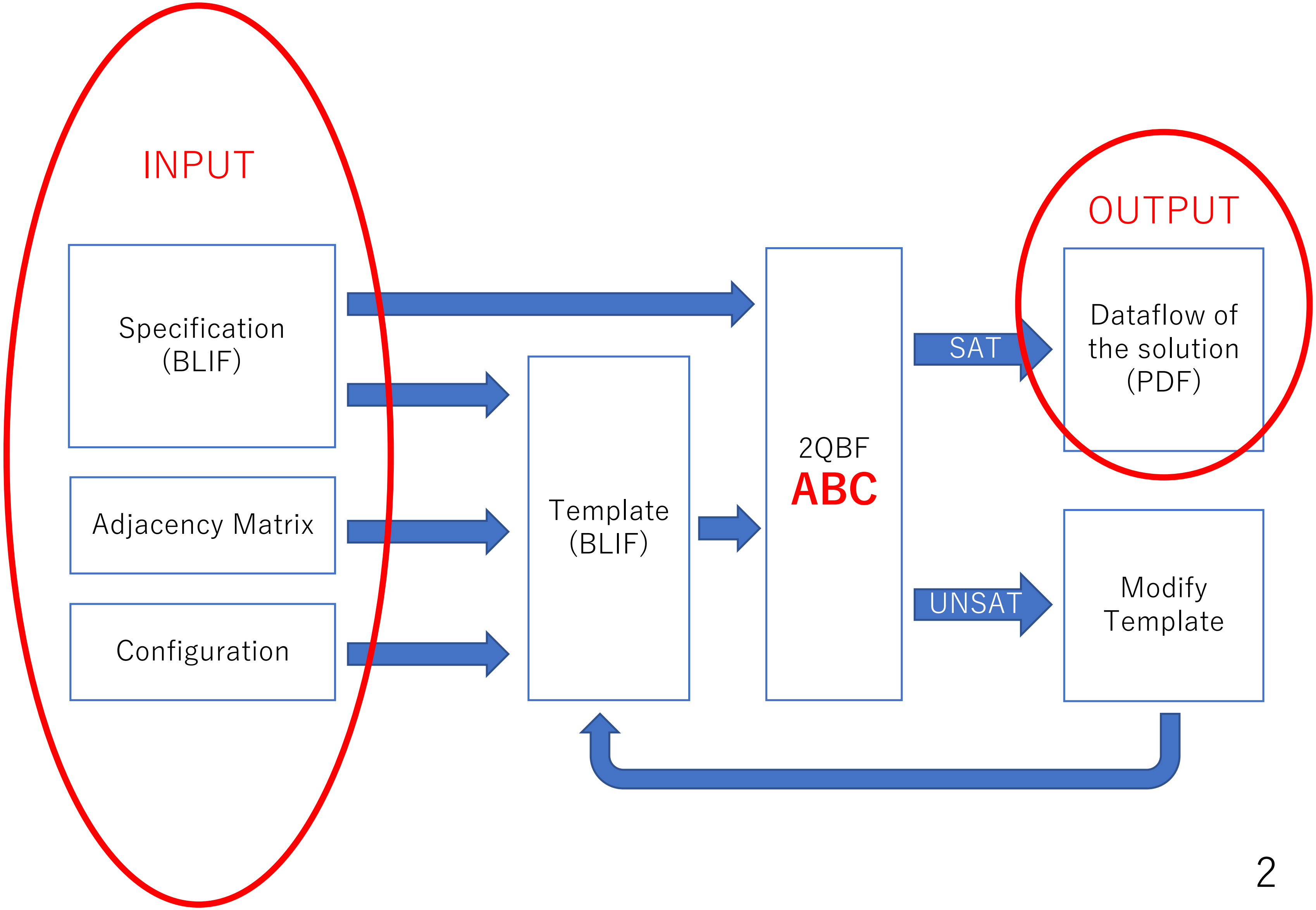
Contents

- Flow and Example without Options 2~20
- Options and Expample 21~42

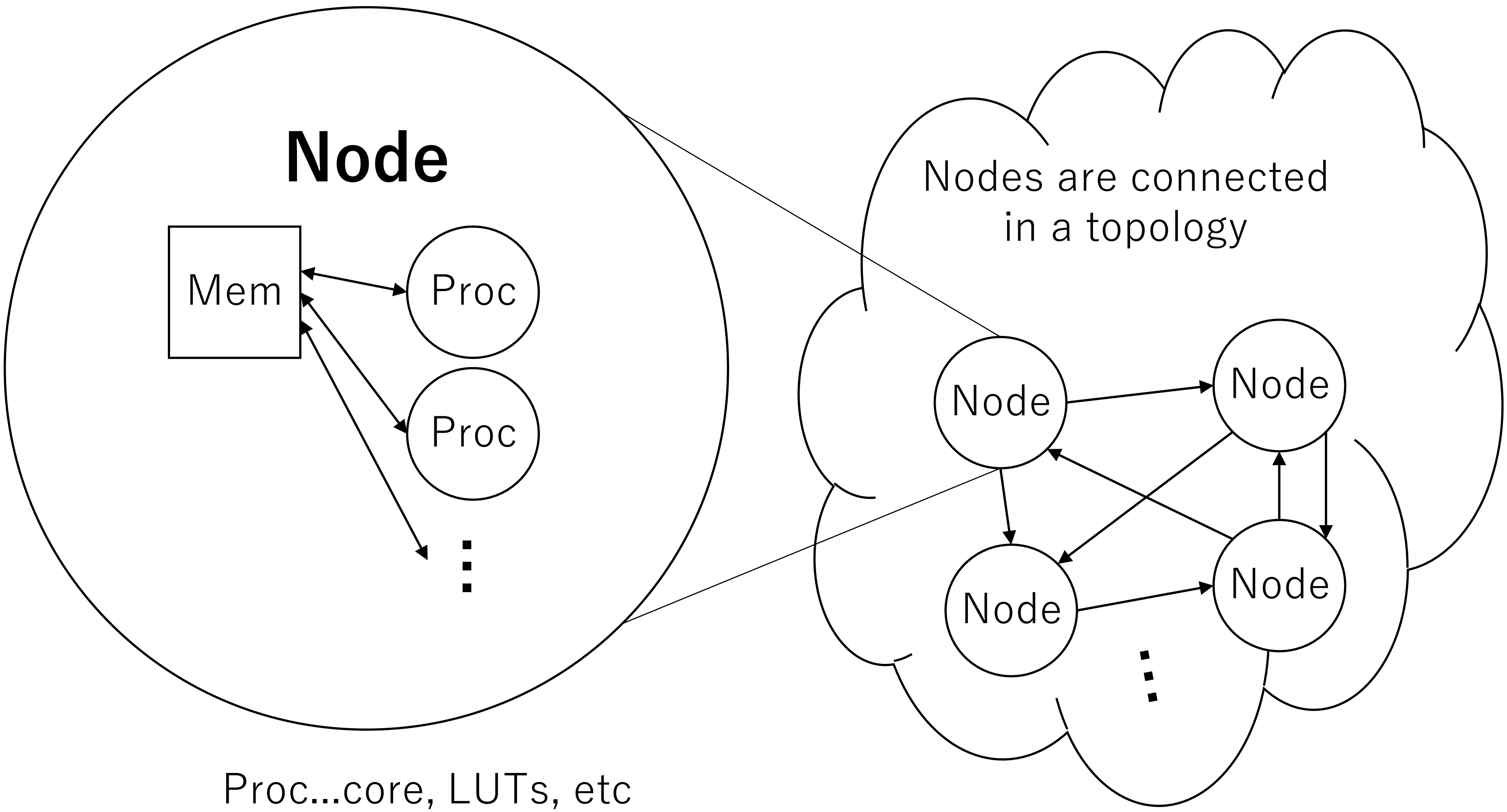
Flow of Synthesizer



INPUT

- Specification
  - Specification must be written in BLIF. So, relation between Primary Input(PI) and Primary Output(PO) must be specified in bit-level.
  - Top module must be specified.
  - You can assume that every variables is 1 bit, in order to perform synthsis faster. It is empirically known that correct data-flow is synthesized even doing so.

Environment(NUMA or NORA)



Proc…core, LUTs, etc

INPUT

- Adjacency Matrix
  - We can create adjacency matrix for the topology of emvironment by assigning number to each node.
  - It is assumed that the topology is represented by weighted connected directed graph. Weight of edge represents the number of bits which can be sent in 1 cycle.
  - If there is an edge from node i to node j, element of adjacency matrix at (i, j) is its weight. Else, it is 0. Element at (i, i) is ignored.
  - Note that we cannot represent BUS with directed graph. Also, bidirectional edge is not allowed.

INPUT

- Configuration
  - It largely depends on template. Let’s start from explanation of template.

Template

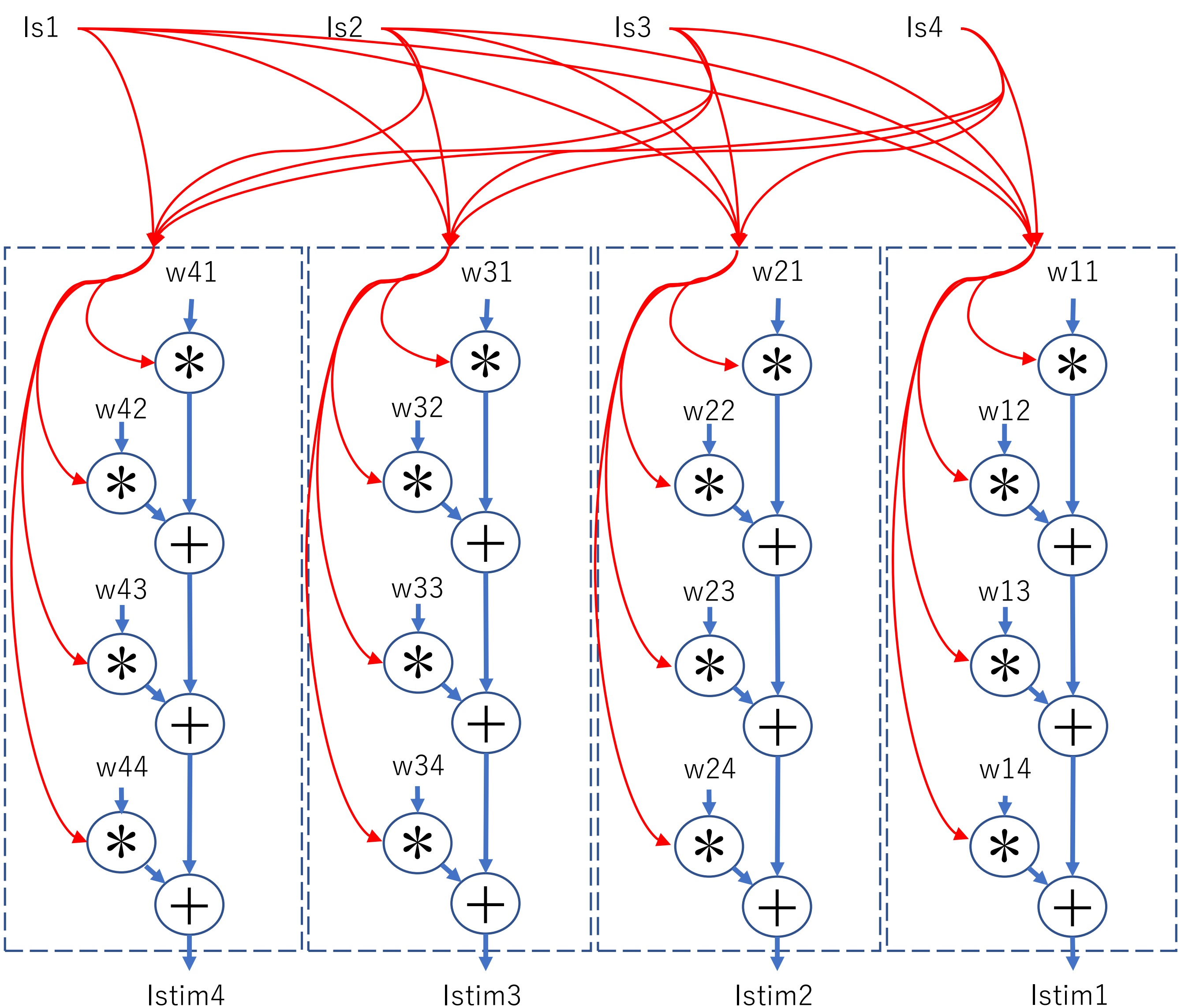
- We defined template based on an algorithm to calculate matrix-vector product with nodes connected in ring-topology.

$$\begin{pmatrix} I_{stim}1 \\ \ldots \\ I_{stim}N \end{pmatrix} = \begin{pmatrix} w(1,1) & \ldots & w(1,N) \\ \ldots & \ldots & \ldots \\ w(N,1) & \ldots & w(N,N) \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ \ldots \\ I_sN \end{pmatrix}$$

with N nodes.

Each of *Is1～IsN* is distributed to only one node initially.

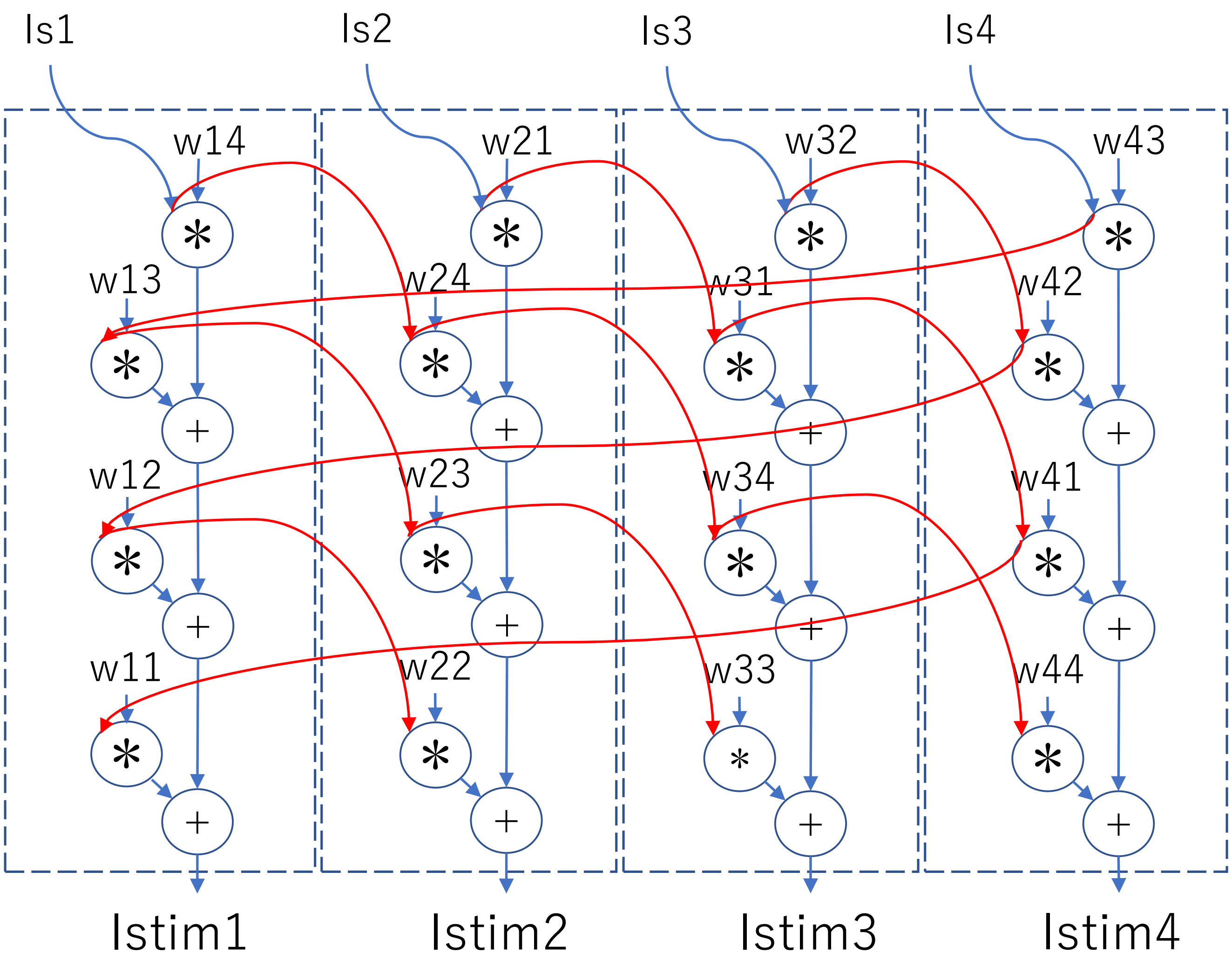
Ordinary Algorithm



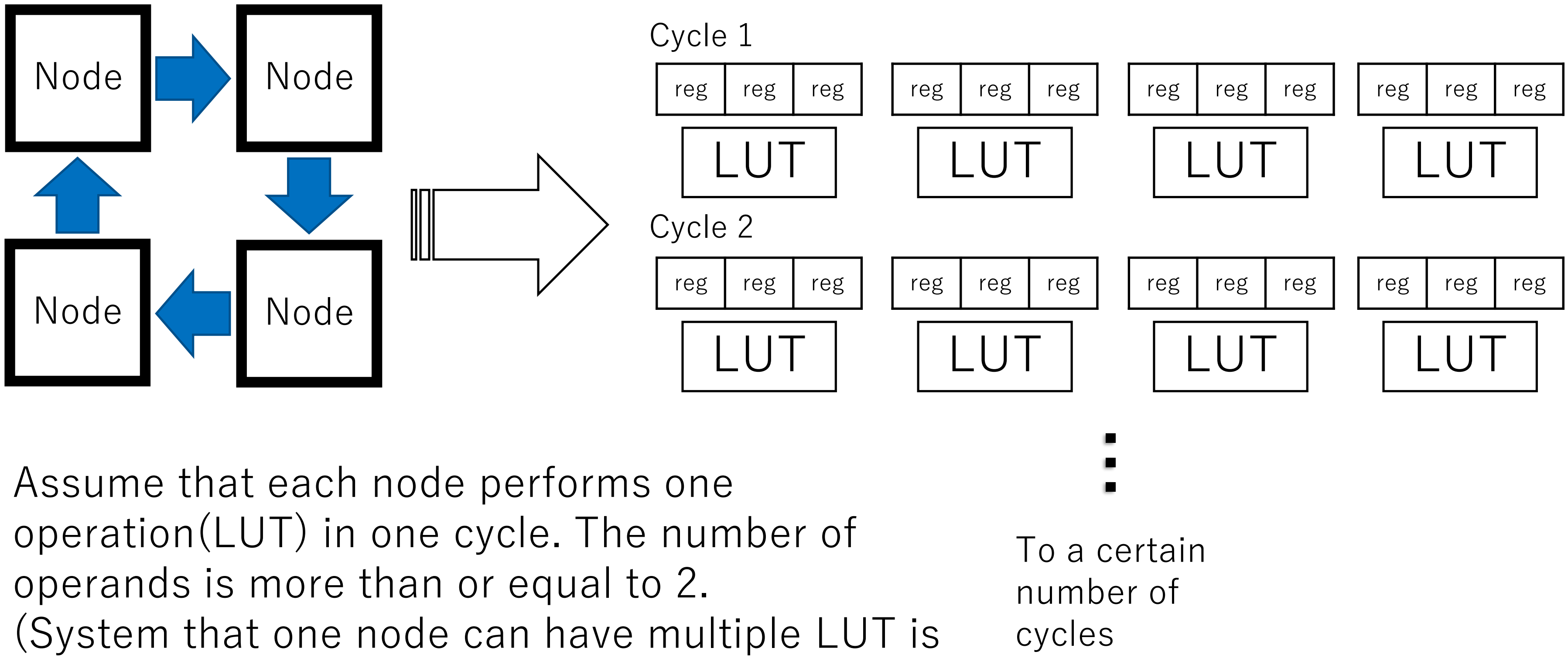
- Send all vector elements to every node initially.
- The communication may become overhead of calculation.

Proposed Algorithm

- Communicate vector elements among nodes by cycle
- This may be more efficient than ordinary method 1, if multiplicaton and communication can be executed simultaneously.



Template Modeling the Algorithm

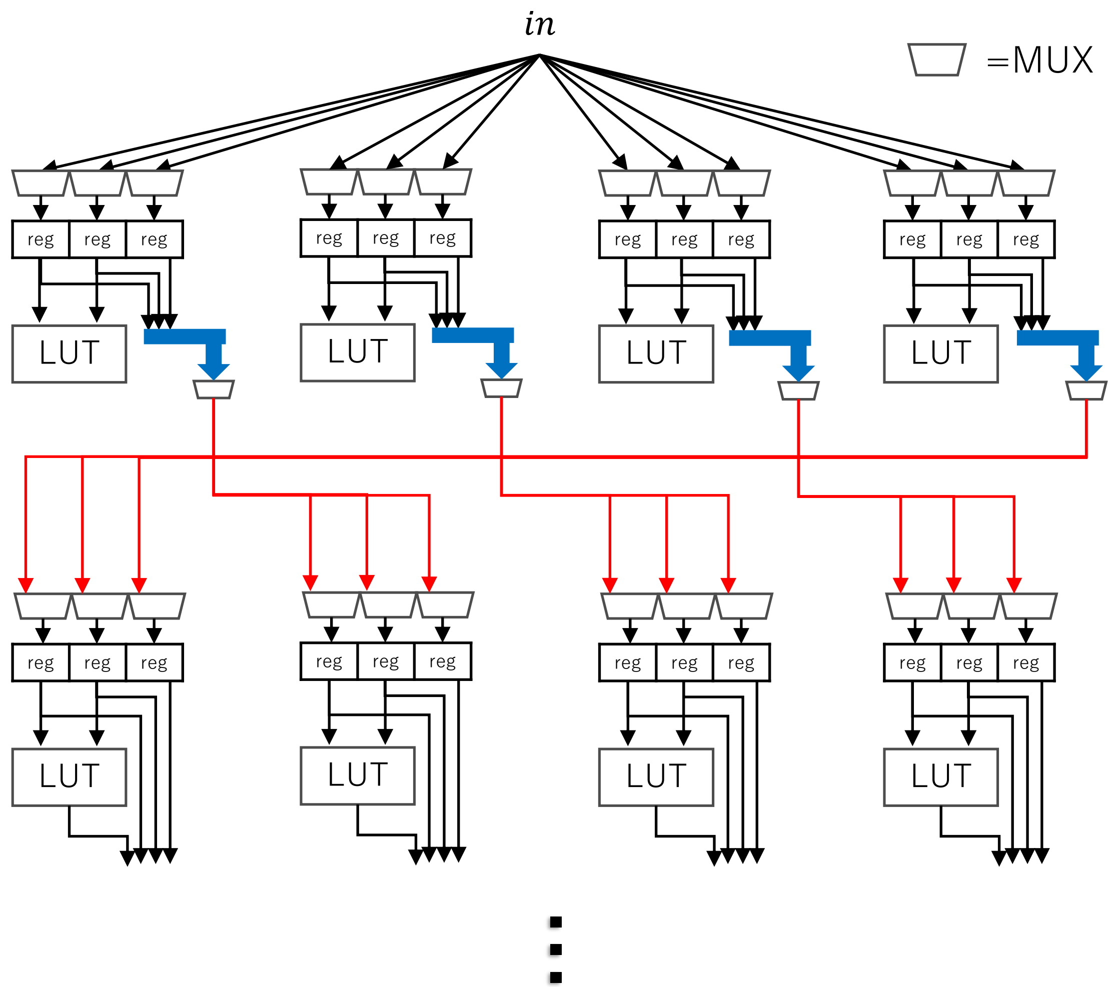


- Assume that each node performs one operation(LUT) in one cycle. The number of operands is more than or equal to 2.
- (System that one node can have multiple LUT is under construction.)
- Each node has a certain number of registers.
- Time frame expansion by a certain number of times.

Template Modeling the Algorithm

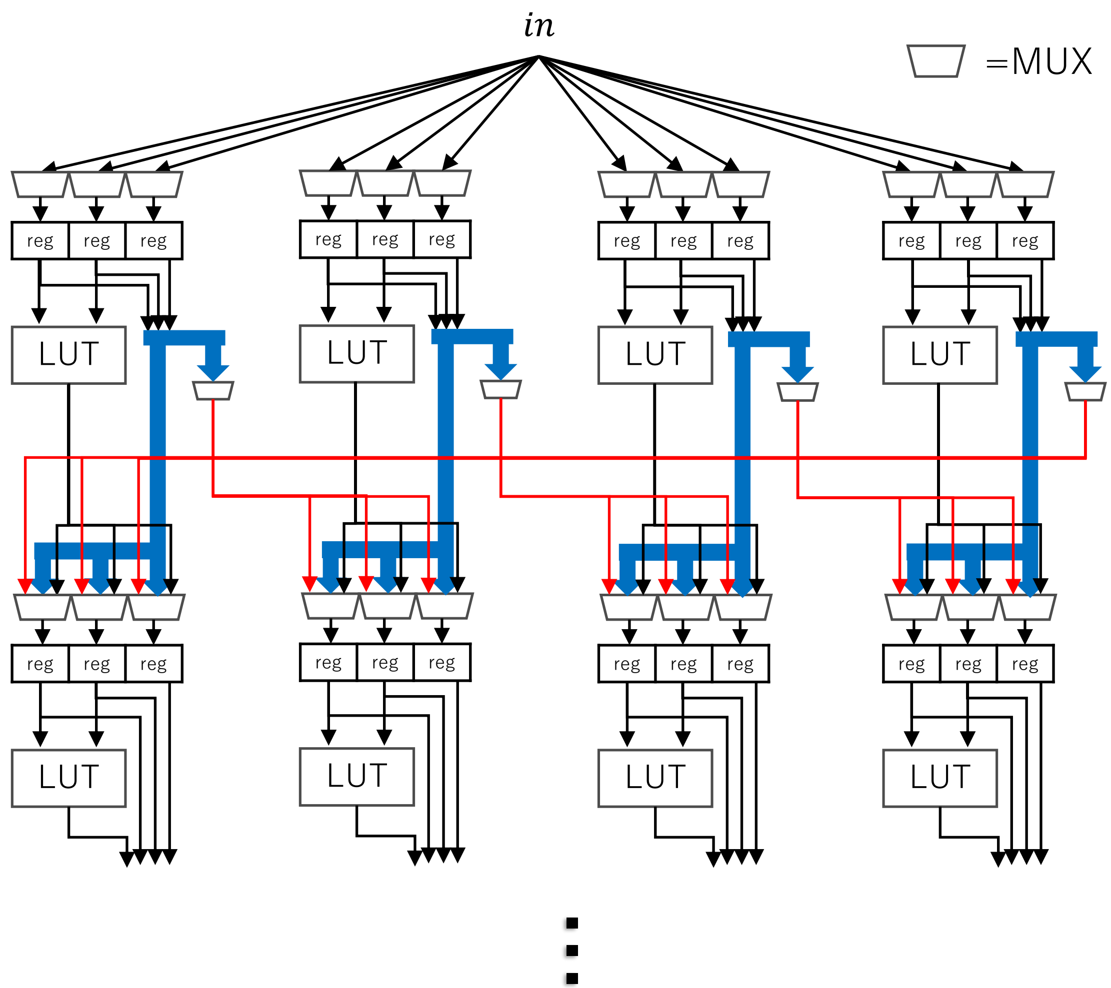
- All inputs of calculation are input to each MUX for register at the first cycle.
- The value sent to connected node is selected from registers.
- The communication is performed simultaneously with an operation in LUT.

- NOTE : Templates can be constructed in the same way for communication structures other than ring.



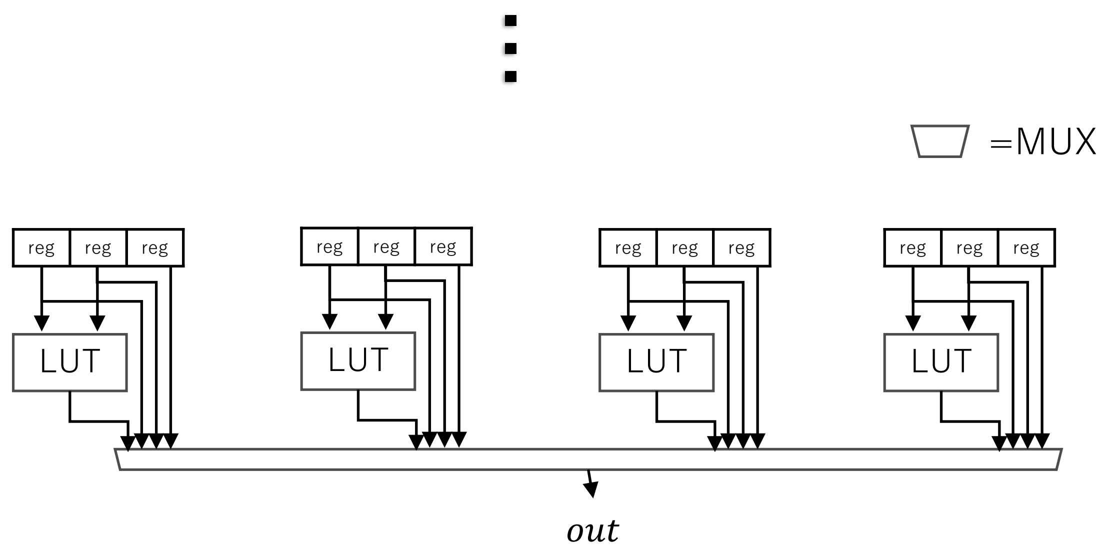
Template Modeling the Algorithm

- The values of registers in the same node can be used in the next cycle.
- A result of LUT operation in the same node can also be used.



Template Modeling the Algorithm

- An output of calculation is selected from values in all nodes at the last cycle.



Synthesis Method

```
for (cycle = 1; TRUE; cycle++)
  for (operand = 2; operand <= max_operand; operand++)
    for (reg = min_reg; reg <= max_reg; reg++)
      Synthesis
        If successful, break all loops.
```

min_reg = #PI / #node (+1 if there is a remainder)

- Upper bounds are given by user.
- Result may change when the order of loops is changed.
- We necessarily get solution with optimum performance if synthesis is tried in order of performance.

INPUT

- Configuration
  - Upper bounds shown in the last slide.
  - Lower bounds can also be specified if necessary.
  - Many options to reduce the search space.

OUTPUT

- Dataflow of the solution is represented in figure.
- Registers' values, LUT functions, Communicated values, and Outputs' values are specified.

Example 1

SPEC.blif

```
#.top matrix
.model matrix
.inputs Is1 w11 w12
.inputs Is2 w21 w22
.outputs Istim1 Istim2

.subckt and in0=Is1 in1=w11 out=Is1w11
.subckt and in0=Is2 in1=w12 out=Is2w12
.subckt xor in0=Is1w11 in1=Is2w12 out=Istim1

.subckt and in0=Is1 in1=w21 out=Is1w21
.subckt and in0=Is2 in1=w22 out=Is2w22
.subckt xor in0=Is1w21 in1=Is2w22 out=Istim2
.end
```

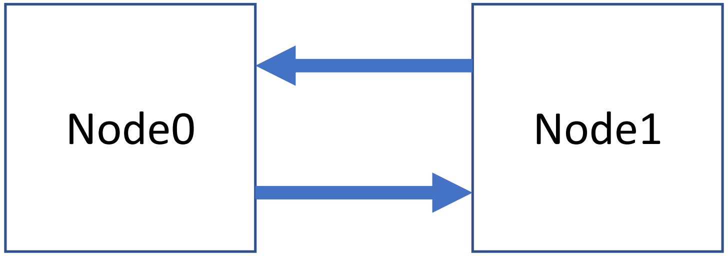
```
.model and
.inputs in0 in1
.outputs out
.names in0 in1 out
11 1
.end

.model xor
.inputs in0 in1
.outputs out
.names in0 in1 out
10 1
01 1
.end
```

Example 1

Adjacency Matrix

0 1
1 0



Rows are separated by newlines.
Columns are separated by space.

Number of nodes equals to size of matrix because it is connected graph.

Example 1

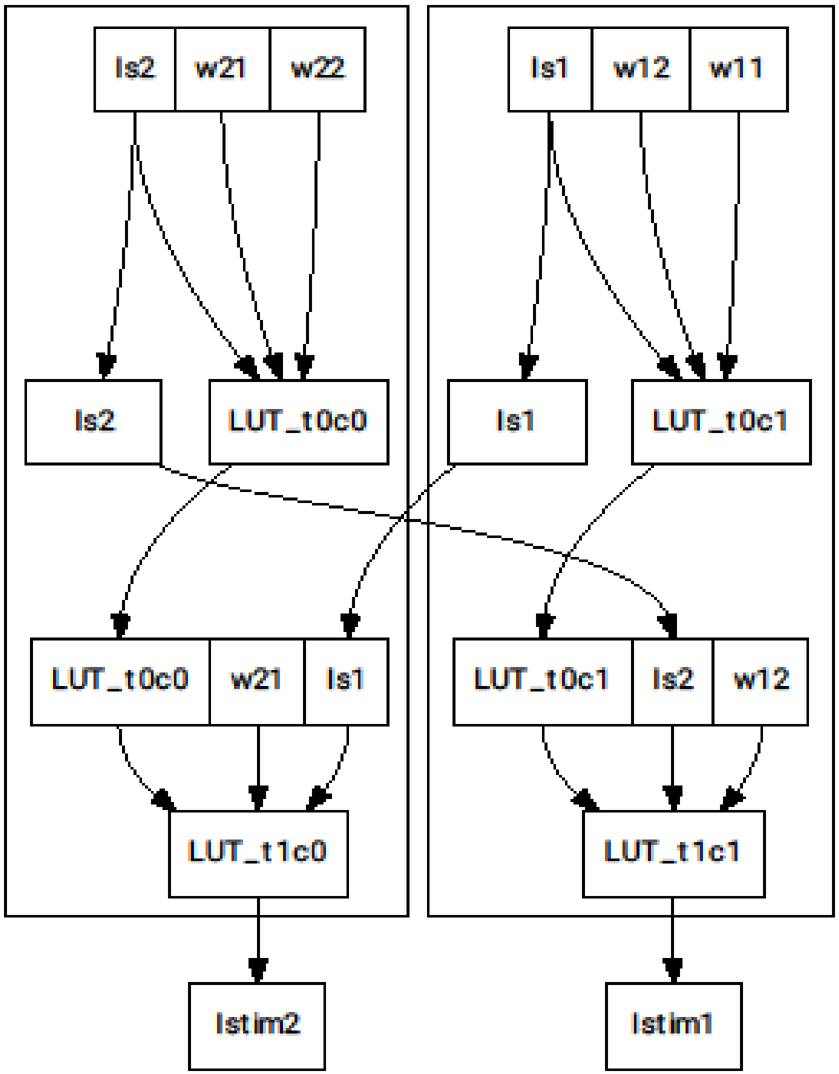
Configuration

number_of_register
-,3
number_of_operands
-,3

Format: (separated by comma)
LowerBound,UpperBound

Example 1

OUTPUT



input	LUT_t0c0	LUT_t0c1	LUT_t1c0	LUT_t1c1
0	1	0	1	0
1	1	0	0	1
2	1	1	1	0
3	1	1	0	1
4	1	0	1	1
5	0	1	0	0
6	1	1	0	0
7	0	0	1	1

`time:1.56694sec`

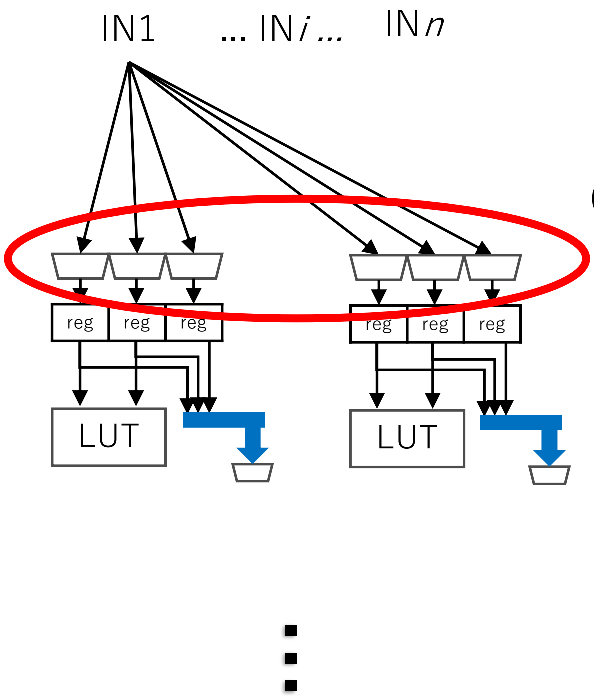
Although LUT function is complicated, the calculation can be performed correctly in this dataflow by assigning
LUT_t0c0 to $Is2 \times w22$,
LUT_t0c1 to $Is1 \times w11$,
LUT_t1c0 to $\mathrm{LUT\_t0c0} + Is1 \times w21$,
LUT_t1c1 to $\mathrm{LUT\_t0c1} + Is2 \times w12$.

Options

1. Each PI is selected by only one MUX.
2. PIs are divided equally and each part is input to one node. The way of division depends on the order of PI in Specification file.
4. Data Selections for registers are shared among nodes at the same cycle except first cycle.
5. “ among designated cycles in the same node. You cannot choose first cycle.
6. Data to be communicated is fixed to value of designated register.
7. POs are divided equally and each node outputs all POs in one part. The way of division depends on the order of PO in Specification file.
8. LUT function is fixed as specified.
11. This restricts data selection for registers at first cycle as specified.
12. “ for registers at secont and subsequent cycles.
13. “ for POs.

Option 3, 9, 10 are not used currently.

Option1



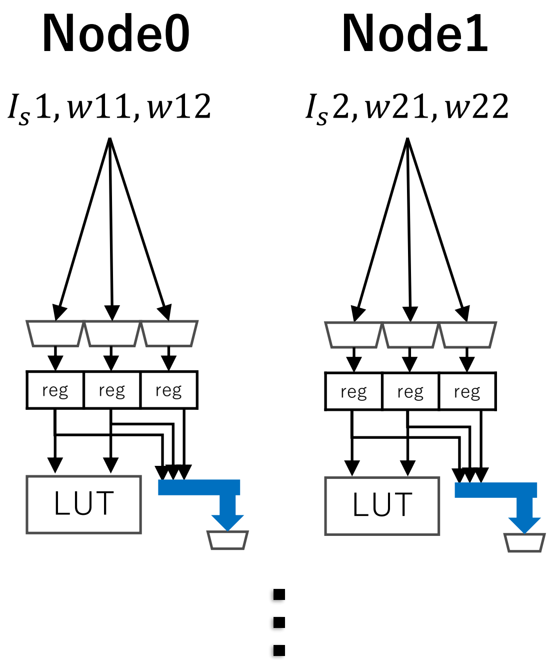
Only one of MUXs can select IN*i*.

Option2

SPEC.blif

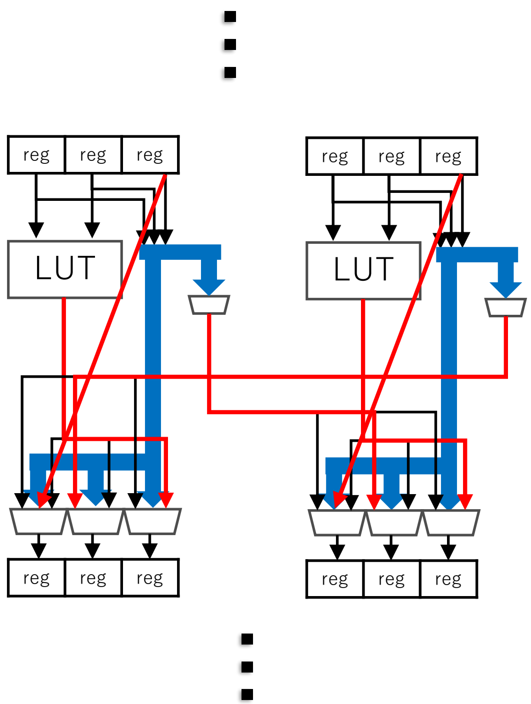
#.top matrix
.model matrix
.inputs Is1 w11 w12
.inputs Is2 w21 w22
.outputs Istim1 Istim2
...

#Node = 2
from Adjacency Matrix



MUXs in Node0 can select $I_s1, w11, w12$.
MUXs in Node1 can select $I_s2, w21, w22$.

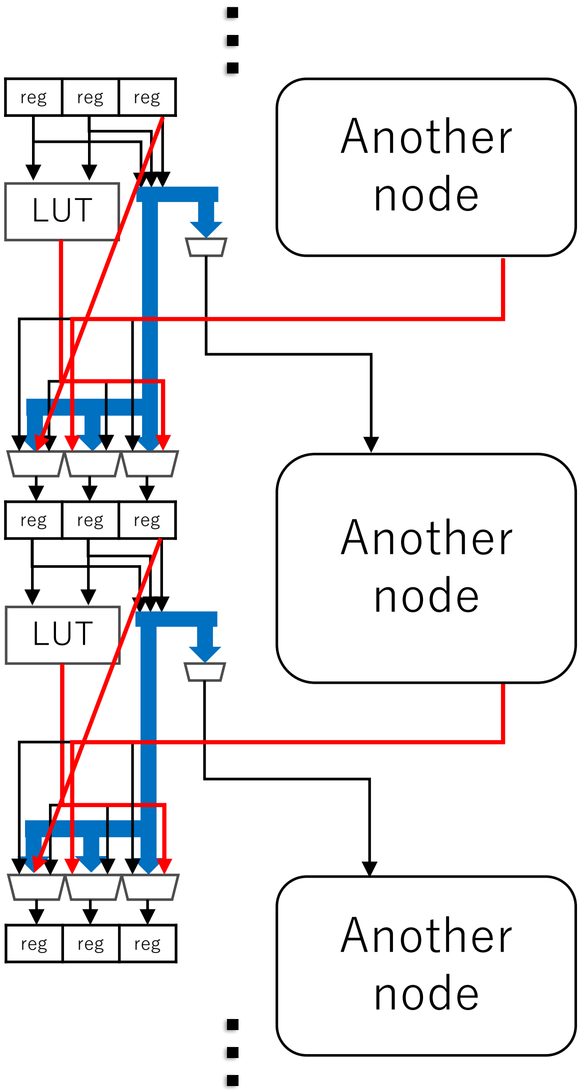
Option4



Red arrows are selected.

Data selections for registers are shared among nodes.

Option5



Red arrows are selected.

Data selections for registers are shared among designated cycles.

You can designate like

Option7 2,3 4,5.6

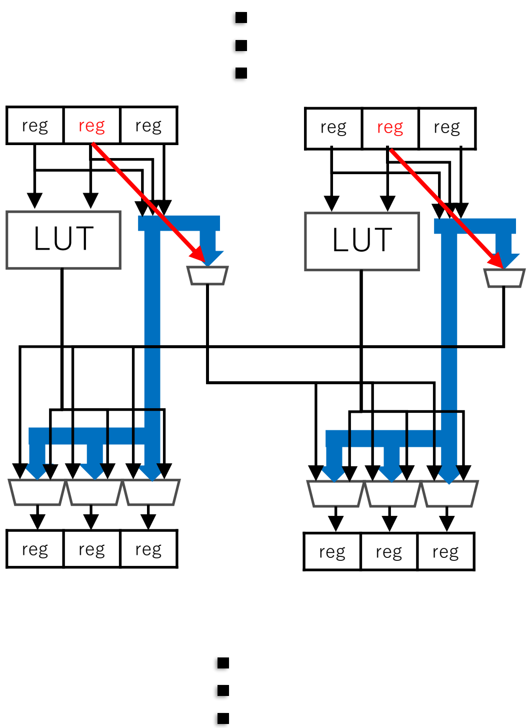
This make data selections shared between {2, 3}, and among {4, 5, 6}.
Data selections can be different between {2, 3} and {4, 5, 6}.

You cannot select cycle 1, because MUXs select not reg but PI at the first cycle.

Option6

Option6
2

Send value of second register always.



You can designate the register whose value is sent.

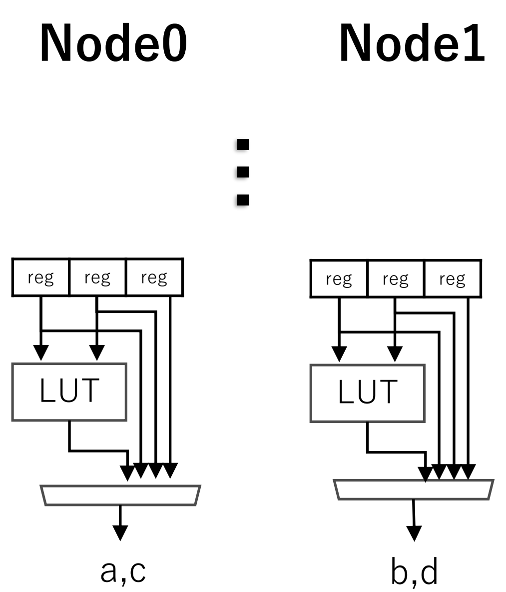
Note that if multiple values are sent in the same edge, this option fixes only one of the values and the other values are selected by MUXs as normally.

Option7

SPEC.blif

#.top matrix
.model matrix
.inputs ...
.outputs a b c d
...

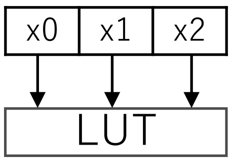
#Node = 2
from Adjacency Matrix



a,c are selected only from values in Node0.
b,d are selected only from values in Node1.

Option8

Option8
f0,f1,f2,f3,f4,f5,f6,f7



x2	x1	x0	LUT out
0	0	0	f0
0	0	1	f1
0	1	0	f2
0	1	1	f3
1	0	0	f4
1	0	1	f5
1	1	0	f6
1	1	1	f7

Option8
0,0,0,0,0,0,0,1

→ AND3

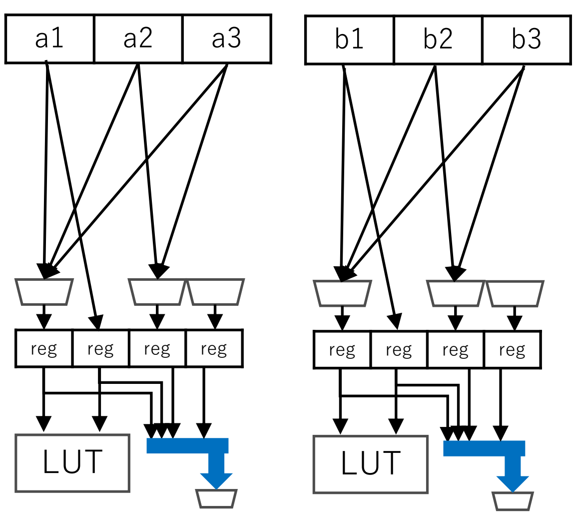
Option11

This option is available only when option2 is enabled.

SPEC.blif

```
#.top test
.model test
.inputs a1 a2 a3
.inputs b1 b2 b3
.outputs c1 c2
...
```

Node0 **Node1**



⋮

#Node = 2
from Adjacency Matrix

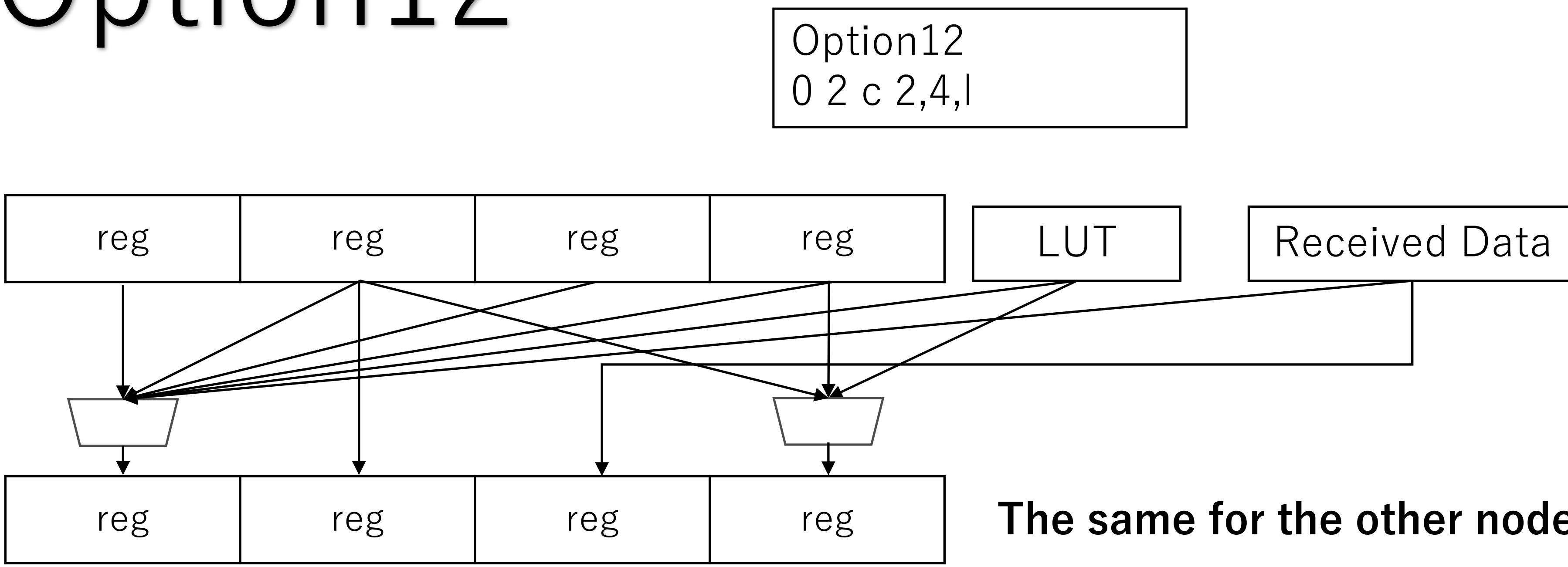
```
Option11
0 1 2,3 -1
```

Case "0" ... Its candidates are all of the divided PIs.
Case "1" ... The first one in divided PIs is directly connected to the register.
Case "2,3" ... Its candidates are second and third ones in divided PIs.
Case "-1" ... It has no candidates.

Note

- If there is no candidates, the value will be 0.
- You may need to modify spec.blif to arrange the order of PIs.
- If the PI is directly conntected to the register, Option1 will not take it into account because there is no MUX.

Option12

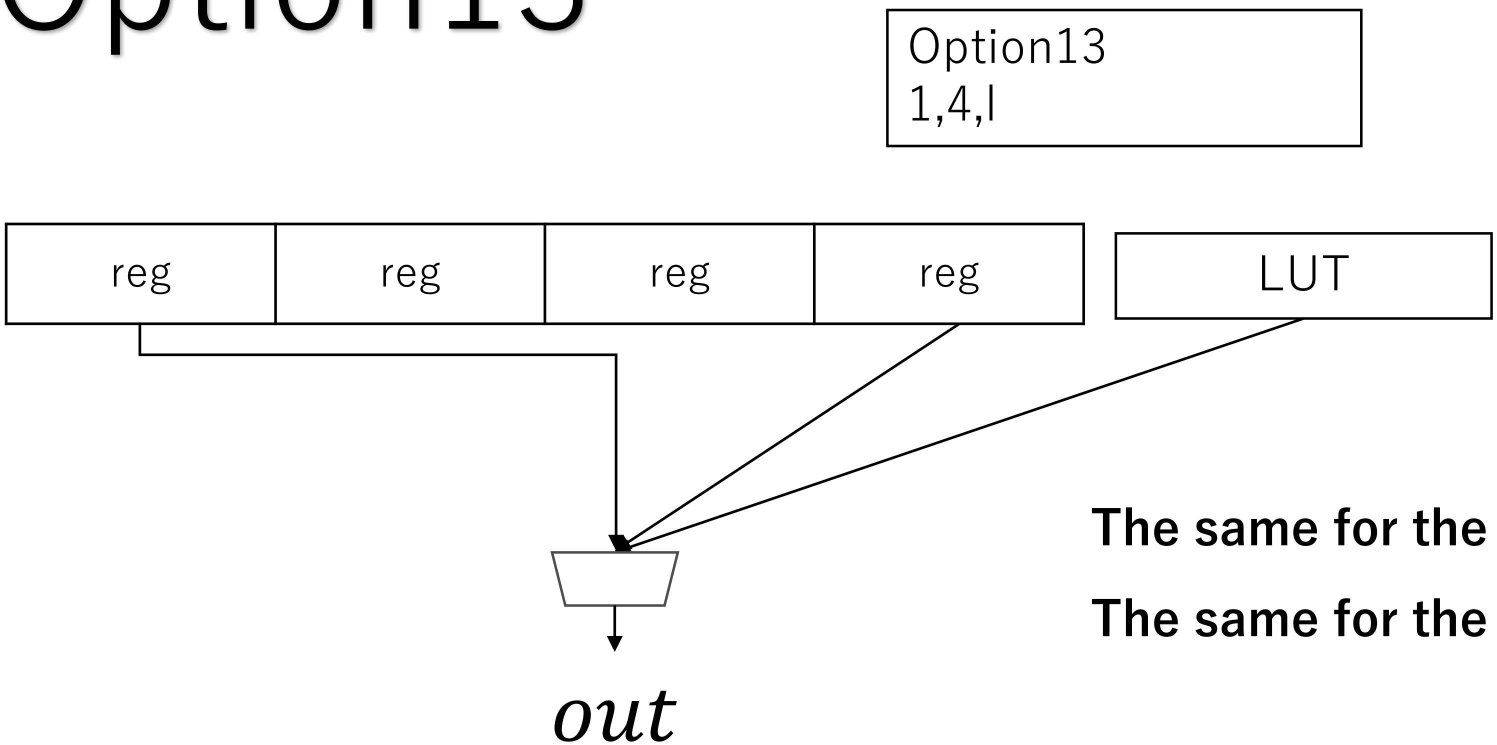


The same for the other nodes.

Case “0” … Its candidates are all registers and received data.
Case “2” … The second register is directly connected to the register.
Case “c” … The received data is directly connected to the register.
Case “2,3,l” … Its candidates are second, fourth registers and LUT output.

Note
If there are multiple data received, only one of them will be assigned directly in the case of “c”. If you want MUX to select them, use “c,c”.

Option13



The same for the other POs.

The same for the other nodes.

The candidates for POs are designated: registers by number, LUT output by “l”.

Example 2

SPEC.blif

```
#.top matrix4
.model matrix4
.inputs Is1
.inputs Is3
.inputs w11 w12 w13 w14 w31 w32 w33 w34
.inputs Is2
.inputs Is4
.inputs w21 w22 w23 w24 w41 w42 w43 w44
.outputs Istim1 Istim2 Istim3 Istim4

.subckt and in0=Is1 in1=w11 out=Is1w11
.subckt and in0=Is2 in1=w12 out=Is2w12
.subckt and in0=Is3 in1=w13 out=Is3w13
.subckt and in0=Is4 in1=w14 out=Is4w14
.subckt xor4 in0=Is1w11 in1=Is2w12 in2=Is3w13 in3=Is4w14 out=Istim1
…
```

Adjacency Matrix

0 1
1 0

NOTICE that PIs are arranged in a particular order.

Example 2

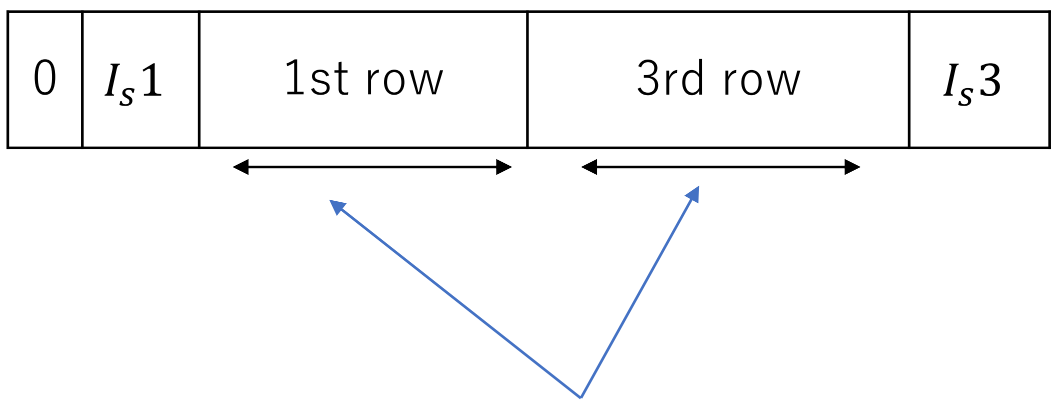
Configuration

number_of_register
11,11
number_of_operands
-,3 ← **ignored by option8**
option1
1 → **Each PI is used only once.**
option2
1 → **PIs are divided equally to nodes.**
option4
1 → **Data selection for registers are shared among nodes except first cycle.**
option5
2,4,6,8 → **“ among designated cycles.**
option6
2
option7
1 → **POs are divided equally to nodes.**
option8
0,1,0,1,0,1,1,0 → **LUT function is fixed to $\boldsymbol{x0 \oplus x1 \cdot x2}$.**
option11
-1 1 3,4,5,6 3,4,5,6 3,4,5,6 3,4,5,6 7,8,9,10 7,8,9,10 7,8,9,10 7,8,9,10 2
option12
0 0 4 5 6 7 8 9 10 11 0

Example 2

option11
-1 1 3,4,5,6 3,4,5,6 3,4,5,6 3,4,5,6 7,8,9,10 7,8,9,10 7,8,9,10 7,8,9,10 2

Node0 .inputs Is1
.inputs Is3
.inputs w11 w12 w13 w14 w31 w32 w33 w34

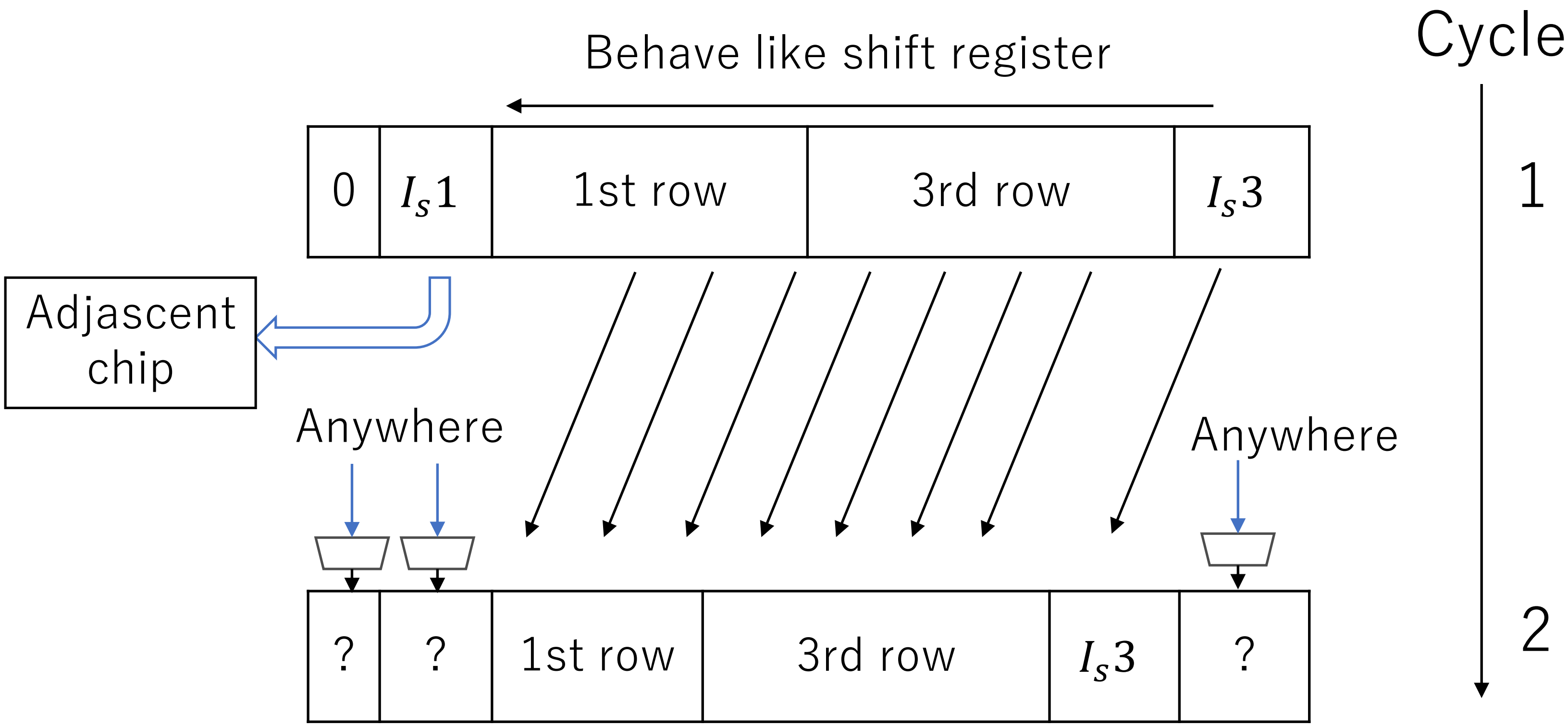


Order inside each box is defined by MUX

The same for node1.

Example 2

option6
2
option12
0 0 4 5 6 7 8 9 10 11 0



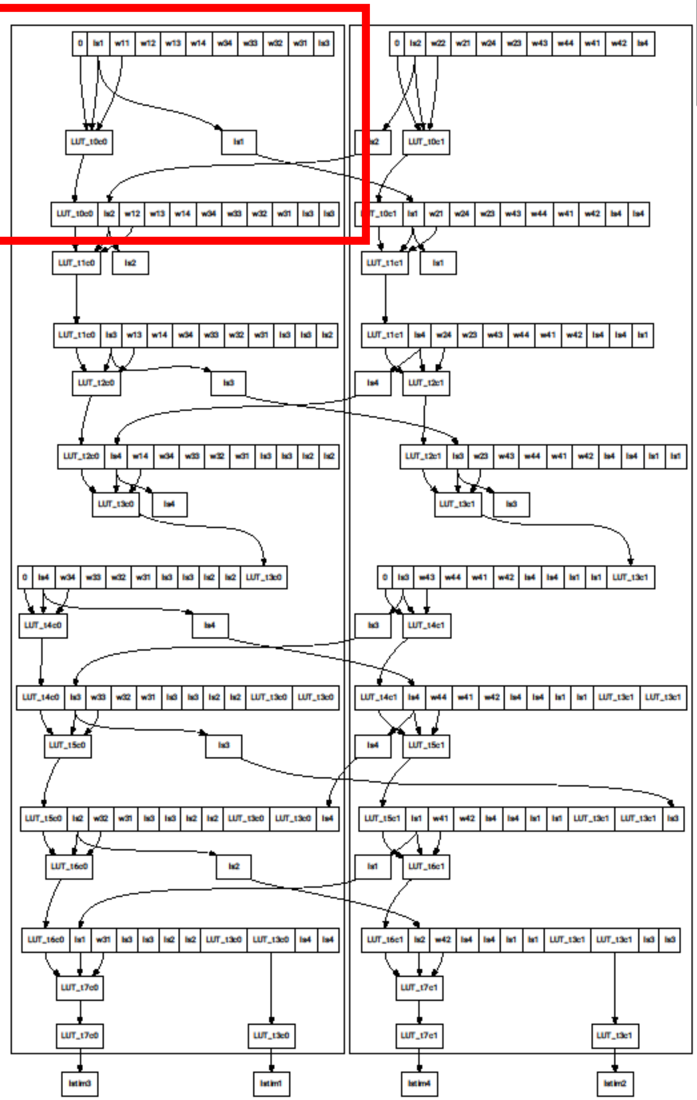
The same for the rest of cycles.

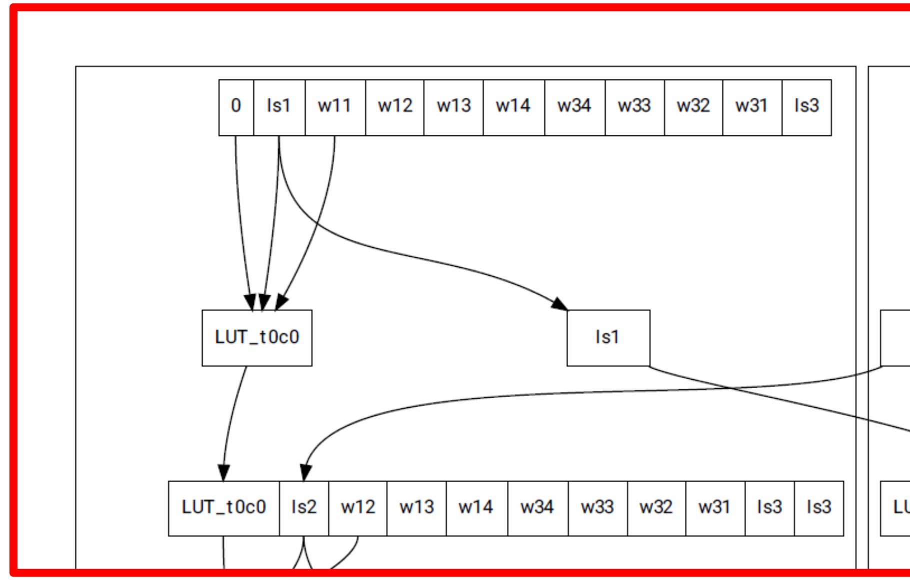
Example 2

time:976.863sec

OUTPUT

Input	LUT
0	0
1	1
2	0
3	1
4	0
5	1
6	1
7	0





Example 3

SPEC.blif

NOTICE that PIs are arranged in a particular order.

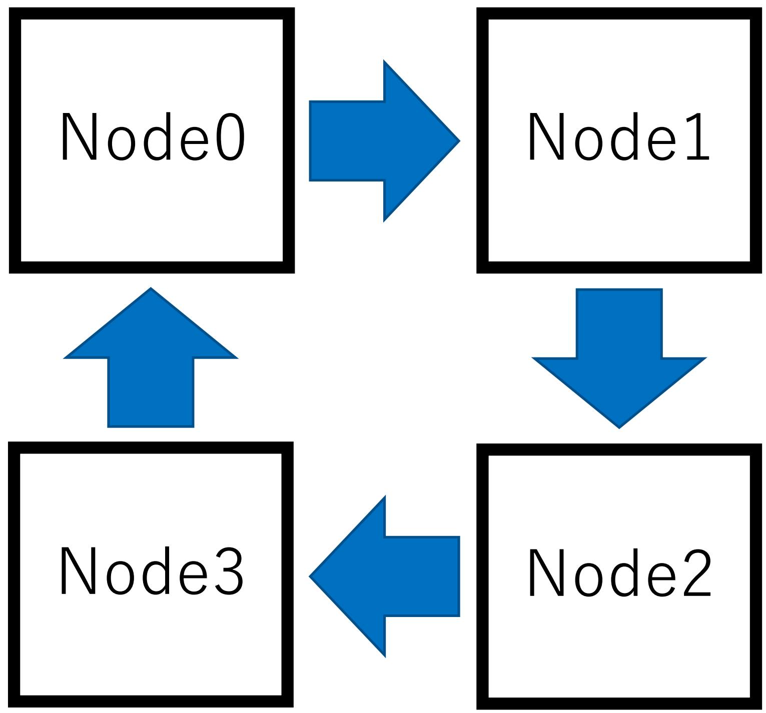
```
#.top matrix8
.model matrix8
.inputs ls1 ls5
.inputs w1_1 w1_4 w1_3 w1_2 w1_5 w1_8 w1_7 w1_6 w5_1 w5_4 w5_3 w5_2 w5_5 w5_8 w5_7 w5_6
.inputs ls2 ls6
.inputs w2_2 w2_1 w2_4 w2_3 w2_6 w2_5 w2_8 w2_7 w6_2 w6_1 w6_4 w6_3 w6_6 w6_5 w6_8 w6_7
.inputs ls3 ls7
.inputs w3_3 w3_2 w3_1 w3_4 w3_7 w3_6 w3_5 w3_8 w7_3 w7_2 w7_1 w7_4 w7_7 w7_6 w7_5 w7_8
.inputs ls4 ls8
.inputs w4_4 w4_3 w4_2 w4_1 w4_8 w4_7 w4_6 w4_5 w8_4 w8_3 w8_2 w8_1 w8_8 w8_7 w8_6 w8_5
.outputs lstim1 lstim2 lstim3 lstim4 lstim5 lstim6 lstim7 lstim8

.subckt and in0=ls1 in1=w1_1 out=ls1w11
.subckt and in0=ls2 in1=w1_2 out=ls2w12
.subckt and in0=ls3 in1=w1_3 out=ls3w13
.subckt and in0=ls4 in1=w1_4 out=ls4w14
.subckt and in0=ls5 in1=w1_5 out=ls5w15
.subckt and in0=ls6 in1=w1_6 out=ls6w16
.subckt and in0=ls7 in1=w1_7 out=ls7w17
.subckt and in0=ls8 in1=w1_8 out=ls8w18
.subckt xor8 in0=ls1w11 in1=ls2w12 in2=ls3w13 in3=ls4w14 in4=ls5w15 in5=ls6w16 in6=ls7w17
in7=ls8w18 out=lstim1
…
```

Example 3

Adjacency Matrix

0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0



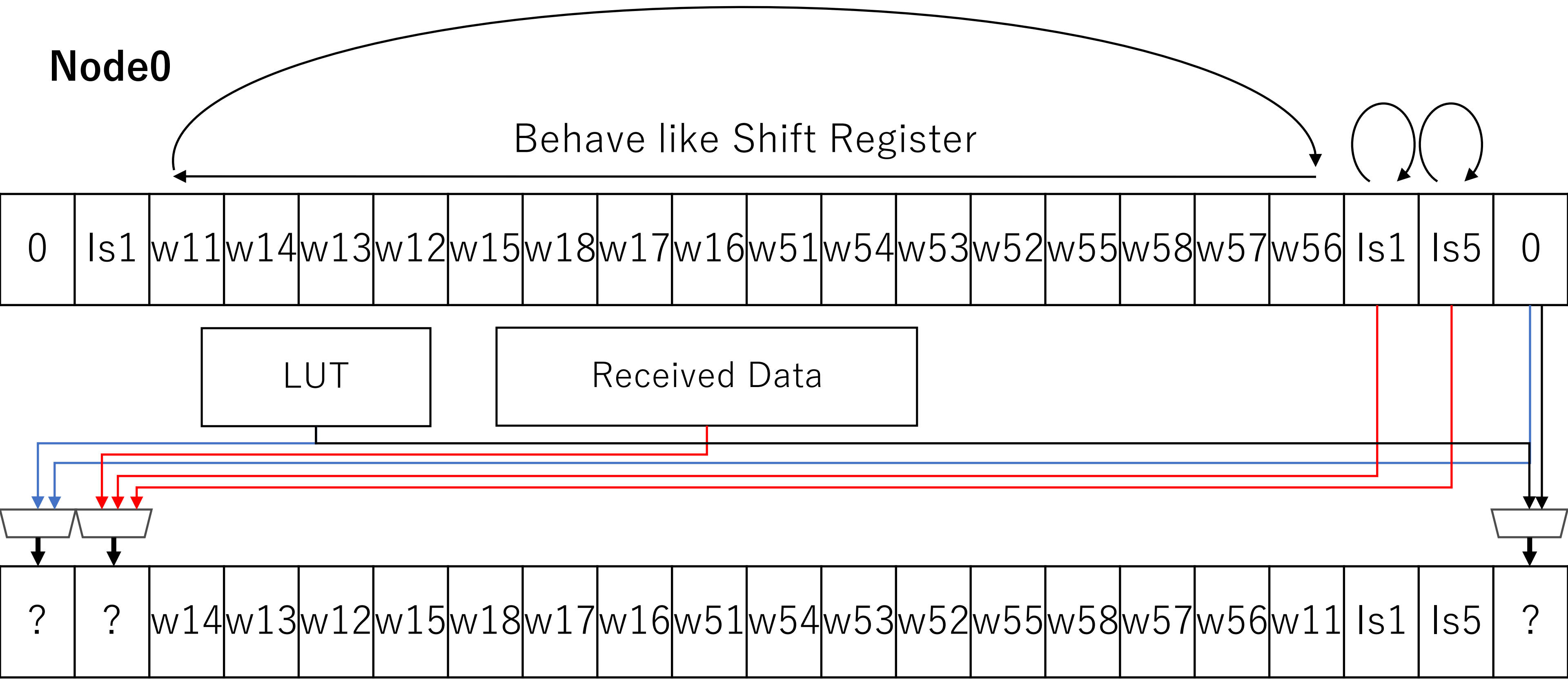
Example 3

Configuration

number_of_register
21,21
number_of_operands
-,3
option1
1
option2
1
option4
1
option5
2,3,4,6,7,8,10,11,12,14,15,16
option6
2
option7
1
option8
0,1,0,1,0,1,1,0
option11
-1 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 1 2 -1
option12
21,l 19,20,c 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 3 19 20 21,l
option13
l,21

Example 3

option11
-1 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 1 2 -1
option12
21,l 19,20,c 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 3 19 20 21,l

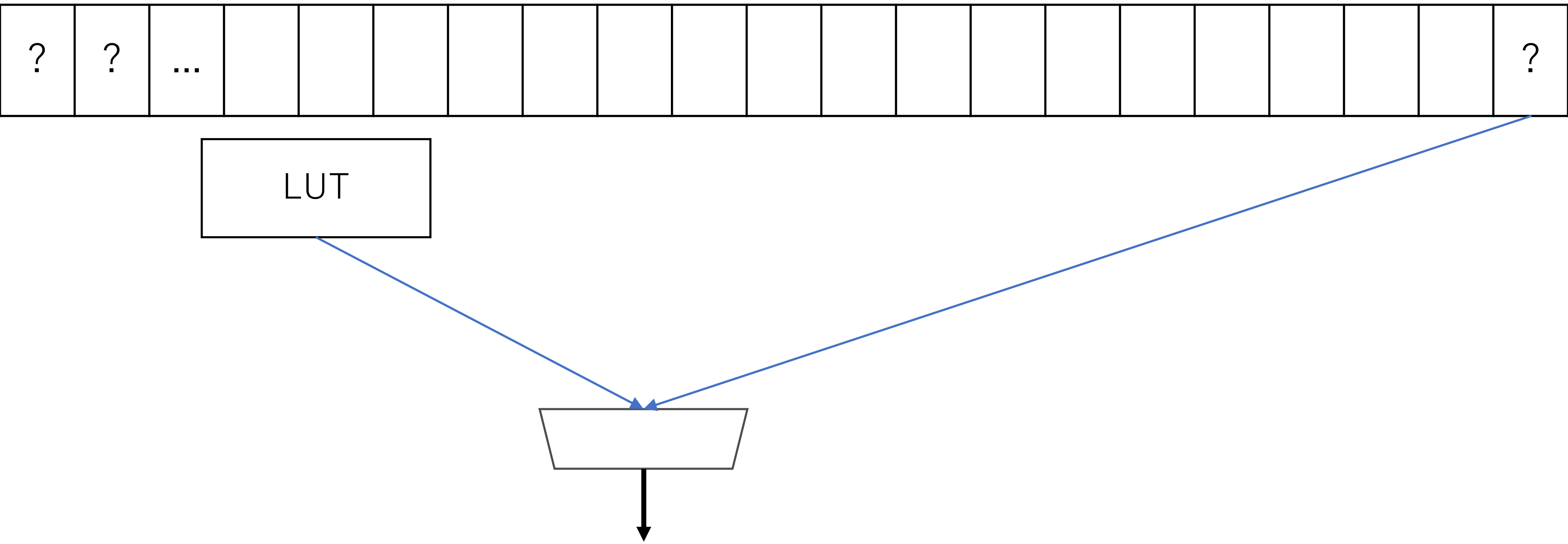


The same for the other node.

Example 3

Node0 at last cycle

option13
I,21



Istim1, Istim5

The same for the other node.

Example 3

OUTPUT time:0.821855

We already know almost all of the dataflow as expressed in option12,13.
Detail of the dataflow is filled in the result, and its correctness was proved.