Design framework for an energy-efficient binary convolutional neural network accelerator based on nonvolatile logic

Daisuke Suzuki\textsuperscript{1a)}, Takahiro Oka\textsuperscript{2,3}, Akira Tamakoshi\textsuperscript{3}, Yasuhiro Takako\textsuperscript{3}, and Takahiro Hanyu\textsuperscript{3}

\textsuperscript{1} School of Computer Science and Engineering, The University of Aizu, Kami-Iawase-90, Ikki-machi, Aizu-Wakamatsu, Fukushima 965-8580 Japan
\textsuperscript{2} Graduate School of Engineering, Tohoku University, 6-6, Aramaki Aza Aoba, Aoba-ku, Sendai, Miyagi 980-8579 Japan
\textsuperscript{3} Research Institute of Electrical Communication, Tohoku University, 2-1-1, Katahira, Aoba-ku, Sendai, Miyagi 980-8577 Japan

\textsuperscript{a)} daisuke@u-aizu.ac.jp

Received April 16, 2021; Revised June 27, 2021; Published October 1, 2021

Abstract: Convolutional neural network (CNN) accelerators, particularly binarized CNN (BCNN) accelerators have proven to be effective for several artificial-intelligence-oriented several applications; however, their energy efficiency should be further improved for edge applications. In this paper, a design framework for an energy-efficient BCNN accelerator based on nonvolatile logic is presented. Designing BCNN accelerators using nonvolatile logic allows for the accelerators to exhibit a massively parallel and ultra-low standby power capability. Thus, a new design can be realized for accelerators that is different from that of conventional accelerators based solely on CMOS. Considering this, we discuss a concrete design considerations of nonvolatile BCNN accelerators. In fact, a systematic design flow of the nonvolatile BCNN is established by combining Vivado HLS and standard electronic design automation tools. As a typical design example, a BCNN accelerator for inferring 32 × 32 pixel MNIST dataset is designed using a 65-nm CMOS technology. By the logic-synthesis result, the proposed BCNN accelerator is estimated to consume 94.2% lower power than that of a conventional BCNN accelerator when the frame rate is 30 frames per second.

Key Words: BCNN, HLS, EDA, ASIC, FPGA

1. Introduction

The internet of things (IoT) where smart devices communicate each other via the Internet has been continuously growing. Owing to this, edge computing where data is collected and processed locally at each IoT device has become an emerging technology [1, 2]. In such edge computing, artificial
intelligence (AI) plays a significant role. A convolutional neural network (CNN) is regarded to be prevalent in modern AI and has achieved remarkable breakthroughs in applications such as image classification and speech recognition. Despite the advantageous features of CNNs, their computational intensity is generally high, hindering their application to power and resource-limited IoT devices. A potential manner in which AI can be implemented in IoT devices is via a binarized CNN (BCNN) [3], where network parameters are expressed in a binary format with an inference accuracy comparable to that of the original CNN. Several researches have been conducted on implementing BCNN accelerators using various hardware platforms such as GPUs [4], ASICs [5–16], and FPGAs [17–24].

On the contrary, standby power consumption due to the leakage current is a critical issue for conventional CMOS-only-based logic LSIs. Because the storage element of the conventional CMOS-only-based LSI is volatile, continuous power supply is required during its operation to maintain the stored information. Consequently, the number of idle components should be minimized for the CMOS-only-based BCNN accelerator design. To solve the above standby-power problem of the CMOS-only-based LSI, various nonvolatile logic LSIs have been proposed [26–34]. Since all the data are stored in the nonvolatile devices, a power-gating technique can be fully utilized and standby power consumption of idle function blocks is eliminated. This feature generates a new design space for the BCNN accelerator, which is different from that for the conventional CMOS-only-based BCNN accelerator. In this paper, making of the nonvolatile BCNN accelerator is discussed. The key concept of designing a nonvolatile BCNN accelerator is sharing processing elements (PEs) for BCNN computations among several layers, which is referred to as a single computation engine architecture. Since the number of PEs required for BCNN computations depends on each layer, several unused PEs exist. Thus, massively parallel computing and low-standby power consumption can be achieved in the nonvolatile BCNN accelerator by applying the power-gating technique to the unused PEs. To design and evaluate a nonvolatile BCNN accelerator, a design framework using high-level synthesis (HLS) and standard electronic design automation (EDA) tools is also presented. As a typical example, a BCNN accelerator for inferring a 32 × 32 pixel MNIST dataset is designed using a 65-nm CMOS technology, and a power reduction of 94.2% compared with that for the conventional BCNN accelerator was achieved when the frame rate is 30 frames per seconds (FPS).

The remainder of this paper is organized as follows. In Section 2, the overview of the BCNN, the nonvolatile logic, and BCNN accelerators are introduced with related works. In Section 3, we discuss the design considerations for the BCNN accelerator, whose key concept is similar to both ASICs and FPGAs. In Section 4, an EDA tool flow for the BCNN accelerator for the evaluation is described. In Section 5, a concrete design example of a BCNN accelerator and performance evaluation results are discussed. Finally, conclusion and future prospects are described in Section 6.

2. Overview of BCNN accelerator and nonvolatile logic

2.1 Overview of BCNN

The CNN comprises a convolution of feature maps and weights. For software applications, the 32-bit floating point (FP32) data representation is commonly used; however, the hardware cost of the FP32 multiplier is extremely high. Therefore, various methods have been introduced to quantify (e.g., INT8: 8-bit integer) feature maps and weights without substantially reducing the accuracy of the CNN.

A BCNN, in which both the feature maps and weights are represented by binarized data format, can significantly reduce the model size and computational complexity with only a marginal loss in accuracy [3]. A typical BCNN model contains convolutional, pooling, and fully connected layers as shown in Fig. 1(a). The first few layers generally capture regional information such as edges and curves, and the last few layers interpret these low-level features into high-level abstractions with the posterior probability assigned for classification. The convolutional layer is the core layer of the BCNN. The input of the convolutional layer is $N$ feature maps with a size of $W \times H$ where $W$ and $H$ correspond to the width and height of an input feature map, respectively. Each filter has a size of $K \times K \times N$, where $K$ is the width and height of the reception field. From the spatial convolution along the first and second dimensions of the input feature maps with the 3D filter, $M$ output feature
maps with a size of $C \times R$ are obtained where $C$ and $R$ correspond to the width and height of the output feature map, respectively. Figure 1(b) shows the pseudo code of a convolutional layer whose operation entails multiplications and additions. In the BCNN, the input feature map (ifm) of Layer 0 is non-binary while that of the other layers are binary. If both ifm and weight are expressed in a binary data format, the convolutional multiply-accumulate operation is replaced by XNOR and bit-count operations [3]. The pooling layer performs sub-sampling across a contiguous region on the output feature map of the convolutional layer and reduces the number of trainable parameters in the network.

In recent years, various BCNN accelerators have been proposed such as GPUs, ASICs, and FPGAs. Table I summarizes the comparison of hardware platforms for the BCNN accelerator. While the GPU exhibits a powerful computation capability, it consumes a considerable amount of power and its area cost is high. The ASIC is fairly advantageous in terms of energy efficiency (i.e. TOPS/W: tera operations per second per watt) but lacks turn-around time (TAT) and hardware flexibility. Currently, various derivations of CNN exist, and the hardware architecture depends on the application of the CNN; therefore, hardware flexibility is important for universal edge-AI hardware. Compared with GPUs, FPGAs can perform BCNN acceleration with less area and power. The reconfigurability of the FPGA is suitable for implementing various CNN accelerators with a short TAT. From this
Table I. Comparison of hardware platforms for the BCNN accelerator.

| Platform | TOPS | TOPS/W | Size | TAT     | HW-Flexibility |
|----------|------|--------|------|---------|----------------|
| GPU      | +++  | +++    | +    | ++++    | N/A            |
| ASIC     | +++  | ++++   | ++++ | +       | ++             |
| FPGA     | ++   | +++    | +++  | ++++    | +++            |

TOPS: Tera operations  TOPS/W: TOPS per Watt  TAT: Turn around time

From the perspective, the FPGA is a suitable hardware platform for the BCNN accelerator.

2.2 Nonvolatile logic

As described in the Introduction section, the standby power consumption due to the leakage current is a critical issue for conventional CMOS-only-based logic LSIs. The combination of the power-gating technique with nonvolatile logic is a fundamental solution to this problem. As shown in Fig. 2, the power supply to the unused circuit components is temporarily turned off during the power-gating mode, thus eliminating wasted standby power consumption. In the nonvolatile logic LSI, the temporary data are stored in a nonvolatile flip-flop (NV-FF) [35–40], backed up to the nonvolatile device, and recalled just after power supply is commenced on. The nonvolatile memory does not require backup and recall operations because data is directly stored and loaded to/from the nonvolatile device. In this manner, the power-gating technique is performed without losing internal data.

![Fig. 2. Impact of nonvolatile logic circuit with power-gating technique.](image)

Figure 3(a) shows the device structure of the magnetic tunnel junction (MTJ) device that comprises of two ferromagnetic layers separated by a tunnel barrier; the magnetization of the pinned layer (PL) is fixed, while that of the free layer (FL) is free to rotate [41–43]. The MTJ device is advantageous in that it is compatible with the CMOS process as it forms them in the back-end of the line. This feature helps achieve low cost and high performance, because the CMOS characteristics and Cu/low-k
Fig. 3. MTJ device: (a) device structure, (b) schematic symbol, and (c) R-I characteristic.

interconnect processes are unchanged. Moreover, because the MTJ device is stacked over the CMOS circuit plane, the effective area of the MTJ device itself is negligible [27]. By controlling the direction of the magnetization of the FL with respect to the PL with a bidirectional current, the MTJ device is configured to a low-resistance state ($R_L$, the corresponding state is $Y = 0$) or a high-resistance state ($R_H$, the corresponding state is $Y = 1$). Thus, the MTJ device is considered to be a variable resistor, wherein the resistance is programmed by the write current, as shown in Fig. 3(b). Figure 3(c) shows the R-I characteristic of the MTJ device. Now, let us consider the initial value of $R_{MTJ}$ is $R_H$. If the current which exceeds $I_{W0}$ is applied from the UE to the LE, $R_{MTJ}$ changes to $R_L$ (thus, $Y$ changes to 0). In contrast, if $R_{MTJ}$ is $R_L$ and the current is applied from the UE to the LE, $Y$ does not change. To change $Y$ from 0 to 1, the current that exceeds $I_{W1}$ must be applied from the LE to the UE.

Figure 4(a) shows a simplified block diagram of the NV-FF. A nonvolatile slave latch is connected to the volatile master latch [33]. As shown in Fig. 4(b), The MTJ-based nonvolatile slave latch is implemented by adding an active load and four write current sources to the standard CMOS latch. Signals namely WB and LB are low active. WB is used for storing the state of CMOS latch to MTJ device and LB is used for recalling the state of the MTJ device to the CMOS latch. During the normal mode, both LB and WB are set to “1”, and the slave latch operates in a manner similar to a CMOS latch. Before the power supply is turned off, the WB signal becomes “0”, and the write current driven by the write current sources flows through two MTJ devices in accordance with the values of $q$ and $q_b$. Then, the temporal state of the CMOS FF is stored in the two MTJ devices in a complementary fashion. After the power supply is turned on, the LB signal becomes “0” and the stored data is recalled to the CMOS latch by sensing the current difference between the two MTJ devices. The active load is used for boosting the current difference. In this manner, the power
supply can be turned on/off immediately without losing the internal state. A simple way to realize a nonvolatile logic LSI is to replace all the CMOS-based FFs with NV-FFs. The use of a nonvolatile logic-in-memory structure further reduces the area cost by merging the storage and logic functions into the same circuit topology [27].

3. Making of nonvolatile BCNN accelerator

As described in Section 2, the main part of the BCNN operation is the iteration of the XNOR and bit-count operations. Three possible architectures for the BCNN accelerator exist. Figure 5(a) shows a streaming architecture that typically comprises of one distinct hardware block for each layer of the target BCNN, where each block is optimized separately to exploit the parallelism of its layer. All the heterogeneous blocks are chained to form a pipeline. The data proceed through different parts
of the neural network as they are streamed through the architecture. In this manner, this design approach exploits the parallelism between layers via pipelining and enables their concurrent execution. Nevertheless, the increased efficiency comes with BCNN-dependent customization and optimization, which is time consuming and requires a high design cost. Figure 5(b) shows a single-computation engine architecture whose design approach favors flexibility over customization. Such an architecture comprises a single computation engine typically in the form of an array of PEs that executes the BCNN layers sequentially. A controller is required for scheduling and controlling the layer-dependent convolutional operation. This design paradigm consists of a fixed architectural template that can be scaled based on the input BCNN and available hardware resources. With this scheme, each BCNN corresponds to a different sequence of microinstructions that are executable by the hardware. Despite the flexibility gains, inefficiencies are introduced due to control mechanisms that resemble those of a processor. Moreover, the one-size-fits-all approach can lead to high variability in the achieved performance across CNNs with different workload characteristics. Recently, processing-in-memory (PIM) architecture has been focused on owing to its highly energy-efficient computing capability [12–16]. As shown in Fig. 5(c), the key idea of PIM is to embed logic units within a memory cell array as well as sense amplifiers to process data by leveraging the inherent parallel computing mechanism and exploiting the large internal memory bandwidth. This could lead to remarkable savings in the off-chip data communication energy and latency. In addition, utilization of the intermediate current signal from the memory element for analog summation is suitable for energy-efficient BCNN operation. On the contrary, a full-custom design is required for the PIM-based accelerator due to its complex mixed-signal features.

Let us consider which architecture is suitable for the nonvolatile BCNN accelerator. In the streaming architecture, each block is optimized separately to exploit the parallelism of its own layer. Thus, the power-gating technique can be applied only to a few idle hardware components. In contrast, several opportunities exist for the single-computation-engine architecture, wherein the power-gating technique can be applied because the number of active PEs is dependent on the processed layer, as shown in Fig. 6. Thus, the single-computation-engine architecture is more suitable than the streaming architecture for the nonvolatile BCNN accelerator. The PIM-based architecture is another possible approach for implementing the nonvolatile BCNN accelerator. However, this approach is strongly dependent on the type of memory element used, and a custom design is required. Therefore, this study focuses on single-computation-engine architecture, and a comparison with the PIM-based approach will be realized in the future.

Figure 7 shows an example of the single-computation-engine BCNN accelerator. The BCNN is pre-trained and the weight parameters are stored in the weight memory.

For the BCNN computation, one corresponding input image is firstly processed by a cluster of \( n_{p0} \) PEs (we call this cluster “C-PE0”) namely PE0(0), PE0(1), ..., PE0\((n_{p0} - 1)\) and binary output feature
maps are generated. Then, layer-by-layer binary convolutional operations are performed by a cluster of $n_{pb}$ PEs (we call this cluster “C-PEb”) namely PEb(0), PEb(1), ..., PEb($n_{pb}$-1). The average pooling unit and intermediate buffer interfaces feature map transmission between two adjacent layers. The output of the last BCNN layer is then proceeded by the output determiner, and the inference result is obtained. As described above, several possibilities exist for applying the power-gating technique; this is discussed later.

4. Chip design flow using Vivado HLS and standard EDA tools

An EDA tool flow is indispensable for designing a practical-scale BCNN accelerator. The use of HLS, which translates high-level languages such as C/C++ into hardware description language (HDL) is quite useful for designing BCNN accelerator. Vivado HLS by Xilinx is one of the most prevalent HLS tools, whose design flow is shown in Fig. 8. We utilize the Verilog code generated by Vivado HLS for both FPGA and ASIC implementation. Figure 9 shows the design flow of the BCNN accelerator. First, an RTL simulation of the Verilog code with the SRAM IP package is performed using Synopsys VCS for functional verification. After the RTL simulation, logic synthesis is performed using Synopsys Design Compiler, and the Verilog code is translated into a gate-level netlist. Simultaneously, the
memory compiler generates SRAM macros using information from HLS result. According to several parameters (array size, bit width, number of I/O ports, etc.), the memory compiler automatically generates a physical layout of the memory macro and its corresponding netlist. By translating the text-level information into a gate-level netlist, the delay information in the netlist is extracted. A post-synthesis simulation is performed to validate the correctness of the synthesized netlist including the delay information. Then, the synthesized gate-level netlist is translated into the physical layout (i.e., place and route) by Cadence Innovus. Innovus also generates delay information which includes the effect of the parasitic components extracted from the physical layout. Finally, a post-layout simulation is performed to validate the chip, including the physical information. This tool flow itself normal. The important point is that nonvolatile BCNN accelerator can be designed by using the same design flow by adding nonvolatile logic circuit IPs. In fact, an EDA tool flow by using MTJ-based logic circuit IPs has already been established and some test chips are fabricated such as a microcontroller unit, an FPGA, etc. [33, 44].

5. Design example and evaluation
A typical BCNN accelerator is designed using a 65-nm CMOS technology, and the design procedure is presented in this section. Table II summarizes the design environment.

Because the network size is small and comprehensive, we apply the BCNN for digit recognition
using 32 × 32 pixel MNIST datasets. Figure 10 shows a BCNN that comprises four convolutional layers for the chip implementation. Parameters $P_N$ and $P_M$ are loop unrolling factors for $N$ and $M$, respectively. The BCNN is trained in advance by PyTorch, and the inference accuracy was obtained as 82%. After the training is completed and weights are determined, the C code for the BCNN is written.

Figure 11 shows the architecture of the BCNN accelerator for the implementation. As described in Section 3, the core block comprises the C-PE0, the C-PEb, the intermediate buffer, the average pooling unit, and the output determiner. The C-PE0 is used for processing for the Layer 0, thus, input images in 8-bit integer format are converted to binary feature maps. By the loop unrolling, $P_M = 6$ PEs (PE0[0], PE0[1], and so on) operates in parallel. At each PE, $P_N = 6$ multiply and accumulate operations are performed in parallel to calculate output feature maps. The calculated output feature maps are stored into intermediate buffer via average-pooling unit. Then, BCNN operations for Layer 1, Layer 2, and Layer 3 are serially performed by using the C-PEb. Similar to the C-PE0, $P_N = 6$ XNOR and bit-count operations are performed in parallel to calculate output feature map at each $P_M$ PEs (PEb[0], PEb[1], and so on). The outputs of the C-PEb are also stored into the intermediate buffer via the average-pooling unit. Finally, the outputs of the C-PEb corresponding to Layer 3 are sent to the output determiner, and the recognition result and completion flag as est[3:0] and done are generated, respectively. After the HLS of the C code, the core block is synthesized by Design Compiler, and the input image and weight memories are compiled by the memory compiler.

Figure 12 shows the generated layout of the BCNN accelerator with I/O pads by Innovus. In this case, the hierarchy of the accelerator is flattened, optimally placed, and routed. Memory macros are placed at the left bottom and right side of the layout, respectively.
To confirm the basic behavior of the BCNN accelerator, three types of simulations were performed: a logic simulation with no delay information; a gate-level simulation with delay information obtained from the logic synthesis; and a post-layout simulation with delay information obtained from the physical layout. Figure 13 shows the simulated waveforms of the BCNN accelerator. We can see that an input image that corresponds to “7” and an operation trigger, namely start, are applied to the core block, and the inference result and operation completion flag, namely done, appear. We can also confirm that the delay information is reflected, and the delay of the post-layout simulation is the longest (3.0 ns for est[3:0] and 3.2 ns for done).

Currently, we have not designed an MTJ-based NV-FF in this 65-nm CMOS technology. However, we can roughly estimate the impact of the power gating technique in the nonvolatile BCNN accelerator because of the idle hardware component. Table III summarizes the energy consumption for inferring one input image. The power consumption data of the C-PEb0 and C-PEb are obtained from the Design Compiler. By using this data, we can calculate active energy and standby energy for computing each layer. The difference between the volatile BCNN accelerator and the nonvolatile one is whether FF is
Simulated waveforms of the BCNN accelerator.

Table III. Summary of energy consumption for inferring one input image.
The power consumption data of the C-PEb0 and C-PEb are obtained from the Design Compiler. The yellow cells indicate that these wasted power can be eliminated.

| Layer | Latency [ns] | C-PE0 Active energy [nJ] | C-PE0 Standby energy [nJ] | C-PEb Active energy [nJ] | C-PEb Standby energy [nJ] | Total Active energy [nJ] | Total Standby energy [nJ] |
|-------|--------------|--------------------------|---------------------------|--------------------------|---------------------------|-------------------------|--------------------------|
| Layer 0 | 39,200       | 79.2                     | 8.49                      | 0                        | 2.00                      | 79.2                    | 10.49                    |
| Layer 1 | 15,000       | 0                        | **3.25**                  | 2.97                     | 0.77                      | 2.97                    | 4.02                     |
| Layer 2 | 3,000        | 0                        | **0.65**                  | 0.59                     | 0.15                      | 0.59                    | **0.80**                 |
| Layer 3 | 400          | 0                        | **0.09**                  | 0.08                     | 0.02                      | 0.08                    | **0.03**                 |
| Total   | 57,600       | 79.2                     | **12.4**                  | 3.64                     | 2.94                      | **82.84**               | **15.34**                |

Wasted standby energy: **3.25 + 0.65 + 0.09 + 2.00 = 5.99 [nJ]**

volatile or nonvolatile. The performance of the NV-FF is almost same as that of the volatile FF. The power overhead of additional components is small and is neglected in this evaluation. So, dynamic power becomes same. For computing the Layer 0, the C-PEb is not used while the C-PE0 is not used for computing the Layer 1, the Layer2, and the Layer 3. These wasted energy consumption data are shown as yellow cells in Table III, that can be reduced to zero by the use of the power-gating technique. As a result, 5.99 nJ of standby power energy can be saved by the use of power-gating technique.

Let us estimate the impact of standby energy reduction under more advanced CMOS technology where leakage current exponentially increases against the transistor size. In Fig. 3(c) of Ref. [45], relationship between the dynamic power per area and gate length and that of leakage power is summarized. By using this figure, we can roughly approximate the active power and standby power of the 28-nm CMOS technology are 1.16x and 8.35x higher than those of the 65-nm CMOS technology. Therefore, standby power energy becomes much larger than active energy consumption in the 28-nm
CMOS technology as shown in Fig. 14. In this case, the use of the nonvolatile BCNN accelerator is more effective for reducing standby energy consumption.

Then, let us discuss relationship between power reduction ratio by the use of power gating technique and FPS in 65nm CMOS technology. One possible application for digit recognition is number (license)
plate recognition. The number plate of composed of several digits, and number of digits must be processed per second depends on the situation (type of road, traffic volume, etc.). If we define the frame rate, the power consumption for inferring one input image, the standby power, and the inference latency as $f$, $P_{inf}$, $P_{stb}$, and $t_{inf}$, respectively, the total power consumption ($P_{total}$) is expressed as follows.

$$P_{total} = f \times \{ P_{inf} \times t_{inf} + P_{stb} \times (1/f - t_{inf}) \}$$

(1)

In the case of nonvolatile BCNN accelerator, $P_{total}$ becomes $f \times P_{inf} \times t_{inf}$. By using this equations the relationship is plotted as shown in Fig. 15. The power saving becomes higher as the FPS becomes lower because almost of the time is idle. As a result, the proposed BCNN accelerator is demonstrated to consume 94.2% lower power than that of a conventional BCNN accelerator when the frame rate is 30 FPS. Note that, this evaluation itself is to simply estimate the impact of power-gating technique and the detailed evaluation with a practical application is a future work. In addition, while power-gating technique can be applied to the volatile system, there will be power overhead because SRAM memories cannot avoid meta-stable state just after power on.

6. Conclusion

In this paper, the design considerations of a nonvolatile BCNN accelerator and its design framework are discussed. A BCNN accelerator for hand-written digit inference is designed using 65-nm CMOS technology, and the impact of the power gating technique with the nonvolatile BCNN accelerator is demonstrated. Because the power-gating technique can be applied in several manner, design spaces for nonvolatile BCNN accelerators should be further explored. For example, the use of pipeline architecture is possible candidate. The use of pipeline also makes it possible to enhance the throughput while the overhead of the controller and registers must be considered. Further, the design environment for the nonvolatile BCNN accelerator should be enhanced by adding nonvolatile logic circuit IPs such as FFs, memories, and other hardware macros. It is also important to explore practical applications that are suitable for the proposed BCNN accelerator and evaluate their impact.

Acknowledgments

This research is supported by JST-CREST(JPMJCR19K3), JST-OPERA, JSPS KAKENHI Grant No.JP16H06300, CIES cons. program, and VDEC. We also thank to Silicon Artist Technology Co. for the technical assistance.

References

[1] W. Shi et al., “Edge computing: vision and challenges,” IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, October 2016.
[2] N. Abbas et al., “Mobile edge computing: a survey,” IEEE Internet of Things Journal, vol. 5, no. 1, pp. 450–465, February 2018.
[3] M. Courbariaux and Y. Bengio, “BinaryNet: training deep neural net-works with weights and activations constrained to +1 or -1,” ArXiv:1602.02830, 2016.
[4] K. Guo et al., “Neural network accelerator comparison,” NICS Lab of Tsinghua University. http://nicsefe.ee.tsinghua.edu.cn/projects/neural-network-accelerator/
[5] A.A. Bahou et al., “XNORBIN: A 95 TOp/s/W Hardware Accelerator for Binary Convolutional Neural Networks,” Proc. IEEE Symp. Low-Power and High-Speed Chips (COOL CHIPS), pp. 1–3, April 2018.
[6] B. Liu, S. Chen, Y. Kang, and F. Wu, “An energy-efficient systolic pipeline architecture for binary convolutional neural network,” Proc. Int. Conf. ASIC (ASICON), pp. 1–4, October-November 2019.
[7] L.G. Rocha, D. Biswas, B.-E. Verhoef, S. Bampi, C.V. Hoof, M. Koenijnenburg, M. Verhelst, and N.V. Helleputte, “Binary CorNET: accelerator for HR estimation from wrist-PPG,” IEEE Trans. Biomedical Circuits and Systems, vol. 14, no. 4, pp. 715–726, August 2020.
[8] A. Ardakani, C. Condo, and W.J. Gross, “A convolutional accelerator for neural networks with
binary weights," IEEE Int. Symp. Circuits and Systems (ISCAS), pp. 1–5, May 2018.

[9] F. Conti, P.D. Schiavone, and L. Benini, “XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference,” IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2940–2951, November 2018.

[10] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, “Hyperdrive: A multi-chip systolically scalable binary-weight CNN inference engine,” IEEE J. Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 2, pp. 309–322, June 2019.

[11] S. Kim, J. Lee, S. Kang, J. Lee, and H. Yoo, “A power-efficient CNN accelerator with similar feature skipping for face recognition in mobile devices,” IEEE Trans. Circuits and Systems I: Regular Papers, vol. 67, no. 4, pp. 1181–1193, April 2020.

[12] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, “An always-on 3.8 µJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS,” IEEE J. Solid-State Circuits, vol. 54, no. 1, pp. 1789–1799, June 2019.

[13] H. Valavi, P.J. Ramadge, E. Nestler, and N. Verma, “A 64-tile 2.4-Mb In-memory-computing CNN accelerator employing charge-domain compute,” IEEE J. Solid-State Circuits, vol. 54, no. 6, pp. 1733–1743, June 2020.

[14] S. Angizi, Z. He, A.S. Rakin, and D. Fan, “CMP-PIM: an energy-efficient comparator-based processing-in-memory neural network accelerator,” Proc. Design Automation Conf. (DAC), pp. 1–6, June 2018.

[15] S. Kim, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-S. Seo, “Vesti: Energy-efficient in-memory computing accelerator for deep neural networks,” IEEE Trans. VLSI Syst., vol. 28, no. 1, pp. 48–61, January 2020.

[16] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, “XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks,” IEEE J. Solid-State Circuits, vol. 55, no. 6, pp. 1733–1743, June 2020.

[17] L. Yang, Z. He, and D. Fan, “A fully onchip binarized convolutional neural network FPGA implementation with accurate inference,” Proc. Int. Symp. Low Power Electronics and Design (ISLPED), pp. 1–6, July 2018.

[18] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren, “A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks,” J. Emerg. Technol. Comput. Syst., vol. 14, no. 2, pp. 18:1–18:16, July 2018.

[19] D.T. Nguyen, T.N. Nguyen, H. Kim, and H.-J. Lee, “A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection,” IEEE Trans. VLSI Syst., vol. 27, no. 8, pp. 1861–1873, August 2019.

[20] G. Chen et al., “Stereoengine: An FPGA-based accelerator for real-time high-quality stereo estimation with binary neural network,” IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 4179–4190, November 2020.

[21] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, “FP-BNN: Binarized neural network on FPGA,” Neurocomputing, vol. 275, pp. 1072–1086, January 2018.

[22] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D. Wang, “FBNB: A fully binarized neural network accelerator,” Proc. Int. Conf. Field Programmable Logic and Applications (FPL), pp. 51–54, August 2018.

[23] H. Nakahara, M. Shimoda, and S. Sato, “A Demonstration of FPGA-Based You Only Look Once Version2 (YOLOv2),” Proc. Int. Conf. Field Programmable Logic and Applications (FPL), pp. 457–458, August 2018.

[24] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang, “FracBNN: accurate and FPGA-efficient binary neural networks with fractional activations,” Proc. Int Symp. FPGA, pp. 171–182, February 2021.

[25] S.I. Venieris, A. Kouris, and C.-S. Bouganis, “Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions,” ACM Comput. Surv., vol. 51, no. 3, pp. 56:1–56:39, June 2018.

[26] D. Suzuki, et al., “Fabrication of a 3000-6-input-LUTs embedded and block-level power-gated
nonvolatile FPGA Chip using p-MTJ-based logic-in-memory structure,” *Symp. VLSI Circuits Dig. Tech. Papers*, pp. 172–173, June 2015.

[27] T. Hanyu et al., “Standby-power-free Integrated Circuits Using MTJ-Based VLSI Computing,” *Proc. IEEE*, vol. 104, no. 10, pp. 1844–1863, October 2016.

[28] M. Kozuma, et al., “Subthreshold operation of CAAC-IGZO FPGA by overdriving of programmable routing switch and programmable power switch,” *IEEE Trans. VLSI Syst.*, vol. 25, no. 1, pp. 125–138, January 2017.

[29] F. Su, K. Ma, X. Li, T. Wu, Y. Liu, and V. Narayanan, “Nonvolatile processors: Why is it trending?,” *Proc. Design, Automation & Test in Europe Conf. (DATE)*, pp. 966–971, May 2017.

[30] Z. Wang et al., “A 130nm FeRAM-based parallel recovery nonvolatile SOC for normally-OFF operations with 3.9×faster running speed and 11× higher energy efficiency using fast power-on detection and nonvolatile radio controller,” *Symp. VLSI Circuits*, pp. 336–337, June 2017.

[31] A. Lee et al., “A ReRAM-based nonvolatile flip-flop with self-write-termination scheme for frequent-OFF fast-wake-up nonvolatile processors,” *IEEE J. Solid-State Circuits*, vol. 52, no. 8, pp. 2194–2207, August 2017.

[32] A. Roohi and R.F. DeMara, “NV-clustering: Normally-off computing using non-volatile datapaths,” *IEEE Trans. Computers*, vol. 67, no. 7, pp. 949–959, July 2018.

[33] M. Natsui, et al., “A 47.14-µW 200-MHz MOS/MTJ-hybrid nonvolatile microcontroller unit embedding STT-MRAM and FPGA for IoT applications,” *IEEE J. Solid-State Circuits*, vol. 54, no. 11, pp. 2991–3003, November 2019.

[34] R. Nebashi et al., “A 171k-LUT Nonvolatile FPGA using Cu Atom-Switch Technology in 28nm CMOS,” *Proc. Int. Conf. Field-Programmable Logic and Applications (FPL)*, pp. 323–327, August-September 2020.

[35] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai, “Nonvolatile magnetic flip-flop for standby-power-free SoCs,” *IEEE J. Solid-State Circuits*, vol. 44, no. 8, pp. 2244–2250, August 2009.

[36] I. Kazi, P. Meinerzhagen, P. Gaillardon, D. Sacchetto, Y. Leblebici, A. Burg, and G. De Micheli, “Energy/reliability trade-offs in low-voltage ReRAM-based non-volatile flip-flop design,” *IEEE Trans. Circuits and Systems I: Regular Papers*, vol. 61, no. 11, pp. 3155–3164, July 2014.

[37] K. Ali, F. Li, S.Y.H. Lua, and C.-H. Heng, “Energy- and area-efficient spin-orbit torque nonvolatile flip-flop for power gating architecture,” *IEEE Trans. VLSI Syst.*, vol. 26, no. 4, pp. 630–638, April 2018.

[38] Y. Seo, X. Fong, and K. Roy, “Fast and disturb-free nonvolatile flip-flop using complementary polarizer MTJ,” *IEEE Trans. VLSI Syst.*, vol. 25, no. 4, pp. 1573–1577, April 2017.

[39] B. Song, S. Cho, S.H. Kang, and S.-O. Jung, “Offset-cancellation sensing-circuit-based nonvolatile flip-flop operating in near-threshold voltage region,” *IEEE Trans. Circuits and Systems—I: Regular Papers*, vol. 66, no. 6, June pp. 2963–2972, August 2019.

[40] A. Jaiswal, R. Andrawis, and K. Roy, “Area-efficient nonvolatile flip-flop based on spin hall effect,” *IEEE Mag. Lett.*, vol. 9, pp. 4303304 1–4303304 4, April 2018.

[41] S. Ikeda, et al., “A perpendicular-anisotropy CoFeB-MgO magnetic tunnel junction,” *Nature Materials*, vol. 9, pp. 721–724, July 2010.

[42] S. Miura, et al., “Insertion layer thickness dependence of magnetic and electrical properties for double-CoFeB/MgO-interface magnetic tunnel junctions,” *IEEE Trans. Magn.*, vol. 55, no. 7, pp. 3401004 1–3401004 4, July 2019.

[43] K. Nishioka, et al., “Novel quad interface MTJ technology and its first demonstration with high thermal stability and switching efficiency for STT-MRAM beyond 2Xnm,” *VLSI Technology Dig. Tech. Papers*, pp. 120–121, June 2020.

[44] M. Natsui et al., “Nonvolatile logic-in-memory LSI using cycle-based power gating and its application to motion-vector prediction,” *J. Solid-State Circuits*, vol. 50, no. 2, pp. 476–489, February 2015.

[45] E. Pop, “Energy dissipation and transport in nanoscale devices,” *Nano Research*, vol. 3, pp. 147–169, March 2010.