Laboratory Exercise 8

Implementing Device Drivers

The purpose of this exercise is to learn how device drivers can be implemented in embedded systems. You will create a PS/2 mouse driver that works with the VGA display available in an Altera DE-series Media Computer. A block diagram of the DE2 Media Computer is shown in Figure 1. The operational details of the PS/2 port and the VGA display are described in the *DE2 Media Computer for the Altera DE2 Board* manual.

Prerequisite: It is necessary to do Laboratory Exercise 7 to gain the needed background knowledge.

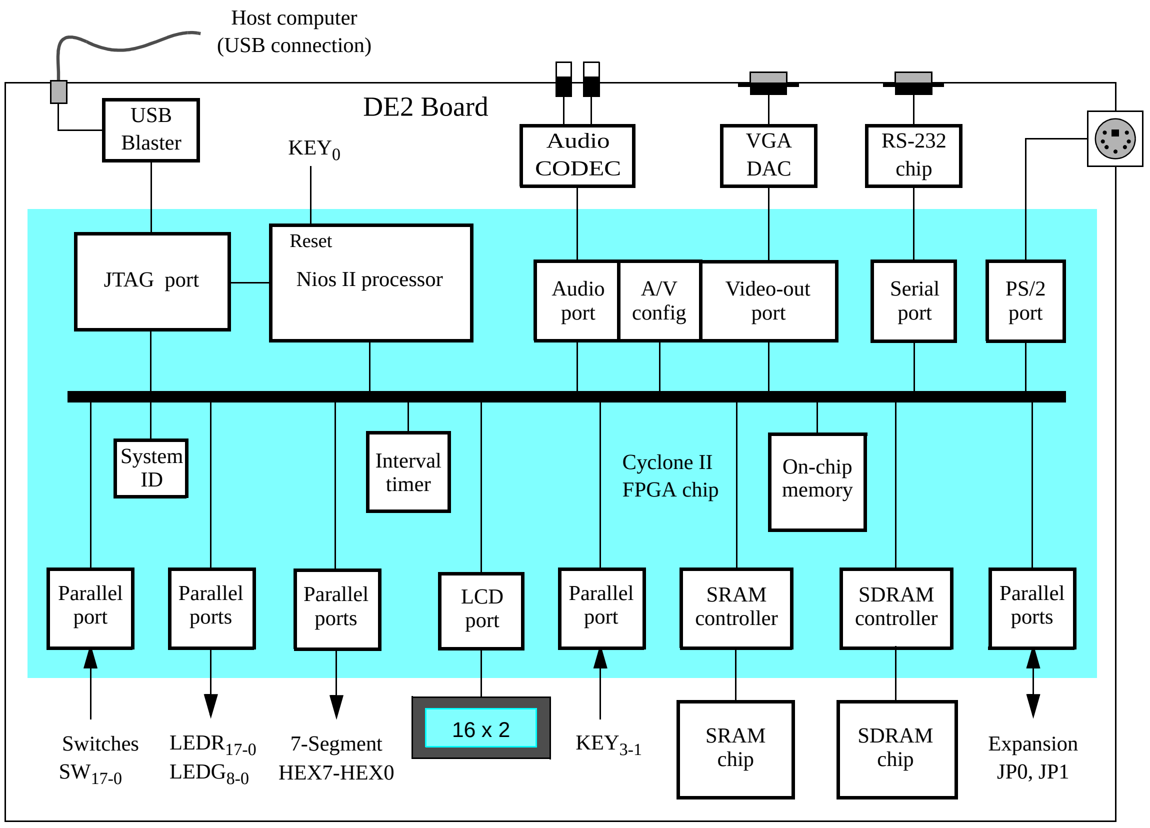

Figure 1: System diagram of the DE2 Media Computer.

Background

A device driver comprises a set of software subroutines that allow computer programs to access a peripheral device, without the need for the programmer to be familiar with the low-level details of how a peripheral device functions. A simple example of a peripheral device we use in computers is a keyboard, which can be connected to a computer via a PS/2 or a USB interface. Programmers who write a C-language program to read a string of characters from the keyboard on a Personal Computer need not know how keystrokes on the keyboard are received by their program. For example, to read an integer from the keyboard, a C programmer uses the statement:

```
scanf("%d", &some_variable);
```

Providing a high-level of abstraction for peripheral devices is the purpose of device drivers. Implementing device drivers often requires a great deal of knowledge on the part of a programmer, not only with respect to how a peripheral device operates, but also the context in which it is used.

Computer mouse is another example of a simple device that can be connected via a PS/2 or a USB interface. Today, a mouse is likely to be connected using the USB connection. Previously, it was common to use the PS/2 interface. Since the PS/2 interface is much simpler, we will use it in this laboratory exercise.

Part I

In this part, you will take the first steps to communicate with the mouse device. To do this, you have to be able to send and receive information from the mouse through the PS/2 port in the DE-series Media Computer.

The PS/2 port in the Media Computer is implemented by a PS/2 Port IP core. This core includes a 256-byte First-In First-Out (FIFO) buffer that stores data received from a PS/2 device.

The interface to the PS/2 port in the Media Computer has two memory-mapped registers, shown in Figure 2.

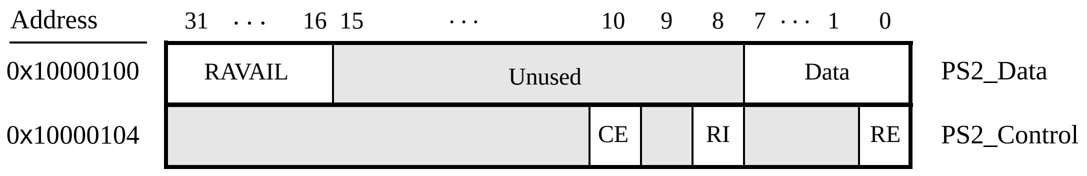

Figure 2: PS/2 Port Memory-Mapped Registers.

The PS2_Data register consists of *RAVAIL* and *Data* fields. The *RAVAIL* field indicates the number of bytes of data received through the PS/2 port that are currently stored in the FIFO buffer. The first byte of the buffer can be read through the *Data* field. Note that reading any part of the PS2_Data register causes the FIFO buffer to eject the first byte from the FIFO. The PS2_Control register is used to enable interrupts for the PS/2 port. Interrupts can be enabled by setting the *RE* field. When *RE = 1*, the PS/2 port generates an interrupt request when *RAVAIL* is greater than 0. While the interrupt is pending, the *RI* field is set to 1, and can be cleared by emptying the PS/2 FIFO. The *CE* field is set by the PS/2 port if an error occurs when a command is sent to a PS/2 device. We will use this register in the later parts of the exercise.

In your first attempt to communicate with the mouse, you will use polling to receive data from the PS/2 port. Do the following:

1. Write a C-language program that polls the PS_Data register and displays the last three bytes received on the 7-segment displays. Your program should first enable communication with the mouse by sending the byte 0xF4 to the PS_Data register. Your program should then continuously read the contents of the PS2_Data register. Each time, check the *RAVAIL* field to determine if the byte in *Data* field is valid data. Only when the number in *RAVAIL* field is larger than 0, should you save the data and display it on the 7-segment displays.
2. Create a new project in the Altera Monitor Program for the Altera DE-series Media Computer and include your source code.
3. Compile and load your program onto the DE-series board.
4. Connect a PS/2 Mouse to the PS/2 port on the board.
5. Run the program and observe the messages received from the mouse device as you move the mouse around and press the mouse buttons.

Part II

In this part, you will enhance your program to properly initialize the mouse device. To do this, you will need to understand some of the commands that can be sent to the mouse (such as the 0xF4 command we used in Part I), as well as the messages that the mouse can send in response to these commands. The necessary commands and mouse responses to the commands are shown in Table 1.

To ensure the proper operation of the mouse device, it is important to reset the mouse and enable it before attempting to process any messages. Thus, your program should first send the reset command to the mouse and

Table 1: Some of the mouse commands along with the expected responses.

Command Description	**Command Byte**	**Response**
Reset the mouse to default mode	0xFF	Responds with a **0xFA** message, followed by a 2-byte message **0xAA00** if successful. A byte **0xFC** will be sent otherwise to indicate an error.
Enable Mouse to send position and button status messages	0xF4	Responds with a single **0xFA** byte if successful.
Disable Mouse	0xF5	Responds with a single **0xFA** byte if successful. Send the Enable mouse command to resume receiving messages about the user's interactions with the mouse.

await a response. If the reset is successful, the mouse will send a 3-byte sequence equal to **0xFAAA00**. Upon receiving this response from the mouse, send the **Enable Mouse** command and wait for the acknowledgement **0xFA**.

Resetting the mouse can fail because of a malfunction, a communication error or because the mouse is not connected to the system. If a mouse malfunctions and does not respond to commands, then it can be treated as not connected. If it responds with the **0xFC** byte, then resend the reset command to ensure that the mouse is reset correctly before proceeding further.

A disconnected mouse may be reconnected at any time and should be initialized properly. Fortunately, when a mouse is plugged into a PS/2 port it will reset itself and send a 2-byte message **0xAA00** as though a reset command was sent to it. Your program should watch for the **0xAA00** message from the mouse. If this sequence is detected, your program should send the reset command to the mouse followed by the mouse enable command.

Write a C-language program that communicates with a mouse through the PS/2 port on the DE-series Media Computer. Your program should correctly reset and enable the mouse device, as well as handle the case of the mouse being reconnected to the system while the program is running.

Part III

In this part, you will learn how to process messages that the mouse device sends to specify the change in its position and the state of its buttons. The change in position is indicated by two 9-bit signed numbers, one for the change in the horizontal position, and one for the change in the vertical position. Moving the mouse to the right causes the horizontal change in mouse's position to be positive, and moving the mouse to the left is indicated by a negative value. Similarly, when you move the mouse forward, the change in vertical position is positive, and when you pull it towards you it is negative. The state of mouse buttons is specified using three bits, one per button in a basic 3-button mouse. When a button is pressed the corresponding bit is set to 1, otherwise the bit is set to 0.

By default, a mouse that is enabled will send 3-byte packets through the PS/2 port to communicate its state. The format of the packet is shown in Table 2.

Table 2: Three-byte mouse data packet.

	bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
byte 1	y overflow	x overflow	y sign	x sign	1	Middle btn	Right btn	Left btn
byte 2	Mouse x movement							
byte 3	Mouse y movement							

The first byte has bit 3 set to 1 to indicate that the two bytes to follow are part of a 3-byte data packet. In addition, it contains the state of three buttons in bits 2 through 0, as well as the sign and overflow bits for the movement along the x and y axes. The remaining two bytes contain 8-bit parts of x and y movement of the mouse.

Write a C-language program that initializes the mouse and processes the 3-byte movement packets received from the mouse. Your program should decode the packets and display the state of the buttons on **LEDR[2..0]**, the change in x position on **HEX[7..4]** and the change in y position on **HEX[3..0]**.

Part IV

In this part, you will handle the mouse messages using an Interrupt Service Routine. The interrupt service routine (ISR) is a function that is executed when an interrupt occurs. An interrupt will occur when the *RE* bit in the PS/2 control register is set to 1 and the PS/2 port has a message ready to be read.

Modify your C-language program from Part III to do this by implementing four functions:

1. initialize_driver - enable the driver to process messages from the mouse. This function should reset the mouse and enable interrupts for the PS/2 port in the DE-series Media Computer.
2. disable_driver - disable the processing of messages from the mouse. Also, disable interrupts for the PS/2 port.
3. get_mouse_change - get the most recent change in position and mouse buttons as reported by the mouse.
4. PS2_ISR - the interrupt service routine that handles the mouse events.

Note that the procedure for dealing with interrupts in a DE-series Media Computer is the subject of Laboratory Exercise 4.

Part V

In this part, you will enhance your device driver to keep track of the location of the mouse and limit its movements to specified boundaries. To do this, you will define five variables: two to keep track of the current horizontal and vertical position of the mouse, one to keep the state of mouse buttons, and two to specify the maximum horizontal and vertical coordinates at which the mouse can be located.

You are to create a mouse device driver comprising the following functions:

1. initialize_driver - as in Part IV.
2. disable_driver - as in Part IV.
3. set_mouse_bounds(max_x, max_y) - set the maximum x and y coordinates a mouse can have.
4. get_mouse_state - read the current position of the mouse on the screen and the state of its three buttons.
5. get_mouse_change - as in Part IV.
6. PS2_ISR - as in Part IV.

Write a program that shows the position of the mouse using **HEX[7..4]** to display the x position, and **HEX[3..0]** to display the y position. Set the maximum x and y coordinates of the mouse to be 319 and 239, respectively.

Part VI

In this part, you are to implement routines to draw the mouse pointer on the screen. You will use the VGA cores in the DE-series Media Computer to draw the mouse pointer on the screen. These cores implement the VGA interface that is able to produce an image on a computer screen at a resolution of 320 by 240 pixels. The image to be drawn on the screen is stored in memory and can be easily changed. Refer to Laboratory Exercise 7 for details that you need to perform the following parts.

To draw a mouse pointer on the screen, it is necessary to define the shape of the pointer as well as a secondary buffer that contains the contents of the screen beneath the mouse pointer. The shape of the pointer is drawn on the screen just like any other image. You should use the secondary buffer to store the part of the screen on which the mouse is being drawn. Then, when the mouse is moved in the future, you will be able to restore the contents of the screen easily and move the mouse to a new location.

To define the shape of the mouse pointer use a matrix of 16 rows by 8 columns. Each entry in the matrix should contain one of three values:

1. 0 - to indicate that the correpsonding pixel of the mouse pointer is black.
2. 1 - to indicate that the correpsonding pixel of the mouse pointer is white.
3. -1 - to indicate that the correpsonding pixel of the mouse pointer is transparent. Transparent pixels are not drawn when the mouse pointer is being drawn.

The matrix and the image of the mouse pointer are given in Figure 3. Note that light blue is used to indicate the transparent pixels.

0	-1	-1	-1	-1	-1	-1	-1
0	0	-1	-1	-1	-1	-1	-1
0	1	0	-1	-1	-1	-1	-1
0	1	1	0	-1	-1	-1	-1
0	1	1	1	0	-1	-1	-1
0	1	1	1	1	0	-1	-1
0	1	1	1	1	1	0	-1
0	1	1	1	1	0	0	0
0	1	1	1	0	-1	-1	-1
0	0	0	1	0	-1	-1	-1
0	-1	0	1	0	-1	-1	-1
-1	-1	-1	0	1	0	-1	-1
-1	-1	-1	0	1	0	-1	-1
-1	-1	-1	-1	0	1	0	-1
-1	-1	-1	-1	0	1	0	-1
-1	-1	-1	-1	-1	0	-1	-1

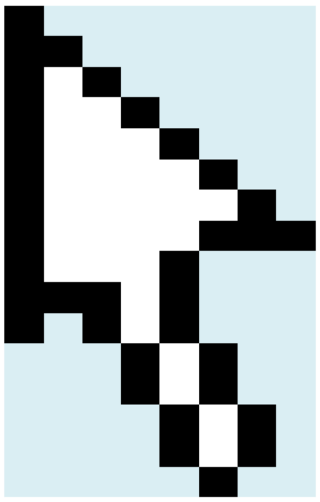

Figure 3: The matrix and the image of the mouse pointer.

The secondary buffer should also consist of 16 rows and 8 columns, where each entry stores the color of the pixel over which the mouse pointer will be drawn.

Adapt your mouse device driver to move the mouse pointer on the screen. Define two new functions for your driver:

1. draw_mouse - store the image underneath the mouse pointer and draw the mouse pointer at the current location. Use the matrix described above to draw the mouse pointer on the screen, assuming that the top-left corner of the mouse pointer image is located at the current location of the mouse. This function should only be visible to the driver.
2. erase_mouse - erase the mouse from the screen by redrawing the image beneath the mouse. This function should only be visible to the driver.

Modify your PS/2 Interrupt Service Routine to call the above two functions whenever the mouse position changes.

Run your program from Part V and connect the VGA output of the DE-series board to a computer monitor. You should be able to see the coordinates of the mouse pointer on the HEX displays and the mouse pointer on the screen, as you move the mouse.

Part VII

In this part, you are to combine the mouse pointer with animation on the screen. Write a program to implement a simple game where a filled rectangle moves and bounces off the edges of the screen. The player moves the mouse to position its pointer on the moving image of the rectangle, at which point clicking the left button on the mouse should change the color of the rectangle and stop its motion. When the button is released, the rectangle should start moving again. The game will be more interesting if you acclerate the speed or shrink the size of the rectangle after each successful catch.

To perform the animation, every 30th of a second you should hide the mouse pointer, erase the rectangle, draw the rectangle at a new location, and then show the mouse pointer again. You should define two new functions:

1. hide_mouse - causes the driver to erase the mouse from the screen. The driver should still process messages from the mouse, and update its position and the state of the buttons, but the mouse pointer should not be drawn on the screen.

2. show_mouse - enables the driver to draw the mouse pointer on the screen. If the pointer was previously hidden, the driver should draw the pointer when this function is called.

To wait a 30th of a second, we can take advantage of the fact the VGA controller continuously redraws the image from the memory onto the screen and one redraw cycle takes a 60th of a second. We can ask the VGA pixel buffer to let us know when it has finished drawing the most recent image, as explained in Laboratory Exercise 7. Waiting for two such cycles, gives us the 30th of a second delay we seek.

When you run your program you may notice that as you move the mouse pointer, parts of it are left on the screen, despite the fact that you were careful to erase and redraw it in both your ISR and in your main program. The reason is that an interrupt can occur while your main program is hiding/redrawing the mouse. You will need to adjust your driver to prevent this.

Preparation

The recommended preparation for this laboratory exercise includes C code for Parts I through VII.