Pong On BoosterPack LCD

Overview:
For this lab, you will be developing a Pong game that will make use of the EDUMKII BoosterPack to display a paddle and a ball bouncing around the screen. We will also make use of timer and ADC joystick interrupts to implement the game.

In Part 1, you implemented a bouncing ball program, where a ball moves across the TFT LCD and bounces off the edges of the screen. In Part 2, you will implement the Pong game using joystick interrupts to control the movement of the paddle.

Project Initialization

The easiest way to begin Part 2 is to make a copy of your Part 1 project and then modify it. You can give the new project a new name, such as BouncingBall_part2 or pong.

MSP Graphics Library:

This section of the lab will only work within projects that contain the appropriate library files as Part 1 of this lab. You should initialize the LCD as described in Part 1 of the lab. This section will walk you through the basics of using the graphics library to draw the paddle and print strings on the BoosterPack's LCD screen. The following resource is available as references for the MSP Graphics Library if desired:

- Graphics Library User Guide: https://github.com/energia/msp432-core/blob/7c9a3689dc4dc3c9a7d00fa6cb51c4ed5b260afb/libraries/EduBPMKII_Screen/LCD_screen%20-%20Reference%20Manual.pdf

The paddle will be drawn as a rectangle using the function dRectangle(). Example 1 shows the example code for drawing and erasing a red paddle on a white background.

```
void erase_paddle(uint16_t x, uint16_t y, uint16_t dx, uint16_t dy) {

    // draw over old paddle with background color to erase it
    myScreen.dRectangle(0, PaddleY, dx, dy, whiteColour);
}

void draw_paddle(uint16_t x, uint16_t y, uint16_t dx, uint16_t dy) {
    myScreen.dRectangle(0, OldPaddleY, dx, dy, redColour);
}
```

Example 1: Code to erase and draw a paddle after proper initialization

Interrupt Service Routine Guidelines

For this lab you will have two interrupt service routines – the SysTick interrupt from Part 1 and now the y-axis joystick interrupt from the ADC. Here are some guidelines for your interrupt service routines.

- You should inititialize the ADC ISR as done in Lab 5 (we are using single sample mode in this assignment).

- If a variable is modified within an ISR it must be declared as a global variable with the keyword volatile.

- Interrupt service routines pause the normal flow of the program to handle an interrupt. It is important that the handlers be short and fast. ISRs usually set flags or update variables that are then used or checked elsewhere in your normal program code.

Part 2 Program Guidelines

In the Part 1 program, you have learned how to move a ball around the screen and bounce off the edges of the screen. In this part, you will use a joystick interrupt to move a paddle up and down along the left side of the screen and detect when the ball bounces off the paddle. If the ball misses the paddle on the left side of the screen, the game is over.

Your project should conform to the following specifications:

- At the start of the game, you should redraw the circle in time-steps of about 500 ms. That is, the SysTick interrupt frequency should be around 2 Hz.
    - Use the Systick timer for accurate timing.
    - Do not draw or erase circles from within the timer's ISR.

- You should have 5 difficulty levels for this game. Each difficulty level corresponds to the velocity of the ball (i.e. difficulty 5 corresponds to 5 pixels traveled per interrupt signal). Upon initialization of the game, you may set the difficulty to a set value but after one game is over you should let the user select the difficulty.

- You should not draw or erase the paddle from within the GPIO ISRs. Instead, you should set a flag within the GPIO ISR and draw the paddle in the main function when it detects that the flag has been set. (The main function will be responsible for clearing the flag after it detects it has been set.)

- The circle should have a radius of 3 or 4 pixels.

- The circle should start near the center of the screen. i.e., $50 < x < 70$, $50 < y < 70$

- The circle should start with an initial velocity of:
    - v_x = (-1 * difficulty) pixels / step
    - v_y = (-1 * difficulty) pixels / step

- The paddle should have a maximum length of 20 pixels and should have a constant dx of 4. For a more challenging game, you can reduce the length of the paddle.

- Your program should keep track of the number of successful hits in a game.

- To detect if the ball hits the paddle, you should check if the leftmost point on the ball i.e. the point (x – radius, y) intersects with the paddle. Thus, when the ball's leftmost point is equal to the x coordinate of the paddle (4), the y coordinate of the ball should be within the range of y values for the paddle. If this occurs, then the ball should bounce off the paddle – you will change the sign of v_x from negative to positive. However, when the ball's leftmost point is equal to the x coordinate of the paddle (4) and the y coordinate of the ball is *not* within the range of y values for the paddle, the game is over.

- At the end of the game, your program should disable all interrupts and print "Game Over" and the number of successful hits to the LCD. Additionally if the user presses S1 on the EDUMKII, you should restart the game. If the user presses S2 on the EDUMKII, you should increase the difficulty of the game. If the difficulty is at 5 when S2 is pressed, you should revert the game difficulty back to 1.