Midterm Project

Gerri Fox

fox.ge@northeastern.edu

Submit date: 10/26/20

Due Date: 10/29/20

**Abstract**

This is a lab report for the Embedded Design class Fall 2020 midterm. This project has two components: a tone generator to generate tones from C4 to C6, and a textual based and interactive game of Snakes and Ladders. Each of these are done using the DE1-SoC board processor and the speaker which is connected to the board and programmed to receive output as well as has an interactive component via the command line with the user. This project involved doing research into programming the DE1-SoC board with a brand new device.

# Introduction

This project has two main components: the first is a tone generator that can play tones C4-C6 continuously and based on user input and the second is a digital game of snakes and ladders. The tone generator is built using the DE1-SoC board and a speaker connected to the board through wires. The speaker is programmed to produce output to generate the tones. The game of Snakes and Ladders is a textual based graphical user interface that allows for a user to input to play the game against the computer. This project assumes knowledge in programming the De1-SoC board and C++.

# Software and Hardware used

* DE1-SoC board
* 2 wires
* 1 speaker
* UART to USB cord
* Power adapter
* MacBook Laptop
* Macbook Terminal

# Lab Steps

1. Connect the speaker to the board via the D0 pin and one of the ground pins.
2. Connect the DE1-SoC board to a laptop via the UART to USB cord.
3. Use screen to connect the board to Terminal.
4. Use the nano editor to write each c++ file to be saved to the board.
5. Run the “make” command to run the Makefile and compile each file.
6. Start the program and choose from the main menu to either start the tone generator or play snakes and ladders.

# Lab Discussion

Overall Design:

I choose an object-oriented design to design this project. I set up my code in .h and .cpp files creating a class to represent each of the components of the midterm. I have a speaker class to represent all of the functionality needed regarding the speaker. And I have a game class to implement all of the functionality needed to run the Snakes and Ladders game. In my .h file I set up all of the variables and methods needed for the project and implement all of the methods in the .cpp file. I have my game class derived from my speaker class as the game utilizes the speaker’s functionality. I then have a main method that sets up the main menu and includes the .h file to access the methods and calls them accordingly based on the option the user selects. I also have a Makefile to compile all of the files conveniently and create an out file to run the program easily.

Tone Generator:

For the tone generator portion, first I worked on getting a singular A4 tone. To achieve this, firstly I initialize the base to write to the DE1-SoC board. To be able to generate tones from C4 to C6, I have set up the respective frequencies and stored them in an array. I have a while loop where the condition is always true for the tone to be continuous, then, based on the note the user inputted, I use the array to determine the frequency and establish the counter which is 200 MHz divided by the respective frequency then divided by 2. Then, I have to write to the direction register in order to set the pins in the JP1 base address to 1 which means they are outputs. Technically, the speaker only uses the D0 pin but I chose to write all the pins to 1 just incase the connection is messed with accidentally. Then, I load the counter to the offset and write 3 to the control offset and start the value to write to the base as 0 as in order to get a tone we use oscillation between 0 and 1. Then, I have chosen an arbitrary number to set as the loop size to determine how long the tones will play for continuously, and each loop I use XOR to change the value to write to from 0 to 1 and visa versa. The program has the ability to take user input and then if it finds the tone it will change the note otherwise it will remain the same as the previous. If the user inputs 0, then the tone generator will stop. When using the timer, it is important to remember the original data written at the timer load, control and interrupt offsets so when we are done using the timer, we write the original values back to avoid having any lag or freezing.

I also created a second tone generator that incorporates the switches on the DE1-SoC board. This interface allows for a more visual, hands on “digital” piano and can still play tones C4 through C6. Starting from the leftmost switch to the rightmost switch, the leftmost one represents the on/off switch. Tones will only play when that switch is in the on position. The next switch represents the lower octave of notes, thus the C4-B4 notes and the next switch represents the upper octave in this case which is C5-B5. To get a C6 tone, the user must have both of the octave switches in the on position. The remaining switches represent the letters of each note from “A” to “G” in alphabetical order. So for example, to play a F4, the user must have the leftmost switch on to allow for tone, the next leftmost switch on to signify the lower octave, and then the second to last switch to the right on to signify the note “F.” This generator starts with playing an A4 tone and plays sound continuously. It switches the tone when the user has a valid switch combination. If at any point the switches are invalid, the tone generator continues playing the most previous valid tone.

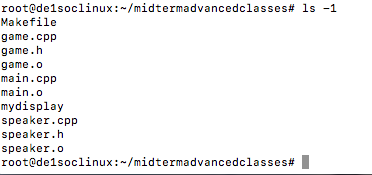
Snakes and Ladders:

For the snakes and ladders portion, I split the game into functions based on each aspect that needs to occur. For example, I have a function to set up the board, print the board, roll the dice, move each player, generate the equation and print the equation. This design was best as it keeps every function having one purpose and is able to easily be put into another function that carries through the entire game. I chose to represent the board as a string array with each element representing a space on the board. The board either shows its respective space number or has “P” signifying where the player/user is or “C” signifying where the computer is. The board has 2 ladders and 2 snakes which are also represented textually. The rolling of the dice is executed through a random function that gets a number between 1 and 6 each time. The game is executed through adding the dice roll to the players current position and advancing forward by that number of spaces unless a ladder or snake is landed on in which the player either boosts forward or goes backward by that many number of spaces on the board. The game is over when either player lands exactly on space 50. The user has the ability to get a power up after each turn by answering a simple math equation, which is also generated randomly. If the user answers the question correctly, they get an extra turn. The game also integrates the speaker as everytime the user answers a question correctly or if the user wins, the speaker plays a tone.

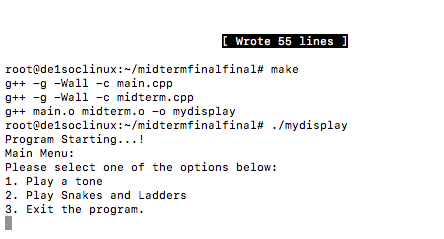
I also created multiple difficulty levels for this game. The above is considered the easy version, then I created a hard version. This version is exactly the same as above except the computer’s dice is randomized to roll either a 4, 5 or 6 to allow for the computer to advance faster. Once the computer reaches space 46 on the board however, then the dice reverts back to a normal dice as it would be impossible for the computer to win without going over the final spot of 50.

This game is executed through “using namespace std” as it utilizes strings and arrays to represent data, as well as the “cout” command to print the game to the textual user interface and “cin” to make the game interactive and read input from the user. I have also split the game into specific functions representing a functionality to keep it easily readable.

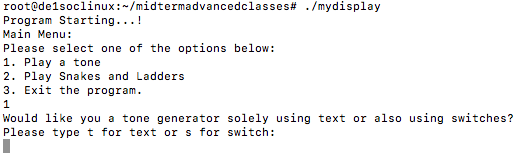
# Results



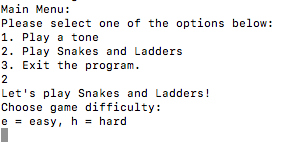
This photo shows the organization of my code into classes and files and all of the files that calling “make” conveniently and easily creates. It creates the runnable program mydisplay, which by calling “./mydisplay” is what starts the program execution.



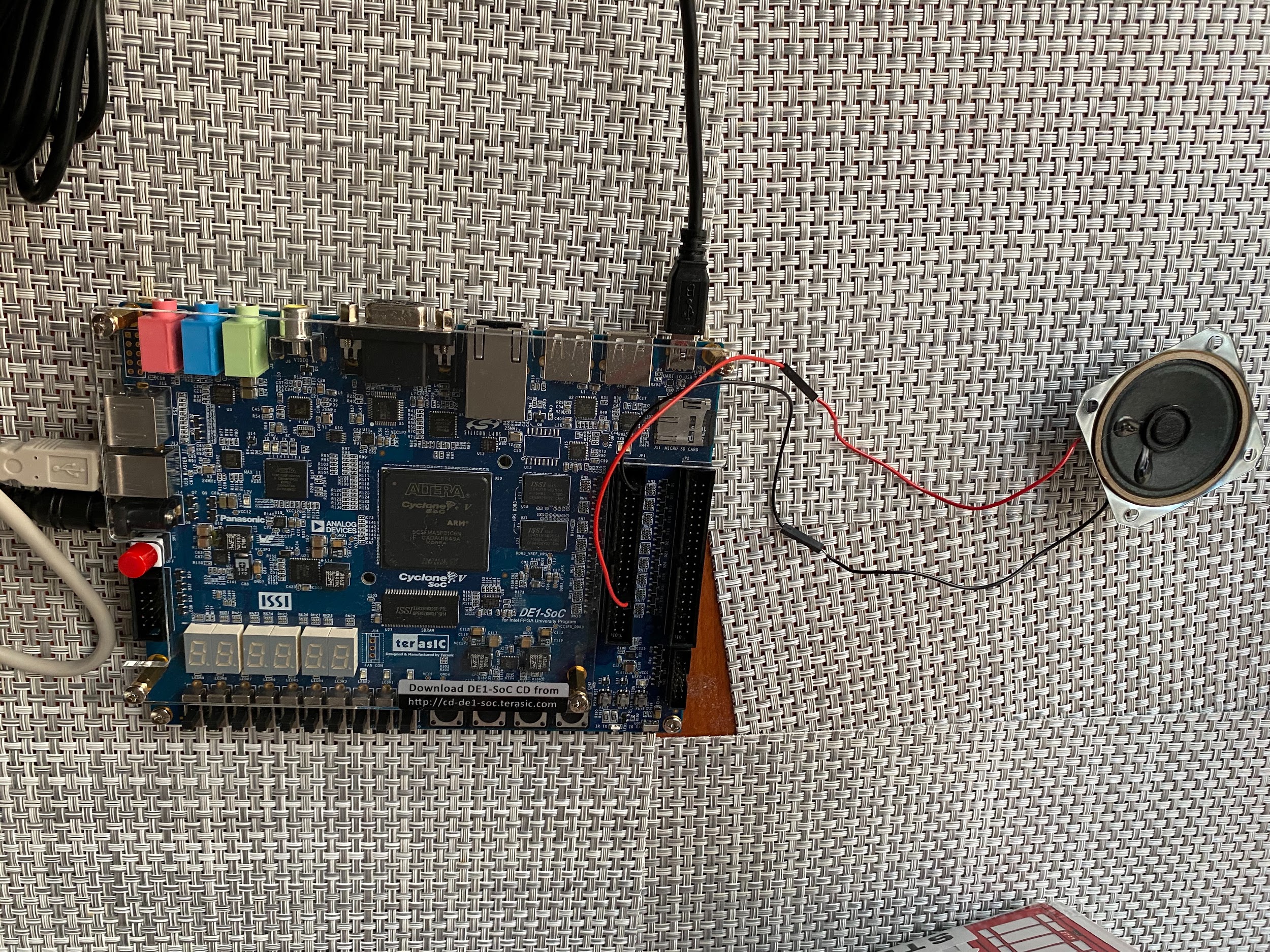
This is a photo of the textual menu provided to the user when they run the program. The user can access the tone generator or play the snakes and ladders game from one easy, central location.

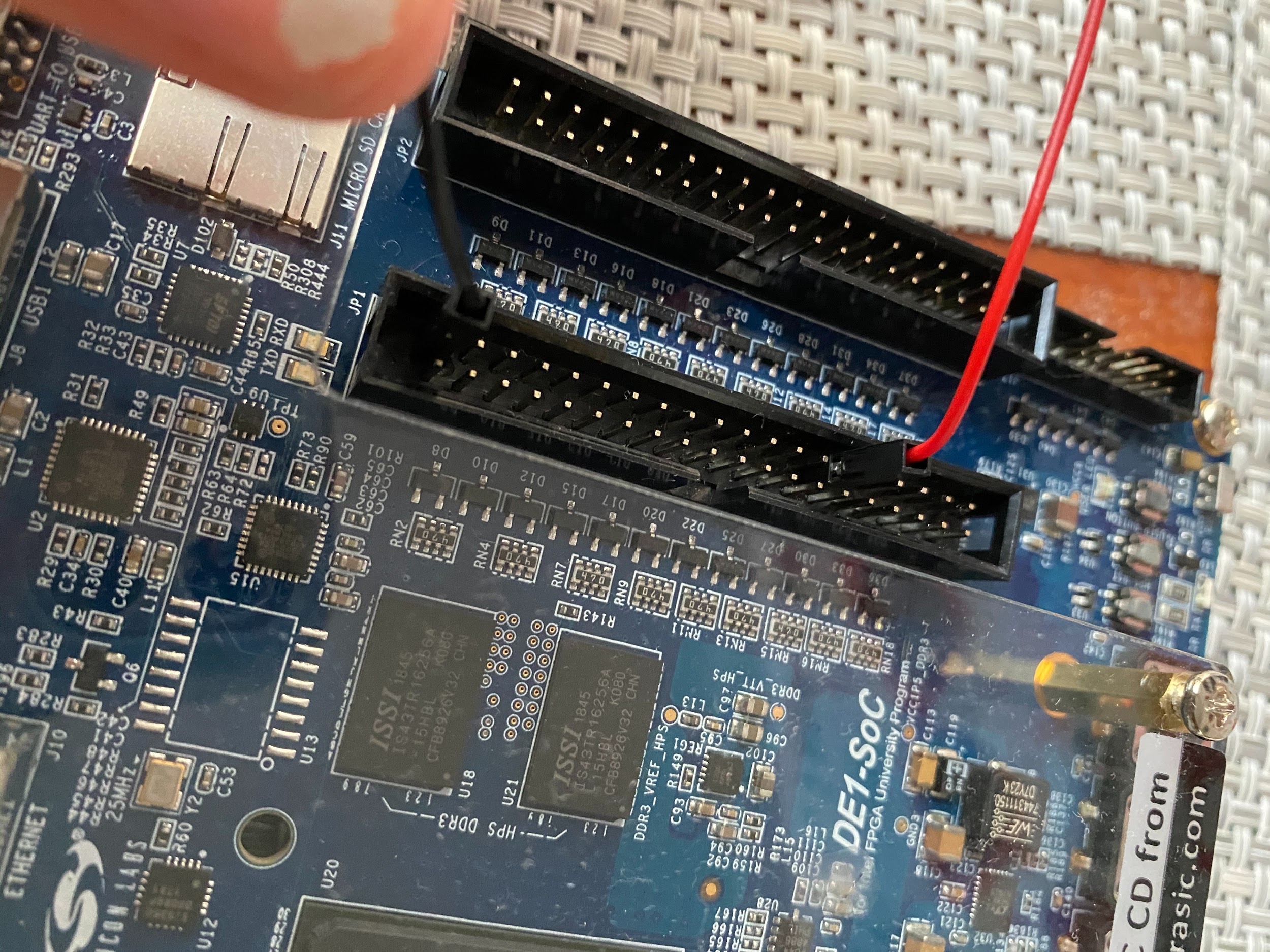


This photo shows the next option within the main menu to select which tone generator to start.

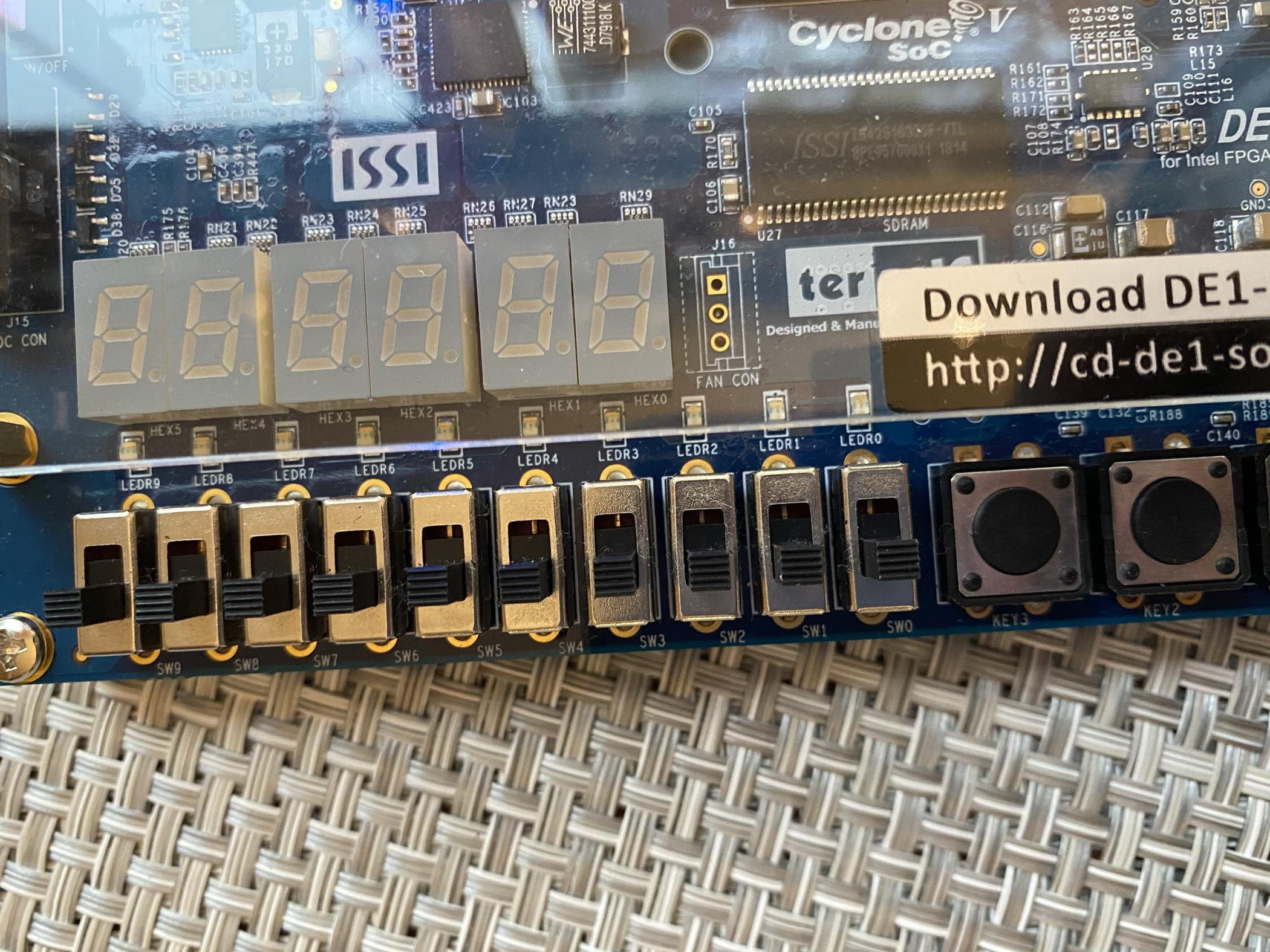


This photo shows the next option within the main menu to select which level of difficulty to play the game of Snakes and Ladders with.

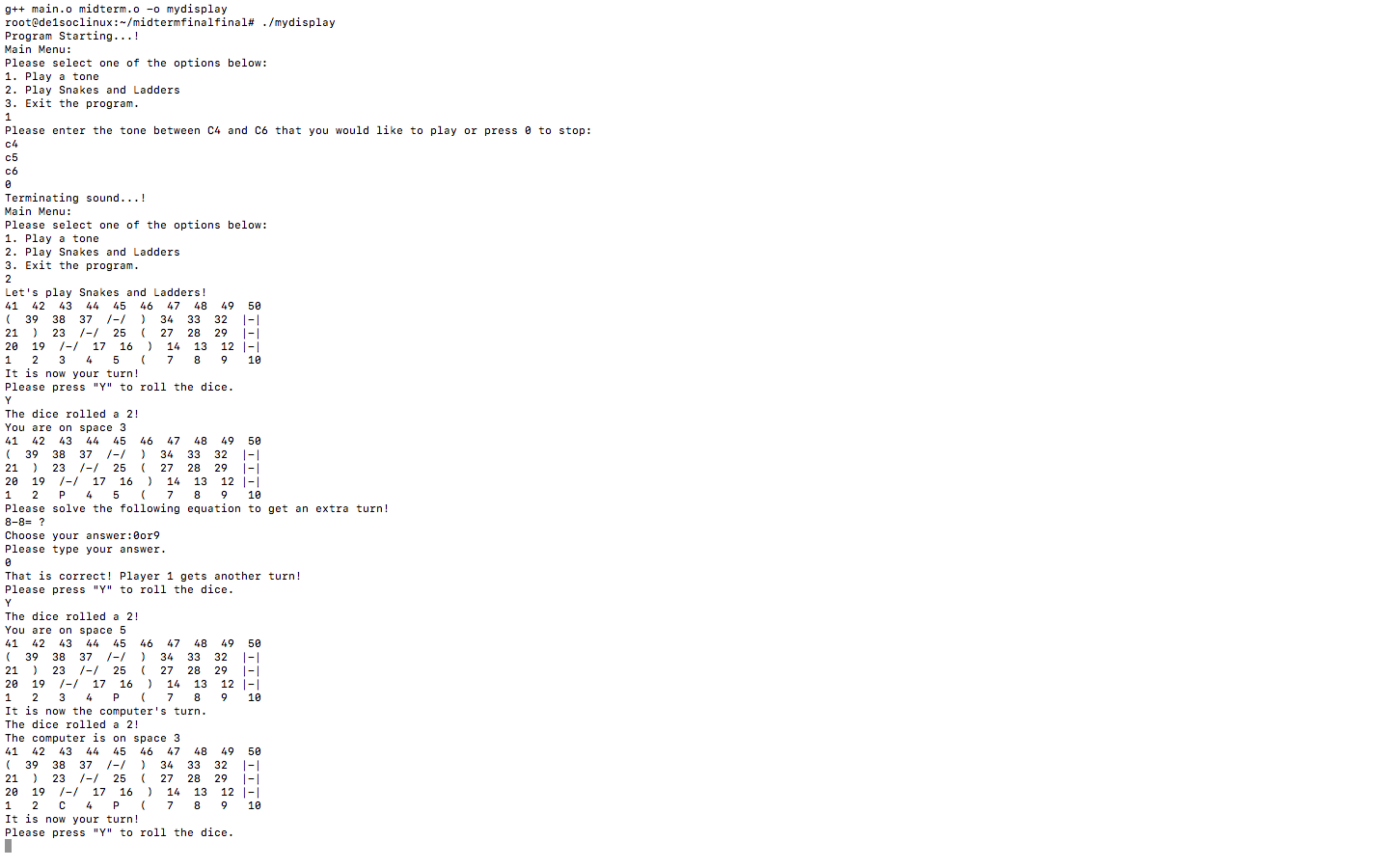




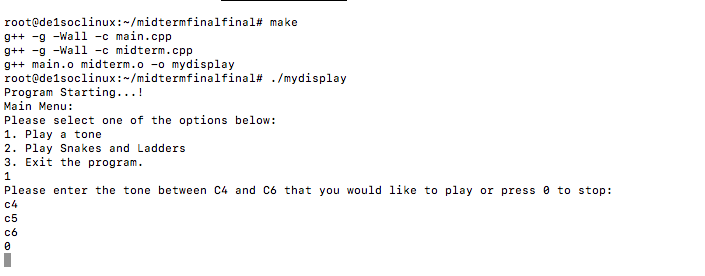
This is a photo of the speaker wired to the DE1-SoC board with one wire connected to the D0 pin on the JP1 base and the other connected to one of the ground pins.



This photo shows an up close photo of the switches in order to explain the use of each in my second tone generator version. SW9 represents the on/off switch. Tones will only play if this is switched on. SW8 represents the first octave of notes, from C4-B4. SW7 represents the second octave of notes, from C5-B5. And to play a C6, both switches must be switched on. SW6-SW0 represent the letters of each note, so letters “A” through “G” respectively.



This is a picture of the textual based graphical user interface for the game of snakes and ladders. This shows each space numbered and shows “P” to represent the user and “C” to represent the computer as well as shows the ladders and snakes on the board. It also shows the prompt for the user to roll the dice and answer the equation and shows the computer taking a turn.



This is a photo of the inputs for the tone generator to show it listening to user inputs and playing the tones inputted continuously and in order.

# Analysis

This project was a very great experience in learning more about programming using the DE1-SoC board and creating a program geared for the user. It was a great experience creating something with the user experience in mind that embeds not only just a command line program but also through the DE1-SoC board to add more features. The project overall turned out to be successful in getting a deeper understanding of programming using the DE1-SoC board for example with figuring out bases and addresses versus offsets and writing to registers to set pins as outputs.

# Conclusion

This project is relevant in the real world as it introduces young kids into the world of programming and provides another form of entertainment that is not a regular board game or electronic device. It is a great way to get kids interested and involved in programming as often kids are not aware of what the field is and the opportunities that lie within it.

# References

1. Prof. Julius Marpaung, “*Lab Report Guide*”, Northeastern University, January 6 2020.

[2] Terasic, “DE1-SoC User Manual”, Terasic.com, January 28, 2019.

# Appendix

Link to video demonstration and explanation: <https://youtu.be/wsXXBG2WBQs>

**speaker.h**

//

// Created by Gerri Fox on 10/26/20.

//

#ifndef MIDTERMCOMBINED\_SPEAKER\_H

#define MIDTERMCOMBINED\_SPEAKER\_H

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <iostream>

using namespace std;

// Physical base address of FPGA Devices

const unsigned int LW\_BRIDGE\_BASE = 0xFF200000; // Base offset

// Length of memory-mapped IO window

const unsigned int LW\_BRIDGE\_SPAN = 0x00DEC700; // Address map size/

// / Cyclone V FPGA device addresses

const unsigned int LEDR\_BASE = 0x00000000; // Leds offset

const unsigned int SW\_BASE = 0x00000040; // Switches offset

const unsigned int KEY\_BASE = 0x00000050; // Push buttons offset

//0xFFFEC600 -0xFF200000 = 0xDEC600

const unsigned int MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET = 0xDEC600;

// Points to LOAD Registerconst

const unsigned int MPCORE\_PRIV\_TIMER\_COUNTER\_OFFSET = 0xDEC604;

// Points to COUNTER Registerconst

const unsigned int MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET = 0xDEC608;

// Points to CONTROL Registerconst

const unsigned int MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET = 0xDEC60C;

// Points to INTERRUPT Register

const unsigned int JP1 = 0x00000060; //speaker offset

const unsigned int DIRECTION\_BIT = 0x00000064; //direction bit offset

//frequencies for tones C4 through C6

const unsigned int tones[15] = {261, 293, 329, 349, 392, 440, 493, 523, 587, 659, 698, 783, 880, 987, 1046};

/\*\*

\* A class to represent the functionality of the speaker.

\*/

class speaker {

public:

/\*\* Write a 4-byte value at the specified general-purpose I/O location. \*

\* @param pBase

\* Base address returned by 'mmap'.

\* @parem offset Offset where device is mapped.

\* @param value Value to be written.

\* \*/

void RegisterWrite(char \*pBase, unsigned int reg\_offset, int value);

/\*\* Read a 4-byte value from the specified general-purpose I/O location.

\* @param pBase Base address returned by 'mmap'.

\* @param offset Offset where device is mapped.

\* @return Value read. \*/

int RegisterRead(char \*pBase, unsigned int reg\_offset);

/\*\* Initialize general-purpose I/O

\* - Opens access to physical memory /dev/mem

\* - Maps memory into virtual address space

\* @param fd File descriptor passed by reference, where the result

\* of function 'open' will be stored.

\* @return Address to virtual memory which is mapped to physical, or MAP\_FAILED onerror.

\* \*/

char \*Initialize(int \*fd);

/\*\* \* Close general-purpose I/O.

\* \* @param pBase Virtual address where I/O was mapped.

\* @param fd File descriptor previously returned by 'open'. \*/

void Finalize(char \*pBase, int fd);

/\*\*

\* Plays tones on the speaker based on user input. The tone generator starts with the inputted tone

\* and plays tones continuously listening for user input to change the tone.

\* @param note the tone to start the tone generator with

\*/

void playTone(string note);

/\*\*

\* Plays tones on the speaker based on user controlling the switches.

\* The tone generator plays a continuous tone based on the switches. The switch to the leftmost, controls

\* the start and stopping of the sound. The next two, control what octave, so the second leftmost corresponds

\* to notes C4-B4 and the next one C5-B5. The remaining switches represent the notes a-g. To play, a C6, the user

\* must switch both of the octave switches. If a switch combination is invalid, the program will continuously play

\* the previous valid input. The tone generator will continue until the start/stop switch is used and all switches

\* are in position 0. The program will start out playing an A4 if just the start switch is switched.

\*/

void playTone();

/\*\*

\* Plays one tone on the speaker to use in the Snakes and ladders game.

\*/

void playOneTone();

/\*\*

\* Plays a happy tone at the end of the snakes and ladders game signifying the user won or the user landed

\* on a ladder or the computer landed on a snake.

\*/

void playHappyTone();

/\*\*

\* Plays a sad tone at the end of the snakes and ladders game signifying the computer won the game or

\* the user landed on a snake or the computer landed on a ladder.

\*/

void playSadTone();

/\*\*

\* Plays one sad tone for when the user answers the math question incorrectly.

\*/

void playOneSadTone();

};

#endif //MIDTERMCOMBINED\_SPEAKER\_H

**speaker.cpp**

//

// Created by Gerri Fox on 10/26/20.

//

#include "speaker.h"

/\*\* Write a 4-byte value at the specified general-purpose I/O location. \*

\* @param pBase

\* Base address returned by 'mmap'.

\* @parem offset Offset where device is mapped.

\* @param value Value to be written.

\* \*/

void speaker::RegisterWrite(char \*pBase, unsigned int reg\_offset, int value) {

\*(volatile unsigned int \*) (pBase + reg\_offset) = value;

}

/\*\* \* Read a 4-byte value from the specified general-purpose I/O location.

\* \* \* @param pBase Base address returned by 'mmap'.

\* \* @param offset Offset where device is mapped.

\* \* @return Value read. \*/

int speaker::RegisterRead(char \*pBase, unsigned int reg\_offset) {

return \*(volatile unsigned int \*) (pBase + reg\_offset);

}

/\*\* \* Initialize general-purpose I/O

\* \* - Opens access to physical memory /dev/mem

\* \* - Maps memory into virtual address space

\* \* @param fd File descriptor passed by reference, where the result

\* \* of function 'open' will be stored.

\* \* @return Address to virtual memory which is mapped to physical, or MAP\_FAILED onerror.

\* \*/

char \*speaker::Initialize(int \*fd) {

// Open /dev/mem to give access to physical addresses

\*fd = open("/dev/mem", (O\_RDWR | O\_SYNC));

if (\*fd == -1) // check for errors in openning /dev/mem

{

cout << "ERROR: could not open /dev/mem..." << endl;

exit(1);

} // Get a mapping from physical addresses to virtual addresses

char \*virtual\_base = (char \*) mmap(NULL, LW\_BRIDGE\_SPAN, (PROT\_READ | PROT\_WRITE), MAP\_SHARED, \*fd, LW\_BRIDGE\_BASE);

if (virtual\_base ==

MAP\_FAILED) // check for errors

{

cout << "ERROR: mmap() failed..." << endl;

close(\*fd); // close memory before exiting

exit(1); // Returns 1 to the operating system;

}

return virtual\_base;

}

/\*\* \* Close general-purpose I/O.

\* \* \* @param pBase Virtual address where I/O was mapped.

\* \* @param fd File descriptor previously returned by 'open'. \*/

void speaker::Finalize(char \*pBase, int fd) {

if (munmap(pBase, LW\_BRIDGE\_SPAN) != 0) {

cout << "ERROR: munmap() failed..." << endl;

exit(1);

}

close(fd); // close memory}

}

void speaker::playTone() {

int fd;

char \*pBase = Initialize(&fd);

int freq = 440;

//original timer location values to write back at end

int originalLoad = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

int originalControl = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

int originalInterrupt = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

if (RegisterRead(pBase, SW\_BASE) == 0000000000) {

cout << "Please switch leftmost switch to start sound." << endl;

cout << "Once you do, you will be prompted back to main menu to "

"restart tone generator." << endl;

}

while (RegisterRead(pBase, SW\_BASE) == 0000000000) {

//wait for user to start sound

}

while (RegisterRead(pBase, SW\_BASE) != 0) {

int switches = RegisterRead(pBase, SW\_BASE); //get updated switch value

switch (switches) {

//start sound with A4 as default

case 512: //all cases converted to decimal

freq = 440;

break;

//A4

case 832:

freq = 440;

break;

//B4

case 800:

freq = 493;

break;

//C4

case 784:

freq = 261;

break;

//D4

case 776:

freq = 293;

break;

//E4

case 772:

freq = 329;

break;

//F4

case 770:

freq = 349;

break;

//G4

case 769:

freq = 392;

break;

//A5

case 704:

freq = 880;

break;

//B5

case 672:

freq = 987;

break;

//C5

case 656:

freq = 523;

break;

//D5

case 648:

freq = 587;

break;

//E5

case 644:

freq = 659;

break;

//F5

case 642:

freq = 698;

break;

//G5

case 641:

freq = 783;

break;

//C6

case 912:

freq = 1046;

break;

//invalid note

default:

//does nothing as it will keep playing previous valid tone continuously until another valid tone

//is entered or the user stops

break;

}

int counter = (200000000 / freq) / 2; //convert 200 MHz to Hz

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1s to the direction bit to set the pins to output

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load the timer offset

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load the control offset

int entervalue = 0x000000; //start the value at 0 this starts the oscillation

while (RegisterRead(pBase, SW\_BASE) == switches) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //switch the value being written to continue the oscillation

RegisterWrite(pBase, JP1, entervalue); //write to the JP1 address

}

}

}

//write back original values to timer addresses after done using

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, originalLoad);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, originalControl);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, originalInterrupt);

cout << "Terminating sound...!" << endl;

Finalize(pBase, fd);

}

void speaker::playTone(string note) {

// Initialize

int fd;

char \*pBase = Initialize(&fd);

int freq;

//represents the notes that the tone generator can recognize

string notes[15] = {"c4", "d4", "e4", "f4", "g4", "a4", "b4", "c5", "d5", "e5", "f5", "g5", "a5", "b5", "c6"};

int loop = 0; //starts the loop

//original timer location values to write back at end

int originalLoad = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

int originalControl = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

int originalInterrupt = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

//continuous loop to get the tones to be continuous

while (true) {

//finds the frequency based on the inputted tone

for (int i = 0; i < 15; i++) {

if (note == notes[i]) {

freq = i;

}

}

int counter = (200000000 / tones[freq]) / 2; //convert 200 MHz to Hz

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1s to the direction bit to set the pins to output

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load the timer offset

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load the control offset

int entervalue = 0x000000; //start the value at 0 this starts the oscillation

while (loop < 5000) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //switch the value being written to continue the oscillation

RegisterWrite(pBase, JP1, entervalue); //write to the JP1 address

loop++;

}

}

//get input from the user to continuously play a tone

string name;

getline(cin, name);

note = name;

loop = 0;

if (note == "0") { //if the user inputs 0 then break and end the tone generator

break;

}

}

//write back original values to timer addresses after done using

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, originalLoad);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, originalControl);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, originalInterrupt);

cout << "Terminating sound...!" << endl;

Finalize(pBase, fd);

}

void speaker::playOneTone() {

int fd;

char \*pBase = Initialize(&fd);

int loop = 0;

//original timer location values to write back at end

int originalLoad = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

int originalControl = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

int originalInterrupt = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

int counter = (200000000 / 1046) / 2; //convert 200 MHz to Hz, just play a single C6 tone

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

int entervalue = 0x000000; //start value

while (loop < 1000) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

//write back original values to timer addresses after done using

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, originalLoad);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, originalControl);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, originalInterrupt);

Finalize(pBase, fd);

}

void speaker::playHappyTone() {

int fd;

char \*pBase = Initialize(&fd);

int loop = 0;

//original timer location values to write back at end

int originalLoad = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

int originalControl = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

int originalInterrupt = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

int counter = (200000000 / 1046) / 2; //convert 200 MHz to Hz, just play a single C6 tone

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

int entervalue = 0x000000; //start value

while (loop < 1000) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

loop = 0;

counter = (200000000 / 783) / 2; //convert 200 MHz to Hz

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

loop = 0;

counter = (200000000 / 880) / 2; //convert 200 MHz to Hz

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

loop = 0;

counter = (200000000 / 987) / 2; //convert 200 MHz to Hz

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

loop = 0;

counter = (200000000 / 1046) / 2; //convert 200 MHz to Hz

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

entervalue = 0x000000; //start value

while (loop < 1000) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

//write back original values to timer addresses after done using

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, originalLoad);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, originalControl);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, originalInterrupt);

Finalize(pBase, fd);

}

void speaker::playSadTone() {

int fd;

char \*pBase = Initialize(&fd);

int loop = 0;

//original timer location values to write back at end

int originalLoad = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

int originalControl = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

int originalInterrupt = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

int counter = (200000000 / 329) / 2; //convert 200 MHz to Hz, just play a single E4 tone

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

int entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

loop = 0;

counter = (200000000 / 293) / 2; //convert 200 MHz to Hz, just play a single D4 tone

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

loop = 0;

counter = (200000000 / 261) / 2; //convert 200 MHz to Hz, just play a single C4 tone

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

//write back original values to timer addresses after done using

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, originalLoad);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, originalControl);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, originalInterrupt);

Finalize(pBase, fd);

}

void speaker::playOneSadTone() {

int fd;

char \*pBase = Initialize(&fd);

int loop = 0;

//original timer location values to write back at end

int originalLoad = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET);

int originalControl = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET);

int originalInterrupt = RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET);

int counter = (200000000 / 261) / 2; //convert 200 MHz to Hz, just play a single C4 tone

RegisterWrite(pBase, DIRECTION\_BIT, 0xffffffff); //write 1 to direction register

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, counter); //load timer

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, 3); //load timer control offset

int entervalue = 0x000000; //start value

while (loop < 500) {

if (RegisterRead(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET) != 0) {

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, 1); // reset timer flagbit

entervalue = entervalue ^ 0x1; //oscillate to get tone

RegisterWrite(pBase, JP1, entervalue); //write to speaker output

loop++;

}

}

//write back original values to timer addresses after done using

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_LOAD\_OFFSET, originalLoad);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_CONTROL\_OFFSET, originalControl);

RegisterWrite(pBase, MPCORE\_PRIV\_TIMER\_INTERRUPT\_OFFSET, originalInterrupt);

Finalize(pBase, fd);

}

**game.h**

//

// Created by Gerri Fox on 10/26/20.

//

#ifndef MIDTERMCOMBINED\_GAME\_H

#define MIDTERMCOMBINED\_GAME\_H

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <iostream>

#include "speaker.h"

using namespace std;

//array for key spaces

const int keySpaces[4] = {10, 17, 34, 39};

//array for key of where to go after landing on a key space

const int destinationSpaces[4] = {30, 35, 5, 21};

//represents each space on the board for snakes and ladders

const string boardval[50] = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10",

"11", "12", "13", "14", "15", "16", "17", "18", "19", "20",

"21", "22", "23", "24", "25", "26", "27", "28", "29", "30",

"31", "32", "33", "34", "35", "36", "37", "38", "39", "40",

"41", "42", "43", "44", "45", "46", "47", "48", "49", "50"};

/\*\*

\* A class to represent the game of Snakes and Ladders.

\*/

class game : public speaker{

public:

int dice; //variable to represent number rolled on dice

string board[50]; //initialize board

//variables for moving players

string tempPlayer1Space;

string tempPlayer2Space;

//initialize player 1 and 2 locations

int player1;

int player2;

//math equation variables

int num1;

int num2;

char sign;

//math equation answer choices

int choice1;

int choice2;

int correctChoice; //keeps track of which choice is the correct choice

/\*\*

\* Rolls the dice by getting a random value from 0 to 6.

\*/

void rollDice();

/\*\*

\* Rolls the dice by getting a random value from 4-6. This is used to make the game harder to win for the user.

\*/

void rollDiceHard();

/\*\*

\* Prints the board to the console in it's proper grid showing numbers of each space

\* or a ladder or snake. Prints each space character with two spaces in between and

\* a newline after every 10 spaces. Board is printed so the numbers increase and a

\* player can weave through the board following the number order without

\* jumping.

\*/

void printBoard();

/\*\*

\* Moves player 1 based on the random dice roll. Resets the value of player 1

\* as well as updates the board to draw the position of player 1. Checks to see

\* if the player has landed on a key space (bottom of ladder or top of snake)

\* and moves player 1 accordingly if necessary. Prints the board if this happens to

\* show user where they first moved to.

\*/

void movePlayer1();

/\*\*

\* Moves player 2 based on the random dice roll. Resets the value of player 2

\* as well as updates the board to draw the position of player 2. Checks to see

\* if the player has landed on a key space (bottom of ladder or top of snake)

\* and moves player 2 accordingly if necessary. Prints the board if this happens to

\* show user where they first moved to.

\*/

void movePlayer2();

/\*\*

\* Gets two random numbers between 0 and 9 as well as uses a random value to set

\* the sign of the equation (either 0 for + or 1 for -). Makes sure that the equation

\* answer is one digit long, updates num2 until it is if it originally isn't.

\* Calls getEquationChoices() to set up the two multiple choice answers

\* the user can pick from.

\*/

void getEquation();

/\*\*

\* Gets two choices for the user to pick from to guess the equation answer.

\* One choice is the correct answer as determined by calling getEquationAnswer.

\* The choice is chosen by a random number, if it is 0, then it is choice 1, if it is 1,

\* then it is choice 2. Then, to get the next choice,

\* it chooses a random number.

\*/

void getEquationChoices();

/\*\*

\* Gets the answer of the equation using the randomly selected numbers and sign.

\*/

void getEquationAnswer();

/\*\*

\* Prints the equation to the console.

\*/

void printEquation();

/\*\*

\* Returns the right answer to the current equation.

\* @return the right answer to the current equation.

\*/

int getCorrectAns();

/\*\*

\* Sets up the board for snakes and ladders by setting the players at the beginning and establishing

\* where the snakes and ladders are set on the board.

\*/

void setUpBoard();

/\*\*

\* Plays the game of snakes and ladders against the computer.

\*/

void playGame();

};

#endif //MIDTERMCOMBINED\_GAME\_H

**game.cpp**

//

// Created by Gerri Fox on 10/26/20.

//

#include "game.h"

void game::setUpBoard() {

//initialize player 1 and 2 starting positions on board

tempPlayer1Space = "1";

tempPlayer2Space = "1";

//initialize player 1 and 2 locations

player1 = 0;

player2 = 0;

//set piece values

for (int i = 0; i < 50; i++) {

board[i] = boardval[i];

}

//assign ladders and snakes

//ladder straight up and down on right side of board

board[10] = "|-|";

board[29] = "|-|";

board[30] = "|-|";

//diagonal ladder

board[17] = "/-/";

board[23] = "/-/";

board[35] = "/-/";

//long curvy snake in middle

board[5] = "(";

board[14] = ")";

board[25] = "(";

board[34] = ")";

//straight snake

board[39] = "(";

board[21] = ")";

}

void game::printBoard() {

//print array with space numbers and snakes and ladders

for (int i = 40; i < 50; i++) {

cout << board[i] + " ";

}

cout << "\n";

for (int i = 39; i >= 30; i--) {

cout << board[i] + " ";

}

cout << "\n";

for (int i = 20; i < 30; i++) {

cout << board[i] + " ";

}

cout << "\n";

for (int i = 19; i >= 10; i--) { //print these backwards as to weave through board

if (i == 11) {

cout << board[i] + " ";

} else {

cout << board[i] + " ";

}

}

cout << "\n";

for (int i = 0; i < 10; i++) {

cout << board[i] + " "; //print each element value with 2 spaces in between

}

cout << "\n";

}

void game::rollDice() {

srand(time(NULL)); // initialize random seed

dice = rand() % 6 + 1; //get a new random dice roll between 1 and 6

cout << "The dice rolled a " << dice << "!" << endl;

}

void game::rollDiceHard() {

srand(time(NULL)); // initialize random seed

dice = rand() % 3 + 4; //get a new random dice roll between 4 and 6

cout << "The dice rolled a " << dice << "!" << endl;

}

void game::movePlayer1() {

if (player1 + 1 + dice > 50) {

cout << "You must roll land less than or exactly on space 50." << endl;

return; //do nothing as cannot go past end

} else {

board[player1] = tempPlayer1Space; //reset board back to normal value

player1 = player1 + dice; //move player based on dice roll

}

for (int k = 0; k < 4; k++) { //check is player has landed on the bottom of a ladder or top of a snake

if (player1 == keySpaces[k]) {

if (player1 == 10 || player1 == 17) {

cout << "You landed on a ladder!" << endl;

playHappyTone();

} else {

cout << "You landed on a snake :(" << endl;

playSadTone();

}

string temp = board[player1]; //get value at where player is about to move to

if (temp == "C") {

board[player1] = "PC"; //update board to show where the player has moved to

} else {

board[player1] = "P"; //update board to show where the player has moved to

}

printBoard(); //to show user updated space before either a snake or ladder

board[player1] = temp; //reset after displaying as player is about to move again

player1 = destinationSpaces[k]; //change player to end of either ladder or snake

cout << "You are now on space " << player1 + 1 << endl;

}

}

if (board[player1] == "C") { //if the computer is currently on the space

board[player1] = "PC";

tempPlayer1Space = boardval[player1]; //get value at where player is about to move to

} else {

tempPlayer1Space = board[player1]; //get value at where player is about to move to

board[player1] = "P"; //update board to show where the player has moved to

}

cout << "You are on space " << player1 + 1 << endl;

}

void game::movePlayer2() {

if (player2 + 1 + dice > 50) {

cout << "The computer must roll land less than or exactly on space 50." << endl;

return; //do nothing as cannot go past end

} else {

board[player2] = tempPlayer2Space; //reset board back to normal value

player2 = player2 + dice; //move player based on dice roll

}

for (int k = 0; k < 4; k++) { //check is player has landed on the bottom of a ladder or top of a snake

if (player2 == keySpaces[k]) {

if (player2 == 10 || player2 == 17) {

cout << "The computer landed on a ladder :(" << endl;

playSadTone();

} else {

cout << "The computer landed on a snake!" << endl;

playHappyTone();

}

string temp = board[player2]; //get value at where player is about to move to

if (temp == "P") {

board[player2] = "PC"; //update board to show where the player has moved to

} else {

board[player2] = "C"; //update board to show where the player has moved to

}

printBoard(); //to show user updated space before either a snake or ladder

board[player2] = temp; //reset after displaying as player is about to move again

player2 = destinationSpaces[k]; //change player to end of either ladder or snake

}

}

if (board[player2] == "P") { //if the computer is currently on the space

board[player2] = "PC";

tempPlayer2Space = boardval[player2]; //get value at where player is about to move to

} else {

tempPlayer2Space = board[player2]; //get value at where player is about to move to

board[player2] = "C"; //update board to show where the player has moved to

}

cout << "The computer is on space " << player2 + 1 << endl;

}

void game::getEquation() {

srand(time(NULL)); // initialize random seed

//get math question

num1 = rand() % 10; //random num between 0 and 9

num2 = rand() % 10; //random num between 0 and 9

if (rand() % 2 == 0) { //determine what sign for the equation

sign = '+';

while (num1 + num2 >= 10) { //if the answer is more than 1 digit

num2 = rand() % 10; //get new num random num between 0 and 9

}

} else {

sign = '-';

while (num1 - num2 < 0) { //if the answer is negative

num1 = rand() % 10; //get new num random num between 0 and 9

}

}

getEquationChoices(); //get answer choices

}

void game::getEquationChoices() {

getEquationAnswer();

if (rand() % 2 == 0) { //determines if choice 1 or 2 will be correct

choice1 = correctChoice;

choice2 = rand() % 10; //get a random answer for the other choice

while (choice2 == choice1) {

choice2 =

rand() % 10; //get another random answer for the other choice if the original equals the correct

}

} else {

choice2 = correctChoice;

choice1 = rand() % 10; //get a random answer for the other choice

while (choice1 == choice2) {

choice1 =

rand() % 10; //get another random answer for the other choice if the original equals the correct

}

}

}

void game::getEquationAnswer() {

int ans;

if (sign == '+') { //if equation uses addition

ans = num1 + num2;

} else { //if equation uses subtraction

ans = num1 - num2;

}

correctChoice = ans;

}

void game::printEquation() {

cout << "Please solve the following equation to get an extra turn!" << endl;

if (sign == '+') {

cout << num1 << "+" << num2 << "= ?" << endl;

} else {

cout << num1 << "-" << num2 << "= ?" << endl;

}

cout << "Choose your answer:" << choice1 << "or" << choice2 << endl;

}

int game::getCorrectAns() {

return correctChoice;

}

void game::playGame() {

cout << "Choose game difficulty: " << endl;

cout << "e = easy, h = hard" << endl;

string diff;

cin >> diff;

if (diff == "e") {

bool won = false;

setUpBoard();

//loop that keeps the game going until one player reaches the end (space 50 exactly)

while (!won) {

printBoard(); //print the board to the console

cout << "It is now your turn!" << endl;

bool loop = true;

while (loop) {

cout << "Please press \"Y\" to roll the dice." << endl;

char y;

cin >> y;

if (y == 'Y' || y == 'y') {

rollDice();

loop = false;

}

}

movePlayer1(); //move player 1 and set up equation

if (player1 == 49) { //player 1 (user) wins

won = true;

break;

}

printBoard();

getEquation(); //set up equation

printEquation(); //print equation

bool loop2 = true;

while (loop2) {

cout << "Please type your answer." << endl;

int ans;

cin >> ans;

if (ans == getCorrectAns()) {

cout << "That is correct! Player 1 gets another turn!" << endl;

playOneTone(); //play a tone when user answers question correct

bool loop3 = true;

while (loop3) {

cout << "Please press \"Y\" to roll the dice." << endl;

char y;

cin >> y;

if (y == 'Y' || y == 'y') {

rollDice();

loop3 = false;

}

}

movePlayer1();//player 1 gets another turn

printBoard(); //print the board

loop2 = false;

} else {

cout << "Sorry that answer is incorrect. The correct answer is " << correctChoice << endl;

playOneSadTone();

loop2 = false;

}

}

if (player1 == 49) { //player 1 (user) wins

won = true;

break;

}

cout << "It is now the computer's turn." << endl;

rollDice();

movePlayer2(); //move player 2

if (player2 == 49) { //player 2 (computer) wins

won = true;

}

}

//out of loop so one player must have won

if (player1 == 49) { //player 1 (user) wins

cout << "You win!" <<

endl;

playHappyTone();

} else { //player 2 (computer) wins

cout << "The computer won." <<

endl;

playSadTone();

}

} else if (diff == "h") {

//array for key spaces

bool won = false;

setUpBoard();

//loop that keeps the game going until one player reaches the end (space 50 exactly)

while (!won) {

printBoard(); //print the board to the console

cout << "It is now your turn!" << endl;

bool loop = true;

while (loop) {

cout << "Please press \"Y\" to roll the dice." << endl;

char y;

cin >> y;

if (y == 'Y' || y == 'y') {

rollDice();

loop = false;

}

}

movePlayer1(); //move player 1 and set up equation

if (player1 == 49) { //player 1 (user) wins

won = true;

break;

}

printBoard();

getEquation(); //set up equation

printEquation(); //print equation

bool loop2 = true;

while (loop2) {

cout << "Please type your answer." << endl;

int ans;

cin >> ans;

if (ans == getCorrectAns()) {

cout << "That is correct! Player 1 gets another turn!" << endl;

playOneTone(); //play a tone when user answers question correct

bool loop3 = true;

while (loop3) {

cout << "Please press \"Y\" to roll the dice." << endl;

char y;

cin >> y;

if (y == 'Y' || y == 'y') {

rollDice();

loop3 = false;

}

}

movePlayer1();//player 1 gets another turn

printBoard(); //print the board

loop2 = false;

} else {

cout << "Sorry that answer is incorrect. The correct answer is " << correctChoice << endl;

playOneSadTone();

loop2 = false;

}

}

if (player1 == 49) { //player 1 (user) wins

won = true;

break;

}

cout << "It is now the computer's turn." << endl;

if (player2 >= 46) { //computer will roll regular die if they are within 3 spaces of end

rollDice();

} else {

rollDiceHard(); //the computer will always roll 4,5,6

}

movePlayer2(); //move player 2

if (player2 == 49) { //player 2 (computer) wins

won = true;

}

}

//out of loop so one player must have won

if (player1 == 49) { //player 1 (user) wins

cout << "You win!" <<

endl;

playHappyTone();

} else { //player 2 (computer) wins

cout << "The computer won." <<

endl;

playSadTone();

}

} else {

cout << "Please enter a valid difficulty input." << endl;

}

}

**Makefile**

#Make File

mydisplay: main.o game.o speaker.o

g++ main.o game.o speaker.o -o mydisplay

main.o: main.cpp game.h speaker.h

g++ -g -Wall -c main.cpp

game.o: game.cpp game.h speaker.h

g++ -g -Wall -c game.cpp

speaker.o: speaker.cpp speaker.h

g++ -g -Wall -c speaker.cpp

clean:

rm main.o game.o speaker.o mydisplay

**main.cpp**

//

// Created by Gerri Fox on 10/23/20.

//

#include "game.h"

/\*\*

\* Main function to run the midterm project program.

\* @return exit

\*/

int main(void) {

game \*m = new game;

int ans = 0; //initialize user input variable

cout << "Program Starting...!" << endl;

while (ans != 3) { //keeps the main menu looking for input unless user quits

cout << "Main Menu:" << endl;

cout << "Please select one of the options below: " << endl;

cout << "1. Play a tone" << endl;

cout << "2. Play Snakes and Ladders" << endl;

cout << "3. Exit the program." << endl;

cin >> ans; //reads user input

string tone; //tone to start playing text based

string tchoice; //choice to choose text or switch tone generator

switch (ans) {

case 1:

cout << "Would like you a tone generator solely using text or also using switches?" << endl;

cout << "Please type t for text or s for switch:" << endl;

cin >> tchoice;

if (tchoice == "t") {

cout << "Please enter the tone between C4 and C6 that you would like to play or press 0 to stop:"

<< endl;

cin >> tone; //gets tone to start tone generator with

m->playTone(tone);

} else if (tchoice == "s") {

m->playTone();

} else {

cout << "Please select a valid option." << endl;

}

break;

case 2:

cout << "Let's play Snakes and Ladders!" << endl;

m->playGame();

break;

case 3:

//does nothing as will exit the while loop

break;

default:

cout << "Please select a valid option." << endl;

}

}

return 0; //exits the program

}