Chapter 6
Folding

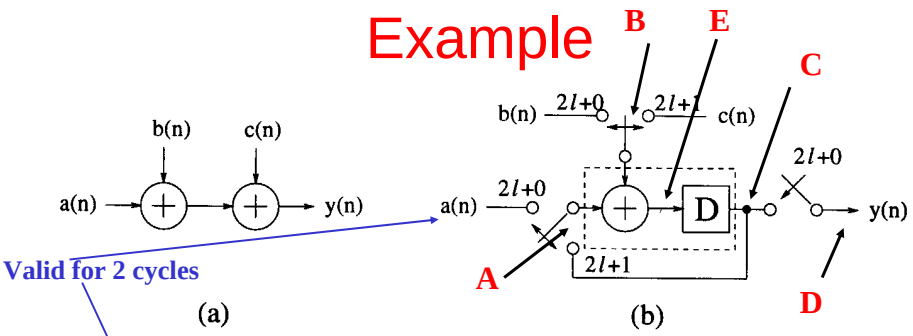
CSE4210 Winter 2012

Mokhtar Aboelaze

Folding

- The folding transformation is used to systematically determine the control circuits in DSP architecture where multiple algorithm operations are time-multiplexed to a single functional unit.
- The hardware is reduced by a factor of N, the time is increased by the same factor.
- May lead to a large number of registers, thus registers minimization techniques are studied.

Example



Valid for 2 cycles

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)

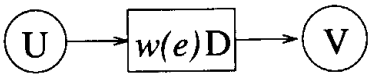
Folding Transformation

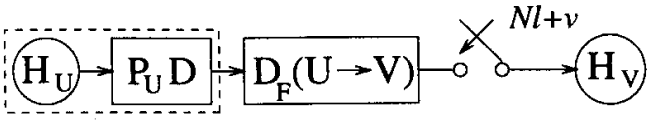
- The objective is to provide a systematic technique for designing control circuits for hardware where several algorithm operations are mapped to the same piece of hardware via time-multiplexing of course.
- We start with a DFG for the algorithm.
- We need the following definitions

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) U →w(e) V
- Folding factor is N
- Node (computation) U l^{th} iteration is performed at time Nl +u
- Node (computation) V l^{th} iteration is performed at time Nl +v
- H_u and H_v are the hardware units U and V are performed at
- H_u and H_v are pipelined by P_u and P_v stages

Folding Transformation





Folding Transformation

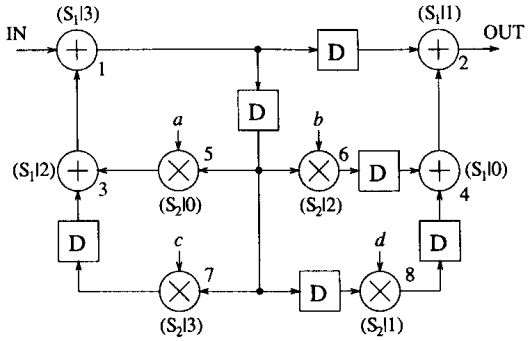
- The results of the l^{th} iteration of node U is available at $Nl+u+P_u$
- Since there are w(e) delays between U and V, the result is needed in the $(l+w(e))^{th}$ iteration of V, which is executed at $N(l+w(e))+v$

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

Folding Transformation

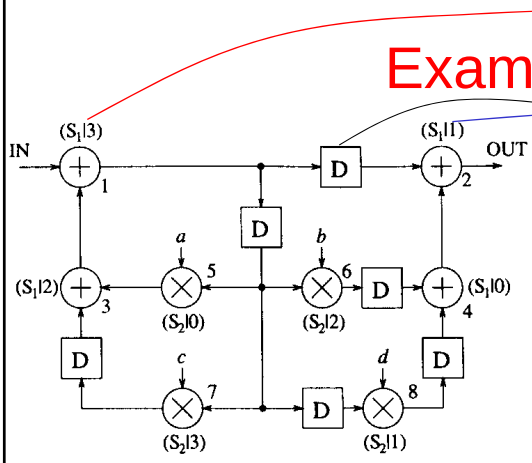
- Folding Set
  - Is an ordered set of operations executed by the same functional unit.
  - Each folding set contains N entries (some of which may be null operations)
  - The J^{th} position within the folding set is executed in the time partition j
  - For example the folding set $S_1 = \{A_1, \phi, A_2\}$ 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 $S_1|2$
  - Folding set is obtained using a scheduling and allocation algorithm

Example



- N=4
- Folding sets are adder S_1={4,2,3,1} and a multiplier S_2={5,8,6,7}
- Addition takes 1 and multiplication 2 time units
- 1-stage adder and 2-stage multiplier

Example



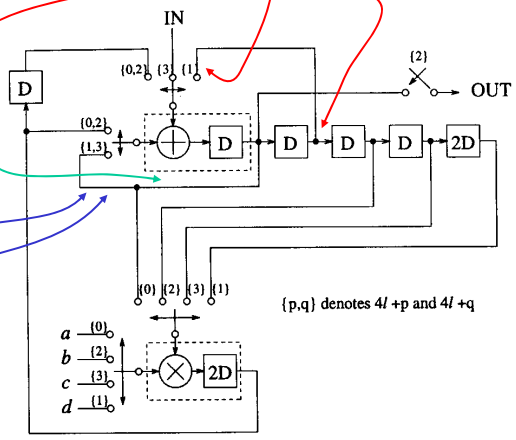
$$D_F(U \xrightarrow{e} V) = Nw(e) - P_u + v - u$$

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

Example

Adder Adder Arrives at After delay of

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



Folding Transformation

- What if some of the D_F's are negative
- Of course we can not implement that
- A condition: $D_F \geq 0$
- We can use retiming of the original graph to get a valid D_F's
- Recall, retiming equation U → V

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

Folding Transformation

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

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

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

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

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

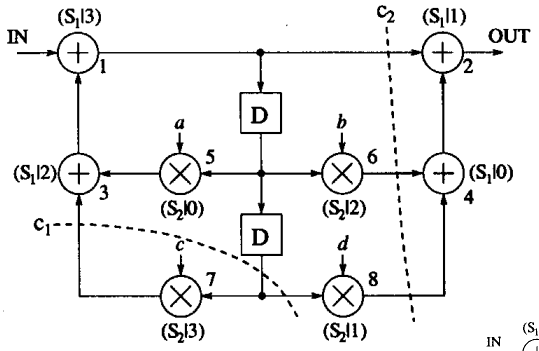
$$r(U) - r(V) \leq \frac{D_F(U \xrightarrow{e} V)}{N}$$

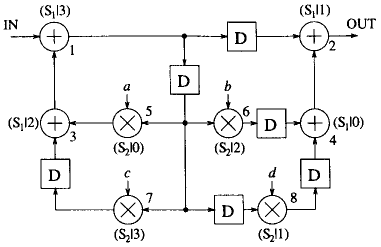
$$r(U) - r(V) \leq \left\lfloor \frac{D_F(U \xrightarrow{e} V)}{N} \right\rfloor$$

Folding Transformation

- We can use the techniques in Chapter 4 to solve for retiming.
- Then we fold the graph → valid transformation

Exercise



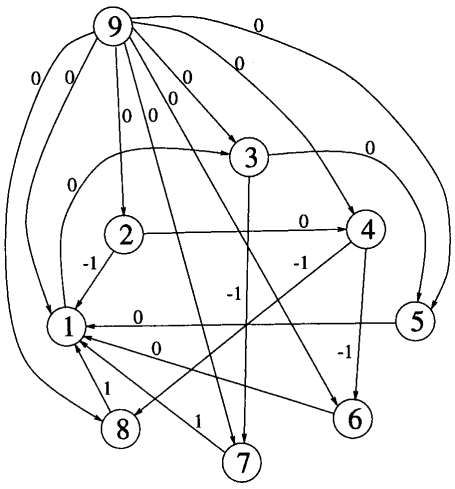


$$r(U) - r(V) \le \left\lfloor \frac{D_F(U \xrightarrow{e} V)}{N} \right\rfloor$$

Exercise

Edge	Folding Equation	Retiming for Folding Constraint
$1 \to 2$	$D_F(1 \to 2) = -3$	$r(1) - r(2) \le -1$
$1 \to 5$	$D_F(1 \to 5) = 0$	$r(1) - r(5) \le 0$
$1 \to 6$	$D_F(1 \to 6) = 2$	$r(1) - r(6) \le 0$
$1 \to 7$	$D_F(1 \to 7) = 7$	$r(1) - r(7) \le 1$
$1 \to 8$	$D_F(1 \to 8) = 5$	$r(1) - r(8) \le 1$
$3 \to 1$	$D_F(3 \to 1) = 0$	$r(3) - r(1) \le 0$
$4 \to 2$	$D_F(4 \to 2) = 0$	$r(4) - r(2) \le 0$
$5 \to 3$	$D_F(5 \to 3) = 0$	$r(5) - r(3) \le 0$
$6 \to 4$	$D_F(6 \to 4) = -4$	$r(6) - r(4) \le -1$
$7 \to 3$	$D_F(7 \to 3) = -3$	$r(7) - r(3) \le -1$
$8 \to 4$	$D_F(8 \to 4) = -3$	$r(8) - r(4) \le -1$

Exercise



Solution

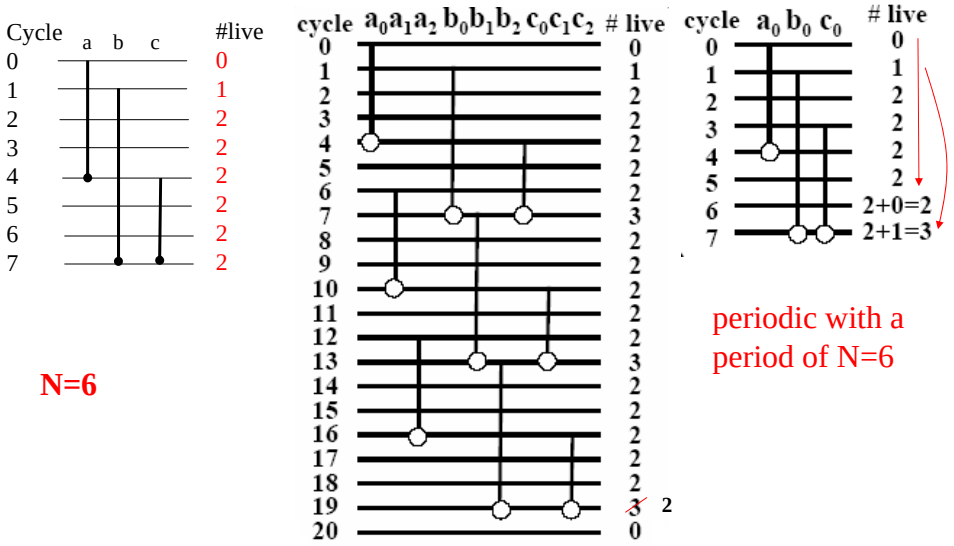
- One solution is
- r(1)=-1 r(2)=0 r(3)=0 r(4)=0
- r(5)=0 r(6)=-1 r(7)=-1 r(8)=-1
- Can we reach the above solution from the method we studies in this course? **NO**
- Another solution
- r(1)=-1 r(2)=0 r(3)=-1 r(4)=0
- r(5)=-1 r(6)=-1 r(7)=-2 r(8)=-2

Registers Minimization Techniques

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

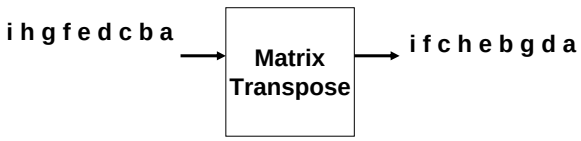
Life Time Analysis

- 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.



Example

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



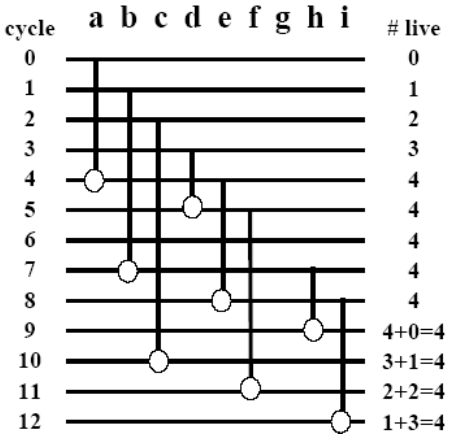
Example

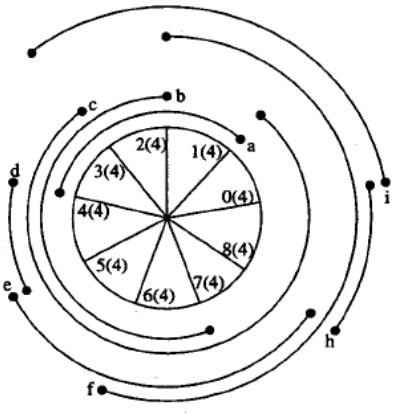
Output time with zero delay

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

+4

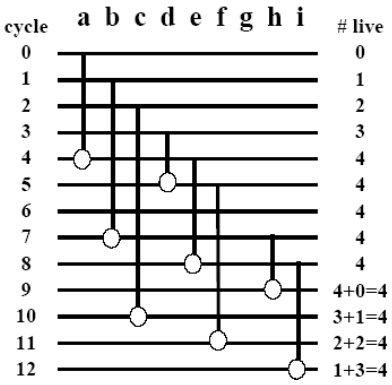
Circular Life-Time Chart





Data Allocation

- Determine the min. number of registers
- Input each variable at the time its life starts. If more than one use multiple registers such that the longest lifetime is allocated to the initial register.
- Each variable is allocated in a forward manner until it is dead or reaches the last register
- Allocation is periodic, all allocation to current iteration repeats itself after after N
- If reaches the last register and not dead allocate backward ((if more than one, choose one that has been allocated backward before), then forward again and so on.



Allocation table

Cycle	I/P	R1	R2	R3	R4	O/P
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**

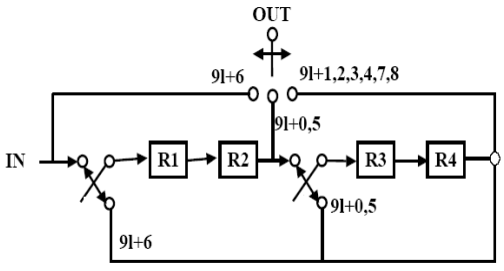
$N \geq 6$

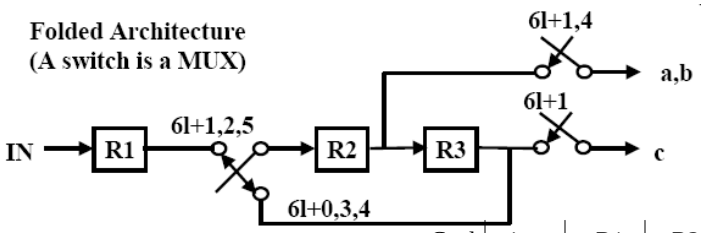
Cycles	a	b	c	#live
0				0
1				1
2				2
3				2
4				2
5				2
6				2+0
7				2+1=3

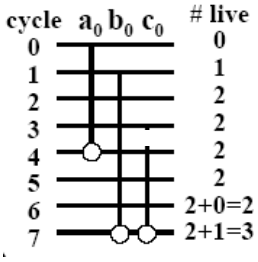
Cycle	input	R1	R2	R3	Output
0	a				
1	b	a			
2		b	a		
3			b	a	
4	c		**a**	b	**a**
5		c	b		
6			c	b	
7			**b**	c	b,c

Synthesis

Cycle	I/P	R1	R2	R3	R4	O/P
0	a					
1	b	a				
2	c	b	a			
3	d	c	b	a		
4	e	e	e	e	**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**







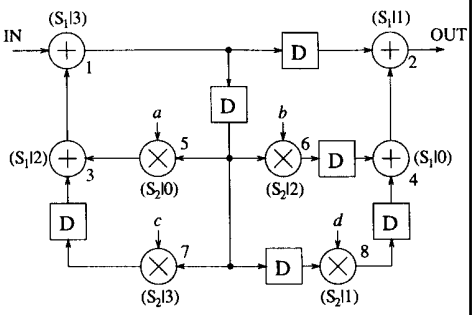
Cycle	input	R1	R2	R3	Output
0	a				
1	b	a			
2		b	a		
3			b	a	
4	c		**a**	b	**a**
5		c	b		
6			c	b	
7			**b**	**c**	b,c

Register Minimization for Folded Architecture

- Perform retiming for folding
- Write the folding equations
- Use the folding equations to construct a lifetime table
- Draw the lifetime chart and determine the minimum number of registers
- Perform forward-backward register allocation
- Draw the folded architecture

Example

- Consider the Biquad example.
- Node u is created at time $u+P_u$
- Node u is consumed at time = $u+P_u$ + $\max_V(D_F(U \rightarrow V)$
- We have already solved this example and we got the folding equations



Biquad Filter

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

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

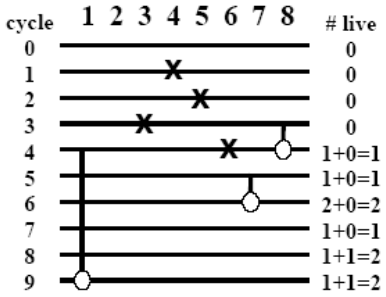
Node U produces data at time=$u+P_u$

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	$3 \rightarrow 4$

T_{out} for node U

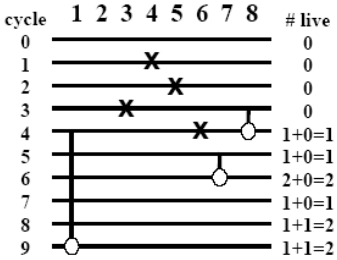
$$u + P_u + \max_v \{D_F(U \rightarrow V)\}$$

Biquad Filter



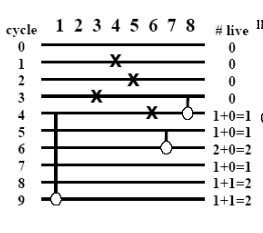
Node	$T_{input} \rightarrow T_{output}$
1	$4 \rightarrow 9$
2	-----
3	$3 \rightarrow 3$
4	$1 \rightarrow 1$
5	$2 \rightarrow 2$
6	$4 \rightarrow 4$
7	$5 \rightarrow 6$
8	$3 \rightarrow 4$

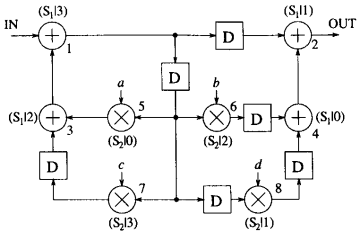
Biquad Filter



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

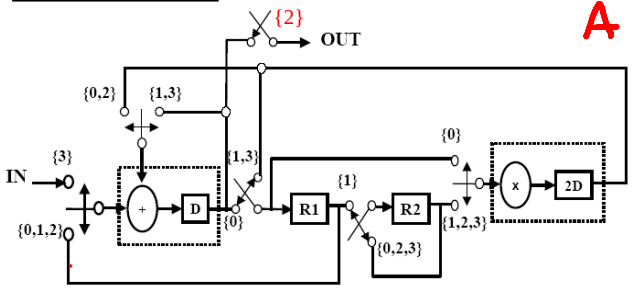
cycle	i/p	R1	R2	o/p
0				
1				
2				
3	n8			
4	n1	(n8)		n8
5	n7	n1		
6		(n7)	n1	n7
7			n1	
8			n1	
9			(n1)	n1





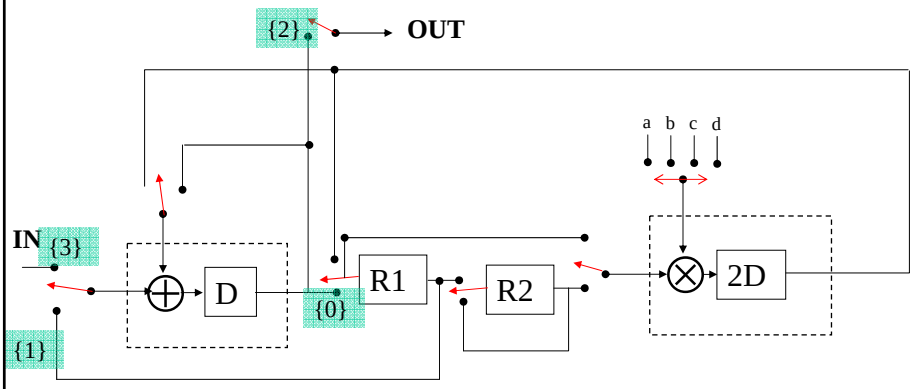
$D_f(1,6) = 2.$

A R R M
4 5 6
0 1 2



Explanation

- Now, we build the architecture (final architecture is in the previous slide) by considering every edge in the graph.
- Combining these partial architecture to make the final one.

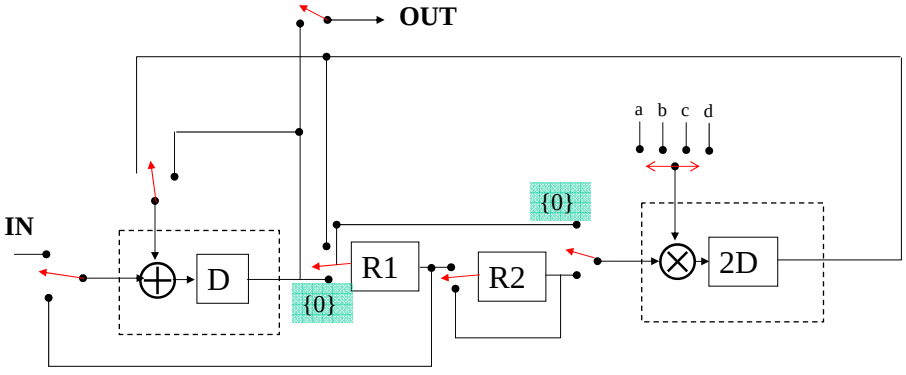


Considering 1→ 2 (Adder to adder)

N1 is produced at time 0 (4ℓ+0) needed by 2 at time 1 (a delay of 1)

An edge from R1 (where the result will be after 1 time unit) to node 2 (adder) that switches at time 4ℓ+1

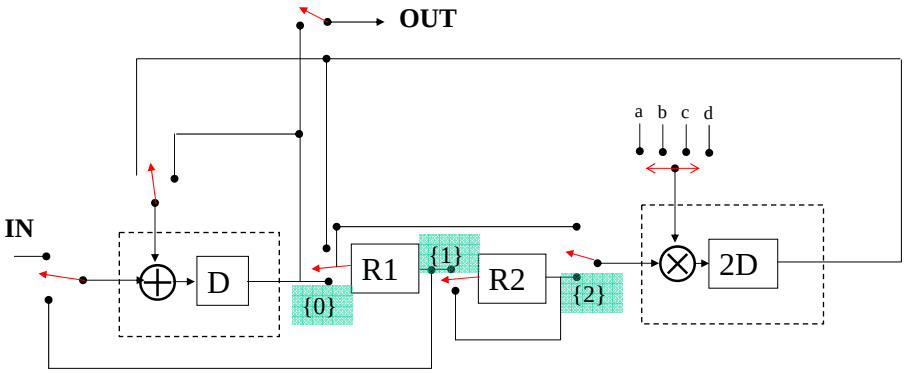
Also, input is switched at 4l+3 and output at 4l+2



1 → 5 (adder to multiplier)

That require 0 delay (D_F(1→5)) created at 0 consumed at 0

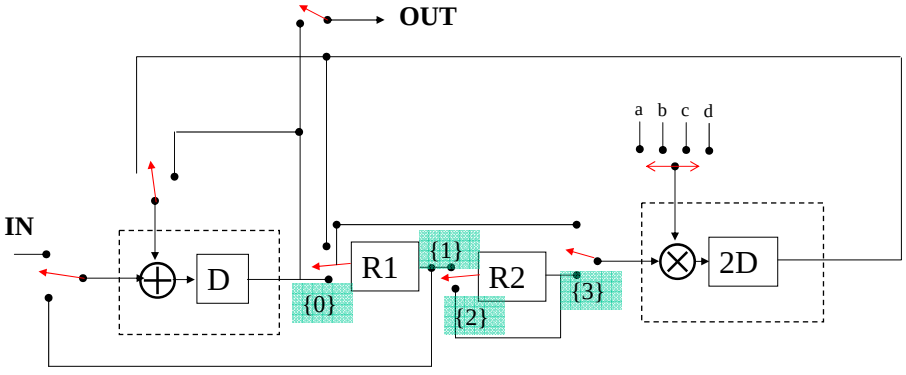
A path from output of adder to input of multiplier switches at 4ℓ+0



1 → 6 adder to multiplier

N1 produces at time 0, needed after a delay of 2 at multiplier at 2

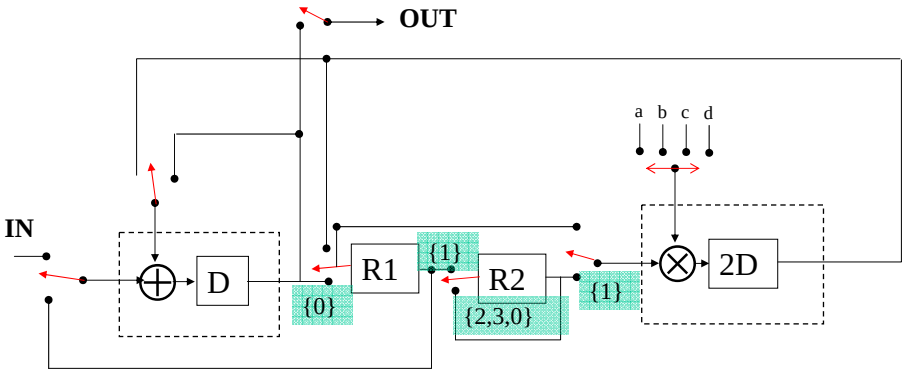
Need a switch to move it from adder to R1 at 0, R1 to R2 at 1, R2 to multiplier at 2



1 → 7 delay of 3

N1 produced at0, needed after 3 at 3

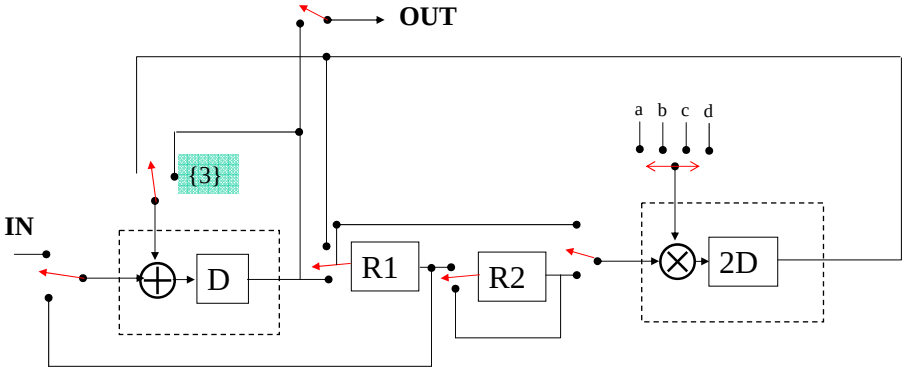
We need a switch to go from adder to R1 at 0, R1 → R2 at 1, R2 → R2 at 2, R2 → multiplier at 3



1 → 8 delay of 5

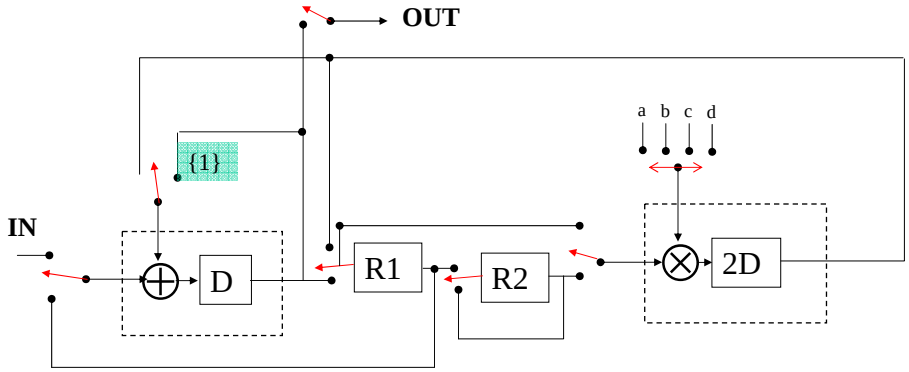
N1 produces result at 0, needed after 5 at multiplier $4\ell+1$

A switch Adder to R1 at 0, R1 →R2 at 1, R2 →R2 at 2, R2 → R2 at 3. R2 →R2 at 4 ($4\ell+0$), R2 → Multiplier at 5 ($4\ell+1$)



3 → 1 delay of 0

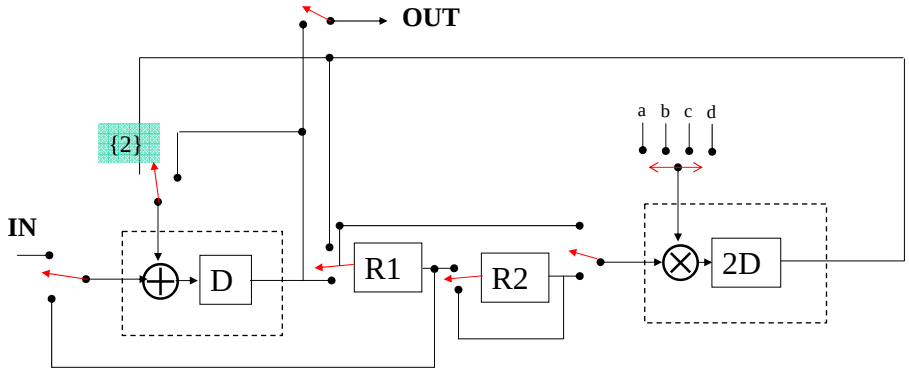
N3 produces result at 3, needed at 3 at adder



4 → 2 delay of 0

N4 produces result at 1, needed at adder at 1 no delay

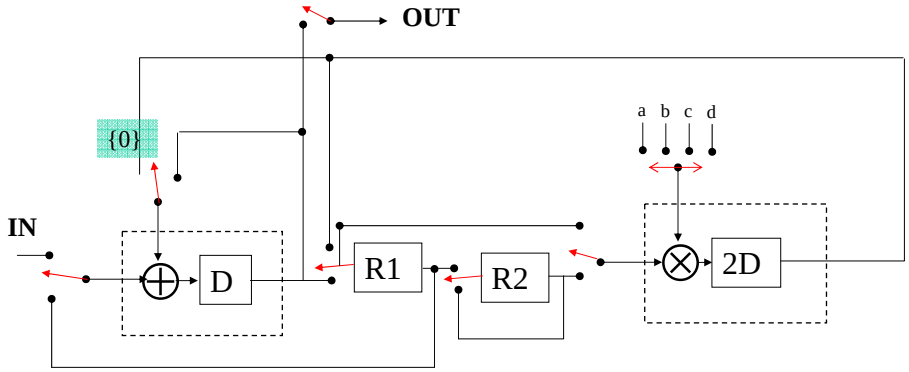
Switch from output of adder to input of adder with no delay at 1



5 → 3 delay of 0

N5 produces at 2, needed at adder with no delay

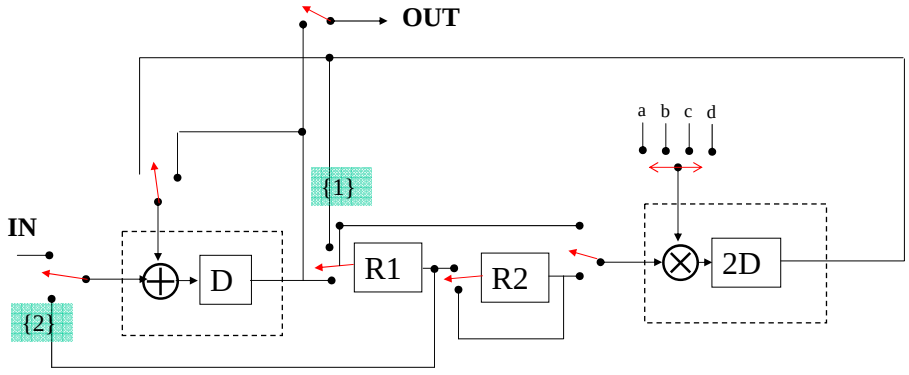
A switch from output of multiplier to input of adder at 2



6 → 4 delay of 0

N6 produces at 4 i.e. 0 (2=2) needed with no delay at adder

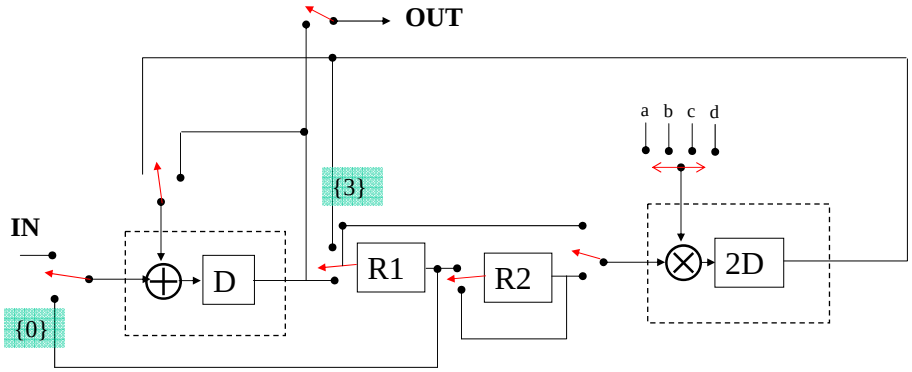
A switch from output of multiplier to input of adder at 0



7 → 3 delay of 1

N7 produces at (3+2)=1, needed after 1 delay at adder

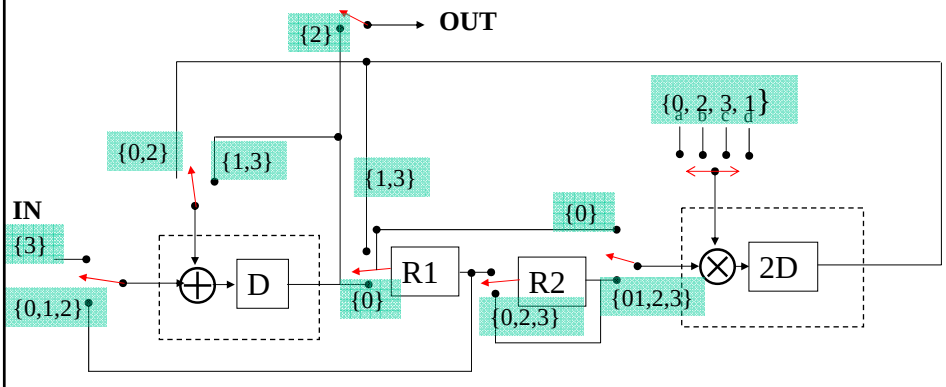
A switch from multiplier to R1 at 1, R1 → adder at 2



8 → 4 delay of 1

N8 produces at (1+2)=3 needed after one delay to input of adder

A switch from Multiplier → R1 at 3 and R1 → adder at 0



Superimposing the switches produces the final architecture

Note that constants a,b,c,d are muxed into multiplier at {0,2,3,1}