Final Project Report

**Creation of Gamebox portable gaming system**

**Using STM32F407**

Jake Bonte, Joseph Krambeer, William Jensen

University of St. Thomas

ENGR 331-01

Professor Nepal

Spring 2016

Table of Contents

**Title Page**..................................................................................................................1

**Table of Contents**.....................................................................................................2

**Executive Summary**................................................................................................3

**Design**........................................................................................................................4

Customer Requirements.........................................................................................4

Design Overview and Analysis..............................................................................5

System Block Diagram..........................................................................................6

Bill of Materials.....................................................................................................7

**Design Validation**....................................................................................................8

**Power Budget Analysis/Real System Power**.......................................................10

**Lessons Learned and Conclusions**......................................................................11

**References**..............................................................................................................13

**Appendices**.............................................................................................................14

Appendix A- Pricing Information.....................................................................14

Appendix B- Revised Power Budget Analysis.................................................15

Appendix C- Electrical Schematic.....................................................................16

Appendix D- Source Code.................................................................................17

Appendix E- Flow Charts..................................................................................37

Appendix F- Game Description.........................................................................42

Appendix F- Mechanical Diagram of System Enclosure ..................................43

**EXECUTIVE SUMMARY:**

The goal of this project was to design a handheld gaming system that allowed easy programming and hardware interfacing. Named Gamebox, our design would allow consumers to gain relevant hardware and software experience in the realm of microprocessors through customization of each relevant aspect. The system utilized a STM32F407 processor and built peripherals, such as a color display, sound, and a tactile button interface, around it to allow interactive gameplay. The processor and peripherals were selected to meet project specifications, allow ease of creation and integration, and be under a certain budget. To streamline the process of interfacing with the hardware of the system, we broke down our algorithms for hardware into: Power Level Detection, Input Signal Handling, Graphics Output, and Audio Output. By using effective algorithms we were to accomplish this task of making an entertaining gaming system as well as allowing consumers to interface to our system with their own games.

Our current prototype successfully shows that the concept of creating a handheld gaming device that can be interfaced with in an open source style, is quite possible and practical. Using these hardware and software components as a base design allows the Gamebox to be further customized by the customer both in hardware and software.

**DESIGN:**

**a) CUSTOMER REQUIREMENTS**

The gaming system**:**

1. Must operate using ARM STM32F407VGT6 processor.
2. Must be smaller than 6inx6inx3in
3. Must be handheld, portable, and ergonomic.
4. Must have a battery life of at least 1 hour.
5. Must have an LED to indicate power status.
6. Must have a tactile button interface with at least a button and a D-pad.
7. Must have a color LCD graphical display at least 2.9in diagonal.
8. Must have a mechanical power switch.
9. Must have audio capability.
10. Must boot up in several seconds.
11. Must have relatively instant input and feedback graphically.
12. Must not burst into flames at any time.
13. Must have a backlight on/off feature.
14. Must remain operation on at least a 1-foot drop test.

**Bonus:**

1. Make an awesome game!

**b) DESIGN OVERVIEW and ANALYSIS**

Our design, while somewhat crude, was made to maximize efficiency given the time constraints. We wanted to base our design off of a system that could run with 4 AA batteries. The reason we chose the 4 AA battery power supply was simply because we had access to a battery holder/pack that held 4 AA batteries. When in series, ideally, the batteries would produce a 6 V output. This forced us to design certain circuits to control our voltages. To power our STM32F407 Discovery Board, we had to create a circuit using a linear regulator to step our voltage from 6 V to 5 V. The regulator we chose, which is outlined in Appendix A, satisfied our required voltage step. The batteries being 6 V total worked in our favor when we designed our audio circuit, which required an audio operational amplifier. The amplifier we used we chose because it was available on hand at St. Thomas. We used the 6 V as rails for our amplifier, and the 3 Vpp sinusoidal output form our Discovery Board as logic for our circuit. Using our Discovery Board, we varied the frequency of the 3 Vpp sinusoidal output to produce different tones. The output of our amplifier ended up being a distorted sinusoidal wave, which, when put through our simple speaker, which was also chosen because it was on hand at St. Thomas, produced an awesome 8-bitlike sound. We designed the buttons in a pulldown like circuit, so when a button was pressed we would get an output voltage. The buttons we original chose were pressure sensitive, and this caused many inexplicable problems in our original D-Pad circuit. When we switched to the current buttons we used, it produced more of a digital output which ended up being more responsive and overall easier to read. We also wanted our final design to have an LED indicator to indicate power status. The LED we chose was a 2 color LED that we wired to show red when power was low, and green when power was high. We made logic for the LED using the STM32F407 Discovery Board's built in power indicator, and tested it at extremely low voltages which yielded the same results. We chose our display based upon its size and cost specifications, and the ease of access we had when coding to the display. At the time, we thought we could code to our display using just a driver and our STM32F407 Discovery Board. As the project went on, however, in order to fully finish, we ended up having to order a controller board with a separate processor in order to reasonably write code to it. The controller board, while handling the commands and initialization sequence of the display, also had a DC-DC boost convertor from 3.3V to 20V built inside. All of these components were housed in a 6x6x2.5 in 3d printed plastic case. We chose to build the case this way for two reasons. Firstly, it would be easy to design and prototype a case relatively fast using a 3d printer. Secondly, it would be large enough and rigid enough to handle any drop tests we put it through. We made appropriate extruded cuts inside the case to externally show our display, buttons, and power switches. When tested on a 1-foot drop test, the case held up extremely well and could probably be rated to survive further distances. Given the time constraints of our project, there are definitely some areas that we could have optimized more.

**c) SYSTEM BLOCK DIAGRAM**



**d) BILL OF MATERIALS AND TOTAL DESIGN COST**

**Parts and Materials for Gamebox:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Cost** | **Quantity** | **Total** |
| LCD TFT DISPLAY | $30.50 | 1 | $30.50 |
| LCD Controller Board | $35.00 | 1 | $35.00 |
| Mechanical Buttons | $0.50 | 6 | $3.00 |
| Plastic Case | $2.50 | 1 | $2.50 |
| Linear Regulator | $1.02 | 1 | $1.02 |
| Speaker and Control Circuit | $5.72 | 1 | $5.72 |
| STM32F407 Discovery Kit | $21.17 | 1 | $21.17 |
| Battery Holder | $1.51 | 1 | $1.51 |
| Switch | $0.50 | 1 | $0.50 |
| Power LED indicator | $0.25 | 1 | $0.50 |
| Transistors for LED | $0.50 | 2 | $1.00 |
| Miscellaneous Expenses | $2.00 | 1 | $2.00 |
| **Total** |  |  | $104.42 |

\*For more in-depth analysis and reference of parts see Appendix A

The total design ended up being more than we predicted. This is primarily because we did not account for the cost of the LCD controller board. That being said, and given the price of prototypes, if produced on a larger scale the price of our design could easily be reduced significantly.

**DESIGN VALIDATION**

To verify the design of our system, we ran a multiple tests to verify that parts worked on their own and as a part of the system as a whole. To test our linear regulator, we created a circuit that powered the linear regulator with varying voltages and had a circuit to observe the output voltage of the linear regulator. We probed this wire with an oscilloscope and observed how the output of our linear regulator changed with different voltages, and we were able to verify it would give us a steady 5V when powered with the expected 6V from the batteries. Then we dropped our voltages below 6V and made sure the dropout of linear regulator was .5V as excepted, and we were able to verify this on the oscilloscope. We then used batteries on place of the power supply and verified that they would be able to power the board and not cause any harm, and after testing we concluded that the batteries would be able to power our system without any harm. To test our audio system, we first powered it with power supply and powered the op-amp with a function generator outputting at 3V to verify that the op-amp could amplify the wave we would send from the STM board. After checking many frequencies throughout the human hearing range, we were satisfied with the quality of the audio, and then we replaced the function generator without STM board outputting a sine wave from its DAC system. We composed many songs with different frequency of notes, and after listening to them we were satisfied with the quality of our audio when it was powered 6V and took the sine wave off our STM board. To test our buttons a, a simple program was created to light up LEDs when buttons were pressed. To get idea of how to de-bounce the presses we hooked up the outputs of the buttons to an oscilloscope and measured times that the signal was bouncing. After accounting for these we added these into our program and verified that the LEDS turned on only when their respective button was pressed, which they did. To test out our power status indicators we powered our board with a power supply and slowly dropped the power supply to our board and made sure above the power threshold our LED was green and below it, it was red, which our testes validated. To test our screen, after we were able to get the screen working we wrote many different colors to fill the screen and made sure we were able to fill some areas of the screen with different colors. We were able to do this with the screen, so then we moved to start combining our parts and verify that they would work together in the design. After combining files connecting parts, we made the base of our prototype. We then checked when everything was combined together that the audio quality would not drop, and after listening to the audio before and after combining the parts, we couldn't hear any change in the audio and were satisfied by it. Also we changed voltages we powered the device with, and verified that the power status LED still worked as before. To check the display and buttons, we created the game and made sure they ran as expected, and we were able to make sure only one game action was made per press. We also made sure that on the d-pad whenever the d-pad was moved in a certain direction it would make the corresponding game action and would not carry out any other actions. We were able to make sure the game did this, so next we verified that the game was being displayed right on the screen. Observing the time to write to the screen, it took around 5 seconds to refresh the entire screen when the player moved, so we decided to only locally refresh a 3x3 box around the player whenever they took an action. This was able to make the game run more smoothly, and the response time on the screen was in a desirable range where the screen would be refreshed by the end of the button press. Then to verify the design as a whole worked, we ran left the game on for 2 hours and made sure it was still playable, and we verified that the game could be on 2 hours and still work with fresh batteries. Also we ran through the level in the game multiple times to make sure the buttons worked as expected and no erratic behavior was happening, and which there was not. To test the stability of the circuits we dropped the assembled product 5 times from 1 foot and shook the design multiple times and made sure that everything was working as it was before the drops/shakes. After each drop/shake the game worked as before, and thus we felt the system was stable enough. Also we measured the overall temperature when the game was running and made sure no component got warm in the 2 hour period, and after touching each component they were roughly all room temperature. The design for our case was loosely based on the Gameboy Advance, but rather than having rounded edges we decided to go more for a box design to maximize space so we could easily install parts. The box itself was too sharp, so we added fillets in on the edges to make it more ergonomic. We chose our placement of buttons in order to maximize tactile response. We took the D-pad placement and button place from the Gameboy Advance's design with exception of the A and B buttons. We stacked the A and B buttons in order to allow our screen to have more room to fit. Lastly, we placed the batter holder/socket in the middle of the back of our case to allow fingers some room to comfortably wrap around the case itself. Overall the case may be too deep/thick to be completely ergonomic for people with small hands, so further could be made.

**POWER BUDGET ANALYSIS**

The finalized power budget analysis has a few changes from our initial proposal. Firstly, the DC-DC boost convertor is no longer needed as we ordered a controller board with that built in. Any voltage changes we made in the circuit will be outlined in the power budget consumption of the controller board. Additionally, the STM32F407 was found to be drawing more current upon loading and running code, therefore we edited that as needed. To analyze the power budget analysis and our results on actual vs predicted power consumption, we measured the total voltage and current drawn by our system in the corresponding ON and Idle modes. The total system when on consumed roughly 1.674 Wa/h at 6.2 V with .27 A being drawn. This number is pretty close to our estimated value in our revised power budget analysis, see Appendix B. Any differences could be because our power consumption varies quite a bit when in the ON and IDLE modes with values of variance reaching 20 mA. We also measured the recorded power by taking the upper end of ranges for the amperage and voltage measurements. To make these measurements we used a voltage supply as our main source and read its current draw. We double checked these multi-meter to assure a certain level of reasonable accuracy. That being said, these factors could contribute to the small difference in our power consumption.

**LESSONS LEARNED AND CONCLUSIONS**

The biggest lesson we learned throughout the project is that we need to go more in-depth into research when choosing parts. We looked at data sheets to decide if the part was good for us, but we truly need to analyze every single component of the part and truly understand what is required to get the part working before jumping in. We definitely bought our display too early, and backed ourselves into a corner when it came to selecting displays. Mainly we bought a display without a controller, and we assumed we could get it working, but in hindsight that would be virtually impossible. This is opposed to the other option of buying a display with a built in controller. We ended up having to buy a controller separately regardless and it nearly doubled the price of our project. Additionally, we had a huge amount of trouble getting our display initialized. After attempting to write an initialization sequence we just couldn’t get our display to work. We ended up re-writing the code and trying to fully understand what the sample initialization sequence was doing and after each of us came up with an initialization code, we had one that worked. This problem stemmed from the fact that the data sheet ended up being misleading because of a line that was described as active low being active high. It also stemmed from a misunderstanding of when the line should dropped in reference to other lines of code. We also learned that through contacting the company that made the display, that they were more than willing help and provide with accurate information. Another huge lesson we learned was soldering. We ran into problems multiple times when it came to poor soldering jobs. In the future we need to test to make sure 100% that the solders are separated/spaced enough apart to not allow any interference. One example of this was our audio, which, although working initially ended up shorting through solder after some use. The problem with trying to separate the solder enough, however, is that we also wanted to keep the size of our components to a minimum. Finding the happy medium is probably what we need to shoot for. Another lesson we learned was button choice. The buttons we chose initially were very pressure sensitive, and when we pushed another button the D-pad, we found we would get a small voltage coming out from other buttons as if they had been slightly pressed. Mainly we learned that ideal components weren’t always acting ideal when created. Another lesson we learned was verifying step by step that our components and circuits worked independently and together with other relevant pieces. If we just verified them independently and then threw the whole project together, it would be hard to debug the electrical design as a whole and be a more difficult thing to do overall. The way we did it, by verifying each combination of parts and each part individually, seemed to work and debug problems well. Another lesson we learned was to stay on top of work. We paced ourselves quite well this project and it helped us debug and complete the project on a timely manner. If we hadn’t put the work in earlier we would have run into problems that we wouldn’t have had time to solve. Another thing we learned was designing and fitting components to a specified area so that they feel and fit into our design requirements. Generally, we also learned how certain circuits such as audio amplifiers, boost convertors, and linear regulators are created and used. We also learned how to optimize code to maximize responsive feedback. We also learned to be price sensitive and how to find certain parts are reasonable prices. Another huge lesson, is we learned how to use utilize our team to maximize our strengths, and let people with said strengths help each other learn about those things. Also due to time constraints, it worked better to allow people to work on what they were best at in order to minimize time it takes to do things, which gave us more time for debugging and create a more polished product. We also learned the utility and power that microcontrollers bring through the variety of applications. Another thing is we designed and ordered a PCB that we ended up not being able to use. While we did learn the practical ordering and design of PCBs, we probably should have made sure we needed the PCB 100% before ordering it.

**References**

Display Datasheet:

<http://www.newhavendisplay.com/specs/NHD-3.5-320240MF-ATXL-1.pdf>

Display Driver Datasheet:

<http://www.newhavendisplay.com/app_notes/NV3035C.pdf>

Speaker Module Datasheet:

<http://www.digikey.com/product-detail/en/cui-inc/CLS0231-L152/102-1561-ND/1630875>

Online Tone Generator:

<http://onlinetonegenerator.com/>

LCD Connector:

<http://www.molex.com/pdm_docs/sd/512961894_sd.pdf>

AA Battery Holder Datasheet:

<http://www.memoryprotectiondevices.com/datasheets/BH24AAW-datasheet.pdf>

Generating sine wave on STM board:

<http://amarkham.com/?p=49>

Display controller datasheet:

<http://www.newhavendisplay.com/app_notes/SSD1963.pdfs>

Display controller board datasheet:

<http://www.newhavendisplay.com/specs/NHD-3.5-320240MF-20%20Test%20Board.pdf>

Example code for SDD1963 LCD controller:

<https://newhavendisplay.com/appnotes/excode/txt/TFT/TFT_35M.txt>

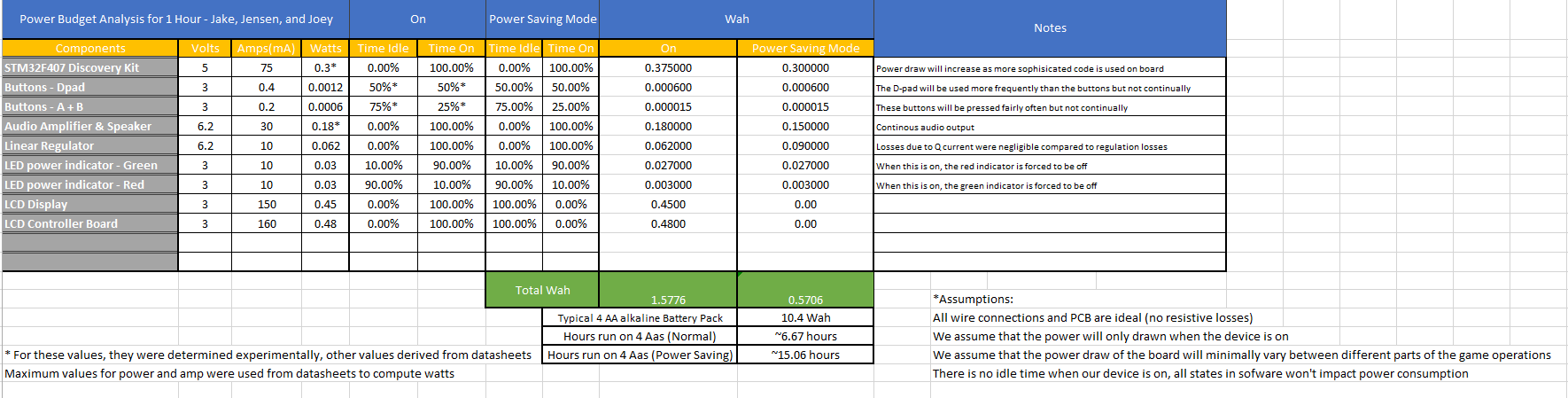
Example of video game source code we looked at (game is DOOM for Linux): <https://github.com/id-Software/DOOM>

**APPENDICIES**

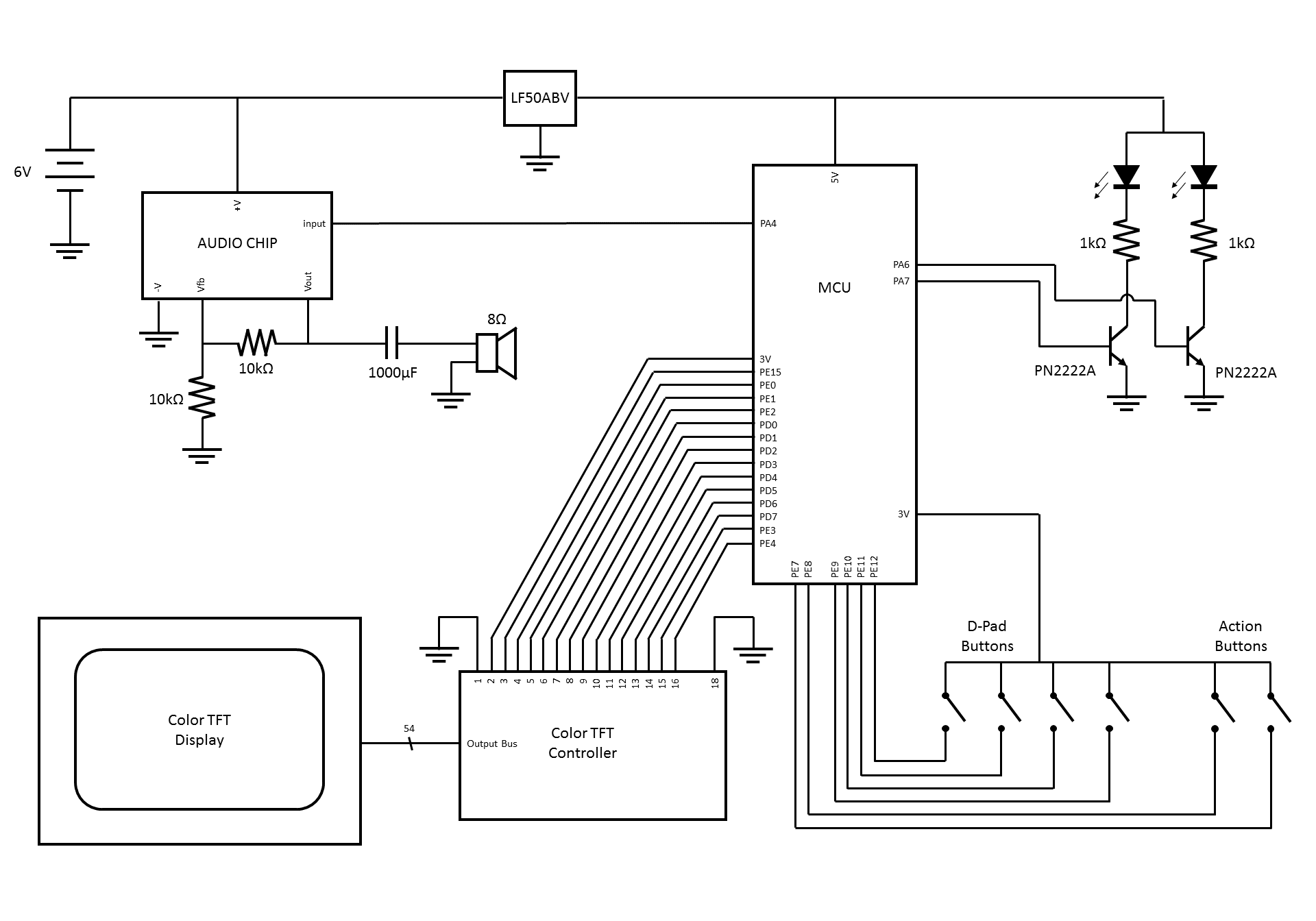
Appendix A - Pricing Information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item | Cost | Quantity | Total | Notes |
| LCD TFT DISPLAY | $30.50 | 1 | $30.50 | <http://www.digikey.com/product-detail/en/newhaven-display-intl/NHD-3.5-320240MF-ATXL--1/NHD-3.5-320240MF-ATXL--1-ND/2165878> |
| LCD Controller Board | $35.00 | 1 | $35.00 | <http://www.digikey.com/product-detail/en/newhaven-display-intl/NHD-3.5-320240MF-20/NHD-3.5-320240MF-20-ND/2424963> |
| Mechanical Buttons | $0.50 | 6 | $3.00 | Average price found online between multiple similar products |
| Plastic Case | $2.50 | 1 | $2.50 | Based upon amount of material used to create the case by the 3d printer (Weighed 100 g, can buy plastic at 1000g for $24.95) |
| Linear Regulator | $1.02 | 1 | $1.02 | <http://www.digikey.com/product-detail/en/stmicroelectronics/LF50ABV-DG/497-14792-5-ND/3945961> |
| Speaker and Control Circuit | $5.72 | 1 | $5.72 | [http://www.digikey.com/product-detail/en/cui-inc/CLS0231-L152/102-1561-ND/1630875 for speaker, http://www.digikey.com/product-detail/en/stmicroelectronics/LM833DT/497-1597-2-ND/591699 for op amp](http://www.digikey.com/product-detail/en/cui-inc/CLS0231-L152/102-1561-ND/1630875) |
| STM32F407 Discovery Kit | $21.17 | 1 | $21.17 | <http://www.digikey.com/product-detail/en/stmicroelectronics/STM32F407G-DISC1/497-16287-ND/5824404> |
| Battery Holder | $1.51 | 1 | $1.51 | <http://www.digikey.com/product-detail/en/mpd-memory-protection-devices/BC4AAW/BC4AAW-ND/66733> |
| Switch | $0.50 | 1 | $0.50 | Bought at Axman cost $.50 |
| Power LED indicator | $0.25 | 1 | $0.50 | Price estimated on digikey using multiple similar products |
| Transistors for LED | $0.50 | 2 | $1.00 | Bought at Axman cost $.50 each |
| Miscellaneous Expenses | $2.00 | 1 | $2.00 | Resistors, Capacitors, Wiring, Hot Glue, Supports etc. |
| Total |  |  | $104.42 |  |

Appendix B - Revised Power Budget Analysis



Appendix C - Electrical Schematic



Appendix D - Source Code for Project

The following code is broken down in such way that functions and variables are grouped together by what their overall purpose is for the device. The parts include items relevant to the power status system, items that were used by the audio system, functions to communicate with the LCD, functions that dealt with the map of the game, items used by the game logic, and the main function of the system.

--------------------------------**Main.c**---------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stm32f4xx.h" // Device header

#include <stdbool.h> // for boolean types

#include "Power.h" // defines how to startup power status checking / LED interfacing

#include "Audio.h" // defines audio output and values

#include "LCD.h" // defines interface methods for LCD screen

#include "Map.h" // defines map variables/types/functions to change map

#include "Game.h" // defines game rules and input handling

Tile tiles[10];//used to store the tiles colors/patterns

//this is used to store indices to the tiles array, thus this will look up what

//tile is at a certain position in the map and then place the corresponding tile there

const int ref\_map[map\_height][map\_width] = //12x16 tiles

{

{0,4,0,2,0,0,0,0,0,4,3,3,0,0,0,2}, //1

{2,2,0,2,0,0,0,0,0,0,0,0,0,0,5,0}, //2

{0,0,0,2,2,2,6,2,0,0,0,0,0,0,0,0}, //3

{0,0,2,2,0,0,0,0,0,0,0,0,3,0,0,0}, //4

{0,0,2,2,0,0,0,0,2,2,0,3,3,3,0,0}, //5

{6,2,2,2,2,2,2,4,3,3,0,0,3,0,0,0}, //6

{0,8,0,0,0,4,2,0,4,3,3,0,0,0,0,0}, //7

{6,5,2,2,2,6,2,2,2,2,2,2,2,2,2,5}, //8

{3,4,3,0,2,0,8,3,3,3,0,0,8,5,8,0}, //9

{3,3,3,3,2,6,0,7,6,7,0,7,5,2,7,5}, //10

{3,0,0,3,2,6,5,2,6,8,6,0,0,7,0,5}, //11

{0,3,3,4,2,0,5,2,8,5,8,5,8,8,0,7} //12

};//ref\_map, determines how the game map will look

//Tile correspondences :

//0 - unvisited tile

//2 - magma tile

//3 - ice tile

//4 - breakable tile

//5 - up tile

//6 - down tile

//7 - left tile

//8 - right tile

int playerX,playerY; //holds player's place on map

int map[map\_height][map\_width]; //holds indices of map tiles

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MAIN METHOD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int main(void)

{

initAudio();//starting audio output logic

init\_power\_status();//starting LED status logic

Delay\_init(); // Initialize Timers

TFT\_IO\_init();// Initialize the I/O

TFT\_init(); // Initiallize the screen

delay\_ms(1000); //waiting a second

init\_tiles(); //creating the tiles used in the game

reset\_map(); //reseting the map, and displaying the screen

game\_button\_init();//setting the state of buttons used in the game logic so that

//they are able to interface with the game

//running the game while it is not won

while( !check\_win() )

{

while( !check\_user\_input() ){}//wait until user has input something

delay\_ms(2);//wait one ms to debounce

input\_handler();//handle input

}//while

TFT\_fillScreen(0x44, 0x77, 0xAA);//winning screen color fill

return 0;

}//main

///////////////////////////////////////////////////////////////////////////////

// INTERUPT HANDLERS //

///////////////////////////////////////////////////////////////////////////////

void TIM4\_IRQHandler(void)

{

//this interrupt handler is used to check the status of the voltage level on the board

//and when it drops below the threshold, change the color of the LED status from green to red

NVIC\_ClearPendingIRQ(TIM4\_IRQn);

if( (PWR->CSR & 4)==4 )

{

GPIOA->ODR &= ~(0x00000040);//toggle status of output bits

GPIOA->ODR |= 0x00000080; //toggle status of output bits

}//if power below threshhold, RED LED on

else

{

GPIOA->ODR &= ~(0x00000080);//toggle status of output bits

GPIOA->ODR |= 0x00000040; //toggle status of output bits

}//if power above threshhold, green LED on

TIM4->SR &= ~TIM\_SR\_UIF;//clearing flag

}//TIM4\_IRQHandler

void TIM7\_IRQHandler(void)

{

//This function is used to output values from the DAC that make up the basic part of

//our audio system. It keeps track of what part of the sine wave is being output, how many

//waves on a certain delay have been output, and what the frequency of notes should be.

#define delayLength 12 //used to tell how many different notes there are

#define speedDelay 64 //used to determine the speed at which the notes are played

static int i = 0; //used to keep track of which part of the sine wave is being output

static int j = 0; //used to keep track which delay value is being used

static int z = 0; //used to keep track of how many times a certain note has been played

static const int delays[delayLength] =

{

270, 250, 230, 210, 210, 230, 250, 270, 250, 230, 250, 230

};//array used to hold delay values

NVIC\_ClearPendingIRQ(TIM7\_IRQn);

TIM7->SR &= ~TIM\_SR\_UIF;//clearing flag

TIM7->CR1 |= 0x00000001; //renabling counting

DAC->DHR12RD = sineParts[i];//putting value in register

DAC->SWTRIGR |= 0x00000001; //outputting value in register

if( i==(WAVE\_RES-1) )//if i has output a full sine wave

{

if(z==(speedDelay-1))//if there have been speedDelay number of waves output

{

TIM7->ARR = delays[j];//getting next delay

j=(j+1)%delayLength; //incrementing j index

}//if z equals speedDelay-1

z=(z+1)%(speedDelay);//increment and mod z index to keep accurate

//count of how many outputs

}//if i equals WAVE\_RES-1

i = (i+1)%WAVE\_RES;//incrementing i index and mod it to get next sine wave index

}//TIM7\_IRQHandler

--------------------------------**Main.c**---------------------------------------

--------------------------------**Power.h**--------------------------------------

#ifndef \_\_POWER\_H\_

#define \_\_POWER\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void init\_power\_status(void);

#endif //\_\_POWER\_H\_

--------------------------------**Power.h**--------------------------------------

--------------------------------**Power.c**--------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stm32f4xx.h"

#include "Power.h"

void init\_power\_status(void)

{

//This function is used to enable a timer and the power status register to occasionally check

//the STM board's onboard voltage and output signals based on whether the power is above or below

//that threshold.

RCC->APB1ENR |= RCC\_APB1ENR\_PWREN;//enabling PWR's clock

RCC->APB1ENR |= RCC\_APB1ENR\_TIM4EN;//enabling TIM4's clock

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN;//enabling port A's clock

PWR->CR |= 0x000000D0;//setting PLS to 110, 2.8V threshold

//PA6 - controls green LED

//PA7 - controls red LED

GPIOA->MODER |= 0x00005000;//set PA6 and PA7 as outputs

GPIOA->ODR |= 0x00000040;//set PA6 output high to power green LED

//setting timer 4 to run at 1KHz and to count to 1000 so that it the counter resets every

//second. (1/1KHz)\*1000 = 1 second

TIM4->PSC = 15999; //no prescalar

TIM4->ARR = 999; //auto reload register

TIM4->DIER |= 0x00000001; //setting interrupt bit enabled

TIM4->CR1 |= 0x00000001; //enabling counting on timer

//enabling interrupt for timer 4

\_\_enable\_irq();

NVIC\_SetPriority(TIM4\_IRQn,0);

NVIC\_ClearPendingIRQ(TIM4\_IRQn);

NVIC\_EnableIRQ(TIM4\_IRQn);

}//init\_power\_status

--------------------------------**Power.c**--------------------------------------

--------------------------------**Audio.h**--------------------------------------

#ifndef \_\_AUDIO\_H\_

#define \_\_AUDIO\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Defines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#define WAVE\_RES 128 //declaring that the sine wave is broken down into 128 parts

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Defintions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void initAudio(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Global Variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//declaring the global variable to hold the sine wave, define in Audio.c

extern const unsigned short sineParts[WAVE\_RES];

#endif //\_\_AUDIO\_H\_

--------------------------------**Audio.h**--------------------------------------

--------------------------------**Audio.c**--------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stm32f4xx.h"

#include "Audio.h" // defines audio output and values

//Sine wave values based on calculation online, see references for the website the values were

//take from

const unsigned short sineParts[WAVE\_RES] =

{

2048, 2145, 2242, 2339, 2435, 2530, 2624, 2717, 2808, 2897,

2984, 3069, 3151, 3230, 3307, 3381, 3451, 3518, 3581, 3640,

3696, 3748, 3795, 3838, 3877, 3911, 3941, 3966, 3986, 4002,

4013, 4019, 4020, 4016, 4008, 3995, 3977, 3954, 3926, 3894,

3858, 3817, 3772, 3722, 3669, 3611, 3550, 3485, 3416, 3344,

3269, 3191, 3110, 3027, 2941, 2853, 2763, 2671, 2578, 2483,

2387, 2291, 2194, 2096, 1999, 1901, 1804, 1708, 1612, 1517,

1424, 1332, 1242, 1154, 1068, 985, 904, 826, 751, 679,

610, 545, 484, 426, 373, 323, 278, 237, 201, 169,

141, 118, 100, 87, 79, 75, 76, 82, 93, 109,

129, 154, 184, 218, 257, 300, 347, 399, 455, 514,

577, 644, 714, 788, 865, 944, 1026, 1111, 1198, 1287,

1378, 1471, 1565, 1660, 1756, 1853, 1950, 2047

};//sine wave of amplitude broken into 128 parts

void initAudio(void)

{

//this function is used to configure the DAC, port A, and timer 7 so that

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN;//enabling port A's clock

RCC->APB1ENR |= RCC\_APB1ENR\_DACEN; //enabling DAC's clock

RCC->APB1ENR |= RCC\_APB1ENR\_TIM7EN; //enabling TIM7's clock

GPIOA->MODER |= 0x00000100; //putting PA4 in output mode

//all of these are for DAC channel 1

DAC->CR |= 0x00000004;//DAC trigger enable

DAC->CR |= 0x00000038;//DAC select software event

DAC->CR |= 0x00000002;//no buffered output

DAC->CR |= 0x00000001;//enable DAC

//TIM7 Setup

TIM7->PSC = 0; //no prescalar

TIM7->ARR = 300;//auto reload register

TIM7->DIER |= 0x00000001; //setting interrupt bit enabled

TIM7->CR1 |= 0x00000008; //One pulse mode

TIM7->CR1 |= 0x00000001; //enabling counting on timer

\_\_enable\_irq();//enabling interrupts

NVIC\_SetPriority(TIM7\_IRQn,1);

NVIC\_ClearPendingIRQ(TIM7\_IRQn);

NVIC\_EnableIRQ(TIM7\_IRQn);

}//init audio

--------------------------------**Audio.c**--------------------------------------

--------------------------------**LCD.h**----------------------------------------

#ifndef \_\_LCD\_H\_

#define \_\_LCD\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Defines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#define startOffset 27 //start of screen is 27 pixels in x direction from 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void TFT\_IO\_init(void);

void TFT\_init(void);

void TFT\_writeCMD(char CMD);

void TFT\_writeDATA(char DATA);

void TFT\_windowSet(int start\_x, int end\_x, int start\_y, int end\_y);

void TFT\_fillScreen(char RED, char GREEN, char BLUE);

void TFT\_fill\_area(int start\_x, int end\_x, int start\_y, int end\_y, unsigned int color );

void TFT\_fill\_pixel(int x,int y, int color);

void delay\_ms(int ms);

void Delay\_init(void);

#endif //\_\_LCD\_H\_

--------------------------------**LCD.h**----------------------------------------

--------------------------------**LCD.c**----------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stm32f4xx.h"

#include "LCD.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Delay Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void Delay\_init()

{

RCC->APB1ENR |= RCC\_APB1ENR\_TIM6EN;//enabling timer 6's clock

TIM6->PSC = 15999; //prescalar to divide 16MHz clock to 1KHz

}//Delay\_init

void delay\_ms(int ms)

{

//this function will delay ms+1 milliseconds as the timer 6 counts at 1KHz

//so once every millisecond

if(ms <= 0){return;}//if value is 0 or less do not delay any time

TIM6->ARR = ms; //set clock to count to how many ms you want delayed

TIM6->CNT = 0; //start count at 0

TIM6->CR1|= TIM\_CR1\_CEN;//enable counting

while(TIM6->SR==0);//wait until uIF flag is set

TIM6->SR=0; //clear uIF flag

}//delay\_ms

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Display Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void TFT\_IO\_init()

{

//This function is used to set the pins used as outputs to the display controller

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIODEN;

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOEEN;

// Port D config, data port

GPIOD->MODER |= 0x00005555;//set PD0-PD7 as outputs

GPIOD->OTYPER &= ~0x000000FF;

GPIOD->PUPDR &= ~0x0000FFFF;

// Port E config, control port

GPIOE->MODER |= 0x00000155;set PE0-PE5 as outputs

GPIOE->OTYPER &= ~0x0000001F;

GPIOE->PUPDR &= ~0x000003FF;

}//TFT\_IO\_init

void TFT\_init()

{

GPIOE->ODR &= ~0x00000008; // CS LOW

GPIOE->ODR |= 0x00000004; // RD HIGH

GPIOE->ODR &= ~0x00000000; // WR LOW

GPIOE->ODR &= ~0x00000010; // RES LOW (start hard reset)

delay\_ms(5);

delay\_ms(10);

GPIOE->ODR |= 0x00000010; // RES HIGH (finish hard reset)

delay\_ms(100);

delay\_ms(10);

TFT\_writeCMD(0x01); // soft\_reset

delay\_ms(10);

delay\_ms(10);

TFT\_writeCMD(0x01); // soft\_reset

delay\_ms(10);

// Start and Lock PLL

delay\_ms(10);

TFT\_writeCMD(0xE0);

TFT\_writeDATA(0x01); //Enable PLL

TFT\_writeCMD(0xE0);

TFT\_writeDATA(0x01); //PLL is locked as system clock

// Configure settings (datasheet page 44)

TFT\_writeCMD(0xB0);

TFT\_writeDATA(0x08); //24bit mode, disable dithering, enable FRC, active low syncs

TFT\_writeDATA(0x80); //TTL mode, TFT mode

TFT\_writeDATA(0x01); //high byte of horizontal screen size

TFT\_writeDATA(0xDF); //low byte of horizontal screen size (479 pixels)

TFT\_writeDATA(0x01); //high byte of vertical screen size

TFT\_writeDATA(0x0F); //low byte of vertical screen size (271)

TFT\_writeDATA(0x00); //RGB sequence

// More settings (datasheet page 78)

TFT\_writeCMD(0xF0);

TFT\_writeDATA(0x00); //set 8 bit pixel interface

// More settings (datasheet page 76)

TFT\_writeCMD(0xE6);

TFT\_writeDATA(0x01); //high byte of pixel clock frequency

TFT\_writeDATA(0x45); //middle byte of pixel clock frequency

TFT\_writeDATA(0x47); //low byte of pixel clock frequency (83271)

//pixel clock is now 100MHz \* (83271+1)/2^20 = 7.941MHz

// More settings (datasheet page 47)

TFT\_writeCMD(0xB4);

TFT\_writeDATA(0x02); //Setting horizontal sync total period

TFT\_writeDATA(0x0D); //set to 525 pixels

TFT\_writeDATA(0x00); //setting non-display period

TFT\_writeDATA(0x2B); //set to 43 pixels

TFT\_writeDATA(0x28); //setting horizontal sync pulse to 40 pixels

TFT\_writeDATA(0x00); //setting start position to 0

TFT\_writeDATA(0x00);

TFT\_writeDATA(0x00);

// More settings (datasheet page 49)

TFT\_writeCMD(0xB6);

TFT\_writeDATA(0x01); //setting total vertical sync total

TFT\_writeDATA(0x1d); //set to 285 pixels

TFT\_writeDATA(0x00); //setting the non-display period

TFT\_writeDATA(0x0C); //set to 12 pixels

TFT\_writeDATA(0x09); //setting vertical pulse width to 9 pixels

TFT\_writeDATA(0x00); //starting location to 0

TFT\_writeDATA(0x00);

// More settings (datasheet page 30)

TFT\_writeCMD(0x2A);

TFT\_writeDATA(0x00); //setting column address start to 0

TFT\_writeDATA(0x00);

TFT\_writeDATA(0x01); //setting column address end to 479

TFT\_writeDATA(0xDF);

// More settings (datasheet page 30)

TFT\_writeCMD(0x2B);

TFT\_writeDATA(0x00); //setting row address start to 0

TFT\_writeDATA(0x00);

TFT\_writeDATA(0x01); //setting row address end to 271

TFT\_writeDATA(0x0F);

// More settings (datasheet page 27)

TFT\_writeCMD(0x13); //enter normal mode

// More settings (datasheet page 40)

TFT\_writeCMD(0x38); //exit idle mode

// More settings (datasheet page 29)

TFT\_writeCMD(0x29); //set display on

}//TFT\_init

void TFT\_writeCMD(char COMMAND)

{

//This function writes data to the LCD controller that is interpreted as data

GPIOE->ODR &= ~0x00000008; // CS LOW

GPIOE->ODR &= ~0x00000001; // Bring RS LOW

GPIOE->ODR &= ~0x00000002; // Bring WR LOW

GPIOD->ODR = COMMAND; // Fill port D with COMMAND

GPIOE->ODR |= 0x00000008; // CS HIGH

GPIOE->ODR |= 0x00000002; // Bring WR HIGH

GPIOE->ODR |= 0x00000001; // Bring RS HIGH

}//TFT\_writeCMD

void TFT\_writeDATA(char DATA)

{

//This function writes data to the LCD controller that is interpreted as a command

GPIOE->ODR &= ~0x00000008; // CS LOW

GPIOE->ODR |= 0x00000001; // Bring RS HIGH

GPIOE->ODR &= ~0x00000002; // Bring WR LOW

GPIOD->ODR = DATA; // Fill port D with DATA

GPIOE->ODR |= 0x00000008; // CS HIGH

GPIOE->ODR |= 0x00000002; // Bring WR HIGH

}//TFT\_writeDATA

void TFT\_windowSet(int start\_x, int end\_x, int start\_y, int end\_y)

{

//This function sets the area that can be written to using the start/ending x/y values

TFT\_writeCMD(0x2A); //setting horizontal area

TFT\_writeDATA((start\_x)>>8);//high byte of start x

TFT\_writeDATA(start\_x); //low byte of start x

TFT\_writeDATA((end\_x)>>8); //high byte of end x

TFT\_writeDATA(end\_x); //low byte of end x

TFT\_writeCMD(0x2B); //setting horizontal area

TFT\_writeDATA((start\_y)>>8);//high byte of start y

TFT\_writeDATA(start\_y); //low byte of start y

TFT\_writeDATA((end\_y)>>8); //high byte of end y

TFT\_writeDATA(end\_y); //low byte of end y

}//TFT\_windowSet

void TFT\_fill\_pixel(int x,int y, int color)

{

//this function fills a particular pixel at the x/y position with the specified color

TFT\_fill\_area(x,x,y,y,color);

}//TFT\_fill\_pixel

void TFT\_fill\_area(int start\_x, int end\_x, int start\_y, int end\_y, unsigned int color )

{

//this function fills a rectangle specified by the start and end x/y's with the

//desired color

int i;

char RED = (color>>16)&0xFF; //mask off red byte from color

char GREEN = (color>>8) &0xFF; //mask off green byte from color

char BLUE = (color>>0) &0xFF; //mask off blue byte from color

TFT\_windowSet(start\_x+startOffset,end\_x+startOffset,start\_y,end\_y);

TFT\_writeCMD(0x2C);

for(i=0; i<(end\_x-start\_x+1)\*(end\_y-start\_y+1); i++)//write (height+1)\*(width+1) times

{

TFT\_writeDATA(RED); //write red byte

TFT\_writeDATA(GREEN);//write green byte

TFT\_writeDATA(BLUE); //write blue byte

}//for i

}//TFT\_fill\_area

void TFT\_fillScreen(char RED, char GREEN, char BLUE)

{

//sets area to entire screen and write the specified color to the screen

int i;

TFT\_windowSet(0x0000, 0x01DF, 0x0000, 0x010F);//set area to entire screen

TFT\_writeCMD(0x2C);

for(i=0; i<130560; i++)

{

TFT\_writeDATA(RED);

TFT\_writeDATA(GREEN);

TFT\_writeDATA(BLUE);

}//for i

}//TFT\_fillScreen

--------------------------------**LCD.c**----------------------------------------

--------------------------------**Game.h**---------------------------------------

#include <stdbool.h> // for boolean types

#ifndef \_\_GAME\_H\_

#define \_\_GAME\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Type Definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//this enum is defined as masks for port E's IDR to check whether a certain input was pressed

//for example if GPIOE->IDR & primary == primary the primary button was pressed

typedef enum \_buttons { primary = 0x0080,

secondary = 0x0100,

down = 0x0200,

left = 0x0400,

up = 0x0800,

right = 0x1000 } Button;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Defintions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

bool check\_win(void);

void apply\_tile\_effects(Button direction);

bool check\_legal\_move(Button direction);

void input\_handler(void);

void break\_blocks(void);

bool check\_user\_input(void);

void game\_button\_init(void);

bool check\_win(void);

#endif //\_\_GAME\_H\_

--------------------------------**Game.h**---------------------------------------

--------------------------------**Game.c**---------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stm32f4xx.h"

#include "Game.h"

#include "Map.h"

#include "LCD.h" //for delay function

//button check order:

//PE7 - primary button

//PE8 - secondary button

//PE9 - down

//PE10 - left

//PE11 - up

//PE12 - right

bool check\_win(void)

{

//goes through every tiles and sees if there are no breakable tiles or unvisited tiles

//if there are not then the player wins, otherwise they continue the game

int i,j;

for(i=0; i<map\_height; i++)

{

for(j=0; j<map\_width; j++)

{

//if any movable tile isnt visited return false

if( (map[i][j]==breakableBlock) || (map[i][j]==unvisitedTile) ){return false;}

}//for j

}//for i

return true;//all tiles that could be visited were visited so they won

}//check\_win

bool check\_user\_input(void)

{

//create mask to only check input values, if value isn't 0 after masking an input signal is

//still high in the IDR

const int mask = primary | secondary | down | left | up | right;

return (( GPIOE->IDR & mask )>0);//check if at least one input value is high in the IDR

}//check\_user\_input

void game\_button\_init(void)

{

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOEEN;

GPIOE->MODER &= 0xFC003FFF;//setting PE7-12 as inputs

GPIOE->PUPDR |= 0x02AA8000;//setting these buttons as pull down

GPIOE->MODER |= 0x04000000;//PE 13 set as output to control whether the backlight

//is enabled or not

}//game\_button\_init

bool check\_legal\_move(Button direction)

{

//this function will tell if the given input is a valid move and should be carried out or not

//this enforces that the player cannot go off the screen and cannot move into breakable tiles

int x = playerX;

int y = playerY;

//checking for illegal moves, such as moving off screen or into breakable block

if(direction==down)

{

if( (playerY == map\_height-1) || (map[y+1][x]==breakableBlock)){return false;}

}

else if(direction==left)

{

if( (playerX == 0) || (map[y][x-1]==breakableBlock) ){return false;}

}

else if(direction==up)

{

if( (playerY == 0) || (map[y-1][x]==breakableBlock)){return false;}

}

else if(direction==right)

{

if( (playerX == map\_width-1) || (map[y][x+1]==breakableBlock)){return false;}

}

return true;

}//check\_legal\_move

void input\_handler(void)

{

//this function checks what buttons were pressed by checking the IDR and executes

//the first button press it finds

if( (GPIOE->IDR & primary ) == primary )

{

break\_blocks();//calls function to break blocks around player

}//primary

else if( (GPIOE->IDR & secondary) == secondary )

{

GPIOE->ODR ^= (1<<13);//toggle state of backlight control pin

//enabling or disabling the backlight

}//secondary

else if( (GPIOE->IDR & down) == down )

{

if(!check\_legal\_move(down) ){return;}//illegal move

playerY++;//change coordinate accordingly

apply\_tile\_effects(down);

}//down

else if( (GPIOE->IDR & left) == left )

{

if(!check\_legal\_move(left) ){return;}//illegal move

playerX--;//change coordinate accordingly

apply\_tile\_effects(left);

}//left

else if( (GPIOE->IDR & up) == up )

{

if(!check\_legal\_move(up) ){return;}//illegal move

playerY--;//change coordinate accordingly

apply\_tile\_effects(up);

}//up

else if( (GPIOE->IDR & right ) == right)

{

if(!check\_legal\_move(right) ){return;}//illegal move

playerX++;//change coordinate accordingly

apply\_tile\_effects(right);

}//right

//wait until all input lines are dropped to allow only one input per button press

while( check\_user\_input() ){}

delay\_ms(10);//delay on ms to debounce end of signal

local\_refresh();//locally refresh the screen

}//input\_handler

void break\_blocks(void)

{

//this checks the 4 tiles to the left, right, top, and bottom of the player and turns

//breakable blocks into unvisited blocks if there are any in the 4 checked areas

int x = playerX;

int y = playerY;

//these check if the tile exists and if it a breakable tile before changing tile

if( (playerY!=map\_height-1) && (map[y+1][x]==breakableBlock) )

{

map[y+1][x]=unvisitedTile; //change tile if both conditions met

}//down

if( (playerX!=0) && (map[y][x-1]==breakableBlock) )

{

map[y][x-1]=unvisitedTile; //change tile if both conditions met

}//left

if( (playerY!=0) && (map[y-1][x]==breakableBlock) )

{

map[y-1][x]=unvisitedTile; //change tile if both conditions met

}//up

if( (playerX!=map\_width-1) && (map[y][x+1]==breakableBlock) )

{

map[y][x+1]=unvisitedTile; //change tile if both conditions met

}//right

}//break\_blocks

void apply\_tile\_effects(Button direction)

{

//this function is used to change unvisited tiles to visited tiles, and carry out the rules

//regarding movement onto visited tiles/ magma, and the directional aspects of directional

//tiles and ice tiles

int x,y;

bool change;

int currentTile;

do

{

//update the coordinated to check and reset change value

x = playerX;//used to tell current tile position

y = playerY;//used to tell current tile position

currentTile = map[y][x];//determines what action (if any) will be done

change = false;//used as flag to tell when to stop looping

local\_refresh();//locally refreshes the map before the action

if(currentTile==ice)

{

//if the tile is ice treat all the ices as if they were directional

//tiles of the way the player moved onto the ice

if (direction==down) {currentTile=downTile; }

else if(direction==left) {currentTile=leftTile; }

else if(direction==up) {currentTile=upTile; }

else if(direction==right){currentTile=rightTile;}

change = true;

}//ice

if(currentTile==visitedTile || currentTile==magma)

{

reset\_map();//they re-moved on a tile or fell into lava, restart game

}

else if(currentTile==unvisitedTile)

{

map[y][x]=visitedTile;//changing unvisited tile to visited tile

}

else if(currentTile==downTile)

{

//move player down one tile if legal move

if( !check\_legal\_move(down) ){return;}

playerY++;

change = true;

direction = down;

}//down directional tile

else if(currentTile==leftTile)

{

//move player to the left if legal move

if( !check\_legal\_move(left) ){return;}

playerX--;

change = true;

direction = left;

}//left directional tile

else if(currentTile==upTile)

{

//move player up if legal move

if( !check\_legal\_move(up) ){return;}

playerY--;

change = true;

direction = up;

}//up directional tile

else if(currentTile==rightTile)

{

//move player right if legal move

if( !check\_legal\_move(right) ){return;}

playerX++;

change = true;

direction = right;

}//right directional tile

} while( change );//do while no changes

}//apply\_tile\_effects

--------------------------------**Game.c**---------------------------------------

--------------------------------**Map.h**---------------------------------------

//for setting up and refreshing map

#ifndef \_\_MAP\_H\_

#define \_\_MAP\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Defines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#define startX 0 //starting x position of the player

#define startY 0 //starting y position of the player

#define map\_height 12 //map is 12 tiles high

#define map\_width 16 //map is 16 tiles wide

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Type Definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//This struct is used to store two colors and the 20x20 bit patter of the tile

//that will be used to display the tile on the map

typedef struct \_tile

{

int colors [2];//unsigned 32 bit integer, used to store color hexs

int pattern[20];//64 bit integer, used to store bit pattern for tiles

} Tile;

//this enum defines the indices of the tiles and where they will be stored in the array

//so tiles[ice] will give back the ice tile

typedef enum \_tileIndex { unvisitedTile = 0,

visitedTile = 1,

magma = 2,

ice = 3,

breakableBlock = 4,

upTile = 5,

downTile = 6,

leftTile = 7,

rightTile = 8,

characterTile = 9 } TileIndex;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void write\_tile(Tile tile, int start\_x, int start\_y);

void init\_tiles(void);

void refresh\_map(void); //probably also want to make local refresh

void local\_refresh(void);

void reset\_map(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Global Variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

extern int playerX, playerY;//global variable to hold player's coordinated

extern Tile tiles[10];//global array to hold the tiles

extern const int ref\_map[map\_height][map\_width];//unchangable map reference, holds indices

extern int map[map\_height][map\_width];//changable map, holds indices

#endif //\_\_MAP\_H\_

--------------------------------**Map.h**---------------------------------------

--------------------------------**Map.c**---------------------------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Header files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stm32f4xx.h"

#include <string.h> // for memcpy

#include "LCD.h"

#include "Map.h"

void reset\_map(void)

{

//This function is used to bring the map to its beginning state and start the level over

//resetting player coordinates

playerX = startX;

playerY = startY;

//resesting map to initial state by copying ref map to map

memcpy(map, ref\_map, sizeof(ref\_map) );

//setting the player start position as visited

map[playerY][playerX]=visitedTile;

//showing the map

refresh\_map();

}//reset\_state

void refresh\_map(void)

{

//this function will refresh the entire screen going left to right, top to bottom redisplaying

//tile based on indices that are stored in the map

int y,x;

for(y=0;y<12;y++)

{

for(x=0;x<16;x++)

{

if(playerY==y && playerX==x){write\_tile(tiles[characterTile],x\*20,y\*20);}

else {write\_tile(tiles[ map[y][x] ],x\*20,y\*20);}

}//for y

}//for x

}//refresh

void local\_refresh(void)

{

//this function checks if the 8 tiles surrounding the player exist and if they do it will

//redisplay that tile, and finally will redisplay the tile the player is at

int x,y;

x = playerX;

y = playerY;

if(x != 0)

{

write\_tile(tiles[ map[y][x-1] ],(x-1)\*20,(y+0)\*20);

}//left

if(y != 0)

{

write\_tile(tiles[ map[y-1][x] ],(x+0)\*20,(y-1)\*20);

}//up

if(x != map\_width-1)

{

write\_tile(tiles[ map[y][x+1] ],(x+1)\*20,(y+0)\*20);

}//right

if(y != map\_height-1)

{

write\_tile(tiles[ map[y+1][x] ],(x+0)\*20,(y+1)\*20);

}//down

if(x !=0 && y != 0)

{

write\_tile(tiles[ map[y-1][x-1] ],(x-1)\*20,(y-1)\*20);

}//upper left

if(x != map\_width-1 && y != 0)

{

write\_tile(tiles[ map[y-1][x+1] ],(x+1)\*20,(y-1)\*20);

}//upper right

if(x !=0 && y != map\_height-1)

{

write\_tile(tiles[ map[y+1][x-1] ],(x-1)\*20,(y+1)\*20);

}//lower left

if(x != map\_width-1 && y != map\_height-1)

{

write\_tile(tiles[ map[y+1][x+1] ],(x+1)\*20,(y+1)\*20);

}//lower right

write\_tile(tiles[characterTile],x\*20,y\*20);//unconditional tile refresh of character's tile

}//local\_refresh

void write\_tile(Tile tile, int start\_x, int start\_y)

{

//this function takes a tile reference and uses it bit pattern to send either its fill color

//or its design color. This is done by masking off level by level the 20 bits that are the design

//for that level, for each of the 20 levels of the tile. It uses the colors it finds to write

//the 20x20 pixel area starting at the start\_x and start\_y position

int i,j,index;

for(i=0;i<20;i++)

{

for(j=0;j<20;j++)

{

index = ( (tile.pattern[i]>>(j))&1 );//checking whether bit is 1 or 0

TFT\_fill\_pixel(start\_x+i,start\_y+j,tile.colors[index]);//filling pixel with color //at index 1 or 0

}//for i

}//for j

}//write\_tile

void init\_tiles(void)

{

//this function is used to set the colors and

//designs of the 10 tiles used to make the game map

// --Note that designs are flipped 90 degrees from what they are set here--

int x;

//format is:

//1). fill color

//2). detail color

//3). design pattern 0-19 (20bits x 20 indices)

x = unvisitedTile;//index is 0, solid black tile

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0x0d0d0d;//detail color: light black

tiles[x].pattern[0] = 0x00000;

tiles[x].pattern[1] = 0x00000;

tiles[x].pattern[2] = 0x00000;

tiles[x].pattern[3] = 0x00000;

tiles[x].pattern[4] = 0x00000;

tiles[x].pattern[5] = 0x00000;

tiles[x].pattern[6] = 0x00000;

tiles[x].pattern[7] = 0x00000;

tiles[x].pattern[8] = 0x00000;

tiles[x].pattern[9] = 0x00000;

tiles[x].pattern[10] = 0x00000;

tiles[x].pattern[11] = 0x00000;

tiles[x].pattern[12] = 0x00000;

tiles[x].pattern[13] = 0x00000;

tiles[x].pattern[14] = 0x00000;

tiles[x].pattern[15] = 0x00000;

tiles[x].pattern[16] = 0x00000;

tiles[x].pattern[17] = 0x00000;

tiles[x].pattern[18] = 0x00000;

tiles[x].pattern[19] = 0x00000;

x = visitedTile;//index 1, black tile with light green x

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0xcc00cc;//detail color: magenta

tiles[x].pattern[0] = 0x80001;

tiles[x].pattern[1] = 0x40002;

tiles[x].pattern[2] = 0x20004;

tiles[x].pattern[3] = 0x10008;

tiles[x].pattern[4] = 0x08010;

tiles[x].pattern[5] = 0x04020;

tiles[x].pattern[6] = 0x02040;

tiles[x].pattern[7] = 0x01080;

tiles[x].pattern[8] = 0x00900;

tiles[x].pattern[9] = 0x00600;

tiles[x].pattern[10] = 0x00600;

tiles[x].pattern[11] = 0x00900;

tiles[x].pattern[12] = 0x01080;

tiles[x].pattern[13] = 0x02040;

tiles[x].pattern[14] = 0x04020;

tiles[x].pattern[15] = 0x08010;

tiles[x].pattern[16] = 0x10008;

tiles[x].pattern[17] = 0x20004;

tiles[x].pattern[18] = 0x40002;

tiles[x].pattern[19] = 0x80001;

x = magma;//red tile with yellow diagonal lines

tiles[x].colors[0] = 0xff0000;//fill color : red

tiles[x].colors[1] = 0xffff00;//detail color: yellow

tiles[x].pattern[0] = 0x00000;

tiles[x].pattern[1] = 0x11004;

tiles[x].pattern[2] = 0x20822;

tiles[x].pattern[3] = 0x40440;

tiles[x].pattern[4] = 0x02200;

tiles[x].pattern[5] = 0x04010;

tiles[x].pattern[6] = 0x20820;

tiles[x].pattern[7] = 0x10448;

tiles[x].pattern[8] = 0x08204;

tiles[x].pattern[9] = 0x00002;

tiles[x].pattern[10] = 0x00840;

tiles[x].pattern[11] = 0x21020;

tiles[x].pattern[12] = 0x42010;

tiles[x].pattern[13] = 0x00102;

tiles[x].pattern[14] = 0x00204;

tiles[x].pattern[15] = 0x10440;

tiles[x].pattern[16] = 0x22020;

tiles[x].pattern[17] = 0x41010;

tiles[x].pattern[18] = 0x00008;

tiles[x].pattern[19] = 0x00000;

x = ice;//aqua colored tile with white diagonal lines

tiles[x].colors[0] = 0x0099ff;//fill color : aqua

tiles[x].colors[1] = 0xFFFFFF;//detail color: grey

tiles[x].pattern[0] = 0x00000;

tiles[x].pattern[1] = 0x08004;

tiles[x].pattern[2] = 0x04802;

tiles[x].pattern[3] = 0x00400;

tiles[x].pattern[4] = 0x40200;

tiles[x].pattern[5] = 0x20000;

tiles[x].pattern[6] = 0x10000;

tiles[x].pattern[7] = 0x00020;

tiles[x].pattern[8] = 0x00810;

tiles[x].pattern[9] = 0x00400;

tiles[x].pattern[10] = 0x00200;

tiles[x].pattern[11] = 0x40000;

tiles[x].pattern[12] = 0x20000;

tiles[x].pattern[13] = 0x02008;

tiles[x].pattern[14] = 0x01004;

tiles[x].pattern[15] = 0x00802;

tiles[x].pattern[16] = 0x40400;

tiles[x].pattern[17] = 0x20000;

tiles[x].pattern[18] = 0x10020;

tiles[x].pattern[19] = 0x00010;

x = breakableBlock;//brown tile with grey x and grey outline

tiles[x].colors[0] = 0x663300;//fill color : brown

tiles[x].colors[1] = 0x2d2d1f;//detail color: grey

tiles[x].pattern[0] = 0xFFFFF;

tiles[x].pattern[1] = 0xC0003;

tiles[x].pattern[2] = 0xA0005;

tiles[x].pattern[3] = 0x90009;

tiles[x].pattern[4] = 0x88011;

tiles[x].pattern[5] = 0x84021;

tiles[x].pattern[6] = 0x82041;

tiles[x].pattern[7] = 0x81081;

tiles[x].pattern[8] = 0x80901;

tiles[x].pattern[9] = 0x80601;

tiles[x].pattern[10] = 0x80601;

tiles[x].pattern[11] = 0x80901;

tiles[x].pattern[12] = 0x81081;

tiles[x].pattern[13] = 0x82041;

tiles[x].pattern[14] = 0x84021;

tiles[x].pattern[15] = 0x88011;

tiles[x].pattern[16] = 0x90009;

tiles[x].pattern[17] = 0xA0005;

tiles[x].pattern[18] = 0xC0003;

tiles[x].pattern[19] = 0xFFFFF;

x = upTile;//black tile with gold upwards facing arrow

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0xff9900;//detail color: gold

tiles[x].pattern[0] = 0x00000;

tiles[x].pattern[1] = 0x00000;

tiles[x].pattern[2] = 0x00000;

tiles[x].pattern[3] = 0x00000;

tiles[x].pattern[4] = 0x00000;

tiles[x].pattern[5] = 0x00010;

tiles[x].pattern[6] = 0x00018;

tiles[x].pattern[7] = 0x0001C;

tiles[x].pattern[8] = 0xFFFFE;

tiles[x].pattern[9] = 0xFFFFF;

tiles[x].pattern[10] = 0xFFFFF;

tiles[x].pattern[11] = 0xFFFFE;

tiles[x].pattern[12] = 0x0001C;

tiles[x].pattern[13] = 0x00018;

tiles[x].pattern[14] = 0x00010;

tiles[x].pattern[15] = 0x00000;

tiles[x].pattern[16] = 0x00000;

tiles[x].pattern[17] = 0x00000;

tiles[x].pattern[18] = 0x00000;

tiles[x].pattern[19] = 0x00000;

x = downTile;//black tile with blue downwards facing arrow

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0x0033cc;//detail color: blue

tiles[x].pattern[0] = 0x00000;

tiles[x].pattern[1] = 0x00000;

tiles[x].pattern[2] = 0x00000;

tiles[x].pattern[3] = 0x00000;

tiles[x].pattern[4] = 0x00000;

tiles[x].pattern[5] = 0x08000;

tiles[x].pattern[6] = 0x18000;

tiles[x].pattern[7] = 0x38000;

tiles[x].pattern[8] = 0x7FFFF;

tiles[x].pattern[9] = 0xFFFFF;

tiles[x].pattern[10] = 0xFFFFF;

tiles[x].pattern[11] = 0x7FFFF;

tiles[x].pattern[12] = 0x38000;

tiles[x].pattern[13] = 0x18000;

tiles[x].pattern[14] = 0x08000;

tiles[x].pattern[15] = 0x00000;

tiles[x].pattern[16] = 0x00000;

tiles[x].pattern[17] = 0x00000;

tiles[x].pattern[18] = 0x00000;

tiles[x].pattern[19] = 0x00000;

x = leftTile;//black tile with red leftwards facing arrow

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0xff0000;//detail color: red

tiles[x].pattern[0] = 0x00600;

tiles[x].pattern[1] = 0x00F00;

tiles[x].pattern[2] = 0x01F80;

tiles[x].pattern[3] = 0x03FC0;

tiles[x].pattern[4] = 0x07FE0;

tiles[x].pattern[5] = 0x00F00;

tiles[x].pattern[6] = 0x00F00;

tiles[x].pattern[7] = 0x00F00;

tiles[x].pattern[8] = 0x00F00;

tiles[x].pattern[9] = 0x00F00;

tiles[x].pattern[10] = 0x00F00;

tiles[x].pattern[11] = 0x00F00;

tiles[x].pattern[12] = 0x00F00;

tiles[x].pattern[13] = 0x00F00;

tiles[x].pattern[14] = 0x00F00;

tiles[x].pattern[15] = 0x00F00;

tiles[x].pattern[16] = 0x00F00;

tiles[x].pattern[17] = 0x00F00;

tiles[x].pattern[18] = 0x00F00;

tiles[x].pattern[19] = 0x00F00;

x = rightTile;//black tile with green rightwards facing arrow

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0x006600;//detail color: green

tiles[x].pattern[0] = 0x00F00;

tiles[x].pattern[1] = 0x00F00;

tiles[x].pattern[2] = 0x00F00;

tiles[x].pattern[3] = 0x00F00;

tiles[x].pattern[4] = 0x00F00;

tiles[x].pattern[5] = 0x00F00;

tiles[x].pattern[6] = 0x00F00;

tiles[x].pattern[7] = 0x00F00;

tiles[x].pattern[8] = 0x00F00;

tiles[x].pattern[9] = 0x00F00;

tiles[x].pattern[10] = 0x00F00;

tiles[x].pattern[11] = 0x00F00;

tiles[x].pattern[12] = 0x00F00;

tiles[x].pattern[13] = 0x00F00;

tiles[x].pattern[14] = 0x00F00;

tiles[x].pattern[15] = 0x07FE0;

tiles[x].pattern[16] = 0x03FC0;

tiles[x].pattern[17] = 0x01F80;

tiles[x].pattern[18] = 0x00F00;

tiles[x].pattern[19] = 0x00600;

x = characterTile;//a black tile with a white colored smilely face

tiles[x].colors[0] = 0x0d0d0d;//fill color : light black

tiles[x].colors[1] = 0xffffff;//detail color: white

tiles[x].pattern[0] = 0x00000;

tiles[x].pattern[1] = 0x00000;

tiles[x].pattern[2] = 0x00000;

tiles[x].pattern[3] = 0x00000;

tiles[x].pattern[4] = 0x00000;

tiles[x].pattern[5] = 0x07000;

tiles[x].pattern[6] = 0x08000;

tiles[x].pattern[7] = 0x103FC;

tiles[x].pattern[8] = 0x20000;

tiles[x].pattern[9] = 0x20000;

tiles[x].pattern[10] = 0x20000;

tiles[x].pattern[11] = 0x20000;

tiles[x].pattern[12] = 0x103FC;

tiles[x].pattern[13] = 0x08000;

tiles[x].pattern[14] = 0x07000;

tiles[x].pattern[15] = 0x00000;

tiles[x].pattern[16] = 0x00000;

tiles[x].pattern[17] = 0x00000;

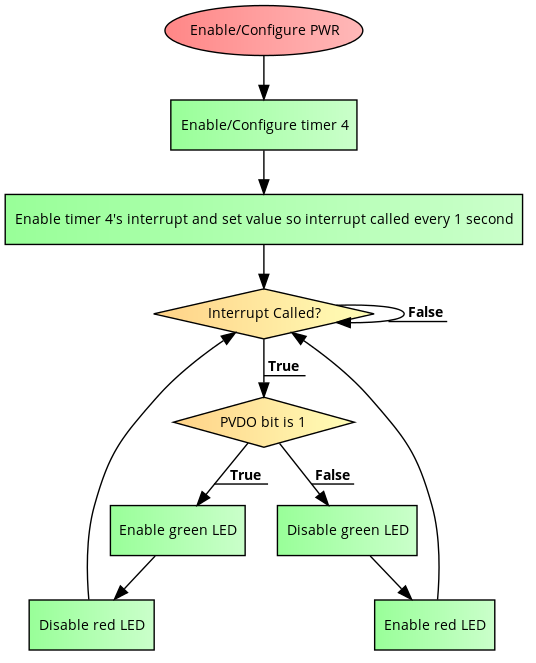
tiles[x].pattern[18] = 0x00000;

tiles[x].pattern[19] = 0x00000;

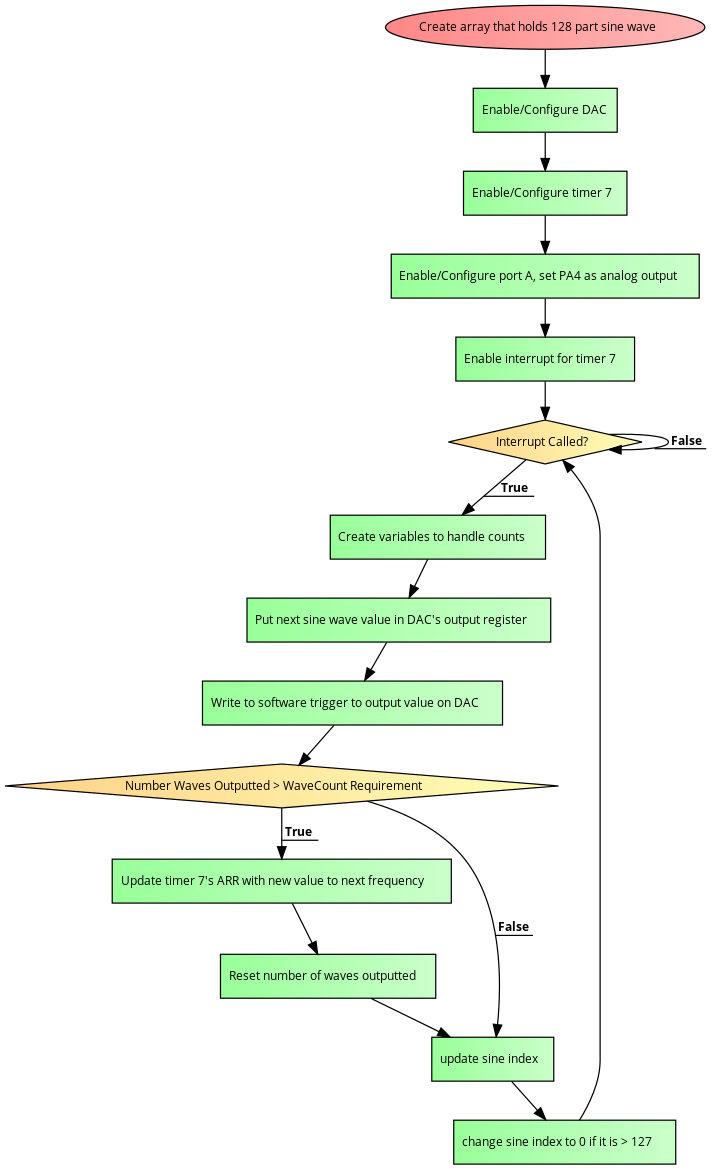
}//init\_tiles

--------------------------------**Map.c**-------------------

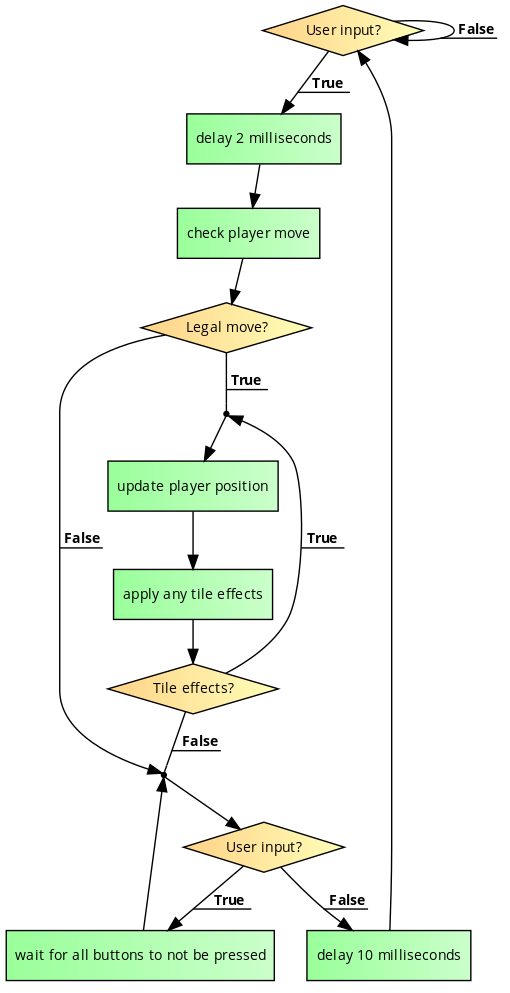
Appendix E - Flowcharts for code

****

Power Status Logic

****

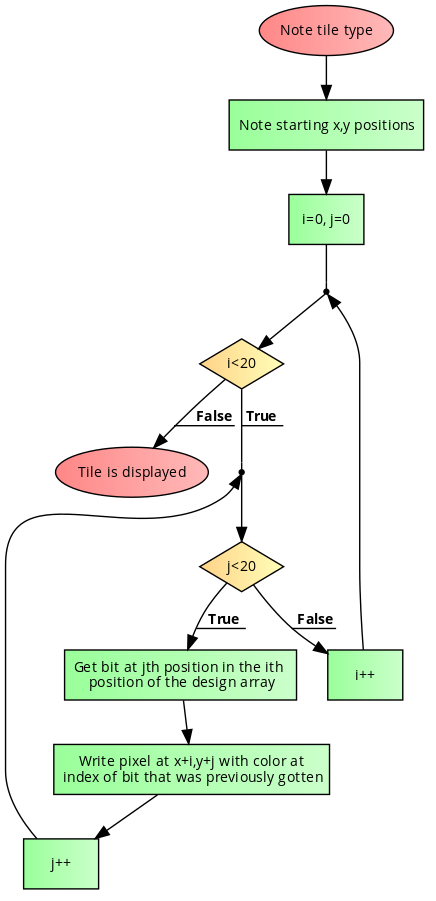
Audio Output Logic



User Input and Basic Game Logic



Logic for Applying Tile Effects



Logic for Displaying a Tile

Appendix F - Game Description

The game that we will first implement on our system consists of a tile based system were the goal of the player is to move once, and only once, across all solid black tiles on the screen putting ‘X’ marks on them. The player will control their movement by the inputs give on the D-pad of the device, and inputs on the buttons will also impact the game. Whenever the player moves, if they are allowed to move into that tile, the tile they moved onto will be marked with a green ‘X’ if it was a black tile before, and the player icon will moved to the appropriate position. The player will not be able to move off the screen, and any movement that would move them off screen will be ignored. In each level, in addition to the solid black tiles, there will be magma tiles, ice tiles, breakable tiles, and direction tiles. Magma tiles are tiles that the player can move into but once they enter they ‘die’ and must restart the level from the start again. Ice tiles are tiles the player can move into but will continue moving the direction they entered the tile from until they hit a non-ice tile. Breakable tiles are tiles the player cannot move into, but by pressing a primary button when next to a breakable tile the player can ‘break’ them and turn them into a normal black tile. Directional tiles are moveable tiles that when the player enters them they must go the direction the arrow points. If a player moves into a tile with a ‘X’ in it they must restart the level from the start position again.



Example screen from the game

In this example of how the game will work, the red tiles are magma, the blue tiles are ice, the orange tiles are breakable tiles, the yellow arrows are direction tiles, and the grey tiles are the immovable tiles. The player in this example would start on the tile outline in light-green dots.

Appendix G - Mechanical Drawings of System Enclosure