CPRE 587 Project Proposal

Jake Hafele, Thomas Gaul
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50014

I. Introduction

The goal of our project was to design and implement multiple fixed point integer MAC units for a fabricated ASIC design. We utilized existing code that has been verified and synthesized from labs 3 and 4, including our integer quantized MAC unit and variable MAC unit. We used these simple designs as a baseline since they have been tested and proven in existing labs, to ensure we can hit the tapeout deadline for the target digital ASIC which is November 11, 2024.

Putting our MAC designs within this chip will allow ISU's ASIC design club Chip Forge the opportunity to test a fresh and novel design that relates to the CPRE 587 coursework. This would provide an opportunity for lab partner Thomas Gaul, other potential students, or Chip Forge members to test a MAC unit or multiplier within the digital ASIC design, and integrate it with the on chip 3 stage open source RISC-V core.

As a personal goal, this project was a good push to capitalize on personal experience of both members related to Chip Forge. Jake Hafele has spent time working with the same process as both a senior design member and a graduate researcher. Utilizing this background helped enable the club to have another design to validate next semester, as well as providing a way for us to save time by knowing the design flow. Thomas Gaul has the opportunity to write C firmware and validate this design in hand next semester, which is a novel opportunity before he graduates and starts his industry career in digital ASIC verification.

The ASIC design process is based on utilizing all open-source tools, including digital verification, synthesis, and place and route operations. The company leading the process is called eFabless, which utilizes the SkyWater 130nm open-source PDK to fabricate designs. Multiple previous senior design teams have designed projects under the sponsored OpenMPW shuttle program. Chip Forge has received a grant for a grant from the National Science Foundation for six independent tape-outs over a 3-semester term.

II. Related Work

Related work for this project falls into two subsections. The first is prior work exploring MAC unit designs or papers regarding the open-source tools we make use of in this project.

A. Related Work on ASIC Design

The following paper [1] compares multiple open source ASIC design flows including RTL simulation, synthesis, gate level simulation, place and route, timing analysis, and floor planning. This gives a perspective of the span of potential open source tools to use in the digital design flow. Notably eFabless supports using OpenLane, which uses iverilog, Yosys, and Netgen, to name a few relevant tools.

B. Related Work on MACs

The first paper we looked at was the Design and Performance Analysis of Multiply-Accumulate (MAC) Unit [2] the paper explores a handful of different multiply hardware designs in the scope of a MAC unit. It implements an Array Multiplier, a Ripple Carry Array Multiplier with Row Bypassing Technique, DADDA Multiplier, and a Wallace Tree Multiplier. These multiplier designs are interesting compared to our Verilog and will flatten into something similar. If the tool does not allow for synthesizing the multiply operator, we will have to implement a design similar to these in hardware.

The next paper we explored is the Design of High-Performance 64-bit MAC unit [3]. This paper employs a different technique for making up their multiplier and uses a modified Wallace multiplier with a carry-save adder. They go into design points and results that are similar to what we want to see on our MAC design(s).

Finally, in this project, we want to have the option to pack multiple data pieces into one 32-bit word that

is passed to the Mac, and then the MAC is calculated for all the pieces. We found a patent that does just this with two 16 bit numbers [4]. We want to expand on this to do 4 8-bit, 8 4-bit, and 16 2-bit.

A novel approach is to use a different data type for MAC units, such as posit [5]. Posit can achieve higher levels of accuracy with the same amount of bits as fp32, or similar levels of accuracy with lets bits. This format includes a single sign bit, an exponent and fraction like fp32, but includes a new regime field. The regime field can represent one or multiple bits, and is a run of either multiple 0's or 1's, and terminated with the opposite value. This means the values can scale to larger proportions with lower amounts of bits, making them beneficial for DNN applications.

C. Quantization

A large motivation for our design is to use variable precision multiply and accumulate blocks to support multiple integer precision levels. One paper utilizes a Minimum Mean Squared Error problem to handle weight and activation input quantization without a full retraining of the model [6]. This is applied to 4 bit integer quantization levels, which is one of the quantization levels that we are implementing in our variable MAC design. The target designs are low power accelerators for pretrained models, in which the low power accelerators will benefit from the low precision computations by utilizing energy savings.

Like the last paper, another related work also focuses on utilizing pretrained fp32 models and converting them to fixed point quantized integers, to reduce the amount of energy and latency overheads required with retraining a model in integer form [7]. Mathematical algorithms are derived to determine the most optimal number of fixed point quantized bits per layer based on a logarithmic equation. These optimizations are performed for many popular datasets, including CIFAR-10 and ImageNet.

Another paper covers training quantized neural networks with as low as 1 bit precision weight and inputs at run time [8]. Similar datasets to before were used, including MNIST, CIFAR-10, and ImageNet. Due to the reduced quantization levels of single bits, most arithmetic operations during training are reduced to bitwise operations. While our implementation of the variable mac unit has no bitwise operations, the lowest level of precision is an int2 multiply and accumulate, which is near in terms of size.

Additional frameworks have been proposed which utilize the speedup of quantized networks, one of which being ReLeQ [9]. This framework utilizes the proximal policy optimization to determine how to quantize hyper parameters. Like other examples, an already trained model is originally used as a baseline, and will be automatically analyzed to determine the most optimal number of bits with a slight loss in accuracy. Within the context of our variable mac unit, we would assume a pre trained model with fixed point values would be used, or converted from fp32 before being inserted into our digital ASIC design.

D. The Iris Dataset

A final aspect we added onto this project was running an actual DNN model on our hardware design. A common toy dataset used in many machine learning explication and tutorials is the Iris dataset first [10]. This data set has 4 data points associated with different dimensions of Iris flower petals and an associated specific iris breed out of 3. This dataset consists of 150 data points and worked out well to make a model we could fit on our device given the memory constraints.

Another piece related to this described a model that had two hidden layers with 50 neurons and 30 neurons and explored activation functions with the Iris dataset. Unfortunately, we could not use their size due to the memory constraints on the processor but we did use their design as a starting point [11]. Additionally, they showed the performance of different activation functions with Relu on all layers, performing as one of the best options. Even though it was not the absolute best we still used it as it was the easiest nonlinear one to do on our hardware.

III. Design

A. Tools

There are multiple required tools for the digital ASIC design flow as supported by eFabless, that are needed for end to end development and verification of a digital ASIC design. We are limited to the toolflow of eFabless, and have to conform to the performance of the open source tools and repository that is given. This can provide challenges, as the open source tools are not as well supported as proprietary ones.

Included tools in the ASIC design flow:

- Iverilog (Digital Verification)
- Yosys (Synthesis)
- OpenROAD (Place and Route, DRC, LVS, STA, etc)
- GTKWave
- KLayout

There are multiple resources we can utilize that have been created as part of ChipForge, the new ASIC design club at Iowa State University. As part of Jake Hafele and Gregory Ling's graduate research, multiple toolflows and tutorials have been created to assist in designing, verifying, and validating digital ASIC designs. This gives us much more confidence that we will be able to produce a tapeout ready design in 3 weeks.

Internal tools from ChipForge:

- Supported toolchain on ISU Linux machines
- Artix-7 FPGA Dev Board
- Documentation for running tools

B. Variable Precision MAC

The variable precision MAC will multiply together both integer activation inputs and weights together and write them to a shared accumulation register. This design is synchronous to the positive edge of a clock. A synchronous active high reset exists to clear the accumulation register when a new MAC operation should start. A synchronous active high enable also exists to control when two data inputs should write and sum with the existing contents of the accumulation register.

These 32 bit inputs can be quantized at 32/16/8/4/2 bit levels, and data can be packed to ensure for more performant throughput between the two 32 bit ports. to pack data at smaller quantized levels, the data and weights to be multiplied together should be in the same bit positions across each port.

```
For example, with a 16 bit quantization:
Data
    [31:16] Data_1
    [15:0] Data_0
Weight
    [31:16] Weight_1
    [15:0] Weight_0

MAC = Data_1 * Weight_1 +
      Data_0 * Weight_0
```

The entire interface to the MAC unit is done over the 128 logic analyzer pins that are connected directly between the management core and digital user design area. The following ports are mapped as input/output on the logic analyzer bus:

```
LA Outputs:
    [127:96] o_ACCUMULATE: Output of accumulate
    [95:0]: NOT USED, tie low

LA Inputs:
    [127:69] NOT USED
    [68] I_RST: Synchronous high reset
    [67] I_EN: Synchronous enable
    [66:64] i_SEL: Quantization level select
    [63:32] I_DATA: Activation inputs
    [31:0] I_WEIGHT: Weight inputs

System inputs
    i_CLK: Clock from wishbone bus

i_SEL Quantization Levels:
    000: 32 bit
    001: 16 bit
    010: 8 bit
    011: 4 bit
    100: 2 bit
```

The input enable control will write a MAC accumulate when the control signal transitions from a 0 to 1, on the positive edge of the signal. This control bit is stored in two DFFs, to guarantee only one MAC is performed per operation from the RISC-V core. This is required since the clock cycles in the digital ASIC design are much faster than the RISCV core, which would lead to a copious amount of MACs per 1 intended operation.

A snippet of the RTL code of the 32 bit integer MAC operations is given in Figure 1. The open source synthesis tool Yosys was able to infer how to synthesize a multiplier and adder block based on the

```
* and +
```

operators in the Verilog language. Additionally, there were cells to support this synthesized list in the open source SkyWater 130nm PDK, enabling passing place and route and implementation in the final design.

C. Unit Level Testing

The unit-level test was designed to fully automate tests with random inputs to try to reach edge cases

```
assign s_mult_32 = i_WEIGHT * i_DATA;
assign s_acc_32 = s_mult_32 + s_acc_reg;
```

Figure 1: Verilog Snippet

and be able to reach testing amounts that we could not otherwise do manually. The unit test file is set up with functions for MACing two 32-bit numbers, four 16-bit numbers, eight 8-bit numbers, sixteen 4-bit numbers, and thirty-two 2-bit numbers. The result of each MAC is against an internal calculation of the expected value. The unit test is set up to feed random numbers up to 1000, 1000, 255, 15, and 3, respectively. It does this on a random amount of times up to 1000 times, being checked for correctness all the time. It is set to run this test on each MAC size, and then finally, there is a test that randomly selects from each version each MAC and resets the MAC unit at random to ensure that a combination of MAC occurs to catch odd behavior. The waveforms are shown in Figure 8. In addition to testing our design, it hopefully can function as a template for future Chip Forge members to use. The design passed these tests with both RTL test and gate-level simulations.

D. Hardening

Hardening is the term used by eFabless which envelops synthesis, placement, routing, and DRC/LVS checks, all in one package of steps. It was critical that we guarantee our design can pass through all of these checks before handing our tested Verilog MAC module to senior design, to ensure that they would have no issues integrating our project with the rest of the design. Hardening of each module is defined by a config .json file. We hardened our mac module individually to generate a synthesized netlist that could be used for gate level testing with the same derived testbench as our RTL sims. To verify our design works as expected in the integrated user framework, we also hardened our design within the user project wrapper module.

Our target design space was 500 by 500 um of design area, with a utilization of 0.55. There is lots of area left in the tapeout chip for projects, so space is a nonissue as of now. The key concern is verifying that the Verilog will synthesize with the use of the multiply and add operators in Verilog.

E. System Level Testing

Once we had the design hooked up to the RISC V core, we wrote a short piece of C code that does one of each MAC one after another, as shown in Figure 12. Going over the math, all the pieces work out properly it is interesting to see how many clock cycles go into MACing two items.

F. Running on FPGA

To start off the testing on the FPGA, we wrote up an interface for every size of MAC with the hardware and an equivalent software MAC implementation function set. With these functions, we wrote the equivalent to the simulation above in the System level testing. As shown in Figure 2 the outputs match between software and hardware as well as the output in Figure 12. This did a basic test of all MAC types. After additional tests verifying the functionality of the design. We started to do a timing analysis of our design vs the software design. We ran into the problem there that multiply was being handled in compile time for the program. To circumvent it we input the variables to be MACed via UART. As such, they varied from 0 to 255 and were inserted randomly by us hitting them into the keyboard at random. This brought us to the next problem the RISC-V core does not have support to multiply instructions. This did not come up previously as it was being completed at compile time. To get any form of comparison we wrote a software implementation for MACs that does not use any multiply instructions. We output the clock time from each MAC and verified it outputs the same between software and hardware as shown in Figure 3. We ran the timing test on various MAC counts from 0 to 100 as shown in Figure 4. We can see in the graph there is a very significant speed-up between software to hardware. This is due to the software implementation taking many clock cycles to do a MAC, whereas our hardware does it in 1 in addition to the cycles to set it in memory to go to the hardware. It can be noted that the FPGA does not run on a linear time with the number of MACs the hardware completes, unlike the software, which is quite linear. Upon some investigation into the generated assembly and comparison between the lower counts of MACs and the higher counts of MACs, there is a difference in how the loops are generated, causing an increase in runtime. Additionally, we did

further evaluation of our design by making a toy model with the Iris dataset [10].

```
Software:  617  Hardware:  617
```

Figure 2: Simple test outputs

```
Software: 0x0003bed8  Time: 0x0000f54b
Hardware: 0x0003bed8  Time: 0x00000e7d
```

Figure 3: Timing test output

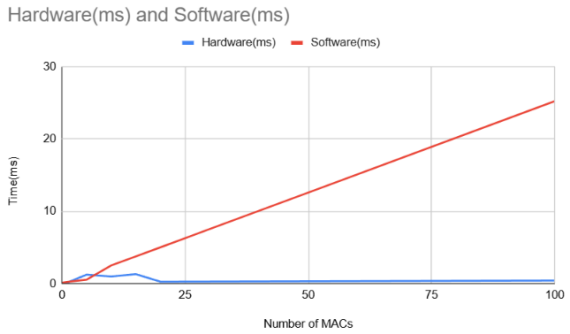

Figure 4: Timing test comparison

G. Tapeout Integration

Our design was shared with the senior design team sddec24-12 which was responsible for completing the first paid tapeout for ChipForge. We shared the same submitted design repository to the ChipForge ISU Gitlab group so they had access to all listed documents. Primarily they needed the mac.sv source file, so it could be instantiated within their user frame.

H. Iris Dataset

To fully realize our hardware MAC accelerator, we wanted to run a true model on it. With limited memory constraints, we chose to do the Iris model, which has four variables for the flowers and an associated Iris species out of three. It has 150 elements, 50 for each flower type, which we split 120 for training and 30 for testing. We trained a dataset with 4 dense layers of sizes 4x10, 10x10, 10x5, and 5x3 and achieved with Relu activation functions as shown in Figure 5. We trained the model with a batch size of 1 and 100 epochs as we saw the best results with these parameters and got a test dataset accuracy of 96 percent, although I suspect there is overfitting going on given the small amount of data we are working with. To run this on our hardware, we shifted all the values to the left 16 bits and dropped the remaining fraction to make it Q16.16 datatype to avoid losing precision but still function without and floating point arithmetic. With this we wrote a function to complete dense layers in C to run on the FPGA with the 32-bit MAC function written previously to test our design. Additionally, we did not write a Softmax function as that would not have been efficient given the hardware, but we could still select the most likely class from the maximum. With this complete, we tested several pieces of data to ensure the code correctly identified the flowers.

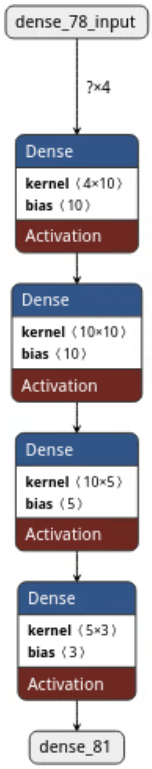

Figure 5: Iris model design in Netron

IV. Final Status

A. Unit Level Testing

Figure 6 below shows the passing simulation results of our RTL unit test for the mac module.

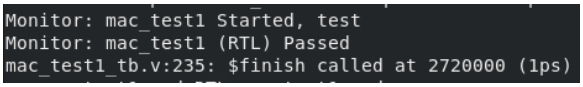

Figure 6: Unit Test RTL Results

Figure 7 below shows the passing simulation results of our GL unit test for the mac module.

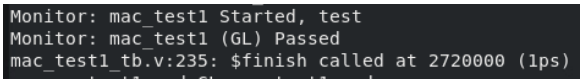

Figure 7: Unit Test GL Results

Figure 8 demonstrates a sample waveform sim for testing randomized data and weight inputs for variable precision levels of our mac module. The unit testbench automatically determines what the expected output should be, and compares it to verify functional correctness with randomized input stimulus.

B. Hardening

Figure 9 showcases a code snippet of the synthesized netlist that we generated through the hardening process. This was important as it was required to run gate level simulations and verify our synthesized netlist, which would be placed in the ASIC tapeout design, had the same functionality as the RTL simulations.

Figure 10 represents the physical design of our MAC unit after going through synthesis and place and route operations. The sample hardening was done to generate a synthesized gate level netlist to simulate, as well as verify our design would pass through hardening without any issues.

We were able to successfully pass our design instantiated within the digital user design area all the way through precheck, as shown in Figure 11. With this complete, we were ready to integrate out design into the Senior Design Tapeout framework.

C. System Level Testing

Figure 12 demonstrates a sample waveform sim for a piece of C code that sets up the processor for using the MAC hardware and does one MAC for each type 32-bit, 16-bit etc. Each value is added to the previous one.

D. FPGA

We were able to verify our hardware on an FPGA and do timing analysis with a software-exclusive design. With the working hardware, we were then able to extend our hardware to work on a functioning DNN model for the Iris dataset.

E. Tapeout

After our design was integrated into a caravel user project design and passes all functional tests, we delivered it to ChipForge to be integrated into their Fall 2024 tapeout. The senior design team was successful in integrating our MAC design into their framework, and were able to successfully pass connectivity checks to ensure our design was still functional after being integrated into a new user design area. The design was synthesized again and passed both remote precheck and tapeout checks, guaranteeing a passing design to eFabless. Currently the tapeout design has been submitted and will be sent off to be fabricated and packaged soon. We are expecting to receive the ASIC in Spring 2024, which Thomas Gaul will be able to validate as part of ChipForge's bring-up process.

V. Documentation

The main documentation for how to interface with the MAC design as well as how to run all of the tools for each stage of the design flow is located in the root README.md within our repository deliverable.

Within our submission, the following key paths are important:

- Verilog RTL:
  `/verilog/rtl/mac.v`
- Unit Tests:
  `/verilog/dv/mac_test1`
- Hardening:
  `/openlane/mac`
- Synthesized netlist:
  `/verilog/gl/mac.v`

VI. Challenges

One design challenge we had was how to handle the enable control. Since the clock in the RISCV core was much slower than the digital design area, we would not be able to assert and deassert the enable control input on the same clock edges as our digital design. To solve this, we created two DFFs to store the two most recent values of the input enable control, which were sampled on the positive edge of the digital domain clock. When the most recent enable register was 1,

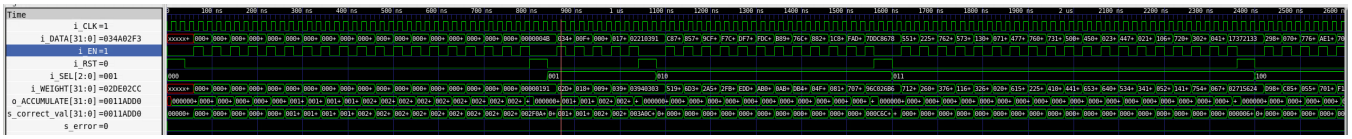

Figure 8: RTL Unit Test Waveforms

```
sky130_fd_sc_hd__nand2_1 _07589_ (.A(_00623_),
    .B(_01850_),
    .VGND(vssd1),
    .VNB(vssd1),
    .VPB(vccd1),
    .VPWR(vccd1),
    .Y(_01872_));
sky130_fd_sc_hd__nand2_1 _07590_ (.A(_01861_),
    .B(_01872_),
    .VGND(vssd1),
    .VNB(vssd1),
    .VPB(vccd1),
    .VPWR(vccd1),
    .Y(_01883_));
sky130_fd_sc_hd__and4_2 _07591_ (.A(net121),
    .B(net243),
    .C(net238),
    .D(net393),
    .VGND(vssd1),
    .VNB(vssd1),
    .VPB(vccd1),
    .VPWR(vccd1),
    .X(_01894_));
sky130_fd_sc_hd__a22o_1 _07592_ (.A1(net121),
    .A2(net238),
    .B1(net393),
    .B2(net243),
    .VGND(vssd1),
    .VNB(vssd1),
    .VPB(vccd1),
    .VPWR(vccd1),
    .X(_01905_));
sky130_fd_sc_hd__and2b_1 _07593_ (.A_N(_01894_),
    .B(_01905_),
    .VGND(vssd1),
    .VNB(vssd1),
    .VPB(vccd1),
    .VPWR(vccd1),
    .X(_01916_));
sky130_fd_sc_hd__nand2b_4 _07594_ (.A_N(_01894_),
    .B(_01905_),
    .VGND(vssd1),
    .VNB(vssd1),
    .VPB(vccd1),
    .VPWR(vccd1),
    .Y(_01927_));
```

Figure 9: GL Netlist

and the past enable register is 0, then a rising edge is detected and the accumulated value is written to the accumulation register. This did not show up in our unit tests, but was caught in our C code testing. Without

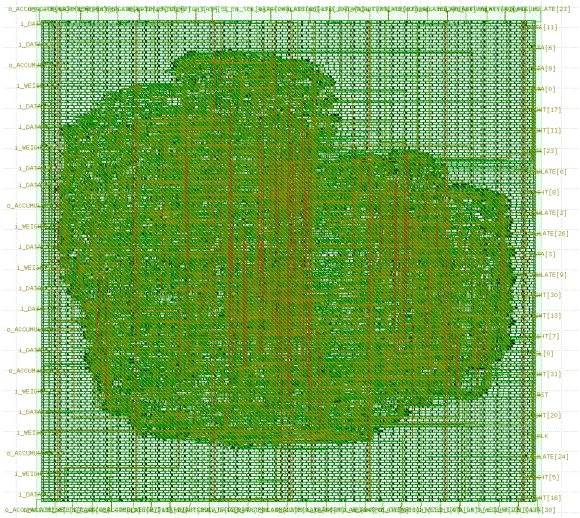

Figure 10: Hardened MAC Module

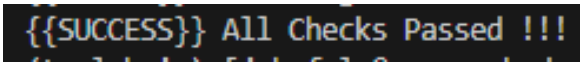

Figure 11: Passing Precheck result

change, it would have made our design nonfunctional after tapeout.

Our schedule was rather accelerated due to the tapeout for the chip being on November 11th. We had to skip starting lab 6 to guarantee our chip would make it on the tapeout, which expedited the design and testing process. Figure 13 shows our planned schedule, which we ended up meeting as expected.

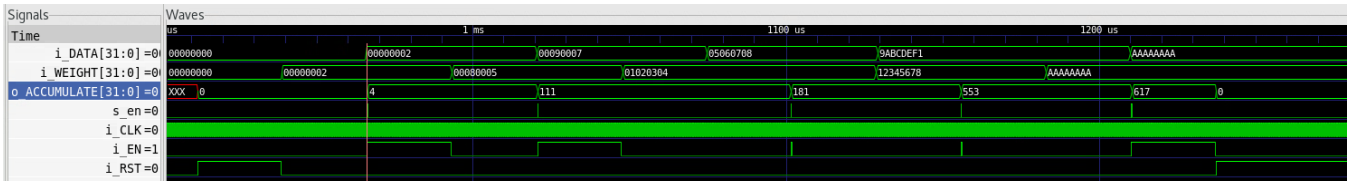

Figure 12: RTL C test Waveforms

Oct 11- Oct 27	Oct 28- Nov 3	Nov 4- Nov 10	Nov 11	Nov 18- Nov 24	Nov 18- Nov 24	Nov 25- Dec 1	Nov 25- Dec 1	Dec 2- Dec 9	
Capture Design Requirements	Auotmated RTL Sims	Test hardware on FPGA	Tapeout Due	Jake suffers through Master's Defense etc	Jake suffers through Master's Defense etc	Thanksgiving break	Lab 6	Lab 6	Jake
RTL Design	GL Sims	Write tests in C		Write up user documenation	Write final report/pres				Thomas
RTL Sims	Iterate on RTL bugs								Both
Initial Synthesis	Integrate into Tapeout Design								

Figure 13: Proposed Schedule

REFERENCES

[1] R. T. Edwards, M. Shalan, and M. Kassem, "Real silicon using open-source eda," *IEEE Design Test*, vol. 38, no. 2, pp. 38–44, 2021.

[2] P. Jagadeesh, S. Ravi, and K. H. Mallikarjun, "Design of high performance 64 bit mac unit," in *2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT)*, pp. 782–786, 2013.

[3] M. Sai Kumar, D. A. Kumar, and P. Samundiswary, "Design and performance analysis of multiply-accumulate (mac) unit," in *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*, pp. 1084–1089, 2014.

[4] D. Deng, A. Jebson, Y. Liao, N. C. Paver, and S. J. Strazdus, "Multiply-accumulate (mac) unit for single-instruction/multiple-data (simd) instructions," Sep 2006.

[5] H. Zhang, J. He, and S.-B. Ko, "Efficient posit multiply-accumulate unit generator for deep learning applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2019.

[6] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3009–3018, 2019.

[7] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," *CoRR*, vol. abs/1511.06393, 2015.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *Journal of Machine Learning Research*, vol. 18, no. 187, pp. 1–30, 2018.

[9] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "Releq : A reinforcement learning approach for automatic deep quantization of neural networks," *IEEE Micro*, vol. 40, no. 5, pp. 37–45, 2020.

[10] R. A. Fisher, "Iris." UCI Machine Learning Repository, 1936. DOI: https://doi.org/10.24432/C56C76.

[11] A. ELDEM, H. ELDEM, and D. ÜSTÜN, "A model of deep neural network for iris classification with different activation functions," in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, pp. 1–4, 2018.