Texas Instruments

TSW 1400

DAC 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 TSW1400 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 over an LVDS or a CMOS interface. A block diagram of the system is shown in Fig. 1.

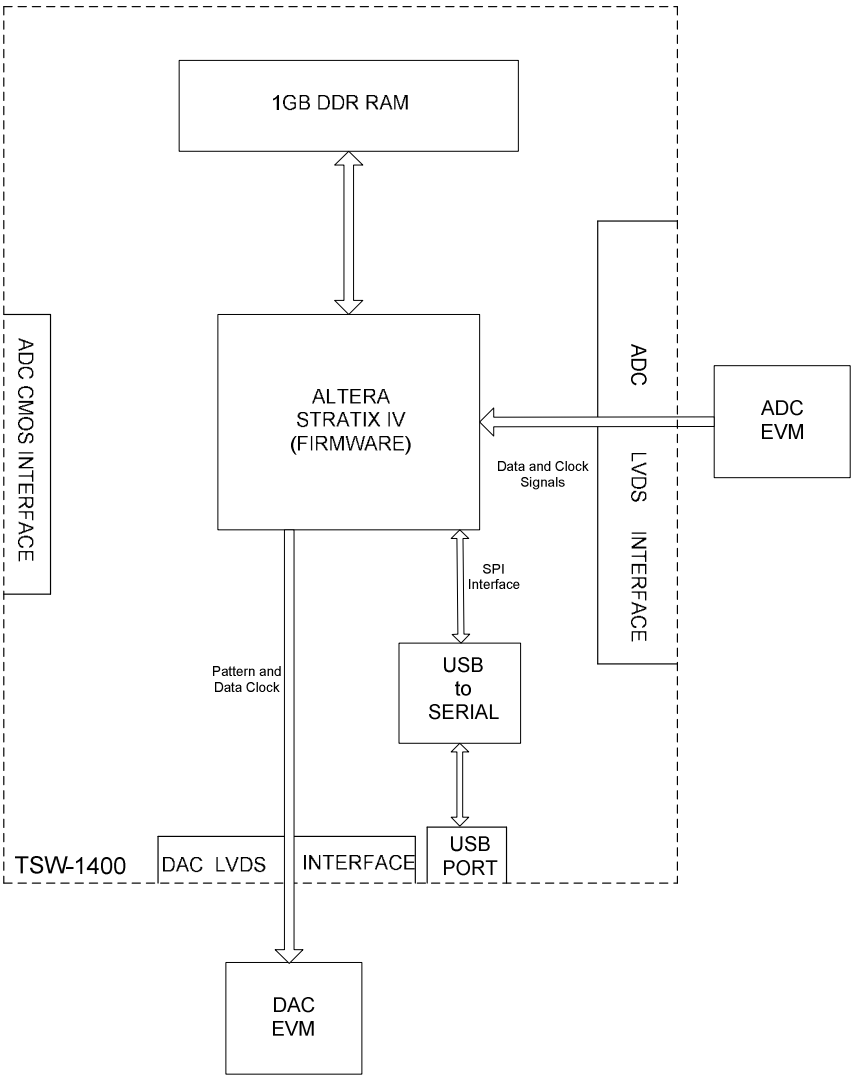

Fig. 1. Block diagram of TSW 1400

- Digital to Analog Converter (DAC)

As mentioned before, TSW1400 comes with its own GUI which sends the desired pattern over a USB interface shown in Fig. 1. 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 USB and transmits it to the SPI interface. The FPGA firmware shown in Fig. 1 stores the data received into an on board 1GB DDR memory. The data from the memory is then read by the firmware and transmitted to the DAC interface which feeds the DAC EVM.

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

- DAC Firmware

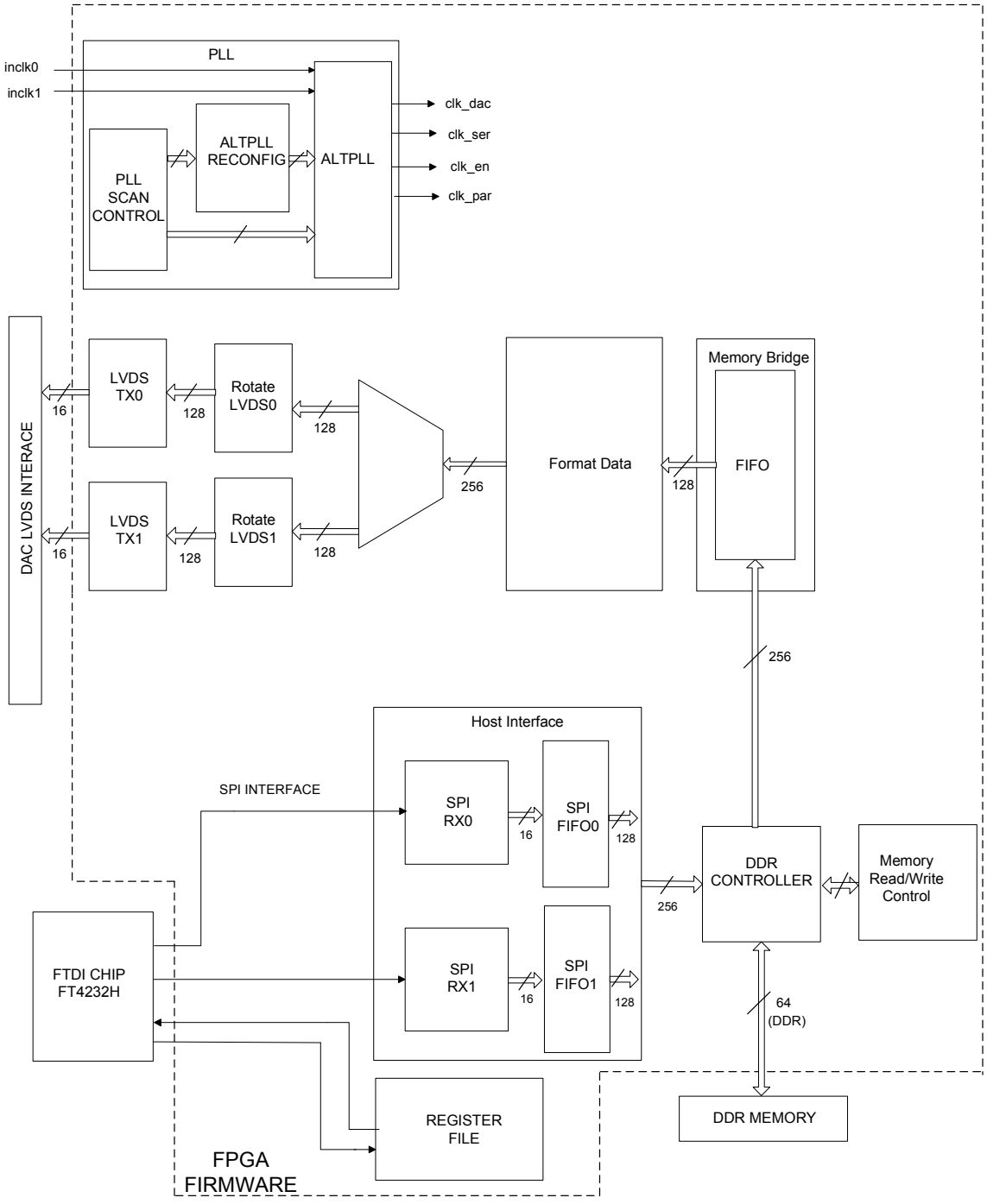

Fig. 2. DAC Firmware block diagram

DAC firmware in the FPGA is the Verilog code that performs all the necessary tasks to receive pattern generated by the GUI and transmit them to the DAC. Fig. 2 shows various module of firmware and the flow of data through them. A detailed description of these modules is provided below.

- Top Level Module

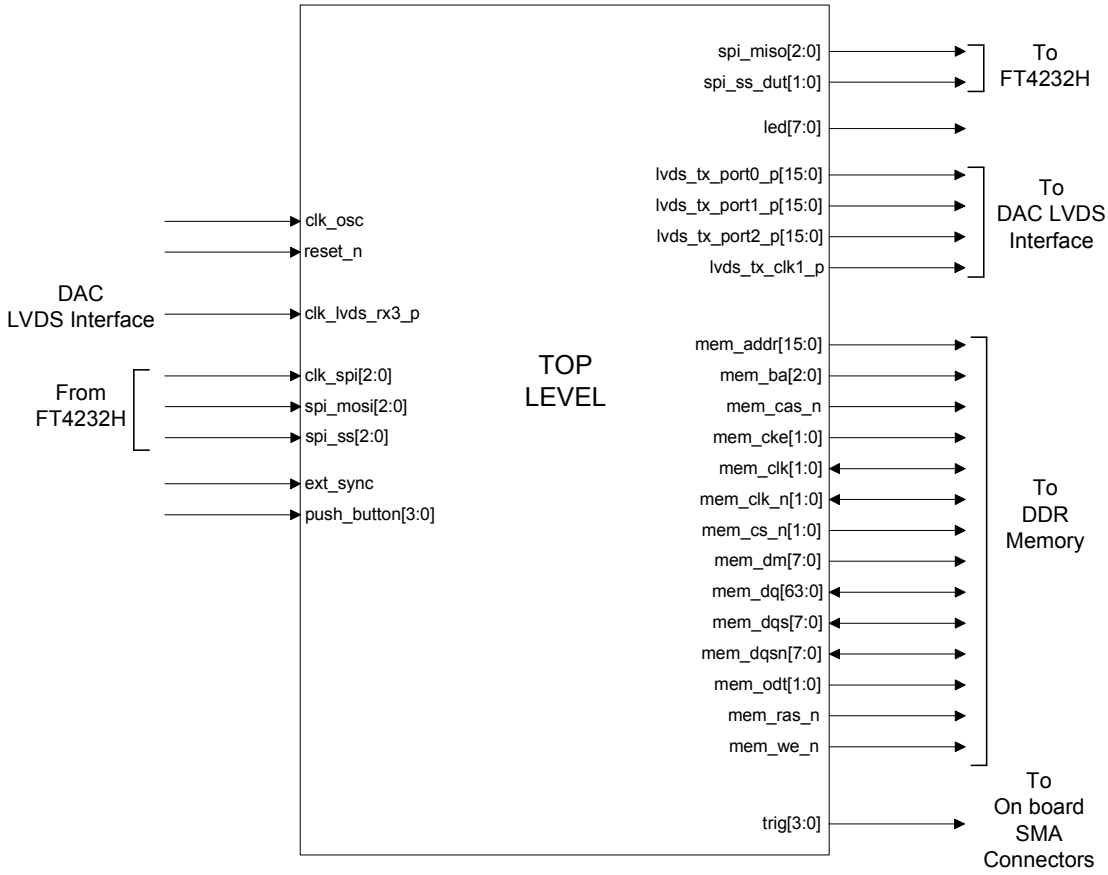

Fig. 3. Top level module *tsw1400_top*

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
clk_lvds_rx3_p	N/A	Input	Clock provided by the DAC EVM
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 the two SPI interfaces which receive pattern or samples sent by the user.

			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 receive the pattern from 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	It serves as an external on board trigger for the firmware to start sending pattern to the DAC
push_button[3:0]	N/A	Input	These are connected to the onboard push buttons. If pressed, push_button[0] is used to disable the sync signal going to the DAC. Connected to sync_en port of format data module If pressed, push_button[1] is used to disable the frame signal going to the DAC. Connected to frame_en port of format data module push_button[3:2] are unused
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 unused 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 reception of pattern by the firmware over SPI interface led[2] : System reset led. led[4:3] : PLL lock indications from the two PLL driving the two LVDS transmitters shown in Fig. 2 led[5] : This LED turns up if bridge FIFO gets empty at any point during data transfer to the DAC (error indication) led[6:7] : Indicates if there is any unread sample left in the two SPI FIFOs (error indication)
lvds_tx_clk1_p	N/A	Output	This serves as the data clock for the DAC
lvds_tx_port0_p[15:0]	lvds_tx_clk1_p	Output	This is one of the two buses transmitting the pattern to the DAC

lvds_tx_port1_p[15:0]	lvds_tx_clk1_p	Output	This is the second of the two buses transmitting the pattern to the DAC
lvds_tx_port2_p[15:0]	N/A	Output	lvds_tx_port2_p[10:0]: Unused lvds_tx_port2_p[11]: This bit makes the sync signal of the DAC lvds_tx_port2_p[14:12]: Unused lvds_tx_port2_p[15]: This bit makes the frame signal of the DAC

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.

- Host Interface

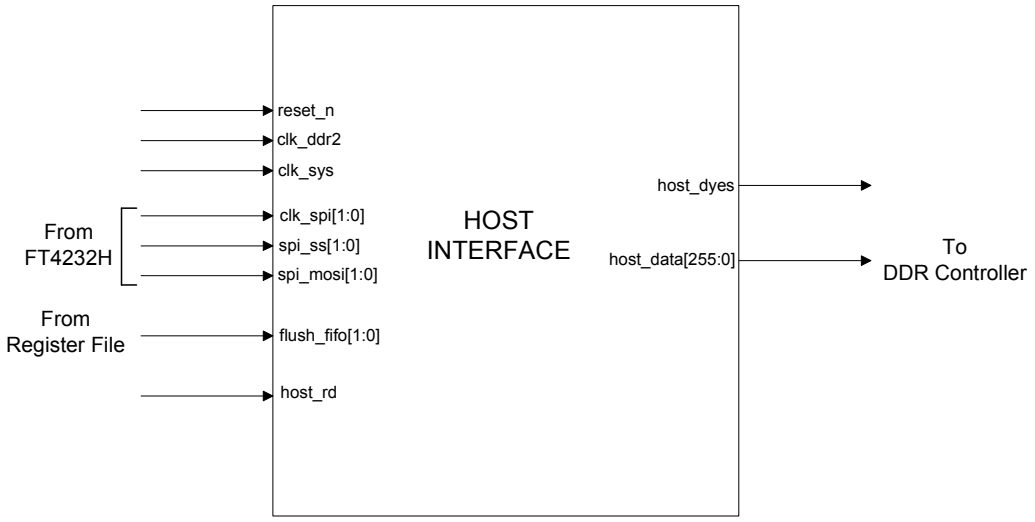

Fig. 4. Host Interface module *hostif*

Table 2
I/O description for *hostif* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
reset_n	N/A	Input	This port is connected to local_init_done port of the DDR controller. The controller asserts this signal once it has completed memory initialization. In this module it serves as asynchronous reset signal
clk_sys	N/A	Input	This is the system clock running at 400 MHz. The clock is sourced from altpll megafunction
clk_ddr2	N/A	Input	The clock runs at 200 MHz, half the rate of system clock clk_sys. It is sourced from the PLL of altera DDR controller

clk_spi[1:0]	N/A	Input	SPI clocks for the two SPI interfaces
spi_ss[1:0]	clk_spi[1:0]	Input	Slave select signal for the two SPI interfaces used to receive the pattern from user interface
spi_mosi[1:0]	clk_spi[1:0]	Input	These are the two SPI interfaces which receive the pattern from user interface
flush_fifo[1:0]	clk_ddr2	Input	Clears the two SPI FIFOs.
host_rd	clk_ddr2	Input	It serves as read request for the two SPI FIFOs. This signal is obtained by ANDing the local_ready signal from DDR controller and local_write_req signal from memory read/write control module
host_dyes	clk_ddr2	Output	This signal is asserted when there are atleast four read words in each of the two SPI FIFOs. It is connected to dowrburst port of memory read/write control module
host_data[255:0]	clk_ddr2	Output	Data read from the two SPI FIFOs.

The pattern or sample data transmitted by user program over SPI is received by the SPI receiver *dumpmem_mspi* module in the host interface. In order to double the data transfer rate, samples are transmitted over two SPI interfaces as shown in Fig. 2. Each SPI receiver is a SPI slave configured for Clock Phase Polarity (CPHA) = 1 and Clock Polarity (CPOL) = 1 (sample on rising edge while transmit on falling edge) as well as 8-bit data length with LSB first.

The receiver gives a 16-bit sample at the output which is stored in SPI FIFO. Output of host interface is 256-bit which is obtained by placing all the 128-bit words from SPI FIFO0 at even word positions and those from SPI FIFO1 at odd word positions. See appendix B to find out the required sequence in which samples must be transmitted for correct operation of the firmware

- DDR Controller

The DDR controller is an altera DDR2 SDRAM Controller megafunction which serves as an interface to the external onboard DDR memory used to store the samples before they are transmitted to the DAC. 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 Bridge

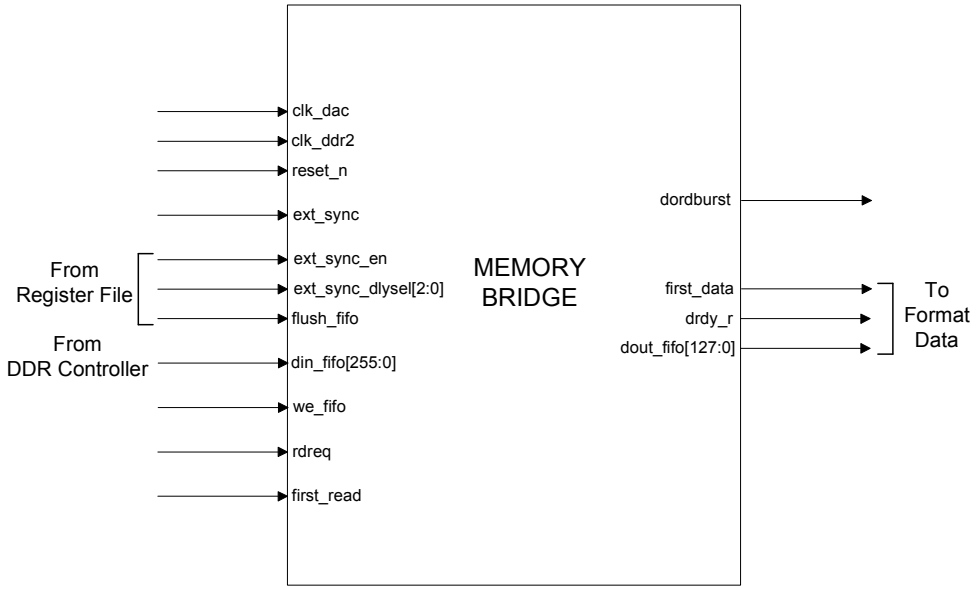

Fig. 5. Memory Bridge module *dumpmem_bridge*

Table 3
I/O description for *dumpmem_bridge* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_dac	N/A	Input	This clock is twice the frequency of the fpga clock from the DAC EVM. The clock is generated by alter PLL megafunction altpll
clk_ddr2	N/A	Input	The clock runs at 200 MHz, half the rate of system clock clk_sys. It is sourced from the PLL of altera DDR controller
reset_n	N/A	Input	Asynchronous reset signal used to reset internal logic
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
flush_fifo	clk_dac	Input	Clears the internal FIFOs
din_fifo[255:0]	clk_ddr2	Input	Data read by the DDR controller from the memory
we_fifo	clk_ddr2	Input	It serves as write request for the internal FIFO. This signal is obtained by ANDing data valid signal loca-rdata_valid from DDR controller and play signal from the register file
rdreq	clk_dac	Input	Connected to rdreq port of format data module
first_read	clk_ddr2	Input	Connected to first_read port of memory read/write control module
dordburst	clk_ddr2	Output	This signal goes to memory read/write control module and serves as read request for the DDR memory. The signal is deasserted if the bridge

			FIFO gets half filled
first_data	clk_dac	Output	This signal is asserted for the first two cycles at the start of a pattern
drdy_r	clk_dac	Output	This is data ready signal asserted when the FIFO gets half filled. Remains high afterwards till reset or if flush_fifo is asserted
dout_fifo[127:0]	clk_dac	Output	Data read from the FIFO

This module contains a FIFO in which the DDR controller writes samples after reading them from DDR memory. Write request to the FIFO is generated with the assertion of *we_fifo* input. This signal is obtained by ANDing data valid signal *loca-rdata_valid* from DDR controller and *play* signal from register file. The data valid signal in turn is asserted in response to DDR memory read request generated by the memory read/write control module to the DDR controller. The *play* flag is asserted by the user through register file to start sending pattern stored in the DDR memory to the DAC.

The output *first_data* is same as *first_read* input and indicates start of a pattern. Note that since output data width is half that of the input, this signal is high for the first two cycles every time a pattern starts.

- Memory Read/Write Control Module

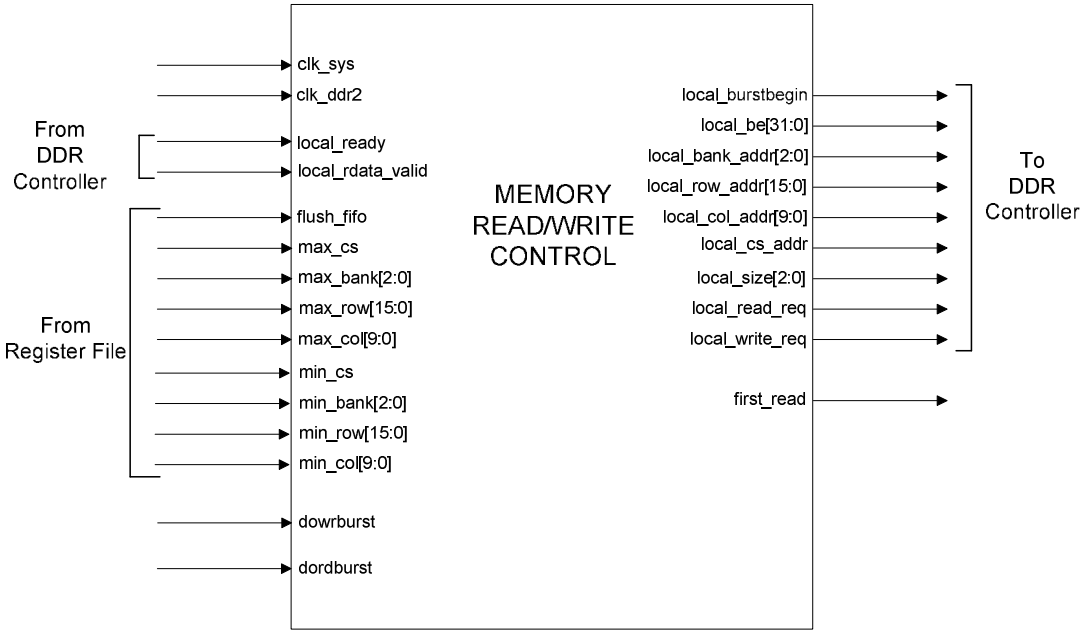

Fig. 6. Memory read/write control module *dumpmem*

Table 4
I/O description for *dumpmem* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_sys	N/A	Input	This is the system clock running at 333.3333 MHz. The clock is sourced from altpll megafunction
clk_ddr2	N/A	Input	The clock runs at 166.66 MHz, half the rate of system clock clk_sys. It is sourced from the PLL of altera DDR controller

local_rdata_valid	clk_ddr2	Input	Valid signal for the data read out of the memory. Connected to local_rdata_valid port of DDR controller
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_ddr2	Input	Resets internal state machine.
max_cs	clk_sys	Input	Maximum chip select address corresponding to pattern length. Connected to mem_max_addr [29] port of register file.
max_bank[2:0]	clk_sys	Input	Maximum memory bank address address corresponding to pattern length. Generated from mem_max_addr [26:28] from register file
max_row[15:0]	clk_sys	Input	Maximum memory row address address corresponding to pattern length. Generated from mem_max_addr [25:10] from register file
max_col[9:0]	clk_sys	Input	Maximum memory column address address corresponding to pattern length. Generated from mem_max_addr [9:0] from register file
min_cs	clk_sys	Input	Minimum chip select corresponding to preamble length. Connected to mem_min_addr [29] port of register file.
min_bank[2:0]	clk_sys	Input	Minimum memory bank address corresponding to preamble length. Generated from mem_min_addr [26:28] from register file
min_row[15:0]	clk_sys	Input	Minimum memory row address corresponding to preamble length. Generated from mem_min_addr [25:10] from register file
min_col[9:0]	clk_sys	Input	Minimum memory column address corresponding to preamble length. Generated from mem_max_addr [9:0] from register file
dowrburst	clk_ddr2	Input	Connected to host_dyes port of host interface module
dordburst	clk_ddr2	Input	Connected to dordbust port of memory bridge module
first_read	clk_ddr2	Output	One shot signal asserted with the first word of a pattern read out of the memory

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 pattern 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- When the two SPI FIFOs in the host interface module are filled above a minimum level, *dowrburst* input is asserted. If then the memory is ready for data transfer, as indicated by *local_ready* input, the module asserts write request *local_write_req* as well as burst begin signal *local_burstbegin* which begins the data transfer to the DDR memory. It is to be noted that both read/write transfer is done in the burst transfer mode with burst length set to 2.

2- Sample bursts are kept being transferred till maximum address of the memory is reached which implies maximum column address, maximum row address as well as maximum bank address of the memory. Note that each one of the maximum address is programmable through register file.

3- After which *local_write_req* is deasserted.

4- If memory is ready for read transfer and *dordburst* input is high, read request is generated by asserting *local_read_req*.

5- Reading is restarted from the starting address every time after one complete pattern has been read[1]. The state machine remains in the read state until a new pattern is sent by the user which is indicated by the assertion of *dowrburst* input.

6- The state machine completes reading current burst and then begins writing the new pattern to the memory starting from the first memory address. This follows all the steps from 1 to 5.

It is to be noted that the data read out of the memory is available at the output of DDR controller. The input signal *loca_rdata_valid* indicates when a valid data is available.

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 to send to the DAC.

- DAC Interface

DAC interface module *dacif* serves as a top level for format data and LVDS TX modules shown in Fig. 2. It receives data from the memory bridge module and transmits it on to two 16-bit LVDS channels. Detail of each of the sub modules is given below.

Format Data

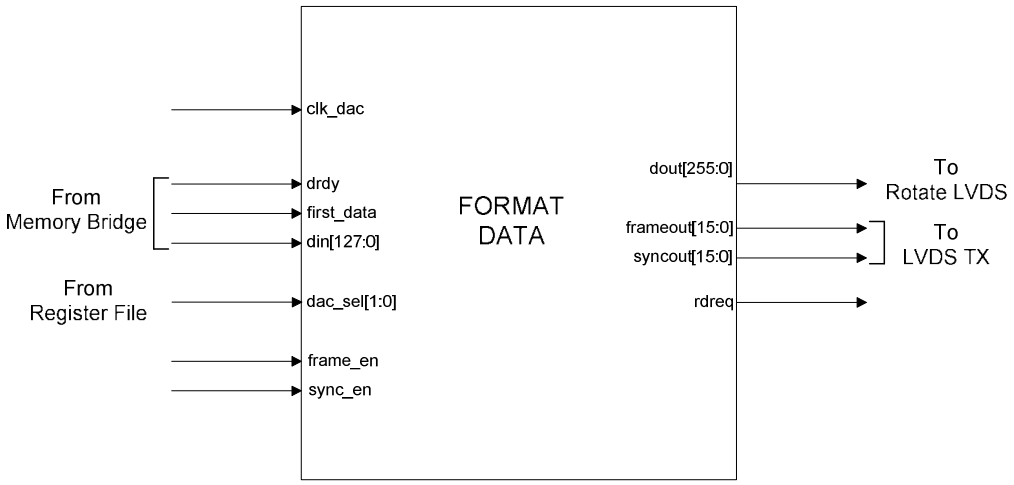

Fig. 7. Format Data module *dacif_formatpdata*

[1] If preamble is also loaded in the memory with the data samples as indicated by non zero minimum memory address, the preamble is sent only once to the DAC, rest of the pattern is repeated in a continuous loop

Table 5
I/O description for *adcif_formatpdata* module

SIGNAL	CLOCK DOMAIN	DIRECTION	DESCRIPTION
clk_dac	N/A	Input	This clock is twice the frequency of the fpga clock from the DAC EVM. The clock is generated by alter PLL megafunction altpll
drdy	clk_dac	Input	Connected to drdy_r port of memory bridge module
first_data	clk_dac	Input	Connected to first_data port of memory bridge module
din[127:0]	clk_dac	Input	Data read out of memory bridge FIFO
dac_sel[1:0]	clk_sys	Input	Connected to dac_sel port of register file
frame_en	N/A	Input	Connected to push_button[1] port in the top level module tsw1400_top
sync_en	N/A	Input	Connected to push_button[0] port in the top level module tsw1400_top
dout[255:0]	clk_dac	Output	Data output
frame_out[15:0]	clk_dac	Output	This is the frame signal required by the DAC. The 16 bits of this signal are for 16 samples contained in the output bus.
sync_out[15:0]	clk_dac	Output	This is the sync signal required by the DAC. The 16 bits of this signal are for 16 samples contained in the output bus.
rdreq	clk_dac	Output	This is the read request for memory bridge FIFO

This module reads 128-bit data from memory bridge module and deserializes it into 256-bit output. For dual bus DACs such as TI's DAC34H84, read request *rdreq* is generated in every cycle once data ready input *drdy* is asserted. In this case, the lower 128 bits contain samples for channels A and B while the upper 128 bits carry samples for channels C and D on alternate word boundary. If a single bus DAC is present for instance TI's DAC3484., read request is generated on alternate cycle after *drdy* goes high. In this case the upper and lower 128 bits in the output carry same data. A programmable *dac_sel* input specifies whether the DAC plugged in is a dual bus DAC or single bus. It must be noted from the preceding discussion that the 256-bit output is valid on alternate cycles of *clk_dac* clock.

The module also generates *frameout* and *syncout* signals required by the DAC. The *syncout* signal is asserted at the start of every pattern as determined by *firstdata* input while the *frameout* signal is asserted after every 64 samples on a single bus.

Rotate LVDS

Upper and lower data 128 bits from the preceding module are fed into one of the two rotate LVDS modules as shown in Fig. 2. This module simply rewires the input data such that when the data is serialized by the LVDS TX module, the 16-bit serialized data is in the conventional bit order (most significant bit in the most significant position and least significant bit in least significant position). The module rearranges the data bits in the same order as they come out of a deserializer. Refer to the format data module description in the TSW1400 ADC firmware section to see how bits are placed in the output of a deserializer.

LVDS TX

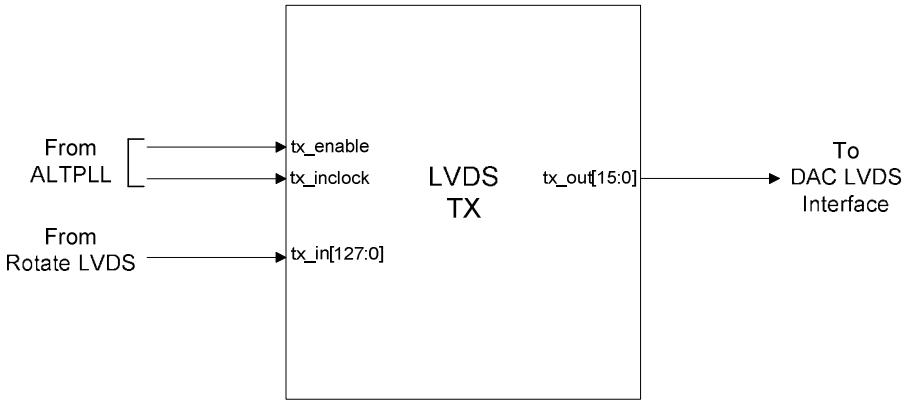

Fig. 8. LVDS Transmitter *dacif_lvds* [2]

For LVDS transmitter module, firmware uses *altlvds TX* which is an altera megafunction. It serializes the 128-bit input data from rotate LVDS module into 16-bit output as shown in Fig. 2. The output of this module feeds DAC directly through the DAC LVDS interface on TSW1400. For dual bus DACs such as TI's DAC34H84, two such transmitter modules are used. Although not shown in Fig. 2, upper 8 bits of *frameout* and *syncout* signals from format data module are also serialized into single bit for input to the DAC. Note that only 8-bits of these signals are used as even for the dual bus DACs, only a single frame and sync input is required.

For clocking scheme of LVDS TX, see phase locked loop description.

- Phase Locked Loop

PLL used is the altera megafunction *altpll* which provides the required clocking. As shown in Fig. 2, *inclk0* is the clock provided by DAC EVM. This serves as the reference clock for the PLL. The frequency of this clock (called FPGA clock) must satisfy (1)

$$FPGA\_clk = (FDATA*ndac*B) / (BUS*8) \quad (1)$$

where FDATA is the data rate (sampling frequency/interpolation), ndac is the number of channels of the device, B is the number of bits in a DAC sample which is always 16-bits and BUS is the width of the output interface, which is 16 for all the currently supported DACs except for DAC34H84 for which it is 32. The factor of 8 in the denominator is used to slow down the incoming clock for the reason which will make apparent shortly.

The second clock input *inclk1* in Fig. 2 is tied internally to zero and has been used to extend PLL input frequency lock range. At the output, *clk_dac* is used by the internal logic and is four times as fast as the reference clock. The two clocks *clk_ser* and *clk_en* serve as fast and slow clocks of the serializer (LVDS TX module) respectively while *clk_par* clocks the synchronizing register which is used to register the data before it is fed to the serializer. These clocks are shown explicitly in Fig. 9.

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

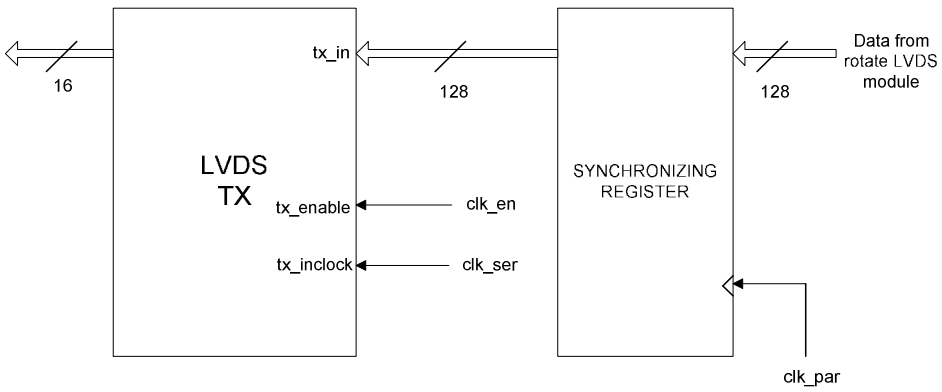

Fig. 9. Clocking scheme for LVDS TX module

Note that this type of clocking scheme has been used due to the fact that the *altlvds* megafunction which is the LVDS transmitter has been used with external PLL option. Moreover the *altpll* megafunction is configured for source synchronous compensation mode. For further information, reader is referred to altera "Clock Networks and PLLs in Stratix IV devices" documentation as well as altera support example on "Using altlvds With External PLL Option".

For the case of dual bus DAC34H84, the two LVDS TX modules are clocked by two different PLL having the same configuration and phase settings. For such a case, DAC348H4 EVM provides two synchronized FPGA clocks to serve as reference clock for the two PLLs. Note that in order to ensure that the output clocks of the two PLLs are synchronized, all the output clocks are generated by multiplying the input clock with a desired factor[3]. It is due to this reason that the input clock is slowed down by using a factor of 8 as given by (1).

The data clock for the DAC is generated by another LVDS transmitter module (not shown in Fig. 2) using 8:1 serialization. This transmitter is clocked by the same PLL as the one which clocks LVDS transmitter of channel A/B in the firmware. Note that the data clock is generated with the serializer so that at the output, the clock is synchronized (edge aligned) with the data

.

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* megafucntion as well as the modules required to reconfigure the PLL.

[3] Synchronization can never be guaranteed among divided down clocks

PLL Reconfiguration

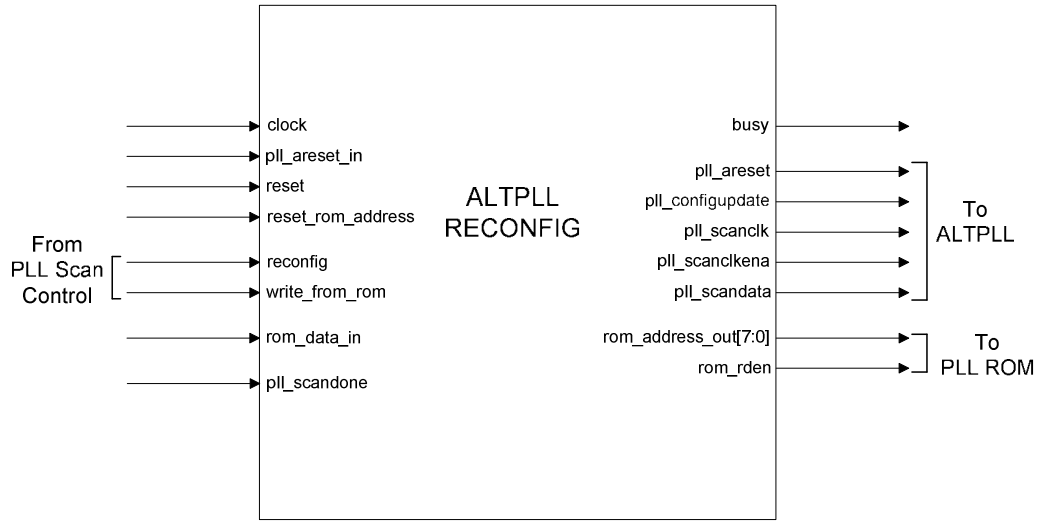

Fig.10. PLL reconfiguration module *ipll_reconfig* [4]

PLL can also be reconfigured in real time using altera *altpll_reconfig* megafunction in order to change the PLL frequency lock range to support various DACs each having different sampling rate. Reconfiguration block is driven by a state machine implemented in *ipll_scanctrl* module.

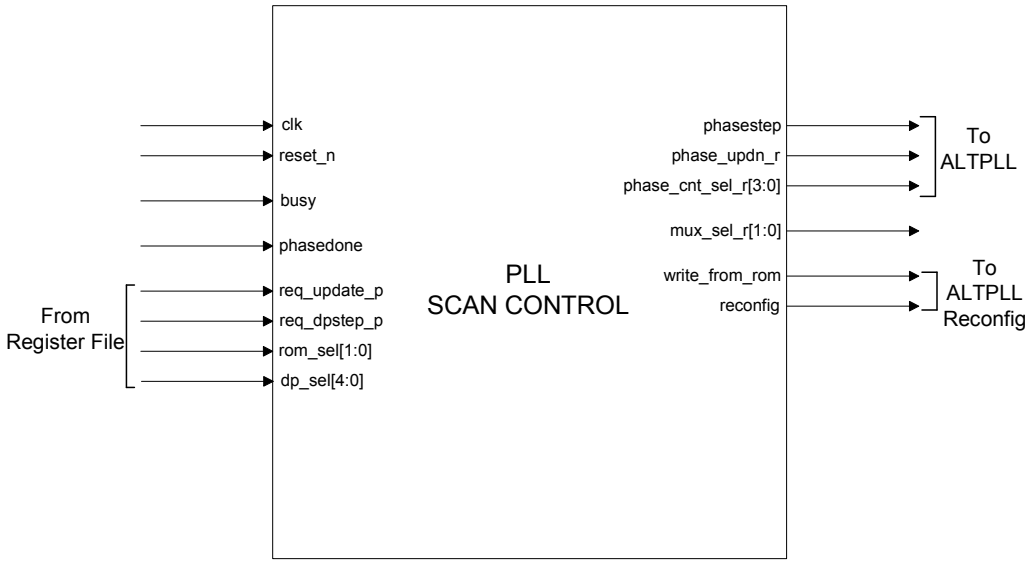

Fig. 11. PLL scan control module *ipll_scanctrl*

Table 6
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

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

			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 (see appendix A). *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

- Register File

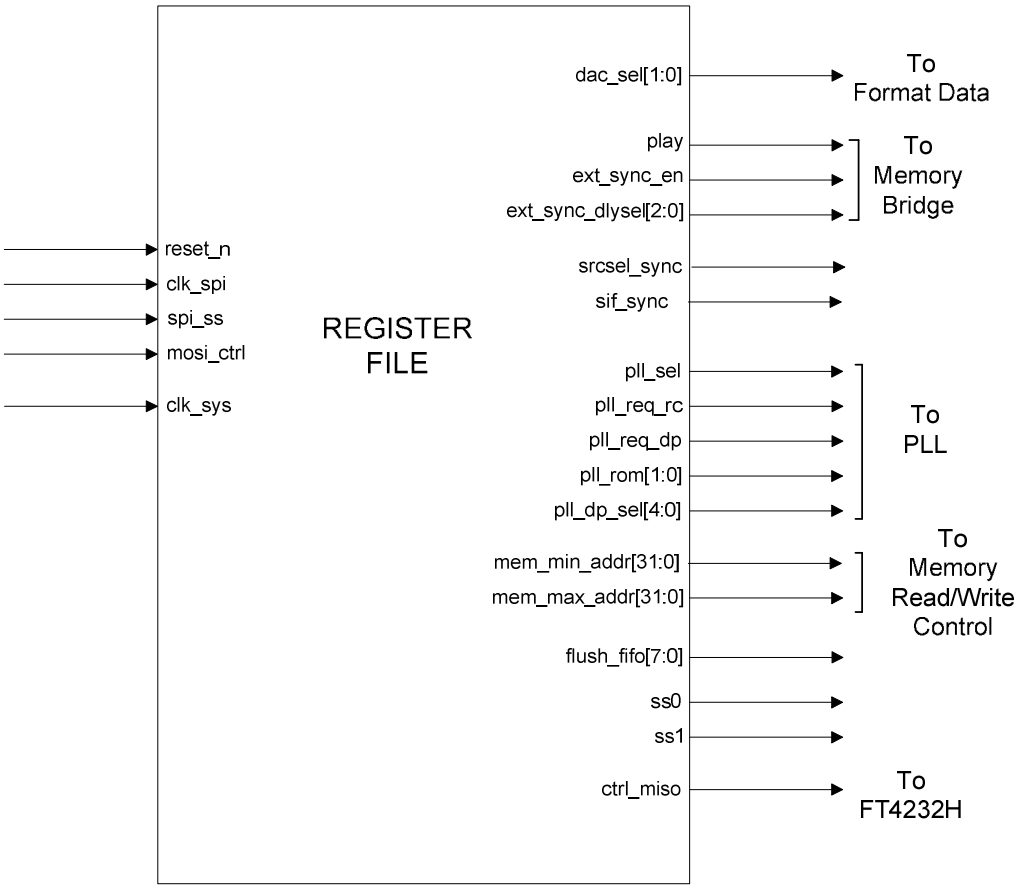

Fig. 12. Register File *dumpmem_config*

Table 7
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 running at 400 MHz. The clock is sourced from altpll megafunction
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
dac_sel[1:0]	clk_sys	Output	It is the DAC select signal which specifies whether the DAC plugged in is a single bus DAC or dual bus
flush_fifo[7:0]	clk_sys	Output	See Config0 register description
play	clk_sys	Output	Host interface module uses play as a trigger to

			start receiving pattern from the user interface
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_sel	clk_sys	Output	This is the PLL select signal which specifies which of the two PLL driving the two LVDS TX (shown in Fig.1) are to be reconfigured
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[31:0]	clk_sys	Output	This specifies the maximum address up to which memory can be written. See Config4-Config7 registers description
mem_min_addr[31:0]	clk_sys	Output	Specifies DAC preamble length. See Config8-Config9 registers description
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 interface in MATLAB through SPI. 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 receive pattern from the user interface.

Detail of various configuration registers is provided below.

Config 0 :

Write Address	Read Address	D7	D6	D5	D4	D3	D2-D1	D0
0x10	0x00					flush_fifo[3]	flush_fifo[2:1]	flush_fifo[0]

D0 : flush_fifo[0]
1'b1 : Resets state machine in memory read/write control module

D2-D1 : flush_fifo[2:1]
2'b11 : Clears the two SPI FIFOs in host interface module

D3 : flush_fifo[3]
1'b1 : Clears the FIFO in memory bridge module

Config 1 :

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

D5-D4 : dac_sel

It provides the information whether the DAC plugged in is single bus or dual bus

2'b11 : Dual bus DAC

For any other value of dac_sel, DAC is assumed to be single bus

Config 2 :

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

D7 : ext_sync_en

1'b1 : Enables external trigger. If enabled, pattern is not be sent to the DAC (by disabling writing to the bridge FIFO) until externally triggered . Works on rising edge

D6-D4 : delay_sel

This signal is used to delay the external trigger by the specified number of cycles of clk_dac 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 output of multiple DACs connected to different TSW1400 boards

D0 : play

1'b1 : Enables writing to the bridge FIFO to start sending the pattern to the firmware

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
PLL ROM selection corresponding to the desired frequency settings
2’b00: 75M < inclk0[5] < 200M.
2’b01: 40M < inclk0 < 80M.
2’b10: 20M < inclk0 < 40M.
2’b11: 10M < inclk0 < 20M

PLL Phase Shift Settings

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

D1 : pll_phase
1’b1 : Enables real time reconfiguration of the PLL output clock phase shift

D2 : pll_sel
This bit selects one of the two PLLs (driving two LVDS TX shown in Fig. 2) for reconfiguration.

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[6]

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

Config 4-Config7 :

These registers are used to program maximum column, row and bank addresses up to which DDR memory is written or read as computed from the length of the DAC pattern.

mem_max_addr : {config7, config6, config5, config4}

mem_max_addr [9:0] : Maximum column address
mem_max_addr [25:10] : Maximum row address
mem_max_addr [26:28] : Maximum bank address
mem_max_addr [29] : Maximum chip select. This should be set zero as there is only one onboard DDR memory

Config 8-Config9 :

The firmware also provides the support of DAC preamble. It is a portion of the loaded pattern embedded in the first few samples and is sent only once to the DAC. When the memory loops around to resend the pattern, it reads from the starting address where the DAC samples are stored, skipping the preamble. Config8 and Config9 registers are used to specify the last address of the preamble The use of two registers limit the preamble size to 64M in multiples of 32.

mem_min_addr : {16’d0, config9, config8}

[5] Note that *inclk0* is the clock provided by the DAC

[6] Clocks have been drawn from post scale counters 0, 2, 3, 4and 5. See altera documentation “Clock Networks and PLLs in Stratix IV devices” for value corresponding to each of these counters

mem_max_addr [9:0]	:	Minimum column address
mem_max_addr [25:10]	:	Minimum row address
mem_max_addr [26:28]	:	Minimum bank address
mem_min_addr [29]	:	Minimum chip select.

Appendix A :

Phase Shifts Settings for PLL Output Clocks

For error free transmission of DAC samples, clocks going to *altlvds_tx* megafunction, as shown in Fig. 9, must be given certain amount of phase shift. The phase shift given depends upon serialization factor as mentioned in the altera design example on "Using altlvds With External PLL Option".

For a serialization factor of 8, the phase shifts to the output clocks of the PLL are given in Table 9

Table 9
Phase Shifts of PLL output clocks

Clocks	Phase Shift
C0 (clk_par)	-22.5^{o}
C2 (clk_dac)	0^{o}
C3 (clk_ser)	-180^{o}
C5 (clk_en)	270^{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	75M-200M	40M-80M	20M-40M	10M-20M
M	1(4)	2	3(4)	6
C0	1	1	1	1
C2	1(4)	2	3(4)	6
C3	1	1	1	1
C5	7(4)	14	33(4)	66

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).

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.

Appendix B :

RECEIVE DATA FORMAT EXPECTED BY THE FIRMWARE

As mentioned in the Host Interface module. samples by the GUI are transmitted over two SPI interfaces. All the I samples (samples corresponding to channels A and C for a four channel device.) are transmitted on *spi_mosi[0]* of the top level module, while the Q samples (samples corresponding to channels B and D) are transmitted on *spi_mosi[1]*.Received samples in the firmware are stored in two FIFOs, SPI_FIFO0 (for I samples) and SPI_FIFO1 (for Q samples) as shown in Fig. 2. The order in which samples are expected by the firmware for single and dual bus type DACs is shown in Table 11

SPI FIFO0 (I SAMPLES)		**SPI FIFO1 (Q SAMPLES)**	
Single Bus	Dual Bus	Single Bus	Dual Bus
A	A	B	B
C	A	D	B
A	A	B	B
C	A	D	B
A	C	B	D
C	C	D	D
A	C	B	D
C	C	D	D

Samples out of the Host Interface module are interleaved in the following order on 256-bit boundary

Signle Bus : { D,C,B,A,D,C,B,A}

Dual Bus : { D,C,D,C,B,A,B,A}

Note that the difference in the order of samples between single and dual bus mode arises from the fact that at the LVDS interface, in case of dual bus DACs, channels A and B are transmitted over one16-bit LVDS bus while those for channels C and D are transmitted over the other bus as shown in Fig. 2. However in case of a single bus DAC, all the channels are transmitted over the same bus one after the other.