“INTERFACE A MOUSE WITH FPGA BORD”

Special Assignment Report

Course on

3EC3109 Advanced Digital System Design

By

Devarsh Jani
(21MECE03)

Department of Electronics and Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

October 2021

Abstract

Mouse is in use commonly used input device to connect to computer as pointing device. The main is to interface USB with p/s2 interface mouse with an BASYS-3 FPGA BOARD. I designed 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 and seven Segment Display. I have displayed left scrolling, right scrolling and button clicks on the onboard LEDs of FPGA. All results are obtained by using XILINX ISE tool and the circuit implemented by using BASYS-3 FPGA.

INDEX

1. Introduction

1.1 Introduction/ Prologue/Background

There are many types of pointing devices available for the modern PC including mice, touchpads, electronic whiteboards, electronic pen etc. Virtually all of these devices communicate on one of two interfaces: Universal Serial Bus (USB) or the P/S2 mouse interface. Older pointing device interfaces include the Apple Desktop Bus (ADB), RS-232 serial port, and the bus mouse interface.

The microcontroller's PS2_CLK and PS2_DAT signals are used to implement a USB interface for communication with a mouse or keyboard. The other signals are to program the FPGA from a USB pen drive connected to the USB-HID port (J2) when the JP1 programming mode jumper is set to "USB".

In this project, we only focus on the USB interface of the onboard microcontroller for communication with a mouse. In this case, the microcontroller behaves like a PS/2 bus and a mouse that connected to the USB-HID port can use the two-wire (clock and data) serial bus (PS/2 protocol) to communicate with the FPGA as the host. The mouse sends clock and data signals to the FPGA.

The simple USB mouse interface is derived using FPGA. The functioning of mouse protocol and the interfacing of USB 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 LEDs of FPGA kit. We have done this project on FPGA system as some have done using microcontrollers. Simulation and synthesis is done by using Xilinx simulation tool.

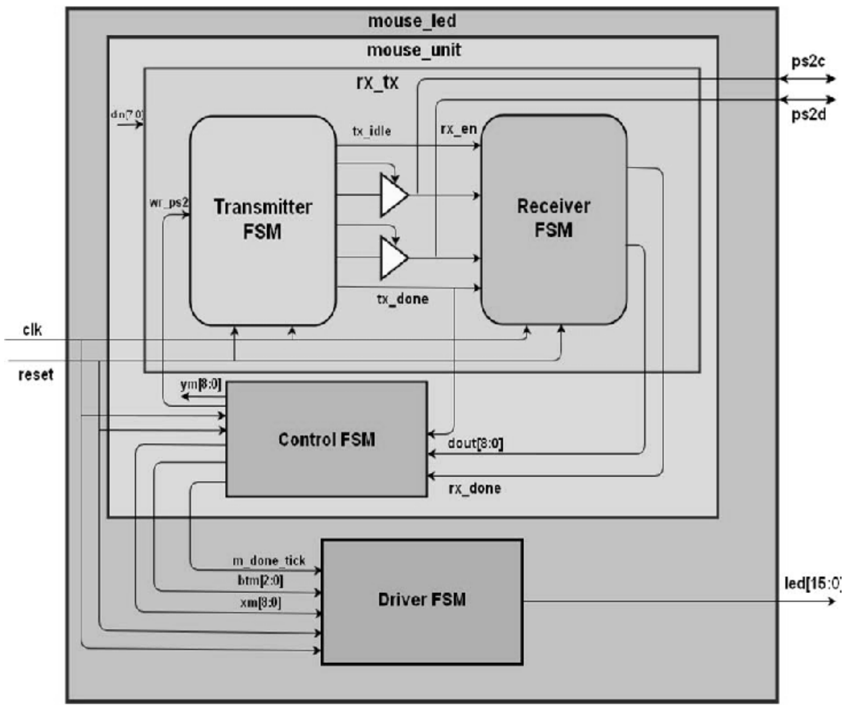
1.2 DESIGN AND ANALYSYS

A standard usb mouse reports the x-axis (right/left) and yaxis (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 usb 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.

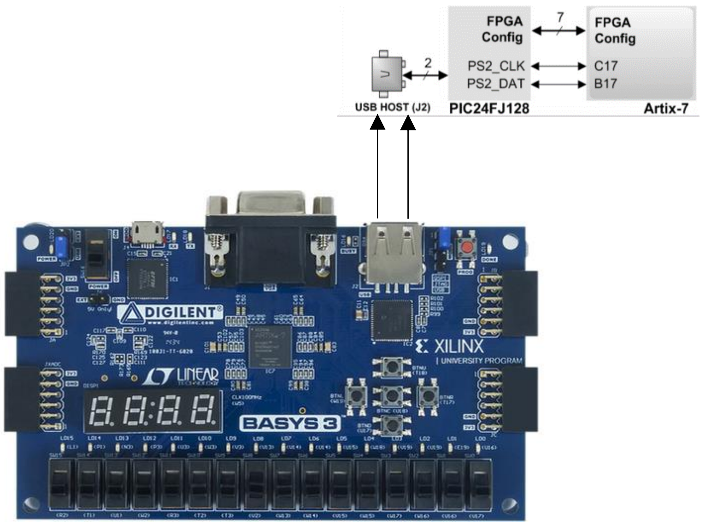
The relationship between the physical distances is defined by the mouse's resolution parameter. The default value of resolution is four counts per millimetre. When a mouse moves continuously, the data is transmitted at a regular rate. The rate is defined by the mouse's sampling rate parameter.

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.



1.3 Interfacing of mouse with bord

The on-board Auxiliary Function Microcontroller PIC24FJ128 allows Basys 3 FPGA to have USB HID host capability. Once the FPGA is programmed, the microcontroller is in USB HID host mode which can interface the FPGA with a mouse or keyboard connected to the USB type-A connector (J2) as shown in the following picture.



2. Literature Review

1.1 Implementation of the mouse interface

The USB mouse implements a bidirectional synchronous serial protocol. 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.

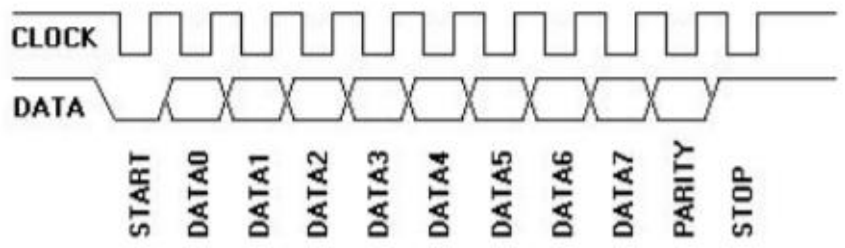
The device always generates the clock signal. If the host wants to send data, it must first inhibit communication from the device by pulling Clock low. The host then pulls Data low and releases Clock. This is the "Request-to-Send" state and signals the device to start generating clock pulses.

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

(A) Device-to-Host Communication (Transmitter):

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.



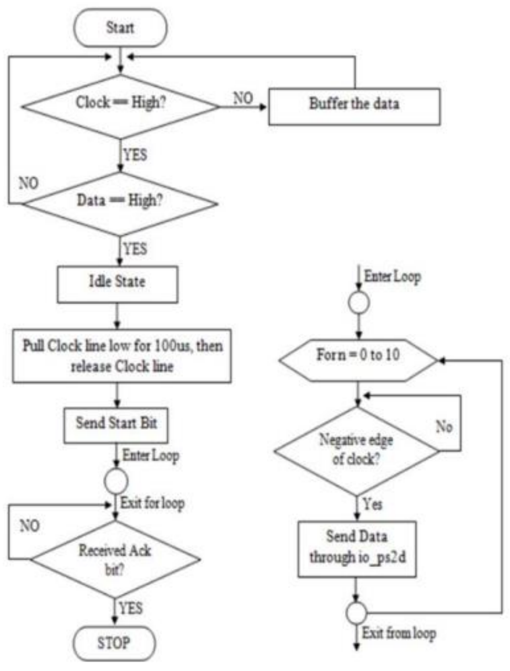


Figure: host to device flowchart.

(B) Host-to-Device Communication (Receiver):

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

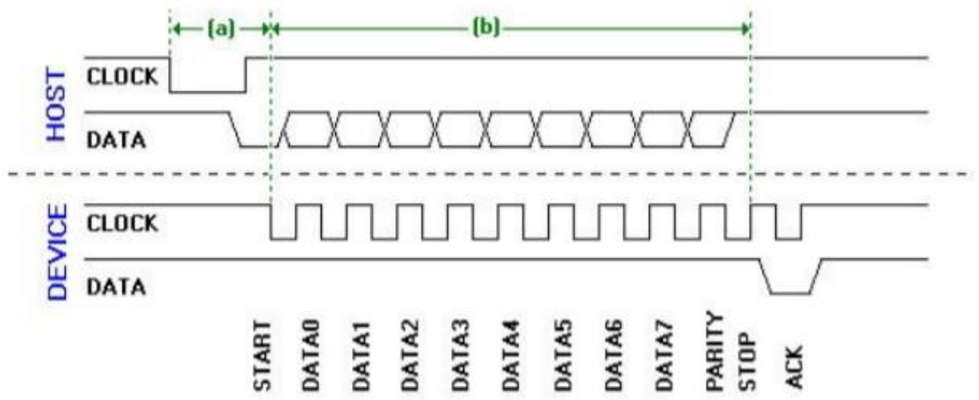
If the host wants to send data, it must first put the Clock and Data lines in a "Request-to-send" state as follows:

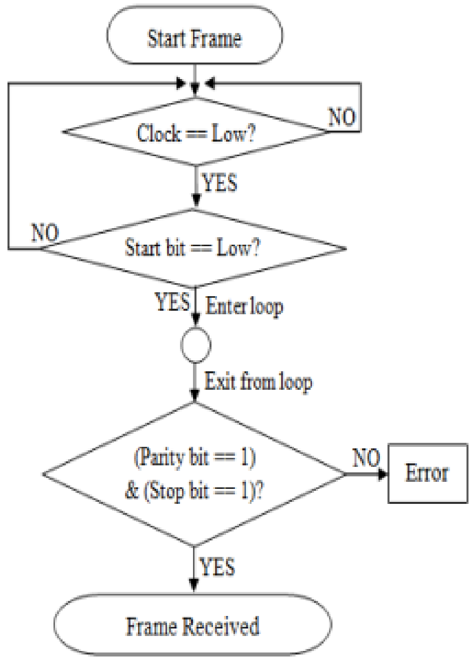
- Inhibit communication by pulling Clock low for at least 100 microseconds.
- Apply "Request-to-send" by pulling Data low, then release Clock.

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 the Clock 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 USB mouse. The process and timing for sending a byte of data to the mouse is illustrated in the flowchart.





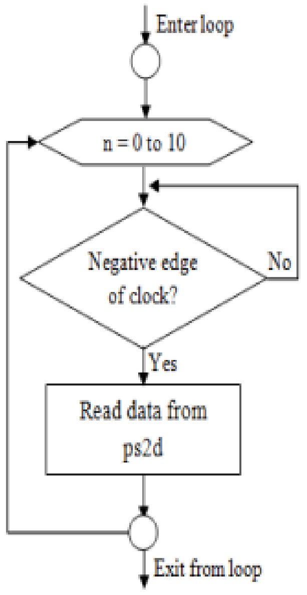


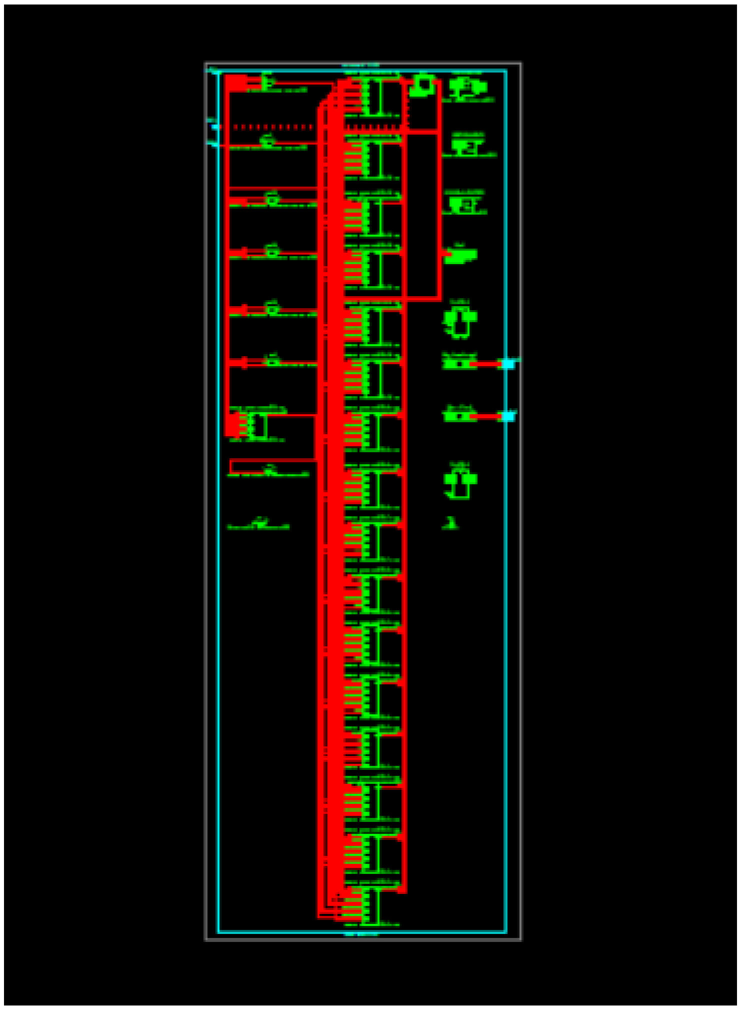
Figure: device to host flowchart

3 RESULT

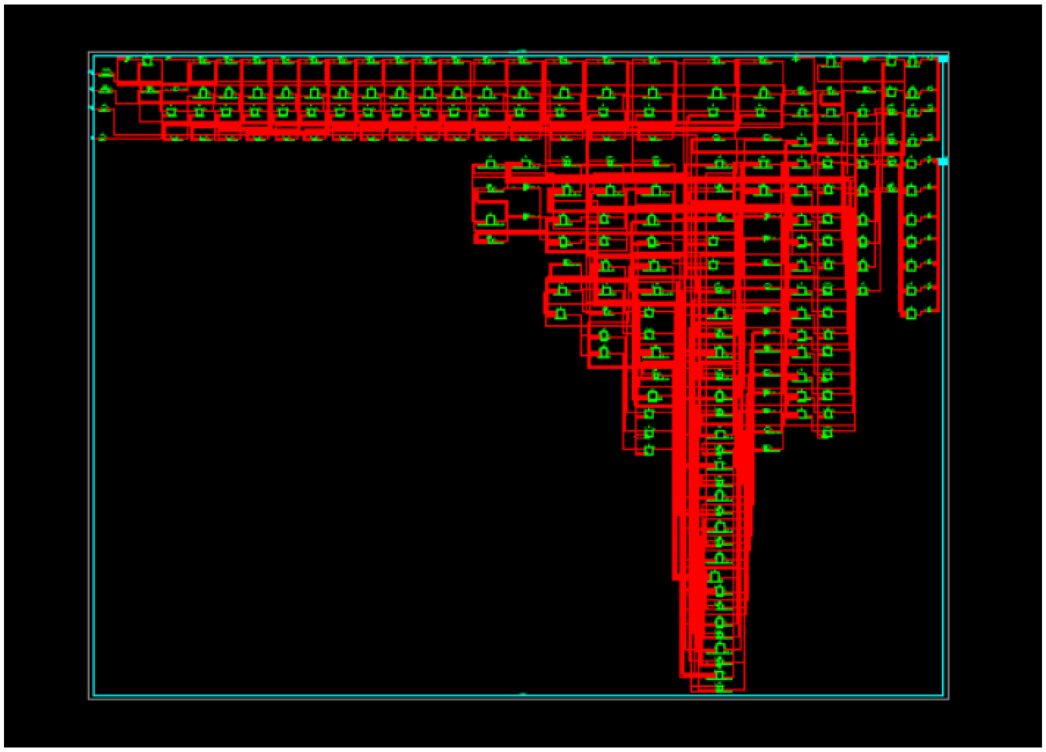
ISE will convert the Verilog description into a set of configuration bits that are used to program the Xilinx part to behave just like the Verilog code. Those configuration bits are in a '.bit' file. We need to set some constraints to put this code on the Xilinx FPGA on the Basys-3 board which is shown in the appendix. In particular, we need to tell ISE which pins on the Xilinx chip we want to assign so that, we can access those from switches and LEDs on the FPGA board. For that we need a "User Constraints File".

The Xilinx software will generate the schematic view of each block of the mouse interface. The schematic views of ps2_unit(mouse_unit) is shown in Figure and Internal schematic mouse interface is shown in Figure.

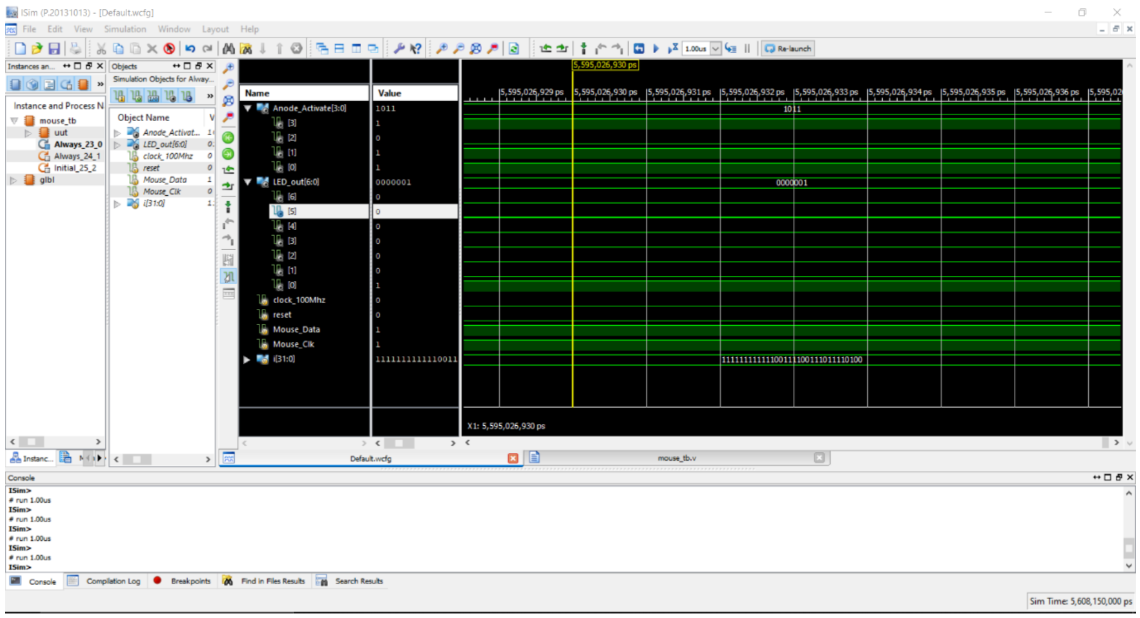
1.1 RTL



1.2 Technology Schematic



1.3 Simulated behavior model:



Name	Value
Anode_Activate[3:0]	1011
[3]	1
[2]	0
[1]	1
[0]	1
LED_out[6:0]	0000001
[6]	0
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	1
clock_100Mhz	0
reset	0
Mouse_Data	1
Mouse_Clk	1
i[31:0]	1111111111110011

```
X1: 5,595,026,930 ps
```

```
ISim>
# run 1.00us
ISim>
# run 1.00us
ISim>
# run 1.00us
ISim>
# run 1.00us
ISim>
```

```
Sim Time: 5,608,150,000 ps
```

4. 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.

References

Website:

Basys 3 Artix-7 FPGA Board:

https://www.xilinx.com/products/boards-and-kits/1-54wqge.html

USB-HID port:

https://en.wikipedia.org/wiki/USB_human_interface_device_class

Paper:

FPGA implementation of mouse interface: https://ieeexplore.ieee.org/document/8300958

Appendix

(A) CODE

```
module mouse_ FPGA(
    input clock_100Mhz,                         // 100 Mhz clock source on Basys 3 FPGA
    input reset,                                // reset
    input Mouse_Data,                           // Mouse PS2 data
    input Mouse_Clk,                            // Mouse PS2 Clock
    output reg [3:0] Anode_Activate,            // anode signals of the 7-segment LED display
    output reg [6:0] LED_out                    // cathode patterns of the 7-segment LED display
    );
    reg [5:0] Mouse_bits;                      // count number of bits receiving from the PS2
                                               mouse
    reg [26:0] one_second_counter;              // counter for generating 1 second clock enable
    wire one_second_enable;                    // one second enable for counting numbers
    reg [15:0] displayed_number;               // Number to be increased and decreased by the
                                               mouse
    reg [3:0] LED_BCD;
              // Signals for displaying on 7-segment LED of Basys 3 FPGA
             reg [20:0] refresh_counter; // the first 19-bit for creating 190Hz refresh rate
            // the other 2-bit for creating 4 LED-activating signals
    wire [1:0] LED_activating_counter;
 // counting the number of bits receiving from the Mouse Data
 // 33 bits to be received from the Mouse
    always @(posedge Mouse_Clk or posedge reset)
    begin
       if(reset==1)
          Mouse_bits <= 0;
       else if(Mouse_bits <=31)
          Mouse_bits <= Mouse_bits + 1;
       else
           Mouse_bits <= 0;
    end
 // Increase/Decrease the number when pressing Left/Right Mouse
    always @(negedge Mouse_Clk or posedge reset)
    begin
       if(reset)
          displayed_number <= 0;
       else begin
          if(Mouse_bits==1) begin
             if(Mouse_Data==1) // if The mouse is left clicked, increase the number
                displayed_number <= displayed_number + 1;
          end
          else if(Mouse_bits==2) begin
            if(Mouse_Data==1&&displayed_number>0)// if The mouse is right clicked, decrease the
number
                displayed_number <= displayed_number - 1;
             end
       end
    end
 // refreshing the 4-digit 7-segment display on Basys 3 FPGA
always @(posedge clock_100Mhz or posedge reset)
begin
    if(reset==1)
        refresh_counter <= 0;
    else
        refresh_counter <= refresh_counter + 1;
end
assign LED_activating_counter = refresh_counter[20:19];
// anode activating signals for 4 LEDs
// decoder to generate anode signals
always @(*)
begin
    case(LED_activating_counter)
    2'b00: begin
        Anode_Activate = 4'b0111;
        // activate LED1 and Deactivate LED2, LED3, LED4
        LED_BCD = displayed_number/1024;
        // the first digit of the 16-bit number
          end
    2'b01: begin
        Anode_Activate = 4'b1011;
        // activate LED2 and Deactivate LED1, LED3, LED4
        LED_BCD = (displayed_number % 1024)/64;
        // the second digit of the 16-bit number
          end
    2'b10: begin
        Anode_Activate = 4'b1101;
        // activate LED3 and Deactivate LED2, LED1, LED4
        LED_BCD = ((displayed_number % 1024)%64)/2;
        // the third digit of the 16-bit number
            end
    2'b11: begin
        Anode_Activate = 4'b1110;
        // activate LED4 and Deactivate LED2, LED3, LED1
        LED_BCD = ((displayed_number % 1024)%64)%2;
        // the fourth digit of the 16-bit number
           end
    endcase
end
// Cathode patterns of the 7-segment LED display
always @(*)
begin
    case(LED_BCD)
    4'b0000: LED_out = 7'b0000001; // "0"
    4'b0001: LED_out = 7'b1001111; // "1"
    4'b0010: LED_out = 7'b0010010; // "2"
    4'b0011: LED_out = 7'b0000110; // "3"
    4'b0100: LED_out = 7'b1001100; // "4"
    4'b0101: LED_out = 7'b0100100; // "5"
    4'b0110: LED_out = 7'b0100000; // "6"
    4'b0111: LED_out = 7'b0001111; // "7"
```

```
        4'b1000: LED_out = 7'b0000000; // "8"
        4'b1001: LED_out = 7'b0000100; // "9"
        default: LED_out = 7'b0000001; // "0"
        endcase
    end
endmodule
```