Midterm Project
Maze Router Accelerator

Analysis

Choose SRAM size

- Total bits: 64 x 64 x 4bits x 2 (one for location map & the other for weight map)
- Filling: No need to read from or write to SRAM.
- Retrace: For every point, need to read from SRAM twice(one for location & the other for weight) and write to SRAM once.
- Read DRAM: Bandwidth of 128 bits/burst.
- Write DRAM: Bandwidth of 128 bits/burst.

SRAM size (word, bits)	Area (um^2)	Bandwidth
2 x (128, 128)	669550.5	256 bits/burst
2 x (256, 64)	428764.0938	128 bits/burst

-> Choose (256, 64) for smaller area.

Storage Method of SRAM

Location map

- 128 bits/burst -> 32 points/burst.
- Store point 0 ~ 15 in SRAM1 and point 16 ~ 31 in SRAM2 at their even addresses.

Weight map

- 128 bits/burst -> 32 points/burst.
- Store point 0 ~ 15 in SRAM2 and point 16 ~ 31 in SRAM1 at their odd addresses.

Storage Method of SRAM

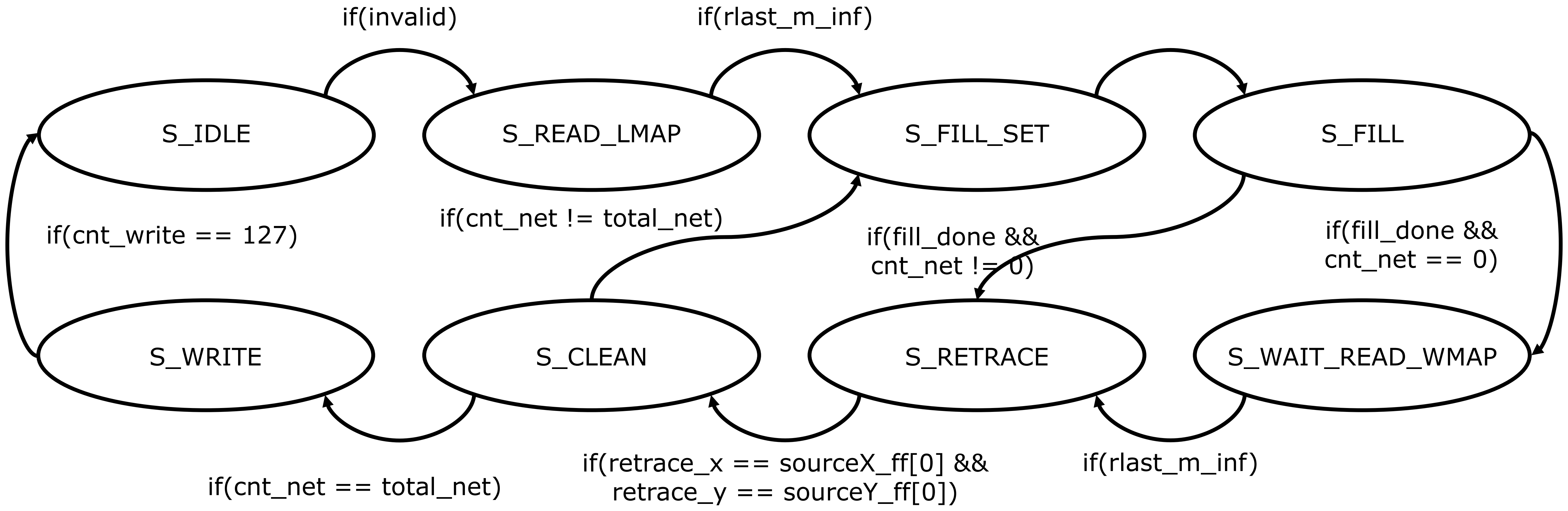
SRAM1

Address	Data
0	L (0, 0) ~ (0, 15)
1	W (0, 16) ~ (0, 31)
2	L (0, 32) ~ (0, 47)
3	W (0, 48) ~ (0, 63)
4	L (1, 0) ~ (1, 15)
5	W (1, 16) ~ (1, 31)
6	L (1, 32) ~ (1, 47)
7	W (1, 48) ~ (1, 63)
⋮	⋮
252	L (31, 0) ~ (31, 15)
253	W (31, 16) ~ (31, 31)
254	L (31, 32) ~ (31, 47)
255	W (31, 48) ~ (31, 63)

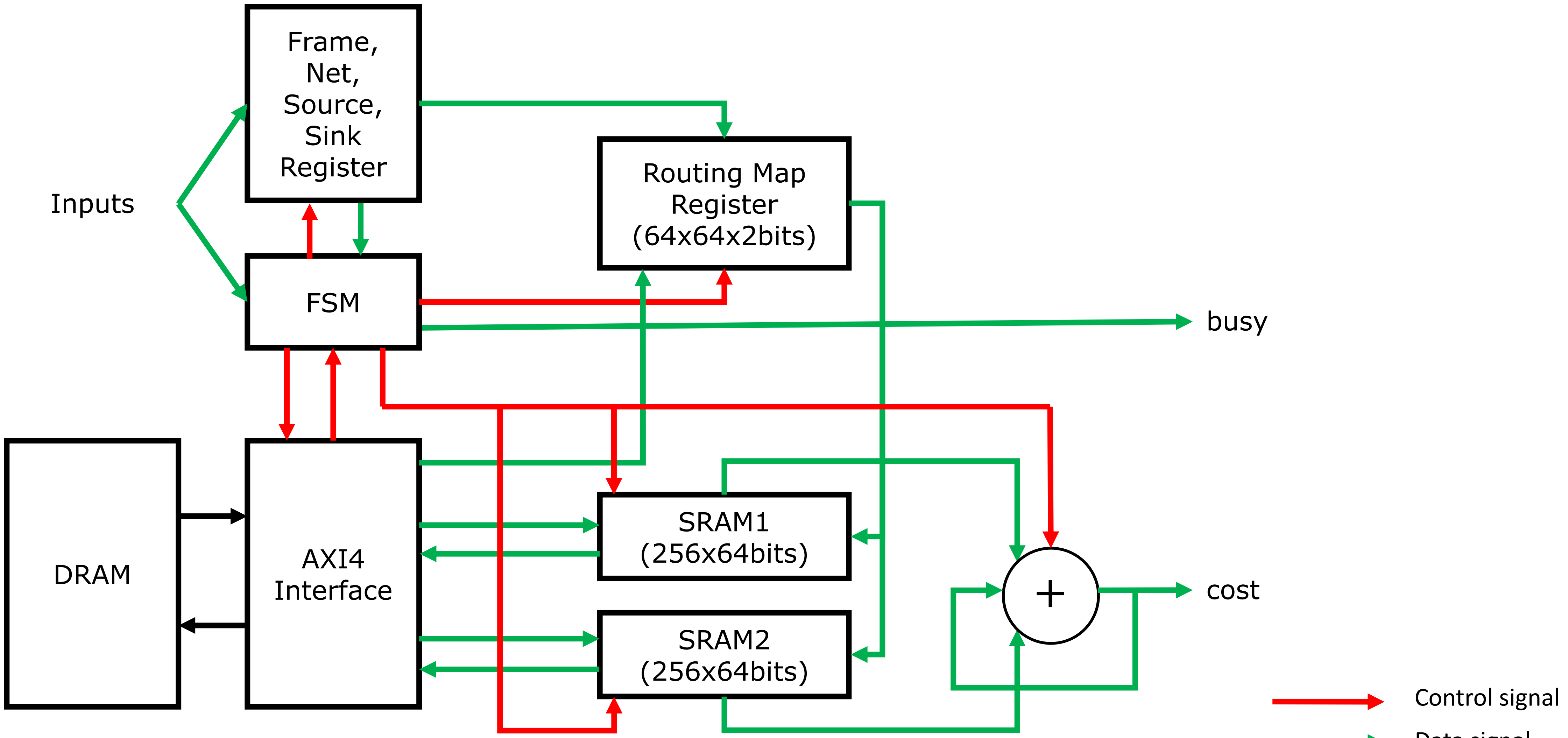
SRAM2

Address	Data
0	L (0, 16) ~ (0, 31)
1	W (0, 0) ~ (0, 15)
2	L (0, 48) ~ (0, 63)
3	W (0, 32) ~ (0, 47)
4	L (1, 16) ~ (1, 31)
5	W (1, 0) ~ (1, 15)
6	L (1, 48) ~ (1, 63)
7	W (1, 32) ~ (1, 47)
⋮	⋮
252	L (31, 16) ~ (31, 31)
253	W (31, 0) ~ (31, 15)
254	L (31, 48) ~ (31, 63)
255	W (31, 32) ~ (31, 47)

FSM

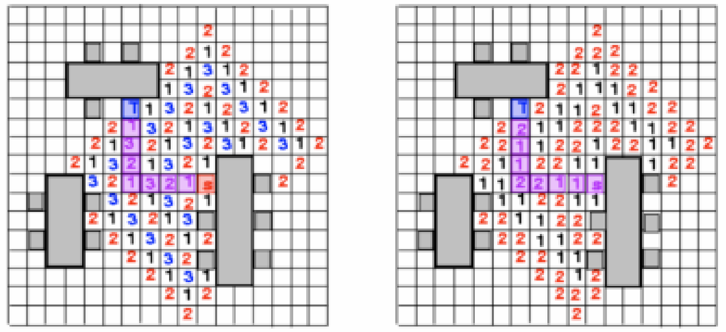

if(invalid)
if(rlast_m_inf)
S_IDLE
S_READ_LMAP
S_FILL_SET
S_FILL
if(cnt_write == 127)
if(cnt_net != total_net)
if(fill_done && cnt_net != 0)
if(fill_done && cnt_net == 0)
S_WRITE
S_CLEAN
S_RETRACE
S_WAIT_READ_WMAP
if(cnt_net == total_net)
if(retrace_x == sourceX_ff[0] && retrace_y == sourceY_ff[0])
if(rlast_m_inf)

Architecture



Filling

- 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, ...

State

- 0 -> Empty.
- 1 -> Blocked.
- 2, 3 -> Code sequence.
- 2 bits routing map register

When state of FSM == S_FILL_SET

- Set source = 3, sink = 0.

When state of FSM == S_FILL

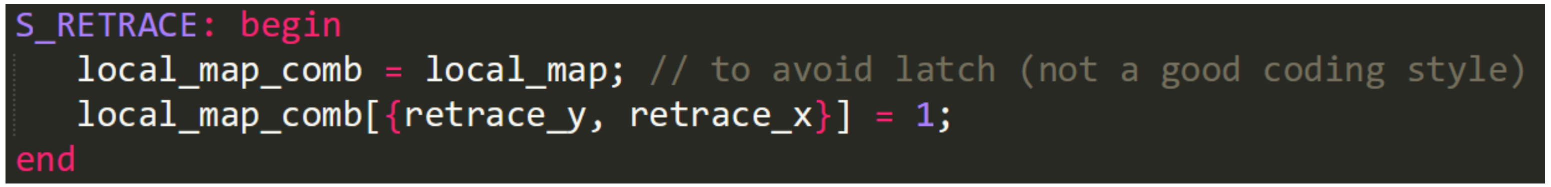
- Use a counter[1:0] to get code sequence.

  ex.   counter[1:0]:   0 -> 1 -> 2 -> 3…

  Code sequence: 2 -> 2 -> 3 -> 3…
- If sink != 0, state of FSM changes to S_RETRACE.

Retrace

- Use counter to count back and decide the path.
- Use signal ‘retrace_x’ and ‘retrace_y’ to store the current position.
- Set the path = 1 (blocked)



```
S_RETRACE: begin
    local_map_comb = local_map; // to avoid latch (not a good coding style)
    local_map_comb[{retrace_y, retrace_x}] = 1;
end
```

Retrace

	counter[0] == 0	counter[0] == 1
SRAM action	Read 2 SRAM (location & weight)	Write 1 SRAM(location)
Routing map register action	Access map’s current, up, down, left & right point	Decide next retrace_x & retrace_y
Other action	Store weight of the current point	Accumulate cost

- For every point, need to read from SRAM twice and write to SRAM once. (need 2 cycles for every point)
- If current position == source point, state of FSM changes to S_CLEAN. (clean routing map register & get new source & sink point).