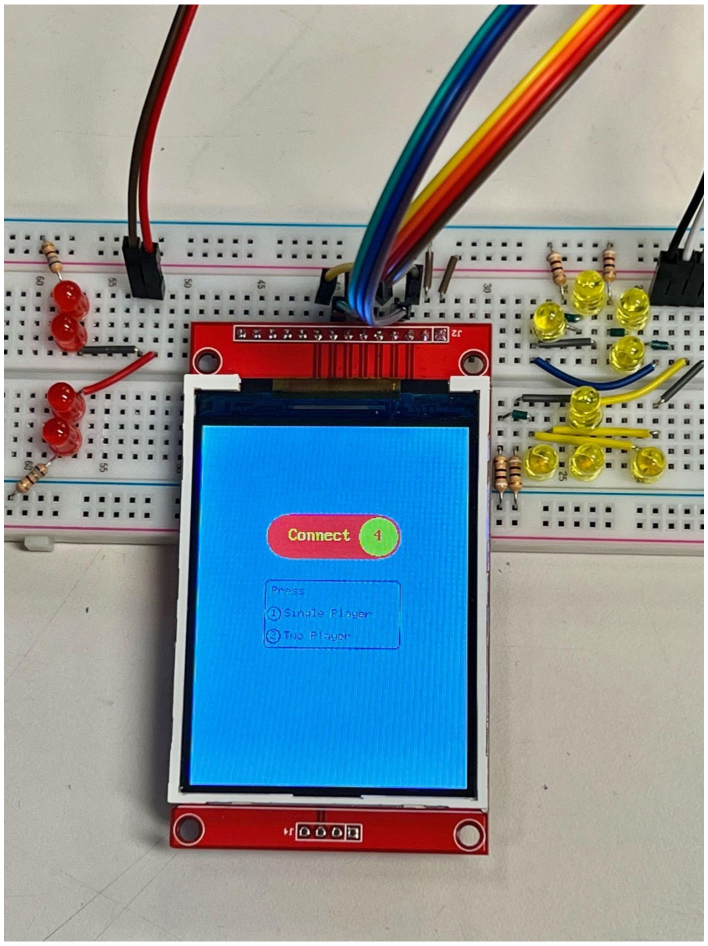
6.115(1) Final Report

Connect Four

Jasmine Wu



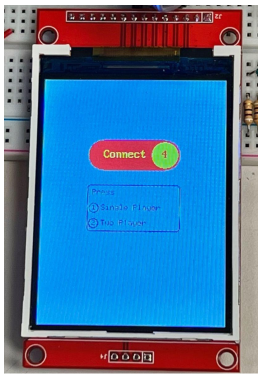
Overview

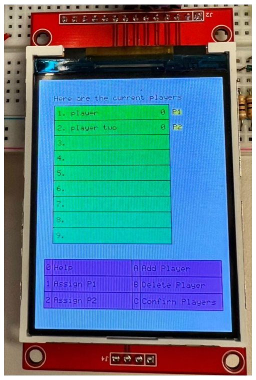
Project Summary

For the final project, a one or two player connect four game was created. In single player mode the user plays against the computer. Users will control the game primarily using a 16 key keypad, with the game state being displayed on a TFT screen.

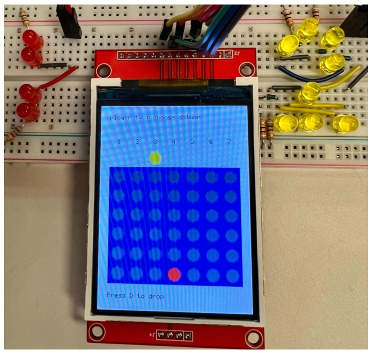
The general structure of gameplay goes as follows (subsequent sections will go into much more detail).

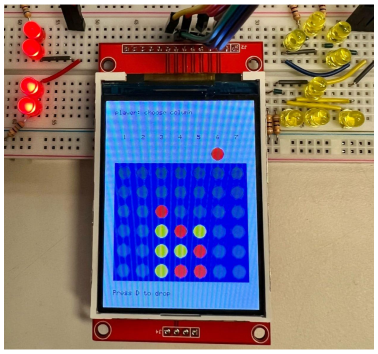
1. The user(s) see the home screen, and choose between single or two player mode.
2. They are taken to a setup screen, where they can create, delete, and assign players for gameplay.



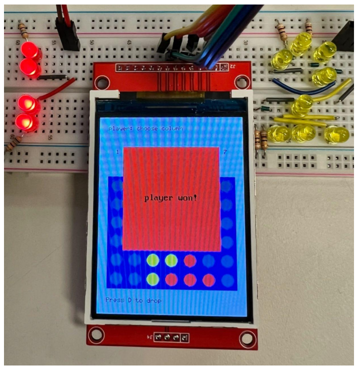


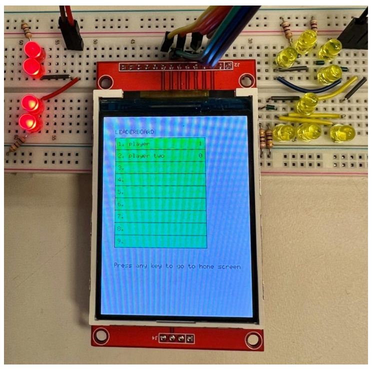
3. Once they confirm player selection, gameplay begins. Players take turns placing chips in columns.





4. When a player wins, a message will be displayed and the program will display the leaderboard.





Pressing any key will take the users back to the home screen, where the process can be repeated.

Report Summary & Notes

The report begins by introducing the hardware components of the project. Then, it will go in depth on the implementation of the gameplay (software). Elements of the project that are closely tied to both software and hardware are covered in the Integration section. Finally, exploration and progress on the Independent Inquiry portion of the project will be discussed.

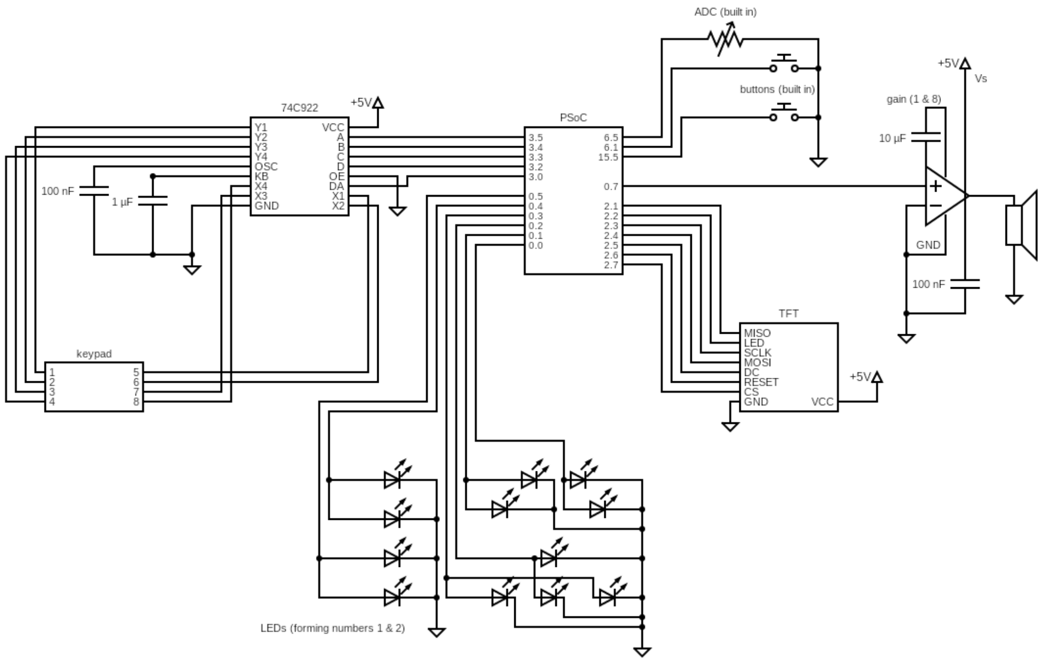
Parts II, III, and IV end with a “Next Steps” section which describe ways in which the scope of the project could be expanded in that specific area.

The relevant code files can be found in the code appendix, but a complete collection of the code can be found on Github at https://github.com/jasmine2000/game_version1 (this is also easier to browse through).

Note: In the context of the game, the term “player” can easily be interpreted as a number of things. As a result, this report aims to refer to the person playing the game as the **user**, the names in the system as **players**, and the 1 or 2 players actually playing the game as **P1** and **P2**.

1. Hardware Diagram and Component Summary

Below is a wiring diagram of the system and a brief description of each component. The functionality and implementation of each peripheral will be covered in the Section III (Integration). The only exception is the keypad- it is used heavily in gameplay and will be covered in Section II (Software).



Component	Purpose
TFT	The TFT screen is used as the primary visuals for the game, showing things such as the list of players or the state of the game.
Keypad + 74C922	The keypad, along with its encoder, is the primary method for users to interact with the game.
ADC + Buttons	These components are part of the PSoC Big Board, and are only used to enter a player's name in the setup state of the game.
Speaker + LM386	The speaker is only used as a form of "negative" feedback, playing an error-like buzzer sound when a user tries to do something that is not allowed.

2. Gameplay Logic (Software)

2.1 Introduction

The major structure of the game is an infinite loop with switch (if) statements that direct the flow based on the current game state (home, setup, choosing, dropping, leaderboard). Besides the state machine managing the game state, there is a second state machine within the setup state, which is responsible for the assigning and modifying of the player list.

The structure of the code is intended to be very "non blocking" - each time through the large loop, only a small action is executed. In order to make this possible, indicator variables are used pretty thoroughly throughout the design.

This section will begin with some context by going over global variables and briefly discussing the implementation of the (widely used) **`get_keypress()`** function. Then, a detailed description of each game state will be given. These descriptions begin with a summary of that state from the user's perspective (with images of the TFT screen), before the implementation of that state is discussed.

The way in which each section interacts with hardware will not be covered here (with the exception of the keypad) - it will be explained in the Integration section. Most code can be found in the Code Appendix, and all code can be found on Github.

2.2 Global Variables[1]

`const int max_players:` Maximum number of players allowed in game. Currently set to 9 to match the numbers on the keypad.
`const int name_length:` Maximum length of a player name.

`int state:` Keeps track of game state. Values map to setup, choosing, dropping, and leaderboard.
`int game_type:` Keeps track of game game type- 1 for single player, 2 for two player.

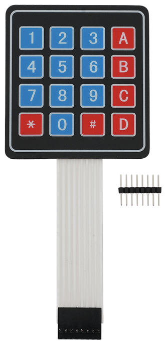
`char all_players[max_players][name_length]:` Holds all the players that have been entered. The array can hold up to **`max_players`** players, each name up to name_length.
`int all_scores[max_players]:` Holds all the player scores in the same order as all_players.
`int num_players:` Number of players currently in the system.

`int player_arr[2]:` Maps a current player to their name. P1's name can be found by taking the value in **`player_arr`**[0] and using it to index into **`all_players`**. The same applies for P2, but the corresponding index is in **`player_arr`**[1].
`int current_player:` Either 0 or 1, an indicator of which player's turn it is. Initialized to 0 (P1 starts).

`int board[6][7]:` Holds the current game board. At a specific index, 0 means it is unoccupied, 1 means it is occupied by P1, and P2 means it is occupied by P2. Initialized to all 0s.

[1] As mentioned, these are not all the global variables - ones only involved in specific states will be discussed when those states are presented.

2.3 Keypad



Keypad Summary

The keypad is used as the primary means of control in all parts of the game. Different numbers/letters are relevant depending on where in the game the user is.

Keypad Implementation

refer to keypad.c

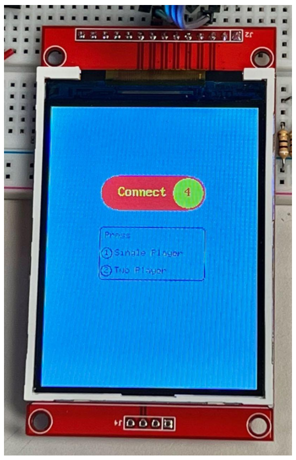
The logic to receive a keypress is almost identical to the one used in assembly code during labs - the program waits for the DA (data available) line to go high, then it reads the output values from A, B, C and D. The difference is that the output isn't being sent to a port, it is being sent to four individual pins. As a result, we need to shift each value left the appropriate number of times then add them together to get the index of the button pressed.

Then, the program maps the index to an actual value - digits are mapped to themself[2], letters are mapped to numbers starting from 10, and symbols are mapped to numbers starting from 20.

The goal of this function is to return an actual value if a valid button was pressed, and 0 otherwise[3]. The definition of a "valid button" changes depending on the usage of the function - during gameplay, a user can only choose numbers 1 through 7. In the setup state, when a player is choosing options, they can choose between 1, 2, A, B, and C.

As a result, we needed a "mode" parameter to determine the intended usage of the function so it could appropriately decide when to output 0 and when to output an actual value. The `num_players` parameter is only used if the mode is player selection and will determine the largest possible player number a user could choose - see the setup section of gameplay logic for a detailed variable description.

2.4 Home



Home Summary (from a user's perspective)

In addition to being a home screen, this is also a "pre-setup" screen where users choose between single and two player mode. Even after entering the setup state, the user can press * on the keypad to return to the home screen and switch a different player mode.

The screen will also be shown when starting a new game after completed games, as selecting the player mode is a necessary step to determining player assignment and flow of the game.

Home Implementation

The home screen implementation is relatively simple - the program displays the screen and waits for user input. If the input is valid, the game type is set, variables are initialized for setup state, and the setup screen is displayed.

[2] The actual number zero is mapped to 30.
[3] See [2].

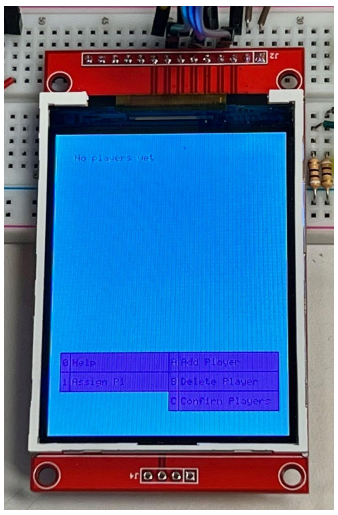
2.5 Setup

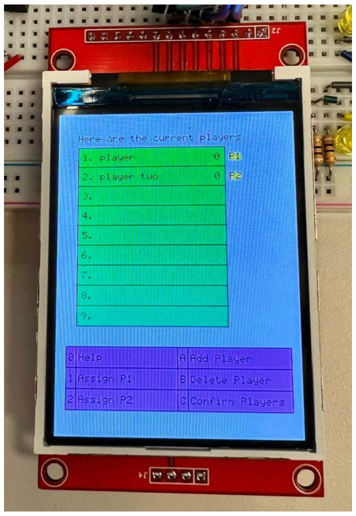
Setup Summary (from a user's perspective)

Until the user "moves on" from the setup state, we will continue to display the following information: (1) All players currently in the system, (2) players currently assigned to P1 and P2, and (3) options available. The option to assign P2 is only available in two player mode.

The number/letter next to the option indicates the corresponding keypad press.

Example screens are shown - the left is single player mode, no players, no assignments (what a user sees when first turning on the game). The right is two player mode, 2 players, both assigned.



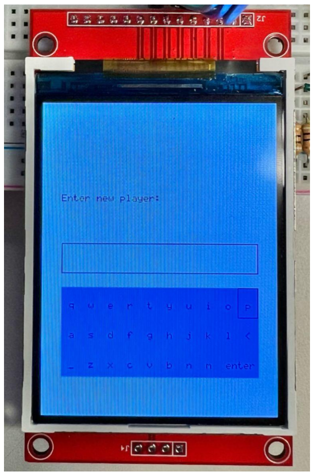


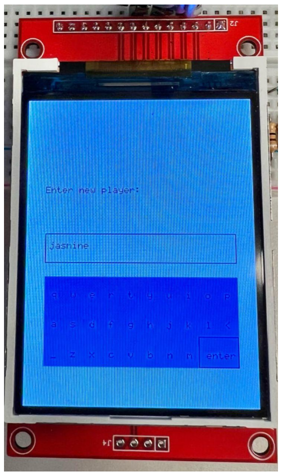
A user will choose one of the options. If it is something they are able to do, then the program will act accordingly. If not, a buzzer sound will be emitted from the speaker to indicate they have done something incorrect. The screen updates itself depending on changes made by the user.

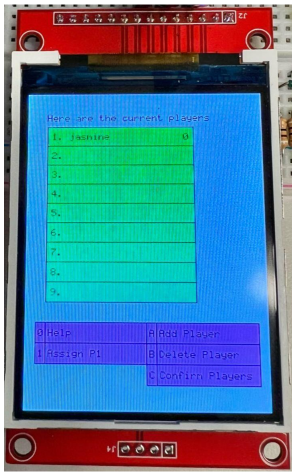
When starting up the game for the first time, there will be no players and empty assignments (left screen). The only successful action that can be done is adding a new player. Only players in the player list can be assigned - new players always need to be added before being assigned.

The workflow of entering a new player is shown. It uses a potentiometer to move across a row, one button to toggle between rows, and one button to "press" a key. Once the name is entered, the user will press the "enter" key, which will save the name and return to the setup screen.

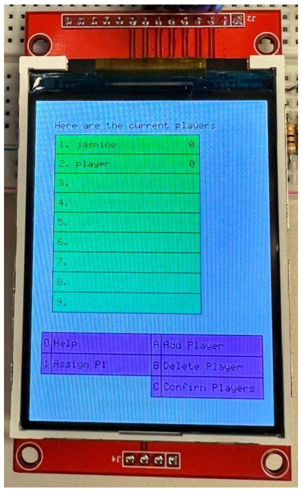
Because this feature is closely tied to hardware components, implementation details will be discussed in the Integration section.



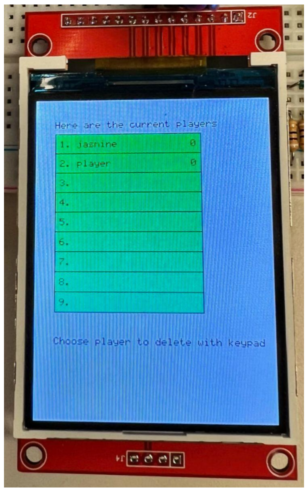




Players can be deleted using the delete command (B). This command was primarily created because only 9 players can be in the system at a time, so if 9 exist and a new person wants to play, an old one needs to be deleted. Choosing a specific player to delete is done with the keypad.

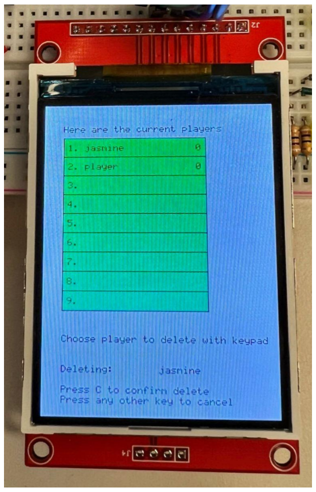

Here are the current players

1. jasmine	0
2. player	0
3.	
4.	
5.	
6.	
7.	
8.	
9.	


Here are the current players

1. jasmine	0
2. player	0
3.	
4.	
5.	
6.	
7.	
8.	
9.	

Choose player to delete with keypad

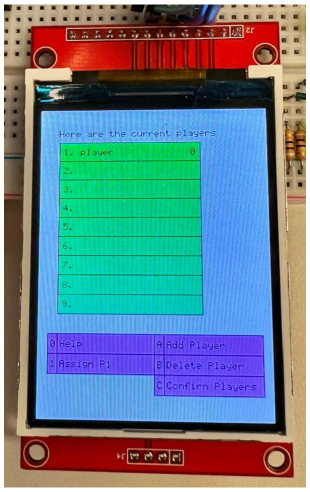

Here are the current players

1. jasmine	0
2. player	0
3.	
4.	
5.	
6.	
7.	
8.	
9.	

Choose player to delete with keypad

Deleting: jasmine

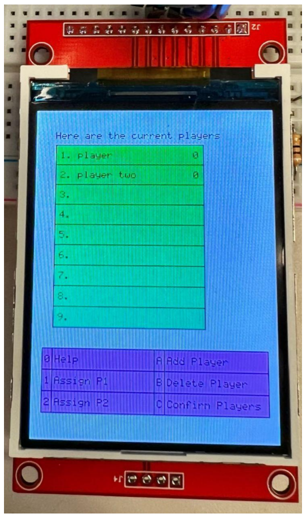
Press C to confirm delete
Press any other key to cancel


Here are the current players

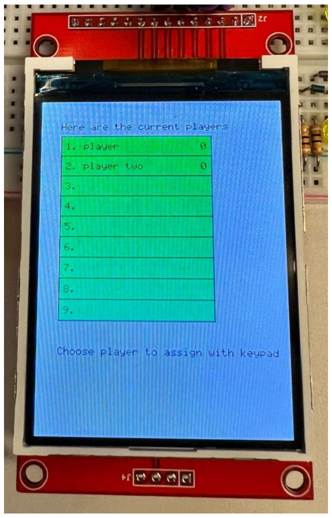
1. player	0
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	

Assigning a player is a similar process, where the user selects a specific player using the keypad. Assignments will be updated after pressing enter. Players can be reassigned more than once.

The images show the process of assigning P1 in two player mode.

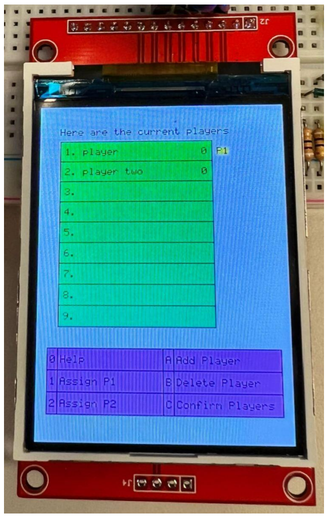

Here are the current players

1. player	0
2. player two	0
3.	
4.	
5.	
6.	
7.	
8.	
9.	


Here are the current players

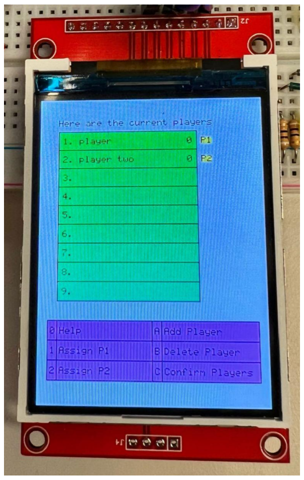
1. player	0
2. player two	0
3.	
4.	
5.	
6.	
7.	
8.	
9.	

Choose player to assign with keypad

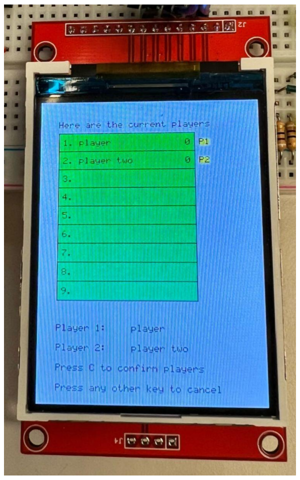

Here are the current players

1. player	0	P1
2. player two	0	
3.		
4.		
5.		
6.		
7.		
8.		
9.		

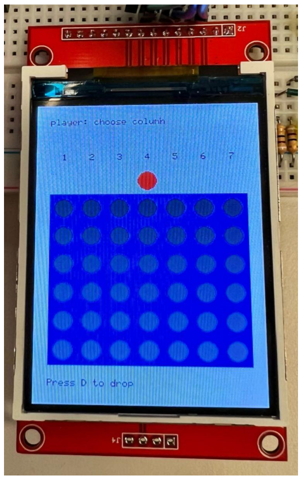
To begin gameplay, all “human” players must be assigned (2 in two player, 1 in single player). The program will display the assignments, and ask the player(s) to confirm. Only after confirming valid assignments will the program move on from the setup state.



```
Here are the current players
1. player         0  P1
2. player two     0  P2
3.
4.
5.
6.
7.
8.
9.

0 Help          A Add Player
1 Assign P1     B Delete Player
2 Assign P2     C Confirm Players
```



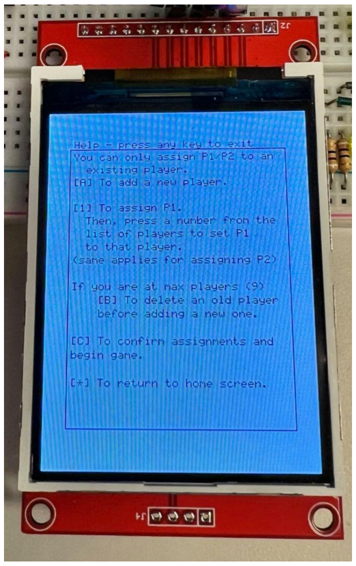
```
Here are the current players
1. player         0  P1
2. player two     0  P2
3.
4.
5.
6.
7.
8.
9.

Player 1:     player
Player 2:     player two
Press C to confirm players
Press any other key to cancel
```



```
player: choose column

 1   2   3   4   5   6   7

Press D to drop
```

After 1+ games have been played, the player assignments will default to the ones that last played.

If the user was unclear with any part of the setup process, they can press 0 to display the help screen. The contents of the help screen are a quick summary of the explanation above.



```
Help - press any key to exit
You can only assign P1/P2 to an
  existing player.
[A] To add a new player.

[1] To assign P1.
  Then, press a number from the
  list of players to set P1
  to that player.
(same applies for assigning P2)

If you are at max players (9)
    [B] To delete an old player
    before adding a new one.

[C] To confirm assignments and
begin game.

[*] To return to home screen.
```

Setup Implementation

refer to setup_state() function in main.c and setup.c.

Intro

Each time we enter the setup state, we get a new "action" from the player(s). This is retrieved using the **`get_keypress()`** function, which the mode set to OPTION_SELECT. The next subsections will describe how the program performs based on the requested action.

Adding Players

If the **`num_players`** variable is equal to **`max_players`**, this means that we have the maximum allowed players and we cannot add more. An error sound will play and the program will return to the beginning of the setup state.

Otherwise, the program will call the **`keyboard_get_player()`** function and receive a name input (up to length 10) from the user. It will put this in **`all_players[num_players],`** because we know that the **`num_players`** variable contains the smallest available index. The implementation of **`keyboard_get_player()`** will be discussed in the Integration section.

Deleting Players

If the **`num_players`** variable is 0, this means that there are no players at all (and therefore none can be deleted), so an error sound will play and the screen will not change. The program will return to the beginning of the setup state.

Otherwise, the program will call the **`get_keypress()`** function again, this time passing in mode PLAYER_SELECT and **`num_players`**. If the player number is valid, the program will prompt the user with a confirmation message that they truly want to delete this player.

If the player confirms the delete, we call a function **`do_delete()`** which starts one index after the deleted player and moves every player back one index in the **`all_players`** array. It always sets the last value in the array to empty. It does the same thing to the **`all_scores`** array, and also "unassigns" P1/P2 if they were set to this player.

Assigning Players

The beginning of this subroutine is identical to the one used to delete players. It checks that players exist, and if they do, receives a number from the user corresponding to a player number.

If the returned number is a proper player number, one of the values in **`player_arr`** will be set (based on whether P1 or P2 is being assigned) and the screen will reflect the change. The program also checks to make sure both P1 and P2 are not the same - if they are, it will only keep the more recent assignment, and unassign the other player.

Confirming Players

First, this routine checks to make sure the proper players are initialized (just P1 for single player, P1 and P2 for two player).

When the system starts, `player_arr` is initialized to `{max_players, max_players}`. This is in invalid index to `all_players` (it is zero indexed with size `max_players`). As a result, if a value in `player_arr` is set to `max_players`, then we know that it hasn't been initialized.

The system also checks to make sure both players in `player_arr` are different, although this should never happen (see end of Assigning Players section).

If all conditions are met, and the player(s) confirm the selections, then the state is changed to choosing and current_player is set to 0 (P1 begins).

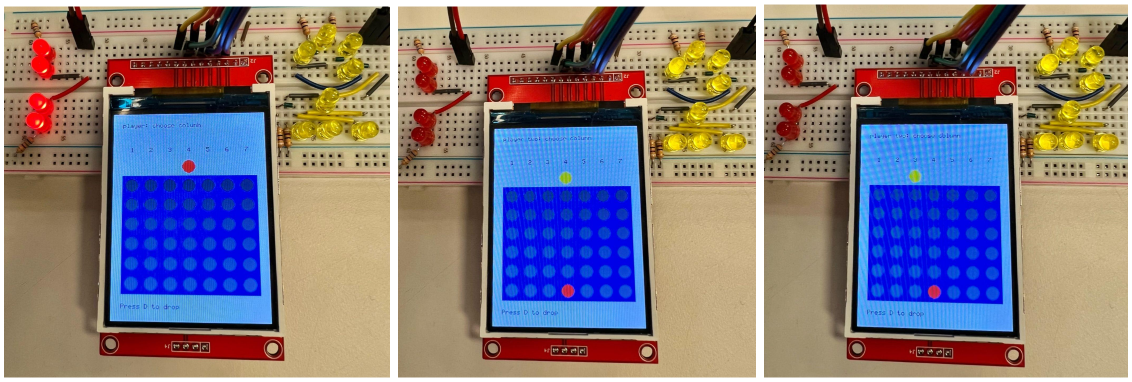
2.6 Choosing

refer to choosing_state() function in main.c and in gameplay.c.

Choosing Summary (from a user's perspective)

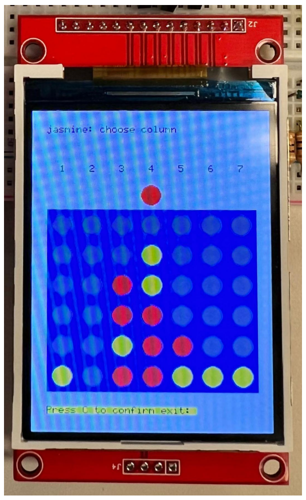
The game will indicate whose turn it is using both LED lights (yellow is difficult to see in photos) along with text at the top of the screen.

As a player goes through the "choosing" process, a chip (corresponding to the player's color) will hover above the board in the column that they have currently selected - this will be referred to as the "preview bar". The player is able to change their mind any number of times, and the chip will move to the column they are selecting. The chip initially defaults to column 4, or the middle column.



When a player has made up their mind, they will press D to drop the piece. Because the chip defaults to column 4, there will always be a column selected when they press "drop". They will move onto the drop state no matter what - the validity of the move will be checked in the drop state.

Players also have the opportunity to exit the game early at this state by pressing *. If they do so, the game will confirm that they want to exit (yellow highlighted text at the bottom of the screen), and send the player to the leaderboard state (with no scores changed).



Choosing Implementation

In the choosing state, the program repeatedly receives new keypress inputs from the player, and branches into different subroutines accordingly. Most of the time, the input will be a valid column, which will result in the "preview bar" getting updated, and the player remaining in the state. If the player "drops" the piece, the state will be changed to dropping.

There are two variables that track the user's interactions. **`selection`** is set to what the player has selected, while **`current_column`** is set to the last valid column they chose and used to update the screen. **`current_column`** is only updated if **`selection`** *is* a valid column. It is possible for the user to choose an option that doesn't correspond to a column, and the preview bar should not attempt to display that change.

If the game is in single player mode and it is the computer's turn, the program calls the helper function **`get_computer_column()`**, which returns a column that the computer should place a chip in. This helper function checks to see if choosing any column would result in either the user or the computer winning. If either is true, the computer will go there (to either prevent the user from winning or to win itself). If not, the computer will randomly select a column and go there.

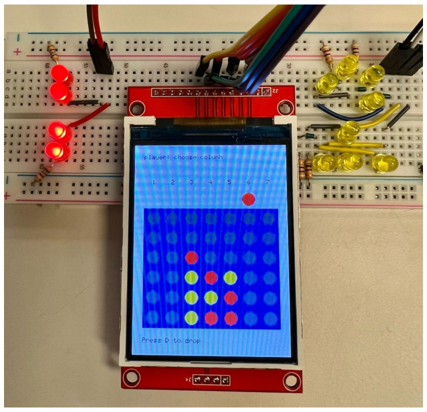
2.7 Dropping

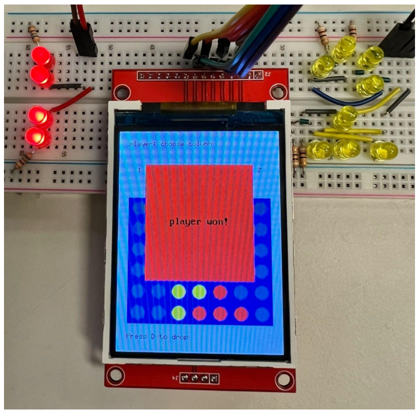
refer to dropping_state() function in main.c and in gameplay.c.

Dropping Summary (from a user's perspective)

This state happens nearly instantaneously for the player. If the move is invalid (the selected column is full), we return to the choosing state but without changing the **`current_player`** value (player goes again).

If the move is valid, the change is reflected in the board and the screen. The board will then be checked for a winner. In most cases (no winner), the players switch turns, and the program returns to the choosing state. If there is a winner, the winner's score in **`all_scores`** is incremented and the game transitions to the leaderboard state. Below is P1 winning.



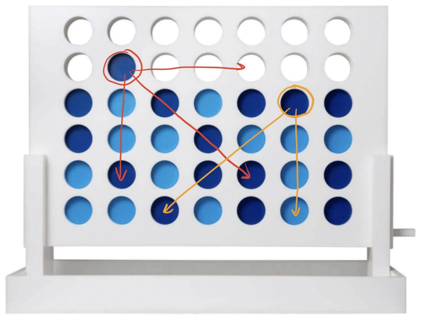


Dropping Implementation

This state begins by calling a helper function to find the lowest row which is unoccupied in the specified column, stored in the variable **`lowest_row`**. If every row is occupied, the function would return a row out of bounds, which will indicate that the column is full and the player needs to reselect. Otherwise, the value at **`board[lowest_row][current_column]`** will be set to the player number that made the move. The screen will reflect that change.

After the piece is dropped, a helper function will be called on the board to check if either player has won[4] (four in a row). This helper function iterates over the entire board, beginning at the top left. At every location, it will check if the piece at that location is also present three more times in possible directions: right, down, diagonal down + left, and diagonal down + right.

If there is no piece at a given location it will move onto the next space. At every location it does check, it only checks directions that don't cross the boundaries of the board. Below is a visualization of which directions the board would check at different points.



[4] This function is also used in get_computer_column(), mentioned in the Choosing Implementation for single player

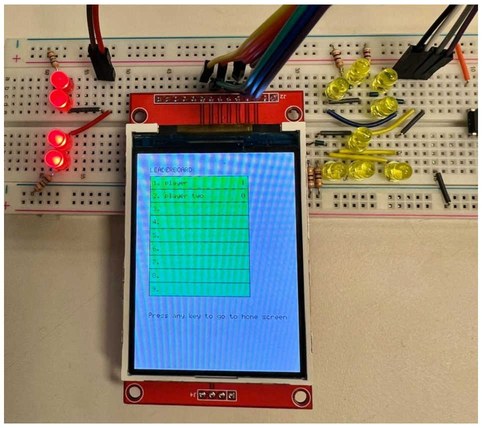
2.8 Leaderboard

refer to leaderboard_state() function in main.c and in leaderboard.c.

Leaderboard Summary (from a user's perspective)

This state is automatically entered when a player either wins the game or exits the game early. If this state was entered because a player won (and not an early exit) the light of the winning player will remain on.

It displays the list of players in order of highest score to lowest score. A user can start a new game by pressing any button on the keypad.



```
LEADERBOARD
1. player        1
2. player two    0
3.
4.
5.
6.
7.
8.
9.

Press any key to go to home screen
```

Leaderboard Implementation

The primary code involved in this section is the sorting of the list before the leaderboard is shown. After every game is completed, the **`all_players`** array is sorted, and then the array itself is just displayed to the player.

A bubble sort algorithm is used to sort the **`all_players`** array, where the algorithm makes *N* passes through the array, where *N* is the length of the array. It looks at each element and the element in front of it, and if the current element is not larger than the next element, it swaps the two.

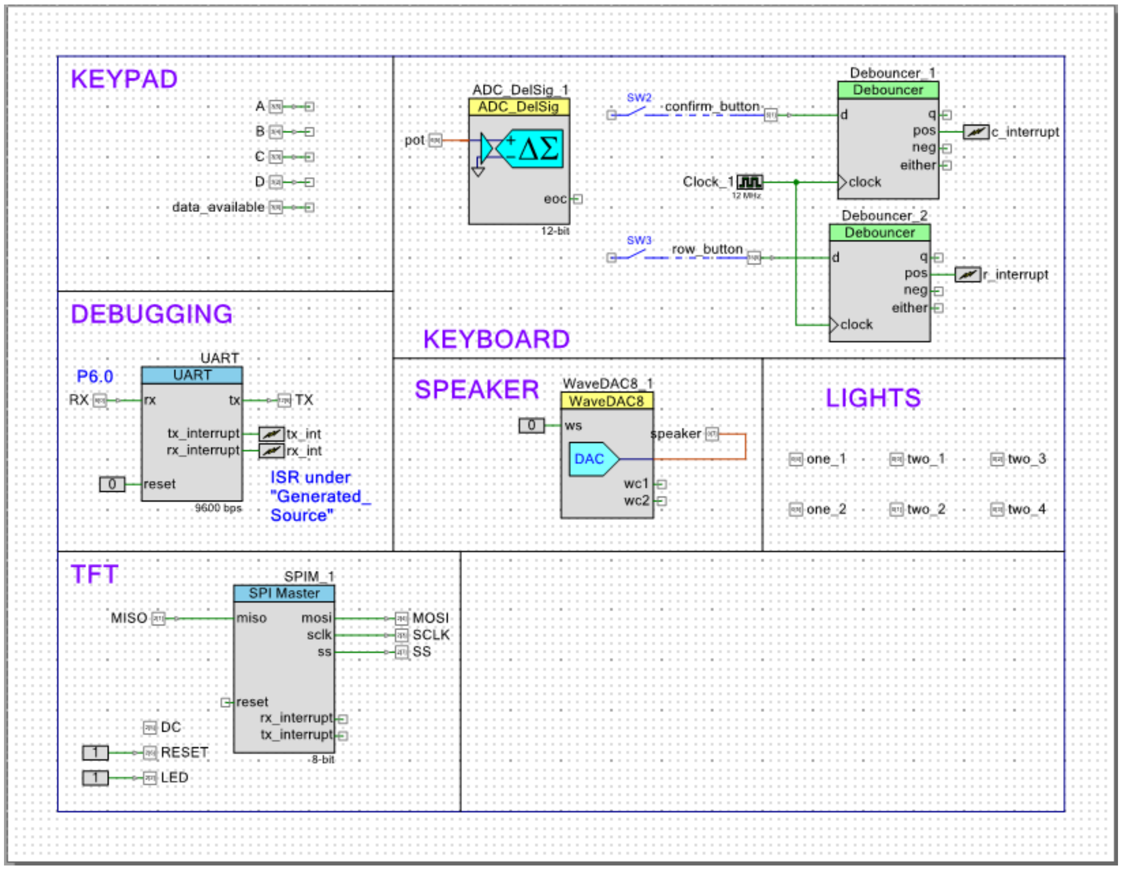
After sorting the array, the player list is displayed the same way it is in setup, but with a parameter passed in set to 1. When this parameter is 1, the text above the list says "LEADERBOARD" instead of "Here are the current players".

2.8 Next Steps (Software)

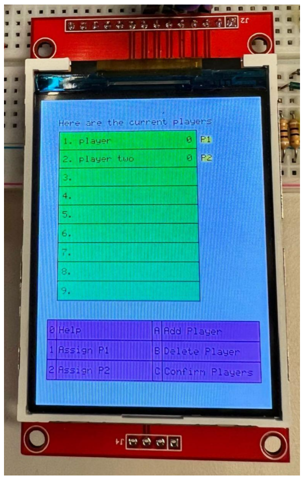
There exist a couple known ways to "break" the system (users can do something that should not be allowed). For example, the system currently allows two players to enter the same name. Implementing known improvements, along with thorough testing to discover unknown issues would help to make the software more reliable.

3. Integration

The following section covers functions and designs that integrate hardware with gameplay.
Below is a screenshot of the TopDesign from Creator for this project, to give some context into additional components of the project.



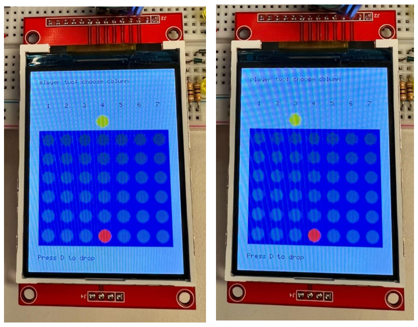
3.1 TFT



Using the SPI Master and the emWin graphics library, controlling the TFT was straightforward; most of the time was spent counting pixels and arranging items on the screen. However, pretty early in the process of working with the TFT, it was noticed that there was a visible lag, especially when drawing on large portions of the screen. An effort was consciously made to "erase" and "redraw" as little as possible.

refer to tft_setup.c
In the setup state, the functions to draw the player list, the player assignments, and the options were separate. Functions were written to only clear certain parts of the screen. Based on the action taken by the player, only certain parts needed to be redrawn. For example, when a player was assigned to P1/P2, the player list doesn't change at all, so only the player assignments are redrawn (yellow highlights on right).

refer to tft_board.c
With gameplay, the same principle was kept in mind. The large board was only drawn once. Pieces being "dropped" are drawn on top of the board, and only the preview bar is erased and redrawn when a player is choosing between columns.

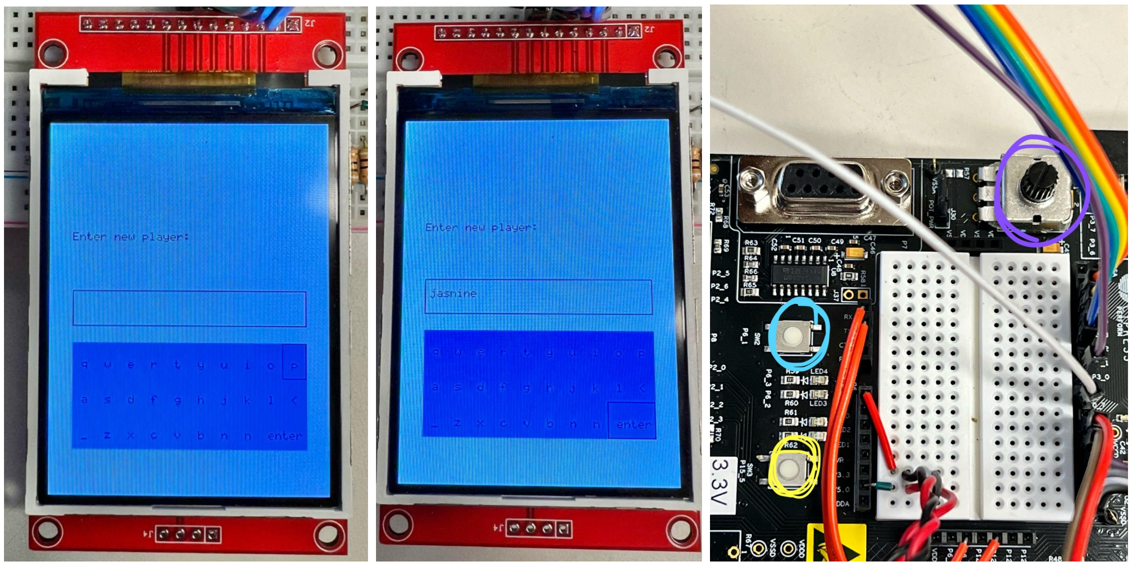


refer to tft_other.c
Remaining TFT related functions, including drawing the home and help screen, can be found in this file.

3.2 Keyboard

Keyboard Summary

The integration of the keyboard is done using the ADC and two buttons. The keyboard is represented in code as an array with 3 rows and 10 columns, with the values shown below the images.



q	w	e	r	t	y	u	i	o	p
a	s	d	f	g	h	j	k	l	< (delete)
_ (space)	z	x	c	v	b	n	m		

The user uses the potentiometer (purple circle) on the PSoC to move across a row, and a button to move down to the next row (yellow circle). There is a black rectangle drawn around the key that the user is "hovering" over.

Once the user lands on the desired key, the user should press the confirm button (blue circle) - this essentially "presses" that key. Once the entire name has been entered, the user should move to the "enter" key and press the confirm button.

Keyboard Implementation (Hardware/PSoC)

Initially, the buttons had a lot of noise, so a debouncer was added in PSoC Creator between the button and the interrupt.

Keyboard Implementation (Software)

refer to keyboard.c

The following section will go over how the keyboard is implemented with the ADC, buttons, and TFT screen. The principle of "drawing" and "erasing" as little as possible will also be shown in this section.

Initialization + Global Variables:
The "next row" and "confirm" buttons each have an interrupt handler which turns on a corresponding flag if the button was pushed. The ADC is initialized, and the entire screen is drawn. The "enter" key is actually just spaces, and the word enter is written over the key.

Global variables are used to hold the horizontal index and current row on the keyboard.

Additionally, the last key that the user was "hovering" over is saved in two ways - first, as a basic number **`prev_letter_index`** (row * 10 + horizontal index) and also as a rectangle **`last_key`** equal to the one drawn around the key. In the image on the previous page, this rectangle would be the one around the letter "p". The reasoning for this will be explained in the next section.

Getting the Player - Part 1
After initializing the screen, interrupts, and ADC, the **`keyboard_get_player()`** function also initializes local variables. Aside from obvious variables (ex. player name) it also holds a number which points to the current index in the name (starting at zero, incrementing as letters get appended).
Then, it continuously calls the **`keyboard_get_letter()`** function.

Get letter:
If the "next row" button has been pressed, the global variable holding the row is incremented and modulus by 3. If the "confirm" button has been pressed, the function simply returns - the position on the keyboard is stored in global variables.

Otherwise, the value read from the ADC is scaled to fall in a bucket from 0-9, representing the horizontal index on the screen. If the user is still on the previous letter, nothing happens. However, if they have moved to a new letter, then a few visual changes need to be made.

First, the black rectangle around the old key needs to be "erased" (function **`tft_erase_last()`**). This is effectively done by drawing the **`last_key`** rectangle in the background color of the keyboard. Then, the keyboard is updated in **`tft_update_keyboard().`** In this function, the new black rectangle is calculated - the process is the same for all keys except the enter key, which is longer than the rest.

The enter key is also the reason for saving the rectangle rather than just the indices - a check only has to be performed when landing on a key, and it doesn't have to be rechecked when "erasing" it.

Getting the Player - Part 2

After the `keyboard_get_letter()` function returns, the letter it was on is read using the indices in the global variables. In normal cases, the letter is inserted into the player name character array at the current index, and the current index is incremented. If the current index is greater than `max_players`, the player name array does not change (name lengths are <= 10). If the letter pressed is the "delete" key, the current index is decremented and the terminator character is written there. If the letter pressed is the "enter" key, the function simply clears the screen and returns to the setup state.

3.3 Speaker

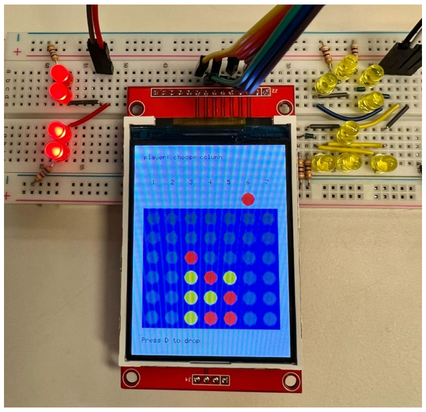
The purpose of the speaker is to play an "error" like sound when the user does something incorrectly. This not only adds effects to the game, but also assists the user in distinguishing between when the game is not responding because they have done something incorrectly versus when it may be broken.

The speaker was wired up with a LM386 to amplify the volume. The input to the LM386 was a pin from the PSoC generating a waveform at 66 Hz. When a user does something that is not allowed (ex. selecting column 9 during gameplay), the sound is played for 0.3 seconds. At this frequency and duration, it resembles an error sound, indicating negative feedback to the user.

3.4 LEDs

The LEDs are wired up in the shape of a "1" and "2", and light up to indicate which player's turn it is. The lights corresponding to the winner will also remain on in the leaderboard state.

At 3.3V, direct input from the PSoC was not enough to drive more than one LED. As a result, LEDs were wired up in parallel to pins on the PSoC, with values of 0 or 1 written depending on the player's turn.



3.5 Next Steps (Integration)

Functionality of peripherals could be increased to interact with the game in more exciting ways. For example, the speaker currently only has one functionality. It could be used to play a sound when a player wins the game, or play a sound on startup.

4. Independent Inquiry (ML Model on PSoC 6)

4.1 Overview

The goal of the independent inquiry portion of the class was to get hands-on experience with machine learning. Selected chapters from *Hands-On Machine Learning* by Aurélion Géron were read. This included all of Part I (Fundamentals of Machine Learning) and half of Part II (Neural Networks and Deep Learning).

The goal was to extend the main project by allowing users to use their voice to control the column selection in the Connect Four game. Initially, this appeared to be a problem mostly concerned with training and improving a ML model, although the scope expanded to other areas as well, including signal processing, understanding ModusToolbox compatibility, and more. A PSoC 6 was used for this portion of the project, as the increased memory and computing power was necessary to support a ML model.

The process of arriving at the current state of the project will be described below, first covering the process of developing the model, then discussing how it was integrated with the PSoC 6. This includes the preprocessing step (recording and transforming audio data) as well as the running of the model.

The system appears functional (records audio, transforms it, outputs a prediction), but the accuracy is pretty low. This section will conclude with a more complete description of the current state, and avenues to explore that could result in a model that predicted the correct values.

Only files prefixed with a * can be found in the code appendix (many of the jupyter notebooks were very long). All the code can be found on Github at https://github.com/jasmine2000/spoken_digits.

4.2 Creating the Model

The training of the model took numerous iterations. The process of arriving at the final model is described below. Existing models are referenced, so for clarity, the model being developed for this project will be referred to as the SD (Spoken Digit) model.

If referencing Github, these files are located in the "scripts" folder. Functions used in almost all models (loading and processing data, etc.) are found in basics.py.

Model V1: Basic (Envelope Detector)

*refer to V1_Envelope.ipynb and *basics.py*
The first iteration of the SD model was mostly focused on general machine learning techniques, which continued to be used in all the following models.

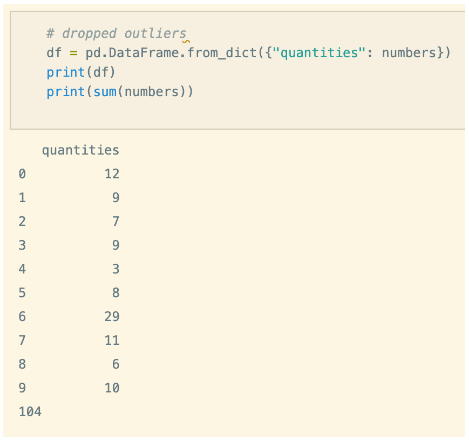
The first step was to locate or collect a dataset that would be used to train the SD model. Géron emphasized the importance of large datasets in machine learning. An open source dataset was found titled the Free Spoken Digit Dataset (FSDD)[5]. The dataset had 3000 recordings total, made up of six speakers speaking each digit 50 times. All the recordings were downloaded and copied into the project.

[5] https://github.com/Jakobovski/free-spoken-digit-dataset

The format of the data was 8kHz wav audio files with nearly no silence on either end. After loading and examining the data[6], it became apparent that while most of the data was around a length of 4000 (~half a second), there were some files with more than 15,000 data points.

The data needed to be padded, or extended with zeros, to be the same length (ML models generally receive fixed size inputs), but it seemed unreasonable to pad all the data to the longest length when it was so far above the mean. As a result, a **`max_length`** was calculated, equal to 2 standard deviations above the mean. All audio files were either padded with zeros to **`max_length`** or dropped if they exceeded the **`max_length`**.

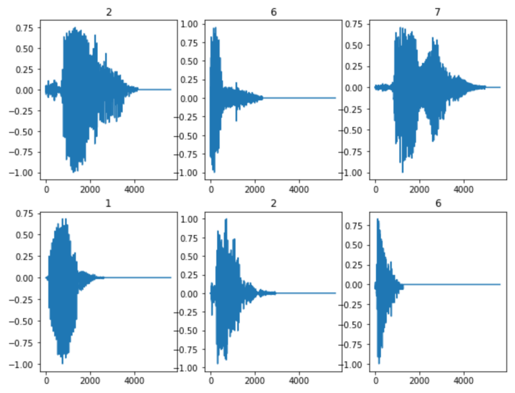
This resulted in 104 of the 3000 files being dropped, with the distribution of dropped numbers shown below. Although the number of 6s being dropped in comparison to other numbers appears alarming, it is only 10% of the 6s (300 recordings per digit), so this was noted but mostly ignored going forward.



```
# dropped outliers
df = pd.DataFrame.from_dict({"quantities": numbers})
print(df)
print(sum(numbers))
```

```
   quantities
0          12
1           9
2           7
3           9
4           3
5           8
6          29
7          11
8           6
9          10
104
```

The audio data was also normalized to ensure that differences in volume would not become correlated with predictions. On the right is a screenshot of some audio samples plotted on graphs.



Once the data was processed properly, it was split into a training set, validation set, and testing set and run through a basic classification model, largely based off of classification models shown in the textbook.

From the images of the audio samples, it may be possible to predict that this model was unsuccessful - the "envelope" of some numbers appear pretty similar (2 and 6). Even after experimenting with layers and hyperparameters, the accuracy of the SD model when evaluated on the test set was only 0.25, clearly not reliable enough to be used.

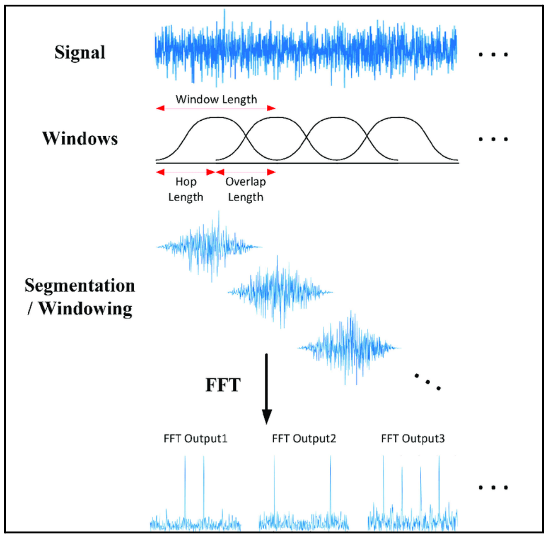
[6] loading audio function found from https://stackoverflow.com/questions/16778878/python-write-a-wav-file-into-numpy-float-array

Model V2: Using Mel Spectrogram

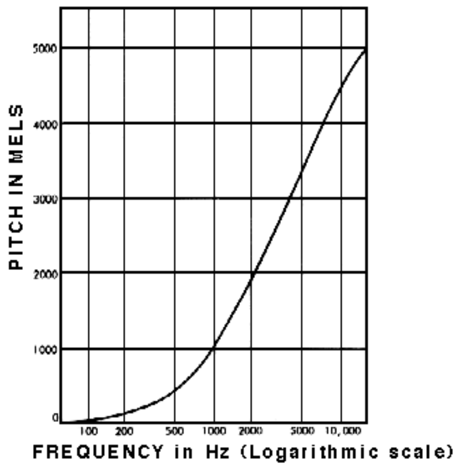
refer to V2_Mel_Spect.ipynb

Some research was then done on machine learning with spoken words, and the concept of the Mel Spectrogram emerged as a useful preprocessing step. A spectrogram is a visual representation of sound over time, and can be represented as a 2D array. To create a basic spectrogram, a short time Fourier transform is run on the data. Here is a simplified summary of STFT, with a visual on the right.

1. Begin with a window length **w** and a hop length **h**. The array containing the audio data will be called **audio**.
2. Isolate the first **w** samples (**audio**[0:**w**]). Run a Fourier transform on this window. Append this to the resulting array as a column.
3. Jump forward **h**, then from there, get the next **w** samples (**audio**[**h**:**h** + **w**]). Repeat the process.
4. A couple notes:
   a. The hop length is often shorter than the window length, so the windows overlap and no frequency data is lost.
   b. A windowing function is run on the window before the FT (especially when there is overlap).

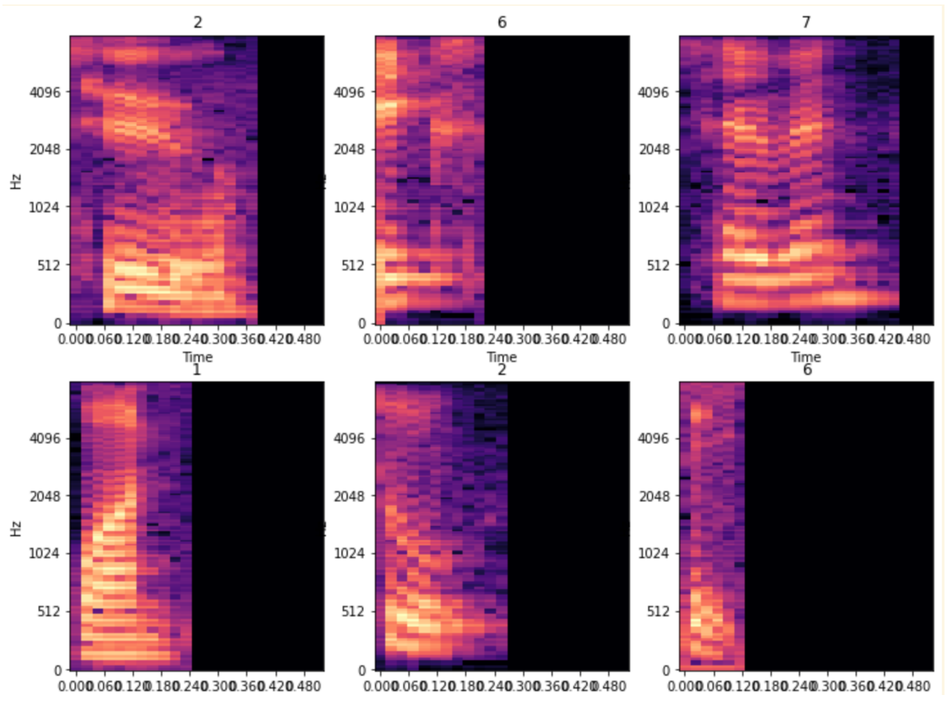


The concept of a Mel scale is based on the idea that humans perceive relative changes in frequency differently than the numerical differences - the same delta between lower frequencies seems much larger than the same delta between higher frequencies. In other words, humans perceive pitch on a logarithmic scale relative to frequency. Here is a visual of the Mel scale[7]:



[7] https://www.sfu.ca/sonic-studio-webdav/handbook/Mel.html

A Mel spectrogram is simply a combination of the two aforementioned concepts - a STFT is run on the data, and the frequencies are scaled into the Mel scale. Below are the same samples, displayed as Mel spectrograms.



The SD model itself needed to be different - the input was now a 2D array rather than a 1D one. After some research into existing audio recognition models, one was developed based on example code from the Tensorflow site. The resulting model had an accuracy of 0.96 on a test set, a satisfactory accuracy to begin loading onto the PSoC 6.

Model V3: Compatibility with ModusToolbox

refer to V3_4_DIY_Spect.ipynb

After creating a satisfactory model, the next goal was to upload it to the ModusToolbox software, which was done through a tool in ModusToolbox called the ML Configurator. The tool uploads a ML model in H5 format, then validates its accuracy with four different data formats (trading off accuracy and memory required)[8] using uploaded or random test data.

Immediately, there was an issue with uploading the SD model - the ModusToolbox ML Configurator only supported Tensorflow version 2.5.0, while the model created used multiple layers only released in version 2.6.0 (the Normalization layer and the Resizing layer). After some attempts to create custom layers that emulated these functions, it was discovered that models with custom elements were also difficult to upload to ModusToolbox. As a result, the Resizing layer was dropped, and the Normalization layer was moved to the preprocessing step (values in spectrograms were normalized before being passed into the model).

[8] formats are (in order of decreasing memory and accuracy) float, int16x16, int16x8, int8x8 where the first number is the resolution of the input data and the second number is the resolution of the model weights

There were also other new considerations as a result of using spectrograms instead of simply using the raw audio data. Although the model now trained and predicted on 2D arrays, it remained true that 1D arrays were more efficient and easier to work with in C. As a result, the model was modified so that the input was the flattened array, but a Reshaping layer was added to transform it back to 2D.

Additionally, part of the preprocessing step required the system to transform the audio from the raw version to a Mel spectrogram. Attempts to integrate libraries that could compute a Mel spectrogram were relatively unsuccessful, as they were mostly in C++. However, integrating libraries with Fourier transforms was successful. It would be possible to implement an STFT using existing implementations of Fourier transforms (in this case, Fast Fourier Transforms or FFT), so the adjustment to the Mel scale was dropped.

The preprocessing step was rewritten with a custom STFT function that called numpy's FFT function. The model was rerun with the training and testing data in this format, and maintained the same accuracy (0.96).

The final model (flattened spectrogram) uploaded and validated successfully on the desktop for all four data formats. In ModusToolbox, after a model is validated in the configurator, it can be used within a project.

However, when attempting to build the program or when validating on the device, an error that seemed to indicate a memory issue was returned. The final model created was 59.8 MB or 59,800 kB. In contrast, the model that was used as an example was 184 kB (300 times smaller). The model clearly needed to be significantly smaller in order to work with the PSoC 6.

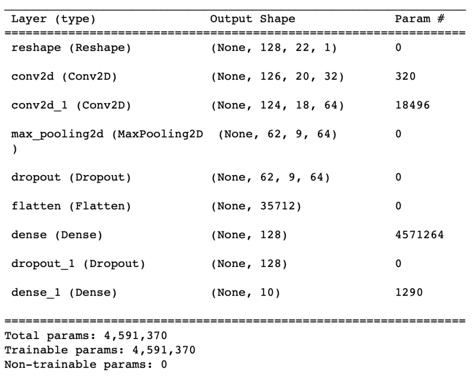
Note: The following sections will refer to rationales made based on knowledge of layer types. **Appendix A** includes a quick overview of three relevant layer types: *Convolutional 2D, Dense,* and *Pooling*. Having knowledge of these layer types is helpful in following along in the next couple sections.

Model V4: Shrinking the Model

refer to V4_2_Columns.ipynb

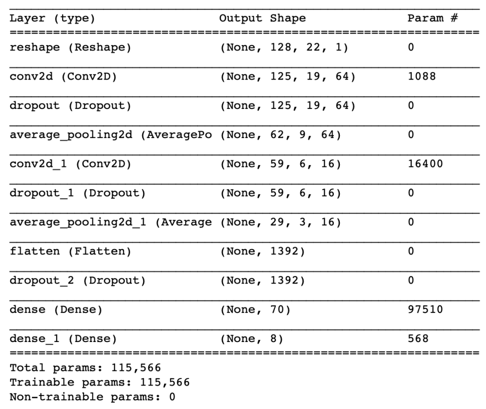
Because the classification was now happening on a spectrogram, or an image, research was done on image classification models with minimal parameters. An example based on the classic MNIST problem had only 36k parameters[9]. The previous version of the SD model had more than 4.5 million parameters, and adapting it based on this example resulted in the model being reduced to 115k parameters.

Although both models had two Convolutional 2D (Conv2D) layers, the second Conv2D layer in the smaller model had 16 filters, compared to 64 filters in the larger model. It also had an additional Pooling layer. Both of these factors reduced the size of the data significantly, which made an exponential difference in the fully connected layers (Dense layers). This can be observed in the figures below - the second to last Dense layer contributes most to the total parameter number, and is much smaller in the model on the right.



```
Layer (type)                 Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 128, 22, 1)        0

 conv2d (Conv2D)             (None, 126, 20, 32)       320

 conv2d_1 (Conv2D)           (None, 124, 18, 64)       18496

 max_pooling2d (MaxPooling2D  (None, 62, 9, 64)        0
 )

 dropout (Dropout)           (None, 62, 9, 64)         0

 flatten (Flatten)           (None, 35712)             0

 dense (Dense)               (None, 128)               4571264

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 4,591,370
Trainable params: 4,591,370
Non-trainable params: 0
```

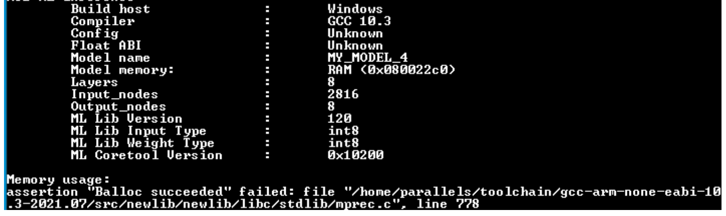
large model



```
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 128, 22, 1)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 125, 19, 64)       1088
_________________________________________________________________
dropout (Dropout)            (None, 125, 19, 64)       0
_________________________________________________________________
average_pooling2d (AveragePo (None, 62, 9, 64)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 59, 6, 16)         16400
_________________________________________________________________
dropout_1 (Dropout)          (None, 59, 6, 16)         0
_________________________________________________________________
average_pooling2d_1 (Average (None, 29, 3, 16)         0
_________________________________________________________________
flatten (Flatten)            (None, 1392)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1392)              0
_________________________________________________________________
dense (Dense)                (None, 70)                97510
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 568
=================================================================
Total params: 115,566
Trainable params: 115,566
Non-trainable params: 0
```

small model

The 1400 kB model successfully uploaded to the PSoC 6. During this step of the process, the STFT preprocessing function was also written (and will be discussed in the next section). An attempt was made to test the model by recording data from the microphone, performing the STFT transformation, and running it through the model. However, there was an issue when the function to run the model was called- the system appeared to freeze completely. After some debugging, it was discovered that the problem was still the model size- although it was small enough to be uploaded to the PSoC, the device was unable to allocate enough memory to perform computations.



```
****************************************************
MTB ML inference
        Build host                :       Windows
        Compiler                  :       GCC 10.3
        Config                    :       Unknown
        Float ABI                 :       Unknown
        Model name                :       MY_MODEL_4
        Model memory:             :       RAM (0x080022c0)
        Layers                    :       8
        Input_nodes               :       2816
        Output_nodes              :       8
        ML Lib Version            :       120
        ML Lib Input Type         :       int8
        ML Lib Weight Type        :       int8
        ML Coretool Version       :       0x10200

Memory usage:
assertion "Balloc succeeded" failed: file "/home/parallels/toolchain/gcc-arm-none-eabi-10
.3-2021.07/src/newlib/newlib/libc/stdlib/mprec.c", line 778
```

[9] https://blog.drorgluska.com/2018/06/tiny-model-for-mnist-dataset.html

Model V5 (Final): Changing Parameters and Pruning the Model

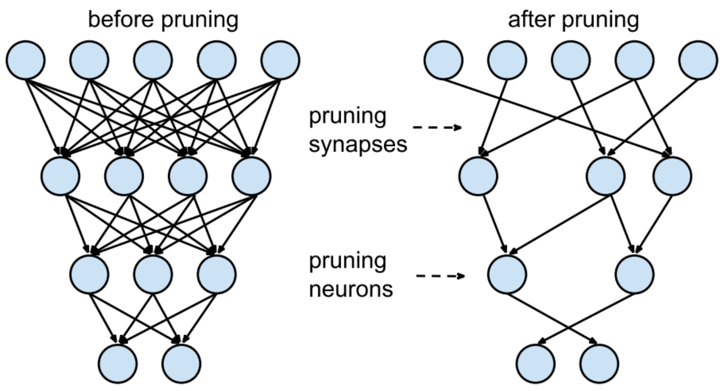
*refer to *V5_4_Params2.ipynb*

The structure of the model (arrangement and number of layers) seemed to be a good balance of size and accuracy. New strategies needed to be used to further reduce the size of the model. The first was to experiment with the customizable aspects of layers. The final model had the following changes made:

1. Reducing the number of filters in the first Conv2D layer to reduce parameters required for the second Conv2D layer.
2. Reducing the output nodes of the second to last Dense layer.

These changes lowered the number of parameters from 115k to 51k, but only reduced accuracy to 0.93.

Another common strategy used to reduce the size of a model is to prune the model, the act of removing parameters/weights that are not influential in model prediction. Below is a visual[10].



Using the Tensorflow Model Optimization package, along with sample code, pruning was performed on the reduced model (51k params). The resulting size of the model was 233 kB.

Models that only employed 1 of the 2 strategies described above were also created, but only the model that utilized both strategies was able to be uploaded and successfully initialized on the PSoC.

Even with a successful model, there was still preprocessing that had to occur on the raw data before it could be input in the model - this preprocessing is covered in the next section.

[10] https://medium.com/@souvik.paul01/pruning-in-deep-learning-models-1067a19acd89

4.3 Preprocessing Data

There were two pieces to the "preprocessing". First, the program had to decide what audio snippet to capture. Then, the program ran the audio snippet through the STFT before it was sent to the ML model.

Capturing Audio

Overview

The PSoC 6 has a built-in PDM PCM microphone, configured to read at 8kHz (same rate as training data). It included a "read async" function, which read microphone data asynchronously and triggered an interrupt when the buffer is full. The buffer size was set to 512.

A button was also used - the user would hold it down to begin recording to the microphone. However, since the audio samples used in training were trimmed to only contain the relevant data, the recording had to be intentional about starting/stopping the recording to avoid noise on either end of the sample.

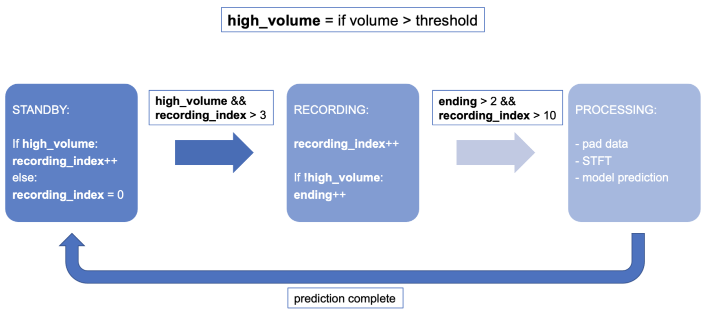
Implementation

refer to sd_src/main.c
The program also allocated an array **`voice_record`** of size 5632, or 11 * 512, to store an audio sample. Every time the interrupt flag is raised (and the button is pressed), the program did two things:

1. It copied all values from the audio sample into **`voice_record`**, beginning at **`recording_index`** * 512
2. It summed up all the values in that given audio sample

If the sum of the values exceeded a threshold, then the **`recording_index`** is incremented. The program would only transition to the recording state if the volume had exceeded the threshold for three iterations in a row. Otherwise, if the volume was low, it reset the **`recording_index`** to zero. This allowed the program to handle short amounts of loud sound without considering it as meaningful audio.

In the recording state, the **`recording_index`** incremented regardless of volume level. This allowed for fluctuations in speaking that caused the volume to temporarily dip below the threshold. To ensure that the recording also had minimal silence at the end, the recording state would also track when the volume was not greater than the threshold. If it was less than the threshold for three consecutive samples, then the program moved into the processing state and the remaining data was padded with zeros. The program would also move into the processing state if the array holding the entire recording was full. In the processing state, the data was padded, and space was allocated for the STFT result. The padded data and the allocated space were passed to the **`stft()`** function.



Processing Audio

Overview

The audio processing component required a STFT to be run on the voice_record array. A simplified implementation was created with no windowing function and where the window length was equal to the hop length (both 256). The reasoning for this was to include all data points while holding as little data as possible, because the ML model was size constrained.

A single file FFT function[11] was found and copied into the project (fft4g_h.c). Given an array to run the FFT on, the length of the array, and allocated space for computation, the function does the FFT in place. The even indices hold the real values, and the odd indices hold the imaginary values.

Implementation

*refer to *sd_src/spectrogram.c*

The spectrogram function looped over the voice_record array in chunks equal to the window length/hop length. This relevant chunk was copied into an array, and the FFT was run on this array. The magnitude of each frequency component was taken by combining the real and imaginary values. These values were stored in the return array (as it was now the right size).

The values in the spectrogram were also converted from amplitude to decibels, as this provided a more representative range of the way changes in amplitude are perceived to human ears. This was done by doing $\log_{10}$(val/max_val) for all the values in the array, where max_val was the maximum value.

An additional part of the data processing step was the normalization of the values. The mean and standard deviation for the values was computed, and used to normalize each value. The program returned to the main loop, where it would take the finished normalized STFT and run it through the ML model.

[11] https://www.kurims.kyoto-u.ac.jp/~ooura/fft.html

4.4 Running the Model

*refer to *sd_src/ml_local_regression.c*

One final "preprocessing" step had to take place before the data could be fed into the model. The data had to be reformatted into a 16 bit integer before it could be fed into the model, because the model was formatted to use 16 bit inputs and 8 bit weights (this was mentioned very briefly at the beginning of Model V3).

Fortunately, the ML package from ModusToolbox included a function that could perform this conversion. After converting the data, it was passed into the model. The output buffer was the size of the output space, where each index contained the probability that that specific index was correct. In other words, finding the index containing the largest value would be the prediction.

4.5 Current State, Known Issues and Next Steps

As mentioned in the overview, the current system is capable of recording an audio input, performing a STFT on it, and running this through the ML model. The output buffer and maximum value were displayed on the serial monitor. Unfortunately, the model predictions were nearly always incorrect.

When this portion of the project was designed, many of the challenges described above were not foreseen, from the signal processing to the model size constraint. The integration of the parts happened much later in the timeline than anticipated, leaving little time to debug each component. However, there are a few hypotheses on why the system doesn't function as expected (which should be explored, given more time).

First, the FFT function, although mostly reliable, would sometimes output either nan (not a number) or extremely large numbers for specific chunks of data. In the current state of the project, unusual values like these were detected and either thrown out or set to a default value, a strategy used to help the code run to completion but likely disrupting the accuracy of the transform.

To debug this would require closely examining the FFT code used for this portion. Although tests were done to ensure the FFT function was working properly, testing using long inputs required for the FFT was time consuming, so edge cases and unusual inputs were not explored.

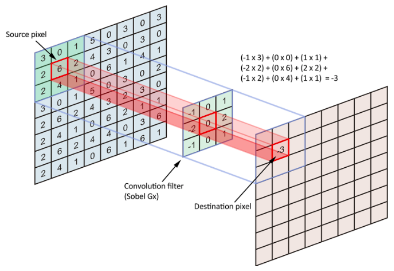
It is also possible that the issue arose in the audio data that was collected. The microphone appeared to be configured properly and accurately detected amplitude changes (the LED on the PSoC would turn on while the program was in the recording state). However it is possible that there was a non-trivial issue. The proposed strategy would be to record audio files using the microphone, and examine the data in comparison to the test files.

Final Thoughts

The project overall was exciting to design and implement, with lots of room for creativity. Although the Independent Inquiry portion of the project didn't work as planned, it was still very interesting, and I learned a lot in the process. I'm not 100% satisfied with the end product but am happy with what I was able to create, and would like to end by thanking Professor Leeb and the staff for the semester and for helping me reach this point.

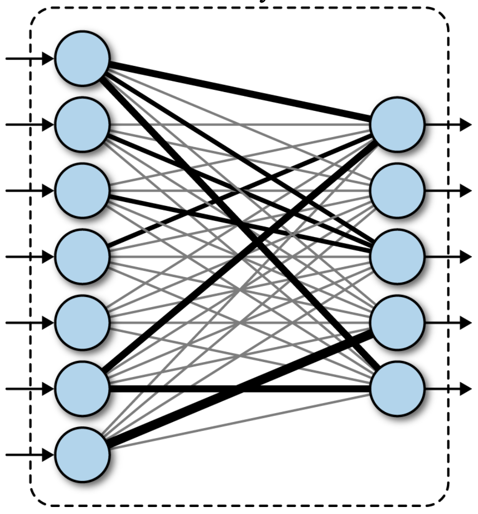
Appendix A

A little background on types of layers: Convolutional 2D (Conv2D) layers take in two parameters - the number of filters, and the kernel/filter size. The example below[12] has a single filter, with a kernel size of 3. The addition of another filter results in an additional output, and all the outputs are stacked into a 3D array. Conv2D layer output sizes are the input x and y (reduced a negligible amount from the filter) with a z equal to the number of filters. The number of parameters is linearly correlated with (number of filters) * (kernel size)^2 * (size of input's z axis).

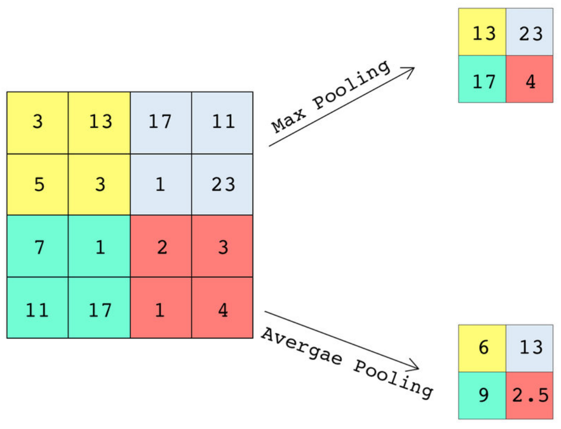

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Dense layers are just fully connected layers which take in a parameter determining the number of outputs. The number of parameters is intuitively (number of inputs) * (number of outputs). Pooling layers will run a function on a 2D window of data, and output a single number. This function could be mean, max, etc. Tensorflow Pooling layers default to a pool size of 2x2 and require no parameters. Visuals for both types of layers are shown below. There are additional layers in the model, but the aforementioned ones have the most impact on parameters required, and therefore size of the model.

dense layer[13]



pooling layer[14]



[12] https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/
[13] https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html
[14] https://www.researchgate.net/figure/Example-of-max-pooling-and-average-pooling-operations-In-this-example-a-4x4-image-is_fig4_332092821

Code Appendix

Table of Contents

(page numbers can be found in the bottom left corner)

```
MAIN.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "setup.h"
#include "gameplay.h"
#include "leaderboard.h"

#include "tft_setup.h"
#include "tft_board.h"
#include "keyboard.h"

#define max_players 9
#define name_length 11

// gameplay constants
#define HOME        0
#define SETUP       1
#define CHOOSING    2
#define DROPPING    3
#define LEADERBOARD 4

#define HELP            30
#define ASSIGN_1        1
#define ASSIGN_2        2
#define ADD_PLAYER      10
#define DELETE_PLAYER   11
#define CONFIRM_PLAYERS 12
#define GO_BACK         20

const int COLUMN_SELECT = 0; // gameplay
const int PLAYER_SELECT = 1; // adding/deleting during setup
const int OPTION_SELECT = 2; // choosing options during setup
const int CONFIRM_SELECT = 3; // confirmation
const int GAME_SELECT = 4;      // select game type

// tft constants
const int list_x = 20;
const int bottom_x = 10;
const int bottom_y = 240;
const int bottom_row_h = 20;

const int my_red = GUI_BLUE;
const int my_yellow = GUI_GREEN + GUI_BLUE;
const int my_blue = GUI_RED;

// global vars
int state;
int board[6][7];

char all_players[max_players][name_length] = {0};   // all players ever
int all_scores[max_players] = {0};                  // corresponding scores (in same order)
int num_players;                                    // total number of players at any time

int player_arr[2] = {max_players, max_players};     // index of current players (from all_players). initialized to invalid
indices on purpose
int current_player;                                 // index of current player (from player_arr)

int game_type;          // 1 or 2 depending on number of players

// gameplay vars used in choosing + dropping
int selection;
int current_column;

// tft constants
const int SCREENX = 240;
const int SCREENY = 320;
const int bottom_x = 10;
const int bottom_y = 240;
const int bottom_row_h = 20;

// state functions
void home_state();
void setup_state();
void choosing_state();
void dropping_state();
void leaderboard_state();

void init()
{
    CyGlobalIntEnable;

    UART_Start();                       // initialize UART
    UART_ClearRxBuffer();
    UART_ClearTxBuffer();

    // init tft
    SPIM_1_Start();
    //emwin
    GUI_Init();                             // initilize graphics library
    GUI_Clear();
    GUI_SetBkColor(GUI_WHITE);
    tft_clear_screen();

    // gameplay init
    num_players = 0;
    state = HOME;

}

int main() {

    init();

    for(;;) {

        switch (state) {
            case HOME:
                home_state();
                break;

            case SETUP:
                setup_state();
                break;

            case CHOOSING:
                choosing_state();
                break;

            case DROPPING:
                dropping_state();
                break;

            case LEADERBOARD:
                leaderboard_state();
                break;

        }
    }
}

void home_state() {
    tft_clear_screen();
    tft_show_home();

    for (;;) {  // while not inputting 1 or 2, stay in loop
        int action = get_keypress(GAME_SELECT, 0);
        if (action > 0) { // game init
            if (action == 1) {
                player_arr[1] = max_players; // unassign second player if previously assigned
            }

            game_type = action;
            current_player = 0;
            clear_board(board);

            tft_clear_screen();
            tft_show_players_and_scores(all_players, all_scores, num_players, 0);
```

```c
            tft_reassign(player_arr);

            state = SETUP;
            break;
        }
    }
}


void setup_state() {
    tft_show_options(game_type);
    int action = get_keypress(OPTION_SELECT, 0);

    switch (action) {
        case ASSIGN_2:
            if (game_type == 1) {
                error();        // can't select second player in single player mode
                break;
            }   // else, go into regular assign player flow
        case ASSIGN_1:
        {
            if (num_players > 0) {
                tft_clear_bottom();
                GUI_DispStringAt("Choose player to assign with keypad", list_x, bottom_y);

                CyDelay(250);
                int player_num = get_keypress(PLAYER_SELECT, num_players);
                switch (player_num) {
                    case 0:
                        error(); // Can't select that player
                        break;

                    default:
                    {
                        int player_index = player_num - 1;
                        int arr_index = action - 1;
                        player_arr[arr_index] = player_index;
                        if (player_arr[!arr_index] == player_index) {   // if player was previously set to other player
                            player_arr[!arr_index] = max_players;
                        }
                        tft_reassign(player_arr);
                    } break;
                }
            } else {
                error();
            }
        } break;

        case ADD_PLAYER:
        {
            if (num_players == max_players) {
                error(); // Can't add any more players
            } else {
                char new_player[name_length];
                tft_clear_screen();
                keyboard_get_player(new_player);

                strcpy(all_players[num_players], new_player);
                num_players ++;

                tft_show_players_and_scores(all_players, all_scores, num_players, 0);
                tft_reassign(player_arr);
            }
        } break;

        case DELETE_PLAYER:
        {
            if (num_players > 0) {

                tft_clear_bottom();
                GUI_DispStringAt("Choose player to delete with keypad", list_x, bottom_y);

                CyDelay(250);
                int delete_number = get_keypress(PLAYER_SELECT, num_players);
                switch (delete_number) {
```

```
                case 0:
                    error(); // Can't delete that");
                    break;
                default:
                {
                    int delete_index = delete_number - 1;

                    GUI_DispStringAt("Deleting: ", list_x, bottom_y + 1.5 * bottom_row_h);
                    GUI_DispStringAt(all_players[delete_index], list_x + 100, bottom_y + 1.5 * bottom_row_h);

                    GUI_DispStringAt("Press C to confirm delete", list_x, bottom_y + 2.5 * bottom_row_h);
                    GUI_DispStringAt("Press any other key to cancel", list_x, bottom_y + 3 * bottom_row_h);

                    CyDelay(250);
                    int c = get_keypress(CONFIRM_SELECT, 0);
                    if (c == 1) {
                        do_delete(all_players, all_scores, player_arr, delete_index);
                        num_players --;

                        tft_clear_screen();
                        tft_show_players_and_scores(all_players, all_scores, num_players, 0);
                        tft_reassign(player_arr);
                    } else {
                        tft_clear_bottom();
                    };

                } break;
            }
        } else {
            error();
    } break;

    case CONFIRM_PLAYERS:
    {
        if (player_arr[0] == max_players) {
            error(); // P1 must be set always
        } else if (player_arr[1] == max_players && game_type == 2) {
            error(); // P2 must be set for 2 player
        } else if (player_arr[0] == player_arr[1]) {
            error(); // Player 1 and 2 are the same. Reassign one of them
        } else {
            tft_clear_bottom();

            GUI_DispStringAt("Player 1: ", list_x, 240);
            GUI_DispStringAt(all_players[player_arr[0]], 100, 240);

            if (game_type == 2) {
                GUI_DispStringAt("Player 2: ", list_x, 260);
                GUI_DispStringAt(all_players[player_arr[1]], 100, 260);
            }

            GUI_DispStringAt("Press C to confirm players", list_x, 280);
            GUI_DispStringAt("Press any other key to cancel", list_x, 300);

            CyDelay(250);
            int c = get_keypress(CONFIRM_SELECT, 0);
            if (c == 1) {
                current_player = 0;
                selection = 4;
                state = CHOOSING;

                tft_clear_screen();
                tft_init_board();
                player_lights(current_player);
            } else {
                tft_clear_bottom();
            };
        }
    } break;

    case GO_BACK:
    {
        memset(player_arr, max_players, sizeof player_arr); // empty player assignments
        state = HOME;
    } break;
```

```
        case HELP:
        {
            tft_clear_screen();
            tft_help_screen();
            get_keypress(CONFIRM_SELECT, 0);    // just waiting for keypress

            tft_clear_screen();
            tft_show_players_and_scores(all_players, all_scores, num_players, 0);
            tft_reassign(player_arr);
        }

        default:
            error(); // printf("Unrecognized action.\n");
            break;
        }
    }
}

void choosing_state() {

    if (game_type == 1 && current_player == 1) {        // single player and its the computers turn
        current_column = get_computer_column(board);    // set column
        CyDelay(500);
        selection = 13;                                 // drop piece
    }

    tft_player_turn(all_players[player_arr[current_player]]);   // show whos turn

    switch (selection) {

        case 13:                // D on keypad
            state = DROPPING;
            break;

        case 20:                // * on keypad (exit)
        {
            CyDelay(250);
            GUI_SetColor(GUI_WHITE);
            GUI_FillRect(10, 290, 240, 320);
            GUI_SetColor(GUI_BLACK);
            GUI_SetBkColor(my_yellow);
            GUI_DispStringAt("Press C to confirm exit: ", 15, 295);
            GUI_SetBkColor(GUI_WHITE);
            int c = get_keypress(CONFIRM_SELECT, 0);
            switch (c) {
                case 1:
                    state = LEADERBOARD;
                    player_lights(2);
                    tft_clear_screen();
                    break;

                default:
                    GUI_SetColor(GUI_WHITE);
                    GUI_FillRect(10, 290, 240, 320);
                    GUI_SetColor(GUI_BLACK);
                    GUI_DispStringAt("Press D to drop", 15, 295);
                    selection = current_column; // set back to last valid entry
                    break;
            }
        } break;

        default:
        {
            if (selection == 0) {
                error(); // Invalid move
            } else {
                current_column = selection;     // set current column if valid column entry

                // update display
                // temp_printboard(board, current_column, current_player + 1);
                tft_preview_choice(current_column, current_player);  // update preview
            }
            CyDelay(250);
            selection = get_keypress(COLUMN_SELECT, 0); // get next user input
        } break;
    }
}

void dropping_state() {
    int column_index = current_column - 1;          // map column value to index
    int lowest_row = find_row(board, column_index); // get row it would fall into

    switch (lowest_row) {
        case 6:
        {
            error();        // column is full
            selection = 4;
            state = CHOOSING;
        } break;

        default:
        {
            // assign value and show on display
            board[lowest_row][column_index] = current_player + 1;
            tft_drop_chip(lowest_row, column_index, current_player);

            int winner = check_winner(board);   // check if someone won
            int winner_index = winner - 1;
            if (winner == 3) {          // 3 == draw
                tft_win_message(winner, "");
                player_lights(2);

            } else if (winner > 0) {   // 1 or 2 = winner
                all_scores[player_arr[winner_index]] ++;
                tft_win_message(winner_index, all_players[player_arr[winner_index]]);   // TODO
                CyDelay(2000);
                tft_clear_screen();

                player_lights(winner_index);
                state = LEADERBOARD;

            } else {
                current_player = (current_player + 1) % 2;      // increment and mod for player index
                selection = 4;

                player_lights(current_player);
                state = CHOOSING;
            }
        } break;
    }
}

void leaderboard_state() {

    sort_names_scores(all_players, all_scores, player_arr); // sort so in order

    tft_show_players_and_scores(all_players, all_scores, num_players, 1);

    GUI_DispStringAt("Press any key to go to home screen", list_x, bottom_y);
    get_keypress(CONFIRM_SELECT, 0);
    player_lights(2);
    state = HOME;
}
```

```
KEYPAD.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

extern const int max_players;

extern const int COLUMN_SELECT; // gameplay
extern const int PLAYER_SELECT; // adding/deleting during setup
extern const int OPTION_SELECT; // choosing options during setup
extern const int CONFIRM_SELECT;    // confirming a choice
extern const int GAME_SELECT;       // game type

const int keypad_map[16] = {
    1,  2,  3,  10,
    4,  5,  6,  11,
    7,  8,  9,  12,
    20, 0,  21, 13
};

int get_keypress(int mode, int next) {

    int data_available;
    int keypad_index;
    int key;

    for (;;) {
        data_available = data_available_Read();
        if (data_available == 1) {
            keypad_index = A_Read() + (B_Read() << 1) + (C_Read() << 2) + (D_Read() << 3);
            break;
        }
    }

    key = keypad_map[keypad_index];

    if (mode == COLUMN_SELECT) {            // 1-7, D, *
        if ((key >= 1 && key <= 7) || key == 13 || key == 20) {
            return key;
        } else {
            return 0;
        }

    } else if (mode == PLAYER_SELECT) {     // 1-next
        if (key > 0 && key <= next) {
            return key;
        } else {
            return 0;
        }

    } else if (mode == OPTION_SELECT) {     // 0-2, A-C, *
        if (key == 1 || key == 2 || (key >= 10 && key <= 12) || key == 20 || key == 30) {
            return key;
        } else {
            return 0;
        }

    } else if (mode == CONFIRM_SELECT) {    // C only
        if (key == 12) {
            return 1;
        } else {
            return 0;
        }

    } else if (mode == GAME_SELECT) {       // 1 or 2
        if (key == 1 || key == 2) {
            return key;
        } else {
            return 0;
        }
    }
    return 0;
}
```

7

```
SETUP.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "setup.h"


void clear_board(int board[6][7]) {
    // makes board all zeros
    int i, j;
    for (i = 0; i < 6; i=i+1) {
        for (j = 0; j < 7; j++) {
            board[i][j] = 0;
        }
    }
}

void do_delete(char all_players[10][11], int all_scores[max_players], int player_arr[2], int player_index) {
    // delete player and shift all back
    int i, j;
    for (i = player_index + 1; i < max_players; i ++) {
        strcpy(all_players[i - 1], all_players[i]); // move everything one back
        all_scores[i - 1] = all_scores[i];
        for (j = 0; j < 2; j++) {
            if (player_arr[j] == i) {
                player_arr[j]--;
            }
        }
    }
    all_players[max_players - 1][0] = '\0';
    all_scores[max_players - 1] = 0;

    for (j = 0; j < 2; j++) {
        if (player_arr[j] == player_index) {
            player_arr[j] = max_players;
        }
    }
}
```

GAMEPLAY.C

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "gameplay.h"


int find_row(int board[6][7], int col) {
    // returns index of lowest unoccupied row
    // returns 6 if all rows are occupied
    int i;
    for (i = 0; i < 6; i++) {
        if (board[i][col] == 0) {
            return i;
        }
    }
    return 6;
}

int check_winner(int board[6][7]) {
    // returns 0 if no one won
    // 1 if player 1 or 2 if player 2 won
    // 3 for draw
    int i, j, k;
    int right, left, vertical;      // if enough space in directions to have four
    int d_right, d_left;
    int player;
    int test_chip = 0;
    for (i = 0; i < 6; i++) {
        for (j = 0; j < 7; j++) {
            right = 0;
            left = 0;
            vertical = 0;
            test_chip = 0;

            player = board[i][j];
            if (player  == 3) {
                test_chip = 1;
            }

            if (player != 0) {
                if (j + 3 <= 6) right = 1;
                if (j - 3 >= 0) left = 1;
                if (i + 3 <= 5) vertical = 1;

                d_right = right && vertical;
                d_left = left && vertical;

                int current_val;
                if (test_chip && board[i][j + 1] != 0) {
                    player = board[i][j + 1];
                }
                if (right) {
                    for (k = 1; k < 4; k++) {
                        current_val = board[i][j + k];
                        if (current_val != player && current_val != 3) {
                            right = 0;
                            break;
                        }
                    }
                }
                if (right) return player;

                if (test_chip && board[i + 1][j] != 0) {
                    player = board[i + 1][j];
                }
                if (vertical) {
                    for (k = 1; k < 4; k++) {
                        current_val = board[i + k][j];
                        if (current_val != player && current_val != 3) {
                            vertical = 0;
                            break;
                        }
                    }
                }
                if (vertical) return player;

                if (test_chip && board[i + 1][j + 1] != 0) {
                    player = board[i + 1][j + 1];
                }
                if (d_right) {
                    for (k = 1; k < 4; k++) {
                        current_val = board[i + k][j + k];
                        if (current_val != player && current_val != 3) {
                            d_right = 0;
                            break;
                        }
                    }
                }
                if (d_right) return player;

                if (test_chip && board[i + 1][j - 1] != 0) {
                    player = board[i + 1][j - 1];
                }
                if (d_left) {
                    for (k = 1; k < 4; k++) {
                        current_val = board[i + k][j - k];
                        if (current_val != player && current_val != 3) {
                            d_left = 0;
                            break;
                        }
                    }
                }
                if (d_left) return player;
            }
        }
    }

    // no winner, looking for full board
    for (i = 0; i < 7; i++) {
        if (board[5][i] == 0) return 0;
    }
    return 3;       // draw
}

int get_computer_column(int board[6][7]) {
    // will return a column that would result in anyone winning
    // if one doesn't exist, returns random column in range 7

    int row, col, winner_result;
    for (col = 0; col < 7; col++) {
        row = find_row(board, col);
        if (row == 6) continue;

        board[row][col] = 3;
        winner_result = check_winner(board);
        board[row][col] = 0;
        if (winner_result > 0) return col + 1;
    }

    int random_num = rand();
    int random_col = (random_num % 7) + 1;
    return random_col;
}
```

```
LEADERBOARD

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include "leaderboard.h"

void sort_names_scores(char all_players[max_players][11], int all_scores[max_players], int player_arr[2]) {
    // sorts all players and scores from
    // highest to lowest
    int i, j, k;
    int sorted;
    char lastplayer[11];
    char player[11];

    for (i = 0; i < max_players - 1; i++) {
        sorted = 1;
        memset(lastplayer, 0, sizeof lastplayer);
        strcpy(lastplayer, all_players[0]); // first player

        for (j = 1; j < max_players; j++) {
            memset(player, 0, sizeof player);
            strcpy(player, all_players[j]); // get player at index
            if (player[0] == '\0') {        // if empty
                break;
            }
            if (all_scores[j] > all_scores[j - 1]) {
                strcpy(all_players[j], lastplayer); // swap players
                strcpy(all_players[j - 1], player);

                int temp = all_scores[j];           // swap scores
                all_scores[j] = all_scores[j - 1];
                all_scores[j - 1] = temp;

                sorted = 0;
                // lastplayer is same

                for (k = 0; k < 2; k ++) {
                    if (player_arr[k] == j) {
                        player_arr[k] = j - 1;
                    } else if (player_arr[k] == j - 1) {
                        player_arr[k] = j;
                    }
                }

            } else {
                memset(lastplayer, 0, sizeof lastplayer);
                strcpy(lastplayer, player);
            }
        }
        if (sorted == 1) break;
    }
}
```

```
KEYBOARD.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include<project.h>
#include<keyboard.h>
#include<tft_other.h>

#include "GUI.h"

// flags
int confirm_flag;
int row_flag;

int h_index;
int row_number;

const int keyboard_color = GUI_LIGHTRED;

const int KEYWIDTH = 20;
const int KEYHEIGHT = 30;
const int KEYBOARD_W = 200;
const int KEYBOARD_X = 20;
const int KEYBOARD_Y = 190;

GUI_RECT last_key;

char keyboard[3][11] = {"qwertyuiop", "asdfghjkl<", "_zxcvbnm  "};

void tft_init_keyboard();
void tft_update_name(char player[11]);
void tft_erase_last();
void tft_update_keyboard();

// INTERRUPTS
CY_ISR( confirm_button_Handler ) {
    confirm_flag = 1;
}

CY_ISR( row_button_Handler ) {
    row_flag = 1;
}


void keyboard_init() {
    // init ADC
    ADC_DelSig_1_Start();                               // start the ADC_DelSig_1
        ADC_DelSig_1_StartConvert();            // start the ADC_DelSig_1 conversion

    // init interrupts
    c_interrupt_StartEx(confirm_button_Handler);
    r_interrupt_StartEx(row_button_Handler);

    // interrupt flags
    confirm_flag = 0;
    row_flag = 0;

    // init keyboard indices
    row_number = 0;
    h_index = 0;

    last_key = (LCD_RECT){
        .x0 = KEYBOARD_X,
        .y0 = KEYBOARD_Y,
        .x1 = KEYBOARD_X + KEYWIDTH,
        .y1 = KEYBOARD_Y + KEYHEIGHT};

    tft_init_keyboard();
}

void keyboard_get_letter() {

    int letter_index;
    int prev_letter_index = 0;

    int adc_ready;
    int16 adcResult;

    for(;;)
    {
        adc_ready = ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT);

        if (row_flag) {
            row_flag = 0;

            row_number ++;
            row_number = row_number % 3;
        }
        if (confirm_flag) {
            confirm_flag = 0;

            return;
        }

                if (adc_ready) {
                        adcResult = ADC_DelSig_1_GetResult16();         // read the adc and assign the value adcResult

            if (adcResult & 0x8000)
            {
                adcResult = 0;       // ignore negative ADC results
            }
            else if (adcResult >= 0xfff)
            {
                adcResult = 0xfff;      // ignore high ADC results
            }

            h_index = (int)(adcResult / 410);

            letter_index = row_number * 10 + h_index;
            if (letter_index != prev_letter_index) {
                tft_erase_last();
                tft_update_keyboard();
                prev_letter_index = letter_index;
            }

            CyDelay(50);                                                                            // delay in milliseconds
                }
    }
}

void keyboard_get_player(char player_name[11]) {
    keyboard_init();

    memset(player_name, 0, 11);
    char next_letter;
    int name_index = 0;
    for (;;) {
        keyboard_get_letter();
        next_letter = keyboard[row_number][h_index];

        if (next_letter == '<') {           // delete
            if (name_index > 0) {
                name_index --;
                player_name[name_index] = '\0';
            }

        } else if (next_letter == ' ') {    // finish
            tft_clear_screen();
            return;

        } else {
            if (next_letter == '_') {
                next_letter = ' ';
            }
            if (name_index < 10) {
                player_name[name_index] = next_letter;
                name_index ++;
            }
        }

        tft_update_name(player_name);
    }
}

void tft_init_keyboard() {
    GUI_SetColor(keyboard_color);
    GUI_FillRect(KEYBOARD_X, KEYBOARD_Y, KEYBOARD_X + KEYBOARD_W, KEYBOARD_Y + KEYHEIGHT * 3);

    GUI_SetBkColor(keyboard_color);
    GUI_SetColor(GUI_BLACK);
    int i, j;
    int charX, charY;
    for (i = 0; i < 3; i++) {
        charY = KEYBOARD_Y + KEYHEIGHT / 2 + KEYHEIGHT * i;
        for (j = 0; j < 10; j++) {
            charX = KEYBOARD_X + KEYWIDTH / 2 + j * KEYWIDTH - 2;
            if (i == 2 && j > 7) break;
            GUI_DispCharAt(keyboard[i][j], charX, charY);
        }
    }
    GUI_DispStringAt("enter", charX, charY);

    GUI_DrawRect(20, 145, 220, 175);
    GUI_SetBkColor(GUI_WHITE);
    GUI_DispStringAt("Enter new player:", KEYBOARD_X, KEYBOARD_Y / 2);
}

void tft_update_name(char player_name[11]) {
    GUI_RECT name_rect = (LCD_RECT){.x0 = 25, .y0 = 150, .x1 = 210, .y1 = 165};

    char displayed_name[21];
    sprintf(displayed_name, "%s          ", player_name);
    GUI_DispStringInRect(displayed_name, &name_rect, GUI_TA_LEFT | GUI_TA_VCENTER);
}

void tft_erase_last() {
    GUI_SetColor(keyboard_color);
    GUI_DrawRectEx(&last_key);
    GUI_SetColor(GUI_BLACK);
}

void tft_update_keyboard() {
    int rectX = KEYBOARD_X + h_index * KEYWIDTH;
    int rectY = KEYBOARD_Y + row_number * KEYHEIGHT;

    GUI_RECT new_key = (LCD_RECT){
            .x0 = rectX,
            .y0 = rectY,
            .x1 = rectX + KEYWIDTH,
            .y1 = rectY + KEYHEIGHT};

    if (row_number == 2 && h_index > 7) {
        new_key.x0 = KEYBOARD_X + 8 * KEYWIDTH;
        new_key.x1 = KEYBOARD_X + 10 * KEYWIDTH;
    }

    GUI_DrawRectEx(&new_key);
    last_key = new_key;

}
```

```
TFT_SETUP.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include<project.h>

#include "tft_setup.h"
#include "GUI.h"

const int list_x = 20;
const int list_y = 35;
const int row_h = 20;
const int row_w = 150;
const int options_w = 220;
const int options_x = 10;
const int options_y = 235;
const int option_num_w = 10;


void tft_clear_bottom() {
    GUI_SetColor(GUI_WHITE);
    GUI_FillRect(0, options_y, 240, 320);
    GUI_SetColor(GUI_BLACK);
}

void tft_show_players(char all_players[max_players][name_length], int next) {
    int writeX = list_x;
    int writeY = list_y - row_h / 2 - 3;

    if (next == 0) { // next available index
        GUI_DispStringAt("No players yet", writeX, writeY);
        return;
    } else {
        GUI_DispStringAt("Here are the current players", writeX, writeY);
    }

    GUI_COLOR background = GUI_LIGHTGREEN;

    GUI_SetColor(background);
    GUI_FillRect(list_x, list_y, list_x + row_w, list_y + max_players * row_h);
    GUI_SetColor(GUI_BLACK);
    GUI_DrawRect(list_x, list_y, list_x + row_w, list_y + max_players * row_h);

    GUI_SetBkColor(background);

    char player[name_length + 5];
    int i;

    writeX = list_x + 5;
    for (i = 0; i < max_players; i ++) {
        memset(player, 0, sizeof player);
        sprintf(player, "%d. %s", i + 1, all_players[i]);

        writeY = list_y + row_h / 2 + i * row_h - 3;
        GUI_DispStringAt(player, writeX, writeY);
        GUI_DrawLine(list_x, list_y + i * row_h, list_x + row_w, list_y + i * row_h);
    }
}

void tft_reassign(int player_arr[2]) {
    GUI_COLOR player_highlight = GUI_LIGHTCYAN;

    int i, j;
    int writeX, writeY;
    writeX = list_x + row_w + 5;
    for (i = 0; i < max_players; i ++) {
        writeY = list_y + row_h / 2 + i * row_h - 3;
        GUI_DispStringAt("   ", writeX + row_w, writeY); // clear previous

        for (j = 0; j < 2; j++) {
            if (player_arr[j] == i) {
                char p_num[5];
                sprintf(p_num, "P%d", j + 1);
                GUI_SetBkColor(player_highlight);
                GUI_DispStringAt(p_num, writeX, writeY);
                GUI_SetBkColor(GUI_WHITE);
            }
        }
    }

}

void tft_show_options(int game_type) {
    GUI_COLOR background = GUI_LIGHTMAGENTA;
    GUI_SetColor(background);
    GUI_FillRect(options_x, options_y, options_x + options_w, options_y + 3 * row_h);
    GUI_SetColor(GUI_BLACK);
    GUI_DrawRect(options_x, options_y, options_x + options_w, options_y + 3 * row_h);

    GUI_SetBkColor(background);

    int i;
    for (i = 1; i < 3; i++) {
        GUI_DrawLine(options_x, options_y + i * row_h, options_x + options_w, options_y + i * row_h);
    }

    GUI_DrawLine(options_x + options_w / 2, options_y, options_x + options_w / 2, options_y + 3 * row_h);
    for (i = 0; i < 2; i++) {
        int xVal = options_x + i * options_w / 2 + option_num_w;
        GUI_DrawLine(xVal, options_y, xVal, options_y + 3 * row_h);
    }

    char option_nums[] = "012ABC";
    char option_msg[6][20] = {{"Help"}, {"Assign P1"}, {"Assign P2"}, {"Add Player"}, {"Delete Player"}, {"Confirm Players"}};

    int j;
    int writeX, writeY;
    for (i = 0; i < 2; i++) {
        writeX = options_x + i * options_w / 2 + 3;
        for (j = 0; j < 3; j++) {
            writeY = options_y + row_h / 2 + j * row_h - 3;
            GUI_DispCharAt(option_nums[i * 3 + j], writeX, writeY);
            GUI_DispStringAt(option_msg[i * 3 + j], writeX + option_num_w, writeY);
        }
    }
    GUI_SetBkColor(GUI_WHITE);

    if (game_type == 1) {
        GUI_SetColor(GUI_WHITE);
        GUI_FillRect(options_x, options_y + 2 * row_h, options_x + optioons_w / 2, options_y + 3 * row_h);
        GUI_SetColor(GUI_BLACK);
    }
}
```

```
TFT_BOARD.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include<project.h>

#include "tft_board.h"
#include "GUI.h"


const int BOARDX = 15;
const int BOARDY = 100;
const int BOARDW = 210;
const int BOARDH = 180;

const int SIDELENGTH = 30;

extern const int my_red;
extern const int my_yellow;
extern const int my_blue;


void tft_init_board()
{
    GUI_SetColor(my_blue);
    GUI_FillRect(BOARDX, BOARDY, BOARDX + BOARDW, BOARDY + BOARDH);

    GUI_SetColor(GUI_GRAY);
    int i, j;
    for (i = 0; i < 6; i++) {
        for (j = 0; j < 7; j++) {
            int centerX = BOARDX + SIDELENGTH / 2 + j * SIDELENGTH;
            int centerY = BOARDY + SIDELENGTH / 2 + i * SIDELENGTH;
            GUI_FillCircle(centerX, centerY, SIDELENGTH / 3);
        }
    }

    char columns[] = "1234567";
    int numbers_x;
    int numbers_y = BOARDY - (SIDELENGTH * 3 / 2);
    for (i = 0; i < 7; i++) {
        numbers_x = BOARDX + SIDELENGTH / 2 + i * SIDELENGTH - 2;
        GUI_DispCharAt(columns[i], numbers_x, numbers_y);
    }
    GUI_SetColor(GUI_BLACK);

    GUI_DispStringAt("Press D to drop", BOARDX, BOARDY + BOARDH + SIDELENGTH / 2);
}


void tft_player_turn(char player_name[11]) {
    GUI_SetColor(GUI_WHITE);
    GUI_FillRect(BOARDX, 0, BOARDX + BOARDW, SIDELENGTH);

    GUI_SetColor(GUI_BLACK);
    char message[30];
    sprintf(message, "%s: choose column", player_name);
    GUI_DispStringAt(message, BOARDX, SIDELENGTH / 2);
}


void tft_preview_choice(int column, int player_num) {
    GUI_SetColor(GUI_WHITE);
    GUI_COLOR chip_color;
    if (player_num == 0) {
        chip_color = my_red;
    } else {
        chip_color = my_yellow;
    }
    int i;
    int preview_x;
    int preview_y =  BOARDY - SIDELENGTH / 2;
    for (i = 0; i < 7; i++) {
        preview_x = BOARDX + SIDELENGTH / 2 + i * SIDELENGTH;
        if (column == i + 1) {
            GUI_SetColor(chip_color);
        }
        GUI_FillCircle(preview_x, preview_y, SIDELENGTH / 3);
        if (column == i + 1) {
            GUI_SetColor(GUI_WHITE);
        }
    }
    GUI_SetColor(GUI_BLACK);
}


void tft_drop_chip(int row, int column, int player_num) {
    GUI_COLOR chip_color;
    if (player_num == 0) {
        chip_color = my_red;
    } else {
        chip_color = my_yellow;
    }
    int centerX = BOARDX + SIDELENGTH / 2 + column * SIDELENGTH;
    int centerY = BOARDY + SIDELENGTH / 2 + (5 - row) * SIDELENGTH;
    GUI_SetColor(chip_color);
    GUI_FillCircle(centerX, centerY, SIDELENGTH / 3);
    GUI_SetColor(GUI_BLACK);
}
```

```
TFT_OTHER.C

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#include<project.h>

#include "tft_other.h"
#include "GUI.h"

#ifndef max_players
    #define max_players 9
#endif
#ifndef name_length
    #define name_length 11
#endif

const int SCREENX = 240;
const int SCREENY = 320;

extern const int my_red;
extern const int my_yellow;
extern const int my_blue;


void tft_clear_screen() {
    GUI_SetColor(GUI_WHITE);
    GUI_FillRect(0, 0, SCREENX, SCREENY);
    GUI_SetColor(GUI_BLACK);
}

void tft_win_message(int winner, char winner_name[11]) {
    GUI_RECT win_rect = (LCD_RECT){.x0 = 40, .y0 = 50, .x1 = 200, .y1 = 220};
    GUI_COLOR win_color;
    if (winner == 0) {
        win_color = my_red;
    } else if (winner == 1) {
        win_color = my_yellow;
    } else {
        win_color = my_blue;
    }

    GUI_SetColor(win_color);
    GUI_SetBkColor(win_color);
    GUI_FillRectEx(&win_rect);
    GUI_SetColor(GUI_BLACK);

    char win_msg[20];
    if (strlen(winner_name) == 0) {
        strcpy(win_msg, "DRAW");
    } else {
        sprintf(win_msg, "%s won!", winner_name);
    }
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringInRect(win_msg, &win_rect, GUI_TA_HCENTER | GUI_TA_VCENTER);
    GUI_SetBkColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font6x8);
}

void tft_show_home() {
    int radius = 20;
    int logo_x = 60;
    int logo_y = 80;
    int logo_w = 120;

    int text_width = 8;
    int text_height = 16;
    int text_y = logo_y + radius - text_height / 2;

    int instruction_y = logo_y + 3 * radius;

    GUI_SetColor(my_red);
    GUI_FillRoundedRect(logo_x, logo_y, logo_x + logo_w, logo_y + 2 * radius, radius);
    GUI_SetFont(&GUI_Font8x16);
    GUI_SetColor(my_yellow);
    GUI_SetBkColor(my_red);
    GUI_DispStringAt("Connect", logo_x + radius, text_y);

    GUI_FillCircle(logo_x + logo_w - radius, logo_y + radius, radius - 3);

    GUI_SetColor(my_red);
    GUI_SetBkColor(my_yellow);
    GUI_DispCharAt('4', logo_x + logo_w - radius - text_width / 2, text_y);

    GUI_SetColor(GUI_BLACK);
    GUI_SetBkColor(GUI_WHITE);

    GUI_SetFont(&GUI_Font6x8);
    GUI_DrawRoundedRect(logo_x, instruction_y, logo_x + logo_w, instruction_y + 3 * radius, 3);

    int writeX = logo_x + 6;
    int writeY = instruction_y + 6;
    GUI_DispStringAt("Press", writeX, writeY);

    writeY = writeY + radius;
    GUI_DispStringAt("1\tSingle Player", writeX, writeY);
    GUI_DrawCircle(writeX + 2, writeY + 4, 6);

    writeY = writeY + radius;
    GUI_DispStringAt("2\tTwo Player", writeX, instruction_y + 2 * radius + 6);
    GUI_DrawCircle(writeX + 2, writeY + 4, 6);

    GUI_SetFont(&GUI_Font6x8);
}
```

BASICS.PY

```
import glob
import random
import numpy as np

import wave
import librosa
import librosa.display


def get_and_shuffle_filenames(dir_name):
    filenames = glob.glob(str(dir_name) + "/*")
    random.shuffle(filenames)
    return filenames

def get_max_length(X_unfiltered):
    X_lengths = [audio.shape[0] for _, audio in X_unfiltered]

    max_length_1 = int(np.mean(X_lengths) + 2 * np.std(X_lengths))
    max_length = int(np.floor(max_length_1 / 256) * 256)
    return max_length

def get_max_length2(X_unfiltered, std):
    X_lengths = [audio.shape[0] for _, audio in X_unfiltered]

    max_length_1 = int(np.mean(X_lengths) + std * np.std(X_lengths))
    max_length = int(np.floor(max_length_1 / 256) * 256)
    return max_length

# https://www.tensorflow.org/tutorials/audio/simple_audio
def decode_audio(file_path):
    # read file to get buffer
    ifile = wave.open(file_path)
    samples = ifile.getnframes()
    audio = ifile.readframes(samples)

    # convert buffer to float32 using NumPy
    audio_as_np_int16 = np.frombuffer(audio, dtype=np.int16)
    audio_as_np_float32 = audio_as_np_int16.astype(np.float32)

    # get largest absolute value
    max_val = np.max(
        np.absolute(
            [np.max(audio_as_np_float32), np.min(audio_as_np_float32)]))
    audio_normalized = audio_as_np_float32 / max_val

    return audio_normalized

def get_label(file_path):
    # label is in the filename
    parts = file_path.split("/")
    label = int(parts[2].split("_")[0])

    return label

def spect(signal, max_length):
    range_val = int(max_length/256)

    spectogram = []
    for i in range(range_val):
        window_fft = np.fft.rfft(signal[i * 256: (i + 1) * 256])[:-1]
        window_fft = list(np.abs(window_fft))
        spectogram.append(window_fft)
    spectogram = np.array(spectogram)
    spectogram = librosa.amplitude_to_db(spectogram, ref=np.max)
    spectogram = spectogram / 20
    return spectogram

def normalize_arr(array):
    mean = np.mean(array)
    std = np.std(array)
    array = array - mean
    array = array / std
    return array
```

```
def split_full(X_full, y_full):
    num_samples = len(X_full)

    tenth = int(num_samples * 0.1)
    eightyth = tenth * 8

    X_train = X_full[:eightyth]
    y_train = y_full[:eightyth]

    X_val = X_full[eightyth: eightyth + tenth]
    y_val = y_full[eightyth: eightyth + tenth]

    X_test = X_full[eightyth + tenth:]
    y_test = y_full[eightyth + tenth:]

    return [(X_train, y_train), (X_val, y_val), (X_test, y_test)]
```

In [1]:
```
import random

# for data, model, training
import pandas as pd
import numpy as np
import tensorflow as tf
import librosa

import matplotlib.pyplot as plt
import seaborn as sns

import basics

# Set the seed value for experiment reproducibility.
seed = 42
random.seed(42)
tf.random.set_seed(seed)
np.random.seed(seed)
```

In [2]:
```
filenames = basics.get_and_shuffle_filenames("./recordings")

print(filenames[:5])
```

```
['./recordings/2_jackson_13.wav', './recordings/6_george_34.wav', './reco
rdings/7_george_5.wav', './recordings/1_yweweler_21.wav', './recordings/2
_george_42.wav']
```

In [3]:
```
X_unfiltered = [(file_path, basics.decode_audio(file_path)) for file_path

# to remove outliers
max_length = basics.get_max_length(X_unfiltered)
print(max_length)
```

```
5632
```

In [4]:
```
X_full = [] # padded X values 0-7
y_full = []

numbers = [0] * 8

for file_path, audio in X_unfiltered:
    x_val = audio
    y_val = basics.get_label(file_path)
    signal_length = audio.shape[0]

    if y_val > 7:
        continue
    if signal_length > max_length:
        numbers[y_val] += 1
        continue

    x_val = np.pad(
        x_val, (0, max_length - signal_length),
        'constant', constant_values=(0, 0))
        x_spect = basics.spect(x_val, max_length)
        x_spect = x_spect.flatten()

        X_full.append(x_spect)
        y_full.append(y_val)

X_full = np.array(X_full)
y_full = np.array(y_full)

print(X_full.shape)

num_samples, sample_w = X_full.shape
print(num_samples)
print(sample_w)

print(y_full[:10])
```

```
(2312, 2816)
2312
2816
[2 6 7 1 2 6 6 4 3 2]
```

In [5]:

```
# dropped outliers
df = pd.DataFrame.from_dict({"quantities": numbers})
print(df)
print(sum(numbers))
```

```
   quantities
0          12
1           9
2           7
3           9
4           3
5           8
6          29
7          11
88
```

In [6]:

```
# normalize data
X_full = basics.normalize_arr(X_full)

# partition into 80:10:10
partitions = basics.split_full(X_full, y_full)

X_train, y_train = partitions[0]
X_val, y_val = partitions[1]
X_test, y_test = partitions[2]

print('Training set size', len(X_train))
print('Validation set size', len(X_val))
print('Test set size', len(X_test))
```

```
Training set size 1848
Validation set size 231
Test set size 233
```

In [7]:

```
# INPUTS ARE NORMALIZED

model = tf.keras.models.Sequential()
```

24

```
model.add(tf.keras.layers.Reshape((128, 22, 1), input_shape=(sample_w,)))

model.add(tf.keras.layers.Conv2D(32, (4, 4), activation='relu', input_sha
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.AveragePooling2D(2,2))

model.add(tf.keras.layers.Conv2D(16, (4, 4), activation='relu'))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.AveragePooling2D(2,2))

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dropout(0.15))
model.add(tf.keras.layers.Dense(30, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='softmax'))

model.build()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 128, 22, 1)        0

 conv2d (Conv2D)             (None, 125, 19, 32)       544

 dropout (Dropout)           (None, 125, 19, 32)       0

 average_pooling2d (AverageP  (None, 62, 9, 32)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 59, 6, 16)         8208

 dropout_1 (Dropout)         (None, 59, 6, 16)         0

 average_pooling2d_1 (Averag  (None, 29, 3, 16)        0
 ePooling2D)

 flatten (Flatten)           (None, 1392)              0

 dropout_2 (Dropout)         (None, 1392)              0

 dense (Dense)               (None, 30)                41790

 dense_1 (Dense)             (None, 8)                 248

=================================================================
Total params: 50,790
Trainable params: 50,790
Non-trainable params: 0
_________________________________________________________________
2022-05-08 18:56:55.384594: I tensorflow/core/platform/cpu_feature_guard.
cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Netwo
rk Library (oneDNN) to use the following CPU instructions in performance-
critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropria
te compiler flags.
```

In [8]:

In [8]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

In [9]:
```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/30
58/58 [==============================] - 4s 56ms/step - loss: 1.9961 - ac
curacy: 0.2197 - val_loss: 1.8697 - val_accuracy: 0.3333
Epoch 2/30
58/58 [==============================] - 3s 54ms/step - loss: 1.7204 - ac
curacy: 0.3804 - val_loss: 1.5022 - val_accuracy: 0.4892
Epoch 3/30
58/58 [==============================] - 3s 53ms/step - loss: 1.2666 - ac
curacy: 0.5498 - val_loss: 1.1202 - val_accuracy: 0.6494
Epoch 4/30
58/58 [==============================] - 3s 53ms/step - loss: 1.0096 - ac
curacy: 0.6456 - val_loss: 0.9437 - val_accuracy: 0.6926
Epoch 5/30
58/58 [==============================] - 3s 53ms/step - loss: 0.8507 - ac
curacy: 0.6953 - val_loss: 0.7806 - val_accuracy: 0.7749
Epoch 6/30
58/58 [==============================] - 3s 54ms/step - loss: 0.7309 - ac
curacy: 0.7597 - val_loss: 0.7337 - val_accuracy: 0.7446
Epoch 7/30
58/58 [==============================] - 3s 52ms/step - loss: 0.6105 - ac
curacy: 0.7873 - val_loss: 0.5504 - val_accuracy: 0.8398
Epoch 8/30
58/58 [==============================] - 4s 61ms/step - loss: 0.5314 - ac
curacy: 0.8111 - val_loss: 0.4898 - val_accuracy: 0.8874
Epoch 9/30
58/58 [==============================] - 4s 62ms/step - loss: 0.4484 - ac
curacy: 0.8463 - val_loss: 0.4503 - val_accuracy: 0.8615
Epoch 10/30
58/58 [==============================] - 3s 47ms/step - loss: 0.3937 - ac
curacy: 0.8631 - val_loss: 0.4106 - val_accuracy: 0.8874
Epoch 11/30
58/58 [==============================] - 3s 53ms/step - loss: 0.3637 - ac
curacy: 0.8772 - val_loss: 0.3825 - val_accuracy: 0.8874
Epoch 12/30
58/58 [==============================] - 3s 54ms/step - loss: 0.3049 - ac
curacy: 0.8988 - val_loss: 0.3790 - val_accuracy: 0.8788
Epoch 13/30
58/58 [==============================] - 3s 49ms/step - loss: 0.2606 - ac
curacy: 0.9118 - val_loss: 0.3768 - val_accuracy: 0.8745
Epoch 14/30
58/58 [==============================] - 3s 49ms/step - loss: 0.2650 - ac
curacy: 0.9058 - val_loss: 0.3788 - val_accuracy: 0.8831
Epoch 15/30
58/58 [==============================] - 3s 48ms/step - loss: 0.2072 - ac
curacy: 0.9378 - val_loss: 0.3484 - val_accuracy: 0.8701
Epoch 16/30
58/58 [==============================] - 3s 47ms/step - loss: 0.2090 - ac
curacy: 0.9286 - val_loss: 0.3323 - val_accuracy: 0.9091
Epoch 17/30
58/58 [==============================] - 3s 51ms/step - loss: 0.1705 - ac
curacy: 0.9416 - val_loss: 0.3036 - val_accuracy: 0.9177
Epoch 18/30
```

```
58/58 [==============================] - 3s 47ms/step - loss: 0.1751 - ac
curacy: 0.9329 - val_loss: 0.3140 - val_accuracy: 0.9221
Epoch 19/30
58/58 [==============================] - 3s 48ms/step - loss: 0.1611 - ac
curacy: 0.9475 - val_loss: 0.2842 - val_accuracy: 0.9221
Epoch 20/30
58/58 [==============================] - 3s 49ms/step - loss: 0.1653 - ac
curacy: 0.9421 - val_loss: 0.2942 - val_accuracy: 0.9134
Epoch 21/30
58/58 [==============================] - 3s 47ms/step - loss: 0.1608 - ac
curacy: 0.9470 - val_loss: 0.2745 - val_accuracy: 0.9221
Epoch 22/30
58/58 [==============================] - 3s 47ms/step - loss: 0.1220 - ac
curacy: 0.9605 - val_loss: 0.2987 - val_accuracy: 0.9177
Epoch 23/30
58/58 [==============================] - 3s 48ms/step - loss: 0.1261 - ac
curacy: 0.9627 - val_loss: 0.3108 - val_accuracy: 0.9264
Epoch 24/30
58/58 [==============================] - 3s 49ms/step - loss: 0.1146 - ac
curacy: 0.9643 - val_loss: 0.2848 - val_accuracy: 0.9264
Epoch 25/30
58/58 [==============================] - 3s 48ms/step - loss: 0.1169 - ac
curacy: 0.9621 - val_loss: 0.2908 - val_accuracy: 0.9307
Epoch 26/30
58/58 [==============================] - 3s 49ms/step - loss: 0.1108 - ac
curacy: 0.9643 - val_loss: 0.2643 - val_accuracy: 0.9351
Epoch 27/30
58/58 [==============================] - 3s 49ms/step - loss: 0.1076 - ac
curacy: 0.9681 - val_loss: 0.2935 - val_accuracy: 0.9351
Epoch 28/30
58/58 [==============================] - 3s 47ms/step - loss: 0.1073 - ac
curacy: 0.9594 - val_loss: 0.2737 - val_accuracy: 0.9351
Epoch 29/30
58/58 [==============================] - 3s 49ms/step - loss: 0.0896 - ac
curacy: 0.9692 - val_loss: 0.2617 - val_accuracy: 0.9524
Epoch 30/30
58/58 [==============================] - 3s 47ms/step - loss: 0.1003 - ac
curacy: 0.9648 - val_loss: 0.2844 - val_accuracy: 0.9307
```

In [10]:

```
model.evaluate(X_test, y_test)
```

```
8/8 [==============================] - 0s 8ms/step - loss: 0.2627 - accur
acy: 0.9313
```

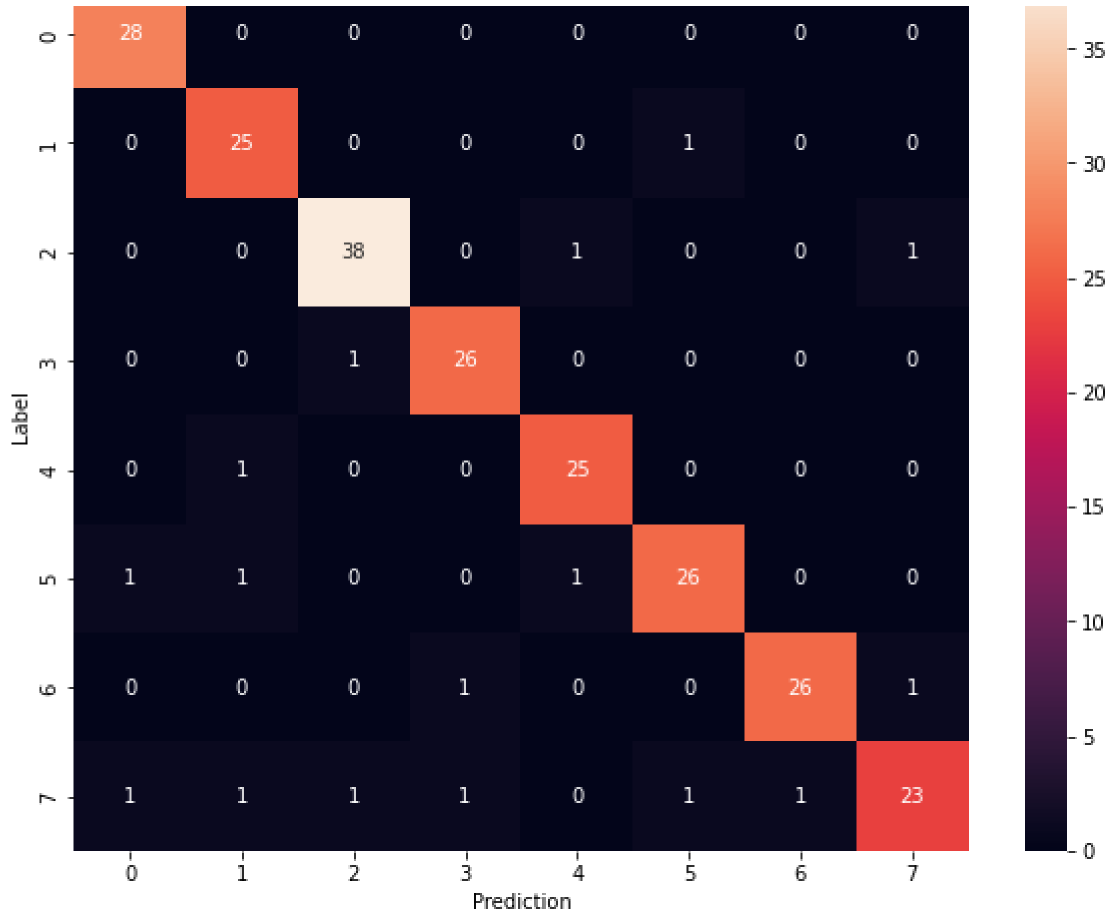
Out[10]:

```
[0.2627321183681488, 0.9313304424285889]
```

In [11]:

```
y_pred = np.argmax(model.predict(X_test), axis=1)

all_labels = list(range(8))

confusion_mtx = tf.math.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=all_labels, yticklabels=all_labels

plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

27



	0	1	2	3	4	5	6	7
0	28	0	0	0	0	0	0	0
1	0	25	0	0	0	1	0	0
2	0	0	38	0	1	0	0	1
3	0	0	1	26	0	0	0	0
4	0	1	0	0	25	0	0	0
5	1	1	0	0	1	26	0	0
6	0	0	0	1	0	0	26	1
7	1	1	1	1	0	1	1	23

In [12]:

```
model.save("v13_params2.h5")
```

In [13]:

```
import tensorflow_model_optimization as tfmot

prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

# Compute end step to finish pruning after 2 epochs.
batch_size = 128
epochs = 15
validation_split = 0.1 # 10% of training set will be used for validation

num_images = len(X_train)
end_step = np.ceil(num_images / batch_size).astype(np.int32) * epochs

# Define model for pruning.
pruning_params = {
      'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(initial_sp
                                                               final_spar
                                                               begin_step
                                                               end_step=e
}

model_for_pruning = prune_low_magnitude(model, **pruning_params)

# `prune_low_magnitude` requires a recompile.
model_for_pruning.compile(loss="sparse_categorical_crossentropy",
             optimizer="adam",
             metrics=["accuracy"])
```

```
model_for_pruning.summary()
```

```
/Users/jasminewu/Documents/6115/II/Spoken_Digits/scripts/venv/lib/python
3.7/site-packages/tensorflow_model_optimization/python/core/sparsity/kera
s/pruning_wrapper.py:238: UserWarning: `layer.add_variable` is deprecated
and will be removed in a future version. Please use `layer.add_weight` me
thod instead.
  trainable=False)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 prune_low_magnitude_reshape  (None, 128, 22, 1)       1
  (PruneLowMagnitude)

/Users/jasminewu/Documents/6115/II/Spoken_Digits/scripts/venv/lib/python
3.7/site-packages/tensorflow_model_optimization/python/core/sparsity/kera
s/pruning_wrapper.py:218: UserWarning: `layer.add_variable` is deprecated
and will be removed in a future version. Please use `layer.add_weight` me
thod instead.
  aggregation=tf.VariableAggregation.MEAN)
/Users/jasminewu/Documents/6115/II/Spoken_Digits/scripts/venv/lib/python
3.7/site-packages/tensorflow_model_optimization/python/core/sparsity/kera
s/pruning_wrapper.py:225: UserWarning: `layer.add_variable` is deprecated
and will be removed in a future version. Please use `layer.add_weight` me
thod instead.
  aggregation=tf.VariableAggregation.MEAN)
 prune_low_magnitude_conv2d   (None, 125, 19, 32)      1058
 (PruneLowMagnitude)

 prune_low_magnitude_dropout  (None, 125, 19, 32)      1
  (PruneLowMagnitude)

 prune_low_magnitude_average  (None, 62, 9, 32)        1
 _pooling2d (PruneLowMagnitu
 de)

 prune_low_magnitude_conv2d_  (None, 59, 6, 16)        16402
 1 (PruneLowMagnitude)

 prune_low_magnitude_dropout  (None, 59, 6, 16)        1
 _1 (PruneLowMagnitude)

 prune_low_magnitude_average  (None, 29, 3, 16)        1
 _pooling2d_1 (PruneLowMagni
 tude)

 prune_low_magnitude_flatten  (None, 1392)             1
  (PruneLowMagnitude)

 prune_low_magnitude_dropout  (None, 1392)             1
 _2 (PruneLowMagnitude)

 prune_low_magnitude_dense (  (None, 30)               83552
 PruneLowMagnitude)

 prune_low_magnitude_dense_1  (None, 8)                490
  (PruneLowMagnitude)

=================================================================
=================================================================
Total params: 101,509
Trainable params: 50,790
Non-trainable params: 50,719
_________________________________________________________________
```

In [14]:

```
import tempfile

logdir = tempfile.mkdtemp()

callbacks = [
  tfmot.sparsity.keras.UpdatePruningStep(),
  tfmot.sparsity.keras.PruningSummaries(log_dir=logdir),
]

model_for_pruning.fit(X_train, y_train,
                  batch_size=batch_size, epochs=epochs, validation_split=
                  callbacks=callbacks)
```

```
Epoch 1/15
13/13 [==============================] - 7s 207ms/step - loss: 0.0743 - a
ccuracy: 0.9778 - val_loss: 0.0410 - val_accuracy: 0.9892
Epoch 2/15
13/13 [==============================] - 3s 203ms/step - loss: 0.0717 - a
ccuracy: 0.9753 - val_loss: 0.0599 - val_accuracy: 0.9838
Epoch 3/15
13/13 [==============================] - 2s 181ms/step - loss: 0.0850 - a
ccuracy: 0.9747 - val_loss: 0.0341 - val_accuracy: 0.9892
Epoch 4/15
13/13 [==============================] - 2s 178ms/step - loss: 0.0574 - a
ccuracy: 0.9832 - val_loss: 0.0515 - val_accuracy: 0.9946
Epoch 5/15
13/13 [==============================] - 2s 174ms/step - loss: 0.0674 - a
ccuracy: 0.9820 - val_loss: 0.0393 - val_accuracy: 0.9838
Epoch 6/15
13/13 [==============================] - 2s 172ms/step - loss: 0.0572 - a
ccuracy: 0.9820 - val_loss: 0.0323 - val_accuracy: 0.9946
Epoch 7/15
13/13 [==============================] - 2s 178ms/step - loss: 0.0654 - a
ccuracy: 0.9765 - val_loss: 0.0426 - val_accuracy: 1.0000
Epoch 8/15
13/13 [==============================] - 2s 173ms/step - loss: 0.5938 - a
ccuracy: 0.7956 - val_loss: 1.1328 - val_accuracy: 0.5622
Epoch 9/15
13/13 [==============================] - 2s 174ms/step - loss: 0.8634 - a
ccuracy: 0.7132 - val_loss: 0.5942 - val_accuracy: 0.8270
Epoch 10/15
13/13 [==============================] - 2s 174ms/step - loss: 0.5410 - a
ccuracy: 0.8334 - val_loss: 0.4570 - val_accuracy: 0.8541
Epoch 11/15
13/13 [==============================] - 2s 177ms/step - loss: 0.3991 - a
ccuracy: 0.8900 - val_loss: 0.3284 - val_accuracy: 0.9297
Epoch 12/15
13/13 [==============================] - 2s 174ms/step - loss: 0.3212 - a
ccuracy: 0.9116 - val_loss: 0.2737 - val_accuracy: 0.9243
Epoch 13/15
13/13 [==============================] - 2s 182ms/step - loss: 0.2759 - a
ccuracy: 0.9218 - val_loss: 0.2475 - val_accuracy: 0.9351
Epoch 14/15
13/13 [==============================] - 2s 178ms/step - loss: 0.2421 - a
```

```
ccuracy: 0.9302 - val_loss: 0.2193 - val_accuracy: 0.9351
Epoch 15/15
13/13 [==============================] - 2s 175ms/step - loss: 0.2299 - a
ccuracy: 0.9314 - val_loss: 0.1974 - val_accuracy: 0.9405
```

Out[14]:

```
<keras.callbacks.History at 0x7fe7eb377bd0>
```

In [15]:

```
model_for_pruning.evaluate(X_test, y_test)
```

```
8/8 [==============================] - 0s 8ms/step - loss: 0.2653 - accur
acy: 0.9142
```

Out[15]:

```
[0.2652705907821655, 0.9141631126403809]
```

In [16]:

```
model_for_export = tfmot.sparsity.keras.strip_pruning(model_for_pruning)
_, pruned_keras_file = tempfile.mkstemp('.h5')
```

SPECTROGRAM.C

```
#include <math.h>
#include <stdio.h>
#include <fft4g_h.h>

#define MAX(x,y) ((x) > (y) ? (x) : (y))

/* random number generator, 0 <= RND < 1 */
#define RND(p) ((*(p) = (*(p) * 7141 + 54773) % 259200) * (1.0 / 259200.0))

#ifndef NMAX
#define NMAX 8192
#define NMAXSQRT 64
#endif

#define audio_length 5632
// #define audio_length 256
#define frame_size 256

const int spect_time = audio_length / frame_size; // 22
const int spect_y = frame_size / 2; // 128

double frame[256];

inline int16_t double_to_fixed(double input) {
        return (int16_t)(round(input * (1 << 13)));
}

void stft(double audio[audio_length], float spectogram[spect_time * spect_y]) {
    // does simplified stft
    // no overlap
    // no windowing function

    int total_length = spect_time * spect_y;

    // allocating space
    int ip[NMAXSQRT + 2];
    double w[NMAX * 5 / 4];

    // n_fft == hop_length for this stft (no overlap)
    int i, j, start_audio, start_spect;
    double a[frame_size];

    int thrown_out = 0;
    float amin = 1e-10;

    double val, max_val;
    max_val = 0;
    for (i = 0; i < spect_time; i++) {
        start_audio = (i - thrown_out) * 256;
        for (j = 0; j < frame_size; j++) { // copy section into array a
            a[j] = audio[start_audio + j];
        }

        rdft(frame_size, 1, a, ip, w);      // do fft

        start_spect = i * spect_y;

        for (j = 0; j < spect_y; j++) {
                if (j == 0) val = fabs(a[0]); // a[1] is some weird number, ignore
                else val = sqrt(pow(a[2 * j], 2) + pow(a[2 * j + 1], 2));    // magnitude of real and imaginary vector at
point

            if (val != val) {
                                thrown_out ++;
                                break;
                        }
            else if (val > 10000) {
                val = 20;
            }

            if (val > max_val) {
                max_val = val;
            }
```

```
                spectogram[start_spect + j] = (float) val;
            }
        }


        float log_max = log10(max_val);

        int i2 = (spect_time - thrown_out) * 256;
        for (i = i2; i < total_length; i++) {
            spectogram[i] = amin;
        }

        float new_val;
        float mean = 0;     // to calculate mean after

        int i_start;
        double temp_sum;
        double temp_mean;
        for (i = 0; i < spect_time; i++) {
            temp_sum = 0;
            i_start = i * spect_y;
                for (j = 1; j < spect_y; j++) {
                        val = spectogram[i_start + j];

                        if (val < amin || val != val) val = amin;                       // if zero or nan

                        new_val = log10(val) - log_max;

                        if (new_val < -80) new_val = -80;

                        spectogram[i_start + j] = new_val;
                        temp_sum = temp_sum + new_val;
                }
                temp_mean = (float) (temp_sum / spect_y);
                mean = mean + temp_mean;
            }
        mean = mean / spect_time;

        // calculate std of spectogram
        double sum = 0;
        for (i = 0; i < total_length; i++) {
            sum = sum + fabs(spectogram[i] - mean) / total_length;
        }
        float std = (float) sqrt(sum);

        for (i = 0; i < total_length; i++) {
            val = (spectogram[i] - mean) / std;
            spectogram[i] = val;
        }
    }
```