Summer Internship: Project Report

Submitted By: JAYANT THATTE

Roll No.: EE09B109

Company: BITMAPPER INTEGRATION TECHNOLOGIES PVT. LTD.

Industry: VLSI/FPGA PLATFORM DEVELOPMENT

Abstract

The first week of the internship period was spent in revising basic concepts of VHDL and Verilog coding. I was also introduced to other hardware design related issues like static timing analysis, simulations, synthesis and basics of PLDs (internal architecture, various types etc.). To get better acquainted with the design and coding techniques, various test-codes were written for testing the various peripherals (starting from basic LED testing and including tests for ADC, DAC, RS232 and Flash memory testing). Through these codes, I got well acquainted with Xilinx design tools such as ISE, Plan Ahead as well as Altera tools like Quartus and more advanced NIOS.

After getting well-versed with design techniques, the project for designing a QDR Controller IP was started. The IP consisted of hardware design code and application software. The hardware design code was written in VHDL and implemented on Xilinx Spartan 6 FPGA. The application software was written in C++ programming language to transact data and commands between computer and the FPGA. At the end of the designing phase, the Controller was tested at a satisfactory data rate of about 100 megabytes per second on the Spartan 6 FPGA. Designing and successful testing of this controller took about 5 weeks.

The last two weeks were spent in studying and understanding the EMIF (External Memory Interface) bus, which is used to interface DSP with its peripherals including FPGA. After acquiring a detailed understanding of the EMIF bus, an application was written to accept data from ADC (interfaced to Xilinx Vertex 4 FPGA) and to transfer it to Texas Instruments DSP. This data was then looped back using the EMIF bus and displayed live on Oscilloscope through DAC interfaced again to FPGA. The data taken from ADC can be output to DAC using the EMIF interface after the data is processed by the DSP.

Table of Contents

Abstract 0

QDR Controller IP 2

Basics of QDR 2

Timing Issues 2

Outline of the Hardware Design 2

Specifications 2

Design Details 2

Application Software 4

Tests Performed 4

EMIF Bus Application 4

Basics of EMIF Bus 4

Design Details: Board Specifications 4

Basic Codes 5

FPGA Program 5

DSP Program 6

Programming DSP PLL using GEL File 6

Basics of GEL File 6

DSP Registers to be Programmed using GEL File 6

Configuring DSP Timing Characteristics 7

Tests 7

References 7

QDR Controller IP

Basics of QDR

A QDR is a type of SRAM that can transfer up to four words in every clock cycle, hence the name Quad Data Rate memory. QDR memory has separate ports for read and write and each port is Dual Data Rate (DDR). The main purpose of QDR is that it enables fast and independent reads and writes, at high clock frequencies without losing bandwidth in due to bus turnaround time as opposed to DDR memories.

Timing Issues

Each port of a Quad Data Rate memory transacts data twice every clock cycle and is designed to operate at high frequencies (typically few hundred MHz) in burst mode, whereas a design implemented on FPGA cannot operate at this frequency. This is one of the main challenges in FPGA designs involving QDR. This problem was overcome by making use of input/output DDRs† (ODDR and IDDR). Timing is a very critical issue in QDR designs. Hence, softwares such as Xilinx Plan Ahead were used to achieve a good timing performance.

† An ODDR accepts two clock inputs (typically a differential clock) and two data inputs, each of them latched on the rising edge of the respective clock. This data then comes to the output via a multiplexer which passes the inputs alternately. Thus an ODDR accepts data from two inputs at a given frequency and outputs data at twice the frequency. An IDDR functions exact opposite of ODDR. It accepts input at a given frequency and gives two outputs at half the frequency.

Outline of the Hardware Design

The various steps involved are:

1. Implementing the design file by file in ISE
2. Download .bit file into Spartan 6 FPGA
3. Checking read and write functionality using GUI based software

Specifications

- Cypress **CY7C15632KV18** Quad Data Rate memory, 72 Mbit, 4 word burst
- Xilinx Spartan 6 FPGA

Design Details

Given below is the block diagram of the QDR Controller IP.

Data sent by the application software interfaced passed on to the FPGA via USB connection. The data is accepted within the FPGA in a USB controller. USB controller consists of two FIFOs, one which accepts data from the USB and the other which sends data to USB. The two FIFOs are controlled by read/write logic of the USB controller.

The command structure consisted of a start byte and an end byte, three bytes each for passing the file size and start address and one byte for the actual command (in this case read or write). The command is decoded in the instruction decoder and important information such as the starting address, file size, read/write etc. are sent to the QDR read-write module.

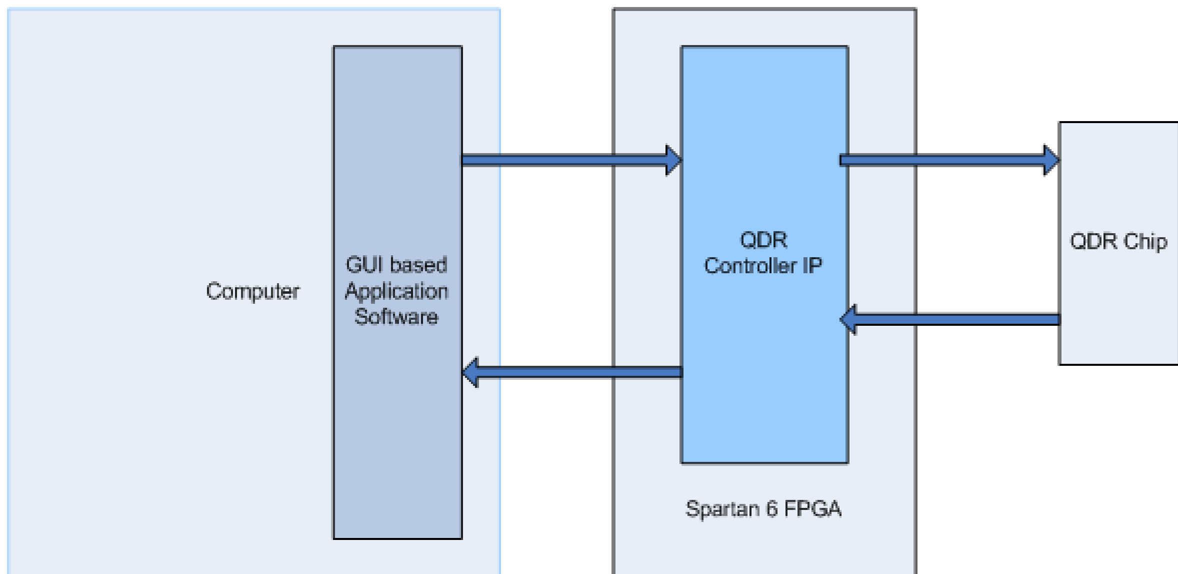

Figure 1: Top Level Block Diagram

The QDR read-write module accepts data coming from USB and passes it on to QDR via ODDRs during the write operation. Similarly, during a read operation, data is read from QDR and passed to USB controller through IDDR. The ODDRs and IDDRs help in reducing the frequency at which FPGA must operate for desired data transaction rate in QDR. Data transacted to or from the QDR using state machines written after taking into consideration the timing characteristics of the QDR memory chip.

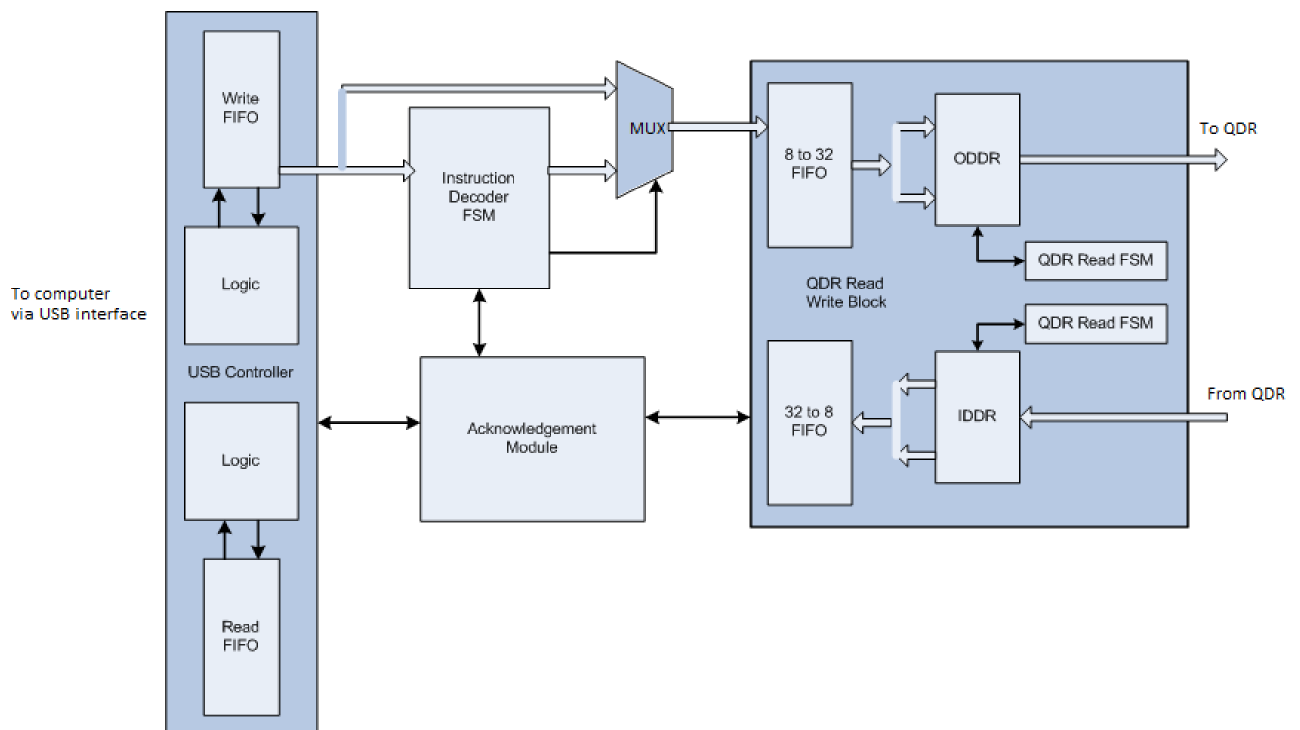

Figure 2: Detailed Block diagram

Application Software

Commands are accepted from the computer via the application software. The command is then processed in hardware and data is read from or written to the QDR memory accordingly. The application software reads data from the test files and sends data to USB using 512 byte buffer. Similarly, data is accepted from USB into the buffer and then written to the output file. The input and output files are compared to test the read and write operations performed on QDR memory.

Tests Performed

- Simple Counter Test: Serial values were written to serial memory locations, read back and verified.
- Data Loopback: After the counter test was successful, the developed IP was tested through read/write loopback testing for files of various sizes. The IP gave a satisfactory performance with a data rate of about **100 megabytes** per second.

EMIF Bus Application

Basics of EMIF Bus

EMIF stands for external memory interface. It is an interface used to connect DSP processors to various peripherals. All devices connected to DSP are mapped in a memory map. DSP is able to communicate with these devices by writing appropriate values to various control registers of the EMIF bus and by reading or writing data to the correct address locations in this memory map.

FPGA DSP interface is a little more complicated as both FPGA as well as DSP need to be programmed. DSP is programmed with an application program written in C++, while the FPGA is programmed with .bit file.

Design Details: Board Specifications

- **FPGA**: Xilinx Virtex 4 **(XC4VSX55-FF1148)** having 55,296 logic cells.
- **DSP**: DaVinci Processor from Texas Instruments (**TMS320DM6467**).
- **ADC: High Speed ADC (ADS5527)**
  - 4 channels.
  - 12 bit Analog to Digital Converter from Texas Instruments.
  - Sampling Speed – 210 Msps.
  - Analog Input range - 2 Vpp.
- **DAC: High Speed DAC (AD9742)**
  - 4 channels.
  - 12 bit Digital to Analog Converter from Texas Instruments.
  - Sampling Speed – 210 Msps.
  - Analog Input range - 2 Vpp.
- **EMIF Bus**: 16 bit data width, Asynchronous.

Basic Codes

FPGA Program

FPGA is programmed using the .bit file in <FOLDER>. Data is accepted from ADC into a FIFO. The clock-out given by ADC is used as write clock for this FIFO. Data from this FIFO is sent to DSP using the bi-directional EMIF (**E**xternal **M**emory **I**nter**f**ace) bus. From DSP data is looped back using the same EMIF bus. This data is accepted in another FIFO from where it is read by DAC. The SYNC-clock for DAC is used as read clock for this DAC-FIFO.

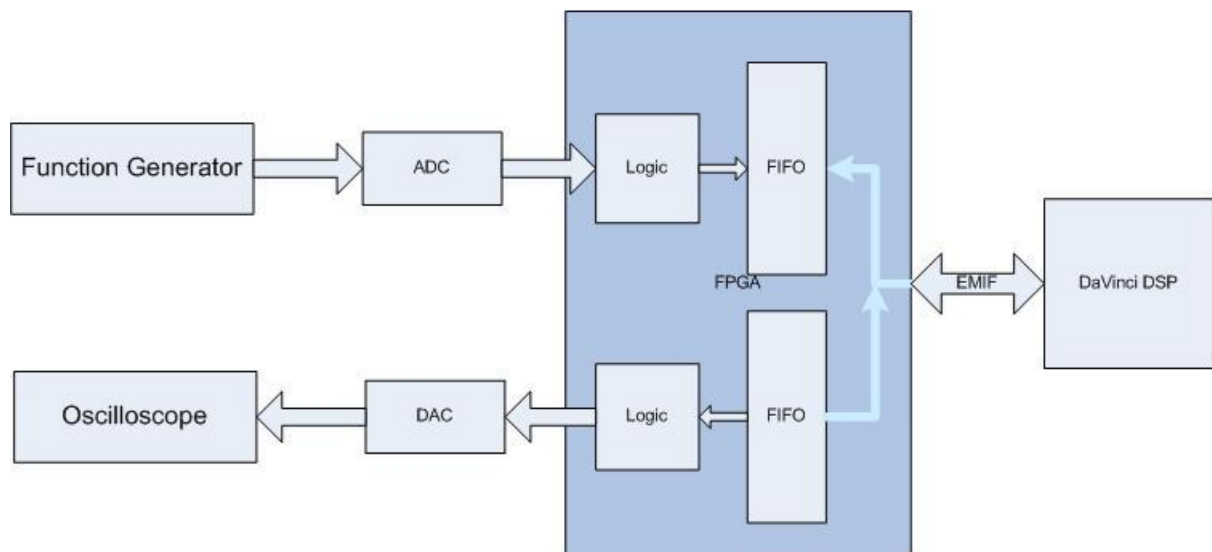

Figure 3: Top Level Block Diagram

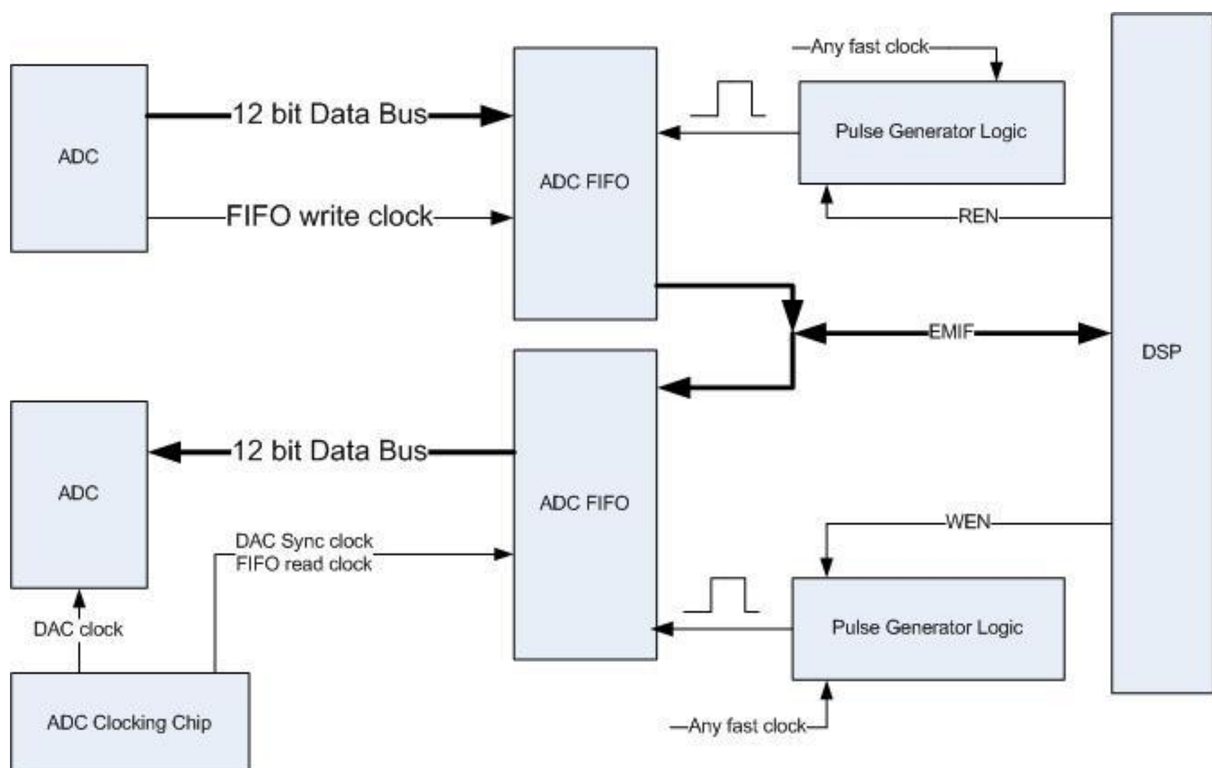

Figure 4: Detail Block Diagram

DSP Program

DSP is coded using Code Composer Studio Version 4. Data is accepted from FPGA using EMIF bus and then sent back using the same bus. Single data is read and then written immediately. Only after the previous data is written back to FPGA, next data is read from it. This is done to maintain a continuous supply of data to the DAC.

Programming DSP PLL using GEL File

Basics of GEL File

GEL file is a C program file which the DSP uses to configure itself before loading the user application program. GEL file is loaded into the DSP at the beginning of "Debug" session in CCSv4. Writing correct setting in the GEL file is necessary for proper functioning of the application program in DSP.

DSP Registers to be Programmed using GEL File

PLL Control Register (PLL_CTL)

- **Function**: Controls all the important settings of PLL
- **Location**: This is a 32 bit register located at address **0x01C40900**.
- **Setting**:
  - ✓ CLKMODE (bit 8): To be set to 1 if using external clock source for PLL (as in our case).
  - ✓ PLLENSRC (bit 5): This bit must be cleared before PLLEN will have any effect.
  - ✓ PLLDIS (bit 4): Setting this bit disables PLL. When using PLL this bit must be cleared.
  - ✓ PLLRST (bit 3): Making this bit 0 asserts reset to PLL. When using PLL this bit must 1.
  - ✓ PLLPWRDN (bit 1): Make this bit 1 to power down PLL; 0 for normal operation.
  - ✓ PLLEN (bit 0): PLL is bypassed when this bit is cleared. Set this bit to use PLL.
- **PLL Initialization:** Following steps must be followed in your GEL file to initialize PLL.
  1. `Set clock mode (CLKMODE)`
  2. `Set PLL to bypass, wait for PLL to stabilize (PLLEN = 0, wait)`
  3. `Reset PLL (PLLRST = 0)`
  4. `Disable PLL (PLLDIS = 1)`
  5. `Power up PLL (PLLPWRDN = 0)`
  6. `Enable PLL (PLLDIS = 0)`
  7. `Wait for PLL to stabilize`
  8. `Load PLL multiplier (Refer PLLM register)`
  9. `Set PLL post dividers (Refer PLLDIVn registers)`
  10. `Wait for PLL to reset (PLLRST = 0, wait)`
  11. `Release from reset (PLLRST = 1)`
  12. `Wait for PLL to re-lock`
  13. `Switch out of bypass mode (PLLEN = 1)`

PLL Multiplier Control Register (PLLM)

- **Function**: Controls PLL multiplier value
- **Location**: This is a 32 bit register located at address **0x01C40910**.
- **Setting**: Multiplier value = PLLM (4 : 0) + 1
- **Note:** There is an allowable range for PLL multiplier (PLLM). There is a minimum and maximum operating frequency for DEV_MXI/DEV_CLKIN, PLLOUT, AUX_MXI/AUX_CLKIN,

and the device clocks (SYSCLKs). The PLL Controllers must be configured not to exceed any of these constraints documented in this section (certain combinations of external clock inputs, internal dividers, and PLL multiply ratios might not be supported). For our device allowed multiplier values are **between 14 and 22**. PLLM value used in the code is 21 implying a multiplier value of 22.

PLL Divider Control Registers (PLLDIVn)

PLLDIVn controls divider value for SYSCLKn. EMIF bus runs on SYSCLK3. Default value of PLLDIV3 is 4. Values for PLLDIV1, 2, 3 are fixed while others are programmable.

Configuring DSP Timing Characteristics

- **Function:** DSP timing characteristics can be configured writing appropriate values to Asynchronous n Configuration Registers (AnCR). There are four such registers, one for each select space. For our application the register corresponding to FPGA (CS4/A3CR) was configured.
- **Location**: This is a 32 bit register located at address **0x20008018**.
- **Setting**:
  - ✓ EW (bit 30): Bit must be cleared to disable Extended Wait. The bit should NOT be set if the device does not have an EM_WAIT pin.
  - ✓ W_SETUP (bits 29:26): Write Setup Time = (W_SETUP + 1)*EMIF Clock Period.
  - ✓ W_STROBE (bits 25:20): Write Strobe Width = (W_STROBE + 1)*EMIF Clock Period.
  - ✓ W_HOLD (bits 19:17): Write Hold Time = (W_HOLD + 1)*EMIF Clock Period.
  - ✓ R_SETUP (bits 16:13): Read Setup Time = (R_SETUP + 1)*EMIF Clock Period.
  - ✓ R_STROBE (bits 12:7): Read Strobe Width = (R_STROBE + 1)*EMIF Clock Period.
  - ✓ R_HOLD (bits 6:4): Read Hold Time = (R_HOLD + 1)*EMIF Clock Period.
  - ✓ TA (bits 3:2): Bus Turn Around Time = (TA + 1)*EMIF Clock Period.
  - ✓ ASIZE (bits 1:0): 00 if EMIF bus width is 8 bit, 01 if 16 bit.
- **Note**: EMIF Clock is DaVinci SYSCLK3. The frequency of this clock may be calculated knowing the PLL setting (enabled/bypassed), multiplier values.

Tests

Live data loopback takes place at a rate of about 400 kHz. Signals fed into ADC can be viewed on an oscilloscope. Signals with frequencies up to 200 kHz get looped back successfully (with little or no glitches). Signals of higher frequencies get somewhat degraded.

References

- Xilinx Documentation
- Cypress CY7C15632KV18 Datasheet
- FTDI Chip FT2232H Datasheet
- Interfacing Xilinx FPGAs to TI DSP Platforms using EMIF Bus: XAPP753 (v2.0.1)
- TMS320DM646x User Guide for Asynchronous EMIF (sprueq7c)
- Creating Device Initialisation GEL fIles (SPRAA74A)
- Code Composer Studio v4 Documentation