ECE 150 FINAL REPORT

Team Name: Blind Key
Team #: 52
Team Members: Amio Rahman, Faduma Ahmed, Samanvay Vajpayee

Table of Contents:

1. Overview

Advances in assisted technology allows those with visual impairments to do common tasks such as writing, browsing the Internet, typing, engaging in online chat, and reading documents. **[1]** Although a wide variety of devices are available, they come with a staggering price tag. The objective of this project is to create a cheaper alternative to the typical braille keyboard for the benefit of those who have visual impairments and cannot use a regular keyboard. Our braille input device allows the users to type by emulating the braille language as buttons. **[2]** The language itself consists of six raised dots in two parallel rows, where possible combinations correspond to different letters/symbols. To integrate this idea, the pins our onion omega will be connected onto a breadboard with 7 buttons (6 to emulate the braille language and 1 for the space buttons). The buttons are so arranged that the orientation of the buttons on the Breadboard is corresponding to the raised dots in the braille script. Users will be able to press different button and the corresponding letter/symbols will be printed onto a text file and would be displayed onto a screen.
Not only is this project useful for those with visual impairments, but it can be applicable in real life settings. For example, several at galleries such as Lakeshore an arts group [**3]** requires visitors to write a survey to get feedback about visitor accommodation. This form of input can allow those with visual impairments to input their ideas, in an easy and hassle-free manner. Since this project is low in cost, it also has the potential to be used by individuals who may not have as much to spend on such expensive products.

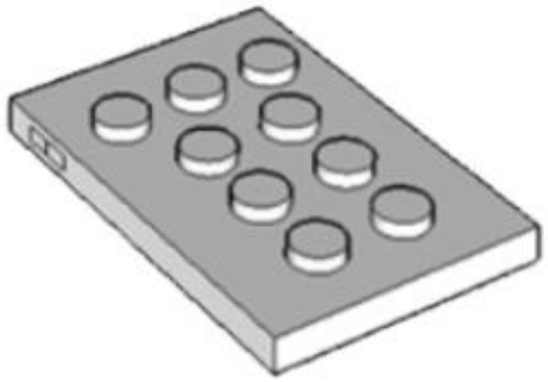

Figure 1: Sketch of proposed design

2. System Design

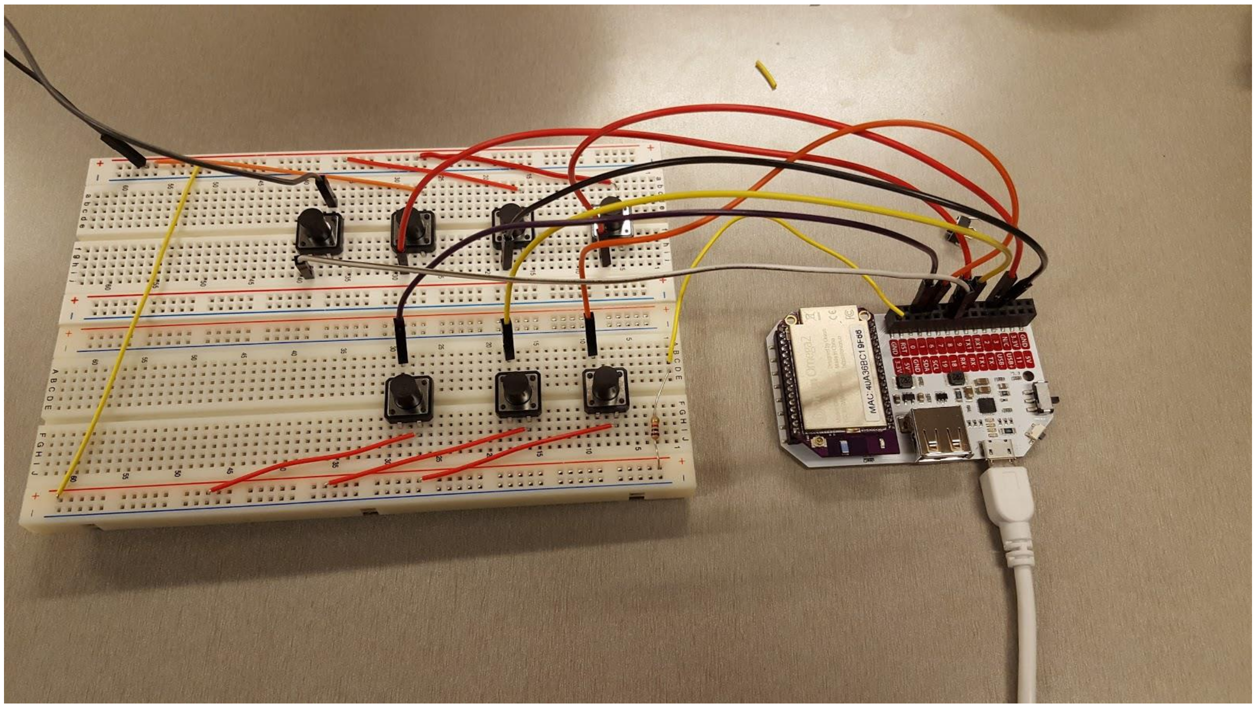

Figure 2: Picture of the hardware components of our project

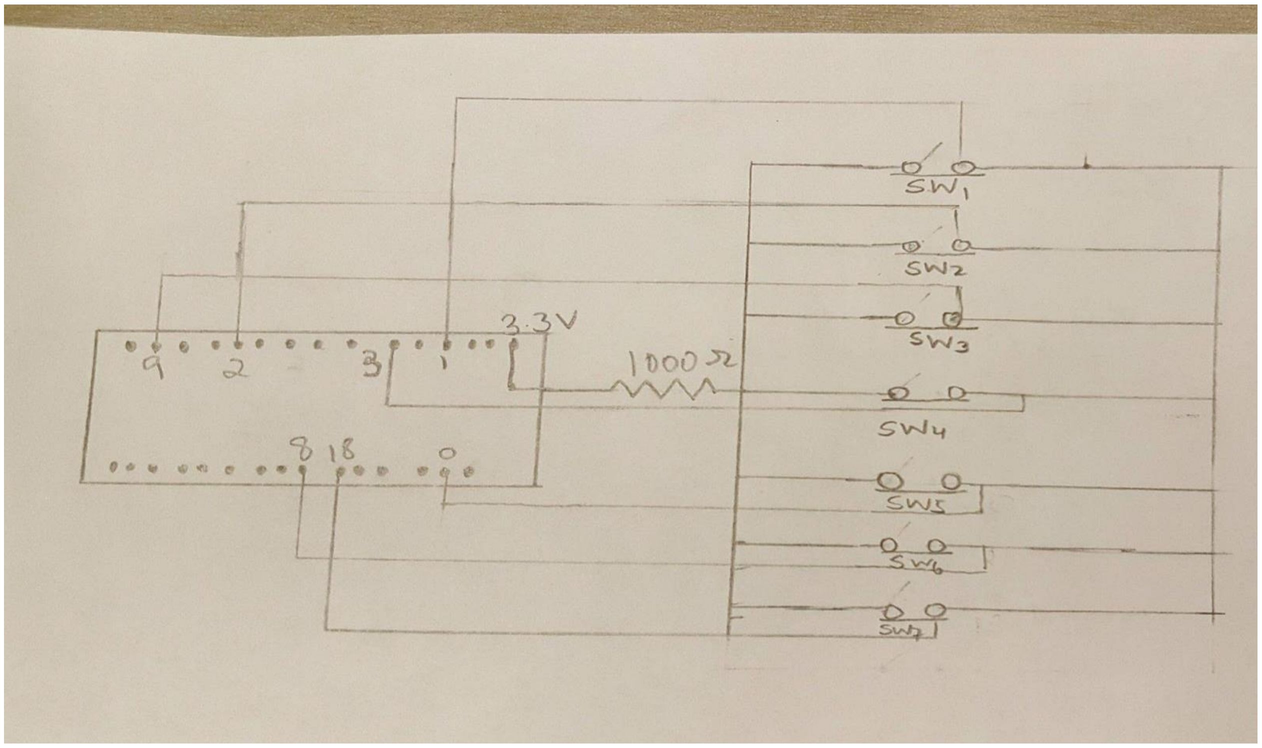

Figure 3: Circuit Diagram

2b. Hardware Components

- Onion Omega
- Power Dock
- Push-Buttons
- Wires
- 1K ohm resistor

2c. Hardware Overview

The hardware consists of push-buttons and jumper wires which are connected in parallel so that each button is only corresponding to one of the GPIO pins from the omega. Each button is assigned to one pin on the power dock so that our program is able to determine which button is pressed by the user.

Firstly, 3.3 Volts is supplied from Onion Omega to one of the terminal strips of the breadboard. A resistor is added in order to restrict the flow of current, thus preventing the possibility of wire overheating and current overflow through the pushbuttons and also to prevent the circuit from short circuiting. Although such an implementation is not necessary for this project as we are not dealing with high magnitudes of current, it is considered good practice to do so. The push buttons are connected in a parallel combination to the power rails and additional wire connects those buttons to their respective GPIO pins on the Onion Omega. This way each button is assigned to one pin on the power dock so that our program is able to differentiate which button is pressed by the user without interrupting the current flow in the rest of the circuit.

Since we have two breadboards attached together, the power rails from one terminal end of the breadboard is connected to the terminal end of the other breadboard. Both are connected together by a yellow wire. Each breadboard holds three buttons and they are placed in between the DIP support of each breadboard. An additional button, acting as the “space bar” is placed on the bottom of the second breadboard. On the circuit diagram, we have 7 push-buttons which are labelled as SW1, SW2, SW3, etc. SW1 to SW6 are required to be pressed in a combination of sequences to form a single letter. SW7 is technically our spacebar and when it is pressed, it will add a space character into our text file.

4. Software Design

a. Function Call Tree

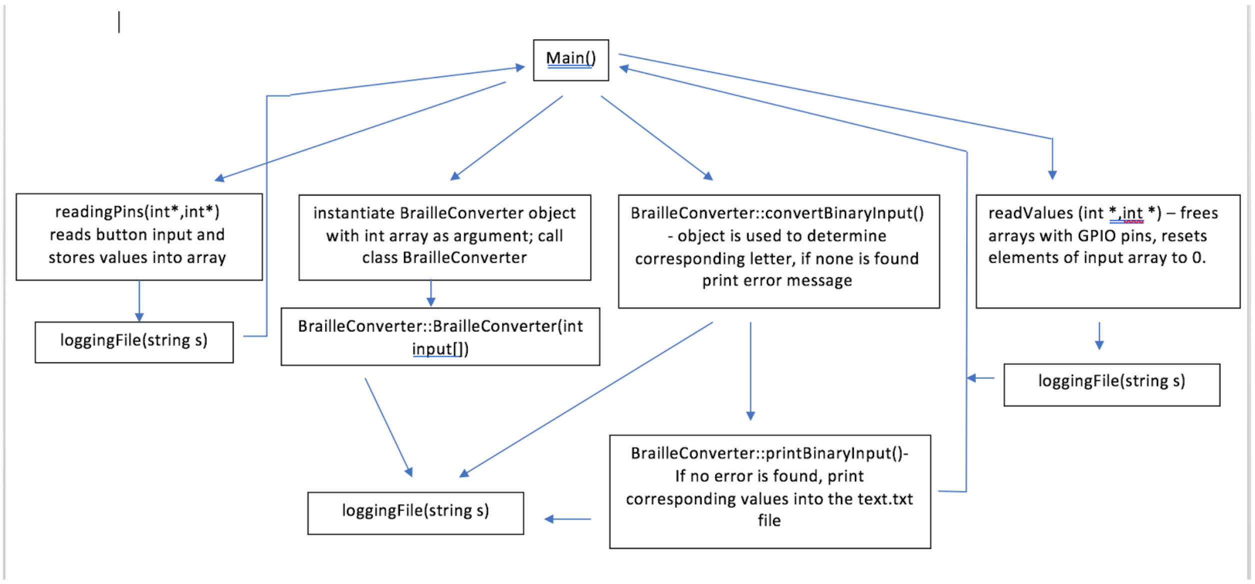
Main()
readingPins(int*,int*) reads button input and stores values into array
loggingFile(string s)
instantiate BrailleConverter object with int array as argument; call class BrailleConverter
BrailleConverter::BrailleConverter(int input[])
BrailleConverter::convertBinaryInput() - object is used to determine corresponding letter, if none is found print error message
readValues (int *,int *) – frees arrays with GPIO pins, resets elements of input array to 0.
loggingFile(string s)
loggingFile(string s)
BrailleConverter::printBinaryInput()- If no error is found, print corresponding values into the text.txt file

Figure 4: Function Call Tree

b. Classes /Function Definitions

***NOTE*: we reimplemented the c++ file "GPIO.cpp" for reading the input values of the push button**

Function Name: Main
Arguments: None
Variable Declarations: int gpioPin[7], int readvalues[7], int *preadValues,
int *pGPIO, BrailleConverter* test
Function Calls: reading Pins(int *, int *), resetValues(int *, int *), class BrailleConverter
Return Type: int

As simulated by the function call tree, most functions are called by main(). Initially, the main function creates 2 arrays. int gpioPin[7] stores which gpio pins are connected to specific buttons. For example gpioPin[0]=1, indicates that that the first button (array index) is connected on to gpio pin 1 on the onion omega. The second array, int readvalues[7], stores the signal reading of the gpio pins. This ensures that if the user presses a button, such signal will be stored into the program. The main function then enters an infinite while loop. In this loop, the function reading Pins(int *, int *) is called.This functions takes in two integer pointers that point to the two arrays initially declared. Listen to each of the seven push buttons and detect if the user pressed any. If so, the corresponding index of readvalues array will be updated to 1.

After this, the main function instantiates a new object from the BrailleConverter class and send the readvalues array as its argument. This way, the object can use its class method, convert Binary Input(), to detect which of the possible input combinations correspond to the user inputs. The class method print Binary Input(), is also called to print the letter/symbol corresponding to the user input onto the text file "test.txt".

The elements of the array readvalues are set back to zero, and frees the gpio pins by calling the function resetValues(int *, int *). This function takes to pointer that point to the arrays gpioPins and readvalues.

The user is then asked if they want to write additional letters or if they want to end the program. If the user wishes to the end the program, the while loop will end and the main function will return 0;

Function Name: readingPins
Arguments: (int *preadValues, int *pGPIO)
Variable Declarations: int test, int index, int value, int pinDirection,
Function Calls: N/A
Return Type: void

Initially we re-implemented this function from the "GPIO.cpp" file which was written by TA. This file was able to read certain gpio pins and check for specific errors such as if the pins were already in use.

When this function is called, two integer pointer variables that point to arrays are passed in as arguments. The function then uses those data members to first check if the gpio pins can be used and can be assigned to a specific button. The function then enters a for loop which reads the input signals of each button. If a button is pressed, the array storing the input signal is changed to a value of one. For example if button one is pressed then preadValues[1]=1.

To repeat this process for all 7 buttons, the for loop is nested inside another for loop which runs 7 times.

Function Name: resetValues
Arguments: (int *preadValues, int *pGPIO)
Variable Declarations: None
Function Calls: N/A
Return Type: void

When this function is called, two integer pointer variables that point to arrays are passed in as arguments. The function then enters a for loop which frees each gpioPin from array referenced by pGPIO. This way pins can be redefined and reused. The array that is referenced by preadValues is also set to zero, so users can have a new input combination.

Class Name: BrailleConverter
Class Members:

- BrailleConverter(int input[]) : This constructor takes an integer array which correspond to the user inputs. It stores the values into the binaryInputs array.
- static void convertBinaryInput(): This method take the binaryInputs array as an argument. It uses a nested loop to iterate through the 2D array brailleValues, and checks to see if any of the sequences correspond to the user inputs. If so, the corresponding letter/symbol on the array letters will be stored into the char array letters. This method has a void return type.
- static void printBinaryInput(): this method will take the character from the char letter and print it on to the text file test.tst. It will first check if the file can be opened. If not an error message will appear.
- static int binaryInputs[7]: this array will store the user's inputs which are passed as an argument when the class object is instantiated
- static char letter: this character will store the letter corresponding to the user's binary inputs. It is initially set to 0.
- static char fileName[]: this character array store the name of the textfile that the method print Binary Input() prints out to.
- static int brailleValues[27][7]: This 2D array stores all the possible user combinations that correspond to the letter/symbol in the character array letters[]
- static char letters[27]: this char array store the possible letters/symbols that the user can input
- ~BrailleConverter(): This destructor closes the outfile.

c. System Independent Components

The class BrailleConverter is an independent component. When an object is in instantiated, we only send an integer as an argument. Since the array is declared and defined before software interaction with the onion omega, it can work without any hardware interaction.
The function resetValues does not interact with the omega, its only purpose is to reset the arrays back to zero.

The logging function, interacts with all aspects of the code. It outputs if the program has reached certain parts of the code and whether it was a desired outcome or an error. As such this component does not depend on the omega. If the omega it not attached,the logging function will not print the parts of the code the that interact with the hardware, on to a textfile.

d. System Dependent Components

The function reading Pins() is a dependent component as it sets all the gpio pins corresponding to specific buttons. It also checks if the pins are already in use, and reads the input signals from the omega. This function cannot run without the onion omega.

e. Logging Infrastructure

Function Name:logging File
Arguments: (string input)
Variable Declarations: char fileName[], struct tm * ,time_t rawtime,
Function Calls: N/A
Return Type: int

This logging function prints on to the text file "logging.txt”. If the text file can be opened, it will print out the string message passed by the argument input as well as the specific date and time the function is called. We included the time to see how fast/slow our program runs. If the function can successfully print onto a text file, it will return 0. If an error arises when the file is being opened, it will return -1.

4. Testing Protocol

We had various tests to ensure that each component of our software and hardware runs correctly. Before we had the hardware component of our project working, we created a class which an array of the binary inputs that were going to be given by the user from the Omega, and their corresponding letter/symbols. Our initial test case was to determine if the correct letter was being outputted. We did this by instantiating an object and passing the char array of '1','0','0','0','0','0','0','0','\0’ which corresponds to the character ‘A’ in our array of letters that we had set from before. We tested for every single letter that corresponded to the binary input to assure ourselves that every letter from the letter array corresponded to every row of the binaryInput char array. From there, we checked if our program was outputting the letters and in correct order into our text.txt file. We also implemented error handling cases in our software. For example, if the user inputted the wrong combination of buttons than the program would output "You have inputted the wrong combination. Please try again.".

After we completed our hardware part of the project, we also made changes to our software which included changing the char array brailleValues and binaryInput into type int for simplicity because the GPIO pins were returning an int value when the buttons were pressed. We attempted to test our hardware component by trying to make one button work. In our first combined attempt for our software and hardware components, we initially gave the user 5 seconds to press a button. We decided it would be better to give the user 1 second because otherwise the time complexity increases and it makes our program much slower because we would be wasting 35 seconds compared to 7 seconds since we have 7 buttons.

After we had completed and combined every aspect of our project, we ran our program to ensure that the log files, text.txt and the hardware part were working properly. To test for this, we decided to write words onto our text.txt file and check the logging.txt file every time to see if the correct output was being printed in there. For example, we typed “C” into the text.txt file by pressing the correct sequences of buttons and then we did the same for the letters “A” and “T”. Our expected result was for text.txt file to have CAT written, the logging.txt file to have written all the combinations of buttons that were pushed for each letter and the time at which they were pushed. The actual outcome turned out to be the same as our expected result which allows us to believe that our system does what it’s supposed to do.

Lastly, we included error handling cases relate to the GPIO pins on the onion omega. If the pins were already in use and couldn’t be assigned to a specific button then the error message "This GPIO pin is in use" would be printed onto the console.

5. Limitation

a. Hardware Limitation:

Due to the fact that the braille numbers coincide with the first ten braille alphabets, were we unable to allow the user to input numbers. To correct this, we will have to add more buttons, whilst keeping into mind that the number of pins available on the Omega are limited and that some of the pins aren't even working properly. To increase the number of GPIO pins available we should also connect our omegas to each other via UART (universal asynchronous receiver-transmitter).
Another hardware limitation was the fact that the onion omega had errors reading several GPIO pins simultaneously. If only one button was pressed the omega would lag and assume others were pressed as well. To avoid this, our software only reads the GPIO pins one at a time.

One major problem with the hardware was that not all the buttons have the same configuration, that is, not every button is Active-Low or Active-High and while coding we assumed that the buttons returned 1 once pressed. Thus, only active-high buttons can be used with our project.

b. Software Limitation:

The software written isn’t memory efficient and has a high runtime complexity. In our main() function, we have an infinite while loop which ends only if the user decided to end the program. Because of this, our program has the potential to run infinitesimally which will cause fatal errors over time. In our infinite while loop a new object instantiate a new object for every iteration which will eventually lead to memory overflow if the user is writing a large amount of text. Although, a fact that should be kept in mind is that even if the user writes a file that contains two full pages worth of content, it wouldn’t still occupy more than 20 Kilobytes of memory space and there probably wouldn’t be a memory overflow for most of the real-life cases but that is still a limitation that could be tackled by using dynamic memory allocation techniques.

We also couldn’t implement backspace because of the limited number of available GPIO pins. If we were to implement this, we would have to write additional code that will read the text file and edit its contents.

Furthermore, we do realize that the software wasn't written following the best programming practices.

6. Improvements & Reflections

Overall, our project turned out to be pretty much along the lines of our proposal and does more or less of what we expected it to do. After subjecting it to a rigorous testing process, we found it to be consistent and following the desired outcomes. Also, we do realize that additional features could have been implemented into both the hardware and software aspect of our project, making it much more efficient.
For example, we could have used an Arduino or a Raspberry Pi dock as it has a superior gpio pin detection compared to the Onion Omega. Also, its corresponding IDE is easy to use and implement.

In terms of software, we could have created a dictionary and added a predictive text and autocorrect feature, making our code much more complex and more useful in real life settings.

We can also implement additional test cases in order to see any errors that arise. This in turn will make our software more robust and have the desired error handling capabilities. We can also implement more of the desired programming practices which will make our program easy to easy and much more organized. Lastly, we could place the BrailleConverter class in a separate header file which once again follows good programming practice.

7. Appendix

a. Source code

Source code for ECE150FinalProject.cpp

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <unistd.h>
#include <fcntl.h>
#include <iostream>
#include <fstream>
#include <ugpio/ugpio.h>
#include <stdio.h>
#include <sstream>

using namespace std;
//created a class that will store the input button and corresponding letters
//Initializing functions
void readingPins(int *preadValues, int *pGPIO);
void resetValues(int *preadValues, int *pGPIO);
int loggingFile(string s);
class BrailleConverter

        {
                // here is where we initilaze all the variable, we will use an array to story the
                //possibe input combinations and have a corresponding array for the letters

                public:
                        BrailleConverter(int input[]);
                        static void convertBinaryInput();
                        static void printBinaryInput();
                        static void printBraille();
                        static int binaryInputs[7];
                        static char letter;
                        static char fileName[];
                        static int brailleValues[27][7];
                        static char letters[27];
                        ~BrailleConverter();

        };
        // initialize the  values of the array
        // values are static because they wont change , stay the same for every instaintiation of the
class
        char  BrailleConverter::letter = '0';
        char BrailleConverter::fileName[]= "text.txt";
        char BrailleConverter::letters[27]
={'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q',

        'R','S','T','U','V','W','X','Y','Z',' '};
        int BrailleConverter::binaryInputs[7] = {1,0,0,0,0,0,0};
        int BrailleConverter::brailleValues[27][7]= {  {1,0,0,0,0,0,0},

          {1,0,1,0,0,0,0},

          {1,1,0,0,0,0,0},

          {1,1,0,1,0,0,0},

          {1,0,0,0,0,0,0},

          {1,1,1,0,0,0,0},

          {1,1,1,1,0,0,0},

          {1,0,1,1,0,0,0},
          {0,1,1,0,0,0,0},

          {0,1,1,1,0,0,0},

          {1,0,0,0,1,0,0},

          {1,0,1,0,1,0,0},

          {1,1,0,0,1,0,0},

          {1,1,0,1,1,0,0},

          {1,0,0,1,1,0,0},

          {1,1,1,0,1,0,0},

          {1,1,1,1,1,0,0},

          {1,0,1,1,1,1,0},

          {0,1,1,0,1,0,0},

          {0,1,1,1,1,1,0},

          {1,0,0,0,0,0,0},

          {1,0,1,0,1,1,0},

          {0,1,1,1,0,1,1},

          {1,1,0,0,0,1,0},

          {1,1,0,1,1,1,0},

          {1,0,0,1,1,1,0},

          {0,0,0,0,0,0,1}
                                                                            };
                //loggingFile("initialialized Static members of object");



        //constructor will set the input values to the binary inputs array(memmber of the class)
        BrailleConverter::BrailleConverter(int input[])
        {
        		for(int i=0; i<7; i++)
        		{
        			binaryInputs[i]= input[i];
        		}

        		loggingFile("Function: BrailleConverter::BrailleConverter || Constructor sets 
        argument as member in object");

        	}
        	//destructor will close the file
        	BrailleConverter::~BrailleConverter()
        	{
        		std::fstream outfile;
        		outfile.close();
        		loggingFile("Function: BrailleConverter::~BrailleConverter || Destructor Closes 
        Outfile");


        	}
        	// this will go through every element in the brailleValues array and check if it corresponds 
        to a values
        	// if it does correspond to a value, that leter will be stored in the letter char
        	// if a corresponding value cannot be found an error message will show
        	void BrailleConverter::convertBinaryInput()
        	{
        		int value=0;
        		int index=0;
        		bool same = true;
        		bool match =false;
        		int counter=0;
        		bool skip=false;
        		int i=0;
        		int j=0;
        		int k=0;
        		letter = '0';
        		bool reached= false;
        		loggingFile("Function: BrailleConverter::convertBinaryInput ||  User input Values 
        are used and checked aganist 2D of possible letters");

        		while(same)
        		{

        			if (binaryInputs[k] == brailleValues[i][j])
        			{
        				value++;
        				match=true;
```

```
                                        if(value==7 && match==true)
                                        {
                                                  letter= letters[i];
                                                  reached = true;
                                                  loggingFile("Function:
BrailleConverter::convertBinaryInput ||  Corresponding letter was found, saved in data member
'letter'");
                                                  same= false;
                                        }
                                        else{
                                                  j++;
                                                  k++;
                                        }
                              }
                              else
                              {         match=false;
                                        value=0;
                                        i++;
                                        if(i==27)
                                        {
                                                  same=false;
                                                  reached==false;
                                        }
                                        j=0;
                                        k=0;
                              }


                    }

                    if(reached == false  || letter == '0')
                    {
                              loggingFile("Function: BrailleConverter::convertBinaryInput
||  Corresponding letter was not found, error message");
                              cout<<"You have inputted the wrong combination. Please try
again."<<endl;
                    }

          }


          void BrailleConverter::printBinaryInput()
          {
                    std::fstream outfile;
                    outfile.open(fileName, ios::out | ios ::app);
                    if (!outfile.is_open())
                    {
```

```
				loggingFile("Function: BrailleConverter::printBinaryInput || textfile
cannot be opened, file error is printed on to a screen");
				cout<<"Error opening file"<<endl;
				return ;
			}
			else{
			loggingFile("Function: BrailleConverter::printBinaryInput || textfile was opened
successfully, corresponding letter is printed on to a screen");
			outfile << letter;

			}
		}


int loggingFile(string input)
{
		char fileName[]= "logging.txt";
   time_t rawtime;
   struct tm * timeinfo;

   time ( &rawtime );
   timeinfo = localtime ( &rawtime );
   std::fstream outfile;
   outfile.open(fileName, ios::out | ios ::app);
		if (!outfile.is_open())
		{
			cout<<"Error opening file"<<endl;
			return -1;
		}


		outfile<< __DATE__<<" "<<__TIME__<<" "<<input<<endl;

		return 0;
}
void readingPins(int *preadValues, int *pGPIO)
{
		int test, index, value;
		int pinDirection;
		for(int i=0; i<7; i++)
				{

					if((test = gpio_is_requested(pGPIO[i])) < 0)
					{
						cerr << "This GPIO pin is in use" << endl;
						return ;
				}
				else
				{
					std::ostringstream oss;
					oss<<"Function: readingPins || gpio pin for button "<<
i+1<<"was read successfully. Starting read";
					std::string in = oss.str();
					loggingFile(in);
					cout << "Starting GPIO pin" << endl;
					if((pinDirection = gpio_request(pGPIO[i], NULL)) < 0)
					{
						cerr << "There was an error setting the pin value"
<< endl;
						return ;
					}
				}
				pinDirection = gpio_direction_input(pGPIO[i]);
				cout << "Starting read button " << i+1 << endl;

				for(int index = 0; index < 3; index++)
				{
					value = gpio_get_value(pGPIO[i]);
					if(value==1)
					{
						preadValues[i]=1;
						std::ostringstream oss;
						oss<<"Function: readingPins || Button "<<i+1<<"
was pressed, value stored in array";
						std::string in = oss.str();
						loggingFile(in);

						cout << "Button "<< i+1<<" was pressed"<< endl;
					}

					value=0;
					sleep(1);
				}
			}

}
void resetValues(int *preadValues, int *pGPIO)
{
	loggingFile("Function:resetValues || GPIO pins are freed and values are set back to zero
for next read");
		for(int i=0; i<7; i++)
		{
```

```
                *(preadValues + i)=0;
                gpio_free(pGPIO[i]);

        }
}

int main(){



        int gpioPin[7]={0};
        int readvalues[7]= {0,0,0,0,0,0,0};
        gpioPin[0]= 1;
        gpioPin[1]= 2;
        gpioPin[2]= 9;
        gpioPin[3]= 3;
        gpioPin[4]= 0;
        gpioPin[5]= 8;
        gpioPin[6]= 18;
        int *preadValues;
        preadValues= readvalues;
        int *pGPIO= gpioPin;
        loggingFile("Function: main || arrays and pointer variables are declared");
        while(1)
        {
                loggingFile("Function:main || calling readingPins() function");
                readingPins(preadValues, pGPIO);



                loggingFile("Function:main || creating new BrailleConverter object
passing readvalues array");
                BrailleConverter* test= new BrailleConverter(readvalues);
                loggingFile("Function:main || calling convertBinaryInput() in object to
find corresponding letters");
                test->convertBinaryInput();


                char value= test->letter;
                char printValues[8]= {'0','0','0','0','0','0','0','\0'};
                for(int i=0; i<7;i++)
                {
                        printValues[i]=(char) ('0' +readvalues[i]);
                }

                if(test->letter != '0')
                {
```

```
					loggingFile("Function:main || read was sucessful printing
corresponding letter on to an array");
					cout<<"The binary inputs are: "<< printValues<<endl;
					cout<<"And the corresponding letter is "<<test->letter<<endl;
					test->printBinaryInput();
				}
				loggingFile("Function:main || calling resetValue() function");
				resetValues(preadValues, pGPIO);



				loggingFile("Function:main || read was sucessful asking user if the want to
continue");
				cout<<"Enter 'y' if you wish to continue or 'n' if you dont"<<endl;
				char input;
				cin>>input;


				if(input=='n')
				{
					loggingFile("Function:main || user does not want to continue,
ending program");
					cout<<"Ending program"<<endl;
					delete preadValues;
					delete pGPIO;
					return 0 ;
				}


			}

			return 0;


		}
```

Source code for makefile (Came with the virtual machine and it was created by TA so that we can compile our program)

```
# main compiler
CXX := g++

TARGET1 := ECE150FinalProject
all: $(TARGET1)

$(TARGET1):
  @echo "Compiling C program"
  $(CXX) $(CFLAGS) $(TARGET1).cpp -o $(TARGET1) $(LDFLAGS) -l$(LIB)

clean:
  @rm -rf $(TARGET1) $(TARGET2)
```

Source code for xCompile.sh (Came with the virtual machine and it was created by TA so that we can compile our program)

```
#!/bin/bash

# define the usage
usage () {
  echo "Arguments:"
  echo "  -buildroot <path to buildroot>"
  echo "  -lib \"<list of additional libraries to link in compile>\""
  echo ""
}

# define the path to the buildroot
BUILDROOT_PATH=""
USER_LIBS=""
bDebug=0


##################
# parse arguments #
while [ "$1" != "" ]
do
  case "$1" in
        # options
        buildroot|-buildroot|--buildroot)
                shift
                BUILDROOT_PATH="$1"
                shift
        ;;
        lib|-lib|--lib)
                shift
                USER_LIBS="$1"
                shift
        ;;
        -d|--d|debug|-debug|--debug)
                bDebug=1
                shift
        ;;
```

```
            -h|--h|help|-help|--help)
                    usage
                    exit
            ;;
           *)
                    echo "ERROR: Invalid Argument: $1"
                    usage
                    exit
            ;;
  esac
done

# check to ensure correct arguments
if [ "$BUILDROOT_PATH" = "" ]
then
  echo "ERROR: missing path to buildroot"
  echo ""
  usage
  exit
fi


# define the toolchain and target names
TOOLCHAIN_NAME="toolchain-mipsel_24kc_gcc-5.4.0_musl"
TARGET_NAME="target-mipsel_24kc_musl"

# define the relative paths
STAGING_DIR_RELATIVE="staging_dir"
TOOLCHAIN_RELATIVE="$STAGING_DIR_RELATIVE/$TOOLCHAIN_NAME"
TARGET_RELATIVE="$STAGING_DIR_RELATIVE/$TARGET_NAME"

# define the toolchain paths
TOOLCHAIN="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE"
TOOLCHAIN_BIN="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/bin"

TOOLCHAIN_INCLUDE="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/include"
TOOLCHAIN_LIB="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/lib"
TOOLCHAIN_USR_INCLUDE="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/usr/incl
ude"
TOOLCHAIN_USR_LIB="$BUILDROOT_PATH/$TOOLCHAIN_RELATIVE/usr/lib"

# define the target paths
TARGET="$BUILDROOT_PATH/$TARGET_RELATIVE"

TARGET_INCLUDE="$BUILDROOT_PATH/$TARGET_RELATIVE/include"
TARGET_LIB="$BUILDROOT_PATH/$TARGET_RELATIVE/lib"
TARGET_USR_INCLUDE="$BUILDROOT_PATH/$TARGET_RELATIVE/usr/include"
```

```
TARGET_USR_LIB="$BUILDROOT_PATH/$TARGET_RELATIVE/usr/lib"

export STAGING_DIR="BUILDROOT_PATH/$STAGING_DIR_RELATIVE"

# define the compilers and such
TOOLCHAIN_CC="$TOOLCHAIN_BIN/mipsel-openwrt-linux-gcc"
TOOLCHAIN_CXX="$TOOLCHAIN_BIN/mipsel-openwrt-linux-g++"
TOOLCHAIN_LD="$TOOLCHAIN_BIN/mipsel-openwrt-linux-ld"

TOOLCHAIN_AR="$TOOLCHAIN_BIN/mipsel-openwrt-linux-ar"
TOOLCHAIN_RANLIB="$TOOLCHAIN_BIN/mipsel-openwrt-linux-ranlib"

# define the FLAGS
INCLUDE_LINES="-I $TOOLCHAIN_USR_INCLUDE -I $TOOLCHAIN_INCLUDE -I
$TARGET_USR_INCLUDE -I $TARGET_INCLUDE"
TOOLCHAIN_CFLAGS="-Os -pipe -mno-branch-likely -mips32r2 -mtune=24kc -fno-caller-
saves -fno-plt -fhonour-copts -Wno-error=unused-but-set-variable -Wno-error=unused-result -
msoft-float -mips16 -minterlink-mips16 -Wformat -Werror=format-security -fstack-protector -
D_FORTIFY_SOURCE=1 -Wl,-z,now -Wl,-z,relro"
#TOOLCHAIN_CFLAGS="-Os -pipe -mno-branch-likely -mips32r2 -mtune=34kc -fno-caller-
saves -fhonour-copts -Wno-error=unused-but-set-variable -Wno-error=unused-result -msoft-float
-mips16 -minterlink-mips16 -fpic"
TOOLCHAIN_CFLAGS="$TOOLCHAIN_CFLAGS $INCLUDE_LINES"

TOOLCHAIN_CXXFLAGS="$TOOLCHAIN_CFLAGS"
#TOOLCHAIN_CXXFLAGS="-Os -pipe -mno-branch-likely -mips32r2 -mtune=34kc -fno-
caller-saves -fhonour-copts -Wno-error=unused-but-set-variable -Wno-error=unused-result -
msoft-float -mips16 -minterlink-mips16 -fpic"
TOOLCHAIN_CXXFLAGS="$TOOLCHAIN_CXXFLAGS $INCLUDE_LINES"

TOOLCHAIN_LDFLAGS="-L$TOOLCHAIN_USR_LIB -L$TOOLCHAIN_LIB -
L$TARGET_USR_LIB -L$TARGET_LIB"

# debug
if [ $bDebug -eq 1 ]; then
  echo "CC=$TOOLCHAIN_CC"
  echo "CXX=$TOOLCHAIN_CXX"
  echo "LD=$TOOLCHAIN_LD"
  echo "CFLAGS=$TOOLCHAIN_CFLAGS"
  echo "LDFLAGS=$TOOLCHAIN_LDFLAGS"
  echo "USER_LIBS=$USER_LIBS"
  echo ""
fi


# first run make clean
make clean
```

```
# run the make command
make \
  CC="$TOOLCHAIN_CC" \
  CXX="$TOOLCHAIN_CXX" \
  LD="$TOOLCHAIN_LD" \
  CFLAGS="$TOOLCHAIN_CFLAGS" \
  LDFLAGS="$TOOLCHAIN_LDFLAGS" \
  LIB="$USER_LIBS"
```

b. Peer Contribution

Amio Rahman: Wrote readingPins() and contributed to int main(), compile/ran code on omega, test cases/debugging, setup hardware component

Faduma Ahmed: Wrote class BrailleConverter and logging file and contributed to int main(), setup hardware component

Samanvay Vajpayee: wrote resetValues() and contributed to int main(), wrote pseudo code, setup hardware component

c. Citations

[1] https://en.wikipedia.org/wiki/Braille_technology
[2] http://www.afb.org/info/living-with-vision-loss/braille/what-is-braille/123
[3] http://lakeshorearts.ca