

## Overview

The Mouse Reference Component is a mouse controller with a PS/2 interface. There are many uses for this component, but the most common is as a mouse on a VGA screen. Most of the initializations and approaches described here involve this common use.

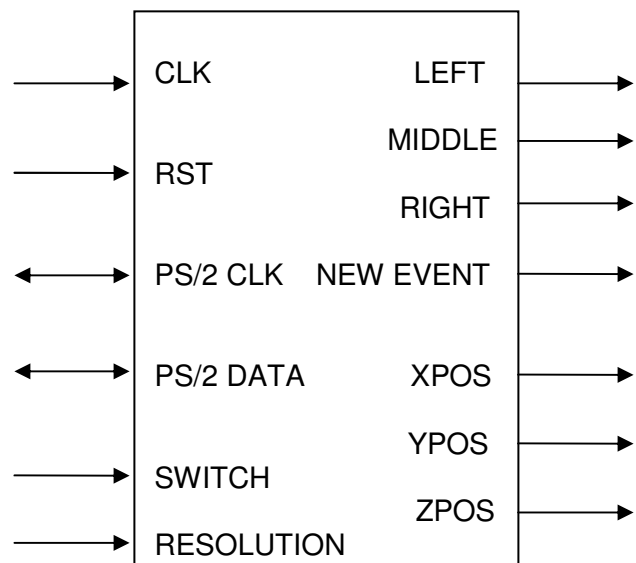
The Mouse Reference Component communicates through the PS/2 interface and is initialized at power-up or reset. Once the scrolling mode is enabled (if the mouse has a wheel), the resolution and sample rate is set, and data reporting is enabled (wait for data packets to arrive from the mouse). Then data movement packets are decoded to generate output signals representing X and Y displacements, button states, and mouse wheel rotation (Z, where available).

## Functional Description

As shown in Figure 2, the Mouse Reference Component is composed of three main modules: the PS/2 interface module, which is responsible for communication between the mouse and the controller; the mouse controller, which receives the data packages from the mouse and outputs the mouse position and button states; and the resolution mouse informer, which provides the active screen boundaries on the X and Y axes. This third component receives the resolution information and calculates the maximum values on the X and Y axes and sets the initial position of the mouse.

## Port Definitions

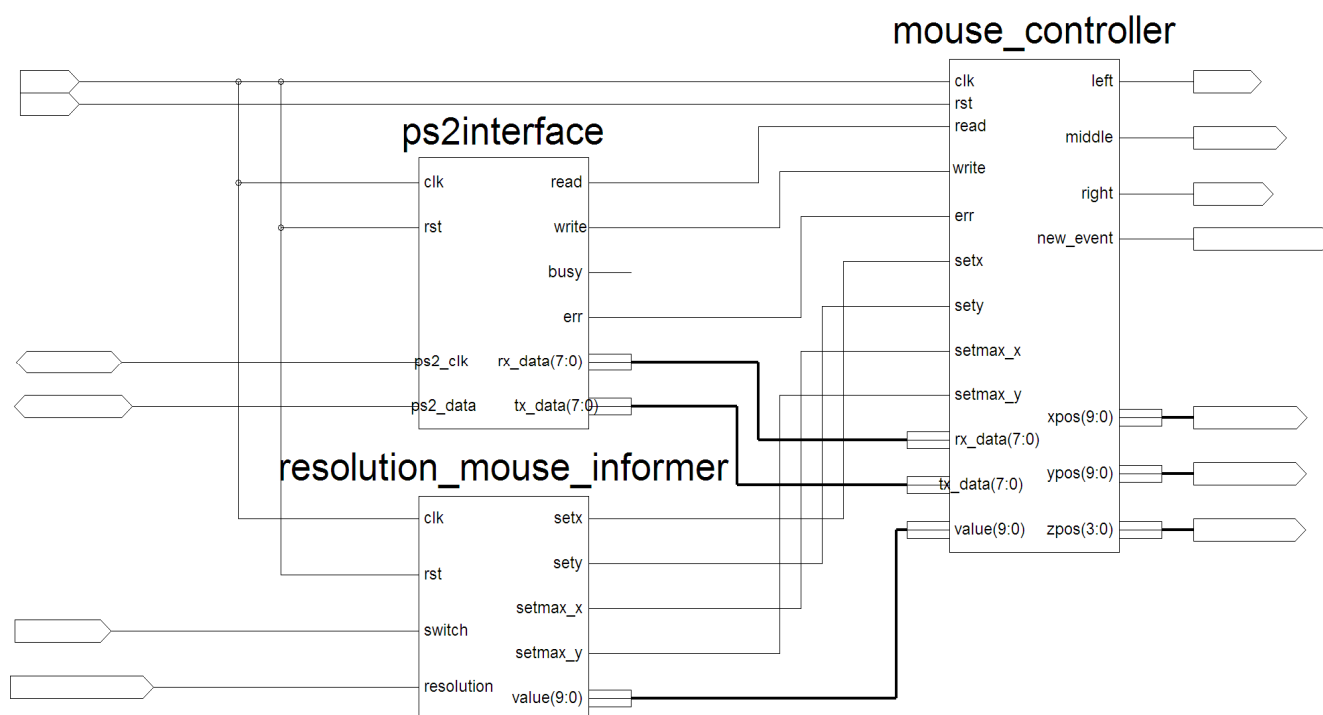
PS/2_clk	input, clock line of the PS/2 interface
PS/2_data	input, clock line of the PS/2 interface
clk	input, system clock signal
rst	input, system reset signal
resolution	input, sets one of two resolution options: 0 for 640x480 or 1 for 800x600
switch	input, active for one clk period when resolution changes.
left	output, state of the left mouse button ('1' if pressed, or '0')
middle	output, state of the middle mouse button ('1' if pressed, or '0')
right	output, state of the right mouse button ('1' if pressed, or '0')
xpos	output, nine bits, the current horizontal position of the mouse
ypos	output, nine bits, the current vertical position of the mouse
zpos	output, four bits, the last Z delta movement received
new_event	output, high for one clk period after a data packet is received and processed



**Figure 1 Mouse Reference Component**

## The PS/2 Interface

The PS/2 interface module implements a generic bi-directional PS/2 interface. It can be used with any PS/2 compatible device and it offers its clients a convenient way to exchange data. The interface transparently wraps the byte to be sent into a PS/2 frame, generates the parity bit, and sends the frame one bit at a time to the device. Similarly, when receiving data from the PS2 device, the interface receives the frame, checks for parity, extracts the useful data, and forwards it to the client. If an error occurs when sending or receiving a byte, the client is informed by setting the err output line high. This way the client can resend the data or issue a resend command to the device.



**Figure 2 Mouse Reference Component Internal Structure**

The physical PS/2 interface uses four lines. The PS/2 6-pin connector pins are assigned as follows<sup>1</sup>:

- |   |                 |
|---|-----------------|
| 1 | Data            |
| 2 | Not Implemented |
| 3 | Ground          |
| 4 | VCC (+5V)       |
| 5 | Clock           |
| 6 | Not Implemented |

The clock line carries the device clock with the frequency range 10 to 16.7KHz (30 to 50μs). When the line is idle it is placed in high impedance. The clock is only generated when the device is sending or receiving data. Data and clock lines are both open-collectors with pull-up resistors to VCC. An open-collector interface has two possible states: low ('0') or high ('Z') impedance.

When a device wants to send a byte, it pulls the clock line low and the host (i.e., the PS/2 interface) recognizes that the device is sending data. When the host wants to send data, it makes a send request. This is done by holding the clock line low for at least 100μs, then the data line is brought low. Next, the clock line is released (placed in high impedance). The device begins generating clock pulses. When receiving data, bits are read from the data line (PS/2\_data) on the falling edge of the

clock (PS/2\_clk). When sending data, the device reads the bits from the data line on the rising edge of the clock. A frame for sending a byte is comprised of an 11-bit value as shown in Figure 3. The frame is sent one bit at a time from the least significant bit, or starting bit, and is received the same way.

10	9	8	7	6	5	4	3	2	1	0
STOP	PAR	D7	D6	D5	D4	D3	D2	D1	D0	START

**Figure 3 PS/2 Frame for Sending a Byte**

STOP	stop bit, always '1'
PAR	parity bit, odd parity; the number of bits of '1' in the data bits together with parity bit is odd
D0-7	data bits
START	start bit, always '0'

When receiving, the frame register is shifted to the right, placing the first bit on the data line at the most significant position. This way the first bit sent will reach the least significant bit of the frame once all the bits have been received.

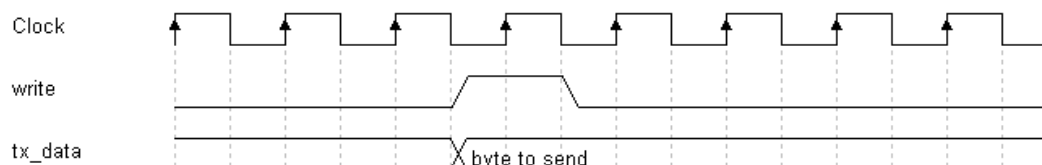
When sending data, the least significant bit of the frame is placed on the data line and the frame is shifted to the right. When releasing the clock line during the send request, the device reads the data line and interprets the data as the first bit of the frame. As a result, when sending, only ten shifts of the frame will be made. For the parity bit, a look-up table is defined in a 256x1 ROM.

While the interface is sending or receiving data, the busy output signal goes high. When the interface is idle, the output signal is low. After sending all the bits in the frame, the device must acknowledge the sent data. The host releases the data line (the clock line is already released) after the last bit is sent. The device brings the data line and the clock line low, in that order, to acknowledge the data. If the device does not acknowledge the data, err output is set.

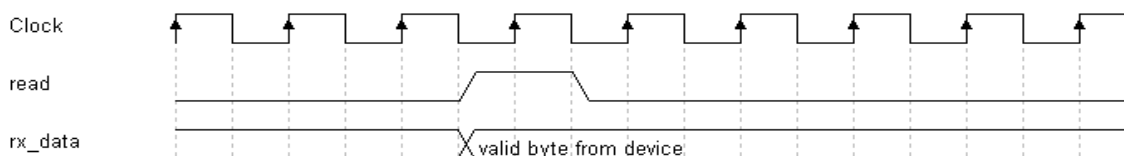
To generate the necessary delay, counters are used. For example, to generate the 100µs delay, a 14-bit counter that counts to 10,000 is used. Because the interface is designed to run at 100MHz, therefore  $10,000 \times 10\text{ns} = 100\mu\text{s}$ .

### Port Definitions for the PS/2 Interface Module

PS/2_clk	in/out pin, clock line of the PS/2 interface
PS/2_data	in/out pin, clock line of the PS/2 interface
clk	input pin, system clock signal
rst	input pin, system reset signal
tx_data	input pin, eight bits, from client data to be sent to the device
write	input pin, from client, should be active for one clock period when then client wants to send data to the device and data to be sent is valid on tx_data
rx_data	output pin, eight bits, to client data received from device
read	output pin, to client for one clock period when new data is available from device
busy	output pin, to client active while sending or receiving data
err	output pin, to client active for one clock period when an error occurred during sending or receiving



**Figure 4 Send Data to Device Timing Diagram**



**Figure 5 Receive Data from Device Timing Diagram**

The PS/2 interface receives bytes from the PS/2 device (mouse or keyboard) and sends them further to the mouse or keyboard controller. Data is sent on the rx\_data output port. The read signal is active for one clock period when a new byte is available on rx\_data. When the err input goes high for one clock (indicating an error while the PS/2 interface received a byte), the controller enters the reset state.

The PS/2 interface also gets bytes from the mouse or keyboard controller and sends them further to the device. Data is received on the tx\_data output port. The write output signal should be active for one clock period when tx\_data contains the command or data to be sent to the mouse or keyboard.

## The Mouse Controller

When in reset state, the controller resets the mouse and begins an initialization procedure. It performs the BAT (Basic Assurance Test) and sends the result of the test to the host, followed by the ID of the mouse, 00h. Next it enters stream mode, but with data reporting disabled. This means it will not send data movement packets when the mouse is moved or the state of the button changes. Also, it doesn't have the scrolling mode enabled.

The controller first resets the mouse, waits for the BAT result and the mouse ID, and then starts the enable scrolling mode procedure. The procedure consists of successively setting the sample rate to 200, then 100, and finally 80. After this is done, the controller requests the mouse ID by sending the F2h byte to the mouse (request ID command). If the received ID is 00h then the mouse does not have a wheel, or if the ID is 03h, then the mouse has entered scrolling mode and will include Z movement information in its data movement packets. Note that data reporting is still disabled at this point.

Next, the resolution is set to eight counts/mm and the sample rate is set to 40 samples per second. The last command sent to the mouse is the enable reporting command (F4h). After this, the mouse sends data movement packets every time the mouse is moved, its button states change, or the mouse wheel is moved (when available). The controller now enters another state, waiting for bytes from the mouse. The mouse sends the acknowledge byte (FAh) after each byte it receives from the host.

When the mouse wheel is disabled, or the mouse does not have a wheel, data movement packets consist of three consecutively sent bytes. When the scrolling mode is enabled, it sends four bytes.

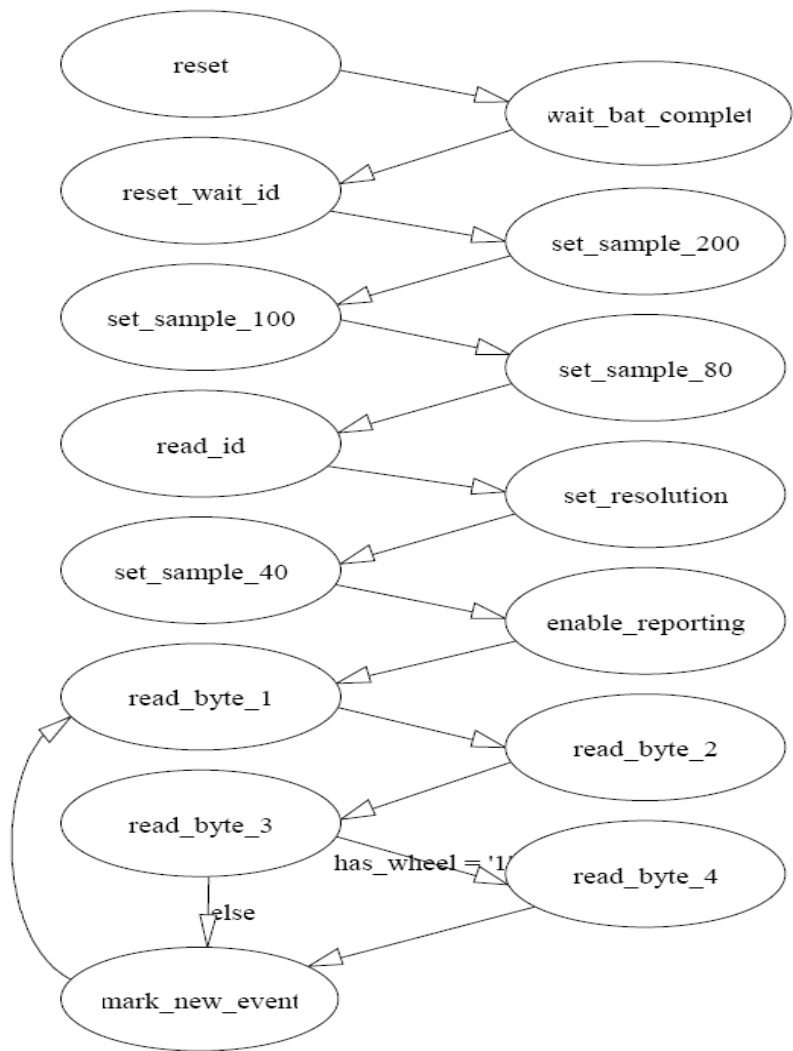


Figure 6 Mouse Controller FSM State Diagram

	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 7 Data Movement Packet Format (Scrolling Mode Enabled)

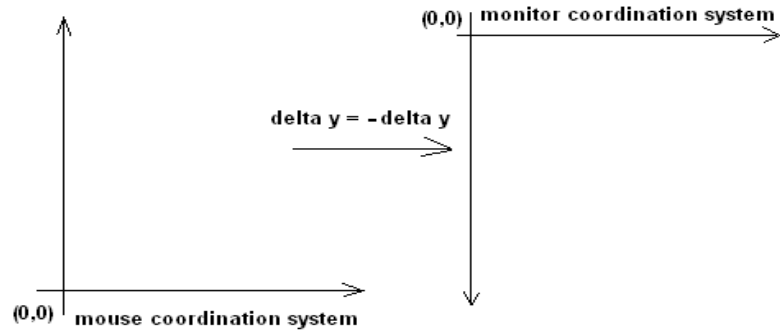
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							
Byte 4	Z Movement							

Figure 8 Data Movement Packet Format (Scrolling Mode Disabled)

The first received byte contains button states on the least significant three bits. If one of these bits is high, then the corresponding mouse button is pressed. The X and Y sign bit is the most significant bit of the X and Y movement value. The X and Y movements are represented using nine-bit two's-complement encoding. Thus, the range of movement is -256 to 255. The delta movement from the

mouse's last position is sent for either axis. The mouse does not keep track of the absolute position of the mouse. The controller uses the delta movements received to keep track of the absolute position.

The mouse uses the lower-left corner for axes origin, meaning the mouse reports a positive displacement on X when moved to the right and negative (X sign bit is '1') when moved to the left. When the mouse is moved upward, the Y displacement is positive, and when moved downward it is negative. The monitor uses the upper-left corner for the axes origin. The controller negates the Y displacement before adding it to the previous Y position to report the position relative to the monitor axes origin.



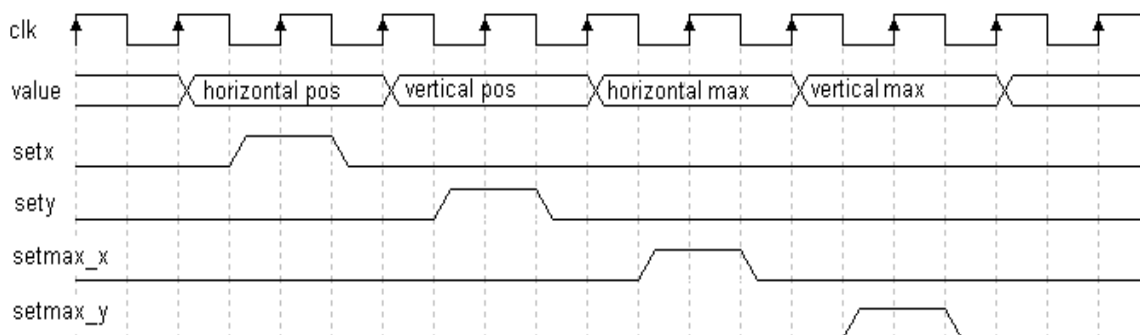
**Figure 9 Y-Axis Inversion**

The current position of the mouse can be set at any time by placing the desired horizontal or vertical position (relative to the monitor's axes origin) on the value input. Then set the set-x input signal (set-y for the vertical position) high for one clock period.

Similarly, the upper bounds of the mouse can be set by placing the value of those bounds on the value input and placing setmax\_x or setmax\_y high for one clock period. This is done when the resolution changes to place the mouse cursor (its hot spot) in the middle of the screen and also prevents the cursor from leaving the visible area.

## Controller Outputs

left	state of the left mouse button ('1' if pressed, or '0')
middle	state of the middle mouse button ('1' if pressed, or '0')
right	state of the right mouse button ('1' if pressed, or '0')
xpos	nine bits, the current horizontal position of the mouse
ypos	nine bits, the current vertical position of the mouse
zpos	four bits, the last Z delta movement received
new_event	high for a clock period after a new data movement packet is received and processed



**Figure 10 Set Mouse Position and Bounds Timing Diagram**

## The Resolution Mouse Informer

The resolution\_mouse\_informer implements the logic that sets the position of the mouse when the FPGA is powered-up or when the resolution changes. It also sets the bounds of the mouse according to the present resolution.

The mouse is centered for the currently-selected resolution and the bounds are set accordingly. This way, the mouse cursor will appear in the center of the screen at start-up, and when the resolution is changed it will not leave the screen. The position, and the bounds, are set by placing the coordinate of the center point on the output value and activating the corresponding set signal (set-x for horizontal position, set-y for vertical position, setmax\_x for horizontal maximum value, and setmax\_y for the vertical maximum value).

### Port Definitions for the Resolution\_mouse\_informer

clk	global clock signal
rst	reset signal
resolution	input pin, from resolution_switcher 0 for 640x480 selected resolution 1 for 800x600 selected resolution
switch	input pin, from resolution_switcher, active for one clock period when resolution changes
value	output pin, 10 bits, to mouse_controller, position on X or Y, max value for X or Y that is sent to the mouse_controller
setx	output pin, to mouse_controller, active for one clock period when the value output contains the horizontal position of the mouse cursor
sety	output pin, to mouse_controller, active for one clock period when the value output contains the vertical position of the mouse cursor
setmax_x	output pin, to mouse_controller, active for one clock period when the value output contains the horizontal maximum position of the mouse cursor
setmax_y	output pin, to mouse_controller, active for one clock period when the value output contains the vertical maximum position of the mouse cursor

---

<sup>i</sup> Source: <http://www.Computer-Engineering.org>, Author: Adam Chapweske, Last Updated: 04/01/03. All information within this article is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. This article is protected under copyright law.