Project Technical Report

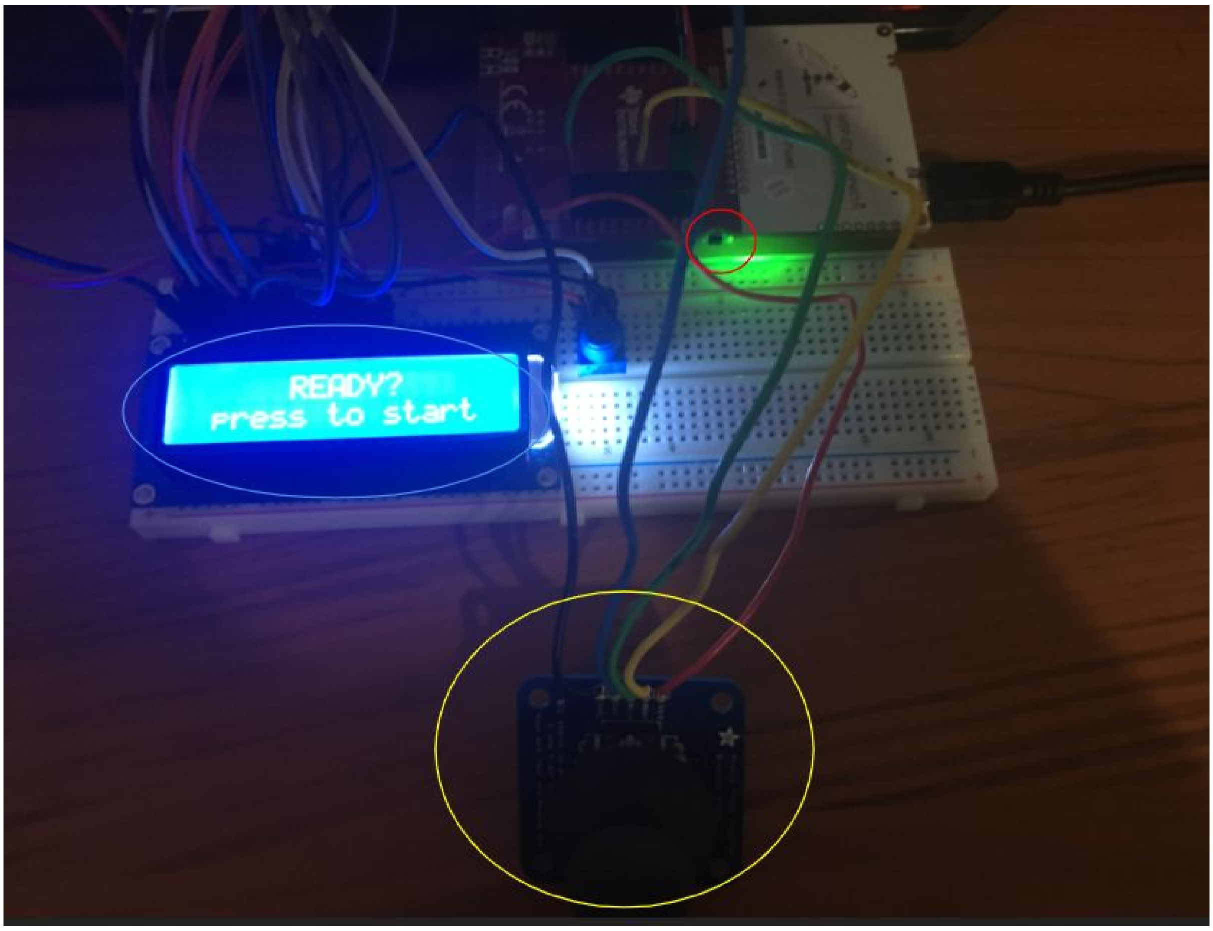
Poject Summary:

The system is an embedded gaming console. The console functions similar to an arcade device in which only power is needed for the system to function, and the game is built into to the processing chip of the system. The player will interact with a joystick and two bottoms during gameplay.

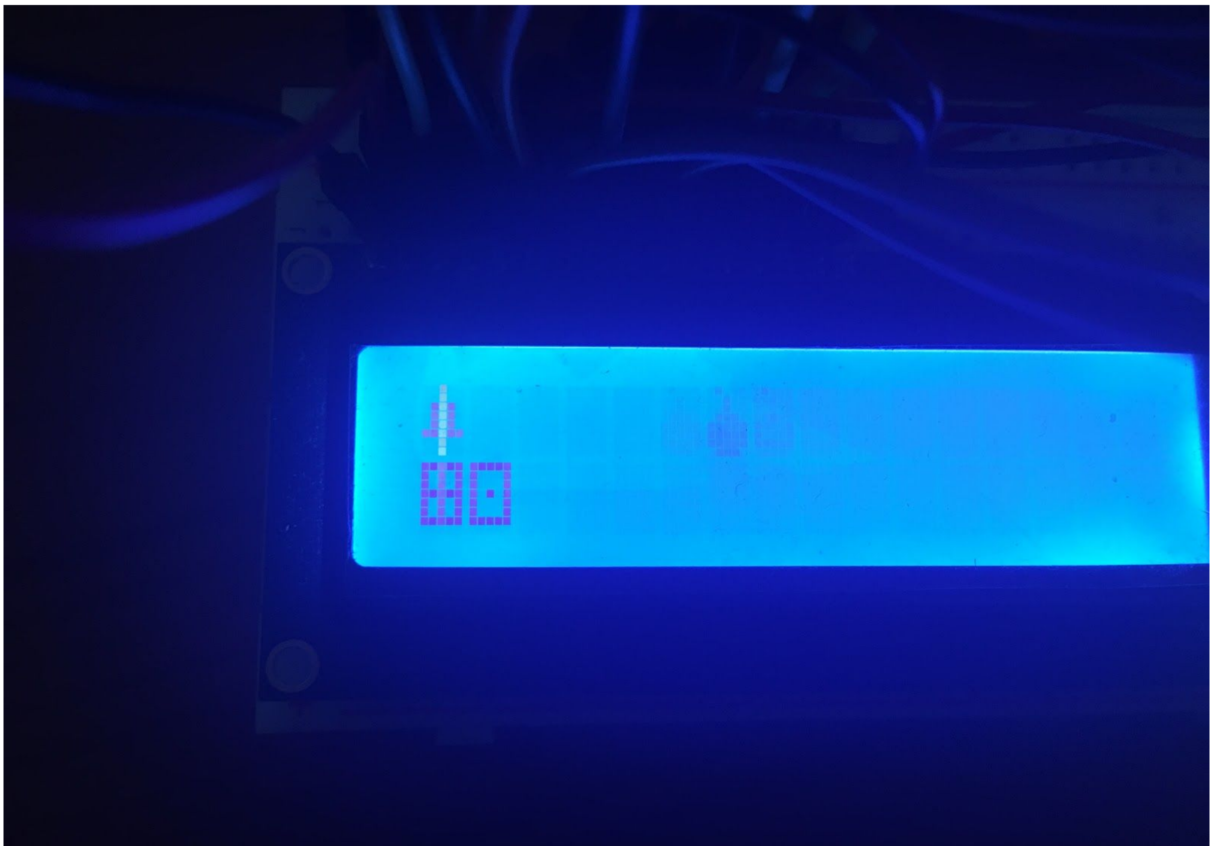
System Functionality:

The system presents a side scroller game in which the player character (the sword) has to survive waves of incoming obstacles (the shields). There are two types of obstacles, type 0 (the shield with a dot in the middle) moves leftwards one block at a time where type 1 (the shield with a cross in the middle) moves leftwards two blocks at a time. When the system is powered, it will start in the starting screen, in which the player will be prompted to press the PUSH1 to start the game. The player character initially start at the left of the screen and the obstacles come out from the right. The movement of the player character is controlled by the attached joystick, where moving the controller up and down will cause the player character to switch rows (the y-axis) appropriate to the input and moving the joystick left and right will case the player character to switch columns (the x-axis) appropriate to the input. Scores increase when an obstacle passes the player character without triggering an game ending event, or when the player character collides with a bonus object. As the score increases, the player will be presented with faster and more frequent obstacles, up to a point where progression will be near impossible. A bonus object is an object that spawns periodically, in the shape of a round fruit. When the bonus object is collected (collides with the player character), in addition to gaining points, the player also gains the ability to fire a projectile (a slash) by pressing PUSH2 (such ability does stack), that travels from the position for the player character all the way towards the right of the screen. The projectile will delete all obstacles that it comes in contact with. A game ending event is triggered when an obstacle comes into contact with the player characters. When this particular event happens, a game over screen will appear on the screen displaying the character “YOU DIED” along with the amount of points the player has gained during gameplay. In which the player can press PUSH1 to go back to the starting screen.

The game starting screen, showing PUSH1 (red), the screen (blue) and the joystick (yellow)



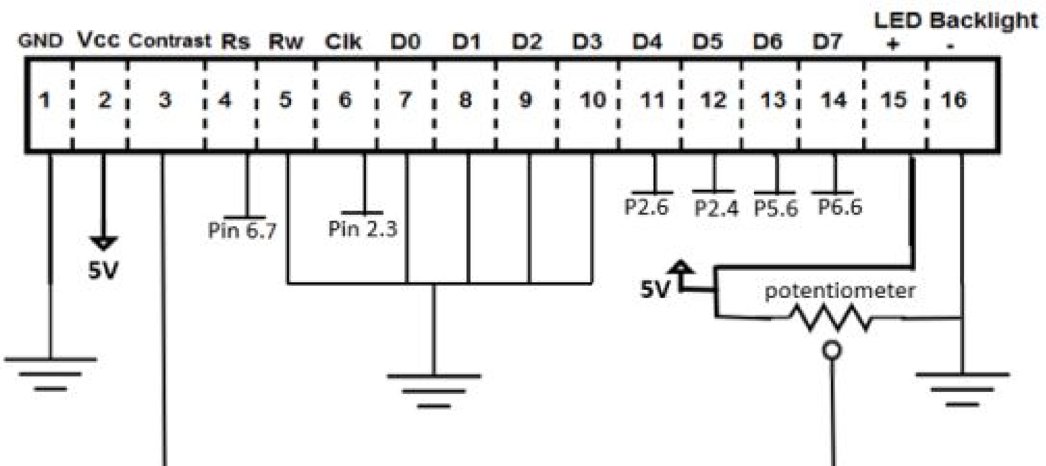
During game play. Showing player character (top left), type 1 obstacle (bot left), type 0 obstacle (bot right) and the bonus object (top right).



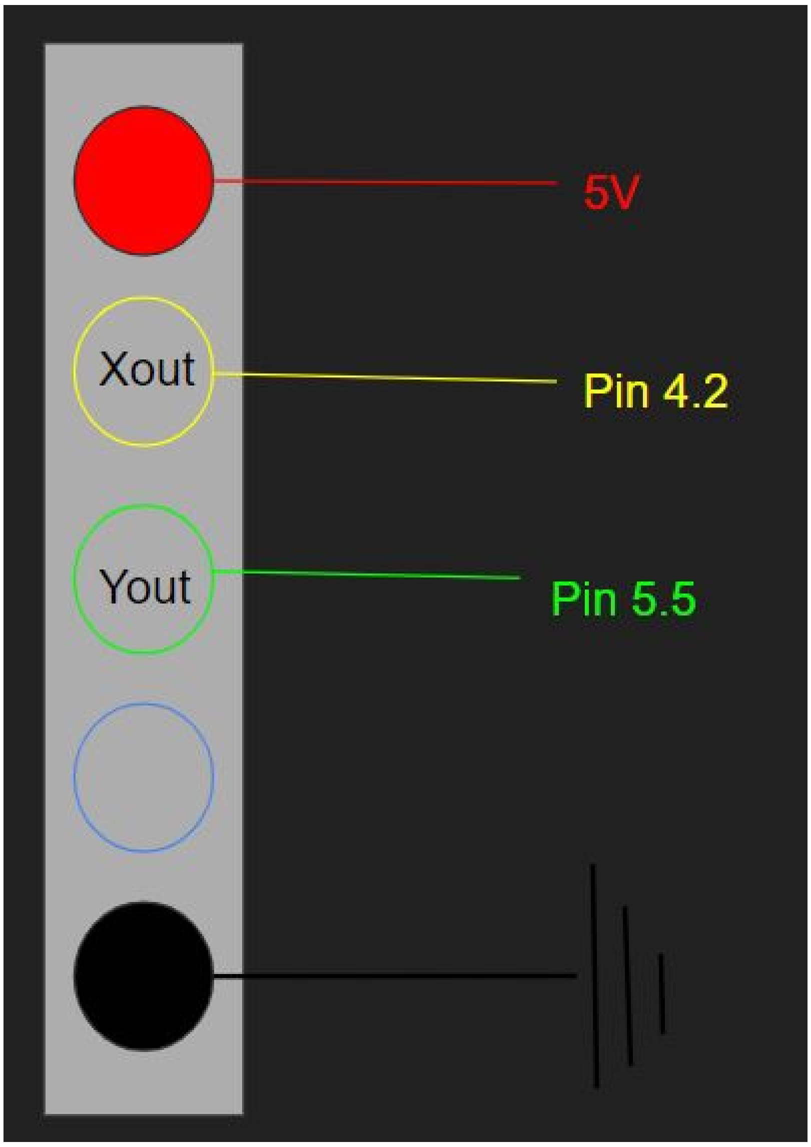
Game over screen, showing "YOU DIED" (top) and the pointe gained during gameplay (bot)

Circuit Design:

Below is the circuit diagram for the LCD screen. D4-D7 are used by the microcontroller to spend information on what to display the screen. The data a being received on each clock pulse by the controller (pin 2.3). Rs is used by the LCD screen to determine what mode the received data should be, either in instruction or character. Example of instruction mode data can be "move cursor" while example of character mode data can be "0" in which the LCD will print a "0" to the cursor location. The potentiometer is used to adjust the contrast of the screen.

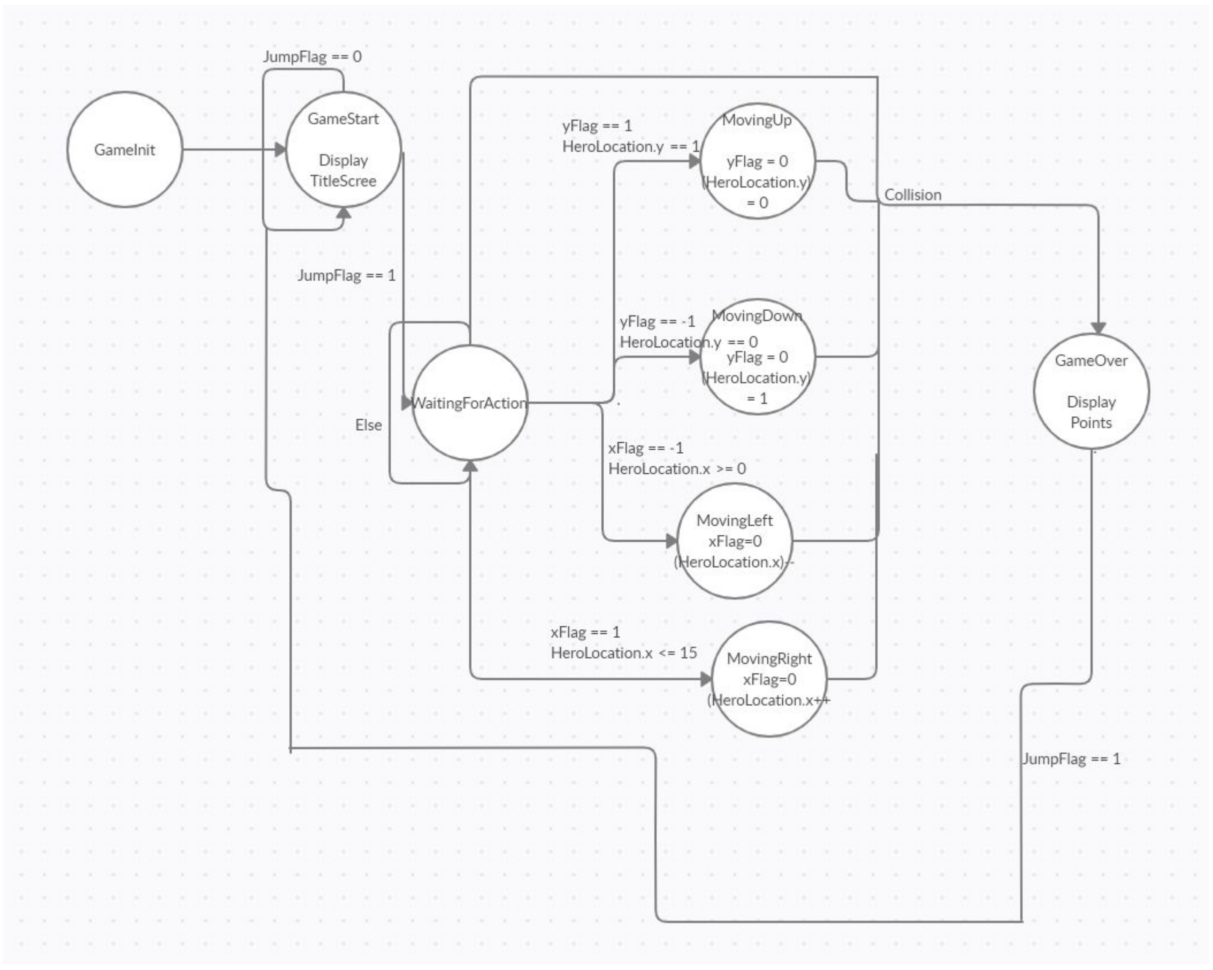


Below is the connection diagram for the joystick. Both Xout and Yout needed to be connected with pins that has analog to digital conversion, since the input is non-binary. The blue pin is the select pin, which is not used in the system due to its functionality being replaced by the buttons built-in on the board. The two buttons (PUSH and PUSH2) will trigger an interrupt service routine when pressed since the two signals should be processed with priority. An interrupt service routine is when the system pauses all current processes to execute the code within the interrupt service routine.



Embedded Control Design:

The following is the state machine used to control the system. Several flags were used to help the system to decide which action it shall take next. When PUSH1 is pressed, jumpFlag will be set to one. Allowing the game to start or restart (depending on current state). The jumpFlag is constantly being reset to avoid the chance that PUSH1 is being pressed outside of its intended states. X and Y flags are set using the X and Y analog inputs of the joystick. When the user moves the joystick in a certain direction pass a threshold, the respective flags will be set in order for the screen accurately reflect user inputs. The joystick inputs are only being read if the game has started, hence the necessity of the waitingForAction state. It also helps eliminate the “busy waiting” problem, in which the processor constantly checks for user input despite there might be none. During each movement states, after the position of the player character has been changed appropriately, the respective flag needs to be reset inorder to prevent the player character from moving in one direction indefinitely when no input has been read.


JumpFlag == 0
GameInit
GameStart Display TitleScree
JumpFlag == 1
Else
WaitingForAction
yFlag == 1 HeroLocation.y == 1
MovingUp yFlag = 0 (HeroLocation.y) = 0
Collision
yFlag == -1 HeroLocation.y == 0
MovingDown yFlag = 0 (HeroLocation.y) = 1
xFlag == -1 HeroLocation.x >= 0
MovingLeft xFlag=0 (HeroLocation.x)--
xFlag == 1 HeroLocation.x <= 15
MovingRight xFlag=0 (HeroLocation.x++
GameOver Display Points
JumpFlag == 1

Enhancements:

Multiple obstacles are implemented by initializing a new struct, containing information such as the position of the obstacle, if the obstacle is active, what type of obstacle is it and if it has been checked for scores or not (this exact struct is also used to implement the bonus and the projectile, more on those later). Then an array of obstacles can be initiated with the size of an array being the absolute maximum amount of obstacles on the screen. When the game starts, numerous obstacles will be activated over time, keeping track of the amount of active obstacles. The game will stop making obstacles active after the current active obstacles equals with the current maximum obstacles. After that, the code checks for all active obstacles and increments their position appropriately, and deletes them when they either gone out of the left of the screen, or comes in contact with a projectile. The current maximum obstacles will be incremented over time as score progresses, up until it has the same value as the absolute maximum amount of obstacles.

The current maximum amount of obstacles and the speed of the obstacles are incremented under the same logic: two integer are initialized, one incrementer and one threshold. The incrementer gets incremented when the player gets a score, and when the incrementer gets pass the threshold, the current max and the speed gets incremented. After that, the incrementer gets reset and the threshold is also reset to a random number, so that the time between each time the game gets harder is different.

The bonus object is implemented using rather similar logic as above. On each cycle (certain amount of time), a variable is being incremented a random amount, and when the variable surpasses a certain fixed threshold, a bonus object is being created, resetting the incrementer. Along side that, on every clock cycle, the code checks for active bonus objects and moves it across the screen, and deleting it when it either exits the left side of the screen, or comes in constant with the player, in which case two points will be granted along with the ability to fire the projectile.

The projectile is implemented by having an interrupt reading the PUSH2 input, and setting a flag when it is pressed. Then on each clock cycle, the code checks for if the button is pressed, and if the player has resources available to fire the projectile. If both are true, a projectile is created in front of the player character. Along side with detecting button presses and creating the projectile, the code also looks for active projecties (max 1) and moves it towards the right of the screen.

Appendix:

All the code for the project are attached for detail