

Software Design Document for Frogger Game in C

University at Buffalo, The State University at New York

EE379 Spring 2019

The information enclosed in this document and the source code generated for this laboratory assignment were generated by Siddhant Khera and Jordan Dycha.


| Written By | Siddhant Khera | | Revision History | | | |
|--|----------------|--|------------------|---------|------------------|-----------------------------------|
| Engineer | Siddhant Khera | | Rev | Date | Session | Approved/Grade |
| Engineer | Jordan Dycha | | Orig | 5/1/19 | Tuesday, 10AM | |
| Teaching Assistant | | | Orig.1 | 5/14/19 | Tuesday, 10AM | |
| | | | | | | |
|  University at Buffalo <i>The State University of New York</i> | | | | | | EE379S19 FROGGER |

Table of Contents

| | |
|---|-----------|
| Table of Contents | 2 |
| 1. Introduction | 3 |
| 1.1. Overview | 3 |
| 1.2. Document Scope | 4 |
| 1.3. Intended Audience | 4 |
| 2. Software Design | 5 |
| 2.1. Embedded System Overview | 5 |
| 2.2. Application Programming Interface | 6 |
| 2.2.1. Function void frog(xx,yy) | 6 |
| 2.2.2. Function void drawSquare(cx,cy,width,height,color) | 7 |
| 2.2.3. Function void display_lives(lives) | 7 |
| 2.2.4. Function void basic() | 8 |
| 2.2.5. Function void collision() | 8 |
| 2.2.6. Function car1(int* rx) | 8 |
| 2.2.7. Function void truck2(int* rx) | 9 |
| 2.2.8. Function car3(int* rx) | 9 |
| 2.2.9. Function truck4(int* rx) | 9 |
| 2.2.10. Function car5(int* rx) | 10 |
| 2.2.11. Function void pad6(int* rx) | 10 |
| 2.2.12. Function void log7(int* rx) | 10 |
| 2.2.13. Function pad8(int* rx) | 11 |
| 2.2.14. Function void log9(int* rx) | 11 |
| 2.2.15. Function void pad10(int* rx) | 11 |
| 2.2.16. Function void frog_up() | 12 |
| 2.2.17. Function void frog_down() | 12 |
| 2.2.18. Function void frog_left() | 12 |
| 2.2.19. Function void frog_right() | 13 |
| 2.2.20. Function int main() | 13 |
| 3. Testing and Verification | 13 |
| 3.1. Objective Verification | 13 |
| 3.2. Function Verification | 14 |
| 3.2.1. Function int main() | 14 |
| 3.2.2. Function void drawSquare(cx, cy, width, height, color) | 14 |
| 3.2.3. Function void frog(xx,yy) | 14 |
| 3.2.4. Function void display_lives(lives) | 14 |
| 3.2.5. Function void basic() | 15 |
| 3.2.6. Function void collision() | 15 |
| 3.2.7. Function void car1(int* rx) | 15 |
| 3.2.8. Function void truck2(int* rx) | 15 |
| 3.2.9. Function void car3(int* rx) | 15 |
| 3.2.10. Function void truck4(int* rx) | 15 |
| 3.2.11. Function void car5(int* rx) | 16 |
| 3.2.12. Function void pad6(int* rx) | 16 |
| 3.2.13. Function void log7(int* rx) | 16 |
| 3.2.14. Function void pad8(int* rx) | 16 |

| | |
|-------------------------------------|----|
| 3.2.15.Function void log9(int* rx) | 16 |
| 3.2.16.Function void pad10(int* rx) | 16 |
| 3.2.17.Function void frog_up() | 17 |
| 3.2.18.Function void frog_down() | 17 |
| 3.2.19.Function void frog_left() | 17 |
| 3.2.20.Function void frog_right() | 17 |
| Glossary | 17 |
| List of Abbreviations | 17 |
| Hardware References | 17 |

1. Introduction

1.1. Overview

The primary objective of this project was to design a Frogger game using C programming language. To accomplish that the basic requirement was to use LandTiger LPC1768 board and Keil Vision Software Suite. Frogger is an arcade game developed in the 1980s and we were expected to replicate the game using C programming language. The basic requirements for this project wanted that the frog dies when it hits the cars in the lower half of the screen or when the frog hits the water or hops off the screen, and can successfully navigate home. Once the frog has navigated home five times, the game should display a text "YOU WON!" The frog started with four lives and if the frog died four times the game should say "LOSER!"

The primary procedure to accomplish all these objectives was to divide and define each part of the game as separate functions. There are individual functions that set up the basic screen, create rows of cars, trucks, logs and lily pads across the screen, handle the frogs movement, that check if the frog hit a car or if the frog is on a lily pad or log and can continue moving. Since each individual function handles everything each of them is simply called in the main function. As part of improving the developed game further, we implemented more features like UART handling, displaying an actual Frogger's sprite as well as changing the speed the cars and trucks move. The software was developed using Keil uVision software, written in C programming language, compiled as ARM assembly code for cortex M3 micro-controller processor.

1.2. Document Scope

This document serves as a record, documentation, and a proof of our work. The main purpose of this document is to document and define our work that we did to complete the tasks assigned for the Final Project, the Frogger arcade game, in EE379 in Spring of 2019. The design covers the functions that were provided by the team of TAs and the professor, as well as the functions that we defined in the project. This document discusses the software implementation, as well as testing and verification on the board.

1.3. Intended Audience

The expected and intended audience for this report is the EE379 session Teaching Assistant's, professor Christopher Fritz, along with Siddhant Khera and Jordan Dycha in the lab session on Tuesday at 10am. Any other students part of EE379, to whom the material be relevant, are invited to view to document but should keep confidentiality in mind when doing so.

2. Software Design

2.1. Embedded System Overview

Embedded Systems are implemented on a hardware platform. The hardware platform we use in this lab is an NXP LandTiger LPC1768 board, which is based on the Cortex M3 ARM based processor. The LCD display on the board is controlled directly by the micro-controller, which also interfaces with the pushbutton. The main function included in this document interfaces with the push-buttons, the UARTs, and controls the LCD display. Within this function there are other functions that are called that change and interact with the display and act on the pushbutton changes. Using this approach where we pre-defined void functions, before the main functions, that call other functions allowed us to significantly reduce the amount of lines of code this game requires. This in turn reduced the total size of the game and allowed for a faster operation of the LPC1768 board. This design can also allow for it to be re-used in a further scope where a similar application is required. The following functions were provided by Professor Christopher Fritz as part of this assignment, and were simply used to implement our design. The functions have not been defined in the API documentation below. Here are the given functions:

| Function | Parameters | Description |
|--------------------|--|--|
| GLCD_Init | none | Called to set up the display |
| GLCD_PutPixel | int x, int y | Draw a pixel at (x,y) in the current text color |
| GLCD_SetTextColor | short color | Change text color of the next pixels or characters drawn to the specified 16-bit color. (This function will only apply to pixels drawn after the call) |
| GLCD_SetBackColor | short color | Changes the background color behind text characters. |
| GLCD_Clear | short color | Clear the screen and set the background color. |
| GLCD_DisplayString | int line int col char font char *string | Draw string on the specified line and column (We'll use this function in Lab 4/5). |

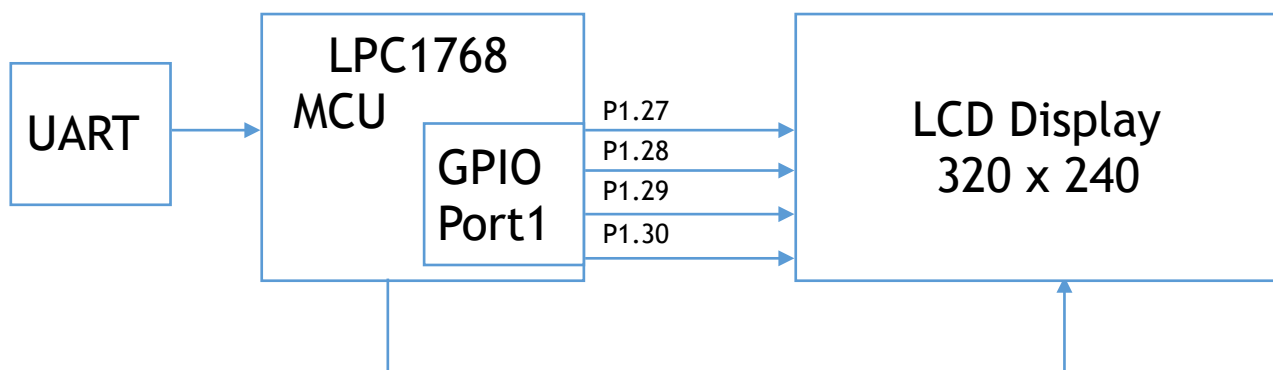


Fig 1: Schematic of the LPC1768 processor with the GPIO ports. LCD and UART used

2.2. Application Programming Interface

The following functions were used in implementing the design required for this lab. The code consists of multiple functions and that each performs a certain task. There are functions mentioned above that are not described in detail below. The game is built on a modular platform and the main() function simply calls particular functions as the next step in the game. The game can be improved or changed in various aspects by simply updating the respective functions. Each function is described in detail below.

2.2.1. Function `void frog(xx,yy)`

| | |
|-------------|--|
| Return type | void |
| Parameters | <code>int xx;</code> <code>int yy;</code> |
| Description | <p>This function is used to print the frog on the screen. Within this function we primarily use GLCD_PutPixel() that has been provided by the professor. GLCD_PutPixel() draws the Frog sprite pixel-by-pixel.</p> <p>The Frog was designed using Microsoft Excel per pixel, and this function maps each pixel as it should appear on the screen to create a sprite. The given coordinates of xx and yy are the two center point of the frog. Within this function, it create 17 pixels in appropriate directions to create the green body of the frog. Knowing the center points of the frog it also draws that red eyes using a set offset centered at the same centrepoint.</p> |

2.2.2. Function `void drawSquare(cx,cy,width,height,color)`

| | |
|-------------|--|
| Return type | void |
| Parameters | <pre> int cx; int cy; int width; int height; int color; </pre> |
| Description | <p>The purpose of this function is to draw a rectnagle according to the measurements defined. This functions works by first calling the function <code>GLCD_SetTextColor(color)</code>; to set the color defined by the user for the solid shape. It then progressively builds the shape by constraining the area it's building to that defined by the user. For a square, it uses the variable <code>i</code> to keep track of the <code>x</code> value, and <code>j</code> to keep track of the <code>y</code> value. Whenever the <code>if</code> statement is confirmed and the constraint is reached, the solid square completes. The constraint is that when <code>i</code> and <code>j</code> equal the sum of the center of the square and half of the side of the square.</p> |

2.2.3. Function `void display_lives(lives)`

| | |
|-------------|--|
| Return type | void |
| Parameters | <pre> int lives; </pre> |
| Description | <p>This function keeps track of the number of lives remaining in the game. Each time a collision function is called, the collision function calls this function to display the remaining lives. There is a counter that is subtracted each time the <code>collision()</code> function is called. This function then displays lives remaining using <code>if</code> statement triggered to the given number of lives remaining. Once the frog is on its final life, this function will not display any lives remaining.</p> |

2.2.4. Function `void basic()`

| | |
|-------------|--|
| Return type | void |
| Parameters | none |
| Description | This function is responsible for drawing the background for the game. This includes the safe zone at the bottom, the road for the cars and trucks to drive on, the safe zone in the middle, the river for the logs and lily pads to float on, and area in between the home squares. This is done by calling multiple <code>drawSquare()</code> functions. The multiple <code>drawSquare()</code> function calls are configured to the width and height of each zone required. Effectively, parallel calls of the <code>drawSquare</code> give. Visual effect of the whole screen being drawn at the same time. |

2.2.5. Function `void collision()`

| | |
|-------------|---|
| Return type | void |
| Parameters | none |
| Description | This function is called when the frog intersects with the edges of the screen, cars or trucks, or the water. " <code>collision()</code> " is responsible for decrementing the variable "lives," and resetting the location of the frog to the beginning, as well as redrawing the whole screen so that the old frog is not visible at it's previous location. |

2.2.6. Function `car1(int* rx)`

| | |
|-------------|---|
| Return type | void |
| Parameters | <code>int* rx;</code> |
| Description | This function displays three cars moving to the right. This is done by keeping track of values in an array which change by three every time the function is called. The cars are displayed by calling the " <code>drawSquare()</code> " function. The condition regarding the collisions of objects is also implemented here and calls the " <code>collision()</code> " function when the condition is met. |

2.2.7. Function void truck2(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three trucks moving to the left. This is done by keeping track of values in an array which change by two every time the function is called. The trucks are displayed by calling the "drawSquare()" function. The condition regarding the collisions of objects is also implemented here and calls the "collision()" function when the condition is met. |

2.2.8. Function car3(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three cars moving to the right. This is done by keeping track of values in an array which change by four every time the function is called. The cars are displayed by calling the "drawSquare()" function. The condition regarding the collisions of objects is also implemented here and calls the "collision()" function when the condition is met. |

2.2.9. Function truck4(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three trucks moving to the left. This is done by keeping track of values in an array which change by three every time the function is called. The trucks are displayed by calling the "drawSquare()" function. The condition regarding the collisions of objects is also implemented here and calls the "collision()" function when the condition is met. |

2.2.10. Function car5(int* rx)

| | |
|-------------|---|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three cars moving to the right. This is done by keeping track of values in an array which change by two every time the function is called. The cars are displayed by calling the "drawSquare()" function. The condition regarding the collisions of objects is also implemented here and calls the "collision()" function when the condition is met. |

2.2.11. Function void pad6(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three lily pads moving to the right. This is done by keeping track of values in an array which change by one every time the function is called. The lily pads are displayed by calling the "drawSquare()" function. Conditions are set using if statements that increment the location of the frog if the frogs body matches the location of any of the three objects in the row. |

2.2.12. Function void log7(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three logs moving to the left. This is done by keeping track of values in an array which change by four every time the function is called. The logs are displayed by calling the "drawSquare()" function. Conditions are set using if statements that decrement the location of the frog if the frogs body matches the location of any of the three objects in the row. |

2.2.13. Function pad8(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three lily pads moving to the right. This is done by keeping track of values in an array which change by two every time the function is called. The lily pads are displayed by calling the "drawSquare()" function. Conditions are set using if statements that increment the location of the frog if the frogs body matches the location of any of the three objects in the row. |

2.2.14. Function void log9(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three logs moving to the left. This is done by keeping track of values in an array which change by four every time the function is called. The logs are displayed by calling the "drawSquare()" function. Conditions are set using if statements that decrement the location of the frog if the frogs body matches the location of any of the three objects in the row. |

2.2.15. Function void pad10(int* rx)

| | |
|-------------|--|
| Return type | void |
| Parameters | int* rx; |
| Description | This function displays three lily pads moving to the right. This is done by keeping track of values in an array which change by three every time the function is called. The lily pads are displayed by calling the "drawSquare()" function. Conditions are set using if statements that increment the location of the frog if the frogs body matches the location of any of the three objects in the row. |

2.2.16. Function `void frog_up()`

| | |
|-------------|---|
| Return type | void |
| Parameters | none |
| Description | This function is used to change the y location of frog by negative eighteen every time the user pushes up on the joystick. Reducing it by eighteen moves the frog 18 pixels up on the screen. Before incrementing the frog the next location, this function redraws the whole screen by calling basic() so that the frog at the previous location is no longer visible once the frog moves. |

2.2.17. Function `void frog_down()`

| | |
|-------------|---|
| Return type | void |
| Parameters | none |
| Description | This function is used to change the y location of frog by positive eighteen every time the user pushes down on the joystick. Reducing it by eighteen moves the frog 18 pixels down on the screen. Before incrementing the frog the next location, this function redraws the whole screen by calling basic() so that the frog at the previous location is no longer visible once the frog moves. |

2.2.18. Function `void frog_left()`

| | |
|-------------|---|
| Return type | void |
| Parameters | none |
| Description | This function is used to change the x location of frog by negative eighteen every time the user pushes left on the joystick. Reducing it by eighteen moves the frog 18 pixels left on the screen. Before incrementing the frog the next location, this function redraws the whole screen by calling basic() so that the frog at the previous location is no longer visible once the frog moves. |

2.2.19. Function `void frog_right()`

| | |
|-------------|--|
| Return type | void |
| Parameters | none |
| Description | This function is used to change the x location of frog by eighteen every time the user pushed right on the joystick. Reducing it by eighteen moves the frog 18 pixels left on the screen. Before incrementing the frog the next location, this function redraws the whole screen by calling <code>basic()</code> so that the frog at the previous location is no longer visible once the frog moves. |

2.2.20. Function `int main()`

| | |
|-------------|--|
| Return type | void |
| Parameters | none |
| Description | This function is responsible for initializing the locations of the cars, trucks, lily pads, and logs. We also initialize the LCD display and call our " <code>basic()</code> " function here. All the object functions are called in an infinite while loop to ensure the movement of the objects. Here we also keep track of when our frog reaches home, and we keep track of number of times we reached home and number of lives remaining. when appropriate this function will print "YOU WON!", or "LOSER!". |

3. Testing and Verification

In this section, the verification of the design is done to see if it meets the requirements. The testing method's used were robust but focused on the function of the gameplay instead of singular functions. Each function was tested and confirmed as part of regular functioning of the game.

3.1. Objective Verification

The testing method for this implementation was on playing the game exhaustively to test if the parameters the game was designed for was being fulfilled or not. To begin, number of lives should have been displayed at all times in the corner of the screen. Then, the frog must have been able to move using the joystick on the board. If the frog was hit by cars/trucks, it should have started from the beginning center-point as well as had one life reduced. Then, the frog should have died if it landed in water, this should

have resulted in a loss of life as well as going back to the starting point. If the frog landed on the turtle or the frog, it should move with the block in the same direction and the speed without dying. Once it navigated through the waters, turtles and logs five successful times in five distinct homes, the game should display “You win!” If it dies four times, the screen should display “LOSER!” Since the testing and verification procedure requires multiple functions in tandem, only some functions were tested independently. All the others were tested visually as part of the exhaustive testing by playing continuously.

3.2. Function Verification

3.2.1. Function `int main()`

The primary purpose of this function was to display and call the functions. Once all functions were set-up and functioning, it was confirmed that everything has been configured correctly in the main function. It was also responsible for handling the input from the UART, and it was confirmed functioning since the frog could move as keys were pressed on the keyboard. This function was also responsible for calling functions which created rows with cars, trucks, logs and lily pads respectively. These functions moved the objects correctly. Finally, it also checked if the frog had entered one of the homes. If the frog does reach home, the function is supposed to increment the home counter by one. Once the home counter reaches 5, or the lives counter reaches 0, appropriate text was to be displayed on the screen. “You won!” And “LOSER!” Were printed in each respective cases successfully.

3.2.2. Function `void drawSquare(cx, cy, width, height, color)`

This function simply created a solid rectangle on the screen at the defined size as required. The functionality of the rectangle was visually inspected as part of the main function discussed prior in terms of it being called upon. Multiple functions called this function to draw the the screen background, cars, trucks, logs and lily pads. Each element was confirmed being drawn appropriately.

3.2.3. Function `void frog(xx,yy)`

This function created a frog seventeen pixels wide. Once the center points of the frog were declared, this function displayed the frog. It was confirmed working optimally since other functions call on this to redraw the frog once input from the UART is received.

3.2.4. Function `void display_lives(lives)`

The primary function for this was to display lives on the right part of the LCD display in the grey zone. Each time the frog died (that is, the collision() function was called), this would redraw the basic display and only print the remaining

lives. It was visually confirmed that the number of lives updated each time the frog died.

3.2.5. Function `void basic()`

This function is called in the `main()` function and is responsible for drawing the background of the game. The safe zones, road, river and spaces at the top were drawn appropriately.

3.2.6. Function `void collision()`

The primary function for this was to decrement the lives counter each time this function was called. It also reset the background, and reset the frog to the original starting point in the game. This function was called each time the position of the frog overlapped with the car, truck or the water. This was confirmed since hitting these elements resulted in a live being reduced and the frog resetting to the original location.

3.2.7. Function `void car1(int* rx)`

The primary function for this was to display three different sized squares in the given row. The sizes of the squares were equal and three cars were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and cars occurred. If so, the `collision()` function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.8. Function `void truck2(int* rx)`

The primary function for this was to display three different sized rectangles in the given row. The sizes of the rectangles were equal and three trucks were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and trucks occurred. If so, the `collision()` function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.9. Function `void car3(int* rx)`

The primary function for this was to display three different sized squares in the given row. The sizes of the squares were equal and three cars were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and cars occurred. If so, the `collision()` function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.10. Function `void truck4(int* rx)`

The primary function for this was to display three different sized rectangles in the given row. The sizes of the rectangles were equal and three trucks were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and trucks occurred. If so, the `collision()` function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.11.Function void car5(int* rx)

The primary function for this was to display three different sized squares in the given row. The sizes of the squares were equal and three cars were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and cars occurred. If so, the collision() function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.12.Function void pad6(int* rx)

The primary function for this was to display three different sized squares in the given row. The sizes of the squares were equal and three turtles/lily pads were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and lily pads occurred. If so, the collision() function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.13.Function void log7(int* rx)

The primary function for this was to display three different sized rectangles in the given row. The sizes of the rectangles were equal and three logs were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and logs occurred. If so, the collision() function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.14.Function void pad8(int* rx)

The primary function for this was to display three different sized squares in the given row. The sizes of the squares were equal and three turtles/lily pads were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and lily pads occurred. If so, the collision() function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.15.Function void log9(int* rx)

The primary function for this was to display three different sized rectangles in the given row. The sizes of the rectangles were equal and three logs were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and logs occurred. If so, the collision() function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.16.Function void pad10(int* rx)

The primary function for this was to display three different sized squares in the given row. The sizes of the squares were equal and three turtles/lily pads were confirmed being displayed visually. This function is also responsible for checking to see if a collision between the frog and lily pads occurred. If so, the collision() function was called. It was also confirmed that this function was not causing an overlap with another function or disruptions.

3.2.17.Function void frog_up()

The primary function for this was to increment the frogs position up by 18 pixels. It was confirmed working as the frog moved when input from either the joystick or UART was received.

3.2.18.Function void frog_down()

The primary function for this was to increment the frogs position down by 18 pixels. It was confirmed working as the frog moved when input from either the joystick or UART was received.

3.2.19.Function void frog_left()

The primary function for this was to increment the frogs position left by 18 pixels. It was confirmed working as the frog moved when input from either the joystick or UART was received.

3.2.20.Function void frog_right()

The primary function for this was to increment the frogs position right by 18 pixels. It was confirmed working as the frog moved when input from either the joystick or UART was received.

Glossary

List of Abbreviations

- LCD - Liquid Crystal Display
- UART - Universal Asynchronous Receiver-Transmitter
- ARM - Advanced RISC Machines
- RISC - Reduced instruction Set Computing

Hardware References

- [1] <https://os.mbed.com/media/uploads/chris/lpc1768-refdesign-schematic.pdf>
- [2] https://www.nxp.com/docs/en/data-sheet/LPC1769_68_67_66_65_64_63.pdf