FPGA and FeFET Hardware Acceleration of CNN based SDR Analysis

Joshua Mayersky and Corey Butts

***Abstract*—A semester project for the Neuromorphic Computers for AI course is presented. Three different hardware acceleration methods for convolutional neural network implementations for were explored for an application space of detecting and correcting signals in software defined radio due to hardware trojans. Two methods using traditional computing architectures: SoC, FPGA, and one method using a matrix of FeFETs.**

***Index Terms*—Software Defined Radio, CNN, FPGA, FeFET, Hardware Acceleration**

I. Introduction

Hardware acceleration of neural network implementation is an attractive and active field of research and development. This is for many reasons, both for processing speed and latency, but also for overall power consumption of a network. While this is a more general concept used in many fields of applications, it's particularly of importance in small, embedded applications, where power consumption and speed of operation are particularly important variables to pay attention to.

A. SDR Analysis

Software defined radio (SDR) is an embedded system that enables the implementation of radio systems, traditionally composed of ASIC hardware, in software giving them the benefit of versatility and potability. It also allows for the integration of other cutting edge software-based technologies such as machine learning algorithms. However, the main drawback of all software is that it can be exploited which can lead to a variety of unwanted outcomes. Utilizing machine learning algorithms against hardware trojans has been explored recently [1] but with varying degrees of success. Not to mention, to our knowledge, there hasn't been a machine learning solution proposed that implements trojan detection and mitigation.

The machine learning model we chose to evaluate for this project is based on the architecture proposed in [2] which uses a bidirectional recurrent neural network (BRNN) to reconstruct anomalous sensor data. Our architecture differs in that we use a one-dimensional convolutional neural network (CNN) in place of the LSTM layer. This was chosen due to the CNNs ability to create similar temporal relationships within time series data as a LSTM network would, but at faster training times and inferencing speeds, the latter of which is crucial within a real-time SDR.

B. Hardware Acceleration of CNNs

Hardware acceleration of deep neural networks and convolutional neural networks is an ongoing field of research. Traditional methods of hardware acceleration make use of FPGAs to offload processing from the main computer [3]. Implementations using unconventional devices and architectures range from using memristors [4] or multiple resistance level RRAM [5] as a synapse in a convolution operation, to domain wall memory [6] based convolutional layers. Recent work on ferroelectric transistors show promise over existing technologies due to lower levels of leakage current and increased computing density [7] [8] [9].

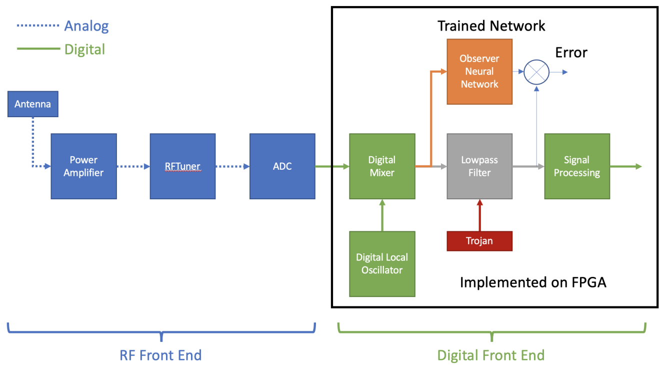


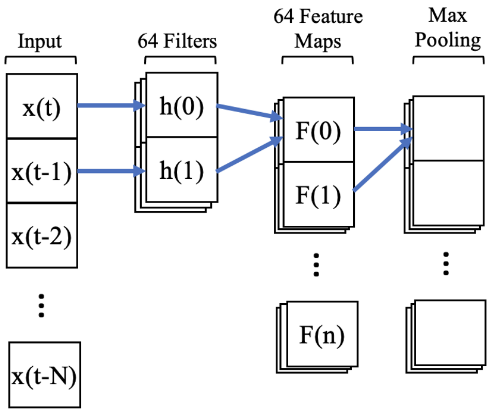
Fig. 1: Block diagram of a SDR circuit with the network observing the low pass filter

II. Simulation Methodology

The goal of this work is to evaluate the effectiveness of three different CNN layer implementations: using SoC, FPGA and FeFETs. The following subsections will outline how each method was implemented.

A. SoC Implementation

For the SoC implementation we chose the Jetson Nano which is an ARM based computer specifically designed for embedded machine learning applications. The original model was implemented in Python using TensorFlow, both of which are supported on the Jetson Nano. The model that we found to be most effective was saved and transfered to the Jetson Nano. Since the focus of this work is on benchmarking the CNN performance, the weights of only the CNN layer were utilized. A network consisting of a single CNN layer was created and the weights of the selected network's CNN layer were transferred to the single CNN layer. We used a 32 sample (e.g. $[x(t), x(t-1), .., x(t-31)]$ input to feed into the network. The output was obtained and the time taken for inferencing was recorded.



$$F(0) = x(t)h(0) + x(t-1)h(1) + b$$

Fig. 2: Visual representation of the convolutional layer

B. FPGA Implementation

FPGA has become very popular in recent years due to its ability to enable parallelism. The CNN layer, as shown in Fig. 2, is implemented by performing a convolution on the input using filters. The convolution done by each filter results in a feature map which gets fed into the rest of the network. Since convolution involves simple math operations (addition and multiplication) and each feature map value is independent of the rest, those operations can be performed in parallel. A Conv2 circuit was modeled in VHDL, which takes five 16-bit inputs:

- 2 input samples
- 2 filter values
- 1 filter bias

and outputs a single 32-bit number. One output from the Conv2 block corresponds to one value in the feature map. Therefore we can scale the whole model to compute every feature map value in parallel. Once the design was modeled, it was simulated using the same input that was generated by the Python script used on the Jetson Nano.

C. FeFET Convolution Operation

Based on the work of Long, et. al. [7] [8] [9], a design for the FeFET weight matrix crossbar array was settled upon. It makes use of FeFETs for a digital AND operation between the input on the gate, and the threshold voltage of the FeFET. The operating principal behind the multiple threshold voltage states in an FeFET is due to the internal remanent polarization that can be switched via an externally applied field. When the polarization is oriented in one direction, the threshold voltage of the transistor is high, whereas when the polarization is oppositely oriented, the threshold voltage of the transistor is low. This can be thought of as being due to an effective modulation of the channel length by ferroelectric polarization inducing charge accumulation and depletion in the channel. In order to visualize this property for the digital AND operation, a truth table is provided in Table I.

TABLE I: FeFET Digital AND Operation

FeFET V_T	Input (Gate)	Output
High (0)	0	Low current (0)
High (0)	1	Low current (0)
Low (1)	0	Low current (0)
Low (1)	1	High current (1)

This logical AND operation allows us to perform 1bit multiplication. By encoding the weight vector of a CNN filter kernel operation as the threshold voltages of an FeFET array, and applying the properly formatted binary input values on the word lines of the FeFET arrays, sequential AND multiplication operations occur. This is the core of the FeFET Matrix Multiplication Acceleration (FeFET-MMA) circuit. A block diagram of a single kernel may be seen in Figure 3. The blocks outlined in the green dashed line makeup the kernel filter, which interfaces with the two 16-bit inputs, a 16-bit bias, as well as a 36-bit output. The internal blocks are the FeFET crossbar array, Peripheral Control Circuitry (outlined in the dashed yellow box), and a final adder for the biasing.

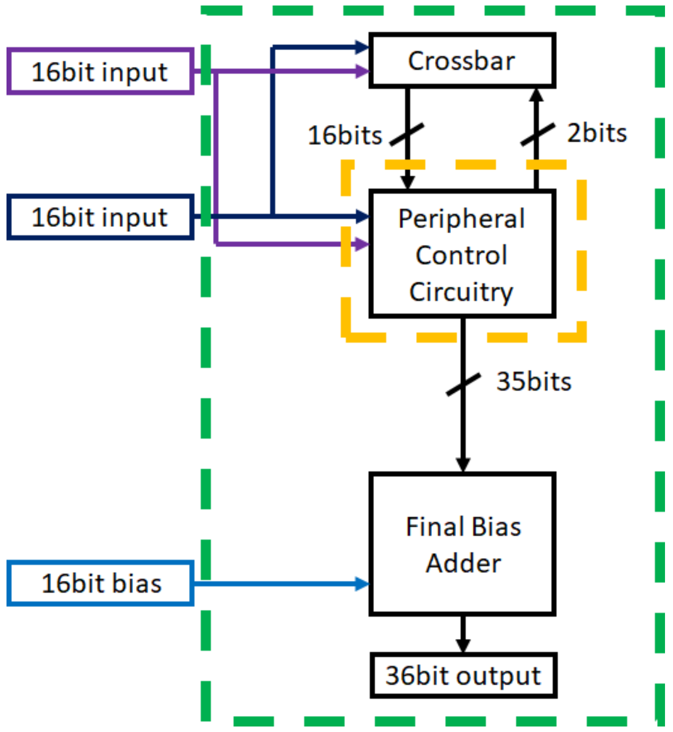


Fig. 3: Block Diagram of the the MMA Kernel

We will now delve into the Peripheral Control Circuitry in order to gain a greater understanding of how the circuit works. An exploded block diagram may be seen in Figure 4. In order to not activate more than one word line at a time, and so enter the the realm of analog computing (which requires the use of power and area expensive ADCs and DACs), we should only activate one word line at a time. This is done by using a 1-hot enable vector and a word line decoder to only activate an appropriate bit-line with the corresponding bit-place of an input in any given clock cycle. The outputs from each bit-line in the crossbar array are fed into individual sense-amplifiers (design based on [10] [11]), whose outputs are then fed into individual counters. This allows us to count how many logic ”1”s have been seen on each bit-line in every clock cycle of operation. The MSB outputs of each sense-amplifier counter must then be added to their more significant neighbors. This is done in the SA Adder module.

Following this manipulation, it must be multiplied by the bit place value 2^X, where X is the bit-place of the input vectors that are being analyzed in the current clock cycle. This is monitored via the bit-place decoder module which reads the outputs from a clock cycle counter. The output of the multiplication operation is then stored in a DFF array through a buffered connection, for later use. The signal to capture the values being output from the multiplier is determined via the Controller, where reset signals for the SA counters are also determined based on the clock counter. In our case, since we have 2 inputs, we dedicate 2 clock cycles per bit of input, plus 1 clock cycle/bit of input for the capture signal, and 1 clock cycle/bit for the reset signal. This allows the output from the multiplication operation to be properly captured, and the Sense-Amplifiers to be reset in preparation for the next bit of input. This circuit dynamically calculates the result of the convolution operation, adding each subsequent bit-place together and storing it in the DFF-2 array. In particular we have Dff_1 + Dff_2 => Dff_2. This reduces the total amount of clock cycles needed for computation, as well as area overhead as more registers would be needed to store the outputs from every bit of input. Buffered connections were used between each input/output of the flip-flops in order to not violate the setup and hold times or inadvertently cause race conditions.

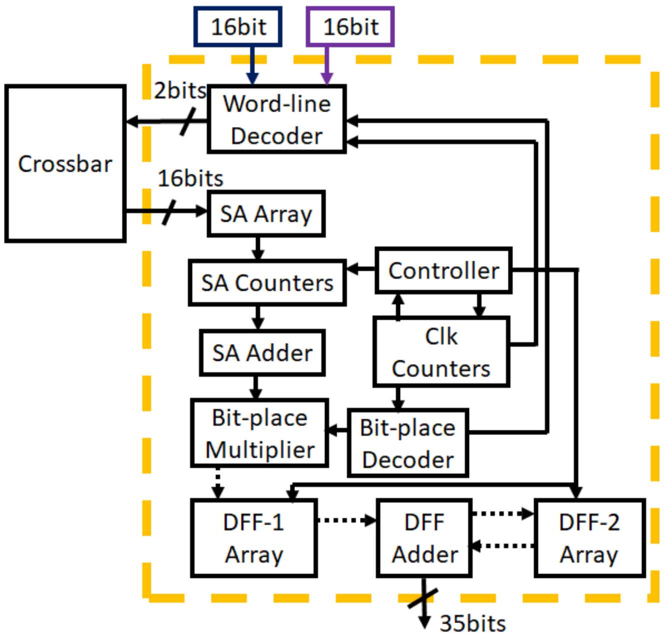


Fig. 4: Block Diagram of the the MMA Peripheral Control Circuitry in the Kernel. Dotted black lines between modules indicates a buffered connection

This design was implemented in HSPICE using variable threshold voltage transistors. The FeFETs were modeled using a traditional MOSFET model at the 45nm technology node, UC Berkeley’s BSIM 4, with the threshold voltage parameter set to be 0.1Vdd for the low threshold voltage state, and 0.9Vdd for the high threshold voltage state. In the rest of the circuit, each normal MOSFET was specified to have a threshold voltage of 0.15Vdd. In all simulations we use Vdd as 1V. The width of each MOSFET used throughout the circuit varied on the gate and sub-circuit used. In general care was taken to ensure a Wp/Wn ratio of 2.5, but this varied in some cases. In particular Wp ranged from 240 nm - 500 nm, while Wn ranged from 120 nm - 300 nm. These widths were experimentally selected to ensure steep transition slopes and to adhere to the clock timing constraints.

III. Results and Discussion

A. SoC

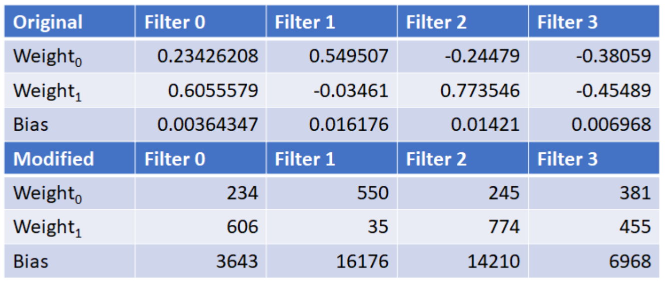
As stated previously, the SoC implementation utilized the same Python script that was already being used to simulate the model. Therefore the output of the implementation was used as a benchmark for the other two implementations. The model was able to achieve a throughput of 12,500 samples/s which is similar to the machine that the model was originally simulated on. This statistic is impressive given that the dimensions of the entire board are much smaller at 80 mm x 100 mm. It is also possible that the throughput could be increased tenfold based of performance boost claims made by Nvidia when using their TensorRT library.

B. FPGA

We were successful at simulating the FPGA implementation. The FPGA was able to achieve an accuracy of 100% in comparison to the Jetson Nano output. However this was using the ”real” data type in VHDL (i.e. floating point precision) which means that the design could not be synthesized. Since floating point arithmetic in VHDL is very difficult and designs typically require a lot of chip space, we decided to implement the design using fixed point precision weights, biases and inputs. Quantization has been shown to maintain network accuracy while reducing memory storage significantly [12]. The design was implemented and simulated using the same input values obtained from the Jetson Nano simulations. The values had to be converted from 32-bit single precision floats to 16-bit half precision floats. The results of the simulation were completely inaccurate. We attribute this inaccuracy to the value conversion process but this issue needs to be looked into further. The FPGA did show some promsing results as far as speed goes. The total delay, from input to output, for the entire network was approximately 15 ns which translates to a throughput of approximately 66 million samples per second. This is incredibly faster than the SoC implementation while being able to fit on a die of 18 mm x 20 mm based on the Kintex7 dimensions.

C. FeFET MMA

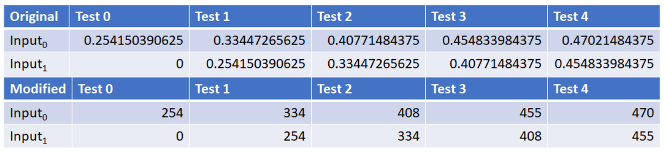
In the design phase of the MMA circuitry development, a few key assumptions were made. First, that the binary numbers used as the weight and input vectors are unsigned numbers. Second, the operations performed are integer, rather than floating point operations. This was done in order to simplify the design process, as accounting for signed numbers increases the peripheral circuitry overhead, which at the time, was already ballooning. In addition, there is a trade-off inherent to the conversion process from floating to fixed point operations. Namely: higher speed, lower area overhead, and lower power usage at the cost of precision. These modifications may be seen in detail in the weight modification in Figure 5 and input modification in Figure 6. In order to convert from the floating point to fixed precision number and maintain the accuracy of the convolution operation, we multiplied all terms by a constant 10^6 and distributed it equally across terms. This resulted in the weights being multiplied by 10^3, the inputs multiplied by 10^3, and the bias multiplied by 10^6. In order to convert back to the floating point result, we can take the output of the circuit and divide it by the afore-multiplied 10^6.



Original	Filter 0	Filter 1	Filter 2	Filter 3
Weight$_0$	0.23426208	0.549507	-0.24479	-0.38059
Weight$_1$	0.6055579	-0.03461	0.773546	-0.45489
Bias	0.00364347	0.016176	0.01421	0.006968
Modified	**Filter 0**	**Filter 1**	**Filter 2**	**Filter 3**
Weight$_0$	234	550	245	381
Weight$_1$	606	35	774	455
Bias	3643	16176	14210	6968

Fig. 5: Original and Modified Weights and Biases for the FeFET-MMA

In order to test the functionality of the MMA circuit, five test vectors were selected and their values converted to fixed point integers. These testing values may be seen in Figure 6. These integer values were then converted to unsigned binary values, and fed into the MMA circuit accordingly.



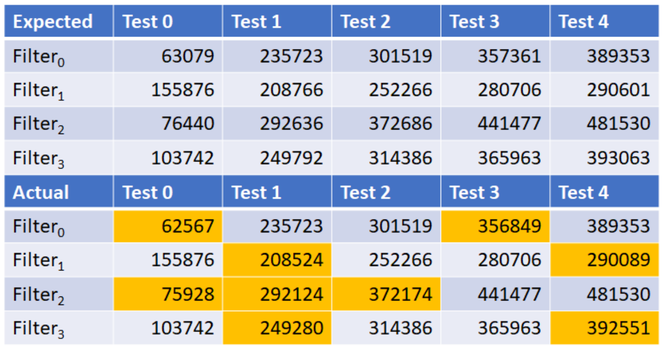
Original	Test 0	Test 1	Test 2	Test 3	Test 4
Input$_0$	0.254150390625	0.33447265625	0.40771484375	0.454833984375	0.47021484375
Input$_1$	0	0.254150390625	0.33447265625	0.40771484375	0.454833984375
Modified	**Test 0**	**Test 1**	**Test 2**	**Test 3**	**Test 4**
Input$_0$	254	334	408	455	470
Input$_1$	0	254	334	408	455

Fig. 6: Original and Modified Inputs for the FeFET-MMA Testing

The expected and actual results from the five test vectors may be seen below in Figure 7. Of the 20 total test outputs, 11 were exactly correct. The remaining 9 incorrect test outputs may be seen highlighted in orange. These incorrect values have an average error of 0.299%, a remarkably low number.

Upon further investigation it was discovered that each incorrect value differs from its expected value by the same amount: 512. When comparing the hamming distance between the expected vs. actual output binary values, it was found that each incorrect vector had a hamming distance of 1, and in the exact same bit-place: 2^9, or 512 in decimal. This bit is stuck at a value of 0. This implies that there is an error in the HSPICE netlist files, likely in the output node for this bitplace. It is likely only an interconnect error near the final stages of output, as the MSBs of each output are correct.

In terms of speed of operation, the clock used was 200MHz, and it took 64 cycles/convolution operation (though this was parallelized slightly by using four kernels). This resulted



Expected	Test 0	Test 1	Test 2	Test 3	Test 4
Filter$_0$	63079	235723	301519	357361	389353
Filter$_1$	155876	208766	252266	280706	290601
Filter$_2$	76440	292636	372686	441477	481530
Filter$_3$	103742	249792	314386	365963	393063
Actual	**Test 0**	**Test 1**	**Test 2**	**Test 3**	**Test 4**
Filter$_0$	62567	235723	301519	356849	389353
Filter$_1$	155876	208524	252266	280706	290089
Filter$_2$	75928	292124	372174	441477	481530
Filter$_3$	103742	249280	314386	365963	392551

Fig. 7: Expected and Actual outputs of the FeFET-MMA Testing, Incorrect Values highlighted in Orange

in a simulated 320 ns/convolution operation, or 9.6μs for four full feature maps. This translates out to an equivalent operating speed of 416,667 samples/second. For the real-world HSPICE simulation, a single test vector of two 16bit numbers convolved with four filter kernels took upwards of 30 minutes of simulation time, and over 1GB of memory. This meant that in order to generate four full feature maps for more accurate comparisons with the alternate implementations, a minimum of 15 hours of simulation time would be needed. This was not done due to time constraints.

IV. Conclusion

The FPGA implementation, while being more difficult to implement, seems to be more attractive than the SoC implementation when considering the throughput required by an SDR as well as the small package that the FPGA can fit into.

FeFET based matrix multiplication for convolutional neural networks remains a promising avenue for hardware acceleration. Though our MMA implementation is not as fast as the FPGA, and cannot handle the full range of inputs and weights, it accurately performs the operations with minimal overhead.

That being said, there are quite a few improvements that could be implemented. First could be reducing the bit-place overhead (this was done assuming the worst case scenario of number addition and multiplication, even though the worst case would never be seen). In addition the capacitive loading of gates was assumed to be constant, whereas it should be adjusted based on the fanout conditions. The transistor sizing could be better optimized to enable higher clock rates and transition delays. The timing of signals could be altered and capacitors tuned to allow for a higher clock rate. In terms of functional improvements, we could alter the design to allow for signed magnitude, and adjust the bit-place multiplication to allow for negative exponents, i.e. floating point precision operations.

We would also like to evaluate the tradeoffs, in terms of speed and accuracy, of using fixed precision values for weights, biases and inputs over single precision floating point values.

References

[1] K. Hasegawa, Y. Shi, and N. Togawa, "Hardware trojan detection utilizing machine learning approaches," in *2018 17th IEEE International*

Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 1891–1896.

[2] S. Jeong, M. Ferguson, and K. Law, "Sensor data reconstruction and anomaly detection using bidirectional recurrent neural network," 03 2019, p. 25.

[3] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "Maloc: A fully pipelined fpga accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2601–2612, 2018.

[4] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–647,647A–647N, Jan 30 2020. [Online]. Available: https://search-proquest-com.proxy.libraries.uc.edu/scholarly-journals/fully-hardware-implemented-memristor/docview/2353091386/se-2?accountid=2909

[5] Z. Dong, Z. Zhou, Z. Li, C. Liu, P. Huang, L. Liu, X. Liu, and J. Kang, "Convolutional neural networks based on rram devices for image recognition and online learning tasks," *IEEE Transactions on Electron Devices*, vol. 66, no. 1, pp. 793–801, 2019.

[6] J. Chung, W. Choi, J. Park, and S. Ghosh, "Domain wall memory-based design of deep neural network convolutional layers," *IEEE access*, vol. 8, pp. 19 783–19 798, 2020.

[7] Y. Long, D. Kim, E. Lee, P. Saha, B. A. Mudassar, X. She, A. I. Khan, and S. Mukhopadhyay, "A ferroelectric fet-based processing-in-memory architecture for dnn acceleration," *IEEE journal on exploratory solid-state computational devices and circuits*, vol. 5, no. 2, pp. 113–122, 2019.

[8] Y. Long, T. Na, P. Rastogi, K. Rao, A. I. Khan, S. Yalamanchili, and S. Mukhopadhyay, "A ferroelectric fet based power-efficient architecture for data-intensive computing," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi-org.proxy.libraries.uc.edu/10.1145/3240765.3240770

[9] Y. Long, E. Lee, D. Kim, and S. Mukhopadhyay, "Flex-pim: A ferroelectric fet based vector matrix multiplication engine with dynamical bitwidth and floating point precision." IEEE, 2020, pp. 1–8.

[10] O. Okobiah, S. P. Mohanty, E. Kougianos, and M. Poolakkaparambil, "Towards robust nano-cmos sense amplifier design: A dual-threshold versus dual-oxide perspective," in *Proceedings of the 21st Edition of the Great Lakes Symposium on Great Lakes Symposium on VLSI*, ser. GLSVLSI '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 145–150. [Online]. Available: https://doi-org.proxy.libraries.uc.edu/10.1145/1973009.1973039

[11] Shalini and A. Kumar, "Design of high speed and low power sense amplifier for sram applications," *International Journal of Scientific & Engineering Research*, vol. 4, no. 7, pp. 402–406, 2013.

[12] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 1131–1135.