An Efficient Reconfigurable Framework for General Purpose CNN-RNN Models on FPGAs

Shulin Zeng[1,2], Kaiyuan Guo[1], Shaoxia Fang[2], Junlong Kang[2],
Dongliang Xie[2], Yi Shan[2], Yu Wang[1,2], Huazhong Yang[1]
[1]Department of Electronic Engineering, Tsinghua University,
and Beijing National Research Center for Information Science and Technology (BNRist)
[2]Deephi Technology Co., Ltd
cengsl14@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn

***Abstract*—Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) have made great progress in machine learning community. Combining CNN and RNN can accomplish more general and complex tasks. Many specially designed hardware accelerators on FPGA or ASIC have been proposed for CNN or RNN, yet few of them focus on CNN-RNN-based models for general purpose applications. In this paper, we propose a complete design framework for deploying general-purpose CNN-RNN-based models on FPGAs. We use Deephi Aristotle and Descartes IPs to build an efficient and reconfigurable hardware system with the support of Deephi's toolchains and Xilinx SDSoC environment. We also design a CNN-RNN-based co-optimization method which can find the IP configuration to achieve the maximum throughput under the given FPGA resources and neural network models. Our implementation on the Xilinx ZU5EG FPGA achieves the throughput of 690.76GOPS and the energy efficiency of 86.34GOPS/W on LRCN network.**

***Index Terms*—framework; FPGA; CNN; RNN; optimization**

I. Introduction

In recent years, machine learning has become a rising star in both academic and industrial community, leading a great revolution in the AI era. The scope of machine learning application has gradually expanded from image recognition[1] to tasks such as speech recognition[2] and image segmentation[3], making machine learning a promising candidate for more and more applications.

Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are both state-of-the-art machine learning models. CNN shows a powerful ability in image classification[4] and is widely used in computer vision community. RNN is famous for its ability to process time-vary sequence and makes a significant contribution to speech recognition[5]. Combining CNN and RNN can accomplish more complicated and general tasks such as image caption[6] and video detection[7]. These applications are very challenging, requiring the abilities to process information both in space and time dimension. In these models, CNN is mainly applied to feature extraction and RNN is used for sequence detection and words generation.

This work was supported by National Key R&D Program of China 2018YFB0105005, National Natural Science Foundation of China (No. 61622403, 61621091), Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608), and Beijing National Research Center for Information Science and Technology (BNRist).

Many specially designed deep-learning accelerators on FPGA or ASIC have become the primary solution for embedded devices. However, there are only a few of studies focusing on both CNN and RNN due to the large gap of computation pattern and resources requirement between them. Shin et al.[8] propose a combined CNN-RNN heterogeneous processor named DNPU on ASIC, but there is a lack of software scheduling for CNN-RNN-based model. Yin et al.[9] design a hybrid processor with high reconfigurable ability to support CNN-RNN-based neural network on ASIC, while our work is based on FPGA and takes advantage of different IPs and toolchains. Guan et al.[10] propose an FPGA-based automated mapping framework named FPDNN. It maps different neural networks to different hardware designs using RTL-HLS templates, yet there isn't any evaluation on CNN-RNN-based models. Zhang et al.[11] present a HLS-based framework for both CNN and RNN with a new design space exploration engine called REALM. However, there could be 10x to 100x better performance using RTL IPs instead of HLS IPs. This paper proposes a complete design framework for deploying general-purpose CNN-RNN-based models on FPGAs. We use Deephi Aristotle and Descartes RTL IPs to build an efficient and reconfigurable hardware system with the support of Deephi's toolchains and Xilinx SDSoC environment. We also design a CNN-RNN-based co-optimization algorithm which can find the IP configuration to achieve the maximum throughput under the given FPGA resources and neural network models.

II. Design Framework

We combine the design flows of Deephi Aristotle and Descartes IPs with the hardware optimization method to form a complete design flow for mapping the CNN-RNN-based network into FPGAs, as shown in Figure 1(a). The overall hardware architecture is composed of two dedicated accelerator IPs on the programmable logic and the ARM CPU on Zynq processing system as shown in Figure 1(b). Deephi Aristotle IP is a high-efficiency instruction-based CNN processor based on Angel-eye[12]. Deephi Descartes IP is

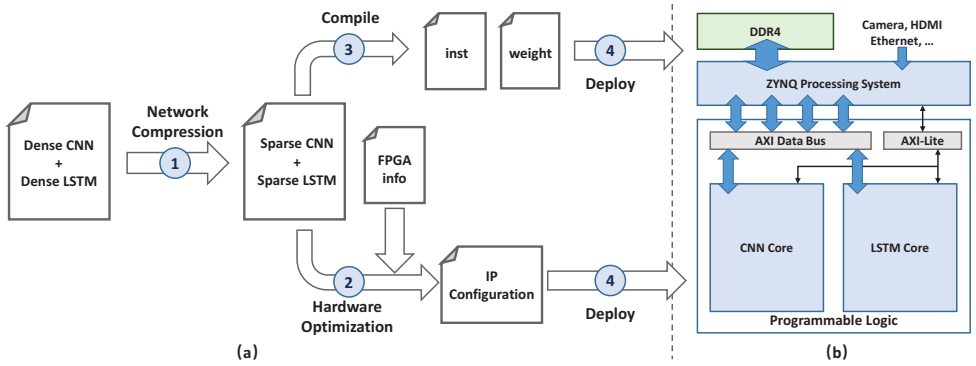

Fig. 1: Overview of the framework. (a)design flow (b)hardware architecture

designed for sparse LSTM acceleration based on ESE[13]. Each of these two IPs has its own design flow including three parts: deep compression, compiling, and software runtime deployment. The complete design flow of our framework contains four steps:

Deep compression. Deephi's compression tool is developed based on deep compression proposed by Han[14]. At this stage, pruning is first applied to both CNN and RNN network to prune away the unnecessary weights. After pruning, we use the dynamic data quantization strategy on both weights and activations to find the best decimal location of each layer and further quantize them to 12-bit integer for LSTM and 8-bit integer for CNN.

Hardware optimization. During this stage, we use the CNN-RNN-based co-optimization method that will be detailed in section III to find the IP configuration with the maximum throughput for both CNN and RNN IP according to the NN model and FPGA resources.

Compile. As for the CNN, the compiler will optimize the calculation time and memory I/O for each layer and map the network descriptor file to hardware instructions. When it comes to LSTM, the compiler is responsible for workload balanced and reorganization of the sparse matrix data.

Deployment. We use Xilinx SDSoC environment[15] to provide unified software-hardware interfaces by packing our accelerator IPs into C-callable IP library. This environment will not hurt the performance of hardware design, while it can provide an efficient and flexible development experience for users.

III. Hardware Optimization

In this section, we first give an overview of general purpose CNN-RNN-based models. Then we propose a CNN-RNN-based co-optimization method for mapping CNN-RNN-based models into FPGAs.

A. Model Topology

The topological structure of the CNN-RNN-based models is usually that CNN is responsible for the feature extraction at the front end. The LSTM cell array implements the processing of multi-frame data at the back end and then generates the text sequence output. The typical model topologies of image caption and video detection are shown in the Figure 2. In the application of image caption, CNN is only called once for each picture, and the extracted image features are repeatedly processed by one LSTM unit until the final sentence sequence is generated. As for the video detection, the runtime dependency of LSTM array extends from one dimension of each image to two dimensions under multiple frames.

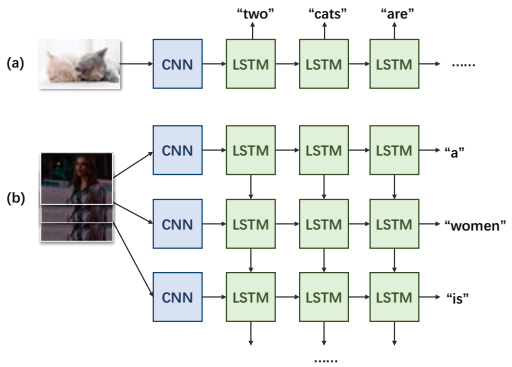

Fig. 2: Typical Model topology of (a)image caption, and (b)video detection.

We first analyze the case of image caption. There are two working modes: single mode and batch mode. In the single mode, one image is processed at a time. In this way, the control flow is quite direct and straightforward as shown in Figure 3(a). One CNN IP and one LSTM IP are all we need to support the flow. In the batch mode, multiple frames can be processed in a pipeline manner as shown in Figure 3(b). Each pipeline consists of one CNN IP and several LSTM IPs. Our optimization goal is to maximize the throughput of a single pipeline where there are one CNN IP and multiple LSTM IPs.

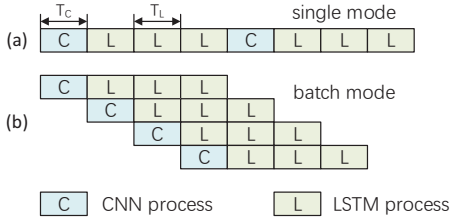

Fig. 3: (a)Single mode, one image at a time. (b)Batch mode, multiple frames in a pipeline manner.

To simplify the analysis, we assume that an optimal average sentence length N can be obtained through data analysis. That is, during the process of each image, one LSTM IP is to be called N times until it can generate the whole sentence. As shown in Figure 4(a), an ideal case is where one CNN IP and one LSTM IP can be pipelined without stalling. Here P and P_{target} means the current and target parallelism of LSTM IP, respectively. And the parallelism means the number of the available LSTM IPs. If the LSTM IP corresponding to the first image is still working on generating the sentence of length N when the CNN IP finishes processing the $(P+1)$-th picture, there will be idle time in the pipeline as shown in Figure 4(b). To prevent the pipeline from stalling, the parallelism of the LSTM IP must be increased to ensure that each frame can always have an available LSTM IP to continue the next calculation after CNN feature extraction. To meet this requirement, the Equation 1 should be satisfied.

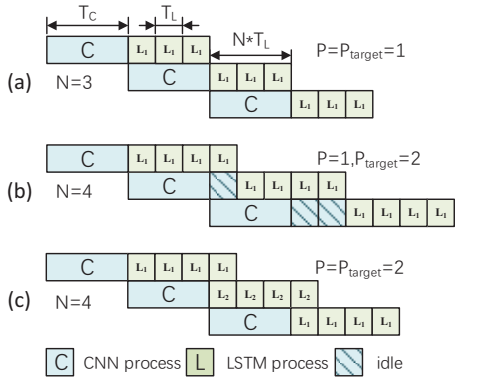

Fig. 4: Different cases of pipeline:(a)Ideal situation. (b)Stalling with idle time. (c)Optimized pipeline without stalling.

$$P \geq P_{target} = \lceil \frac{N \times T_L}{T_C} \rceil \tag{1}$$

Here the T_C and T_L denote the ideal computation time required by the CNN IP and LSTM IP, respectively. Then we can further get the P_{target} according to Equation 1. T_C and T_L are both related to the NN model and IP configurations. They can be each calculated as shown in Equation 2 and Equation 3.

$$T_C = C_m / P_m = C_m / roofline(C_m, S_m, P_c, M_c) \tag{2}$$

$$T_L = \frac{W_{total} \times compression\_ratio \times 2}{PE\_num \times 2 \times freq\_PE} \tag{3}$$

Here the C_m, P_m, and S_m denote the total calculation amount, the actual performance on a specific hardware platform and the memory I/O amount of the NN model, respectively. The P_c and M_c each represent the maximum attainable performance and the memory bandwidth of the hardware platform. We use the roofline model detailed in [16] to compute the performance. As for the T_L, the W_{total} and the PE_num denote the total size of the weight matrix and the computation capabilities of each channel (measured in Ops/cycle), respectively.

B. Pipeline Optimization

As discussed in section II, we use Deephi Aristotle IP for CNN and Deephi Descartes IP for LSTM. CNN IP has strong scalability with multiple IP configurations corresponding to different resources footprint and performance, while achieving the same accurary for the same NN models. LSTM IP supports the configurations of various parallelism by increasing resources usage. To find the IP configuration with the maximum throughput under given hardware resources and NN models, we set up an optimization model for in-depth analysis as shown in Equation 4.

$$\begin{aligned} &P^*, CNN\_id^* = \underset{P, CNN\_id}{\arg\max}\ Tht_{C-L}(N, n, T_C, T_L) \\ &s.t. \qquad r_C[CNN\_id] + P \cdot r_L \leq r_{FPGA} \\ &T_C = t_C[CNN\_id], \quad CNN\_id = 0, 1, \cdots, n_C - 1 \end{aligned} \tag{4}$$

$$Tht_{C-L} = \begin{cases} \frac{n}{n \cdot T_C + N \cdot T_L}, & P \geq P_{target} \\ \frac{n \cdot P}{P \cdot T_C + (n+P-1) \cdot N \cdot T_L}, & P < P_{target} \end{cases} \tag{5}$$

It takes the sentence length N, the number of image n, the ideal computation time T_C of CNN, and T_L of LSTM as input. Under the restriction that the resources footprint of the two IPs, r_C and r_L, do not exceed the FPGA resources r_{FPGA}, it will find the corresponding parallelism P of the LSTM IP and the CNN IP configuration CNN_id with the maximum throughput Tht_{C-L} of the CNN-RNN-based model, which can be calculated using Equation 5. Here the n_C denotes the number of CNN IP configuration, which is set to the best performance one when CNN_id is 0, and the most cost-efficient one when CNN_id is $n_C - 1$.

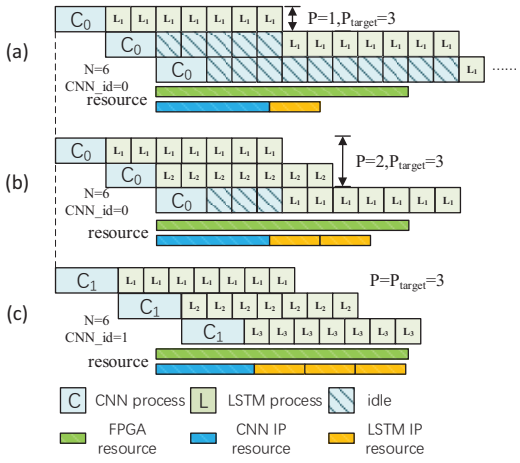

Fig. 5: Different stages of optimization:(a)Initialization. (b)Middle stage. (c)Final optimized pipeline.

Our proposed optimization method is detailed in Algorithm 1. At first, initializes P to 1 and CNN_id to 0 as shown in Figure 5(a). Then use as many resources as possible to increase the P so that P_{target} can be satisfied based on the Algorithm 1 from line 4 to line 11. If not, more resources need to be spared by reducing the performance of the CNN

IP, as the case in Figure 5(b). Next, compare the throughput of different CNN IP configurations to make sure the optimal throughput is found, as shown in line 12 to line 23 of the Algorithm 1. Finally, return the throughput and corresponding IP configurations. A typical result is shown in Figure 5(c).

Algorithm 1 Pipeline optimization method for CNN-RNN-based model

1: set $n \rightarrow \infty$
2: **function** OPTI-CNN-LSTM(N, n_C, $t_C[]$, $r_C[]$, T_L, R_L, R_{FPGA})
3: $P = 1$, $T_C = t_C[0]$, $R_C = r_C[0]$
4: **if** $T_C < N \times T_L$ **then**
5: $Tht_{C-L} = 1/T_C, CNN_{id} = 0$
6: **else**
7: $P_{target} = \lceil \frac{N \times T_L}{T_C} \rceil$
8: $P_{max} \leftarrow get\_max\_P(R_C, R_L, R_{FPGA})$
9: **if** $P_{max} \geq P_{target}$ **then**
10: $P = P_{target}, Tht_{C-L} = 1/T_C$, exit
11: **else**
12: **for** $i = 0 \rightarrow n_C - 1$ **do**
13: $P = P_{max}, Tht = \frac{P}{N \times T_L}$
14: $T_C = t_C[i]$, $R_C = r_C[i]$
15: $P_{max} \leftarrow get\_max\_P(R_C, R_L, R_{FPGA})$
16: $P_{target} = \lceil \frac{N \times T_L}{T_C} \rceil$
17: **if** $P_{max} < P_{target}$ **then**
18: continue
19: **else if** $Tht_{C-L} < 1/T_C$ **then**
20: $P = P_{target}, CNN\_id = i$
21: $Tht_{C-L} = 1/T_C$, exit
22: **else**
23: $CNN\_id = i - 1$, exit
24: **return** P, CNN_id, Tht_{C-L}

When it comes to video detection in the batch mode, the control flow is the same as image caption of $T_c \geq T_L$. As for the other case, it is different in that the first LSTM process of each image is followed by the same LSTM process of the last image, but not the CNN process of the current image. However, we can still apply the same optimization method to it and find the IP configurations with the maximum throughput.

IV. EXPERIMENT

A. Experiment Setup

We use four different CNN IP configurations for experiment. The DSP number of each is 256, 512, 768, 1024 and their equivalent computation parallelism (measured in Ops/cycle) is 1024, 2048, 3072, 4096, respectively. And the width of their external memory port is 256-bit. Then we use Equation 2 to compute the CNN IP ideal computation latency of seven different CNN models: AlexNet, SqueezeNet, VGG16, GoogLeNet, ResNet50, InceptionV3, DenseNet121. As for the LSTM IP, PE_num is 8. The dimension of the LSTM model is 1024×1024 with 60% sparsity. Both CNN IP and LSTM IP run at 200MHz. We evaluate our algorithm based on the resources of Xilinx ZU5EG FPGA, and the resource utilization of each IP is showed in Table I. We run two tests of N equal to 10 and 20 compared with the single mode, where P is 1 and CNN_id is 0.

TABLE I: Resources of Different IPs and FPGA

	CNN_id	LUT	FF	BRAM	DSP
CNN-B1024	3	27689	36759	69	256
CNN-B2048	2	46259	78985	133	512
CNN-B3072	1	65210	113989	201	768
CNN-B4096	0	82169	148686	250	1024
ESE-8PE	NA	2375	3589	8	23
FPGA-ZU5EG	NA	117120	234240	884	1248

B. Experiment Results

TABLE II: Experiment Results (N=10,LSTM-1024×1024)

CNN Model	CNN_id	P	Tht_{batch} (frame/s)	$Perf_{batch}$ (GOP/s)	Tht_{single} (frame/s)	Speedup
AlexNet	0	2	108.46	122.23	92.70	1.17
SqueezeNet1.1	1	19	1162.79	883.72	410.88	2.83
VGG16	0	1	43.69	716.47	40.83	1.07
GoogLeNet	0	6	332.23	797.34	218.57	1.52
ResNet34	0	4	204.92	901.64	155.24	1.32
Inception V3	0	3	136.61	874.32	112.90	1.21
DenseNet121	0	3	163.13	554.65	129.47	1.26

TABLE III: Experiment Results (N=20,LSTM-1024×1024)

CNN Model	CNN_id	P	Tht_{batch} (frame/s)	$Perf_{batch}$ (GOP/s)	Tht_{single} (frame/s)	Speedup
AlexNet	0	4	108.46	165.62	92.70	1.17
SqueezeNet1.1	3	27	568.18	659.09	411.72	1.38
VGG16	0	2	43.69	733.95	40.83	1.07
GoogLeNet	1	10	306.75	858.90	207.26	1.48
ResNet34	0	7	204.92	983.61	155.24	1.32
Inception V3	0	5	136.61	928.96	112.90	1.21
DenseNet121	0	6	163.13	619.90	129.47	1.26

Table II and Table III show that the throughput and IP configurations vary with the NN model. We can calculate the maximum accessible parallelism of LSTM IP by 9 at the beginning, which means most of the network can be pipelined without stalling when their P_{target} is less than 9. In this case, the bottleneck of the pipeline lies in the latency of CNN IP. As for those whose P_{target} is larger than 9, our optimization algorithm can explore the IP configurations to achieve better throughput. Take SqueezeNet as an example. It chooses B3072 CNN IP when N is 10 and B1024 when N is 20. That is because the throughput of the pipeline is limited by the calculation of LSTM. Our optimization method will choose to reduce the performance of the CNN IP in exchange for the enhancement of LSTM IP. A similar case is GoogLeNet. It chooses B3072 CNN IP when N is 20, showing the ability of our algorithm to find the IP configurations with the best throughput.

Compared with the single mode, the optimized batch mode can gain a better throughput up to 2.83x on SqueezeNet and an average of 1.3x for seven different CNN models with LSTM, which is salient enough to boost the performance for a specific application.

C. Comparison with Related Work

Here we compare our design with other related works evaluated on the LRCN network, which consists of AlexNet

and a two-layer LSTM network. As for the configuration of our work, the sparsity of CNN and LSTM network is 0.2. We choose B4096 configuration for CNN IP and the PE_num of LSTM is 32. The working frequency is 200MHz and the target FPGA is ZU5EG. The system performance and power consumption of the accelerator are based on the simulator. The results are shown in the following table IV.

TABLE IV: Comparison with Related Work

	This work	VLSI'17[9]	FPL'17[11]
Hardware	FPGA-ZU5EG	ASIC	FPGA-VC709
Frequency	200MHz	200MHz	100MHz
Model	LRCN	LRCN	LRCN
Throughput	690.76GOPS	381.2GOPS	36.25GOPS
Latency	0.009s	0.016s	0.040s
Power	8W	297.8mW	23.6W
Efficiency	86.34GOPS/W	1.28TOPS/W	1.54GOP/W

As for the performance results, our FPGA-based accelerator system can achieve the highest throughput, 1.8x and 19x better than the ASIC and the HLS-based FPGA design, respectively. This is mainly due to the network compression technology and the software-hardware co-optimization method, so that the CNN-RNN-based network can run more efficiently on FPGA with specially designed RTL IPs. The advantages of ASIC-based work[9] are mainly reflected in the extremely low power consumption, which is the essential difference caused by the hardware platform. However, the 8W power consumption of our design can meet the needs of most low-power application scenarios, and it is 56x more energy efficient than the HLS-based design [11], showing that Deephi RTL IPs for DNN are powerful and highly efficiency. In addition, one of the great advantages of our design is the full-stack development and deployment toolchains for FPGA. Compared with ASIC, it has more flexibility and expandability with the help of Xilinx SDSoC environment. It could greatly shorten the users development time and make them focus on the algorithm design.

V. Conclusion

This work proposes an efficient and reconfigurable framework for deploying general-purpose CNN-RNN-based models on FPGAs using Deephi's accelerator IPs and toolchains. We design a CNN-RNN-based co-optimization method which can find the IP configurations with the maximum throughput under given FPGA resources. And we show it can achieve an average of 1.3x and up to 2.83x speedup for seven different CNN models with LSTM.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," pp. 770–778, 2015.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *Computer Science*, no. 4, pp. 357–361, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, 2012, pp. 1097–1105.

[5] A. Graves, A. R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[6] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," pp. 3156–3164, 2014.

[7] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in *Computer Vision and Pattern Recognition*, 2015, p. 677.

[8] D. Shin, J. Lee, J. Lee, and H. J. Yoo, "14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks," in *Solid-State Circuits Conference*, 2017, pp. 240–241.

[9] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 tops/w reconfigurable hybrid-neural-network processor for deep learning applications," in *VLSI Circuits, 2017 Symposium on*, 2017, pp. C26–C27.

[10] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2017, pp. 152–159.

[11] X. Zhang, X. Liu, A. Ramachandran, C. Zhuge, S. Tang, P. Ouyang, Z. Cheng, K. Rupnow, and D. Chen, "High-performance video content recognition with long-term recurrent convolutional network for fpga," in *International Conference on Field Programmable Logic and Applications*, 2017, pp. 1–4.

[12] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[13] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, and Y. Wang, "Ese: Efficient speech recognition engine with sparse lstm on fpga," 2017.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[15] V. Kathail, J. Hwang, W. Sun, Y. Chobe, T. Shui, and J. Carrillo, "Sdsoc:a higher-level programming environment for zynq soc and ultrascale+ mpsoc," pp. 4–4, 2016.

[16] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Acm/sigda International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170.