Abdallah Elmi

Arcade Embedded System: Project Report

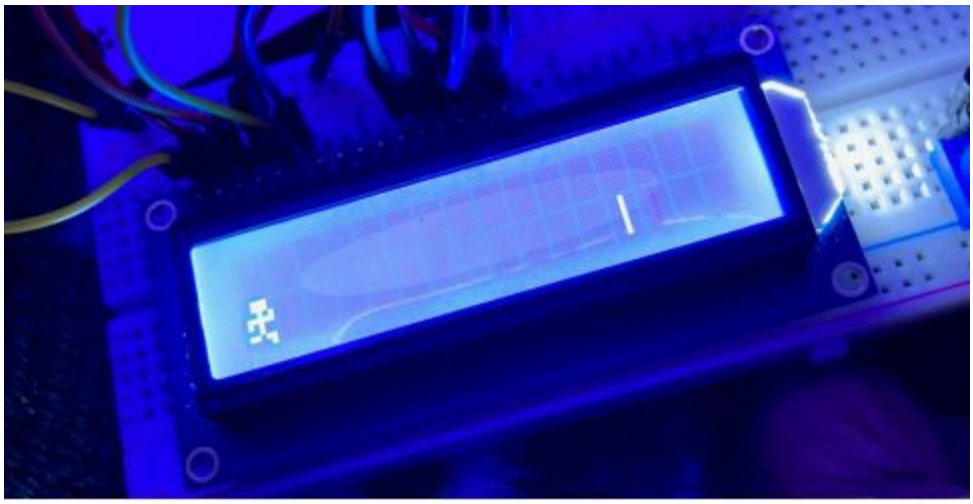
Section #1

The project that was made for this course was an embedded system. It consisted of a microcontroller which would do all the thinking of the game, a joystick and button to input controls for gameplay, a display for visuals and a buzzer for sound effects. The game is one where there is a character on the screen who must jump over obstacles that are moving towards him.

Section #2 Project Summary

The system displays the character on the screen at the start of the game and then an obstacle proceeds to appear on the screen that moves towards the character (fig 1). The character can jump using the button or move forwards or backwards using the joystick. Jumping cannot be done at the same time as other movements.

Fig 1



The joystick (fig 3) can read movements to the left and the right and to jump a separate button (fig 2) must be pressed.

Fig 2

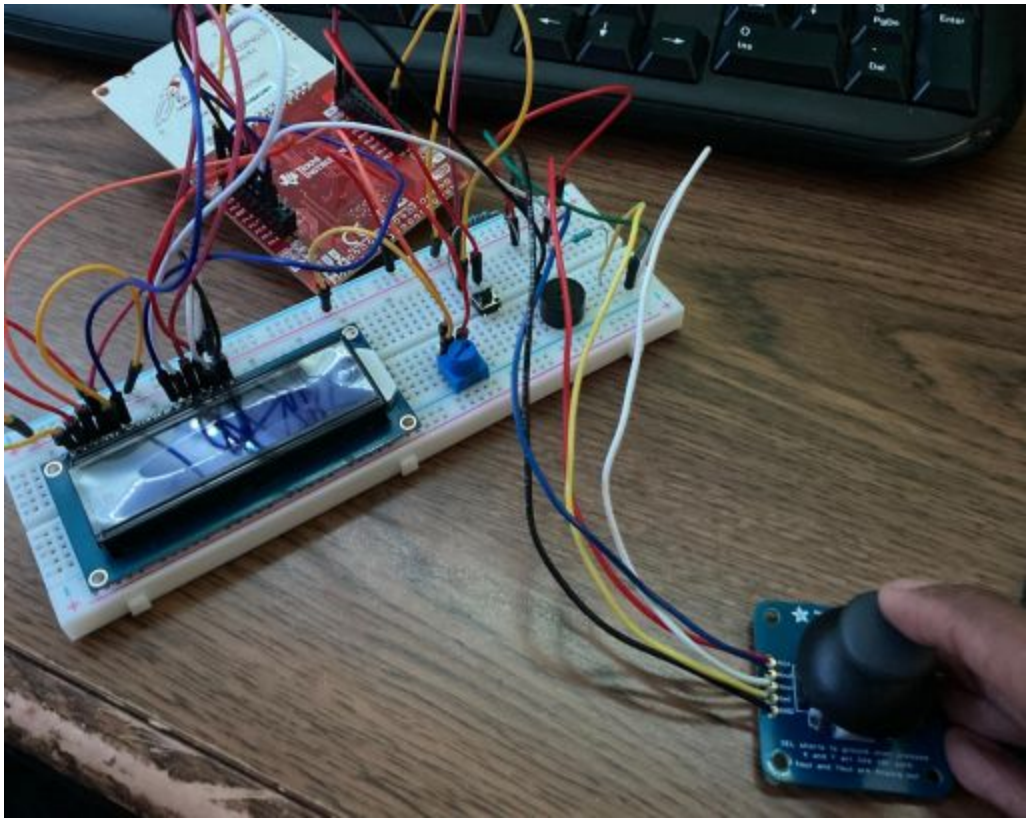
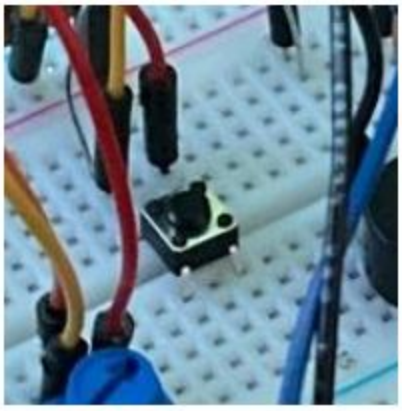
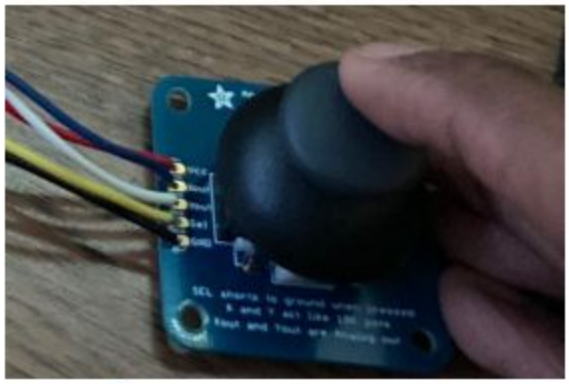


Fig 3



The button used to jump

Fig 4



The joystick used to control the characters movements

Section #3

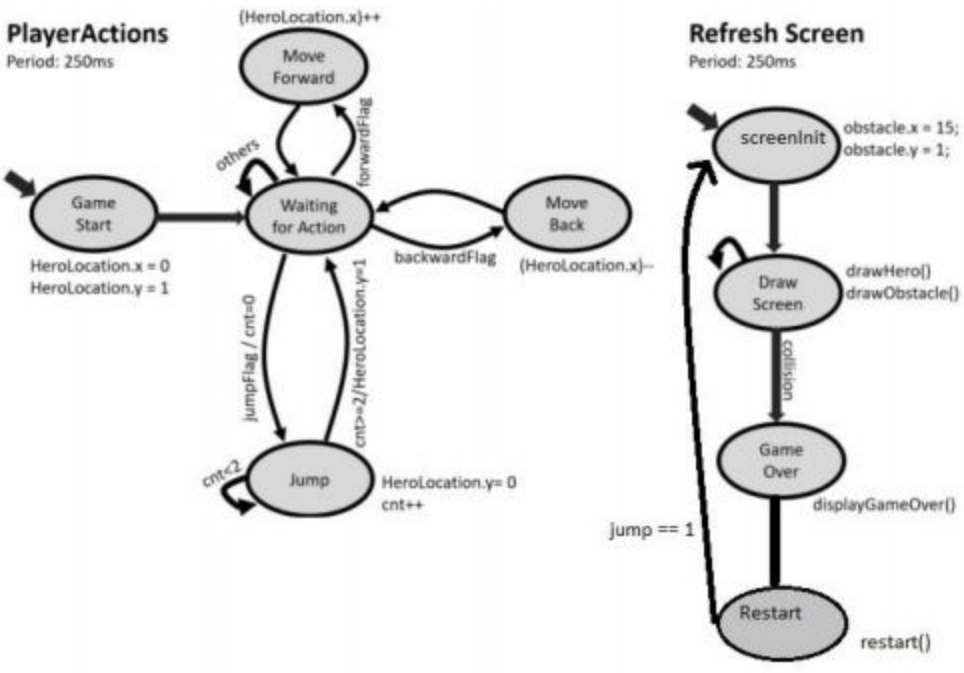
The circuit has the microcontroller as the center of all of it. The peripherals and everything else are supplied with voltage by the microcontroller and are connected to it to also give information about what the peripherals are doing. They are also connected to the microcontroller so that outputs like the lcd can also display things correctly. The way the circuit will detect moving forward or moving backward is with the analog reading from the joystick. There is an upper and lower threshold for both and when they are exceeded the her will move forward or backward accordingly. The button is used to jump and there is an interrupt system routine called playerActionTimer that sets jump to 1 when it is pressed. The buzzer is connected to the microcontroller by an analog pin and ground. The microcontroller sends the signal for the buzzer to make the right sound when the circumstances in the code are met.

Section #4 : Embed Control Design

The logic for the arcade game uses 2 state machines (fig 6). One is the one that will control the players actions based on the inputs from the game controls and the other is for the screen and what it is displaying. For the player actions there are 5 states. The first one is the Game Start state where the hero location is set for the start of the game and from there it will go to the next state which is the Waiting for Action state where the next state after this is determined based on player inputs. From here if the flags for if the player moved forward, backward or jumped are set to 1 based on the player inputs the next state will be determined accordingly. In the Move Forward and Move Backward states the hero location is either incremented to move forward or decremented to move backward and then will return to the Waiting for Action state. However, for the Jump state the hero location is moved up so that they are in the air and then this is held for however long the jump is set to last for. To make sure that this occurs there is a variable that counts for that period and the state will be held until it is met.

For the refresh screen there are 3 states but 4 with my enhancements. The first state is the screenInit state where the obstacle that moves on the screen is initially drawn on the screen at the far side of the screen. The next state is the DrawScreen state where the hero and obstacle are drawn and erased repeatedly in their new positions while the game continues. When the condition for game over is met, meaning the hero and obstacle are in the same location then the next state is the game over state. This is where the screen erases everything and displays “game over” instead. The fourth state is the Restart state and that is one of the enhancements. (state machine below)

Fig 6



Section #5

There were a couple enhancements that I had added. The first one is a title screen and count down where Hero Runner is displayed and then there is a count down from 3 down to 1 before the game begins. This was implemented by extending the delay time when the screen was set up to accommodate for the time of the countdown before the game begins. The code for displaying all of this on the lcd was also put into the screenInit state so that it would display upon game restart which is another enhancement. Another enhancement is the addition of the buzzer for sound effects. This was added into the circuit by connecting it to one of the microcontroller's analog pins and putting it in series with a resistor before going to ground. The buzzer was set to make a sound in the playerActionTimer interrupt system routine so that when a jump was detected it would buzz. The buzzer is also used in the GameOver state and blares for 5 seconds while displaying game over on the screen. The next enhancement that was added was the score which each point is determined by if the hero position is directly above the obstacle. Whenever this condition is met, the score is incremented by 1. When the game ends and goes to the game over screen there is a line underneath that says the score. To account for the limited space on the lcd being 16 when the score is put on the screen it accounts for double digit scores. The score is also set to zero in the screenInit state so that when the game is restarted the score is reset. This brings the final enhancement which is the option to restart the game. This was done by adding a fourth state in the refresh screen states. This state is gone to after being in the GameOver state for 5 seconds. In this state the screen says to press jump to restart. While this is shown the system waits for jump to be pressed. If it is pressed, then the game restarts and goes back to the screenInit state and starts from the beginning.

Appendix

ArcadeGame

```
// include the library code:
#include <LiquidCrystal.h>
#include "OneMsTaskTimer.h"

//initialize buzzer
const int buzzer = P5_2;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(P6_7, P2_3, P2_6, P2_4, P5_6, P6_6);

enum PlayerActionsStates {PlayActionInit, GameStart, WaitingForAction, MoveForward,
MoveBack, Jump};
PlayerActionsStates playerActionState;

enum ScreenStates {screenInit, DrawScreen, GameOver, Restart};
ScreenStates screenState;

//struct for heroPos and obstacle
typedef struct xy_struct{
  int x;
  int y;
} XY;

int obstaclePosition = 16;
XY heroPos;

void setupArcadeGame() {
    Serial.begin(9600);

  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  //title screen and countdown for initial game start
  delay(500);
  lcd.setCursor(0, 0);
  lcd.print("   Hero Runner  ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
  delay(1000);
  lcd.setCursor(0, 0);
  lcd.print("        3       ");
```

```
  lcd.setCursor(0, 1);
  lcd.print("                ");
  tone(buzzer, 2000, 100);
  delay(1000);
  lcd.setCursor(0, 0);
  lcd.print("        2       ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
  tone(buzzer, 1500, 100);
  delay(1000);
  lcd.setCursor(0, 0);
  lcd.print("        1       ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
   tone(buzzer, 1000, 100);
  delay(1000);
  clearScreen();

}


void loopArcadeGame() {
  delay(10);
}
```

PlayerActions

```
//Pin definitions
const int selectPin = P4_6;
const int XoutPin = P5_0;
const int buzzer1 = P5_2;

//initialize variables
int forwardX = 0;
int backwardX = 0;
int jump = 0;
int cnt = 0;
int timer = 0;

OneMsTaskTimer_t timerTask = {250,  playerActionTimerISR, 0, 0};

void setupPlayerActions(){
  Serial.begin(9600);
```

```
  playerActionState = PlayActionInit;

  //Timer Interrupt SetUp
  OneMsTaskTimer::add(&timerTask);
  OneMsTaskTimer::start();

  //Joystick setup
  pinMode(selectPin, INPUT);
}

//loop through player actions by reading the sensors and then
//going to the state machine for actual actions
void loopPlayerActions(){
  while(timer ==0) {delay(10);}
  timer = 0;
  readSensors();
  playerActionTickFunc();
  delay(10);
}

void playerActionTickFunc(){

    //State Transitions
  switch (playerActionState){
        case PlayActionInit:
        playerActionState = GameStart;
        break;

    case GameStart:
      playerActionState = WaitingForAction;
      break;

    case WaitingForAction:

      if (jump == 1){
          playerActionState = Jump;
          cnt = 0;
          jump = 0;
      }
      else if (forwardX == 1){
        playerActionState = MoveForward;
        forwardX = 0;
      }
      else if (backwardX == 1){
```

```
      playerActionState = MoveBack;
      backwardX = 0;
    }
    break;

  case MoveForward:
    playerActionState = WaitingForAction;
    break;

  case MoveBack:
    playerActionState = WaitingForAction;
    break;

  case Jump:
    //check if hero has been in air for long enough before going back to next state
    if (cnt >= 3){
        playerActionState = WaitingForAction;
        heroPos.y = 1;
    }
    //increment for jump to be held
    cnt++;
    break;

 }


//PlayActionInit, GameStart, WaitingForAction, MoveForward, MoveBack, Jump

//State Actions
switch (playerActionState){
  case GameStart:
    heroPos.x = 0;
    heroPos.y = 1;
    lcd.setCursor(0, 0);
    lcd.print("               ");
    lcd.setCursor(0, 1);
    lcd.print("               ");
    break;

  case WaitingForAction:
     break;

  case MoveForward:
      (heroPos.x)++;
```

```
        break;

    case MoveBack:
        (heroPos.x)--;
        break;

    case Jump:
        heroPos.y = 0;
        break;
  }
}

void readSensors(){
  //Read in the stick position
  int joystickXPos = analogRead(XoutPin);
  Serial.println(joystickXPos);
  //strange threshold values to account for drift
  if (joystickXPos > 977){
    forwardX = 1;
  }
  else if (joystickXPos < 965){
    backwardX = 1;
  }
}

void playerActionTimerISR(){
  int jumpVal = analogRead(selectPin);
  //detect jump
  if (jumpVal < 600){
    jump = 1;
    //make sound when character jumps
    tone(buzzer1, 1200, 100);
  }
  timer = 1;
}
```

RefreshScreen

```
#include <String.h>

XY oldHeroPos;
XY obstaclePos;
```

```
//declare variables
int screenTimerFlag = 0;
int score = 0;

byte hero[8] = {
  B01110,
  B01110,
  B00101,
  B11111,
  B10100,
  B00100,
  B11011,
  B00001,
};

OneMsTaskTimer_t screenTimerTask = {250, screenTimerISR, 0, 0};

void setupRefreshScreen(){
  Serial.begin(9600);
  delay(4800);

  // Set your screen state to its initial state
  screenState = screenInit;

  //Timer Interrupt SetUp
  OneMsTaskTimer::add(&screenTimerTask);
  OneMsTaskTimer::start();
  lcd.createChar(0, hero); // set up hero

}

void loopRefreshScreen(){

  while(screenTimerFlag == 0){
    delay(10);
  }
  screenTimerFlag = 0;
  refreshScreen();
  delay(10);
}

void drawHero(){
  lcd.setCursor(oldHeroPos.x, oldHeroPos.y);
```

```
  lcd.print(" ");
  oldHeroPos = heroPos;
  lcd.setCursor(heroPos.x, heroPos.y);
  lcd.write(byte(0));
}

void eraseHero(){
  lcd.setCursor(oldHeroPos.x, oldHeroPos.y);
  lcd.print(" ");
}

void drawObstacle(){
  lcd.setCursor(obstaclePos.x, obstaclePos.y);
  lcd.print(" ");

  (obstaclePos.x)--;
  if (obstaclePos.x < 0){
    obstaclePos.x = 16;
    //obstaclePos.x = rand () % 17;
    obstaclePos.y = 1;
    //obstaclePos.y = rand() % 2;
  }
  lcd.setCursor(obstaclePos.x, obstaclePos.y);
  lcd.print("|");
}

void refreshScreen(){

  //state transitions
  switch(screenState){
    case screenInit:
      screenState = DrawScreen;
      break;

    case DrawScreen:
      if(obstaclePos.x == heroPos.x && obstaclePos.y == heroPos.y){
        screenState = GameOver;
      }

      else if (obstaclePos.x == heroPos.x && obstaclePos.y != heroPos.y){
        score++;
      }
      break;
```

```
    case GameOver:
      screenState = Restart;
      break;


    case Restart:
      if (jump == 1){
        clearScreen();
        screenState = screenInit;
      }
      break;
  }

//actual states
  switch(screenState){
    case screenInit:
    //set initial screen
      score = 0;
      heroPos.x = 0;
      heroPos.y = 1;
      obstaclePos.x = 15;
      obstaclePos.y = 1;

      //title screen for when game is restart
      lcd.setCursor(0, 0);
      lcd.print("   Hero Runner  ");
      lcd.setCursor(0, 1);
      lcd.print("                ");
      delay(1000);
      lcd.setCursor(0, 0);
      lcd.print("        3       ");
      lcd.setCursor(0, 1);
      lcd.print("                ");
      tone(buzzer, 2000, 100);
      delay(1000);
      lcd.setCursor(0, 0);
      lcd.print("        2       ");
      lcd.setCursor(0, 1);
      lcd.print("                ");
      tone(buzzer, 1500, 100);
      delay(1000);
      lcd.setCursor(0, 0);
      lcd.print("        1       ");
      lcd.setCursor(0, 1);
```

```
      lcd.print("                ");
      tone(buzzer, 1000, 100);
      delay(1000);
      clearScreen();
      break;

    case DrawScreen:
      drawHero();
      drawObstacle();
      break;

    case GameOver:
      gameOver();
      break;

    case Restart:
      restart();
      break;
  }
}

void gameOver(){
  lcd.setCursor(0, 0);
  lcd.print("   Game Over !  ");
  lcd.setCursor(0, 1);
  //two different lcd.prints to account for single or double variable scores
  if(score < 10){
      lcd.print("    Score: ");
      lcd.print(score);
      lcd.print("    ");
  }
  else{
      lcd.print("   Score: ");
      lcd.print(score);
      lcd.print("    ");
  }

  //buzzer when in gameover
  for(int i = 0; i < 10; i++){
  tone(buzzer, 300, 500);
  delay(500);
  }
}
```

```
//displays restart prompt
void restart(){
  lcd.setCursor(0, 0);
  lcd.print("   PRESS JUMP   ");
  lcd.setCursor(0, 1);
  lcd.print("   TO RESTART   ");
}

//clears the screen to make it blank
void clearScreen(){
  lcd.setCursor(0, 0);
  lcd.print("                ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
}

void screenTimerISR(){
  screenTimerFlag = 1;
}
```