SystemC기반 CNN 시뮬레이터 구현

김진영*, 이승수*, 김예준*, 임승호*, 조상영*
*한국외국어대학교 컴퓨터.전자시스템공학부
kestrel08@naver.com ,ab72002338@kakao.com , idoyo7@gmail.com,
slim@hufs.ac.kr, sycho@hufs.ac.kr

SystemC-based CNN Simulator

Jinyoung Kim*, Seungsu Lee*, Yejun Kim*,
Seung-Ho Lim*, Sang-Young Cho*
*Div. of Computer and Electronic Systems Engineering, Hankuk University of Foreign Studies
kestrel08@naver.com ,ab72002338@kakao.com , idoyo7@gmail.com,
slim@hufs.ac.kr, sycho@hufs.ac.kr

요 약

최근 엣지 컴퓨팅과 같은 임베디드 디바이스에서 CNN과 같은 딥러닝 모듈을 수행하기 위해서 하드웨어 설계 및 구현이 많이 진행되고 있다. 이러한 임베디드 시스템에 필요한 CNN모듈을 위한 하드웨어 설계를 위해서 먼저 모델링을 통해서 시뮬레이션이 필요하다. 본 논문에서는 오픈 라이센스를 이용한 RISC-V로 딥러닝 시뮬레이터를 제작하였다. SystemC로 구현된 RISC-V를 Virtual Platform로 시뮬레이터의 제작을 하여 시뮬레이팅을 하였고, SystemC의 특징인 모듈화와 모듈간 통신에 유의하여 시스템을 구성하였다. CNN 알고리즘을 참조하여 Convolution, Activation, Pooling 연산의 기능을 하는 시스템을 구성하였다.

1. 서론

최근 4차 산업혁명에서의 발전으로 AI 및 딥러닝에 대한 관심사가 크게 증가하고 있으며 이에 따라 딥러닝 연산기에 대한 수요도 같이 증가하였다. CNN과 같은 딥러닝 연산은 연산량이 많아 일반적으로 GPU와 같은 고속 연산기를 이용하여 소프트웨어 딥러닝 연산을 수행한다.

최근 엣지 컴퓨팅과 같은 컴퓨터 시스템에서는 단말기 자체에서의 딥러닝 연산을 수행하기 위해서 저사양 프로세서에 최적화된 하드웨어 딥러닝 연산기를 가지도록 설계되고 있다. 하지만 하드웨어 설계 및 구현은 검증이 까다롭고 비용과 시간이 많이 소모되기 때문에 딥러닝 하드웨어 모듈의 소프트웨어 모델링을 통한 신속한 설계 검증 과정이 필요하다.

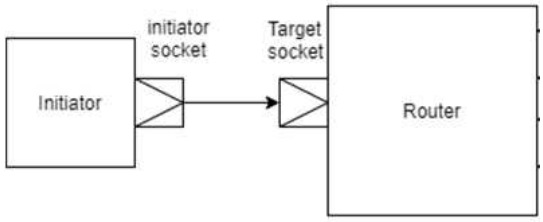
SystemC는 C++라이브러리 형태의 이벤트 기반 시뮬레이션 인터페이스로 ESL수준의 프로그래밍 언어이다[1]. SystemC를 이용하면 VHDL등을 이용하여 하드웨어 모델링을 하는 것보다 더 복잡한 수준의 구현을 쉽게 설계하는데 유리하고, ESL(Electronic System Level),와 TLM(Transaction Level Modeling)과 같은 시스템 레벨의 모델링 및 검증을 수행할 수 있다. 또한 하드웨어 RISC-V[2] 프로세서는 버클리 오픈소스 라이센스로 RISC 기반 프로세서로써 프로세서 명령어 세트가 단순하고 파워소모가 기존 프로세서보다 효율적이어서 최근 주목받고 있는 임베디드용 프로세서 이다. 본 논문에서는 최근 곽광받고 있는 임베디드 프로세서 코아인 RISC-V 프로세서 코어를 기반으로 SystemC를 이용하여 CNN 모듈을 설계하였다. 본논문에서 설계 및 구현한 SystemC 기반 CNN 시뮬레이터는 임베디드 디바이스에서 CNN 하드웨어 모듈을 검증하는데 효율적으로 사용될 수 있다.

2. RISC-V 기반 CNN

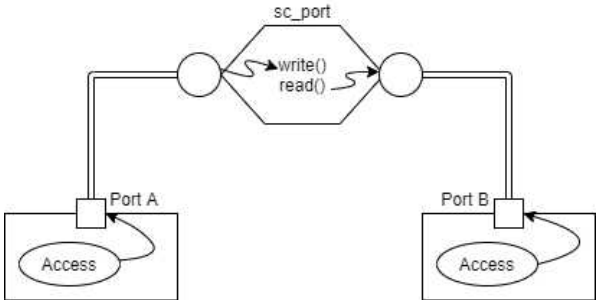
2.1 SystemC

SystemC 언어를 이용하면 VHDL등의 언어로 코딩을 하는것보다 복잡한 수준의 구현에 유리하고, ESL(Electronic System Level),와 TLM(Transaction Level Modeling)기반의 구현이 가능하다. SystemC

는 하드웨어 요소를 모듈로 구성할 수 있으며, 모듈은 서브모듈을 가질 수 있다. 모듈은 하나의 독립적인 프로세싱 단위가 도며, 모듈 내부에는 프로세스 또는 쓰레드를 가질 수 있다. SystemC는 하드웨어 모델링을 수행하기 때문에, SystemC의 시작과 동시에 각 모듈은 독립적으로 동시에 수행된다. 모듈간 통신은 Bus 또는 Channel을 통해서 데이터를 송수신할 수 있다. 그림 1은 모듈과 모듈 간의 Channel을 이용한 Socket 통신의 예를 도식화한 것이다.



(그림 1) SystemC Socket 통신

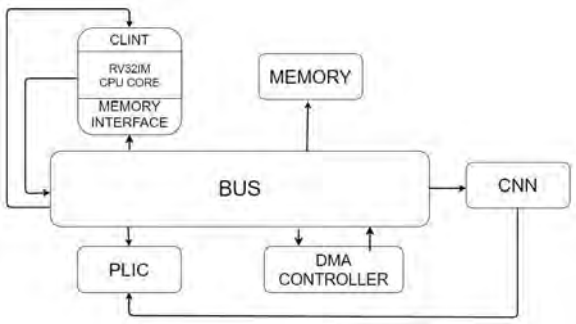


(그림 2) SystemC Port 통신

(그림2)은 channel을 이용한 방식은 직접 data를 넘겨주는 방식으로, SystemC에서는 같은 모듈 내 channel을 통해 내부 모듈끼리 데이터를 주고받는다. Router 모듈만 bus를 이용하여 데이터를 MEMORY로부터 가져오고, 그 외 다른 모듈은 모두 channel을 이용한 통신을 한다.

2.2 RISC-V VP

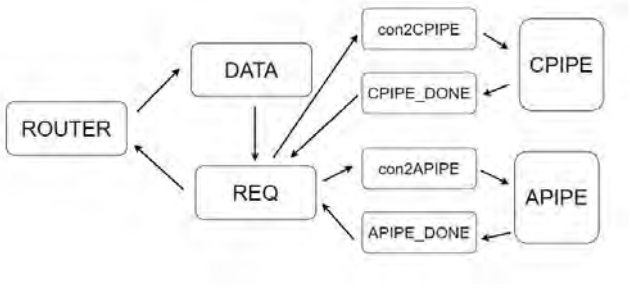
SystemC를 이용한 RISC-V 기반 CNN 시뮬레이터를 구현하기 위해서 기존에 SystemC로 구현된 RISC-V Virtual Platform(RISC-V VP)[3,4]에 CNN 모듈을 구현하였다. RISC-V VP는 RISC-V의 작동 방식을 Virtual Platform 형태로 구현한 SystemC기반 오픈소스 코드이며, 전체적인 구조는 그림 3에 나타난 바와 같으며, 각 부분은 SystemC 모듈로 구현되어 있다. 각 모듈의 기능은 다음과 같다. RV32IM CPU core 모듈은 RISCV-VP 32bit CPU core를 구현한 것이고, CLINT(Core Local Interrupt Controller)와 Memory Interface를 포함한다. PLIC(Platform Level Interrupt Controller)는 core이외의 모든 interrupt를 처리해주는 모듈이다. Memory 모듈은 DRAM과 같은 메모리를 모델링한 모듈이다. BUS 모듈은 SystemC 2 Version의 TLM(Transaction Level Modeling)을 기반으로 구현된 bus 통신 모듈이다. 모든 모듈은 BUS에 연결되며, CPU core의 Initiator Socket을 통해서 지정된 Address영역에 해당되는 Target Socket에 연결된 모듈과 통신을 수행한다.



(그림 3) RISC-V VP 모듈 구조도

2.3 CNN 모듈

우리는 이와 같이 기본 구성된 RISCV-VP 모듈에 CNN 모듈을 추가하여 CNN 딥러닝이 가능한 하드웨어 시뮬레이터를 구성하였다. SystemC 기반으로 설계한 CNN 모듈의 전체 구성도는 그림 4와 같다 .



(그림 4) CNN 모듈 구조도

모듈들에서 전체적으로 실행하는 흐름에는 다섯가지 종류가 있다. Data preprocessing : file에 padding등을 붙여 MEMORY에 연속되게 load 하는

작업. Data request : CONTROLLER에서 LOADER를 통해 MEMORY의 data를 요청하는 작업. Data response : CONTROLLER가 요청한 data를 MEMORY에서 LOADER를 통해 넘겨주는 작업. CPIPE : CONTROLLER의 data로 Convolution 연산을 해주는 작업. APIPE : CPIPE 연산 결과를 pooling, activation 해주는 작업.

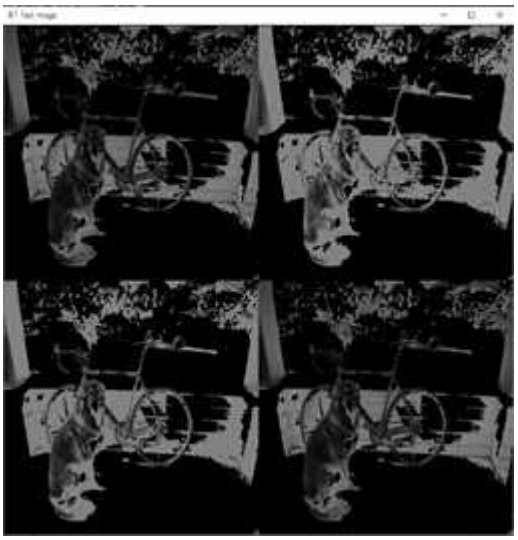
각 모듈의 기능과 동작은 다음과 같다. Router는 REQ로부터 데이터 요청이 들어오면 bus를 통해 MEMORY로부터 데이터를 읽고 쓰는 기능을 담당한다. DATA는 CNN연산에 필요한 input, weight, bias값을 b_address에 근거해 CNN_MODULE내부에서 공유하는 cache에 저장해주는 기능을 담당한다. REQ는 필요한 데이터의 정보, 데이터의 위치를 b_address와 m_address라는 변수에 저장하여 ROUTER로 데이터 요청을 하는 기능을 담당한다.

CPIPE, APIPE 연산 결과를 s_address 변수를 통해 MEMORY로 저장 요청 한다. Con2CPIPE는 REQ 모듈로부터 신호를 받아 실행되며, CPIPE가 연산을 한번 진행하기 위해 필요한 데이터를 받아 CPIPE 모듈에게 전송하는 모듈이다. CPIPE는 Con2CPIPE로부터 인계받은 데이터로 convolution 연산을 진행하는 모듈. 연산 종료 후 결과를 CPIPE_DONE에 전송한다. CPIPE_DONE은 CPIPE로부터 받은 결과를 내부공유 cache에 저장한다. cache 용량이 초과될 경우 REQ를 통해 MEMORY로 저장을 요청한다.

Con2APIPE는 REQ 모듈로부터 실행 신호를 받아 APIPE가 연산을 한번 진행하기 위해 필요한 데이터를 받아 APIPE 모듈에게 전송하는 모듈이다. APIPE는 Con2APIPE로부터 받은 데이터로 activation과 pooling 연산을 진행하는 모듈. 연산 종료후 결과를 APIPE_DONE에 전송한다. APIPE_DONE은 APIPE로부터 받은 결과를 내부 cache에 저장한다/ cache 용량이 초과 될 경우 REQ를 통해 MEMORY로 저장을 요청한다.

딥러닝에 흔히 사용되는 Dog 사진을 시뮬레이터로 연산을 하였다. 그림 5는 본 논문의 CNN 모듈 연산기 중에서 CPIPE 모듈을 이용한 Convolution 연산을 수행한 상태의 이미지들이다. 그림에서 보는 바와 같이 정상적인 컨벌루션이 수행되는 것을 확인할 수 있으며, Weight값이 커짐에 따라 CPIPE의 output이미지 변화가 심하게 이루어지는 것을 확인할 수 있다. 그림 6은 CNN 모듈 중 Activation 및 Pooling 연산을 수행하는 APIPE 모듈에서 Leaky Relu 방식의 Activation을 진행하게 될 경우의 결과를 나타낸 것이다. 그림에서 보는 바오 같이 원본이미지의 색상반전이 일어나 사진이 어두워지는 것을 확인할 수 있다.

(그림 5) CPIPE OUTPUT



(그림 6) APIPE OUTPUT

3. 결론

엣지 컴퓨팅과 같은 IoT 및 모바일 컴퓨팅 시스템 및 영상처리 시스템에서는 단말기 자체에서의 딥러닝 연산을 수행하기 위해서 하드웨어 딥러닝 연산기를 설계 및 구현하고 있다. 하지만 하드웨어 설계 및 구현은 검증이 까다롭고 비용과 시간이 많이 소

모되기 때문에 딥러닝 하드웨어 모듈의 소프트웨어 모델링을 통한 신속한 설계 검증 과정이 필요하다.

본 논문에서는 SystemC를 이용한 RISC-V VP 기반의 CNN 모듈을 설계하고 구현하였다. 설계 및 구현한 CNN 모듈은 SystemC를 기반으로 한 ESL 모델링 레벨이며, RISC-V Core와 연동 하여 RISC-V 기반의 임베디드 프로세서를 위한 CNN 모듈 검증이 효율적으로 활용될 수 있다.

Acknowledgments

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학지원사업의 연구결과로 수행되었음 (2019-0-01816)

4. 참고문헌

[1] D.C. Black, J. Donovan, B. Bunton, and A. Keist, “SystemC : From The Ground up”, Eklectic Ally, Inc.

[2] A. Waterman, Y.S. Lee, and R. Avizienis, D. Patterson, and K. Asanovi´c ”The RISC-V Instruction Set Manual“, Volume II: Privileged Architecture, July, 2016, https://people.eecs.berkeley.edu/~krste/papers/riscv-privileged-v1.9.pdf

[3] Github, RISC-V Virtual Prototype, https://github.com/agra-uni-bremen/riscv-vp

[4] V. Herdt, D. Groβe, H. M. Le and R. Drechsler, "Extensible and Configurable RISC-V Based Virtual Prototype," 2018 Forum on Specification & Design Languages (FDL), Garching, pp. 5-16, 2018.

[5] NVIDA, NVIDIA Deep Learning Accelerator, [Online]. Available: https://nvdla.org

[6] OSCI, Open SystemC Initiative (OSCI), TLM-2.0 User Manual, Jun. 2008.

[7] F. Ghenassia, “Transation-Level Modeling with SystemC : TLM Concepts and Applications for Embedded Systems,” Springer, Nov. 2005.