NETWORK ON CHIP CONVOLUTIONAL NEURAL NETWORK MICRO-ARCHITECTURE SPECIFICATION

Ahmed Abuhjar
abuhjar@usc.edu

Matthew Conn
connmatt@usc.edu

EE552 - Asynchronous VLSI Design
April 18th, 2021

Viterbi School of Engineering
University of Southern California

Revision History

Revision	Date	Author(s)	Description
1.0	04/18/21	Ahmed Abuhjar & Matthew Conn	Initial version

Contents

1. Introduction

This document describes the Microarchitecture of the core components and building blocks of the Network on Chip (NoC) and Convolutional Neural Network (CNN). Selected of the outputs of each sub-module are included in this report.

The Machine Learning Accelerator in this project is modeled in an asynchronous design style using System-Verilog. The protocol used for communication is a bundled-data protocol, with 4-phase handshaking. The design is composed of an array of Processing Elements (PEs), a NoC, and other Arithmetic/Storage units. Each component will have its own test bench provided for simulation and verification. Some of the enhancements implemented in this project are to improve the latency. This includes an increase in the local storage of the processing elements to accommodate an entire row of pixels. A row-stationary design is also adopted in this project, which will benefit from local PE-PE links.

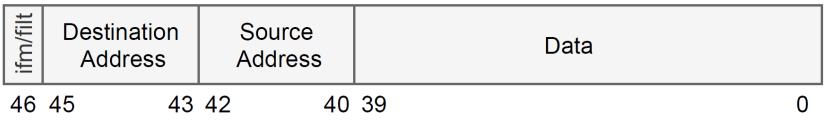
This document covers the following topics:

- High-level architecture of some of core components
- Description of the functionality for each component
- Gate level module(s)
- Output files, transcripts and waveforms

2. Packet Structure

DESCRIPTION:

Each packet to be routed through the network on chip is of fixed length of 47 bits. Given such a small packet size, the network can transmit the entire packet from one node to another without breaking it into multiple flits. The packet format is as described in the figure below.



PACKET FORMAT:

- [46] ifm/filt: This bit indicates to which memory (filter or pixel) in the PE the data should be loaded.
    - The packetizer of the sending memory will decide depending on the received data! (controlled merge).
    - The depacketizer of the receiving PE will decide depending on this bit field.
    - The packetizer of the sending PE will decide depending on the received data! (controlled merge)
- [45:43] Destination Address: 3 bit address of a node to which data should be sent.
- [42:40] Source Address: 3 bit address of a node from which data is sent
- [39:0] Data:
    - 40-bit data sent from Mem to PE/ PE to Adder/ Adder to Mem.
    - 24-bit filter sent from PE to another PE.

3. Switch Module

DESCRIPTION:

Each router will have three parametrized switches. Each switch will be interfacing with one input channel of a 2-way 3-input router. Depending on whether the input channel is a parent or a child, the switches will have two different routing processes. The received packet will be routed to the appropriate output channel using the mask of the router and the destination address. The two routing processes of the switch are summarized below:

- Switch with a parent input channel (1 per router):
  - If the first unmasked bit of destination address is 0, forward packet to child-1 output channel.
  - If the first unmasked bit of destination address is 1, forward the packet to child-2 output channel.
- Switch with a child input channel (2 per router):
  - If destination address == router address, send forward data to the parent output channel.
  - Otherwise, forward sata to the other child output channel.

OUTPUT DUMP FILE:

(1) Packet with destination 011 sent on parent input
(mask=000 address=000) and received by child 1.
received : 10111001111111111111111111111111111111111111111

(2) Packet with destination 011 sent on parent input
(mask=110 address=100) and received by child 2.
received : 10111001111111111111111111111111111111111111111

(3) Packet with destination 011 sent on child input
(mask=110 address=100) and received by other child.
received : 10111001111111111111111111111111111111111111111

4. Merge Module

DESCRIPTION:

Each router will have three merge modules. Each merge will be interfacing with one output channel of a 2-way 3-input router. It will first receive a select signal from the arbiter, which will then determine which input channel should be forwarded to the output channel of the merge.

OUTPUT DUMP FILE:

Select = 0 : A Selected

Select = 1 : B Selected

A = 36, B = 1, Select = 1, Output = 1

A = 35, B = 13, Select = 1, Output = 13

A = 37, B = 18, Select = 1, Output = 18

A = 13, B = 54, Select = 1, Output = 54

A = 45, B = 12, Select = 1, Output = 12

A = 6, B = 5, Select = 0, Output = 6

A = 37, B = 55, Select = 0, Output = 37

A = 15, B = 50, Select = 0, Output = 15

A = 40, B = 5, Select = 0, Output = 40

A = 61, B = 45, Select = 1, Output = 45

A = 35, B = 10, Select = 0, Output = 35

A = 32, B = 42, Select = 1, Output = 42

A = 22, B = 19, Select = 1, Output = 19

A = 19, B = 43, Select = 1, Output = 43

A = 2, B = 46, Select = 1, Output = 46

A = 15, B = 35, Select = 0, Output = 15

A = 10, B = 60, Select = 0, Output = 10

A = 10, B = 1, Select = 0, Output = 10

A = 56, B = 9, Select = 1, Output = 9

A = 54, B = 6, Select = 0, Output = 54

5. Arbiter Module

DESCRIPTION:

The arbiter module will be granting access to one of the two input channels requesting access to one of the merges. If two channels are requesting access at the same time, the arbiter will randomly select the winner, and output the Select signal to be received by the merge.

OUTPUT DUMP FILE:

sel = 0 : Req_1 gets access

sel = 1 : Req_2 gets access

TEST Req_1 ONLY

Req_1 gets access: sel = 0

TEST Req_2 ONLY

Req_2 gets access: sel = 1

TEST BOTH Req_1 & Req_2

Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 2 gets access: sel = 1
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 1 gets access: sel = 0
Req 2 gets access: sel = 1
Req 1 gets access: sel = 0
Req 2 gets access: sel = 1
Req 1 gets access: sel = 0

6. Arbitrated Merge Module

DESCRIPTION:

The arbitrated merge module will be composed of the arbiter and the merge modules. Its functionality will be an amalgam of both functionalities of the arbiter and the merge. This is in contrast to a fair merge or other type which would guarantee all input channels have an opportunity to use the router. During the final phase of the project, we will determine whether a fair merge is necessary to prevent deadlock.

OUTPUT DUMP FILE:

SELECT A- INPUT CHANNEL

A = 36, B = 1, Output = 36

SELECT B- INPUT CHANNEL

A = 9, B = 35, Output = 35

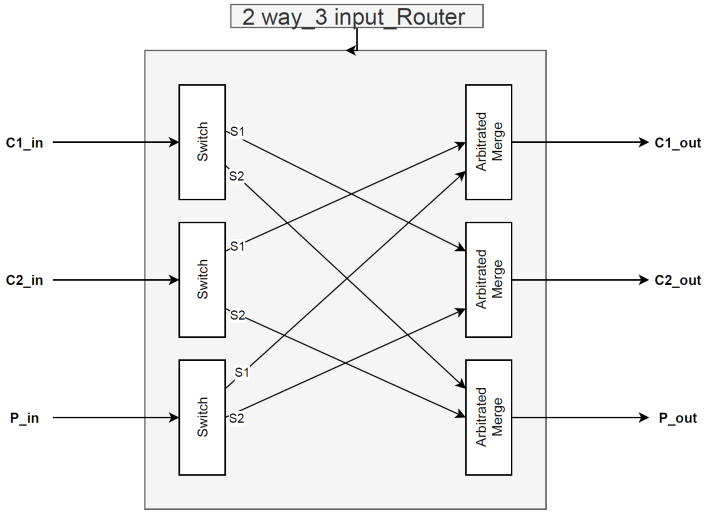
SELECT RANDOM INPUT CHANNEL

A = 13, B = 13, First Output = 13, Second Output = 13
A = 37, B = 18, First Output = 37, Second Output = 18
A = 1, B = 13, First Output = 1, Second Output = 13
A = 54, B = 61, First Output = 54, Second Output = 61
A = 45, B = 12, First Output = 45, Second Output = 12
A = 57, B = 6, First Output = 57, Second Output = 6
A = 5, B = 42, First Output = 5, Second Output = 42
A = 37, B = 55, First Output = 37, Second Output = 55
A = 18, B = 15, First Output = 15, Second Output = 18
A = 50, B = 14, First Output = 50, Second Output = 14
A = 40, B = 5, First Output = 40, Second Output = 5
A = 28, B = 61, First Output = 28, Second Output = 61
A = 45, B = 37, First Output = 37, Second Output = 45
A = 35, B = 10, First Output = 35, Second Output = 10
A = 0, B = 32, First Output = 32, Second Output = 0
A = 42, B = 29, First Output = 42, Second Output = 29

7. Router Module

DESCRIPTION:

The router module is composed of three switch modules and three arbitrated merges. Given the routing paradigm described in the Switch Module section, and the arbitration logic described in the Arbiter Module section, a representation of the internal components of the 2-way 3-input router can be determined as described in the figure below. The router's overall latency is: FL = 2, BL = 1.



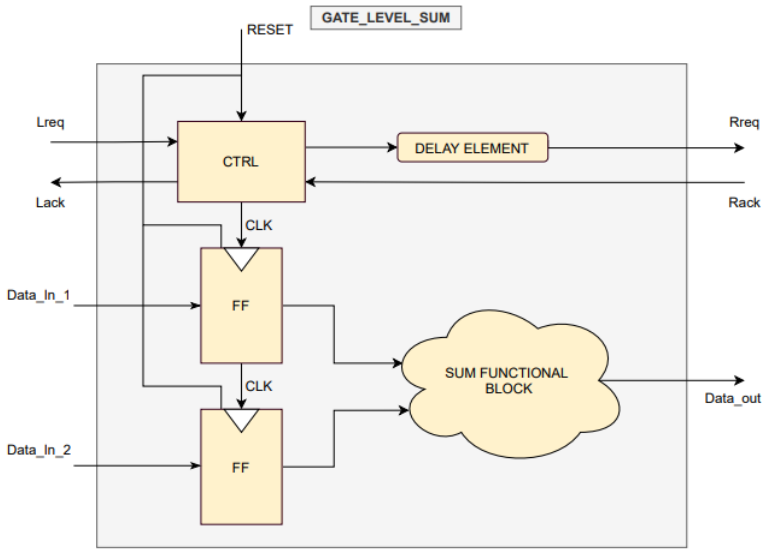
OUTPUT DUMP FILE:

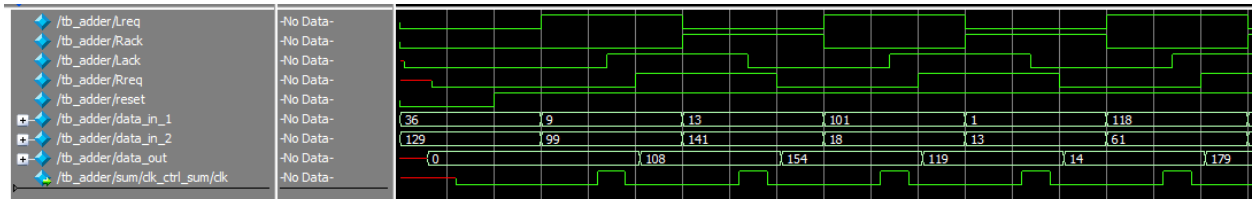
PACKET FROM CHILD-1 TO PARENT
mask = 110, address = 100, dest_addr = 100
Successful Packet From C1 to P
PACKET FROM CHILD-2 TO PARENT
mask = 110, address = 100, dest_addr = 100
Successful Packet From C2 to P
PACKET FROM CHILD-1 TO CHILD-2
mask = 110, address = 100, dest_addr = 001
Successful Packet From C1 to C2
PACKET FROM CHILD-2 TO CHILD-1
mask = 110, address = 100, dest_addr = 010
Successful Packet From C2 to C1
PACKET FROM PARENT TO CHILD-2
mask = 110, address = 100, dest_addr = 001
Successful Packet From P to C2

8. Gate Level

DESCRIPTION:

A transistor/gate level implementation of the adder is provided. The adder is built using a bundled data micropipeline design, featuring a control block and delay line. The delay line is implemented using simple #<delay period> statements in Verilog, which is akin to a simple inverter-based design. A more complex design would use some NAND gates to speed up the reset time. The control block generates a clock signal to feed two flip flops (DFFs), which hold data waiting to be processed. On each positive clock edge, the flip flops pass new data to the combinational (addition) block, and the outputs are stable until the next rising clock edge. The delay line ensures that the R_ACK signal (going to the receiver block) does not arrive until after the sum output has finished calculating. The gate level implementation also features a reset control, which connects directly to the control block and DFFs.

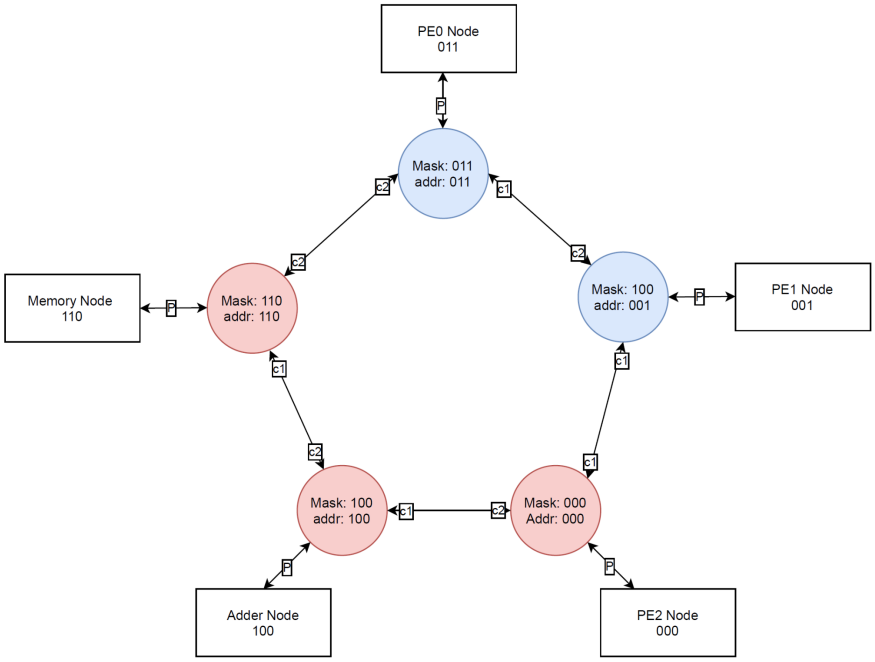




9. NoC

DESCRIPTION:

The NoC module is a top-level module, and it is composed of five routers and interfacing channels that form a Ring topology as described in figure below. Each router will have a different mask and address associated with it. Each router in the network will have three input channels and three output channels. The packet injected into the network will be routed either clockwise or counterclockwise depending on the shortest path to its destination.

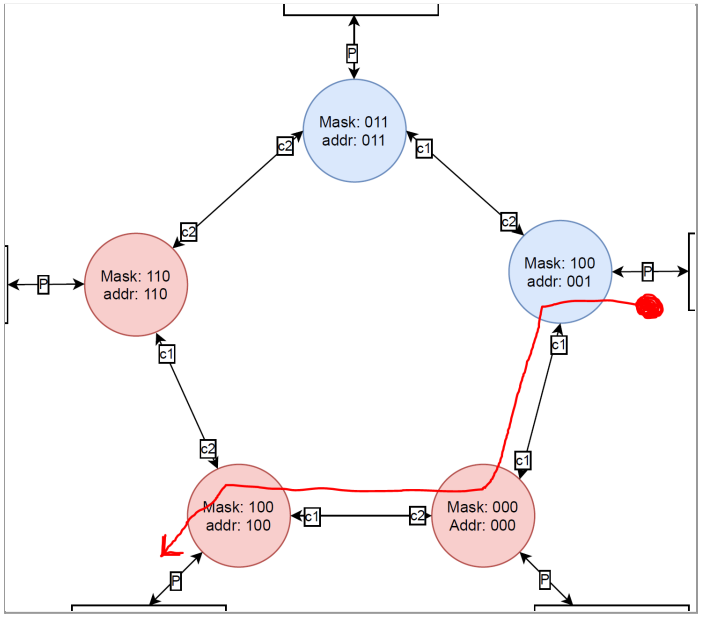


OUTPUT TRANSCRIPT:

Two different tests for the NoC are illustrated below

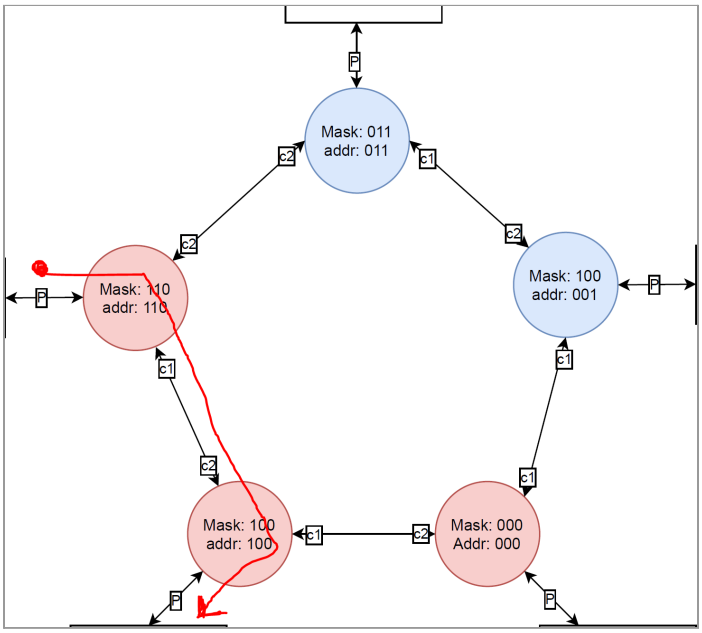
Test #97: The packet is injected from PE1 node and its destination is the Adder node. The NoC will then choose the shortest path to route the packet as depicted in the diagram below.

```
# ------------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------------
# ****************************** TB TEST #97 @ Time: 5380ns ******************************
# Source Node: 001, Destination Node: 100, Data: 00000000000000000000000000001011011101,
# ------------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------------
#
# Router Instance: tb_noc.noc_tester.PE1_router.switch_p_in
# Router Address: 001, Mask: 100, Packet Destination: 100
# Received packet from a parent node
# Sent = 11000010000000000000000000000000000001011011101 to child-1
# ------------------------------------------------------------------------------------
#
# Router Instance: tb_noc.noc_tester.PE2_router.switch_c1_in
# Router Address: 000, Mask: 000, Packet Destination: 100
# Received packet from a child node
# Sent = 11000010000000000000000000000000000001011011101 to other child
# ------------------------------------------------------------------------------------
#
# Router Instance: tb_noc.noc_tester.Adder_router.switch_c1_in
# Router Address: 100, Mask: 100, Packet Destination: 100
# Received packet from a child node
# Sent = 11000010000000000000000000000000000001011011101 to parent
# ------------------------------------------------------------------------------------
#
```



Test #99: The packet is injected from MEM node and its destination is the PE1 node. The NoC will then choose the shortest path to route the packet as depicted in the diagram below.

```
# ------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------
# ****************************** TB TEST #99 @ Time: 5496ns ******************************
# Source Node: 110, Destination Node: 100, Data: 00000000000000000000000000001010111110,
# ------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------
#
# Router Instance: tb_noc.noc_tester.MEM_router.switch_p_in
# Router Address: 110, Mask: 110, Packet Destination: 100
# Received packet from a parent node
# Sent = 01001100000000000000000000000000001010111110 to child-1
# ------------------------------------------------------------------------------------------------------
#
# Router Instance: tb_noc.noc_tester.Adder_router.switch_c2_in
# Router Address: 100, Mask: 100, Packet Destination: 100
# Received packet from a child node
# Sent = 01001100000000000000000000000000001010111110 to parent
# ------------------------------------------------------------------------------------------------------
#
#
```



10. Work to be Completed

Functional Blocks (adder + PE)
System-Level Integration and Testing