1 Introduction

A PIC was connected to a USB mouse. An existing driver was modified to transmit mouse information to an FPGA. Existing code on the FPGA was modified to receive this information and transmit it to VGA.

2 Design and Testing Methodology

The process of designing and testing the lab occurs as designated below.

2.1 Design

Each component of the lab was designed as designated below.

2.1.1 PIC

The existing driver for the mouse was modified such that it could now store the mouse's current position based on the previous state of the mouse along with the mouse movement information.

The modifications to the code were made primarily through the user of libraries to set up the SPI connection. The SPI2 channel was chosen primarily due to it not conflicting with any existing setup on the board.

Framed SPI mode was chosen since it was found that it was relatively easy to cause the SPI communications to be irreversibly desynchronized if either the PIC or the FPGA was disconnected and restarted independently of the other one.

Position calculation and bounds checking are performed on the PIC since it was easier to program than on the FPGA.

32 bits were transmitted at a time such that 10 bits for both the x coordinate and the y coordinate could be sent, along with 3 bits for the button information.

The fastest SPI transmission rate possible was chosen by using the smallest valid clock divisor.

Transmission occurred solely when the mouse sent input through the USB since there would be no need to update the state on the FPGA if there was no input from the mouse.

2.1.2 FPGA

The FPGA was configured to accept framed SPI input from the PIC. The module used to receive the SPI signal transmits a finished signal that indicates when the transmission is complete.

A background was rendered using various functions of x-position, y-position, and a clock based counter to create a moving pattern in the background.

The mouse was rendered using circles and ellipses out of personal choice, and the colors were set to change based on button input in order to make sure the buttons were also operating correctly.

The clocks used in the rendering used vsync in order to refresh based on the screen refresh rate to avoid tearing.

Otherwise, aside from the SPI receiver and graphics drawing, the given FPGA code was left as is.

2.2 Testing

Testing occurred primarily on a as-needed basis. Aside from debugging, as long as the mouse rendered on the screen and moved as expected, additional testing was not found necessary.

3 Technical Documentation

The designs used in the lab are illustrated in the following code and diagrams.

3.1 PIC

The following header illustrates the edited mouse driver.

```
/* Mouse_demo.c
Class-compliant USB mouse driver for PIC32
A trimmed-down version of the "USB Host - HID Mouse" demo program
from the Microchip Application Libraries, retreived June 2010 from
the Microchip website.

*/
/*  Matthew Watkins
Fall 2010
   Simplified 25 October 2011 David_Harris & Karl_Wang@hmc.edu
    Modified 31 October 2014 Henry_Huang@hmc.edu
*/

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

    USB Mouse Host Application Demo

Description:
    This file contains the basic USB Mouse application demo. Purpose of the demo
    is to demonstrate the capability of HID host . Any Low speed/Full Speed
    USB Mouse can be connected to the PICtail USB adapter along with
    Explorer 16 demo board. This file schedules the HID ransfers, and interprets
    the report received from the mouse. X & Y axis coordinates, Left & Right Click
    received from the mouse are diaplayed on the the LCD display mounted on the
    Explorer 16 board. Demo gives a fair idea of the HID host and user should be
    able to incorporate necessary changes for the required application.

* File Name:       Mouse_demo.c
* Dependencies:    None
* Processor:       PIC24FJ256GB110
* Compiler:        C30 v2.01
* Company:         Microchip Technology, Inc.

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated
(the Company) for its PICmicro Microcontroller is intended and
supplied to you, the Companys customer, for use solely and
```

```
exclusively on Microchip PICmicro Microcontroller products. The
software is owned by the Company and/or its supplier, and is
protected under applicable copyright laws. All rights are reserved.
Any use in violation of the foregoing restrictions may subject the
user to criminal sanctions under applicable laws, as well as to
civil liability for the breach of the terms and conditions of this
license.

THIS SOFTWARE IS PROVIDED IN AN AS IS CONDITION. NO WARRANTIES,
WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

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

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "usb_config.h"
#include "USB\usb.h"
#include "USB\usb_host_hid_parser.h"
#include "USB\usb_host_hid.h"
#include <plib.h>
#include <P32xxxx.h>


// *****************************************************************************
// *****************************************************************************
// Data Structures
// *****************************************************************************
// *****************************************************************************

typedef enum _APP_STATE
{
    DEVICE_NOT_CONNECTED,
    DEVICE_CONNECTED, /* Device Enumerated  - Report Descriptor Parsed */
    READY_TO_TX_RX_REPORT,
    GET_INPUT_REPORT, /* perform operation on received report */
    INPUT_REPORT_PENDING,
    ERROR_REPORTED
} APP_STATE;

typedef struct _HID_REPORT_BUFFER
{
    WORD  Report_ID;
    WORD  ReportSize;
//    BYTE* ReportData;
    BYTE  ReportData[4];
    WORD  ReportPollRate;
}   HID_REPORT_BUFFER;
```

```
// ****************************************************************************
// ****************************************************************************
// Internal Function Prototypes
// ****************************************************************************
// ****************************************************************************
BYTE App_DATA2ASCII(BYTE a);
void AppInitialize(void);
BOOL AppGetParsedReportDetails(void);
void App_Detect_Device(void);
void App_ProcessInputReport(void);
BOOL USB_HID_DataCollectionHandler(void);


// ****************************************************************************
// ****************************************************************************
// Macros
// ****************************************************************************
// ****************************************************************************
#define MAX_ALLOWED_CURRENT             (500)         // Maximum power we can supply in mA
#define MINIMUM_POLL_INTERVAL           (0x0A)        // Minimum Polling rate for HID reports is 10m

#define USAGE_PAGE_BUTTONS              (0x09)

#define USAGE_PAGE_GEN_DESKTOP          (0x01)


#define MAX_ERROR_COUNTER               (10)


// ****************************************************************************
// ****************************************************************************
// Global Variables
// ****************************************************************************
// ****************************************************************************

APP_STATE App_State_Mouse = DEVICE_NOT_CONNECTED;

HID_DATA_DETAILS Appl_Mouse_Buttons_Details;
HID_DATA_DETAILS Appl_XY_Axis_Details;

HID_REPORT_BUFFER  Appl_raw_report_buffer;

HID_USER_DATA_SIZE Appl_Button_report_buffer[3];
HID_USER_DATA_SIZE Appl_XY_report_buffer[3];

BYTE ErrorDriver;
BYTE ErrorCounter;
BYTE NumOfBytesRcvd;

BOOL ReportBufferUpdated;
BOOL LED_Key_Pressed = FALSE;

BYTE currCharPos;
BYTE FirstKeyPressed ;

short mouseMvt = 0;

//****************************************************************************
//****************************************************************************
// Main
//****************************************************************************
//****************************************************************************


int main (void)
{
     BYTE i;
        int  value;

        value = SYSTEMConfigWaitStatesAndPB( GetSystemClock() );

        // Enable the cache for the best performance
        CheKseg0CacheOn();

        INTEnableSystemMultiVectoredInt();

        TRISF = 0xFFFF;
   TRISE = 0xFFFF;
   TRISB = 0xFFFF;
   TRISG = 0xFFFF;

        // Initialize USB layers
        USBInitialize( 0 );
        while(1)
        {
            USBTasks();
            App_Detect_Device();

            switch(App_State_Mouse)
            {
                case DEVICE_NOT_CONNECTED:
                             USBTasks();

                             if(USBHostHID_ApiDeviceDetect()) /* True if report descriptor is parsed
                             {
                                App_State_Mouse = DEVICE_CONNECTED;
                             }
                    break;
                case DEVICE_CONNECTED:
                           App_State_Mouse = READY_TO_TX_RX_REPORT;
                    break;
                case READY_TO_TX_RX_REPORT:
                             if(!USBHostHID_ApiDeviceDetect())
                             {
                                App_State_Mouse = DEVICE_NOT_CONNECTED;
                             }
                             else
                             {
                                App_State_Mouse = GET_INPUT_REPORT;
                             }

                    break;
                case GET_INPUT_REPORT:
                            if(USBHostHID_ApiGetReport(Appl_raw_report_buffer.Report_ID,0,
                                                         Appl_raw_report_buffer.ReportSize, Appl_raw_
                            {
                               /* Host may be busy/error -- keep trying */
                            }
                            else
                            {
                                App_State_Mouse = INPUT_REPORT_PENDING;
                            }
                            USBTasks();
                    break;
                case INPUT_REPORT_PENDING:
                           if(USBHostHID_ApiTransferIsComplete(&ErrorDriver,&NumOfBytesRcvd))
                            {
                                if(ErrorDriver ||(NumOfBytesRcvd != Appl_raw_report_buffer.ReportSiz
                                {
                                  ErrorCounter++ ;
                                  if(MAX_ERROR_COUNTER <= ErrorDriver)
                                     App_State_Mouse = ERROR_REPORTED;
                                  else
                                     App_State_Mouse = READY_TO_TX_RX_REPORT;
                                }
                                else
                                {
                                  ErrorCounter = 0;
                                  ReportBufferUpdated = TRUE;
                                  App_State_Mouse = READY_TO_TX_RX_REPORT;

                                  App_ProcessInputReport();
                                }
                            }
                    break;

               case ERROR_REPORTED:
                    break;
                default:
                    break;

            }
        }
}


void App_ProcessInputReport(void)
{
    BYTE  data;
short xMvmt, yMvmt;
   /* process input report received from device */
```

```
    USBHostHID_ApiImportData(Appl_raw_report_buffer.ReportData, Appl_raw_report_buffer.ReportSize
                             ,Appl_Button_report_buffer, &Appl_Mouse_Buttons_Details);
    USBHostHID_ApiImportData(Appl_raw_report_buffer.ReportData, Appl_raw_report_buffer.ReportSize
                             ,Appl_XY_report_buffer, &Appl_XY_Axis_Details);


    xMvmt = (signed char) Appl_XY_report_buffer[0]; // Get X-axis movement from report
    TRISD = 0;
PORTD = Appl_Button_report_buffer[0] | (Appl_Button_report_buffer[1] << 1) | (Appl_Button_report_buf
// Write to LEDs to show mouse is working
    yMvmt = (signed char) Appl_XY_report_buffer[1]; // Get Y-axis movement from report

}

//****************************************************************************
//****************************************************************************
// USB Support Functions
//****************************************************************************
//****************************************************************************

BOOL USB_ApplicationEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
    switch( event )
    {
        case EVENT_VBUS_REQUEST_POWER:
            // The data pointer points to a byte that represents the amount of power
            // requested in mA, divided by two.  If the device wants too much power,
            // we reject it.
            if (((USB_VBUS_POWER_EVENT_DATA*)data)->current <= (MAX_ALLOWED_CURRENT / 2))
            {
                return TRUE;
            }
            else
            {
              //  UART2PrintString( "\r\n***** USB Error - device requires too much current *****\r\
            }
            break;

        case EVENT_VBUS_RELEASE_POWER:
            // Turn off Vbus power.
            // The PIC24F with the Explorer 16 cannot turn off Vbus through software.
            return TRUE;
            break;

        case EVENT_HUB_ATTACH:
        //    UART2PrintString( "\r\n***** USB Error - hubs are not supported *****\r\n" );
            return TRUE;
            break;

        case EVENT_UNSUPPORTED_DEVICE:
        //    UART2PrintString( "\r\n***** USB Error - device is not supported *****\r\n" );
            return TRUE;
            break;
        case EVENT_CANNOT_ENUMERATE:
        //   UART2PrintString( "\r\n***** USB Error - cannot enumerate device *****\r\n" );
            return TRUE;
            break;

        case EVENT_CLIENT_INIT_ERROR:
        //    UART2PrintString( "\r\n***** USB Error - client driver initialization error *****\r\n"
            return TRUE;
            break;

        case EVENT_OUT_OF_MEMORY:
        //    UART2PrintString( "\r\n***** USB Error - out of heap memory *****\r\n" );
            return TRUE;
            break;

        case EVENT_UNSPECIFIED_ERROR:   // This should never be generated.
        //    UART2PrintString( "\r\n***** USB Error - unspecified *****\r\n" );
            return TRUE;
            break;

case EVENT_HID_RPT_DESC_PARSED:
 #ifdef APPL_COLLECT_PARSED_DATA
     return(APPL_COLLECT_PARSED_DATA());
     #else
 return TRUE;
 #endif
break;

        default:
            break;
    }
    return FALSE;
}


/****************************************************************************
  Function:
    BYTE App_HID2ASCII(BYTE a)
  Description:
    This function converts the HID code of the key pressed to coressponding
    ASCII value. For Key strokes like Esc, Enter, Tab etc it returns 0.
***************************************************************************/
BYTE App_DATA2ASCII(BYTE a) //convert USB HID code (buffer[2 to 7]) to ASCII code
{
   if(a<=0x9)
    {
      return(a+0x30);
    }

   if(a>=0xA && a<=0xF)
    {
      return(a+0x37);
    }
   return(0);
}


/*****************************************************************************
  Function:
    void App_Detect_Device(void)
  Description:
    This function monitors the status of device connected/disconnected
    None
*****************************************************************************/
void App_Detect_Device(void)
{
  if(!USBHostHID_ApiDeviceDetect())
  {
     App_State_Mouse = DEVICE_NOT_CONNECTED;
  }
}


/*****************************************************************************
  Function:
    BOOL USB_HID_DataCollectionHandler(void)
  Description:
    This function is invoked by HID client , purpose is to collect the
    details extracted from the report descriptor. HID client will store
    information extracted from the report descriptor in data structures.
    Application needs to create object for each report type it needs to
    extract.
    For ex: HID_DATA_DETAILS Appl_ModifierKeysDetails;
    HID_DATA_DETAILS is defined in file usb_host_hid_appl_interface.h
    Each member of the structure must be initialized inside this function.
    Application interface layer provides functions :
    USBHostHID_ApiFindBit()
    USBHostHID_ApiFindValue()
    These functions can be used to fill in the details as shown in the demo
    code.

  Return Values:
    TRUE    - If the report details are collected successfully.
    FALSE   - If the application does not find the the supported format.

  Remarks:
    This Function name should be entered in the USB configuration tool
    in the field "Parsed Data Collection handler".
    If the application does not define this function , then HID cient
    assumes that Application is aware of report format of the attached
    device.
*****************************************************************************/
BOOL USB_HID_DataCollectionHandler(void)
{
  BYTE NumOfReportItem = 0;
  BYTE i;
  USB_HID_ITEM_LIST* pitemListPtrs;
```

```
USB_HID_DEVICE_RPT_INFO* pDeviceRptinfo;
HID_REPORTITEM *reportItem;
HID_USAGEITEM *hidUsageItem;
BYTE usageIndex;
BYTE reportIndex;

pDeviceRptinfo = USBHostHID_GetCurrentReportInfo(); // Get current Report Info pointer
pitemListPtrs = USBHostHID_GetItemListPointers();   // Get pointer to list of item pointers

BOOL status = FALSE;
 /* Find Report Item Index for Modifier Keys */
 /* Once report Item is located , extract information from data structures provided by the parser
 NumOfReportItem = pDeviceRptinfo->reportItems;
 for(i=0;i<NumOfReportItem;i++)
  {
     reportItem = &pitemListPtrs->reportItemList[i];
     if((reportItem->reportType==hidReportInput) && (reportItem->dataModes == (HIDData_Variable|HI
         (reportItem->globals.usagePage==USAGE_PAGE_GEN_DESKTOP))
      {
         /* We now know report item points to modifier keys */
         /* Now make sure usage Min & Max are as per application */
          usageIndex = reportItem->firstUsageItem;
          hidUsageItem = &pitemListPtrs->usageItemList[usageIndex];

          reportIndex = reportItem->globals.reportIndex;
          Appl_XY_Axis_Details.reportLength = (pitemListPtrs->reportList[reportIndex].inputBits +
          Appl_XY_Axis_Details.reportID = (BYTE)reportItem->globals.reportID;
          Appl_XY_Axis_Details.bitOffset = (BYTE)reportItem->startBit;
          Appl_XY_Axis_Details.bitLength = (BYTE)reportItem->globals.reportsize;
          Appl_XY_Axis_Details.count=(BYTE)reportItem->globals.reportCount;
          Appl_XY_Axis_Details.interfaceNum= USBHostHID_ApiGetCurrentInterfaceNum();
      }
      else if((reportItem->reportType==hidReportInput) && (reportItem->dataModes == HIDData_Variab
         (reportItem->globals.usagePage==USAGE_PAGE_BUTTONS))
      {
         /* We now know report item points to modifier keys */
         /* Now make sure usage Min & Max are as per application */
          usageIndex = reportItem->firstUsageItem;
          hidUsageItem = &pitemListPtrs->usageItemList[usageIndex];

          reportIndex = reportItem->globals.reportIndex;
          Appl_Mouse_Buttons_Details.reportLength = (pitemListPtrs->reportList[reportIndex].inputB
          Appl_Mouse_Buttons_Details.reportID = (BYTE)reportItem->globals.reportID;
          Appl_Mouse_Buttons_Details.bitOffset = (BYTE)reportItem->startBit;
          Appl_Mouse_Buttons_Details.bitLength = (BYTE)reportItem->globals.reportsize;
          Appl_Mouse_Buttons_Details.count=(BYTE)reportItem->globals.reportCount;
          Appl_Mouse_Buttons_Details.interfaceNum= USBHostHID_ApiGetCurrentInterfaceNum();
      }
  }

 if(pDeviceRptinfo->reports == 1)
  {
      Appl_raw_report_buffer.Report_ID = 0;
      Appl_raw_report_buffer.ReportSize = (pitemListPtrs->reportList[reportIndex].inputBits + 7)/8
//        Appl_raw_report_buffer.ReportData = (BYTE*)malloc(Appl_raw_report_buffer.ReportSize);
        Appl_raw_report_buffer.ReportPollRate = pDeviceRptinfo->reportPollingRate;
        status = TRUE;
    }

    return(status);
}
```

3.2 FPGA

The following header illustrates the edited VGA driver.

```
// vga.sv
// 20 October 2011 Karl_Wang & David_Harris@hmc.edu
// Modified 31 October 2014 Henry_Huang@hmc.edu
// VGA driver with character generator

module vga(input  logic       clk,
           input  logic       spi_clk,
           input  logic       spi_in,
           input  logic       spi_fsync,
           output logic       vgaclk,// 25 MHz VGA clock
           output logic       hsync, vsync, sync_b,// to monitor & DAC
           output logic [7:0] r, g, b); // to video DAC

  logic [9:0]  x, y;
  logic [7:0]  r_int, g_int, b_int;
  logic [31:0] spi_word;
  logic [9:0]  x_cursor;
  logic [9:0]  y_cursor;
  logic        finished;
  logic [2:0]  buttons;

  // Use a PLL to create the 25.175 MHz VGA pixel clock
  // 25.175 Mhz clk period = 39.772 ns
  // Screen is 800 clocks wide by 525 tall, but only 640 x 480 used for display
  // HSync = 1/(39.772 ns * 800) = 31.470 KHz
  // Vsync = 31.474 KHz / 525 = 59.94 Hz (~60 Hz refresh rate)
  pll vgapll(.inclk0(clk),.c0(vgaclk));

  // generate monitor timing signals
  vgaController vgaCont(vgaclk, hsync, vsync, sync_b,
                        r_int, g_int, b_int, r, g, b, x, y);



  spi_frm_slave slave(spi_clk, spi_in, spi_fsync, spi_word, finished);

  mouse_reader reader(vsync, spi_word, finished, x_cursor, y_cursor, buttons);

  // user-defined module to determine pixel color
  videoGen videoGen(vsync, x, y, x_cursor, y_cursor,
                    buttons, r_int, g_int, b_int);
endmodule

module vgaController #(parameter HMAX   = 10'd800,
                                 VMAX   = 10'd525,
HSTART = 10'd152,
WIDTH  = 10'd640,
VSTART = 10'd37,
HEIGHT = 10'd480)
  (input  logic       vgaclk,
               output logic       hsync, vsync, sync_b,
 input  logic [7:0] r_int, g_int, b_int,
 output logic [7:0] r, g, b,
 output logic [9:0] x, y);

  logic [9:0] hcnt, vcnt;
  logic       oldhsync;
  logic       valid;

  // counters for horizontal and vertical positions
  always @(posedge vgaclk) begin
    if (hcnt >= HMAX) hcnt = 0;
    else hcnt++;
 if (hsync & ~oldhsync) begin // start of hsync; advance to next row
   if (vcnt >= VMAX) vcnt = 0;
      else vcnt++;
    end
    oldhsync = hsync;
  end

  // compute sync signals (active low)
  assign hsync = ~(hcnt >= 10'd8 & hcnt < 10'd104); // horizontal sync
  assign vsync = ~(vcnt >= 2 & vcnt < 4); // vertical sync
  assign sync_b = hsync | vsync;

  // determine x and y positions
  assign x = hcnt - HSTART;
  assign y = vcnt - VSTART;

  // force outputs to black when outside the legal display area
  assign valid = (hcnt >= HSTART & hcnt < HSTART+WIDTH &
                  vcnt >= VSTART & vcnt < VSTART+HEIGHT);
  assign {r,g,b} = valid ? {r_int,g_int,b_int} : 24'b0;
endmodule

module videoGen(input  logic       clk,
                input  logic [9:0] x, y, x_cursor, y_cursor,
                input  logic [2:0] buttons,
              output logic [7:0] r_int, g_int, b_int);

  logic [7:0] ch;
  logic [23:0] background_rgb;

  background(clk, x, y, background_rgb);
  draw_cursor(x, y, x_cursor, y_cursor, buttons,
              background_rgb, {r_int, g_int, b_int});
endmodule

// draw a cursor centered at {x_cursor, y_cursor} that changes
// color based on the buttons pressed
module draw_cursor(input  logic [9:0]  x, y, x_cursor, y_cursor,
                   input  logic [2:0]  buttons,
                   input  logic [23:0] background_rgb,
                   output logic [23:0] rgb);
  logic [23:0] cursor_color;
  logic        in_cursor;
  logic        in_hitbox;
  in_ellipse cursor(x, y, x_cursor, y_cursor, 140,1,0, in_cursor);
  in_disk hitbox(x, y, x_cursor, y_cursor, 8, in_hitbox);


  assign cursor_color = {buttons[2]? 8'h80 : 8'hFF,
                         buttons[1]? 8'h80 : 8'hFF,
                         buttons[0]? 8'h80 : 8'hFF};

  assign rgb = in_hitbox ? 24'hFFFFFF :
               in_cursor ? {8'((9'(cursor_color[23:16])
                                +background_rgb[23:16])>>1),
                            8'((9'(cursor_color[15: 8])
                                +background_rgb[15: 8])>>1),
                            8'((9'(cursor_color[ 7: 0])
                                +background_rgb[ 7: 0])>>1)}:
               background_rgb;

endmodule

// Calculate if a point lies in an disk centered at
// {cent_x, cent_y} with radius sqrt(rad_squared).
module in_disk  (input  logic [9:0]  x, y, cent_x, cent_y,
                 input  logic [21:0] rad_squared,
                 output logic        is_in);
  logic signed [10:0] d_x, d_y;
  logic [21:0] dist_squared;
  always_comb begin
    d_x = (x-cent_x);
    d_y = (y-cent_y);
    dist_squared = d_x*d_x + d_y*d_y;
    is_in = (dist_squared < rad_squared);
  end
endmodule

// Calculate if a point lies in an ellipse centered at
// {cent_x, cent_y} with x_radius (sqrt(rad_squared) >> x_reduce)
// and y_radius (sqrt(rad_squared) >> y_reduce).
module in_ellipse  (input  logic [9:0]  x, y, cent_x, cent_y,
                    input  logic [31:0] rad_squared,
                    input  logic [1:0]  x_reduce, y_reduce,
                    output logic        is_in);
  logic signed [15:0] d_x, d_y;
  logic [31:0] dist_squared;
  always_comb begin
    d_x = (x-cent_x) << x_reduce;
    d_y = (y-cent_y) << y_reduce;
    dist_squared = d_x*d_x + d_y*d_y;
    is_in = (dist_squared < rad_squared);
  end
endmodule

// creates an interesting gradient as a background based on
// the timing and position.
module background(input  logic       clk,
                  input  logic [9:0] x_screen, y_screen,
                  output logic [23:0] rgb);
logic [9:0] diff, sum, x, y;
logic [19:0] product;
logic [7:0] r, g, b, x_wave, y_wave, intensity;
logic [10:0] counter;

always_ff @(posedge clk) begin
  counter <= counter + 1;
end

always_comb begin
  x = x_screen;
  y = y_screen;

  // diagonal gradient functions that move with time
  diff = 10'(x - y + (counter >> 0));
  sum  = 10'(x+y - (counter >> 0));
  product = diff*sum;

  // the moving curved gradients
  intensity = 8'(product[11:4] + (counter << 2)) >> 3;

  // the shifting "blinds"
  x_wave = (8'((x<<2)-counter)>>>5) + (8'(-(x<<2)-counter)>>>5);
  y_wave = (8'((y<<2)-counter)>>>6) + (8'(-(y<<2)-counter)>>>6);

  // have each color play separate roles in each pattern
  r = 8'h70 + x_wave + y_wave;
  g = 8'h80 + ~intensity;
  b = 8'hC0 + intensity + x_wave + y_wave;
  rgb = {r,g,b};
end

endmodule

// reads input from a frame enabled SPI to word_input
// and signals the end of the transmission with finished
module spi_frm_slave #(parameter WIDTH_POWER = 5, WIDTH = 2**WIDTH_POWER)
                      (input  logic               spi_clk,
                       input  logic               serial_input,
                        input  logic                fsync,
                        output logic [(WIDTH-1):0] word_input,
                        output logic                finished);
  logic [(WIDTH_POWER-1):0] counter = 0;

  always_ff @ (posedge spi_clk) begin
    word_input <= finished ? word_input :
                            {word_input[(WIDTH-2):0], serial_input};
    counter <= finished ? 1 : counter + 1;
    finished <= ~fsync ? '0 : finished ? '1 : counter == 0;
  end

endmodule

// converts the values from word_input to recreate the
// mouse state in the form of position and buttons.
module mouse_reader  (input  logic        sync_clk,
                      input  logic [31:0] spi_word,
                      input  logic        finished,
                      output logic [9:0]  x_pixel, y_pixel,
                      output logic [2:0]  button_state);
  logic [22:0] complete_word;
  always_ff @ (posedge finished) begin
    complete_word <= spi_word[22:0];
  end
  always_ff @ (posedge sync_clk) begin
    {button_state, x_pixel, y_pixel} = complete_word;
  end
endmodule
```

4 Results and Discussion

All the tasks were done as requested.

The cursor moves as desired on the screen when the mouse is moved, and stays within bounds.

The monitor displays a background along with a mouse and changes the mouse color based on button presses.

5 Conclusion

In this lab, a PIC was programmed to read input from a USB mouse, and pass on relevant mouse position to the FPGA, which displayed it on the screen.

This lab took approximately 6 hours to complete with an additional 2 hours for the commenting and writeup.