EPR402 Lab Book

Wednesday, 06 July 2022 01:42

Ishaque Ramedies

U16023405

Project no: AO5

Project Title:
A school intercom system that filters inappropriate phrases and plays emails as announcements in the voice of the sender.

Description:

With the changing nature of schooling and the rotational systems in place can no longer be taken for granted that teachers making announcements and the students hearing them are in the same physical location. Furthermore, it no longer makes sense to employ someone whose sole job is collating announcements and making them at the specified time.

Good engineering design can eliminate the need for announcers by allowing teachers to submit announcements via email and then broadcasting the announcement to the students in a voice that mimics that of the sender. Natural language processing, speech synthesis and digital signal processing can be combined to achieve this outcome in real time with no human intervention, thus solving a very real problem in an elegant and efficacious way.

The challenge in this particular project will be to design and build a system that can remove inappropriate phrases from text, convert it to human sounding speech, and play it back without any human intervention. This project involves the design and construction of an integrated system, linguistic analysis of text, the use of signal processing algorithms, and the implementation of these algorithms on an embedded platform. The project will require the student to master a significant amount of theoretical background. Implementation should be from first principles. This means that while algorithms will be based on literature, the implementation will be developed from scratch by the student and no libraries may be used for any core processing functionality.

The student will be expected to deliver a system that can linguistically analyse text, remove inappropriate phrases, convert the text to speech signal that is designed to sound like the sender, and play it back over a speaker system. The system performance must be compared to the relevant literature.

Index

Wednesday, 06 July 2022 01:46

Design criteria

Wednesday, 06 July 2022 01:49

Ishaque Ramedies at 2022/07/06 02:09
The project will involve a couple of subsystems to make the thing work as intended by the project title and description.
I'm listing the specifications here so that I can focus on these while doing the project.

- The system should be able to receive text emails and display the messages it receives.
- The system should linguistically analyse the text and generate information for the text to speech synthesizer to produce.
- Relating to the point above it should have a profanity filter and spell checker.
- I need to create a text-to-speech synthesizer using formant analysis, which involves bandpass filters and DSP processing.
- I need to output the information on a speaker and it has to output about 60dB.
- An intercom casing should be designed and nitty gritty things like power supply and operation buttons can be designed for afterwards.

In order to accomplish most of this work, I need to have a lot of research papers that handle formant synthesis, text to speech synthesis, and natural language processing techniques. I also need to do a bit of research on the board that I want to use to implement my formant synthesis on, and work on the budget for the project as soon as possible.

The extent of the research I have done so far that I want to add to the lab book is that of **formant synthesis**. I briefly want to mention what it is. It involves selecting 5 frequencies of the persons voice when a sound is made and combining these frequencies in a train to form a synthesised sound. I think that digital filter libraries may be used for this but I just have to make sure that this is the case. I don’t think it is core processing, it is essentially just filtering, and I don't want to have to do all the coefficient calculation by hand. However I might have to add some FIR filter calculations just to back up my work.

Main criteria

Wednesday, 06 July 2022 01:27

Ishaque Ramedies at 2022/07/06 02:23

- So the main thing I need to include for the board specifications is the fact that I require DSP capabilities. I researched a bit and found out that I need an ARM cortex M4 processor or anything later than M4, like M5 for instance. The only reason why I need this type of processor is because the instructions are optimized for DSP algorithms when compared to the M3. However, if it turns out that I don't use any sort of DSP processing then the performance will be the same as the M3.
- Another thing that I might have to consider is the processor speed. For this I need to learn about the clock cycles that it will perform in a second. From the time that my board receives an email it will perform the natural language processing, which shouldn’t take too long to operate but I should still calculate an estimate of the instructions that will be used. I'll discuss the calculations in another section.
- After figuring out the clock speed of the processor, another good criteria to look at would be the support that the board offers. I want to select a board that has a decent software that I might be familiar with such as STM32 Cube IDE for instance. It has to be reliable and easily reproducible, so the software that I can program it on has to be established. I'm ok with using any other software that supports microcontroller development, but if it doesn't for instance have digital filter libraries then I don't want to use it.
- Then I have to consider my budget, my Project leader indicated to me that I shouldn't be too frugal when it comes to the board. I have to make sure that I get something good. So I want to spend my efforts on researching it properly before making a decision. I should order a board by the end of this week, and if I order it locally, then the board should arrive next week. I'll make an exhaustive list of the boards I consider and I'll tabulate my results to come.

One thing that I am uncertain about when it comes to the boards, is that if I program a chip on an embedded platform, how can I transfer it over to circuitry (PCB). I need to ask my study leader what he thinks about this.

Clock speed requirement

Wednesday, 06 July 2022 02:45

Ishaque Ramedies at 2022/07/06 02:47
These calculations are based on what I think I will be doing for my project:

Firstly the email system. This will not be considered as part of the processing so it shouldn't contribute much to the clock speed.

Next is the natural language processing and the linguistics analysis technique that I select. ~~The synthesizer will be rule based, so most likely I will follow a rule based structure for the technique~~. The linguistics analysis technique does not have to directly align with the synthesizer. The approaches for a technique is doing sentence parsing, doing a rules-based analysis, or a neural network approach (but for this I require a lot of samples, and I can't factor in time or resources into this kind of approach). I have to research how much time sentence parsing and rules based analysis would take. For rules based I would require a large set of rules, since the English language is very large and ever changing. For either one I would require a database of text words. I either have to create this myself or find one online.

Calculations, IC

Friday, 08 July 2022 09:40

Ishaque Ramedies at 2022/07/08 10:28
Research on linguistics analysis: According to determining the instruction cycle count to determine what clock speed you should be running for a certain application, it is **not possible to determine the exact amount** at most times (even with code available). There are often lots of variables to take into consideration. However, one can make a rough estimation and then account for any overhead instructions. So here are just **rough estimations** for the instruction count that will be required and how to calculate it.

The basic performance equation: https://www.d.umn.edu/~gshute/arch/performance-equation.xhtml

According to the equations, I need to determine the **instruction count (IC)**, then the **clocks per instruction (CPI)** and then the **clock time (CT).** The clock time is the execution time. So for my application **I want an execution time of 10 seconds from the moment that the email is received.** This is a decent amount of time to wait for an announcement.

Instruction count: **A CISC processor will be better than a RISC processor** since they have built in looping so that they can accomplish as much as several hundred RISC instruction executions. Also instruction count is loop dominated.

So the instruction count is based on the **length of the announcement.** For this implementation let's assume that it will be no longer than **130 words.** This is the average amount of words that can be read at average speed. If we assume the worst case then assume that for every word that it has to go over the underlying process, i.e. the entire **NLP process. NLP consists of Sentence detection, tokenization, lemmatization and cleaning, part of speech tagging.** So let's say that each sentence is approximately **10 words long,** meaning that there are **13 sentences.** So that will be **13 instructions for sentence detection. Tokenization will be 130 instructions for every word. Lemmatization and cleaning** is more complex so let's just assume for now that it will use a look up table to perform it **(260 instructions)**. Then **part of speech tagging will be another 130 instructions.** In total for the NLP:

IC_nlp = 13+130+260+130 = 533

Then I have to factor in the **linguistics analysis technique.** There's **sentence parsing, rules-based and neural network based**. If its neural network based, then it simply has to be trained on an external system and then the analysis will be a simple one-to-one instruction process. However it will require a lot of **samples** and most likely a **very long training time and inaccuracies due to the lack of time and resources.** So I should rather **not consider a neural network.**

Sentence parsing: Sentence parsing is a good example of looping. It basically uses a tree like structure to break up the sentence into words with identifiers. The disadvantages is that it loses out on **sentiment**. I'm not sure if sentiment is a core requirement, however it does seem **simple** enough to implement. The overall algorithm would parse each sentence (13 of them) with a tree like structure. The overall algorithm shouldn't take long to implement however it does need a lot of rules in place. Therefore it should take about 130 instruction cycles of if else rules. (ideal case).

Rules based: Rules based also requires rules in place for the identification to be accurate. I think assuming 130 instructions ideally is correct.

Finally we consider the text-to-speech synthesizer. The formant analysis will work with 5 bandpass filters, for real time filtering. If each band will have a narrow frequency pass band, (maybe 5 Hz) then

the amount of coefficients for a digital filter will not be many. Let's assume it will be around 16 coefficients (2^4). If each coefficient will be executed in a MAC manner, i.e. multiply and accumulate, according to the structure of digital filters, and each MAC takes a single instruction cycle, then that would be 16 coefficients times 5 filters times 130 words times 10 diphones for every word:

Ictts = 16*5*130*10=104 000

So total instruction count should be close to 105 000.

CPI and clock time

Friday, 08 July 2022 11:05

Ishaque Ramedies at 2022/07/08 11:18
According to the basic performance equation, CPI is affected by instruction-level parallelism and by instruction complexity. Without it simple instructions would take 4 or more cycles to execute. To determine this parameter I would have to look at boards that I want to consider (the instruction timing diagrams).

Clock time is just the inverse of the clock frequency. So if the clock frequency is 1 GHz then it would take 1ns for an instruction to execute. Based on the 105000 instructions that are present already. If I want the program to execute in 10 seconds, then I require a board with processor clock of 105000/10 = 10500 Hz, if CPI is 1. However if CPI is not 1, let's say 5. Then the speed will increase by factor of 5. i.e. 52500 Hz. And if I want it to be even faster, maybe 5 seconds, then the board will be required to be 2 times faster. So 105kHz. For my implementation I do not require too much processing power on the processor according to what the algorithms require **(given that this is a rough estimation)**. However I might need a lot of storage for my lookup tables. Also, the algorithms may not work as intended and be several times more instructions than intended.

I think any processor within MHz range should be adequate enough for my system. It should also have a decent amount of memory, there are about 170000 words in the language, and each word will be approximately 5 characters long, that is 5 bytes. So I need about 850 KB for the dictionary alone. Not to mention all the filters and algorithms. It should have no less than 2MB memory at the very least.

Board choice

Friday, 08 July 2022 11:37

Ishaque Ramedies at 2022/07/08 11:43
To ensure that I'm on the right track with the whole calculation, I think I will look at other research papers and determine what kinds of boards have been used and if mine is good enough.

For now, the options that I have within my budgets will be looked at one at a time.

~~Option 1: 3857 -~~
~~FEATHER M4 EXPRESS ATSAMD51J19~~

~~From <https://www.digikey.co.za/en/products/detail/adafruit-industries-llc/3857/9553567>~~

~~It's a 32 bit MCU, embedded evaluation board. With a 120MHz ARM Cortex M4. Not sure if I can use this, since its compatible with Arduino IDE, however it is also compatible with Python language. There is no EEPROM and only 512KB flash and 192 KB RAM. However it seems that there is a SPI Flash with 2MB of storage. It's cheap and might be a good idea if used correctly. Price R368. There's also a similar board to this but a different model.~~

I looked at the requirements of the boards that we are allowed to use and it's safe to assume that I cannot use this board because it can use the Arduino IDE.

~~Option 1: DEV-16267~~

~~Uses the ARM Cortex M0 and Cortex M4 MCU 32-bit platform. The clock speed runs at 204MHz with low power consumption. There is a 3 stage pipeline, so processing will be faster than what was calculated for. There is 2MB flash memory and analog output, so it should be fine to interface with a speaker. It also includes UART and I2C so it should be fine to interface with an ESP email module.~~

I checked the support for this device and apparently it works similarly to Arduino, so I won't be considering this device.

There is only 1 option that I can consider at this moment that is in stock.

Option:
CY8CKIT-062S4

From <https://www.digikey.co.za/en/products/detail/cypress-semiconductor-corp/CY8CKIT-062S4/15652909>

It's a PSoC device which is a programmable system on chip device. It runs with an ARM Cortex M0+ and an M4. It is programmable from the M0+, however it is possible to independently designate processes between the cores. So I might implement the email on the M0+ and all the algorithms on the M4. It has Arduino shields but it doesn't use the Arduino IDE so that's fine. As for peripherals, I need serial communication for the email system and PWM for the speaker output.

It supports PSOC Creator, which is a new software that I haven't used before and it should be interesting to handle. It supports only C language which is fine, I should be able to accomplish my work here. The only problem is how to interface the email module with the device. The email library that I was planning on using was the Arduino library, but that runs in C++. I need to research how to use this library with the PSOC.

Re-evaluation of board selection

Wednesday, 27 July 2022 10:51

So I got the PSoC and was using it for about a week and got UART running for debug purposes, however as I went on to start working on the email system requirement, I found that I needed a library to implement it. I was looking for assistance on this and found no such libraries for this device. So instead I started looking at how I could interface the device with a library from a different device. I bought an ESP01 wifi module and found out that I would have to interface it. After that I would have to use a email library. The library isn't really an issue, however, interfacing the ESP01 module with the device is a big challenge. I now need to consider a different board with a WiFi peripheral and this sets me back about 3 weeks in progress. I've been asking some students and some seem to say that using an ODROID is good, but the issue is the community and support for it. At this point the processor doesn't matter, since my algorithm will run on any kind of processor. So I need to draw up a system diagram and include all of the peripherals that I will be considering. Once I have this diagram, I can decide which ODROID I want to use.

I'll just list the features as I see them from a description.

Firstly the user input will be email, so I need to be able to have internet access, meaning Wifi. And also the email will be implemented with a library so I need to look for library support for this. Also my actual requirement is to display the email on a display device. So I need to get something like an OLED display.

1. WiFi internal or able to access internet.
2. Email library (most likely C, or python). I found a C library that I can use, and the python one looks easy to implement.
3. Interfacing with display module.

Next I will have the output to consider, it will be output from a speaker, so I need to look into that.

4. Speaker interfacing, so maybe a device that supports DAC output like a 2 pin speaker.

Then I have to consider my actual algorithm. I need access to a dictionary so that I can do comparisons for my spell checker as well as my profanity filter. The software should also support signal generation and signal output. I think I can manage this if the IDEs are just like python or even C.

5. Supporting platform like C, C++, Python (preferably). Not any sort of machine languages since the implementation is on a software level. And hardware interfaces are done with drivers and libraries.
6. Strong processor, with adequate memory to store dictionaries.

New Board Choice

Thursday, 28 July 2022 09:26

I have decided on the ODROID C4 board. It runs a Linux OS or alternatively android, but I won't be using that. I can download any IDE with the ODROID, but I found that I can use a python wiring library to configure hardware peripherals of the board. So it is ideal. I have already starting looking at libraries that I can use for databases and the email system. Details about the board can be found here: https://www.hardkernel.com/shop/odroid-c4/

I bought it from cyber connect as it was the only one in stock locally.

Interfacing the device

Monday, 18 July 2022 10:32

I just received the board and in the meantime have been looking at tutorial videos on PSoC creator and installed the software. I didn't realise that it came with a thermistor, cap sense functionality, and an ambient light sensor. I don't think It will be used for anything in my project but I guess it's useful features.

I just attempted to connect to the serial interface which should already have the default settings configured. I just plugged the USB cable in and followed the box instructions. I managed to view the default set up, which is the ambient light sensor and the thermistor readings on a serial port. Going to attempt to run an example program on my device.

I tried to get anything to run on the device with PSoC creator, but for some reason the software keeps on hanging when I want to program the device. I think I'll stop working for now and continue tomorrow.

Programming the device

Tuesday, 19 July 2022 10:46

I'm going to continue working today where I left off yesterday. I'm going to try programming the device.

I tried once again and the software still hangs when it is supposed to program the device. There is an error in PSoC creator: **psoc creator does not support using it at this time**. I'm starting to get a little worried about my device compatibility and if I'm using the correct project settings. I'm going to attempt to solve this issue.

From what I have seen in forums, the solution is to reset the device or update the KitProg firmware (the interface that is used to program the M0+) or to simply restart the software. I tried all of this and It does not work. I'm starting to think that my device is obsolete and not supported anymore. I was looking at the product release and it seems to be relatively new. The only other option is possibly to update PSoC creator or downgrade it if it's too new and only the older version supports it according to another forum.

I tried to change the software version but still the same thing. I'll have to read any type of information that I can find on the device (datasheets, guides, etc, relating only to my specific device). I found that the software that I should use is ModusToolbox, which is apparently an upgrade of PSoC creator, but was not mentioned in many places. I also found some tutorial videos on ModusToolbox (not many but still a decent a few to help get started). I will download the software today and try to configure the device tomorrow.

Programming the device (continued)

Wednesday, 20 July 2022 10:57

So yesterday I ended off with downloading ModusToolbox. I started watching some tutorial videos and setting up a device seems simple enough. I'm going to start adding links and images when I start getting somewhere. The goal is to attempt to program my device with an example project today and then possibly add on to it with my own project. Ok, so it finally started working. I am able to program my device with template projects. I still need to properly learn how to add peripherals on my own, but it doesn't look too bad for now. I'm going to continue watching videos in the meantime, but here is a list of the steps that I followed to get here:

- I first looked for some resources on my device: CY8CKIT-062S4. (Don't look for resources on PSoC 6, as that category is too broad. I found this document: https://www.infineon.com/dgdl/Infineon-CY8CKIT_062S4_PSoC62S4_pioneer_kit_guide-UserManual-v01_00-EN.pdf?fileId=8ac78c8c7e7124d1017e962f98992207
- On page 16/40 of the document, it describes that it requires ModusToolbox, so I downloaded the software and followed the steps for setting up an example.
- The example I used was UART so that I can check the output on terminal.

That's about all I needed to do to get an example running. I will implement UART on my own for debugging and also implement it to work with the wifi module.

Proposal revision and Budget

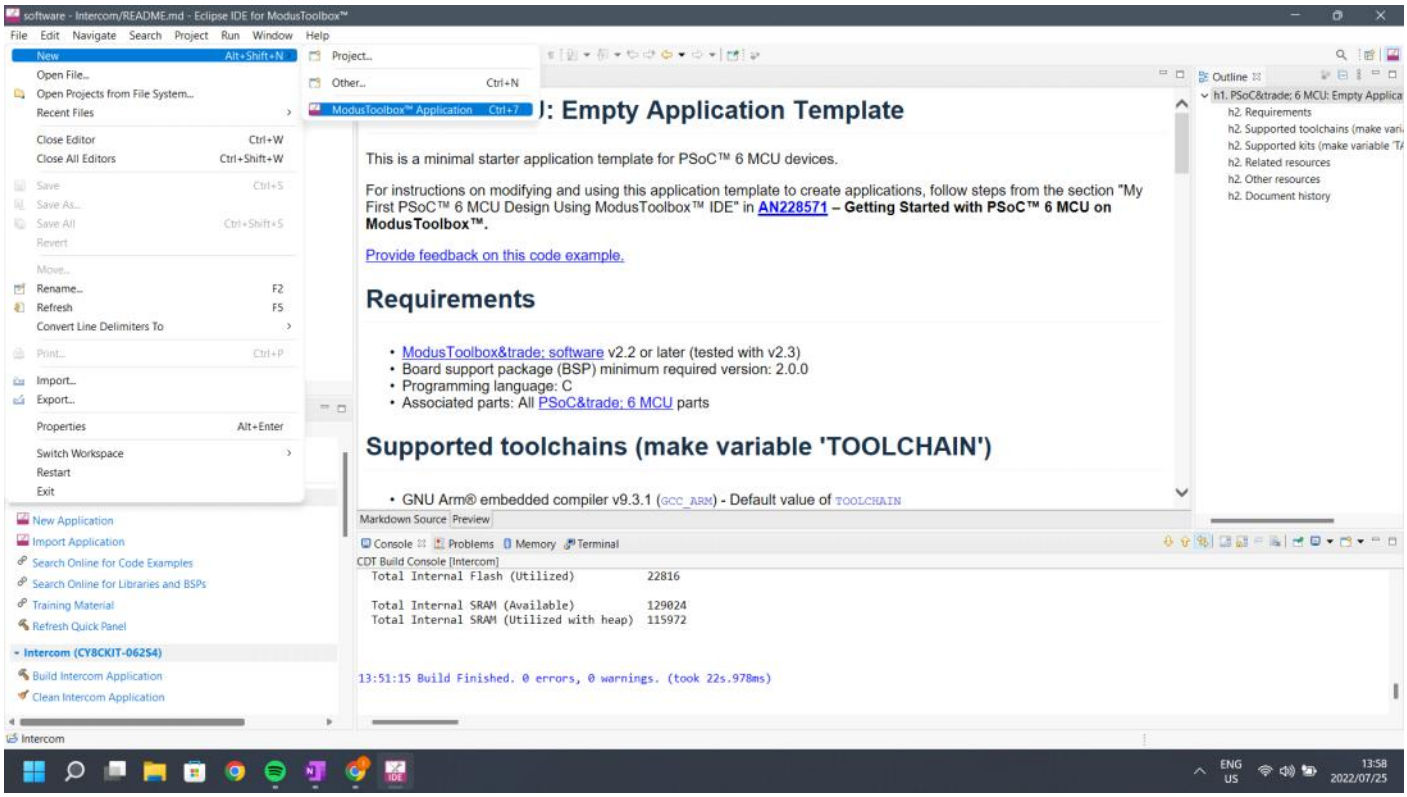
Thursday, 21 July 2022 11:02

I think the lecturer maybe misread my proposal template or maybe I didn’t explain my thing properly. I am going to revise my template after emailing him for some guidance on how to improve my thing, this needs to get done now. I'm also going to look for a budget template on Clickup and with regard to that I need to look for an email module to use and start researching ways to output some audio out of my system. I'm going to work on these things in the meantime and continue when I have everything sorted out.

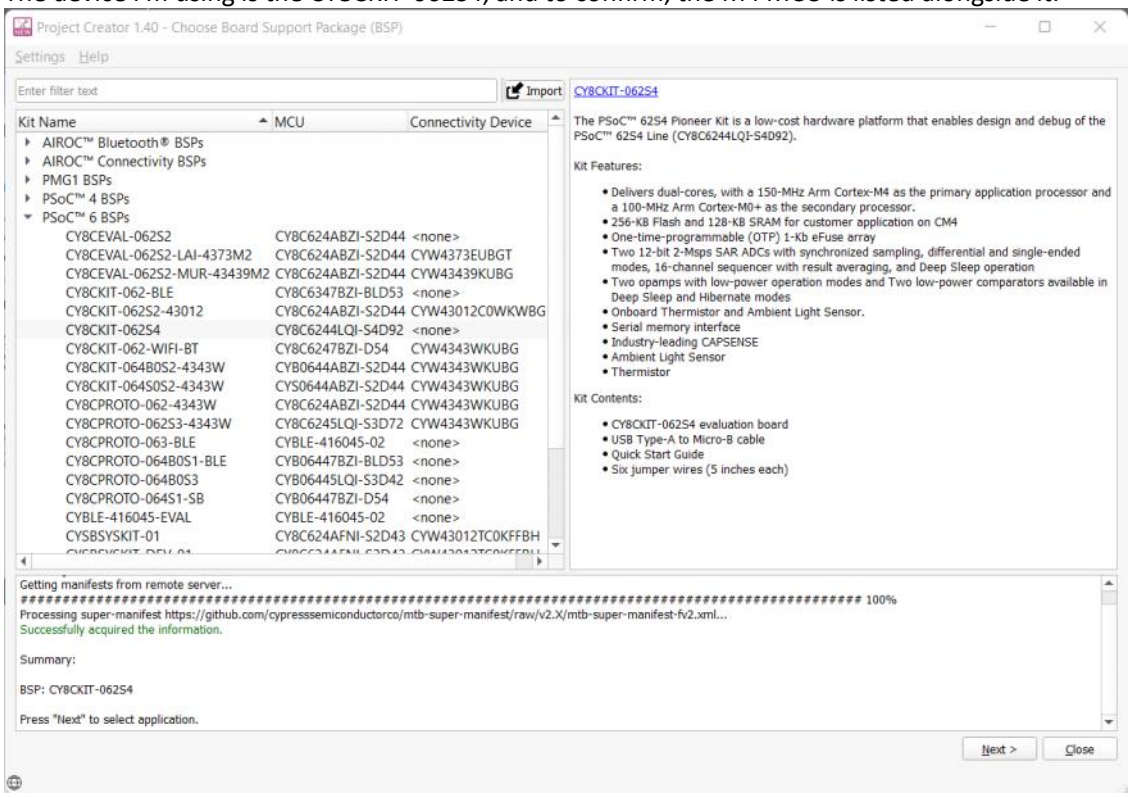
Project set up

Monday, 25 July 2022 12:20

So to get started with ModusToolBox, I have to create a modus application:
File->New->ModusToolbox App

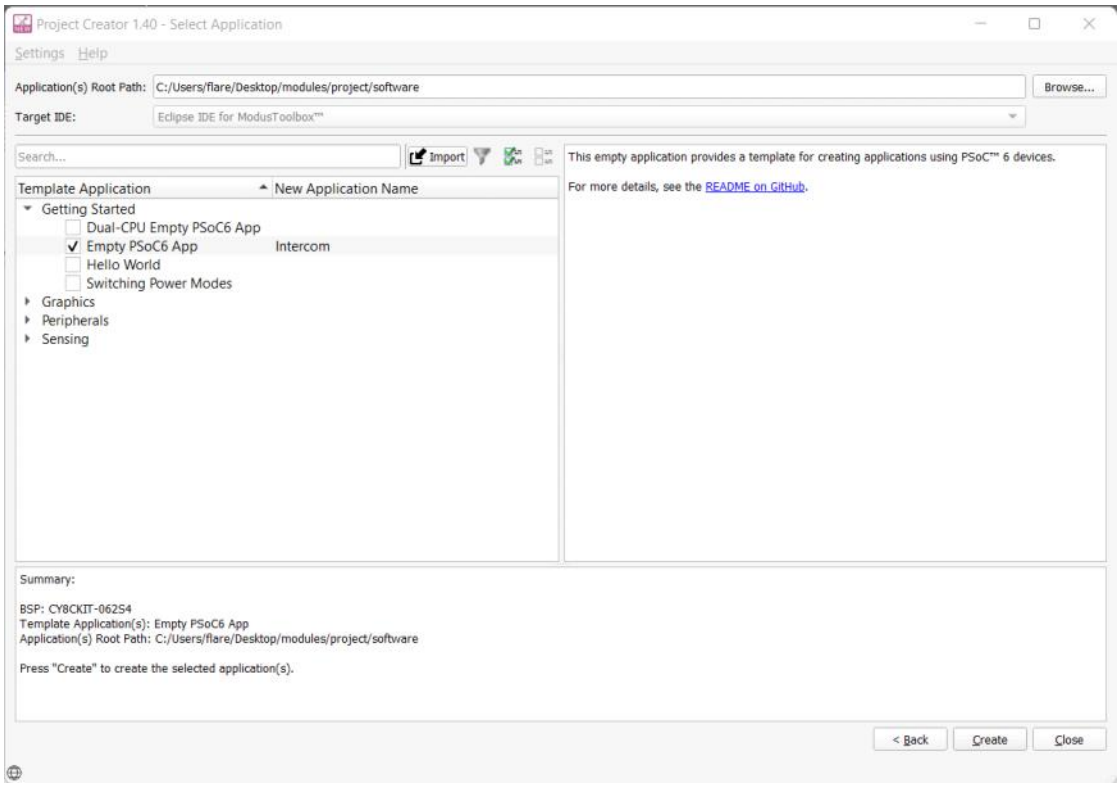


After that I select the device I'm using in the Project creator
The device I'm using is the CY8CKIT-062S4, and to confirm, the M4 MCU is listed alongside it.

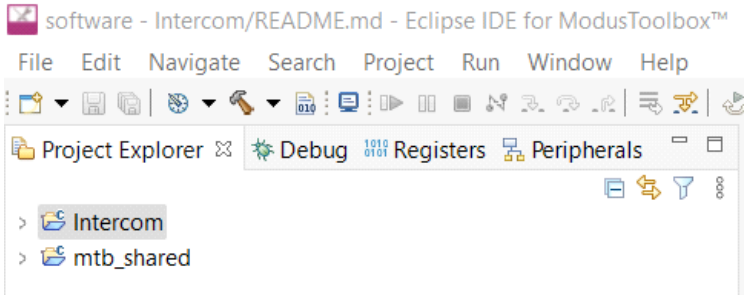


After this I just select the template, in this case I'll just use an empty PSoC6 template and name the application 'Intercom' for my intercom system. It might be useful to set up a dual CPU empty app, but I'm not sure how that will work so for now I'll just do this. I might want to try that out later in

some spare time.



After setting up the project, it shows up in the left hand side of the project explorer tab.

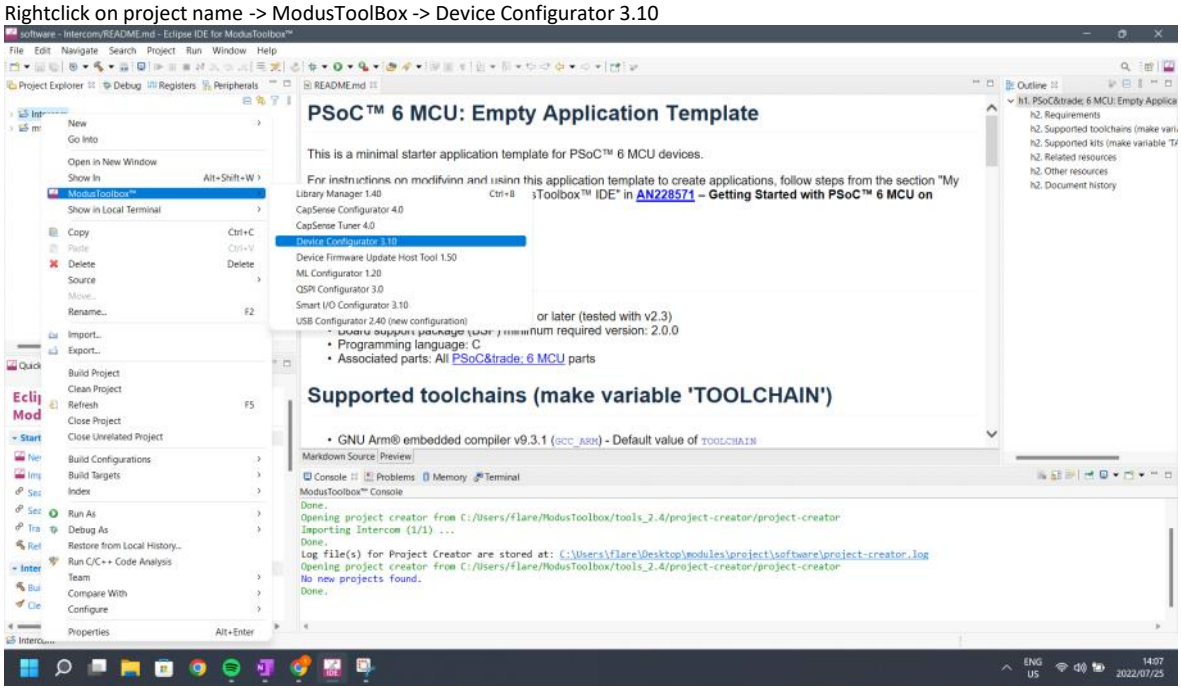


Setting up UART peripheral

Monday, 25 July 2022 14:06

It is important to note that the project should first be built successfully before setting up a UART is possible. Only when the build is completed, then the following menu will appear when right clicking on the project.

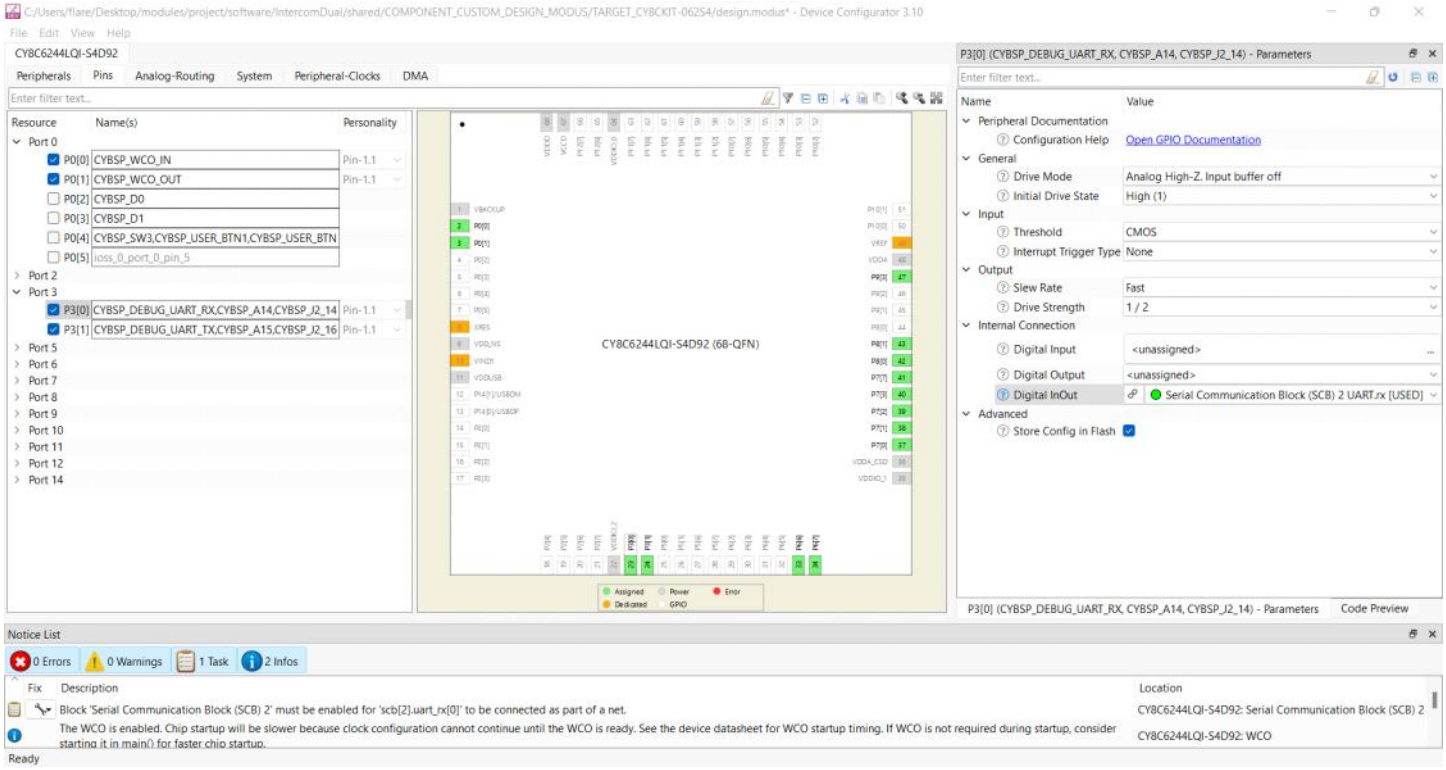
Rightclick on project name -> ModusToolBox -> Device Configurator 3.10



NOTE: I set up another project using the dual core template as mentioned before. Because as I was about to set up the UART interface I saw that I could configure UART through the KitProg3, which is the M0+ as well as the M4. I prefer to set it up on the M0+ since it will always be active when not in use. The M4 is for the heavy processing tasks.

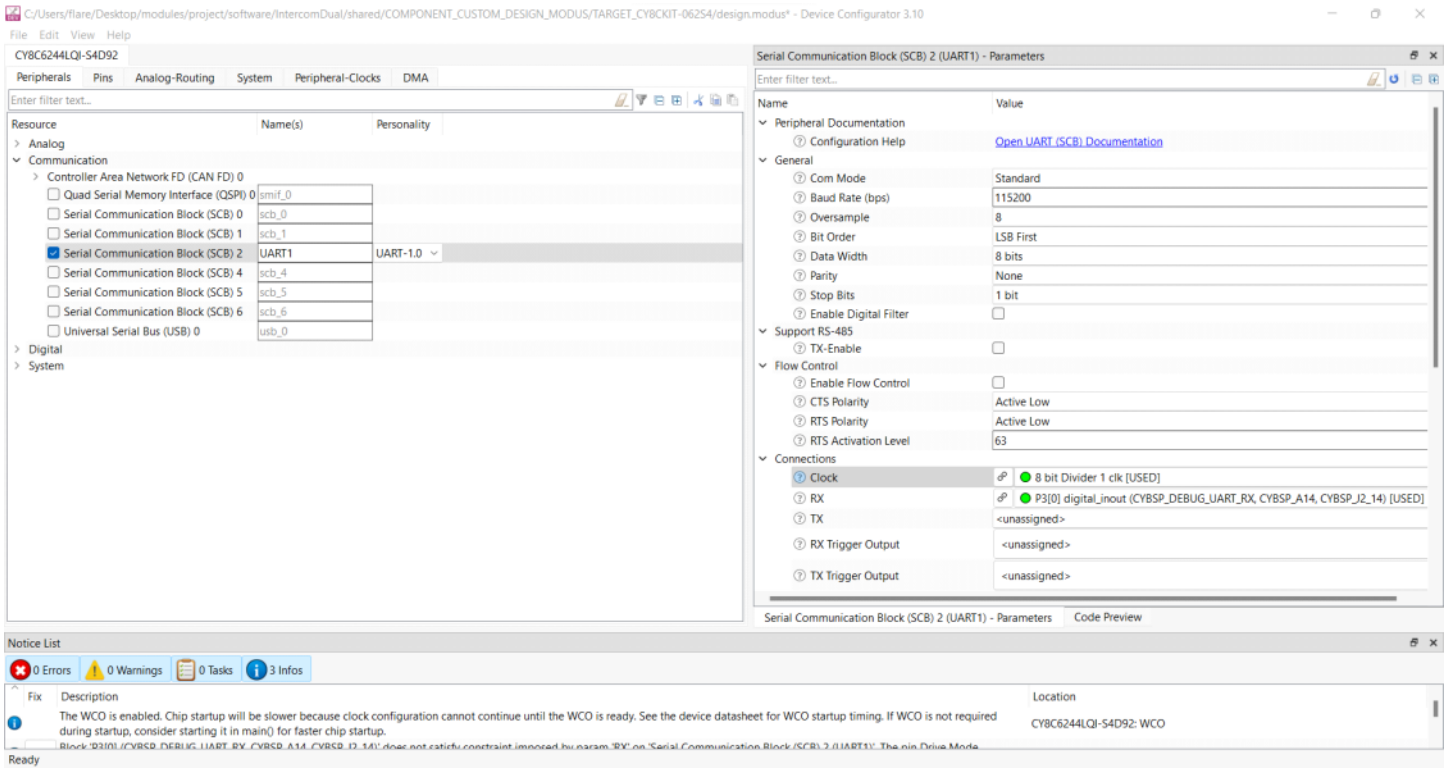
The device configurator on the M0+ side of the project:

I see that I need to configure pin3.0 and pin3.1 as seen at the back of the board. I can also look up a datasheet for the pins to configure RX and TX on. And after that I have to select UART as the Digital InOut as seen on the right. I can click the link next to the digital in out which will let me select the SCB that I have to enable.



As seen under the peripherals tab, I need to enable SCB2, and I'll just give it a name that I can

reference in my code 'UART1', and the personality has to be set for UART. Also I need to enable a clock for the baud rate to match. On the right side under connections, I just use the 8 bit divider 1 clock, and it automatically sets the divider for me to achieve a baud of 115200. Important to know if I'm on the right track, I have to constantly check the notice list below if there are any tasks that have to be sorted out. In the previous snippet it told me to set up the SCB2, so I knew how to carry on from there.



After all of this is done I can carry on to the Code to print data on the terminal. I first have to save the device configurator to apply the set up.

Hardware evaluation

Thursday, 28 July 2022 09:33

Upon further inspection of the board, it has no support for wifi modules. I cannot set up any wifi module with it, and there is little support for any email client. The way that files are set up in the software make it confusing to know where to add libraries to and if the libraries are even supported. I need to re-evaluate my board choice.

Re-evaluation of board selection

Also, I am restructuring the rest of my report to just include lab book content, since I don't want to jump back and forth between sections. The next section will contain the rest of my lab book information.

Budget and minutes

Thursday, 28 July 2022 09:36

Today I am going to complete my budget document, organize my meeting minutes and then set up my project report.

Budget and Minutes are complete.

ODROID C4 setup

Monday, 01 August 2022 09:52

This weekend, I got an HDMI-VGA converter and I got a screen to use as well as a 64GB microsd card to boot ubuntu from.

Firstly I googled how to get started with odroid c4. I came across this link.

https://wiki.odroid.com/odroid-c4/getting_started/os_installation_guide?redirect=1

I downloaded Ubuntu MATE 20.04 desktop image and flashed the microsd using etcher. The process didn't take too long. I then connected the screen up to the odroid and put the microsd card into the odroid and plugged the power supply in to power the board.

I was faced with the login screen, there was no password however, but for administrative requests the password required is 'odroid'. I then attempted to download an IDE for coding python in, like pycharm however it doesn't support an ARM64 compiler or something like that. I started looking for alternatives and came across a way to install python packages. The os is preinstalled with python 3 so it shouldn't be a huge problem to overcome. I do however need my system to download and install all updates before I can install python packages so that's all I did for then. It's seems that it will take a while.

Project report

Monday, 01 August 2022 09:59

Today I am working on my project report and organizing all of my notes for the week. We have a lot of ERP information to organize so I'll continue with my report for today and tomorrow and continue attempting to install an IDE on my device in the meantime.

Continued hardware setup

Tuesday, 16 August 2022 15:39

The odroid installed all updates successfully, I still need a screen adapter so that I can use the board on campus, however, I think I can just bring my own screen and set that up on campus. I attempted to install python packages on the terminal and it seems to work. There was a slight issue with internet connection, so I had to restart the system. It is now connected. This might be a problem in terms of field conditions, if it has to constantly be restarted in order to operate with internet connection. If it happens again it will be noted. The python package numpy has been installed and tested with a generic text editor. It works.

I also attempted to install an IDE for the device so that I can identify if it is possible to code in python. Python seems to be the most ideal programming language for audio files and filter implementations. Unfortunately amd64 compilers are not supported, but arm64 compilers are supported. The python IDE I am now using is VIM. It is difficult to use so I decided that I'm going to code on my laptop and then transfer the files over at a later date for testing.

Email client attempt

Wednesday, 17 August 2022 23:47

I attempted to receive emails on a python file. I was just following instructions from a website. But then as I continued to read, it mentioned that gmail does not support other softwares using the service with only a password anymore. So I have to use a gmail API. I left this for the time being because I have to now create a gmail account for the system and then I can attempt to receive emails. I want to get this out of the way though, so I'll attempt to finish it soon. I will post the links and screenshots of the sources and steps I followed.

Klatt synthesizer configuration

Tuesday, 16 August 2022 15:47

I started implementation of a resonator, which is another term for a bandpass filter. The filter is modelled in such a way that I can change the resonant frequency of it and the bandwidth of it. I used the equations from the Klatt synthesizer paper from 1980. I'll just put some screenshots of the diagram for the resonator and what I did so far in code in the following pages.

Ok so this first diagram is a block diagram of the entire synthesizer. The R's represent the resonators. I have to configure the synthesizer in a way that there is a cascade section (top) and a parallel section (right). There are also elements that I am not sure about yet, but the diagram looks very promising. It seems to have an impulse generator on the left, and the RN is nasal resonators, and RG is glottal resonators. I have no idea what those are but I'm assuming it's some sort of noise signal that you add to the synthesized signal. There are also low pass filters LPF for the impulse response that sets up the diagram. I have to learn about that as well. The terms A are for amplitude control, I'm not sure what that is right now but I will read up more on the document.

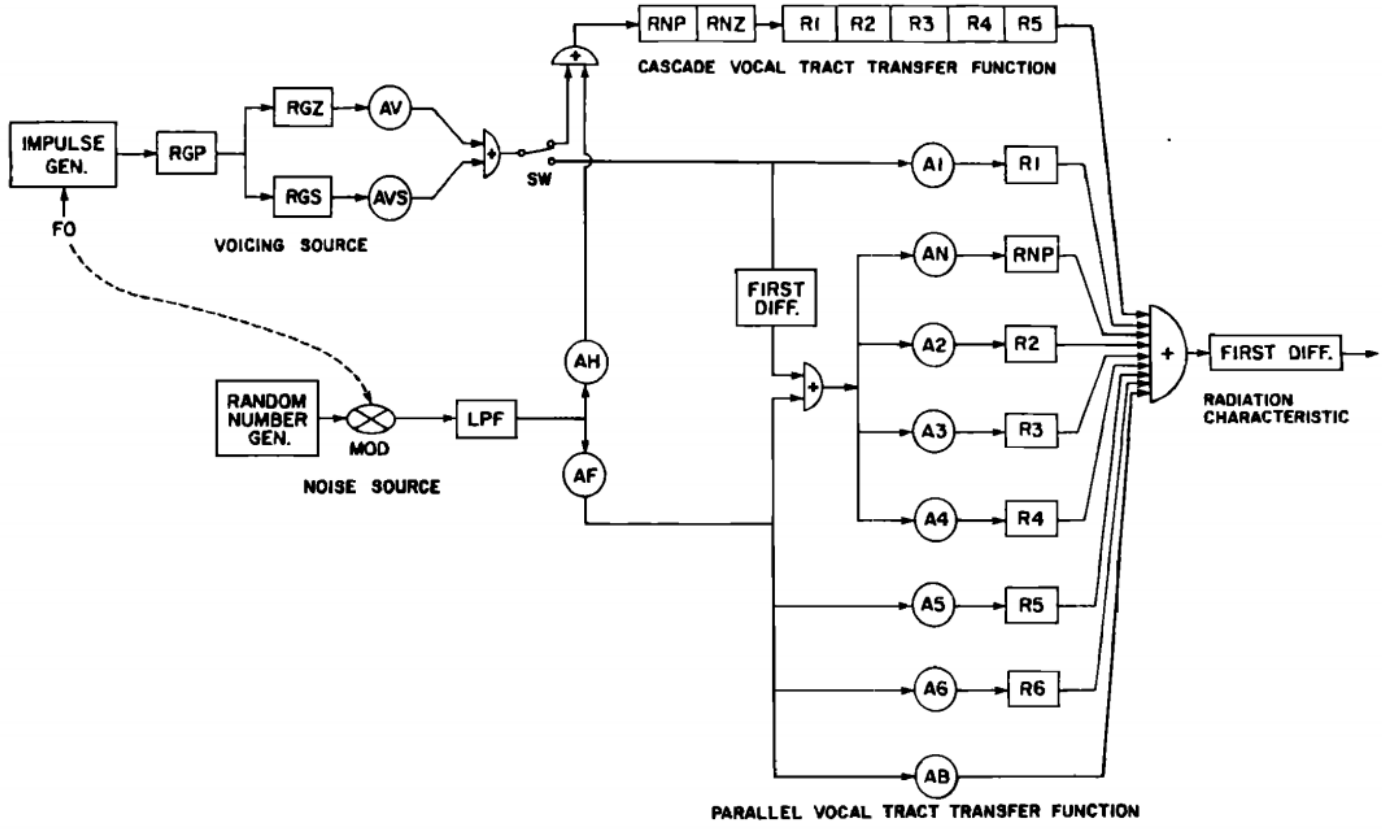


FIG. 6. Block diagram of the cascade/parallel formant synthesizer. Digital resonators are indicated by the prefix *R* and amplitude controls by the prefix *A*. Each resonator *Rn* has an associated resonant frequency control parameter *Fn* and a resonance bandwidth control parameter *Bn*.

This is the cascade and parallel configurations mentioned earlier.

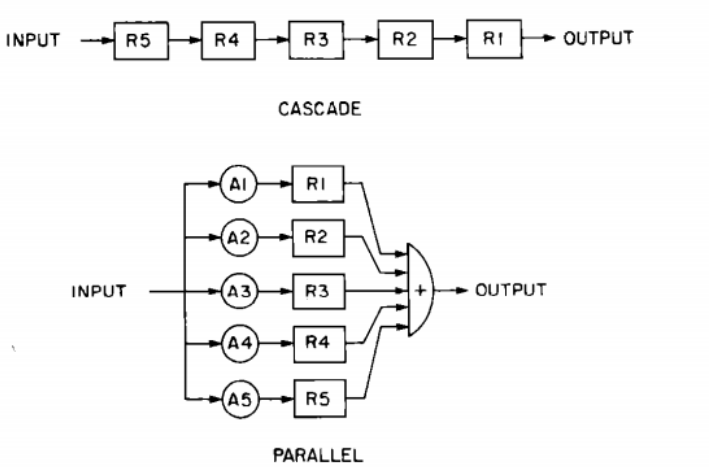


FIG. 3. The transfer function of the vocal tract may be simulated by a set of digital formant resonators *R* connected in cascade (the output of one feeding into the input of the next), or by a set of resonators connected in parallel (where each resonator must be preceded by an amplitude control *A*).

Resonator implementation

Wednesday, 17 August 2022 22:39

So for the information I found on resonators, This equation that relates the filter output to the input is actually quite useful. The syntax is just a bit confusing, but it basically means that the output is updated with the previous calculated values of the output y(nT-T) and y(nT-2T). These values are just modelled as buffers, after looking at some examples of how it was done, not with the exact same configuration, it was pretty simple to implement overall. The A,B,C terms are constants that are defined from a transfer function of the filter model.

Digital resonators

The basic building block of the synthesizer is a digital resonator having the properties illustrated in Fig. 5. Two parameters are used to specify the input–output characteristics of a resonator, the resonant (formant) frequency F and the resonance bandwidth BW. Samples of the output of a digital resonator, $y(nT)$, are computed from the input sequence, $x(nT)$, by the equation

$$y(nT) = Ax(nT) + By(nT - T) + Cy(nT - 2T), \quad (1)$$

where $y(nT - T)$ and $y(nT - 2T)$ are the previous two sample values of the output sequence $y(nT)$. The constants A, B, and C are related to the resonant frequency F and the bandwidth BW of a resonator by the impulse-invariant transformation (Gold and Rabiner, 1968)

$$C = -\exp(-2\,PI\,BW\,T),$$

$$B = 2\exp(-PI\,BW\,T)\cos(2\,PI\,F\,T), \quad (2)$$

$$A = 1 - B - C,$$

The filter model is shown here, it has unit delays, but it can be implemented using a matched z-transform. So I just needed to derive the expression for the transfer function from the diagram, but then I saw that it was already given just below the diagram. The spectrum of the output uses a sample frequency of 5000Hz, and the goal is to try and get the output that is shown in this diagram using a bandwidth of 50Hz and resonant frequency of 1000Hz.

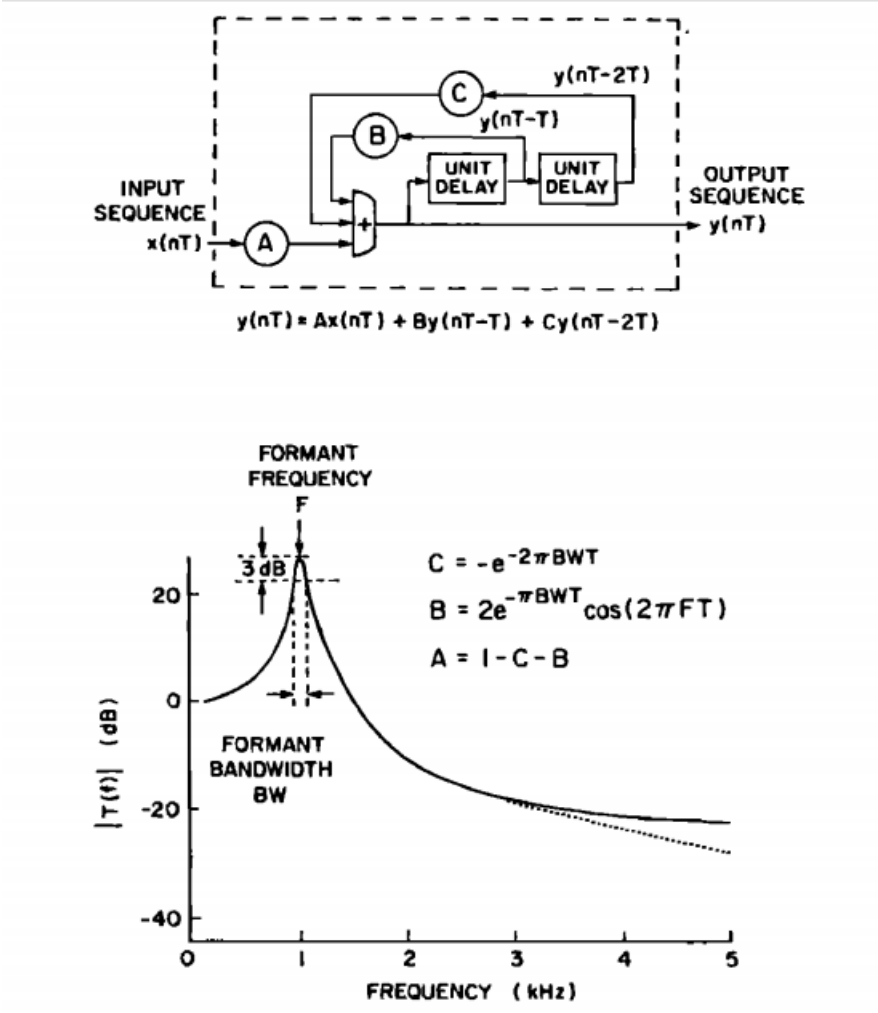


FIG. 5. The digital resonator shown in the form of a block diagram in the upper part of the figure has a transfer function (magnitude of the ratio of output to input in the frequency domain) as shown. In this example, $F = 1000$ Hz and $BW = 50$ Hz. The transfer function of a corresponding analog resonator is shown by the dotted line.

One of the important parts of the resonator is the transfer function. This confirms the frequency response of the filter. The filter output itself will only show the time response. So, it's a bit less intuitive. But if the transfer function yields the same response as the one shown in the figure above, it is most likely correct.

A digital resonator is a second-order difference equation. The transfer function of a digital resonator has a sampled frequency response given by

$$T(f) = \frac{A}{1. - Bz^{-1} - Cz^{-2}}, \qquad (3)$$

Resonator implementation (python)

Wednesday, 17 August 2022 22:47

Filter class:

I made a filter class which takes the constants as input, the resonant frequency and bandwidth have to be predefined, but I'll try and make it more generic later. So far this is what I have though.

```python
#The code below implements a digital resonator, the basic
#building block of the synthesizer.
#Derived from Klatts paper: 1979

#imports
import numpy as np
import pylab as pl

#now implementing the configuration:
#y(nT) = Ax(nT)+By(nT-T)+Cy(nT-2T)

class digital_resonator:
    def __init__(self,_A,_B,_C):
        self.A = _A
        self.B = _B
        self.C = _C
        self.buffer1 = 0
        self.buffer2 = 0

    def filter(self,x):
        y = self.A*x + self.buffer1*self.B + self.buffer2*self.C
        self.buffer2 = self.buffer1
        self.buffer1 = y
        return y
```

So it might look a bit complicated but it really isn't that complicated. A, B, and C, are the constants, and buffer1 and buffer2 are the previous 2 samples that the filter outputs, given by the equation on the last page. Those are essentially the unit delays from the diagram.

Ok so the summation in the diagram is just the same as the output y, and that output y is given as A*x + B*buffer1 + C*buffer2.

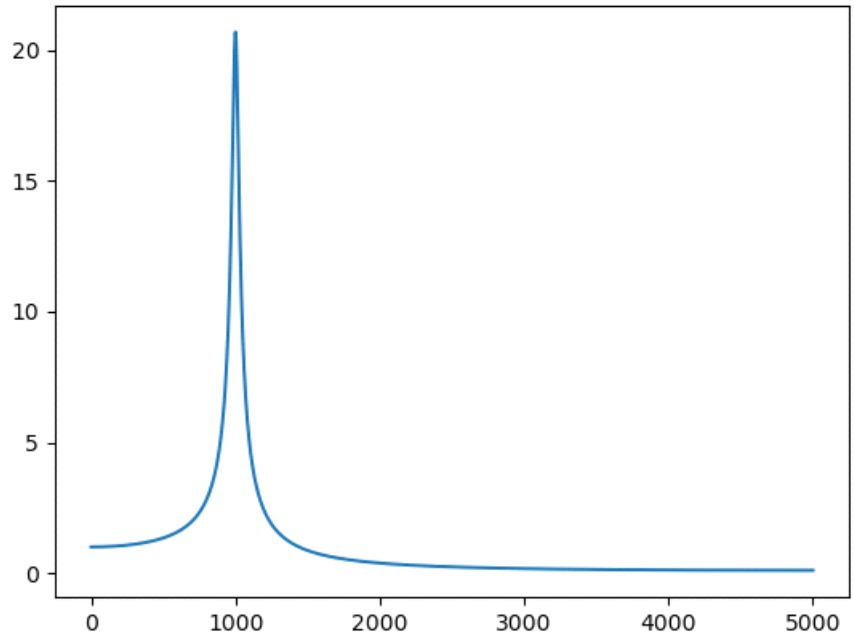
And then because of the unit delays, the buffers are updated, so buffer2 gets updated with buffer1 value and buffer1 gets updated with y value. Important here that the filter has to run through a for loop with a input signal. That input signal can be anything, but I just made it an impulse response. I will have to experiment with this later.

Ok now for the rest of the code, as seen in the figure I defined the constants, sampling frequency, and other parameters. To create a resonator, I just call the class function with whatever my desired constants should look like and then I Plot the filtered signal with any amount of samples. I just made an impulse at input 20 to see what it would look like. I can plot it there and then I also plot the transfer function below that.

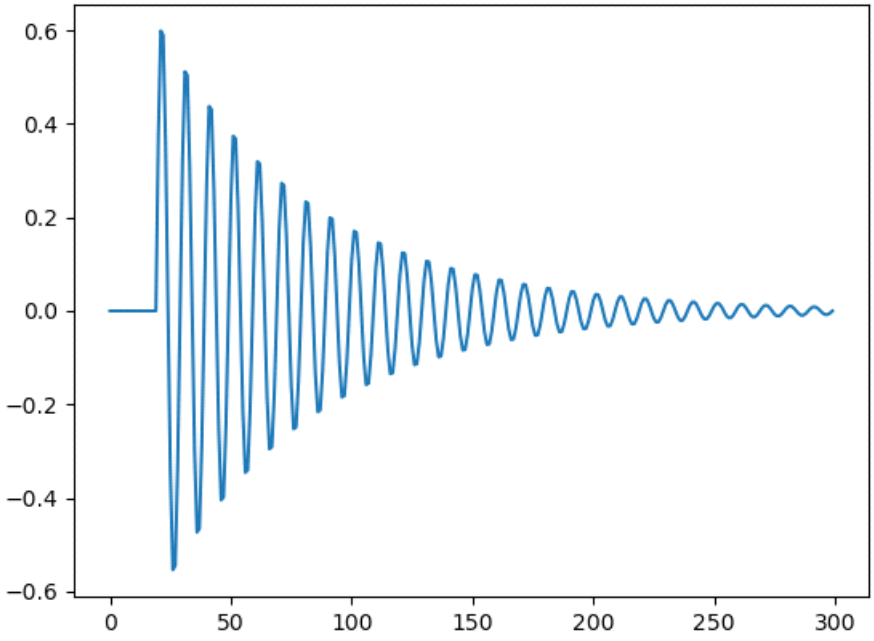
```
#These parameters differ for each sound that will be generated
#Resonant frequency is F in Hz
#Resonance bandwidth is BW in Hz

F = 1000
BW = 50

#The sampling frequency S changes accuracy of sound signal
#Sampling period is T
S = 5000
T = 1/S

#constants
C = -np.exp(-2*np.pi*BW*T)
B = 2*np.exp(-np.pi*BW*T)*np.cos(2*np.pi*F*T)
A = 1-B-C


#creating the filter
r1 = digital_resonator(A, B, C)

x = np.zeros(5000)
x[20] = 1
y = np.zeros(5000)

#resonator response
for i in range(len(x)):
    y[i] = r1.filter(x[i])
```

```
pl.plot(y)

#frequency response
#plot omega from 0 to pi
w = np.arange(0,np.pi,0.01)
#convert omega to z values using the relationship
z = np.exp(1j*w)

h = (A)/(1-(B*z**-1) -(C*z**-2))

pl.plot(np.linspace(0,2500,len(h)),np.abs(h))
```

The output of this transfer function looks like this. Note that in the pdf they mentioned that sampling period and it's 1/F. They also said that it was 0.0001 for the 5kHz simulation. However that would mean that the sampling frequency is 10kHz not 5kHz. Because 5k sampling frequency corresponds to T = 0.0002, which is not what they said in the pdf. So for the parameters, S = 10kHz, then T = 0.0001s, F= 1kHz, BW = 50Hz, and linspace time axis is 5000Hz. The transfer function in the PDF was going down to -20dB at 5000Hz, but maybe that doesn’t matter in this case because 0dB can be considered the noise floor. It is still a little troublesome that the simulation doesn't align with the expected outcome.



As for the frequency response, with x[20] = 1 :



I'm not entirely sure if this is correct, but I'm going with this for now. I want to try sending in a sine wave, and then monitor the result, or I could just simulate a bandpass filter and attempt to input an impulse signal. Anyways, I need to confirm that this works before continuing.

Email client implementation

Monday, 22 August 2022 09:21

Going to attempt the email client today. The first dependency required is below:

```
$ pip3 install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib
```

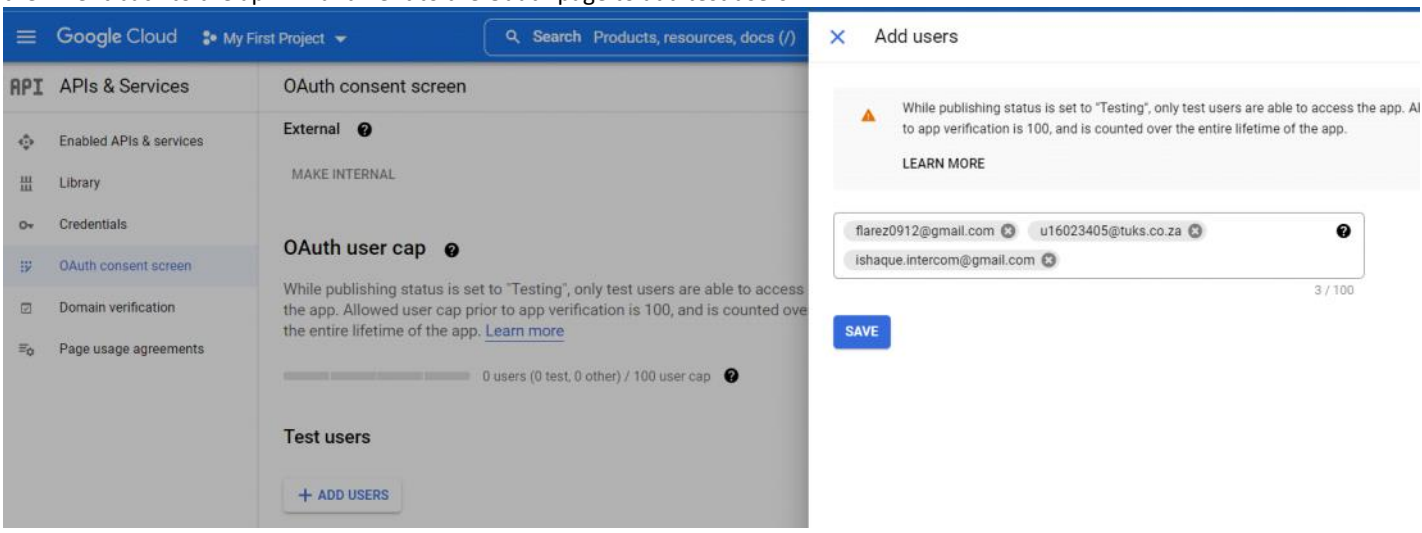
I'm going to try to proceed without this for now. I created an email address for the intercom. The details are

Email: ishaque.intercom@gmail.com
Pass: 16023405

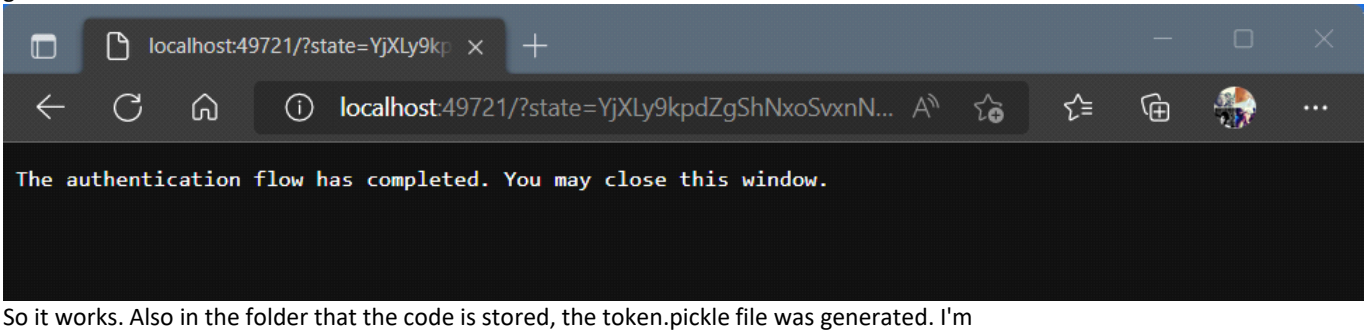
I can access the email, so on to the next step. I had to go to this link and search gmail api and enable it https://console.developers.google.com/apis/dashboard . It now seems to be enabled. I am now following the rest of the steps according to this link: https://www.thepythoncode.com/article/use-gmail-api-in-python#Reading_Emails

I clicked create credentials, top right. I went through a bunch of steps and then created a Desktop app. I downloaded the credentials and it's now in a .json file. I'll make a copy of the file and rename it credentials.json. Now for the code. It is simply copy pasta.

In the code, I changed the email to the email I enabled an API with. I then authenticate save the credentials to an authentication token. I'm getting the error: No module named 'googleapiclient'. This probably has to do with the dependencies so I'll install it now. I installed the dependency on cmd terminal and it installed successfully. Without the $ sign. I then attempted to run the code with the first 2 code snippets copied. It said that I had to input a user to access the IR intercom project. I selected the email account that I set up initially but it had an error saying that access was denied. I then went back to the api link andwent to the Oauth page to add test users:



I added all accounts that I currently have just in case it doesn't work on other accounts. This should work now. I then went back to try logging in with the initial email: ishaque.intercom@gmail.com. I get this



So it works. Also in the folder that the code is stored, the token.pickle file was generated. I'm assuming it stores the credentials now. If I run the code again, it runs without me having to sign in

on the web browser. I should try this again on the hardware device to make sure everything is up and running. I can now use code for searching email and reading email. It is the only functionality we are interested in.

Ok so it works. I sent 2 emails to the intercom address and whenever I run the code it searches for emails with the Subject line that matches the Subject line in the code that I defined. It will retrieve the same message over and over again when rerunning the code. I also noticed that folders are created every time an email is read, and the folder is created multiple times for each email. So I need to get rid of those folders after displaying them. They contain the original HTML message.

So my next goal is to modify the code so that it loops until it receives an email, then when it does receive an email with the specific subject line, it exits the loop and deletes all other emails to save storage space. It could also check who the person is that sent the email and if it's not in the database of people who it is registered to, it will return to the loop, but for now I just want to store the input text to a variable and also store the sender's email address, and implement the delete email function, after reading.

I'm implementing the looping first. I managed to loop the code until it receives an email.

```
180    # get emails that match the query you specify
181    results = search_messages(service, "nada")
182    while(results == []):
183        results = search_messages(service, "nada")
184
185    print("gottee")
```

There is a bit of a delay when sending an email and when receiving an email. It depends on the internet connection I guess for the servers. Over all wasn't too bad maybe about 5-10 seconds before an email was received. Now I have to check how I can store only the message from the email and display it.

I just completed the implementation of the text only message extraction. It is however limited, because it only allows the user to input the entire email into a single paragraph for now. When it detects a \r, it stops getting the message and takes the content on the inside as the actual message. There are no error correction checks or anything like that as of yet. The last remaining thing to do is only reading new messages and not rereading old messages. After doing this, I will make the email client more generic and test it by importing it in another file, if that works then it should be ready for integration.

I think a good way to handle this is to only read messages that are unread and then mark them as read, that way it will only read unread messages and leave everything else with the same subject lines.

I decided to delete all messages with the subject line before reading messages, its simpler.

The issue with the paragraphs I see that I'm making a problem where there isn't one. I will just retrieve the entire email. The disclaimer thing is something that has to be removed from the email itself. For personal emails, the footer does not get sent. Therefore I will just have to edit my email itself in order to remove the footer or disclaimer from sending. If you think about it. The email sender doesn't send a disclaimer as a message, so my system does not need to account for this.

Now I just need to implement the display system, which is an external window display. I will need to figure out how to do this because currently the email is displaying on python console like this:

```
In [18]: runfile('C:/Users/flare/Desktop/modules/project/software/email client/
mail_client.py', wdir='C:/Users/flare/Desktop/modules/project/software/email
client')
From: Ishaque Ramedies <u16023405@tuks.co.za>
Date: Mon, 22 Aug 2022 14:20:16 +0200
Subject: School
To: ishaque.intercom@gmail.com
==============================================
Let's goo

--
This message and attachments are subject to a disclaimer. Please refer to
http://www.it.up.ac.za/documentation/governance/disclaimer/
<http://www.it.up.ac.za/documentation/governance/disclaimer/> for full
details.
```

Which technically is the display of the email, just not in an external window. I will get back to this at a later date, but for now it seems that email client implementation is complete. I just need to make it generic so that I can test it by importing it in another file.

Resonator verification and cascading

Tuesday, 23 August 2022 11:11

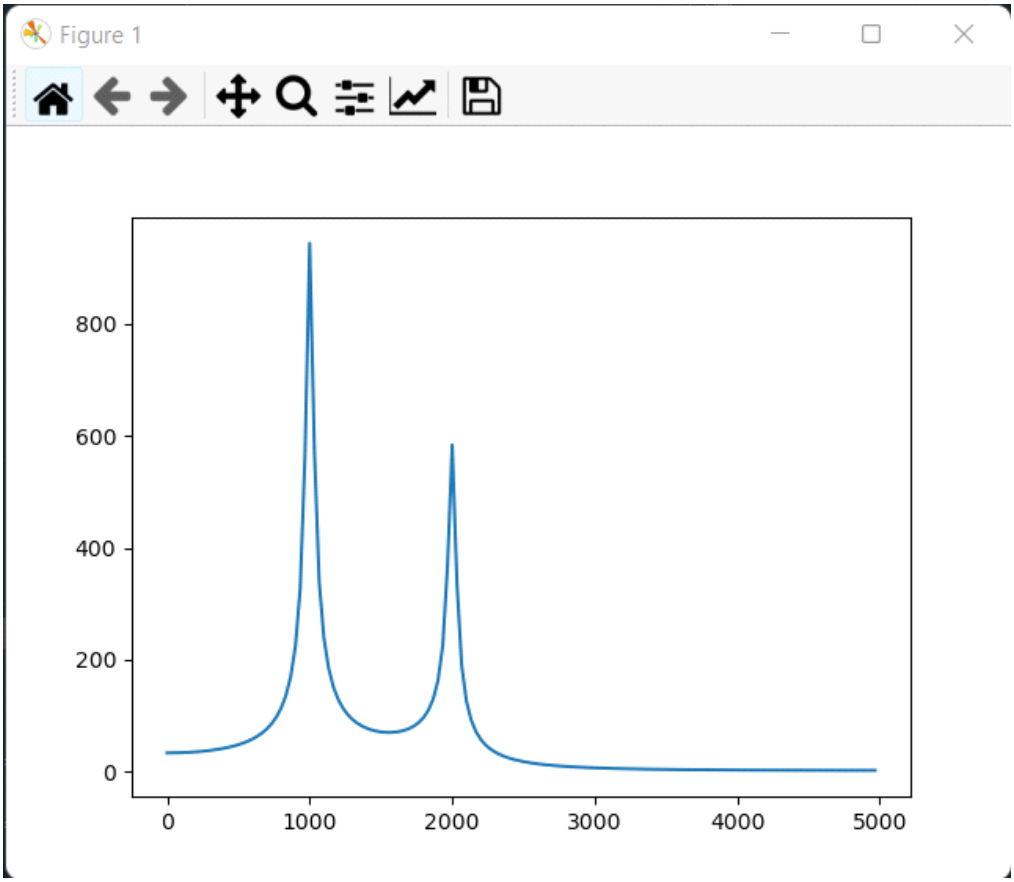
Today I decided to add the email client to the main program, so that it is integrated and I don't have to worry about it. I decided to make copies of all the files and put it into the main program folder, that way I don't have to use other imports and it makes the code look cleaner.

The other thing that I am going to do is try to verify my resonator by **plotting the inverse z-transform in matlab**. If it gives the same output then I don't have anything to worry about. However if the plot is different, I will need to re-evaluate my code and determine whether or not it is ok to use the implementation I have right now.

I'm not getting very far in MATLAB, at this point I will just have to use what I have and assume that it is correct. I'm going to find a new way to plot the filter signal's frequency response without the need for the transfer function. I think that is the best approach to take now.

I managed to plot the fft of the filter with an impulse response. The magnitude of the output signal seems to depend on the impulse response magnitude. I'm still not getting the -20dB towards the high frequency ranges. And I don't know why. There's nothing I can do about it now though. I'm going to start working on cascading or parallelizing the filters so that I get actual signals that look good. If I can combine just 2 filters I'll be on the right track.

I managed to cascade 2 filters. It was actually very simple. The input of a signal gets filtered, and then the output of that filter gets put into a different filter with a different transfer function, meaning that the transfer function A,B,C values change. So I need to create a function that gets constants given a resonant frequency and a bandwidth. I also need to create a dictionary for resonant frequencies eventually. And start testing, oh boy it's going to be a long process. This is the output of cascading a 1000Hz resonant frequency filter with a 2000Hz resonant frequency filter. Both filters have bandwidths of 50Hz. I now realise that I have to produce a signal that occurs over longer duration, so the input signal will have to be an impulse train. That will be my next task.



Klatt synthesizer components and sections

Wednesday, 31 August 2022 01:32

I found the software used for the Klatt synthesis tool. It was also made based on the paper that I am using. I decided to make my implementation similar to that, but I need to understand the different components involved in the synthesizer and how the whole thing works so that I can use it properly. The last few days I have messed around with it a little bit and I found that the formants are designed for in such a way that it splits the values up as a list of values from a certain range. For instance, if I record a vowel in Praat, and the formant for that vowel starts at 500Hz, and at the end of the vowel sound, the formant is at 400Hz, then the formant would be broken up into pieces from 500 to 400. Like 500, 499, 498, … 400, for instance if it was a linear thing. It also depends how many samples I have. Long story short, I can modify 3 formant frequencies and it produces a sound that I want. So that's a big win for me.

Currently I am figuring out how the impulse train generator is supposed to work for this task. So that I can document it. My goal for the rest of this week is implementing all of the components with understanding and developing separate tests for them so that I can put it in my final report.

My goals are to implement:

- Resonator, this is kind of complete already.
- Impulse train generator, currently working on this.
- Mixer
- Amplifier
- First difference module, I don't know what this is yet
- Low-pass filter, should be simple enough since it's based on a resonator just with a different equation
- Normalizer
- Noise generator, this is kind of important
- Switch, this is a simple 1 or 0 to choose between cascade or parallel synthesis

After these components are done I have to connect them in different sections:

- Voicing source section
- Noise source
- Cascade vocal tract
- Parallel vocal tract
- Radiation characteristic
- The output module, i.e. the sound conversion thing.

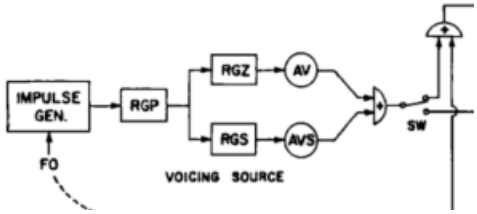
These are all according to the Block diagram of the formant synthesizer.

Also side note, I am using a different audio conversion library. It seems easier to use. It's called simpleaudio but it has some requirements. For windows, I needed some SDK installation, however I am a bit worried for the linux platform on the ODROID. Soon, I will have to see if this library can be used on ODROID. If it installs successfully then that's fine and I can continue working on my laptop. Other than that I seem to be going in the right direction.

Impulse train generator implementation

Wednesday, 31 August 2022 01:49

The impulse generator is part of the voicing source of the synthesizer as seen in the figure below. The information given in the paper is the following:



A. Voicing source

The structure of the voicing source is shown at the top left in Fig. 6. Variable control parameters are used to specify the fundamental frequency of voicing ($F0$), the amplitude of normal voicing (AV), and the amplitude of quasi-sinusoidal voicing (AVS).

An impulse train corresponding to normal voicing is generated whenever $F0$ is greater than zero. The amplitude of each impulse is determined by AV, the amplitude of normal voicing in dB. AV ranges from about 60 dB in a strong vowel to 0 dB when the voicing source is turned off. Fundamental frequency is specified in Hz; a value of $F0 = 100$ would produce a 100-Hz impulse train. The number of samples between impulses, $T0$, is determined by $SR/F0$, e.g., for a sampling rate of 10 000 and a fundamental frequency of 200 Hz, an impulse is generated every 50th sample.

Under some circumstances, the quantization of the fundamental period to be an integral number of samples might be perceived in a slow prolonged fundamental frequency transition as a sort of staircase of mechanical sounds (similar to the rather unnatural speech one gets by setting $F0$ to a constant value in a synthetic utterance), but the problem is not sufficiently serious to merit running the source model of the synthesizer at a higher sampling rate. If desired, some aspiration noise can be added to the normal voicing waveform to partially alleviate the problem and create a somewhat breathy voice quality.

So what the paper is saying is that F0, AV, and AVS are user specified parameters and according to the parameter table below, these values are variable (V) and values range between 0 and 80dB and 0 and 500Hz. So the fundamental frequency has to be greater than 0Hz for the impulse train to generate a, well, impulse train. The fundamental frequency refers to the frequency of the speaker. I assume that my fundamental frequency can be measured with Praat. Also I think that the impulse train has to be generated according to that frequency. The paper also says that the amplitude is determined by the amplitude of voicing AV, so that is another thing that I will need to determine where it comes from for my voice. It also says that the number of samples between impulses is SR/F0, which is sample rate over fundamental frequency. Sample rate is already predefined, so based on my fundamental frequency, that will determine the amount of samples between impulses. Ok, doesn't seem too bad, I just need to generate an impulse every n samples according to this formula. Apparently it is not recommended to go for higher sampling rates, so I will just stick to 10000 for now.

TABLE I. List of control parameters for the software formant synthesizer. The second column indicates whether the parameter is normally constant (C) or variable (V) during the synthesis of English sentences. Also listed are the permitted range of values for each parameter, and a typical constant value.

N	V/C	Sym	Name	Min	Max	Typ
1	V	AV	Amplitude of voicing (dB)	0	80	0
2	V	AF	Amplitude of frication (dB)	0	80	0
3	V	AH	Amplitude of aspiration (dB)	0	80	0
4	V	AVS	Amplitude of sinusoidal voicing (dB)	0	80	0
5	V	F0	Fundamental freq. of voicing (Hz)	0	500	0

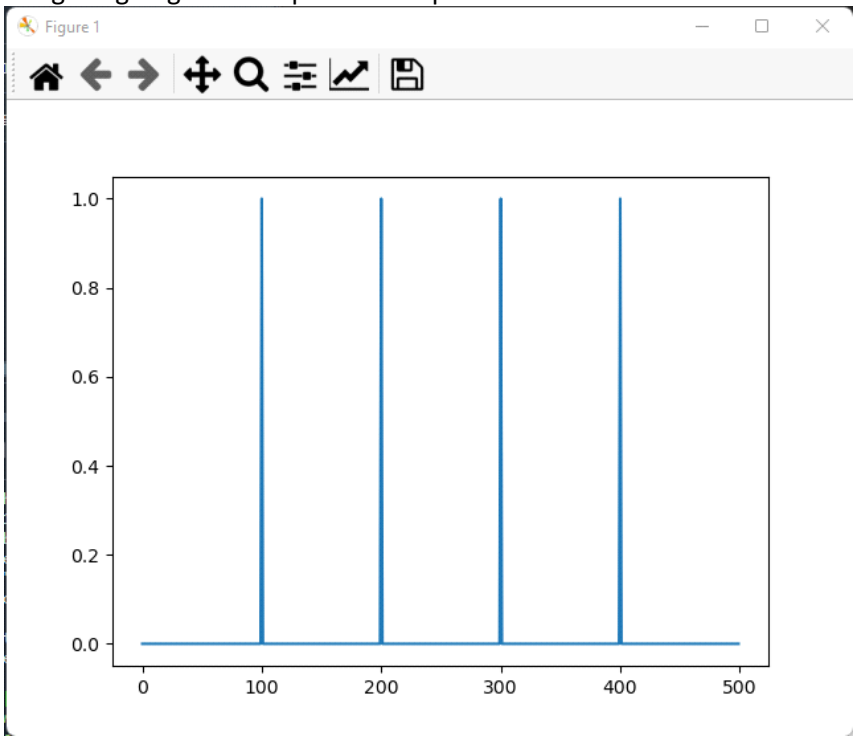
The impulse generator:

The strategy is just to generate a value of 1 every time I reach a value of SR/F0 samples which I will call the glottal pulse period. So I need a varaible for the amount of samples I have and I need to determine my fundamental frequency, then use a loop to go through a signal and generate 1s every n samples. Seems easy enough.

So this is what I came up with, the sampling rate is defined as 10000, and the fundamental frequency was just assumed to be 100Hz for now, that means that the glottal pulse period n is 10000/100 which is 100 samples. I assumed that the signal would be 500 samples long and it would be a generic 0dB amplitude signal. The pulse index tracks which pulse was generated last. It then does a for loop on the signal to generate 1s everywhere when it reaches 100 samples. So it is expected to be at 100, 200, 300, 400, 500. The output is shown below.

```
7   import numpy as np
8   import pylab as pl
9
10  #assume sample rate is 10000 and fundamental frequency is 100Hz
11  SR = 10000
12  F0 = 100
13  SAMPLES = 500
14  glottal_period = SR/F0
15  signal = np.zeros(SAMPLES)
16
17  #tracks where the last pulse was generated at
18  pulse_index = 0
19  for i in range(SAMPLES):
20      if i - pulse_index >= glottal_period:
21          signal[i] = 1
22          pulse_index = i
23
24  pl.plot(signal)
25
```

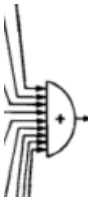
This is more or less what was expected. We don't really need that last pulse at the 500th sample because the voice will end. Well it seems that the impulse train is successfully generated. I already did some cascading and parallelizing but I want to make it a bit more generic in the code. The next thing I'm going to attempt is the amplifier and mixer.



Mixer, amplifier, buffer implementations

Wednesday, 31 August 2022 02:24

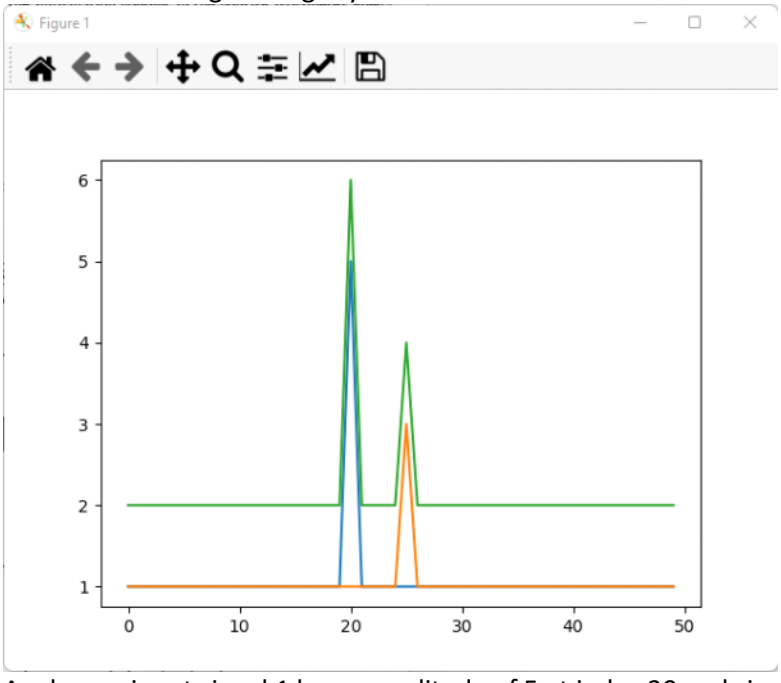
The **mixer** is indicated by the + sign in the Klatt block diagram shown below:



As seen in the image, it should later be generic, but the whole idea is that it should add signals. This is simple to implement in python because signals will just be lists. To add an entire list to another list, I use 'listname[:]', which indexes the entire list. The code below is the implementation:

```
#mixer
#the mixer is just the summing component in the synthesizer indicated with a + sign.
#so it simply adds signals and generates a single output.

length = 50
#signal 1:
inp1 = np.ones(length)
inp1[20] = 5
#signal 2:
inp2 = np.ones(length)
inp2[25] = 3
#if these two are added, I'm expecting a 2 signal
output = np.zeros(length)
output[:]=inp1[:]+inp2[:]


pl.plot(inp1)
pl.plot(inp2)
pl.plot(output)
```

I made the two signals slightly different so that it is easier to see the result:



As shown, input signal 1 has a amplitude of 5 at index 20 and signal 2 has a amplitude of 3 at index 25. When these two signals are added, the resulting output signal looks correct.

Buffer:

Buffers are indicated by the lines in the synthesizer that connect 2 components. The buffer will just set the next component's input to the previous component's output. So it simply acts as a wire. I'm

not going to show the implementation of it here, it will be done in a more generic way for the system.

Amplifier:

Next is the amplification control, which is used mainly for the parallel synthesis vocal tract, and the controls for the voicing source. The amplitude will be specified in decibels (dB) and this value will have to be converted to an adequate value to add to the signals. The formula is

$$A(dB) = 10\log_{10}(A)$$

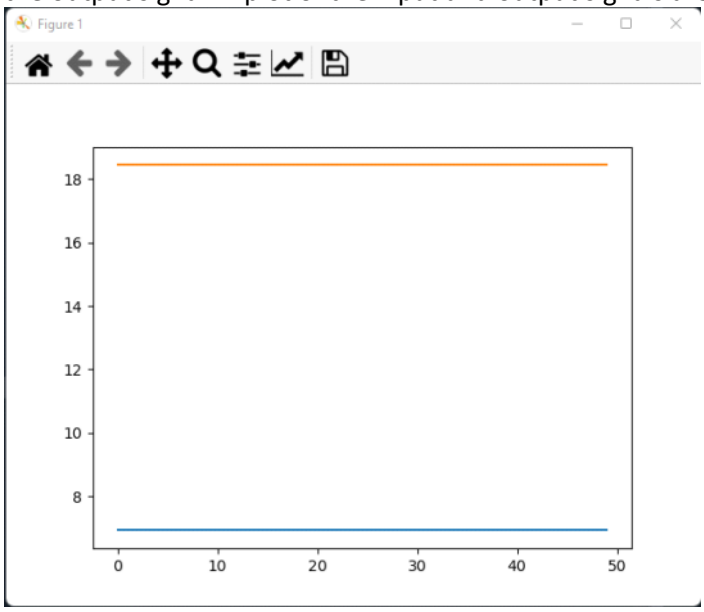
So if the amplitude to be added is specified in dB, then to get A for a signal would simply be the inverse of log which is:

$$A = \sqrt{10^{A(dB)/10}}$$

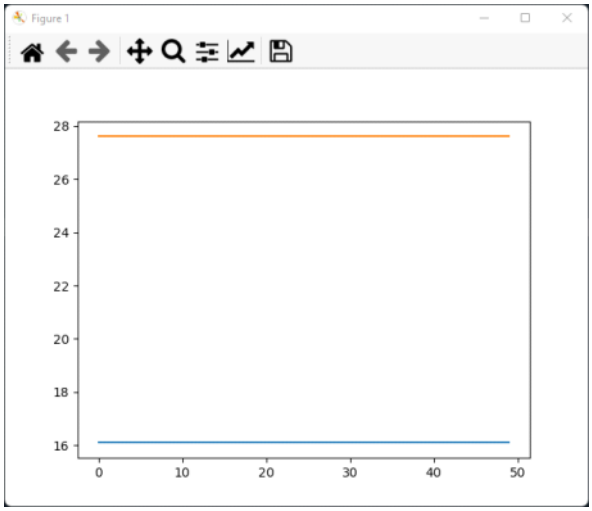
Then to add the amplification to the signal, this value is simply multiplied with the signal list. Let me try implementing this and see the result.

```
11  #amplifier
12  #specify amplification in dB
13  db = 10
14
15  input = np.ones(50)*2
16  output = np.zeros(50)
17
18  #amplifier does a conversion to signal value
19  db = np.sqrt(10)**(db/10)
20
21  output[:] = input[:]*db
22
23  pl.plot(10*np.log(input))
24  pl.plot(10*np.log(output))
25
```

As seen in the code snippet, the db increase should be 10db. The input voltage signal is around 2V is magnitude. So the db is converted to a voltage and then multiplied with the input signal to produce the output signal. A plot of the input and output signals are shown:



In this figure we can see that the input signal is close to about 8dB, and the output signal has been amplified to around 18dB which corresponds to the 10dB increase. So amplitude can successfully be manipulated. I just want to show one more result here to verify this.



I change the input signal to 5V, as seen in this figure amplification goes from 16dB to around 27.5dB. There is a larger discrepancy with larger voltage signals. The signals mainly used will be in the 50dB to 60dB range, for voicing, so there should be a 3 or 4 dB difference if a larger change in amplification is required. It should not be too much of a problem though.

First diff, low-pass, normalizer implementation

Wednesday, 31 August 2022 15:39

Ok, so I have tested and verified the other components so far. I'm generalizing them while working on other components as I make the entire system similar to the repository I found on git. The next component is the first difference component. In the diagram it is simply indicated with the symbol that has first diff on it.

There is one of these as input to the parallel vocal tract and on the output by the radiation characteristic.

1960). The transformation is simulated in the synthesizer by taking the first difference of lip–nose volume velocity:

$$p(nT) = u(nT) - u(nT - T)\,. \qquad (7)$$

The paper shows that the first difference characteristic is simply the signal minus the signal at a previous time step. This above indicates the FD for the radiation output.

The implementation will simply reflect this equation.

The low-pass filter will be implemented as a -6dB/octave low-pass filter according to the document.

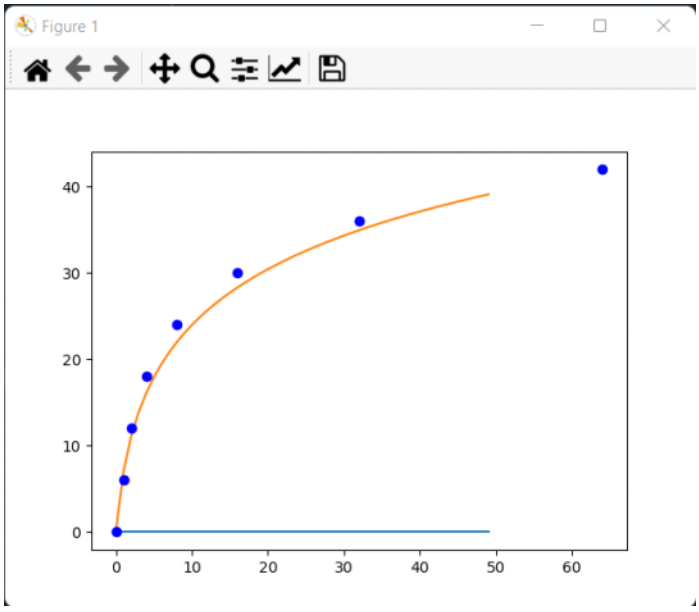
D. Frication source

A turbulent noise source is simulated in the synthesizer by a pseudo-random number generator, a modulator, an amplitude control AF, and a –6 dB/octave low-pass digital filter LPF, as shown previously in Fig. 6. The spectrum of the frication source should be

They don't specifically say how it should be implemented but as long as the output signal reflects a 6db/oct, it should be ok. I'll find out the reasoning why later.

```
 8   import numpy as np
 9   import pylab as pl
10   import matplotlib.pyplot as plt
11
12   #lowpass filter as a one-zero 6db/oct filter
13
14   inp = np.ones(50)
15   out = np.zeros(50)
16   out[0] = inp[0]
17
18   for i in range(1,50):
19       out[i] = inp[i]+out[i-1]
20
21   pl.plot(10*np.log(inp))
22   pl.plot(10*np.log(out))
23   pl.plot(0,0,'bo')
24   pl.plot(1,6,'bo')
25   pl.plot(2,12,'bo')
26   pl.plot(4,18,'bo')
27   pl.plot(8,24,'bo')
28   pl.plot(16,30,'bo')
29   pl.plot(32,36,'bo')
30   pl.plot(64,42,'bo')
```

As seen, the every time there's an octave (it doubles in frequency) there is a 6dB change. As seen by the plot points.



That was straightforward. Next is the normalizer.

Normalizer:

The normalizer is simply taking the signal and dividing it by its absolute value so that the maximum it reaches is 1. I won't show this implementation here.

Noise generator and switch implementation

Wednesday, 31 August 2022 22:57

Noise generator:

The noise generator according to the document is a Gaussian distribution which is a normal distribution. I should just generate noise with a mean and standard deviation using the NumPy library. That is simple enough. I will use a mean of 0 and standard deviation of 1.

The switch:

It sends two outputs, one to the cascade tract and the other to the parallel tract. Depending on the switch value the other signal it sends will be a zero signal. There's not really much to say about this, only that it's an if statement.

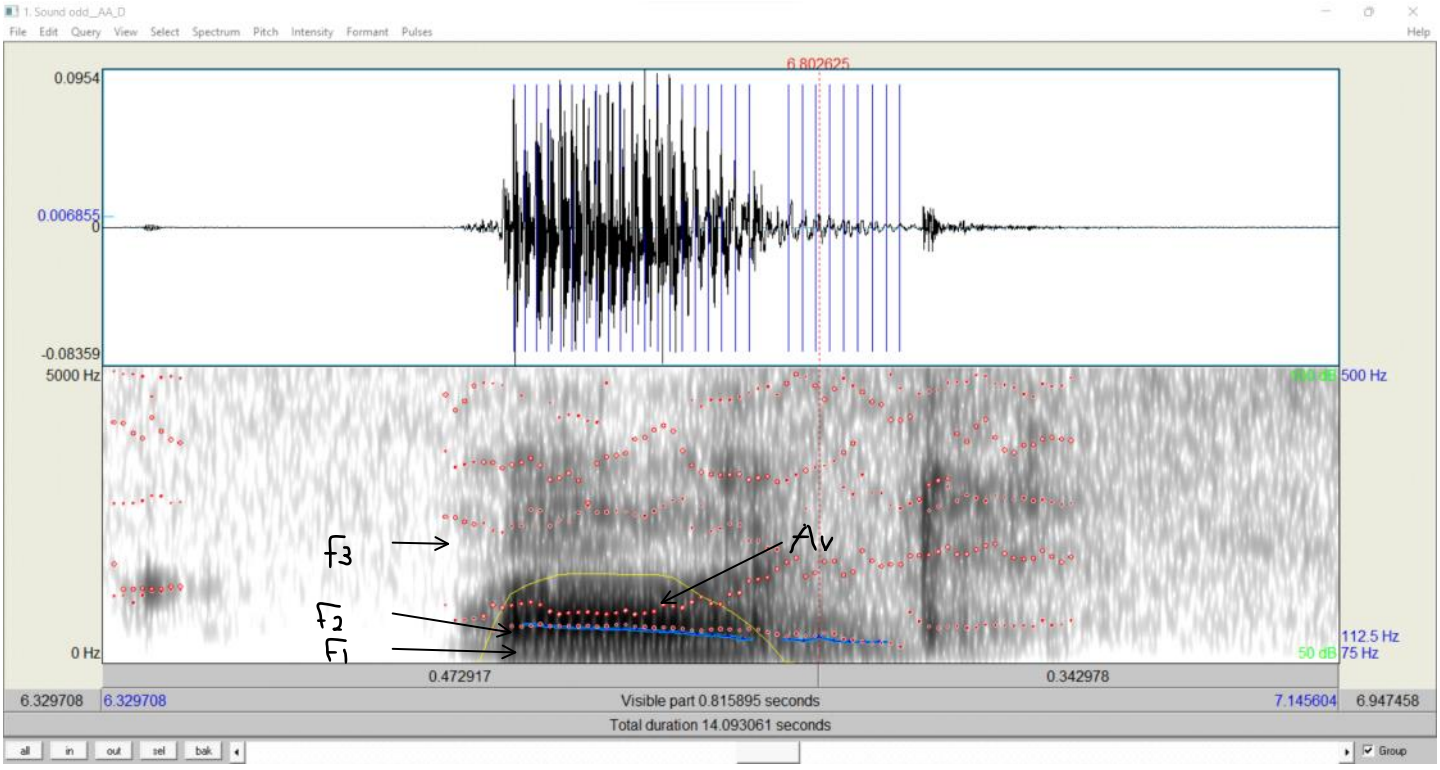
The next thing would be to implement the different sections by connecting all of these components. And then after that, connecting the sections into a single synthesizer.

Synthesis theory, Praat, audio recordings.

Monday, 05 September 2022 09:20

Ok, so since the last time that I documented anything regarding the synthesizer, I managed to get audio output from the synth esizer. The audio output with the specifications given in the document sound very robot like. It does however successfully synthesize vowel sounds. I have not yet tried consonants but I thought that for now I should add a little theory on what specifications are required in order to produce a sound that sounds similar to my own voice. There are actually 3 main parameters to look out for. The parameters are formant frequencies (with their bandwidths), the amplitude of voicing (AV), and the fundamental speaking frequency (F0). In addition to this it is very useful to know the kind of change these parameters take over time so that they can be adequately modelled into the synthesizer.

Firstly I'm going to present an image of a synthesized audio of myself saying odd.

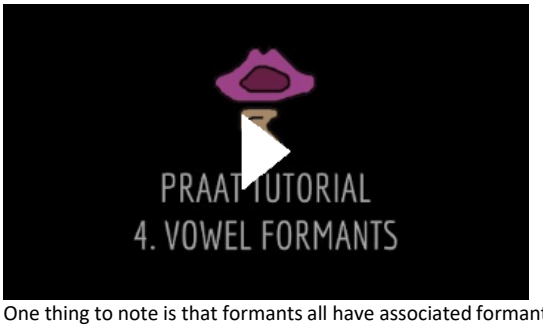


As seen in the image, I have broken down the word 'odd' into it's respective phoneme parts, i.e. AA D. This is how the word i s pronounced basically. The AA sound is the first 0.47 seconds and the D sound is the 0.34 seconds after that. Obviously the AA sound does not play over the entir e 0.47 second duration but it does play for about half of that. I'm going to discuss these parameters that make up the first phoneme sound now (the AA sound).

Formant frequencies:

The formant frequencies (indicated in red) are the frequencies of the peaks of the sound impulse. These frequencies make up t he overall sound of the output signal. Altering the formant frequencies will ultimately change the sound of what is being said as a matter of speaking. For instance if I change the first formant frequency to something else then it might make an IY sound instead of an AA sound. According to Klatt, the first 3 formants a re significant formants, in that, these are the formants that will affect human perception the most. It is therefore important to note down the first 3 formants. The re are 2 ways to identify formants. Firstly, if making use of Praat's features to analyse signals, then from the spectrogram (the lower part), it can be seen tha t the formants are already estimated. I tried to indicated where the first three formants are in the image (F1,F2,F3). The first formant is where the blue line is (i t is not the blue line), as seen it almost forms a straight line over time, but has a slight decline. The second formant is more messy, and it is above the first forman t. It appears to be a straight line for some time and then towards the end, the formant starts increasing in frequency (starts going up). The third formant is the ne xt dark patch above the second formant. It seems to vary between increasing and decreasing frequency over the duration of the vowel. The fourth and fifth fo rmants are above the third formant and can be identified by looking at either the dark patches or the red dots in the spectrogram. The second way to identify a formant is by zooming into the impulse response of the sound itself (the top part), however this is more difficult. This link contains a very good explanation of ho w to measure formants in Praat.

Praat Tutorial: 4. Vowel Formant Measurements



One thing to note is that formants all have associated formant bandwidths. And it can be thought of as the area of the dark s haded areas around the formant means. This value for each formant also has to be adequately tracked.

Amplitude of Voicing AV:

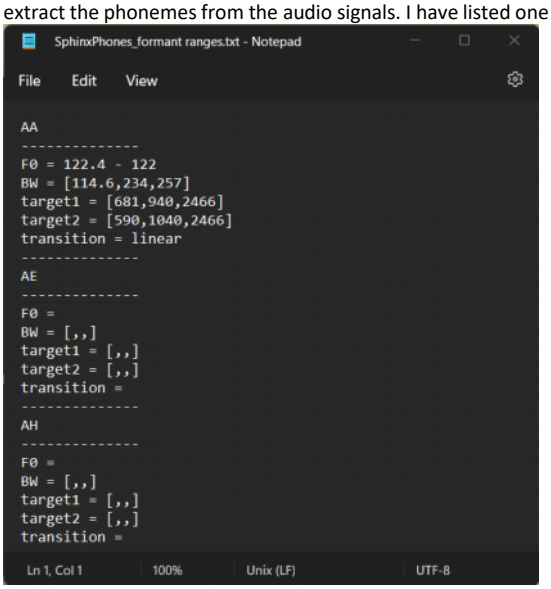
The amplitude of voicing is to put in simple terms, the intensity or loudness with which the speaker is speaking. So for my c ase, in the image, it is indicated by the yellow line and measured in decibels (dB). This parameter is important as it also contributes to how a speaker sounds. For in stance if I generally have a deeper voice, then the amplitude of voicing might be higher than that of a female voice. It is also important to know the shape of t he amplitude of voicing over time. In this image, It is seen that I have more intensity on the AA sound, and almost no intensity for the D sound.

Fundamental frequency (F0):

This is the natural frequency of my vocal tract. It is the rate that one could say that the impulses are generated from my vo ice. This is indicated on the blue line where the first formant is (in this case), previously mentioned. A more generalized way of describing fundamental frequency, is to describe it as the pitch of your voice. The fundamental frequency of the speaker is usually in the lower frequency region. Like 100 Hz area. Females usually h ave higher fundamental frequencies than males. This parameter is important as this is distinguishable between people, as shown in the project proposal/ research conducted for specifications.

Other considerations:

Just in general to sum up what I have done after simulating the general specifications in Klatt's document. I made several re cordings of myself saying words to extract the phonemes from the audio signals. I have listed one of them so far in a text file like this



As seen for this particular sound file, the first formant starts at 681Hz, and ends at 590Hz. The second one starts at 940 an d ends at 1040, and the third at 2466, with a lot of variance. The fundamental frequency of my voice is about 122.4 to 122Hz. The bandwidths are also specified and the transition appears to be linear transitions mostly. The output audio that I obtained sounded very much like the AA sound. The only issue is that this sound s eems a bit short.

Where to go from here:

So now that I can generate audio from parameters in Praat. I just need to generalize the audio generation sequence, so that w hen it detects a certain vowel, it assigns the correct parameters and duration. I also need to attempt to concatenate audio files so that I can hear what a word sounds like. This should take me about a week to complete. I can then start with the natural language processing and linguistics analysis. I looked a bit into that and so far It seems that I require a lexicon that has the words, parts of speech, and the pronunciation of the word. That's the main thing that I need to proceed. Data collection is still out of the question. That would take up too much time which I do not have much of. I have to work on my research project practical 2 alo ngside this, so I will try to complete my practical early this week and then continue working on project, hopefully by Thursday or Friday. So far I am on schedule.

Email client / Synthesizer code flow and UML class block diagram

Monday, 05 September 2022 10:38

I think it is important to know the flow of my code and all the functions that is involved with it, for this I am presenting a code flow diagram and I will maybe insert a class UML block diagram or something for the synthesizer and email client. Which are my only implementations. If I have time to do it this week then I will put them in here.

Testing Klatt parameters

Monday, 10 October 2022 10:04

Ok, so I ran into a bit of a problem with consonant sounds, however, I can still fix the issue. And the way to do that is to compare the parameters of the Klatt synthesizer with the output waveform to see what happens with each one, without further ado, I'm going to do tests on the AA in odd, sound. Let's go over parameters one at a time.

Ok, these are all the parameters:

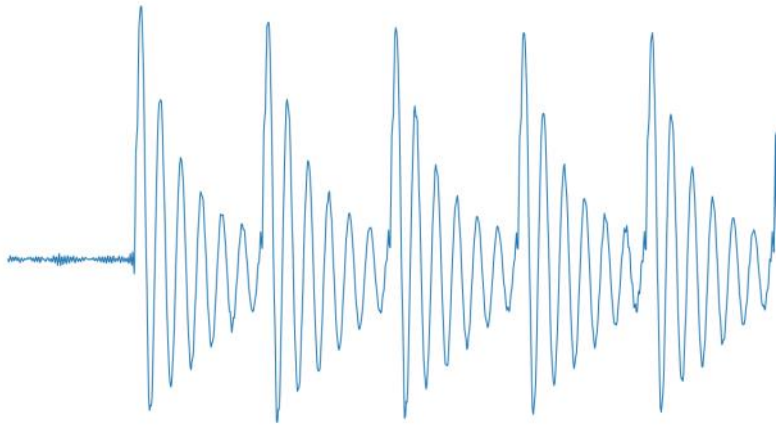
```
                A1=0, A2=0, A3=0, A4=0, A5=0,
 self.FS = FS
 self.DUR = DUR
 self.FORMANTS = FORMANTS
 self.NUMSAMPLES = round(FS*DUR)
 self.DT = 1/FS
 self.F0 = np.ones(self.NUMSAMPLES)*F0
 self.FF = [np.ones(self.NUMSAMPLES)*FF[i] fo
 self.BW = [np.ones(self.NUMSAMPLES)*BW[i] fo
 self.AV = np.ones(self.NUMSAMPLES)*AV
 self.AVS = np.ones(self.NUMSAMPLES)*AVS
 self.AH = np.ones(self.NUMSAMPLES)*AH
 self.AF = np.ones(self.NUMSAMPLES)*AF
 self.FNZ = np.ones(self.NUMSAMPLES)*FNZ
 self.SW = np.ones(self.NUMSAMPLES)*SW
 self.FGP = np.ones(self.NUMSAMPLES)*FGP
 self.BGP = np.ones(self.NUMSAMPLES)*BGP
 self.FGZ = np.ones(self.NUMSAMPLES)*FGZ
 self.BGZ = np.ones(self.NUMSAMPLES)*BGZ
 self.FNP = np.ones(self.NUMSAMPLES)*FNP
 self.BNP = np.ones(self.NUMSAMPLES)*BNP
 self.BNZ = np.ones(self.NUMSAMPLES)*BNZ
 self.BGS = np.ones(self.NUMSAMPLES)*BGS
 self.A1 = np.ones(self.NUMSAMPLES)*A1
 self.A2 = np.ones(self.NUMSAMPLES)*A2
 self.A3 = np.ones(self.NUMSAMPLES)*A3
 self.A4 = np.ones(self.NUMSAMPLES)*A4
 self.A5 = np.ones(self.NUMSAMPLES)*A5
 self.A6 = np.ones(self.NUMSAMPLES)*A6
 self.AN = np.ones(self.NUMSAMPLES)*AN
 self.AB = np.ones(self.NUMSAMPLES)*AB
```

I wont be touching FS, that's the sample frequency, DUR, which is duration (the length of the audio), FORMANTS, which is the amount of formants, NUMSAMPLES, which is a constant depending on sample frequency and duration, and DT is period which is a given, so I'll start at F0 and go down. If I do consonants then I'll test the formant amplitudes maybe. I'll see how it goes.

F0:
0 = sounds like noise
F0 anything, affects the fastness of the signal, with lower F the signal is spread out, but with higher F0 the signal sounds faster and squished together. So basically just keep this fundamental frequency slightly varying so that it doesn't sound like a robot.

FF: formant frequencies, should affect formants but how?
This is what it looks like with formants

```
(duration=0.5,F0beg=100,F0end=100,f1beg=625,f1end=610,\
 f2beg=920,f2end=1100,f3beg=2499,f3end=2666,linear=True,
 bw1=130,bw2=248,bw3=451,Abeg=60,Aend=None)
```



The formants dictate the amount of amplitudes seen and the average shape of the wave. So it is the reason why it looks like this. The only way to discern the individual formants from this is by looking into spectrograms and by looking into mathematical equations that involve trig identities to add waves that have different frequencies. BW is similar to this, its best observed in **a spectrogram of frequency analysis.**

AV:

So there is a set noise floor, and by adjusting AV higher, the amount of noise heard in the output gets reduced a lot. It basically increases amplitude of a signal. The lower the amplitude, the more the noise can be heard because the overall signal is closer to the noise floor.

AH:

The amplitude you are adding to the noise floor,
This kinda wasting time, Ima go back to work.