Unfolding

Shao-Yi Chien

Introduction (1/4)

- **Unfolding** is a transformation technique that can be applied to a DSP program to create a new program describing more than one iterations of the original program
- Unfolding factor J: J consecutive iterations
- Also called as loop unrolling

loop unrolling technique also appears in compiler design. could be executed when the number of looping cycle is known. It increases compile time overhead while speeds up runtime.
(static)
(dynamic)

Introduction (2/4)

- For the DSP algorithm

  for $n=0 \rightarrow \infty$

  - $y(n)=ay(n-9)+x(n)$
- Replace n with 2k and 2k+1

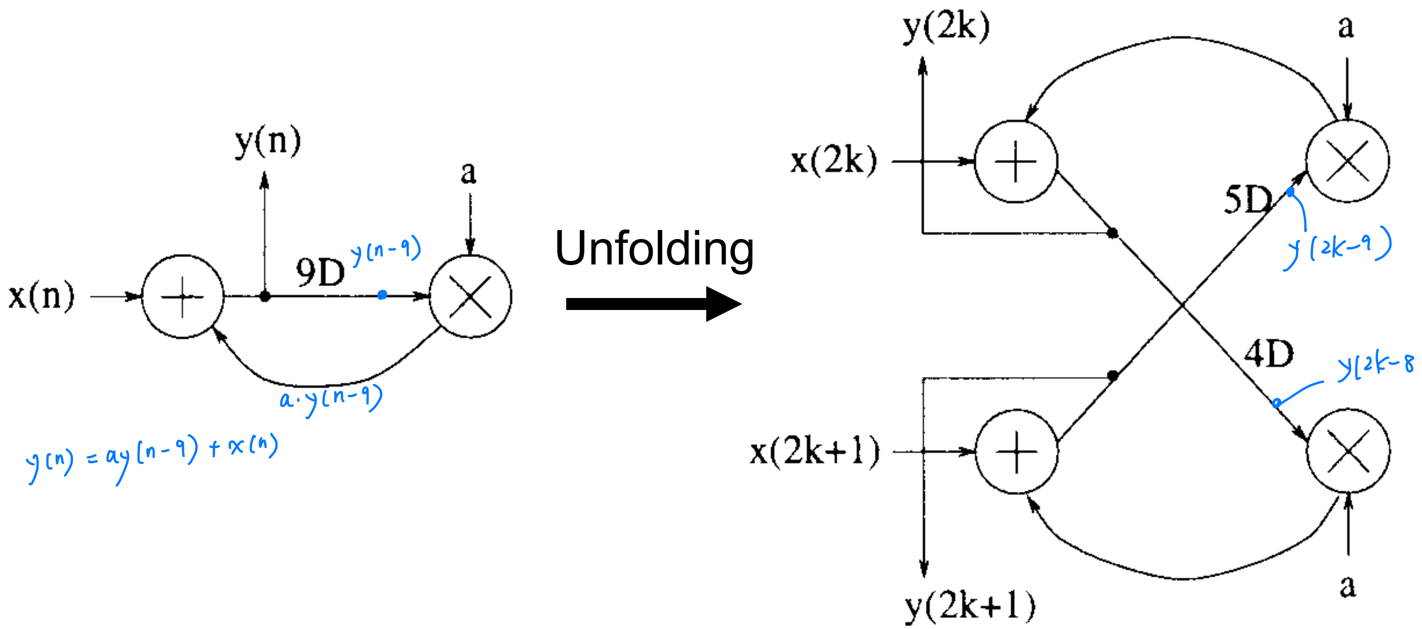
  for $n=0 \rightarrow \infty$

  - $y(2k)=ay(2k-9)+x(2k)$
  - $y(2k+1)=ay(2k-8)+x(2k+1)$
- It is an unfolded algorithm with J=2!

Introduction (3/4)

$y(2k) = ay(2k-9) + x(2k)$

$y(2k+1) = ay(2k-8) + x(2k+1)$



- Note that, in unfolded systems, each delay is J-slow

Introduction (4/4)

- Applications of unfolding
  - To reveal hidden concurrent so that the program can be scheduled to a smaller iteration period
  - To design parallel architecture

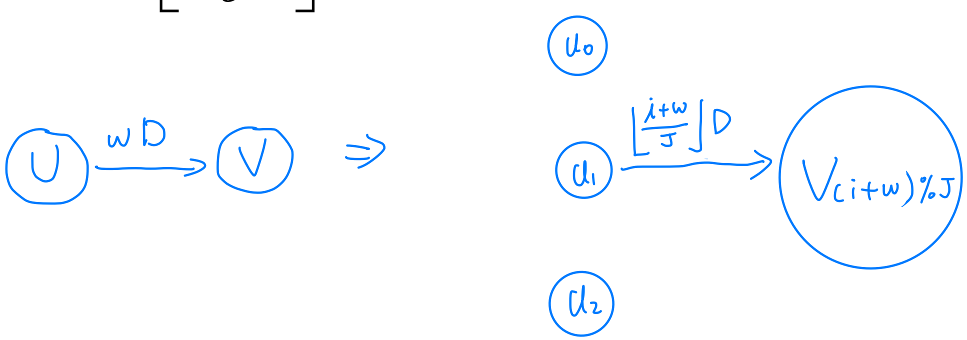
Algorithm for Unfolding

- In the J-unfolded DFG
  - For each node U in the origin DFG, there are J nodes with the same function as U
  - For each edge in the original DFG, there are J edges

produce J copies of DFG D to D′

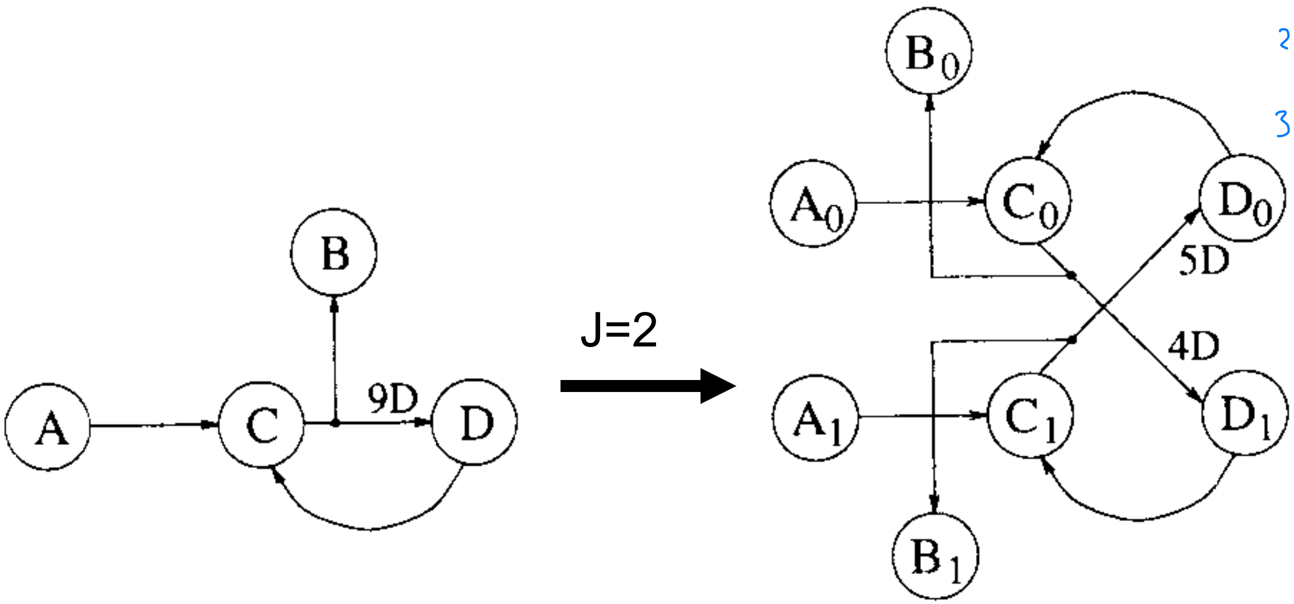
Algorithm for Unfolding

- For each node U in the original DFG, draw the J nodes $U_0, U_1, \ldots, U_{J-1}$
- For each edge U→V with w delays in the original DFG, draw the J edges $U_i \rightarrow V_{(i+w)\%J}$ with $\left\lfloor \frac{i+w}{J} \right\rfloor$ delays for i=0, 1, …, J-1



Example 1 of Unfolding

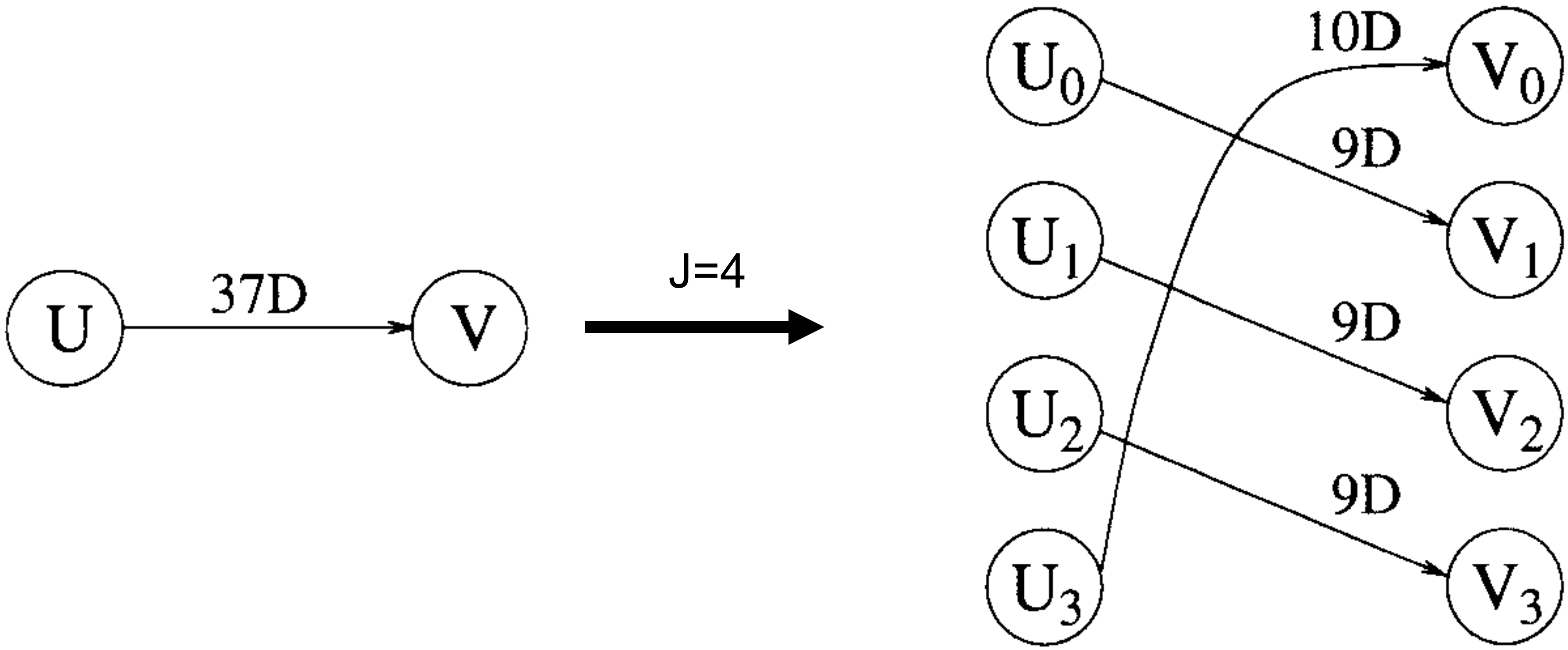
1. Duplicate J nodes
2. construct links with no delay
3. $U_i \xrightarrow[\lfloor \frac{i+w}{J} \rfloor D]{} V_{(i+w)\%J}$



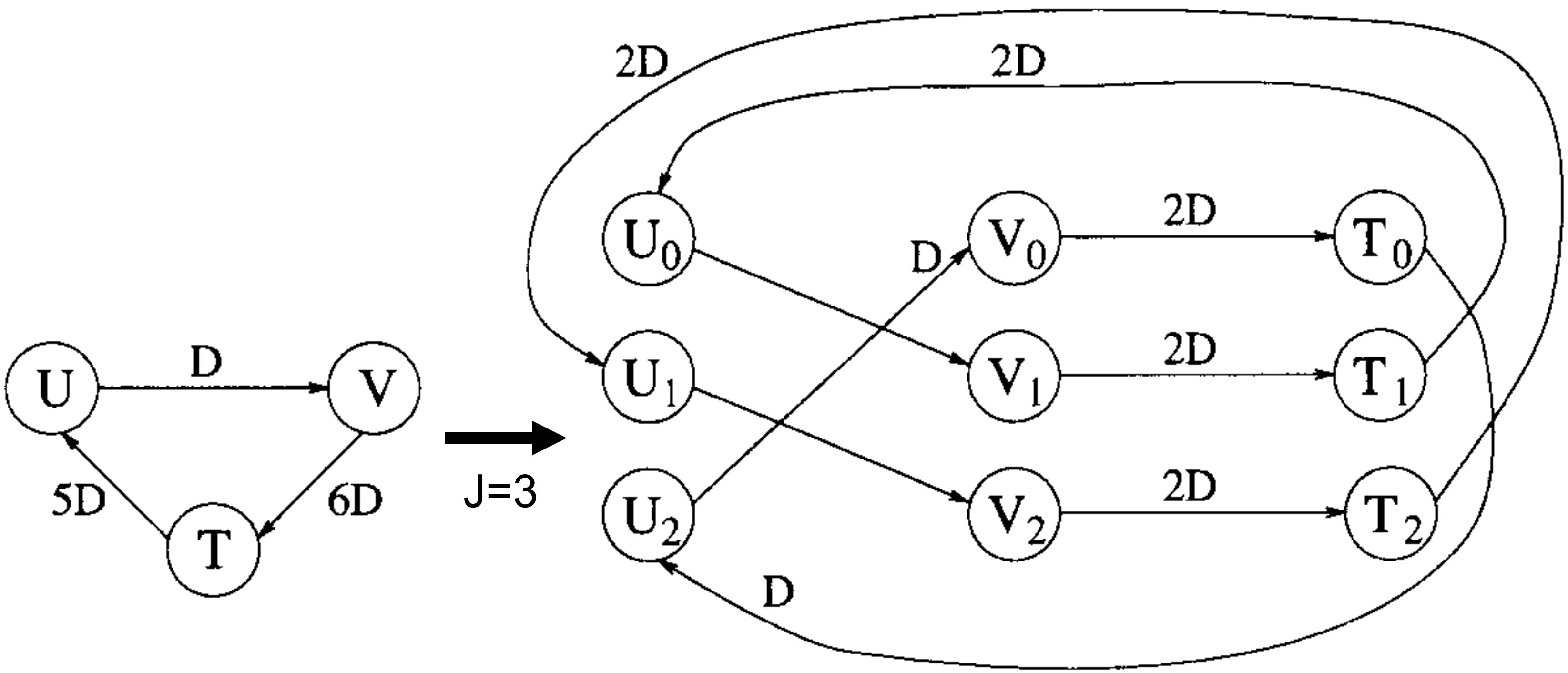
$$C_0 \rightarrow \overset{=D_1}{D_{(0+9)\%2}} \text{ with } \left\lfloor \frac{0+9}{2} \right\rfloor^{=4} \text{delays}$$

$$C_1 \rightarrow \overset{=D_0}{D_{(1+9)\%2}} \text{ with } \left\lfloor \frac{1+9}{2} \right\rfloor^{=5} \text{delays}$$

Example 2 of Unfolding



Example 3 of Unfolding

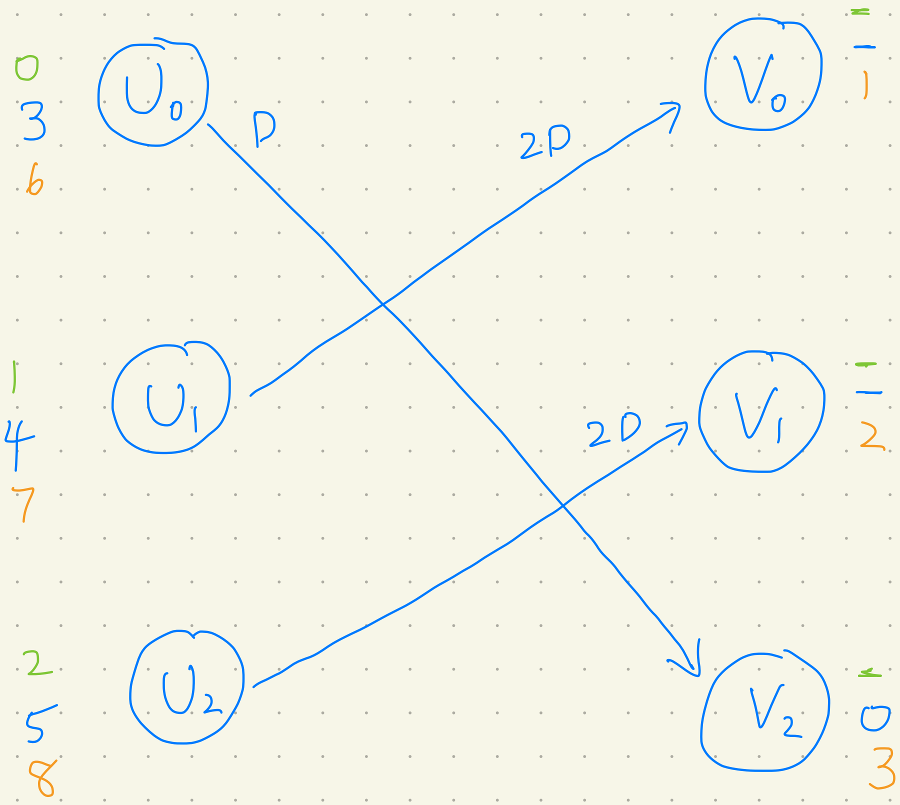


$$U_i \xrightarrow{\left\lfloor \frac{i+w}{J} \right\rfloor D} V_{(i+w)\%J}$$

$K = 0\ 1\ 2$

$$U \xrightarrow{5D} V$$

n	U	V
0	0	—
1	1	—
2	2	—
3	3	—
4	4	—
5	5	0
6	6	1
7	7	2
8	8	3



Proof of the Unfolding Algorithm (1/2)

- Unfolding preserve precedence constraints of a DSP program
- For $U_i \rightarrow V_{(i+w)\%J}$ with $\left\lfloor \frac{i+w}{J} \right\rfloor$ delays

  output of U_i in the k-th iteration will be connected to $V_{(i+w)\%J}$ in the (k+ $\left\lfloor \frac{i+w}{J} \right\rfloor$)-th iteration
- In the original DFG, it corresponds to:

  output of U in the (Jk+i)-th iteration will be connected to V in the $(J(k+\left\lfloor \frac{i+w}{J} \right\rfloor)+(i+w)\%J)$-th iteration

Proof of the Unfolding Algorithm (2/2)

$$\overbrace{J\left(\cancel{k}+\left\lfloor\frac{i+w}{J}\right\rfloor\right)+(i+w)\%J}^{\text{arrive } V}-\overbrace{(\cancel{Jk}+i)}^{\text{left } U}$$

$$=\left(J\left\lfloor\frac{i+w}{J}\right\rfloor+(i+w)\%J\right)-i$$

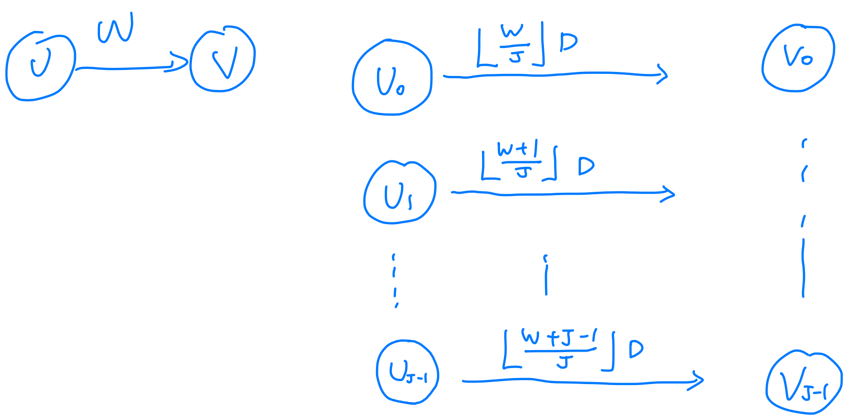
$$=(i+w)-i=w$$

- So the precedence constraints are preserved correctly

Properties of Unfolding (1/5)

- Unfolding preserves the number of delays in a DFG

$$\left\lfloor \frac{w}{J} \right\rfloor + \left\lfloor \frac{w+1}{J} \right\rfloor + \cdots + \left\lfloor \frac{w+J-1}{J} \right\rfloor = w$$



$\Rightarrow$ Reg after transform $= \sum$ reg after transform

Properties of Unfolding (2/5)

- J-unfolding of a loop l with w_l delays in the original DFG leads to gcd(w_l, J) loops in the unfolded DFG, and each of these gcd(w_l, J) loops contains w_l/gcd(w_l, J) delays and J/gcd(w_l, J) copies of each node that appears in l

Properties of Unfolding (3/5)

- For a loop in origin loop A→A traversed p times with w_l delay elements
- In the unfolded DFG: $A_i \rightarrow A_{(i+pw_l)\%J}$
- This path form a loop if $i = (i + pw_l)\%J$

$\Rightarrow J \mid pw_l$ min. $pw_l = \mathrm{lcm}(J, w_l)$

$\gcd \cdot \mathrm{lcm} = J \cdot w_l$

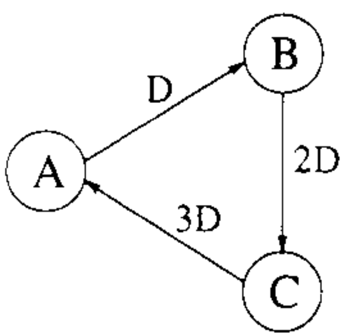
$\gcd \cdot p \cdot w_l = J \cdot w_l$

$p = \frac{J}{\gcd(J, w_l)}$, thus there are $\frac{J}{p} = \gcd(J, w_l)$ loops

Properties of Unfolding (4/5)

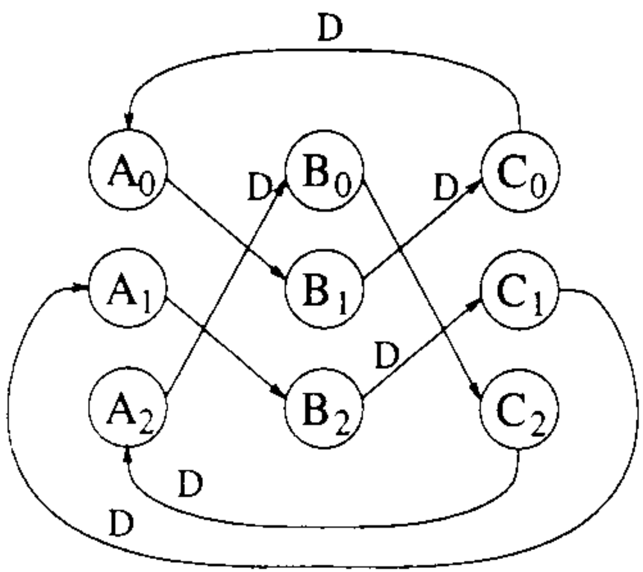
3-loops = gcd(3,6)

2-loops = gcd(6,4)



Origin DFG

w_l=6



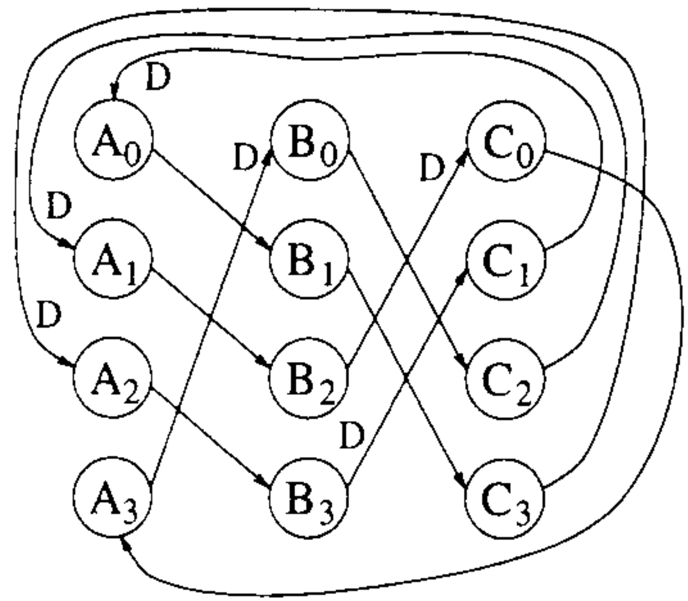
J=3

For a loop, i=(i+6p)%3

p=1

Loop: A→B→C→A

Consists of 3 loops



J=4

For a loop, i=(i+6p)%4

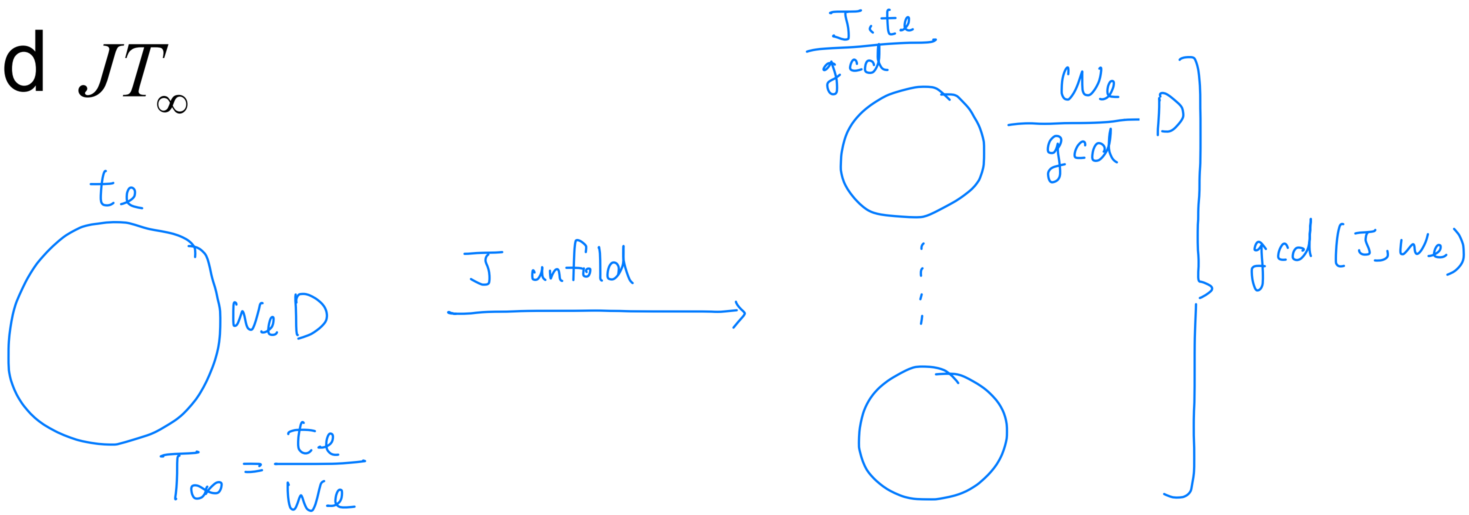
p=2

Loop: A→B→C→A→B→C→A

Consists of 2 loops

Properties of Unfolding (5/5)

- Unfolding a DFG with iteration bound T_∞ results in a J-unfolded DFG with iteration bound JT_∞



$$T_\infty = \frac{\frac{J \cdot t_e}{gcd}}{\frac{w_e}{gcd}} = J \cdot \frac{t_e}{w_e} = J T_\infty$$

Retiming with Unfolding (1/2)

- Consider a path with w delays in the original DFG. J-unfolding of this path leads to (J-w) paths with no delays and w paths with 1 delay each, when w<J $\xrightarrow{2D}$ $\xrightarrow{J=3}$ $\rightarrow$ $\xrightarrow{D}$ $\xrightarrow{D}$ may form new critical path,
- Any path in the original DFG containing J or more delays leads to J paths with 1 or more delays in each path. Therefore, a path in the original DFG with J or more delays cannot create a critical path in the J-unfolded DFG

Retiming with Unfolding (2/2)

- The critical path of the unfolded DFG can be c if there exists a path in the original DFG with computation time c and less than J delay elements

- If $D(U,V) >= c$, $W_r(U,V) = W(U,V) + r(V) - r(U) >= J$
- $w(e) + r(V) - r(U) >= 0$ feasibility constraint

In order to form no new critical paths, retiming before unfolding

Applications of Unfolding

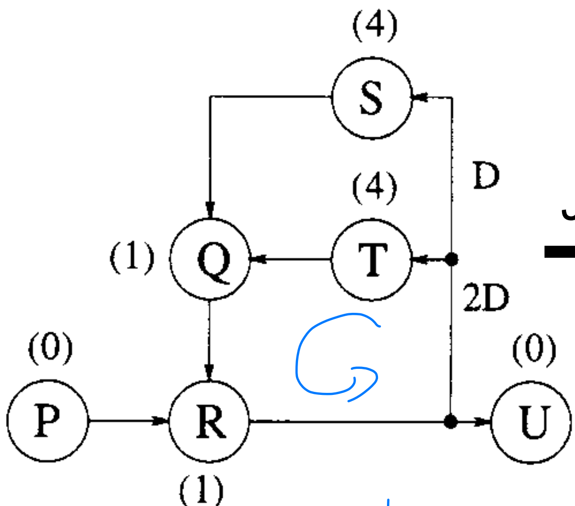
- Sample period reduction
- Parallel processing
  - Word-level parallel processing
  - Bit-level parallel processing

Sample Period Reduction (1/5)

e.g. when fine-grain pipelining wouldn't work. Like a given IP.

- In some cases, the DSP program cannot be implemented with iteration period equal to the iteration bound ➔ use unfolding
- First case: there is a node in the DFG that has computation time greater than T_∞
  - If t_U is greater than the iteration bound, then $\lceil t_U / T_\infty \rceil$-unfolding should be used

Sample Period Reduction (2/5)

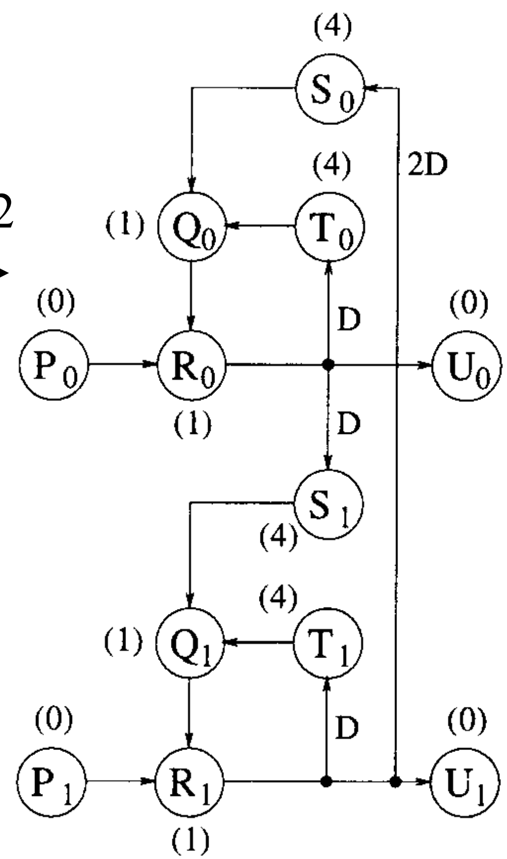


$\frac{4+1+1}{2} = 3$

$T_\infty = 3$

Minimum sample period=4

J= $\lceil 4/3 \rceil = 2$



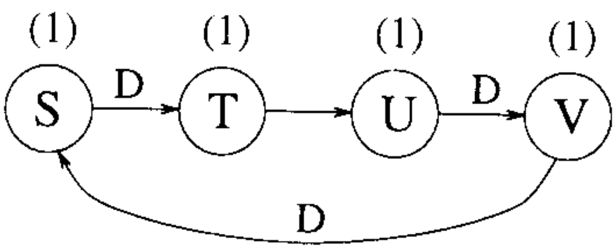
$T_\infty = 6$

Minimum sample period=6/2

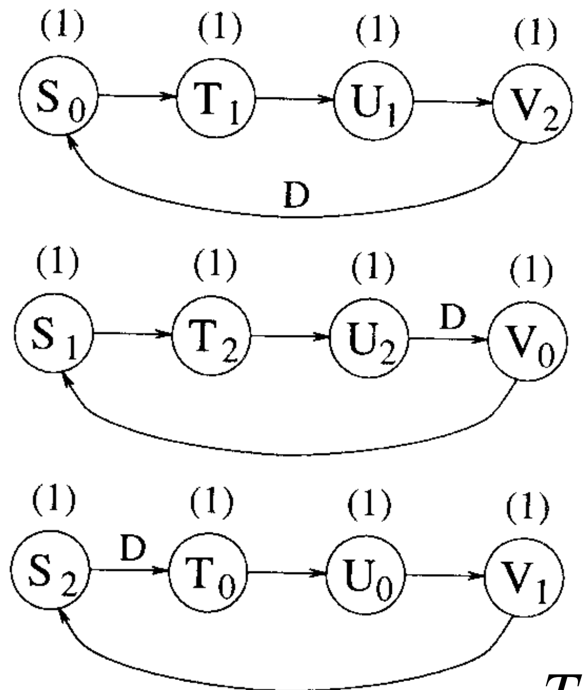
Sample Period Reduction (3/5)

- Second case: the iteration bound is not an integer
  - If a critical loop bound is of the form t_l/w_l, where t_l and w_l are mutually coprime, then w_l-unfolding should be used

Sample Period Reduction (4/5)



J=3



T_∞ =4/3
Minimum sample period=2

T_∞=4
Minimum sample period=4/3

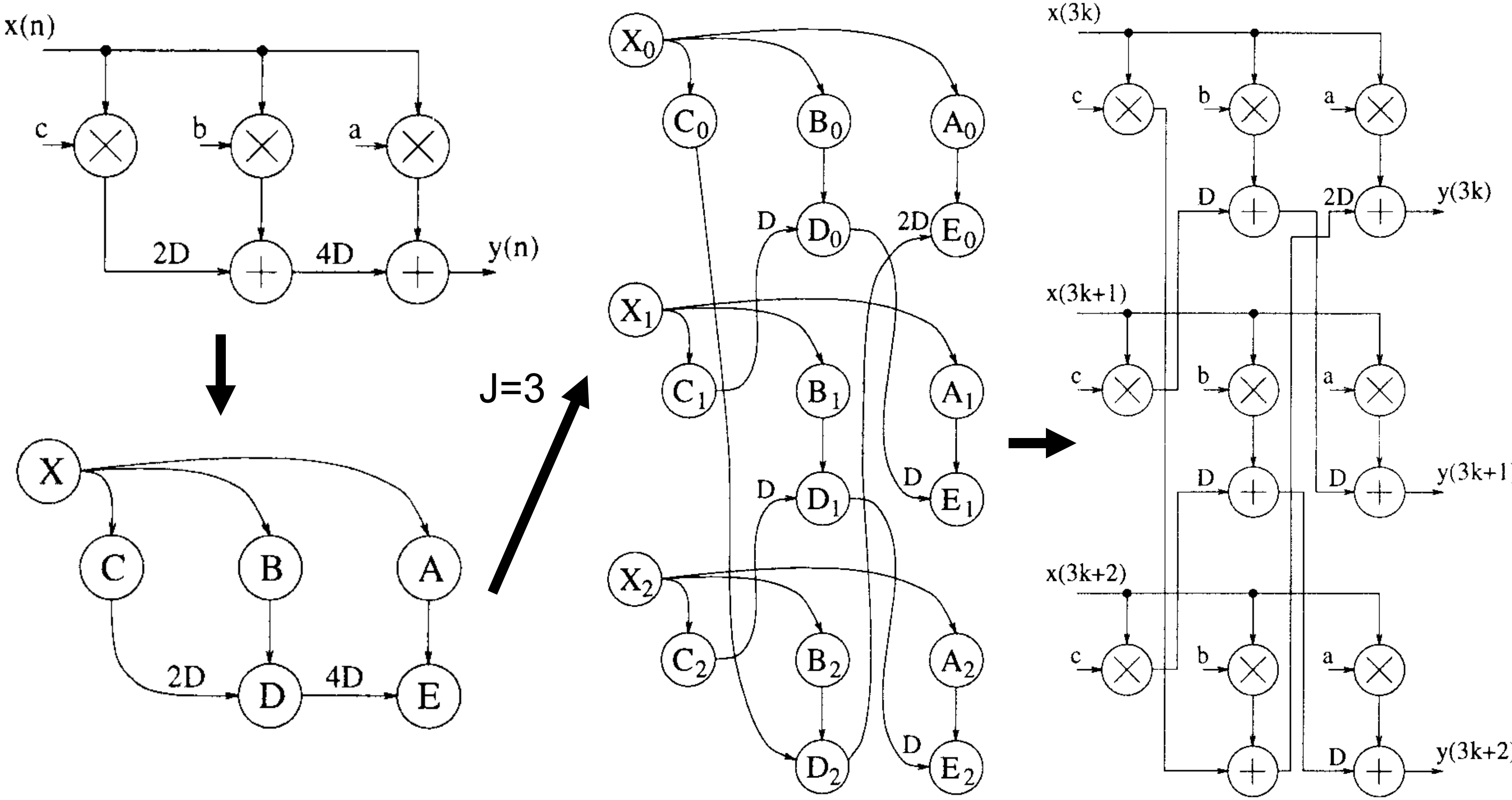
Sample Period Reduction (5/5)

- For both cases, where the longest node computation time is larger than the iteration bound T_∞, and T_∞ is not an integer
  - J is the minimum value such that JT_∞ is an integer and is greater than or equal to the longest node computation time

Word-Level Parallel Processing (1/2)

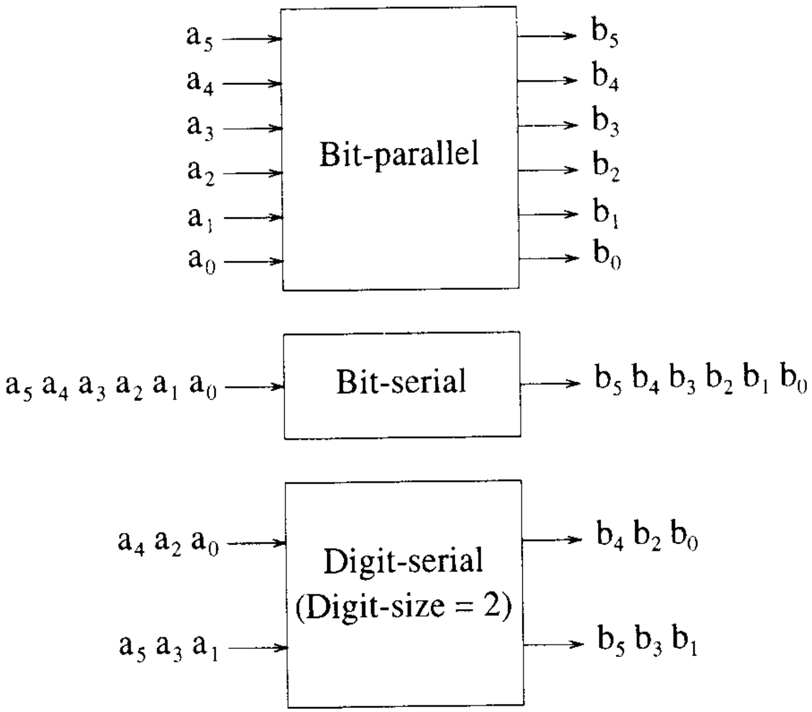
- The unfolding technique can be used to design a word-parallel architecture from a word-serial architecture
  - Unfolding a word-serial architecture by J creates a word-parallel architecture that processes J words per clock cycle
  - Parallel processing

Word-Level Parallel Processing (2/2)



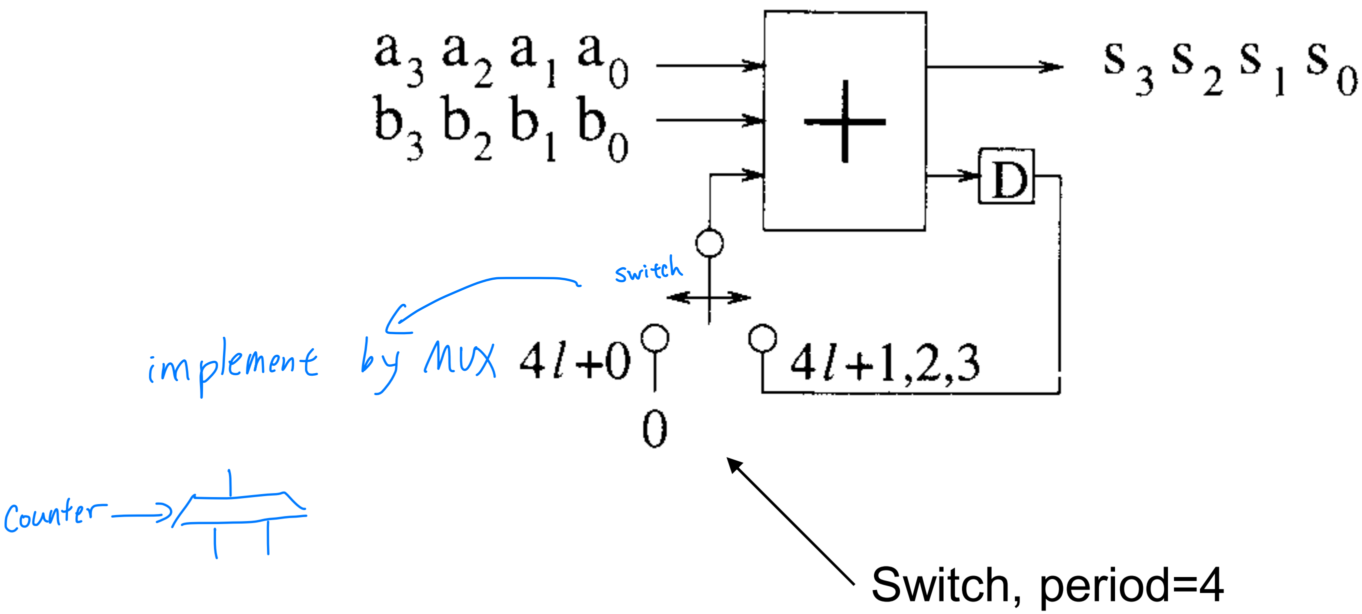
Bit-Level Parallel Processing (1/6)

- Bit-parallel and bit-serial architecture can be derived from bit-serial architectures using the unfolding transformation
  - Bit-serial
  - Bit-parallel: word-length W
  - Digit-serial: N digits



Bit-Level Parallel Processing (2/6)

- Bit-serial adder for W=4

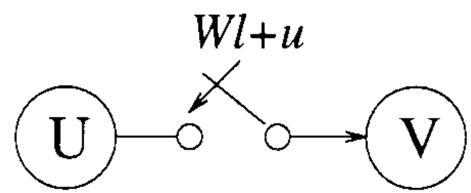


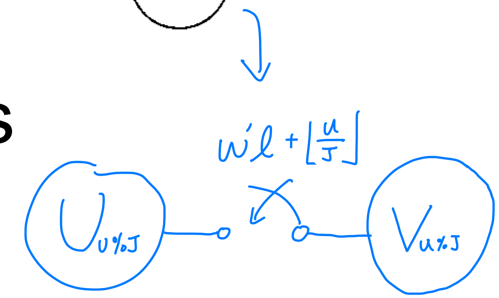
Bit-Level Parallel Processing (3/6)

- Unfolding the switch
  - Assume W=W'J
  - Assume all edges have no delays
  - Write the switch instance as

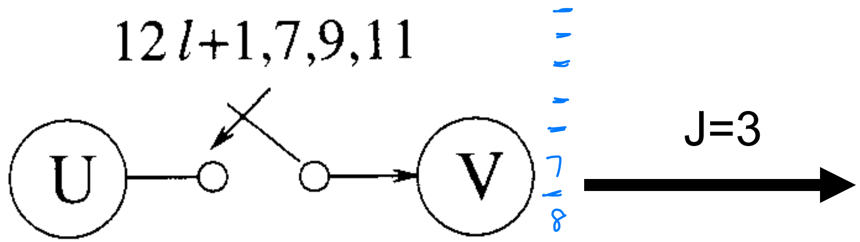
$$Wl + u = J\left(W'l + \left\lfloor \frac{u}{J} \right\rfloor\right) + (u\%J).$$

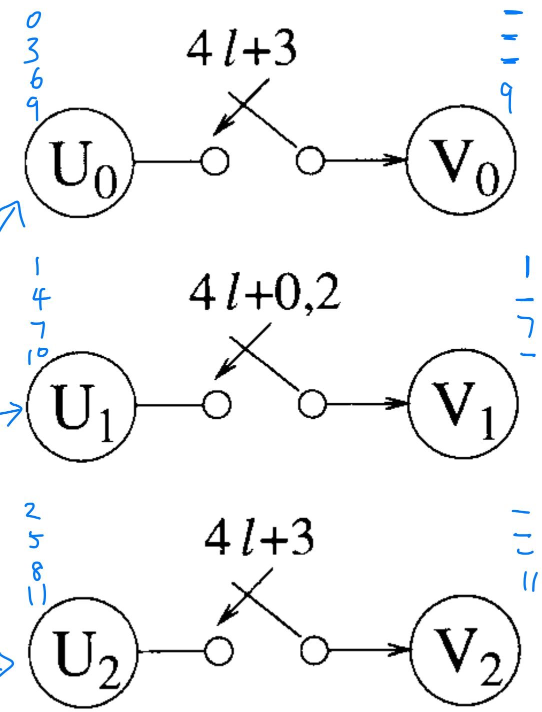
  - Draw an edge with no delays in the unfolded graph from the node $U_{u\%J}$ to the node $V_{u\%J}$, which is switched at time instance $\left(W'l + \left\lfloor \frac{u}{J} \right\rfloor\right)$





Bit-Level Parallel Processing (4/6)

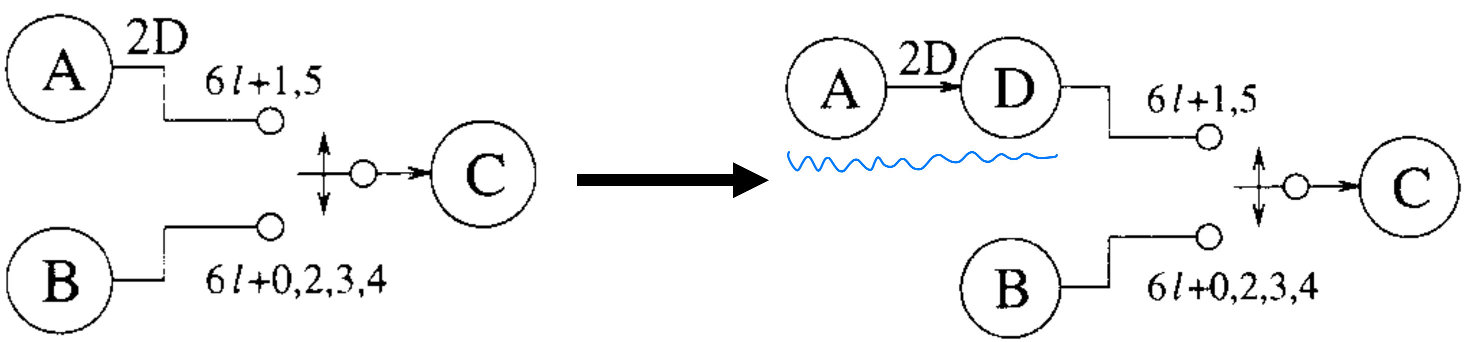




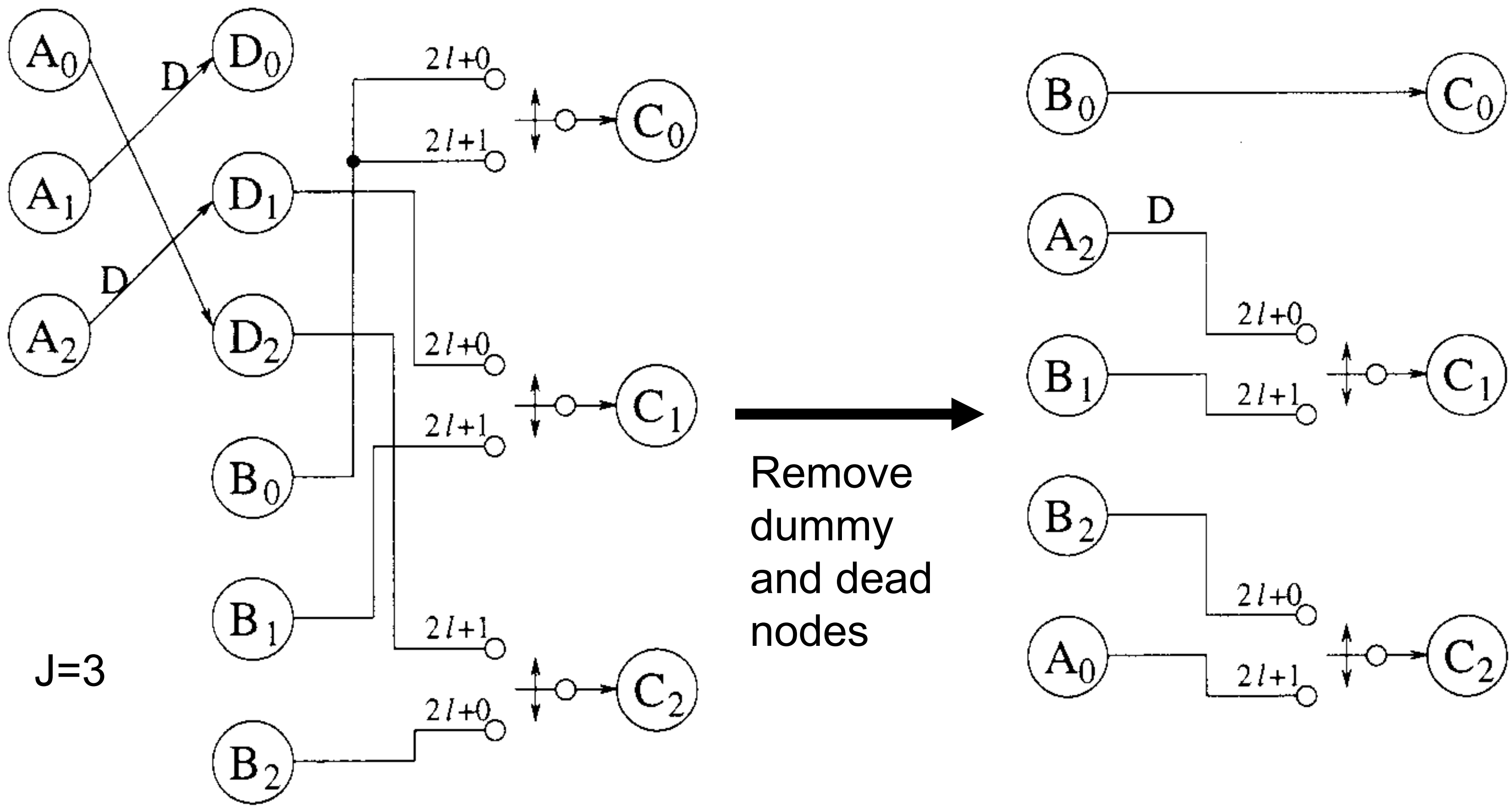
$$
\begin{aligned}
12l+1 &= 3(4l+0)+1 \\
12l+7 &= 3(4l+2)+1 \\
12l+9 &= 3(4l+3)+0 \\
12l+11 &= 3(4l+3)+2.
\end{aligned}
$$

Bit-Level Parallel Processing (5/6)

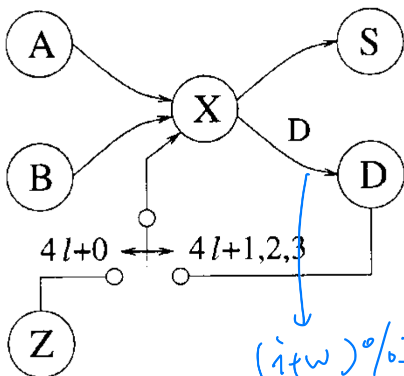
- For edges with delays
  - Add dummy nodes



Bit-Level Parallel Processing (6/6)



Bit-Parallel Adder



$4l+0 = 4(l+0)+0$

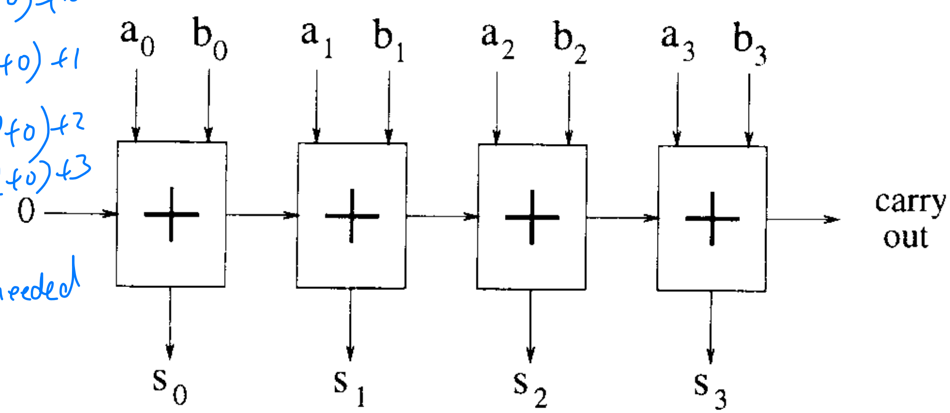
$4l+1 = 4(l+0)+1$

$4l+2 = 4(l+0)+2$

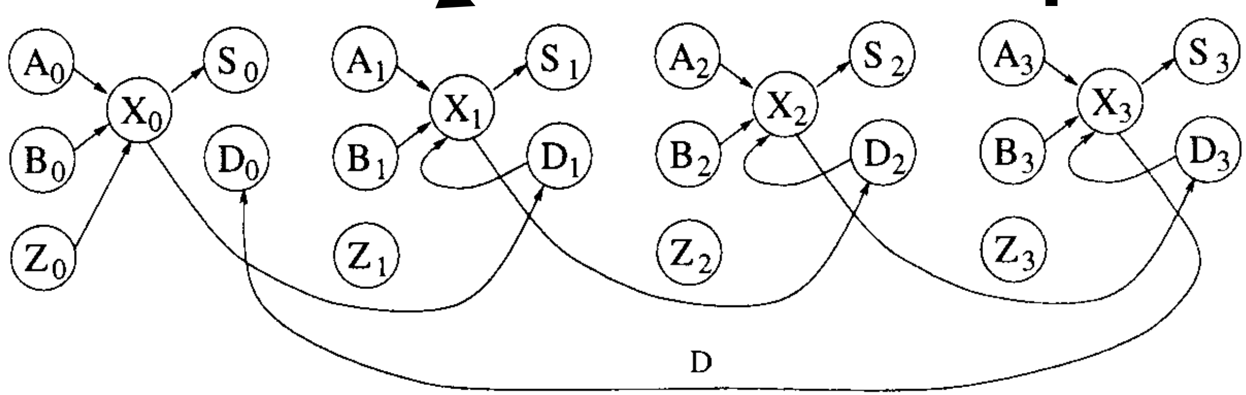
$4l+3 = 4(l+0)+3$

⇒ no switch needed

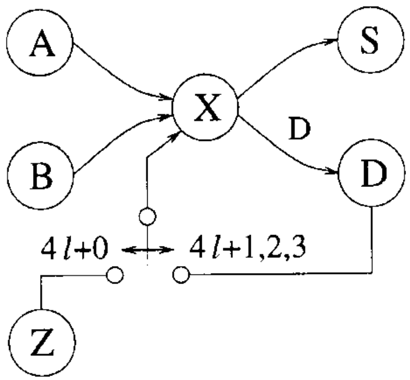
$(i+w)\%J$



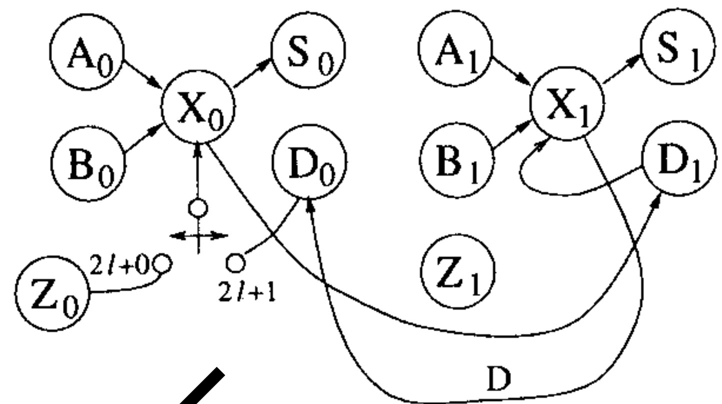
J=W=4



Digit-Serial Adder (1/4)



J=2

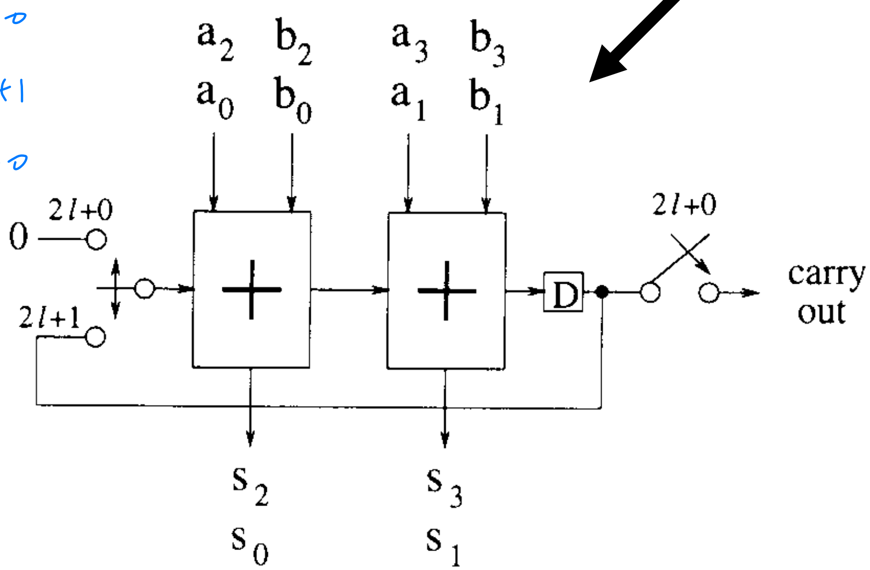


$4l+0 = 2(2l+0)+0$

$4l+1 = 2(2l+0)+1$

$4l+2 = 2(2l+1)+0$

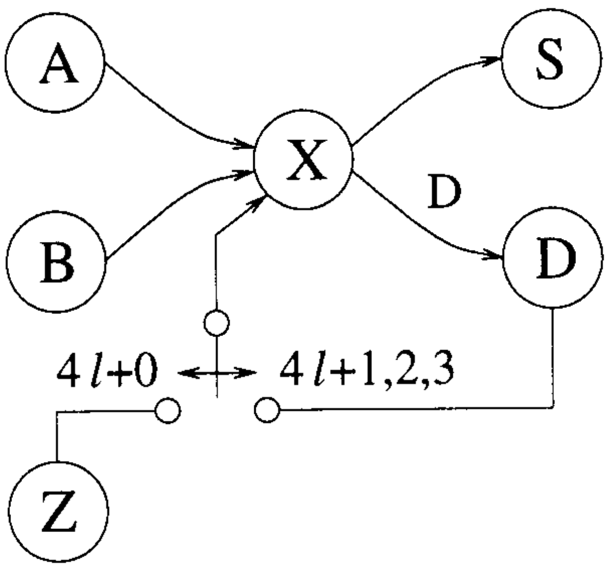
$4l+3 = 2(2l+1)+1$



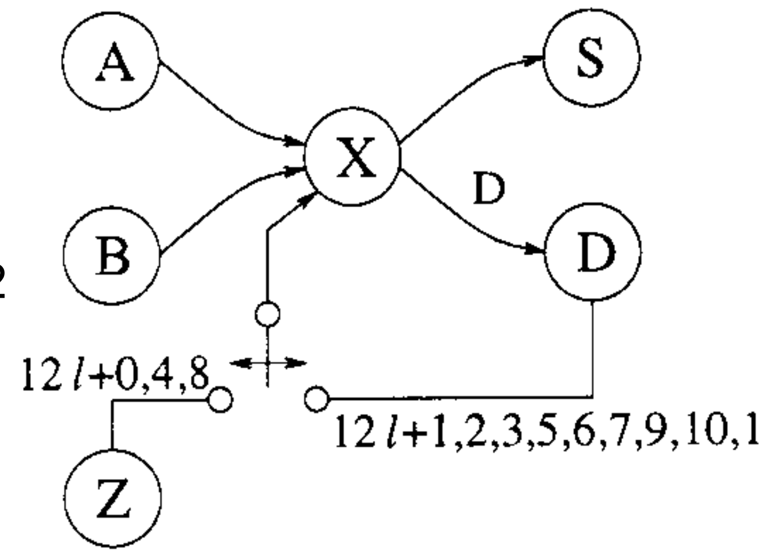
Digit-Serial Adder (2/4)

- If W is not a multiple of the unfolding factor J
  - L=lcm{W,J}
  - Replace the period of the switch W as L

Digit-Serial Adder (3/4)



J=3
W=4
L=lcm{3,4}=12



Digit-Serial Adder (4/4)