`WinoGen`: A Highly Configurable Winograd Convolution IP Generator for Efficient CNN Acceleration on FPGA

Mingjun Li[*1], Pengjia Li[*2], Shuo Yin[1], Shixin Chen[1], Beichen Li[2], Chong Tong[2], Jianlei Yang[3], Tinghuan Chen[2], Bei Yu[1]

[1]Chinese University of Hong Kong [2]Chinese University of Hong Kong, Shenzhen
[3]Beihang University

Highlights

- A Chisel-based highly configurable Winograd convolution IP generator, `WinoGen`, is developed via algorithm-architecture co-design approaches.
- A Structured Direct Winograd Convolution (`SDW`) algorithm is proposed as a novel and efficient reformulation of Winograd convolution, enabling `WinoGen` to generate IPs according to a given **arbitrary** tile size and kernel size.
- Highly paralleled and fully pipelined architecture is designed as the IP template. The generated IPs are **compatible with multiple kernel sizes and tile sizes**.
- An architecture optimizer is built based on resource and latency models, enabling `WinoGen` to generate IPs with optimal configurations.

Basic Problem Formulation & Workflow

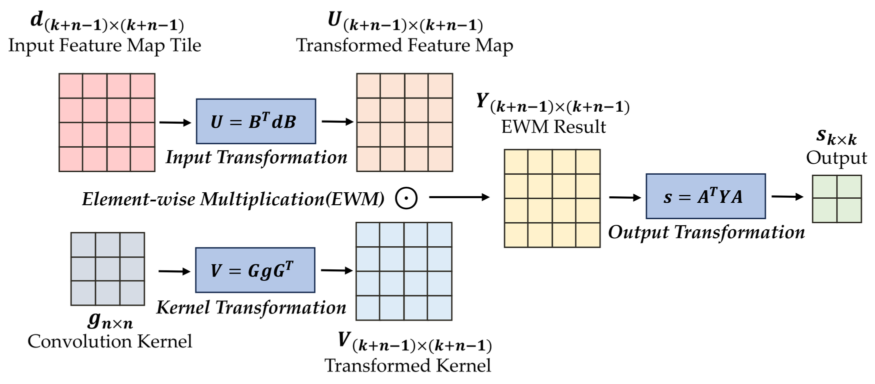

Figure 1. Winograd convolution flow defined by $F(k, n)$.

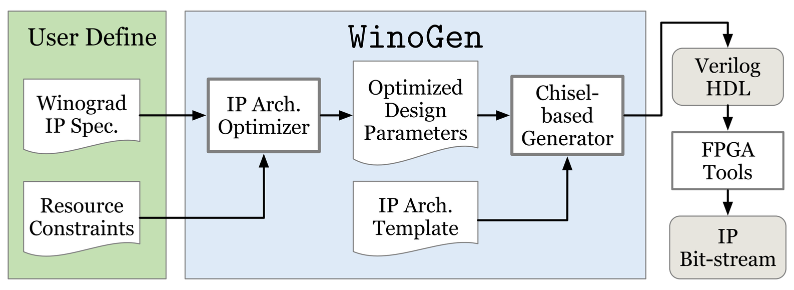

Figure 2. `WinoGen` automation workflow.

Two-dimensional Winograd convolution is defined by $F(k, n)$: $s = A^\top \left[B^\top dB \odot GgG^\top\right] A$, where s is a $k \times k$ output tile, g is an $n \times n$ kernel and d is a $(k+n-1) \times (k+n-1)$ input feature map tile. $\omega = k+n-1$ is denoted as the Winograd filter size.

Structured Direct Winograd Convolution

Structured Direct Winograd Convolution (`SDW`) is composed of **Recursive Winograd Input Transformation** (`RIT`) and **Blockwise Winograd Output Transformation** (`BOT`). It is based on our hardware-friendly Winograd polynomial sequences:

$$m_i(x) = \begin{cases} x, & i = 0, \\ x + (-1)^i \times 2^{\left\lfloor \frac{i-1}{2} \right\rfloor}, & 0 < i < \omega - 1, \\ x - \infty, & i = \omega - 1, \end{cases} \quad (1)$$

$$M_i^{(\omega)}(x) = \begin{cases} \frac{1}{m_i(x)} \prod_{j=0}^{\omega-2} m_j(x), & 0 \le i < \omega - 1, \\ \prod_{j=0}^{\omega-2} m_j(x), & i = \omega - 1. \end{cases} \quad (2)$$

Recursive Feature in $B^\top$

When computing matrix-vector multiplication with $B^\top$, there is a recursive relationship:

$$B_{i_2}^{(\omega)^\top} \cdot d_j = B_{i_1}^{(\omega-2)^\top} \cdot \{d_j \circledast c[m_\Delta(x)]\}, \quad (3)$$

d_j denotes the j-th column of the input tile d, $\circledast$ denotes cross-correlation, $m_\Delta(x)$ denotes additional polynomial(s) from $F(\omega-2)$ to $F(\omega)$, and $c[\,]$ means the coefficient vector of polynomial.

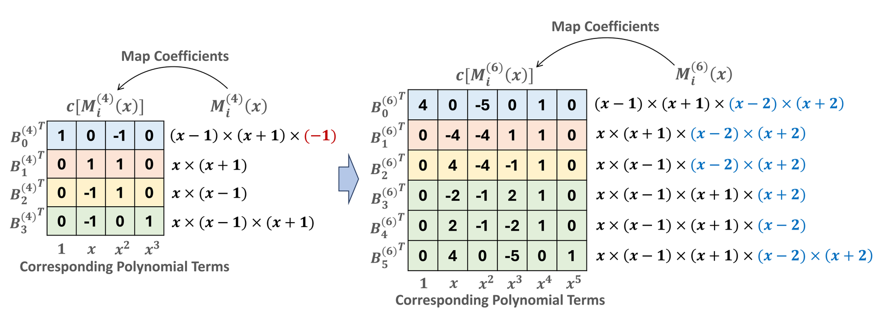

Figure 3. Input Transformation Matrix $B^\top$ for $F(\omega = 4)$ and $F(\omega = 6)$. The blue terms in $M_i^{(\omega)}(x)$ are additional polynomials $m_\Delta(x)$.

Recursive Winograd Input Transformation (`RIT`)

In `RIT`, for $F(\omega)$ denoting the set of $F(k, n)$ with a same ω, compute $D_j = B^{(\omega)^\top} d_j$ recursively:

1. perform correlation of d_j with the coefficient vectors of $m_\Delta(x)$ to get vectors h_1, h_2 and h_3;
2. multiply $B^{(\omega-2)^\top}$ with h_3;
3. continue to perform correlation on h_1 and h_2;
4. concatenate the results to get D_j.

The correlation computation can be transformed into add/sub-operations and shift operations. The final result can be obtained similarly through $U = (B^\top D)^\top$.

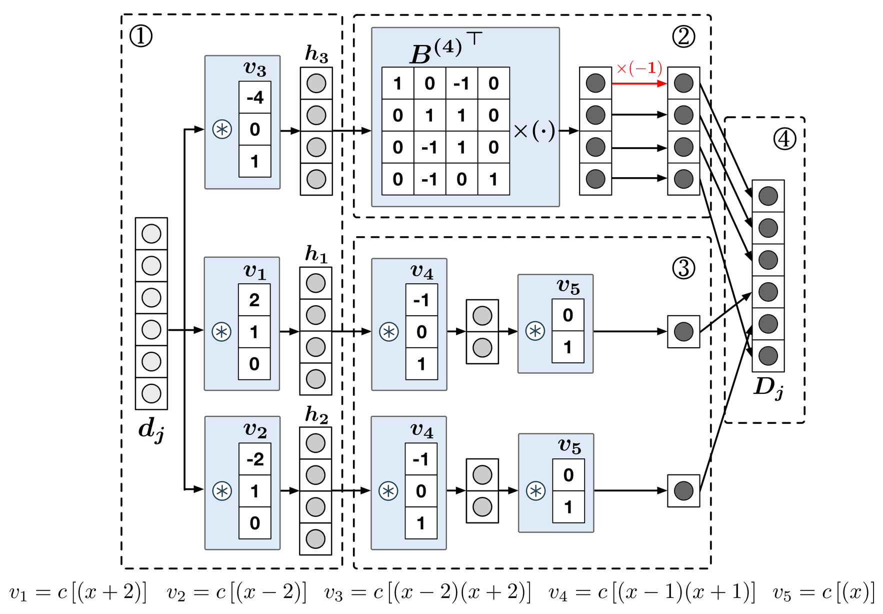

Figure 4. Recursive Input Transformation for $F(\omega = 6)$. Only $D_j = B^{(6)^\top} d_j$ is shown.

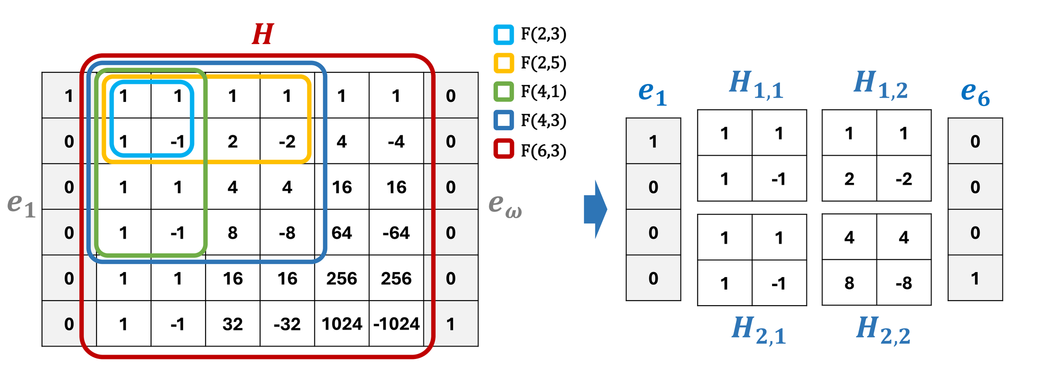

Figure 5. Sharing mechanism and block partition of $A^\top$ (using $F(6,3)$ and $F(4,3)$ as examples).

Sharing Mechanism and Value Distribution Pattern of $A^\top$

When k or n increases, the center part of $A^\top$ expands accordingly in units of 2×2 cells. For each cell, the value distribution pattern can be summarized as:

$$H_{i,j} = 2^r \begin{bmatrix} 1 & 1 \\ 2^t & -2^t \end{bmatrix}, \quad (4)$$

where $r = 2(i-1)(j-1)$ and $t = (j-1)$.

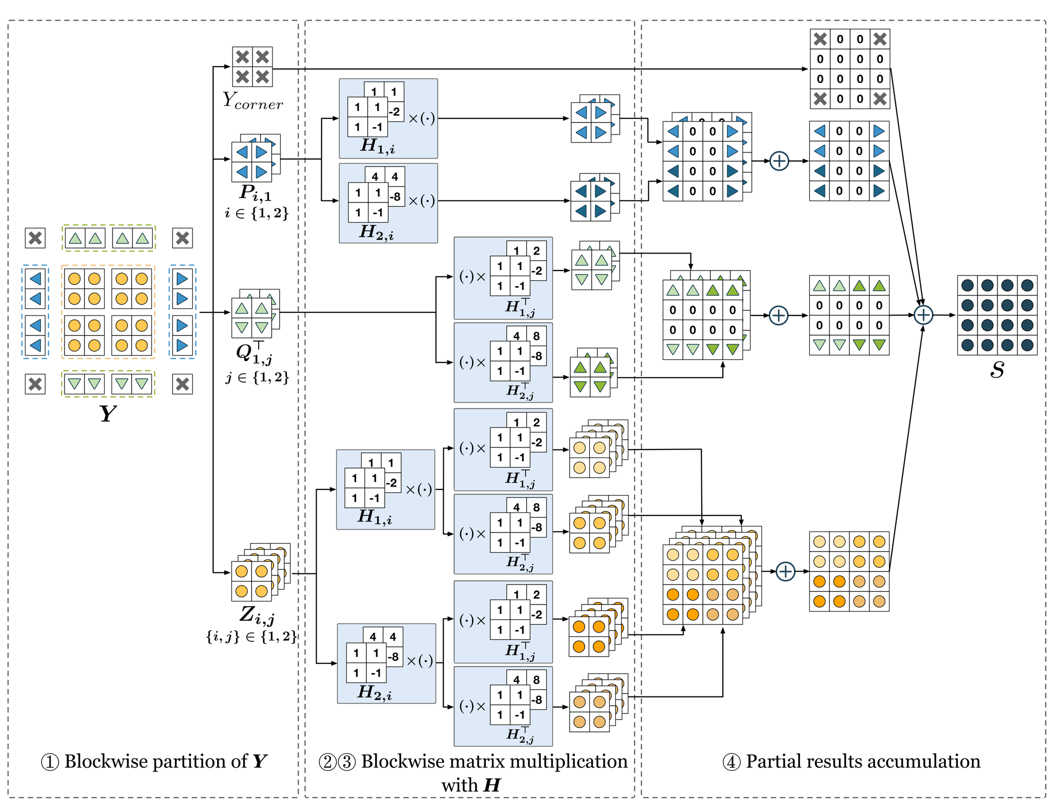

Figure 6. Blockwise Output Transformation for $F(4, 3)$.

Blockwise Winograd Output Transformation (`BOT`)

`BOT` divides $A^\top$ and Y into 2×2 sub-matrices and conduct:

$$\begin{aligned} & A^\top Y A \\ &= \begin{bmatrix} e_1 & H & e_\omega \end{bmatrix} \times \begin{bmatrix} Y(1,1) & q_1^\top & Y(1,\omega) \\ p_1 & Z & p_\omega \\ Y(\omega,1) & q_\omega^\top & Y(\omega,\omega) \end{bmatrix} \times \begin{bmatrix} e_1^\top \\ H^\top \\ e_\omega^\top \end{bmatrix}^\top \\ &= \begin{bmatrix} Y(1,1) & 0 & Y(1,\omega) \\ 0 & 0 & 0 \\ Y(\omega,1) & 0 & Y(\omega,\omega) \end{bmatrix} + H \begin{bmatrix} p_1 & 0 & p_\omega \end{bmatrix} + \begin{bmatrix} q_1^\top \\ 0 \\ q_\omega^\top \end{bmatrix} H^\top \\ &\quad + HZH^\top, \end{aligned} \quad (5)$$

The multiplication between $H_{i,j}$ and the sub-matrices of Y is then turned into a set of add/sub-operations and shift operations.

WinoGen IP Architecture

`SDW` algorithm provides a unified approach to transform arbitrary forms of Winograd input transformation and output transformation into addition, subtraction, and shift operations. Based on `SDW` method, following hardware architecture is proposed as a template to generate IPs:

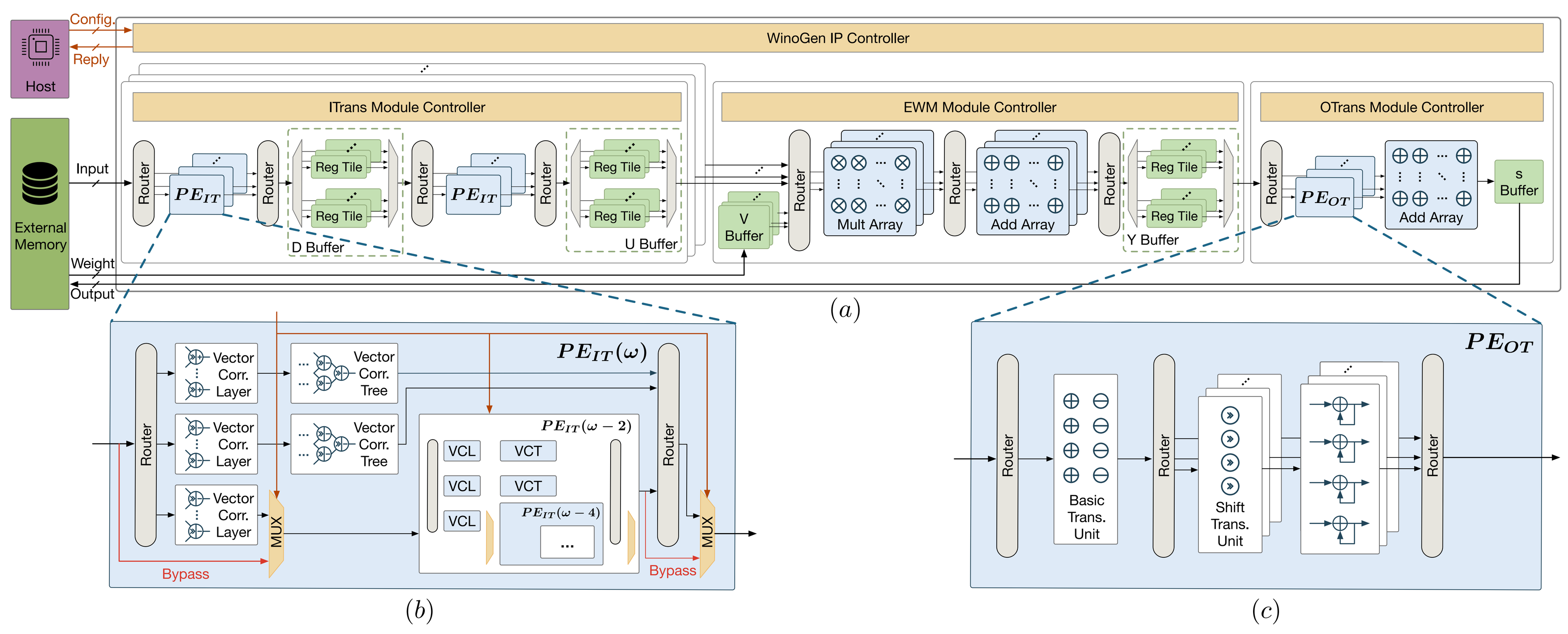

Figure 7. Architecture of the Winograd acceleration IP generated by `WinoGen`: (a) the overall architecture of `WinoGen` IP; (b) the detailed architecture of PE_{IT}; (c) the detailed architecture of PE_{OT}.

Input Transformation Module

- Computes Winograd input transformation using `RIT` method.
- Takes input tile d column by column.
- PN_{IT} PEs in each two groups for parallelism.
- ITrans Module constructed under $F(\omega)$ supports any $F(\omega')$ as long as $\omega' \le \omega$.

Output Transformation Module

- Computes Winograd output transformation using `BOT` method.
- Takes 2×2 sub-matrices of Y as input.
- PN_{OT} PEs for parallelism.
- OTrans Module constructed under $F(k, n)$ supports any $F(k', n')$ as long as $k' \le k$.

Element-wise Multiplication Module

- PN_C Multiplier Arrays deployed in parallel for PN_C input channels.
- PN_{EWM} multipliers in each Multiplier Array.
- EWM Module constructed under $F(\omega)$ supports any $F(\omega')$ as long as $\omega' \le \omega$.

WinoGen Optimizer

- Models the utilization of DSP and LUT as resource constraints.
- Models the throughput as optimization target.
- Sets IP hardware parameters PN_{IT}, PN_{EWM}, PN_{OT} and PN_C

Experiments

IP-level Evaluation and Operator-level Comparison

- **Static Configurable**: `WinoGen` can generate IP compatible with arbitrary forms of Winograd convolution for acceleration.
- **Dynamic Configurable**: generated IP supports multiple convolution kernel sizes and input/output feature tile sizes under a unified architecture.
- In cases where hardware resources are abundant, `WinoGen` can create IPs with ultra-high computational performance (over **1000** GOPS).
- For resource-limited scenarios, `WinoGen` can utilize its optimizer to generate optimized IPs.

Table 1. Optimized WinoGen IP under different resource constraints (* is used to test the throughput of the IP)

IP Type	Config. Type	IP Config.				Resource Util. (Constraint)			Thro. /GOPS				Supported Winograd Type
		PN_{IT}	PN_{EWM}	PN_{OT}	PN_C	DSP	LUT	FF	1×1 Kernel	3×3 Kernel	5×5 Kernel	7×7 Kernel	
F(4,1)	PN_{min}	1	1	1	1	4	1689	2191	1.308	5.559	-	-	F(2,3)*, F(4,1)*
	Constrained	4	4	4	1	16(20)	2267(2400)	2901	5.232	22.236	-	-	
	PN_{max}	4	4	4	4	64	4858	7579	36.624	92.868	-	-	
F(4,3)	PN_{min}	1	1	1	1	6	2441	3587	1.308	9.883	7.121	-	F(2,5)*, F(4,3)*; F(2,3), F(4,1)*
	Constrained	2	2	3	4	48(64)	8267(8400)	11678	12.208	123.824	86.764	-	
	PN_{max}	6	6	9	4	144	17472	18476	36.624	371.472	260.292	-	
F(6,3)	PN_{min}	1	1	1	1	8	3757	5703	1.308	12.508	16.023	7.930	F(2,7)*, F(4,5)*, F(6,3)*; F(2,5), F(4,3), F(6,1)*; F(2,3), F(4,1)
	Constrained	8	8	16	2	128(128)	31899(40000)	25307	35.316	412.02	517.968	255.06	
	PN_{max}	8	8	16	4	256	41093	40238	82.404	835.812	1041.168	511.428	

System-level Evaluation and Network-level Comparison

- DSP efficiency reaches $\mathbf{3.80}$ GOPS/DSP and energy efficiency reaches $\mathbf{652.77}$ GOPS/W.
- When computing VGG16, our design achieves DSP efficiency up to $\mathbf{3.26}$ GOPS/DSP and energy efficiency up to $\mathbf{559.83}$ GOPS/W, thus showing $\mathbf{2.45}\times$ and $\mathbf{3.10}\times$ improvements compared with state-of-the-arts.

Table 2. WinoGen evaluation and comparison with state-of-the-art designs.

	[4]		[2]	[3]	Vitis-AI		[1]		WinoGen IP-F(4,3)×12			WinoGen IP-F(6,3)×5		
Platform	KU115		VCU118	VU9P	ZCU102		ZCU102		ZCU104			ZCU104		
CNN Model	VGG16	AlexNet	VGG16	VGG16	VGG16	YOLOv2	VGG16	YOLOv2	VGG16	AlexNet	YOLOv2	VGG16	AlexNet	YOLOv2
Freq. (MHz)	235	220	200	167	281		214		317			317		
Supported Precision	8 bit / 16 bit	8 bit / 16 bit	16 bit	12 bit	8 bit		8-16 bit		8-16 bit			8-16 bit		
DSP (% of total)	4318 (78%)	4318 (88%)	3224 (47%)	5163 (75%)	1926 (76%)		2345 (93%)		1728 (100%)			1280 (74.1%)		
LUT (% of total)	257862 (39%)	257862 (40%)	-	-	-		-		209664 (91%)			205465 (89.2%)		
Power (W)	22.3	22.9	-	45.9	-		-		8.25			7.45		
Thro. (GOPS)	4022	3265	3772	3775.7	1225.2	1008	3120.3	1717.7	4453.2	4190.8	2408.6	4170.7	4863.1	2640.8
DSP Eff. (GOPS/DSP)	0.991	0.764	1.17	0.65	0.636	0.523	**1.33**	0.73	2.58	2.43	1.39	**3.26**	**3.80**	2.06
Energy Eff. (GOPS/W)	**180.4**	98.3	-	73.5	-	-	-	-	539.78	507.98	291.95	**559.83**	**652.77**	354.47

References

[1] Xinheng Liu, Yao Chen, Cong Hao, Ashutosh Dhar, and Deming Chen. WinoCNN: Kernel sharing winograd systolic array for efficient convolutional neural network acceleration on FPGAs. In *ASAP*, pages 258–265. IEEE, 2021.

[2] Junzhong Shen, You Huang, Mei Wen, and Chunyuan Zhang. Toward an efficient deep pipelined template-based architecture for accelerating the entire 2-D and 3-D CNNs on FPGA. *TCAD*, 39(7):1442–1455, 2019.

[3] Hanchen Ye, Xiaofan Zhang, Zhize Huang, Gengsheng Chen, and Deming Chen. HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation. In *DAC*. IEEE, 2020.

[4] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs. In *ICCAD*. IEEE, 2018.