CECS 347 - Fall 2023 - Project 3

Space Invader Game

By

Edward Ramos – Maryith Garzon - Rod Agatep (GROUP 9)

6 December 2023

Design a Space Invader Game that is displayed on a Nokia 5110 LCD

Introduction

The purpose of this project is to design, code, and build a space invader game with a minimum of three different shaped invaders (enemies) and one spaceship (player). The spaceship will be controlled by a potentiometer. The breadboard left button will start the game, and the right button will fire a bullet.

Operation

Video: https://youtu.be/zl5rA7izmJQ?feature=shared

This program runs on a TM4C123G ARM Cortex Microcontroller; the code is written on Keil uVision5 and is downloaded to the board using the included USB cable. In this project, the LCD will display "Space – Invader – Press SW2 – To Start" (the hyphens represent the next line). The user will press switch 2 to start the game; the potentiometer is used to move the spaceship from left ot right, the right button on the breadboard will fire a bullet. When a bullet is fired, the speaker will output a sound; if an enemy is hit by the bullet, another sound will be heard and your score counter will increase by 1. When the game ends, the LCD will display "Game Over - Nice Try! – Your Score – *users score*".

Theory

Nokia 5110 LCD (BLUE)

Displays the Space Invaders Game; the main character, the bullets/lasers, the enemies, and the GAME OVER screen.

LM386 Audio Amplifier

Generates the simple sounds for shotting and hit, connected to the DAC circuit.

Speaker / Buzzer

Outputs the firing and explosion sound.

4-Bit R2R DAC Circuit

Generates the simple sounds for shotting and hit, connected to the LM386 audio amplifier.

Potentiometer

Aka a variable resistor, max value of 10k ohms. Used to control the spaceship from left to right.

Pushbuttons

2 pushbuttons on the breadboard. Left button (PD0) is used to start the game, right button (PD1) is used to fire a bullet at the enemies.

Resistors

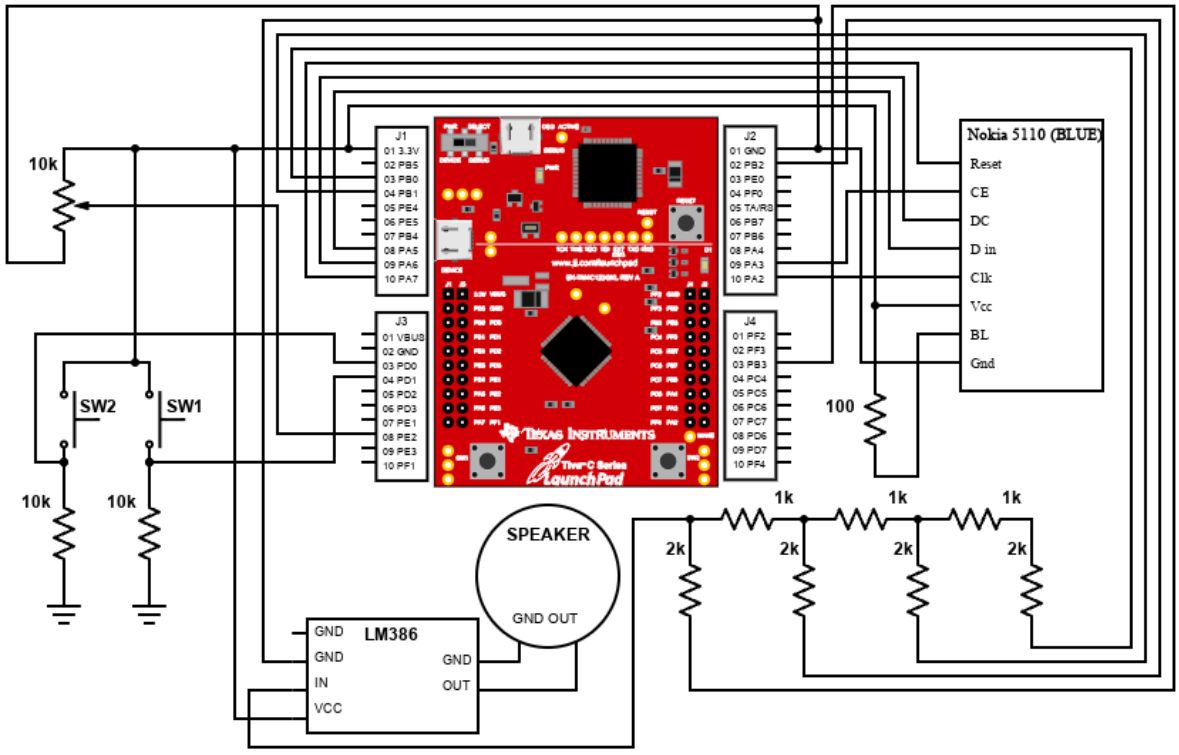
Resistors of a variety of values are used throughout our project, and in the 4-bit R2R DAC circuit.

Systick

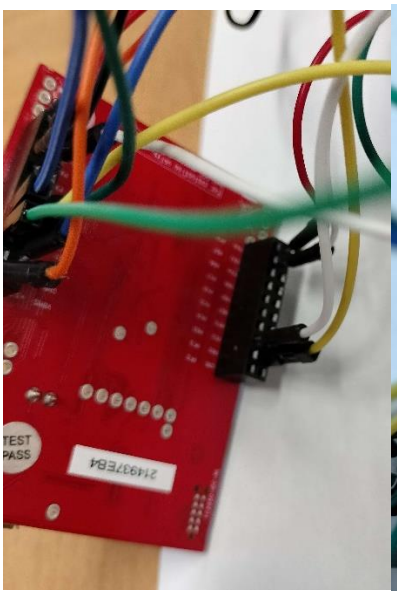
Screen refresh at 10 Hz (with interrupt enabled)

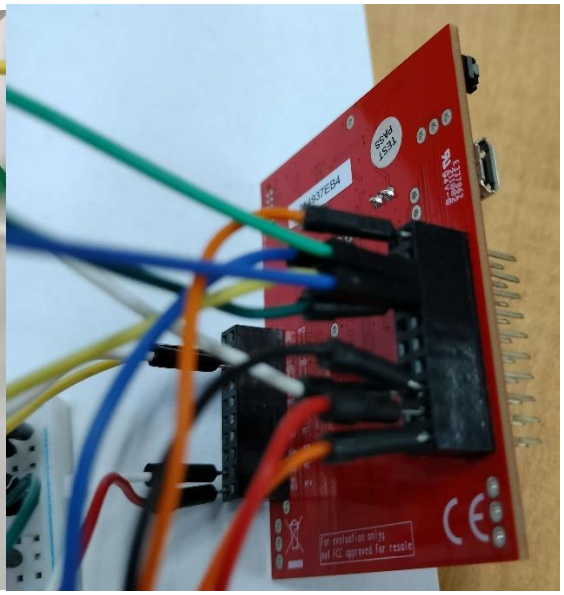
Hardware Design

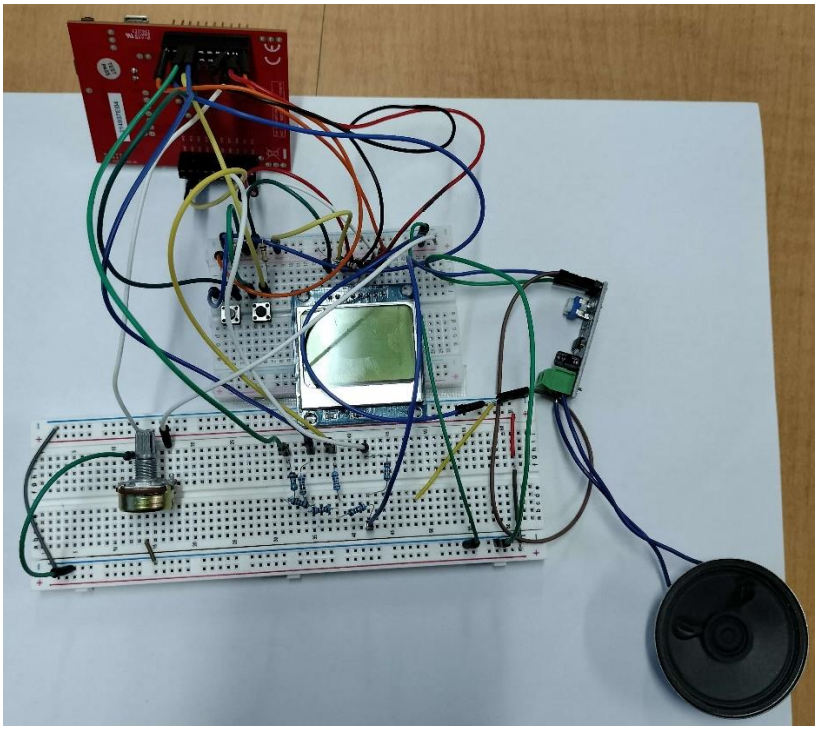
SCHEMATIC

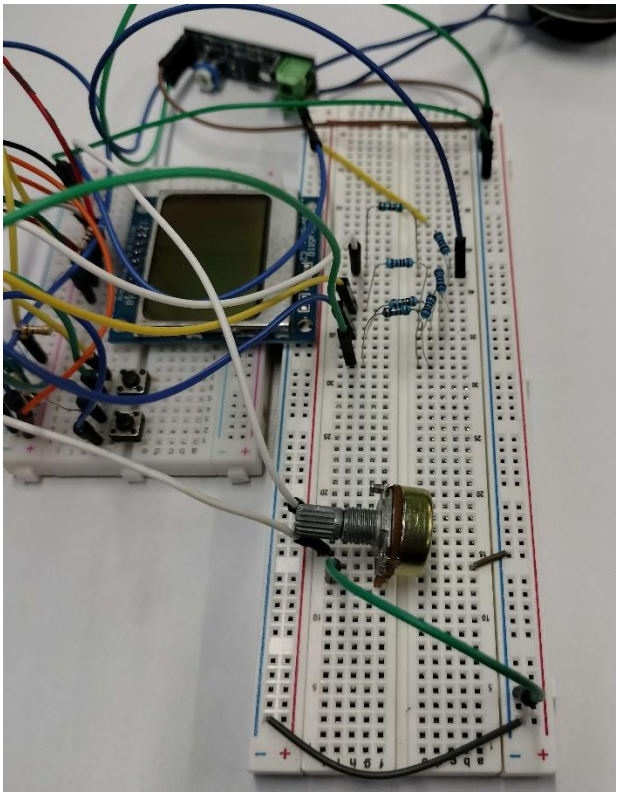


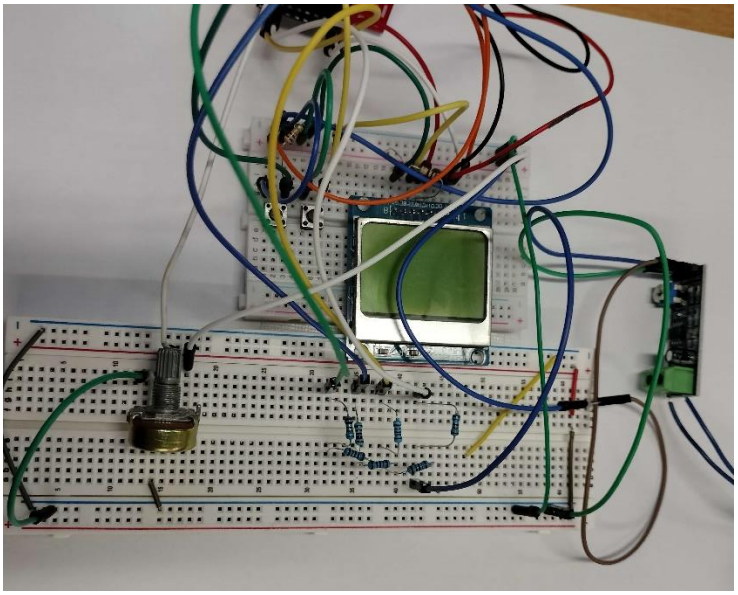
LAUNCHPAD AND CIRCUIT

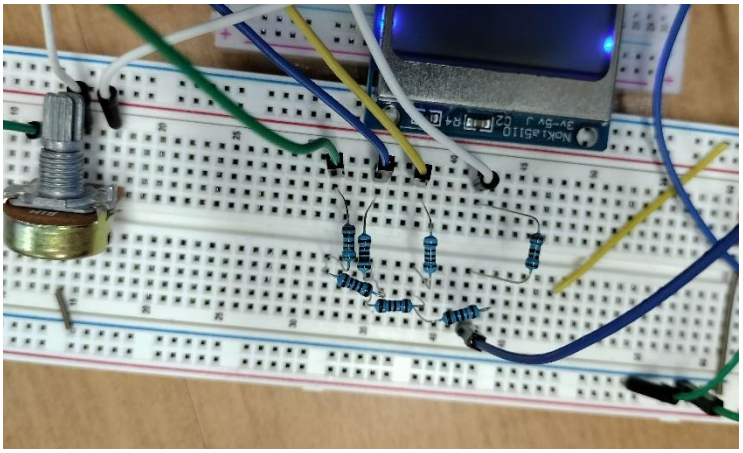


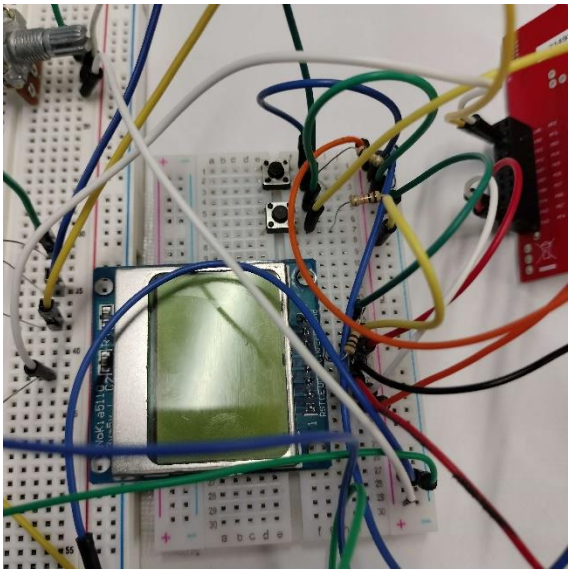


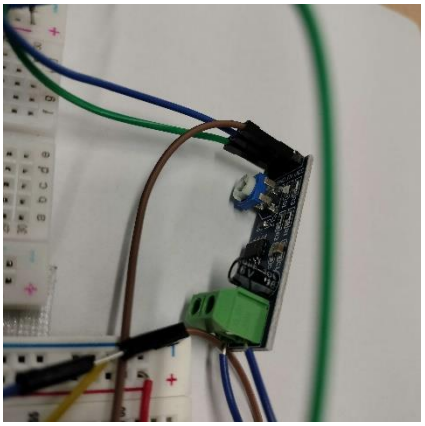


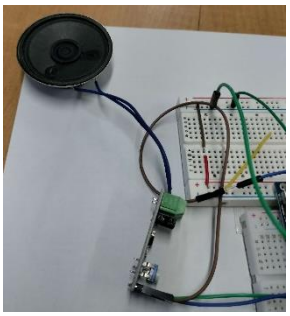






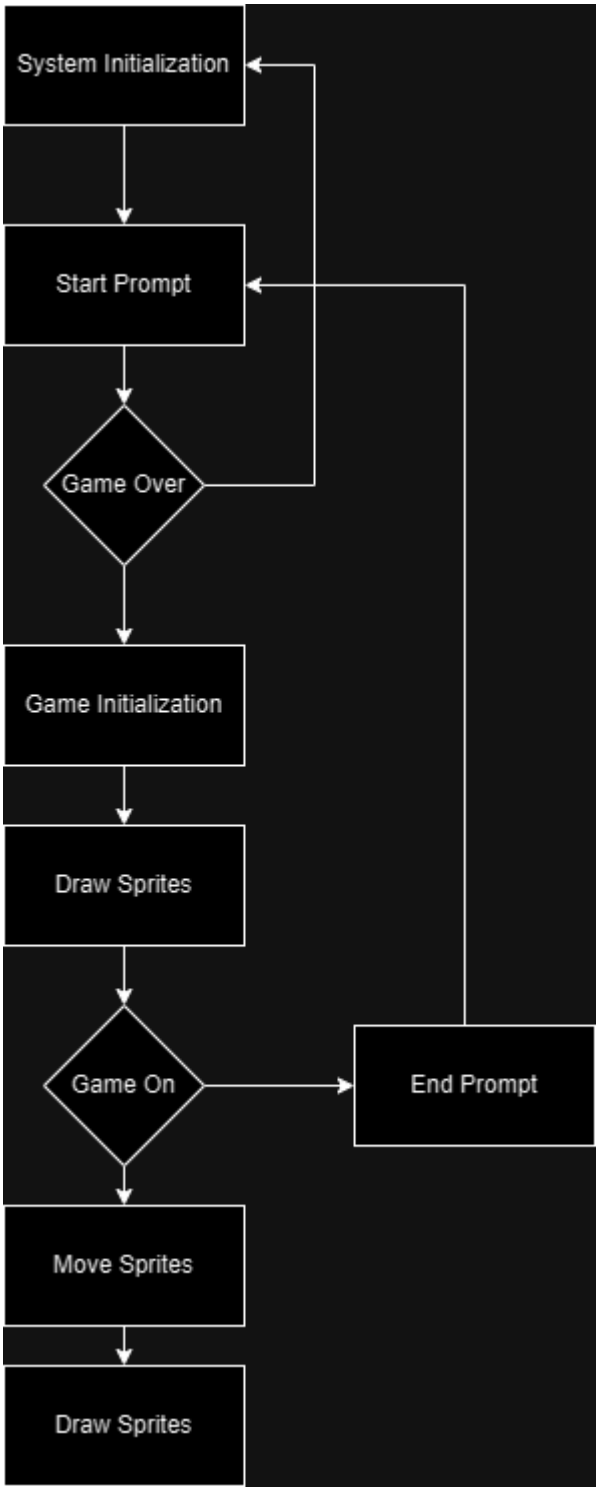






Software Design

Flowchart



SOURCE CODE

SpaceInvadersV5.c

```
#include "tm4c123gh6pm.h"
#include "Nokia5110.h"
#include "PLL.h"
#include <stdint.h>
#include "ADC.h"
#include "Sound.h"
#include "SysTick.h"  // for SysTick_Init()
#include <stdbool.h>
/*
#include "ADC.h"
*/
//#include "Switches.h" // optional module for teh two onboard switches

//  function delays 3*ulCount cycles
//void Delay(uint32_t ulCount);
void Delay100ms(void);
```

```
enum game_status{OVER,ON};
enum life_status{DEAD, ALIVE};
enum enemy_posture{CLOSE, OPEN};

#define PLAYERW     ((unsigned char)PlayerShip0[18])
#define PLAYERH     ((unsigned char)PlayerShip0[22])
#define ENEMY10W    16
#define LASERH      9
#define LASERW      2
#define BULLETH     LASERH
#define BULLETW     LASERW

#define MAX_ENEMYX MAX_X-ENEMY10W


struct State {
  unsigned long x;      // x coordinate
  unsigned long y;      // y coordinate
  const unsigned char *image; // ptr->image
  long life;            // 0=dead, 1=alive
};
typedef struct State STyp;
STyp Enemy[3];
STyp PlayerShip;
STyp Bullet, SmallExplosion;

// Function prototypes
extern void EnableInterrupts(void);  // defined in startup.s
extern void DisableInterrupts(void); // defined in startup.s
```

```
257  void Game_Init(void);
258  void Move(void);
259  void Draw(void);
260  void Start_Prompt(void);
261  void End_Prompt(void);
262  void Switch_Init(void);
263  void System_Init(void);
264
265  // global variables used for game control
266  uint8_t time_to_draw=0;
267  uint8_t delay=0;
268  uint8_t game_s=OVER;
269  uint8_t score=0;
270  int trigger_btn;
271  uint8_t counter = 0;
272  unsigned short playerScore = 0;
273
274  void GPIOPortD_Handler(void){
275    Delay100ms();
276    if(GPIO_PORTD_RIS_R&0x01){ //sw2 Start the game, change status to ON
277      //Software acknowledges the flag and clears the read-only RIS bit by setting ICR
278      GPIO_PORTD_ICR_R = 0x01;
279      game_s=ON;
280    }
281
282    if(GPIO_PORTD_RIS_R&0x02){ //sw1 shoot a bullet
283      //Software acknowledges the flag and clears the read-only RIS bit by setting ICR
284      GPIO_PORTD_ICR_R = 0x02;
285      trigger_btn = 1;
286    }
```

```
287  }
288  int main(void){
289    System_Init();
290
291    while(1){
292      Start_Prompt();
293        while(game_s==OVER){}; //Starts with over because it is not beginning
294
295        Game_Init(); // all sprites: 3 sprites
296        Draw();
297        while (game_s==ON) {
298          if (time_to_draw){
299            Move();
300            Draw();
301            time_to_draw = 0;
302          }
303        }
304      End_Prompt();
305    }
306    }
307
308  void System_Init(void){
309    DisableInterrupts();
310    PLL_Init(Bus80MHz);                    // set system clock to 80 MHz
311    SysTick_Init();
312    Switch_Init();
313    ADC_Init();
314    Nokia5110_Init();
315    Nokia5110_ClearBuffer();
316    Nokia5110_DisplayBuffer();      // draw buffer
```

```
317   EnableInterrupts();
318 }
319
320 // Display the game start prompt
321 void Start_Prompt(void){
322   Nokia5110_Clear();
323   Nokia5110_OutString("             Space       Invaders    Press SW2   To Start");
324   Delay100ms();                 // for board: delay ~1 sec at 80 MHz
325 }
326
327 // Display the game end prompt for 2 seconds
328 void End_Prompt(void){
329     delay = 0;
330     Nokia5110_Clear();
331     Nokia5110_OutString(" Game Over   Nice Try!   Your Score: ");
332     Nokia5110_OutUDec(playerScore);
333     while(delay<30){};
334 }
335
336 // Initialize the game: initialize all sprites and
337 // reset refresh control and game status.
338 void Game_Init(void){
339   time_to_draw=0;
340
341   //game_s=OVER;
342     playerScore = 0; // reset score
343
344   // Version 2: add enemy initialization with close posture.
345     uint8_t i;
346     for(i=0;i<4;i++){
```

```
347       Enemy[i].x = 20*i;
348       Enemy[i].y = 10;
349       Enemy[i].image = SmallEnemyPointA[i];
350       Enemy[i].life = 1;
351   }
352
353
354   // Version 3: add player ship initialization
355   PlayerShip.x = MAX_X/2;
356   PlayerShip.y = MAX_Y-1;
357   PlayerShip.image = PlayerShip0;
358   PlayerShip.life = 1;
359
360   // Version 4: Add bullet initialization: you can choose Laser or Missile
361   Bullet.x = PlayerShip.x;
362   Bullet.y = PlayerShip.y+8;
363   Bullet.image = Laser0;
364   Bullet.life = 0;
365
366   SmallExplosion.x = Bullet.x;
367   SmallExplosion.y = Bullet.y;
368   SmallExplosion.life = 0;
369
370 }
371
372 // Update positions for all alive sprites.
373 void Move(void){
374   Sound_Init();
375   uint8_t i;
376   uint8_t num_life = 0;
```

```
  unsigned long ADC_value;
  unsigned int playershipPosition;

  // Move Bullet

  // V2: Move enemies, check life or dead: dead if right side reaches right screen border or detect a hit
  for(i=0;i<4;i++){
    if(Enemy[i].x < MAX_ENEMYX){
      if(Enemy[i].image==SmallEnemyPointA[i]){
        Enemy[i].image=SmallEnemyPointB[i];
      }
      else if(Enemy[i].image==SmallEnemyPointB[i]){
        Enemy[i].image=SmallEnemyPointA[i];
      }
      Enemy[i].x += 1;
      num_life++;
    }else{
      Enemy[i].life = 0;
    }
  }
  num_life--;

  // read ADC value for player ship
  ADC_value = ADC0_InSeq3();
  playershipPosition = ADC_value*(SCREENW-18)/4095;
  // update player ship
  PlayerShip.x = playershipPosition;
```

```
if (trigger_btn) {
    Bullet.x = PlayerShip.x + 8;
    Bullet.y = PlayerShip.y - 7;
    Bullet.life = 1;
    trigger_btn = 0;
    Sound_Shoot();
}
```

```
for (int i = 0; i < 3; i++) {
    if(Bullet.life && Enemy[i].life && Bullet.y - BULLETH <= Enemy[i].y && Bullet.x + BULLETW >= Enemy[i].x && Bullet.x + BULLETW <= Enemy[i].x + ENEMY10W && Bullet.x <= Enemy[i].x + ENEMY10W && Bullet.x >= Enemy[i].x){
      Enemy[i].life = 0;
      SmallExplosion.x = Enemy[i].x;
      SmallExplosion.y = Enemy[i].y;
      SmallExplosion.life = 1;
      Bullet.life = 0;
      playerScore++;
      num_life--;
  }
}
```

```
426 if (Bullet.life && Bullet.y > 2) {
427     Bullet.y -= 2;
428  } else if (Bullet.life) {
429     Bullet.life = 0;
430  }
431
432 if (SmallExplosion.life) {
433     counter++;
434     SmallExplosion.image = (SmallExplosion.image == SmallExplosion0) ? SmallExplosion1 : SmallExplosion0;
435
436     if (counter == 3) {
437         Sound_Explosion();
438         SmallExplosion.life = 0;
439         counter = 0;
440     }
441  }
442
443   if (num_life==0 && SmallExplosion.life == 0) {
444     game_s = OVER;
445   }
446  }
447
448  // Update the screen:
449  // clear display and update the screen with the
450  // current positions of all sprites that are alive.
451 void Draw(void){
452    static uint8_t enemy_posture = CLOSE;  // enemy start with close posture: SmallEnemyPointA
453    uint8_t i;
454
```

```
455    if (game_s==OVER) return;
456
457    Nokia5110_ClearBuffer();
458
459    // V2: Update live enemies' positions in display buffer
460    for(i=0;i<3;i++){
461      if(Enemy[i].life == ALIVE){
462        Nokia5110_PrintBMP(Enemy[i].x, Enemy[i].y, Enemy[i].image, 0);
463      }
464    }
465
466    // Update the player ship position in display buffer
467    Nokia5110_PrintBMP(PlayerShip.x, PlayerShip.y, PlayerShip.image, 0);
468
469
470    // Update the bullet position in display buffer if there is one.
471
472      if(Bullet.life){
473      Nokia5110_PrintBMP(Bullet.x, Bullet.y, Bullet.image,0);
474    }
475
476   if(SmallExplosion.life){
477      Nokia5110_PrintBMP(SmallExplosion.x, SmallExplosion.y, SmallExplosion.image,0);
478    }
479
480   Nokia5110_DisplayBuffer();      // Update the display with information in display buffer
481  }
482
```

```
482
483 // Control screen refresh rate.
484 void SysTick_Handler(void){
485   //1 sec delay
486   time_to_draw = 1;
487   delay = delay+1;
488
489 }
490
491 // Initialize the onboard two switches.
492 void Switch_Init(void){
493     SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD; // Activate D clocks
494   while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOD)==0){};
495
496   GPIO_PORTD_CR_R = 0x03; //Allow changes to PD0 and PD1
497   GPIO_PORTD_AMSEL_R &= ~0x03;      // 3) disable analog function
498   //GPIO_PORTD_PUR_R |= 0x03;      //     enable weak pull-up on PF4
499   GPIO_PORTD_PCTL_R &= ~0x000000FF; //  configure PF4,0 as GPIO
500   GPIO_PORTD_DIR_R &= ~0x03;          // 6) PD0 and PD1 input
501   GPIO_PORTD_AFSEL_R &= ~0x03;       // 7) no alternate function
502   GPIO_PORTD_DEN_R |= 0x03;          // 8) enable digital pins PD0 and PD1
503   GPIO_PORTD_IS_R &= ~0x03; // Edge Sensitive
504   GPIO_PORTD_IBE_R &= ~0x03; //Not both edges
505   GPIO_PORTD_IEV_R &= ~0x03; // Falling Edge event
506   GPIO_PORTD_ICR_R |= 0x03; //clear flag
507   GPIO_PORTD_IM_R |= 0x03;       // (f) arm interrupt on PD0 and PD1
508   NVIC_PRI7_R = (NVIC_PRI0_R&0x1FFFFFFF)|0x00400000; // priorty 4
509   NVIC_EN0_R |= 0x08;
510 }
511
512 void Delay100ms(void){unsigned long volatile time;
513   time = 727240*100/91;  // 1 sec
514   while(time){
515         time--;
516   }
517     for (time=0;time<1000;time=time+10) {
518     }
519 }
```

CODE EXPLANATION

Main Function:

Initialization:

Calls System_Init() to set up the system components.

Enters an infinite loop with while(1) for continuous execution.

Game Loop:

Displays a start prompt using Start_Prompt().

Enters a loop while the game state (game_s) is in the "OVER" state, waiting for the player to initiate the game.

Calls Game_Init() to set up the game, including sprite initialization.

Displays the game state using Draw().

Enters another loop while the game state is "ON."

If it's time to draw (time_to_draw), moves sprites using Move() and updates the display using Draw().

Resets time_to_draw after drawing.

Game Over:

Calls End_Prompt() when the game is over, displaying the final score.

System_Init Function:

System Initialization:

Disables interrupts.

Initializes system components such as PLL, SysTick, Switch, ADC, and Nokia5110.

Clears the display buffer and displays it.

Enables interrupts.

Start_Prompt Function:

Game Start Prompt:

Clears the display.

Displays a message prompting the player to start the game by pressing SW2.

Introduces a delay.

End_Prompt Function:

Game Over Prompt:

Initializes a delay counter.

Clears the display.

Displays a game over message along with the player's score.

Delays for a certain period before returning.

Game_Init Function:

Game Initialization:

Resets variables and sets up sprites for the game.

Initializes the player's score.

Initializes enemy sprites, player ship, bullet, and explosion.

Move Function:

Sprite Movement:

Initializes sound.

Moves enemies and updates their status.

Reads the ADC value for the player ship's position.

Moves the player ship based on the ADC input.

Manages bullet movement, collisions, and updates.

Manages small explosions.

Checks conditions for game state transitions.

Draw Function:

Display Update:

Sets the enemy posture.

Clears the display buffer.

Updates the display buffer with the positions of live enemies, the player ship, bullet, and explosions.

Displays the updated buffer on the Nokia5110 screen.

Conclusion

The first thing that we did when starting this project was to connect our launchpad with the LCD Screen. We followed the port connections as shown in the Lecture 7 slides for the blue version. Next, we added the two pushbuttons on the board. We implemented an 80MHz system clock using PLL. The beginning and ending (game over) screen was also coded. For version 2, we made the screen refresh at 10Hz using systick timer with interrupt enabled. The enemy aliens now move from left to right and have animation; if an enemy reaches the right edge of the screen, they die, 3 deaths and its fame over.

For version 3, we implemented the main character (spaceship) and made it move with the potentiometer. For version 4, we added the shooting/firing feature to switch 1; rising edge interrupt is used in this step. For the final version, 5, we constructed a 4-bit DAC circuit to generate the sounds for shotting and hit. We then demoed version 5. We also added more than 3 enemies for the extra credit portion and got 3 extra points.

A major difficulty was with our live demo, we had trouble explaining the line of code that deals with the bullet hitting the enemy.

After completing this project, we learned the most about SSI and the Nokia 5510 LCD Screen. The way that the sprites and sounds are generated was interesting to learn about. Everyone in our group will be taking CECS 447 and 490a (senior project); the skills we learned from this class will help us in our future courses and our future carrers in Computer Engineering!