Project Portfolio

Hello, I'm David Young, a senior at Washington University in St. Louis, pursuing a Master's in Electrical Engineering alongside a dual degree in Liberal Arts Engineering from Wheaton College, IL. My passions lie in analog and digital circuit design, controls, and embedded systems. I aspire to leverage my expertise to develop advanced circuits and innovative solutions for organizations applying technology to diverse challenges.

Linkedin:
www.linkedin.com/in/david-young-27808hi

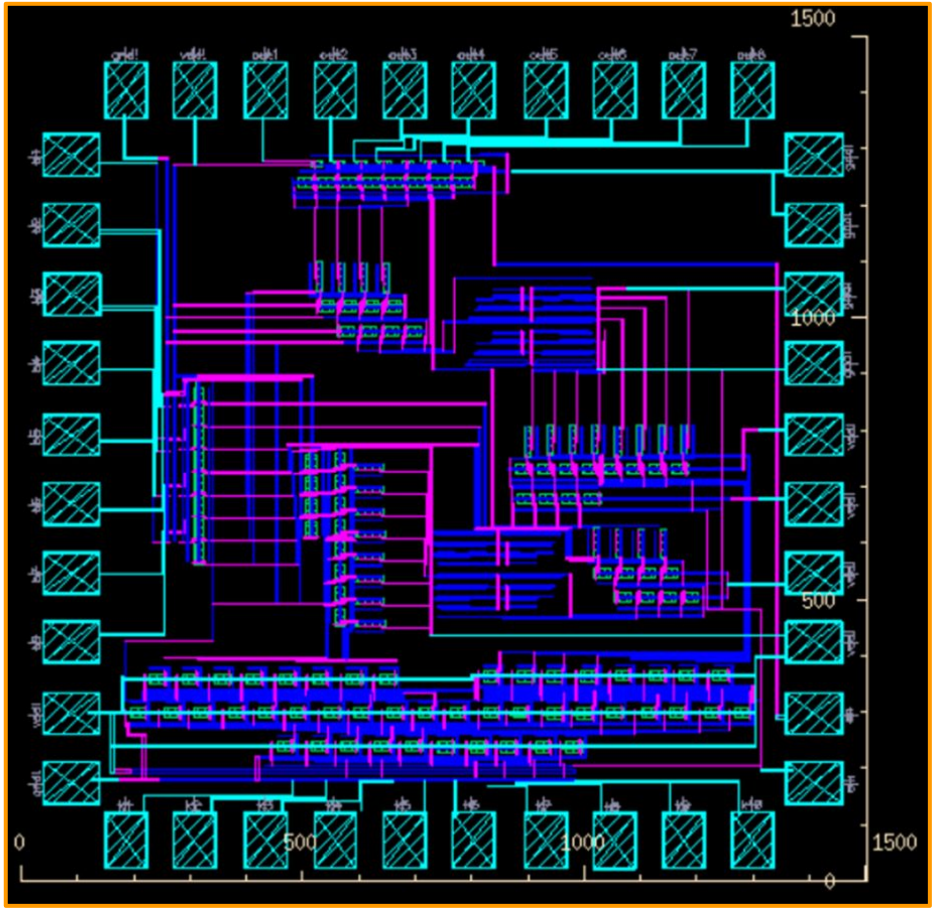
Designed, laid out, and performed timing analysis of a digital IC chip in Cadence Virtuoso that implements Simplified DES (SDES) to encrypt an 8-bit input using a 10-bit key.

Objective:
In a team of three, design, layout, simulate, and perform timing analysis of a digital chip that uses Simplified Data Encryption Standard (SDES). The final layout must fit within a 1.5mm x 1.5mm area constraint.

Process:

- Designed CMOS-level logic gates including NOT, AND, NAND, XOR, 2-to-1 MUX, and D Flip-Flops.
- Constructed higher-level SDES functional blocks such as IP, fk1, fk2, key generation, S-boxes, and inverse permutation using these gates.
- Created custom transistor-level schematics and layouts for each block using standard cell design practices.
- Simulated the full SDES chip using a 10 MHz clock and a 2.5 MHz enable signal.
- Verified the functionality and timing performance of the circuit through transient simulation.
- Ensured all components and the full chip layout adhered to the 1.5 mm × 1.5 mm area requirement.

Figure 1 - Chip Layout. Refer to project report on Github to see details on circuit schematic and layout.



Outcomes:
Our chip output the correct binary sequence for an assigned input and key. The final layout fit within the specifications.

Designed and implemented a scalable greenhouse Irrigation system that uses an ESP32 and Raspberry Pi to sense moisture and automate watering.

Objective:

Developed a scalable greenhouse irrigation system that uses wireless moisture sensors, automated valves, and a Raspberry Pi-based controller to deliver water precisely when needed. The system aims to improve plant health, reduce manual labor, and support future remote access via GUI. This design project was completed with two others.

Process:

Built and iterated through 3 working versions:

- **V1:** Wired prototype with moisture sensors and basic valve control.
- **V2:** Introduced wireless moisture sensing via ESP32 and MQTT protocol.
- **V3**: Added multi-valve watering and time division multiplexing for moisture sensing, real-time monitoring, and GUI framework.

Calibrated sensors, designed proportional control logic, and soldered final hardware on perfboards.

Outcomes:

- Sustained tomato plants at the target **80% soil moisture** over a week.
- Enabled remote moisture tracking and modular expansion (1 sensor/valve per plant).
- Demonstrated proof-of-concept for automated irrigation with GUI groundwork in place for future deployment.

Figure 1 - System Watering and Sensing the Moisture of 3 tomato plants.

Designed and simulated a cascaded control system for a drone using a linearized state-space model derived from a nonlinear dynamics framework. The goal was to stabilize the drone’s motion in 3D space using force and moment control.

Process:

- Derived trim conditions and linearized the nonlinear drone model to obtain A and B matrices.
- Verified system controllability.
- Designed four cascaded controllers (Z, L, M, N) to manage vertical position, roll, pitch, and yaw.
- Tuned gains based on desired closed-loop bandwidth and sampling rates (100 Hz for force, 500 Hz for moment loops).
- Simulated full system in Simulink and validated controller performance via step responses.

Outcomes:

- The linearized drone maintained stable responses in all axes with proper convergence to step commands.
- Cascaded control architecture functioned as expected with each controller providing smooth transitions.
- Successfully implemented mixer logic to translate control outputs to motor-level PWM signals.
- Gained strong understanding of cascaded control design despite initial unfamiliarity.

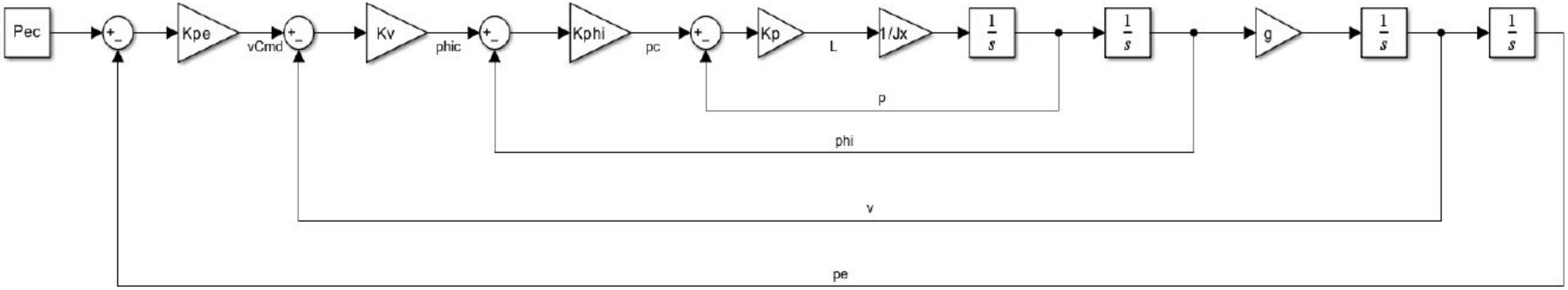


Figure 1 - L (Roll Moment) Cascaded Controller block diagram

Designed and Simulated an Instruction Cache using SystemVerilog, verifying functionality through a testbench in the Intel Quartus Prime environment.

Objective:
Design, code and simulate a 32-bit read-only instruction cache (as seen in **Figure 1**[1]) in System Verilog. The cache should be direct-mapped with a capacity of 128 bytes.

Process:
Modularized the cache into combinational logic and sequential logic, then verified its functionality through the use of a testbench and waveform analysis in the Intel Quartus environment.

Outcomes:
Developed an instruction cache using SystemVerilog capable of delivering instructions with a period of 100 ps, assuming that instructions from the main memory are returned in one clock cycle.

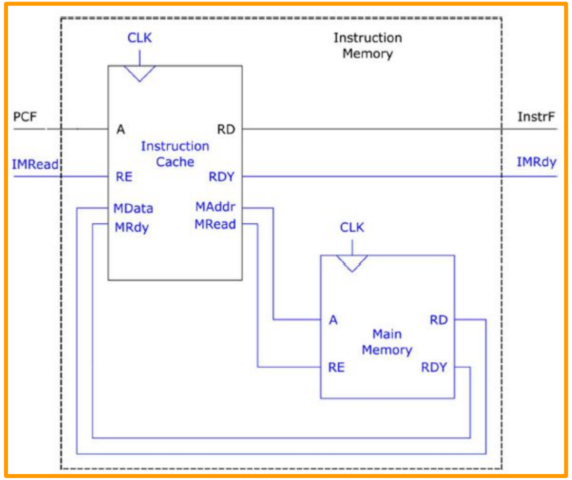


Figure 1 - Instruction Memory Overall Design

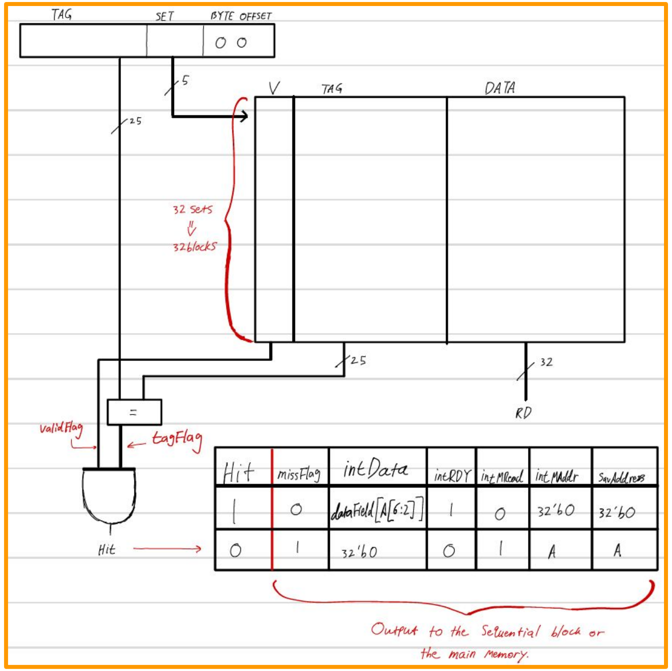


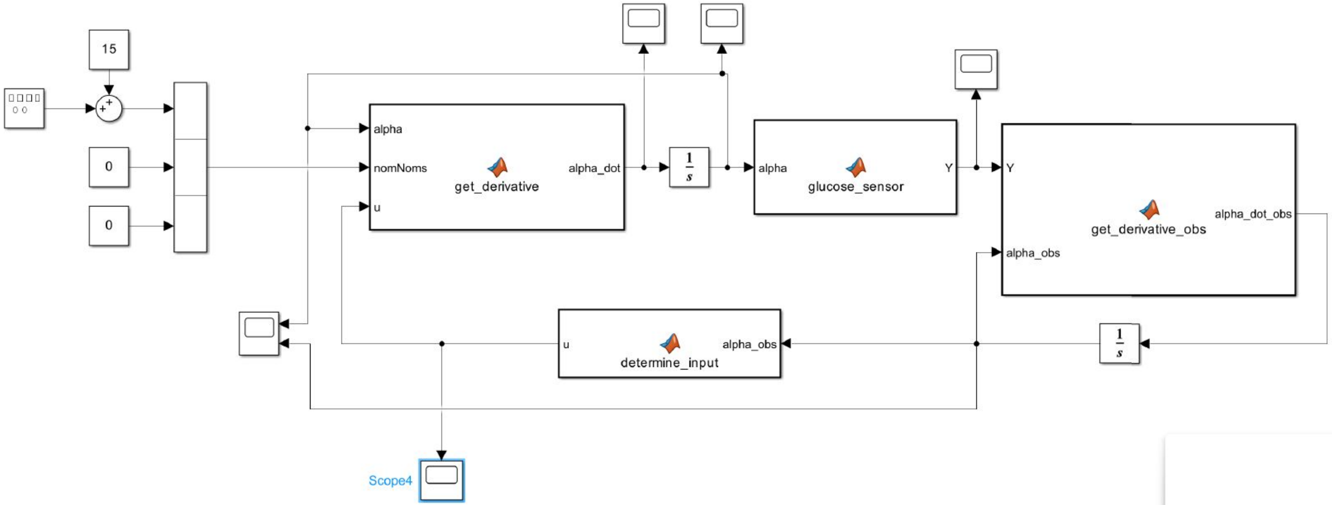
Figure 2 - Instruction Cache Combinational Logic

Designed a controller for an insulin pump through the use of linearization techniques, state feedback, and state observer.

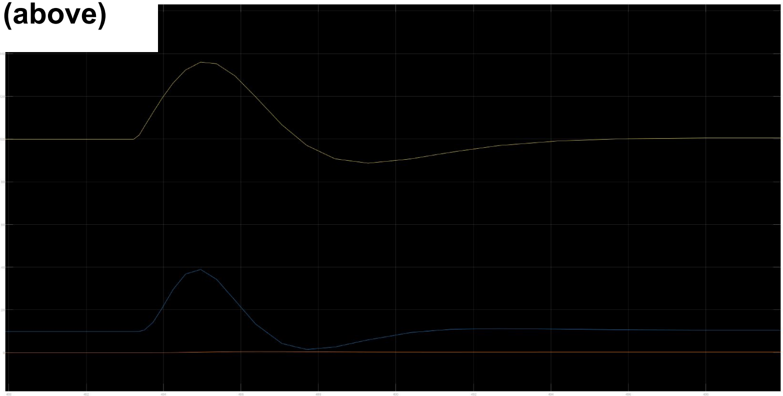
Objective:
Collaborate with team member(s) to design a controller that regulates the glucose level of the system to a specific level while accounting for disturbances to the glucose level (meals). Use MATLAB to design and simulate the controller.

Process:
We linearized the system about the baseline values of glucose and insulin. Then, we determined our state feedback and observer gain vectors.

Outcomes:
The implemented controller achieved a gain margin of 25% and an infinite phase margin, as determined by the Nyquist Stability Criterion. With this design, the blood glucose level returned to the specified baseline value within 15 minutes of a meal, exhibiting minimal overshoot.



Simulink Block Diagram of the system (above)



System Plot with Controller (right): y-axis (mU/dL), x-axis (sec.), Yellow - Glucose level (Baseline around 100 mU/dL), and Dark Blue - Insulin (mU/dL)

Trained a model using the least-square approximation (LSA) to identify a handwritten zero from a handwritten non-zero integer.

Objective:

Use MATLAB to train a model using the LSA to identify a zero and a non-zero integer from the “MNIST dataset of handwritten digits” [2]. Calculate the error rate, false positive, and false negative rates.

Process:

Imported images from the MNIST dataset into MATLAB, obtaining matrix A and vector y. We solved for **θ** using the LSA as seen in **Figure 1**. We applied our trained model (**θ**) to new test images and used the equation in **Figure 2** to create our own label vector ŷ which was compared to the actual label vector y to calculate error.

Outcomes:

When we used 5000 images to train our model, our error rate, false positive, and false negative rates were 1.94%, 1.15%, and 9.39%. When we used only the first 500 images to train our model, our error rate, false positive, and false negative rates were 22.72%, 20.64%, and 43.26% which was expected as we used fewer images to train the model. **Figure 3** shows the weights of each feature when using 5000 images to train the model.

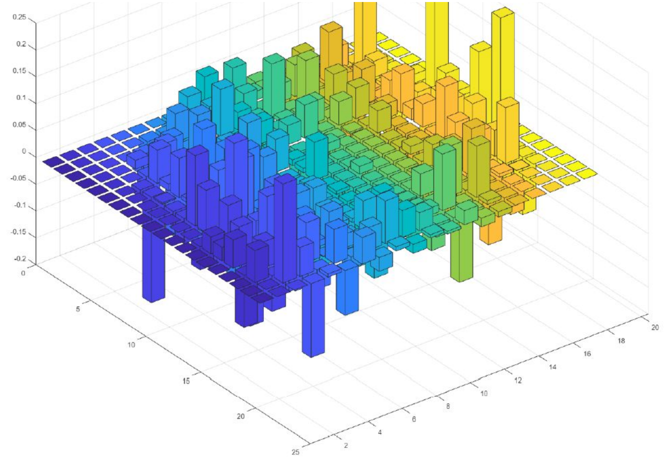
$$min||A\theta - y||^2$$

Figure 1 (above) - Least-Square Approximation (LSA), where we want to minimize the square difference between Aθ and y.

$$\hat{y} = \widetilde{f}(x) = sign(\theta_1 \cdot f_1(x) + \cdots + \theta_M \cdot f_M(x))$$

Figure 2 (above) - Equation and vector used to determine if the model identified a zero or non-zero integer. A positive signed ŷ means the model predicted a zero while a negative sign means a non-zero prediction. ŷ_i is the prediction for the i-th image.

Figure 2 (right) - 3-D graph of the θ vector. Taller bars mean that the pixel has a greater influence in determining if the model “sees” a zero or a non-zero integer.



Designed and Simulated a digital circuit in VHDL to compute and display the four roots of a quartic equation on an FPGA, utilizing time-multiplexed 7-segment displays and the Xilinx Vivado environment.

Objective:

Use the Xilinx Vivado environment to implement the schematic in **Figure 1** [3] in VHDL. This involved coding the roots capture process, multiplexer (MUX), N-bit counter, and decoder to display the four roots of the quartic equation:

4,194,304 - 491,520x + 17,920x2 - 240x3 + x4 =0

in hexadecimal on the 7-segment display.

Process:

The roots capture process utilized a difference engine to compute the four roots of the quartic equation in hexadecimal. The MUX was responsible for selecting which hexadecimal digit to display at any given time. The decoder determined which of the eight 7-segment displays was activated. Additionally, the three most significant bis (MSBs) of the N-bit counter was used to cycle through the select signals for the MUX and decoder, enabling time-multiplexed display functionality.

Outcomes:

The final implementation successfully displayed the four roots of the quartic equation in hexadecimal format on the 7-segment display. The fourth root was computed and displayed within **2350 ns.**

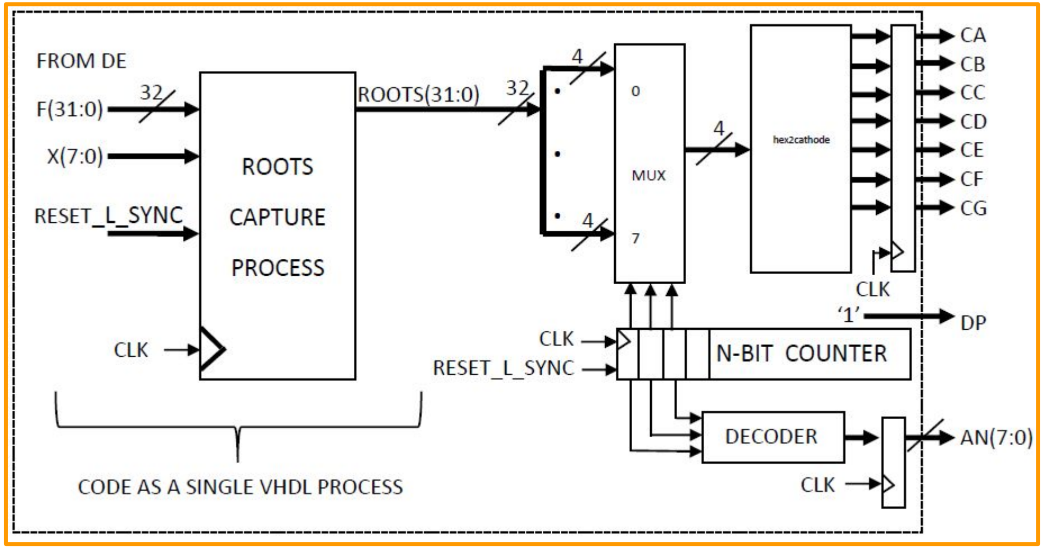


Figure 1 - Schematic of the circuitry to be implemented onto the FPGA.

References:

[1] Chamberlain, R., Homework 4 Handout, Fall 2024

[2] Y. LeCun, “The MNIST dataset of handwritten digits.”

[3] William, R., CSE 260M Lab #2, Fall 2023