**FPGA IMPLEMENTATION OF MOUSE INTERFACE**

Batch No: 5

1.DesuAbhinay 21ECB0B13

2.Elikanti Sandeep 21ECB0B14

3.Ganipaka Sujan 21ECB0B15



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY, WARANGAL**

**2022-2023**

**CERTIFICATE**

This is to certify that the dissertation work entitled **FPGA IMPLEMENTATION OF MOUSE INTERFACE** is a bonafied record of work carried out work by Desu Abhinay (21ECB0B13) , Elikanti Sandeep (21ECB0B14) and Ganipaka Sujan (21ECB0B15) submitted to faculty of “Electronics and Communication Engineering Department” , in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in “Electronics and Communication Engineering” at National Institute of Technology, Warangal during academic year (2022-2023).

**Dr. P. Prithvi Dr. V. Narendar**

Assistant Professor Assistant Professor

Department of Electronics and Department of Electronics and

Communication Engineering Communication Engineering

National Institute of Technology National Institute of Technology

Warangal Warangal

**ACKNOWLEDGEMENT**

I would like to express my deepest gratitude to my faculty in-charge **Dr. P. Prithvi,**

**Assistant Professor,** Department of Electronics and Communication Engineering, and **Dr. V. Narendar**, **Assistant Professor**, Department of Electronics and Communication Engineering, National Institute of Technology, Warangal for her constant supervision, guidance, suggestions and encouragement during this project.

ABSTRACT:

PS/2 mouse is commonly used input device to connect to Computer as pointing device. The goal is to interface PS/2 mouse with an ARTIX-7 NEXYS-4 FPGA. We try to design a bidirectional mouse Interface, where FPGA controls all the functions in the project from taking the commands from mouse to providing the information of protocol functioning using LEDs. We will be displaying left scrolling, right scrolling and button clicks on the onboard LEDs of FPGA. All results are obtained by using XILINX VIVADO tool and the circuit will be implemented by using ARTIX-7 NEXYS-4DDR FPGA.

**TABLE OF CONTENTS:**

1. INTRODUCTION 7
2. DESIGN AND ANALYSIS 8
3. IMPLEMENTATION 9

**3.1.** Transmitter – Device to Host communication 9

**3.2** Receiver - Host to Device communication 10

**3.3** Control FSM 12

**3.4** Driver FSM 12

1. VERILOG CODE 13
2. APPLICATIONS 25
3. FPGA IMPLEMENTATION 25
4. CONCLUSION 27

**LIST OF FIGURES:**

Figure 1: Block diagram of mouse interface

Figure 2: Device-to-host communication

Figure 3: Host to device communication flow chart

Figure 4: Device to Host communication flow chart

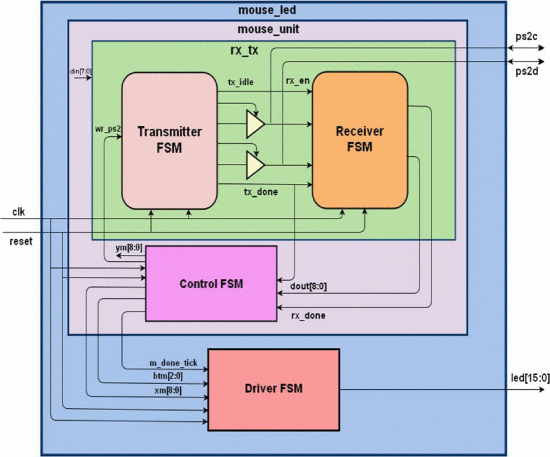
Figure 5: Detailed host-to-device communication.

**1.INTRODUCTION:**

There are many types of pointing devices available for the modern PC including mice, trackballs, touchpad’s, electronic whiteboards, etc. Virtually all of these devices communicate on one of two interfaces: Universal Serial Bus (USB) or the PS/2 mouse interface. Older pointing device interfaces include the Apple Desktop Bus (ADB), RS-232 serial port, and the bus mouse interface.

The PS/2 mouse interface originally appeared in IBM's "Personal System/2" computers in the late 80's and it remains a widely-supported interface. The PS/2 mouse interface utilizes a bidirectional serial protocol to transmit movement and buttonstate. The controller, in turn, may send a number of commands to the mouse to set the report rate, resolution, reset the mouse, disable the mouse, etc. The host provides the mouse with a 5V ~100 mA power supply.

The simple PS/2 mouse interface is derived using FPGA. The functioning of mouse protocol and the interfacing of PS/2 mouse is understood clearly by the output obtained from the LEDs connected to the output pins of FPGA. The left click, right click and x-axis movement from mouse is given to FPGA. Driver FSM of the design increments or decrements or clears the 16 bit registers depending upon the action performed by mouse like movement or left click or right click of the mouse and displays it on the 16 LEDs of FPGA.We have done this project on FPGA system as some have done using microcontrollers. Simulation and synthesis is done by using Xilinx simulation tool.



**2. DESIGN AND ANALYSIS:**

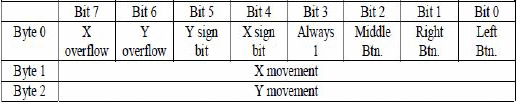
A standard PS/2 mouse reports the x-axis (right/left) and y-axis (up/down) movement and the status of the left button, middle button, and right button. The amount of each movement is recorded in a mouse's internal counter. When the data is transmitted to the host, the counter is cleared to zero and restarts the counting. The content of the counter represents a 9- bit signed integer in which a positive number indicates the right or up movement, and a negative number indicates the left or down movement.

The PS/2 mouse interface utilizes a bidirectional serial protocol to transmit movement and button-state data to the computer's auxiliary device controller (part of the keyboard controller) . The controller, in turn, may send a number of commands to the mouse to set the report rate, resolution, reset the mouse, disable the mouse, etc. The host provides the mouse with a 5V ~100 mA power supply.

The relationship between the physical distances is defined by the mouse's resolution parameter. The default value of resolution is four counts per millimetre. When a mouse moves continuously, the data is transmitted at a regular rate. The rate is defined by the mouse's sampling rate parameter.

The default value of the sampling rate is 100 samples per second. If a mouse moves too fast, the amount of the movement during the sampling period may exceed the maximal range of the counter. The counter is set to the maximum magnitude in the appropriate direction. Two

Overflow bits are used to indicate the conditions. The standard PS/2 mousesendsmovement/buttoninformationtothehost using the 3-byte packet.



**3. IMPLEMENTATION**

The PS/2 mouse implements a bidirectional synchronous serial protocol[7]. The bus is "idle" when both lines are high (open-collector). This is the only state where the mouse is allowed begin transmitting data. The host has ultimate control over the bus and may inhibit communication at any time by pulling the Clock line low.

The device always generates the clock signal. If the host wants to send data, it must first inhibit communication from the device by pulling Clock low. The host then pulls Data low and releases Clock. This is the "Request-to-Send" state and signals the device to start generating clock pulses.

Summary: Bus States

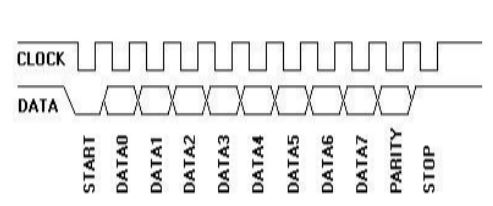
* Data = high, Clock = high: Idle state.
* Data = high, Clock = low: Communication Inhibited.
* Data = low, Clock = high: Host Request-to-Send.

All data is transmitted one byte at a time and each byte is sent in a frame consisting of 11 bits. These bits are:

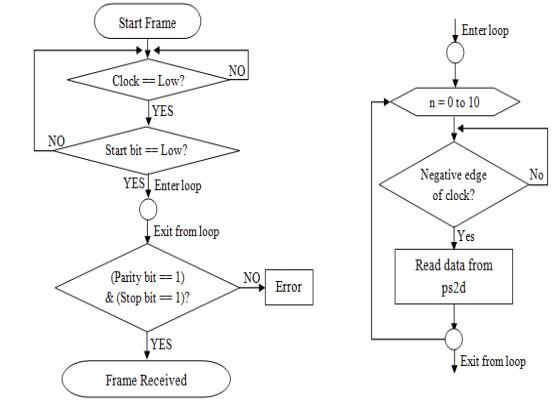
* 1 start bit. This is always 0.
* 8 data bits, least significant bit first.
* 1 parity bit (odd parity).
* 1 stop bit. This is always 1.

**3.1. Transmitter – Device to Host communication**

The device in the description is mouse and host is FPGA board. When the mouse wants to send information, it first checks the Clock line to make sure it's at a high logic level. If it's not, the host is inhibiting communication and the device must buffer any to-be-sent data until the host releases Clock. The Clock line must be continuously high for at least 50 microseconds before the device can begin to transmit its data.



The transmitter performs the function of transmitting data serially from the mouse and received by to the control FSM as a byte of data. The flowchart clearly explains the process of Host to device communication.



**3.2. Receiver - Host to Device communication**

The packet is sent a little differently in host-to-device communication. First of all, the PS/2 device always generates the clock signal.

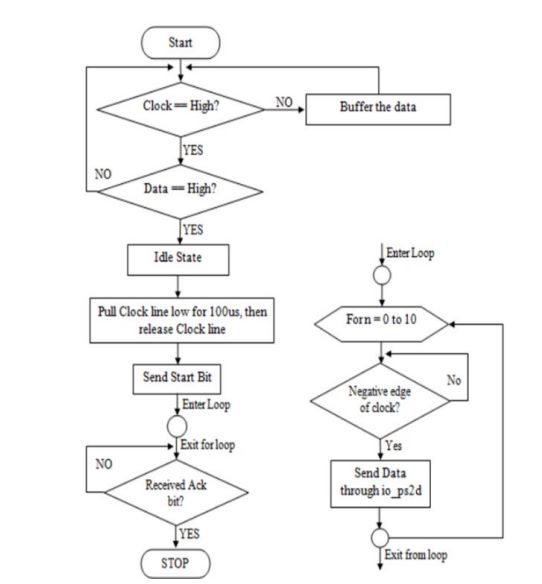
If the host wants to send data, it must first put the Clock and Data lines in a "Request-to-send" state as follows:

* Inhibit communication by pulling Clock low for at least 100 microseconds.
* Apply "Request-to-send" by pulling Data low, then release Clock.

The device should check for this state at intervals not to exceed 10 milliseconds. When the device detects this state, it will begin generating Clock signals and clock in eight data bits and one stop bit. The host changes the Data line only when theClock line is low, and data is read by the device when Clock is high.

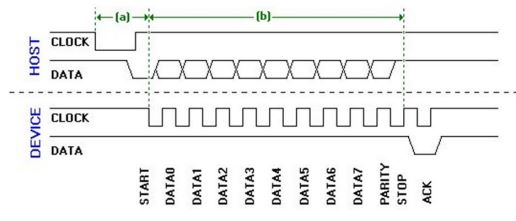
After the stop bit is received, the device will acknowledge the received byte by bringing the Data line low and generating one last clock pulse. If the host does not release the Data line after the 11th clock pulse, the device will continue to generate clock pulses until the Data line is released.

The transmitter modules main function was to transmit data provided to it on the tx\_data line serially to the mouse. There are specific timing guidelines that need to be followed while communicating with the PS/2 mouse. The process and timing for sending a byte of data to the mouse is illustrated in the flowchart.



The host may abort transmission at time before the 11th clock pulse (acknowledge bit) by holding Clock low for at least 100 microseconds.

Figure separates the timing to show which signals are generated by the host, and which are generated by the PS/2 device. Notice the change in timing for the "ack" bit--the data transition occurs when the Clock line is high (rather than when it is low as is the case for the other 11 bits.)



(a) The time it takes the device to begin generating clock pulses after the host initially takes the Clock line low, must be no greater than 15ms.

(b) The time it takes for the packet to be sent must be no greater than 2ms.

If either of these time limits is not met, the host should generate an error. Immediately after the "ack" is received, the host may bring the Clock line low to inhibit communication while it processes data. If the command sent by the host requires a response, that response must be received no later than 20 ms after the host releases the Clock line. If this does not happen, the host generates an error.

Other than sending the data as outlined above the receiver also needed to make sure that when it’s trying to receive data the mouse should not be sending any data. This is made sure in our code by checking that that the clock is held high for enough time without a transition.

**3.3. Control FSM**

It implements the logic to receive the data from receiver FSM and separates the button data and movement data.

**3.4. Driver FSM**

It collects the x-movement data and button data from the control FSM and drives the LEDs on the FPGA board.

**4. VERILOG CODE**

4.1 Transmitter

module MouseTransmitter(

//Standard Inputs

input RESET,

input CLK,

//Mouse IO - CLK

input CLK\_MOUSE\_IN,

output CLK\_MOUSE\_OUT\_EN, // Allows for the control of the Clock line

//Mouse IO - DATA

input DATA\_MOUSE\_IN,

output DATA\_MOUSE\_OUT,

output DATA\_MOUSE\_OUT\_EN,

//Control

input SEND\_BYTE,

input [7:0] BYTE\_TO\_SEND,

output BYTE\_SENT

);

// Clk Mouse delayed to detect clock edges

reg ClkMouseInDly;

always@(posedge CLK)

ClkMouseInDly<= CLK\_MOUSE\_IN;

//Now a state machine to control the flow of write data

reg [3:0] Curr\_State, Next\_State;

reg Curr\_MouseClkOutWE, Next\_MouseClkOutWE;

reg Curr\_MouseDataOut, Next\_MouseDataOut;

reg Curr\_MouseDataOutWE, Next\_MouseDataOutWE;

reg [15:0] Curr\_SendCounter, Next\_SendCounter;

reg Curr\_ByteSent, Next\_ByteSent;

reg [7:0] Curr\_ByteToSend, Next\_ByteToSend;

//Sequential

always@(posedge CLK) begin

if(RESET) begin

Curr\_State<= 4'h0;

Curr\_MouseClkOutWE<= 1'b0;

Curr\_MouseDataOut<= 1'b0;

Curr\_MouseDataOutWE<= 1'b0;

Curr\_SendCounter<= 0;

Curr\_ByteSent<= 1'b0;

Curr\_ByteToSend<= 0;

end else begin

Curr\_State<= Next\_State;

Curr\_MouseClkOutWE<= Next\_MouseClkOutWE;

Curr\_MouseDataOut<= Next\_MouseDataOut;

Curr\_MouseDataOutWE<= Next\_MouseDataOutWE;

Curr\_SendCounter<= Next\_SendCounter;

Curr\_ByteSent<= Next\_ByteSent;

Curr\_ByteToSend<= Next\_ByteToSend;

end

end

always@\* begin

//default values

Next\_State = Curr\_State;

Next\_MouseClkOutWE = 1'b0;

Next\_MouseDataOut = 1'b0;

Next\_MouseDataOutWE = Curr\_MouseDataOutWE;

Next\_SendCounter = Curr\_SendCounter;

Next\_ByteSent = 1'b0;

Next\_ByteToSend = Curr\_ByteToSend;

case(Curr\_State)

//IDLE

4'h0: begin

if(SEND\_BYTE) begin

Next\_State = 4'h1;

Next\_ByteToSend = BYTE\_TO\_SEND;

end

Next\_MouseDataOutWE = 1'b0;

end

//Bring Clock line low for at least 100 microsecs i.e. 5000 clock cycles @ 50MHz

4'h1: begin

Next\_MouseDataOut = 1'b1;

end

//Release Data line

4'h7: begin

Next\_State = 4'h8;

Next\_MouseDataOutWE = 1'b0;

end

4'h8: begin

if(~DATA\_MOUSE\_IN)

Next\_State = 4'h9;

end

//Wait for device to bring Clock line low

4'h9: begin

if(~CLK\_MOUSE\_IN)

Next\_State = 4'hA;

end

//Wait for device to release both Data and Clock lines

4'hA: begin

if(DATA\_MOUSE\_IN & CLK\_MOUSE\_IN) begin

Next\_State = 4'h0;

Next\_ByteSent = 1'b1;

end

end

default: begin

Next\_State<= 4'h0;

Next\_MouseClkOutWE<= 1'b0;

Next\_MouseDataOut<= 1'b0;

Next\_MouseDataOutWE<= 1'b0;

Next\_SendCounter<= 0;

Next\_ByteSent<= 1'b0;

Next\_ByteToSend<= 0;

end

endcase

end

//Assign OUTPUTs

//Mouse IO - CLK

assign CLK\_MOUSE\_OUT\_EN = Curr\_MouseClkOutWE;

//Mouse IO - DATA

assign DATA\_MOUSE\_OUT = Curr\_MouseDataOut;

assign DATA\_MOUSE\_OUT\_EN = Curr\_MouseDataOutWE;

//Control

assign BYTE\_SENT = Curr\_ByteSent;

endmodule

4.2. Receiver

module MouseReceiver(

//Standard Inputs

input RESET,

input CLK,

//Mouse IO - CLK

input CLK\_MOUSE\_IN,

//Mouse IO - DATA

input DATA\_MOUSE\_IN,

//Control

input READ\_ENABLE,

output [7:0] BYTE\_READ,

output [1:0] BYTE\_ERROR\_CODE,

output BYTE\_READY

);

// Clk Mouse delayed to detect clock edges

reg ClkMouseInDly;

always@(posedge CLK)

ClkMouseInDly<= CLK\_MOUSE\_IN;

//A simple state machine to handle the incoming 11-bit codewords

reg [2:0] Curr\_State, Next\_State;

reg [7:0] Curr\_MSCodeShiftReg, Next\_MSCodeShiftReg;

reg [3:0] Curr\_BitCounter, Next\_BitCounter;

reg Curr\_ByteReceived, Next\_ByteReceived;

reg [1:0] Curr\_MSCodeStatus, Next\_MSCodeStatus;

reg [15:0] Curr\_TimeoutCounter, Next\_TimeoutCounter;

//Sequential

always@(posedge CLK) begin

if(RESET) begin

Curr\_State<= 3'b000;

Curr\_MSCodeShiftReg<= 8'h00;

Curr\_BitCounter<= 0;

Curr\_ByteReceived<= 1'b0;

Curr\_MSCodeStatus<= 2'b00;

Curr\_TimeoutCounter<= 0;

end else begin

Curr\_State<= Next\_State;

Curr\_MSCodeShiftReg<= Next\_MSCodeShiftReg;

Curr\_BitCounter<= Next\_BitCounter;

Curr\_ByteReceived<= Next\_ByteReceived;

Curr\_MSCodeStatus<= Next\_MSCodeStatus;

Curr\_TimeoutCounter<= Next\_TimeoutCounter;

end

end

always@\* begin

//defaults to make the State Machine more readable

Next\_State = Curr\_State;

Next\_MSCodeShiftReg = Curr\_MSCodeShiftReg;

Next\_BitCounter = Curr\_BitCounter;

Next\_ByteReceived = 1'b0;

Next\_MSCodeStatus = Curr\_MSCodeStatus;

Next\_TimeoutCounter = Curr\_TimeoutCounter + 1'b1;

//The states

case (Curr\_State)

3'b000: begin

//Falling edge of Mouse clock and MouseData is low i.e. start bit

if(READ\_ENABLE &ClkMouseInDly& ~CLK\_MOUSE\_IN & ~DATA\_MOUSE\_IN) begin

Next\_State = 3'b001;

Next\_MSCodeStatus = 2'b00;

end

Next\_BitCounter = 0;

end

//Read successive byte bits from the mouse here

3'b001: begin

if(Curr\_TimeoutCounter == 50000) // 1ms timeout

Next\_State = 3'b000;

else if(Curr\_BitCounter == 8) begin // if last bit go to parity bit check

Next\_State = 3'b010;

Next\_BitCounter = 0;

end else if(ClkMouseInDly& ~CLK\_MOUSE\_IN) begin //Shift Byte in

if(Curr\_TimeoutCounter == 100000) // 1ms timeout for 100MHz Clock

Next\_State = 3'b000;

if (CLK\_MOUSE\_IN & DATA\_MOUSE\_IN) begin

Next\_ByteReceived = 1'b1;

Next\_State = 3'b000;

end

end

//DEFAUlT STATE

default: begin

Next\_State = 3'b000;

Next\_MSCodeShiftReg = 8'h00;

Next\_BitCounter = 0;

Next\_ByteReceived = 1'b0;

Next\_MSCodeStatus = 2'b00;

Next\_TimeoutCounter = 0;

end

endcase

end

assign BYTE\_READY = Curr\_ByteReceived;

assign BYTE\_READ = Curr\_MSCodeShiftReg;

assign BYTE\_ERROR\_CODE = Curr\_MSCodeStatus;

endmodule

4.3. Transceiver

module MouseTransceiver(

//Standard Inputs

input RESET,

input CLK,

//IO - Mouse side

inout CLK\_MOUSE,

inout DATA\_MOUSE,

// Mouse data information

// Added reg declaration for MouseStatus, MouseX and MouseY as it is necessary to do

// non-blocking assignments

output reg [3:0] MouseStatus,

output reg [7:0] MouseX,

output reg [7:0] MouseY,

output reg [7:0] MouseScroll,

// Seg7 Display

output [3:0] SEG\_SELECT,

output [7:0] LED\_OUT

);

// X, Y Limits of Mouse Position e.g. VGA Screen with 160 x 120 resolution

parameter [7:0] MouseLimitX = 160;

parameter [7:0] MouseLimitY = 120;

// MouseScroll limit was set to 255 to fit in a byte

parameter [7:0] MouseLimitScroll = 255;

/////////////////////////////////////////////////////////////////////

//TriState Signals

//Clk

reg ClkMouseIn;

wire ClkMouseOutEnTrans;

//Data

wire DataMouseIn;

wire DataMouseOutTrans;

wire DataMouseOutEnTrans;

//Clk Output - can be driven by host or device

assign CLK\_MOUSE = ClkMouseOutEnTrans ? 1'b0 : 1'bz;

//Clk Input

assign DataMouseIn = DATA\_MOUSE;

//Clk Output - can be driven by host or device

assign DATA\_MOUSE = DataMouseOutEnTrans ?DataMouseOutTrans : 1'bz;

/////////////////////////////////////////////////////////////////////

//This section filters the incoming Mouse clock to make sure that

//it is stable before data is latched by either transmitter

//or receiver modules

reg [7:0]MouseClkFilter;

always@(posedge CLK) begin

if(RESET)

ClkMouseIn<= 1'b0;

else begin

//A simple shift register

MouseClkFilter[7:1] <= MouseClkFilter[6:0];

MouseClkFilter[0] <= CLK\_MOUSE;

//falling edge

if(ClkMouseIn& (MouseClkFilter == 8'h00))

ClkMouseIn<= 1'b0;

//rising edge

else if(~ClkMouseIn& (MouseClkFilter == 8'hFF))

ClkMouseIn<= 1'b1;

end

end

///////////////////////////////////////////////////////

//Instantiate the Transmitter module

wire SendByteToMouse;

wire ByteSentToMouse;

wire [7:0] ByteToSendToMouse;

MouseTransmitterT(

//Standard Inputs

.RESET (RESET),

.CLK(CLK),

//Mouse IO - CLK

.CLK\_MOUSE\_IN(ClkMouseIn),

.CLK\_MOUSE\_OUT\_EN(ClkMouseOutEnTrans),

//Mouse IO - DATA

.DATA\_MOUSE\_IN(DataMouseIn),

.DATA\_MOUSE\_OUT(DataMouseOutTrans),

.DATA\_MOUSE\_OUT\_EN(DataMouseOutEnTrans),

//Control

.SEND\_BYTE(SendByteToMouse),

.BYTE\_TO\_SEND(ByteToSendToMouse),

.BYTE\_SENT(ByteSentToMouse)

);

///////////////////////////////////////////////////////

//Instantiate the Receiver module

wire ReadEnable;

wire [7:0] ByteRead;

wire [1:0] ByteErrorCode;

wire ByteReady;

MouseReceiverR(

//Standard Inputs

.RESET(RESET),

.CLK(CLK),

//Mouse IO - CLK

.CLK\_MOUSE\_IN(ClkMouseIn),

//Mouse IO - DATA

.DATA\_MOUSE\_IN(DataMouseIn),

//Control

.READ\_ENABLE (ReadEnable),

.BYTE\_READ(ByteRead),

.BYTE\_ERROR\_CODE(ByteErrorCode),

.BYTE\_READY(ByteReady)

);

//Instantiate the Master State Machine module

wire [7:0] MouseStatusRaw;

wire [7:0] MouseDxRaw;

wire [7:0] MouseDyRaw;

wire [7:0] MouseDScrollRaw;

wire SendInterrupt;

MouseMasterSMMSM(

//Standard Inputs

.RESET(RESET),

.CLK(CLK),

//Transmitter Interface

.SEND\_BYTE(SendByteToMouse),

.BYTE\_TO\_SEND(ByteToSendToMouse),

.BYTE\_SENT(ByteSentToMouse),

//Receiver Interface

.READ\_ENABLE (ReadEnable),

.BYTE\_READ(ByteRead),

.BYTE\_ERROR\_CODE(ByteErrorCode),

.BYTE\_READY(ByteReady),

//Data Registers

.MOUSE\_STATUS(MouseStatusRaw),

.MOUSE\_DX(MouseDxRaw),

.MOUSE\_DY(MouseDyRaw),

.MOUSE\_DSCROLL(MouseDScrollRaw),

.SEND\_INTERRUPT(SendInterrupt)

/\*

//Pre-processing - handling of overflow and signs.

//More importantly, this keeps tabs on the actual X/Y

//location of the mouse.

wire signed [8:0] MouseDx;

wire signed [8:0] MouseDy;

wire signed [8:0] MouseDScroll;

wire signed [8:0] MouseNewX;

wire signed [8:0] MouseNewY;

wire signed [8:0] MouseNewScroll;

if Movement Overflow bit is 1 {

if Sign bit is negative {

MouseDx = -256 ( {1, 00000000} )

} else if Sign bit is positive {

MouseDx = 255 ( {0, 11111111} )

}

} else if Movement Overflow bit is 0 {

// Receive incoming dx value and append sign bit because of 2's complement

MouseDx = {Sign bit, MouseDxRaw[7:0]}

}

\*/

assign MouseDx = (MouseStatusRaw[6]) ? (MouseStatusRaw[4] ? {MouseStatusRaw[4],8'h00} :

{MouseStatusRaw[4],8'hFF} ) : {MouseStatusRaw[4],MouseDxRaw[7:0]};

assign MouseDy = (MouseStatusRaw[7]) ? (MouseStatusRaw[5] ? {MouseStatusRaw[5],8'h00} :

{MouseStatusRaw[5],8'hFF} ) : {MouseStatusRaw[5],MouseDyRaw[7:0]};

// Assign proper expression to MouseDScroll.

// Since the mouse scroll data only changes the 4 least significant bits,

// sign extension can be safely used.

assign MouseDScroll = {MouseDScrollRaw[3], MouseDScrollRaw[7:0]};

// calculate new mouse position

assign MouseNewX = {1'b0,MouseX} + MouseDx;

assign MouseNewY = {1'b0,MouseY} + MouseDy;

// calculate new scroll wheel value

assign MouseNewScroll = {1'b0,MouseScroll} + MouseDScroll;

always@(posedge CLK) begin

if(RESET) begin

MouseStatus<= 0;

MouseX<= MouseLimitX/2;

MouseY<= MouseLimitY/2;

MouseScroll<= MouseLimitScroll/2;

end else if (SendInterrupt) begin

//Status is stripped of all unnecessary info

MouseStatus<= MouseStatusRaw[3:0];

//X is modified based on DX with limits on max and min

if(MouseNewX< 0)

MouseX<= 0;

else if(MouseNewX> (MouseLimitX-1))

MouseX<= MouseLimitX-1;

else

MouseX<= MouseNewX[7:0];

// If new Mouse Position is less than 0

if(MouseNewY< 0)

// Set MouseY to minimum screen Y position

MouseY<= 0;

// If new Mouse Position is greater than max screen Y value

else if(MouseNewY> (MouseLimitY-1))

// Set MouseY to maximum screen Y position

MouseY<= MouseLimitY-1;

// If not outside boundary, receive new Mouse Y position value

else

MouseY<= MouseNewY[7:0];

// Scroll value is modified based on DScroll with limits on its max and min value

// Unlike the mouse coordinates, the scroll value isn't capped at the max and min value,

// thus code to make it wrap around has been added.

if(MouseNewScroll< 0)

MouseScroll<= MouseLimitScroll;

else if(MouseNewScroll>MouseLimitScroll)

MouseScroll<= 0;

else

MouseScroll<= MouseNewScroll[7:0];

end

end

endmodule

4.4 Top module

module TOP(

input RESET, input CLK,

inout CLK\_MOUSE,

inout DATA\_MOUSE,

output [3:0] MouseStatus,

output [7:0] MouseScroll,

output [3:0] SEG\_SELECT,

output [7:0] LED\_OUT

);

// MouseX and MouseY direction values

wire [7:0] MouseX;

wire [7:0] MouseY;

MouseTransceiver T (

//Standard Inputs

.RESET(RESET),

.CLK(CLK),

//IO - Mouse side

.CLK\_MOUSE(CLK\_MOUSE),

.DATA\_MOUSE(DATA\_MOUSE),

// Mouse data information

.MouseStatus(MouseStatus),

.MouseX(MouseX),

.MouseY(MouseY),

.MouseScroll(MouseScroll),

.SEG\_SELECT(SEG\_SELECT),

.LED\_OUT(LED\_OUT)

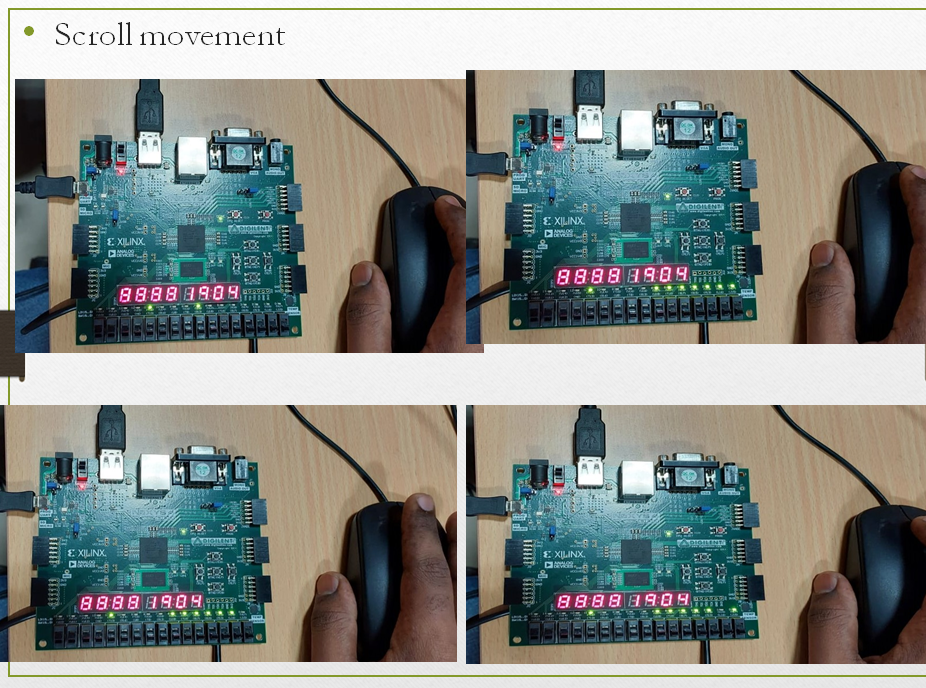
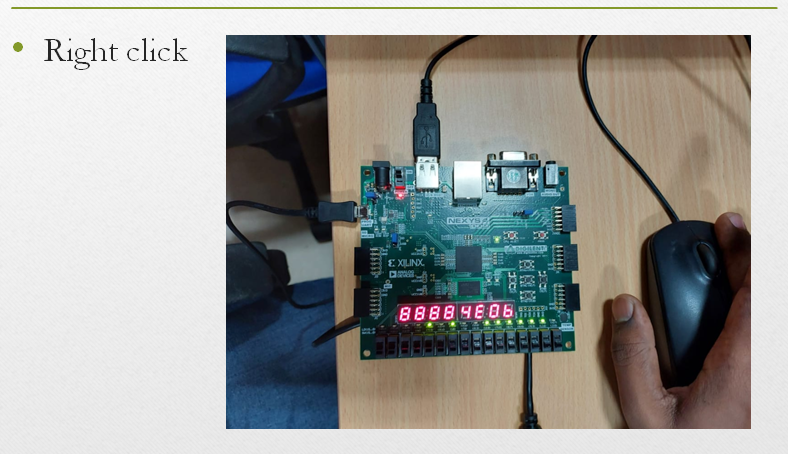
);

Endmodule

**5. APPLICATIONS**

1. Gaming: Many gamers prefer to use a PS/2 mouse for gaming due to its faster response time and more reliable connection compared to USB.
2. Industrial and embedded systems: PS/2 interfaces are commonly used in industrial and embedded systems due to their reliability, durability, and simplicity.
3. Legacy systems: Older computers may not have USB ports, making a PS/2 mouse the only option for input devices.
4. Kiosk systems: PS/2 interfaces are also commonly used in kiosk systems where a touch screen or a keyboard and mouse may be required.
5. Security systems: PS/2 interfaces are used in some security systems, such as biometric access control systems, where a mouse may be used for input.

**6. FPGA Implementation**



**7. CONCLUSION**

The mouse interface using FPGA was successfully implemented and studied. The mouse is an inexpensive two directions movement sensor that makes this device adequate to be used in certain applications of electronics, mechatronics and so on. If an application wants to use more such sensors, then they must be connected to a system via an interface. This paper presents such an interface to connect the mouse. It is true that the use of FPGA is a costly affair particularly for small industries but when it is implemented through ASIC in large units then the cost will be less.

We can use other devices like Optical mouse, PS/2 keyboard, UART and many more to understand their functioning algorithms and to interface them with FPGA.