

iFPGA - Intermittent Intelligent FPGA Platform

Design Document

sdmay20-38

Client: Henry Duwe

Advisers: Henry Duwe

Team Members/Roles

Justin Sung - Embedded Systems Engineer

Zixuan Guo - Systems Diagram Expert

Jake Meiss - Electrical Engineer

Andrew Vogler - FPGA Design Engineer

Jake Tener - Software Technician

Team Email: sdmay20-38@iastate.edu

Team Website: <http://sdmay20-38.sd.ece.iastate.edu>

Executive Summary

Engineering Standards and Design Practices

Hardware and software we will use in this project:

- Powercast P2110b RF Energy Harvester
 - RF to DC Converter
 - Boost Converter
 - Voltage Monitor
- FPGA Circuit Board
 - MicroSemi's Igloo nano AGLN250V2
- Microphone
 - MEMS microphone
- Capacitor
 - Electrostatic double-layer capacitor
- Regulators
 - Boost and Buck converters to manage voltage to the load
- Neural Network
 - Tensorflow Lite Model with 3 fully connected layers
- Microcontroller
 - TI's MSP430FR5994

Engineering Standards we are applying to our project from the IEEE Code of Ethics:

- Honesty about the functionality and usefulness (#'s 3 & 6)
 - Intellectual integrity for previous work is necessary for eventual published research on the platform
- Emphasis on Teamwork (#'s 7, 8, & 9)
- To make the highest quality product within our capability (#'s 5 & 6)

Summary of Requirements

- Design batteryless PCB-based FPGA system
- FPGA performing computation on low power
- Design application that can be accelerated onto the FPGA
- Accurate neural network predictions
- Data exportable by UART
- Ability to checkpoint progress in a program
 - Intermittent execution on an FPGA platform with frequent power cycling

Applicable Courses from Iowa State University Curriculum

- CPRE488
- CPRE381

- EE330
- CPRE281
- CPRE288

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project:

- Using Libero (an FPGA design tool)
- How to do independent research
- IP-Block research
- Machine Learning
- Python
- C++
- Neural Networks
- EAGLE
- PCB Layout Design
- Sound Parsing and Analysis

Table of Contents

1 Introduction	6
Acknowledgement	6
Problem and Project Statement	6
Operational Environment	6
Requirements	6
Intended Users and Uses	7
Assumptions and Limitations	7
Expected End Product and Deliverables	7
2. Specifications and Analysis	7
Proposed Design	7
Design Analysis	13
Development Process	13
Design Plan	13
3. Statement of Work	14
3.1 Previous Work And Literature	14
3.2 Technology Considerations	14
3.3 Task Decomposition	15
3.4 Possible Risks And Risk Management	15
3.5 Project Proposed Milestones and Evaluation Criteria	15
3.6 Project Tracking Procedures	16
3.7 Expected Results and Validation	16
4. Project Timeline, Estimated Resources, and Challenges	16
4.1 Project Timeline	16
4.2 Feasibility Assessment	18
4.3 Personnel Effort Requirements	18
4.4 Other Resource Requirements	20
4.5 Financial Requirements	20
5. Testing, and Implementation	20

Interface Specifications	20
Hardware and software	20
Functional/Non-Functional Testing	21
Process	23
6. Results	24
6.1 Hardware	24
6.2 Software	31
6.3 FPGA and MCU System	35
7. Closing Material	37
7.1 Conclusion	37
semester 1	37
semester 2	38
7.2 References	38
7.3 Appendices	38
I: Operation Manual	38
II: Alternative / Initial Versions of the Design	49
III: Other considerations	50
IV: Code (optional)	50
V: References	51

List of figures/tables/symbols/definitions

2.1 Top level Diagram	9
2.1 Embedded System Architecture Diagram	10
2.1 Software Design Diagram	10
2.2 Software Acceleration Diagram	11
5.4 Test Plan Flow Diagram	23
6.11 Power Cast Energy Harvester Schematic	25
6.12 Regulators Schematic	25
6.13 Schematic for the Master MSP430 Microcontroller	26
6.14 Schematic for the Microsemi Igloo Nano FPGA	26

6.15 Schematic of the slave MSP430 Microcontroller	27
6.16 Schematic of the Capacitor Bank	27
6.17 Schematic of switches, reset, and oscillator control circuits	28
6.18 Schematic of Header	28
6.19 EAGLE Printed Circuit Board Layout Design	29
6.20 Final Fabricated Printed Circuit Board	30

1 Introduction

1.1 ACKNOWLEDGEMENT

Henry Duwe - Advised and set concrete goals for the team to work towards.

Narayanan Vishak - Assisting in the hardware design development.

Sahu Rohit - Assisting in the software design and development.

1.2 PROBLEM AND PROJECT STATEMENT

Develop a hardware and software solution to intermittently execute a program targeted on a low power FPGA platform that can withstand multiple power cycling events.

As IoT applications become more prolific and integrated into society, the demand for more efficient and powerful devices grow. Addressing these extreme resource requirements and computational demand results in stress on the power supply. Batteries are the predominant source of power for IoT devices. However, batteries are unsustainable and require maintenance and replacement relatively often in the life span of devices. To confront these inadequate power sources, Self-harvesting power technology shows promise. They require little to no human intervention upon deployment and have significantly longer lifespans compared to batteries. Self-harvesting device can

The iFPGA is a low power designed FPGA platform powered completely by the powercast harvester device to intermittently execute an audio recognition program with resilience against frequent power cycling events. If successful, the iFPGA prototype will uncover novel design solutions to address unreliable power sources, and broaden the feasible areas where IoT can be applied. The prototype will lead to more advanced designs that will build and improve upon any shortcomings of the final prototype. For our research, the iFPGA was targeted at performing audio recognition and classification.

1.3 OPERATIONAL ENVIRONMENT

The iFPGA will be used within the lab environment, so environmental considerations were low priority.

1.4 REQUIREMENTS

Functional Requirements

- Batteryless
 - Power provided by means of RF Energy Harvesting
- Data transmission off-chip
 - UART
- Program execution that can pause and continue while power toggling
 - Checkpointing in the software

Non-Functional Requirements

- Low Power
 - Must be able to detect, perform computations, and transmit data at low power
- Little to no human intervention after deployment

1.5 INTENDED USERS AND USES

The product is designed for analyzing audio based data on an FPGA with an emphasis on low power design. It can be applicable to areas such as IoT research and a prototype for developers to expand the study and project further.

1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions

- This project will be used by Dr. Duwe and his research team.
- The end result of the project is meant to be a prototype design for research, not a marketable project.

Limitations

- Cost of prototype shall not exceed 200 USD
- Battery-less

1.7 EXPECTED END PRODUCT AND DELIVERABLES

- 1) A PCB prototype that can accelerate a portion of the audio recognition pipeline that demonstrates the ability to complete the program successfully with frequent power cycling events throughout its execution. It will be powered by a radio frequency harvester feeding into a capacitor bank. A demo to show these capabilities suffice in a successful end product.
- 2) Comprehensive documentation describing and explaining parts and how to interface with the platform. Justification of our design choices will be included with the end product prototype.

2. Specifications and Analysis

2.1 PROPOSED DESIGN

Our target solution is an FPGA powered by RF harvesting that can perform simple computations within a low power cycle. We will be using the Powercast RF energy harvester which will provide enough power to run the FPGA and the MCU (MSP430). The first MCU will handle MFCC generation and hold the digital audio data to be analyzed. The FPGA, Microsemi's Igloo Nano, will be able to operate on low power in order for this to be possible. Our FPGA will perform part of our software computation: matrix multiplication; and speed it accelerating inference on the FPGA. The FPGA will receive the scaled MFCC coefficients, matrix multiply them against the fixed weights determined by the neural network, and the intermediate data produced will be sent to the second MCU for storage and assembly of the prediction vector.

Power Management:

- In order to fulfill the requirement for our project to be batteryless, we must begin with energy harvesting. We will be using a 3-watt, 915 MHz wifi transmitter that will be providing the power to our system. This radio frequency signal will be received by a directional patch antenna and then converted to DC power by a Powercast P2110B RF energy harvester. At this point, the converted DC power will be stored in the capacitor. The capacitor node will be connected to both a boost converter and a voltage monitor. The voltage monitor will supervise the voltage at the capacitor node, checking for an upper threshold of 1.25V and a lower threshold of 1.02V. The capacitor will begin charging up until the upper voltage threshold is reached, triggering the voltage to be amplified by a boost converter which in turn will provide power to the MSP430 as well as 2 discrete voltage regulators. The MSP430 will control the two regulators that will then provide power to the Igloo Nano FPGA when computations are necessary. The capacitor will be discharged as it provides power to the load (Microcontroller and FPGA) until the capacitor voltage hits the lower threshold of 1.02V in which power to the load is lost and the storage device begins to recharge. Since this process will be continuous, power to the load will be intermittent which must be taken into account in the remainder of the design.

FPGA/Microcontroller Design:

- When powered on by the MSP430, the Igloo Nano will boot and begin its computation. The MSP430 receives digital audio data and performs MFCC generation. Once the final scaled MFCC has been generated, it passes it through the neural network, which is a series of matrix multiplications between the scaled MFCC and the fixed weights of each layer. These matrix multiplications are what the FPGA accelerates. The MSP430 sends the MFCC data to the FPGA, where it performs a pipelined matrix multiplication via a multiply-accumulate hardware design with the fixed weights pre loaded into its non-volatile memory, in which it sends the intermediate data to the second MSP430 for storage. Once all of the matrix multiplication operations have concluded, the FPGA passes the final prediction vector to the first MSP430 to be transmitted out of the system. The second MSP430's role is to store all of the intermediate data produced by the FPGA and the weights from the neural network.
- The choice to use the MSP430 as our microcontroller was from a suggestion from Dr. Duwe. Although we could have searched for others, this was his recommendation given his experience in the area and so we thought that to be the best microcontroller for the job. Our FPGA, the Igloo Nano, was chosen after looking at the different products between Lattice and Microsemi. We first came to the conclusion to use a Microsemi product because we verified that we could use Libero, an FPGA design tool, for free. We also found that Lattice was more vague on the product descriptions, making Microsemi a better company for the job. We then narrowed down to the Nano because it is low power and because it still had the most computation power given it's low power.

Software:

- Since our project entails audio classification, we developed a neural network for our device to use in inference with each new sound. In order to do this, we used UrbanSound 8k's dataset of 8,732 sounds with 10 unique classifications. Then we sample them into a digital time series and generate spectrograms of each sound (an array of floats indicating its frequency, amplitude, and change in time). With this data, we are able to train a model to know what type of sound correlates to a given label.
- Upon receiving a new sound, our software will sample the sound, then generate its spectrogram, and then use this in inference with our model's weights in order to predict the class of the sound (Fig 2.3). This inference is then going to be transmitted off of the device to a target system.
- Software has been developed in Python for modular testing before integration, then replicated in C++ for acceleration onto hardware, this brought its own challenges that will be discussed later in this document.
- When accelerating and uploading the software to the devices, the sound analysis (sampling and MFCC generation) will be loaded onto one MSP430, and the trained model onto the other. The Igloo Nano will be passed weights of the model from one MSP430, and the calculated MFCC coefficients from the other, and perform the matrix multiplication and build the output vector (the success of this implementation has been affected by Covid-19.)

Output:

- The Igloo Nano and MSP430 will conduct this software on a given segment of sound, then transmit a resultant string including the sound's classification, and a date time object of when the sound was heard.

Top Level Diagram

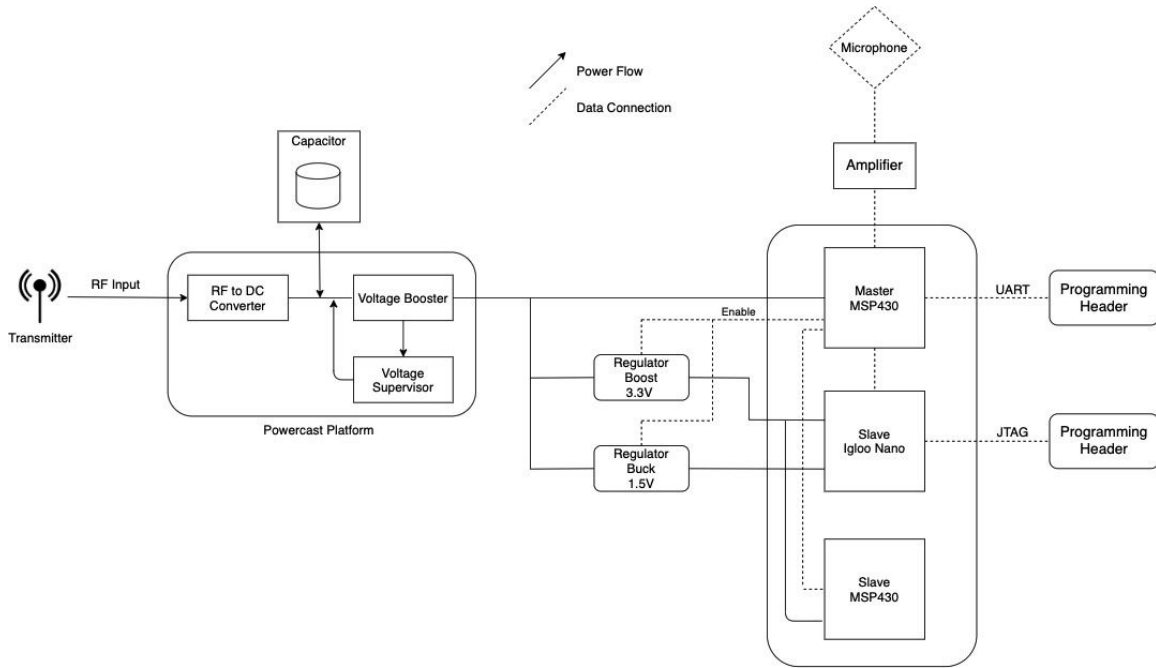


Figure 2.1 Top level design diagram

Embedded System Architecture Diagram

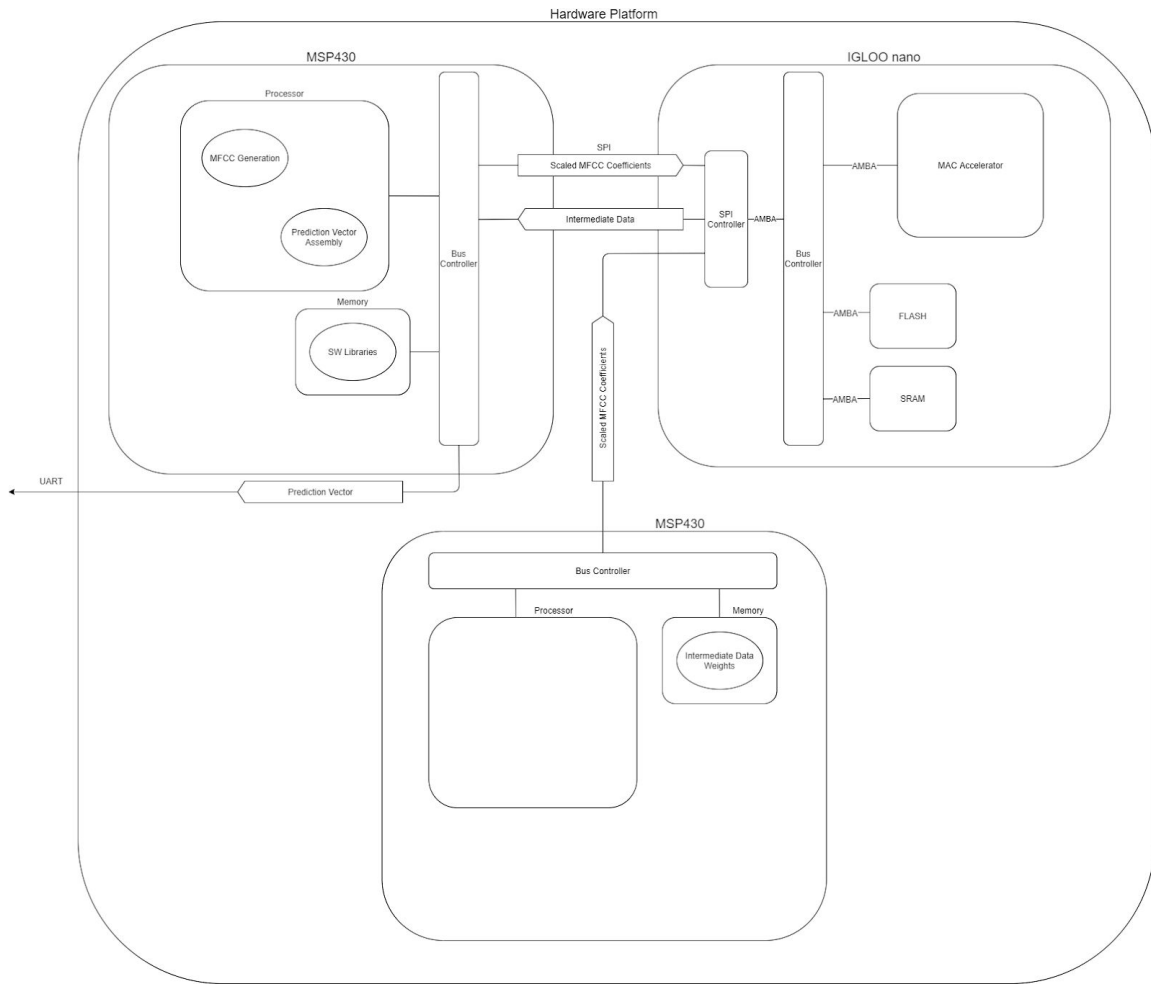


Figure 2.2 Embedded System Architecture design diagram

Software Process (PC Model Testing)

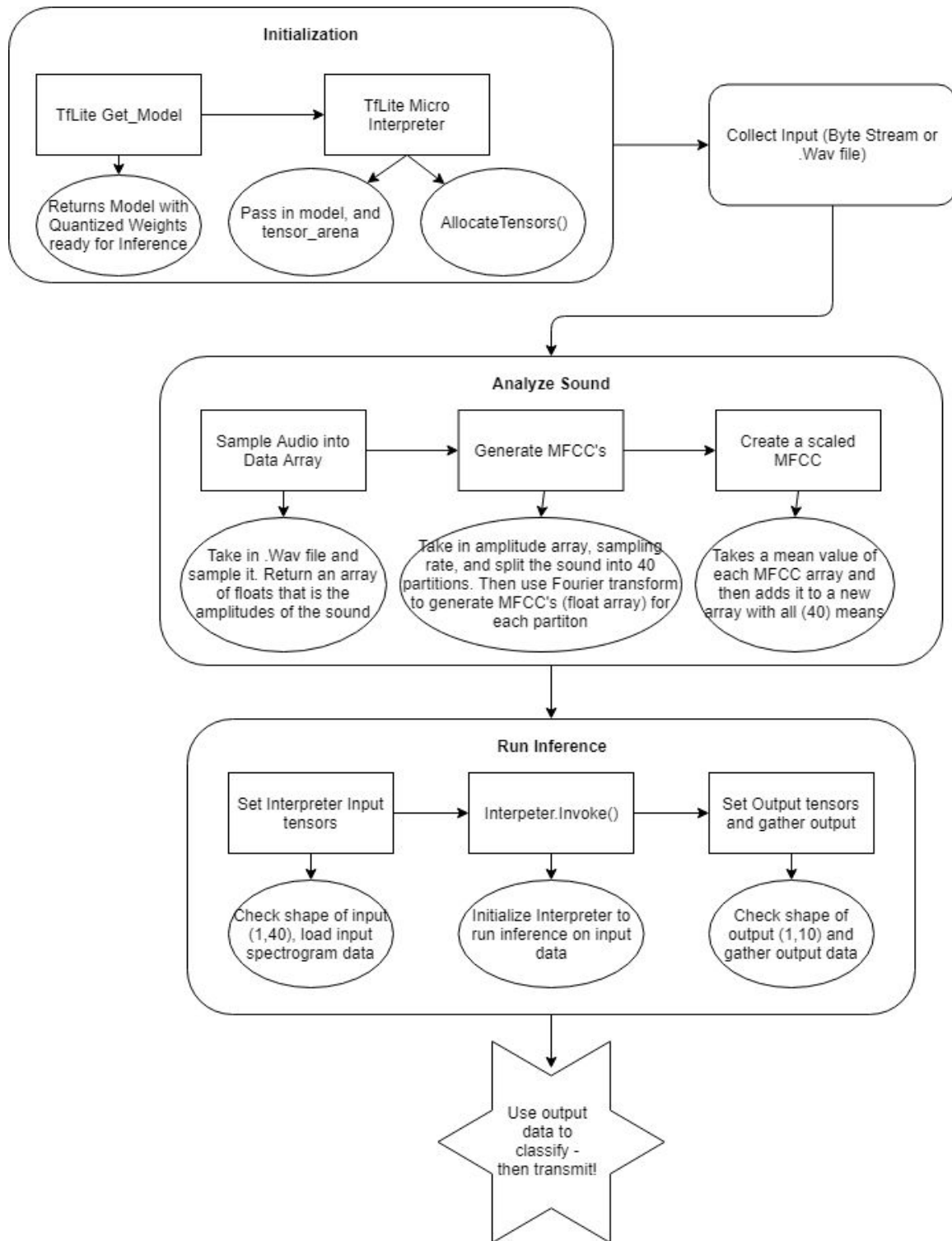


Figure 2.3 Software design diagram

Software Acceleration Process

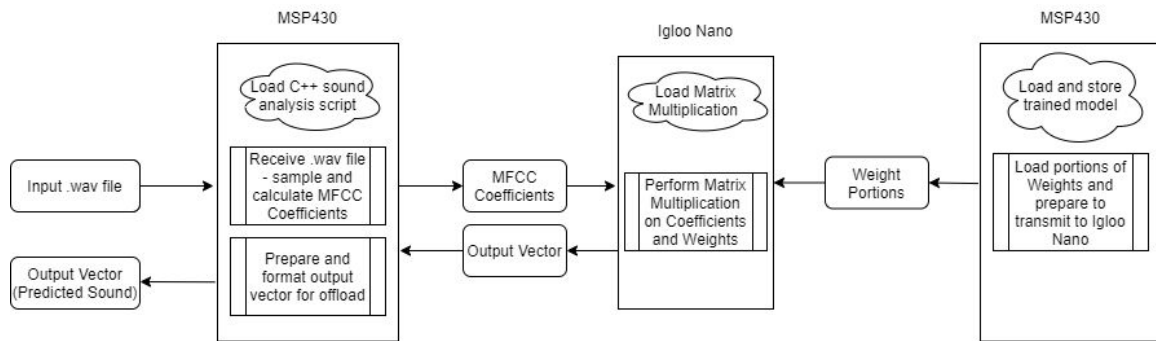


Figure 2.4 Software acceleration diagram

2.2 DESIGN ANALYSIS

Strengths

- Performing computations on an indirect powered platform
- Experience with Test Plans

Weaknesses

- Possibility of insufficient energy harvested resulting in an inability to provide power to the load
- Low power design provides limitations to computational intensity and memory capabilities
- Lack of data security on the platform
- Hardware specifically targeted to perform acceleration on one applications, not flexible for different applications

2.3 DEVELOPMENT PROCESS

We are using a waterfall-agile mix for our workflow. Since many of the tasks in the early phase we are walking into with a lot of unknowns, we have to go back and forth with decisions as we learn more about FPGA design. We carry out a new set of tasks each week. Some of the smaller decisions don't have a specific due date (hence the waterfall method) but the bigger decisions, such as choosing an FPGA have a more concrete deadline (more of the agile mindset).

2.4 DESIGN PLAN

Once the technology and materials have been secured, we plan to employ a design process similar to integration by parts and modular design. For both the hardware and software, the first step will be to create and design the modular parts of the project (e.g. read-and-write functionality module, hardware accelerator module, FPGA-to-microcontroller interface module, etc.) and verify that they work as intended through rigorous testing. Once each module has been created and verified for functionality, the next step would be to integrate these modules together and verify functionality of the combined modules. Repeating this integration and verification until all modules are combined into a single, fully integrated design.

Modular design will streamline the progress made since errors will be corrected upon discovery rather than allowing them to propagate further into the design and complicating the debugging process. Integration by parts will also facilitate this streamlined progress since logical errors will be caught upon testing the modules and ensures that errors will be corrected at the source and not be allowed to propagate downstream.

3. Statement of Work

3.1 PREVIOUS WORK AND LITERATURE

Embedded System for Acquisition and Enhancement of Audio Signals Paper

This paper describes previous research into an FPGA-based accelerator targeted at audio capture and filtering. It describes the embedded system architecture, signal processing steps and signal enhancement that can be accelerated using hardware. The system architecture was helpful in formulating the eventual hardware diagram. The link to the paper is provided in the appendix.

Efficient Processing of Deep Neural Networks: A Tutorial and Survey Paper

This paper went into the details of current efforts and strategies in accelerating neural network computations on FPGAs and resources-exhaustible hardware platforms. It states that the portion with the highest potential to be hardware accelerated in a neural network pipeline is the multiply-and-accumulate (MAC) operations. These operations have also been shown to result in significant reduction in energy consumption.

Market survey comparisons of components for the platform explained in the previous sections in detail.

Self-producing energy devices exist, but none that try to achieve something as costly, both in power and computational power, such as what the iFPGA proposes. No other device in its class can sustain itself solely by radio-frequency derived power while attempting costly computations.

3.2 TECHNOLOGY CONSIDERATIONS

Low power and budget are the two main factors that guided the technology choices made. Since the end product is a prototype to assess the feasibility of such a platform, the budget was restricted and considerations were made in light of it. The low power constraint also affected the technology choices.

3.3 TASK DECOMPOSITION

- Decide on components that fit the project constraints and functional/non-functional requirements.
 - Perform market survey with limitation considerations.
- Design the hardware platform.
 - Research into existing hardware architecture targeted at audio acceleration.
 - Design and verify portions of the completed hardware architecture and integrate together until the full system has been created and verified.
- Design the software to handle the audio pipeline acceleration and intermittent programming.
 - Design and develop the software to be integrated with the hardware platform.
 - Modify the software to handle intermittent programming.
- Integrate (2) and (3).
- Perform system-level test plan

3.4 POSSIBLE RISKS AND RISK MANAGEMENT

- FPGA low power boot sequence
 - Given the nature of low power applications, we still need to make sure the RF harvester is feeding the FPGA enough power to boot on the FPGA and run computations without shutting down mid-computation.
- FPGA maximum power consumption
 - When the FPGA is boosting, there will be an inrush current that goes through each component on the FPGA, this step may consume a lot of power in the capacitor. We need to do more research on the calculation on capacitor and consumption.
- FPGA-Microcontroller link
 - There is very little documentation about how to send data back and forth between FPGAs and Microcontrollers. We are going to have very little help when figuring out how to get the data through even though the data being transferred is simple.
- The Data-pipeline on FPGA and I/O resources
 - Data will pass through the data-pipeline on the FPGA needs to assign each I/O and memory resources, we need to make sure we have a good management on using these resources and optimize the data pipeline or its structure.

3.5 PROJECT PROPOSED MILESTONES AND EVALUATION CRITERIA

Key Milestones:

- Component finalization list
- FPGA component integration
- Power analysis and component integration
- Intermittent programming design
- Completed hardware and software integration

3.6 PROJECT TRACKING PROCEDURES

Weekly group meetings, weekly advisor meetings, and weekly work period meetings.

3.7 EXPECTED RESULTS AND VALIDATION

Originally, our expected results and validation was to deploy the platform in the field and be able to: (1) capture audio from its environment, (2) perform computation on the data, and (3) transmit the results from the computation to an external device. Frequent demonstrations will guarantee that each milestone is functioning as expected and that progress is being made in the right direction and in the right way.

However, due to COVID-19, we were not able to meet these expectations. Instead, we readjusted our end goal: Proving that our system has the capabilities to perform the feats that we propose via simplified prototype designs. These simplified designs would entail: (1) a hardware accelerator design, (2) a functioning software pipeline, (3) PCB design, and (4) communication between the FPGA and the MCU.

4. Project Timeline, Estimated Resources, and Challenges

4.1 PROJECT TIMELINE

Note: Highlighted section was the plan until school went online

Task Name	Start Date	End Date
Semester 1		
All Groups		
Explore previous work	Aug 30th, 2019	Sept 6th, 2019
Embedded System		
Research and choose FPGA	Sept 6th, 2019	Sept 27th, 2019
Prototype FPGA Programs	Sept 27th, 2019	Oct 18th, 2019
IP-Block Research Analysis	Oct 18th, 2019	Nov 15th, 2019
Matrix Multiplication Research and Prototyping	Nov 15th, 2019	Dec 6th, 2019
Software		
Research Possible Applications	Sept 13th, 2019	Sept 27th, 2019

Research Neural Networks and Machine Learning	Sept 27th, 2019	Oct 3rd, 2019
Develop and Train UrbanSound 8k Neural Network (Keras Model)	Oct 3rd, 2019	Oct 17th, 2019
Develop Testing Script for Neural Network	Oct 17th, 2019	Oct 25th, 2019
Quantize UrbanSound 8k Keras Model for Acceleration (Converts to Tflite Model)	Oct 25th, 2019	Nov 8th, 2019
Develop Testing Script for Tflite Model	Nov 8th, 2019	Nov 15th, 2019
Upload Tflite Model onto MSP430 and Run Test Case Inference	Nov 15th, 2019	Current
Research Sound Analysis with Intention to Convert Python Libraries to C Code	Nov 29th, 2019	Current
Power		
Research of RF Energy Harvesting Platform	Sept 6th, 2019	Sept 27th, 2019
FPGA Power Consumption Analysis	Sept 16th, 2019	October 25th, 2019
Power Management Design	Oct 18th, 2019	Current
System Level Architecture Diagram	Nov 25th, 2019	Current
Other:		
Presentation & Design Doc Preparation	Nov 22nd, 2019	Dec 9th, 2019
Semester 2		
Integration		
PCB Design Completion and Order	Jan 13th, 2020	Feb 7th, 2020
Transfer Python Code to MSP430	Jan 13th, 2020	Jan 24th, 2020
Fourier Transformations (FPGA Software)	Jan 13th, 2020	Jan 24th, 2020
Sound Spectrogram Generation	Jan 24th, 2020	Feb 14th, 2020
Data Transfer	Jan 24th, 2020	Mar 13th, 2020
Testing		
Input .wav files for hardware/electrical testing	Feb 14th, 2020	Mar 13th, 2020

Refining		
Tweak hardware/electrical based on testing	Mar 13th, 2020	Apr 3rd, 2020
Tweak software based on testing	Apr 3rd, 2020	Apr 17th, 2020
Other		
Buffer Time / Documentation Updating	Apr 17th, 2020	Apr 27th, 2020
Final Project Presentation Preparation	Apr 27th, 2020	May 1st, 2020
New Plans after Spring Break		
Revise Plans	Mar 23rd, 2020	Mar 25th, 2020
Finish individual components / Figure out what can be integrated given circumstances	Mar 25th, 2020	Apr 8th, 2020
Make final documentation and presentation	Apr 8th, 2020	Apr 22nd, 2020
Revise documentation and presentation	Apr 22nd, 2020	Apr 30th, 2020

4.2 FEASIBILITY ASSESSMENT

Our realistic projection of the project will be a custom PCB with the Igloo nano and the MSP430. Given audio data, it will be able to generate a scaled spectrogram, pass it through the neural network to obtain the inference data, and be able to classify the audio data. The Igloo nano will become a hardware accelerator for the matrix multiplication used in generating the inference data.

4.3 PERSONNEL EFFORT REQUIREMENTS

Our personal effort will be very high throughout the year. This has been deemed a difficult senior project by our advisor, so it will be tough for us. We will be spending a very high volume of time during the fourth cycle (Build individual pieces, test, and demo) as we get ready to show off our project ideas in order to begin prototyping.

As semester two has changed our end results, our effort on our final product has stayed high. Although we are not able to test and demo our integrated product we have put a very high emphasis on completing the individual components and having them well documented.

Name	Overall Contributions to the Project
Jacob Tener	Neural Network Research, Analysis and Application. Constructed quantized Urban Sound 8k Model. Sound Parsing and Analysis. Applied CNN to microcontroller.
Jake Meiss	Energy harvesting research, power management design, system level architecture, Electrical Schematics, PCB Design
Andrew Vogler	FPGA Market Research, Building FPGA example projects, IP-Block Research, Requirements, Schedule Flow
Zixuan Guo	FPGA Testing research and deal with the data pipeline, Hardware design flow
Justin Sung	FPGA Market Research, FPGA Designing and Research, hardware/software design flow

4.4 OTHER RESOURCE REQUIREMENTS

- Microsemi's Igloo nano AGLN250V2
- Libero SoC IDE for Microsemi's FPGAs
- TI's MSP430FR5994
- Powercast P2110b RF Energy Harvester

4.5 FINANCIAL REQUIREMENTS

The financial requirements for our project is to keep everything we need to buy under \$200.

5. Testing, and Implementation

5.1 INTERFACE SPECIFICATIONS

- Libero SoC V11.9
- ModelSim
 - Verifies that the hardware works as intended.
- Synplify PRO
 - Verifies the power constraints are satisfied with the hardware design

5.2 HARDWARE AND SOFTWARE

- Igloo Nano ALGN250 Starter Kit
 - This was a reasonably priced FPGA and had all the specs we were looking for.
- Libero SoC V11.9
 - This is a tool that MicroSemi offers for free to use on all their FPGA products

- Powecast P2110B
 - A 915 MHz RF energy harvester that will be used to provide power to downstream devices
- Lab Instrumentation
 - Includes measurement devices such as multimeters and oscilloscopes used for testing and analysis

5.3 FUNCTIONAL/NON-FUNCTIONAL TESTING

Our testing phase was never executed due to COVID-19 complications. We meant to get more specific about our requirements once we integrated the device.

Test ID:	Test Type:	Grouping:	Given/When/Then	Verified ?	Additional Comments
	Functional:	Unit Tests:			
F-U-101			Given an RF Harvester Unit, When running, Then it shall produce the power as specified in our power analysis.		Add % error when better defined...
F-U-102			Given our sound classification software (training set and input), When the program executes, Then the input sound shall be accurately classified into the appropriate category.		See Test No 203 & 204 for Accuracy Percentage
F-U-103			Given the results of computation in memory, Then the results shall be exportable by an offline UART.		
F-U-104			Given a microcontroller, When a sound is given as an input to the microcontroller, Then the software embedded on the microcontroller shall output a classification for the input sound.		
F-U-105			Given an FPGA, When a		

			sound classification result sent as an input to the FPGA, Then the result shall be stored in the FPGA's memory.		
		Integration Tests:			
F-I-200			Given an FPGA and a Microcontroller, When the microcontroller has executed it's computation, Then the result shall be sent as an input to the FPGA.		
F-I-201			Given an RF Harvester and an FPGA, When the RF Harvester is running, Then the RF Harvester shall apply sufficient power to keep the FPGA running in steady, computation, and data storage states.		
		System Tests:			
F-S-300			Given an iFPGA system, Then it should not be placed under mild/extreme weather conditions but rather kept at room temperature.		
F-S-301			Given an iFPGA system, When and IF the FPGA shuts off mid-computation, Then the FPGA should not need to be reprogrammed to continue the computation.		
F-S-302			Given an iFPGA system, When the FPGA is toggled on->off->on, Then the computation should continue to run as well as data transfer once turned on again.		
		Acceptance			

		Tests:			
F-A-400			Given an iFPGA system, Then sufficient documentation shall be provided enabling the user to fix and understand the entire system.		
	Non-Functional Tests	Performance Tests:			
NF-P-500			Given the system's FPGA (Igloo Nano) , Then it should not run above ___ Watts		
NF-P-501			Given the system's FPGA (Igloo Nano) , Then the voltage applied to the device shall not exceed its maximum threshold (based on it's datasheet).		
NF-P-502			Given the system's microcontroller (MSP430) , Then the voltage applied to the device shall not exceed its maximum threshold (based on it's datasheet).		
NF-P-503			Given our sound classification software, Then the Training Accuracy shall be a minimum of 50%.		
NF-P-504			Given our sound classification software, Then the Testing Accuracy shall be a minimum of 50%.		

5.4 PROCESS

Requirements are verified by the Requirement Verification Matrix specified in section 5.3.

The general flow of testing will follow the diagram below.

Test Plan Flow Diagram

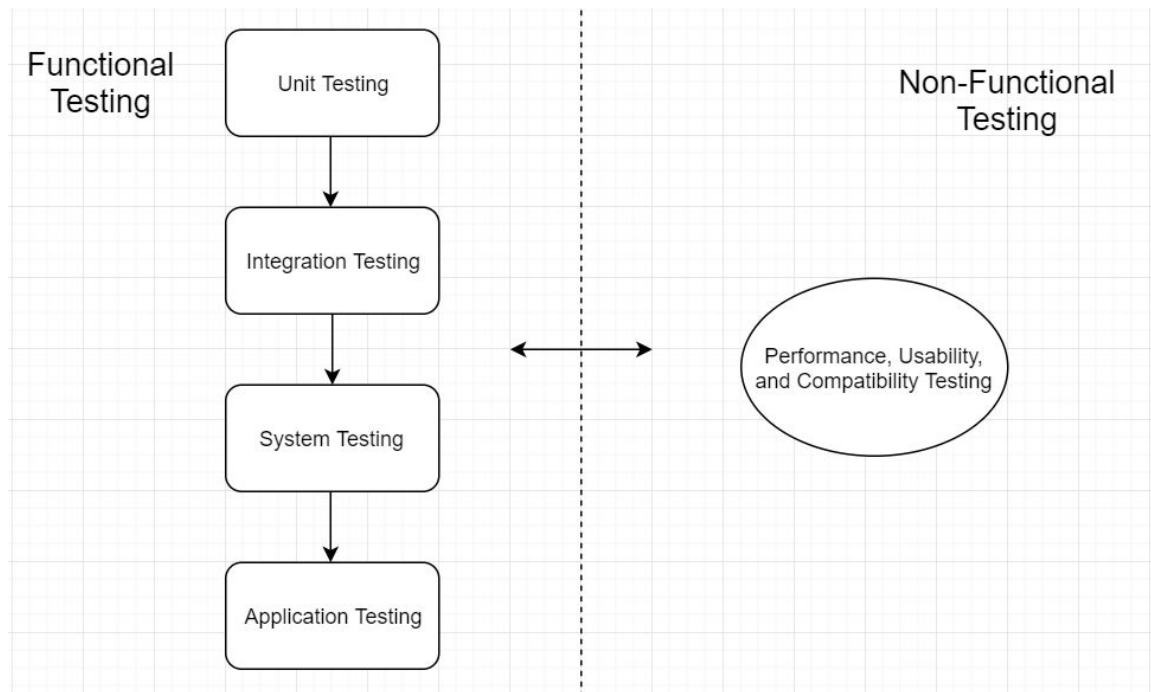


Figure 5.1 Test plan flow diagram

6. Results

Due to COVID-19, we were able to achieve 3 of our 4 adjusted goals. We developed the MAC hardware and confirmed through simulations and testing that it functioned as we intended. The software pipeline was completely converted into C to be placed on the MSP430 and was working as intended through testing. The PCB design was completed and the physical board was delivered on time. The last goal, establishing data communication between the MSP430 and the Nano was not achieved.

6.1 HARDWARE

- Schematics
 - Below you will find the full electrical schematics for our design

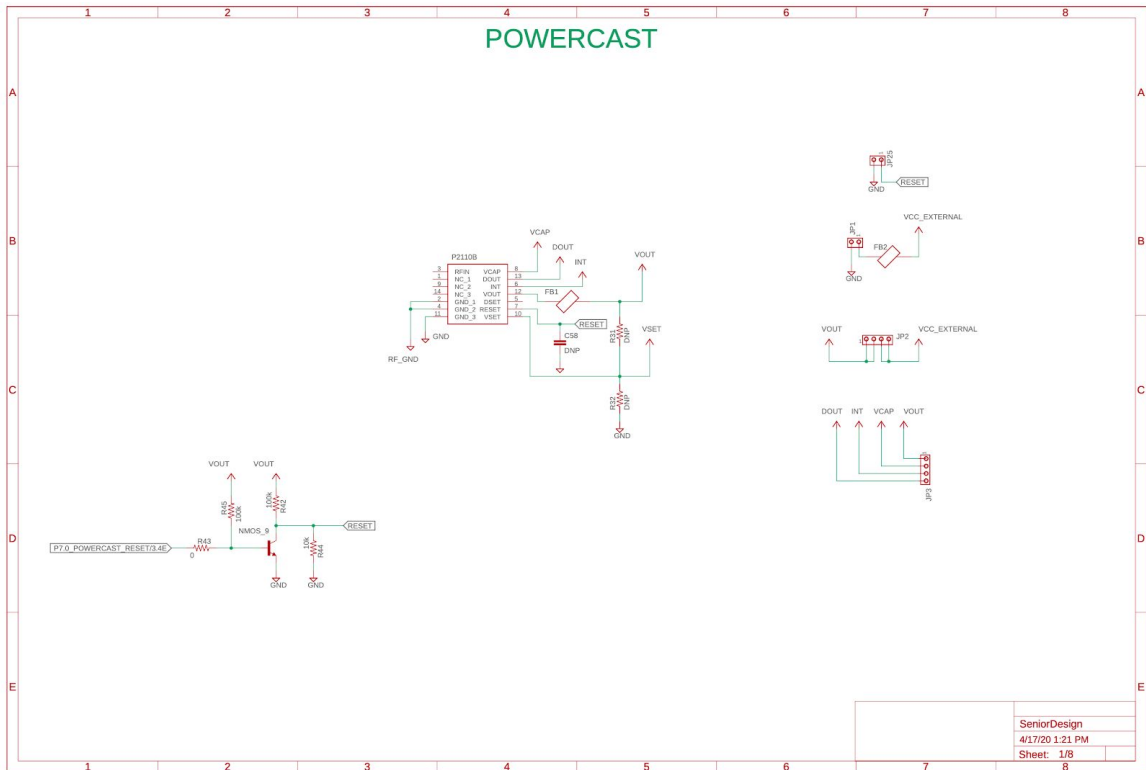


figure 6.11 - Powercast Energy Harvester Schematic

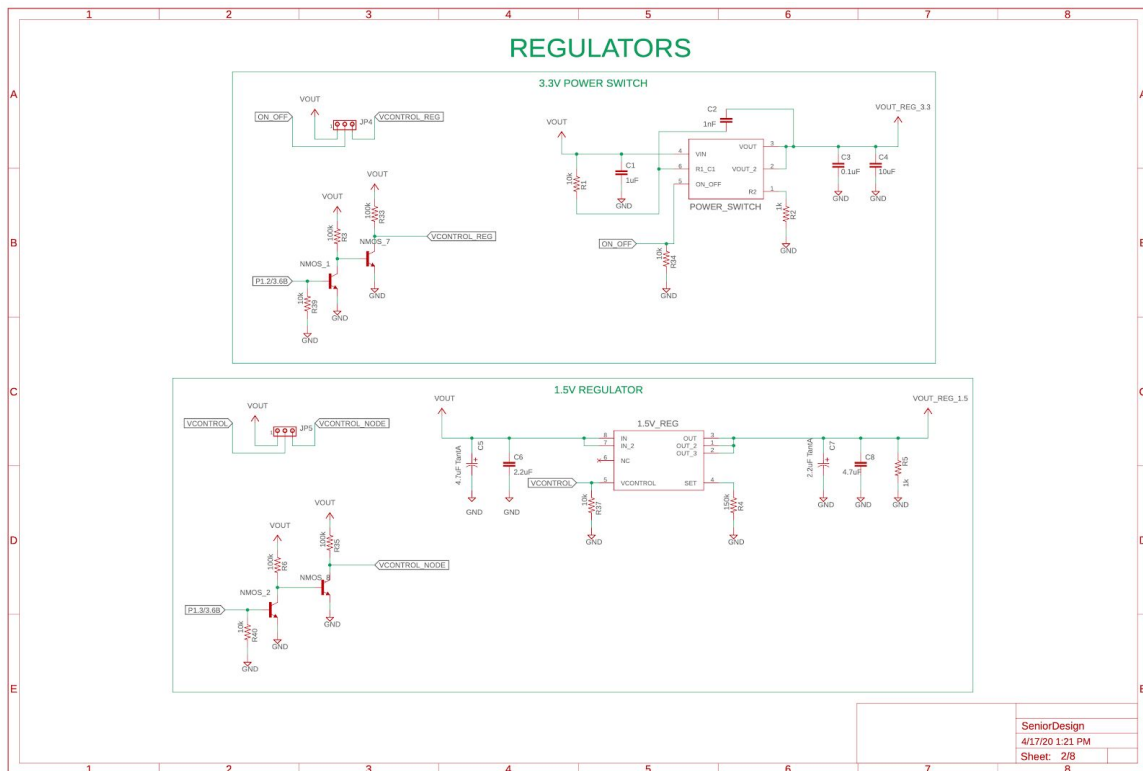


figure 6.12 - Regulators Schematic

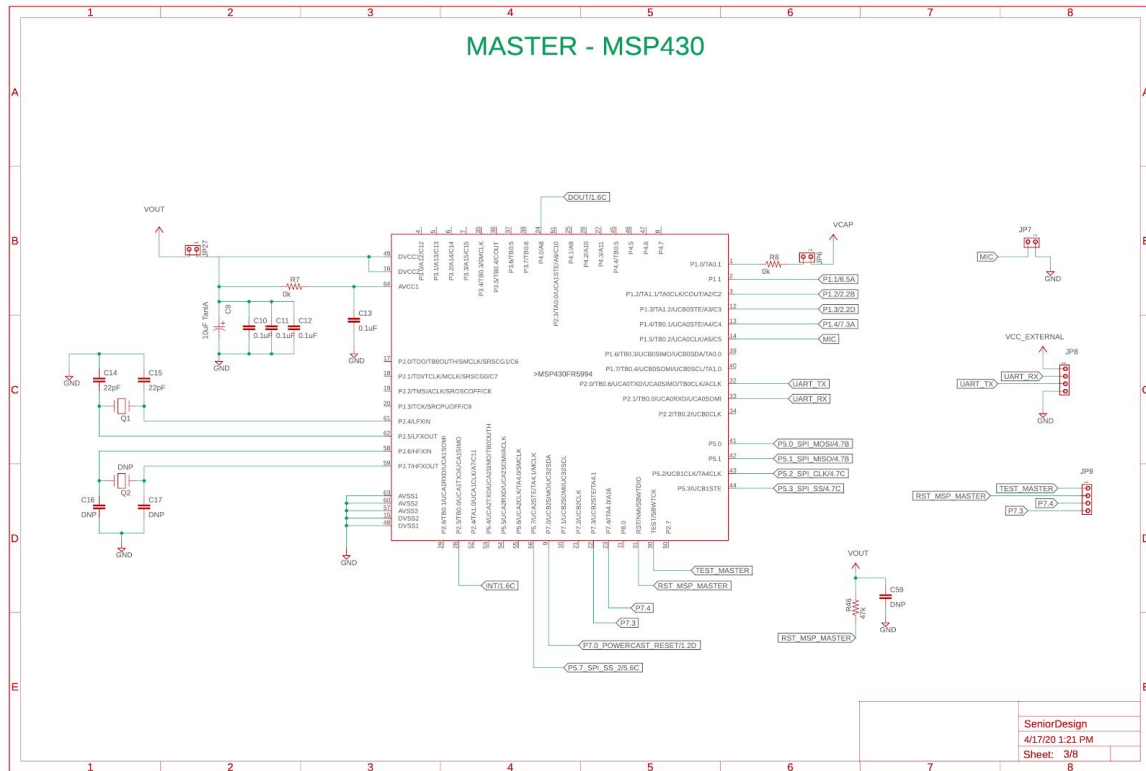


figure 6.13 - Schematic for the Master MSP430 Microcontroller

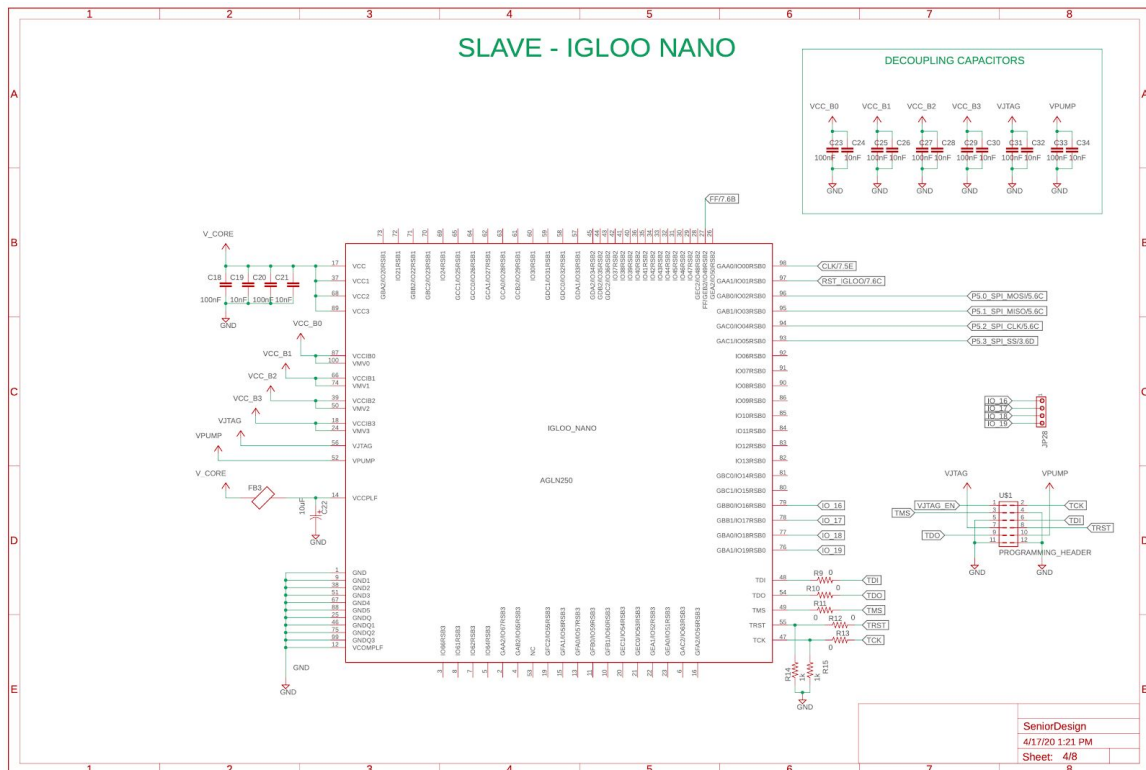


figure 6.14 - Schematic for the Microsemi Igloo Nano FPGA

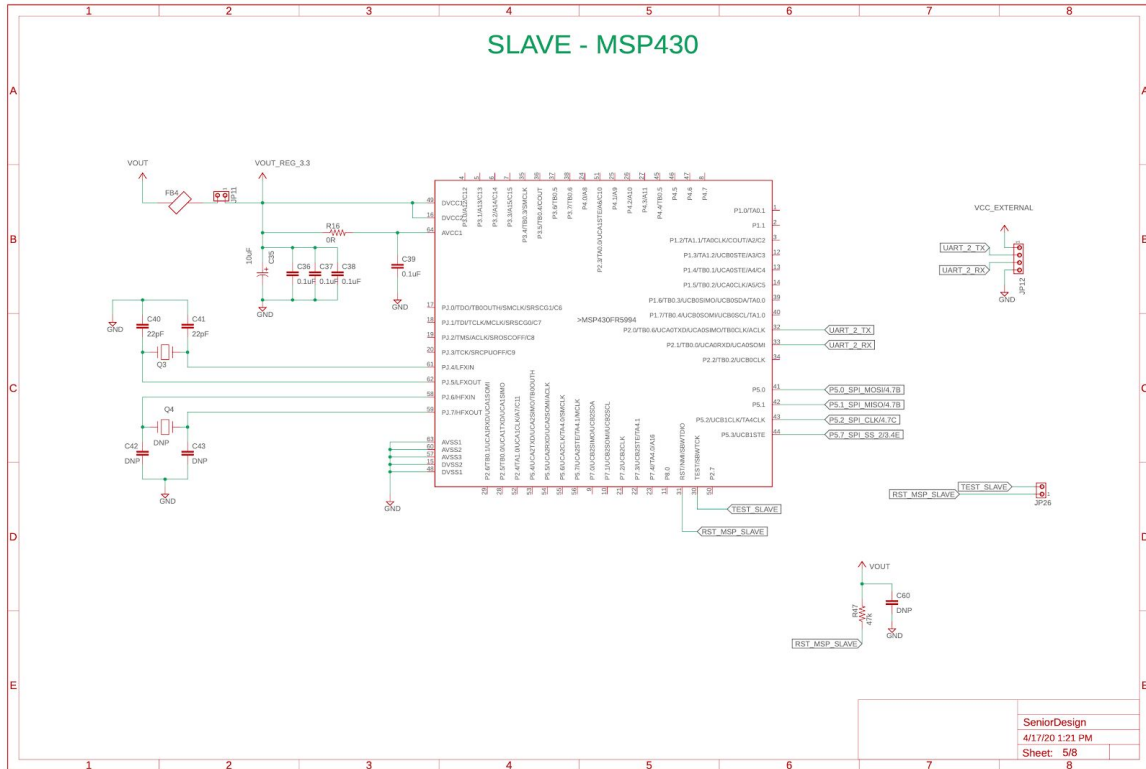


figure 6.15 - Schematic of the slave MSP430 Microcontroller

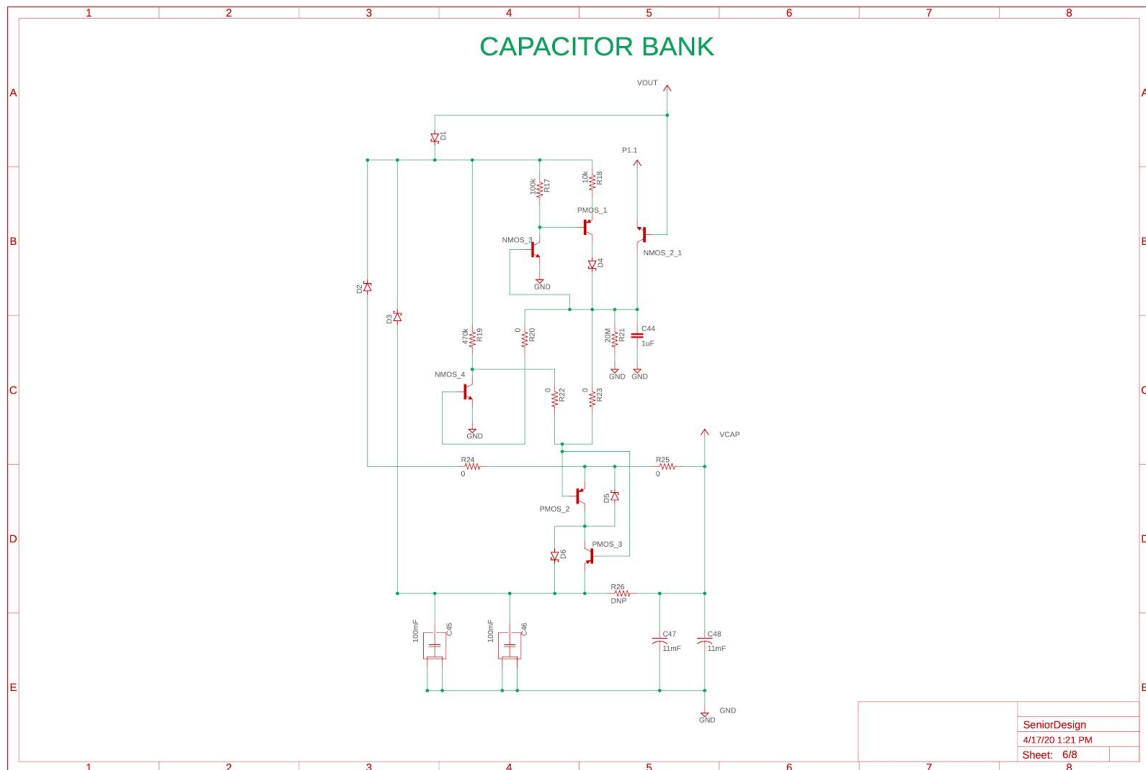


figure 6.16 - Schematic of the Capacitor Bank

- Printed Circuit Board Layout
 - Below you will find the printed circuit board layout design followed by images of the fabricated board

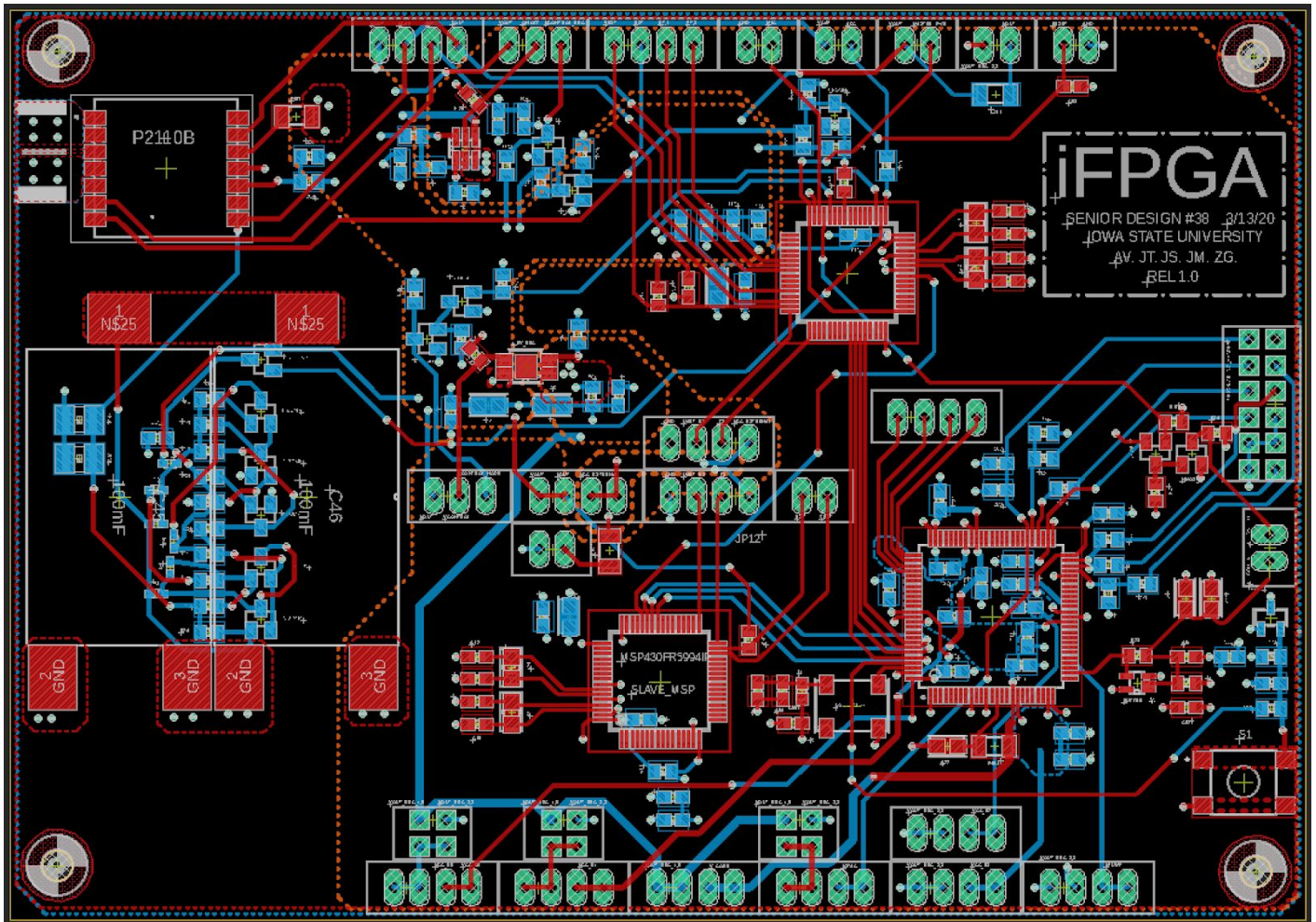


figure 6.19 - EAGLE Printed Circuit Board Layout Design

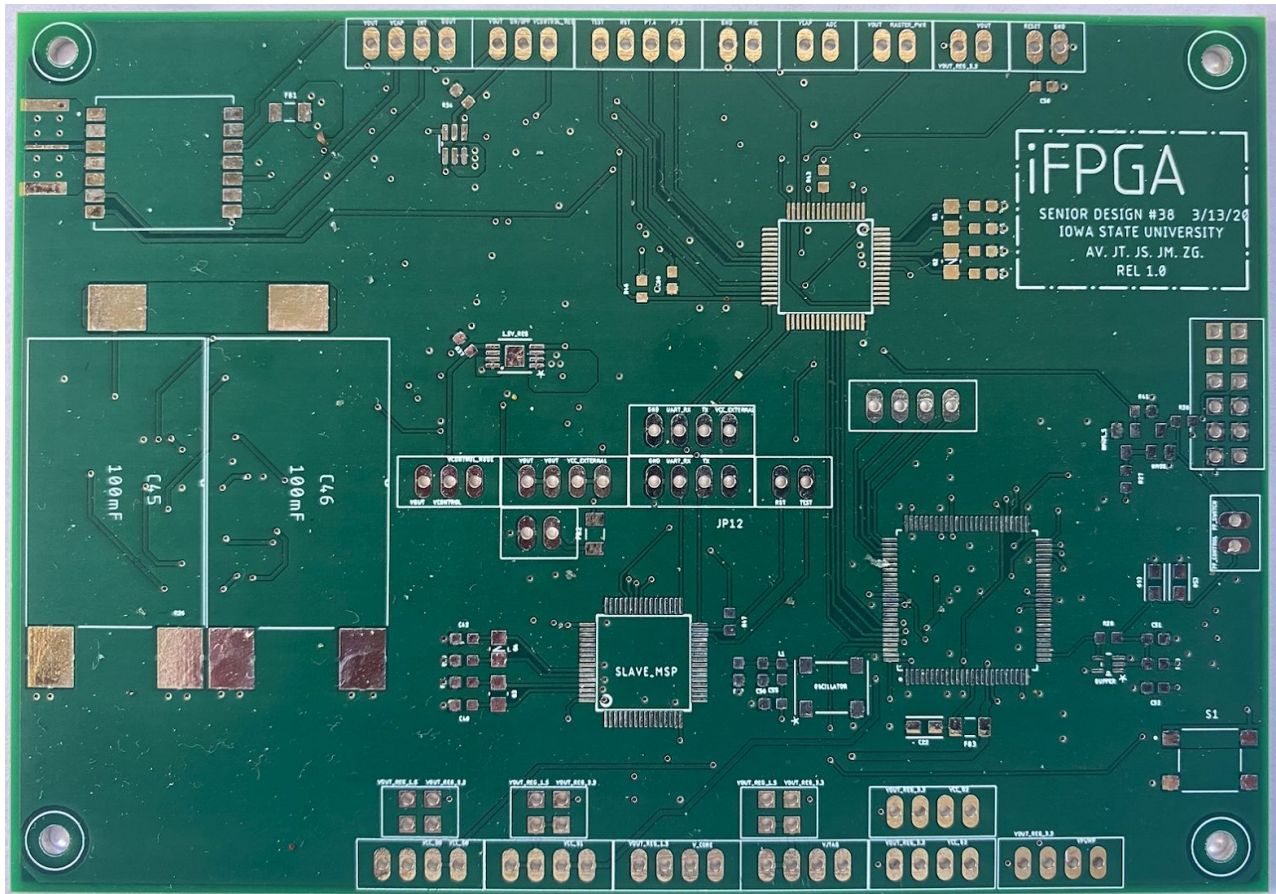


figure 6.20 - Final Fabricated Printed Circuit Board

Due to Covid-19, this is the extent of our results in reference to hardware deliverables. We have created a full working electrical schematic of our system, designed and fabricated a Printed circuit board, but were unable to populate the board and perform comprehensive testing of our system.

6.2 SOFTWARE

When running the “golden” case of having PC computation power and being able to run Python, classifications are produced very accurately and quickly. When running the following script:

Inputs to the script are: Quantized tflite model and .wav file (which is located in fold 5 and is of type street music)

```
import time
import librosa
import numpy as np
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.models import load_model
import tensorflow as tf
import numpy as np
import tensorflow as tf
import cProfile

def extract_feature(file_name):
    try:
        audio_data, sample_rate = librosa.load(file_name, sr=17000, res_type='kaiser_fast');
        mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=40);
        mfccscaled = np.mean(mfccs.T,axis=0);

    except Exception as e:
        print("Error encountered while parsing file: ", file)
        return None, None

    return np.array([mfccscaled])

# Load TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path=r"C:\Users\Jake\Desktop\senior design learning\SoundClassificationCode\saved_models\model_quantized_mlp.tflite")
interpreter.allocate_tensors()
# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

#try a prediction
filefile = 'UrbanSound8K/audio/fold5/36263-9-0-5.wav'

# Test model on random input data.
input_shape = input_details[0]['shape']
input_data = np.array(extract_feature(filefile), dtype=np.float32)

#print(input_data)
interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

# The function 'get_tensor()' returns a copy of the tensor data.
# Use 'tensor()' in order to get a pointer to the tensor.
output_data = interpreter.get_tensor(output_details[0]['index'])
#print(output_data[0])

class_label = ["air_conditioner", "car_horn", "children_playing", "dog_bark", "drilling", "engine_idling", "gun_shot", "jackhammer", "siren", "street_music"]
le = LabelEncoder()
le.fit_transform(class_label)

for i in range(len(output_data[0])):
    category = le.inverse_transform(np.array([i]))
    print(category[0], "\t\t : ", format(output_data[0][i], '.32f') )
```

Output is produced very quickly showing the following vectors:

```

air_conditioner      : 0.00542636681348085403442382812500
car_horn             : 0.00219394499436020851135253906250
children_playing    : 0.22465281188488006591796875000000
dog_bark            : 0.00172298587858676910400390625000
drilling            : 0.00126965437084436416625976562500
engine_idling       : 0.16991312801837921142578125000000
gun_shot            : 0.00016103865345939993858337402344
jackhammer          : 0.03553473949432373046875000000000
siren                : 0.00035691427183337509632110595703
street_music        : 0.55876839160919189453125000000000

```

The neural network calculations predict that the sound has a 55% chance to be street music, and that is correct! Since this vector had the majority, this is what would be outputted. In this case, we are able to use sound library [Librosa](#) in python which is an extremely handy tool. All of the sound analysis is performed in the first 3 lines of code.

As we move into the C++ implementation, things begin to get a little trickier. We were able to find another sound analysis library called [Aquila](#), and it helped out a little bit, but analysis still becomes a lot more involved when using a lower level programming language. When running the following file:

```

#include "C:\Users\Jake\Miniconda3\include\Python.h"
#include "aquila/global.h"
#include "aquila/source/generator/SineGenerator.h"
#include "aquila/transform/Mfcc.h"
#include "aquila/source/Wavefile.h"
#include "aquila/source/FramesCollection.h"
#include "aquila/source/Frame.h"
#include <algorithm>
#include <cstdint>
#include <iostream>
#include <iterator>
#include <fstream>
#define PY_SSIZE_T_CLEAN

```



```

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        std::cout << "Usage: wave_iteration <FILENAME>" << std::endl;
        return 1;
    }

    // string filename = "C:\\Users\\Jake\\Desktop\\streetmusic.wav"
    // Aquila::WaveFile wav(filename);
    Aquila::WaveFile wav(argv[1]);
    std::cout << "Loaded file: " << wav.getFilename()
    << " (" << wav.getBitsPerSample() << "b)" << std::endl;
    std::cout << wav.getWaveSize() << std::endl;
    std::cout << wav.getBytesPerSample() << std::endl;
    std::cout << wav.getBytesPerSec() << std::endl;
    std::cout << wav.getSampleFrequency() << std::endl;

    std::cout << "# of Samples" << wav.getSamplesCount() << std::endl;

    std::vector<double> m_data;

    wav *= (1/32767.0);
    for (std::size_t i = 0; i < wav.getSamplesCount(); ++i)
    {
        // std::cout << "Sample #:" << i << " " << wav.sample(i) << std::endl;
    }

    double FRAME_SIZE = 8192;
    double HOP_LENGTH = FRAME_SIZE - wav.getSamplesCount()/40 +100;
    std::cout << "value: " << HOP_LENGTH << std::endl;
    Aquila::FramesCollection frames(wav, FRAME_SIZE, HOP_LENGTH);
    Aquila::Mfcc mfcc(FRAME_SIZE);
    std::cout << "frame count:" << frames.count() << std::endl;
    double total = 0.0;
    double average = 0.0;
    double scaledArray[40] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
};

    for(int i = 0; i < frames.count(); i++)
    {
        Aquila::Frame frame = frames.frame(i);
        auto mfccValues = mfcc.calculate(frame);
        total = 0.0;
        average = 0.0;
        for(int j = 0; j<mfccValues.size(); j++){
            total += mfccValues[j];
        }
        average = total / mfccValues.size();
        //std::cout << "value: " << total << std::endl;

        scaledArray[i] = average;
        std::cout<<"Frame #"<<i<<"Length of array:" << mfccValues.size() << " mfcc coefficients: " <<std::endl;
        //
        //         std::copy(
        //             std::begin(mfccValues),
        //             std::end(mfccValues),
        //             std::ostream_iterator<double>(std::cout, " "));
    }
    std::ofstream outfile ("C:\\Users\\Jake\\Desktop\\senior design learning\\SoundClassificationCode\\MFCC.txt", std::ofstream::out);
    for(const double &index : scaledArray){
        outfile<< index <<",";
        // std::cout << "value: " << index << std::endl;
    }
    outfile.close();
}

```

A .txt file is populated with the generated MFCC Coefficients needed to perform classification. In our intended implementation, this would just be uploaded to an MSP430 and the coefficients could be transmitted to the Nano for our MAC process with the model weights. However, for the high level software testing, these coefficients need to be ran through model prediction to make sure that the C++ implementation of sound analysis is performing similarly to the python version. We will do that with the following python script:

```

import time
import librosa
import numpy as np
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.models import load_model
import tensorflow as tf
import numpy as np
import tensorflow as tf
import cProfile

```

```

def inference():
    # Load TFLite model and allocate tensors.
    interpreter = tf.lite.Interpreter(model_path='C:\Users\Jake\Desktop\senior design learning\SoundClassificationCode\saved_models\model_quantized_mlp.tflite')
    interpreter.allocate_tensors()
    # Get input and output tensors.
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    # Test model on random input data.
    input_shape = input_details[0]['shape']
    input_data = np.array([[
        -7.57547, -3.24882, -3.28906, -3.01968, -2.79623, -2.64877, -2.37866, -1.92549, -2.51433, -2.29119, -2.53459, -4.02543, -6.91553, -8.7626, -7.94557, -5.14043, -3.12745, -3.86068, -2.82138,
    ]], dtype=np.float32)

    #print(input_data)
    interpreter.Set_tensor(input_details[0]['index'], input_data)
    interpreter.invoke()

    # The function 'get_tensor()' returns a copy of the tensor data.
    # Use 'tensor()' in order to get a pointer to the tensor.
    output_data = interpreter.get_tensor(output_details[0]['index'])
    print(output_data[0])

    class_label = ["air_conditioner", "car_horn", "children_playing", "dog_bark", "drilling", "engine_idling", "gun_shot", "jackhammer", "siren", "street_music"]
    le = LabelEncoder()
    le.fit_transform(class_label)

    for i in range(len(output_data[0])):
        category = le.inverse_transform(np.array([i]))
        print(category[0], "\t\t: ", format(output_data[0][i], '.32f') )

inference();

```

Here the Coefficients from the .txt file are copied over and entered into the input_data array (formatting issues prevented us from loading the .txt file directly) and, output is produced as follows:

```

air_conditioner           : 0.00956509076058864593505859375000
car_horn                  : 0.00185369467362761497497558593750
children_playing         : 0.00036051828647032380104064941406
dog_bark                  : 0.14305174350738525390625000000000
drilling                  : 0.23240849375724792480468750000000
engine_idling             : 0.03721703588962554931640625000000
gun_shot                  : 0.01222001947462558746337890625000
jackhammer                : 0.02769083902239799499511718750000
siren                     : 0.00519357295706868171691894531250
street_music              : 0.53043901920318603515625000000000

```

The coefficients generated through the C++ model still work with our model, as street music is still the greatest probability vector at 53%. There is a slight reduction in confidence, which may affect overall accuracy.

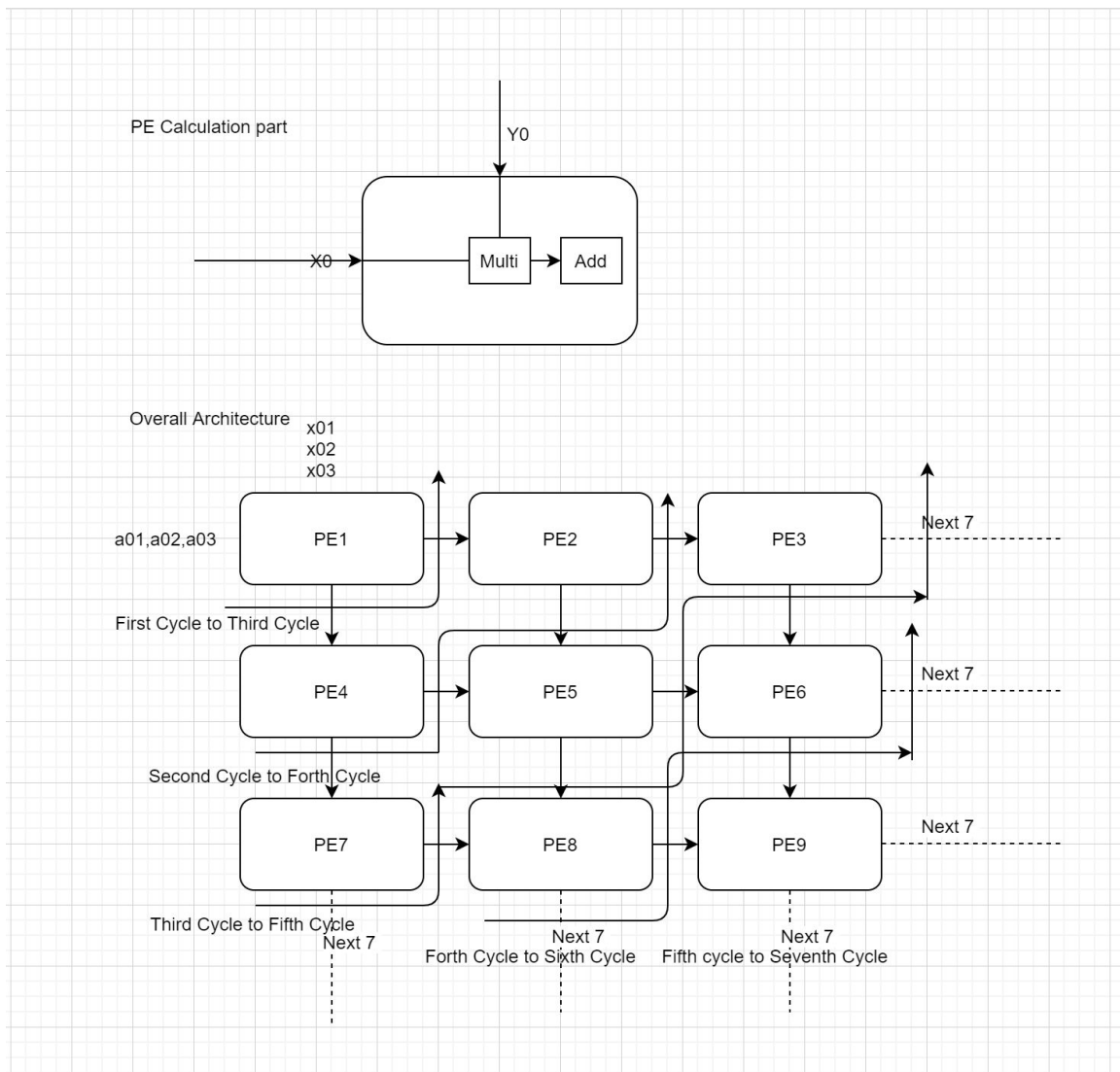
Unfortunately, due to Covid-19 complications and time restrictions, modular and complete testing of the C++ implementation was not achieved and therefore could maintain some inaccuracies in the prediction process. Since the model was generated with the Librosa sound library, some of the sampling and MFCC calculations may differ from the Aquila calculations.

Upon further testing, it may be needed that a new model be trained using aquila to analyze each .wav file and generate model weights. The Librosa training script has been omitted here in the result section, but will be included in the appendix.

Model upload and testing on MSP430 has been performed by Sahu Rohit and other grad students, who had an existing implementation of neural networking on the MSP and attempted to apply it to our neural network application. Since we were not able to implement our software due to Covid-19, we did not dive further into testing the model on the hardware.

6.3 FPGA AND MCU SYSTEM

- MAC accelerator Processor
 - Designed a 10 x 10 matrix multiplier dependent on a systolic array algorithm (VHDL) and simulated on the Modelsim. List below is the algorithm structure and the part of the code.
 - Systolic Array Structure



```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity PES is
port(clk, reset      : in  std_logic;
      matrix         : in  std_logic_vector(3 downto 0);
      i_data0        : in  std_logic_vector(7 downto 0);
      i_data1        : in  std_logic_vector(3 downto 0);
      o_out0         : out std_logic_vector(7 downto 0)
      -- o_out1      : out std_logic_vector(3 downto 0)
);

end PES;

architecture behavioral of PES is
    signal t1      : std_logic_vector(7 downto 0);
begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            if (reset = '1') then
                t1      <= x"00";
                o_out0 <= x"00";
                --o_out1 <= x"0";
            else
                t1 <= matrix * i_data1;
                o_out0 <= t1 + i_data0;
                --o_out1 <= i_data1;
            end if;
        end if;
    end process;
end behavioral;

```

- Part of Code for the VHDL Structural behavioral, Combine PE(Calculation Part) block with the matrix multiplication

```

begin
-- First cycle
U1: PES port map(clk => clk, reset => rst, matrix => s01, i_data0 => x"00", i_data1 => a01, o_out0 => b1);
R1: reg_delay port map(clk => clk, D => a02, Q => p1);
R2: reg_delay port map(clk => clk, D => a21, Q => p2);
R3: reg_delay port map(clk => clk, D => a11, Q => p3);
R4: reg_delay port map(clk => clk, D => a01, Q => p4);
-- Second cycle
U2: PES port map(clk => clk, reset => rst, matrix => s02, i_data0 => b1, i_data1 => p3, o_out0 => b2);
U3: PES port map(clk => clk, reset => rst, matrix => s01, i_data0 => x"00", i_data1 => p1, o_out0 => b3);
U4: PES port map(clk => clk, reset => rst, matrix => s11, i_data0 => x"00", i_data1 => p4, o_out0 => b4);
--R30: reg_delay port map(clk => clk, D => b2, Q => p30);
R5: reg_delay port map(clk => clk, D => a03, Q => p5);
R6: reg_delay port map(clk => clk, D => a12, Q => p6);
R7: reg_delay port map(clk => clk, D => p2, Q => p7);
R8: reg_delay port map(clk => clk, D => a22, Q => p8);
R9: reg_delay port map(clk => clk, D => p4, Q => p9);
R10: reg_delay port map(clk => clk, D => p1, Q => p10);
R11: reg_delay port map(clk => clk, D => p3, Q => p11);
--Third cycle
U5: PES port map(clk => clk, reset => rst, matrix => s21, i_data0 => x"00", i_data1 => p9, o_out0 => b5);
U6: PES port map(clk => clk, reset => rst, matrix => s12, i_data0 => b4, i_data1 => p11, o_out0 => b6);
U7: PES port map(clk => clk, reset => rst, matrix => s03, i_data0 => b2, i_data1 => a21, o_out0 => c01);
U8: PES port map(clk => clk, reset => rst, matrix => s01, i_data0 => x"00", i_data1 => p5, o_out0 => b7);
U9: PES port map(clk => clk, reset => rst, matrix => s02, i_data0 => b3, i_data1 => p6, o_out0 => b8);
U10: PES port map(clk => clk, reset => rst, matrix => s11, i_data0 => x"00", i_data1 => p10, o_out0 => b9);
R12: reg_delay port map(clk => clk, D => p5, Q => p12);
R13: reg_delay port map(clk => clk, D => p11, Q => p13);
R14: reg_delay port map(clk => clk, D => p6, Q => p14);
R15: reg_delay port map(clk => clk, D => a23, Q => p15);
R16: reg_delay port map(clk => clk, D => p8, Q => p16);
R30: reg_delay port map(clk => clk, D => p16, Q => p30);
R35: reg_delay port map(clk => clk, D => p30, Q => p35);
--R40: reg_delay port map(clk => clk, D => p35, Q => p40);
R17: reg_delay port map(clk => clk, D => p10, Q => p17);
R18: reg_delay port map(clk => clk, D => p7, Q => p18);
R31: reg_delay port map(clk => clk, D => p18, Q => p31);
--R36: reg_delay port map(clk => clk, D => p31, Q => p36);
R19: reg_delay port map(clk => clk, D => a13, Q => p19);
--Forth cycle
U11: PES port map(clk => clk, reset => rst, matrix => s02, i_data0 => b7, i_data1 => p19, o_out0 => b10);
U12: PES port map(clk => clk, reset => rst, matrix => s11, i_data0 => x"00", i_data1 => p12, o_out0 => b11);
U13: PES port map(clk => clk, reset => rst, matrix => s03, i_data0 => b8, i_data1 => p35, o_out0 => c02);
U14: PES port map(clk => clk, reset => rst, matrix => s12, i_data0 => b9, i_data1 => p14, o_out0 => b13);

```

7. Closing Material

7.1 CONCLUSION

SEMESTER 1

So far, we have gotten through most of the thorough description of the design. Our software can do sound classification, we know which part of the software we are going to accelerate, and we have an understanding of how we are going to attach it to our embedded hardware. On the electrical side, we have done power analysis for the FPGA as well as done the number crunching so we know what electrical components we are going to buy. For the Embedded System, we have picked out a board as well as the ip-blocks we are going to use when programming, and the channels of communication between devices and to external devices. Our goal for this project is to create a PCB-based battery-less FPGA platform that can accelerate software computations. Most of this goal is achieved in simply making the physical device and programming it. Making a PCB is one of the tasks to be taken care of in the beginning of the semester. The battery-less capability should be feasible based on our power calculations but it may need some tweaking as we learn more about the software. The acceleration will be feasible as long as we give ourselves enough time to test and make tweaks to our design as necessary. Since we won't know if the part of the software we choose to accelerate will actually accelerate, it is important that we start the next semester running. We

are finishing the semester with an in-depth analysis of how to integrate the individual components of the project. The best way to make sure this whole design can actually accelerate a computation is to build it as quickly as possible up front so in the case the software doesn't actually accelerate, we have time to figure out why and make appropriate changes to our design so that it does.

SEMESTER 2

Second semester has proven to be both very challenging and rewarding for our project. Due to Covid-19, we were forced to redefine many of our deliverables but we were able to make considerable progress on our project nonetheless. In terms of hardware, we were able to create a powerflow that was able to withstand our criteria, decide on a capacitance that would be big enough to allow our system to boot up and perform necessary computations while not big enough to cause serious delay through charging time, complete full electrical schematics, and design and fabricate a PCB that is awaiting population and testing due to the extenuating circumstances.

Software complications have been in attempting to implement the "golden" python system into a lower level language suitable for upload onto the embedded system. We were able to develop a new sound analysis pipeline in C++ that produces similar enough data to be implemented on the MSP's. It's similar accuracy to the python scripts results have not been easy to produce.

Establishing data communications via SPI between the MSP430 and the Nano took more time than we had anticipated. Lack of documentation for core configuration and processor programming led to the delay in achieving what we planned to accomplish. Eventually, data communication with the varying cores locally within the Nano functioned as intended, but the addition of the MSP430 resulted in unsuccessful attempts. Memory I/O operations were demonstrated to function as intended. The MAC hardware was designed and through simulations and testing, was confirmed to function as intended.

Although the semester did not seem to go quite like any of us expected, we have effectively laid the groundwork for an intelligent intermittent FPGA system.

7.2 REFERENCES

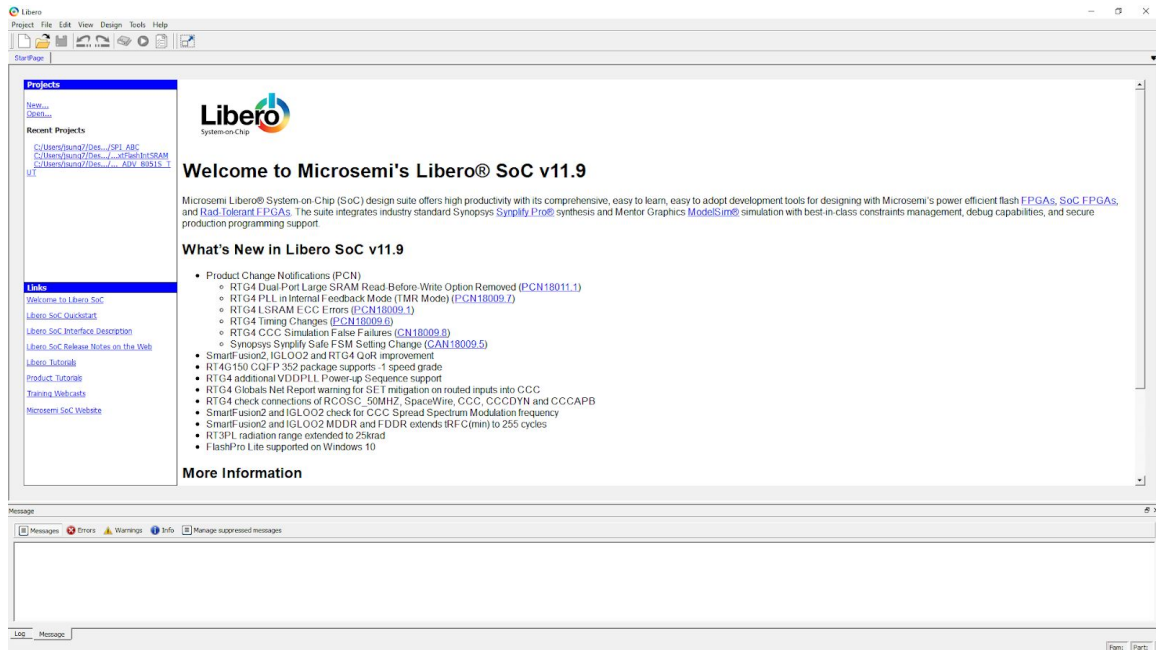
K. Kowalczyk, S. Wozniak, T. Chyrowicz and R. Rumian, "Embedded system for acquisition and enhancement of audio signals," *2016 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, Poznan, 2016, pp. 68-71.

7.3 APPENDICES

I: OPERATION MANUAL

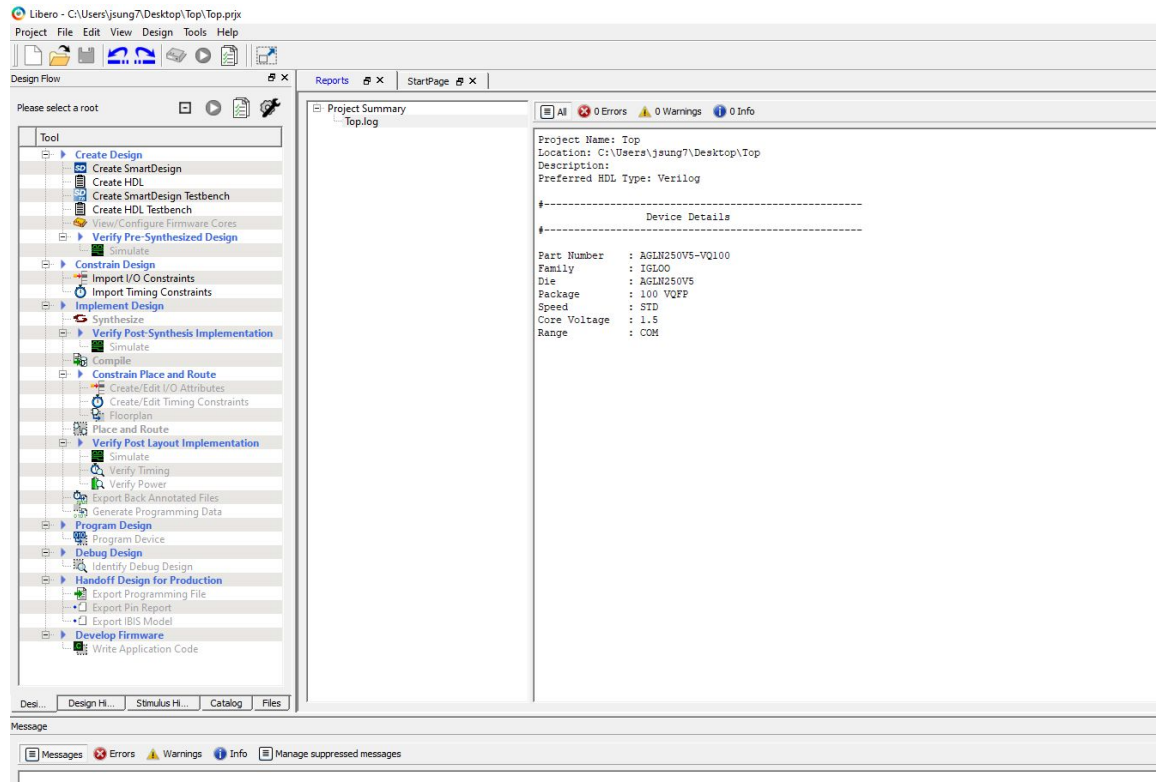
Downloading Libero SoC v11.9

1. Libero SoC v11.9 is the last version that supports Igloo Nano development. It can be through the Microsemi website. Follow the instruction for a node-locked silver license.
2. Open the application and you should see the home page:



SPI Communication Design on the Igloo Nano

1. In the upper tab, go to **Project**, name the project, and hit **Next**. Choose:
 - a. Family : IGLOO
 - b. Die: AGLN250V2
 - c. Package: 100VQFP
2. Hit **Finish**. This should be what you see:

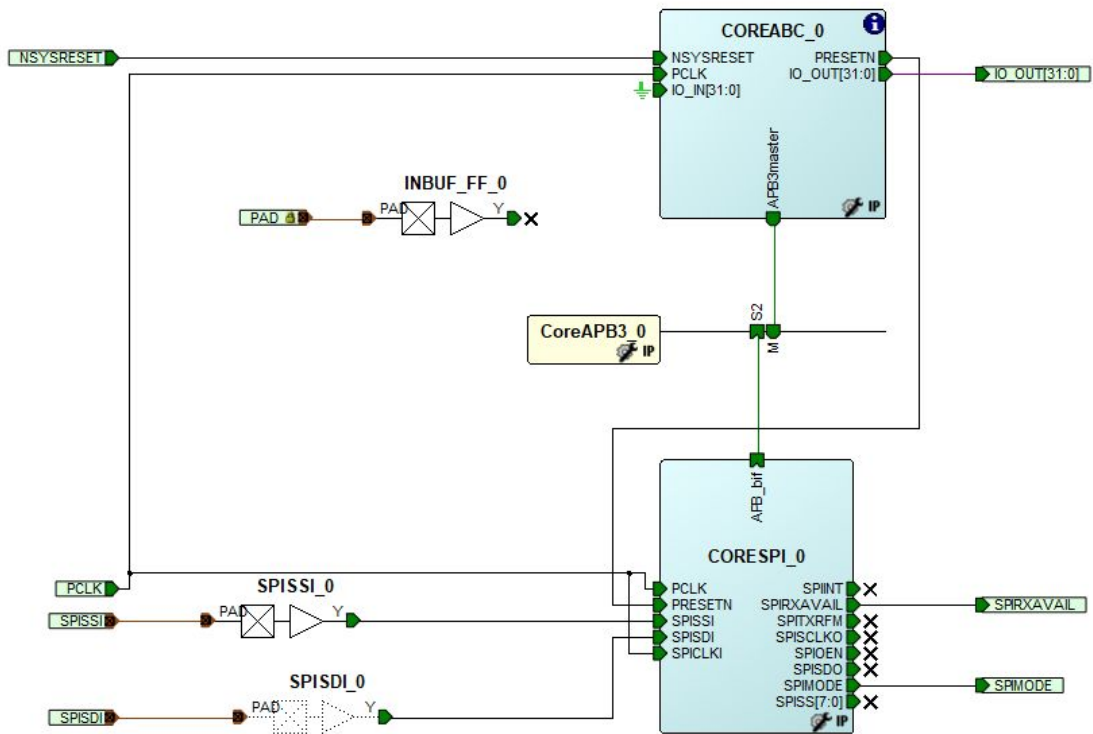


3. On the left double click **Create SmartDesign**.
4. On the bottom of the left hand panel, click **Catalog**.
Search and add to the canvas:
 - a. CoreABC
 - b. CoreAPB3
 - c. CoreSPI

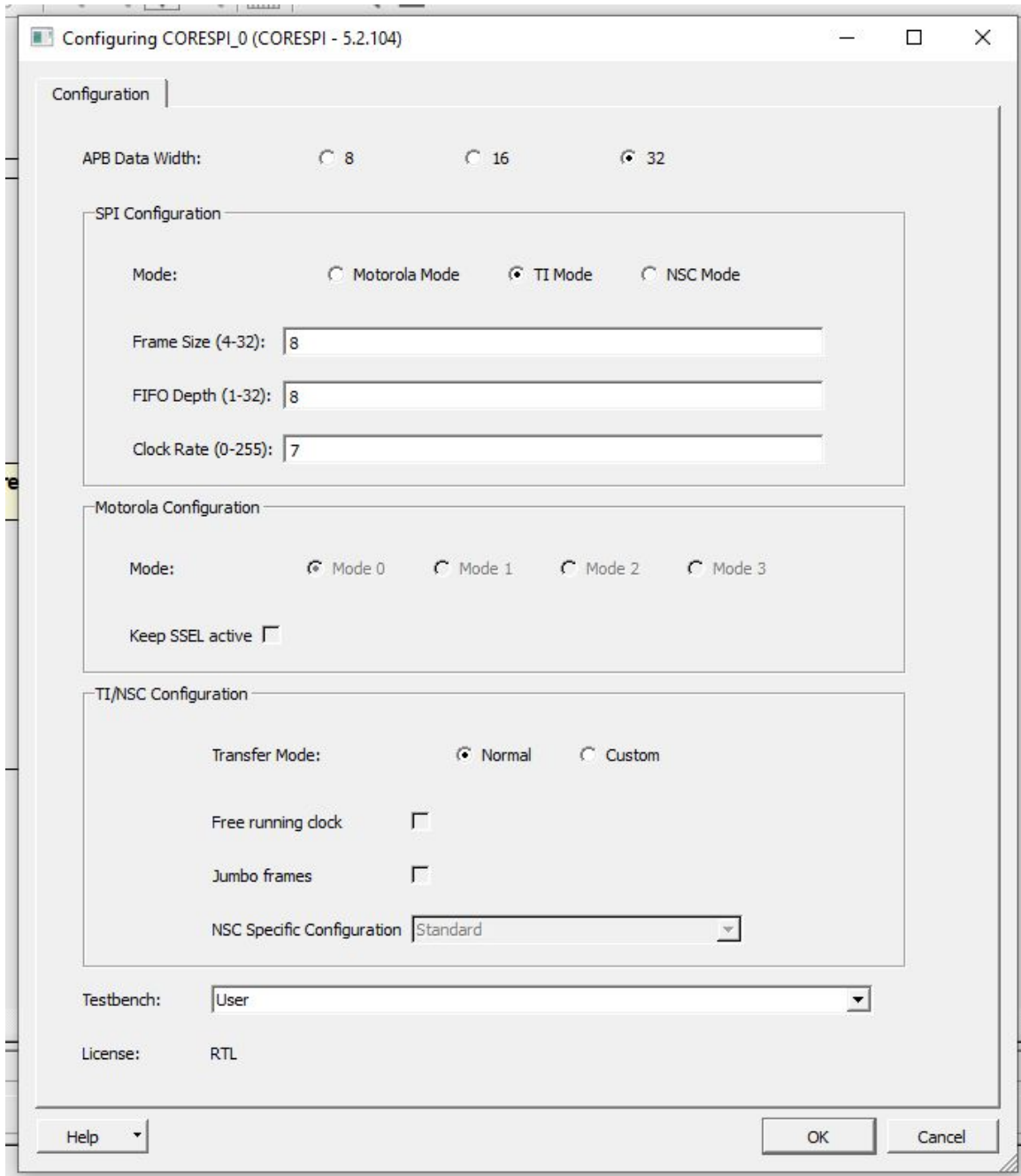
CoreABC is the processor in charge of coordinating the other cores and data transmission. CoreAPB3 is the bus controller designed for CoreABC. It uses the AMBA bus protocol. CoreSPI is the SPI controller in charge of receiving and transmitting data.

Core Configuration and Linkage

1. Top Level:
This design depicts the basic hardware configurations for SPI communication.



2. CoreSPI:



3. CoreABC:

The screenshot shows a configuration window titled "Configuring COREABC_0 (COREABC 3.7.101)". It has three tabs: "Parameters" (selected), "Program", and "Analysis".

Size Settings

- Data Bus Width : 32
- Number of APB Slots : 3
- APB Slot Size : 64k locations
- Maximum Number of Instructions : 32
- Z Register Size (Bits) : Disabled
- Number of I/O Inputs : 32
- Number of I/O Flags : 0
- Number of I/O Outputs : 32
- Stack Size : 16
- Init/Config Address Width : 11

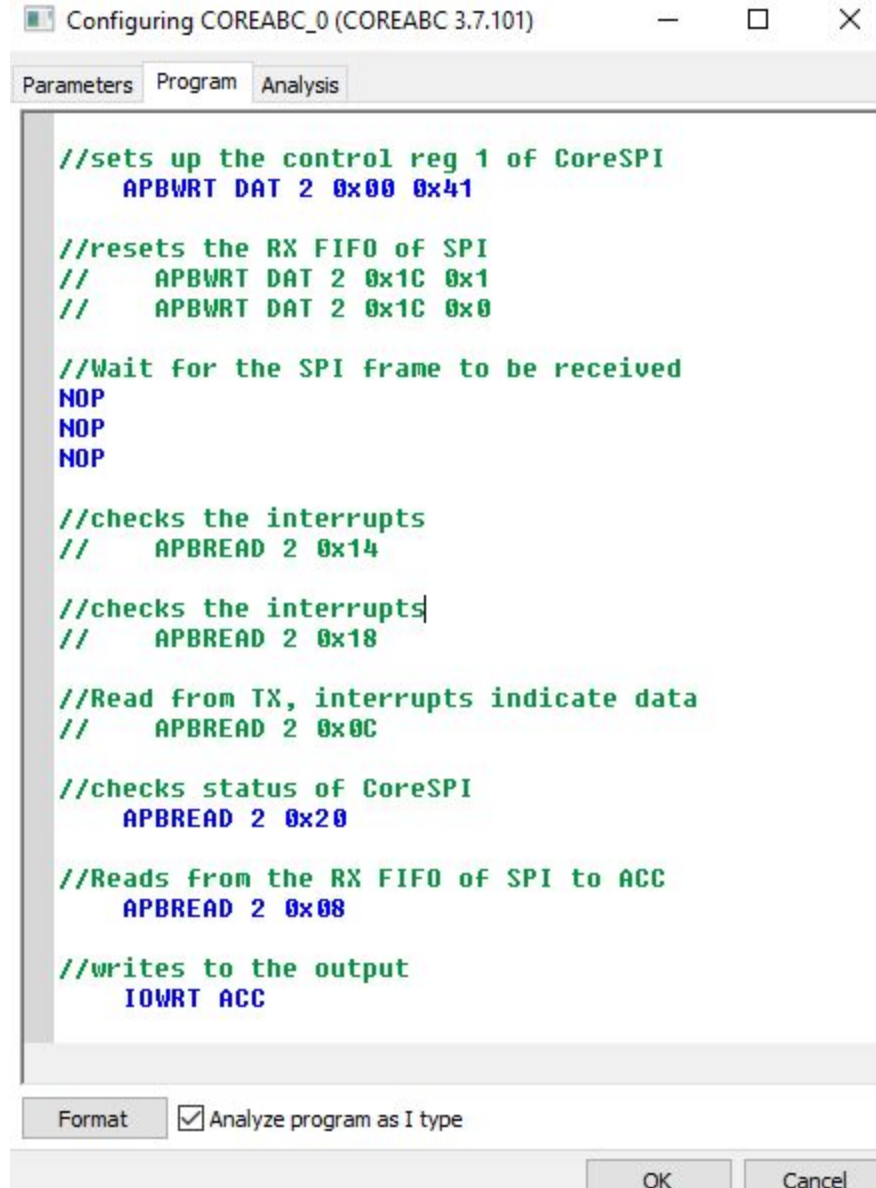
Memory and Interrupt

- Instruction Store : Hard (FPGA Logic Elements)
- Instruction Store APB Access : None
- Use Calibration NVM :
- Internal Data/Stack Memory :
- ALU Operations from Memory :
- APB Indirect Addressing :
- Supported Data Sources : Accumulator and Immediate
- Interrupt Support : Disabled
- ISR Address : 1

Optional Instructions

AND, BITCLR, BITTST : <input checked="" type="checkbox"/>	XOR, CMP : <input checked="" type="checkbox"/>
OR, BITSET : <input checked="" type="checkbox"/>	ADD, SUB, DEC, CMPEQ : <input checked="" type="checkbox"/>
INC : <input checked="" type="checkbox"/>	SHL, ROL : <input checked="" type="checkbox"/>
SHR, ROR : <input checked="" type="checkbox"/>	CALL, RETURN, RETISR : <input checked="" type="checkbox"/>
PUSH, POP : <input checked="" type="checkbox"/>	APBWRT ACM : <input checked="" type="checkbox"/>
IOREAD : <input checked="" type="checkbox"/>	IOWRT : <input checked="" type="checkbox"/>
MULT : Not Implemented	

This assembly depicts basic interfacing with the CoreSPI.



```
//sets up the control reg 1 of CoreSPI
  APBWRT DAT 2 0x00 0x41

//resets the RX FIFO of SPI
//  APBWRT DAT 2 0x1C 0x1
//  APBWRT DAT 2 0x1C 0x0

//Wait for the SPI frame to be received
NOP
NOP
NOP

//checks the interrupts
//  APBREAD 2 0x14

//checks the interrupts|
//  APBREAD 2 0x18

//Read from TX, interrupts indicate data
//  APBREAD 2 0x0C

//checks status of CoreSPI
  APBREAD 2 0x20

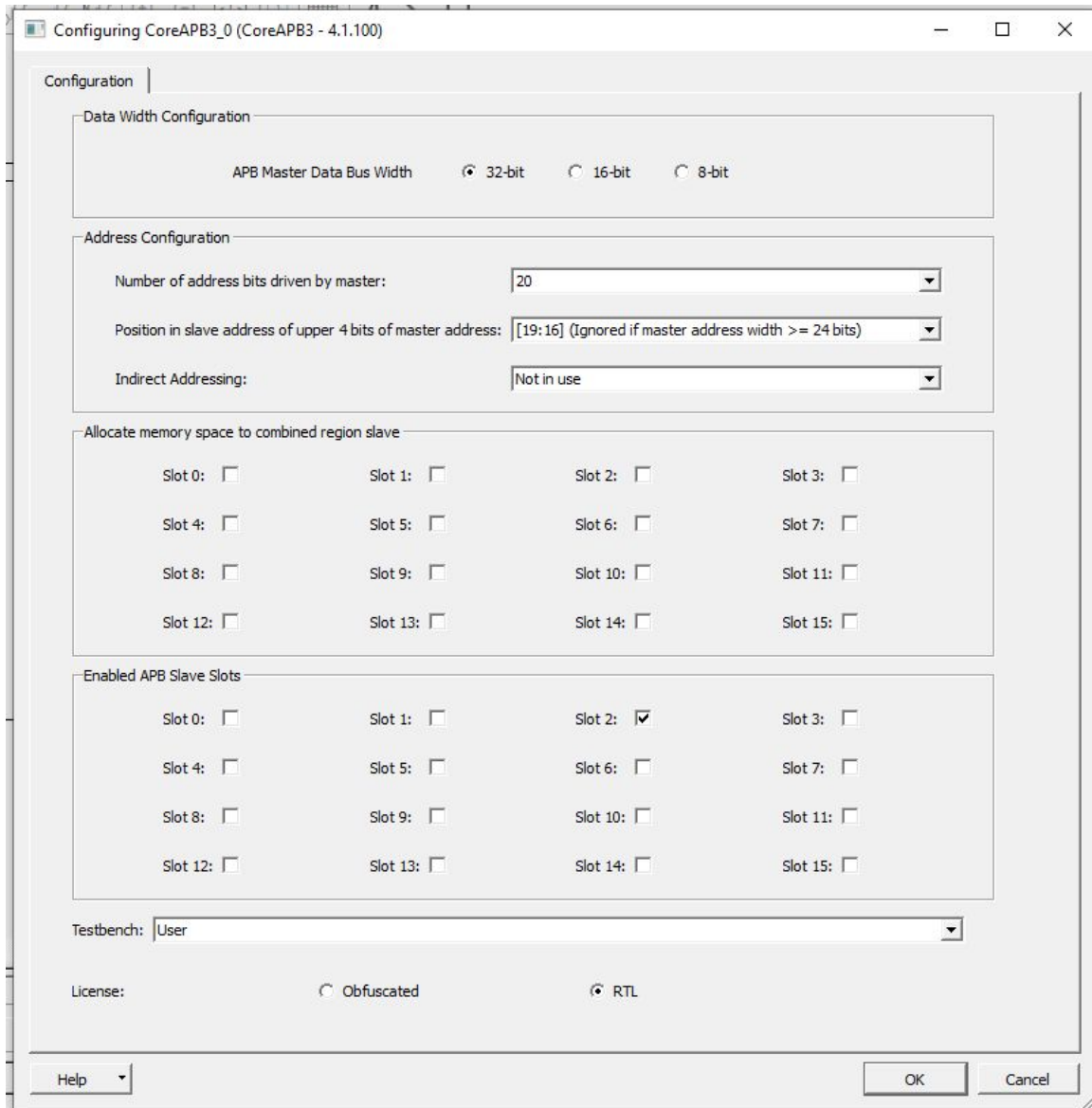
//Reads from the RX FIFO of SPI to ACC
  APBREAD 2 0x08

//writes to the output
  IOWRT ACC
```

Format Analyze program as I type

OK Cancel

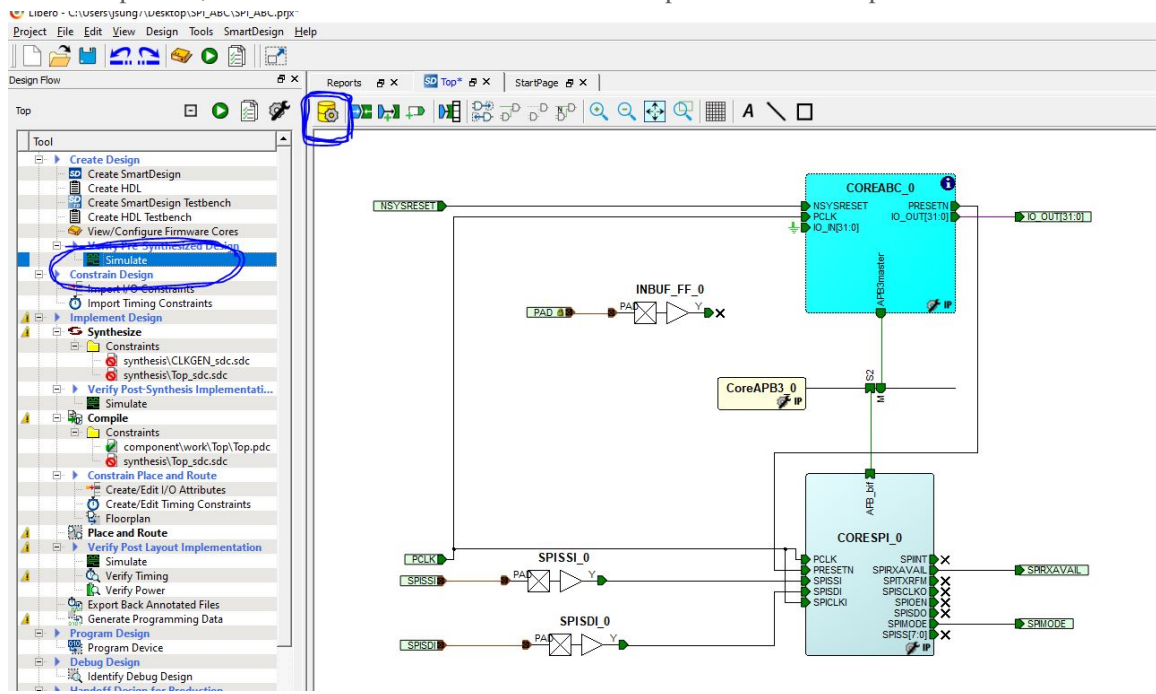
4. CoreAPB3:



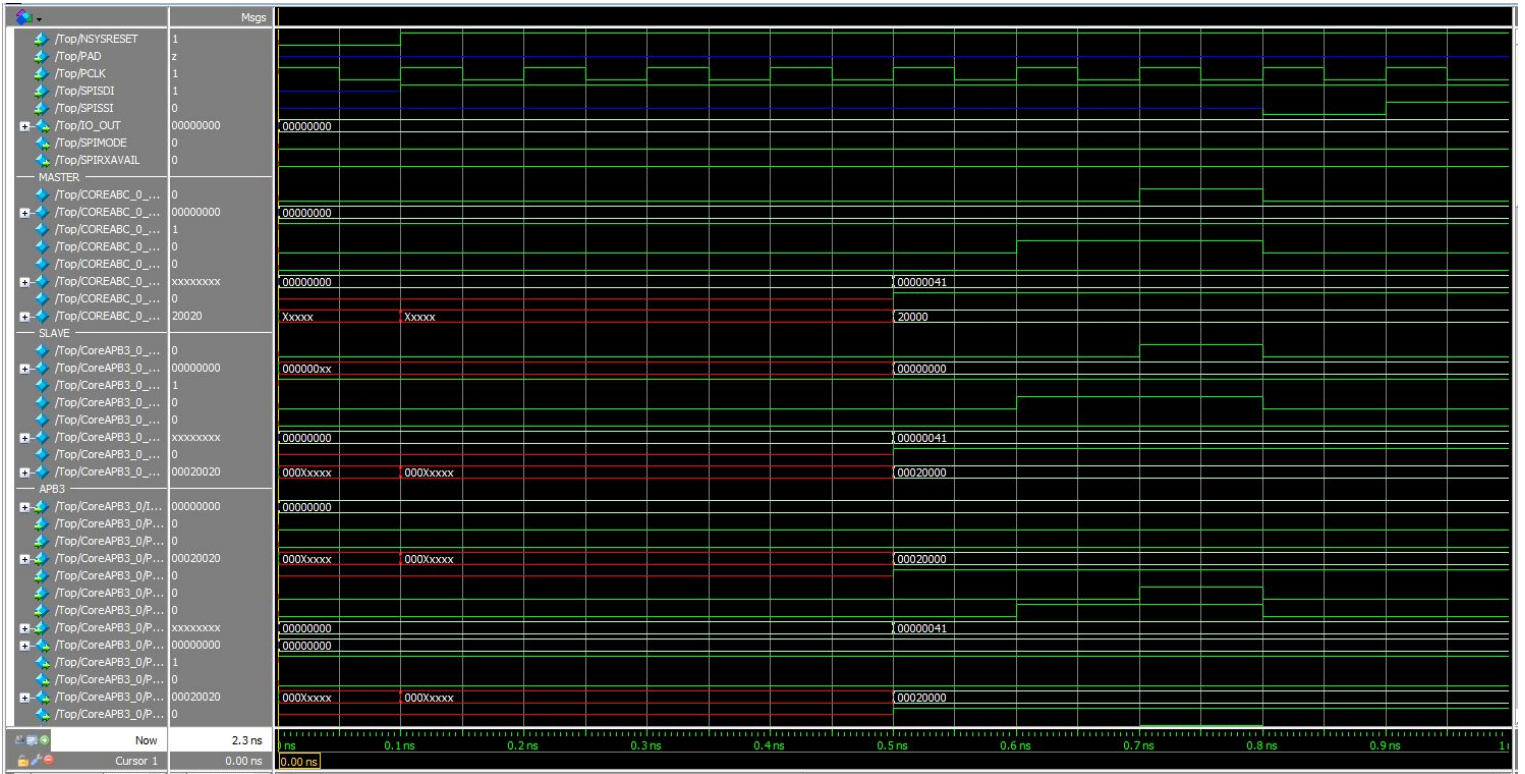
Testing

1. Once the canvas has been completed and the core configurations are done, hit the yellow **Generate Component** button on the top panel of the canvas.

- After completion, double click **Simulate** in the left hand panel. This will open ModelSim.



- This should be similar to what you want:



Replicating Software Neural Network Creation

Download Urbansound 8k dataset of sounds and sound classifications here:

<https://urbansounddataset.weebly.com/urbansound8k.html>

Now make sure your python version is up to date and that you have Librosa installed.

Run the following python script to generate your model (I use Anaconda to run my .py files):

```
def extract_features(file_name):  
  
    try:  
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')  
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)  
        mfccsscaled = np.mean(mfccs.T,axis=0)  
  
    except Exception as e:  
        print("Error encountered while parsing file: ", file)  
        return None  
  
    return mfccsscaled  
  
import librosa  
import numpy as np  
  
def extract_feature(file_name):  
  
    try:  
        audio_data, sample_rate = librosa.load(file_name, res_type='kaiser_fast')  
        mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=40)  
        mfccsscaled = np.mean(mfccs.T,axis=0)  
  
    except Exception as e:  
        print("Error encountered while parsing file: ", file)  
        return None, None  
  
    return np.array([mfccsscaled])  
  
def print_prediction(file_name):  
    prediction_feature = extract_feature(file_name)  
  
    predicted_vector = model.predict_classes(prediction_feature)  
    predicted_class = le.inverse_transform(predicted_vector)  
    print("The predicted class is:", predicted_class[0], '\n')  
  
    predicted_proba_vector = model.predict_proba(prediction_feature)  
    predicted_proba = predicted_proba_vector[0]  
    for i in range(len(predicted_proba)):  
        category = le.inverse_transform(np.array([i]))  
        print(category[0], "\t\t : ", format(predicted_proba[i], '.32f') )  
  
# Load various imports
```



```

import pandas as pd
import os

# Set the path to the full UrbanSound dataset
fulldatasetpath = '/Users/Jake/Desktop/senior design learning/SoundClassificationCode/UrbanSound8K/audio/'
metadatapath = 'UrbanSound8K/metadata/UrbanSound8k.csv'

metadata = pd.read_csv(metadatapath)

features = []

# Iterate through each sound file and extract the features
for index, row in metadata.iterrows():

    file_name = os.path.join(os.path.abspath(fulldatasetpath), 'fold'+str(row["fold"])+ '/', str(row["slice_file_name"]))
    class_label = row["class"]
    data = extract_features(file_name)
    features.append([data, class_label])

# Convert into a Panda dataframe
featuresdf = pd.DataFrame(features, columns=['feature', 'class_label'])

print('Finished feature extraction from ', len(featuresdf), ' files')

from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

# Convert features and corresponding classification labels into numpy arrays
X = np.array(featuresdf.feature.tolist())
y = np.array(featuresdf.class_label.tolist())

# Encode the classification labels
le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))

# split the dataset
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, yy, test_size=0.2, random_state = 42)

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.optimizers import Adam
from keras.utils import np_utils
from sklearn import metrics

num_rows = 40
num_columns = 1
num_channels = 1

num_labels = yy.shape[1]
filter_size = 2

# Construct model

model = Sequential()
model.add(Dense(256, input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(228))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels))
model.add(Activation('softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

# Display model architecture summary
model.summary()

# Calculate pre-training accuracy
score = model.evaluate(x_test, y_test, verbose=1)
accuracy = 100*score[1]

print("Pre-training accuracy: %.4f%%" % accuracy)

```



```

from keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 100
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/test_weights.best.basic_mlp.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(x_test, y_test), callbacks=[checkpointer], verbose=1)
model.save

duration = datetime.now() - start
print("Training completed in time: ", duration)

# Evaluating the model on the training and testing set
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])

#try a prediction
filefile = 'UrbanSound8K/audio/fold5/Durham at 245.wav'
print_prediction(filefile)

```

Running this script will take upwards of 15 minutes or more on a pc as this will train the model using the Urbansound dataset which contains over ~8,700 .wav files. The script also reports the training and testing accuracies, both of which should be very high. The script also takes in an input sound and runs a prediction on it. This is written as “filefile” at the end of the script. Feel free to upload your own sound as i did and see which classification it is most similar to. Here my clip of durham at 2:45 is most similar to children playing - it is just a sound of murmuring students studying, like children playing at a park. Also make sure that all of the paths are updated with your current paths rather than mine.

Once you have your trained model, you can quantize it to save space should you want to upload it to an embedded system, you can read more about this here:

https://www.tensorflow.org/lite/performance/post_training_quantization

Now you can use the python testing script from section 6.2 to test any .wav file you'd like to see what classification the neural network thinks your sound is.

If you would like to develop the C++ version of the testing script. You can access the Aquila github here (The actual website seems to be down for the moment):

<https://github.com/zsiciarz/aquila>

Once you compile the aquila examples, access the mfcc_calculation file and insert the C++ program in 6.2 in place of the current file and compile and run.

You can now replicate the software portion of this project.

II: ALTERNATIVE / INITIAL VERSIONS OF THE DESIGN

Version 1:

Version 1 had a single MSP430 connected to the Nano with a microphone where the FPGA would be accelerating the MFCC generation. This version was revised once we discovered that the FPGA did not have enough memory space to store all the intermediate data produced in MFCC

generation. This version was conceptualized before we researched the chips and found the specifications that the MSP430 and the Nano had.

Version 2:

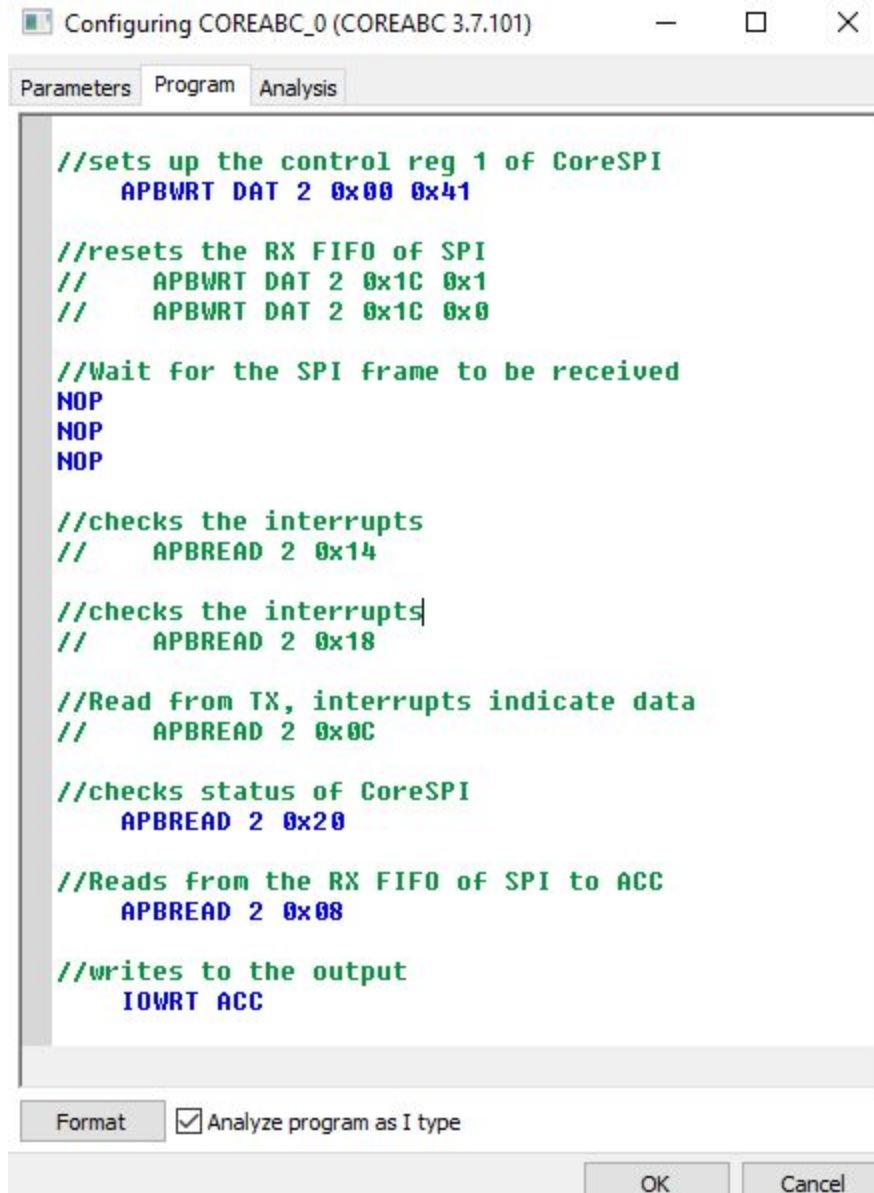
Version 2 had a single MSP430 with the Nano where the acceleration target was the matrix multiplication associated with inference. This version was revised once we discovered that a single MSP430 would not be able to store the entire software program and the intermediate data produced from the Nano. This version was discovered to be infeasible once we discovered the amount of C libraries required in audio recognition.

III: OTHER CONSIDERATIONS

A big thing we didn't consider until the second semester was the importance of making an informed decision about the FPGA we wanted to use. During semester one we put ourselves in a hole by thinking the first thing we needed to figure out was which FPGA we were going to use for our project. We didn't know a lot about FPGAs at the time and there is still a lot to learn but it occurred to us that we didn't know nearly enough about the FPGA we chose until after we chose it. If we were to do it all over again we would have waited until we knew more specifics about the software we wanted to run before making a decision about the FPGA we wanted to use.

IV: CODE (OPTIONAL)

Code for Core_ABC:



V: REFERENCES

Microsemi Tutorials

<https://www.microsemi.com/product-directory/libero-soc/5507-libero-soc-v11-9-archive#documents>

Efficient Processing of Deep Neural Networks: A Tutorial and Survey

<https://arxiv.org/pdf/1703.09039.pdf>

