NCTU-EE IC LAB – Spring 2024

Midterm Project

Design: Maze Router Accelerator (MRA)

Data Preparation

1. Extract test data from TA's directory:

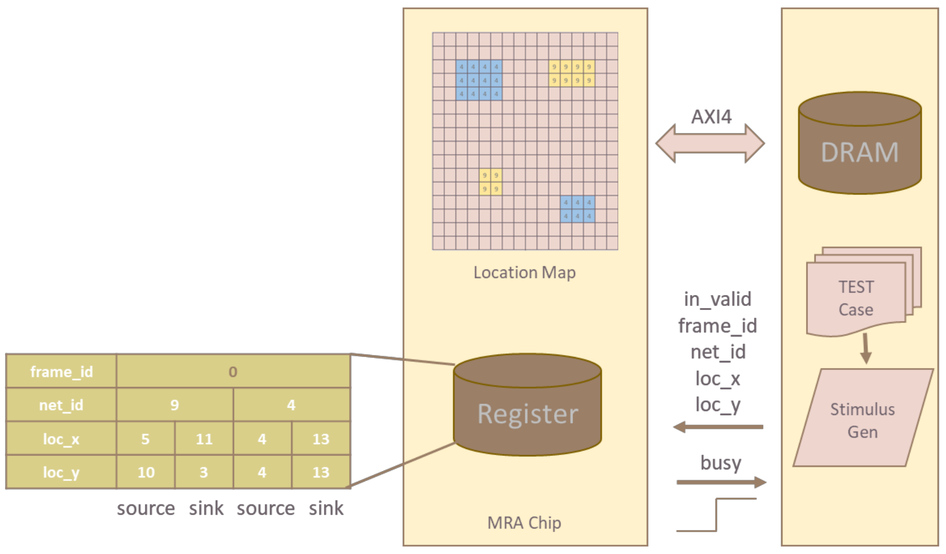
```
% tar xvf ~iclabTA01/Midterm_Project.tar
```

Design Description

Wire routing, often referred to simply as "routing", is a pivotal stage in IC (Integrated Circuit) design. Following the placement phase, the routing process introduces the necessary wires to connect the components in accordance with the IC's design rules. All routing tools share a fundamental objective: they begin with predefined polygons representing terminals on cells. Each polygon is linked to a specific net, either by name or number. The router's main responsibility is to generate geometries that ensure terminals linked to the same net are connected. At the same time, terminals from different nets should remain unconnected, while adhering to all design guidelines.

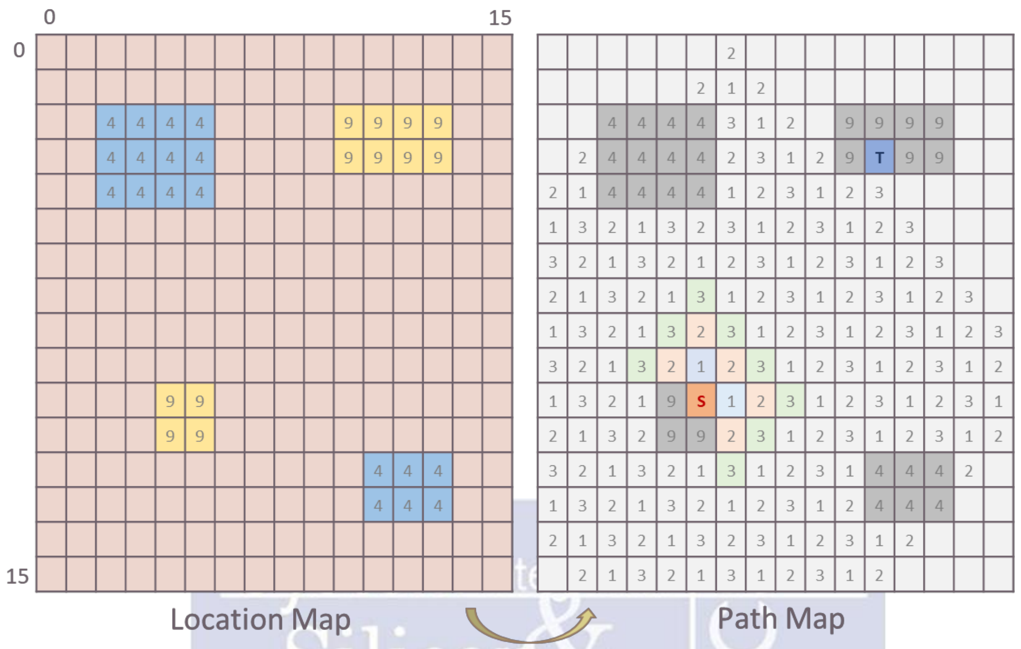
In this project, you'll be developing a robust router called the "Maze Router". This router will handle multiple un-routed targets stored in DRAM. Your primary objective is to ensure that all terminals linked to a specific net are interconnected. Let's delve deeper into how the Maze Router operates.

Here is step by step illustration:
Step 0: Get Input & Fetch DRAM



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

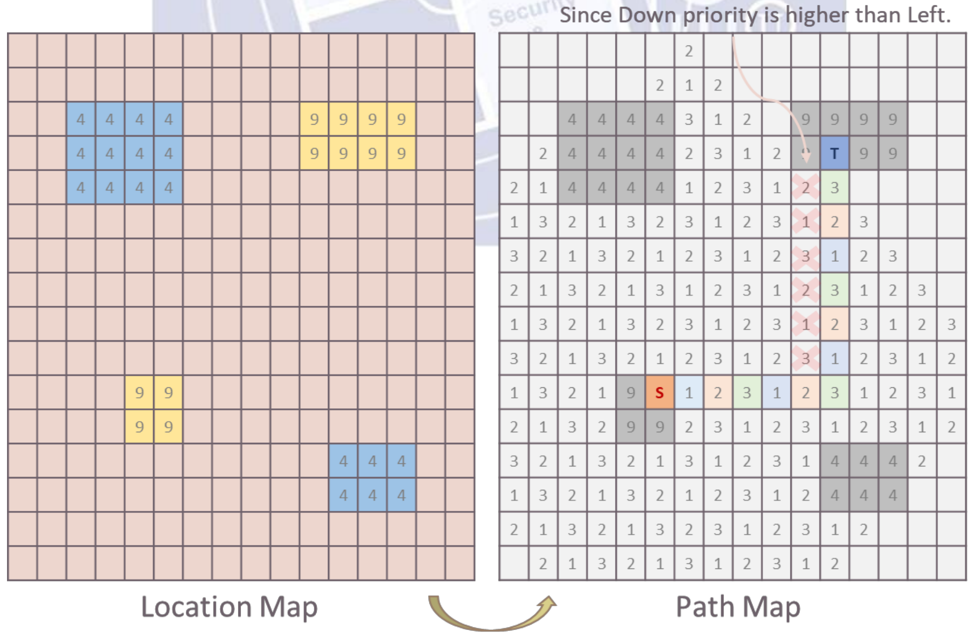
Step 1: Filling Path Map



"Wave Propagation" from S (source)
With sequence "1,2,3,1,2,3,…"
Until reached T (sink)

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

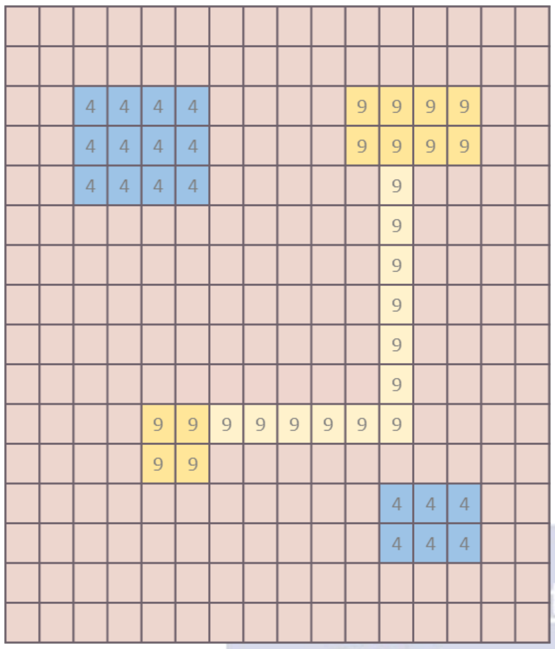
Step 2: Retrace Path Map



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



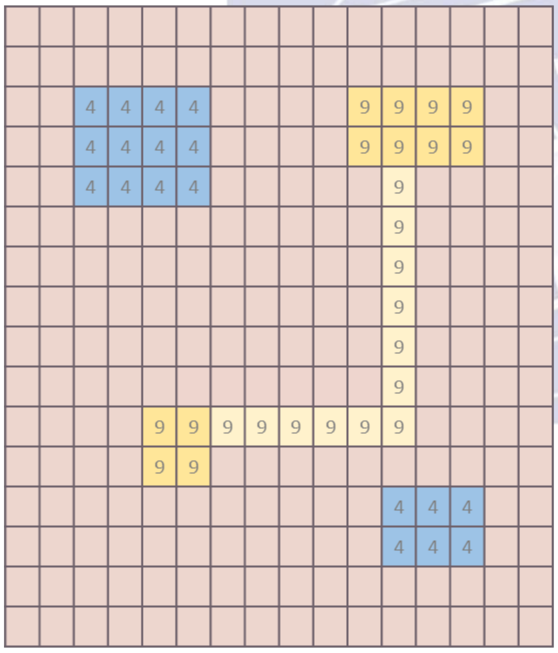
Loop: Until all targets routed

Repeat Step.1 ~ Step.4

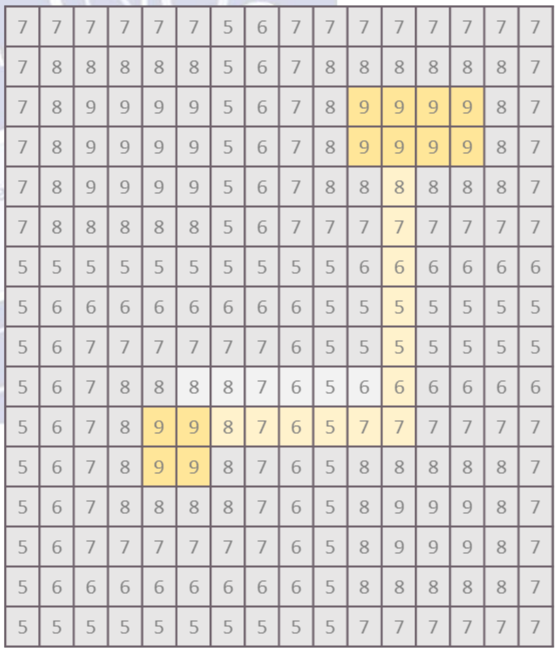
frame_id	0			
net_id	9		4	
loc_x	5	11	4	13
loc_y	10	3	4	13
	source	sink		

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

Step 4: Calculate Path Cost



Location Map

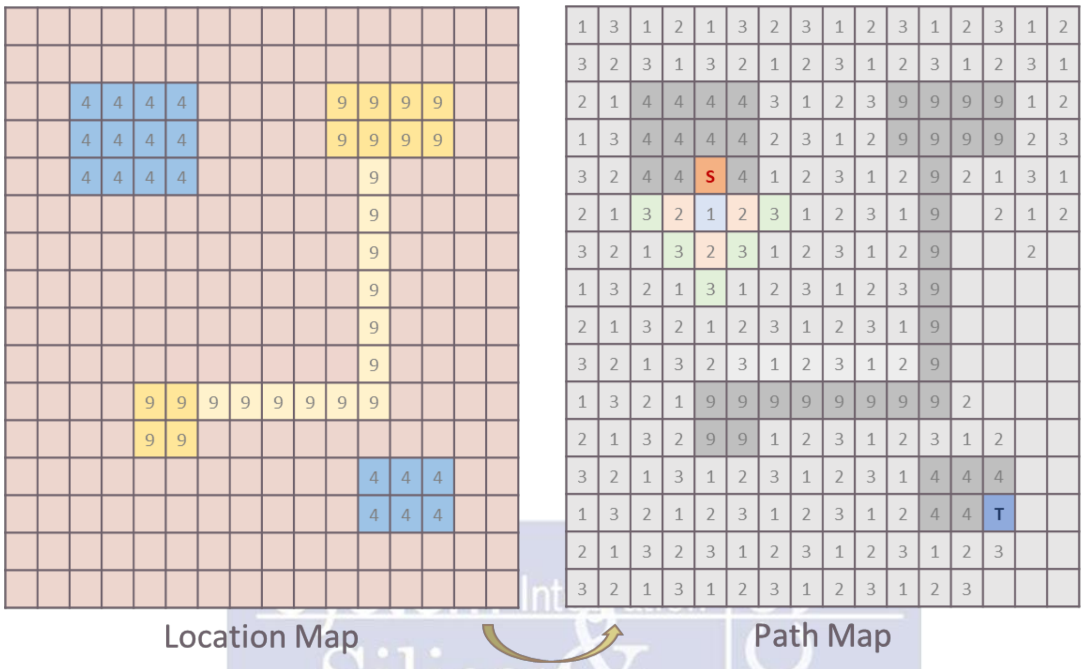


7	7	7	7	7	7	5	6	7	7	7	7	7	7	7	7
7	8	8	8	8	8	5	6	7	8	8	8	8	8	8	7
7	8	9	9	9	9	5	6	7	8	9	9	9	9	8	7
7	8	9	9	9	9	5	6	7	8	9	9	9	9	8	7
7	8	9	9	9	9	5	6	7	8	8	8	8	8	8	7
7	8	8	8	8	8	5	6	7	7	7	7	7	7	7	7
5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6
5	6	6	6	6	6	6	6	6	5	5	5	5	5	5	5
5	6	7	7	7	7	7	7	6	5	5	5	5	5	5	5
5	6	7	8	8	8	8	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 Map (store in DRAM)

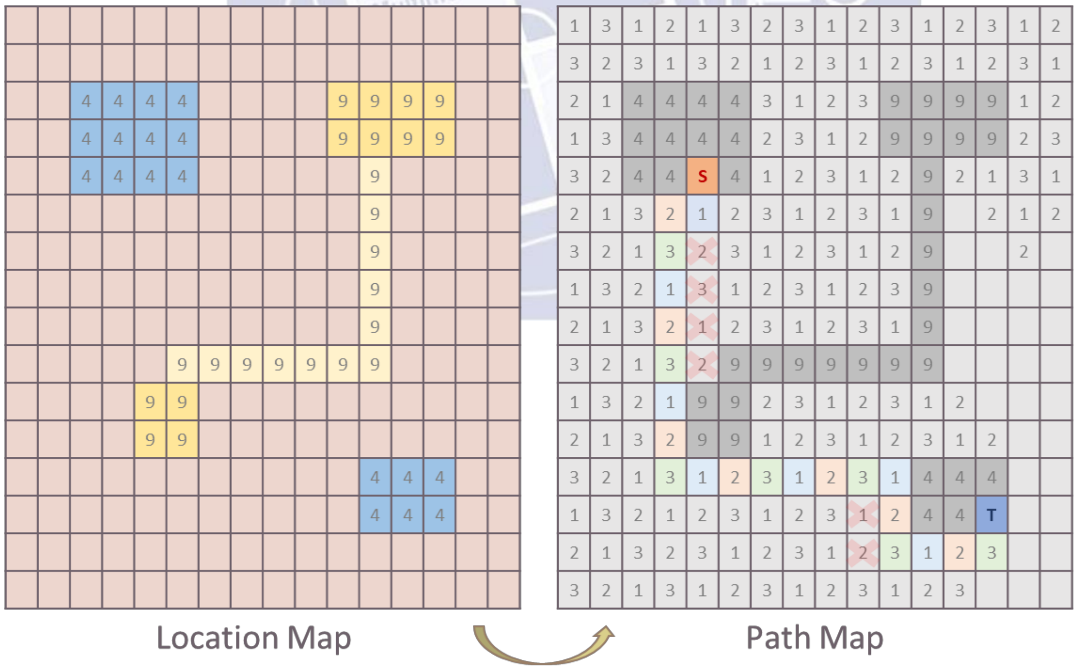
Accumulate weight on the path from Source to Sink.
Ex: 8+7+6+5+7+7+6+5+5+6+7+8 = 77
Record it as target #1 path cost.
(Total cost += target #1 cost)

Loop2 >> Step 1: Filling Path Map



"Wave Propagation" from S (source)
With sequence "1,2,3,1,2,3,…"
Until reached T (sink)

Loop2 >> Step 2: Retrace Path Map

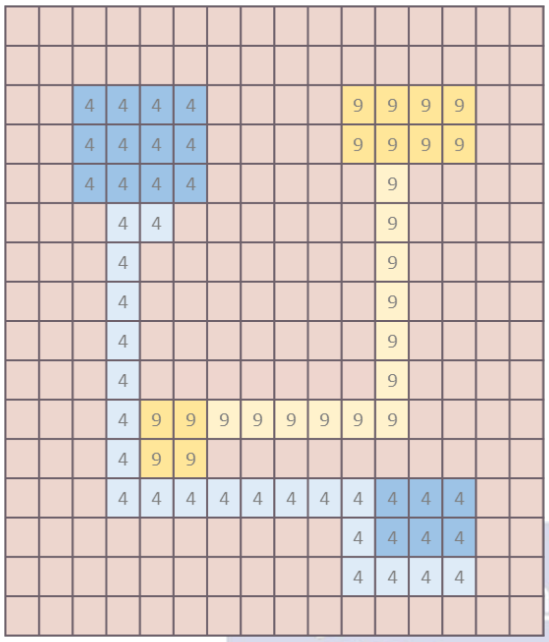


Retrace Priority

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)

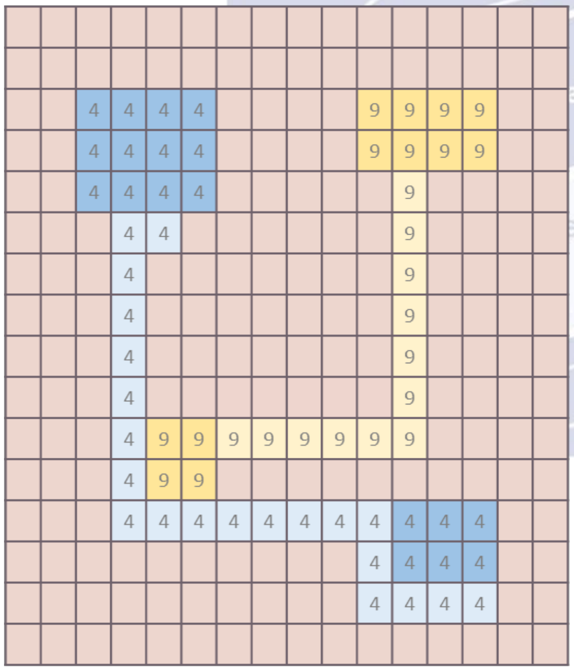
Loop2 >>Step 3: Update Location Map



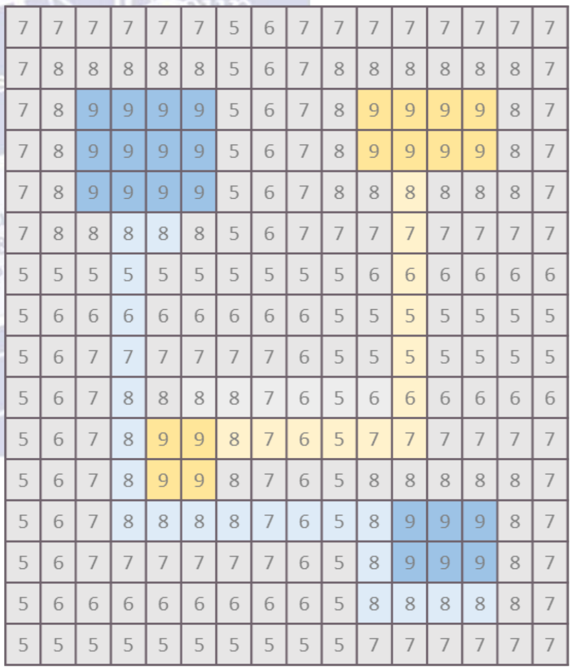
All targets find path

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

Loop2 >>Step 4: Calculate Path Cost



Location Map

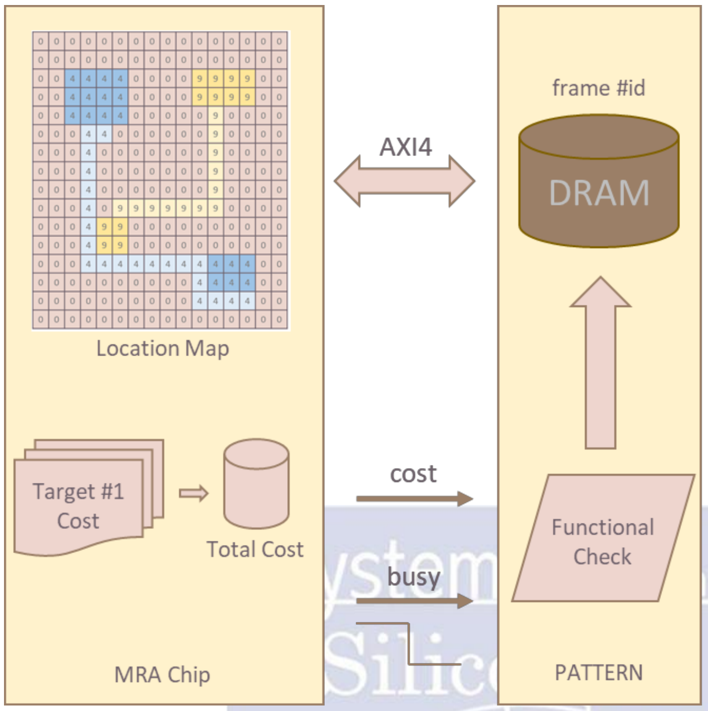


7	7	7	7	7	7	5	6	7	7	7	7	7	7	7	7
7	8	8	8	8	8	5	6	7	8	8	8	8	8	8	7
7	8	9	9	9	9	5	6	7	8	9	9	9	9	8	7
7	8	9	9	9	9	5	6	7	8	9	9	9	9	8	7
7	8	9	9	9	9	5	6	7	8	8	8	8	8	8	7
7	8	8	8	8	8	5	6	7	7	7	7	7	7	7	7
5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6
5	6	6	6	6	6	6	6	6	5	5	5	5	5	5	5
5	6	7	7	7	7	7	7	6	5	5	5	5	5	5	5
5	6	7	8	8	8	8	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 Map (store in DRAM)

Accumulate weight on the path from Source to Sink.
Ex:8+8+5+6+7+8+8+8+8+8+8+8+8+7+6+5+8+8+8+8+8+8 = 156
Record it as target #2 path cost.
(Total cost += target #2 cost)

Step 5: Write Back & Output



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)**

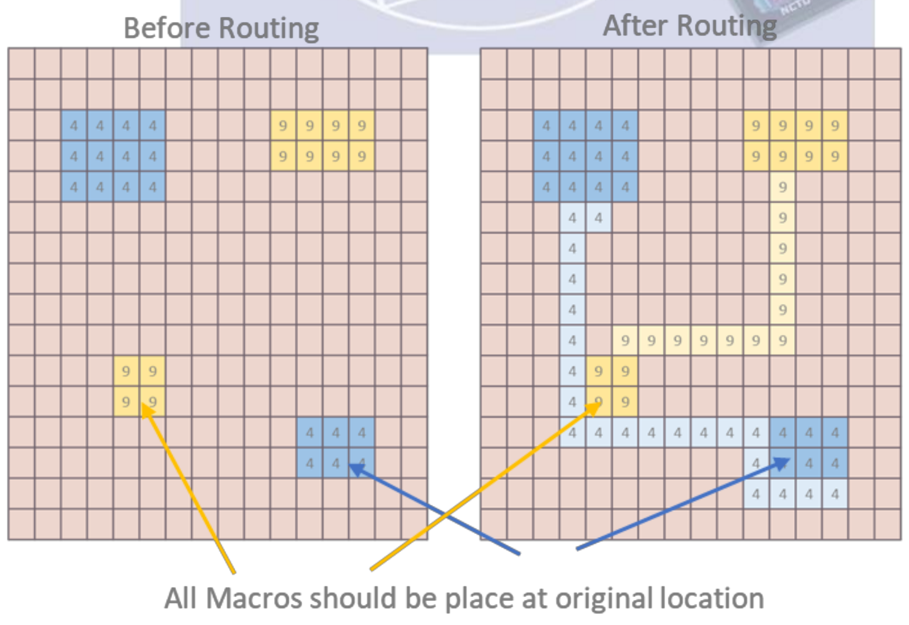
(3) Check weight output is correct with your routed location map in DRAM **(cost check)**

After processing one location map and all data is written back to DRAM, **busy** would be pull low so that pattern could check answer in DRAM. Note that routing result could be different with TA's method, only you reach the routing goal, you can pass.

Here is some check point in routed result:

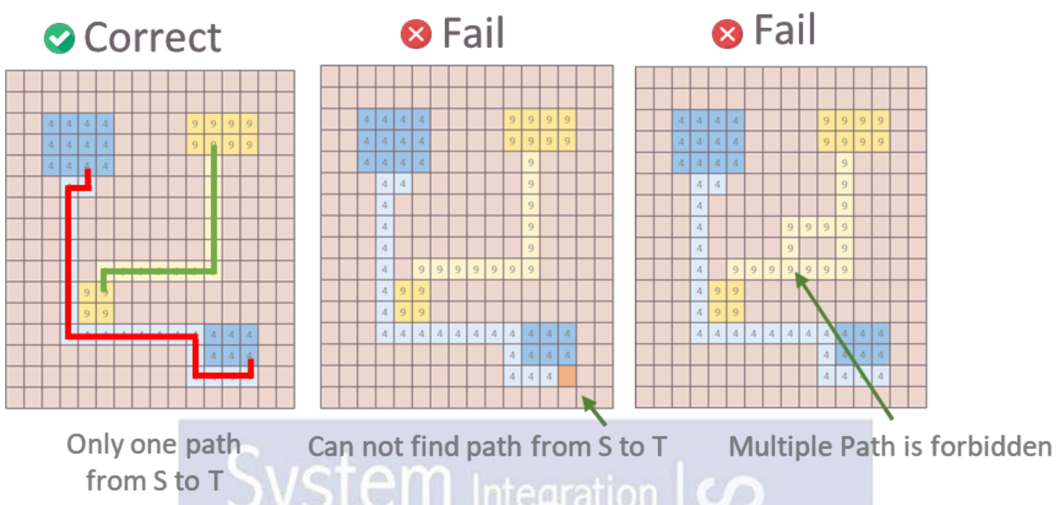
(1) Consistent Check:

Check original target is consistent with original location map store in DRAM.



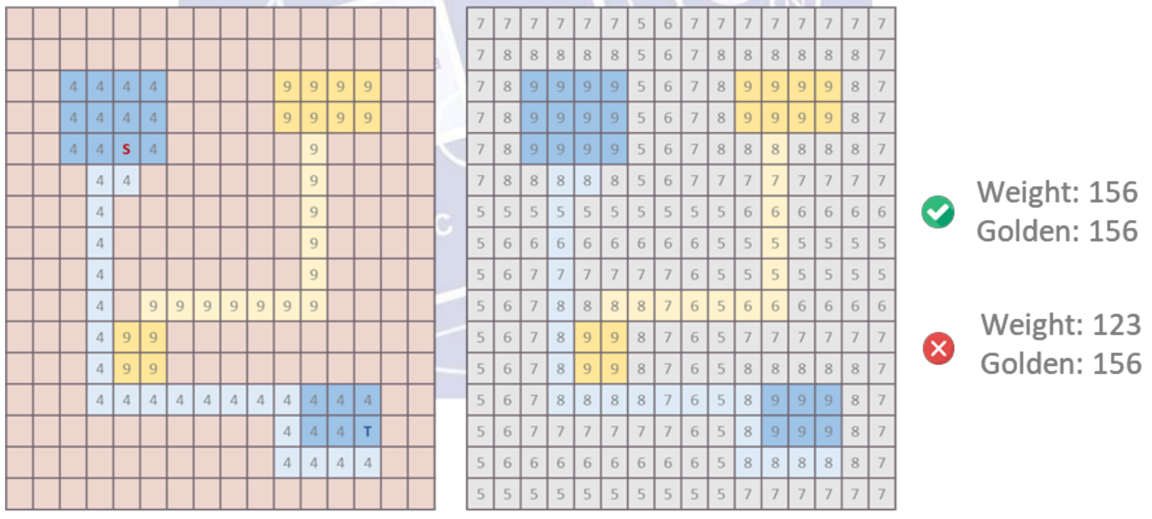
(2) Connectivity Check

Check routed location map store in DRAM to see if source can reach sink or not.


Correct
Fail
Fail
Only one path from S to T
Can not find path from S to T
Multiple Path is forbidden

(3) Cost Check

Check weight output is correct with your routed location map in DRAM



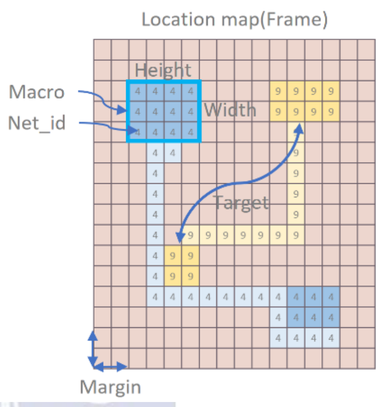
Calculating golden total weight in pattern, compare to MRA.v output

Note that only if (1) pass, then will check (2), then (3). Any of violation of rules will fail this Midterm.

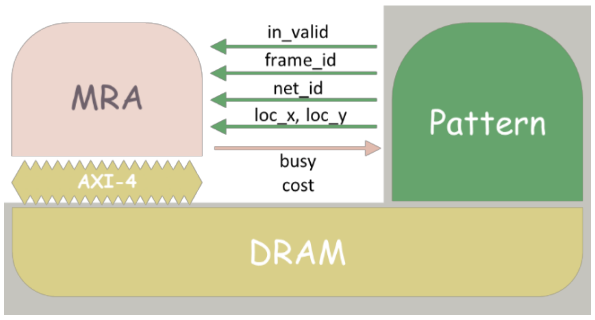
Now, we need to formulate this problem more specifically with well-define SPEC. Hence, let me introduce definition/SPEC precisely for in this midterm project.

SPEC for Test Case Generation

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 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 within 1000 units, i.e. no case over 1000 units
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



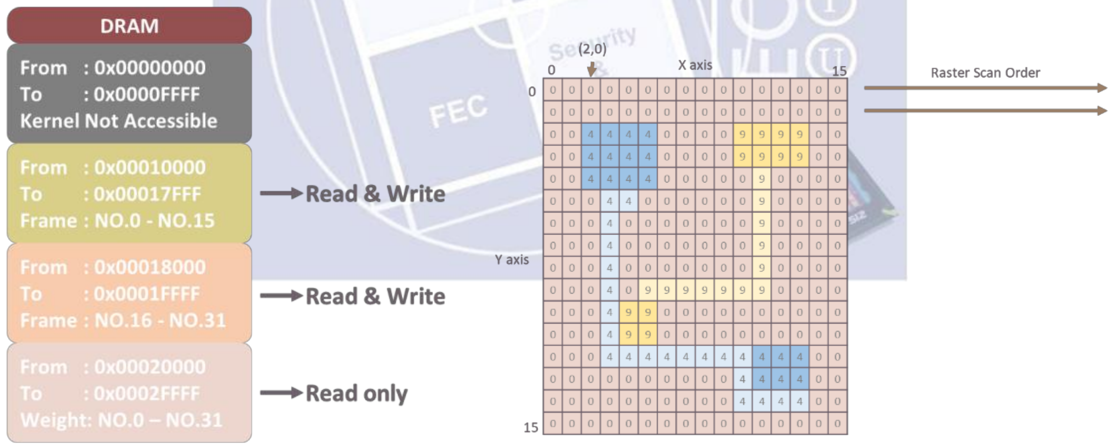
System Architecture



First, pattern will give the **frame_id** of Location Map and Weight in DRAM, also send the corresponding **net_id**, source location and sink location with the **loc_x**, **loc_y** in a row when **in_valid** is high. Note that the number of targets will not be provided. It depends on how many **net_id** you get when in_valid is high. Then, your chip should fetch frame data stored in DRAM via AXI-4. While your design is dealing with current task, **busy** should be high. After all processing is done, and all data is written back to DRAM, **busy** should be pull low so that pattern could check the answer in DRAM and output cost is consistent with golden one. On the other hand, pattern can directly access DRAM for data checking and overwriting without any constrain. In this project, all frames are 64 x 64, 4-bit net id location map, and all Weight Maps are 64 x 64, 4-bit integer. The DRAM behavior model (pseudo_DRAM.v) is provided by TA.

Memory Mapping

Example of data in memory:



Since all 16 frames are 64 x 64, 4-bit location map, we need 2048 bytes to store single frame. Each address in DRAM can store 1 byte (8 bits). Thus, the address of frame No.0 is range from 0x0001_0000 to 0x0001_07FF, No.1 is ranged from 0x0001_0800 to 0x0001_0FFF and so on. Every frame is **in raster scan order** in DRAM, each byte in DRAM represent 2 position value, where [3:0] means 1st element, and [7:4] means 2nd element.

Mapping Example:

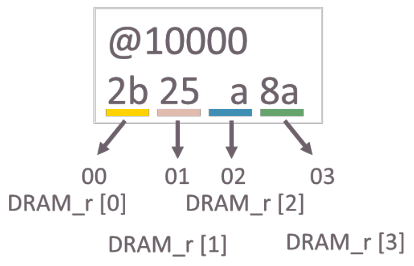
For example, 0x0001_0000 stores the (0, 0) [3:0] and (1,0) [7:4] value of Frame NO.0, 0x0001_0001 stores the (0, 2) [3:0] and (0, 3) [7:4] of Frame NO.0.

Variable in pseudo_DRAM.v

```
reg [7:0]    DRAM_r    [0:196607]; (Address from 00000000 to 0002FFFF)
```

```
@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
```

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 …

As for Weight Map, all of them are 64 x 64, 4-bit integer, and also **in raster scan order** in DRAM. The address of Weight No.0 is range from 0x0002_0000 to 0x0002_07FF, No.1 is ranged from 0x0002_0800 to 0x0002_0FFF and so on.

Fetch DRAM:

Given **frame_id** = 5, that means your need to fetch DRAM 0x0001_2800 ~ 0x0001_2FFF for Location map and 0x0002_2800 ~ 0x0002_2FFF for Weight map with AXI4 protocol.

Note that Memory Address is Byte Offset as shown in above example. Each address is mapped to single byte.

Please refer to Midterm_Project_Note.pdf for more detail information of DRAM.

You should **generate the SRAM** to store the required data. TA will check if you truly use memory, and if **NOT**, you will **Fail** the demo.

Control Interface

Inputs

1. You will receive 1 number **frame_id** [4:0] which is the index of both Location Map and Weight Map.
2. You will receive several numbers of **net_id** [3:0] which identify (1) NET_ID of target that stores in Location Map, (2) Routing Sequence. One **net_id** will show up 2 cycles.
3. Numbers targets would not be given. It is based on how many **net_id** you received when in_valid is high.
4. For each target, you will receive 2 numbers of **loc_x, loc_y** [5:0] which identify (1) Source location of target at 1st cycle, (2) Sink location of target at 2nd cycle.
5. Input signal **frame_id, net_id, loc_x, loc_y** are valid when **in_valid** is high, otherwise would be unknown value.
6. There is **only 1 reset** before the first pattern, thus, your design must be able to reset automatically.
7. All inputs will be changed at clock ***negative*** edge.
8. The next input pattern will come in 3 cycles after **busy** falls.

Name	Width	Functional Description
rst_n	1	Asynchronous reset and active-low
clk	1	Clock for MRA chip
in_valid	1	When in_valid is high, all the other three input is valid.
frame_id	5	Index of Location/Weight map, range in No. 0 ~ No. 31
net_id	4	NET ID that stores in Location map, one location map may contain 1~15 targets, so net_id will range in 1~15
loc_x	6	Indicate location (x coordinate) of either source or sink terminal for routing target, range in 0~63
loc_y	6	Indicate location (y coordinate) of either source or sink terminal for routing target, range in 0~63

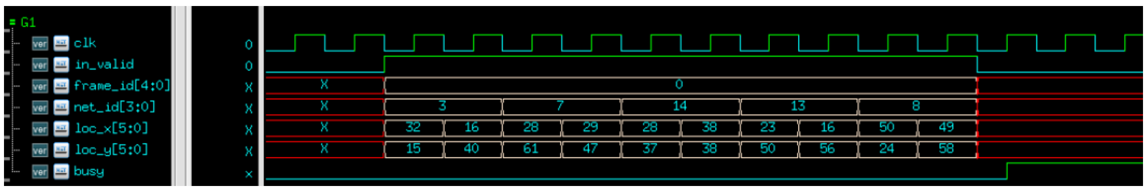


Fig 1. Input waveform example

Outputs

1. Your routed result should be written back to DRAM. After **busy is pulled low, pattern will check the correctness of the value inside DRAM.**
2. All outputs are synchronized at clock **positive** edge.
3. **busy** should be low after initial reset.
4. **busy** should not be raised when **in_valid** is high.
   The test pattern will check whether your data in DRAM and **cost** is correct or not at the first clock ***negative*** edge after busy pulled low.

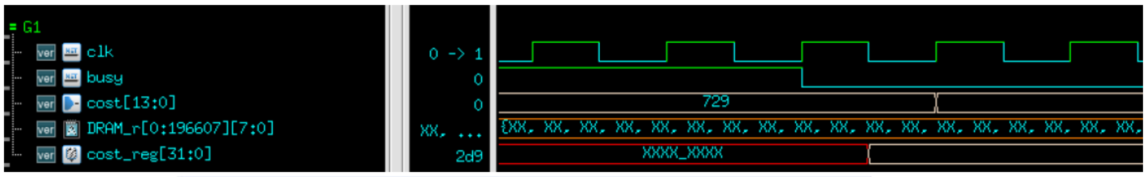


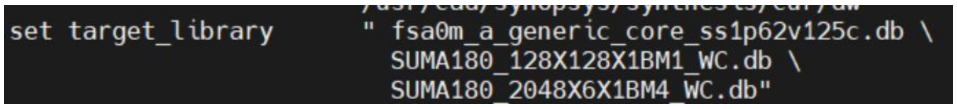
Fig 2. Output waveform example

AXI4 IO

Please refer to Midterm_Project_AXI4.pdf

Specifications

1. Top module name: **MRA**(File name: **MRA.v**)
2. It is **asynchronous** reset and **active-low** architecture.
3. All the data are in **unsigned** format.
4. The total cell area should not larger than **2,500,000 *μm*^2**. The latency of your design in each pattern should not be larger than **1,000,000 cycles**.
5. The maximum clock period is set to **15 ns**, and **you can determine the clock period by yourself**.
6. The input delay and the output delay should be **half** of the clock period. For example, if clock period is 10ns, the input delay and the output delay will be 5ns.
7. The output loading is set to 0.05.
8. The synthesis result of data type cannot include any **LATCH** (check in syn.log).
9. The slack in the end of MRA.timing should be **non-negative** and the result should be **MET**.
10. Since memories are used in this project, the syn.tcl and .synopsys_dc.setup in Lab05 may be a good reference one.



```
set target_library      " fsa0m_a_generic_core_ss1p62v125c.db \
                          SUMA180_128X128X1BM1_WC.db \
                          SUMA180_2048X6X1BM4_WC.db"
```

11. **Note that maximum synthesis time could not be over 3 hours (Normally, this design would not over 3 hours)**
12. No **ERROR** is allowed in every simulation/synthesis.

Grading Policy

1. The performance is determined by latency and cycle time of your design. The smaller the performance index is, the higher score you can get.

   Score = SampleCase(20%) + Functionality(50%)+ Performance(30%)

   SampleCase (20%): 2 simple case, 2 medium case and 2 hard case
   Functionality(50%): Random test
   Performance(30%): Total Latency x Cycle Time

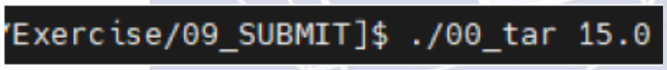
2. **Note that cell area must within 2,500,000 um^2, or you will fail**
3. Performance points can be obtained when both Sample Case and Functionality pass
4. 2nd demo will get 30% off of total score.

TA will demo your design with the DRAM latency ranged from 300 to 500, not fixed.

Note

1. Please submit your files under 09_SUBMIT before 12:00 at noon:
   Due Day: **1st Demo : 04/15** **2nd Demo : 04/17**
   If uploaded files violate the naming rule, you will get 5 deduct points

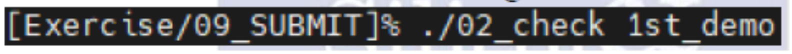
- In this lab, you can adjust your clock cycle time. Consequently, make sure to key in your clock cycle time after the command like the figure below. It means that the TA will demo your design under this clock cycle time:



```
'Exercise/09_SUBMIT]$ ./00_tar 15.0
```

- After that, you should check the following files under **09_SUBMIT/Midterm_Project_iclabXXX/**

  RTL design : **MRA_iclab???.v**, (??? is your account number)
  Memory File : **MEMORY_NAME_iclab???.v**
  MEMORY_NAME_iclab???.db
  File List for Customized SRAM: **filelist_iclab???.f**
  Clock period: **clock_period_iclab???.txt** **(ex: 20.0_iclab099.txt)**
  If you miss any files on the list, you will fail this lab.
  Then use the command like the figure below to check if the files are uploaded or not.



```
[Exercise/09_SUBMIT]% ./02_check 1st_demo
```

Example:

You will submit MRA_iclab099.v, 20.0_iclab099.txt

Given your memory name is RA1SH512, and your submitted memory file

RA1SH512_iclab099.v, RA1SH512_iclab099.db, filelist_iclab099.f

And you modified in filelist_iclab099.f

../04_MEM/RA1SH512.v

(Note that in filelist_iclab099.f, no _iclab099 after your module name)

Besides, you can provide **multiple** memory specs in this lab.

If the uploaded files violating the naming rule, you will get **5 deduct points**.

If you omit any file, you will fail demo.

2. Template folders and reference commands:

   01_RTL/ (for RTL simulation) **./01_run_vcs_rtl**

   ./01_run_vcs_rtl FUNC

   ./01_run_vcs_rtl SAMPLE

   02_SYN/ (for Synthesis) **./01_run_dc_shell**

   Remember modify .tcl to fit your memory db name (refer to Lab05)

   (Check the design which contains **Latch** and **Error** or not in **syn.log**)

   (Check the design's timing in /Report/MRA.timing to see if the slack is **MET**)

   03_GATE_SIM/ (Gate Level simulation) **./01_run_vcs_gate**

   You can key in **./09_clean_up** to clear all log files and dump files in each folder

   04_MEM/ (Memory location)

   You should generate your memories and put the required files (.v and .db) here.

   09_SUBMIT/ (submit your files) ./00_tar ./01_submit ./02_check

You should make sure the clock period values identical in
00_TESTBED/PATTERN.v and 02_SYN/syn.tcl

Sample Case

3 Cases: Simple, **Medium**, **Hard**

_0: Simple

_1: Medium

_2: Hard

00_TESTBED/DRAM/

  dram.dat

  You can write a pattern to read this file and send to chip

00_TESTBED/TEST_CASE/

  Input.txt

  You can write a pattern to read this file and send to chip

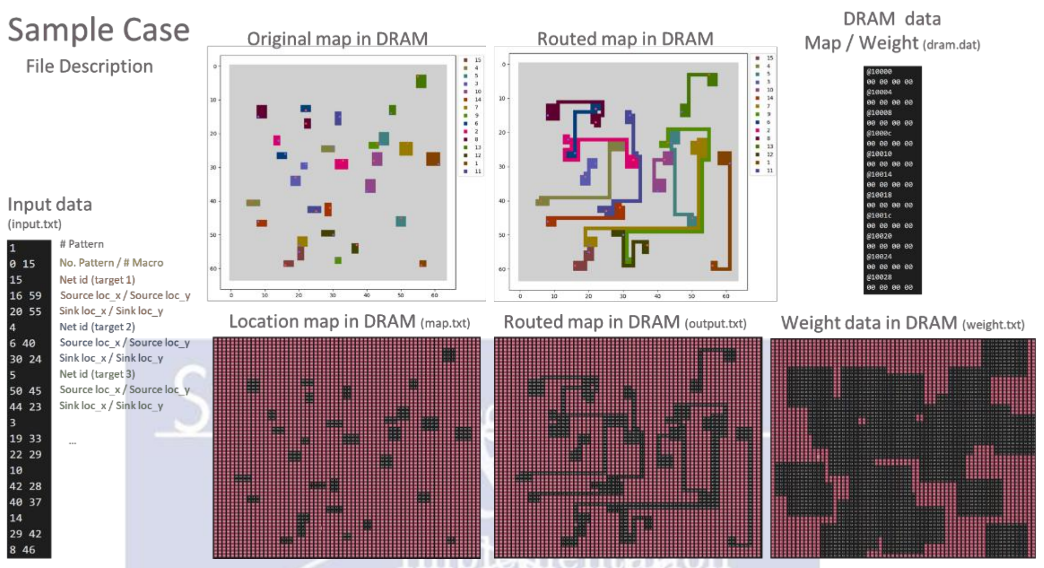
Map.txt, Output.txt, Weight.txt

Provide for you to debug

00_TESTBED/Picture/

***.png**

Provide for you to debug



```
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
```

```
@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
```

Hint

1. **Recommend development flow:**
   I. Establish a PATTERN for testing your circuit
   II. Ensure your design can fetch data from DRAM with AXI4
   III. Ensure your design can write back to DRAM with AXI4
   IV. Design the Memory usage of whole algorithm
   V. Design the Architecture of circuit and FSM for each part
   VI. Implementation of algorithm (01 → 02 → 03)
   VII. Verify circuit functionality with more test case
   VIII. Find out bottleneck and do some optimization
   IX. Optimize latency – Algorithm and Architecture
   X. Optimize area – Memory and Circuit Design
   XI. Optimize routed cost – Algorithm (optional)
2. **TA provide a pseudo code for those not familiar with this algorithm**

Algorithm 1 Filling Path Map

```
Traverse Sequence = 1,2,3,1,2,3,...
CLEAN *node list*
PUSH *source* into *node list*
**while** *node list* is not empty **do**
    POP *node list* as *current node*
    **for** *neighbor* is Down, Up, Right, Left Node **do**
        **if** *neighbor* is *sink* **then**
            *current node* = *neighbor*
            break while loop
        **else if** *neighbor* is inside region, not block, not traversed **then**
            TRAVERSED *neighbor* (Update Path Map)
            PUSH *neighbor* into *node list*
        **end if**
    **end for**
**end while**
**if** *current node* is not *sink* **then**
    ERROR
**end if**
```

Algorithm 2 Retrace Path Map

```
Traverse Sequence = 3,2,1,3,2,1,...
*current node* = *sink*
**while** *current node* is not *source* **do**
    **for** *neighbor* is Down, Up, Right, Left Node **do**
        **if** *neighbor* is right Traverse Sequence and inside region **then**
            Update Location Map
            *current node* = *neighbor*
            break For loop
        **end if**
    **end for**
**end while**
```