

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
HAYSTACK OBSERVATORY  
WESTFORD, MASSACHUSETTS 01886

Phone: (781) 981-5407  
Fax: (781) 981-0590

25 April 2006

To: Mark 5 Group

From: B. Fanous

Subject: Mark 5B Data output Module (DOM) Hardware Design Document

## **Overview**

The Mark5B is a data recording and playback system for radio astronomy systems. Strictly speaking, the DOM (Data Output Module) is the playback mode of the data I/O card in the Mark5B system, though the system itself is often referred to as a DOM when configured for playback mode. The DOM mode is invoked by installing playback specific code into the FPGA in the Mark5B I/O PCI card in the system backplane.

The DOM is a data playback machine which accepts data from the StreamStor I/O board (also resident in the backplane) over an FPDP<sup>1</sup> bus and transmits this data through an MDR-80 output port. The DOM conforms to the VSI-H<sup>2</sup> specification and is capable of playing back recorded VLBI data through a VSI output port. The DOM can also be used as a Station Unit to playback VLBI data to a correlator. Other test modes (a Test Vector Generator and Receiver) are also included in the DOM.

Typically, a phase calibration tone extraction (phase cal) subsystem is resident in the DOM. The connection to that subsystem will be discussed here, but the details will be discussed in the *Phase Calibration Tone Extraction Subsystem Hardware Design Document*.

---

<sup>1</sup> FPDP – Front Panel Data Port. See ANSI/VITA 17-1998 specification.

<sup>2</sup> VSI-H . VLBI Standard Interface (Hardware)

## ***Conventions***

Signal names will be written in `courier` font.

Negative logic signals will be appended with a # sign.

**Registers** will be written in bold.

Register bit fields will be written enclosed in <angular brackets>

## Pin List

NAME	VHDL port type	Description
FPDP_clk_in	in std_logic	FPDP clock input set to 33 MHz by StreamStor
FPDP_dvalid_n	in std_logic	FPDP data valid (active low)
FPDP_data	in std_logic_vector(31 downto 0)	FPDP data bus
FPDP_dir_n	in std_logic	FPDP direction ('0' for DOM)
FPDP_suspend_n	out std_logic	FPDP suspend (active low) to StreamStor
FPDP_nrdy_n	out std_logic	FPDP not ready (active low) to StreamStor
interrupt	out std_logic	Interrupt signal to PLX chip
plx_ld	inout std_logic_vector(15 downto 0)	Data to/from PLX chip
plx_la	in std_logic_vector(23 downto 2)	Address bus from PLX chip
plx_lbe1_n	in std_logic	Least Significant bit of internal local bus from PLX chip
plx_ads_n	in std_logic	Address/Data strobe from PLX chip
plx_clk	in std_logic	PLX local bus clock
plx_lw_r_n	in std_logic	PLX local bus write (high) /read (low) strobe
sdram_ras_n	out std_logic	SDRAM Row Address Strobe
sdram_cas_n	out std_logic	SDRAM Column Address Strobe
sdram_we_n	out std_logic	SDRAM Write Enable
sdram_rege	out std_logic	SDRAM register enable - we are using registered DIMMs
sdram_s0_n	out std_logic	SDRAM chip select
sdram_s2_n	out std_logic	SDRAM chip select
sdram_cke	out std_logic	SDRAM clock enable
sdram_dqmb	out std_logic_vector(7 downto 0)	SDRAM data mask bits - held low
sdram_ba	out std_logic_vector(1 downto 0)	SDRAM bank address
sdram_a	out std_logic_vector(12 downto 0)	SDRAM address bus
sdram_data	inout std_logic_vector(71 downto 0)	SDRAM data bus
sdram_scl	out std_logic	Held Low - for SDRAM serial chip which is unused
sdram_sa	out std_logic_vector(2 downto 0)	Held Low - for SDRAM serial chip which is unused
sdram_sda	in std_logic	Not used/floating - for SDRAM serial chip which is unused
sdram_brd_clk_out	out std_logic_vector(1 downto 0)	clock signal to SDRAM DIMM
sdram_brd_clk_fbk_in	in std_logic	clock feedback signal from SDRAM DIMM for internal DCM for deskew
sdram_brd_clk_fbk_out	out std_logic	clock feedback signal to SDRAM DIMM for internal DCM for deskew
rbs	out std_logic_vector(31 downto 0)	Output data bus
qvalid	out std_logic	Output data validity
rot1pps_bocf	out std_logic	ROT1PPS output in VSI mode, BOCF output in SU mode, '0' in TVG or TVR modes

rclk_out_board	out std_logic	Output data clock
r1pps	out std_logic	Output R1PPS in VSI mode, Output PPS in TVG mode, '0' in SU or TVR modes
sdram_clk_in	in std_logic	SDRAM Interface 80 MHz clock input
dps1pps	in std_logic	Input 1PPS
dpsclk	in std_logic	Input clock for DOM Back End
spare	out std_logic_vector(1 downto 0)	Software controllable Spare output signals
external_reset	in std_logic	External hard reset from PLX chip
ad9850_clk_control	out std_logic_vector(10 downto 0)	Control signals to AD9850 clock generator chip for alternate Back End clock.
internal_clk_in	in std_logic	Alternate Back End clock generated by AD9850
rotmon	out std_logic	Output 1PPS signal (ROT1PPS) for oscilloscope monitoring at SMB connector
LA_header0	out std_logic_vector(16 downto 0)	Logic Analyzer connector J13
LA_header1	out std_logic_vector(16 downto 0)	Logic Analyzer connector J14
LA_header2	out std_logic_vector(16 downto 0)	Logic Analyzer connector J15
dvr_swA	out std_logic	LED row enable A
dvr_swB	out std_logic	LED row enable B
dvr_swC	out std_logic	LED row enable C
dvr_swD	out std_logic	LED row enable D
dvr_r0	out std_logic	LED red column 0 enable
dvr_r1	out std_logic	LED red column 1 enable
dvr_g0	out std_logic	LED green column 0 enable
dvr_g1	out std_logic	LED green column 1 enable
dvr_b0	out std_logic	LED blue column 0 enable
dvr_b1	out std_logic	LED blue column 1 enable
en_vsi_in	out std_logic	Held low - board related
en_vsi_out	out std_logic	Held high - board related
fpdp_not2	out std_logic	Held high - FPDPII mode
fpdp_a2b	out std_logic	Held high - board related
fpdp_regenl	out std_logic	Held low - board related
dotmon	out std_logic	Held low - DIM related
fpdp_xpio2	out std_logic	Held low - board related

## ***Major Functional Blocks***

The DOM is divided into three major functional blocks (not including the phase cal): the Front End, the SDRAM Interface, and the Back End. There is also a Local Bus Interface which simply translates the PLX chip's local bus signals to a more convenient format internal bus. There are also various global registers and clock related buffers and DCMs. The major functional blocks can be seen in Figure 1.

### **Front End**

The DOM Front End data path is shown in Figure 2. The DOM Front End datapath is essentially a chain of alternating sub-blocks and FIFOs. It is the job of each sub-block to keep the preceding FIFO empty and keep the subsequent FIFO full. Each sub-block contains logic to prevent overflow and underflow of the surrounding FIFOs.

The DOM Front End comprises all functional blocks in the data path between the FPDP interface to the StreamStor I/O card and the Mark 5B I/O board SDRAM Interface. The data in Front End is in the TOT (taken on tick) domain, meaning it is referenced to the record time. The Front End datapath includes an FPDP compatible interface, a disk frame header remover and checker (Strip-header block), a data unpacker (Unpack), and a full 32x32 crossbar (Xbar). Data is deposited into a FPGA-internal RAM for use by the SDRAM Interface. The entire datapath in the Front End is in the FPDP clock domain (33 MHz).

Data is also tapped off from the Front End for delivery to the phase cal subsystem.

Each sub-block is discussed in detail below.

#### *FPDP Interface*

The FPDP bus compatible interface to the DOM is the port by which data enters the DOM from the StreamStor I/O card. The 33 MHz clock for this sub-block, and the entire Front End datapath, is sourced by the StreamStor card once it begins attempting playback. This clock is subsequently used for the entire DOM Front End datapath. Therefore it is critical that this clock be available and stable prior to the Front End sub-blocks being taken out of reset. The DOM FPDP Interface has the ability to hold-off data delivery from the StreamStor by asserting its *SUSP#* signal. Until the FPDP interface sub-block is enabled via the software, data is held-off in this manner.

The FPDP interface block accepts data from the StreamStor I/O card and places it in a 127 element FIFO for use by the Strip-header sub-block. Incoming FPDP data is held off, as above, when this FIFO is more than 75% full. The StreamStor card can continue to pass data on the FPDP bus for up to 16 clock cycles after *SUSP#* has been asserted (by FPDP specification), and the extra 25% of room in the FIFO will be sufficient, should this happen. The *SUSP#* signal will

be deasserted once the FIFO drops below 50% full. This method should protect the FIFO from overflow.

### *Strip-Header*

The data delivered to the DOM by the StreamStor over the FPDP bus contains Disk Frame Headers to mark the record time on the disk. The format and placement of the headers is described in the *Mark5B Specification*. This is a non-data-replacement format. It is the job of the Strip-header sub-block to take data out of the FPDP Interface FIFO, remove these headers and determine if there is a header error.

There are a few different header errors which are checked for by the Strip-header sub-block. A header must occur every 2504 32-bit data words (2500 are data, 4 are header). If a header is misplaced, a header error is generated. The first word of the header must be the predefined SYNC word. If the SYNC word is incorrect, a header error is generated. Headers have a serial number in the second word which simply increments by 1 every disk frame header, starting at 0 and rolling over, after every one second of recorded data. This serial number is referred to as the *disk frame count*. The first Disk Frame Header that the Strip-header sub-block sees must be a disk frame count of 0, meaning data must start at the beginning of a recorded second. If the disk frame count does not increment and rollover correctly (based on the DOM's knowledge of the sampling rate set by the control software in the **Disk Frames per Second Register**) a header error is generated. Data playback is halted on a header error, the ERROR LED is illuminated and the software <header\_err> bit is set in the **Status Register**.

It is also the job of the Strip-header sub-block to determine which samples are valid and which are invalid. This is done through the use of predefined invalid code words. There are two invalid code words – those generated by the StreamStor card (due to a StreamStor error, etc) and those generated by the recorder (DIM) (due to actual invalid record data). These 32-bit words can be set independently to the same or different values via software. When the Strip-header sees a 32-bit data word matching either of the invalid words, the data is marked invalid. When this data reaches the output port of the DOM, it will be marked as such by a de-assertion of the QVALID signal. The invalid words are set in the **StreamStor Invalid Reg0**, **StreamStor Invalid Reg1**, **DIM Invalid Reg0**, and **DIM Invalid Reg1** registers.

It should be noted that invalid words can occur during the data or the Disk Frame Headers. If an invalid word occurs during a Disk Frame Header, true checking of the header becomes impossible and that header word is assumed to be valid. As such, the DOM keeps an internal disk frame counter to keep track of record time, just in case a disk frame header is marked invalid. This internal counter is also used to check headers.

The Strip-header sub-block also notifies the controlling software when it has received the start of a second of recorded data via the TOT Interrupt. This interrupt can be masked or unmasked like all DOM interrupts. When a disk frame header of Disk Frame count 0 is found (or should be found at that time, based on the DOM's internal disk frame counter), a TOT interrupt is generated. The last two Disk Frame Header words (of Disk Frame count 0) denote the VLBA time code are posted to registers (**DiskFrame VLBA Time Code0** and **DiskFrame VLBA Time**

**Code1)** for checking by the software. Also a count of recorded seconds is posted in the **TOTCount** register. This register is 16-bits wide and rolls over by itself at full-scale. The **TOTCount** register should immediately increment to 1 once data is read off the FPDP bus because the first header should be the beginning of a second.

The Strip-header sub-block passes pure data (Disk Frame Headers removed) to a 31 element FIFO. At this point the data is still 32-bits wide but carries two tag bits – validity and TOT.

### *Unpack*

Data is transmitted and received (during record and playback, respectively) over the FPDP bus as 32-bit wide words. However, it is not always desirable to record 32-bits per time sample. Say, for example, only 4 channels at 2-bits/sample need to be recorded, yielding 8-bits per clock. That is to say there are 8 *active bit streams*. To store this efficiently on disk, the recorder (DIM) packs 4 of these sets of samples into one 32-bit word. So each 32-bit word on the disk contains 4 time samples. The various packing schemes are discussed in the *Mark5B Specification*. It is the job of the Unpack sub-block to undo the data packing that the recorder has done. So, in the example, the Unpack sub-block would create 4 time samples of 8-bits each. The native word size of the data path is 32-bits through the DOM Front End, so the Unpack would create 4 32-bit words with only the lower 8-bits having meaning.

The unpack function is performed by a loadable shift register. The **Unpack Code** software register indicates how many active bit streams have been recorded. The register shifts out however many active bit streams are indicated by the **Unpack Code** register each clock cycle. The shift register is reloaded after  $32/(\text{active number of bit streams})$  shifts. The active bit streams are in the low order bits. Figure 2 has another example of this. The following crossbar can move these bits to the appropriate bit stream. Because the Disk Frame Headers mark 32-bit wide words as valid and/or TOST, the Unpack sub-block cannot determine which subsection of a 32-bit wide word is actually the valid/TOST sample. Therefore all parts of a 32-bit wide word which is to be Unpacked are given the same validity/TOST state. So it is possible for up to 32 consecutive samples coming out of the Unpack sub-block to be marked as the TOST sample.

### *Xbar*

The 32x32 crossbar (*Xbar*) allows any bit stream to be rerouted to any other bit stream. In fact, any input bit stream can be the source of multiple output bit streams. The source of each of the 32 output bit streams is set in the 32 **Xbar Slice Setting Regs**. At reset the values of these registers make the crossbar act as a straight-through circuit. The crossbar is implemented as an array of 32-1 multiplexers each fed with all 32 input bit streams.

### *Xbar RAM*

This RAM is where data, which has passed through the DOM Front End, is deposited. It is interesting only in that its read and write sizes are unequal. The Xbar feeds it with 36-bit wide data (32-data + validity + TOT + 2 pad bits<sup>1</sup>), but the subsequent SDRAM Interface does 72-bit

---

<sup>1</sup> The 2 pad bits exist throughout the datapath in the entire system. These exist because of the native port size of some of the Xilinx primitives. The synthesis tool (XST) *should* strip out the unused ones from the datapath. Essentially the pad bits should exist in the VHDL code only and not in the actual implementation.

wide reads. The RAM is 128 x36 or 64x72. The write data clock is at the StreamStor 33 MHz clock, but the read data clock rate is at 80 MHz. The SDRAM Interface tries to keep this RAM empty (without underflow).

## SDRAM Interface (SDRAM\_xface)

The SDRAM Interface is shown in Figure 3.

An external 256 MB Registered ECC SDRAM DIMM<sup>1</sup> is used as a data buffer in the DOM datapath. Data is written to the SDRAM continuously by the DOM Front End and read out by the DOM Back End. The SDRAM Interface manages the flow of data into and out of the SDRAM DIMM. The software has limited control of the read pointer, which allows for large jumps in the data going to the DOM output. The ability to control the read pointer is useful, for example, in compensating for the earth's rotation or skipping over data that was recorded while an antenna was moving from source to source. The SDRAM Interface attempts to keep the buffer half full at all times, so while the buffer is capable of holding up to 2 seconds<sup>2</sup> of data at a 32 Ms/s recording rate, only 1 second of unread data is ever actually in the buffer.

The software has control of the read pointer at discrete times which differ depending on output mode. The new read pointer is assigned by the **SDRAM Address0** and **SDRAM Address1**. In Station Unit output mode, values written to these registers take effect at the start of the next Correlator Frame (explained in the DOM Back End section). In the three VSI modes (output, TVG, and TVR) the values written to these registers take effect immediately after the next unsuppressed PPS (explained in the DOM Back End section).

The data is written to and read from the SDRAM in 72 bit word blocks – two 32-bit data words, two validity bits and two TOT bits; the rest are unused. However, the **SDRAM Address0** and **SDRAM Address1** specify a precise 32-bit word to be the start of the reads. To implement this effectively, the SDRAM Interface reads the 72-bit word containing the desired 32-bit data word from the external SDRAM, and then extracts the appropriate 32-bit word internally in the SDRAM Receiver module.

The SDRAM Interface is capable of performing four different operations on the external SDRAM DIMM: read thirty-two 32-bit words (1kb), write thirty-two 32-bit words (1kb), SDRAM refresh, and SDRAM initialization. The SDRAM Interface is broken down into several sub-blocks: an SDRAM Arbiter that manages the read and write pointers, manages the SDRAM refresh counter, and determines what operation (read, write, refresh, initialization) should happen when; an SDRAM core which is actually a collection of mini-blocks that implement each of the four SDRAM functions – read, write, refresh, and initialization; an SDRAM receiver that extracts the appropriate 32-bit word pointed to by the read pointer from an SDRAM read burst; and an SDRAM Clocking Module that simply creates the different SDRAM clocks from an external oscillator.

---

<sup>1</sup> The DIMM part number used on the Mark 5B I/O board is Micron part number: MT9LSDT3272G-133

<sup>2</sup> The buffer is actually capable of holding more than 2 seconds of data (at 1024 Mbps FPDP aggregate data rate) because the RAM can hold  $2^{31}$  bits (not including ECC bits) but some of the RAM is never used for simplicity of implementation.

## *SDRAM Arbiter*

The Arbiter controls the SDRAM Interface. It manages reading and writing of the SDRAM in such a way as to keep the buffer approximately half full. It also refreshes the SDRAM at appropriate intervals to avoid losing data.

A simplified diagram for the SDRAM Arbiter state machine is shown in Figure 4. The SDRAM Arbiter goes through a startup sequence after it is enabled where the external RAM buffer is initialized, filled almost completely<sup>1</sup>, and refreshed as appropriate. After the startup sequence, the SDRAM Arbiter enters its main operating states. A bit is set in the **Status Register** once the startup is completed. The Arbiter waits in its “IDLE” state until data is needed to be read from the SDRAM to keep the SDRAM Receiver (discussed later) FIFOs filled, until data is available in the XBAR RAM to be written to the SDRAM buffer (subject to the requirement to keep the buffer approximately half full), or until an SDRAM Refresh is required. Note that reads are performed whether the read will drop the buffer below half full or not because reads are of such high priority.

If so many reads are performed without any writes to the SDRAM (because of no new data in the XBAR RAM) that the read and write pointers cross, then the SDRAM Arbiter will transition to the “FINISHED” state and will notify the SDRAM Receiver that no more SDRAM reads will be performed.

The order of priority from the “IDLE” state is (in decreasing order): “FINISHED”, “REFRESH”, “READ”, and finally “WRITE”.

SDRAM refreshes are performed every 7  $\mu$ s to satisfy the specified refresh rate of the registered ECC SDRAM we are using. The refresh counter is implemented as a simple counter but clocked by a divided-down, 10 MHz version of the SDRAM data clock of 80 MHz.

To perform the four SDRAM functions of Initialization, Read, Write, and Refresh, the SDRAM Arbiter passes a 2-bit command code and an address (if needed) to the SDRAM core sub-block. The Arbiter also supplies an `sdram_cmd_strobe` to the SDRAM core as a “go” signal to begin executing the function indicated by the command code. Key to the Arbiter operation is the fact that the SDRAM Core sub-block provides a return `ready` signal to the Arbiter once it is done executing a command, thus allowing the SDRAM Arbiter to issue a new command code. This handshaking of `sdram_cmd_strobe` followed by `ready` prevents commands from being missed.

---

<sup>1</sup> While the SDRAM buffer is typically only half full during operation, it is filled almost completely during initialization because typical VLBI Station Unit applications will set the starting read pointer address to somewhere near the midpoint of the RAM. This is to account for the source wavefront propagation time to various stations.

## *SDRAM Core*

The SDRAM Core sub-block is actually nothing much more than a parser of the command code from the Arbiter to activate one of four mini-blocks, each of which executes one of the four commands. These mini-blocks are *rd\_core*, *wr\_core*, *sdram\_init\_core*, and *autoref\_core* and when one of these is active it controls the RAS#, CAS#, etc SDRAM signals.

- *rd\_core* – This mini-block performs the read function by controlling the RAS#, CAS#, WE# and CS# signals to the SDRAM DIMM, as well as, the external address bus. Two bursts reads to consecutive locations are performed with each burst being eight 72-bit words in length. So 1 kb blocks of data are read each time. The *rd\_core* also precharges all the rows when the read is done. The *rd\_core* takes twenty-six 80 MHz clock cycles to complete such an operation. Only burst reads are performed and reads may only start at 1kb block boundaries.
- *wr\_core* – This mini-block performs the write function by controlling the RAS#, CAS#, WE# and CS# signals to the SDRAM, as well as, when different parts of the read address are placed on the external address bus. Two burst writes to consecutive locations are performed with each burst being eight 72-bit words in length. So 1kb blocks of data are written each time. The *wr\_core* also precharges all the rows when the write is done. The *wr\_core* takes twenty-six 80 MHz clock cycles to complete such a write. Only burst writes are performed and writes may only start at 1kb block boundaries.
- *sdram\_init\_core* – This mini-block performs the SDRAM standard startup sequence (see the datasheet for the MT9LSDT3272) as well as loading of the mode register. The mode register is set to a value of 0000000110011 binary. This corresponds to using sequential burst lengths of eight 72-bit words with a CL=3 (CAS# latency). Note there is a 100  $\mu$ s wait during the standard startup sequence.
- *autoref\_core* – This mini-block performs a refresh of the SDRAM by controlling the RAS#, CAS#, WE# and CS# signals to the SDRAM. The refresh takes nine 80 MHz clock cycles.

## *SDRAM Receiver*

The SDRAM Receiver is shown in Figure 5.

The SDRAM Receiver performs several important functions in linking the SDRAM Interface to the DOM Back End. The receiver reduces the native SDRAM word size of 72-bits back to the DOM Back End native word size of 34-bits (32-data, 1 validity, 1 TOT). It also monitors the fullness of the internal dual-ported block RAM at the entry point of the DOM Back End called the CFDR and tries to keep it almost full. Finally, if the first read after a read pointer adjustment (either due to a CF interrupt or unsuppressed PPS, depending on mode) points to a data word in the middle of what would be a 1 kb *rd\_core* operation, the SDRAM Receiver throws away the unneeded words in the burst before the pointed-to word and passes the desired word and all subsequent words to the CFDR.

As data enters the FPGA from an SDRAM read operation, it is registered and passed to one of a pair of dual-ported block RAMs called an *offsetable\_twin\_ram* (because there are two – they are

bank switched). The data is read out of the `offsetable_twin_ram` (or `twin_ram` for short) once it has filled up past some offset memory address corresponding to the value set in the **SDRAM Address0** and **SDRAM Address1** registers. In this way, a data read pointer can be set to a location that would be in the middle of a `rd_core` operation of bursts. Once data begins to flow out of a `twin_ram`, data will continue to flow as the SDRAM Receiver attempts to keep the `twin_ram` empty (while the SDRAM Arbiter keeps issuing read commands in an attempt to keep the `twin_ram` full). Data is taken out of the `twin_ram` until the CFDR is almost full and then the SDRAM Receiver waits to empty the `twin_ram` further until there is more room in the CFDR as it is more important not to overflow the CFDR than to keep the `twin_ram` empty.

On CF interrupts or unsuppressed PPSes when the read pointer changes, a `twin_ram` bank switch happens and data flows through the other `twin_ram`. This is an architectural complication from a previous incarnation of the SDRAM Receiver, which can likely be removed. But the system works as is with bank switched RAMs so it was left unchanged.

Note that bank switches happen after read operations are complete so that burst read data is not sent to two banks. Data left in a switched-from bank is unwanted and can be ignored. The write address to the `twin_rams` are reset to 0x0 on a bank switch; the read address is set based on the values in **SDRAM Address0** and **SDRAM Address1**. This starting read address is referred to as an *offset* – hence `offsetable_twin_ram`.

The word width change from 72 to 34 bits happens because the read and write ports of the `twin_rams` are not equal (read is 34 and write is 72).

The SDRAM Receiver attempts to keep the CFDR almost full (in fact, allows for only 8 empty locations) by looking at the read and write addresses of the CFDR. The SDRAM Receiver controls the write address and the read address is controlled by the `delay_gen` block in the DOM Back End which runs at RCLK rate. However, the address comparison logic runs at the SDRAM 80 MHz.

When the SDRAM Receiver is told by the Arbiter that the SDRAM is empty (i.e. the Arbiter is in the “FINISHED” state), the Receiver empties the current `twin_ram` and then signals the DOM Back End that the SDRAM is empty by asserting, yet another, “FINISHED” signal.

### *SDRAM Clock Module*

The SDRAM Interface requires a variety of clocks of high stability which are provided by the Xilinx Digital Clock Managers (DCMs). There are two DCMs in the SDRAM Interface.

DCM0 takes the 80 MHz signal from the external oscillator and replicates it for use in the FPGA. The key point to this DCM is that its feedback signal is a version of the 80 MHz clock that is fed to the board and then back into the FPGA into the DCM’s feedback pin. This effectively deskews the FPGA internal logic relative to the board to help eliminate board level clock skew. This external feedback clock has a long enough trace on the board to account for the skew of the data and control signals to the external SDRAM DIMM.

DCM1 is used to create a divided down 10 MHz clock from the 80 MHz clock. The 10 MHz clock is used in the refresh counter.

Both DCMs provide status bits in the **Status Register** to indicate when they have locked. The SDRAM Interface blocks should not be enabled until after the DCMs have locked as described in the various *Operation* manuals for the DOM. There is also a bit in the **Status Register** to indicate if the 80 MHz from the external oscillator has stopped, which would cripple the system.

## Back End

The DOM Back End is shown in Figure 6.

The DOM Back End is where the difference in the four operating modes of the DOM (Station Unit, VSI output, TVG, and TVR) come into play. Data enters the DOM Back End through the CFDR, a dual-ported RAM which is fed by the SDRAM Interface. From the CFDR the data can be passed to one of three data sinks, depending on mode: the Station Unit Output module, the VSI Output module, or the TVR. In the case of TVG mode, no disk based data is required, the DOM manufactures a test pattern in the TVG to be sent through the output port. The Back End also contains other sub-blocks specific to certain modes. The Correlator Frame Header RAM (CFHR) holds the correlator frame headers for station unit mode. The BOCF<sup>1</sup> Generator (BOCF\_gen) creates the BOCF signal for station unit mode and controls the size and rate of correlator frames. Common to all data sinks is the Delay Generation (del\_gen) sub-block which creates the addresses for reads of the CFDR in such a manner as to allow for skips or repeats of data based on a linear model. This feature is useful, for example, in removing some effects of the earth's rotation in VLBI data when in station unit mode. The Timing Subsystem is also common to all blocks and provides the RCLK and various PPS signals to blocks that need them in the DOM Back End.

### *CFDR*

The Correlator Frame Data RAM is a 128 element (34-bits wide : 32-data bits, 1 validity, 1 TOT) dual-ported RAM. It is addressed on the read side by the Delay Generation sub-block at the frequency of RCLK. The CFDR is addressed on the write side by the SDRAM Interface's SDRAM Receiver at 80 MHz. The data is written into sequential addresses and usually read out of sequential addresses (except for read address jumps caused by the Delay Generator). Unsuppressed PPSes (discussed below) or BOCFs cause both the read and write pointers into the CFDR to reset to 0. In general, the CFDR acts as a FIFO (except for the aforementioned address jumps and resets).

### *Delay Generation*

The Delay Generation sub-block simply generates read addresses to the CFDR. The Delay Generation block has an input read strobe and an output read strobe. Every cycle the input read strobe is asserted a new address is calculated and supplied to the CFDR. The address is calculated using a linear model based on the values in **Delay Error Reg0** , **Delay Error Reg1** which contain the *delay error* and the **Delay Rate Reg0** and **Delay Rate Reg1** which contains the *delay rate*. Normally the address to the CFDR increments by 1 for each input read strobe to the Delay Generator. The 18-bit delay rate is added repeatedly to the 32-bit *delay error* on each input read strobe. As this sum continues to grow it will produce a carry-out, 33<sup>rd</sup> bit. When the carry-out is produced the address stops incrementing by 1 and either increments by 2 or by 0 depending on the state of the <del\_gen\_mode> bit in **Delay Rate Reg1**. On an unsuppressed PPS in VSI or TVG mode, or on BOCF assertion in station unit mode, the address sum returns to 0 and a new delay model is loaded based on the values in **Delay Error Reg0** , **Delay Error Reg1**,

---

<sup>1</sup> Beginning of Correlator Frame.

**Delay Rate Reg0** and **Delay Rate Reg1**. The read address to the CFDR returns to 0 at this point as well.

### *Timing Subsystem*

The timing subsystem generates and distributes the system RCLK and PPSes.

The RCLK enters the FPGA from either the DPSCLK from the MDR-14 connector on the board, or the software can select to use the “internal” clock – a programmable oscillator located on the board. The selected clock source is then further divided down to the appropriate frequency as determined by the value <rclk\_rate\_code> in **RCLK PPS Rate Register**. The divided down clock is the system RCLK. The “internal” clock is configured via the **ICLK Control** register.

The PPS enters the FPGA as the DPS1PPS from the MDR-14 connector on the board and starts a hardware counter that produces an internal 1PPS signal based on <PPS\_div\_code> in **RCLK PPS Rate Register**. Alternatively, if no DPS1PPS exists in the system, the hardware PPS counter can be started on enabling of the timing subsystem if the <use\_internal\_pps> bit is set in the **RCLK PPS Rate Register**. In either case, a hardware counter that generates the system PPS which is simply started by an external signal (DPS1PPS or enable if <use\_internal\_pps>=1) ensures perfect periodicity of the PPS signal. The DPS1PPS input, if it exists, forms the DOM1PPS interrupt.

Three copies of this PPS signal are formed. One copy is stretched to match the output baud rate of the VSI output<sup>1</sup>. This copy is pipelined to match the pipeline of the VSI output data. Another copy is passed unchanged as the ROT1PPS interrupt and ROTMON signal. The third copy is used to trigger various events through the system (such as loading of a new SDRAM read pointer as described above). This third copy is a *suppressible* PPS and the value of the <suppress\_pps> bit field in the **System PPS Suppress Register** determines whether the pulse will be passed to the rest of the system as a trigger or will be suppressed. This feature is useful for starting output at an exact PPS. The <suppress\_pps> bit can be set or cleared during operation as needed to trigger various events (depending on mode – noted below).

A high level block diagram of the Timing Subsystem is shown in Figure 7

### BOCF Generator

The BOCF Generator simply creates a BOCF signal which goes to the output port for use in Station Unit mode as a signal to the correlator. The BOCF Generator is synchronized to the first unsuppressed PPS it sees after it is enabled. The length of the BOCF assertion time can be adjusted by software to be 240, 480, 960, or 1920 RCLK cycles in length. The deassertion time of the BOCF can also be specified in RCLK cycles and together with the assertion time, these parameters determine the BOCF frequency. Note that the Correlator Frame length is given by the RCLK cycles not by the number of data words per Correlator Frame. That is to say, if a `su0_prescl` (described below) of 2 is used, the number of RCLKs per Correlator Frame will

---

<sup>1</sup> Note that the PPS is only output in VSI output mode, not in Station Unit Mode. The TVG output PPS is on the R1PPS line but does not need to be baud-rate stretched as the TVG output rate always matches the RCLK rate.

be twice the number of data words per Correlator Frame. To calculate the number of RCLKs per BOCF, consider first the number of number of RCLKs per Correlator Frame, subtract the length of the BOCF, and then subtract 1.

Example:

2 Correlator Frames/second, suo\_prescl = 2, 32 MHz playback, and 480 RCLKs/BOCF.

⇒ The BOCF deassertion time =  $16,000,000 - 480 - 1 = 15999519$  RCLKs.

The BOCF Generator produces copies of the BOCF for internal use as well as a pipelined version to the FPGA output. The pipeline length accounts for the latency of the SU Output sub-block's output pipeline in such a way that the first header output and the rising edge of the BOCF are coincident in time at the FPGA output.

### *VSI Output*

The VSI Output sub-block simply takes data from the CFDR and passes it to the FPGA output which goes to the VSI output connector (MDR-80). The VSI Output block begins operation on the first unsuppressed\_pps it sees after being enabled, if <vsio\_run> is set to '1'. As long as all subsequent PPS signals are suppressed, data transfer will continue. On any subsequent unsuppressed\_pps data transfer will temporarily stop and the QVALID signal will be deasserted indicating the data lines at the VSI output connector hold invalid data. The read pointer to the SDRAM is now moved to a new address indicated by the **SDRAM Address0** and **SDRAM Address1** registers. After 90 RCLK cycles data will begin to flow out the FPGA beginning from this new read pointer and QVALID is reasserted. This allows discrete jumps in time at the one second interval. Note that the output ROT1PPS signal is pipelined significantly to match the data latency from the SDRAM. So when data from the new read pointer starts to flow out the FPGA the ROT1PPS signal will be pulsed. The timing diagram in Figure 8 illustrates the timing related to the PPS, data and validity of a discrete jump in time at the VSI Output. The output will go idle (data flow will stop) if an unsuppressed PPS occurs while <vsio\_run> is set to '0'. If the SDRAM buffer is completely emptied the VSI Output will go to its "FINISHED" state, where data flow is idle.

### *CFHR*

The Correlator Frame Header RAM (CFHR) is a pair of dual-ported 240x16 RAMs accessed by the software and the DOM Station Unit output module (below). The CFHR is broken into Bank A and Bank B. The software writes into one bank while the SU output module reads out of the other. Bank A holds Correlator Frame Header data for even BOCFs while Bank B holds Header data for odd BOCFs. Each of the 240 RAM locations in a bank holds a Header output word associated with all 16 output channels (elements are output time samples). Note that the bank to be read from by the SU output is alternated at each BOCF starting from Bank A<sup>1</sup>. Also, note that the toggling between Bank A and Bank B happens as long as the CFHR is enabled, whether or

---

<sup>1</sup> Actually, on reset the readout bank is Bank B, but the 1<sup>st</sup> BOCF immediately toggles the readout pointer to Bank A, so no header data is read from Bank B on startup. Header data for the 1<sup>st</sup> BOCF should be placed in Bank A.

not data is being output from the DOM. The number of BOCFs which have occurred since enabling of in the BOCF Generator, can be found by looking at the **DOM Correlator Frame Interrupt Counter** register.

### *SU Output*

The SU Output is another data sink for the CFDR. It takes data from the CFDR and passes it through to the VSI output connector (MDR-80). The SU Output block begins operation on the first unsuppressed PPS following the <suo\_run> bit being set (while the SU Output block is enabled). At this PPS the SU Output transmits data in the CFHR to the VSI Output during the BOCF high time. When the BOCF goes low, the SU Output continues transmitting data to the VSI output connector, but now the data source is the CFDR. The data source alternates between CFHR and CFDR while the BOCF is high or low respectively, as long as the <suo\_run> bit remains set. Once the <suo\_run> bit is cleared, data transmission stops at the end of the current correlator frame. Data transmission will be restarted on the next unsuppressed PPS after the <suo\_run> bit is re-set. If the SDRAM buffer empties completely, the SU Output block will transition to its “FINISHED” state and cease data output.

### *TVG*

The TVG or Test Vector Generator is an alternate data output source. The TVG is enabled via software and begins playing VSI Standard TVG pattern data out the VSI output following an unsuppressed PPS. Data is played back one 32-bit data word per RCLK. The VSI Standard alternative baud rates (multiple RCLKs per data word) are not implemented.

### *TVR*

The TVR or Test Vector Generator is an alternate data sink. Recorded TVG pattern can be verified using the TVR. The TVR takes data from the CFDR on TVR software enable. An unsuppressed PPS is needed only once to load delay parameters and an SDRAM start address. However, the unsuppressed PPS should happen before the TVR is enabled. All delay parameters should be set to 0.

The TVR provides feedback to the software as to the quality of the TVG pattern it sees via four software registers and a software interrupt. The software interrupt is generated each time the TVR sees the TOST data word (the data is marked in the internal data path has having been taken-on the second tick – TOST). When the interrupt is generated a full second of data has been processed. The TVR only processes statistics for one of the 32 bits in the data word each second. The bit index can be changed via the software using the **TVR Bit to Sum Reg**. At each New\_TVRSums\_INT, the bit indicated by the value in the **TVR Bit to Sum Reg** is set to be processed in the subsequent second. The statistic for that bit’s TVG pattern are ready the following interrupt. That is to say, a bit index is set by software, the index is recognized by the FPGA on the next New\_TVRSums\_INT, one second of data is processed for that bit index, and on the subsequent New\_TVRSums\_INT statistics for the previous second are ready.

TVR data statistics for a given index bit appear in the **TVR Sum Reg0**, **TVR Sum Reg1**, **TVR Bias Reg0**, and **TVR Bias Reg1** registers. The first two registers hold the error weight of the data stream relative to the expected TVG pattern at the bit index previously loaded in the **TVG Bit to Sum Reg** over the previous second. The **TVR Bias Reg0**, and **TVR Bias Reg1** registers hold a 2's complement value indicating the DC bias of the bitstream indicated by the index register over the previous second. These registers are large enough to accommodate 32 Ms/s data.

Note that a `New_TVRSums_INT` will occur immediately upon enabling of the TVR sub-block because the first data word read off the StreamStor disk is a TOST word.

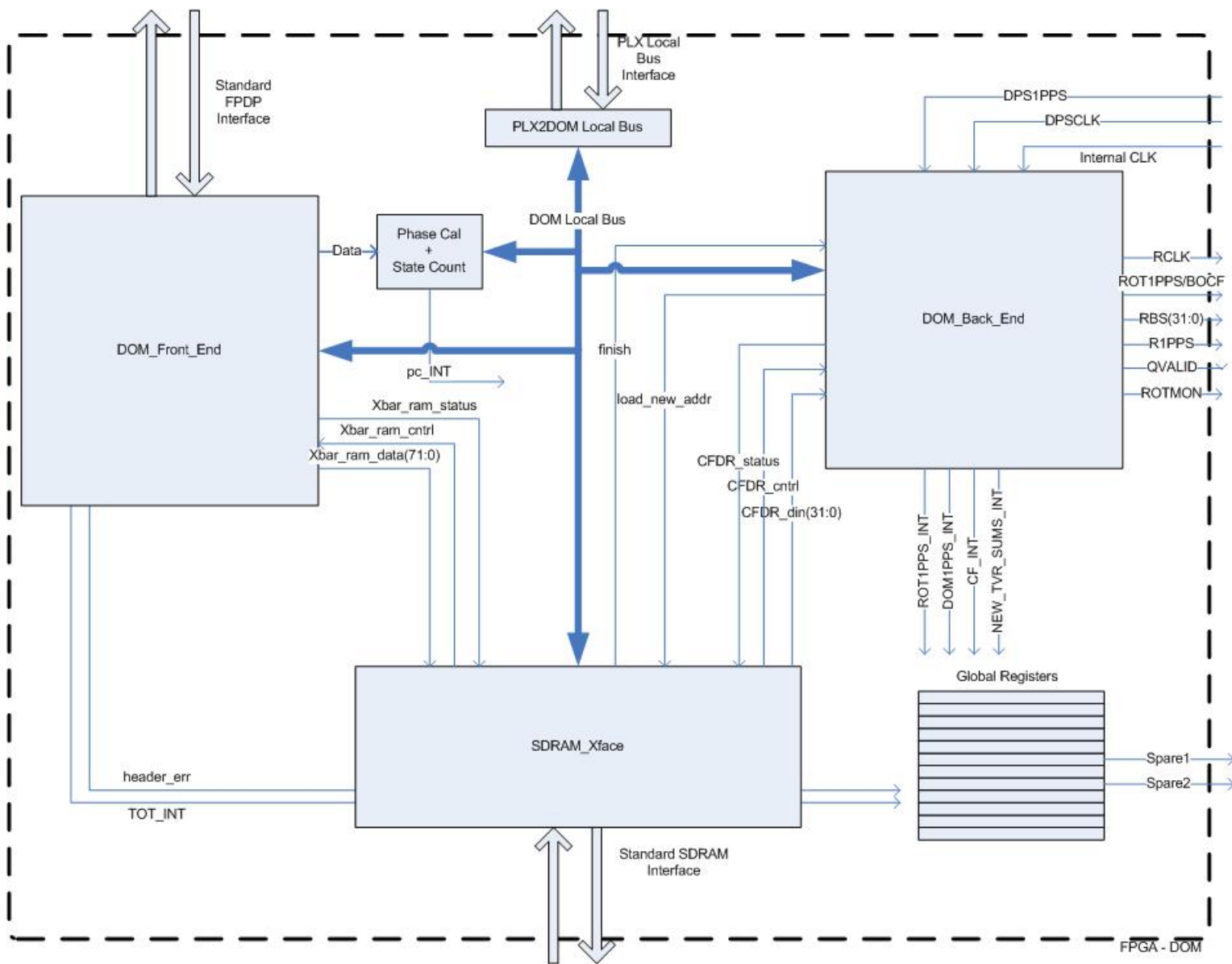
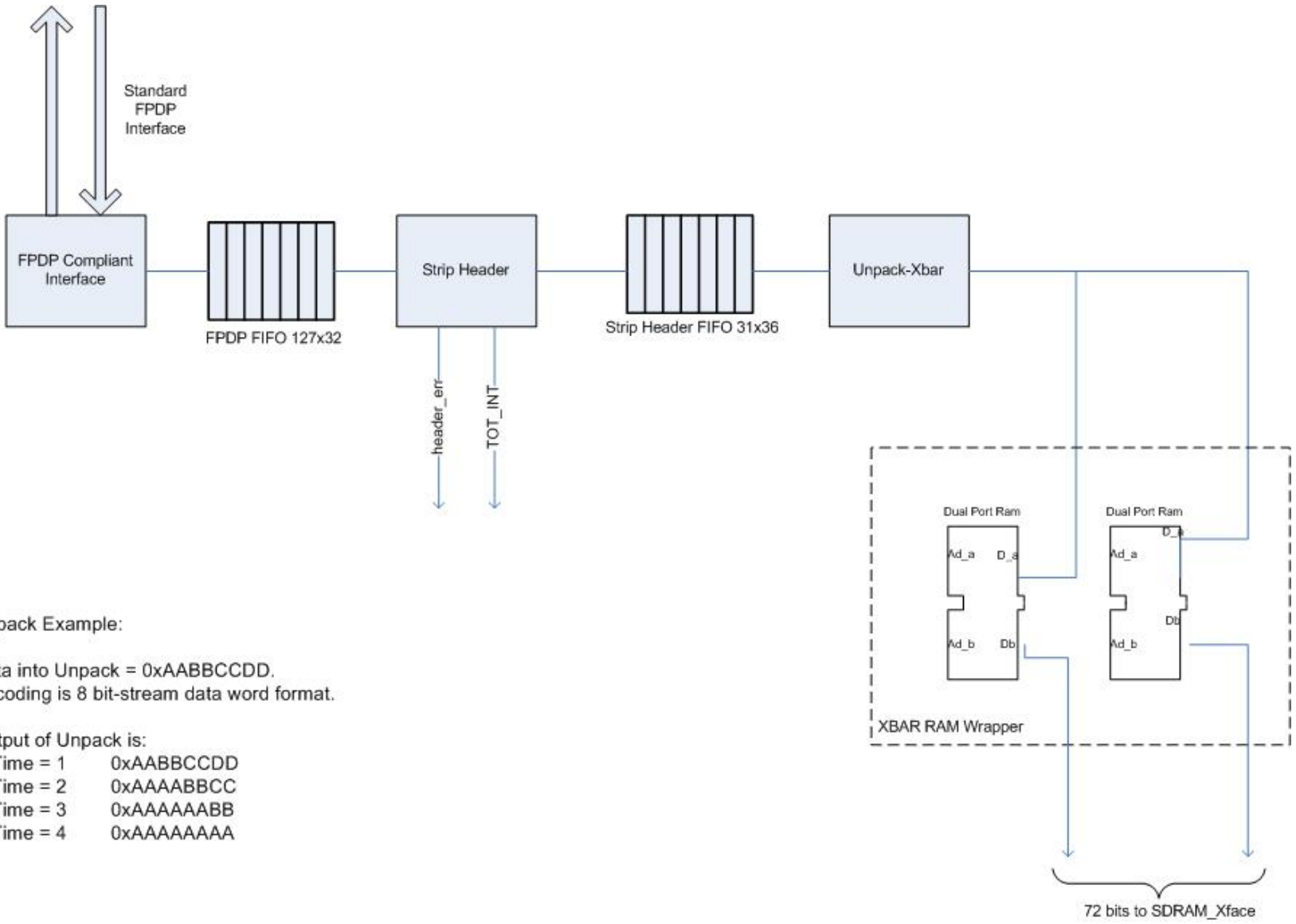


Figure 1 DOM Major Functional Blocks – border line indicates FPGA boundary.



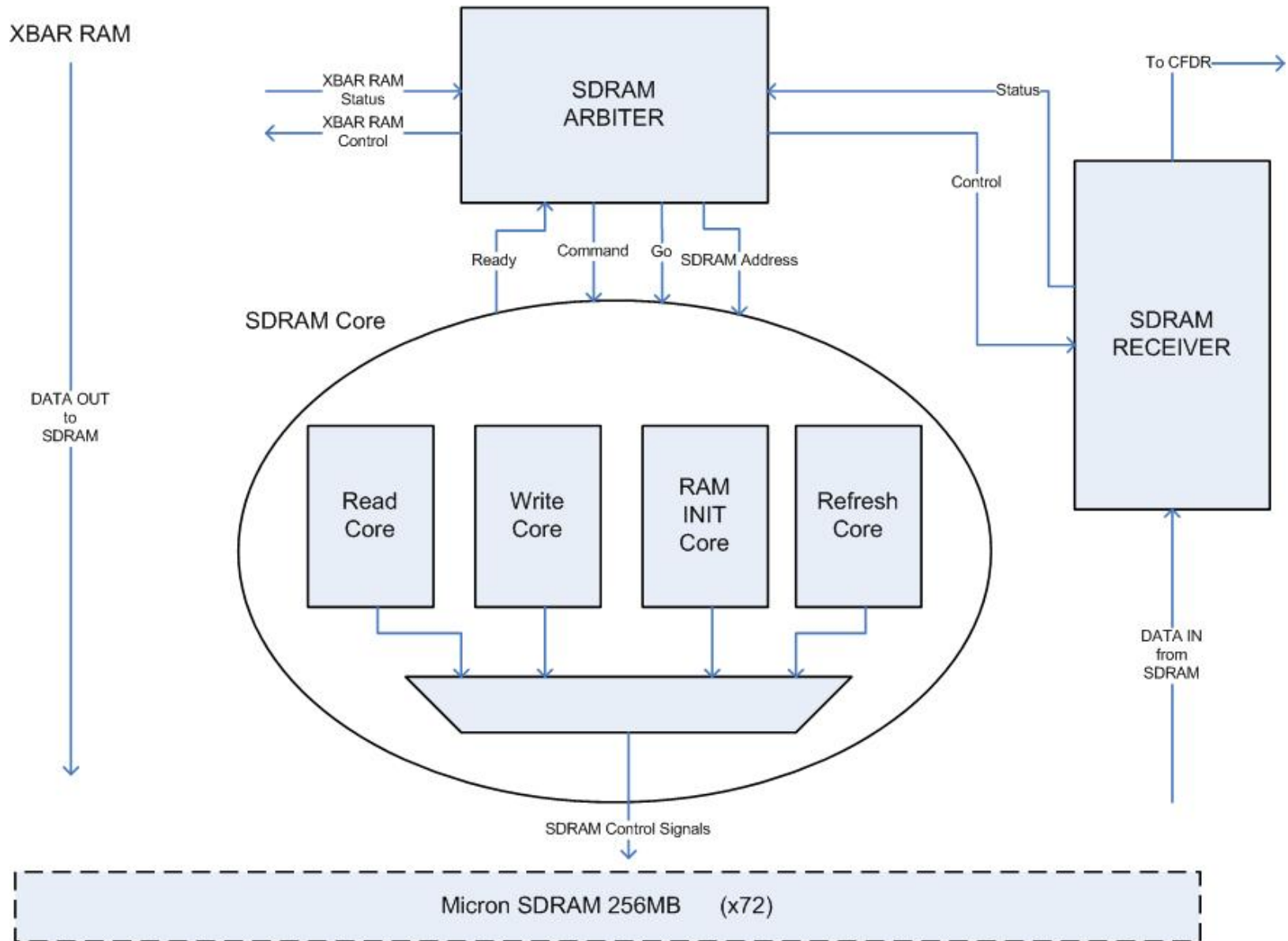
**Unpack Example:**

Data into Unpack = 0xAABBCCDD.  
 Encoding is 8 bit-stream data word format.

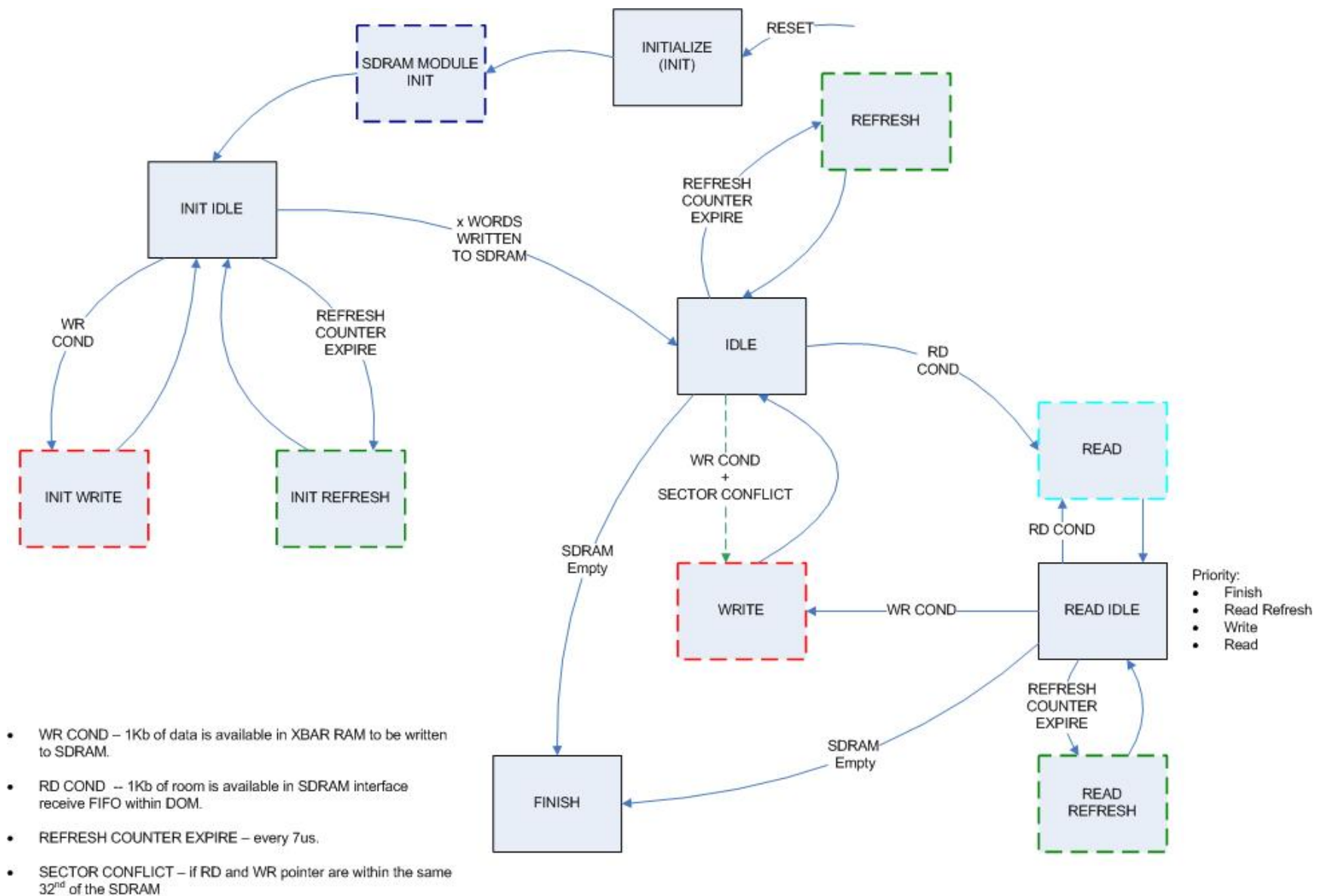
Output of Unpack is:

@Time = 1	0xAABBCCDD
@Time = 2	0xAAAABBCC
@Time = 3	0xAAAAAABB
@Time = 4	0xAAAAAAA

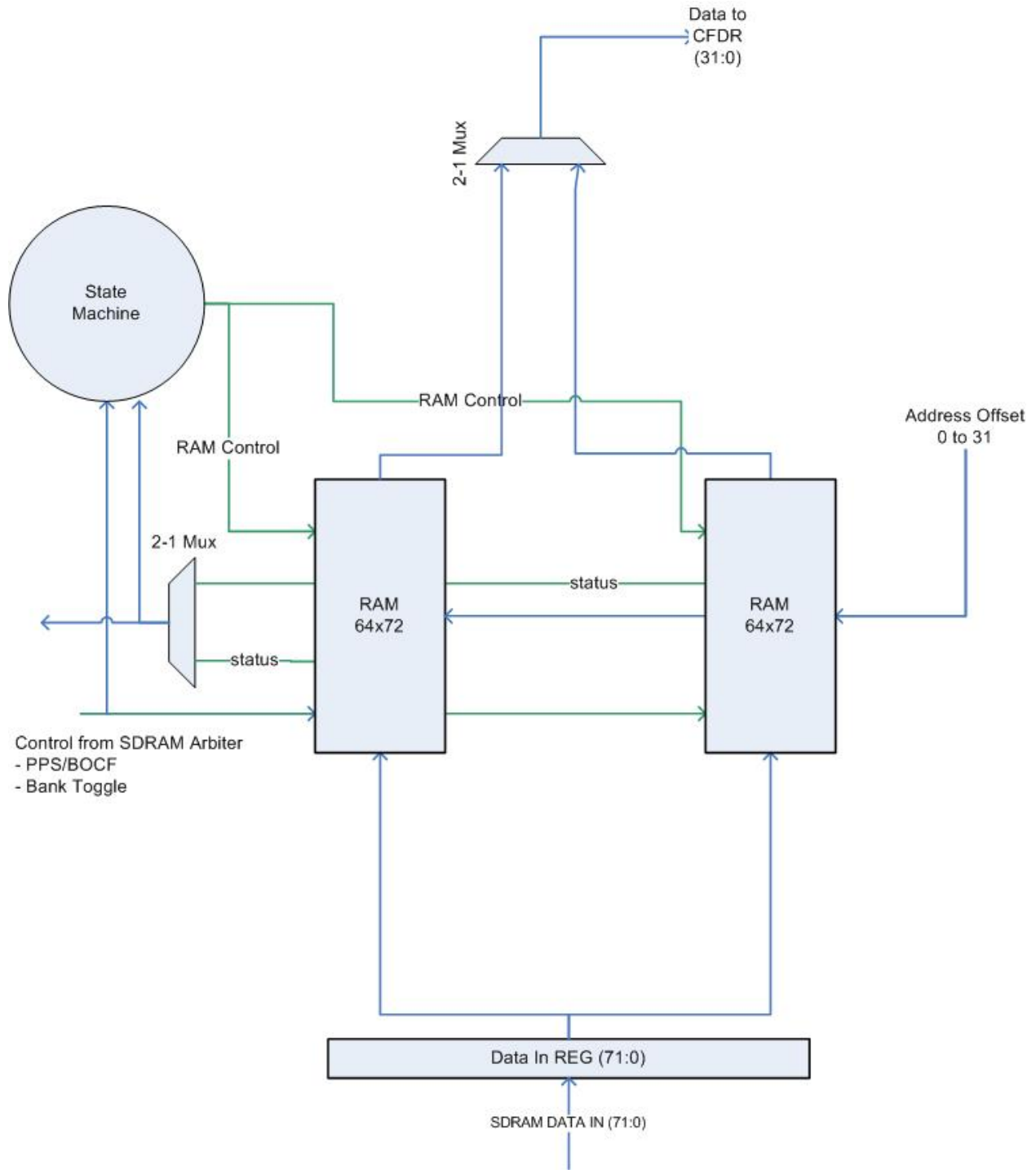
**Figure 2 DOM Front End Data Path**



**Figure 3 SDRAM Interface**



**Figure 4 SDRAM Arbitrer State Machine**



**Figure 5 SDRAM Receiver**

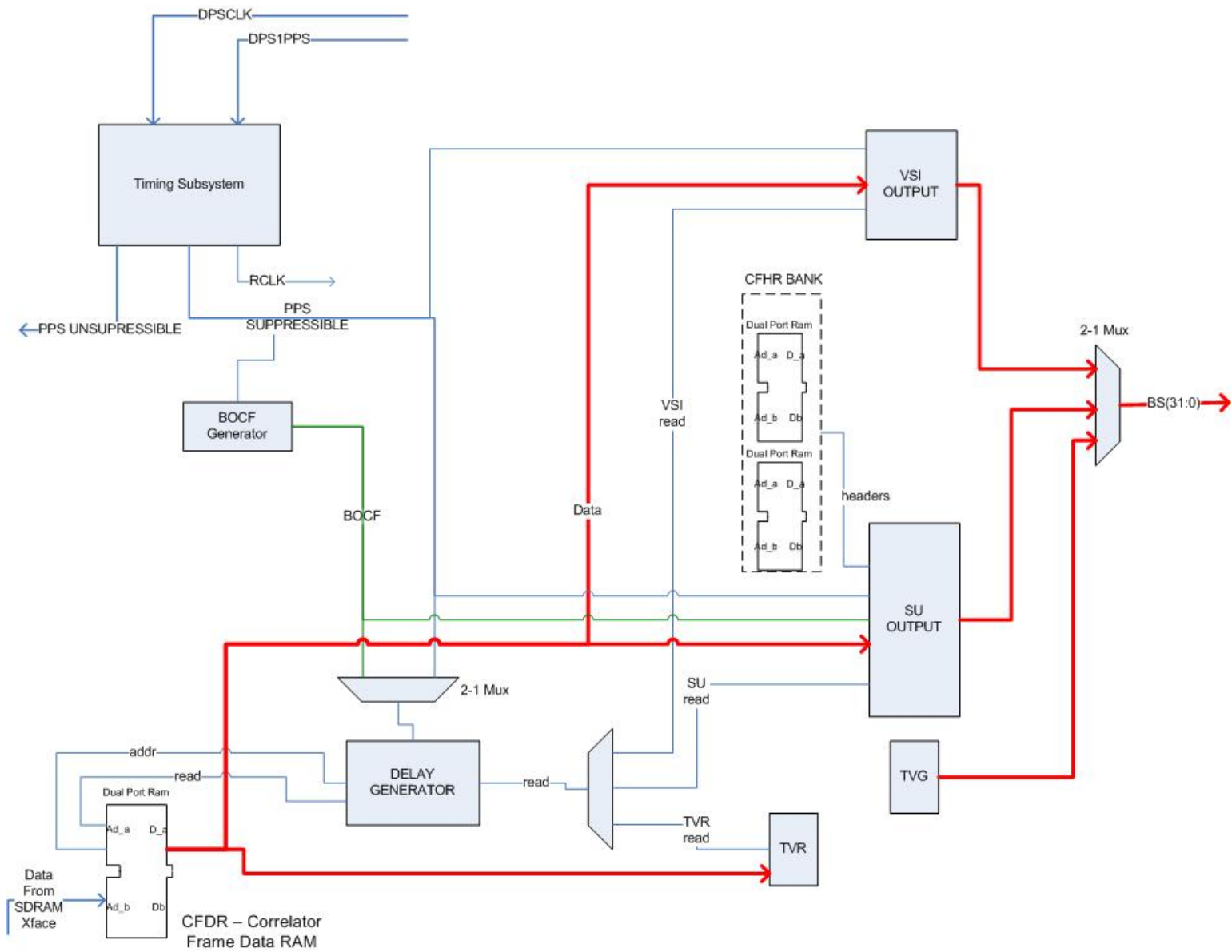


Figure 6 DOM Back End Data Path

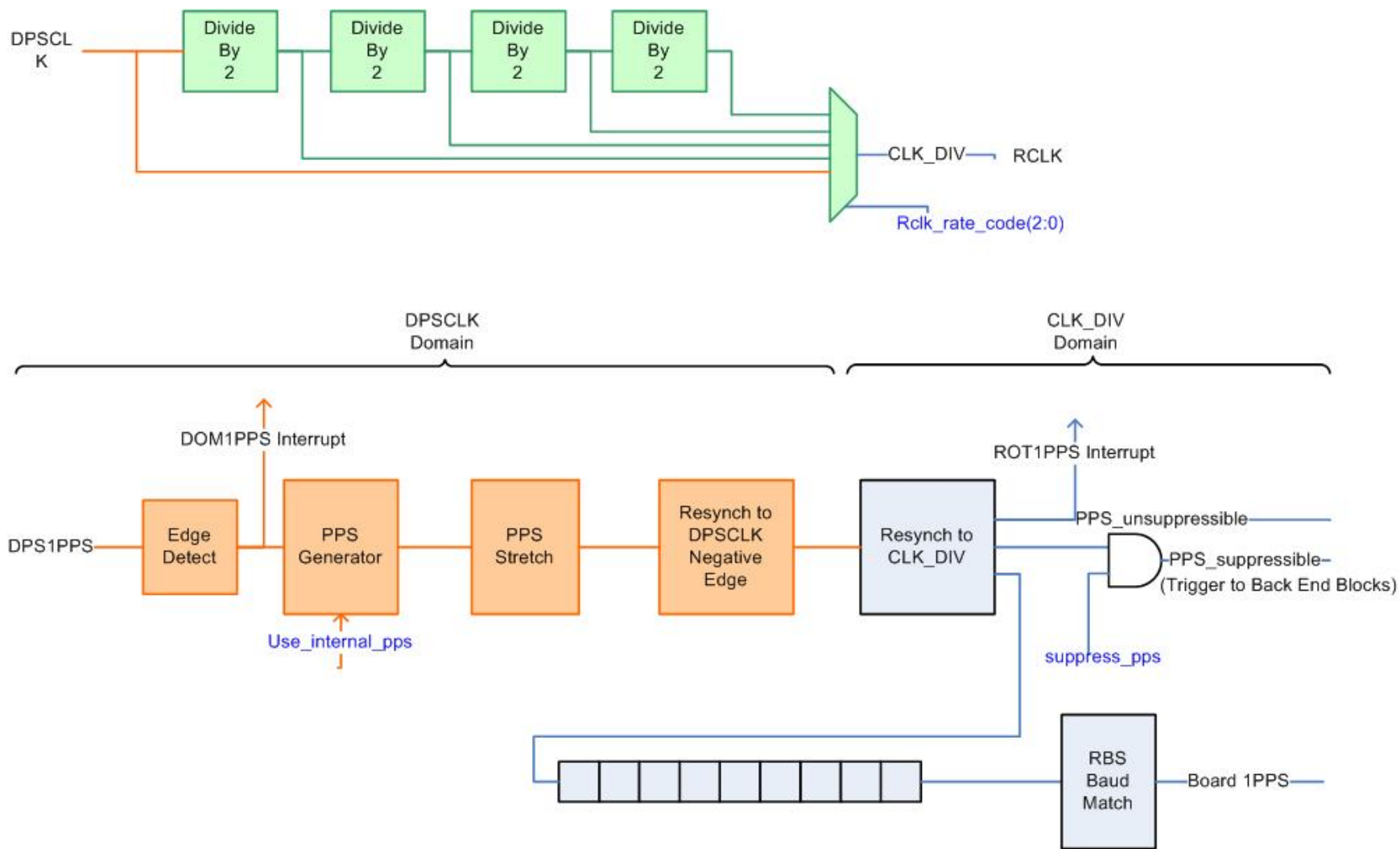
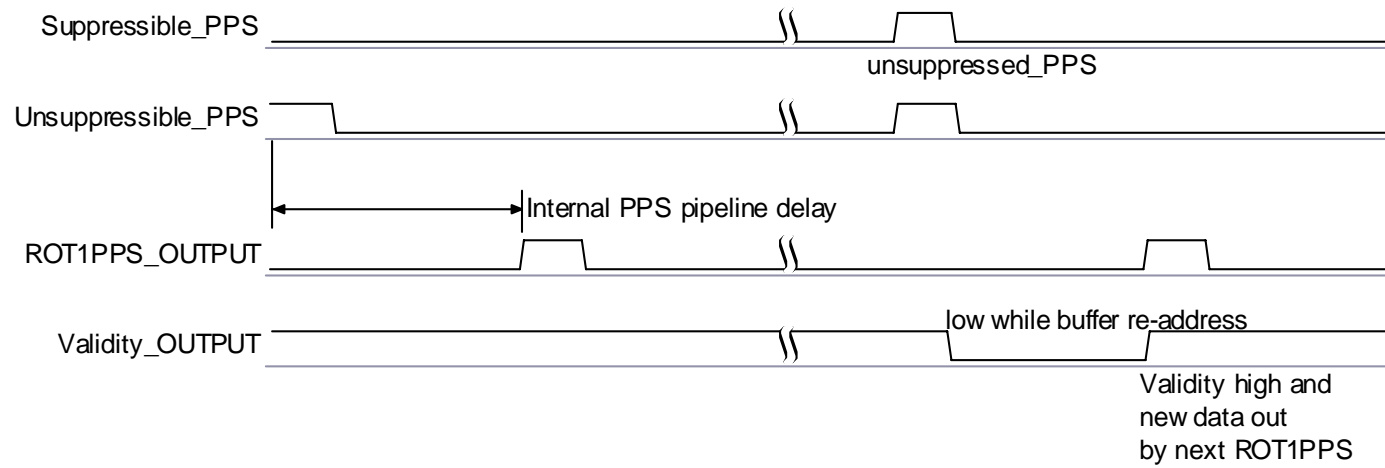


Figure 7 Timing Subsystem



**Figure 8 VSI Output Timing Diagram**