

# DEVELOPMENT AND TESTING OF VHDL INTERFACES FOR HIGH SPEED MEMORY BUFFERING AND DATA TRANSMISSION ON FPGA DEVELOPMENT KIT FOR HIGH SPEED DIGITIZER

**Pragati Gupta, Vikas Kumar Rai**

<sup>1</sup>Student, Dept. of Electronics and communication, SIT, Mathura, India

<sup>2</sup>Associate Professor, Dept. of Electronics and communication, SIT, Mathura, India

\*\*\*

**Abstract** - High speed physics experiments involve data acquisition and transmission from large number of channels. Transmission of huge amount of data within a short period of time is a demand of today's world. This paper deals with the development of VHDL interfaces for high speed memory buffering and data transmission. High speed digitizers sample and digitize analog signals upto 250 MSPS (mega samples per second). ADC data is processed using suitable signal processing algorithm. In order to achieve the high data throughput, memory buffering is used. Processed data is transferred over communication channel using PCI express. This paper deals with the High speed data transfer at 400 MHz clock on DDR3 memory.

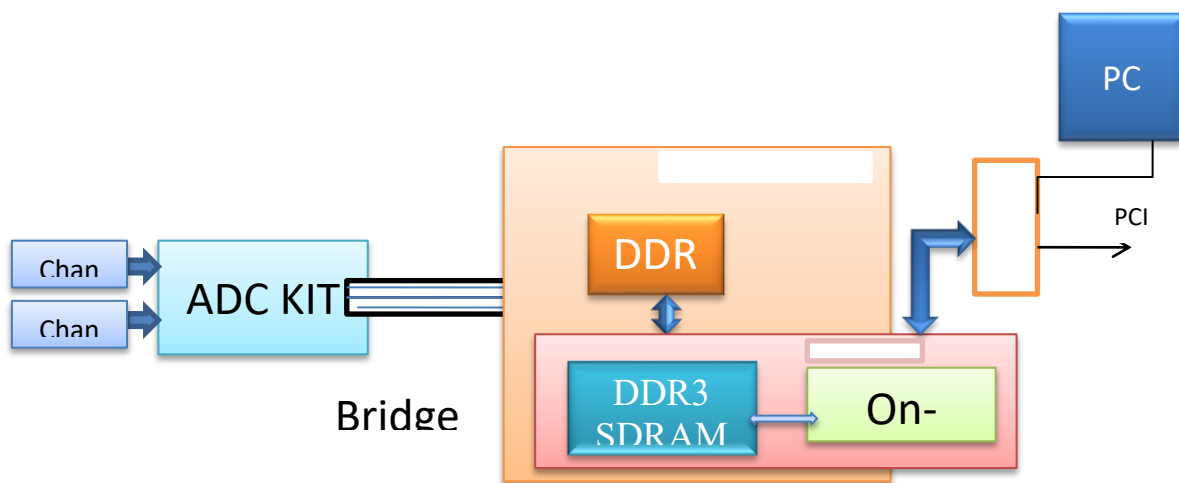
**Key Words:** External memory interface, qsys(quartus system integration tool), DMA controller, Avalon interfaces, Memory buffering.

## 1. INTRODUCTION

High speed digitizers produce large amount of data during data acquisition. Online digital data processing algorithms requires large amount of data to be buffered till the processing is being carried out. Processed data has to be transmitted to Data Archiving PC. In order to achieve the high data throughput, memory buffering is used. The scope of this paper is to transfer (read/write) the high speed data at 400Mhz clock on DDR3 SDRAM. This data is transferred over communication channel using PCI express.[5]

### 1.1 Cyclone V FPGA DEVICE:

The Cyclone V GT FPGA development board provides a hardware platform for developing and prototyping low-power, high-performance, and logic-intensive designs using Cyclone V GT FPGA device. The board provides a wide range of peripherals and memory interfaces to facilitate the development of Cyclone V GT designs. Design advancements and innovations, such as the PCI Express hard IP, partial reconfiguration, and hard memory [1]controller implementation ensure that designs implemented in the Cyclone V GTs operate faster, with lower power.



**Fig -1:** Block Diagram of the device

## 1.2 EXTERNAL MEMORY INTERFACE:

DDR3 SDRAM is the third generation of SDRAM. DDR3 SDRAM is internally configured as an eight-bank DRAM and uses an 8n pre-fetch architecture to achieve high-speed operation. The 8n pre-fetch architecture is combined with an interface that transfers two data words per clock cycle at the I/O pins. A single read or write operation for DDR3 SDRAM consists of a single 8n-bit wide, one-clock-cycle data transfer at the internal DRAM core and eight corresponding n-bit wide, one-half clock cycle data transfers at the I/O pins.[4]

UNIPHY: It is a physical layer of the external memory interface. It converts the double data rate interface of high speed memory devices to a full rate or half rate interface for use within an FPGA.

## 1.3 Avalon Interface specifications:

Avalon interfaces simplify the system design by allowing to easily connection in the components in FPGA. The Avalon interface family define interfaces appropriate for streaming high speed data, reading and writing registers and memory, and controlling off-chip devices. System interconnect is a high-bandwidth structure for connecting components, and that allows us to connect IP cores to other IP cores with various interfaces.[8] We are using Avalon streaming interface, Avalon memory mapped interface and Avalon conduit interface, Avalon clock and Avalon reset interface.

## 1.4 DMA Controller:

The direct memory access (DMA) controller core with Avalon interface performs bulk data transfers, reading data from a source address range and writing the data to a different address range. An Avalon Memory-Mapped (Avalon-MM) master peripheral, such as a CPU, can offload memory transfer tasks to the DMA controller. While the DMA controller performs memory transfers, the master is free to perform other tasks in parallel. [4]

The DMA controller transfers data as efficiently as possible, reading and writing data at the maximum pace allowed by the source or destination. The DMA controller is capable of performing Avalon transfers with flow control, enabling it to automatically transfer data to or from a slow peripheral with flow control (for example, UART), at the maximum pace allowed by the peripheral.

## 1.5 Memory buffering:

A memory buffer register (MBR) or memory data register (MDR) is the register in a computer's processor, or central processing unit, CPU, that stores the data being transferred to and from the immediate access storage. It contains the copy of designated memory locations specified by the memory address register. It acts as a buffer allowing the processor and memory units to act independently without being affected by minor differences in operation.[6] A data item will be copied to the MBR ready for use at the next clock cycle, when it can be either used by the processor for reading or writing or stored in main memory after being written.

This register holds the contents of the memory which are to be transferred from memory to other components or vice versa. A word to be stored must be transferred to the MBR, from where it goes to the specific memory location, and the arithmetic data to be processed in the ALU first goes to MBR and then to accumulated register, and then it is processed in the ALU.

## 2. READ/WRITE DDR3 SDRAM:

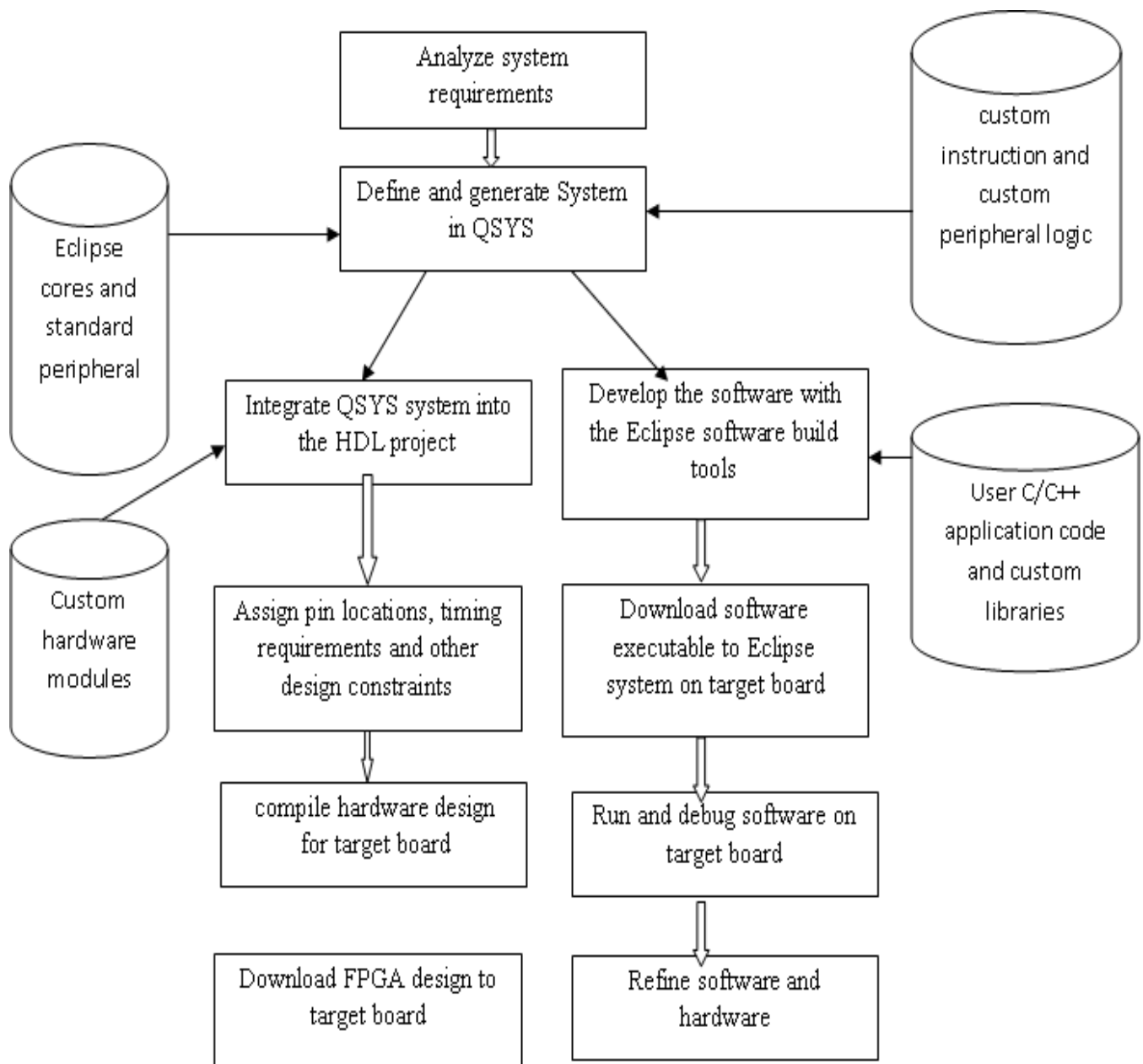
Here, we are reading and writing the DDR3 SDRAM by giving the software commands in the NIOS Embedded design suite using the Eclipse processor.

There are following components required in our NIOS system-

- Clock.
- DDR3 SDRAM controller with UniPHY.
- Nios II processor.
- On-chip memory.
- JTAG UART.

After creating a qsys system design and instantiation in the Quartus, pin assignment is performed to connect the design to the hardware for the proper read/write of SDRAM.

For the configuration of the FPGA we have programmed our design on the FPGA design toolkit using the programmer. The .sof file which contains the hardware design is programmed on FPGA. Writing a software program in eclipse tool to write/read DDR3 SDRAM[6]



**Chart -1:** Eclipse EDS system development flow.

Using the eclipse NIOS tool we can easily communicate with the device for the efficient transfer of the data. Here, Chart 1 shows the development flow of eclipse NIOS II tool. After creating a new project in the embedded design suite processor, we created a software program and generated the board support package (BSP) file. We are accessing the DDR3 by writing and reading to some specific addresses by giving the commands in the EDS processor. Figure 2 shows the Eclipse console window for DDR3 read/write on specified addresses.[7]

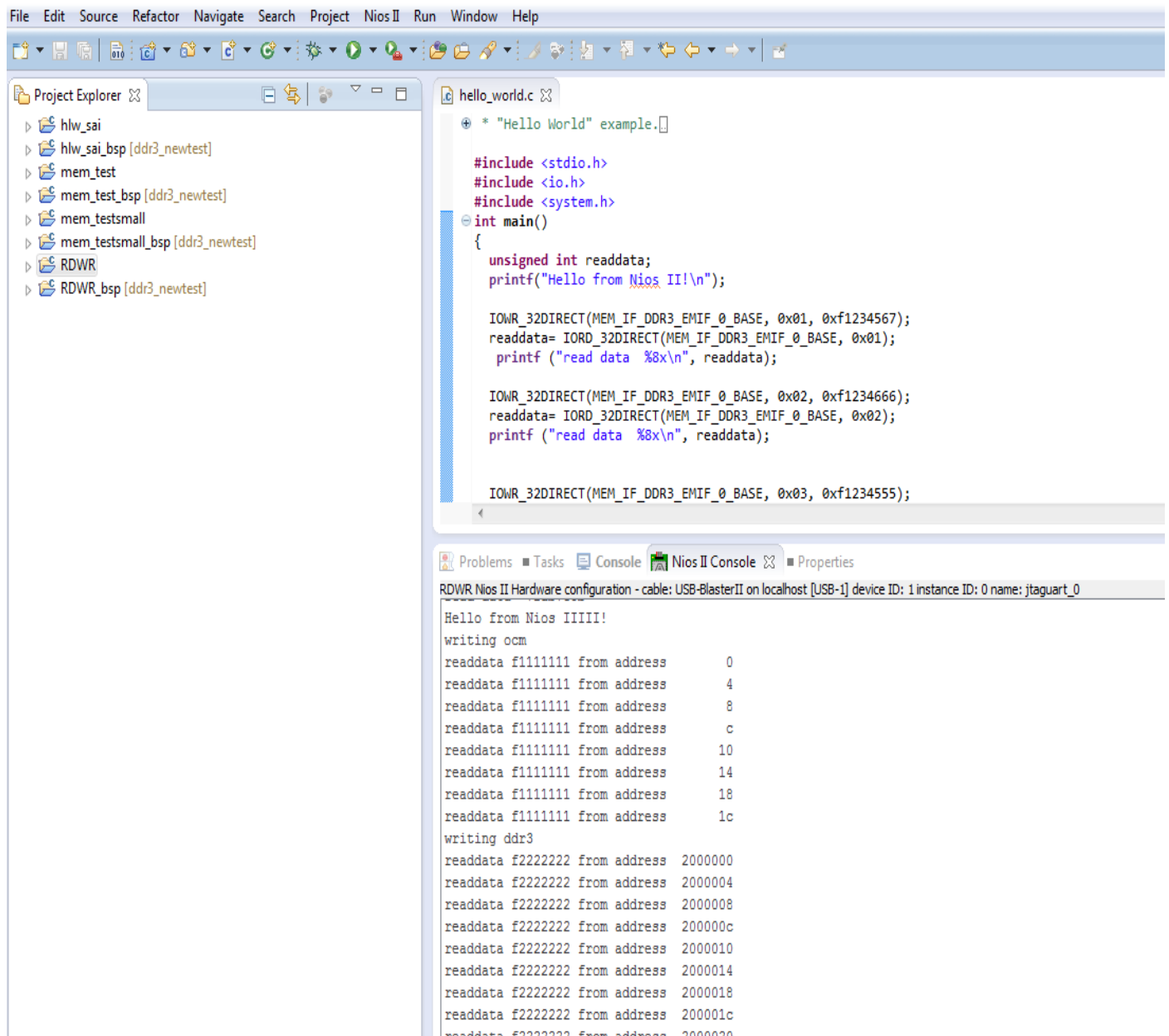


Fig -2: Eclipse console window showing DDR3 read/write on specified addresses.

### 3. Throughput analysis of DDR3 SDRAM Controller:

We are designing a system to transfer the data in between the on-chip memory and the DDR3 SDRAM controller.

Firstly, we write our data on the on-chip memory.

Secondly, we initialized our DMA Controller by giving the read address (address of the memory from which the data is to be read), write address (address of the memory from which the data is to be write), DMA transaction length (in bytes), status register and control port register. In our case the reading memory is on-chip memory and the writing memory is DDR3 SDRAM. After writing the control port command our data is transferred directly from on-chip memory to the DDR3 SDRAM.

Thirdly, reading and printing the data from DDR3 SDRAM serially.

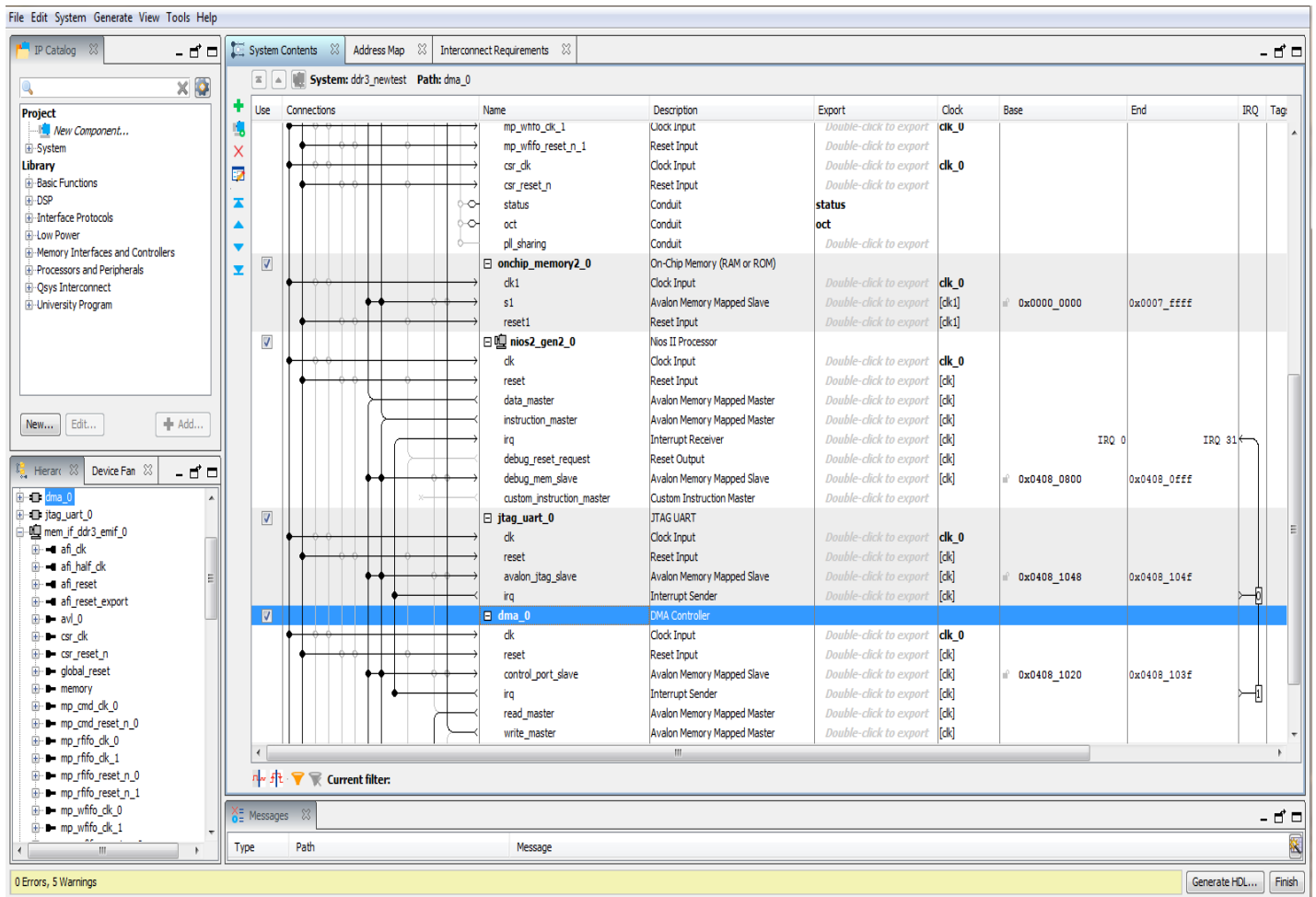


Fig -3: Complete QSYS design for the transfer of data through DMA controller.

We have successfully transferred our data through DMA between the memories as shown in the figure 4 the console is showing the read-data that is transferred in the DDR3 SDRAM. The main objective is to find the time taken in the transfer of the data between the On-chip memory and the DDR3 SDRAM.

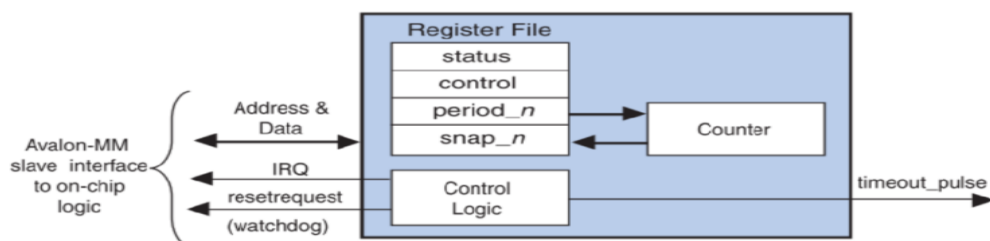


Fig -4: Interval timer core block diagram.

For the calculation, the time of the transfer we are using the interval timer in our design. There are following steps used in this design:

1. Adding timer in the system integration tool of our later design. Initialization of the parameters for the interval timer.
2. Generation of the design and then integrating the QSYS design with the System design tool project.
3. Simulation and compilation of the final design.
3. Programming the design on the programmer.
4. Initializing the timer before the DMA controller control port command in embedded design suite processor.
5. Reading the counter snapshot before and after the transfer. This Timer works in countdown mode already.



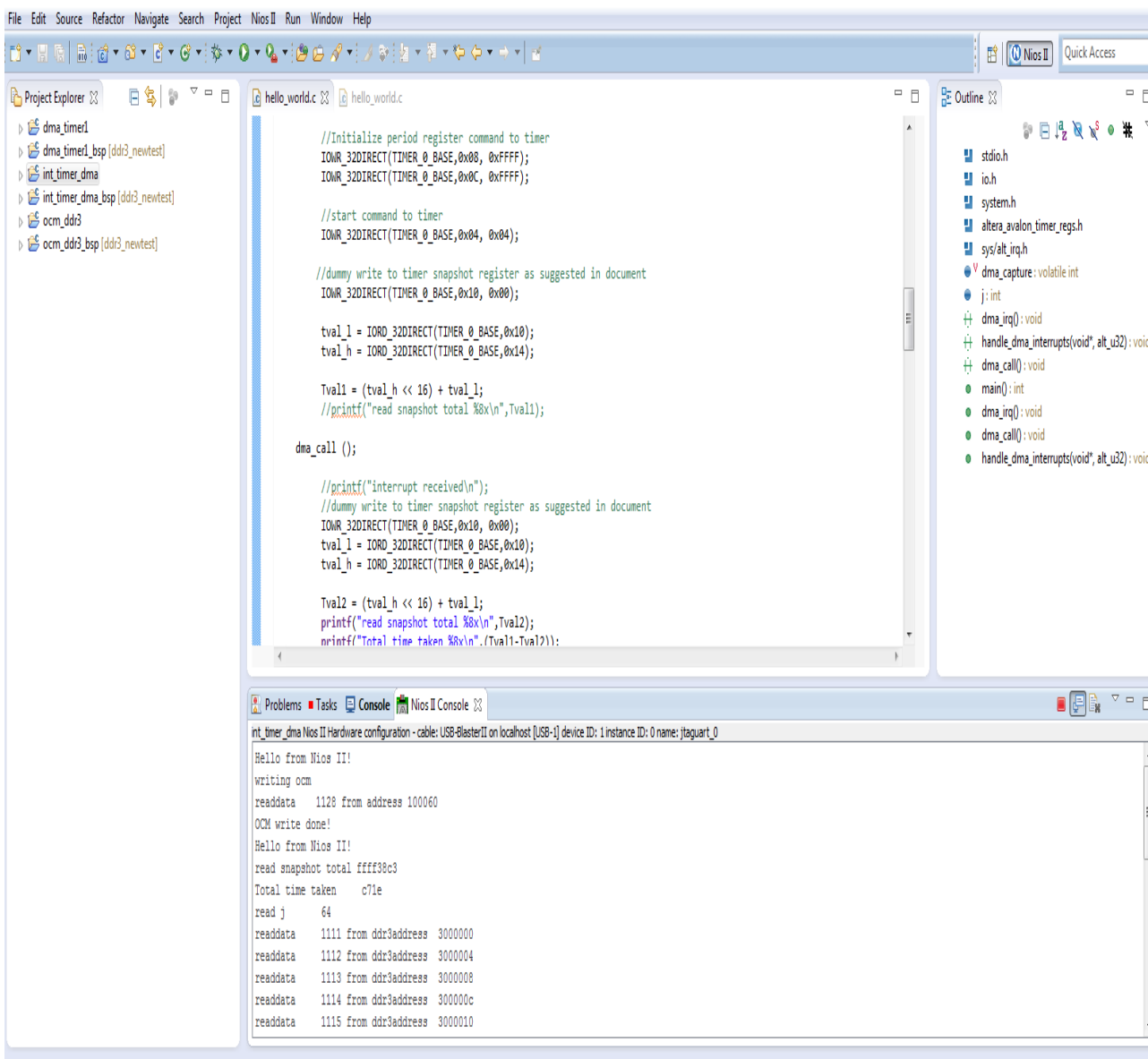
5. During or after the transaction, the CPU can determine if a transaction is in progress, or if the transaction ended (and how) by examining the DMA controller's status register. Writing the status register clears the DONE bit and acknowledges the IRQ. We can also read the status register to cross-check our transaction.

The transaction can be burst or the normal mode. In burst transfer mode, the DMA controller performs the burst transactions on its master read and write mode. Maximum burst transaction of DMA controller is 1024 words. We are reading the timer snapshot before and after the transfer.

### 3. CONCLUSIONS

We can find out the transfer time taken by DMA controller using a timer. Every transferring of data takes a different period of time because of the delay in the processing. We can calculate an average time of the transfer. We have calculated time of the transfer in two different modes, the burst mode and the normal mode.

Although, the time calculated in two different modes is approximately same. The figure 6 and figure 7 shows the DMA transfer time in two different modes.



```
File Edit Source Refactor Navigate Search Project NiosII Run Window Help
Project Explorer
dma_timer1
dma_timer1_bsp [ddr3_newtest]
int_timer_dma
int_timer_dma_bsp [ddr3_newtest]
ocm_ddr3
ocm_ddr3_bsp [ddr3_newtest]
hello_world.c
//Initialize period register command to timer
IOWR_32DIRECT(TIMER_0_BASE,0x08, 0xFFFF);
IOWR_32DIRECT(TIMER_0_BASE,0x0C, 0xFFFF);

//start command to timer
IOWR_32DIRECT(TIMER_0_BASE,0x04, 0x04);

//dummy write to timer snapshot register as suggested in document
IOWR_32DIRECT(TIMER_0_BASE,0x10, 0x00);

tval_l = IORD_32DIRECT(TIMER_0_BASE,0x10);
tval_h = IORD_32DIRECT(TIMER_0_BASE,0x14);

Tval1 = (tval_h << 16) + tval_l;
//printf("read snapshot total %8x\n",Tval1);

dma_call ();

//printf("interrupt received\n");
//dummy write to timer snapshot register as suggested in document
IOWR_32DIRECT(TIMER_0_BASE,0x10, 0x00);
tval_l = IORD_32DIRECT(TIMER_0_BASE,0x10);
tval_h = IORD_32DIRECT(TIMER_0_BASE,0x14);

Tval2 = (tval_h << 16) + tval_l;
printf("read snapshot total %8x\n",Tval2);
printf("Total time taken %8x\n",(Tval1-Tval2));

Nios II Console
int_timer_dma Nios II Hardware configuration - cable: USB-BlasterII on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0
Hello from Nios II!
writing ocm
readdata 1128 from address 100060
OCM write done!
Hello from Nios II!
read snapshot total ffff38c3
Total time taken c71e
read j 64
readdata 1111 from ddr3address 3000000
readdata 1112 from ddr3address 3000004
readdata 1113 from ddr3address 3000008
readdata 1114 from ddr3address 300000c
readdata 1115 from ddr3address 3000010
```

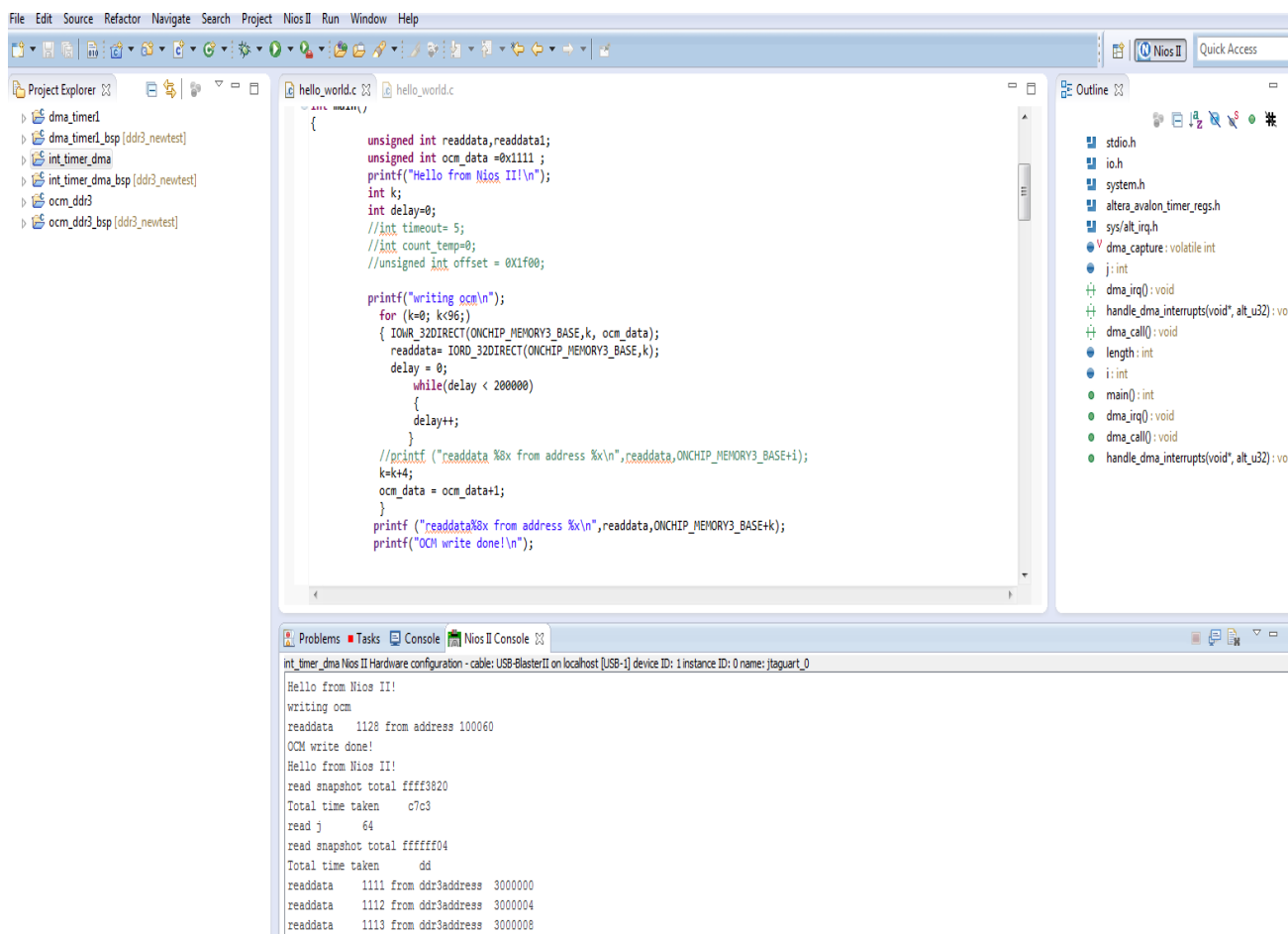
**Fig -6:** DMA transfer in burst mode.

The estimated theoretical time calculated is 508 us, and we can see our result is nearly to this value. We have calculated the time taken by the commands in one DMA transfer that is equal to 2.20 us in burst mode. By that we can calculate the

transfer time taken in 100 times running of DMA transfer commands as the DMA initialization commands is also taking some unit of time in processing. And the rest of the time is taken by the DMA controller in the actual DMA transfer between the On-chip memory and DDR3 SDRAM. We can calculate the throughput of our transfer process.

**Table 1:** Controller transfer timings in burst mode.

Average time in DMA transfer	Avg. time taken by the commands of DMA controller for 100times initialization	Throughput obtained
510.12 us	220 us	196 Mbytes/sec



**Fig -7:** DMA transfer in normal mode.

**Table 2:** Controller transfer timings in normal mode.

Average time in DMA transfer	Avg. time taken by the commands of DMA controller for 100times initialization	Throughput obtained
511.17 us	227 us	195 Mbytes/sec



To transfer the high speed data, we need data buffering in order to achieve the high data throughput. This work results the transfer (read/write) of the high speed data at 400Mhz clock on DDR3 SDRAM. The transfer speed is very fast because the DMA controller doesn't occupy the CPU in the processing. The average time of DMA controller in 100 times transferring of 1 Kbyte of data is 510 us. The throughput obtained by transferring 100Kbytes of data is 196 Mbytes/sec. This data is further transferred over communication channel using PCI express.[3]

#### 4. SUMMARY AND FUTURE SCOPE

In the present work we are transferring our data on a clock of 400Mhz on the DDR3 SDRAM. We can transfer this data on higher clock frequency. Throughput can be increased by using efficient memory buffering technique. This work is proposed to be used in data acquisition system for the accelerator which would collect data from the detectors which will be sending over the high speed interfaces as PCI express. This analog data from detector is digitized using ADC, which generate output data of 2Gbps.

So, the next target of this work is to transmit the data at high speed (2Gbps or more) from the DDR3 SDRAM depending on the trigger condition to the PCI Express. This will reduce the hardware requirement and increase the throughput and reliability of the system.

#### REFERENCES

- [1] Reference Manual: Altera Cyclone V GT FPGA Development Board, Jan 2015.
- [2] Internetworking Technology Overview, University of St. Andrews, June 1999
- [3] User Guide: Triple-Speed Ethernet MegaCore Function, Corporation.
- [4] Quartus Sytem Design Handbook: Corporation, Jun 2014.
- [5] Tom Shanley, Ravi Budruk, and Don Anderson, "PCI Express System Architecture," Mindshare, Addison-Wesley publishing company, 2003.
- [6] Altera Reference Manual: External memory interface handbook, corporation, Sept 2016.
- [7] Bhasker, J., A VHDL Synthesis Primer, Allentown, PA : Star Galaxy Publishing, 1995.
- [8] San Jose, "Avalon Interface Specifications", 101 Innovation Drive, Software Version 13.1,December 2013.