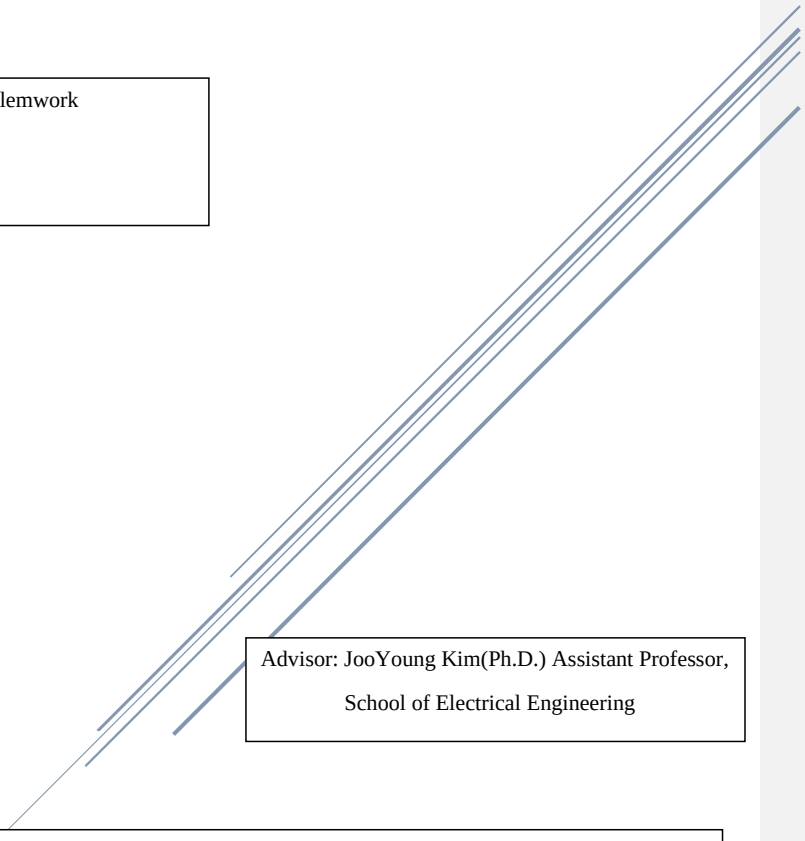
Energy-Efficient XNOR-COUNT Convolution based Embedded Processor System for Edge Inference

Yonas Girma Kelemwork

20170758

12/7/2020



Advisor: JooYoung Kim(Ph.D.) Assistant Professor,

School of Electrical Engineering

School of Electrical Engineering

Korea Advanced Institute of Science and Technology(KAIST)

IBEX-BCX: A Low Cost RISC-V Processor with Bit-Count Instruction Extension for Edge Inference on IoT Devices

Abstract: Deep Neural Networks that ~~are~~ have multiple neural network layers are now ubiquitous for a variety of AI application including image and speech recognition. Embedded devices that are connected to sensors, such as cameras and microphones that constantly gather information about the local environment and send the data to cloud for analysis and prediction on a model such as DNN. This work tries to efficiently run inference on an open-source, RISC-V core from PULP (Parallel Ultra-low Power Processing Platforms) called IBEX. We mapped the binary convolution layer of a pre-trained CNN onto the core by substituting MAC operations with XNOR-COUNT operations and using an efficient packing mechanism to introduce data parallelization.

1. Introduction

After Bain[1] and James[2] made a significant progress in devising the theoretical basis of neural networks, many work has been done to model and implement neural networks on computers[3]-[8]. However it became fairly obvious that simple neural network fail to models many applications effectively. Consequently, large neural networks had to be introduced to approximate useful functions.

Deep Neural Networks (DNNs), which are neural networks (NNs) consisting of many layers, are state of the art machine learning models for a variety of applications including vision and speech tasks. Embedded devices are attractive targets for machine learning applications as they are often connected to sensors, such as cameras and microphones that constantly gather information about the local environment. Currently, for sensor data to be utilized in such applications, end devices transmit captured data in a streaming fashion to the cloud, which then performs prediction using a model such as a DNN. Ideally, we would leverage DNNs to run on the end devices, by performing prediction directly on streaming sensor data and sending the prediction results, instead of the sensor data, to the cloud. However, the majority of these devices are severely constrained in processing power, available memory, and battery power, making it challenging to run inference directly. Even in environments equipped with the necessary resources speeding up a DNN can greatly improve the users experience when dealing with large databases, thus having a compact near state-of-the-art performing DNN is always a welcome improvement [10]. One promising effort is training networks whose weights can be transformed into some quantized representations with a minimal loss of performance [11], [12]. Quantized weights are used in the feedforward step at every training iteration, so that the trained weights are robust enough to the known quantization noise caused by a limited precision. It was shown that 10 and 12 bits are enough to represent gradients and storing weights for implementing a state-of-the-art network. Nevertheless, those quantized DNNs still need to employ arithmetic operations such as multiplication and addition, on fixed-point values. Thus even though they are faster than floating point, they still require relatively complex logic, and have a significant power consumption.

This work demonstrates our effort to accelerate inference on a simple open-source RISC-V based system by migrating pre-trained probabilistic binary neural network onto the RISC-V platform with the IBEX core at the heart of it.

2. Background

2.1 Binary Neural Network

One mechanism for reducing the memory and computational requirements of Neural Networks is reducing the bit-width of the parameters and activations. As mentioned above this can be achieved either during training [13]; [14] or using post-training mechanism [15], [16]. In the most extreme case we can use a single bit to represent a single weight or activation elements. Binary neural networks are advantageous in reducing the memory requirements which ideally could be up to 32x, and feedforward operations could be carried out using an XNOR gate and a simple bit-counting logic producing upto 58x speedup on CPU [17].

[18] Introduces probabilistic training method for neural network with binary weights and binary activations. In this work, we use this framework to train our binary convolutional layers and floating point fully connected layers network and import the binary convolutional weights and parameters to the RISC-V system for inference. We use XNOR-bitcount to replace MAC operation and introduce a packing mechanism to our core to realize XNOR-bitcount, MAC binary domain equivalence as well as introduce data parallelism.

.2 IBEX Core

Ibex is an open-source, small and efficient, 32-bit, in-order RISC-V core with a 2-3 stage-pipeline that implements RV32IMCB instruction set architecture. The core does provide any mechanism for data parallelization, which impedes any possibility of using it for resource intensive parallel applications. [19]

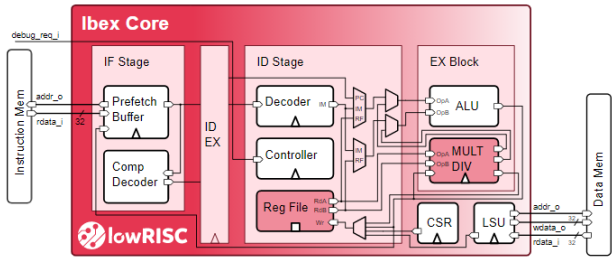


Figure 1: Microarcitecture of Ibex core.

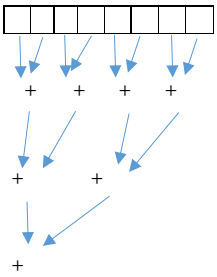
It is highly configurable with a small area ranging from 17.44-58.74(kGE) depending on the configuration and activated features. The core has three multiplier implementations, a single cycle Multiplier, which uses three parallel 17-bit x 17-bit multiplication units and a 34-bit accumulator, as the name suggest it completes a MUL in 1 cycle and MULH in 2 cycles. The tradeoff is larger area and higher power consumption for better performance. The second multiplier implementation is Fast Multi-cycle Multiplier, in a reasonable tradeoff between area and performance, it completes uses a single 17-bit x 17-bit multiplier with a 34-bit accumulator. MUL instruction takes 3 cycles while MULH takes 4.The third one is a slow multiplier, which computes multiply in clog2(operator_b) + 1 cycle for MUL and 33 cycles for MULH using a Baugh-Wooley multiplier.[20]

Recently, an optional Bit manipulation extension has also been added, with XNOR, bit count and packing logics. The parallel prefix counter is of the structure of a Brent-Kung Adder. In the first log2(width)

stages, the sum of the n preceding bit lines is computed for the bit lines at positions 2**n-1 (power-of-two positions) where n denotes the current stage. In stage n=log2(width), the count for position width-1 (the MSB) is finished. For the intermediate values, an inverse adder tree then computes the bit counts for the bit lines at positions m = 2**(n-1) + i*2**(n-2), where i = [1 ... width / 2**(n-1)-1] and n = [log2(width) ... 2]. Thus, at every subsequent stage the result of two previously unconnected sub-trees is summed, starting at the node summing bits [width/2-1 : 0] and [3*width/4-1: width/2] and moving to iteratively sum up all the sub-trees. The inverse adder tree thus features log2(width) - 1 stages the first of these stages is a single addition at position 3*width/4 - 1. It does not interfere with the last stage of the primary adder tree. These stages can thus be folded together, resulting in a total of 2*log2(width)-2 stages [21]. The packing instruction is implemented to concatenate the half of each operand. In this work we use the already made units and modify the packing mechanism to implement data parallelism.

4. Instruction Set Extension

Bit-Count: The original bit count unit design and implementation utilizes the R. Brent, H. T. Kung(1982) parallel adder which has primary and intermediate steps, which cumulatively takes 2*log2(width)-2 stages. However, for our design computing the bit count upto the position width-1, suffices, which will be ready after log2(width) steps. With our design we use width/2 parallel adders to compute the bit count up to width-1.



Firgure 2: Bit count parallel adder

Packing: The default packing instructions implemented are PACKU, which concatenates the upper width/2 bits of the two operands, PACKH, which concatenates the lower width/4 bits of the two operands and pads the upper width/2 bits of the resulting register with zeros and PACK, which concatenats the lower width/2 bits of the operands. If we were to use the above packing implementation we will only be able to XNOR two weights with two activations at a time limiting the performance to 2x parallelism. Additionally, since our weights and activations are all binary utilizing 16 bits to represent a single weight out of the possible two values is a waste of the 15 bits. Our work mitigates this waste and increases parallelism by using only the sign bit to represent a single parameter. The domain of the pre-trained weights is [-1,1], in order to keeps the equivalence between XNOR and Multiplication we should represented -1 with bit value 0, and 1 with bit value 1. In our design we concatenate the upper width-1 bits of our storage register and not gated sign bit of the operand to be packed. This method allows us to pack a matrix up to 32 number of elements. Most convolutional kernels used in deep neural networks are less than 5x5, thus 32 bit registers are large enough to parallelize 25 MAC operations into a two cycle XNOR-Bit Count operation on IBEX core.

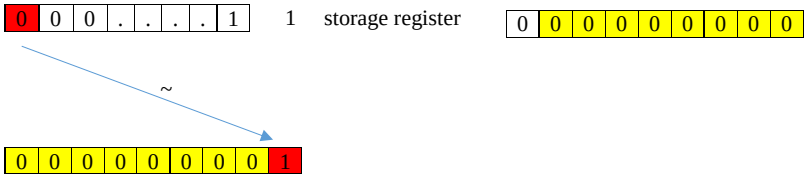


Figure 3: Sign bit based bit packing

FPU is not incorporated in the design of the ibex core, hence we need to carry out pre-processing the inputs outside of the core and load the binarized input to the system's memory unit.

5. Software

Our pre-trained binary neural network gives us kernels whose domain is restricted to [-1, 1]. However the values are stored in 32 bit words that need to be preprocessed and packed. Packing can be done on the core, however we need to load the integer values to our memory which would go against the motivation of our work. Hence, the kernels need to be packed and loaded to our memory in advance, or the kernel values can be deleted from the system's memory once packing is done with a single traversal.

In our current implementation, we are packing a single kernel into a single 32 bit word. This results in (size_of_kernel)x less memory usage. Another important consideration during packing is initialization of the storage 32-bit word. Initially, the 32- bit word for the input activations and kernel have to be of opposing binary values, so that the xnor of the unused bits are zeros. This is necessary because in most cases we would not be using all of the bit in a word. Thus, when counting a 32 bit word resulting from XNORing the activation and kernel, we need to suppress the unused bits from affecting the result of the count operation.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Int a= Input storing byte before packing

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Int b= Kernel storing byte before packing

1	-1
1	1

Int a_[2][2]=Input activation

-1	1
-1	1

Int b_[2][2]=Kernel

Int c_= a_ conv b_ = -2

| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

Int a=Input storing byte after packing a 2x2 input activation

| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

Int b=Kernel storing byte after packing 2x2 kernel

Int c= a XNOR b

0	0	0	0	0	0	0	1

Count(c)= 1 this indicates we have 1 one's and 3 zeros, -1 is a majority by 2 hence we can say the result is -2.

The output from the count operation is used in 2 ways depending on the type of convolution layer. If the convolution layer is followed by other convolution layers we binerize the bit count output by using a threshold value which is kernel_size/2. If the bit count result is greater than the threshold our output activation is 1 since the MAC equivalent would have resulted a positive value, else if the bit count output is less than the threshold the output activation is set to 0 because the MAC equivalent would have resulted in a negative value. In the case that, the layer following the convolution is a fully connected network with floating point operations, the output activation is set as (number of ones – number of zeros), because we would need the exact value of the MAC operation not only the sign.

6. Evaluation

Commented [JK3]: We need implementation results here (synthesis results on FPGA, energy consumption)

The evaluation is implemented by simulating a simple system composed of: an ibex core, single memory for instruction and data, a basic peripheral to write ASCII output to a file and halt simulation form software, a basic timer peripheral capable of generating interrupts based on the RISC-V machine timer registers and a software framework to build simple programs for it, on verilator. We trained a neural network with 2 convolutional layers and 2 fully connected layer on MNIST externally and saved the parameters onto the simple IBEX core based system's memory. We also, pre-processed the input with normalization and sign function and loaded it to the RISC-V system. Next we made a baseline C inference program that uses MAC and an inference program that uses XNOR-Count and our custom packing mechanism. The output from the two convolutions is computed externally, due to the lack of FPU on our system.

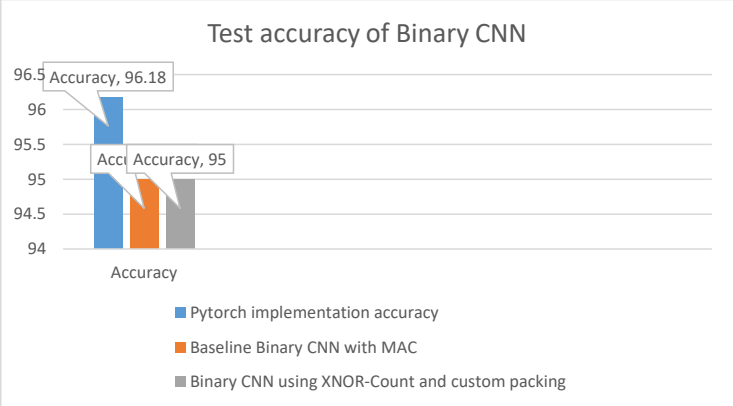


Figure 4: Test accuracy for MAC based and XNOR-bitCount based implementations of Binary CNN.

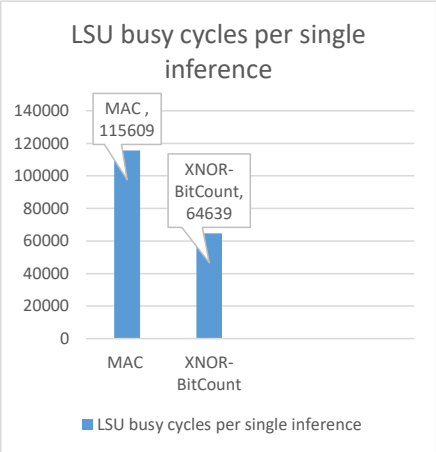


Figure 5: LSU busy cycles per single inference

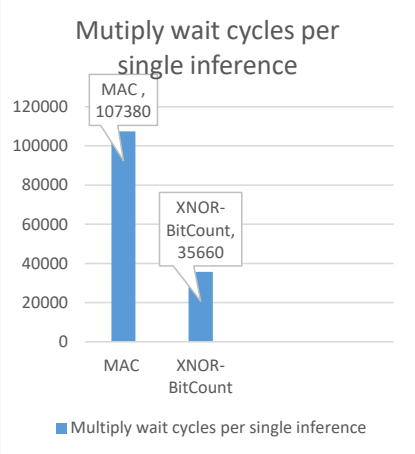


Figure 6: Multiply wait cycles per single inference

Theoretically, since we have to fetch one activation value and one element of a kernel before each MAC operation, by pre-packing the weights we would expect a maximum LSU workload reduction of 50%. Our work results in 44% decrease in the LSU Busy cycles. This is mainly due to reduction in memory access for weights. The compression of each kernel into a single 32 bit word reduces the frequency of memory access by providing a means to fetch kernel size per each load compared to MAC implementation where each element of a kernel has to be fetched separately. However, there is still considerable LSU load because we are representing each value of the output activation matrix using a 32 bit word. The multiplication wait cycles have also been reduced by 66.7% by replacing multiplication and addition with XNOR-BitCount. The multiplication waits unaccounted for by the XNOR replacement are divisions used for index calculation during max-pooling.

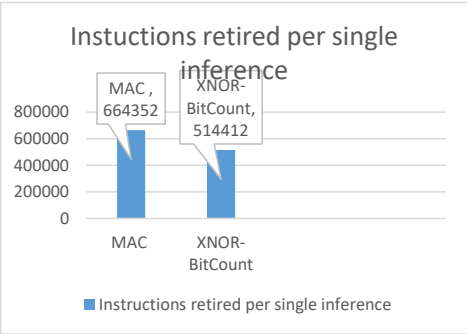


Figure 7: Instructions retired per single inference

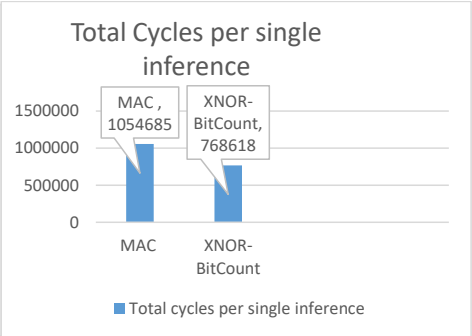


Figure 8: Total cycles per single inference

Compared to MAC implemented Binary CNN, the XNOR-Binary CNN albeit not significant has lower code density which saves memory usage. Our work reduces the overall cycles needed by about 26% compared to the baseline MAC based implementation. Even though there is a significant performance improvement in terms of LSU and Multiplication wait cycles, it is not reflected in the total cycles. The bottleneck is an increase in the number of conditional branches due to packing input activation sub-matrices before each XNOR operation. Reuse issue and subsequent pooling layers prevent us from packing the output activation to their respective input bit positions on the fly without the need to produce the intermediate activation arrays.

Our work also reduces weight memory usage by kernel size. Once we go through the weights and pack each kernel into a single 32 bit word, we can delete the imported weights and create space for future usage.

7. Conclusion

This work has shown that Binary neural networks are strong candidates in the quest to push inference towards edge devices. It also presents future opportunities to explore methods of adding more parallelization, designing input activation access mechanisms that accounts for reuse issues to pack the activation matrices in a single traversal, and explore algorithms to pack intermediate activations on the fly without the need for additional storage. Subsequently, we can augment fixed point FPU to compute quantized fully connected layers on the platform.

References

[1] A. Bain, Mind and Body the Theories of Their Relation by Alexander Bain. Henry S. King & Company, 1873.
[2] W. James, The principles of psychology. Read Books Ltd, 2013.
[3] C. S. Sherrington, "Experiments in examination of the peripheral distribution of the fibres of the posterior roots of some spinal nerves. Part ii," Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character, vol. 190, pp. 45–186, 1898.
[4] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115–133, 1943.
[5] D. O. Hebb, "The organization of behavior: A neurophysiological approach," 1949.
[6] B. Farley andW. Clark, "Simulation of self-organizing systems by digital computer," Transactions of the IRE Professional Group on Information Theory, vol. 4, no. 4, pp. 76–84, 1954.
[7] N. Rochester, J. Holland, L. Haibt, and W. Duda, "Tests on a cell assembly theory of the action of the brain, using a large digital computer," IRE Transactions on information Theory, vol. 2, no. 3, pp. 80–93, 1956.
[8] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review, vol. 65, no. 6, p. 386, 1958.
[9] Bradley McDanel, Surat teerapittayanon, H.T. Kung, "Embedded Binarized Neural Networks", 2017
[10] Salakhutdinov, R. and Hinton, G. "Semantic hashing" International Journal of Approximate Reasoning, 50(7):969 – 978, 2009.
[11] Fiesler, E., Choudry, A., and Caulfield, H. J. "Weight discretization paradigm for optical neural networks." In The Hague'90, 12-16 April, pp. 164–173. International Society for Optics and Photonics, 1990.
[12] Hwang, K. and Sung, W. "Fixed-point feedforward deep neural network design using weights +1, 0, and -1." In 2014 IEEE Workshop on Signal Processing Systems (SiPS), Oct 2014.
[13] Karen Ullrich, Edward Meeds, and MaxWelling. "Soft weight-sharing for neural network compression." arXiv preprint arXiv:1702.04008, 2017.
[14] Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. "Variational network Quantization". In International Conference on Learning Representations, 2018
[15] Christos Louizos, Karen Ullrich, and Max Welling. "Bayesian compression for deep learning. In Advances in Neural Information Processing Systems", pp. 3290–3300, 2017.
[16] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149, 2015.
[17] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." In European Conference on Computer Vision, pp. 525–542. Springer, 2016.
[18] Oran Shayer, Dan Levi, and Ethan Fetaya. "Learning discrete weights using the local reparameterization trick." In International Conference on Learning Representations, 2018
[19] Schiavone, Pasquale Davide, et al. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications." 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS 2017)
[20] [5] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," IEEE Trans Comp., vol. C-22, no. 12, pp. 1045-1047, Dec. 1973.
[21] R. Brent, H. T. Kung, "A Regular Layout for Parallel Adders",(1982).
[22] Jorn W.T. Peters, Max Welling, "Probabilistic Binary Neural Networks", arXiv preprint arXiv:1809.03368, 2018
[23] McDanel, Bradley et al. "Embedded Binarized Neural Networks." EWSN (2017).
[24] Minje Kim, Paris Smaragdis, "Bitwise Neural Networks". ICML Workshop on Resource-Efficient Machine Learning, Lille, France, Jul. 6-11, 2015
[25] Zhang, Jianhao, et al. "dabnn: A super fast inference framework for binary neural networks on arm devices." Proceedings of the 27th ACM International Conference on Multimedia. 2019.
[26] ervan, Branimir & Knezovic, Josip. (2019). Deep Learning Accelerator on Programmable Heterogeneous System with RISC-V Processor.