Introduction of Processor Design for AI Applications

L07 – Resource Sharing (folding)

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

http://gr.xjtu.edu.cn/web/pengjuren

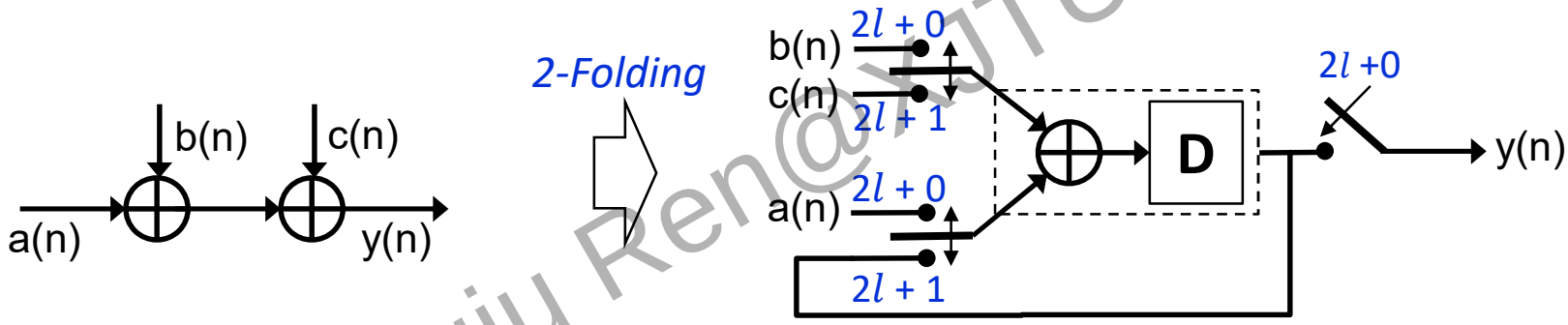
Folding (Resource Sharing)

- **Folding** transform is used to systematically determine the control circuits in data-stream architectures where multiple algorithm operations are time-multiplexed to a single functional unit
  - Trading area for time
  - Reducing the number of hardware functional units by a factor of **N** at the expense of increasing the computation time by a factor of **N**

Resource Sharing (folding)

Folding is a technique to reduce the silicon area by **time multiplexing** many algorithm operations into single functional units (such as adders and multipliers)

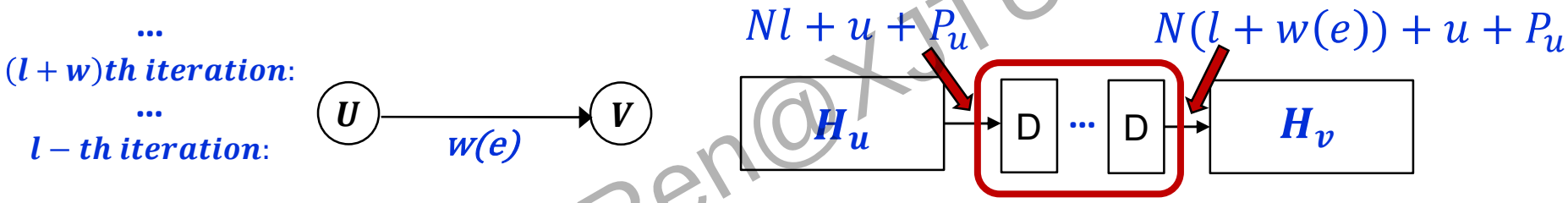
$$y(n) = a(n) + b(n) + c(n)$$



Cycle	Adder Input(left)	Adder Input(top)	System Output
0	$a(0)$	$b(0)$	-
1	$a(0) + b(0)$	$c(0)$	-
2	$a(1)$	$b(1)$	$a(0) + b(0) + c(0)$
3	$a(1) + b(1)$	$c(1)$	-
4	$a(2)$	$b(2)$	$a(1) + b(1) + c(1)$
5	$a(2) + b(2)$	$c(2)$	-

Folding Transformation

- N is the **folding factor** i.e., the number of operations folded to a single functional unit.
- $Nl + u$ and $Nl + v$ are respectively the time units at which l-th iteration of the nodes U and V are scheduled. u and v are called **folding orders** (time partition at which the node is scheduled to be executed) and satisfy $0 \leq u, \mathrm{v} \leq N - 1$



- H_u and H_v are functional units that execute u and v respectively. H_u is pipelined by P_u stages, U_l is available at $Nl + u + P_u$.
- Edge $U \xrightarrow{e} V$ has $w(e)$ delays => the output of l-th iteration of U (U_l) is used by $(l + \mathrm{w(e)})th$ iteration of node V, which is executed at $N(l + w(e)) + v$ So, the result should be stored for :

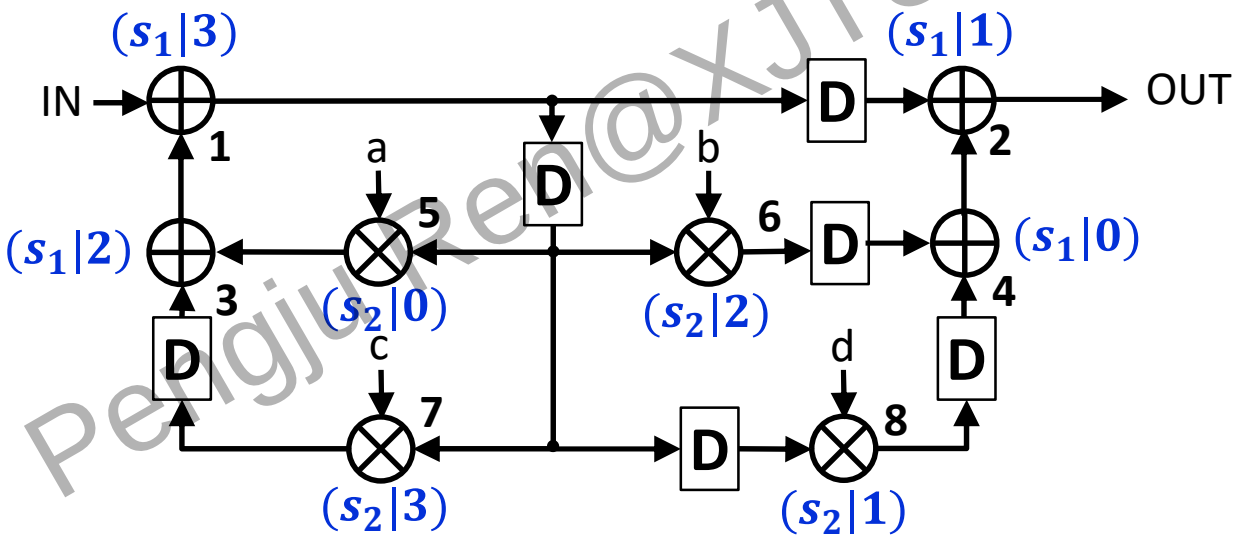
$$D_F\left(U \xrightarrow{e} V\right) = \left[\mathrm{N}(l + \mathrm{w(e)}) + v\right] - \left[\mathrm{N}l + u + P_u\right]$$

$$\Rightarrow D_F\left(U \xrightarrow{e} V\right) = \mathrm{Nw(e)} - P_u + v - u$$

Folding set and Biquad filter

Folding Set : An ordered set of N operations executed by the same functional unit. The operations are ordered from 0 to $N-1$. For example, Folding set $S_1 = \{A_1, \emptyset, A_2\}$ is for folding order $N=3$. A_1 has a folding order of 0 and A_2 of 2 and are respectively denoted by $(S_1|0)$ and $(S_2|2)$.

Example: Folding a retimed Biquad filter by $N=4$

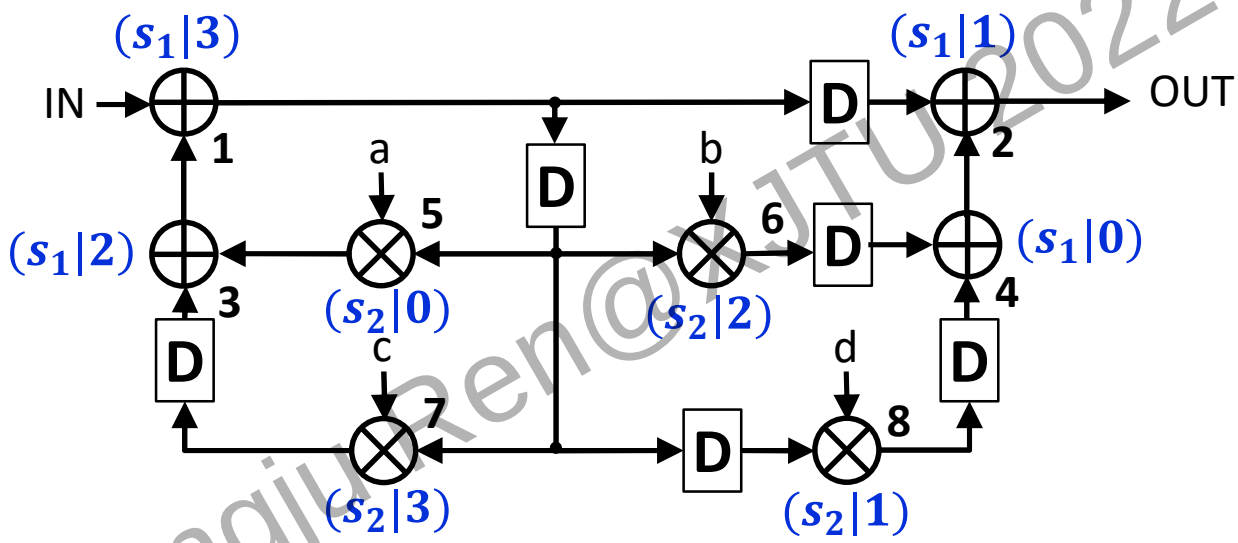


Addition time = 1u.t., Multiplication time = 2u.t., 1 stage pipelined adder and 2 stage pipelined multiplier (i.e., $P_A = 1$ and $P_M = 2$)

The folding sets are $S_1(Adder) = \{4,2,3,1\}$ and $S_2(Mult) = \{5,8,6,7\}$

Folding Transform — Biquad filter(1)

$$D_F\left(U \xrightarrow{e} V\right) = \mathrm{Nw(e)} - P_u + \mathrm{v} - \mathrm{u}$$

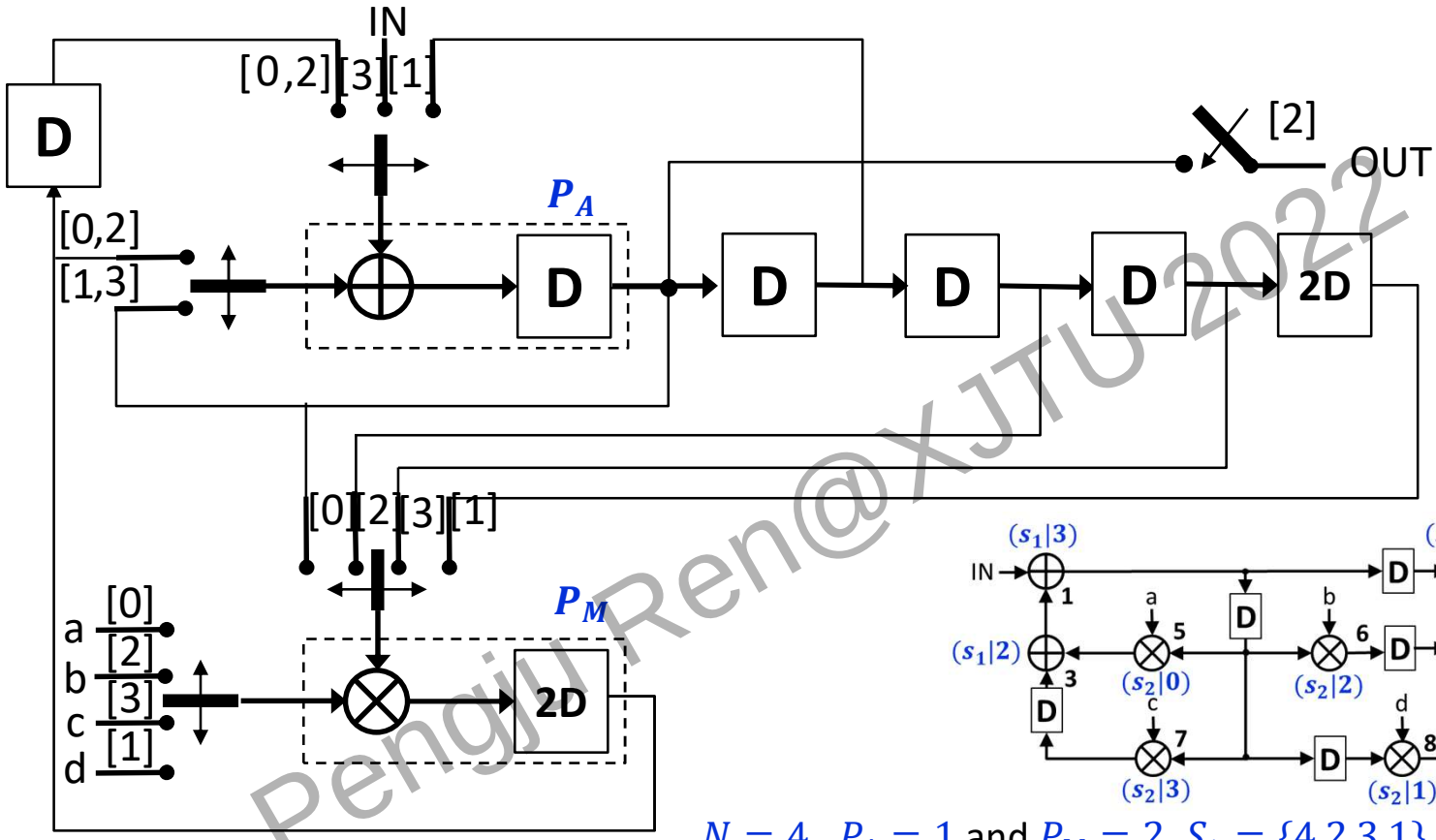


$N = 4,\ P_A = 1$ and $P_M = 2, S_1 = \{4,2,3,1\}$ and $S_2 = \{5,8,6,7\}$

$$D_F(1 \to 2) = 4(1) - 1 + 1 - 3 = 1$$
$$D_F(1 \to 6) = 4(1) - 1 + 2 - 3 = 2$$
$$D_F(1 \to 8) = 4(2) - 1 + 1 - 3 = 5$$
$$D_F(4 \to 2) = 4(0) - 1 + 1 - 0 = 0$$
$$D_F(6 \to 4) = 4(1) - 2 + 0 - 0 = 0$$
$$D_F(8 \to 4) = 4(1) - 2 + 0 - 1 = 1$$

$$D_F(1 \to 5) = 4(1) - 1 + 0 - 3 = 0$$
$$D_F(1 \to 7) = 4(1) - 1 + 3 - 3 = 3$$
$$D_F(3 \to 1) = 4(0) - 1 + 3 - 2 = 0$$
$$D_F(5 \to 3) = 4(0) - 2 + 2 - 0 = 0$$
$$D_F(7 \to 3) = 4(1) - 2 + 2 - 3 = 1$$

Folding Transform — Biquad filter(2)



$N = 4,\ P_A = 1$ and $P_M = 2, S_1 = \{4,2,3,1\}$ and $S_2 = \{5,8,6,7\}$

$D_F(1 \to 2) = 4(1) - 1 + 1 - 3 = 1$
$D_F(1 \to 6) = 4(1) - 1 + 2 - 3 = 2$
$D_F(1 \to 8) = 4(2) - 1 + 1 - 3 = 5$
$D_F(4 \to 2) = 4(0) - 1 + 1 - 0 = 0$
$D_F(6 \to 4) = 4(1) - 2 + 0 - 0 = 0$
$D_F(8 \to 4) = 4(1) - 2 + 0 - 1 = 1$

$D_F(1 \to 5) = 4(1) - 1 + 0 - 3 = 0$
$D_F(1 \to 7) = 4(1) - 1 + 3 - 3 = 3$
$D_F(3 \to 1) = 4(0) - 1 + 3 - 2 = 0$
$D_F(5 \to 3) = 4(0) - 2 + 2 - 0 = 0$
$D_F(7 \to 3) = 4(1) - 2 + 2 - 3 = 1$

Retiming for Folding (1)

For a folded system to be realizable $D_F(U \rightarrow V) \geq 0$ for all edges.

Once valid folding sets have been assigned, retiming can be used to either satisfy this property or determine that the folding sets are not feasible, that is , if $D'_F(U \rightarrow V)$ is the folded delays in the edge $U \rightarrow V$ for the retimed graph then $D'_F(U \rightarrow V) \geq 0$. So,

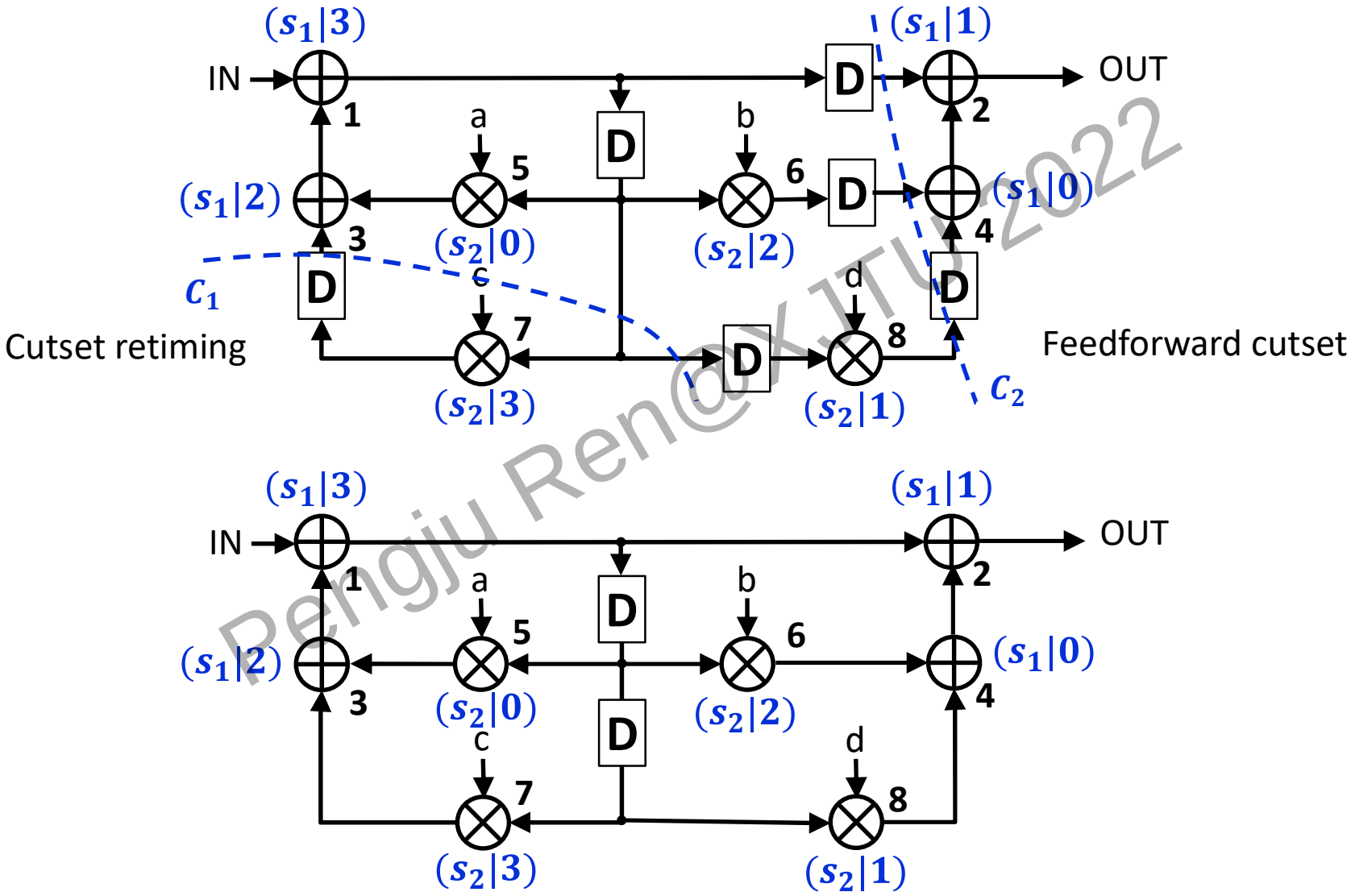
$Nw_r(e) - P_U + v - u \geq 0$ … where $w_r(e) = w(e) + r(V) - r(U)$

$\Rightarrow N(w(e) + r(V) - r(U)) - P_U + v - u \geq 0$

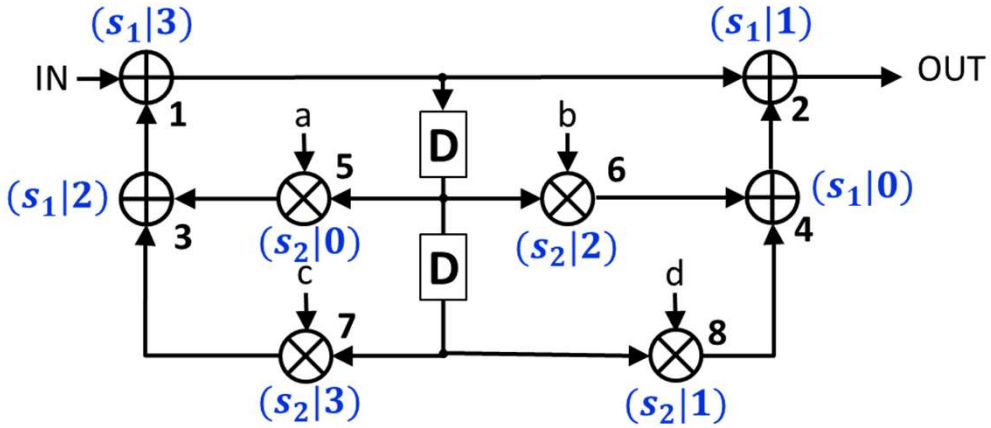
$\Rightarrow \; r(U) - r(V) \leq D_F(U \rightarrow V)/N$

$\Rightarrow \; r(U) - r(V) \leq \lfloor D_F(U \rightarrow V)/N \rfloor$

Retiming for Folding (2)



Retiming for Folding (3)

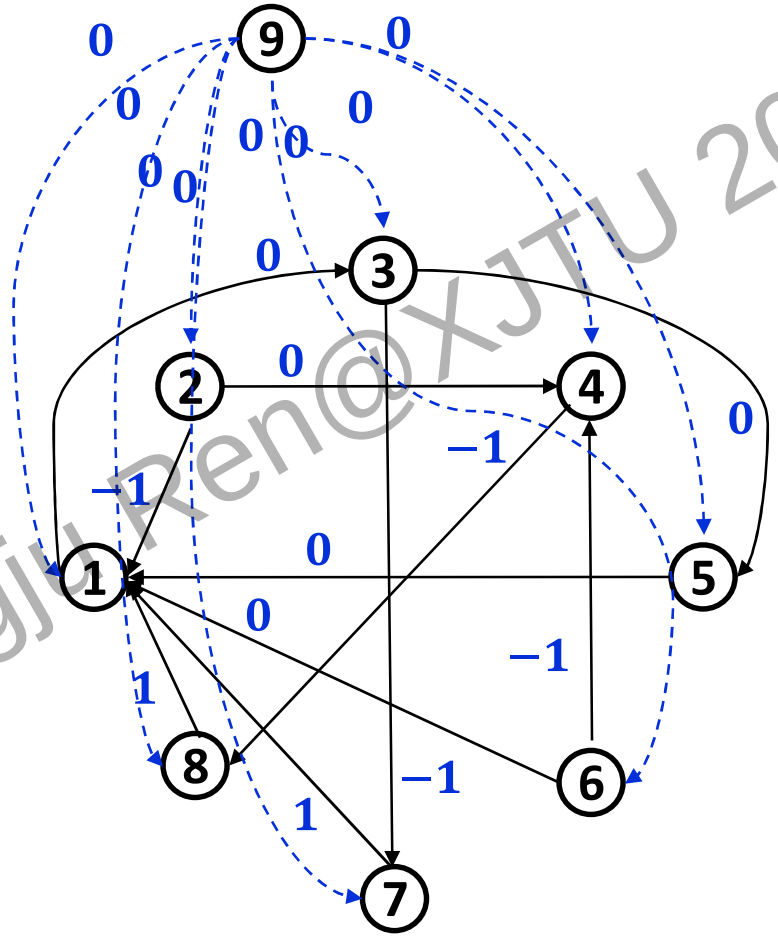


$$r(U) - r(V) \leq \lfloor D_F(U \rightarrow V)/N \rfloor$$

Edge	Folding Equations	Retiming for Folding Constraint
$1 \rightarrow 2$	$D_F(1 \rightarrow 2) = -3$	$r(1) - r(2) \leq -1$
$1 \rightarrow 5$	$D_F(1 \rightarrow 5) = 0$	$r(1) - r(5) \leq 0$
$1 \rightarrow 6$	$D_F(1 \rightarrow 6) = 2$	$r(1) - r(6) \leq 0$
$1 \rightarrow 7$	$D_F(1 \rightarrow 7) = 7$	$r(1) - r(7) \leq 1$
$1 \rightarrow 8$	$D_F(1 \rightarrow 8) = 5$	$r(1) - r(8) \leq 1$
$3 \rightarrow 1$	$D_F(3 \rightarrow 1) = 0$	$r(3) - r(1) \leq 0$
$4 \rightarrow 2$	$D_F(4 \rightarrow 2) = 0$	$r(4) - r(2) \leq 0$
$5 \rightarrow 3$	$D_F(5 \rightarrow 3) = 0$	$r(5) - r(3) \leq 0$
$6 \rightarrow 4$	$D_F(6 \rightarrow 4) = -4$	$r(6) - r(4) \leq -1$
$7 \rightarrow 3$	$D_F(7 \rightarrow 3) = -3$	$r(7) - r(3) \leq -1$
$8 \rightarrow 4$	$D_F(8 \rightarrow 4) = -3$	$r(8) - r(4) \leq -1$

Retiming for Folding (4)

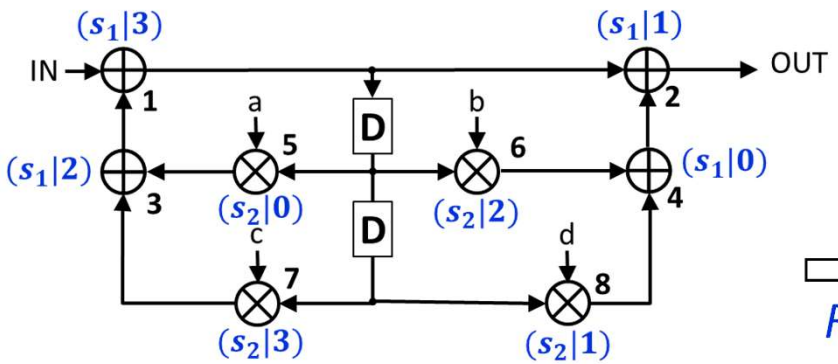
Retiming for Folding Constraint
$r(1) - r(2) \leq -1$
$r(1) - r(5) \leq 0$
$r(1) - r(6) \leq 0$
$r(1) - r(7) \leq 1$
$r(1) - r(8) \leq 1$
$r(3) - r(1) \leq 0$
$r(4) - r(2) \leq 0$
$r(5) - r(3) \leq 0$
$r(6) - r(4) \leq -1$
$r(7) - r(3) \leq -1$
$r(8) - r(4) \leq -1$



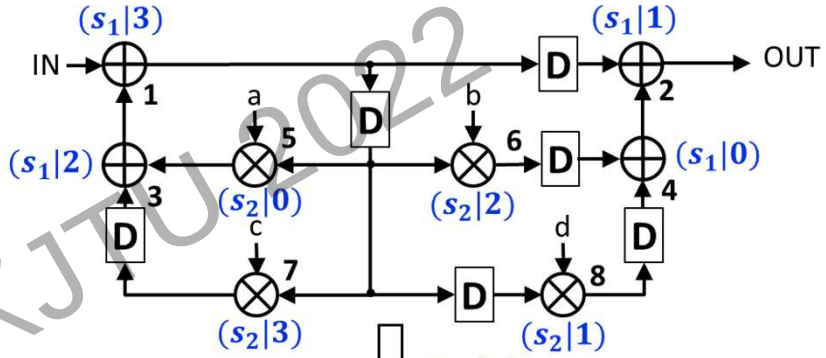
$r(1) = -1$
$r(2) = 0$
$r(3) = -1$
$r(4) = 0$
$r(5) = -1$
$r(6) = -1$
$r(7) = -2$
$r(8) = -1$

Retiming for Folding (5)

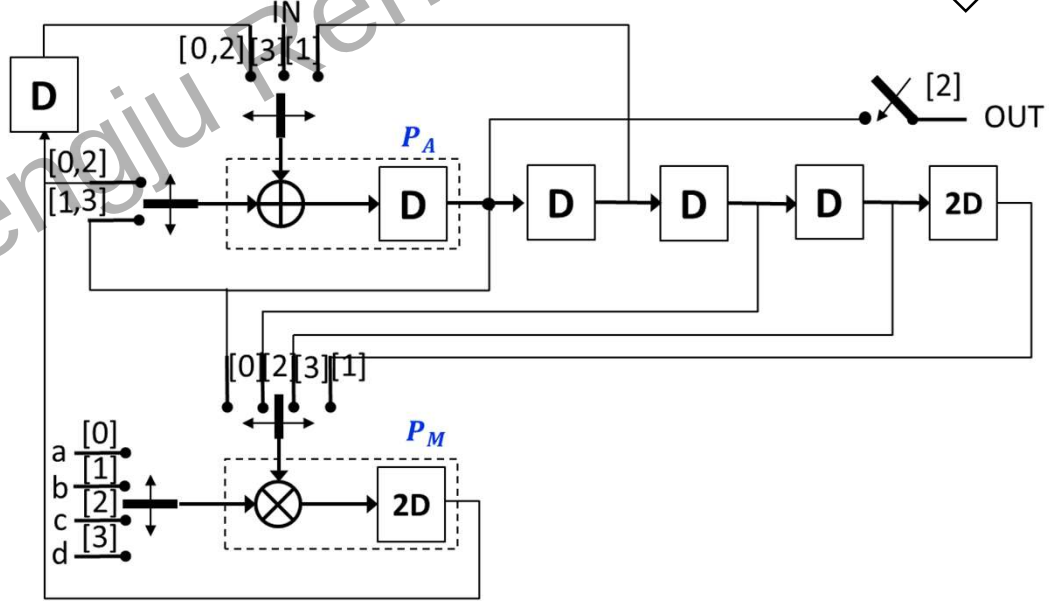
$r(1) = -1$
$r(2) = 0$
$r(3) = -1$
$r(4) = 0$
$r(5) = -1$
$r(6) = -1$
$r(7) = -2$
$r(8) = -1$



Retiming



Folding

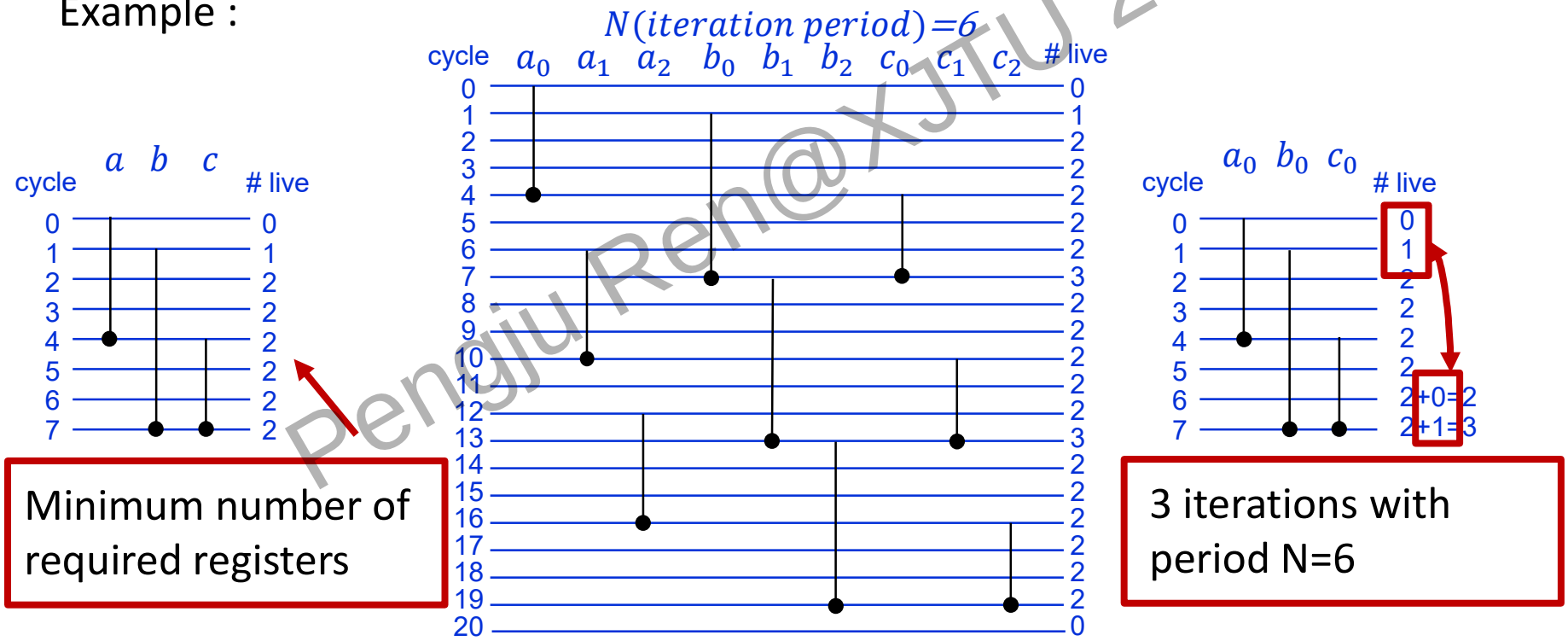


Register Minimization Technique (1)

Lifetime analysis is used for **register minimization techniques** in a Data-stream hardware. A 'data sample or variable' is live from the time it is produced through the time it is consumed. After that it is dead.

Linear lifetime chart : Represents the lifetime of the variables in a linear fashion.

Example :



Note : Linear lifetime chart uses the convention that the variable is not live during the clock cycle when it is produced but live during the clock cycle when it is consumed.

Register Minimization Technique (2)

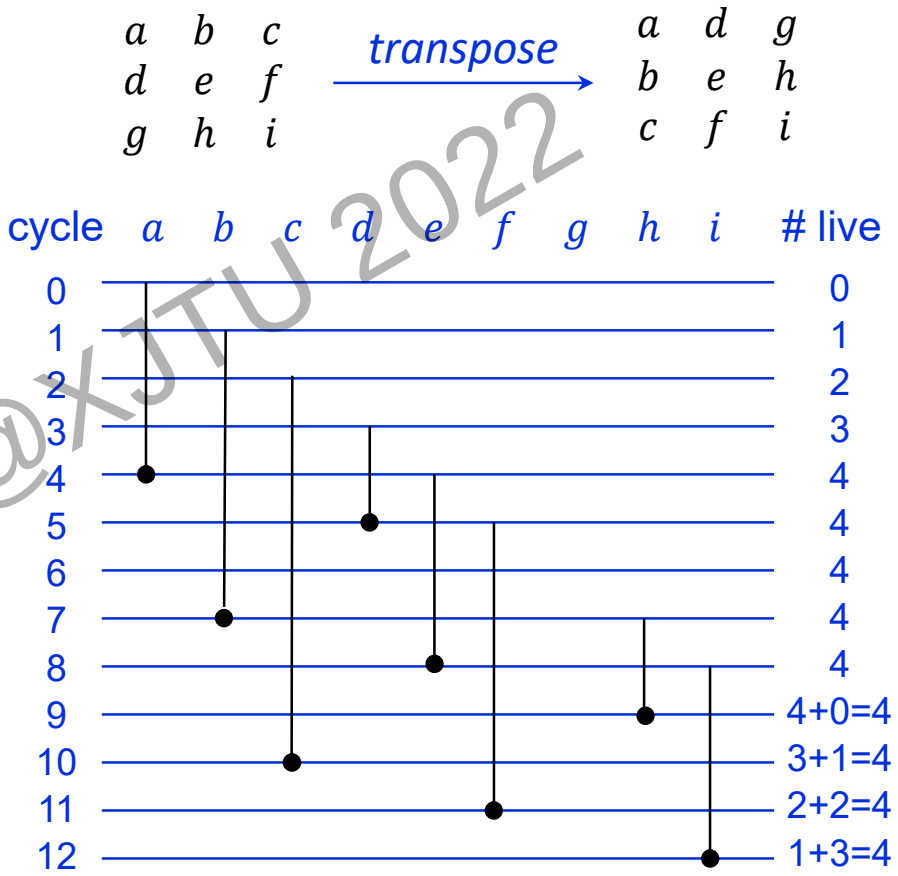
- Due to the periodic nature of Data-stream programs the lifetime chart can be drawn for only one iteration to give an indication of the # of registers that are needed. This is done as follows :
  - Let N be the iteration period
  - Let the # of live variables at time partitions $n \geq N$ be the # of live variables due to *0-th* iteration at cycles $n - kN$ for $k \geq 0$. In the example, # of live variables at cycle $7 \geq N(= 6)$ is the sum of the # of live variables due to the *0-th* iteration at cycles 7 and $(7 - 1 \times 6) = 1$, which is $2 + 1 = 3$.

Example: Register Minimization Technique

$+\ abs(-4)$

Sample	T_{in}	T^{*}_{zlout}	T_{diff}	T_{out}	Life
a	0	0	0	4	$0 \rightarrow 4$
b	1	3	2	7	$1 \rightarrow 7$
c	2	6	4	10	$2 \rightarrow 10$
d	3	1	-2	5	$3 \rightarrow 5$
e	4	4	0	8	$4 \rightarrow 8$
f	5	7	2	11	$5 \rightarrow 11$
g	6	2	-4	6	$6 \rightarrow 6$
h	7	5	-2	9	$7 \rightarrow 9$
i	8	8	0	12	$8 \rightarrow 12$

T_{zlout}: zero-latency output time

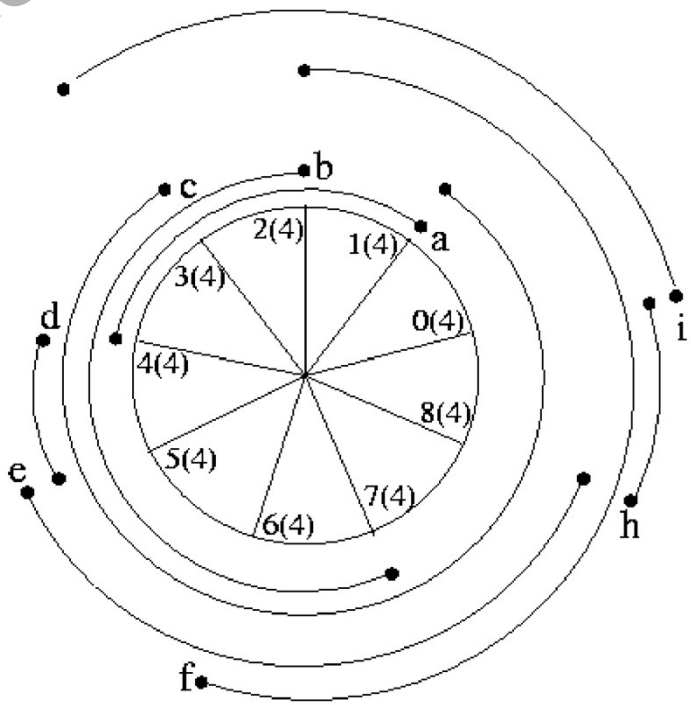


To make the system causal a latency of 4 is added to the difference so that T_{out} is the actual output time.

Circular lifetime chart

- Useful to represent the periodic nature of the data-stream programs.
- In a circular lifetime chart of periodicity N, the point marked $i(0 \leq i \leq N-1)$ represents the time partition i and all time instances $\{(Nl+i)\}$ where l is any non-negative integer.
- For example : If $N = 8$, then time partition $i = 3$ represents time instances $\{3,\ 11,\ 19\ldots\}$

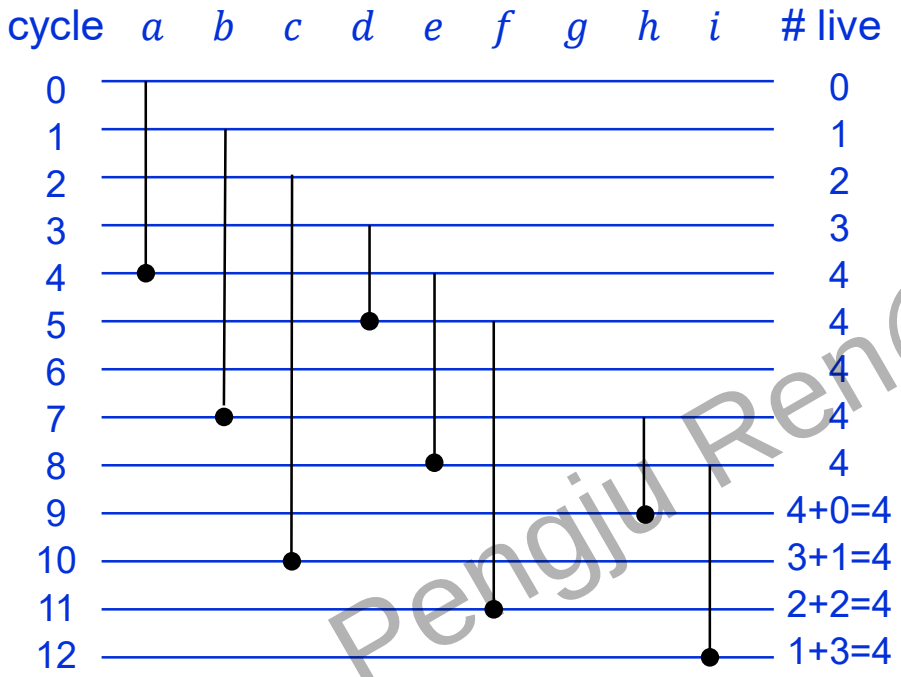
Note : Variable produced during time unit j and consumed during time unit k is shown to be alive from '$j+1'$ to 'k'. The numbers in the bracket in the adjacent figure correspond to the # of live variables at each time partition



Steps for Forward-Backward Register allocation

- Determine the minimum number of registers using lifetime analysis.
- Input each variable at the time step corresponding to the beginning of its lifetime. If multiple variables are input in a given cycle, these are allocated to multiple registers with preference given to the variable with the longest lifetime.
- Each variable is allocated in a forward manner until it is dead or it reaches the last register. In forward allocation, if the register i holds the variable in the current cycle, then register $i + 1$ holds the same variable in the next cycle. If $(i + 1)$-th register is not free then use the first available forward register.
- Being periodic the allocation repeats in each iteration. So hash out the register R_j for the cycle $l + N$ if it holds a variable during cycle l.
- For variables that reach the last register and are still alive, they are allocated in a backward manner on a first come first serve basis.
- Repeat steps 4 and 5 until the allocation is complete.

Example : Forward backward Register Allocation(1)

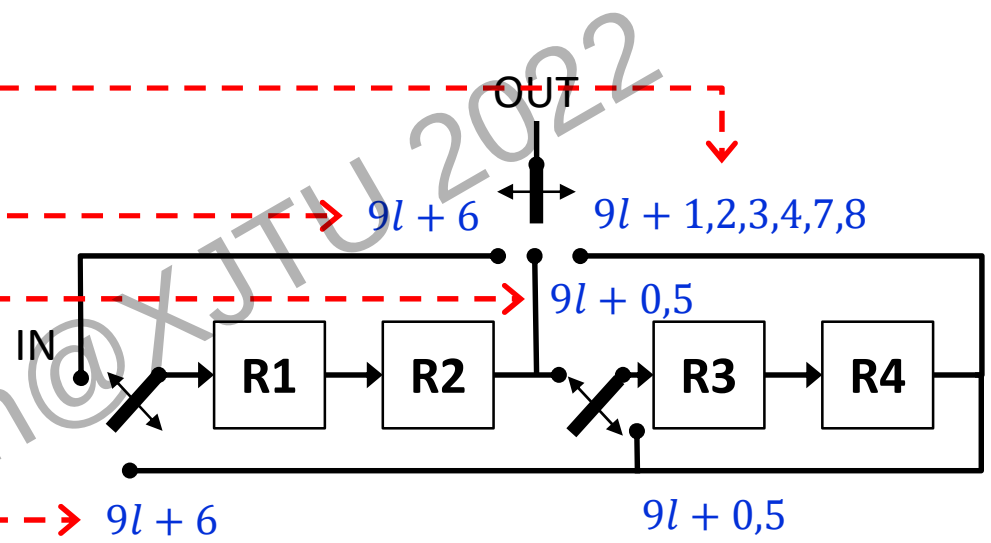


Cycle	Input	R1	R2	R3	R4	output
0	a					
1	b	a				
2	c	b	a			
3	d	c	b	a		
4	e	d	c	b	a	a
5	f	e	d	c	b	d
6	g	f	e	b	c	g
7	h	c	f	e	b	b
8	i	h	c	f	e	e
9		i	h	c	f	h
10			i	f	c	c
11				i	f	f
12					i	i

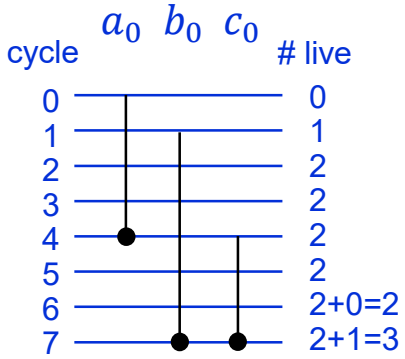
Note : Hashing is done to avoid conflict during backward allocation.

Example : Forward backward Register Allocation(2)

Cycle	Input	R1	R2	R3	R4	output
0	a					
1	b	a				
2	c	b	a			
3	d	c	b	a		
4	e	d	c	b	a	a
5	f	e	d	c	b	d
6	g	f	e	b	c	g
7	h	c	f	e	b	b
8	i	h	c	f	e	e
9		i	h	c	f	h
10			i	f	c	c
11				i	f	f
12					i	i

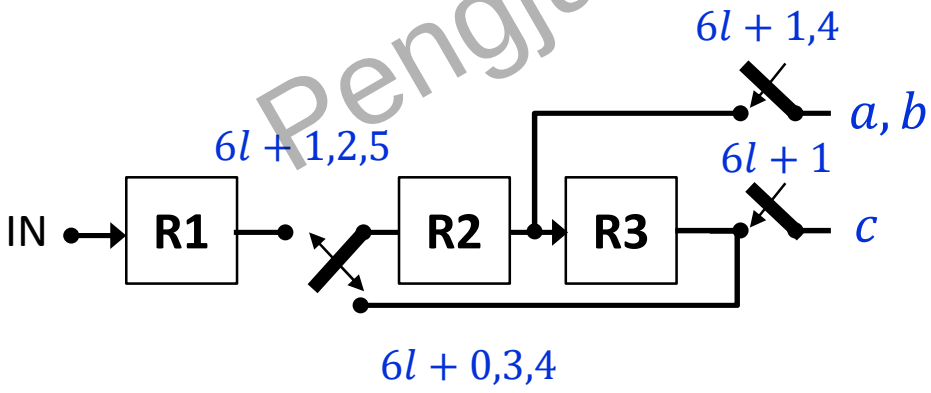


Example : Forward backward Register Allocation(3)



period N=6

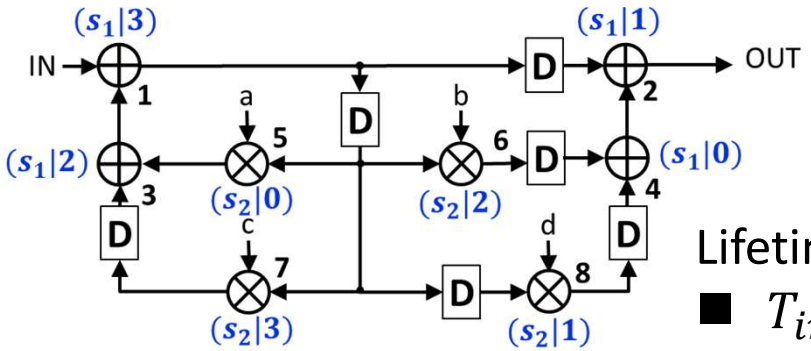
Cycle	Input	R1	R2	R3	output
0	a_0				
1	b_0	a_0			
2		b_0	a_0		
3			b_0	a_0	
4	c_0		a_0	b_0	a_0
5		c_0	b_0		
6	a_1		c_0	b_0	
7	b_1	a_1	b_0	c_0	b_0, c_0



Register minimization in folded architectures(1)

1. Perform retiming for folding
2. Write the folding equations
3. Use the folding equations to construct a lifetime table
4. Draw the lifetime chart and determine the required number of registers
5. Perform forward-backward register allocation
6. Draw the folded architecture that uses the minimum number of registers

Register minimization in folded architectures(2)



Lifetime table:

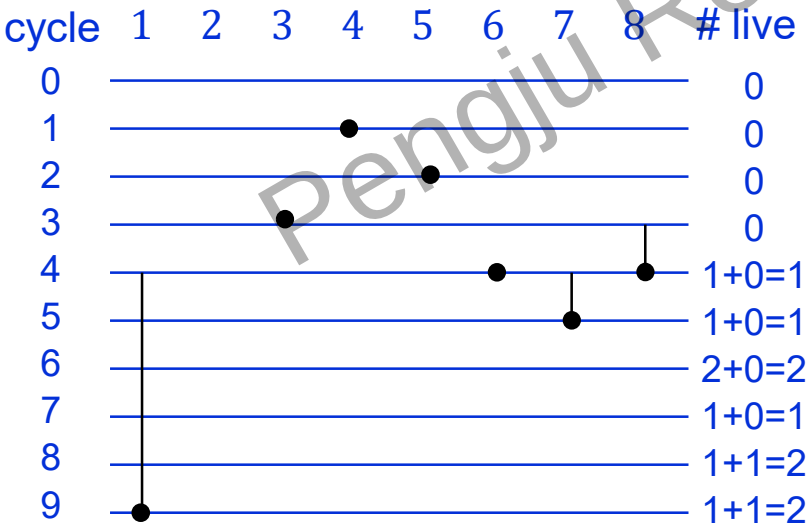
- T_{input} of node U is $u + P_u$
- T_{output} of the node U is $u + P_u + \max\{D_F(U \rightarrow V)\}$

$$D_F(1 \rightarrow 2) = 4(1) - 1 + 1 - 3 = 1$$
$$D_F(1 \rightarrow 5) = 4(1) - 1 + 0 - 3 = 0$$
$$D_F(1 \rightarrow 6) = 4(1) - 1 + 2 - 3 = 2$$
$$D_F(1 \rightarrow 7) = 4(1) - 1 + 3 - 3 = 3$$
$$D_F(1 \rightarrow 8) = 4(2) - 1 + 1 - 3 = 5$$
$$D_F(3 \rightarrow 1) = 4(0) - 1 + 3 - 2 = 0$$
$$D_F(4 \rightarrow 2) = 4(0) - 1 + 1 - 0 = 0$$
$$D_F(5 \rightarrow 3) = 4(0) - 2 + 2 - 0 = 0$$
$$D_F(6 \rightarrow 4) = 4(1) - 2 + 0 - 0 = 0$$
$$D_F(7 \rightarrow 3) = 4(1) - 2 + 2 - 3 = 1$$
$$D_F(8 \rightarrow 4) = 4(1) - 2 + 0 - 1 = 1$$

Node	$T_{input} \rightarrow T_{output}$
1	$4 \rightarrow 9$
2	-
3	$3 \rightarrow 3$
4	$1 \rightarrow 1$
5	$2 \rightarrow 2$
6	$4 \rightarrow 4$
7	$5 \rightarrow 6$
8	$3 \rightarrow 4$

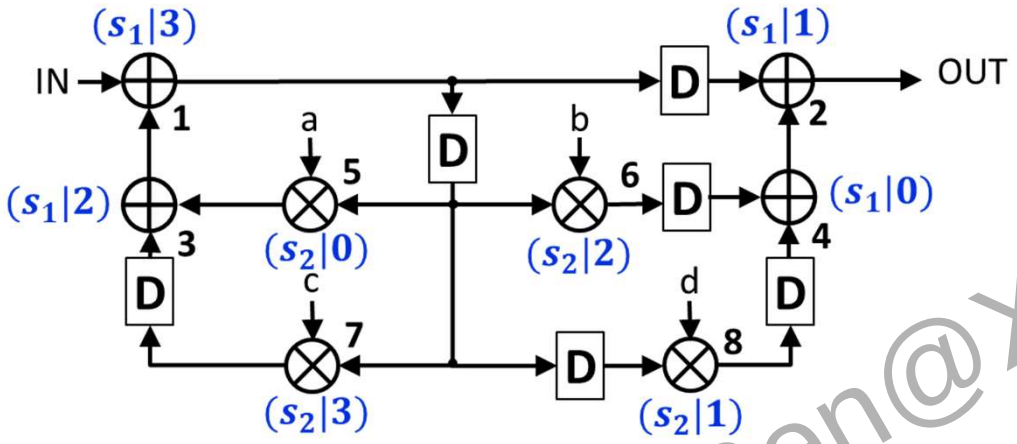
Register minimization in folded architectures(3)

Node	$T_{input} \rightarrow T_{output}$
1	$4 \rightarrow 9$
2	-
3	$3 \rightarrow 3$
4	$1 \rightarrow 1$
5	$2 \rightarrow 2$
6	$4 \rightarrow 4$
7	$5 \rightarrow 6$
8	$3 \rightarrow 4$



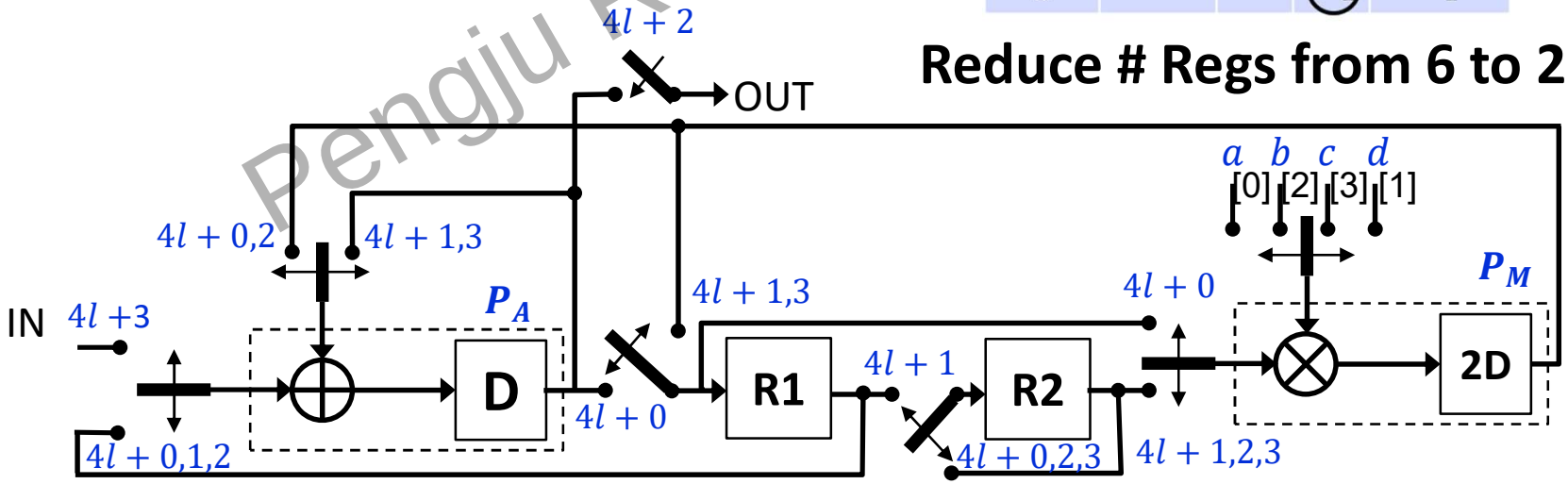
Cycle	Input	R1	R2	output
0				
1				
2				
3	n_8			
4	n_1	n_8		n_8
5	n_7	n_1		
6		n_7	n_1	n_7
7			n_1	
8			n_1	
9			n_1	n_1

Register minimization in folded architectures(4)



Cycle	Input	R1	R2	output
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

Reduce # Regs from 6 to 2



Next Lecture： Systolic Array