iFPGA

Team sdmay20-38

Justin Sung - Embedded Systems Engineer
Zixuan Guo - Systems Diagram Expert
Jake Meiss - Electrical Engineer
Andrew Vogler - FPGA Design Engineer
Jake Tener - Software Technician

Client/Advisor: Dr. Henry Duwe

Project Vision

- What was the project attempting to accomplish?
    - To create a self-sustaining low-power system capable of carrying out computations
- Why?
    - Current battery production
    - Self-sustaining energy
- A very high level approach
    - Supply self-sustaining power for a low power FPGA
    - Use an external MCU to execute sound classification computation and accelerate a part of the software through an FPGA

Conceptual/Visual Sketch

- What we planned to design
  - A batteryless FPGA system capable of speeding up computations.
  - Performing Sound Classification on the Embedded System
- Targeted Clientele
  - Henry Duwe and his research assistants.
- What is Unique about the approach?
  - Designing a foundational model for any batteryless FPGA computation

Functional Requirements

- Batteryless and Data transmission off-chip
  - Power provided by means of RF Energy Harvesting
  - UART

- FPGA
  - Ability on accelerating the calculation for data from MCU on FPGA
  - Execute the sound classification on the MCU
  - There will be a checkpointing in the software that can allow the program execution pause and continue while power toggling

Non-Functional Requirements

- Performance & Compatibility & Usability
  - Voltage and Power thresholds
  - Accuracy of measurements
  - Compatible with other testbench

Project Plan

- Semester 1
  - Research and Develop Design
  - Choose Parts
  - Finalize Design
- Integration
  - Develop and order PCB
  - Develop and deploy software onto embedded system
  - Integrate PCB and embedded system
- Testing
  - Boot Sequence
  - Run Computation
  - Data Flow and Storage
- Finalize System
  - Provide useful and complete documentation

Risks & Mitigation

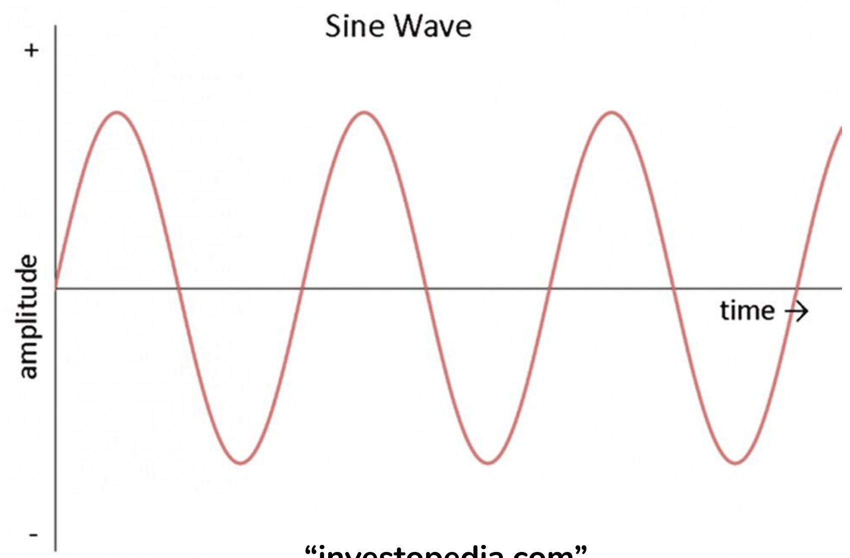
- Platform power consumption
  - The power requirements for the platform should be addressed by the power design
- Intermittent execution handling
  - Intermediate data will be lost upon FPGA power-down during computation.
  - Software checkpointing
- Broad project scope
  - Progress reviews at meetings

Software Design

Steps in the Software Process:

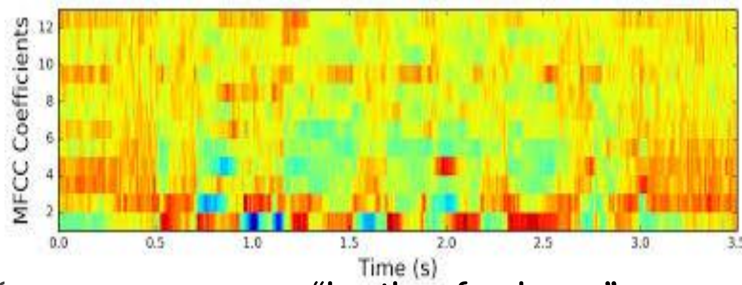
- Performing Sound Analysis (Sampling, MFCC Generation)
- Neural Network Creation and Training (Keras Model, Python Script)
- Develop Testing Script for input sound (Python)
- Quantize Model in preparation to upload to Embedded System (Tf Lite Model)
- Profile Testing Script to determine target process for acceleration
- Develop low-level C++ testing script to interface with Embedded System
- Test C++ script to determine accuracy in relation to the Python script
- Upload Model and testing script onto Embedded System

Sound Analysis



“investopedia.com”

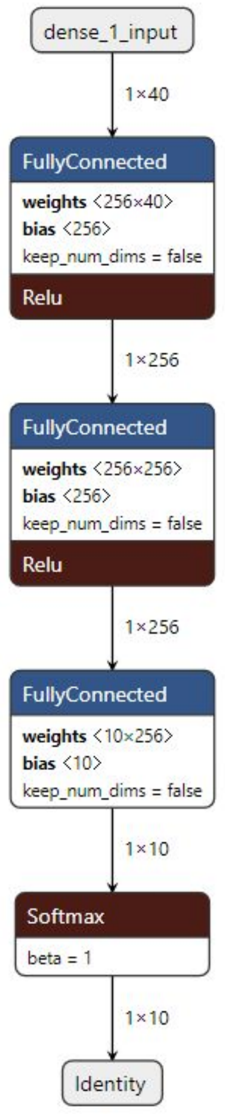
- Sampling
  - Take an analog sound and convert it into a digital array
  - Sounds are sampled at 22,050 times per second
  - Stored in a long array known as audio_data (~80,000 floats)
- Generate MFCC
  - Frame the audio data into 40 frames
  - Using Fourier transform, gather frequency of sound at each time T for its amplitude for each frame
  - Generate a value (MFCC Coefficient) to represent these 3 values
  - Combine values into an array that represents an MFCC for each of the 40 frames
  - Average each MFCC to one value and add it to a new array “Scaled MFCC”
- Scaled MFCC
  - Represents the MFCC of the entire sound, used in training and as input for classification
- Librosa (Python model) vs Aquila (Embedded System Application)



“haythamfayek.com”

Neural Network Process

- Audio Classification Model
  - Trained with UrbanSound 8k's dataset of 8,732 sounds
  - 10 Unique Classifications
  - Input Shape (1,40)
  - Output shape (1,10)
- Weights of the Model
  - The weights are trained by generating a MFCC spectrogram of each sound sample and pairing it with its classification
- Prototyped in Python on PC as a Keras Model
- Quantized into Tf Lite model
- Test Script takes an input .wav file and runs inference with the model
  - Outputs Classification



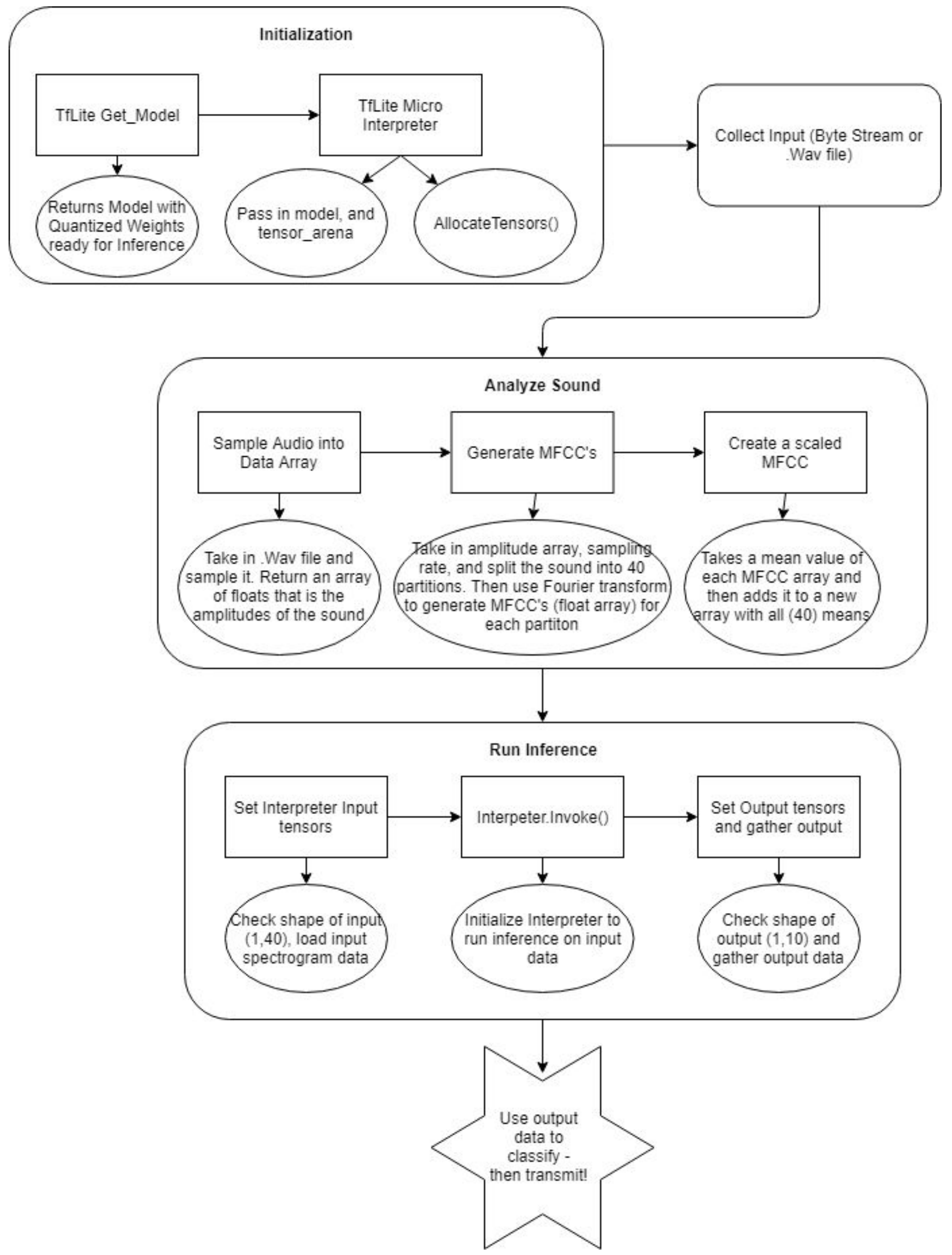
Uploading Process to Embedded System

Targeting Inference/Prediction on the FPGA and Analysis on the Microcontroller

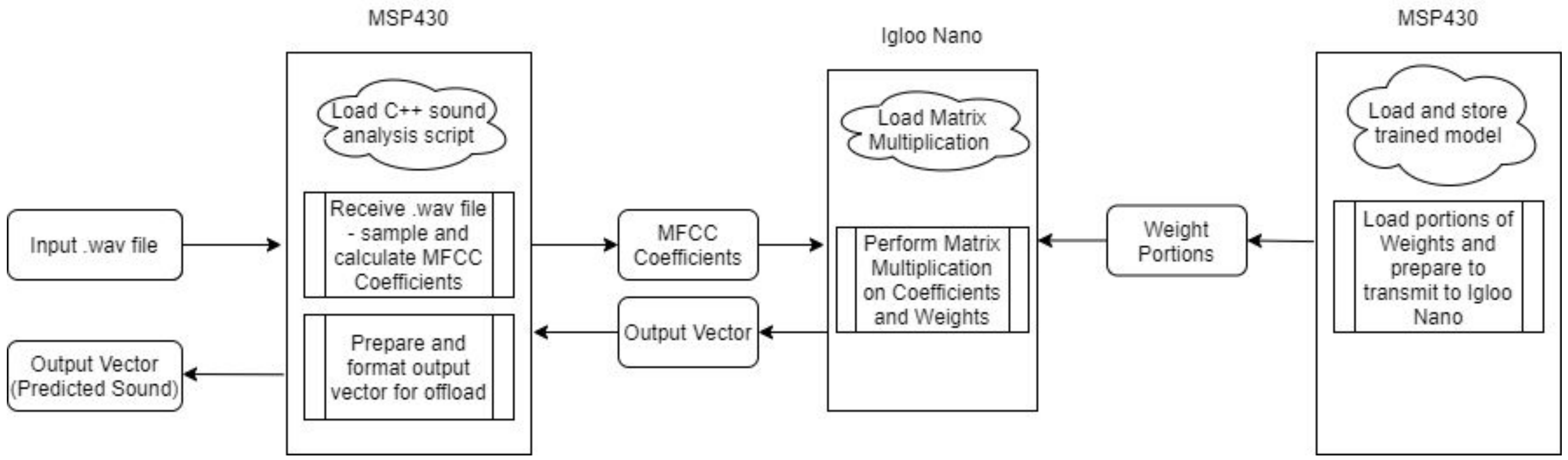
- Converting MFCC generation from Librosa method to C functions
    - Using Aquila sound analysis library
    - Sampling in integers, rather than floats, and even after conversion sampling values are slightly off
    - MFCC functions operate differently than Librosa as well, so MFCC values are also slightly off
- Covid-19 Implications
    - Due to the social distancing, our embedded system was not developed/tested enough to handle the software application, so we are unable to put the process into action, but the C++ testing script showed 50% accuracy on a given sound

"it.emcelettronica.com""

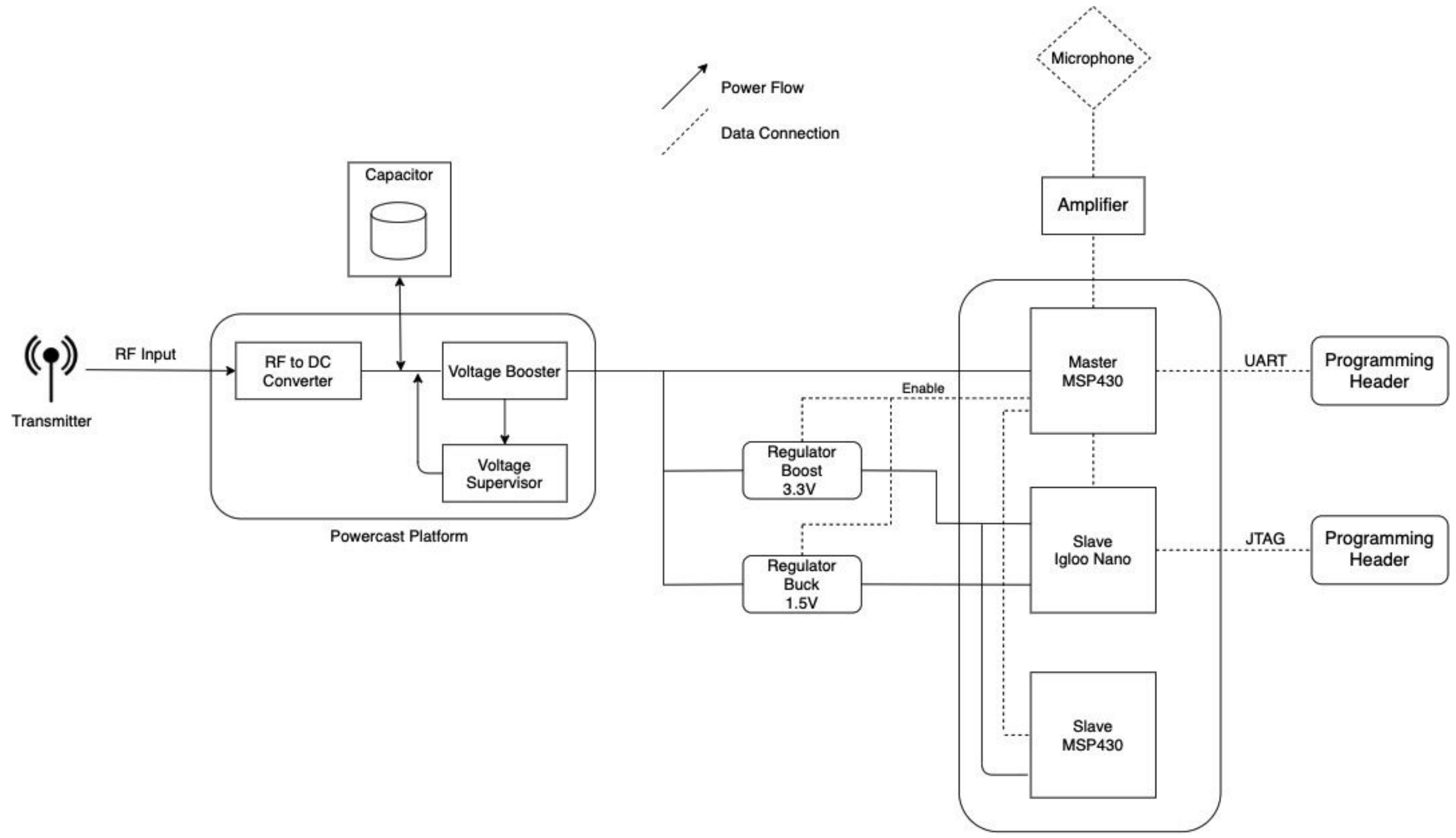
Software Design Flow


Initialization
TfLite Get_Model
TfLite Micro Interpreter
Returns Model with Quantized Weights ready for Inference
Pass in model, and tensor_arena
AllocateTensors()
Collect Input (Byte Stream or .Wav file)
Analyze Sound
Sample Audio into Data Array
Generate MFCC's
Create a scaled MFCC
Take in .Wav file and sample it. Return an array of floats that is the amplitudes of the sound
Take in amplitude array, sampling rate, and split the sound into 40 partitions. Then use Fourier transform to generate MFCC's (float array) for each partiton
Takes a mean value of each MFCC array and then adds it to a new array with all (40) means
Run Inference
Set Interpreter Input tensors
Interpeter.Invoke()
Set Output tensors and gather output
Check shape of input (1,40), load input spectrogram data
Initialize Interpreter to run inference on input data
Check shape of output (1,10) and gather output data
Use output data to classify - then transmit!

Intended Software Implementation Diagram

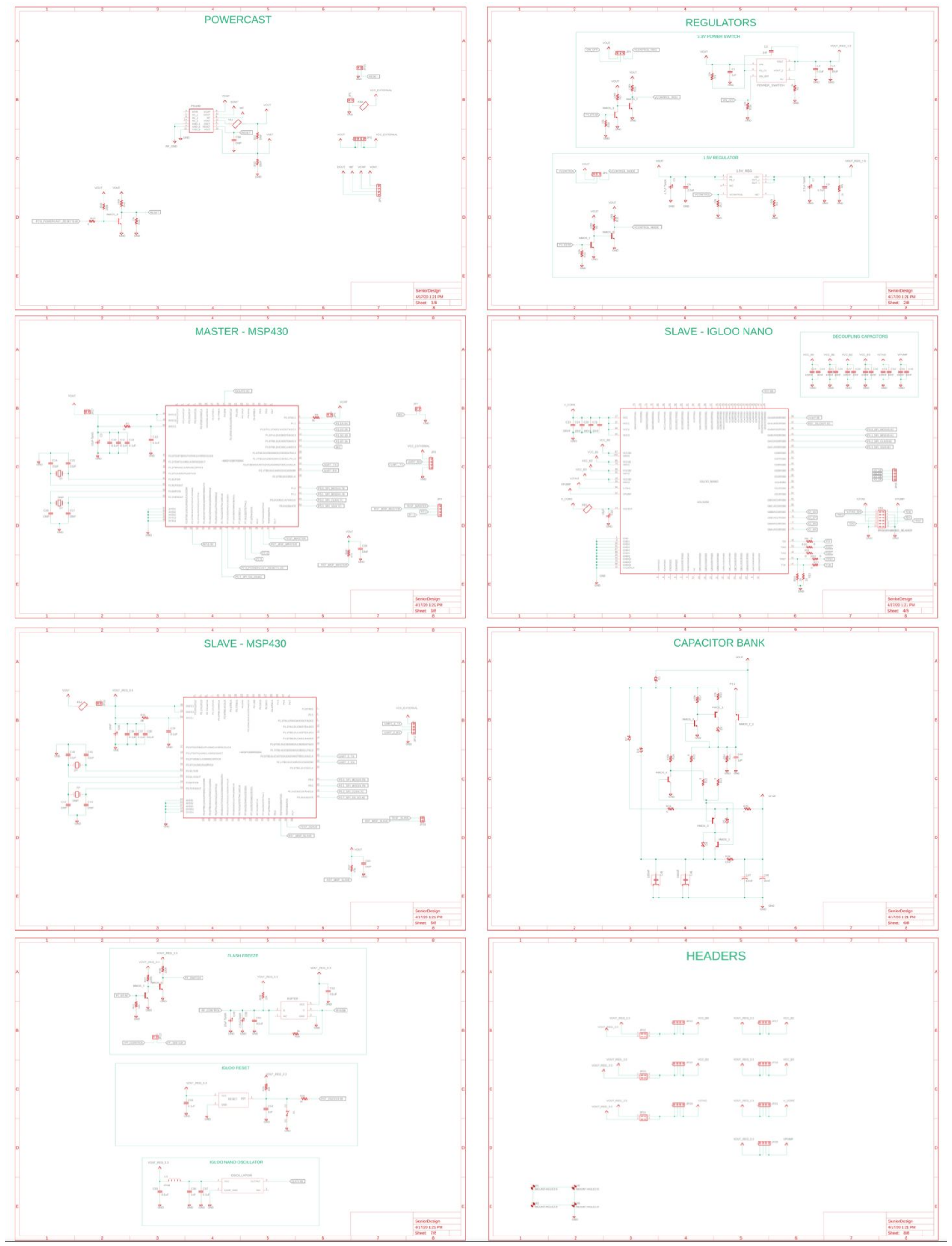

MSP430
Load C++ sound analysis script
Receive .wav file - sample and calculate MFCC Coefficients
Prepare and format output vector for offload
Input .wav file
Output Vector (Predicted Sound)
MFCC Coefficients
Output Vector
Igloo Nano
Load Matrix Multiplication
Perform Matrix Multiplication on Coefficients and Weights
Weight Portions
MSP430
Load and store trained model
Load portions of Weights and prepare to transmit to Igloo Nano

Power Management Diagram



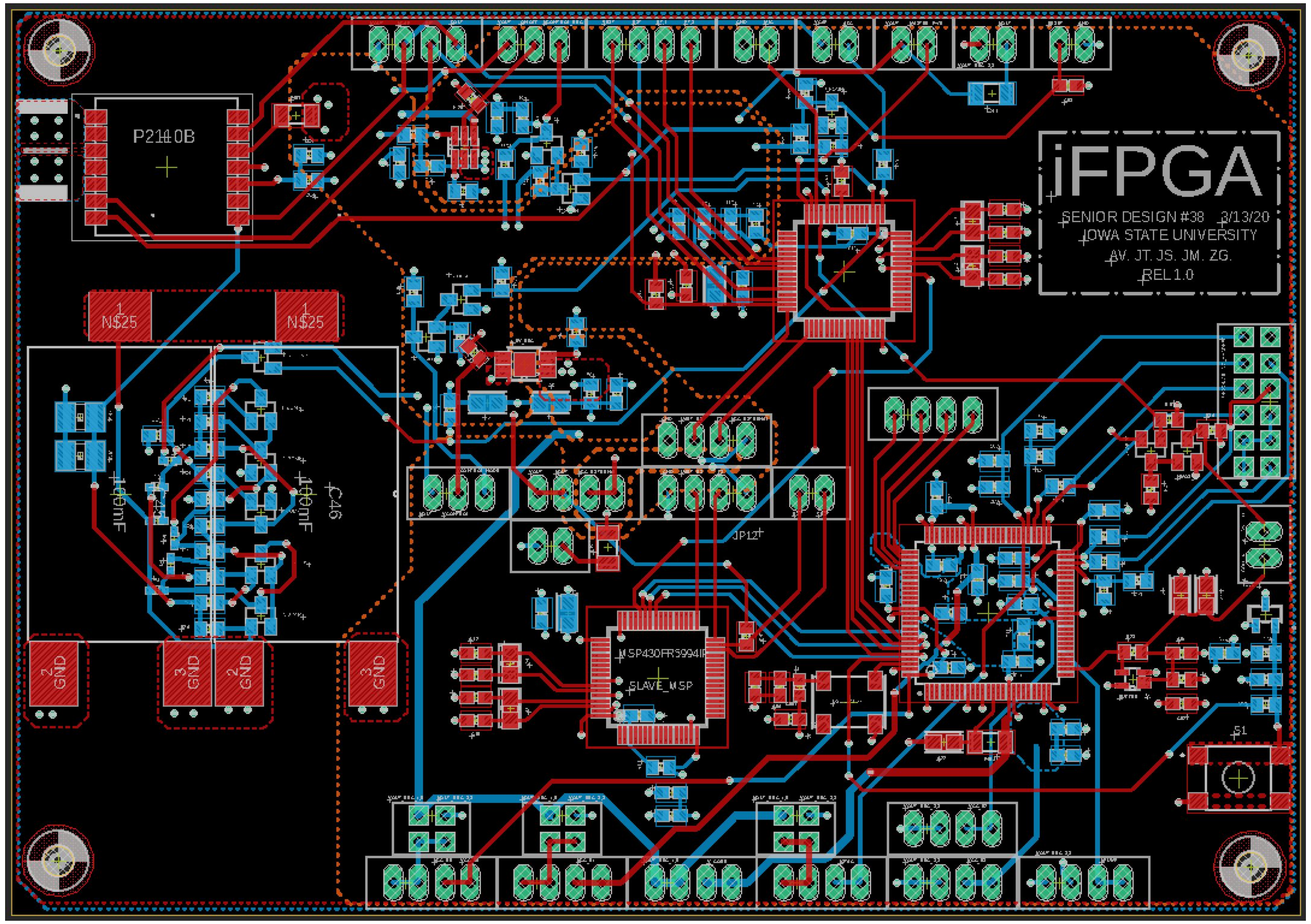
Electrical Schematics

- Powercast
- Microcontrollers
- FPGA
- Challenges
    - Control Circuits
        - Regulators
        - Flash Freeze
        - Reset
    - Programmable Capacitor Bank



Printed Circuit Board

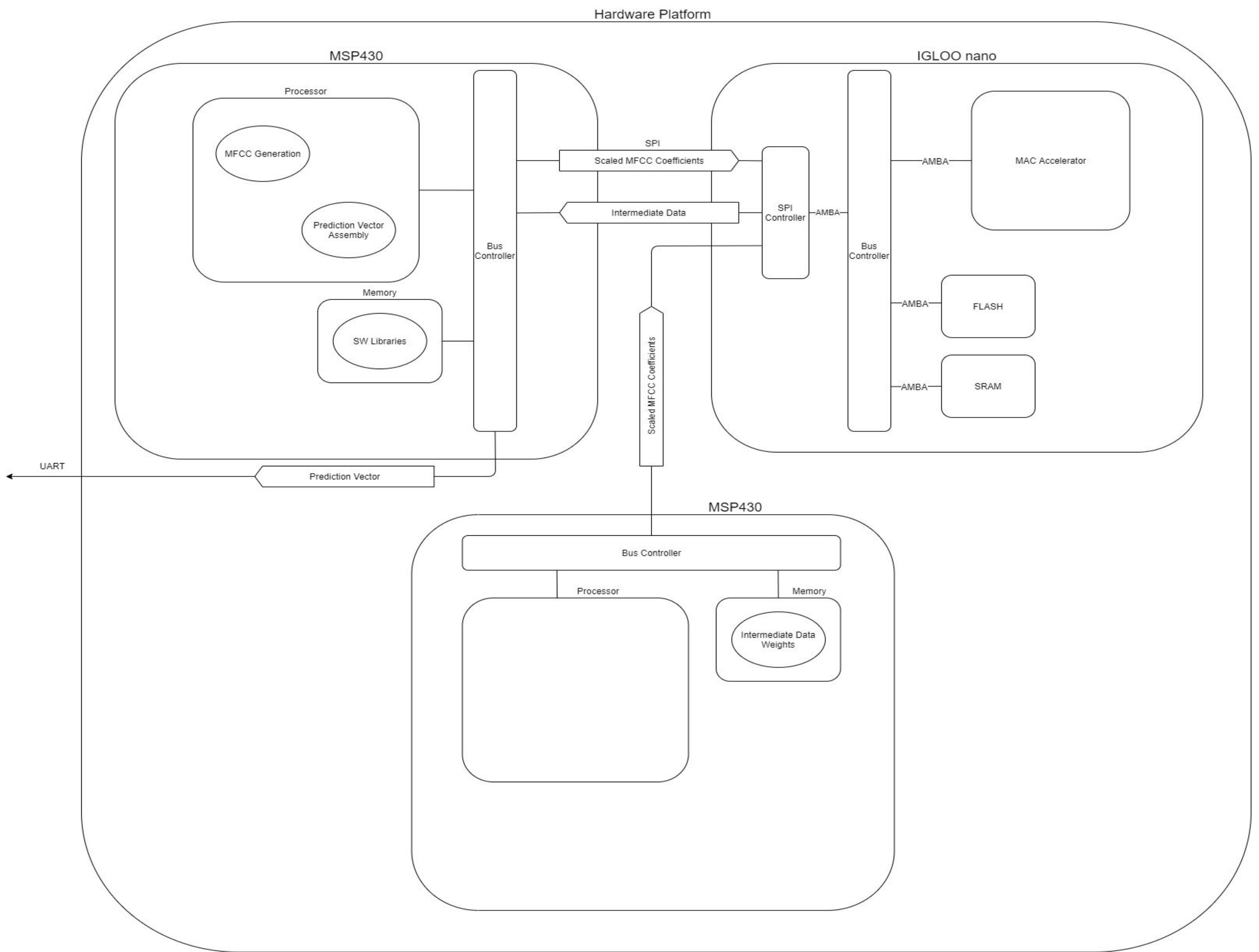
- 4 layer Board
- 3 power rails
- Challenges
  - RF Antenna Specs
  - Art, not a science
- PCB has been fabricated, awaiting population and testing

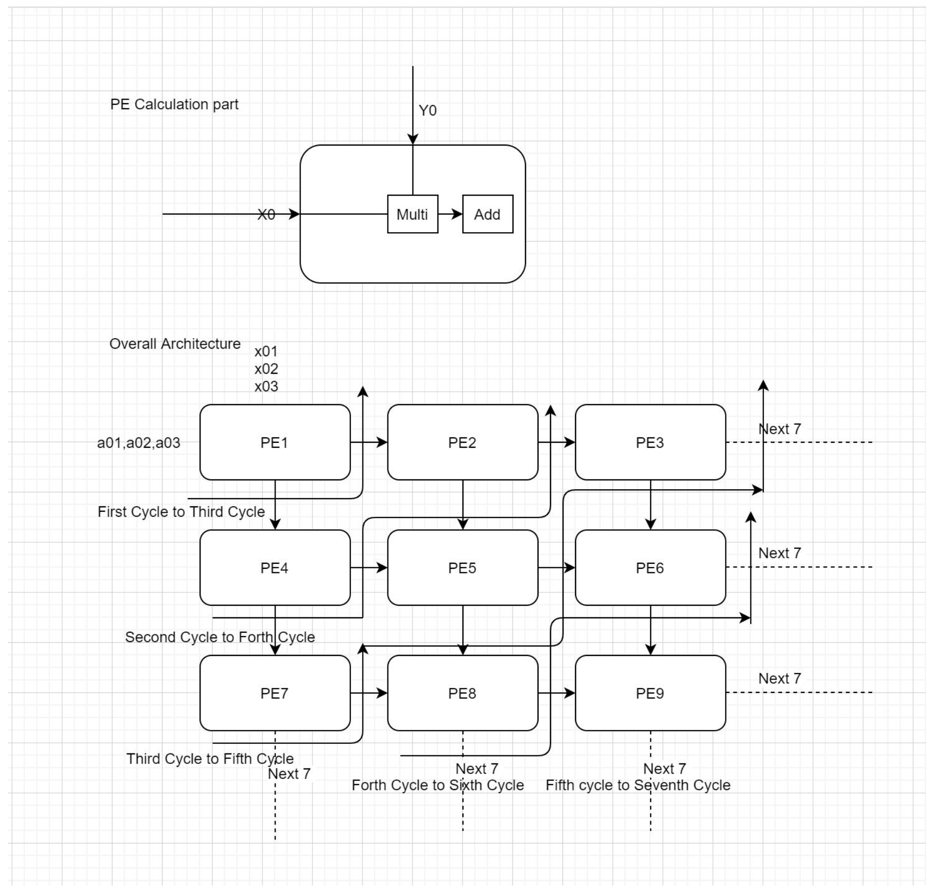


Embedded System Architecture

- First MSP430 handles most of the software execution
  - Processor handles software execution
  - Memory contains the program and necessary libraries
  - Data communication between the MSP430s and Nano
- Nano will work as a hardware accelerator.
  - MAC hardware targeting inference
  - Memory stores neural network weights and intermediate data
- Second MSP430 handles intermediate data and assembly.
  - Memory stores the all of the data produced from the Nano and the neural network weights

Embedded System Architecture Flow and MAC Design



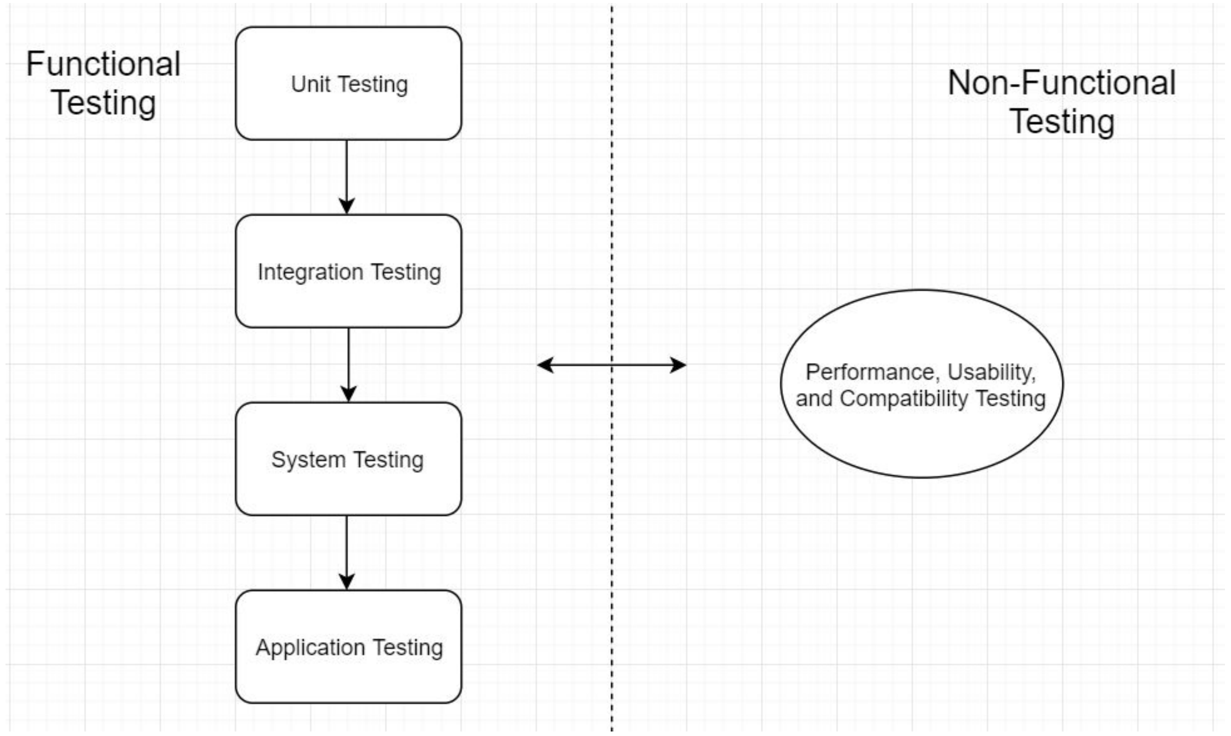


Prototype Implementations

- IGLOO Nano Example Projects
  - Interfacing projects
  - Read and write projects
- Sound Classification
  - Setup the software pipeline and passed a sound recording of Durham through the pipeline

Test Plan

- How is testing performed?
  - Software tests
  - Power analysis
  - Observing output
- Component/Unit Testing
  - Independent functionality of each component
  - White/Black box testing
- Interface/integration testing
  - Power supply
  - I/O between Microcontroller and FPGA
- System level testing/Acceptance testing
  - Mostly by Non-functional tests (Is enough power supplied? Does data flow as expected?)



Engineering Standards and Design Practices

- 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)
- Waterfall Model & Agile Sprints

Conclusion

- Lessons learned
  - Run tests early
  - Make plans early but prepare to have them change

- What we would have done differently
  - Choose specific FPGA after software scope is defined better