Innervator: Hardware Acceleration for Neural Networks

Implementing AI into FPGAs with VHDL

CSC-494 Independent Study

Nomenclature

To innervate means “to supply something with nerves.”

Innervator is, aptly, an implementer of *artificial neural networks* within *Programmable Logic Devices*.

Foreword

- Prior to starting this project, I had no experience or training with artificial intelligence (“AI”), electrical engineering, or hardware design;
- Hardware design is a complex field—an “unlearn” of computer science; and
- Combining the two ideas, AI & hardware, transformed this project into a unique proof of concept.

Synopsis

- Inspired by biological brains, AI neural networks are modeled in mathematical formulae that are inherently concurrent;

- AI applications are widespread but suffer from *general-purpose* computer processors that execute algorithms in step-by-step sequences; and

- Programmable Logic Devices (“PLDs”) allow for digital circuitry to be predesigned for data-tailored and massively parallelized operations.

AI Briefly Visited

A Simple Concept Over-Complicated by Corporate Marketing and Academia

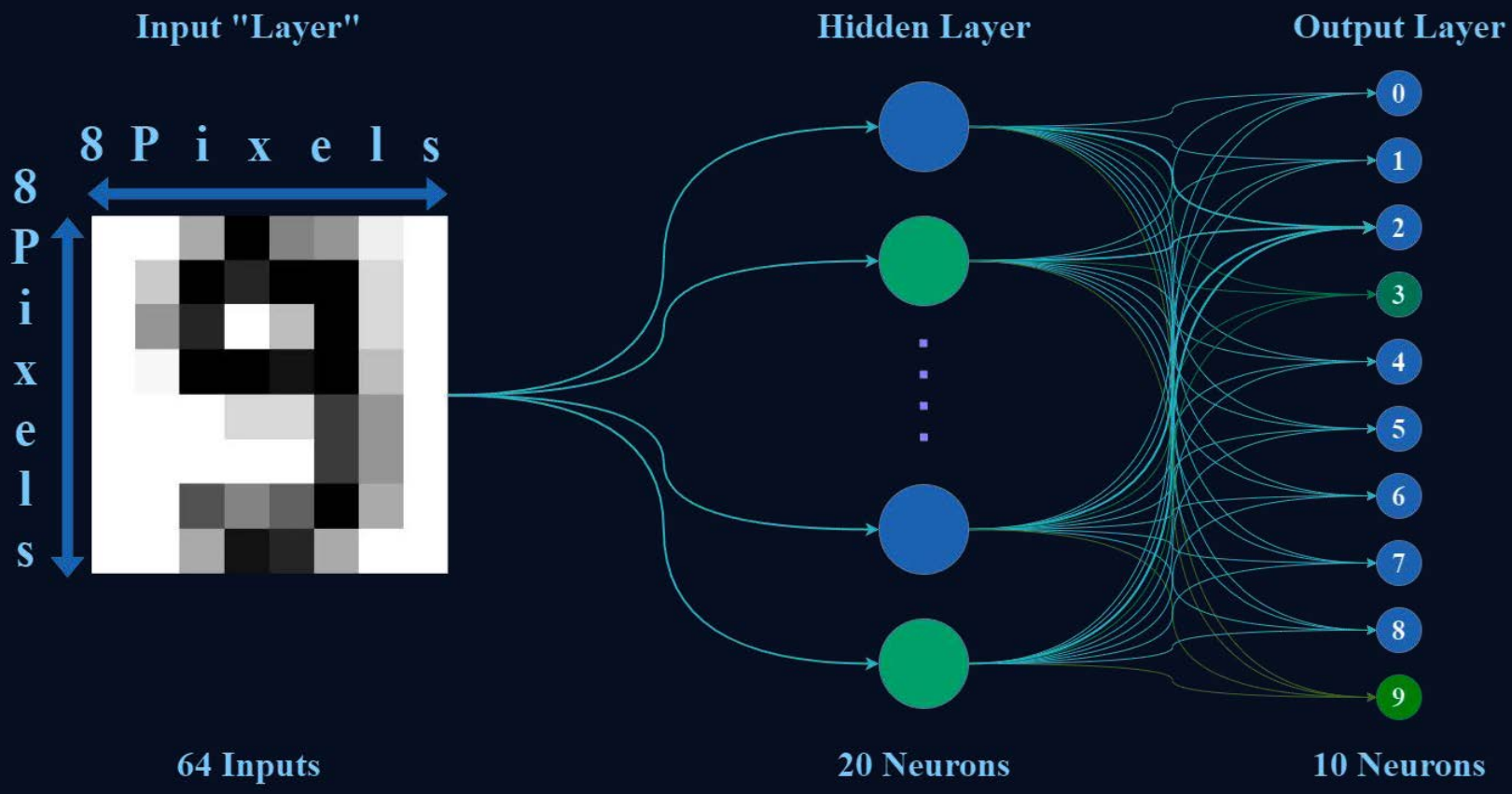
The average human brain has about one hundred billion neuron cells. While many of these neurons perform trivial operations (e.g., detecting horizontal/vertical lines), a conglomerate of them could be used to perform increasingly sophisticated tasks.

How AI Thinks?

We use branching (i.e., if-then decisions) daily in our lives: “if it rains tomorrow, I will bring an umbrella; else, if it’s sunny, I will visit the zoo.”

Neurons Operate in a Similar Manner!

They do not necessarily branch on Boolean (strictly True/False) algebra, but they do so in a statistical and fuzzy-logic sense:



My neural network consists of 2 “densely connected” layers & 30 neurons.

How was the Neural Network Trained?

In order to first learn what AI was all about, I built my own (open-source) library to create & train neural networks, using Python & NumPy.

The Network was trained over 1709 images of 8x8 pixel handwritten digits from the MNIST-8 dataset; it achieved a 98% accuracy.

The Problem with Using Traditional Hardware for AI Tasks

- CPUs predominantly operate in a sequence;
- CPUs have an unpredictable latency; and
- Consequently, GPUs are being "retrofitted" for AI, but they also come at the cost of increased space & power consumption.

Programmable Logic Devices (PLDs)

PLDs are digital circuitry that can be customized down to the specific parameters of a network, ensuring data-tailored computation and algorithmic parallelism.

Massive Parallelism

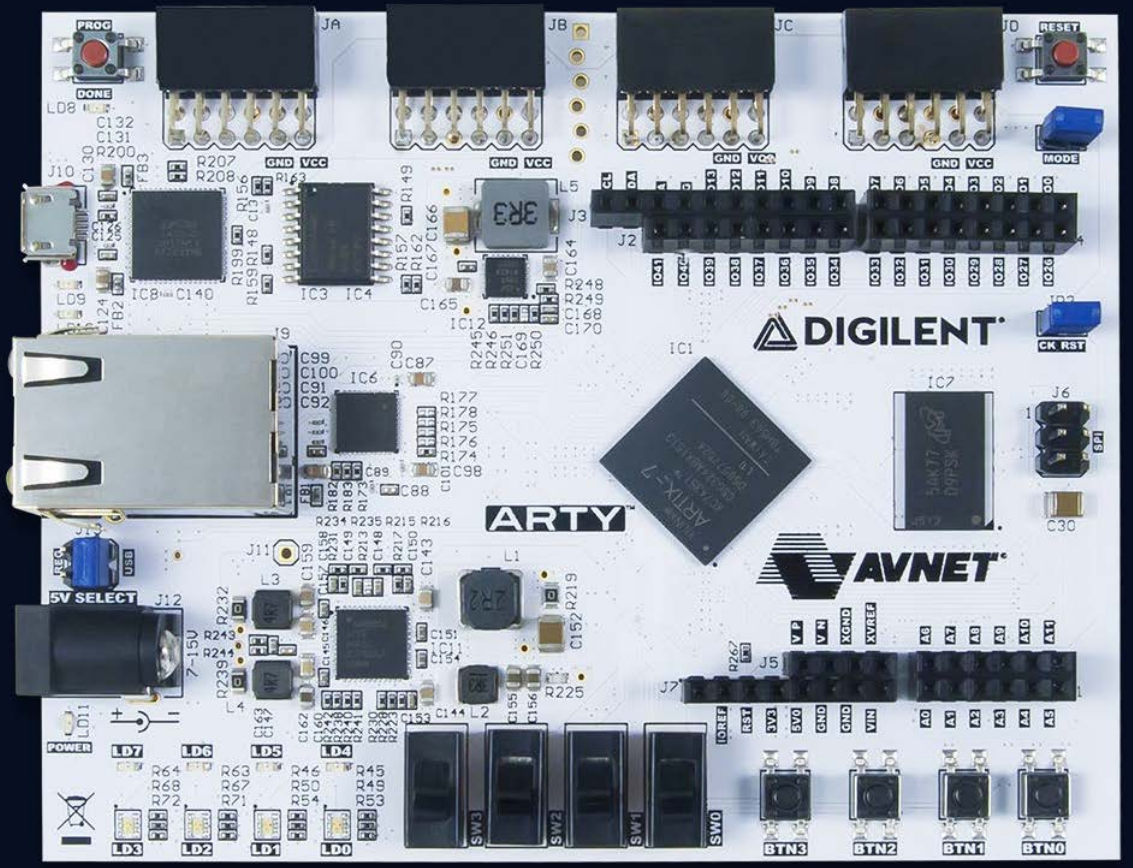
How can PLDs Effortlessly Perform Many Operations in Parallel?

One analogy I have thought of is the flow of water within pipes: as long as you have the water capacity (in case of PLDs, electricity), it doesn't matter how many branches a pipe splits to, for they all fill (i.e., "process") at the same time. In contrast, a CPU would use a few buckets and manually fill each pipe from a nearby well (memory).

Field-Programmable Gate Arrays (FPGAs)

Furthermore, a subgroup of PLDs are the dynamically reconfigurable FPGAs; they are reusable and can have subsequent customized designs swapped out “in-the-field.”

In my project, I used a low-cost, small-scale "*Xilinx Artix-7*" FPGA.

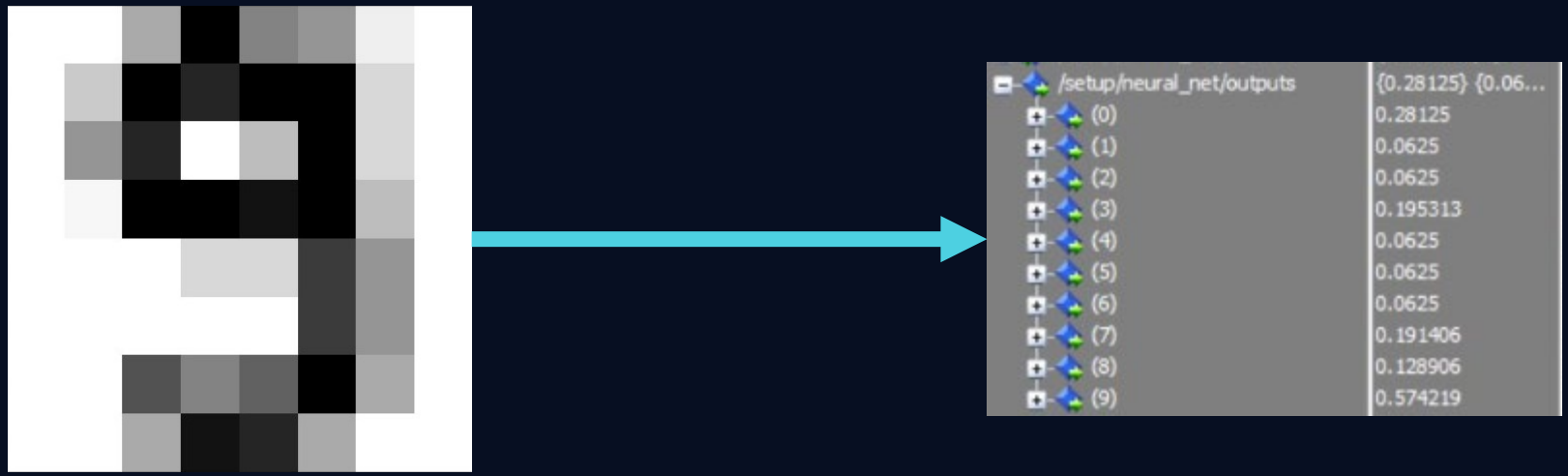


Innervator

Hardware Acceleration for Neural Networks

Innervator provides the infrastructure to automatically infer the suitable hardware design based on a given neural network's parameters.

Other than being portable, it is highly configurable and comes with some other helpful utilities (e.g., a receiver/transmitter), as well.



A Simulation of Innervator, which was given a sample image of the Digit 9, after ~1000 nanoseconds.

My Workflow

- **Language:** VHDL-2008
- **Synthesizer:** Xilinx Vivado v2023
- **Simulator:** Mentor Graphics ModelSim v2016

A Brief Preview of a Handful of Challenges I have Faced

Hardware design requires significant "unlearning."

A single line of Python code can take upwards of 800+ lines of carefully crafted VHDL.

Debugging is difficult: once you synthesize a buggy VHDL code into an FPGA, it'll be very hard to find what went wrong.

Besides, synthesizing itself takes dozens of minutes, each time, and simulators can be unreliable.

A Brief Preview of a Handful of Challenges I have Faced (cont.)

You cannot synthesize floating-point (i.e., fractional) numbers directly into hardware.

Even basic mathematical operations (e.g., division) cannot be taken for granted.

Endless timing failures force you to painstakingly re-design your approach every time.

Toolchains & language are incredibly buggy.

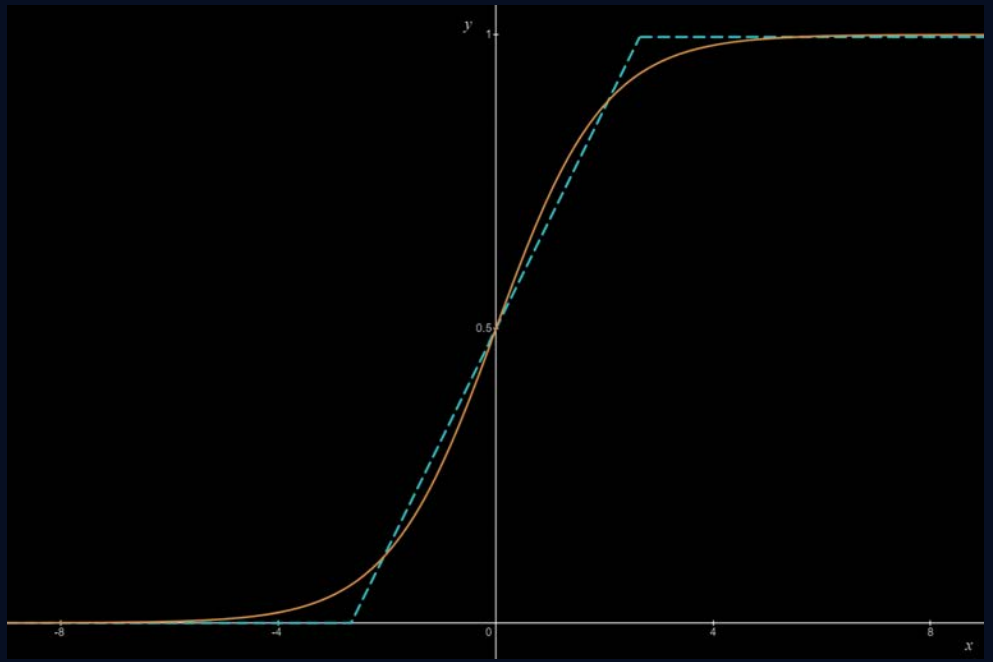
Et cetera, et cetera, et cetera…

Approximation of the Sigmoid Activation Function

Given that the Sigmoid function contains a division and exponent, implementing it in hardware would be very costly;

I manually designed a linearized, fixed-point approximation with comparable accuracy:

$$\frac{1}{1+e^{-x}} \approx .1875 \cdot x + .5$$



Some of the Bug Reports I Submitted to IEEE and Xilinx

https://gitlab.com/IEEE-P1076/VHDL-Issues/-/issues/311

https://gitlab.com/IEEE-P1076/VHDL-Issues/-/issues/312

https://support.xilinx.com/s/question/0D54U00008CO8pTSAT/bug-fileopen-is-not-consistent-with-ieee-standards

https://support.xilinx.com/s/question/0D54U00008ADvMRSA1/bug-in-vhdl-textioread-overload-of-real-datatypes-size-mismatch-in-assignment

Innervator

Free and Open-Source

Dual-licensed under GNU LGPL-3.0 and CERN-OHL-W-2.0, Innervator is a free and open-source project; you may find its source code and other material in its GitHub repository:

github.com/Thraetaona/Innervator

Potential Alternatives

- ASICs;
- Analogue Hardware; and
- Integer-Based Neural Network Algorithms.

CSC-494 Independent Study

Questions?

How the Neural Network was Trained (for Q/A)

In order to first learn what AI was all about, I built my own program to create, train, and test generalized neural networks. Written in Python and using NumPy, I had a network with 2 dense layers (with 20 and 10 neurons, respectively) go over 1709 images of 8x8 pixel handwritten digits, from the MNIST-8 dataset, 100 epochs (i.e., iterations).

More technically, I used the Sigmoid formula in place of each neuron's activation function, and the MSE (Mean Squared Error) was used to back-propagate and optimize the Network's parameters (i.e., weights and biases).

Afterward, the Network was tested on a series of 50 never-seen images outside of its training data, and it yielded a 98% accuracy on recognizing their digits correctly.

However, I was not limited to image digit recognition and could choose to specialize the neural network in other areas, like sound analysis, too; digit recognition was chosen due to it being more visually appealing as a proof of concept.

Although this AI library was more of a miscellaneous sub-project in comparison to the much more nuanced hardware aspect of the overall project, I still decided to open-source it under the MIT license at https://github.com/Thraetaona/Innervate

How are These PLDs Programmed? (for Q/A)

In the past, they were manually drawn as blueprints. Then, as a part of the VHSIC program by the U.S. DoD, the *VHSIC Hardware Description Language* ("VHDL") was initiated in 1981. After cooperation from industry, VHDL's first version was released in 1985, shortly after Xilinx had invented the first FPGA in its previous year. Eventually, VHDL was given to the IEEE to standardize and develop in the future.

VHDL and Verilog (another HDL) are the two most popular languages in hardware design. For this project, I chose to use a subset of **VHDL-2008.**

Hardware design itself is very time-consuming and, similarly, the toolchain and ecosystem that surround it have also been very slow to adapt to changes; for instance, the latest version of VHDL was released in 2019, but state-of-the-art tools (e.g., Xilinx Vivado, Intel Quartus, etc.) are still working on supporting the 2008 version of VHDL fully.

Unfortunately, in stark contrast to how stable and mature the tools in software design are, I have found hardware design toolchains to be incredibly buggy and unreliable, making it magnitudes more challenging to learn the basics. In fact, I reported several such bugs and limitations myself.