Chapter 5

Algorithm and Architecture Exploration in IP Core Design

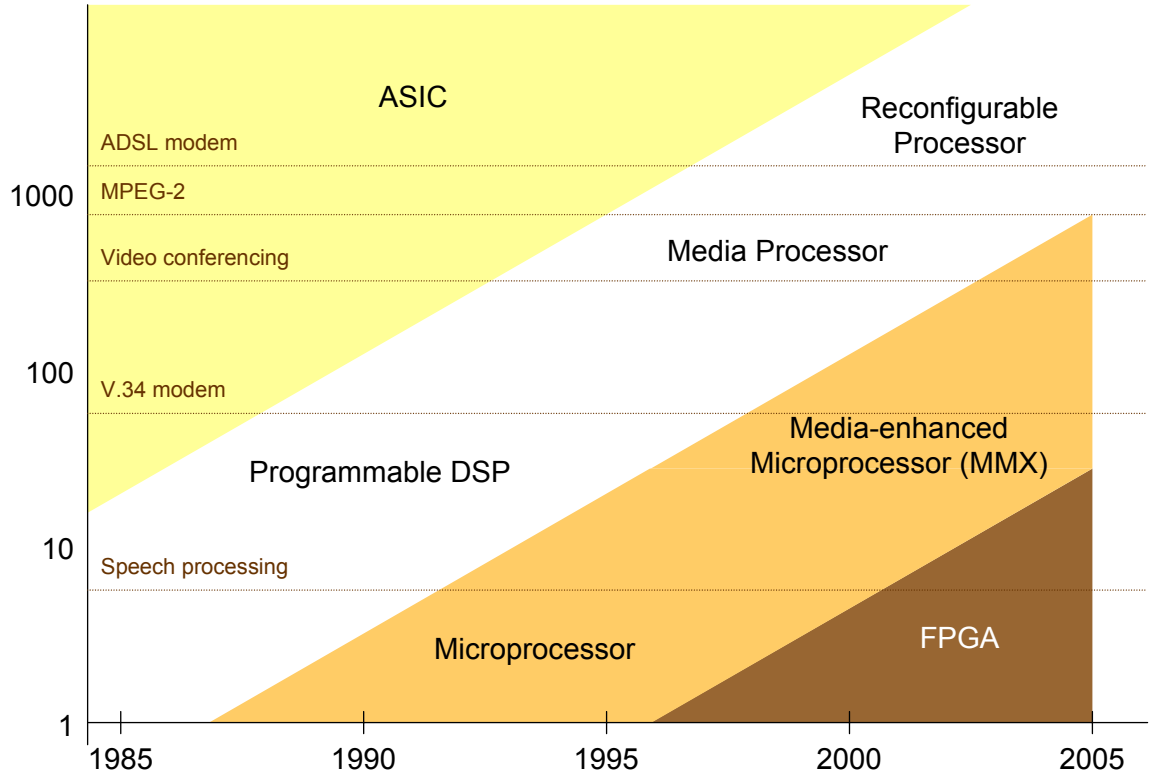
Chein-Wei Jen

VLSI Signal Processing Group
Department of Electronics Engineering
National Chiao Tung University

Outline

- ***Introduction***
- Algorithm Representation and Iteration Bound
- Retiming
- Unfolding and Folding Transformation
- Bit-serial Architectures
- Case Study – DWT Processor

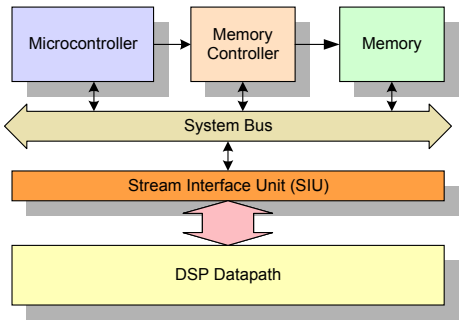
Evolution of Applications



Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Introduction
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Hybrid Approach

- As the complexity of embedded computing grows rapidly, it is common to accelerate critical tasks with hardware, which operates under the software control
- Off-the-shelf components or licensed IP cores significantly shorten the time to market
- The traffic on the system bus can be reduced by
  - Cache
    - optimized for micro-controllers
    - explores the ***spatial & temporal locality*** with high-speed access
  - Stream interface unit (SIU)
    - customized for DSP datapaths
    - explores ***data reuse*** existing in most DSP algorithms with high-bandwidth data streaming
    - performs ***data format conversion*** as necessary



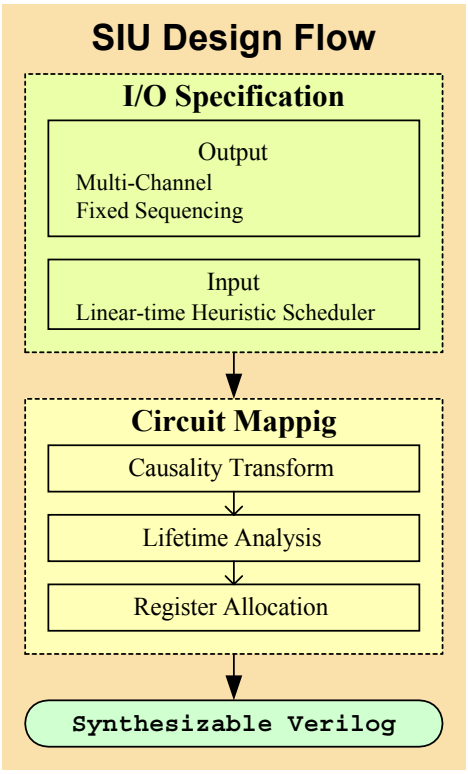
SIU Generation

Stage 1: I/O specification

specify the number of I/O channels & data sequencing

Stage 2: circuit mapping

equivalent to the multi-channel data format conversion (DFC) problem



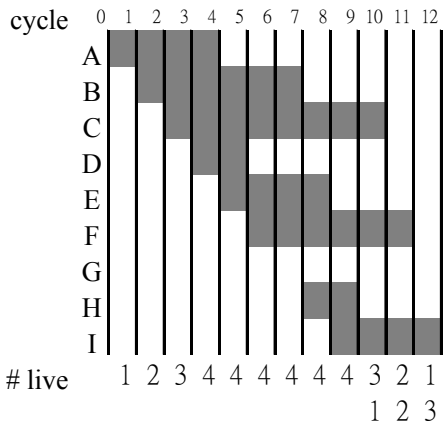
Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Introduction
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Example 5-1 (1/2)

- Matrix transposition
  input sequence: ABCDEFGHI
  output sequence: ADGBEHCFI

Step 1: lifetime analysis

Sample	Tinput	Tzlout	Tdiff	Toutput	Life Period
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

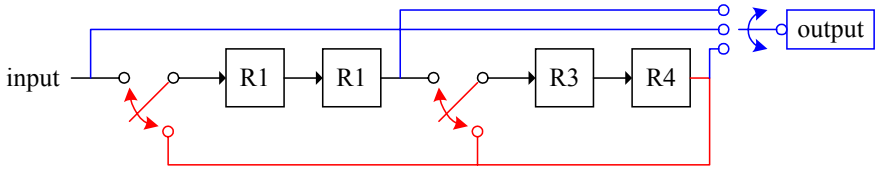


Example 5-1 (2/2)

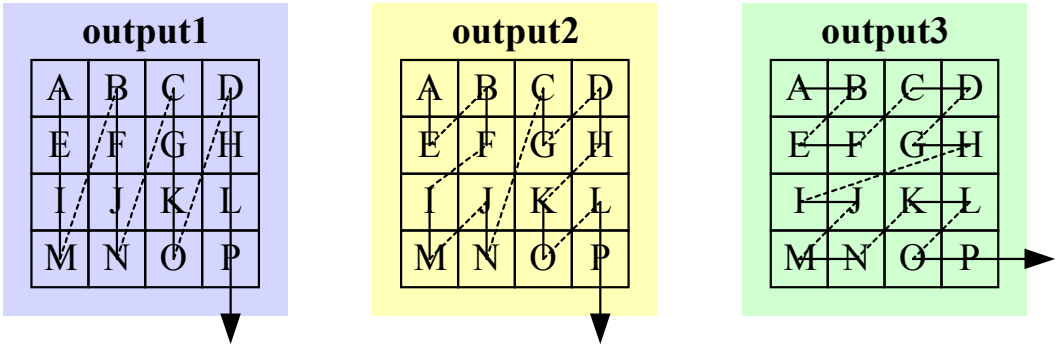
Step 2: forward-backward register allocation

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)	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



Example 5-2 (1/4)

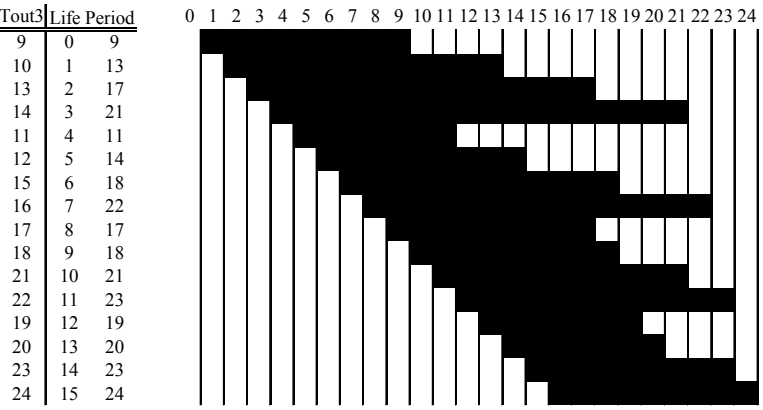


- Sequential execution (on processors)
  - cache effectively explores the temporal locality to reduce the system bus traffic
- Parallel execution (on DSP datapath)
  - SIU minimizes the system bus bandwidth by introducing a small buffer and latency

Example 5-2 (2/4)

- Assume raster-scan input
- Re-transmission of any item is prohibited to minimize bandwidth
- Latency: the most negative difference between T_{in} (data arriving time) and T_{zl} (data consumed time with zero latency)
- Lifetime: birth (T_{in}) till death (the last consumed time i.e. MAX [T_{out}])

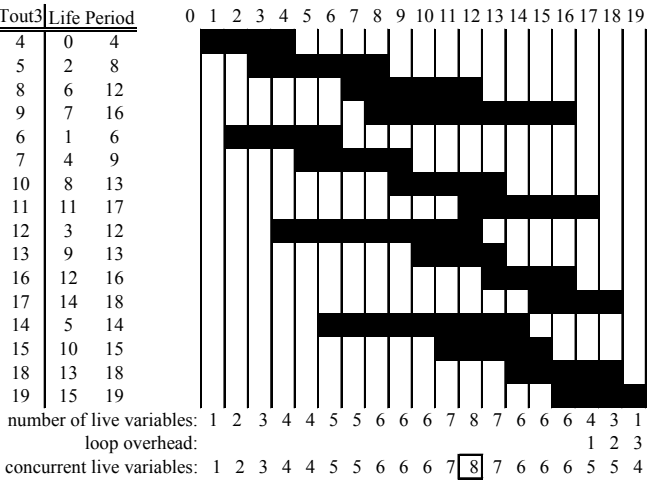
	Tinput	Tzl1	Tdiff1	Tout1	Tzl2	Tdiff2	Tout2	Tzl3	Tdiff3	Tout3	Life Period	
A	0	0	0	9	0	0	9	0	0	9	0	9
B	1	4	3	13	2	1	11	1	0	10	1	13
C	2	8	6	17	8	6	17	4	2	13	2	17
D	3	12	9	21	10	7	19	5	2	14	3	21
E	4	1	-3	10	1	-3	10	2	-2	11	4	11
F	5	5	0	14	3	-2	12	3	-2	12	5	14
G	6	9	3	18	9	3	18	6	0	15	6	18
H	7	13	6	22	11	4	20	7	0	16	7	22
I	8	2	-6	11	4	-4	13	8	0	17	8	17
J	9	6	-3	15	6	-3	15	9	0	18	9	18
K	10	10	0	19	12	2	21	12	2	21	10	21
L	11	14	3	23	14	3	23	13	2	22	11	23
M	12	3	-9	12	5	-7	14	10	-2	19	12	19
N	13	7	-6	16	7	-6	16	11	-2	20	13	20
O	14	11	-3	20	13	-1	22	14	0	23	14	23
P	15	15	0	24	15	0	24	15	0	24	15	24



Example 5-2 (3/4)

- "***Small Min*** [T_{zl}] ***first***" reduces latency first
- When variables with identical Min[T_{zl}] exist, latency does not depend on how these variables are scheduled. "***Small Max*** [T_{zl}] ***first***" schedules late-dying ones last to efficiently decrease their lifetime and hence storage requirement

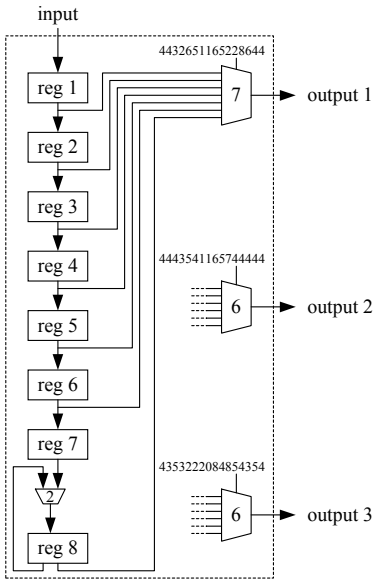
	Tinput	Tzl1	Tdiff1	Tout1	Tzl2	Tdiff2	Tout2	Tzl3	Tdiff3	Tout3	Life Period	
A	0	0	0	4	0	0	4	0	0	4	0	4
B	2	4	2	8	2	0	6	1	-1	5	2	8
C	6	8	2	12	8	2	12	4	-2	8	6	12
D	7	12	5	16	10	3	14	5	-2	9	7	16
E	1	1	0	5	1	0	5	2	1	6	1	6
F	4	5	1	9	3	-1	7	3	-1	7	4	9
G	8	9	1	13	9	1	13	6	-2	10	8	13
H	11	13	2	17	11	0	15	7	-4	11	11	17
I	3	2	-1	6	4	1	8	8	5	12	3	12
J	9	6	-3	10	6	-3	10	9	0	13	9	13
K	12	10	-2	14	12	0	16	12	0	16	12	16
L	14	14	0	18	14	0	18	13	-1	17	14	18
M	5	3	-2	7	5	0	9	10	5	14	5	14
N	10	7	-3	11	7	-3	11	11	1	15	10	15
O	13	11	-2	15	13	0	17	14	1	18	13	18
P	15	15	0	19	15	0	19	15	0	19	15	19



Example 5-2 (4/4)

- Forward-backward register allocation

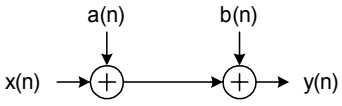
T	input	reg 1	reg 2	reg 3	reg 4	reg 5	reg 6	reg 7	reg 8	output 1	output 2	output 3
0	A											
1	E	A										
2	B	E	A									
3	I	B	E	A								
4	F	I	B	E	A(1,2,3)					A	A	A
5	M	F	I	B(3)	E(1,2)					E	E	B
6	C	M	F	I(1)	B(2)	E(3)				I	B	E
7	D	C	M(1)	F(2,3)	I	B				M	F	F
8	G	D	C(3)	M	F	I(2)	B(1)			B	I	C
9	J	G	D(3)	C	M(2)	F(1)	I			F	M	D
10	N	J(1,2)	G(3)	D	C	M		I		J	J	G
11	H(3)	N(1,2)	J	G	D	C	M		I	N	N	H
12	K	H	N	J	G	D	C(1,2)	M	I(3)	C	C	I
13	O	K	H	N	J(3)	G(1,2)	D		M	G	G	J
14	L	O	K(1)	H	N			D(2)	M(3)	K	D	M
15	P	L	O(1)	K	H(2)	N(3)			D	O	H	N
16		P	L	O	K(2,3)	H			D(1)	D	K	K
17			P	L(3)	O(2)		H(1)			H	O	L
18				P	L(1,2)	O(3)				L	L	O
19					P(1,2,3)					P	P	P



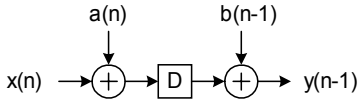
Parallelism

- Granularity
  - job, task, thread, data or bit level
- Data passing
  - by value, address or name
- Concurrency control
  - centralized (programmed)
  - data-driven, data-flow (data availability) & demand-driven
- Communication
  - dedicated network structure
  - shared memory
- Problem size
  - number of modules
  - memory size

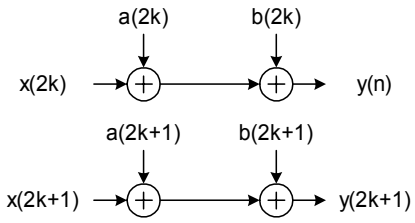
Basic Transformation



- ***Pipelining***
  - reduce the critical path by introducing pipelining registers along the critical path



- ***Parallel processing***
  - increase the sampling rate by replicating hardware to process several inputs in parallel and produce several outputs simultaneously



Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Introduction
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Power Reduction via Architectural Transformation

- Propagation delay

$$T_{\text{pd}} = \frac{C_{\text{charge}} \cdot V_0}{k(V_0 - V_t)^2}$$

  - C_{charge}: the capacitance to be charged or discharged in a single clock cycle (along the critical path)
  - V_0: the supply voltage; V_t: the threshold voltage
  - k: technology parameter
- (Dynamic) power dissipation

$$P = C_{total} \cdot V_0^2 \cdot f$$

  - C_{total}: the total capacitance of the CMOS circuit

Pipelining for Low Power (1/2)

- For an ***M-level*** pipelined architecture, the critical path is reduced to ***1/M*** and the capacitance to be charged or discharged in a single cycle (i.e. C_{charge}) is also reduced to ***1/M***
- If the same clock speed is maintained (i.e. $f = 1/T_{pd}$), only ***1/M*** of the non-pipelined capacitance is required to be charged or discharged, which suggests voltage reduction
- Suppose the voltage can be reduced to $\beta \cdot V_0$, the power consumption of a pipelined architecture becomes

$$P_{pipelined} = C_{total} \cdot (\beta \cdot V_0)^2 \cdot f$$
$$= \beta^2 \cdot P_{non-pipelined}$$

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Introduction
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Pipelining for Low Power (2/2)

- Solving β
  - propagation delay of the original architecture

$$T_{\text{non-pipelined}} = \frac{C_{\text{charge}} \cdot V_0}{k(V_0 - V_t)^2}$$

  - propagation delay of the pipelined architecture

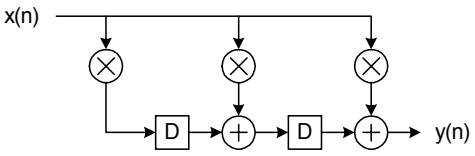
$$T_{\text{pipelined}} = \frac{\frac{C_{\text{charge}}}{M} \cdot \beta \cdot V_0}{k(\beta \cdot V_0 - V_t)^2}$$

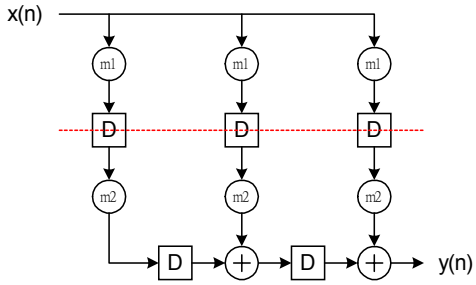
  - setting the above two equations equal, the following quadratic equation can be obtained to solve β

$$M(\beta \cdot V_0 - V_t)^2 = \beta \cdot (V_0 - V_t)^2$$

Example 5-3

- Consider the following two FIR filters.





- Assume
  - the multiplication and the addition take ***10 u.t.*** and ***2 u.t.*** respectively
  - the capacitance of the multiplier is ***5*** times that of an adder
  - In the fine-grain pipelined filter, the multiplier is broken into ***m1***and ***m2***
    - with ***6*** and ***4*** u.t. computation time
    - and ***3*** and ***2*** times capacitance of an adder
  - supply voltage V_0 and threshold voltage V_t are ***5*** and ***0.6*** respectively
- What is the supply voltage of the pipelined architecture if the clock periods are identical?
- What is the relative power consumption?

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Introduction
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Solution

- Calculate C_{charge}: let C_M, C_A, C_{m1} and C_{m2} be the capacitance of the multiplier, the adder, and the two fine-grain pipelining parts of the pipelined FIR filter respectively
  - Non-pipelined: $C_{charge} = C_M + C_A = 6C_A$
  - Pipelined: $C_{charge} = C_{m1} = C_{m2} + C_A = 3C_A$

  Notice the pipelining level $M = 2$ and the charging capacitance of the pipelined filter is one-half of that of the original one
- Solve β

$$2 \cdot (\beta \cdot 5 - 0.6)^2 = \beta \cdot (5 - 0.6)^2$$
$$50 \cdot \beta^2 - 31.36 \cdot \beta + 0.72 = 0$$
$$\beta = 0.6033 \text{ or } 0.0239 \text{ (invalid, } \because \beta \cdot V_0 = 0.1195 < V_t)$$

- The supply voltage of the pipelined filter is

$$V_{pipelined} = \beta \cdot V_0 = 3.0165$$

- Power consumption ratio is $\beta^2 = 36.4\%$

Parallel Processing for Low Power (1/2)

- For an ***L-parallel*** architecture, the charge capacitance remains the same, but the total capacitance (i.e. C_{total}) is increased ***L*** times
- The clock speed of the ***L-parallel*** architecture is reduced to ***1/L*** (i.e. $f = 1/LT_{pd}$) to maintain the same sample rate, which means the C_{charge} is charged or discharged ***L*** times longer.
- The supply voltage can be reduced to $\beta \cdot V_0$, since more time is allowed to charged or discharged the same capacitance. Thus the power consumption of the parallel architecture becomes

$$P_{parallel} = (L \cdot C_{total}) \cdot (\beta \cdot V_0)^2 \cdot \frac{f}{L}$$
$$= \beta^2 \cdot P_{non-parallel}$$

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Introduction
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Parallel Processing for Low Power (2/2)

- Solving β
  - propagation delay of the original architecture

$$T_{\text{non-parallel}} = \frac{C_{\text{charge}} \cdot V_0}{k(V_0 - V_t)^2}$$

  - propagation delay of the pipelined architecture

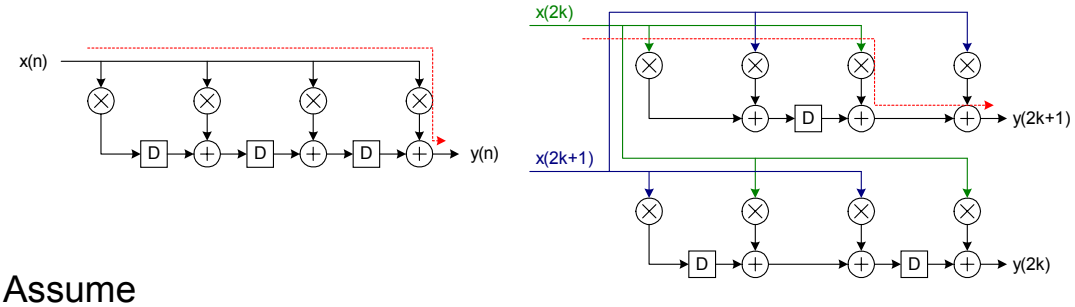
$$L \cdot T_{\text{parallel}} = \frac{C_{\text{charge}} \cdot \beta \cdot V_0}{k(\beta \cdot V_0 - V_t)^2}$$

  - setting these two propagation delays equal, the following quadratic equation can be obtained to solve β

$$L(\beta \cdot V_0 - V_t)^2 = \beta \cdot (V_0 - V_t)^2$$

Example 5-4

- Consider the following two FIR filters, with critical paths denoted in dash lines respectively



- Assume
  - the multiplication and the addition take ***8 u.t.*** and ***1 u.t.*** respectively
  - the capacitance of the multiplier is ***8*** times that of an adder
  - both architectures operate at the sample period of ***9 u.t.***
  - supply voltage V_0 and threshold voltage V_t are ***3.3*** and ***0.45*** respectively
- What is the supply voltage of the parallel architecture?
- What is the relative power consumption?

Solution

- Calculate C_{charge}: let C_M and C_A be the capacitance of the multiplier and the adder respectively
  - Non-parallel: $C_{charge} = C_M + C_A = 9C_A$
  - Parallel: $C_M + 2C_A = 10C_A$

  Notice the charging capacitance of the two architectures are not equal, and we cannot use the equation directly
- Solve β

$$T_{non-parallel} = \frac{9C_A \cdot V_0}{k(V_0 - V_t)^2} \text{ and } T_{parallel} = \frac{10C_A \cdot \beta \cdot V_0}{k(\beta \cdot V_0 - V_t)^2}$$

$$\because T_{parallel} = 2 \cdot T_{non-parallel}$$

$$5 \cdot \beta \cdot (V_0 - V_t)^2 = 9 \cdot (\beta \cdot V_0 - V_t)^2$$

$$98.01 \cdot \beta^2 - 67.3425 \cdot \beta + 1.8225 = 0$$

$$\beta = 0.6589 \text{ or } 0.0282 \text{ (invalid, } \because \beta \cdot V_0 = 0.09306 < V_t)$$

- The supply voltage of the parallel filter is $V_{pipelined} = \beta \cdot V_0 = 2.17437$
- Power consumption ratio is $\beta^2 = 43.41\%$

Outline

- Introduction
- *Algorithm Representation and Iteration Bound*
- Retiming
- Unfolding and Folding Transformation
- Bit-serial Architectures
- Case Study – DWT Processor

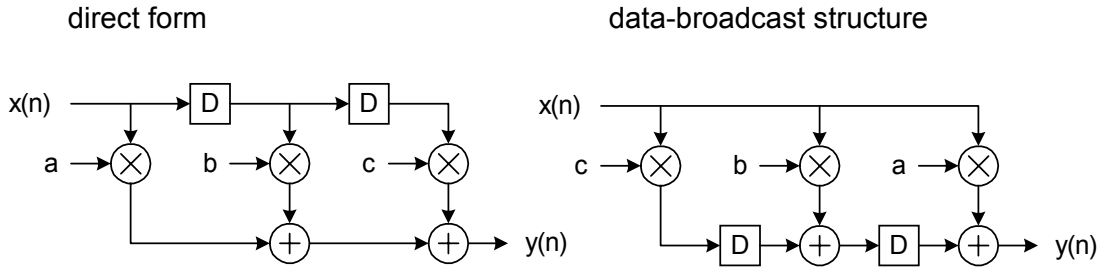
Representations of DSP Algorithms

- Mathematical formulation
  e.g. y(n)=ax(n)+bx(n-1)+cx(n-2), for n=1 to infinity
- For architectural design, the mathematical formulae can be translated into
  - behavioral description
    - **applicative languages**: set of equations (e.g. Silage)
    - **prescriptive languages**: sequence of actions (e.g. C)
    - **descrptive languages**: hardware description (e.g. Verilog)
  - graphical representation
    efficient for investigating and analyzing dataflow properties and exploring the inherent parallelism
    - **block diagram**
    - **signal flow graph (SFG)**
    - **data flow graph (DFG)**
    - **dependence graph (DG)**

Block Diagram

- A block diagram consists of functional blocks connected with directed edges, which can be constructed at different levels of abstraction
- A system can be represented using various block diagrams

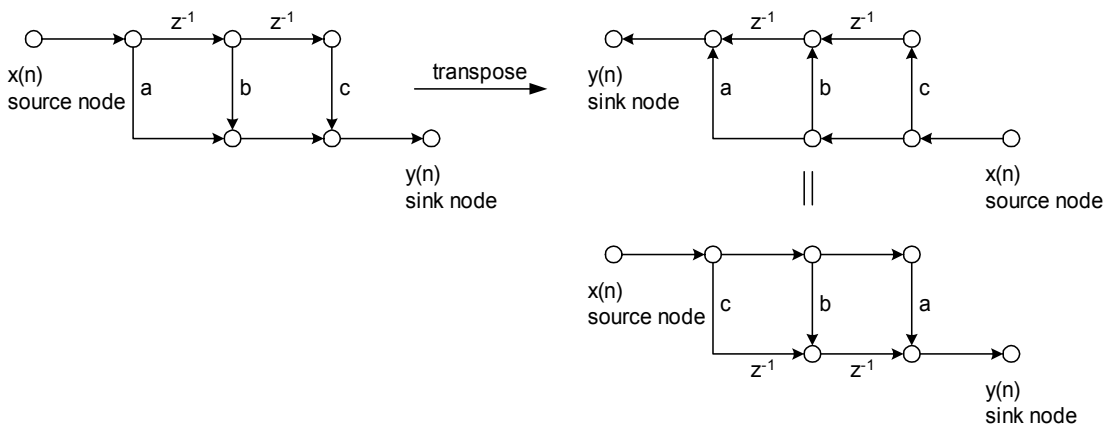
e.g. 3-tap FIR filter



Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Algorithm Representation
Chein-Wei Jen cwjen@ee.nctu.edu.tw

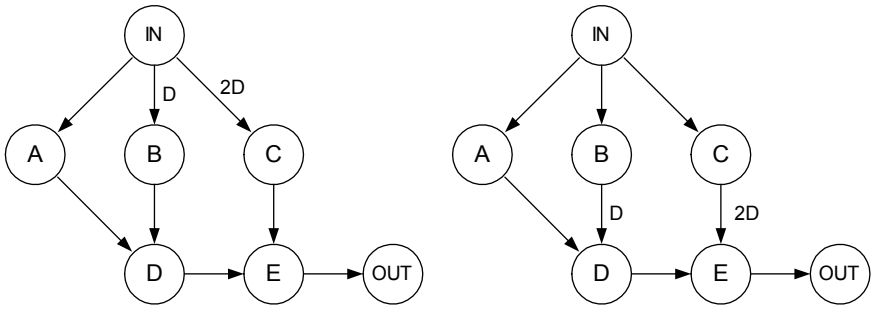
Signal Flow Graph (SFG)

- A signal flow graph is a collection of nodes and directed edges
  - the nodes represent computations or tasks
  - a directed edge (j,k) is a branch from node j to node k, which denotes a linear transformation



Data Flow Graph (DFG)

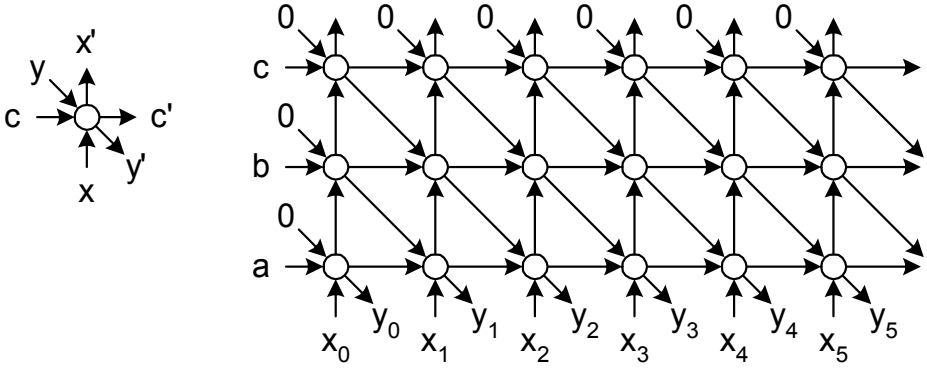
- The nodes represent computations, functions, or tasks while the directed edges represent data paths (i.e. communications between nodes) with a non-negative number of delays
- DFG captures the data-driven properties of the DSP algorithm - any node can start the computation (i.e. node fire) whenever all its input data are available



- Intra-iteration precedence constraint: edges without delay
  Inter-iteration precedence constraint: one or more delays

Dependency Graph (DG)

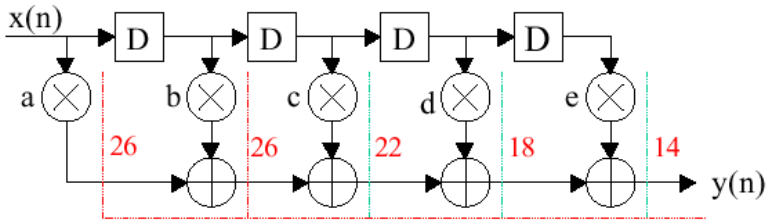
- A directed graph that shows the dependence of the computation in the DSP algorithm
  - node: computation (used only once)
  - edge: precedence constraint (no cycle in a DG)
- DG exhibits the inherent parallelism and dataflow constraints with least structural bias



Critical Path

- 4 possible paths
  - input node to delay element
  - input node to output node
  - delay element to delay element
  - delay element to output
- ***Critical path***: the path with the longest computation time among all paths without delay elements
- Clock period is bounded by the computation time of the critical path

  Example: assume T_A = 4ns and T_M = 10ns



critical paths = 26ns (red lines)

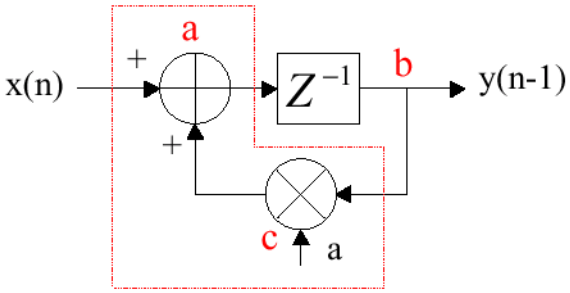
Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Algorithm Representation
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Iteration Period

- The time required to execute one iteration of the algorithm

$$y(n) = a \cdot y(n-1) + x(n)$$

$$i.e. \qquad H(z) = \frac{1}{1 - a \cdot z^{-1}}$$



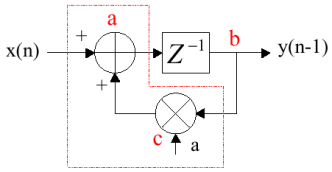
- Assume T_A and T_M represent the execution times of the adder and the multiplier respectively. The iteration period is T_A+T_M. The sample period T_S must satisfy $T_S \geq T_A + T_M$

- ***Iteration rate***: the number iterations executed per second
- ***Sample rate*** (***throughput***): the number of samples processed per second

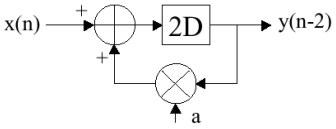
Loop Bound

- Loop: a directed path that begins and ends at the same node
- ***Loop bound*** of the loop j: T_j / W_j, where T_j is the loop computation time and W_j is the number of delays in the loop

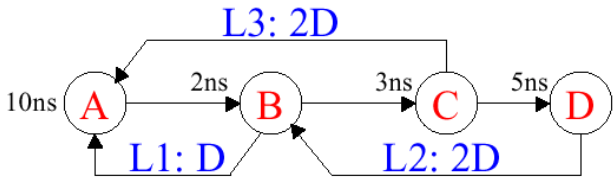
  Examples:
  - assume T_A+T_M =10ns,
    a, b, c, a is a loop with loop bound = 10ns



  - y(n)=ay(n-2)+x(n)
    assume T_A+T_M =10ns, the loop bound = 5ns



  -



$$T_{L1} = (10+2)/1 = 12ns$$
$$T_{L2} = (2+3+5)/2 = 5ns$$
$$T_{L3} = (10+2+3)/2 = 7.5ns$$

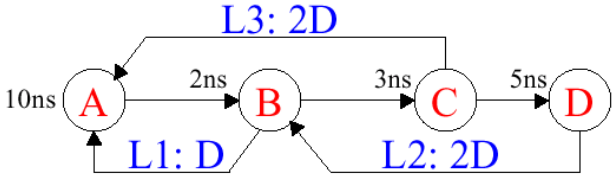
Iteration Bound

- ***Critical loop***: the loop with the maximum loop bound
- ***Iteration bound***: the loop bound of the critical loop

$$T_\infty = \max_{j \in L} \left\{ \frac{T_j}{W_j} \right\}$$

where L is the set of loops in the DSP system,
T_j is the computation time of the loop j and
W_j is the number of delays in the loop j

Example:



$$T_{L1} = (10+2)/1 = 12ns$$
$$T_{L2} = (2+3+5)/2 = 5ns$$
$$T_{L3} = (10+2+3)/2 = 7.5ns$$
$$T_\infty = \max_{l \in L}\{12,\ 5,\ 7.5\}$$

Longest Path Matrix Algorithm

- Let d be the number of delays in the DFG. A series of matrices $L^{(m)}$, where $m = 1, 2, \ldots, d$, are constructed such that $l_{i,j}^{(m)}$ is the longest computation time of all paths from delay element d_i to d_j that passes through exactly $(m\text{-}1)$ delays. If such a path does not exist $l_{i,j}^{(m)} = -1$
- $L^{(1)}$ is usually obtained from the DFG
  - the longest path between any two delay nodes
- Higher order matrices are computed recursively as

$$l_{i,j}^{(m)} = \max[-1, l_{i,k}^{(1)} + l_{k,j}^{(m)}]$$

for k belongs to K, where K is the set of integers k in the interval $[1,d]$ such that neither $l_{i,k}^{(1)} = -1$ nor $l_{k,j}^{(m)} = -1$

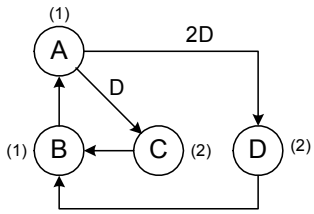
- The iteration bound is given by

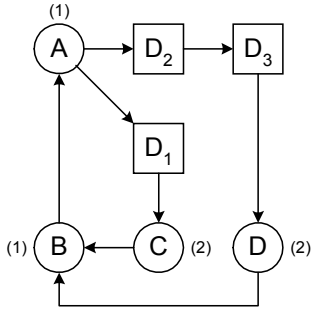
$$T_\infty = \max\left\{\frac{l_{i,i}^{(m)}}{m}\right\} \text{ for } i, m \in \{1,2,\ldots,d\}$$

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Algorithm Representation
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Example 5-5

- Perform the LPM algorithm on the following DFG to calculate the iteration bound





$$L^{(1)} = \begin{bmatrix} 4 & 4 & -1 \\ -1 & -1 & 0 \\ 4 & 4 & -1 \end{bmatrix}$$

$$L^{(2)} = \begin{bmatrix} 8 & 8 & 4 \\ 4 & 4 & -1 \\ 8 & 8 & 4 \end{bmatrix}$$

$$L^{(3)} = \begin{bmatrix} 12 & 12 & 8 \\ 8 & 8 & 4 \\ 12 & 12 & 8 \end{bmatrix}$$

$$T_\infty = \max\left\{4, \frac{8}{2}, \frac{4}{2}, \frac{12}{3}, \frac{8}{3}\right\} = 4$$

Outline

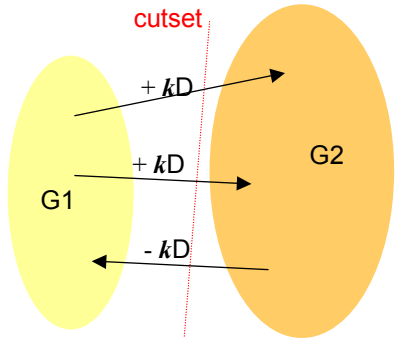
- Introduction
- Algorithm Representation and Iteration Bound
- ***Retiming***
- Unfolding and Folding Transformation
- Bit-serial Architectures
- Case Study – DWT Processor

Cutset Pipelining

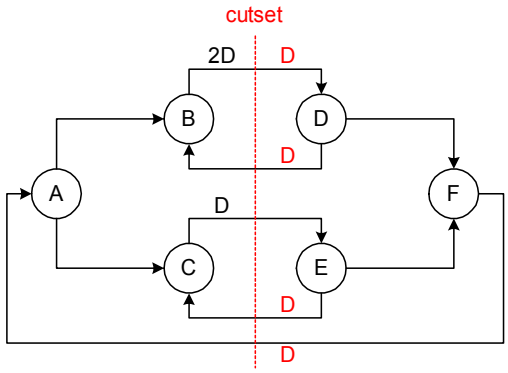
- ***Cutset***: a set of edges of a graph such that if these edges are removed from the graph, the graph becomes ***disjoint***
- ***Feed-forward cutset***: the data move in the forward direction on all edges of the cutset
- In a synchronous DFG (SDFG),
  - communications occurs on the edges only
  - data flow through the functional units directly and the operations are synchronized with the arranged ***data arrival times*** (via delay insertion)

    c.f. in a RISC-type computing machine, data are processed only through the register file, and operations are synchronized by instruction sequencing
- ***Cutset pipelining***: because the data movement between the two ***disjoint*** sub-graphs only occurs on the ***feed-forward cutset***, delaying or advancing the data movement along all edges on the cutset by the same amount of time do not change the architecture's behavior

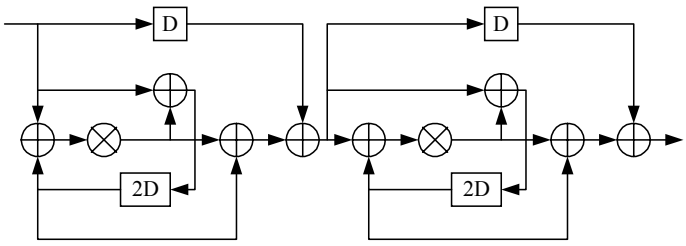
Cutset Retiming



Example



Example 5-6

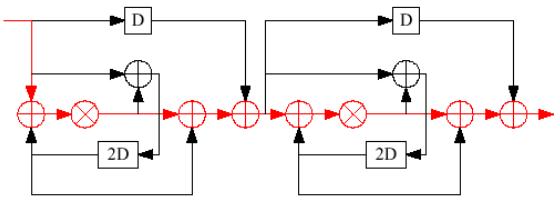


- Consider the wave digital filter shown above. Assume that each multiply operation requires 20 nsec and each add operation require 8 nsec
  - Calculate the iteration bound by inspection
  - What is the critical path
  - Retime this filter to achieve a critical path equal to its iteration bound

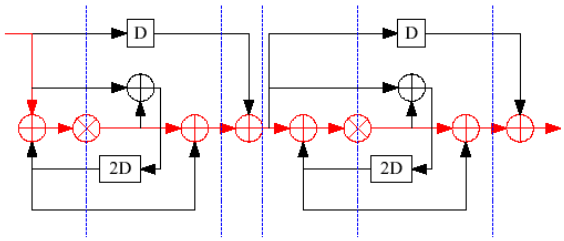
Solution

(a) $T_\infty = \frac{T_M + 2T_A}{2} = 18$

(b) $6T_A + 2T_M = 88$



(c) Because $T_M > T_\infty$, fine-grain pipelining is required and the multiplication units are divided into two sub-parts with 10-unit computation time each. The cutset for retiming is shown as following.



Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Retiming
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Node Retiming

- The cutset can be selected such that one of the disjoint sub-graphs contains only one node
- ***Retiming value r(V)***: defined as the number of delays moved from each output edge of the node V to each of its input edges
- ***Feasibility constraints***: for each directed edge UV of the retimed SDFG, the number of delay elements must be non-negative

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

where $w_r(e)$ and $w(e)$ are the number of delay elements on the edge $\overrightarrow{UV}$ after and before retiming respectively

- For any SDFG, there is a retiming design space defined by the system of inequalities (each edge contributes an inequality)

Properties of Retiming

- ***Property 1***

  The weight of a retimed path p = $V_0, V_1, \ldots, V_k$ is given by

$$w_r(p) = w(p) + r(V_k) - r(V_0)$$

- ***Property 2***

  Retiming does not change the number of delays in a cycle
- ***Property 3***

  Retiming does not alter the iteration bound in a DFG as the number of delays in a cycle does not change
- ***Property 4***

  Adding a constant value j to the retiming value of each node does not change the mapping from G to G_r

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Retiming
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Retiming for Minimum Clock Period (1/3)

- $\Phi(G)$: the minimum feasible clock period is defined as

$$\Phi(G) = \max\{t(p) : w(p) = 0\}$$

- ***In the retiming design space (defined by the system of inequalities), find a retimed SDFG with the minimum*** $\Phi(G)$
- Definition:
  - ***W(U,V)***: the minimum number of registers on any path from node U to node V

$$W(U,V) = \min\{w(p) : U \xrightarrow{p} V\}$$

  - ***D(U,V)***: the maximum computation time among all paths from U to V with weight W(U,V)

$$D(U,V) = \max\{t(p) : U \xrightarrow{p} V \text{ and } w(p) = W(U,V)\}$$

Retiming for Minimum Clock Period (2/3)

Step 1: Compute W(U,V) and D(U,V)

- Let M = t_{max}n, where t_{max} is the maximum computation time of the nodes in G and n is the number of nodes in G
- Form a new graph G', which is the same as G except the edge weights are replaced by w'(e) = Mw(e) – t(u) for all edges from U to V
- Solve the all-pair shortest path problem on G' (using ***Floyd-Warshall algorithm***). Let S'_{UV} be the shortest path from U to V
- If U = V,
  - W(U,V) = 0 and D(U,V) = t(U)

  else,
  - W(U,V) = ceiling (S'_{UV}/M) and D(U,V) = MW(U,V) - S'_{UV} + t(V)

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Chein-Wei Jen cwjen@ee.nctu.edu.tw

All-Pair Shortest-Path Algorithm

- Floyd-Warshall Algorithm

```
for V=1 to n
        for U=1 to n
            r^(1)(U,V) = w(U --e--> V)
for k=1 to n
        for V=1 to n
                for U=1 to n
                    r^(k+1)(U,V) = min{r^(k)(U,V), r^(k)(U,K) + r^(k)(k,V)}
for k=1 to n
        for U=1 to n
        if  r^(k)(U,U) < 0
        return FALSE and exit
return True and exit
```

Retiming for Minimum Clock Period (3/3)

Step 2: Given a clock period c, find a feasible solution such that

$$\Phi(G) \le c$$

- The retimed design space is now defined by
  - feasibility constraints

$$r(U) - r(V) \le w(e)$$

  - critical path constraints

$$r(U) - r(V) \le W(U,V) - 1$$
$$\text{for all nodes U and V in G, such that } D(U,V) > c$$

i.e.

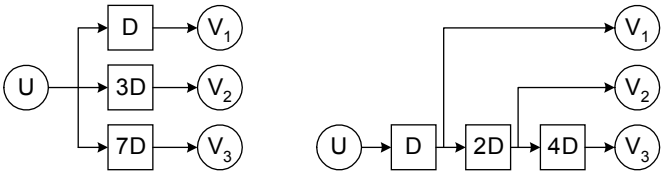
$$\text{if } D(U,V) > c, r(U) - r(V) \le W(U,V) - 1$$
$$\Leftrightarrow \text{if } W(U,V) + r(V) - r(U) < 1, D(U,V) \le c$$

- If the system of inequalities is not ϕ, try a smaller c

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Retiming
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Retiming for Minimum Registers

- ***Find a feasible solution with minimum number of registers in the design space constrained by feasibility and critical-path constraints (the refined retimed space with minimum clock period)***
- Multiple fan-out problem



- Modeled as an ***integer linear programming*** (ILP) problem

  Minimize $\sum q(U)$, while satisfying
  - longest fanout constraints $w(e) + r(V) - r(U) \le q(U)$
  - feasibility constraints $r(U) - r(V) \le w(e)$
  - critical path constraints $r(U) - r(V) \le W(U,V) - 1$

    $\text{for all nodes U and V in G, such that } D(U,V) > c$

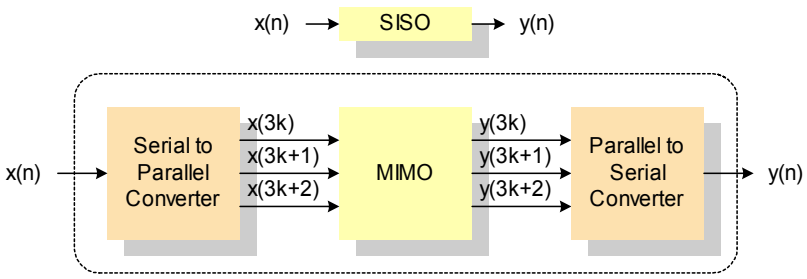
Note solving the ILP model is NP-hard, and a gadget model can be found in the literature.

Outline

- Introduction
- Algorithm Representation and Iteration Bound
- Retiming
- ***Unfolding and Folding Transformation***
- Bit-serial Architectures
- Case Study – DWT Processor

Parallel Processing

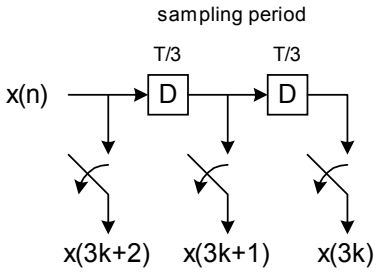
- Block processing
  - the number of inputs processed in a clock cycle is referred to as the ***block size***



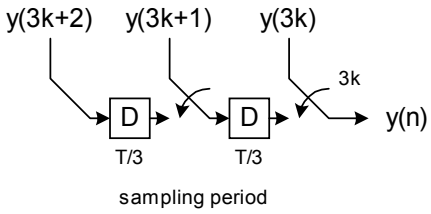
  - at the ***k-th*** clock cycle, three inputs x(3k), x(3k+1), and x(3k+2) are processed simultaneously to generate y(3k), y(3k+1), and y(3k+2)

Conversion between Serial & Parallel

- Serial to parallel converter



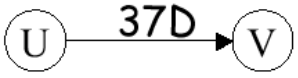
- Parallel to serial converter

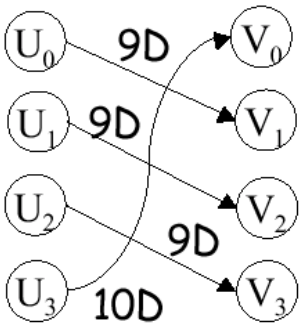


Unfolding Transformation

- For each node U in the original DFG, draw J node U_0, U_1,…, U_{J-1}
- For each edge UV with w delays in the original DFG, draw the J edges $U_iV_{(i+w)\%J}$ with floor[(i+w)/J] delays for i=0,1,…, J-1

  Example





- Unfolding of an edge with w delays in the original DFG produces J-w edges with no delays and w edges with 1delay in J-unfolded DFG for w < J
- Unfolding preserves precedence constraints of a DSP algorithm

Preservation of Precedence Constraints

Original DFG

$$U \xrightarrow{w} V$$

J-unfolded DFG

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

Iteration	U	V
J-unfolded DFG	k	$k+\left\lfloor \frac{i+w}{J} \right\rfloor$
Original DFG	$Jk+i$	$J\left(k+\left\lfloor \frac{i+w}{J} \right\rfloor\right)+(i+w)\%J$
Difference	$J\left\lfloor \frac{i+w}{J} \right\rfloor+(i+w)\%J-i=(i+w)-i=w$	

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Unfolding & Folding
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Delay Preservation

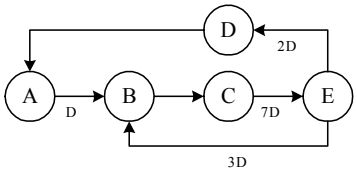
- Unfolding preserves the number of delays in a DFG
- Let $w = m \times J + n$, where $m, n \in \{0 \cup \aleph\}$ $0 \le n \le (J-1)$

$$\left\lfloor \frac{w}{J} \right\rfloor = m$$
$$\vdots$$
$$\left\lfloor \frac{w+(J-n-1)}{J} \right\rfloor = \left\lfloor \frac{m \times J + n + (J-n-1)}{J} \right\rfloor = \left\lfloor \frac{m \times J + (J-1)}{J} \right\rfloor = m$$
$$\left\lfloor \frac{w+(J-n)}{J} \right\rfloor = \left\lfloor \frac{m \times J + n + (J-n)}{J} \right\rfloor = \left\lfloor \frac{(m+1) \times J}{J} \right\rfloor = m+1$$
$$\vdots$$
$$\left\lfloor \frac{w+J-1}{J} \right\rfloor = m+1$$

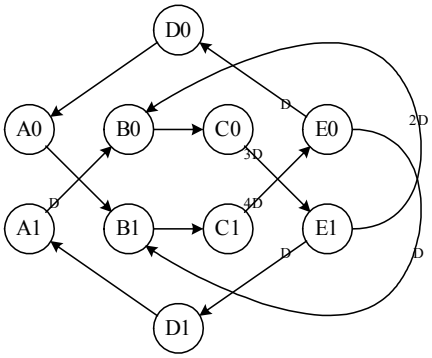
$$\therefore \left\lfloor \frac{w}{J} \right\rfloor + \cdots + \left\lfloor \frac{w+(J-n-1)}{J} \right\rfloor + \left\lfloor \frac{w+(J-n)}{J} \right\rfloor + \left\lfloor \frac{w+J-1}{J} \right\rfloor$$
$$= m \times (J-n) + (m+1) \times n$$
$$= m \times J + n$$
$$= w$$

Example 5-7

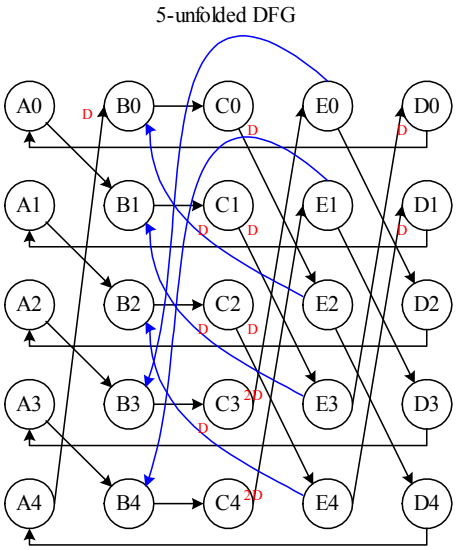
- Unfold the following DFG using folding factor 2 and 5



2-unfolded DFG



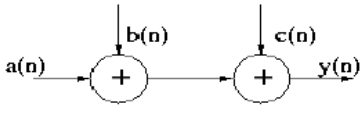
5-unfolded DFG

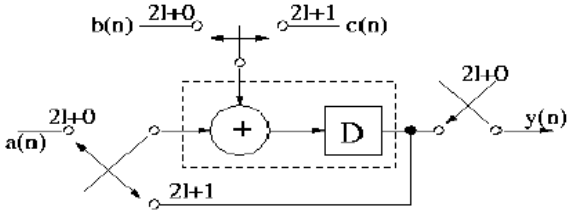


Folding Transformation

- Folding transformation time-multiplexes several algorithm operations (e.g. multiply or add) onto single functional unit to reduce silicon area

  Example



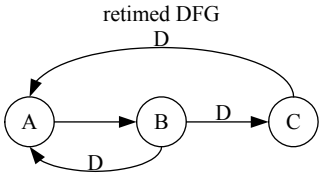


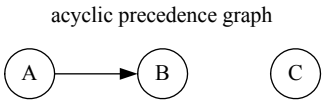
- Procedures
  - **operation scheduling & allocation**
  - **delay calculation**
  - **causality transformation**
  - **SIU generation**

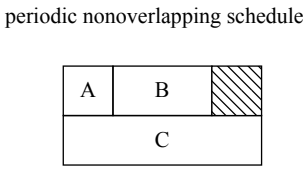
Acyclic Precedence Graph

A graph obtained from a DFG by removing all edges that contain delay

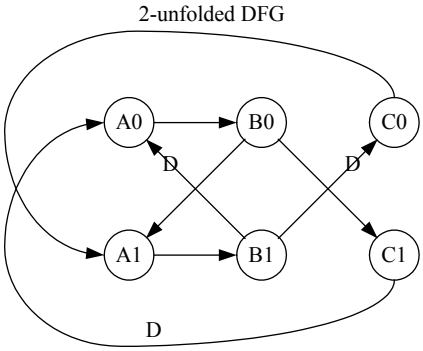
- Intra-iteration scheduling
  - obviously, the schedule can be improved by retiming

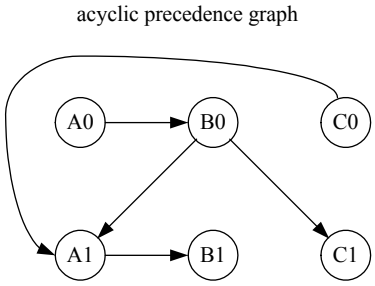


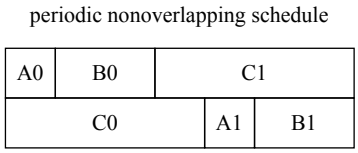




- Scheduling for multiple iterations (by unfolding)





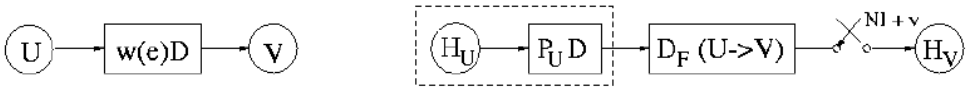


Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Unfolding & Folding
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Basic Scheduling Techniques

- As soon as possible (ASAP)
- As late as possible (ALAP)
- Force-directed scheduling (time-constrained)
- List scheduling (resource constrained)

Delay Calculation



- Nl+u and Nl+v are respectively the time units at which l-th iteration 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 <= u,v < N
- N is the folding factor (i.e. the number of operations folded to a single functional unit
- H_u and H_v are functional units that execute u and v respectively
- H_u is pipelined into P_u stages with output available at $Nl+u+P_u$
- Edge UV has w(e) delays; 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:

  $D_F(UV) = [N(l+w(e))+v]–[Nl+P_u+u]$

  $D_F(UV) = Nw(e)-P_u+v–u$

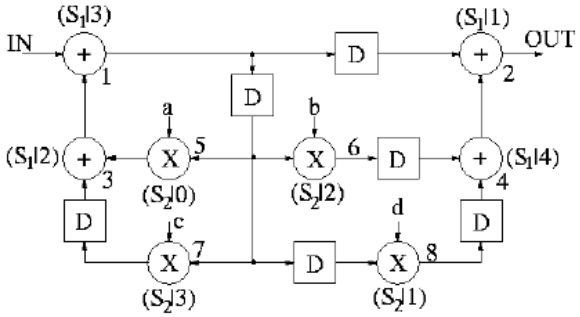
Causality Transformation

- A folded architecture is realizable if and only if all delays $D_F(UV)$ are non-negative
- Retiming for folding
  - All D'_F (UV) of the retimed graph G' are non-negative
  - $N\ w_r(e)–P_U+v–u >= 0$ … where $w_r(e) = w(e)+r(V)-r(U)$

    $N(w(e)+r(V)–r(U))-P_U+v–u >= 0$

    $r(U)–r(V) <= D_F\ (UV) / N <= \text{floor}\ [D_F\ (UV) / N]$

Example 5-8



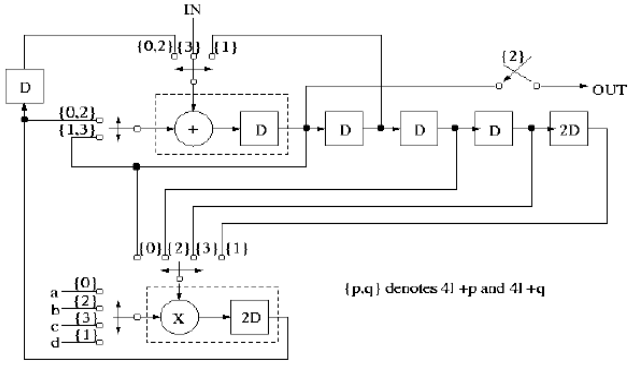
- Fold the above biquad filter by N=4
- Assume the adder and multiplier take 1 and 2 computation time with 1 and 2 stages of pipelining respectively (i.e. P_A=1, P_M=2)
- The folding set is S_A={4,2,3,1} and S_M={5,8,6,7}

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Unfolding & Folding
Chein-Wei Jen cwjen@ee.nctu.edu.tw

Solution (1/2)

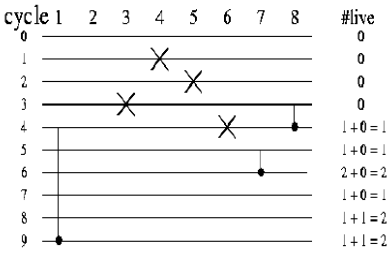
- Folding equations

$D_F(1\rightarrow2) = 4(1) - 1 + 1 - 3 = 1$
$D_F(1\rightarrow5) = 4(1) - 1 + 0 - 3 = 0$
$D_F(1\rightarrow6) = 4(1) - 1 + 2 - 3 = 2$
$D_F(1\rightarrow7) = 4(1) - 1 + 3 - 3 = 3$
$D_F(1\rightarrow8) = 4(2) - 1 + 1 - 3 = 5$
$D_F(3\rightarrow1) = 4(0) - 1 + 3 - 2 = 0$
$D_F(4\rightarrow2) = 4(0) - 1 + 1 - 0 = 0$
$D_F(5\rightarrow3) = 4(0) - 2 + 2 - 0 = 0$
$D_F(6\rightarrow4) = 4(1) - 2 + 0 - 2 = 0$
$D_F(7\rightarrow3) = 4(1) - 2 + 2 - 3 = 1$
$D_F(8\rightarrow4) = 4(1) - 2 + 0 - 1 = 1$

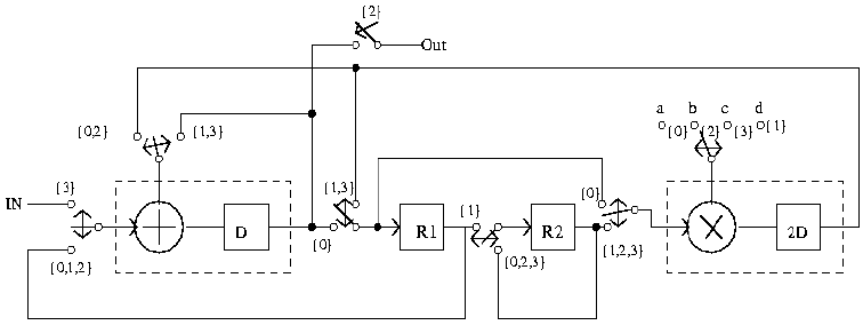


Solution (2/2)

- Folded architecture with minimum registers



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_7	n_8	n_7
7			n_7	
8			n_1	
9			n_1	n_1



Outline

- Introduction
- Algorithm Representation and Iteration Bound
- Retiming
- Unfolding and Folding Transformation
- ***Bit-serial Architectures***
- Case Study – DWT Processor

Bit-level Multiplication

- Notation

  A W-bit fixed point two's complement number A is represented as :

  $$A=a_{w-1}.a_{w-2}...a_1.a_0$$

  where the bits ai, $0 \le i \le W-1$, are either 0 or 1, and the msb is the sign bit.

  The value of this number is in the range of $[-1, 1 - 2^{-W+1}]$ and is given by :

  $$A = - a_{w-1} + \Sigma\, a_{w-1-i}2^{-i}$$

  For bit-serial implementations, constant word length multipliers are considered. For a W×W bit multiplication the W most-significant bits of the (2W-1)-bit product are retained.

- Parallel Multipliers :

  $$A = a_{w-1}.a_{w-2}...a_1.a_0 = -a_{w-1} + \sum_{i=1}^{W-1} a_{w-1-i}2^{-i}$$

  $$B = b_{w-1}.b_{w-2}...b_1.b_0 = -b_{w-1} + \sum_{i=1}^{W-1} b_{w-1-i}2^{-i}$$

  Their product is given by :

  $$P = -p_{2W-2} + \sum_{i=1}^{2W-2} p_{2W-2-i}2^{-i}$$

Tabular form

				a_3	a_2	a_1	a_0
				b_3	b_2	b_1	b_0
				$-a_3b_0$	a_2b_0	a_1b_0	a_0b_0
			$-a_3b_1$	a_2b_1	a_1b_1	a_0b_1	
		$-a_3b_2$	a_2b_2	a_1b_2	a_0b_2		
	$-a_3b_3$	a_2b_3	a_1b_3	a_0b_3			
p_6	p_5	p_4	p_3	p_2	p_1	p_0	

Baugh-Wooley Multiplier

				a_3	a_2	a_1	a_0
				b_3	b_2	b_1	b_0
				$\overline{a_3b_0}$	a_2b_0	a_1b_0	a_0b_0
			$\overline{a_3b_1}$	a_2b_1	a_1b_1	a_0b_1	
		$\overline{a_3b_2}$	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	$\overline{a_2b_3}$	$\overline{a_1b_3}$	$\overline{a_0b_3}$			
x_3	x_2	x_1	x_0				

Canonical Signed Digit (CSD)

- Representation of a signed binary that contains the fewest number of ones (c.f. Booth recoding)
- A = $a'_{W-1}.\, a'_{W-2}...\, a'_1.\, a'_0$ = 2's complement number
  Its CSD representation is $a_{W-1}.\, a_{W-2}...\, a_1.\, a_0$
- Algorithm for 2's complement to CSD conversion

$a'_{-1} = 0;$
$\gamma_{-1} = 0;$
$a'_W = a'_{W-1};$
for (i = 0 to W-1)
{
$\theta_i = a'_i \oplus a'_{i-1};$
$\gamma_i = \overline{\gamma_{i-1}}\theta_i;$
$a_i = (1 - 2a'_{i+1})\gamma_i;$
}

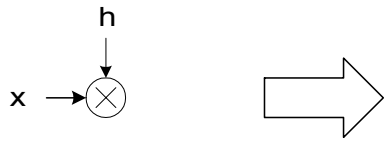
Example 1.01110011

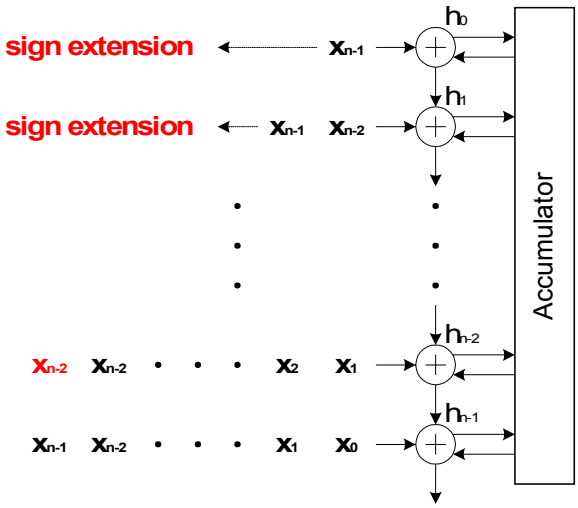
i	W	W-1							0	-1
a'_i	1	1	0	1	1	1	0	0	1	1	
θ_i		1	1	0	0	1	0	1	0	1	
γ_i		0	1	0	0	1	0	1	0	1	
$1 - 2a'_{i+1}$		-1	-1	1	-1	-1	-1	1	1	-1	
a_i		0	-1	0	0	-1	0	1	0	-1	

Bit-Serialization of SDFG (1/5)

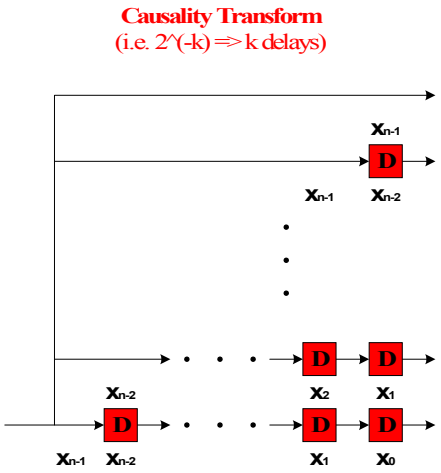
Serialization of Functional Units

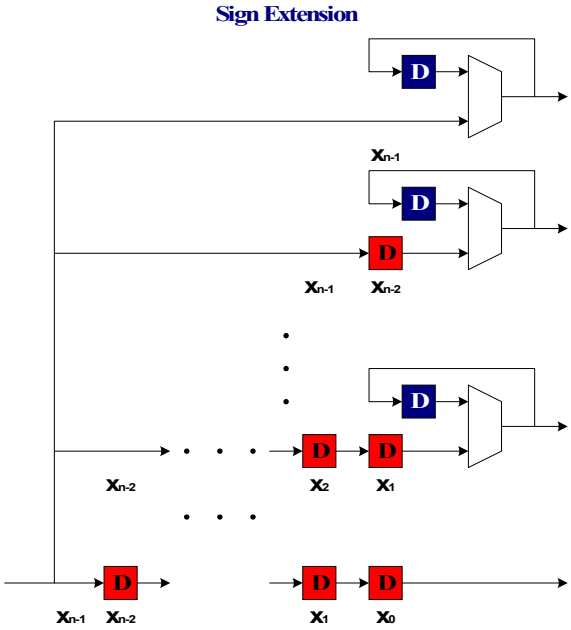
e.g. an N-bit multiplier





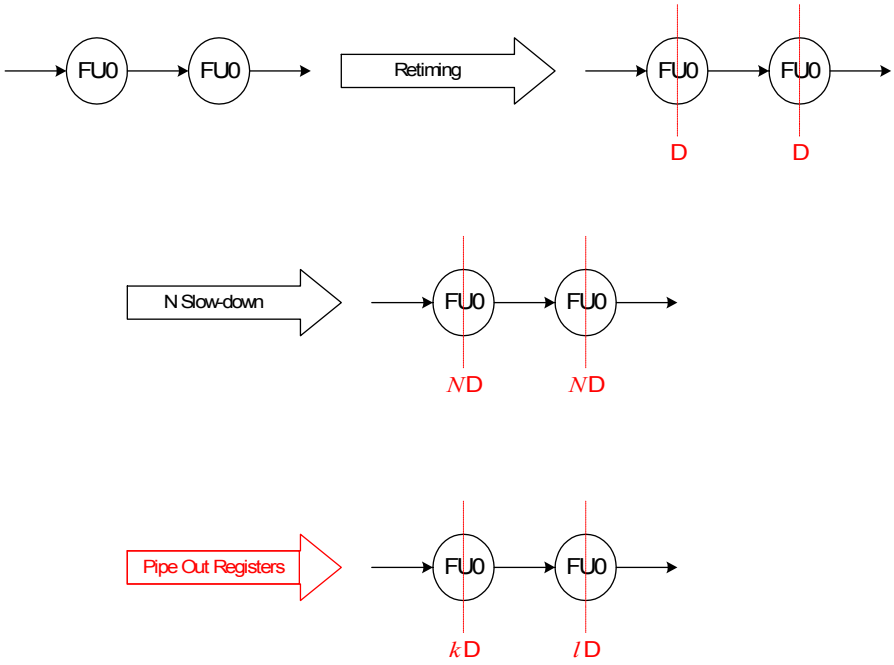
Bit-Serialization of SDFG (2/5)





Bit-Serialization of SDFG (3/5)

Serialization of Acyclic SDFG



Chein-Wei Jen cwjen@ee.nctu.edu.tw

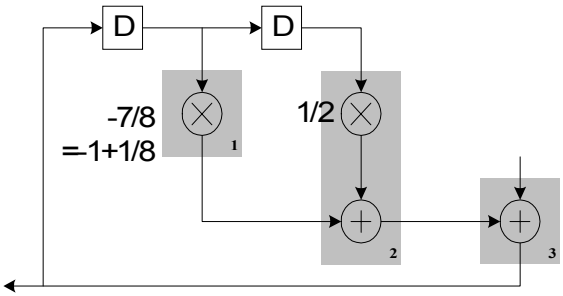
Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Bit-Serial Architectures

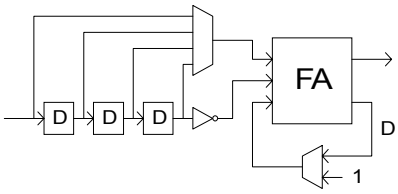
Bit-Serialization of SDFG (4/5)

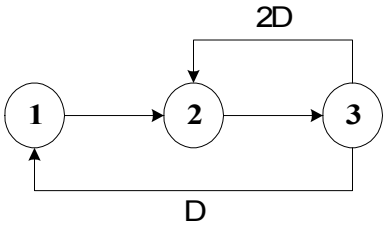
- Serialization of Cyclic SDFG
  - Find a valid retiming set such that ***each FU to be serialized has a delay element***. If no valid retiming set exists, ***slow down the SDFG by J to release the constraints***, and keep trace of the number of delay elements in each loop (L).
  - ***N*** slow-down the SDFG for bit-serialization.
  - For constant multiplications, pipe out redundant delay elements (constant coefficient optimization).
  - If the SDFG was ***slowed down by J in Step 1***, try to cancel it completely or partially while the number of synchronization delay elements of each loop is larger than multiple of ***N***L.

Chein-Wei Jen cwjen@ee.nctu.edu.tw

Example 5-9 (1/3)



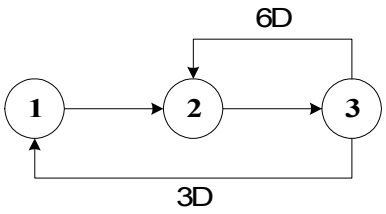


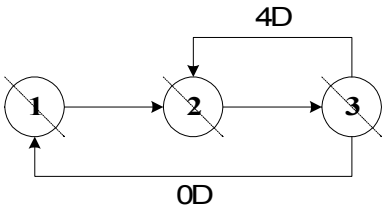


		7	**6**	**5**	**4**	**3**	2	1	0	IN
	7	6	5	4	3	2	1	0		
7	6	5	4	3	2	1	0			
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**			

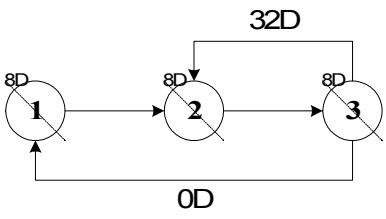
Example 5-9 (2/3)

Step 1: 3-slow down & retime the SDFG

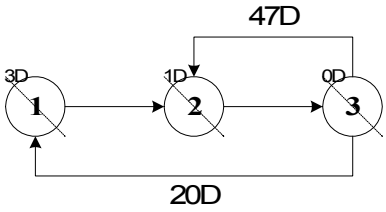




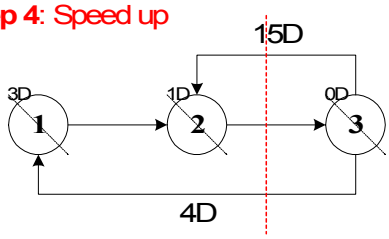
Step 2: 8-slow down the SDFG

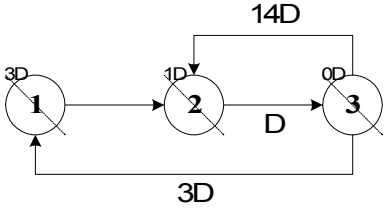


Step 3: Pipeout delays
(lump the synchronization elements)



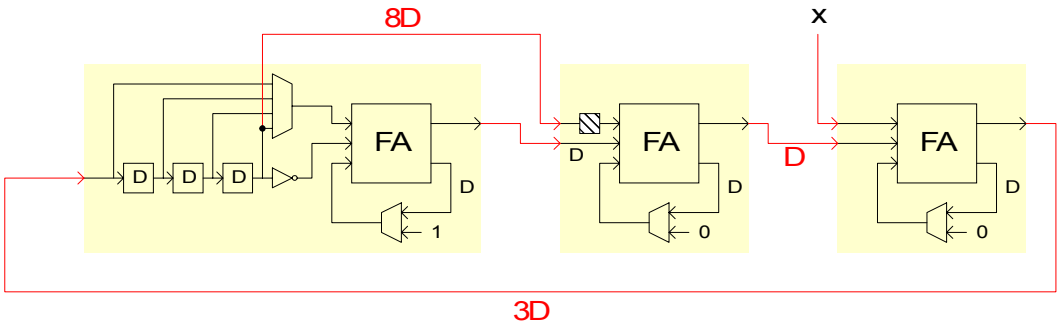
Step 4: Speed up





Example 5-9 (3/3)

Resulting serial architecture



Bit-Serialization of SDFG (5/5)

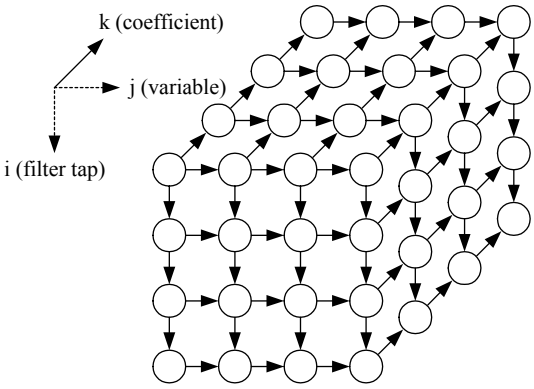
- An ***acyclic SDFG*** can always be bit-serialized, but it is not true for ***cyclic SDFGs*** because of the ***loop synchronization*** problem.
  - constant coefficient optimization with increased wordlength
  - digit serialization
  - hardware slow down
- If the design requires 100% hardware utilization but no independent stream exists (thus slowdown cannot be applied), Skip the J-slowdown in step 1&4 in the procedure and a ***minimum wordlength*** can be derived for bit-serialization.
- More internal delay would be required if the constant coefficient needs more than two terms (e.g. 11/16 = 1- 1/4 - 1/16) and the critical path is constrained to 1 FA delay.

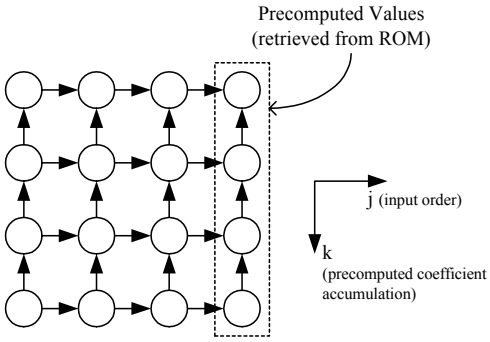
Distributed Arithmetic (DA)

$$\sum_{i=0}^{L-1} h_i \cdot x_i = \sum_{i=0}^{L-1}\left(\sum_{j=0}^{N-1} h_{i,j} \cdot 2^{-j}\right)\cdot x_i = \sum_{j=0}^{N-1}\left(\sum_{i=0}^{L-1} h_{i,j} \cdot x_i\right)\cdot 2^{-j}$$

L *N*-bit MAC operations

N $2^L \times N$-bit table lookup + 1 *N*-bit multiplication





Outline

- Introduction
- Algorithm Representation and Iteration Bound
- Retiming
- Unfolding and Folding Transformation
- Bit-serial Architectures
- ***Case Study – DWT Processor***

Spectral Analysis

- Fourier transform

$$F(w)=\int_{-\infty}^{\infty} f(t)\cdot e^{jwt}dt$$

  - for stationary signal
- Short term Fourier transform (STFT)

$$F(w,\tau)=\int_{-\infty}^{\infty} f(t)\cdot g^{*}(t-\tau)\cdot e^{jwt}dt$$

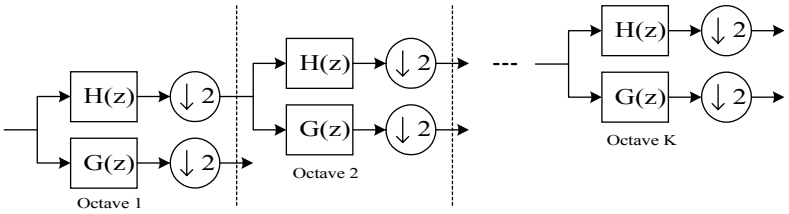
  - tradeoff between temporal and spectral resolution (Principle of Uncertainty)
- Wavelet transform

$$W(u,s)=\int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{s}}\Psi^{*}\left(\frac{t-u}{s}\right)dt = f\otimes\overline{\Psi}_{s}(u)$$

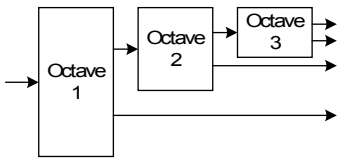
Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Case Study - DWT Processor
Chein-Wei Jen cwjen@ee.nctu.edu.tw

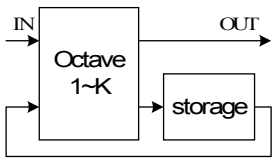
Pyramid Algorithm

- Iterative fashion with the identical filtering module instead of computing all convolutions

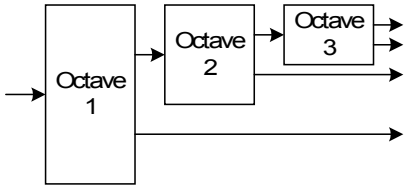


- How to increase the hardware utilization

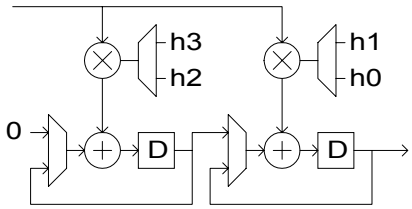




Intra-Octave Folding



- Folded filters
  (shared MACs between H/L filters or among taps inside one filter)

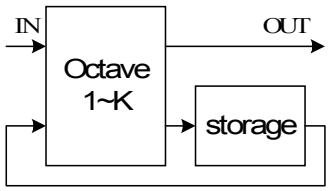


- Digit-serial filters

Chapter 5 - Algorithm and Architecture Exploration in IP Core Design
Case Study - DWT Processor
Chein-Wei Jen cwjen@ee.nctu.edu.tw

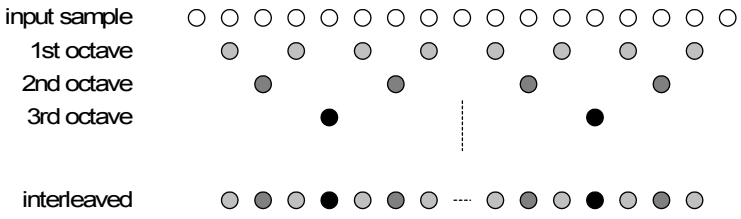
Inter-Octave Folding

Using one set of filter module for all octaves

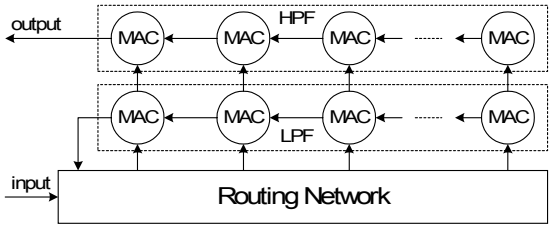


- Blocking scheduling
  - storage size of O(N)
- Non-blocking scheduling
  - storage size of O(KL)

  e.g. Modified Recursive Pyramid Algorithm (MRPA)



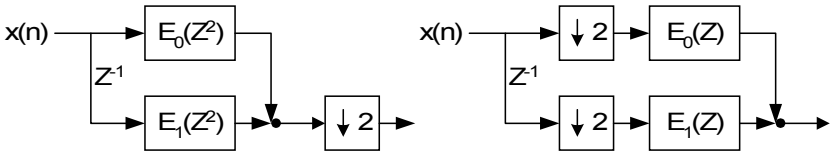
- DWT architecture design
  - filtering module
  - data generation



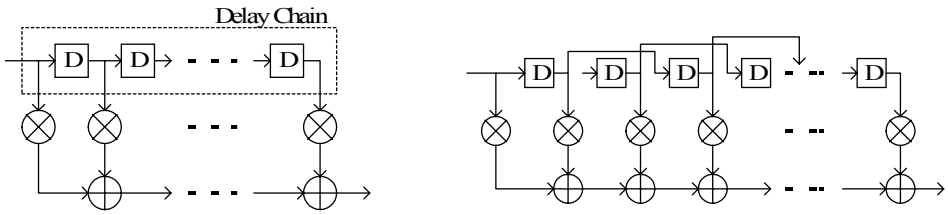
Polyphase Decomposition

$$H(z)=\sum_{-\infty}^{\infty}h(2n)z^{-2n}+z^{-1}\sum_{-\infty}^{\infty}h(2n+1)z^{-2n}$$

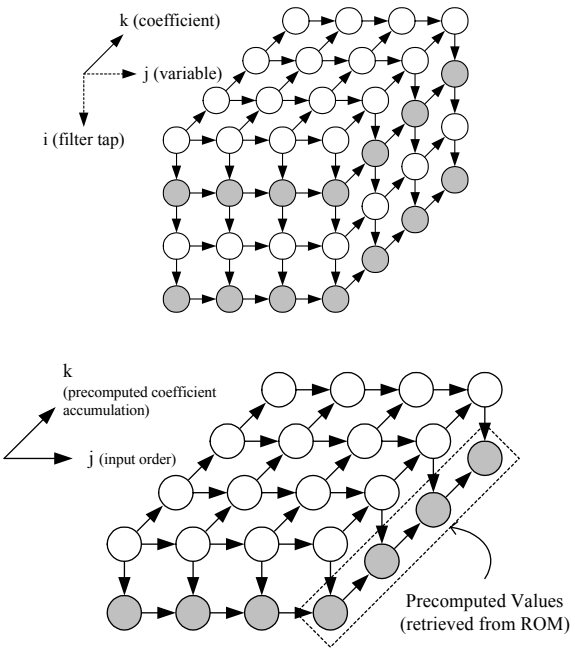
Decimation or interpolation filters



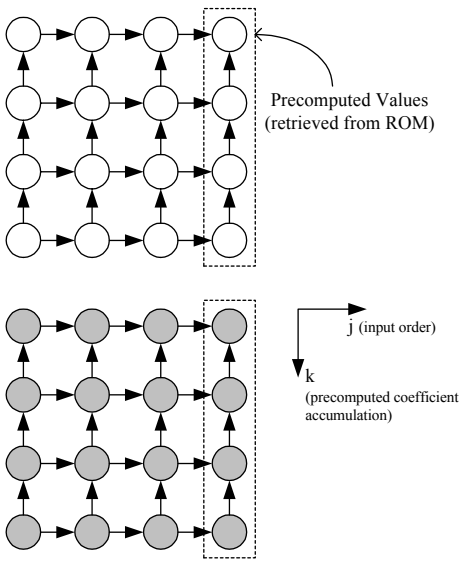
Skip unnecessary computations (to be discarded or null input)



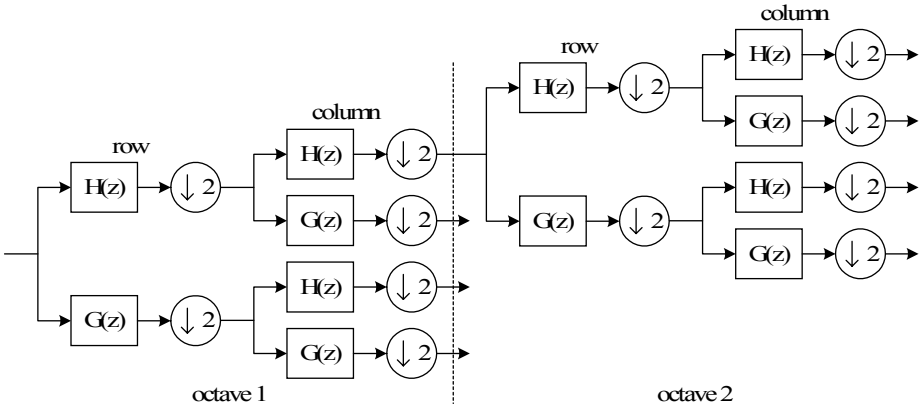
DA Filters with Polyphase Decomposition



N 2×2$^{L/2}$×*N*-bit table lookup +
2 *N*-bit multiplication



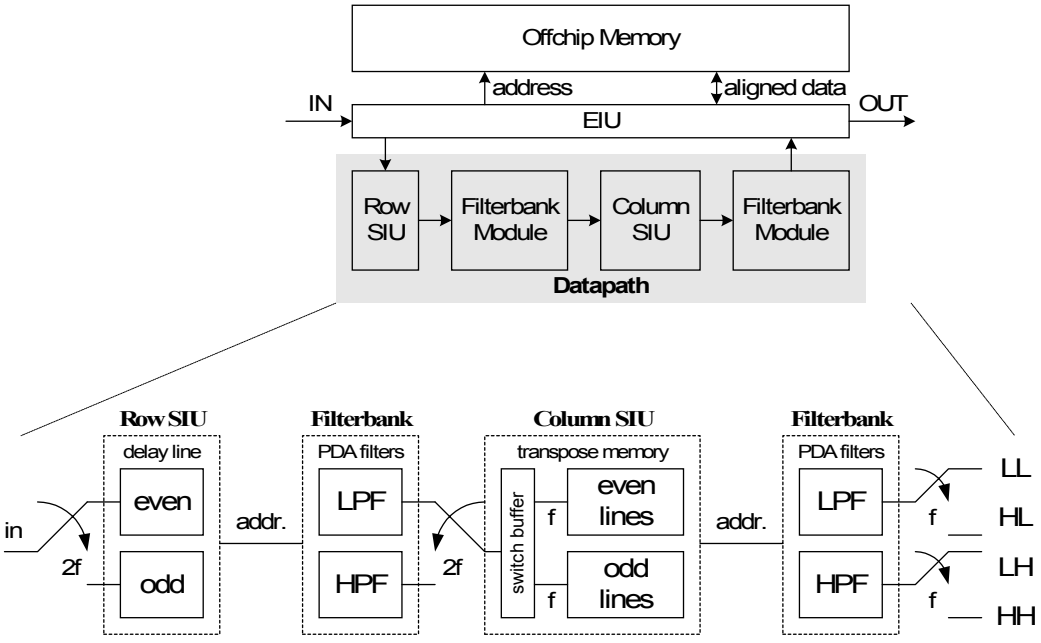
2-D DWT



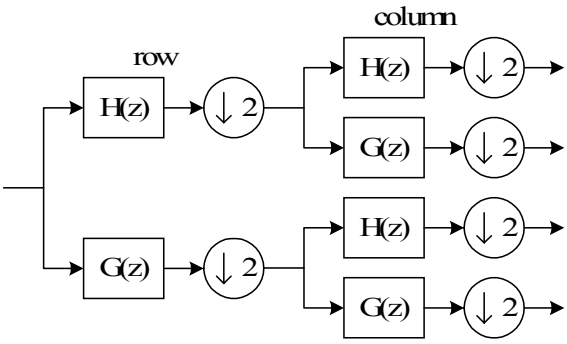
- The data rate of one octave is only 1/4 of its preceding octave
- A transpose memory is required between row & column filtering modules (not necessarily the same)
- Any combination of the aforementioned 1-D architectural transformation techniques can be applied

A 2-D DWT Architecture

Blocking, inter-octave folded architecture with interleaved column filtering



Interleaved Column Filters



- Scaling down of the DA-based hardware is not trivial
- Actually, the total data rate is the same inside one octave
- Interleaving (hardware slow down) can be utilized here to multiplexes multiple independent streams into the hardware

Chein-Wei Jen cwjen@ee.nctu.edu.tw