공학석사 학위논문

에너지 효율적인 YOLO 가속기를 위한 Filter Switching 최적화

Filter Switching Optimization for

An Energy-Efficient YOLO Accelerator

2022 년 2 월

서울대학교 대학원

전기 정보 공학부

임 경 종

에너지 효율적인 YOLO 가속기를 위한 Filter Switching 최적화

지도 교수 이 혁 재

이 논문을 공학석사 학위논문으로 제출함
2022 년 2 월

서울대학교 대학원
전기 정보 공학부
임 경 종

임경종의 공학석사 학위논문을 인준함
2022 년 2 월

위 원 장 김 태 환 (인)

부위원장 이 혁 재 (인)

위 원 심 재 웅 (인)

초 록

You-only-look-once (YOLO) 와 같은 convolutional neural network (CNN) 을 기반으로 한 object detector 들은 우수한 성능을 달성하지만 높은 연산 복잡성과 함께 큰 memory 대역폭을 필요로 한다. 특히 제한된 power budget과 상대적으로 적은 on-chip memory footprint를 요구하는 edge device를 위해 object detector를 설계하는 것은 challenging 하다. 또한 CNN 기반 object detector들은 복잡하고 반복적인 연산을 위해 잦은 external memory access (EMA) 를 필요로 하게 된다. 하지만 잦은 EMA는 가속기의 성능 및 전력 소모 저하를 야기하는 주 원인이 된다. 이를 해결하기 위하여 이전 연구들은 EMA 횟수를 최소화하는 dataflow에 대해 초점을 맞춰왔으며, 이를 통해 전력소모를 최소화하였다. 하지만 EMA를 줄이기 위해 on-chip memory footprint 가 많아지는 문제가 야기되었으며, 특히 EMA를 통한 전력 소모 감소를 달성하였지만, 추가적으로 가속기 내부 자원들의 전력 소모 향상을 위한 연구까지는 도달하지 못하였다.

본 논문에서는 위 문제들을 해결하여 edge device에서도 제한된 자원을 효율적으로 활용될 수 있도록 energy 및 memory 효율적인 YOLO CNN 가속기를 제안한다. 첫번째로 YOLO 가속기 내부의 전력 소모 감소를 위하여 filter switching activity를 99.56% 줄일 수 있는 dataflow를 소개한다. 두번째로는 filter 와 feature map을 효율적으로 사용하기 위하여 multi-bank로 구성된 on-chip memory를 layer별로 reuse할 수 있는 방식을 제안한다. 이 방식을 통해 상대적으로 적은 on-chip memory를 사용하면서도 feature map에 대한 EMA를 완전히 제거함으로써 off-chip communication에 대한 전력 소모를 줄일 수

있다. 세번째로는 FPGA의 block-RAM (BRAM) 에 친화적인 memory bank를 설계하여 filter의 weight와 feature map이 layer별로 재사용 될 수 있도록 한다. 이를 통해 on-chip memory의 utilization을 낮추고 효율성을 높일 수 있게 된다. 마지막으로 본 논문에서 제안하고 있는 YOLO 가속기는 Tiny YOLOv2 모델을 사용하여 Xilinx ZC706 FPGA board에 implementation 하였다. Implementation 결과 5.05W의 전력 소모를 달성함과 동시에 370.5 GOPS의 throughput을 달성하였으며, 하드웨어 자원은 640개의 DSP 와 322.5 개의 BRAM을 사용하였다. 성능 대비 에너지 및 memory 효율은 각각 73.39 GOPS/W, 1.15 GOPS/BRAM을 달성하였다.

주요어: YOLO, FPGA, 가속기, on-chip 메모리, 에너지
학 번: 2020-29972

목 차

표 목차

그림 목차

제 1 장 서 론

1.1 연구의 배경

최근 Convolutional neural network (CNN) 는 image classification, segmentation 그리고 object detection까지 다양한 computer vision 작업에 널리 적용되고 있다. 이 결과, CNN을 기반으로 하는 가속기에 대한 연구 및 활용이 학계 및 산업 분야에서 폭 넓게 진행되고 있다. CNN을 가속하기 위해 graphic processing unit (GPU) 가 널리 활용되고 있지만 GPU는 상당히 많은 전력을 소모한다는 단점이 존재한다. 따라서 GPU 대비 에너지 효율이 좋은 application-specific integrated circuits (ASIC) 과 field programmable gate arrays (FPGA)를 활용한 domain-specific 가속기에 대한 연구가 좀 더 활발히 진행되고 있다. 하지만 ASIC의 경우 설계 및 개발 후 결과 확인까지 상당히 긴 turn-around 시간이 소요되는 반면, FPGA는 reconfigurable 할 뿐만 아니라 개발 시간을 ASIC 대비 상당히 단축시킬 수 있어 FPGA를 기반으로 한 가속기 연구는 GPU 및 ASIC 대비 좀 더 폭 넓게 진행되고 있다.

Internet-of-things (IoT) 장비와 자율주행 자동차와 같은 많은 application에서 사용되고 있는 CNN-based object detector들은 주목할만한 성과를 이루고 있다. 대표적인 CNN-based object detector model로서 you-only-look-once (YOLO) [3], single-shot multi-box detector (SSD) [12] 및 faster region-based CNN (RCNN) [13] 을 예로 들 수 있다. 이 detector 사이에서 YOLO model은 수행 시간 (latency) 과 검출 정확도 (accuracy) 사이에서 적절한 trade-off를 이루고 있으며, 또한 edge device를 위한 tiny version의 YOLO

network도 지원하기 때문에 FPGA-based 가속기에 가장 활발히 적용되고 있다.

하지만 tiny version을 통해 YOLO model의 사이즈를 최소화한다 하여도 edge device에 implement 되기 위해서는 하드웨어 관점에서의 노력이 필요하다. Edge device는 적은 data storage를 가지고 있는 분명한 제약사항이 존재하며, 또한 edge computing 역시 저전력으로 수행되어야 하기 때문이다. 따라서 YOLO model의 edge computing을 위한 가속기 연구가 많이 발표되었으며, 특히 저전력 설계 및 하드웨어 자원 최소화 방안에 대해 활발하게 연구가 이루어지고 있다. 하지만 저전력 설계에 대한 연구는 일반적으로 external memory access (EMA)를 줄이는 dataflow에 초점이 맞춰 진행되었으며, 하드웨어 자원 최소화에 대한 연구는 주로 연산 유닛인 DSP에 집중되었다.

본 연구는 위 초점에서 벗어나 가속기 내부에서 발생하는 switching activity에 초점을 맞춘 저전력 방안에 대해 수행되었으며, 하드웨어 자원은 data storage 측면에 초점을 맞추어 진행되었다.

1.2 연구의 내용 및 논문 구성

본 논문은 크게 세가지 최적화 방안을 제시하면서 에너지 및 메모리 효율적인 YOLO 가속기를 소개함과 동시에 FPGA implementation 결과를 분석한다. 세가지 최적화 방안은 아래와 같다.

1) Dataflow: 본 연구는 가속기 내부 하드웨어 자원의 에너지 절감을 위한 depth-wise filter switching을 제안하고 이를 통해 기존 방식 대비 평균 99.56%의 filter switching activity를 감소시킨다.
2) On-chip memory reuse: 본 연구는 feature map에 대한

external memory access로부터 발생하는 전력 소모를 없애기 위하여 feature map에 대해 on-chip memory만을 활용함과 동시에 on-chip memory 사용량을 줄이기 위한 layer-wise on-chip memory reuse 방식을 제안한다.

3) BRAM-aware memory bank 설계: 본 연구는 비효율적인 BRAM의 활용을 방지하기 위하여 on-chip memory bank 설계 시 BRAM에서 지원 가능한 aspect ratio를 고려하여 on-chip memory 효율성을 높인다. 또한 이를 통하여 filter weight와 feature map을 layer 별로 재사용할 수 있도록 한다.

위 방안을 적용한 YOLO 가속기는 Tiny YOLOv2 모델을 적용하여 Xilinx ZC706 FGPA board에 implementation 되었으며, 370.5 GOPS의 throughput 성능을 달성함과 동시에 5.05W의 적은 전력을 소모하였다. 총 640개의 DSP와 322.5개의 BRAM이 사용되었으며, 성능 대비 에너지 효율은 73.39 GOPS/W를 달성하여 이전 YOLO 가속기 연구들 대비 우수한 에너지 효율을 얻었다.

본 논문은 다음과 같이 구성되었다. 2장에서는 배경 및 관련된 연구에 대해서 살펴보며, 3장에서는 FPGA 내부 하드웨어 자원들의 전력 소모 감소에 크게 기여한 depth-wise filter switching dataflow에 대해 소개한다. 4장에서는 memory 효율성을 향상시키기 위한 YOLO 가속기의 전반적인 구조를 소개함과 동시에, layer-wise on-chip memory reuse 방식과 BRAM-aware memory bank 설계 방식을 제안하고 이에 대해 자세히 다룬다. 5장에서는 제안하고 있는 YOLO 가속기의 FPGA implementation 결과를 분석하고 이전 YOLO 가속기 연구와 성능을 비교하며, 마지막으로 6장을 통해 본 논문을 마무리한다.

제 2 장 관련 연구

2.1 YOLO object detection 알고리즘

Object detection 이란 동영상 혹은 사진에서 사물의 위치를 확인하고 어떤 사물인지 확인하는 computer vision 분야에서 널리 사용되는 기술 중에 하나이다. 최근 몇 년간 convolutional neural network (CNN) 의 발전과 함께 CNN을 기반으로 한 object detection 알고리즘이 등장하였으며 성공적인 성과를 이루었다. 대표적인 예로 single-shot multi-box detector (SSD) [12] 및 faster region-based CNN (RCNN) [13] 등의 알고리즘이 존재한다. 위 알고리즘은 CNN을 기반으로 성공적인 object detection 성능을 제시하였지만, 단일 알고리즘만을 실행해서 동작할 수 없는 구조적인 단점을 가지고 있다.

반면 you-only-look-once (YOLO) [3] 연구는 단일 알고리즘만 적용한 CNN 기반의 object detection 알고리즘을 제안하였으며, 빠른 수행 속도와 높은 정확도의 결과를 보여주었다. 즉, YOLO는 단일 CNN을 통해서 class probability, bounding box 및 confidence score를 동시에 예측할 수 있는 구조적 특징을 가지고 있다. 또한 각 YOLO 알고리즘마다 좀 더 간단하고 가벼운 구조를 지니고 있는 tiny version이 존재하여, edge device에서 활용 가능성을 높인다는 장점을 지니고 있다. 따라서 YOLO object detection 알고리즘은 real-time 구현이 가능한 빠른 수행 속도와 높은 정확도, 그리고 제한된 자원을 요구하는 edge device에서 좀 더 활발히 활용되고 있으며 이를 적용한 가속기 연구 역시 활발하게 진행되고 있다.

앞서 설명한 바와 같이 YOLO는 단일 CNN으로 구성되어 있으며, 그림 2.1의 tiny YOLOv2 network 구조는 YOLO의 단일 CNN의 예를

나타내고 있다. 각 layer는 convolution, batch normalization, pooling 및 activation function을 바탕으로 구성이 되어 있다. 그림 2.2와 같이 CNN의 최종 output은 S×S grid image에 대해서 bounding box, confidence score 및 각 class 별 probability map을 도출하게 되며 이 결과를 바탕으로 post-processing을 통해 최종 object detection 결과를 도출하게 된다.

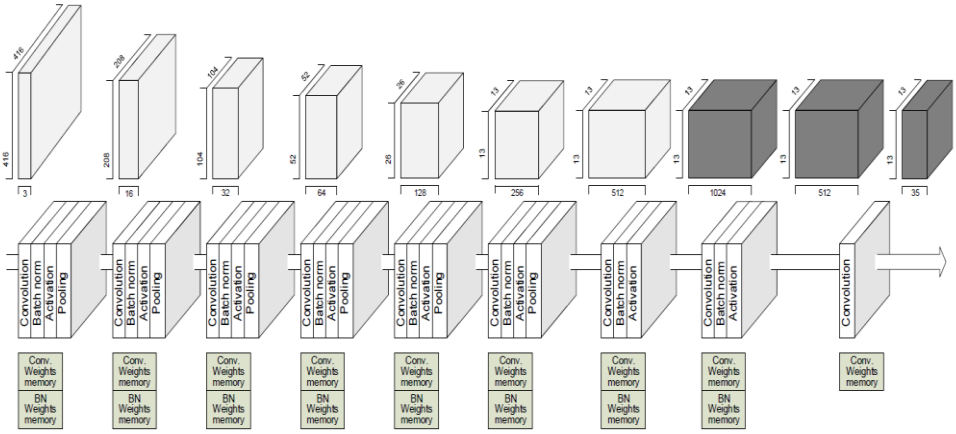


그림 2.1 Tiny YOLOv2 network 구조

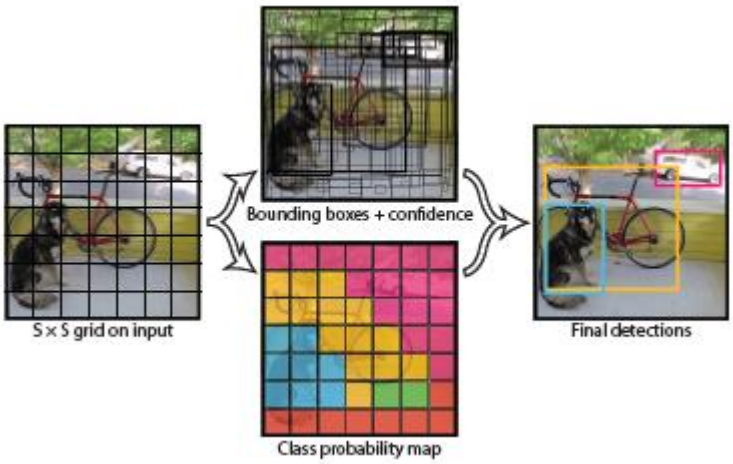


그림 2.2 Single CNN을 통한 YOLO의 최종 object detection 방식

2.2 YOLO 가속기 관련 연구

이전 YOLO 가속기 연구들은 크게 두가지의 architecture를 활용하여 구현되었다. 가장 일반적인 가속기 구조인 single-layer architecture와 layer 간 pipelining을 가능하게 한 streaming architecture이다. 이번 장에서는 이 두 architecture를 사용한 YOLO 가속기들을 살펴본다.

2.2.1 Single-layer Architecture

가장 일반적인 가속기 구조인 single-layer architecture는 layer를 순차적으로 수행하는 방식으로 가속기가 운용된다. 즉, 이전 layer의 연산이 다 마무리된 다음에 다음 layer의 연산을 수행할 수 있으며, 이러한 architecture를 활용한 YOLO 가속기 연구는 활발히 진행되어 왔다 [5] [8] [9]. 이러한 가속기들은 weight와 feature map을 위한 전용 on-chip buffer들이 존재하며, 병렬 연산을 위한 일정량의 data를 off-chip memory로부터 전달받아 on-chip memory에 저장한 후 연산 유닛에 전달해주면서 수행을 하게 된다. Layer의 결과물인 feature map들은 다시 off-chip memory에 저장이 되며 이러한 과정을 여러 layer에 걸쳐 반복하게 된다.

이전 연구 [5] 에서는 416x416 및 288x288의 input feature map 사이즈를 지원하는 multi-resolution YOLO 가속기를 설계하였다. 해당 YOLO 가속기는 feature map에 대한 external memory 접근을 줄이기 위하여 convolutional layer, batch normalization layer 및 leaky-ReLU activation layer를 fusing 하였지만, layer의 input 및 output feature map에 대해서는 여전히 off-chip memory에 접근하도록 구현되었다.

또 다른 연구인 [8] 에서는 real-time application을 위한 high throughput YOLO 가속기를 소개하였다. 해당 연구는 pipeline 구조에서 발생하는 data dependency에 대해 분석하고 dependency 때문에 발생하는 latency 저하를 해소하기 위해 그림 2.3과 같이 FIFO를 두어 MAC 연산을 pipelining 하는 방식을 제안하였다. 이를 통해 high throughput을 달성하였지만, 기존 K×K filter에 대해 conventional loop iteration 방식을 사용하여 27.2W 라는 성능 대비 높은 전력 소모의 결과를 얻었다.

연구 [9] 에서는 FPGA의 한정된 computational capability에 초점을 맞추어 DSP 자원의 사용량 및 energy를 줄이기 위하여 그림 2.4와 같이 input feature map 및 filter에 Winograd 및 fast Fourier transform (FFT)를 적용하였다. 하지만 이 방식은 on-chip memory의 high bandwidth를 요구하기에 on-chip memory의 partition으로 인한 memory 비효율성 증가 및 전반적인 on-chip memory 사용량 증가를 야기하는 단점을 지니고 있다.

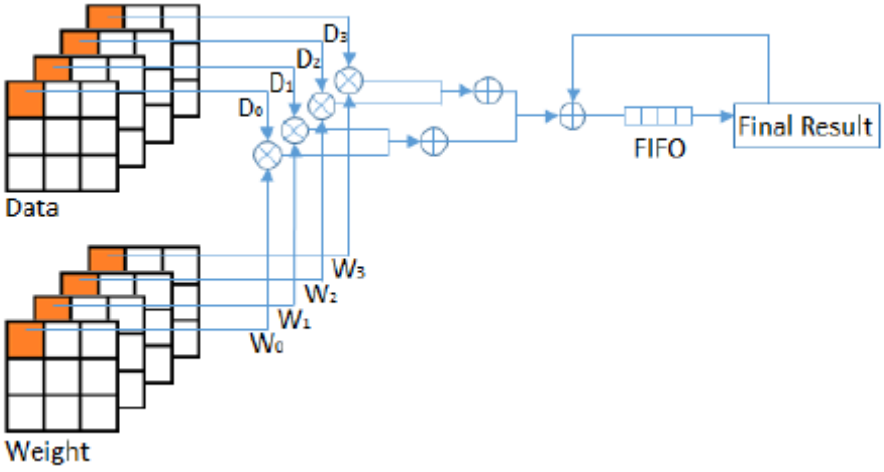


그림 2.3 Pipelined MAC [8]

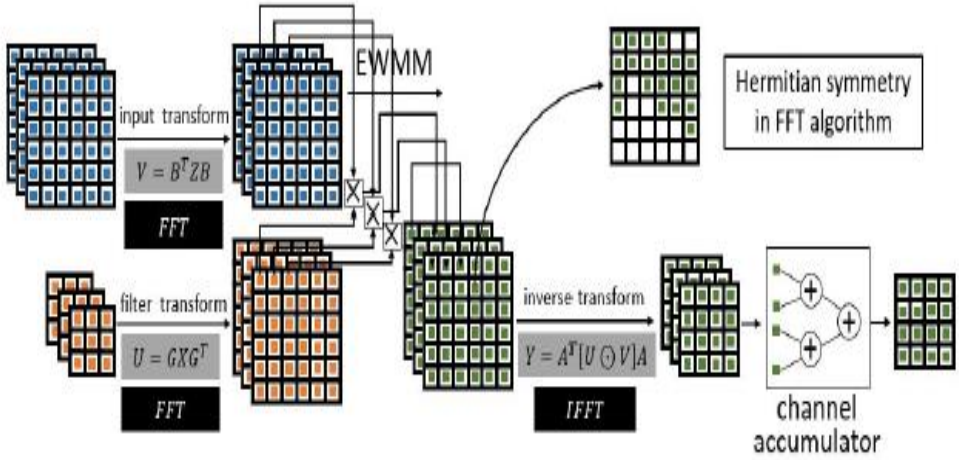


그림 2.4 FFT for input feature map and filter [9]

2.2.2 Streaming Architecture

Feature map에 대한 off-chip memory access는 가속기의 energy와 latency 측면에서 많은 손실을 야기한다. 이러한 문제를 해결하기 위해 나온 architecture가 바로 streaming architecture이다. 그림 2.5은 일반적인 streaming architecture의 구조를 보여주고 있다. Single-layer architecture와 달리 각 layer마다 전용 processing engine이 존재하고 각 processing engine은 feature map, weight를 위한 buffer 및 convolution, pooling 등의 연산을 위한 processing element들이 존재하게 된다. 이와 같은 architecture는 layer 간의 pipelining을 가능하게 하고, feature map에 대한 off-chip memory의 접근을 없앨 수 있다는 장점이 있지만, layer 마다 하드웨어 자원을 할당해야 하므로 상대적으로 긴 layer의 depth를 가지는 모델에는 적합하지 않은 단점이 있다. 또한 weight에 대한 external memory 접근이 가속기의 latency에 대한 bottleneck이 되지 않기 위해 on-chip memory에 대한 high footprint를 요구하는 단점이 존재하기도

한다. streaming architecture를 적용한 YOLO 가속기는 [6], [7] 와 같은 이전 연구에서 구현되었다.

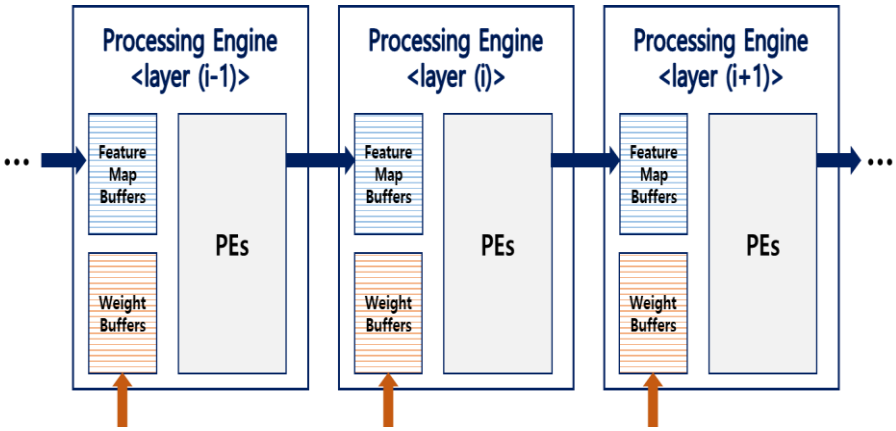


그림 2.5 일반적인 streaming architecture 구조

연구 [6] 에서는 앞서 이야기 한 high on-chip memory footprint에 대한 문제를 해결하고자, weight에 대해서는 binary bit, activation에 대해서는 4~6 bit로 quantization을 적용하여 on-chip memory 사용량을 줄이고자 하였다. 하지만 이와 같이 적은 bit로 quantization을 하게 되면 accuracy drop이 커질 뿐만 아니라, quantization-aware training에 대해 소모되는 시간 역시 무시할 수 없다. 전력 소모 측면에서는 feature map에 대한 off-chip memory 접근을 제거했기 때문에, 앞서 다룬 single-layer architecture를 적용한 연구들보다 향상된 8.7W의 결과를 보여주었지만 edge computing에 충분한 수치라고 보기는 어려운 수준이다.

연구 [7] 에서도 역시 streaming architecture를 적용하여 하드웨어 자원을 주어진 자원에 맞춰 partitioning 하는 framework를 개발함과 동시에 YOLO 가속기를 구현하여 소개하였다. 한정된 자원을

바탕으로 자원을 분배하는 방식은 edge device에 적합할 수 있으나, YOLO 가속기를 구동하기 위한 전력 소모가 명시되지 않아 energy 측면에서의 결과를 확인하기 어려우며, on-chip memory의 효율성 역시 성능대비 낮은 결과를 보여주었다. 또한 filter에 대한 loop iteration 역시 그림 2.6과 같이 기존 conventional 방식인 zigzag 방식을 사용하고 있어 내부 자원의 전력 소모에 대해 고려가 이루어지지 않았음을 확인할 수 있다.

앞으로 3장 및 4장에서는 위에서 다룬 관련 연구들 대비 energy 및 on-chip memory 효율성을 향상시키기 위한 방식들을 제안하고 관련된 내용을 자세히 설명한다.

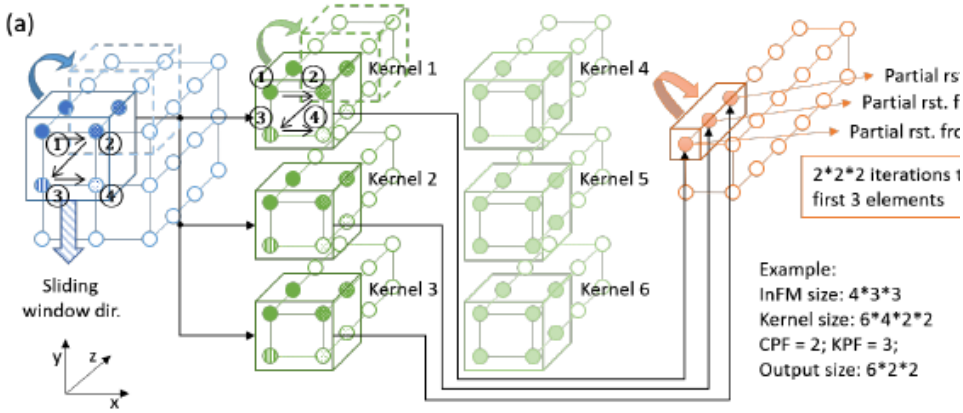


그림 2.6 zigzag loop iteration 방식 [7]

제 3 장 Depth-wise Filter Switching Dataflow

3.1 Conventional Dataflow 분석

그림 3.1에서 볼 수 있듯이 NVDLA [5] 및 기존 YOLO 가속기 [7] [8] 에서는 T_i (= input channel에 대한 병렬 연산 factor) 개의 MAC 유닛들이 연산을 매 cycle 동안 병렬처리 수행하기 위해 $1\times1\times T_i$ 의 input tensor에 접근하게 된다. $K\times K\times T_i$ tensor에 대해 연산하기 위해서 $1\times1\times T_i$ tensor가 zigzag 순서로 연산이 진행되며 총 $K\times K$ cycle이 소요된다. 즉 3×3 filter size에 대해서 총 9 cycle을 소요하게 된다. 그림 3.2 와 같이 pseudo code로 살펴보면 $K\times K$ kernel에 대한 이중 iteration loop가 존재하여, zigzag 방향으로 loop를 수행하게 된다. 하지만 filter의 weight를 매 cycle 마다 새롭게 load 하는 것은 전력 소모 측면에서 매우 비효율 적이다. 수학적으로 zigzag 순서로 연산을 수행 시 발생하는 filter switching activity의 횟수는 식 (3.1) 과 같이 계산된다.

$$N_{FSA\text{-}zigzag} = K \times K \times W \times H \times ceil\,(IN\_CH / T_i) \qquad (3.1)$$

여기서 K > 1 이고, W, H 및 IN_CH는 각각 feature map의 width, height 그리고 input channel을 나타낸다. 특히 filter는 output channel에 대한 병렬 연산을 위하여 T_o (= output channel에 대한 병렬 연산 factor) 만큼의 weight set를 같은 feature map과 공유하면서 연산하기 때문에 filter에 대한 switching은 feature map에 대한 switching보다 더 큰 영향을 미치는 요소라고 할 수 있다. 위와 같이 병렬 연산을 위해 on-chip memory에 pre-fetch 되어 있는 weight를

register에 load해야 하는데 이때 필요한 register bit의 양이 상당히 크며 식 (3.2) 와 같이 계산할 수 있다.

$$\#\ of\ bits\ for\ weight\ registers = W_{bit} \times T_i \times T_o \qquad (3.2)$$

여기서, W_{bit}는 weight의 bit를 말한다. 또한 T_o 만큼의 filter weight를 연산 유닛에 전달해주기 위해서는 최소 T_o 만큼의 on-chip memory를 두고 pre-fetch 해야 한다. 따라서 register 와 on-chip memory의 사용량이 많을 수밖에 없기 때문에 전력 소모 역시 switching activity에 민감하게 반응할 수밖에 없다. 결국 잦은 filter switching은 가속기 내부 하드웨어 자원들의 전력 소모를 증가시키게 된다.

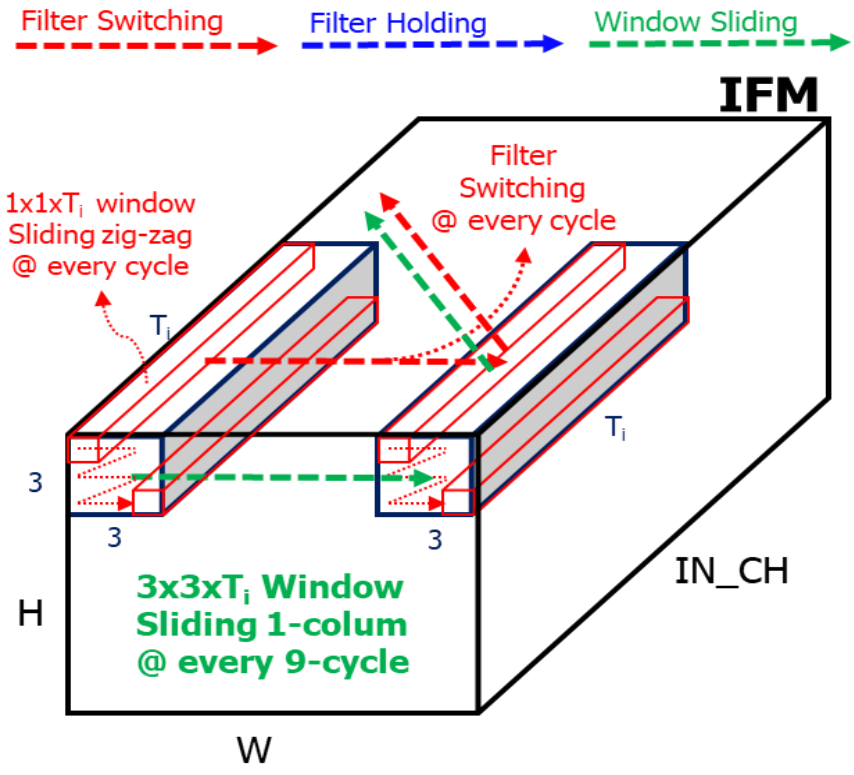


그림 3.1 Conventional dataflow의 zigzag filter switching 방식

```
#pragma Unroll "ti"
for (ti=0; ti<IN_CH; ti+=(Ti)) {
    ...
    // iteration for (K x K) kernel
    for (i=0; i<K; i++) {
        for (j=0; j<K; j++) {
            ofm[to][row][col] +=
                ifm[ti][S*row+i][S*col+j] *
                wgt[to][ti][i][j];
        }
    }
}
```

그림 3.2 Pseudo code: K×K kernel에 대한 이중 iteration loop

3.2 Depth-wise Filter Switching Dataflow

3.2.1 Depth-wise Filter Switching Dataflow

3.1 장에서 다룬 바와 같이 conventional dataflow는 하드웨어 내부 자원의 잦은 switching activity로 인하여 전력 소모의 증가를 야기한다. 이를 해결하고 가속기 내부의 전력 소모 감소를 위하여 본 연구는 depth-wise filter switching을 위한 새로운 dataflow를 제안한다. 그림 3.2에서 나타내듯이 $1\times1\times T_i$ 구조로 연산을 수행하는 기존 방식 대신에 $K\times K\times(T_i/K^2)$의 형태로 연산을 수행하도록 변경하였다. 즉, K = 3인 경우 $3\times3\times(T_i/9)$ 형태로 연산을 수행하게 된다. 우선 pseudo code로 dataflow가 어떻게 변경되었는지 살펴보면, 그림 3.3과 같이 기존 K×K iteration loop는 unroll 되고, 대신 input channel에 대한 parallel factor T_i는 K^2배 감소하게 된다.

그림 3.4는 K = 3, T_i = 36 인 경우에 대해서 설명하고 있다. 3×3×4 feature map window에 대해 convolution 연산을 수행하기 위해 weight register들은 on-chip memory 로부터 weight 값을 load하게

되고 3×3×4 window가 줄의 끝까지 도달할 때까지 값이 유지하게 된다 (그림 3.4의 파란색 점선). 3×3×4 window는 input channel 방향으로 그 다음 depth를 위해 진행하게 되고, 이 때 새로운 filter 가 weight register에 load 되게 되며 (그림 3.4의 빨간색 점선) 앞선 depth에서 수행한 바와 같이 동일하게 줄의 끝 방향을 향해 연산을 수행하게 된다. 이러한 과정은 input channel을 향해 반복되게 되며, 완료 시 그 다음 줄의 첫번째 depth로 이동하여 다시 반복하게 된다. 따라서 filter의 switching은 같은 depth에 대해서는 변하기 않게 되며 depth가 바뀔 때마다 발생하게 된다. 따라서 제안하고 있는 depth-wise filter switching dataflow를 통해 filter switching activity 횟수를 계산해보면 식 (3.3) 과 같이 계산될 수 있다.

$$N_{FSA\text{-}proposed} = H \times IN\_CH / (T_i / 9) \tag{3.3}$$

이를 통해 3×3 convolutional layer에 대해 그림 3.5와 같이 하드

```
#pragma Unroll "ti"
for (ti=0; ti<IN_CH; ti+=(Ti/(K*K))) {
    ...
    #pragma Unroll "i" and "j"
    for (i=0; i<K; i++) {
        for (j=0; j<K; j++) {
            ofm[to][row][col] +=
                ifm[ti][S*row+i][S*col+j] *
                wgt[to][ti][i][j];
        }
    }
}
```

그림 3.3 Depth-wise filter switching pseudo code

웨어 자원인 on-chip memory, weight register 및 DSP에 대해 read enable rate 및 switching activity를 감소시킬 수 있다.

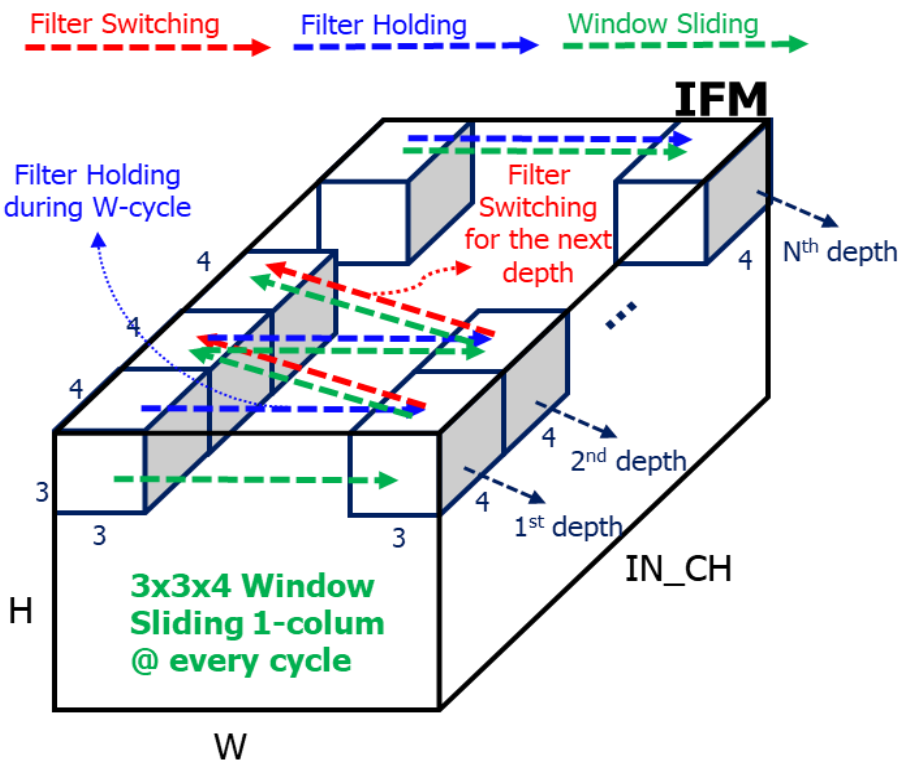


그림 3.4 Depth-wise filter switching 방식

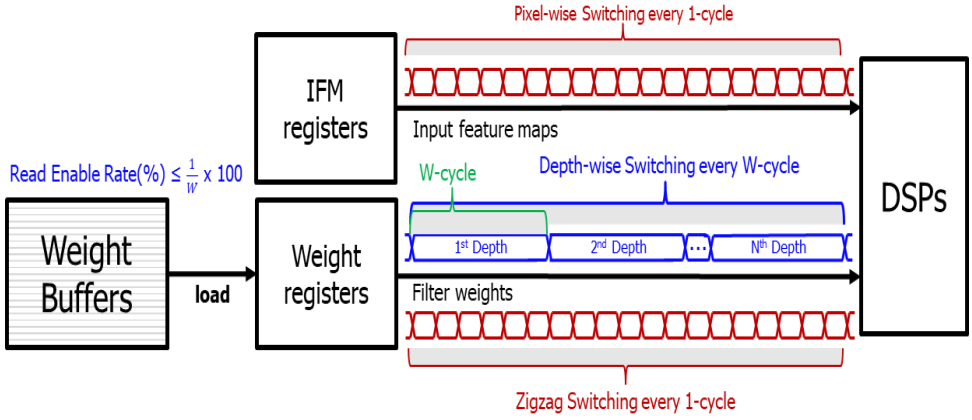


그림 3.5 가속기 내부 하드웨어 자원에 대한 switching activity

3.2.2 Filter Switching Activity 분석

표 3.1 은 depth-wise filter switching을 통해 conventional 방식 대비 Tiny YOLOv2 모델의 layer별로 얼마나 switching activity가 줄어드는지 나타내고 있다.

표 3.1 Tiny-YOLOv2 모델의 layer별 filter switching activity

Layer	Filter	In_ch①	W②	H③	FSA④ (1-frame)		
					Zigzag	Proposed	Reduced ratio (%)
1	3×3	3	416	416	1557504	416	99.97
2	3×3	16	208	208	389376	832	99.79
3	3×3	32	104	104	97344	832	99.15
4	3×3	64	52	52	48672	832	98.29
5	3×3	128	26	26	24336	832	96.58
6	3×3	256	13	13	12168	832	93.16
7	3×3	512	13	13	22815	1664	92.71
8	3×3	1024	13	13	44109	3328	92.46
9	3×3	512	13	13	195	195	0.00
Total	-	-	-	-	2196519	9763	99.56

식 (3.1) 및 (3.3)을 통해 각 방식의 switching activity를 계산할 수 있으며, 3×3 convolutional layer에 대해 기존 방식 대비 switching

① Input channel
② Width
③ Height
④ FSA: Filter Switching Activity

activity를 현저하게 감소시켜, 전체 layer에 대해 평균 99.56%의 감소율을 나타내었다. 특히 YOLO 모델은 convolutional layer의 대부분이 3×3 filter size로 구성이 되어 있다는 특징을 지니고 있어 제안하는 dataflow를 통해 switching activity 절감 효과를 극대화할 수 있다. 1×1 convolutional layer에 대해서는 $3\times3\times(T_i/9)$ 구조에서 간단하게 $1\times1\times T_i$ 형태로 변경이 가능하여 기존 방식과 동일한 dataflow를 사용할 수 있다.

3.2.3 전력 소모 분석

3.2.1에서 이야기하는 바와 같이 depth-wise filter switching을 통해 switching activity를 99.56% 감소시킬 수 있다. 이 감소율이 하드웨어 내부 자원의 전력 소모를 각각 얼만큼 줄일 수 있는지를 알아보도록 한다.

병렬 연산 수행을 위해 filter의 weight를 load하는 register의 전력 소모를 분석하기 위해 우선 CMOS의 전력 소모를 살펴보면 식 (3.4)와 같이 dynamic power와 static power로 나눌 수 있다.

$$P_{total} = P_{dyn} + P_{static} \tag{3.4}$$

static power는 dynamic power 대비 굉장히 작으므로 무시할 수 있으며, dynamic power는 (3.5) 와 같이 표현할 수 있다.

$$P_{dyn} = \alpha \times f \times C_{eff} \times V_{dd}^2 \tag{3.5}$$

여기에서 α는 switching activity factor, f는 switching frequency, C_{eff}

는 effective capacitance 마지막으로 V_{dd}는 supply voltage를 의미한다. 따라서 식 (3.5) 를 통해 알 수 있듯이 CMOS로 구성된 register의 전력 소모는 switching activity에 linear한 함수임을 알 수 있다. 즉 depth-wise filter switching을 통해 줄어든 99.56%에 비례하여 filter의 weight를 load 하고 있는 register의 전력 소모 역시 99% 이상 줄일 수 있음을 뜻한다.

FPGA의 on-chip memory 자원인 BRAM의 경우 memory cell array 영역과 내부 logic 영역으로 나눌 수 있다. BRAM의 전력 소모는 Xilinx Power Estimator [17] tool을 이용하여 예측이 가능한다. 이 때 입력해줘야 하는 read enable rate 값은 가속기 마다 다르기 때문에 maximum (=100%), typical (=75%) 및 minimum (=50%)의 세가지 조건으로 분류해서 BRAM의 전력 소모 감소율을 예측하였으며, 표 3.2는 각 조건 별 BRAM의 전력 소모의 감소율을 나타내고 있다.

표 3.2 BRAM의 전력 소모 예측 감소율[⑤]

Cond.	Read enable rate (%)			P_{BRAM} (mW) [⑥]		
	Zigzag	Proposed	Reduced ratio	Zigzag	Proposed	Reduced ratio
Max.	100.00	0.44	99.56	7.85	0.41	94.8%
Typ.	75.00	0.33	99.56	5.98	0.40	93.3%
Min.	50.00	0.22	99.56	4.11	0.39	90.6%

⑤ 36Kb-BRAM에 대해 200MHz의 주파수로 전력 소모 예측
⑥ $P_{BRAM} = P_{VCCINT} + P_{VCCBRAM}$

표 3.2에서 볼 수 있듯이 BRAM의 전력 소모를 기존 zigzag 방식 대비 최소 90.6%에서 최대 94.8%까지 줄일 수 있는 것을 확인할 수 있다.

제안하는 dataflow는 그림 3.5과 같이 weight의 switching activity를 줄임으로써 input tensor와 weight의 곱셈을 수행하는 DSP의 내부에서 발생하는 glitch를 줄일 수 있는 효과가 있다. 특히 전력 소모 측면에서 살펴볼 때, multiplier는 ISCAS'89 benchmark circuit 중에서 glitch에 가장 영향을 많이 받는 연산이며 glitch가 발생하지 않는 경우와 비교했을 때 glitch는 곱셈 연산의 전력 소모를 3.7배 증가시킬 수 있다 [18]. Depth-wise filter switching을 통해 줄어드는 glitch의 비율은 weight 와 input tensor 모두 8-bit 이므로, 그림 3.6 와 같이 conventional 8 × 8 multiplier 기반으로 분석하였다. 8 × 8 multiplier는 8개의 half adder, 48개의 full adder 및 64개의 AND gate 로 이루어져 있으며, 각 gate count (2-input NAND gate 기준) 는 식 (3.6) ~ (3.8)와 같이 계산된다.

$$Half\ adder = 8 \times 5 = 40\ g/c \tag{3.6}$$

$$Full\ adder = 48 \times 9 = 432\ g/c \tag{3.7}$$

$$AND\ gate = 64 \times 2 = 128\ g/c \tag{3.8}$$

multiplier의 연산자인 weight 가 static 한 경우에 (그림 3.6의 Y0~Y7을 weight로 간주) AND gate에서 glitch 발생하지 않게 된다. AND gate는 8 × 8 multiplier의 전체 gate count (2-input NAND gate 기준) 의 21.3% 를 차지하고, depth-wise filter switching을 통해 줄어드는 switching activity ratio는 99.56% 이므로 8 x 8 multiplier에서

depth-wise filter switching을 통해 줄어드는 glitch의 비율 식 (3.9)와 같이 21.2% 가 된다.

$$Glitch\ \ 감소율\ = 99.56 \times 0.213 = 21.2\% \qquad (3.9)$$

따라서 제안된 dataflow를 통해 곱셈 연산에서 발생하는 glitch를 줄일 수 있으므로 glitch로 인한 DSP의 전력 소모 역시 감소시킬 수 있다.

종합해보면, depth-wise filter switching dataflow를 통해 가속기 내부 하드웨어 자원인 register, on-chip memory 및 DSP의 전력 소모를 감소시킬 수 있어 에너지 효율을 높일 수 있게 된다.

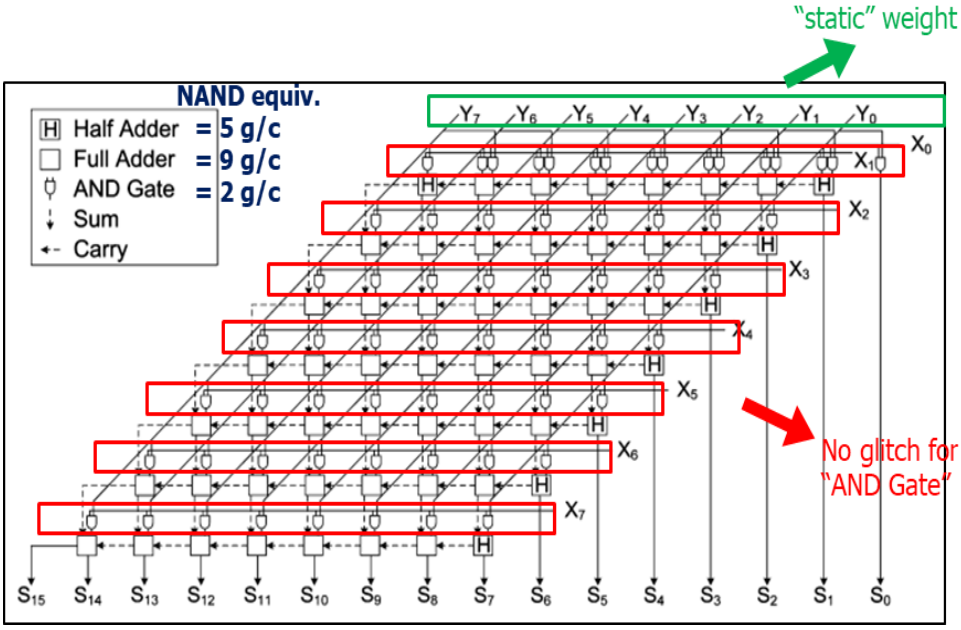


그림 3.6 8 × 8 multiplier의 glitch 분석

제 4 장 YOLO CNN 가속기 Architecture

4.1 Layer-wise On-chip Memory Reuse

Filter의 weight와 feature map을 저장하기 위해 요구되는 pattern과 양이 다르기 때문에 대다수의 domain-specific 가속기는 그림 4.1-3과 같이 weight 및 feature map를 각각 따로 운용할 수 있도록 buffer를 따로 두어 설계를 한다. 하지만 이 구조를 이용하여 off-chip memory 접근을 없애기 위해 feature map을 전부 저장하기에는 on-chip memory 사용량이 많아지는 단점이 있다 [5] [8]-[10]. Feature map에 대해 off-chip memory access를 없애기 위하여 그림 4.4와 같이 layer 간 pipelining을 가능하게 만든 streaming architecture를 활용한 가속기도 등장하였지만, 각 layer별로 buffer를 할당해야 하므로 on-chip memory footprint가 높아질 수 있다 [6] [7].

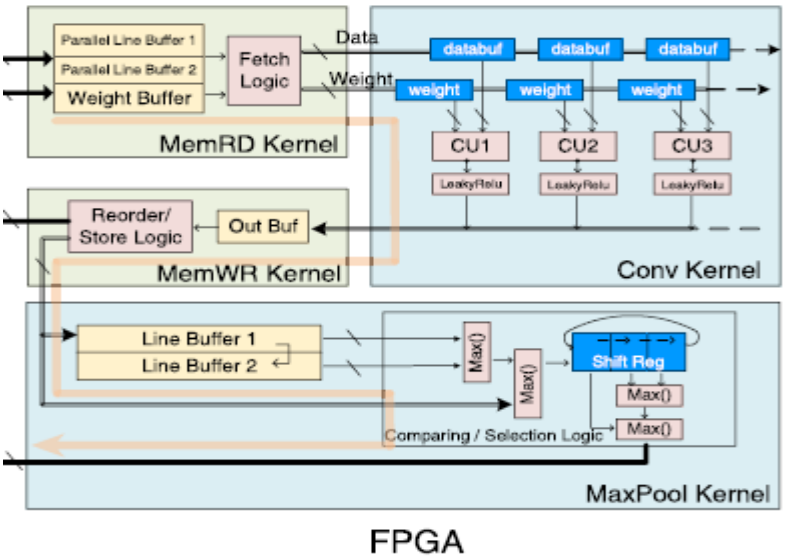


그림 4.1 dedicated hardware accelerator [5]

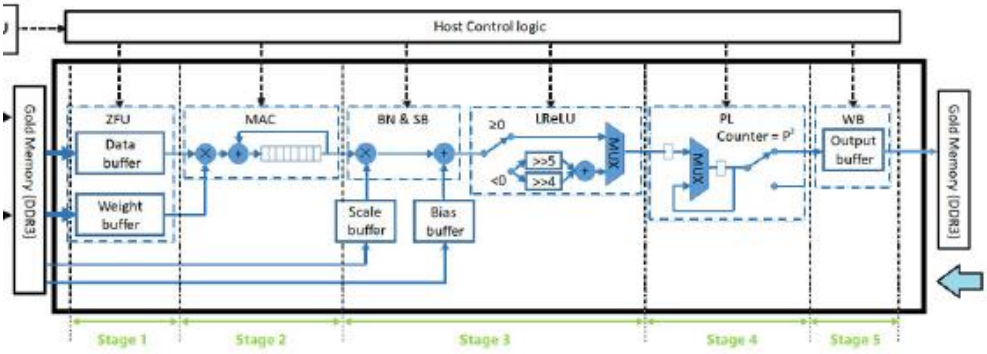


그림 4.2 A novel FPGA accelerator [8]

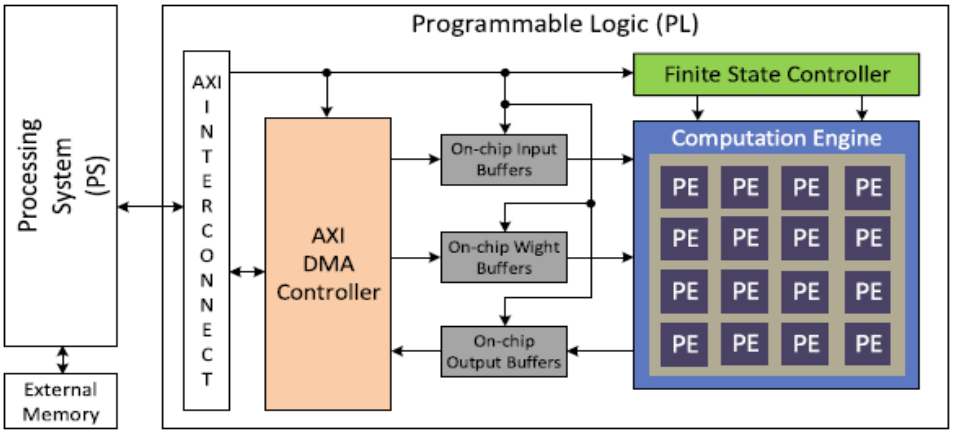


그림 4.3 YOLO accelerator using WALLACE tree adder [10]

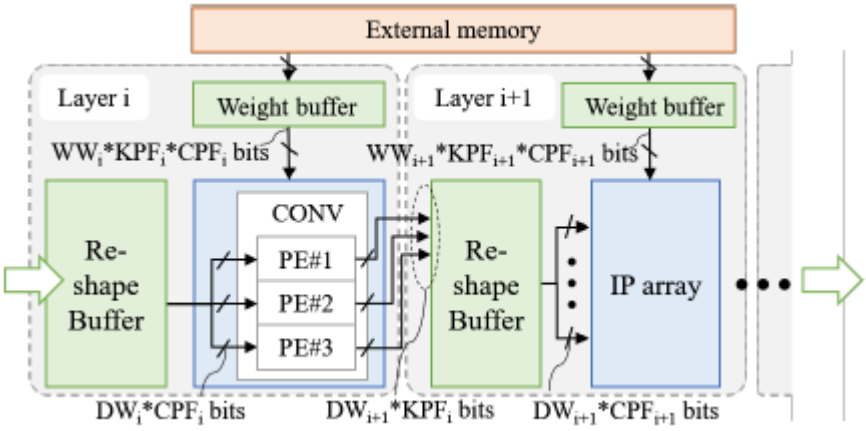


그림 4.4 DNN builder with streaming architecture [7]

위와 같은 문제를 해결하기 위해 본 논문에서는 layer-wise on-chip memory reuse 방식을 제안한다. 이 방식을 위해 그림 4.5 와 같이 multiple banks, encoder 및 decoder 로 구성된 layer-wise reusable on-chip memory (LR-OCM) 을 설계하였다.

YOLO 모델은 그림 4.6과 같이 feature map의 data volume이 layer를 지날수록 pooling layer에 의해 점점 작아지게 되는 반면, weight의 경우 output channel의 수가 증가하기 때문에 weight의 data volume이 급격하게 증가하게 된다. 따라서 layer 별로 on-chip memory를 weight와 feature map 사이에서 재사용할 수 있도록 설계하였다. 즉 weight와 feature map에 대해 각각 따로 할당된 buffer를 사용하지 않고 global 한 on-chip memory를 두어 layer 별로 data량을 분석하여 재사용하도록 운용하기 때문에, 불필요한 on-chip memory 사용을 줄일 수 있는 효과가 발생하게 된다.

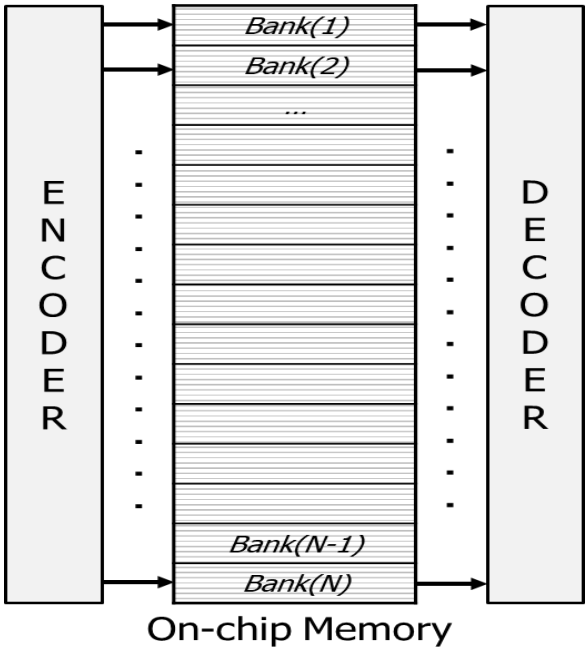


그림 4.5 Layer-wise reusable on-chip memory

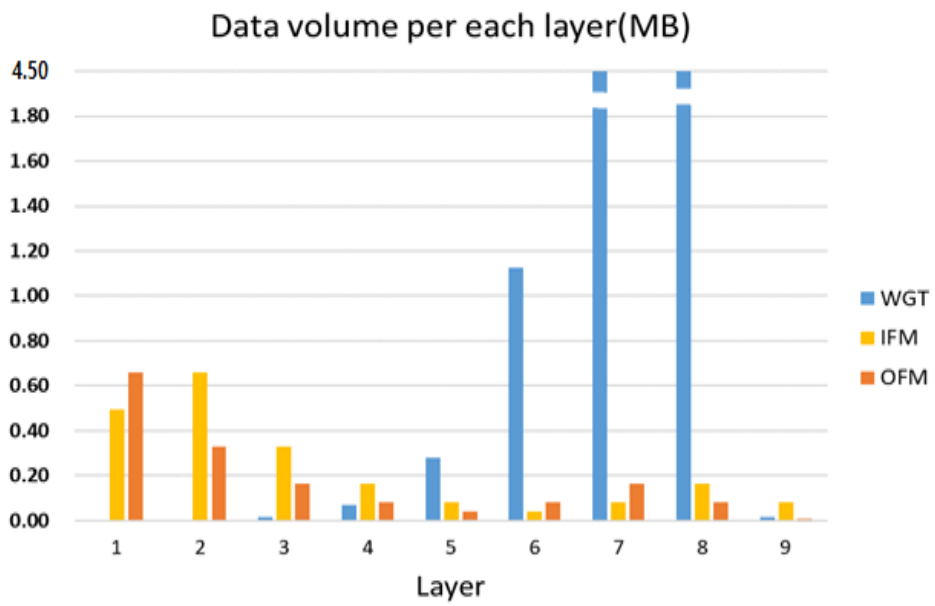


그림 4.6 Tiny-YOLOv2 모델의 layer별 data volume

이를 활용하여 기존 연구들 대비 상대적으로 적은 on-chip memory를 사용하여 output feature map 전부 on-chip에 저장하여, output feature map에 대한 off-chip memory access를 완전히 제거하였다. Single-layer architecture를 활용한 구조에서 중간의 모든 feature map을 off-chip memory access 없이 on-chip memory 만을 사용하여 구동하는 것은 상당히 많은 on-chip memory 사용량을 요구하기 때문에, 이전 YOLO 가속기 연구에서는 구현된 적이 없는 것으로 확인된다. 즉, 제안하고 있는 layer-wise on-chip memory reuse 방식을 사용하여 output feature map에 대해 on-chip memory 만을 활용한 single-layer architecture 구조의 YOLO 가속기는 본 논문에서 처음으로 소개되는 것으로 확인된다. 그림 4.7은 LR-OCM의 encoder, decoder 및 multi-bank on-chip memory 로 구성된 세부 구조를 표현하고 있다. On-chip memory의 모든 bank들은 encoder 및

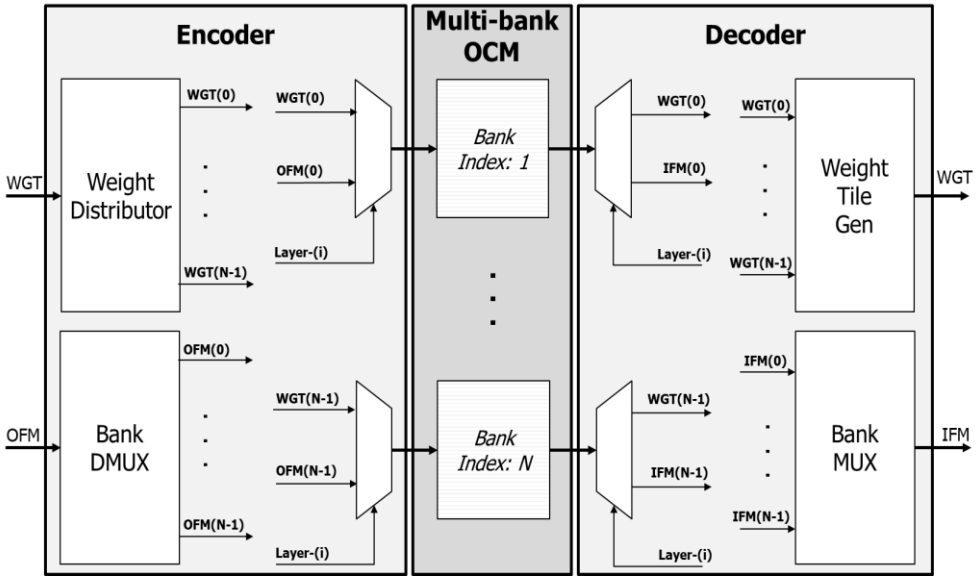


그림 4.7 Layer-wise reusable on-chip memory의 세부

decoder와 연결이 되어 있다.

Encoder는 external memory 접근을 통해 들어오는 weight data를 input으로 받게 되고, Weight Distributor 유닛을 통해 weight가 분배되게 되며, layer index에 따라 선택된 bank 들에 weight가 저장되게 된다. 또한 encoder의 Bank DMUX 유닛은 들어오는 output feature map (첫번째 layer의 경우에는 input feature map) 을 row index에 따라 선택된 bank 들에 저장하게 된다. 이렇게 output feature map을 위해 선택된 bank 들은 자연스럽게 그 다음 layer의 input feature map을 저장하고 있는 bank로 활용되게 된다.

Decoder는 convolution 연산을 위해 weight를 저장하고 있는 bank들로부터 T_o 만큼의 weight를 Weight Tile Gen 유닛에 load 하게 되고, weight register로 구성된 Weight Tile Gen 유닛은 load 된 weight를 각 연산 유닛에 맞게끔 분배하고 전달하는 역할을 수행한다. 또한 Bank MUX 유닛을 통해 input feature map을 row 순서에 맞춰

bank로부터 읽어온 후 LR-OCM 외부의 line buffer에 pre-fetch 하는 역할을 수행한다. 위와 같은 방식으로 on-chip memory는 weight와 feature map 사이에서 layer 별로 재사용이 되고, 그림 4.8은 각 bank들이 어떻게 layer별로 재사용이 됐는지 구체적으로 보여준다. 대표적인 예로 그림 4.9와 같이 231번째 bank의 사용되는 data를 분석해보면, layer별로 사용되는 data type이 가변적인 것을 알 수 있다. LR-OCM의 bank 개수는 최대로 사용되는 layer에 의해 결정되게 되는데, Tiny-YOLOv2 모델에서는 layer-6에서 weight bank 256개, feature map bank 26개, 총 282개가 사용되게 된다.

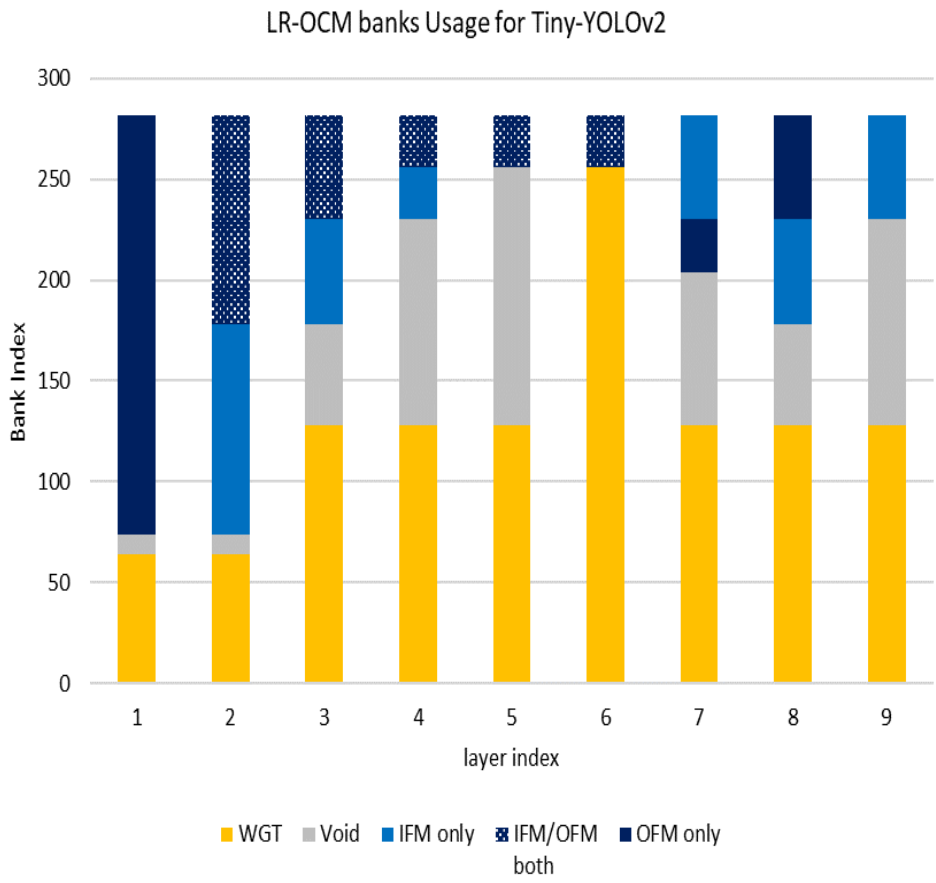


그림 4.8 layer별 multi-bank 사용 방식

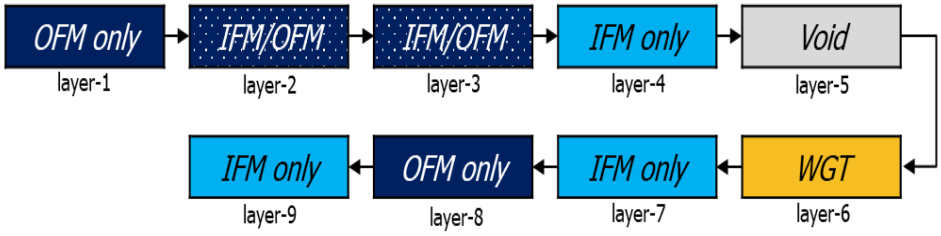


그림 4.9 231번 bank의 layer별 data 사용 분석

이전의 많은 가속기 연구에서 알 수 있듯이 일반적으로 input 및 output feature map에 대해서 두개의 buffer를 두어 ping-pong 방식으로 재사용할 수 있는 double buffering 방식을 사용한다. 하지만 초반 layer에서의 feature map size가 크기 때문에 double buffering 방식은 더 많은 on-chip memory 사용량을 필요로 하게 된다. 이를 방지하기 위해 row 순서대로 처리할 수 있도록 설계하였으며, 이에 따라 LR-OCM은 초반 layer에 대해서 LR-OCM 외부에 존재하는 line buffer들에 pre-fetch 하는 방식을 사용하였다. 그림 4.10과 같이 input feature map을 line buffer에 pre-fetch 하여 bank를 비우고 그 row의 결과 값을 다시 비워진 bank에 overwrite 할 수 있도록 하여 feature map 전체를 double buffering 할 필요가 없어지도록 설계하였다. 그림 4.8의 “IFM/OFM both” index는 앞서 설명한 방식대로 운용되는 bank들을 나타낸다. 위 방식을 통하여 초반 layer에 대한 on-chip memory 사용량을 절약할 수 있다. 반면 후반부의 layer들은 feature map의 사이즈가 작아져 전제 on-chip memory 사용량에 영향을 주지 않기 때문에 double buffering을 할 수 있는 기능을 지원하게끔 설계되었으며, 그림 4.8의 layer-7, layer-8에서 볼 수 있듯이 “OFM only”, “IFM only” 별로 따로 저장하는 bank가 존재하

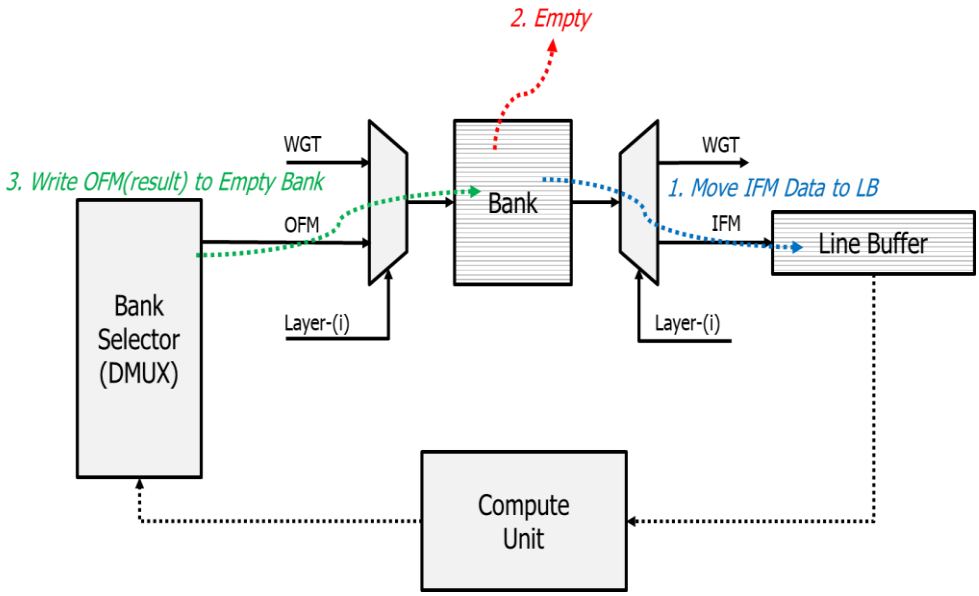


그림 4.10 “IFM/OFM both” bank의 사용 방식

는 것을 확인할 수 있다. Weight에 대한 off-chip memory access의 횟수를 줄이고 및 off-chip memory access 때문에 발생할 수 있는 latency bottleneck을 피하기 위하여, layer-1부터 layer-6 까지는 weight를 전부 on-chip memory에 저장할 수 있도록 했으며, 이전 layer에서 feature map을 위해 사용되었던 bank들을 재사용하기 때문에 on-chip memory 사용량 증가 없이 가능하도록 설계가 되었다. Layer-7, layer-8에서는 weight의 data volume이 4.5 MB로 on-chip memory에 전부 저장하기에는 매우 크기 때문에 off-chip memory에 접근하며 일정 weight volume을 caching 및 double buffering 할 수 있도록 하였다.

4.2 BRAM-aware Memory Bank Design

FPGA는 on-chip memory 역할을 하는 block-RAM (BRAM) 이

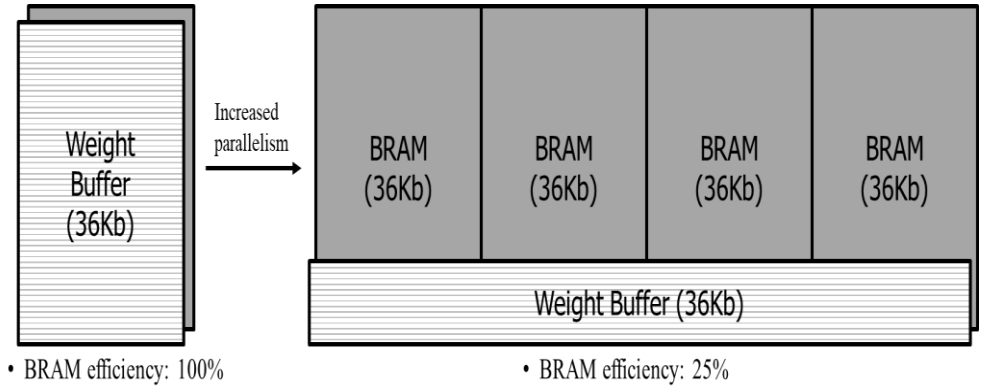


그림 4.11 Parallelism 증가에 따른 BRAM efficiency 감소

존재하지만 BRAM이 제공하는 aspect ratio는 제한되어 있으며 일반적으로 bit width 가 좁고 depth가 긴 형태를 지니고 있다 (e.g., 32K×1, 16K×2, 8K×4, 4K×9, 2K×18, 1K×36, 512×72). 이전 많은 연구들은 가속기가 더 많은 병렬 연산을 수행할수록 weight에 대한 on-chip memory bandwidth가 증가하기 때문에 width가 더 넓은 weight buffer를 사용해야 한다고 지적하고 있다 [19]-[21]. 하지만 BRAM의 aspect ratio는 위에서 설명한 것과 같이 제약사항이 존재하기 때문에, 같은 data 저장 용량을 지닌 buffer라도 더 넓은 buffer width를 사용할수록 BRAM의 사용량이 증가하게 되는 문제점을 지니고 있다. 그림 4.11은 36Kb weight buffer를 width를 4배 늘리고 depth를 1/4 배 줄였을 때 1개의 BRAM으로 사용 가능한 buffer가 4개까지 증가될 수 있는 예를 보여주고 있다. 즉 BRAM의 efficiency는 100%에서 25%까지 감소될 수 있다는 것을 보여준다.

하지만 가속기의 구조가 연산 속도에 맞추기 위해 매 cycle 마다 weight load를 요구한다면, memory width를 넓혀 bandwidth를 높여줘야 하는 것은 불가피한 선택이다. 만약 memory를 펼치지

않는다면 가속기의 throughput에 영향을 미치기 때문에 BRAM의 사용량과 throughput은 서로 trade-off 관계에 놓이게 된다. 또한 제약된 BRAM의 aspect ratio에 설계자가 원하는 buffer의 shape을 mapping 시키는 것 역시 어려운 일이며, 특히 본 논문에서 제안하는 바와 같이 weight와 feature map을 재사용 할 수 있도록 공유하는 buffer라면 BRAM의 aspect ratio에 mapping하는 것은 더 어려운 과제이다.

이러한 문제를 해결하고 BRAM efficiency를 향상시키기 위하여 본 논문에서는 depth-wise filter switching dataflow의 이점을 적극 활용하여 BRAM-aware memory bank를 설계하였다. 3장에서 다뤘듯이 depth-wise filter switching을 통해 매 cycle 마다 weight load를 요구하는 구조에서 벗어나 depth마다 한 번씩 load할 수 있게끔 dataflow가 설계되었다. 따라서 weight buffer에 대해 요구하는 band-

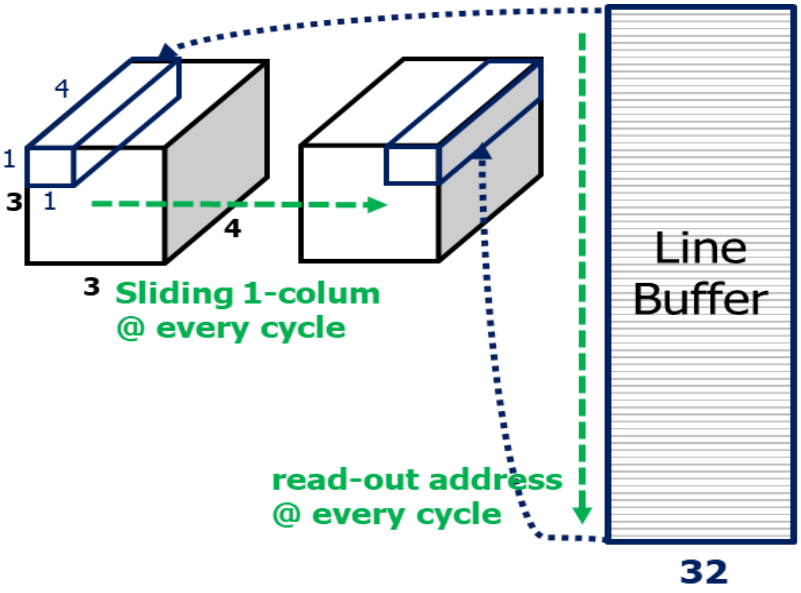
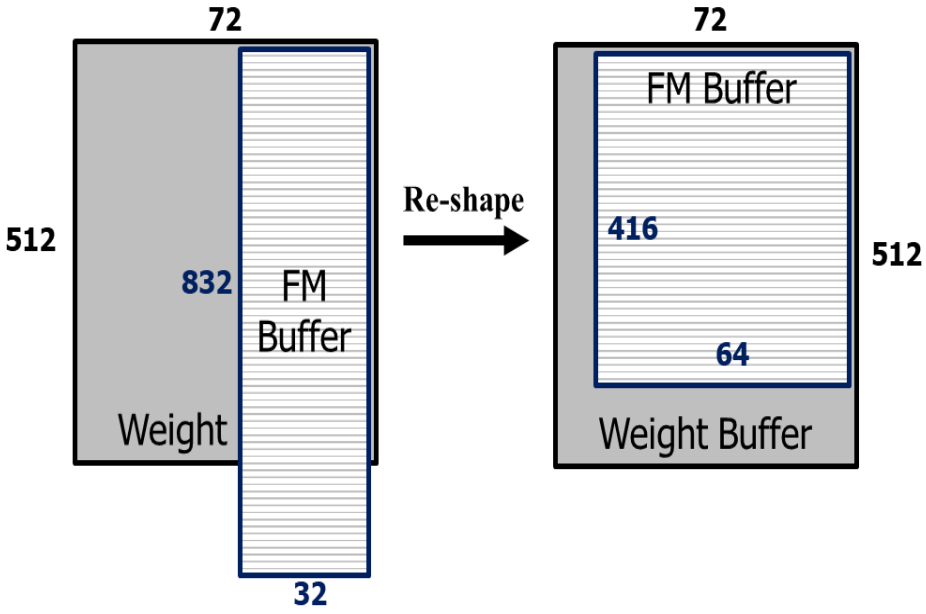


그림 4.12 feature map buffer share for 1-row



(a) (b)

그림 4.13 (a) feature map buffer reshape 전 (b) feature map buffer reshape 후

width를 줄일 수 있어 더 이상 wide한 buffer를 필요로 하지 않게 된다. 따라서 BRAM이 지원하는 aspect ratio를 벗어나지 않고 36Kb-BRAM의 512×72의 aspect ratio에 맞추기 위해 address당 3×3×W_{bit}[⑦]를 저장할 수 있도록 weight buffer의 width를 설정하였다.

Feature map buffer의 aspect ratio는 3×3×4 sliding window 구조의 장점을 활용하여 설정하였다. 3×3×4 window를 매 column sliding 하기 위해서는 그림 4.12 와 같이 1×1×4의 input feature map tensor가 매 cycle 마다 read-out 되어야 하며, Tiny-YOLOv2 모델의 1-row를 위해 832의 depth가 필요로 하게 된다. 따라서

⑦ W_{bit} = 8-bit for weight data

832x32[⑧]의 buffer aspect ratio가 필요하지만 이는 그림 4.13(a)와 같이 weight buffer와 공유할 수 없는 shape 되며, 공유하더라도 하나의 BRAM을 더 cascade하여 공유해야 하므로 비효율적인 BRAM 사용을 피할 수 없게 된다. 따라서 weight buffer와 input feature map이 하나의 BRAM만을 이용해 재사용 될 수 있는 bank를 만들기 위하여 feature map buffer를 그림 4.13(b)와 같이 re-shape 하고 weight 와 feature map buffer 모두 36Kb-BRAM에서 재사용될 수 있도록 구성하였다. 이와 같이 BRAM-aware memory bank 설계를 통해서 가속기의 throughput 손실 없이 BRAM의 효율성을 높이고 불필요한 사용량을 줄일 수 있게 하였다.

4.3 Overall Architecture

그림 4.14은 본 논문에서 제안한 방식을 적용하여 설계한 YOLO 가속기의 전반적인 architecture를 보여주고 있다. 가속기는 external memory와 통신을 담당하는 Direct Memory Access (DMA) 유닛, input feature map의 window를 생성하고 column 방향으로 sliding 하는 역할을 수행하는 Sliding Window Gen (SWG) 유닛, pooling layer의 연산을 수행하는 Pooling 유닛, 4.1에서 다룬 LR-OCM 그리고 convolution, accumulation, batch-normalization 및 activation을 수행하는 Processing Elements (PEs) 로 구성되어 있다.

Direct memory access (DMA) 유닛은 128-bit AXI4 bus를 통해서 external memory에 저장되어 있는 weight 와 input feature map을 읽어오는 역할을 수행한다. 중간 layer의 output feature map은 전부 on-chip memory에 저장이 되지만 마지막 layer의 output feat-

[⑧] Input tensor = 8-bit

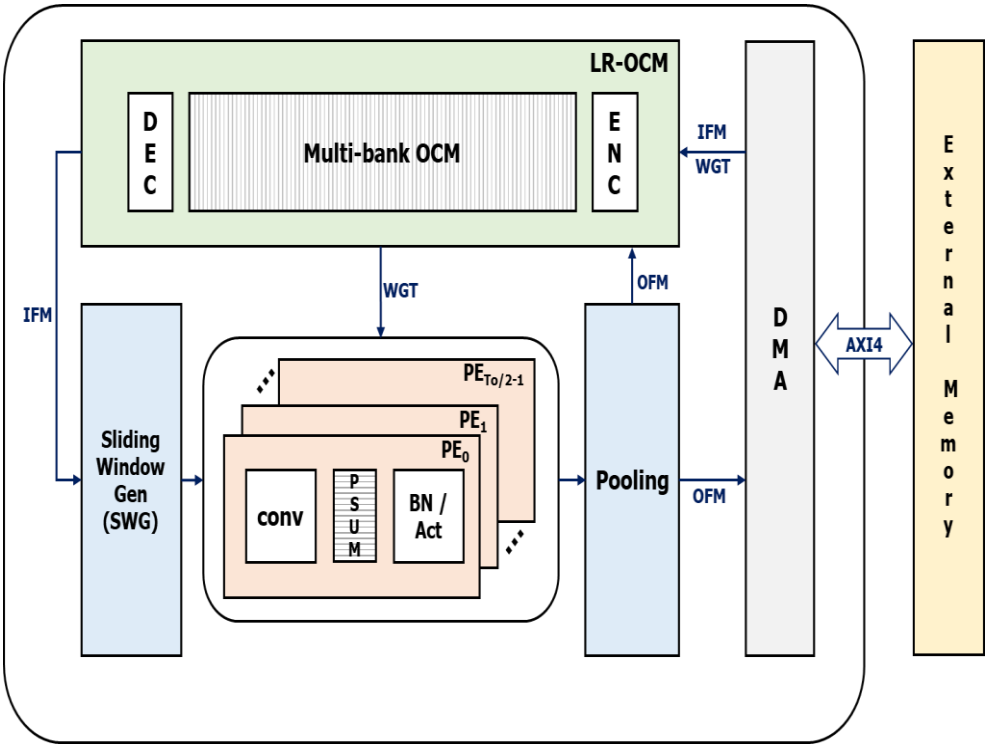


그림 4.14 제안하는 YOLO 가속기의 overall architecture

ure map은 DMA를 통해서 external memory 에 write 되게 된다. Line buffer 들로 구성된 SWG 유닛은 그림 4.15와 같이 3×3×4 window를 생성하고 1 column씩 sliding 하기 위해 line buffer 들을 sequential 하게 read-out한다. 생성된 3×3×4 window는 1×1×36 형태로 vectorization 후 PE unit에 전달하게 된다. SWG unit에서 생성된 feature map window는 $T_o/2$ 개의 PE들에게 공유되고, 이에 mapping이 되는 weight set 역시 LR-OCM의 decoder로부터 각 PE로 전달되게 된다. 그림 4.16은 PE의 세부 block diagram을 나타내고 있다. CONV 유닛에서 multiply-accumulate (MAC) 연산은 수행하게 되고, partial summation 연산을 위한 partial sum (Psum) buffer가 존재한다. Input channel에 대한 Psum 연산이 마무리되면 BN_ACT 유

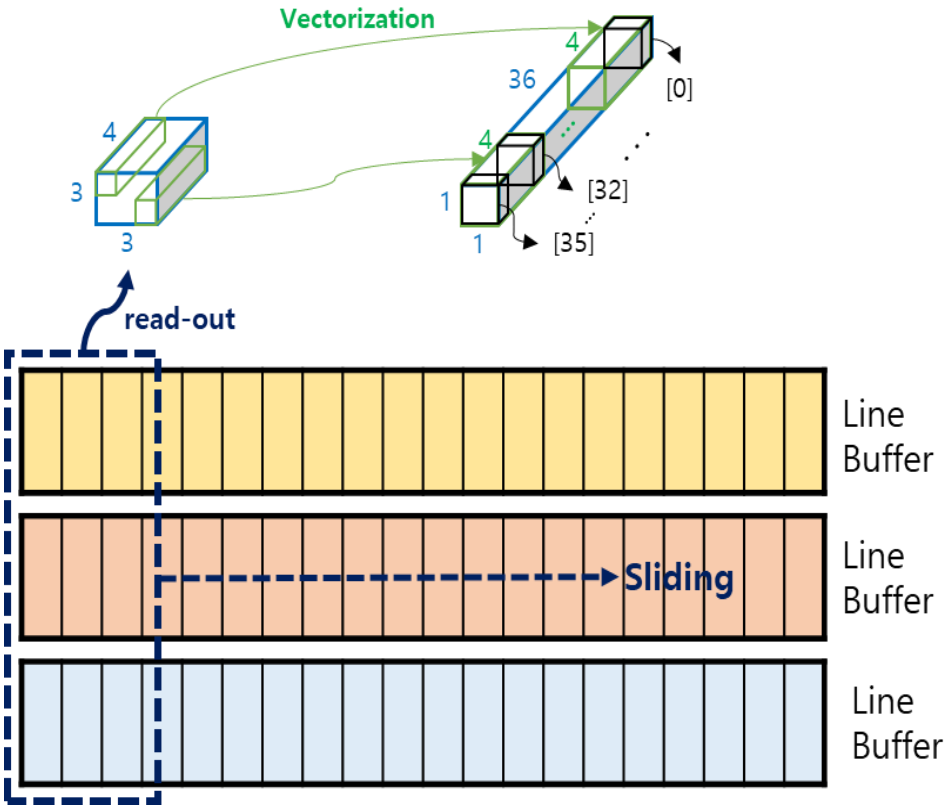


그림 4.15 SWG 유닛의 3×3×4 window 생성 및 vectorization

닛에서 batch-normalization 연산 및 activation 연산을 수행하게 된다. 하나의 PE에서 두개의 weight set (WGT_{To-odd}, $WGT_{To-even}$) 를 전달받아 연산을 수행하게 되는데 이는 multiplication 연산을 수행하는 DSP를 효율적으로 활용하여 DSP 자원의 낭비를 막기 위함이다. FPGA의 DSP를 살펴보면 그림 4.17 와 같이 pre-adder logic이 존재하고 25bit x 18bit의 multiplication 연산을 수행할 수 있도록 되어있다. DSP에서 제공되는 multiplication bit-width가 weight, input bit-width인 8-bit 보다 훨씬 크기 때문에 8bit 곱셈 연산 하나만 수행하면 자원의 낭비가 발생하게 된다. 따라서 DSP를 최대한 효율적으로 활용하기 위하여, 그림 4.18 와 같이 두개의 weight를

25bit input A에 concatenation 하여 하나의 multiplier를 가지고 두개의 multiplication을 수행할 수 있도록 설계하였다. 이를 통해 그림 4.14에서 볼 수 있듯이 T_o개의 PE가 아닌 $T_o/2$개의 PE를 가지고 연산을 수행이 가능하게 된다. Pooling 유닛은 max pooling을 수행하게 되면 stride=1, 2에 대해 지원할 수 있도록 설계되었다. Pooling의 결과인 output feature map은 off-chip memory access를 줄이기 위해 다시 LR-OCM으로 전달이 되고 encoder에 의해서 선택된 bank에 저장이 되게 된다. 반면 마지막 layer에서의 최종 output feature map은 DMA를 통해서 external memory에 저장이 되게 된다.

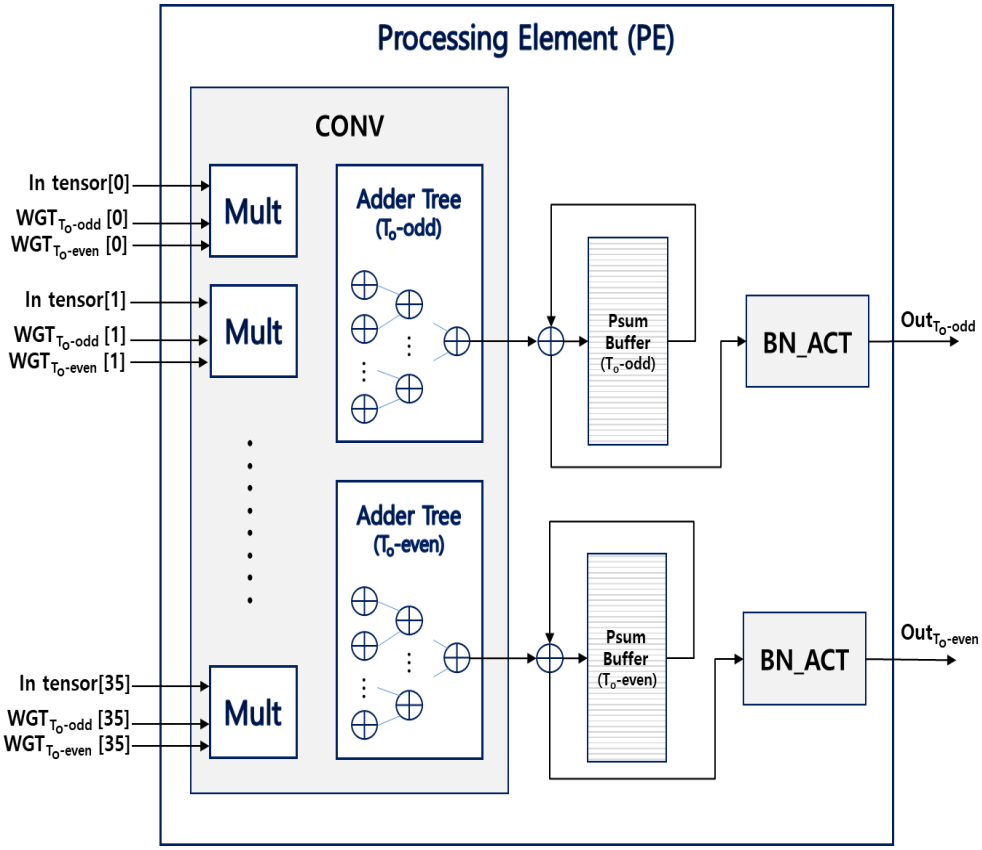


그림 4.16 PE block diagram

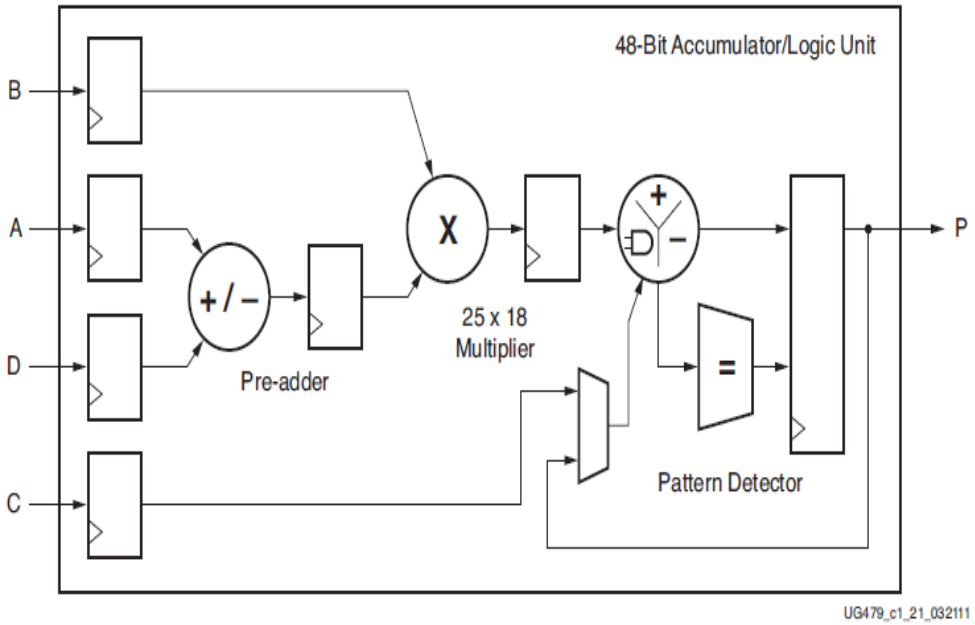


그림 4.17 FPGA DSP48E1 slice functionality

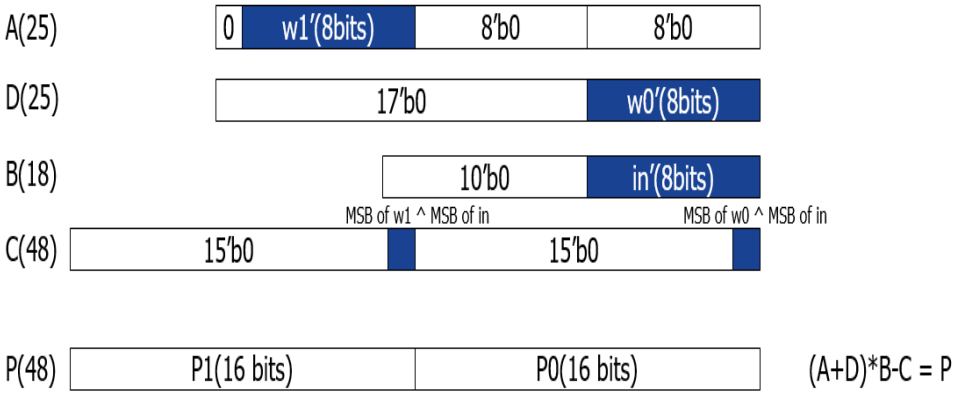


그림 4.18 Double multiplication in DSP

제 5 장 실험 결과 및 분석

5.1 실험 결과

제안하고 있는 YOLO 가속기는 Tiny-YOLOv2 모델을 적용하였으며, Xilinx ZC706 FPGA board에 200MHz의 주파수로 implementation 되었다. Parallel factor인 T_i와 T_o는 각각 36과 32로 설정하였으며, weight, activation, scale 및 bias는 각각 8-bit, 8-bit, 16-bit, 16-bit으로 설계되었다.

FPGA Implementation 결과 총 640개 DSP, 322.5개의 36Kb-BRAM이 사용되었으며, 370.5 GOPS의 throughput 성능을 달성하였다. 반면 전력 소모는 5.05W로 낮출 수 있었으며, energy 효율성 및 memory 효율성은 각각 73.39 GOPS/W, 1.15 GOPS/BRAM을 달성하였다. Weight와 feature map 저장을 위해 총 282개의 BRAM이 LR-OCM에 사용되었으며, 추가적으로 40.5개의 BRAM이 SWG, Psum, Pooling 및 DMA 유닛에서 사용되었다. Convolution 연산을 위해서 총 576개 (= $T_i \times T_o$ / 2) 가 사용되었으며, batch-normalization의 경우 Psum 의 결과값이 DSP에서 지원하는 25bit보다 크기 때문에 두개의 DSP가 cascade 되어 batch-normalization의 scale multiplication을 수행하였다. 그 결과 총 64개의 DSP가 batch-normalization 연산을 위해 사용되었다.

그림 5.1은 전력 소모를 각 자원 별로 break-down 하여 보여주고 있다. 그림에서 알 수 있듯이 총 282개의 BRAM을 사용했지만 BRAM의 전력 소모는 0.087W로 depth-wise filter switching dataflow를 통해 전력 소모를 최소화할 수 있음을 확인하였으며, logic 역시 0.498W로 상당히 적은 수치를 나타내고 있음을 확인할 수 있다.

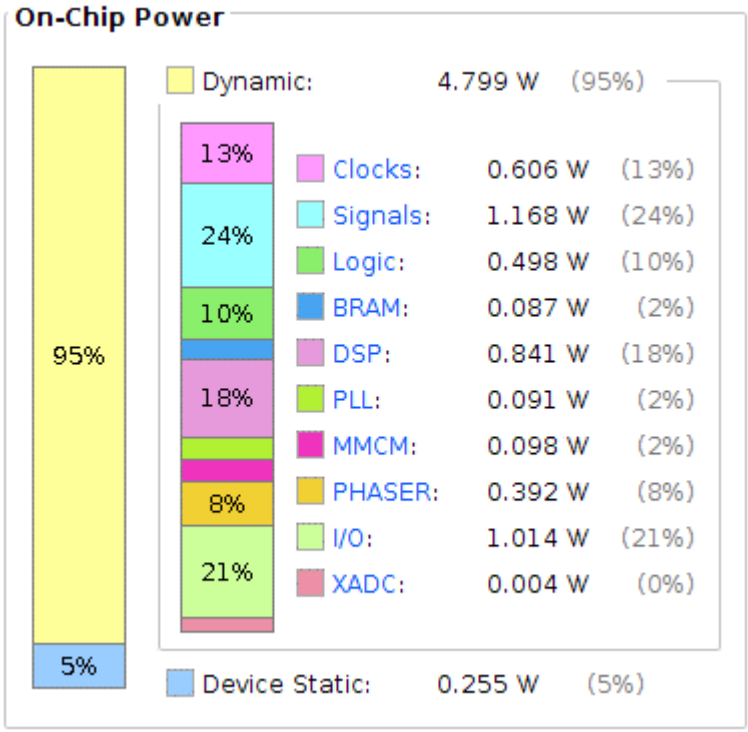


그림 5.1 전력 소모 break-down

5.2 결과 분석 및 비교

제안하고 있는 YOLO 가속기의 architecture는 single-layer 방식으로 동작하면서 중간 feature map을 external memory 접근 없이 on-chip memory 만을 활용하여 동작하도록 설계되었다. 이와 같이 동작하는 architecture는 이전 YOLO 가속기 연구에 등장한 적이 없는 것으로 확인되기 때문에, 성능 비교를 위하여 single-layer architecture를 사용한 YOLO 가속기와 streaming architecture를 활용한 YOLO 가속기 모두 비교 연구에 포함하였다. 이전 가속기 연구 [5] [8]는 intel FPGA board에 implementation 했기 때문에 BRAM의 사용량이 20Kb-BRAM을 기준으로 결과가 명시되어 있다[⑨]. 따라서

[⑨] Intel FPGA board의 BRAM size: 20Kb

공정한 성능 비교를 위하여 20Kb-BRAM 사용량을 36Kb-BRAM으로 환산하였다. 표 5.1은 본 논문에서 제안하는 YOLO 가속기와 이전 YOLO 가속기 연구와의 성능 비교를 위하여 결과값을 나타내고 있다.

Single-layer architecture를 활용한 YOLO 가속기 [5] [8] [9] 와 성능을 비교해 보면, 본 논문의 가속기는 energy efficiency 측면에서 각각 3.82배, 3.43배 및 2.73배 향상된 결과를 보여주고 있다. 이전 연구들은 feature map load 및 store를 위하여 계속적으로 off-chip memory에 대한 접근을 해야 하기 때문에 더 많은 전력소모를 하고 있는 것으로 판단된다. 또한 본 가속기는 feature map 저장을 위해 전부 on-chip memory를 활용했음에도 불구하고 memory efficiency 측면에서 각각 2.35배, 3.11배, 1.20배 향상되었음을 확인할 수 있다. 이는 layer별로 on-chip memory를 weight와 feature map사이에서 재사용을 했기 때문에 향상된 것으로 분석된다.

Streaming architecture를 사용한 YOLO 가속기 [6] [7] 와 비교해 보면, memory efficiency 측면에서 각각 1.26배, 1.64배 향상된 결과를 확인하였다. 이는 steaming architecture 구조에서 높은 throughput을 달성하기 위해 weight load 및 store 이 bottleneck이 되지 않기 위해 각 layer별로 weight buffer를 많이 사용할 가능성이 높은 것으로 분석된다. Energy efficiency 측면에서 보면 [7]는 전력 소모 값이 없어 비교하지 못하였으며, [6] 대비 1.37배 향상된 결과를 나타내었다. 특히 [6]는 weight를 binary bit로 quantization 하고 activation을 4~6 bit으로 quantization 했음에도 불구하고 본 가속기는 memory와 energy 측면에서 더 우수한 결과를 나타내었다.

표 5.1 이전 YOLO 가속기 연구와의 성능 비교

	ICCAD'18 [7]	**TCAD'19 [9]**	**TVLSI'19 [6]**	**IEEE ACESS'20[8]**	**JRTIP'21 [5]**	**This Work**
Platform	ZC706	ZC706	VC707	Arria-10	Arria-10 GX1150	ZC706
Network	Tiny-YOLOv2	Tiny-YOLO	Tiny-YOLOv2	Tiny-YOLOv2	Tiny-YOLOv2	Tiny-YOLOv2
Frequency(MHz)	200	166	200	200	190	200
Architecture	streaming	single-layer	streaming	single-layer	single-layer	single-layer
Precision (W, A)	16	16	(1, 6)	(8, 16)	8	8
Image Size	1280x384	224x224	416x416	416x416	416x416	416x416
Throughput (GOPS)	234	201.1	464.7	731.9	500	370.6
BRAMs (36Kb)	333	545	513	759(1366*)	1027(1849*)	322.5
DSPs	680	900	168	410	884	640
Power (W)	N/A	9.4	8.7	27.2	26	5.05
Memory Efficiency (GOPS/BRAM)	0.7	0.37	0.91	0.96	0.49	1.15
Energy Efficiency (GOPS/W)	N/A	21.39	53.41	26.91	19.23	73.39

제 6 장 결 론

본 논문에서는 energy와 memory 효율성을 향상시킬 수 있는 Tiny-YOLOv2 가속기를 제안하였다. 가속기 내부 하드웨어 자원의 전력 소모를 감소시키기 위하여 depth-wise filter switching dataflow를 제안하였으며, layer-wise on-chip memory reuse 방식을 통해 on-chip memory의 사용량을 최소화하였다. 또한 BRAM-aware memory bank design을 통해 on-chip memory의 효율성을 증가시켰으며, weight와 feature map 모두 재사용할 수 있는 bank를 설계하였다. 이를 통해 single-layer architecture를 기반으로 322.5개의 BRAM만으을 사용하여 feature map 대한 off-chip memory access 없이 동작할 수 있도록 설계하였다. 이 결과, 본 논문에서 제안하고 있는 YOLO 가속기는 이전 연구보다 우수한 73.39 GOPS/W의 energy efficiency 및 1.15 GOPS/BRAM의 memory efficiency를 달성하였다.

참고 문헌

[1] NVDLA Deep Learning Accelerator, https://nvdla.org. 2017.

[2] M. Kim et al., "A Real-Time 17-Scale Object Detection Accelerator With Adaptive 2000-Stage Classification in 65 nm CMOS," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 10, pp. 3843-3853, Oct. 2019

[3] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 7263-7271.

[4] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, “A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA,” In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18), pp. 31–40.

[5] Xu, K., Wang, X., Liu, X. et al., "A dedicated hardware accelerator for real-time acceleration of YOLOv2," J Real-Time Image Proc 18, pp. 481–492, 2021

[6] D. T. Nguyen, T. N. Nguyen, H. Kim and H. Lee, "A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 8, pp. 1861-1873, Aug. 2019.

[7] X. Zhang et al., "DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs," 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018, pp. 1-8.

[8] S. Li, Y. Luo, K. Sun, N. Yadav and K. K. Choi, "A Novel FPGA Accelerator Design for Real-Time and Ultra-Low Power Deep Convolutional Neural Networks Compared With Titan X GPU," in IEEE Access, vol. 8, pp. 105455-105471, 2020.

[9] Y. Liang, L. Lu, Q. Xiao and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 4, pp. 857-870, April 2020.

[10] F. U. D. Farrukh et al., "Power Efficient Tiny Yolo CNN Using Reduced Hardware Resources Based on Booth Multiplier and WALLACE Tree Adders," in IEEE Open Journal of Circuits and Systems, vol. 1, pp. 76-87, 2020.

[11] C. Ding, S. Wang, N. Liu, K. Xu, Y. Wang, and Y. Liang, "REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs," In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), pp. 33–42.

[12] Liu W. et al., “SSD: Single Shot MultiBox Detector,” In European conference on computer vision, pages 21‑37, Springer, 2016.

[13] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” arXiv preprint arXiv:1506.01497, 2015.

[14] J. Li et al., "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 343-348.

[15] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks,” In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15), pp.161–170.

[16] A. Stoutchinin, F. Conti, and L. Benini, “Optimally scheduling cnn convolutions for efficient memory access,” arXiv preprint arXiv:1902.01492, 2019.

[17] Xilinx, Inc., “Power Methodology Guide”, 2013.

[18] J. Lamoureux, G. G. F. Lemieux and S. J. E. Wilton, "GlitchLess: Dynamic Power Minimization in FPGAs Through Edge Alignment and Glitch Filtering," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 11, pp. 1521-1534.

[19] J. Vasiljevic and P. Chow, "Using buffer-to-BRAM mapping approaches to trade-off throughput vs. memory use," 2014 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, pp. 1-8.

[20] M. Kroes, L. Petrica, S. Cotofana, and M. Blott. “Evolutionary bin packing for memory-efficient dataflow inference acceleration on FPGA,” In Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20), pp. 1125–1133..

[21] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana and M. Blott, "Memory-Efficient Dataflow Inference for Deep CNNs on FPGA," 2020 International Conference on Field-Programmable Technology (ICFPT), 2020, pp. 48-55.

Abstract

Filter Switching Optimization for An Energy-Efficient YOLO Accelerator

Kyeongjong Lim

Electrical and Computer Engineering

The Graduate School

Seoul National University

Convolutional neural network (CNN) based object detectors such as the you-only-look-once (YOLO) achieve remarkable performance but come with high computing complexity and a large memory bandwidth. Therefore, it is challenging to design an accelerator for such object detectors on edge devices, which have a limited power budget and relatively small on-chip memory footprint. Especially, CNN-based object detectors require frequent external memory access for complex and periodic computation. However, the frequent off-chip memory access is the main reason for decreasing the performance in terms of throughput and energy-efficiency.

To address this issue, previous works focus on the dataflow for reduce the number of external memory access resulting in power consumption decrease. However, it causes high on-chip memory footprint to reduce external memory access and also the power consumption of internal hardware resources is not investigated from these works.

In this paper, we propose an energy- and memory-efficient CNN accelerator for YOLO. First, we propose a novel dataflow which reduces the amount of filter switching by 99.56% on average. Second, we propose a layer-wise on-chip memory reuse scheme in which multi-bank on-chip buffers are efficiently utilized for both feature maps (FMs) and filters without the need to access external memory for FMs. Third, we present FPGA BRAM-aware memory bank design to be reused between weight and feature map across layers. The proposed design is implemented on a Xilinx ZC706 FPGA and consumes power of 5.05W but achieves a throughput rate of 370.5 GOPS for Tiny-YOLOv2 while using only 640 DSPs and 322.5 BRAMs. Our design achieves energy efficiency of 73.39 GOPS/W, thus outperforming those in previous works.

Keywords: YOLO, FPGA, Accelerator, On-chip memory, Energy
Student Number: 2020-29972