

# Midterm Project 2023 Fall

ICLAB

Author: Lai Lin-Hung @ Si2 Lab

# Routing Problem

## Maze Route Algorithm

- “An algorithm for path connection and its application,”  
Lee, IRE Trans. Electronic Computer, EC-10, 1961.

- Discussion mainly on single-layer routing

- **Strengths**

- Guarantee to find connection between 2 terminals if it exists
- Guarantee minimum path

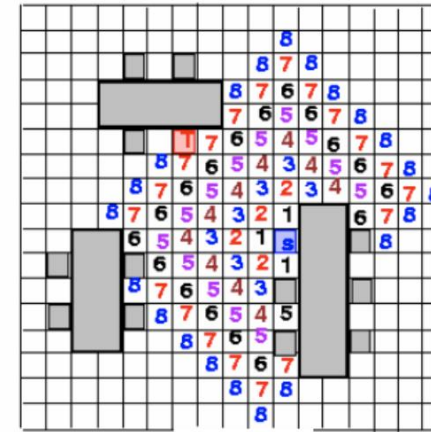
- **Weaknesses**

- Requires **large memory** for dense layout
- **Slow**

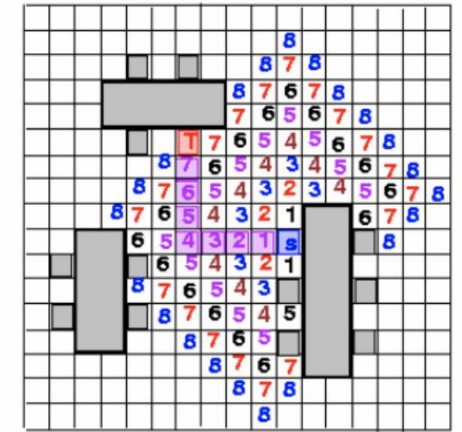
- **Applications:**

- CAD: Detailed routing
- Game Industry: End-to-End path finding
- Robotic: Road Planning

- Find a path from  $S$  to  $T$  by “wave propagation”.



Filling

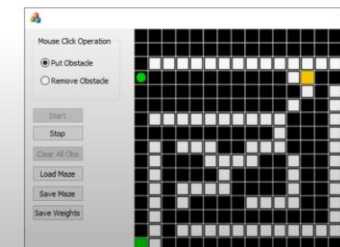


Retrace

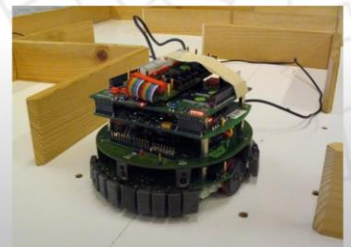
- Time & space complexity for an  $M \times N$  grid:  $O(MN)$   
(huge!)



© VLSI and Circuit Design -



© Lee Algorithm Mazesolver



© Wikipedia

# Routing Problem

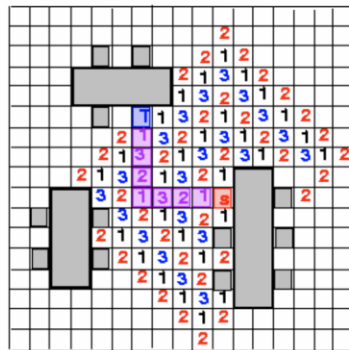
How can we optimize this problem?

## Midterm Goal:

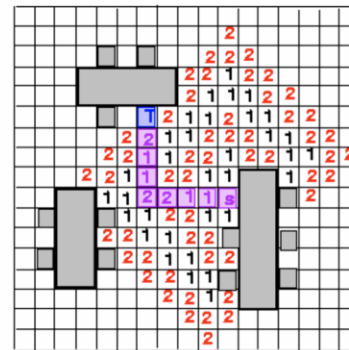
Try to design a 2-end router from both algorithm level and circuit level to reach a fast routing with less memory(area) penalty. i.e. Consider hardware awareness algorithm and some skill you learnt from Lab01~Lab06.

### Reducing Memory Requirement

- Akers's Observations (1967)
  - Adjacent labels for  $k$  are either  $k-1$  or  $k+1$ .
  - Want a labeling scheme such that each label has its preceding label different from its succeeding label.
- Way 1: coding sequence 1, 2, 3, 1, 2, 3, ...; states: 1, 2, 3, *empty*, *blocked* (3 bits required)
- Way 2: coding sequence 1, 1, 2, 2, 1, 1, 2, 2, ...; states: 1, 2, *empty*, *blocked* (need only 2 bits)



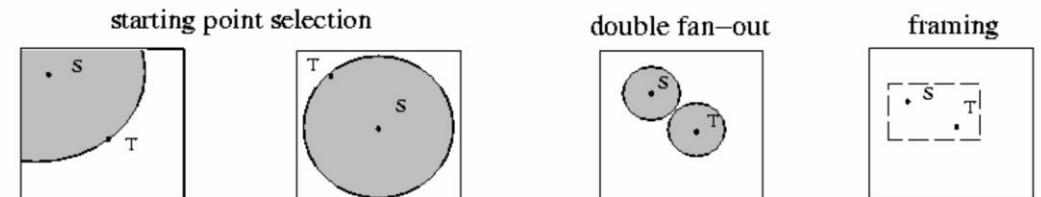
Sequence: 1, 2, 3, 1, 2, 3, ...



Sequence: 1, 1, 2, 2, 1, 1, 2, 2, ...

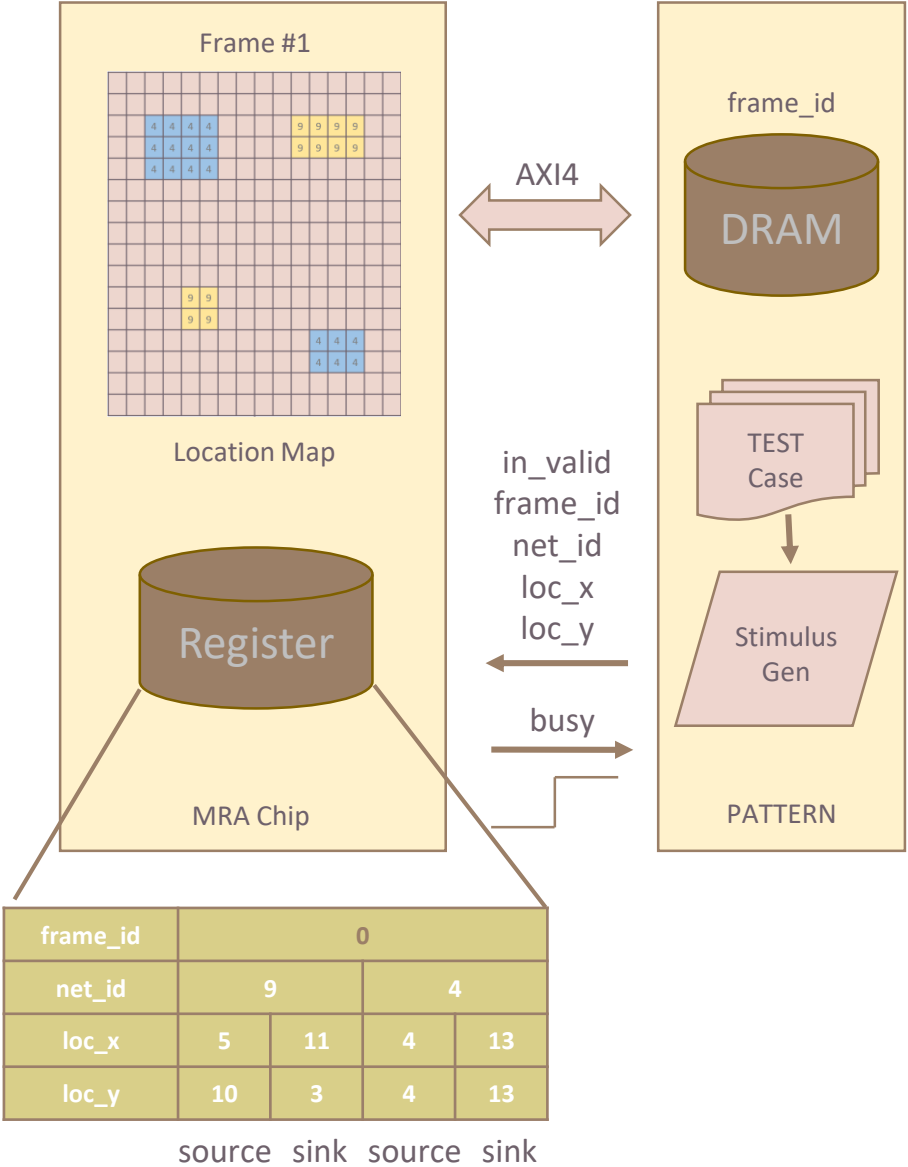
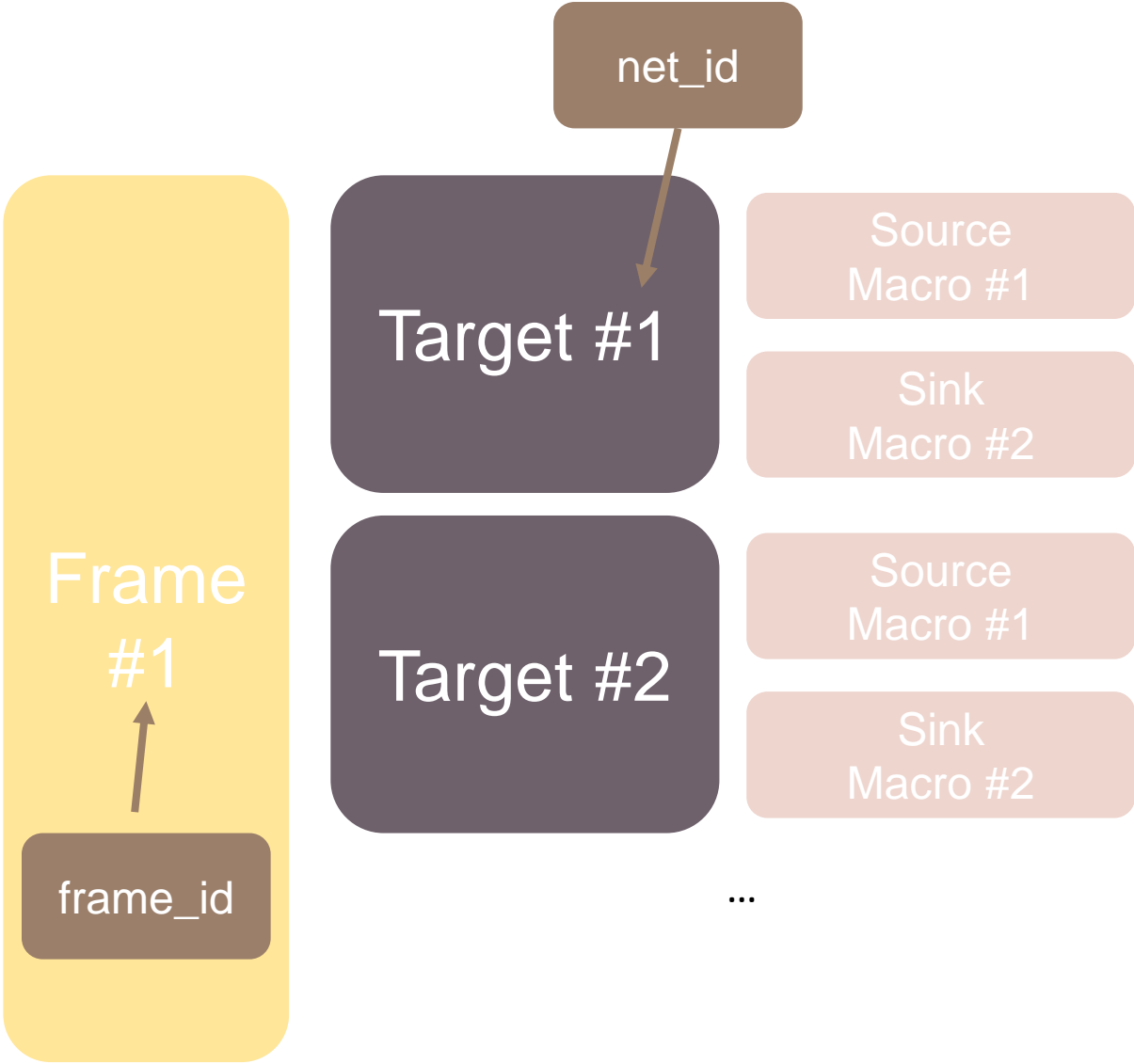
### Reducing Running Time

- Starting point selection: Choose the point farthest from the center of the grid as the starting point.
- Double fan-out: Propagate waves from both the source and the target cells.
- Framing: Search inside a rectangle area 10--20% larger than the bounding box containing the source and target.
  - Need to enlarge the rectangle and redo if the search fails.

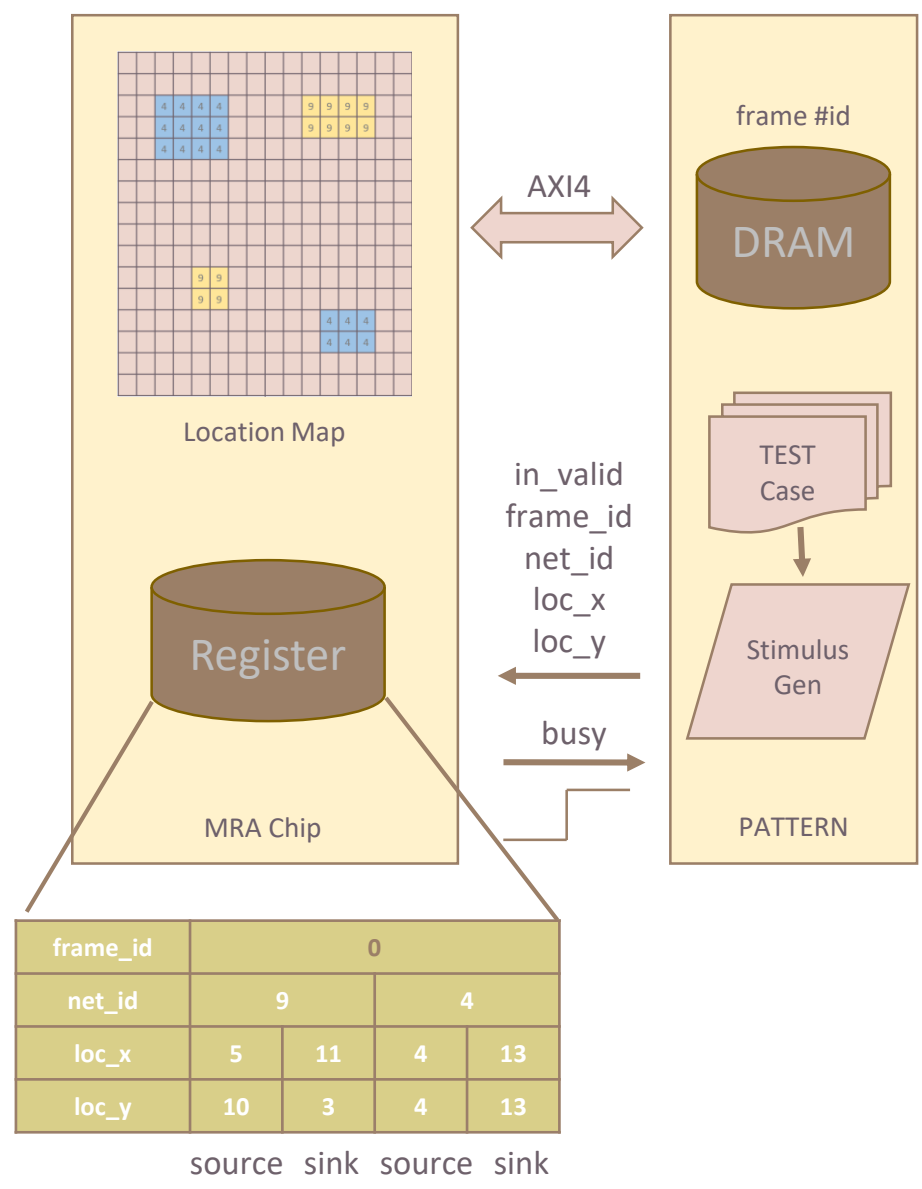


# Problem Definition

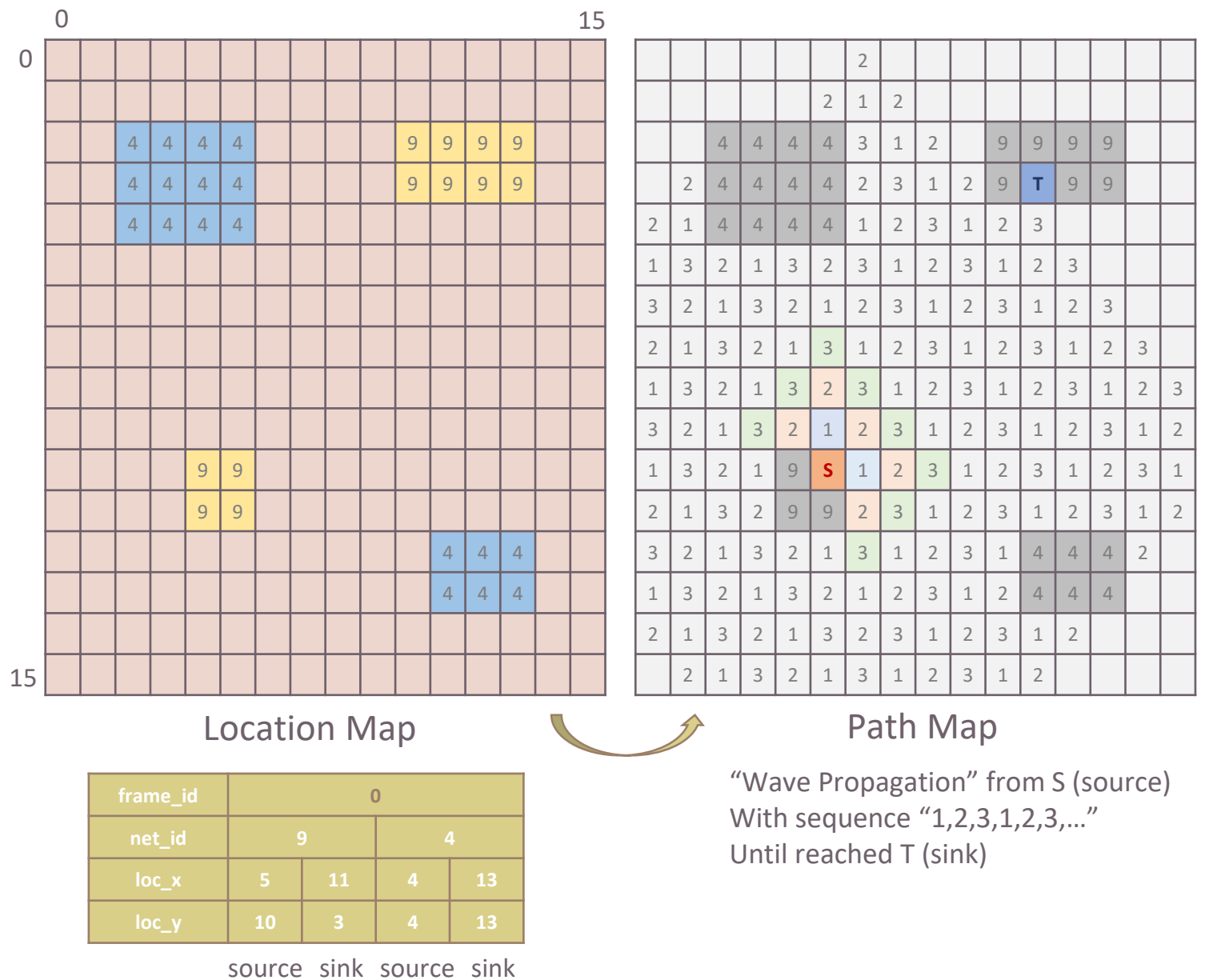
Description of object



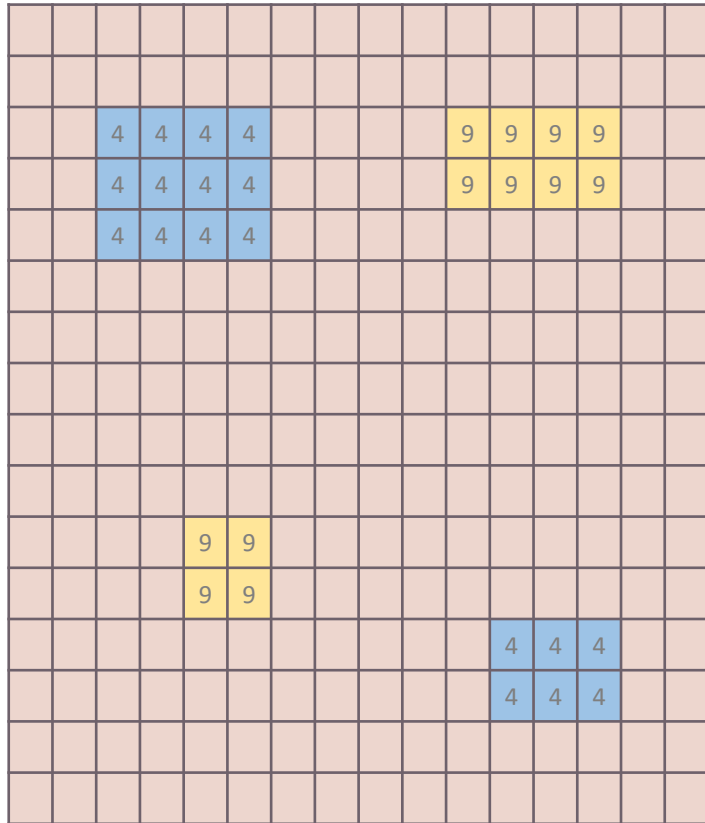
# Step.0 Get Input & Fetch DRAM



# Step.1 Filling Path Map



## Retrace Path Map

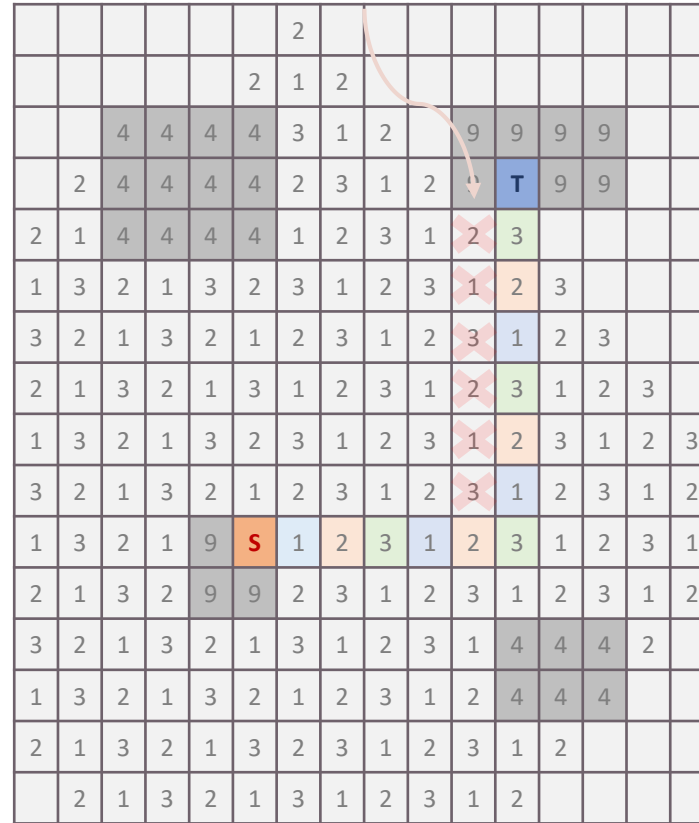


## Location Map

## Retrace Priority

- 1) Down ( $y+1$ )
- 2) UP ( $y-1$ )
- 3) Right ( $x+1$ )
- 4) Left ( $x-1$ )

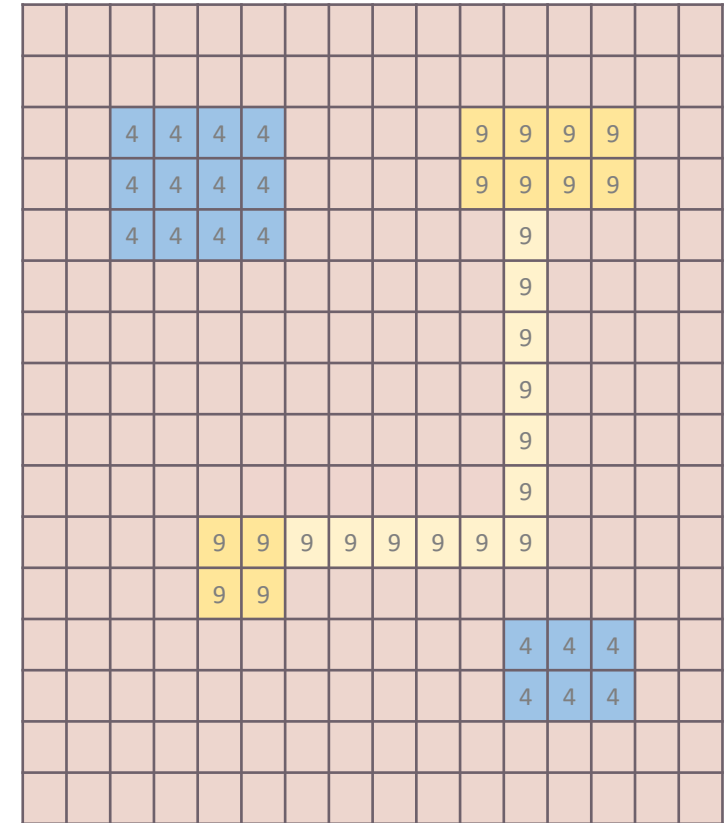
Since Down priority is higher than Left.



## Path Map

“Retrace” from T (sink)  
Based on **retrace priority** to discover  
right back-trace sequence, i.e.  
“3,2,1,3,2,1...” or “2,1,3,2,1,3...” or  
“1,3,2,1,3,2...” Until reached S (source)

## Update Location Map

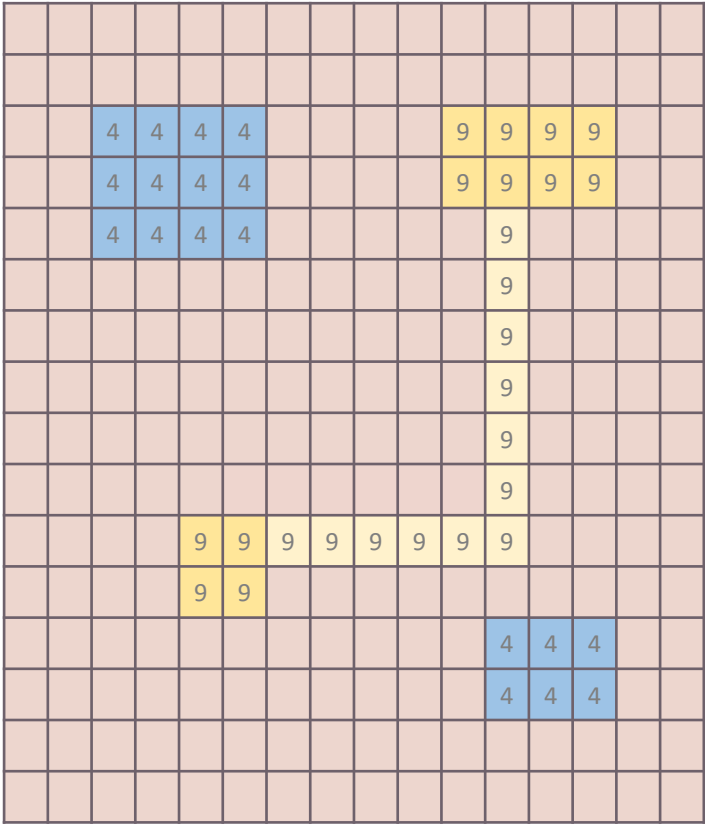


Pattern will check if source can reach sink or not. (connectivity check)

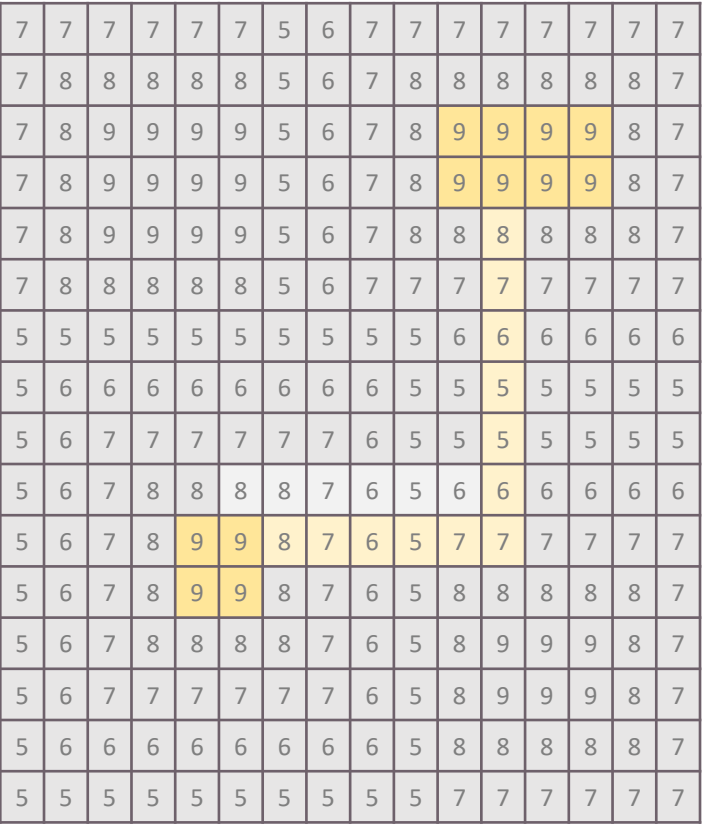
Step.4  
Calculate Path Cost



Repeat  
Step.1 ~ Step.4



Location Map



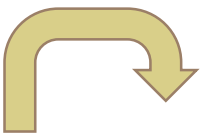
Weight Map (store in DRAM)

frame_id	0			
net_id	9		4	
loc_x	5	11	4	13
loc_y	10	3	4	13

source sink

Do it again for another target  
until all targets find the path.

Accumulate weight on the path from  
Source to Sink.  
Ex:  $8+7+6+5+7+7+6+5+5+6+7+8 = 77$   
Record it as target #1 path cost.  
(Total cost += target #1 cost )



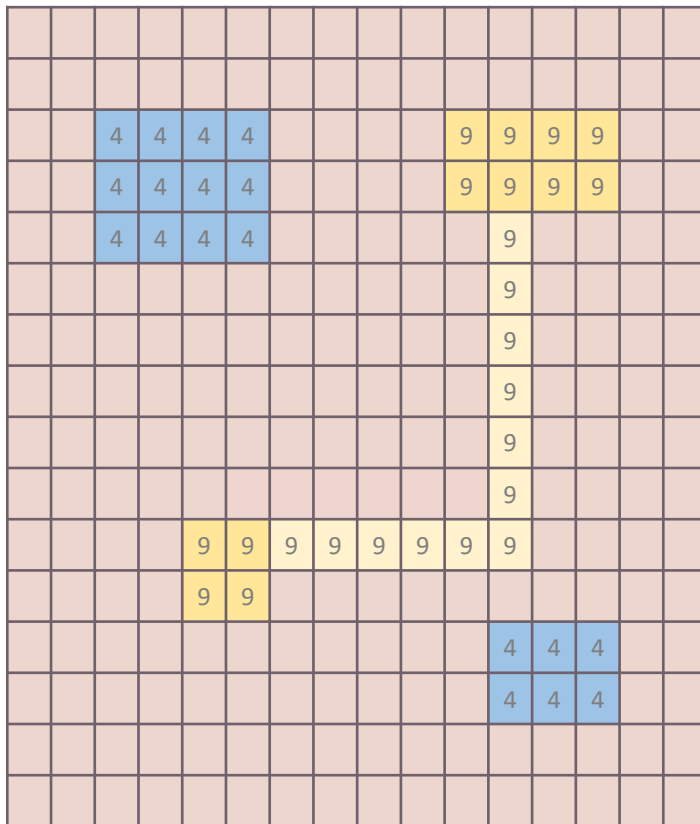
Repeat  
Step.1 ~ Step.4

## Step.1 Filling Path Map

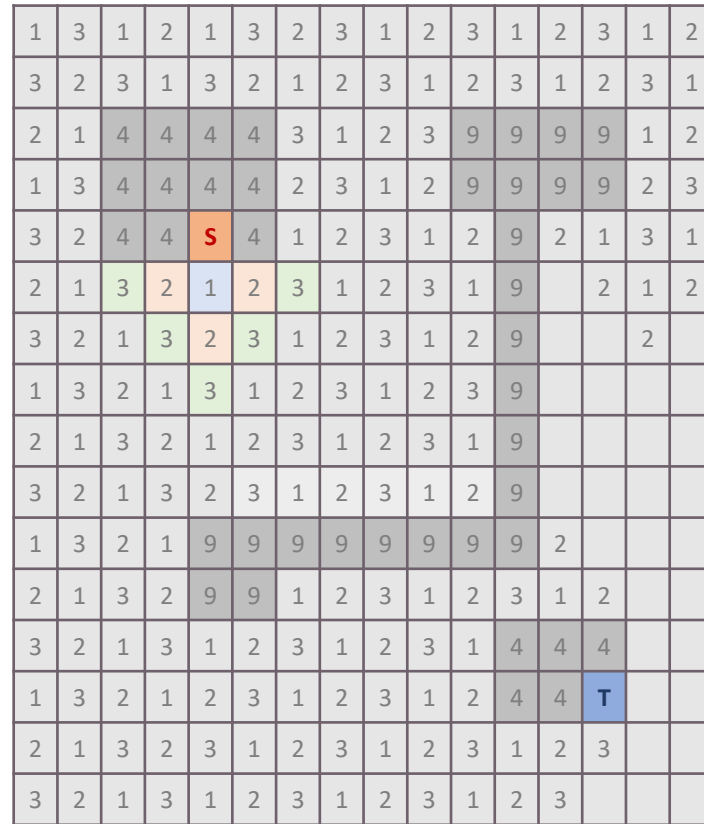
frame_id	0			
net_id	9		4	
loc_x	5	11	4	13
loc_y	10	3	4	13

source sink

Do it again for another target  
until all targets find the path.



Location Map

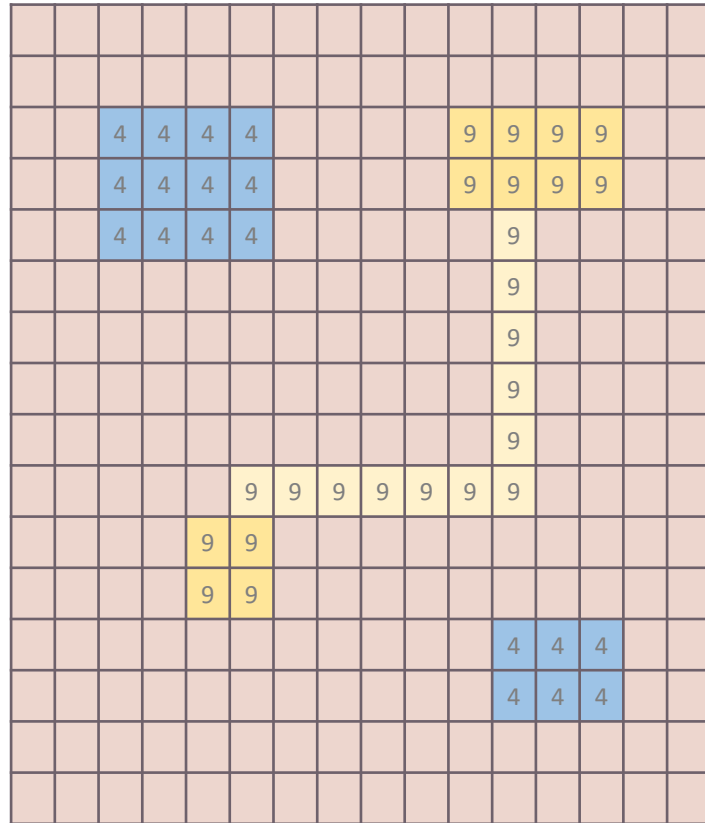


Path Map

“Wave Propagation” from S (source)  
With sequence “1,2,3,1,2,3,...”  
Until reached T (sink)



## Step.2 Retrace Path Map



Location Map



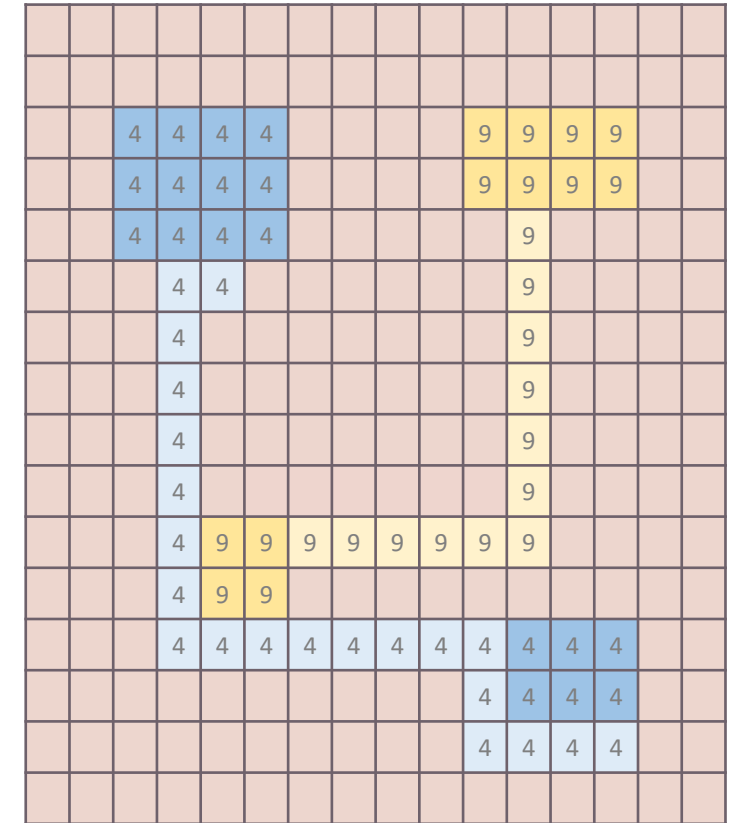
Path Map

### Retrace Priority

- 1) Down (y+1)
- 2) UP (y-1)
- 3) Right (x+1)
- 4) Left (x-1)

“Retrace” from T (sink)  
Based on **retrace priority** to discover right back-trace sequence, i.e.  
“3,2,1,3,2,1...” or “2,1,3,2,1,3...” or  
“1,3,2,1,3,2...” Until reached S (source)

## Step.3 Update Location Map



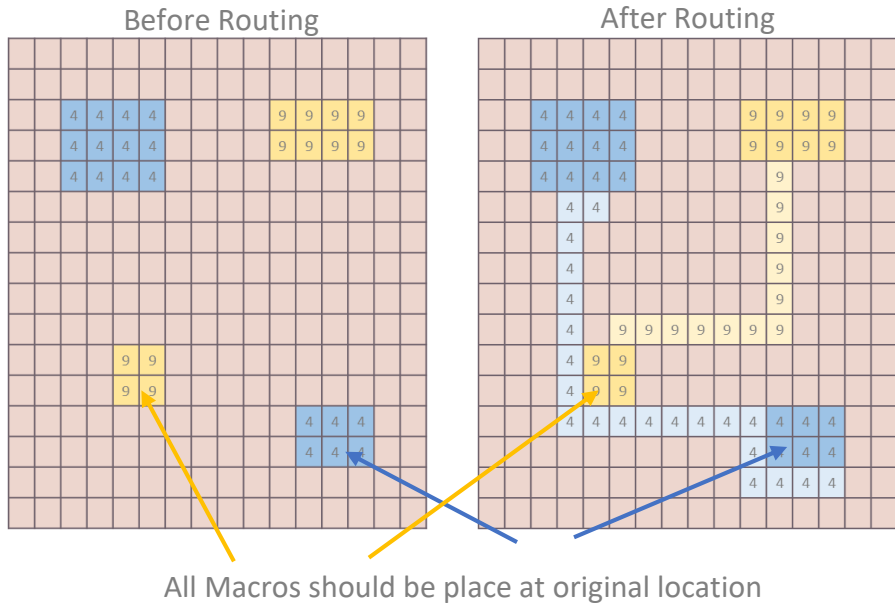
Pattern will check if source can reach sink or not. (connectivity check)



# PATTERN Design by You

## Functional Check

### (1) Consistent Check

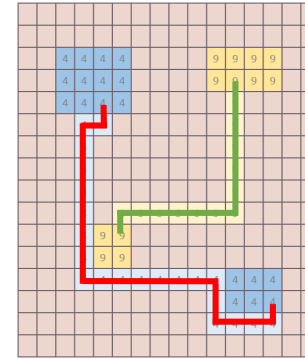


### Pattern will check : (1) -> (2) -> (3)

- (1) Check original target is consistent with original location map store in DRAM. **(consistent check)**
- (2) Check routed location map store in DRAM to see if source can reach sink or not. **(connectivity check)**
- (3) Check weight output is correct with your routed location map in DRAM **(cost check)**

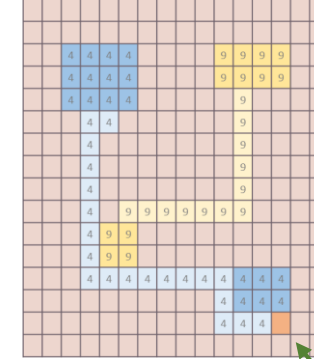
### (2) Connectivity Check

✓ Correct



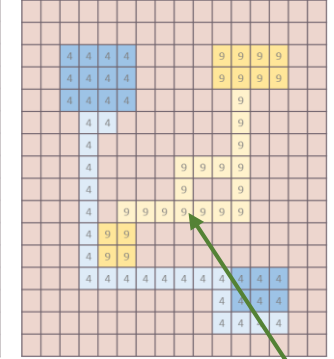
Only one path  
from S to T

✗ Fail



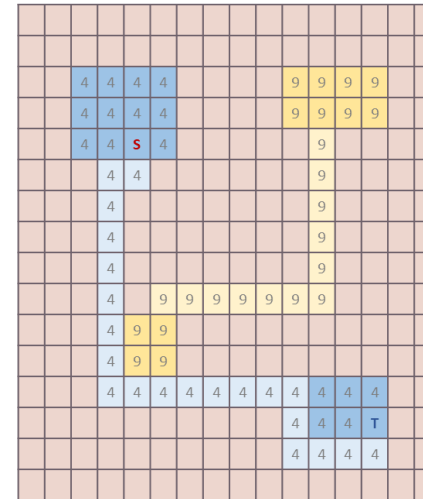
Can not find path from S to T

✗ Fail



Multiple Path is forbidden

### (3) Cost Check



7	7	7	7	7	5	6	7	7	7	7	7	7	7	7	7
7	8	8	8	8	5	6	7	8	8	8	8	8	7		
7	8	9	9	9	9	5	6	7	8	9	9	9	8	7	
7	8	9	9	9	9	5	6	7	8	8	8	8	8	7	
7	8	9	9	9	9	5	6	7	8	8	8	8	8	7	
7	8	8	8	8	8	5	6	7	7	7	7	7	7	7	
5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6
5	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5
5	6	7	7	7	7	7	6	5	5	5	5	5	5	5	5
5	6	7	8	8	8	8	7	6	5	6	6	6	6	6	6
5	6	7	8	9	9	8	7	6	5	7	7	7	7	7	7
5	6	7	8	9	9	8	7	6	5	8	8	8	8	8	7
5	6	7	8	8	8	8	7	6	5	8	9	9	9	8	7
5	6	7	7	7	7	7	7	6	5	8	9	9	9	8	7
5	6	6	6	6	6	6	6	6	5	8	8	8	8	8	7
5	5	5	5	5	5	5	5	5	5	7	7	7	7	7	7

✓ Weight: 156  
Golden: 156

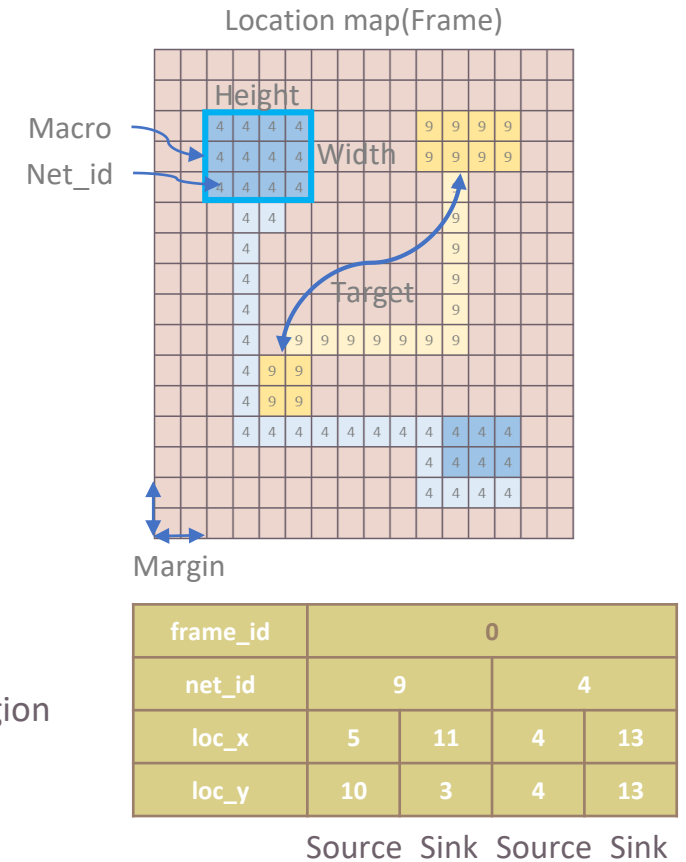
✗ Weight: 123  
Golden: 156

Calculating golden total weight in pattern,  
compare to MRA.v output

# SPEC of Routing Map

## Specification:

- (1) One **“Location map”** called a **“Frame”**, with 64 by 64 4-bit array
- (2) Each **“Frame”** has its **“Frame\_id”**, identify the address store in DRAM
- (3) **“Location map”** would include several routing **“Target”**, # of targets would range in 1~15
- (4) Each **“Target”** owns its **“NET\_ID”**, **“NET\_ID”** would range from 1~15
- (5) One **“Target”** consisted of 2 **“Macro”**, one is **“Source”**, the other is **“Sink”**
- (6) **“Macro”** height and width would be ranged in 2~6 if # of targets less than 11, otherwise would be 2~4
- (7) **“Macro”** has one **“Terminal”**, which must be located at the outermost region of the **“Macro”**
- (8) Location of **“Terminal”** would be send by input **“loc\_x”** and **“loc\_y”**, fist is **“Source”**, followed by **“Sink”**
- (9) **“Location map”** record all **“Macro”** location, identified by a 4-bit value **“NET\_ID”**, while 0 represent empty region
- (10) **“Margin”** of **“Location map”** would be outermost 2 rows and 2 columns
- (11) **“Macro”** would never place at the **“Margin”** area
- (12) **“Target”** is routed means only one path is highlighted with **“NET\_ID”** from **“Source Terminal”** to **“Sink Terminal”** in **“Location map”**
- (13) **“Length”** means path grid number from **“Source Terminal”** to **“Sink Terminal”** exclusive itself when **“Target”** is routed
- (14) **“Length”** of each **“Target”** is limited with in 1000 units, i.e. over 1000 units case will be drop out
- (15) **“Weight”** means path weighted sum from **“Source Terminal”** to **“Sink Terminal”** exclusive itself when **“Target”** is routed
- (16) **“Location map”** routing success means all **“Target”** is routed in **“Location map”** (not unique solution)
- (17) **“Location map”** must be routing success with given approach
- (18) **“Cost”** of routing result means the accumulation **“Weight”** when **“Location map”** routing success



# Grading in Midterm

$$\textit{Score} = \textit{SampleCase}(20\%) + \textit{Functionality}(50\%) + \textit{Performacne}(30\%)$$

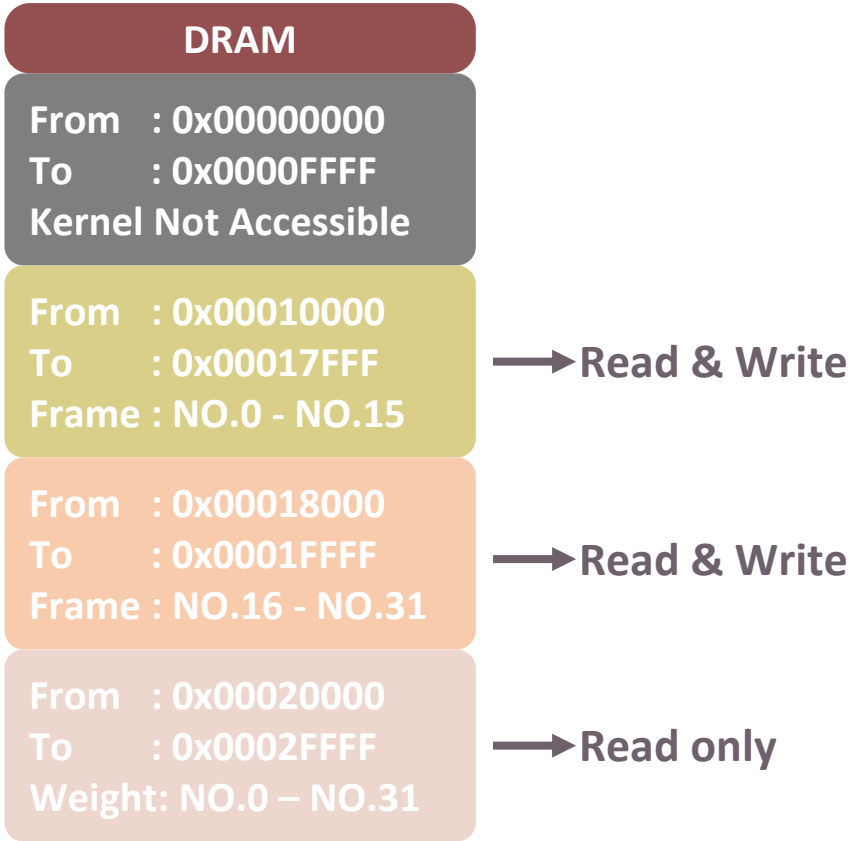
Finite State Machine

FSM + PATTERN

$$\textit{Performacne}(30\%) = \textit{Rank}(\textit{Latency} * \textit{Cycle Time})$$

# DRAM Provided by TA

## Address Mapping



## Memory Size:

Frame size =  $64 \times 64 \times 4 = 16,384$  bits = 2048 bytes

Weight size =  $64 \times 64 \times 4 = 16,384$  bits = 2048 bytes

Total = 32 Frames and 32 Weights

## Address Example:

Fra. No.0    0x0001\_0000 ~ 0x0001\_07FF  
Fra. No.1    0x0001\_0800 ~ 0x0001\_0FFF  
Fra. No.2    0x0001\_1000 ~ 0x0001\_17FF  
Fra. No.3    0x0001\_1800 ~ 0x0001\_1FFF

We. No.0    0x0002\_0000 ~ 0x0002\_07FF  
We. No.1    0x0002\_0800 ~ 0x0002\_0FFF  
We. No.2    0x0002\_1000 ~ 0x0002\_17FF  
We. No.3    0x0002\_1800 ~ 0x0002\_1FFF

# DRAM

Provided by TA

## Fetch DRAM in your Design

### DRAM

From : 0x00000000  
To : 0x0000FFFF  
Kernel Not Accessible

From : 0x00010000  
To : 0x00017FFF  
Frame : NO.0 - NO.15

→ Read & Write

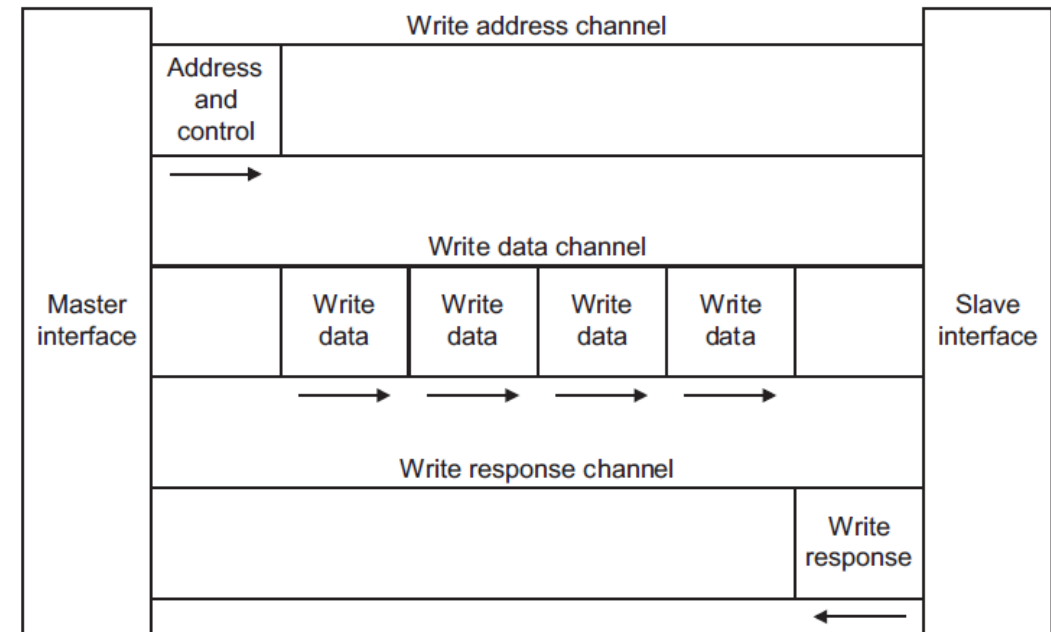
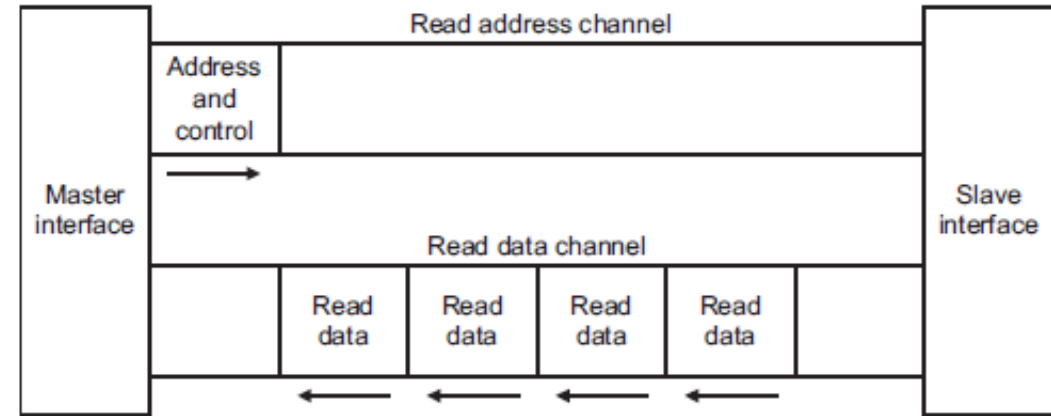
From : 0x00018000  
To : 0x0001FFFF  
Frame : NO.16 - NO.31

→ Read & Write

From : 0x00020000  
To : 0x0002FFFF  
Weight: NO.0 – NO.31

→ Read only

- AXI4 Protocol: Refer to Midterm Project AXI4

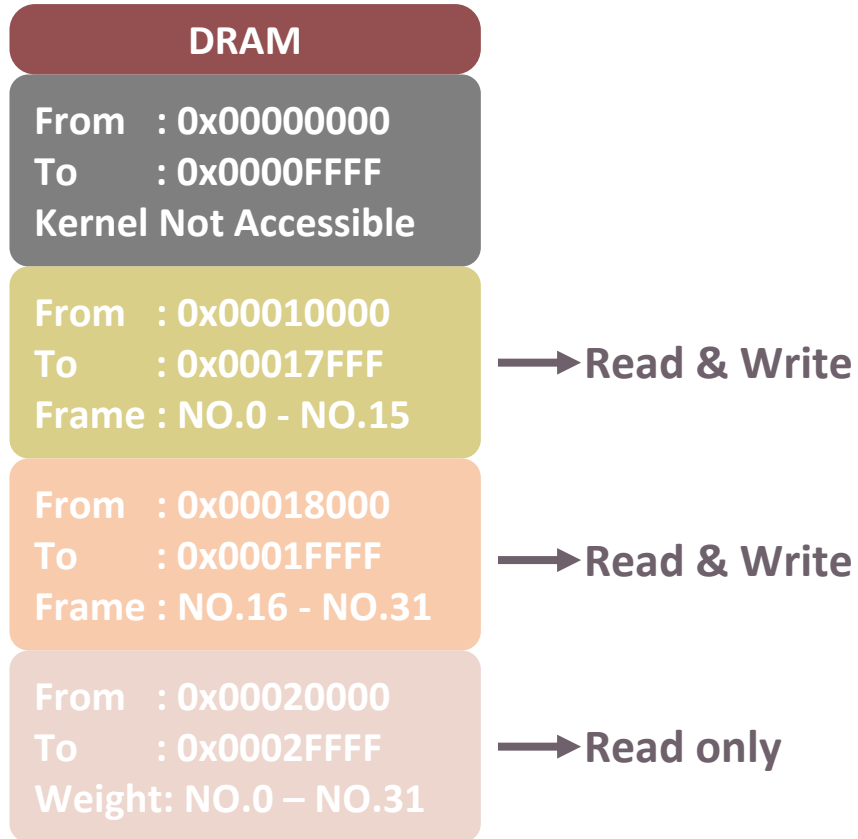


# DRAM

Provided by TA

## PATTERN Check Method

- How to directly access DRAM in pattern.



### Declare DRAM in PATTERN

```
pseudo_DRAM u_DRAM(  
    .clk(clk),  
    .rst_n(rst_n),  
  
    .awid_s_inf(awid_s_inf),  
    .awaddr_s_inf(awaddr_s_inf),  
    .awsizs_s_inf(awsizs_s_inf),  
    .awburst_s_inf(awburst_s_inf),  
    .awlen_s_inf(awlen_s_inf),  
    .awvalid_s_inf(awvalid_s_inf),  
    .awready_s_inf(awready_s_inf),  
  
    .wdata_s_inf(wdata_s_inf),  
    .wlast_s_inf(wlast_s_inf),  
    .wvalid_s_inf(wvalid_s_inf),  
    .wready_s_inf(wready_s_inf),  
  
    .bid_s_inf(bid_s_inf),  
    .bresp_s_inf(bresp_s_inf),  
    .bvalid_s_inf(bvalid_s_inf),  
    .bready_s_inf(bready_s_inf),  
);
```

**Note:** You should declare DRAM in pattern not design, and if your design contains DRAM unit, you will fail demo, i.e. you can only access DRAM data by AXI4 protocol in you design.

**Variable in pseudo\_DRAM.v** (do not modify)

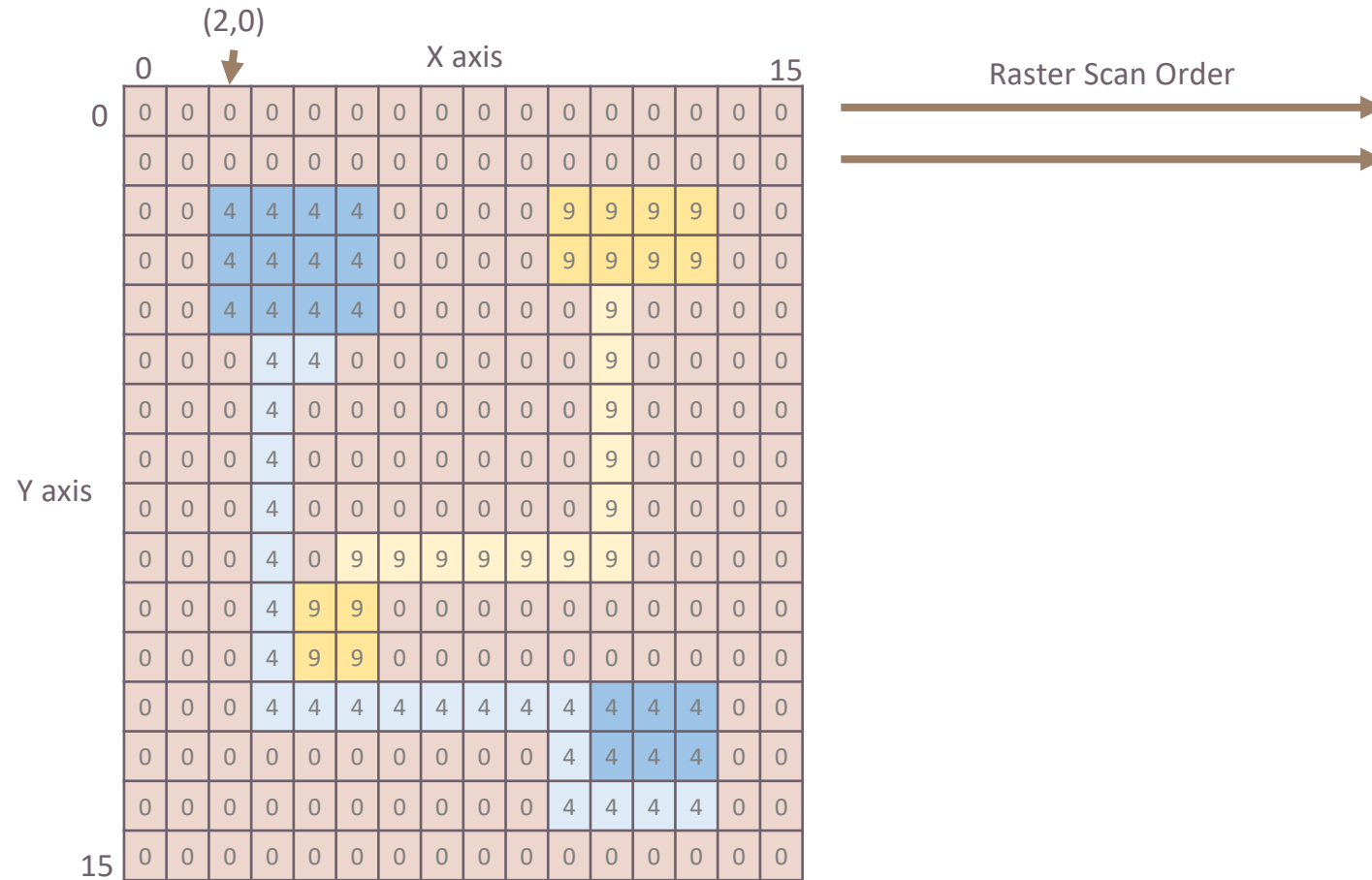
reg [7:0] DRAM\_r [0:196607]; (Address from 00000000 to 0002FFFF)

**Access submodule element** (Pattern may use it to check data store in DRAM is correct or not)

u\_DRAM.DRAM\_r[temp\_addr]



## Location Map



# DRAM

Provided by TA

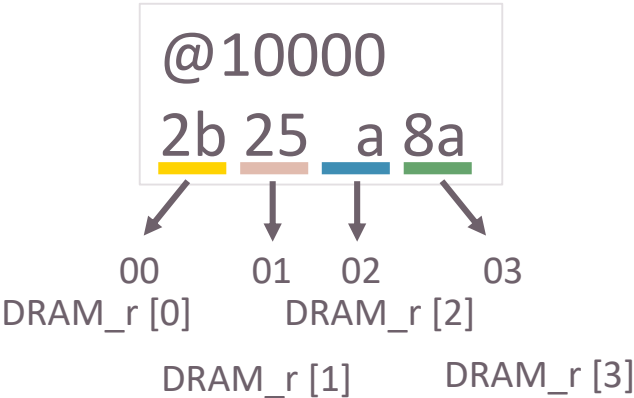
## .dat file example

../00\_TESTBED/DRAM/DRAM.dat

```
@10000
2b 25 a 8a
@10004
2b a8 29 34
@10008
bf 8a 5 79
@1000c
6d c5 29 a
@10010
73 a7 a 94
@10014
85 62 42 a4
@10018
ea dd 80 8b
@1001c
26 45 4a 1c
```

## Variable in pseudo\_DRAM.v

reg [7:0] DRAM\_r [0:196607]; (Address from 00000000 to 0002FFFF)



DRAM_r		
Address	[7:4]	[3:0]
[0]	2	B
[1]	2	5
[2]	0	A
[3]	8	A
[4]	2	B
[5]	A	8
[6]	2	9
[7]	3	4

Real Sequence in Location Map:  
(Raster Scan Order)

B 2 5 2 A 0 A 8 B 2 8 A 9 2 4 3 ...

# DRAM Provided by TA

## NOTE: YOU MAY USE

- You may modify the following part in ../00\_TESTBED/pseudo\_DRAM.v.

.dat file path      ➔ `parameter DRAM_p_r = "../00_TESTBED/DRAM/dram.dat";`

DRAM latency      ➔ `parameter DRAM_R_LAT = 90, DRAM_W_LAT = 100, MAX_WAIT_READY_CYCLE = 300;`

- If you want to refresh dram, you may use the following code.

```
$readmemh("../00_TESTBED/DRAM/dram.dat", u_DRAM.DRAM_r);
```

```
`ifdef FUNC
`define LAT_MAX 20
`define LAT_MIN 1
`endif
`ifdef PERF
`define LAT_MAX 20
`define LAT_MIN 1
`endif
```

# Sample Case Given by TA

3 Cases: Simple, Medium, Hard

**Input.txt, dram.dat** -> You can write a pattern to read this file and send to chip

**Map.txt, Output.txt, Weight.txt, \*.png** -> Provide for you to debug

DRAM

Picture

TEST\_CASE

For\_student > TEST\_CASE

For\_student > DRAM

For\_student > Picture

名稱

名稱

名稱

input\_0.txt

input\_1.txt

input\_2.txt

map\_0.txt

map\_1.txt

map\_2.txt

output\_0.txt

output\_1.txt

output\_2.txt

weight\_0.txt

weight\_1.txt

weight\_2.txt

dram\_0.dat

dram\_1.dat

dram\_2.dat

Original\_0\_0.png

Original\_0\_1.png

Original\_1\_0.png

Original\_1\_1.png

Original\_2\_0.png

Original\_2\_1.png

Routed\_0\_0.png

Routed\_0\_1.png

Routed\_1\_0.png

Routed\_1\_1.png

Routed\_2\_0.png

Routed\_2\_1.png

**Note:** TA may gen more data to test the functionality. Sample case is aim to make you easier understand this router problem.

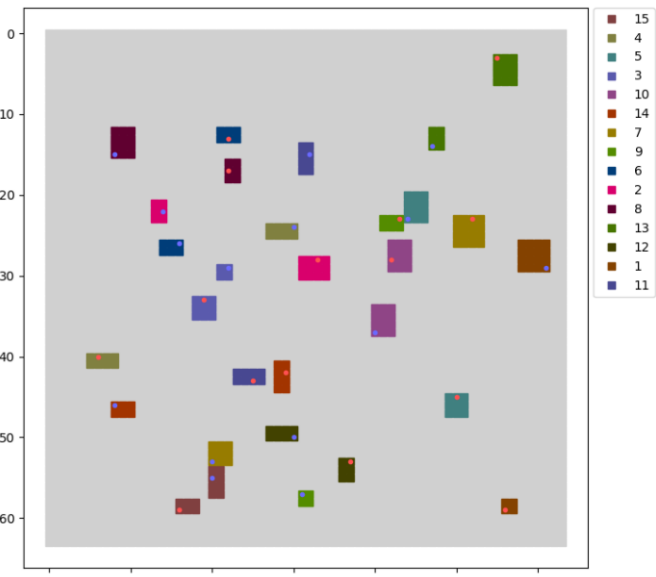
# Sample Case

## File Description

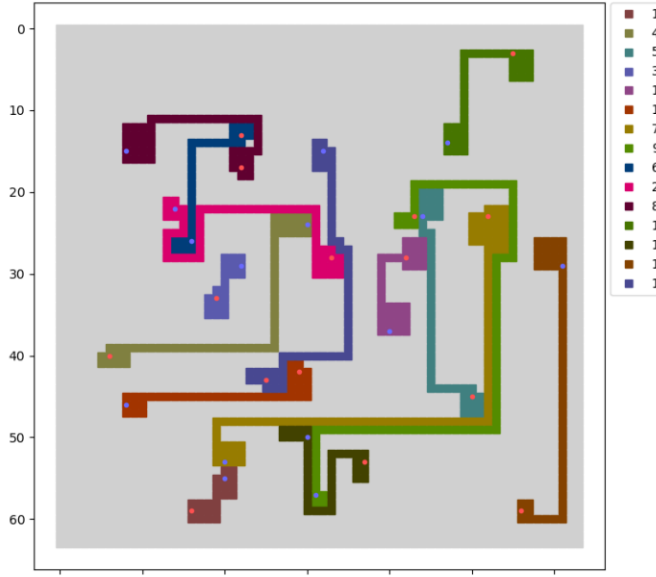
## Input data (input.txt)

1	# Pattern
0 15	No. Pattern / # Macro
15	Net id (target 1)
16 59	Source loc_x / Source loc_y
20 55	Sink loc_x / Sink loc_y
4	Net id (target 2)
6 40	Source loc_x / Source loc_y
30 24	Sink loc_x / Sink loc_y
5	Net id (target 3)
50 45	Source loc_x / Source loc_y
44 23	Sink loc_x / Sink loc_y
3	
19 33	...
22 29	
10	
42 28	
40 37	
14	
29 42	
8 46	

Original map in DRAM



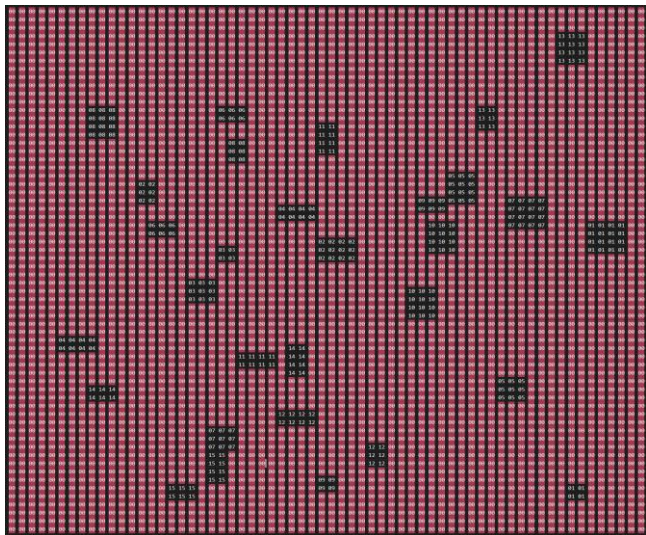
Routed map in DRAM



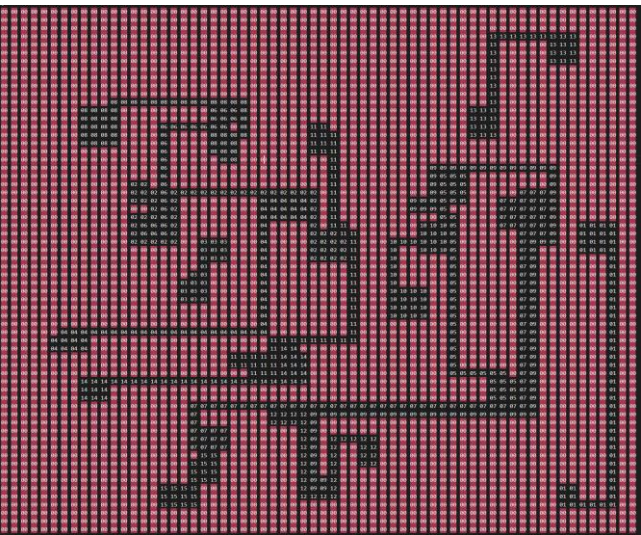
DRAM data  
Map / Weight (dram.dat)

```
@10000
00 00 00 00
@10004
00 00 00 00
@10008
00 00 00 00
@1000c
00 00 00 00
@10010
00 00 00 00
@10014
00 00 00 00
@10018
00 00 00 00
@1001c
00 00 00 00
@10020
00 00 00 00
@10024
00 00 00 00
@10028
00 00 00 00
```

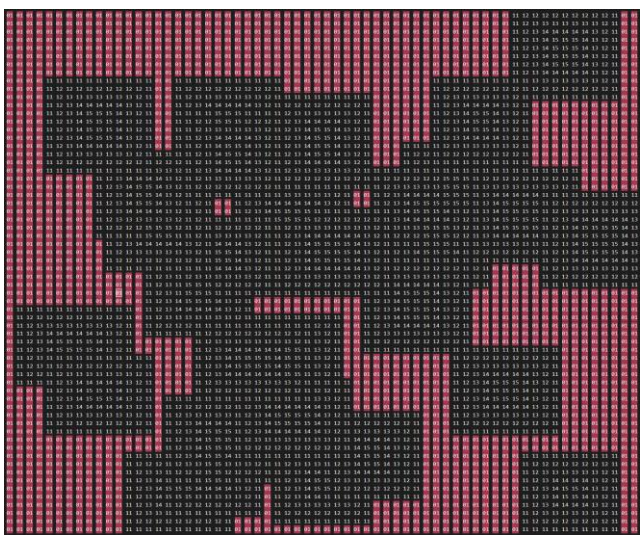
Location map in DRAM (map.txt)



Routed map in DRAM (output.txt)



Weight data in DRAM (weight.txt)



# Overall system block

