DSP Design – Lecture 8

Strength Reduction

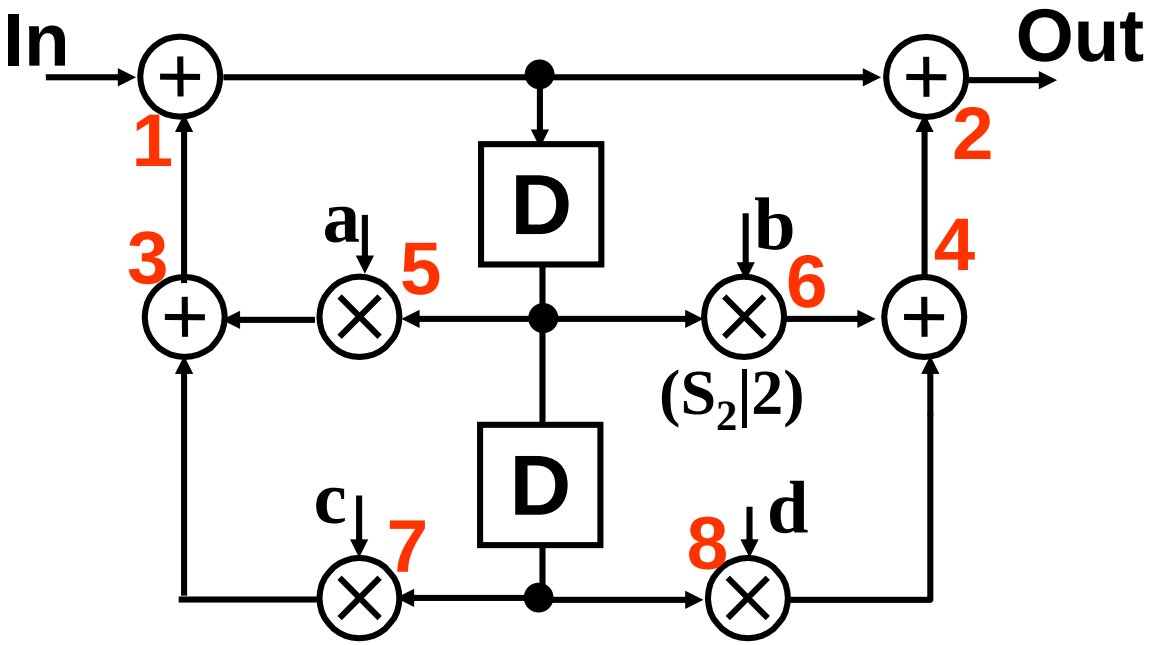
Fredrik Edman

fredrik.edman@eit.lth.se

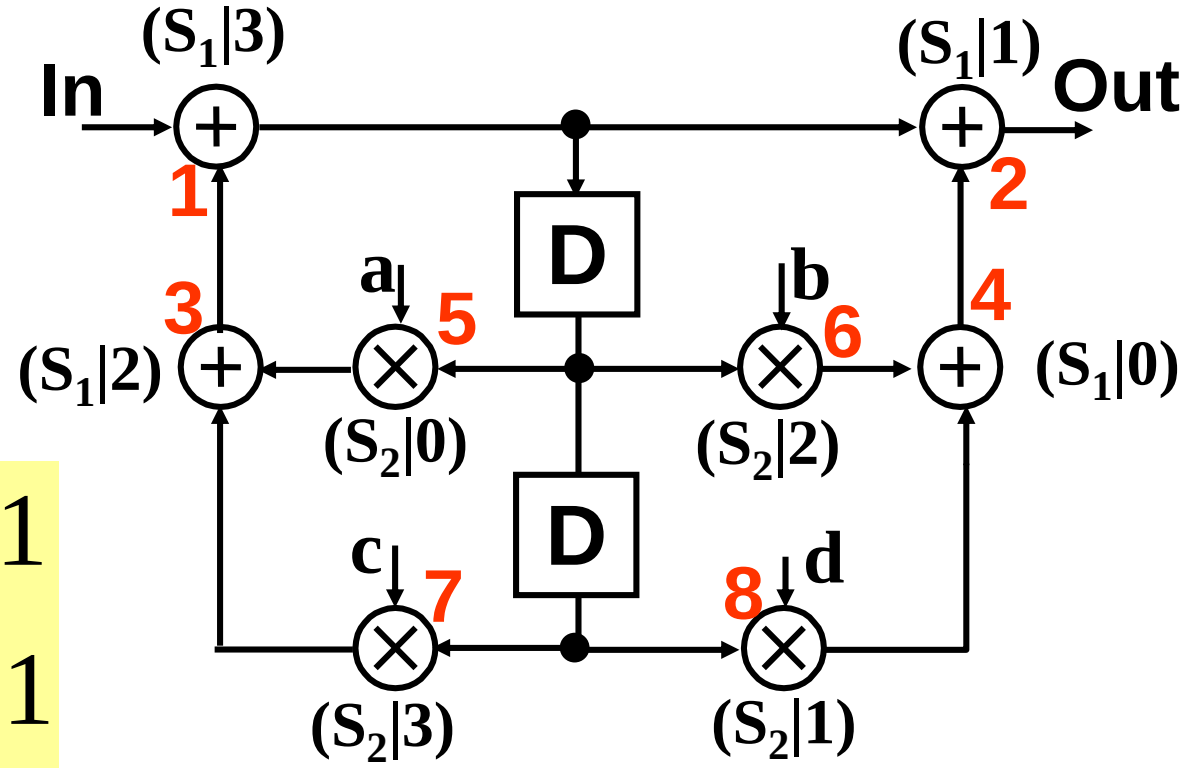
Folding (cont.)

Chapter 6

Ex. Folding of Biquad filter



Ex. Folding of Biquad filter



$$T_{adder} = 1$$

$$P_{adder} = 1$$

$$T_{mult} = 2$$

$$P_{mult} = 2$$

Additions

$$S_1 = \{4,2,3,1\}$$

Multiplication

$$S_2 = \{5,8,6,7\}$$

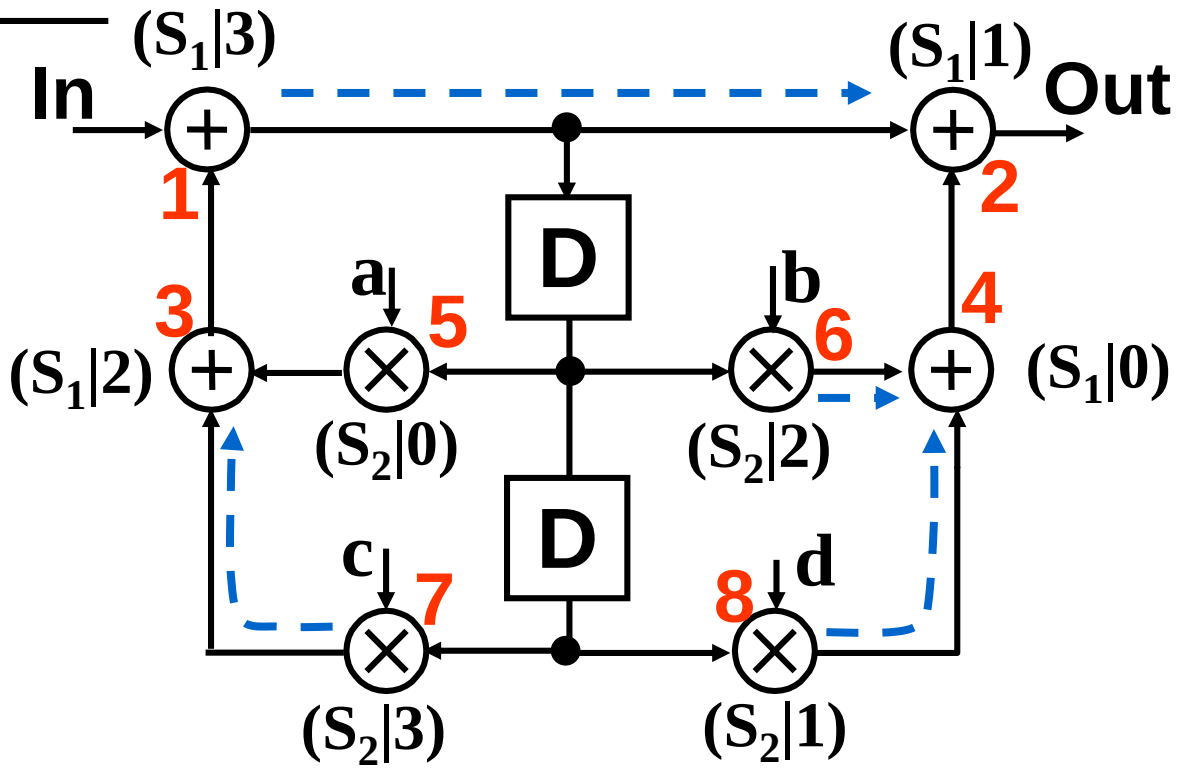
Folding of Biquad filter, N=4

receive send

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

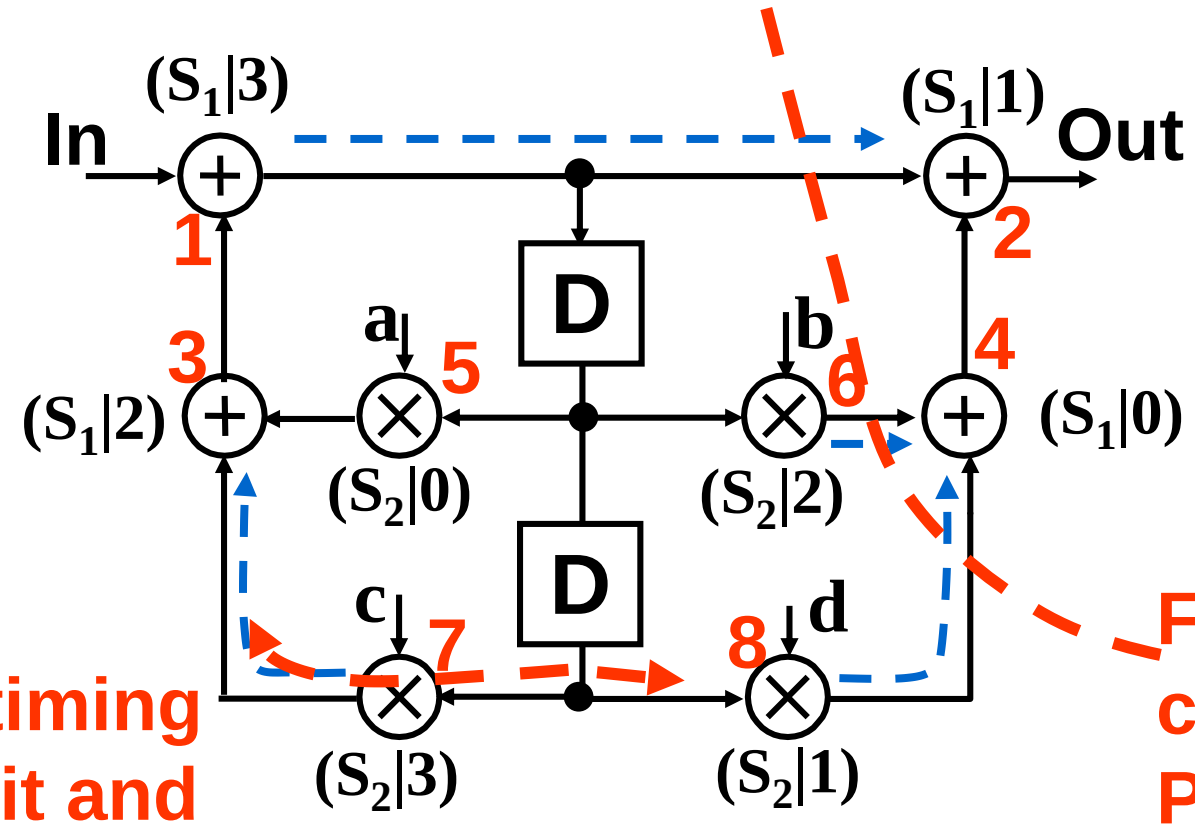
A delay between two edges can not be negative!

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



$D_F(U \rightarrow V) < 0$ ➡ **Not Valid folding**

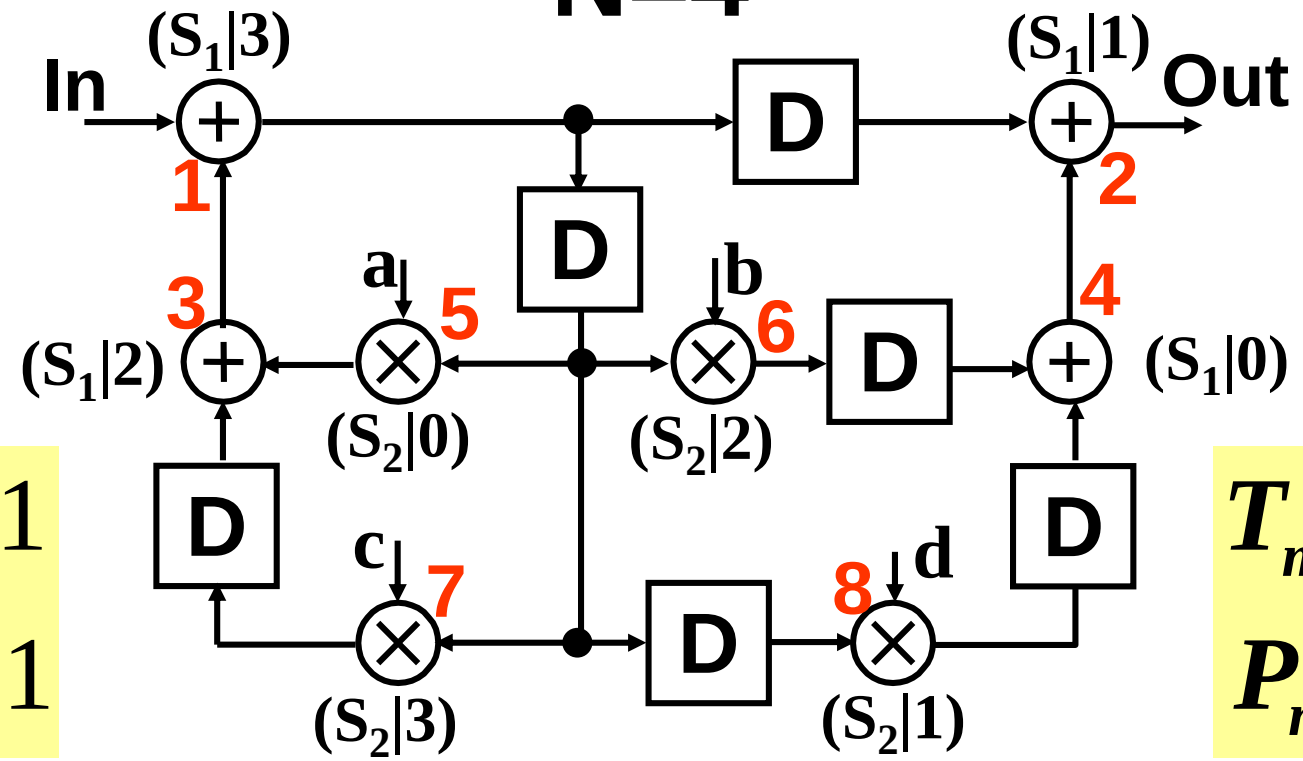
Retiming: Folding of Biquad filter, N=4



Retiming Split and move delay

Feedforward cutset ⇨ Pipelining

Folding of retimed Biquad filter, N=4



$$T_{adder} = 1 \qquad P_{adder} = 1$$

$$T_{mult} = 2 \qquad P_{mult} = 2$$

Additions

$$S_1 = \{4,2,3,1\}$$

Multiplication

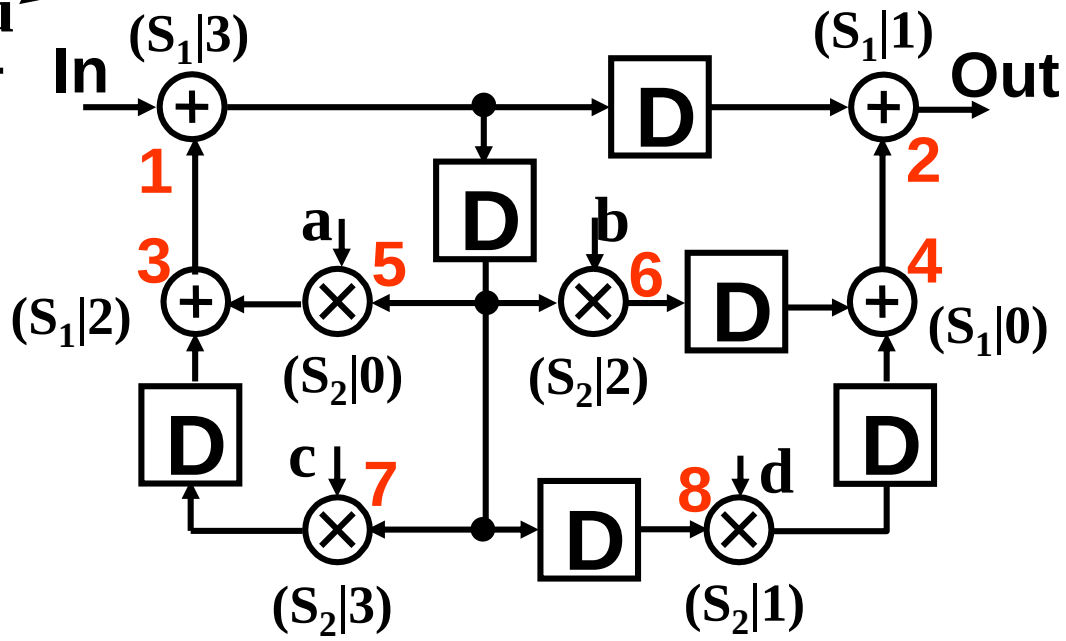
$$S_2 = \{5,8,6,7\}$$

Folding of retimed Biquad filter, N=4

receive send

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

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



Valid folding

$$D_F(U \rightarrow V) \geq 0$$

Folding of retimed Biquad filter, N=4

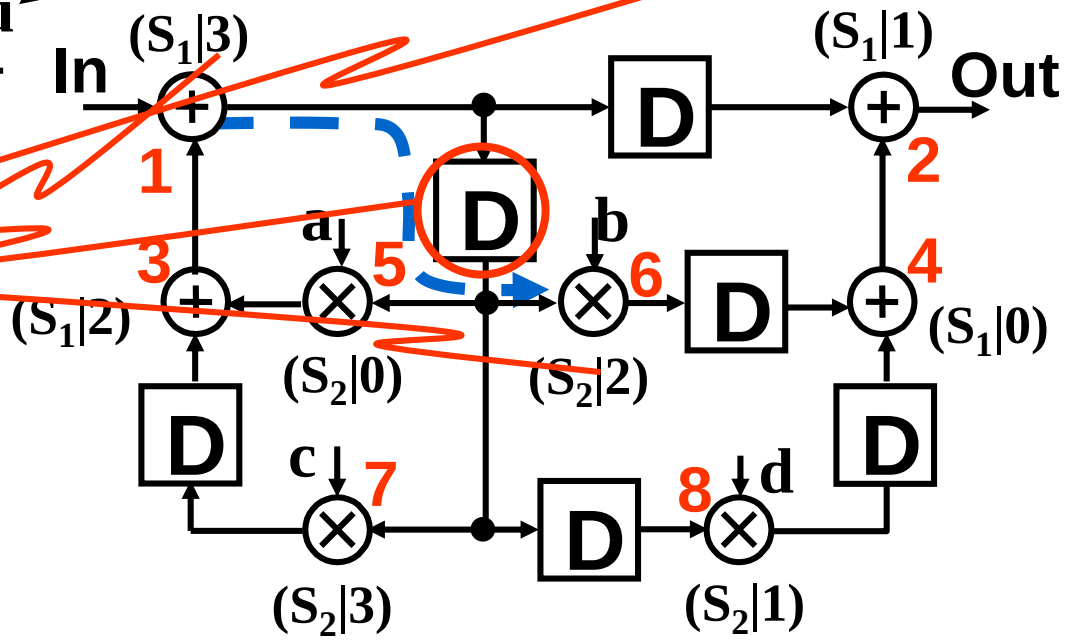
receive

send

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

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

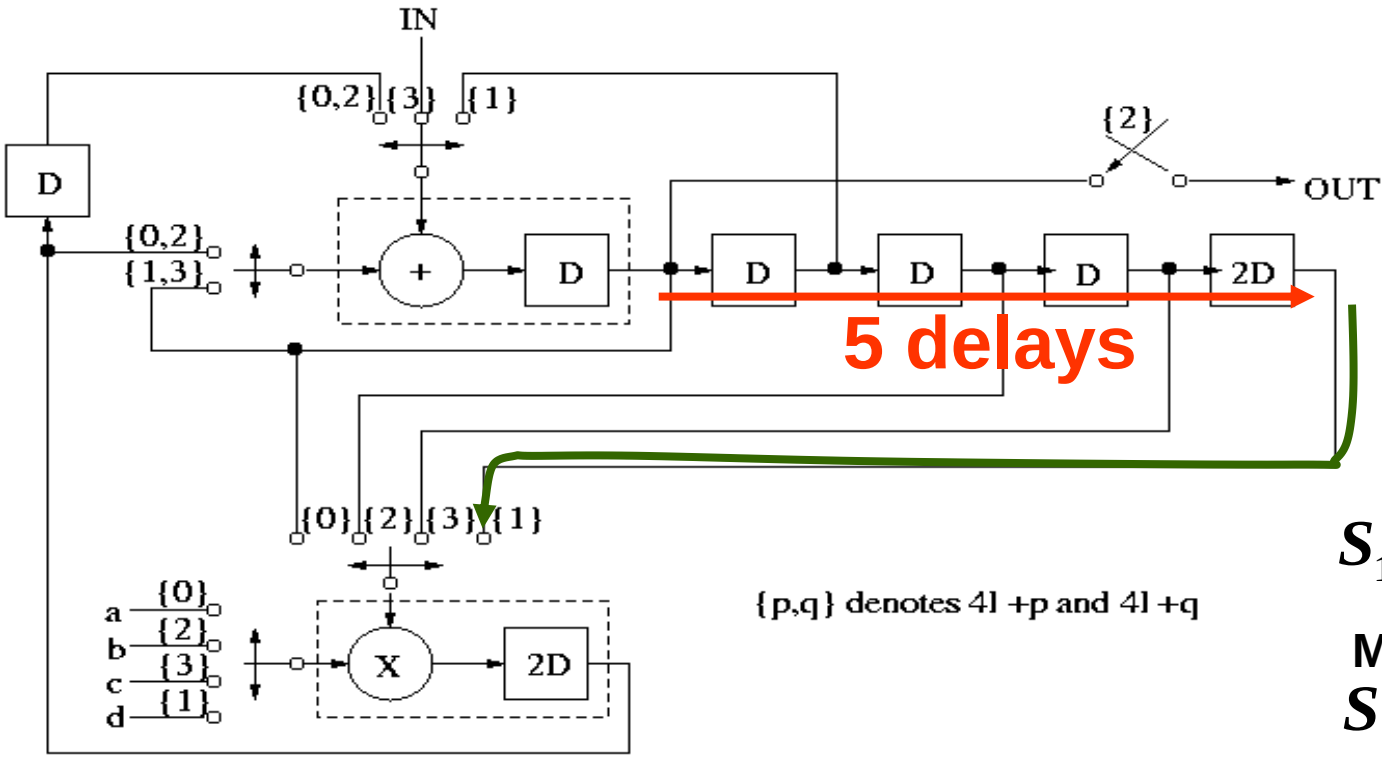
$P_{adder} = 1$



Valid folding

$$D_F(U \rightarrow V) \geq 0$$

Folding of retimed Biquad filter, N=4



Additions

$$S_1 = \{4,2,3,1\}$$

Multiplication

$$S_2 = \{5,8,6,7\}$$

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

$D_F(1 \rightarrow 8) = 4(2) - 1 + 1 - 3 = 5$ ⇨ path from add to mult with 5D

Node 8 has folding order 1 ⇨ switch close at 1

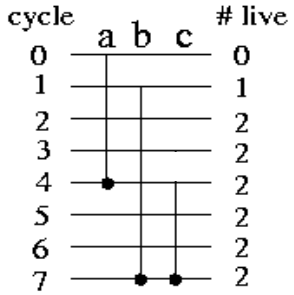
Folding & Register Minimization

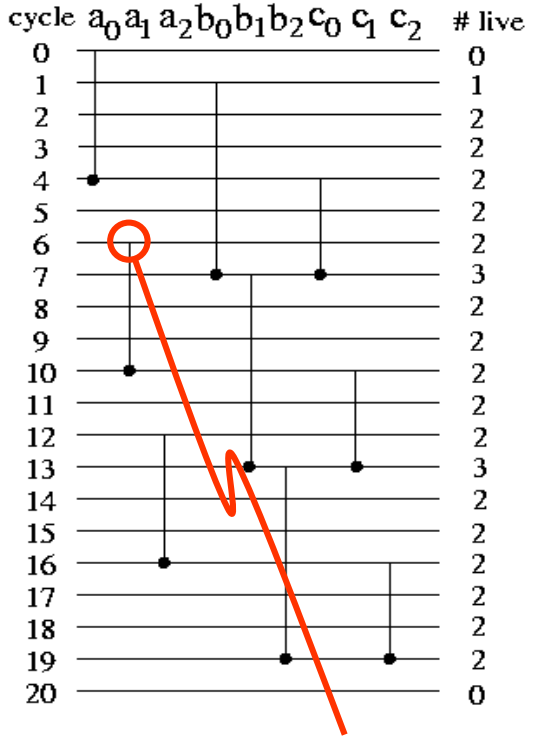
Chapter 6.3

Register/Storage Minimization

Folding inserts register. Lifetime analysis is used for register minimization techniques in a DSP hardware.

- A variable is live from the time it is produced until the time it is consumed. After that it is dead.
- Linear lifetime chart : Represents the lifetime of the variables in a linear fashion.
- Convention: a variable is
  - not live during the clock cycle when it is produced
  - but live during the clock cycle when it is consumed.

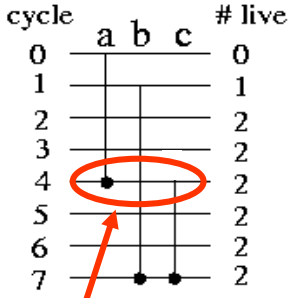


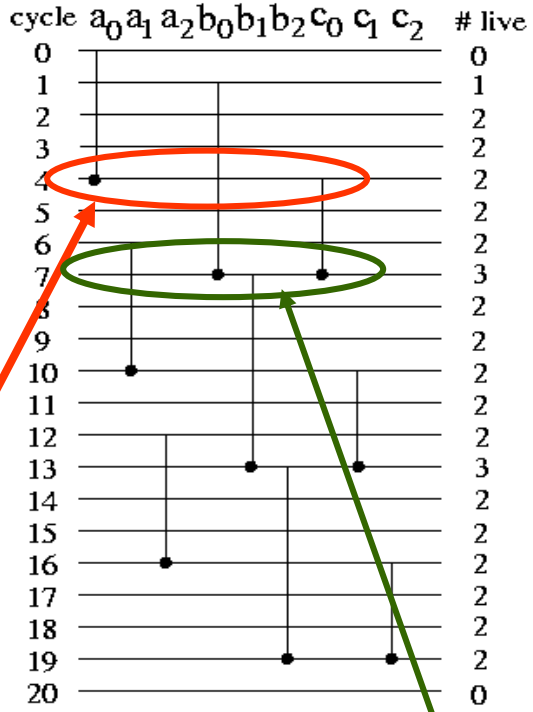


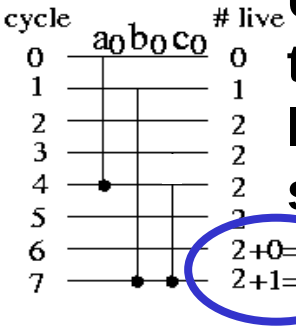
One iteratiom 6 cc ⇨ N=6

Register Minimization

Max. number of live variables ⇨ Min. number of registers







Use previous iter. to avoid drawing lifetime chart over several iterations

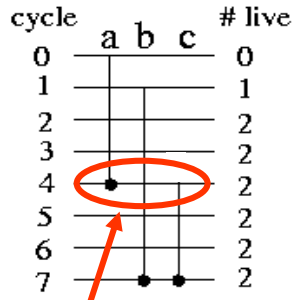
2 live variables

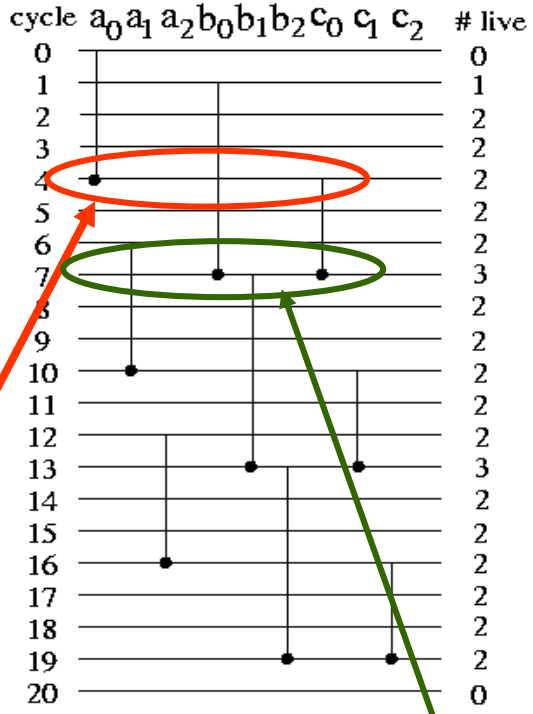
But 3 if several iterations

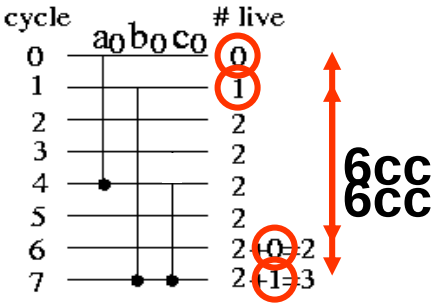
2 live variables in iteration

Register Minimization

Max. number of live variables ⇨ Min. number of registers







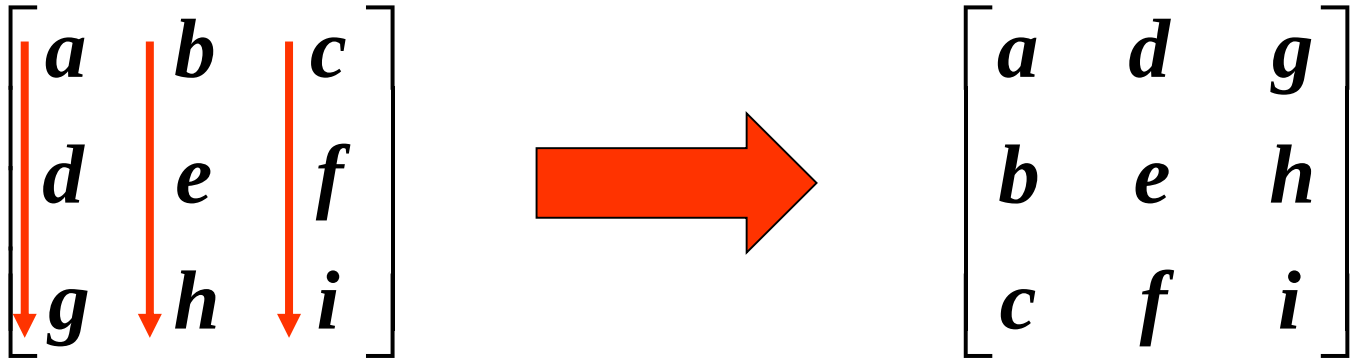
2 live variables

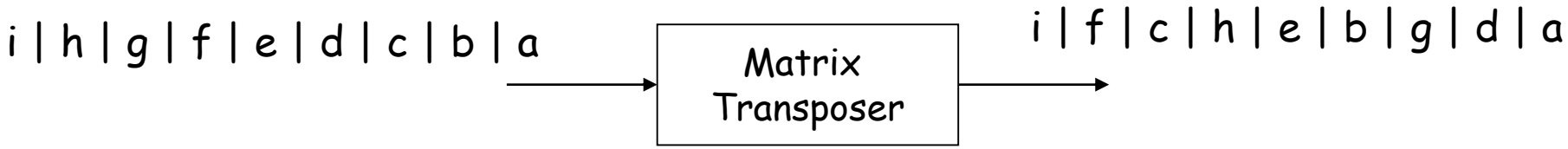
But 3 if several iterations

2 live variables in iteration

Example of a systematic way of working with lifetime charts

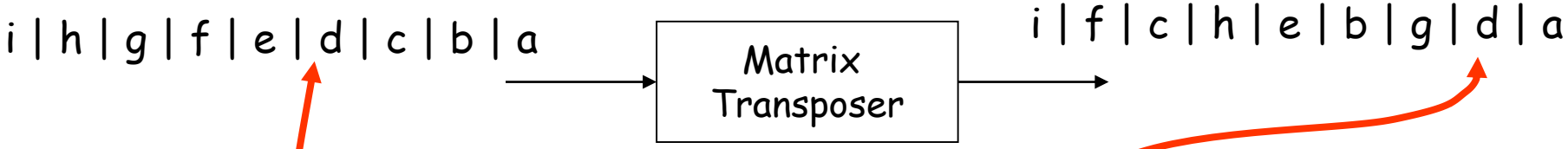
3x3 Matrix Transpose





One iteration = 9 clock cycles

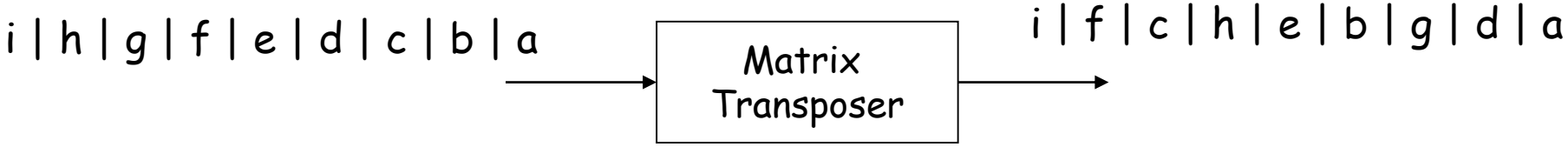
Lifetime Table - 3x3 Matrix Transpose



Sample	T_{in}	T_{zlout}	T_{diff}	T_{out}	Life
a	0	0			
b	1				
c	2				
d	3	1	-2		
e	4	4			
f	5	7			
g	6	2			
h	7	5			
i	8	8			

Out before In

Lifetime Table - 3x3 Matrix Transpose



Sample	T_{in}	T_{zlout}	T_{diff}	T_{out}	Life
a	0	0	0		
b	1	3	2		
c	2	6	4		
d	3	1	-2		
e	4	4	0		
f	5	7	2		
g	6	2	-4		
h	7	5	-2		
i	8	8	0		

$T_{diff} = T_{zlout} – T_{input}$, where T_{zlout} = zero latency

3x3 Matrix Transpose

Sample	T_{in}	T_{zlout}	T_{diff}	T_{out}	Life
a	0	0	0		
b	1	3	2		
c	2		4		
d	3	1	-2	**5**	
e	4	4	0		
f	5	7	2		
g	6	2	-4		
h	7	5	-2		
i	8	8	0		

if $T_{diff} < 0$ not causal ⇨
add latency = $|T_{negative\ diffmax}|$ for all nodes

3x3 Matrix Transpose

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		4		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

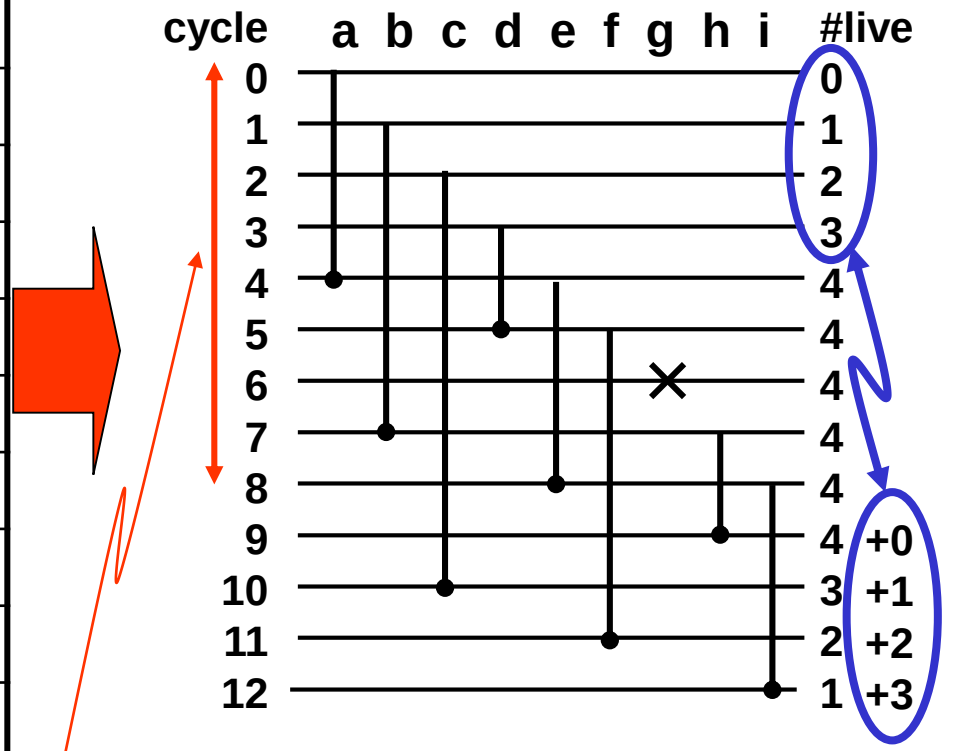
if $T_{diff} < 0$ not causal ⇨

add latency $T_{lat} = |T_{negative\ diffmax}|$ for all nodes

$T_{out} = T_{zlout} + T_{lat}$

Lifetime chart 3x3 Matrix Transpose

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

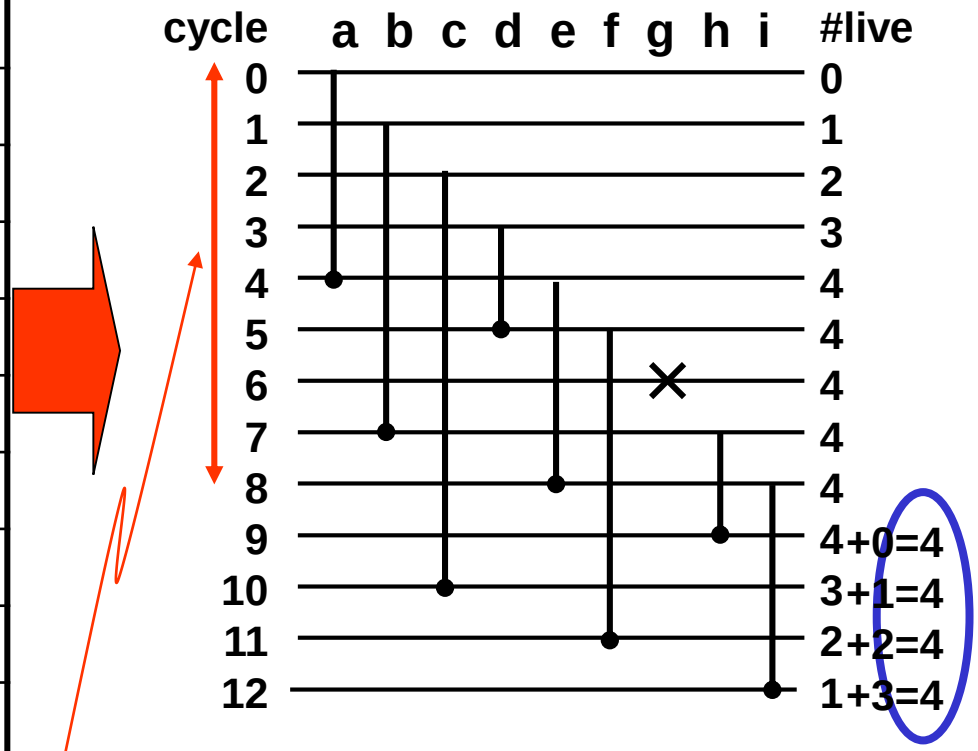


One iteration = 9 clock cycles

Contribution from next iteration

Lifetime chart 3x3 Matrix Transpose

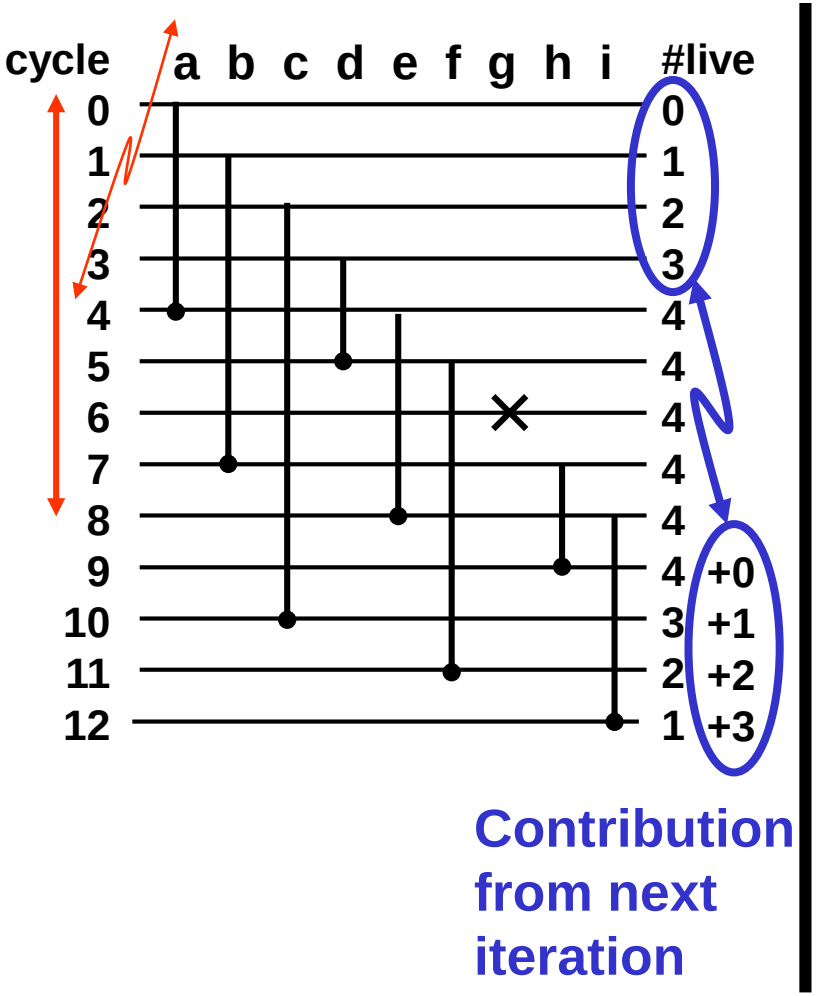
Sample	T_{in}	T_{zlout}	T_{diff}	T_{out}	Life
a	0	0	0	4	0→4
b	1	3	2	7	1→7
c	2	6	4	10	2→10
d	3	1	-2	5	3→5
e	4	4	0	8	4→8
f	5	7	2	11	5→11
g	6	2	-4	6	6→6
h	7	5	-2	9	7→9
i	8	8	0	12	8→12

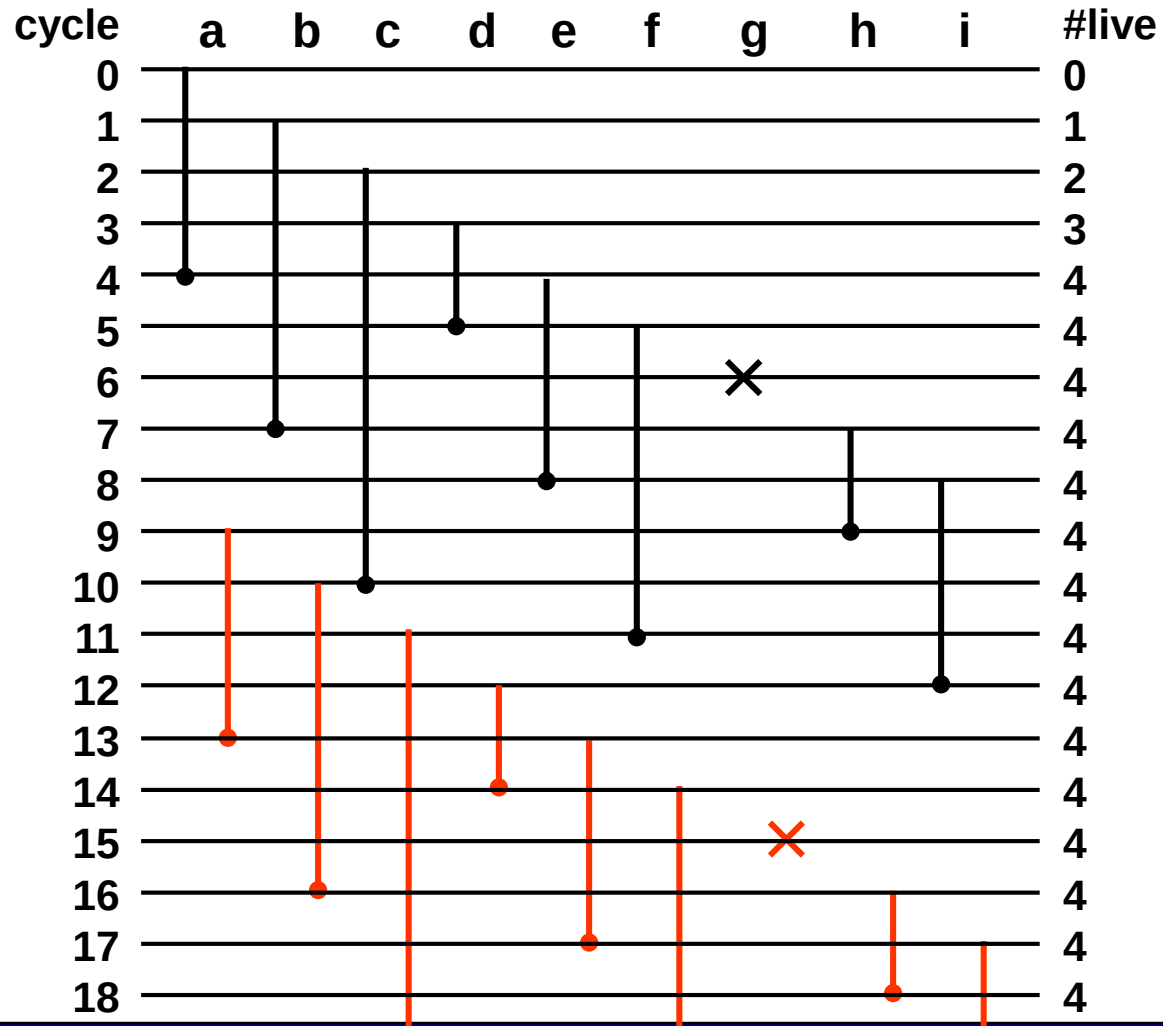


One iteration =
9 clock cycles

Lifetime chart

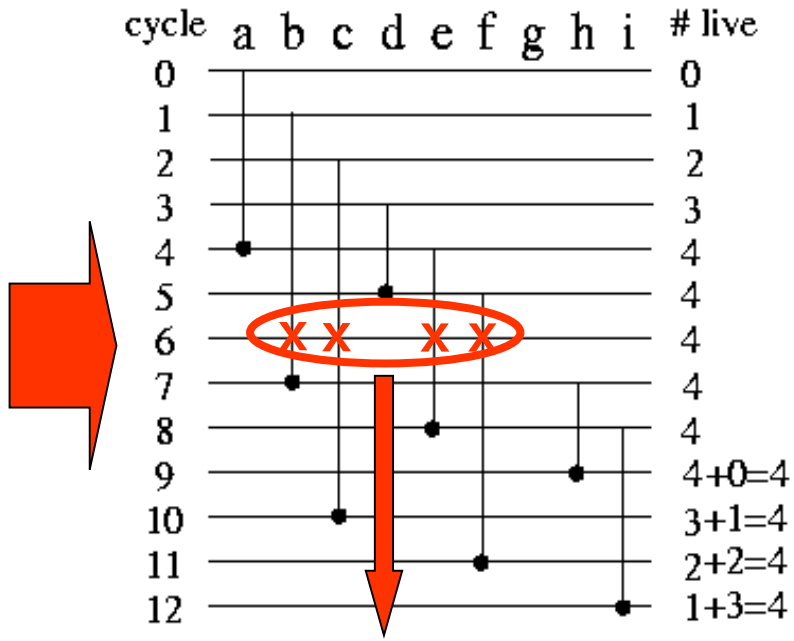
One iteration =
9 clock cycles





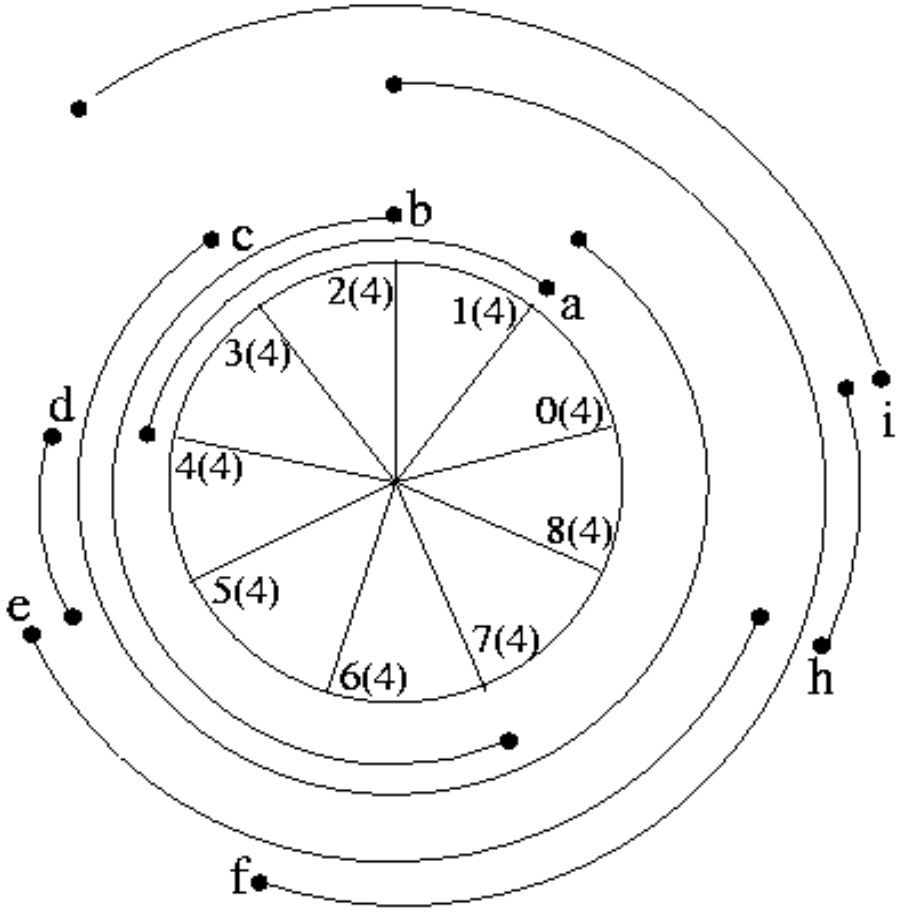
Lifetime chart 3x3 Matrix Transpose

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



max #live = 4 registers

Circular lifetime chart



Useful to represent the periodic nature of the DSP programs.

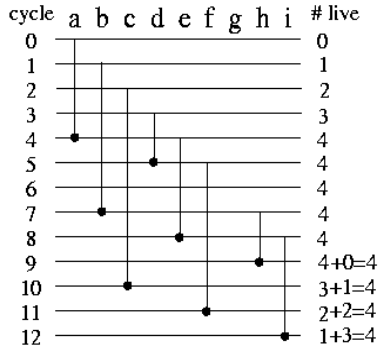
- 0 – (N-1) pies
- Number in parantheses represents the number of live variables at each time instance

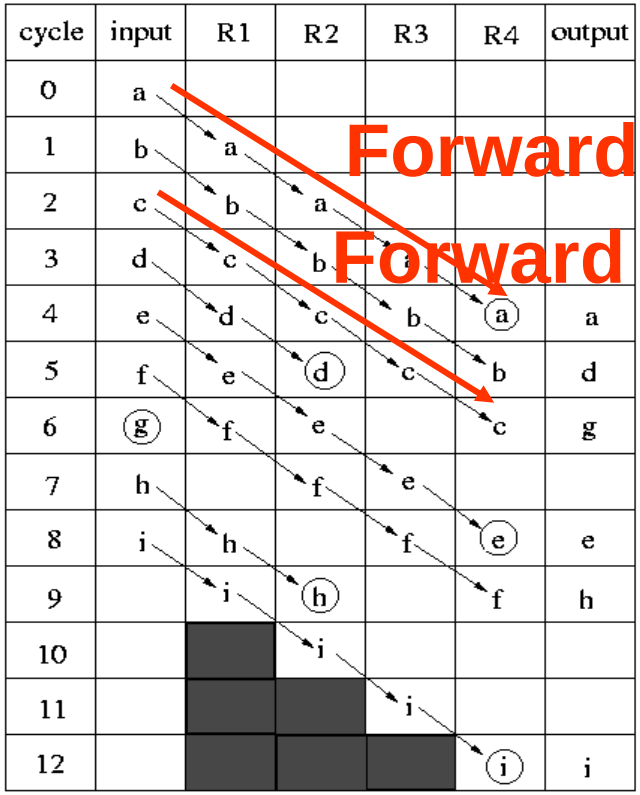
Forward Backward Register Allocation Technique

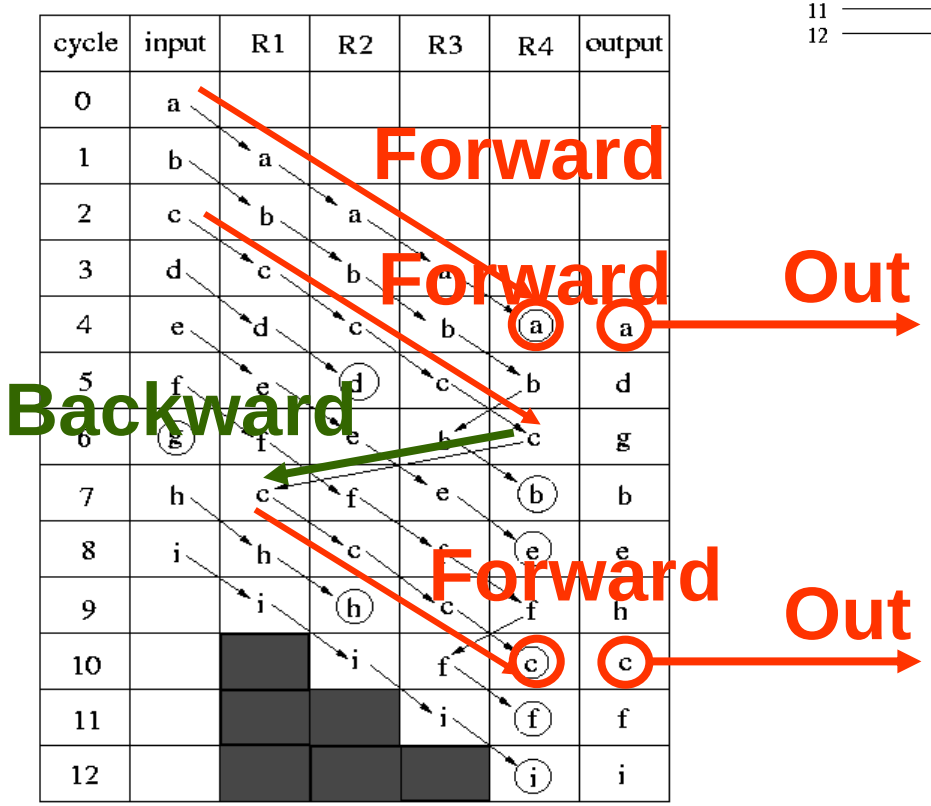
Steps for Forward-Backward Register allocation :

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 *I* + N if it holds a variable during cycle *I*.**
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.**

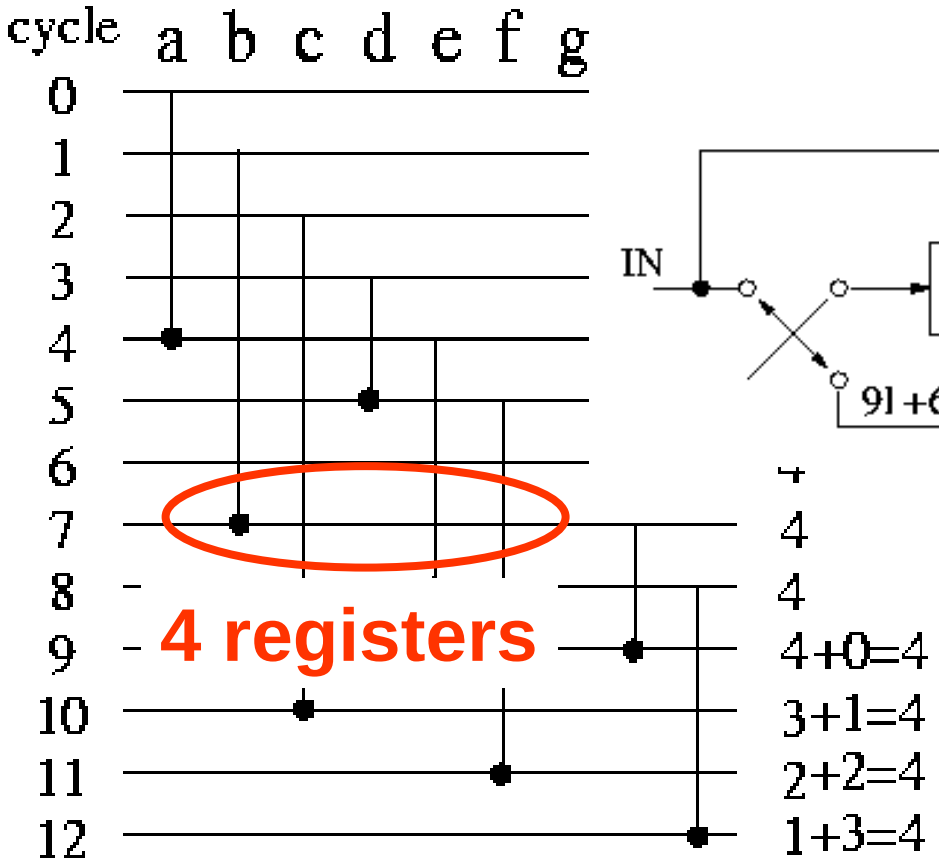
Forward Backward Register Allocation Technique

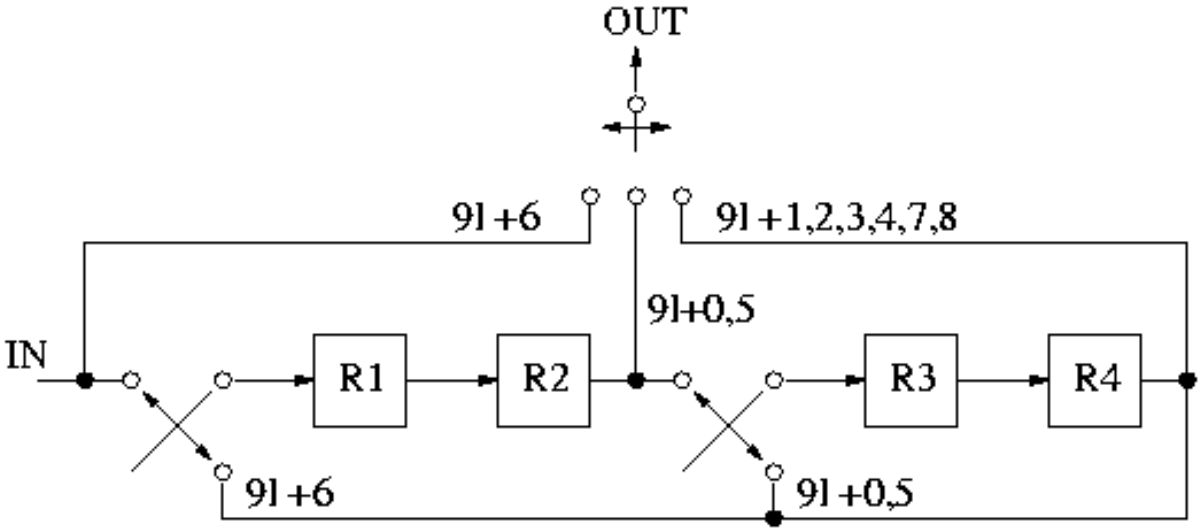






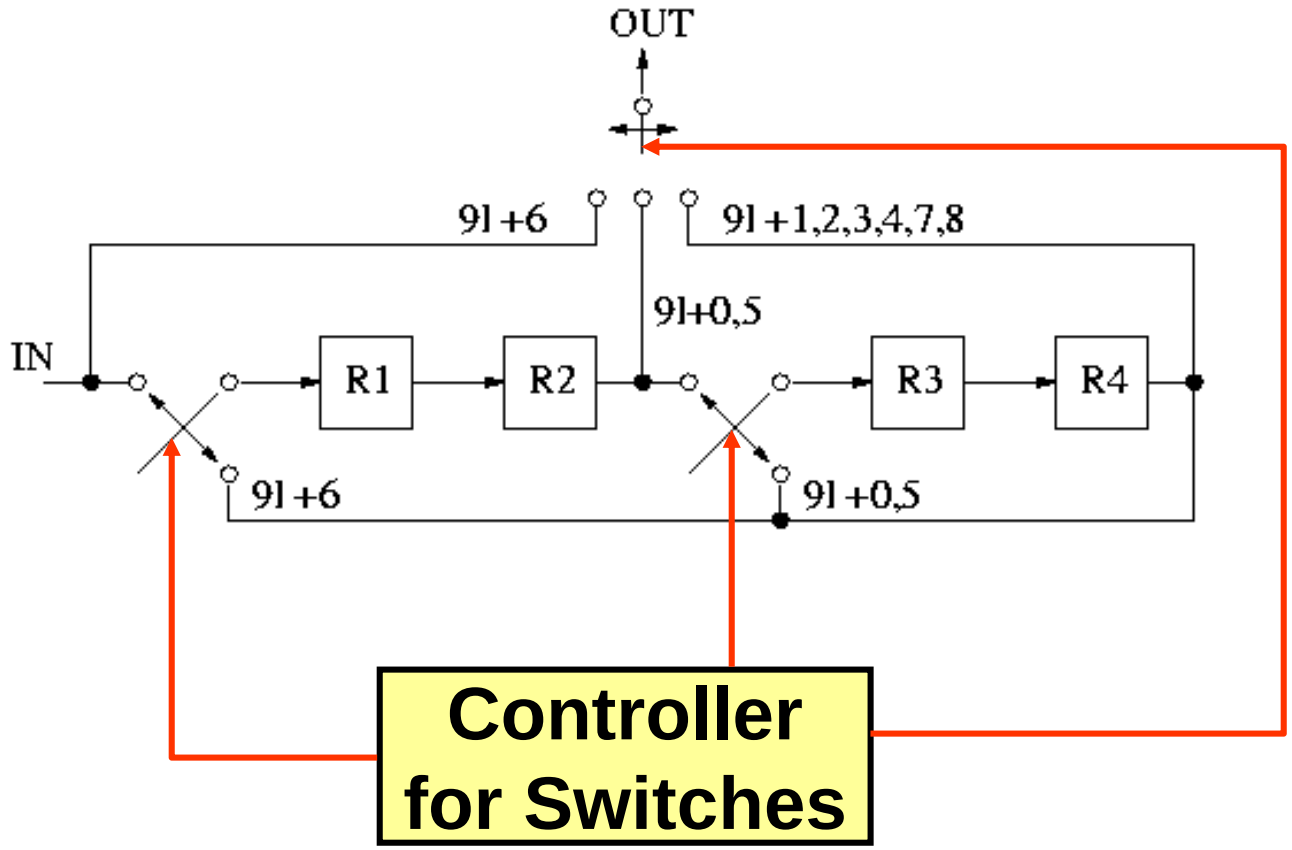
Lifetime chart 3x3 Matrix Transpose





Possible to use memory with 4 positions

Controller for Folded Architecture



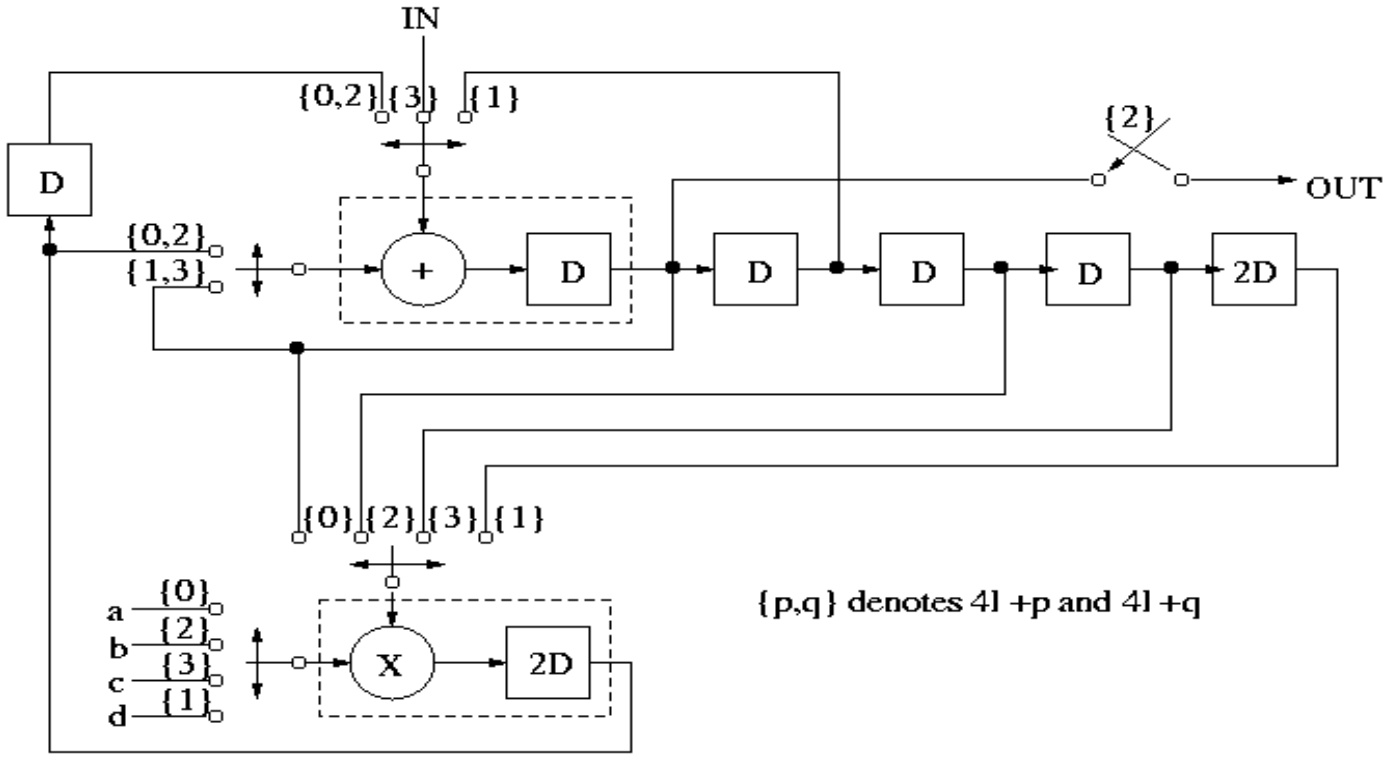
Algorihm for Register Minimization Procedure

1. Perform retiming for folding. (6.2)
2. Write the folding equations. (6.2)
3. Construct a lifetime chart and determine the required number of registers. (6.3.1)
4. Perform forward-backward register allocation. (6.3.2)
5. Draw the register minimized folded architecture.

Folding & Register Minimization

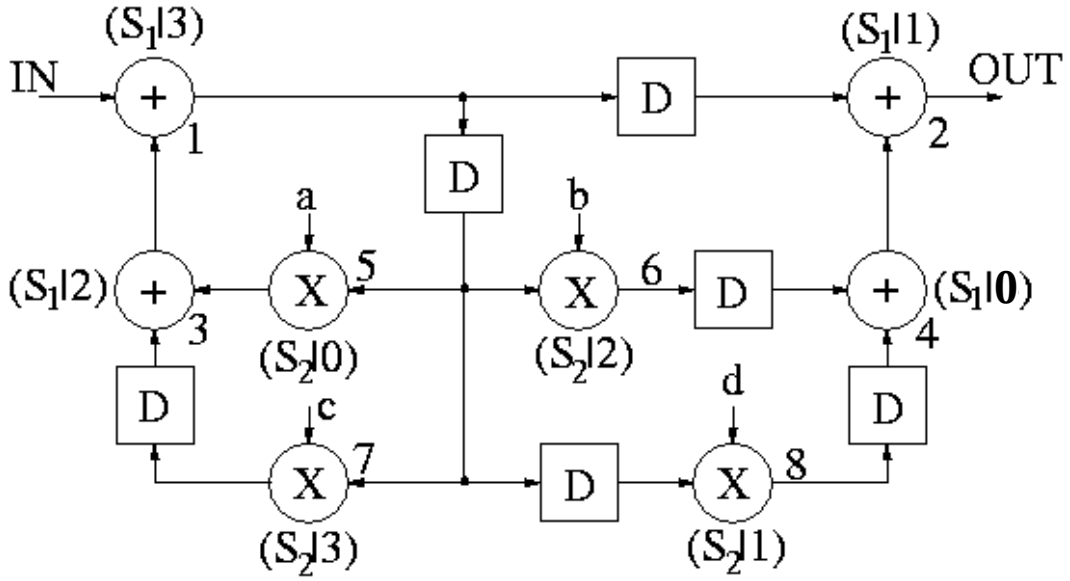
Biquad Filter (cont.)

Folded and Retimed Biquad Filter



Let's see if we can reduce # of registers?

Register Minimization of Retimed Biquad filter



One entry for each node:

- $T_{input} = u + P_u$, u=folding order, P_u=pipeline time unit data is produced

- $T_{output} = u + P_u + \max_V\{D_F(U \rightarrow V)\}$,

$\max_V\{D_F(U \rightarrow V)\}$ = (longest folded path)

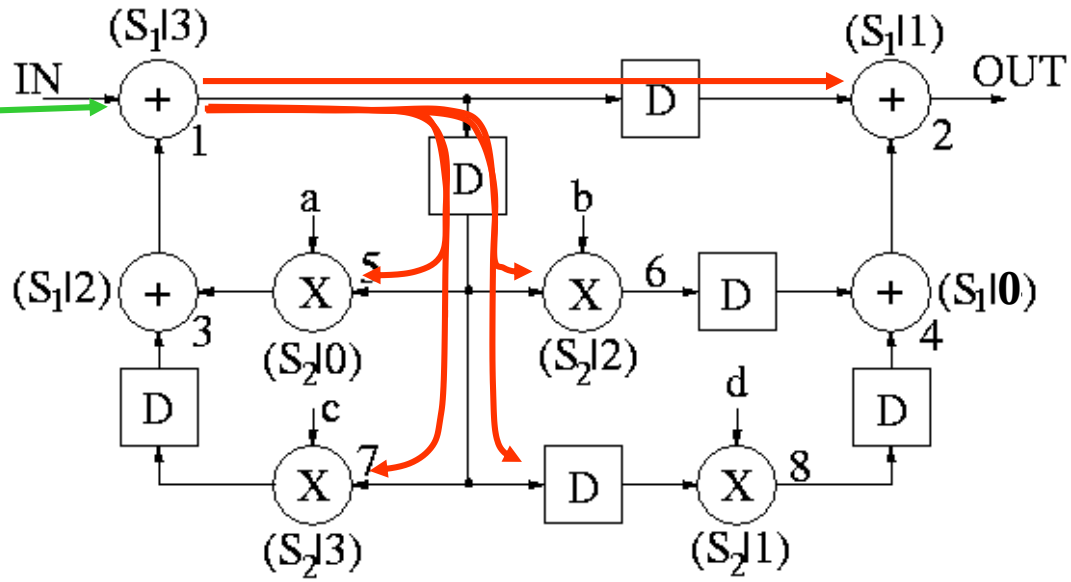
Register Minimization of Biquad filter

$D_F(U \rightarrow V) = Nw(e) - P_u + v - u$ ← send

receive

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

node 1 → 2,5,6,7,8



One entry for each node:

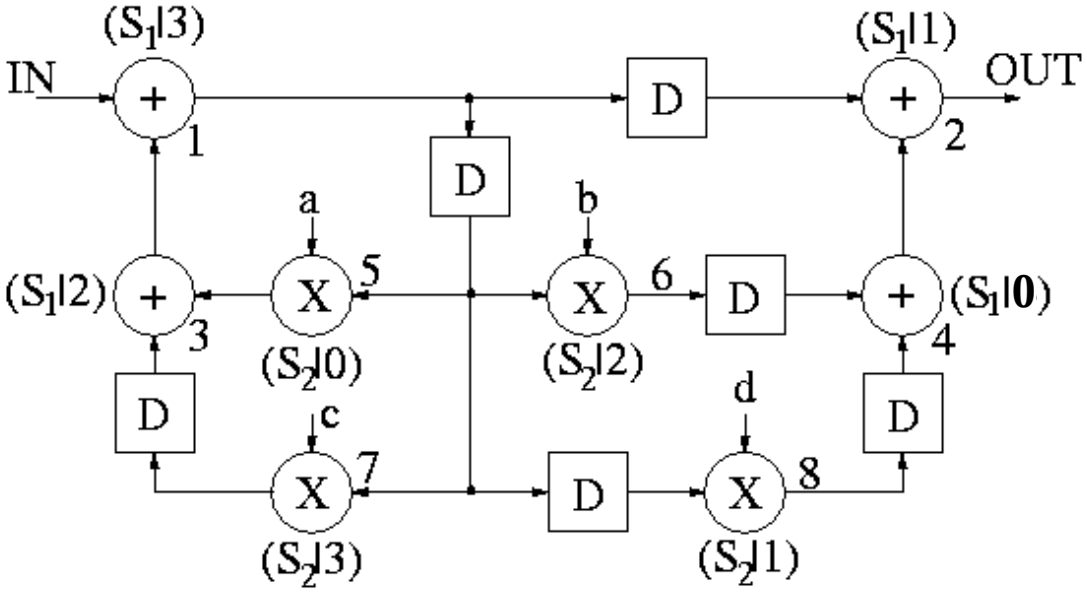
- $T_{input} = u + P_u = 3 + 1 = 4$

- $T_{output} = u + P_u + \max_V\{D_F(U \rightarrow V)\} = 3 + 1 + \max V\{1,0,2,3,5\} = 9$

Register Minimization of Biquad filter

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

Node	$T_{in}\rightarrow T_{out}$
1	4→9
2	--
3	2+1=3→ 2+1+0=3
4	4+1=5→ 4+1+0=5
5	0+2=2→ 0+2=2
6	2+2=4→ 2+2=4
7	3+2=5→ 3+2+1=6
8	1+2=3→ 1+2+1=4

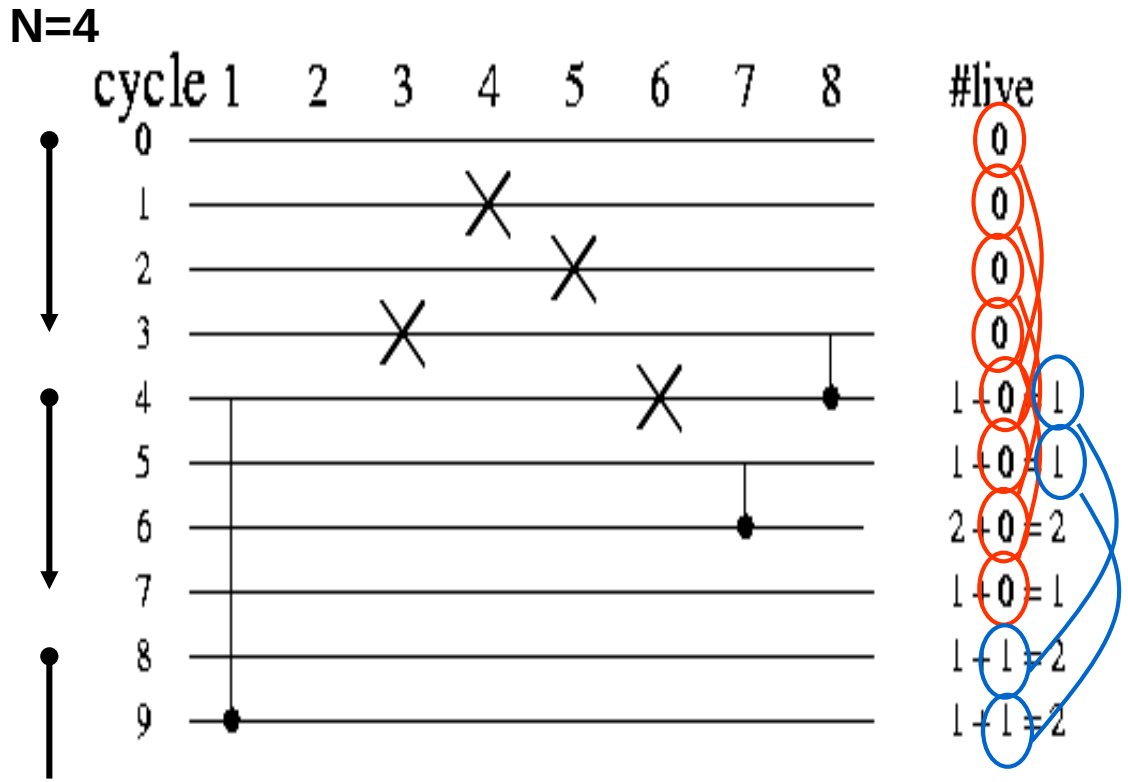


One entry for each node:

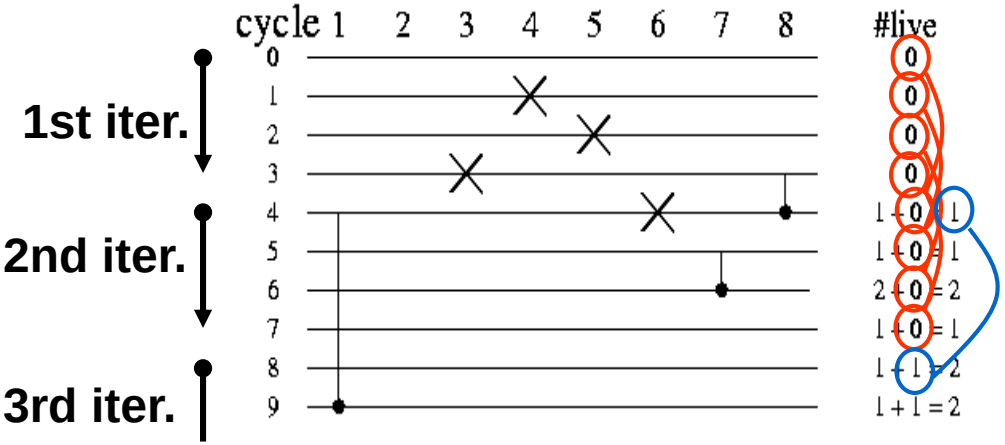
- $T_{input} = u + P_u$
- $T_{output} = u + P_u + \max_V\{D_F(U \rightarrow V)\}$

Lifetime chart of Biquad filter...

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

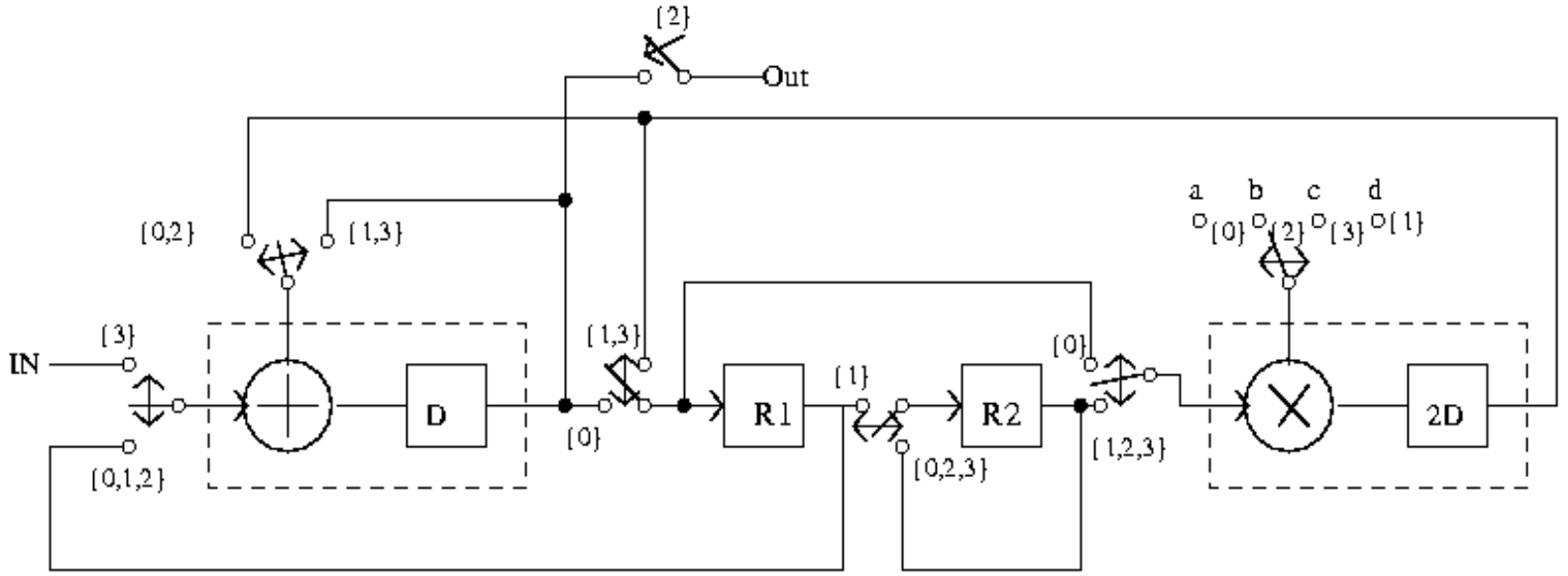


...and Register Allocation

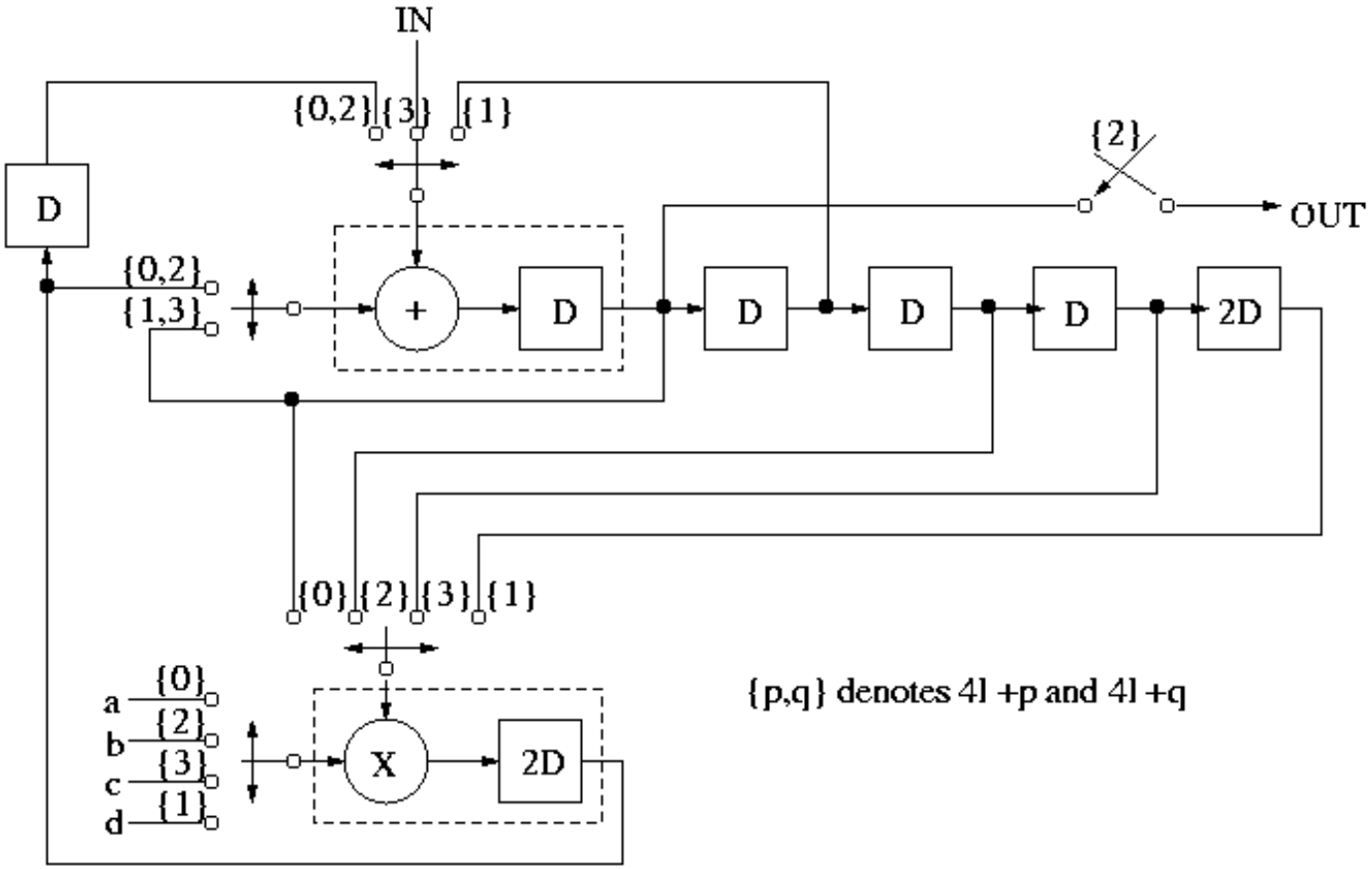


cycle	input	R1	R2	output
0				
1				
2				
3	n_8			
4	n_1	n_8		n_8
5	n_7	n_1		
6		n	n_1	n_7
7			n_7	
8			n_7	
9			n	n_1

Folded architecture is drawn with minimum # of registers.



Previous non min reg. architecture



Strength Reduction

Chapter 9

Algorithmic Strength Reduction

Reduce the number of strong operations

- Reduces the switched capacitance
- Area?
- Speed?

Algorithmic Strength Reduction

What is a strong operation?

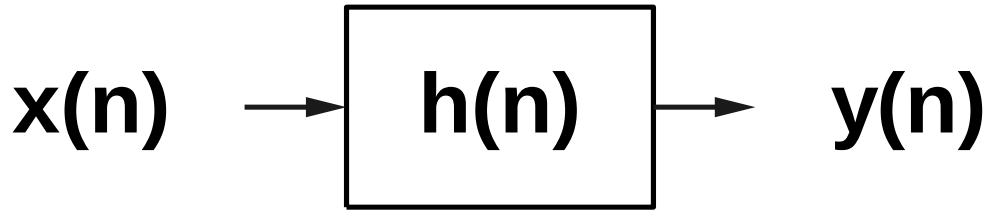
Ex. Multiplication is strong compared to addition

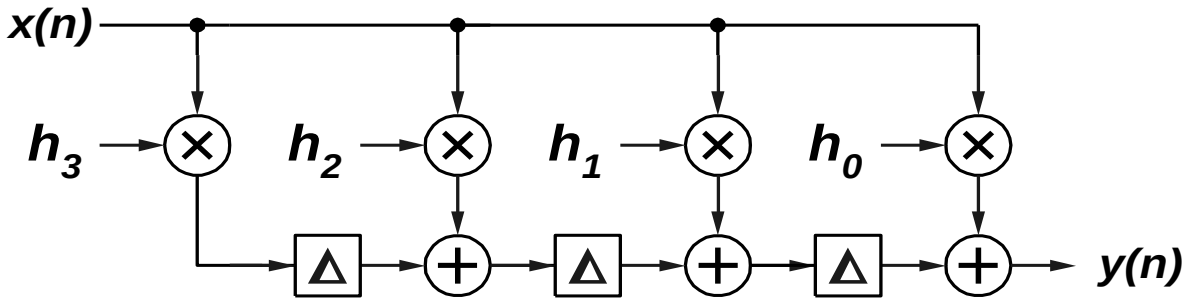
Strength Reduction

In parallel filters

Polyphase FIR Filter

$$Y(z) = H(z)X(z) = \sum_{n=0}^{N-1} h(n)z^{-n} \sum_{n=0}^{\infty} x(n)z^{-n}$$





Idea: Split to an L-parallel filter and reduce the strength

Split the input sequence into odds and evens

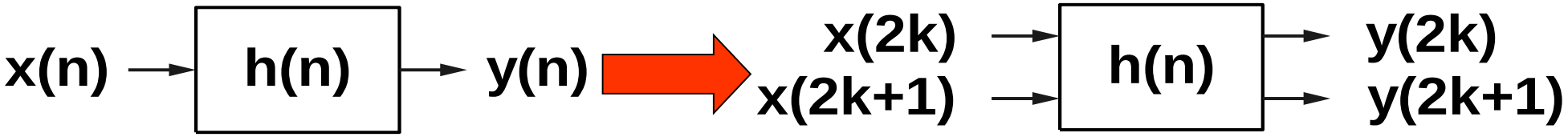
$$X(z) = x(0) + x(1)z^{-1} + x(2)z^{-2} + x(3)z^{-3} + x(4)z^{-4} + x(5)z^{-5} + \cdots$$

$$X(z) = x(0)z^{-0} + x(2)z^{-2} + x(4)z^{-4} + \cdots$$

$$+ z^{-1}\left(x(1) + x(3)z^{-2} + x(5)z^{-4} + \cdots \right)$$

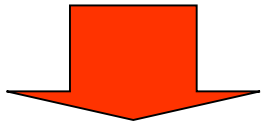
$$X(z) = X_0(z^2) + z^{-1}X_1(z^2)$$

$X(z^2)$ *are Z – transform of* $x(2k)$



Split the filter accordingly

$$H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3} + h(4)z^{-4} + h(5)z^{-5} + \cdots$$



$$H(z) = H_0(z^2) + z^{-1}H_1(z^2)$$

and

$$Y(z) = H(z)X(z) =$$

$$\left(H_0(z^2) + z^{-1}H_1(z^2)\right)\left(X_0(z^2) + z^{-1}X_1(z^2)\right)$$

$$\left(H_0(z^2) + z^{-1}H_1(z^2)\right)\left(X_0(z^2) + z^{-1}X_1(z^2)\right)$$

$$H_0(z^2)X_0(z^2) + z^{-2}H_1(z^2)X_1(z^2) +$$

$$z^{-1}\left(H_0(z^2)X_1(z^2) + H_1(z^2)X_0(z^2)\right)$$

$y(2k)$ $\quad Y_0(z^2) = X_0(z^2)\,H_0(z^2) \quad + \quad z^{-2}X_1(z^2)\,H_1(z^2)$

$y(2k+1)$ $\quad Y_1(z^2) = X_0(z^2)\,H_1(z^2) \quad + \quad X_1(z^2)\,H_0(z^2)$

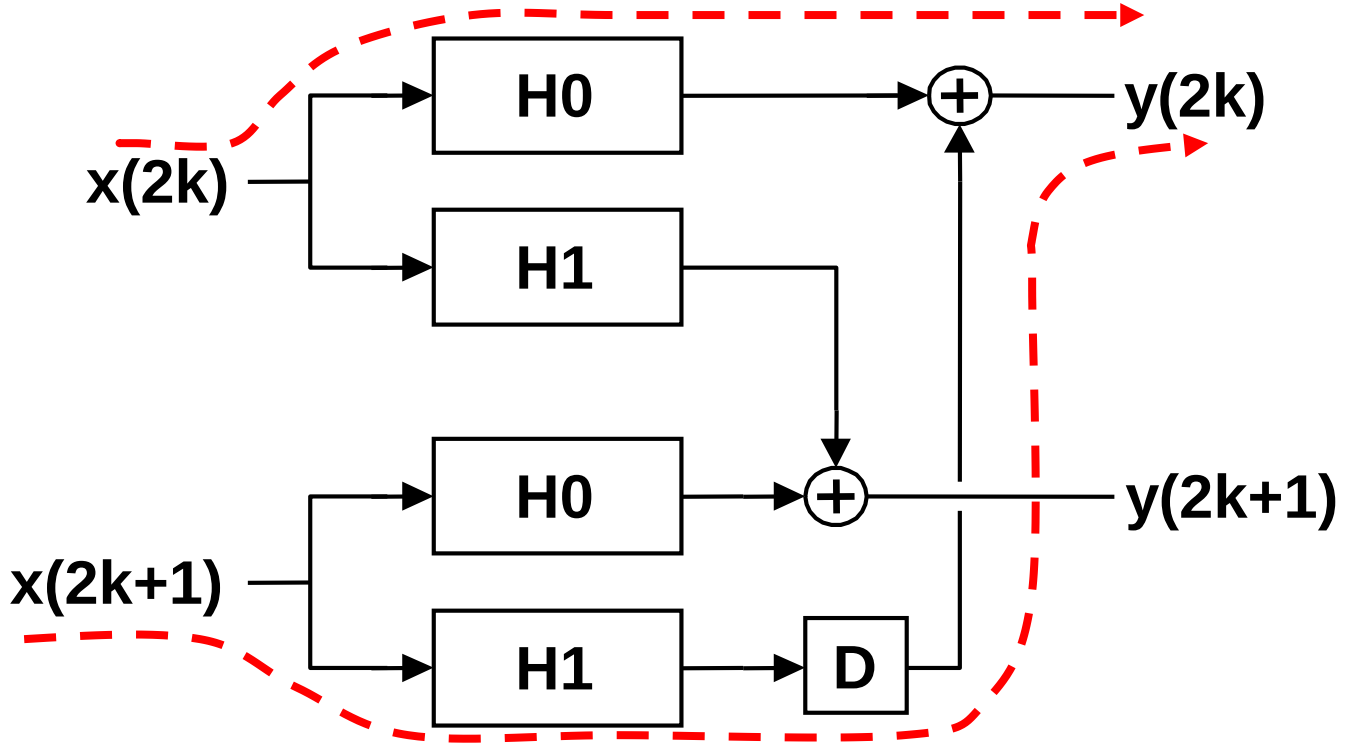
Polyphase Filter: 2–parallel filter

Split h(n) in two N/2 filters

H0: Even Coef.

H1: Odd Coef.

2N mults and 2(N-1) adds



$$Y_0(z^2) = X_0(z^2)\,H_0(z^2) + z^{-2}X_1(z^2)\,H_1(z^2)$$

$$Y_1(z^2) = X_0(z^2)\,H_1(z^2) + X_1(z^2)\,H_0(z^2)$$

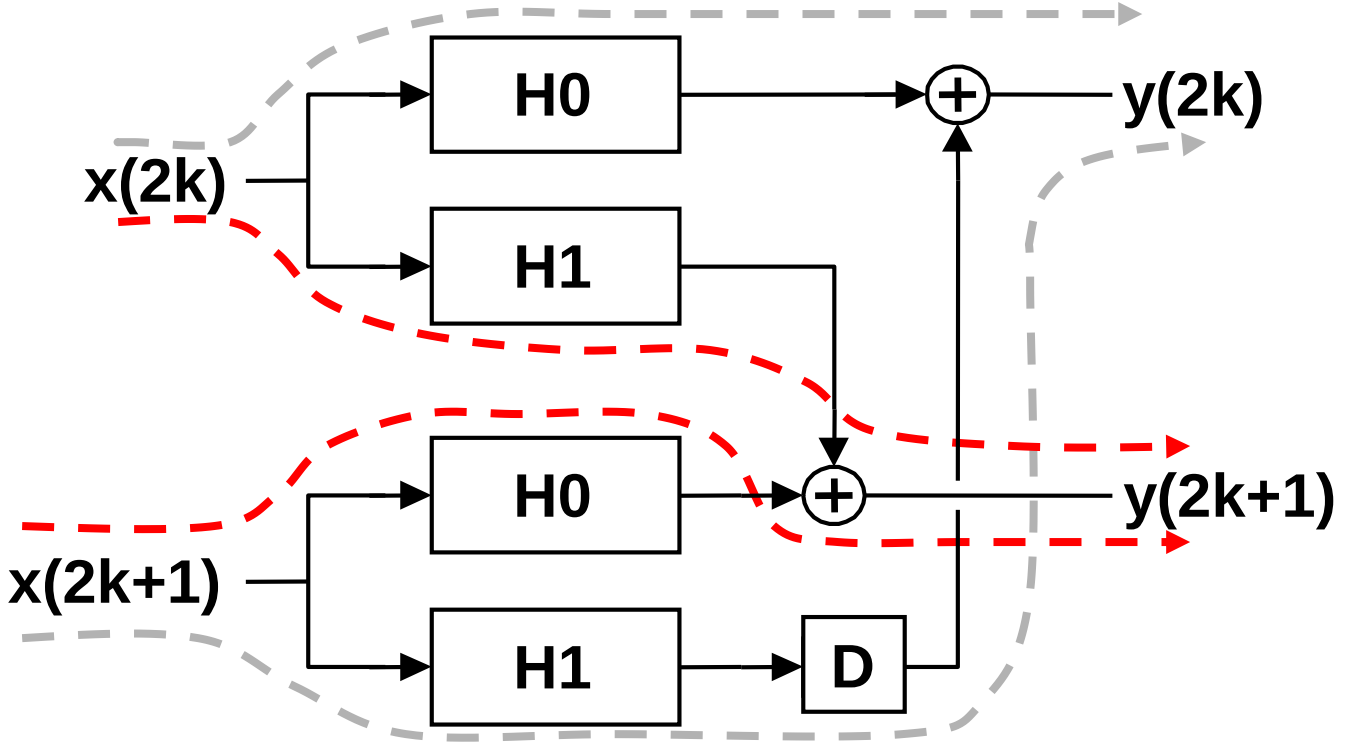
Polyphase Filter: 2–parallel filter

Split h(n) in two N/2 filters

H0: Even Coef.

H1: Odd Coef.

2N mults and 2(N-1) adds

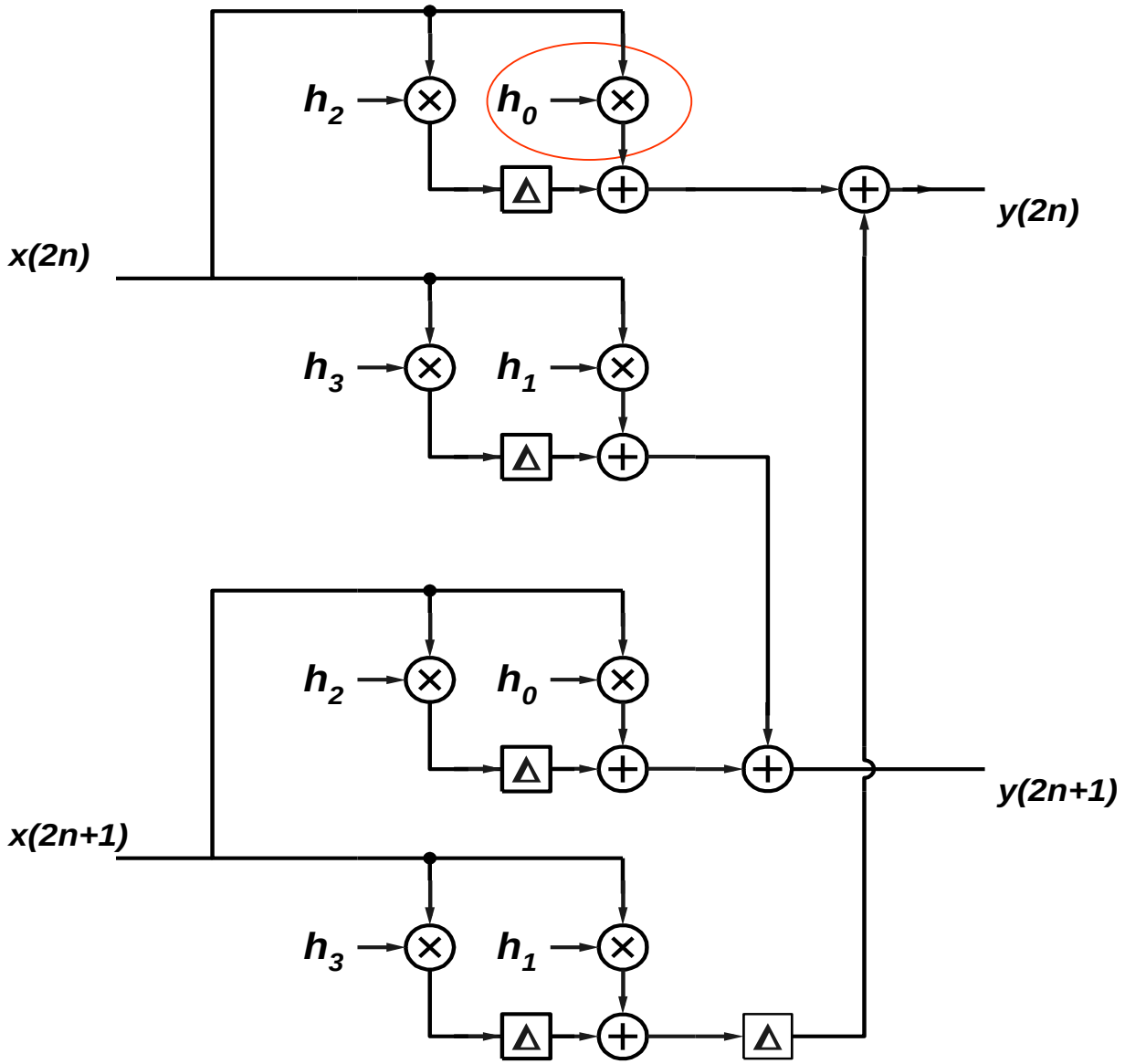


$$Y_0(z^2) = X_0(z^2)\,H_0(z^2) + z^{-2}X_1(z^2)\,H_1(z^2)$$

$$Y_1(z^2) = X_0(z^2)\,H_1(z^2) + X_1(z^2)\,H_0(z^2)$$

Example

$y_0 = x_0 h_0$

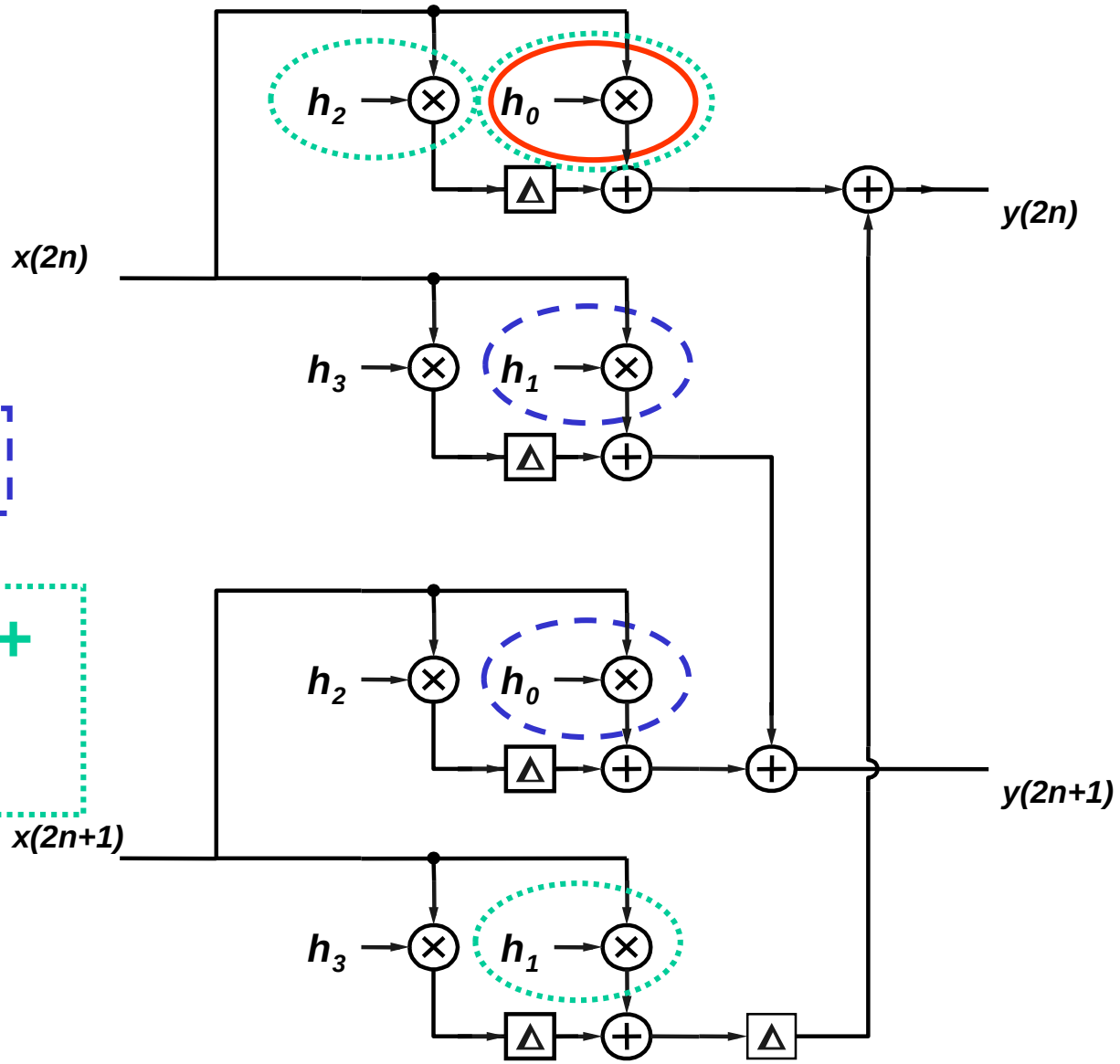


Example

$$y_0 = x_0 h_0$$

$$y_1 = x_0 h_1 + x_1 h_0$$

$$y_2 = x_0 h_2 + x_1 h_1 + x_2 h_0$$



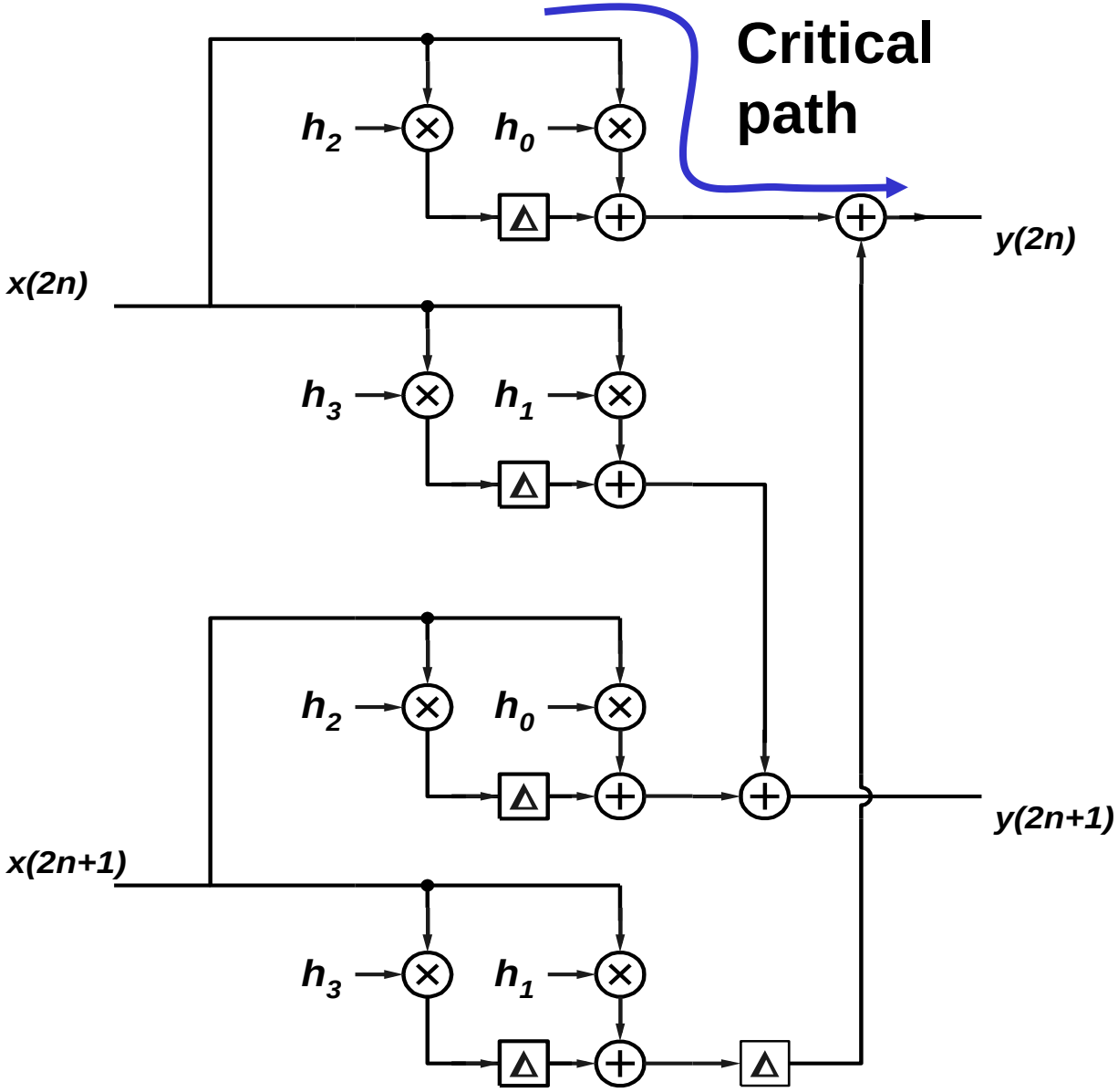
Example

No Strength Reduction

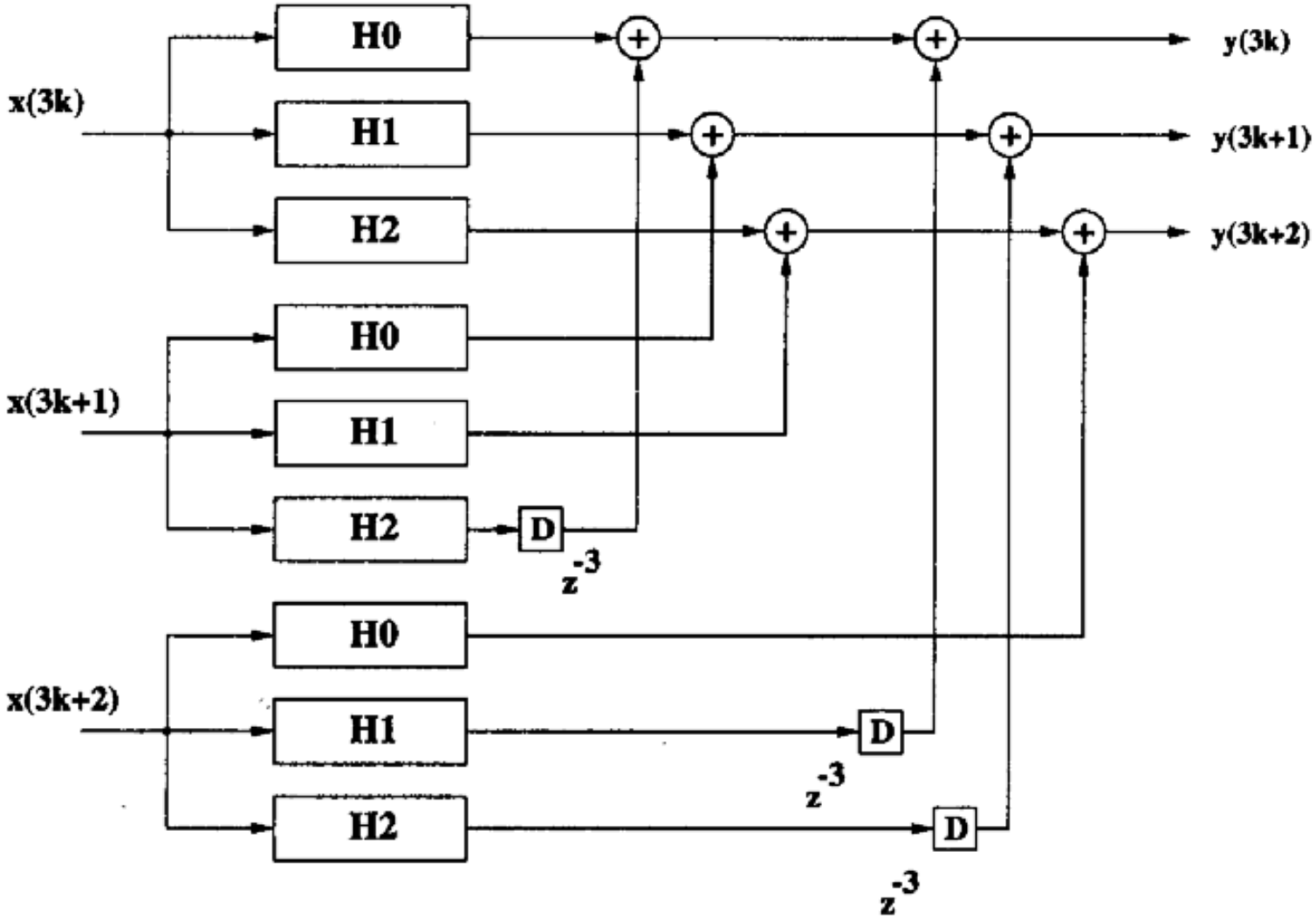
2N multiplications per two samples

Power can be saved!

Slightly increased critical path but 2 sample periods



Ex. A 3-phase Polyphase Filter



Polyphase Filters

Filter	Mult	Add	Subfilters
FIR	N	N-1	1
2-Parallel	2 N	2(N-1)	4
3-Parallel	3 N	3(N-1)	9
L-Parallel	L N	L(N-1)	L^2

However, they give L samples each cycle

Although the polyphase formula does not reduce the parallell filter complexity, it can be exploited to derive fast parallel FIR filters!

Polyphase Filter

$$H_0 = \{h_0, h_2, h_4, \ldots\}$$

$$H_1 = \{h_1, h_3, h_5, \ldots\}$$

$$H_0 + H_1 = \{h_0 + h_1, h_2 + h_3, h_4 + h_5, .$$

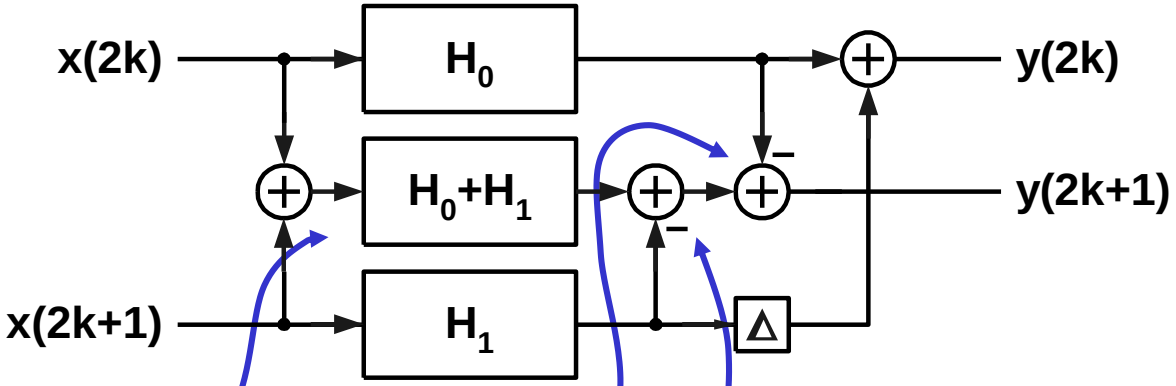
$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = H_1 X_0 + H_0 X_1 \end{cases}$$

…rewrite into…

$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = (H_0 + H_1)(X_0 + X_1) - H_0 X_0 - H_1 X_1 \end{cases}$$

Fast FIR Filters

$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = H_1 X_0 + H_0 X_1 \end{cases}$$

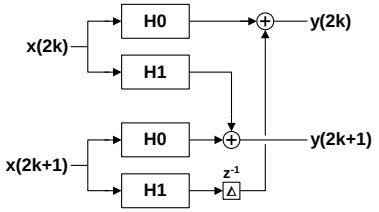


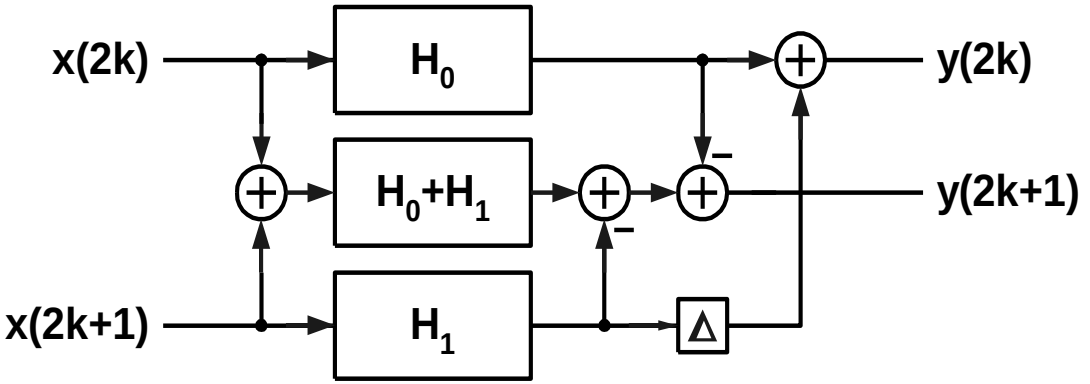
$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = (H_0 + H_1)(X_0 + X_1) - H_0 X_0 - H_1 X_1 \end{cases}$$

Strength Reduction - Fast FIR Filters

Poly phase FIR

$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = H_1 X_0 + H_0 X_1 \end{cases}$$

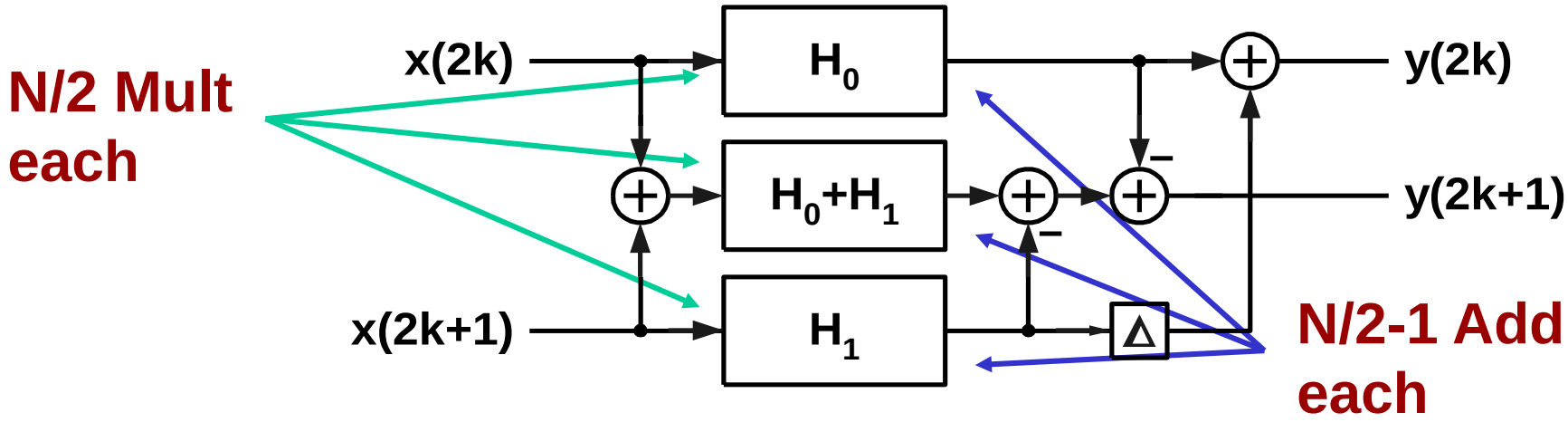




Fast FIR

$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = (H_0 + H_1)(X_0 + X_1) - H_0 X_0 - H_1 X_1 \end{cases}$$

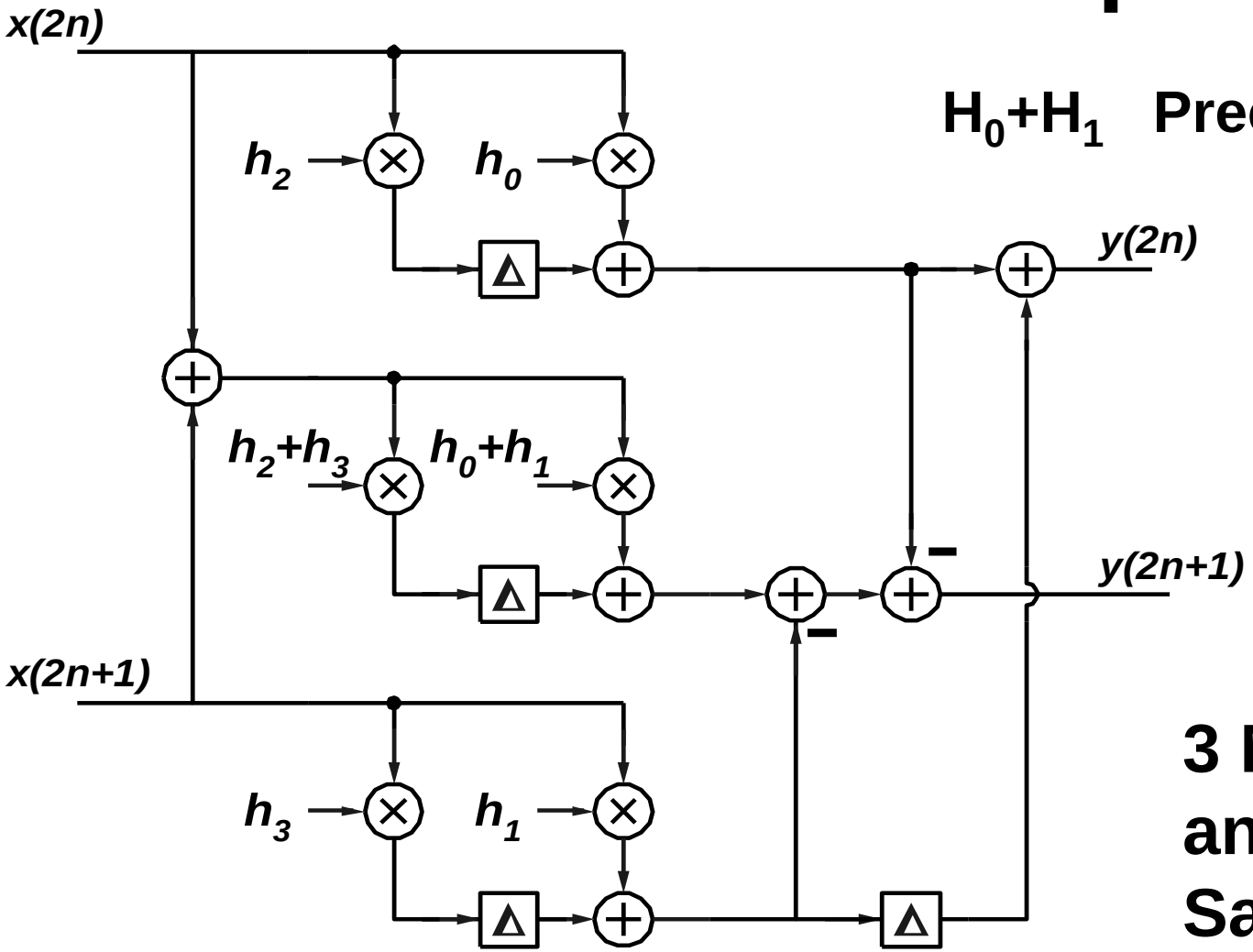
Reduced Complexity



Total

- **3N/2 = 1.5N mults compared to 2N**
- **3(N/2 - 1) + 4 = 1.5N + 1 adds compared to 2(N - 1)**

Fast FIR Example

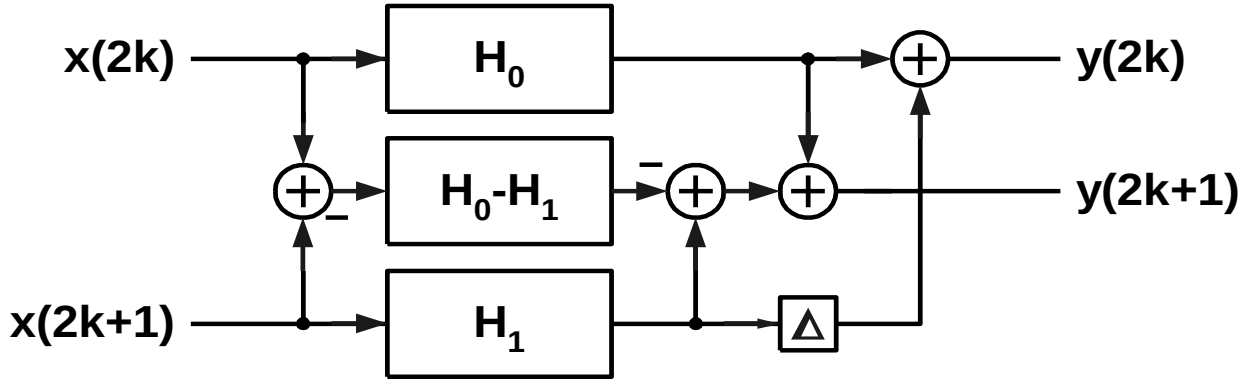


H_0+H_1 Precomputed

3 Multiplications and 3.5 Add per Sample

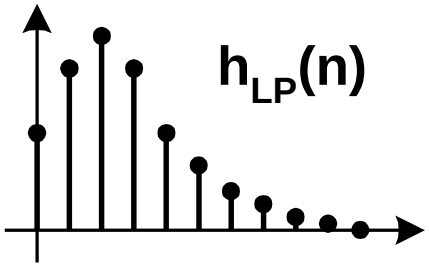
Alternative Structure

$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = H_1 X_0 + H_0 X_1 \end{cases}$$



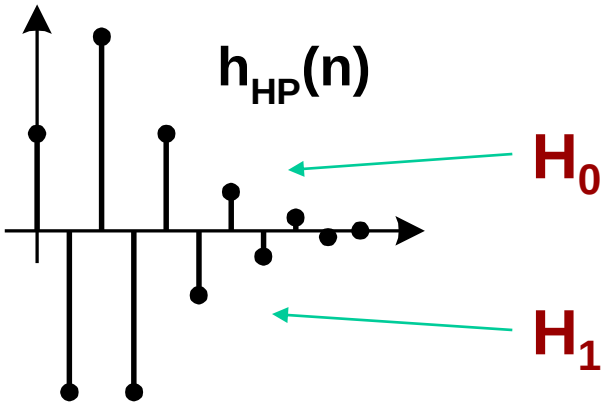
$$\begin{cases} Y_0 = H_0 X_0 + z^{-2} H_1 X_1 \\ Y_1 = H_0 X_0 + H_1 X_1 - (H_0 - H_1)(X_0 - X_1) \end{cases}$$

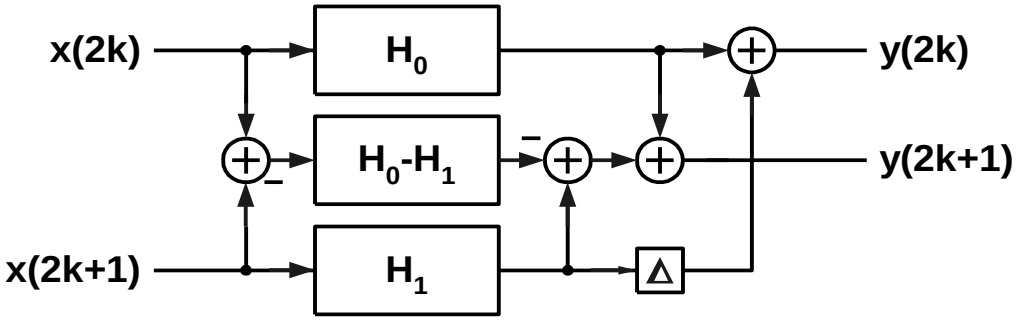
Alternative Structure



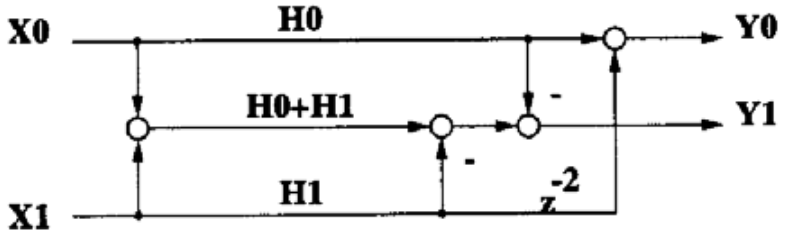
H_0-H_1 might be better for lowpass and H_0+H_1 might be better for highpass filters

- **Saves power**



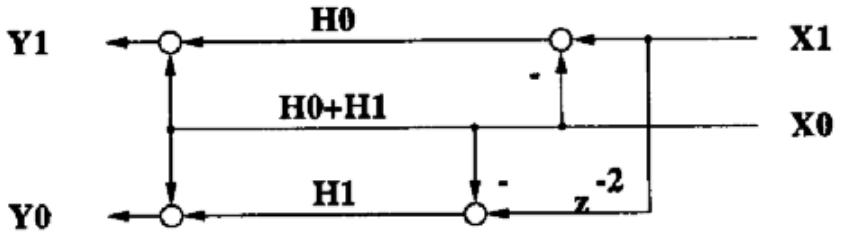


What happens if we transpose?



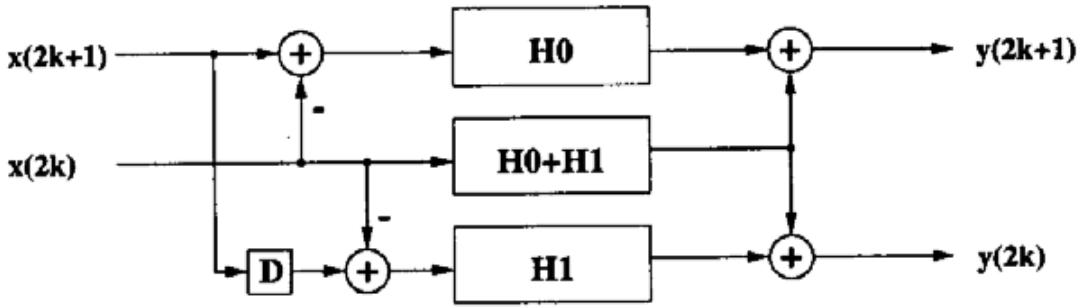
(a)

Transposition: Reverse the edges and interchange in- outputs

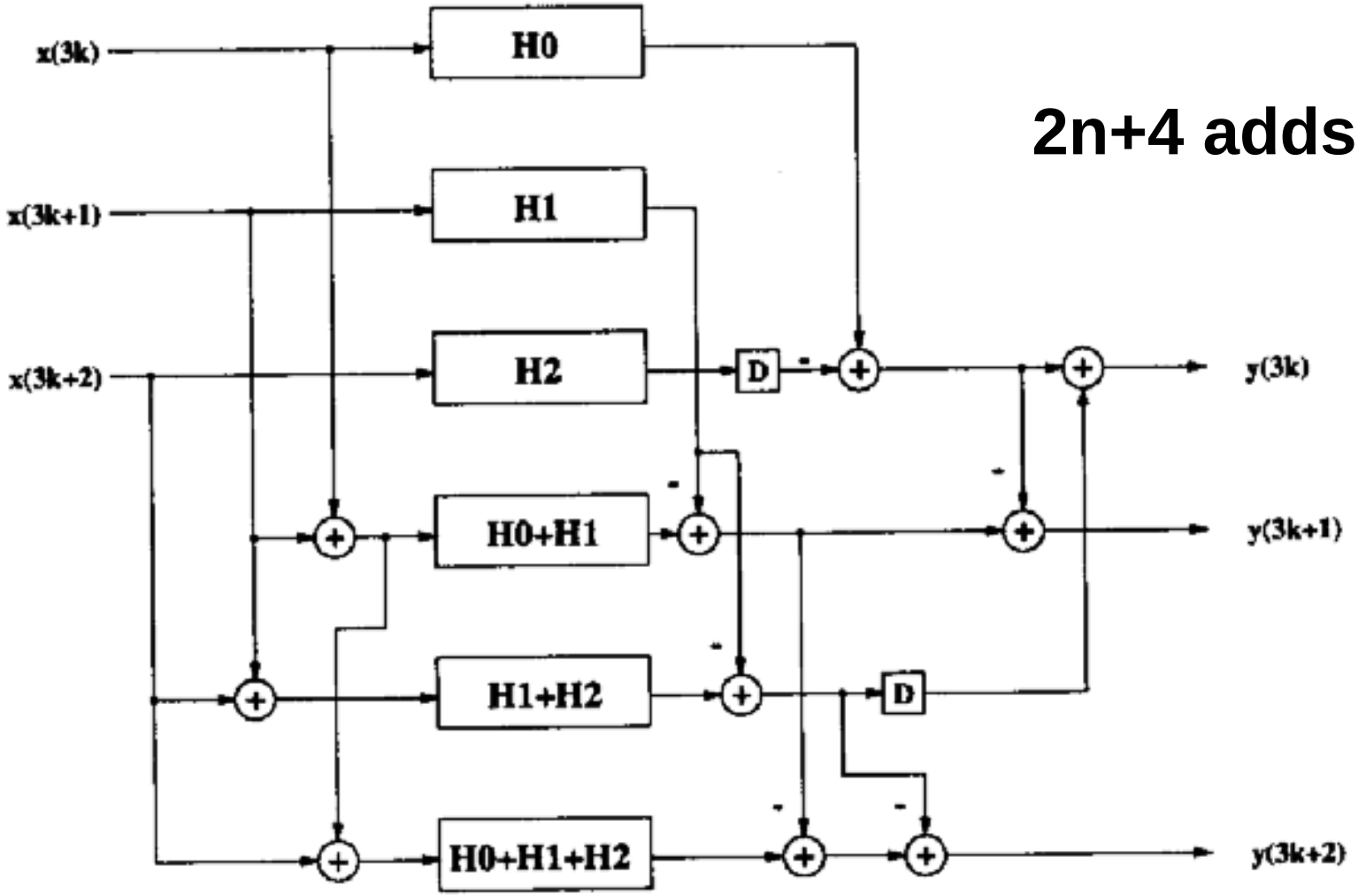


(b)

Transposition does not change the complexity of the filter algorithm.

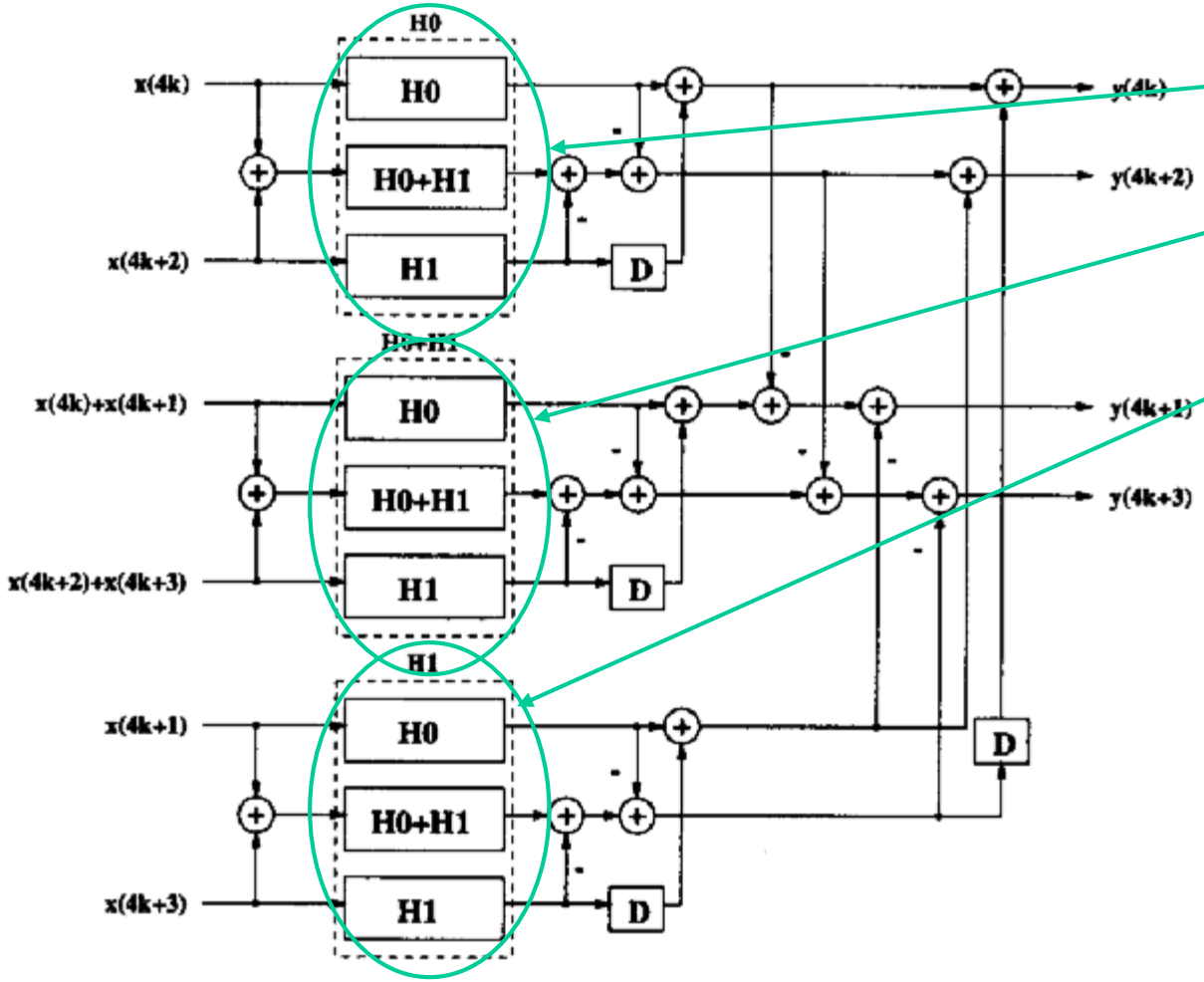


Ex. Reduced Complexity 3-parallel



2n+4 adds

Reduced Complexity 4-parallel



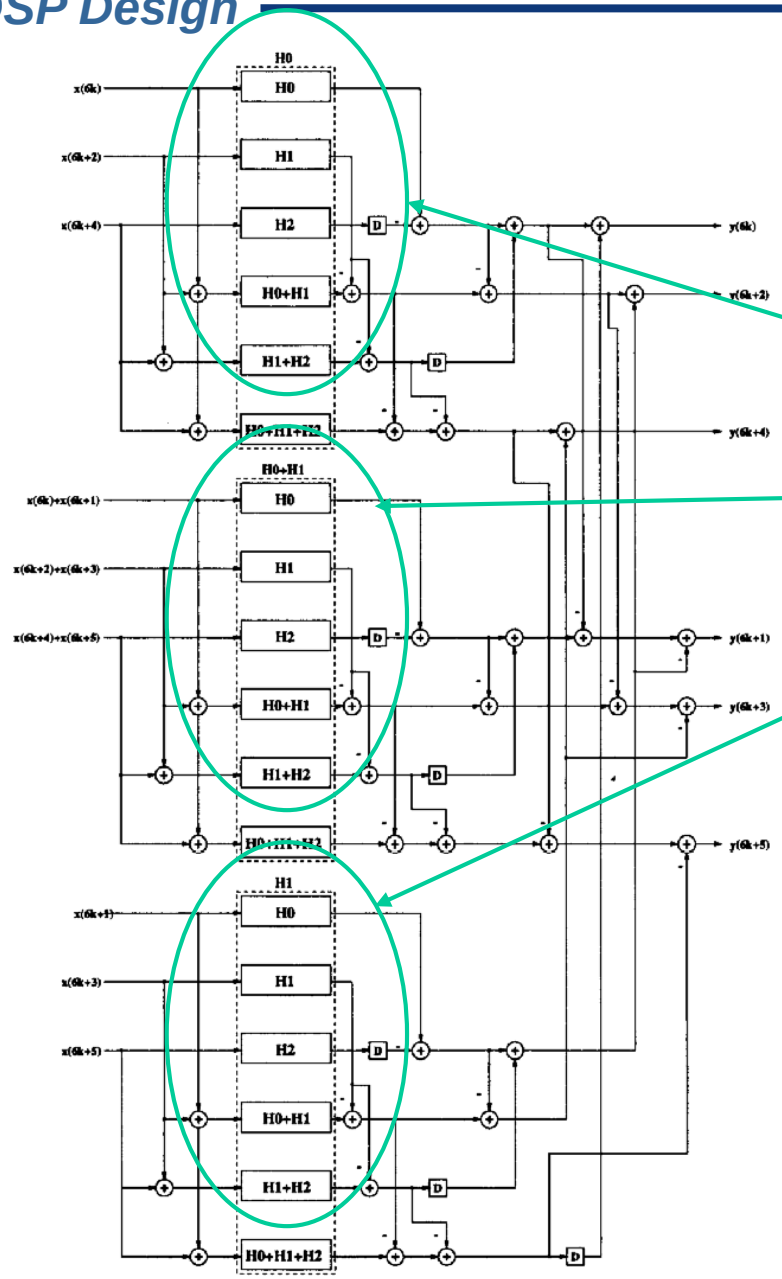
H_0

H_0+H_1

H_1

Two Cascaded 2-parallel filters

Fast FIR 6-parallel



3-parallel filters in a 2 parallel structure

Polyphase Filters

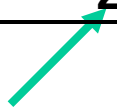
Filter	Traditional Mults	Traditional Adds	Fast FIR Mults	Fast FIR Adds
FIR	N	N-1		
2-Parallel	2 N	2(N-1)	1.5N	1.5N+1
3-Parallel	3 N	3(N-1)	2N	2N+4
4-Parallel	4 N	4(N-1)	9N/4	20+9(N/4-1)

(n×m)-parallel filter with n=2, m=2

Comparison Polyphase Filters

N=8 Filter	Traditional Mults	Adds	Reduced Complexity Mults	Adds
FIR	8	7		
2-Parallel	16	14	12	11
3-Parallel	24	21	16	20
4-Parallel	32	28	18	29

N=1000 Filter	Traditional Mults	Adds	Reduced Complexity Mults	Adds
FIR	1000	999		
2-Parallel	2000	1998	1500	1499
3-Parallel	3000	2997	2000	2004
4-Parallel	4000	3996	2250	2261



Reduced Complexity: 44%

Strength Reduction

Various Techniques

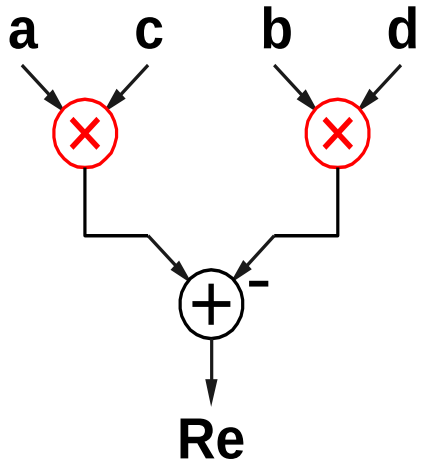
Complex Multiplication

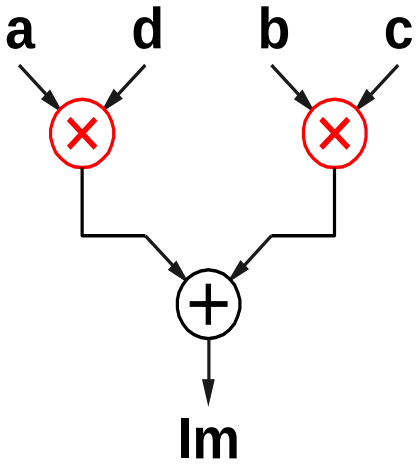
$$(a + jb)(c + jd) =$$

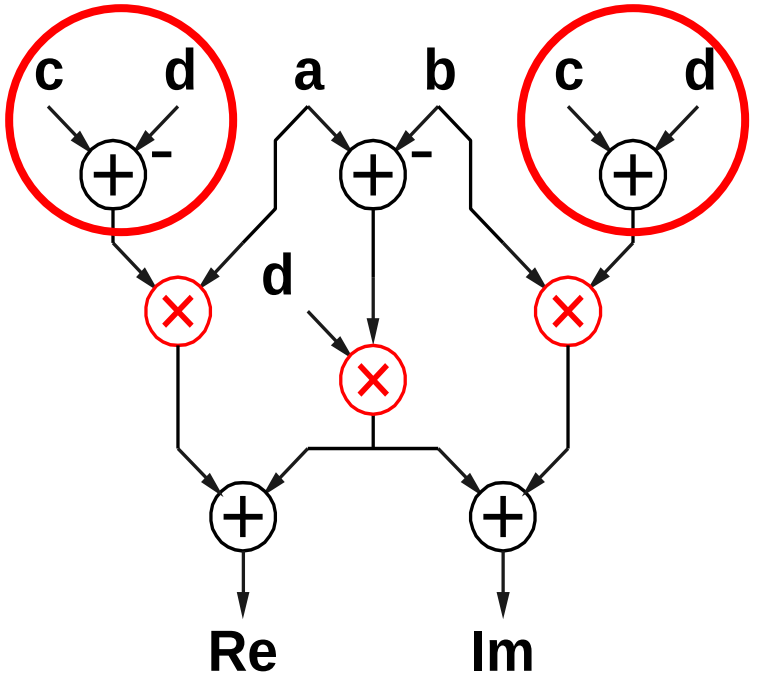
$$\mathrm{Re} = ac - bd = a(c - d) + d(a - b)$$

$$\mathrm{Im} = ad + bc = b(c + d) + d(a - b)$$

Coefficients
(Pre-calculated)



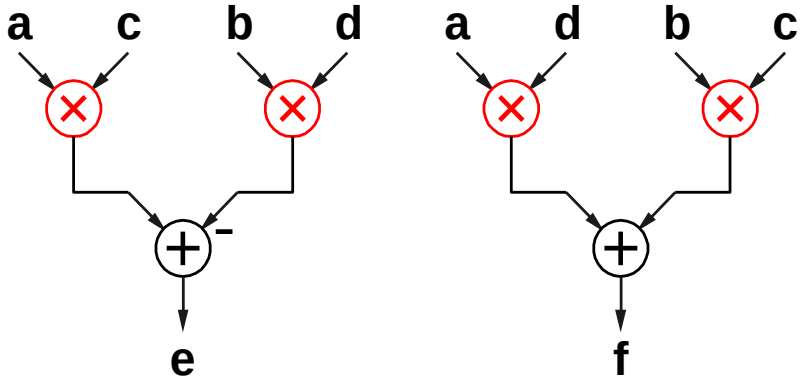




Complex Multiplication

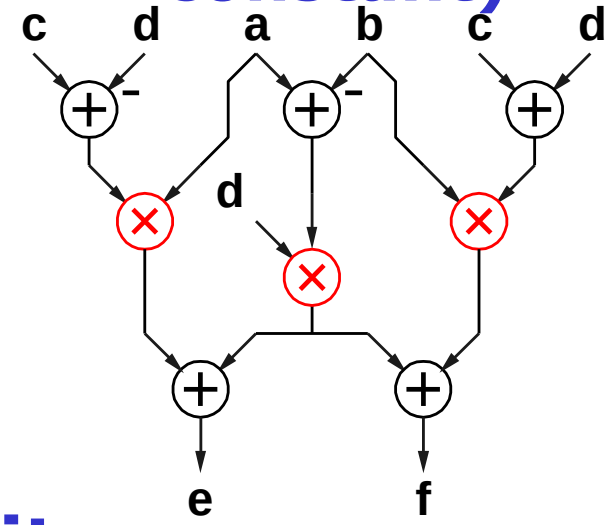
4 multiplications

2 additions



3 mult and 5 add

(3 add if c & d constant)



- **Less switched capacitance**
- **However, increased critical path**

Arithmetic shift

An **arithmetic shift** is a shift operator (also known as a **signed shift)**. The two basic types are the **arithmetic left shift** and the **arithmetic right shift**.

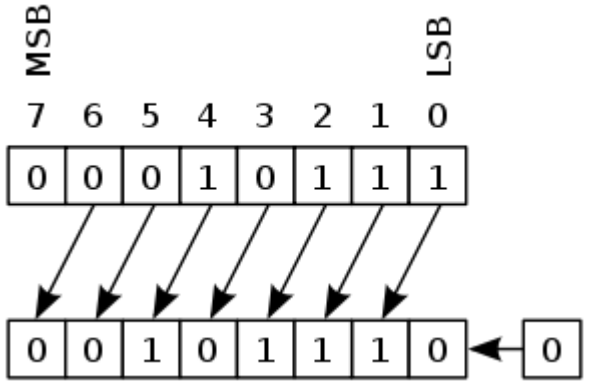
Arithmetic shifts can be useful as efficient ways of performing multiplication or division of signed integers by powers of two.

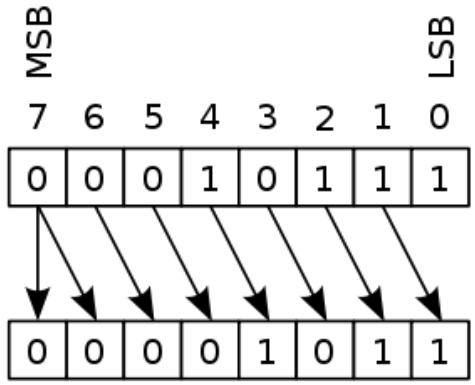
Shifting left by *n* bits on a signed or unsigned binary number has the effect of multiplying it by 2^n.

Shifting right by *n* bits on a two's complement *signed* binary number has the effect of dividing it by 2^n, but it always rounds down (towards negative infinity).

This is different from the way rounding is usually done in signed integer division (which rounds towards 0).

This discrepancy has led to bugs in more than one compiler!!





Source: Wikipedia

Single-Constant Multiplication

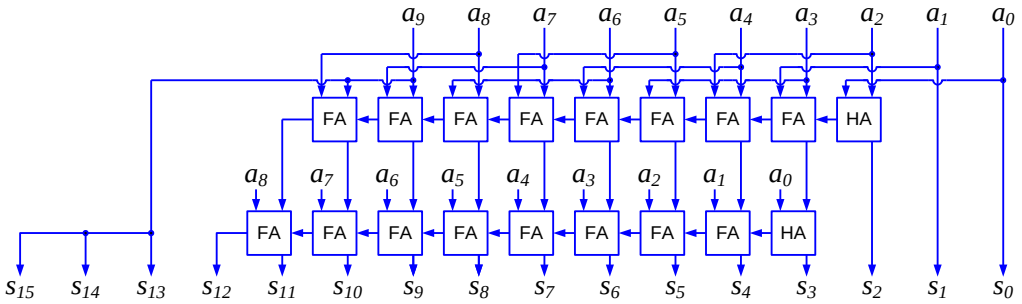
Idea: Find new structures using additions/subtractions and shifts

(multiplications by 2^n)

Example:

$$C_{10} = 00001101_2 = 13_{10}$$

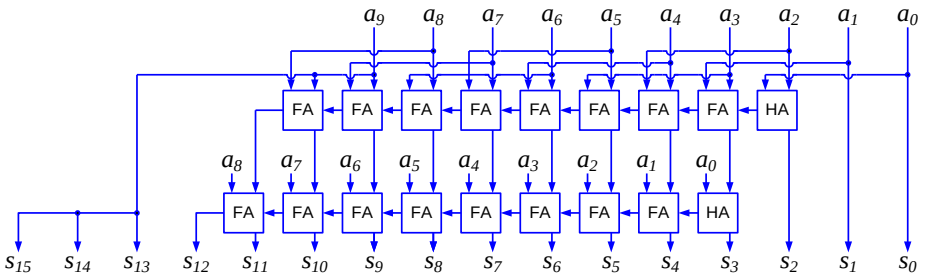
Coefficient from the Hilbert filter



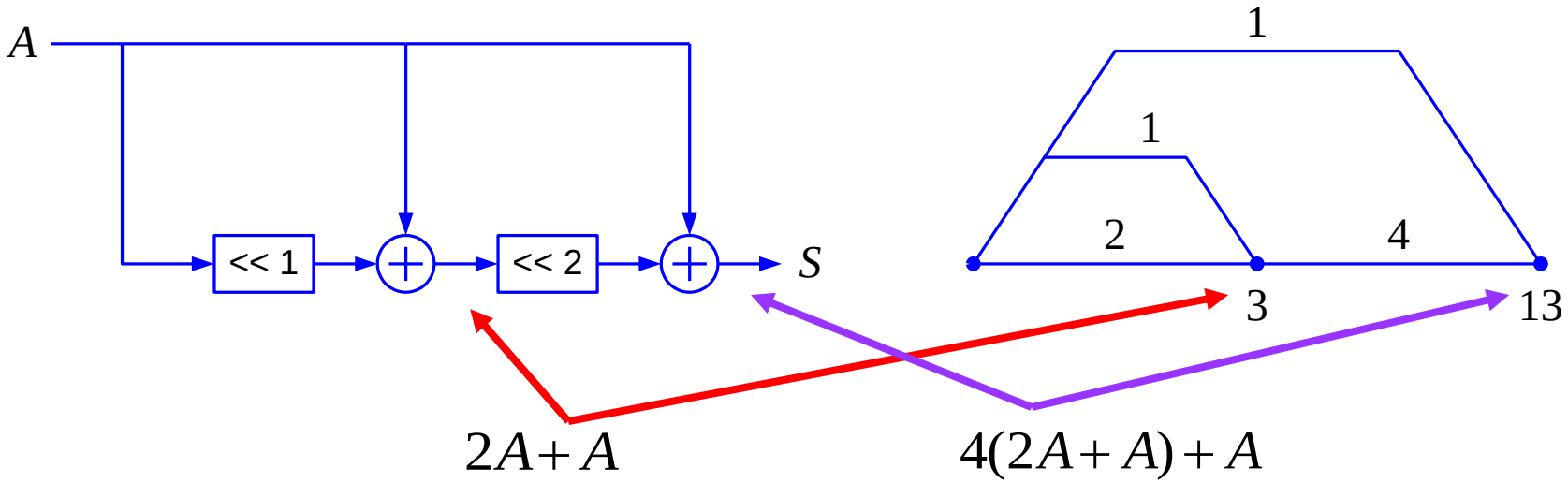
Single-Constant Multiplication

$$C_{10} = 00001101_2 =$$

$$= 1+4+8 = 1+4(2+1) = 13$$



Graph Representation (Not SFG/DFG!)

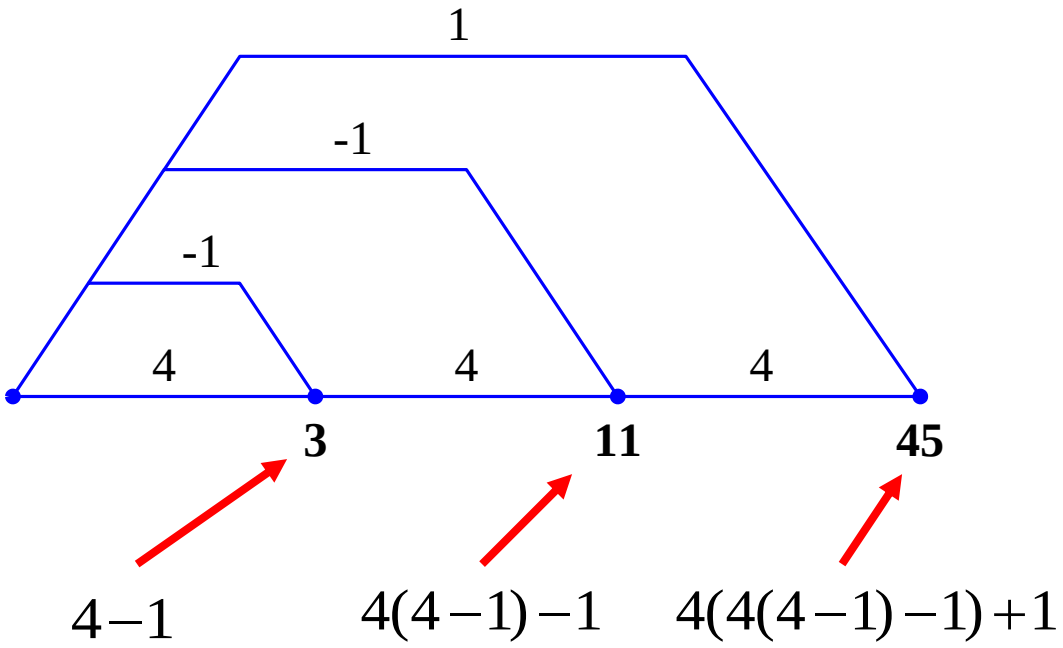


Single-Constant Multiplication

Example (CSD)

$$C = 45 = 64 - 16 - 4 + 1 = 10\bar{1}0\bar{1}01$$

$$C = 45 = 4(4(4-1)-1)+1$$



Single-Constant Multiplication

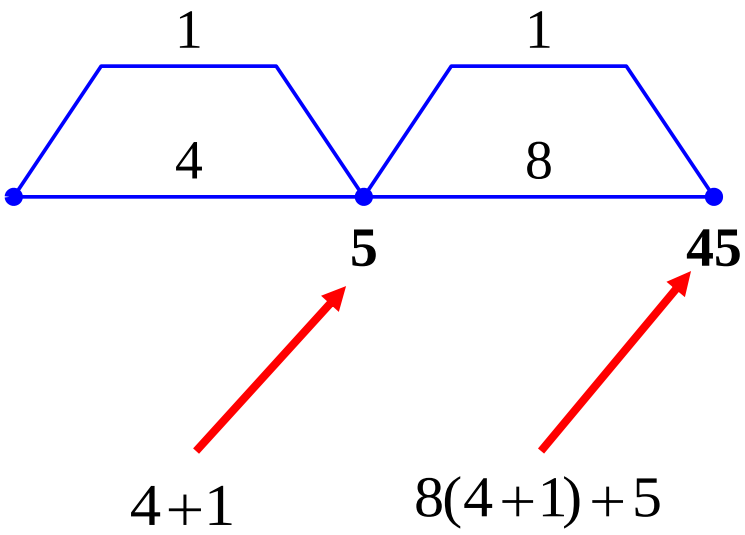
CSD

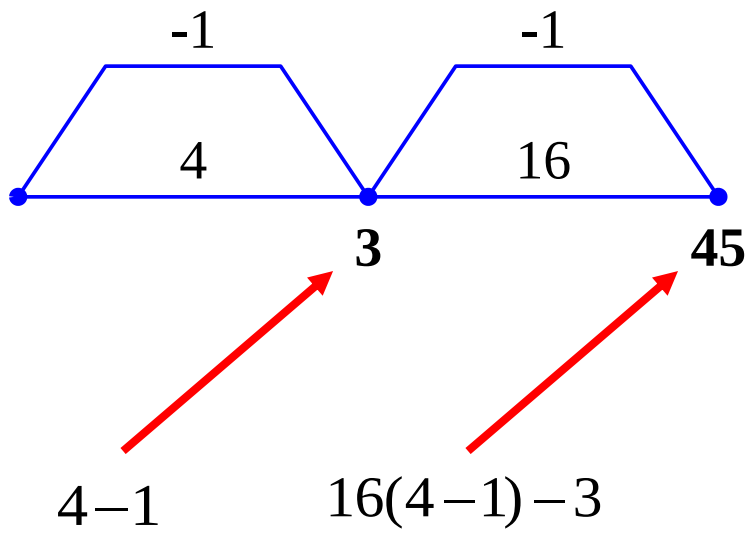
$$C = 64 - 16 - 4 + 1 = 45$$

However:

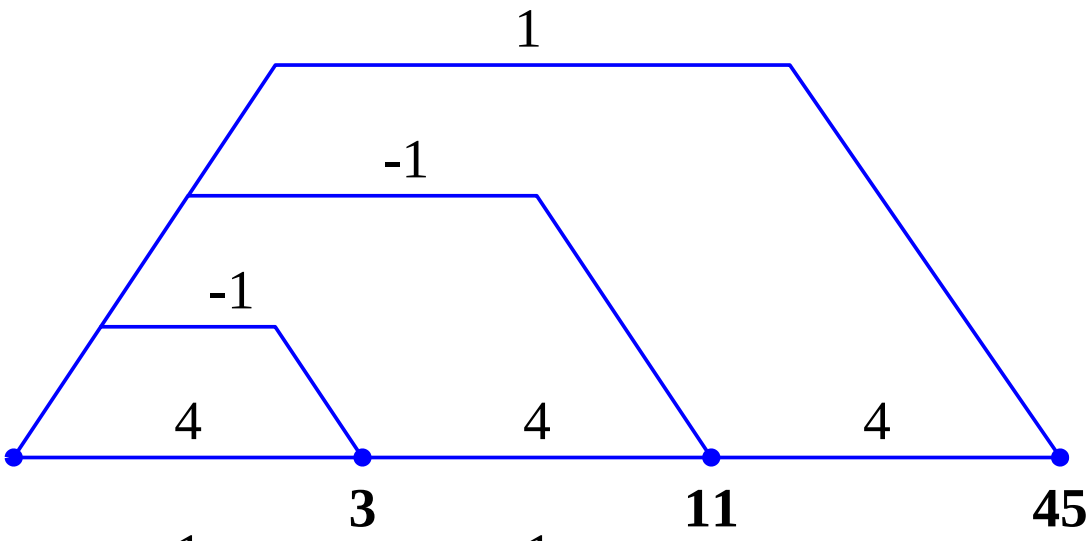
$$C = 5 \times 9 = (1+4)(1+8) = 45$$

$$C = 3 \times 15 = (4-1)(16-1) = 45$$

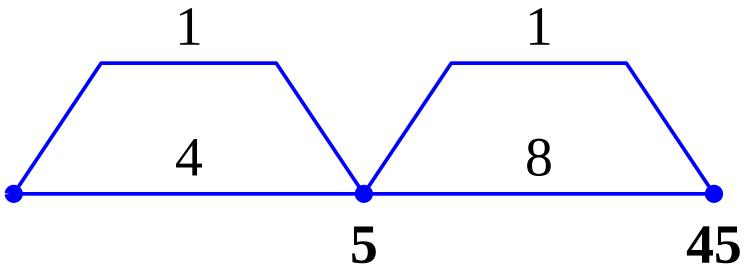




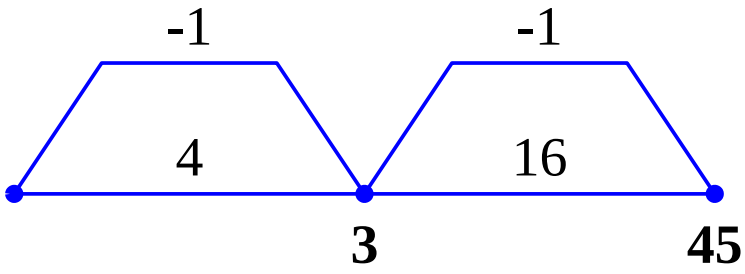
Single-Constant Multiplication



3 Add:s



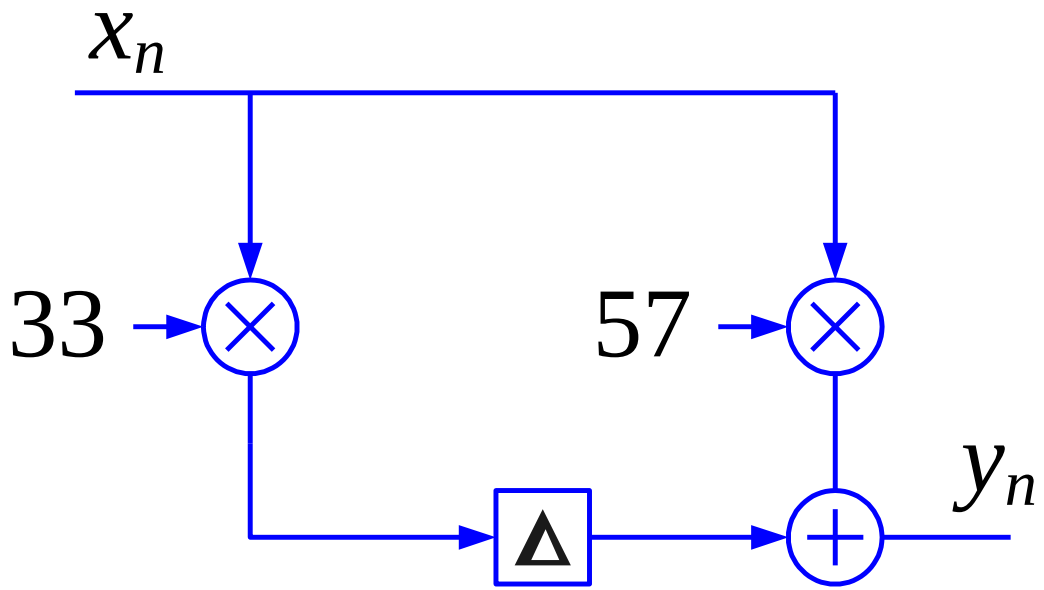
2 Add:s



2 Sub:s

Multiple-Constant Multiplication (MCM)

Consider the coefficient set $C = \{33, 57\}$

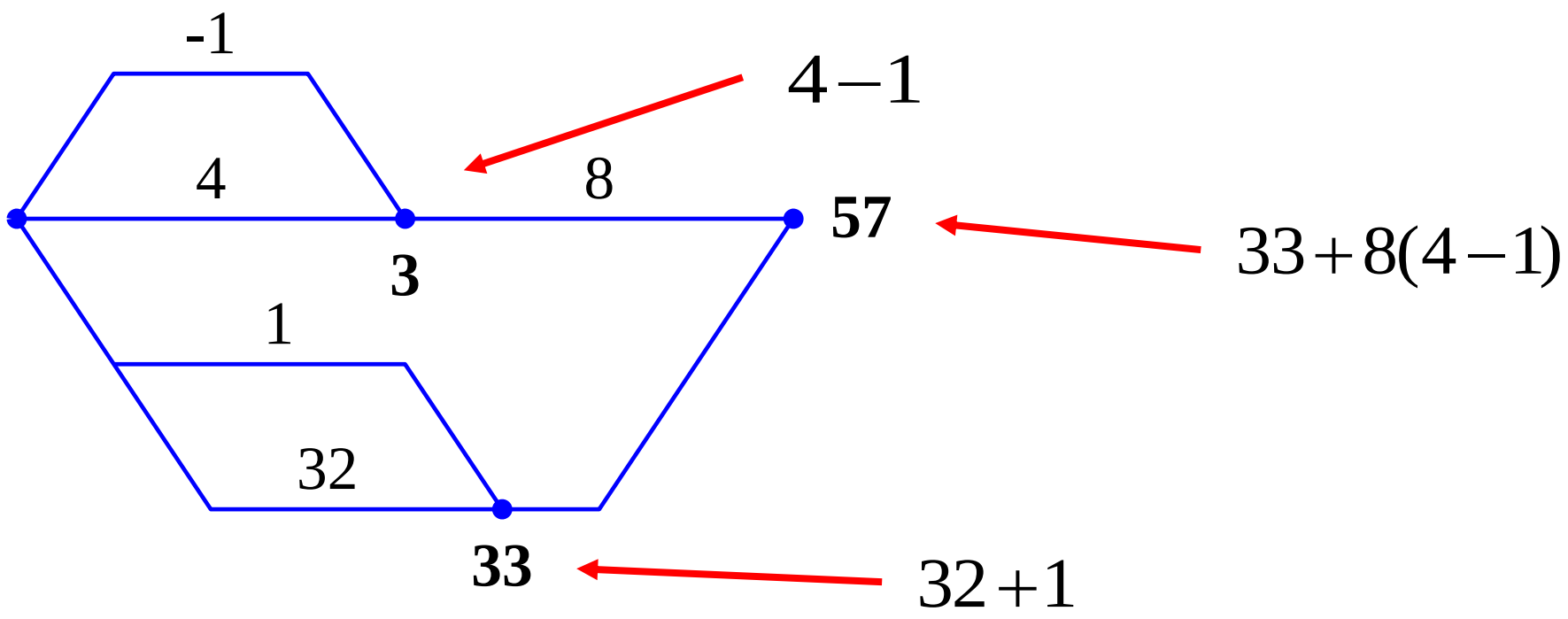


Only transposed form

Multiple-Constant Multiplication (MCM)

$C = \{33, 57\}$

$$C_0 = 32 + 1 = 33 \qquad C_1 = C_0 + 24 = 33 + 8(4-1) = 57$$



To find optimal MCM:s is a complex problem

Multiple-Constant Multiplication (MCM)

The Hilbert Filter (Transposed)

$C = \{13, 21, 64\};$

Add 3 to the set

$C = \{3, 13, 21, 64\}$

or

add 5 to the set

$C = \{5, 13, 21, 64\}$

Multiple-Constant Multiplication (MCM)

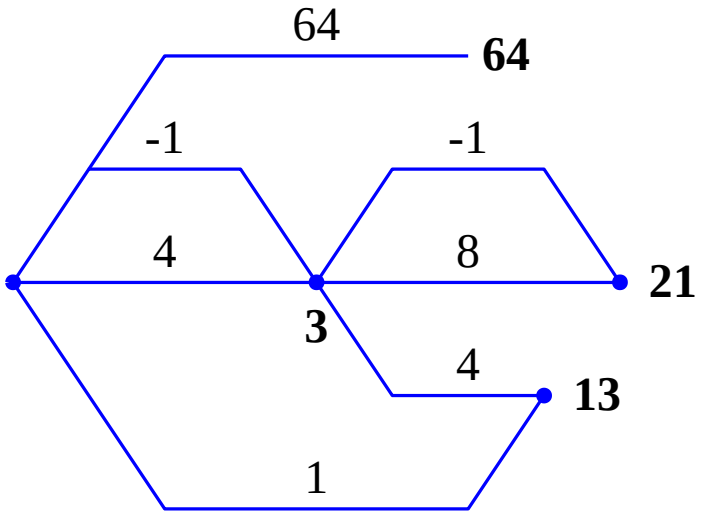
$C_i = \{3, 13, 21, 64\}$

$C_0 = 4 - 1 = 3$

$C_1 = 4(4-1) + 1 = 13$

$C_2 = (4-1)(8-1) = 21$

$C_3 = 64$



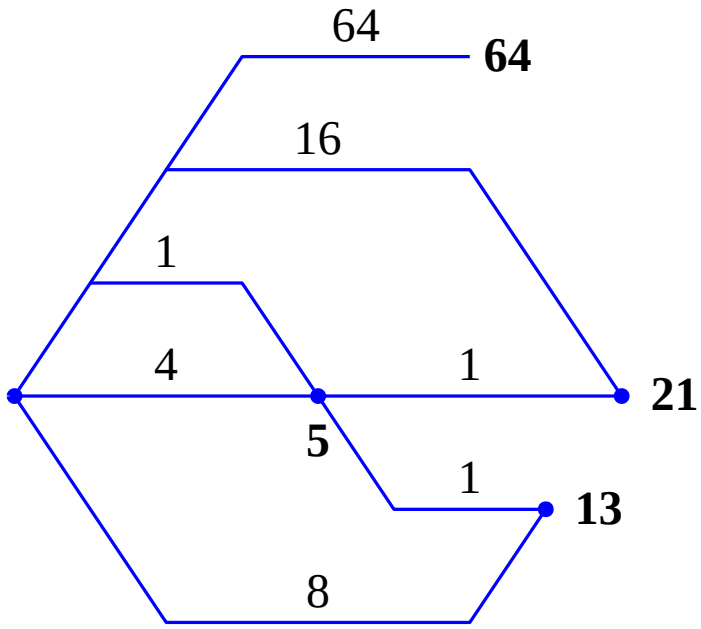
$C_i = \{5, 13, 21, 64\}$

$C_0 = 4 + 1 = 5$

$C_1 = (4+1) + 8 = 13$

$C_2 = (4+1) + 16 = 21$

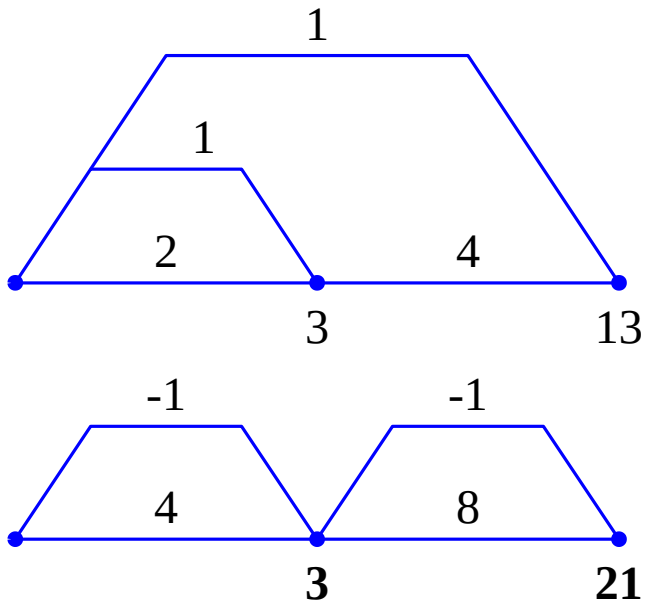
$C_3 = 64$



Multiple-Constant Multiplication (MCM)

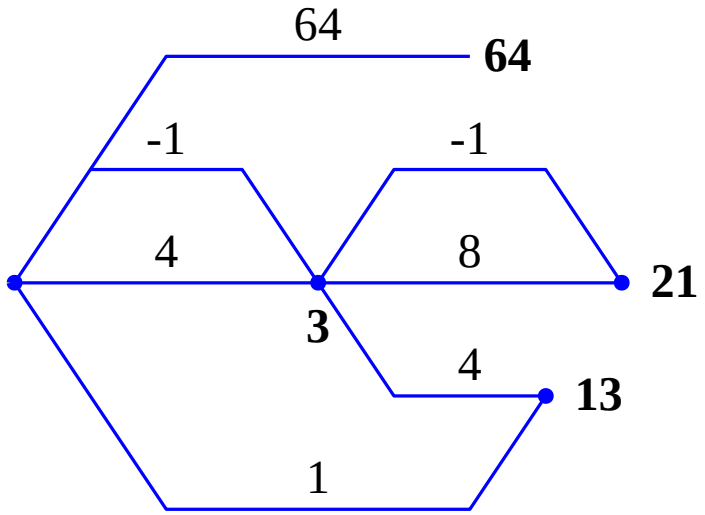
Single-Constant Multiplication:

4 Add:s



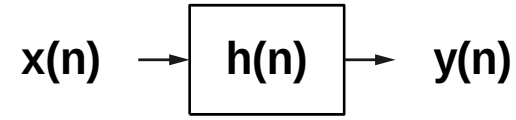
Multiple-Constant Multiplication:

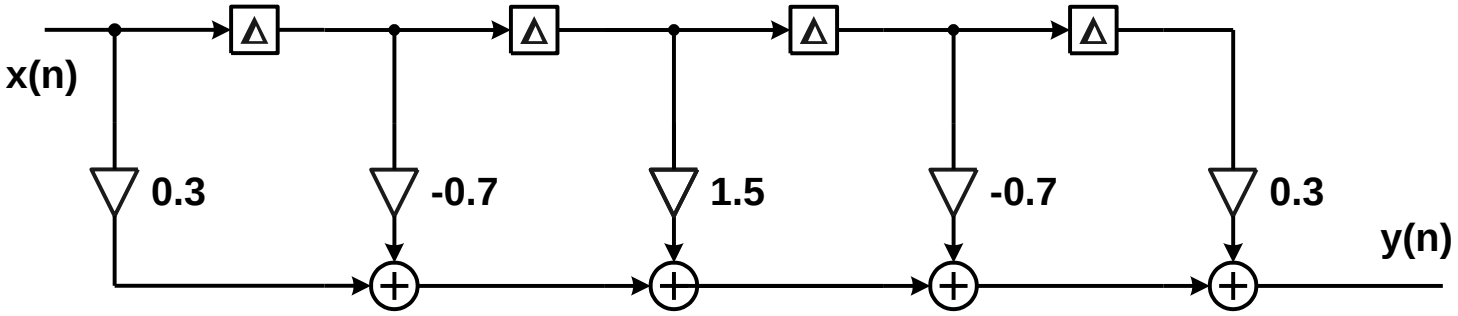
3 Add:s



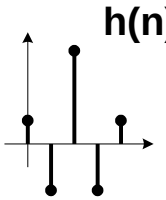
Large savings in long filters!

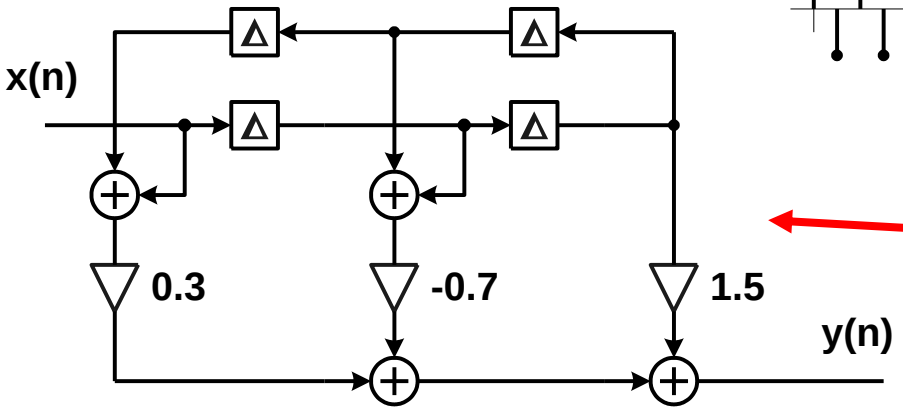
Linear Phase FIR Filter





5 multiplications 4 additions

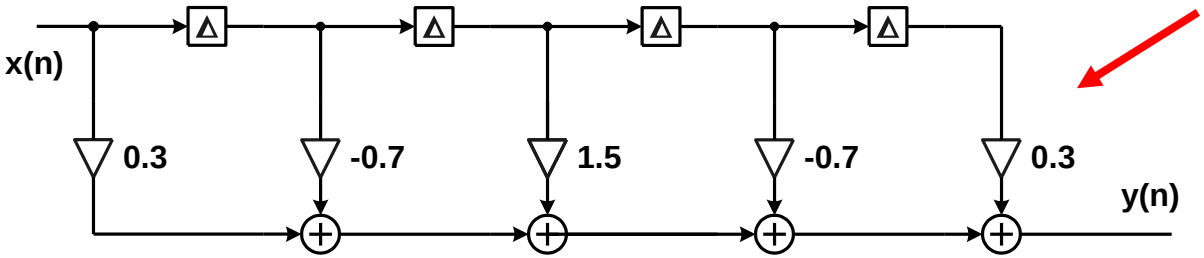




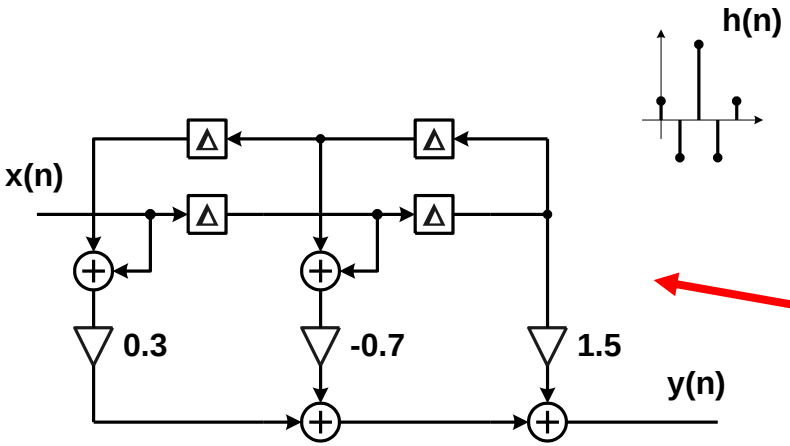
3 multiplications 4 additions

Shorter critical path, as well!

Linear Phase FIR Filter



n multiplications
n-1 additions

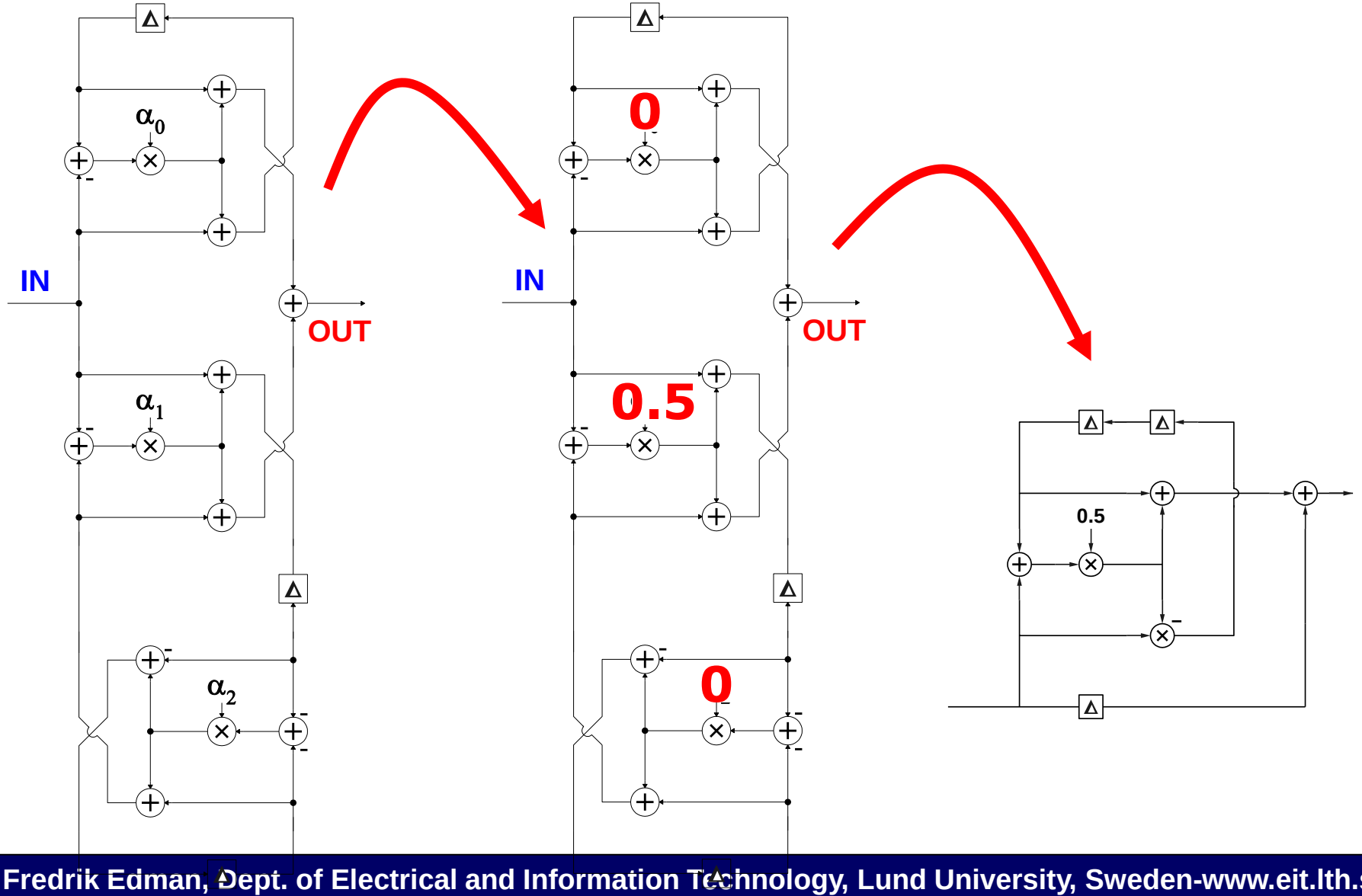


$\frac{n+1}{2}$ multiplications (n odd)

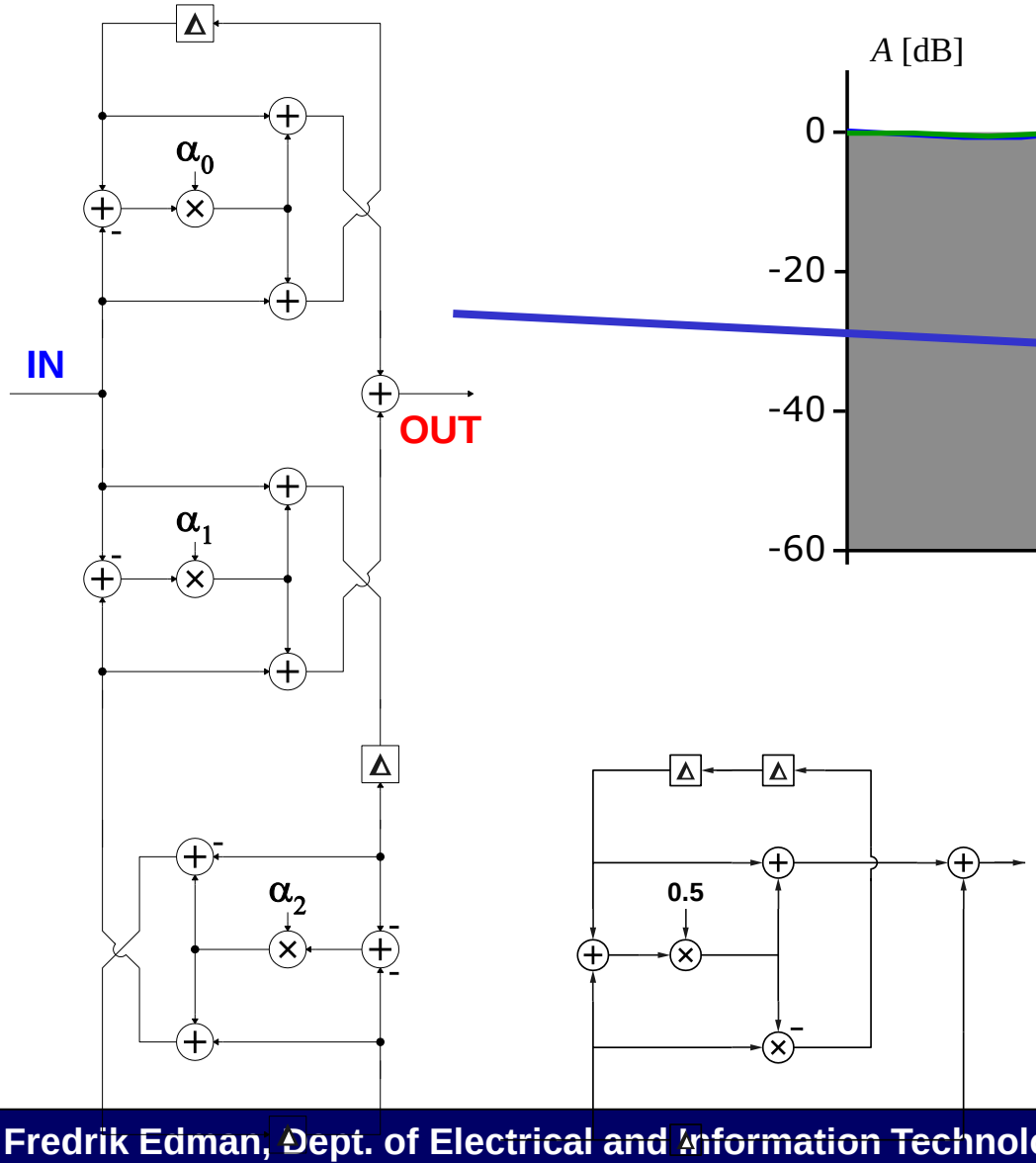
$\frac{n}{2}$ multiplications (n even)

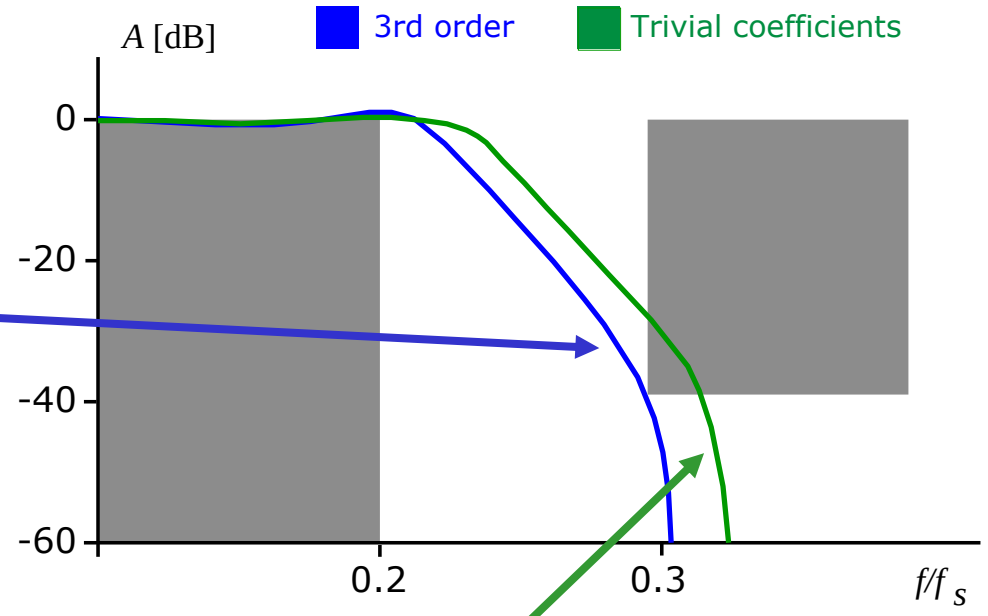
n-1 additions

IIR Filter: Try with Trivial Coefficients



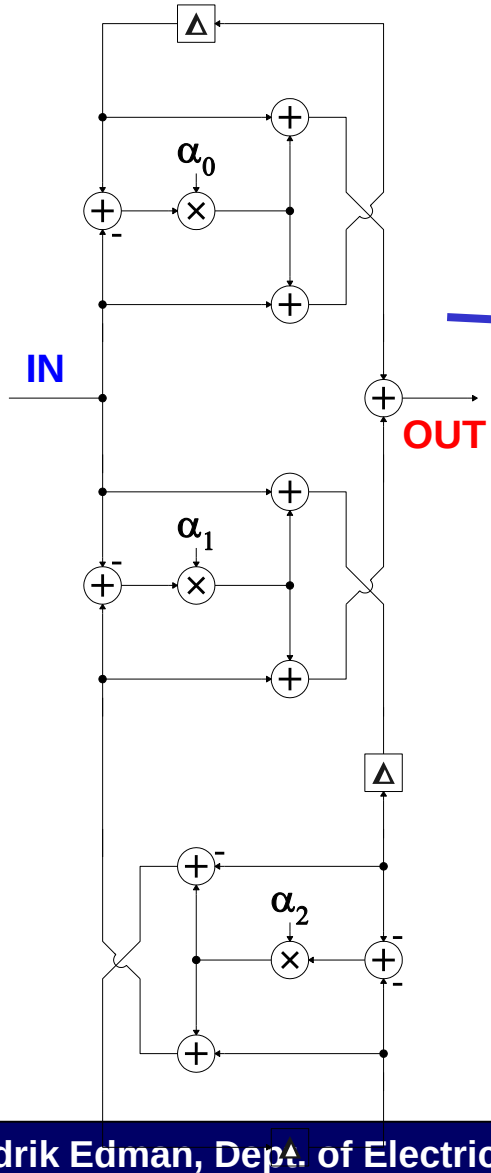
IIR Filter: Try with Trivial Coefficients

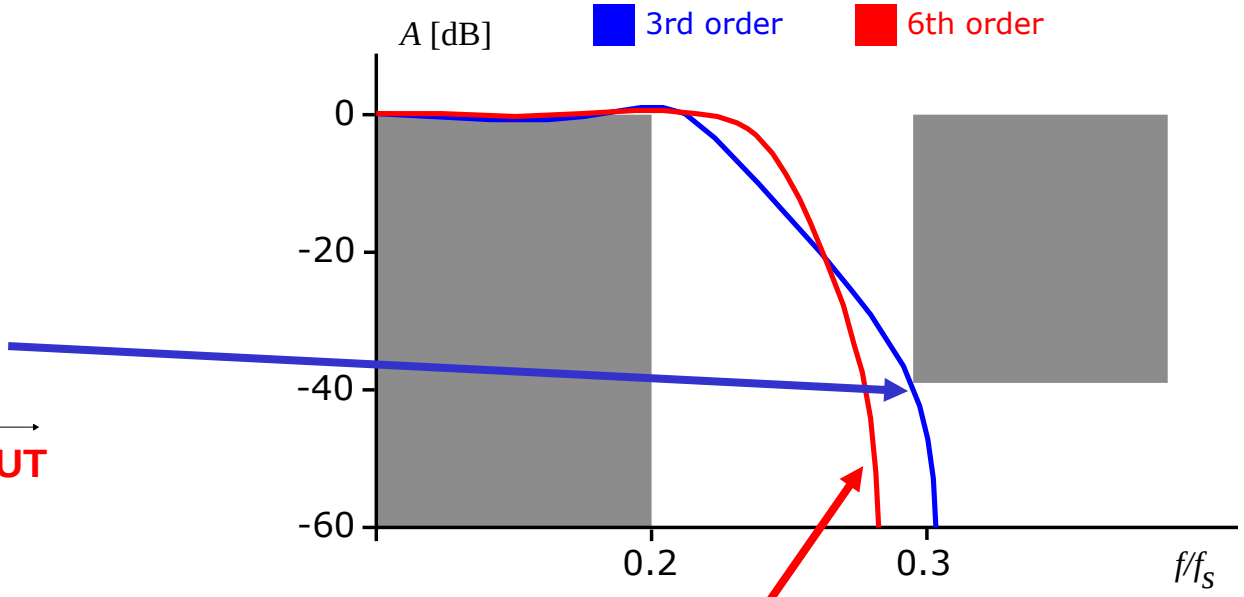


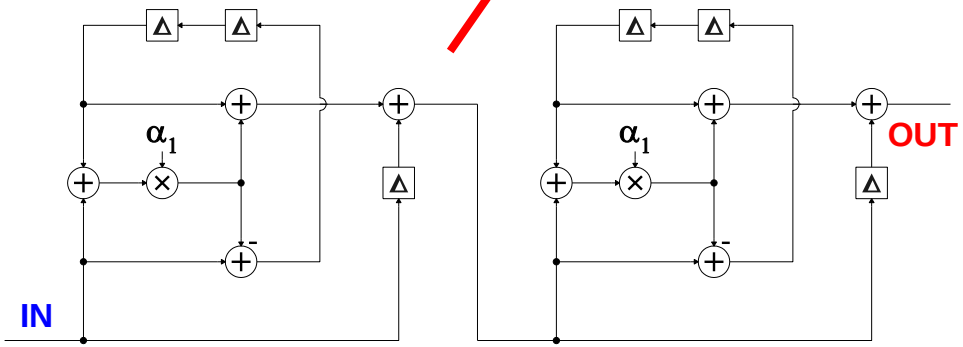


It did not meet the specification

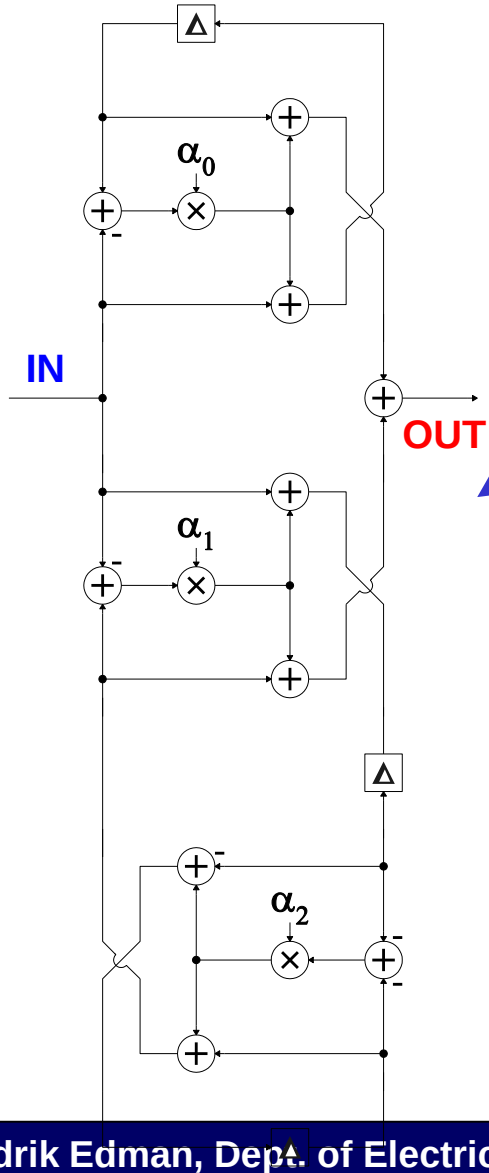
IIR Filter: Two Cascaded filters







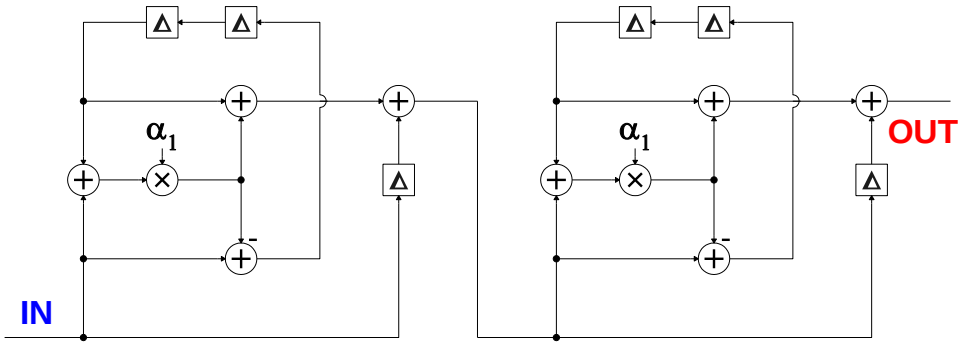
IIR Filter: Two Cascaded filters



10 Add:s and
3 Multiplications

10 Add:s

No Multiplications
(only shifts)



End of Lecture