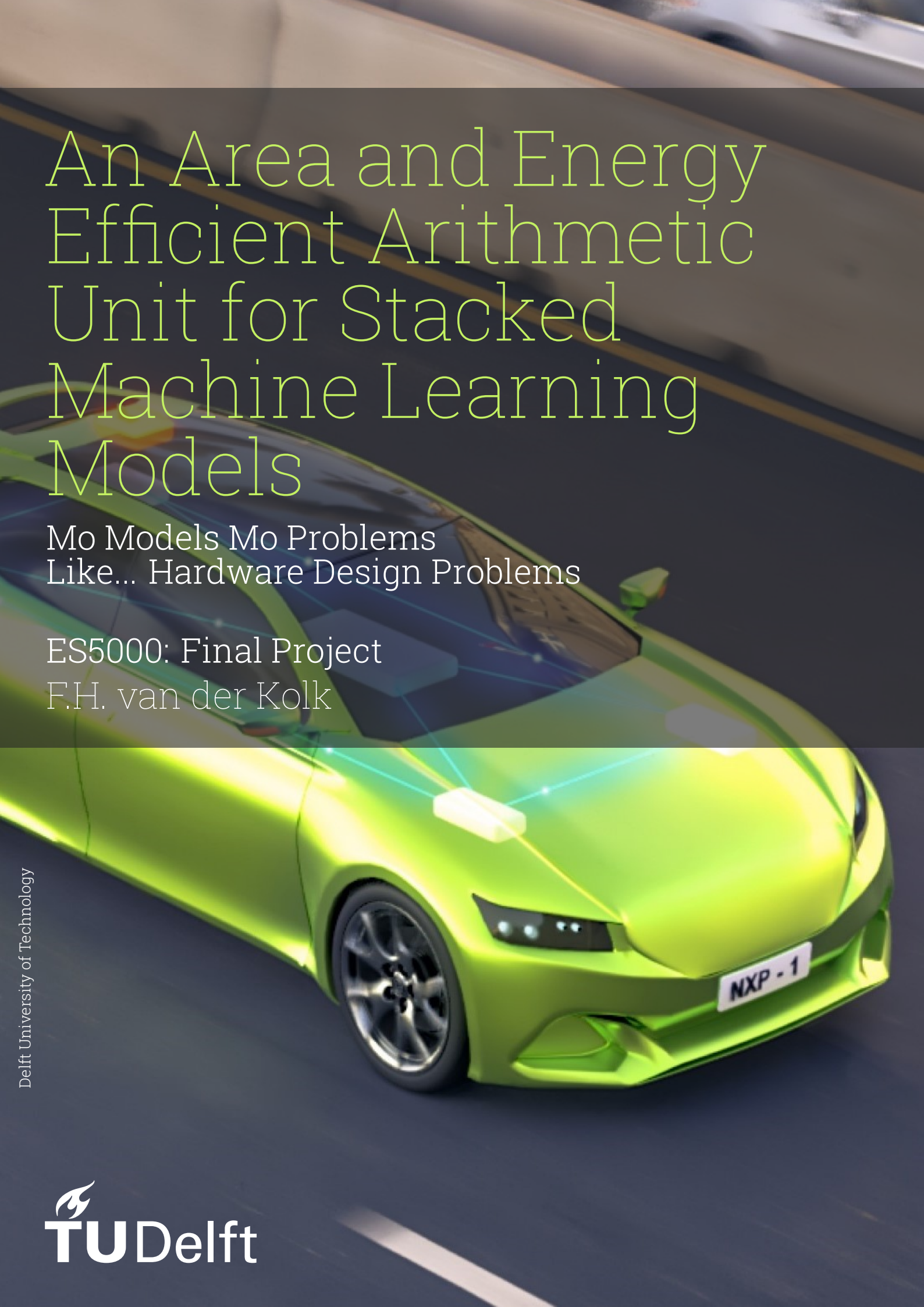


An Area and Energy Efficient Arithmetic Unit for Stacked Machine Learning Models

Mo Models Mo Problems
Like... Hardware Design Problems

ES5000: Final Project

F.H. van der Kolk

Delft University of Technology

TU Delft

An Area and Energy Efficient Arithmetic Unit for Stacked Machine Learning Models

Mo Models Mo Problems Like... Hardware Design Problems

by

F.H. van der Kolk

Student Name	Student Number
Floris van der Kolk	4961404

University Supervisors: Said Hamdioui & Anteneh Gebregiorgis
NXP Supervisor: René van den Berg
Project Duration: April, 2023 - January, 2024
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

For my master's thesis I designed an energy and area efficient arithmetic unit for a stacked machine learning model. The model was designed by a previous student for the prediction of a chip's remaining lifetime. The goal of the overarching project at NXP Semiconductors was to create a lifetime prediction system that can be cheaply implemented into various embedded automotive devices. The aim was for the system to have a distributed architecture and to be independent from other on-chip systems. This focused my design on area-efficiency, while prediction speed was much less important in the context of semiconductor aging.

I found that the parallel "Reduced Area Multiplier" most efficiently performs the many multiplications in the SML model, Invariant Integer Division cheaply provides support for division, and supporting operations at multiple bit-precision through configurability requires additional chip area for little energy benefit.

I conducted my thesis research under supervision of Professor Said Hamdioui in the Computer Engineering research group. My daily supervisors were Anteneh Gebregioris, assistant professor at TU Delft, and René van den Berg, IC/System Architect at NXP. My graduation committee is joined by Professor Georgi Gaydajiev.

I want to thank René and Anteneh for their guidance each and every week. They helped me a lot in bringing structure to my literature study and design process, and provided valuable feedback during the writing of this report. I also want to thank Martin Pos at NXP for teaching me about linux, genus and scripting and quickly helping me whenever I had problems in those environments. Thank you.

Finally, thanks to my parents, Rona Vree and Rick van der Kolk, as well as my girlfriend, Chenzhi Hu, for their consistent support during this thesis project and before.

F.H. van der Kolk
Nijmegen, January 2024

Abstract

Machine learning on edge devices performs crucial identification or prediction tasks while limiting the amount of data that needs to be transmitted to more centralized computing nodes. However, strict area and energy requirements necessitate specialized hardware developed for the requirements of the device and model. This thesis is concerned with developing an area and energy arithmetic unit as part of the implementation of a stacked machine learning model in embedded automotive devices. The model in question was previously designed to perform lifetime prediction with the goal of improving the reliability of semiconductor devices used in various automotive applications.

This thesis aims to achieve area and energy efficiency by exploiting the commonalities in the arithmetic operations of several of the internal learners of the stacked machine learning model. The use of a weighted figure of merit, taking into account area, energy and delay, allow for simple comparisons of designs at any operation frequency and easy insight into the changes in the merit of designs if device requirements were to change. A sweep of the percentage of multiplications in the workload also gave insight into how design choices may change due to future redesigns of the stacked machine learning model.

It was found that the MAC, multiply, divide and accumulate operations of the internal learners can best be supported by one arithmetic unit containing a ”Reduced Area” parallel multiplier (still taking up most of the area), a small dedicated accumulator and invariant integer division using the multiplier. It was also found that the ability to reconfigure the multiplier for different levels of bit-precision does not yield performance improvement for the expected precision distribution.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
3-NN	Three Nearest Neighbors
ASIC	Application Specific Integrated Circuit
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
FA	Full Adder
FoM	Figure of Merit
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
HA	Half Adder
HTOL	High Temperature Operating Life
IC	Integrated Ciruit
IID	Invariant Integer Division
KPI	Key Performance Indicator
MAC	Multiply-and-Accumulate
ML	Machine Learning
PP	Partial Product
RBF	Radial Basis Function
RCA	Ripple Carray Adder
RF	Random Forest
SVM	Support Vector Machine
SVR	Support Vector Regression
SWP	Sub-Word Parallel
TD	True Division

1 Introduction

1.1. Motive

All around us electronic devices are measuring increasingly unmanageable amounts of data on everything from voice control to weather sensors. The traditional way of making use of these data is by transmitting it to servers and data centers where it can be processed into useful information. However, this leads to too much load on the networks between the measurement and processing device, risks leaking sensitive data and prohibits real time responses in those measuring ”edge” devices. Therefore, deployment of artificial intelligence on the edge has become a popular practice in recent years.

This same development of electronic devices appearing in every application imaginable also puts pressure on the field of semiconductor reliability. Old methods of accelerated lifetime testing that could only ensure a rated lifetime on the day of production are becoming more expensive as lifetime requirements increase. Integrated circuits (IC) are used in unpredictable conditions and in life-critical components, like in cars and medical devices. Ensuring reliability in these scenarios requires better lifetime estimations and the ability to measure and predict remaining lifetime in the field. This is where machine learning (ML) has proven to be a valuable tool [1], [2].

Machine learning models can also transform large quantities of data into a smaller amount of useful information that is worth recording. However, these mathematical models often require lots of power hungry arithmetic that devices on the edge can ill afford to spend their energy and chip area on. For this reason, a whole field of research has sprung up around energy and area efficient implementation of ML models. Most of this research is directed towards one model in particular, Neural networks. This makes sense, as neural networks excel at finding patterns in large amounts of data, supporting great progress in image recognition and natural language processing.

A bit less attention is given to situations when less measurement data is available. Here classical ML models such as Linear Regression, Decision Trees, k-Nearest Neighbors and Support Vector Machines (SVMs) can regularly outperform neural networks in accuracy [3]. On top of this, classical ML models usually require less hardware resources to be trained and implemented and they benefit from a higher level of interpretability than neural networks. As such, these models still have a fair amount of research dedicated to developing efficient hardware implementations [4], [5].

This thesis focuses on one particular case of classical machine learning, model stacking, which has received very little attention from computer hardware engineers. Model stacking is a method of combining the outputs (predictions) of multiple separate ML models (base-learners) by using these base-predictions as inputs for yet another ML model (a meta-learner). By leveraging the qualities of several different types of ML models, model stacking can perform particularly well with small training data sets and do well to avoid over-fitting on the training data. The hardware benefits of the classical models that make up the stacked ML model are also maintained [6]. This means fewer resources are needed for training and a higher degree of interpretability compared to neural networks. Yet, the diversity in base-learner models does reflect in the complexity of the arithmetic and the data storage of Stacked ML (SML) models, requiring well designed hardware to run efficiently. To make the imple-

mentation of stacked machine learning models on edge applications feasible, research into hardware implementation is crucial, yet absent in literature. That is why this thesis focuses on answering the question: How do you create low energy and area efficient hardware for stacked machine learning models?

1.2. Problem Statement

This thesis deals with a design problem concerning the hardware implementation of a Stacked ML model as part of a project for interns at NXP Semiconductors, an international semiconductor manufacturer based in the Netherlands. The goal of the project is to explore the feasibility of performing lifetime prediction inside embedded automotive devices. The eventual aim is cheap integration of such a lifetime prediction model into a large selection of semiconductor devices within cars.

To achieve this, the efficient use of chip area is paramount for the hardware that runs the model. As a result, the most important Key Performance Indicator (KPI) for the hardware design of the stacked ML model is area efficiency. The other KPIs used to evaluate the design results of this thesis are energy and delay. Of these, energy is more important than delay, as semiconductor lifetime prediction is not a problem that requires high prediction speeds. Meanwhile vehicles do have limitations on their power supply.

Prior research into lifetime prediction for embedded automotive devices [7] presents a stacked ML model that will provide insight into the hardware requirements of SML models in general and serve as the target model to optimize for in this thesis. The SML model predicts the lifetime of a new data point consisting of 30 features. These 30 features are divided equally among 6 base learners specified to be the following models: Ridge Regression, Radial Basis Function (RBF) Kernel Regression, Polynomial Kernel Regression, Support Vector Regression (SVR), 3 Nearest Neighbors (3-NN) and Random Forest (RF). The predictions of these base-learners is then used by another RF model meta-learner to generate a final prediction.

The individual learners of the SML model each have literature which covers their hardware implementation, as discussed in section 2.1.4. However, fully independent implementation of the 7 machine learning models based on these existing solutions cannot lead to an area efficient design. Efficient hardware for the SML model described above will require the exploitation of similarities between the learners to reduce resource usage. No such hardware has previously been made, nor has this topic been discussed in literature. Therefore, such efficient SML model hardware forms the design problem and the research gap this thesis aims to fill.

1.3. Research Questions

Exploration into the hardware requirements of these ML models, see background subsection 2.1.4, shows common denominators in the multiply, accumulate, and divide operations of the 7 models. The four regression models require multiply-and-accumulate (MAC) operations, the 3-NN and RF models need accumulation and division and lastly the kernel regression models ask for a large quantity of multiplication, accumulation and division. Allowing the 7 learners in the SML model to share one area and energy optimized arithmetic unit is important. While workload optimized arithmetic units are not a new concept, the unique characteristics of SML models should lead to novel findings.

One such model characteristic that needs mentioning is the unequal number of bits that individual base-learners may need to perform at an adequate level of accuracy. Reducing the number of bits used to describe the values in an ML model will save hardware resources. However, in Stacked ML models this benefit may be lost due to a single base- or meta-learner requiring high bit-precision arithmetic.

As such, the contributions of this thesis will stem from the efforts of both efficiently combining multiply, accumulate and divide operations into one arithmetic unit and maintaining the benefits of reduced bit-precision ML models.

This thesis aims to answer the following two research questions and sub-questions:

1. How do you integrate the arithmetic operations of multiple classical ML models into one area and energy efficient arithmetic unit?

    1.1. How can multiple models efficiently share one unit for their basic MAC operations?
    1.2. Can division be efficiently integrated with a MAC unit?

1.3. Can kernel calculations be efficiently performed on a MAC unit?

2. How do you maintain the benefits of reduced bit-precision ML models in hardware that executes models with various levels of bit-precision?

1.4. Contributions

The key contributions of this thesis and their significance are as follows:

- **The design of an area and energy optimized arithmetic unit for the stacked machine learning model:**
  The discovery that a parallel multiplier, specifically a Reduced Area multiplier, a second dedicated accumulator, and support for Invariant Integer Division using the existing multiplier are best performing design choices for the workload of the SML model. This is a crucial step towards the complete proof of concept of an on-chip lifetime prediction system for NXP.
- **An example of the design of dedicated hardware for SML model:** Besides the design choices made in this thesis the result also give insight into alternative scenarios when application requirements or workloads are different. This will be useful for future researchers as there is currently a lack of literature on the subject.
- **Performance figures of the ASIC implementation of Invariant Integer Division in 28nm CMOS technology:**
  This is useful and previously lacked statistic for engineers considering this division method.

1.5. Thesis Structure

The next chapter, chapter 2, contains background on lifetime prediction, machine learning, the SML model and hardware for classical ML models. It also discusses related works on computer arithmetic hardware. The methodology of the experiments in this thesis and descriptions of hardware designs are presented in chapter 3. Chapter 4 analyses the subsequent results and finally chapters 5 and 6 contain the discussion and the conclusion.

2

Background and Related Works

This chapter consists of a Background and a Related Works section. The Background section discusses the required background for this thesis. There are subsections on lifetime prediction, the stacked ML model in question, machine learning, existing hardware for it, and an analysis of the stacked ML model's hardware requirements. The Related Works section compares the various existing partial solutions to the design problem in several subsections. These were inspiration for the design choices compared in the methodology and results of this thesis. The related works section ends with an existing system best suited to compare the results of this thesis to.

2.1. Background

The background of this thesis' research is sketched in several subsections. Subsection 2.1.1 explains the problem of IC aging and importance of lifetime prediction in guaranteeing reliability. Then, subsection 2.1.2 introduces the stacked machine learning model that is central to this thesis in more detail. Subsection 2.1.3 lays out a brief and basic overview of machine learning and the ML models that are relevant for a proper understanding of the design problem. Subsection 2.1.4 describes existing hardware solutions to the individual learners of the SML model, showing where ML hardware is already well-developed. Finally, Subsection 2.1.5 shows how the research questions were defined based on the hardware requirements for the SML model.

2.1.1. Lifetime Prediction

The never ending march of Moore's law has made semiconductor devices more and more complex, small, and versatile. As a result, there has been a continuous trend of Integrated Circuits (ICs) appearing in an even wider range of applications. With these systems in phones, fridges, doorbells and cars, the term Internet of Things (IoT) as sprung up to express how they all process and share information at an unthinkable scale. As this trend continues, it keeps getting more important that IC can be relied upon to last long and not fail unexpectedly, especially in life-critical applications like self-driving cars and medical devices.

It has been long identified that electrical devices exhibit a distinct pattern of failure. The probability distribution of device failure shows its highest peaks shortly after production (early failure) and near the end of the expected lifetime (wear-out). Earlier failure of semiconductor devices is caused primarily by manufacturing defects, while later failures are caused by degradation due to use and environmental factors [1]. In the modern day, most semiconductor reliability is achieved through accelerated lifetime tests. The reliability requirements of any particular IC is tested by running the device at higher temperatures, voltages or higher mechanical stress and watching when they fail. By doing this, it becomes possible to say with reasonable certainty that most IC will survive for a certain minimal lifetime [1].

However, because every semiconductor device is different due to imperfect manufacturing and each device is used differently in different environments, it is impossible to guarantee that some ICs will not fail significantly sooner without more insight into what is happening inside the device. Thankfully, when it comes to wear-out failure, there are several well understood processes that pre-stage failure and can be detected. Examples of these are Electromigration, Stress Migration, Negative Bias Temperature

Instability, Time-Dependent Dielectric Breakdown, and Hot Carrier Injection [8]. By monitoring these processes, it becomes possible to predict imminent failure of ICs or even their longer term remaining lifetime. However, this is not an easy task. The existence of one of these failure mechanisms rarely means failure of the entire device. The extreme and increasing complexity of ICs makes it hard to set up a model that can predict when too much aging has taken place. This is where Machine Learning has great potential, and existing technology in this direction will be discussed at the end of this subsection.

The advantages of accurate models would be immense. Currently, a lot of over-engineering takes place to guarantee lifetime, meanwhile, the possibility of unexpected early failure stays. Also, electronic service providers need to spend lots of resources to respond quickly to deal with these unexpected failures. With accurate models of IC failure, systems can be designed more accurately for their requirements and when failure is predicted in the field well ahead of time it is possible to apply fine-tuned corrective measures like adjustment of operating conditions or replacing the IC at a convenient and safe time [1].

Existing research towards this purpose shows that mainly Support vector machines and Neural Networks are being used to predict hardware faults. [9] used an SVM to predict failure in hardware components. Around the same time, [10] created an Artificial Neural Network accelerated on an FPGA to detect faults in Automotive systems. A few years later, [2] explored the accuracy of an SVM predictor at different numbers of monitored flip-flops and time-sampling rates. Finally, [8] uses Fast Fourier Transform (FFT) to obtain sufficient information from fault signals to train and use a Convolutional Neural Network for the prediction of a wide range of faults caused by a wide range of aging mechanisms. The last two being from 2019 and 2022 respectively shows that this is still an active field of research.

2.1.2. The Stacked Machine Learning Model

The Stacked Machine Learning model, from now on referred to as the SML model, was previously designed by [7] for predicting future HTOL (High-Temperature Operating Life) test measurements based on earlier HTOL measurements. This was the first step in developing a lifetime prediction system for embedded automotive devices. [7] found that stacked machine learning achieved significantly higher prediction accuracy than individual models especially when performing view splitting before training. Although this is partly attributed to the small dataset used to train the HTOL prediction models, the SML model still shows to be a promising model for the lifetime prediction of embedded automotive devices. Research on stacked ML models is hard to come by. As a result, there is a necessity to invest in research on the hardware implementation of the SML model. In this thesis, hardware is designed for the SML model. Therefore it is important to know the details of the model.

For the next subsection on machine learning, it is important to remember a couple of facts about the SML model that is central to this thesis. The SML model is trained on a data set of 90 data points with 30 features. The 30 features are split into 6 sets of 5 features. This provides smaller data sets for 6 base-learners which are each different types of ML models. The predictions of each of the 6 base-learners, that work fully independently of each other, are used as the input data for a meta-learner that makes a final prediction on lifetime. Some background on the different models (learners) that make up the SML model is given in the next subsection. The 6 base-learners are Linear (Ridge) Regression, RBF Kernel Regression, Polynomial Kernel Regression, Support Vector Regression, 3 Nearest Neighbors and Random Forest Regression models. The meta-learner is another Random Forest regressor.

2.1.3. Machine Learning

Machine learning is a form of Artificial Intelligence that, instead of being explicitly programmed, uses a computer algorithm to find relations within sets of data. Often these relations are in the form of a model that describes how a dependent variable y relates to multiple independent variables x. For example, how a person's height (variable y) may relate to their age, sex and the height of each of their parents (variables x). Finding such a model using a machine learning algorithm is called training, which is done with a training data set.

A training data set is a collection of data points recording their independent variables x. Such a data set can be called "labeled" if these data points also include the correct values for the y variable. This distinction defines the first categorization within Machine Learning: Supervised Learning and Unsupervised Learning, which are done with labeled and unlabeled training data sets respectively. The advantage of unsupervised learning is that unlabeled data are easier to obtain, allowing for more data for the model to train on, which can improve the model performance. However, supervised learning

learning allows for more control over the ML model and can perform more reliably on less complex problems. Often unsupervised learning is used when it is not clear what kind of useful information can be obtained from a large amount of data. On the other hand, supervised learning is used when a specific value or characteristic of the data needs to be identified [11].

ML models concern themselves with predicting the dependent variable y. Whether this y variable can take on a continuous range of values or discrete classes, marks the distinction between regression models and classification models. For this thesis most relevant are regression models. These are used in the SML model, because future lifetime, the SML model's dependent variable y, is a continuous variable, whether it is expressed in days, months or years.

Once a model is trained, it is capable of providing predictions about the y variables of new unlabeled data points. This is called inference. To test the performance of the model, inference can be performed on data points that are labeled, but not trained on. In the testing phase, it is important to use a sufficiently large and realistic set of data that did not appear in training. Under that condition, a good accuracy figure of the model can be obtained that will reflect its performance when deployed in the field.

The rest of this subsection will give short explanations about the concepts and equations behind the internal ML models that appear in the SML model and ends with some details on model stacking.

Linear Regression

Linear regression is a very simple ML model. It describes the relationship between independent variable x and dependent variable y using Equation 2.1 [12]

$$y = \hat{x}^T \hat{w} + b \tag{2.1}$$

Here independent variable x is represented by a vector of its features $\hat{x}$ and $\hat{w}$ contains weights for each feature. b is the bias. The weights and biases are obtained during the training of the model. The SML contains specifically a Ridge Regression model, but this "Ridge" refers to a specific training method that is not relevant for inference.

Kernel Regression

Kernel Regression is a regression model that adds a transformation of the input data. This allows the model to not only find linear, but also non-linear relationships within the training data [13]. Kernel Regression follows Equation 2.2 [12]

$$y = b + \sum_{i=1}^{n} \alpha_i K(x, c_i) \tag{2.2}$$

Here n is the number of training data points, a_i are the weights that are obtained during the training of the model and $K(x, c_i)$ is the kernel transformation of the new data point x in respect to the i^{th} training data point c_i. There are many kernels, described by different equations, that are used for various purposes. However, for this thesis the relevant ones are the Radial Basis Function (RBF) kernel (or Gaussian kernel), the Polynomial kernel and the Linear kernel. Only these will be considered as they are part of the SML model. The kernels are described by equations 2.3, 2.4 and 2.5.

$$K(x, c_i) = e^{-\frac{||x - c_i||^2}{2\sigma^2}} \tag{2.3}$$

Here, σ is the variance of the training data and $||x - c_i||$ is the Euclidean distance between point x and c_i, which will later described by equation 2.6. The RBF kernel is an infinite dimensional kernel. This allows for a regression line of any shape, thus being able to follow the most complex and high dimensional patterns in the data [14]. The RBF kernel is also most easily interpreted as a similarity kernel. When x and c_i are close together in their original feature space the Euclidean distance $||x - c_i||$ becomes small, which means the kernel becomes big.

$$K(x, c_i) = (\hat{x}^T \hat{c}_i + b)^d) \tag{2.4}$$

Here, b is a free parameter to be determined during model training and d is the degree of the polynomial. A higher degree means higher-order non-linear relationships in the data can be described by the ML model. Too high of a degree leads to overfitting. Notice that if d is 1, the polynomial kernel becomes almost the same as the linear kernel [14].

$$K(x, c_i) = \hat{x}^T \hat{c}_i \tag{2.5}$$

k-Nearest Neighbors

A k-NN regression algorithm is conceptually quite simple. The provided test data point is compared with each data point in the training data by calculating the distance between the two points in the n dimensional feature space. Where n is the number of features each data point has. This can either be done as Euclidean distance or Manhattan distance, equations 2.6 and 2.7. The average of the dependent variables y in the k training data points that are the nearest to the test data point is then the predicted value of the y variable of the test data point.

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_n - b_n)^2} \tag{2.6}$$

$$d(a,b) = |a_1 - b_1| + |a_2 - b_2| + ... + |a_n - b_n| \tag{2.7}$$

Random Forest

A Random Forest is a group of decision trees, which are ML models in their own right. Each decision tree is trained to make predictions about the dependent variable y based on the features of a test variable x. The function of the Random Forest, in case of regression, is to take as the final prediction for y the average prediction of all the trees in the forest. In the case of classification, this final prediction would be the most common prediction made by the trees. This makes a Random Forest a case of ensemble learning called Bagging [15].

A decision tree makes predictions about the value of the dependent variable y by making a series of often binary decisions that lead it along the branches of the decision tree. Each decision determines which branch is followed next and therefore what the next decision node is. At each decision node, the value of a feature of test variable x is compared to a threshold that is determined during the training of the model. The maximum number of decisions needed to be taken before a prediction is obtained is called the depth of the tree.

In an RF model, the trees of the forest are often trained on different random selections of features of different random portions of the training data points. This is called view splitting. The depth of the decision trees is also kept purposefully low. By doing this, each individual decision tree can only make prediction based on simple observations made on different portions of the training data, but the collection of hundreds of trees, depending on the problem size, provides high accuracy and robustness to noise in the data. [15].

Model Stacking

Model stacking is a method within machine learning where multiple models are combined to obtain better performance. To be specific, in model stacking, there are multiple independent models, called base-learners, which are trained independently and give independent predictions about data points provided to them. The predictions of the base learners about data points in the training data set are then used to train yet another model, the meta learner. The meta-learner is the tool to combine all the differing predictions of all the base-learners. Originally introduced by [6] as Stacked Generalization, model stacking is known to improve accuracy and correct for the biases of the base learners [16]. On top of that, ensemble learning, a category which includes all other forms of combining the results of multiple models, is know to reduce variance [17].

2.1.4. Hardware for ML Models

From a hardware perspective the learners of the SML model fit into 4 categories. This subsection briefly discusses the existing solutions to these separately.

Basic Regression

The first category is basic regression. These are regression models that, ignoring the training methods, merely require sufficiently fast MAC operations and enough memory bandwidth. In the SML model, these are Linear Regression and SVR. MAC operation design can be split into two problems: multiplier design and multiplier-accumulator integration. Multipliers are an important part of the State of the Art for this thesis and are therefore discussed in 2.2.1. MAC operations are basic arithmetic that has been relevant for many Digital Signal Processing (DSP) applications far into the 20th century. As such there is a lot of research focusing on the integration of accumulation into multiplication. Recently, MAC operations have gained even more importance due to its relevance of machine learning, including every type of neural network.

[18] seems to have been the first to make a MAC unit by integrating accumulation into a Baugh-Wooley multiplier to save area and delay. Almost a decade later, [19] merged accumulate and multiply operations by using a Modified Booth Algorithm, with an emphasis on delay reduction. [20] reduces energy consumption with conditional evaluation in their CSAs. [21] made self-timed, pipelined versions of both Modified Booth2 and Modified Baugh-Wooley multipliers in order to reduce area and delay. [22] reduced area and delay using modified 4:2 compressors. [23] proposed that accumulation can be performed in the final adder of the multiplier, reducing mostly energy and delay. Finally, the recent trend of approximate arithmetic is also applied in MAC operations by [24] and [25]. The second of these was specifically designed for Neural Networks.

Kernel Regression

The category kernel regression contains any model that utilizes kernels. Within the SML model, these are RBF kernel regression and Polynomial kernel regression. Both kernels require a fairly large number of arithmetic operations, mostly centered around multiplication, especially when compared to their linear counterparts. In particular, the RBF kernel needs support for an exponential. For this reason there has been several research papers that discuss the subject of RBF kernels.

[26] use an RBF kernel for an SVM model classifying images. To provide hardware support for this they use an FPGA and represent the exponential function of the RBF kernel as a summation of a hyperbolic sine and hyperbolic cosine, which is supported by the FPGA. [27] designed fully combinational hardware for an RBF kernel neural network. They use the Taylor series expansion of the exponential function and a look-up table to perform kernel calculations. Specifically concerning the exponential functions, [28] optimized the hardware implementation using the Taylor expansion of the exponential functions in general. Finally, [29] recently presented a stochastic architecture that can perform multiple different types of kernel calculations for SVM models. Their design provides lower hardware complexity and for the RBF kernel higher accuracy.

k-Nearest Neighbors

The k-NN algorithm consists of two parts: finding the nearest neighbors and computing the average y value. The latter is relevant for this thesis research and the hardware solutions will be discussed in subsection 2.2. Finding the nearest neighbors has a varied collection of hardware solutions presented in the literature.

[30] presents a new method to perform approximate NN search by quantizing the feature space to improve search quality and memory usage. [31] designed a set of very high throughput parallel FPGA architectures using Manhattan distance calculation. [32] presents an accelerator that can change its precision and they also compare the performance differences between Euclidean and Manhattan distance calculations. Lastly, [33] combines in-memory computing SRAM and top-k sorting hardware to accelerate a k-NN model.

Random Forest

The hardware implementation of random forest has a very wide range of solutions in literature ranging from basic digital CMOS arithmetic to in-memory computing [4] to the use of special transistor technologies [34], [35]. As this thesis remains within the bounds of digital CMOS arithmetic, the following solutions exist within the design space.

[36] tested the performance of reduced precision RF models on FPGA using stochastic rounding. [37] designed hardware architectures for three different types of decision tree ensembles focusing on area optimization. Their designs include a pipeline of a couple of decision nodes or a fully sequential design with only a single comparison being done at a time. [38] implemented multi-valued decision diagram based RFs on an FPGA for extra fast and energy-efficient inference and compared the result to GPU and CPU implementations. They also did experiments to see the relationship between bit-precision and classification accuracy. [39] also looked at FPGA implementation, but they explored the trade-offs of three levels of reconfigurability. Depending on the ML application there will be higher or lower constraints on the time the architecture needs to switch from one forest to another. But faster switching times mean higher usage of FPGA resources. Finally, [40] proposes an early stopping mechanism from random forests to save energy and time. The idea is simply to stop inference when sufficiently many trees in the forest provide the same classification, leading to shorter classification times for easier inputs.

2.1.5. SML Requirements

This subsection gives an overview of all the required arithmetic operations and data storage necessary to perform inference with the SML model and explains what common denominators were identified within the SML model's learners that allow for optimization.

Figure 2.1 shows the flow of data through a system that can perform inference with the SML model. The system has one input and one output. The input is a set of 30 feature values of a data point of which the lifetime is to be predicted, the independent variable x. These 30 features are measurements of electrical or environmental parameters of the IC. The single output is a prediction of remaining lifetime, dependent variable y.

Inside the Inference System are blocks that represent arithmetic operations or stored data. Between these blocks are colored arrows that visualize how data from the system's input and memory are used for arithmetic operations and how the results of these operations are used in other operations or are presented at the output of the system. For example, in Linear Regresssion (yellow), 5 of the 30 feature values from the input and 5 weight values from memory are combined in the Dot Product block to produce a prediction of lifetime. At the bottom of the figure, this lifetime prediction joins five others to become the RF Meta-Learner's (red) input.

Memory Blocks

For organization, all the memory blocks are placed together near the center of the system. Although each block represents the storage of data with a certain function in the SML model, the figure is not meant to say anything about how this memory is implemented. In an actual hardware design, this data may well be stored at different addresses within the same memory unit that is shared with the entire embedded system, or be stored in separate specialized memory units. The function of each block is as follows:

- **Data Memory:** The Data Memory stores the training data points. These are used by the Nearest Neighbors (blue), kernel regression (two light shades of green) and SVR (dark green) learners, as previously shown in equations 2.6, 2.3, 2.4 and 2.5 respectively.
- **Weight Memory:** The Weight Memory contains the trained weight values for the Linear, Kernel and SVR regression learners.
- **Threshold memory:** The Threshold Memory contains the thresholds for each of the three decision nodes in each of the 100 decision trees in both the RF base-learner (purple) and the RF meta-learner (red).
- **Leaf Memory:** The Leaf Memory contains the lifetime prediction values that belong to each of the 4 leaves of each of the 100 decision trees of both the RF base-learner and the RF meta-learner.

Arithmetic Blocks

The arithmetic blocks in the figure represent mathematical operations necessary for inference using the SML model, but do not give any indication as to how they could be implemented in hardware. The function of each block is as follows:

- **Kernel Calculator:** The Kernel Calculator block takes two inputs and gives one output. The input should be two data points (containing 5 features each) and the output is the result of a kernel between the two data points. RBF, Polynomial and Linear kernel calculations should be supported.
- **Nearest Neighbor Finder:** The Nearest Neighbor Finder block calculates the Euclidean distance between two data points (each containing 5 feature values). The y values of the three nearest neighbors are sent to the Averager block.
- **Dot Product:** The Dot Product block produces a dot product with a five feature value $\hat{x}$ and a five value weight vector $\hat{w}$. This is the only operation within the Linear Regression base-learner.
- **MAC operator:** The MAC operator block multiplies and accumulates the $K(x, c_i)$ and α weights for the kernel regression and SVR base-learners.
- **Averager:** The Averager block takes the average of either the y values from the Nearest Neighbor Finder or the predictions from the trees in the RF learners.

- **Decision Tree Solver:** The Decision Tree Solver block compares five feature values from the input and compares it many times with different threshold values from memory. It outputs the leaf nodes that the comparisons lead to such that the associated lifetime values can be retrieved from memory. This is the main operation of the two RF learners.

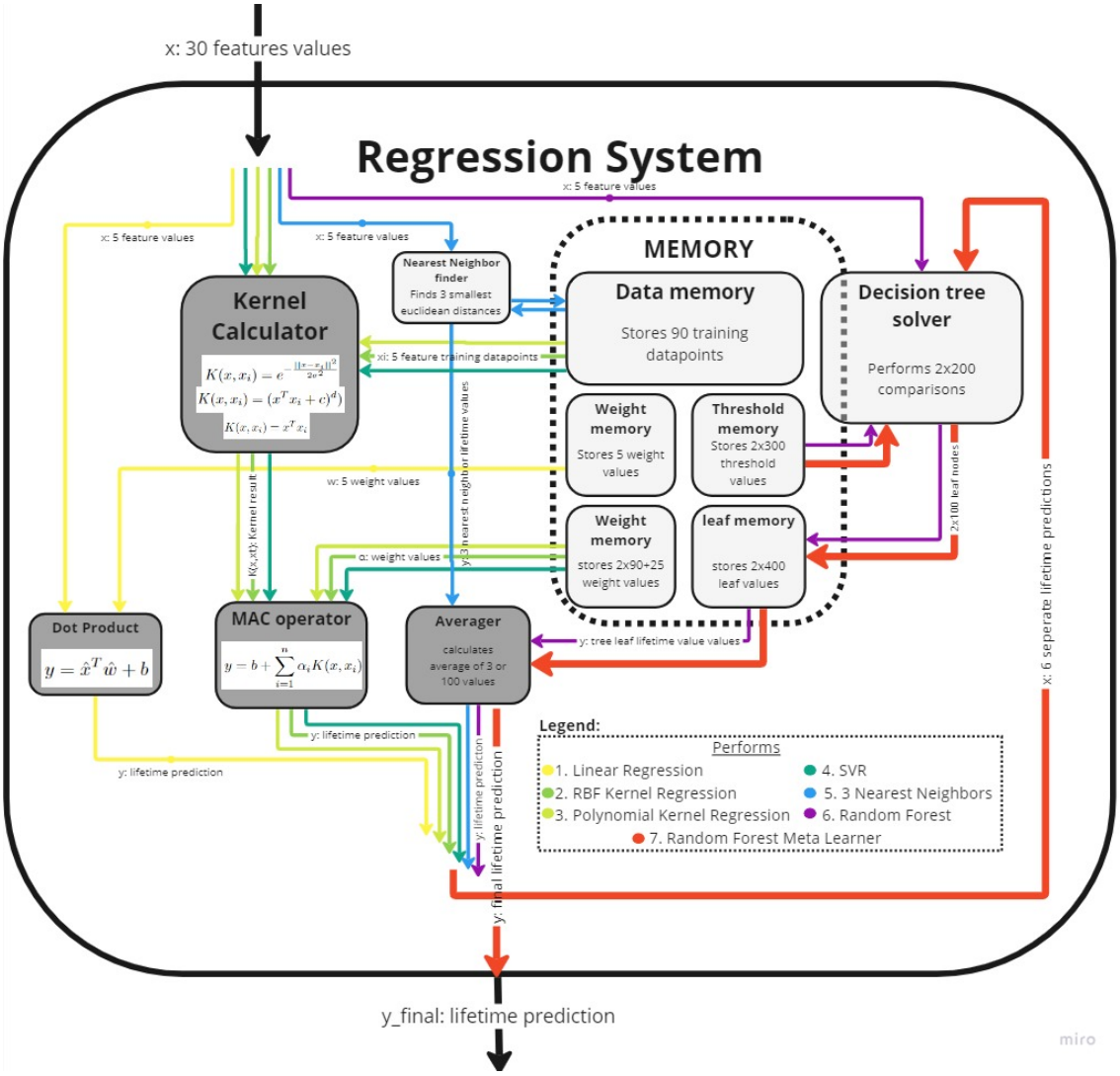


Figure 2.1: Data flow diagram of the SML model. Each learner is colour-coded.

Common Denominators

Using figure 2.1, it becomes easier to identify common denominators in the hardware requirements of the base- and meta-learners of the SML model. The Kernel Calculator, Dot Product, MAC operator and Averager blocks were selected in this thesis to design an area and energy-optimized unit for. The reason for this choice comes from a combination of their high arithmetic intensity and the similarities among the workloads of these four blocks.

The Decision Tree Solver also has a relatively intensive workload, however, the comparison operations are quite different from all the other blocks and the more complex control makes integration with other blocks more difficult. From subsection 2.1.4 it is also clear that there an many existing solutions for RF models. These factors indicate little potential for innovation relevant to the SML model.

The Nearest Neighbor Finder does resemble the operations of the kernel calculator and the MAC operator and could be integrated with them. However, optimal hardware design for this operation would depend heavily on the types of searching algorithms and distance measures that are best for the specific k-NN learner in the SML model, which is best left for separate research.

The most mathematically intensive block in the inference system is the Kernel Calculator and it also shares MAC, multiply, accumulate and division operations with the other three grey-shaded blocks. Although there is considerable research into separate Kernel Calculation hardware, for an optimal area implementation resources likely need to be shared with other blocks.

The Dot Product and MAC Operator blocks are mainly extremely similar to each other and to a lesser extent to both the Kernel Calculator and Averager due to the high level of multiplication and accumulation respectively. They are therefore central in any unified hardware design for the SML model.

Lastly, the Averager shares division with the Kernel Calculator and accumulation with the MAC operator. Crucially, the Averager also requires the implementation of a divider with a very light workload, which indicates a high potential for area reduction if hardware resources can be shared with more arithmetically intensive blocks.

Section 3.1 will go into more detail about the arithmetic unit workload that results from the combination of the Kernel Calculator, Dot Product, MAC operator and Averager blocks.

2.2. Related Works

There is no real state of the art hardware for stacked machine learning models. This forms the obvious gap in research that this thesis addresses. For this reason, most of this section focuses on existing solutions to some of the sub-problems of designing hardware for an SML model. The best solutions in literature will also serve as inspiration during the design process covered in methodology chapter 3. To provide some point of comparison for the SML arithmetic unit designed in this thesis, this section ends by presenting a piece of literature about a general-purpose system that would be relatively well suited for the SML workload.

Of course, this makes quantitative comparison with the result of this thesis difficult. However, more general purpose plat Instead of existing hardware for SML models, this related works section only discusses and compares existing solutions to some of the sub-problems of designing hardware for an SML model. The best solutions in literature will also serve as inspiration during the design process covered in methodology chapter 3.

Subsection 2.2.1 explains the state of Area Efficient Multipliers. Subsection 2.2.2 talks about multipliers that can operate at different levels of bit-precision. Lastly, subsection 2.2.3 covers the existing ways to combine multiplication and division into one arithmetic unit.

2.2.1. Area Efficient Multipliers

The arithmetic unit design in this thesis is centered around multiplication. It is a combination of one of the most common and resource-intensive operations that must be supported by the arithmetic unit. With the embedded application's stringent requirements on area, the state of the art on area efficient multipliers is very important to discuss.

Sequential Multiplier

Multipliers come in two main categories that can each be optimized for Area efficiency in their own way. In general, a reduction in area is traded off for increase in delay. Nothing represents this trade-off more than sequential multipliers. Sequential multipliers are a class of multipliers that characterize themselves with having registers that save intermediate results such that logic gates can be reused in the next clock cycle when the multiplication continues. Every cycle only a part of the multiplier is multiplied with the multiplicand, which requires fewer logic gates to perform. The results of these smaller multiplications are accumulated until the full multiplication is completed. The radix of the multiplier indicates how many of the multiplier bits are multiplied with the multiplicand per clock cycle. For instance, a radix-2 multiplier only multiplies one bit per cycle, while a radix-16 multiplier multiplies 4 bits per cycle.

Sequential multipliers are generally smaller than parallel multipliers. Although at similar clock speeds the delay is much larger, the shorter critical path allows for higher frequency operation. The main drawback of sequential multipliers is higher energy consumption, which is a result of the activity of the registers that does not occur in parallel multipliers.

Sequential multipliers were more popular in the past when larger feature sizes made large amounts of logic expensive and power dissipation caused less of a bottleneck. However, some relatively recent papers propose energy improvements. Of particular interest is the radix-16 sequential multiplier by

[41]. The preliminary calculation of $3X$ (three times the multiplicand) reduces CSA (carry save adder) stages and as such reduces the the critical path length and energy consumption. [42] also proposed in improvement on the power consumption of sequential multipliers by introducing a set of measures aimed at reducing activity. This led to a 30% decrease in power consumption, however also a 34% increase in area.

Parallel Multipliers

When the entire multiplication is performed in one clock cycle, we speak of parallel (or combinational) multipliers. These multiplications happen in three steps: Partial Product Generation, Reduction Tree and Final Addition. Multiplier designs often focus on optimizing one of these steps and can therefore sometimes be combined. The simplest multiplier to implement is the array multiplier. In an NxN bit multiplication the N^2 partial products (PP) are generated with XOR gates and these PPs are then combined by a set of half- and full-adders that form a perfect N by N array, which functions as both the Reduction Tree and the Final adder [43].

Although it is very small, the array multiplier is not optimized for delay and also has a relatively high energy consumption. More optimally designed reduction tree multipliers, like Wallace and Dadda trees, significantly reduce delay, but trade this in for some extra area and much higher complexity. Energy consumption can be both higher or lower depending on the specific design. A particularly small reduction tree multiplier is the Reduced Area (RA) multiplier, which simultaneously maintains a similar delay to Wallace trees [44].

Parallel multiplication can also be optimized by changing the generation of the partial products. The most famous example of this is Booth Encoding which reduces the number of PPs by encoding them. This significantly increases the complexity of the multiplier but can lead to better area, energy and delay [43].

2.2.2. Precision-Scalable Multipliers

Quantization (reducing bit-precision) of ML model can often reduce area, energy and delay in exchange for relatively little reduction in model performance [45]. This is no different in the SML model discussed in this thesis. However, not all models can run on the same precision [7]. Hardware support for multiple levels of bit-precision can therefore be a uniquely important consideration for the area and energy-efficient hardware design for a Stacked ML model.

Conveniently, [46] benchmarked a large selection of MAC units for the specific application of Embedded Neural Networks. However, for our purposes, it provides an invaluable comparison for the state of the art. Figure 2.2 shows how Sub-Word Parallel (SWP) MACs perform energy and area efficiently at both full precision and half precision. Meanwhile, Divide and Conquer (D&A) designs are slightly better for half precision, especially area-wise, but at full precision, which is expected to be common the the Stacked ML model, it performs significantly worse. In this thesis, SWP MACs, presented by [47] and further developed by [48], will be regarded as the state of the art.

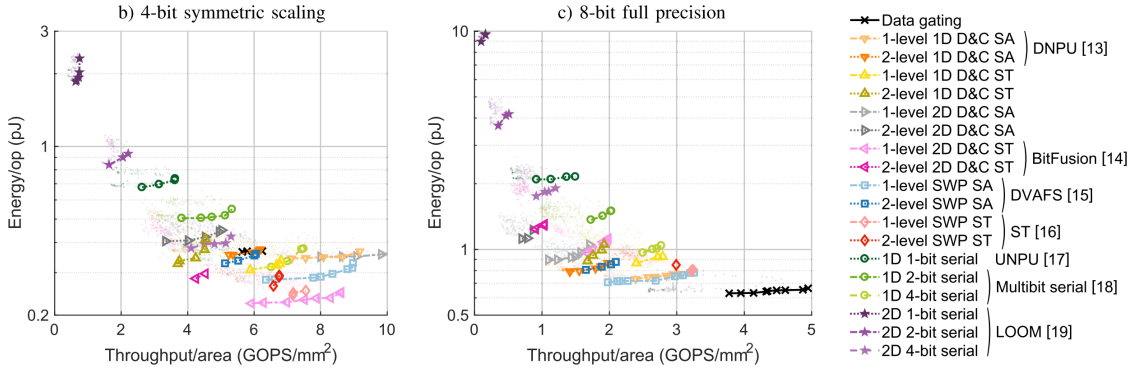


Figure 2.2: Performance comparison of a set of configurable precision MAC units [46, p.708]

2.2.3. Combined Multiplier and Divider

Another relevant form of arithmetic unit for a stacked ML model is combined multiply and divide units. These may offer an efficient way to provide the ability to perform division for the 3-NN and RF learners despite the overall uncommon occurrence of the operation.

True Divider

There have been a few designs combining multiplication and division, often including square roots [49]–[51]. The only recent one combines 32-bit multiplication and division through radix-2^{16} multiplication and radix-512 division. If this design was to be used for the SML model, it would have to be redesigned to operate at a lower radix to perform 16-bit multiplications and division. All three of the designs are also notably sequential designs, as this is the most practical way to perform true division. However, this also limits multiplication to strictly sequential operation, which may not be optimal for the SML design.

Invariant Integer Division

Due to a unique trait of the SML model there is one more option that is arguably the most promising. Invariant Integer Division (IID) is a method of division where only a multiplication, an addition and a shift operation are used. It is only possible under the condition that the divisors of all the necessary divisions are integers and known during the compilation of the program, i.e. invariant [52] [53]. For the SML model this means that all dividers must be integers and when the model is loaded into memory, the divisor must be known. As long as this is true, IID can provide division with very little hardware overhead. This method of division appears to be the best suited for our area and energy-efficient arithmetic unit, but due to the lack of literature discussing ASIC implementations of the method, there is no indication of its performance.

RISC-V Core for IoT applications

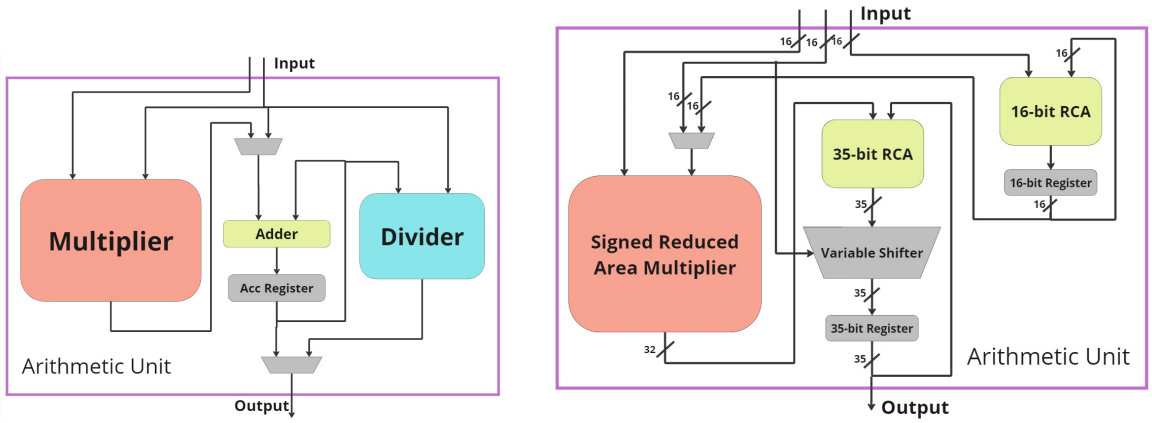
Finding a hardware platform that can run the workload of the SML model is not a challenge. However, the platform has to compare with the SML model specific, low energy and area, arithmetic unit designed in this thesis. Though there are many ML accelerators specifically designed for edge applications, they are almost always model-specific. The most popular model, neural networks, does not often feature division. If they do, their operations can not be freely programmed as required for the SML model. Finally, there are the usual difficulties when comparing the performance of two hardware designs. They have to match in aspects like technology node, supply voltage, bit width, in- or exclusion of memory and/or control.

[54] presents a small and energy-efficient core with relatively general-purpose computing capabilities. It supports division, multiplication, accumulation and MAC operations at various levels of bit precision. However, not all perfect for comparison with this thesis. Firstly, the RISC-V core was designed in 65-nm CMOS technology, which is two generations older than the 28-nm used here. However, the paper does provide estimated performance figures for a 28-nm process. The next issue is the missing power and throughput data for a supply voltage of $0.9V$. The power and throughput were presented to be $1 - 68mW$ and $0.15 - 2.35GOPS$ at supply voltages of $0.46 - 1.1V$. The best solution here is to linearly extrapolate between the supply voltages to obtain the power and throughput at $0.9V$. In regards to the circuit area, $0.68mm^2$ was provided for the 65-nm technology node. Thankfully, with the gate equivalent of $46.9kGE$ that is reported, it becomes possible to estimate the area at 28-nm using a NAND gate area of $0.378\mu m^2$. Finally, there is the issue that the entire core is designed for a maximum operand bit-width of 32 bits. This is double the size needed for the SML model. The only way to compare to the 16-bit arithmetic unit in this thesis, is by a fairly blunt estimation of what would happen to the area, power and throughput if this core would operate at 16-bit precision. The area and power are assumed to reduce by 75% and the throughput is expected to grow by 100%. The area, power and throughput figures of the RISC-V core at $0.9V$ supply and in 28-nm technology would then be $4432\mu m^2$, $11.77mW$ and $3.325GOPS$. In the discussion chapter, these values will be translated into the area, energy and delay KPIs used in this thesis.

3

Methodology

This chapter lays out the design process of the arithmetic unit. It started with a basic template design, shown in figure 3.1a, and ended with the final design, shown in figure 3.1b. This chapter starts with a clear definition of the workload of the arithmetic unit to be designed, section 3.1. The rest of the chapter contains a detailed description of the design process which is executed in three phases. Section 3.2 describes the architecture-level design phase in which three critical considerations about the arithmetic unit were made. Next, section 3.3 introduces an experiment done to further validate the results of the architecture-level design phase. Finally, section 3.4 covers a set of further improvements that were considered to optimize the arithmetic unit design. These improvements were often inspired by observations in earlier results.



(a) Arithmetic Unit Template

(b) Final Arithmetic Unit design

Figure 3.1: The template design, with which the design process started, next to the final area and energy efficient design.

3.1. Workload

In the literature review section 2.1.5, SML Requirements, the Kernel Calculator, Dot Product, MAC operator and Average blocks were introduced, and shown in figure 2.1. The area and energy efficient arithmetic unit to be designed in this thesis must support the functions of all four of these blocks. To perform useful design exploration, it is important to know in detail what the workload of all of those blocks is.

This section splits the total workload into a *Basic Workload* and a *Kernel Workload*. The Basic Workload originates from the Dot Product, MAC operator and Averager blocks. The Kernel Workload originates from the Kernel Calculator. The metric for this split is based on how characteristic the operations are for SML models in general. The Basic Workload covers the more common operations in any stacked machine learning model, including but not limited to the SML model central to this thesis. The

composition of the Kernel workload depends greatly on the number and the types of kernels used in the SML model. Besides this, the Basic Workload also serves an important role as a more manageably-sized workload, while being similar in composition to the entire workload. This will be useful for setting up the experiments performed in this thesis. The two workloads are laid out in table 3.1 and 3.2, and explained in two subsections.

3.1.1. Basic Workload

Table 3.1 shows an overview of the workload. The Basic Workload consists mainly of MAC operations from the Dot Product, and MAC Operator blocks, and accumulation from the Averager block.

Table 3.1: An overview of all the arithmetic operations required to perform the Basic Workload to the arithmetic unit to be designed. The operations fall under four bold printed categories and each originate from one of three arithmetic blocks and one of six ML models.

Arithmetic Blocks	ML models	MACs	Multiplications	Divisions	Additions
Dot Product	Ridge Regression	5	0	0	1
MAC operator	RBF Kernel Regression	90	0	0	1
	Polynomial Kernel Regression	90	0	0	1
	Support Vector Regression	5	0	0	1
Divider	3-Nearest Neighbors	0	0	1	3
	Random Forest	0	0	2	200
	Total	**190**	**0**	**3**	**207**

Dot Product Workload

The Dot Product block only supports the Ridge Regression base-learner. This is a linear regression model with only a dot product between a 5 feature input vector $\hat{x}$ and a 5-value weight vector $\hat{w}$. This means a workload of 5 MAC operations.

MAC Operator Workload

The MAC Operator block supports the MAC operations of 3 base-learners between the Kernel $K(x, c_i)$ values and the weights α. The RBF Kernel Regression model creates kernels between 90 input vectors and 90 training data points. Each of these kernels is multiplied and accumulated with a weight α. This is the same for the Polynomial kernel regressor, meaning together they require 180 MAC operations. Finally, the SVR model only creates kernels for 5 support vectors, meaning 5 MAC operations instead of 90. In total, the MAC Operator Workload is 185 MAC operations.

Averager Workload

The Averager block supports the averaging for the 3-NN and RF base-learners and the RF meta-learner. Averaging means the summation of a number of values, then division by that number of values. For the 3-NN model, this is 3 accumulations and one division by 3. The RF models have 100 predictions from 100 trees to take the average of. In total, the Average Workload is 203 accumulations and 3 divisions (by 3, 100 and 100).

3.1.2. Kernel Workload

The overview of the Kernel Workload is shown in table 3.2. The Kernel Workload originates only from the Kernel Calculator block. There are three different types of Kernel values $K(x, c_i)$ to be calculated: RBF, Polynomial and Linear Kernels. In total, the Kernel Workload involves 1375 MAC operations, 450 Multiplications, 90 Divisions and 540 Additions. The makeup of this workload is covered per kernel type in the next three paragraphs.

RBF Kernel Workload

The workload of the RBF kernel is based on equation 2.3 and the implementation of [27]. By keeping in mind that in the SML model there are 5 features per data point and 90 data points to calculate the kernel for, it becomes possible to calculate the required arithmetic operations. $\frac{||x-x_i||^2}{2\sigma^2}$ requires 5 additions, 5 MAC operations and 1 division. The exponent of e is calculated with one addition, 4 multiplications and 5 MAC operations. Multiplying by 90 for the number of training data points, the RBF kernel workload is: 450 Additions, 900 MAC operations, 360 multiplications and 90 divisions.

Table 3.2: An overview of all the arithmetic operations required to perform the Kernel Workload to the arithmetic unit to be designed. The operations fall under four bold printed categories and each originate from one of three arithmetic blocks and one of six ML models.

Arithmetic Blocks	ML models	MACs	Multiplications	Divisions	Additions
Kernel Calculator	RBF Kernel Regression	900	360	90	450
	Polynomial Kernel Regression	450	90	0	90
	Support Vector Regression	25	0	0	0
	Total	**1375**	**450**	**90**	**540**

Polynomial Kernel Workload

The workload of the Polynomial kernel can easily be determined from kernel equation 2.4. The dot product between $\hat{x}$ and $\hat{c}_i$ is performed in 5 MAC operations because both vectors contain 5 features. The addition of b one addition and power of d means one multiplication, because $b = 2$ in the SML model. Multiplying those operations by 90 for the number of training data points results in a Polynomial kernel workload of 90 additions, 450 MAC operations and 90 multiplications.

Linear Kernel Workload

Lastly and most simply, the Linear kernel is expressed with equation 2.5 which performs the dot product of two 5 feature vectors for each of the 5 support vectors. This mean 25 MAC operations.

3.2. Architecture-Level Design Phase

In the Architecture-level design phase three critical design aspects were considered. The focus is on how multiplication and division is implemented and how the two operations should relate to each-other.

3.2.1. Arithmetic Unit Template and Design Aspects

During the architecture-level design, the focus is on the Basic Workload, because this is the most characteristic workload for a stacked machine learning arithmetic unit and it simplifies the acquisition and analysis of the results. The Basic Workload also effectively requires support for the same arithmetic operations as the Kernel workload. These are MAC operations, accumulation and division. This design stage will make use of a template, shown in figure 3.1a, which will be used as a sort of base design from which three critical design aspects will be explored. The Multiplier and Divider are presented as black boxes, yet to be designed, and the structure of the accumulation is worked out in a simple and somewhat area-efficient way such that can be used for both MAC operations and accumulations. There will be little attention to this last part of the template during this design stage as the impact of the accumulator design is minimal when compared to the multiplier and divider.

Three Critical Design Aspects

There are three critical design aspects that are expected to have a large impact on the area, energy and delay performance and also the architectural structure of the arithmetic unit.

The first concerns the nature of the multiplier: Whether the multiplier is sequential to some level, or fully parallel. This aspect is expected to have a large impact on the area, energy and delay of the arithmetic unit because multiplication makes up a very significant part of both the Basic and the Kernel workloads. Once a conclusion about this basic aspect of the multiplier is made, further exploration into specific multiplier designs can be done.

The second design aspect is the choice between a separate divider, that is independent from the multiplier, or some form of combined division, where division is integrated with the multiplier. Specifically, the separate divider will be True Division (TD), which can do division between any 16-bit number at any time. The form of combined division that will be considered is Division by Invariant Integer (IID) introduced in State of the Art subsection 2.2.3. IID is such a promising simplification of the division problem, surely achieving a reduction of delay and likely also area, that it must be compared to an area-efficient divider in an early stage of design.

The third design aspect is the option of precision configurability. It is important to explore to what extent energy and/or delay can be reduced through this method and whether this can lead to a viable trade-off compared to its drawback of increased area. To simplify the options, the focus of precision

configurability will be on the multiplier. As mentioned, the multiplier has the largest impact on the energy and delay, which configurability may be able to reduce.

To determine the best choice in each of these three critical design aspects, each will be evaluated by their performance in the three Key Performance Indicators (KPI), area, energy and delay. The next subsection will introduce three multiplier designs and two divider designs that will each represent one of the options that arise from the three design aspects.

3.2.2. Design Aspect 1: Parallel or Sequential Multiplier

This subsection describes the two designs that represent the parallel and the sequential options for multipliers. The parallel multiplier is an array multiplier, and the sequential multiplier is a radix-16 multiplier.

Sequential Radix-16 Multiplier

The sequential multiplication option is based on the Radix-16 sequential multiplier proposed by [41]. The main change is the addition of precision configurability, which is the only considered option in the domain of sequential multipliers in this design phase. The reason for this is that no change in the behavior of the multiplication is required to do lower precision multiplication. The multiplier simply has to do fewer cycles to do lower precision multiplication. The only addition in terms of area is a mux that can decide how much the output needs to be shifted to display the correct multiplication result.

The entire multiplier is shown in figure 3.2. The way this particular 16-bit sequential radix-16 multiplier works is as follows. The multiplication is performed over the course of 4 cycles. Every cycle four bits of the multiplier are multiplied with the multiplicand. This 4- by 16-bit multiplication is achieved through the addition of the result of the previous cycle with two new values. These two values are chosen based on the 4 multiplier bits that are used in the current cycle. This particular multiplier creates 4 options for each of the new values to be added in this cycle by performing addition between the multiplicand and doubled version of itself (The RCA in the Precomputing part of figure 3.2). After the 4 multiplication cycles are complete the remaining values in the sum and carry registers are added together to become the upper half of the multiplication result.

Precision configurability is achieved by decreasing the number of multiplication cycles and shifting the result right according to the precision. Due to the multiplier being radix-16, the supported levels of precision are 16-, 8- and 4-bit, taking 4, 2 and 1 cycles to complete respectively.

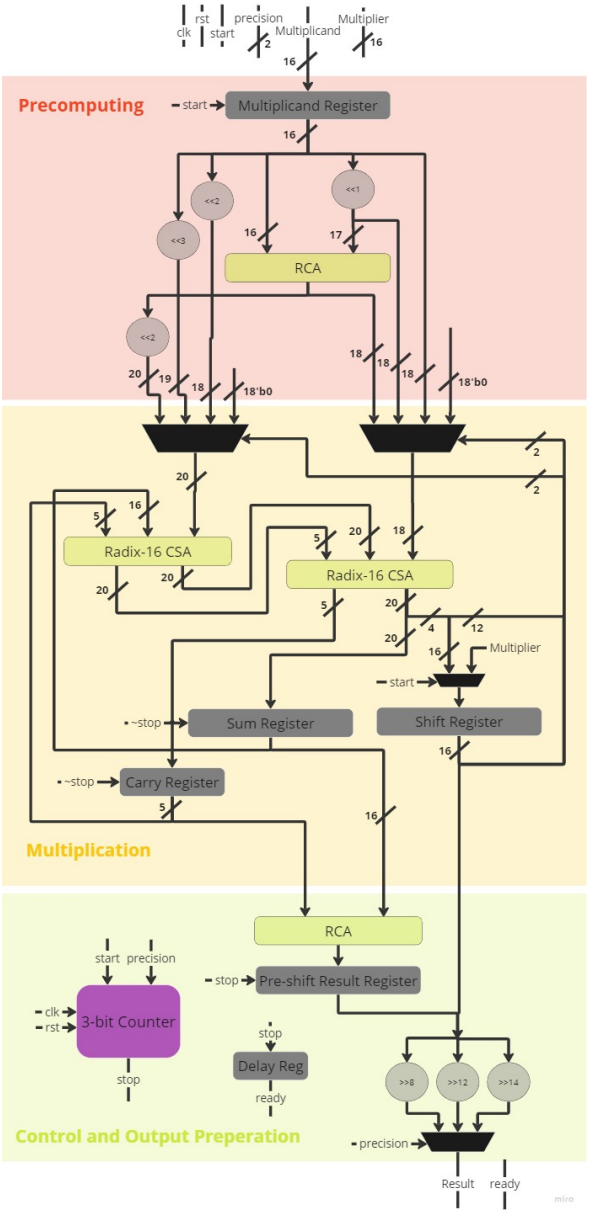


Figure 3.2: Radix-16 Sequential Multiplier schematic

Array Multiplier

The Array Multiplier represents the design option of a non-configurable fully parallel multiplier and was chosen for its low complexity and area. It

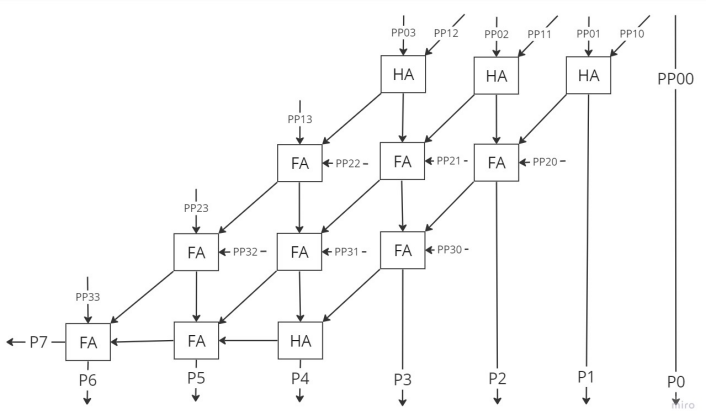


Figure 3.3: 4-bit Array Multiplier

will also be most comparable to the chosen configurable parallel multiplier.

Figure 3.3 shows an Array Multiplier for the multiplication of two 4-bit numbers. Scaled up to 16-bit multiplication, the Array Multiplier requires 256 AND gates for PP generation and 15 Half Adders (HA) and 225 Full Adders (FA) for reduction and final addition.

3.2.3. Design Aspect 2: Separate or Combined Division

This subsection describes the proposed designs both division implemented separately from the multiplier and combined with the multiplier. The separate design is a low-area non-restoring radix-2 divider. The combined design is Invariant Integer Division, making use of the existing multiplier.

Sequential radix-2 divider

The True Divider option is represented by a radix-2 non-restoring sequential divider. The schematic of this divider is shown in figure 3.4. This divider was chosen because it is close to the smallest way a dedicated divider can be implemented in terms of area. As the number of divisions in the workload is low, a divider with less delay or higher energy efficiency will have little impact on the KPIs of the arithmetic unit. A non-restoring radix-2 divider is also quite simple to understand and implement.

The divider has two non-control-related inputs: The divider and the dividend. The goal of the division is to obtain the quotient. In radix-2 division, this is achieved over the course of 16 clock cycles, where in every cycle one bit of the quotient is obtained. At the start of division the dividend is put into the Quotient Register and the divider into the Divider Register. Each cycle of the division goes as follows. First, the most significant bit of the Quotient Register is moved from the Quotient Register to the Remainder Register. Then, the sign of the Remainder Register is checked. If it is positive, the divisor must be subtracted from the remainder, thus the value in the Divisor Register is inverted and added to the values in the Remainder Register, together with a least significant bit '1' to correctly change the sign through 2's compliment notation. These two 16-bit values and a single '1' bit are added together within the same 16-bit RCA, visible in green in figure 3.4. If instead the sign bit indicates that the Remainder Register contains a negative value, the positive version of the divisor is added to the remainder.

Every cycle, one correct quotient digit is obtained from the new remainder and is saved in the least significant bit location of the Quotient Register. After 16 clock cycles all bits of the dividend have been shifted out of the Quotient Register and all 16 bits of the correct quotient have been saved. At that point the Quotient Register contains the results of the division and presents it at the single 16-bit output of the division module.

3.2.4. Invariant Integer Division

Invariant Integer Division is achieved using equation 3.1 to 3.4 [55]. Here, x is the bit-precision of the division, D is the divider and N is the dividend. If D is an invariant integer, division using these four equations is possible. What makes IID valuable for the SML arithmetic unit, is that equations 3.1 and 3.2 can be performed by the processor used for training the SML model. As soon as the number of

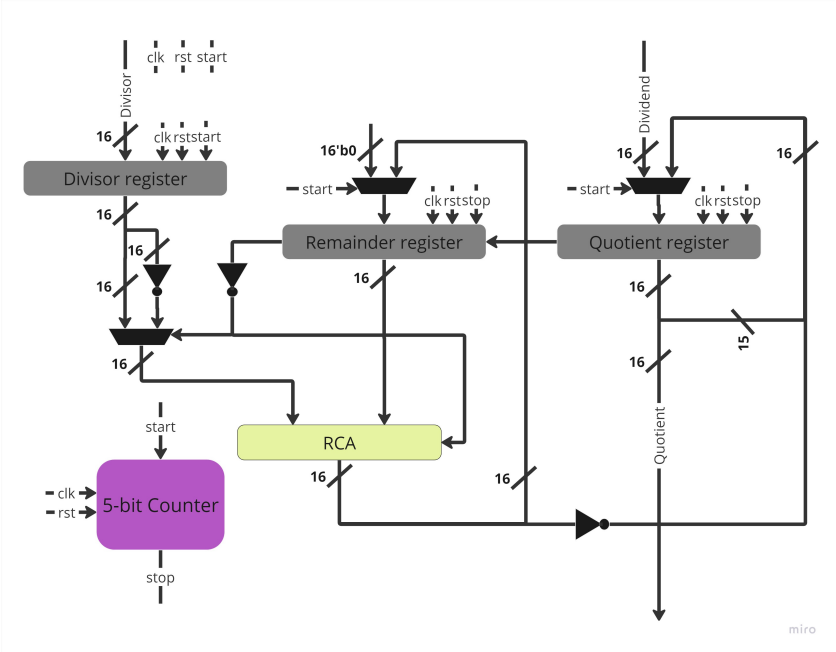


Figure 3.4: Radix-2 Non-restoring Sequential Divider Schematic

trees in the RF models and the k in the k-NN model are known D and x of equations 3.1 and 3.2 are also known.

$$k = x + \lceil log_2 D \rceil \tag{3.1}$$

$$a = \left\lceil \frac{2^k}{D} \right\rceil - 2^x \tag{3.2}$$

$$b = \left\lfloor \frac{Na}{2^x} \right\rfloor \tag{3.3}$$

$$\left\lfloor \frac{N}{D} \right\rfloor = \left\lfloor \frac{\lfloor \frac{N-b}{2} \rfloor + b}{2^{k-x-1}} \right\rfloor \tag{3.4}$$

On the SML inference hardware, which includes the arithmetic unit, the following two things happen. One, k and a are stored in memory and provided to the arithmetic unit. Two, during division, the arithmetic unit performs equation 3.3 and 3.4. This only requires the arithmetic unit to support multiplication, addition and a variable shifting of the output. This is visualized in figure 3.5.

For the implementation of IID into the SML hardware it was possible to simplify the equation describing the operation. This simplification is shown on the right side of equation 3.5. $\lfloor \frac{Na}{2} \rfloor$ is calculated by the multiplier followed by a bit shift to the right. Here, a comes from memory and N from the result register (N is the result from the previous accumulation operations performed for averaging). The addition of $N2^{x-1}$ is done by the accumulator, after the obvious $x-1$ bit shift to the right. Finally, the newly implemented Variable Shifter performs the final shifting by k. Where k is also provided by the memory.

$$\left\lfloor \frac{N}{D} \right\rfloor = \left\lfloor \frac{\lfloor \frac{N-b}{2} \rfloor + b}{2^{k-x-1}} \right\rfloor = \left\lfloor \frac{\lfloor \frac{N-\lfloor \frac{Na}{2^x} \rfloor}{2} \rfloor + \lfloor \frac{Na}{2^x} \rfloor}{2^{k-x-1}} \right\rfloor = \left\lfloor \frac{\lfloor \frac{Na}{2} \rfloor + N2^{x-1}}{2^{k-1}} \right\rfloor \tag{3.5}$$

3.2.5. Design Aspect 3: Precision Configurability

This subsection describes in detail the design of the Sub-word Parallel Array Multiplier, which is the proposed application of hardware configurability to support different levels of bit precision inside the parallel multiplier option.

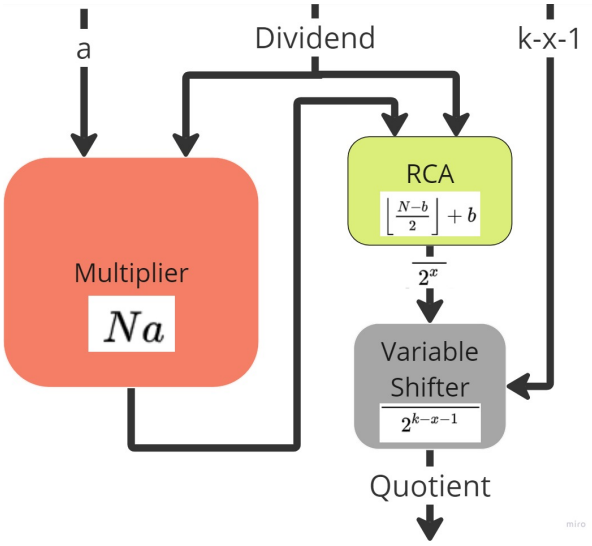


Figure 3.5: Invariant Integer Division

Sub-word Parallel Array Multiplier

For the precision configurable parallel multiplier a Sub-Word Parallel Array Multiplier was designed based on the work of [48]. As mentioned in State of the Art section 2.2.2, the precision-scalable multiplier performs best at full precision compared to the other options and is easily compared to the non-precision-scalable array multiplier.

For the architecture-level design phase, the Sub-Word Parallel Array Multiplier is implemented entirely by changing the generation of the Partial Products and performing a shifting operation after the multiplication. Besides the usual full 16-bit precision, 8-bit and 4-bit precision are also supported in the configurable design. During 8-bit precision multiplication, the four operands of the two 8-bit multiplications are expected to be presented to the arithmetic unit's two input operands in accordance with equations 3.6. For the eight operands of 4-bit precision multiplication, this should be done according to equations 3.7. In these equations, a is to be multiplied with b, c with d, e with f and g with h.

$$operand1 = \{a_{7-0}, c_{7-0}\}, \quad operand2 = \{d_{7-0}, b_{7-0}\} \tag{3.6}$$

$$operand1 = \{a_{3-0}, c_{3-0}, e_{3-0}, g_{3-0}\}, \quad operand2 = \{h_{3-0}, f_{3-0}, d_{3-0}, b_{3-0}\} \tag{3.7}$$

Inside the confarray multiplier module, there are muxes that can force to zero the Partial Product bits of pairs of operands that are not to be multiplied. For example, the Partial Product between bit b_1 and bit c_0 should be forced to '0' because b and c are not to be multiplied by each other. This is visualized with table 3.3 for a smaller multiplier that supports 8, 4 and 2-bit multiplication.

Table 3.3: A visualization of how undesired Partial Products are forced to zero during reduced precision multiplication

(a) 4-bit precision

	a_3	a_2	a_1	a_0	c_3	c_2	c_1	c_0
b_0	PP_{70}	PP_{60}	PP_{50}	PP_{40}	0	0	0	0
b_1	PP_{71}	PP_{61}	PP_{51}	PP_{41}	0	0	0	0
b_2	PP_{72}	PP_{62}	PP_{52}	PP_{42}	0	0	0	0
b_3	PP_{73}	PP_{63}	PP_{53}	PP_{43}	0	0	0	0
d_0	0	0	0	0	PP_{34}	PP_{24}	PP_{14}	PP_{04}
d_1	0	0	0	0	PP_{35}	PP_{25}	PP_{15}	PP_{05}
d_2	0	0	0	0	PP_{36}	PP_{26}	PP_{16}	PP_{06}
d_3	0	0	0	0	PP_{37}	PP_{27}	PP_{17}	PP_{07}

(b) 2-bit precision

	a_1	a_0	c_1	c_0	e_1	e_0	g_1	g_0
b_0	PP_{70}	PP_{60}	0	0	0	0	0	0
b_1	PP_{71}	PP_{61}	0	0	0	0	0	0
d_0	0	0	PP_{52}	PP_{42}	0	0	0	0
d_1	0	0	PP_{53}	PP_{43}	0	0	0	0
f_0	0	0	0	0	PP_{34}	PP_{24}	0	0
f_1	0	0	0	0	PP_{35}	PP_{25}	0	0
h_0	0	0	0	0	0	0	PP_{16}	PP_{06}
h_1	0	0	0	0	0	0	PP_{17}	PP_{07}

Finally, the result of the multiplication has to be shifted to the right depending on the precision of the multiplication that is occurring. During full precision, there is no shift. During half/8-bit precision, the result is shifted right by 8 bits. Finally, during quarter/4-bit precision, the result is shifted right by 12 bits.

Table 3.4: Percentage by which each type of operations occurs within the total number of operations of the Basic, Kernel and combined workloads.

Workload	Multiplication + MAC	Division	Accumulation + MAC	Total number of operations
Basic	0% + 47.5% = 47.5%	0.75%	51.75% + 47.5% = 99.25%	100%
Kernel	21.25% + 64.92% = 86.17%	4.25%	9.58% + 64.92% = 74.50%	100%
Basic + Kernel	17.87% + 62.15% = 80.02%	3.69%	16.28% + 62.15% = 78.44%	100%

3.3. Workload Variation Experiment

As discussed in section 3.1, the arithmetic unit's workload is split into a Basic and a Kernel Workload. Only the Basic Workload is used during the architecture-level design phase to provide a less complex workload that contains the operations most typical for a Stacked Machine learning model. However, without considering the kernel workload it is not possible to claim that the arithmetic unit is well suited to perform the kernel operations of the SML model. As such, this section introduces the experiment used to verify whether the optimal architecture-level design is also best suited for the Kernel Workload.

This experiment does not involve setting up a very complex program that can measure the performance of each architecture-level design on exactly the workload expected from the particular kernels in the SML model. Instead, it relies on the fact that the most significant difference between the Basic and Kernel Workloads is a different number of "multiplication involving operations", referring to multiplications as well as MAC operations. Table 3.4 shows how much percent of each workload each type of operation occurs. Here, MAC operations are counted for both multiplications and accumulations, because during a MAC operation, both a multiplication and an accumulation occur.

The most important numbers to compare in table 3.4 are the percentages of multiplication, division and accumulation in the Basic + Kernel Workload versus just the Basic Workload. The percentage of multiplications increases by 32.5%. This is the most important difference between the Basic workload and the total SML workload. Firstly, because it is the largest absolute increase. But also because multiplication has a particularly large impact on KPIs.

The 20.8% decrease in accumulations is not nearly as important, as additions are far less expensive in terms of the area, energy and delay to perform, they have much less impact on the performance of the arithmetic unit as a whole. Although the number of divisions does increase with the biggest factor factor, the divisions still make up such a small part of the workload that their impact on the KPIs of the arithmetic unit is minimal.

For this reason, the experiment proposed in this section will involve only testing the changes in the KPIs of the 6 architecture-level designs under different percentages of multiplication in the workload. Specifically, this will be done by altering the number of accumulation operations in the Basic Workload. Increasing the number of accumulations in the workload decreases the multiplication percentage, and vice versa. By obtaining KPI figures for the architecture-level design at a wide range of multiplication percentages, it will become possible to conclude which design is best for which workload and whether or not that is the same design for both the Basic and the combined Workloads.

3.4. Further Design Improvement Phase

This section aims to optimize the best-performing architecture-level design from the previous section by proposing changes that either improve performance or add necessary functionality. The four improvements that will be discussed in the next four subsections are Signed Multiplier, Configurable Signed Multiplier, Dedicated Accumulator and Reduced Area Multiplier.

3.4.1. Signed Multiplication

Signed multiplication is a trait that was always going to be necessary for the real workload of the SML model. However, designing signed versions of the three different multipliers would require an investment of time and effort that was not expected to return interesting results. The impact of signed multiplication is expected to be similar for all multiplier designs.

Signed multiplication was implemented in the array multiplier using the Baugh-Wooley algorithm [56]. This algorithm achieves signed multiplication by inverting any partial product that is produced from the sign bit of one of the two input operands. Besides this, a single '1' bit is added to the result at significance $n/2$ where n is the number of bits in the input operands. Finally, the sign bit of the result

is also inverted. These alterations are visualized for a 4-bit multiplier in figure 3.6. Here, the partial products that are inverted are printed in red, and the '1' bit is fed to the carry input of the final RCA.

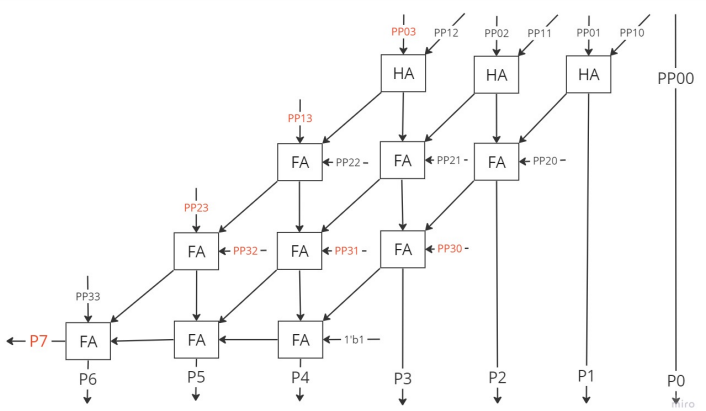


Figure 3.6: Invariant Integer Division

3.4.2. Configurable Signed Multiplier

The results of the implementation of signed multiplication (results section 4.3.2) show that the energy consumption of signed multiplication is much higher than that of unsigned multiplication. This is cause to consider reconfigurability in the signed multiplier.

Although little energy reduction was achieved by the Sub-Word Parallel Array Multiplier in the architecture-level design phase, this can be explained by the fact that unsigned low-precision multiplications do not cause switching activity on the left side of the array multiplier. For example, when multiplying two 8-bit numbers in a 16-bit multiplier, $PP_{12,8}$ from bits a_{12} and b_8 will remain '0' and cause not switching activity. Unsigned low-precision multiplications happen relatively efficiently on an unsigned non-configurable array multiplier.

In a signed array multiplier, negative values insert a large number of '1' bits into the higher significance left side of the array. Returning to the same example, $PP_{12,8}$ can still be '1' if a and b are negative values. Reconfigurability may be able to solve this. However, unlike the Sub-Word Parallel Array Multiplier, it is not possible to simply block inputs to large sections of the array. The inversion of partial products of the Baugh-Wooley algorithm, earlier visualized in figure 3.6, needs to be moved to different locations in the array as well [56][57].

The best-performing way to achieve the configurable signed multiplier found in this design phase was to only support 16- and 8-bit precision. The reconfiguration between the two levels of precision happens in two stages. Firstly, if the precision is 8-bit, the upper half of the input operands are forced to 0 to prevent switching activity in the upper end of the multiplier. Secondly, the partial products that must be inverted to achieve signed multiplication at the two levels of precision have an extra gate that switches between inverting or not inverting its input depending on the precision. These two steps are shown for an 8-bit multiplier in figure 3.5. There were also designs tested that supported 4-bit precision and designs where only the PPs were gated instead of the inputs. However, they performed slightly worse. Their performance will also be shown in the Results chapter.

3.4.3. Dedicated Accumulator

The dedicated accumulator is the simplest of the further design improvement attempts. In sections 3.1 and 3.3 about the workload of the arithmetic unit it became clear that a significant part of the operations in the workload only involve accumulation. These accumulations are performed by taking the average of a set of values that originate from the 3-NN and RF models. So far, these 16-bit values are being accumulated in a large post-multiplication accumulator and occupying valuable clock cycles of the arithmetic unit. This is inefficient. As such, in this subsection, a dedicated accumulator is proposed.

The dedicated accumulator is added to the arithmetic unit with the idea that it allows the arithmetic unit to spend fewer cycles on accumulation. Therefore, a dedicated accumulator also needs dedicated connections to memory. This means an extra arithmetic unit input is added, called "operand3", which is

Table 3.5: A visualization of how in 4-bit versus 8-bit precision both the inputs and the Partial Product are changed. In 4-bit precision, the more significant half of the input operands are forces to '0' and the inverted PPs are moved.

(a) 8-bit precision

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_0	PP_{70}	PP_{60}	PP_{50}	PP_{40}	PP_{30}	PP_{20}	PP_{10}	PP_{00}
b_1	PP_{71}	PP_{61}	PP_{51}	PP_{41}	PP_{31}	PP_{21}	PP_{11}	PP_{01}
b_2	PP_{72}	PP_{62}	PP_{52}	PP_{42}	PP_{32}	PP_{22}	PP_{12}	PP_{02}
b_3	PP_{73}	PP_{63}	PP_{53}	PP_{43}	PP_{33}	PP_{23}	PP_{13}	PP_{03}
b_4	PP_{74}	PP_{64}	PP_{54}	PP_{44}	PP_{34}	PP_{24}	PP_{14}	PP_{04}
b_5	PP_{75}	PP_{65}	PP_{55}	PP_{45}	PP_{35}	PP_{25}	PP_{15}	PP_{05}
b_6	PP_{76}	PP_{66}	PP_{56}	PP_{46}	PP_{36}	PP_{26}	PP_{16}	PP_{06}
b_7	PP_{77}	PP_{67}	PP_{57}	PP_{47}	PP_{37}	PP_{27}	PP_{17}	PP_{07}

(b) 42-bit precision

	0	0	0	0	a_3	a_2	a_1	a_0
b_0	0	0	0	0	PP_{30}	PP_{20}	PP_{10}	PP_{00}
b_1	0	0	0	0	PP_{31}	PP_{21}	PP_{11}	PP_{01}
b_2	0	0	0	0	PP_{32}	PP_{22}	PP_{12}	PP_{02}
b_3	0	0	0	0	PP_{33}	PP_{23}	PP_{13}	PP_{03}
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

fed directly to the dedicated accumulator. The accumulator does not have to be as large as the existing one. 16 bits is all that is needed. If the result of the accumulator exceeds 16 bits, it cannot enter the multiplier for the IID operation.

This does introduce a constraint to the precision of the 3-NN and RF models. However, this is inherent to the size of the multiplier and if the 3-NN or the RF model cannot adhere to it, it would not be impossible to increase the size of the multiplier. On top of this, there are arguments for why the two models will not need higher precision. The constraint on the 3-NN model is relatively minor. The size of the outputs of the Nearest Neighbor finder needs to be decreased from 16 to 14 bits. For the RF model, experiments have shown that it can like perform quite well at 4-bit precision [7]. Accumulating 100 4-bit values stays nicely below a 16-bit result. That is why the current setup with a 16-bit dedicated accumulator and a 16-bit multiplier is expected to be sufficient for the SML model.

3.4.4. Reduced Area Multiplier

The final design improvement attempt revisits the multiplier. Although the current array multiplier is small, the delay and energy consumption are sub-optimal. The most worthwhile multiplier to consider in this short optimization section is the RA multiplier previously mentioned in State of the Art section 2.2.1. Introduced by [44], the RA multiplier follows a simple concept. The RA multiplier design is only a description of its reduction tree. At every level of the reduction tree, any set of 3 Partial Product bits is added together using a Full Adder (FA). This reduces the number of PPs quickly and will eventually lead to some of the least significant bits of the result being obtained before the final RCA stage. However, in the RA multiplier, extra Half Adders are employed on the least significant bit that still contains PPs to increase the speed at which result bits are obtained. This reduces the width of the final RCA and therefore saves area.

As a result of this reduction tree, the RA multiplier achieves less delay and energy for approximately the same area as the array multiplier. Working out the number of Full- and Half adders leads to an area of 222 FAs and 16 HAs. This is slightly less than the 225 FAs and 15 HAs of the array multiplier. The critical path of the RA multiplier is significantly less than the array multiplier at only 8 FA delays and 22 HA delays, while the array multiplier takes 16 FA delays and 14 HA delays.

4

Results

This chapter shows and analyses the results of the architecture-level design phase, the workload experiment and the further design improvement phase. These three phases are covered in separate sections. Each section starts with the experimental setup, then discusses the results, and finishes by presenting the conclusion of the design phases.

4.1. Architecture-Level Design

4.1.1. Experimental Setup

All the experimental results in this thesis were obtained through simulation and synthesis using Cadence Xcelium and Genus. First, the hardware designs were worked out in Verilog and tested for their functionality using a testing testbench. Then a Basic Workload testbench was written for each design. Lastly, a bash script was written that aided in running Xcelium and Genus on a range of frequencies for each design.

To visualize and analyze the results, a different bash script was used to extract the slack, area and power numbers from the results. These frequency, slack, area and power numbers were combined into Figure of Merit values using excel and plotted using Matlab. Details about specific steps in this process now follow.

Workload Testbench

The Workload testbenches are different for each design taking into account the different ways that each design needs to be controlled and have data supplied to it. However, each testbench has the same goal: Realistically representing the execution of the Basic Workload on each of the designs. However, the testbenches are still synthetic, meaning it does not perform the arithmetic operations with the exact values and in the exact order that the SML model would. The reason for representing the workload in this manner is the fact that the internal values of the base- and meta-learners were not readily available with the *sklearn* library used to design the models. The order in which each operation of the workload should be executed is also unknown, as the details of this would depend on the speed and behavior of surrounding hardware like the Decision Tree Solver and the memories, which have not yet been designed.

Each testbench performs the 190 MAC operations, the 207 additions and the 3 divisions of the Basic Workload. First, the MAC operations are performed in 3 loops that each do 5 MAC operations, then reset the arithmetic unit. The first loop runs 15 times at a precision of 16 bits. The second loop also runs 15 times but at a precision of 8 bits. The third loop only runs 8 times and at a precision of 4 bits. This precision distribution was chosen based on the reduced precision performance of the SML model [7].

Then the accumulations and divisions are combined such that they perform the three averaging operations that they are supposed to. First 100 accumulations are performed with random values that are small enough to not cause an overflow, then a division by 100. This is repeated one more time for 100 values and then for 3 values.

Results generation

Three bash scripts were used in the generation of the results. The first, generate_vcd, is supplied a design name, a range of frequencies and a precision distribution. The script then runs Xcelium for that design at the desired frequencies and precision distribution (precision distribution refers to the number of MAC loop cycles at 16-,8-, and 4-bit precision). The vcd activity files are saved in a folder for the next script to use.

The next script, frequency_sweep, must be supplied with a design name and a range of frequencies. It runs genus for that design at each of those frequencies and uses the vcd activity files previously obtained.

Finally, the last script, frequencies_sweep_data, collects the slack, area and power data from all of the result files and saves it in a text file that can be exported to excel.

Figure of Merit

To aid in comparing the results of the 7 different architecture-level designs and the various further design improvement attempts, a Figure of Merit was defined. This FoM takes into account the three important KPIs of area, energy and delay and their relative importance. Equation 4.1 shows the formula by which the FoM is calculated for any design at any frequency.

$$FoM = \frac{10^8}{Area^3 Energy^2 Delay^1} \quad \text{for } Slack \geq 0 \tag{4.1}$$

In this FoM equation, area, energy and delay are located inside the denominator because larger values in each of these three KPIs means that the design performs worse. Area is raised to a factor of 3 to represent its status as the most important KPI. Similarly, Energy has less impact with a factor of 2 and delay is the least important with a factor of 1. In the numerator of the fraction 10^8 is added. This is done to shift the results of the FoM equation back to the range of integers to aid in the reading of plots. Finally, the FoM equation is only used if the slack of the design is positive. Slack refers to the time difference between the longest signal path and the clock period. If the slack of a design is negative, it means that the synthesis tool could not find an architecture that could adhere to the clock frequency. Therefore, if the slack is negative, the FoM is set to 0 because the design is not functional at that frequency.

4.1.2. Results

Figure 4.1 and 4.2 show the performance of the designs with parallel and sequential multipliers respectively. The quality of each design is expressed on the y-axis in the Figure of Merit defined in equation 4.1. The synthesis results were collected for clock frequencies from 20 to 600 MHz in steps of 20 MHz. The designs are labeled in the legend with a naming convention that first states the multiplier of the design, then the divider, separated by an & symbol.

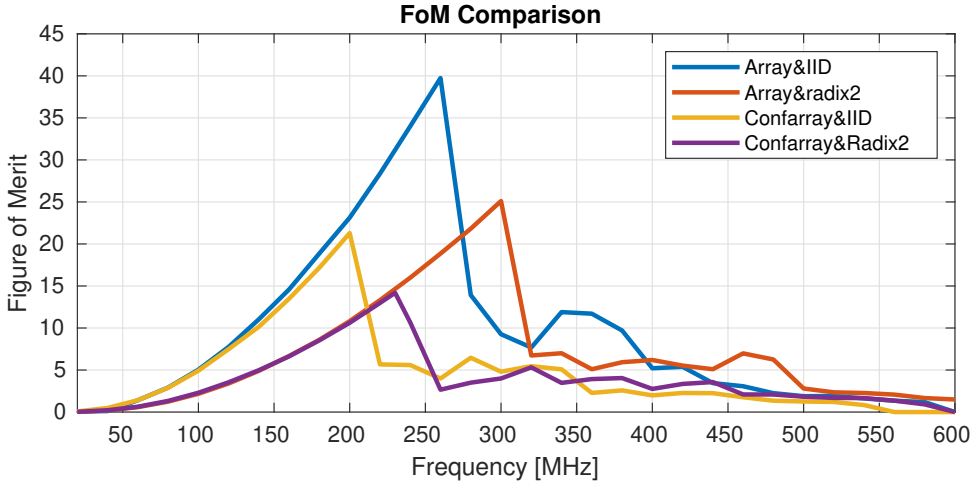


Figure 4.1: Performance comparison of the architecture-level designs containing parallel multipliers at different clock frequencies. Performance is expressed with a Figure of Merit that is based on area, energy and delay measurements.

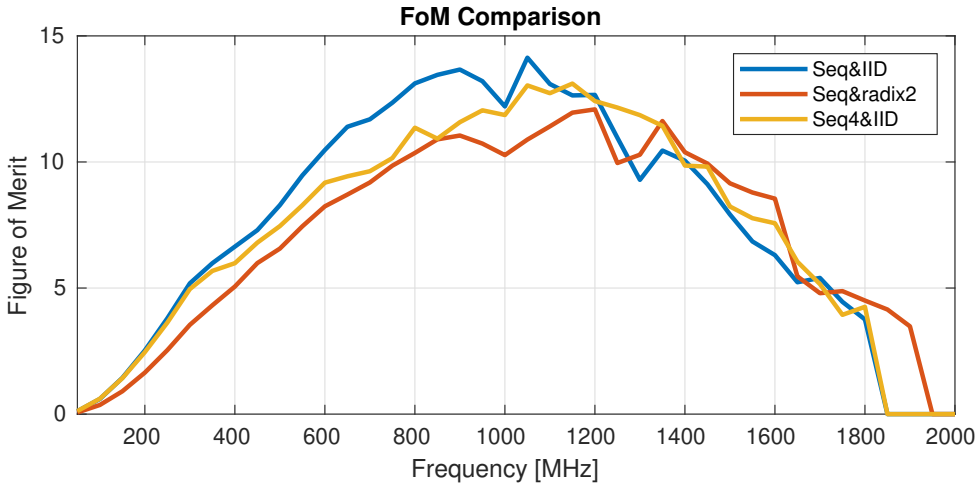
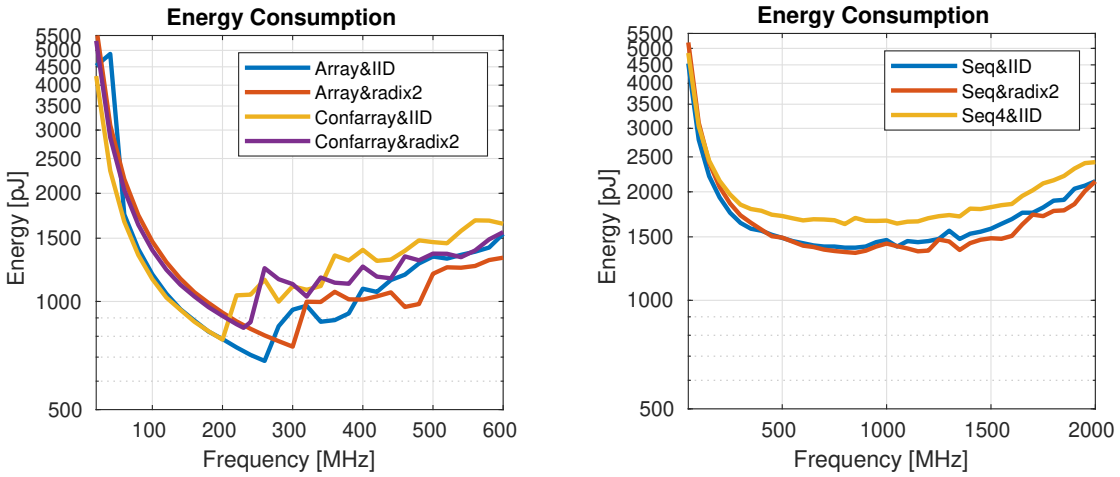


Figure 4.2: Performance comparison of the architecture-level designs containing sequential multipliers at different clock frequencies. Performance is expressed with a Figure of Merit that is based on area, energy and delay measurements.

Initial Observations

The first thing to spot about these results is the difference in the scale of the axes between figures 4.1 and 4.2. The x-axis of the sequential design results goes up to 2 GHz compared to 600 MHz on the parallel designs. This is of course a choice made in the collection of the results, but it is caused by the entirely expected observation that the sequential designs perform best, and maintain positive slack, at far higher frequencies.

The y-axis reveals a more interesting observation. The FoM of all the sequential designs is much lower than almost all the parallel designs. This is entirely a result of the energy consumption of the sequential designs being much higher. It was expected that the energy consumption of the sequential designs would be higher due to the extra activity of storing intermediate results. However, the higher energy consumption has more impact than expected. The lowest energy consumption that any sequential design can achieve is 1.64 to 2 times higher than the lowest energy consumptions of the parallel designs. Comparing the lowest energy consumptions makes sense as the optimal operation frequency of each design (highest FoM) is mostly determined by its energy consumption.



(a) Architecture-level designs with parallel multipliers

(b) Architecture-level designs with sequential multipliers

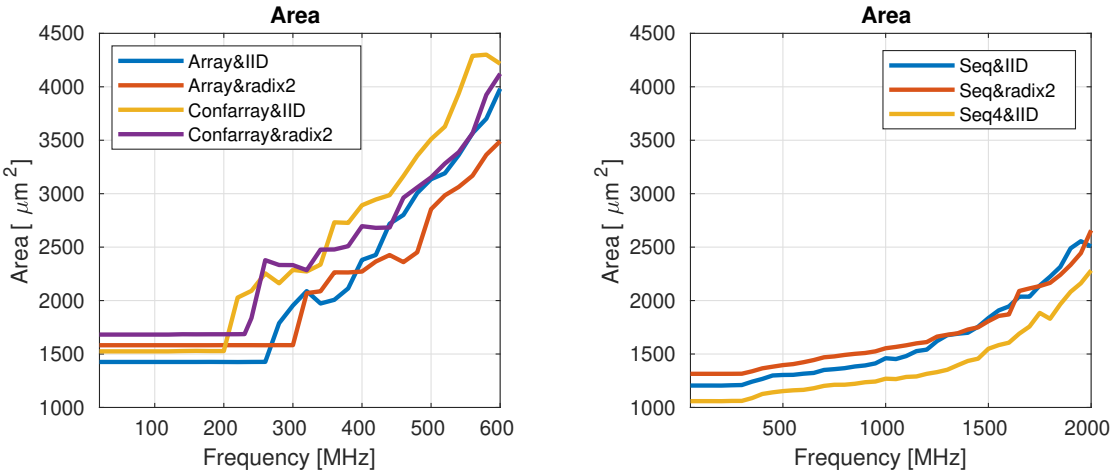
Figure 4.3: The energy consumption of each architecture-level design to run the Basic Workload

Continuing to the lines inside plots 4.1 and 4.2, all 7 designs display a common pattern. The FoM is lowest at the highest and the lowest frequencies and peaks somewhere in the middle frequencies. This hill shape is a very normal phenomenon. When moving towards the lowest frequencies, the area

decreases but approaches a minimal non-zero area. Meanwhile, the delay increases continuously towards infinity and the energy increases too, due to the increasing run time and the static power of the hardware which does not depend on frequency.

On the high frequency side, the hardware resources needed to achieve shorter clock periods make the area grow exponentially. The delay decreases only linearly and the energy consumption follows the growth of the area as more and larger transistors increase the static and dynamic power. When the clock frequency becomes too high the synthesis tool can no longer find a way to shorten the critical delay path to adhere to the short clock periods and the resulting negative slack means the hardware will not function. This negative slack sets the FoM to zero, which is clearly visible on the right side of figure 4.2.

Although the sequential designs display a very natural hill shape, leading to a couple of good-FoM frequencies for each design, the parallel designs instead show a large spike in FoM at very specific frequencies. When looking at the area plot of the parallel designs in figure 4.4, it becomes possible to formulate some explanation for these very specific optimal frequencies. What seems to happen is that there is a certain minimal area implementation of the parallel designs which causes the area to plateau at lower frequencies. With the area fixed, and presumably, also the exact synthesized design fixed, the energy consumption and delay become solely dependent on the frequency. This leads to the smooth exponential increase of FoM with frequency that occurs on the left side of the FoM peaks of the parallel designs.



(a) Architecture-level designs with parallel multipliers

(b) Architecture-level designs with sequential multipliers

Figure 4.4: The Area of each architecture-level design

On the right side of the peaks, the sharp drops in the figure of merit are the result of the sudden jumps in area that occur between 200 and 300 MHz. Clearly, the genus synthesizer determines that a lot of hardware has to be added to adhere to the shorter clock periods. However, it is not clear why so much of this additional hardware has to be added between these specific frequencies. For example, why does Array&IID not need any additional hardware between 20 and 260MHz, a very large amount between 260 and 300MHz, and then less again between 300 and 400MHz? The best explanation is that a lot of the signal paths in the low-frequency designs had the same propagation delay. Once clock periods finally became shorter than this collection of critical paths, a lot of restructuring and increasing of transistor sizes was necessary.

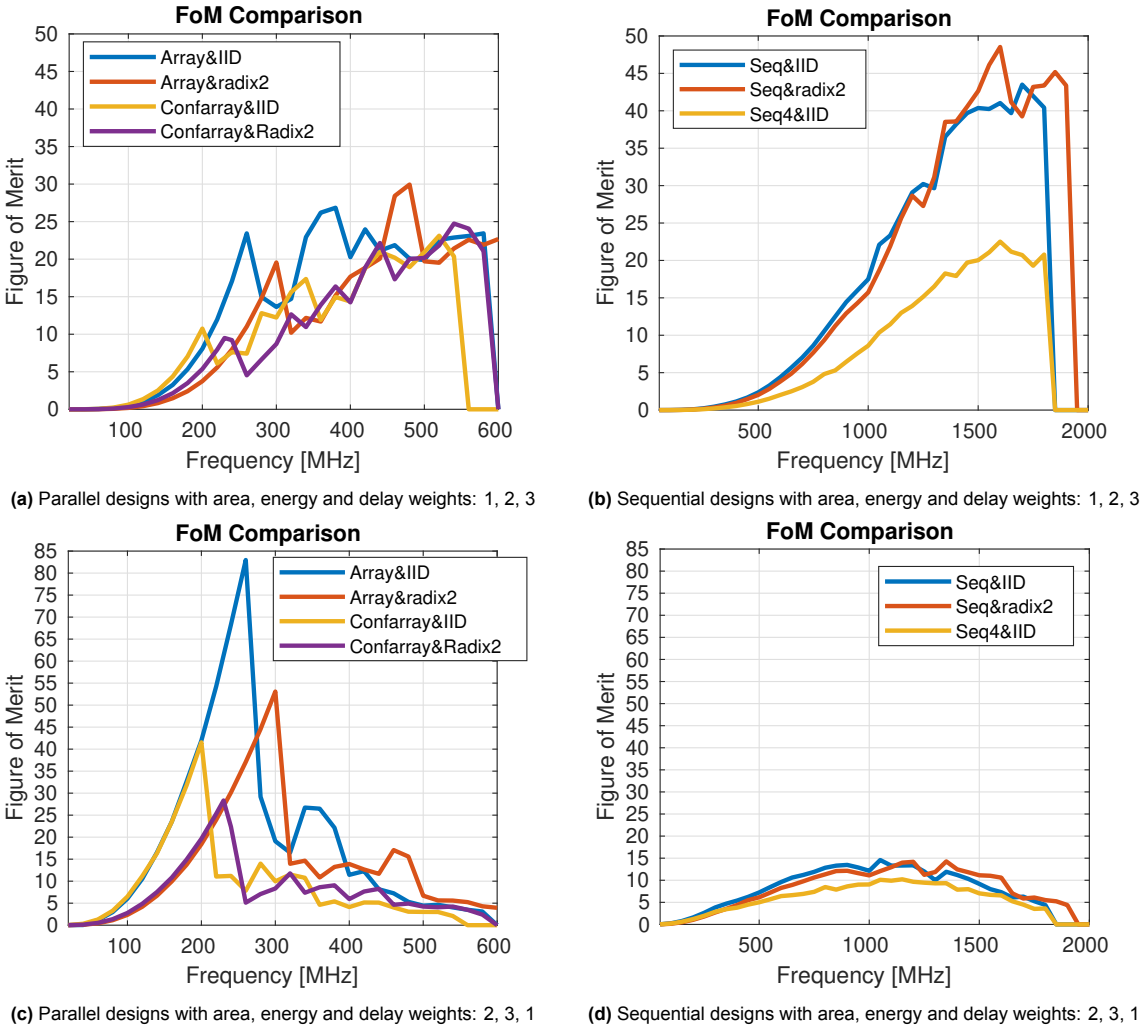
Important Design Observations

Looking at the differences between individual lines in the FoM figures (4.1 and 4.2), it becomes possible to make the observations that are most important for this design stage. Firstly, designs with Invariant Integer Division (IID) universally reach higher FoM than designs with True Division (radix2). This is the result of lower energy and area at lower frequencies. In the parallel designs, the longer critical path of the IID designs led to an earlier jump in area, after which the IID designs stop out-competing the TD designs. However, at their FoM peak, the IID designs clearly win in terms of area and energy, and have only a slightly longer delay.

The second observation is that the type of multiplier has the largest impact on performance and the non-configurable parallel multiplier (array) delivers the best KPIs. As mentioned before, the sequential multiplier leads to such a significant increase in energy consumption that it cannot compete with the parallel multipliers. While comparing the configurable and non-configurable multipliers, it is interesting to observe that the two multipliers have almost the same FoM and the same energy consumption. The added area on the configurable array trades off perfectly with decreased delay within the FoM formula. However, the longer critical path on the configurable multiplier designs means that more area is needed to achieve higher frequencies when compared to non-configurable multiplier designs.

Different KPI Weights

Before concluding this architecture-level results section, it is interesting to look at the impact of the FoM formula. Doing this could be useful for future reference when changes in the context of the application domain of this SML model's arithmetic unit lead to different levels of importance of the 3 KPIs. The FoM equation 4.1 gives the highest importance to the area by raising the variable to the power of 3. This means that area is given a weight of 3. As such, energy is given a weight of 2 and delay a weight of 1. Figure 4.5 shows the FoM plots at different distributions of the weights given to the three KPIs.



(a) Parallel designs with area, energy and delay weights: 1, 2, 3

(b) Sequential designs with area, energy and delay weights: 1, 2, 3

(c) Parallel designs with area, energy and delay weights: 2, 3, 1

(d) Sequential designs with area, energy and delay weights: 2, 3, 1

Figure 4.5: The FoM of the architecture-level designs with different weights assigned to the three KPIs. The area, energy and delay weight are 3,2 and 1 respectively in equation 4.1

The first new distribution of weights is area, energy, delay: 1, 2, 3. One could imagine that a distribution like this would apply to something like a supercomputer, not constrained by area, and focused on the highest performance. In this context, there are two interesting observations. Firstly, Sequential multiplier designs, in particular the radix-16 multiplier, start greatly out-competing the parallel designs.

Secondly, True Division becomes more valuable than IID. Both of these phenomena have the same cause. Both sequential multiplication and True Division have shorter critical paths, resulting in less heavy area and energy trade-offs at the higher clock frequencies that help drop the delay as low as possible.

The second new distribution of weights is area, energy, delay: 2, 3, 1. This represents energy critical edge applications; scenarios like Internet of Things applications or medical devices like pacemakers. Here, the similarity to this thesis' automotive lifetime prediction application becomes apparent in that no different observations can be made. Sequential multipliers still perform worse than parallel multipliers, albeit even worse, and array multipliers and IID outperform their alternatives.

4.1.3. Architecture-Level Conclusions

Having analyzed the results of the architecture-level design phase, a couple of conclusions had a steering effect on the rest of the design process. Firstly, sequential multipliers cannot compete with parallel multipliers in an area and energy-critical application and therefore, the arithmetic unit should contain a parallel multiplier. Secondly, Invariant Integer Division is a more energy and area efficient way to perform division and therefore is the superior choice for an SML model's arithmetic unit. Thirdly, precision configurability does not provide energy improvement and does not decrease delay if the clock frequency is a variable that can be chosen freely. Therefore, it is not a viable option for the arithmetic unit. There are two limited scenarios where precision configurability could provide better performance. This is if the precision distribution of the workload is greatly skewed to lower precisions. Or if the clock frequency is fixed at a relatively low value and the importance of area is decrease relative to the importance of delay.

4.2. Workload Variation Experiment

4.2.1. Experimental Setup

For the workload variation experiment, the same RTL design files are used as in the architecture-level design phase. The collection of the results is performed differently. A new variable inside the basic workload testbench allows for the variation of the number of additions performed. By increasing the number of additions, the percentage of multiplication involving operations (MAC operations and multiplication) decreases. A new bash script runs the Xcelium simulation and the genus synthesis for various levels of multiplication percentage in the workload. The script requires as input the design name, a clock frequency, and a range of multiplication percentages. Inside the loop of the script, the multiplication percentages are translated to numbers of additions, which are then changed within the workload testbench. The results in the following subsection were obtained by setting the clock frequency of each design to exactly the frequency at which the FoM was highest in the previous experiment, and the range of multiplication percentages was set to be from 5% to 95% in steps of 5%.

4.2.2. Results

Figure 4.6 shows the results of the workload variation experiment. The x-axis is labeled to be the percentage of multiplication involving operations within the workload. The y-axis shows the Figure of Merit. There are 7 lines, one for each architecture-level design, synthesized at each of their optimal clock frequencies. From this figure already some observations can be made about the performance of each design depending on the percentage of multiplication of the workload. However, by making use of a simple transformation of the data, the behaviors become more apparent. This was done to create the next figure. Figure 4.7 shows a normalized version of the same data. The y-axis shows by how much % each design deviates from the mean FoM at that multiplication fraction. This makes it much easier to compare the designs with each other, especially at lower multiplication fractions, where the absolute FoM nears 0.

A couple of observations can be made from figure 4.7 about the FoM behavior of the architecture-level designs when the fraction of multiplications in the workload changes. The most notable feature is that all parallel multiplier designs show a rising trend across the entire range, while all sequential multiplier designs have a falling trend. This is entirely expected, as the sequential designs have less hardware dedicated to multiplications. Meanwhile, accumulation, the other large fraction of the workload, has one RCA and one register dedicated to it across all 7 designs. Therefore, at very low multiplication fractions, the sequential design benefits from having less hardware dedicated to an operation that

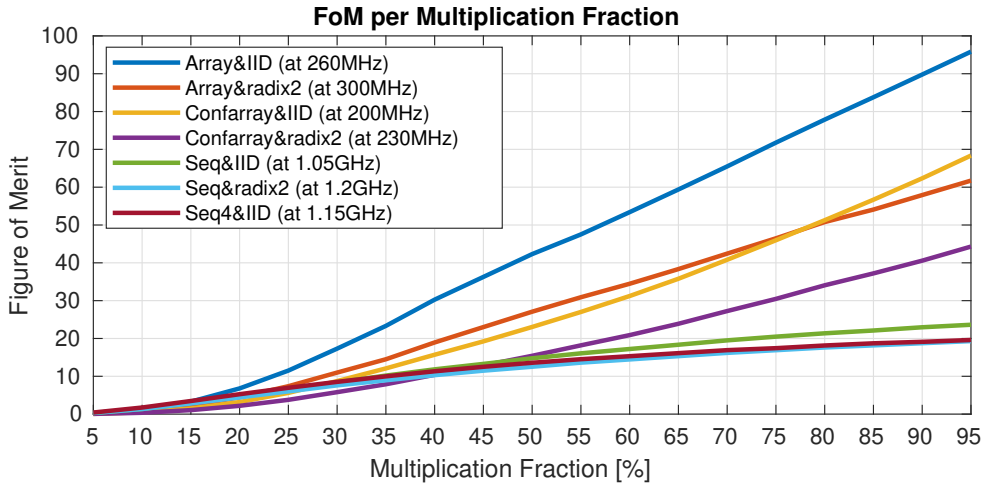


Figure 4.6: The Figure of Merit of each architecture-level design at different fractions of the workload involving multiplication

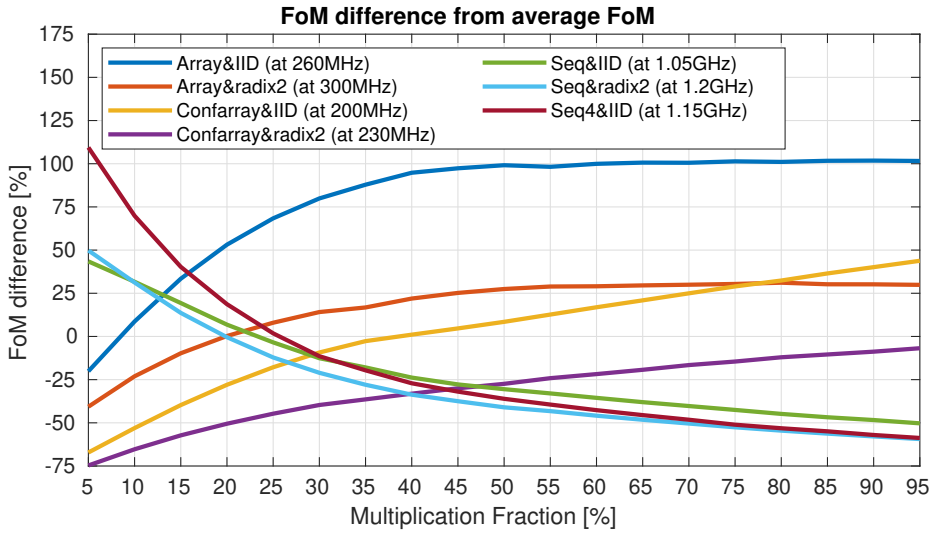


Figure 4.7: Each architecture-level design at different fractions of the workload involving multiplication. The y-axis shows the percentage by which each design's FoM is better or worse than the average of all 7 designs at that particular multiplication fraction

barely occurs. Between 5 and 50% multiplications, this trend between parallel and sequential designs is the cause of the largest relative shift that occurs between designs.

On the right half of the plot, a different trend appears. The sequential and array multiplier designs remain mostly at the same FoM percentage, while the configurable precision designs rise. The configurable precision and IID design even overtakes the non-configurable and radix-2 design. This shows that the viability of precision configurable designs also increases when the fraction of multiplications in de workload increases (besides the reasons mentioned in conclusion subsection 4.1.3). This makes sense, yet again this has to do with the amount of hardware resources dedicated to multiplication. The Sub-Word Parallel array multiplier adds area in the form of muxes that control what signals enter the array. This additional area provides lower delay, especially when a large portion of the workload contains multiplications.

Finally, and most relevant for the conclusion, the array multiplier and IID design obtains the highest FoM among the designs anywhere above 20% multiplication in the workload. In fact, between 47.5% multiplications, the Basic Workload, and 80.02% multiplications, the total Workload, the Array&IID design has a particularly large margin over the other designs.

4.2.3. Workload Variation Conclusions

The goal of the workload variation experiment was to verify whether the non-precision-configurable parallel multiplier with IID is the best architecture-level design for the Basic workload as well as the Kernel workload. The results clearly show that this is indeed the case. The non-configurable parallel multiplier and IID design even obtains its largest margin over the other designs within the relevant range of multiplication fractions. Therefore, it can be concluded that a design with a non-precision-configurable parallel multiplier and IID is the best architecture-level design for an area and energy-efficient arithmetic unit for an SML model.

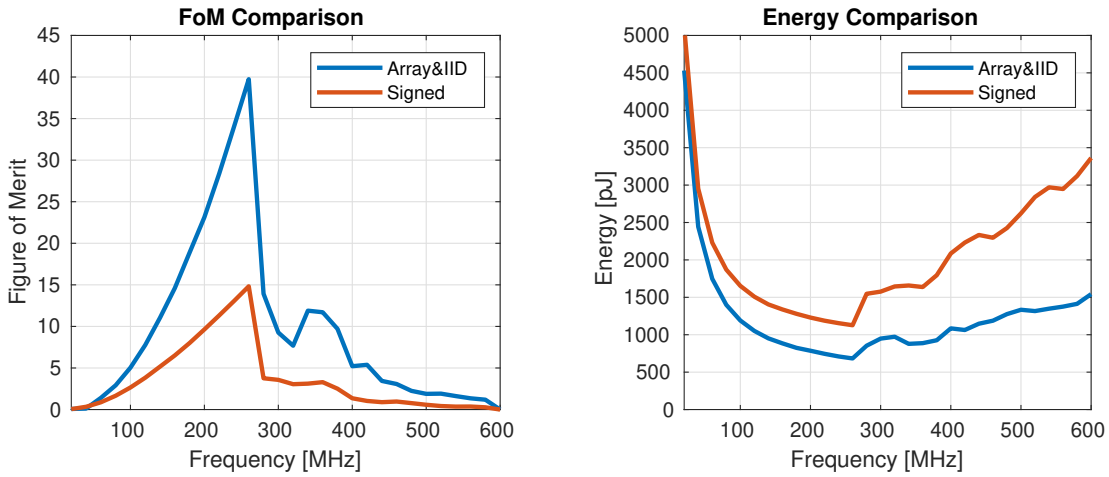
4.3. Futher Design Improvement Results

4.3.1. Setup

For this design phase the setup is the same as for the architecture-level design phase. The designs are worked out in Verilog and tested in Cadence Xcelium, then their activity when performing the Basic Workload is extracted from Xcelium and used in Genus to obtain the area, energy, delay and slack data. This is once again done over a range of clock frequencies.

4.3.2. Signed Multiplication

Figure 4.8 shows the results of implementing signed multiplication in the architecture-level design, array&IID. As expected, the impact of signed multiplication on the performance lies entirely in the energy consumption. The delay is the same and the area shows an insignificant difference. Only the energy consumption changes for the worse. At the optimal clock frequency, the energy consumption of the unsigned design is 683 pJ. In comparison, the signed design consumes 1127 pJ. That is a 1.65 times increase in energy consumption, which can entirely account for the FoM difference, as the FoM formula factors energy twice.



(a) A Figure of Merit comparison between the best architecture-level design and its signed version

(b) The energy consumption of the signed multiplier design compared to the best architecture-level design

Figure 4.8: The results of the signed multiplier design

As explained in methodology subsection on the configurable signed multiplier 3.4.2, the increase in energy consumption within the signed multiplier is caused by the sign extension of otherwise small values that in an unsigned notation would contain many zeros, which would not cause any activity in the higher significance ends of the multiplier array. This hypothesis is supported by the observation in the energy plot of figure 4.8 that the difference in energy consumption is much smaller at lower clock frequencies. This means that the different in energy consumption is a result of higher dynamic power in the signed multiplier and not a result of higher static power as well.

In conclusion, signed multiplication costs a lot of extra energy, and hurts the FoM as a result. With such higher energy consumption is reasonable to consider special ways of decreasing this energy consumption. For this reason, as mentioned in the methodology section, the next design optimization explores the possibility of employing configurability in the signed multiplier.

4.3.3. Configurable Signed Multiplication

The results of the configurable signed multiplier design are displayed in figures 4.9 and 4.10. The three lines shown in each of the figures are labeled Signed, Confsigned2 and Confsigned1. Signed refers to the signed multiplier of the previous subsection, Confsigned2 refers to the most optimal configurable signed multiplier achieved and Confsigned1 is one of the previous attempts.

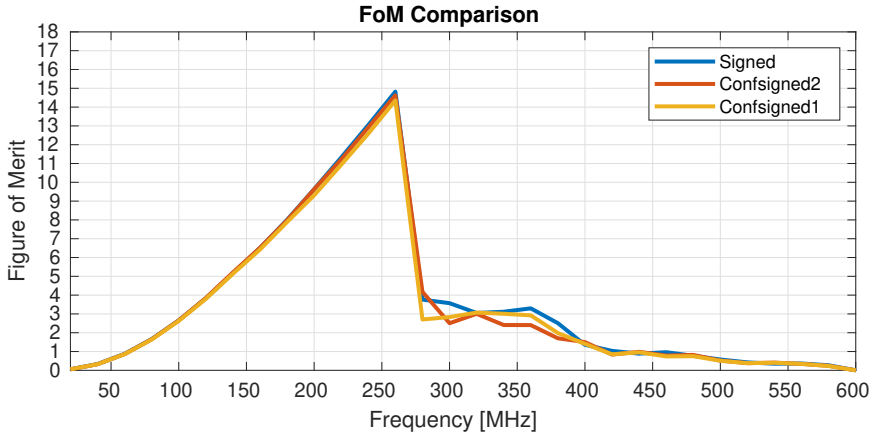
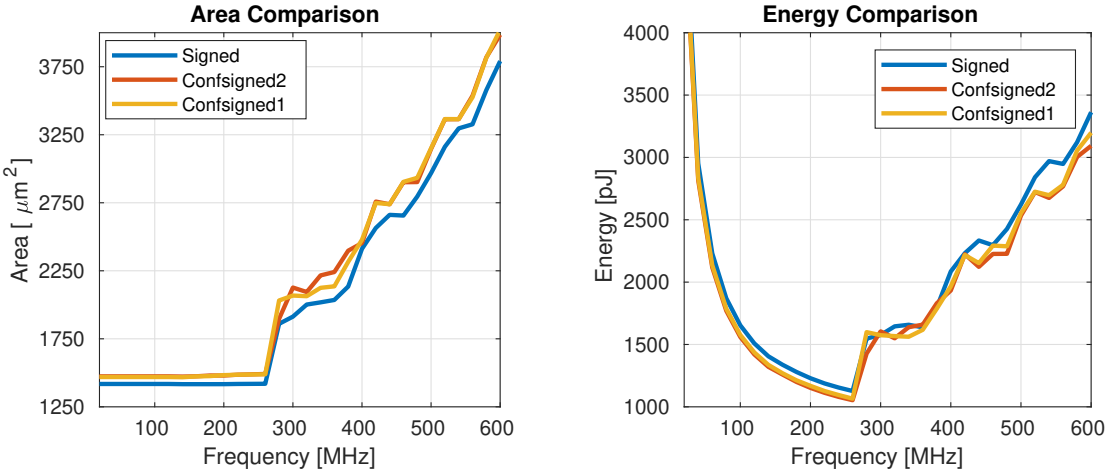


Figure 4.9: A Figure of Merit comparison between the non-configurable signed multiplier and two different implementations of configurable signed multipliers



(a) Area comparison

(b) Energy comparison

Figure 4.10: The area and energy figures of the configurable signed multplier compared to their non-configurable counterpart

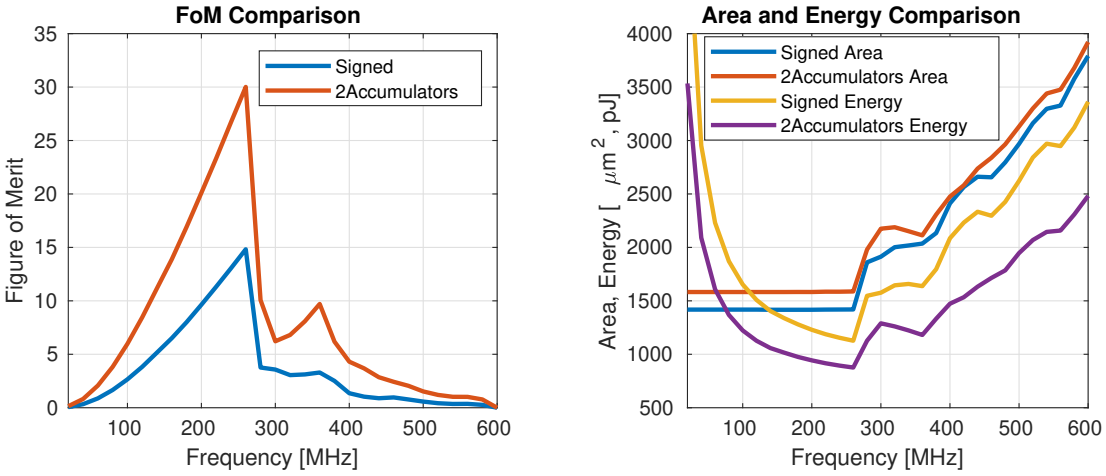
From the FoM plot, it quickly becomes apparent that the configurable signed multiplier does not provide the improved performance expected. The FoMs of all three designs are almost identical. Figure 4.10 provides some insight. The energy consumption of the configurable designs is lower, despite the larger circuit area. However, the importance of the area negates this benefit. The delay is the same for all three design, as the number of cycles to complete the workload, nor the optimal clock frequency is different among the designs.

It must be concluded that the configurable signed array multiplier does not provide the benefit to the SML model's arithmetic unit. However, the unexpectedly small changes in area and energy indicate that the configurable signed array multiplier is likely not implemented as intended and more time should be invested into it to achieve a stronger conclusion.

4.3.4. Dedicated Accumulator

Figure 4.11 shows the results of the dedicated accumulator design. Figure 4.11a compares the FoM of the new dedicated accumulator design, named 2Accumulators, with the signed multiplier design. To

the right, figure 4.11b shows the area and energy consumption of the dedicated accumulator design compared to that of the signed multiplier design.



(a) FoM comparison

(b) Area and energy comparison

Figure 4.11: The results from a signed version of the dedicated accumulator design, labeled with 2Accumulators. The results are compared with the signed array&IID design

From these results, the following observations can be made. Firstly, the FoM of the dedicated accumulator design is higher than that of the signed multiplier design across the entire frequency spectrum. It is also interesting to point out that the optimal frequency of the dedicated accumulator design remains the same at 260MHz. Despite the not-insignificant redesign of the way accumulation and division occur, this is to be expected. The extra dedicated accumulator is not likely to increase the length of the longest critical path. Data from the new accumulation register can be sent to the multiplier for IID, but this datapath is very similar to how division is performed in the signed multiplier design. Meanwhile, the delay of the accumulation that occurs in the new 16-bit accumulator is easily less than that of the 35-bit accumulator.

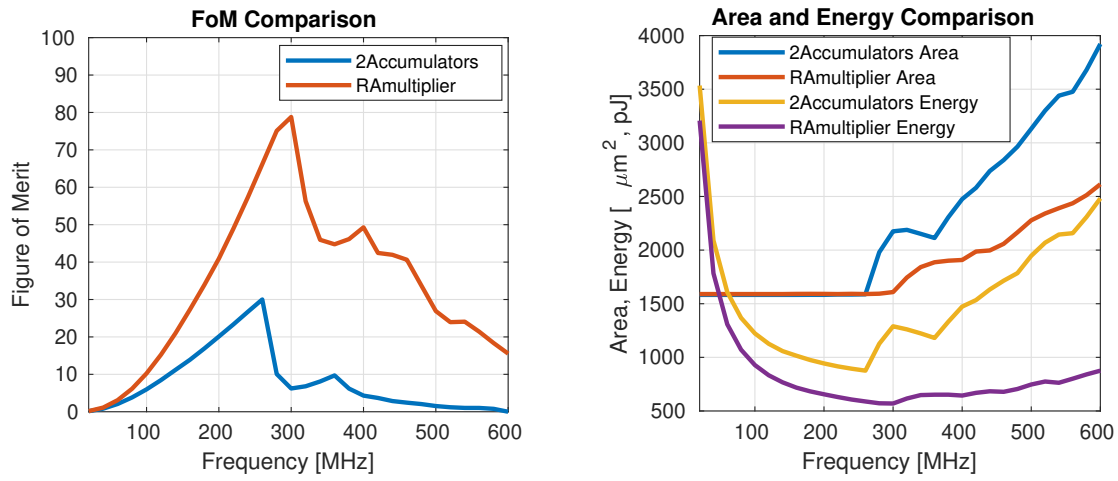
A lot of the FoM improvement comes from the decrease in the amount of clock cycles it takes to complete the workload. Instead of 483 cycles, the workload now takes 282 cycles. This was the target of the design and accounts for most of the FoM improvement. However, figure 4.11b also shows that the dedicated accumulator design saves energy. Although energy reduction was expected, the limited increase in power consumption by the dedicated accumulator helped the FoM reach unexpected heights.

In conclusion, a dedicated accumulator is a very worthwhile addition to any area and energy efficient arithmetic unit. Although the advantages in energy and delay are very dependent on the fraction of accumulation operations in the workload, the area trade-off is very small.

4.3.5. Reduced Area Multiplier

The results of the Reduced Area (RA) multiplier design are plotted in figure 4.12. Figure 4.12a compares the FoM of the RA multiplier design with that of the till thus far best-performing dedicated accumulator design. The RA multiplier was integrated into the dedicated accumulator design, meaning the RA multiplier design also contains two accumulators. This means that the FoM difference between the two lines is entirely the result of replacing the signed array multiplier with a signed RA multiplier. Figure 4.12b shows the area and energy performances of the two designs. The delay of the two designs at any particular frequency is the same, as replacing the array multiplier with an RA multiplier has not changed the number of cycles it takes to complete the basic workload.

The first observation to make about these results is that the RA multiplier provides a surprisingly large performance improvement. The largest advantage of the RA multiplier over the array multiplier was expected to be the much shorter critical path of in the RA multiplier. As mentioned in subsection 3.4.4, the longest path through the RA multiplier is only 8 FA delays and 22 HA delays, while the path through the array multiplier is 16 FA delays and 14 HA delays. Quantitatively, the expected reduction in delay would be around 20%. With a frequency increase from 260 to 300 MHz (the x-axis indices of



(a) FoM comparison

(b) Area and energy comparison

Figure 4.12: The results from a signed version of the RA multiplier design. The results are compared with the dedicated accumulator design, labeled 2Accumulators. The RA multiplier design also contains the dedicated accumulator.

the FoM peaks), that reduction in delay has revealed itself to be true. However, the reduction in energy consumption has had a much larger impact. The inefficiency of the array multiplier regarding energy was known, but with a 35% energy decrease, the RA multiplier is mostly better than the array multiplier because of energy.

Another unexpected feature of the RA multiplier is that there is much less of an abrupt jump in area and energy at a particular frequency step. Until now, all the parallel multiplier designs have had a very specific optimal frequency caused by a several hundred μm^2 increase in the area once the longest critical path of the low-frequency ”base design” becomes the long of the clock period for the first time. In figure 4.12b this occurs to the 2Accumulators design between 260 and 300 MHz. With these new results, it becomes clear that this phenomenon was for the most part feature of the array multiplier. The RA multiplier design still shows a small jump in area just above 300MHz, its optimal frequency, but much less than the array multiplier. The effect of this is that the RA multiplier design does relatively well at frequencies above 300MHz. This means that if the importance of low delay increases due to changes in the applications domain, the RA multiplier design can easily be synthesised at higher frequencies and compete with the sequential designs, unlike the array multiplier designs.

To conclude, the RA multiplier has proven to be a much better parallel multiplier for the SML model’s arithmetic unit, achieving figures of Merit three times higher than the array multiplier. This is caused primarily by an improved energy efficiency, but also a decrease in delay. Finally, the RA multiplier is also more versatile in terms of its clock frequency, being able to efficiently use additional area to operate at higher clock frequencies.

5 Discussion

This chapter starts by summarising all the findings that are relevant to the research questions, and then compares them as best as possible to any state of the art or related works. The next section considers the implications of the findings. Finally, the chapter ends with a critical review of the limitations of the study from a perspective of the solutions explored and the robustness of the experimental setup, as well as giving recommendations both related and unrelated to these limitations.

5.1. Findings

The architecture-level design, workload variation experiment and further design improvements in this thesis were performed to design an area and energy-efficient arithmetic unit for an SML model. The results of these design efforts lead to the following findings.

- Parallel multipliers are better suited for the multiplication heavy workloads of the SML model due to lower energy consumption than sequential multipliers.
- Invariant Integer Division, when feasible to implement, is a more area and energy-efficient way to perform division than sequential dividers.
- Precision configurable multipliers do not provide sufficient reduction in power and delay to warrant the additional area and path length.
- Sequential multipliers are feasible if delay is a more important KPI than energy consumption.
- Signed multiplication is much more energy-consuming than unsigned multiplication and this consumption is not easily reduced through configurability.
- The basic workload of the SML model is greatly and efficiently accelerated through the inclusion of an extra dedicated accumulator.
- RA multipliers are much more energy efficient than array multipliers.
- RA multipliers are more easily synthesised for higher clock frequencies leading to reduced delay.

5.2. State of the Art Comparison

A true state of the art for stacked ML model hardware does not exist. So, a simple comparison between designs is not possible. Instead, in this section, the performance of the area and energy efficient arithmetic unit designed will be evaluated in two steps. First, this thesis' findings in regards to the configurable multipliers, the RA multiplier and Invariant Integer are compared to those in literature. Second, the final area, energy and delay figures of the arithmetic unit are compared to those of the RISC-V core, a more generate purpose platform that was introduced in related works section 2.2.3.

5.2.1. Quantitative Comparions

Judging whether or not these findings correspond to those found in literature is hard to do quantitatively. This is because all of the arithmetic architectures used are inspired by older literature using older technology nodes.

The closest to a quantitative comparison is possible for the sub-word parallel array multiplier. Figure 2.2 in the related works section shows it compared to an array multiplier with data gating. It is possible to combine the energy and throughput-per-area figures from the three plots into one in an 8:15:15 ratio to account for the precision distribution used for the experimental setup of this thesis. This way, it is fairly accurate to say that according to [46] the sub-word parallel array multiplier should achieve 0.446pJ per operation versus 0.45pJ for the normal array multiplier. The throughput for its part is $6.47GOPS/mm^2$ versus $6.447GOPS/mm^2$. Those are all very similar numbers, indicating that the literature says the configurable array multiplier in this thesis should perform similarly to the non-configurable array multiplier. This is not observed. Throughput per area was especially worse at all frequencies for the configurable array multiplier. The difference in the KPIs observed could be due to some imperfection in the Verilog implementation of the array multiplier that causes the strange jump in area and energy past a certain clock frequency. Another factor causing worse results in this thesis could be due to the setup of the Figure of Merit. This Figure of Merit more heavily penalizes area compared to delay. This is not done for the throughput-per-area metric in figure 2.2.

5.2.2. Qualitative Comparisons

Qualitatively, literature has shown before that sequential multipliers are not as energy efficient as parallel multipliers, agreeing with the findings of this thesis.

Literature shows really good performance for configurable signed multipliers [57], disagreeing with our findings. This could be caused by inefficient implementation on the side of this thesis, but the difference in technology node between [57] and this thesis (180nm and 28nm respectively) is likely also a factor. 28nm CMOS technology experiences more leakage power compared to active power than 180nm technology [58]. This reduces the benefit of preventing activity in the signed array multiplier using configurability. Another factor that makes it hard to accurately compare results is the precision distribution, which is a metric that is not clearly stated by [57].

The last qualitatively comparable finding is about the RA multiplier. Here literature claimed better area, energy and delay [44]. Although our results don't agree with this area claim, this was already expected by counting the number of full and half adders required for the multiplier. Significant improvement in energy and delay was indeed observed as seen in literature.

5.2.3. No Comparable Literature

Very Hard to compare are the following two findings. Firstly, there was no Application Specific Integrated Circuit (ASIC) implementation of IID found in literature. This makes the IID results presented in this thesis unique. However, the results do adhere to what would be expected based on the description of the IID architecture, less area and probably less energy as well.

A similar situation is true for the dedicated accumulator. This circuit addition is conceptually so simple that it is not the subject of any literature. However, based on the description of the component and the workload it is supposed to run, it is naturally expected that the delay should decrease drastically and energy reduction can also be expected. All in exchange for minor increase in area.

5.2.4. Comparison to the RISC-V core

The energy-efficient RISC-V core for IoT applications presented by [54] provides the best comparable hardware platform to compare with the arithmetic unit designed in this thesis. With some estimation, covered in section 2.2.3 of related works, the core has an area of $4432\mu m^2$, a power of $11.77mW$ and a throughput of $3.325GOPS$. Using these figures the energy and delay of the RISC-V core when performing the Basic Workload can be calculated. Table 5.1 shows the resulting comparison.

It is important to take this as merely a 0^{th} order comparison. The figures of the RISC-V core are the result of two estimations, first for the correct technology node, and then for the correct bit-width. On top of that, the area and energy figures of the RISC-V core would be slightly lower if they did not include the control circuitry, like the arithmetic unit of this thesis. However, the comparison does indicate the merits of arithmetic unit designed in this thesis. The arithmetic unit is indeed more area and energy efficient than this existing solution in the form of a RISC-V Core.

Table 5.1: Performance comparison of the RISC-V Core and this thesis' arithmetic unit in executing the Basic Workload

	RISC-V Core [54]	This Work
Area	4432 μm^2	1609 μm^2
Energy	1179 pJ	569 pJ
Delay	100 ns	940 ns

5.3. Implications

The combined findings of this thesis indicate what types of arithmetic to consider or not to consider during hardware design for an SML model. A well-optimised area and energy efficient arithmetic unit is provided for area-constrained edge applications for which a similar SML model is designed. However, the impact of changes, in the application domain and in the workload of the SML model, are considered regularly to show how this thesis' findings can provide insight into different design scenarios.

This thesis also reveals a situation in which Invariant Integer Division is an invaluable solution to the problem of division and shows how to implement it into an ASIC containing a MAC unit.

Finally, it serves as a clear example of why array multipliers are rarely the best multiplier design, and a more optimized multiplier should always eventually be chosen. The RA multiplier was the final improvement proposal of the further design improvement phase, which means that a shorter design phase could have resulted in this multiplier improvement never being implemented. This is so notable, because it caused the largest single increase in Figure of Merit observed in the results.

5.4. Limitations and recommendations

Although this thesis provides an in-depth comparison of the performance of a set of sufficiently well-selected hardware solutions to support a well-defined ML workload, there are still some solutions to explore, experiments to improve and problems to solve.

5.4.1. More Solutions to Explore

The first limitation of this thesis lies in the severely limited assortment of multipliers considered within the design phases. The multiplication heavy workload of the SML model means multipliers have the largest impact on the Figure of Merit. This thesis could not perform a comparison between many multipliers as much of the focus was also directed toward the divisions and accumulations of the workload. However, what is provided is some clear findings about the choice of multipliers, in the better performance of parallel multipliers. It is also reasonable to argue that the RA multiplier is very well suited for the task. However, multiplier design is where future research can still provide energy and delay improvement. As mentioned in related works, Booth Encoding is a very promising technology to look into.

5.4.2. Experiments to Improve

Another opportunity of this thesis research is the simplification of the SML model's workload. This has two aspects. First, in the definition of the basic and kernel workloads, the order and timing of the operations regarding the larger, yet to be designed, SML hardware was disregarded. This was reasonable due to the SML model's potential to require redesign when new training data becomes available and the fact that some characteristics were not yet well defined. The other aspects of simplification concerns the further design improvement phase. This design phase was solely performed with the basic workload, not taking into account how relative performance between designs may shift when the kernel workload is included. Especially the FoM of the dedicated accumulator design will change with more multiplication centred workloads. Due to these two aspects of simplification, it is recommended to rerun the relevant experiments with more specific workloads once the specifics of the SML model are more defined in the future.

5.4.3. More SML Hardware Problems to Solve

Lastly, there is one more recommendation that is less of a product of a limitation. This thesis targets the area and energy efficient design of an arithmetic unit. It was defined in the hardware requirements section 2.1.5 that this thesis would only concern itself with the kernel calculations, MAC operations and averaging that occurs in the SML model. However, it is possible to integrate the Nearest Neighbors

calculator into the arithmetic unit. This was only not chosen because specialised hardware already exists for it. Whether these hardware solutions are better than integration with the arithmetic unit was not proven in this thesis. Also, the decision tree solver will be a large part of any SML model hardware system and there are many solutions for decision tree solving that all have their merits. Thirdly, there is also a necessity for optimised data storage for the SML model. Extensive research is still required into these parts of the SML model before a proof of concept for an embedded automotive lifetime prediction unit can fully take shape.

6 Conclusion

This chapter starts by restating the research aims and questions. Then, based on the analysis in the previous results and discussion chapters, answer the research questions. Finally, the key contributions of this study are summarised and the study is briefly reflected upon.

6.1. Research Aims

This thesis' research aimed to design part of a hardware platform for a specific stacked machine learning model and to gather knowledge about the process of hardware design for SML models such that the findings in this thesis can be generalised. Through identification of common denominators in the SML model's hardware requirements, the most interesting aspect of the hardware was identified within the MAC, multiply, divide and accumulate operations of the model. As a result the aim was placed on an area and energy efficient arithmetic unit for these operations and around it the research questions were defined.

6.2. Research Questions

This thesis attempted to answer the following two research questions and sub-questions:

1. How do you integrate the arithmetic operations of multiple classical ML models into one area and energy efficient arithmetic unit?
    1.1. How can multiple models efficiently share one unit for their basic MAC operations?
    1.2. Can division be efficiently integrated with a MAC unit?
    1.3. Can kernel calculations be efficiently performed on a MAC unit?
2. How do you maintain the benefits of reduced bit-precision ML models in hardware that executes models with various levels of bit-precision?

The answers found to these questions are as follows:

1. Identify shared arithmetic operations among the ML models. Dedicate most of the hardware area to an area and energy efficient parallel multiplier, like an RA multiplier. Support the likely small quantity of divisions through IID, if possible, because it benefits from the good multiplier already implemented. Lastly, provide a dedicated accumulator if there are many accumulations in the workload to prevent low multiplier utilization.
    1.1. Select a parallel multiplier that is known for low area and energy, like an RA multiplier.
    1.2. Yes, in the SML model, Invariant Integer Division provides support for division with very little area overhead and with more energy efficiency than true division implemented separately from the multiplier.
    1.3. Yes, the multiplication heavy workload of kernel calculations benefits from the same emphasis on energy efficient ,and relatively speedy, multiplication by parallel multipliers as the MAC operations.

2. Under the expected precision distribution of the SML model multiple bit precisions are best supported by not dedicating any additional resources on configurable hardware.

6.3. Contributions

By compiling the findings from the results and the subsequently answered research questions it is possible to present the most important contribution of this thesis. Figure 6.1 shows a high-level overview of the area and energy efficient arithmetic unit for SML models designed for this thesis. It contains the parallel multiplier, specifically the RA multiplier, the dedicated accumulator and the support for invariant integer division. Each of these parts also feature in the answers to the research questions. They were the best performing hardware solutions in terms of area and energy efficiency found for the arithmetic requirements of the SML model. This schematic forms the groundwork for the realisation of a proof of concept for automotive lifetime prediction in hardware.

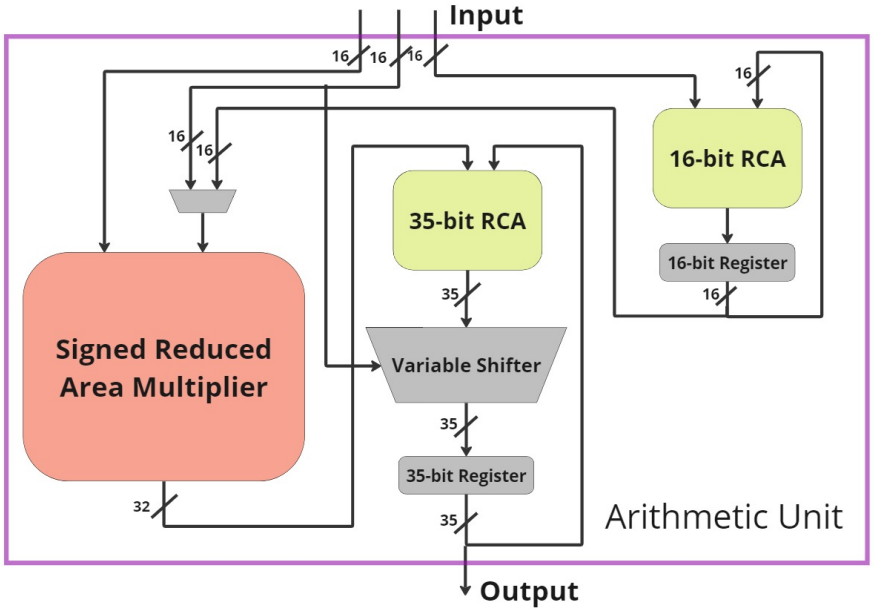


Figure 6.1: A schematic of the SML model's area and energy efficient arithmetic unit.

The next contribution is to the field of hardware design for machine learning models. There is a complete lack of research into hardware specifically for stacked ML models. Therefore, this thesis can function as a start to a body of knowledge about design problems specific to SML models. These are issues like identifying common denominators in the hardware requirement of the base- and meta learners of stacked ML models and in combining the MAC heavy workloads of most machine learning models with support for accumulation and division necessary for other base- and meta learners that may appear in a stacked ML model.

Finally, the results of this study contain area and energy figures of the ASIC implementation of various computer arithmetic designs, notably Invariant Integer Division, in the relatively up to date 28nm CMOS technology node. Especially the IID figures can be a useful contribution to any researcher considering the merits of IID as the literature found for this thesis only discusses the mathematical possibility and implementation using standard processor instruction sets.

6.4. Reflection

This thesis aimed to create hardware for an embedded automotive lifetime prediction system bases on a stacked machine learning model. An exploration of the hardware requirement of the model identified the need of an area and energy efficient arithmetic unit that could support MAC, multiply, divide and accumulate operations.

The design of this arithmetic unit was performed in three phases. The architecture-level design

phase compared solutions to three critical design aspects, the workload variation experiment verified the previous results for an expanded workload, and the further design improvement phase found two more improvements on the results of the architecture-level design.

As a result, this thesis produced well supported answers to its research questions, a first of a kind example of hardware design for stacked ML models and a design for an area and energy efficient arithmetic unit.

References

[1] E. Landman, N. Brousard, and T. Naishlos, "A novel approach to in-field, in-mission reliability monitoring based on deep data," in *2020 IEEE International Reliability Physics Symposium (IRPS)*, 2020, pp. 1–8. DOI: 10.1109/IRPS45951.2020.9128846.

[2] M. B. T. Arunkumar Vijayan Krishnendu Chakrabarty, "Machine learning-based aging analysis," in Springer, 2019, ch. 9.

[3] M. Sewak, S. K. Sahay, and H. Rathore, "Comparison of deep learning and the classical machine learning algorithm for the malware detection," in *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2018, pp. 293–296. DOI: 10.1109/SNPD.2018.8441123.

[4] M. Kang, S. K. Gonugondla, S. Lim, and N. R. Shanbhag, "A 19.4-nj/decision, 364-k decisions/s, in-memory random forest multi-class inference accelerator," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 7, pp. 2126–2135, 2018. DOI: 10.1109/JSSC.2018.2822703.

[5] H. Elhosary, M. H. Zakhari, M. A. Elgammal, M. A. Abd El Ghany, K. N. Salama, and H. Mostafa, "Low-power hardware implementation of a support vector machine training and classification for neural seizure detection," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1324–1337, 2019. DOI: 10.1109/TBCAS.2019.2947044.

[6] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80023-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005800231.

[7] K. du Buf, "Model stacking performance comparisons for lifetime estimation of cmos ics," Available at https://repository.tudelft.nl/islandora/object/uuid:886fa02c-809d-4435-a6c4-2280a961892e?collection=education, M.S. thesis, Delft Univerity of Technology, Jun. 2023.

[8] K. Khalil, O. Eldash, A. Kumar, and M. Bayoumi, "Machine learning-based approach for hardware faults prediction," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 11, pp. 3880–3892, 2020. DOI: 10.1109/TCSI.2020.3010743.

[9] A. Chigurupati, R. Thibaux, and N. Lassar, "Predicting hardware failure using machine learning," in *2016 Annual Reliability and Maintainability Symposium (RAMS)*, 2016, pp. 1–6. DOI: 10.1109/RAMS.2016.7448033.

[10] S. Shreejith, B. Anshuman, and S. A. Fahmy, "Accelerated artificial neural networks on fpga for fault detection in automotive systems," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 37–42.

[11] J. Delua, *Supervised vs. unsupervised learning: What's the difference?* Available at https://www.ibm.com/blog/supervised-vs-unsupervised-learning/ (26/12/2023).

[12] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008. DOI: 10.1214/009053607000000677.

[13] A. Radhakrishnan, *Lecture 3: Kernel regression*, https://web.mit.edu/modernml/course/lectures/MLClassLecture3.pdf, Accessed: 16/11/2023, Jan. 2022.

[14] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A primer on kernel methods," *Kernel methods in computational biology*, vol. 47, pp. 35–70, 2004.

[15] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[16] L. Breiman, "Stacked regressions," *Machine Learning*, vol. 24, pp. 49–64, 1996. DOI: https://doi.org/10.1007/BF00117832.

[17] C. Zhang and Y. Ma, *Ensemble Machine Learning*. New York: Springer, 2012. DOI: https://doi.org/10.1007/978-1-4419-9326-7.

[18] D. Poornaiah, R. Haribabu, and M. Ahmad, “Design and vlsi implementation of a novel concurrent 16-bit multiplier-accumulator for dsp applications,” in *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, 1993, 385–388 vol.1. DOI: 10.1109/ICASSP.1993.319136.

[19] F. Elguibaly, “A fast parallel multiplier-accumulator using the modified booth algorithm,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 9, pp. 902–908, 2000. DOI: 10.1109/82.868458.

[20] V. Bartlett and E. Grass, “A low-power concurrent multiplier-accumulator using conditional evaluation,” in *ICECS’99. Proceedings of ICECS ’99. 6th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.99EX357)*, vol. 2, 1999, 629–633 vol.2. DOI: 10.1109/ICECS.1999.813186.

[21] S. Smith, R. DeMara, J. Yuan, M. Hagedorn, and D. Ferguson, “Null convention multiply and accumulate unit with conditional rounding, scaling, and saturation,” *Journal of Systems Architecture*, vol. 47, no. 12, pp. 977–998, 2002, ISSN: 1383-7621. DOI: https://doi.org/10.1016/S1383-7621(02)00060-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762102000607.

[22] A. Abdelgawad and M. Bayoumi, “High speed and area-efficient multiply accumulate (mac) unit for digital signal prossing applications,” in *2007 IEEE International Symposium on Circuits and Systems*, 2007, pp. 3199–3202. DOI: 10.1109/ISCAS.2007.378152.

[23] T. T. Hoang, M. Själander, and P. Larsson-Edefors, “High-speed, energy-efficient 2-cycle multiply-accumulate architecture,” in *2009 IEEE International SOC Conference (SOCC)*, 2009, pp. 119–122. DOI: 10.1109/SOCCON.2009.5398079.

[24] T. Yang, T. Sato, and T. Ukezono, “An approximate multiply-accumulate unit with low power and reduced area,” in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 385–390. DOI: 10.1109/ISVLSI.2019.00076.

[25] M. Esmali Nojehdeh, L. Aksoy, and M. Altun, “Efficient hardware implementation of artificial neural networks using approximate multiply-accumulate blocks,” in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 96–101. DOI: 10.1109/ISVLSI49217.2020.00027.

[26] M. Qasaimeh, A. Sagahyroon, and T. Shanableh, “Fpga-based parallel hardware architecture for real-time image classification,” *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp. 56–70, 2015. DOI: 10.1109/TCI.2015.2424077.

[27] N. P. Thanh, Y.-S. Kung, S.-C. Chen, and H.-H. Chou, “Digital hardware implementation of a radial basis function neural network,” *Computers Electrical Engineering*, vol. 53, pp. 106–121, 2016, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2015.11.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790615003924.

[28] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, “Hardware implementation of the exponential function using taylor series,” in *2014 NORCHIP*, 2014, pp. 1–4. DOI: 10.1109/NORCHIP.2014.7004740.

[29] S.-I. Chu, J.-H. Yan, T.-H. Chien, B.-H. Liu, C.-Y. Lien, and X.-R. Yu, “Hardware implementation of multi-kernel function for support vector machine classification using stochastic logic,” in *2022 IEEE 11th Global Conference on Consumer Electronics (GCCE)*, 2022, pp. 812–813. DOI: 10.1109/GCCE56475.2022.10014235.

[30] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011. DOI: 10.1109/TPAMI.2010.57.

[31] I. Stamoulias and E. S. Manolakos, “Parallel architectures for the knn classifier – design of soft ip cores and fpga implementations,” *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, Sep. 2013, ISSN: 1539-9087. DOI: 10.1145/2514641.2514649. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2514641.2514649.

[32] H. Kaul, M. A. Anders, S. K. Mathew, *et al.*, “14.4 a 21.5m-query-vectors/s 3.37nj/vector reconfigurable k-nearest-neighbor accelerator with adaptive precision in 14nm tri-gate cmos,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 260–261. DOI: 10.1109/ISSCC.2016.7418006.

[33] J. Saikia, S. Yin, Z. Jiang, M. Seok, and J.-s. Seo, “K-nearest neighbor hardware accelerator using in-memory computing sram,” in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6. DOI: 10.1109/ISLPED.2019.8824822.

[34] G. Pedretti, C. E. Graves, S. Serbryakov, *et al.*, “Tree-based machine learning performed in-memory with memristive analog cam,” *Nature Communications*, vol. 12, no. 5806, 2021. DOI: 10.1038/s41467-021-25873-0.

[35] X. Yin, F. Müller, A. F. Laguna, *et al.*, “Deep random forest with ferroelectric analog content addressable memory,” *arXiv preprint arXiv:2110.02495*, 2021.

[36] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *International conference on machine learning*, PMLR, 2015, pp. 1737–1746.

[37] R. Struharik, “Decision tree ensemble hardware accelerators for embedded applications,” in *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*, 2015, pp. 101–106. DOI: 10.1109/SISY.2015.7325359.

[38] H. Nakahara, A. Jinguji, S. Sato, and T. Sasao, “A random forest using a multi-valued decision diagram on an fpga,” in *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, 2017, pp. 266–271. DOI: 10.1109/ISMVL.2017.40.

[39] X. Lin, R. S. Blanton, and D. E. Thomas, “Random forest architectures on fpga for multiple applications,” in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 415–418.

[40] F. Daghero, A. Burrello, C. Xie, *et al.*, “Adaptive random forests for energy-efficient inference on microcontrollers,” in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2021, pp. 1–6. DOI: 10.1109/VLSI-SoC53125.2021.9606986.

[41] S. Amanollahi and G. Jaberipur, “Fast energy efficient radix-16 sequential multiplier,” *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 73–76, 2017. DOI: 10.1109/LES.2017.2714259.

[42] M. Mottaghi-Dastjerdi, A. Afzali-Kusha, and M. Pedram, “Bz-fad: A low-power low-area multiplier based on shift-and-add architecture,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 2, pp. 302–306, 2009. DOI: 10.1109/TVLSI.2008.2004544.

[43] S. Immareddy and A. Sundaramoorthy, “A survey paper on design and implementation of multiplier for digital system applications,” *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4575–4603, 2022. DOI: 10.1007/s10462-021-10113-0.

[44] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, *Parallel reduced area multipliers*, 1995. DOI: 10.1007/BF02407084.

[45] C. De Sa, M. Leszczynski, J. Zhang, *et al.*, “High-accuracy low-precision training,” *arXiv preprint arXiv:1803.03383*, 2018.

[46] V. Camus, L. Mei, C. Enz, and M. Verhelst, “Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697–711, 2019. DOI: 10.1109/JETCAS.2019.2950386.

[47] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, “Dvafs: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 488–493. DOI: 10.23919/DATE.2017.7927038.

[48] L. Mei, M. Dandekar, D. Rodopoulos, *et al.*, “Sub-word parallel precision-scalable mac engines for efficient embedded dnn inference,” in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019, pp. 6–10. DOI: 10.1109/AICAS.2019.8771481.

[49] J. H. P. Zurawski and J. B. Gosling, “Design of a hih-speed square root multiply and divide unit,” *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 13–23, 1987. DOI: 10.1109/TC.1987.5009445.

[50] M. Ercegovac and T. Lang, “Implementation of module combining multiplication, division, and square root,” in *IEEE International Symposium on Circuits and Systems,*, 1989, 150–153 vol.1. DOI: 10.1109/ISCAS.1989.100314.

[51] J. E. Stine, A. Phadke, and S. Tike, “A recursive-divide architecture for multiplication and division,” in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, 2011, pp. 1179–1182. DOI: 10.1109/ISCAS.2011.5937779.

[52] T. Granlund and P. L. Montgomery, “Division by invariant integers using multiplication,” *SIGPLAN Not.*, vol. 29, no. 6, pp. 61–72, Jun. 1994, ISSN: 0362-1340. DOI: 10.1145/773473.178249. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/773473.178249.

[53] T. Granlund and P. L. Montgomery, “Division by invariant integers using multiplication,” in *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, ser. PLDI ’94, Orlando, Florida, USA: Association for Computing Machinery, 1994, pp. 61–72, ISBN: 089791662X. DOI: 10.1145/178243.178249. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/178243.178249.

[54] M. Gautschi, P. D. Schiavone, A. Traber, *et al.*, “Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017. DOI: 10.1109/TVLSI.2017.2654506.

[55] Wikipedia contributors, *Division algorithm — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Division_algorithm&oldid=1183246511, [Online; accessed 11-December-2023], 2023.

[56] M. Sjalander and P. Larsson-Edefors, “Multiplication acceleration through twin precision,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 9, pp. 1233–1246, 2009. DOI: 10.1109/TVLSI.2008.2002107.

[57] Z. Huang and M. Ercegovac, “Two-dimensional signal gating for low-power array multiplier design,” in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 1, 2002, pp. I–I. DOI: 10.1109/ISCAS.2002.1009884.

[58] T. Iizuka, “Cmos technology scaling and its implications,” in *Digitally-Assisted Analog and Analog-Assisted Digital IC Design*, Cambridge University Press, 2015, pp. 1–10.