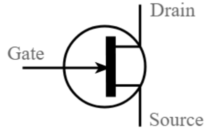
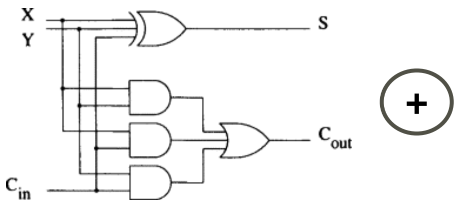
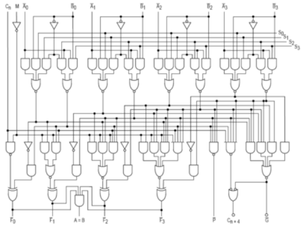
CGRA (Coarse Grain Reconfigurable Array)



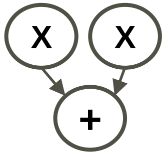
Transistor Gate



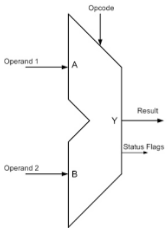
Fixed Operation (E.g., Adder)



Fixed Function

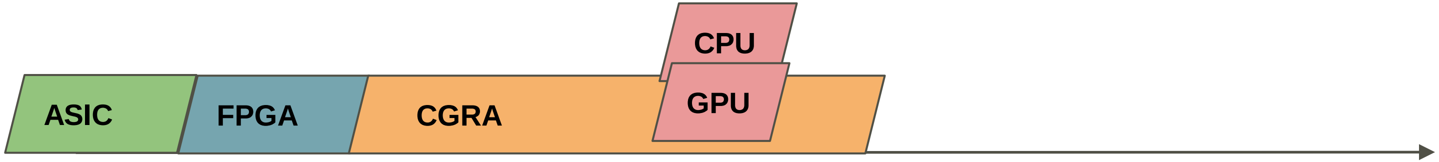


Data Flow Graph



ALU

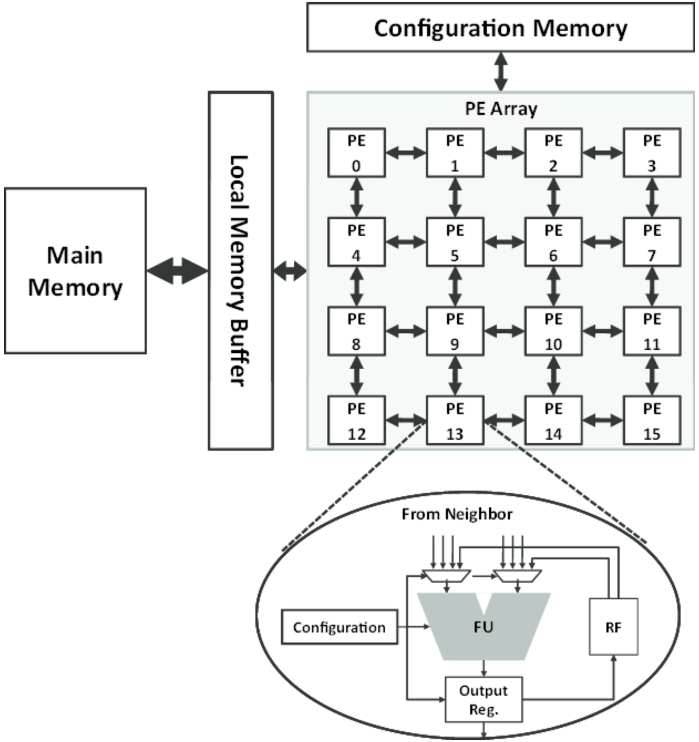
Configurable Component Inside PE **PE**



None **Gate** **Fixed Operation** **Fixed Function**

- Kim, Yongjoo & Lee, Jongeun & Shrivastava, Aviral & Paek, Yunheung. (2011). Memory Access Optimization in Compilation for Coarse-Grained Reconfigurable Architectures. ACM Trans. Design Autom. Electr. Syst.. 16. 42. 10.1145/2003695.2003702.

CGRA (Coarse Grain Reconfigurable Array)

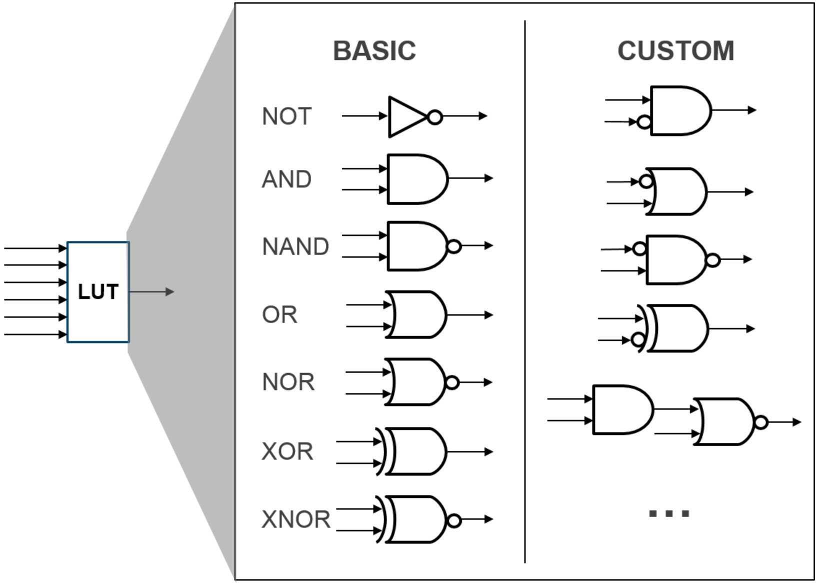


- Kim, Yongjoo & Lee, Jongeun & Shrivastava, Aviral & Paek, Yunheung. (2011). Memory Access Optimization in Compilation for Coarse-Grained Reconfigurable Architectures. ACM Trans. Design Autom. Electr. Syst.. 16. 42. 10.1145/2003695.2003702.

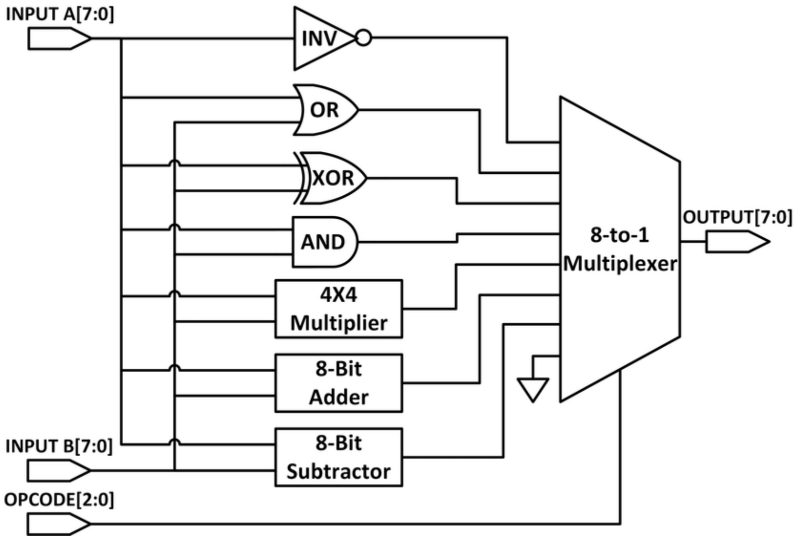
UCLA Samueli Electrical & Computer Engineering

CGRA VS FPGA: Compute Primitive

FPGA: Bit-wise



CGRA: Customized Word Length



- https://www.rapidwright.io/docs/FPGA_Architecture.html
- Wan, Tutu & Karimi, Yasha & Stanacevic, Milutin & Salman, Emre. (2019). AC Computing Methodology for RF-Powered IoT Devices. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. PP. 1-12. 10.1109/TVLSI.2019.2894531.

UCLA Samueli Electrical & Computer Engineering

Accelerate compute

```
for (int idx = 0; idx < n; idx++)
{
    A[idx] = B[idx] * C[idx] + D[idx];
}
```

Von neumann V.S Data-flow

CPU

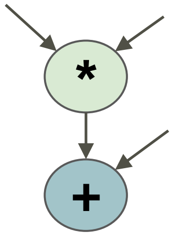
Instruction *(fetch, decode…)*
Data Load
X = B * C
Data Store
Instruction *(fetch, decode…)*
Data Load
A = X + D
Data Store

GPU

Instruction *(fetch, decode…)*		
Data Load		
X = B * C	X = B * C	X = B * C
Data Store		
Instruction *(fetch, decode…)*		
Data Load		
A = X + D	A = X + D	A = X + D
Data Store		

CGRA

Instruction *(fetch, decode…)*		
Data Load		
X = B * C	X = B * C	X = B * C
A = X + D	A = X + D	A = X + D
Data Store		

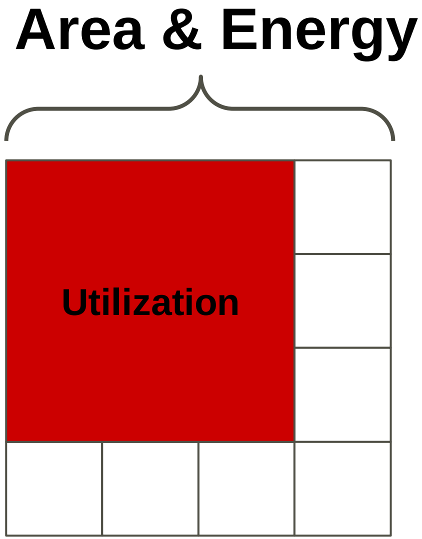


UCLA Samueli Electrical & Computer Engineering

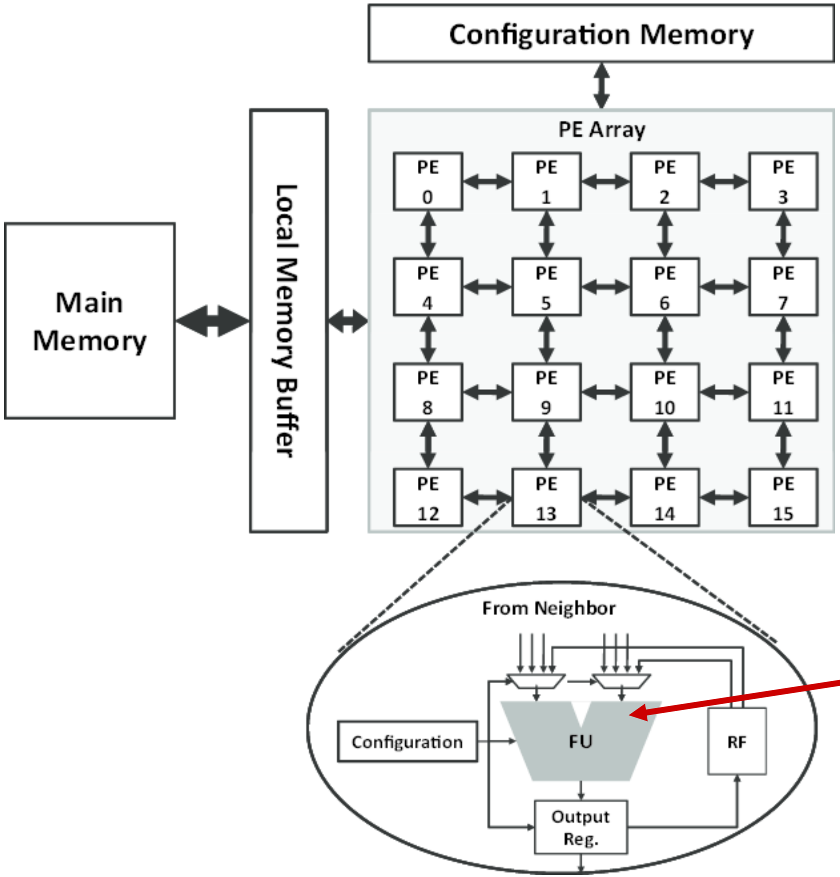
Performance Metric

- **Compute efficiency:**
  Hardware
  - GOP/s/mm
  - GOP/s/mw

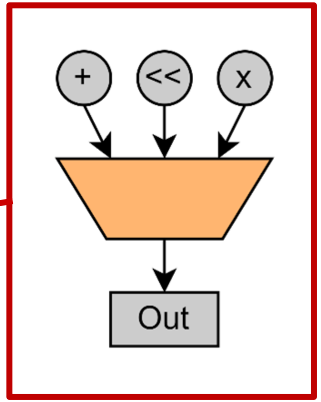
- **Resource utilization:**
  Software
  - Spatial
  - Temporal



CGRA Compute

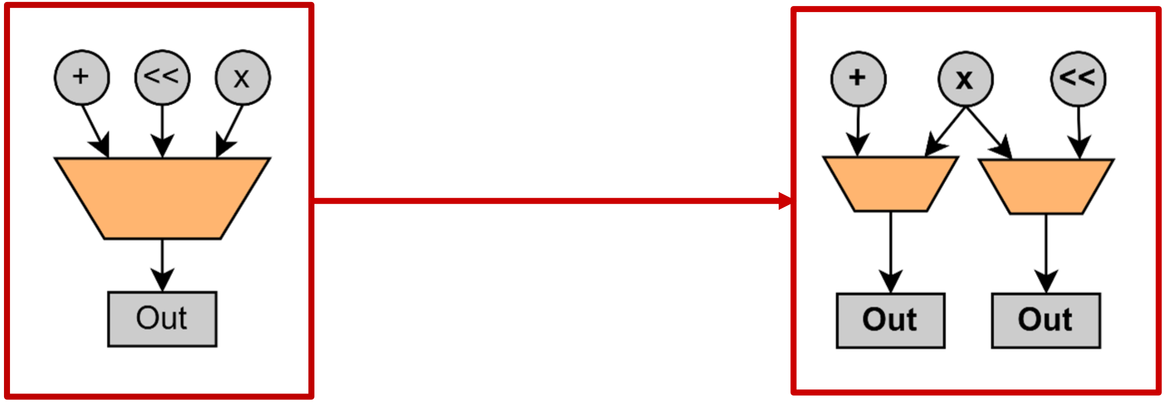


Is this Processing Element (PE) design efficient?



CGRA Compute

```
for (int idx = 0; idx < n; idx++)
{
    X[idx] = B[idx] * C[idx];
    Y[idx] = B[idx] << C[idx];
}
```



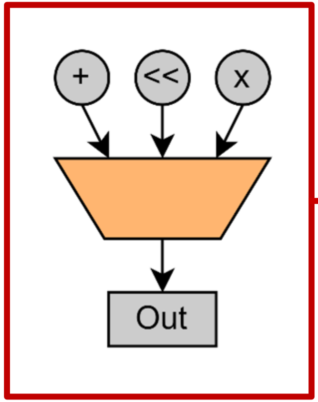
CGRA Compute

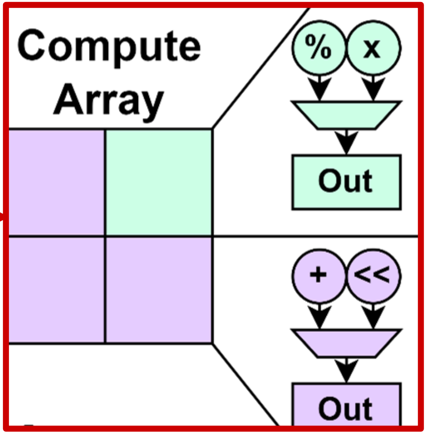
```
for (int idx = 0; idx < n; idx++)
{
    A[idx] = B[idx] * C[idx] + D[idx];
}
```

UCLA Samueli
Electrical & Computer Engineering

CGRA Compute

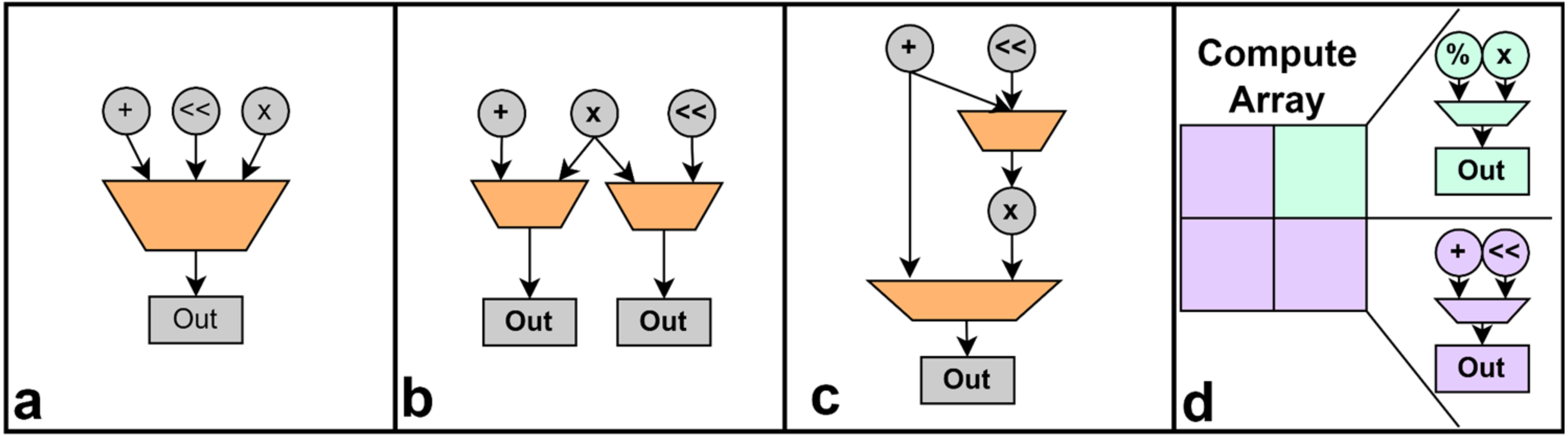
```
for (int idx = 0; idx < n; idx++)
{
    if (idx % 2 == 0) {X[idx] = B[idx] % C[idx];}
    else{ Y[idx] = B[idx] * C[idx]; }
}
for (int idx = 0; idx < 3n; idx++)
{
    if (idx % 2 == 0) { X[idx] = B[idx] + C[idx]; }
    else { Y[idx] = B[idx] << C[idx]; }
}
```



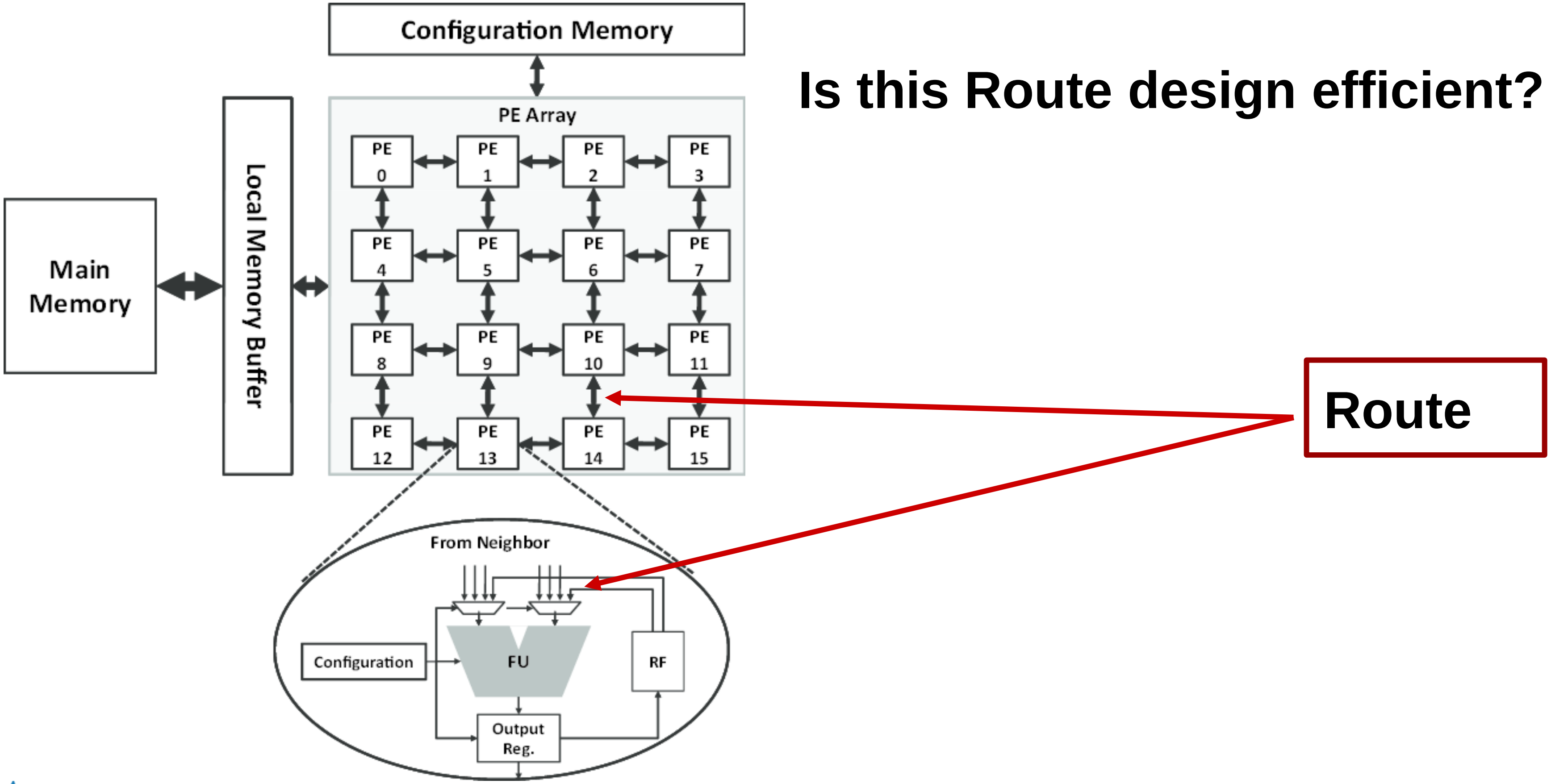


CGRA Compute

ALU PE | Multi-Path | Expression | heterogeneity



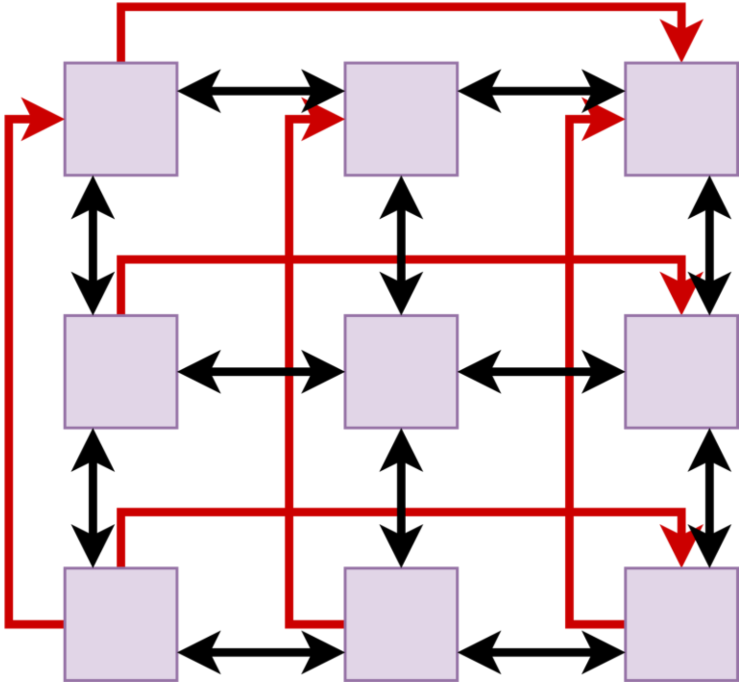
CGRA Route



Is this Route design efficient?

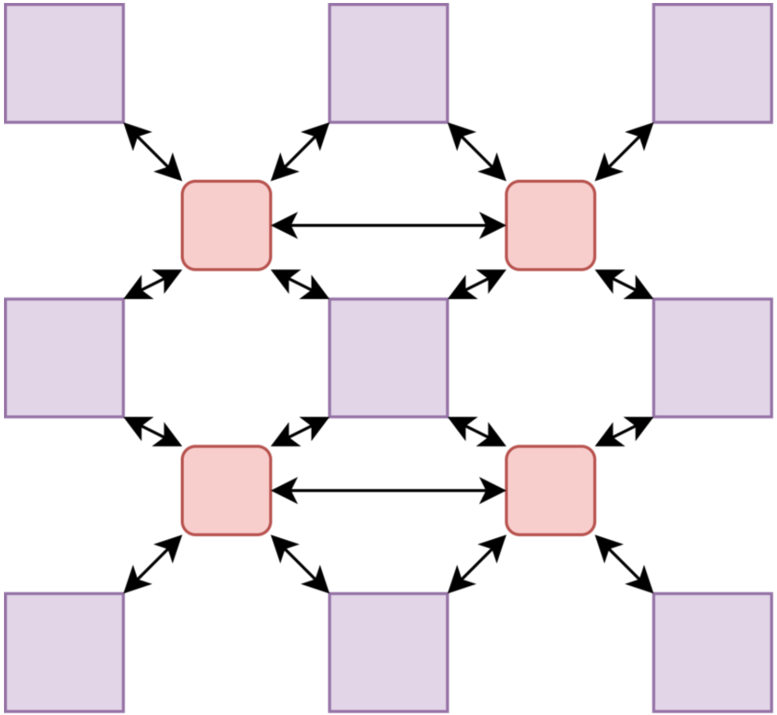
CGRA Route

Multi-hop Route: communication over distances greater than a single tile through direct route paths

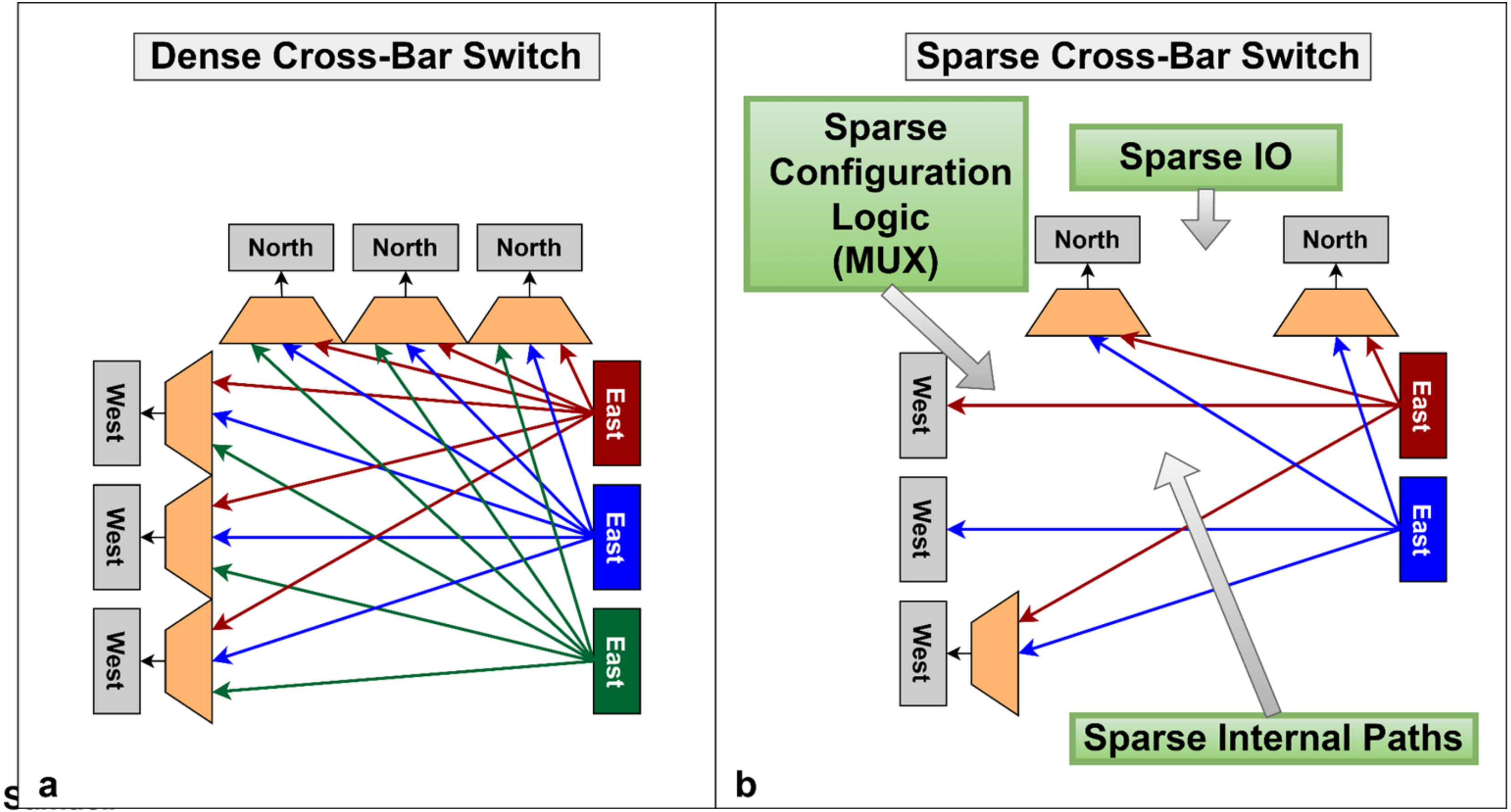


CGRA Route

Decoupled Compute and Route: eliminate PE route-through



CGRA Route


Dense Cross-Bar Switch
North
North
North
West
West
West
East
East
East
a
Sparse Cross-Bar Switch
Sparse Configuration Logic (MUX)
Sparse IO
North
North
West
West
West
East
East
Sparse Internal Paths
b

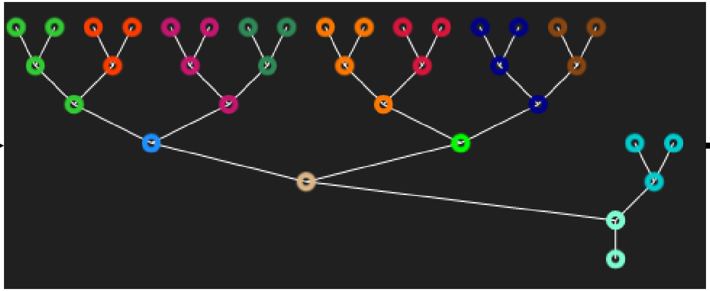
CGRA Compiler

High-level Language

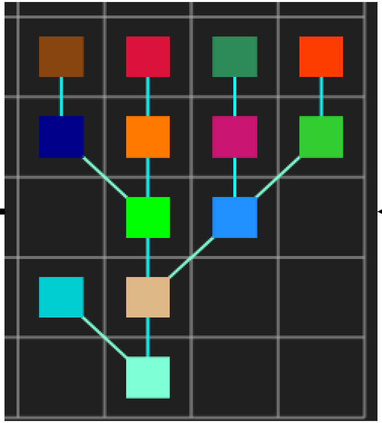
```
int MyKernel()
{
        int i[4];
        int w[4];
        int output = i[0] * w[0];
        output += i[1] * w[1];
        output += i[2] * w[2];
        output += i[3] * w[3];

        return output;
}
```

Data-flow Graph



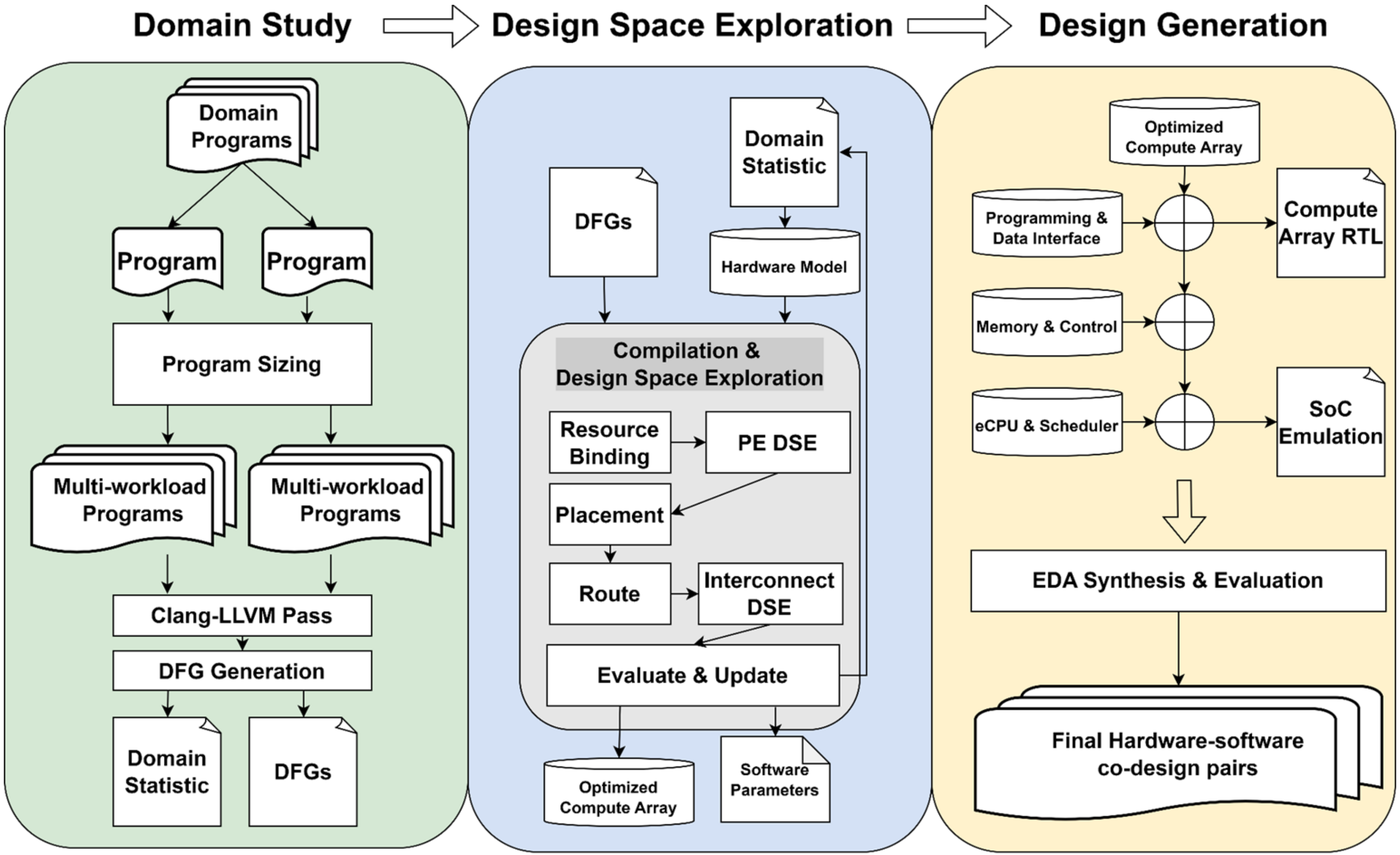
Program Mapping



Configuration Sequence

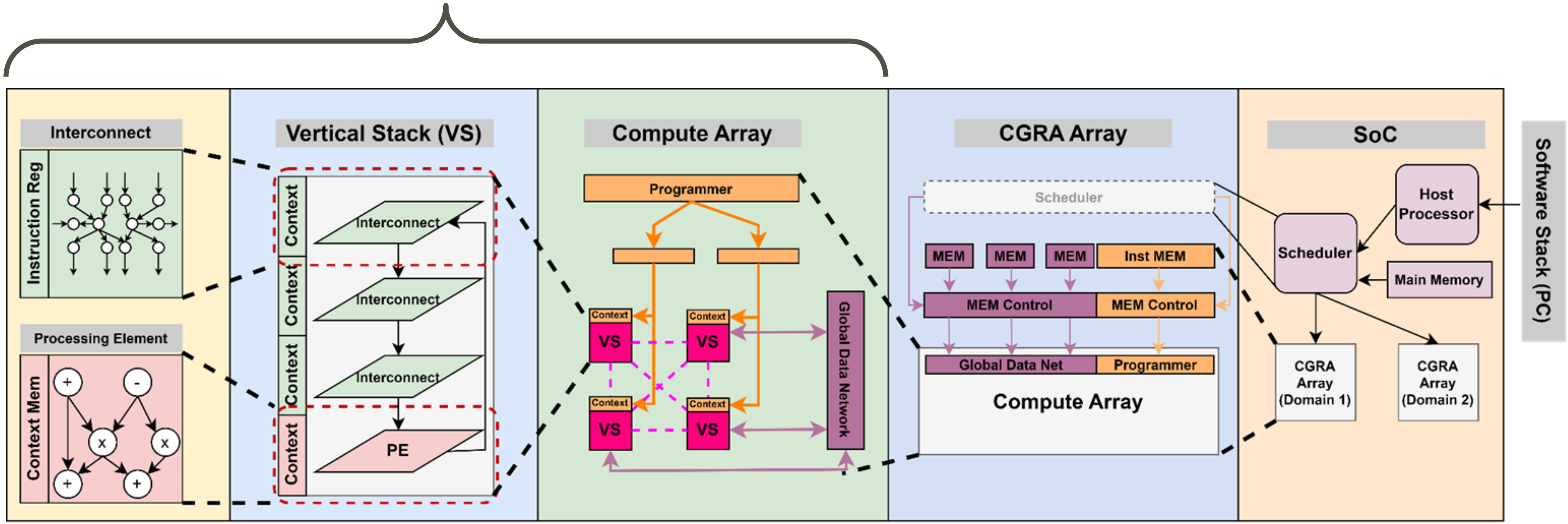
```
#20
rst = 1'b0;
configIn =7'b0000000;
controlIn =36'b110100110000100100000000000000000000;
#2
configIn =7'b1011100;
controlIn =36'b000000001000010000000000000000000000;
#2
configIn =7'b0101100;
controlIn =36'b001000000000000010000000000000000000;
#2
configIn =7'b0000001;
controlIn =36'b000000000100000000000001111000000000;
#2
configIn =7'b0001010;
controlIn =36'b000000000000000000111100000000000000;
```

CADA Tool Flow

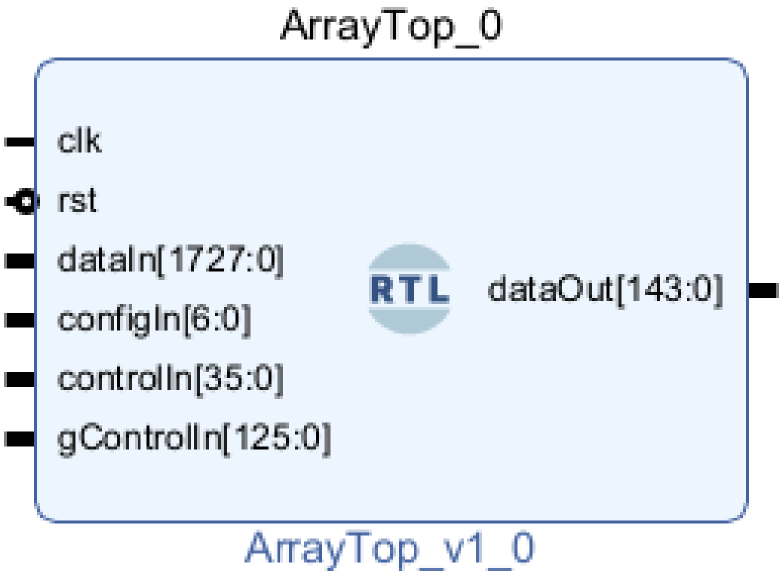


CADA Design Hierarchy

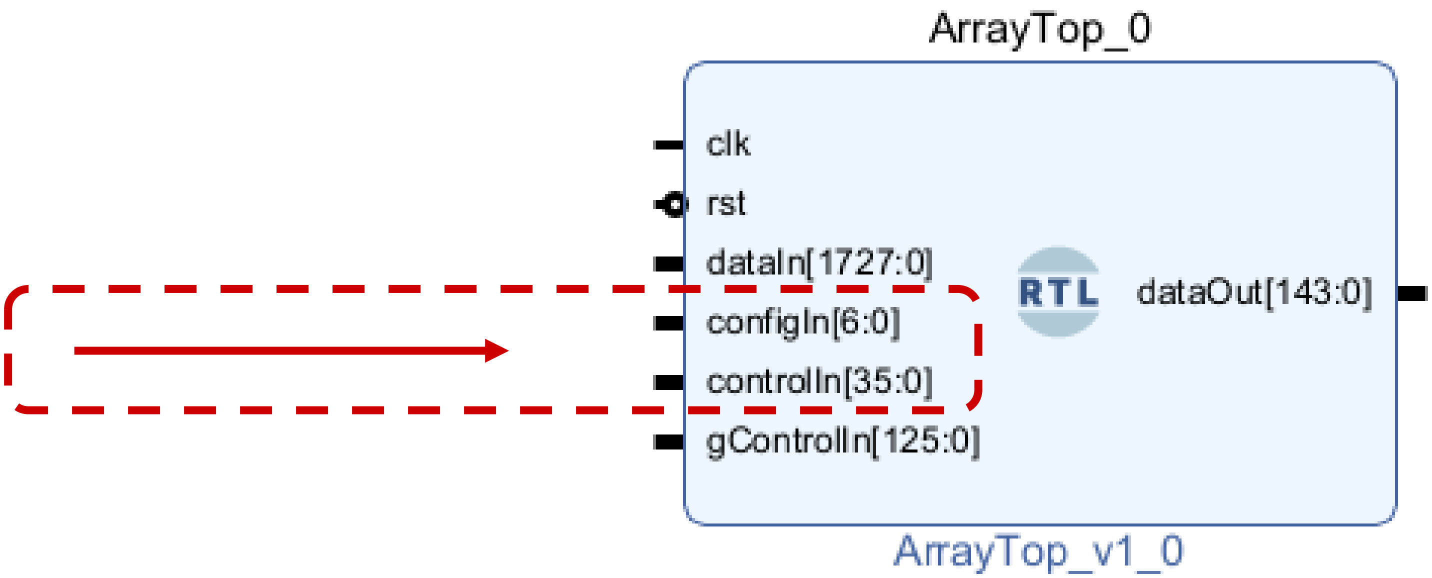
Accessible to EE216B



Understand CADA Compute Array

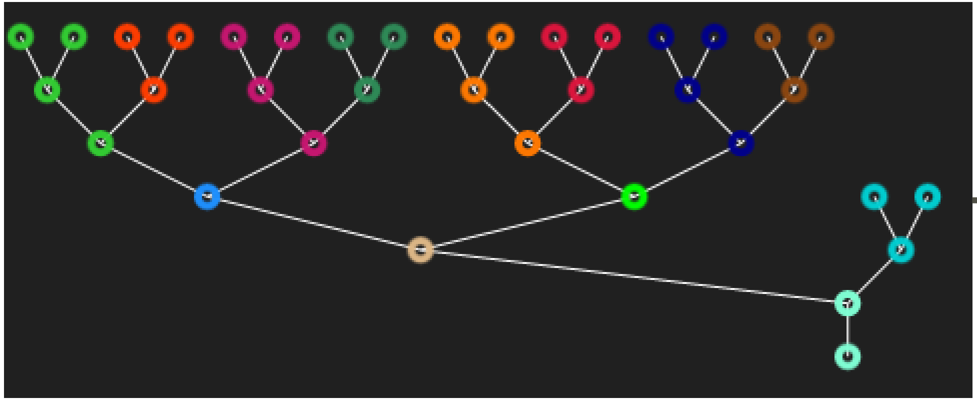


Configuration Programming

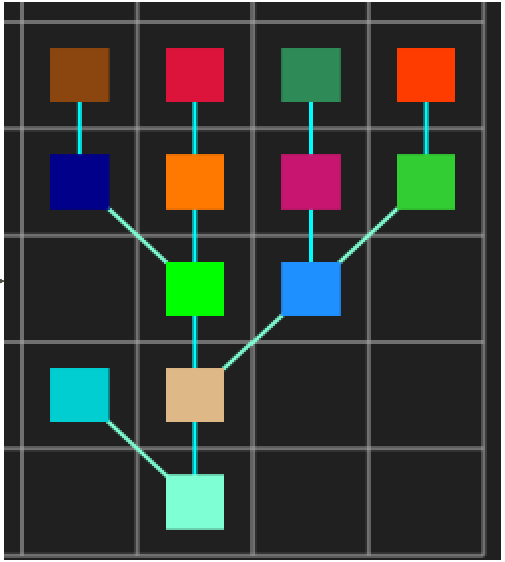


Mapping of “3x3 ConvolutionTree”

DFG Partition

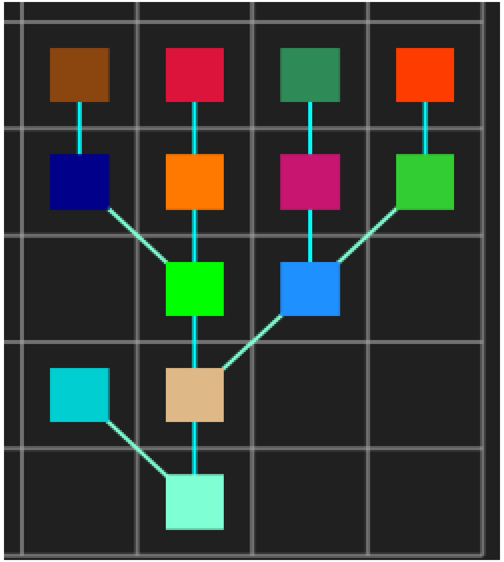


Spatial Mapping

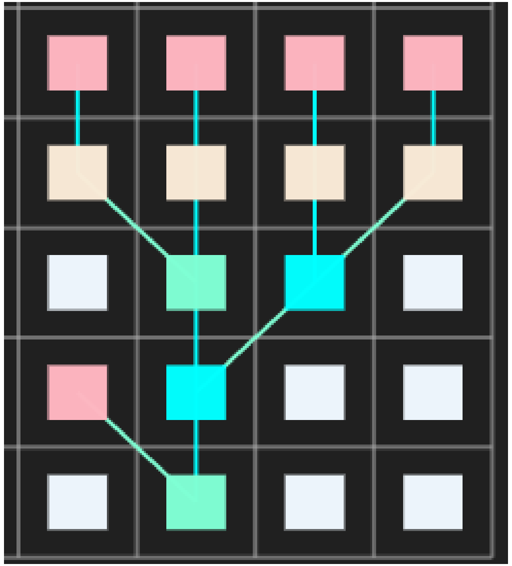


Configuration of "3x3 ConvolutionTree"

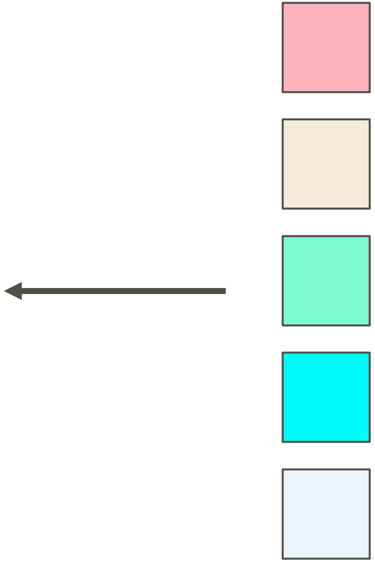
Soft-map



Configuration



5 unique Configurations

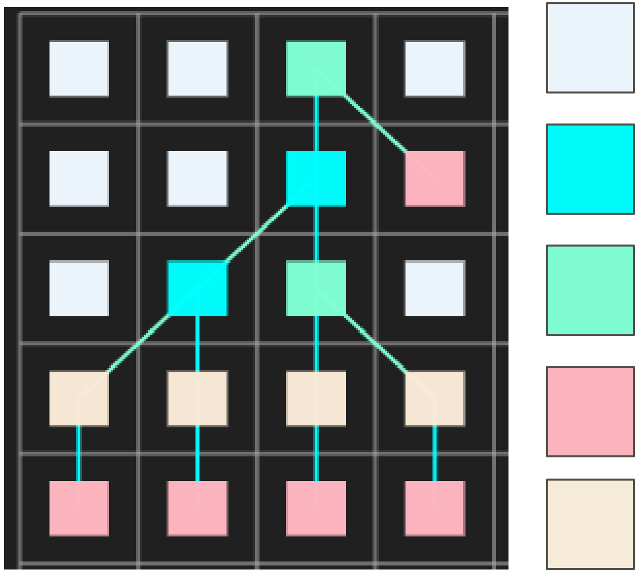


5 Programming cycle

UCLA Samueli Electrical & Computer Engineering

Configuration Sequence

Configuration



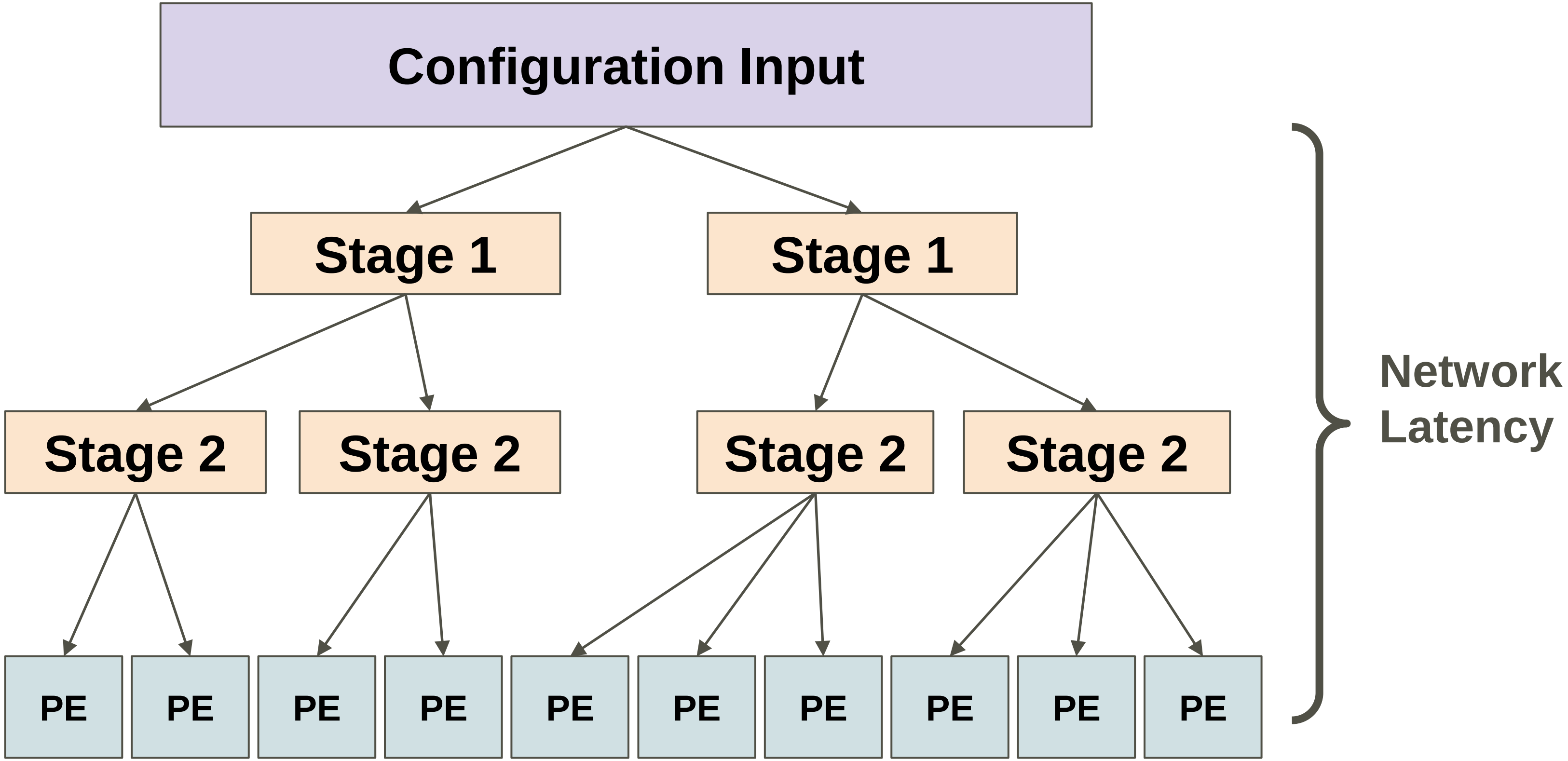
Testbench (Verilog)

```
#20
rst = 1'b0;
configIn =7'b0000000;
controlIn =36'b110100110000100100000000000000000000;
#2
configIn =7'b1011100;
controlIn =36'b000000001000010000000000000000000000;
#2
configIn =7'b0101100;
controlIn =36'b001000000000000100000000000000000000;
#2
configIn =7'b0000001;
controlIn =36'b000000000100000000000000111100000000;
#2
configIn =7'b0001010;
controlIn =36'b000000000000000000111100000000000000;
```

Configuration

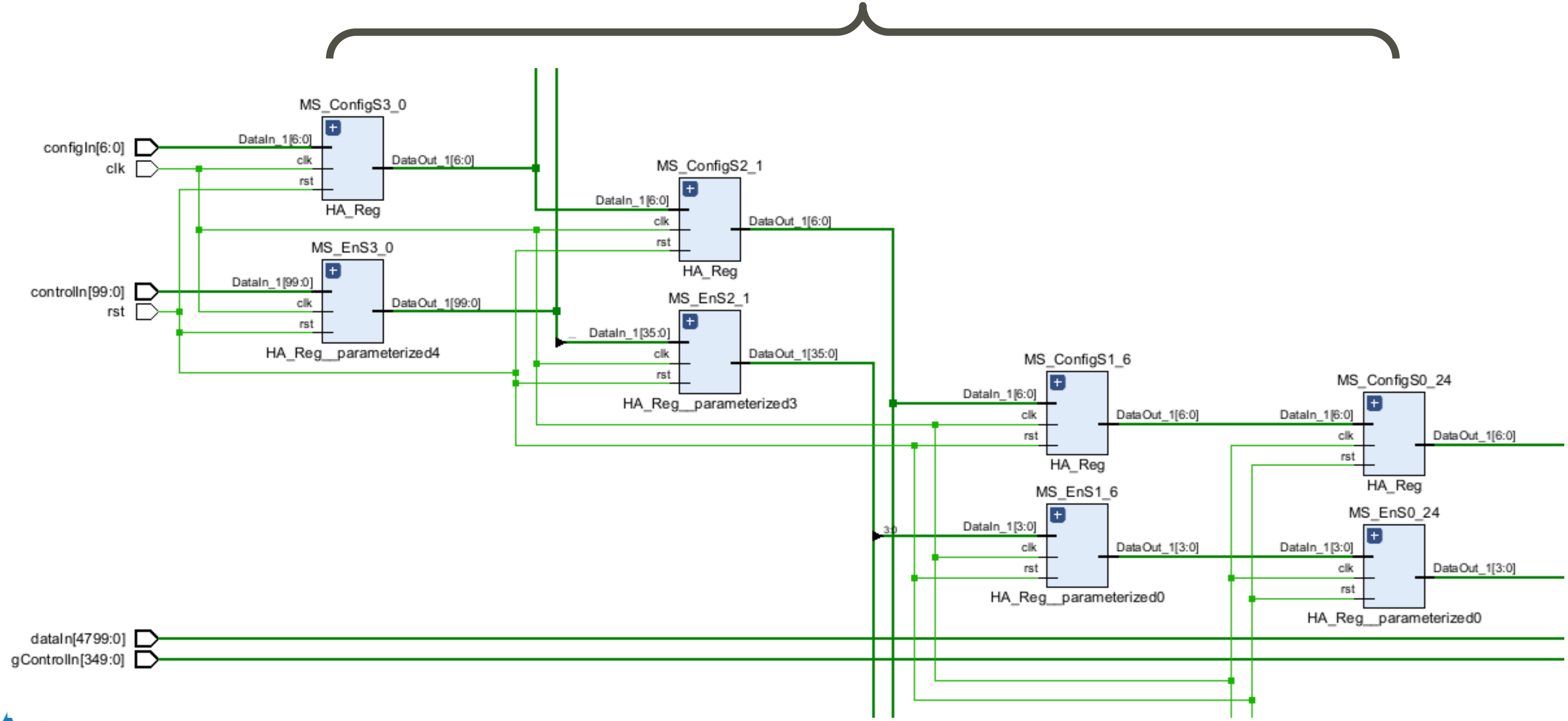
Position (one-hot)

Program Broadcast



Programming Network Latency

4 Stages = 4 cycle latency

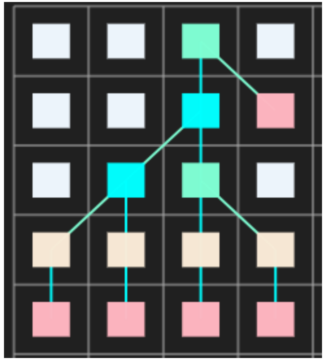


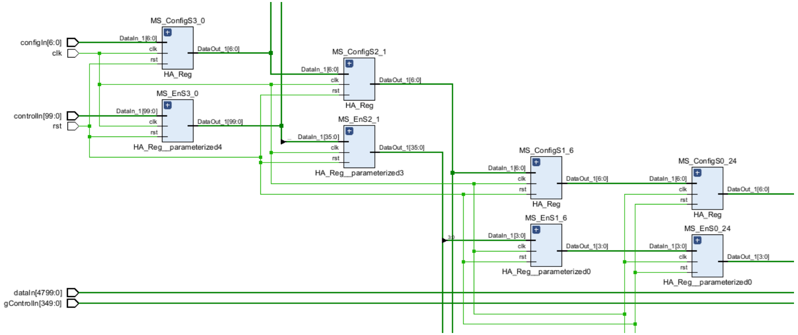
Programming Latency

Total Programming Latency:
5 + 4 = 9

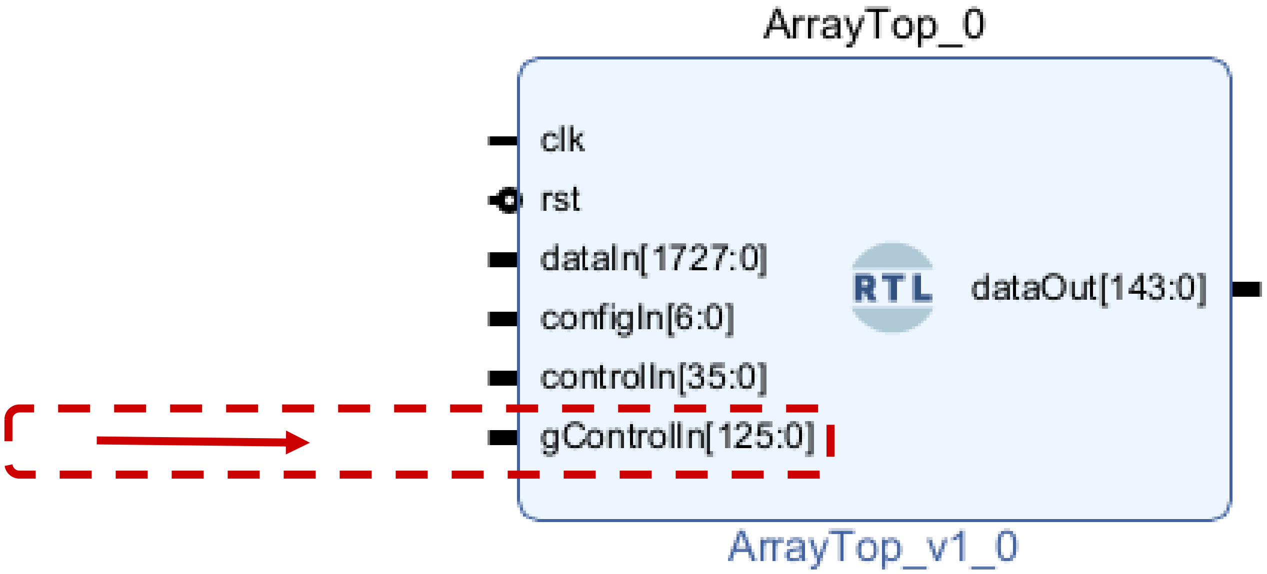
5 cycle

4 cycle



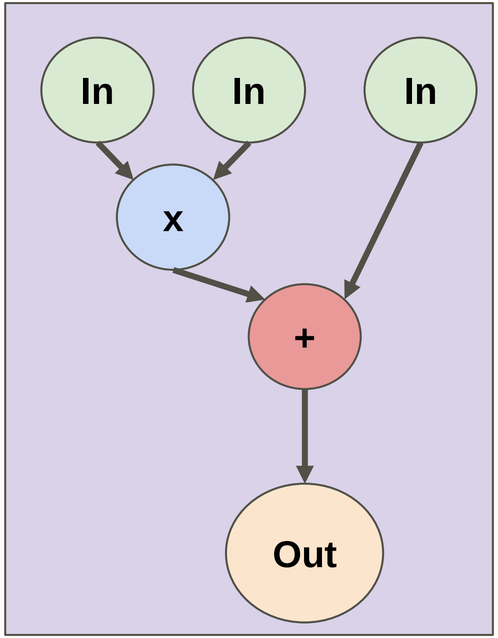


Global IO Control

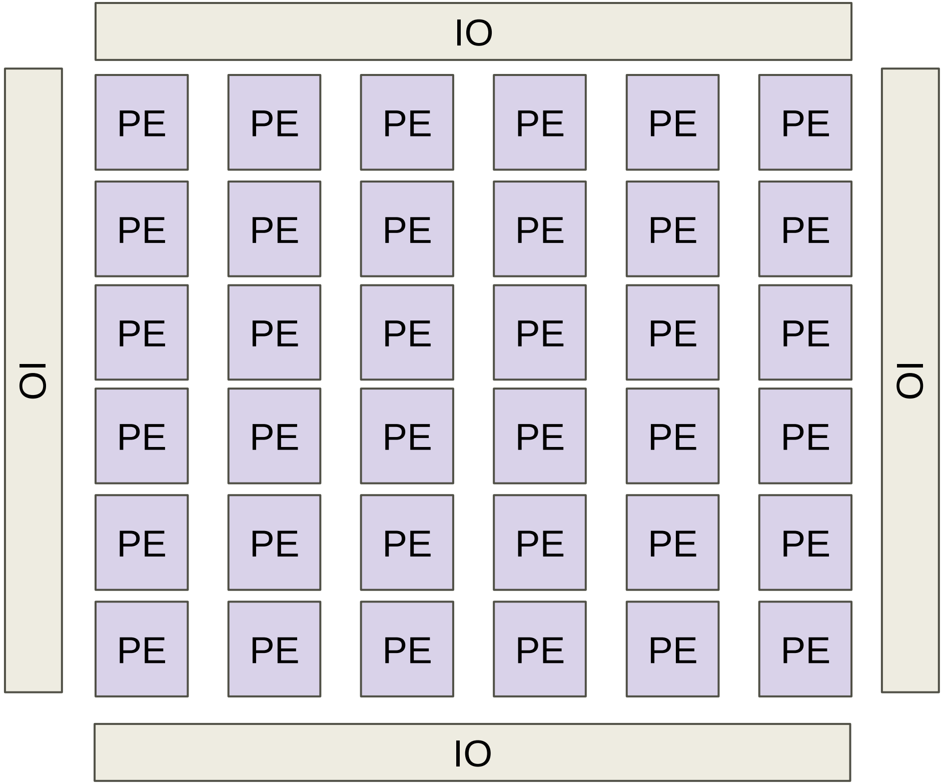


Global IO

PE Architecture

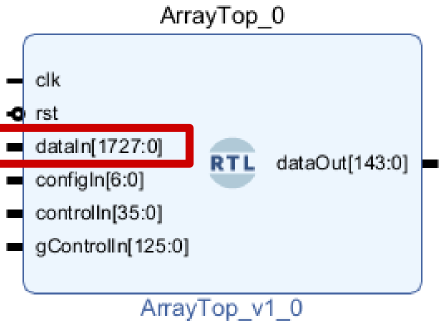


6x6 PE Array



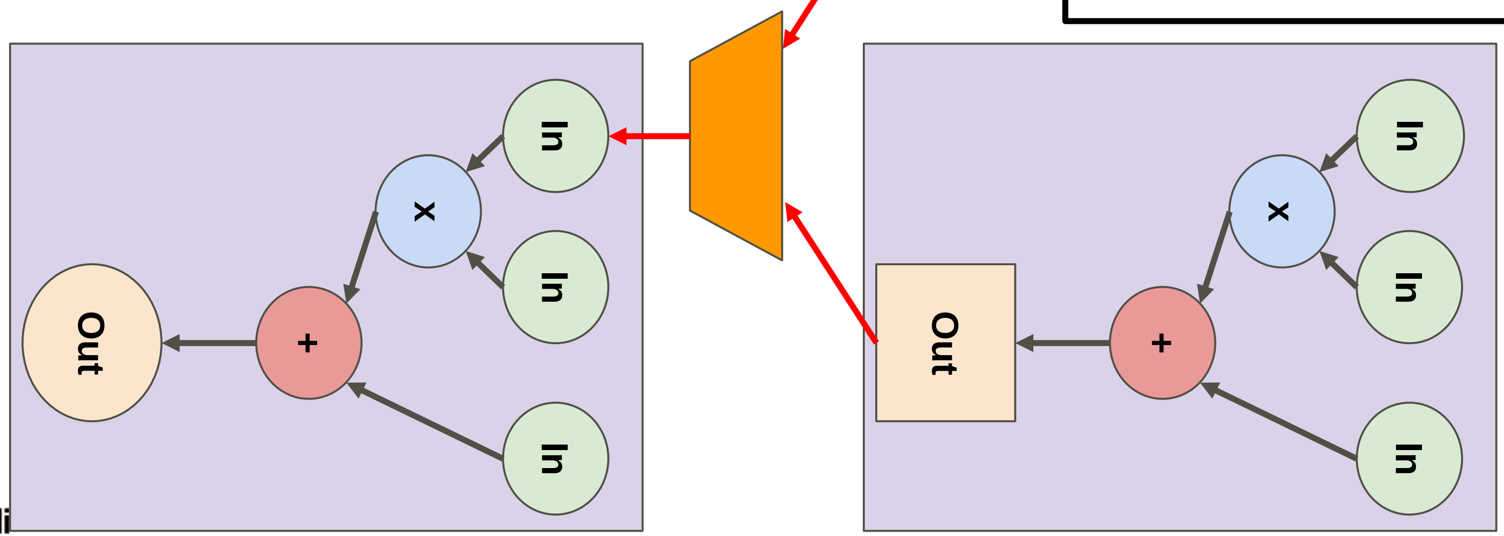
Input Data IO

Data IO

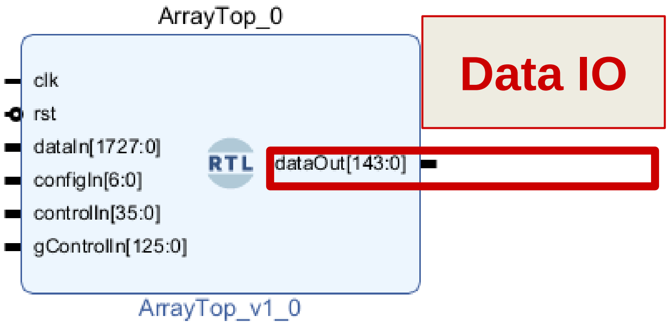


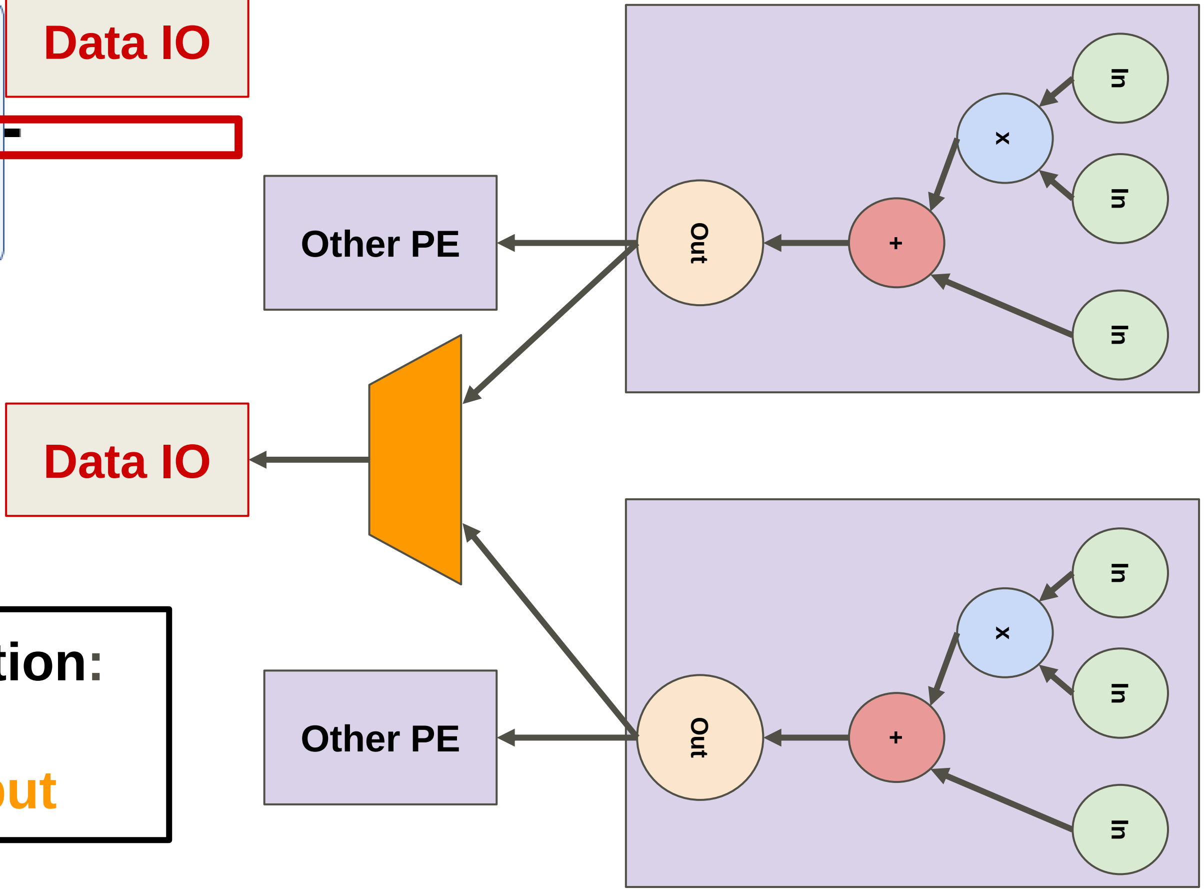
Input configuration:

From IO or
From other PE



Output Data IO

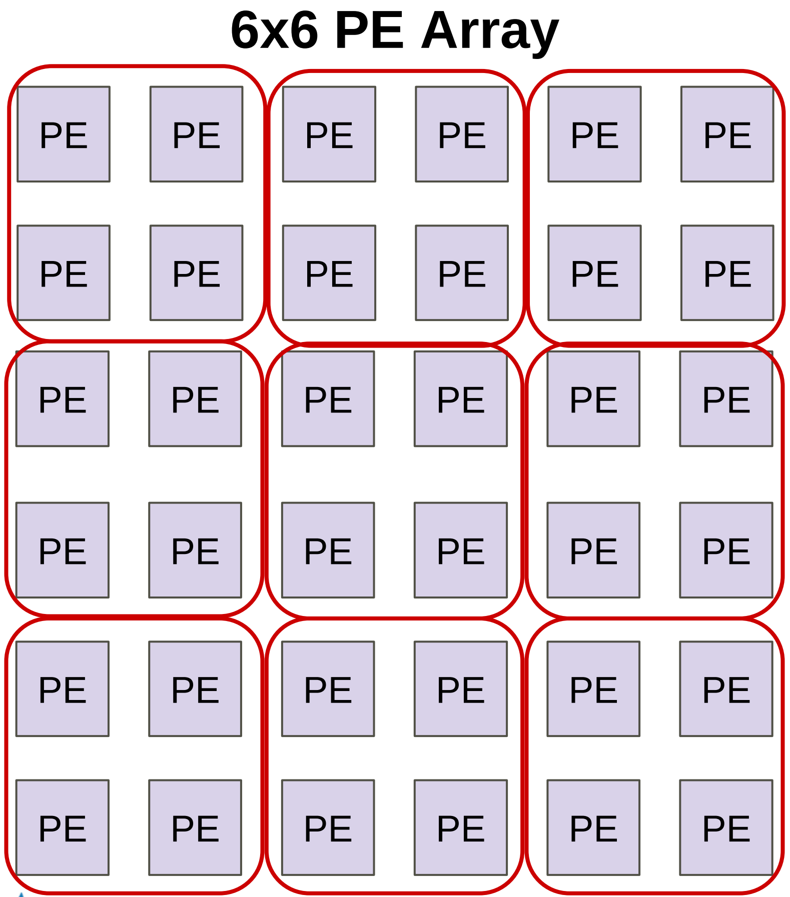




Output configuration:

Select 1 of N output

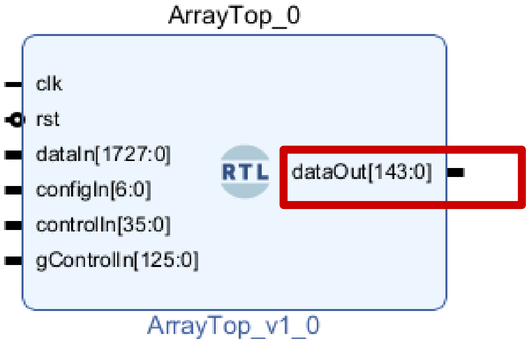
Global Output IO Arrangement



PE output to IO:

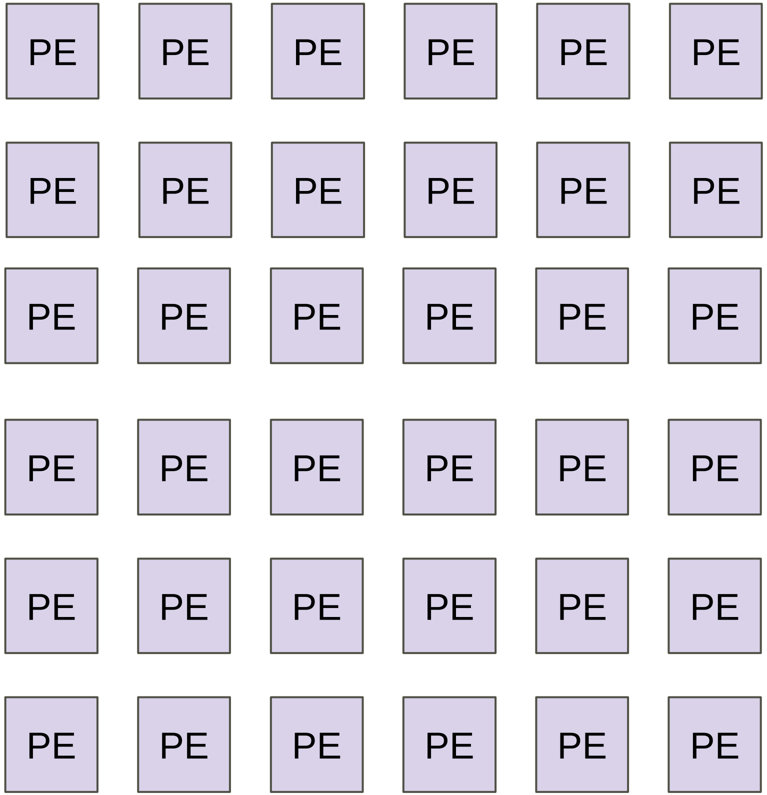
9 tiles = 9 output * 16bit = 144 bits

Each tiles use 4-input-MUX=
2 configuration bits
Total 18 bits



Global Input IO Arrangement

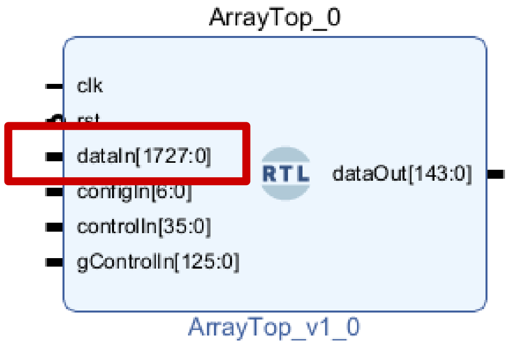
6x6 PE Array



IO to PE inputs:

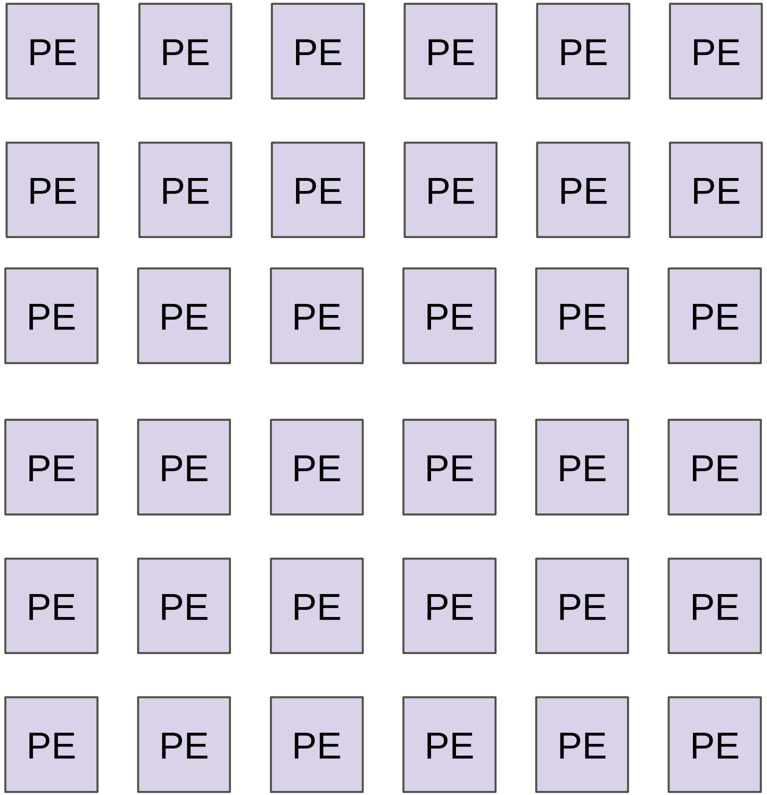
36 PEs, each PE has 3 inputs
36 * 3 = 108 inputs
108 * 16 = 1728 total Input bits

Each input use a 2-input-MUX
108 * 1 bit = 108 configuration bits

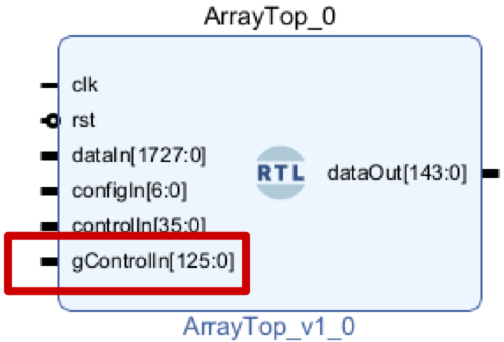


Global IO Control

6x6 PE Array

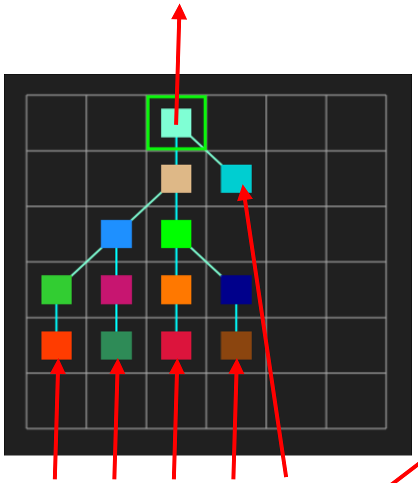


IO Configuration:
18 output configuration +
108 input configuration
= 126 IO configuration bits



Global IO Mapping

Output: selectedChannel (second tile : [31:16])



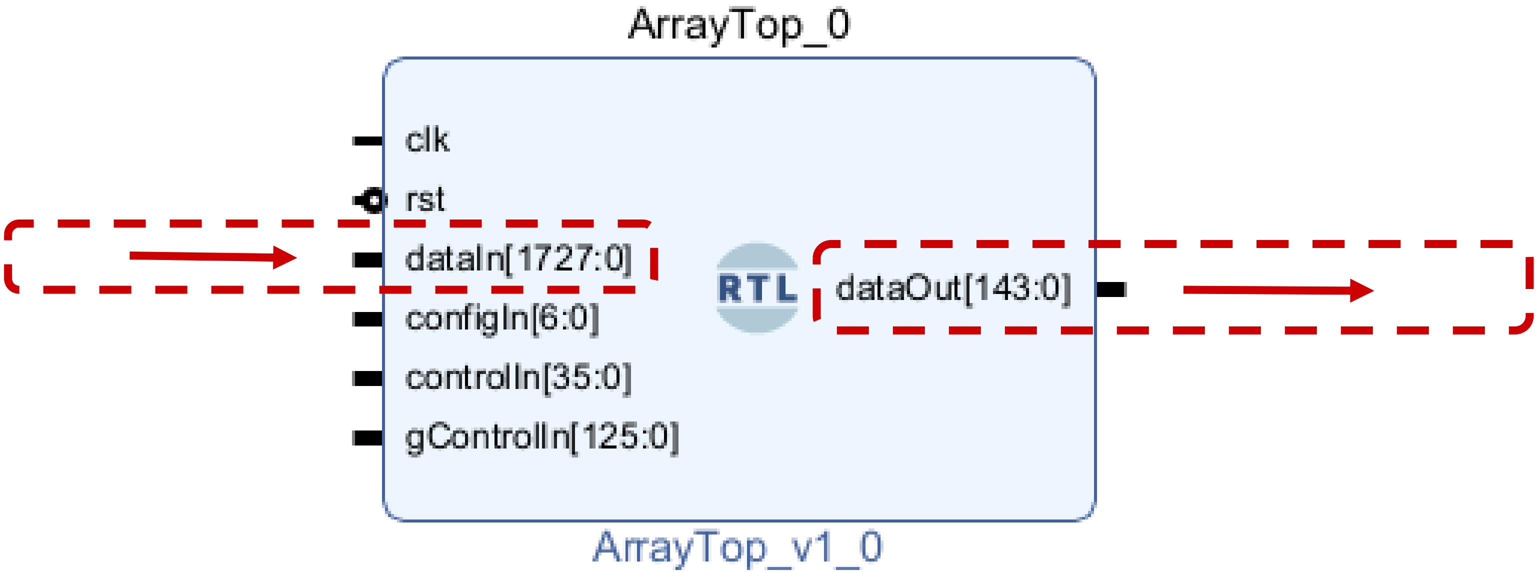
Input

```
#2
assign selectedChannel =  dataOut[31:16];
gcontrolIn =126'b000000000000000000000000000000000000000000110110110110000000110110110110000000000000000000000000011000000000000000000000000000;
```

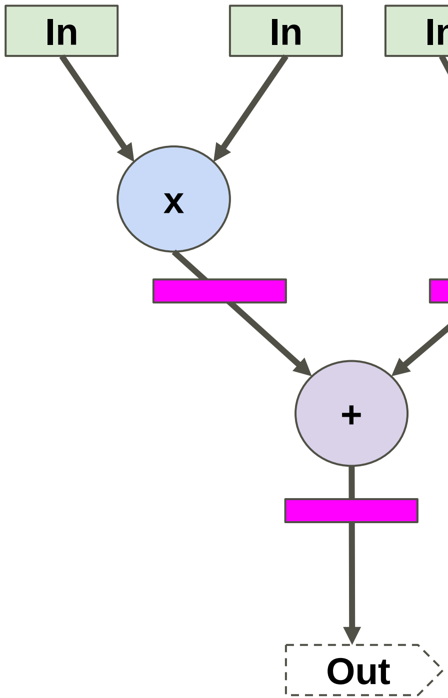
Output Configuration **Input Configuration**

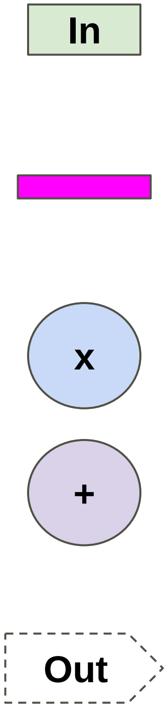
UCLA Samueli Electrical & Computer Engineering

Data In and Out



Data Flow Graph (DFG)





Is a register:
1 cycle latency

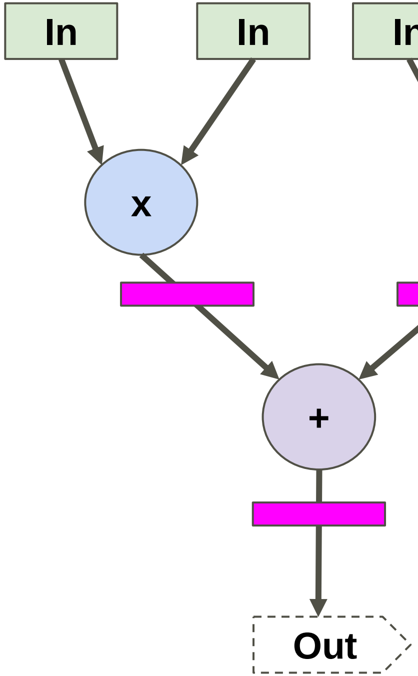
Is a pipeline register:
1 cycle latency

Are combinational circuits:
0 cycle latency

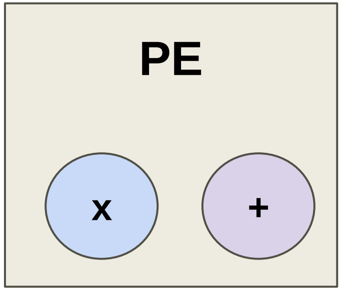
Is a wire:
0 cycle latency

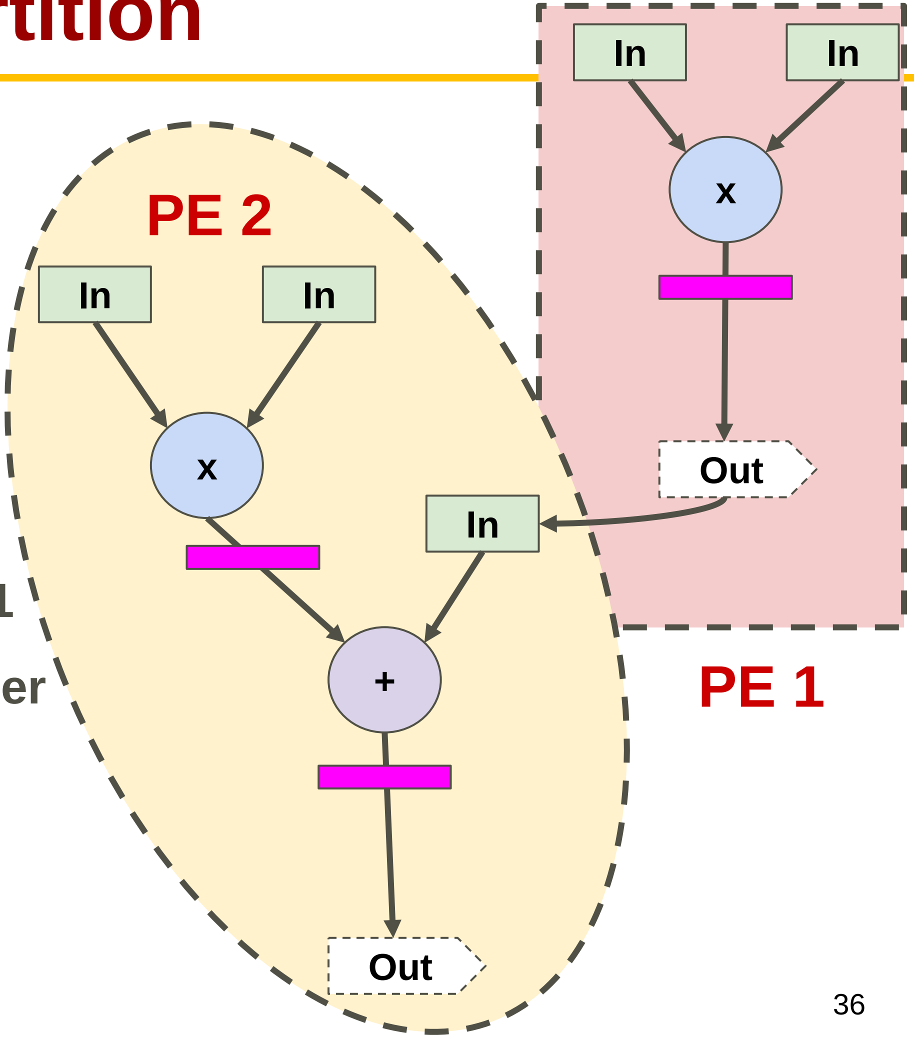
What is the path latency?

DFG Partition

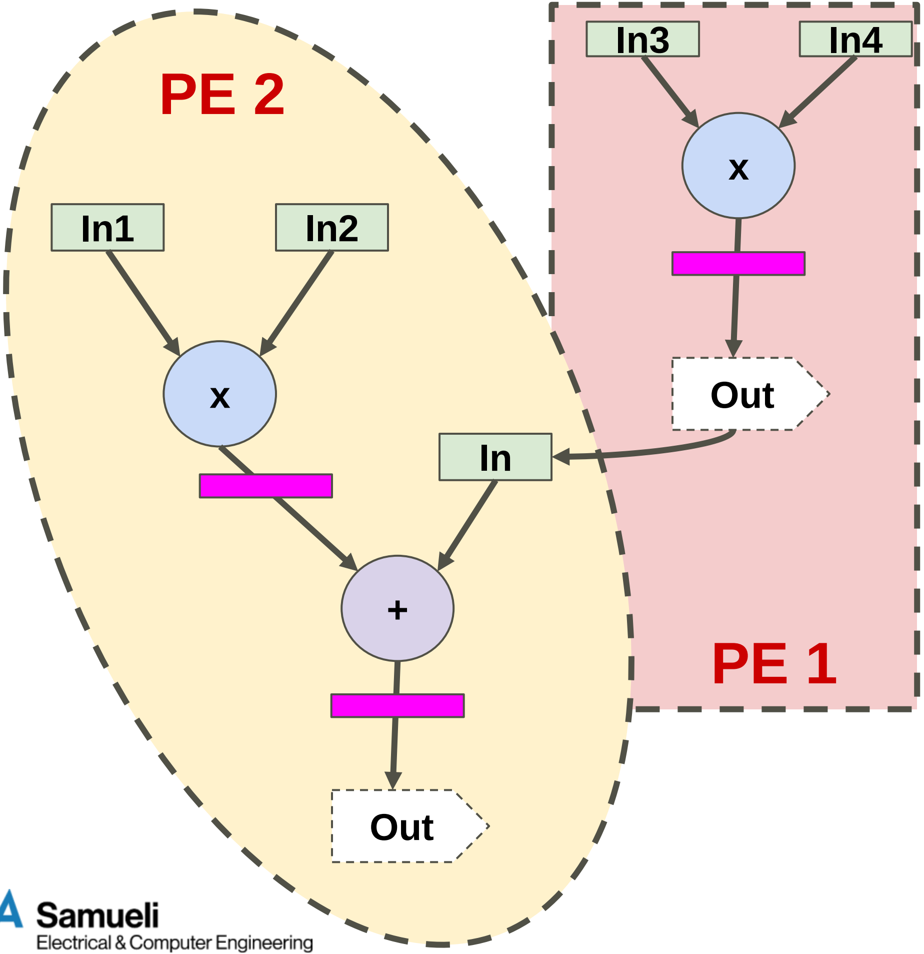


PE only supports 1 multiplier and 1 adder



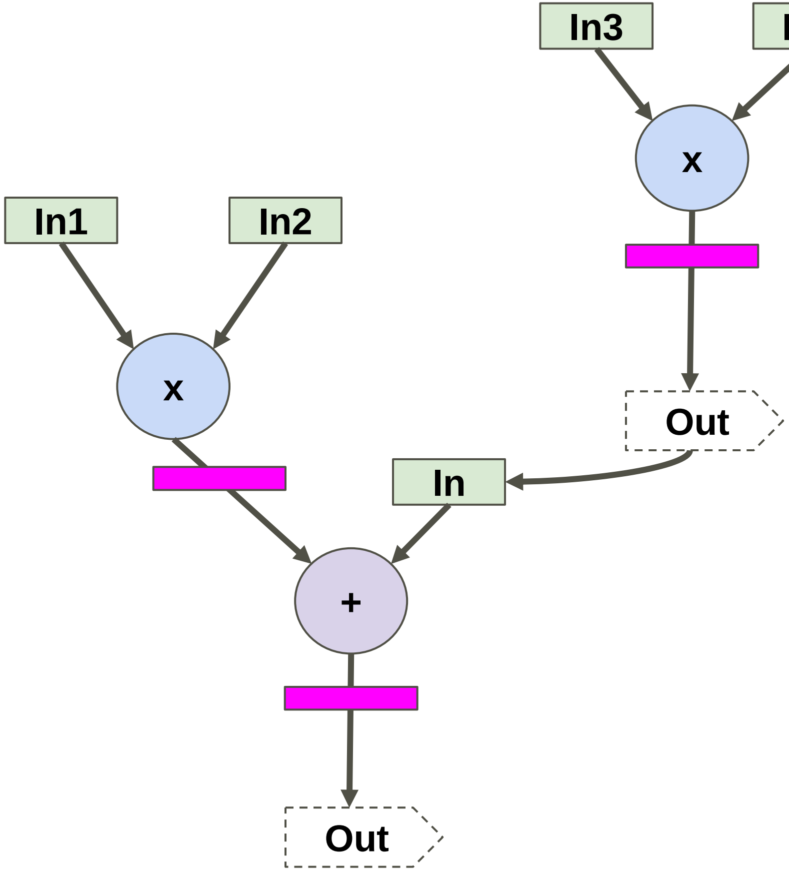


Mapping Path Latency



What is the new path latency?

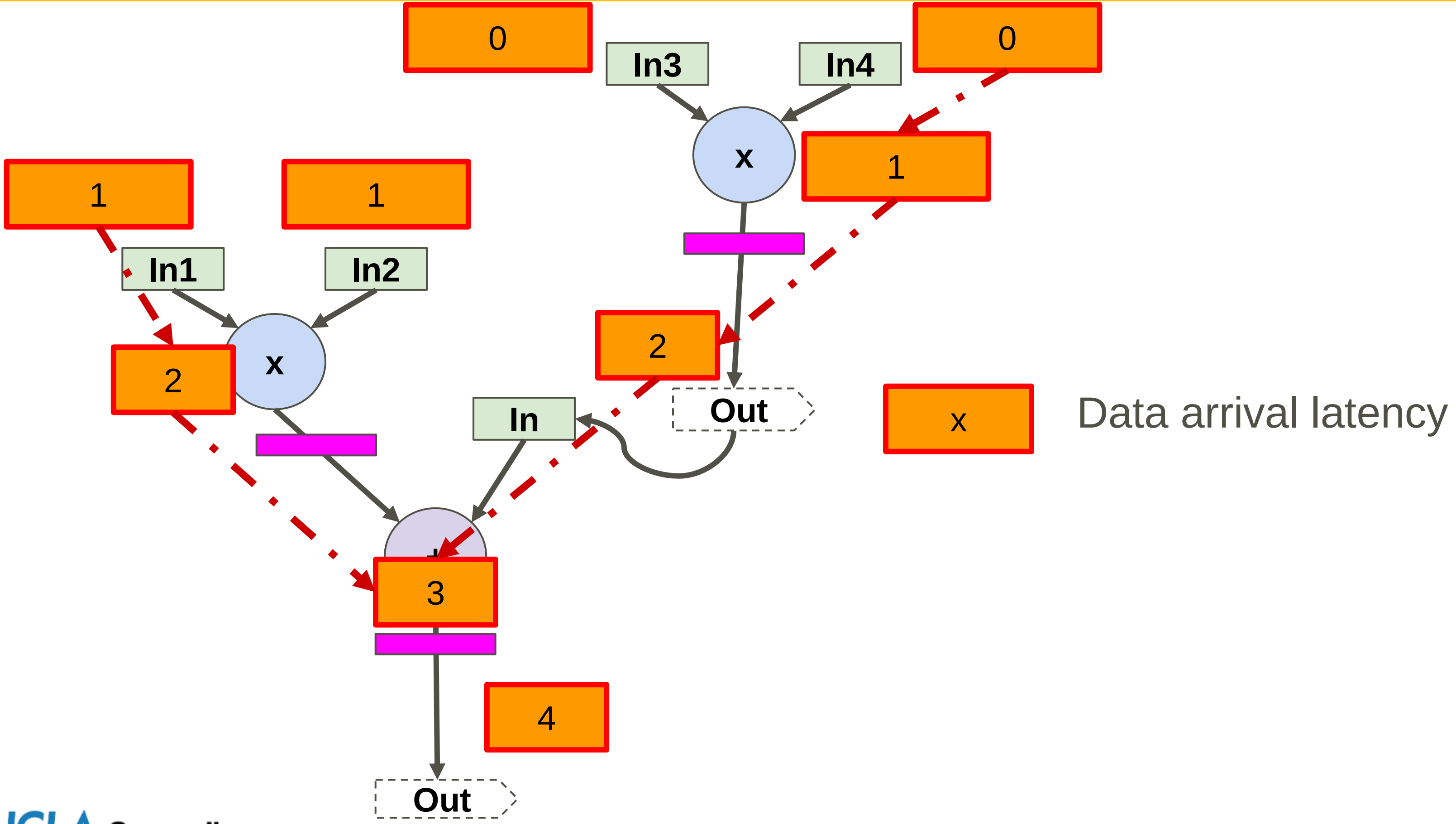
Mapping Input Latency



Do <in1, in2, in3, in4> data come in at the same time?

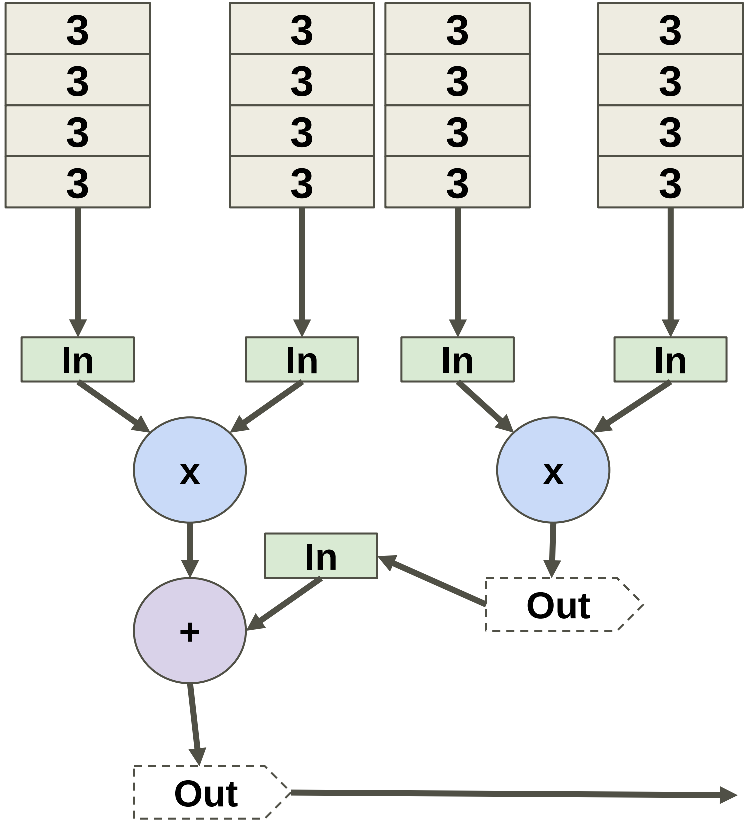
What are the input latency for inputs?

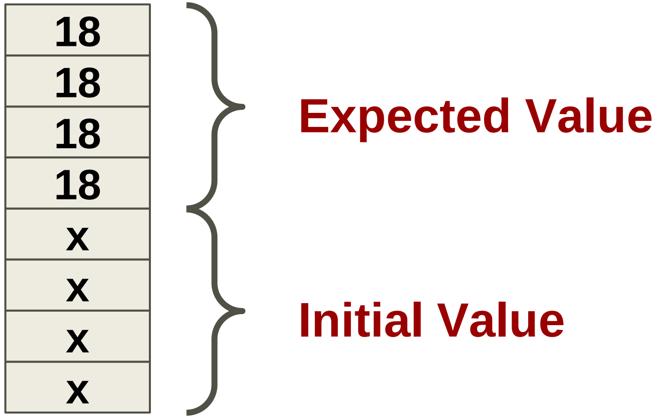
Mapping Input Latency



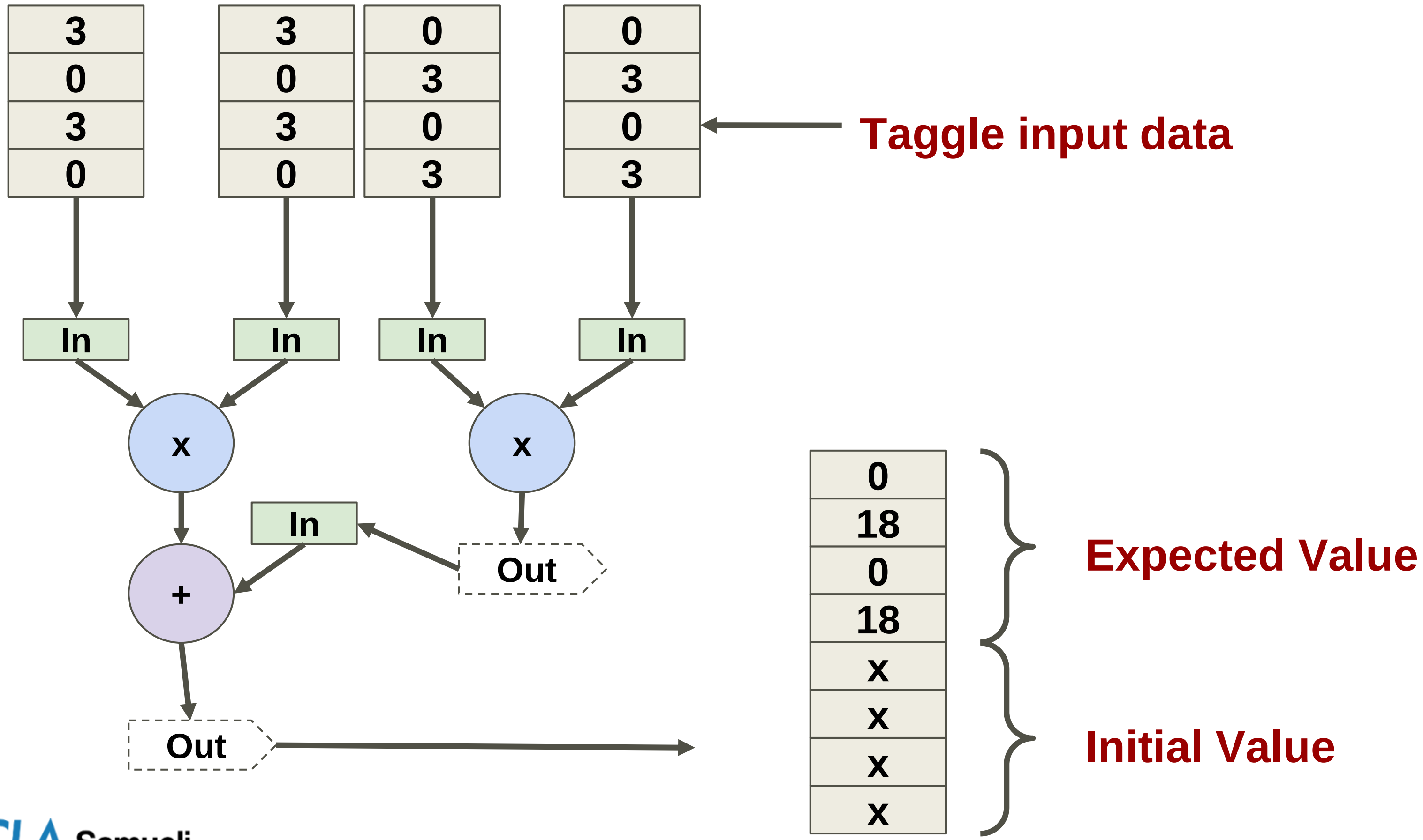
Behavior Test

Testbench to verify functionality





Input Latency Verification



Input Latency Verification