Networked Air Hockey Video Game

CSEE 4840 Project Report - May 2010

Ankita Nayak

Computer Engineering Department

aan2120@columbia.edu

Aimee Sanchez

Computer Engineering Department

ams2330@columbia.edu

Vinay Sharma

Computer Engineering Department

vs2330@columbia.edu

Kapil Verma

Computer Engineering Department

kv2208@columbia.edu

Contents

1. Introduction

The networked air hockey game is an addictive Ethernet based interactive game. It is a representation of real-life air hockey game. It is a two-player game with two PC's connected over Ethernet and the game application running on the both machines. The two monitors act as playing surface for both the players and the screens are limited to help the user view the puck and paddle and also to prevent the puck and paddle from leaving the screen. Slots are present at either end of the screen, which serve as goals. There is score tab at the bottom of the screen, that keeps on updating the score as the players score goals.

The paddles, which are the controls for the players, are moved with help of the mouse. The puck starts from the center and the players can hit the puck when it enters their domain and the puck's velocity and direction are modulated according to the way the players hit it. Although, the game is played on two different screens, each player is able to view the movement of the puck and the movement of the other players paddle, due to networked property implemented in the project, where the coordinates of the puck and paddle are exchanged between the two machines.

1.1 Gameplay

The system starts at default, with the puck at the centre of the screen/table and the two paddles for each player at their respective area and the score as zero. When the game begins, the puck starts moving automatically in some arbitrary direction and the user will control the movement of his/her paddle with the help of PS/2 mouse.

The score gets updated each time the puck goes in the goal which is the scoring area. The puck starts from the center point whenever, a goal is made. The game continues until the score reaches 8, which is the maximum score (changed from 10, which was thought of previously to be maximum score). As soon as the one of the players score reaches 8, then its game over and the player with the score 8 wins the game.

The players can modulate the speed and the direction of the puck with the way they hit the puck. Thus, they can keep their opponent constantly on an edge on what to expect. The game ensures a healthy competition between players.

It goes without saying that mastering the networked air hockey game requires a lot of practice and if anyone requires coaching, then we have two very talented players in our team who are more than helpful to provide guidance on how to play the game.

1.2 Game Configuration

The configuration of the game is simple and easy to handle. One just needs to know how to move a mouse with good coordination so that one can move the paddle and of course use it to hit the puck on the screen!!

The entire graphics of the game is displayed on a VGA with resolution of 640 by 480 pixels. There is horizontal restriction in the paddle movements of the players. The scorekeeping is maintained at the bottom of the screen for both the players. The goals are on either end of the game graphics and anytime a goal is scored the puck resumes its movement from the center of the screen toward the player who scored the goal. Entire kinematics is explained in the section 2.2.1.

2. Architecture

The basic architecture of the game on one machine consists of using NIOS processor along with, VGA, PS/2 Mouse and Memory (fig. 1). The two machines are networked with the help of Ethernet connections (fig. 2).

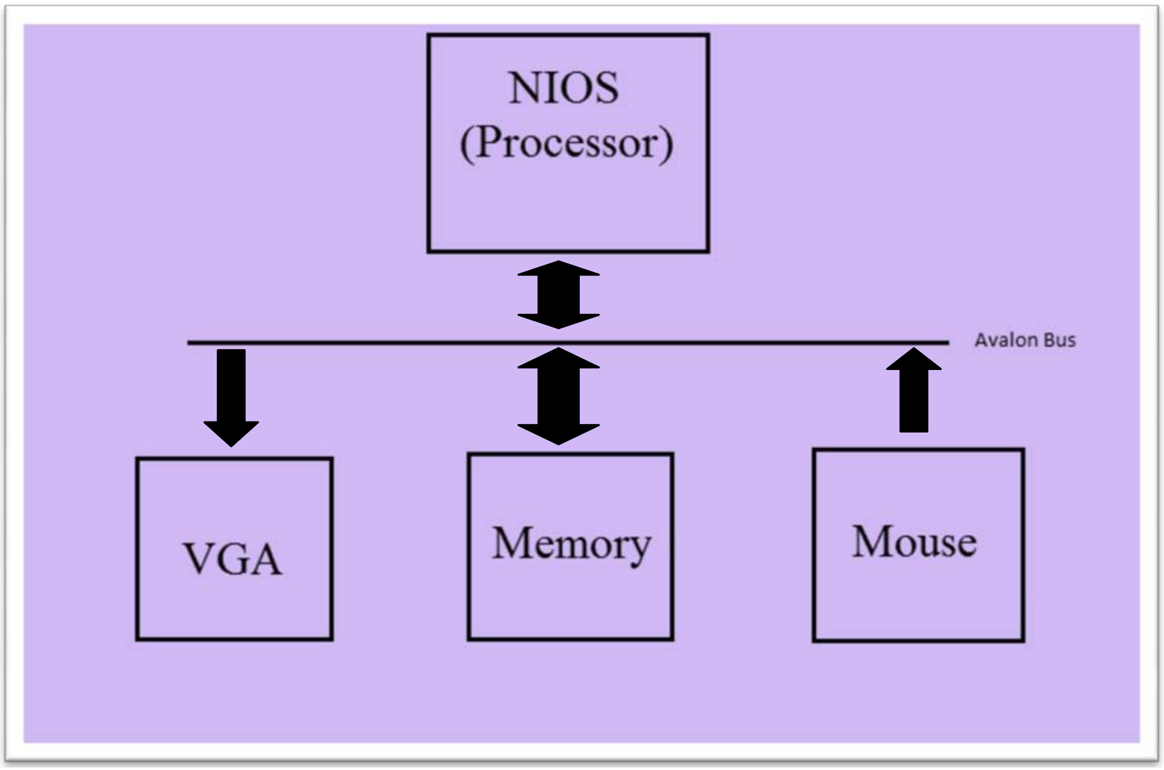

Figure 1: Basic Architecture of Single Terminal

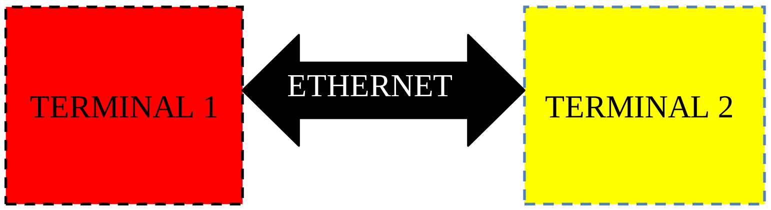

Figure 2: Terminals connected via Ethernet

The main task of modules and the processor are :-

- Memory/SRAM: The SRAM stores the data used in the software program that runs on the NIOS processor, and is accessed via a SRAM controller Avalon component.
- Ethernet: The module interconnects two systems and keeps on sending information from one system to another and vice versa.
- VGA: The VGA controller generates the whole display and keeps control over it.
- PS2/Mouse: The peripheral is used to move the paddle and is controlled by the user.

- CPU/ NIOS II Processor: This is the center of the system. All the peripheral control and communication, puck and paddle dynamics, including movement, collision and also transferring of contents from one system to another and vice versa.

We classify the architecture and implementation of the game into two categories, hardware and software :-

2.1 Hardware

The hardware architecture is broadly classified into three categories, VGA, Ethernet, and PS/2 Port.

2.1.1 VGA Controller

The configuration of the 15-pin DE 15 VGA connector is given below.

2.1.1.1 Configuration

VGA connectors and cables carry analog component RGBHV (red, green, blue, horizontal sync, vertical sync) video signals, and data. On laptop computers or other small devices, a mini-VGA port is sometimes used in place of the full-sized VGA connector.

It is also called a RGB connector, D-sub 15, mini sub D15, mini D15 connector, HD-15, or HD15 (High Density 15), DE-15, or DE15, distinguishing it from a connector of the same size but with only two rows of pins.

Pin1	RED
Pin2	Green
Pin3	Blue
Pin4	ID2
Pin5	GND
Pin6	RGND
Pin7	GGND
Pin8	BGND
Pin9	+5V
Pin10	GND
Pin11	ID0
Pin12	ID1
Pin13	Hsync
Pin14	Vsync
Pin15	ID3

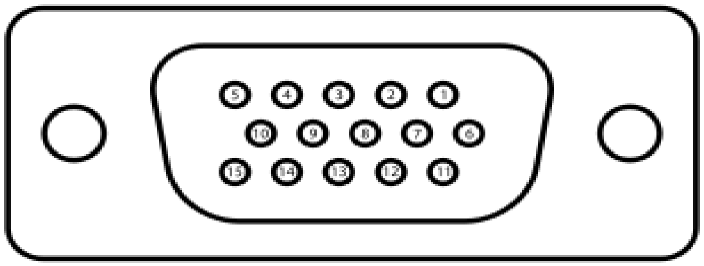

Figure 3: VGA Connector

2.1.1.2 Display

The monitor is the screen visible to the user, which represents the table and the playing area, which is blue in color. The players are able to move the paddle (white circle) back and forth, but the movement is approximately to the quarter-length of the table (denoted by the white solid line) and not the half-length of table (denoted by the white solid line). The restriction is implemented in software.

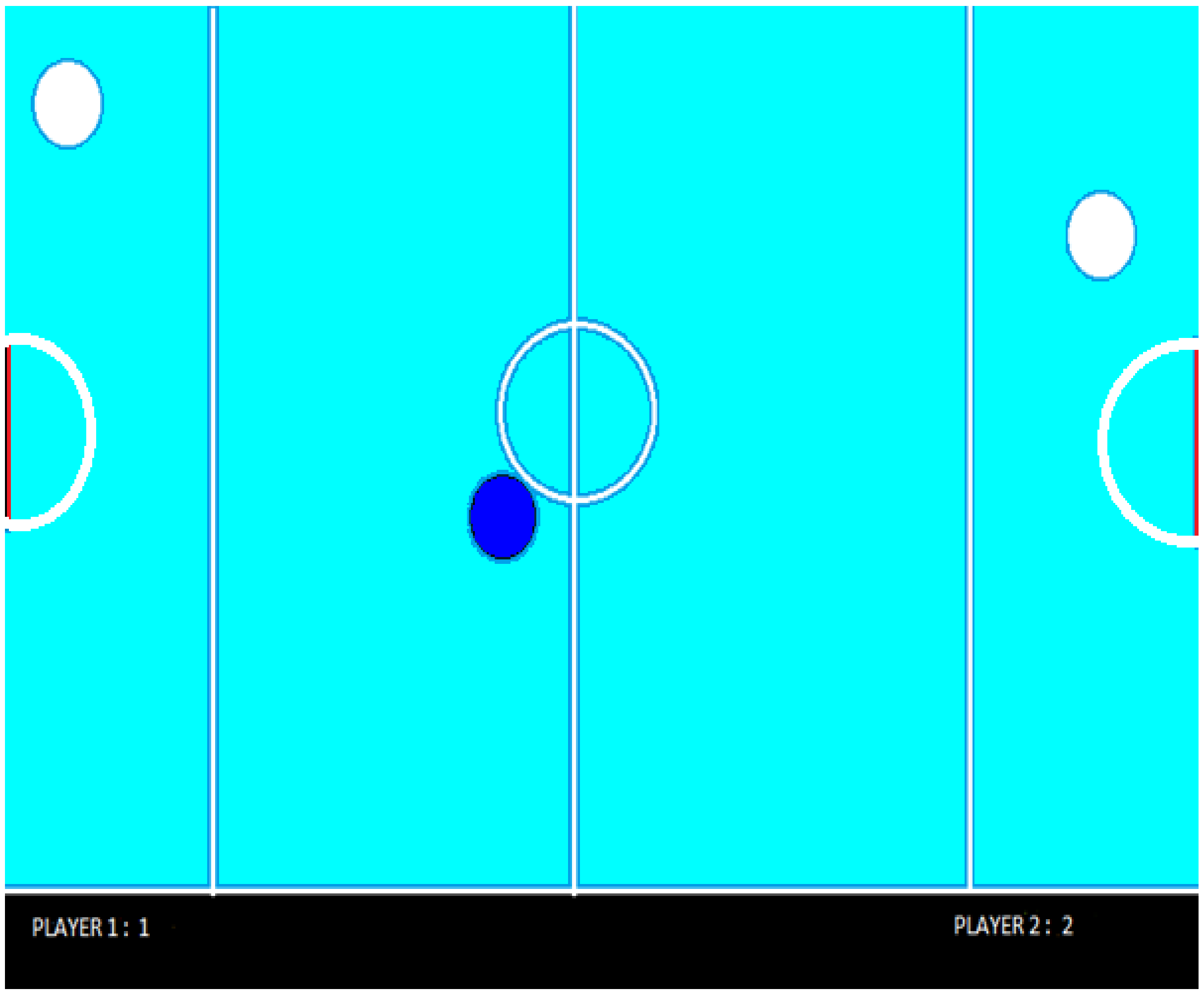

Figure 4: GUI

The puck (blue circle) is placed in the centre of the enveloped in white ring. The red line represents the length of the goal, and for extra help in display of the goal area, a half-white circle is drawn with the diameter as goal length. This helps, in more clear reception of the goal area.

Below the playing area, a score tab keeps on displaying the current score and if a goal is scored, then the score is updated in the score tab of that respected player, which is done by bit mapping. This helps the players to check the current score and their position, whether they are leading or lagging behind.

All the commands for display are written in the vga raster VHDL file. The methods for mapping the whole display area are present in that file. The circle generation is done with the help of distance and circle equations and look up table to figure out the distance.

2.1.1.3 Timing

Raster graphics are resolution dependent. They cannot scale up to an arbitrary resolution without loss of apparent quality. This property contrasts with the capabilities of vector graphics, which easily scale up to the quality of the device rendering them. Raster graphics deal more practically than vector graphics with photographs and photo-realistic images, while vector graphics often serve better for typesetting or for graphic design.

The details used for timing for raster are as below:

640 × 480 Resolution

25.175 MHz Dot Clock

31.469 kHz Line Frequency

59.94 Hz Field Frequency

pixels	**role**
8	Front Porch
96	Horizontal Sync
40	Back Porch
8	Left border
640	Active
8	Right border
800	total per line

lines	**role**
2	Front Porch
2	Vertical Sync
25	Back Porch
8	Top Border
480	Active
8	Bottom Border
525	total per field

Active-low Horizontal and Vertical sync signals.

2.1.2 PS/2 Port

The player can control the movement of their paddle with the help of PS/2 Mouse. The speed of the paddle varies in accordance to the speed of the movement of the mouse. A component in SOPC is created for the PS/2 port. The component communicates between the ps/2 port and the FPGA board.

The Altera PS2 verilog files are added in the system to make the game compatible with the PS2 port. Thereby, it ensures that there is hardware connection between the board and the component added at the PS2 port and that connection is exploited by the hardware to read and write the data from the PS2 port.

2.1.3 Ethernet Controller

An essential aspect of the architecture is the networking component. Effective communication through the network is essential for effective gamplay. As previously mentioned, the players interact through an Ethernet connection. Packets are sent through a direct port-to-port connection.

2.1.3.1 The DM9000A Device

As part of the networking interface, we use the Altera DE2 board's DM9000A chip, which is a fully integrated Ethernet MAC controller that includes a fast 10/100 Mbps transceiver. We have used software programs provided by the professor to interact with the device. Additionally, we made modifications to the provided routine to handle Ethernet generated interrupts, which will be discussed later.

2.2 Software

The software of the system is categorized in three categories, Kinematics of Game, Ethernet implementation, PS/2 Mouse implementation.

2.2.1 Kinematics of Game

Modeling the interactions and movements of the puck and paddle will be conducted in software. We are assuming the interaction between the paddle and the puck involves elastic collisions. We are neglecting the effect of friction of the table on the puck. If the puck hits any of the right or the left sides, the direction of the puck needs to be reversed in X direction. Similarly if the puck hits any of the top or bottom sides, the direction needs to be reversed in Y direction.

Simulating the physics enviornment involves two processes:

- Collision handling
- Velocity and direction modulation of the puck

2.2.1.1 Collision Handling

Collision is said to have occured when the distance between the centres of the two circles is equal to the sum of the two radii. The centre of the paddle is given by (user_x, user_y) and the centre of the puck is (puck_x, puck_y) and both the balls have equal radius R1. Therefore the distance between the centres of the two balls is given by:

d = sqrt (((puck_x - user_x) * (puck_x - user_x)) + ((puck_y - user_y) *(puck_y - user_y)))

Then the condition for collision occurence will be

$$d = 2R1$$

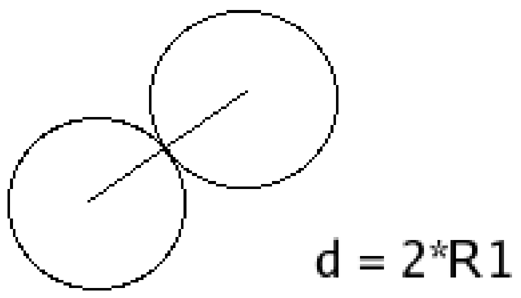

Fig. 5: Collision Handling

However there is a finite time step in the simulation calculation, so most often we would not be able to find the above condition i.e. the exact time when the ball collision really occurred. So, we need to check if d <= 2R1. Then we know that collision should have occurred.

2.2.1.2 Physics Simulation

For simulating the physics of the game, we need to know the velocity of the paddle which is a function of the rate of movement of the mouse. From the PS2 protocol, the x and y-co-ordinates of the previous location of the mouse is obtained. Thus velocity in a particular direction is a function of difference in the previous position and the new position. The magnitude of this difference determines the speed while the sign determines the direction of the movement of the paddle.

Let m_puck and m_paddle be the mass of the puck and paddle respectively and u_puck and u_paddle be the initial speed of the puck and the paddle respectively and v_puck and v_paddle be the speed of the puck and paddle after collision respectively. According to our dynamics, v_paddle will be zero.

Thus assuming elastic collision between the puck and the paddle, at the moment of collision the momentum conservation can be written as:

m_puck * u_puck + m_paddle * u_paddle= m_puck * v_puck

So, if the player wishes to hit the puck with a higher force, it is formulated as an increase in velocity of the mouse at the time of collision.

To simulate this physics in software the following strategy was applied.

Assume the following instance of collision:

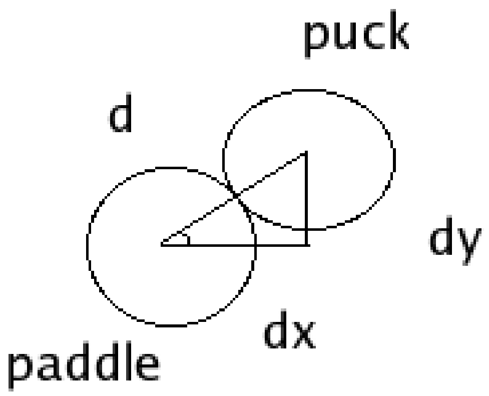

Fig. 6: Calculation of Deflection Velocities

To calculate the velocities it is necessary to obtain the angle at which the collision happens. The ratio of dx/dy gives the ratio of the velocity of the puck in x direction and y direction respectively. Thus the value (dx/d) gives the cosine of the angle which determines the speed of the puck in x direction while (dy/d) gives the sine of the angle which determines the speed of the puck in y direction. Thus the tan of the angle (dy/dx) determines the angle of deflection of the ball. These factors are used to modulate the velocity and direction of the puck after deflection. These deflection velocities also depend on the speed and direction of the mouse at that instant. Thus the net speed of the paddle factored by a value of (dx/d) and (dy/d) gives the final speed of the puck in X direction and Y direction respectively.

2.2.2 Ethernet Implementation

During communication, players are sending packets back and forth through a network. The systems are connected in a direct point-to-point configuration through a Cat5 Ethernet cable. IP protocol is used to construct the packets and the bytes are set with corresponding values.

2.2.2.1 UDP Protocol

The total length of a packet is 106 bytes. As part of the protocol, the packet includes 14 bytes of Ethernet MAC header, 20 bytes of IP header, 8 bytes for UDP header, and 64 bytes for the payload. We do not implement checksum calculation therefore these bytes are set to zero.

2.2.2.1.2 Sending IP Packets

Sending packets is very straightforward. For transmission, the application makes a call to TransmitPacket(...) routine. This routine writes appropriate values into registers and waits until transmission is done and clears the NSR (network status register). Packets in both terminals are sent in every iteration of the inner for(;;) loop in the application. The packet sent by the master terminal includes puck and player one paddle coordinates determined by the mouse movement on the local system. The subordinate sends only player two paddle coordinates also determined by the movement of the mouse on the local system.

2.2.2.1.3 Receiving Packets and Ethernet Interrupt Controller

When a packet is received, the system triggers an interrupt. In the application we have used Altera's alt_irq_register() function, which sets up a callback to our interrupt handler routing named ethernet_interrupt_handler(). This method issues a call to ReceivePacket(...) to verify that the received packet is “good” and writes the data into a parameter. Then the interrupt handler processes the data packet received and if it is “good” then the coordinates are immediately written to the hardware for display . The master terminal receives the coordinates of player two's paddle and the subordinate on the opposite end receives the positions of the puck and player one's paddle.

2.2.3 PS/2 Mouse Implementation

2.2.3.1 PS/2 Mouse Movement Data Packet

The standard PS/2 mouse interface supports the following inputs: X (right/left) movement, Y (up/down) movement, left button, middle button, and right button. The mouse periodically reads these inputs and updates various counters and flags to reflect movement and button states. The standard PS/2 mouse sends movement and button information to the host using the following 3-byte packet (Figure 3):

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X movement							
Byte 3	Y movement							

Figure 4: Mouse Protocol

The values of movement are 9-bit 2's complement integers, where the most significant bit appears as a "sign" bit in byte 1 of the movement data packet. Their value represents the mouse's offset relative to its position when the previous packet was sent, in units determined by the current resolution. The range of values that can be expressed is -255 to +255. If this range is exceeded, the appropriate overflow bit is set. Using these X and Y coordinates we will be controlling the movement of the paddles. However, there needs to be a horizontal limitation to the movement of the paddle.

2.2.3.2 PS/2 Mouse Software

Interfacing the PS2 mouse peripheral was a challenge in this project. We developed the ps2_mouse.c file entirely. The Altera alt_up_ps2_port.c file was used to get the data that the ps2 mouse writes to the data register. Any commands to the PS2 mouse are also interfaced through the alt_up_ps2_port.c by using the write function that writes to the ps2 port. The ps2_mouse.c file also handles the direction of the paddle and its position.

The following settings were made to the PS2 mouse when used in the stream mode:

1- Mouse Reset
2- Sample Rate: 200
3- Resolution 4counts/mm
4- Scaling 2:1

The directions of the paddle were handled based on the sign bits in the first byte of the ps2 mouse protocol. When either x or y sign bit is set, two’s complement value of the x movement data or y movement data is taken respectively.

2.3 SOPC

The SOPC builder has the following components created in it:-

- CPU-NIOS Processor
- SRAM-Memory
- VGA-Vga Raster component
- JTAG_UART
- DM9000A-Ethernet Component
- PS2-Altera PS2 port component

The JTAG_UART is the highest interrupt priority request line and after that, the PS2 is given the interrupt priority and then comes the Ethernet.

3. Task Division

Our project was more of group work than individual effort. Brain storming in the group on various design implementations to be made, helped us get rid of some of the pitfalls that we may have run into individually.

However, if still division needs to be done, then broadly the contributions were as follows:

Kapil	:	Hardware Implementation, Ethernet
Vinay	:	PS2 mouse, Physics simulation
Aimee	:	Ethernet, Circle Generation
Ankita	:	Physics Simulation, PS2 mouse

4. Lessons Learned

- Important to understand the details of each component
- Make use each member’s individual strengths
- UNITED WE STAND
- Always leave space for adjustments or additions

- Learned to work in distracting environment.

- Difficult to integrate hardware and software.

5. Advice for Future Students

For an onerous project class like Embedded Systems, where team play matters a lot, a right balance of collaboration and individual efforts is very necessary. Also for projects involving integration of many components, as in our case, where we worked on Ethernet, mouse and VGA, understanding the details of every component is very necessary.

Finally, it is very necessary to keep the backup of your latest working code, because one may never know when and how things could go wrong!

6. Acknowledgement

We would like to thank Prof. Edwards for enhancing our project specifications which added more challenge to it and giving important pointers at every stage. We are also grateful to the Teaching Assistants Baolin Shao and Scott Schuff who helped us sail through the hurdles we faced during our project implementation.

7. References

- http://www.cs.columbia.edu/~sedwards/classes/2010/4840/Davicom-DM9000A-Application-Notes.pdf
- http://www.cs.columbia.edu/~sedwards/classes/2010/4840/DE2_UserManual.pdf
- http://www.phy.ntnu.edu.tw/ntnujava/index.php?topic=4
- http://en.wikipedia.org

8. Source Code

8.1 VHDL Code

Main File- Top entity

8.1.1 Lab3_Vga.vhd

```
--
-- DE2 top-level module that includes the simple VGA raster generator
-- Team NAHVG
--

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

entity lab3_vga 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 : inout 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 std_logic_vector(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 lab3_vga;

architecture datapath of lab3_vga is

  signal reset : std_logic ;
  signal clk25 : std_logic := '0';
begin

  reset <= '1';

      process(CLOCK_50)
      begin
      if rising_edge(CLOCK_50) then
      clk25 <=  not clk25;
      end if;
      end process;
      ENET_CLK <= clk25;

AH1: entity work.air_hockey port map (
    reset_n => reset,
    clk => CLOCK_50,
--------------------------------SRAM--------------------------------------
-
    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,
---------------------------------Ethernet----------------------------------
--
    ENET_CMD_from_the_DM9000A => ENET_CMD,
    ENET_CS_N_from_the_DM9000A => ENET_CS_N,
    ENET_DATA_to_and_from_the_DM9000A => ENET_DATA,
    ENET_INT_to_the_DM9000A => ENET_INT,
    ENET_RD_N_from_the_DM9000A => ENET_RD_N,
    ENET_RST_N_from_the_DM9000A => ENET_RST_N,
    ENET_WR_N_from_the_DM9000A => ENET_WR_N,
---------------------------------PS2 Port----------------------------------
--
    PS2_CLK_to_and_from_the_ps2 => PS2_CLK,
    PS2_DAT_to_and_from_the_ps2 => PS2_DAT,
---------------------------------VGA---------------------------------------
--
    VGA_CLK_from_the_vga => VGA_CLK,
    VGA_HS_from_the_vga => VGA_HS,
    VGA_VS_from_the_vga => VGA_VS,
    VGA_BLANK_from_the_vga => VGA_BLANK,
    VGA_SYNC_from_the_vga => VGA_SYNC,
    VGA_R_from_the_vga => VGA_R,
    VGA_G_from_the_vga => VGA_G,
    VGA_B_from_the_vga => VGA_B
  );

  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 <= '1';
--  ENET_CLK <= '0';

  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 => 'Z');
  AUD_ADCLRCK <= 'Z';
  AUD_DACLRCK <= 'Z';
  AUD_BCLK    <= 'Z';
  GPIO_0      <= (others => 'Z');
  GPIO_1      <= (others => 'Z');

end datapath;
```

8.1.2 de2_vga_raster.vhd

```
---------------------------------------------------------------------------
-----
--
-- Simple VGA raster display
-- Team NAHVG
---------------------------------------------------------------------------
-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is

  port (
    reset_n : in std_logic;
    clk   : in std_logic;
---------------------------------------------------------------------------
------

    read : in std_logic;
    write : in std_logic;
    chipselect : in std_logic;
    address : in unsigned(4 downto 0);
    readdata : out unsigned(15 downto 0);
    writedata : in unsigned(15 downto 0);

    VGA_CLK,                           -- Clock       -- Should be 25.125
MHz
    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]
    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is

  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  constant RECTANGLE_HSTART : integer := 100;
  constant RECTANGLE_HEND   : integer := 540;
  constant RECTANGLE_VSTART : integer := 100;
  constant RECTANGLE_VEND   : integer := 380;

  -- Signals for the video controller
  signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;

  signal vga_hblank, vga_hsync,
  vga_vblank, vga_vsync : std_logic;  -- Sync. signals

  constant goal_upper_slot         :  integer := 252;
  constant goal_lower_slot         :  integer := 212;
  signal user_cxstart              :  unsigned(9 downto 0) :=
"1000001101";                 --525
  signal user_cystart              :  unsigned(9 downto 0) :=
"0001110000";                 --112
  signal puck_cxstart              :  unsigned(9 downto 0) :=
"0101000000";                 --320
  signal puck_cystart            :  unsigned(9 downto 0) := "0011101000";
          --232
  constant radius                  :  integer := 10;
  constant centre_radius                :  integer := 20;
  constant small_center_radius     :  integer := 18;
  constant gradius_inner                :  integer := 18;
  constant gradius                           :  integer := 20;

  constant sqr_radius               :  integer := 100;
  constant sqr_centre_radius       :  integer := 400;
  constant sqr_small_center_radius:  integer := 324;
  constant sqr_gradius_inner       :  integer := 324;
  constant sqr_gradius                       :  integer := 400;

  signal ethernet_cxstart          :  unsigned(9 downto 0) :=
"0001111111";                 --127
  signal ethernet_cystart          :  unsigned(9 downto 0) :=
"0001111111";                 --127
  constant center_cxstart          :  unsigned(9 downto 0) :=
"0101000000";                 --320
  constant center_cystart          :  unsigned(9 downto 0) :=
"0011101000";                 --232
```

```
  constant center_gxstart_l       :  unsigned(9 downto 0) :=
"0000000000";               --0
  constant center_gxstart_r       :  unsigned(9 downto 0) :=
"1010000000";               --640
  constant center_gystart         :  unsigned(9 downto 0) :=
"0011101000";               --232
  signal vline,goal,goal_outer_circle, goal_inner_circle,
  circle,ethernet_circle,center_circle,small_center_circle,puck_circle,
  score_player1,score_player2, sc_player_1, sc_player_2 : std_logic :=
'0';
  signal vline_off                      :  integer := 0;
  signal clk25 : std_logic :='0';
  signal reset : std_logic;
  signal tmp : unsigned (9 downto 0) := (others => '0');

  signal player_1 , player_2 : integer :=0;
  type player1 is array ( 0 to 7) of unsigned ( 0 to 50);
  constant crange : player1 := (
"011110001000000011110001000101111100111111001100000",
"010001001000000010000100100010100000010000100110000 0",
"010001001000000010000100100010100000010000100110011 0",
"011110001000000010000100011100100000011111100110000 0",
"010000001000000011111100001000111000011000000110000 0",
"010000001000000010000100001000100000010100000110011 0",
"010000001000000010000100001000100000010010000110000 0",
"010000000111110010000100001000111110010001100110000 0");

type player2 is array ( 0 to 7) of unsigned ( 0 to 54);
constant crange1 : player2 := (
"0111100010000000111100010001001111100111111001111100000",
"0100010010000000100001001000100100000010000100000110000 0",
"0100010010000000100001001000100100000010000100000110011 0",
"0111100010000000100001000111000100000011111100111110000 0",
"0100000010000000111111000010000111000011000000110000000 0",
"0100000010000000100001000010000100000010100000110000011 0",
"0100000010000000100001000010000100000010010000110000000 0",
"0100000001111100100001000010000111110010001100111110000 0");

type scoredecode is array ( 0 to 7) of unsigned ( 0 to 71);
constant crange2 : scoredecode := (
"001111100010000000111110001111100010001000111110001111100011111000111110 0"
,
"001000100010000000000100000000100010001000100000001000000000000100010001 00"
,
"001000100010000000000100000000100010001000100000001000000000000100010001 00"
,
"001000100010000000111110000000100011111000111110001000000000000100001000100"
,
"001000100010000000100000001111100000000010000000100011111000000100000111 1100"
,
"001000100010000000100000000000100000000100000000100010001000000100000010 00100"
,
"001000100010000000100000000000100000000100000000100010001000000100000010 00100"
,
"00111111000100000011111100011111100000000100011111100011111100000010000001111100"
);

-- Lookup table for squares of integers 0 to 20
 type mem is array (0 to 20) of integer;
 constant table : mem := (
0,1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400);

begin

reset <= not reset_n;

--------------------------------Peripheral--------------------------------
----------------------
process (clk)
  begin
    if rising_edge(clk) then
      clk25 <= not clk25;
    end if;
  end process;

process(clk)
  begin
if rising_edge(clk) then    --Make sure every logic is on rising edge of
clock
 if reset_n='1' then         --To synchronise it
  if chipselect = '1' then
    if write ='1' then
      if address = "00000" then
       user_cxstart <= writedata(9 downto 0);
      elsif address = "00001" then
       user_cystart <= writedata(9 downto 0);
        elsif address = "00010" then
       ethernet_cxstart <= writedata(9 downto 0);
      elsif address = "00011" then
       ethernet_cystart <= writedata(9 downto 0);
      elsif address = "00100" then
       puck_cxstart <= writedata(9 downto 0);
      elsif address = "00101" then
      puck_cystart <= writedata(9 downto 0);
      elsif address <= "00111" then
      player_1 <=to_integer(writedata(9 downto 0));
      elsif address <= "01000" then
      player_2 <=to_integer(writedata(9 downto 0));
        end if;
    end if;
  end if;
 end if;
end if;
end process;



  -- Horizontal and vertical counters
```

```
HCounter : process (clk25)
begin
  if rising_edge(clk25) then
    if reset = '1' then
      Hcount <= (others => '0');
    elsif EndOfLine = '1' then
      Hcount <= (others => '0');
    else
      Hcount <= Hcount + 1;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk25)
begin
  if rising_edge(clk25) then
    if reset = '1' then
      Vcount <= (others => '0');
    elsif EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk25)
begin
  if rising_edge(clk25) then
    if reset = '1' or EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk25)
begin
  if rising_edge(clk25) then
    if reset = '1' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
```

```
        vga_hblank <= '1';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_vsync <= '1';
      elsif EndOfLine ='1' then
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (clk25)
  begin
    if rising_edge(clk25) then
      if reset = '1' then
        vga_vblank <= '1';
      elsif EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;
---------------------------------------Vertical  Line gen code------------
--------------------
  Linegen : process(clk25)
  begin
     if rising_edge (clk25) then
           if reset = '1' then
                 vline <= '0';
           elsif ((140 = Hcount-HBACK_PORCH-HSYNC)or(500 = Hcount-
HBACK_PORCH-HSYNC)or(320 = Hcount-HBACK_PORCH-HSYNC))
                       and Vcount<=464+VSYNC+VBACK_PORCH then

                       vline <= '1';
           else
                 vline <= '0';
           end if;
     end if;
 end process;

----------------------------------------------------------------------------
--------------------
```

```
--------------------------------------User Control------------------------
--------------------
  CircleGen1 : process(clk25)
  begin
  if rising_edge(clk25) then
            if reset = '1' then
                  circle <= '0';
            elsif Hcount>= user_cxstart+HBACK_PORCH+HSYNC and Vcount >=
user_cystart+VBACK_PORCH+VSYNC
                  and (Hcount-user_cxstart-HBACK_PORCH-HSYNC)<=radius and
(Vcount-user_cystart-VBACK_PORCH-VSYNC) <=radius then --4th quadrant
                  if ((table(TO_INTEGER(Hcount-user_cxstart)-HBACK_PORCH-
HSYNC) + table(TO_INTEGER(Vcount-user_cystart)-VBACK_PORCH-VSYNC))   <=
sqr_radius ) then
                        circle <= '1';
                   else
                        circle <= '0';
                   end if;
            elsif Hcount>= user_cxstart+HBACK_PORCH+HSYNC and Vcount <=
user_cystart+VBACK_PORCH+VSYNC
                        and (Hcount-user_cxstart-HBACK_PORCH-HSYNC)<=radius
and (user_cystart-Vcount+VBACK_PORCH+VSYNC) <=radius then  --1st quadrant
                  if ((table(TO_INTEGER(Hcount-user_cxstart)-HBACK_PORCH-
HSYNC)  + table(TO_INTEGER(user_cystart-Vcount)+VBACK_PORCH+VSYNC))<=
sqr_radius) then
                        circle <= '1';
                   else
                        circle <= '0';
                   end if;
            elsif Hcount<= user_cxstart+HBACK_PORCH+HSYNC and Vcount <=
user_cystart+VBACK_PORCH+VSYNC
                  and (user_cxstart-Hcount+HBACK_PORCH+HSYNC)<=radius and
(user_cystart-Vcount+VBACK_PORCH+VSYNC) <=radius then          --2nd
quadrant
                  if ((table(TO_INTEGER(user_cxstart-
Hcount)+HBACK_PORCH+HSYNC)  + table(TO_INTEGER(user_cystart-
Vcount)+VBACK_PORCH+VSYNC))  <= sqr_radius) then
                        circle <= '1';
                   else
                        circle<='0';
                   end if;
            elsif Hcount<= user_cxstart+HBACK_PORCH+HSYNC and Vcount >=
user_cystart+VBACK_PORCH+VSYNC
                  and (user_cxstart-Hcount+HBACK_PORCH+HSYNC)<=radius and
(Vcount-user_cystart-VBACK_PORCH-VSYNC) <=radius then     --3rd quadrant
                  if ((table(TO_INTEGER(user_cxstart-
Hcount)+HBACK_PORCH+HSYNC)  + table(TO_INTEGER(Vcount-user_cystart)-
VBACK_PORCH-VSYNC)) <= sqr_radius) then
                        circle <= '1';
                   else
                        circle<='0';
                   end if;
            else
                        circle <= '0';
            end if;
        end if;
    end process;

---------------------------------------------------------------------------
----------------------

---------------------------------------Second User Control(Ethernet)------
----------------------
    CircleGen2 : process(clk25)
    begin
    if rising_edge(clk25) then
            if reset = '1' then
                  ethernet_circle <= '0';
            elsif Hcount>= ethernet_cxstart+HBACK_PORCH+HSYNC and Vcount
>= ethernet_cystart+VBACK_PORCH+VSYNC
                  and (Hcount-ethernet_cxstart-HBACK_PORCH-HSYNC)<=radius
and (Vcount-ethernet_cystart-VBACK_PORCH-VSYNC) <=radius then --4th
quadrant
                  if (table(TO_INTEGER(Hcount-ethernet_cxstart)-
HBACK_PORCH-HSYNC)+ table(TO_INTEGER(Vcount-ethernet_cystart)-VBACK_PORCH-
VSYNC) <= sqr_radius) then
                        ethernet_circle <= '1';
                   else
                        ethernet_circle <= '0';
                   end if;
            elsif Hcount>= ethernet_cxstart+HBACK_PORCH+HSYNC and Vcount
<= ethernet_cystart+VBACK_PORCH+VSYNC
                  and (Hcount-ethernet_cxstart-HBACK_PORCH-HSYNC)<=radius
and (ethernet_cystart+VBACK_PORCH+VSYNC-Vcount) <=radius then  --1st
quadrant
                  if (table(TO_INTEGER(Hcount-ethernet_cxstart)-
HBACK_PORCH-HSYNC) + table(TO_INTEGER(ethernet_cystart-
Vcount)+VBACK_PORCH+VSYNC)   <= sqr_radius) then
                        ethernet_circle <= '1';
                   else
                        ethernet_circle <= '0';
                   end if;
            elsif Hcount<= ethernet_cxstart+HBACK_PORCH+HSYNC   and Vcount
<= ethernet_cystart+VBACK_PORCH+VSYNC
                  and (ethernet_cxstart+HBACK_PORCH+HSYNC-Hcount)<=radius
and (ethernet_cystart+VBACK_PORCH+VSYNC-Vcount) <=radius then       --2nd
quadrant
                  if (table(TO_INTEGER(ethernet_cxstart-
Hcount)+HBACK_PORCH+HSYNC)  + table(TO_INTEGER(ethernet_cystart-
Vcount)+VBACK_PORCH+VSYNC)   <= sqr_radius) then
                        ethernet_circle <= '1';
                   else
                        ethernet_circle<='0';
                   end if;
            elsif Hcount<= ethernet_cxstart+HBACK_PORCH+HSYNC   and Vcount
>= ethernet_cystart+VBACK_PORCH+VSYNC
```

```
                 and (ethernet_cxstart+HBACK_PORCH+HSYNC-Hcount)<=radius
and (Vcount-ethernet_cystart-VBACK_PORCH-VSYNC) <=radius then   --3rd
quadrant
                 if (table(TO_INTEGER(ethernet_cxstart-
Hcount)+HBACK_PORCH+HSYNC)+table(TO_INTEGER(Vcount-ethernet_cystart)-
VBACK_PORCH-VSYNC) <= sqr_radius) then
                       ethernet_circle <= '1';
                  else
                       ethernet_circle<='0';
                  end if;
           else
                       ethernet_circle <= '0';
           end if;
      end if;
  end process;

---------------------------------------------------------------------------
-----------------------
------------------------------center circle generation--------------------
------------------------

  CircleGen3 : process(clk25)
  begin
  if rising_edge(clk25) then
           if reset = '1' then
                 center_circle <= '0';
           elsif Hcount>= center_cxstart+HBACK_PORCH+HSYNC and Vcount >=
center_cystart+VBACK_PORCH+VSYNC then --4th quadrant
                 if (Hcount-center_cxstart-HBACK_PORCH-HSYNC) <=
centre_radius and (Vcount-center_cystart-VBACK_PORCH-VSYNC)
<=centre_radius
                       and (table(TO_INTEGER(Hcount-center_cxstart)-
HBACK_PORCH-HSYNC) +table(TO_INTEGER(Vcount-center_cystart)-VBACK_PORCH-
VSYNC)<= sqr_centre_radius) then
                       center_circle <= '1';
                  else
                       center_circle <= '0';
                  end if;
           elsif Hcount>= center_cxstart+HBACK_PORCH+HSYNC and Vcount <=
center_cystart+VBACK_PORCH+VSYNC then  --1st quadrant
                 if (Hcount-center_cxstart-HBACK_PORCH-HSYNC) <=
centre_radius and (center_cystart+VBACK_PORCH+VSYNC-Vcount)
<=centre_radius
                       and (table(TO_INTEGER(Hcount-center_cxstart)-
HBACK_PORCH-HSYNC) +table(TO_INTEGER(center_cystart-
Vcount)+VBACK_PORCH+VSYNC) <= sqr_centre_radius) then
                       center_circle <= '1';
                  else
                       center_circle <= '0';
                  end if;
           elsif Hcount<= center_cxstart+HBACK_PORCH+HSYNC    and Vcount
<= center_cystart+VBACK_PORCH+VSYNC then           --2nd quadrant
```

```
                  if (center_cxstart+HBACK_PORCH+HSYNC-Hcount) <=
centre_radius and (center_cystart+VBACK_PORCH+VSYNC-Vcount)
<=centre_radius
                        and (table(TO_INTEGER(center_cxstart-
Hcount)+HBACK_PORCH+HSYNC) + table(TO_INTEGER(center_cystart-
Vcount)+VBACK_PORCH+VSYNC) <= sqr_centre_radius) then
                        center_circle <= '1';
                   else
                        center_circle<='0';
                   end if;
            elsif Hcount<= center_cxstart+HBACK_PORCH+HSYNC     and Vcount
>= center_cystart+VBACK_PORCH+VSYNC   then     --3rd quadrant
                  if (center_cxstart+HBACK_PORCH+HSYNC-Hcount) <=
centre_radius and (Vcount-center_cystart-VBACK_PORCH-VSYNC)
<=centre_radius
                        and (table(TO_INTEGER(center_cxstart-
Hcount)+HBACK_PORCH+HSYNC) + table(TO_INTEGER(Vcount-center_cystart)-
VBACK_PORCH-VSYNC) <= sqr_centre_radius) then
                        center_circle <= '1';
                   else
                        center_circle<='0';
                   end if;
            else
                        center_circle <= '0';
            end if;
      end if;
  end process;
--------------------------------------------------------------------------
-------------------------

---------------------------------------small center circle generation-----
-------------------------
CircleGen4 : process(clk25)
  begin
  if rising_edge(clk25) then
            if reset = '1' then
                  small_center_circle <= '0';
            elsif Hcount>= center_cxstart+HBACK_PORCH+HSYNC and Vcount >=
center_cystart+VBACK_PORCH+VSYNC then --4th quadrant
                  if (Hcount-center_cxstart-HBACK_PORCH-HSYNC) <=
small_center_radius and (Vcount-center_cystart-VBACK_PORCH-VSYNC)
<=small_center_radius
                        and (table(TO_INTEGER(Hcount-center_cxstart)-
HBACK_PORCH-HSYNC)+table(TO_INTEGER(Vcount-center_cystart)-VBACK_PORCH-
VSYNC)<= sqr_small_center_radius) then
                        small_center_circle <= '1';
                   else
                        small_center_circle <= '0';
                   end if;
            elsif Hcount>= center_cxstart+HBACK_PORCH+HSYNC and Vcount <=
center_cystart+VBACK_PORCH+VSYNC  then  --1st quadrant
                  if (Hcount-center_cxstart-HBACK_PORCH-HSYNC) <=
small_center_radius and (center_cystart+VBACK_PORCH+VSYNC-Vcount)
<=small_center_radius
```

```
                        and (table(TO_INTEGER(Hcount-center_cxstart)-
HBACK_PORCH-HSYNC)+table(TO_INTEGER(center_cystart-
Vcount)+VBACK_PORCH+VSYNC) <= sqr_small_center_radius) then
                        small_center_circle <= '1';
                   else
                        small_center_circle <= '0';
                   end if;
            elsif Hcount<= center_cxstart+HBACK_PORCH+HSYNC     and Vcount
<= center_cystart+VBACK_PORCH+VSYNC then            --2nd quadrant
                  if (center_cxstart+HBACK_PORCH+HSYNC-Hcount)<=
small_center_radius and (center_cystart+VBACK_PORCH+VSYNC-Vcount)
<=small_center_radius
                        and (table(TO_INTEGER(center_cxstart-
Hcount)+HBACK_PORCH+HSYNC)+table(TO_INTEGER(center_cystart-
Vcount)+VBACK_PORCH+VSYNC) <= sqr_small_center_radius) then
                        small_center_circle <= '1';
                   else
                        small_center_circle<='0';
                   end if;
            elsif Hcount<= center_cxstart+HBACK_PORCH+HSYNC     and Vcount
>= center_cystart+VBACK_PORCH+VSYNC then       --3rd quadrant
                  if (center_cxstart+HBACK_PORCH+HSYNC-Hcount)<=
small_center_radius and (Vcount-center_cystart-VBACK_PORCH-VSYNC)
<=small_center_radius
                        and (table(TO_INTEGER(center_cxstart-
Hcount)+HBACK_PORCH+HSYNC)+table(TO_INTEGER(Vcount-center_cystart)-
VBACK_PORCH-VSYNC) <= sqr_small_center_radius) then
                        small_center_circle <= '1';
                   else
                        small_center_circle<='0';
                   end if;
            else
                        small_center_circle <= '0';
            end if;
      end if;
  end process;
------------------------------------------------------------------------------
---------------------------
-----------------------------------------------------puck gen code--------
---------------------------
  CircleGen5 : process(clk25)
  begin
  if rising_edge(clk25) then
            if reset = '1' then
                  puck_circle <= '0';
            elsif Hcount>= puck_cxstart+HBACK_PORCH+HSYNC and Vcount >=
puck_cystart+VBACK_PORCH+VSYNC then --4th quadrant
                  if (Hcount-puck_cxstart-HBACK_PORCH-HSYNC)<= radius and
(Vcount-puck_cystart-VBACK_PORCH-VSYNC)<=radius
                        and (table(TO_INTEGER(Hcount-puck_cxstart)-
HBACK_PORCH-HSYNC)+table(TO_INTEGER(Vcount-puck_cystart)-VBACK_PORCH-
VSYNC))<= sqr_radius then
                        puck_circle <= '1';
                   else
```

```
                        puck_circle <= '0';
                   end if;
            elsif Hcount>= puck_cxstart+HBACK_PORCH+HSYNC and Vcount <=
puck_cystart+VBACK_PORCH+VSYNC then  --1st quadrant
                  if (Hcount-puck_cxstart-HBACK_PORCH-HSYNC)<= radius and
(puck_cystart+VBACK_PORCH+VSYNC-Vcount)<=radius
                        and (table(TO_INTEGER(Hcount-puck_cxstart)-
HBACK_PORCH-HSYNC)+table(TO_INTEGER(puck_cystart-
Vcount)+VBACK_PORCH+VSYNC))<= sqr_radius then
                        puck_circle <= '1';
                   else
                        puck_circle <= '0';
                   end if;
            elsif Hcount<= puck_cxstart+HBACK_PORCH+HSYNC and Vcount <=
puck_cystart+VBACK_PORCH+VSYNC then           --2nd quadrant
                  if (puck_cxstart+HBACK_PORCH+HSYNC-Hcount)<= radius and
(puck_cystart+VBACK_PORCH+VSYNC-Vcount)<=radius
                        and (table(TO_INTEGER(puck_cxstart-
Hcount)+HBACK_PORCH+HSYNC)+table(TO_INTEGER(puck_cystart-
Vcount)+VBACK_PORCH+VSYNC))<= sqr_radius then
                        puck_circle <= '1';
                   else
                        puck_circle<='0';
                   end if;
            elsif Hcount<= puck_cxstart+HBACK_PORCH+HSYNC and Vcount >=
puck_cystart+VBACK_PORCH+VSYNC then      --3rd quadrant
                  if (puck_cxstart+HBACK_PORCH+HSYNC-Hcount)<= radius and
(Vcount-puck_cystart-VBACK_PORCH-VSYNC)<=radius
                  and (table(TO_INTEGER(puck_cxstart-
Hcount)+HBACK_PORCH+HSYNC)+table(TO_INTEGER(Vcount-puck_cystart)-
VBACK_PORCH-VSYNC))<= sqr_radius then
                        puck_circle <= '1';
                   else
                        puck_circle<='0';
                   end if;
            else
                        puck_circle <= '0';
            end if;
      end if;
  end process;

---------------------------------------------------------------------------
-------------------------
---------------------------------------------Goal Gen Code-----------------
-------------------------
  Goalgen : process(clk25)
  begin
      if rising_edge (clk25) then
            if reset = '1' then
                  goal_outer_circle <= '0';
            --------------------------Left Goal-----------------------
            elsif (Hcount>= center_gxstart_l + HBACK_PORCH+HSYNC and Vcount
>= center_gystart+VBACK_PORCH+VSYNC
```

```
                             and Hcount <= center_gxstart_l + gradius +
HBACK_PORCH + HSYNC) then                          -- to take it to next
condition

                 if (Hcount-center_gxstart_l - HBACK_PORCH -
HSYNC)<=gradius and (Vcount-center_gystart-VBACK_PORCH-VSYNC) <= gradius
                 and (table(TO_INTEGER(Hcount-center_gxstart_l )-
HBACK_PORCH - HSYNC) + table(TO_INTEGER(Vcount-center_gystart)-
VBACK_PORCH-VSYNC)) <= (sqr_gradius) then
                       goal_outer_circle <= '1';
                 else
                       goal_outer_circle <= '0';
                 end if;

           elsif (Hcount>= center_gxstart_l + HBACK_PORCH + HSYNC and
Vcount <= center_gystart + VBACK_PORCH + VSYNC
                             and Hcount <= center_gxstart_l + gradius +
HBACK_PORCH + HSYNC) then

                 if (Hcount-center_gxstart_l - HBACK_PORCH -
HSYNC)<=gradius and (center_gystart-Vcount+VBACK_PORCH+VSYNC) <= gradius
                 and (table(TO_INTEGER(Hcount-center_gxstart_l )-
HBACK_PORCH - HSYNC) + table(TO_INTEGER(center_gystart-
Vcount)+VBACK_PORCH+VSYNC)) <= (sqr_gradius) then
                             goal_outer_circle <= '1';
                 else
                             goal_outer_circle <= '0';
                 end if;

      -------------------------------------Right Goal--------------------
-----------------------------------------
           elsif (Hcount <= center_gxstart_r + HBACK_PORCH + HSYNC and
Vcount <= center_gystart + VBACK_PORCH + VSYNC )then

                 if (center_gxstart_r + HBACK_PORCH + HSYNC - Hcount)
<=gradius and (center_gystart+VBACK_PORCH+VSYNC-Vcount) <= gradius
                 and (table(TO_INTEGER(center_gxstart_r - Hcount)+
HBACK_PORCH + HSYNC) + table(TO_INTEGER(center_gystart-
Vcount)+VBACK_PORCH+VSYNC))<= (sqr_gradius) then
                       goal_outer_circle <= '1';
                 else
                       goal_outer_circle <= '0';
                 end if;
           elsif (Hcount <= center_gxstart_r+HBACK_PORCH+HSYNC and Vcount
>= center_gystart+VBACK_PORCH+VSYNC) then

                 if (center_gxstart_r + HBACK_PORCH + HSYNC - Hcount)
<=gradius and (Vcount-center_gystart-VBACK_PORCH-VSYNC) <= gradius
                 and (table(TO_INTEGER(center_gxstart_r - Hcount)+
HBACK_PORCH + HSYNC) + table(TO_INTEGER(Vcount-center_gystart)-
VBACK_PORCH-VSYNC))<= (sqr_gradius) then
                       goal_outer_circle <= '1';
                 else
                       goal_outer_circle<='0';
                 end if;
           else
                 goal_outer_circle <= '0';
           end if;
     end if;
  end process;
-----------------------------------------goal inner semi circle
generation-------------------------------------------------------
goalgeninner: process(clk25)
begin
     if rising_edge (clk25) then
           if reset = '1' then
                 goal_inner_circle <= '0';
elsif (Hcount>= center_gxstart_l + HBACK_PORCH+HSYNC and Vcount >=
center_gystart+VBACK_PORCH+VSYNC
                             and Hcount <= center_gxstart_l + gradius +
HBACK_PORCH + HSYNC) then
     if (Hcount-center_gxstart_l - HBACK_PORCH - HSYNC) <=gradius_inner
and (Vcount-center_gystart-VBACK_PORCH-VSYNC) <=gradius_inner
        and (table(TO_INTEGER(Hcount-center_gxstart_l)-HBACK_PORCH-HSYNC)
+ table(TO_INTEGER(Vcount-center_gystart)-VBACK_PORCH-VSYNC))<=
(sqr_gradius_inner) then
                             goal_inner_circle <= '1';
                 else
                             goal_inner_circle <= '0';
                 end if;
elsif (Hcount>= center_gxstart_l + HBACK_PORCH + HSYNC and Vcount <=
center_gystart + VBACK_PORCH + VSYNC
                             and Hcount <= center_gxstart_l + gradius +
HBACK_PORCH + HSYNC) then
     if (Hcount-center_gxstart_l - HBACK_PORCH - HSYNC) <=gradius_inner
and (center_gystart+VBACK_PORCH+VSYNC-Vcount) <=gradius_inner
        and (table(TO_INTEGER(Hcount-center_gxstart_l)-HBACK_PORCH-HSYNC)
+ table(TO_INTEGER(center_gystart-Vcount)+VBACK_PORCH+VSYNC))<=
(sqr_gradius_inner) then
                             goal_inner_circle <= '1';
                 else
                             goal_inner_circle <= '0';
                 end if;
elsif (Hcount <= center_gxstart_r + HBACK_PORCH + HSYNC and Vcount <=
center_gystart + VBACK_PORCH + VSYNC )then
     if (center_gxstart_r + HBACK_PORCH + HSYNC - Hcount) <=gradius_inner
and (center_gystart+VBACK_PORCH+VSYNC-Vcount) <=gradius_inner
     and (table(TO_INTEGER(center_gxstart_r - Hcount)+ HBACK_PORCH +
HSYNC )+ table(TO_INTEGER(center_gystart-Vcount)+VBACK_PORCH+VSYNC))<=
(sqr_gradius_inner) then
                       goal_inner_circle <= '1';
                 else
                       goal_inner_circle <= '0';
                 end if;

elsif (Hcount <= center_gxstart_r+HBACK_PORCH+HSYNC and Vcount >=
center_gystart+VBACK_PORCH+VSYNC) then
```

```
      if (center_gxstart_r + HBACK_PORCH + HSYNC - Hcount) <=gradius_inner
and (Vcount-center_gystart-VBACK_PORCH-VSYNC)<=gradius_inner
      and (table(TO_INTEGER(center_gxstart_r - Hcount)+ HBACK_PORCH +
HSYNC )+ table(TO_INTEGER(Vcount-center_gystart)-VBACK_PORCH-VSYNC))<=
(sqr_gradius_inner) then
                      goal_inner_circle <= '1';
                 else
                      goal_inner_circle<='0';
                 end if;
else
                 goal_inner_circle <= '0';
            end if;
      end if;
  end process;
----------------------------------goal generation-------------------------
-------------------------------------
Goalgeneration : process(clk25)
  begin
    if rising_edge (clk25) then
        if reset = '1' then
            goal <= '0';
        elsif (Vcount >= goal_lower_slot+VSYNC+VBACK_PORCH and Vcount <=
goal_upper_slot+ VSYNC+VBACK_PORCH ) and ((Hcount=HSYNC+HBACK_PORCH) or
(Hcount= HTOTAL-HFRONT_PORCH-1)) then
            goal <= '1';
        else
            goal <= '0';
        end if;
    end if;
  end process;
---------------------------------------------------------------------------
--------------------------------------
Scoregenplayer1  : process(clk25)
variable htmp: unsigned (9 downto 0);
variable vtmp: unsigned (9 downto 0);
variable ctmp: unsigned (0 to 50) ;

      begin
            if rising_edge(clk25) then
                 if reset =  '1' then
                      score_player1 <= '0';
                 elsif (((Hcount >= HBACK_PORCH + HSYNC and Hcount < 52 +
HBACK_PORCH + HSYNC) )
                      and (Vcount > 468 + VSYNC+ VBACK_PORCH  and Vcount
<= 476 +  VSYNC+ VBACK_PORCH)) then

                         htmp := Hcount - HSYNC - HBACK_PORCH -0 ;
                         vtmp := Vcount - VSYNC - VBACK_PORCH - 469 ;
                         ctmp := ( others => '0' ) ;
                         ctmp := crange(TO_INTEGER(vtmp)) ;
                         if ctmp(TO_INTEGER( htmp ) ) = '1' then
                         score_player1 <= '1' ;
                         elsif ctmp(TO_INTEGER(htmp)) = '0' then
                         score_player1 <= '0' ;
                         end if ;
                  else
                        score_player1 <= '0';
                  end if;
            end if;
      end process;
--------------------------------------------------Player2-----------------
---------------------------
Scoregenplayer2  : process(clk25)
variable htmp1: unsigned (9 downto 0);
variable vtmp1: unsigned (9 downto 0);
variable ctmp1: unsigned (0 to 54) ;

      begin
            if rising_edge(clk25) then
                  if reset =  '1' then
                        score_player2 <= '0';
                  elsif (((Hcount >= HBACK_PORCH + HSYNC +500 and Hcount <
556 + HBACK_PORCH + HSYNC) )
                        and (Vcount > 468 + VSYNC+ VBACK_PORCH  and Vcount
<= 476 +  VSYNC+ VBACK_PORCH)) then

                           htmp1 := Hcount - HSYNC - HBACK_PORCH - 500 ;
                           vtmp1 := Vcount - VSYNC - VBACK_PORCH - 469 ;
                           ctmp1 := ( others => '0' ) ;
                           ctmp1 := crange1(TO_INTEGER(vtmp1)) ;
                           if ctmp1(TO_INTEGER( htmp1 ) ) = '1' then
                           score_player2 <= '1' ;
                           elsif ctmp1(TO_INTEGER(htmp1)) = '0' then
                           score_player2 <= '0' ;
                           end if ;
                  else
                        score_player2 <= '0';
                  end if;
            end if;
      end process;
---------------------------------------------------------------------------
---------------------------
-----------------------------------------socre keepig for player 1 -------
---------------------------
PLAYER1_score    : process(clk25)
variable htmp1: unsigned (9 downto 0);
variable vtmp1: unsigned (9 downto 0);
variable ctmp1: unsigned (0 to 71) ;

      begin
            if rising_edge(clk25) then

                  if reset =  '1' then
                        sc_player_1 <= '0';
                  elsif (((Hcount >= HBACK_PORCH + HSYNC +56 and Hcount <
63 + HBACK_PORCH + HSYNC) )
                        and (Vcount > 468 + VSYNC+ VBACK_PORCH  and Vcount
<= 476 +  VSYNC+ VBACK_PORCH)) then
```

```
                          htmp1 := Hcount - HSYNC - HBACK_PORCH - 56 ;
                          vtmp1 := Vcount - VSYNC - VBACK_PORCH - 469 ;
                          ctmp1 := ( others => '0' ) ;
                          ctmp1 := crange2(TO_INTEGER(vtmp1)) ;
                          if ctmp1((TO_INTEGER( htmp1 ) )+ (player_1 * 8)
) = '1' then --use logical bitwise shift instead of multiply since 8 is a
power of two
                          sc_player_1 <= '1' ;
                          elsif ctmp1((TO_INTEGER(htmp1))+ (player_1 * 8))
= '0' then
                          sc_player_1 <= '0' ;
                          end if ;
                 else
                       sc_player_1 <= '0';
                 end if;
           end if;
     end process;
--------------------------------------------------------------------------
-------------------------------
-----------------------------------------------score keeping for player 2-
---------------------------
PLAYER2_score    : process(clk25)
variable htmp1: unsigned (9 downto 0);
variable vtmp1: unsigned (9 downto 0);
variable ctmp1: unsigned (0 to 71) ;

     begin
           if rising_edge(clk25) then

                 if reset =  '1' then
                       sc_player_2 <= '0';
                 elsif (((Hcount >= HBACK_PORCH + HSYNC + 558 and Hcount <
565 + HBACK_PORCH + HSYNC) )
                       and (Vcount > 468 + VSYNC+ VBACK_PORCH  and Vcount
<= 476 +  VSYNC+ VBACK_PORCH)) then
                          htmp1 := Hcount - HSYNC - HBACK_PORCH - 558 ;
                          vtmp1 := Vcount - VSYNC - VBACK_PORCH - 469 ;
                          ctmp1 := ( others => '0' ) ;
                          ctmp1 := crange2(TO_INTEGER(vtmp1)) ;
                          if ctmp1((TO_INTEGER( htmp1 ) )+ (player_2 * 8)
) = '1' then --use logical bitwise shift instead of multiply since 8 is a
power of two
                          sc_player_2 <= '1' ;
                          elsif ctmp1((TO_INTEGER(htmp1))+ (player_2 * 8))
= '0' then
                          sc_player_2 <= '0' ;
                          end if ;
                 else
                       sc_player_2 <= '0';
                 end if;
           end if;
     end process;
--------------------------------------------------------------------------
---------------------------
```

```
VideoOut: process (clk25, reset_n)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif rising_edge (clk25) then
        if goal= '1' then
           VGA_R <= "1111111111";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
           elsif puck_circle = '1' then
           VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "1111111111";
           elsif circle ='1' or ethernet_circle ='1' then
           VGA_R <= "1111111111";
        VGA_G <= "1111111111";
        VGA_B <= "1111111111";
        elsif goal_inner_circle = '1' then    --- maintaining the
hierarchichal order of gen the graphics.
           VGA_R <= "0000000000";
        VGA_G <= "1111100111";
        VGA_B <= "1111111111";
           elsif vline = '1' or goal_outer_circle = '1' then
           VGA_R <= "1111111111";
        VGA_G <= "1111111111";
        VGA_B <= "1111111111";
           elsif small_center_circle = '1' then   --- maintaining the
hierarchichal order of gen the graphics.
           VGA_R <= "0000000000";
        VGA_G <= "1111100111";
        VGA_B <= "1111111111";
           elsif center_circle = '1' or score_player1 =  '1' or
sc_player_1 = '1' or score_player2 = '1' or sc_player_2 = '1' then
        VGA_R <= "1111111111";
        VGA_G <= "1111111111";
        VGA_B <= "1111111111";
        elsif ((vga_hblank = '0' and vga_vblank ='0')and Vcount <= 464
+VSYNC+VBACK_PORCH) then  -- to leave space in the end for player name and
score.
        VGA_R <= "0000000000";
        VGA_G <= "1111100111";
        VGA_B <= "1111111111";
        else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk25;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

8.1.3 DM9000_IF.v

```
--
-- Ethernet Component
-- Team NAHVG
--

module DM9000A_IF(     //    HOST Side
                             iDATA,oDATA,iCMD,
                             iRD_N,iWR_N,
                             iCS_N,iRST_N,
                             oINT,
                             //    DM9000A Side
                             ENET_DATA,ENET_CMD,
                             ENET_RD_N,ENET_WR_N,
                             ENET_CS_N,ENET_RST_N,
                             ENET_INT
                             );
//    HOST Side
input [15:0]      iDATA;
input             iCMD;
input             iRD_N;
input             iWR_N;
input             iCS_N;
input             iRST_N;
output      [15:0]      oDATA;
output                  oINT;
//    DM9000A Side
inout [15:0]      ENET_DATA;
output                  ENET_CMD;
output                  ENET_RD_N;
output                  ENET_WR_N;
output                  ENET_CS_N;
output                  ENET_RST_N;
input             ENET_INT;

assign      ENET_DATA   =     ENET_WR_N   ?     16'hzzzz    :     iDATA ;
assign      oDATA       =     ENET_DATA   ;

assign      ENET_CMD    =     iCMD;
assign      ENET_RD_N   =     iRD_N;
assign      ENET_WR_N   =     iWR_N;
assign      ENET_CS_N   =     iCS_N;
assign      ENET_RST_N  =     iRST_N;
assign      oINT        =     ENET_INT;

endmodule
```

8.1.4 Altera_UP_Avalon_PS2.v

```
/*****************************************************************************
****
 *
*
 * Module:       Altera_UP_Avalon_PS2
*
 * Description:
*
 *      This module connects the PS2 core to Avalon.
*
 *
*     Team NAHVG

*****************************************************************************
***/

/*
 *
 * Data Register Bits
 * Read Available 31-16, Incoming Data or Outgoing Command 7-0
 *
 * Control Register Bits
 * CE 10, RI 8, RE 0
 *
 **/

module Altera_UP_Avalon_PS2 (
      // Inputs
      clk,
      reset,

      address,
      chipselect,
      byteenable,
      read,
      write,
      writedata,

      // Bidirectionals
      PS2_CLK,                           // PS2 Clock
      PS2_DAT,                           // PS2 Data

      // Outputs
      irq,
      readdata,
      waitrequest
);
```

```
/***********************************************************************
****
 *                             Parameter Declarations
*

***********************************************************************
***/


/***********************************************************************
****
 *                               Port Declarations
*

***********************************************************************
***/
// Inputs
input                   clk;
input                   reset;

input                   address;
input                   chipselect;
input           [3:0] byteenable;
input                   read;
input                   write;
input           [31:0]      writedata;

// Bidirectionals
inout                   PS2_CLK;
inout                   PS2_DAT;

// Outputs
output                          irq;
output          [31:0]      readdata;
output                          waitrequest;

reg             [31:0]      readdata;

/***********************************************************************
****
 *                             Constant Declarations
*

***********************************************************************
***/


/***********************************************************************
****
 *                   Internal wires and registers Declarations
*

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

```
// Internal Wires
wire       [7:0] data_from_the_PS2_port;
wire                   data_from_the_PS2_port_en;

wire                   get_data_from_PS2_port;
wire                   send_command_to_PS2_port;
wire                   clear_command_error;
wire                   set_interrupt_enable;

wire                   command_was_sent;
wire                   error_sending_command;

wire                   data_fifo_is_empty;
wire                   data_fifo_is_full;

// Internal Registers
reg              [31:0]     data_register;
reg              [31:0]     control_register;

reg              [7:0] data_in_fifo;
reg              [15:0]     data_available;

// State Machine Registers

/***************************************************************************
****
 *                          Finite State Machine(s)
*

***************************************************************************
***/


/***************************************************************************
****
 *                              Sequential logic
*

***************************************************************************
***/

always @(posedge clk)
begin
      if (reset == 1'b1)
            readdata <= 32'h00000000;
      else if (chipselect == 1'b1)
      begin
            if (address == 1'b0)
                  readdata <= {data_available, 8'h00, data_in_fifo};
            else
                  readdata <= control_register;
      end
end
```

```
always @(posedge clk)
begin
      if (reset == 1'b1)
            control_register <= 32'h00000000;
      else
      begin
            if (error_sending_command == 1'b1)
                  control_register[10] <= 1'b1;
            else if (clear_command_error == 1'b1)
                  control_register[10] <= 1'b0;

            control_register[8] <= ~data_fifo_is_empty &
control_register[0];

            if ((chipselect == 1'b1) && (set_interrupt_enable == 1'b1))
                  control_register[0]  <= writedata[0];
      end
end

/*****************************************************************************
****
 *                              Combinational logic
*

*****************************************************************************
***/

assign irq              = control_register[8];
assign waitrequest      = send_command_to_PS2_port &
                                    ~(command_was_sent |
error_sending_command);

assign get_data_from_PS2_port  = chipselect & byteenable[0] & ~address &
read;
assign send_command_to_PS2_port= chipselect & byteenable[0] & ~address &
write;
assign clear_command_error     = chipselect & byteenable[1] & address &
write;
assign set_interrupt_enable    = chipselect & byteenable[0] & address &
write;

/*****************************************************************************
****
 *                                Internal Modules
*

*****************************************************************************
***/

Altera_UP_PS2 PS2_Serial_Port (
      // Inputs
      .clk                                    (clk),
      .reset                                        (reset),
	.the_command					(writedata[7:0]),
	.send_command					(send_command_to_PS2_port),

	// Bidirectionals
	.PS2_CLK					(PS2_CLK),
	.PS2_DAT					(PS2_DAT),

	// Outputs
	.command_was_sent				(command_was_sent),
	.error_communication_timed_out	(error_sending_command),

	.received_data					(data_from_the_PS2_port),
	.received_data_en				(data_from_the_PS2_port_en)
);

scfifo	Incoming_Data_FIFO (
	// Inputs
	.clock				(clk),
	.sclr			(reset),

	.rdreq				(get_data_from_PS2_port &
~data_fifo_is_empty),
	.wrreq				(data_from_the_PS2_port_en &
~data_fifo_is_full),
	.data			(data_from_the_PS2_port),

	// Bidirectionals

	// Outputs
	.q				(data_in_fifo),

	.usedw				(data_available),
	.empty				(data_fifo_is_empty),
	.full			(data_fifo_is_full)

	// synopsys translate_off
	,
	.almost_empty	(),
	.almost_full	(),
	.aclr			()
	// synopsys translate_on
);
defparam
	Incoming_Data_FIFO.add_ram_output_register	= "ON",
	Incoming_Data_FIFO.intended_device_family	= "Cyclone II",
	Incoming_Data_FIFO.lpm_numwords			= 256,
	Incoming_Data_FIFO.lpm_showahead		= "ON",
	Incoming_Data_FIFO.lpm_type			= "scfifo",
	Incoming_Data_FIFO.lpm_width			= 8,
	Incoming_Data_FIFO.lpm_widthu			= 8,
	Incoming_Data_FIFO.overflow_checking		= "OFF",
	Incoming_Data_FIFO.underflow_checking		= "OFF",
	Incoming_Data_FIFO.use_eab			= "ON";
endmodule
```

8.1.5 Altera_UP_PS2.v

```
/*****************************************************************************
****
 *
*
 * Module:       Altera_UP_PS2
*
 * Description:
*
 *      This module communicates with the PS2 core.
*
 *
*     Team NAHVG

*****************************************************************************
***/

module Altera_UP_PS2 (
      // Inputs
      clk,
      reset,

      the_command,
      send_command,

      // Bidirectionals
      PS2_CLK,                                // PS2 Clock
      PS2_DAT,                                // PS2 Data

      // Outputs
      command_was_sent,
      error_communication_timed_out,

      received_data,
      received_data_en             // If 1 - new data has been received
);

/*****************************************************************************
****
 *                          Parameter Declarations
*

*****************************************************************************
***/


/*****************************************************************************
****
 *                            Port Declarations
*
```

```
***************************************************************************
***/
// Inputs
input           clk;
input           reset;

input [7:0] the_command;
input           send_command;

// Bidirectionals
inout           PS2_CLK;
inout           PS2_DAT;

// Outputs
output                  command_was_sent;
output                  error_communication_timed_out;

output      [7:0] received_data;
output                  received_data_en;

/**************************************************************************
****
 *                            Constant Declarations
*

***************************************************************************
***/
// states
localparam  PS2_STATE_0_IDLE            = 3'h0,
                PS2_STATE_1_DATA_IN             = 3'h1,
                PS2_STATE_2_COMMAND_OUT         = 3'h2,
                PS2_STATE_3_END_TRANSFER    = 3'h3,
                PS2_STATE_4_END_DELAYED         = 3'h4;

/**************************************************************************
****
 *                  Internal wires and registers Declarations
*

***************************************************************************
***/
// Internal Wires
wire            ps2_clk_posedge;
wire            ps2_clk_negedge;

wire            start_receiving_data;
wire            wait_for_incoming_data;

// Internal Registers
reg         [7:0] idle_counter;

reg                     ps2_clk_reg;
reg                     ps2_data_reg;
reg                    last_ps2_clk;

// State Machine Registers
reg         [2:0] ns_ps2_transceiver;
reg         [2:0] s_ps2_transceiver;

/***************************************************************************
****
 *                         Finite State Machine(s)
*

***************************************************************************
***/

always @(posedge clk)
begin
      if (reset == 1'b1)
            s_ps2_transceiver <= PS2_STATE_0_IDLE;
      else
            s_ps2_transceiver <= ns_ps2_transceiver;
end

always @(*)
begin
      // Defaults
      ns_ps2_transceiver = PS2_STATE_0_IDLE;

    case (s_ps2_transceiver)
      PS2_STATE_0_IDLE:
            begin
                  if ((idle_counter == 8'hFF) &&
                              (send_command == 1'b1))
                        ns_ps2_transceiver = PS2_STATE_2_COMMAND_OUT;
                  else if ((ps2_data_reg == 1'b0) && (ps2_clk_posedge ==
1'b1))
                        ns_ps2_transceiver = PS2_STATE_1_DATA_IN;
                  else
                        ns_ps2_transceiver = PS2_STATE_0_IDLE;
            end
      PS2_STATE_1_DATA_IN:
            begin
                  if ((received_data_en == 1'b1)/* && (ps2_clk_posedge ==
1'b1)*/)
                        ns_ps2_transceiver = PS2_STATE_0_IDLE;
                  else
                        ns_ps2_transceiver = PS2_STATE_1_DATA_IN;
            end
      PS2_STATE_2_COMMAND_OUT:
            begin
                  if ((command_was_sent == 1'b1) ||
                        (error_communication_timed_out == 1'b1))
                        ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
                  else
                        ns_ps2_transceiver = PS2_STATE_2_COMMAND_OUT;
```

```
			end
		PS2_STATE_3_END_TRANSFER:
			begin
				if (send_command == 1'b0)
					ns_ps2_transceiver = PS2_STATE_0_IDLE;
				else if ((ps2_data_reg == 1'b0) && (ps2_clk_posedge ==
1'b1))
					ns_ps2_transceiver = PS2_STATE_4_END_DELAYED;
				else
					ns_ps2_transceiver = PS2_STATE_3_END_TRANSFER;
			end
		PS2_STATE_4_END_DELAYED:
			begin
				if (received_data_en == 1'b1)
				begin
					if (send_command == 1'b0)
						ns_ps2_transceiver = PS2_STATE_0_IDLE;
					else
						ns_ps2_transceiver =
PS2_STATE_3_END_TRANSFER;
				end
				else
					ns_ps2_transceiver = PS2_STATE_4_END_DELAYED;
			end
		default:
				ns_ps2_transceiver = PS2_STATE_0_IDLE;
		endcase
end

/*************************************************************************
****
 *                              Sequential logic
*

*************************************************************************
***/

always @(posedge clk)
begin
	if (reset == 1'b1)
	begin
		last_ps2_clk	<= 1'b1;
		ps2_clk_reg	<= 1'b1;

		ps2_data_reg	<= 1'b1;
	end
	else
	begin
		last_ps2_clk	<= ps2_clk_reg;
		ps2_clk_reg	<= PS2_CLK;

		ps2_data_reg	<= PS2_DAT;
	end
end
```

```
always @(posedge clk)
begin
      if (reset == 1'b1)
            idle_counter <= 6'h00;
      else if ((s_ps2_transceiver == PS2_STATE_0_IDLE) &&
                  (idle_counter != 8'hFF))
            idle_counter <= idle_counter + 6'h01;
      else if (s_ps2_transceiver != PS2_STATE_0_IDLE)
            idle_counter <= 6'h00;
end

/*****************************************************************************
****
 *                              Combinational logic
*

*****************************************************************************
***/

assign ps2_clk_posedge =
                  ((ps2_clk_reg == 1'b1) && (last_ps2_clk == 1'b0)) ? 1'b1
: 1'b0;
assign ps2_clk_negedge =
                  ((ps2_clk_reg == 1'b0) && (last_ps2_clk == 1'b1)) ? 1'b1
: 1'b0;

assign start_receiving_data       = (s_ps2_transceiver ==
PS2_STATE_1_DATA_IN);
assign wait_for_incoming_data     =
                  (s_ps2_transceiver == PS2_STATE_3_END_TRANSFER);

/*****************************************************************************
****
 *                                Internal Modules
*

*****************************************************************************
***/

Altera_UP_PS2_Data_In PS2_Data_In (
      // Inputs
      .clk                                    (clk),
      .reset                                        (reset),

      .wait_for_incoming_data                 (wait_for_incoming_data),
      .start_receiving_data             (start_receiving_data),

      .ps2_clk_posedge                  (ps2_clk_posedge),
      .ps2_clk_negedge                  (ps2_clk_negedge),
      .ps2_data                               (ps2_data_reg),

      // Bidirectionals
	// Outputs
	.received_data				(received_data),
	.received_data_en			(received_data_en)
);

Altera_UP_PS2_Command_Out PS2_Command_Out (
	// Inputs
	.clk					(clk),
	.reset						(reset),

	.the_command				(the_command),
	.send_command				(send_command),

	.ps2_clk_posedge			(ps2_clk_posedge),
	.ps2_clk_negedge			(ps2_clk_negedge),

	// Bidirectionals
	.PS2_CLK				(PS2_CLK),
	.PS2_DAT				(PS2_DAT),

	// Outputs
	.command_was_sent			(command_was_sent),
	.error_communication_timed_out	(error_communication_timed_out)
);

endmodule
```

8.1.6 Altera_UP_PS2_Command_Out

```
/***************************************************************************
****
 *
*
 * Module:       Altera_UP_PS2_Command_Out
*
 * Description:
*
 *      This module sends commands out to the PS2 core.
*
 *
*     Team NAHVG

***************************************************************************
***/


module Altera_UP_PS2_Command_Out (
	// Inputs
	clk,
	reset,

	the_command,
      send_command,

      ps2_clk_posedge,
      ps2_clk_negedge,

      // Bidirectionals
      PS2_CLK,
      PS2_DAT,

      // Outputs
      command_was_sent,
      error_communication_timed_out
);

/***************************************************************************
****
 *                             Parameter Declarations
*

***************************************************************************
***/

// Timing info for initiating Host-to-Device communication
//   when using a 50MHz system clock
parameter   CLOCK_CYCLES_FOR_101US      = 5050;
parameter   NUMBER_OF_BITS_FOR_101US    = 13;
parameter   COUNTER_INCREMENT_FOR_101US = 13'h0001;

//parameter CLOCK_CYCLES_FOR_101US      = 50;
//parameter NUMBER_OF_BITS_FOR_101US    = 6;
//parameter COUNTER_INCREMENT_FOR_101US = 6'h01;

// Timing info for start of transmission error
//   when using a 50MHz system clock
parameter   CLOCK_CYCLES_FOR_15MS       = 750000;
parameter   NUMBER_OF_BITS_FOR_15MS           = 20;
parameter   COUNTER_INCREMENT_FOR_15MS  = 20'h00001;

// Timing info for sending data error
//   when using a 50MHz system clock
parameter   CLOCK_CYCLES_FOR_2MS        = 100000;
parameter   NUMBER_OF_BITS_FOR_2MS      = 17;
parameter   COUNTER_INCREMENT_FOR_2MS   = 17'h00001;

/***************************************************************************
****
 *                               Port Declarations
*

***************************************************************************
***/
// Inputs
input                  clk;
input                  reset;
input       [7:0] the_command;
input             send_command;

input             ps2_clk_posedge;
input             ps2_clk_negedge;

// Bidirectionals
inout             PS2_CLK;
inout             PS2_DAT;

// Outputs
output      reg         command_was_sent;
output      reg         error_communication_timed_out;

/*****************************************************************************
****
 *                            Constant Declarations
*

*****************************************************************************
***/
// states
parameter   PS2_STATE_0_IDLE                        = 3'h0,
                  PS2_STATE_1_INITIATE_COMMUNICATION= 3'h1,
                  PS2_STATE_2_WAIT_FOR_CLOCK              = 3'h2,
                  PS2_STATE_3_TRANSMIT_DATA               = 3'h3,
                  PS2_STATE_4_TRANSMIT_STOP_BIT           = 3'h4,
                  PS2_STATE_5_RECEIVE_ACK_BIT             = 3'h5,
                  PS2_STATE_6_COMMAND_WAS_SENT      = 3'h6,
                  PS2_STATE_7_TRANSMISSION_ERROR          = 3'h7;

/*****************************************************************************
****
 *                  Internal wires and registers Declarations
*

*****************************************************************************
***/
// Internal Wires

// Internal Registers
reg               [3:0] cur_bit;
reg               [8:0] ps2_command;

reg               [NUMBER_OF_BITS_FOR_101US:1] command_initiate_counter;

reg               [NUMBER_OF_BITS_FOR_15MS:1]       waiting_counter;
reg               [NUMBER_OF_BITS_FOR_2MS:1]        transfer_counter;

// State Machine Registers
reg               [2:0] ns_ps2_transmitter;
reg               [2:0] s_ps2_transmitter;
```

```
/***************************************************************************
****
 *                          Finite State Machine(s)
*

***************************************************************************
***/

always @(posedge clk)
begin
      if (reset == 1'b1)
            s_ps2_transmitter <= PS2_STATE_0_IDLE;
      else
            s_ps2_transmitter <= ns_ps2_transmitter;
end

always @(*)
begin
      // Defaults
      ns_ps2_transmitter = PS2_STATE_0_IDLE;

    case (s_ps2_transmitter)
      PS2_STATE_0_IDLE:
            begin
                  if (send_command == 1'b1)
                        ns_ps2_transmitter =
PS2_STATE_1_INITIATE_COMMUNICATION;
                  else
                        ns_ps2_transmitter = PS2_STATE_0_IDLE;
            end
      PS2_STATE_1_INITIATE_COMMUNICATION:
            begin
                  if (command_initiate_counter == CLOCK_CYCLES_FOR_101US)
                        ns_ps2_transmitter = PS2_STATE_2_WAIT_FOR_CLOCK;
                  else
                        ns_ps2_transmitter =
PS2_STATE_1_INITIATE_COMMUNICATION;
            end
      PS2_STATE_2_WAIT_FOR_CLOCK:
            begin
                  if (ps2_clk_negedge == 1'b1)
                        ns_ps2_transmitter = PS2_STATE_3_TRANSMIT_DATA;
                  else if (waiting_counter == CLOCK_CYCLES_FOR_15MS)
                        ns_ps2_transmitter =
PS2_STATE_7_TRANSMISSION_ERROR;
                  else
                        ns_ps2_transmitter = PS2_STATE_2_WAIT_FOR_CLOCK;
            end
      PS2_STATE_3_TRANSMIT_DATA:
            begin
                  if ((cur_bit == 4'd8) && (ps2_clk_negedge == 1'b1))
                        ns_ps2_transmitter = PS2_STATE_4_TRANSMIT_STOP_BIT;
                  else if (transfer_counter == CLOCK_CYCLES_FOR_2MS)
```

```
                        ns_ps2_transmitter =
PS2_STATE_7_TRANSMISSION_ERROR;
                  else
                        ns_ps2_transmitter = PS2_STATE_3_TRANSMIT_DATA;
            end
      PS2_STATE_4_TRANSMIT_STOP_BIT:
            begin
                  if (ps2_clk_negedge == 1'b1)
                        ns_ps2_transmitter = PS2_STATE_5_RECEIVE_ACK_BIT;
                  else if (transfer_counter == CLOCK_CYCLES_FOR_2MS)
                        ns_ps2_transmitter =
PS2_STATE_7_TRANSMISSION_ERROR;
                  else
                        ns_ps2_transmitter = PS2_STATE_4_TRANSMIT_STOP_BIT;
            end
      PS2_STATE_5_RECEIVE_ACK_BIT:
            begin
                  if (ps2_clk_posedge == 1'b1)
                        ns_ps2_transmitter = PS2_STATE_6_COMMAND_WAS_SENT;
                  else if (transfer_counter == CLOCK_CYCLES_FOR_2MS)
                        ns_ps2_transmitter =
PS2_STATE_7_TRANSMISSION_ERROR;
                  else
                        ns_ps2_transmitter = PS2_STATE_5_RECEIVE_ACK_BIT;
            end
      PS2_STATE_6_COMMAND_WAS_SENT:
            begin
                  if (send_command == 1'b0)
                        ns_ps2_transmitter = PS2_STATE_0_IDLE;
                  else
                        ns_ps2_transmitter = PS2_STATE_6_COMMAND_WAS_SENT;
            end
      PS2_STATE_7_TRANSMISSION_ERROR:
            begin
                  if (send_command == 1'b0)
                        ns_ps2_transmitter = PS2_STATE_0_IDLE;
                  else
                        ns_ps2_transmitter =
PS2_STATE_7_TRANSMISSION_ERROR;
            end
      default:
            begin
                  ns_ps2_transmitter = PS2_STATE_0_IDLE;
            end
      endcase
end

/*****************************************************************************
****
 *                               Sequential logic
*

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

```
always @(posedge clk)
begin
	if (reset == 1'b1)
		ps2_command <= 9'h000;
	else if (s_ps2_transmitter == PS2_STATE_0_IDLE)
		ps2_command <= {(^the_command) ^ 1'b1, the_command};
end

always @(posedge clk)
begin
	if (reset == 1'b1)
		command_initiate_counter <= {NUMBER_OF_BITS_FOR_101US{1'b0}};
	else if ((s_ps2_transmitter == PS2_STATE_1_INITIATE_COMMUNICATION)
&&
			(command_initiate_counter != CLOCK_CYCLES_FOR_101US))
		command_initiate_counter <=
			command_initiate_counter + COUNTER_INCREMENT_FOR_101US;
	else if (s_ps2_transmitter != PS2_STATE_1_INITIATE_COMMUNICATION)
		command_initiate_counter <= {NUMBER_OF_BITS_FOR_101US{1'b0}};
end

always @(posedge clk)
begin
	if (reset == 1'b1)
		waiting_counter <= {NUMBER_OF_BITS_FOR_15MS{1'b0}};
	else if ((s_ps2_transmitter == PS2_STATE_2_WAIT_FOR_CLOCK) &&
			(waiting_counter != CLOCK_CYCLES_FOR_15MS))
		waiting_counter <= waiting_counter +
COUNTER_INCREMENT_FOR_15MS;
	else if (s_ps2_transmitter != PS2_STATE_2_WAIT_FOR_CLOCK)
		waiting_counter <= {NUMBER_OF_BITS_FOR_15MS{1'b0}};
end

always @(posedge clk)
begin
	if (reset == 1'b1)
		transfer_counter <= {NUMBER_OF_BITS_FOR_2MS{1'b0}};
	else
	begin
		if ((s_ps2_transmitter == PS2_STATE_3_TRANSMIT_DATA) ||
			(s_ps2_transmitter == PS2_STATE_4_TRANSMIT_STOP_BIT) ||
			(s_ps2_transmitter == PS2_STATE_5_RECEIVE_ACK_BIT))
		begin
			if (transfer_counter != CLOCK_CYCLES_FOR_2MS)
				transfer_counter <= transfer_counter +
COUNTER_INCREMENT_FOR_2MS;
		end
		else
			transfer_counter <= {NUMBER_OF_BITS_FOR_2MS{1'b0}};
	end
end

always @(posedge clk)
```

```
begin
      if (reset == 1'b1)
            cur_bit <= 4'h0;
      else if ((s_ps2_transmitter == PS2_STATE_3_TRANSMIT_DATA) &&
                  (ps2_clk_negedge == 1'b1))
            cur_bit <= cur_bit + 4'h1;
      else if (s_ps2_transmitter != PS2_STATE_3_TRANSMIT_DATA)
            cur_bit <= 4'h0;
end

always @(posedge clk)
begin
      if (reset == 1'b1)
            command_was_sent <= 1'b0;
      else if (s_ps2_transmitter == PS2_STATE_6_COMMAND_WAS_SENT)
            command_was_sent <= 1'b1;
      else if (send_command == 1'b0)
                  command_was_sent <= 1'b0;
end

always @(posedge clk)
begin
      if (reset == 1'b1)
            error_communication_timed_out <= 1'b0;
      else if (s_ps2_transmitter == PS2_STATE_7_TRANSMISSION_ERROR)
            error_communication_timed_out <= 1'b1;
      else if (send_command == 1'b0)
            error_communication_timed_out <= 1'b0;
end

/*************************************************************************
****
 *                            Combinational logic
*

*************************************************************************
***/

assign PS2_CLK    =
      (s_ps2_transmitter == PS2_STATE_1_INITIATE_COMMUNICATION) ?
            1'b0 :
            1'bz;

assign PS2_DAT    =
      (s_ps2_transmitter == PS2_STATE_3_TRANSMIT_DATA) ?
ps2_command[cur_bit] :
      (s_ps2_transmitter == PS2_STATE_2_WAIT_FOR_CLOCK) ? 1'b0 :
      ((s_ps2_transmitter == PS2_STATE_1_INITIATE_COMMUNICATION) &&
            (command_initiate_counter[NUMBER_OF_BITS_FOR_101US] == 1'b1))
? 1'b0 :
                  1'bz;

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

```
 *                               Internal Modules
*

*****************************************************************************
***/

endmodule
```

8.1.7 Altera_UP_PS2_Data_In.v

```
/*****************************************************************************
****
 *
*
 * Module:       Altera_UP_PS2_Data_In
*
 * Description:
*
 *      This module accepts incoming data from a PS2 core.
*
 *
*     Team NAHVG

*****************************************************************************
***/


module Altera_UP_PS2_Data_In (
      // Inputs
      clk,
      reset,

      wait_for_incoming_data,
      start_receiving_data,

      ps2_clk_posedge,
      ps2_clk_negedge,
      ps2_data,

      // Bidirectionals

      // Outputs
      received_data,
      received_data_en             // If 1 - new data has been received
);


/*****************************************************************************
****
 *                            Parameter Declarations
*
```

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

/*************************************************************************
****
 *                              Port Declarations
*

**************************************************************************
***/
// Inputs
input                   clk;
input                   reset;

input                   wait_for_incoming_data;
input                   start_receiving_data;

input                   ps2_clk_posedge;
input                   ps2_clk_negedge;
input                   ps2_data;

// Bidirectionals

// Outputs
output reg  [7:0] received_data;

output reg              received_data_en;

/*************************************************************************
****
 *                            Constant Declarations
*

**************************************************************************
***/
// states
localparam  PS2_STATE_0_IDLE            = 3'h0,
                  PS2_STATE_1_WAIT_FOR_DATA    = 3'h1,
                  PS2_STATE_2_DATA_IN               = 3'h2,
                  PS2_STATE_3_PARITY_IN        = 3'h3,
                  PS2_STATE_4_STOP_IN               = 3'h4;

/*************************************************************************
****
 *                  Internal wires and registers Declarations
*

**************************************************************************
***/
// Internal Wires
reg               [3:0] data_count;
reg               [7:0] data_shift_reg;
```

```
// State Machine Registers
reg              [2:0] ns_ps2_receiver;
reg              [2:0] s_ps2_receiver;

/***************************************************************************
****
 *                          Finite State Machine(s)
*

***************************************************************************
***/

always @(posedge clk)
begin
      if (reset == 1'b1)
            s_ps2_receiver <= PS2_STATE_0_IDLE;
      else
            s_ps2_receiver <= ns_ps2_receiver;
end

always @(*)
begin
      // Defaults
      ns_ps2_receiver = PS2_STATE_0_IDLE;

    case (s_ps2_receiver)
      PS2_STATE_0_IDLE:
            begin
                  if ((wait_for_incoming_data == 1'b1) &&
                              (received_data_en == 1'b0))
                        ns_ps2_receiver = PS2_STATE_1_WAIT_FOR_DATA;
                  else if ((start_receiving_data == 1'b1) &&
                              (received_data_en == 1'b0))
                        ns_ps2_receiver = PS2_STATE_2_DATA_IN;
                  else
                        ns_ps2_receiver = PS2_STATE_0_IDLE;
            end
      PS2_STATE_1_WAIT_FOR_DATA:
            begin
                  if ((ps2_data == 1'b0) && (ps2_clk_posedge == 1'b1))
                        ns_ps2_receiver = PS2_STATE_2_DATA_IN;
                  else if (wait_for_incoming_data == 1'b0)
                        ns_ps2_receiver = PS2_STATE_0_IDLE;
                  else
                        ns_ps2_receiver = PS2_STATE_1_WAIT_FOR_DATA;
            end
      PS2_STATE_2_DATA_IN:
            begin
                  if ((data_count == 3'h7) && (ps2_clk_posedge == 1'b1))
                        ns_ps2_receiver = PS2_STATE_3_PARITY_IN;
                  else
                        ns_ps2_receiver = PS2_STATE_2_DATA_IN;
            end
      PS2_STATE_3_PARITY_IN:
            begin
                  if (ps2_clk_posedge == 1'b1)
                        ns_ps2_receiver = PS2_STATE_4_STOP_IN;
                  else
                        ns_ps2_receiver = PS2_STATE_3_PARITY_IN;
            end
      PS2_STATE_4_STOP_IN:
            begin
                  if (ps2_clk_posedge == 1'b1)
                        ns_ps2_receiver = PS2_STATE_0_IDLE;
                  else
                        ns_ps2_receiver = PS2_STATE_4_STOP_IN;
            end
      default:
            begin
                  ns_ps2_receiver = PS2_STATE_0_IDLE;
            end
      endcase
end

/***************************************************************************
****
 *                              Sequential logic
*

***************************************************************************
***/


always @(posedge clk)
begin
      if (reset == 1'b1)
            data_count  <= 3'h0;
      else if ((s_ps2_receiver == PS2_STATE_2_DATA_IN) &&
                  (ps2_clk_posedge == 1'b1))
            data_count  <= data_count + 3'h1;
      else if (s_ps2_receiver != PS2_STATE_2_DATA_IN)
            data_count  <= 3'h0;
end

always @(posedge clk)
begin
      if (reset == 1'b1)
            data_shift_reg                <= 8'h00;
      else if ((s_ps2_receiver == PS2_STATE_2_DATA_IN) &&
                  (ps2_clk_posedge == 1'b1))
            data_shift_reg   <= {ps2_data, data_shift_reg[7:1]};
end

always @(posedge clk)
begin
      if (reset == 1'b1)
            received_data          <= 8'h00;
	else if (s_ps2_receiver == PS2_STATE_4_STOP_IN)
		received_data	<= data_shift_reg;
end

always @(posedge clk)
begin
	if (reset == 1'b1)
		received_data_en	<= 1'b0;
	else if ((s_ps2_receiver == PS2_STATE_4_STOP_IN) &&
			(ps2_clk_posedge == 1'b1))
		received_data_en <= 1'b1;
	else
		received_data_en <= 1'b0;
end

/*****************************************************************************
****
 *                              Combinational logic
*

*****************************************************************************
***/


/*****************************************************************************
****
 *                                Internal Modules
*

*****************************************************************************
***/


endmodule
```

8.2 C Code

8.2.1 Master – Main.c

```
//main.c

//TEAM NAHVG

#include "basic_io.h"
#include "DM9000A.h"
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "ps2_mouse.h"
#include "VGA.h"
#include <string.h>
#include <math.h>
#define IOWR_VGA_DATA(base, offset, data) \
IOWR_16DIRECT(base, (offset) * 2, data)
#define MAX_MSG_LENGTH 128
#define delay 6000

// Ethernet MAC address.  Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F
};

unsigned int receive_buffer_length;
unsigned char receive_buffer[1600];
unsigned long checksum;
int data_mouse;
int ctrl_mouse;
int collision_ethernet=0;
int bytes_mouse;
int data_reg1;
int ethernet_x = 520, ethernet_y = 200;
int xdiff = 4, ydiff = 4;
int puck_x = 320, puck_y = 232;
int score1 =0 ,score2 =0;
int vel_pad_x = 0, vel_pad_y = 0, prev_pos_x =0, prev_pos_y = 0;
int temp;

int user_x = 20 , user_y = 20;
int prev_puck_x =0, prev_puck_y=0;
int vel_pad_et_x = 0, vel_pad_et_y = 0, prev_et_pos_x =0,
prev_et_pos_y = 0;
int vel_pad = 1, v = 0, collision = 0;
int vel_pad_et = 1;
int vel_puck = 1, vel_puck_x = 0, vel_puck_y = 0;
int d=1, dx=0,dy=0;
float ax=0.0,ay=0.0;
int counter= 10000;
int signx=1, signy=1;
int sx, sy;
float  ratio = 0.0;
extern int x_sign, y_sign;
#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38

#define UDP_PACKET_PAYLOAD (transmit_buffer +
UDP_PACKET_PAYLOAD_OFFSET)



unsigned char transmit_buffer[] = {
          // Ethernet MAC header
          0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC
address
          0x01, 0x60, 0x6E, 0x11, 0x02, 0x0F, // Source MAC address
          0x08, 0x00,                         // Packet Type: 0x800 =
IP

          // IP Header: bytes 14-33
          0x45,                // version (IPv4), header length = 20
bytes
          0x00,                // differentiated services field
          0x00,0x9C,           // total length: 20 bytes for IP header
+
                               // 8 bytes for UDP header + 128 bytes
for payload
          0x00, 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 : 8 bytes
          0x67,0xd9, // source port port (26585: garbage)????
          0x27,0x2b, // destination port (10027: garbage)????
          0x00,0x88, // length (136: 8 for UDP header + 128 for data)
          0x00,0x00, // checksum: 0 = none

          // UDP payload
          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
        };



static void ethernet_interrupt_handler()
{
      unsigned int receive_status;
      int i;
      receive_status = ReceivePacket(receive_buffer,
&receive_buffer_length);
      if (receive_status == DMFE_SUCCESS)
      {
                if (receive_buffer_length >= 14)
           {

                if (receive_buffer[12] == 8 && receive_buffer[13] == 0
&& receive_buffer_length >= 34)
                {
                           // An IP packet
                           if (receive_buffer[23] == 0x11)
                      {
                           // A UDP packet
                           if (receive_buffer_length >=
UDP_PACKET_PAYLOAD_OFFSET)
                           {

                                      ethernet_x =
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET]+500;
                                      IOWR_VGA_DATA(VGA_BASE, 2,
ethernet_x);
                                      ethernet_y =
((receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+2] << 8) & 0xffff) +
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+1];
                                      IOWR_VGA_DATA(VGA_BASE, 3,
ethernet_y);
                               }

                           }
                      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 */

      //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);
}


//static unsigned short packetId = 0;
int main()
{





  int curMsgChar = 0;


  unsigned int packet_length;


    // Print a friendly welcome message
  printf("4840 Lab 2 started\n");


  // Initalize the DM9000 and the Ethernet interrupt handler
  DM9000_init(mac_address);


  alt_irq_register(DM9000A_IRQ, NULL,
(void*)ethernet_interrupt_handler);

  // Initialize the keyboard
  printf("Please wait three seconds to initialize mouse\n");
  clear_FIFO();


  switch (get_mode()) {
  case PS2_KEYBOARD:
    printf("Error: Keyboard detected on PS/2 port\n");
    goto ErrorExit;
    break;
  case PS2_MOUSE:
        printf("Mouse Detected!\n");
        init();
       IOWR_VGA_DATA(VGA_BASE, 2, ethernet_x);
       IOWR_VGA_DATA(VGA_BASE, 3, ethernet_y);
        IOWR_VGA_DATA(VGA_BASE, 8, score2);
        IOWR_VGA_DATA(VGA_BASE, 7, score1);
     break;
  default:
    printf("Error: Unrecognized or no device on PS/2 port\n");
    goto ErrorExit;
  }

  printf("Ready!!!");

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


for (;;) {



      capture_mouse(&user_x , &user_y);


      IOWR_VGA_DATA(VGA_BASE, 0, user_x);
      IOWR_VGA_DATA(VGA_BASE, 1, user_y);

      IOWR_VGA_DATA(VGA_BASE, 4, puck_x);
      IOWR_VGA_DATA(VGA_BASE, 5, puck_y);

      puck_x=puck_x-xdiff;
      puck_y=puck_y-ydiff;

     if (((ethernet_x-puck_x)*(ethernet_x-puck_x) + ((ethernet_y-
puck_y)*(ethernet_y-puck_y))) <= 400)
      {
       collision_ethernet=1;
      }
     temp =  ( ((puck_x-user_x)*(puck_x-user_x)) + ((puck_y-
user_y)*(puck_y-user_y))  );

     if (temp <= 400)
     {
      collision = 1;
     }

     if( (puck_y>=212 && puck_y<=252) && (puck_x <= 11 || puck_x >=
629))      //when ball goes in the goal
       {
           puck_y = 232;
           xdiff=  ydiff = 4;
        if ( puck_x <= 11)
         {                                   //update player2 scores
if puck goes in player1 goal
          score2++;
          IOWR_VGA_DATA(VGA_BASE, 8, score2);
         }
        else if (puck_x >= 629)
         {                                   //update player1 scores
if puck goes in player2 goal
          score1++;
          IOWR_VGA_DATA(VGA_BASE, 7, score1);
         }
        puck_x = 320;
       }


     if(puck_y>454)
      {
       ydiff=ydiff*(-1);
       xdiff=xdiff;
      }
      else if (puck_x>629)
      {
       xdiff=xdiff*(-1);
       ydiff=ydiff;
      }
      else if(puck_y<11)
      {
       ydiff=ydiff*(-1);              // Bounce back code at screen
edges
       xdiff=xdiff;
      }
      else if(puck_x<11)
      {
       xdiff=xdiff*(-1);
       ydiff=ydiff;
      }

  //collision handling
```

```
    if (collision == 1)
      {
       vel_pad_x = user_x - prev_pos_x;
            vel_pad_y = user_y - prev_pos_y;

            d = sqrt(((user_x - puck_x)*(user_x - puck_x)) + ((user_y
- puck_y)*(user_y - puck_y)));  //calculates distance between the two
centres
            dx = (user_x - puck_x);
            dy = (user_y - puck_y);

       vel_pad =  sqrt(((vel_pad_x)*(vel_pad_x)) +
((vel_pad_y)*(vel_pad_y)));   //velocity of paddle


      if (vel_pad == 0 || ( (vel_pad_x == 0) &&  (vel_pad_y == 0)))
//paddle is stationary
           {
           xdiff = xdiff * (-1);
           ydiff = ydiff * (-1);

            }

   // direction modulation of puck depending on the direction of
paddle


      if ( (vel_pad_x > 0 ) && (vel_pad_y > 0) )
       {
           xdiff = ((dx * vel_pad)/(12 *d))*(-1);
           ydiff = ((dy * vel_pad)/(12 *d))*(-1);
       }
      else if ( (vel_pad_x < 0 ) && (vel_pad_y < 0) )
       {
           xdiff = ((dx * vel_pad)/(12 *d));
           ydiff = ((dy * vel_pad)/(12 *d));
      }
      else if ( (vel_pad_x > 0 ) && (vel_pad_y < 0) )
       {
           xdiff = ((dx * vel_pad)/(12 *d));
           ydiff = ((dy * vel_pad)/(12 *d))*(-1);
      }
      else if ( (vel_pad_x < 0 ) && (vel_pad_y > 0) )
       {
            xdiff = ((dx * vel_pad)/(12 *d))*(-1);
            ydiff = ((dy * vel_pad)/(12 *d));
       }
      else
           {
            xdiff = ((dx * vel_pad)/(12 *d))*(-1);
            ydiff = ((dy * vel_pad)/(12 *d))*(-1);
```

```
 }



 if (xdiff == 0 && ydiff == 0)
      {
       xdiff = -4;
       ydiff = -4;
      }

// set minimum speed limits

 if (xdiff!= 0 && xdiff > 0 && xdiff < 4)
       {
       xdiff = 4;
       }

if (ydiff!= 0 && ydiff > 0 && ydiff < 4)
       {
       ydiff = 4;
       }

if (ydiff!= 0 && ydiff < 0 && ydiff > -4)
      {
       ydiff = -4;
      }

if (xdiff!= 0 && xdiff < 0 && xdiff > -4)
      {
       xdiff = -4;
      }

// set maximum speed limits

if (xdiff > 10)
      {
       while (xdiff > 4)
            {
            int q;
            for (q = 0; q < 1000; q++ );
             xdiff --;
            }
       }

  if (ydiff > 10)
{
while (ydiff > 4)
{
       int q;
       for ( q = 0; q < 1000; q++ );
       ydiff --;
 }
```

```
      }


      if (xdiff < -10)
      {
       while (xdiff < -4)
      {
             int q;
             for (q = 0; q < 1000; q++ );
             xdiff ++;
       }
      }


      if (ydiff < -10)
      {
       while (ydiff < -4)
      {
       int q;
             for (q = 0; q < 1000; q++ );
             ydiff ++;
       }
      }

      prev_pos_x = user_x;
      prev_pos_y = user_y;

      collision = 0;
 }

      //collision handling for second paddle

     if (collision_ethernet == 1)
     {

         vel_pad_et_x = ethernet_x - prev_et_pos_x;
         vel_pad_et_y = ethernet_y - prev_et_pos_y;

         d = sqrt(((ethernet_x - puck_x)*(ethernet_x - puck_x)) +
((ethernet_y - puck_y)*(ethernet_y - puck_y)));
         dx = (ethernet_x - puck_x);
         dy = (ethernet_y - puck_y);

         vel_pad_et =  sqrt(((vel_pad_et_x)*(vel_pad_et_x)) +
((vel_pad_et_y)*(vel_pad_et_y)));

         if (vel_pad_et == 0 && ((vel_pad_x == 0) &&  (vel_pad_y ==
0)))
         {

       xdiff = xdiff * (-1);
       ydiff = ydiff * (-1);
```

```
   }


   if ( (vel_pad_et_x > 0 ) && (vel_pad_et_y > 0) )
   {

       xdiff = ((dx * vel_pad_et)/(12 *d))*(-1);
       ydiff = ((dy * vel_pad_et)/(12 *d))*(-1);
    }
   else if ( (vel_pad_et_x < 0 ) && (vel_pad_et_y < 0) )
   {

       xdiff = ((dx * vel_pad_et)/(12 *d));
       ydiff = ((dy * vel_pad_et)/(12 *d));
   }
   else if ( (vel_pad_et_x > 0 ) && (vel_pad_et_y < 0) )
   {
       xdiff = ((dx * vel_pad_et)/(12 *d));
       ydiff = ((dy * vel_pad_et)/(12 *d))*(-1);
   }
   else if ( (vel_pad_et_x < 0 ) && (vel_pad_et_y > 0) )
   {
       xdiff = ((dx * vel_pad_et)/(12 *d))*(-1);
       ydiff = ((dy * vel_pad_et)/(12 *d));
   }
   else
   {
       xdiff = ((dx * vel_pad_et)/(12 *d))*(-1);
       ydiff = ((dy * vel_pad_et)/(12 *d))*(-1);
   }





  if (xdiff == 0 && ydiff == 0)
      {
       xdiff = -4;
       ydiff = -4;
      }

 // set minimum speed limits

  if (xdiff!= 0 && xdiff > 0 && xdiff < 4)
       {
       xdiff = 4;
       }

 if (ydiff!= 0 && ydiff > 0 && ydiff < 4)
       {
       ydiff = 4;
       }

 if (ydiff!= 0 && ydiff < 0 && ydiff > -4)
      {
       ydiff = -4;
      }

 if (xdiff!= 0 && xdiff < 0 && xdiff > -4)
      {
       xdiff = -4;
      }

 // set maximum speed limits

 if (xdiff > 10)
      {
       while (xdiff > 4)
            {
            int q;
            for (q = 0; q < 1000; q++ );
             xdiff --;
            }
       }

   if (ydiff > 10)
 {
 while (ydiff > 4)
 {
       int q;
       for ( q = 0; q < 1000; q++ );
       ydiff --;
  }
 }


 if (xdiff < -10)
 {
  while (xdiff < -4)
 {
       int q;
       for (q = 0; q < 1000; q++ );
       xdiff ++;
  }
 }


 if (ydiff < -10)
 {
  while (ydiff < -4)
 {
  int q;
       for (q = 0; q < 1000; q++ );
```

```
                ydiff ++;
          }
         }


            prev_et_pos_x = ethernet_x;
            prev_et_pos_y = ethernet_y;

            collision_ethernet = 0;
        }


        //corner cases for puck hitting the screen

        if( puck_x >= 630 || puck_x <= 10 )
         {
           puck_x = prev_puck_x;

         }

        if( puck_y >= 458|| puck_y <= 10 )
         {
          puck_y = prev_puck_y;

         }


        // game over condition
        if(score1==8 || score2==8)
         {
            exit(0);
         }

          UDP_PACKET_PAYLOAD[curMsgChar++]=user_x & 0xFF;
          UDP_PACKET_PAYLOAD[curMsgChar++]=user_y & 0xFF;
          UDP_PACKET_PAYLOAD[curMsgChar++]=(user_y>>8) & 0xFF;

          UDP_PACKET_PAYLOAD[curMsgChar++]= (puck_x & 0xFF);
          UDP_PACKET_PAYLOAD[curMsgChar++]=(puck_x>>8) & 0xFF;
          UDP_PACKET_PAYLOAD[curMsgChar++]=puck_y & 0xFF;
          UDP_PACKET_PAYLOAD[curMsgChar++]=(puck_y>>8) & 0xFF;

          UDP_PACKET_PAYLOAD[curMsgChar++]=score1 & 0xFF;
          UDP_PACKET_PAYLOAD[curMsgChar++]=score2 & 0xFF;
          UDP_PACKET_PAYLOAD[curMsgChar] = 0; // Terminate the string

          packet_length = 8 + curMsgChar;
          transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
          transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length &
0xff;
         TransmitPacket(transmit_buffer, UDP_PACKET_PAYLOAD_OFFSET +
curMsgChar + 1);
```

```
     // reset data
     for (curMsgChar=12; curMsgChar>0; curMsgChar--)
     {
       UDP_PACKET_PAYLOAD[curMsgChar] = 0;
     }

     prev_puck_x = puck_x;
     prev_puck_y = puck_y;

}//end for
  return 0;
  ErrorExit:
  printf("Program terminated with error condition\n");
  exit(0);
}
```

8.2.2 Slave-Main.c

```
//
// Slave-main.c
// Team NAHVG
//

#include "basic_io.h"
#include "DM9000A.h"
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "ps2_mouse.h"
#include "VGA.h"
#include <string.h>
#define IOWR_VGA_DATA(base, offset, data) \
IOWR_16DIRECT(base, (offset) * 2, data)
#define MAX_MSG_LENGTH 128


// Ethernet MAC address.  Choose the last three bytes yourself
unsigned char mac_address[6] = { 0x01, 0x60, 0x6E, 0x11, 0x01, 0x0E
};

unsigned int interrupt_number;

unsigned int receive_buffer_length;
unsigned char receive_buffer[1600];
unsigned long checksum;
int data_mouse;
int ctrl_mouse;
int bytes_mouse;
int data_reg1;
int puck_x = 320,puck_y = 232;


#define UDP_PACKET_PAYLOAD_OFFSET 42
#define UDP_PACKET_LENGTH_OFFSET 38

#define UDP_PACKET_PAYLOAD (transmit_buffer +
UDP_PACKET_PAYLOAD_OFFSET)
unsigned char transmit_buffer[] = {
            // Ethernet MAC header
         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC
address
         0x01, 0x60, 0x6E, 0x11, 0x01, 0x0E, // Source MAC address
         0x08, 0x00,
         // IP Header: bytes 14-33
         0x45,                // version (IPv4), header length = 20
bytes
         0x00,                // differentiated services field
         0x00,0x9C,           // total length: 20 bytes for IP header
+
                              // 8 bytes for UDP header + 128 bytes
for payload
         0x00, 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 : 8 bytes
         0x67,0xd9, // source port port (26585: garbage)????
         0x27,0x2b, // destination port (10027: garbage)????
         0x00,0x88, // length (136: 8 for UDP header + 128 for data)
         0x00,0x00, // checksum: 0 = none
         // UDP payload
         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
     };




static void ethernet_interrupt_handler() {
    unsigned int receive_status;
    //printf("running ethernet interrupt routing\n");
    receive_status = ReceivePacket(receive_buffer,
&receive_buffer_length);
    if (receive_status == DMFE_SUCCESS) {
       if (receive_buffer_length >= 14) {
            //  A real Ethernet packet
            //printf("len=%d 12 = %d
13=%d\n",receive_buffer_length,receive_buffer[12],receive_buffer[13]);
           if (receive_buffer[12]==8 && receive_buffer[13]==0 &&
receive_buffer_length>=34){
                // An IP packet
                if (receive_buffer[23] == 0x11) {
                    // A UDP packet
                    if (receive_buffer_length >=
UDP_PACKET_PAYLOAD_OFFSET) {
                       IOWR_VGA_DATA(VGA_BASE, 2,
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET]);
                       int y =
((receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+2] << 8) & 0xffff) +
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+1];
                       IOWR_VGA_DATA(VGA_BASE, 3, y);
                       puck_x =
((receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+4] << 8) & 0xffff) +
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+3];
                       IOWR_VGA_DATA(VGA_BASE, 4, puck_x);
                       puck_y =
((receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+6] << 8) & 0xffff) +
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+5];
                       IOWR_VGA_DATA(VGA_BASE, 5, puck_y);
                       IOWR_VGA_DATA(VGA_BASE, 7,
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+7]);
                       IOWR_VGA_DATA(VGA_BASE, 8,
receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+8]);
                      // printf("p1 = %d p2 =
%d",receive_buffer[UDP_PACKET_PAYLOAD_OFFSET+7],receive_buffer[UDP_PAC
KET_PAYLOAD_OFFSET+8]);
                    }
               } 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);
}//end of interrupt handler

int main(){

 int user_x = 20 , user_y = 20;
 int prev_puck_x =0, prev_puck_y=0;

 int curMsgChar = 0;
 unsigned int packet_length=0;
  // Print a friendly welcome message
 printf("4840 Lab 2 started\n");

 // Initalize the DM9000 and the Ethernet interrupt handler
 DM9000_init(mac_address);
 interrupt_number = 0;

alt_irq_register(DM9000A_IRQ, NULL,
(void*)ethernet_interrupt_handler);

 // Initialize the mouse
 printf("Please wait three seconds to initialize mouse\n");
 clear_FIFO(); //clear for PS2 device
 switch (get_mode()) {
 case PS2_KEYBOARD:
        printf("Error: Keyboard detected on PS/2 port\n");
   goto ErrorExit;
   break;
 case PS2_MOUSE:
       printf("Mouse Detected!\n");
       init();
    break;
 default:
   printf("Error: Unrecognized or no device on PS/2 port\n");
   goto ErrorExit;
 }

 printf("Ready!!!!\n");
 // Clear the payload
 for (curMsgChar=MAX_MSG_LENGTH-1; curMsgChar>0; curMsgChar--)
 {
   UDP_PACKET_PAYLOAD[curMsgChar] = 0;
 }

for (;;){

     capture_mouse(&user_x , &user_y);

     IOWR_VGA_DATA(VGA_BASE, 0, user_x+500);
     IOWR_VGA_DATA(VGA_BASE, 1, user_y);

     UDP_PACKET_PAYLOAD[curMsgChar++]=user_x & 0xFF;
     UDP_PACKET_PAYLOAD[curMsgChar++]=user_y & 0xFF;
     UDP_PACKET_PAYLOAD[curMsgChar++]=(user_y>>8) & 0xFF;
     UDP_PACKET_PAYLOAD[curMsgChar] = 0; // Terminate the string

     packet_length = 8 + curMsgChar;
     transmit_buffer[UDP_PACKET_LENGTH_OFFSET] = packet_length >> 8;
     transmit_buffer[UDP_PACKET_LENGTH_OFFSET + 1] = packet_length &
0xff;


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

 }//end infinite loop
  return 0;
    ErrorExit:
        printf("Program terminated with error condition\n");
        exit(0);
}
```

8.2.3 PS2Mouse.c

```
//
//   Mouse.c
//   Team  NAHVG
//

#include "ps2_mouse.h"
#include<stdio.h>
#include "alt_up_ps2_port.h"
```

```
int m_dir_x=0;
int m_dir_y=0;
int prev_m_dir_x = 0;
int prev_m_dir_y = 0;
#define STATUS_ERROR 0x00
#define STATUS_ACK 0xFA
#define PRESSED  1
#define RELEASED  2

int myread()
{
     int data = 0;
     while( 1 )
     {
          data = read_data_reg();
          if( ( ( data >> 16) & 0xFFFF ) != 0 )
          {
               return data;
          }
     }
}

void write1(int data)
{
    write_data_byte(data);
}

void init()
{
     write1( MOUSE_RESET );
     myread( );
     myread( );
     write1( MOUSE_SET_SAMPLE_RATE );
     myread( );
     write1( 0x28 );             //200
     myread( );
     write1( MOUSE_SET_RESOLUTION );
     myread( );
     write1( 0x02);         // Resolution is 4count/mm
     myread( );
     write1 ( 0xE7 );  // scaling 2:1
     myread( );
     write1( MOUSE_STREAM_MODE );
     // changed to stream mode
     myread( );
}

int update1( )
{
     int x_sign = 0;
     int y_sign = 0;
```

```
      int data[ 10 ] = { 0, };
      // wait for ack

      //write_data_byte(0xEB);
      IOWR(ALT_UP_PS2_BASE, 0, 0xEB);
      while( (myread( ) & 0xFF) != STATUS_ACK );
      int loop = 3;
      while( loop )
      {
            data[ 3 - loop ] = myread();
               if( data[ 3 - loop ] == STATUS_ERROR )
            {
                  return 0;
            }
            if( ( unsigned char ) (data[ 3 - loop ] & 0xFF ) == (
unsigned char ) STATUS_ACK )
                  {
                  loop = 3;
                  }
            else
                  {
                  loop--;
                  }
      } // set button

         if( ( ((data[ 0 ] >> 4 ) & 0x01 == 1 )))
      {
            x_sign = 1;
      }
         if( ( (data[ 0 ] >>5 ) & 0x01 == 1 ))
      {
            y_sign = 1;
      }

         // printf("data[0]=%d data[1]=%d data[2]=%d\n
",data[0],data[1],data[2]);
         m_dir_x = data[1] & 0xFF;
         m_dir_y = data[2] & 0xFF;

      if( x_sign == 0 )
         {
            m_dir_x = 0 - (data[ 1 ] & 0xFF);

         }
      else
      {
            m_dir_x =  (~(data[ 1 ] & 0xFF) + 1 ) & 0xFF;
         }


         if( y_sign == 0 )
         {
            m_dir_y = 0 - ( data[ 2 ] & 0xFF );
         }
         else
         {
            m_dir_y = ( ~( data[ 2 ] & 0xFF ) + 1 ) & 0xFF;
         } // remove mouse's pop data

         if ( ( m_dir_y >= 100 ) || ( m_dir_y <= -100 ) )
      {
            m_dir_y = prev_m_dir_y;
      }
         if ( ( m_dir_x >= 200 ) || ( m_dir_x <= -200 ) )
         {
            m_dir_x = prev_m_dir_x;
      }

         if ( ( m_dir_x == -24 ) || ( m_dir_x == -40 )  || ( m_dir_x ==
-8 ) )
         {
            m_dir_x = prev_m_dir_x;
         }
         if(m_dir_x == -8)
         {
            m_dir_x = 0;
      }
         return 1;
}


void capture_mouse(int *x , int *y)
{
      if (update1())
      {
            //  printf("mdirx =%d\n",m_dir_x);
            *x -= (m_dir_x);
                  *y += m_dir_y;
                  if ( *x> 129)
                        *x = 129;
                  if (*y > 454)
                        *y = 454;
                  if ( *x < 10)
                        *x = 10;
                  if (*y < 10)
                        *y = 10;
                  // printf("dir_x = %d dir_y = %d\n", *x, *y);
      }
      else
            init();
}
```