Bitwise Systolic Array Architecture for Runtime-Reconfigurable Multi-precision Quantized Multiplication on Hardware Accelerators

Yuhao Liu[1,3], *Student Member, IEEE*, Salim Ullah[2], Akash Kumar[2,3], *Senior Member, IEEE*

[1]Dresden University of Technology, Germany [2]Ruhr University Bochum, Germany

[3]Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI Dresden/Leipzig), Germany

Email: yuhao.liu1@tu-dresden.de, {salim.ullah, akash.kumar}@rub.de

***Abstract*—Neural network accelerators have been widely applied to edge devices for complex tasks like object tracking, image recognition, etc. Previous works have explored the quantization technologies in related lightweight accelerator designs to reduce hardware resource consumption. However, low precision leads to high accuracy loss in inference. Therefore, mixed-precision quantization becomes an alternative solution by applying different precision in different layers to trade off resource consumption and accuracy. Because regular designs for multiplication on hardware cannot support the precision reconfiguration for a multi-precision Quantized Neural Network (QNN) model in runtime, we propose a runtime reconfigurable multi-precision multi-channel bitwise systolic array design for QNN accelerators. We have implemented and evaluated our work on the *Ultra96* FPGA platform. Results show that our work can achieve $1.3185\times$ to $3.5671\times$ speedup in inferring mixed-precision models and has less critical path delay, supporting higher clock frequency ($250MHz$).**

I. Introduction

Recent research of edge hardware devices widely applied *Neural Networks* (NN) on state-of-the-art applications, such as autonomous driving, the Internet of Things, wearable devices, voice and image recognition, etc. Considering the conflict between limited resources on the edge device and continually extending sizes of neural network models, related works explored the *Quantized Neural Network* (QNN) to reduce storage and hardware resource consumption by applying lower precision. For instance, *NVDLA* [1] and *Vitis DPU* [2] support the *INT8* 8-bit quantization in their deep learning processor designs. *FINN* [3, 4], *HLS4ML* [5], *LogicNets* [6], etc. proposed different frameworks to generate specialized inference accelerator designs on FPGA for the given low-precision trained (< 8 bits) QNN models to reduce the on-chip resource consumption. However, various prior works have presented a higher accuracy loss in lower-precision quantized network models. For instance, the 1-bit quantized *Multilayer Perceptron* (MLP) model shown in work [7] of Su et al. has an $8\times$ higher memory saving rate than the 8-bit quantized model applying the same network structure. However, the error of 1-bit models is about 32.7% higher than the 8-bit model. Therefore, to trade off the low resource consumption and high accuracy loss in QNN hardware accelerator designs, the works from *HAQ* [8], Chen et al. [9], Tang et al. [10], etc. explored mixed-precision quantization by using different precision in different layers. Compared to uniform quantization schemes

TABLE I: Inference Accuracy of Quantized Network Models Applying Unified-Precision and Mixed-Precision Schemes

Network Type		Precision Settings in Four Layers of TFC and TCV models						
		1/1/1/1	2/2/2/2	1/2/4/8	4/1/2/8	4/4/4/4	8/8/8/8	Float
TFC	Accuracy/%	92.29	96.37	95.91	-	97.55	97.36	97.89
	Weights/*Byte*	7376	14752	9984	-	29504	59008	236032
TCV	Accuracy/%	96.26	98.96	-	98.79	99.10	99.14	99.14
	Weights/*Byte*	29848	59696	-	55712	119392	238784	955136

TABLE II: Comparison of Unified- and Mixed-Precision Quantized MLPs Inferred on FPGA-based NN Accelerator

Design	Precision	LUT	FF	BRAM	Frequency	Latency	Accuracy
Vivado IP	8/8/8/8	24090	22175	135	150MHz	137.654us	97.74%
	1/2/4/8					131.059us	95.96%

across all layers (either high or low precision), mixed-precision quantized networks have middle-level inference accuracy and memory consumption for weight storage. We trained six tiny MLPs and six tiny *Convolution Neural Networks* (CNNs) models based on the *Brevitas* [11] with different precision to evaluate the accuracy loss and memory saving. *Tiny MLP* (TFC) and *Tiny CNN* (TCV) models are trained with the *MNIST* dataset [12, 13]. The TFC models comprise four layers with 64, 64, 64, and 10 neurons, respectively. TCV models have two convolution layers, each followed by a 2×2 max pooling layer. Following the final pooling layer are two fully connected layers. Each convolution layers have 64 3×3 kernels, while two fully connected layers have 64 and 10 neurons, respectively. To achieve maximum compression of network weights, we apply lower precision to layers with a higher number of weights. Therefore, as shown in Table I, TFC applies 1/2/4/8-bit quantization, and TCV applies 4/1/2/8-bit quantization, respectively, as their mixed-precision schemes. Results show that 8-bit quantized models have the highest accuracy, similar to 32-bit floating-point-based networks. The two 1-bit models have the lowest accuracy with the least memory storage for weights. Meanwhile, two mixed-precision have balanced accuracy and memory requirements.

A. Motivation

Mixed-precision QNNs show the potential to achieve a better and more flexible trade-off between resource consumption and accuracy loss. Prior works explored the design of related accelerators better to support the inference of mixed-precision

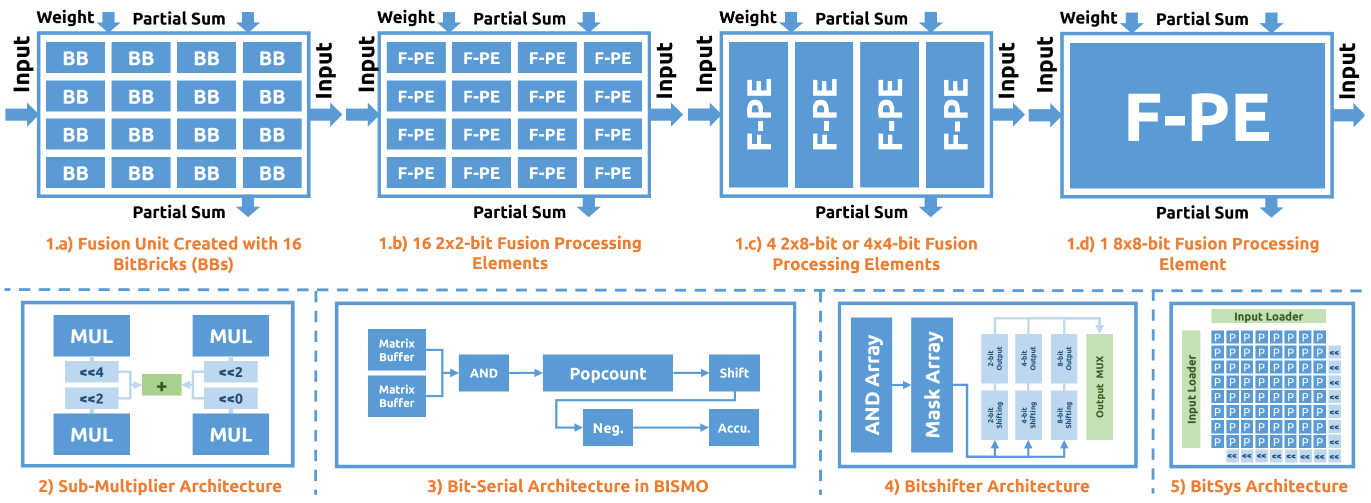

Weight
Partial Sum
Input
BB
F-PE
Partial Sum
1.a) Fusion Unit Created with 16 BitBricks (BBs)
1.b) 16 2x2-bit Fusion Processing Elements
1.c) 4 2x8-bit or 4x4-bit Fusion Processing Elements
1.d) 1 8x8-bit Fusion Processing Element
MUL
<<4
<<2
+
<<0
2) Sub-Multiplier Architecture
Matrix Buffer
AND
Popcount
Shift
Neg.
Accu.
3) Bit-Serial Architecture in BISMO
AND Array
Mask Array
2-bit Output
4-bit Output
8-bit Output
2-bit Shifting
4-bit Shifting
8-bit Shifting
Output MUX
4) Bitshifter Architecture
Input Loader
P
<<
5) BitSys Architecture

Fig. 1: Architectures of Prior Works and *BitSys*

networks on hardware. Results report that utilizing fixed-precision multipliers diminishes the performance advantages of mixed-precision accelerators. As shown in Table II, one previous work of Liu et al. [14] implemented one single-layer NN accelerator on *Ultra96-V2* FPGA platform with 64 8-bit integer *Vivado* multiplier IPs to infer one 8-bit quantized MLP and one 1/2/4/8-bit mixed-precision quantized MLP trained by *Brevitas* [11] with *MNIST* dataset [12, 13]. Both MLPs have four layers with 64, 64, 64, and 10 neurons, respectively. Table II listed the average inference latency of one *MNIST* input, computed by averaging the total latency of 1000 times inputs. The results indicate that the inference speed of the mixed-precision MLP has not significantly improved compared to the uniformly 8-bit quantized network. Because the input width of 8-bit *Vivado* multiplier IP cannot be reconfigured as 1/2/4-bit in runtime, all input data must be unified and extended to the largest precision, 8 bits. As a result, the inference acceleration of the mixed-precision model can only benefit from the transmission speedup between off-chip memory and FPGA based on low-precision data, not from the computation. Therefore, if multipliers can reconfigure the input precision and channel number in runtime, for instance, reset a single-channel 8-bit input as a dual-channel 4-bit input for signed 8/4-bit quantized layers, the inference of mixed-precision network models can be sped up on hardware.

B. Contributions

Prior works, such as *PIR-DSP* [15], *BitFusion* [16], *Multiplier-Tree* [14], *Bitshifter* [14], etc., explored the designs of multi-precision multipliers. Extending on our abstract in [17], we proposed a *Bitwise Systolic Array Architecture* (BitSys) in this manuscript supporting quantized multi-precision multi-channel runtime reconfigurable multiplication for neural network accelerator designs. The key features and contributions of this work are:

- We implemented one systolic-array-based multiplier, *BitSys*, based on the bitwise (1-bit) processing element and optimized it with LUT primitive for FPGA. Our design supports runtime reconfiguration for signed/unsigned 8/4/2/1-channel 1/2/4/8-bit multiplication. Moreover, this multiplier is specially designed to support the XNOR multiplication for the *Binarized Neural Network* (BNN) in *FINN* [3, 4].
- We extended our multiplier as a Multiply-Accumulator (MAC) to implement one single-layer accelerator and one systolic array accelerator and evaluate them for the mixed-precision model inference acceleration.

We evaluated our multipliers, MAC, and accelerator implementations on the *Ultra96-V2* FPGA platform and compared them with previous works. The synthesis and implementation report in *Vivado* shows our designs have low critical path delay from $1.357ns$ to $1.719ns$. The measurement result proves that our systolic array accelerator is $1.3185\times$ to $3.5671\times$ faster in the inference of mixed-precision networks than previous works.

C. Organization

This manuscript is structured as follows: Section II compares our *BitSys* design with related works. Section III introduces implementations of *BitSys* architecture. Section IV shows the evaluation results on *Ultra96-V2* platform compared with related works. Section V concludes the contents of this paper.

II. Background

A. Classification of Prior Multi-precision Multiplier Designs

Previous work explored different schemes for multi-precision multiplier designs, which can be classified by bit-serial/bit-parallel architectures and fixed/variable input widths.

Bit-serial multipliers execute the bitwise processing for multiplication in serial, such as *BISMO* [22], the work of Ienne et al. [23], the work of Shafer et al. [24], etc. For example, as shown in Figure 1.3, *BISMO* loads the inputs with the batch size of k-bit to execute the pipelined processing in serial. For m-bit inputs, it takes $\frac{m}{k}$ clock cycles to complete the multiplication. As a result, low-precision multiplication consumes fewer clock cycles than high-precision. Therefore, this design scheme can support temporal reconfiguration for different precision in runtime by

TABLE III: Differences between the BitSys Architecture and Previous Works

Work	Platform	Accu. or Approx.	No DSP	Signed or Unsigned	Available Precision										
					1×1	2×2	4×4	4×16	8×8	8×16	9×9	16×16	18×27	24×24	32×32
Guo et al. [18]	FPGA	Approx.	√	Signed	×	×	×	√	×	√	×	×	×	×	×
Neda et al. [19]	FPGA	Approx.	√	Signed	×	×	×	×	√	×	×	√	×	×	×
Shun et al. [20]	FPGA	Accu.	√	Signed	×	×	×	×	√	×	×	√	×	√	√
Pfänder et al. [21]	FPGA	Accu.	√	Both	×	×	×	×	√	√	×	√	×	√	√
PIR-DSP [15]	FPGA	Accu.	×	Both	×	√	√	×	×	×	√	×	√	×	×
Multiplier-Tree [14]	FPGA	Accu.	√	Both	√	√	√	×	√	×	×	√	×	×	√
Bitshifter [14]	FPGA	Accu.	√	Both	√	√	√	×	√	×	×	√	×	×	√
BitSys (Ours)	**FPGA**	**Accu.**	√	**Both**	√	√	√	×	√	×	×	×	×	×	×

completing more multiplications for lower precision in m clock cycles. However, for n times inputs, this scheme requires $n \times m$ cycles in computation, which leads to a high inference latency in hardware accelerators.

Therefore, most prior works are designed as bit-parallel architectures based on sub-multiplier schemes as shown in Figure 1.2, which generate one output per clock cycle, such as the works of Neda et al. [19], Guo et al. [18], Liu et al. [14], Pfänder et al. [21], and *PIR-DSP* [15]. For $2n \times 2n$-bit multiplication, $A \times B = A_0B_0 \times 2^{2n} + (A_1B_0 + A_0B_1) \times 2^n + A_1B_1$, if two inputs are split as four n-bit data, A_0, A_1, B_0, and B_1, the multiplication result is computed by summing the products of four $n \times n$-bit sub-multiplier results by 2^{2n}, 2^n, and 1 separately, which can be converted as $2n/n/0$-bit preset left-shifting. Therefore, if we bypass the outputs of two sub-multipliers with n-bit left shifting, the sum of four sub-multipliers is dual-channel $n \times n$-bit multiplication. Otherwise, the result is single-channel $2n \times 2n$-bit multiplication.

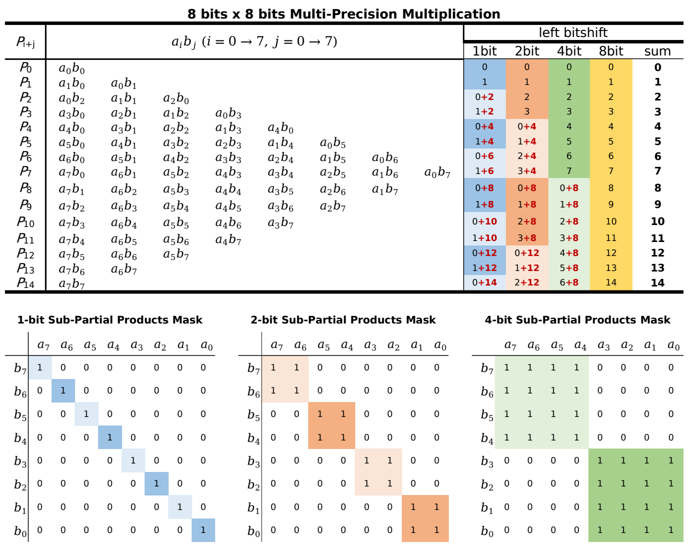
However, bypassing two sub-multipliers leads to low hardware efficiency. The works of Li et al. [25], Dai et al. [26], and *BitFusion* [16] explored another scheme to utilize all sub-multipliers in different precision. For instance, the *BitFusion* architecture shown in Figure 1.1 implemented sixteen 2-bit multipliers, *BitBricks* (BBs), as the basic processing elements, *F-PE*, to organize a systolic array. Based on the principle of sub-multiplier architecture designs, sixteen 2-bit multipliers in *BitFusion* can create four 4-bit multipliers and one 8-bit multiplier. The major difference is, as shown in Figure 1.1c, *BitFusion* applies the reconfigurable, not preset, left-shifters. Therefore, four 2-bit multipliers created a large *F-PE* to support both 2×8-bit and 4×4-bit multiplications to utilize all BBs with different input widths as 10 and 8 bits. The variable input width complicates the data streaming control designed as a series of multiplexers and registers. In principle, the *BitFusion* presents a multi-precision systolic array, not a multi-precision multiplier. Only the F-PE in in Figure 1.1c is the reconfigurable multiplier. For instance, as shown in Figure 1.1b, c, and d, *BitFusion* works as a 4×4, 1×4, 1×1 systolic array separately. This design limited the scenario of *BitFusion* architecture as the tensor processing unit.

Differing from the designs mentioned above, Liu et al. [14] proposed a *Bitshifter* architecture inspired by the *BISMO* [22] converting the multiplication as the combination of bitwise AND and left-shifting. This is a bit-parallel multi-precision multiplier with a fixed input width. The result of N-bit multiplication, $A \times B = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 2^{i+j} a_i b_j$, is the sum of $2^{i+j} a_i b_j$. a_i and b_j are the bit values of A and B, $a_i b_j$ is the bitwise AND, and 2^{i+j} can be converted as the preset left-shifting. Therefore, as shown in Figure 1.5, *Bitshifter* architecture computes all $a_i b_j$ with bitwise AND first, then filters the unnecessary results with the mask for different precision and applies the corresponding left-shifting to compute partial products.

B. Comparison between the BitSys and Previous Works

Considering the motivation in section IA, we target to explore a multi-precision multiplier design to speed up the computation of mixed-precision QNN models on hardware. To this end, we exclude the bit-serial multiplier scheme in our scope because of its long computation latency. To simplify the data steaming control and deploy our multiplier in variable scenarios of the existing hardware designs, such as the systolic array, single-layer accelerator, etc., we have not adopted the architecture similar to the *BitFusion* and works of Li et al. [25] and Dai et al. [26]. Therefore, our *BitSys* architecture presented a bit-parallel and input-width-fixed multi-precision multiplier design, inspired by *BitShifter* [14] and *BitFusion* [16] by converting the multiplication with bitwise operation with partial product mask and computing them with a systolic array. Table III compared it with related works. In this table, $\surd$ and $\times$ mean the selected features, like available precision, are applied in the corresponding works or not:

- Both two inputs of *BitSys* support multi-channel reconfiguration for variable precision. The work of Guo et al. [18] only supports 1/2-channel $2N/N \times M$-bit multiplication.
- Our work supports accurate computing, not the approximate designs of Neda et al. [19] and Guo et al. [18].
- Shun et al. [20] proposed an accurate multi-precision multiplier based on *Radix-4 Booth* multiplier. However, it is designed for 8/16/24/32-bit multiplication, which is unsuitable for the 1/2/4/8-bit multiplication we targeted for low-precision QNN models.
- Pfänder et al. [21] extended the work of Shun et al. [20] as serial processing to reduce resource consumption. In contrast to this work, *BitSys* adopts the bit-parallel architecture to speed up computation in hardware accelerators.
- *PIR-DSP* [15] focuses on designing multi-precision multipliers based on DSP slices of FPGA. However, the input widths of *DSP48*/*DSP58* resources in *Xilinx* FPGA are wider than 1/2/4/8-bit QNN models. Meanwhile, DSP slices cannot process the XNOR multiplication in BNN. Therefore, DSP slices are inefficient and unsuitable in designing our *BitSys* architecture.



8 bits x 8 bits Multi-Precision Multiplication

P_{i+j}	$a_i b_j$ ($i = 0 \to 7,\ j = 0 \to 7$)								left bitshift 1bit	2bit	4bit	8bit	sum
P_0	a_0b_0								0	0	0	0	**0**
P_1	a_1b_0	a_0b_1							1	1	1	1	**1**
P_2	a_0b_2	a_1b_1	a_2b_0						0+**2**	2	2	2	**2**
P_3	a_3b_0	a_2b_1	a_1b_2	a_0b_3					1+**2**	3	3	3	**3**
P_4	a_4b_0	a_3b_1	a_2b_2	a_1b_3	a_4b_0				0+**4**	0+**4**	4	4	**4**
P_5	a_5b_0	a_4b_1	a_3b_2	a_2b_3	a_1b_4	a_0b_5			1+**4**	1+**4**	5	5	**5**
P_6	a_6b_0	a_5b_1	a_4b_2	a_3b_3	a_2b_4	a_1b_5	a_0b_6		0+**6**	2+**4**	6	6	**6**
P_7	a_7b_0	a_6b_1	a_5b_2	a_4b_3	a_3b_4	a_2b_5	a_1b_6	a_0b_7	1+**6**	3+**4**	7	7	**7**
P_8	a_7b_1	a_6b_2	a_5b_3	a_4b_4	a_3b_5	a_2b_6	a_1b_7		0+**8**	0+**8**	0+**8**	8	**8**
P_9	a_7b_2	a_6b_3	a_5b_4	a_4b_5	a_3b_6	a_2b_7			1+**8**	1+**8**	1+**8**	9	**9**
P_{10}	a_7b_3	a_6b_4	a_5b_5	a_4b_6	a_3b_7				0+**10**	2+**8**	2+**8**	10	**10**
P_{11}	a_7b_4	a_6b_5	a_5b_6	a_4b_7					1+**10**	3+**8**	3+**8**	11	**11**
P_{12}	a_7b_5	a_6b_6	a_5b_7						0+**12**	0+**12**	4+**8**	12	**12**
P_{13}	a_7b_6	a_6b_7							1+**12**	1+**12**	5+**8**	13	**13**
P_{14}	a_7b_7								0+**14**	2+**12**	6+**8**	14	**14**

1-bit Sub-Partial Products Mask

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_7	1	0	0	0	0	0	0	0
b_6	0	1	0	0	0	0	0	0
b_5	0	0	1	0	0	0	0	0
b_4	0	0	0	1	0	0	0	0
b_3	0	0	0	0	1	0	0	0
b_2	0	0	0	0	0	1	0	0
b_1	0	0	0	0	0	0	1	0
b_0	0	0	0	0	0	0	0	1

2-bit Sub-Partial Products Mask

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_7	1	1	0	0	0	0	0	0
b_6	1	1	0	0	0	0	0	0
b_5	0	0	1	1	0	0	0	0
b_4	0	0	1	1	0	0	0	0
b_3	0	0	0	0	1	1	0	0
b_2	0	0	0	0	1	1	0	0
b_1	0	0	0	0	0	0	1	1
b_0	0	0	0	0	0	0	1	1

4-bit Sub-Partial Products Mask

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_7	1	1	1	1	0	0	0	0
b_6	1	1	1	1	0	0	0	0
b_5	1	1	1	1	0	0	0	0
b_4	1	1	1	1	0	0	0	0
b_3	0	0	0	0	1	1	1	1
b_2	0	0	0	0	1	1	1	1
b_1	0	0	0	0	1	1	1	1
b_0	0	0	0	0	1	1	1	1

Fig. 2: 1/2/4/8 Channels 8/4/2/1 bits Multiplication and Corresponding Partial Products Masks

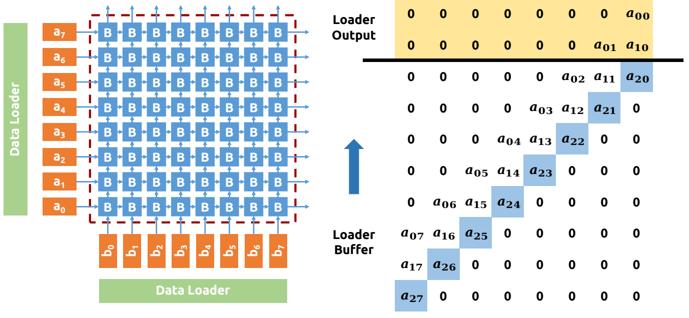

Data Loader
a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
B
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
Data Loader
Loader Output
0 0 0 0 0 0 0 a_{00}
0 0 0 0 0 0 a_{01} a_{10}
0 0 0 0 0 a_{02} a_{11} a_{20}
0 0 0 0 a_{03} a_{12} a_{21} 0
0 0 0 a_{04} a_{13} a_{22} 0 0
0 0 a_{05} a_{14} a_{23} 0 0 0
0 a_{06} a_{15} a_{24} 0 0 0 0
a_{07} a_{16} a_{25} 0 0 0 0 0
a_{17} a_{26} 0 0 0 0 0 0
a_{27} 0 0 0 0 0 0 0
Loader Buffer

Fig. 3: Bitwise Systolic Array (left) and Input Loader (right)

- As shown in Figure 1.4 and Figure 1.5, we fused the AND array and Mask array in *Bitshifter* [14] as a bitwise systolic array inspired by *BitFusion* [16] for higher throughputs. The left shifters and output generation stages for different precision are fused as output generation pipelines in *BitSys*.
- The processing elements in *BitSys* execute 1-bit operations, supporting higher clock frequency with lower critical path delay. Moreover, the XNOR multiplication in *Bitshifter* [14] is computed in an individual module. We fused it in our 1-bit processing elements to save the hardware resources.
- We implemented the single-layer accelerator and systolic array accelerator based on *BitSys* to show its potential to be applied in different designs.

III. Implementation

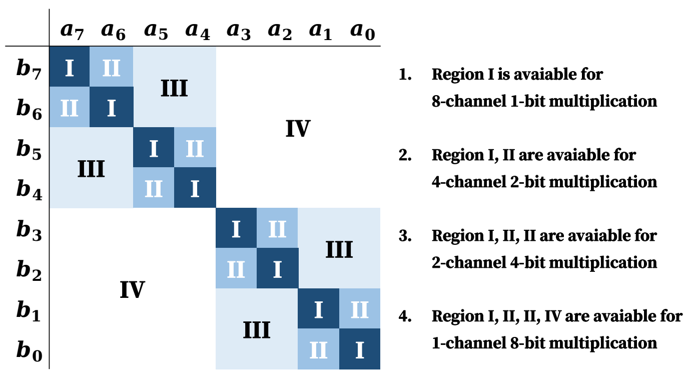
A. Mathematics Principle

For N-bit multiplication, $A \times B = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 2^{i+j} a_i b_j$, $(0 \leqslant i,\ j \leqslant n-1)$, a_i and b_j are the bit value of A and B, 2^{i+j} can be replaced as left shifting, $\ll (i+j)$, and $a_i b_j$ is the bitwise AND. As shown in Equation 1, if we define $a_i b_j$ as sub-partial products, for N-bit multiplication, we can reorganize all sub-partial products as $2N-1$ groups. Each group applies the same left-shifting value M, $(0 \leqslant M \leqslant 2N-2)$. Therefore, we can define the sum of one group as the partial products, $P_M = \sum_{M=i+j} a_i b_j$. Therefore, all $a_i b_j$ are organized as the triangular-aligned structure shown in Figure 2. The bold parts in Equation 1 represent the sign bit with $\pm$ in the multiplication. When it is negative, the multiplication is signed. Therefore, by switching to add or subtract the AND results of $a_k b_{n-1}$ and $a_{n-1} b_k$ $(0 \leqslant k < n-1)$ from partial products, the multiplication can be reconfigured as signed/unsigned computing. After applying the corresponding left-shifting value for each partial product, their sum is the product of N-bit multiplication.

Based on the basic mathematics principle mentioned above, as shown in Figure 2, we extend it for runtime reconfigurable multi-channel multi-precision multiplication: Using 8×8-bit multi-precision multiplication as an example, P_{i+j} $(0 \leqslant i \leqslant 7,\ 0 \leqslant j \leqslant 7)$ are the partial products in this computation, which are the sum of corresponding sub-partial products, $a_i b_j$, shown in the same row of P_{i+j} in the second column of Figure 2. For different precision, three sub-partial product masks shown in Figure 2 select the desired sub-partial products, $a_i b_j$, for 8/4/2 channel 1/2/4-bit multiplications. For instance, for the dual-channel 4×4-bit multiplication, two green squares in the 4-bit sub-partial products mask of Figure 2 select the desired $a_i b_j$ in computation. The filtered $a_i b_j$ are set as zero, and one green square in the 4-bit sub-partial products mask selects the $a_i b_j$ for one channel. Based on the same principle, the four orange and eight blue squares in 2/1-bit masks select the desired sub-partial products for corresponding 4/8 channels. All $a_i b_j$ are used to compute single-channel 8×8-bit multiplication. Therefore, we can compute all sub-partial products first, reconfigure the mask in runtime to filter the undesired $a_i b_j$ for different precision and channels, and then compute the sum of filtered $a_i b_j$ as partial products, P_{i+j}. Considering the lower hardware utilization efficiency when more sub-partial products are filed as zero in lower precision, compared with the disabled sub-multipliers in the previous works shown in the Figure 1.2 of section II, this is a common trade-off to achieve the multi-precision reconfiguration for related bit-parallel input-width-fixed multiplier designs.

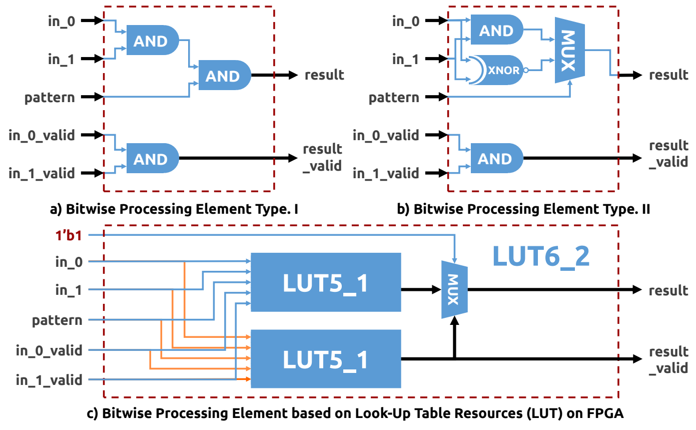
$$
\begin{aligned}
A \times B =& \langle a_{n-1} a_{n-2} \ldots a_1 a_0 \rangle_{bin} \times \langle b_{n-1} b_{n-2} \ldots b_1 b_0 \rangle_{bin} \\
=& (\pm \mathbf{2^{n-1} a_{n-1}} + 2^{n-2} a_{n-2} + \ldots + 2^1 a_1 + 2^0 a_0) \\
& \times (\pm \mathbf{2^{n-1} b_{n-1}} + 2^{n-2} b_{n-2} + \ldots + 2^1 b_1 + 2^0 b_0) \\
=& [(a_{n-1} b_{n-1}) \ll 2n-2] \\
& + [(\pm \mathbf{a_{n-1} b_{n-2}} + \pm \mathbf{a_{n-2} b_{n-1}}) \ll 2n-3] \\
& + \ldots \\
& + [(a_1 b_0 + a_0 b_1) \ll 1] \\
& + [(a_0 b_0) \ll 0]
\end{aligned}
\tag{1}
$$

After we get the value of partial products, P_{i+j}, the multiplier needs to apply the corresponding left shifting to P_{i+j} and sum them as the multi-channel results. Therefore, as shown in the *left bitshift* column of Figure 2, for example, when the multiplier executes 8-channel 1-bit multiplication, each channel needs two partial products and applies 0/1-bit left shifting separately. For instance, the result in the first channel of 1-bit multiplication is $(P_0 \ll 0) + (P_1 \ll 1)$. Actually, the 1-bit operation only needs one P_i in each channel, such as P_0 for the first channel. However, to keep the output as 8-channel-2-bit, the P_1 is used



a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
I II III IV
1. Region I is avaiable for 8-channel 1-bit multiplication
2. Region I, II are avaiable for 4-channel 2-bit multiplication
3. Region I, II, II are avaiable for 2-channel 4-bit multiplication
4. Region I, II, II, IV are avaiable for 1-channel 8-bit multiplication

Fig. 4: Bitwise Processing Element Location in Systolic Array



in_0
in_1
pattern
in_0_valid
in_1_valid
AND
AND
AND
result
result _valid
a) Bitwise Processing Element Type. I
AND
XNOR
MUX
AND
result
result _valid
b) Bitwise Processing Element Type. II
1'b1
LUT6_2
LUT5_1
LUT5_1
MUX
c) Bitwise Processing Element based on Look-Up Table Resources (LUT) on FPGA

Fig. 5: Design of Bitwise Processing Element

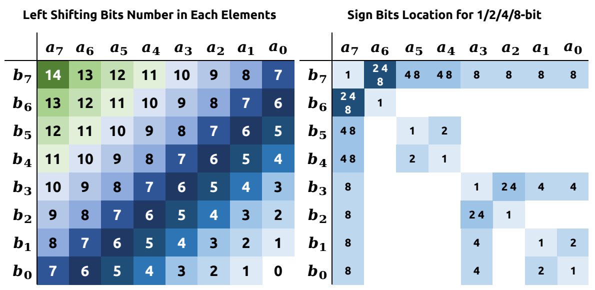
as a placeholder, and its a_1b_0 and a_0b_1 are filtered as 0 by 1-bit sub-partial product masks. Based on the same principle, for instance, we can infer that P_3 is also a placeholder partial product for the 1st channel of 2-bit multiplication. Considering the total output width of this 8-bit multiplier is 16 bits, the output widths of one channel in 1/2/4-bit modes are 2/4/8 bits. Therefore, in the final output, each 2/4/8-bit output from the i-th channel in 1/2/4-bit multiplication needs a channel offset by left-shifting to $(i-1)\times 2$, $(i-1)\times 4$, and $(i-1)\times 8$ bits to avoid conflict with the $(i-1)$-th channel. As shown in the *sum* of *left bitshift* column in Figure 2, for each partial product, P_k $(0 \leqslant k \leqslant 14)$, the sums of partial product left shifting (black numbers in *left bitshift* column) and channel offset left shifting (red numbers in *left bitshift* column) are always k in all 1/2/4/8-bit multiplication modes. Therefore, differing from the individual three left shifting stages in *Bitshifter* architecture [14] shown in Figure 1.4, our *BitSys* applied the same left shifting for each partial product in all 1/2/4/8-bit modes. In conclusion, the computation of the runtime reconfigurable multi-precision multiplication in our work can be converted into four steps:

1) Computing all a_ib_j $(0 \leqslant i \leqslant n-1,\ 0 \leqslant j \leqslant n-1)$.
2) Filtering to get the desired a_ib_j with corresponding sub-partial products mask for different precision.
3) Computing the partial products, P_k $(0 \leqslant k \leqslant 2n-2)$, and applying k-bit left shifting.
4) Computing the sum of P_k as the final output.

B. Bitwise Systolic Array Architecture for Multi-precision Multiplier

To execute the first two steps mentioned above, we implemented a bitwise systolic array as shown in Figure 3 (left) and the input loader as shown in Figure 3 (right). The input loader works to prepare the inputs for the bitwise systolic array. For instance, in an 8-bit multiplier, the input loader implements a tiny FIFO buffer with eight 8-bit registers, loading one new input in the diagonal (blue bits) and pushing the data from the bottom to the top as the loader outputs (yellow part). One bitwise systolic array requires two input loaders. The bitwise systolic array we implemented in Figure 3 (left) consists of bitwise processing elements, which fused the sub-partial products mask and 1-bit arithmetic operations. Considering the multiplication in the BNN presented in FINN [3, 4] is the XNOR operation, which represents the -1 as *'0'* and +1 as *'1'*, we need two kinds of bitwise processing elements as shown in Figure 5a and Figure 5b: Type.I element switches between 1-bit XNOR and AND operation for 1-bit or 2/4/8-bit multiplication. Type.II element switches between 1-bit AND operation and zero output according to the sub-partial product mask in variable precision. Figure 4 presents the location mapping of bitwise processing elements and when they are available for different precision according to the sub-partial product masks: Type.I elements are located in *Region I* and Type.II elements are located in *Region II/III/IV*. For instance, when the multiplier works in 1-bit mode, the processing elements in *Region I* output the results of 1-bit XNOR, and other elements output 0. When precision is 4-bit, the processing elements in *Region I/II/III* output the results of 1-bit AND, and other elements output 0. One pattern signal generated according to the current precision controls the output switching of one bitwise processing element. Furthermore, because when two inputs of the 1-bit XNOR are '0', the output is '1', the bitwise processing element needs input and output a valid signal for the following adder to avoid mistake accumulation when no input is loaded. Therefore, as shown in Figure 5c, we define one bitwise processing element as 6-bit input and 2-bit output module: 2-bit input, 2-bit input valid, 1-bit pattern switching, 1-bit input is always '1' to enable 2-bit output, 1-bit output, and 1-bit output valid signal. Therefore, one bitwise processing element for both types can be implemented as one *LUT6_2* primitive in *Xilinx* FPGA.

For the second two steps in the computation of our *BitSys*, the multiplier needs to compute the value of partial products, P_k $(0 \leqslant k \leqslant 2n-2)$, apply the left-shifting to them, and add all P_k as the final output. As shown in Figure 6 (left), numbers in this figure are the left-shifting bits applied to the outputs of their located bitwise processing elements. Therefore, the sum of the bitwise processing element results with the same left-shifting bits, which are in the same diagonal, is a partial product. Considering the signed multiplication in Equation 1, the numbers in Figure 6 (right) represent that, in which precision, the outputs of bitwise processing elements they located need to be subtracted from partial products. For instance, a_7b_6 needs to be subtracted in 2/4/8-bit multiplication because a_7 is a sign bit in this precision. a_7b_7 needs to be subtracted in 1-


Left Shifting Bits Number in Each Elements

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_7	14	13	12	11	10	9	8	7
b_6	13	12	11	10	9	8	7	6
b_5	12	11	10	9	8	7	6	5
b_4	11	10	9	8	7	6	5	4
b_3	10	9	8	7	6	5	4	3
b_2	9	8	7	6	5	4	3	2
b_1	8	7	6	5	4	3	2	1
b_0	7	6	5	4	3	2	1	0

Sign Bits Location for 1/2/4/8-bit

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_7	1	2 4 8	4 8	4 8	8	8	8	8
b_6	2 4 8	1						
b_5	4 8		1	2				
b_4	4 8		2	1				
b_3	8				1	2 4	4	4
b_2	8				2 4	1		
b_1	8				4		1	2
b_0	8				4		2	1

Fig. 6: Left-Shifting of Diagonal & Signed Elements

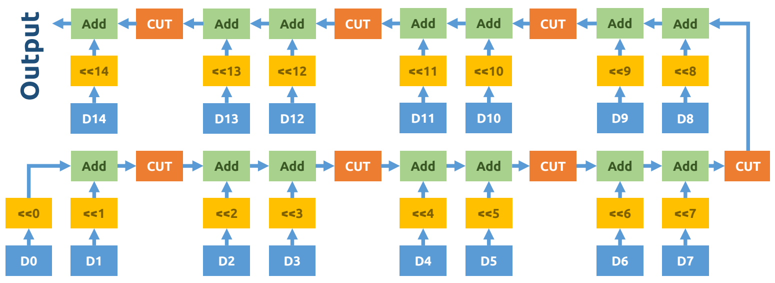


Fig. 7: Design of Output Generator Pipeline

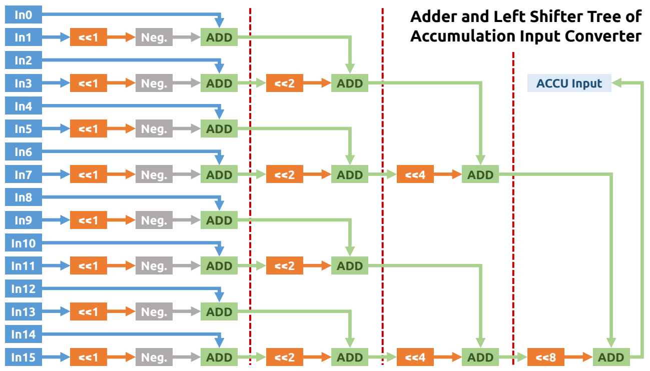


Fig. 8: Multi-Precision Accumulator Input Converter

bit multiplication because the XNOR output is signed output, representing -1 as '0' and +1 as '1'. Moreover, because both a_7 and b_7 are sign bits in 2/4/8-bit multiplication, $a_7 b_7$ does not need to be subtracted. After finishing the computation of partial products, P_k, our *BitSys* multiplier loads them as the inputs, D_k, of the output generator pipeline shown in Figure 7 to apply k-bit left-shifting and sum the left-shifted partial products as final output. Considering the sum of signed partial products generates the carry bits in computation and influences the result in the next channel, we insert the carry-cutter modules in the output generator pipeline to limit the output width. For instance, in 1-bit multiplication, all carry-cutters are enabled to limit the output width of 8 channels; in 2-bit multiplication, only the carry-cutters after $D_{3,7,11}$ are enabled to limit the output width of 4 channels. Because the bitwise systolic array generates the partial product from P_0 to P_{14} sequentially and executes multiple computations simultaneously, our output generator pipeline is designed for pipelined parallel processing. For instance, in the 1st cycle, the bitwise systolic array outputs the D_0 of MUL_0, and the output generator pipeline left-shifts it to 0-bit. In the 2nd cycle, the bitwise systolic array outputs the D_0 of MUL_1 and D_1 of MUL_0. The output generator pipeline applies the 0/1-bit left-shifting on them separately and adds the D_0 and 1-bit left-shifted D_1 of MUL_0 together for the next step.

C. Single-Layer and Systolic Array Accelerator Implementation based on BitSys

To evaluate the multiplier based on our *BitSys* architecture, we implemented one single-layer accelerator and one systolic array accelerator as shown in Figure 9 and Figure 10. Both accelerators consist of four components: 1) Input Loader (Orange), 2) *BitSys* Multiplier (Yellow), 3) Accumulator (Green), and 4) Activation Module (Gray). Both accelerators contain 64 multipliers. The single-layer accelerator implements these multipliers as 8 neurons. Each neuron consists of 8 multipliers and 16 input loaders. The systolic array accelerator implements these multipliers as an 8×8 systolic array with 16 input loaders. Both single-layer and systolic array accelerators implemented a state machine to control the inference of network models, which loads and stores the layer settings, like input length and precision, in a FIFO of FPGA. To reconfigure the multipliers for different layer precision, the state machine uses three clock cycles to load the precision data from FIFO and rewrite the registers for multiplier settings.

Considering the output of *BitSys* multiplier is multi-channel, if we implement the corresponding accumulator and activation module for all channels, one multiplier needs to connect with eight accumulators and eight activation modules at maximum (for 1-bit mode). However, when the multiplier works on higher precision, the required accumulators and activation modules are less than 1-bit mode because of fewer output channels, leading to low hardware efficiency. Therefore, we connect each multiplier with one accumulator and activation module in both accelerators. To this end, we implemented a tree-structure-based pipelined input converter shown in Figure 8 for the accumulator to sum all channels of multiplier output: Multiplier outputs 16-bit data to this input converter as in_{0-15}. The left-shifters (Orange) in Figure 8 apply the bit weight, 2^i, to in_i by passing through four shifting-and-adding layers in this tree structure. Because if A is a signed value, $A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$. a_i is the bit value of A and a_{n-1} is the sign bit. Therefore, we insert one value inverter (Neg. Block) in the first shifting-and-adding layer to negate the left-shifted sign bit. For different precision, different numbers of value inverters are enabled. For instance, for 8-bit dual-channel input, only the inverters connected with in_7 and in_{15} are enabled. Furthermore, we applied the *Multi-Thresholds* activation function from FINN [3, 4] to design our activation module, which fused the activation and output re-quantization as multi-thresholds. This activation function required 1/3/15/255 thresholds to generate 1/2/4/8-bit output. The number of thresholds smaller than the accumulator output is the final output. Therefore, to reduce resource consumption and improve hardware efficiency, we only implement one comparator in each activation module, sequentially loading the thresholds to compare with the accumulator output.

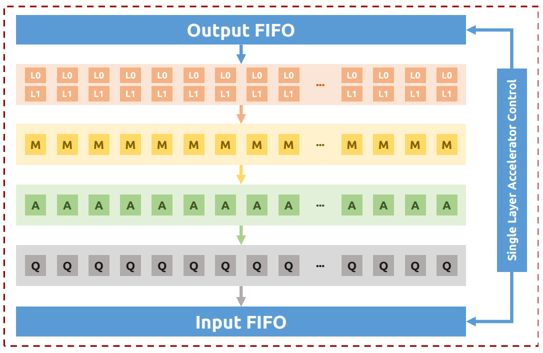


Fig. 9: *BitSys*-based Single-Layer Accelerator

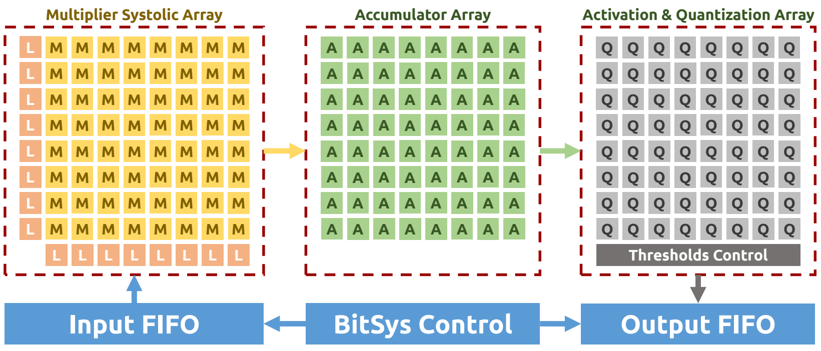


Fig. 10: *BitSys*-based Systolic-Array Accelerator

IV. Evaluation

A. Experiment Setup

We evaluate the Multiplier (MUL), Multiply-Accumulator (MAC), and accelerator instances of our *BitSys* architecture on *Ultra96-V2* FPGA platform (*Zynq UltraScale+ ZU3EG*). Considering the discussion in section II-B, we selected the works of Liu et al. [14] as the baseline. All accelerators are evaluated by the TFC models we trained as the same as the network used in [14] with the *Brevitas* and *MNIST* dataset, which have been introduced in section I.

B. Multiplier and Multiply-Accumulator Comparison

Table IV shows the implementation results from *Vivado*: we implemented six instances, including one pure-Verilog-designed and one LUT-primitive-optimized variant, *BitSys-base* and *BitSys-LUT*, for 1/2/4/8-bit signed/unsigned MULs and MACs of our *BitSys*. As the baseline, we implement the MUL and MAC instances of *Multiplier-Tree* and *Bitshifter* from Liu et al. [14] as *MTee-base* and *Bitshifter-base*, supporting 1/2/4/8-bit signed/unsigned reconfigurable multiplication, and insert the registers between the sub-multipliers of *Multiplier-Tree* and AND/Mask/Shifting stages of *Bitshifter* as shown in Figure 1.2 and Figure 1.4 to create their pipelined instances, *MTee-pipe* and *Bitshifter-pipe*, for higher clock frequency.

For MUL comparison, two *Bitshifter* instances consume fewer LUTs with less total path delay than *Multiplier-Tree* instances. The pipelined *Multiplier-Tree* and *Bitshifter* instances, *MTee-pipe* and *Bitshifter-pipe*, consume more LUTs and FFs than their basic instances, *MTee-base* and *Bitshifter-base*, with significant decrease in total path delay. Compared with these baseline instances, both MULs of *BitSys* have the lower total path delay: the lowest total path delay belongs to *BitSys-LUT*, which is 65.36%, 44.97%, 62.18%, and 33.51% of *Bitshifter-pipe*, *Bitshifter-base*, *MTee-pipe*, and *MTee-base*. The LUT-primitive-optimization of *BitSys-LUT* instances decreased the resource consumption and total path delay compared with *BitSys-base*. The LUT consumption of *BitSys-LUT* shows no advantages with the same or higher numbers than *MTee-base*, *Bitshifter-base*, and *Bitshifter-pipe* as 100.00%, 101.45%, and 103.86%.However, we discussed the *Area Delay Products* (ADPs) in Table IV, which are the products between LUT consumption and total path delay. The lowest ADP of *BitSys* instances implies that our work achieved an efficient design with a good balance between performance and resource utilization. Based on the post-implementation timing simulation in *Vivado*, we analyzed the power consumption of all MUL instances with 16000 times random multiplication under the highest available clock shown in Table IV. Our *BitSys* instances have the highest power consumption. However, their *Power Delay Products* (PDPs) are lower than the other four instances, which are the products between power and total path delay. This means that our design has better power efficiency and achieves a good balance between minimizing power usage and maximizing speed. Moreover, the *Computation Cycles* column in Table IV shows that *BitSys* architecture has a longer pipeline path in computation than other instances, which explains the low total path latency and high FF consumption of our work. Differing from the MUL instances, MAC instances of *Multiplier-Tree* cost less LUT than *Bitshifter* because we fused the accumulator input converter design of *Multiplier-Tree* by summing the results of sub-multipliers and passing it to higher precision multipliers. The output of this multiplier is the sum of all channels. Following the same trend as MULs, The MAC instances of our *BitSys* consume more resources and power than other instances with less total path delay, lower ADP, and lower PDP. The low power consumption of MACs compared with MULs is caused by the different testbench and longer pipeline as shown in *Computation Cycles* column. We simulated the MACs with 4096 times random multiplication and accumulation for each precision. Before starting the next round of computation for another precision, MACs need to wait to finish the accumulation of the current precision. In summary, our *BitSys* architecture has a better design optimization between the balance of hardware consumption, power usage, and processing speed, supporting the highest clock frequency with the lowest total path delay.

C. Neural Network Accelerator Comparison

Table V is the implementation and real measurement results of accelerators we implemented on *Ultra96-V2* platform, including six single-layer accelerator based on *Vivado IP*, *MTee-base*, *MTee-pipe*, *Bitshifter-base*, *Bitshifter-pipe*, and *BitSys-LUT*, and one systolic array accelerator based on *BitSys-LUT* to compare the difference between single-layer accelerator architecture and systolic array architecture. The column of

TABLE IV: Resource Consumption of Multipliers (MUL) and Multiply-Accumulators (MAC)

Design	Type	Precision	Instance Setting Signed/Unsigned	Instance Setting Accurate/Approximate	Resource Consumption LUT	Resource Consumption FF	Frequency	Total Delay (ns)	Area-Delay Products	Dynamic Power (mW)	PDP ($mW \times ns$)	Compuation Cycles BIN	2-bit	4-bit	8-bit
MTree-base [14]	MUL	1/2/4/8	Both	Accurate	383	42	$250MHz$	3.820	1463.06	87	332.34	1	1	1	1
MTree-pipe					429	282	$375MHz$	2.282	978.98	125	285.25	5	5	5	5
Bitshifter-base [14]					345	37	$300MHz$	3.156	1088.82	107	337.69	1	1	1	1
Bitshifter-pipe					337	339	$375MHz$	2.171	731.627	122	264.86	1	9	9	9
BitSys-base					416	463	$500MHz$	1.433	596.128	156	223.55	22	22	22	22
BitSys-LUT					350	525	$500MHz$	1.419	496.65	159	225.62	22	22	22	22
MTree-base [14]	MAC	1/2/4/8	Both	Accurate	398	199	$250MHz$	3.397	1352.01	79	268.36	6	6	6	6
MTree-pipe					495	388	$250MHz$	2.828	1399.86	102	288.46	10	10	10	10
Bitshifter-base [14]					505	198	$300MHz$	3.084	1425.27	102	314.57	6	6	6	6
Bitshifter-pipe					538	506	$375MHz$	2.164	1164.23	109	235.88	6	14	14	14
BitSys-base					597	633	$375MHz$	2.072	1236.98	103	213.42	27	27	27	27
BitSys-LUT					541	689	$500MHz$	1.716	928.36	134	229.94	27	27	27	27

TABLE V: Resource Consumption of Previous and BitSys Accelerators on *Ultra96V2* FPGA Platform

Design	Type	Precision	LUT Number	LUT Rate	FF Number	FF Rate	BRAM Number	BRAM Rate	Frequency	Latency/μs
Vivado IP [19]	Single-Layer	8/8/8/8 1/2/4/8	24090	34.14%	22175	15.71%	135	62.50%	$150MHz$	137.654 131.059
MTree - base [19]	Single-Layer	1/2/4/8	37020	52.47%	22500	15.94%	138	63.89%	$100MHz$	69.27
Bitshifiter - base [19]	Single-Layer	1/2/4/8	42952	60.87%	22486	15.93%	138	63.89%	$125MHzz$	56.658
MTree - pipe	Single-Layer	1/2/4/8	47163	66.84%	42100	29.83%	138	63.89%	$150MHz$	48.443
Bitshifiter - pipe	Single-Layer	1/2/4/8	50212	71.16%	50393	35.71%	138	63.89%	$150MHz$	48.799
BitSys - LUT	Single-Layer	1/2/4/8	46570	66.00%	54352	38.51%	138	63.89%	$150MHz$	49.057
BitSys - LUT	Systolic Array	1/2/4/8	44468	63.02%	64176	45.48%	139.5	64.58%	$250MHz$	36.741

Latency/μs is the single frame inference delay by averaging the total inference latency of 1000 MNIST inputs.

Vivado IP-based single-layer accelerator consumes the least LUTs and FFs with the longest single-frame inference latency compared with other accelerators because it does not support multi-precision multiplication. Both *Bitshifter* accelerators cost more hardware resources than *Multiplier-Tree*. The pipelined accelerator of both *Multiplier-Tree* and *Bitshifter* support higher clock frequency than their basic accelerators. According to the total path delay of MUL and MAC shown in Table IV, in principle, the single-layer accelerators of *BitSys-LUT* and *Bitshifter-pipe* should support higher clock frequency than *MTee-base*, *MTee-pipe*, and *Bitshifter-base*. However, because the single-layer accelerator contains one more complex state machine than the systolic array for data streaming control to load the activations and weights from DDR to neurons and schedule the computation with a limited number of neurons and multipliers, $150MHz$ is the highest frequency that can be supported in our current single-layer accelerator architecture. The systolic array accelerator implemented with *BitSys-LUT* consumes 95.49% of LUTs but 118.07% of FFs with $250MHz$ compared with its single-layer accelerator. Comparing the two structures shown in Figure 9 and Figure 10, systolic array accelerator of *BitSys-LUT* requires much fewer input loaders that single-layer accelerator, which causes the LUT consumption decreasing shown in Table V. For the average inference latency, all accelerators implemented with multi-channel multi-precision multiplier have a high speed-up compared with *Vivado IP*-based accelerator. The single-layer accelerators of *Bitshifter-pipe* and *MTee-pipe* are 116.1% and 142.99% faster than *Bitshifter-base* and *MTee-base* because of higher clock frequency. Compared with the single-layer accelerator of *Bitshifter-pipe* and *MTee-pipe*, and *BitSys-LUT*, *BitSys-LUT* instance is 0.53% and 1.25% slower than *Bitshifter-pipe* and *MTee-pipe* with same frequency. Considering the *Computation Cycles* shown in Table IV, *BitSys-LUT* instance infers slower because of its long pipeline path. For the same reason, the single-layer accelerator of *Bitshifter-pipe* is 0.73% slower than *MTee-pipe*. However, because the systolic array structure simplified the data streaming control in a single-layer accelerator, the inference latency of *BitSys-LUT* benefits both from the higher clock frequency and denser computation, which can highly efficiently utilize the fully pipelined design in our *BitSys* architecture. Therefore, the systolic array accelerator of *BitSys-LUT* supports $250MHz$ and is 356.71%, 188.54%, 148.77%, 131.85%, 132.82%, and 133.52% faster than the single-layer accelerators of *Vivado IP*, *MTee-base*, *MTee-pipe*, *Bitshifter-base*, *Bitshifter-pipe*, and *BitSys-LUT* with mixed-precision TFC network.

V. Conclusion

In this manuscript, we present one multiplier design based on fully pipelined bitwise systolic array architecture, *BitSys*, supporting the runtime reconfigurable multi-precision multi-channel multiplication. The evaluation shows that our *BitSys* architecture has a low critical path delay to support higher clock frequency compared with previous works. In the acceleration of the mixed-precision network model, our work is more than 131.85% faster than original *Multiplier-Tree* and *Bitshifter* architecture and about 356.71% faster compared with Vivado-IP-based accelerator. For our future work, we plan to explore the ASIC implementation of our *BitSys* architecture with emerging memory technologies, such as *Racetrack Memory* (RTM).

VI. Acknowledgements

This work is supported by the *Center for Scalable Data Analytics and Artificial Intelligence* (ScaDS.AI) *Dresden/Leipzig* and *Deutsche Forschungsgemeinschaft* (DFG) under the *X-ReAp* project (Project number 380524764).

References

[1] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. "Research on NVIDIA Deep Learning Accelerator". In: *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. 2018, pp. 192–195.

[2] Vinod Kathail. "Xilinx Vitis Unified Software Platform". In: *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, 173–174.

[3] Yaman Umuroglu et al. "Finn: A framework for fast, scalable binarized neural network inference". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017, pp. 65–74.

[4] Michaela Blott et al. "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11.3 (2018), pp. 1–23.

[5] Farah Fahim et al. "hls4ml: An Open-Source Co-Design Workflow to Empower Scientific Low-Power Machine Learning Devices". In: *Research Symposium on Tiny Machine Learning*. 2021.

[6] Yaman Umuroglu et al. "LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications". In: *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2020, pp. 291–297.

[7] Jiang Su et al. "Accuracy to throughput trade-offs for reduced precision neural networks on reconfigurable logic". In: *International Symposium on Applied Reconfigurable Computing*. Springer. 2018, pp. 29–42.

[8] Kuan Wang et al. "HAQ: Hardware-Aware Automated Quantization With Mixed Precision". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[9] Weihan Chen, Peisong Wang, and Jian Cheng. "Towards Mixed-Precision Quantization of Neural Networks via Constrained Optimization". In: *CoRR* abs/2110.06554 (2021). arXiv: 2110.06554.

[10] Chen Tang et al. "Mixed-Precision Neural Network Quantization via Learned Layer-Wise Importance". In: *European Conference on Computer Vision*. Springer. 2022, pp. 259–275.

[11] Alessandro Pappalardo. *Xilinx/brevitas*. 2023.

[12] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[13] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[14] Yuhao Liu et al. "High Flexibility Designs of Quantized Runtime Reconfigurable Multi-Precision Multipliers". In: *IEEE Embedded Systems Letters* (2023), pp. 1–1.

[15] SeyedRamin Rasoulinezhad et al. "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks". In: *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2019, pp. 35–44.

[16] Hardik Sharma et al. "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 2018, pp. 764–775.

[17] Yuhao Liu, Salim Ullah, and Akash Kumar. "BitSys: Bitwise Systolic Array Architecture for Multi-precision Quantized Hardware Accelerators". In: *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2024, pp. 220–220.

[18] Chuliang Guo et al. "A Reconfigurable Approximate Multiplier for Quantized CNN Applications". In: *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2020, pp. 235–240.

[19] Negar Neda et al. "Multi-Precision Deep Neural Network Acceleration on FPGAs". In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2022, pp. 454–459.

[20] Zhou Shun et al. "A VLSI architecture for a Run-time Multi-precision Reconfigurable Booth Multiplier". In: *2007 14th IEEE International Conference on Electronics, Circuits and Systems*. 2007, pp. 975–978.

[21] Oliver A. Pfänder et al. "Configurable Blocks for Multi-precision Multiplication". In: *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*. 2008, pp. 478–481.

[22] Yaman Umuroglu, Lahiru Rasnayake, and Magnus Själander. "Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing". In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2018, pp. 307–3077.

[23] P. Ienne and M.A. Viredaz. "Bit-serial multipliers and squarers". In: *IEEE Transactions on Computers* 43.12 (1994), pp. 1445–1450.

[24] Andrew G. Shafer, Lyndsi R. Parker, and Earl E. Swartzlander. "The fully-serial pipelined multiplier". In: *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. 2011, pp. 1817–1822.

[25] Kai Li et al. "A Precision-Scalable Energy-Efficient Bit-Split-and-Combination Vector Systolic Accelerator for NAS-Optimized DNNs on Edge". In: *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. 2022, pp. 730–735.

[26] Liuyao Dai et al. "An Energy-Efficient Bit-Split-and-Combination Systolic Accelerator for NAS-Based Multi-Precision Convolution Neural Networks". In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2022, pp. 448–453.