Implementation of FPGA-Based Logic Blocks for High Speed Signal Processing System

Cao Bin, Chong Ming Ying, Kyaw Swa Maung, Lai Yoon Fei, Manoj Kumar Dey, and Sandeep Dattaprasad

***Abstract*—Processing digital signals acquired from high speed Analog Front Ends (AFE) is of interest in many consumer electronics and PC end applications. This paper presents a modular and structured architecture for processing high speed signals using Field Programmable Gate Arrays (FPGA). In particular it describes the various programmable elements necessary and challenges involved in building such a signal processing system. The approach described here is using Altera's Stratix III and Cyclone II FPGAs. The development was performed using Altera's Quartus 9.1 software environment.**

***Index Terms*—Digital signals, ADC, high speed, field programmable gate array.**

I. Introduction

Due to the consistent improvement in the FPGA technology, developing signal processing systems on a FPGA using Hardware Description Language (HDL) is not only highly flexible and efficient but also advantageous due to the ease of use. Digital systems of tremendous complexity can be implemented on a single FPGA device. However, the performance and efficiency of processing signals > 1GHz largely depends on efficiency of architecture and the optimization techniques used in the design.

With development of fast Analog-to-Digital Converters (ADC) and other high speed interface technologies with processing speed of above 5G Samples / sec (GSps), there is a need for developing an efficient and reliable signal processing and acquisition system which plays a major factor in design and development of modern electronic systems.

The purpose of this paper is to present a modular and structured FPGA architecture and the design challenges involved in developing such a system. The design and experiments were done with data coming from EV8AQ160, a Quad ADC from E2V. Following section describes the different design blocks involved.

Fig. 1 shows the logic blocks of the architecture described in subsequent sections.

Manuscript received April 29, 2012; revised June 7, 2012.

Chong Ming Ying, Kyaw Swa Maung, Lai Yoon Fei, Manoj Kumar Dey, and Sandeep Dattaprasad are with Seagate Techonology, EAO, Advanced Development Engineering, 7000, Ang Mo Kio Ave 5, Singapore 569877(e-mail: sandeep.dattaprasad@gmail.com).

Cao Bin is with Institute for Infocomm Research, Embedded Systems Department, Singapore 138632

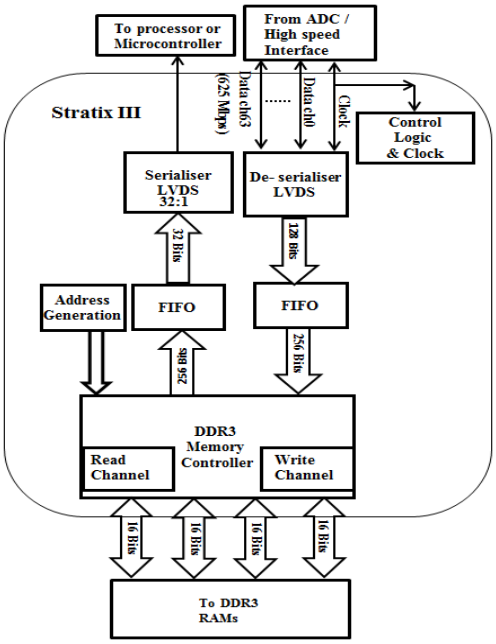

Fig. 1. FPGA logic blocks

II. Design Blocks

A. Stage 1: De-Serializer / FIFO 1

The first step is to de-serialize the incoming ADC data. There are 8 groups of outputs from the ADC, each containing 8 bits. The incoming 64-bit ADC value is converted into 128-bit single data rate (SDR) ADC value through a LVDS receiver. ADC chips work at one-channel mode and1:2 DMUX mode. The 128-bit data is clocked on the rising edge of clock signal.

The data is then sent to LVDS FIFO which is instantiated using Altera's parameterizable Megacore® IP functions. The Dual Clock mixed width Megafunction (DCFIFO) supports different write input data and read output data widths. This application has an incoming data stream width of 128 bits and the output data width would be 256 bits. The FIFO depth can be adjusted to any value between 2048 words to 131072 words as required.

To ensure the contents don't get corrupted, the OVERFLOW_CHECKING parameter is turned on. This ensures that the write request signal is ignored when FIFO is full and hence prevents any corruption of data. Similarly, the UNDERFLOW_CHECKING is turned on to ensure the read request is ignored when FIFO is empty. The synchronization stages from the write control logic to the read control logic and vice versa is set to 4.

B. Stage 2: DDR3 Memory Controller

The data from the FIFO will be stored in a DDR3 memory device and retrieved as and when a read instruction is received. To accomplish this, it's imperative to design

1) Memory controller and
2) PHY interface.

The memory controller used is Altera's High Performance Memory Controller II (HPC II). It is generated using Megafunction HPC II in Quartus software. This function initializes the memory devices, manages SDRAM banks, translates read / write requests from local interface in to SDRAM commands and also takes care of reordering the commands. The frequency of the controller can either be equal to the memory interface frequency (full-rate) or half of the memory interface frequency (half-rate). For a half-rate controller, the memory clock runs twice as fast as the clock provided to the local interface; so data buses on the local interface are four times as wide as the memory data bus. For a full-rate controller, the memory clock runs at the same speed as the clock provided to the local interface, so the data buses on the local interface are two times as wide as the memory data bus. Each read or writes request on the local interface fits into a single memory read or writes command on the memory interface, simplifying the controller design.

The Command Queue Look-Ahead Depth is set to 4 and local maximum burst count is 4.

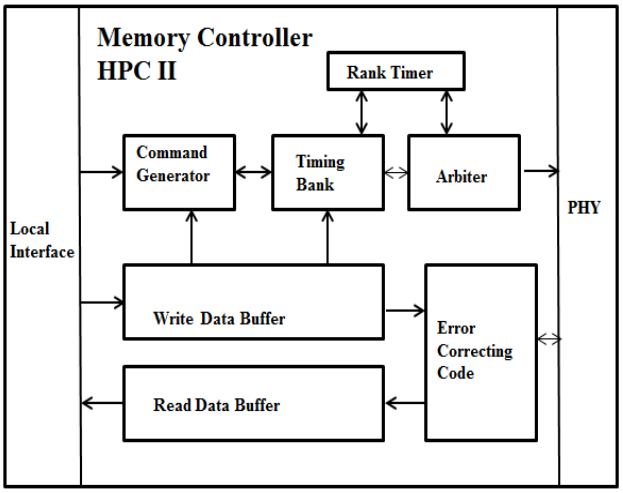

Fig. 2. Memory controller

Along with the controller, ALTMEMPHY is generated using Altera's Megafunction. The ALTMEMPHY is a PHY interface between a memory controller and memory devices and performs read and write operations to the memory.

The PLL reference clock frequency for ALTMEMPHY is set to 20MHz and the memory clock frequency for DDR3-800 is set to 400MHz.

In addition, the dynamic parallel on-chip termination (OCT) is enabled in the PHY settings. A dedicated clock is used for clocking address/commands. Depending on the number of layers the PCB has and expected skew on the signals, the board skew is set in board timing parameters tab. For our application based on the PCB it was set to 20 ps.

The driver controls the communication between the ADC and DDR3 during write operations and also between DDR3 and the user interface during read operations. The driver is responsible for processing the read requests, write requests, generating DDR3 addresses, generating sync header and End Of Packet (EOP). The driver performs all these operations with a state machine. The state machine is responsible for transitioning from one read/write state to another and at the same time update the address counter of the DDR3 rows and columns. The state machine waits for the write requests to receive the data from ADC. Once it receives the write requests it checks to see if the controller is ready. The controller asserts "local ready" signal after which the write cycle begins. The data from LVDS FIFO is sent to the row, column and bank address of the DDR3 as generated by the address generator. The write cycle continues until the ADC is disabled by the user interface.

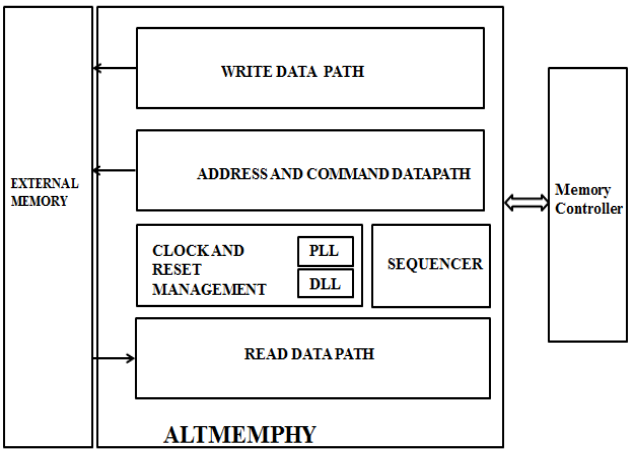

Fig. 3. ALTMEMPHY architecture

C. Stage 3: DDR3 Memory Controller Driver

The buffer in DDR3 memory is circular, so once the memory fills up, it begins to over write from the starting address and upon detecting the required trigger event it writes until the trigger delay time expires and then stops. Designing a circular buffer is especially important since when triggered, it is imperative to capture the pre-trigger and post-trigger events.

Once the ADC is stopped, the user interface issues a read request to fetch the data from the DDR3. The state machine services this read request after receiving local ready signal from controller by reading the rows, columns and banks of DDR3. The data thus read is passed on to the next stage.

In addition to this, the driver inserts a header at the start of the packet and an End of Packet frame at the end. This is for verifying the data integrity during transmission. The user interface needs to verify the header and EOP when it receives the packet from Stratix III.

D. Stage 4: FIFO 2

The data read from stage 3 is sent to a second FIFO in Stratix III. It acts as a buffer between LVDS transmitter and the memory controller read channel. FIFO2 is instantiated from Altera's Megafunction Dual Clock FIFO (DCFIFO) which has an input width of 256 bits, Output width of 32 bits and Depth of 128 words. The FIFO is set to have 4 sync stages. The size of FIFO 2 hence would be 256-bit × 128 = 1024 Bytes.

E. Stage 5: LVDS Serializer / transmitter

The LVDS transmitter (Tx) is generated by MegaWizard Plug-in Manager using Altera's IP ALTLVDS. The 32 bit data stream from FIFO 2 is serialized with a serialization factor of 32. The Output data rate is 640Mbps.

III. User Interface and Secondary FPGA

When there are multiple ADC channels, the data is collected from respective Stratix III FPGAs. A secondary FPGA or micro controller, whose objective is to receive data from all the different channels, check for data integrity, convert the commands to user interface protocol such as PCI or PCIe or USB and send it to the user interface. In this application, an external Cyclone II FPGA was used for this purpose since it offers the design flexibility and is easy to integrate with Stratix III. The internal architecture of the design implemented on the secondary FPGA is as shown below.

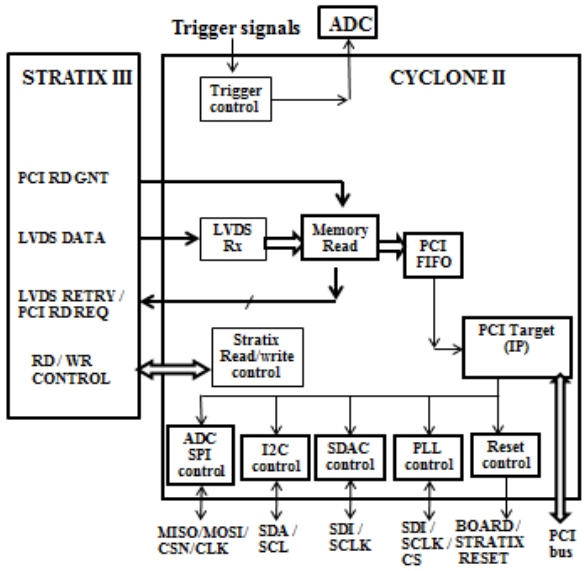

Fig. 4. Secondary FPGA

Once the trigger signal is detected by trigger circuitry, Cyclone II sends a command to stop sampling and Stratix III ends the write cycle to DDR3. Cyclone II generates a read request to read the data in the DDR3 that is interfaced with Stratix III. When Stratix III responds with the requested data, Cyclone II needs to de-serialize the incoming LVDS data. This is done using LVDS receiver.

The data is then sent to a memory read block. When ADC data are stored in DDR3 memory, the memory read block issues memory read request to Stratix III FPGA. When memory read grant signal is received, memory read block receives ADC data and clocks. The data is then checked for correctness by looking at it's header and EOP and then sent in to a PCI FIFO. When errors are found in the received ADC data, LVDS retry signals are sent out to Stratix III to request a re transmission of ADC data again.

The PCI FIFO is a buffer between PCI target and LVDS Rx. It is generated by Quartus software, and is an instantiation of Altera's Megafunction Dual Clock FIFO (DCFIFO). The width of the FIFO is 32 bits and depth is 1024 words.

The FIFO then sends the data to a PCI target IP which converts the incoming data in PCI protocol and sends it to PC host. PCI target core is an instantiation of Altera's Megafunction PCI target (pci_t32). PCI target is a bridge between PCI master (residing on mother board) and local memory mapped interfaces. It runs at 33MHz and its bus is 32-bit wide. It is set to have following PCI configuration: Device ID: 4; Vendor ID: 0x1172; Revision ID: 1; Subsystem ID: 0x389; Subsys Vendor ID: 0x1172; Minimum Grant: 0; Maximum Latency: 0; BAR0 = 2MBytes;

Cyclone II also has trigger control circuit. Trigger control circuit is enabled by settings in trigger control register inside PC host driver. It is responsible for enabling or disabling the ADC depending on whether or not the trigger condition is met. Once the delay counter expires, the trigger control block disables ADC from getting any more samples. This ensures that the post trigger events are captured.

IV. PC Host Driver

The driver on the PC host communicates with cyclone II FPGA through PCI bus. The driver provides GUI for the user to set up PCI register and configuration space. With appropriate pre-processing, this methodology can be employed to process signals from variety of high speed signals (> 1 Giga Hz range) from interfaces such as PCIe, USB, DDR, Ethernet...etc. to PC host.

V. Issues Encountered and Workarounds

As described before, the memory controller driver needs to wait for local ready from the memory controller IP, which lets the user logic- know if the controller is ready to accept a transfer command, or if it is busy and thus the user logic must wait. A common issue with the local ready signal is that it permanently goes low preventing any further commands from being accepted by the controller, effectively resulting in the controller being 'locked up'.

The most common cause of this is the timing error in the Altera memory controller. In particular, when bursts of writes are requested, any delay in providing all beats of that write access to the controller before another command is requested could cause this issue. As the controller waits for the missing write beat, it's command FIFO will fill up with other commands (usually reads) and once full it fails to de-assert local ready indicating that it is busy and thus the driver/user logic must wait. This issue has been documented by Altera [10].

One possible workaround that we implemented is - when it locks up, the Stratix stops sending data and PCI interface issues a retry, resetting the memory controller. This effectively restarts the state machine inside the controller and it issues local ready signal resolving the lock up.

References

[1] P. J. Ashenden, "*The designer's guide to VHDL*," Morgan Kaufmann, 2008.

[2] P. J. Ashenden, J. P. Mermet, and R. Seepold, "*System-on-chip methodologies and design languages*," Springer, 2001

[3] P. Z. Peebles, Jr, "*Digital Communication Systems*," Prentice-Hall, INC.
[4] Reference Manual, "*External Memory Interface Handbook*," Altera Corporation, vol. 3, 2011.
[5] A. M. U. Guide, "*External Memory PHY Interface (ALTMEMPHY)*," Altera Corporation, 2011
[6] A. M. U. Guide "*I/O Buffer (ALTIOBUF)*," Altera Corporation, 2011
[7] AN 436: "*Using DDR3 SDRAM in Stratix III and Stratix IV Devices*", Altera Corporation
[8] A. Wiki: "*Local ready signal issues with alter external memory controller, IP*," [Online]. Available: http://www.alterawiki.com/wiki/Local_ready_signal_issues_with_Altera_external_memory_controller_IP,Altera,Corporation, November 2010.