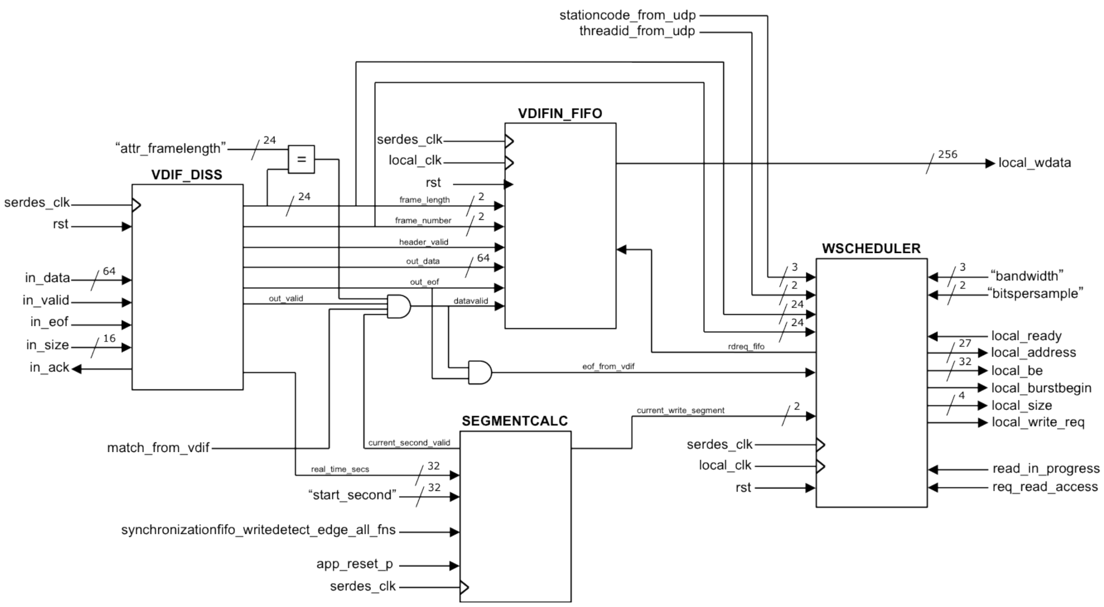
Write Side PKT_RX

The Write Side of the PKT_RX module takes care of receiving VDIF frames and writing their data payload in DDR3 memory.

Architecture



Block diagram Write Side

The modules are described in detail below.

VDIF_DISS module

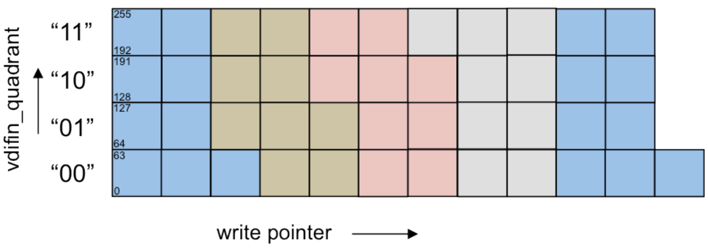
The VDIF disassembler extracts header data from the VDIF frame. Data that is currently used: frame number, frame length and time stamp. Frame number and frame length are needed in an early stage of the data storage process, which is performed by the VDIFIN_FIFO module. The header_valid signal flags the availability of valid frame number and frame length data.

VDIFIN_FIFO module

The VDIFIN_FIFO module stores the incoming VDIF frame to a FIFO before it is send to the DDR3 memory. The 64-bit data that is coming in from the VDIF_DISS module is first stored in a 256-bit register and then it is forwarded to the FIFO. The order it is written into the FIFO is the same as it will be written into the DDR.

There is a demultiplexer at the input of the 256-bit register. This demux is controlled by a signal called vdifin_quadrant. Upon assertion of the header_valid signal, coming from VDIF_DISS, vdifin_quadrant is loaded with a preset value, determined by the product of the 2 LSBs of the frame_length and the 2 LSBs of the frame_number. Writing to the register is done when in_data_valid = '1'. This is the case when the data coming from VDIF_DISS is valid, the time_stamp in the VDIF frame matches the expected time, the frame_length matches the configured length and data is meant for the appropriate station/thread pair. If data is written adjacently to the FIFO and to the DDR then it makes the process of reading out data from the DDR easier.

The next figure shows five frames with a frame_length of 9, which is actually 9 x 64 bits.



frame_number	vdifin_quadrant
0 (LSBs = "00")	"01" x "00" = "00"
1 (LSBs = "01")	"01" x "01" = "01"
2 (LSBs = "10")	"01" x "10" = "10"
3 (LSBs = "11")	"01" x "11" = "11"
4 (LSBs = "00")	"01" x "00" = "00"

Write order

Each time vdifin_quadrant reaches value "11" or the end of frame signal is received, the data stored in the 256-bit register is written into the FIFO.

SEGMENTCALC module

The segment calculator determines the current segment in DDR where data will be stored. The next figure shows the memory format. Since 8 stations will be supported per front node, there will be 8 of these sections in total, 1 for each station.

Station X	Thread D	Polarization R	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
		Polarization L	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
	Thread C	Polarization R	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
		Polarization L	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
	Thread B	Polarization R	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
		Polarization L	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
	Thread A	Polarization R	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"
		Polarization L	Segment "11"
			Segment "10"
			Segment "01"
			Segment "00"

The first 3 columns are defined outside this module, either by the udp_packetizer module or by a control register. The last column represents the memory segment. In each segment 1 second of data will be stored. This means that per Station/Thread/Polarization 4 seconds of data are stored.

Upon an application reset the current_write_segment and nominal_write_segment pointers are reset to segment "00". Following the reset signal, the current_write_second signal is loaded with a predefined (CSR) value, applied through start_second. Now the incoming real_time_secs signal (field in the VDIF header) is compared to current_write_second AND current_write_second +/- 1. If there is a match, then data will be written to the currently appointed segment in memory. Otherwise, it will be written to the next or previous segment.

Every time a set_batchsize_req is detected, the current_write_second and nominal_write_segment values are incremented. So nominal_write_segment is only changed every set_batchsize_req, while current_write_segment can change on a per frame basis, but always based on the value of nominal_write_segment.

This module also indicates to the VDIFIN_FIFO module that the incoming data is valid to be stored since the timestamp is correct.

WSCHEDULER module

The WSCHEDULER module consists of arbitration logic to handle write and read requests and logic that generates the interface signals to the DDR controller.

Upon assertion of then end of frame input, a write request is started. The arbitration logic checks whether there is a read request as well. If so, then that has precedence over a write request. If neither a read request is done nor a read transfer is currently in progress, then a write transfer is started.

The data coming out of the VDIFIN_FIFO module is 256 bits wide. This is also the size of the data bus to the DDR controller. Since the length of a VDIF frame is in multiples of 8 bytes, the boundaries are not on a 256-bit basis, but on a 64-bit basis. Output signal local_be indicates which portion of the local_wdata is to be stored in the DDR.

To prevent gaps between the frames that are written to DDR, it is necessary to calculate the exact start_address and end_address of the memory location. These addresses are calculated as follows.

start_address = frame_number * frame_length + current_write_segment_base

end_address = (((frame_number + 1) * frame_length) – 1) + current_write_segment_base

The current_write_segment_base value comes from a lookup table that is addressed by: bandwidth(3 bits) & bitspersample(2 bits) & current_write_segment(2 bits). This lookup table contains the segment boundaries for a given configuration. The boundaries that are stored in the table are: (bandwidth * 2 * bitspersample) / 64) * N, where N = [0, 1, 2, 3]. So for every configuration there are 4 entries in the table. For example: bandwidth = 16 MHz, bitspersample = 2. The boundaries for the 4 segments are 0, 1×10^6, 2×10^6, 3×10^6. The precision of these boundaries are in multiples of 8 bytes (64 bits).

The address that is applied to the DDR controller is a concatenation of 3 components:

local_address = stationcode & threadid & write_address

Components 'stationcode' and 'threadid' are supplied by top level logic, while 'write_address' is generated by the WRITEOP module.

WRITEOP module (part of WSCHEDULER)

WRITEOP is part of the WSCHEDULER module. It contains the actual write controller and address generator. The start_address indicates the first address location where data must be stored. The external local_address that is applied to the DDR controller has 256-bit precision, while start_address has 64-bit precision. This means that internal boundaries are considered on multiples of 64 bits, while externally the boundaries are on 256 bits. To obtain the external address from the internal address, the 2 LSBs are cut off. Output signal local_be takes then care of the 64-bit precision. The next picture shows the relation between start_address, end_address, internally used address, write_address and byte enable (be). The frame length of the 4 frames in this example is 9 (multiples of 64 bits).

start_address	0			9			18			27			
end_address	8			17			26			35			
int_address	0	4	8	9	13	17	18	22	26	27	31	35	
write_address	0	1	2	2	3	4	4	5	6	6	7	8	
byte_enable	FFFFFFFF	FFFFFFFF	000000FF	FFFFFF00	FFFFFFFF	0000FFFF	FFFF0000	FFFFFFFF	00FFFFFF	FF000000	FFFFFFFF	FFFFFFFF	
	FRAME 0			FRAME 1			FRAME 2			FRAME 3			

Address relations

The WRITEOP module generates the read request signal for VDIFIN_FIFO. With a latency of 1 clock cycle, data will come out of the FIFO and will be written into the DDR.