# System Design Document: Hardware Accelerated Market Order Packet Generation

Ankur Gupta [*1], Dhananjay Palshikar [†1], Mithila Paryekar [‡1], Sushant Bhardwaj [§1], and Yasser Mohammed [¶1]

[1]Department of Electrical Engineering,Columbia University

March 20, 2012

## Abstract

This document specifies the design of Hardware Accelerated Market Order Packet Generator. The designed system aims at accelerating the release of packets on the network. Optimization is acheived in terms of reducing the latency, decreasing the data uploaded on the Avalon bus which will eventually lead to power optimization. A software application running on a soft-processor would change the transaction-data going over the network in runtime. This document describes the overall architecture of the system, along with describing the design of the custom ethernet accelerator. Typically the ethernet controller should be capable of receiving and sending data over the network. Our implementation will accelerate the **sending** of data to the network. The receiving of data will still be handled in software.

***Keywords.*** *FPGA, Nios II,ethernet accelerator*

## 1 Introduction

Various applications now days require the fast transmission of data in the order of giagbytes per second. High frequency trading used in stock markets is one such example. A major bottleneck in any such system is the overhead of the Operating System due to the limitation of CPU speed.This occurs since packetization is implemented using software application. The speed of these systems can be greatly increased if packetization is done in hardware instead.

The duration of time we plan to minimize is: from the point when the user initiates the transaction through a software application, to the point when the ethernet controller puts the data on the network.

Our system would implement the transport layer processing along with the software interface of DM9000A ethernet controller on an FPGA. The FPGA will also have on board a soft-processor which would serve as an end point and a source of input for our custom hardware component. The system will be developed for the Altera DE-2 FPGA board.

[*]ag3187@columbia.edu
[†]dp2575@columbia.edu
[‡]mmp2178@columbia.edu
[§]sb3322@columbia.edu
[¶]ym2364@columbia.edu

## 2 Architecture

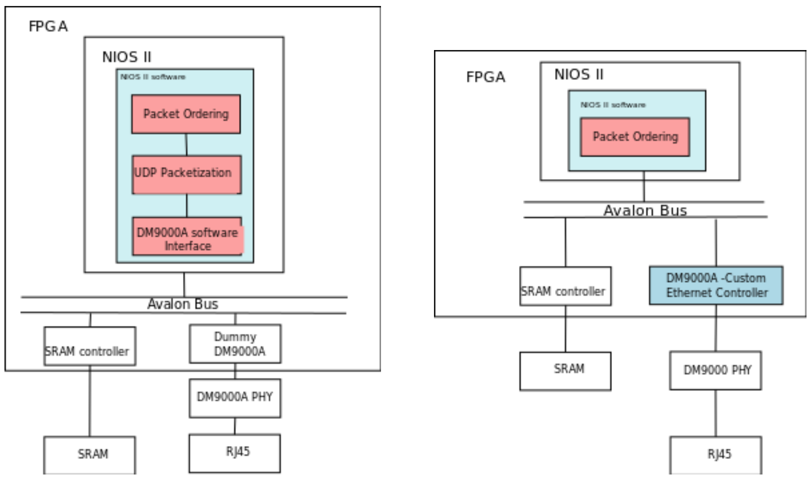
In the Lab 2 assignment of the Embedded Systems Course, we implemented UDP-chat application which implemented the UDP packetization in software. The architecture of the Lab 2 assignment is shown in $Fig.1(a)$, and is further described in the following section. The system we would implement has been shown in $Fig.1(b)$. The accelerated ethernet controller would listen on the avalon bus, for data from the application.

### 2.1 Lab 2 Architecture

In the Lab 2 assignment the software on the NIOS-2 generated data in the form of UDP packets. It was then sent to the ethernet controller via the DM9000A software interface, as can be seen from $Fig.1(a)$. Every time there was data to be sent, the entire packet was sent to the ethernet controller. In terms of a stock market, an architecturally similar system will have the application running on an Operating System, which adds its overheads. The software application will have to compete with other processes in order to get scheduled. Although Lab 2 does not have an Operating System on the NIOS-II processor, its architecture is similar in the sense that UDP protocol is handled in the software. Lab 2 provided us a good starting point for the design of our project.

### 2.2 System Architecture

Data packets sent over the network follow a protocol similar to the OUCH protocol.OUCH is a digital communications protocol that allows customers of the NASDAQ (National Association of Securities Dealers Automated Quotations) to conduct business in the options market adapted for streaming(FAST) [1], where often the difference in contents of the UDP packet are restricted to a few fields in the payload, (assuming UDP header remains the same;fixed sender, fixed recipient). Thus, the System would implement the UDP packet generation on the FPGA, in effect replacing the NIOS II software block in $Fig.1(a)$, with the DM9000A block in $Fig.2(a)$. The software component on NIOS would now only send limited filed information amounting to 12 bytes(see $Table1$).



(a) Lab 2 (b) Our System

Figure 1: System Architecture

| Price(5) | Name(4) | Buy/Sell(1) | Quantity(2) |
|---|---|---|---|

Table 1: UDP-Payload Format

| Offset | Wait(Y/N) | Data |
|---|---|---|

Table 2: Instruction Set for DM9000A

## 2.3 DM9000A accelerated block for UDP

Our task for the project would be to build the DM9000A block as an Avalon peripheral on the Altera-II FPGA. The implemented peripheral will store a generic pre-decided UDP packet in its internal memory. Every time a request on the Avalon Bus is made by the NIOS-II, the packet payload contents are minimally modified and passed over to the DM9000A PHY interface. In all, the DM9000A accelerated block would have the following components :

- Avalon Peripheral Component, for reading from the avalon bus.
- UDP Packetization Component, for modifying the contents of UDP packets
- DM9000A PHY interface would further have two components
  - DM9000A initialization component, to initialize the ethernet Physical Layer Component for the first time a packet is sent.
  - DM9000A Communication component: sends UDP packets to the physical layer.

This component would be made in VHDL which would be synthesized and deployed on the FPGA chip on-board the Altera DE-2 evaluation Board. The next section shows the format of UDP payload used for the project.

## 2.4 UDP-Payload Format

The numbers seen with the field names indicate the size of each field in bytes. The format shown above could be revised along the course of the project.

## 2.5 Instruction Set for DM9000A

In order to minimize communication over the avalon bus, we intend to design a protocol which could eliminate the need for the DM9000A controller to wait for each field, in case only a subset of fields have changed. Table 2 ,shows the format of instruction which would be sent over the avalon bus. The $Offset$ field in the instruction decides which address has to be written to. Within the DM9000A controller this would be interpreted as the payload field to be modified. $Wait$ tells the controller to wait for more fields and not send the packet right away. $Data$ carries the content of the modified field.

## 2.6 DM9000A controller State Description

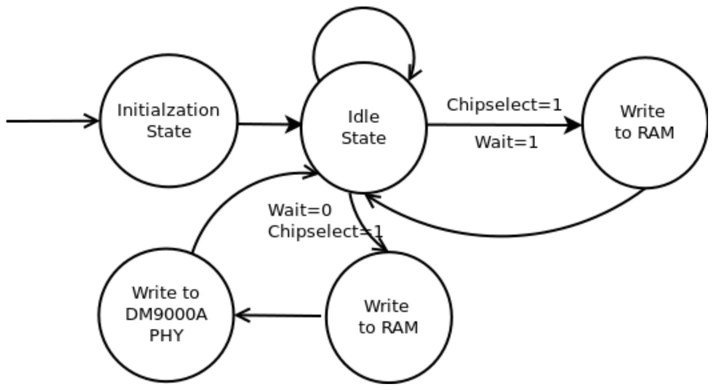


Figure 2: State Diagram for DM9000A controller

The ethernet controller would be modeled on the the state digram show in $Fig.$2. The figure is self explanatory and follows the protocol explained in $section$2.5. An important point to notice is, that the controller writes to PHY only when $wait = 0$ ,implying that there are no more changes to be made, and the packet could be sent in its current form.

## 3 Milestones & Plan of execution

The following list shows our Milestones, some of which have been accomplished.

- Design of the system **- Complete**
- UDP-test **-Complete** : This test verified if the UDP packets sent through the software code on NIOS-II were readable. This was done by receiving the packets on a UDP server running on an X86 machine.
- Milestone -I Get the DM9000A module to send a fixed packet from its internal memory to the network. At this point, the initialization of DM9000A will be handeled in software. Develop software interface to update data value at this internal memory.
- Milestone -II DM9000A is now an Avalon peripheral, it would listen to the bus, and change the contents of the packet as per the instruction from software application , and then send the packet over the network. A dummy version of the protocol mentioned in $section2.5$ would be implemented.
- Milestone -III Implement the complete protocol mentioned in $section2.5$ which leads to bus optimizaion. Calculate and trasmit correct checksum value. Also, implement the initialization of DM9000A in hardware (which was up till now in software). The system will be able to handle reception of data packets in software.

## 4 Scope & Limitation of the implementation

The performance gain acheived with this project cannot be compared in performance to the commercial systems deployed at trading firms, primarily due to the limitations imposed by the Altera DE2 Board. The limitations are listed below:

- *Slow Clock:* Altera DE2 Evaluation Board having lower clock frequency
- *Slow Ethernet:* The absence of Gigabyte ethernet on board Altera DE-2 Board.
- *Avalon Bus vs Shared Memory:* A bus handles multiple devices, snooping on the bus for reading and writing is slower than accessing a shared memory.

## References

[1] What is ouch. URL `http://www.nasdaqtrader.com/Trader.aspx?id=ouch`.