DAEBI: A Tool for <u>D</u>ata Flow and <u>A</u>rchitecture <u>E</u>xplorations of <u>Bi</u>nary Neural Network Accelerators

Mikail Yayla[1,2], Cecilia Latotzke[3], Robert Huber[1], Somar Iskif[1], Tobias Gemmeke[3], and Jian-Jia Chen[1,2]

[1] Technical University of Dortmund, Germany
[2] Lamarr Institute for Machine Learning and Artificial Intelligence, Germany
[3] RWTH Aachen University, Germany
{mikail.yayla, robert.huber, somar.iskif, jian-jia.chen}@tu-dortmund.de
{latotzke, gemmeke}@ids.rwth-aachen.de

Abstract. Binary Neural Networks (BNNs) are an efficient alternative to traditional neural networks as they use binary weights and activations, leading to significant reductions in memory footprint and computational energy. However, the design of efficient BNN accelerators is a challenge due to the large design space. Multiple factors have to be considered during the design, among them are the type of data flow and the organization of the accelerator architecture. To the best of our knowledge, a tool for the design space exploration of BNN accelerators with regards to these factors does not exist.
In this work, we propose DAEBI, a tool for the design space exploration of BNN accelerators, which enables designers to identify the most suitable data flow and accelerator architecture. DAEBI automatically generates VHDL-code for BNN accelerator designs based on user specifications, making it convenient to explore large design spaces. Using DAEBI, we conduct a design space exploration of BNN accelerators for traditional CMOS technology using an FPGA. Our results demonstrate the capabilities of DAEBI and provide insights into the most suitable design choices. Additionally, based on a decision model, we provide insights for the design of BNN accelerator specifications that use emerging beyond-CMOS technologies.

Keywords: Binarized neural networks · Digital circuit design · Data flow · Hardware architecture · FPGA · ASIC

1 Introduction

Neural networks (NNs) have been applied successfully in various fields, such as image and speech recognition, natural language processing, and autonomous driving. They surpass traditional algorithms and human performance in various challenges, e.g., Convolutional Neural Networks (CNN) in the ImageNet Challenge [11] or in the PhysioNet Challenge [13]. However, NNs rely on a large number of parameters and need to perform a massive amount of computations to achieve high accuracy, leading to high resource demand. Yet, many use cases require efficient and intelligent decision making, which necessitate the inference to be performed on the edge to reduce latency and increase privacy. However, edge devices provide only limited resources in terms of energy and computational units as well as memory, posing a profound challenge for the design of efficient yet capable AI systems on such devices.

Traditional NN accelerators require substantial data transfers between memory and processing units, resulting in considerable energy consumption and latency [15]. To address the cost of data transfers, the word-length of the data is commonly reduced. The most extreme form of word-length reduction is binarization [25]. NNs with binary weights and activations, known as Binary Neural Networks (BNNs), can reduce the memory footprint of floating-point NNs by a factor of 32× while maintaining high accuracy [4,9,17]. Furthermore, BNNs reduce the computational energy required for processing by replacing the energy-intensive multiply-accumulate (MAC) operations with XNOR and popcount operations, which can be implemented more efficiently than in higher-precision NNs [2].

BNN accelerators have been implemented in both digital and analog ICs and several recent studies have demonstrated their efficiency as well as effectiveness [1,3,18,27,34]. BNN accelerators and general NN accelerators typically utilize classical CMOS-based ICs like microcontrollers, FPGAs, and ASICs. However, beyond-CMOS technologies are emerging rapidly as an alternative. Examples of emerging-beyond CMOS technologies are Resistive Random-Access Memory (RRAMs) [26], Ferroelectric Field-Effect transistors (FeFETs) [7,24,32], and Magnetoresistive Random-Access Memory (MRAM) [6,22].

In addition to the technology choice, two types of data flow methods, i.e., output stationary (OS) and weight stationary (WS), defined in [8], can be used efficiently for BNN accelerators [7]. In OS, new input activations and weights are streamed in each cycle, which necessitates only one accumulation register per computing unit. In WS, the weights are programmed into the XNOR gates once and reused as much as possible for multiple input activations, such that the number of new weight writes to the XNOR gates is minimized. This however necessitates a large amount of registers for storing intermediate results. The benefit of OS is a lower area footprint than in WS, as the partial sums are not intermediary stored but directly accumulated. The benefit of WS is the high reuse of the weights, requiring a significantly smaller number of rewrites to the XNOR gates compared to OS.

In emerging technologies, such as NVM-based XNOR gates, the cost of writing new weights is typically larger than applying input activations and reading the output. WS can be more efficient in these cases, see [3,7] in Table 1. However, Table 1 also shows that there is not clear pattern in the choice of WS or OS for BNN acceleration. The reason is that the decision process for the optimal data flow is highly time consuming for designers of BNN accelerators, because it requires to design all possible BNN accelerator versions for comparisons. Therefore, a tool which automatically generates the BNN accelerator designs for dif-

Table 1: BNN accelerator data flows with CMOS and beyond-CMOS technology

Data flow	CMOS based accelerator	Beyond-CMOS based accelerator
WS	[18]	[3, 7]
OS	[1, 34]	[27]

ferent data flow configurations would be highly beneficial for the design space exploration of BNN accelerators in the industry and in the research community.

In this work, we present the tool DAEBI, which enables designers to identify fast and conveniently the most suitable data flow and architecture for the BNN accelerators, for both classical CMOS and emerging beyond-CMOS technologies. Our contributions are as follows:

– We present DAEBI, a tool for the design-space exploration of BNN accelerators regarding different types of data flow and architectures. DAEBI automatically generates VHDL-code of BNN accelerator designs based on the user specifications.
– We further propose a decision rule for the data flow choice when different technologies are used to implement the BNN accelerator.
– To demonstrate the capabilities of DAEBI, we conduct a design space exploration of BNN accelerators with regards to data flow and accelerator architectures for the traditional CMOS technology on an FPGA. Furthermore, based on our decision model, we provide insights for the design of BNN accelerator specifications which use emerging beyond-CMOS technologies.

The paper is structured as follows. In Sec. 2 we introduce the basics of BNNs, BNN accelerators, and the data flow options OS and WS. In Sec. 3, we present our tool DAEBI and describe its usage and implementation. We then present the decision rule for determining the data flow of BNN accelerators in Sec. 4. Finally, we demonstrate the capabilities of DAEBI in Sec. 5. The DAEBI tool is fully open-source and available at `https://github.com/myay/DAEBI`.

2 System Model

We assume for a convolution layer of an NN a weight matrix **W** with dimensions $(\alpha \times \beta)$, where α is the number neurons and β the number of weights of a neuron. The input matrix **X** has dimensions $(\gamma \times \delta)$, where $\beta = \gamma$ (i.e., matrix multiplication between **W** and **X** can be performed) and δ is the number of convolution windows, i.e., unfolded kernels in the input. We leave out any layer indices for brevity. Every convolution (1D, 2D, etc.) of a conventional NN can be mapped to this matrix notation.

In general, each convolution layer in an NN (fully connected, 2D convolution, other convolution types) computes its outputs by performing the matrix multiplication $\mathbf{W} \times \mathbf{X}$, resulting in an output matrix with dimensions $\alpha \times \delta$. A matrix multiplication is performed by scalar products of different combinations of rows from **W** and columns from **X**. These scalar products are the MAC operations. An activation function is specified to convert the convolution layer outputs (MAC values) to the activations **A**.

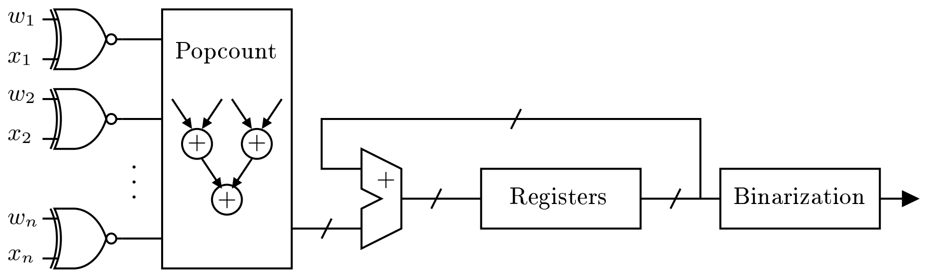

Fig. 1: Overview of a BNN computing unit.

2.1 Binarized Neural Networks (BNNs)

In BNNs, the weights and activations are binarized. The output of a BNN layer can be computed with

$$2 * \text{popcount}(\text{XNOR}(\mathbf{W}, \mathbf{X})) - \#bits > \mathbf{T}, \tag{1}$$

where XNOR($\mathbf{W}, \mathbf{X}$) computes the XNOR of the rows in $\mathbf{W}$ with the columns in $\mathbf{X}$ (analogue to matrix multiplication), *popcount* counts the number of set bits in the XNOR result, $\#bits$ is the number of bits of the XNOR operands, and $\mathbf{T}$ is a vector of learnable threshold parameters, with one entry for each neuron. The thresholds are computed with the batch normalization parameters, i.e., $T = \mu - \frac{\sigma}{\psi}\eta$, where each neuron has a mean μ and a standard deviation σ over the result of the left side of Eq. (1), and ψ and η are learnable parameters. For further details about the batch normalization parameters, please refer to [9, 23]. Finally, the comparisons against the thresholds produce binary values.

2.2 BNN Accelerators

The high-level overview of a BNN accelerator computing unit is shown in Fig. 1. The design is inspired by the studies in [10, 21]. The binary inputs and weights, which are in form of bitstrings of length n, are loaded into the XNOR gates. The XNOR gates (representing the binary multiplication) return the result of the XNOR operations as a bitstring of length n as well. Then, the popcount unit counts the number of bits that are “1”. Subsequently, the result of the popcount unit is accumulated in the registers. The binarizer returns binary value once all accumulations are completed.

Multiple computing units of the form in Fig. 1 can be used in parallel to increase the throughput. Such accelerators are organized with m computing units and n XNOR gates per computing unit, i.e., they have size $(m \times n)$, which determines the workload they can process. Accelerators of size $(m \times n)$ can further be embedded into a higher hierarchy, i.e., multiple accelerators of size $(m \times n)$ on the same chip.

In general, hardware (HW) is designed, synthesized, and evaluated in Electronic Design Automation (EDA) tools. This is done by creating the description of the HW and its behavior in a hardware description language (HDL), such as VHDL. The final HW designs will always be in some form of HDL, and could be engineered or generated in different ways.

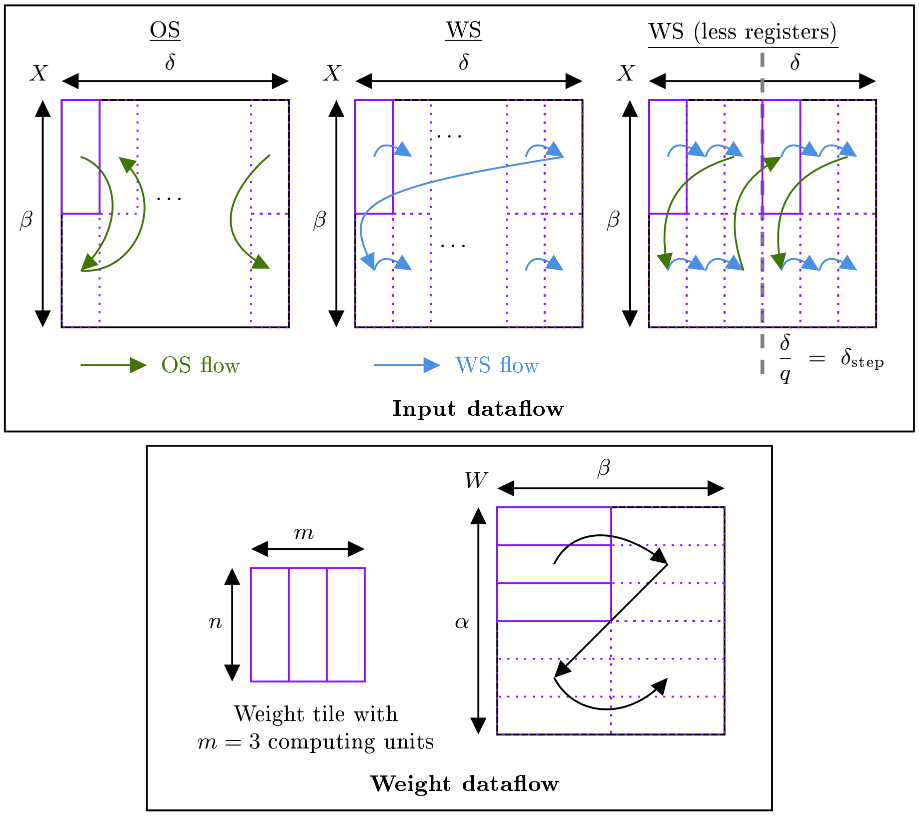

Fig. 2: OS and WS data flows.

2.3 Data flow in BNN Accelerators: OS and WS

To classify the data flow in BNNs, we use the categorization proposed in [8]. Applicable to BNNs in an efficient way are the output stationary (OS) and the weight stationary (WS) approach. For the workload, we use the matrix notation of the weight matrix $\mathbf{W}$ with dimensions $\alpha \times \beta$ and input matrix $\mathbf{X}$ with dimensions $\beta \times \delta$.

The OS data flow is shown in Fig. 2 (top left). In OS, an input of length n is retrieved from the first column of the input matrix (input column $\delta_{\text{step}} = 1$). This input is broadcasted to all computing units. The popcount result of the XNOR between inputs and weights is then stored in the accumulator. In the next iteration, the subsequent n weights of the currently computed set of neurons are loaded into the computing units. Then the subsequent n inputs are applied, which are also in the input column with $\delta_{\text{step}} = 1$. Afterwards, the popcount values are accumulated. This continues until all β of the neurons are processed, taking $\lceil \frac{\beta}{n} \rceil$ iterations for a neuron and input column combination. When the input column processing is completed, the accumulator is reset and the next input column ($\delta_{\text{step}} = 2$) is processed. In total, $\delta \lceil \frac{\beta}{n} \rceil$ iterations are needed for one neuron. When the first set of neurons have been processed, the next set of neurons is processed and it is repeated for $\lceil \frac{\alpha}{m} \rceil$ iterations.

The WS data flow is shown in Fig. 2 (top middle). Note that WS requires a certain number of registers per computing unit to store intermediate popcount values, as opposed to OS, which uses only one accumulation register per computing unit. In WS, the corresponding input (input column $\delta_{\text{step}} = 1$) of length n is retrieved from the input matrix and is broadcasted to all computing units as well. The popcount result of the XNOR between inputs and weights is then stored in the first register ($\delta_{\text{step}} = 1$). Then, the the subsequent input (from input column $\delta = 2$) of length n is retrieved and applied to all computing units. The popcount result is stored in the second register ($\delta_{\text{step}} = 2$). This continues until all columns in the input matrix are processed, i.e., when $\delta_{\text{step}} = \delta$. Note that the loaded weights stay the same for all δ_{step}. When δ columns have been processed, then the next n weights are loaded into the computing units. The process is repeated again, i.e., for $\delta_{\text{step}} = 1$, the result is added to the first register, for $\delta_{\text{step}} = 2$, the result is added to the second register, etc., taking $\delta \lceil \frac{\beta}{n} \rceil$ iterations for the set of neurons. When the first set of neurons have been processed, the next set of neurons is processed and in total there are $\lceil \frac{\alpha}{m} \rceil$ iterations. The number of required registers can be high (see Table 4). WS can also be used with less registers than δ, i.e., with $\frac{\delta}{q}$, where q is the register reduction factor. WS with less registers works the same way as WS, but only until the all registers are full. Then, an OS step is performed by loading a different set of weights. The WS data flow is continued again until all registers have been iterated and the process repeats (see Fig. 2).

A summary of the required number of executions of computing units and their resources is described in Table 2. The preferred data flow used for NVMs is WS, since it minimizes the number of writes to the XNOR gates and in NVMs typically the writes to memory cells are more costly than the reads [7]. However, neither for classical CMOS-based designs nor for emerging beyond-CMOS designs exists a clear recipe for the most suitable data flow choice between OS and WS in the case of BNNs.

Table 2: Number of registers, number of weight loads, and number of invocations for different data flow approaches in BNN accelerators.

Specification	OS	WS	WS (less registers)
Nr. of registers	m	δm	$\lceil \frac{\delta}{q} \rceil m$
Nr. of weight writes	$\delta \alpha \lceil \frac{\beta}{n} \rceil$	$\alpha \lceil \frac{\beta}{n} \rceil$	$q \alpha \lceil \frac{\beta}{n} \rceil$
Nr. of accelerator invocations	$\delta \lceil \frac{\alpha}{m} \rceil \lceil \frac{\beta}{n} \rceil$	$\delta \lceil \frac{\alpha}{m} \rceil \lceil \frac{\beta}{n} \rceil$	$\delta \lceil \frac{\alpha}{m} \rceil \lceil \frac{\beta}{n} \rceil$

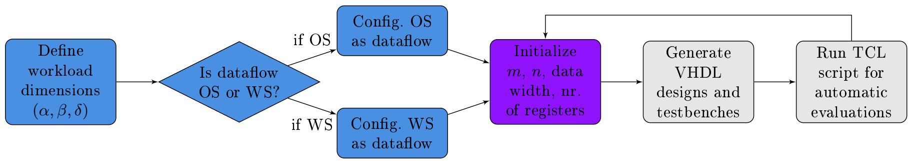
Define workload dimensions (α, β, δ)
Is dataflow OS or WS?
if OS
if WS
Config. OS as dataflow
Config. WS as dataflow
Initialize m, n, data width, nr. of registers
Generate VHDL designs and testbenches
Run TCL script for automatic evaluations

Fig. 3: Workflow of our DAEBI tool. Blue: data flow, purple: accelerator architecture. Blue and green are specifications from user. Gray: automatic steps performed by the tool.

3 Our Tool DAEBI

DAEBI enables designers of BNN accelerators to evaluate whether OS or WS is the most suitable data flow approach for their specific accelerator architecture and technology. The code of DAEBI is fully open source and available at `https://github.com/myay/DAEBI`.

In the following, in Sec. 3.1, we describe the high-level overview of our DAEBI tool and its workflow. Then, in Sec. 3.2, we explain the implementation and structure of DAEBI, as well as the BNN accelerator designs that are available in our tool.

3.1 High-Level Overview of DAEBI

The workflow of DAEBI is shown in Fig. 3. The user first defines the type of data flow used (OS or WS) and then defines the accelerator architecture (m, n, and the nr. of registers in case of WS). Subsequently, the user specifies the workload that needs to be processed by the accelerator (α. β, δ). From these inputs, the BNN accelerator is generated automatically in the form of VHDL with corresponding realistic simulated workloads in testbenches to run the designs. Then, the generated files can be loaded into EDA tools for syntheses and evaluations based on TCL scripts. After the EDA tools return the results, the user can reconfigure the accelerator design and repeat the steps for performing design space explorations.

3.2 Implementation of DAEBI and the Hardware Designs

Our implemented BNN accelerator, which includes options for OS and WS data flows and different architecture configurations is designed in VHDL. For automatic code generation from templates, we use the templating tool Jinja2. We also use Python scripts for various steps of creating designs for enabling configurability regarding the data flow and accelerator architecture. All the code regarding the accelerator and its related tools are released as open source in `https://github.com/myay/DAEBI`.

The BNN accelerator is configurable, i.e., with respect to the number of computing units m (accelerator elements in parallel), the number of XNOR gates n per computing unit, the number of required bits in the registers, and the number of registers in the WS approaches. Furthermore, for the specified number of XNOR gates, the popcount unit (realized by an adder tree with $\log_2(n)$ levels) is generated with corresponding minimal number of bits in the adders and pipeline registers. The generated hardware designs in our tool for the OS data flow and WS data flow are shown in Fig. 4 and 5 respectively. Both designs operate in a pipelined fashion. Note that the OS design only needs one accumulation register. The WS design needs δ registers (less registers can also be used). The registers in WS are implemented as a register file, which requires ports for read and write data, register selection, and write enable.

In the `rtl/` folder is the VHDL code of all the components used for OS and WS designs. For the OS design, there are VHDL files for the XNOR gates, the XNOR gate array, the popcount unit (with registers and adders), a simple accumulator, and the binarizer. For the WS design, the components are the same, except that it uses a regfile with multiple registers and ports. If the user want to change any of these subcomponents, modifications have to be performed here. However, with modifications, the functionality of the higher-level design needs to be tested.

For building the designs of entire accelerators with WS and OS (and for multiple computing units in parallel), we use the templates in the `templates/` folder. If the user wants to change the high-level design of the OS or WS hardware, modification have to performed in the files of this folder. Here we also include the templates of the testbenches used for evaluation. Note that when

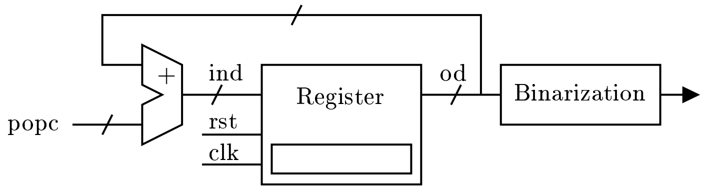

Fig. 4: Design of OS data flow in our DAEBI tool for one computing unit.

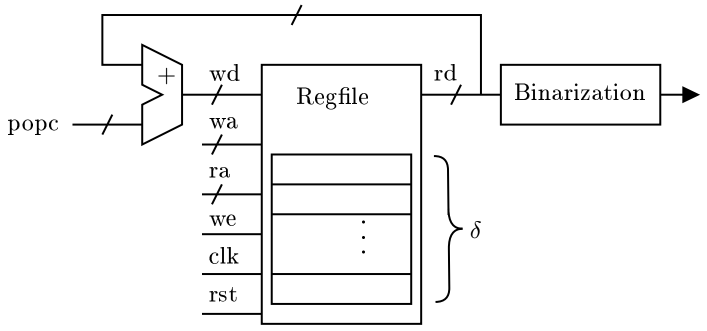

Fig. 5: Design of WS data flow in our DAEBI tool for one computing unit.

the debug flag is set during the accelerator architecture definition, the results of the computations are simulated in the testbench and are printed in the console.

In the `sim/` folder, the tests for the subcomponents and the computing units can be found. The tests have been written in cocotb with a high number of test cases. The tests can be rerun by the user with more test cases and with different seeds. In case of modifications on any part of the design, the tests should be rerun and, if needed, changed to test the correctness of modified designs. A script for testing all components and the higher-level designs is provided this folder.

4 Decision Model for using OS or WS

In general, the major energy cost in NNs is due to data transfers [15]. Here, we focus on the energy of the data transfer. For the cost of OS, C_{OS}, E_{OS} equals $2\times$ the read energy E_{RD} per Partial Sum (PS), since the weights and inputs are changed each cycle. The area A_{OS} is one register (REG) per computing unit as the resulting popcount is computed and stored in one register (see Eq. (2)). The processing time T_{OS} is described in Table 3. All in all, the cost is considered to be a product of energy E, area A, time T.

$$C_{\mathrm{OS}} = A_{\mathrm{OS}} * E_{\mathrm{OS}} * T_{\mathrm{OS}} = 1\mathrm{REG} * 2\frac{E_{\mathrm{RD}}}{\mathrm{PS}} * (\lceil \log_2(n) \rceil + 4)cycle \qquad (2)$$

WS uses as many registers as input columns δ per computing unit. The input columns δ are predetermined by the NN model and can be high (e.g. 50176, see ResNet-18 in Table 4). Using less registers is also possible with the register reduction factor in Sec. 2.3. However, WS reuses the weights which are a key driver of its efficiency. This weight reuse allows to keep the once loaded weights in the computing units until all δ columns have been processed. The cost C_{WS} increases significantly with δ but is for small n lower than C_{OS} (see Eq. (3)). Hence, WS requires less data transfers compared to OS.

$$C_{\mathrm{WS}} = A_{\mathrm{WS}} * E_{\mathrm{WS}} * T_{\mathrm{WS}} = \delta\mathrm{REG} * \frac{E_{\mathrm{RD}}}{\mathrm{PS}} * (\lceil \log_2(n) \rceil + 6) * 3cycle \qquad (3)$$

As both options, WS and OS, are valid for a BNN data flow, the open question remains, when to use which option. For this, we introduce a threshold τ which

Table 3: Number of clock cycles needed by our designs

Component	OS	WS
XNOR array	-	-
Popcount unit	$\lceil \log_2(n) \rceil + 3$	$\lceil \log_2(n) \rceil + 3$
Accumulator	1	3
Binarizer	-	-

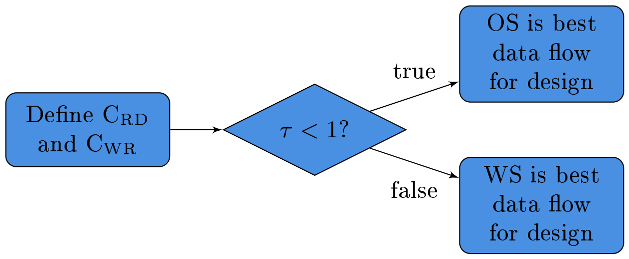
Define C_{RD} and C_{WR}
$\tau < 1?$
true
false
OS is best data flow for design
WS is best data flow for design

Fig. 6: Decision diagram for τ and the data flow selection.

indicates the optimal trade-off point for each data flow option (see Eq. 4). As the cost C_{OS} is for most cases except for small n and small δ larger than the cost C_{WS}, it might seem obvious to choose always OS. However, the cost C_{RD} and C_{WR} for writing data depends highly on the technology [33]. Here, C_{WR} can cost 146× more than C_{RD}. It is worth noting that, C_{WR} as well as C_{RD}, depend only on time and energy as no additional registers are needed for the data movement between the global buffer and XNOR gates. The threshold τ takes $\frac{C_{\text{WR}}}{C_{\text{RD}}}$ as well as $\frac{C_{\text{OS}}}{C_{\text{WS}}}$ into account (see Eq. 4). Eq. 4 can be transformed to Eq. 6. To conclude, it is best to use OS up to τ equals 1.

$$\tau = \frac{C_{\text{WR}}}{C_{\text{RD}}} * \frac{C_{\text{OS}}}{C_{\text{WS}}} \tag{4}$$

$$\frac{C_{\text{WR}}}{C_{\text{RD}}} = \frac{T_{\text{WR}} * E_{\text{WR}}}{T_{\text{RD}} * E_{\text{RD}}} \tag{5}$$

$$\tau_{\text{our design}} = \frac{2 * T_{\text{WR}} * E_{\text{WR}}}{\delta * T_{\text{RD}} * E_{\text{RD}}} * \frac{(\lceil \log_2(n) \rceil + 4)}{(\lceil \log_2(n) \rceil + 6) * 3} \tag{6}$$

5 Evaluation

We first introduce the experiment setup in Sec. 5.1. Then, in Sec. 5.2, we use our tool DAEBI to perform design space explorations for BNN accelerators for a CMOS-based FPGA, with regards to data flow and architecture. Finally, in Sec. 5.3, we provide insights based on our decision model for using OS or WS on different beyond-CMOS technologies.

5.1 Experiment Setup

We use our DAEBI tool to perform a design space exploration of BNN accelerators with respect to the data flow configurations (OS and WS) and different architectures (m, n). For the traditional CMOS-based case study in this work we use the Zynq Ultrascale+ ZCU104 evaluation board FPGA.

For area, energy, and latency estimations we synthesize our design for the FPGA using Vivado. We use the out-of-context mode to avoid I/O limitations. To evaluate the designs generated with DAEBI, we use the generated testbenches

Table 4: Maximum matrix dimensions of the weight matrix $\mathbf{W}$ and input $\mathbf{X}$ in three typical BNN architectures used for efficient edge inference

NN architecture	Dataset	Top-1 accuracy	$\mathbf{W}: \max(\alpha), \max(\beta)$	$\mathbf{X}: \max(\gamma), \max(\delta)$
VGG3 [31]	FMNIST	90.68%	2048, 3136	3136, 196
VGG7 [31]	CIFAR10	90.37%	1024, 8192	8192, 1024
ResNet-18 [29]	ImageNet	58.71%	512, 4608	4608, 50176

and create saif files (which store information about the switching activity based on simulations with testbenches). The testbenches contain representative workloads (i.e., α, β, δ, as shown in Table 4) for the BNNs, such that the HW is operated in a fully utilized manner. To estimate the energy consumption after the synthesis and implementation steps in Vivado, we feed the saif-files created during post-implementation simulation with the testbenches to the power analyzer. For area estimations, we rely on the utilization report in Vivado. For the latency, we measure the number of clock cycles the designs needs to compute one data frame and multiply that by the clock frequency, which is always maximized such that no timing errors occur.

5.2 Results of Experiments for Classical CMOS Technology

In the following, we present the results of design space exploration using our tool DAEBI for the classical CMOS technology on the FPGA. The resulting designs are analyzed with regards to their costs in energy, area and time. To identify the optimal design, the different designs are compared in a Pareto plot. Here, two objectives, the energy and the area-time (AT) complexity, are used to identify the points with the best trade-off. We consider that the Look-Up Tables (LUTs) represent the area cost, as they are the most important elements to build computing units in FPGAs. Time in our evaluation is the required processing time per sample. The results are shown for OS and WS (and different architecture configurations) in Fig. 7.

Regarding the comparison between OS and WS, we observe that OS performs better in AT and energy by around one order of magnitude. This is due to the fact that in our FPGA, the read and write costs are the same, and using more registers in WS increases the area cost significantly. As expected, more registers, e.g. from 196 to 1024 (we show this case for n downto 64 since the resource use becomes too high), lead to higher cost in AT and energy. Furthermore, we observe that in all cases, designs become better as n increases. Hence, the designs with the largest n, i.e., $n = 512$ in our case, lead to the Pareto-optimal solutions.

The experiment results further show that the resource increases more with m (multiple computing units in parallel) than with n (i.e., increasing the number of XNOR gates in one component and thereby increasing the the number levels in the popcount unit). For example when $n = 256,\ m = 2$ instead of $n = 512,\ m = 1$ is used, the second option is always better. Therefore, doubling m and halving n does not lead to more efficient designs for both OS or WS. As examples for this case we show two points with $m = 2$ for OS in Fig. 7.

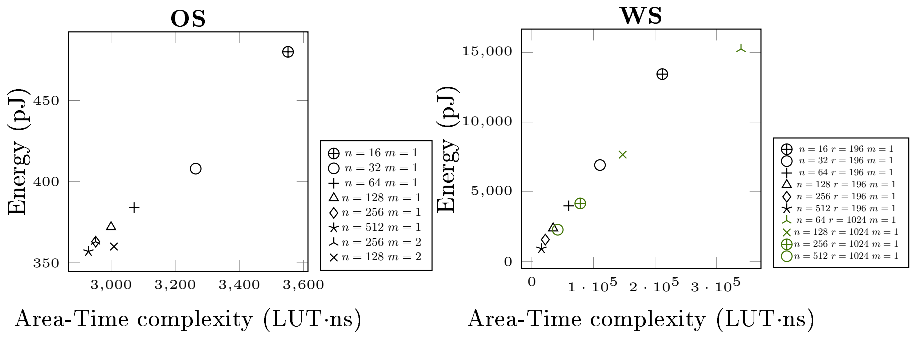

Fig. 7: OS (left) and WS (right) data flow designs with different architecture configurations evaluated on Xilinx Zynq Ultrascale+ ZCU104 FPGA. n: Nr. of XNOR gates. m: Nr. of computing units in parallel. r: Nr. of registers in WS. Two separate plots are shown due to the large differences among OS and WS.

5.3 Insights for Beyond-CMOS Technology

In Table 5, the cost for different memory technologies are presented. As some memory technologies may vary in read and write cost per device, we state the cost with regards to the range in our references [5, 12, 14, 16, 19, 20, 28, 30]. The result of the quotient of cost in Eq. 6 differ with regards to the initial read and write cost for the memory technology. Hence, we state two different δ values for the minimum read and write cost and maximum read and write cost of each technology. The cost refers here to the read and write delay as well as read and write energy. Eq. 6 is used with $n = 512$ because this value performed best in the analysis in Fig. 7.

If $\tau < 1$, OS is the best data flow, whereas if $\tau \geq 1$ WS is the best data flow. Thus, the δ in Table 5 is chosen such that $\tau < 1$ is achieved. As δ can be only an integer value, all numbers smaller than the presented δ enable a $\tau \geq 1$. Furthermore, the smaller δ is always used in the following as the threshold to determine whether OS or WS shall be used. This is due to the fact that the range of cost for each device type has to be taken into account to prevent unfavorable data flow choices. This means that for SRAM memory technology, always a data flow of OS is preferable. However, for designs with reasonable sized δ, WS can be a better data flow than OS. This is especially the case for FeFET, FeRAM, PCM, and ReRAM, i.e., WS is preferred as data flow for FeFET for a $\delta < 20$, for FeRAM for a $\delta < 10$, for PCM for a $\delta < 10$, and for ReRAM for a $\delta < 8$. For STT-RAM, only small δ allow a reasonable choice of OS, i.e., for a $\delta < 5$ WS is preferred as data flow.

To conclude, Table 5 summarizes for different memory technologies the cases in which the data flow types OS or WS should be used. The initial technology dependent cost can be changed to support other technologies. Afterwards, DAEBI can be used to verify the decision with evaluations in EDA tools.

Table 5: Evaluation of τ based on our decision model in Sec. 4. Read and write cost are also shown for different memory technologies with $n = 512$.

Memory technology	T_{RD} (ns)	T_{WR} (ns)	E_{RD} (pJ)	E_{WR} (pJ)	δ for min cost $\tau < 1$	δ for max cost $\tau < 1$
SRAM [5,19]	$\tilde{0}.2$-2	$\tilde{0}.2$-2	574	643	≥ 1	≥ 1
STT-RAM [5,19,28]	2-35	3-50	550	3243	≥ 6	≥ 5
ReRAM [5,14,30]	$\tilde{1}0$	$\tilde{5}0$	1.6-2.9	4-14	≥ 8	≥ 14
PCM [5,28]	20-60	20-150	12.4	210.3	≥ 10	≥ 25
FeRAM [5,28]	20-80	50-75	12.4	210	≥ 25	≥ 10
FeFET [12,16,20]	0.279	0.55	0.28	4.82	≥ 20	≥ 20

6 Conclusion

We present DAEBI, a tool for the design space exploration of BNN accelerators that enables designers to identify the most suitable data flow and architecture. DAEBI automatically generates VHDL-code for BNN accelerator designs based on user specifications, making it convenient to explore large design space. Using DAEBI, we perform a design space exploration for a classical CMOS-based FPGA to demonstrate the tool's capabilities. In addition to the automatic design generation, we also provide guidance on how to choose between OS and WS for hardware based on emerging beyond-CMOS technologies. We believe that DAEBI will be valuable to both research and industry for exploring the design of efficient BNN hardware in resource-constrained AI systems.

Acknowledgements This paper has been supported by Deutsche Forschungsgemeinschaft (DFG) project OneMemory (405422836), by the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis" (project number 124020371), subproject A1 (http://sfb876.tu-dortmund.de) and by the Federal Ministry of Education and Research of Germany and the state of NRW as part of the Lamarr-Institute for ML and AI, LAMARR22B.

References

1. Ando, K., et al.: BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W. IEEE Journal of Solid-State Circuits (JSSC) **53**(4), 983–994 (2017)
2. Andri, R., Cavigelli, L., Rossi, D., Benini, L.: YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. In: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 236–241 (2016)
3. Bertuletti, M., noz Martín, I.M., Bianchi, S., Bonfanti, A.G., Ielmini, D.: A Multilayer Neural Accelerator With Binary Activations Based on Phase-Change Memory. IEEE Transactions on Electron Devices **70**(3), 986–992 (2023)

4. Blott, M., et al.: FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS) **11**(3), 1–23 (2018)
5. Boukhobza, J., Rubini, S., Chen, R., Shao, Z.: Emerging NVM: A Survey on Architectural Integration and Research Challenges **23**(2), 1084–4309 (2017)
6. Chang, L., et al.: PXNOR-BNN: In/with spin-orbit torque MRAM preset-XNOR operation-based binary neural networks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **27**(11), 2668–2679 (2019)
7. Chen, X., Yin, X., Niemier, M., Hu, X.S.: Design and optimization of FeFET-based crossbars for binary convolution neural networks. In: 2018 Design, Automation, Test in Europe (DATE). pp. 1205–1210 (2018)
8. Chen, Y.H., Emer, J., Sze, V.: Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). pp. 367–379 (2016)
9. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
10. Dave, A., Frustaci, F., Spagnolo, F., Yayla, M., Chen, J.J., Amrouch, H.: HW/SW Codesign for Approximation-Aware Binary Neural Networks. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **13**(1), 33–47 (2023)
11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09 (2009)
12. George, S., et al.: Nonvolatile memory design based on ferroelectric FETs. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–6 (2016)
13. Goldberger, A.L., et al.: PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation" **101**(23), e215–e220 (2000)
14. Hirtzlin, T., et al.: Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks. In: 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). pp. 288–292 (2019)
15. Horowitz, M.: 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). pp. 10–14. IEEE (2014)
16. Ko, D.H., Oh, T.W., Lim, S., Kim, S.K., Jung, S.O.: Comparative Analysis and Energy-Efficient Write Scheme of Ferroelectric FET-Based Memory Cells. IEEE Access **9**, 127895–127905 (2021)
17. Latotzke, C., Gemmeke, T.: Efficiency versus accuracy: a review of design techniques for dnn hardware accelerators. IEEE Access **9**, 9785–9799 (2021)
18. Li, G., Zhang, M., Zhang, Q., Lin, Z.: Efficient binary 3D convolutional neural network and hardware accelerator. Journal of Real-Time Image Processing **19**(1), 61–71 (2022)
19. Li, Y., Chen, Y., Jones, A.K.: A Software Approach for Combating Asymmetries of Non-Volatile Memories. In: Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design. pp. 191–196. ISLPED '12 (2012)
20. Ni, K., Li, X., Smith, J.A., Jerry, M., Datta, S.: Write Disturb in Ferroelectric FETs and Its Implication for 1T-FeFET AND Memory Arrays. IEEE Electron Device Letters **39**(11), 1656–1659 (2018)
21. Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G., Marr, D.: Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In: 2016 International Conference on Field-Programmable Technology (FPT). pp. 77–84 (2016)

22. Resch, S., et al.: PIMBALL: Binary neural networks in spintronic memory. ACM Transactions on Architecture and Code Optimization (TACO) **16**(4), 1–26 (2019)
23. Sari, E., Belbahri, M., Nia, V.P.: How Does Batch Normalization Help Binary Training? arXiv:1909.09139 (2019)
24. Soliman, T., et al.: Efficient FeFET Crossbar Accelerator for Binary Neural Networks. In: 2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP). pp. 109–112 (2020)
25. Stadtmann, T., Latotzke, C., Gemmeke, T.: From quantitative analysis to synthesis of efficient binary neural networks. In: 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA). pp. 93–100. IEEE (2020)
26. Sun, X., et al.: Fully parallel RRAM synaptic array for implementing binary neural network with (+ 1,- 1) weights and (+ 1, 0) neurons. In: 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 574–579 (2018)
27. Sunny, F.P., Mirza, A., Nikdast, M., Pasricha, S.: Robin: A Robust Optical Binary Neural Network Accelerator. ACM Transactions on Embedded Computing Systems (TECS) **20**(5), 1–24 (2021)
28. Suresh, A., Cicotti, P., Carrington, L.: Evaluation of emerging memory technologies for HPC, data intensive applications. In: 2014 IEEE International Conference on Cluster Computing (CLUSTER). pp. 239–247 (2014)
29. Tu, Z., Chen, X., Ren, P., Wang, Y.: AdaBin: Improving Binary Neural Networks with Adaptive Binary Sets (2022)
30. Wu, Q., et al.: A Non-volatile Computing-in-Memory ReRAM Macro using Two-bit Current-Mode Sensing Amplifier. In: 2021 IEEE 10th Non-Volatile Memory Systems and Applications Symposium (NVMSA). pp. 1–6 (2021)
31. Yayla, M., Chen, J.J.: Memory-efficient training of binarized neural networks on the edge. In: Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (2022)
32. Yayla, M., et al.: Reliable Binarized Neural Networks on Unreliable Beyond Von-Neumann Architecture. IEEE Transactions on Circuits and Systems I: Regular Papers **69**(6), 2516–2528 (2022)
33. Zangeneh, M., Joshi, A.: Performance and Energy Models for Memristor-Based 1T1R RRAM Cell. In: Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI '12). pp. 9–14 (2012)
34. Zhang, Y., Chen, G., He, T., Huang, Q., Huang, K.: ViraEye: An Energy-Efficient Stereo Vision Accelerator with Binary Neural Network in 55 nm CMOS. In: Proceedings of the 28th Asia and South Pacific Design Automation Conference. pp. 178–179 (2023)