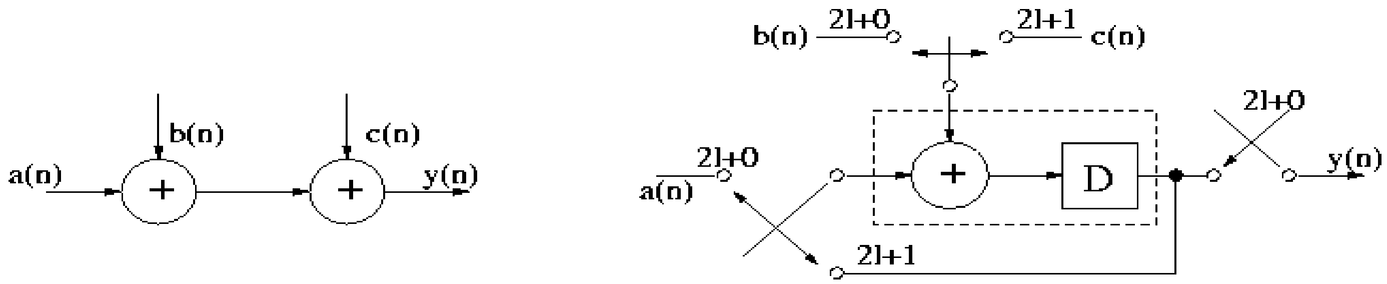
Folding

Vassilis Paliouras, Integrated System Design , ECE Dept, U. Patras

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)



(a)

(b)

- Fig(a) shows a DSP program : $y(n) = a(n) + b(n) + c(n)$.
- Fig(b) shows a folded architecture where 2 additions are folded or time-multiplexed to a single pipelined adder
- One output sample is produced every 2 clock cycles ⇒ input should be valid for 2 clock cycles.

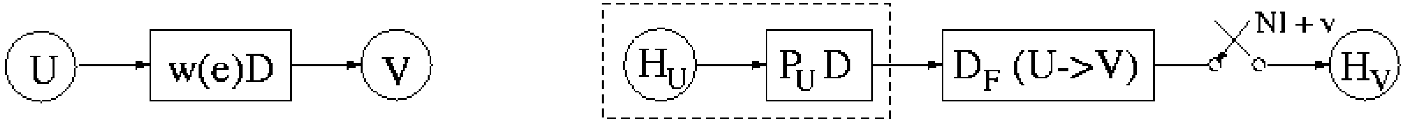
Folding Assumptions

- Assume the data on the input of a folded realization, valid for N cycles before changing, where N is the number of algorithm operations executed on a single functional unit in hardware.

CHARACTERISTICS:

- Reduction of functional units ➔ need for a formal procedure for FOLDING
- Increase of registers ➔ need for register minimization techniques

Folding Transformation



(a) (b)

- 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 iterations 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 $0 \le u, v \le N - 1$.
- H_U and H_V are functional units that execute u and v respectively.
- H_U is pipelined by P_U stages and its output is available at $Nl + u + P_U$.
- Edge $U \rightarrow V$ has $w(e)$ delays $\Rightarrow$ the l-th iteration of U is used by $(l + w(e))$ th iteration of node V, which is executed at $N(l + w(e)) + v$.

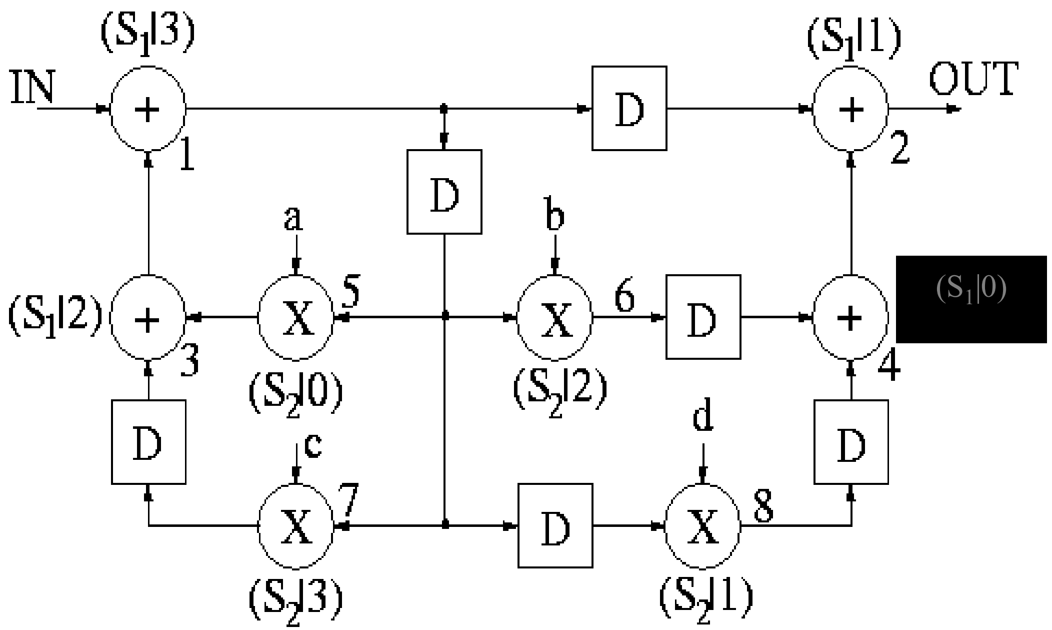
  So, the result should be stored for :

$$D_{F(U\rightarrow V)} = \left[N\left(l + w(e)\right) + v\right] - \left[Nl + P_U + u\right] \Rightarrow$$

$$D_F(U\rightarrow V) = Nw(e) - P_U + v - u$$

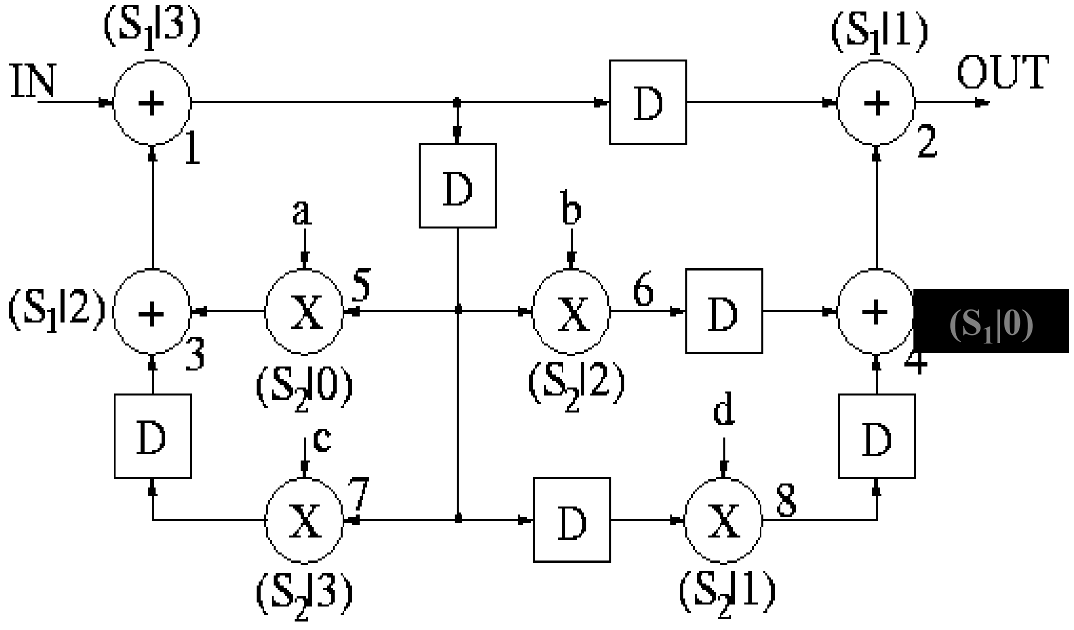
(independent of l)

- Folding Set : An ordered set of N operations executed by the same functional unit. The operations are ordered from 0 to $N-1$. Some of the operations may be null.
  For example, Folding set $S_1 = \{A_1, \emptyset, A_2\}$, for folding order $N = 3$.
- A_1 has a folding order of 0, denoted by $(S_1|0)$,
- A_2 has a folding order of 2, denoted by $(S_1|2)$, and (During time instances $3l+1$ the unit is not utilized).
- Example: Folding a retimed biquad filter by $N = 4$. (Each unit is executed every 4 u.t.)



Addition time = 1 u.t. Multiplication time = 2 u.t.
Units are clocked every 1 u.t.
1-stage pipelined adder (P_A=1) and 2-stage pipelined multiplier (P_M=2)

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

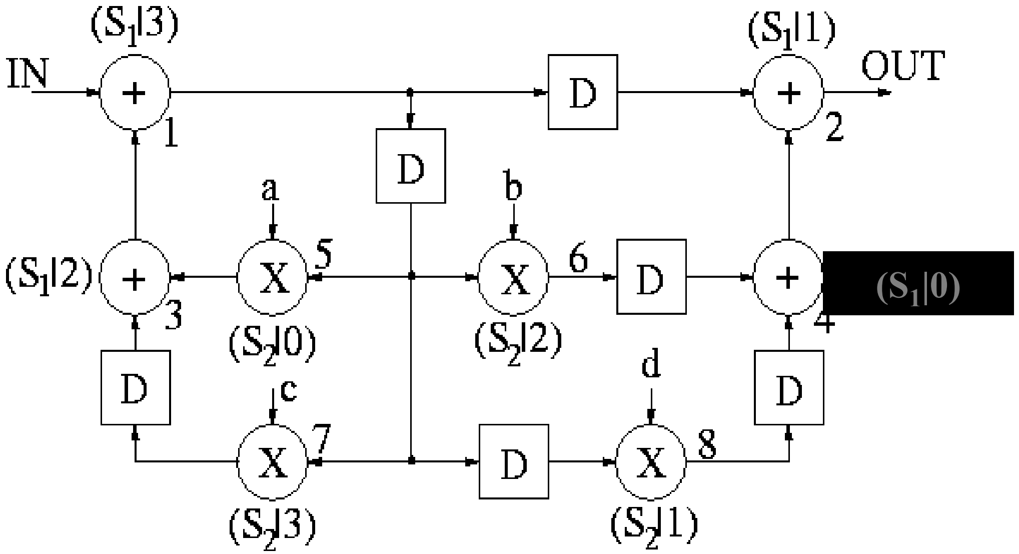


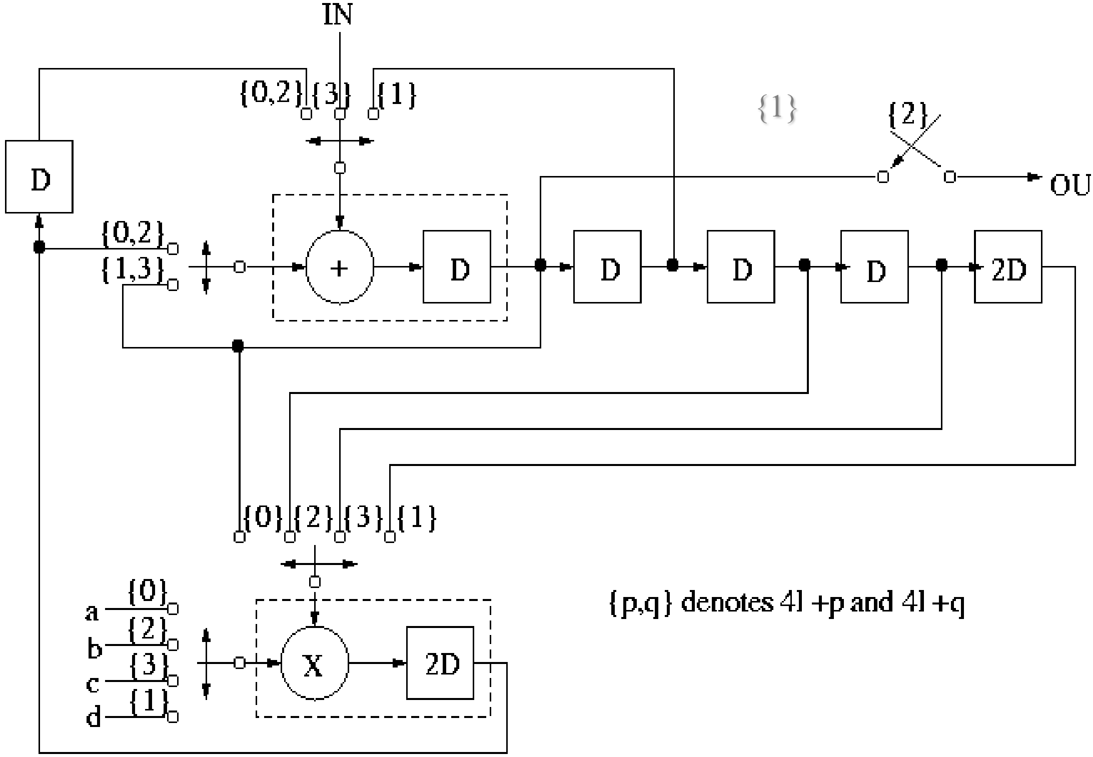
Folding equations for each of the 11 edges are as follows:

$$D_F(U \to V) \;=\; Nw(e) \;-\; P_U \;+\; v - u$$

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

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





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

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

Retiming for Folding

–For a folded system to be realizable, $D_{F(U \rightarrow V)} \geq\ 0$ for all edges.

–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 $w_r(e) = w(e) + r(V) - r(U)$

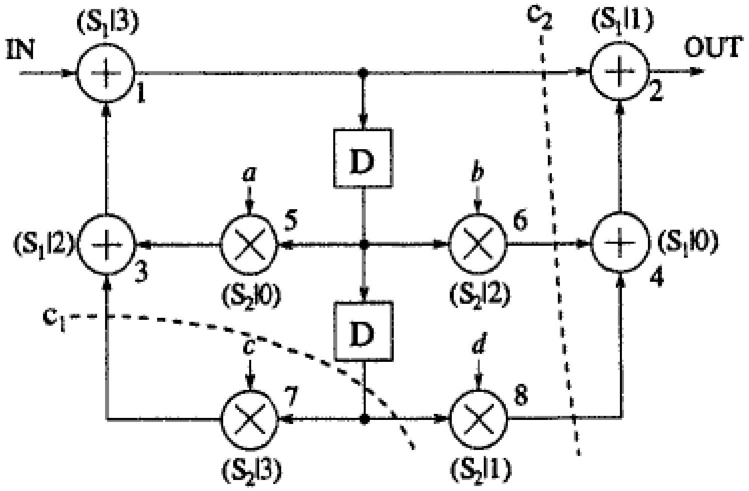
$\Rightarrow N(w(e) + r(V) - r(U)) - P_U + v - u \geq 0$

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

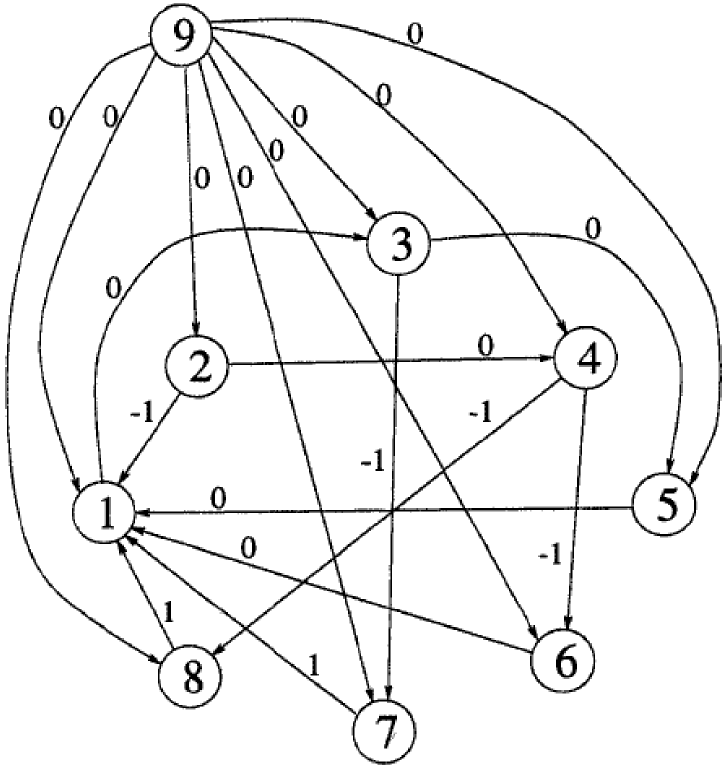
(since retiming values are integers)

Once the above set of constraints is found, optimization algorithms can be used (as for retiming techniques) to solve the system of inequalities:

1. Draw a constraint graph,
2. If it contains negative cycles, retime it, using the shortest path from node $n + 1$ to node I,
3. Derive the folding equations using a scheduling and allocation algorithm (see Appendix B)
4. Apply the folding transformation



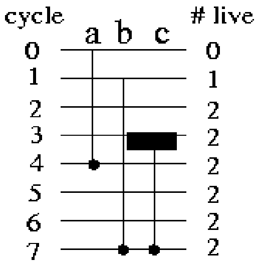
Edge	Folding Equation	Retiming for Folding Constraint
$1 \to 2$	$D_F(1 \to 2) = -3$	$r(1) - r(2) \leq -1$
$1 \to 5$	$D_F(1 \to 5) = 0$	$r(1) - r(5) \leq 0$
$1 \to 6$	$D_F(1 \to 6) = 2$	$r(1) - r(6) \leq 0$
$1 \to 7$	$D_F(1 \to 7) = 7$	$r(1) - r(7) \leq 1$
$1 \to 8$	$D_F(1 \to 8) = 5$	$r(1) - r(8) \leq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

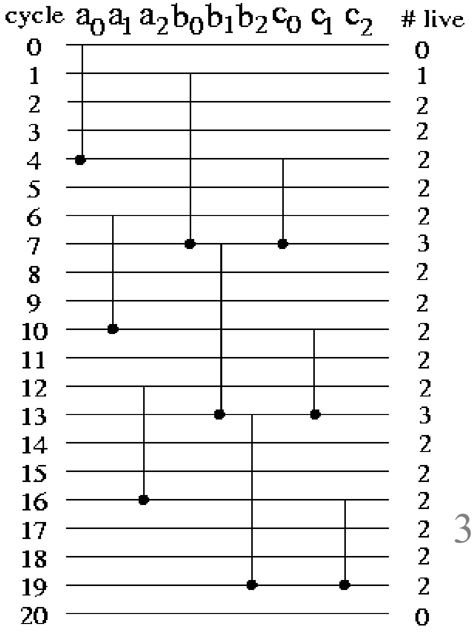


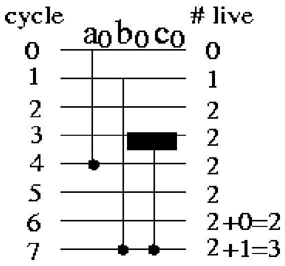
Edge	Folding Equation	Retiming for Folding Constraint
$1 \rightarrow 2$	$D_F(1 \rightarrow 2) = -3$	$r(1) - r(2) \leq -1$
$1 \rightarrow 5$	$D_F(1 \rightarrow 5) = 0$	$r(1) - r(5) \leq 0$
$1 \rightarrow 6$	$D_F(1 \rightarrow 6) = 2$	$r(1) - r(6) \leq 0$
$1 \rightarrow 7$	$D_F(1 \rightarrow 7) = 7$	$r(1)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

Register Minimization

- Register Minimization Technique : Lifetime analysis is used for register minimization techniques in a DSP 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 v
- Example :







$N = 6$

3 consecutive iterations

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

Periodic DSP programs

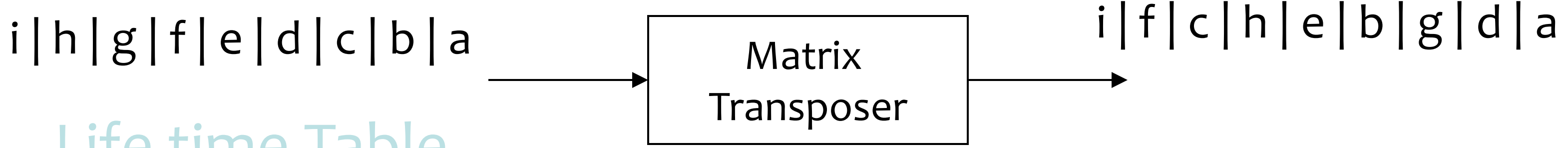
- Due to the periodic nature of DSP programs the lifetime chart can be drawn for only one iteration to give an indication of the # of registers that are needed.

- This is done as follows :
  - Let N be the iteration period
  - Let the # of alive variables at time partitions $n \geq N$ be the # of alive variables due to 0-th iteration at cycles $n - kN$ for $k \geq 0$.

In the example, the # of alive variables at cycle $7 \geq N$ $(=6)$ is the sum of the # of alive variables due to the 0-th iteration at cycles 7 and $(7 - 1\times6) = 1$, which is 2 + 1 = 3.

Matrix transpose example

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} \longrightarrow \begin{vmatrix} a & d & g \\ b & e & h \\ c & f & i \end{vmatrix}$$



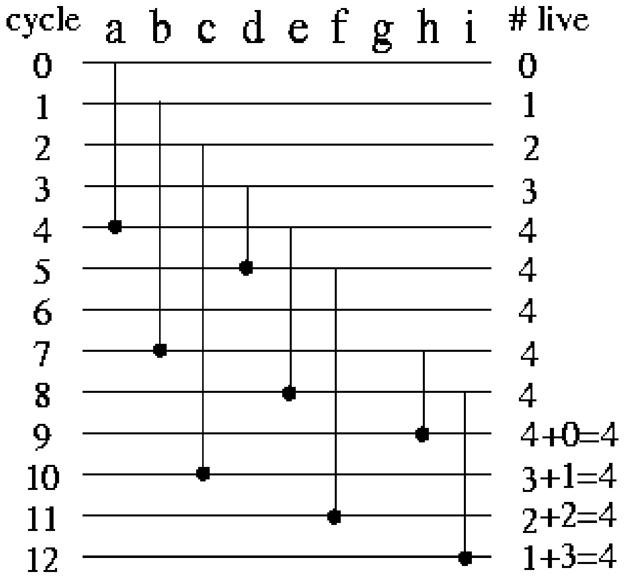
i | h | g | f | e | d | c | b | a

Matrix Transposer

i | f | c | h | e | b | g | d | a

Life time Table

Sample	T_{in}	T_{zlout}	T_{diff}	T_{out}	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



cycle	# live
0	0
1	1
2	2
3	3
4	4
5	4
6	4
7	4
8	4
9	4+0=4
10	3+1=4
11	2+2=4
12	1+3=4

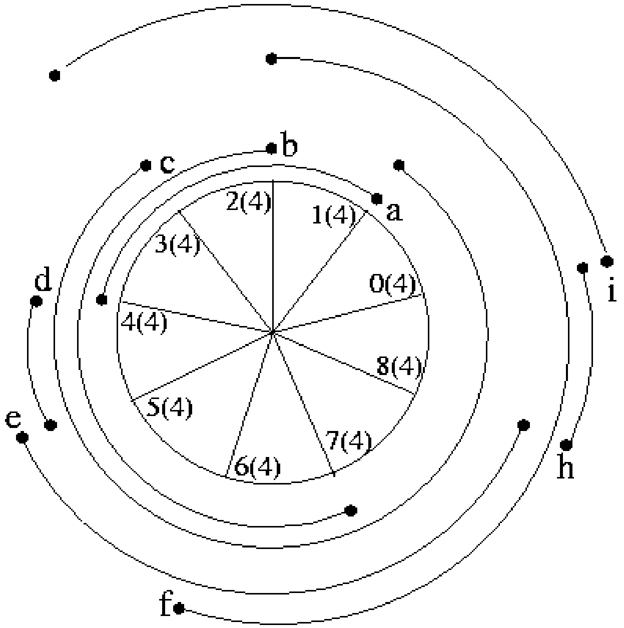
N=9

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

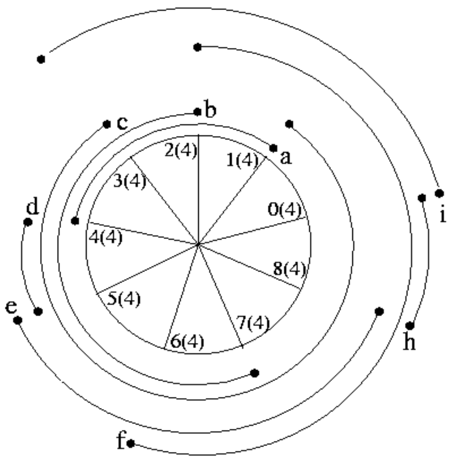
- Circular lifetime chart : Useful to represent the periodic nature of the DSP programs.
- In a circular lifetime chart of periodicity N, the point marked i $(0 \leq i \leq N - 1)$ represents the time partition i and all time instances $\{(N\,l + i)\}$, where l is any non-negative integer.
- For example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

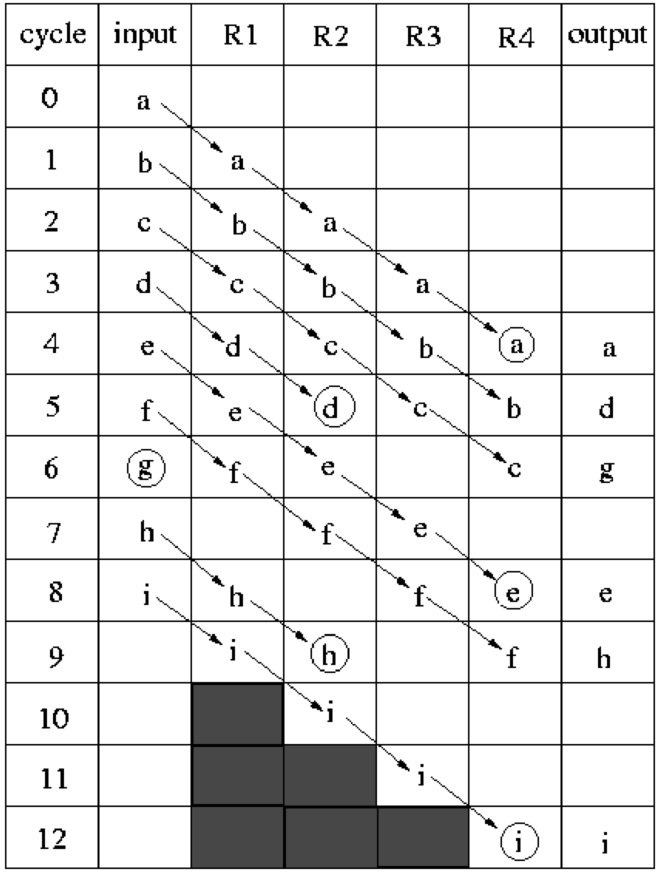
Notes :

- A 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 alive variables at each time partition.



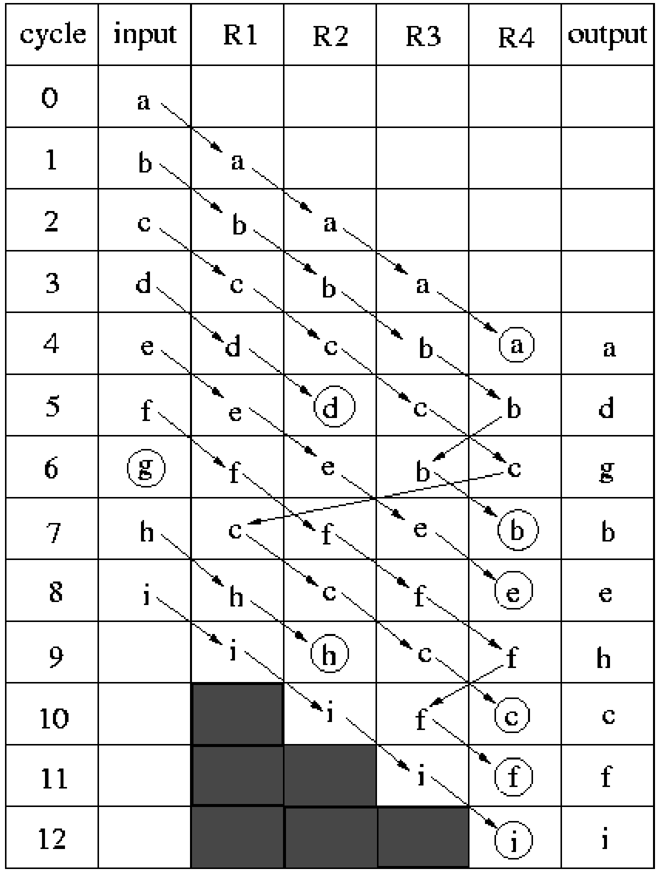
Data Allocation





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
5	f	e	ⓓ	c	b	d
6	ⓖ	f	e		c	g
7	h		f	e		
8	i	h		f	ⓔ	e
9		i	ⓗ		f	h
10		■	i			
11		■	■	i		
12		■	■	■	ⓘ	i

N=9



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
5	f	e	ⓓ	c	b	d
6	ⓖ	f	e	b	c	g
7	h	c	f	e	ⓑ	b
8	i	h	c	f	ⓔ	e
9		i	ⓗ	c	f	h
10		■	i	f	ⓒ	c
11		■	■	i	ⓕ	f
12		■	■	■	ⓘ	i

N=9

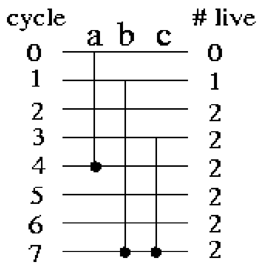
Forward-Backward Register Allocation Technique for the Matrix Transposer Example

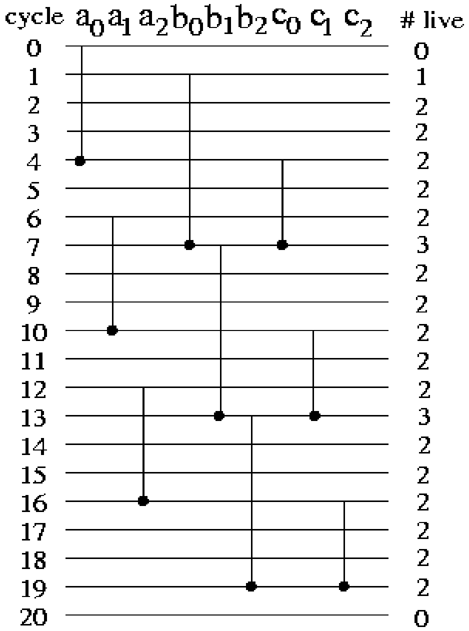
Note : Hashing is done to avoid conflict during backward allocation.

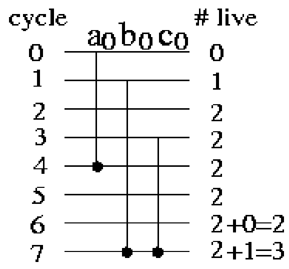
Steps for Forward-Backward Register allocation

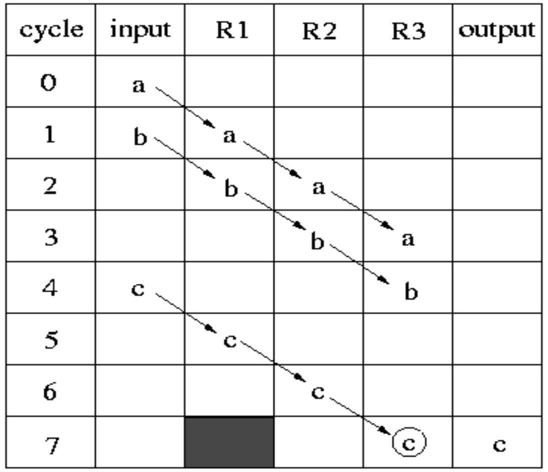
1. Determine the minimum number of registers using lifetime analysis.
2. Input each variable at the time step corresponding to the beginning of its lifetime. If multiple variables are input in a given cycle, these are allocated to multiple registers with preference given to the variable with the longest lifetime.
3. 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.
4. 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.
5. 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.
6. Repeat steps 4 and 5 until the allocation is complete.

Example: Forward-backward Register Allocation

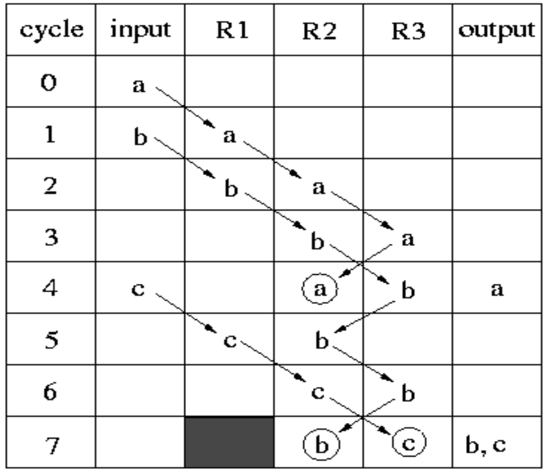




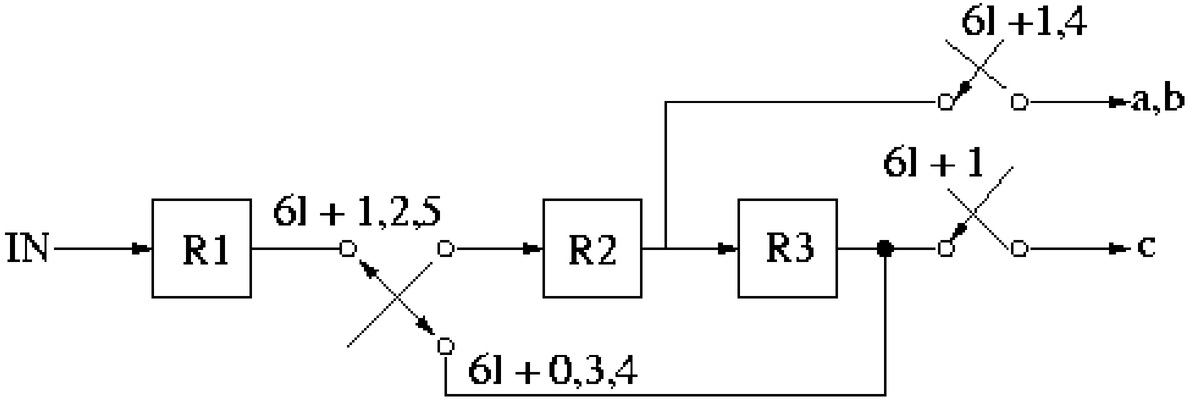




cycle	input	R1	R2	R3	output
0	a				
1	b	a			
2		b	a		
3			b	a	
4	c			b	
5		c			
6			c		
7		▨		c	c



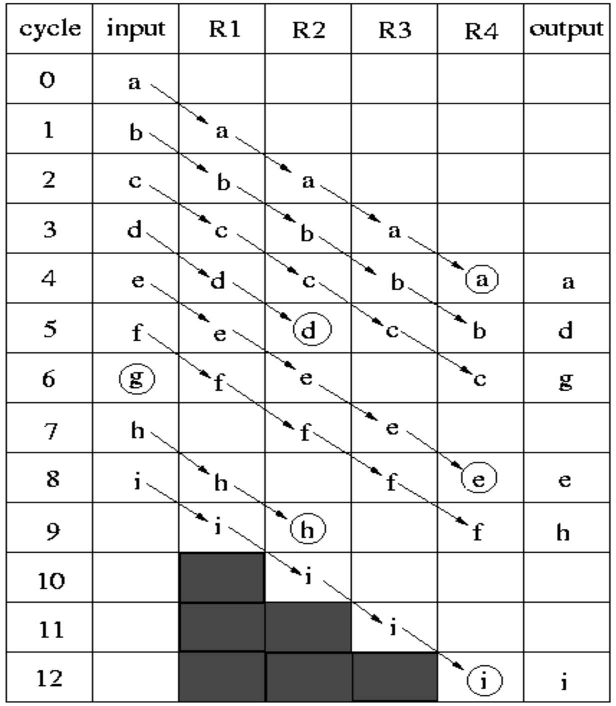
cycle	input	R1	R2	R3	output
0	a				
1	b	a			
2		b	a		
3			b	a	
4	c		a	b	a
5		c	b		
6			c	b	
7		▨	b	c	b, c



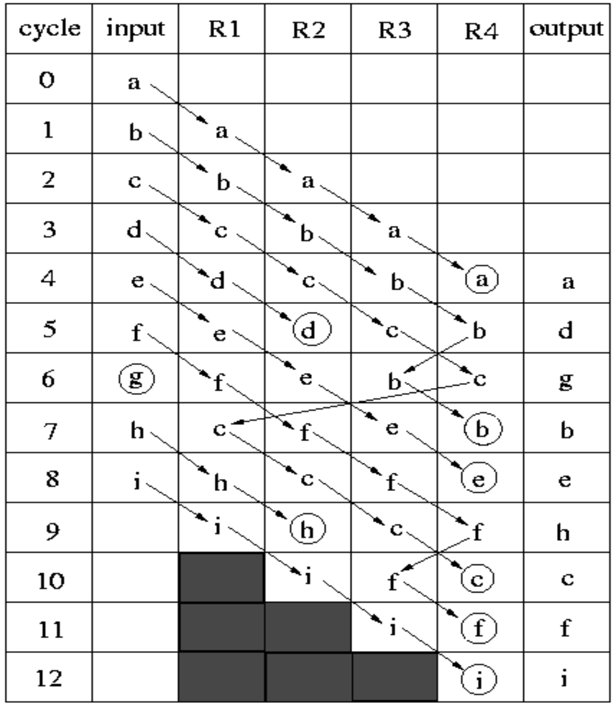
$N = 6$

3 consecutive iterations

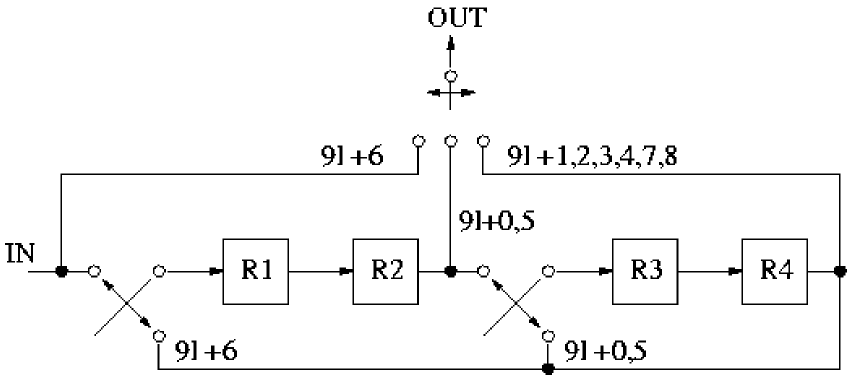
- Folded architecture for matrix tranposer :



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		c	g
7	h		f	e		
8	i	h		f	e	e
9		i	h		f	h
10		■	i			
11		■	■	i		
12		■	■	■	i	i



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

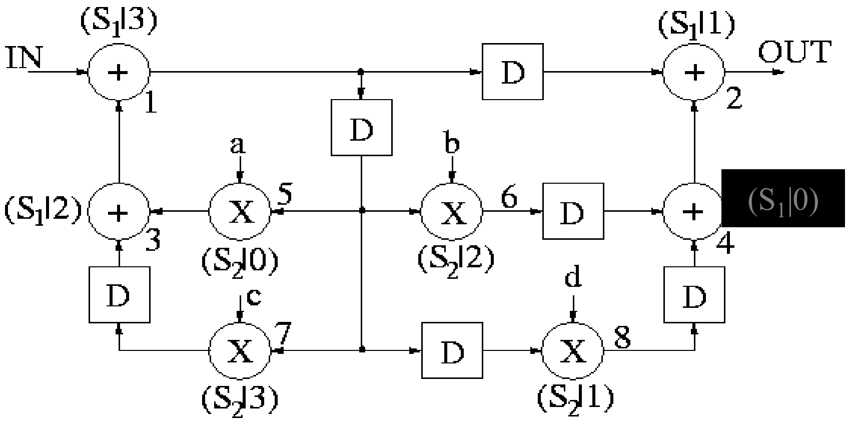


Register minimization in folded architectures

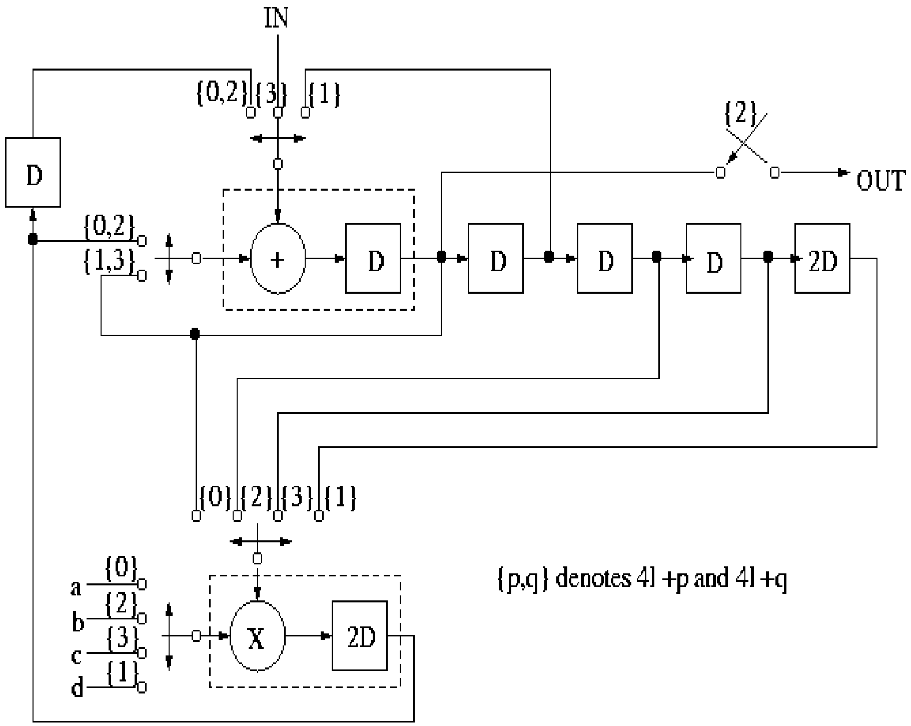
- ➢ Perform retiming for folding
- ➢ Write the folding equations
- ➢ Use the folding equations to construct a lifetime table
- ➢ Draw the lifetime chart and determine the required number of registers
- ➢ Perform forward-backward register allocation
- ➢ Draw the folded architecture that uses the minimum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

• Example : Biquad Filte

Steps 1 & 2 have already been done



The architecture with no register minimization



{p,q} denotes 4l +p and 4l +q

$D_F(1\rightarrow2) = 4(1) - 1 + 1 - 3 = 1$

$D_F(1\rightarrow5) = 4(1) - 1 + 0 - 3 = 0$

$D_F(1\rightarrow6) = 4(1) - 1 + 2 - 3 = 2$

$D_F(1\rightarrow7) = 4(1) - 1 + 3 - 3 = 3$

$D_F(1\rightarrow8) = 4(2) - 1 + 1 - 3 = 5$

$D_F(3\rightarrow1) = 4(0) - 1 + 3 - 2 = 0$

$D_F(4\rightarrow2) = 4(0) - 1 + 1 - 0 = 0$

$D_F(5\rightarrow3) = 4(0) - 2 + 2 - 0 = 0$

$D_F(6\rightarrow4) = 4(1) - 2 + 0 - 2 = 0$

$D_F(7\rightarrow3) = 4(1) - 2 + 2 - 3 = 1$

$D_F(8\rightarrow4) = 4(1) - 2 + 0 - 1 = 1$

Step 3: The lifetime table is then constructed. The 2nd row is empty as DF(2→U) is not present. Note : As retiming for folding ensures causality, we need not add any latency.

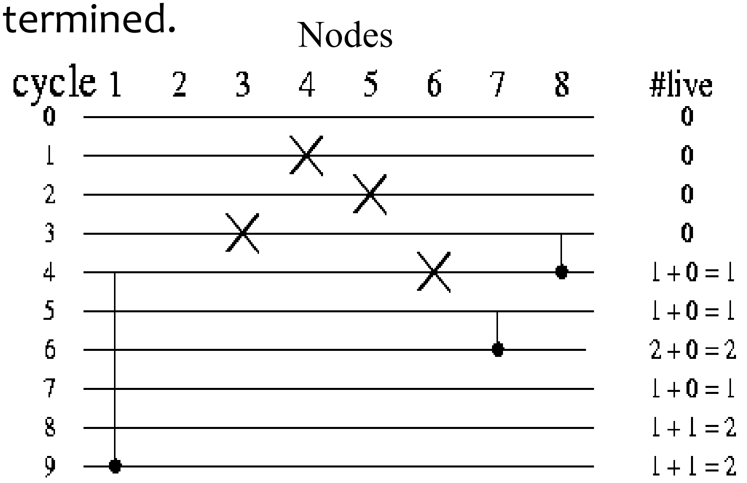
$$T_{in}(U)=u+P_u$$

$$T_{out}(U)=u+P_u+\max(v)\{D_F(U\text{->}V)\}$$

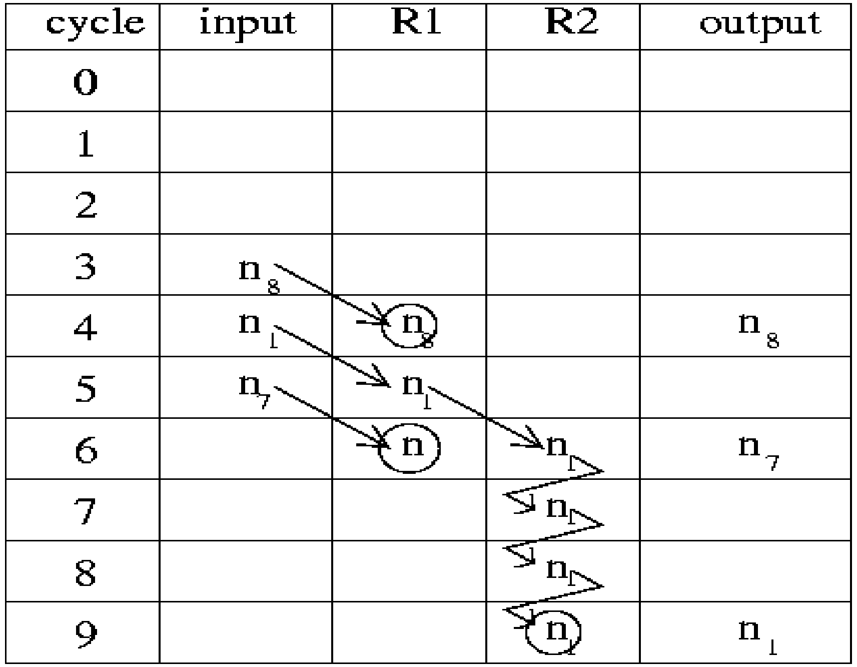
Node	$T_{in}\rightarrow T_{out}$
1	4→9
2	--
3	3→3
4	1→1
5	2→2
6	4→4
7	5→6
8	3→4

N=4 n_i is the output of node i

Step 4: Lifetime chart is constructed and registers determined.

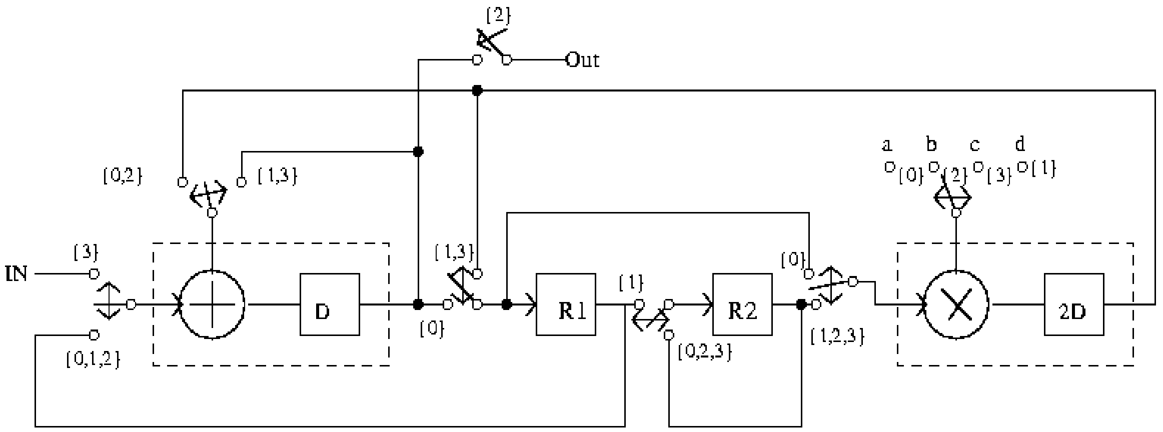


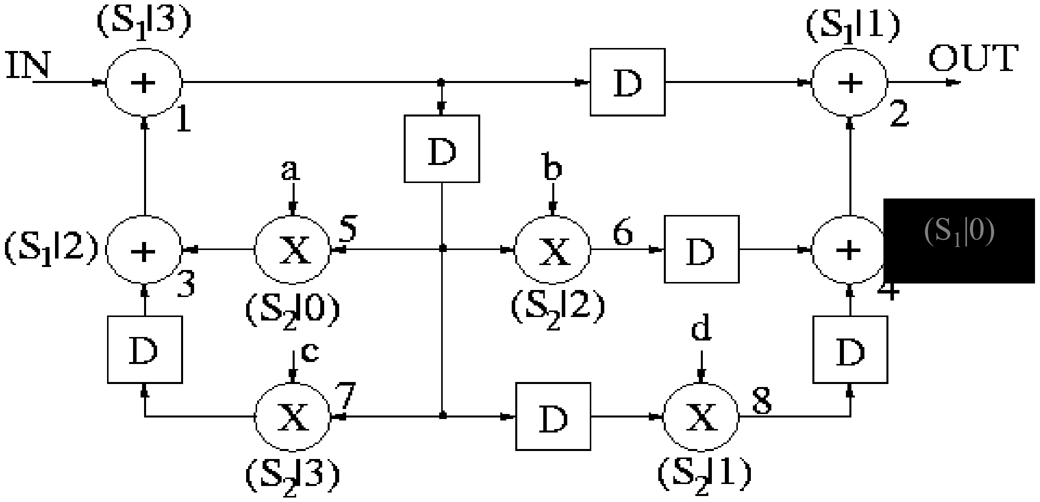
Step 5: Forward-backward register allocation



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	n_1	n_7
7			n_1	
8			n_1	
9			n_1	n_1

Folded architecture is drawn with minimum # of registers.





$D_F(1\rightarrow2) = 4(1) - 1 + 1 - 3 = 1$

$D_F(1\rightarrow6) = 4(1) - 1 + 2 - 3 = 2$

$D_F(1\rightarrow8) = 4(2) - 1 + 1 - 3 = 5$

$D_F(4\rightarrow2) = 4(0) - 1 + 1 - 0 = 0$

$D_F(6\rightarrow4) = 4(1) - 2 + 0 - 2 = 0$

$D_F(8\rightarrow4) = 4(1) - 2 + 0 - 1 = 1$

$D_F(1\rightarrow5) = 4(1) - 1 + 0 - 3 = 0$

$D_F(1\rightarrow7) = 4(1) - 1 + 3 - 3 = 3$

$D_F(3\rightarrow1) = 4(0) - 1 + 3 - 2 = 0$

$D_F(5\rightarrow3) = 4(0) - 2 + 2 - 0 = 0$

$D_F(7\rightarrow3) = 4(1) - 2 + 2 - 3 = 1$