Implementation of a CNN accelerator on an Embedded SoC Platform using SDSoC

Sang-Soo Park
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
po092000@hanyang.ac.kr

Kyeong-Bin Park
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
lay1523@naver.com

Ki-Seok Chung
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
kchung@hanyang.ac.kr

ABSTRACT
Today, Convolution Neural Networks (CNN) is adopted by various application areas such as computer vision, speech recognition, and natural language processing. Due to a massive amount of computing for CNN, CNN running on an embedded platform may not meet the performance requirement. In this paper, we propose a system-on-chip (SoC) CNN architecture synthesized by high level synthesis (HLS). HLS is an effective hardware (HW) synthesis method in terms of both development effort and performance. However, the implementation should be optimized carefully in order to achieve a satisfactory performance. Thus, we apply several optimization techniques to the proposed CNN architecture to satisfy the performance requirement. The proposed CNN architecture implemented on a Xilinx's Zynq platform has achieved 23% faster and 9.05 times better throughput per energy consumption than an implementation on an Intel i7 Core processor.

CCS Concepts
• **Hardware** → Hardware-software codesign**;** • **Computer systems organization → Embedded systems;**

Keywords
CNN; AI; HW/SW Co-Design; HLS; LeNet-5; FPGA; SDSoC;

1. INTRODUCTION
Convolution Neural Network (CNN) is a type of Artificial Neural Network (ANN) inspired by neural cells. CNN is a technology that recognizes a specific pattern by the way people perceive objects. CNN attracts attention due to its excellent classification capability in image recognition fields [1]. Currently, CNN is adopted by video, speech, natural language processing and it has achieved remarkable success in many application areas [2-4].

To achieve high classification capability, CNN commonly employs deep layers composed of many processing layers and therefore, the computation amount of CNN is very huge [5]. To speed up the CNN processing, many studies have employed graphics processing units (GPUs) [6]. Employing GPUs may be an effective method to achieve high performance. But the development library and environment for mobile GPUs in embedded systems is limited [7]. Also, GPUs may not be suitable for battery driven embedded systems such as smart phones due to its high-power consumption [8]. Therefore, for embedded systems, only CPUs are mainly employed for accelerating CNN [9-10].

Another alternative method may be the development of a purely HW architecture, which offers both high performance and good energy efficiency [11]. However, designing Application-specific Integrated Circuits (ASICs) at the register-transfer level (RTL) needs high design effort and time.

Recently, a field programmable gate array (FPGA) vendor, Xilinx announced a high-level synthesis (HLS) tool called SDSoC with C/C++ development environment to generate a HW-SW co-design on a heterogenous FPGA-CPU platform. HLS enables developers to design such an accelerator with a low cost and a short time-to-market. The key feature of SDSoC is that it transforms C/C++ design descriptions into a HW accelerator with relatively low effort and time, and it is possible to achieve high performance with reasonable energy consumption. In this paper, we propose a system-on-chip (SoC) CNN accelerator that is synthesized by SDSoC. Hardware compilation should be carried out with appropriate optimization options in order to achieve satisfactory performance. The HW logic generated by the default compilation option of SDSoC doesn't meet the target performance because operational dependencies of the CNN algorithm are rather complicated. Thus, we apply several optimization techniques to the proposed CNN architecture to satisfy the target performance. Our implementation achieves 9.05 times better throughput per energy consumption and 23% faster than an x86-based quad-core desktop processor.

The rest of this paper is organized as follows: Section 2 introduces CNN. Section 3 presents the proposed CNN accelerator. Section 4 shows our experiment results. Section 5 concludes this paper.

2. CONVOLUTION NEURAL NETWORK
CNN is inspired by neuroscience. Through development over 20 years, CNN has received attention in areas such as natural language processing and computer vision. Today, some object recognition systems based on CNN can recognize objects with super-human accuracy [12]. CNN recognizes an object by feature extraction and classification. Feature extraction with convolutions and sub-samplings is a step to find invariances of an input image such as lines and edges. Classification with full-connections selects an object of the most likely class based on the extracted features.

ICDSP 2018, February 25–27, 2018, Tokyo, Japan

ACM ISBN 978-1-4503-6402-7/18/02$15.00
https://doi.org/10.1145/3193025.3193041

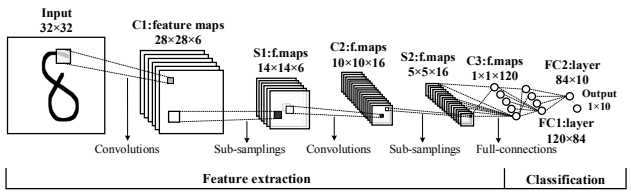

Figure 1. The architecture of LeNet-5

Using convolution, sub-sampling, and fully-connected layers, CNN can achieve a highly accurate classification performance.

Figure 1 shows the architecture of LeNet-5 [13]. It is configured with three convolution layers, two sub-sampling layers, and two fully-connected layers. LeNet-5 extracts output results called feature maps from one 32×32 input image through multiple convolution and sub-sampling layers. Finally, the extracted 120 feature maps of the 1×1 resolution are passed through the fully-connected layers to get the final classification result, which is a digit from 0 to 9.

2.1 Convolution Layer

In the convolution layer, element-wise multiplications between an input feature map and a convolution kernel are carried out. Figure 2 shows the process that is composed of several steps in a convolution layer.

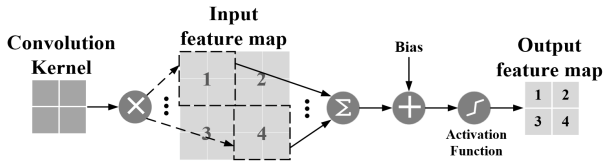

Figure 2. The process of convolution layer

First, the input feature map is convolved with a convolution kernel. Second, the sum of weighted results from the first stage and the bias is calculated. Finally, the sum result is filtered with an activation function such as sigmoid, tanh, and ReLu. If an N×N input feature map and an m×m convolution kernel are used in a convolution layer, the output feature map is determined as (N-m+1)×(N-m+1). The process in the overall convolution layer may be summarized as Equation (1).

$$O_t^{(x,y)} = f'\left(\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} W_t^{(i,j)} \cdot I^{(x+i,y+j)} + bias\right) \quad (1)$$

In Equation (1), *I* and *O* indicate the input and output feature map, respectively. A two-dimensional coordinate *(x, y)* indicates the position in the output feature map. *W* indicates a convolution kernel while *i*, *j* and *t* denote the width, the height and the type indices of the convolution kernel, respectively. The activation function is denoted by *f'*. For example, convolution layer 1 in LeNet-5 uses one input feature map with the resolution of 32×32, 6 types of convolution kernels with the resolution of 5×5 and biases. As the result, 6 output features.

2.2 Sub-sampling Layer

Following the execution of a convolution layer, the sub-sampling layer reduces the size of feature maps from the previous layer. This layer is frequently used in CNN with the purpose to gradually reduce the spatial size of the number of features and the computational complexity of the neural network. In addition, this layer is useful to avoid a problem called overfitting.

2.3 Fully-connected Layer

After the feature extraction with multiple convolution and sub-sampling layers, fully-connected layers follow. The term "fully-connected" means that all neurons in the previous layer are connected to all neurons in the next layer. This layer is used to classify an input image into various categories based on training datasets. For example, LeNet-5's last layer in Figure 1 has 10 possible outputs and each output corresponds a digit from 0 to 9.

3. IMPLEMENTATION OF CNN ACCELERATOR

3.1 Platform Overview

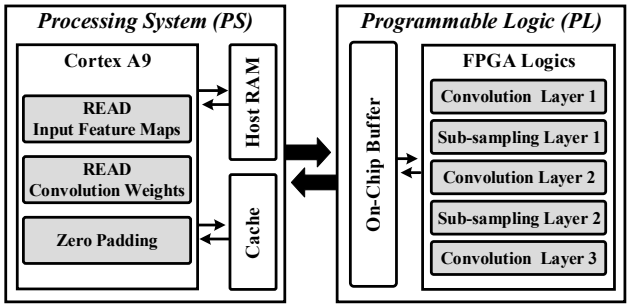

Figure 3. Proposed accelerator in FPGA-CPU Platform

In this paper, we implement a CNN accelerator on a Xilinx's Zynq platform using SDSoC [14]. Figure 3 shows the target platform. Zynq consists of Processing System (PS) and Programmable Logic (PL). PS is a dual-core ARM processor and PL is an FPGA logic module which consists of a memory block called BRAM, reconfigurable logic blocks such as look-up table (LUT), flip-flops (FFs), and digital signal processors (DSPs).

In our implementation, considering the characteristic of PS and PL, we assigned operation processes into two modules. In PS, the host ARM processor reads input feature maps, convolution kernels, and biases from the RAM in PS (Host RAM). Then, PS executes zero-padding of the input feature maps and sends data to on-chip buffers in PL through interconnection called Accelerator Coherency Port (ACP). After zero-padding, the feature extraction which consists of multiple convolution and sub-sampling operations is carried out iteratively by the logic mapped in PL. After both convolution and sub-sampling are completed, the result data is sent back to Host RAM in PS.

3.2 Proposed Implementation Method

In theory, it would be advantageous to calculate partial sums of several output feature maps in parallel. However, due to the limitations of memory resource and bandwidth, calculating several output feature maps in parallel is not adequate in the FPGA-based HW design. Therefore, calculating only one output feature map in parallel is conducted in our proposed implementation and the process of calculating one output feature map is conducted by computing several partial sums in parallel.

To parallelize the convolution layer, loop unrolling and loop pipelining techniques are applied by adding corresponding high-level synthesis pragmas as shown in Figure 4. With ***#HLS unroll*** pragma, SDSoC generates unrolled C codes. Loop unrolling is a key to achieve high performance through parallelization. It is used to make parallel instantiations of the loop body. It creates multiple HW logic blocks corresponding to the loop iteration counter. Therefore, the resource utilization rate in FPGA increases. Loop

```
// Load input feature map, convolution kernel, bias to buffer
for(b=0; b <B; b++) { // # of input image
  for(do =0; do <D; do ++) {    // output feature map
    for(ir=0, ic=0; ir <IR, ic<IL; ir ++, ic++) { //input feature map
    // Tiling based computation part
      for(kd=0; kd <KD; kd ++) { // kernel depth
        #pragma HLS pipeline
        for(kr=0; kr <KR; kr ++) { // kernel row
          #pragma HLS unroll
          for(kc=0; kc <KC; kc++) { // kernel col
            #pragma HLS unroll
          element[kc] = input[b][kd][ir+kr][ic+kc] * kernel[do][kd][kr][kc]
        }
        // Sum of column element in same row line
          sum_srl[kr] =    element[0] + …. element[KC-1];
      }
      // Sum of multiple row lines
        sum_mrl[kd] =    sum_srl[0] + …. sum_srl[KR-1];
    }
    // Sum of all row lines and Activation function
      result = sum_mrl[0] + …. sum_mrl[KD-1]
      output[b][do][ir][ic] = Activation(result + bias[do])
} } }
```

Figure 4. Pseudo-code of the proposed CNN accelerator

pipelining is also a powerful technique to improve throughput by executing loop iterations in an overlapping fashion. With ***#HLS pipeline*** pragma, each iteration is divided into small stages and a pipeline logic that meets the timing constraints is generated.

The processing code in the convolution layer is composed of deeply nested loops that need to be synthesized carefully. Inner loops have low iteration counts and the outer loops have high iteration counts. For the best utilization of these characteristics, it is desirable to apply the loop unrolling to the inner loops to maximize parallelism and apply the loop pipelining to the outer loops to maximize throughput. In SDSoC, when the pipelining is applied to deeply nested loops, the innermost loop is fully unrolled automatically. However, when the unrolled loop has several memory accesses or complex operations, the synthesized circuit may suffer from timing violations

Moreover, in order to maximize performance gain from these optimization methods, some loops are modified by eliminating some dependencies. In the convolution layer, there is some reduction case where multiple operations attempt to access the same variable and update the value, called Read-After-Write (RAW) dependency. The loops in the convolution layer are not suitable to parallelizing optimization methods because such RAW dependency forces the computation to be serialized.

To overcome such problems, in our implementation, we apply a loop tiling method so that the synthesized HW can carry out computations in parallel. The loop tiling is a method to partition a loop's iteration into smaller blocks so that memory access patterns, and therefore, data dependencies may be changed.

A loop tiling method turns out to be very effective for CNN processing acceleration. By intelligently grouping the processing of the convolution layer, it becomes possible that a HW logic block that can carry out the computation in parallel may be synthesized. We divide the computing process for the partial sum of an output feature map into several steps: kernel columns (*kc*), kernel rows (*kr*), and kernel depths (*kd*). Figure 5vshows the acceleration structure with the loop tiling method for the proposed CNN accelerator. The loop iterations of computing one output feature map are tiled into blocks with the tile size of (*KC*, *KR*, *KD*) and the loops labeled as "kernel row" and "kernel column" are selected to be unrolled and the loop "kernel depth" is selected to be loop-pipelined. Table 1 shows the count of loop iterations in each convolution layer.

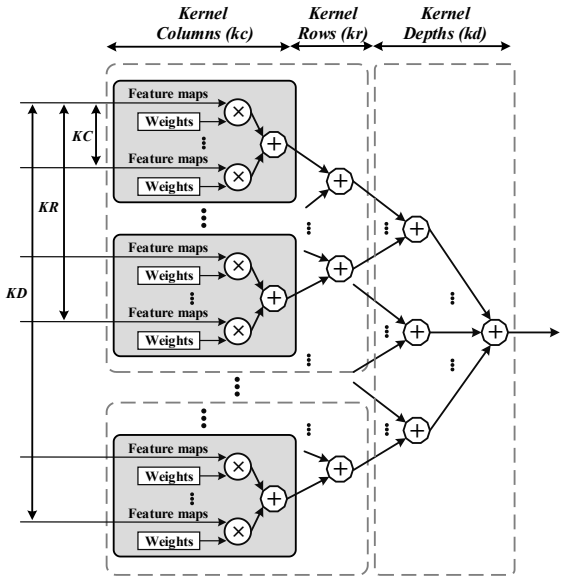

Figure 5. The accelerator structure with tiling technique

Table 1. The count of iteration in convolution layer

Convolution Layer	Kernel Depth (KD)	Kernel Rows (KR)	Kernel Columns (KC)
1	1	5	5
2	6	5	5
3	16	5	5

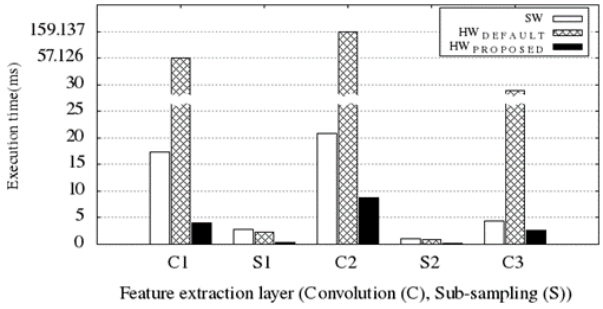

Figure 6. Execution time comparison of implementation

Figure 6 shows the comparison of execution time between the proposed CNN accelerator and the SW implementation in PS. The execution time of each implementation is the result of each layer in processing 10 batches, which is the number of input images. The proposed method based on tiling ($HW_{PROPOSED}$) shows better execution time performance. In the default HW implementation where dependency exists ($HW_{DEFAULT}$), it takes 2.41 times on average more than the SW implementation where everything is run only on PS. However, the proposed method shows 4.4 times on average less. Thus, it is verified that the proposed method is an effective technique to improve the execution time.

4. RESULT AND ANALYSIS

In this paper, we propose a HW implementation of the LeNet-5 architecture on an embedded SoC platform. We used the Xilinx

Zynq zc706 device which contains a dual ARM Cortex A9 core processor as PS and Kintex-7 FPGA as PL. The HW accelerator is synthesized with SDSoC v2016.3. We compare the proposed implementation with other conventional implementations in terms of performance and power efficiency. Three different versions of software implementations are compared. First, SW_{PS} represents a parallel software implementation of the CNN on PS. SW_{i7} represents a software implementation of Intel i7-6700 with 8 threads, and SW_{RP} is another software implementation on a Raspberry Pi3 platform with 4 threads. Raspberry Pi3 is one of the most popular embedded platforms [15]. HW_D represents a synthesized HW based on the default CNN architecture and HW_O is the proposed implementation with the application of the proposed loop optimization methods. All the software implementations are parallelized by OpenMP pragmas with the optimization option, OpenMP-O3. The operating clock speed of each implementation is summarized in Table 2.

To evaluate performance of the proposed method, the MNIST benchmark [16] was used as the test workload. MNIST is one of the most widely used benchmarks in pattern recognition. It consists of handwritten images for a single digit from 0 to 9 and it has 55,000 training sets and 10,000 test sets. The execution time and the energy dissipation for executing the feature extraction during classification of 10,000 images have been measured.

Table 2. The specification of implementations

Metrics	SW_{RP}	SW_{i7}	SW_{PS}	HW_D	HW_O
Core	Cortex A53	X86	Cortex A9	Cortex A9 / FPGA	
Clock (GHz)	1.3	3.5	1.0	1.0 (PS) / 0.17 (FPGA)	
# of Core	4	4	2	2 (PS)	

4.1 Performance and Energy Consumption

Table 3 shows comparison of the execution time between the proposed CNN accelerator and the other implementations. We also measured power consumption and energy dissipation of each implementation using a pluggable power meter.

Overall, HW_O takes less execution time than the other implementations. HW_O shows a speedup of up to 3.66 in the convolution and sub-sampling process compared to SW_{PS}. However, the execution time of HW_D is the longest. There are two main reasons why HW_D is the slowest. First, the dependency in HW_D forces the HW execution to be serialized, and thus, the pipeline is heavily stalled. Secondly, the operating frequency of FPGA (0.166 GHz) is much lower than PS (at least 0.6 GHz).

Also, HW_O shows the best energy consumption. Compared to SW_{i7}, HW_O is 7.29 times better with respect to energy consumption. Especially, compared to SW_{RP}, HW_O is 91% faster while dissipating a similar level of energy.

Table 3. Performance comparison (10,000 Images)

Metrics	SW_{RP}	SW_{i7}	SW_{PS}	HW_D	HW_O
Execution Time (s)	30.76	19.78	58.8	312.7	16.07
Power (W)	4.75	65	7.2	7.68	10.98
Energy (J)	146.11	1285.7	423.36	2401.5	176.45

4.2 HW Resource Utilization

Table 4 summarizes the FPGA utilization of the synthesized HW. When the proposed loop optimization techniques are applied, the HW size increases significantly because the HW logic is optimized for faster parallel execution. Even if it seems that the size of HW_O is much bigger than that of HW_D, in FPGA devices, all the resource is built-in. So, utilizing more resource to improve performance is commonly justified

Table 4. FPGA HW utilization

Resource	DSP	BRAM	FF	LUT
HW_D	25	2	10173	8731
HW_O	59	97	59198	39837

4.3 Overall Performance Comparison

Table 5 shows comparison of implementations in terms of throughput, which is the number of processed images that can be processed per second. The proposed method, HW_O achieves 622.28. Also, Throughput/Energy has been computed, and HW_O shows a much better than others. Compared to x86 based SW_{i7}, it is 9.05 times better throughput per energy consumption. Therefore, we conclude that the proposed implementation is very effective for both execution time and energy efficiency.

Table 5. Performance efficiency

Metrics	SW_{RP}	SW_{i7}	SW_{PS}	HW_D	HW_O
Performance (Image/s)	325.1	505.56	170.1	31.98	622.28
Performance per Energy	2.23	0.39	0.4	0.01	3.53

5. CONCLUSION

In this paper, an implementation for LeNet-5 on an embedded SoC platform was presented. The proposed implementation was optimized with various techniques. Especially, a technique called loop tiling was very effective to optimize the design. A high-level synthesis tool called SDSoC was used to synthesize the HW accelerator, and the synthesized design shows much better throughput per energy consumption than other conventional implementations on either desktop or embedded platforms.

6. ACKNOWLEDGMENTS

This work was supported by the Technology Innovation Program (10076583, Development of free-running speech recognition technologies for embedded robot system) funded By the Ministry of Trade, Industry & Energy(MOTIE, Korea).

7. REFERENCES

[1] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[2] Hsu, W. N., Zhang, Y., Lee, A., & Glass, J. R. (2016). Exploiting Depth and Highway Connections in Convolutional Recurrent Deep Neural Networks for Speech Recognition. In INTERSPEECH (pp. 395-399).

[3] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 1725-1732).

[4] Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. arXiv preprint arXiv:1702.01923.

[5] Cong, J., & Xiao, B. (2014, September). Minimizing computation in convolutional neural networks. In International conference on artificial neural networks (pp. 281-290). Springer, Cham.

[6] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., & Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759.

[7] Alzantot, M., Wang, Y., Ren, Z., & Srivastava, M. B. (2017, June). RSTensorFlow: GPU Enabled TensorFlow for Deep Learning on Commodity Android Devices. In Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications (pp. 7-12). ACM.

[8] Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678.

[9] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678). ACM.

[10] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[11] Chen, Y. H., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 52(1), 127-138.

[12] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[13] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[14] Kathail, V., Hwang, J., Sun, W., Chobe, Y., Shui, T., & Carrillo, J. (2016, February). SDSoC: A Higher-level Programming Environment for Zynq SoC and Ultrascale+ MPSoC. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 4-4). ACM.

[15] Raspberry pi 3, https://www.raspberrypi.org/

[16] LeCun, Y., Cortes, C., & Burges, C. J. (2010). MNIST handwritten digit database. AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2.