Design Space Exploration for Softmax Implementations

Zhigang Wei*, Aman Arora†, Pragenesh Patel‡, Lizy John§
The Laboratory for Computer Architecture, Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, United States
*zw5259@utexas.edu, †aman.kbm@utexas.edu, ‡f20160183@goa.bits-pilani.ac.in, §ljohn@ece.utexas.edu

***Abstract*—Deep Neural Networks (DNN) are crucial components of machine learning in the big data era. Significant effort has been put into the hardware acceleration of convolution and fully-connected layers of neural networks, while not too much attention has been put on the Softmax layer. Softmax is used in terminal classification layers in networks like ResNet, and is also used in intermediate layers in networks like the Transformer. As the speed for other DNN layers keeps improving, efficient and flexible designs for Softmax are required. With the existence of several ways to implement Softmax in hardware, we evaluate various softmax hardware designs and the trade-offs between them. In order to make the design space exploration more efficient, we also develop a parameterized generator which can produce softmax designs by varying multiple aspects of a base architecture. The aspects or knobs are parallelism, accuracy, storage and precision. The goal of the generator is to enable evaluation of tradeoffs between area, delay, power and accuracy in the architecture of a softmax unit. We simulate and synthesize the generated designs and present results comparing them with the existing state-of-the-art. Our exploration reveals that the design with parallelism of 16 can provide the best area-delay product among designs with parallelism ranging from 1 to 32. It is also observed that look-up table based approximate LOG and EXP units can be used to yield almost the same accuracy as the full LOG and EXP units, while providing area and energy benefits. Additionally, providing local registers for intermediate values is seen to provide energy savings.**

***Index Terms*—Softmax, DNN, Machine LearningDesign Space Exploration**

I. Introduction

Deep Neural Networks (DNN) have become one of the most important technologies for machine learning. There has been a rapid development of hardware for accelerating inference or training process of DNNs. While most architectures focus on speeding up the convolution and fully-connected layers, there are only a few researchers who have proposed optimizations in hardware for softmax layer, which serves as a key component in DNNs. Therefore, more research is required to explore efficient architectures for softmax.

Softmax is usually used for multi-category classification as the last layer in neural networks like ResNet or MobileNet. It is also used as an activation layer in intermediate layers in some networks, for example in Transformer and Capsule network.

The major challenge of the softmax hardware is to implement efficient exponential units and division units. The naive implementation is not very hardware friendly because it easily causes overflow, requires large amounts of storage, and includes divider and exponential units which are generally costly. Some researchers have proposed architectures for softmax [6] [4] [8] [5] [12]. However, most of these designs can only support a fixed number of inputs and the hardware required increases proportional to the number of inputs, and they generally support only one precision. Hence, these designs are not flexible. The focus of many prior designs is on providing efficient implementations of the exponent unit, e.g. LUT based [6] or FSM based [5]. Geng et al. [4] uses bit-shifts for division. The design in [12] is not pipelined. Li et al. [8] uses FIFOs to store all input values increasing the area significantly. Not all designs support fixed point and floating point data types, limiting their application to either training or inference.

Although existing designs may perform well with one particular accuracy or parallelism in one scenario, the performance may not remain when architects want to tune the design. Additionally, tuning the existing hardware design may be time-consuming and requires lots of extra work. There are several limitations in the existing softmax hardware designs:

- The support for different parallelism values is poor, which makes the performance of their designs not scale well with increasing input data sizes.
- They do not support various precisions which limits their design to machine learning training or inference.
- Designs may consume large area while the trade-offs between area and accuracy is not clear.

There exist signification trade-offs in the aspects mentioned above. Different DNNs have different number of inputs for the softmax layer. Different accelerators have different budgets for area and delay of softmax layer. Different applications have different tolerance for classification accuracy. A one-size-fits-all softmax architecture can not satisfy all the requirements in a space with such diversity. Adhoc methods of exploration can leave out efficient architectures leading to inefficient accelerators. So, we believe a tunable generator that can generate multiple designs with different architectures can be very valuable to perform design space exploration. To the best of our knowledge, no such tool exists in the open source community. Our contributions in this paper are summarized

2160-052X/20/$31.00 ©2020 IEEE
DOI 10.1109/ASAP49362.2020.00017

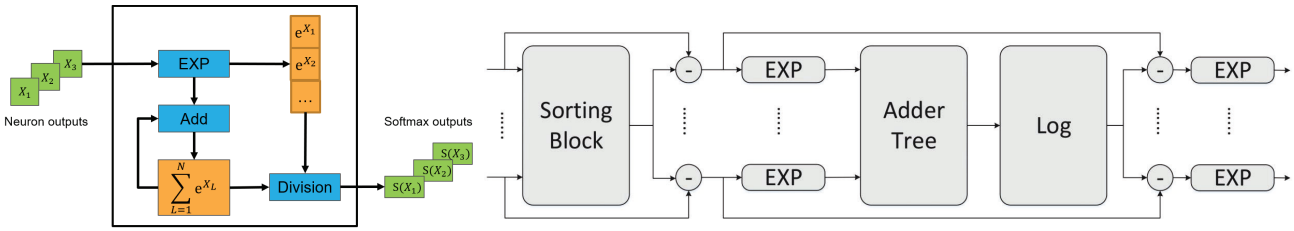

Fig. 1: (a) Naive softmax architecture (as shown in [4] (on the left) (b) softmax architecture proposed in [13] with the required EXP units (on the right)

as follows:

- We propose a base architecture that is amenable to adjustment based on various parameters such as parallelism, accuracy and precision. This architecture can support any number of input values and can work for the forward pass of softmax for both inference and training.
- We develope a generator called SoftGen that generates softmax designs. The generator is a software to generate softmax verilog code. This generator is controlled by various knobs - parallelism, accuracy, storage, precision - that can take multiple values. Based on the values of the knob, the generator dumps a design and a testbench.
- We perform design space exploration using our developed generator and proposed base architecture. We evaluate the various generated softmax designs and discuss the trade-offs between area, delay, power and accuracy.

Our design space exploration reveals observations from three aspects. For the parallelism ranging from 1 to 32, the energy-delay product of the design decreases with the increasing parallelism until the parallelism reaches 16. It is also observed that the approximate LUT-based LOG and EXP units yield almost the same accuracy but are more energy and area efficient compared to the full accurate implementations. Additionally, local registers used to store the intermediate results can reduce the number of memory accesses, therefore, provide energy savings, however, extra area overhead is required.

The rest of paper is organized as follows: Section II gives the backgound information of the softmax hardware design. In Section III, we describe the details of the base architecture used by our generator. Section IV introduces how we automate the designs generation and tools used for evaluation.V presents the various exploration experiments we conducted along with the results observed from these experiments. Section VI concludes this paper and points out the future work.

II. Background

The formula to calculate the M-th neuron in a softmax layer is described as below:

$$P(Mth\ category) = \frac{e^{X_M}}{\sum_{L=1}^{N} e^{X_L}} \tag{1}$$

where X_L is the output of the L-th neuron and N is the number of categories. The most straightforward way to implement it is shown in Figure 1(a). However, several problems exist in such a design: first of all, this kind of design requires significant amount of storage to store the exponential results because the number of classification outputs can be in thousands or millions. Secondly, such a design includes expensive division units. Normal division unit can consume large area and requires significant amounts of time to execute which increases the power consumption and cause difficulties to the further pipeline the design. Thirdly, it cannot leverage the parallelism existing in softmax calculation, therefore, it performs poorly when the number of inputs increases.

Researchers have tried several approaches to tackle the above problems, Kouretas et al. [6] uses LUT to approximate the exponential calculation. Hu et al. [5] leverage the stochastic computing to conduct the softmax execution.

Yuan [13] introduced an efficient hardware architecture for softmax layer as shown in Figure 1(b). Equation 1 is adapted into a more hardware friendly form:

$$P(Mth\ category) = e^{(X_M - X_{max}) - \ln(\sum_{L=1}^{N} e^{(X_L - X_{max})})} \tag{2}$$

This avoids large silicon area consumption and accuracy loss caused by division units, and down-scaling technique is applied to exponential units to overcome the potential overflow problem. However, several problems remain. For DNNs with large number categories, this design will require large number of exponential units, which is not realistic. Additionally, the architecture modifies the meaning of outputs; it generates the magnitude of each classification rather than the probability (the last exponential stage is missing). Lastly, no quantitative evaluation of the design is provided in the paper.

Du et al. [3] optimized Yuan's architecture [13] to process the data serially so that it can perform classification of infinite categories. But the total cycles to accomplish the softmax operation increases exponentially with the number of input values. FIFOs are utilized to store intermediate data. The depth of these FIFOs increases proportionally to the input size indicating a large area requirement. The authors take advantage of the distribution of inputs in softmax layers to avoid some calculations for input values that are out of range, but that does not work in all cases, e.g. training.

We create our base architecture based on Yuan's design due to its scalability and pipelining features.

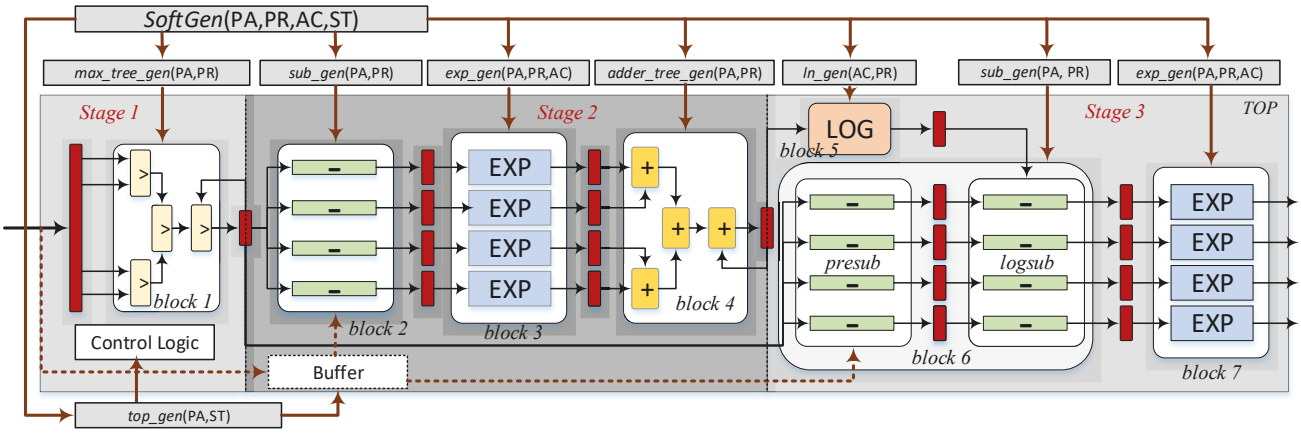

Fig. 2: Baseline Architecture used for design exploration. Softmax generator allows knobs: PA: Parallelism, PR: Precision, AC: Accuracy, ST: Storage. The diagram shows the hardware with PA=4. The value of ST controls the presence of buffer and dotted path.

III. Architecture

Our baseline architecture allows the generator Softgen to create various designs based on the knobs. This architecture is shown in Fig. 2. The architecture is logically divided into 3 stages and physically divided into 7 blocks:

- **Stage 1:** This stage includes block 1 (max). It finds the largest value from all the input values. X_{max}
- **Stage 2:** This stage includes blocks 2, 3, 4.
  - Block 2 (subtraction) finds the difference between each input value and the max value. $X_L - X_{max}$
  - Block 3 (exponent) generates the exponential of the results from Block 2. $e^{(X_L - X_{max})}$
  - Block 4 (adder tree) adds all the exponential values up. $\sum_{L=1}^{N} e^{(X_L - X_{max})}$
- **Stage 3:** This stage includes blocks 5, 6, 7.
  - Block 5 (log) calculates the natural logarithm of the result from Block 4. $\ln(\sum_{L=1}^{N} e^{(X_L - X_{max})})$. Let's call this $XLOG$.
  - Block 6 is composed of two sets of subtractors (called presub and logsub) to calculate $X_M - X_{max} - XLOG$
  - Block 7 calculates the final result. $e^{X_M - X_{max} - XLOG}$

Stage 2 can only be triggered once the max value is found by *Stage 1*. *Stage 3* can be triggered only when *Stage 2* is finished (i.e. the Adder tree has finished adding all values). The timeline for a design generated by SoftGen can be seen in Fig. 3. Within each stage, operations are pipelined to reduce the latency significantly. In other words, blocks within a stage start before the previous block is finished (e.g. ADD in stage 2 starts before SUB in stage 2 is finished). There are latches after each block in the design. Some blocks like the adder tree, max block, exponential unit are pipelined internally as well.

Note that the number of inputs for softmax operation can be different than the amount of parallelism in the design. For example, a design could have a parallelism of 4 (4 values read and processed together in the design, 4 subtractors in block 2, 4 exponential units in block 3, etc.), but still process a tensor

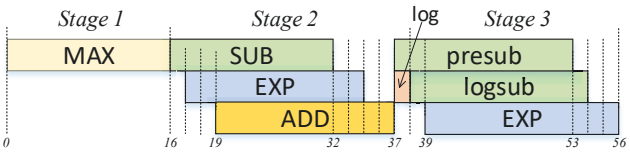

Fig. 3: Timeline for the architecture (Parallelism=4, Number of input values=64, Storage=mem)

with, say, 512 input values. In these cases, the control unit orchestrates the data movement such that all 512 values are processed in groups, with 4 values entering the design at a time.

The following sections provide details of how the designs of various blocks in the architecture are modified to enable the knobs of the generator:

A. Max block (block 1) and Adder tree (block 4)

The Parallelism and Precision knobs affect the architecture of the Max block. Based on the precision, floating point or fixed point comparators are instantiated in this block. In a fully serial implementation, the Max block only requires 1 comparator. For Parallelism $>= 2$, the generated Max block is composed of a comparator tree. The number of levels of comparators in the Max block = $log_2(N) + 1$, where N is the value of the Parallelism knob. The $+1$ is required to handle the cases where the number of inputs values is larger than the parallelism of the design. For the Parallelism=4 and input values=512 case mentioned above, the max value from 4 input values is stored in a buffer and is compared with the max value from the next 4 values by this additional comparator. The comparator tree is pipelined. Based on the delays of the various blocks in the library we used [10] (more details can be found in section IV) we add pipeline registers after every 3 comparator levels. The Adder tree is similar to the comparator tree, except that we add pipeline registers in this tree after every adder.

B. Subtractors (block 2 and block 6)

The value of the Parallelism knob governs the number of subtractors needed in these blocks. The type of the subtractors (floating point or fixed point) depends on the value of the Precision knob. Block 6 is divided into two parts: logsub and presub. For each input, the value $X_L - X_{max}$ is calculated by block 2 and is required again by block 6 logsub (to calculate $X_M - X_{max} - XLOG$). In [3], the authors save the temporary values $X_L - X_{max}$ in FIFOs in the design. In our architecture, we add additional subtractors (block 6 presub) to calculate the difference again. This saves significant area (for FIFOs), but adds 1 cycle of latency and requires additional subtractor(s).

C. Exponential units (block 3 and block 7)

There are multiple ways for designing hardware to compute exponent [14][4][12][5][8][3]. In our generator, we provide the Accuracy knob to choose between two implementations of the exponential unit. The first one is the exponential unit provided by the DesignWare library ([10]). We provide a second reduced-area, low-accuracy option that uses LUT-based Piecewise Linear Function (PLF) approach from [4]. The architecture utilizing PLF technique for 16-bit floating point EXP unit is shown in Fig. 4. A user may also choose a fixed-point data format using the Precision knob. For that, LUT-based fixed-point EXP units have been implemented as well that follow a similar architecture with LUTs storing fixed point values and excludes the fixed to floating point converter.

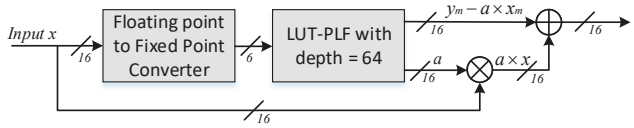

Fig. 4: Architecture of the float16 EXP unit used by the generator

PLF is generally used to approximate non-linear functions with a small number of linear pieces [2]. PLF technique approximates the computation of e^x by using the linear equation in N continuous intervals uniformly defined over a finite range of $x \in [x_m^1, x_p^N]$, with each interval having a slope a^n.

$$f^n(x) = a^n \times (x - x_m^n) + y_m^n = a^n \times x + (y_m^n - a^n \times x_m^n) \quad (3)$$

$$\text{where } x \in [x_m^n, x_p^n],\ y_m^n = e^{x_m^n},\ n \in [1, N]$$

Following the implementation in [3], input data in the range of [-8, 0] is considered as valid and data less than -8 is mapped to the last entry in the Look up Table (LUT-PLF). There are two reasons for this. 1) Values input to EXP unit will always be either zero or negative because the maximum input value is subtracted from each input in block 2. 2) Since $e^0/e^{-8} \approx 2980.958$ and $e^{-8} = 0.000335$, it is deemed safe to ignore this small value.

For the 16-bit floating point exponential unit, the LUT-PLF is built to store the 16-bit floating point value of the slope a^n and the pre-computed $(y_m^n - a^n \times x_1^n)$ for 64 equally divided intervals in the data range of [-8, 0]. An input x is converted from floating point to fixed point format that is used to select the PLF-LUT entry closest to x. That is further followed by a multiplication $(a^n \times x)$ and an addition to compute $f^n(x)$ as per equation 3.

D. Natural logarithm unit (block 5)

There are multiple ways for designing hardware to compute natural log [12][3][11]. For the natural logarithm (LOG) unit, we provide the Accuracy knob to choose between two implementations from our generator. The first one is the LOG unit provided by the DesignWare library ([10]). We also provide a second reduced-area, low-accuracy option that follows the ICISLog algorithm mentioned in [11]. For the 16-bit floating point LOG unit, we use a modified LUT-based architecture of the ICISLog algorithm implementation in [1]. A user may also choose a fixed-point data format using the Precision knob. For that, fixed-point LOG units have been implemented as well that follow a similar architecture with LUTs storing fixed point values and includes a floating to fixed point converter. Since real valued logarithm is only defined for positive numbers, a positive floating point number can be represented as:

$$val = 2^{exp} \times (1.mantissa)$$

Using the multiplicative property of the logarithm function, we get:

$$ln(val) = ln(2) \times exp + ln(1.mantissa) \quad (4)$$

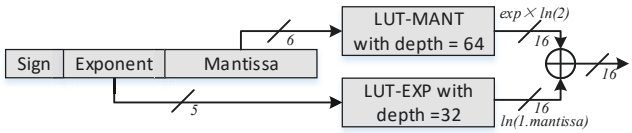

Fig. 5: Architecture of the float16 LOG unit used by the generator

In Fig. 5 we provide the block diagram of the LOG unit based on Eq. 4. All the exponent bits and the first 6 mantissa bits are used to select the $ln(2) \times exp$ and $ln(1.mantissa)$ 16-bit floating point values from look up tables LUT-EXP and LUT-MANT respectively. The outputs from the look up tables are added to obtain the final output $ln(x)$.

E. Comparison with existing architectures

Table I compares various attributes of our architecture with existing designs. Our architecture overcomes many limitations that are present in other architectures and through the generator, we provide exploration of various attributes to allow a DNN hardware architect to make informed decisions within the constraints of an application.

IV. Experimental methodology

In this section, we discuss the tools we used to conduct the experiment. The flow to conduct the experiment can be summarized in the following steps:

Feature	Ours	[3]	[13]	[6]	[4]	[12]	[5]	[8]
Support for any number of input values	Y	Y	N	N	Y	N	N	Y
Hardware increases proportional to input size	G	N	Y	Y	N	Y	Y	N
Needs costly/accurate division unit	N	N	N	N	N	N	N	Y
Uses LOG based modified softmax formula	Y	Y	Y	N	N	Y	Y	N
Uses LUT based EXP or LOG units	G	Y	Y	Y	Y	N	Y	Y
Uses internal storage to store input values for reuse	G	Y	N	N	Y	N	N	Y
Supports fixed and floating point values	G	N	N	N	N	Y	N	N
Is completely serial and hence has high latency	G	Y	N	N	Y	N	N	Y
Is completely parallel and hence has high area	G	N	Y	Y	N	N	Y	N
Redoes subtraction instead of storing temp results	Y	N	N	N	N	N	N	N
Down scaling for EXP (”max - val”)	Y	Y	Y	Y	N	Y	N	N
Adder tree used for additions	Y	N	Y	N	N	Y	N	N
Uses stochastic computing methods	N	N	N	N	N	N	Y	N
Applicable to both training and inference	Y	N	Y	N	N	Y	N	N

TABLE I: Comparing the features of various softmax architectures (Y=Yes, N=No, G=Provided through generator for trade-off analysis)

- prepare the blocks of basic arithmetic
- synthesize, simulate and verify the blocks
- use the generator SoftGen to generate softmax designs
- synthesize, simulate and verify the softmax models

The circuit designs of the first step are already mentioned in III. We used Synopsys tools for synthesis. All synthesis is performed under 45nm technology with FreePDK45 academic library [9]. The area values in our results are post-synthesis and pre-placement/pre-routing areas. We used CACTI [7] to analyze the energy consumption of memory accesses. A single port on-chip memory is assumed to contain the input values required by softmax. Each memory location is wide enough to store the input values required in one memory read, based on the parallelism knob. We also assume that read/write latency for read/write from the on-chip memory is 1 clock.

A. SoftGen

Figure 6 provides an overview of the flow and architecture of the generator. The inputs to the generator are values of various knobs that control different aspects of the softmax architecture described in Section III. The outputs of the generator are a set of Verilog design files including module definitions of each block and the top-level module. The top-level module puts all the blocks together, along with the control logic. The generator also produces a simple testbench that can be used to verify the sanity of the design. The Makefile available with the generator dumps the design and the testbench, compiles and simulates the code, and generates a CSV file that lists the observed output values from the softmax Verilog design, the expected output values from a Python based CPU model and the difference between the two.

The generation is composed of two components: Verilog templates and python scripts. The Verilog templates contains the skeleton design and testbench corresponding to our architecture with various tags present in it at various locations to customize the design. The Python scripts process the template, replace the tags with Verilog code based on the knobs specified when running the generator and dump Verilog files during the process.

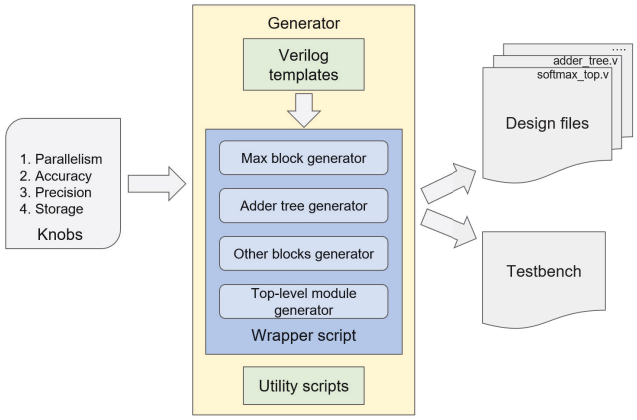

Fig. 6: Flow and architecture of the softmax generator

The Python scripts are organized hierarchically to make the generator modular and easily changeable. There are separate generator scripts for the adder tree and the max block. The utility scripts generate inputs for the simulation, expected outputs and the CSV containing the difference.

B. Design spaces of softmax

With the knob support of the SoftGen, we explore the hardware implementation trade-offs of softmax in 4 different aspects:

1) **Parallelism:** This knob controls the amount of parallelism in the generated design. Currently, this knob can take a value of any power of 2 (including 2^0=1). A value of 1 implies a fully serial design. Such a design has

one compute unit in each block, and consumes the least amount of area. But it takes the most amount of clock cycles. As the value of this knob increases, the design's parallelism increases. That means more compute blocks are added, increasing the area and power consumption. As an example, a value of 4 will generate a design which has more area but smaller latency. This knob is useful to study the trade-off between area, power and delay.

2) **Accuracy:** This knob controls which EXP and LOG implementations are used in the softmax design. All the blocks in the design, except the EXP and LOG blocks, have full accuracy. For EXP and LOG blocks, we support choosing between a highly accurate implementation from the Synopsys DesignWare [10] library, or a less accurate implementation using LUTs (as described in the sections III-C and III-D). The LUT based implementations are more area efficient. This knob can be used to study the trade-off between accuracy of results and the area of the design.
3) **Precision:** This knob controls the precision (data type) for all the compute units used by the design. We currently support 4 data types: int8, int32, float16, float32. This knob is driven by system requirements. For example, it has been shown that for inference, int8 is sufficient, but float16 is more optimal for training. This knob mainly changes the compute blocks in the design. The control logic remains the same. So, the area of the design and the clock frequency is affected by this knob, but not the latency in clock cycles.
4) **Storage:** As can be seen from the architecture described in Section III, input values stored in the on-chip memory are required 3 times during the softmax operation - for calculating the max value, for calculating difference of inputs from the max value and for finally calculating the probabilities. These values can either be read from the on-chip memory whenever required (consuming SRAM access delay and energy every time), or they could be read once from the on-chip memory and stored in registers internal to the softmax unit (consuming area and static power) and used directly. This knob is used to select between these two choices (NOREG or REG), to study the trade-off between delay, area, energy and power. Internal storage is used by the design in [3].

C. Implementation of the baseline design

We chose the design from Du et al. [3] as our baseline since their design gives the best implementation of [13] to our knowledge. However, we can not directly compare our designs with the results their papers because they use a different technology node (65 nm) and a different design library. We instead use their architecture and our design blocks and library to estimate various metrics for their design. An approximation of the baseline design can be generated by our generator with the settings: Parallelism=1, Accuracy=LUT, Precision=fixed32, Storage=REG, except for one main difference. The authors of [3] take advantage of the distribution of inputs in softmax layers to avoid some calculations for input values that are out of range, but that limits their circuit's use for training. Instead we support the full range of input values.

V. Results and Discussion

In this section, we discuss the observations from the experiments when we sweep the configurations of parallelism, accuracy and storage in the first three subsections. We also compared our generated designs with the state-of-the-art architecture and the discussion is in the last subsection.

A. Exploration with the Parallelism knob

For this experiment, we varied the values of the Parallelism knob across 1,2,4,8,16,32. The other knobs were kept fixed (Accuracy=LUT, Storage=mem, Precision=float16). The number of input values used in this experiment was fixed at 1024. The normalized post-synthesis area and the number of cycles consumed by each generated design are plotted in Figure 7. Also plotted is the area-delay product. As expected, with increasing parallelism, the number of cycles reduces, but the area increases. We see the area delay product value reduces and then starts to increase, implying the design with Parallelism=16 is the best. However, designs with Parallelism values of 8, 16 have very similar values of area-delay product and hence are good choices. For larger values of Parallelism, the power consumption of the design can also be expected to increase, because more compute units are working in parallel.

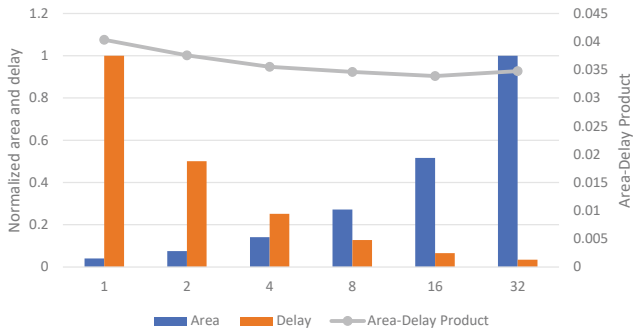

Fig. 7: Trade-off between area and number of cycles with varying values of the Parallelism knob (1024 input values, Accuracy=LUT, Storage=mem, Precision=float16))

B. Exploration with the Accuracy knob

To see the effect of the Accuracy knob, we generated two designs - one with LUT based implementations of EXP and LOG blocks, and another with DesignWare [10] implementations of these blocks. Other knobs were kept fixed (Parallelism=4, Storage=NOREG, Precision=float16). We chose various ranges of inputs and generated random values in those ranges and fed them to the two designs. We then compared the results against the results obtained from a simple Python based CPU model. Table II shows the comparison. LUT based implementations are less accurate, but this is generally acceptable for DNNs.

Range	Max error with DW	Max error with LUT	Avg error with DW	Avg error with LUT
-0.1 to 0.1	8.80E-06	5.04E-05	7.21E-06	3.55E-05
-1 to 1	2.40E-06	2.90E-04	5.31E-07	8.38E-05
-10 to 5	5.70E-06	4.31E-03	3.11E-07	1.69E-03
5 to 10	1.22E-03	1.23E-03	2.45E-04	4.93E-04
-8 to -4	5.70E-06	7.60E-04	6.69E-07	2.29E-04
-8 to 8	3.77E-03	4.65E-03	2.45E-04	2.05E-03

TABLE II: Accuracy evaluation for DesignWare and LUT-based implementations (512 input values, Parallelism = 8, Storage=NOREG, Precision=float16) LUT based implementations are less accurate but generally still acceptable for DNNs

Table III shows the variation of area and delay of the whole softmax design with these two Accuracy options. We can see from the first two rows of the table that the LUT based design has a smaller area, but delay is higher with the design using DesignWare blocks because the DesignWare blocks are not pipelined (our LUT based EXP unit has a pipeline stage in it) and so the design could only run at a reduced clock frequency. Since they are available as IP blocks, we could not modify them. We also synthesized the design using LUTs at the max frequency at which the design using DesignWare could be synthesized. The area reduced significantly with this optimization and the power reduced as well.

Design	Cycles	Delay (*us*)	Power (*mW*)	Energy (*nJ*)	Area (um^2)
Design with LUT, max freq (294MHz)	201	0.67	10.19	6.82	279711
Design with DW, max freq (250MHz)	199	0.79	8.38	6.67	283300
Design with LUT, iso freq (250MHz)	201	0.80	6.87	5.52	220178

TABLE III: Trade-off between various metrics with different values of the Accuracy knob (512 input values, Parallelism = 8, Storage=NOREG, Precision=float16). Power/Energy numbers are from Synopsys Design Vision.

C. Exploration with the Storage knob

There are two values of the Storage knob - NOREG and REG - as described in Section IV. For this experiment, we fix the Parallelism knob to 4, Accuracy knob to LUT and Precision knob to float16. We vary the number of inputs from 32 to 1024, and generate two designs for each case - one that re-reads inputs from on-chip memory whenever required and another that stores the input values in registers after reading them once. The resulting chart is shown in Figure 8. Registers cause the area of the design to increase significantly with increasing number of input values. But for the design with on-chip memory re-reads, the area does not change as we increase number of input values. We calculated energy consumed by the additional registers in the design with Storage=REG and the energy consumed by additional on-chip memory re-reads in the design with Storage=NOREG. The energy consumed for re-reading inputs from on-chip memory is higher than the energy consumed for reading/writing to/from internal storage registers. Since the on-chip memory re-read latency can be hidden behind other operations, the delay for the design with on-chip memory re-reads is not different from the delay for the design with registers. For the design with registers, the number of registers required to store inputs is equal to the number of input values, and so the design becomes less flexible. So, there is a tradeoff between area, energy and flexibility.

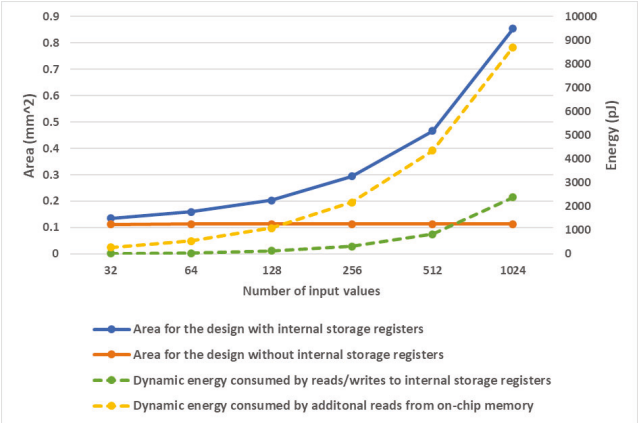

Fig. 8: Area and energy evaluation with different values of the Storage knob with various number of input values (Parallelism = 4, Accuracy=LUT, Precision=float16)

D. Comparison with the state-of-the-art

Table IV compares various metrics of the design from [3] with some variations of the designs generated by our generator. The "Add. energy" column refers to the additional energy consumed because of internal storage registers in the designs with Storage=REG, and the additional energy consumed because of memory re-reads in the designs with Storage=NOREG. We can see that a design with Parallelism=1, Storage=NOREG (second row in the table) is much more area efficient, but consumes more energy. Changing Parallelism=2 and Storage=NOREG (fourth row) results in a faster design, but with more area consumption.

Design	Area (mm^2)	Cycles	Add. energy (*pJ*)
Design in [3]	0.807	1542	830.24
Design with PA=1, ST=NOREG	0.059	1542	4351.48
Design with PA=2, ST=REG	0.828	775	830.24
Design with PA=2, ST=NOREG	0.085	775	4351.48
Design with PA=4, ST=REG	0.835	392	830.24
Design with PA=4, ST=NOREG	0.138	392	4351.48

TABLE IV: Comparing various metrics for some designs generated by the generator with the design in [3]. PA=Parallelism, ST=Storage, PR=Precision, AC=Accuracy. All designs were synthesized for a clock frequency of 250 MHz, processed 512 input values, have the same precision (fixed32) and have the same accuracy (LUT).

One of the important issues mentioned in [3] is that in their design, as the number of input values increases, the total

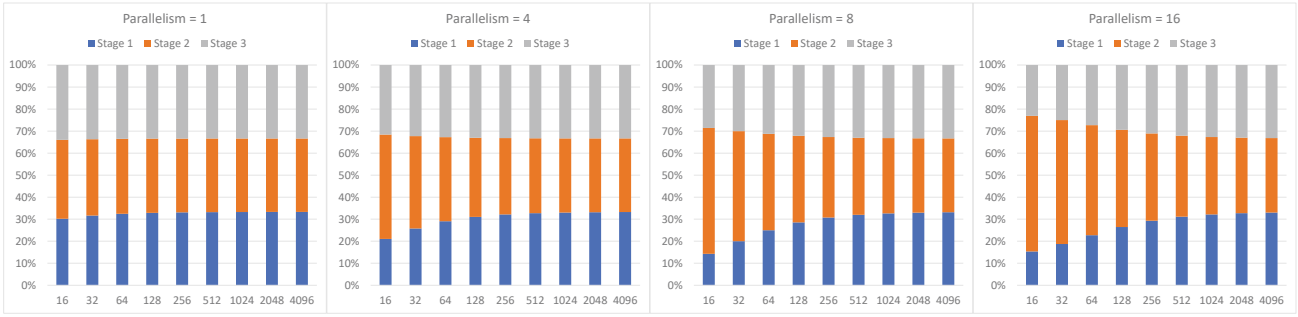

Fig. 9: Cycle consumption in each stage of the generated design with various values of the Parallelism knob (x-axis: number of input values, y-axis: percentage of cycles consumed) Stages defined in Section III-E.

computing time increases exponentially, and the time taken by the Max block dominates the total computing time because of the poor scalability of sorting logic in the Max block. Figure 9 shows the results from a similar study we conducted using various designs generated by our generator. In this case, the other knobs were Storage=NOREG, Accuracy=LUT, Precision=float32. We can see that these designs are easily pipelineable to handle multiple data sets during Training since we can keep each stage busy at the same time. For larger input sizes, the designs are very balanced. We spend almost equal time in each stage. For smaller input sizes, stage 2 does consume relatively more time especially with high values of Parallelism, but these scenarios are not very common.

VI. Conclusion

There are many tradeoffs in the design of softmax, the multi-category classification layer in neural networks. In this paper, we perform design tradeoff evaluation of softmax using SoftGen, an open-source tool[1] that we created that generates softmax designs by controlling the values of parallelism, accuracy, precision and storage. The architecture used by our generator eliminates the shortcomings in existing designs such as limited parallelism, limited precision options, etc. We show the results of trade-off analysis using these knobs in the paper. In terms of parallelism, it is found that the architecture with parallelism of 16 can provide the best area-delay product among all the parallelism ranging from 1 to 32. It is also observed that LUT-based EXP and LOG units can help to make the design more energy and area efficient with almost the same accuracy. Additionally, providing local registers to store the intermediate results are seen to yield energy savings.

This work can be extended in many ways. Currently, we only support input sizes that are a power-of-2 (including $2^0 = 1$). We plan to add support for other knobs and other values of the existing knobs. While variations of LOG and EXP units, and bfloat16 or other precision settings can be added to the framework, this paper presents several important insights on softmax designs and demonstrates a methodology for parameterizable design generation and design space exploration of softmax.

[1]The tool is available at https://github.com/georgewzg95/softmax

VII. Acknowledgement

We thank all the anonymous reviewers for the detailed comments on the paper. This work was supported in part by the National Science Foundation grant 1763848. Any opinions, findings, conclusions, or recommendations are those of the authors and do not necessarily reflect the views of these funding agencies.

References

[1] N. Alachiotis and A. Stamatakis, "Efficient floating-point logarithm unit for fpgas," 05 2010, pp. 1 – 8.

[2] H. Amin, K. M. Curtis, and B. R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," *IEE Proceedings - Circuits, Devices and Systems*, vol. 144, no. 6, pp. 313–317, Dec 1997.

[3] G. Du, C. Tian, Z. Li, D. Zhang, Y. Yin, and Y. Ouyang, "Efficient softmax hardware architecture for deep neural networks," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: ACM, 2019, pp. 75–80. [Online]. Available: http://doi.acm.org/10.1145/3299874.3317988

[4] X. Geng, J. Lin, B. Zhao, A. Kong, M. M. S. Aly, and V. Chandrasekhar, "Hardware-aware softmax approximation for deep neural networks," in *Computer Vision – ACCV 2018*, C. Jawahar, H. Li, G. Mori, and K. Schindler, Eds. Cham: Springer International Publishing, 2019, pp. 107–122.

[5] R. Hu, B. Tian, S. Yin, and S. Wei, "Efficient hardware architecture of softmax layer in deep neural network," 11 2018, pp. 1–5.

[6] I. Kouretas and V. Paliouras, "Simplified hardware implementation of the softmax activation function," 05 2019, pp. 1–4.

[7] H. Labs. (2008) Cacti - an integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. [Online]. Available: https://www.hpl.hp.com/research/cacti/

[8] Z. Li, H. Li, X. Jiang, B. Chen, Y. Zhang, and G. Du, "Efficient fpga implementation of softmax function for dnn applications," 11 2018, pp. 212–216.

[9] NCSU. (2018) Freepdk45. [Online]. Available: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[10] Synopsys. (2018) Designware library - datapath and building block ip. [Online]. Available: https://www.synopsys.com/dw/buildingblock.php

[11] O. Vinyals and G. Friedland, "A hardware-independent fast logarithm approximation with adjustable accuracy," in *2008 Tenth IEEE International Symposium on Multimedia*, Dec 2008, pp. 61–65.

[12] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, "A high-speed and low-complexity architecture for softmax function in deep learning," 10 2018, pp. 223–226.

[13] B. Yuan, "Efficient hardware architecture of softmax layer in deep neural network," in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, Sep. 2016, pp. 323–326.

[14] W. Yuan and Z. Xu, "Fpga based implementation of low-latency floating-point exponential function," vol. 2013, 01 2013, pp. 237–240.