Ajinkya Joshi

Undergraduate Electrical and Computer Engineering Senior (BSEE)
Northeastern University - Boston, MA

Engineering Portfolio

Website: https://ajinkyaweb.vercel.app/
Github: https://github.com/ajinkyaj238
LinkedIn: https://www.linkedin.com/in/ajinkya-joshi-01b1a8246/
Email: ajinkyaj238@gmail.com
Tel: +1 (857) 264-6860

Last Updated: January 2nd, 2025

Summary and Interests

As an ECE major, I've been exposed to many different subfields that require the development of a wide range of skills. Over the years, I have worked on and developed numerous projects that have helped me build a strong foundation in both hardware and software design. This portfolio displays some of the work that might be of interest to you.

Due to the widespread nature of ECE, I've had to develop my ability to learn new skills and work on any project fast. And so have been able to excel working in interdisciplinary environments. This is reflected in my work experience and my completed/ongoing project work shown in this portfolio.

Overall, I have narrowed my interests to full time roles for FPGA/ASIC Engineering positions, Embedded/Firmware positions and Digital/Analog (Electronic Design) Engineering positions.

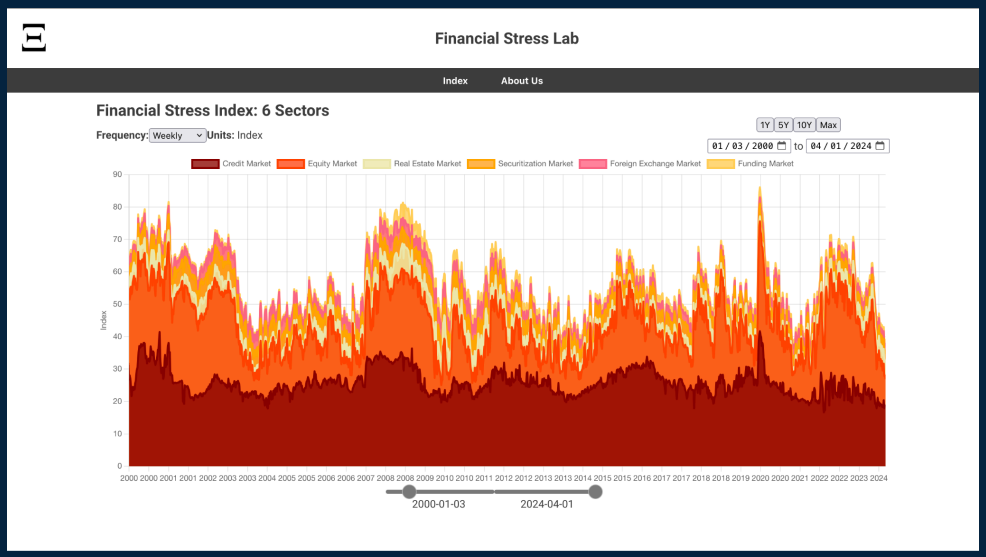
As a consequence of projects and classwork, I've developed somewhat of a can-do attitude. I understood that there is no technical skill that I cannot learn and apply to a project. It all comes down to the project that I work on and what would be the most efficient tool I can use to get the job done.

Table of Contents

Featured Work

Fullstack Financial Stress Webapp

Developed for the Financial Stress lab for Fintech and Quantitative Analytics.



Overview:

The aim was to develop a foundation for a full stack web app that displays the findings of work done on computing financial stress within the lab. The app was designed to be memory safe, be able to automate collection of raw data and be able to compute financial stress for upcoming days. The app currently displays research conducted on 6 sector financial stress. Work on the app was completed as a part of a COOP. Structure was designed to make it easy for the next developer to continue working on the site.

Technical Details:

- Frontend developed using React and ChartJs was used for the interactive graph.
- PostgreSQL was used as the database to store time series data, It was hosted locally.
- Backend was extended to use MongoDB as a cloud db option.
- Backend developed in python. Included automated scripts to collect specific time series' through the FRED api and manually gathered bloomberg data.
- Backend automated scripts apply statistical transformations and techniques with the aid of libraries such as Pandas, Numpy and Scipy.
- API endpoints were created using Flask to connect the react frontend and python backend allowing me to send a json of time series data.
- Memory safety techniques were employed by hiding api URLs using .env files.

- Console logging was done through python's logging library and all code was stored in an organizational github setup by me.

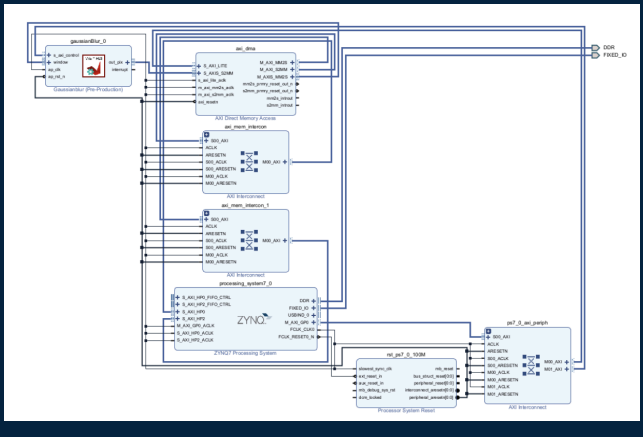
Reflection:
This web app was designed from the ground up by me through a process akin to building with legos. I started work on the postgreSQL database largely primarily to store, process and sift through a lot of financial time series data I collected while working on another project in this lab. I initially started off with Matlab to conduct statistical transformations but moved to python when I needed to automate collection of data via FRED. These scripts were extended to store data within the PostgreSQL and MongoDB databases. I developed these scripts into a backend through flask after which I started on the frontend with React and Javascript's various libraries.

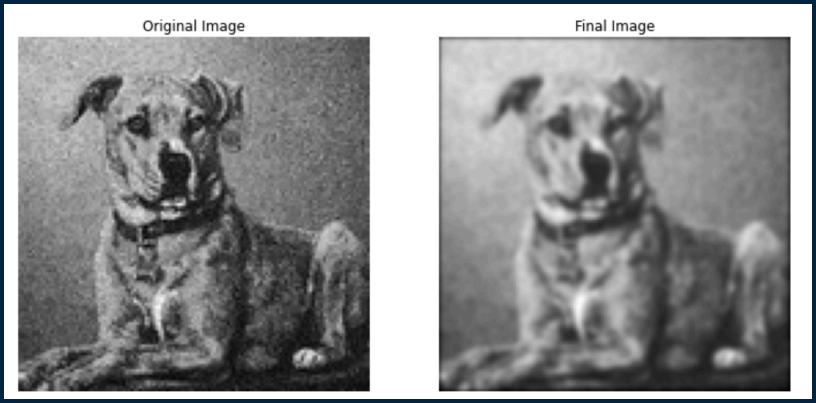
Going into this, I would have had no idea what to do or where to start. The whole process of building this app was identifying an issue (ex: how do I send data between the frontend and the backend?) researching solutions (create api endpoints and have the frontend request data from the backend) and applying tools to fix that issue (Use flask as a web framework for python). I had no experience with NoSQL databases, Javascript, web development, React, web hosting, console logging, using restful APIs or creating api endpoints. Within the span of 2 months I was able to take the loose python scripts I was working on into a foundational web app that is actively being worked on by the next COOP.

Undertaking this project made me more proactive and confident in my abilities to work on any task. I developed a can-do attitude to new problems. I applied the skills I learned in my lab into my own personal website.

Gaussian Blur on an FPGA

Learn more about the project: https://github.com/ajinkyaj238/Image_Filter/tree/main/Final





Overview:

The Gaussian Blur executed on the Xilinx PYNQ-Z2 board (ZYNQ-7000 SoC) is an image processing project that involves blurring an input image using the computing power of hardware and the data processing of software. The aim of this project is to denoise a grayscale image of varying sizes using the gaussian blur filter executed on hardware using a 5x5 gaussian blur kernel and observe a speedup in performance with the hardware design exploration.

The IP logic involved iterating through each pixel of the input image matrix; multiplying the overlapping areas between the kernel and the image. I referred to the overlapping area as the window. The operation between the kernel and the window is a weighted sum that outputs the final pixel value of the output image. The weights attached to the kernel determines what image filter the

The execution time for software was approximately 29.2 seconds. The ARM processor is running at a clock speed of 650MHz (ARM Cortex A9 on the ZYNQ-7000 SoC). This

is 6.5x higher than the ZYNQ-7000 FPGA's 100MHz. Therefore a relative speedup in the hardware execution would be computing a blur at a time less than 189.6 seconds. Any execution time below that would be considered a relative speedup.

The blur time taken by my hardware IP took 59.2 seconds. When taking into account clock speeds, the FPGA has a relative speedup of 3.2x. If this outcome was extrapolated, whereby the FPGA had the same clock speed as the processor of 650MHz, the total time at that clock speed would be 9.13 seconds. Click on this sentence, or the URL above, for more details about the project.

Technical Details:

- Project was divided into two halves - the software portion and the hardware FPGA IP.
- Software portion involved converting an image into grayscale using OpenCV for each pixel to be processed as an integer value between 0-255.
- Matrix was flattened and transferred to the IP using the AXI-Lite interface for the final image.
- AXI-Stream successfully worked for smaller images. However, lack of physical space on the FPGA required the use of AXI-Lite instead.
- Influential IP design methods to improve performance involved:
    - Converting all ints to 8 bit ints; lowering space complexity by a fourth.
    - Rewriting the output pixel values onto the space used up by the input. If the kernel does not overlap with the row of the pixel, only then will the pixel be written onto the matrix.
    - Weights were converted to integers by multiplying by 1000 to avoid floating point operations.
- The IP was programmed using Vitis (gui) via C++ HLS. The IP was transferred to Vivado before converting it to bitstream for the ZYNQ-7000's FPGA's LUTs.

Reflection:

In my second FPGA project, I explored the design paradigm of programming, testing, and deploying filters directly on FPGAs. Unlike my first project, where FPGAs were used as a platform to validate designs intended for CPUs, this project involved implementing filters and logic on FPGAs, either for fabricating ASICs or as the final IP in the design cycle.

This project not only taught me the technicalities behind HLS, vitis and AXI interfaces but also taught me the importance of accounting for the physical limitations of the target device and optimizing designs for peak performance. I discovered the advantages of creating custom hardware IPs and recognized the computational efficiency FPGAs offer. This project highlighted how the simplicity of filters enables hardware IPs to execute operations more efficiently than CPUs (relative to their clock

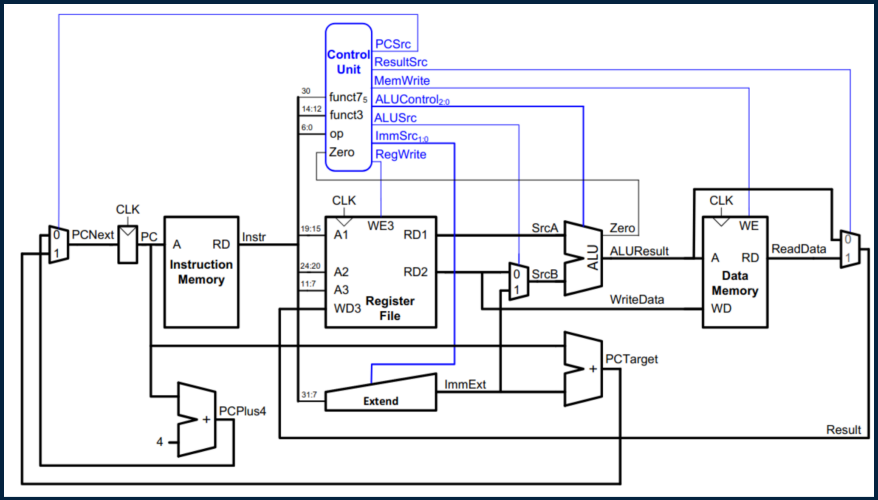
speeds). Designing for Hardware tends to be more efficient in executing operations in contrast to CPU's that deal with loading and storing data as a majority of the instructions in its operation and data storage is determined by the ISA of the CPU. As a result I developed a nuanced understanding of when to use FPGAs over MCUs, based on performance and application requirements.

Overall, I was surprisingly enthusiastic about my project, and no matter how many times my design failed my tests, I immediately knew where to debug. This project allowed me to think through a hardware first perspective that will prove immensely useful when I work in the industry.

On further research into IP filters, I found out that apple uses a selective upscale filter on the apple vision pro. To elaborate, the filter only upscales the area where the eye is directly looking at. This gives the effect essentially emulating a peripheral vision and saving on time and energy needed to render the entire image.

Single Cycle Risc-V Processor on an FPGA

Learn more about the project: https://github.com/ajinkyaj238/RISC-V_Processor-



Overview:

This project involved creating a single cycle RISC-V processor on an FPGA. The aim of this project was to teach me how a processor operates, how to write logic using SystemVerilog on Vivado and to implement designs on the ZYNQ-7000 FPGA. This project involved constructing different parts of the processor by either building it by scratch or by using IP blocks. The design was then simulated on the PYNQ-Z2 Board. The Processor was programmed using RISC-V Assembly to do basic arithmetic operations.

Technical Details:

- The IP was made up to specification of a single cycle RISC-V processor.
- Behavioural and structural SystemVerilog code was used to make the ALU, registerfile, instruction decoder and program counter. Datamemory was imported from Vivado. The logic blocks were connected to a topfile.
- Testbenches ensured proper functionality of all combinational and sequential logic units.

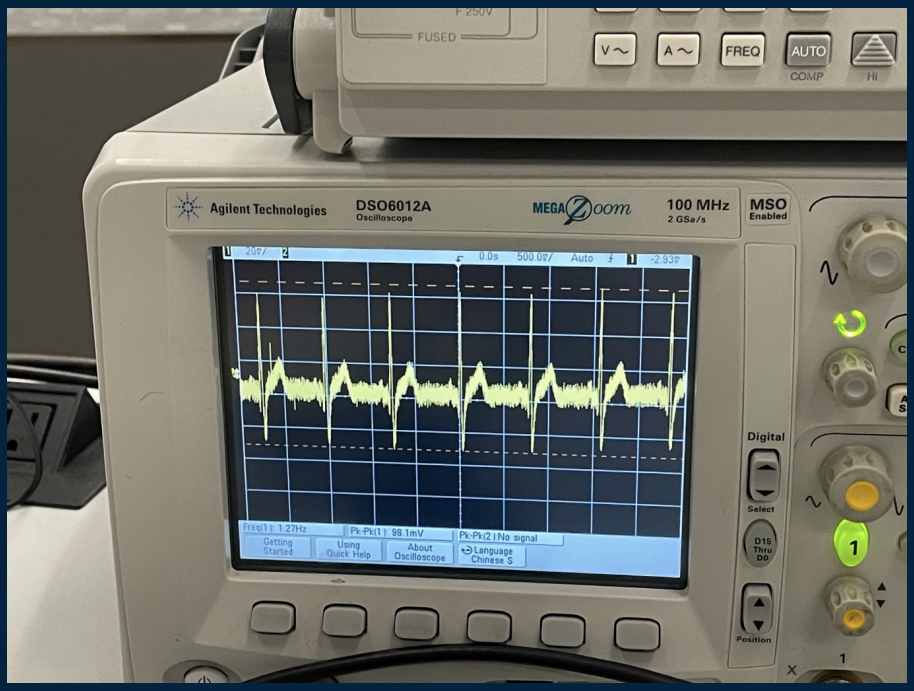
Reflection:

The mystery behind how computers worked is the primary reason I chose ECE as my major. This project cleared up most foundational level questions I had about how processors generally worked. Quite surprisingly, most instructions executed by my processor (or any processor) has to do with loading, storing data or instructions in registers and the datamemory. In fact, the ALU is used less for operations that the user is interested in (such as outputting the answer of an arithmetic operation) and more for operations regarding retrieving data (incrementing the program counter)

This project taught me a lot about SystemVerilog HDL design. I went from creating behavioural logic for the ALU all the way to structural logic by integrating muxes in my topfile. I also learned how to use an FPGA as a prototyping tool for creating logic that can later be used to fab processors. I also learned a lot about the RISC processor architectures whether it be RISC-V or ARM; moreover, I learned of the benefits and limitations that RISC has over CISC ISAs.

This project did leave me with a lot of different questions on cpu design. I went on to investigate multicycle RISC-V designs. This led me down a computer architecture rabbit hole where I learned of the intricacies of pipelining instructions, how branch prediction is used to resolve loops, and different types of caches to save data faster. Along with many techniques architects use to make processors.

Electrocardiogram System



Overview:
This project involved building an electrocardiogram, to acquire signals from an electrode attached to a human, to clean up the signal and observe it on an oscilloscope. More specifically, the aim was to build a circuit that acquire input signals from the electrodes using the instrumental amplifier, and filter out the high and low frequency signals and amplify the overall signal using the operational amplifiers, observe the output through the oscilloscope and electronically filter and graph the ECG via Matlab.

Technical Details:

- Filter bandpass was chosen to be between 0.1Hz to 200Hz.
    - Originally a range between 0.1Hz to 20Hz for a heart rate was considered (6 - 1200 BPM) even though 1200 BPM is impossible for a human to reach.
    - Eventually 200Hz was considered as there was no real problem when it came to sampling way higher than needed.
    - Practically, there was no need to filter high frequencies. It was only added to get rid of high frequency interference from EM signals.
- The AD627 instrumental amplifier was used to acquire the signal from the electrodes. Amplifier was set up to achieve a signal gain of 25.
- High pass filter built with the LT1490 op-amp was set up to have a gain of 2 and have a cutoff frequency of less than 0.1Hz. Low pass filter was also built

with the LT1490 op-amp and was setup to have unitary gain with a cutoff of greater than 200Hz
- Signal was observed via an oscilloscope before being exported via an ADC to Matlab where a FIR filter was applied to clean up the final PQRST heart wave.

Reflection:
As a final project for my circuits and signals class, this project taught me a lot about electronic design, acquiring signals, filtering via hardware and MATLAB and how to observe signals on an oscilloscope. A lot of forethrough went into choosing each component. Specific resistor values were chosen to achieve a specific gain and specific capacitor values to affect the cutoff frequency. The overall system gain was chosen to be 50 after trial and error in observing it through an oscilloscope. This project proved to be the culmination of everything I had learned up till that point in circuit theory.

Future Work

Digitally Controlled Analog Synthesizer

Current Overview:

This device is being built as a part of my final year capstone project. The aim is to build an analog synthesizer that can have its voltage controlled oscillator (VCO) parameters directly controlled by a processor (raspberry pi). The synth generates its signal via a VCO. Voltage controlled filters (VCF) are used to shape the signal to form a particular sound. The raspberry pi communicates with the circuit using I2C, SPI and GPIO signals. Output signal of the VCO-VCF circuit feeds into the pi to observe and view the signal. Motorised nobs communicate directly to the pi as a means of tweaking the ADSR (Attach, decay, sustain and release) values to generate different sounds. Plans include interfacing the system with MIDI to allow it to be extended to other peripherals.