Bouncing Ball on BoosterPack LCD

Overview:
For this assignment you will be developing a Pong game that will make use of the EDUMKII BoosterPack's LCD. You will use SysTick and ADC14 joystick interrupts to implement the game.

This lab will be split into two parts. In this part, you will implement a bouncing ball program, where a ball or circle moves across the BoosterPack's LCD and bounces off the edges of the screen. In the second part, you will implement the Pong game using GPIO interrupts to control the movement of the paddle.

Project Initialization

In order to complete this lab, you must become familiar with some important functions of the MSP graphics library in Energia.

MSP Graphics Library:

This section will explain the basics of using the graphics library to draw a ball on the BoosterPack's LCD screen. The following resources are available as references for the Graphics Library:

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

First, we need to make sure to include the necessary libraries and functions to initialize our LCD screen. The necessary libraries are presented in figure 1. The "Screen_HX8353E.h" library is written in C++ which establishes the screen as an object with assignable attributes. The begin function initializes the screen, and the clear function sets a solid color background for the screen (black in this case). As always you must remember to disable the watchdog.

```
// Core library for code-sense
#include <driverlib.h>

// Include necessary libraries
#include "SPI.h"
#include "Screen_HX8353E.h"
Screen_HX8353E myScreen;

void setup() {
    //Initialize myScreen
    myScreen.begin();
    myScreen.clear(blackColour);
}
```

Figure 1: LCD Initialization

The important function for this lab is the “circle” function. This will draw a circle on the screen with a user defined radius and center coordinate. Example 1 shows code for drawing and erasing a white circle on a black background. Notice that in this example, our screen is initialized as “myScreen.”

```
void erase(int16_t x, int16_t y, int16_t radius) {
    // draw over old with background color to erase it
    myScreen.circle(x, y , radius, blackColour);
}

void draw(int16_t x, int16_t y, int16_t radius) {
    // draw circle with default foreground color
    myScreen.circle(x, y , radius, whiteColour);
}
```

Example 2: Code used to erase and draw a circle after proper initialization.

Part 1 Program: Bouncing Ball

For this part of the lab you will develop a program that will use SysTick interrupts to make a ball move across the screen with a constant velocity and bounce off the edges of the screen.

Your program must satisfy the following specifications:

- You should redraw the circle in time-steps of about 500 ms. The clock signal for SysTick is MCLK in our processor is 3 MHz so you must set the SysTick period to flag the interrupt at a rate of 2 Hz. For more information check out the SysTick documentation:

http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP432_Driver_Library/latest/exports/driverlib/msp432_driverlib_3_21_00_05/doc/MSP432P4xx/html/driverlib_html/group__systick__api.html

- Do not draw or erase circles from with the SysTick ISR. Instead, set a flag in the SysTick ISR to signal the main function to erase the old ball and draw the new ball.

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

- The circle or ball should start somewhere near the center of the screen. The screen is 128x128 pixels so both x and y pixel coordinates range from 0 to 127. The origin, point (0,0), is in the upper-left corner.

- The circle should start with an initial velocity of :
    - vx = 1 pixel / time-step

    - vy = -1 pixel / time-step

- If adding the x-velocity to the current x-coordinate will push the radius of the circle up to the screen's left or right boundaries, then the x-velocity should be negated. (aka vx = -vx). An alternate method is to use a flag to indicate if the ball is moving in the positive or negative x direction.

- Similarly, for the y-velocity, if adding the y-velocity to the current y-coordinates will push the radius of the circle up to the screen's top or bottom boundaries then the y-velocity should be negated. An alternate method is to use a flag to indicate if the ball is moving in the positive or negative y direction.

- A simple flowchart is shown in Figure 3. This flowchart shows the update of x, y, vx and vy being done before the circle is erased and redrawn. This ordering is not required. However, by keeping the delay small between the erasure of the old circle and the drawing of the new circle, it will make the movement of the circle appear smoother. This program sequence implies that the old x and old y values are still available for erasing the old circle after x and y are updated. You can easily keep track of the previous x and y values in variables, such as *oldx* and *oldy*. You don't have to follow this flowchart exactly, but you do need to do all the calculations and drawing and erasing the ball within the main function. You should try to make the motion of the ball as smooth as possible.

- Note that in order to update vx and vy, your program needs to check if the circle has reached one or perhaps two boundaries. Since the vx and vy velocities have magnitudes of one, your program can always detect when the circle just touches one or two of the boundaries. Note that it is possible for the ball to go into a corner and hit two boundaries simultaneously.

- Avoid using Serial.print() during the main loop of your program and your ISR in order to maintain accurate timing. (Printing to the serial terminal is very slow.)

- We recommend using uint16_t for x, y and radius since these are specified in the prototypes of the graphics library functions.

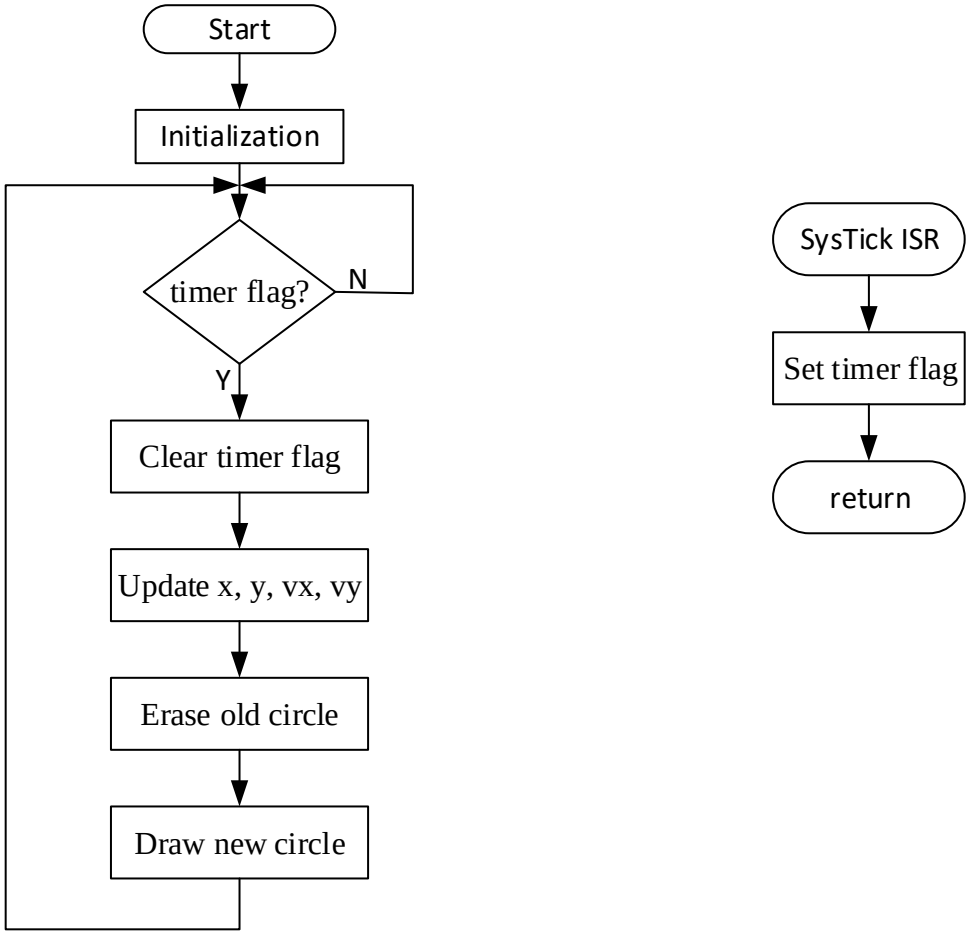

Figure 3. Flowcharts for main function and SysTick ISR

Extra Challenge Assignment (Optional)

To make the bouncing ball program more interesting, you can modify the program as follows:

- When the ball bounces off the wall at x=0 or x=127, increase the *magnitude* of the x-velocity by one up to a maximum of 15.
- Similarly, when the ball bounces off the wall at y=0 or y=127, increase the *magnitude* of the y-velocity by one up to a maximum of 15.
- The ball must ***never*** disappear off the screen. If the velocity would move the ball past an edge, your program should calculate where the ball should be if it bounced off the edge when it reached it.
- Keep track of the number of times the ball hits the wall. After the ball bounces off the walls 50 times, reset the velocities to have magnitude 1. The ball should keep moving in the current direction and from the current position with the new velocities. The ball should speed up each time it hits the wall, as before.