PONG - New and Improved

ENSC 332

Nicholas Flandin (301062840)
Dan Hendry (30113878)

Simon Fraser University
School of Engineering Science

August 9, 2010

Course Instructor: M. Moallem

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This report describes an implementation of the classic two player game PONG using an HCS12 microprocessor and a serial controlled, OLED (organic light emitting diode) screen. Chapter 2 describes various functionality required by the project including an overview of the methods used to control the OLED. A description of the techniques used in communication between the HCS12 and OLED, as well as the method used to create pseudrandom numbers, and an overview of techniques which allow C and assembly code to be used in tandem is also presented. Chapter 3 contains details of the PONG program itself, including how the players' paddles and the ball are moved and controlled, and how the LCD (liquid crystal display) is used to display a scores. Finally, a complete listing of source code is presented in appendix A.

1.1 Rules and Game Objectives

The rules of this game are quite simple, and very similar to the original PONG's. Two players, using buttons, are able to move rectangular 'paddles' on the OLED screen. Movement is constrained in the vertical direction, within the bounds of the screen. A ball bounces between the two paddles. The ball begins in the center of the screen, and begins moving in a pseudorandomly selected direction, continuing to move in the along the same vector until it reaches either the top or bottom of the screen, which cause it to assume the opposite velocity in the y-direction; or either the left or right side of the screen, which causes the player whose paddle is on the opposite side to the screen to have scored a point. Either player may reflect the ball, using their paddle, by having the ball come into contact with the paddle, causing the ball to take on a pseudorandom velocity in the opposite x-direction. The game objective of each player, scoring on their opponent is tracked on the LCD screen on the Dragon12 board.

1.2 System Components

The figure below illustrates the basic configuration of the system used in this project, which is an integrated 'Dragon12' board consisting of an HCS12 microprocessor, with a variety of other hardware. One of the integrated devices on the Dragon12 is an LCD directly connected to a digital I/O port of the microprocessor. It consists of two-lines of alphanumeric text and is used to display the current score. A full color, 320 by 240 pixel OLED external to the Dragon12 board was connected to another one of the microprocessor's digital I/O ports, and communicates with it via RS232 protocol, for the purpose of serving as the game display.

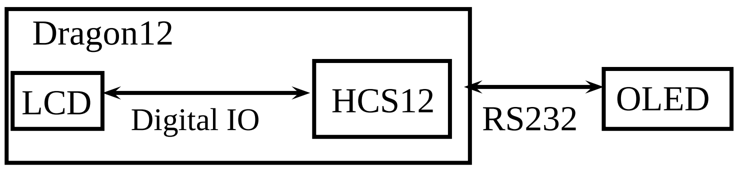

Figure 1.1: System Diagram

Chapter 2

Support Functionality

2.1 Serial Interface

Communication between the HCS12 micro-controller and OLED uses RS-232, a two wire, duplex, asynchronous serial communication protocol. Of the two hardware communication blocks provided by the HCS12, SCI1 with external pins PS2 and PS3, was used to send and receive commands as described in section 2.2. Code used to interact with the relevant hardware registers may be seen in appendix A.3. A list of pertinent registers is given in Table 2.1.

Table 2.1: Registers used for serial port control

Register Name	Function	Value
SCI1BDH	MSB baud rate selector	0
SCI1BDL	LSB baud rate selector	156 (For 9600 Buad)
SCI1CR1	Control Register	0
SCI1CR2	Control Register	0x0C (enable transmit and receive circuits)

2.2 OLED and Drawing Commands

The OLED is controlled using simple commands sent over the serial interface. Code related to OLED control is presented in appendix A.5. The majority of the file contains functions which wrap serial communication commands to implement specific functionality. Three general classes of OLED commands used for this project are detailed in sections 2.2.1 through section 2.2.3. Each function follows the following general template.

1. Send command byte

2. Send additional data required by the command

3. Wait for confirmation to be sent from the OLED (either an ACK or NAK response byte)

2.2.1 Device Control Commands

Two device control commands are used in this project and implemented in the OLED specific code file (appendix A.5). The first is used to initialize the OLED and is part of the initialization function. The second, OLED_Clear(), reverts all drawing commands and causes the screen to reset to its default color (black).

2.2.2 Drawing Commands

Basic drawing functions were created to interact with the OLEDs graphics processor. These include functions to draw lines, rectangles, circles, triangles and single pixels. Each requires a set of verticies to be sent consisting of a pair of 16-bit numbers representing x and y coordinates. Most functions also require a 16-bit integer representing the color to be sent. A function to convert a standard triplet of red, green and blue colors into a specially formatted 16 bit integer is shown. Drawing commands are only issued after the screen has been reset. During game play, only the function used to animate is used.

2.2.3 Animation

Gameplay animations are created using a simple copy-paste command. During each update cycle (see section 3.1), the copy-paste command is used to move one area of the screen to another. The screen bounds for gameplay objects, such as the player paddles and the ball, are stored in memory or computed on the fly to be used used when calling this command. One limitation when using this technique is that it does move the area of the screen, but copies it. As such, a blank buffer area must be included around each visual object such that when moving it, edges of the object dont appear to be left behind. Without using much more complicated logic, this requirement has three important impacts. First of all, it limits the maximum speed objects can move. This maximum speed is determined by the size of the buffer area. Second, objects never actually appear to touch. This is a particularly poor user interface feature given paddles and balls never contact. Finally, it severely limits the extent to which background textures and images may be implemented.

2.2.4 Limits of Baud Rate

The OLED module being used was severely constrained based on the baud rate it operates at. The only baud rate at which commands could be successfully sent was 9600. During gameplay, three animation commands are sent each frame. A command to update the balls position, and two to update player paddle locations, conditional on interface buttons having been pressed. Due to the limited rate commands can be sent at, the speed of the ball may appear to change if players move their paddles. A speed factor is used to correct for this effect. The number of pixels the ball is moved each update cycle depends on whether players are also moving their paddles. To minimize truncation errors associated with integer math, this speed factor is implicitly considered some decimal value multiplied by 1000.

2.3 Random Values - Linear Feedback Shift Register

Ball speed has the possibility of being randomized whenever it hits a players paddle. A linear feedback shift register is used to generate pseduorandom numbers. The basic technique used to generate pseduorandom is given below.

1. Start with some seed value.
2. Each time a random value is requested compute a binary value (1 or 0) based on the current random value. An XOR is performed on three existing bits.
3. Perform a logical shift on the current random value, sifting in the value of the new bit.

2.4 Functions - Calling Assembly From C

A major component of this project was combining C and assembly code. Assembly code may be 'in-lined' within a C function. For clarity however, all assembly used in this project was abstracted into dedicated functions, callable from C. Any function written in assembly but callable from C should have a header file with the C function prototype. The function name is simply an assembler label within the .asm source file. The *XDEF* directive must also be used on the function name (assembly label).

To effectively write C functions in assembly, it is important to know how arguments are passed to a function and how values are returned from a function. By examining assembler code generated by the C compiler, the following rules were determined to be the calling convention for functions. It is important to note that the rules listed below are by no means exhaustive, were experimentally determined, and are valid only for simple data types such as *int*s and *char*s.

- The *CALL* instruction is used to invoke a function (from the assembly compiled C code) and assembly returns when the *RTC* instruction is used. The CALL instruction stores three bytes on the stack.
- 8 bit return values are stored in the B register.
- 16 bit return values are stored in the D register (A + B).
- The last function argument is stored in the D (16 bit) or B (8 bit) register when calling.
- All other function arguments are pushed onto the stack, ordered from first to second last. When accessing these values a 3 position offset must be used to account for the stack locations used by the *CALL* function.

- It is very important that any space allocated on the stack by the assembly subroutine be cleaned up before it returns.

2.5 Button Interfacing

Four buttons from the keypad are used as player controls. The assembly code used to detect button presses, is given in appendix A.9. This code is of particular interest as it demonstrates the implementation of a C function, taking two *char* arguments and returning a *char* value, in assembly. The function name, *chkbtn* is simply a label. The second function argument, is placed in the B register when the function is called. It is used to create a mask for selecting the row of column of buttons to check. The firs argument, the return value to use, is accessed via the stack using the *4,sp* relative address. Four in this case is used to skip over the three stack locations used by the *CALL* instruction. Finally, the value the function is to return is loaded into the B register before any *RTC* instruction is used.

Chapter 3

PONG Code

3.1 Main Program Loop

Core logic for the PONG game is implemented in C. A main function containing an infinite loop is documented as part of appendix A.1. This file also contains support functions and code used to control the ball. The program starts by performing various initialization actions. These include:

- Setting the data direction register for port A such that it may be used to power and detect keypad button presses.
- Setting PB0 as an output. It is connected to the OLED module and used for resetting.
- Initializing the serial communication hardware used for interfacing to the OLED.
- Resetting and initializing the OLED module.
- Initializing and displaying the initial score on the LCD display.
- Initializing the C structures used for tracking each players paddle.

After initialization, the program enters an 'outer' infinite loop then an 'inner' loop which breaks whenever a player scores. Each iteration of this inner loop encapsulates an animation step including checking if players are moving their paddles, visually changing paddle locations if necessary, and updating the ball location. A speed factor is used to account for the limited baud rate supported by the OLED module; see section 2.2.4 for more information.

3.2 Player Paddles

Appendix A.6 and A.7 contains the code used for controlling and displaying player paddles. The data for each paddle is encapsulated in a C structure. Functions related to paddle control interact with these structures declared in *main.c* (appendix A.1). Checks are performed in the 'paddle_move' function to ensure the player is unable to move their paddle off the screen in either direction.

3.3 Ball Movement

Code related to ball movement is included in the *main.c* file. There are a few important points which should be mentioned. First, graphics functions are called int the init fucntion to create an image of a soccer ball. Due to limitation of the OLED, the center pentagon had to be drawn as a set of five triangle. Additionally, triangle vertices's must be drawn in a counter clockwise order.

Most core logic related to the ball is implemented in its move function. Specifically, there is code used to detect when one player scores on another. Randomization of the balls direction vector is also performed whenever it hits a players paddle.

3.4 LCD Scoreboard

The LCD, which is used by the system as a scoreboard is controlled through the use of four assembly-coded subroutines, which call another three communications subroutines, (all included in one source file), each of which performs a very specific function. These may be seen in appendix A.11. The code was split into these subroutines to handle initialization, the left player scoring, the right player scoring, updating the display output, sending communications data to the LCD sending ASCII characters individually, and a delay respectively. The reason for the separate right and left player scoring subroutines is that it is very difficult to handle pass-through variables when calling an assembly-coded subroutine from a C-coded program. The initialization code simply sets the memory where the scores are set to zero, and the left and right player scoring subroutines increment their respective scores. The LCD update subroutine sends configuration data to the LCD and converts the player scores to BCD, then to ASCII, and sends them to the LCD, along with some text for the benefit of the user. There are two subroutines which handle sending communications data and characters to be displayed on the LCD, to the port which connects the microcontroller to the LCD, they are separated for ease of coding in other sections, one being use for sending communications data, and the other for sendigng ASCII characters, these subroutines send a single byte at each call, four bits at a time, and largely deal with configuring the port and handling delays to allow communications to take place. There is also a simple delay function which is used by the LCD communications subroutines to provide proper signal pulse width.

Chapter 4

Results and Conclusion

Despite the limiting OLED baud rate, PONG was successfully implemented on a screen external to the HCS12 and Dragon12 development board. Functions written in assembly were created and found to be reliably callable from C code. Serial control was found to be an effective means to send commands to an external device. The Dragon12's LCD screen was also used in tandem with other display mechanisms.

Appendix A

Project Code Files

A.1 main.c

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */

#include "graphics.h" //Drawing functions
#include "interface.h" //Serial interface

#include "paddle.h"
#include "chkbtnasm.h"

#include "displayScores.h"

#define ever (;;)

paddle p1_aloc;
paddle p2_aloc;

paddle *p1 = &p1_aloc;
paddle *p2 = &p2_aloc;

//Ball data
int ball_x_min;
int ball_y_min;
int ball_x_max;
int ball_y_max;

int ball_dir_y;
int ball_dir_x;
char ball_wait_to_die;

#define P_HEIGHT 8
#define P_WIDTH 36
#define BALL_R 16

unsigned int lfsr = 0xACE1;
unsigned int rand()
{
	//Return a random value
	//Pseudo random numbers generated by linear feedback shift register

	unsigned int bit;
	bit  = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5) ) & 1;
	lfsr =  (lfsr >> 1) | (bit << 15);

	return lfsr;
}

void ball_init()
{
	char l = 6;
	char s = 3;
	unsigned int cx = 240/2 - 94;
	unsigned int cy = 320/2;
	int fillcolor = GetRGB(0, 0, 0);

	ball_x_min = cx - BALL_R;
```

```
        ball_x_max = cx + BALL_R;

        ball_y_min = cy - BALL_R;
        ball_y_max = cy + BALL_R;

        ball_dir_x = 4;

        // Randomize the y direction
        if(rand() % 2)
        {
                ball_dir_y = 6;
        }
        else
        {
                ball_dir_y = -6;
        }


        ball_wait_to_die = 0;

        //Basic ball
        OLED_DrawCircle(cx, cy , BALL_R - 7, 0, GetRGB(255, 255, 255));


        //Triangle patches of the pentagon
        OLED_DrawTriangle(
                cx - s, cy - l,
                cx - (s+l - 2), cy,
                cx, cy,
                fillcolor);
        OLED_DrawTriangle(
                cx + l, cy - s,
                cx - s, cy - l,
                cx, cy,
                fillcolor);
        OLED_DrawTriangle(
                cx + l, cy - s,
                cx, cy,
                cx + l, cy + s,
                fillcolor);
        OLED_DrawTriangle(
                cx, cy,
                cx - s, cy + l,
                cx + l, cy + s,
                fillcolor);
        OLED_DrawTriangle(
                cx - (s+l - 2), cy,
                cx - s, cy + l,
                cx, cy,
                fillcolor);

        // Seams
        OLED_DrawLine(
                cx - (s+l - 2), cy,
                cx - (s+l - 2) - 5, cy,
                fillcolor);
        OLED_DrawLine(
                cx - s, cy - l,
                cx - s*3, cy - l*3,
                fillcolor);
        OLED_DrawLine(
                cx + l, cy - s,
                cx + l*3, cy - s*3,
                fillcolor);
        OLED_DrawLine(
                cx + l, cy + s,
                cx + l*3, cy + s*3,
                fillcolor);
        OLED_DrawLine(
                cx - s, cy + l,
                cx - s*3, cy + l*3,
                fillcolor);
}

char ball_move(int speedfactor)
{
        char switch_dir = 0;

        //Check side bounce
        if(ball_x_min + ball_dir_x < 0 || ball_x_max + ball_dir_x > 240)
        {
```

```
        ball_dir_x = -ball_dir_x;
}

//See if the ball has been lost
if(ball_y_min + ball_dir_y < 0)
{
        return -1;
}

else if(ball_y_max + ball_dir_y > 320)
{
        return 1;
}

// See if the ball is in the p1 'court'
if(ball_y_min + ball_dir_y <= p1->y + p1->height && ~ball_wait_to_die)
{
        if(p1->x > ball_x_max || p1->x + p1->width < ball_x_min)
        {
                //Player has missed
                ball_wait_to_die = 1;
        }
        else
        {
                //Player is has saved
                switch_dir = 1;
        }
}

// See if the ball is in the p2 'court'
if(ball_y_max + ball_dir_y >= p2->y && ~ball_wait_to_die)
{
        if(p2->x > ball_x_max || p2->x + p2->width < ball_x_min)
        {
                //Player has missed
                ball_wait_to_die = 1;
        }
        else
        {
                //Player is has saved
                switch_dir = -1;
        }
}

//Peform bounce off of paddle
if(switch_dir)
{
        //Add some randomness to the balls direction
        switch(rand() % 8)
        {
                // Normal
                case 0:
                {
                        ball_dir_y = 4*switch_dir;
                        ball_dir_x = 4*(1 - 2*(((unsigned int)ball_dir_x) >> 15));
                        break;
                }
                // Faster in y
                case 1:
                {
                        ball_dir_y = 6*switch_dir;
                        ball_dir_x = 4*(1 - 2*(((unsigned int)ball_dir_x)>> 15));
                        break;
                }
                // Faster in x
                case 2:
                {
                        ball_dir_y = 4*switch_dir;
                        ball_dir_x = 6*(1 - 2*(((unsigned int)ball_dir_x) >> 15));
                        break;
                }
                // Even faster in y
                case 3:
                {
                        ball_dir_y = 7*switch_dir;
                        ball_dir_x = 4*(1 - 2*(((unsigned int)ball_dir_x)>> 15));
                        break;
                }
                // Even faster in x
                case 4:
                {
```

```
                    ball_dir_y = 4*switch_dir;
                    ball_dir_x = 7*(1 - 2*(((unsigned int)ball_dir_x) >> 15));
                    break;
                }
                //No speed change
                default: ball_dir_y = -ball_dir_y; break;
            }
        }

        //Do the screen copy
        OLED_CopyPaste( ball_x_min, ball_y_min,
                        ball_x_min + (ball_dir_x*1000)/speedfactor, ball_y_min + (
                            ball_dir_y*1000)/speedfactor,
                        2*BALL_R + 1,
                        2*BALL_R + 1);

        //Update position variables
        ball_x_min += (ball_dir_x*1000)/speedfactor;
        ball_x_max += (ball_dir_x*1000)/speedfactor;

        ball_y_min += (ball_dir_y*1000)/speedfactor;
        ball_y_max += (ball_dir_y*1000)/speedfactor;

        return 0;
}

void main(void)
{
        char result = 0;

        //Variables for delays
        unsigned int temp = 0;
        unsigned int temp2 = 0;

        EnableInterrupts;

        DDRB = 0x01; // Not really needed -> just need pb0
        DDRA = 0x0F; // For keypannel

        //Basic initialization
        SCI_Init();
        OLED_Init();
        initLCD();
        updatedisplay();

        //Initialize the paddle structures
        paddle_init(p1, PADDLE_RIGHT, P_WIDTH, P_HEIGHT);
        paddle_init(p2, PADDLE_LEFT, P_WIDTH, P_HEIGHT);

        for ever //Valley girl style
        {
                //Clear the screen for full redraw
                OLED_Clear();

                //Draw the player paddles, one red the other blue
                paddle_draw(p1, 4, 1, GetRGB(255,0,0));
                paddle_draw(p2, 4, 1, GetRGB(0,0,255));

                //Draw the ball
                ball_init();

                result = 0;

                //A brief delay for the players to recoup
                for(temp2=0; temp2 < 300; temp2++)
                for(temp=0; temp < 50000; temp++) {}

                while(result == 0)
                {
                        char p1move;
                        char p2move;
                        int speedfactor = 1800;

                        p1move = chkbtn(4, 1);
                        p2move = chkbtn(4, 4);

                        paddle_move(p1, p1move);
                        paddle_move(p2, p2move);

                        //Speed the ball up if copy-paste commands have been sent for the paddles
                        if(p1move) speedfactor -= 400;
```

```
				if(p2move) speedfactor -= 400;

				result = ball_move(speedfactor);
			}

			if(result > 0)
			{
				rscored();
			}
			else if(result < 0)
			{
				lscored();
			}

			updatedisplay();
		}
}
```

A.2 interface.h

```c
#ifndef _INTERFACE_H
#define _INTERFACE_H

//Module for serial communication

void SCI_Init();

char SCI_InStatus();

char SCI_InChar();

void SCI_OutChar(char data);

void SCI_OutWord(unsigned int data);

#endif /* _INTERFACE_H */
```

A.3 interface.c

```c
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */

#include "interface.h"

#define RDRF 0x20 // Receive Data Register Full Bit
#define TDRE 0x80 // Transmit Data Register Empty Bit

void SCI_Init()
{

        SCI1BDH = 0;

        // 24000000/(16 x 9600)
        SCI1BDL = 156;

        // Simple configuration
        SCI1CR1 = 0;

        // Enable TX and RX functionality
        SCI1CR2 = 0x0C;
}

char SCI_InChar()
{
        while((SCI1SR1 & RDRF) == 0){};
        return(SCI1DRL);
}

void SCI_OutChar(char data)
{
        while((SCI1SR1 & TDRE) == 0){};
        SCI1DRL = data;
}

// Checks if new input is ready, TRUE if new input is ready
char SCI_InStatus()
{
        return(SCI1SR1 & RDRF);
}

void SCI_OutWord(unsigned int data)
{
        SCI_OutChar(data >> 8);
        SCI_OutChar(data & 0xFF);
}
```

A.4 graphics.h

```
#ifndef _GRAPHICS_H
#define _GRAPHICS_H


#define OLED_DETECT_BAUDRATE     0x55

#define OLED_CLEAR                              0x45
#define OLED_COPYPASTE                  0x63

#define OLED_LINE                               0x4C
#define OLED_CIRCLE                             0x43
#define OLED_PUTPIXEL                   0x50
#define OLED_RECTANGLE                  0x72
#define OLED_TRIANGLE                   0x47

#define OLED_ACK   0x06  // Ok
#define OLED_NAK   0x15  // Error


#define OLED_DIE_INIT    1
#define OLED_DIE_CLEAR   2
#define OLED_DIE_DRAW    3

void OLED_Die(char code);

void OLED_ResetDisplay(void);

int OLED_GetError(void);

void OLED_Init(void);

int GetRGB(int red, int green, int blue);

void OLED_Clear(void);

void OLED_PutPixel(unsigned int x, unsigned int y, int color);

void OLED_DrawCircle(unsigned int x, unsigned int y, unsigned int radius, char filled, int color);

void OLED_DrawRectangle(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int
    color);

void OLED_DrawLine(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int color);

void OLED_DrawTriangle(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2,
    unsigned int x3, unsigned int y3, int color);

void OLED_CopyPaste(unsigned int xs, unsigned int ys, unsigned int xd, unsigned int yd, unsigned
    int width, unsigned int height);

#endif /* _GRAPHICS_H */
```

A.5 graphics.c

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */

#include "graphics.h"
#include "interface.h"


void OLED_Die(char code)
{
        PORTB = code;

        //Something has gone wrong, do nothing more my young Padawan
        while(1) {}

        return;
}

void OLED_ResetDisplay(void)
{
        //Stagnating variators
        unsigned int x,y;
```

```
        //Drop it like its hot
        PORTB &= ~0x01;

        //Delay - make sure it has time to shutdown
        for(x=0;x<1000;x++)
                for(y=0;y<1000;y++);

        //Send the reset pin high again
        PORTB |= 0x01;

        //Give it some time to boot back up
        for(x=0;x<2000;x++)
                for(y=0;y<1000;y++);

        return;
}

// Return codes:
//      0 = ack
//      1 = nak
//      2 = unknown
int OLED_GetError(void)
{
        byte incomingByte = OLED_ACK;

        //Wait for valid data
        while (!SCI_InStatus()) {}

        //Retrieve that data
        incomingByte = SCI_InChar();

        //Check the response
        if(incomingByte == OLED_ACK)
        {
                //Everything is OK
                return 0;
        }
        else if(incomingByte == OLED_NAK)
        {
                //Curse the electron god.
                //I believe his name is bill.
                return 1;
        }
        else
        {
                //Pull your hair out and jump up and down screaming with frustration.
                //Or start break dancing, your choice really.
                return 2;
        }
}

void OLED_Init(void)
{
        unsigned int x,y;

        //Assum reset is connected to B0
        DDRB |= 0x01;
        PORTB |= 0x01;

        OLED_ResetDisplay();

        //Short delay - let the OLED restart
        for(x=0;x<1000;x++)
                for(y=0;y<1000;y++);


        SCI_OutChar(OLED_DETECT_BAUDRATE);
        if (OLED_GetError())
        {
                OLED_Die(OLED_DIE_INIT);
        }

        return;
}

int GetRGB(int red, int green, int blue)
{
        int outR = ((red * 31) / 255);
        int outG = ((green * 63) / 255);
        int outB = ((blue * 31) / 255);
```

```
        return (outR << 11) | (outG << 5) | outB;
}

void OLED_Clear(void)
{
        SCI_OutChar(OLED_CLEAR);

        if (OLED_GetError())
        {
                OLED_Die(OLED_DIE_CLEAR);
        }

        return;
}

void OLED_PutPixel(unsigned int x, unsigned int y, int color)
{
        SCI_OutChar(OLED_PUTPIXEL);

        SCI_OutWord(x);
        SCI_OutWord(y);

        SCI_OutWord(color);

        if (OLED_GetError())
        {
                OLED_Die(OLED_DIE_DRAW);
        }

        return;
}

void OLED_DrawCircle(unsigned int x, unsigned int y, unsigned int radius, char filled, int color)
{
        SCI_OutChar(OLED_CIRCLE);
        SCI_OutWord(x);
        SCI_OutWord(y);

        SCI_OutWord(radius);
        SCI_OutWord(color);

        if (OLED_GetError())
        {
                OLED_Die(OLED_DIE_DRAW);
        }

        return;
}

void OLED_CopyPaste(unsigned int xs, unsigned int ys, unsigned int xd, unsigned int yd, unsigned
    int width, unsigned int height)
{
        SCI_OutChar(OLED_COPYPASTE);
        SCI_OutWord(xs);
        SCI_OutWord(ys);

        SCI_OutWord(xd);
        SCI_OutWord(yd);

        SCI_OutWord(width);
        SCI_OutWord(height);

        if (OLED_GetError())
        {
                OLED_Die(OLED_DIE_DRAW);
        }

        return;
}

void OLED_DrawRectangle(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int
    color)
{
        SCI_OutChar(OLED_RECTANGLE);
        SCI_OutWord(x1);
        SCI_OutWord(y1);

        SCI_OutWord(x2);
        SCI_OutWord(y2);

        SCI_OutWord(color);
```

```
	if (OLED_GetError())
	{
		OLED_Die(OLED_DIE_DRAW);
	}

	return;
}

void OLED_DrawLine(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2, int color)
{
	SCI_OutChar(OLED_LINE);
	SCI_OutWord(x1);
	SCI_OutWord(y1);

	SCI_OutWord(x2);
	SCI_OutWord(y2);

	SCI_OutWord(color);

	if (OLED_GetError())
	{
		OLED_Die(OLED_DIE_DRAW);
	}

	return;
}

void OLED_DrawTriangle(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y2,
    unsigned int x3, unsigned int y3, int color)
{
	SCI_OutChar(OLED_TRIANGLE);
	SCI_OutWord(x1);
	SCI_OutWord(y1);

	SCI_OutWord(x2);
	SCI_OutWord(y2);

	SCI_OutWord(x3);
	SCI_OutWord(y3);

	SCI_OutWord(color);

	if (OLED_GetError())
	{
		OLED_Die(OLED_DIE_DRAW);
	}

	return;
}
```

A.6 paddle.h

```c
#ifndef _PADDLE_H
#define _PADDLE_H

typedef struct
{
        int x;
        int y;
        unsigned char width;
        unsigned char height;
} paddle;

typedef enum PaddleTypeType
{
        PADDLE_RIGHT,
        PADDLE_LEFT
} PaddleType;

void paddle_init(paddle* ppaddle, PaddleType ptype, unsigned char width, unsigned char height);

void paddle_draw(paddle* ppaddle, unsigned char buffer_x, unsigned char buffer_y, int color);

void paddle_move(paddle* ppaddle, int amt);


#endif /* _PADDLE_H */
```

A.7 paddle.c

```c
#include "derivative.h"      /* derivative-specific definitions */

#include "paddle.h"
#include "graphics.h"

void paddle_init(paddle* ppaddle, PaddleType ptype, unsigned char width, unsigned char height)
{
        ppaddle->x = 240/2 - width/2;
        ppaddle->height = height;
        ppaddle->width = width;

        switch(ptype)
        {
                case PADDLE_RIGHT:
                {
                        ppaddle->y = 0;
                        break;
                }
                case PADDLE_LEFT:
                {
                        ppaddle->y = 319 - height;
                        break;
                }
                default:
                {
                        // Die
                        for(;;){}
                        break;
                }
        }


}
void paddle_draw(paddle* ppaddle, unsigned char buffer_x, unsigned char buffer_y, int color)
{
        OLED_DrawRectangle(     ppaddle->x + buffer_x, // x min
                                                ppaddle->y + buffer_y, // y min
                                                ppaddle->x + ppaddle->width - buffer_x, // x max
                                                ppaddle->y + ppaddle->height - buffer_y, // y max
                                                color);
}

void paddle_move(paddle* ppaddle, int amt)
{
        if(ppaddle->x + amt < 0)
        {
                // Would be off screen on the top - dont update
```

```
                return;
        }
        else if(ppaddle->x + ppaddle->width + amt > 239)
        {
                // Would be off screen on the bottom - dont update
                return;
        }

        if(amt != 0)
        {
                //Only send update commands when needed

                OLED_CopyPaste( ppaddle->x, ppaddle->y,
                                                ppaddle->x + amt, ppaddle->y,
                                                ppaddle->width + 1 ,
                                                ppaddle->height + 1);

                ppaddle->x += amt;
        }
}
```

A.8 chkbtnasm.h

```
#ifndef _CHKBTN_ASM_H
#define _CHKBTN_ASM_H

char chkbtn(char amt, char btncol);

#endif /* _CHKBTN_ASM_H */
```

A.9 chkbtnasm.asm

```
; export symbols
        XDEF chkbtn

; Include derivative-specific definitions
        INCLUDE 'derivative.inc'

; code section
ChkBtnCode:     SECTION


; Return: char
; Arg 1: char amt
; Arg 2: char btncol
chkbtn:

        ; Create the mask for detecting button presses
        ; The amount to shift is passed to this function (chkbtn)
        ; in register B.
        LDAA #%0001000
chkbtn_sl:
        TSTB
        BEQ chkbtn_sld
        LSLA
        DECB
        BRA chkbtn_sl
chkbtn_sld:
        PSHA ; Store the mask on the stack

        ;Check first button
        MOVB #%00000010,PORTA
        LDAA PORTA
        ANDA 0,sp ;Apply the mask
        BEQ chkbtn_skp1 ; Skip ahead since the button has not been pressed
        LDAB 4,sp ; Load the return value (second function argument) into B - the result
        LEAS 1,sp ; Clear the space allocated for the mask
        MOVB #0,PORTA ; Power down the buttons
        RTC
chkbtn_skp1:

        ;Check second button
        MOVB #%00000001,PORTA
        LDAA PORTA
        ANDA 0,sp ;Apply the mask, its almost like halloween
        BEQ chkbtn_skp2
        LDAB 4,sp ; Load the return value
        NEGB ; Opposite direction (down)
        LEAS 1,sp ; Clear the allocation for the mask
        MOVB #0,PORTA
        RTC
chkbtn_skp2:

        ; Niether button was pressed, return 0 (load it into B)
        LDAB #0
        LEAS 1,sp
        MOVB #0,PORTA
        RTC
```

A.10 displayScores.h

```
#ifndef _DISPLAYSCORES_H
#define _DISPLAYSCORES_H

void initLCD(void);
void lscored(void);
void rscored(void);
void updatedisplay(void);

#endif /* _DISPLAYSCORES_H */
```

A.11 displayScores.asm

```
        XDEF initLCD, lscored, rscored, updatedisplay        ;declare subroutines

        INCLUDE 'derivative.inc'


LCD_DATA        EQU PORTK       ;set values for LCD connection
LCD_CTRL        EQU PORTK
RS      EQU mPORTK_BIT0
EN      EQU mPORTK_BIT1


MY_EXTENDED_RAM: SECTION
;-----------------------
R1:  DS.B  1   ;R variables are used for controlling timing (using delays) of communication with
    LCD
R2:  DS.B  1   ;and allocated here
R3:  DS.B  1
TEMP:  DS.B  1 ;TEMP is used in the communication subroutines to store value of accumulator A
    temporarily, and allocated here
lscore:  DS.B  1       ;allocate memory for left player score
rscore:  DS.B  1       ;allocate memory for right player score
eeeCode:     SECTION


DisplayCode: SECTION

;code section
;------------------------------
initLCD:         ;initializes scores
    movb 00,lscore     ;set scores to zero
    movb 00,rscore

        RTC
;------------------------------
lscored:         ;handles left player scoring

        LDAA    lscore  ;load left player score
        ADDA    #1      ;increment left player score
        STAA    lscore  ;store left player score in memory
        RTC
;------------------------------
rscored:         ;handles right player scoring

        LDAA    rscore  ;load left player score
        ADDA    #1      ;increment left player score
        STAA    rscore  ;store left player score in memory
        RTC

;------------------------------
updatedisplay:  ;handles updating display with current score

        LDAA  #$FF      ;the following lines send a few values to the port to prepare the LCD for
            the message
        STAA  DDRK
        LDAA  #$33
        JSR     COMWRT4
        JSR   DELAY
        LDAA  #$32
        JSR     COMWRT4
        JSR   DELAY
        LDAA    #$28
        JSR       COMWRT4
        JSR       DELAY
        LDAA    #$0E
```

```
JSR       COMWRT4
JSR    DELAY
LDAA      #$01
JSR       COMWRT4
JSR    DELAY
LDAA      #$06
JSR       COMWRT4
JSR    DELAY
LDAA      #$80
JSR       COMWRT4
JSR    DELAY
LDAA      #'L'    ;this is where the message begins, with the word 'left'
JSR       DATWRT4
JSR    DELAY
LDAA   #'E'
JSR       DATWRT4
JSR    DELAY
LDAA   #'F'
JSR       DATWRT4
JSR    DELAY
LDAA   #'T'
JSR       DATWRT4
JSR    DELAY
LDAA   #'-'
JSR       DATWRT4
JSR    DELAY

LDAA lscore       ;loads left player score into accumulator A
DAA               ;converts left player score into 2 digit binary coded decimal
TAB               ;copies BCD left player score to accumulator B
LSRA              ;shifts accumulator A right four times, to select the first digit of the
    decimal number
LSRA              ;shift
LSRA              ;shift
LSRA              ;shift
ANDB #$0F         ;mask first decimal digit of the left player score, to select only the
    second bit
ADDA #$30         ;convert first decimal digit of left player score to ASCII
ADDB #$30         ;convert second decimal digit of left player score to ASCII
JSR       DATWRT4        ;sends first decimal digit of left player score to the LCD
JSR    DELAY
TBA                      ;move second decimal digit of left player score to accumulator A
JSR       DATWRT4        ;sends second decimal digit of left player score to the LCD
JSR    DELAY

LDAA      #' '           ;the following code is similar to the code above, and sends a
    label, then
JSR       DATWRT4        ;converts the right player score to BCD and sends it to the screen
JSR    DELAY
LDAA      #'R'
JSR       DATWRT4
JSR    DELAY
LDAA   #'I'
JSR       DATWRT4
JSR       DELAY
LDAA      #'G'
JSR       DATWRT4
JSR       DELAY
LDAA      #'H'
JSR       DATWRT4
JSR       DELAY
LDAA      #'T'
JSR       DATWRT4
JSR       DELAY
LDAA   #'-'
JSR       DATWRT4
JSR    DELAY

LDAA      rscore
DAA
TAB
LSRA
LSRA
LSRA
LSRA
ANDB #$0F
ADDA #$30
ADDB #$30
JSR       DATWRT4
JSR    DELAY
TBA
```

```
        JSR         DATWRT4
        JSR    DELAY
        RTC


;--------------------------------------------------
COMWRT4:                            ;handles sending communication data to LCD ofur bits at a time
                  STAA  TEMP        ;pushes value in accumulator A to memory
                  ANDA  #$F0        ;masks right four digits in accumulator A
                  LSRA              ;shift right twice, to adjust value
                  LSRA
                  STAA  LCD_DATA            ;sends accumulator A value to the LCD's port
                  BCLR  LCD_CTRL,RS         ;clears LCD RS pin value in case it was set
                  BSET  LCD_CTRL,EN         ;enables LCD control pin
                  NOP                       ;short delay
                  NOP
                  NOP
                  BCLR  LCD_CTRL,EN         ;clears LCD control pin
                  LDAA  TEMP                ;reloads value from beginning of subroutine to accumulator
                      A
                  ANDA  #$0F                ;performs same operations as code above, to send four
                      least significant bits in accumulator A to the LCD
        LSLA
        LSLA
                STAA   LCD_DATA
                  BCLR  LCD_CTRL,RS
                  BSET  LCD_CTRL,EN
                  NOP
                  NOP
                  NOP
                  BCLR  LCD_CTRL,EN
                  RTS
;-------------------------
DATWRT4:                            ;handles sending ASCII characters to LCD, similar to COMWRT, but
    in separate subroutine for easier differentiation elsewhere in code
                  STAA    TEMP
                  ANDA    #$F0
                  LSRA
                  LSRA
                  STAA    LCD_DATA
                  BSET    LCD_CTRL,RS
                  BSET    LCD_CTRL,EN
                  NOP
                  NOP
                  NOP
                  BCLR    LCD_CTRL,EN
                  LDAA    TEMP
                  ANDA    #$0F
        LSLA
      LSLA
                STAA    LCD_DATA
                BSET    LCD_CTRL,RS
                  BSET    LCD_CTRL,EN
                  NOP
                  NOP
                  NOP
                  BCLR    LCD_CTRL,EN
                  RTS
;------------------------------
DELAY           ;delay subroutine

        PSHA              ;Save Reg A on Stack
        LDAA     #1
        STAA     R3
;-- 1 msec delay. The Serial Monitor works at speed of 48MHz with XTAL=8MHz on Dragon12+ board
;Freq. for Instruction Clock Cycle is 24MHz (1/2 of 48Mhz).
;(1/24MHz) x 10 Clk x240x100=1 msec. Overheads are excluded in this calculation.
L3      LDAA     #100
        STAA     R2
L2      LDAA     #240
        STAA     R1
L1      NOP           ;1 Intruction Clk Cycle
        NOP           ;1
        NOP           ;1
        DEC      R1   ;4
        BNE      L1   ;3
        DEC      R2   ;Total Instr.Clk=10
        BNE      L2
        DEC      R3
        BNE      L3
;-------------------------
```

```
        PULA                    ; Restore Reg A
        RTS
;-------------------
```