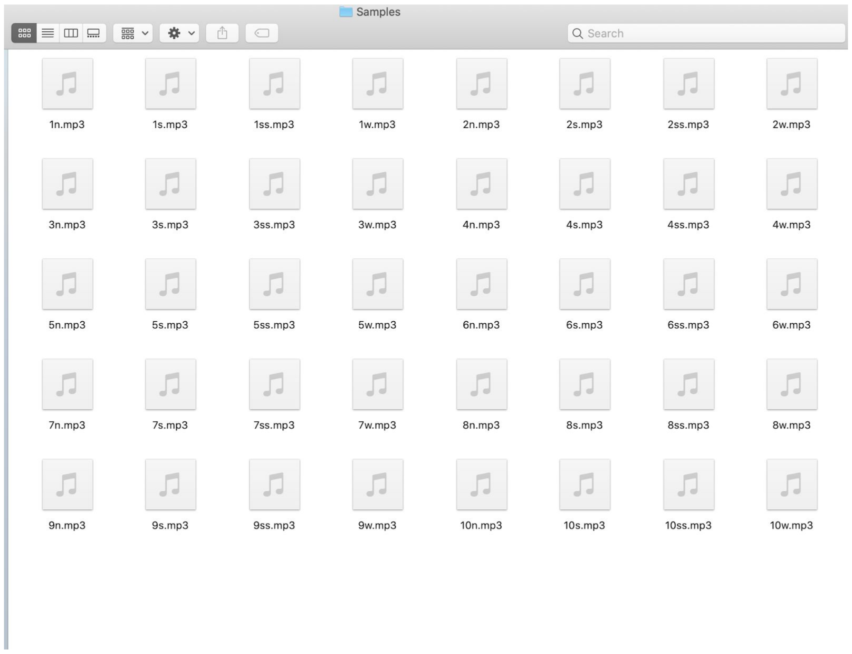
Week 12

Main tasks completed this week:

- Selected a paragraph from the Alice text as test sample
    - Sorted the text
    - Parsed audio file for each line of text
    - Generated audio files at 50%, 100%, 150% and 200% speed



  n: normal speed at 100% original
  w: slower speed at 50% original
  s: faster speed at 150% original
  ss: much faster speed at 200% original

- Meeting with prof.Boutin and discussed how to display text at various speed by inserting pulse between words
- Researched about "sleep" function for pausing between word
  https://www.pythoncentral.io/pythons-time-sleep-pause-wait-sleep-stop-your-code/

Plan for next week:

- Whitepaper review
- Pulse configuration at normal speed

Week 11 (Week 10-Spring Break)

Main tasks completed this week:

- DD3 doc: Updated measurement for specifications, citation, audio syncing portion
- Figured out the skill assessment replacement
- Integration of all buttons with Cody's subsystem

- Figured out how to generate speed-up and slow-down audio file:
    - First method: Matlab Phase Coder (https://www.ee.columbia.edu/~dpwe/resources/matlab/pvoc/)
        - Able to generate audio file in different speed but sound weird if not adjusting pitch and also only produced .wav file
    - Second method: Online Audio Speed Changer (https://www.audiospeedchanger.com/)
        - Able to generate audio file in different speed with more original sound and supportive with .mp3 file output

Plan for next week:

- Test parsing sample on book (paragraph)
- Generate more audio files with different speed

Week 9

Main tasks completed this week:

- Updated subsystem block diagram, flow diagrams, schematic, test result for most spec
- Completed urgent senior design plan

Plan for next week:

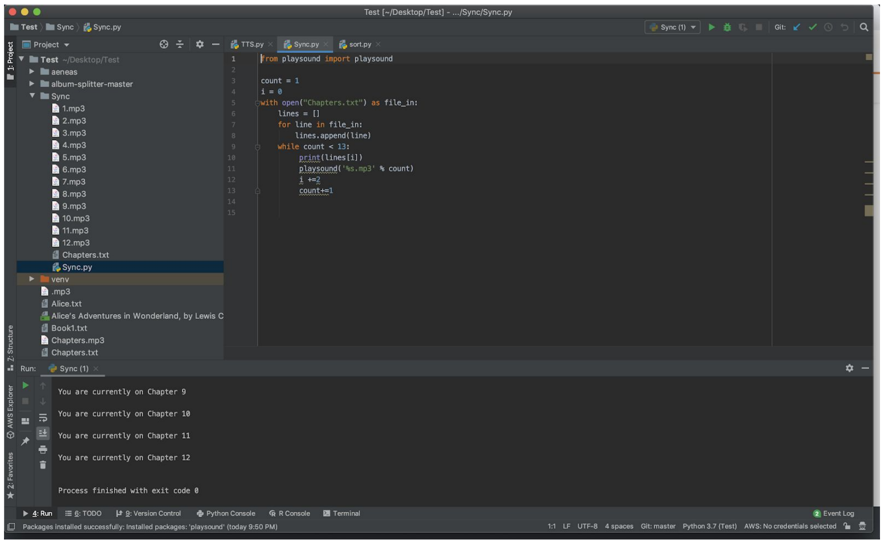
- Continue on integration with other subsystems
- PCB design for speed button

https://www.mathworks.com/help/audio/examples/measure-audio-latency.html

Week 8

Main tasks completed this week:

- Successfully wrote a algorithm to demo the way we are syncing the audio on micro-controller using Python



```python
from playsound import playsound

count = 1
i = 0
with open("Chapters.txt") as file_in:
    lines = []
    for line in file_in:
        lines.append(line)
    while count < 13:
        print(lines[i])
        playsound('%s.mp3' % count)
        i +=2
        count+=1
```

```
You are currently on Chapter 9

You are currently on Chapter 10

You are currently on Chapter 11

You are currently on Chapter 12


Process finished with exit code 0
```

Packages installed successfully: Installed packages: 'playsound' (today 9:50 PM)

    - How we plan for on board syncing to work: we will have separate folder called "Chapters" and "Page number" for parsed audio files and text, once the text display notifies the specific line of the text array for text-to-speech output, we would have the file-path for which audio file is going to output on the speaker/audio jack using DAC and then the file-path variables will be updated to match the position.
- Updated block diagram and demo plan for speed control portion
- Successfully updated prescaler for LED timer frequency in ADC interrupt, however ADC value fluctuate so much even without using potentiometer knob tp adjust the speed, so switched to using bounceless push button to set 4 different states for 4 prescaler values
    - able to update prescaler values (speed divider) everytime push button is pressed and after 4 values will cycle back to the first one
    - text-to-speech (audio output) will maintain constant speed as playing for now and has its own clocking frequency, but I will still try to make speeded up audio files just in case
    - speed divider will only be sent to text display at this point

Plan for next week:

- Revise design document, update subsystem details
- Start integrating speed control with other subsystems

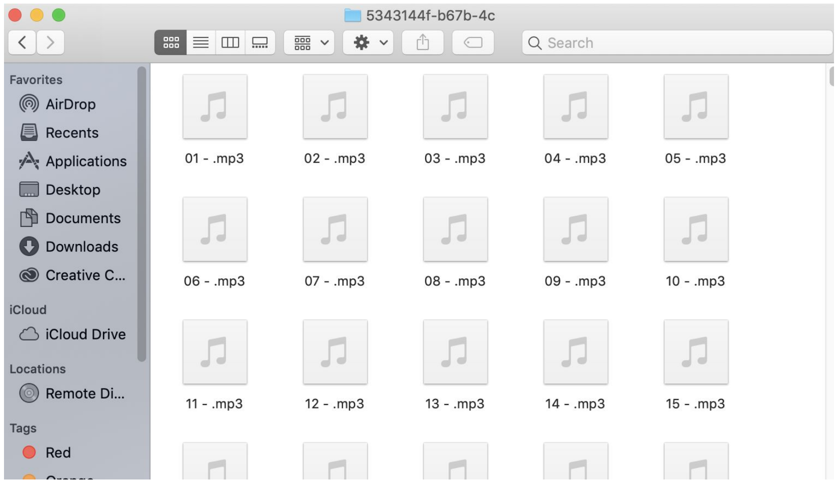
<u>Week 7</u>

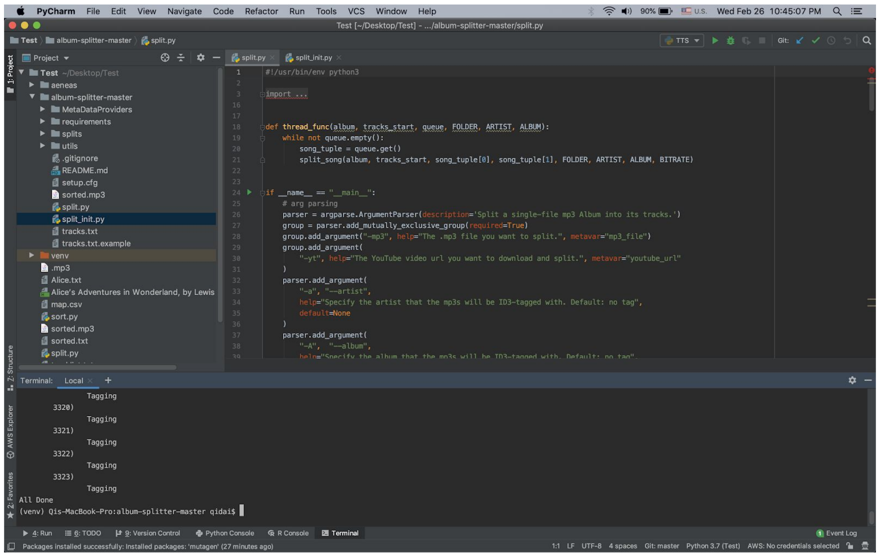
Main Tasks completed this week:

- Updated subsystem diagram, specs and flow diagram
- Reconfigured ADC
    - using single mode with 100ms delay, 8 bit resolution and 50ms conversion gap

        - using interrupt-based mode
            - ADC will work as potentiometer turns around, however, it kept updating its measurement value even without using knob??
            - LED configuration successfully, but could not update its pre-scaler value and had to do it manually, which means it would not reflect the update prescaler value for different ADC measurements
- Converted sorted text file to mp3 using gTTS, mapped the audio file with original text file, tested the accuracy of map file in srt format using Lectora Online
    - Mapping accuracy tested increased 95%, which meant the timestamps matched the audio and the text as expected
- Imported .csv map file into excel and converted timestamps in seconds into hh:mm:ss format and exported as .txt for audio parsing
    - successfully parsed audio file into fragments of audio files, each file corresponded to a line of the book with file name in order
      reference: https://github.com/crisbal/album-splitter

        - audio files are expected to save on the usb drive or sd card, line position would be tracked and then the position control would notify the text-to-speech output to play the corresponding audio file for that specific line
            - accuracy anticipated to be average 75% depending on the html file we are using since the parsed audio file matched the sorted .txt file of the ebook
            - parsing process time for 2h47 audio: approximately 15 mins for over 3000 audio clips (217.3 MB)





```python
#!/usr/bin/env python3

import ...


def thread_func(album, tracks_start, queue, FOLDER, ARTIST, ALBUM):
    while not queue.empty():
        song_tuple = queue.get()
        split_song(album, tracks_start, song_tuple[0], song_tuple[1], FOLDER, ARTIST, ALBUM, BITRATE)


if __name__ == "__main__":
    # arg parsing
    parser = argparse.ArgumentParser(description='Split a single-file mp3 Album into its tracks.')
    group = parser.add_mutually_exclusive_group(required=True)
    group.add_argument("-mp3", help="The .mp3 file you want to split.", metavar="mp3_file")
    group.add_argument(
        "-yt", help="The YouTube video url you want to download and split.", metavar="youtube_url"
    )
    parser.add_argument(
        "-a", "--artist",
        help="Specify the artist that the mp3s will be ID3-tagged with. Default: no tag",
        default=None
    )
    parser.add_argument(
        "-A",  "--album",
        help="Specify the album that the mp3s will be ID3-tagged with. Default: no tag",
```

```
                Tagging
        3320)
                Tagging
        3321)
                Tagging
        3322)
                Tagging
        3323)
                Tagging
All Done
(venv) Qis-MacBook-Pro:album-splitter-master qidai$
```

Packages installed successfully: Installed packages: 'mutagen' (27 minutes ago)

Plan for next week:

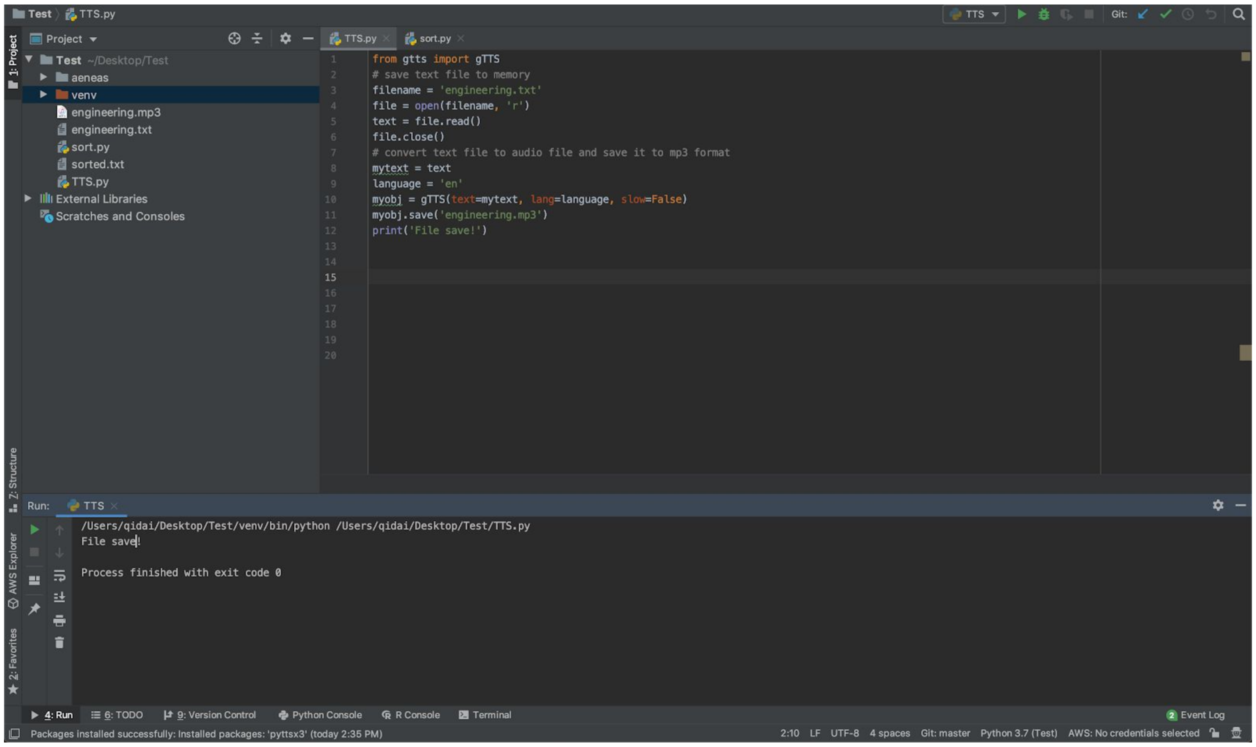
- Ask for help on ADC configuration with LED frequency (either update on pre-scaler or using PWM)
- Check on updates with position control on playing selected audio files

Week 6

Main Tasks completed this week:

- Tested ADC configuration using potentiometer under 3.3 V
    - Speed divider for the led blinking frequency was not reflected
        - found out that this was caused by the errors in ADC conversion(ADC data was jumping all over the place even without changing potentiometer) → need to reconfigure ADC and see what happened
- Completed sorting the file by comma, which will be better for mapping

- Generated audio for text file using gTTS(https://www.geeksforgeeks.org/convert-text-speech-python/), which would be useful if the user does not have a audio on file
    - The whole process took more time than I think, around 13mins to finish processing the whole textbook and produced a 2h30min audio file for the book, and it took around 1min to produce the .srt mapping file

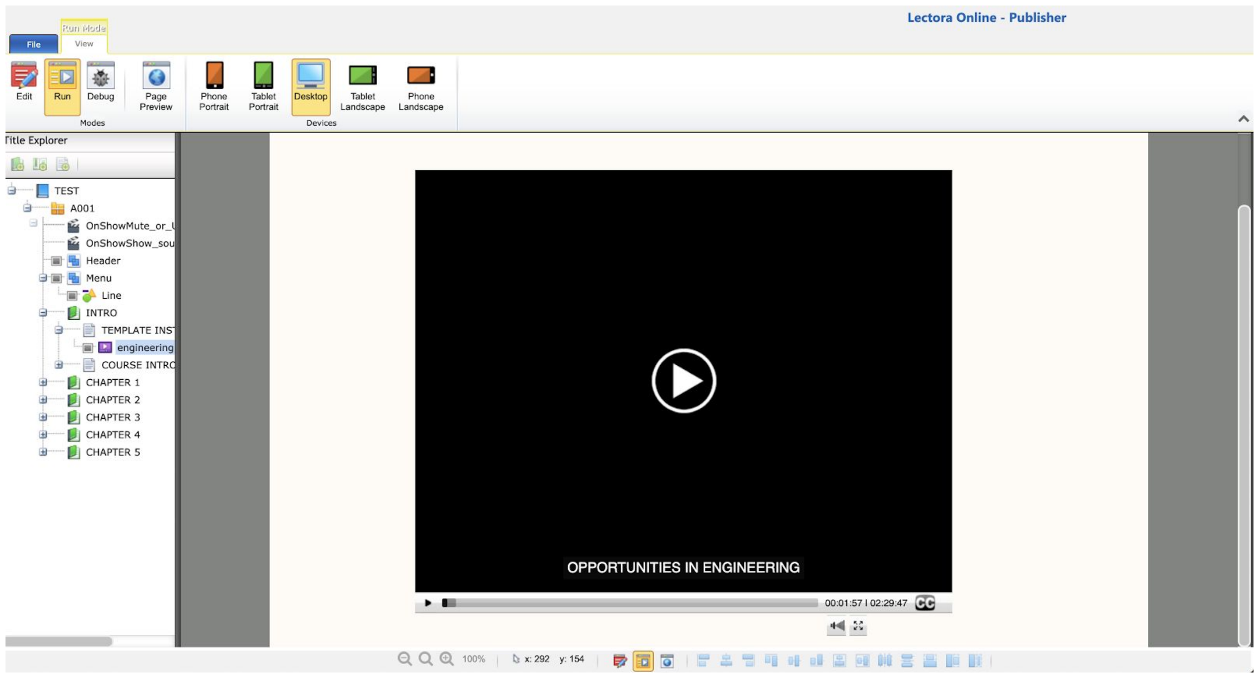


```python
from gtts import gTTS
# save text file to memory
filename = 'engineering.txt'
file = open(filename, 'r')
text = file.read()
file.close()
# convert text file to audio file and save it to mp3 format
mytext = text
language = 'en'
myobj = gTTS(text=mytext, lang=language, slow=False)
myobj.save('engineering.mp3')
print('File save!')
```

```
/Users/qidai/Desktop/Test/venv/bin/python /Users/qidai/Desktop/Test/TTS.py
File save!

Process finished with exit code 0
```

- In order to test the accuracy of the mapping file, I used this online audio captioning web app, which allowed me to input the video of the audio file and the generated mapping file to create the captioning (for this process I used .mp4 for audio and .srt for mapping instead of .mp3 and .xml) https://www.trivantis.com/blog/add-closed-captions-video-audio-lectora/



- I have played the closed captioning to my teammates and the text matched large portion of the audio but when it got to the later part, captioning started to fell behind the audio for 1-2 phrases (audio file is based on the original text file while the mapping file is based on the sorted file, which I removed some of the irrelevant punctuation signs like "*" and ",", so the audio contained reading of the signs while the mapping file did not)

Plan for next week:

- Reconfigure ADC using single mode instead
- Tested captioning for audio based on the sorted file and see if accuracy increased

Note for the week:

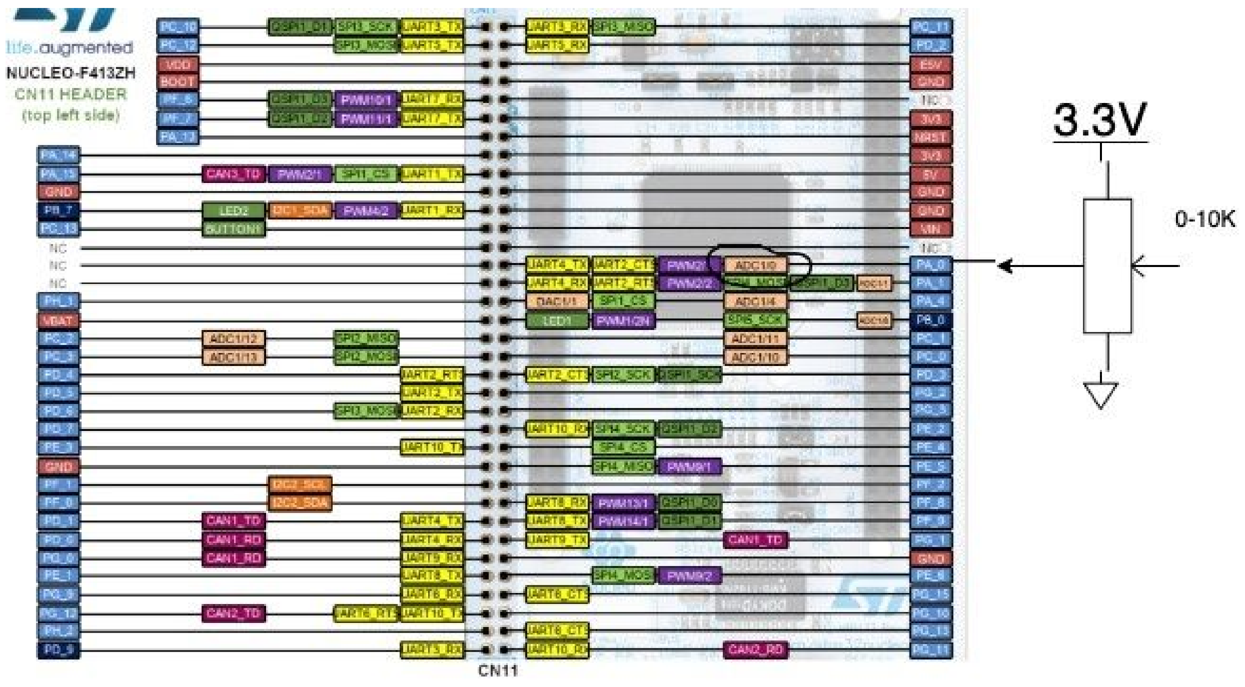
Thoughts: Plan to implement this method because i need to continuously record measurement data and send it to the DMA for future usage.

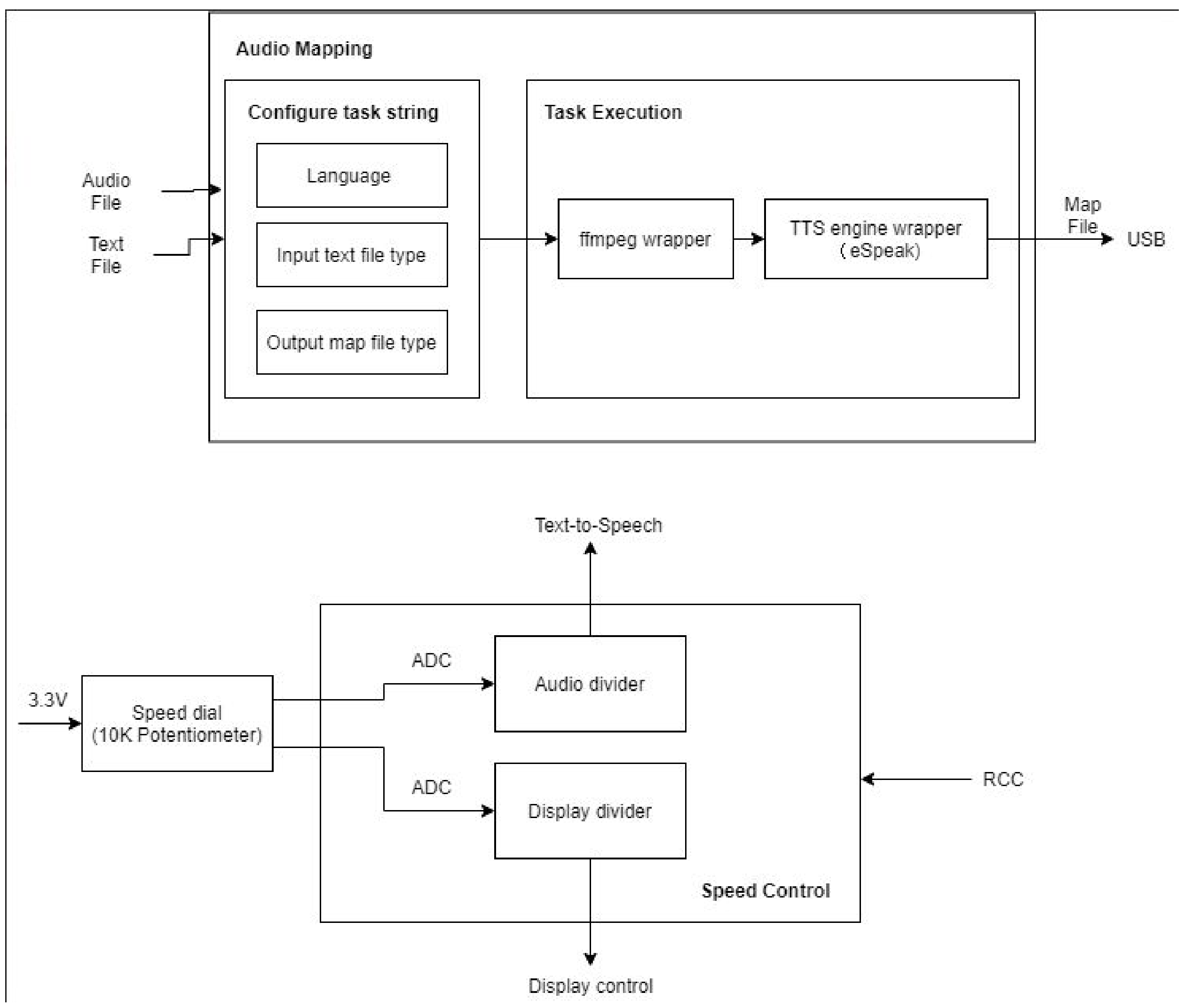
Week 5

Main Tasks completed this week:

- Completed flow diagram for subsystem
- Completed preparation for managers' meeting

- Updated schematic and team system diagram with updated ADC pin in





Plan for next week:

- Complete initialization for ADC in continuous mode

Weekly Research:

how does ADC work on stm32f4 in single mode for multiple channel input:

Thoughts:

This is really helpful especially when I am planning to use this program as one of my resource in figuring how how to code for ADC exactly from start to end and how I can continue to work on the assignment based on this conversion structure.

Intro to ffmpeg:

https://www.ffmpeg.org/about.html

Thoughts: basically this is a CLT convert multimedia file between format and able to decode, encode, transcode, mux, demux, stream, filter and play.

Intro to eSpeak: (TTS)

http://espeak.sourceforge.net/

Thoughts: eSpeak is a compact open source software speech synthesizer, which will be able to converts text to phonemes with pitch and length information and be adapted as a front end for another speech synthesis engine

More explanation on aeneas library:

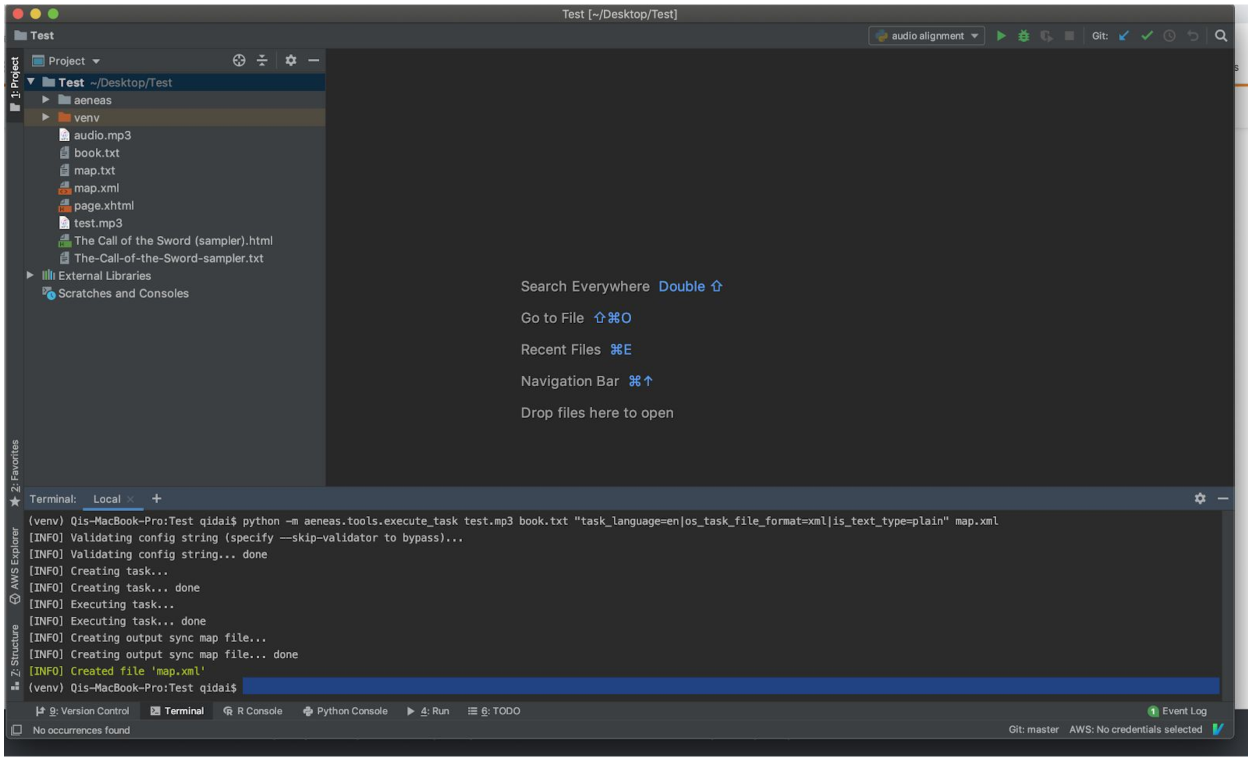
https://github.com/readbeyond/aeneas/blob/master/wiki/HOWITWORKS.md

Using the Sakoe-Chiba Band Dynamic Time Warping (DTW) algorithm to align the Mel-frequency cepstral coefficients (MFCCs) representation of the given (real) audio wave and the audio wave obtained by synthesizing the text fragments with a TTS engine, eventually mapping the computed alignment back onto the (real) time domain.

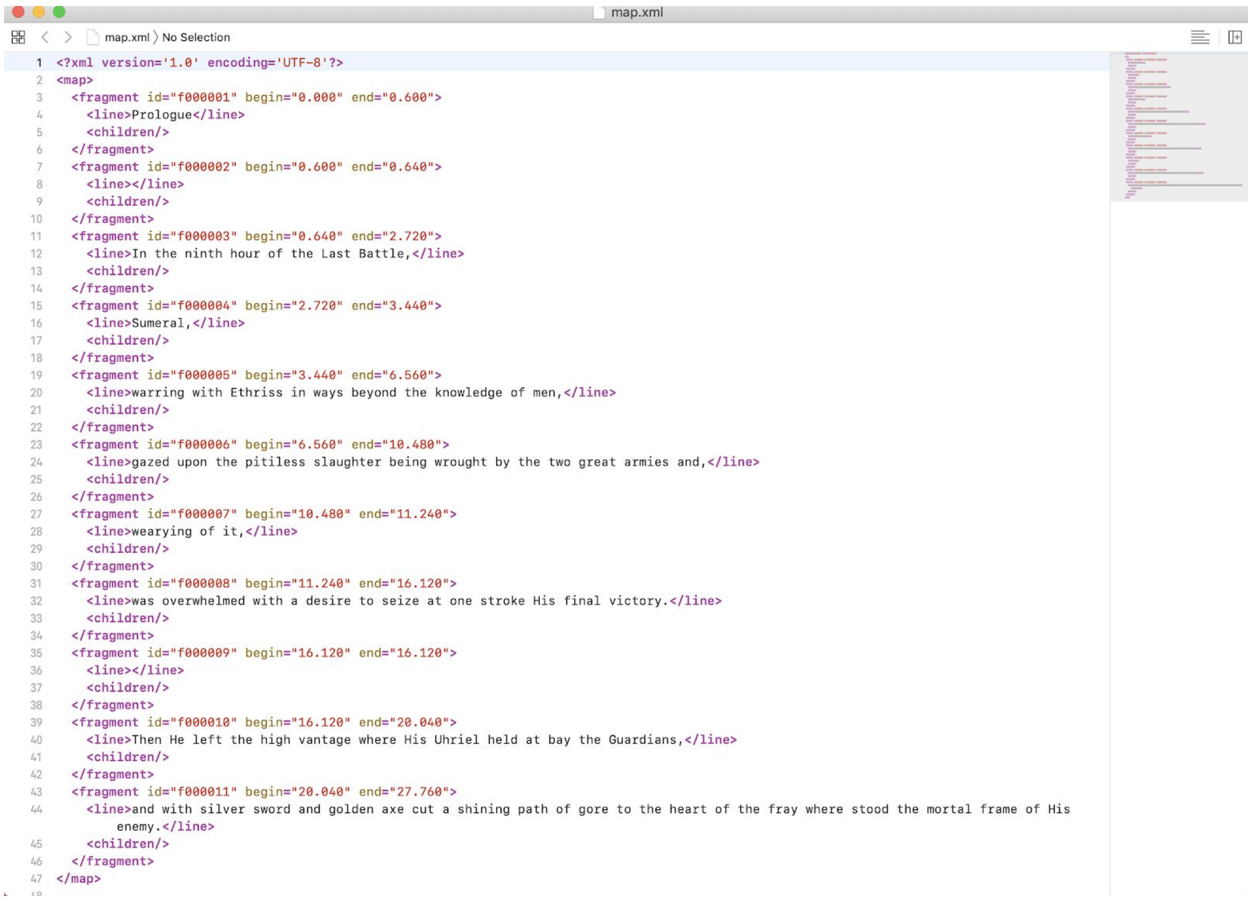
Week 4

Main tasks completed for the week:

- Updated block diagram and subsystem design document
- Researched more on the audio and ext synchronization method in the aeneas library in Python and downloaded on Pycharm
  - Successfully generated audio-text-alignment using this library and output a mapping file with all ids for each fragment in .xml format, the parsing is by line(phrases separated by comma), the input format is .txt and .mp3
  - The process time for audio parsing the whole ebook I selected is around 40 sec
  - according to my experiment, using a sorted plain .txt (with all lines separated by comma) or using .xhtml will work better in mapping it; while using unsorted plain .txt, the map file will randomly mapped it using blanks and paragraphs.

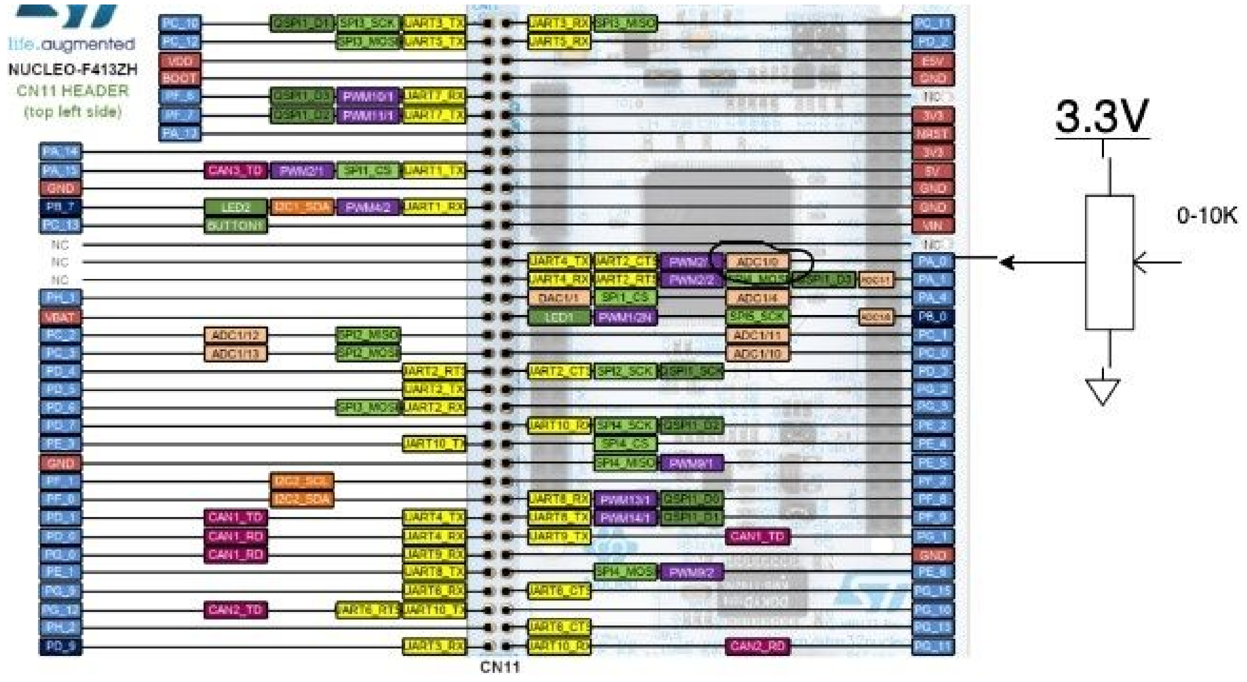


```
(venv) Qis-MacBook-Pro:Test qidai$ python -m aeneas.tools.execute_task test.mp3 book.txt "task_language=en|os_task_file_format=xml|is_text_type=plain" map.xml
[INFO] Validating config string (specify --skip-validator to bypass)...
[INFO] Validating config string... done
[INFO] Creating task...
[INFO] Creating task... done
[INFO] Executing task...
[INFO] Executing task... done
[INFO] Creating output sync map file...
[INFO] Creating output sync map file... done
[INFO] Created file 'map.xml'
(venv) Qis-MacBook-Pro:Test qidai$
```



```
<?xml version='1.0' encoding='UTF-8'?>
<map>
  <fragment id="f000001" begin="0.000" end="0.600">
    <line>Prologue</line>
    <children/>
  </fragment>
  <fragment id="f000002" begin="0.600" end="0.640">
    <line></line>
    <children/>
  </fragment>
  <fragment id="f000003" begin="0.640" end="2.720">
    <line>In the ninth hour of the Last Battle,</line>
    <children/>
  </fragment>
  <fragment id="f000004" begin="2.720" end="3.440">
    <line>Sumeral,</line>
    <children/>
  </fragment>
  <fragment id="f000005" begin="3.440" end="6.560">
    <line>warring with Ethriss in ways beyond the knowledge of men,</line>
    <children/>
  </fragment>
  <fragment id="f000006" begin="6.560" end="10.480">
    <line>gazed upon the pitiless slaughter being wrought by the two great armies and,</line>
    <children/>
  </fragment>
  <fragment id="f000007" begin="10.480" end="11.240">
    <line>wearying of it,</line>
    <children/>
  </fragment>
  <fragment id="f000008" begin="11.240" end="16.120">
    <line>was overwhelmed with a desire to seize at one stroke His final victory.</line>
    <children/>
  </fragment>
  <fragment id="f000009" begin="16.120" end="16.120">
    <line></line>
    <children/>
  </fragment>
  <fragment id="f000010" begin="16.120" end="20.040">
    <line>Then He left the high vantage where His Uhriel held at bay the Guardians,</line>
    <children/>
  </fragment>
  <fragment id="f000011" begin="20.040" end="27.760">
    <line>and with silver sword and golden axe cut a shining path of gore to the heart of the fray where stood the mortal frame of His
        enemy.</line>
    <children/>
  </fragment>
</map>
```

- Learned how to find the correct registers and pins for my subsystem (ex: find RCC→ RCC AHB1 peripheral clock enable→look for ADC) on https://www.st.com/content/ccc/resource/technical/document/reference_manual/group0/81/ea/88/1f/97/9e/4a/d0/DM00305666/files/DM00305666.pdf/jcr:content/translations/en.DM00305666.pdf
- Updated schematic



Goal for next week:

- Prepare for manager meeting
- Research a way to connect divider in c with audio parsing in python

Research note:

Analog to Digital Conversion

https://www.mccdaq.com/PDFs/specs/Analog-to-Digital.pdf

Thoughts:

Some definitions:

ADC: converts a analog voltage to binary number(1s/0s) and then to a base 10 digital number

MSB: the farthest bit to the left

A helpful diagram to understand how ADC works:

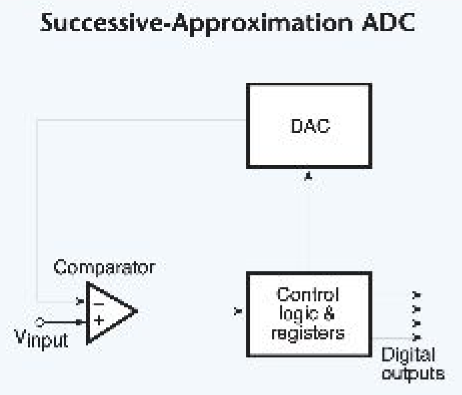


Fig. 2.01. *Interestingly, this ADC uses a digital-to-analog converter and a comparator. The logic sets the DAC to zero and starts counting up, setting each following bit until it reaches the value of the measured input voltage. The conversion is then finished and the final number is stored in the register.*

The ADC on our controller is 12 bit (2^12 = 4096 resolution), which means that the voltage will be converted to a 12 bit digital number by successively comparing each digit until getting the closest voltage and then it would be assigned to certain speed for both the display divider and the audio divider.

The RCC basically set up the cycle of conversion time (sampling time), like how frequently the machine sample the voltage and convert it digitally.

Vsys = 3.3V

Resolution/System Voltage = ADC reading/Analog Voltage Measured

Conversion time (Tc) = N*Tclk(depending on RCC?)

aeneas python library:

https://pypi.org/project/aeneas/1.4.0.0/

Useful info:

-the input format can support .txt (with plain/parsed/subtitles/unparsed format) or .xhtml(with all ids and class attributed)

-the input audio file can be in all ffmpeg supported format including both mp3. and .wav

-the output sync map can be CSV, JSON, RBSE, SMIL, SSV, TSV, TTML, TXT, VTT, XML

Thoughts: this library can support a variety of input and output format, which includes the ones we are using for our ebook; however, it is a challenge to import a python library on our microcontroller without using like micropython to support the language. So i think it is more effective to pre-processed all the books and save them on the usb and then the text-to-speech system can simply call the audio and xml file and play them in the required speed, which can reduce the work load of microcontroller as well.

how to use XML to data drive the application to play sounds based on the content of the xml file?

https://stackoverflow.com/questions/21584503/playing-sound-files-based-on-xml-in-c-sharp/21585656

Thoughts:

I think the xml file contains all the ids to identify the audio intervals for the correct line of text, which will be accessible for text-to-speech system to play the sound based on these ids.

Text cleaning method:

https://machinelearningmastery.com/clean-text-machine-learning-python/

Thoughts: When we try to map ebook, I found out that when mapping the whole book it would be mapped based on blanks and paragraphs. So it is better to pre-clean the text document into parsed lines or removed punctuations so that it would work better.

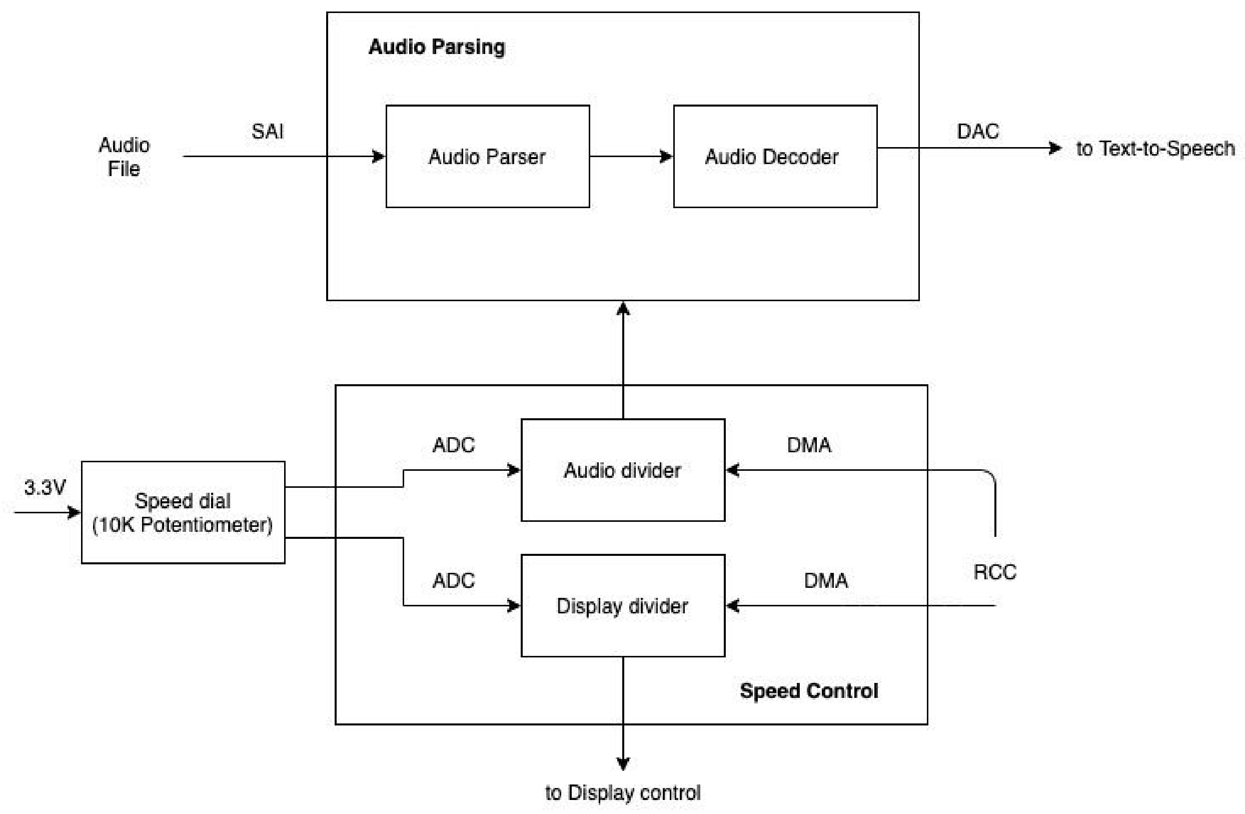
==

Week 3

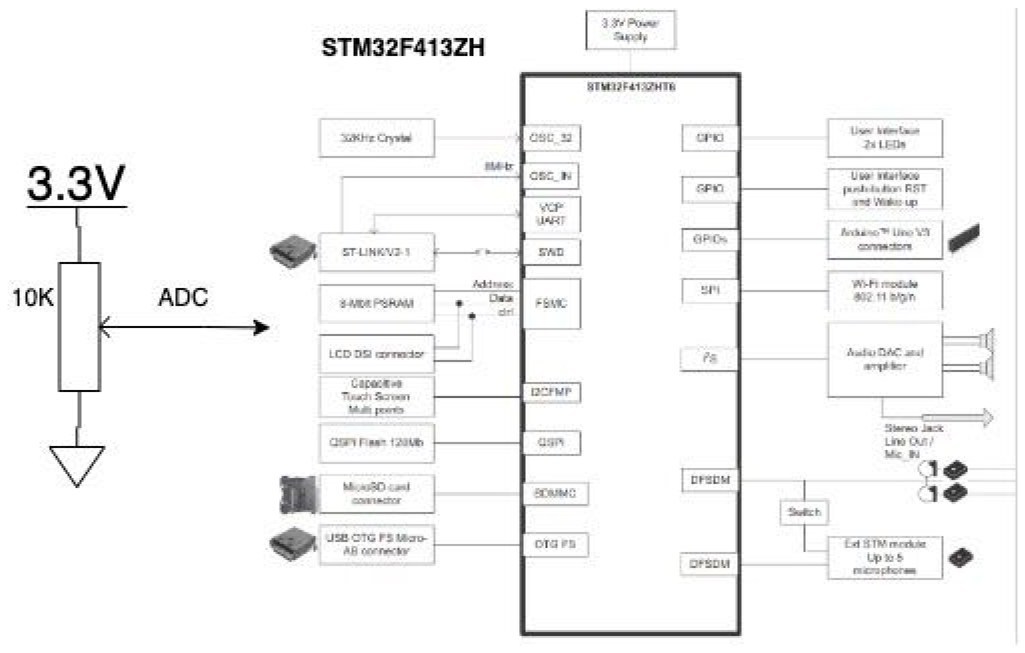
Main tasks completed for the week:

- Surveyed braille users and got feedback on how they expect the design to be in general
- Design document formation
    - Discussed individual subsystem division including block diagram details
    - Wrote revision log and subsystem description
    - Updated requirement

Block diagram:



Schematic:



- Researched audio parsing (definition, algorithm, articles)
    - .wav file input format and .xml output format
    - How to parse .wav file
    - How to tag and sync audio with text

        - Researched closed-caption standards
- Researched speed control design (schematic, potentiometer design, algorithm)
- Researched existent braille reader design
- Researched common reading speed for braille users and speaking speed for audiobook
- Researched STM32F413ZH specs document
- Updated project timeline Gantt chart

Goal for next week:

- Start looking into the detailed specs of microcontroller and figure out the correct pin to use
- Understand how a speed dial work and how to pass speed information
- Figure out how RCC plays in the speed control
- Start looking into the algorithm and the overall logical structure

Research note:
A library for syncing text with audio fragment:
https://www.readbeyond.it/aeneas/
https://github.com/readbeyond/aeneas/blob/master/aeneas/cdtw/cdtw_driver.c
http://software.sil.org/downloads/r/readingappbuilder/Reading-App-Builder-07-Using-aeneas-for-Audio-Text-Synchronization.pdf
Thoughts:
Once we have parsed audio we can load it into this synchronizer and make the text sync with the audio and output .xml file, however it seemed like it mostly work in python

Specs for microcontroller STM32F413ZH:
https://www.st.com/content/ccc/resource/technical/document/user_manual/group0/ad/82/4d/ae/bb/7a/42/9f/DM00340446/files/DM00340446.pdf/jcr:content/translations/en.DM00340446.pdf
Thoughts:
This source will help me identify the specs of the microcontroller we are using and make sure that I mark up the correct pins and signal inputs/outputs.

Braille reading speed:
https://www.researchgate.net/publication/228463176_Measuring_Braille_reading_speed_with_the_MNREAD_test
Thoughts:
The maximum reading speed is 185 words/min and minimum speed is 65 words/min for braille readers, so the speed should be range from 0.25 to 6 letters/sec for a reasonable slow to fast range for readers.

Common speaking speed:
https://virtualspeech.com/blog/average-speaking-rate-words-per-minute
Thoughts:
Common Audiobook speaking speed between 150 - 160 wpm, which is the upper range that people comfortably hear and vocalise words, which means the audio speed should range from 2 letters/sec to 11 letters/sec.

Audio parsing in C for WAV file:
http://truelogic.org/wordpress/2015/09/04/parsing-a-wav-file-in-c/
Libsndfile is a C library for reading and writing files containing sampled sound:
http://www.mega-nerd.com/libsndfile/
Audio parsing code reference:
https://github.com/rakyll/audio/blob/master/riff/parser.go
Thoughts:
In order to parse .wav file, we will have to store all the header info as well as the audio data and then parse them based on each line of the text file using markup language. However, I have to make sure what are the elements needed to sort the audio signal and output the write format for decoder to mark up the audio-text file.

Design and Implementation of an Audio Parser and Player:
http://docsdrive.com/pdfs/medwelljournals/jeasci/2017/5301-5306.pdf
Thoughts:
This research article gave ideas on how to input and parse an audio and actually use ID3 tag to specify each audio fragment and play it.

Cornell portable e-book for the blind design:
https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2017/mmm389_ahs278/mmm389_ahs278_final_report_2/mmm389_ahs278_final_report_2.html
Thoughts:
A 10K potentiometer for controlling speed and convert the resistance to reading speed (ADC), and the displaying speed will mainly depend on the common reading speed for braille reader while audio will share a different speed range so that it will not be messed up by the displaying speed. The specific details to convert the resistance to speed also requires clock and other parameters that I have to consider very carefully in writing my own algorithm.

How does potentiometer work?
https://randomnerdtutorials.com/electronics-basics-how-a-potentiometer-works/
Thoughts: this website provides a basic schematic diagram for the potentiometer that I am going to use to connect my circuit.

What is a .XML file format?
https://www.howtogeek.com/357092/what-is-an-xml-file-and-how-do-i-open-one/
https://www.rev.com/blog/close-caption-file-format-guide-for-youtube-vimeo-netflix-and-more
Thoughts: The .XML file XML is a markup language created by the World Wide Web Consortium (W3C) to define a syntax for encoding documents that both humans and machines could read. It does this through the use of tags that define the structure of the document, as well as how the document should be stored and transported.

Week 1-2
Main tasks completed for the week:

- Learned the difference between contracted and uncontracted braille language
- Detailed the design of braille reader
    - Design document write-up
        - Executive description
        - User story
        - Requirement
        - Influential factors
    - Discussed functionalities of the device
- Searched for potential mini speaker for audio output of the device
- Searched for potential mini push-pull solenoid products on the market
    - Compare prices between buying and building solenoid
    - Estimated specs for solenoid actuators

Goal for next week:

- Deciding which subsystem to work on
- Complete the design document revision
- Brainstorm questions to ask the DRC person
- More background research
- Create a project timeline

Research notes:
Portable Braille reader inspiration:
https://www.youtube.com/watch?v=N5iTWgvzhU8
Thoughts:
As shown in the video, the solenoids were all attached to the pin key pads directly, which means that when the actuator received the instruction from the microcontroller, the actuator pushed the pins up and down.

How to make a mini size solenoid:
https://www.youtube.com/watch?v=DvHiPvuWDPg
Thoughts:
Considering our budget is tight and the cheapest mini push-pull solenoid on the market would be around $5 each and we need around 48 solenoids, therefore, we may have to build our own solenoids if necessary.

Possible speaker choice for our audio system:
https://www.radioshack.com/products/radioshack-8-ohm-mini-speaker?variant=20332224901&utm_medium=cpc&utm_source=google&utm_campaign=Google%20Shopping&gclid=CjwKCAiAgqDxBRBTEiwA59eEN0Wc5nuAtmsIiGCme7ke9AVYahdqlP_jP2SLQW7PASx0L5ksvPXzRhoC2HMQAvD_BwE
Thoughts:
It is an 8-ohm speaker with reasonable size and price, however, not sure if this will supply enough volume needed for our device.

How to translate text to Braille using C:
http://liblouis.org/documentation/liblouis.html#lou_005fhyphenate
How to translate Grade 2 Braille from text using Python:
https://github.com/LazoCoder/Braille-Translator
Thoughts:
First of all, we have to make sure that the translator recognizes the “break point” of each sentence or paragraph; second, we have to make sure that representation of numbers and letters are differentiated because they usually use the same representation in Braille; third, there should be an escape code”.” for the capital letters.
The module code size is 11.74kB (2.93 for printing Braille→ maybe switch to connecting to the actuator end)

Grade 1 vs Grade 2 Braille:
http://www.acb.org/tennessee/braille.html
Thoughts:
In grade 2 Braille, a cell can represent a shortened form of the word, so if we want to use contracted Braille then we can consider using this translation above.

How to extract plain text from html file using C:
https://stackoverflow.com/questions/15319329/extract-plain-text-from-an-html-file-in-c
How to extract plain text from html file using Python:
https://github.com/Alir3z4/html2text
Thoughts:

Since we are using html file for our input text format, the converter control will have to first extract text from html file then they can store the converted Braille for the use of further text-to-speech processing.