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

© 2024 Tristan Robitaille

Abstract

Insomnia is a wide-spread sleep disorder affecting hundreds of million of people worldwide. New treatments are being developed to address this issue, including neuromodulation, which aims to modify nervous activity in the brain to improve sleep quality. Effective neuromodulation requires live, practical and accurate assessment of the user's sleep stage. The state-of-the-art for sleep staging involves polysomnography, which may require up to 24 sensors, technician supervision and manual annotation by a sleep expert for the most accurate results. Needless to say, polysomnography is not suitable for widespread, frequent, at-home use. Wearable devices, such as in-ear sensors, may be able to provide automatic sleep staging. In this work, we investigate the feasibility of running an AI model in-ear via an ASIC accelerator for automatic sleep staging. This thesis present a vision transformer-based model achieving 82.9% accuracy on the MASS SS3 dataset with a model size of 31.59kB. It also develops an ASIC accelerator to measure the power, area and latency cost of running such a model. The accelerator consumes 3.046mW on average, has an area of $3.86mm^2$ and an inference latency of 6.97ms. The results show that the accelerator is viable for in-ear use in all aspects except area. Further work is suggested to reduce the power consumption and area of the accelerator by 72.2% and 54.8%, respectively.

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

Note: The code is available on my GitHub repository: TristanRobitaille/engsci-thesis

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 prototype 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 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 course, 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.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog-to-Digital Converter

AFE Analog Front-End

AI Artificial Intelligence

ASIC Application-Specific Integrated Circuit

BVP Blood Volume Pulse

CiM Compute-in-Memory

CMOS Complimentary Metal Oxide Semiconductor

CSV Comma-Separated Values

ECG Electrocardiography

EEG Electroencephalography

EMG Electromyography

EOG Electrooculography

FSM Finite State Machine

GSR Galvanic Skin Response

HDF5 Hierarchical Data Format 5

IP Intellectual Property

ISA Instruction Set Architecture

MAC Multiply-Accumulate

MASS Montreal Archive of Sleep Studies

MHSA Multi-Head Self-Attention

MLP Multi-Layer Perceptron

nMOS N-Channel Metal Oxide Semiconductor

PE Processing Element

PPA Power, Performance and Area

PSG polysomnography

RTL Register Transfer Level

STAGES Stanford Technology Analytics and Genomics in Sleep

TPU Tensor Processing Unit

TSMC Taiwan Semiconductor Manufacturing Company

VCD Value Change Dump

RNN Recurrent Neural Network

CNN Convolutional Neural Network

DNN Deep Neural Network

1 Introduction

As reported by Chaput *et al.* [1], insomnia impacts around 24% of Canadians adults. Detection and classification of sleep stages, known as *sleep staging*, followed by neuromodulation has been recently found by Yoon [2] to be a promising treatment against insomnia. The current stage-of-the-art for sleep staging involves the use of polysomnography to measure biosignals (at least 19 sensors are required, as explained by Levin and Chauvel [3]) and manual annotation by a sleep expert, which requires, on average, 2 hours of work [4]. This technique also does not provide neuromodulation. To address these downsides while effectively treating insomnia, we propose an in-ear device performing Electroencephalography (EEG) sensing, sleep staging and neuromodulation. To maximize treatment potential, the device should be as small and portable as possible such that it can be used at home.

This thesis focuses on the development of a deep learning model to perform sleep staging and on the design of an accelerator ASIC module to perform in-situ inference with said model. In the end, it aims to prove, by simulations, the viability of such an accelerator in order to potentially integrate it in the in-ear device. Multiple authors [5]–[7] have published high-accuracy results using a deep learning approach to sleep staging, and have done so with significantly fewer sensors than polysomnography. However, these AI models run on standard computers as software frameworks and are thus unsuitable for a lightweight, integrated solution. Google sells small custom AI-accelerators (such as the Coral Edge TPU) that could run these AI models, but they still consume too much power (at least 1W, [8]) and do not readily integrate with custom neuromodulation hardware.

The proposed solution should match the accuracy of traditional polysomnography and published models in the literature with a power consumption low enough that the whole system can be powered for at least a full-night on a battery that fits in-ear. The silicon area of the accelerator should allow it to fit, along with the rest of the system, on an integrated circuit fit for integration in an earbud-type device. This document provides an overview of the literature in both automatic sleep staging using Artificial Intelligence (AI) and in Application-Specific Integrated Circuit (ASIC) accelerator design, establishes technical requirements for the device, describes the model used and hardware design, evaluates their performance and discusses improvements and future work.

2 Problem Statement and Technical Requirements

In light of the background of Section 1, the problem statement of this thesis is to develop a deep learning model and an ASIC accelerator for automatic sleep staging in order to determine whether such a system can be used in an earpiece-type device for automatic sleep staging. The results of this thesis will guide future development in the field.

Table I indicates precise design goals and their justification, which helps guide design decisions and development effort. For example, to reach the target model size, time will be spent evaluating the impact of hyperparameters to find the combination that gives the lowest size while meeting the desired accuracy. For the AI accelerator, since inference power and clock frequency are inversely proportional, we must focus on reducing energy per inference. From first principles, this implies reducing the amount of charge that is displaced within the chip. Since the physical properties are locked for the target 65nm node, we focus on reducing the number of operations, simplifying operations, limiting data movement and reducing control logic.

To determine the average power consumption constraint, the battery capacity of the Airpods Pro, which was found in a teardown to be 0.16Wh, can be considered as a reference [9]. Assuming a goal of 10h of battery life (for a full night of sleep), the overall power consumption must stay below 16mW, on average. The speaker coil driving circuitry will consume most of the power and, leaving enough power budget for the analog front-end and overhead in the system, in addition to a safety factor, 5mW seems like a reasonable target for the accelerator. Regarding the target area and model size, a reasonable chip package for this application is a 4mm x 4mm Chip-Scale Package (CSP). According to IPC standard J-STD-012, a CSP overall area is no more than 120% of the die, which leaves $13mm^2$ of die area [10]. To further contextualize, Apple's H1 processor, which is used in the Airpods Pro, has a similar die area of $12mm^2$ [11].

Again, a significant portion will be consumed by the coil driver, RISC-V processor and memory. Therefore, a maximum area of 15%, or $2mm^2$, for the accelerator seems to be a safe option. According to Liu and Kursun, the area of a standard 6T SRAM cell is $0.75\mu m^2$ [12]. To leave enough area for the compute elements, controllers and routing, reserving $0.75mm^2$ of the accelerator area to weights is reasonable. This provides up to 1Mb, or 125kB, ignoring memory access overhead, for the weights of the model. Finally, as mentioned above the accuracy should be close to published literature in this field to provide effective treatment. For this, 80% is a reasonable goal. Current polysomnography (PSG) divides the night in 30s "sleep epochs". To limit too much of an offset between the end of a 30s epoch and its predicted sleep stage (essentially a phase offset), which would add inaccuracy to the overall system and potentially reduce its effectiveness, a maximum phase offset of 10% (3s) is appropriate. Finally, a clock of 200MHz is a typical upper-bound for low-power microcontroller-type systems. To avoid needing two separate

Table I: Design goals for AI model and ASIC accelerator

Type	Goals	Justification
Model	Size $< 125\,\mathrm{kB}$	Help reach ASIC area/power goals
	Accuracy $> 80\%$	Competitive with state-of-the-art
ASIC	$P_{\mathrm{avg}} < 5\,\mathrm{mW}$	System to function for whole night
	$A_{\mathrm{total}} < 2\,\mathrm{mm}^2$	Fit in ear (65nm node)
	$T_{\mathrm{inference}} < 3\,\mathrm{s}$	Maximum phase offset of 10%
	$f_{\mathrm{Max}} > 200\,\mathrm{MHz}$	Compatibility with rest of system

clock domains, which entails more area and power, the accelerator should be able to run at 200MHz.

3 Literature Review

3.1 Machine Learning for Sleep Staging

Deep learning for sleep staging has been studied since around 2017. Broadly speaking, basic Deep Neural Network (DNN) cames first, followed by Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) [13]. The transformer is a relatively new type of neural network based around the concept of "attention" and particularly suited for sequence inputs since it can process the input in parallel [14]. Since its introduction in 2017 [15], the transformer has been used for sleep staging tasks. Indeed, Dai *et al.* developed a transformer-like model without decoders which used three input EEG channels and achieved 87.2% accuracy on the popular SleepEDF-20 dataset [16]. Similarly, Phan *et al.* developed a model with a focus on outputting easily-interpretable confidence metrics for clinicians. They found that a significant impediment to the adoptation of automatic sleep staging in clinics is lack of trust from clinicians as they perceive the system to be a "black box". Their model ingests multiple sleep epochs for each inference, which allowed the team to achieve 84.9% accuracy on the SleepEDF-78 dataset. Eldele *et al.* managed an accuracy of 85.6% on SleepEDF-78 using a single-channel, single-epoch attention-based model [7].

In recent years, the accuracy of sleep staging by ML models has plateaued. In fact, Phan *et al.* claim that AI-based sleep-staging in heathly patients has been solved fully as the accuracy has reached the "almost perfect" level of Cohen's kappa [4]. However, none of the models presented above meet our constraints. Indeed, we require a lightweight, single-channel, single-epoch model. Most models have more than 1M parameters [13]; even the smallest model by Eldele *et al.* has above 500k 32-bit float weights, which far exceeds the 125kB constraint. Furthermore, none have been optimized to run on custom hardware. Thus, there is a need to develop a novel lightweight transformer.

3.2 AI Accelerator Hardware

AI accelerators are a very active area of research at the moment. Significant gains in latency and power are possible by designing hardware optimized for machine learning. Indeed, CPUs lack in memory bandwidth and parallelism and have significant control overhead as required to process any arbitrary program. GPUs have high cost and high power consumption and typically have less available memory than a CPU. Hardware (FPGA and ASICs), on the other hand, can be fast and energy efficient but suffer from limited flexibility [17]. The first AI accelerator ASIC, publised in 2014 by Chen *et al.* and named *DianNao*, could perform 452G 16b fixed-point computations per second [18]. It was quickly surpassed by *ShiDianNao*, which was 1.87x faster and used 60x less energy [19].

Modern models are massive and suffer from memory access penalties. Thus, the most optimized architectures limit data movement as much as possible by placing the compute elements directly *in memory*, meaning that less charge is switched per operation (DRAM access requires 200x more energy than on-chip memory [20]) and the overhead and bottleneck effect of the data bus is drastically reduced. For example, Arora *et al.* describe "CoMeFa", a Compute-in-Memory (CiM) module for FPGA BRAM which managed to reduce energy comsumption by 55%. They placed up to 160 single-bit processing element per BRAM (20kbit) and perform operations in a bit-serial manner instead of the traditional bit-parallel [21]. Similarly, Wang and colleagues presented a similar BRAM CiM module which sped up compute by up to 2.3x at the cost of 1.8% increase in area [22]. A high-level architecture that often uses a number of CiM is known as "dataflow". Unlike the traditional von Neumann architecture, it does not rely on instructions and banks of memory cells to perform the necessary computation, but instead organizes different compute elements sequentially and in parallel to match a model's architecture. For instance, Farabet *et al.* present a runtime-reconfigurable dataflow architecture based on nine pipelined "processing tiles" that can be reconfigured in $\sim 10^4$-10^5 permutations [23].

Another area of research is concerned with the software-hardware co-design opportunities afforded by tight integration between the model and the hardware running its inference. Ding and team describe an impressive design where both the training backpropagation and inference hardware are modified. By ensuring that weight matrices consist of an array of circulant matrices, the team is able to reduce time-complexity of vector-matrix mutliply from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. This technique reduced storage needs by 400x-4000x, enabling the model to be stored in on-chip memory and reducing energy consumption by 60x-70x compared to naïve FPGA implementation [20]. Taking a different route, by developing a so-called "Block-Balanced Pruning" technique to pack a pruned weight matrix and its index information in an FPGA BRAM, Qi *et al.* managed to double inference speed compared to a GPU [24]. Another example of software-hardware co-design is the work by Zhi *et al.* who developed a "Clipped staircase ReLU" activation function optimized for their custom CiM processing element and use of quantized weights. Their work consumed 5x less energy and 100x-1000x fewer memory accesses [25].

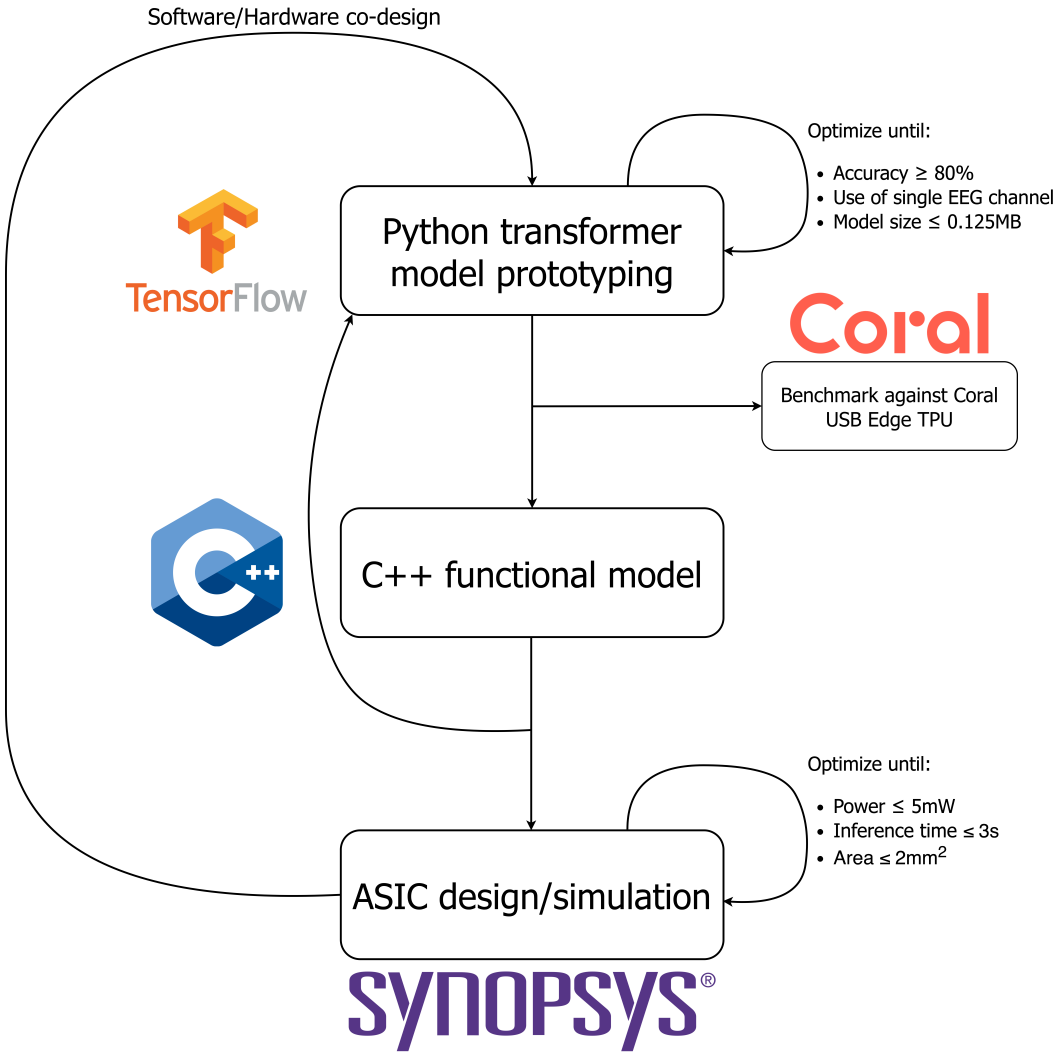
3.3 Lessons From The Literature

The literature reviews presented above offer promising design directions to help reach the design goals. Firstly, a small model is critically missing from the literature since even the smallest model is 16x too large for our project. Secondly, CiM and dataflow architectures are proven techniques to decrease power consumption and offer promising inspiration for the design of the ASIC. Finally, good software-hardware co-design should not be overlooked as it can offer significant gains.

4 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 subsections. 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 1, these three steps allow iterations to converge on a design that meets the requirements described in Section 2.

Figure 1: Three step workflow for designing an AI accelerator



4.1 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 Tensor Processing Unit (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 PSG recordings with 21 EEG channels [26]. 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 [27]).
- 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 Compute Canada's 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. Table II shows the hyperparameters used for training the model. These have been empirically determined to yield the best accuracy with reasonable training time. Figure 2 shows the accuracy of the model as a function of the number of epochs, which is shown to converge at around 100 epochs. Furthermore, Figure 3 indicates that a dropout rate of 30% is optimal for this model.

4.2 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

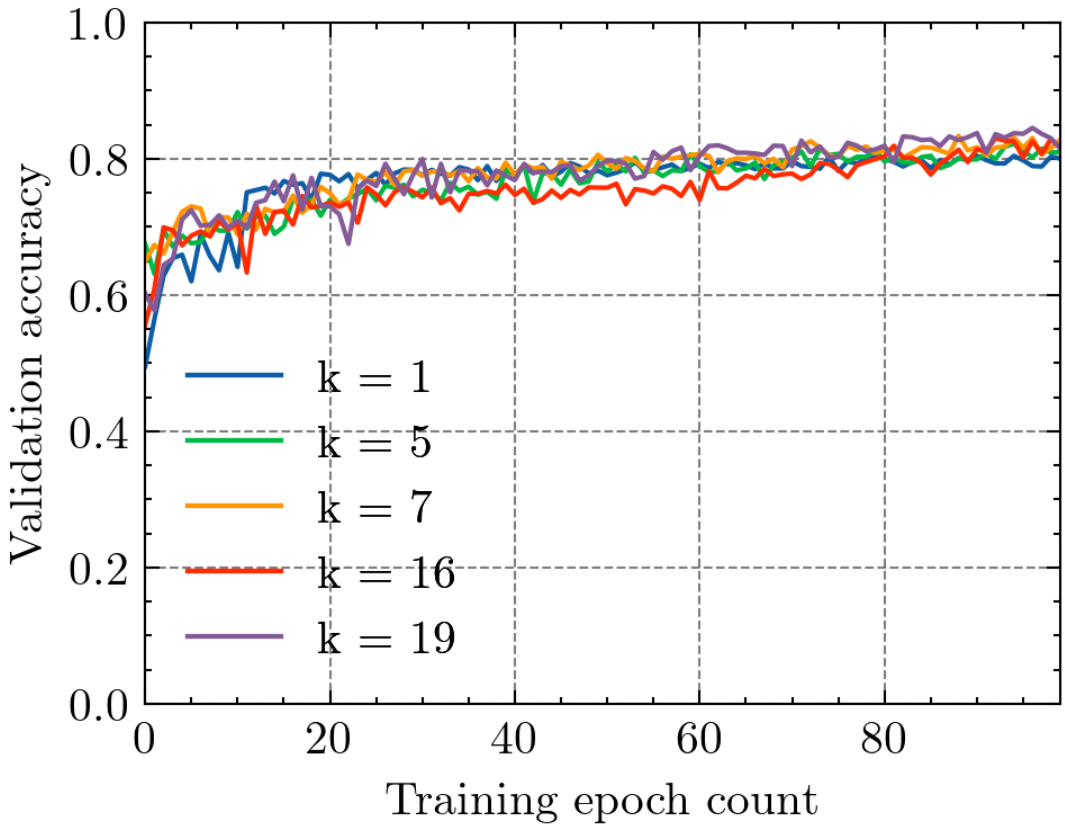


Figure 3: Testing set accuracy as a function of dropout rate during training

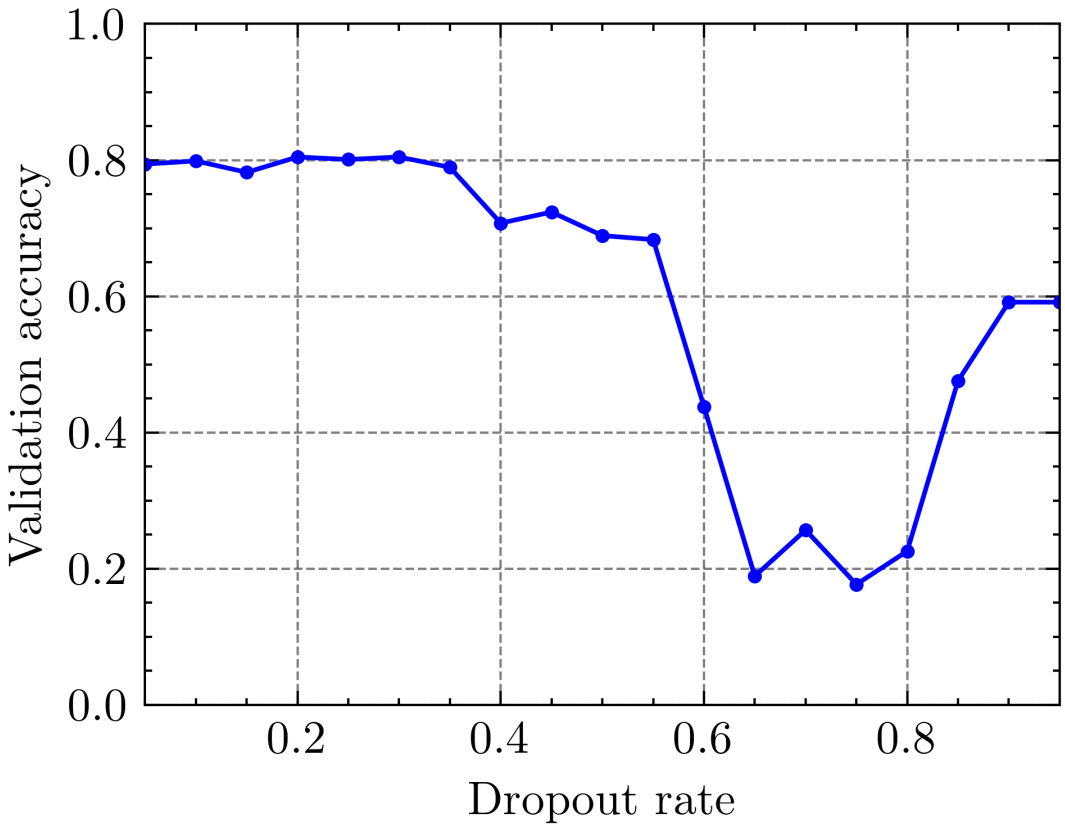


Table II: Training hyperparameters for vision transformer model

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, with the aim of helping subsequent SystemVerilog development, uses a similar structure to hardware coding (cycle-level parallelism, use of FSM, limited function calls). It is organized identically to the hardware design, with a Master module that 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. The functional simulation is used to collect metrics that are more difficult to measure in high-level software or 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), `rapidcsv` for Comma-Separated Values (CSV) file I/O (storing fixed-point accuracy study results and layer output from the TensorFlow model) and `fpm` for fixed-point math.

4.3 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 interfaces, typedefs, 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.

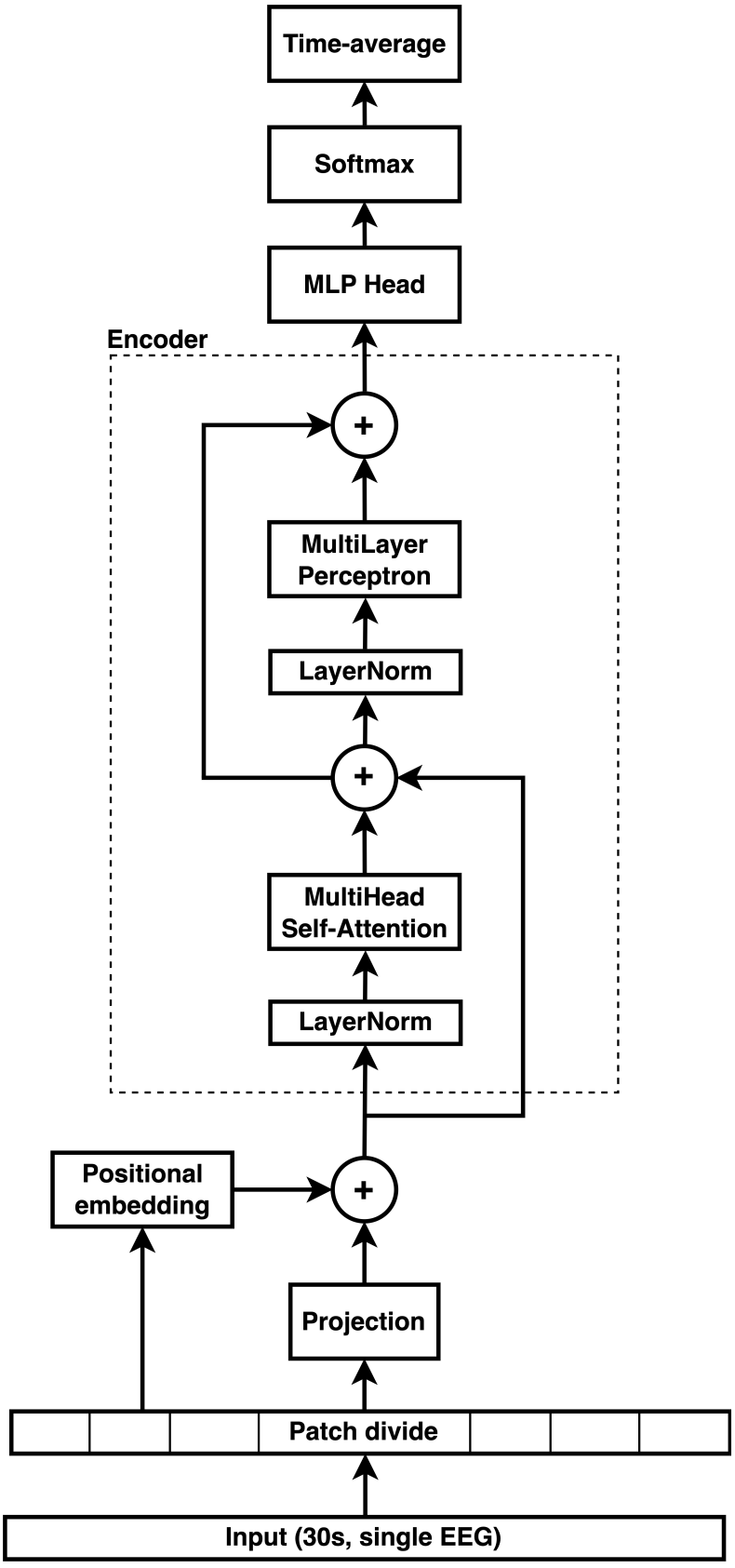
The first three tools are open-source and compatible with all major operating systems, providing a familiar, local and OS-agnostic development environment. Testbenches are written individually for each of the compute modules discused in Section 6.7 where multiple constrained-random inputs are passed to the modules and their output is compared, within tolerance, to a reference values computed in software. The Master and CiM modules are tested individually with a testbench that emulates the signals and timing that each expect from the other.

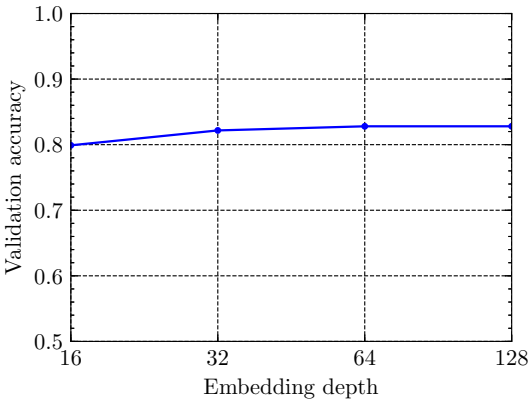
The last two tools are proprietary and are used to evaluate the performance of the design against requirements.

5 Vision Transformer Model Design

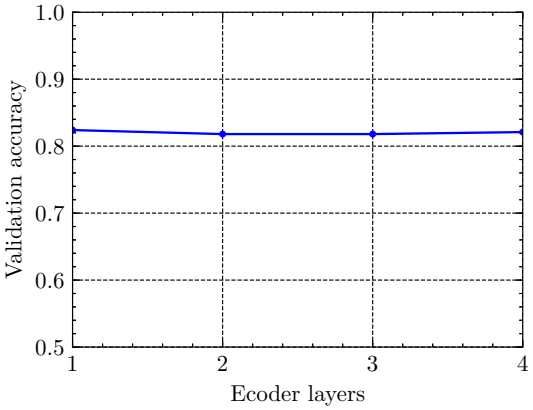
This section describes the architecture of the vision transformer model used in this thesis, which is shown Figure 4. Since sleep staging as presented here is a "sequence-to-one" problem, feedback is not applicable and thus the decoder stack present in a more traditional transformer is not needed. The patch divide step and first dense step (known as "patch projection" in [28]) 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 6, 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 embedding depths of 32 or 128 (see Figure 5a) but offers lower energy consumption when paired with 60 patches due to increased utilization. This is discussed further in Section 6.9. The model then applies a learned 1D positional embedding to the patches to allow the model to learn positional infromation of the EEG stream. The model then applies a single encoder layer consisting of Multi-Head Self-Attention (MHSA) and Multi-Layer Perceptron (MLP) layers. 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 LayerNorm layers first normalize the inputs to a Gaussian over the second dimension, and then scale and shift the normalized values using learnable parameters (γ and β, respectively) 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 layers are 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 III summarizes the hyperparameters of the model.

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

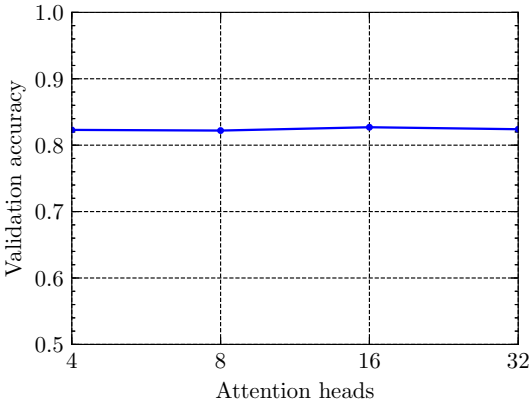




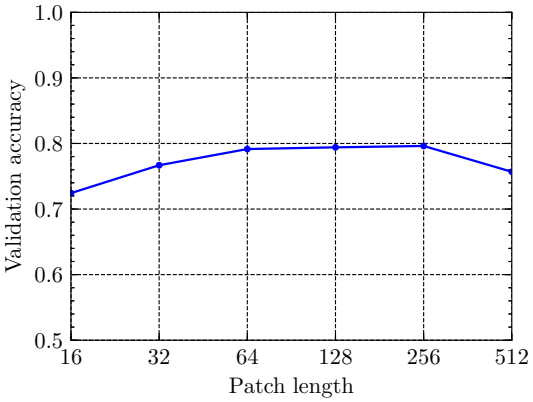
(a) Accuracy vs. `d_model`



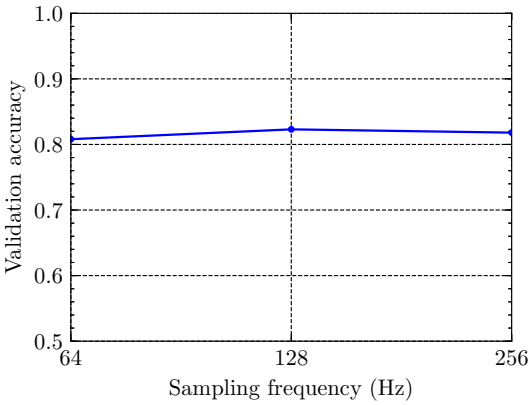
(b) Accuracy vs. encoder layer count



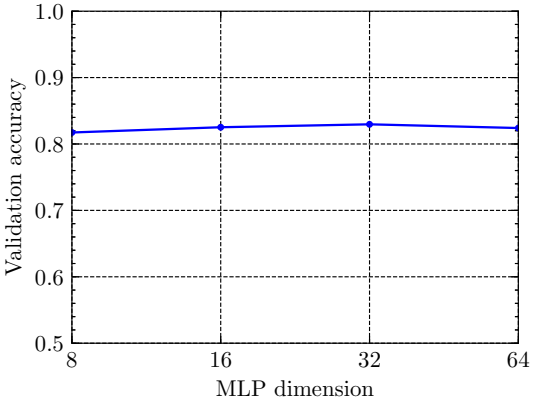
(c) Accuracy vs. number of heads



(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 III: 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

6 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, 64 so-called CiM modules that perform the actual computation and a shared bus for data and control signals.

6.1 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 set of relatively large memory banks, 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 Multiply-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 as mentioned in Section 6.9

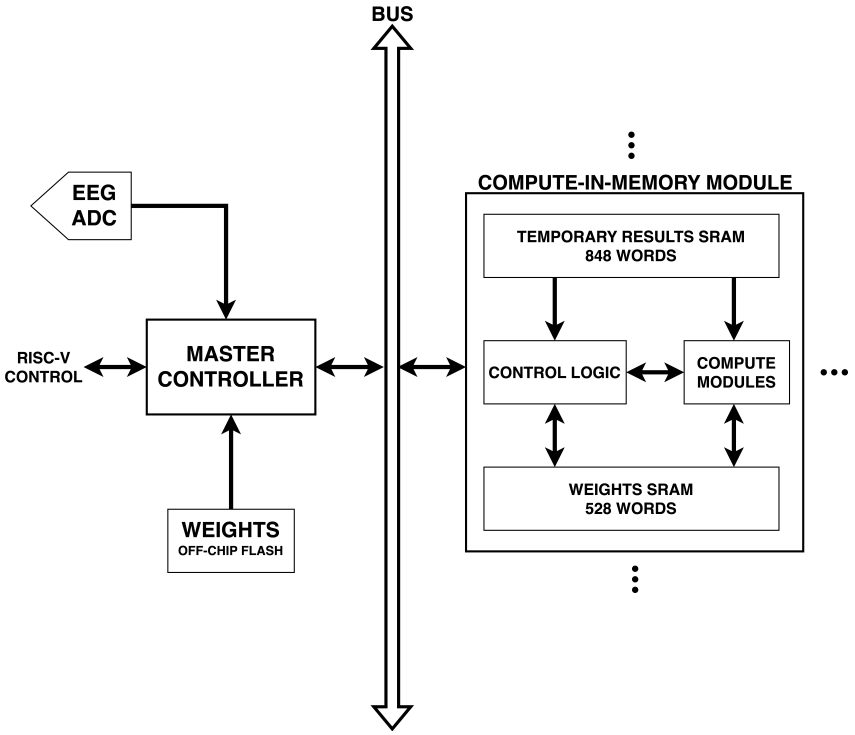


Figure 6: High-level architecture of the ASIC accelerator

6.2 Data and Control Bus

The data and control bus is a bidirectional bus that connects the Master module and all the CiM modules. One device may communicate at a time 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 IV. 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, or vice-versa). The CiMs are autonomous with these operations. They do not need involvement from the Master other than to start the broadcast. Synchronization is maintained with the `START_PISTOL` broadcast, which instructs the CiMs to go to the next high-level step in their Finite State Machine (FSM).

Table IV: 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)

6.3 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.

6.4 Compute-in-Memory: Architecture

The CiM module is the heart of the accelerator. Its architecture is presented in Figure 7. 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 across CiM. The following sections describe different aspects of the CiM module.

6.5 Compute-in-Memory: Memory

The CiM module contains two single-port memory banks: one for intermediate results and one for model weights 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 a 65nm 6T SRAM cell measuring 0.525µm^2. It has an aspect ratio of 4, which, as seen in Figures 8 and 9, provides the lowest overhead for the given capacity. The memories offer single-cycle latency. They can also be switched into a low-power mode when not in use. Table V shows various metrics of the memory banks for operation at 1.0V and 25°C.

Table V: Area, leakage power and cycle of the SRAM memory banks

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

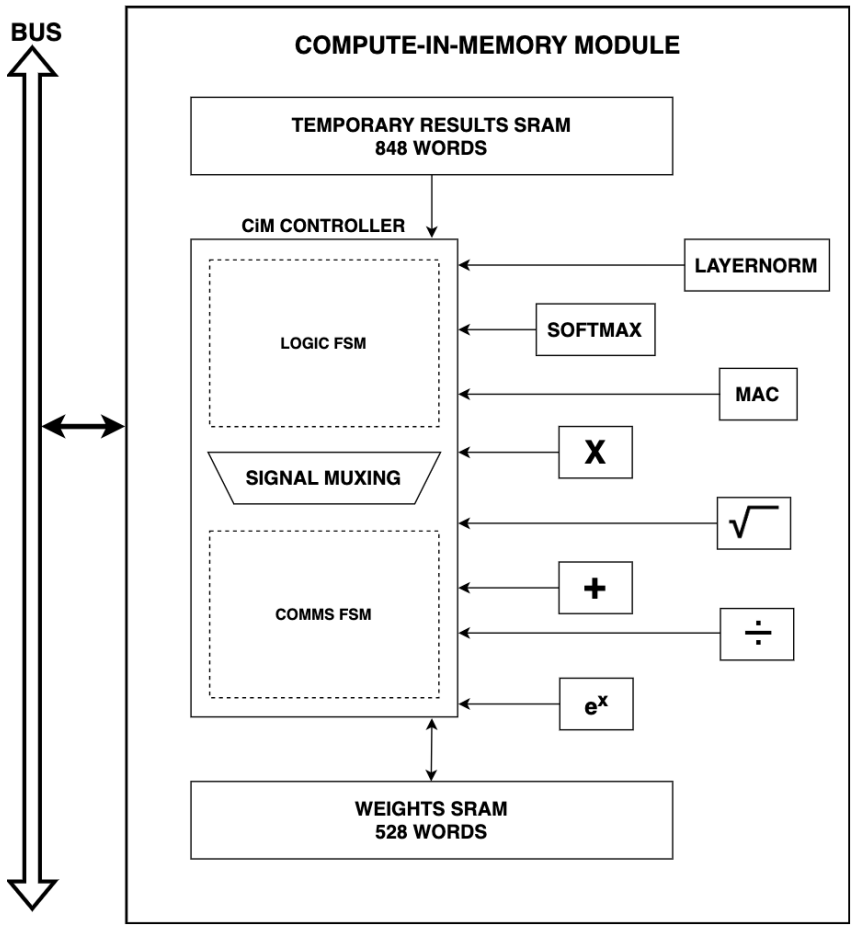


Figure 7: Architecture of the Compute-in-Memory module

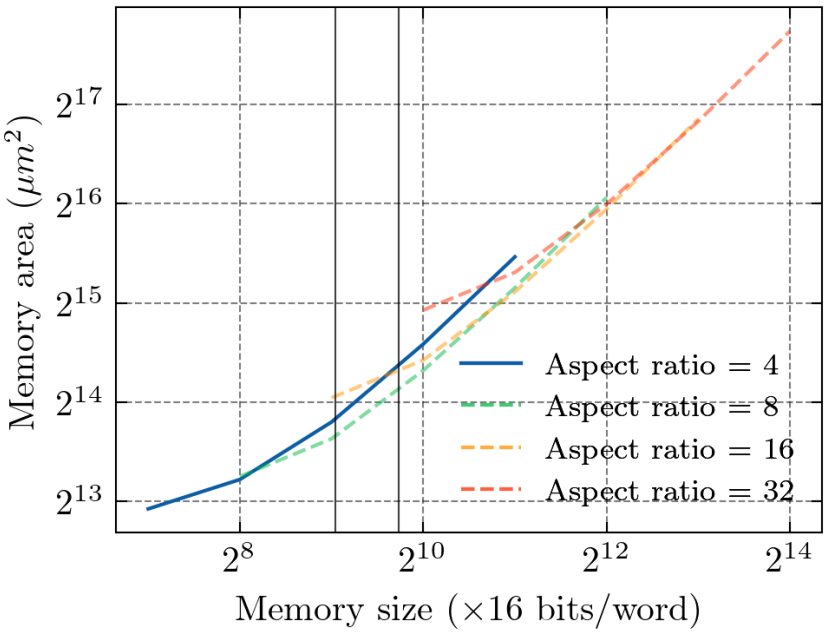


Figure 8: Area of memory banks vs capacity for different aspect ratios

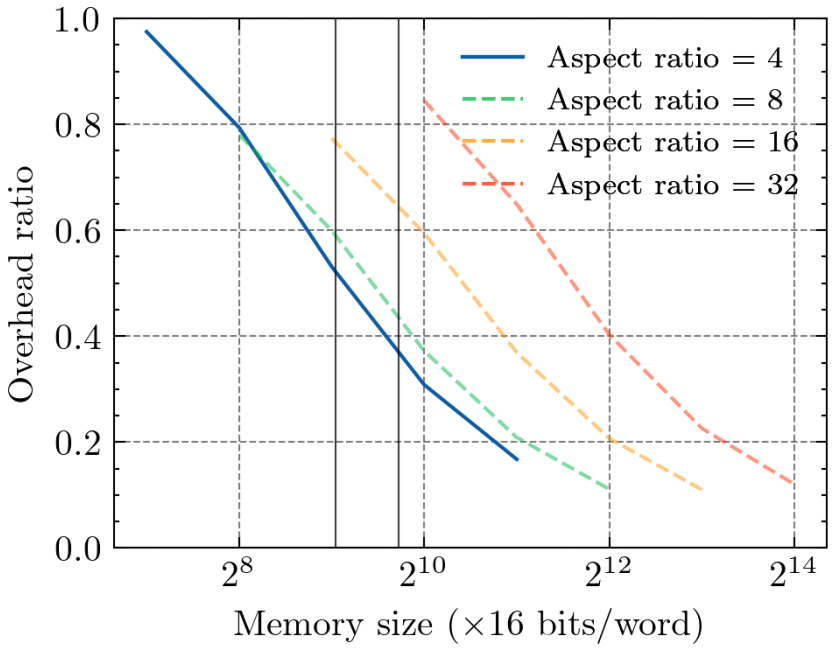


Figure 9: Overhead of memory as a fraction of total area for different aspect ratios

6.6 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 for temporary results internal to the compute modules and Q6.10 for storage. 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 10 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. Although the ideal number of fractional bits is 8, the design uses 10 fractional bits to provide a small margin of error and guard against divides by zero with other input nights.

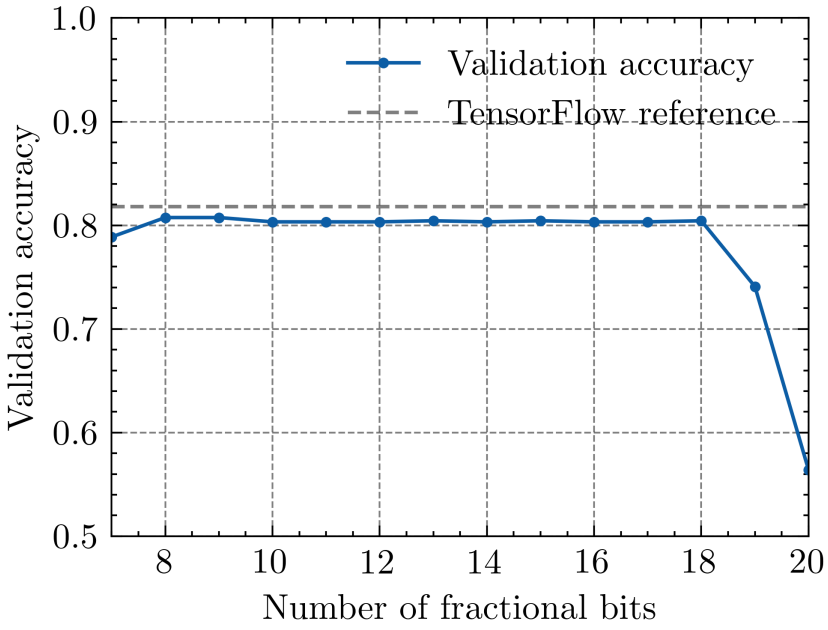


Figure 10: Model accuracy vs number of fractional bits in fixed-point format

6.7 Compute-in-Memory: Compute Modules

This section describes the design and performance metrics of the various compute Intellectual Property (IP) modules used in the design. 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 VI shows the performance metrics of the compute modules. The working principles of each modules is described briefly in subsequent sections.

Note that all measurement in Table VI are given for standard 65nm Taiwan Semiconductor Manufacturing Company (TSMC) process. 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 1000 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 normalizing the total energy used by the number of operations performed in the testbench.

Table VI: 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	21	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

- Leakage power: Synthesis with the power optimization effort set to `high`, and the leakage power was extracted from the `report_power` command report.
- F_{max}: The `report_timing` command was used to determine the maximum frequency of the design.

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.

6.7.1 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.

6.7.2 Multiplier

The multiplier is very similar to the adder. One difference is that it uses Gaussian rounding (also known as banker's rounding). This method rounds up or down in an alternating fashion. This reduces the bias in the output that is commonly observed

with standard rounding methods, which is particularly important in MAC or LayerNorm 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.

6.7.3 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.

6.7.4 Exponential

The exponential module computes the natural exponential, e^x, of a fixed-point number x. It uses a combination of the identities of exponentials 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 module 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 11 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 (64-bit) 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 0.992%. This is because the quantization of fixed-point dominates the Taylor series error. Therefore, using a 3rd order Tarlor series expansion to approximate the exponential function is a good balance between accuracy and latency/energy. Note that this 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.

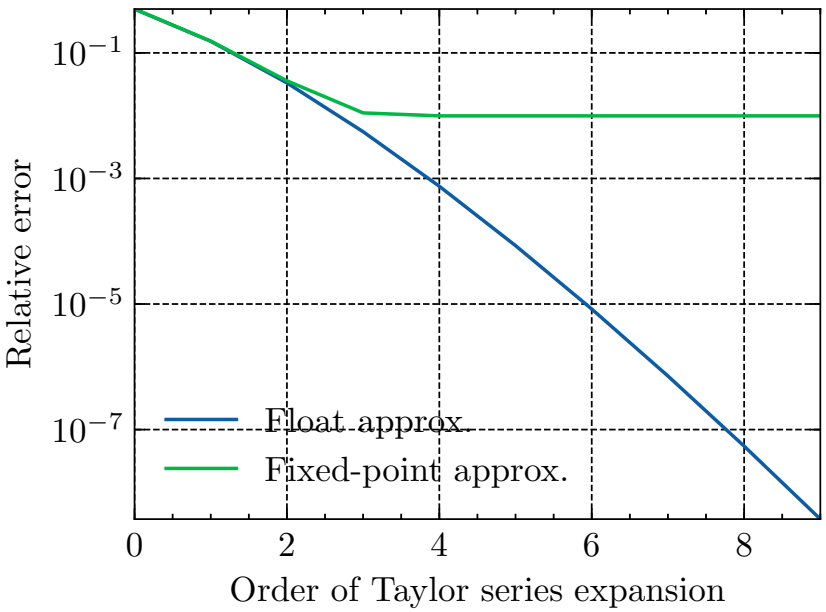


Figure 11: Approximation error of the exponential vs Taylor series expansion order

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

6.7.6 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. Similarly 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. As reported by Ramachandran *et al*, Swish is similar to SiLU and helps resolve the vanishing gradient problem in backpropagation, leading to a higher accuracy for a given number of training epochs [29]. 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 format, 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.

6.7.7 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.

6.7.8 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 5, 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.

6.8 Clock Gating

This design makes use of clock gating, which is a technique to reduce dynamic power consumption by disabling the clock signal to modules that are not in use. This is handled automatically by the Synopsys Design Compiler tool.

6.9 A Note About Software-Hardware Co-Design

To maximize utilization (and thus time- and physical efficiency) and reduce latency (thus reducing inference energy), the model and accelerator were designed together. An example of this software-hardware co-design concerns the number of CiM in the accelerator. 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 CiMs, all vector operations can be computed simulatenously, avoiding extra overhead for control and data movement while limiting the amount of unused silicon. This is the main reason behind the choice of 64 samples for the patch length, which also yields 60 patches per sleep epoch. Furthermore, the embedding depth of the model is a power

of 2, meaning that all divisions (such as in the LayerNorm) with this number can be accomplished with inexpensive bit shifts.

7 Results: Evaluation of Performance Metrics

This Section presents and discusses the most salient aggregate high-level results of the model and hardware design as shown in Table VII. It can be seen that the model meets the requirements summarized in Table I. Indeed, the model size (63.18kB) is below the 125kB constraint, and the accuracy (82.9%) is higher than the 80% target. It is also nearly 32× smaller than the smallest state-of-the-art model for automatic sleep staging presented by Eldele *et al.*, which has 500,000 32-bit floating point parameters [7].

The ASIC accelerator meets some of the requirements. The inference latency (6.97ms) is well below the maximum allowed latency of 3s. The PE utilization (50.8%) of the accelerator suggests that there is non-negligible time overhead (mostly from inter-CiM communication) that could be reduced in future work. The PE utilization computes the amount of time that any one compute module is busy or refreshing its output relative to the total inference time. The toral area is 3.86mm^2, which is above the target of 2mm^2. The effective power of 3.046 mW is below the ceiling of 5mW.

The effective total power considers the effects of power gating, which is a common technique used in modern processors to reduce power consumption. The idea is to turn off power to the ASIC accelerator when it is not in use through one or more low-side N-Channel Metal Oxide Semiconductor (nMOS) transistor (known as a "sleep transistor"). Investigating this technique for 65nm Complimentary Metal Oxide Semiconductor (CMOS) technology, Sathanur *et al.* found that leakage power can be reduced by up to 95% using power gating [30]. The effective total power is calculated as the sum of the average dynamic power from inference energy and the leakage power pro-rated with power gating. It assumes a sleep epoch duration of 30s. The average dynamic power is computed for an inference with the techniques of analyzing a VCD file as described in Section 6.

With an inference latency of 6.97ms and a sleep epoch duration of 30s, the ASIC accelerator only runs for 0.0232% of the time, meaning that the effective leakage power is reduced by up to 94.98%, reducing the effective leakage power to 2.947mW. The accelerator consumes a total of 63.07mW during inference, which brings the effective power consumption to 3.046mW with power gating. The energy per inference is 433.29µJ. The f_{max} is 758MHz, which is above the target of 200MHz. This gives us ample leeway to modify the design to reduce power consumption or area further. The f_{max} is limited by the multiplier module as shown in Table VI.

7.1 Comparison Against the Coral Edge TPU

The Coral Edge TPU is a low-power USB accelerator designed by Google optimized for inference. It consumes up to 2W and can process up to 4TOPS [8]. It represents one of the AI accelerator with the lower power consumption available in the market, and as

Table VII: Principal results of the model and hardware design

Metric	Value	Meets requirement?
31-fold accuracy	82.9%	Yes
# of parameters	31,589	Yes
Size	63.18kB	Yes
Inference latency	6.87ms	Yes
Area	3.86mm^2	No
PE utilization	50.8%	N/A
Leakage power	56.48mW	N/A
Average dynamic power	6.59mW	N/A
Effective total power	3.046mW	Yes
Energy/inference	433.29µJ	N/A
f_{Max}	758MHz	Yes

such is a good reference point for comparison. The TensorFlow model was converted to a TensorFlow Lite model and run 2000 times on the Coral Edge TPU to compare the results. The mean inference time was 0.754ms with a standard deviation of 0.038ms. Although the Coral Edge TPU is faster than the ASIC accelerator, it consumes too much power to be used for this project. Furthermore, the integrated circuit measures roughly 5mm by 5mm, which is too large for the target application. It also doesn't not readily integrate with the rest of the system, such as the AFE for sensing, the external memory for weight storage or the neuromodulation coil driver.

8 Results Analysis & Future Work

Based on the design and results discussions of Sections 5, 6 and 7, the design as presented is not recommended to be used further for the sleep staging earpiece project. This section discusses suggested improvements and future work to maximize the utility of this project. Tables VIII and IX summarize the suggested improvements and attempts to quantify their benefits wherever reasonable.

8.1 Vision Transformer Model Improvements

The vision transformer can be improved in different ways, from training to simplification of operations. Firstly, the model could be pre-trained on the Stanford Technology Analytics and Genomics in Sleep (STAGES) dataset [31], which contains PSG recordings from 1500 patients, significantly more than the 62 nights available in the dataset (MASS SS3 [26]) currently used. Although MASS is widely used for model benchmarking, it is worthwhile to try pre-training the model on STAGES and perform transfer learning and k-fold validation on the MASS dataset. In addition, the model should be trained and validated on the Expanded Sleep-EDF dataset, which is the most widely used dataset in the literature [32]. Once the annotated EEG data measured in-ear is available, the model will need to be trained on the new dataset. Weight pruning could further reduce the size of the model. Pruning consists of eliminating a subset of weights to reduce model size and compute with minimal loss of accuracy at the cost of more complicated control logic. In some cases, pruning may also increase accuracy. Indeed, Chen *et al.* applied 50% pruning on DeiT-Small, a vision transformer model, and observed a 0.28% accuracy boost. Furthemore, it would be beneficial to eliminate the γ and β learned parameters used to scale and shift the normalized values in the LayerNorm steps. These add six broadcast transpose operations to the inference, which represents 9.69% of the total inference time and effective total power. Finally, it would be beneficial to explore an architecture that uses more than one EEG electrode or different input signals such as heartrate or temperature as a means of increasing accuracy further. Most models in the literature use 2-5 electrodes [33], [34]. This improvement is particularly relevant given the earbud form factor which can accommodate multiple contact eletrodes and other biosignals. Indeed, in the summer of 2023, Apple was granted a patent for an in-ear "biosignal sensing device" covered with different types of electrodes such as Electromyography (EMG), Electrooculography (EOG), Electrocardiography (ECG), Blood Volume Pulse (BVP) and Galvanic Skin Response (GSR) [35]. This modification will undoubtedly increase the area and latency, and an evaluation of whether the potential increase in accuracy is worth it should be performed.

It would also be important to rewrite the model in efficient C code and run on a microcontroller to determine how far from targets a firmware approach might be. Multiple

Table VIII: Suggested improvements to the model and their potential benefits

Improvement	Accuracy Δ	Latency Δ	Area Δ
STAGES and Expanded Sleep-EDF	Unknown	No change	No change
Weight pruning	Unknown	↓	↓
γ and β elimination	Unknown	0.666ms/-9.69%	No change
Additional channels and electrodes	↑	↑	↑

authors have published relevant work for the RISC-V Instruction Set Architecture (ISA), such as an extension to handle vector operations [36] or compilation support for a RISC-V + CGRA heteregenous architecture [37]. Running the model on a microprocessor would save the significant development and verification effort of an ASIC accelerator and allow for a more flexible design since the model could be changed with a firmware update. However, the power consumption and area of a vector- or CGRA-augmented microcontroller might be higher than the ASIC accelerator.

8.2 Accelerator ASIC Improvements

The bulk of suggested improvements to this project are in the hardware design. It can be made much smaller and more power-efficient. The first recommendation is to use a centralized architecture for the memory. This will save significant area and leakage power mainly from the reduction of memory overhead and deduplication of area dedicated to storing addresses. Specifically, for centralizing memory alone, Figure 9 shows that, using two banks holding 15,794 words (enough to contain the full model) and 4 banks for the intermediate results will reduce the memory overhead from 49.5% (weighted average of the overhead of the parameters memory and the weights memory in the current CiM architecture) to 12.0%, saving 0.931mm^2. The total leakage will also decrease by 13.75mW. Centralizing the memory will also slightly reduce the inference latency since some compute elements that operate on two intermediate results, such as some configurations of the MAC will be able to load the data in parallel. However, this reduction is expected to be small.

The second recommendation is to use a centralized compute architecture with a single instance of each of the compute elements of Section 6.7. According to Table VI, this will save 1.183mm^2 of area and 25.80mW of leakage power. The dynamic power will also decrease slightly due to reduction in wiring, although an exact figure is hard to estimate without an implemented design. Although inference time will increase, it will remain far under the requirement of 3s. In fact, the inference time is expected to be less than 64 times longer than the distributed architecture. Indeed, as mentionned in Section 7, the CiM spends roughly 50.8% percent of its time in compute. The rest is mainly data broadcast.

In data broadcast, the majority of the time (4/7 cycles) is spent waiting for data from the single-port memory. With the centralized memory banks and a careful addressing scheme, the inference time is expected to increase by $64 \times (1 - 0.508 * 4/7) = 45.42\times$ or 0.312s.

The complete centralization of the memory and compute will eliminate the need for the bus. This will save area and power, although the exact amount is hard to estimate without an implemented design. Finally, it may be interesting to explore the impact of using a different Q format for the weights and intermediate results at different layers in the model. The current design uses a constant Q6.10 format for stored data. It may be possible to use fewer bits overall if the Q format can be changed during inference for layers that are known to produce large numbers. This can save storage area and shrink the compute units; however, it will increase the complexity of the control logic and require storing the Q format to use for each layer. The net impact is hard to estimate without a design, and is expected to be fairly small so this recommendation should not be prioritized. For this, the functional simulation of the accelerator should be extended to perform the study. Shifting away from the `fpm` library to a more flexible fixed-point library or writing one from scratch to properly emulate non-powers-of-two bitwidths will be necessary to implement this change.

A key insight from Section 7 is that the leakage power represents 96.7% of the effective power consumption. A key factor to reducing effective power consumption is power gating agressiveness. To reduce effective power consumption further and extend battery life of the final device, the power gating investigation of [30] should be extended to further reduce leakage, perhaps by cascoding sleep transistors or adjusting their body bias to shift their threshold voltage and reduce subthreshold leakage. Stacking transistors will reduce f_{Max} and increase dynamic energy given the increase in capacitance, but given that both these metrics are well below their constraints, this is acceptable. Cascoding sleep transistors will also increase area, although this is expected to a minor increase.

Table IX: Suggested improvements to the ASIC accelerator and their potential benefits

Improvement	Dyn. energy Δ	Leakage Δ	Latency Δ	Area Δ
Centralized memory	Slight $\downarrow$	-13.75 mW	Slight $\downarrow$	-0.931mm^2
Centralized compute	Slight $\downarrow$	-25.80mW	+0.305s	-1.183mm^2
Bus elimination	Slight $\downarrow$	Slight $\downarrow$	No change	Slight $\downarrow$
Optimize Q format	Indeterminate	Indeterminate	Indeterminate	Indeterminate
Power gating	Slight $\uparrow$	Significant $\downarrow$	Slight $\uparrow$	Slight $\uparrow$

With the quantified suggested improvements, the leakage power of the ASIC would be reduced from 56.48mW to 16.93mW. Ignoring improvements to power gating, this reduces the effective power consumption from 3.046 mW to 0.848mW, a 72.2% reduction.

The area would decrease from 3.86mm^2 to 1.746mm^2, a 54.8% reduction, which brings the total area to just under the objective of 2mm^2.

Section 7 has shown that the current design is not suitable for the sleep staging earpiece project as is. However, there is a clear path forward to designing an AI accelerator with significantly reduced power consumption and area, which are the two most important metrics for the earpiece project.

9 Conclusion

This thesis has presented a vision transformer-based model for automatic sleep staging and an ASIC accelerator to run the model. This work is part of a larger project aimed at developing an in-ear device capable of acoustic neuromodulation to treat insomnia, which requires live, accurate and practical sleep staging. The model achieved 82.9% accuracy on the MASS SS3 dataset using a single EEG input and has a size of 31,589 parameters. The ASIC accelerator consumes 3.046mW on average, has an area of 3.86mW^2 and an inference latency of 6.97ms. The results show that performing sleep staging in-ear is not only possible but also practical. The model meets all the requirements summarized in Table I, except for the area of the accelerator. Future work should focus on reducing the power consumption and area of the accelerator. Centralizing compute and memory can reduce power consumption and area by 72.2% and 54.8%, respectively. It is also advised to investigate running the model on a RISC-V processor with vector extension or CGRA add-on for additional flexibility and decreased development risk and timeline.

References

[1] Jean-Philippe Chaput, Jessica Yau, Deepa P. Rao, et al. "Prevalence of insomnia for Canadians aged 6 to 79". In: *Health Reports* 29.12 (2018).

[2] Ho-Kyoung Yoon. "Neuromodulation for Insomnia Management". In: *Sleep Medicine and Psychophysiology* 28.1 (2021), pp. 2–5.

[3] Jessica Vensel Rundo and Ralph Downey. "Chapter 25 - Polysomnography". In: *Clinical Neurophysiology: Basis and Technical Aspects*. Ed. by Kerry H. Levin and Patrick Chauvel. Vol. 160. Handbook of Clinical Neurology. Elsevier, 2019, pp. 381–392. DOI: https://doi.org/10.1016/B978-0-444-64032-1.00025-4. URL: https://www.sciencedirect.com/science/article/pii/B9780444640321000254.

[4] Huy Phan and Kaare Mikkelsen. "Automatic sleep staging of EEG signals: recent development, challenges, and future directions". In: *Physiological Measurement* 43.4 (2022), 04TR01.

[5] Micheal Dutt, Surender Redhu, Morten Goodwin, et al. "SleepXAI: An explainable deep learning approach for multi-class sleep stage identification". In: *Applied Intelligence* 53.13 (2023), pp. 16830–16843.

[6] Mingyu Fu, Yitian Wang, Zixin Chen, et al. "Deep learning in automatic sleep staging with a single channel electroencephalography". In: *Frontiers in Physiology* 12 (2021), p. 628502.

[7] Emadeldeen Eldele, Zhenghua Chen, Chengyu Liu, et al. "An attention-based deep learning approach for sleep stage classification with single-channel EEG". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 29 (2021), pp. 809–818.

[8] *USB Accelerator datasheet*. Version 1.4. Coral. 2019. URL: https://coral.ai/static/files/Coral-USB-Accelerator-datasheet.pdf.

[9] Jeff Suovanen. *Airpods Pro Teardown*. Tech. rep. Ifixit, 2019.

[10] *Implementation of Flip Chip and Chip Scale Technology*. Joint Industry Standard, 1996.

[11] Ramish Zafar. *AirPods 2 Teardown: H1 Chip With Bluetooth 5.0, Same Batteries, and Water-Repellent Coating on Charging Case Board*. Tech. rep. Wccftech, 2019.

[12] Zhiyu Liu and Volkan Kursun. "Characterization of a novel nine-transistor SRAM cell". In: *IEEE transactions on very large scale integration (VLSI) systems* 16.4 (2008), pp. 488–492.

[13] Huy Phan, Kaare Mikkelsen, Oliver Y Chén, et al. "Sleeptransformer: Automatic sleep staging with interpretability and uncertainty quantification". In: *IEEE Transactions on Biomedical Engineering* 69.8 (2022), pp. 2456–2467.

[14] Kai Han, Yunhe Wang, Hanting Chen, et al. "A survey on vision transformer". In: *IEEE transactions on pattern analysis and machine intelligence* 45.1 (2022), pp. 87–110.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[16] Yang Dai, Xiuli Li, Shanshan Liang, et al. "MultiChannelSleepNet: A Transformer-based Model for Automatic Sleep Stage Classification with PSG". In: *IEEE Journal of Biomedical and Health Informatics* (2023).

[17] Yunxiang Hu, Yuhao Liu, and Zhuovuan Liu. "A survey on convolutional neural network accelerators: GPU, FPGA and ASIC". In: *2022 14th International Conference on Computer Research and Development (ICCRD)*. IEEE. 2022, pp. 100–107.

[18] Tianshi Chen, Zidong Du, Ninghui Sun, et al. “Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning”. In: *ACM SIGARCH Computer Architecture News* 42.1 (2014), pp. 269–284.

[19] Zidong Du, Robert Fasthuber, Tianshi Chen, et al. “ShiDianNao: Shifting vision processing closer to the sensor”. In: *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 2015, pp. 92–104.

[20] Caiwen Ding, Siyu Liao, Yanzhi Wang, et al. “Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices”. In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 2017, pp. 395–408.

[21] Aman Arora, Tanmay Anand, Aatman Borda, et al. “CoMeFa: Compute-in-Memory Blocks for FPGAs”. In: *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2022, pp. 1–9.

[22] Xiaowei Wang, Vidushi Goyal, Jiecao Yu, et al. “Compute-capable block RAMs for efficient deep learning acceleration on FPGAs”. In: *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2021, pp. 88–96.

[23] Clément Farabet, Yann Lecun, Koray Kavukcuoglu, et al. “Large-Scale FPGA-Based Convolutional Networks”. In: *Scaling up Machine Learning: Parallel and Distributed Approaches*. Ed. by Ron Bekkerman, Mikhail Bilenko, and JohnEditors Langford. Cambridge University Press, 2011, pp. 399–419. DOI: 10.1017/CBO9781139042918.020.

[24] Panjie Qi, Yuhong Song, Hongwu Peng, et al. “Accommodating transformer onto FPGA: Coupling the balanced model compression and fpga-implementation optimization”. In: *Proceedings of the 2021 on Great Lakes Symposium on VLSI*. 2021, pp. 163–168.

[25] Ruoyu Zhi, Ryan Jurasek, Wolfgang Hokenmaier, et al. “Opportunities and Limitations of in-Memory Multiply-and-Accumulate Arrays”. In: *2021 IEEE Microelectronics Design & Test Symposium (MDTS)*. IEEE. 2021, pp. 1–6.

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

[27] 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.

[28] 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).

[29] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Searching for activation functions”. In: *arXiv preprint arXiv:1710.05941* (2017).

[30] A Sathanur, Andrea Calimera, Antonio Pullini, et al. “On quantifying the figures of merit of power-gating for leakage power minimization in nanometer CMOS circuits”. In: *2008 IEEE International Symposium on Circuits and Systems*. IEEE. 2008, pp. 2761–2764.

[31] Guo-Qiang Zhang, Licong Cui, Remo Mueller, et al. “The National Sleep Research Resource: towards a sleep data commons”. In: *Journal of the American Medical Informatics Association* 25.10 (2018), pp. 1351–1358.

[32] PhysioToolkit PhysioBank. “Physionet: components of a new research resource for complex physiologic signals”. In: *Circulation* 101.23 (2000), e215–e220.

[33] Lan Zhuang, Minhui Dai, Yi Zhou, et al. “Intelligent automatic sleep staging model based on CNN and LSTM”. In: *Frontiers in Public Health* 10 (2022), p. 946833.

[34] Huy Phan, Oliver Y Chén, Minh C Tran, et al. “XSleepNet: Multi-view sequential model for automatic sleep staging”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.9 (2021), pp. 5903–5915.

[35] Erdrin Azemi, Ali Moin, Anuranjini Pragada, et al. “Biosignal Sensing Device Using Dynamic Selection of Electrodes”. US20230225659A1. 2023.

[36] Matteo Perotti, Matheus Cavalcante, Nils Wistoff, et al. “A “New Ara” for vector computing: an open source highly efficient RISC-V V 1.0 vector processor design”. In: *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2022, pp. 43–51.

[37] Xiaoyi Ling, Takahiro Notsu, and Jason Anderson. “An Open-Source Framework for the Generation of RISC-V Processor + CGRA Accelerator Systems”. In: *2021 24th Euromicro Conference on Digital System Design (DSD)*. 2021, pp. 35–42. DOI: 10.1109/DSD53832.2021.00015.

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 X 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 X: 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 XI: Line and file count per file type in the codebase

File type	File count	Line count	Percent of total
Python	25	4270	29.1%
SystemVerilog	26		19.3%
TeX	18	670	26.4%
C++	10	1250	15.3%
Shell	20	300	3.5%
Other		20	6.4%
Total		13,000	100%

C Reflection on Learnings and Experience Gained

This project has been a significant learning experience for me. I have learned a great deal about artifical intelligence and the design and implementation of hardware systems. I'm including this section to formalize my reflections on experience gained, experience I wasn't able to gain and what I would have done differently.

C.1 Acquired Experience

Firstly, I gained good knowledge in the basics of artificial intelligence through Andrew Ng's Deep Learning Specialization on Coursera and practical experience using TensorFlow. I also reinforced my skills in C++, SystemVerilog and LaTeX and was able to use industry-standard tools such as Synopsys Design Compiler and ARM Artisan IP. I developed my own local workflow for developing RTL with Verilator, CocoTB and GTKWave, whose flexibility will be a great asset and enabler in my future projects.

That being said, the largest takeway from this project is a direct appreciation for the complexity of hardware-software co-design. This project took me from the high-level frameworks of Python to bit-level arithmetic, FSM and cycle-level parallelism. Through this, I've realized that owning the full-stack is powerful and gives significant design freedom to optimize the system. In turn, this can prove destabilizing as essentially all aspects of the design have compounding pros and cons. It is thus critical to develop flexible, accurate and actionable functional simulations to evaluate different aspects of the design before committing to a full implementation. I am glad to have done that to some extent with the C++ model and various Python studies, but, in retrospect, more time should have been spent desiging and obtaining proxy metrics to determine the ideal high-level architecture. For instance, I did not need to design the full system before being able to obtain PPA figures for the memory and compute units. This would have allowed me to make more informed decisions about the architecture and potentially save time in the

long run. However, I think such learning can only be appreciated once an architect goes through the full design cycle at least once, so I am glad to have had the opportunity to earn this wisdom early in my career, which I will carry in future projects. I'm realizing that an informed and complete analysis of a design without necessarily implementing it is more insightful and impactful than a somewhat underdiscussed but working design

C.2 Topics Warranting Further Exploration

I would have liked to have had more time to explore design synthesis and implementation. I only managed to make use of Design Compiler and Artisan IP for about six weeks, and I would have liked to be able to learn more about their different features, optimization strategies and how to contol them. I would have also liked to have had more time to explore the impact of different of memory compilation. Finally, this project stopped at design synthesis. Implementation and post-routing simulations are other areas that I did not have time to explore.

C.3 Alternative Approaches for Consideration

In my opinion, the three-steps plan presented in Section 4 was a good approach to the project. However, it was too coarse and I would add two steps: a step between the model and the functional simulation to translate the model to software-style C++ instead of hardware-style C++. This allows for easier testing of design choices such as fixed-point strategies and allows to collect metrics such as exact number of different types of operations and memory accesses without getting slowed down by hardware-style coding. The second additional step would be to write the functional simulation from the bottom up along with coding RTL hardware modules in parallel. This would allow progressively more precise evaluation of the PPA metrics and provide more time to make architecture decisions.

This page intentionally left blank.