CSE4210 Architecture and Hardware for DSP

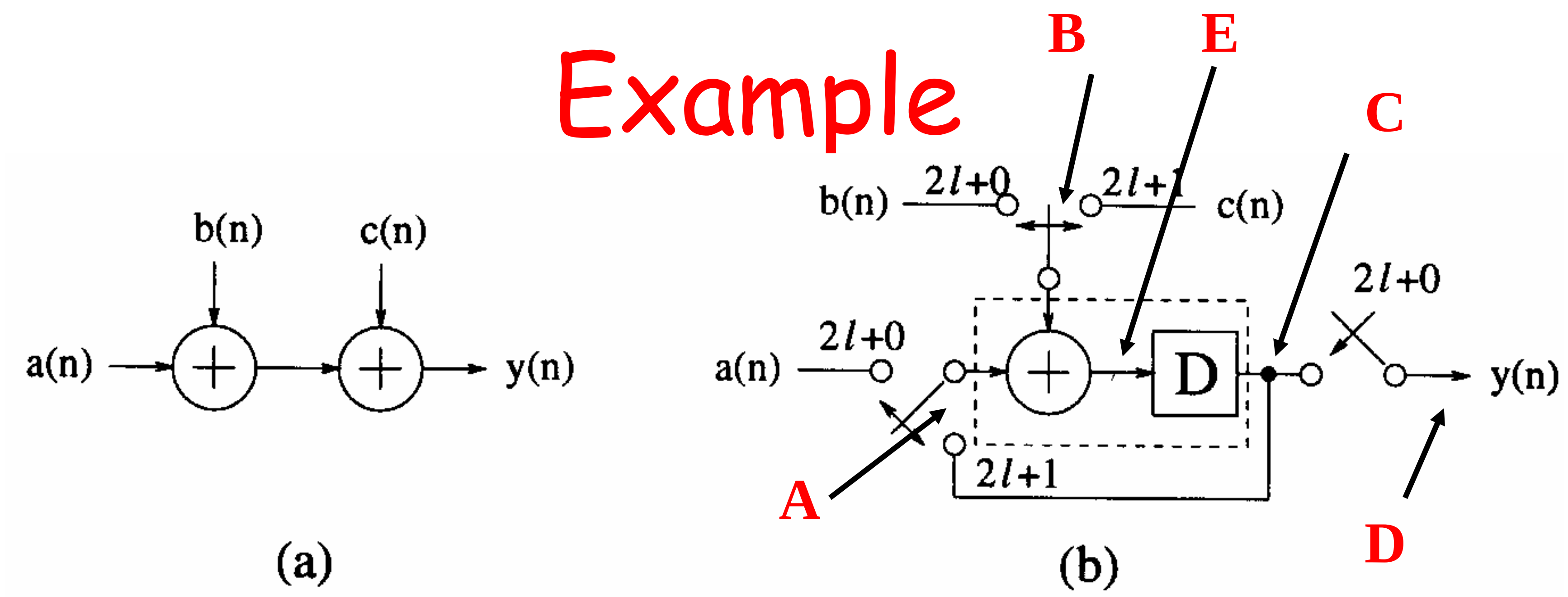
Chapter 6

Folding

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



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)

$$U \xrightarrow{w(e)} V$$

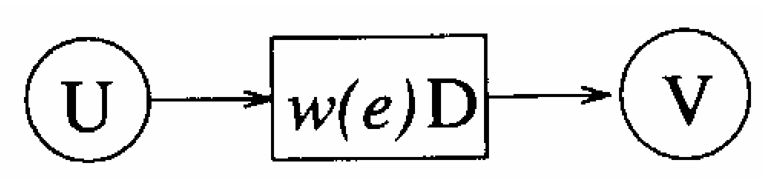
- Folding factor is N
- Node (computation) U l^{th} iteration is performed at time Nl +u
- Node (computation) V l^{th} iteration is performed at time Nl +v, u and v are the folding order of U and V <N-1 (time partition scheduled for).
- H_u and H_v are the hardware units U and V are performed at
- H_u and H_v are pipelined by P_u and P_v stages

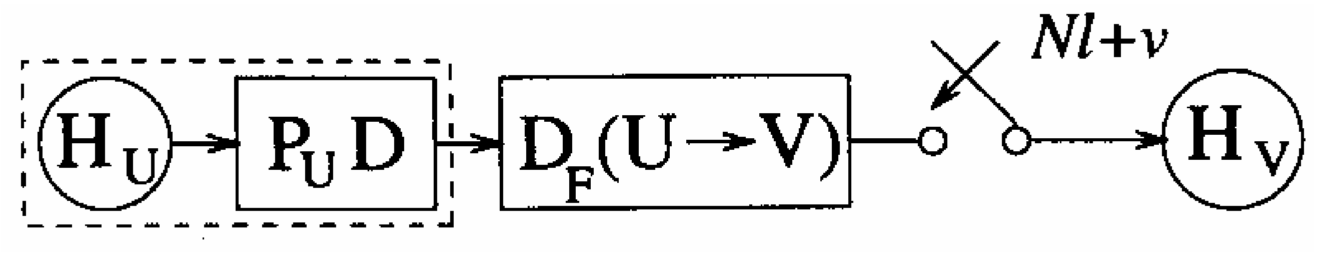
Folding Transformation

- 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, we need to store it for

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

Folding Transformation

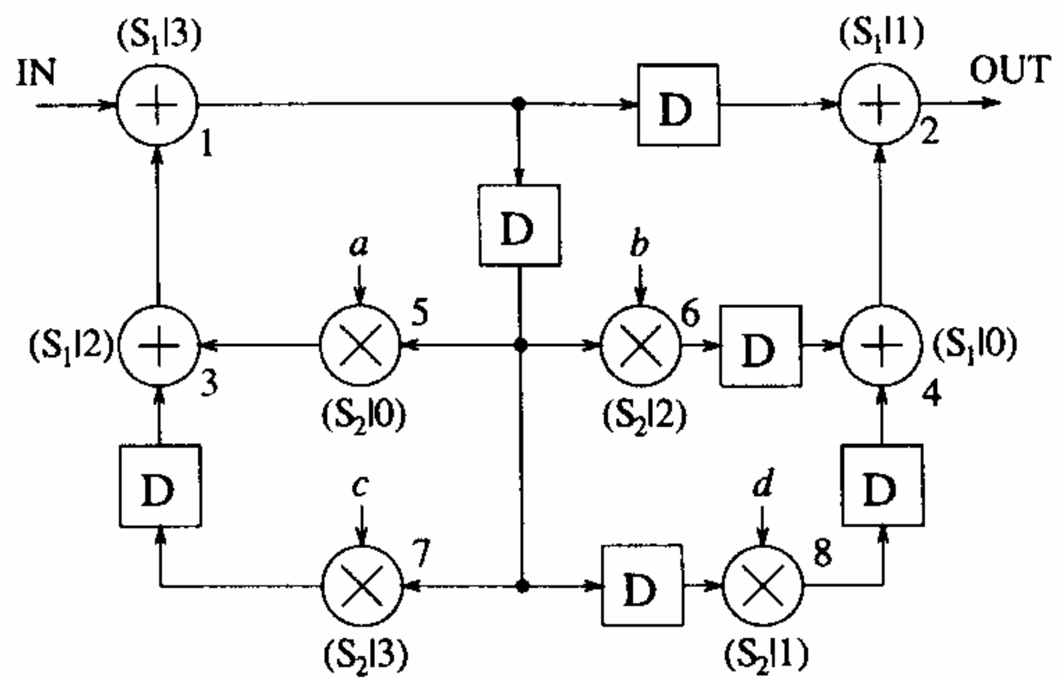




Folding Transformation

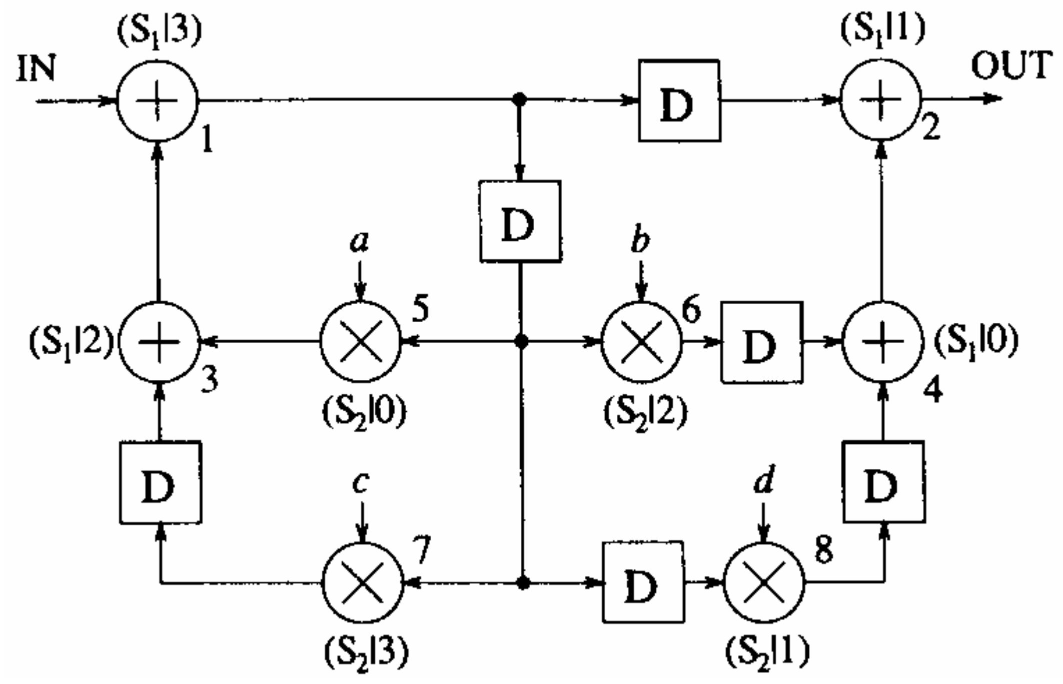
- Folding Set
  - Is an ordered set of operations executed by the same functional unit.
  - Each folding set contains N entries (some of which may be null operations)
  - The Jth position within the folding set is executed in the time partition j
  - For example the folding set $S_1 = \{A_1, \phi, A_2\}$ for N=3
  - A_1 is performed during the 0th time partition $S_1|0$, while A_2 is done in the 2nd 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

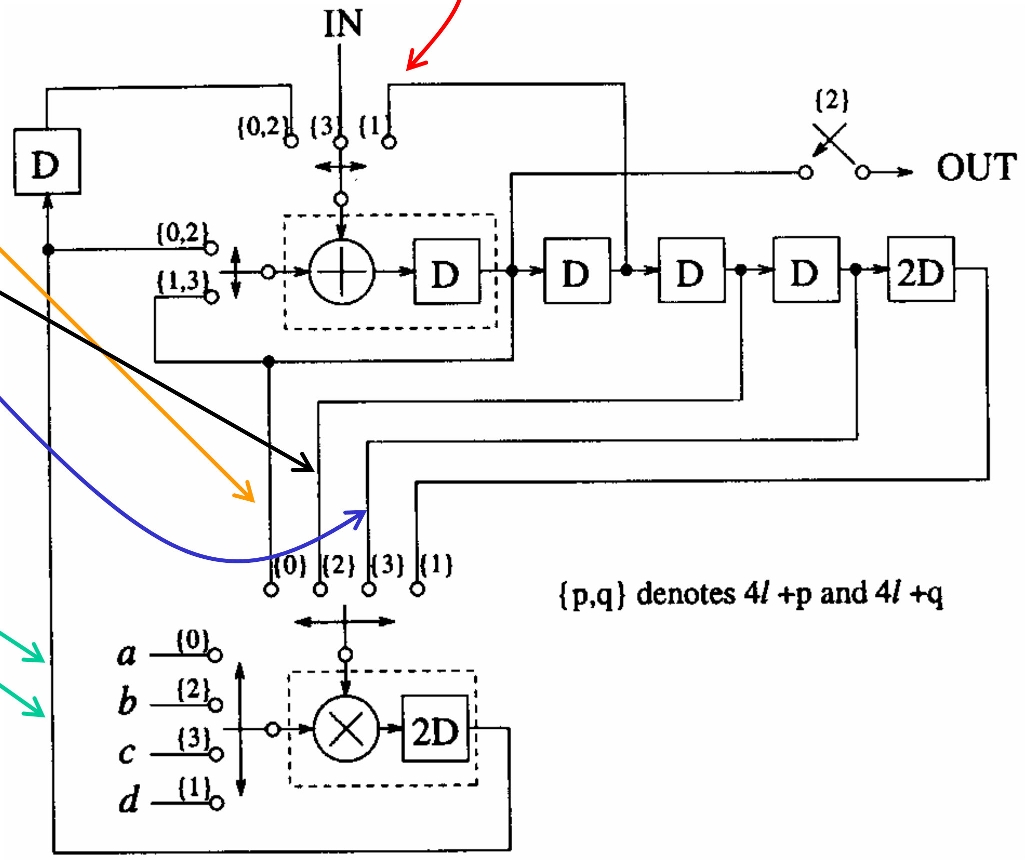
Example



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

Example

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



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

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

Folding Transformation

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

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

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

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

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

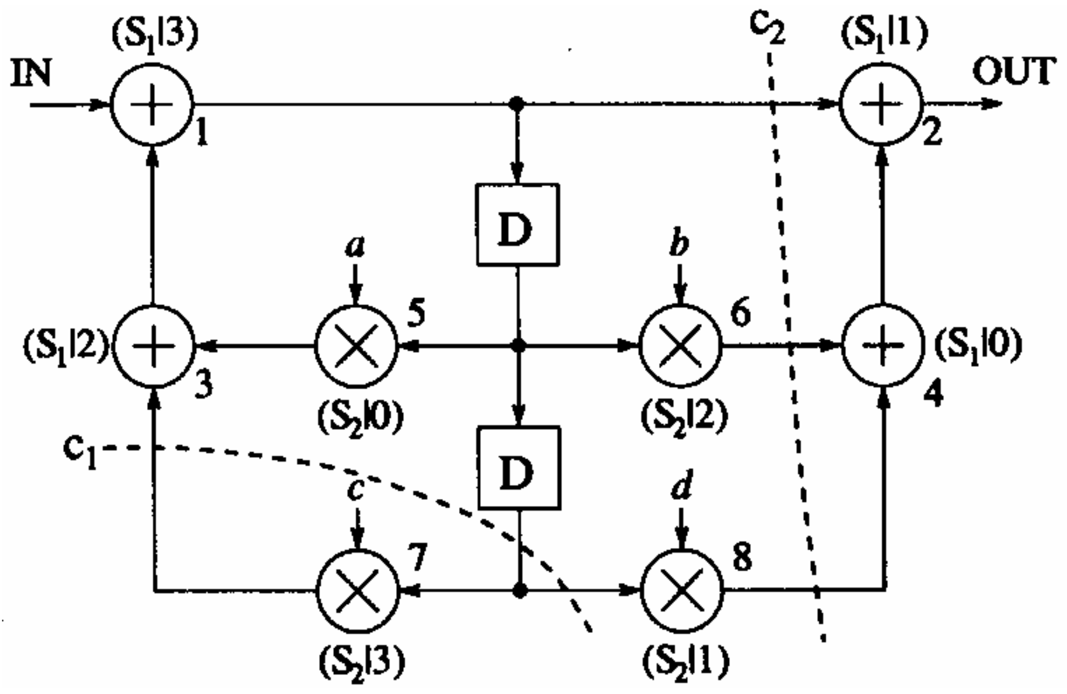
$$r(U) - r(V) \leq \frac{D_F(U \xrightarrow{e} V)}{N}$$

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

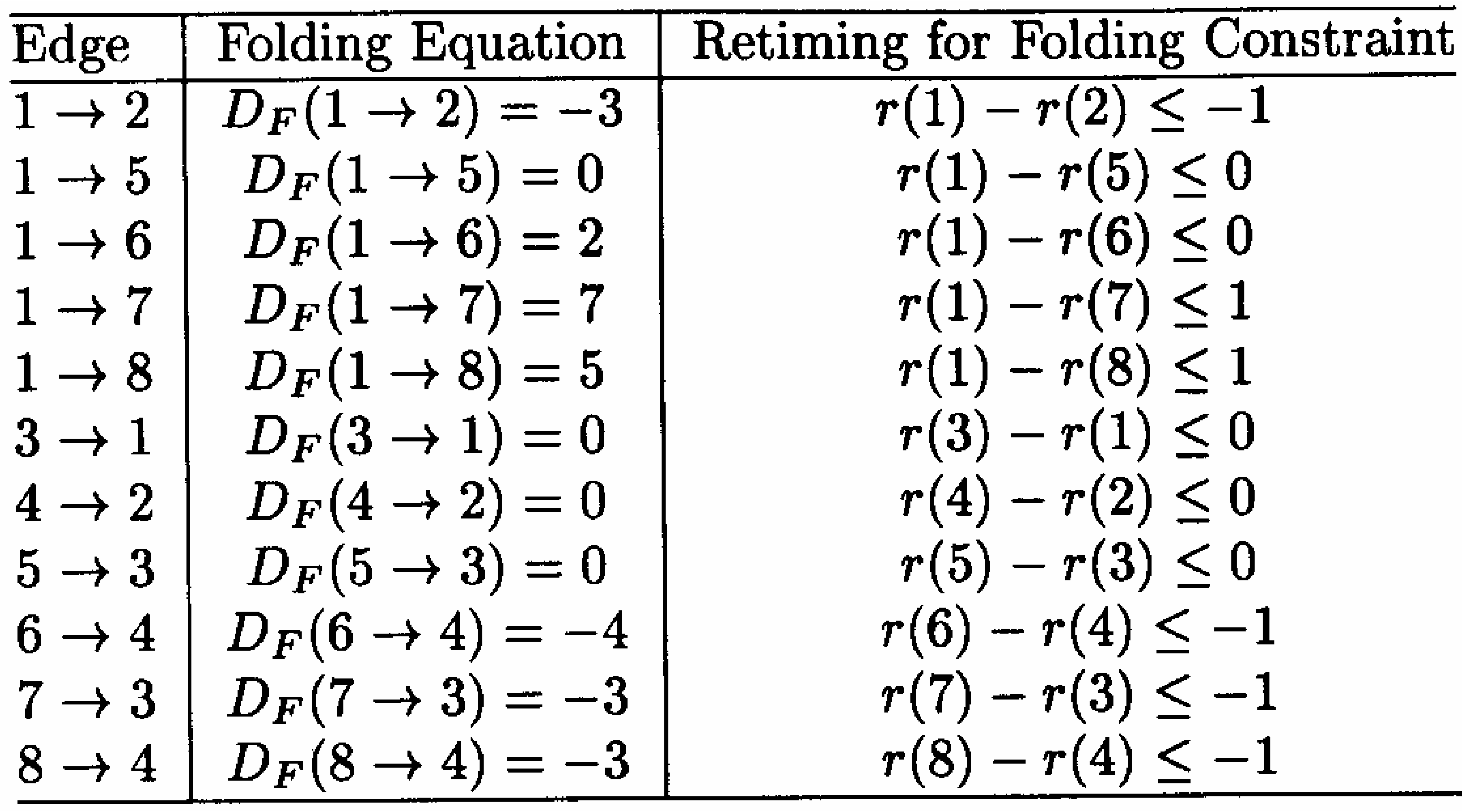
Folding Transformation

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

Exercise

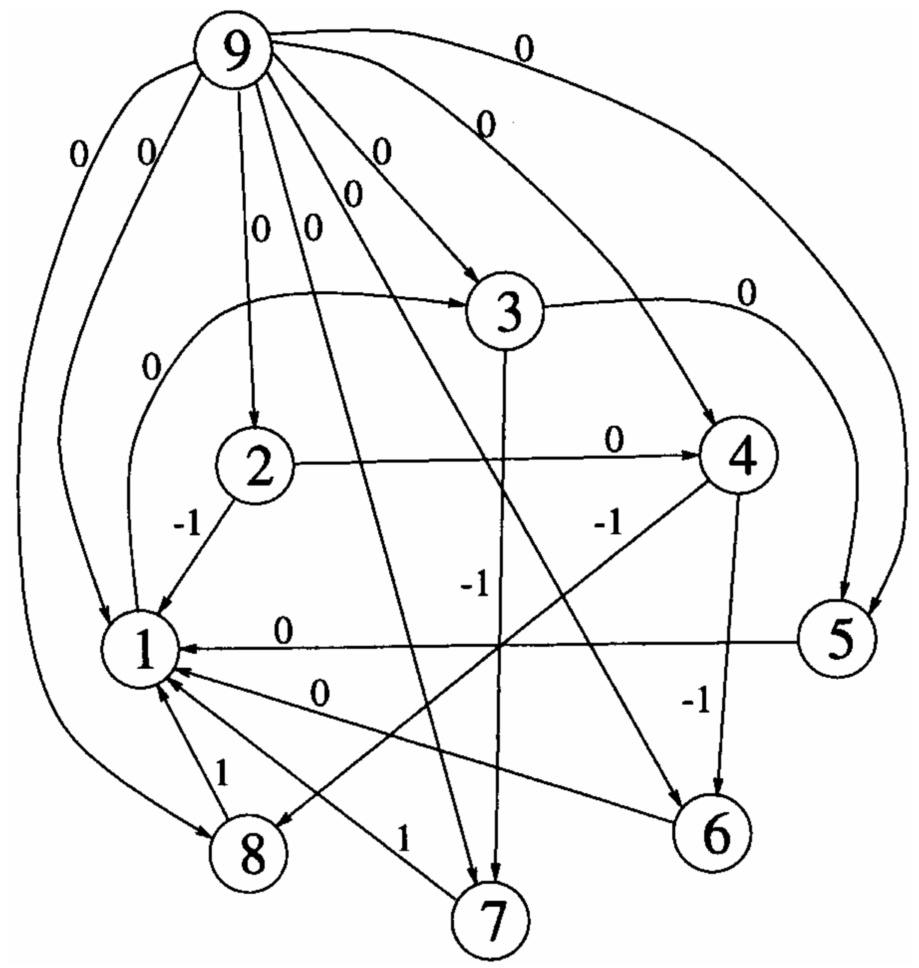


Exercise



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

Exercise

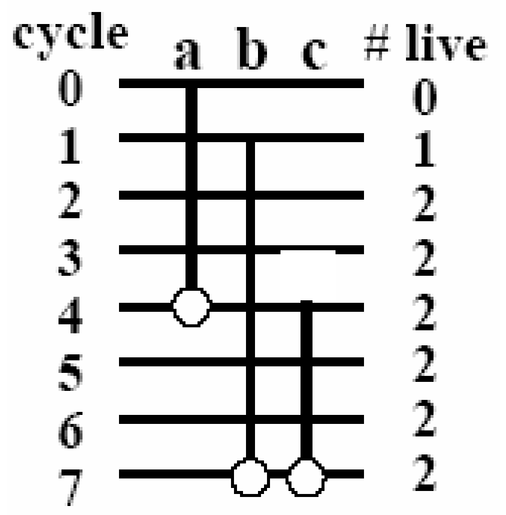


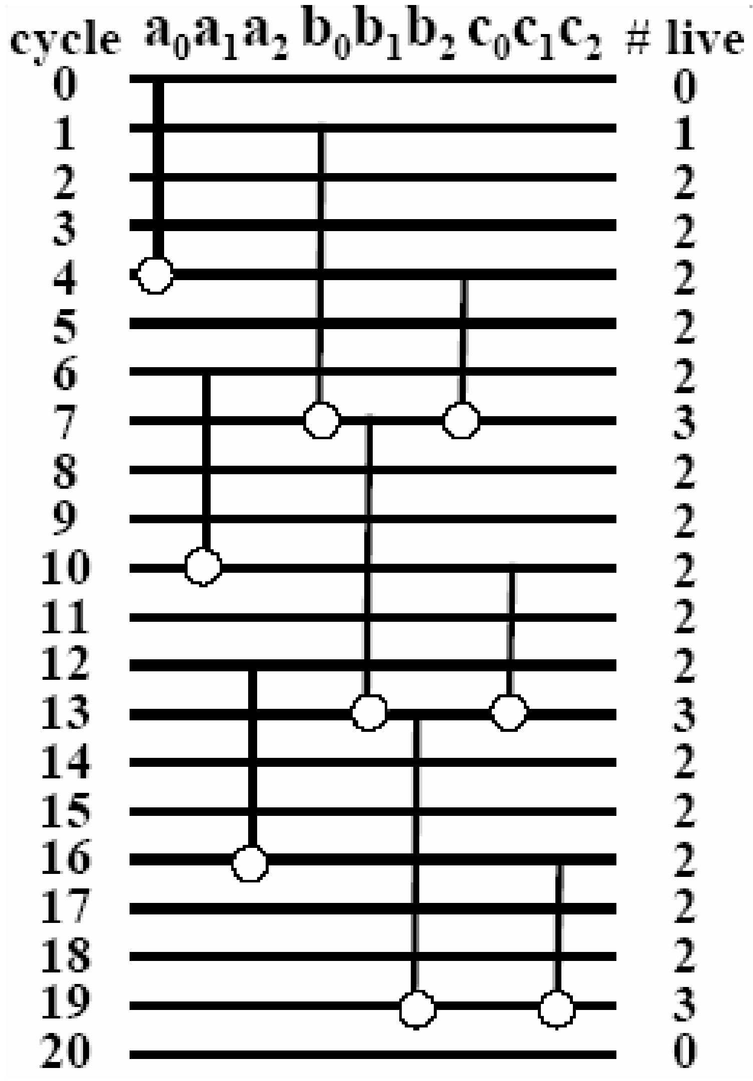
Registers Minimization Techniques

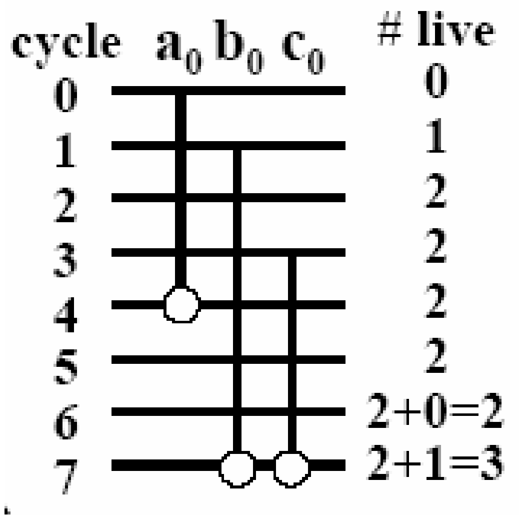
- The objective is to minimize the number of registers in the implementation of a DSP algorithm. Topics
  - Life time analysis
  - Data allocation using forward-backward register allocation
  - Register minimization in folded architecture
  - Examples

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.


cycle
a
b
c
# live
0 1 2 3 4 5 6 7
0 1 2 2 2 2 2 2


cycle
a_0 a_1 a_2 b_0 b_1 b_2 c_0 c_1 c_2
# live
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
0 1 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 3 0


cycle
a_0 b_0 c_0
# live
0 1 2 3 4 5 6 7
0 1 2 2 2 2 2+0=2 2+1=3

Example

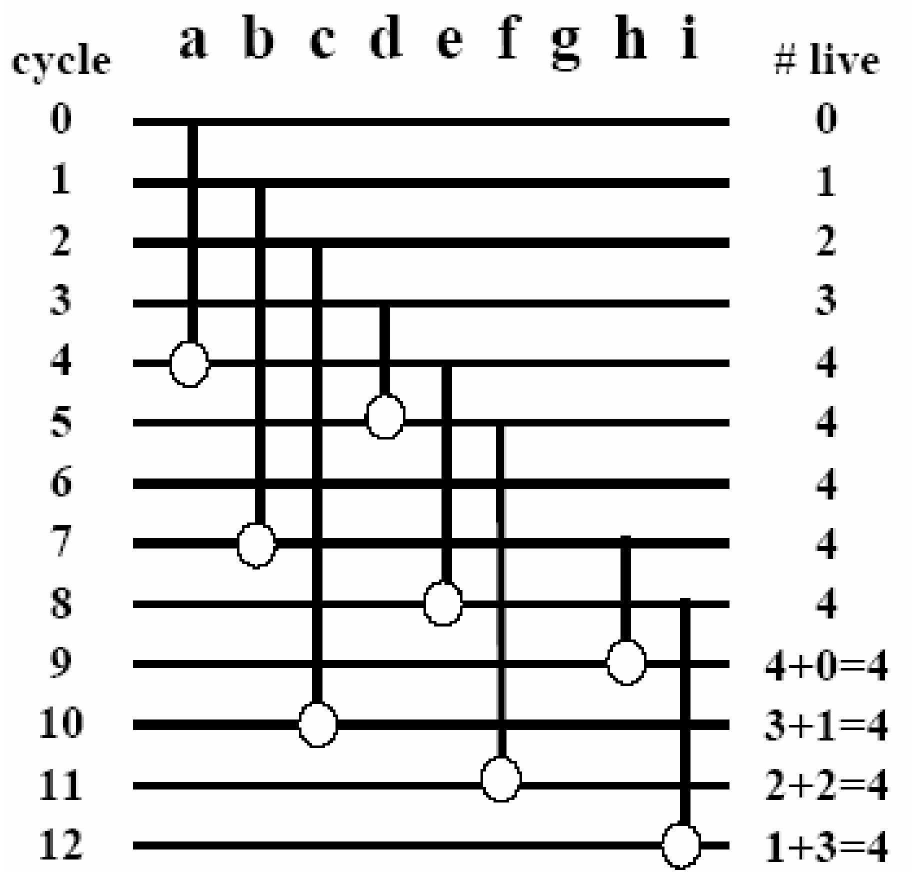
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \xrightarrow{\text{Transpose}} \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

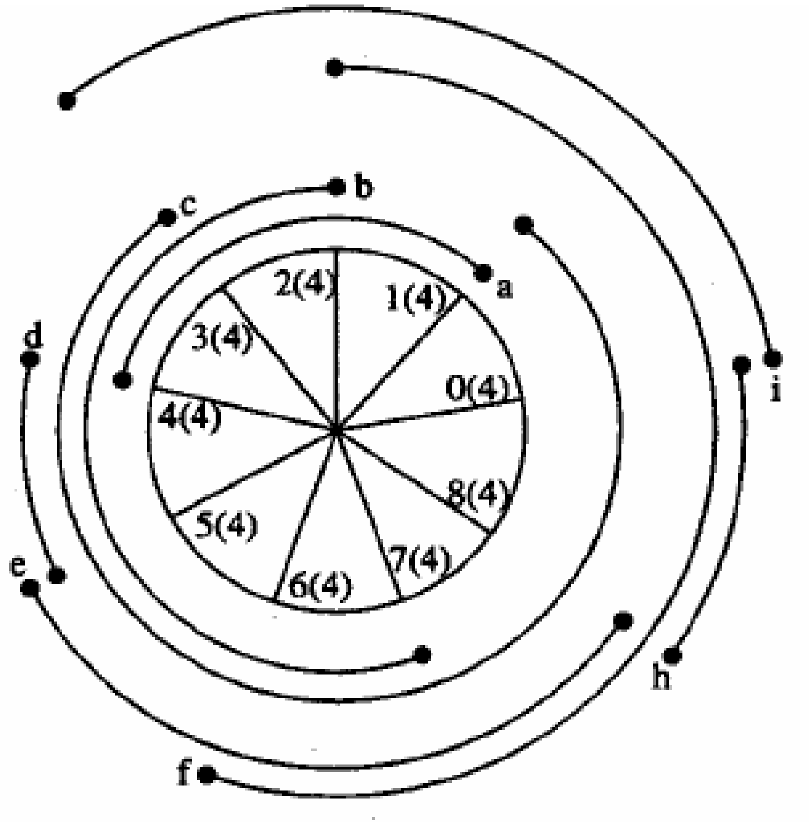
$$\begin{tikzcd} \text{i h g f e d c b a} \arrow[r] & \boxed{\text{Matrix Transpose}} \arrow[r] & \text{i f c h e b g d a} \end{tikzcd}$$

Example

Sample	T_{input}	T_{zout}	T_{diff}	T_{output}	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	$8 \rightarrow 12$

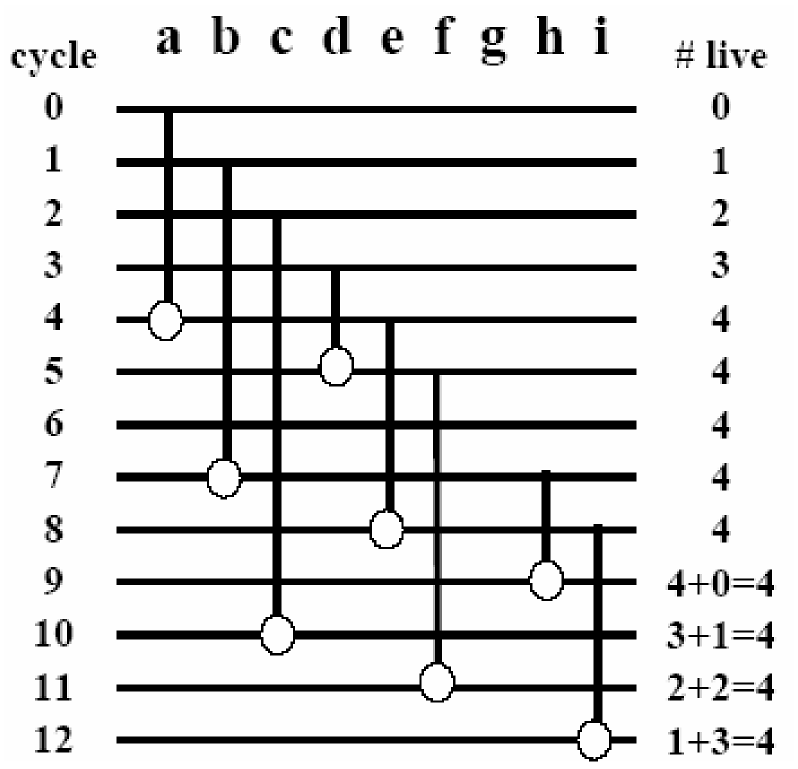
Circular Life-Time Chart





Data Allocation

- Determine the min. number of rgisters
- 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.



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