Texas Instruments

TSW 1400

ADC FIRMWARE DESIGN DOCUMENT

11th DEC. 2011

TSW 1400 is a next generation of pattern generator and data capture card used to evaluate performance of different high speed Analog to Digital (ADC) and Digital to Analog Converters (DAC). For the case of an ADC, by capturing the sampled data over an LVDS/CMOS interface, TSW 1400 can be used to match ADC performance against the data sheet. Together with the accompanying GUI, it is a complete system to capture as well as assail the data samples. TSW1400 also enables the user to generate and send the desired pattern or data samples to a DAC. A block diagram of the system is shown in Fig. 1.

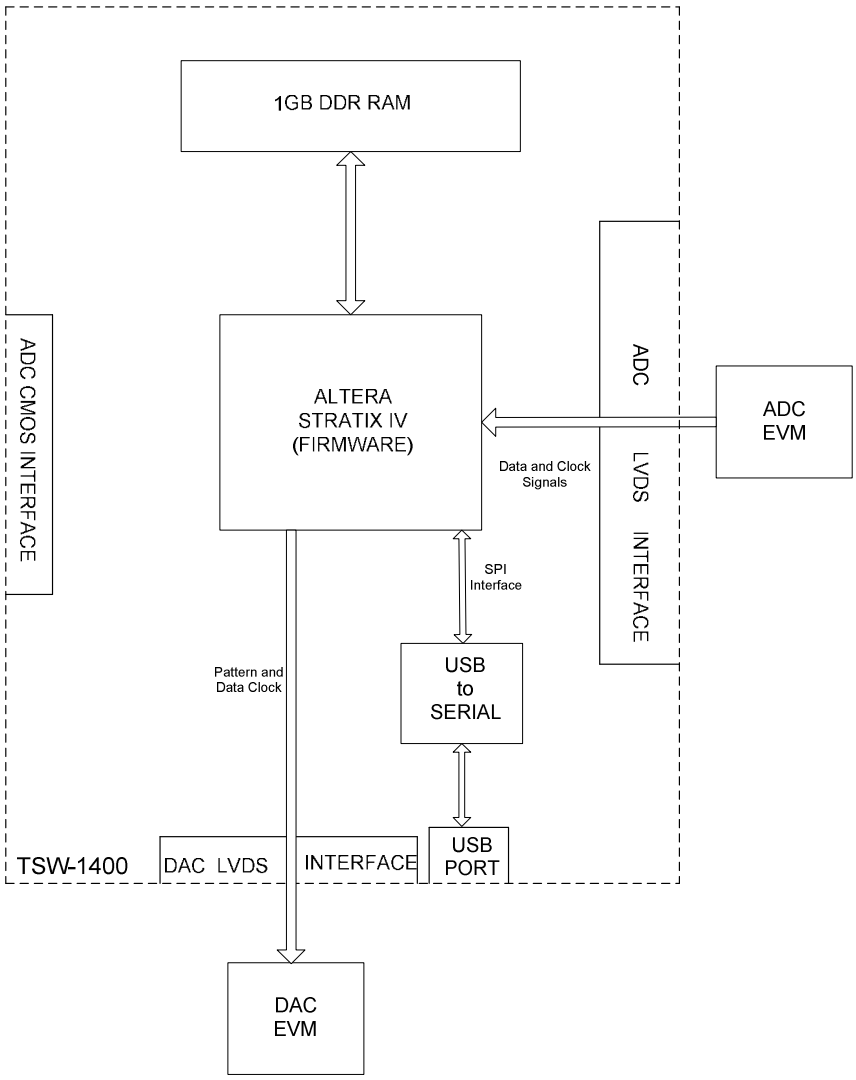

Fig. 1. Block diagram of TSW 1400

- Analog to Digital Converter (ADC)

As shown in the block diagram, TSW1400 is plugged into an Evaluation Module (EVM) for an ADC through LVDS (or CMOS) interface. Sampled data from the ADC is then taken by the firmware in ALTERA Stratix IV FPGA which after deserializing and formatting the data, stores it into an external onboard 1GB DDR RAM. It is this onboard memory which enables TSW1400 to store up to 512M data samples each of 16-bits. To acquire data, firmware in the FPGA reads the data from memory and transmits it on Serial Peripheral Interface (SPI). An onboard FTDI chip FT4232H which is a high speed USB 2.0 to UART/MPSSE converter carrying a Multi-Protocol Synchronous Serial Engine takes data from SPI and transmits it to the USB port.

TSW1400 comes with a GUI that acquires data from USB port and displays the sampled data in time and frequency domain as well as computes various performance related parameters such as Signal to Noise Ratio (SNR) and Spurious Frequency Dynamic Range (SFDR).

Below is provided the description of firmware design for LVDS ADCs. The design for the CMOS case is very similar.

- FPGA Firmware

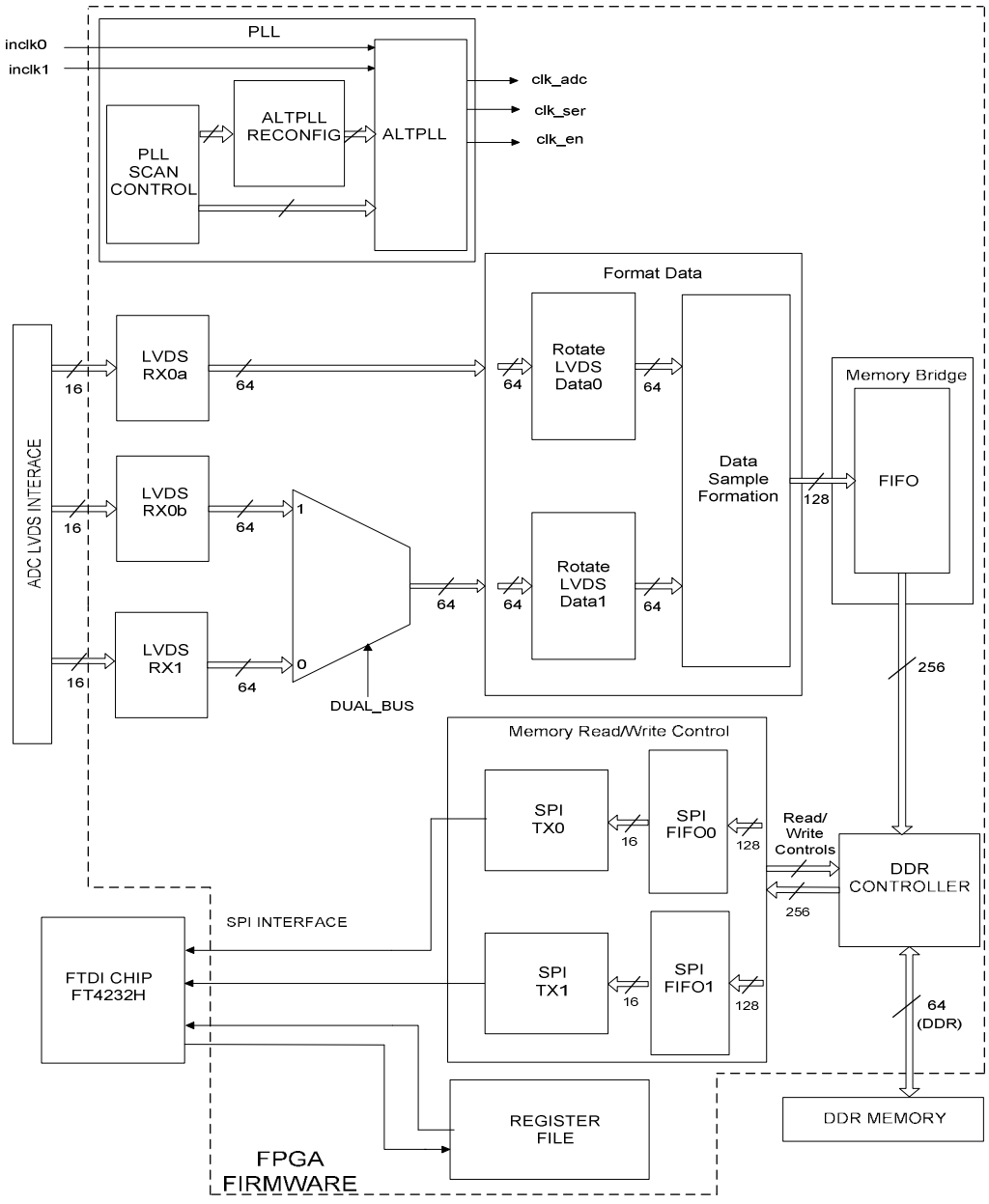

Fig. 2. Firmware block diagram

Firmware in the FPGA is the Verilog code that performs all the necessary tasks to deserialize the data received from ADC and then reformat it into individual samples. The firmware also communicates to the FTDI chip to send the data to the GUI. Fig. 2 shows various modules of firmware and the flow of data through them. A detailed description of these modules is provided below.

- Top Level Module

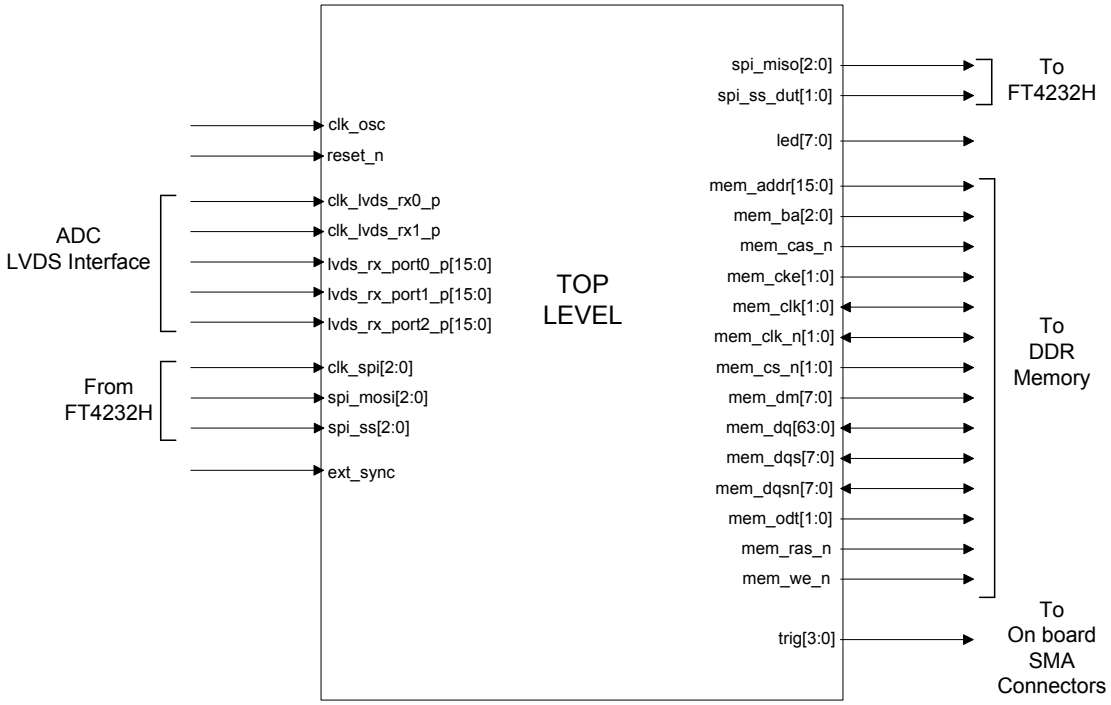

Fig. 3. Top level module *tsw1400_top*

Table 1
I/O description for *tsw1400_top*

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_osc	N/A	Input	100 MHz clock from onboard oscillator
reset_n	N/A	Input	Asynchronous signal used to reset the internal logic as well as FIFOs
clk_lvds_rx0_p	N/A	Input	DDR clock from the ADC
clk_lvds_rx1_p	N/A	Input	DDR clock from the second ADC (for future support)
lvds_rx_port0_p[15:0]	clk_lvds_rx0_p	Input	Data from ADC
lvds_rx_port1_p[15:0]	clk_lvds_rx1_p	Input	Data from the second ADC(for future support)
lvds_rx_port2_p[15:0]	clk_lvds_rx0_p	Input	This port provides additional 16-bit input data for DUAL BUS ADCs
clk_spi[2:0]	N/A	Input	SPI clocks from FT4232H for the three SPI interfaces
spi_mosi[2:0]	clk_spi[2]	Input	spi_mosi[1:0] are unused.. spi_mosi[2] is the SPI interface used to program the register file.
spi_ss[2:0]	clk_spi[2:0]	Input	spi_ss[1:0] are the slave select signals for the two SPI interfaces used to transmit ADC data samples to the user interface. spi_ss[2] is the slave select signal for the SPI interface used to program the register file.
ext_sync	N/A	Input	If enabled, it is an external trigger used to start data capture from the ADC.
trig[3:0]	N/A	Output	The four ports to which either the external

			trigger or the software trigger is routed
spi_miso[2:0]	clk_spi[2:0]	Output	spi_miso[1:0] are used to transmit data samples to the two SPI interfaces. spi_miso[2] is used by the user interface to read the configuration registers in the register file.
spi_ss_dut[1:0]	-	-	Unused
led[7:0]	N/A	Output	This is connected to the seven LEDs on TSW1400 to provide visual status of various signals. led[1:0] : SPI slave select signals spi_ss[1:0], used to indicate transmission of data samples over SPI interface led[2] : System reset led. led[3] : PLL lock indication corresponding to the clock from the first ADC led[4] : PLL lock indication corresponding to the clock from the second ADC (for future support) led[5] : Indicates that memory interface is ready to use after memory initialization is complete. led[6:7] : Indicates that the two SPI FIFOs are empty

Note that all the outputs starting with *mem* have been derived directly from the altera DDR2 SDRAM Controller megafunction. If interested, reader is referred to the corresponding altera documentation for description of those signals.

- Phase Locked Loop[1]

PLL used is the altera megafunction *altpll* which provides the required clocking. ADC gives data over LVDS interface with respect to a DDR clock *inclk0* from which PLL generates a sampling clock *clk_ser* twice as fast that samples the data at every rising edge. As can be noted from the Fig. 2, two clock inputs exist at the PLL. The second clock input *inclk1* is tied internally to zero and has been used to extend the PLL input frequency lock range. Furthermore, two more clocks at the output are the result of PLL being used in source synchronous mode. Since clock and data at the input arrive at the same time from ADC, this mode allows the same phase relationship to be maintained between the clock and data at the output. These clocks are sinked as shown in Fig. 4(a) and the corresponding timing diagram is shown in Fig. 4(b).

[1] For CMOS case, the internal logic is driven by the same clock that is provided by the CMOS ADC. No separate PLL is necessary as the supported CMOS ADCs run at much slower rate than their LVDS counterparts

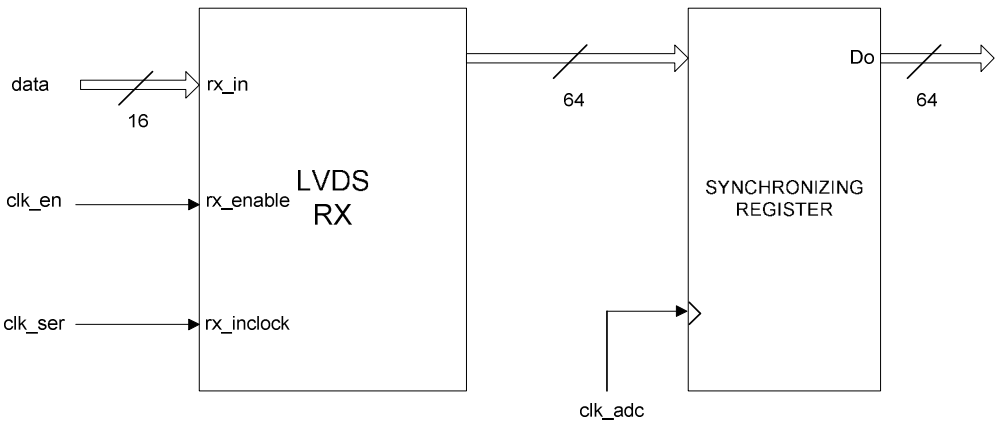

Fig. 4(a). Clocking for source synchronous mode

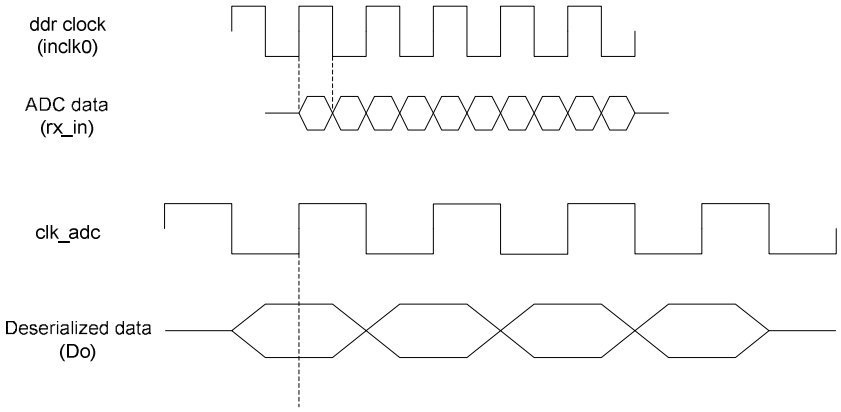

Fig. 4(b). Phase relationship between clock and data

Note that the source synchronous mode for LVDS compensation aligns the clock at the center of data eye. For the ADCs which already send centrally aligned data, a different phase shift setting is used for the three output clocks (see appendix A). For further information on using LVDS receiver with external PLL, reader is referred to altera "Clock Networks and PLLs in Stratix IV devices" documentation as well as altera design example on "Using altlvds With External PLL Option".

In the firmware, a module *ipll_top* serves as the top level module for the PLL. As shown in Fig. 2, this module contains various sub modules including the *altpll* megafunction as well as the modules required to reconfigure the PLL.

PLL Reconfiguration

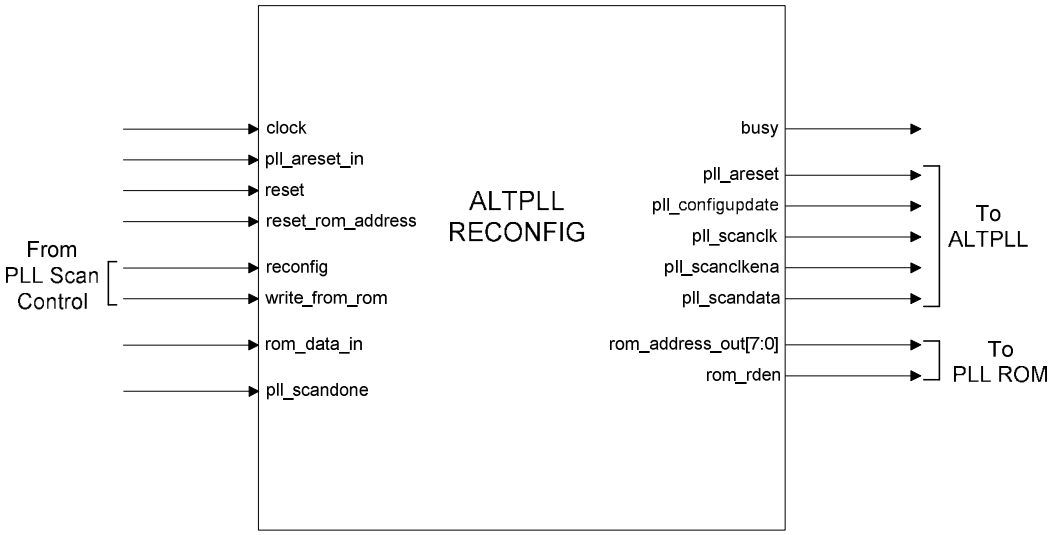

Fig.5. PLL reconfiguration module *ipll_reconfig* [2]

PLL can also be reconfigured in real time using altera *altpll_reconfig* megafunction in order to change the PLL frequency lock range to support Texas Instruments wide range of high speed ADCs with different sampling rates. Reconfiguration block is driven by a state machine implemented in *ipll_scanctrl* module.

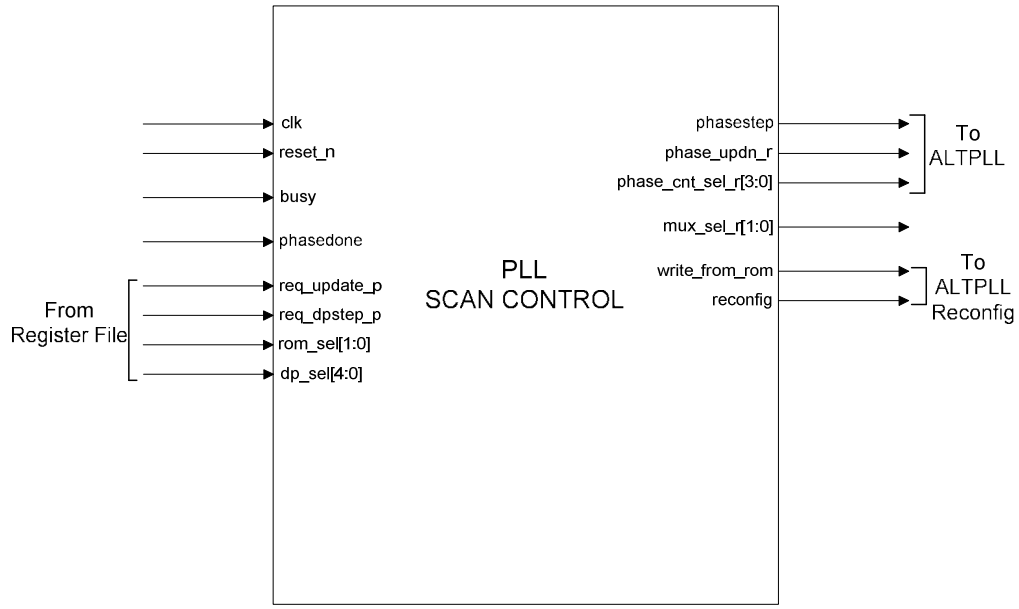

Fig. 6. PLL scan control module *ipll_scanctrl*

[2] For the description of I/O ports, see altera documentation on ALTPLL RECONFIG megafunction.

Table 2
I/O description for *ipll_scanctrl* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk	N/A	Input	This clock is sourced from clk_scan port of altera DDR controller and is 66.66MHz i.e. 2/3 of clk_osc
reset_n	N/A	Input	Asynchronous reset signal. Resets the internal state machine
busy	clk	Input	Connected to busy port of altpll_reconfig megafunction
phasedone	clk	Input	Connected to phasedone port of altpll megafunction.
req_update_p	clk	Input	Asserted for the PLL reconfiguration request. This is a one shot signal generated from the pll_req_rc port of the register file
req_dpstep_p	clk	Input	Asserted for the PLL phase reconfiguration request. This is a one shot signal generated from the pll_req_dp port of the register file
rom_sel[1:0]	clk_sys	Input	This port selects one of the PLL ROMs during PLL reconfiguration depending upon the required clock frequency settings. Connected to pll_rom port of the register file
dp_sel[4:0]	clk_sys	Input	dp_sel[0] : Selects the direction of phase shift (increment or decrement). Connected to pll_dp_sel[0] port of the register file dp_sel[4:1] : Select the PLL counter for which the phase tap settings need to be reconfigured Connected to pll_dp_sel[4:1] port of the register file
phasestep	clk	Output	This signal feeds the pll_phasestep port of altpll megafunction
phase_updn_r	clk	Output	Selects the direction of phase shift (increment or decrement) as determined by dp_sel[0] input
phase_cnt_sel_r	clk	Output	PLL phase counter whose VCO tap settings need to be adjusted
mux_sel_r[1:0]	clk	Output	Selects the required PLL ROM as determined by rom_sel input
write_from_rom	clk	Output	Feeds the write_from_rom port of altpll reconfig megafunction
reconfig	clk	Output	Feeds the reconfig port of altpll reconfig megafunction

The reconfiguration block reprograms the pre and post scale counters of PLL for new counter clock frequency settings. The .mif file for each of the different pll lock ranges is first generated and stored in *ipll_configROM* which is an altera megafunction. Whenever reconfiguration request is generated by the user, *req_update_p* flag is asserted. This causes the scan control module to assert *write_from_rom* signal at which the reconfiguration module starts reading data from the rom selected by *mux_sel* output which in turn is generated by *rom_sel* user input. Afterwards, the *reconfig* flag is asserted and *ipll_reconfig* module starts reconfiguring the pll using *pll_configupdate* and *pll_scandata* signals.

The .mif file does not contain any information about the phase settings of the output clock. Every time PLL is reconfigured, the phase tap settings revert to what was originally mentioned in the *altpll* megafunction. Therefore, every time the PLL is reconfigured, in order to keep the same phase shifts of the output clocks, it is imperative to reconfigure all the phase taps as well. *altpll* megafunction provides the provision to dynamically change the phase settings of the PLL. Phase reconfiguration request is generated by the user through *req_dp_p* signal. Depending upon the user input through *dp_sel,* the scan control module selects the PLL counter *phase_cnt_sel_r* whose phase is to be adjusted and direction for the phase shift *phase_updn_r* and asserts the *phasestep* flag. For a more detailed description of the steps required to

reconfigure phase settings, reader is referred to altera's "Clock Networks and PLLs in Stratix IV devices" documentation.

- ADC Interface

ADC interface, *adc_if* module serves as the top level for LVDS RX and format data modules as shown in Fig. 2. It does three crucial jobs, deserializes the data from the ADC, aligns it with the correct frame clock sequence as well as formats the binary data from the ADC into a complete sample for each ADC channel. When the data arrives from ADC LVDS interface, before feeding it to the LVDS receiver, it is rewired into the same bit order as is coming from the output of the ADC and stored in an internal vector *rx0a_in.*. This is necessary due to the fact that the LVDS interface might not be physically connected to the ADC output in the same order as the output of the ADC itself. Note that how rewiring is done on the input data depends on the type of ADC plugged into the board as each ADC is different as well as has different interconnections with the LVDS interface. Two more vectors *rx0b_in* and *rx1_in* rewire and store data from the other two LVDS ports (see *LVDS RX* description).

LVDS RX

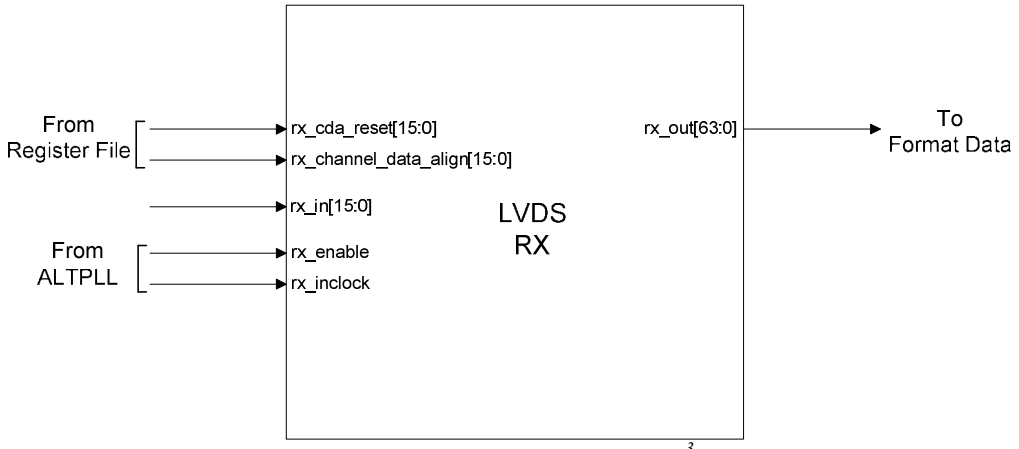

Fig. 7. LVDS receiver module *adcif_lvds* [3]

Firmware uses *altlvds* deserializer LVDS receiver, an altera megafunction that serves two purposes. It deserializes the data from ADC as well as performs the bit slip operation in order to align the word boundaries with respect to the ADC frame clock. Detecting correct frame clock sequence is imperative to tracking start and end of a sample. Since the 64-bit data (for a deserialization factor of 4) coming out of LVDS deserializer is not necessarily aligned with correct frame clock sequence, it is important to perform the bitslip operation. The bitslip request is generated by the TSW1400 GUI through register file by asserting an internal *bitslipx* flag which is connected to *rx_channel_data_align* port of the LVDS receiver block. The program keeps generating the bitslip request until it finds the correct frame clock sequence (which is different for different ADC). Once the frame clock is locked, only then the data is considered valid. An example timing diagram for the case of TI's ADS5281 is shown in Fig. 8.

[3] For the description of I/O ports, see altera documentation on ALTLVDS megafunction.

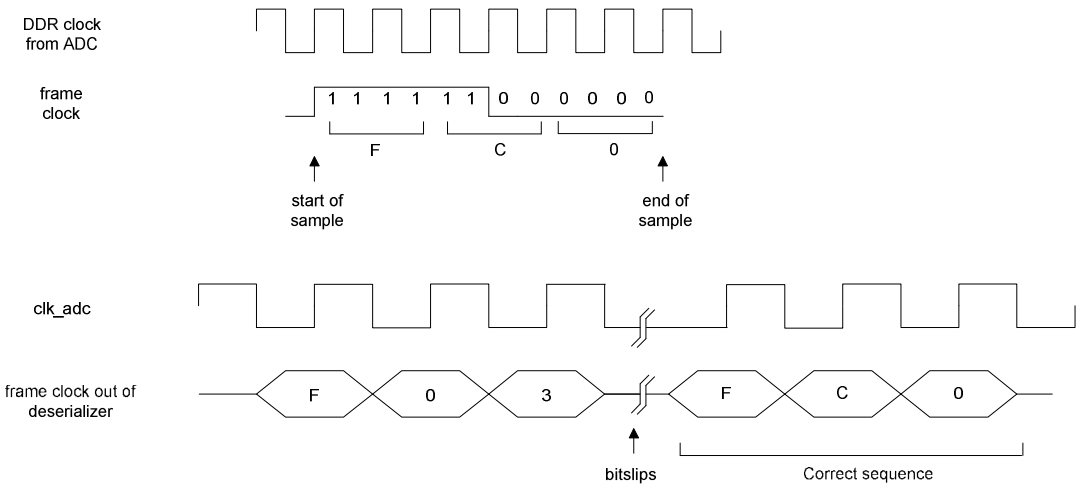

Fig.8. Timing diagram showing the search for correct frame clock sequence

Note that the bit slip operation is kept being performed till the correct frame clock sequence 0xFC0 is found. Frame clock from ADC arrives at bit position 10 of 3rd LVDS port *lvds_rx_port2_p* in the top level module *tsw1400_top* which in turn is tied to an internal signal *dinx[10]* in *adcif* module.

As can be noted in the top level *tsw1400_top* scaffolding in Fig. 3 as well as in the firmware block diagram in Fig. 2, there are three LVDS input ports. *lvds_rx_port0_p* is the port capturing the data from an ADC, port *lvds_rx_port1_p* is to provide the support for capturing the data from two ADCs simultaneously while the 3rd port *lvds_rx_port2_p* provides an additional port for the dual bus ADCs having wider than 16-bits output. For instance TI's ADS5400 is capable of transmitting data on two channels each of 16-bits. It is to be noted that *lvds_rx_port0_p* is always active while a multiplexer selects either *lvds_rx_port1_p or lvds_rx_port2_p* depending upon DUAL_BUS define directive.

Recall that the 64-bit data out of an LVDS RX has to be registered as shown in Fig. 4(a). This is done in the top level ADC interface module *adc_if*.

FORMAT DATA

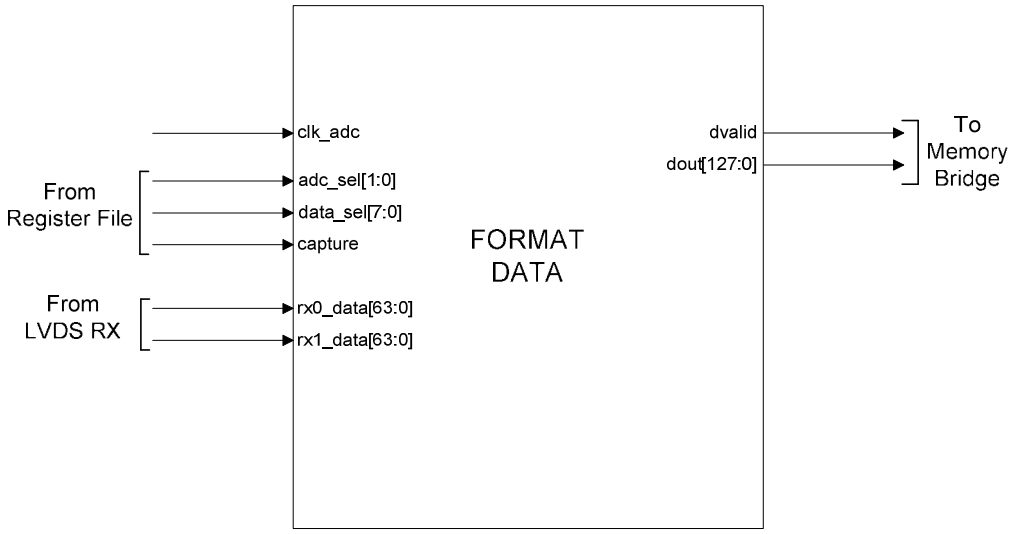

Fig. 9. Format data module *adcif_formatpdata*

Table 3
I/O description for *adcif_formatpdata* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_adc	N/A	Input	This clock, derived from the PLL, is half the rate of the DDR clock coming from the ADC (inclk0).
adc_sel[1:0]	clk_sys	Input	Connected to adc_sel port of register file
data_sel[7:0]	clk_sys	Input	Connected to data_sel port of register file
capture	clk_sys	Input	Connected to capture port of register file
rx0_data[63:0]	clk_adc	Input	Deserialized data from the LVDS RX0. This is the output of synchronizing register used to register data from an LVDS RX.
rx1_data[63:0]	clk_adc	Input	Deserialized data from the LVDS RX1. This is the output of synchronizing register used to register data from an LVDS RX.
dvalid	clk_adc	Output	Data valid signal
dout[127:0]	clk_adc	Output	Output data

This module is responsible for arranging the data from the deserializer into individual samples for every ADC channel. A module *adcif_rotatelvds* first rearranges the data from the deserializer. This rearrangement is necessary as the deserialized data has all the bits having same bit positions but arriving in different cycles placed together. This can be understood with Fig. 10(a) and 10(b). Note that the *adcif_rotatelvds* module rearranges the data in the desired order as shown in Fig. 10(b)

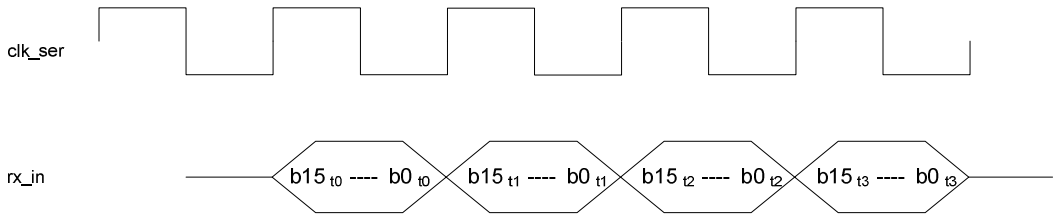

Fig. 10(a). Data at the input of desrializer *adcif_lvds*

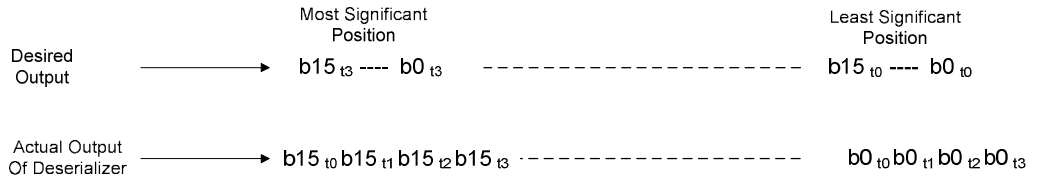

Fig. 10(b). Ouput of deserializer *adcif_lvds* and *adcif_rotatelvds* modules

For some ADCs, such as TI's ADS62p49, the polarity of DDR clock from the ADC is physically swapped on the LVDS interface. This causes a problem for instance in the case of ADS62p49 which always outputs even byte of the sample at positive edge followed by odd byte at the negative. However the LVDS receiver always samples the data at the positive edge of the *clk_ser* (which is double the rate of DDR clock from the ADC *inclk0*). Due to swapping of clock polarity, every time in the output of deserializer even byte of first of the two samples is missing. *adcif_rotatelvds* module also accounts for this clock swapping by extracting this even byte from a delayed internal vector *out_pre_d1_r*. The ADC for which this type of clock swapping has been done is selected by *data_sel* input in the format data module.

Output of the module *adcif_rotatelvds* gives the data in the same format as provided by the ADC except that it is deserialized. However, in order to form a complete sample of each ADC channel, reordering of the data bits is done outside *adcif_rotatelvds* within format data module. This rearrangement of the bits is different for different ADCs depending upon number of output bits from the ADC, order of the bits as well as number of channels. Table 4 describes various internal vectors used in *adcif_formatpdata* module to reorder bits of different types of ADCs.

Table 4
Vectors used to reorder data bits from different types of ADCs

Vectors	**Description**
dinx_intlv	This is used for the ADCs which provide even and odd bytes of a sample on alternate cycles. Even and odd bytes are interleaved together such that they form a complete sample
dinx_4w	This is used for the ADCs which at a time output 4 bits of every channel
dinx_2w_r	This is used for the ADCs which at a time output 2 bits of every channel
dinx_1w	This is used for the ADCs which at a time output only a single bit of every channel

After the reordering of bits, the data becomes arranged in the conventional bit order (MSB in most significant and LSB in the least significant position). Since the code also provides support to capture data from two ADCs or for a single dual bus ADC as mentioned in LVDS RX description, all the vectors in Table 4 are indexed 1 and 0. Vectors with index x =0 are used for the ADC whose data is received on *lvds_rx_port0_p* (input to the top level module *tsw1440_top*) while those with x=1 are associated with the ADC data received on either *lvds_rx_port1_p* for the case of two ADCs or *lvds_rx_port2_p* for the case of a single dual bus ADC. A vector *adc_sel* which is programmed by the user through the register file specifies whether to transmit samples from a single ADC or both ADCs.

Note that output *dout* of *adcif_formatpdata* is 128 bits wide. For the case of two ADCs, the lower 64-bits correspond to the ADC connected to *lvds_rx_port0_p* while the upper 64-bits correspond to the ADC connected to *lvds_rx_port1_p*. For the case of a single dual bus ADC, for instance TI's ADS58C48, the lower 64-bits correspond to channels A,B,C,D while the upper 64-bits correspond to channels E,F,G,H with channel A in least significant position and H in most significant. If only a single ADC is used without any dual bus operation, *dout* is still a 128 bit vector, however, now the data is valid on alternate cycles.

The assertion of data valid signal *dvalid* in *adcif_formatpdata* module follows the description of *dout* mentioned above. After the frame clock is locked, and if *capture* command has been sent, *dvalid* signal is high on every clock cycle for the case of two ADCs or a single dual bus ADC while it is high on alternate cycle for the case of a single ADC with out dual bus operation.

Frame Clock Hunt

The data format module also performs the vital job of sending frame clock sequence to the TSW1400 GUI (see LVDS RX section for a detailed description of frame clock search). The GUI, through the register file, asserts the flag *data_sel[3]* when it is searching for the frame clock. On the assertion of this flag, *adcif_formatpdata* sends the frame clock sequence on the data bus *dout*. The frame clock sequence is interleaved with the data samples on alternate words[4]. Table 5 describes correct frame clock sequence for different Texas Instrument's ADCs and the associated vectors in the *adcif_formatpdata* module which contain the actual sequence.

[4] Note that once the frame clock is locked, no frame clock sequence is sent. Only data samples are sent to the GUI

Table 5
Correct frame clock sequence for various TI's ADCs

ADC	Correct Frame Clock Sequence	Associated Vector
ADS6443	Correct Sequence 'F0',	*din0_2w_frm*
ADS5281	Correct Sequence 'FC0'	*din0_1w_frm*
ADS5493	Correct Sequence '3'	*din0[19:16]*
ADS642x	Correct Sequence 'E3' or '38'	*din0_2w_frm*

Number of frame clock bits required to form the correct sequence depend upon the serialization factor of an ADC. Note that only those ADCs have been mentioned in Table 5 for which frame clock is necessary to detect start of a sample. The GUI tries to find the corresponding sequence of the device or any possible combination of that sequence (for instance, for ADS5493, FC0 C0F or 0FC are all valid to lock to the frame clock, similarly for other ADCs). Once a valid frame clock sequence is found by the GUI, it deasserts *data_sel[3]* flag causing the module logic to start sending data samples. It must be noted that even though frame clock can be locked at any possible combination of corresponding sequence in Table 5, the data is latched by the *adcif_formatpdata* module only AT the sequence given in the table.

- Memory Bridge

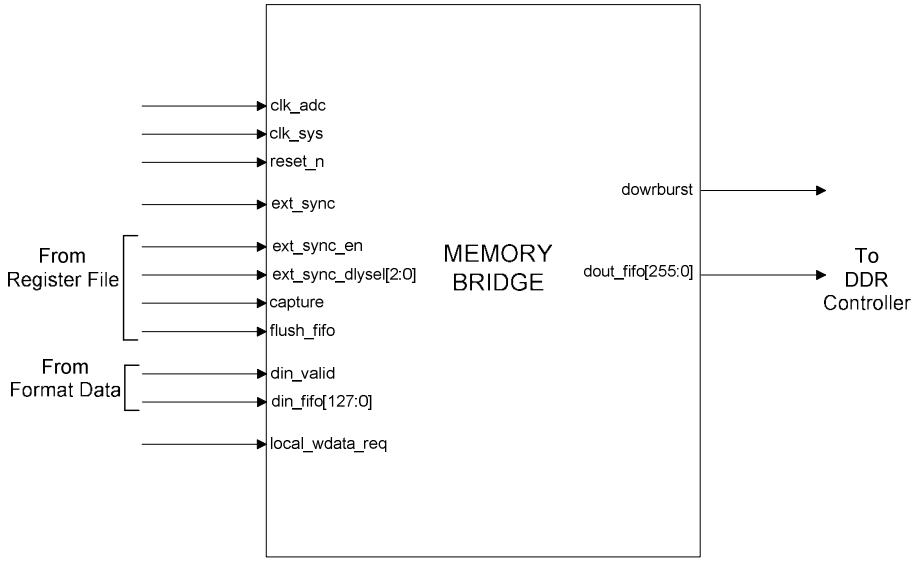

Fig. 11. Memory Bridge module *dumpmem_bridge*

Table 6
I/O description for *dumpmem_bridge* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_adc	N/A	Input	This clock, derived from the PLL, is half the rate of the DDR clock coming from the ADC (inclk0).
clk_sys	N/A	Input	Since the altera DDR controller is used as a half rate controller, this clock is 166 .66 MHz which is half the rate of the system clock (clk_sys port of memory read/write control module *dumpmem*)
reset_n	N/A	Input	Asynchronous reset signal.
ext_sync	N/A	Input	Connected to ext_sync port of top level module tsw1400_top
ext_sync_en	clk_sys	Input	Connected to ext_sync_en port of the register file
ext_sync_dlysel[2:0]	clk_sys	Input	Connected to ext_sync_dlysel port of the register file

capture	clk_sys	Input	Connected to capture port of register file
flush_fifo	clk_adc	Input	Clears the internal FIFO. It is a delayed version of flush_fifo[2] port from the register file
din_valid	clk_adc	Input	Data valid signal
din_fifo[127:0]	clk_adc	Input	Data
local_wdata_req	clk_sys	Input	This signal is asserted when data is to be written in the memory. It serves as read request signal for the internal FIFO. Obtained by ANDing local_write_req and local_ready signals from memory read/write control and ddr controller modules repectively.
dowrburst	clk_sys	Output	Asserted if there are atleast four read words in the internal FIFO
dout_fifo[255:0]	clk_sys	Output	Data from the internal FIFO

This module contains a FIFO that acts as a bridge between format data module and DDR controller. FIFO starts capturing data when *capture* flag is asserted. If the external trigger enable flag *ext_sync_en* is asserted, the module does not capture the data until an external trigger *ext_sync* is applied through the TRIG port on TSW1400. It should be noted that after the external trigger is applied, capture command must still be asserted to start data capture into the FIFO.

Data is read from the FIFO on to *dout_fifo* bus whenever write request *local_wdata_req* is asserted. A do write burst signal *dowrburst* makes sure that there are at least four read words in the FIFO before data can be read into the DDR memory.

- DDR Controller

The DDR controller is an altera DDR2 SDRAM Controller megafunction which serves as an interface to the external onboard DDR memory. Read and write operations with the DDR memory are governed by the controller which in turn follows a state machine in memory read/write control module *dumpmem*. The controller here has been used as a half rate controller. For further details, reader is referred to the corresponding altera documentation.

- Memory Read/Write Control

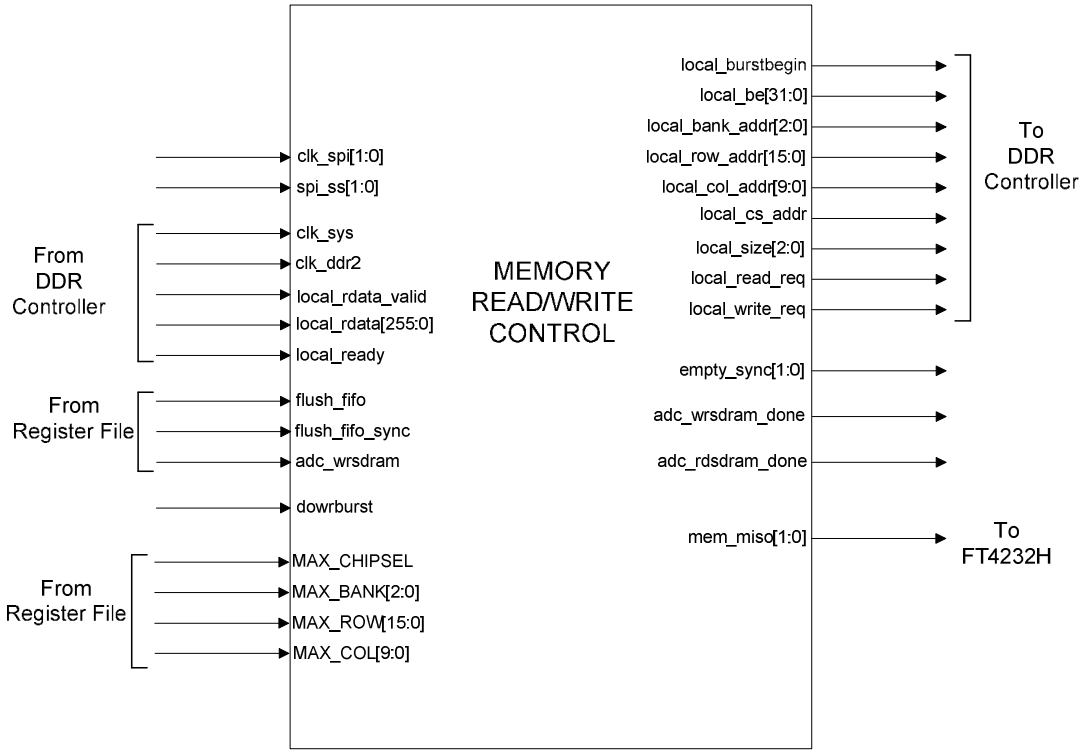

Fig. 12. Memory read/write control module *dumpmem*

Table 7
I/O description for *dumpmem* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_spi[1:0]	N/A	Input	SPI clocks from FT4232H for the two SPI interfaces in the module
spi_ss[1:0]	clk_spi[1:0]	Input	Slave select signal for the two SPI interfaces
clk_sys	N/A	Input	This is the system clock ruuning at 333.33 MHz sourced from the PLL of altera DDR controller
clk_ddr2	N/A	Input	The clock runs at 166.66 MHz, half the rate of system clock clk_sys. This clock is same as clk_sys port of memory bridge module *dumpmem_bridge* and is derived from the PLL of altera DDR controller
local_rdata_valid	clk_ddr2	Input	Data valid signal
local_rdata[255:0]	clk_ddr2	Input	Data read by the DDR controller from the memory
local_ready	clk_ddr2	Input	Indicates that the DDR controller is ready to accept data. Connected to local_ready port of DDR controller
flush_fifo	clk_sys	Input	Resets both SPI transmitters. Connected to flush_fifo[0] port of the register file
flush_fifo_sync	clk_ddr2	Input	Clears both internal FIFOs as well as resets the state machine. . It is a delayed version of flush_fifo[1] port from the register file

adc_wrsdram	clk_ddr2	Input	It is two clock cycles delayed version of capture signal from the register file
dowrburst	clk_ddr2	Input	Connected to dowrburst port of memory bridge module
MAX_CHIPSEL	N/A	Input	Maximum chip select signal. Since there is only one DDR memory, this signal is tied to 0
MAX_BANK[2:0]	clk_sys	Input	This signal is connected to mem_max_addr[18:16] ports of register file. This defines the bank address up to which the memory will be written
MAX_ROW[15:0]	clk_sys	Input	This signal is connected to mem_max_addr[15:0] ports of register file. This defines the row address up to which the last bank of the memory (MAX_BANK) will be filled
MAX_COL[9:0]	N/A	Input	This signal defines the number of columns up to which the last row (MAX_ROW) of the last bank (MAX_BANK) will be written. It is hard wired to maximum possible columns i.e. 1016
empty_sync[1:0]	clk_ddr2	Output	Empty signals for the two internal FIFOs
adc_wrsdram_done	clk_ddr2	Output	The signal is asserted for two clock cycles by the state machine after the memory has been written up to the maximum address.
adc_rdsdram_done	clk_ddr2	Output	The signal is asserted by the state machine after the memory has been read up to the maximum address.
mem_miso[1:0]	clk_spi[1:0]	Output	This port is used to transmit data samples over the two SPI interfaces. Feeds spi_miso[1:0] port of top level module tsw1400_top

Note that all the output ports with their names starting with *local* feed the corresponding ports in the altera DDR2 SDRAM controller megafunction. For the description of these ports, see corresponding altera documentation.

This module initiates all the data transfers to and from the DDR memory. It generates the read and write requests to the DDR controller which in turn reads from or writes the samples into the memory. The module is also responsible for generating read/write address.

A state machine in the module controls the generation of above signals. The flow of the state machine is as follows

1- Capture command sent by the user asserts *adc_wrsdram* flag. If then the memory is ready for the transfer, as indicated by *local_ready* input, do write burst signal *dowrburst* is checked. If it is set, the module asserts the write request flag *local_write_req* which begins the data transfer. It is to be noted that both read/write transfer is done in the burst transfer mode.

2- Sample bursts are kept being transferred till maximum programmed address of the memory is reached which implies maximum column address, maximum row address as well as maximum bank address of the memory.

3- After which *local_write_req* is deasserted and *adc_wrsdram_done* is asserted for one cycle

4- If the memory is ready for read transfer, then step 3 is followed by assertion of read request *local_read_req* signal

5- Read request is deasserted if either of the two spi FIFOs shown in Fig. 2 get filled above a maximum threshold level or maximum address of the memory is reached. *local_read_req* flag is cleared and *adc_rdsdram_done* is asserted for one cycle.

It is to be noted that the GUI only starts reading the samples once the capture bit (in the register file) is cleared by *adc_wrsdram_done* signal. During the memory write cycles, GUI keeps reading the capture register to check if all the samples have been written into the memory.

As mentioned above, the module also generates read/write addresses. The onboard DDR memory consists of 8 banks. In each bank there are 16376 rows and in each row there are 1016 columns. The module generates the addresses for each one of the three. The module writes the memory up to the programmable address location which is computed by the TSW1400 GUI from the number of samples user wants capture. It must be noted that column address is not user programmable i.e. each row is completely filled up to the maximum number of columns. Hence number of samples captured by the user must always be multiple of 4096.

The 256-bits data read from the memory is transferred into two spi FIFOs 128-bits each such that all even words are transferred into spi FIFO0 and all odd words into spi FIFO1.

SPI Transmitter

Samples from the two FIFOs are read by two SPI transmitters SPI TX0, SPI TX1 as shown in Fig. 2. Each SPI interface is configured for Clock Phase Polarity (CPHA) = 1 and Clock Polarity (CPOL) = 1 (sample on the rising edge while transmit on falling edge) as well as 8-bit data length with LSB first.

Read request to each SPI FIFO in *dumpmem* module is generated as soon as first 128-bit word arrives in the FIFO. The slave select input *spi_ss* is active low and data is transmitted by the SPI transmitter *dumpmem_mspi* module at the first negative edge of *clk_spi* after the *spi_ss* goes low. It should be noted that both transmitters work in parallel in order to double the data transfer rate.

- Register File

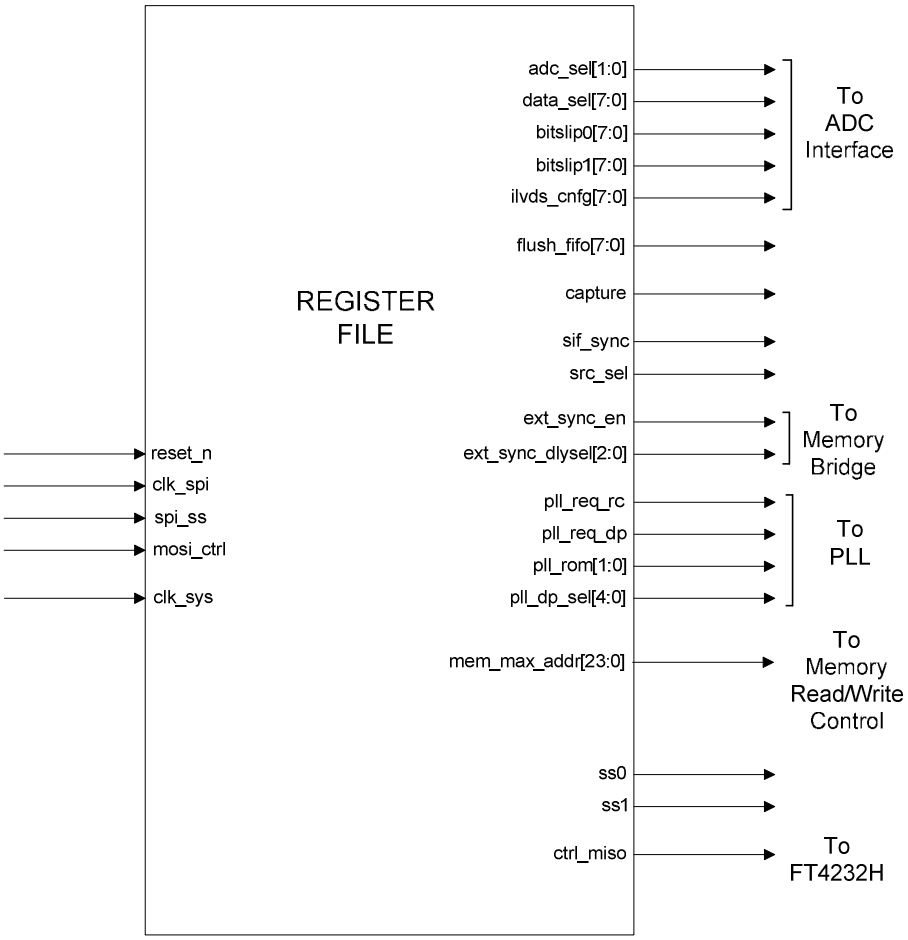

Fig. 13. Register File *dumpmem_config*

Table 8
I/O description for *dumpmem_config* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
reset_n	N/A	Input	Asynchronous reset signal. It resets the internal logic as well as reconfigure the configuration registers with their default values
clk_sys	N/A	Input	This is the system clock ruuning at 333.33 MHz sourced from the PLL of altera DDR controller
clk_spi	N/A	Input	spi clock for the SPI interface. Connected to clk_spi[2] port of the top level module tsw1400_top
spi_ss	clk_spi	Input	SPI slave select signal.
mosi_ctrl	clk_spi	Input	Carries the data to program the configuration registers. Connected to port spi_mosi[2] of the top level module tsw1400_top
adc_sel[1:0]	clk_sys	Output	It is an ADC select signal which specifies number of input LVDS ports at ADC LVDS interface from which the data will be accepted (see the

			Config1 register description)
data_sel[7:0]	clk_sys	Output	See Config7 register description
bitslip0[7:0]	clk_sys	Output	See Config4 register description
bitslip1[7:0]	clk_sys	Output	See Config5 register description
ilvds_cnfg[7:0]	clk_sys	Output	See Config6 register description
flush_fifo[7:0]	clk_sys	Output	See Config0 register description
capture	clk_sys	Output	Capture command sent by the user to start data capture from the ADC
ext_sync_en	clk_sys	Output	External trigger enable signal. See Config2 register description
ext_sync_dlysel[2:0]	clk_sys	Output	See Config2 register description
sif_sync	clk_sys	Output	Software trigger generated by the user
srcsel_sync	clk_sys	Output	Trigger source select signal. See Config2 register description
pll_req_rc	clk_sys	Output	PLL reconfiguration request.
pll_req_dp	clk_sys	Output	PLL phase reconfiguration request
pll_rom[1:0]	clk_sys	Output	Selects the required PLL ROM from which reconfiguration data is to be read
pll_dp_sel[4:0]	clk_sys	Output	pll_dp_sel[0] : Selects the direction of phase shift (increment or decrement). pll_dp_sel[4:1] : Select the PLL counter for which the phase tap settings need to be reconfigured. See Config3 register description
mem_max_addr[23:0]	clk_sys	Output	Defines the address up to which DDR memory is filled
ss0 ss1	-	-	Unused
ctrl_miso	clk_spi	Output	This port is used by the user interface to read the configuration registers over SPI. Connected to spi_miso[2] port of top level module tsw1400_top

The register file is used to program various configuration registers allowing the user to set various user selectable parameters in the firmware. These are programmed by the user TSW1400 GUI through SPI interface. The SPI interface in this case is implemented using the Bit-Banging technique and has the same configuration as that of the other two SPI interfaces used to transmit ADC samples.

Detail of various configuration registers is provided below.

Config 0 :

Write Address	Read Address	D7	D6	D5	D4	D3	D2-D0
0x10	0x00						flush_fifo

D2-D0 : flush_fifo

3'b111 : Clears all internal FIFOs

Config 1 :

Write Address	Read Address	D7	D6	D5-D4	D3	D2	D1	D0
0x11	0x01			adc_sel				

D5-D4 : adc_sel

2’b00, 2’b01 : Transmits samples from ADC connected to lvds_port0 only
2’b10 : Transmits samples from ADC connected to lvds_port1 only
2’b11 : Transmits samples from both ADCs[5]

Config 2 :

Write Address	Read Address	D7	D6-D4	D3	D2	D1	D0
0x12	0x02	ext_sync_en	delay_sel		sif_sync	srcsel_sync	capture

D7 : ext_sync_en

1’b1 : Enables external trigger. If enabled, data capture will not start until externally triggered (TP13 on TSW1400). Works on rising edge

D6-D4 : delay_sel
This signal is used to delay the external trigger by specified number of cycles of the clk_adc clock in the memory bridge module

D2 : sif_sync

1’b1 : Acts as a software trigger. This bit is asserted by the user whenever trigger is to be applied through the user’s command

D1 : srcsel_sync

1’b1 : Routes the external trigger ext_sync, input port of the top level module, to the trig output port of the top level.

1’b0 : Routes the software trigger sif_sync signal to the trig output port of the top level. Used to synchronize multiple TSW1400 boards for ADC data capture

D0 : capture

1’b1 : Generates data capture command. If external trigger is enabled, trigger must be applied before data capture command can be sent

Config 3 :

PLL Reconfiguration

Write Address	Read Address	D7	D6	D5-D4	D3	D2	D1	D0
0x13	0x03			pll_rom				pll_reconfig

D0 : pll_reconfig
1’b1 : Enables real time reconfiguration of the PLL counters’ clock frequencies

D4-D5 : pll_rom

[5] Must be set 2’b11 for dual bus ADC.

PLL ROM selection corresponding to the desired frequency settings

2'b00: 300M < inclk0[6] < 625M.
2'b01: 150M < inclk0 < 300M.
2'b10: 80M < inclk0 < 150M.
2'b11: 40M < inclk0 < 80M

PLL Phase Shift Settings

Write Address	Read Address	D7-D3	D2	D1	D0
0x13	0x03	pll_counter_sel		pll_phase	

D1 : pll_phase
1'b1 : Enables real time reconfiguration of the PLL's output clocks' phase shifts

D7-D3 : pll_counter_sel
D3 : Selects direction of phase shift (increment 1 or decrement 0)
D7-D4 : Selects PLL post scale counter of the output clock to be phase shifted[7]

Note that the bits D0 and D1 must not be asserted simultaneously.

Config 4 :

Write Address	Read Address	D7	D6	D5	D4	D3	D2	D1	D0
0x14	0x04								bit_slip

D0 : bit_slip
1'b1 : Generates bitslip request for LVDS_RX0[8]

Config 5 :

Write Address	Read Address	D7	D6	D5	D4	D3	D2	D1	D0
0x15	0x05								bit_slip

D0 : bit_slip
1'b1 : Generates bitslip request for LVDS_RX1

Config 6 :

Write Address	Read Address	D7	D6	D5	D4	D3	D2	D1	D0
0x16	0x06				lvds1_reset				lvds0_reset

D0 : lvds0_reset
1'b1 : Resets data alignment circuitry of LVDS_RX0[9]

D1 : lvds1_reset
1'b1 : Resets data alignment circuitry of LVDS_RX1

Config 7 :

---

[6] Note that *inclk0* is the DDR clock coming form ADC (Bit clock)

[7] Clocks have been drawn from post scale counters 0, 3 and 5. See altera documentation "Clock Networks and PLLs in Stratix IV devices" for value corresponding to each of these counters

[8] Whenever a bit slip request is generated, it applies to all 16 inputs of LVDS receiver. Also for multiple bitslips, bitslip request must be generated as many times

[9] Applies to all 16 input channels of LVDS receiver

Write Address	Read Address	D7	D6	D5-D3	D2	D0-D1
0x17	0x07			data_sel[5:3]		data_sel[1:0]

D0-D1 : Used for correct formatting of data. Read format data module description

D5-D4 : Serves the same purpose as D0-D1, except that they are used to format the data from the other 16-bit input bus (for dual bus ADCs)

D3 : Asserted during search for correct frame clock sequence

Config8-Config10 :

These registers are used to program maximum row and bank addresses up to which DDR memory is written or read. The *mem_max_addr* output holds all the maximum addresses

mem_max_addr : {config10, config9, config8}

mem_max_addr [15:0] : Maximum row address
mem_max_addr [18:16] : Maximum bank address

The maximum row address can be as high as 3FFF while that for the bank is 7. It is must be noted that maximum row address is the number of rows up to which the last bank of the maximum bank address will be filled. For all the previous banks, all rows will be filled. In the firmware, this is taken care of by a *prelast_addr_hit* signal in the memory read/write control module. Also note that since the number of samples to be captured must be multiple of 4096, the number of columns to be written is not programmable as each row is always filled up to its maximum number of columns of 3F8.

Config11 :

This register contains the Interface ID for a device family. All the devices which share the same firmware are assigned one unique Interface ID. This register is read only and is read by the TSW1400 GUI to figure out which type of firmware is loaded in the FPGA

Config14 :

This is a read only register and carries the latest firmware version number

Appendix A :

Phase Shifts Settings for PLL Output Clocks

For the LVDS receiver to sample the ADC data correctly, clocks going to *altlvds_rx* megafunction, as shown in Fig. 4(a), must be given certain amount of phase shift. The phase shift given depends upon deserialization factor and whether the data from the ADC is centrally aligned or edge aligned, as mentioned in the altera design example on "Using altlvds With External PLL Option".

For a deserialization factor of 4, the phase shifts to the three output clocks of the PLL are given in Table 9

Table 9
Phase Shifts of PLL output clocks

Clocks	Edge Aligned	Centrally Aligned
C0 (clk_adc)	-45^o	0
C3 (clk_ser)	-180^o	0
C5 (clk_en)	180^o	225^o

To realize phase shift, the PLL uses VCO delay cycles in order to delay the output clocks. For a finer resolution, phase taps are used where 1 phase tap is $1/8^{th}$ of a VCO cycle. These delay settings are different for different PLL lock ranges as shown in Table 10

Table 10
Delay Settings for the PLL Counters

Counter	300M-625M		150M-325M		80M-160M		40M-80M	
	Edge	Center	Edge	Center	Edge	Center	Edge	Center
M	1(4)	1	2	1	3	1	3	1
C0	1	1	1	1	1	1	1	1
C3	1	1	1	1	1	1	1	1
C5	3(4)	3(4)	6	6	11	11	11	11
K	1	1	1	1	1	1	2	2

The quantity in parentheses is the number of phase taps and the one outside is the number of VCO cycles by which the corresponding clock is delayed. M counter shifts all of the output counters (C0-C6), K counter is used to divide VCO clock by 2 (if K=2) in order to generate slower clocks form the PLL[10].

As mentioned in the PLL reconfiguration section, every time after reconfiguring the PLL lock range, the settings for the initial VCO delay cycles and phase taps revert to what was mentioned in the *altpll* megafunction. Therefore in order to program the correct phase shifts, the output counters' delay settings have to be reconfigured according to Table 10. Note that the settings in Table 10 have been found using the *altpll* megafunction.

[10] Unlike other counters, K counter can not be dynamically programmed. Therefore K=2 value is never used, however, K=1 seems to work fine even for 40M-80M range.