# ENEL 387 Project Final Report
# Apr. 3, 2017

**Daniel Shevtsov**
**SID: 200351253**

# Introduction

This project implements a device capable of running multiple basic video games. At the core of the device is the STM32F100RB micro-controller, and the device comes with a basic controller with an analog stick, two buttons, a menu button, and a buzzer for audio cues. Games run on a 4x16 text display, with games also making use of four LEDs to display the game status to the player. When the system starts, the user is greeted with a menu to pick between available games, and at any point during a game, the user may choose to return to the menu and select a different game to play.

# Hardware Description

**Block Diagram:**


**System Inputs:**

The following is a list of inputs available to users of the system, and their purpose:

- Analog Joystick
  - The analog joystick provides movement and selection capabilities. With it, the user may navigate through the interface and select games. Different games can use the joystick differently (e.g. character movement, option selection)
- Button A and Button B
  - Buttons A and B are general purpose user buttons. In general, A is used to ‘accept’ an action, and B is used to ‘cancel’ an action. Within a game, these buttons may be bound to different operations depending on game controls.
- Menu Button
  - The menu button serves to allow the user to return to the main menu and select games. Returning to the menu can be done from any game, at any point in the game.

**System Outputs:**

The following is a list of outputs available to the system, and their purpose:

- LCD Display
  - The 4x16 text display is the main output of the system. It is the primary device that users use to interact with the menu and various video games. Different video games can use the device differently, and each may have its own user interface.

- 4 LEDs
    - The 4 LEDs complement the LCD display by providing additional information to the user during gameplay. Different games may implement usage of these LEDs differently, with features such as displaying remaining player health, or flashing when a round ends.
- Buzzer
    - The buzzer is a form of basic audio output from the system. It provides the user with different cues depending on the game played. For example, it may alert the user when they take damage, or when a round completes, accompanied by visual cues from the LEDs and LCD display.

**Electrical Schematic:**

Please see file `doc/schematic.pdf` for the full electrical schematic of the device.

**Datasheets:**

Please see files under `doc/datasheets` for vendor datasheets of components used in this device. See `doc/functional_specification.docx` for more information on this device.

Vendor datasheet description:

- `doc/datasheets/4x16_lcd.pdf`
    - Datasheet for NHD-0416BZ-FL-GBW 4x16 LCD Module
- `doc/datasheets/piezoelectronic_buzzer.pdf`
    - Datasheet for TDK PS1240P02BT Piezoelectronic Buzzer
- `doc/datasheets/pushbutton.pdf`
    - Datasheet for E-SWITCH KS-01Q-01 Pushbutton
- `doc/datasheets/thumbstick.pdf`
    - Datasheet for Adafruit MINI 2-AXIS ANALOG THUMBSTICK

# Software Description

**Program Listing:**

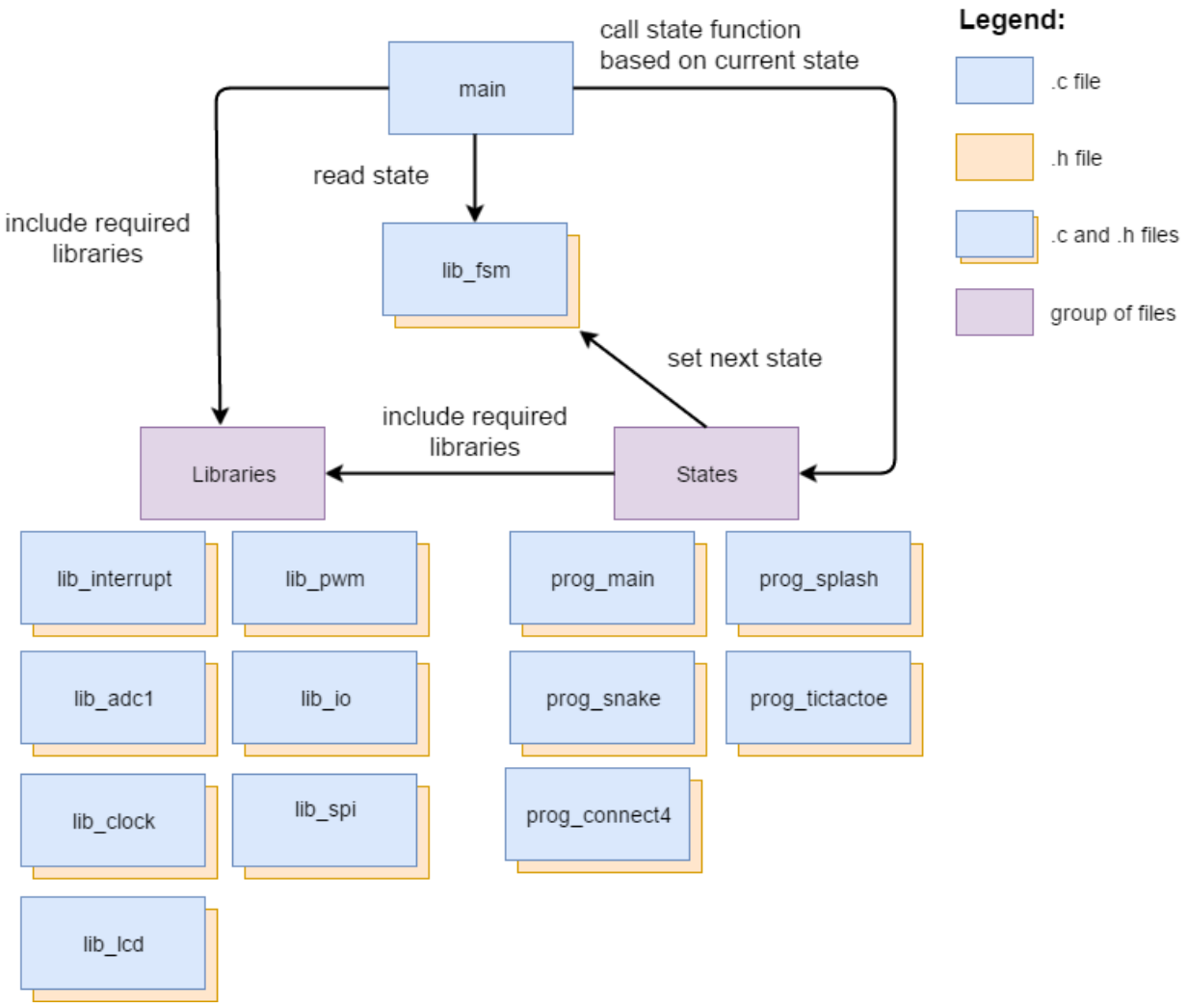
Please see directory `src/` for a complete listing of the software program written in the C language for this project.

**System Functions:**

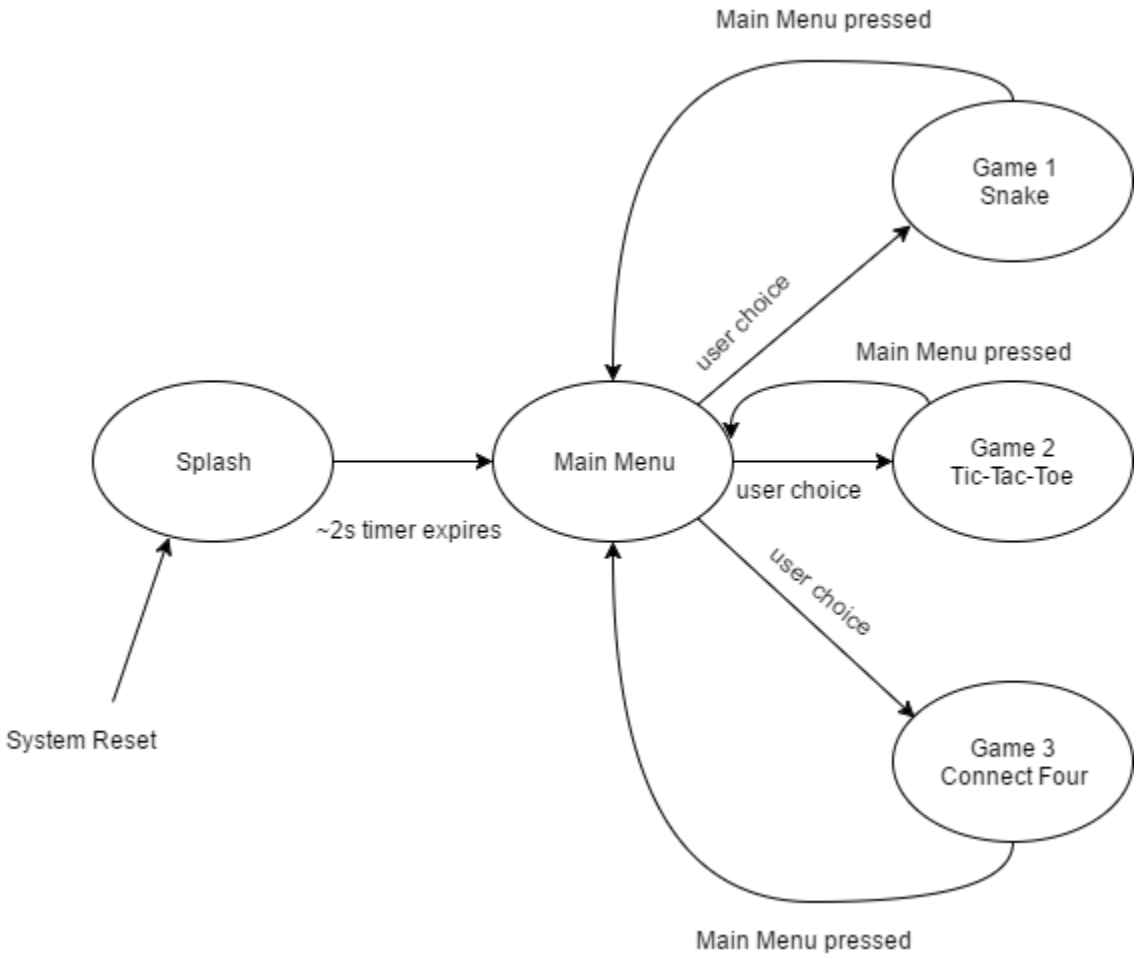
- **The system displays a splash screen during boot**
  - A splash screen identifying the system's name, author, and version will be displayed for several seconds to the user prior to the main menu. This screen is only displayed during the initial boot, and not when switching games.
- **The system boots to the Main Menu**
  - The entry state of the system after the splash screen is always the main menu, from which the user can begin interacting with the system.
- **The user may select one of three games in the menu, with navigation controls provided on the screen**
  - A list of games and controls are shown on the screen, allowing the user to select a game from the list of games that the system supports.
- **Game 1 – Snake clone**
  - The user may play a clone of the game snake, in which the analog stick allows for the user to move the head of the snake, and the goal is to avoid obstacles while collecting tokens. Once a certain number of tokens is collected, or the player hits an obstacle or the edge of the screen, the game ends, at which point the user is given the option to begin a new round of the game.
- **Game 2 – Tic-Tac-Toe**
  - The user may play a game of tic-tac-toe. The user plays vs. an AI with a randomly selected turn order. Standard rules of tic-tac-toe apply, and once a game is deemed finished, the user is given the option to begin a new round of the game. The analog stick in this game allows the user to select which cell to place their icon in.

- **Game 3 – Connect Four**
    - The user may play a game of connect four. The user plays vs. an AI with a randomly selected turn order. Standard rules of connect four apply, and once a game is deemed finished, the user is given the option to begin a new round of the game. The analog stick in this game allows the user to select which cell to place their icon in.
- **At any point, the user can press the Main Menu button to return to the main menu screen.**
    - The system allows the user to preemptively exit any game and return to the main menu to select another game. This can be done from any game, during any time in a game.

## Software Design:



## System State Diagram:




## System States Table:

The table below maps states to the indices that they are represented by in the program:

| INDEX | STATE | IMPLEMENTED IN |
|---|---|---|
| 0 | Splash | prog_splash.c |
| 1 | Main Menu | prog_main.c |
| 2 | Game 1 – Snake | prog_snake.c |
| 3 | Game 2 – Tic-Tac-Toe | prog_tictactoe.c |
| 4 | Game 3 – Connect Four | prog_connect4.c |

**Testing Documentation:**

**[No date (Feb – Mar)]**

Confirmed working operation of microcontroller, LEDs, and LCD display during ENEL 387 labs.

**[Mar. 11, 2017]**

Started tracking testing procedure. Laid out components on breadboard and wired the thumbstick and the buzzer to interface with the microcontroller.

**[Mar. 22, 2017]**

Struggles with configuring buzzer. Datasheet lists 70dB/10cm @ 3V rectangular wave, 4KHz frequency, but the device was very quiet in this configuration.

Created a small program to iterate through a range of frequencies for a square wave using PWM @ 50% duty cycle to observe the maximum volume that could be produced by the buzzer.

Created a simple program to test the operation of the buzzer, thumbstick, and SysTick timer interrupt. Thumbstick tested to be working successfully, outputting a value between 0x000 and 0xFFF in the X and in the Y directions. Used SysTick to successfully create alarm-like beeping with the buzzer, although the buzzer was still too quiet.

**[Mar. 23, 2017]**

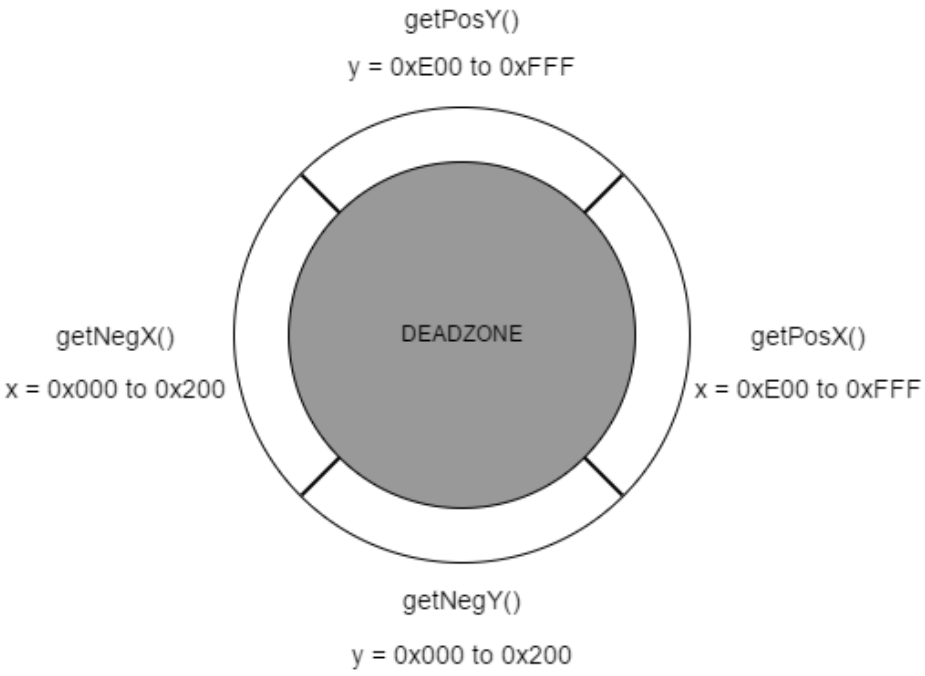
Wired the three buttons to the microcontroller following the electrical schematic and wrote a small test program to ascertain their correct operation. Button B worked as intended, but buttons A and MENU were stuck at ON state.

**[Mar. 24, 2017]**

Tested same program again, this time all of the buttons worked correctly. This might be an issue with some contacts, but I could not replicate it in my testing.

**[Mar. 26, 2017]**

Created a driver for the thumbstick to allow detection of 4 directions (positive and negative x and y axes) and tested with a simple program that turns on an LED corresponding to a direction of the thumbstick. See diagram below for a graphical representation of the driver's operation.




Created the basic skeleton for the state machine based on the state diagram and table. Each state prints out its own name to the LCD display to indicate which state is currently active.

Created the interrupt handler for PD2 (Menu Button) as EXTI0_IRQHandler to always change state to main menu whenever menu button is pressed (except when the state is splash screen or main menu).

**[Mar. 28, 2017]**

Completed splash screen state and menu state. Confirmed that splash screen enters main menu state and that main menu state can go to all the game states. Also confirmed that the menu button interrupt returns from any game to main menu.

Designed the state machine so that each state can behave as its own 'program' or sub-program of the entire program. Each state corresponds to a C file that can have its own private variables and methods, but each one has some public methods that are called by the state machine depending on the state. At this stage the complete state diagram has been implemented in the program.

Buzzer stopped working again, even though code involving it did not change. The buzzer would output a very quiet sound before, but now it does not.

Started programming Game 1 – Snake. Creating a simplified version of the game where the player must simply collect 4 tokens to win, and hitting the edges of the screen is the losing condition. The snake does not currently grow with the

number of tokens, but this could be extended. The snake is represented by ‘O’ and the tokens by ‘*’. The grid is 4x16, i.e. the size of the LCD’s character grid. A concept of the game is shown in the figure below:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | O | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | * | | | |
| 2 | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | |

Testing revealed that the microcontroller apparently cannot use the C random number generator with a random seed. Using a constant seed instead. This makes games feel less ‘random’ unfortunately.

Implemented most of the game logic of the snake game state. During testing found bugs in converting 2D array to 4 1D arrays to print to LCD.

**[Mar. 29, 2017]**

Fixed bug with 2D -> 1D array conversion – needed a null terminator character ‘\0’ for the 1D arrays since they are arrays of characters and the println() functions output character arrays until they reach the null terminator.

Completed snake game state. Tested that the game ends when player reaches any boundary (either an edge or a corner of the grid). Tested that the game ends with victory when the player collects 4 tokens.

Started programming Game 2 – Tic-Tac-Toe. Creating a version of the game in which the player selects X (first) or O (second), then plays vs. the computer (the computer simply places their symbol in a random free location). The game ends according to standard tic-tac-toe rules (player fills a row, column, or diagonal with their symbol). A concept of the game is shown in the figure below:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | X | | |
| 1 | | O | X |
| 2 | | | |

Completed most of this game state, other than the process of checking the board to find a winner.

**[Mar. 30, 2017]**

Completed tic-tac-toe game state. Played a few times to test whether the game resolves correctly in different situations (i.e. full row, full column, forward and back diagonals, all 9 cells filled and there was a winner, all 9 cells filled and there was not a winner).

Started programming Game 3 – Connect Four. Creating a simple version of the game in which a player must stack 4 of their icons either in a row, column or diagonally, similar to tic-tac-toe. Unlike tic-tac-toe, the player selects the column to drop their icon down to from the top of the column, so the player can move their selection cursor from right to left only. There is an indicator at the top right corner to show if it is currently the player or the AI's turn to play. A concept of the game is shown in the figure below (area shaded blue represents cells that the user can select from):

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | A | |
| 2 | | B | B | |
| 3 | A | A | B | A |

Completed connect four game state. This state was simple to implement as I made the tic-tac-toe game state generic and easy to modify. Much of the logic between these two games is similar (connect n icons, n by n board, turn based) so this game could be easily implemented by using the code from that state, changing some constants and changing the functionality of user inputs (rather than placing an icon directly in a desired cell, icons must be stacked from bottom to top).

Tested whether the game resolves correctly in different situations (i.e. full row, full column, stacked forward or backwards diagonal, draw).

**[Apr. 4, 2017]**

Finally resolved issue with buzzer not turning on. Measured pin PA8 with a multimeter and noticed that there was no voltage at that pin, even though it was supposedly configured to output a square wave. Found an error in the PWM driver for the buzzer. The buzzer works now, but is still much quieter than would be preferred.

# Conclusion

**Features Checklist:**

| FEATURE | COMPLETED |
|---|---|
| Thumbstick wired + implemented driver | ✓ |
| Buzzer driver wired + implemented driver | ✓ |
| Buttons wired + implemented driver | ✓ |
| LCD wired + implemented driver | ✓ |
| The system displays a splash screen during boot | ✓ |
| The system boots to the Main Menu | ✓ |
| The user may select one of three games in the menu, with navigation controls provided on the screen | ✓ |
| Game 1 – Snake clone | ✓ |
| Game 2 – Tic-Tac-Toe | ✓ |
| Game 3 – Connect Four | ✓ |
| At any point, the user can press the Main Menu button to return to the main menu screen. | ✓ |

**Unresolved Issues:**

- Buzzer volume is too quiet
- The framerate of some games feels slow and there is some input delay. This can be attributed to everything in the code running on one thread and the usage of a delay function in some other functions such as sending commands to the LCD contribute to decreasing the responsiveness of input. Sometimes if a button is clicked and released quickly, it might not be registered since the processor is in a delay method. This could have been solved by using interrupts for the inputs, but this could complicate the program as different states call different code in response to user input.