EE-552 Class Project - Network on Chip Spiking Neural Network implementation

Rev. 1.0 (subject to update)
last updated: March 13, 2023

Spring 2023

1. Introduction

Motivated by industrial large-scale spiking neural networks TrueNorth and Loihi, we propose as a class project to model and implement a machine learning accelerator using an array of processing elements described using System Verilog. Unlike TrueNorth and Loihi, however, we will target more traditional machine learning algorithms (e.g., convolutional neural networks (CNN)) and include a discussion of how CNNs have been mapped to asynchronous Spiking Neural Networks (SNNs). This will focus on the application of this technology in a domain that is quite relevant to today's chip designers.

The class project will involve modeling the behavior of the neurons and associated network on chip (NoC) as well as implementing components in an asynchronous design style. The project will thus include aspects of architecture, micro-architecture, circuit design, modeling and verification. We will be using a collection of industrial language and tools (SystemVerilog).

The class project will be broken into several phases. Students will self-organize into teams of 1,2 or 3 students and grades will be given to each team (special permission from Prof. Beerel needed for teams of 4). It will include a required report (at least 10 pages) and class presentation that will be graded for both content and clarity. All students are expected to participate in the class presentation which is expected to be 8 minutes long. If needed due to time constraints some presentations will be held outside of normal class times.

2. **Project Stages**

Stage 1: Team formation and project scope description. The project proposal needs to contain a description of the project design as well as several opportunities for enhancements. Teams will self-form and describe their intended design. Teams can be 1, 2 or 3 persons. Teams of the smaller size will be expected to do a baseline design. Teams of larger sizes will be expected to add additional features and enhancements to the design.

a. After building your team, you need to submit a single-page proposal with details of your design. (e.g. NoC details (topology, routing scheme, enhancements), SNN details (baseline enhancements).

b. Task partitioning between members of the group.

c. Specify which blocks (baseline: at least one) of your design will be implemented to gate-level, implementing a template covered in class.

d. We will review and approve your proposal, providing feedback.

e. Submit your team proposal using DEN's website, including the list of group members.

f. Due on Tuesday, March 21, 2023, 11:59pm.

g. Filling your team information in the shared google sheets: https://docs.google.com/spreadsheets/d/12kGqqznhQI99iOceG4s9fzglDMTg9KkpdNlwxEria-U/edit#gid=0

Note: Your proposal should be about the right amount of work proportional to your group size to get approved. Worth 10% of the project grade.

Stage 2: Project progress report, micro-architecture specification and design. Teams are expected to submit a project report with detailed description of the architecture and microarchitecture design. This will provide an opportunity for the teaching team to give feedback to the team individually. (This is due on April 4, 2023, 11:59pm – not a joke). The progress report should include the following sections:

a. Micro-Architectural Design. Design each micro-architectural block using SystemVerilogCSP (SVCSP).

b. Hand-in SystemVerilogCSP code for each block in your design, including the selected gate-level block in SystemVerilog (SV). Give a brief explanation on functionality.

Note: Code files can be attached inside a folder in your report submission.

Note: An example code of connecting the gate-level block with SystemVerilogCSP blocks can be found in DEN -> Assignments -> Project - Stage 2 -> *shim.sv*.

c. Hand-in simulations (transcript files for SVCSP, waveforms for SV) showing correctness of operation for each block (as much as you have completed, we expect to have about 40%-50% of the whole design). Identify test cases you used to make sure each block is functioning correctly. Maybe, you will need to provide multiple testbenches to accomplish this.

d. Micro-architectural design and testbench of the gate-level component, using SystemVerilog.

e. Gate-level block simulations and waveforms.

f. Hand-in as many as required top-level and block level simulations (transcripts), showing correctness of results, along with a brief explanation.

g. It is okay to make substantial design changes as you see fit; this should be part of your progress report (issue motivating design change and proposed resolution)

h. Submit using DEN's website, including required documents and files in a zip file

Stage 3: Class Presentation & demo. Each team will give an 8-10 minute presentation in the last week of classes, typically no more than 12 slides. Each team member should present/talk about the design. Teams of larger size (3 students) can take an additional 2 minutes.

a. Power-point presentation overview should accompany your design. The presentation should review the architecture, micro-architecture, gate-level design and results.

b. The demo should include running a simulation along with comprehensive explanations.

c. The provided time needs to be distributed between the demo and the power-point presentation.

d. Submit your presentation the day previous the presentation using DEN's website.

Presentation on week 15 (April 25 – April 28) – during class/discussion time.

Stage 4: Final report submission. The final report is due on May 2, 2023. It should incorporate any last-minute design fixes and feedback from the class presentation.

Document results: The document should be the culmination, combination, refinement and feedback, along with the results and summaries.

Folder submission: Submit all your design code, including testbenches, transcripts, waveforms, a readme file.

Submit using DEN's website, including required documents and files in a zip file.

3. Grading

The grading of the class project is defined by the following rubric:

- Stage 1 Project scope (10 points)
- Stage 2 Progress Report & code submission (30 points)
- Stage 3 Project Presentation & demo (30 points)
    - Project quality (15) – as defined below and conveyed by your presentation & demo.
        - Tasks completed (e.g., architecture design and description, SystemVerilog design, gate-level design, etc.)
        - Design correctness verification – simulation showing correctness of results.
        - Design analysis – detailed study of throughput, latency, TOPs, bottlenecks, design & dataflow comparisons, etc.
        - Changes in plan from proposal and the rationale - this should include things that did not work out.
        - Novelty & significance: what did you add to the resources that you started with? How challenging was your project objective? (Team size will be considered)
    - Presentation quality (15)
        - Slide quality
        - Presentation organization

        - Presentation delivery
        - Q&A session
        - NOTE: One score for the entire group (typically), all members must present (talk and explain), otherwise the grade will be low.
- Stage 4 Project Final Report (30 points)
    - Project Quality (20) – as defined above and conveyed by your report
    - Report Quality (10)
        - Clearly stating project objectives
        - Clearly describing proposed architecture and micro-architecture
        - Clear description of test cases
        - Clear description of analysis of design (performance, latency, bottlenecks, tradeoffs, others ...)
        - Clearly stating accomplishments (see quality above)
        - Clearly describing future work (if you were to continue this or if another team picked up from where you left off)
        - Proper references (included any figures you did not create)

4. Project Details

The baseline SoC project is described in this section. It is encouraged to provide enhancements and improve the design in different aspects related to the SNN implementation. e.g., performance, reduced complexity, scalability, etc...

4.1 Model Architecture

The project is explained in two parts: The NoC and the SNN convolution implementation. For testing, a baseline testbench (provided to students) will provide control signals and check correctness of operation **on a 5x5 filter and 25x25, input feature map (ifmap) matrices with stride=1, padding=0.**

4.2 Part 1: Network on Chip (NoC)

This is composed of routers and links. The key idea of using a NoC is to allow communication among multiple network clients (Nodes) in a chip.
Instead of connecting top-level modules by routing dedicated wires, they are connected to a network that routes packets between them. Sharing the wiring resources between many communication flows makes more efficient use of wires: when one client is idle, other clients continue to make use of the network resources. Some important properties of a NoC are:

• NoC topology: NoC comprised of routers interconnected by point-to-point links. Network topology can vary depending on system needs, module sizes and placements. Some common topologies include tree, mesh and ring.

• NoC routing techniques: Routing techniques can be classified into mainly two types, deterministic and adaptive. In deterministic routing the path from source to destination is fixed while adaptive routing allows the path to change depending upon the current state of load on the network.

• NoC switching techniques: Switching techniques can be classified into mainly two types, circuit switching and packet switching. In circuit switching the entire path is set up and reserved before an actual data is sent, while in packet switching data is forwarded on per hop basis. Each packet contains data as well as control information.

• NoC flow control: Flow control mainly addresses the issue of correct operation of the network. It determines how the networks' resources, such as channel bandwidth and buffer capacity, are allocated to packets traversing the network, ensuring that the network does not accept more packets than it can transmit successfully.

4.3 Design of NoC

Each team has freedom to choose the NoC topology, routing techniques, switching techniques and the flow control.

Basic NoC Requirements

- Number of PEs: at least 9

- Number of MEMs: at least 2 (1 for ifmap, 1 for filter)
- Topology used: Any
- Routing: Any
- Switching technique: Any
- Packet size: Each team fixes the required packet size.
- Router: FL=2ns, BL=1ns.

Note: The FL and BL values listed are to be implemented in each router's sub-modules.

Other considerations you may be interested

- Design of PEs: design different types of PE with various functions. For example, PE for convolution, PE for pooling, PE for partial summation, etc.
- NoC topology: use multiple NoCs. For example, one NoC to transmit ifmap spikes and the other to transmit filter values. If you use multiple NoCs, each NoC may have different packet sizes.
- Design of MEM: 1 for ifmap, 1 for filter, 1 for output.
- Packet size: increase the packet size to transfer more than one input in ifmap or more than 1 weight in filter.

4.4 SNN Implementation

For detailed CNN/SNN operation, please refer to lecture explanation (week 6). On top of each node's functionality, described below, the node needs to packetize and depacketize data transfers between nodes. See Figure 2.

Packetization refers to wrapping" the data sent out. This implies creating a packet header that contains the sender and destination addresses. Then the header is concatenated to the data. The sender, destination and data are packet fields. The packet is forwarded to the router that connects to the node. See Figure 1 for a packet field example (16-bits). Additionally, the packetizer reports (transcript) the sender, destination and data.

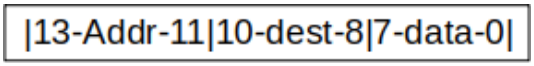


Figure 1: NoC packet example

Depacketization is the opposite to packetization. Unwrap the packet and report (transcript for debugging) after that operation, provide out the data

to the functional block. Additionally, reports (transcript) the sender, destination, and data on the received packet. See discussion week 9 for details.

Functional block is composed of a Finite State Machine (FSM) and sub-modules that implements the required node functionality. The SNN PE is comprised by 3 basic modules:

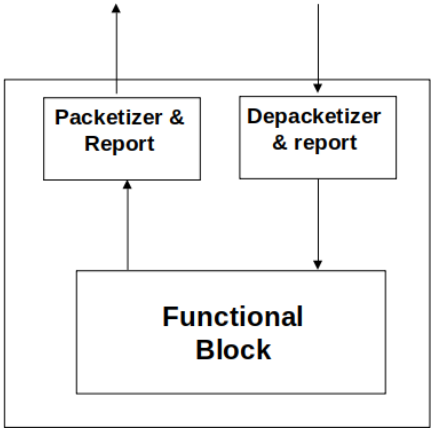


Figure 2: Generic node block

Memory (M) This module is responsible for loading the filter or ifmap arrays, providing to the filters (constant) or input spike array (ifmap) at each timestep. A memory will also receive the output spike matrices at each timestep.

Memory operation example:

- Upon a request (if_spike_map, of_spike_map, filter, or psum), memory will store or send out the requested data.
- The memory will respond to requests with a fixed latency, and can store additional temporary values as necessary
- Receives the output spike results at the end of each timestep.

You are allowed to design different memory operations if necessary.

Processing element (PE). Note that the packet arrives to the depacketizer unit, and the data (8-bits for filters, 13 bits for partial sum) is sent to the functional unit that implements the Processing element.

Processing element operation example:

- The Processing element contains 2 register files (RF), one to store the filter weights, the other to store the ifmap spikes to be multiplied. Note that the accumulator and comparator can also store a value inside.
- Perform conditional accumulation of filter weights (based on the binary IF map).
- When finished with a spike and residue output, the packetizer creates the packets with the corresponding destinations.

You are allowed to design PE with other operations if necessary.

Sum unit (S). This module is responsible for summing partial results from PEs, subtract the threshold, and sending the result to the Memory unit.

Sum unit operation example:

- The packet arrives via NoC input interface to the depacketizer unit, then it is forwarded to the functional sum block.
- If the sum block receives a partial sum (psum) from each Processing element (P1, P2, …). As soon as it receives all operands, it proceeds with the summation.
- Subtract the threshold from the accumulation results. Generate the output spikes and the residue output.
- The result is forwarded to the packetizer unit, that creates the packet with the destination (Memory) and send it to the NoC output interface.

You are allowed to design partial SUM node and use multiple SUM nodes.
You are allowed to design a separate Threshold node for subtracting the threshold instead of performing this operation in the SUM nodes.

2D convolution operation. For details, please check lecture week 6.

SNN component requirements

The FL and BL values listed are to be implemented in each node's sub-module.

- Memory (M): FL=12ns, Bl=4ns. Note the time penalty accessing the memory!

- Processing element internal components: FL=2ns, BL=2ns. Memory limit: each PE can hold no more than 32 filter values (8b each), 32 input spikes (1b each) and 32 partial sum (13b).
- Sum (s): FL=2ns, BL=2ns.
- Packetizer/depacketizer: FL=1ns, BL=1ns. Same values for all nodes.

A testbench providing input feature map (25x25), filter (5x5) and collecting the output (21x21) for 2 timesteps will be provided. The details of the testbench will be released later.

Some suggested improvements/enhancements related to CNN implementation are:

- (Medium) Expanding size of Input Feature Map to support not only 25x25 but also other sizes (e.g., 32x32).
- (Medium) Considering support not only a 5x5 filter but also a 3x3 filter or a 7x7 filter.
- (Hard) Instead of supporting just one filter, support many filters.

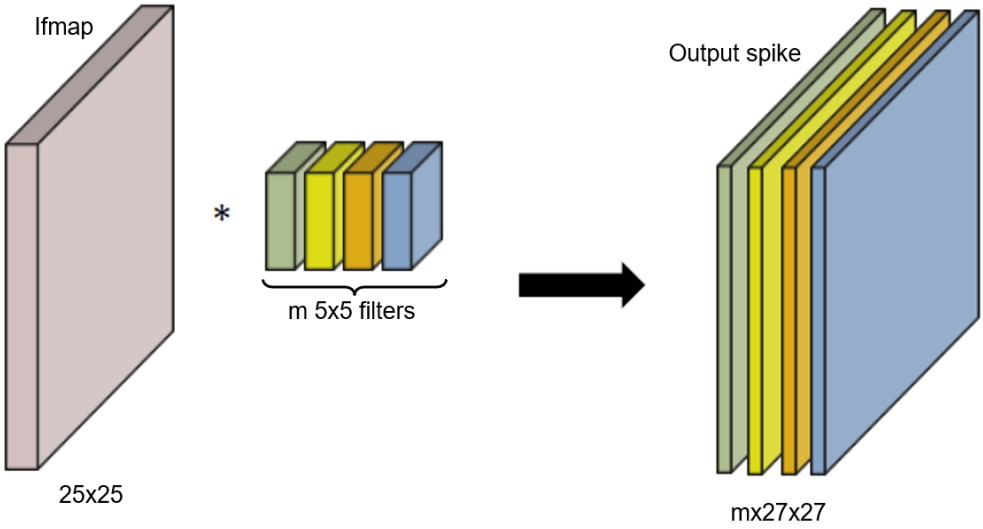


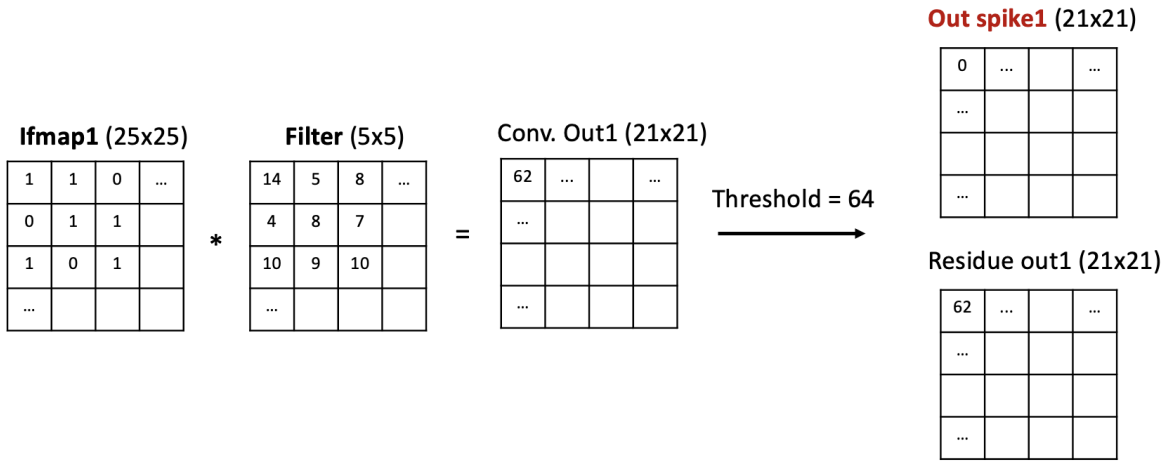
Illustration of multi-channel filters

- (Hard) Support multiple layers. In this case, you may need PE with a pooling function to connect the layers. We will clarify how to handle multiple layers in the upcoming discussion section.

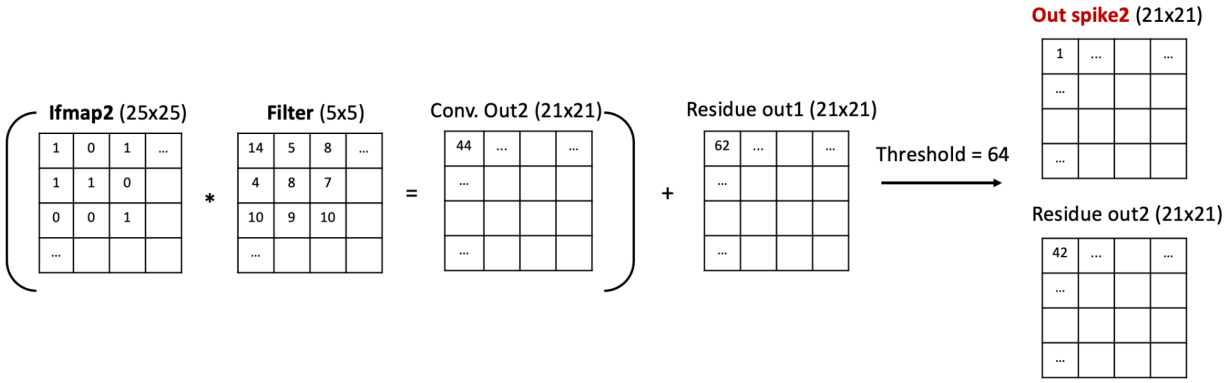
More details on the project

Function of SoC-SNN

a. 1-layer operation with 2 timesteps (Baseline):
   Timestep 1:

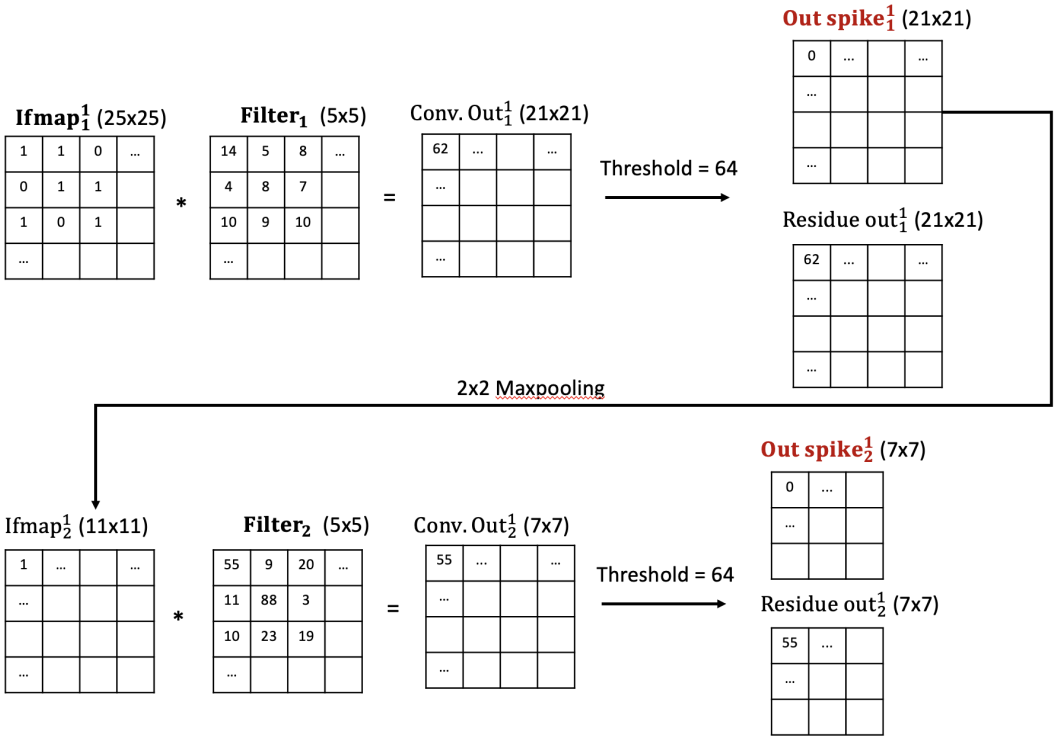


   Timestep 2:

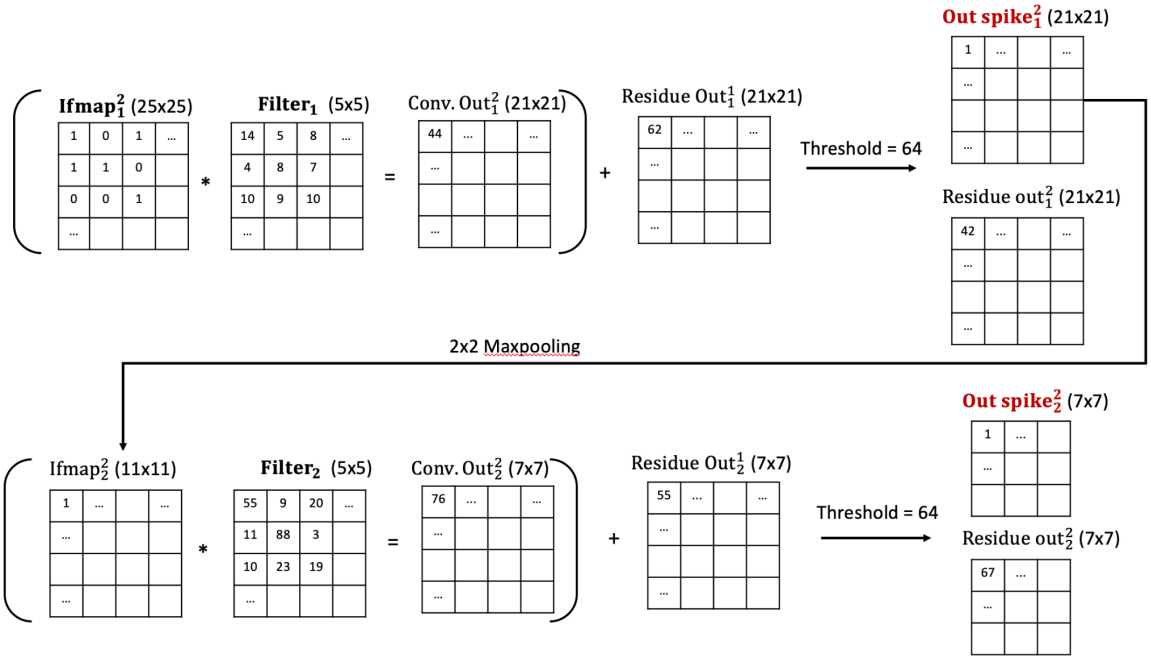


   The provided testbench will provide values in Ifmap1, Ifmap2, Filter, and collect Out spike1 and Out spike2. The testbench will also check whether the collected results are correct.

b. 2-layer operation with 2 timesteps (Enhancement):
   The superscript is the timestep index, and the subscript is the layer index.
   Timestep 1:

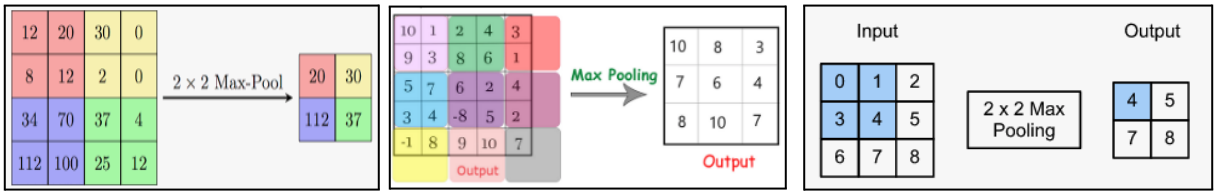


Timestep 2:



The provided testbench will provide the values of the first layer's ifmap at both timesteps and both layers' filter values, and collect the out spike of each layer at each timestep.

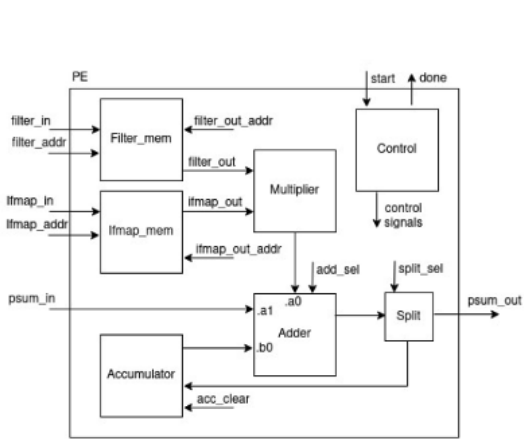
2x2 Max Pooling examples:



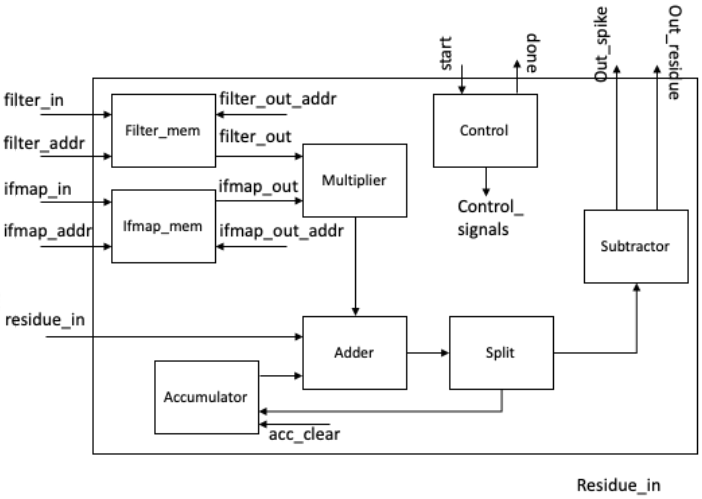
Example 1 Example 2 Example 3

Function of PE

a. If your PE node only supports the convolution operation, you need another node to operate the summation with the previous residue out, subtract the threshold and output current out spike and the new residue out. You can also extend the function of your PE by adding the summation and subtraction operations. An example is shown below.



PE for convolution

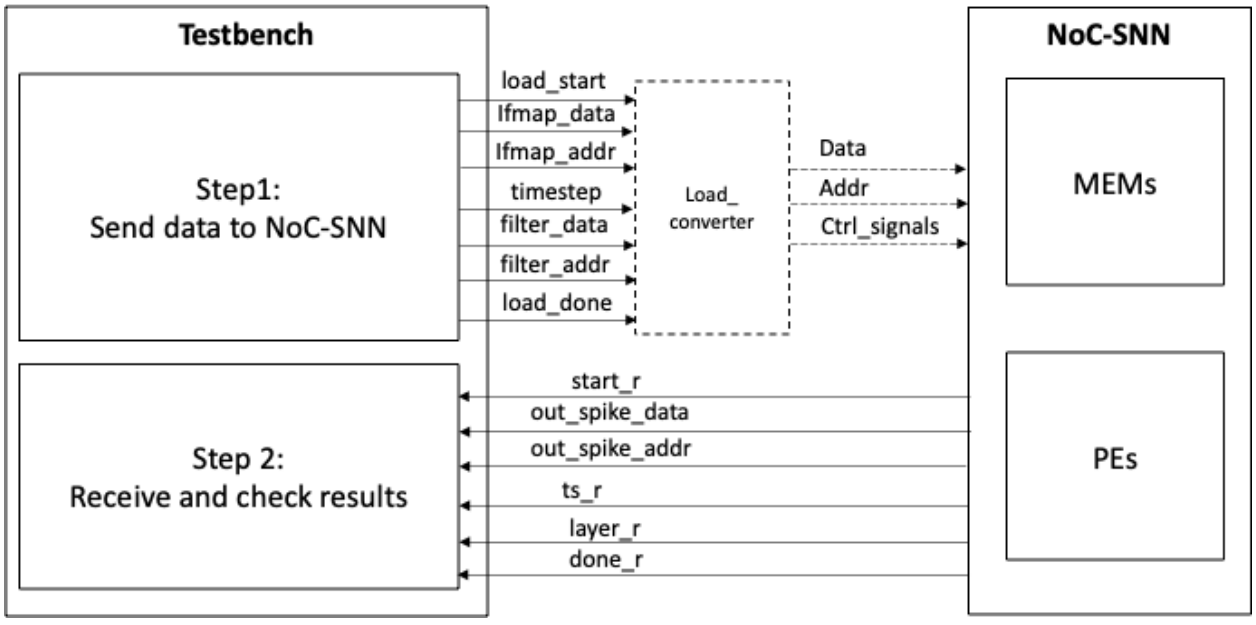


PE for convolution, adding residue, and subtracting threshold

TestBench

Provided files:

1. **testbench.sv** to test the noc_snn.
2. a **noc_snn.sv** simulating the behavior of the noc_snn. You could try to run the testbench with the provided noc_snn.sv as dut to understand the testing flow. After finishing your own design, replace the provided noc_snn.sv with yours.
3. **The input files and the golden results**. The values of the filter are in filter.txt. ifmap1.txt and ifmap2.txt provide the input feature map at timestep 1 and timestep 2, respectively. out_spike1.txt and out_spike2.txt provide the golden spike results at timestep 1 and timestep 2, respectively. out_residue1.txt and out_residue2.txt record the output residue at timestep 1 and timestep 2, respectively. Note that out_residue1.txt and out_residue2.txt are only for you to check your intermediate results, and the testbench does not check them.
4. **snn.py** generates the input files and the golden results. This helps to provide more test patterns if you need them.



Overview of the testing

Testing flow:

1. The testbench sends token 1 via ‘load_start’, indicating the start of loading filter values and ifmaps. The testbench will send the filter first and then ifmap at timestep 1 and ifmap at timestep 2.
2. After sending the filter and ifmaps, the testbench sends token 1 via ‘load_done’ indicating that all inputs have been sent.
3. The testbench waits for the token 1 from ‘start_r’ to receive the outputs.
4. After receiving token 1 from ‘start_r’, the testbench begins to receive the outputs from noc_snn and compare the results with the golden spikes. During this step, the token from ‘ts_r’ indicates the timestep of the current output spike, and the token from ‘layer_r’ indicates the layer of the current output spike.
5. Once receiving all the results, the testbench waits for token 1 from ‘done_r’ to finish the testing.

Checking results:

The auto-generated ‘test.dump’ and ‘transcript.dump’ record the checking results.

Notes:

- You could design a load_converter to customize the control signals and memory address if needed.
- You could modify the testbench if needed.