CSEE 4840
EMBEDDED SYSTEM DESIGN
Final Report, Spring 2012

Hardware Acceleration of Market Order Decoding

Amandeep Chhabra
Manu Dhundi
Prabhat Godse
Adil Sadik

LIST OF FIGURES

CSEE 4840 FINAL REPORT

Hardware Acceleration of Market Order Decoding

Department of Computer Science

Spring 2012

Abstract

High Frequency Trading (HFT) has recently received a lot of attention and became an important part of financial markets where low latency trading plays a significant role. This project presents a hybrid hardware/software system to accelerate the decoding of UDP packets containing the market equity orders received (over a network) by brokers or exchanges. It is implemented by using Field Programmable Gate Array (FPGA). The format of received data is predefined by design team which is very close to NASDAQ data format, the UDP version of OUCH protocol. In our design, the software application handles the processing of received data while UDP offloading and decoding of received market data packet is done through customized logic block within FPGA. We have reduced the overall software latency of receiving market orders through UDP payload by implementing market data decoding and reception in hardware.

1. Introduction

1.1 High Frequency Trading and FPGA: A Brief Overview

High frequency trading (HFT) refers to the set of techniques within electronic trading of sockets and derivatives, where a large number of orders are injected into the market at sub-millisecond round-trip execution times [1]. In electronic trading of stocks, orders are send in electronic form to a stock exchanges. The stock exchanges match the bid and ask orders to execute a trade. Information about outstanding orders are distributed or made visible to market percipients through feeds. The feeds carry pricing information and are multicasted to the market participants through standardized protocols [2]. In most cases stock information is transmitted in the form of UDP

packet over Ethernet. A fast and efficient method of decoding of the UDP data stream is mandatory. Moreover the latency to make a decision on received order and transmit the information about outstanding orders should be as low as possible. In this scenario, FPGA has caught the eye of high frequency traders because of its potential to offer reduced round-trip latency of market equity orders.

1.2 Contribution of this Project

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. Through this project, we developed a reconfigurable hybrid hardware-software system where the latency to receive and decode market data packet is significantly reduced by implementing hardware accelerator in FPGA. 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 without software intervention, checks if it is a market equity order data and processes it if so, and then passes it to the software. We implemented the software driver for DM9000 Ethernet controller in VHDL and thus obviated the need of high latency operation of reading a data packet for PHY through software. Moreover the checksum and decision making process (selection between market order and garbage packet) is also implemented in hardware which runs in clock speed.

2. Design

2.1Protocol Stack

The protocol layers associated with each transaction is presented below:

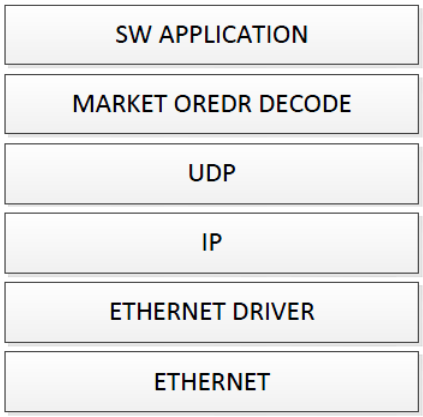


Figure 1 Layers

Ethernet Layer: It is the lowest layer of the protocol stack. It is responsible to send and receive packets through the network. It defines required data encoding mechanism like CRC to ensure the correctness or data integrity. It also identifies access points or nodes through Media Access Control (MAC) address associated with the received packet.

IP Layer: IP defines as Internet Protocol and seats just above the ETHERNET layer. IP is used to multicast messages to different nodes. It basically groups a group of computers and assign unique address to each one of them. Therefore it is used in most exchanges as feeds are required to be multicasted as explained in the *overview of HFT* section.

UDP Layer: UPD refers to User Datagram Protocol. Application layer use UPD as a mean to send messages to the nodes. It ensures data integrity through checksum. In this design, the checksum is computed in hardware through the checksum control register provided by DM9000 Ethernet controller. In case of bad checksum, the DM9000 PHY discards the packet and doesn't generate any interrupt.

Market Order Decoding: This layer strips off the header of received UDP packet and decodes the market order. There are four fields in the marker order defined by NASDAQ- Stock, Quantity, Price and Bid/Offer. This data is passed to application layer which maintain a logbook of all the matched and outstanding market orders.

Application Layer: Application layer does book-keeping and applies software and algorithmic techniques to match buy-sell orders or transmit feeds about outstanding market orders. Detailed analysis of various algorithms and statistical models associated with this layer is beyond the scope of this project.

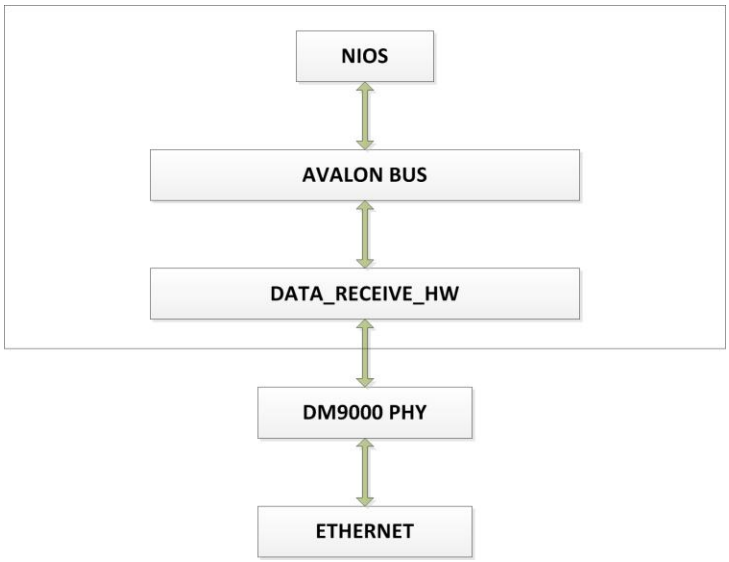


Figure 2 High Level Block Diagram

2.2 Custom Hardware

This section describes the design of customized hardware module and it's interaction with the Software. Figure 1 depicts the overall structure of the system.

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 receives data from the DM9000A_PHY and stores it in an internal local memory on the FPGA system and then fires an interrupt to the NIOS application. Its functionality is basically similar to DM9000A driver software, which is slow and has high latency (when compared to the latency of a hardware system). Data_Receiver_HW speeds up the receive operation to the order of the clock speed of hardware system. It implements the following three state machines and controls the interaction with PHY through read and writes.

1. io_write_data 2. io_read_data 3. dm9000_hw

NAME	TYPE	OPRATION
io_write_data	Mealy FSM	Write to PHY- (register address and data)
io_read_data	Mealy FSM	Read from PHY- (Status/Ready flags, received data packet)
dm9000_hw	Moore FSM	Generate Controls for io_write_hw & io_read_hw FSMs

The corresponding state diagrams can be found in the *Documentation of Design Specs* section of this report.

Reception of Data: *dm9000_hw*, on the interrupt from the DM9000A PHY, generate the control-signals to drive *io_read_data* and *io_write_data* state machine. *io_read_data* reads the received packet from PHY buffer and saves it in an internal RAM. It then checks whether the data received is the market equity order data or not. This is done by checking the destination UDP port of the data received.

The data is then forwarded to the Data Decoder module in case it is market equity order . Data Decode module strips the headers of the market equity order which is in the form of the UDP version of OUCH protocol. It also extracts the four fields (Stock, Quantity, Price, Bid/Offer) of market order packet and save them in four different location in internal local memory on the FPGA system. Afterwards the Data_Receive_HW module fires an interrupt to the application.

A flow diagram is presented bellow which highlights the above sequence of operation

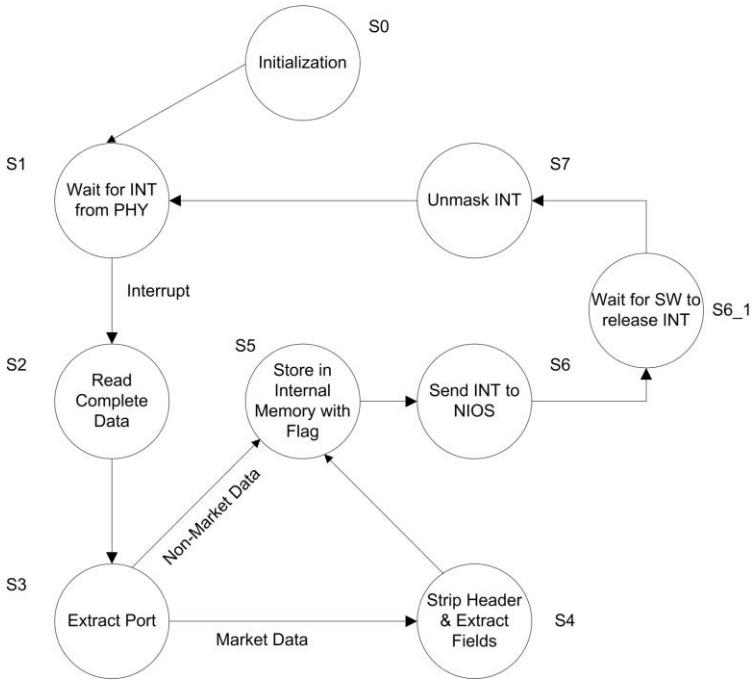


Figure 3: State Diagram representing sequence of operation of HW block

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. Arrival of data will be indicated by an *interrupt* from PHY. So, the interrupt from PHY triggers it to move to *state 2* where it reads the complete data from the buffer of DM9000A_PHY. *State 3* determines the packet received is Market Data/Non Market Data. If the packet received is Market Data it proceeds to *state 4*, where the headers of the packet are stripped off and the four fields (Stock, Quantity, Price, Bid/Offer) are extracted from the packet. In *state 5* the four data fields are stored in the internal memory with a *flag* for "Market Data." If the received packet is not market data then *state 3* make a transition to *state 5* where the received data packet is stored in an internal RAM with corresponding flag. The flag is set to '1' when market data is received. Otherwise it is set to '0'. The software uses this flag to differentiate between market data and normal data. Afterwards, an interrupt is sent to the NIOS in *state 6* and the customized hardware block reset DM9000 and goes back to *state 1* to wait for a new packet.

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

2.3 Software

The software workflow is presented in figure 2(b). After initializing DM9000, SW sets the custom hardware block in receive mode and then waits for a HW interrupt which indicates that custom hardware block has successfully received a packet from DM9000 PHY. State S3 represents this state. After getting the interrupt from hardware, software reads a flag from a predefined RAM address. This flag is set by hardware block. Flag '0' and '1' respectively represents market order and non-market order as mentioned earlier. In case of market order, the software reads the four fields (Stock, Quantity, Price, Bid/Offer) from the RAM and then save them in two Linked List. *Bid List* is order in ascending order while *Offer List* is ordered as descending order. These two linked list is used to identify if there is a possible match between bid and offer. In case of a match, a deal is made and acknowledgement is sent. On the other hand, if the received data is of non-market type then the software just read the data and displays the length on console.

Figure 4 State D eration of SW application

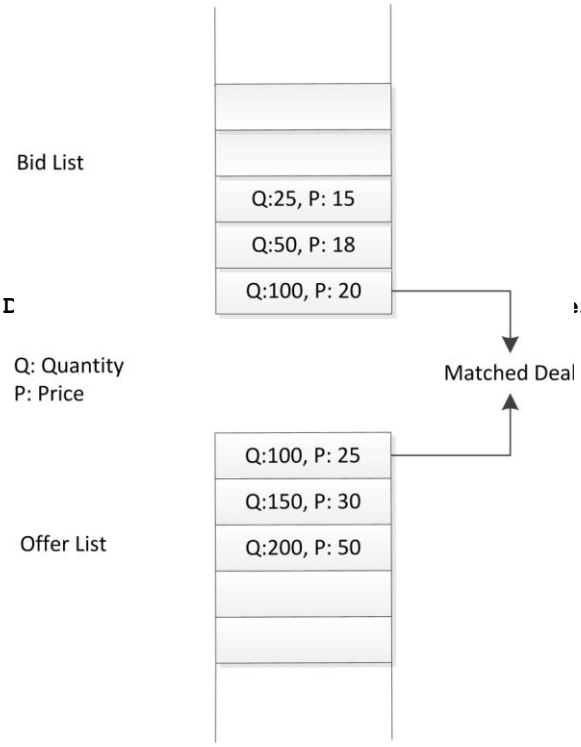


Figure 5 Linked list for Offer and Bid

Figure 2(c) represents the graphical view of two linked list and it also exemplifies a simple Bid-Offer deal. As shown above, Bid List and Offer List data structure saves all the data extracted from market order packets. Head nodes are used to make a decision. After matching a deal, the nodes are deleted from the list.

3. Documentation of Design Specifications

3.1 Detailed Block Diagram

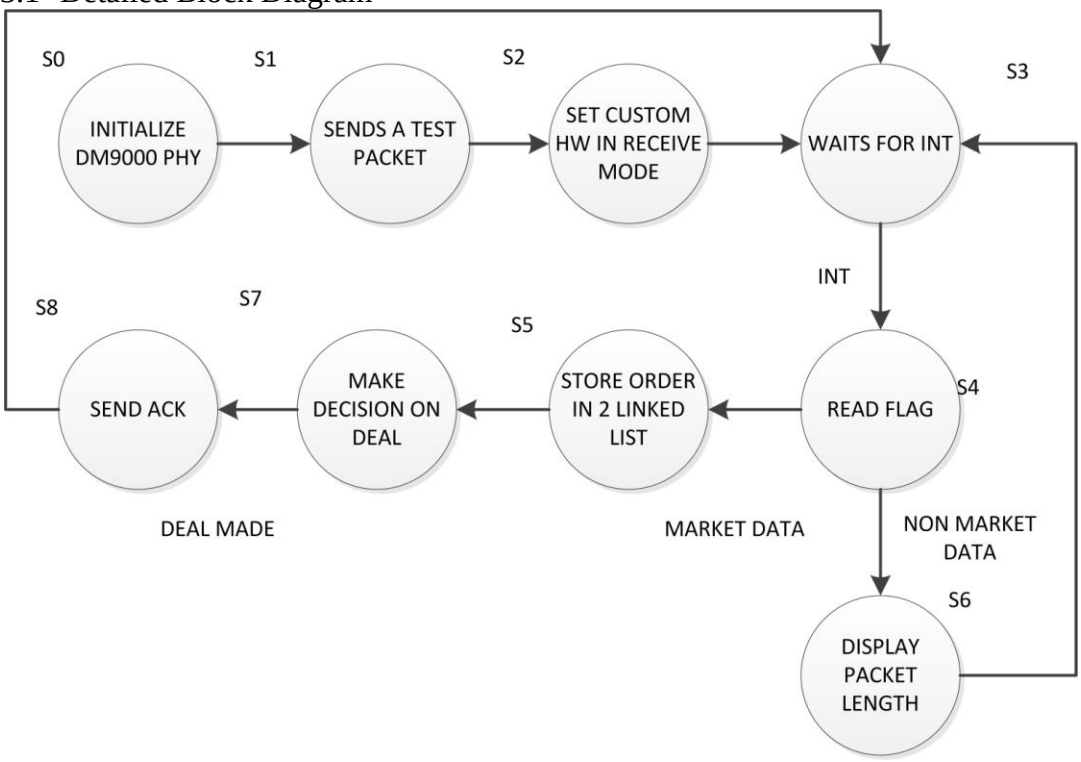


Figure 1 represented a very abstract and high level block diagram of the architecture. Here in **figure 3** we present a more detailed view of the custom logic block (DATA_RECEIVE_HW). Description about control signals and datapath allocation is also presented in detail.

io_read_data block

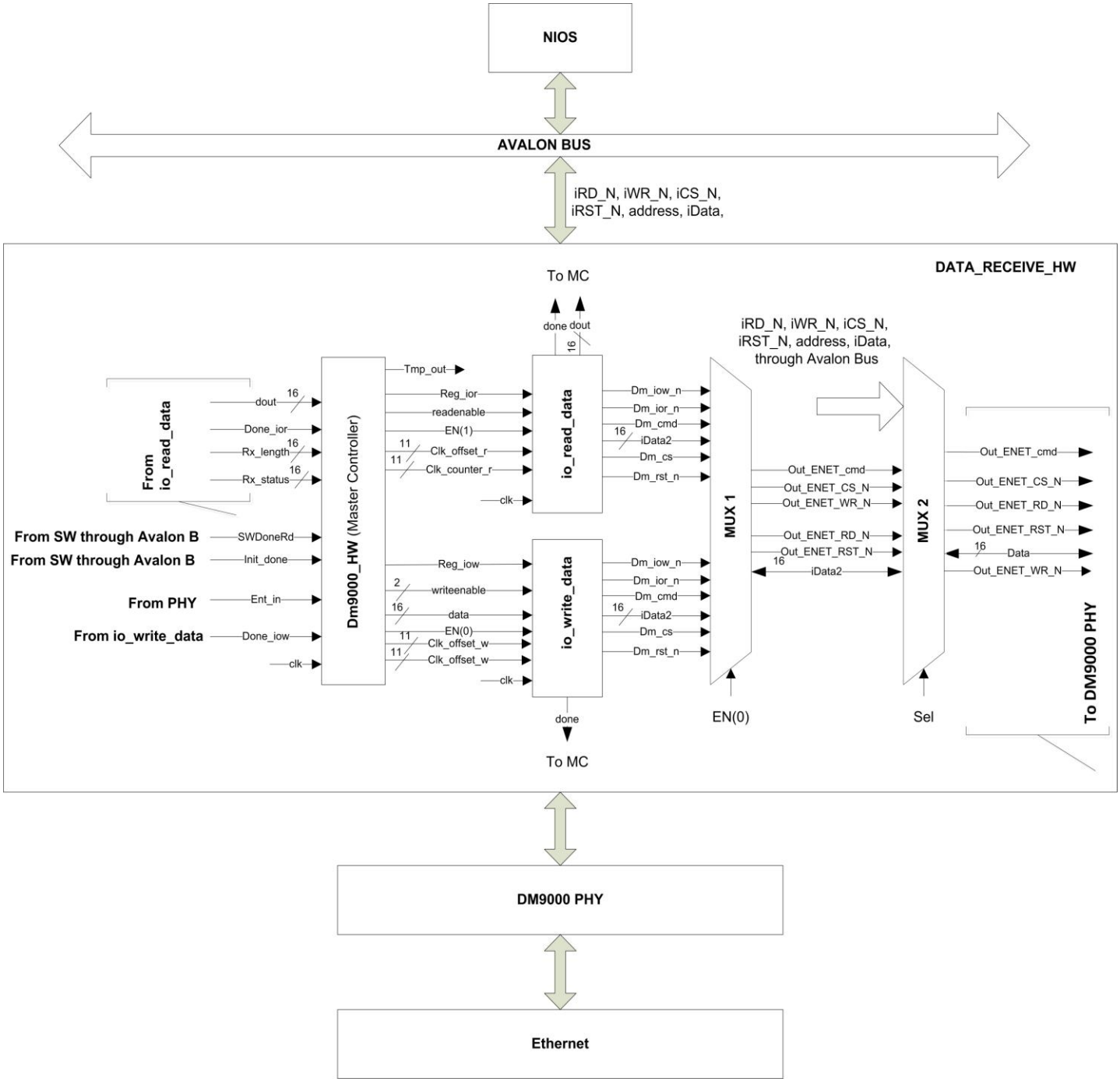


Figure 6 Detailed View of Custom HW block

This is a Mealy type FSM and it implements the *dm9000a_ior(int reg)* and *IORD(dm9000_base, IO_data)* functions in hardware. The register address, data and the required control signal to met timing constraint is supplied by *dm9000_hw.* The set of input, output signals and state diagram can be found in section **3.2**.

io_write_data block
This is a Mealy type FSM and it implements the *dm9000a_iow(int reg, int data)* and *IOWR(dm9000_base, IO_data, 0xXX)* functions in hardware. Like *io_read_data, dm9000_hw* takes the responsibility to keep this state machine to behave in a predefined way which can interact with DM9000a PHY without any glitch. The set of input, output signals and state diagram can be found in section **3.2**.

Dm9000_hw block
This is the main controller built as a Moore type FSM. As mentioned earlier, it generates the control signals to drive *io_read_data* and *io_write_data.*

MUX 1
At a time, only one operation can take place in the PHY- either WRITE to a register or READ from a register. io_read_data and io_write_data generates the required control signal to do these operation. Therefore, it is required to ensure atomicity so that io_write_data and io_read_data can't interact with PHY simultaneously. It is done by incorporating a MUX and the select signal is tied with the Enable signal of io_write_data. It ensures the atomicity among read and write operation.

MUX 2
Initialization of DM9000 controller is done through software. Initialization is done only once at the beginning and hence we didn't found any compelling reason to bring this operation in hardware level. MUX 2 switches controls between HW and SW while accessing PHY. Presumably, it grants SW to write to the PHY only at the beginning during initialization. Control signal for MUX 2 (Sel) is set by SW at the beginning

3.2 State Machines

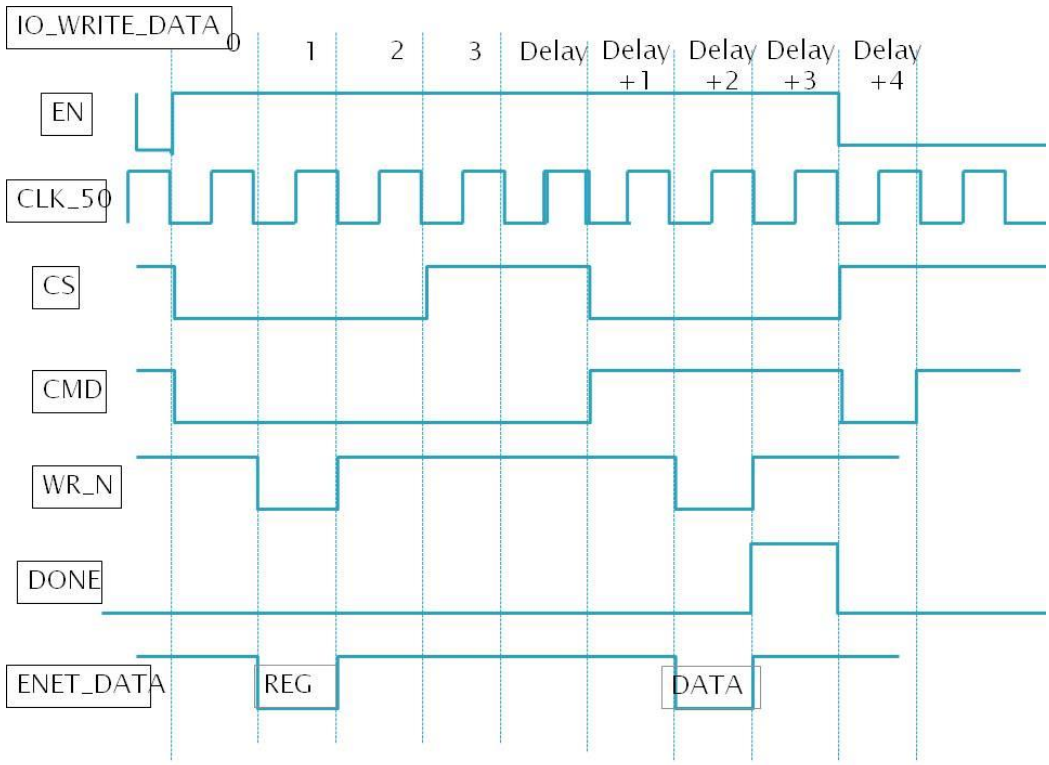
3.2.1 io_write_data

Type: Mealy Machine

reg	IN	8	Register address
data	IN	16	Data
clk_offset	IN	11	Used to meet timing requirement
clk_cnt	IN	11	Used to meet timing requirement

clk	IN	1	system clock
writeonly	IN	2	switch operation between io_addr, io_data
EN	IN	1	Enable the block
dm_iow_n	OUT	1	used to write to a register in PHY
dm_ior_n	OUT	1	used to read from a register of PHY
dm_cmd	OUT	1	Command to differentiate btw Data & Reg add.
done	OUT	1	High when job is done
dm_cs	OUT	1	chip select
cd_rst_n	OUT	1	Reset
iData2	INOUT	16	write and read to PHY

The state machine is shown in the next page.



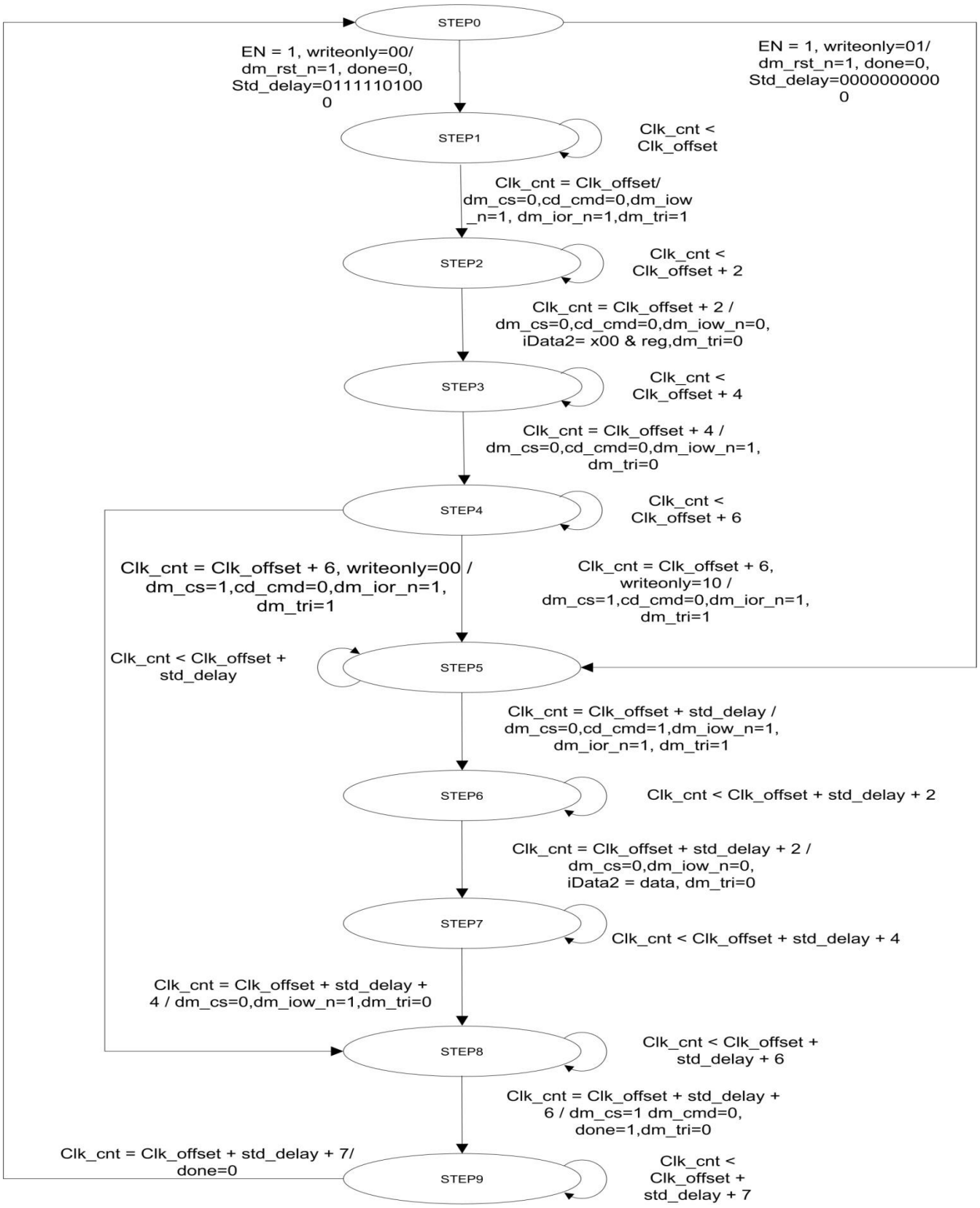


Figure 7 State Diagram of io_write_data

3.2.2 dm9000_hw

Type: Moore Machine

clk	IN	1	system clock
done_ior	IN	1	high when io_read_data finishes reading
done_iow	IN	1	high when io_write_data finishes writing
init_done	IN	1	pull to high by SW after initialization
dout	IN	16	data out
rx_length	IN	16	length of rx packet
rx_status	IN	16	states if rx packet
swdonerd	IN	1	Pulled to high after SW finishes reading
reg_ior	OUT	8	register address sent to io_read_data
reg_iow	OUT	8	register address sent to io_write_data
data	OUT	16	data sent to io_read_data/io_write_data
readenable	OUT	1	control signal to io_read_data
writeenable	OUT	2	control signal to io_write_data
en	OUT	2	enable io_write_data / io_read_data block
tmp_out	OUT	16	output the valid read data from PHY
GP_o	OUT	1	high when the packet is read successfully
clk_offset_r	OUT	11	clock offset for io_read_data
clk_offset_w	OUT	11	clock offset for io_write_data
clk_counter_r	OUT	11	clock counter for io_read_data
clk_counter_w	OUT	11	clock counter for io_write_data

The state diagram is shown in the next page.


S00
INIT_DONE = 1
S0
ENT_IN = 1
S1
Reg_iow=imr, Rx_status= x0000 Data = PAR_set, EN = 01 writeenable = 00
Clk_count_w = Clk_count_w + 1
Done_iow = 0
Done_iow = 1
S2
Reg_ior = MRCMDXC EN = 10, readenable = 0 Clk_count_r = Clk_count_r + 1
Done_ior = 0
Done_ior = 1
S3
EN=10, readenable = 0 Clk_count_r = Clk_count_r + 1
Done_ior = 0
Done_ior = 1
S4
Rx_ready = DOUT and x03
Count_delay >= std_delay
S5
count_delay= count_delay+1
Count_delay < std_delay
Count_delay = std_delay
Rx_ready(0)=1 Rx_ready(1)=0
Rx_ready(1)=1, Rx_ready(0)=0 Or Rx_ready(1)=1, Rx_ready(0)=1
Rx_ready(0)=0
Rx_ready(1)=0
finish
IF0
Reg_iow = MRCMD EN = 01, Writeenable = 10 Clk_count_w = Clk_count_w + 1
Done_iow = 0
Done_iow = 1
IF1
count_delay= count_delay+1
Count_delay < std_delay
Count_delay = std_delay
IF2
EN=10, readenable = 0 Clk_count_r = Clk_count_r + 1
Done_ior = 0
Done_iow = 1
IF3
Rx_status = rx_status_in
IF4
count_delay= count_delay+1
Count_delay < std_delay
ES0
Reg_iow=ncr
data= x0003
writeenable = 00, EN = 01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
Done_iow = 1
ES1
count_delay= count_delay+1
Count_delay < std_delay
Count_delay >= std_delay
ES2
Reg_iow=ncr
data= x0000
writeenable = 00, EN =01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
Done_iow = 1
ES3
Reg_iow=ncr
data= x0003
writeenable = 00, EN = 01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
Done_iow = 1
ES4
count_delay= count_delay+1
Count_delay < std_delay
Count_delay >= std_delay
ES5
Reg_iow=ncr
data= x0000
writeenable = 00, EN =01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
Done_iow = 1
ES6
Reg_iow=ncr
data= ncr_set
writeenable = 00, EN =01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
ES7
Reg_iow=x08
data= bptr_set
writeenable = 00, EN =01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
ES8
Reg_iow=ncr
data= x0003
writeenable = 00, EN =01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
Done_iow = 1
ES9
count_delay= count_delay+1
Count_delay < std_delay
Count_delay >= std_delay
ES10
Reg_iow=ncr
data= x0000
writeenable = 00, EN =01
Clk_count_w =
Clk_count_w + 1
Done_iow = 0
Done_iow = 1

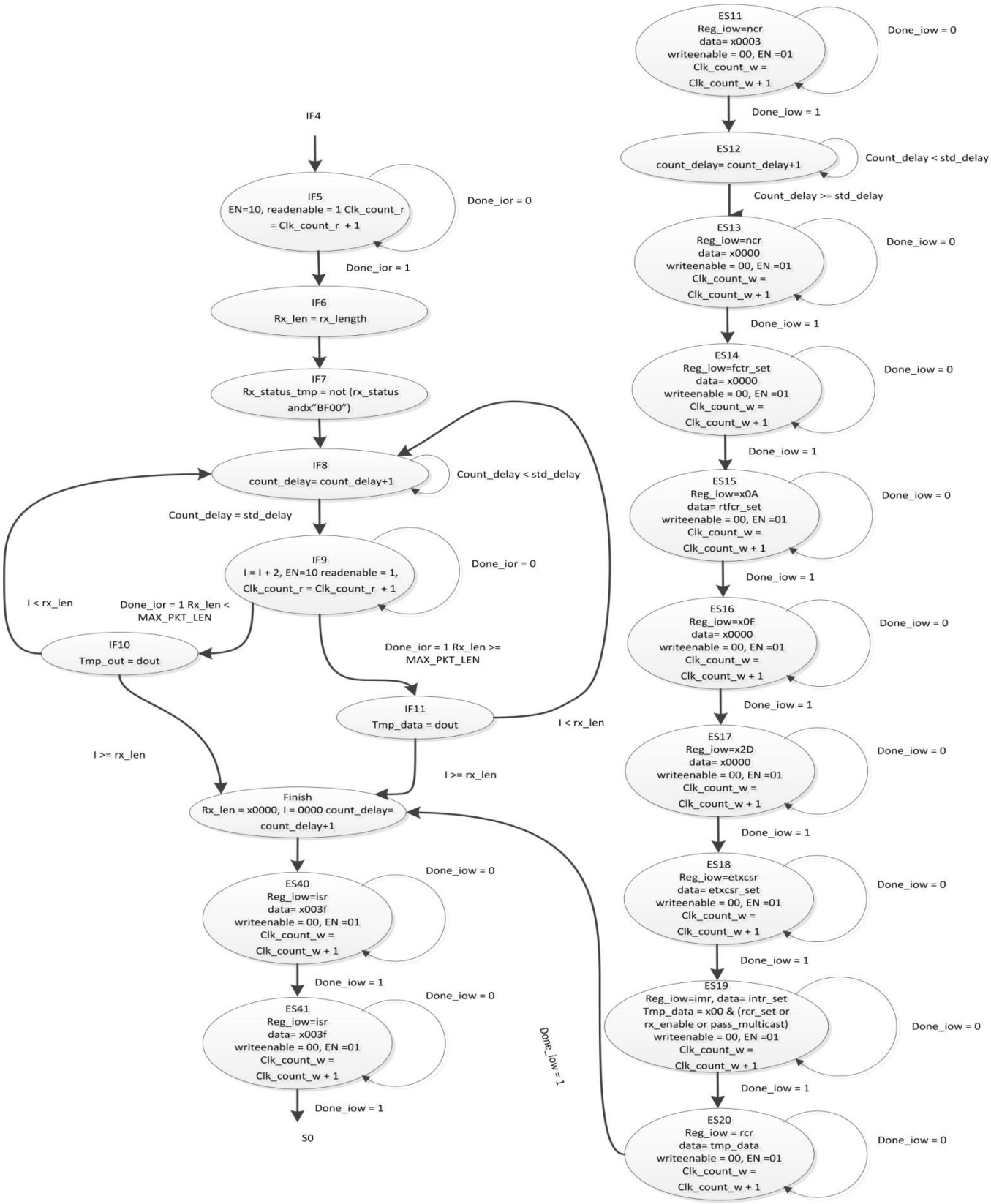


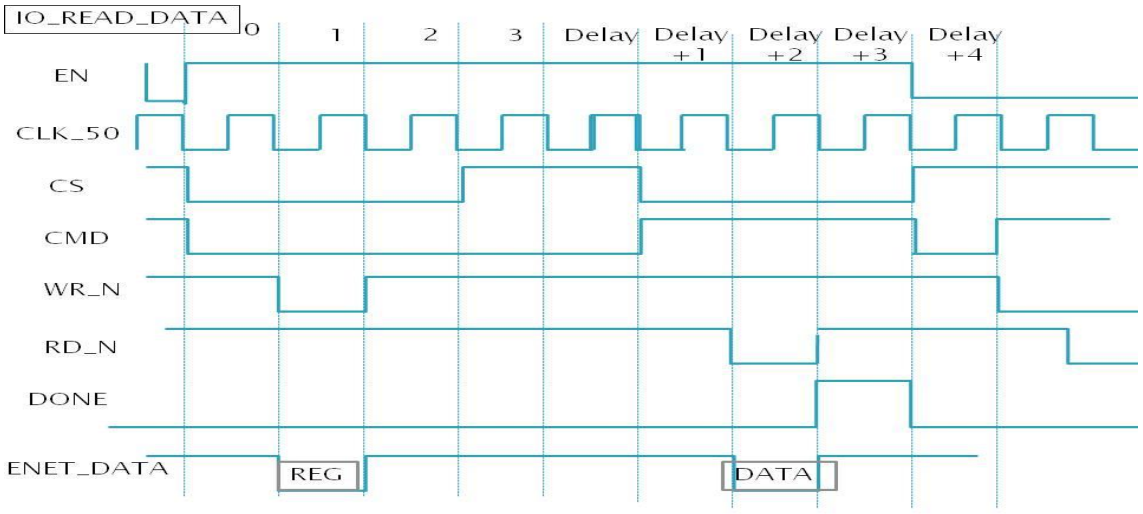
Figure 8 dm9000_hw State Diagram

3.2.3 **io_read_data**

Type: Mealy Machine

reg	IN	8	register address
clk_offset	IN	11	clock offset to meet timing requirement of PHY
clk_cnt	IN	1	clock counter to meet timing requirement of PHY
clk	IN	1	system clock
readonly	IN	1	?
en	IN	1	high to make the block enable
dout	OUT	16	data read from PHY (sent to dm900_hw)
dm_iow_n	OUT	1	used to write data from PHY
dm_ior_n	OUT	1	used to read data from PHY
done	OUT	1	high when job is done
dm_cs	OUT	1	chip select
dm_rst_n	OUT	1	reset
iData2	INOUT	16	bidirectional data bus to write and read from PHY

The state diagram is shown in the next page.



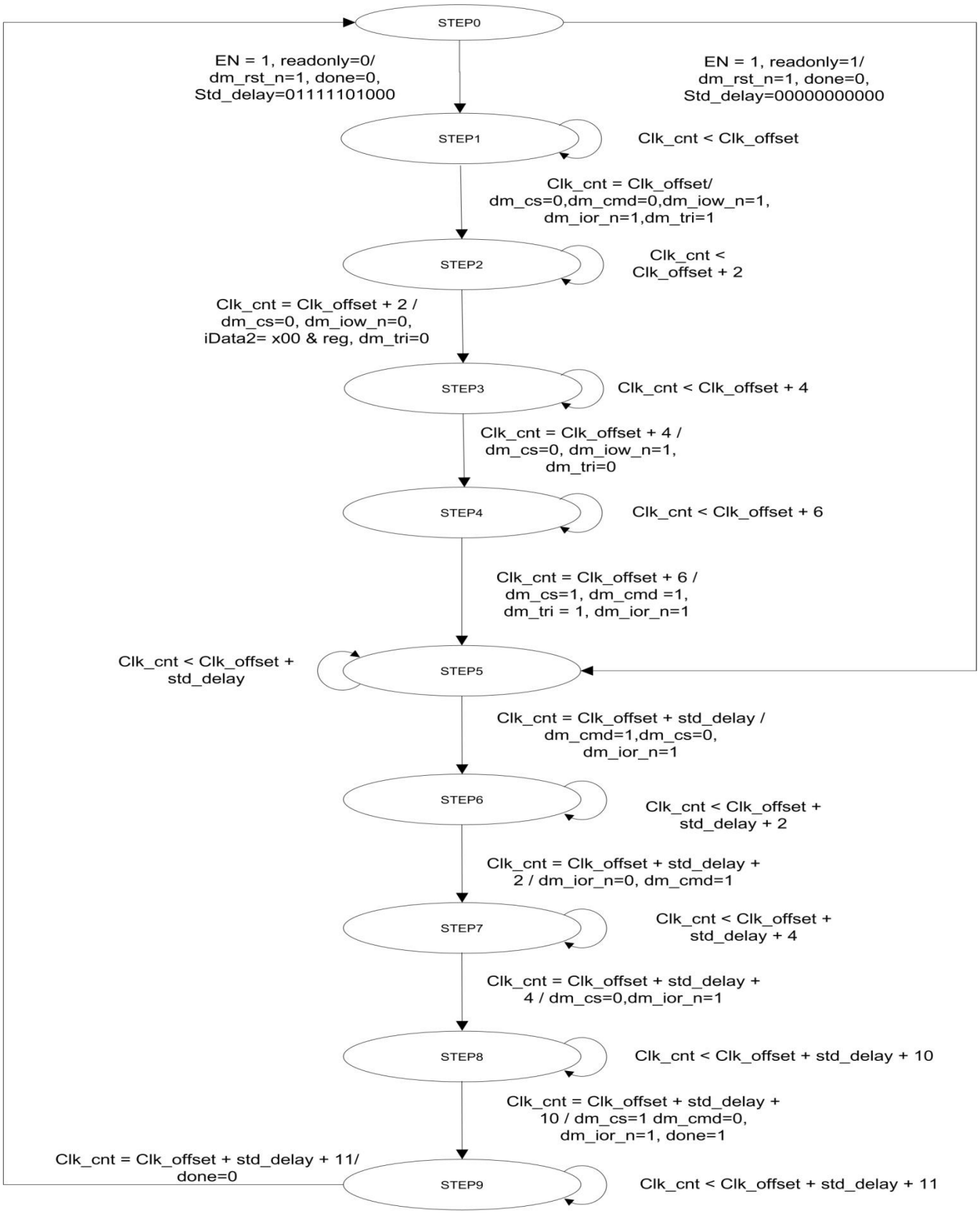


Figure 9 State Diagram of io_read_data

4. Verification of State Machines

The functionality of three state machines is verified by using vector waveform generator of Altera Quartus II.

Dm9000_hw: The simulated waveform is shown below. Here, the clock period is taken as 10ns and for testing purpose. The waveform represents the generated control signals to drive io_write_data and io_read_data when dm9000_hw get interrupted by DM9000 PHY and reads a data packet of length 1. The simulated result perfectly matches with expected outputs.

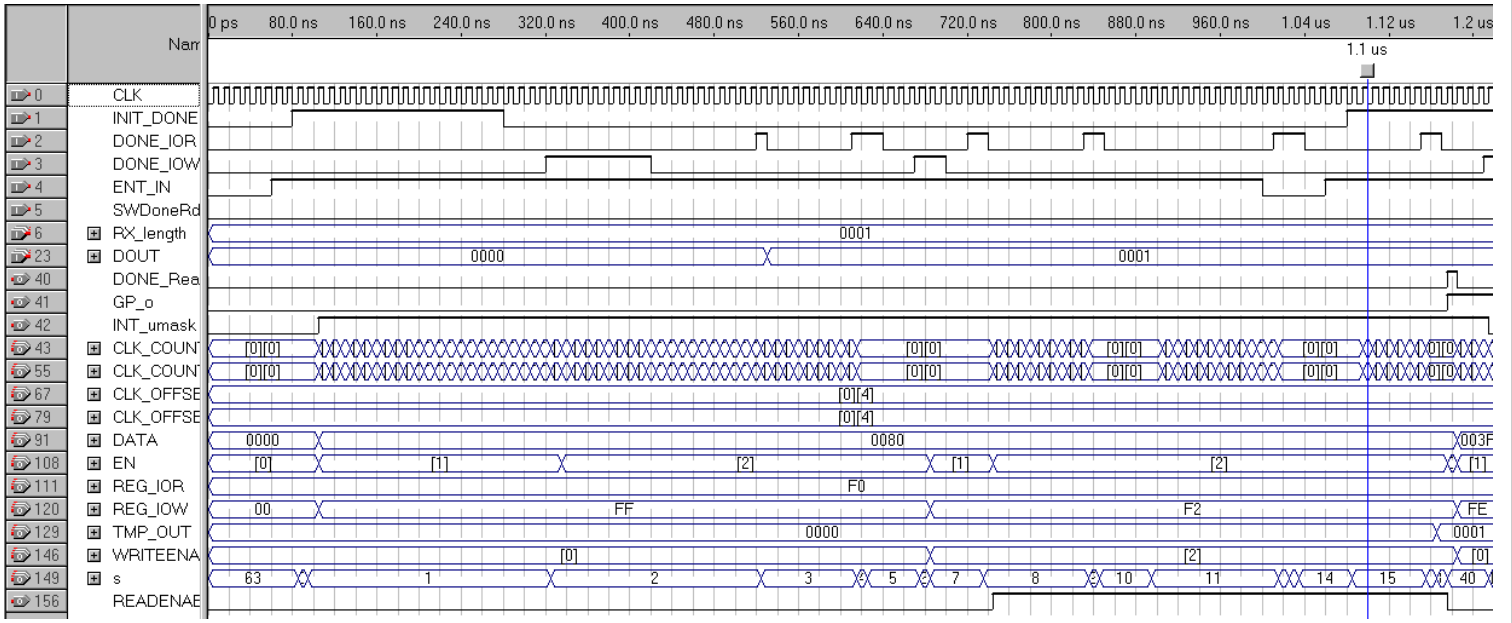


Figure 10 Functional Simulation result for dm9000_hw state machine

Io_read_data:

Case I: Case I checks and verifies the functionality for io_read_data FSM when readonly is '0'. It basically simulates the functionality of *dm9000_ior* which was defined in dm9000 software driver. As expected, the simulation result verifies that the state machine performs all the operation from state 0 to 9 and generates correct signals to interact with PHY.

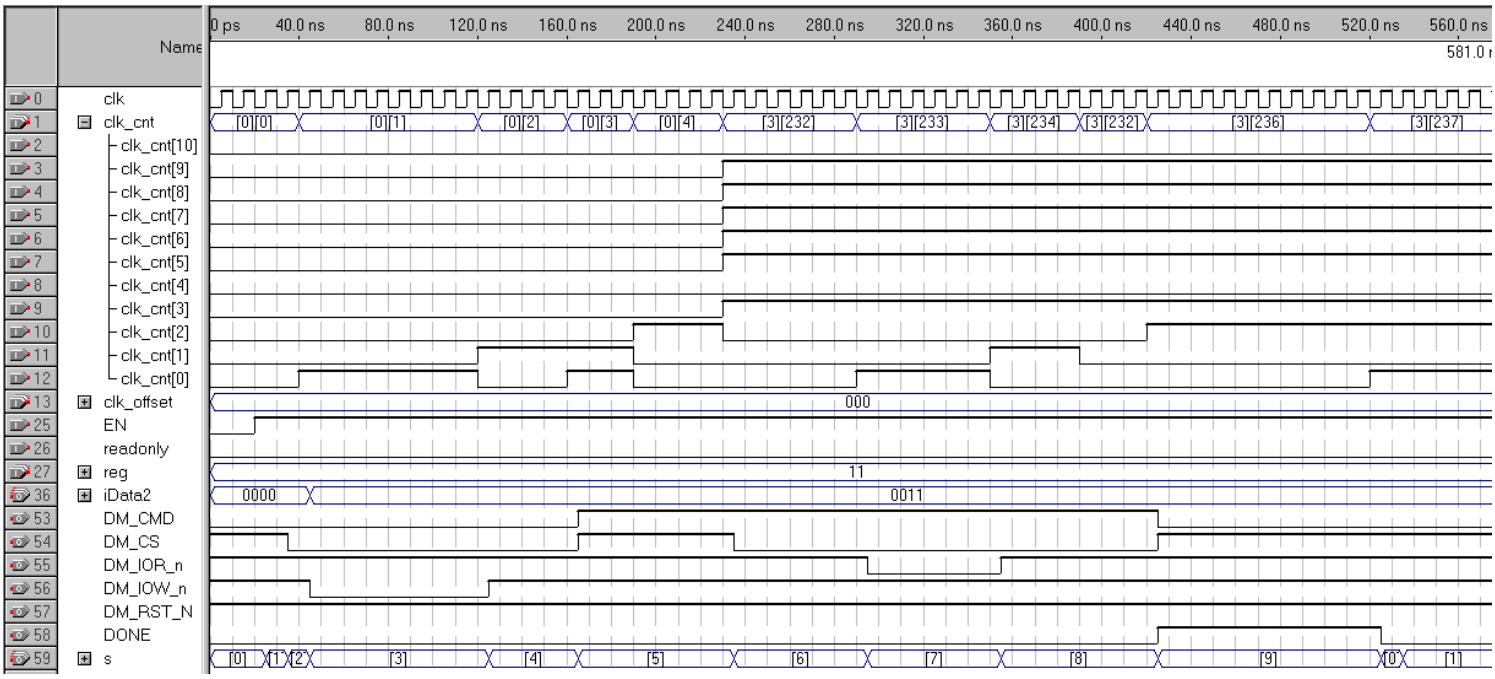


Figure 11 Functional Simulation result for io_read_data state machine [case I]

Case II: Case II simulates the functionality when readonly is '1'. It basically mimics the operation of *IOR* function defined in software driver. As defined in state machine specification, this case, it jumps to state 5 from state 1 and generates the expected output signals.

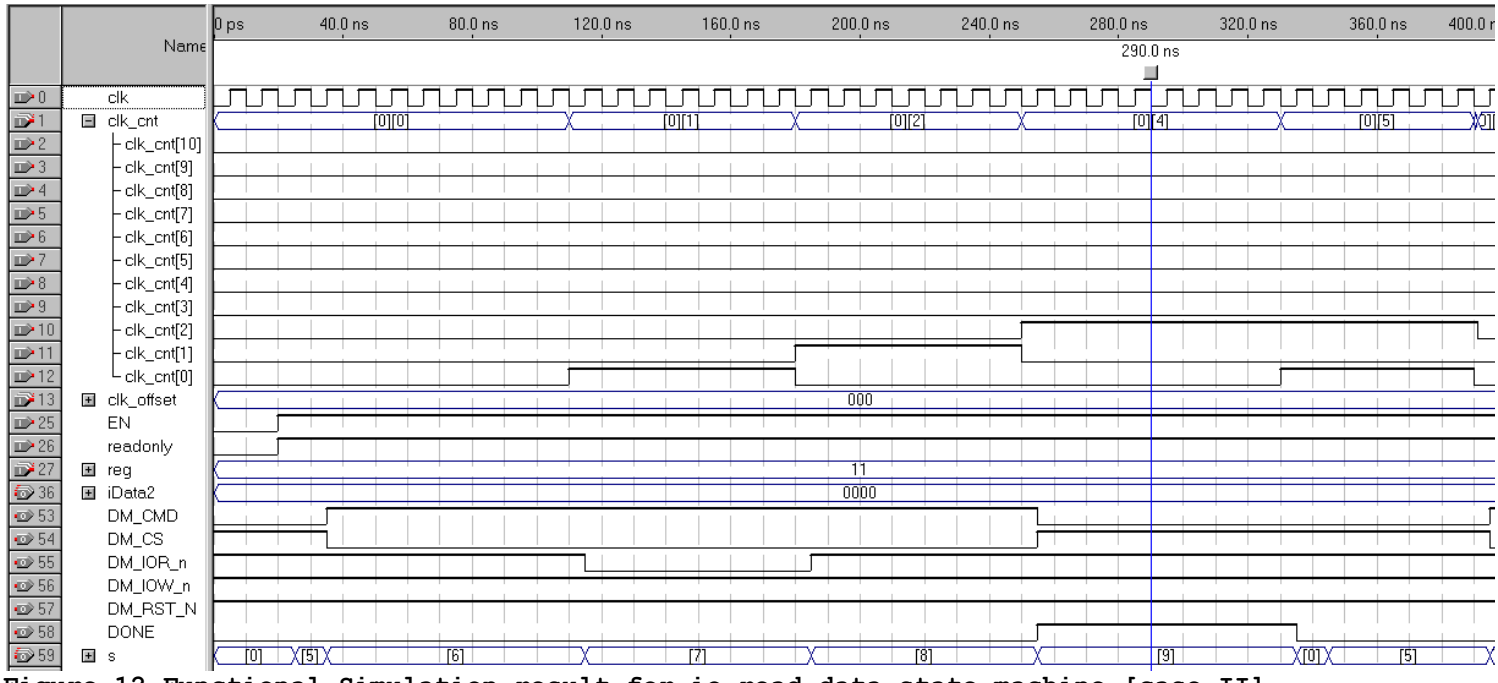


Figure 12 Functional Simulation result for io_read_data state machine [case II]

Io_write_data:

Case I: Case I checks and verifies the functionality for io_write_data FSM when writeenable is "00". It corresponds to the function *dm9000_iow* defined in driver software. The simulation result is shown below. As expected the state machine go through all the states (output "s" in the figure) and generates correct output signals to interact with the PHY.

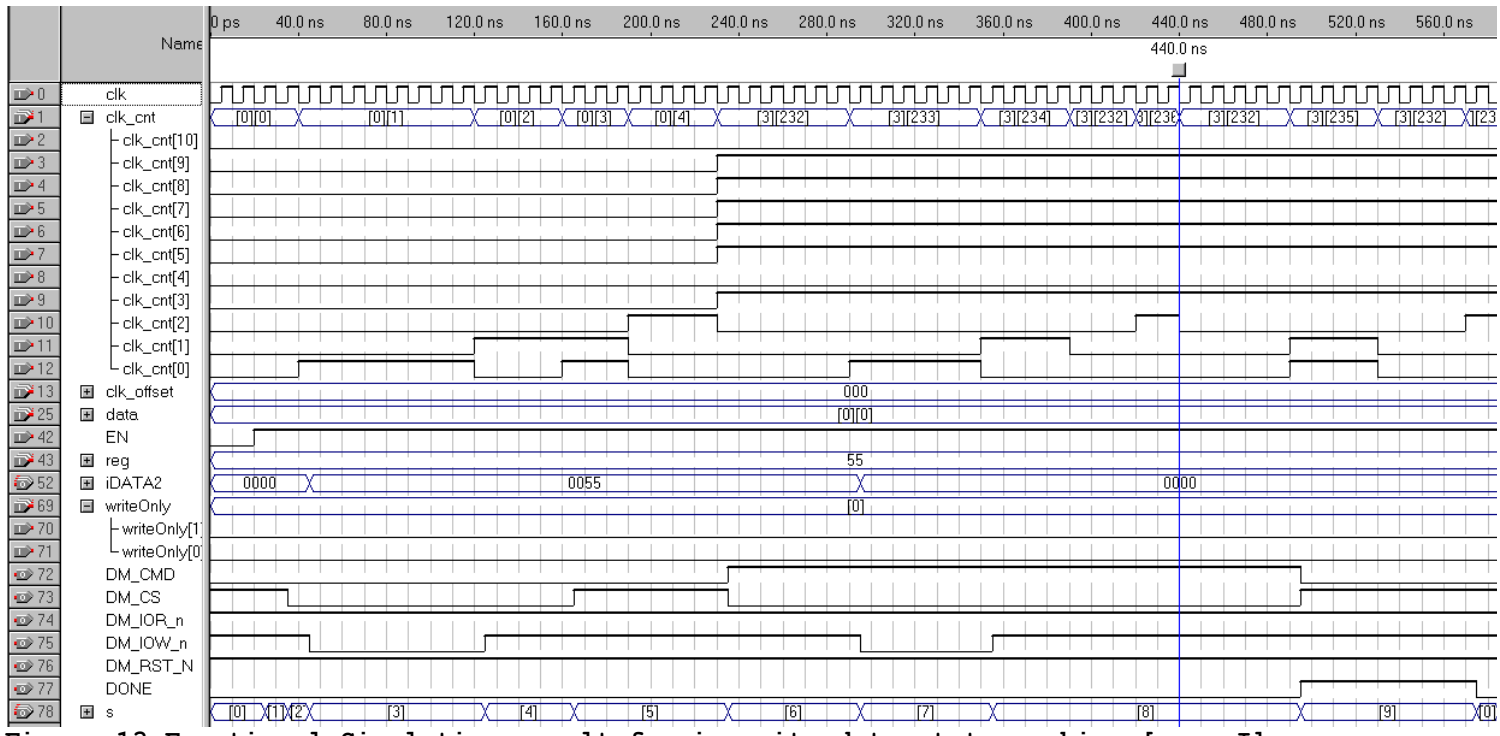


Figure 13 Functional Simulation result for io_write_data state machine [case I]

Case I: Case II simulates the state machine for writeenable = "01". It mimics the functionality of *IOW* function of the software driver. Result found from functional simulation is shown in the next page.

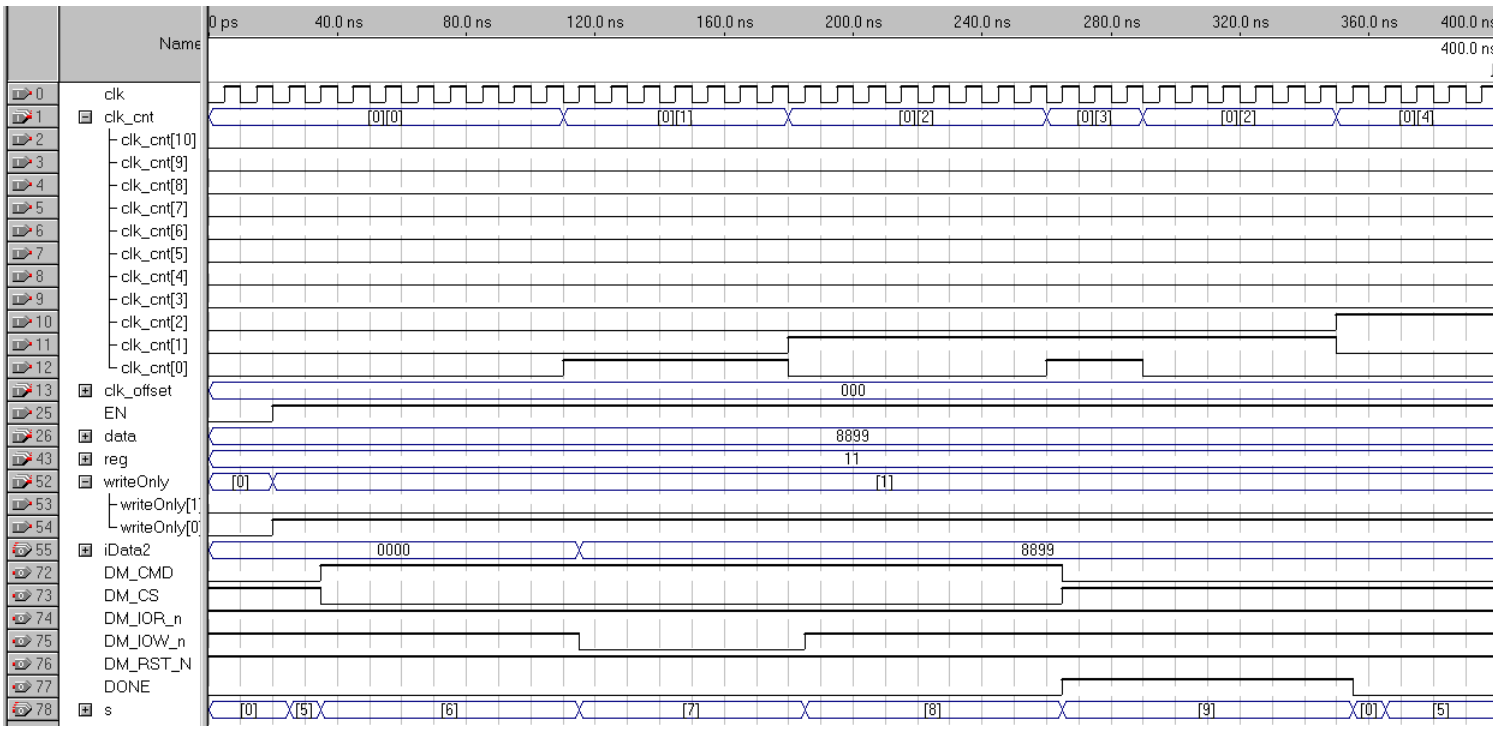


Figure 14 Functional Simulation result for io_write_data state machine [case II]

5. Runtime Verification : Signal Tap

Changes in input and output signals of custom hardware block are monitored through SignalTap analyzer. Through SignalTap, we were able to verify it's functionality by interpreting its runtime behavior. A snapshot of the runtime signal status is captured through signalTap analyzer. In figure 15, ENET_DATA represents the received data from DM9000 PHY.

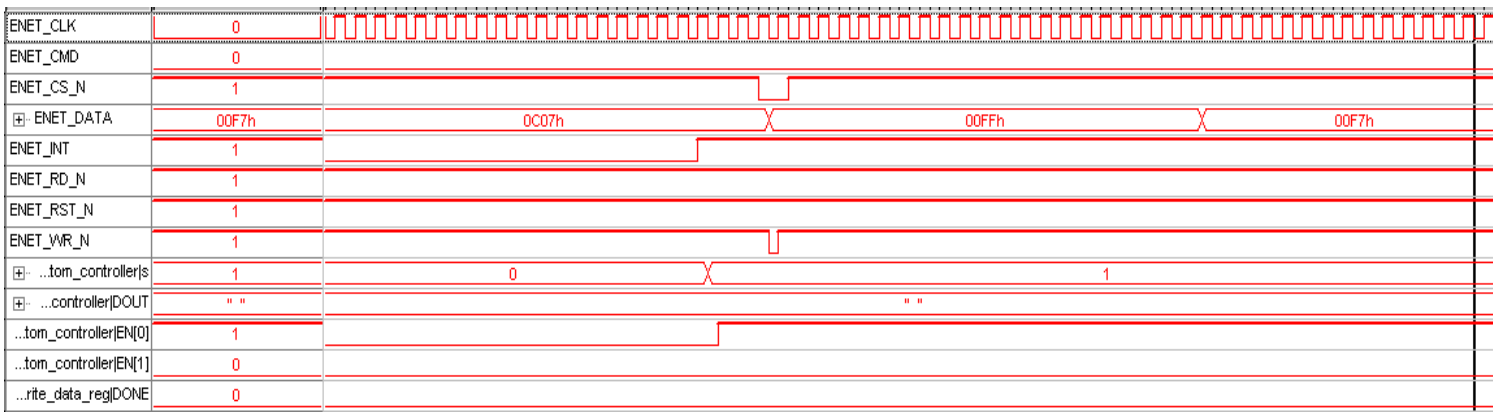


Figure 15 Snapshot from SignalTap Analyzer

6. Results

- We successfully received Market Order Packet and extracted the fields.
- Time taken for Hardware to read and decode a packet is 982 clock cycles on a 50MHz clock.
- Time taken for complete Hardware and Software system to read, decode and process a packet is 14606 clock cycles on a 50MHz clock.
- By estimating that each packet we receive is around 500 bits, we estimate our hardware receive speed to be 25Mbps.

7. Advice for future projects

Advice to future students will be as following:

- Make sure to start integrating contribution from different team mates as soon as possible because HW/SW systems will become from a very simple system to a very complex system in a matter of seconds.
- It is highly recommended to use OnChip Debugging Capabilities provided by FPGA manufacturer such as SignalTap Analyzer by Altera. It was heavily used in our project and was very helpful to debug the system while it is being emulated on FPGA in real time.
- Please use IC components which have better documentation otherwise it will be very hard to established communication between your system and the given piece of hardware.
- In the Software, do not hold the interrupts for a long time. That is, don't do lot of processing in the interrupt routine.

8. Task Division

TEAM MEMBER	CONTRIBUTION
Amandeep	Established communication between DM9000A phy using VHDL module. Designed and implemented IO_WRITE_DATA and IO_READ_DATA module to write and read from a particular registers of DM9000A. Designed test procedure and debug DM9000A_HW module using SignalTap Analyzer. Written VHDL Module for MULTIPLEXER between dm9000A_HW and software communication to DM9000A phy.
Adil	Generated and implemented dm9000_hw controller from the software driver. Verified functionality of io_read_data, dm9000_hw and io_write_data through functional simulation. Final Report

Team Member	Contribution
Manu	Established Hardware Software interface between DM9000A PHY, software and our custom hardware using SOPC builder and VHDL. Successfully Written and tested software for our HW/SW system to read extracted data fields from our custom hardware, store it an linked list based, make stock trades and display it console. Written and tested C program to send Ethernet packets so it can be used as a test bench to evaluate our overall system performance.
Prabhat	Implemented dm9000_hw controller from the software driver finite State machine with Adil. Modified Lab2 and developed a one of the two test benchmarks to generate Ethernet packet for debugging our final HW/SW system. Implemented Checksum checking using DM9000A phy to verify correctness of received Ethernet packets.

9. Source Codes

(will reformat)

9.1 VHDL

io_read_data

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;


entity io_read_data is
        port(
                reg : in std_logic_vector(7 downto 0); -- register address
                dout : out std_logic_vector(15 downto 0);
                clk_offset : in std_logic_vector(10 downto 0); -- clock offset
                clk_cnt : in std_logic_vector(10 downto 0);
                clk : in std_logic;
                readonly: in std_logic; -- if readonly=0, module performs all the opeartion from STEP0 to STEP9 and works
similar to dm9000a_ior function from DM9000A.c

                -- if readonly=1 module mimicks to operation of IORD function from DM9000A.h
                EN : in std_logic;                      -- signal from controller to wake this hardware module
                DM_IOW_n : out std_logic;       -- write_enable(active low) signal to ENET_WR_N
                DM_IOR_n : out std_logic;-- read_enable(active low) signal to ENET_RD_N
                DM_CMD : out std_logic;         -- command signal ENET_CMD Index/Data Select, 0=Index , 1 = Data
                DONE: out std_logic;            -- sends an indication to module calling io_read_data module when the read
operation is completed.
                iData2 : inout std_logic_vector(15 downto 0); -- iData2 is being fed to ENET_Data via tristate buffer
                DM_CS: out std_logic;                                   -- Chipselect to ENET_CS_N (active low)
                DM_RST_N: out std_logic;                -- DM_RST to ENET_RST_N reset_n
                s : buffer std_logic_vector(3 downto 0) -- for debugging purposes only
        );
end io_read_data;

architecture rtl_comb of io_read_data is

type states is (STEP0,STEP1, STEP2, STEP3,STEP4,STEP5,STEP6,STEP7,STEP8,STEP9);
signal y: states;

--signal STDdelay: std_logic_vector(10 downto 0):="00000001010"; -- binary of 10
signal STDdelay: std_logic_vector(10 downto 0):="00001100100"; -- binary of 100
--signal STDdelay: std_logic_vector(10 downto 0):="01111101000"; -- binary of 1000



begin -- begining of architecture

process(clk_cnt, EN)
begin
if (EN = '0') then
        y <= STEP0;
        DM_CS<='1';
        DM_RST_N<='1';
        DM_IOW_n <= '1';
        DM_IOR_n <= '1';
        DM_CMD   <= '0';
elsif (clk'EVENT and clk = '1') then
case y is
        when STEP0=>if (EN = '1') then
                                                                DM_RST_N<='1';
```

```
                                                                DONE <= '0';
                                                                if(readonly='1') then

        STDdelay<=(others=>'0');
                                                                        y <= STEP5;
                                                                else
                                                                        --
STDdelay<="00000001010";--10

        STDdelay<="00001100100";--100
                                                                        --
STDdelay<="01111101000";--1000
                                                                        y <= STEP1;
                                                                end if;

                                                        else
                                                                y<=STEP0;
                                                        end if;

        when STEP1=>if clk_cnt = clk_offset then
                                                                DM_CS<='0';
                                                                DM_CMD   <= '0';-- -- Index/Data Select,
0=Index , 1 = Data
                                                                DM_IOW_n <= '1';
                                                                DM_IOR_n <= '1';
                                                                y<=STEP2;
                                                        else
                                                                y<=STEP1;
                                                        end if;
        -- STEP2 bring WR signal to active low and start writing data on the databus
        when STEP2=>if clk_cnt = clk_offset + 1 then --
                                                                DM_CS<='0';
                                                                DM_IOW_n <= '0';
                                                                iData2   <= X"00" & reg;
                                                                y<=STEP3;
                                                        else
                                                                y<=STEP2;
                                                        end if;
        -- STEP3 bring WR signal to active high
        when STEP3=>if clk_cnt = clk_offset + 2 then
        -- adding delay
                                                                DM_CS<='0';
                                                                DM_IOW_n <= '1';
                                                                y<=STEP4;
                                                        else
                                                                y<=STEP3;
                                                        end if;
        -- STEP4 make CMD signal high and Chipselect to active low and enable(DM_tri) tristate buffer on databus to HIGH Z
        when STEP4=>if clk_cnt= clk_offset + 3 then --
                                                                DM_CS<='1';
                                                                DM_CMD   <= '1';
                                                                DM_IOR_n <= '1';
                                                                y<=STEP5;
                                                        else
                                                                y<=STEP4;
                                                        end if;
        -- STEP5 wait for 1us =
        when STEP5=>if clk_cnt = clk_offset + STDdelay then
                                                                DM_CMD   <= '1';
                                                                DM_CS<='0';
                                                                DM_IOR_n <= '1';
                                                                y<=STEP6;
                                                        else
                                                                y<=STEP5;
                                                        end if;
        when STEP6=>if clk_cnt = clk_offset + STDdelay+ 1 then
                                                                DM_IOR_n <= '0';
                                                                DM_CMD   <= '1';
                                                                y<=STEP7;
                                                        else
```

```
                                                                    y<=STEP6;
                                                            end if;

        when STEP7=>if clk_cnt = clk_offset + STDdelay+2 then
                                                                    DM_CS<='0';
                                                                    DM_IOR_n <= '1';
                                                                    y<=STEP8;
                                                            else
                                                                    y<=STEP7;
                                                            end if;

        when STEP8=>if clk_cnt = clk_offset + STDdelay+4 then
                                                                    DM_CS<='1';
                                                                    DM_CMD   <= '0';
                                                                    DM_IOR_n <= '1';
                                                                    DONE<='1';
                                                                    y<=STEP9;
                                                            else
                                                                    y<=STEP8;
                                                            end if;

        when STEP9=>if(clk_cnt=clk_offset + STDdelay + 5) then
                                                                    DONE<='0';
                                                                    y<=STEP0;
                                                            else
                                                                    y<=STEP9;
                                                            end if;
        end case;
        end if;
end process;


PROCESS(y)              ---------------------------- STATE VARIABLE
BEGIN
        IF y =STEP0 THEN
                s <= "0000";
        ELSIF y =STEP1 THEN
                s <= "0001";
        ELSIF y =STEP2 THEN
                s <= "0010";
        ELSIF y =STEP3 THEN
                s <= "0011";
        ELSIF y =STEP4 THEN
                s <= "0100";
        ELSIF y =STEP5 THEN
                s <= "0101";
        ELSIF y =STEP6 THEN
                s <= "0110";
        ELSIF y =STEP7 THEN
                s <= "0111";
        ELSIF y =STEP8 THEN
                s <= "1000";
        ELSIF y =STEP9 THEN
                s <= "1001";
        end if;
end process;
end rtl_comb;
```

io_write_data.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;


entity io_write_data is
	port(
				reg : in std_logic_vector(7 downto 0);
				data : in std_logic_vector(15 downto 0);
				clk_offset : in std_logic_vector(10 downto 0);
				clk_cnt : in std_logic_vector(10 downto 0);
				clk : in std_logic;
				writeOnly: in std_logic_vector(1 downto 0); -- if writeOnly = "00" or "11" normal operation write data to Reg-
ister reg

                                                                                -- if writeOnly = "01" write data to location
addressed by a previous statement

                                                                                -- if writeOnly ="10" write a register address
to databus
				EN : in std_logic;
				--interrupt : in std_logic;
				--reset : out std_logic;
				DM_IOW_n : out std_logic;
				DM_IOR_n : out std_logic;
				DM_CMD : out std_logic;
				DONE: out std_logic;
				iDATA2 : inout std_logic_vector(15 downto 0);
				DM_CS: out std_logic;
				DM_RST_N: out std_logic;
				s : buffer std_logic_vector(3 downto 0)
	);
end io_write_data;

architecture rtl_comb of io_write_data is

type states is (STEP0,STEP1, STEP2, STEP3,STEP4,STEP5,STEP6,STEP7,STEP8,STEP9);
signal y: states;

--signal STDdelay: std_logic_vector(10 downto 0):="00000001010"; -- binary of 10
signal STDdelay: std_logic_vector(10 downto 0):="00001100100"; -- binary of 100
--signal STDdelay: std_logic_vector(10 downto 0):="01111101000"; -- binary of 1000


--signal reset:std_logic:='1';


begin -- begining of architecture


process(clk_cnt, EN)
begin
if (EN = '0') then
	DONE <= '0';
	y <= STEP0;
	DM_CS<='1';
	DM_RST_N<='1';
	DM_IOW_n <= '1';
	DM_IOR_n <= '1';
elsif (clk'EVENT and clk = '1') then
```

```
case y is
	when STEP0=>if (EN = '1') then
DM_RST_N<='1';
DONE <= '0';
if(writeOnly="01") then

	STDdelay<="00000000000";
y<=STEP5;
else
--
STDdelay<="01111101000";--1000

	STDdelay<="00001100100";--100
--
STDdelay<="00000001010";--10
y <= STEP1;

end if;
else
y<=STEP0;
end if;
	when STEP1=>if clk_cnt = clk_offset then
DM_CS<='0';
DM_CMD  <= '0';-- -- Index/Data Select,
0=Index , 1 = Data

DM_IOW_n <= '1';
DM_IOR_n <= '1';
y<=STEP2;
else
y<=STEP1;
end if;
	-- STEP2 bring WR signal to active low and start writing register address on the databus
	when STEP2=>if clk_cnt = clk_offset + 1 then --
DM_CS<='0';
DM_IOW_n <= '0';
DM_CMD  <= '0';
iDATA2  <= X"00" & reg;
y<=STEP3;
else
y<=STEP2;
end if;
	-- STEP3 bring WR signal to active high
	when STEP3=>if clk_cnt = clk_offset + 2 then
	-- adding delay
DM_CS<='0';
DM_CMD  <= '0';
DM_IOW_n <= '1';
y<=STEP4;
else
y<=STEP3;
end if;
	-- STEP4 make CMD signal high and Chipselect to active low and enable(DM_tri) tristate buffer on databus to HIGH Z
	when STEP4=>if clk_cnt = clk_offset + 3 then --
DM_CS<='1';
DM_CMD  <= '0';
DM_IOR_n <= '1';
if(writeOnly="10") then
y<=STEP8;
else
y<=STEP5;
end if;
else
y<=STEP4;
end if;
	-- STEP5 wait for 1us =
	when STEP5=>if clk_cnt = clk_offset + STDdelay then
DM_CS<='0';
DM_CMD  <= '1';-- -- Index/Data Select, 0=Index , 1
= Data
DM_IOW_n <= '1';
```

```
                                                                DM_IOR_n <= '1';
                                                                y<=STEP6;
                                                        else
                                                                y<=STEP5;
                                                        end if;
        -- STEP6 bring WR signal to active low and start writing data on the databus
        when STEP6=>if clk_cnt = clk_offset + STDdelay+ 1 then
                                                                DM_CS<='0';
                                                                DM_IOW_n <= '0';
                                                                iDATA2   <= data;
                                                                y<=STEP7;
                                                        else
                                                                y<=STEP6;
                                                        end if;

        when STEP7=>if clk_cnt = clk_offset + STDdelay+2 then
                                                                DM_CS<='0';
                                                                DM_IOW_n <= '1';
                                                                y<=STEP8;
                                                        else
                                                                y<=STEP7;
                                                        end if;

        when STEP8=>if clk_cnt = clk_offset + STDdelay+3 then
                                                                DM_CS<='1';
                                                                DM_CMD   <= '0';
                                                                DONE<='1';
                                                                y<=STEP9;
                                                        else
                                                                y<=STEP8;
                                                        end if;

        when STEP9=>if(clk_cnt=clk_offset+STDdelay+4) then
                                                                DONE<='0';
                                                                y<=STEP0;
                                                else
                                                                y<=STEP9;
                                                end if;
        end case;
        end if;
end process;


PROCESS(y)              ---------------------------- STATE VARIABLE
BEGIN
        IF y =STEP0 THEN
                s <= "0000";
        ELSIF y =STEP1 THEN
                s <= "0001";
        ELSIF y =STEP2 THEN
                s <= "0010";
        ELSIF y =STEP3 THEN
                s <= "0011";
        ELSIF y =STEP4 THEN
                s <= "0100";
        ELSIF y =STEP5 THEN
                s <= "0101";
        ELSIF y =STEP6 THEN
                s <= "0110";
        ELSIF y =STEP7 THEN
                s <= "0111";
        ELSIF y =STEP8 THEN
                s <= "1000";
        ELSIF y =STEP9 THEN
                s <= "1001";
        end if;
end process;
end rtl_comb;
```

multiplexer_2_1_data.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity multiplexer_2_1_data is
	port(in0, in1: in std_logic_vector(15 downto 0);
                                        s : in std_logic;
                                        z : out std_logic_vector(15 downto 0));
end multiplexer_2_1_data;

architecture comb of multiplexer_2_1_data is
begin
                        z <=        in0 when s = '0' else
                                                        in1 when s = '1' else
                                                        (others => 'X');

end comb;
```

dm9000_hw.vhd

```
-- DM9000 receive packet module in VHDL
-- State machine to generate control signals for read and write to registers

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

ENTITY dm9000_hw IS

PORT(

-- inputs
	CLK,
	DONE_IOR,
	DONE_IOW,
	INIT_DONE,
	ENT_IN                          : IN STD_LOGIC;
	DOUT                            : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
	RX_length: IN STD_LOGIC_VECTOR(15 DOWNTO 0);
	RX_statusIn : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
	SWDoneRd : IN STD_LOGIC;


-- outputs
	REG_IOR                         : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
	Debug_RX_LEN       : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
	REG_IOW                         : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
	DATA                            : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
	READENABLE                      : OUT STD_LOGIC;
	-- added for new design
	WRITEENABLE                     : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
```

```
        EN                                          : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);                 -- modi-
fied for new design
        TMP_OUT                                     : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);

        I_DEBUG                             : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);

        DONE_Reading_Packet : OUT STD_LOGIC;
        GP_o                                : OUT STD_LOGIC;
        CLK_OFFSET_R                : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
        CLK_OFFSET_W                : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
        CLK_COUNTER_R               : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
        CLK_COUNTER_W               : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);

        INT_umask                           : OUT STD_LOGIC;   -- s40 once unmasking of interrupts is done

-- not required for design but useful to debug
        s: BUFFER STD_LOGIC_VECTOR(5 downto 0)

);

END dm9000_hw;


ARCHITECTURE behavior OF dm9000_hw IS

TYPE State_type IS(s00, s0, s1,s2,s3,s4,s5,if0, if1, if2, if3, if4, if5, if6, if7, if8, if9, if10, if11, es0,
                                        es1, es2, es3, es4, es5, es6, es7, es8, es9, es10, es11, es12, es13, es14, es15, es16, es17,
es18, es19, es20, finish,s40,s41);

SIGNAL                                  ISR                                                                 :
STD_LOGIC_VECTOR(7 DOWNTO 0) :=  x"FE";
SIGNAL                                  IMR                                                                 :
STD_LOGIC_VECTOR(7 DOWNTO 0) :=  x"FF";
SIGNAL                                  MRCMDX                                                              :
STD_LOGIC_VECTOR(7 DOWNTO 0) :=  x"F0";
SIGNAL                                  MRCMD                                                   : STD_LOGIC_VECTOR(7
DOWNTO 0) :=  x"F2";
SIGNAL                                  NCR                                                                 :
STD_LOGIC_VECTOR(7 DOWNTO 0) :=  x"00";
SIGNAL                                  ETXCSR                                                  : STD_LOGIC_VECTOR(7
DOWNTO 0) :=  x"30";
SIGNAL                                  RCR                                                                 :
STD_LOGIC_VECTOR(7 DOWNTO 0) :=  x"05";
SIGNAL                                  RX_ENABLE                                               : STD_LOGIC_VECTOR(7
DOWNTO 0) :=  x"01";
SIGNAL                                  PASS_MULTICAST                                  : STD_LOGIC_VECTOR(7 DOWNTO 0)
:=  x"08";
SIGNAL                                  RCR_Set                                                 : STD_LOGIC_VECTOR(7
DOWNTO 0) :=  x"30";


SIGNAL                          PAR_set                                                 : STD_LOGIC_VECTOR(15 DOWNTO 0)
:=  x"0080";
SIGNAL                                  NCR_Set                                                 : STD_LOGIC_VECTOR(15
DOWNTO 0) :=  x"0000";
SIGNAL                                  BPTR_Set                                        : STD_LOGIC_VECTOR(15 DOWNTO 0)
:=  x"003F";
```

```
SIGNAL                          FCTR_Set                        : STD_LOGIC_VECTOR(15 DOWNTO 0)
:=  x"005A";
SIGNAL                          RTFCR_Set                       : STD_LOGIC_VECTOR(15
DOWNTO 0) :=  x"0029";
SIGNAL                          ETXCSR_Set                      : STD_LOGIC_VECTOR(15
DOWNTO 0) :=  x"0083";
SIGNAL                          INTR_Set                        : STD_LOGIC_VECTOR(15 DOWNTO 0)
:=  x"0081";


SIGNAL                          K, L, M, M_tmp, J               : BOOLEAN;
SIGNAL                          N_S                                     : State_type:=s00;
SIGNAL                          GOODPACKET                      : STD_LOGIC         := '1';
SHARED VARIABLE         COUNT_DELAY                     : INTEGER                   := 0;
CONSTANT                        STD_DELAY                       : INTEGER
        := 10;
CONSTANT                        TEMP_DELAY                      : INTEGER
        := 10;
SIGNAL                          RX_READY                        : STD_LOGIC_VECTOR(15
DOWNTO 0);
SIGNAL                          RX_STATUS                       : STD_LOGIC_VECTOR(15
DOWNTO 0);
SIGNAL                          RX_STATUS_TMP                   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL                          RX_LEN                          : STD_LOGIC_VECTOR(15
DOWNTO 0);
SIGNAL                          I                                       :
STD_LOGIC_VECTOR(15 DOWNTO 0) := X"0000";
SIGNAL                          MAX_PACKET_LENGTH               : STD_LOGIC_VECTOR(15 DOWNTO 0)
:= x"05F2";

-- temp_reg and temp_data will be replaced by hardcoded register address and data
SIGNAL                          TEMP_REG                        : STD_LOGIC_VECTOR(7
DOWNTO 0);                      -- dummy register address,
SIGNAL                          TEMP_DATA                       : STD_LOGIC_VECTOR(15
DOWNTO 0);          -- dummy data, just to check functionality
SIGNAL                          TEMP_COUNT                      : STD_LOGIC_VECTOR(10
DOWNTO 0); -- :=  x"00";
SIGNAL                          TEMP_COUNT_OFF                  : STD_LOGIC_VECTOR(10 DOWNTO 0)
:= "00000000010";

signal                          debug                                   : std_logic; -- := '1';
signal                          done_state_machine      : std_logic     := '0';


BEGIN
DONE_Reading_Packet <= done_state_machine;

PROCESS(clk)

BEGIN
IF rising_edge(clk) then


CASE N_S IS

                        WHEN s00 => IF (INIT_DONE = '1') THEN
                                                                N_S <= s0;
```

```
                                        END IF;

                WHEN s0 =>  IF ENT_IN = '1' THEN
                                                                N_S <= s1; --s1;
                                        END IF;

                WHEN s1 =>
                                        REG_IOW <= IMR;   --TEMP_REG;                   --
DM9000A_IOW(IMR, PAR_SET);
                                        RX_STATUS <= X"0000";
                                        GOODPACKET <= '0';
                                        INT_umask <= '1';
                                        done_state_machine <= '0';

                                        DATA <= PAR_Set;  --TEMP_DATA;
                                        EN(0) <= '1';
                                        EN(1) <= '0';
                                        WRITEENABLE <= "00";
                                        N_S <= s1;
                                        TEMP_COUNT <= TEMP_COUNT + 1;

                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= s2;
                                        END IF;

                WHEN s2 =>  REG_IOR <= MRCMDX;    --TEMP_REG;                   -- RX_READ =
DM9000_IOR(MRCMDX);
                                        EN <= "10";
                                        READENABLE <= '0';

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOR = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= s3;
                                        END IF;

                WHEN s3 =>  RX_READY <= DOUT;                                   --
RX_READ = IORD(BASE, IO_DATA);

                                        EN <= "10";
                                        READENABLE <= '0';
                                        --READENABLE <= '1';
                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOR = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= s4;
                                        END IF;

                WHEN s4 =>  RX_READY <= (DOUT and x"03");
        -- (JUST READ)RX_READ = IORD(BASE, IO_DATA);
                                        N_S <= s5;

                WHEN s5 =>  COUNT_DELAY := COUNT_DELAY + 1;
                                                IF(COUNT_DELAY = STD_DELAY )THEN

                                                        IF (RX_READY(1) = '0' and
RX_READY(0) = '1') THEN
```

```
                                                                N_S <= if0; --s6;

        -- modified for debug --

                                                        ELSIF  (RX_READY(1) = '1' and
RX_READY(0) = '0')  OR (RX_READY(1) = '1' and RX_READY(0) = '1') THEN

                                                                N_S <= es0;             --s18;

                                                        ELSIF (RX_READY(1) = '0' and
RX_READY(0) = '0') THEN
                                                                N_S <= finish; --s33;
                                                        END IF;
                                                        COUNT_DELAY := 0;
                                                END IF;
                        --s6--
                        WHEN if0 => REG_IOW <=  MRCMD;  --TEMP_DATA;
                                                EN <= "01";
                                                WRITEENABLE <= "10";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= if1;--s7;
                                                END IF;
                        --s7--
                        WHEN if1 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                                IF COUNT_DELAY >= STD_DELAY THEN
                                                        COUNT_DELAY := 0;
                                                        N_S <= if2; --s8;
                                                END IF;
                        --s8--
                        WHEN if2 =>  EN <= "10";
                                                READENABLE <= '1';

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOR = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= if3; --s9;
                                                END IF;
                        --s9--
                        WHEN if3 =>  RX_STATUS <= RX_statusIn;
                                                N_S <= if4; --s10;

                        --s10--
                        WHEN if4 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                                IF COUNT_DELAY >= STD_DELAY THEN
                                                        COUNT_DELAY := 0;
                                                        N_S <= if5; --s11;
                                                END IF;
                        --s11--
                        WHEN if5 =>  EN <= "10";
                                                 READENABLE <= '1';

                                                TEMP_COUNT <= TEMP_COUNT + 1;
```

```
                                        IF DONE_IOR = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= if6; --s12;
                                        END IF;

                --s12--
                WHEN if6 =>     RX_LEN <= RX_length;
                                                                        N_S <= if7; --s13;

                --s13--
                WHEN if7 => RX_STATUS_TMP <= NOT (RX_STATUS and x"BF00");
                                        K <= (RX_LEN < MAX_PACKET_LENGTH);

                                        N_S <= if8; --s14;

                --s14--
                WHEN if8 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                        IF COUNT_DELAY >= STD_DELAY THEN
                                                I <= I + X"0002";
                                                COUNT_DELAY := 0;
                                                N_S <= if9; --s15;
                                        ELSE
                                                N_S <= if8; --s14;
                                        END IF;
                --s15--
                WHEN if9 =>  EN <= "10";
                                         READENABLE <= '1';
        --                               I <= I + X"0002";
                -- need to double check.

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOR = '1'  and K = TRUE THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= if10; --s16;
        --                                      GP_o <= '0';
                                        ELSIF  DONE_IOR = '1'  and K = FALSE THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= if11; --s17;
                                                GP_o <= '0';
                                        ELSIF  DONE_IOR = '0' THEN
                                           N_S <= if9;
                                        END IF;
                --s16--
                WHEN if10 => TMP_OUT <= DOUT;
                                         J <= (I < RX_LEN);


                                        IF (I < RX_LEN) THEN
                                                N_S <= if8;    --s14;
                                        ELSE
                                                N_S <= finish;  --s39;
                                        END IF;
                --s17--
                WHEN if11 =>  TEMP_DATA <= DOUT;   -- Condition for bad packet
                                         J <= I < RX_LEN;

                                        IF I < RX_LEN THEN
```

```
                                                N_S <= if8;  --s14;
                                        ELSE
                                                N_S <= finish; --s33;
                                        END IF;
 -- original s18, else part----------------------------------------------------------------------------
                        --s18--
                        WHEN es0 =>  REG_IOW <= NCR;          --TEMP_REG;                                   -- need to
double check
                                                DATA <= X"0003";  --TEMP_DATA;
                                                EN <= "01";
                                                WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es1;  --s19;
                                                END IF;
                        --s19--
                        WHEN es1 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                                IF COUNT_DELAY = TEMP_DELAY THEN
                                                        COUNT_DELAY := 0;
                                                        N_S <= es2;  --s20;
                                                END IF;
                        --s20--
                        WHEN es2 =>  REG_IOW <= NCR;          --TEMP_REG;
                                                DATA <= X"0000";  --TEMP_DATA;
                                                EN <= "01";
                                                WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es3;  --s21;
                                                END IF;
                        --s21--
                        WHEN es3 =>  REG_IOW <= NCR;          --TEMP_REG;  dm9000a_iow (NCR, 03)
                                                DATA <= X"0003";  --TEMP_DATA;
                                                EN <= "01";
                                                WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es4;  --s22;
                                                END IF;
                        --s22--
                        WHEN es4 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                                IF COUNT_DELAY >= TEMP_DELAY THEN
                                                        COUNT_DELAY := 0;
                                                        N_S <= es5;  --s23;
                                                END IF;
                        --s23--
                        WHEN es5 =>  REG_IOW <= NCR;          --TEMP_REG;
                                                DATA <= X"0000";  --TEMP_DATA;
                                                EN <= "01";
                                                WRITEENABLE <= "00";
```

```
                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es6;  --s24;
                                        END IF;
                        --s24--
                        WHEN es6 =>  REG_IOW <= NCR;                    --TEMP_REG;
                                         DATA <= NCR_Set; --TEMP_DATA;
                                         EN <= "01";
                                         WRITEENABLE <= "00";

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es7;  --s25;
                                        END IF;
                        --s25--
                        WHEN es7 =>  REG_IOW <= x"08";                  --TEMP_REG;
                                         DATA <= BPTR_Set;              --TEMP_DATA;
                                         EN <= "01";
                                         WRITEENABLE <= "00";         -- DONE UPTO THIS

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es8; --s26;
                                        END IF;


--                      --s26--
                        WHEN es8 =>  REG_IOW <= NCR;                    --TEMP_REG;
                                         DATA <= x"0003";   --TEMP_DATA;
                                         EN <= "01";
                                         WRITEENABLE <= "00";         -- DONE UPTO THIS

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es9;
                                        END IF;

                        --s27--
                        WHEN es9 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                        IF COUNT_DELAY >= TEMP_DELAY THEN
                                                COUNT_DELAY := 0;
                                                N_S <= es10;
                                        END IF;
                        --s28--
                        WHEN es10 =>  REG_IOW <= NCR;                   --TEMP_REG;
                                         DATA <= x"0000";   --TEMP_DATA;
                                         EN <= "01";
                                         WRITEENABLE <= "00";         -- DONE UPTO THIS

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
```

```
                                                N_S <= es11;
                                        END IF;
                --s29--
                WHEN es11 =>  REG_IOW <= NCR;                   --TEMP_REG;
                                        DATA <= x"0003";   --TEMP_DATA;
                                        EN <= "01";
                                        WRITEENABLE <= "00";         -- DONE UPTO THIS

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es12;
                                        END IF;
                --s30--
         WHEN es12 =>  COUNT_DELAY := COUNT_DELAY + 1;

                                        IF COUNT_DELAY >= TEMP_DELAY THEN
                                                COUNT_DELAY := 0;
                                                N_S <= es13;
                                        END IF;
        --s31--
         WHEN es13 =>  REG_IOW <= NCR;                  --TEMP_REG;
                                        DATA <= x"0000";   --TEMP_DATA;
                                        EN <= "01";
                                        WRITEENABLE <= "00";         -- DONE UPTO THIS

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es14;
                                        END IF;

--                      --s32--
                WHEN es14 =>  REG_IOW <= X"09";       --TEMP_REG;
                                        DATA <= FCTR_Set;               --TEMP_DATA;
                                        EN <= "01";
                                        WRITEENABLE <= "00";

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es15; --s27;
                                        END IF;
                --s33--
                WHEN es15 =>  REG_IOW <= X"0A";       --TEMP_REG;
                                        DATA <= RTFCR_Set;                      --TEMP_DATA;
                                        EN <= "01";
                                WRITEENABLE <= "00";

                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                        IF DONE_IOW = '1' THEN
                                                TEMP_COUNT <=  (OTHERS => '0');
                                                N_S <= es16;  --s28;
                                        END IF;
                --s34--
                WHEN es16 =>  REG_IOW <= x"0F";       --TEMP_REG;
                                        DATA <= X"0000";  --TEMP_DATA;
                                        EN <= "01";
```

```
                                        WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es17;  --s29;
                                                END IF;
                        --s35--
                        WHEN es17 =>  REG_IOW <= X"2D";                 --TEMP_REG;
                                                DATA <= X"0080";  --TEMP_DATA;
                                                EN <= "01";
                                        WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es18;  --s30;
                                                END IF;
                        --s36--
                        WHEN es18 =>  REG_IOW <= ETXCSR;                --TEMP_REG;
                                                DATA <= ETXCSR_Set;         --TEMP_DATA;
                                                EN <= "01";
                                        WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es19; --s31;
                                                END IF;
                        --s37--
                        WHEN es19 =>  REG_IOW <= IMR;         --TEMP_REG;
                                                DATA <= INTR_Set;               --TEMP_DATA;
                                                EN <= "01";
                                        WRITEENABLE <= "00";
                                                TEMP_DATA <= X"00" & (RCR_Set  OR RX_ENABLE          OR
PASS_MULTICAST);

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= es20;  --s32;
                                                END IF;
                        --s38--
                        WHEN es20 =>  REG_IOW <= RCR;         --TEMP_REG;
                                                DATA <= TEMP_DATA;
                -- ????
                                                EN <= "01";
                                        WRITEENABLE <= "00";

                                                TEMP_COUNT <= TEMP_COUNT + 1;
                                                IF DONE_IOW = '1' THEN
                                                        TEMP_COUNT <=  (OTHERS => '0');
                                                        N_S <= finish;  --s33;
                                                END IF;
                        --s39--
                        WHEN finish =>  GP_o <= '1';
                                RX_LEN <= x"0000";
                                                I <= x"0000";
```

```
                                                --N_S <= s0;
                                                        --EN <= "00";
                                                        done_state_machine <= '1';
                                                        N_S <= s40;
                                                        EN <= "00";

                                --s40--
                                WHEN s40 => done_state_machine <= '0';
                                                IF SWDoneRd = '0' THEN
                                                        REG_IOW <= ISR;   --TEMP_REG;  dm9000a_iow(ISR, 0x3F);l
                                                        DATA <= X"003F";  --TEMP_DATA;
                                                        EN <= "01";
                                                        WRITEENABLE <= "00";

                                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                                        IF DONE_IOW = '1' THEN
                                                                TEMP_COUNT <=  (OTHERS => '0');
                                                                INT_umask <= '0';
                                                                N_S <= s41;

                                                        END IF;

                                                END IF;

                                WHEN s41 =>             INT_umask <= '1';
                                                        REG_IOW <= IMR;  --TEMP_REG;  dm9000a_iow(IMR, INTR_set);
                                                        DATA <= INTR_Set;               --TEMP_DATA;
                                                        EN <= "01";
                                                        WRITEENABLE <= "00";

                                                        TEMP_COUNT <= TEMP_COUNT + 1;
                                                        IF DONE_IOW = '1' THEN
                                                                TEMP_COUNT <=  (OTHERS => '0');
                                                                N_S <= s0;
                                                                EN <= "00";
                                                        END IF;

                        END CASE;
                END IF;
END PROCESS;

CLK_COUNTER_R <= TEMP_COUNT;
CLK_COUNTER_W <= TEMP_COUNT;
CLK_OFFSET_R  <= TEMP_COUNT_OFF;
CLK_OFFSET_W  <= TEMP_COUNT_OFF;
I_DEBUG <= I;

PROCESS(N_S)            ---------------------------- STATE VARIABLE
                        BEGIN
                                        IF N_S = s00 THEN
                                                s <= "111111";
                                        ELSIF N_S = s0 THEN
                                                s <= "000000";
                                        ELSIF N_S =s1 THEN
                                                s <= "000001";
                                        ELSIF N_S =s2 THEN
                                                s <= "000010";
                                        ELSIF N_S =s3 THEN
```

```
                s <= "000011";
ELSIF N_S =s4 THEN
                s <= "000100";
ELSIF N_S =s5 THEN
                s <= "000101";
ELSIF N_S =if0 THEN
                s <= "000110";
ELSIF N_S =if1 THEN
                s <= "000111";
ELSIF N_S =if2 THEN
                s <= "001000";
ELSIF N_S =if3 THEN
                s <= "001001";
ELSIF N_S =if4 THEN
                s <= "001010";
ELSIF N_S =if5 THEN
                s <= "001011";
ELSIF N_S =if6 THEN
                s <= "001100";
ELSIF N_S =if7 THEN
                s <= "001101";
ELSIF N_S =if8 THEN
                s <= "001110";
ELSIF N_S =if9 THEN
                s <= "001111";
ELSIF N_S =if10 THEN
                s <= "010000";
ELSIF N_S =if11 THEN
                s <= "010001";
ELSIF N_S =es0 THEN
                s <= "010010";
ELSIF N_S =es1 THEN
                s <= "010011";
ELSIF N_S =es2 THEN
                s <= "010100";
ELSIF N_S =es3 THEN
                s <= "010101";
ELSIF N_S =es4 THEN
                s <= "010110";
ELSIF N_S =es5 THEN
                s <= "010111";
ELSIF N_S =es6 THEN
                s <= "011000";
ELSIF N_S =es7 THEN
                s <= "011001";
ELSIF N_S =es8 THEN
                s <= "011010";
ELSIF N_S =es9 THEN
                s <= "011011";
ELSIF N_S =es10 THEN
                s <= "011100";
ELSIF N_S =es11 THEN
                s <= "011101";
ELSIF N_S =es12 THEN
                s <= "011110";
ELSIF N_S =es13 THEN
                s <= "011111";
ELSIF N_S =es14 THEN
```

```
                    s <= "100000";
                ELSIF N_S =es15 THEN
                    s <= "100001";
                ELSIF N_S =es16 THEN
                    s <= "100010";
                ELSIF N_S =es17 THEN
                    s <= "100011";
                ELSIF N_S =es18 THEN
                    s <= "100100";
                ELSIF N_S =es19 THEN
                    s <= "100101";
                ELSIF N_S =es20 THEN
                    s <= "100110";
                ELSIF N_S = finish THEN
                    s <= "100111";
                ELSIF N_S = s40 THEN
                    s <= "101000";
                ELSIF N_S = s41 THEN
                    s <= "101001";
                ELSE s <="111000";
            END IF;
        END PROCESS;
END BEHAVIOR;
```

ha_dm9000aCustom.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity dm9000a is
```

```
 port(
                        signal iRD_N,iWR_N,iCS_N,iRST_N: in std_logic;
                        signal address    : in  unsigned(9 downto 0);
      signal iDATA: in std_logic_vector(15 downto 0);
                        signal oDATA: out std_logic_vector(15 downto 0);
                        signal oINT: out std_logic;
                        --  DM9000A Side
      signal ENET_DATA: inout std_logic_vector(15 downto 0);
                        signal ENET_CMD,
            ENET_RD_N,ENET_WR_N,
            ENET_CS_N,ENET_RST_N: out std_logic;
      signal ENET_INT: in std_logic;
                        signal clockInput: in std_logic;
                        signal enableSignal: in std_logic_vector(15 downto 0)
     );
end dm9000a;
architecture behavior of dm9000a is

component io_read_data is
            port(
                                    reg : in std_logic_vector(7 downto 0);
                                    dout : out std_logic_vector(15 downto 0);
                                    clk_offset : in std_logic_vector(10 downto 0);
                                    clk_cnt : in std_logic_vector(10 downto 0);
                                    clk : in std_logic;
                                    readonly: in std_logic;

                                    EN : in std_logic;
                                    DM_IOW_n : out std_logic;
                                    DM_IOR_n : out std_logic;
                                    DM_CMD : out std_logic;
                                    DONE: out std_logic;
                                    iDATA2 : inout std_logic_vector(15 downto 0);
                                    DM_CS: out std_logic;
                                    DM_RST_N: out std_logic;
                                    s : buffer std_logic_vector(3 downto 0)
             );
end component;


component io_write_data is
            port(
                                    reg : in std_logic_vector(7 downto 0);
                                    data : in std_logic_vector(15 downto 0);
                                    clk_offset : in std_logic_vector(10 downto 0);
                                    clk_cnt : in std_logic_vector(10 downto 0);
                                    clk : in std_logic;
                                    writeOnly: in std_logic_vector(1 downto 0); -- if writeOnly = "00" or "11" normal operation write data to Reg-
ister reg

                                                                                                    -- if writeOnly = "01" write data to location
addressed by a previous statement

                                                                                                    -- if writeOnly ="10" write a register address
to databus
                                    EN : in std_logic;
                                    --interrupt : in std_logic;
                                    --reset : out std_logic;
```

```
                        DM_IOW_n : out std_logic;
                        DM_IOR_n : out std_logic;
                        DM_CMD : out std_logic;
                        DONE: out std_logic;
                        iDATA2 : inout std_logic_vector(15 downto 0);
                        DM_CS: out std_logic;
                        DM_RST_N: out std_logic;
                        s : buffer std_logic_vector(3 downto 0)
        );
end component;

component multiplexer is
        port(
                        s : in std_logic;
                        signal mux_IO_read_data_ENET_CMD,       -- Command/Data Select, 0 = Command, 1 = Data
                        mux_IO_read_data_ENET_CS_N,             -- Chip Select
                        mux_IO_read_data_ENET_WR_N,             -- Write
                        mux_IO_read_data_ENET_RD_N,             -- Read
                        mux_IO_read_data_ENET_RST_N : std_logic;        -- Reset
                        signal iData2read: std_logic_vector(15 downto 0);

                        signal mux_IO_write_data_ENET_CMD,      -- Command/Data Select, 0 = Command, 1 = Data
                        mux_IO_write_data_ENET_CS_N,            -- Chip Select
                        mux_IO_write_data_ENET_WR_N,            -- Write
                        mux_IO_write_data_ENET_RD_N,            -- Read
                        mux_IO_write_data_ENET_RST_N: std_logic;        --
                        signal iData2write:std_logic_vector(15 downto 0);

                        out_ENET_CMD,       -- Command/Data Select, 0 = Command, 1 = Data
                        out_ENET_CS_N,          -- Chip Select
                        out_ENET_WR_N,          -- Write
                        out_ENET_RD_N,          -- Read
                        out_ENET_RST_N : out std_logic;         -- Reset
                        iData2: out std_logic_vector(15 downto 0)
                        );
end component;



component dm9000_hw IS
PORT(

        -- inputs
        CLK,
        DONE_IOR,
        DONE_IOW,
        INIT_DONE,
        ENT_IN                  : IN STD_LOGIC;
        DOUT                    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        RX_length               : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        RX_statusIn             : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        SWDoneRd                : IN STD_LOGIC;

        -- outputs
        REG_IOR                 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        REG_IOW                 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
        DATA                                    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        READENABLE                              : OUT STD_LOGIC;
        -- added for new design
        WRITEENABLE                             : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        EN                                      : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);          -- modified for new design
        TMP_OUT                                 : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        DONE_Reading_Packet : OUT STD_LOGIC;

        I_DEBUG                                 : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        INT_umask       : OUT STD_LOGIC;
        GP_o                                    : OUT STD_LOGIC;
        CLK_OFFSET_R            : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
        CLK_OFFSET_W            : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
        CLK_COUNTER_R           : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);
        CLK_COUNTER_W           : OUT STD_LOGIC_VECTOR (10 DOWNTO 0);

        -- not required for design but useful to debug
        s: BUFFER STD_LOGIC_VECTOR(5 downto 0)

);
END component;




component debug_intermediate_signals_modules is
port(

        RXbuffer0: in std_logic_vector(15 downto 0);
        RXbuffer1: in std_logic_vector(15 downto 0);
        RXbuffer2: in std_logic_vector(15 downto 0);
        RXbuffer3: in std_logic_vector(15 downto 0);
        RXbuffer4: in std_logic_vector(15 downto 0);
        RXbuffer5: in std_logic_vector(15 downto 0);
        RXbuffer6: in std_logic_vector(15 downto 0);
--      RXbuffer01: in std_logic_vector(15 downto 0);
--      RXbuffer11: in std_logic_vector(15 downto 0);
--      RXbuffer21: in std_logic_vector(15 downto 0);
--      RXbuffer31: in std_logic_vector(15 downto 0);
--      RXbuffer41: in std_logic_vector(15 downto 0);
--      RXbuffer51: in std_logic_vector(15 downto 0);
--      RXbuffer61: in std_logic_vector(15 downto 0);
        messageCounter:in integer;
        stateCounter: in std_logic_vector(5 downto 0);
        timeHwRead:out std_logic_vector(31 downto 0);
        timeSwHwRead:out std_logic_vector(31 downto 0);
        clk: in std_logic
 );
end component;

-- signal from io_read_data module to multiplexor
signal mux_IO_read_data_ENET_DATA : std_logic_vector(15 downto 0);    -- DATA bus 16Bits
signal mux_IO_read_data_ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
mux_IO_read_data_ENET_CS_N,                -- Chip Select
mux_IO_read_data_ENET_WR_N,                 -- Write
mux_IO_read_data_ENET_RD_N,                -- Read
```

```
mux_IO_read_data_ENET_RST_N : std_logic;                    -- Reset

-- signal from io_write_data module to multiplexor
signal mux_IO_write_data_ENET_DATA : std_logic_vector(15 downto 0);    -- DATA bus 16Bits
signal mux_IO_write_data_ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
mux_IO_write_data_ENET_CS_N,                -- Chip Select
mux_IO_write_data_ENET_WR_N,                -- Write
mux_IO_write_data_ENET_RD_N: std_logic;                -- Read
signal mux_IO_write_data_ENET_RST_N: std_logic;        --

signal clk_count: std_logic_vector(10 downto 0) ;
signal DONE: std_logic;
signal iDATA2: std_logic_vector(15 downto 0);
signal iDATA2read: std_logic_vector(15 downto 0);
signal iDATA2write: std_logic_vector(15 downto 0);
signal oDATA2: std_logic_vector(15 downto 0);
signal oDATA2read: std_logic_vector(15 downto 0);
signal oDATA2write: std_logic_vector(15 downto 0);


signal sel:std_logic:='0';
signal muxSwitch_read_write:std_logic:='0';

-- output signals for multiplexor. Mapped to physical pins of DM9000A
--signal inter_ENET_DATA: std_logic_vector(15 downto 0);
signal inter_ENET_CMD,
        inter_ENET_RD_N,inter_ENET_WR_N,
        inter_ENET_CS_N,inter_ENET_RST_N,inter_ENET_INT, inter_ENET_INTHW: std_logic;

-- RAM is used to store data received from DM9000A PHY
type ram_type is array(0 to 165) of unsigned(15 downto 0) ;
signal RAM:ram_type:=(others=>(others=>'0'));
signal RAMarrayOffset:integer:=15;
signal ram_address : unsigned(9 downto 0);
--signal DMPHYInterruptFlag : std_logic := '1';
signal counter: unsigned(31 downto 0) := (others=>'0');
signal customHWInt : std_logic := '0';

-- signals below are used to communicate between dm9000_hw controller to both io_read_data and io_write module
signal  REG_IOR                 : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal  REG_IOW                 : STD_LOGIC_VECTOR(7 DOWNTO 0);
signal  DATA                    : STD_LOGIC_VECTOR(15 DOWNTO 0);
signal  READENABLE              : STD_LOGIC;
        -- added for new design
signal  WRITEENABLE             : STD_LOGIC_VECTOR(1 DOWNTO 0);
signal  EN                              : STD_LOGIC_VECTOR(1 DOWNTO 0);

signal CLK_OFFSET_R             : STD_LOGIC_VECTOR (10 DOWNTO 0);
signal CLK_OFFSET_W             : STD_LOGIC_VECTOR (10 DOWNTO 0);
signal CLK_COUNTER_R        : STD_LOGIC_VECTOR (10 DOWNTO 0);
signal CLK_COUNTER_W        : STD_LOGIC_VECTOR (10 DOWNTO 0);
signal  DONE_IOR                : STD_LOGIC;
signal  DONE_IOW                : STD_LOGIC;
SIGNAL  INIT_DONE               : STD_LOGIC;
SIGNAL  ENT_IN                  : STD_LOGIC;
SIGNAL  DOUT                    : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL  TMP_OUT                 : STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL GP_o                             : STD_LOGIC;
```

```
SIGNAL INT_umask                        : STD_LOGIC;

constant ramLength: integer:=150;
signal ramIndexCounter1:integer:=0;
signal DONE_Reading_Packet: STD_LOGIC;
TYPE State_type IS(s0,s1,s2);
TYPE State_type1 IS(readS0,readS1,readS2);
SIGNAL   nextState: State_type:=s0;
SIGNAL nState:State_type1:=readS0;


SIGNAL SWDoneRd : STD_LOGIC := '1';

signal messageCounter:integer:=0;

--
signal timeHwRead:std_logic_vector(31 downto 0);
signal timeSwHwRead:std_logic_vector(31 downto 0);
signal stateCounter:std_logic_vector(5 downto 0);
--begining of Architecture definition
begin
ram_address <= address(9 downto 0);
ENET_DATA  <= iDATA when (iWR_N='0' and sel='0') else
                                                        iDATA2 when (inter_ENET_WR_N='0' and sel='1') else
                                                        (others => 'Z');


process (clockInput)
        begin
        if rising_edge(clockInput) then
                        if iRST_N = '0' then
                                customHWInt <= '0';
                        else
                                if DONE_Reading_Packet= '1' then
                                        customHWInt <= '1';
                                        SWDoneRd <= '1';
                                elsif iWR_N = '0' and iCS_N = '0' then
                                        if customHWInt = '1' then
                                                customHWInt <= '0'; -- important: reset the irq
                                                SWDoneRd <= '0';
                                                ramIndexCounter1 <= 0;
                                        end if;
                                        elsif INT_umask = '0' then
                                        SWDoneRd <= '1';
                                end if;
                        end if;



                        if inter_ENET_CS_N = '0' then
                                if inter_ENET_RD_N = '0' then
                                        if(ramIndexCounter1<=ramLength) then
                                                RAM(ramIndexCounter1+RAMarrayOffset)<= unsigned(oDATA2);
                                                ramIndexCounter1<=ramIndexCounter1+1;
                                        end if;
                                end if;
                        end if;
```

```
                        if iRST_N = '0' then
                                oDATA <= (others => '0');
                        else
                                if iCS_N = '0' then
                                        if iRD_N = '0' then
                                                oDATA <= std_logic_vector(RAM(to_integer(ram_address)));
                                                RAM(1) <= unsigned(oDATA2);
                                        elsif iWR_N = '0' then
                                                RAM(to_integer(ram_address)) <= unsigned(iDATA);
                                        end if;
                                else
                                        --Read from the RAM here
                                        sel <= std_logic_vector(RAM(2))(0);
                                        if sel = '1' then
                                                oINT <= customHWInt;
                                        else
                                                oINT <= inter_ENET_INT;
                                   end if;

                                        RAM(3) <= unsigned(timeHwRead(15 downto 0));
                                        RAM(4) <= unsigned(timeHwRead(31 downto 16));
                                        RAM(5) <= unsigned(timeSwHwRead(15 downto 0));
                                        RAM(6) <= unsigned(timeSwHwRead(31 downto 16));



                                end if;
                        end if;
                end if;
end process;

process(sel, iCS_N, iRST_N, ENET_INT, inter_ENET_CMD, inter_ENET_CS_N, inter_ENET_RST_N)
begin
        case sel is
                when '0'=> oDATA2<= ENET_DATA;
                                                        ENET_CMD   <= std_logic(address(0));
                                                        ENET_RD_N  <= iRD_N;
                                                        ENET_WR_N <= iWR_N;
                                                        ENET_CS_N <= iCS_N;
                                                        ENET_RST_N <= iRST_N;
                                                        inter_ENET_INT<=ENET_INT;
                when others =>
                                                        oDATA2<=ENET_DATA;
                                                        ENET_CMD   <= inter_ENET_CMD;
                                                        ENET_RD_N  <= inter_ENET_RD_N;
                                                        ENET_WR_N <= inter_ENET_WR_N;
                                                        ENET_CS_N <= inter_ENET_CS_N;
                                                        ENET_RST_N <= inter_ENET_RST_N;
                                                        inter_ENET_INTHW<=ENET_INT;
        end case;

end process;

        MUX: multiplexer port map(
                                s=>EN(0),
                                mux_IO_read_data_ENET_CMD=>mux_IO_read_data_ENET_CMD,
                                mux_IO_read_data_ENET_CS_N=>mux_IO_read_data_ENET_CS_N,
```

```
                                        mux_IO_read_data_ENET_WR_N=>mux_IO_read_data_ENET_WR_N,
                                        mux_IO_read_data_ENET_RD_N=>mux_IO_read_data_ENET_RD_N,
                                        mux_IO_read_data_ENET_RST_N=>mux_IO_read_data_ENET_RST_N,
                                        iDATA2read=>iDATA2read,

                                        mux_IO_write_data_ENET_CMD=>mux_IO_write_data_ENET_CMD,
                                        mux_IO_write_data_ENET_CS_N=>mux_IO_write_data_ENET_CS_N,
                                        mux_IO_write_data_ENET_WR_N=>mux_IO_write_data_ENET_WR_N,
                                        mux_IO_write_data_ENET_RD_N=>mux_IO_write_data_ENET_RD_N,
                                        mux_IO_write_data_ENET_RST_N=>mux_IO_write_data_ENET_RST_N,
                                        iData2write=>iDATA2write,

                                        out_ENET_CMD=>inter_ENET_CMD,
                                        out_ENET_CS_N=>inter_ENET_CS_N,
                                        out_ENET_WR_N=>inter_ENET_WR_N,
                                        out_ENET_RD_N=>inter_ENET_RD_N,
                                        out_ENET_RST_N=>inter_ENET_RST_N,
                                        iData2=>iData2
                                        );



read_data_reg : io_read_data port map(

        reg => REG_IOR,
        clk_offset=> CLK_OFFSET_R,
        clk_cnt =>  CLK_COUNTER_R,
        clk =>clockInput,
        readonly=> READENABLE,

        EN => EN(1),
        DM_IOW_n =>mux_IO_read_data_ENET_WR_N,
        DM_IOR_n =>mux_IO_read_data_ENET_RD_N,
        DM_CMD => mux_IO_read_data_ENET_CMD,
        DONE=>DONE_IOR,
        iDATA2 => iData2read,
        DM_CS=>mux_IO_read_data_ENET_CS_N,
        DM_RST_N=>mux_IO_read_data_ENET_RST_N
);


write_data_reg:  io_write_data port map(
        reg => REG_IOW,
        data => DATA,
        clk_offset=>CLK_OFFSET_W,
        clk_cnt =>CLK_COUNTER_W,
        clk =>clockInput,
        writeOnly=> WRITEENABLE,
        EN => EN(0),
        DM_IOW_n =>mux_IO_write_data_ENET_WR_N,
        DM_IOR_n =>mux_IO_write_data_ENET_RD_N,
        DM_CMD => mux_IO_write_data_ENET_CMD,
        DONE=>DONE_IOW,
        iDATA2 => iData2write,
        DM_CS=>mux_IO_write_data_ENET_CS_N,
        DM_RST_N=>mux_IO_write_data_ENET_RST_N
);
```

```
custom_controller: dm9000_hw PORT MAP(

-- inputs
                        CLK=>clockInput,
                        DONE_IOR=>DONE_IOR,
                        DONE_IOW=>DONE_IOW,
                        INIT_DONE=> sel, --INIT_DONE,
                        ENT_IN=> inter_ENET_INTHW,              --ENT_IN,
                        DOUT=>std_logic_vector(RAM(ramIndexCounter1-1+RAMarrayOffset)),
                        RX_length=>std_logic_vector(RAM(3+RAMarrayOffset)),
                        RX_statusIn=>std_logic_vector(RAM(2+RAMarrayOffset)),

                        SWDoneRd => SWDoneRd,
                        INT_umask => INT_umask,
                        -- outputs
                        REG_IOR =>REG_IOR,
                        REG_IOW=>REG_IOW,
                        DATA=>DATA,
                        READENABLE      =>READENABLE,
                        WRITEENABLE=>WRITEENABLE,
                        EN=>EN,
                        TMP_OUT =>TMP_OUT,

                        GP_o=>  GP_o,
                        CLK_OFFSET_R    =>CLK_OFFSET_R,
                        CLK_OFFSET_W    =>CLK_OFFSET_W,
                        CLK_COUNTER_R   =>CLK_COUNTER_R,
                        CLK_COUNTER_W =>CLK_COUNTER_W,
                        DONE_Reading_Packet => DONE_Reading_Packet,
                        s=>stateCounter
);


debuging: debug_intermediate_signals_modules PORT MAP(

            RXbuffer0=>std_logic_vector(RAM(24+RAMarrayOffset)),
            RXbuffer1=>std_logic_vector(RAM(25+RAMarrayOffset)),
            RXbuffer2=>std_logic_vector(RAM(26+RAMarrayOffset)),
            RXbuffer3=>std_logic_vector(RAM(27+RAMarrayOffset)),
            RXbuffer4=>std_logic_vector(RAM(28+RAMarrayOffset)),
            RXbuffer5=>std_logic_vector(RAM(29+RAMarrayOffset)),
            RXbuffer6=>std_logic_vector(RAM(30+RAMarrayOffset)),
            messageCounter=>messageCounter,
            stateCounter=>stateCounter,
            timeHwRead=>timeHwRead,
            timeSwHwRead=>timeSwHwRead,
            clk=>clockInput
 );

end behavior;
```

multiplexer.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity multiplexer is
	port(
		s : in std_logic;
		signal mux_IO_read_data_ENET_CMD,	-- Command/Data Select, 0 = Command, 1 = Data
		mux_IO_read_data_ENET_CS_N,		-- Chip Select
		mux_IO_read_data_ENET_WR_N,		-- Write
		mux_IO_read_data_ENET_RD_N,		-- Read
		mux_IO_read_data_ENET_RST_N : std_logic;		-- Reset
		signal iData2read: std_logic_vector(15 downto 0);

		signal mux_IO_write_data_ENET_CMD,	-- Command/Data Select, 0 = Command, 1 = Data
		mux_IO_write_data_ENET_CS_N,		-- Chip Select
		mux_IO_write_data_ENET_WR_N,		-- Write
		mux_IO_write_data_ENET_RD_N,		-- Read
		mux_IO_write_data_ENET_RST_N: std_logic;		--
		signal iData2write:std_logic_vector(15 downto 0);

		out_ENET_CMD,	-- Command/Data Select, 0 = Command, 1 = Data
		out_ENET_CS_N,		-- Chip Select
		out_ENET_WR_N,		-- Write
		out_ENET_RD_N,		-- Read
		out_ENET_RST_N : out std_logic;		-- Reset
		iData2: out std_logic_vector(15 downto 0)
		);
end multiplexer;

architecture rtl_comb of multiplexer is


signal sel:std_logic;
begin
	-- s=1 select IO_write_data signals
	-- s=0 select IO_read_data signals

mux2_1_CMD: entity work.multiplexer_2_1 port map (
			in0=>mux_IO_read_data_ENET_CMD,
			in1=>mux_IO_write_data_ENET_CMD,
			s=>s,
			z=>out_ENET_CMD
);


mux2_1_CS_N: entity work.multiplexer_2_1 port map (
			in0=>mux_IO_read_data_ENET_CS_N,
			in1=>mux_IO_write_data_ENET_CS_N,
			s=>s,
			z=>out_ENET_CS_N
);

mux2_1_WR_N: entity work.multiplexer_2_1 port map (
			in0=>mux_IO_read_data_ENET_WR_N,
			in1=>mux_IO_write_data_ENET_WR_N,
			s=>s,
			z=>out_ENET_WR_N
);
```

```
mux2_1_RD_N: entity work.multiplexer_2_1 port map (
                        in0=>mux_IO_read_data_ENET_RD_N,
                        in1=>mux_IO_write_data_ENET_RD_N,
                        s=>s,
                        z=>out_ENET_RD_N
);


mux2_1_RST_N: entity work.multiplexer_2_1 port map (
                        in0=>mux_IO_read_data_ENET_RST_N,
                        in1=>mux_IO_write_data_ENET_RST_N,
                        s=>s,
                        z=>out_ENET_RST_N
);

mux2_1_iData2: entity work.multiplexer_2_1_data port map (
                        in0=>iData2read,
                        in1=>iData2write,
                        s=>s,
                        z=>iData2
);


end rtl_comb;
```

proj1.vhd --- top level file

```
--
-- DE2 top-level module that includes the simple VGA raster generator
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity proj1 is

  port (
    -- Clocks

    CLOCK_27,                           -- 27 MHz
    CLOCK_50,                           -- 50 MHz
    EXT_CLOCK : in std_logic;                   -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);        -- Push buttons
    SW : in std_logic_vector(17 downto 0);        -- DPDT switches

    -- LED displays

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
```

```
      : out std_logic_vector(6 downto 0);
    LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0);      -- Red LEDs

    -- RS-232 interface

    UART_TXD : out std_logic;                  -- UART transmitter
    UART_RXD : in std_logic;                   -- UART receiver

    -- IRDA interface

--    IRDA_TXD : out std_logic;                  -- IRDA Transmitter
    IRDA_RXD : in std_logic;                   -- IRDA Receiver

    -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                 -- Low-byte Data Mask
    DRAM_UDQM,                                 -- High-byte Data Mask
    DRAM_WE_N,                                 -- Write Enable
    DRAM_CAS_N,                                -- Column Address Strobe
    DRAM_RAS_N,                                -- Row Address Strobe
    DRAM_CS_N,                                 -- Chip Select
    DRAM_BA_0,                                 -- Bank Address 0
    DRAM_BA_1,                                 -- Bank Address 0
    DRAM_CLK,                                  -- Clock
    DRAM_CKE : out std_logic;                  -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);      -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);  -- Address bus
    FL_WE_N,                                   -- Write Enable
    FL_RST_N,                                  -- Reset
    FL_OE_N,                                   -- Output Enable
    FL_CE_N : out std_logic;                   -- Chip Enable

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
    SRAM_UB_N,                                 -- High-byte Data Mask
    SRAM_LB_N,                                 -- Low-byte Data Mask
    SRAM_WE_N,                                 -- Write Enable
    SRAM_CE_N,                                 -- Chip Enable
    SRAM_OE_N : out std_logic;                 -- Output Enable

    -- USB controller

    OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
    OTG_ADDR : out std_logic_vector(1 downto 0);    -- Address
    OTG_CS_N,                                  -- Chip Select
    OTG_RD_N,                                  -- Write
    OTG_WR_N,                                  -- Read
    OTG_RST_N,                                 -- Reset
    OTG_FSPEED,                    -- USB Full Speed, 0 = Enable, Z = Disable
    OTG_LSPEED : out std_logic;     -- USB Low Speed, 0 = Enable, Z = Disable
```

```
OTG_INT0,                       -- Interrupt 0
OTG_INT1,                       -- Interrupt 1
OTG_DREQ0,                        -- DMA Request 0
OTG_DREQ1 : in std_logic;             -- DMA Request 1
OTG_DACK0_N,                      -- DMA Acknowledge 0
OTG_DACK1_N : out std_logic;            -- DMA Acknowledge 1

-- 16 X 2 LCD Module

LCD_ON,             -- Power ON/OFF
LCD_BLON,             -- Back Light ON/OFF
LCD_RW,             -- Read/Write Select, 0 = Write, 1 = Read
LCD_EN,             -- Enable
LCD_RS : out std_logic;     -- Command/Data Select, 0 = Command, 1 = Data
LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

-- SD card interface

SD_DAT,             -- SD Card Data
SD_DAT3,              -- SD Card Data 3
SD_CMD : inout std_logic;   -- SD Card Command Signal
SD_CLK : out std_logic;     -- SD Card Clock

-- USB JTAG link

TDI,                -- CPLD -> FPGA (data in)
TCK,                -- CPLD -> FPGA (clk)
TCS : in std_logic;         -- CPLD -> FPGA (CS)
TDO : out std_logic;        -- FPGA -> CPLD (data out)

-- I2C bus

I2C_SDAT : inout std_logic; -- I2C Data
I2C_SCLK : out std_logic;   -- I2C Clock

-- PS/2 port

PS2_DAT,              -- Data
PS2_CLK : in std_logic;     -- Clock

-- VGA output

VGA_CLK,                    -- Clock
VGA_HS,                     -- H_SYNC
VGA_VS,                     -- V_SYNC
VGA_BLANK,                    -- BLANK
VGA_SYNC : out std_logic;             -- SYNC
VGA_R,                    -- Red[9:0]
VGA_G,                    -- Green[9:0]
VGA_B : out unsigned(9 downto 0);           -- Blue[9:0]

--  Ethernet Interface

ENET_DATA : inout std_logic_vector(15 downto 0);    -- DATA bus 16Bits
ENET_CMD,         -- Command/Data Select, 0 = Command, 1 = Data
ENET_CS_N,                    -- Chip Select
ENET_WR_N,                    -- Write
ENET_RD_N,                    -- Read
```

```
    ENET_RST_N,                             -- Reset
    ENET_CLK : out std_logic;                   -- Clock 25 MHz
    ENET_INT : in std_logic;                    -- Interrupt

    -- Audio CODEC

    AUD_ADCLRCK : inout std_logic;                -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                  -- ADC Data
    AUD_DACLRCK : inout std_logic;                -- DAC LR Clock
    AUD_DACDAT : out std_logic;                 -- DAC Data
    AUD_BCLK : inout std_logic;                 -- Bit-Stream Clock
    AUD_XCK : out std_logic;                    -- Chip Clock

    -- Video Decoder

    TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
    TD_HS,                              -- H_SYNC
    TD_VS : in std_logic;                   -- V_SYNC
    TD_RESET : out std_logic;               -- Reset

    -- General-purpose I/O

    GPIO_0,                             -- GPIO Connection 0
    GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
    );

end proj1;

architecture datapath of proj1 is

  signal clk25 : std_logic := '0';
  signal reset_n : std_logic;
  signal counter : unsigned(15 downto 0);
    signal mux_NIOS_ENET_DATA : std_logic_vector(15 downto 0);    -- DATA bus 16Bits
    signal mux_NIOS_ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
    mux_NIOS_ENET_CS_N,                             -- Chip Select
    mux_NIOS_ENET_WR_N,                              -- Write
    mux_NIOS_ENET_RD_N,                             -- Read
    mux_NIOS_ENET_RST_N : std_logic;                                -- Reset



    signal mux_RECV_ENET_DATA : std_logic_vector(15 downto 0);    -- DATA bus 16Bits
    signal mux_RECV_ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
    mux_RECV_ENET_CS_N,                             -- Chip Select
    mux_RECV_ENET_WR_N,                              -- Write
    mux_RECV_ENET_RD_N: std_logic;                              -- Read
    signal mux_RECV_ENET_RST_N: std_logic;                      --

signal clk_count : std_logic_vector(10 downto 0) := (others => '0');
signal enableWrite: std_logic;
signal enableRead: std_logic;
signal muxSwitch:std_logic:='0';



begin
```

```
 process (CLOCK_50)
 begin
  if rising_edge(CLOCK_50) then
    clk25 <= not clk25;
 if counter = x"ffff" then
      reset_n <= '1';
    else
      reset_n <= '0';
      counter <= counter + 1;
    end if;
  end if;
 end process;



 LEDR(17) <= '1';
 LEDR(16) <= '0';

nios : entity work.nios_system port map (
   clk                        => CLOCK_50,
   reset_n                    => reset_n,
--    leds_from_the_leds          => LEDR(15 downto 0),
--centreH_from_the_vga                          => centreH,
--centreV_from_the_vga                          => centreV,
   SRAM_ADDR_from_the_sram     => SRAM_ADDR,
   SRAM_CE_N_from_the_sram     => SRAM_CE_N,
   SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
   SRAM_LB_N_from_the_sram     => SRAM_LB_N,
   SRAM_OE_N_from_the_sram     => SRAM_OE_N,
   SRAM_UB_N_from_the_sram     => SRAM_UB_N,
   SRAM_WE_N_from_the_sram     => SRAM_WE_N,

                              ENET_CMD_from_the_dm9000aCustom    =>ENET_CMD,
                              ENET_CS_N_from_the_dm9000aCustom =>ENET_CS_N,
                              ENET_DATA_to_and_from_the_dm9000aCustom  =>ENET_DATA,
                              ENET_INT_to_the_dm9000aCustom  => ENET_INT,
                              ENET_RD_N_from_the_dm9000aCustom  => ENET_RD_N,
                              ENET_RST_N_from_the_dm9000aCustom=> ENET_RST_N,
                              ENET_WR_N_from_the_dm9000aCustom  => ENET_WR_N,
                              enableSignal_to_the_dm9000aCustom => SW(15 downto 0)

--                            ENET_CMD_from_the_dm9000aCustom    =>mux_NIOS_ENET_CMD,
--                            ENET_CS_N_from_the_dm9000aCustom =>mux_NIOS_ENET_CS_N,
--                            ENET_DATA_to_and_from_the_dm9000aCustom  =>ENET_DATA,
--                            ENET_INT_to_the_dm9000aCustom  => ENET_INT,
--                            ENET_RD_N_from_the_dm9000aCustom  => mux_NIOS_ENET_RD_N,
--                            ENET_RST_N_from_the_dm9000aCustom=> mux_NIOS_ENET_RST_N,
--                            ENET_WR_N_from_the_dm9000aCustom  => mux_NIOS_ENET_WR_N,
--                            enableSignal_to_the_dm9000aCustom => SW(4)

   );


--mux : entity work.multiplexer port map (
--                            s=>muxSwitch,
--                            clk=>CLOCK_50,
--   -- mux_NIOS_ENET_DATA => mux_NIOS_ENET_DATA,    -- DATA bus 16Bits
--    mux_NIOS_ENET_CMD => mux_NIOS_ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
```

```
--    mux_NIOS_ENET_CS_N => mux_NIOS_ENET_CS_N,                              -- Chip Select
--    mux_NIOS_ENET_WR_N => mux_NIOS_ENET_WR_N,                               -- Write
--    mux_NIOS_ENET_RD_N => mux_NIOS_ENET_RD_N,                              -- Read
--    mux_NIOS_ENET_RST_N => mux_NIOS_ENET_RST_N,                             -- Reset
--                          -- Interrupt
--
--    --mux_RECV_ENET_DATA => mux_RECV_ENET_DATA,    -- DATA bus 16Bits
--    mux_RECV_ENET_CMD=>mux_RECV_ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
--    mux_RECV_ENET_CS_N=>mux_RECV_ENET_CS_N,                                -- Chip Select
--    mux_RECV_ENET_WR_N=>mux_RECV_ENET_WR_N,                                 -- Write
--    mux_RECV_ENET_RD_N=>mux_RECV_ENET_RD_N,                                -- Read
--    mux_RECV_ENET_RST_N=>mux_RECV_ENET_RST_N,                               -- Reset
--
--    --out_ENET_DATA =>ENET_DATA,    -- DATA bus 16Bits
--    out_ENET_CMD=>ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
--    out_ENET_CS_N=>ENET_CS_N,                                 -- Chip Select
--    out_ENET_WR_N=>ENET_WR_N,                                  -- Write
--    out_ENET_RD_N=>ENET_RD_N,                                 -- Read
--    out_ENET_RST_N=>ENET_RST_N,                                -- Reset
--    out_ENET_INT=>ENET_INT                      -- Interrupt
--
--                                                );


--recieve : entity work.recieve_packet port map (
--
--    ENET_DATA =>mux_RECV_ENET_DATA,    -- DATA bus 16Bits
--    ENET_CMD=>mux_RECV_ENET_CMD,          -- Command/Data Select, 0 = Command, 1 = Data
--    ENET_CS_N=>mux_RECV_ENET_CS_N,                                -- Chip Select
--    ENET_WR_N=>mux_RECV_ENET_WR_N,                                 -- Write
--    ENET_RD_N=>mux_RECV_ENET_RD_N,                                -- Read
--    ENET_RST_N=>mux_RECV_ENET_RST_N,                               -- Reset
--    ENET_INT =>ENET_INT,                      -- Interrupt
--                          clk=>clk25
--
--);

process (CLOCK_50)
  begin -- process DO_CLKDIV
                          --if(CLOCK_50'event) then
                                  --enableWrite<=SW(2);
                                  enableRead<=SW(2);
                                  muxSwitch<=SW(0);
                                  if SW(1) = '0' then -- asynchronous reset (active low)
                                          clk_count <= (others => '0');
     else
       clk_count <= clk_count + 1;
     end if;

                          --end if;
 end process ;
```

```
--write_data_reg : entity work. io_write_data port map(
--                                      reg =>X"10",
--                                      data =>X"FF0F",
--                                      clk_offset=>X"00",
--                                      clk_cnt=> clk_count,
--                                      clk =>clk25,
--
--                                      EN =>enableWrite,
--                                      --interrupt : in std_logic;
--                                      --reset : out std_logic;
--                                      --reset_clk_cnt
--                                      DM_IOW_n =>mux_RECV_ENET_WR_N,
--                                      DM_CMD =>mux_RECV_ENET_CMD,
--                                      DONE=>LEDR(15),
--                                      DM_SD =>mux_RECV_ENET_DATA
--                                      --s
--              );

--read_data_reg : entity work. io_read_data port map(
--                                      reg =>X"01",
--                                      clk_offset=>(others => '0'),
--                                      clk_cnt =>clk_count,
--                                      clk =>CLOCK_50,
--
--                                      EN =>enableRead,
--                                      interrupt : in std_logic;
--                                      reset : out std_logic;
--                                      DM_IOW_n =>mux_RECV_ENET_WR_N,
--                                      DM_IOR_n =>mux_RECV_ENET_RD_N,
--                                      DM_CMD => mux_RECV_ENET_CMD,
--                                      DM_SD => ENET_DATA,
--                                      DM_CS=>mux_RECV_ENET_CS_N,
--                                      DM_RST_N=>mux_RECV_ENET_RST_N
--              );



 HEX7    <= "0001001"; -- Leftmost
 HEX6    <= "0000110";
 HEX5    <= "1000111";
 HEX4    <= "1000111";
 HEX3    <= "1000000";
 HEX2    <= (others => '1');
 HEX1    <= (others => '1');
 HEX0    <= (others => '1');         -- Rightmost
 LEDG    <= (others => '1');
-- LEDR    <= (others => '1');
 LCD_ON   <= '1';
 LCD_BLON <= '1';
 LCD_RW <= '1';
 LCD_EN <= '0';
 LCD_RS <= '0';

 SD_DAT3 <= '1';
 SD_CMD <= '1';
```

```
 SD_CLK <= '1';

 SRAM_DQ <= (others => 'Z');
-- SRAM_ADDR <= (others => '0');
 --SRAM_UB_N <= '1';
 --SRAM_LB_N <= '1';
-- SRAM_CE_N <= '1';
-- SRAM_WE_N <= '1';
-- SRAM_OE_N <= '1';

 UART_TXD <= '0';
 DRAM_ADDR <= (others => '0');
 DRAM_LDQM <= '0';
 DRAM_UDQM <= '0';
 DRAM_WE_N <= '1';
 DRAM_CAS_N <= '1';
 DRAM_RAS_N <= '1';
 DRAM_CS_N <= '1';
 DRAM_BA_0 <= '0';
 DRAM_BA_1 <= '0';
 DRAM_CLK <= '0';
 DRAM_CKE <= '0';
 FL_ADDR <= (others => '0');
 FL_WE_N <= '1';
 FL_RST_N <= '0';
 FL_OE_N <= '1';
 FL_CE_N <= '1';
 OTG_ADDR <= (others => '0');
 OTG_CS_N <= '1';
 OTG_RD_N <= '1';
 OTG_RD_N <= '1';
 OTG_WR_N <= '1';
 OTG_RST_N <= '1';
 OTG_FSPEED <= '1';
 OTG_LSPEED <= '1';
 OTG_DACK0_N <= '1';
 OTG_DACK1_N <= '1';

 TDO <= '0';

--  ENET_CMD <= '0';
--  ENET_CS_N <= '1';
--  ENET_WR_N <= '1';
--  ENET_RD_N <= '1';
--              ENET_RST_N <= reset_n;
               ENET_CLK <= clk25;

 TD_RESET <= '0';

 I2C_SCLK <= '1';

 AUD_DACDAT <= '1';
 AUD_XCK <= '1';

 -- Set all bidirectional ports to tri-state
 DRAM_DQ     <= (others => 'Z');
 FL_DQ       <= (others => 'Z');
 SRAM_DQ     <= (others => 'Z');
```

```
OTG_DATA    <= (others => 'Z');
LCD_DATA    <= (others => 'Z');
SD_DAT      <= 'Z';
I2C_SDAT    <= 'Z';
--ENET_DATA   <= (others => '0');
AUD_ADCLRCK <= 'Z';
AUD_DACLRCK <= 'Z';
AUD_BCLK    <= 'Z';
GPIO_0      <= (others => 'Z');
GPIO_1      <= (others => 'Z');

end datapath;
--
-- DE2 top-level module that includes the simple VGA raster generator
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
```

multiplexer_2_1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity multiplexer_2_1 is
        port(in0, in1: in std_logic;
                                        s : in std_logic;
                                        z : out std_logic);
end multiplexer_2_1;

architecture comb of multiplexer_2_1 is
begin
        z <=    in0 when s = '0' else
                                        in1 when s = '1'else
                                        'X';
end comb;
```

9.2 C

DM9000a.c

```
#include <stdio.h>
#include "DM9000A.h"
#include "basic_io.h"

void dm9000a_iow(unsigned int reg, unsigned int data)
{
  IOWR(DM9000ACUSTOM_BASE, IO_addr, reg);
  usleep(STD_DELAY);
  IOWR(DM9000ACUSTOM_BASE, IO_data, data);
}

unsigned int dm9000a_ior(unsigned int reg)
{
  IOWR(DM9000ACUSTOM_BASE, IO_addr, reg);
  usleep(STD_DELAY);
  return IORD(DM9000ACUSTOM_BASE, IO_data);
```

```
}


void phy_write(unsigned int reg, unsigned int value)
{
  /* set PHY register address into EPAR REG. 0CH */
  dm9000a_iow(0x0C, reg | 0x40); /* PHY register address setting,
               and DM9000_PHY offset = 0x40 */

  /* fill PHY WRITE data into EPDR REG. 0EH & REG. 0DH */
  dm9000a_iow(0x0E, ((value >> 8) & 0xFF));   /* PHY data high_byte */
  dm9000a_iow(0x0D, value & 0xFF);            /* PHY data low_byte */

  /* issue PHY + WRITE command = 0xa into EPCR REG. 0BH */
  dm9000a_iow(0x0B, 0x8);                     /* clear PHY command first */
  IOWR(DM9000ACUSTOM_BASE, IO_data, 0x0A);  /* issue PHY + WRITE command */
  usleep(STD_DELAY);
  IOWR(DM9000ACUSTOM_BASE, IO_data, 0x08);  /* clear PHY command again */
  usleep(50);  /* wait 1~30 us (>20 us) for PHY + WRITE completion */
}

/* DM9000_init I/O routine */
unsigned int DM9000_init(unsigned char *mac_address)
{
  unsigned int  i;

  /* set the internal PHY power-on (GPIOs normal settings) */
  dm9000a_iow(0x1E, 0x01);  /* GPCR REG. 1EH = 1 selected
              GPIO0 "output" port for internal PHY */

  dm9000a_iow(0x1F, 0x00);  /* GPR  REG. 1FH GEPIO0
              Bit [0] = 0 to activate internal PHY */

  msleep(5);        /* wait > 2 ms for PHY power-up ready */
  dm9000a_iow(ISR, 0x3F);   // Reset interrupt
  dm9000a_iow(IMR, PAR_set); // Mask interupt
  /* software-RESET NIC */
  dm9000a_iow(NCR, 0x03);   /* NCR REG. 00 RST Bit [0] = 1 reset on,
              and LBK Bit [2:1] = 01b MAC loopback on */
  usleep(20);       /* wait > 10us for a software-RESET ok */
  dm9000a_iow(NCR, 0x00);   /* normalize */
  dm9000a_iow(NCR, 0x03);

  usleep(20);

  dm9000a_iow(NCR, 0x00);


  /* set GPIO0=1 then GPIO0=0 to turn off and on the internal PHY */
  dm9000a_iow(0x1F, 0x01);  /* GPR PHYPD Bit [0] = 1 turn-off PHY */
  dm9000a_iow(0x1F, 0x00);  /* PHYPD Bit [0] = 0 activate phyxcer */
  msleep(10);       /* wait >4 ms for PHY power-up */

  /* set PHY operation mode */
  phy_write(0,PHY_reset);   /* reset PHY registers back to the default state */
  usleep(50);               /* wait >30 us for PHY software-RESET ok */
  phy_write(16, 0x404);     /* turn off PHY reduce-power-down mode only */
  phy_write(4, PHY_txab);   /* set PHY TX advertised ability:
                   ALL + Flow_control */
  phy_write(0, 0x1200);     /* PHY auto-NEGO re-start enable
                  (RESTART_AUTO_NEGOTIATION +
                  AUTO_NEGOTIATION_ENABLE)
                  to auto sense and recovery PHY registers */
  msleep(5);                /* wait >2 ms for PHY auto-sense
                  linking to partner */

  /* store MAC address into NIC */
  for (i = 0; i < 6; i++)
    dm9000a_iow(16 + i, mac_address[i]);

  /* clear any pending interrupt */
```

```
  dm9000a_iow(ISR, 0x3F);  /* clear the ISR status: PRS, PTS, ROS, ROOS 4 bits,
              by RW/C1 */
  dm9000a_iow(NSR, 0x2C);  /* clear the TX status: TX1END, TX2END, WAKEUP 3 bits,
              by RW/C1 */

  /* program operating registers~ */
  dm9000a_iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
            (and disable this MAC loopback mode back to normal) */
  dm9000a_iow(0x08, BPTR_set); /* BPTR  REG.08  (if necessary) RX Back Pressure
              Threshold in Half duplex moe only:
              High Water 3KB, 600 us */
  dm9000a_iow(0x09, FCTR_set);  /* FCTR  REG.09  (if necessary)
               Flow Control Threshold setting
               High/ Low Water Overflow 5KB/ 10KB */
  dm9000a_iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary)
               RX/TX Flow Control Register enable TXPEN, BKPM
               (TX_Half), FLCE (RX) */
  dm9000a_iow(0x0F, 0x00);      /* Clear the all Event */
  dm9000a_iow(0x2D, 0x80);      /* Switch LED to mode 1 */

  /* set other registers depending on applications */
  dm9000a_iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
  /* enable interrupts to activate DM9000 ~on */
  dm9000a_iow(IMR, INTR_set);   /* IMR REG. FFH PAR=1 only,
              or + PTM=1& PRM=1 enable RxTx interrupts */

  /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
  dm9000a_iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
  /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */
  /*Enable IP header checksum*/
  dm9000a_iow(CSCR, CSCR_set);
  dm9000a_iow(RCSCSR,RCSCSR_set);
  /* RETURN "DEVICE_SUCCESS" back to upper layer */
  return  (dm9000a_ior(0x2D)==0x80) ? DMFE_SUCCESS : DMFE_FAIL;
}

unsigned int TransmitPacket(unsigned char *data_ptr, unsigned int tx_len)
{
  unsigned int i;

  /* mask NIC interrupts IMR: PAR only */
  dm9000a_iow(IMR, PAR_set);

  /* issue TX packet's length into TXPLH REG. FDH & TXPLL REG. FCH */
  dm9000a_iow(0xFD, (tx_len >> 8) & 0xFF);  /* TXPLH High_byte length */
  dm9000a_iow(0xFC, tx_len & 0xFF);         /* TXPLL Low_byte  length */

  /* wirte transmit data to chip SRAM */
  IOWR(DM9000ACUSTOM_BASE, IO_addr, MWCMD);  /* set MWCMD REG. F8H
                     TX I/O port ready */
  for (i = 0; i < tx_len; i += 2) {
    usleep(STD_DELAY);
    IOWR(DM9000ACUSTOM_BASE, IO_data, (data_ptr[i+1]<<8)|data_ptr[i] );
  }

  /* issue TX polling command activated */
  dm9000a_iow(TCR , TCR_set | TX_REQUEST);  /* TXCR Bit [0] TXREQ auto clear
                       after TX completed */

  /* wait TX transmit done */
  while(!(dm9000a_ior(NSR)&0x0C))
    usleep(STD_DELAY);

  /* clear the NSR Register */
  dm9000a_iow(NSR,0x00);

  /* re-enable NIC interrupts */
  dm9000a_iow(IMR, INTR_set);

  /* RETURN "TX_SUCCESS" to upper layer */
  return  DMFE_SUCCESS;
```

```
}

unsigned int ReceivePacket(unsigned char *data_ptr, unsigned int *rx_len)
{
  unsigned char rx_READY, GoodPacket;
  unsigned int  Tmp, RxStatus, i;

  RxStatus = rx_len[0] = 0;
  GoodPacket=FALSE;

  /* mask NIC interrupts IMR: PAR only */
  dm9000a_iow(IMR, PAR_set);

  /* dummy read a byte from MRCMDX REG. F0H */
  rx_READY = dm9000a_ior(MRCMDX);

  /* got most updated byte: rx_READY */
  rx_READY = IORD(DM9000ACUSTOM_BASE,IO_data)&0x03;
  usleep(STD_DELAY);

  /* check if (rx_READY == 0x01): Received Packet READY? */
  if (rx_READY == DM9000_PKT_READY) {

    /* got RX_Status & RX_Length from RX SRAM */
    IOWR(DM9000ACUSTOM_BASE, IO_addr, MRCMD); /* set MRCMD REG. F2H
                      RX I/O port ready */
    usleep(STD_DELAY);
    RxStatus = IORD(DM9000ACUSTOM_BASE,IO_data);
    usleep(STD_DELAY);
    rx_len[0] = IORD(DM9000ACUSTOM_BASE,IO_data);

    /* Check this packet_status GOOD or BAD? */
    if ( !(RxStatus & 0xBF00) && (rx_len[0] < MAX_PACKET_SIZE) ) {
      /* read 1 received packet from RX SRAM into RX buffer */
      for (i = 0; i < rx_len[0]; i += 2) {
    usleep(STD_DELAY);
    Tmp = IORD(DM9000ACUSTOM_BASE, IO_data);
    data_ptr[i] = Tmp & 0xFF;
    data_ptr[i+1] = (Tmp>>8) & 0xFF;
      }
      GoodPacket = TRUE;
    } else {
      /* this packet is bad, dump it from RX SRAM */
      for (i = 0; i < rx_len[0]; i += 2) {
    usleep(STD_DELAY);
    Tmp = IORD(DM9000ACUSTOM_BASE, IO_data);
      }
      printf("\nError\n");
      rx_len[0] = 0;
    }
  } else if (rx_READY) { /* status check first byte:
                rx_READY Bit[1:0] must be "00"b or "01"b */

    /* software-RESET NIC */

    dm9000a_iow(NCR, 0x03);   /* NCR REG. 00 RST Bit [0] = 1 reset on,
             and LBK Bit [2:1] = 01b MAC loopback on */
    usleep(20);       /* wait > 10us for a software-RESET ok */
    dm9000a_iow(NCR, 0x00);   /* normalize */
    dm9000a_iow(NCR, 0x03);
    usleep(20);
    dm9000a_iow(NCR, 0x00);
    /* program operating registers~ */
    dm9000a_iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
              (and disable this MAC loopback mode back to normal) */
    dm9000a_iow(0x08, BPTR_set);  /* BPTR  REG.08  (if necessary) RX Back Pressure
                 Threshold in Half duplex moe only:
                 High Water 3KB, 600 us */
    dm9000a_iow(0x09, FCTR_set);  /* FCTR  REG.09  (if necessary)
                 Flow Control Threshold setting High/Low Water
                 Overflow 5KB/ 10KB */
```

```
    dm9000a_iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary)
                RX/TX Flow Control Register
                enable TXPEN, BKPM (TX_Half), FLCE (RX) */
    dm9000a_iow(0x0F, 0x00);      /* Clear the all Event */
    dm9000a_iow(0x2D, 0x80);      /* Switch LED to mode 1 */
    /* set other registers depending on applications */
    dm9000a_iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
    /* enable interrupts to activate DM9000 ~on */
    dm9000a_iow(IMR, INTR_set);   /* IMR REG. FFH PAR=1 only,
                or + PTM=1& PRM=1 enable RxTx interrupts */
    /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
    dm9000a_iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
      /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */
  }

  return GoodPacket ? DMFE_SUCCESS : DMFE_FAIL;
}
```

hfr_altera.h

```
#ifndef HFR_ALTERA_H_
#define HFR_ALTERA_H_


typedef int(*rcvCbk)(unsigned short, unsigned char*);

/*******************************************************
* altera_init: initialize the board/ ethernet interface
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int altera_init(rcvCbk pRecvCallback);

/*******************************************************
* altera_send: send data pData over usConnectionId
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int altera_send(unsigned short usConnectionId, char* pData,
                unsigned int uiDataLen);



/*******************************************************
* displayMetrics: close the board
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int displayMetrics(void);

/*******************************************************
* altera_close: close the board
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int altera_close(void);

#endif /*HFR_ALTERA_H_*/
```

hfr_altera.c

```
      #include "basic_io.h"
```

```
#include "DM9000A.h"
#include <alt_types.h>
#include <ctype.h>
#include <string.h>
#include "hfr_altera.h"
#include "hfr_config.h"


#define MAX_MSG_LENGTH 240
#define MASK 0xFF

#define FLAG        2 //33+15
//#define

#define PORT        37 //22+15
#define ORDERID_H     40  //25+15
#define ORDERID_L     41  //26+15
#define NAME_H      42   //27+15
#define NAME_L      43   //28+15
#define PRICE_H     44  //29+15
#define PRICE_L     45  //30+15
#define QUANT_H     46  //31+15
#define QUANT_L     47  //32+15
#define BUY_SELL    48 //33+15

#define LEN_NO_MARKET 18   //3+15


// Ethernet MAC address.  Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x01, 0x60, 0x6E, 0x12, 0x03, 0x10  };

unsigned int interrupt_number;

unsigned int receive_buffer_length;
unsigned char receive_buffer[1600] = {0}; /* Be careful about this harcoded value*/

#define UDP_PACKET_PAYLOAD_OFFSET (42)
#define UDP_PACKET_LENGTH_OFFSET 38

#define IP_PACKET_ID_OFFSET  (18)

#define IP_HEADER_OFFSET  (14)
#define IP_HEADER_SIZE  (20)
#define IP_HEADER_CHECKSUM_OFFSET  (24)

#define UDP_PACKET_PAYLOAD (transmit_buffer + UDP_PACKET_PAYLOAD_OFFSET )

#define HWREAD_16(OFFSET) IORD_16DIRECT(DM9000ACUSTOM_BASE, (OFFSET)*4)
#define HWWRITE_16(OFFSET, DATA) IOWR_16DIRECT(DM9000ACUSTOM_BASE, (OFFSET)*4, DATA)
//IOWR_16DIRECT(DM9000ACUSTOM_BASE, 2*4, 1);

static unsigned short int gIPPacketIDNum = 0;
static rcvCbk gRecvCallback = NULL;

extern unsigned int charArray4ToInt(unsigned char* pBuffer);

unsigned char transmit_buffer[] = {
  // Ethernet MAC header
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC address
  0x01, 0x60, 0x6E, 0x12, 0x03, 0x10, // Source MAC address
  0x08, 0x00,                         // Packet Type: 0x800 = IP

  // IP Header
  0x45,               // version (IPv4), header length = 20 bytes
  0x00,               // differentiated services field
  0x01,0x9C,          // total length: 20 bytes for IP header +
                      // 8 bytes for UDP header + 240 bytes for payload
  0x3d, 0x35,         // packet ID
  0x00,               // flags
  0x00,               // fragment offset
  0x80,               // time-to-live
```

```
  0x11,                // protocol: 11 = UDP
  0x00,0x00,           // header checksum: incorrect
  0xc0,0xa8,0x01,0x01, // source IP address
  0xc0,0xa8,0x01,0xff, // destination IP address

  // UDP Header
  0x67,0xd9, // source port port (26585: garbage)
  0x27,0x2c, // destination port (10027: garbage)
  0x00,0xF8, // length (248: 8 for UDP header + 240 for data)
  0x00,0x00, // checksum: 0 = none

  // UDP payload (240 bytes) (First 5 bytes are for name)
  0x20, 0x20, 0x20, 0x20, 0x3A, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67,
  0x74, 0x65, 0x73, 0x74, 0x20, 0x6d, 0x73, 0x67
};

/********************************************************
 * Computes the checksum of ipHeader of iLength
 *
*******************************************************/
unsigned short int IPCheckSum(unsigned char* ipHeader, int iLength)
{
    long sum = 0;  /* Sum is 4 bytes */
    int count = 0;
    unsigned short tempSum = 0;
    /* Compute the checksum */
    while(iLength > 1){
        tempSum = *(ipHeader);
        /* Copy the ipHeader lower byte to higher byte of tempSum */
        tempSum = (tempSum << 8) + 0x00;
        tempSum = tempSum + (*(ipHeader+1));
        sum = sum +  tempSum; /* 2 bytes of ipHeader used */
        ipHeader += 2; /* Move 2 bytes */
        if(sum & 0x80000000){   /* if high order bit set (when 4 bytes of sum may not be
enuf)*/
               sum = (sum & 0xFFFF) + (sum >> 16);
        }
        iLength -= 2;
        count++;
    }

    if(iLength){       /* if  ipHeader has odd bytes */
        sum = sum + (unsigned short)(*(ipHeader));
```

```
    }

    while(sum>>16){ /* Add the contents in 3rd and 4th byte to first 2 bytes */
        sum = (sum & 0xFFFF) + (sum >> 16);
    }

    return(~sum); /* take 1's compliment and return */
}


/*********************************************************
 * handle the ethernet interrupts
 *
*********************************************************/
static void ethernet_interrupt_handler() {
    unsigned int receive_status;
    char *p;
    unsigned short usIPCheckSum = 0;
    unsigned char ucDatabuffer[100] = {0};
    unsigned short usCount = 0;

    if(HWREAD_16(FLAG) == 1){ /* Check if market data or not */
        if(HWREAD_16(PORT)==0x2b27){
            /* Copy OrderId */
            for(usCount = 0; usCount < 2; usCount++){
               ucDatabuffer[usCount] = ((HWREAD_16(ORDERID_H)) >> usCount*8) & 0xFF;
            }
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+2] = ((HWREAD_16(ORDERID_L)) >> usCount*8) & 0xFF;
            }
            /* Copy Price */
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+4] = ((HWREAD_16(PRICE_H)) >> usCount*8) & 0xFF;
            }
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+6] = ((HWREAD_16(PRICE_L)) >> usCount*8) & 0xFF;
            }

            /* Copy Name */
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+8] = ((HWREAD_16(NAME_H)) >> usCount*8) & 0xFF;
            }
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+10] = ((HWREAD_16(NAME_L)) >> usCount*8) & 0xFF;
            }

             /* Copy Quantity */
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+13] = ((HWREAD_16(QUANT_H)) >> usCount*8) & 0xFF;
            }
            for(usCount = 0; usCount < 2; usCount++){
                ucDatabuffer[usCount+15] = ((HWREAD_16(QUANT_L)) >> usCount*8) & 0xFF;
            }


             /* Copy Buy/Sell*/
            ucDatabuffer[12] = HWREAD_16(BUY_SELL);
            if(gRecvCallback){ /* Check if the application has registered the recv cbk */
                gRecvCallback(1,ucDatabuffer);
            }
        }
        else{
            printf("\n Non Market Packet\n");
            printf("Length %d\n",HWREAD_16(LEN_NO_MARKET));
        }
        //msleep(1);
        HWWRITE_16(0, 0);
    }
    else{
        printf("lab2 default\n");
        receive_status = ReceivePacket(receive_buffer, &receive_buffer_length);
```

```
        if (receive_status == DMFE_SUCCESS) {
            printf("\n\nReceive Packet Length = %d\n", receive_buffer_length);

            if (receive_buffer_length >= 14) {
                /* A real Ethernet packet */
                if (receive_buffer[12] == 8 && receive_buffer[13] == 0 &&
                receive_buffer_length >= 34) {
                    /* An IP packet */
                    /* Check IP Header Checksum */
                    usIPCheckSum = IPCheckSum(receive_buffer+IP_HEADER_OFFSET,
    IP_HEADER_SIZE);

                    if(usIPCheckSum){
                        printf("received checksum fail; discarding the packet\n");
                    }
                    else{
                        printf("received checksum successs\n");
                    }

                    if (receive_buffer[23] == 0x11) {
                        /* A UDP packet */
                        if (receive_buffer_length >= UDP_PACKET_PAYLOAD_OFFSET) {
                            /* receive_buffer has max of 1600 bytes. Hence read only 1600
    bytes */
                            receive_buffer[1599] = 0;
                            p = receive_buffer + UDP_PACKET_PAYLOAD_OFFSET;
                            //if(!usIPCheckSum){
                            if(1){
                                printf("First Byte: %x\n",receive_buffer[0]);
                                printf("Received: %s\n",receive_buffer
+UDP_PACKET_PAYLOAD_OFFSET);
                                if(gRecvCallback){ /* Check if the application has regis-
tered
    the recv cbk */
                                    gRecvCallback(1,p);
                                }
                            }
                        }
                    }
                    else{
                        printf("Received non-UDP packet\n");
                    }
                }
                else {
                    printf("Received non-IP packet\n");
                }
            }
            else {
                printf("Malformed Ethernet packet\n");
            }
        }
        else {
            printf("Error receiving packet\n");
        }

      /* Display the number of interrupts on the LEDs */
      interrupt_number++;


      /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by RW/C1 */
      dm9000a_iow(ISR, 0x3F);

      /* Re-enable DM9000A interrupts */
      dm9000a_iow(IMR, INTR_set);
    }
    return;
}


/*****************************************************
```

```
 * altera_init()
 *
*******************************************************/
int altera_init(rcvCbk pRecvCallback)
{
    unsigned short usCount = 0;
    short retVal = 0;

    /* Make sure we are writing to DMA PHY directly */
//    #define HWWRITE_16(OFFSET, DATA) IOWR_16DIRECT(DM9000ACUSTOM_BASE, (OFFSET)*4, DA-
TA)
    HWWRITE_16(2, 0);    //DM9000ACUSTOM_BASE, 2*4, 0);

    /* Initalize the DM9000 and the Ethernet interrupt handler */
    retVal = DM9000_init(mac_address);
    if(retVal == DMFE_SUCCESS){
        printf("DM9000_init successful\n");
    }
    else
    {
        printf("DM9000_init fail\n");
        return err_Board;
    }
    interrupt_number = 0;

    //printf("init reading %x\n", IORD_16DIRECT(DM9000ACUSTOM_BASE, 0));

    /* Register interrupt handler */
    alt_irq_register(DM9000ACUSTOM_IRQ, NULL, (void*)ethernet_interrupt_handler);

    /* Clear the payload */
    for (usCount= MAX_MSG_LENGTH-1; usCount>0; usCount--) {
        UDP_PACKET_PAYLOAD[usCount] = 0;
    }

    /* Register Recv Callback */
    gRecvCallback = pRecvCallback; /* pRecvCallback could be NULL! */
    return SUCCESS;
}

/*******************************************************
* altera_send: send data pData over usConnectionId
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int altera_send(unsigned short usConnectionId, char* pData,
                    unsigned int uiDataLen)
{
    short sRetVal = SUCCESS;
    unsigned short usCount = 0;
    unsigned int uiPacketLength = 0;
    unsigned short usIPCheckSum = 0;

    /* Set to software send mode */
    HWWRITE_16(2, 0);
    if(!pData){
        return err_BadInput; /* TBD: application err codes */
    }

    strcpy(UDP_PACKET_PAYLOAD, pData);
    UDP_PACKET_PAYLOAD[uiDataLen] = 0; /* End of data */

    /* Increment IP packet ID*/
    gIPPacketIDNum++;
    transmit_buffer[IP_PACKET_ID_OFFSET] = gIPPacketIDNum >> 8;
    transmit_buffer[IP_PACKET_ID_OFFSET + 1] = gIPPacketIDNum & 0xff;


    /* Compute IP Header Checksum */
    usIPCheckSum = IPCheckSum(transmit_buffer+IP_HEADER_OFFSET, IP_HEADER_SIZE);
```

```
    transmit_buffer[IP_HEADER_CHECKSUM_OFFSET] = usIPCheckSum >> 8;
    transmit_buffer[IP_HEADER_CHECKSUM_OFFSET+1] = usIPCheckSum & 0xff;

    /* Check IP Header Checksum */
    usIPCheckSum = IPCheckSum(transmit_buffer+IP_HEADER_OFFSET, IP_HEADER_SIZE);
    if(usIPCheckSum){
        printf("checksum fail\n");
    }
    else{
       // printf("checksum successs\n");
    }

    /* Update packet length */
    uiPacketLength = UDP_PACKET_PAYLOAD_OFFSET + uiDataLen;
    transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = uiPacketLength >> 8;
    transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = uiPacketLength & 0xff;

    /* Send UDP packet */
    if (TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET + uiDataLen +
            1)==DMFE_SUCCESS) {
        printf("\nMessage sent successfully\n");
        //printf("Bytes sent: %d\n",UDP_PACKET_PAYLOAD_OFFSET + uiDataLen - 1);
        //printf("Bytes: %s\n",UDP_PACKET_PAYLOAD);
    }
    else {
        printf("\nMessage sending failed\n");
        sRetVal = err_Board;
    }

    /* reset data */
    for (usCount=MAX_MSG_LENGTH-1; usCount>0; usCount--) {
    UDP_PACKET_PAYLOAD[usCount] = 0;
    }

    /* Set the IP CheckSum Fields to Zero */
    transmit_buffer[IP_HEADER_CHECKSUM_OFFSET + 1] = 0x00;
    transmit_buffer[IP_HEADER_CHECKSUM_OFFSET] = 0x00;

    msleep(1);

    /* Set to custom hardware receive mode */
    HWWRITE_16(2, 1);

    return sRetVal;
}

/*******************************************************
* displayMetrics: close the board
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int displayMetrics(void)
{
    unsigned char ucDatabuffer[100] = {0};
    unsigned short usCount = 0;
    unsigned int usHWTime = 0;
    unsigned int usHWSWTime = 0;

   /* Copy HW read time */
    for(usCount = 0; usCount < 2; usCount++){
        ucDatabuffer[1 - usCount] = ((HWREAD_16(4)) >> usCount*8) & 0xFF;
    }
    for(usCount = 0; usCount < 2; usCount++){
        ucDatabuffer[1 - usCount +2] = ((HWREAD_16(3)) >> usCount*8) & 0xFF;
    }

    /* Copy HW/SW read time */
    for(usCount = 0; usCount < 2; usCount++){
        ucDatabuffer[1 - usCount+4] = ((HWREAD_16(6)) >> usCount*8) & 0xFF;
    }
```

```
    for(usCount = 0; usCount < 2; usCount++){
        ucDatabuffer[1 - usCount+6] = ((HWREAD_16(5)) >> usCount*8) & 0xFF;
    }

    usHWTime = charArray4ToInt(ucDatabuffer);
    printf("HW read time      %u\n",usHWTime);
    usHWSWTime = charArray4ToInt(ucDatabuffer+4);
    printf("Overall read time %u\n",usHWSWTime);
    return SUCCESS;
}

/*******************************************************
* altera_close: close the board
*
*
* return : 0 on success, -1 on failure
*******************************************************/
int altera_close(void)
{
    /* un register the interuupt handler */
    return SUCCESS;
}
```

hfr_main.c

```
/*
 * CSEE 4840 Project
 * High Frequency Reader
 *
 * hfr_main.c: Has main function, Board init, Read/Write
 *
 * Created by : Manu
 * Modified by : Adil, Amandeep, Manu, Prabhat
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include "hfr_altera.h"
#include "hfr_config.h"

#include "basic_io.h"

#define AGGREGATERECVNUM 100


/* Count of total orders received */
unsigned int gPacketCount = 0;

/* Flag indicating to process the received orders */
int gPacketsReceivedFlag = 0;
```

```
/* Bid or Offer order */
typedef enum eStockType {
 eOffer = 0,
 eBid
}eStockType;

/* Temperory structure to copy data from the interrupt handler */
typedef struct StockData{
   unsigned char cOrderId[ODRIDSIZE+1];
   unsigned char cStock[HFRSTOCKSIZE+1];
   unsigned char cPrice[HFRPRICESIZE+1];
   unsigned char cNumShares[HFRNUMSHARESSIZE+1];
   unsigned char cBidFlag[HFRBIDOFFERSIZE+1];  /* Zero means Offer/ 1 means bid*/
}odrData;

static odrData gRecvData ;

/* Doubly Linked List to Keep Offer and Bid orders */
typedef struct marketData{
   unsigned int iOrderId;
   unsigned char cStock[HFRSTOCKSIZE+1];
   unsigned int iPrice;
   unsigned int iNumShares;
   struct marketData* pNextNode;
   struct marketData* pPreviousNode;
}marketData;

/* Head and Tail Pointer of the lists */
static marketData* pHeadNodeBid = NULL;
static marketData* pTailNodeBid = NULL;
static marketData* pHeadNodeOffer = NULL;
static marketData* pTailNodeOffer = NULL;


/*************************************************************
* charArray4ToInt
* 4 byte char array to int
* (only positive integers considered)
*
*************************************************************/
unsigned int charArray4ToInt(unsigned char* pBuffer)
{
   unsigned int iValue = 0;
   int i = 0, temp =0;
   if(NULL == pBuffer){
      return 0; // indicates error
   }
   for(i = 3; i >=0; i--){
      temp = pBuffer[i];
      iValue = (iValue | temp << (3-i)*8);
   }
```

```
    return iValue;
}

/************************************************************
* addNode()
*
*
*
************************************************************/
static void addNode(eStockType eBidFlag)
{
    marketData *pTemp = NULL;
    marketData *pCurrentNode = NULL;

    pTemp = (marketData*)malloc(sizeof(marketData));
    if(pTemp == NULL){
        printf("packetCount: %d\n", gPacketCount);
        printf("Error in add Node; probably no memory available \n");
        return;
    }

    /* Copy the values into the node */
    pTemp->iOrderId = charArray4ToInt(gRecvData.cOrderId);
    pTemp->iNumShares = charArray4ToInt(gRecvData.cNumShares);
    pTemp->iPrice = charArray4ToInt(gRecvData.cPrice);
    strcpy(pTemp->cStock,gRecvData.cStock);
    pTemp->cStock[HFRSTOCKSIZE] = 0;

#if 0 /* Check if the data received is as sent */
    if(pTemp->iPrice != 400){
      printf("Errrr Price: order id %d\n",pTemp->iOrderId);
    }
    if(pTemp->iNumShares != 500){
        printf("Errrr Num Shares: order id %d\n",pTemp->iOrderId);
    }
    if(strcmp(pTemp->cStock,"goog")){
        printf("Errrr Stock: order id %d\n",pTemp->iOrderId);
    }

#endif

    if(eBidFlag){ /* Bid List is in Descending order */
        if(pHeadNodeBid == NULL){ /* First element of the list*/
            pTemp->pPreviousNode = NULL;
            pTemp->pNextNode = NULL;
            pHeadNodeBid = pTemp;
            pTailNodeBid = pTemp;
        }
        else{
            pCurrentNode = pHeadNodeBid;
            /* Find a location to place the node */
            while((pCurrentNode) && (pCurrentNode->iPrice > pTemp->iPrice)){
```

```
            pCurrentNode = pCurrentNode->pNextNode;
        }
        if(pCurrentNode){
            if(pCurrentNode->pPreviousNode){
                pCurrentNode->pPreviousNode->pNextNode = pTemp;
                pTemp->pNextNode = pCurrentNode;
                pTemp->pPreviousNode = pCurrentNode->pPreviousNode;
                pCurrentNode->pPreviousNode = pTemp;
            }
            else{/* pTemp is the 1st node (head node) in the list now */
                pTemp->pPreviousNode = NULL;
                pTemp->pNextNode = pHeadNodeBid;
                pHeadNodeBid->pPreviousNode = pTemp;
                pHeadNodeBid = pTemp;
            }
        }
        else{ /* This is the last node in the list now */
            pTemp->pPreviousNode = pTailNodeBid;
            pTemp->pNextNode = NULL;
            pTailNodeBid->pNextNode = pTemp;
            pTailNodeBid = pTemp;
        }
    }
}
else{ /* Offer list is in ascending order */
    if(pHeadNodeOffer == NULL){ /* First element of the list*/
        pTemp->pPreviousNode = NULL;
        pTemp->pNextNode = NULL;
        pHeadNodeOffer = pTemp;
        pTailNodeOffer = pTemp;
    }
    else{
        pCurrentNode = pHeadNodeOffer;
         /* Find a location to place the node */
        while((pCurrentNode) && (pCurrentNode->iPrice < pTemp->iPrice)){
            pCurrentNode = pCurrentNode->pNextNode;
        }
        if(pCurrentNode){
            if(pCurrentNode->pPreviousNode){
                pCurrentNode->pPreviousNode->pNextNode = pTemp;
                pTemp->pNextNode = pCurrentNode;
                pTemp->pPreviousNode = pCurrentNode->pPreviousNode;
                pCurrentNode->pPreviousNode = pTemp;
            }
            else{/* pTemp is the 1st node (head node) in the list now */
                pTemp->pPreviousNode = NULL;
                pTemp->pNextNode = pHeadNodeOffer;
                pHeadNodeOffer->pPreviousNode = pTemp;
                pHeadNodeOffer = pTemp;
            }
        }
        else{ /* This is the last node (tail node) in the list now */
```

```
            pTemp->pPreviousNode = pTailNodeOffer;
            pTemp->pNextNode = NULL;
            pTailNodeOffer->pNextNode = pTemp;
            pTailNodeOffer = pTemp;
         }
      }
   }
   return;
}

/*************************************************************
* deleteNode()
*
*
*
*************************************************************/
static void deleteNode(marketData* pNode, eStockType eBidFlag)
{
   if(pNode){
      /* Create appropriate links */
      if((pNode->pPreviousNode) && (pNode->pNextNode)){
         pNode->pPreviousNode->pNextNode = pNode->pNextNode;
         pNode->pNextNode->pPreviousNode = pNode->pPreviousNode;
      }

      /* The node being deleted is the head node */
      else if (pNode->pNextNode){
         if(eBidFlag){
            pHeadNodeBid = pNode->pNextNode;
         }
         else{
            pHeadNodeOffer = pNode->pNextNode;
         }
         pNode->pNextNode->pPreviousNode = NULL;
      }

      /* The node being deleted is the tail node */
      else if (pNode->pPreviousNode) {
         if(eBidFlag){
            pTailNodeBid = pNode->pPreviousNode;
         }
         else{
            pTailNodeOffer = pNode->pPreviousNode;
         }
         pNode->pPreviousNode->pNextNode = NULL;
      }

      else { /* The only node in the list */
         if(eBidFlag){
            pHeadNodeBid = NULL;
            pTailNodeBid = NULL;
         }
```

```
            else{
               pHeadNodeOffer = NULL;
               pTailNodeOffer = NULL;
            }
         }

         /* Finally free the node */
         free(pNode);
      }
   }

/*************************************************************
* makeDeals()
*
* Makes deals based on price (Only price)
*
*************************************************************/
static void makeDeals()
{
   while((pHeadNodeBid) && (pHeadNodeOffer) && (pHeadNodeBid->iPrice >= pHeadNodeOffer->iPrice) ){
      printf("Deal made:\n BidId: %u\t OfferId: %u\n",
               pHeadNodeBid->iOrderId, pHeadNodeOffer->iOrderId);
      deleteNode(pHeadNodeBid,1);
      deleteNode(pHeadNodeOffer,0);
   }
}

/*************************************************************
* printList()
*
* Prints the Bid list or Offer List
*
*************************************************************/
static void printList(eStockType eBidFlag)
{
   int count = 0;
   marketData* pTemp = NULL;

   /* Start at the head of Bid list or Offer list*/
   if(eBidFlag){
      pTemp = pHeadNodeBid;
      printf("Printing Bid List\n");
   }
   else{
      pTemp = pHeadNodeOffer;
      printf("Printing Offer List\n");
   }

   /* Print the list until there are nodes */
   while(pTemp){
      printf("OrderId: %u\t",pTemp->iOrderId);
      printf("Stock: %s\t",pTemp->cStock);
```

```
        printf("Price: %u\t",pTemp->iPrice);
        printf("NumShares: %u\n",pTemp->iNumShares);
        count++;
        pTemp = pTemp->pNextNode;
    }
    //printf("Num packets in the list is %d\n",count);
    return;
}

/*************************************************************
* displayStockData()
*
* Displays data copied into the temporary structure
*
*************************************************************/
void displayStockData(void)
{
    printf("OrderId: %s\n",gRecvData.cOrderId);
    printf("Stock: %s\n",gRecvData.cStock);
    printf("Price: %s\n",gRecvData.cPrice);
    printf("NumShares: %s\n",gRecvData.cNumShares);
    printf("BidFlag: %s\n",gRecvData.cBidFlag);
}

/*****************************************************************************
* recvCallbackFunc()
*
*
*
*****************************************************************************/
int recvCallbackFunc(unsigned short usConnectionId, unsigned char* pData)
{
    if(!pData){
        printf("Error found in recvCallbackFunc\n");
        return -1;
    }

    /* Copy the decoded maket order data byte by byte into the structure */

    /* Data is assumed to be in the following format :
     * Order Id  : 4 bytes
     * Stock     : 4 bytes
     * Price     : 4 bytes
     * Num Shares: 4 bytes
     * Bid/Offer : 1 byte */

     /* Order Id */
     memcpy(gRecvData.cOrderId, pData, ODRIDSIZE);
     gRecvData.cOrderId[ODRIDSIZE] = 0;

     /* Price */
     memcpy(gRecvData.cPrice, pData + ODRIDSIZE, HFRPRICESIZE);
```

```
    gRecvData.cPrice[HFRPRICESIZE] = 0;

    /* Stock name */
    memcpy(gRecvData.cStock, pData + ODRIDSIZE +HFRPRICESIZE, HFRSTOCKSIZE);
    gRecvData.cStock[HFRSTOCKSIZE] = 0;

    /* Bid/ Offer Flag */
    memcpy(gRecvData.cBidFlag, pData + ODRIDSIZE + HFRSTOCKSIZE+ HFRPRICESIZE, HFRBIDOFFERSIZE);
    gRecvData.cBidFlag[HFRBIDOFFERSIZE] = 0;

    /* Number of Shares */
    memcpy(gRecvData.cNumShares, pData + ODRIDSIZE + HFRSTOCKSIZE+ HFRPRICESIZE + HFRBIDOFFERSIZE,
        HFRNUMSHARESSIZE);
    gRecvData.cNumShares[HFRNUMSHARESSIZE] = 0;

   /* Increment the packet (order) count */
   gPacketCount++;

   /* Add order to the list */
   addNode(gRecvData.cBidFlag[0]);

   /* Display the elements in the lists */
   if(gPacketCount % AGGREGATERECVNUM == 0){
      gPacketsReceivedFlag = 1;
   }
   return SUCCESS;
}


/*************************************************************
* main()
*
*
*
*************************************************************/
int main()
{
   int iCount = 0;
   marketData* pTemp = NULL;

   short sRetVal = 0;
   unsigned short usConnectionId = HFR_CONNECTIONID;

   /* Initialize the board being used */
   sRetVal = altera_init(recvCallbackFunc);
   printf("altera_init done\n");
   if(sRetVal != SUCCESS){
      printf("error in altera_init\n");
      goto ErrorExit;
   }
   /* Wait for board to initialize */
   usleep(3000000);
```

```
/* Send test packet using Software to make sure board is initialized properly */
sRetVal = altera_send(usConnectionId, "Test Packet", 11);
if(sRetVal != SUCCESS){
   printf("error in altera_send\n");
   goto ErrorExit;
}

/* Wait for packet(s) to be received */
while(1){
   if(gPacketsReceivedFlag){
      gPacketsReceivedFlag = 0; /* Reset the flag */


      printList(eOffer); /* Print the Offer List */
      printList(eBid); /* Print the bid List */

      /* Display the number of elements in Offer list */
      pTemp = pHeadNodeOffer;
      iCount = 0;
      while(pTemp){
         pTemp = pTemp->pNextNode;
         iCount++;
      }
      printf("Num elements in Offer List is %d\n",iCount);

      /* Display the number of elements in Bid list */
      pTemp = pHeadNodeBid;
      iCount = 0;
      while(pTemp){
         pTemp = pTemp->pNextNode;
         iCount++;
      }
      printf("Num elements in Bid List is %d\n",iCount);

      makeDeals();

      printList(eOffer); /* Print the Offer List */
      printList(eBid); /* Print the bid List */
      displayMetrics(); /* Show HW read time and HW+SW read time */
      /* Switch to Software mode and send ack */
      altera_send(1, "Packets Received Ack", 20);
   }
}

/* Close and exit gracefully */
sRetVal = altera_close();
if(sRetVal != SUCCESS){
   printf("error in altera_close\n");
   goto ErrorExit;
}
```

```
    return 0;

    /* Error */
    ErrorExit:
        printf("Program terminated with an error condition\n");

    return 1;
}
```

HFR_ALTERA.H

```
#ifndef HFR_ALTERA_H_
#define HFR_ALTERA_H_


typedef int(*rcvCbk)(unsigned short, unsigned char*);

/*************************************************************
* altera_init: initialize the board/ ethernet interface
*
*
* return : 0 on success, -1 on failure
*************************************************************/
int altera_init(rcvCbk pRecvCallback);

/*************************************************************
* altera_send: send data pData over usConnectionId
*
*
* return : 0 on success, -1 on failure
*************************************************************/
int altera_send(unsigned short usConnectionId, char* pData,
                unsigned int uiDataLen);



/*************************************************************
* displayMetrics: close the board
*
*
* return : 0 on success, -1 on failure
*************************************************************/
int displayMetrics(void);

/*************************************************************
* altera_close: close the board
*
*
* return : 0 on success, -1 on failure
*************************************************************/
int altera_close(void);

#endif /*HFR_ALTERA_H_*/
```

10. Reference

[1] J.A. Brogaard, “High Frequency Trading and its Impact on Market Quality,” 5th Annual Conference on Empirical Legal Studies, 2010.

[2] http://www.nasdaqtrader.com/Trader.aspx?id=ouch