Vision Transformer Accelerator ASIC for In-Ear Sleep Staging

by

Tristan Robitaille

Supervisor: Professor Xilin Liu
April 2024

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

This page intentionally left blank.

ESC499 Engineering Science Thesis

Vision Transformer Accelerator ASIC for In-Ear Sleep Staging

Tristan Robitaille
Student number: 1006343397
Email: tristan.robitaille@mail.utoronto.ca

Supervisor: Professor Xilin Liu
Email: xilinliu@ece.utoronto.ca

April 12th, 2024

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Keywords: Sleep staging, ASIC accelerator, vision transformer, computer architecture

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Xilin Liu, for his guidance and support throughout the project. He has given me the freedom to explore new ideas and had provided me with the support and tools I needed.

I would also like to thank my father, Claude Robitaille, for letting me remotely use his workstation to train the model and run the accuracy study. He has also helped review the code for the functional simulation.

In addition, I owe much to the professors who have taught me the fundamentals of computer architecture at the University of Toronto - Profs. Jason Anderson, Natalie Enright-Jerger, Andreas Moshovos and Mark C. Jeffrey.

Throughout this project, I have made extensive use the Compute Canada cluster, which has provided me with the computational resources I needed to run the simulations and train the model. I would like to thank the staff at Compute Canada for their initiative. I am also appreciative of the tools provided by the Canadian Microelectronics Corporation, which have been instrumental in the hardware implementation of the accelerator.

I would also like to acknowledge the work of Professors Lisa Romkey and Alan Chong who organized this thesis project for us, ensuring a structured and productive environment.

Finally, I would like to thank my family and friends for their support and encouragement throughout this project. I am grateful for their patience and understanding during this time.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog-to-Digital Converter

AFE Analog Front-End

ASIC Application-Specific Integrated Circuit

CiM Compute-in-Memory

CMOS Complimentary Metal Oxide Semiconductor

CSV Comma-Separated Values

EEG Electroencephalography

HDL Hardware Description Language

HDF5 Hierarchical Data Format 5

IP Intellectual Property

PE Processing Element

PPA Power, Performance, and Area

MAC Mulitiply-Accumulate

MASS Montreal Archive of Sleep Studies

MHSA Multi-Head Self-Attention

MLP Multi-Layer Perceptron

PSG Polysomnography

RTL Register Transfer Level

TSMC Taiwan Semiconductor Manufacturing Company

VCD Value Change Dump

FSM Finite State Machine

See [1]. I am making an Application-Specific Integrated Circuit (ASIC). It's small, low-power and fast. It's better than Google's.

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2 Background

2.1 Problem Statement

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.2 Technical Goals and Requirements

The main objectives of the model are:

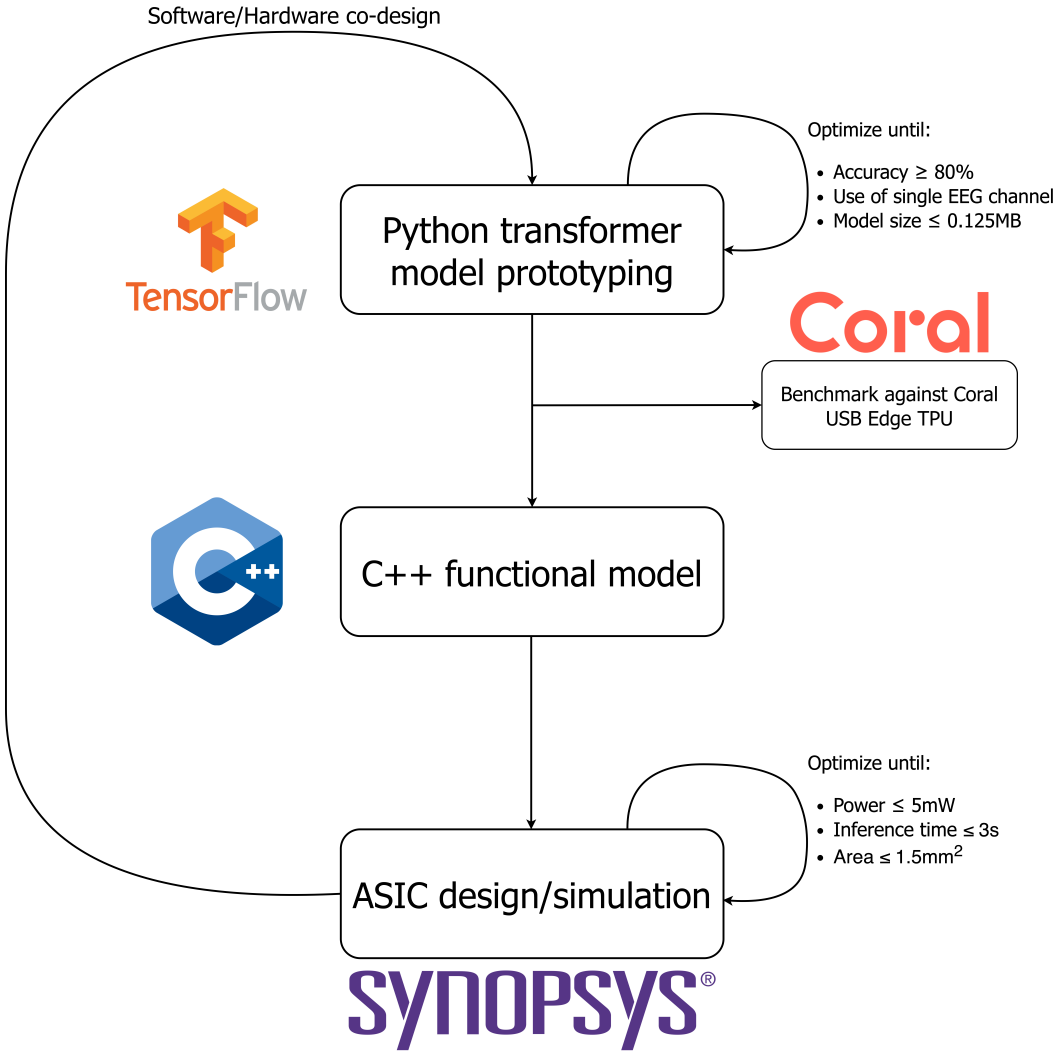
- To use as few weights as possible to reduce the area and energy consumption of the accelerator.
- To reduce the number of matrix transpose operations to reduce latency by limiting data movement.
- To maintain high accuracy in sleep staging.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3 How to Design an AI Accelerator

This section describes the workflow used for this project, which was divided into three main steps: model prototyping, accelerator functional simulation and accelerator hardware implementation. The tools used in each step are described in the following sections. The goal is to expose progressively more layers of abstraction to make hardware/software co-design and debugging easier. As can be seen in Figure 3, these three steps allow iterations to converge on a design that meets the requirements described in Section 2.2.

Figure 1: Three step workflow for designing an AI accelerator



3.1 Model Prototyping, Data Processing and Accuracy Measurements

The first step in designing an accelerator is prototyping the model that the accelerator will run. Here, we prioritize productivity of development and profiling over performance.

In this project, the model was developed in TensorFlow, a widely-used Python framework maintained by Google. Its popularity implies that it has a large community of developers and is well-documented. TensorFlow also provides a high-level API that allows for rapid prototyping of models. Furthermore, it easily converts to TensorFlow Lite, which is compatible with the Google Edge TPU device that will be used as a reference point of available commercial hardware. The model was developed in Python 3.11 and TensorFlow 2.14. The model was trained on the Montreal Archive of Sleep Studies (MASS) SS3 dataset, which contains 62 nights of Polysomnography (PSG) recordings with 21 Electroencephalography (EEG) channels [2]. The 16-bit raw PSG data was preprocessed manually with the following steps:

- Pruning of epochs of unknown sleep stage.
- Downsampling from 256Hz to 128Hz to reduce model size and inference energy.
- Filtering with 60Hz notch filter to remove noise from AC mains coupling.
- Filtering with 0.3-100Hz bandpass filter to remove noise (as recommended in [3]).
- Offset by half of the scale to replicate the unsigned 16-bit format expected from the Analog-to-Digital Converter (ADC) in the final hardware.

In addition, the two light sleep stages (N1 and N2) were merged into one stage to simplify the model. Finally, the nights were concatenated and shuffled. All training and hyperparameter search took place on the Compute Canada Cedar cluster through remote SSH access.

Accuracy against PSG ground truth was assessed through repeated 31-fold validation: the model is trained on 60 nights and tested on the remaining two nights. The best accuracy of 5 runs is recorded, and the process is repeated another 30 times until all pairs of nights have been tested. The training set represents 90% of the 60 training nights. The final accuracy is the average of the best accuracies of each validation fold. This provides measurements that are robust against night-to-night variability in the dataset and resulting inference performance. Table I shows the hyperparameters used for training the model. These have been empirically determined to yield the best accuracy with reasonable training time. Figure 3.1 shows the accuracy of the model as a function of the number of epochs, which is shown to converge at around 100 epochs. Furthermore, 3.1 indicates that a dropout rate of 30% is optimal for this model.

3.2 Accelerator Functional Simulation

To prototype the accelerator architecture, run more accurate studies to determine the impact of design choices and write the model in a way that can be easily translated to

Figure 2: Testing set accuracy for 5 randomly-selected folds as a function of epoch

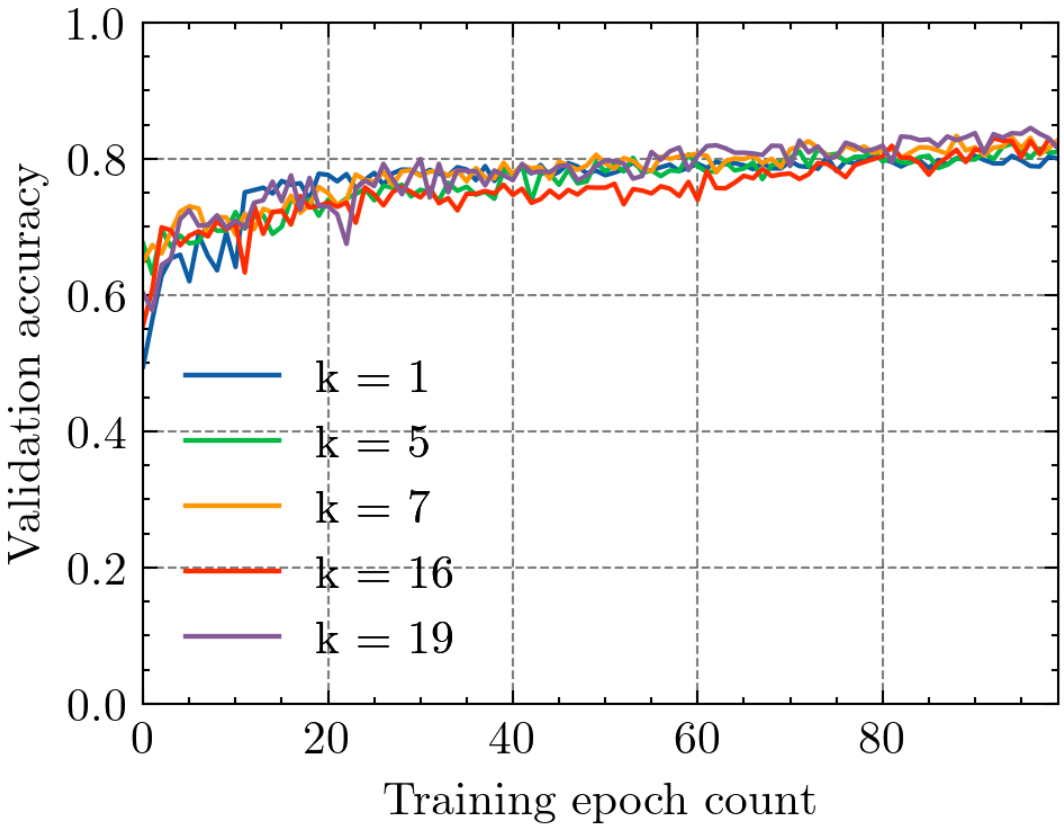


Figure 3: Testing set accuracy for vs. dropout rate during training

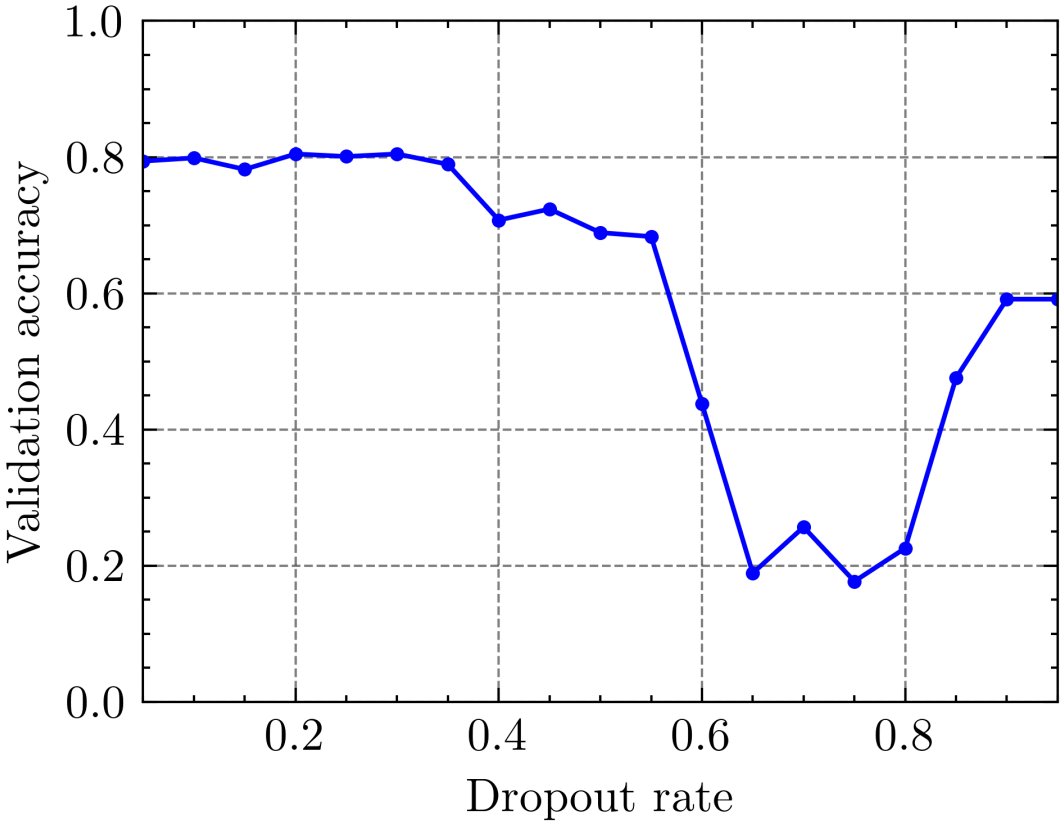


Table I: Training hyperparameters for vision transformer model

Hyperparameter	Value
Learning rate schedule	$\sqrt{d_{model}} * min(\sqrt{step}, step/4000^{1.5})$
Initial learning rate	0.001
Batch size	16
# of epochs	100
Dropout rate	30%
Class weights	1.0 $\forall$ {Wake, REM, N1/N2, N3/N4}
Optimizer	Adam
Data downsampling	256Hz $\rightarrow$ 128Hz
Data filtering	60Hz notch $\rightarrow$ 0.3-100Hz bandpass $\rightarrow$ 16b quantization

hardware, a functional simulation was written. The simulation is written in C++ and, in the aim of helping subsequent SystemVerilog development, uses a similar structure to the SystemVerilog code (cycle-level parallelism, use of FSM, limited function calls). It is organized identically to the hardware design, with a `master` module controls high-level operation of `CiM` modules. It also makes use of compute modules written using the same fixed-point format and approximation as the hardware. This functional simulation is used to collect metrics that are difficult to measure in hardware, such as the distribution of inputs to certain operations, the distribution of intermediate results, the exact number of all types of operations, etc. This information can be used to optimize the hardware design. Finally, it provides an easy way to validate the operations by cross-checking each step with reference outputs from the TensorFlow model. The only non-standard libraries used are `armadillo` for compute verification, `HighFive` for Hierarchical Data Format 5 (HDF5) file I/O (storing model parameters and EEG data) and `rapidcsv` for Comma-Separated Values (CSV) file I/O (storing fixed-point accuracy study results).

3.3 Accelerator Hardware Implementation

The final step in the workflow is the hardware implementation of the accelerator. The hardware is written in SystemVerilog and uses the same structure as the functional simulation. SystemVerilog was chosen as it provides many more "quality of life" features than plain Verilog, such as classes, interfaces, packages, assertions, etc., resulting in higher-quality and more productive code. Several tools are used to design the hardware:

- Verilator: Register Transfer Level (RTL) compiler and linter.
- CocoTB: Python testbenching framework.
- Gtkwave: Value Change Dump (VCD) waveform viewer.

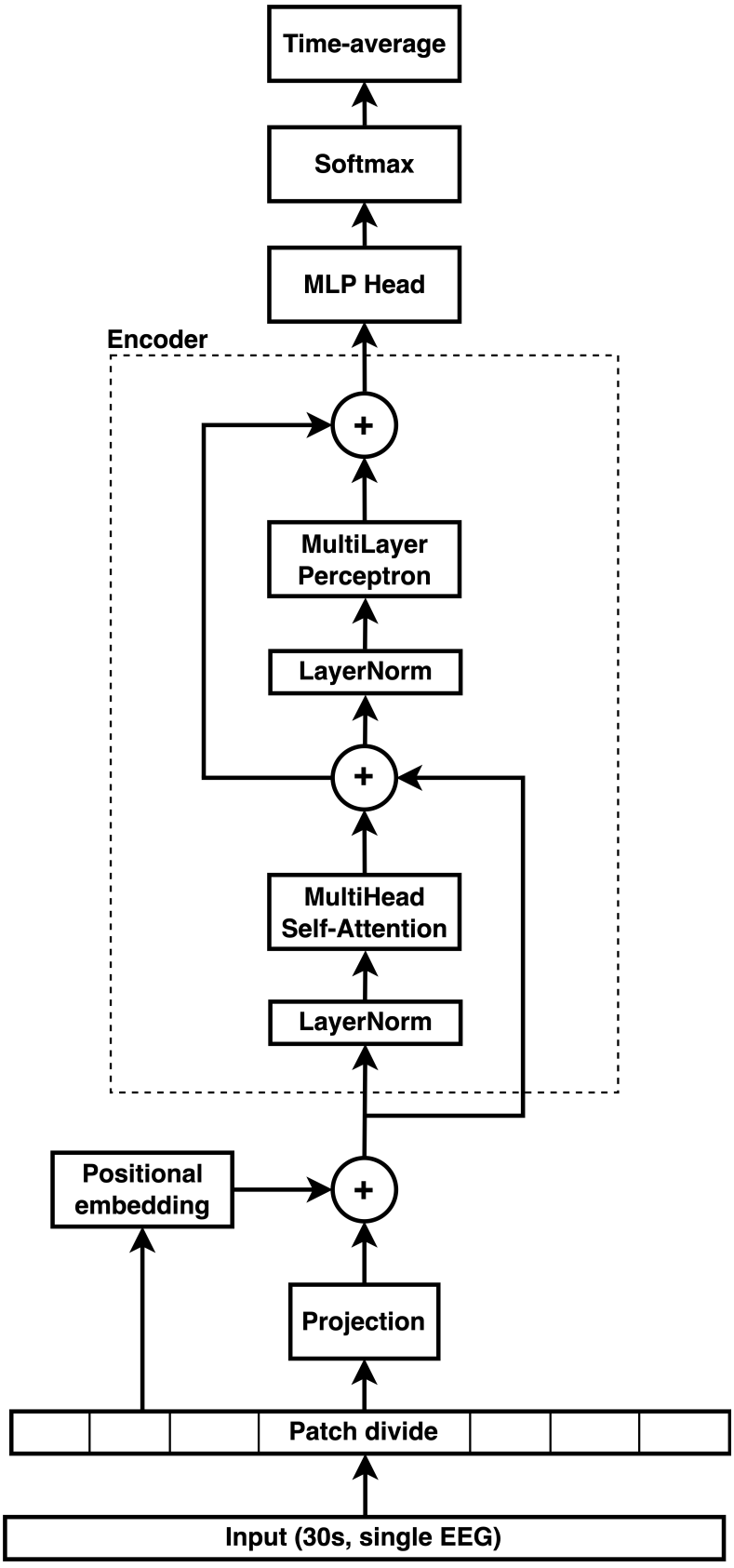
- ARM Artisan Physical IP: SRAM compiler.
- Synopsys Design Compiler: Synthesis and performance evaluation tool.

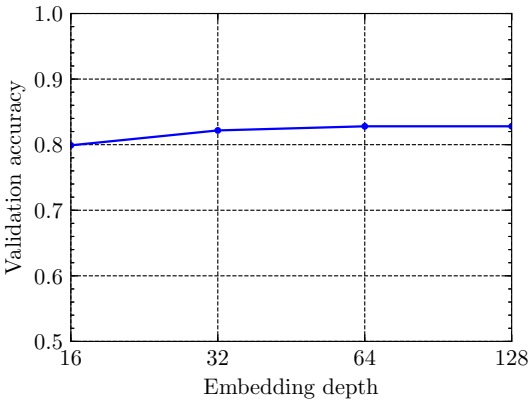
The first three tools are open-source and compatible with all major operating systems, providing a familiar, local and OS-agnostic development environment. The last two tools are proprietary and are used to evaluate the performance of the design against requirements.

4 Vision Transformer Model Design

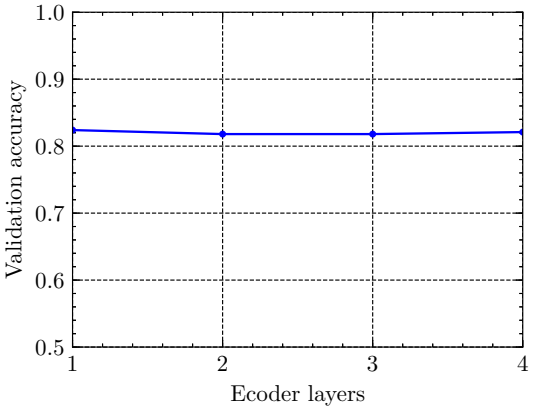
This section describes the architecture of the vision transformer model used in this thesis, which is shown Figure 4. Since sleep staging as presented here is a "sequence-to-one" problem, feedback is not applicable and thus the decoder stack present in a more traditional transformer is not needed. The patch divide step and first dense (known as "patch projection" in [4]) step are performed as data comes in from the EEG ADC to reduce temporary storage usage. The patch divide step splits the input into 60 patches of 64 samples. The resulting nearly square matrix helps maximize utilization of the accelerator described in section 5, yielding shorter inference time and lower inference energy. The patch projection step adds a layer of learned weights to the input to allow the model to capture more complex features from the input waveform. The projection depth is known as "embedding depth" or `d_model` and is a hyperparameter of the model. The model uses an embedding depth of 64, which was determined through hyperparameter search to yield accuracies similar to embeddeding depths of 32 or 128 (see Figure 5a) but offers lower energy consumption when paired with 60 patches. The model then applies a learned 1D positional embedding to the patches to allow the model to learn encoder positional infromation to the EEG stream. The model then applies a single encoder layer consisting of a Multi-Head Self-Attention (MHSA) and Multi-Layer Perceptron (MLP) layer. Figure 5b shows that accuracy does not increase as more encoder layers are added, so the model uses only one encoder layer to reduce the model size. The two LayerNorm layers first normalize the inputs to a Gaussian over the second dimension, and then scale and shift the normalized values using learnable parameters with the goal of faciliting learning and containing compute unit inputs throuh reducing covariate shift. The MHSA layer uses a triplet of three-dimensional weights (known as "key", "query", "value") to favour certain regions of the encoder input. The MHSA layer contains the majority of the weights in the model and is the most computationally expensive part of the model. The third dimension of the weights is known as the "number of heads", and the model uses 8 as it was found to yield the highest accuracy as seen in Figure 5c. The MLP head layer is a simple parametrized sequence of dense layers. Here, a single matrix was found to yield the highest accuracy. The final dense and softmax layer is used to map the model output to the sleep stage classes. Finally, the model takes the window average of the last three softmax vectors to stabilize the sleep stage noise. This addition has increased the model's accuracy by 2.5% and is a novel technique in sleep staging models. Table II summarizes the hyperparameters of the model.

Figure 4: High-level transformer architecture for in-situ sleep staging

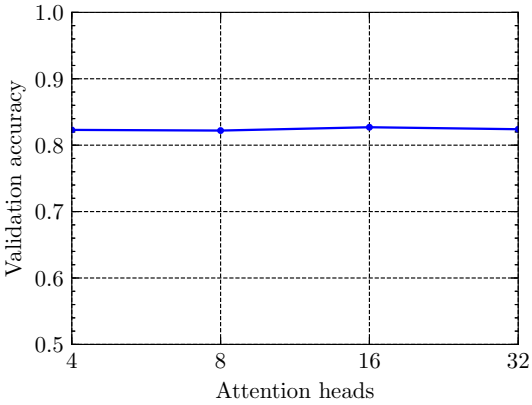




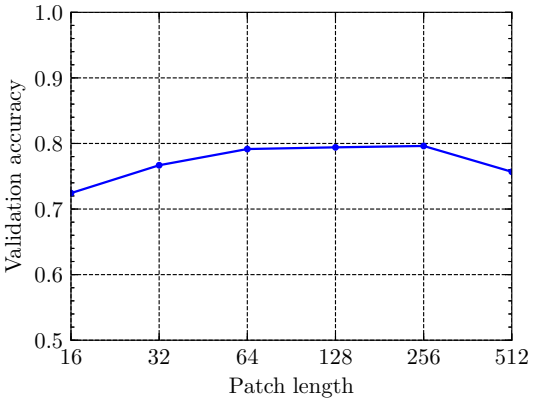
(a) Accuracy vs. `d_model`



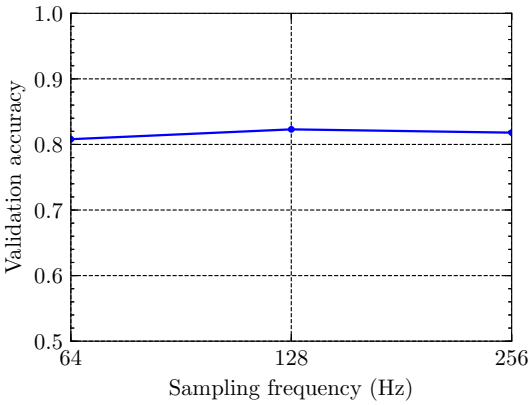
(b) Accuracy vs. encoder layer count



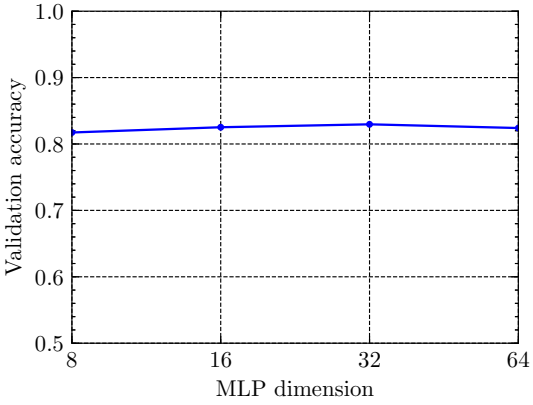
(c) Accuracy vs. number of heads



(d) Accuracy vs. patch length



(e) Accuracy vs. sampling frequency



(f) Accuracy vs. MLP dimension

Figure 5: Hyperparameter search results for the vision transformer model

Table II: Metrics and hyperparameters for vision transformer model

Metric	Value
Input channel	Cz-LER
Size (# of weights)	31,589
Sampling frequency	128 Hz
Clip length	30s
Patch length	64 samples
Embedding depth (d_{model})	64
# of attention heads	8
# of encoder layers	1
MLP dimension	32
MLP head depth	1
Output averaging depth	3 samples

5 ASIC Accelerator Architecture

This section describes and justifies the design of the ASIC accelerator that will run the vision transformer model. It is split into several sections to describe the various aspects of the accelerator. A high-level overview of the accelerator is shown in Figure 6. As can be seen, it comprises a single Master module responsible for interfacing with the host system and number of so-called Compute-in-Memory (CiM) modules that perform the actual computation and a shared bus for data and control signals. For reference, the memory for shared parameters and intermediate results occupies 65% of the area, the compute units occupy 20% of the area and the control logic occupies 15% of the area.

5.1 Centralized vs. Distributed Architecture

The first major design aspect of the accelerator is whether to use a centralized or distributed architecture. A centralized architecture has a single compute unit that performs all operations and a single memory bank, while a distributed architecture has multiple compute units, each with their own memory. Of course, linear algebra lends itself very well to parallelization as each Mulitiply-Accumulate (MAC) operation is independent of the others. The centralized architecture is simpler to design, has less overhead and lower area, but is much slower. It is relatively hard to accurately predict the Power, Performance, and Area (PPA) of different architectures before implementation. However, designing a distributed architecture first can be a good starting point as it can easy be scaled up and down to optimize the design. This is the reason this design uses a distributed architecture.

The design uses 64 CiM to maximize Processing Element (PE) utilization of the ASIC.

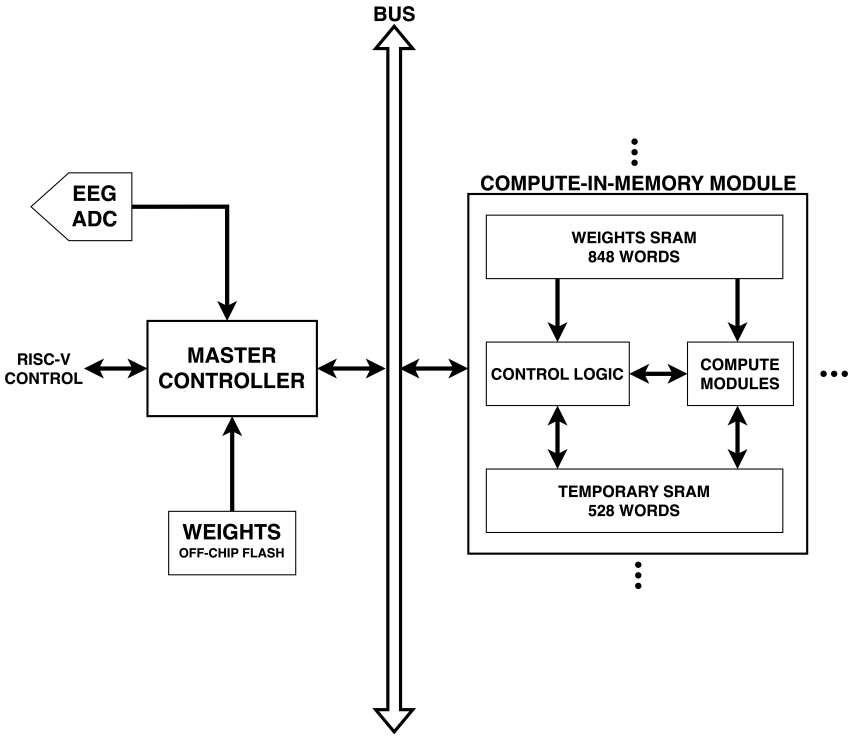


Figure 6: High-level architecture of the ASIC accelerator

Indeed, most weights and intermediate results matrices have at least one dimension that is 61 (number of patches + classification token) or 64 (embedding depth of the model). With 64 CiM, all vector operations can be computed at once, avoiding extra overhead for control and data movement while limiting the amount of unused silicon.

5.2 Data and Control Bus

The data and control bus is a bidirectional bus that connects the Master module and all the CiM modules. Once device may communicate at a each and each is connected to the bus through tri-state buffers. The bus is 58 bits wide and contains different fields, as summarized in Table III. There are 10 different operations `Op`, as described in Appendix A. The bus being directly connected to all modules allows for two important features: broadcast and synchronization. There are two types of broadcast: dense and transpose. Dense broadcast is used to send the same data to all CiM modules and is used during matrix-matrix multiplications, while transpose broadcast is used to transpose the matrix (a CiM holds a row instead of a column). The CiMs are autonomous with these operations. They do not need do not need involvement from the Master other than to start the operation. Synchronization is maintained with the `START_PISTOL` broadcast, which instructs the CiMs to go to the next step in their Finite State Machine (FSM).

Table III: Fields of the data and control bus

Field	Width (bits)	Description
`Op`	4	Opcode of the instruction/data
`ID`	6	ID of the sender or target CiM
`Data`	3×16	Data (up to $3 \times$ Q6.10)

5.3 Master Architecture

The Master module is responsible for interfacing with the host system. It is a simple module that receives signals from the host microprocessor to start parameter load and start a new inference. It also interfaces with the external memory holding the parameters and the ADC in the EEG Analog Front-End (AFE). It is responsible for directing the CiM to perform dense or transpose data broadcasts and for synchronizing their high-level (step-by-step or multi-step) operations.

5.4 Compute-in-Memory: Architecture

The CiM module is the heart of the accelerator. It is responsible for performing the actual computation. Each CiM module has its own memory for intermediate results and parameters along with control logic and several compute units needed to perform all computation in the model. The CiM module is designed to be as autonomous as possible to limit control overhead. It follows the dataflow model of computation, where data is passed from one step of the inference to the next without Master involvement, unless data needs to be broadcast or transposed. The following sections describe different aspects of the CiM module.

5.5 Compute-in-Memory: Memory

The CiM module contains two single-port memory banks: one for intermediate results and one for parameters containing 848 and 528 words, respectively. Having two separate banks allows for simultaneous read/write to both banks. The memory is generated by ARM's Artisan IP memory compiler and is 65nm 6T SRAM 0.525µm^2. It has an aspect ratio of 4, which, as seen in Figures 7 and 8, provides the lowest overhead for the given capacity. The memories are single-cycle. They can also be switched into a low-power mode when not in use. Table IV shows various metrics of the memory banks for operation at 1.0V and 25°C.

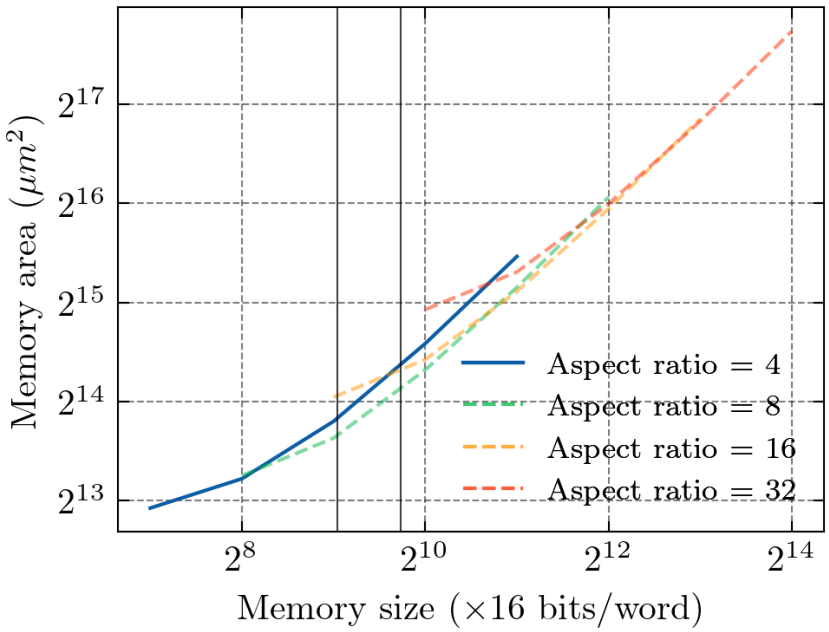


Figure 7: Area of memory banks vs capacity for different aspect ratios

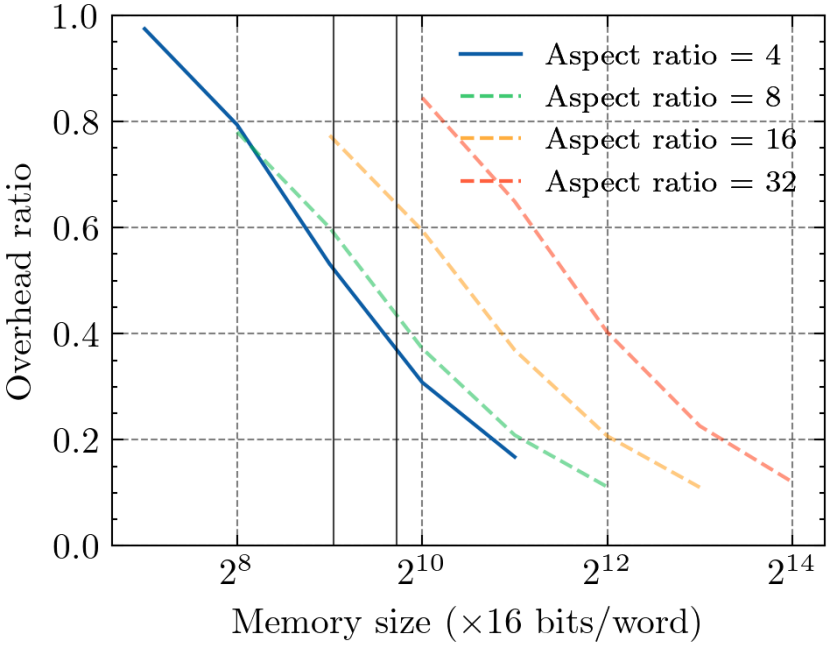


Figure 8: Overhead of memory as a fraction of total area for different aspect ratios

Table IV: Line and file count per file type in the codebase

Metric	528 words	848 words
Area	16682.54µm^2	22124.25µm^2
Leakage (nominal)	0.122mW	0.173mW
Leakage (data retention)	0.0754mW	0.055mW
Cycle time	0.852ns	0.865ns

5.6 Compute-in-Memory: Fixed-Point Accuracy

All computation in the CiM module is done in fixed-point format. The model uses Q22.10 format, which has 22 integer bits (including one sign bit) and 10 fractional bits. This format was chosen because it significantly reduces the area and power consumption of the accelerator compared to floating-point format. The fixed-point format is also sufficiently accurate for the model. To determine the accuracy of the fixed-point format, an error study was conducted on the functional simulation. Using the `fpm` library, all data operations were performed in fixed-point format. The model was ran on a randomly-selected night of sleep data and the output was compared to the output of the TensorFlow model and the ground truth. Figure 9 shows the error of the fixed-point format compared to the ground truth as a function of the number of fractional bits (on a 32-bit fixed-point number). As can be seen, the accuracy peaks at 8 bits of fractional precision, only 1.0% below the accuracy of the TensorFlow model (which uses 32-bit floating-point format). We can also notice that accuracy drops significantly starting at 19 fractional bits. This is because some intermediate results overflow when the numbers have less than 12 integer bits, especially because of operations such as MAC, softmax and LayerNorm, which accumulate numbers over a length of 64. One limitation of the `fpm` library is that it can only represent numbers with a total bit count of 4, 8, 16, 32 or 64 bits, so the ideal number of integer bits cannot precisely be determined. The hardware uses Q8.8 (16-bit) for storage and Q22.10 (32-bit) for computation. The computation format uses more integer bits to avoid overflow errors discussed above and two additional fractional bits to avoid inaccuracies caused by divide by zeroes.

5.7 Compute-in-Memory: Compute Modules

This section describes the design and performance metrics of the various compute Intellectual Property (IP) modules use by the CiM modules. Each is custom-designed for this project. Each module works with signed (2's complement) fixed-point representation. To avoid overflow, the modules use internal temporary variables of fixed-point format Q22.10. Table V shows the performance metrics of the compute modules. The working principles of

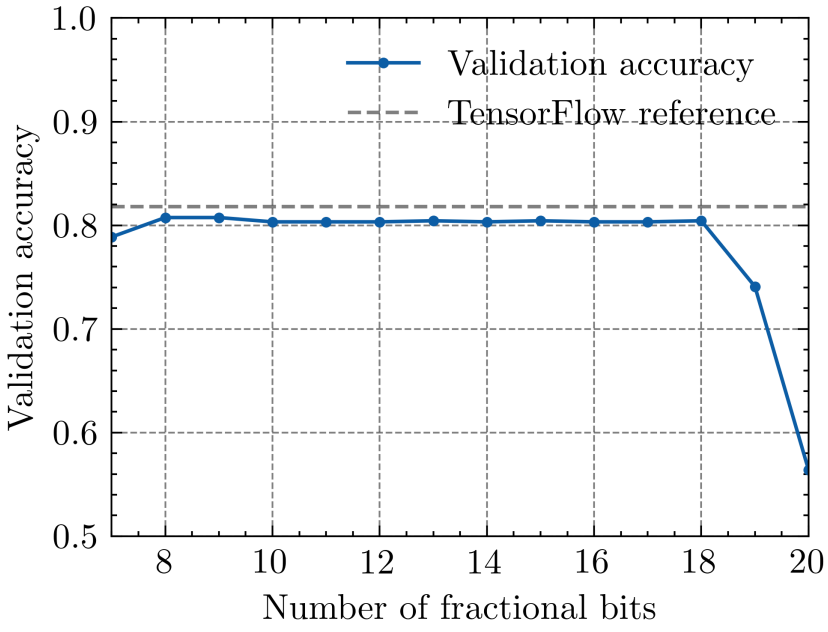


Figure 9: Model accuracy vs number of fractional bits in fixed-point format

each modules is described briefly in subsequent sections.

Note that all measurement in Table V are given for standard 65nm Taiwan Semiconductor Manufacturing Company (TSMC) process with a 100MHz clock. To determine these metrics, the following methodology was used with Synopsys Design Compiler 2017.09 running on UofT's EECG cluster:

- Area: Synthesis with the area optimization effort set to `high`, and the area was extracted from the `report_area` command report.
- Cycle/op: The latency was observed when running a single operation on a pre-synthesis simulation.
- Energy/op: A single-instance testbench running 10000 operations was designed, and a `.saif` file was generated from the VCD dump file of the testbench using Synopsys' `vcd2saif` utility. This provides an average activity factor for each node, yielding an accuracy that is adequate for this discussion. The energy per operation was calculated by multiplying the total dynamic power by the time to complete the 10000 operations, divided by 10000.
- Leakage power: Synthesis with the power optimization effort set to `high`, and the leakage power was extracted from the `report_area` command report.
- F_{max}: The `report_timing` command was used to determine the maximum frequency of the design.

Table V: Performance metrics of the compute modules

Module	Area	Cycle/op	Energy/op	Leakage power	F_{max}
Adder	450.4µm^2	1	0.99pJ	11.87µW	6.67GHz
Multiplier	3535.2µm^2	1	7.05pJ	90.50µW	1.59GHz
Divider	1719.9µm^2	35	23.44pJ	34.56µW	1.11GHz
Exponential	2442.2µm^2	24	62.73pJ	47.10µW	7.14GHz
Square Root	1325.2µm^2	17	18.32pJ	26.30µW	0.758GHz
MAC[1]	—	386	820.20pJ	—	—
MAC[2]	3129.8µm^2	391	839.32pJ	69.40pJ	2.17GHz
MAC[3]	—	456	941.68pJ	—	—
Softmax	2341.1µm^2	2024	1972.5pJ	51.47µW	1.20GHz
LayerNorm	3836.89µm^2	1469+494	1705.7pJ	78.39µW	0.877GHz
Total	18780.69µm^2	N/A	N/A	409.59µW	0.758GHz

[1] No activation function applied
[2] Linear activation function applied
[3] Swish activation function applied

It must be noted that the measurements for all composite compute units (i.e. units that make use of shared resources) *exclude* the area/power/etc. of the shared resources. Including them would result in misleadingly high figures, given that they are explictly designed to share resources. The total area of the CiM provides figures more representative of this integration.

5.7.1 Adder

The adder is a single-cycle, combinational module that adds two fixed-point numbers. It uses a ripple-carry adder architecture. The adder has a latency of 1 cycle, which simplifies the logic that uses it. It also provides an overflow flag. To reduce dynamic power consumption, the adder only updates its output when the `refresh` signal is high.

5.7.2 Multiplier

The multiplier is very similar to the adder. One difference is that it uses Gaussian rounding (also known as banker's rounding). This rounding method rounds 0.5 to the nearest even number. This reduces the bias in the output that is commonly observed with standard rounding methods, which is particularly important in MAC operations where the error can accumulate. The multiplier also has a latency of 1 cycle and provides an overflow flag. Like the adder, the multiplier only updates its output when the `refresh` signal is high.

5.7.3 Divider

The divider is more complicated than the adder and multiplier. It performs bit-wise long-division and has a latency of $N+Q+3$ cycles, where N is the number of integer bits and Q is the number of fractional bits. The divider also provides flags for overflow and divide-by-zero and done/busy status signals. The module start division on an active-high pulse of the `start` signal and provides the result when the `done` signal is high. The divider module is mostly used in the MAC module during computation of the Swish activation function.

5.7.4 Exponential

The exponential module computes the exponential e^x of a fixed-point number x. It uses a combination of the identities of exponential and a Taylor series approximation around zero to compute the exponential. Specifically, the module transforms the exponential as such:

$$e^x = 2^{\frac{x}{\ln(e)}} = 2^z = 2^{\lfloor z \rfloor} 2^{z-\lfloor z \rfloor} \tag{1}$$

The compute can then easily compute $2^{\lfloor z \rfloor}$ as an inexpensive bit-shift operation and $2^{z-\lfloor z \rfloor}$ as a Taylor series approximation. To determine a reasonable number of terms to use for the Taylor series expansion, an accuracy study was ran. Figure 10 shows the relative error of the exponential module as a function of the order of the Taylor series expansion for both fixed-point (Q22.10) approximation and float (64b) approximation. As can be seen, the error decreases with an increase in the order of the expansion. However, for the fixed-point approximation, it converges to a minimum error of $\tilde{0}.992\%$. This is because the quantization of fixed-point dominates the Taylor series error. Therefore, using a 3rd order Tarlor series expansion to appriximate the exponential function is a good balance between accuracy and latency/energy. Note that these error was measured over the input range of [-4, 4]. According to the functional simulation, this corresponds to roughly ± 3 standard deviations from the mean of inputs to the exponential function. To further speed up the computation, the exponential module uses a lookup table to store the Taylor series coefficients as well as $1/ln(e)$. To reduce area, the exponential module does not instantiate its own adder and multiplier modules. Rather, it accesses the adder and multiplier modules in the CiM module shared with other compute units. The latency is 24 cycles.

5.7.5 Square Root

The square root module computes the square root of a fixed-point number using an iterative algortihm. It has a latency of $(N+Q)//2+1$ cycles, where // denotes integer division. The module provides flags for overflow and negative radicand and start/busy/done sig-

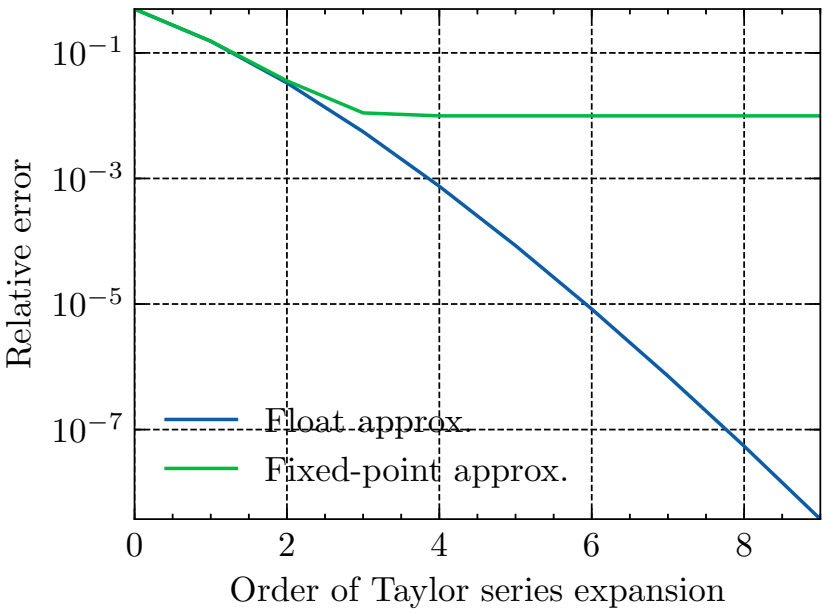


Figure 10: Approximation error of the exponential vs Taylor series expansion order

nals. The module starts computation on an active-high pulse of the `start` signal and provides the result when the `done` signal is high.

5.7.6 Multiply-Accumulate

The MAC module performs a vector dot-product for a given pair of base addresses for the data and length of the vector and applies a selectable activation function to the result. Similar to the exponential module, it uses shared adder, multiplier, divider and exponential modules in the CiM module. It can implement three activation functions: none, linear and Swish. For a nominal length of 64 (which corresponds to the embedding depth of the model, a very common value for matrix dimensions in the model) and Q22.10 formate, the latencies are 386, 391 and 456, respectively. Note that, although the Swish activation function comprises a divider operation, the MAC compute latency can still be kept fairly short because the divisor is the same for all elements. The module can thus perform the division once and multiply by the inverse, which is a single-cycle operation.Finally, the MAC module can be directed to choose the second vector from weights or intermediate results memory.

5.7.7 Softmax

The softmax module computes the softmax function of a vector of fixed-point numbers. Similarly to the MAC module, it uses shared adder, mulitplier, divider and exponential modules and provides `busy` and `done` signals. For a 64-element Q22.10 vector, the latency

is 2024 cycles. This is significantly longer than other vector compute modules such as the MAC because, in the softmax operation, each element is exponantiated individually.

5.7.8 LayerNorm

The final compute module is the LayerNorm module. It computes the Layer Normalization of a vector of fixed-point numbers. As described in section 4, the LayerNorm operation consists of a normalization of the vector on the horizontal dimension followed by scaling and shifting using learnable parameters on the vertical dimension. Because each CiM module stores one vector at a time, the LayerNorm operation must be separated into two stages with a matrix transpose broadcast between the two. The latency for the first half is 1469 cycles and the latency for the second half is 494 cycles. The module provides `busy` and `done` signals and is controlled with a `half-select` and `start` pulse signals. Because the length of the vector is constrained to be a power of two, the module uses bit-shifting instead of division for the normalization operation to decrease latency and energy per operation.

5.8 A Note About Software-Hardware Co-Design

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6 Results: Evaluation of Performance Metrics

This presents and discusses the most saliant high-level results of the model and hardware design.

6.1 Vision Transformer

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6.2 Accelerator

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

7 Future Work

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

8 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

References

[1] Xilin Liu and Andrew G Richardson. "Edge deep learning for neural implants: a case study of seizure detection and prediction". In: *Journal of Neural Engineering* 18.4 (2021), p. 046034.

[2] CEAMS. *SS3 Biosignals and Sleep stages*. Version V1. 2022. DOI: 10.5683/SP3/9MYUCS. URL: https://doi.org/10.5683/SP3/9MYUCS.

[3] Akara Supratak, Hao Dong, Chao Wu, et al. "DeepSleepNet: A model for automatic sleep stage scoring based on raw single-channel EEG". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25.11 (2017), pp. 1998–2008.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. "An image is worth 16x16 words: Transformers for image recognition at scale. arXiv 2020". In: *arXiv preprint arXiv:2010.11929* (2010).

A Bus Operations

This section details the instructions that can be performed on the bus in more details than is warranted in the main body of the thesis. Table VI describes each instruction along with the fields on the bus.

B Codebase Statistics

It may be interesting to the reader to appreciate the size of the codebase needed to develop a project of similar scale. The code for this project is available in my GitHub repository. The following table provides a breakdown of the number of lines of code in the project.

In addition, there have been 200 commits to the repository.

Table VI: Bus operations and their fields

Opcode	Description	Sender	ID	Data[0]	Data[1]	Data[2]
NOP	No instruction	All	-	-	-	-
PATCH_LOAD_BROADCAST_START_OP	Start loading an EEG patch	Master	0-63	tx_addr	Length	rx_addr
PATCH_LOAD_BROADCAST_OP	EEG patch data	CiM	0-63	Data	Data	Data
DENSE_BROADCAST_START_OP	Start dense broadcast	Master	0-63	tx_addr	Length	rx_addr
DENSE_BROADCAST_DATA_OP	Dense broadcast data	CiM	0-63	Data	Data	Data
PARAM_STREAM_START_OP	Start streaming weights	Master	0-63	tx_addr	Length	-
PARAM_STREAM_OP	Weight data	Master	0-63	Data	Data	Data
TRANS_BROADCAST_START_OP	Start transpose broadcast	Master	0-63	tx_addr	Length	-
TRANS_BROADCAST_DATA_OP	Transpose data broadcast	CiM	0-63	Data	Data	Data
PISTOL_START_OP	CiM to execute next step	Master	-	-	-	-
INFERENCE_RESULT_OP	Contains inferred sleep stage	CiM #0	0	Sleep stage	-	-

Table VII: Line and file count per file type in the codebase

File type	File count	Line count	Percent of total
Python	12	3000	33.7%
SystemVerilog	12	2500	30.4%
C++	12	1250	18.9%
TeX	12	670	8.2%
Shell	12	300	4.3%
Other	12	20	4.5%
Total	60	13,000	100%

C Reflection on Learnings and Experience Gained

This page intentionally left blank.