Chapter 8

Folding

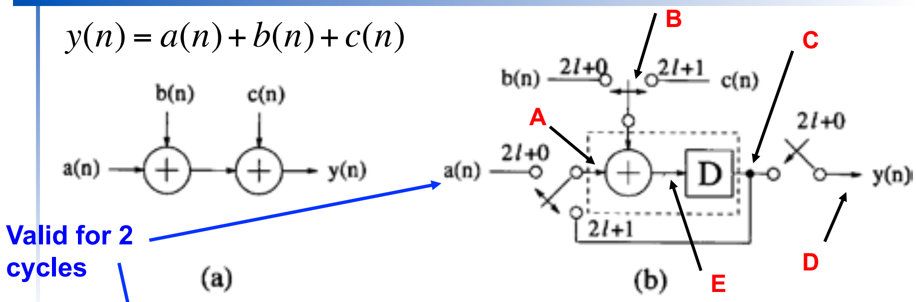
Instructor: Prof. Peter Lian
Department of Electrical Engineering & Computer Science
Lassonde School of Engineering
York University

Introduction to Folding

- Folding is a technique to reduce the silicon area by time-multiplexing many algorithm operations into single functional unit.
- The hardware is reduced by a factor of *N*, the time is increased by the same factor.
- In general, the data on the input of a folded realization is assumed to be valid for *N* cycles before changing.
- May lead to a large number of registers, thus need to minimize the registers.

A Folding Example

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



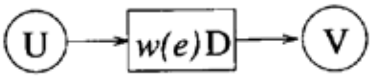
Cycle	A	B	E	C	D
0	a(0)	b(0)	a(0)+b(0)	—	—
1	a(0)+b(0)	c(0)	a(0)+b(0)+c(0)	a(0)+b(0)	—
2	a(1)	b(1)	a(1)+b(1)	a(0)+b(0)+c(0)	a(0)+b(0)+c(0)
3	a(1)+b(1)	c(1)	a(1)+b(1)+c(1)	a(1)+b(1)	—
4	a(2)	b(2)	a(2)+b(2)	a(1)+b(1)+c(1)	a(1)+b(1)+c(1)
5	a(2)+b(2)	c(2)	a(2)+b(2)+c(2)	a(2)+b(2)	—
6	a(3)	b(3)	a(3)+b(3)	a(2)+b(2)+c(2)	a(2)+b(2)+c(2)

Folding Transformation

- The objective is to provide a systematic technique for designing control circuits for hardware where several algorithm operations are mapped to the same piece of hardware via time-multiplexing of course.
- The folding transformation starts with a DFG for the algorithm.

Definitions in Folding Transformation

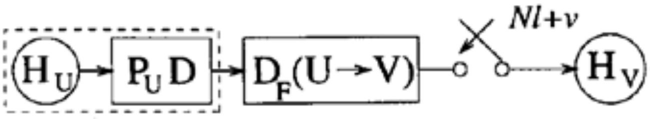
- U and V are two nodes in the original DFG.
- U and V are connected via an edge e with a delay $w(e)$, i.e. $U \xrightarrow{w(e)} V$



- Folding factor is N
- Node U (computation): l^{th} iteration is performed at time $Nl + u$
- Node V (computation): l^{th} iteration is performed at time $Nl + v$
- u and v are folding order of the nodes U and V that satisfy $0 \leq u, v \leq N-1$

Definitions in Folding Transformation

- H_u and H_v are the hardware units used at U and V for performing computation.
- H_u and H_v are pipelined by P_u and P_v stages



Folding Transformation

- The results of the l^{th} iteration of node U is available at $Nl+u+P_u$
- Since there are $w(e)$ delays between U and V, the result of the l^{th} iteration of the node U is used by the $(l+w(e))^{th}$ iteration of the node V, which is executed at $N(l+w(e))+v$.
- The result must be stored for

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

Folding Set

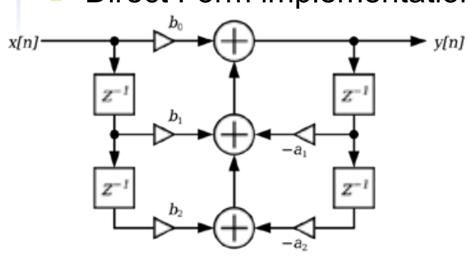
- Is an ordered set of operations executed by the same functional unit.
- Each folding set contains N entries (some of which may be null operations)
- The j^{th} position within the folding set is executed during the time partition j
  - For example the folding set $S_1 = \{A_1, 0, A_2\}$ for $N=3$
  - A_1 is performed during the 0^{th} time partition $S_1|0$, while A_2 is done in the 2^{nd} time partition $S_1|2$
  - Due to null operation in position 1, the function unit is not used at time instances $3l+1$
- Folding set is obtained using a scheduling and allocation algorithm

Example of a Biquad Filter

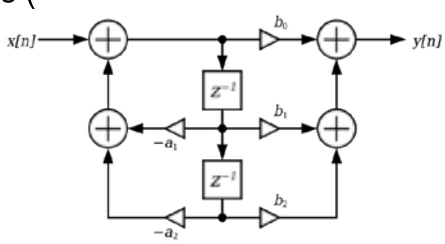
- A biquad filter is a second-order recursive filter.
- The z-transform transfer function is:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

- Direct Form implementations (



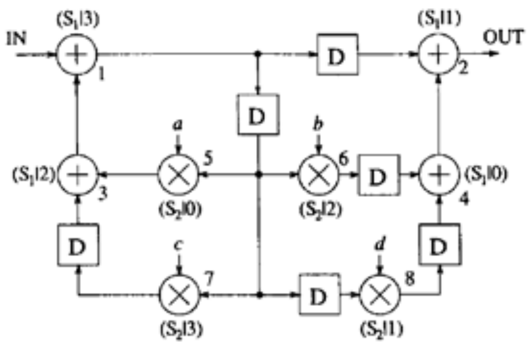
Direct Form 1



Direct Form 2

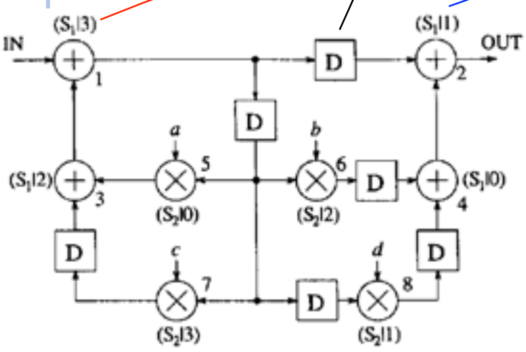
Example of a Biquad Filter

- N=4, the iteration period of the folded hardware is 4 u.t., i.e. each node is executed exactly once every 4 u.t.
- Folding set is given for an adder S_1={4,2,3,1} and a multiplier S_2={5,8,6,7}
- T_A=1 u.t., T_M=2 u.t., and P_a=1, P_m=2



Writing Folding Equations

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



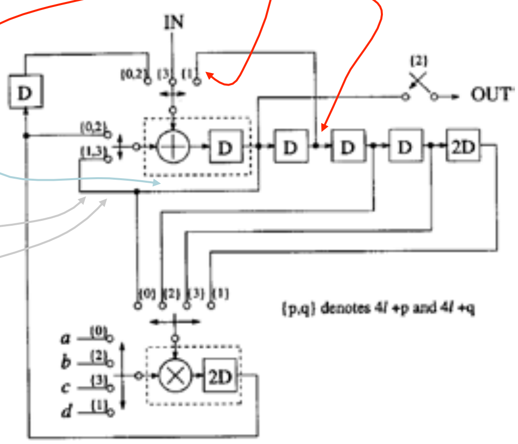
$$
\begin{aligned}
D_F(1 \to 2) &= 4(1) - \boxed{1} + 1 - 3 = 1\\
D_F(1 \to 5) &= 4(1) - \boxed{1} + 0 - 3 = 0\\
D_F(1 \to 6) &= 4(1) - \boxed{1} + 2 - 3 = 2\\
D_F(1 \to 7) &= 4(1) - \boxed{1} + 3 - 3 = 3\\
D_F(1 \to 8) &= 4(2) - \boxed{1} + 1 - 3 = 5\\
D_F(3 \to 1) &= 4(0) - \boxed{1} + 3 - 2 = 0\\
D_F(4 \to 2) &= 4(0) - \boxed{1} + 1 - 0 = 0\\
D_F(5 \to 3) &= 4(0) - \boxed{2} + 2 - 0 = 0\\
D_F(6 \to 4) &= 4(1) - \boxed{2} + 0 - 2 = 0\\
D_F(7 \to 3) &= 4(1) - \boxed{2} + 2 - 3 = 1\\
D_F(8 \to 4) &= 4(1) - \boxed{2} + 0 - 1 = 1.
\end{aligned}
$$

Folded Biquad Filter

Nodes 1, 2, 3, 4 are adders
Nodes 5, 6, 7, 8 are multipliers

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

$$
\begin{aligned}
D_F(1 \to 2) &= 4(1) - 1 + 1 - 3 = 1\\
D_F(1 \to 5) &= 4(1) - 1 + 0 - 3 = 0\\
D_F(1 \to 6) &= 4(1) - 1 + 2 - 3 = 2\\
D_F(1 \to 7) &= 4(1) - 1 + 3 - 3 = 3\\
D_F(1 \to 8) &= 4(2) - 1 + 1 - 3 = 5\\
D_F(3 \to 1) &= 4(0) - 1 + 3 - 2 = 0\\
D_F(4 \to 2) &= 4(0) - 1 + 1 - 0 = 0\\
D_F(5 \to 3) &= 4(0) - 2 + 2 - 0 = 0\\
D_F(6 \to 4) &= 4(1) - 2 + 0 - 2 = 0\\
D_F(7 \to 3) &= 4(1) - 2 + 2 - 3 = 1\\
D_F(8 \to 4) &= 4(1) - 2 + 0 - 1 = 1.
\end{aligned}
$$



Folding Condition

- For a folded system to be realizable, $D_F \geq 0$ must hold for all of the edges in the DFG.
- What if some of the D_F's are negative?
  - Of course we can not implement it.
- We can apply retiming to the original graph to get a valid D_F's
- Recall, retiming equation U → V

$$w_r(e) = w(e) + r(V) - r(U) \geq 0$$

Retiming for Folding

$D'_F(U \xrightarrow{e} V)$ is the delays in the folded retimed graph

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

$$D'_F(U \xrightarrow{e} V) = N\big(w(e) + r(V) - r(U)\big) - P_u + v - u$$

$$D'_F(U \xrightarrow{e} V) = Nw(e) - P_u + v - u + Nr(V) - Nr(U)$$

$$D'_F(U \xrightarrow{e} V) = D_F(U \xrightarrow{e} V) + Nr(V) - Nr(U) \geq 0$$

$$r(U) - r(V) \leq \frac{D_F(U \xrightarrow{e} V)}{N}$$

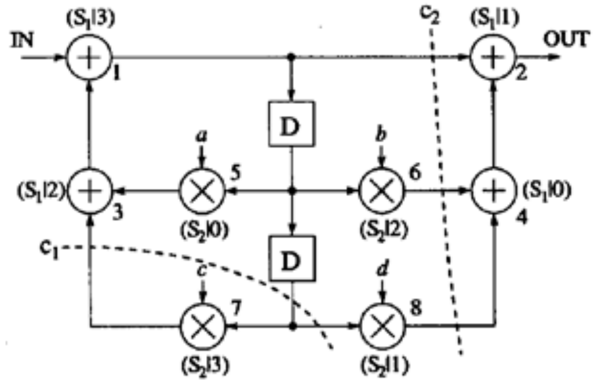
$$r(U) - r(V) \leq \left\lfloor \frac{D_F(U \xrightarrow{e} V)}{N} \right\rfloor$$

Retiming for Folding

- We can use the techniques in Chapter 4 to solve for retiming.
- Then we fold the retimed DFG

Activity 1

Given the biquad filter below, (1) find folding equations, (2) can it be folded? If not, retiming it.



Register Minimization

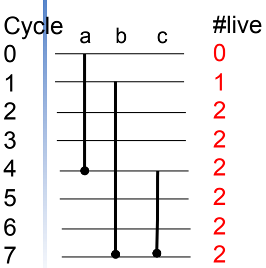
Registers Minimization Techniques

- The objective is to minimize the number of registers in the implementation of a DSP algorithm.
- Techniques involved
  - Life time analysis
  - Data allocation using forward-backward register allocation
  - Register minimization in folded architecture

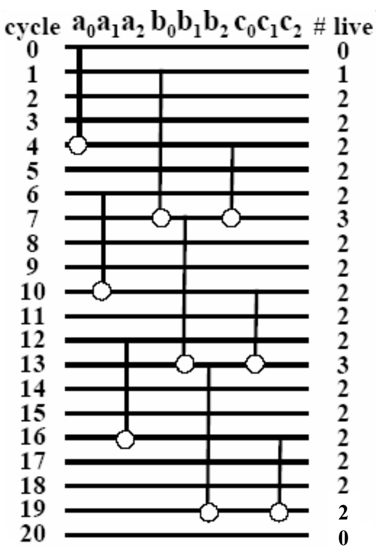
Life Time Analysis

- A data sample (variable) is alive from the time it is produced, until the time it is consumed (dead).
- During that time, the variable is stored in a register.
- The maximum number of live variables at any time is the minimum number of registers required for the implementation.
- We use the convention that the variable is not alive during the cycle it is produced in, and alive during the cycle it is consumed in.

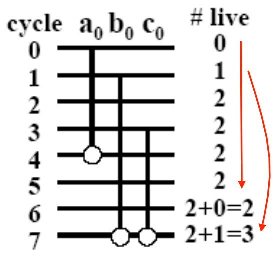
Linear Lifetime Chart



Linear lifetime chart, N=6



Showing 3 periods



Periodic with a period of N=6

The number of live variables at time partition n>=N, is the sum of the live variables due to the 0^{th} iteration at cycles n and n-kN for 0<=k<=n-k

Example

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

i h g f e d c b a → **Matrix Transpose** → i f c h e b g d a

Example

Output time with zero latency

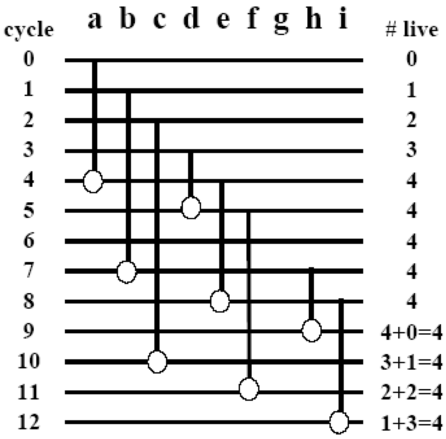
$$T_{diff} = T_{zlout} - T_{input}$$

Sample	T_{input}	T_{zlout}	T_{diff}	T_{output}	Life
a	0	0	0	4	0→4
b	1	3	2	7	1 →7
c	2	6	4	10	2 →10
d	3	1	-2	5	3 →5
e	4	4	0	8	4 →8
f	5	7	2	11	5 →11
g	6	2	-4	6	6 →6
h	7	5	-2	9	7 →9
i	8	8	0	12	8 →12

$$+T_{lat} = -(-4)$$

The latency, T_{lat}, is the most negative value of T_{diff}

Example



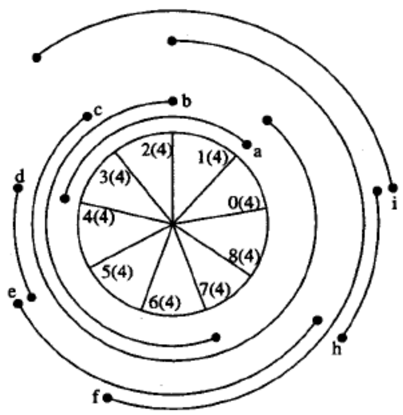
Sample	Life period
a	0→4
b	1 →7
c	2 →10
d	3 →5
e	4 →8
f	5 →11
g	6 →6
h	7 →9
i	8 →12

N=9

n-kN=9-1*9=0
n-kN=10-1*9=1
n-kN=11-1*9=2
n-kN=12-1*9=3

Circular Life-Time Chart

- Useful to represent the periodic nature of the DSP programs
- The point marked I ($0 \le i \le N-1$) represents the time partition I and all time instances $Nl+I$
- 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 number of live variable at each time partition.



Data Allocation

Step 1: Determine the min. number of registers using lifetime analysis

Step 2: Input each variable at the time step corresponding to the beginning of its lifetime. If multiple variables are input in a given cycle, use multiple registers such that the variable with longest lifetime is allocated to the initial register.

Step 3: Each variable is allocated in a forward manner until it is dead or reaches the last register

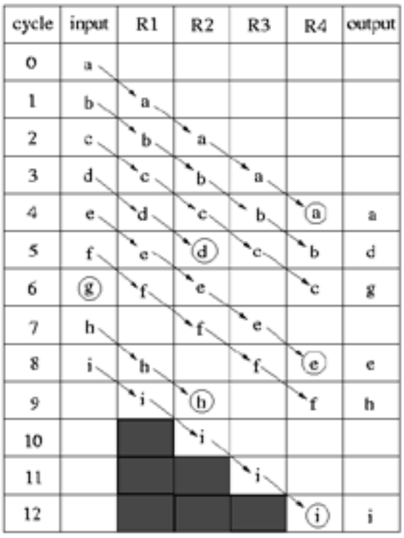
Data Allocation

Step 4: Since the allocation is periodic, the allocation of the current iteration also repeats itself after N.

Step 5: For variables that reach the last register and not yet dead, the remaining life period is calculated, and these variables are allocated to a register in a backward manner on a first-come first-served basis. (if more than one, choose one that has been allocated backward before, then forward again and so on).

Step 6: Repeat steps 4 and 5 until the allocation is complete.

Data Allocation Example



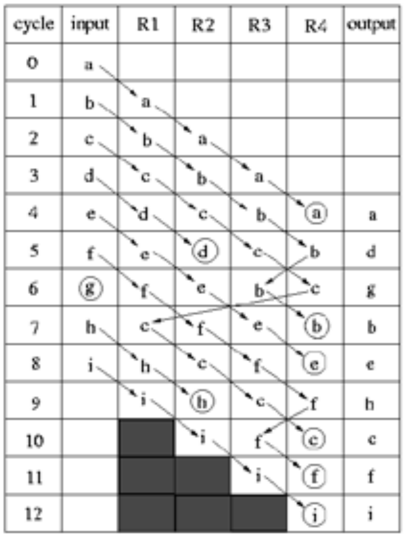
Sample	Life period
a	0→4
b	1 →7
c	2 →10
d	3 →5
e	4 →8
f	5 →11
g	6 →6
h	7 →9
i	8 →12

Note: Hashing is done to avoid conflict during backward allocation

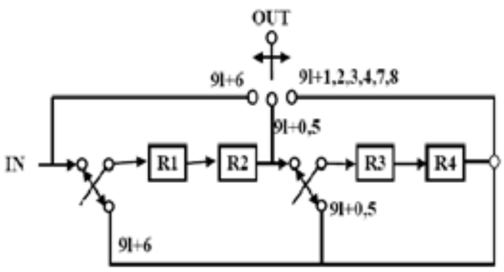
Data Allocation Example

Sample	Life period
a	0→4
b	1 →7
c	2 →10
d	3 →5
e	4 →8
f	5 →11
g	6 →6
h	7 →9
i	8 →12

Note: Hashing is done to avoid conflict during backward allocation



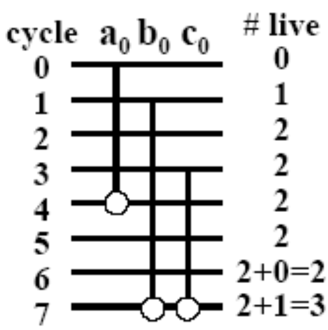
Synthesized Architecture



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

Activity 2

Given the linear lifetime chart below, (1) derive the data allocation using forward-backward register allocation; (2) synthesis the architecture.



Register Minimization for Folded Architecture

Step 1: Perform retiming for folding

Step 2: Write the folding equations

Step 3: Use the folding equations to construct a lifetime table

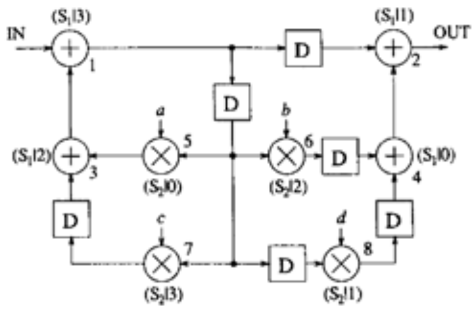
Step 4: Draw the lifetime chart and determine the minimum number of registers

Step 5: Perform forward-backward register allocation

Step 6: Draw the folded architecture that uses the minimum number of registers

Example: Biquad Filter

- Consider the Biquad filter example presented in slide 10.



- Steps 1 & 2 have already been done.

Step 3: Construct a lifetime table

Node U produces data at time $u+P_u : T_{input} = u + P_u$,

T_{out} for node $U = u + P_u + \max_V \{D_F(U \to V)\}$

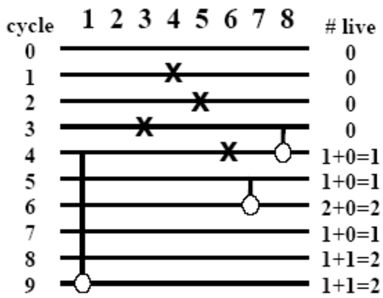
$D_F(U,V)=Nw(e)-P_u+v-u$

$$
\begin{aligned}
D_F(1\to 2) &= 4(1)-1+1-3=1\\
D_F(1\to 5) &= 4(1)-1+0-3=0\\
D_F(1\to 6) &= 4(1)-1+2-3=2\\
D_F(1\to 7) &= 4(1)-1+3-3=3\\
D_F(1\to 8) &= 4(2)-1+1-3=5\\
D_F(3\to 1) &= 4(0)-1+3-2=0\\
D_F(4\to 2) &= 4(0)-1+1-0=0\\
D_F(5\to 3) &= 4(0)-2+2-0=0\\
D_F(6\to 4) &= 4(1)-2+0-2=0\\
D_F(7\to 3) &= 4(1)-2+2-3=1\\
D_F(8\to 4) &= 4(1)-2+0-1=1.
\end{aligned}
$$

Example: Node 1 is created at time 4 and consumed at 4+max(1,0,2,3,5)=9

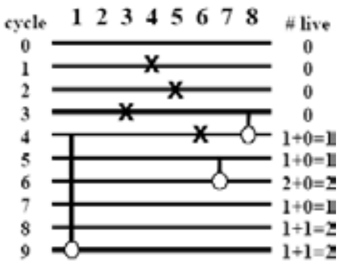
Node	$T_{input} \to T_{output}$
1	$4 \to 9$
2	-----
3	$3 \to 3$
4	$1 \to 1$
5	$2 \to 2$
6	$4 \to 4$
7	$5 \to 6$
8	$3 \to 4$

Step 4: Construct lifetime chart



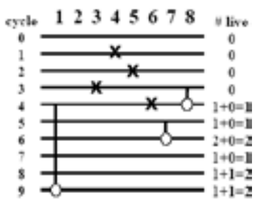
Node	$T_{input} \to T_{output}$
1	$4 \to 9$
2	-----
3	$3 \to 3$
4	$1 \to 1$
5	$2 \to 2$
6	$4 \to 4$
7	$5 \to 6$
8	$3 \to 4$

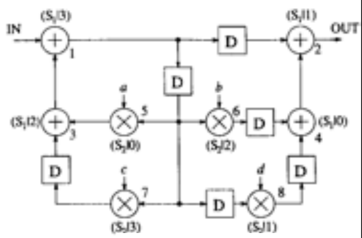
Step 5: Register allocation

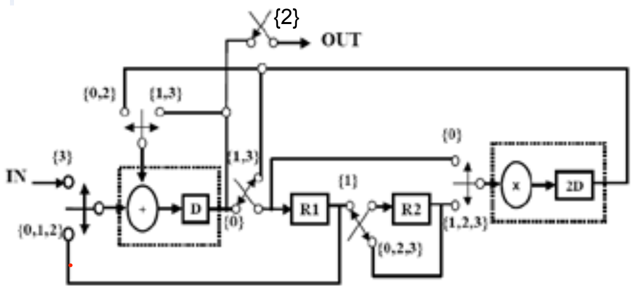


cycle	i/p	R1	R2	o/p
0				
1				
2				
3	n8			
4	n1	(n8)		n8
5	n7	n1		
6		(n7)	n1	n7
7			n1	
8			n1	
9			(n1)	n1

Step 6: Draw folded architecture







cycle	i/p	R1	R2	o/p
0				
1				
2				
3	n8			
4	n1	(n8)		n8
5	n7	n1		
6		(n7)	n1	n7
7			n1	
8			n1	
9			(n1)	n1

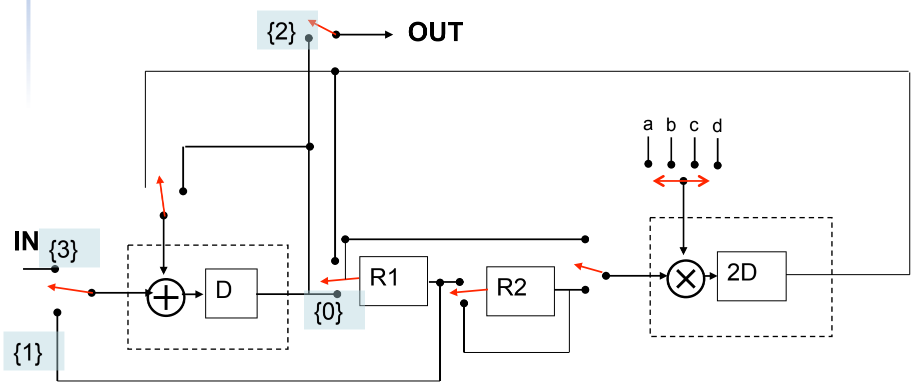
Synthesize

- To synthesize the architecture, which is shown in the previous slide, let's build partial architecture by considering each edge based on the folding equations.
- The final architecture is the combination of these partial architectures.

$$
\begin{aligned}
D_F(1 \to 2) &= 4(1) - 1 + 1 - 3 = 1 \\
D_F(1 \to 5) &= 4(1) - 1 + 0 - 3 = 0 \\
D_F(1 \to 6) &= 4(1) - 1 + 2 - 3 = 2 \\
D_F(1 \to 7) &= 4(1) - 1 + 3 - 3 = 3 \\
D_F(1 \to 8) &= 4(2) - 1 + 1 - 3 = 5 \\
D_F(3 \to 1) &= 4(0) - 1 + 3 - 2 = 0 \\
D_F(4 \to 2) &= 4(0) - 1 + 1 - 0 = 0 \\
D_F(5 \to 3) &= 4(0) - 2 + 2 - 0 = 0 \\
D_F(6 \to 4) &= 4(1) - 2 + 0 - 2 = 0 \\
D_F(7 \to 3) &= 4(1) - 2 + 2 - 3 = 1 \\
D_F(8 \to 4) &= 4(1) - 2 + 0 - 1 = 1.
\end{aligned}
$$

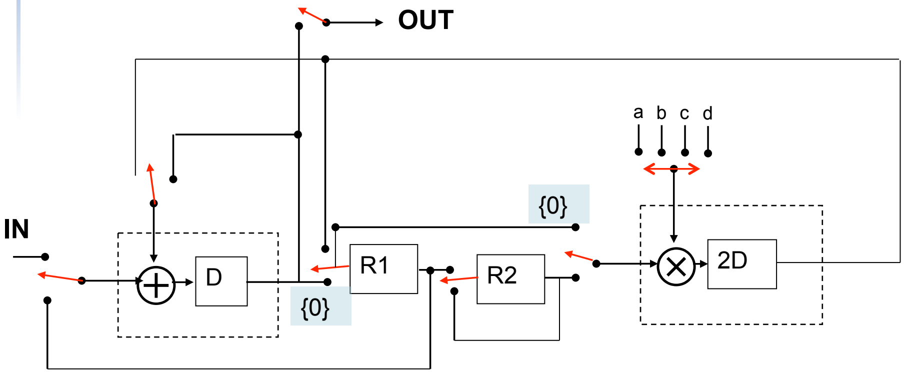
1→ 2 delay 1 (Adder to adder)

- N1 is produced at time 0 (4l+0) needed by N2 at time 1 (a delay of 1)
- An edge from R1 (where the result will be available after 1 time unit) to node 2 (adder) that switches at time 4l+1
- Also, input is switched at 4l+3 and output at 4l+2



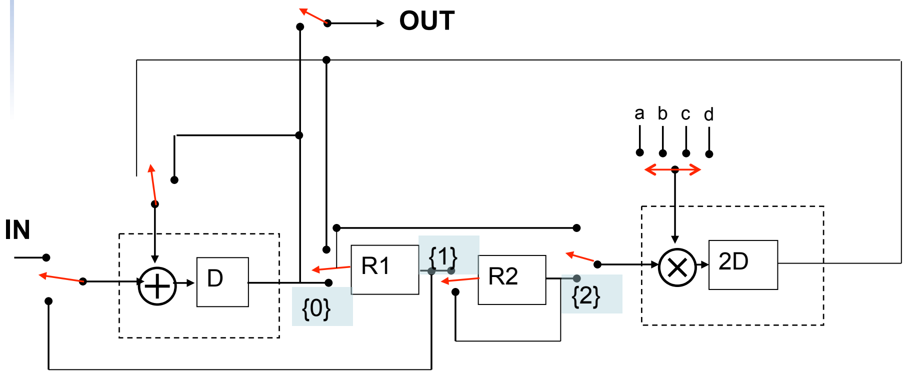
1 → 5 delay 0(adder to multiplier)

- created at 0 consumed at 0
- A path from output of adder to input of multiplier switches at 4l+0



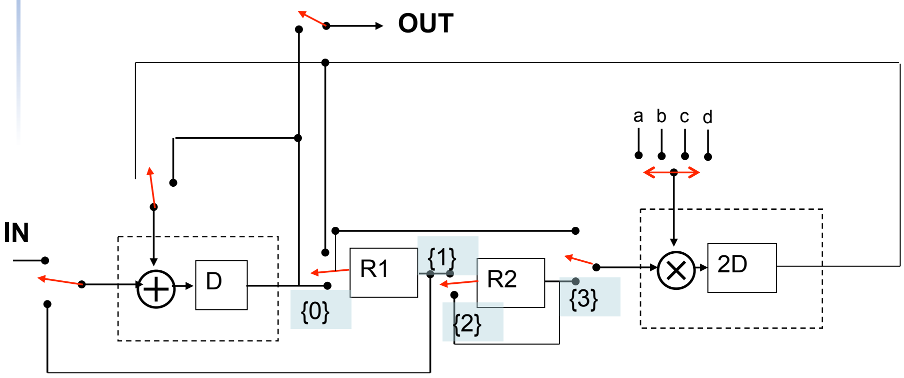
1 → 6 (adder to multiplier)

- N1 produces at time 0, needed after a delay of 2 at multiplier at 2
- Need a switch to move it from adder to R1 at 0, R1 to R2 at 1, R2 to multiplier at 2



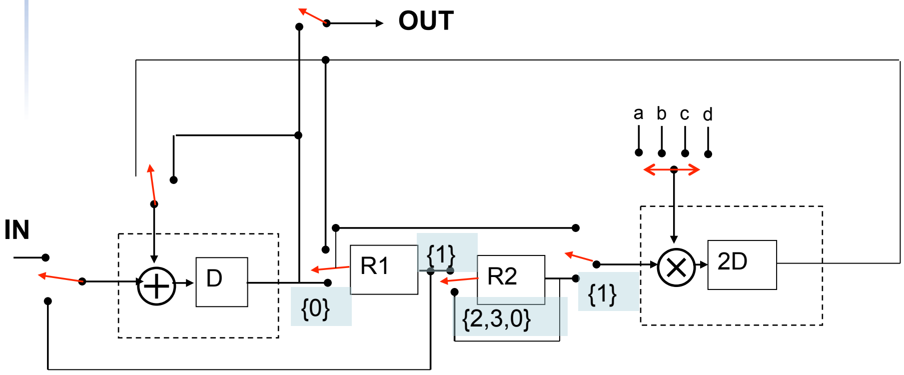
1 → 7 delay of 3 (A to M)

- N1 produced at 0, needed after 3 at 3
- We need a switch to go from adder to R1 at 0, R1 → R2 at 1, R2 → R2 at 2, R2 → multiplier at 3



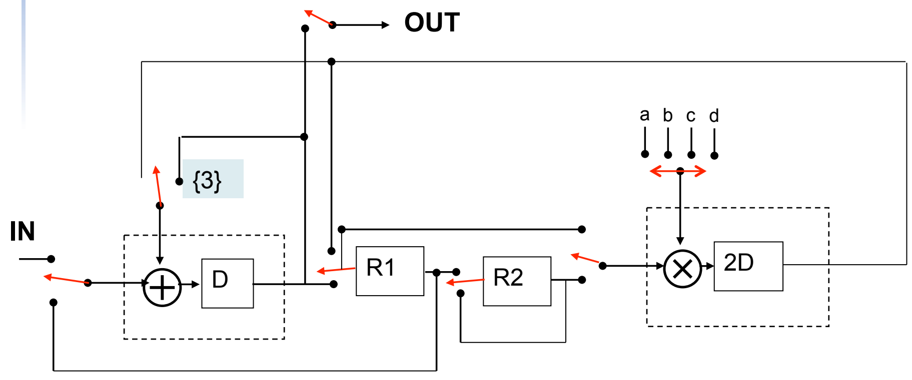
1 → 8 delay of 5 (A to M)

- N1 produces result at 0, needed after 5 at multiplier 4l+1
- Need a switch to go from Adder to R1 at 0, R1 →R2 at 1, R2 →R2 at 2, R2 → R2 at 3. R2 →R2 at 4 (4l+0), R2 → Multiplier at 5 (4l+1)



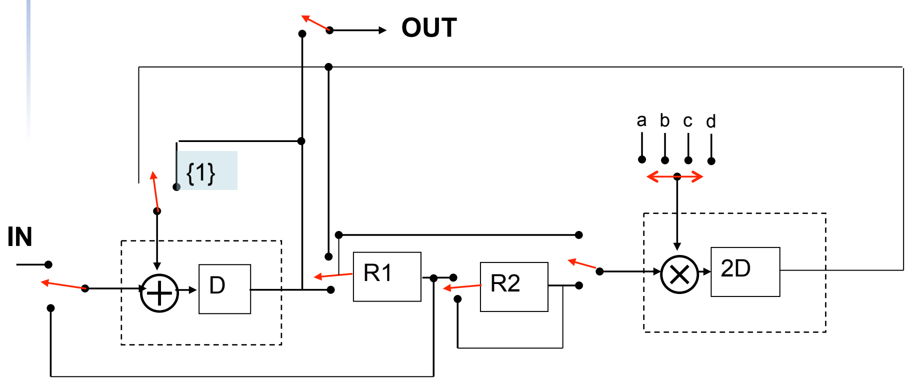
3 → 1 delay of 0 (A to A)

- N3 (adder) produces result at 3, needed at 3 at adder



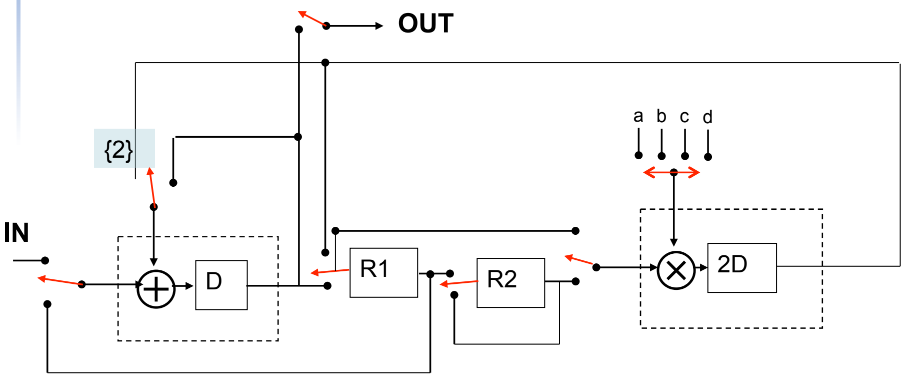
4 → 2 delay of 0 (A to A)

- N4 produces result at 1, needed at adder at 1 no delay
- Switch from output of adder to input of adder with no delay at 1



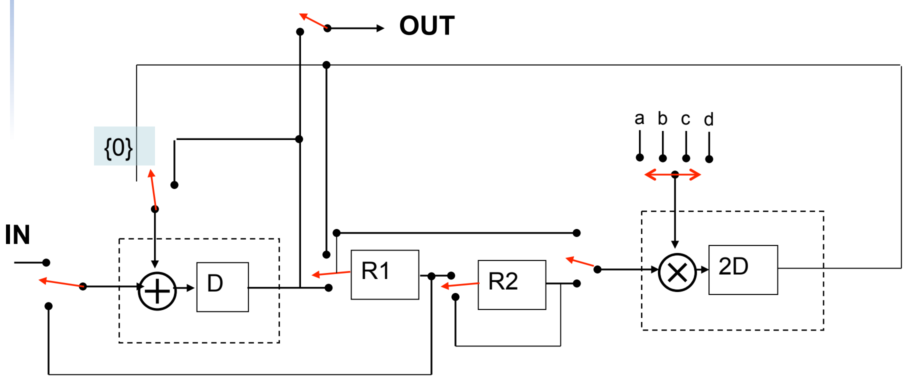
5 → 3 delay of 0 (M to A)

- N5 produces at 2, needed at adder with no delay
- A switch from output of multiplier to input of adder at 2



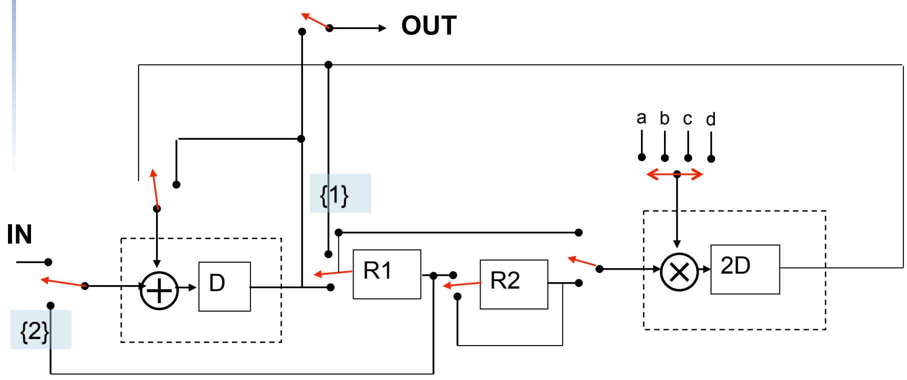
6 → 4 delay of 0 (M to A)

- N6 produces at 4 i.e. 4l+0, needed with no delay at adder
- A switch from output of multiplier to input of adder at 0



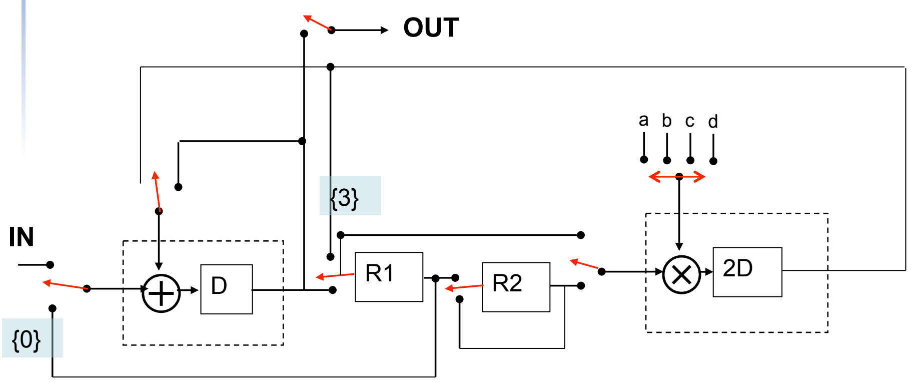
7 → 3 delay of 1 (M to A)

- N7 produces at 5, i.e. 4l+1, needed after 1 delay at adder
- A switch from multiplier to R1 at 1, R1 → adder at 2



8 → 4 delay of 1 (M to A)

- N8 produces at 3 needed after one delay to input of adder
- A switch from Multiplier → R1 at 3 and R1 → adder at 0



Final structure

- Superimposing the switches produces the final architecture
- Note that constants a,b,c,d are muxed into multiplier at {0,2,3,1}