Columbia University, Spring 2012
CSEE 4840: Embedded System
Project Design

Hardware Acceleration of Market Order Decoding

Adil Sadik Amandeep Chhabra Manu Dhundi Prabhat Godse

Abstract

"Hardware Accelerated Packet Decoding of Market Equity Orders" is a dedicated hardware/software system to accelerate the decoding of UDP packets containing market equity orders received (over a network) by brokers or exchanges. It is implemented using Field Programmable Gate Array (FPGA). The format of data received is predefined by design team which is very close to NASDAQ data format, the UDP version of OUCH protocol. The software application handles the processing of received data. The main goal of this project is to reduce the software latency of receiving frames with UDP payload.

Introduction

In high frequency stock trading, there is a huge demand of low latency systems for reading data from and writing data into the networks. Generally the read/write operations are handled in software which limits its speed. Recently, there has been an effort in shifting most of the software operation into hardware. Field Programmable Gate Arrays (FPGAs) provides ability of creating reconfigurable hardware along with flexibility of software interactions with the hardware. In our project we plan to build an embedded system for accelerating the decoding of orders received by the exchange.

The synthesizable hardware reads data from the network (Ethernet) at very high speeds, close to the clock frequency. It receives data (market equity orders or any other data) from Ethernet connection, handles the received data in a custom hardware, checks if it is a market equity order data and processes it if so, and further passes it to the software. Software could further process the data read. The hardware can be used as a TCP offload engine in high speed network interfaces.

In this document, a design is presented for the same

Design

Hardware Description

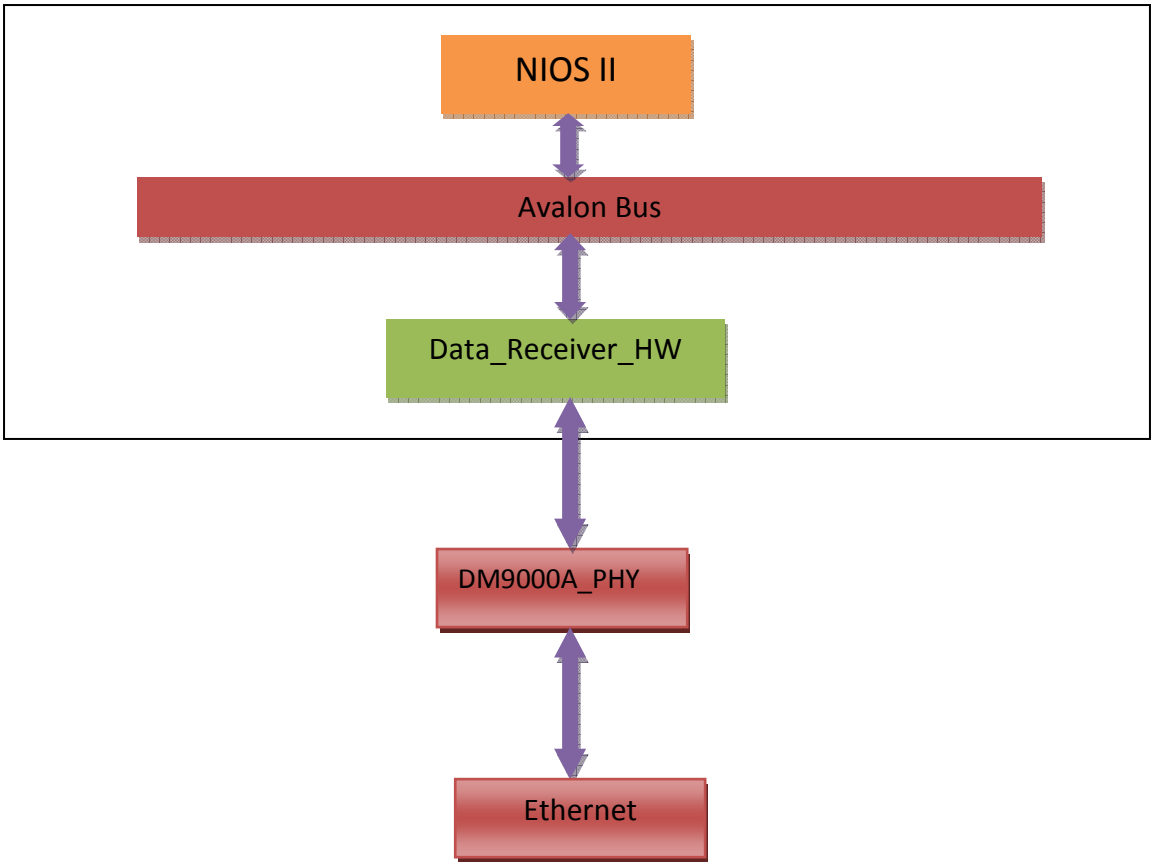


Figure 1: Block Diagram

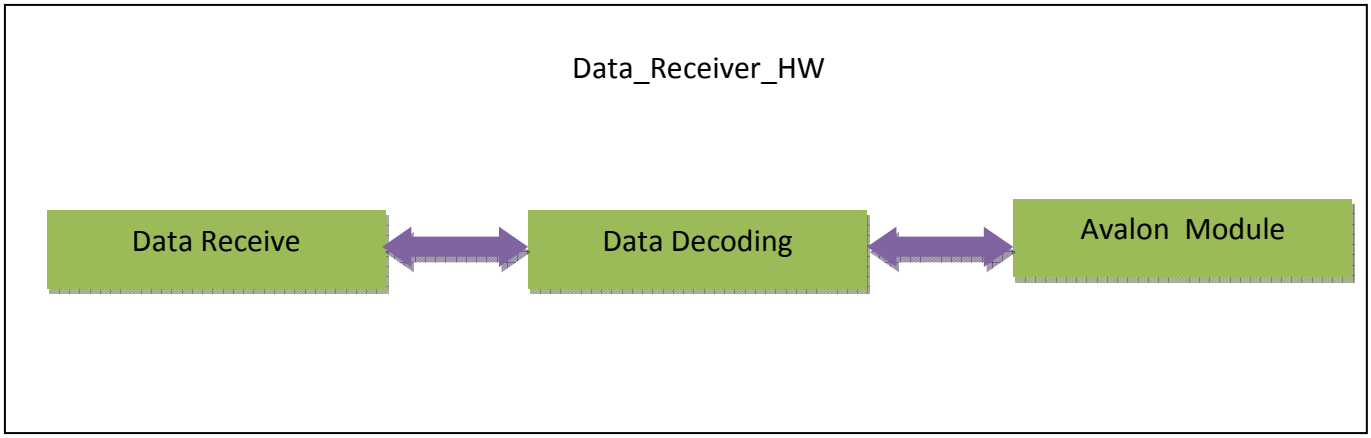


Figure 2: Data_Receiver_HW

DM9000A_PHY

DM9000A_PHY receives data from Ethernet. It is controlled by the custom hardware (Data_Receiver_HW) on FPGA.

Data_Receiver_HW

Data_Receiver_HW is a custom hardware peripheral on FPGA. It has 3 main modules: (i)Data Receive (ii)Data Decode (iii)Avalon Module. It receives data from the DM9000A_PHY and stores it in an internal local memory on the FPGA system and fires an interrupt to the NIOS application. The Data_Receiver_HW will be developed using VHDL. This is similar to DM9000A controller software, which is slow and has high latency (when compared to the latency of a hardware system). Data_Receiver_HW will speed up the receive operation to the order of the clock speed of hardware system. The 3 modules are explained below.

(i) Data Receive:

This module, on the interrupt from the DM9000A PHY, reads the data from the PHY buffer. It then checks whether the data received is the market equity order data. This is done by checking the destination UDP port of the data received. It forwards the received data to Data Decoder module in case it is market equity order . It then fires corresponding interrupt to the application. That is, we have 1 interrupt in case the received data is market equity order and another interrupt for normal data.

(ii) Data Decode:

This module computes the checksum of the data received, strips the headers of the market equity order which is in the form of the UDP version of OUCH protocol. Copies the required fields of market equity order in internal local memory on the FPGA system.

(iii) Avalon Module:

NIOS application interacts with this module to read the received data from the internal local memory on the FPGA system.

The features are described in the state diagrams below.

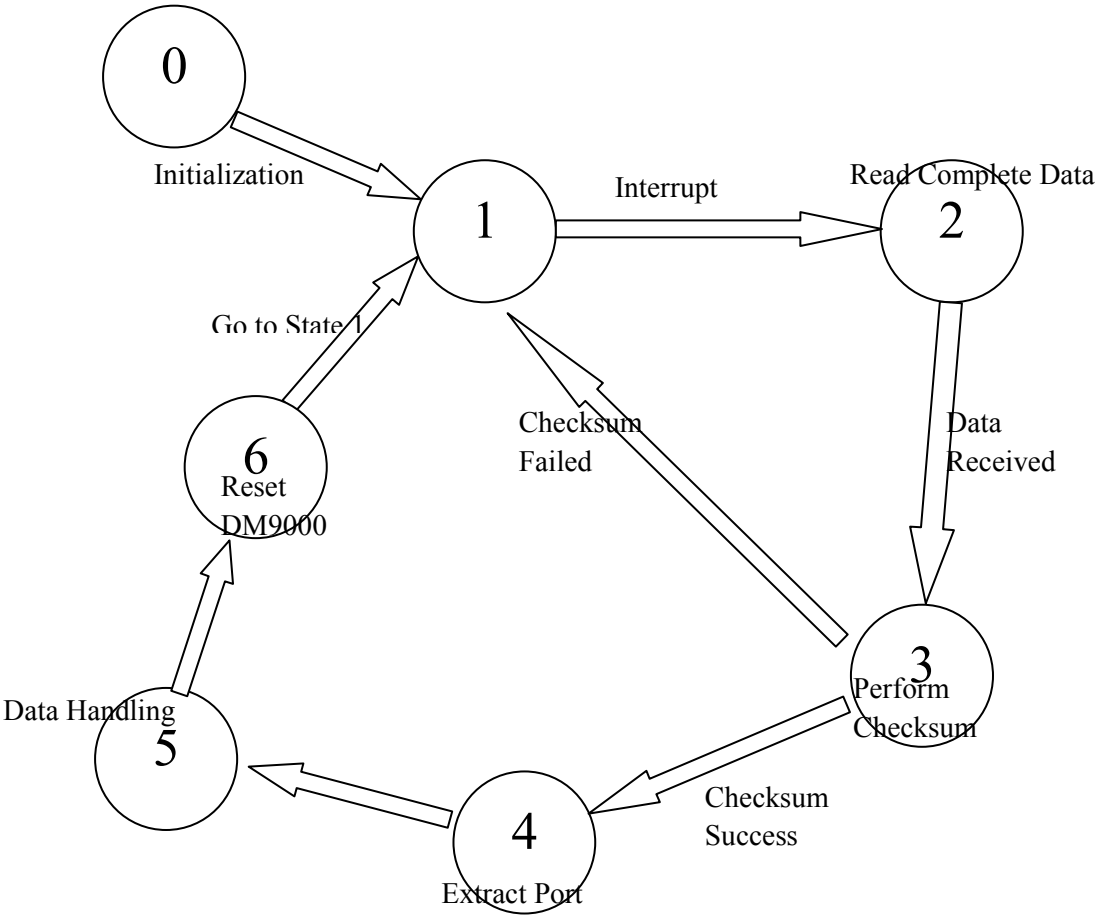


Figure 3: Flow Diagram 1

After the initialization at state 0, the Data Receive module moves to state 1 and waits for new data to arrive at DM9000A_PHY buffer. When data arrives, it moves to state 2 where it reads the complete data from the buffer of DM9000A_PHY. Then in state 3, checksum is performed and compared with the checksum in the data to ensure the validity of data received. In case of checksum fail, it drops the packet and goes back to state 1 continue to wait for new data. If checksum is proper, then in state 4 the data is identified whether it is market equity order data. If so fields then data is handled in state 5 which is explained in the below flow diagram. Then it again goes back to state 1 and the process continues.

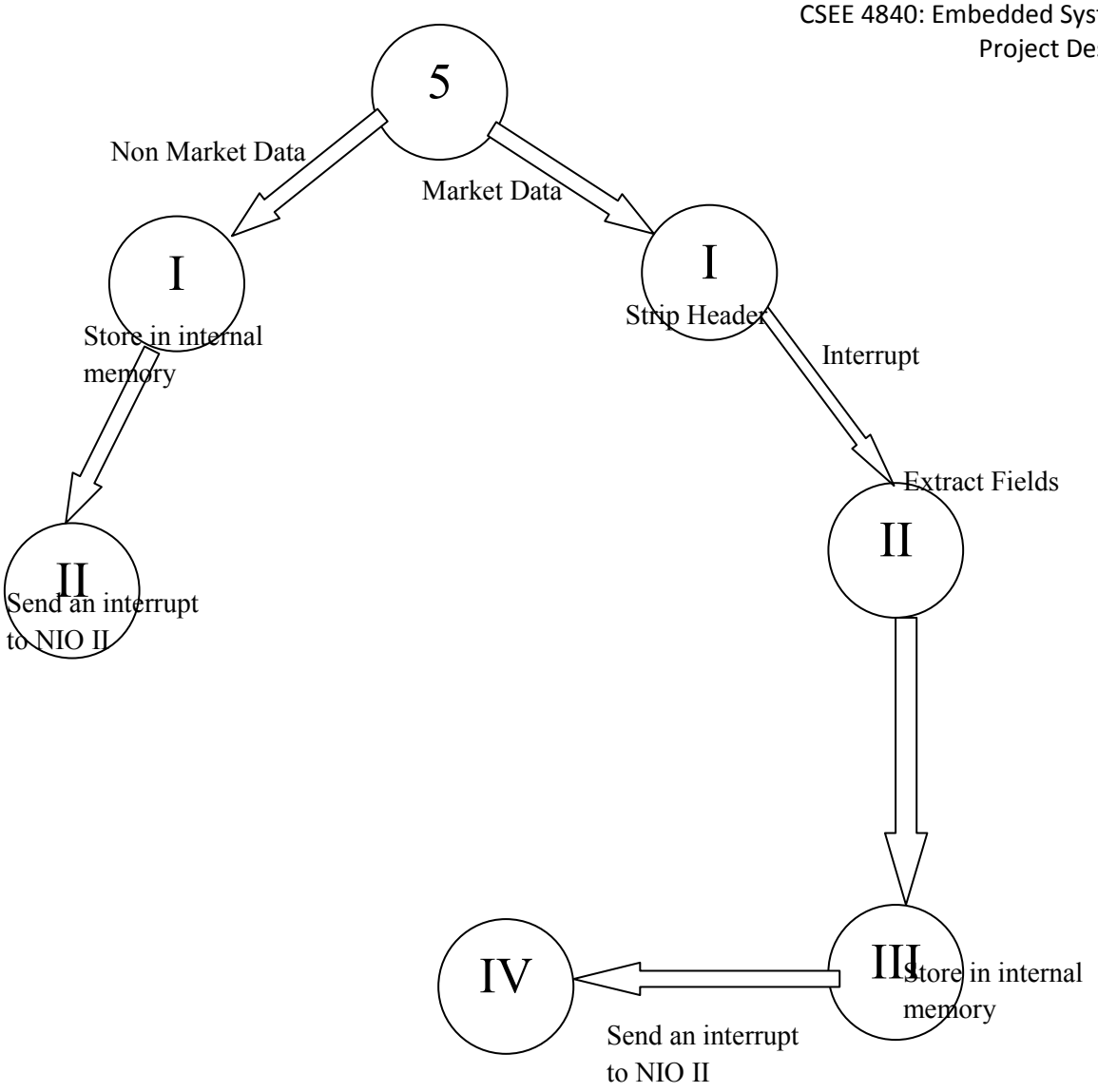


Figure 4: Flow Diagram 2

The state Step 5 determines the packet received is Market Data/Non Market Data. If the packet received is Market Data it proceeds to Stage I, where the headers of the packet are stripped of. Then in Stage II the four fields (Stock, Quantity, Price, Buy/Sell) are identified from the stripped data packet. In Stage III the four data fields are stored in the internal memory and finally Stage IV sends an interrupt to NIOS II.

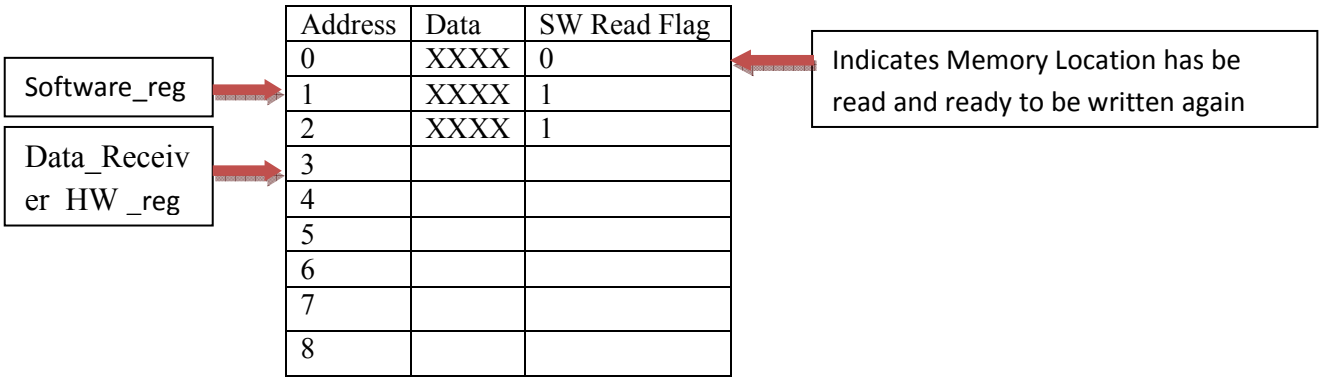
If the packet received is Non Market Data then the data is stored in another internal memory location and interrupt is sent to NIOS II.

Software Description

The software application will read and process data stored in SRAM. For now we plan just to display the data (stock trading order) read. Software could further process the that is read.

Pseudo Code

```
for(;;)
{
        check_if(SW Read Flag){ perform data operation;}
}
```



Milestones

1) March 27:

- Create a custom hardware block to:
  - Receive interrupt,
  - Read the packet in hardware into internal buffer,
  - Fire interrupt and allow Nios II to read in the full packet data from the internal buffer within the accelerated decoder block.

2) April 10:

- Implement accelerated packet decoding of orders packet in hardware, and extract out the necessary fields, and allow NIOS to read those four fields.

3) April 24:

- Displaying of orders, software for handling received orders, and generic optimizations (checksum offloading).

4) May 10:

Final presentation and report.

Future Scope

Do some actual (as done in High Frequency Trading industry) processing on the received stock trading data. Develop/merge with High Frequency Trading Data sender and have one system that can send and receive at very high speed and small latency.