



Filters in FPGA

Filter Equations

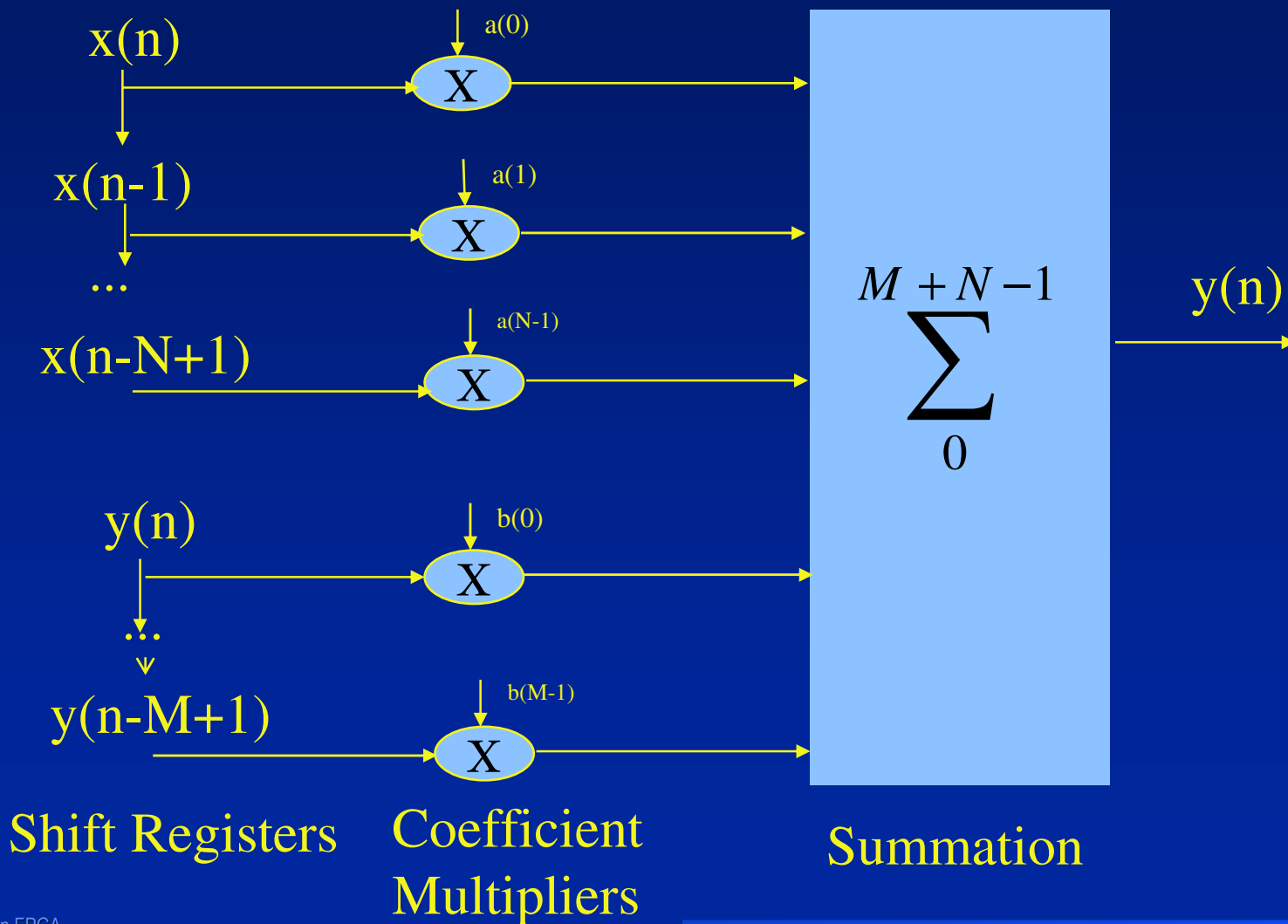
- ◆ A general Finite Impulse Response (FIR) Filter

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

- ◆ A general Infinite Impulse Response (IIR) Filter

$$y(n) = \sum_{k=0}^{N-1} a(k)x(n-k) + \sum_{k=0}^{M-1} b(k)y(n-k)$$

General Filter Structure



Filter Structure Analysis

- ◆ Shift Registers differentiate FIR and IIR
- ◆ Coefficient Multiplication and Summation is a sum of products equation

$$y = \sum_{k=0}^{N-1} c_k \cdot x_k$$

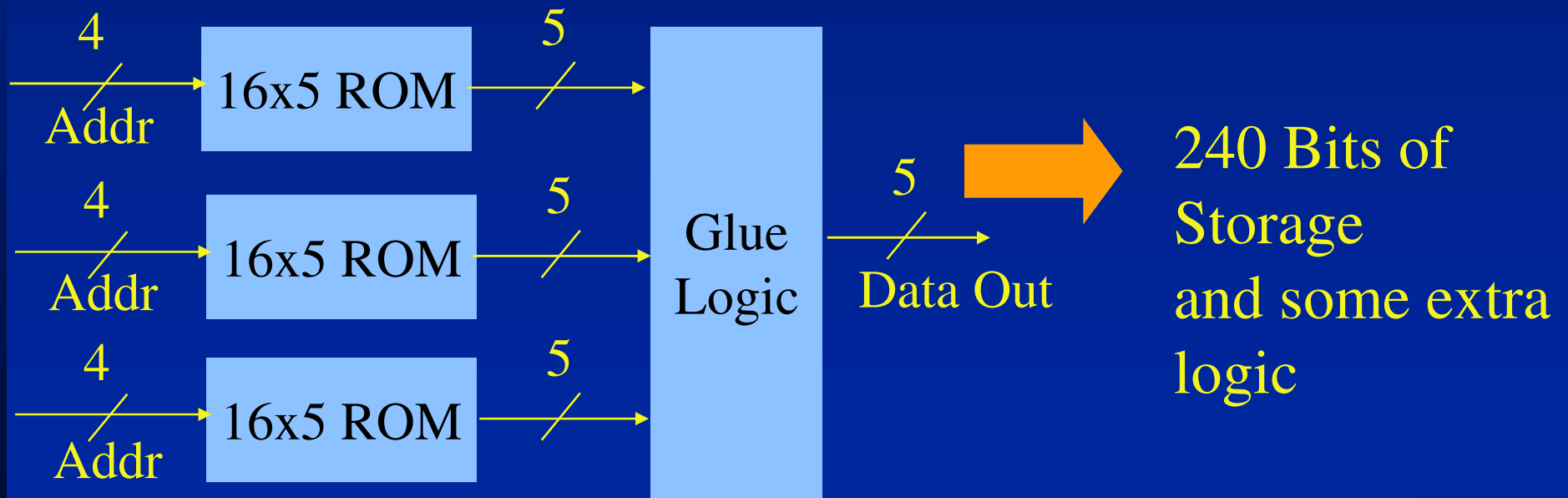
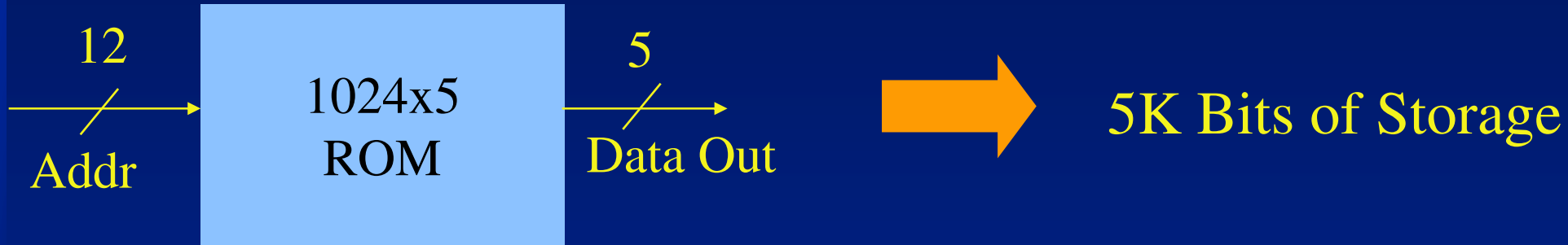
where c_k are coefficients and x_k are inputs to the sum of product equation.

— c_k is a constant for a given filter response

Basic Idea in Distributed Arithmetic

- ◆ Any Function can be realized by a huge memory.
- ◆ In DA, we break this huge memory in smaller memories, and stitch them using extra logic.
- ◆ In Virtex Series of FPGAs, the smallest memory sizes are 16 and 32 deep.
- ◆ The stitching logic consist mostly of Arithmetic Units like adders and subtractors. Use fast carry logic to implement them.

Basic Idea in DA



Basic Idea in Distributed Arithmetic

- ◆ The trick lies in getting a smart smaller partitioned memory and an appropriate stitching logic.
- ◆ For Filters to be implemented in FPGAs, there is a structured way to use Distributed Arithmetic.

A FIR Filter with Constant Coefficients

An N tap FIR Filter Equation given below

$$y(n) = \sum_{i=0}^{N-1} (C_i \times x(i))$$

is further expanded to

$$y(n) = \sum_{i=0}^{n-1} \left[\left\{ \sum_{k=0}^{c-1} (2^k \times C_{ik}) \right\} \times \left\{ \sum_{j=0}^{b-1} (2^j \times x_{ij}) \right\} \right]$$

where c-bits wide coefficients and b-bits wide inputs are represented in terms of its bits along with its weights.

Distributed Arithmetic in FIR

$$y(n) = \sum_{i=0}^{n-1} \left[\left\{ \sum_{k=0}^{c-1} (2^k \times C_{ik}) \right\} \times \left\{ \sum_{j=0}^{b-1} (2^j \times x_{ij}) \right\} \right]$$

$$y(n) = \sum_{j=0}^{b-1} \left[\sum_{i=0}^{n-1} \left[\left\{ \sum_{k=0}^{c-1} (2^k \times C_{ik}) \right\} \times x_{ij} \right] \times 2^j \right]$$

$$sda(j) = \sum_{i=0}^{n-1} \left[\left\{ \sum_{k=0}^{c-1} (2^k \times C_{ik}) \right\} \times x_{ij} \right]$$

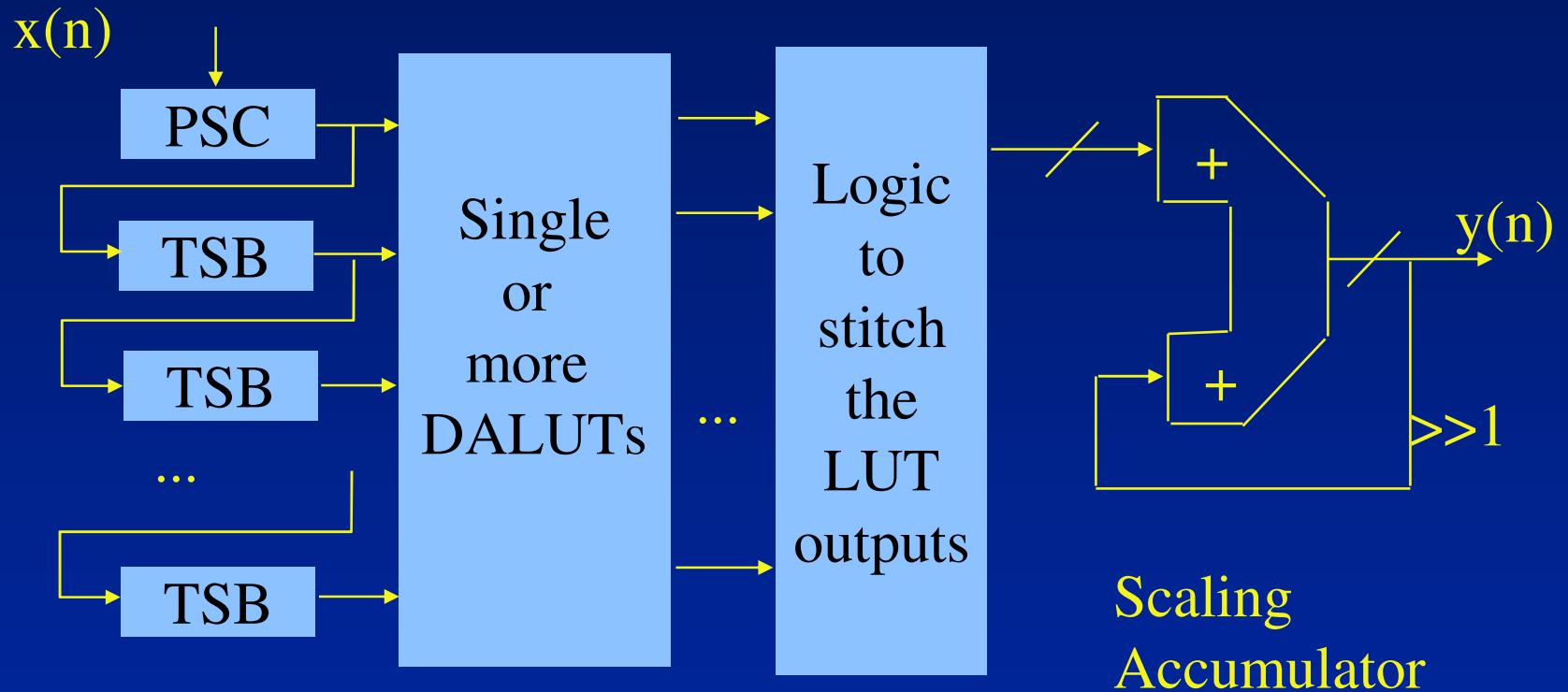
Distributed Arithmetic in FIR

- ◆ Filter can now be written as a sum of the common term $sda(j)$

$$y(n) = \sum_{j=0}^{b-1} 2^j \times sda(j)$$

- The term sda can now be implemented using Distributed Arithmetic LUT, whose inputs are the bits x_{ij}
- This sum is equivalent to a scaling accumulator

Distributed Arithmetic in FIR



PSC : Parallel to Serial Converter

TSB : Timing Skew Buffer

TSB in Xilinx FPGA

- ◆ TSB (Timing Skew Buffer) is a Serial In Serial Out Shift Register
- ◆ For XC4000 series use counter and synchronous RAM primitive RAM16x1 to implement this. Can be used to build delays of up to 17

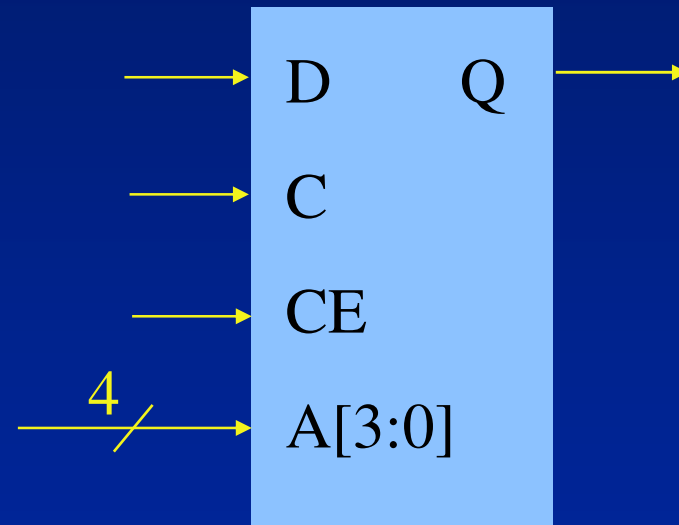


TSB with a delay of (N+1)

TSB in Xilinx FPGA

- ◆ For Virtex Series of FPGAs, a RAM Unit can be programmed as a Shift Register

Signal A specifies the delay number



SRL16E

TSB in Xilinx FPGA

- ◆ For delays greater than 17 use multiple Synchronous RAM or Shift Register LUT primitives.
- ◆ Use single bit in single bit out TSB to build multiple bits in multiple bits out TSB. A $m \times n$ -bit TSB is a shift register shifting n bits each cycle with a delay of m .

Distributed Arithmetic in FIR

- ◆ Filter Realizations are ways of implementing a sum of product equation

$$y = \sum_{k=0}^{N-1} c_k \cdot x_k$$

- ◆ In FPGAs, hardware configuration is very flexible. A host of implementations are possible
 - Fully Serial Arithmetic to Fully Parallel Arithmetic
 - Distributed Arithmetic, Multiply-Accumulate or Other Possible Implementations
- ◆ We present DA approach with optional multi-bit operations. Very popular for FPGAs.

Distributed Arithmetic in FIR

- ◆ The number of system clock cycles, one has to wait for the result to appear, determine the throughput.
 - Achieve high throughput with more parallel units.
- ◆ Latency is the number of cycles taken for first output to appear after reset.
 - High latency for Xilinx due to register rich implementation is not an issue in pipelined DSP applications

Common Terms in Distributed Arithmetic for 4 Tap FIR Filter

$$C_{33}C_{32}C_{31}C_{30}$$

$$X_{33}X_{32}X_{31}X_{30}$$

$$C_{23}C_{22}C_{21}C_{20}$$

$$X_{23}X_{22}X_{21}X_{20}$$

$$C_{13}C_{12}C_{11}C_{10}$$

$$X_{13}X_{12}X_{11}X_{10}$$

$$C_{03}C_{02}C_{01}C_{00}$$

$$X_{03}X_{02}X_{01}X_{00}$$

$Y_{330}Y_{320}Y_{310}Y_{300}$	$Y_{230}Y_{220}Y_{210}Y_{200}$	$Y_{130}Y_{120}Y_{110}Y_{100}$	$Y_{030}Y_{020}Y_{010}Y_{000}$
$Y_{331}Y_{321}Y_{311}Y_{301}$	$Y_{231}Y_{221}Y_{211}Y_{201}$	$Y_{131}Y_{121}Y_{111}Y_{101}$	$Y_{031}Y_{021}Y_{011}Y_{001}$
$Y_{332}Y_{322}Y_{312}Y_{302}$	$Y_{232}Y_{222}Y_{212}Y_{202}$	$Y_{132}Y_{122}Y_{112}Y_{102}$	$Y_{032}Y_{022}Y_{012}Y_{002}$
$Y_{333}Y_{323}Y_{313}Y_{303}$	$Y_{233}Y_{223}Y_{213}Y_{203}$	$Y_{133}Y_{123}Y_{113}Y_{103}$	$Y_{033}Y_{023}Y_{013}Y_{003}$

$$y_{ijk} = C_{ij} * x_{ik}$$

— Single Input Bit

Address Bits:

$X_{00} X_{10} X_{20} X_{30}$

— Two Input Bits

Address Bits:

$X_{00} X_{01} X_{10} X_{11}$

— Four Input Bits

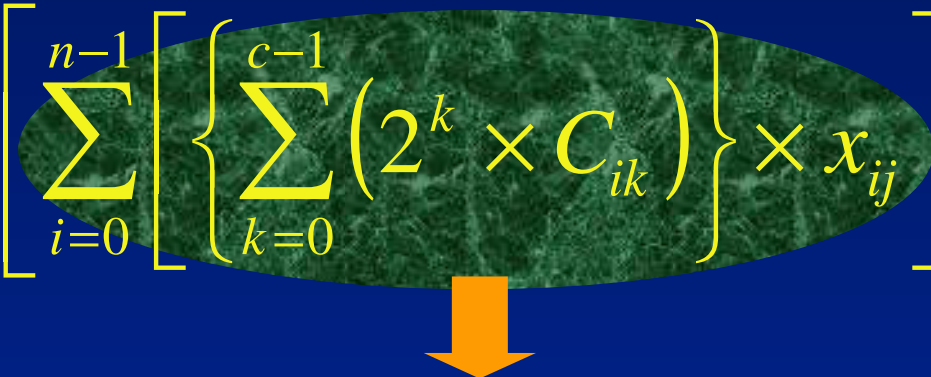
Address Bits:

$X_{00} X_{01} X_{02} X_{03}$

Distributed Arithmetic in FIR

- ◆ *Multi bit Covering*: The three implementations with single, double or four input bits covered fit nicely in 4 input DALUTs. Comparative speed can be achieved by registering as many stages as possible.
- ◆ *Multi bit Arithmetic*: A particular performance can be achieved by processing a particular number of bits in each cycle.
- ◆ For a required multi bit arithmetic, choose an appropriate covering.

Single Input-Bit Implementation of 4 Tap FIR Filter

$$y(n) = \sum_{j=0}^{b-1} \left[\sum_{i=0}^{n-1} \left[\left\{ \sum_{k=0}^{c-1} (2^k \times C_{ik}) \right\} \times x_{ij} \right] \times 2^j \right]$$


$sda(j)$ is the common term in the implementation
 $sda(j)$ are accumulated in the following fashion

$$y(n) = \sum_{j=0}^{b-1} 2^j \times sda(j)$$

Single Input Bit Implementation

$$C_{33}C_{32}C_{31}C_{30}$$

$$X_{33}X_{32}X_{31}X_{30}$$

$$C_{23}C_{22}C_{21}C_{20}$$

$$X_{23}X_{22}X_{21}X_{20}$$

$$C_{13}C_{12}C_{11}C_{10}$$

$$X_{13}X_{12}X_{11}X_{10}$$

$$C_{03}C_{02}C_{01}C_{00}$$

$$X_{03}X_{02}X_{01}X_{00}$$

$$Y_{330}Y_{320}Y_{310}Y_{300}$$

$$Y_{230}Y_{220}Y_{210}Y_{200}$$

$$Y_{130}Y_{120}Y_{110}Y_{100}$$

$$Y_{030}Y_{020}Y_{010}Y_{000}$$

$$Y_{331}Y_{321}Y_{311}Y_{301}$$

$$Y_{231}Y_{221}Y_{211}Y_{201}$$

$$Y_{131}Y_{121}Y_{111}Y_{101}$$

$$Y_{031}Y_{021}Y_{011}Y_{001}$$

$$Y_{332}Y_{322}Y_{312}Y_{302}$$

$$Y_{232}Y_{222}Y_{212}Y_{202}$$

$$Y_{132}Y_{122}Y_{112}Y_{102}$$

$$Y_{032}Y_{022}Y_{012}Y_{002}$$

$$Y_{333}Y_{323}Y_{313}Y_{303}$$

$$Y_{233}Y_{223}Y_{213}Y_{203}$$

$$Y_{133}Y_{123}Y_{113}Y_{103}$$

$$Y_{033}Y_{023}Y_{013}Y_{003}$$

$$y_{ijk} = C_{ij} * x_{ik}$$

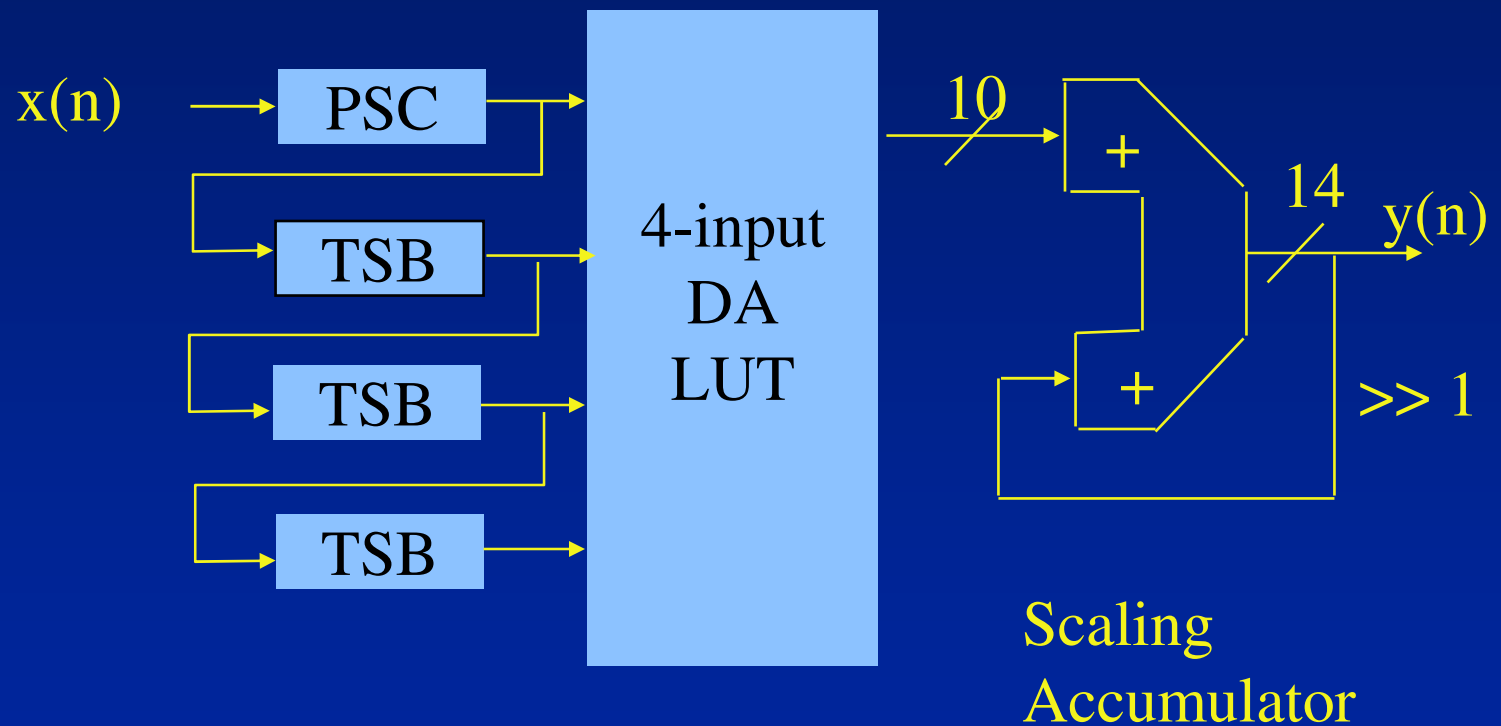
Common DALUT addressed by $(x_{0j} x_{1j} x_{2j} x_{3j})$ for j^{th} partial product.

Single Input-Bit Implementation of 4 Tap FIR Filter

- ◆ The common term sda is implemented using Distributed Arithmetic Look Up Table (DA LUT)
- ◆ The scaling accumulation can be performed with appropriate parallelism to achieve the required throughput

Fully Serial Single Input-Bit Implementation of 4 Tap FIR

- ◆ Assume coefficient width is 8 and input bit width is 4



DALUT contents of a 4 Tap FIR

- ◆ The Filter Equation is $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$

DALUT

Contents are shown here for the addresses from 0 to 15. This DALUT will fit in a 4 input LUT.

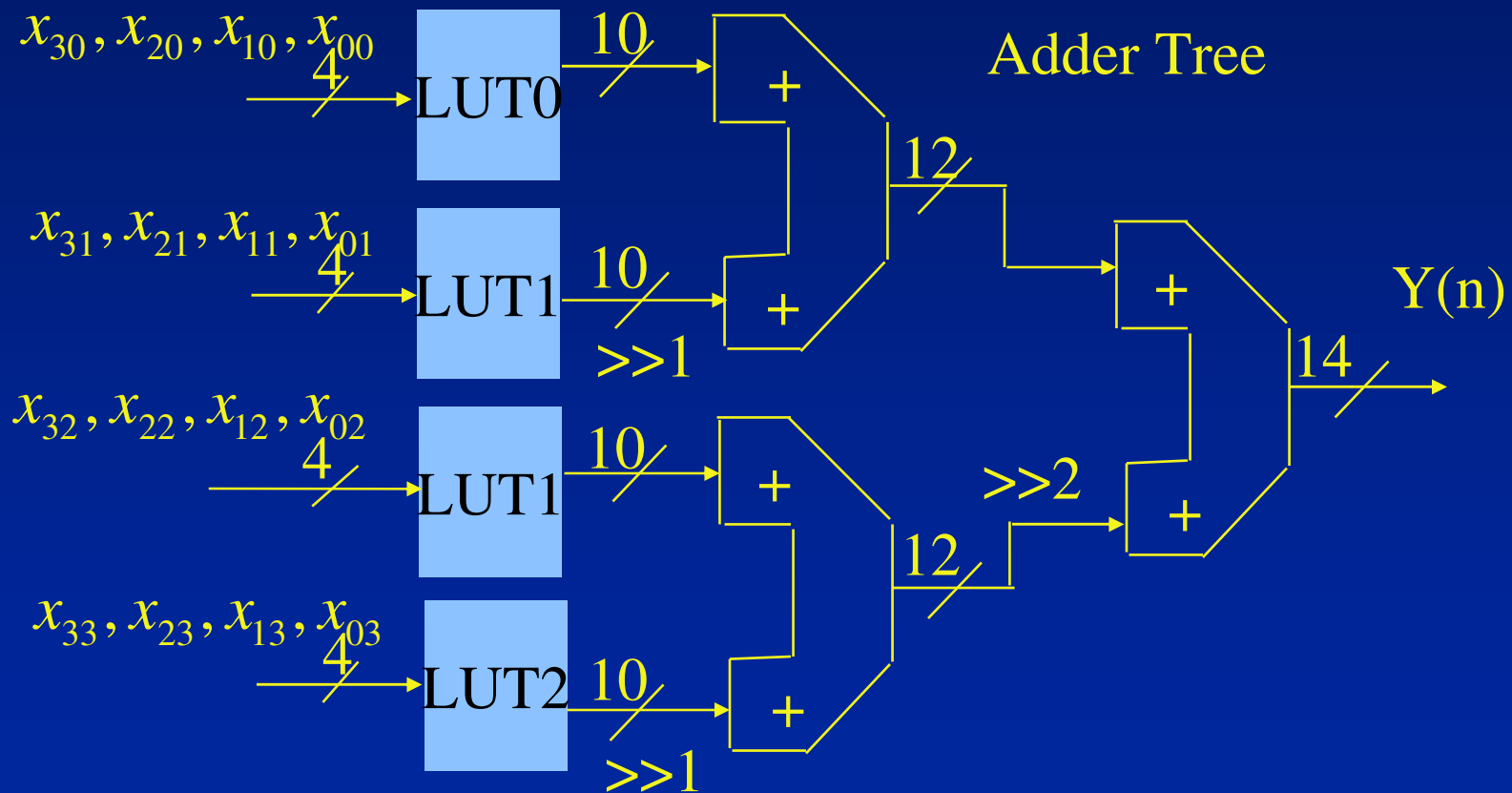
0	0	8	C_3
1	C_0	9	$C_3 + C_0$
2	C_1	10	$C_3 + C_1$
3	$C_1 + C_0$	11	$C_3 + C_1 + C_0$
4	C_2	12	$C_3 + C_2$
5	$C_2 + C_0$	13	$C_3 + C_2 + C_0$
6	$C_2 + C_1$	14	$C_3 + C_2 + C_1$
7	$C_2 + C_1 + C_0$	15	$C_3 + C_2 + C_1 + C_0$

Fully Parallel Single Input-Bit Implementation of 4 Tap FIR Filter

- ◆ Remove Shift Registers and Scaling Accumulator from the Serial Implementation
- ◆ Replicate DALUT as many times as there are input bits
- ◆ Add the DALUT outputs using an adder tree with proper shifting
- ◆ Achieves 4 times the throughput of the fully serial implementation.

Fully Parallel 4-Tap Filter

- Assume coefficient width is 8 and input bit width is 4.

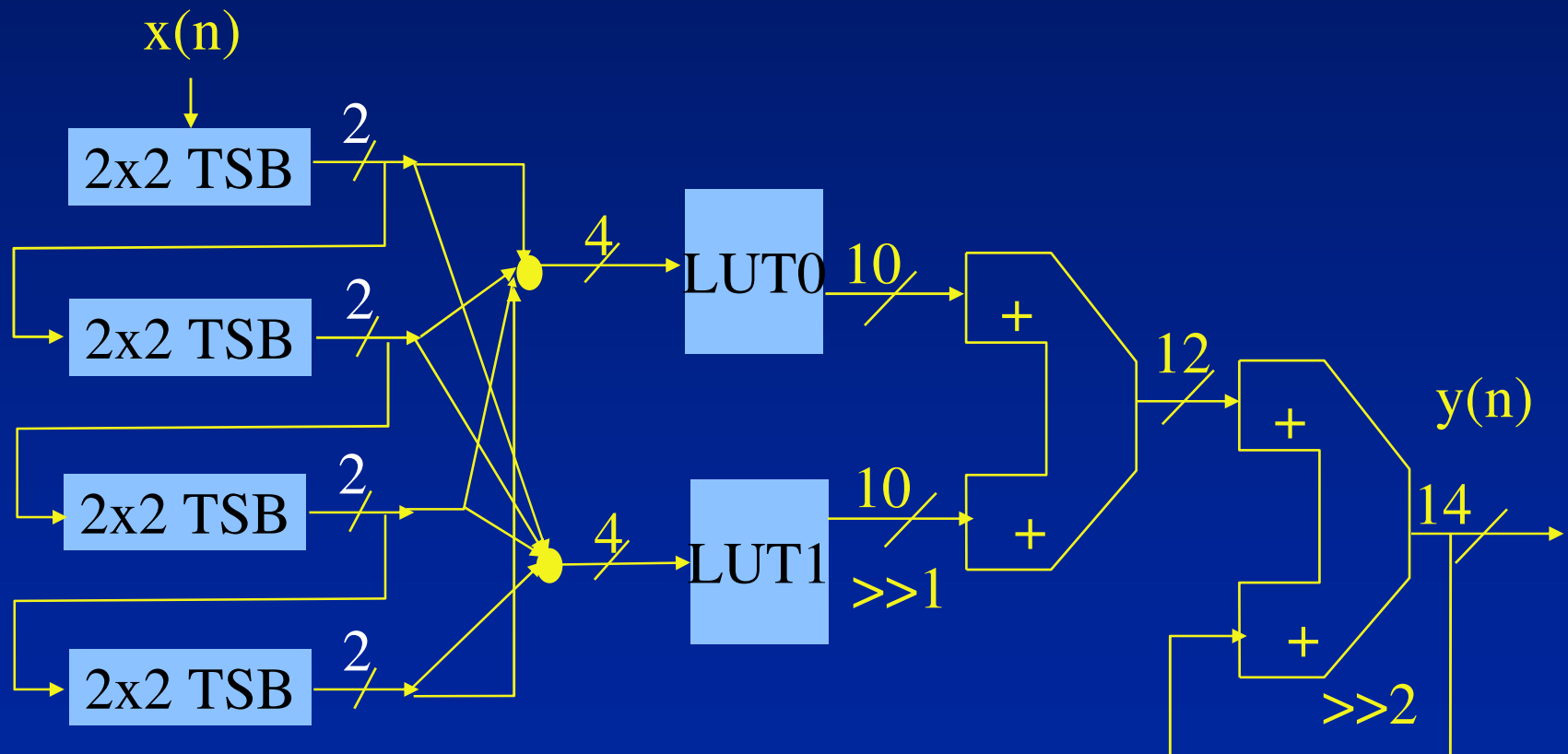


Half Parallel Single Input-Bit Implementation of 4 Tap FIR Filter

- ◆ *1x1* TSB changes to *2x2* TSB
- ◆ DALUTs are replicated twice
- ◆ Add the DALUT outputs using an adder tree with proper shifting
- ◆ Achieves 2 times the throughput of the fully serial implementation

Half Parallel 4-Tap Filter

- Assume coefficient width is 8 and input bit width is 4.

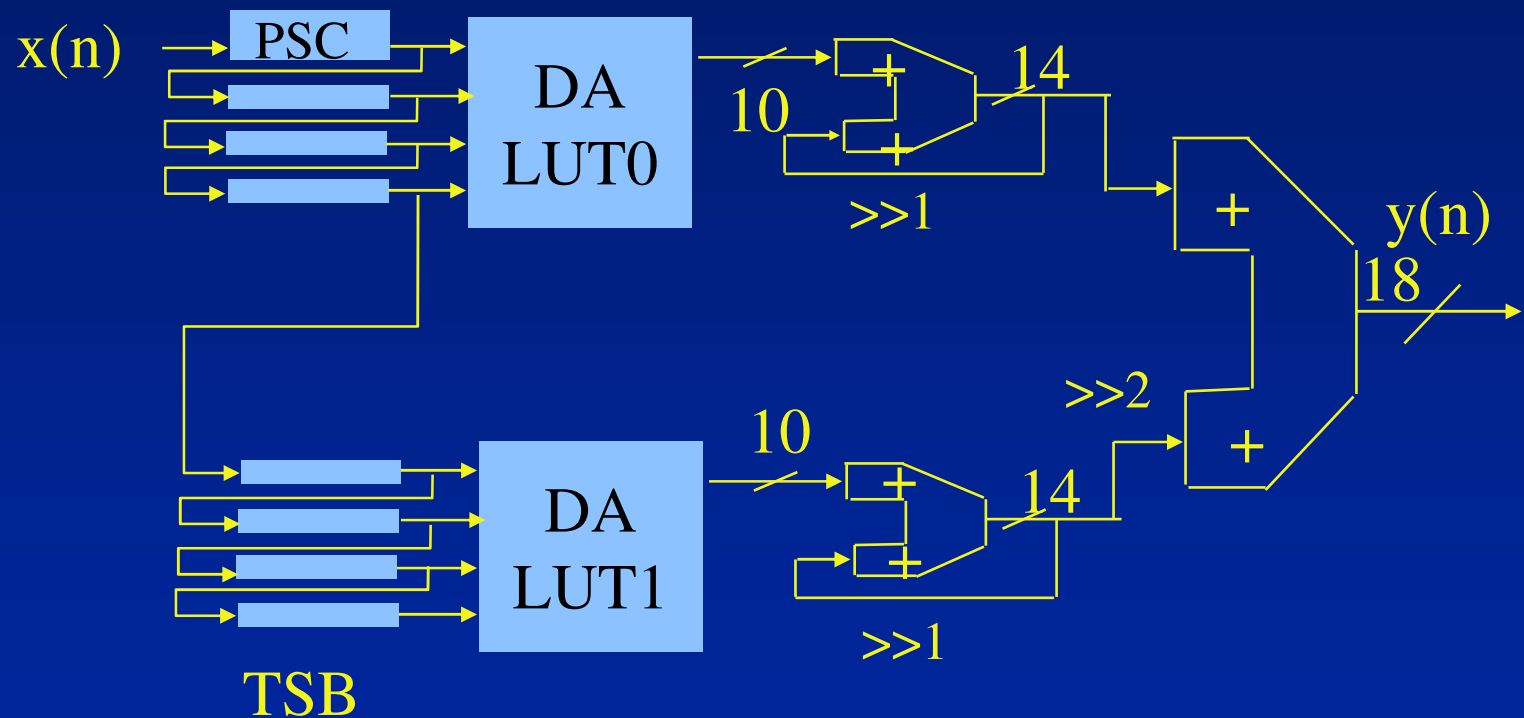


Salient points in FIR Implementation

- ◆ For achieving a higher throughput, replicate the DALUTs, modify the TSBs, Adder Tree and the Scaling Accumulator
- ◆ No TSBs and Scaling Accumulator in the fully parallel case

Fully Serial Single Input-Bit Implementation of 8 Tap FIR

- ◆ Assume coefficient width is 8 and input bit width is 4



Salient points in large FIR Filter Implementations

- ◆ Use 4 input DALUT by grouping inputs in groups of 4. These are conveniently implemented in 4 input Logic Blocks.
- ◆ Connect these DALUTs using Adder Tree with appropriate shifts to get the correct result.
- ◆ For a serial implementation, use scaling accumulator to accumulate for output.

Two Input Bits Implementation

$$C_{33}C_{32}C_{31}C_{30}$$

$$X_{33}X_{32}X_{31}X_{30}$$

$$C_{23}C_{22}C_{21}C_{20}$$

$$X_{23}X_{22}X_{21}X_{20}$$

$$C_{13}C_{12}C_{11}C_{10}$$

$$X_{13}X_{12}X_{11}X_{10}$$

$$C_{03}C_{02}C_{01}C_{00}$$

$$X_{03}X_{02}X_{01}X_{00}$$

$$Y_{330}Y_{320}Y_{310}Y_{300}$$

$$Y_{230}Y_{220}Y_{210}Y_{200}$$

$$Y_{331}Y_{321}Y_{311}Y_{301}$$

$$Y_{231}Y_{221}Y_{211}Y_{201}$$

$$Y_{130}Y_{120}Y_{110}Y_{100}$$

$$Y_{030}Y_{020}Y_{010}Y_{000}$$

$$Y_{131}Y_{121}Y_{111}Y_{101}$$

$$Y_{031}Y_{021}Y_{011}Y_{001}$$

$$Y_{332}Y_{322}Y_{312}Y_{302}$$

$$Y_{232}Y_{222}Y_{212}Y_{202}$$

$$Y_{333}Y_{323}Y_{313}Y_{303}$$

$$Y_{233}Y_{223}Y_{213}Y_{203}$$

$$Y_{132}Y_{122}Y_{112}Y_{102}$$

$$Y_{032}Y_{022}Y_{012}Y_{002}$$

$$Y_{133}Y_{123}Y_{113}Y_{103}$$

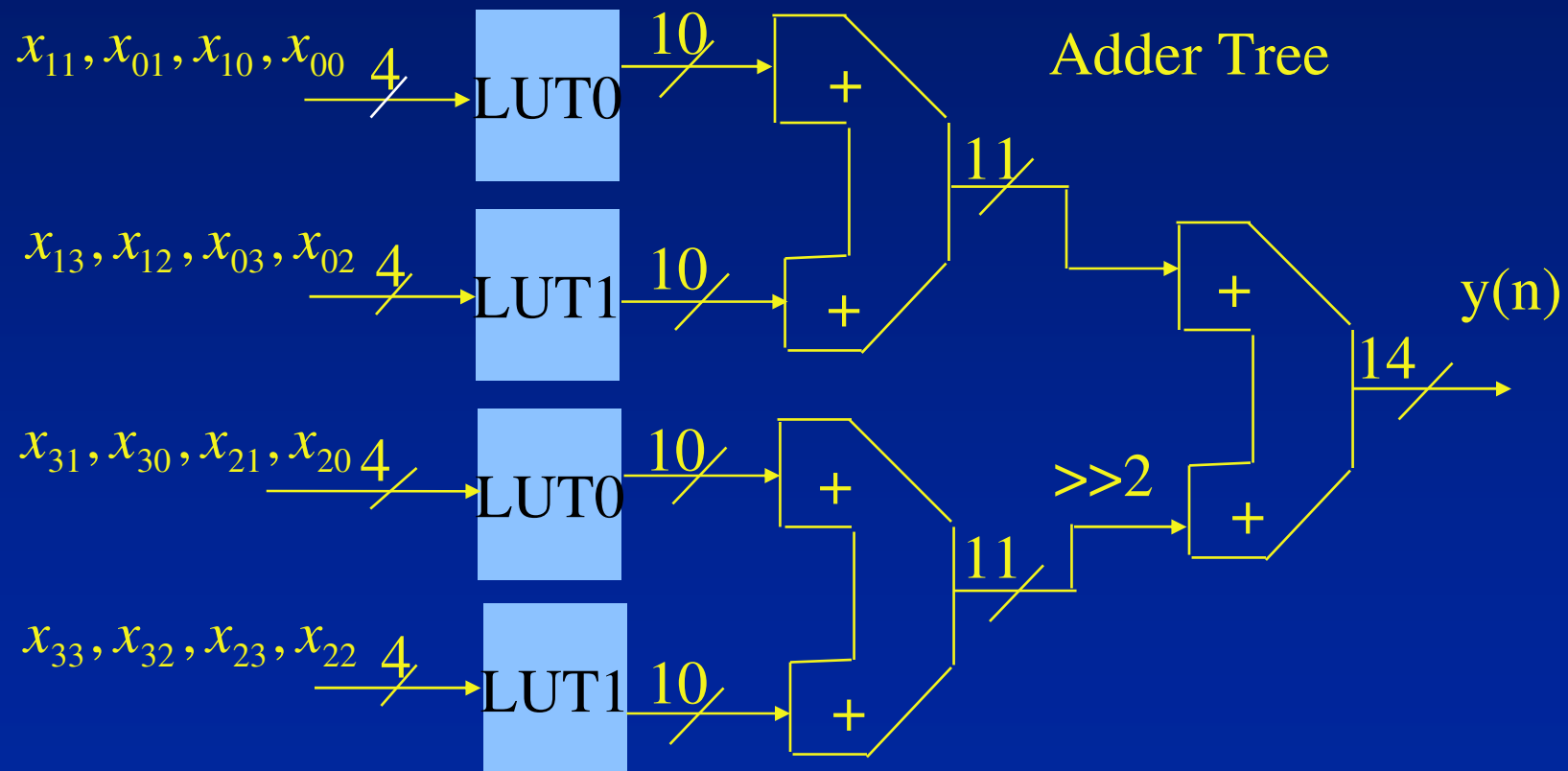
$$Y_{033}Y_{023}Y_{013}Y_{003}$$

Another common DALUT addressed by $(x_{22} x_{23} x_{32} x_{32})$ for a common partial product.

A common DALUT addressed by $(x_{00} x_{01} x_{10} x_{11})$ for a common partial product.

Fully Parallel Two Input Bit Implementation for 4 Tap FIR

- ◆ Assume coefficient width is 8 and input bit width is 4.



DALUT contents for Two Bit

- ◆ The Filter Equation is $y=C_0x_0+C_1x_1+C_2x_2+C_3x_3$

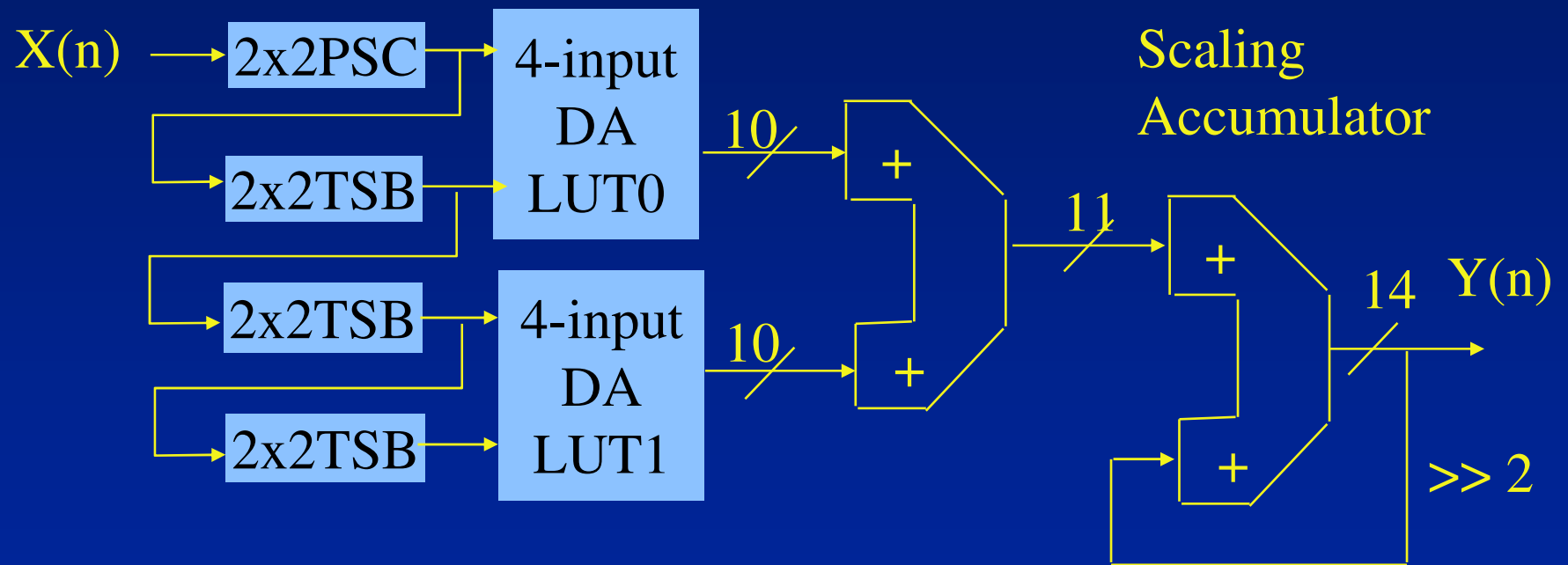
0	0	8	$2*C1$
1	$C0$	9	$2*C1+C0$
2	$2*C0$	10	$2*C1+2*C0$
3	$3*C0$	11	$2*C1+3*C0$
4	$C1$	12	$3*C1$
5	$C1+C0$	13	$3*C1+C0$
6	$C1+2*C0$	14	$3*C1+2*C0$
7	$C1+3*C0$	15	$3*C1+3*C0$

LUT0

0	0	8	$2*C3$
1	$C2$	9	$2*C3+C2$
2	$2*C2$	10	$2*C3+2*C2$
3	$3*C2$	11	$2*C3+3*C2$
4	$C3$	12	$3*C3$
5	$C3+C2$	13	$3*C3+C2$
6	$C3+2*C2$	14	$3*C3+2*C2$
7	$C3+3*C2$	15	$3*C3+3*C2$

LUT1

Fully Serial Two Input Bit Implementation for 4 Tap FIR



Four Input Bits Implementation

$C_{33}C_{32}C_{31}C_{30}$
 $X_{33}X_{32}X_{31}X_{30}$

$C_{23}C_{22}C_{21}C_{20}$
 $X_{23}X_{22}X_{21}X_{20}$

$C_{13}C_{12}C_{11}C_{10}$
 $X_{13}X_{12}X_{11}X_{10}$

$C_{03}C_{02}C_{01}C_{00}$
 $X_{03}X_{02}X_{01}X_{00}$

$Y_{330}Y_{320}Y_{310}Y_{300}$
 $Y_{331}Y_{321}Y_{311}Y_{301}$
 $Y_{332}Y_{322}Y_{312}Y_{302}$
 $Y_{333}Y_{323}Y_{313}Y_{303}$

$Y_{230}Y_{220}Y_{210}Y_{200}$
 $Y_{231}Y_{221}Y_{211}Y_{201}$
 $Y_{232}Y_{222}Y_{212}Y_{202}$
 $Y_{233}Y_{223}Y_{213}Y_{203}$

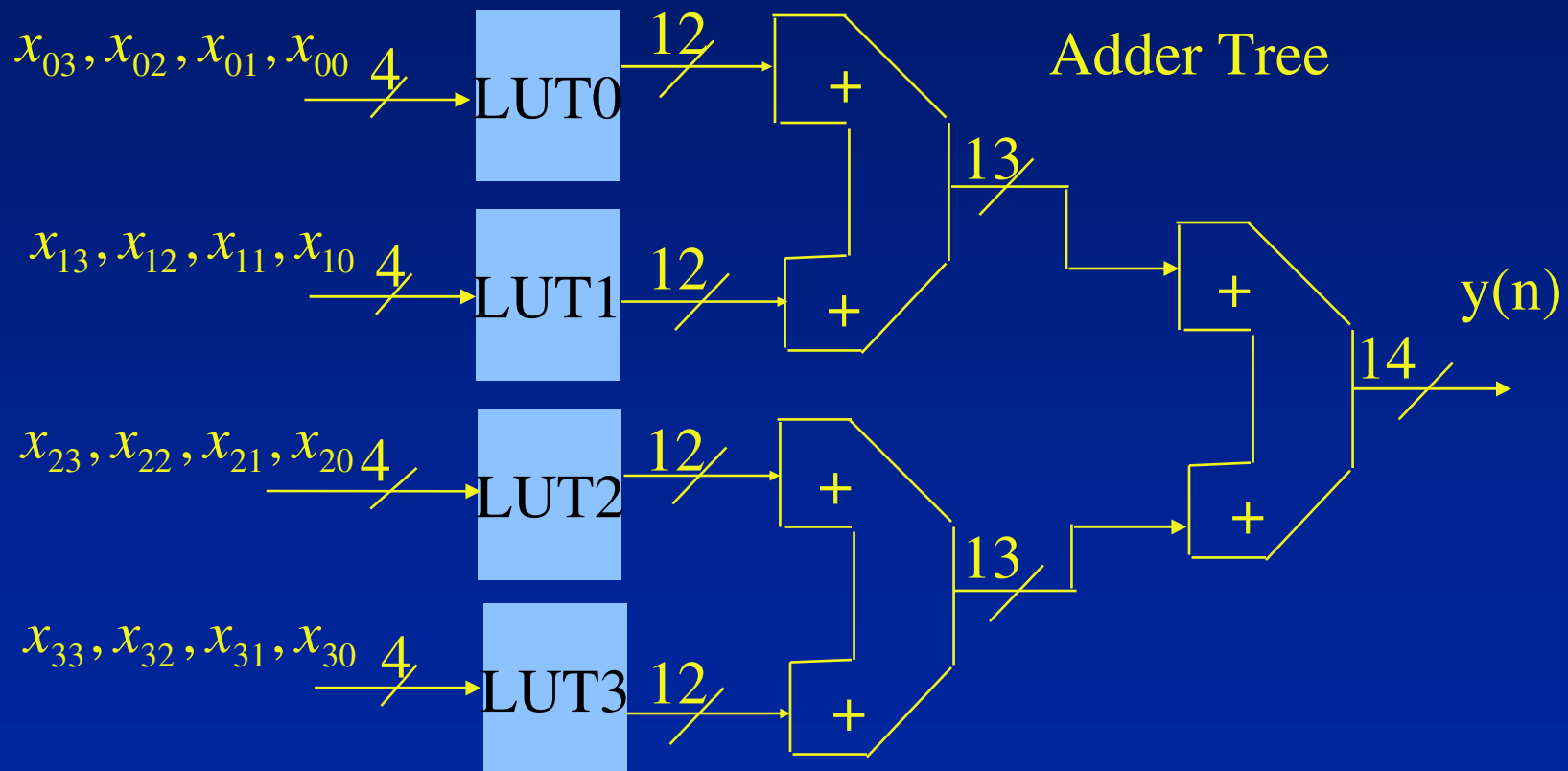
$Y_{130}Y_{120}Y_{110}Y_{100}$
 $Y_{131}Y_{121}Y_{111}Y_{101}$
 $Y_{132}Y_{122}Y_{112}Y_{102}$
 $Y_{133}Y_{123}Y_{113}Y_{103}$

$Y_{030}Y_{020}Y_{010}Y_{000}$
 $Y_{031}Y_{021}Y_{011}Y_{001}$
 $Y_{032}Y_{022}Y_{012}Y_{002}$
 $Y_{033}Y_{023}Y_{013}Y_{003}$

Four Different DALUTs for each tap addressed by 4 bits of input data x. DALUTs will be common if number of bits in input data x is more than 4.

Fully Parallel Four Input Bit Implementation for 4 Tap FIR

- ◆ Assume coefficient width is 8 and input bit width is 4.



DALUT contents of a 4 Tap FIR

- ◆ The Filter Equation is $y = C_0x_0 + C_1x_1 + C_2x_2 + C_3x_3$

DALUT contents are shown here for the coefficient C_0 . Similar DALUT contents can be derived for C_1 , C_2 and C_3 .

0	0	8	8*C0
1	C0	9	9*C0
2	2*C0	10	10*C0
3	3*C0	11	11*C0
4	4*C0	12	12*C0
5	5*C0	13	13*C0
6	6*C0	14	14*C0
7	7*C0	15	15*C0

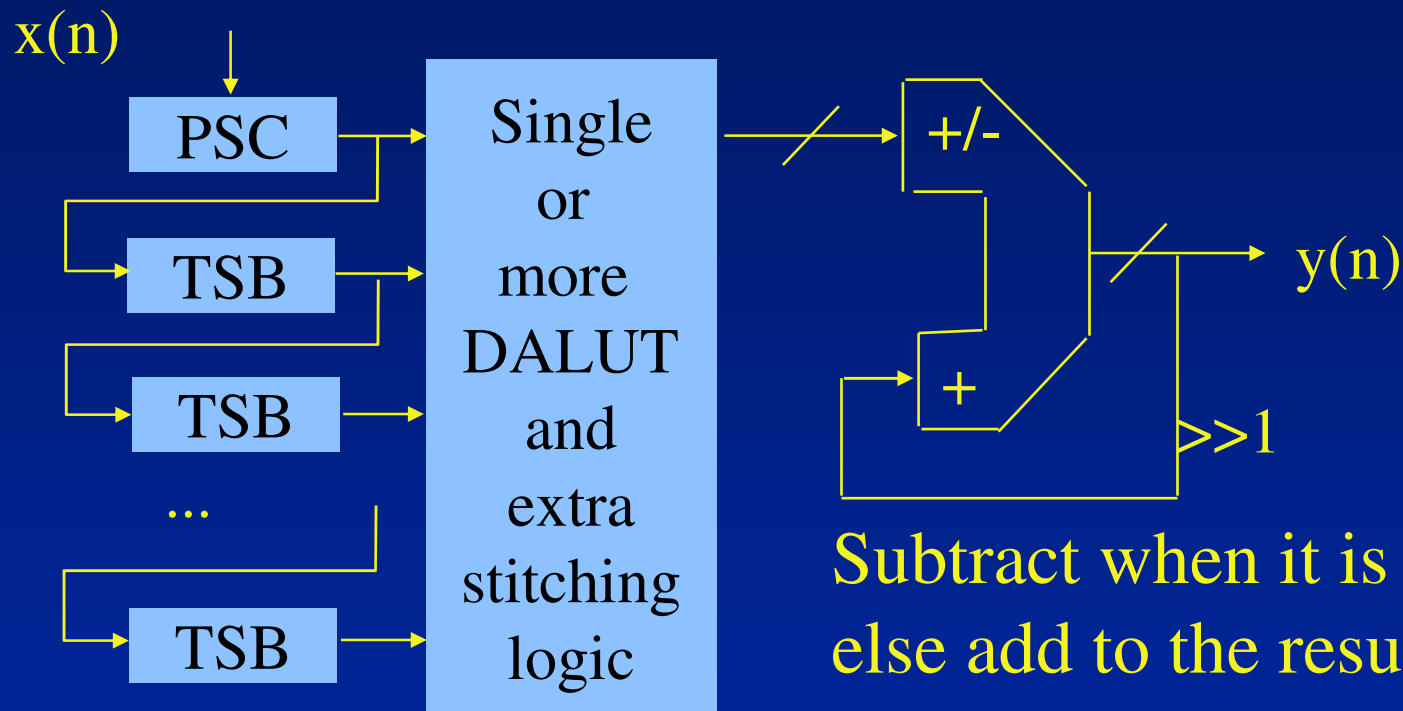
Different Distributed Arithmetic choices and their comparisons

- ◆ Range of throughputs available for different coverings
- ◆ For real designs, choose a throughput for the required bit rate
- ◆ One of the coverings single, two or four input bits will give the most optimal result

FIR Implementations: Signed Numbers

- ◆ Signed Coefficient Numbers are stored as twos complement in DA LUTs
- ◆ Signed Data Inputs
 - For serial implementation, subtract the partial product if sign bit is 1
 - For parallel implementations, the DALUT contents are calculated properly for sign bits

Signed Numbers in Serial Distributed Arithmetic for FIR



Subtract when it is sign bit
else add to the result

PSC : Parallel to Serial Converter

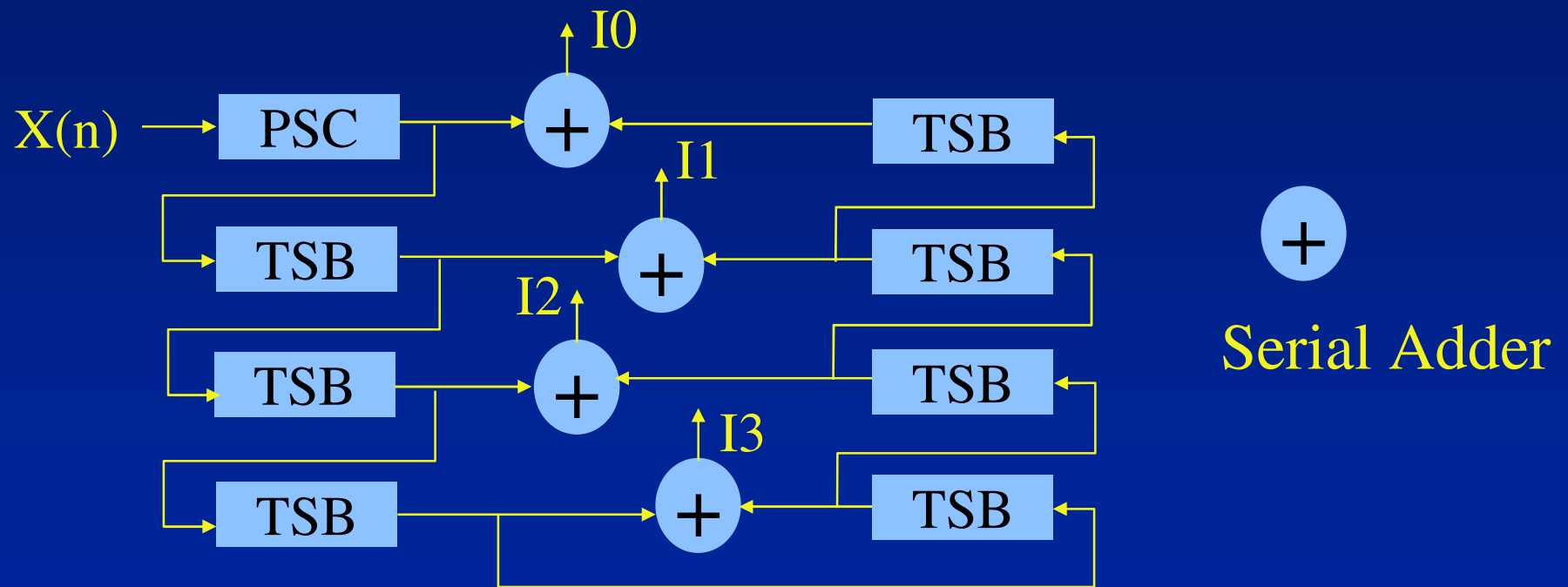
TSB : Timing Skew Buffer

Symmetrical FIR Filter

- ◆ For an N tap symmetrical FIR Filter, $h(k)$ is same as $h(N-k)$ for k less than $N/2$.
- ◆ Popular because phase response is constant
- ◆ Can be implemented in hardware using $N/2$ tap Filter.

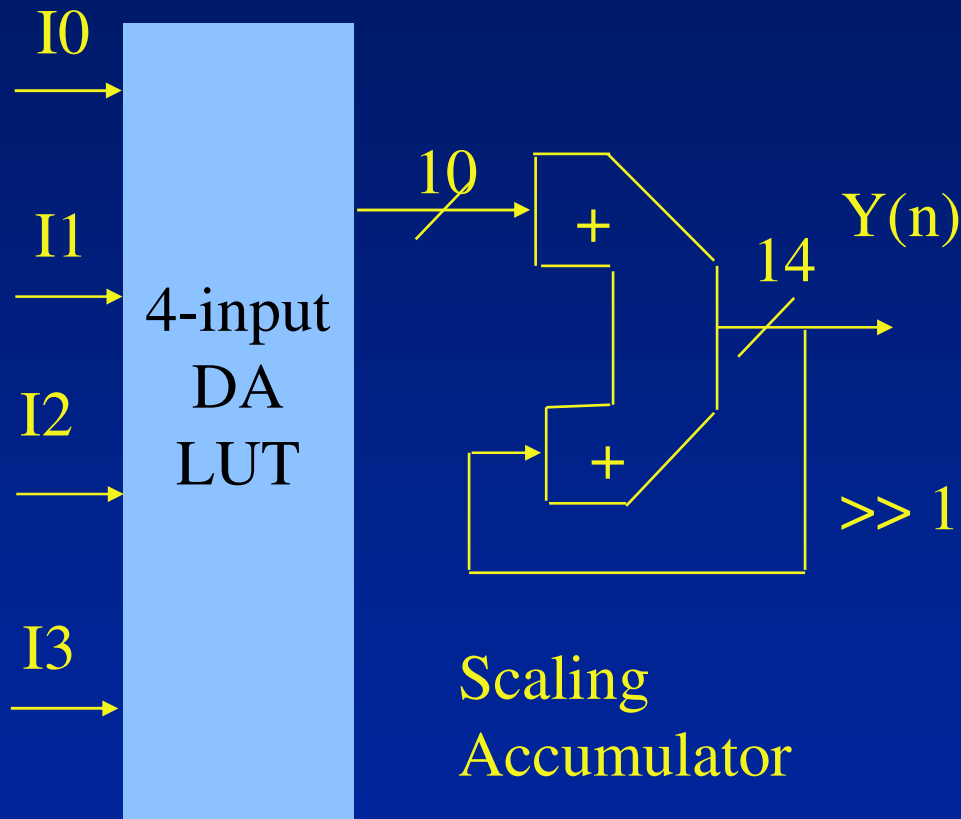
8 tap Symmetrical FIR Filter -I

- ◆ Assume coefficient width is 8 and input bit width is 4
- ◆ $y = C_0*(x_0+x_7)+C_1*(x_1+x_6)+C_2*(x_2+x_5)+C_3*(x_3+x_4)$



Shift Register Arrangements

8 tap Symmetrical FIR Filter -II



0	0	8	C3
1	C0	9	C3+C0
2	C1	10	C3+C1
3	C1+C0	11	C3+C1+C0
4	C2	12	C3+C2
5	C2+C0	13	C3+C2+C0
6	C2+C1	14	C3+C2+C1
7	C2+C1+C0	15	C3+C2+C1+C0

DALUT Contents

Comment on Distributed Arithmetic Implementations

- ◆ A wide array of implementation with varying throughput can be obtained in a standard way by using Distributed Arithmetic.
- ◆ In certain cases, it might be possible that a better implementation exists. However DA results into good implementations in a consistent manner.

IIR Filter

- ◆ A general Infinite Impulse Response (IIR) Filter

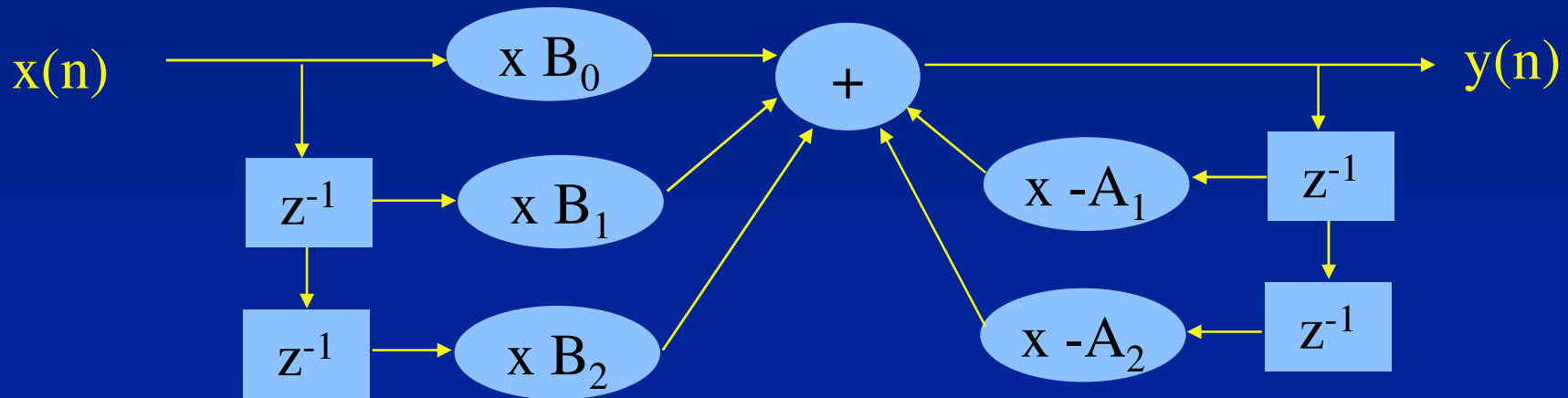
$$y(n) = \sum_{k=0}^{N-1} a(k)x(n-k) + \sum_{k=0}^{M-1} b(k)y(n-k)$$

- ◆ Difficult to design a stable high order IIR Filter. A Biquad Filter is more popular. High Order Filters are cascaded Biquad Filters.

Biquad IIR Filter

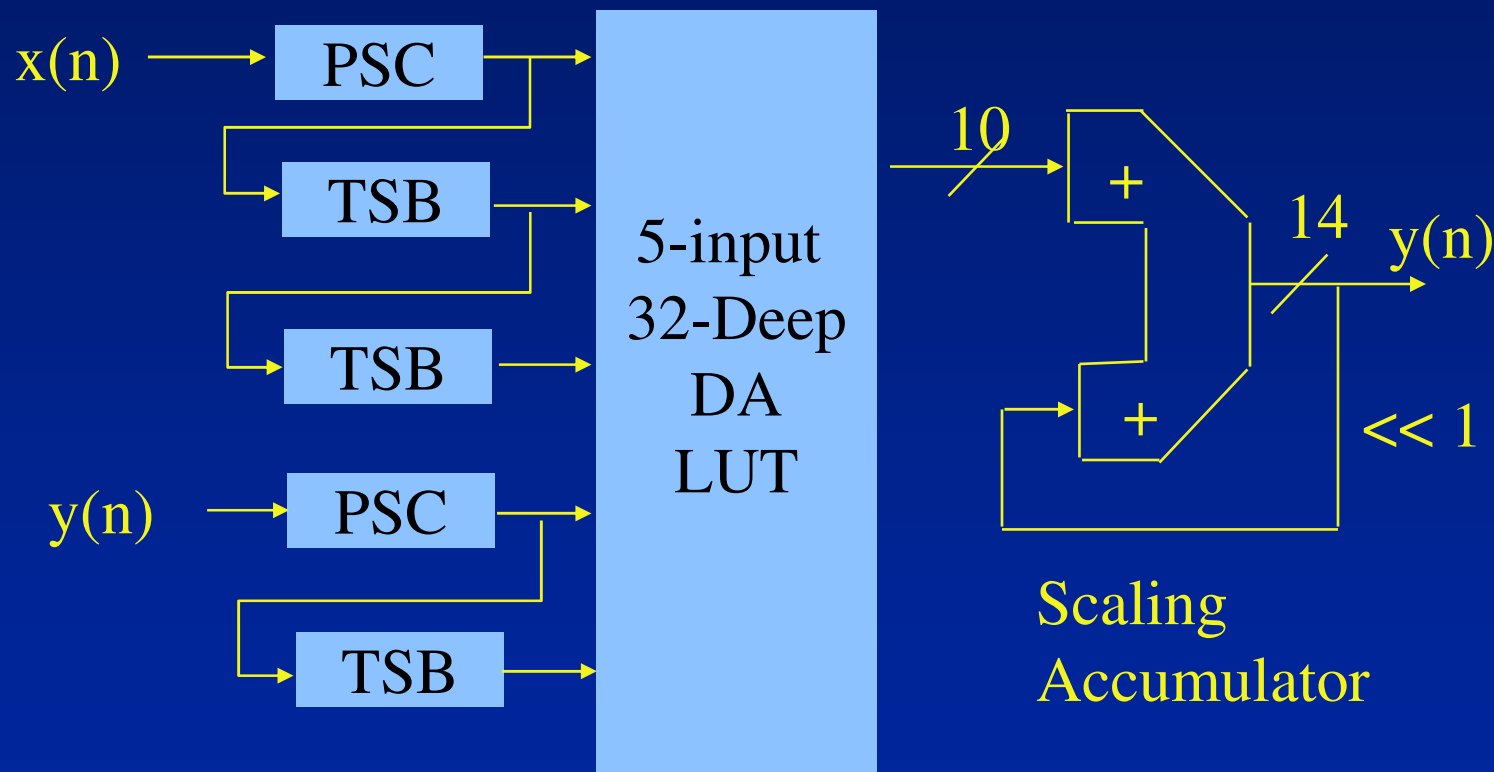
- ◆ Biquad Filter Equation

- $y(n) = B_0 * x(n) + B_1 * x(n-1) + B_2 * x(n-2) - A_1 * y(n-1) - A_2 * y(n-2)$



Serial Biquad IIR Implementation

- ◆ Assume coefficient width is 8 and input bit width is 4



No Registers in DALUT

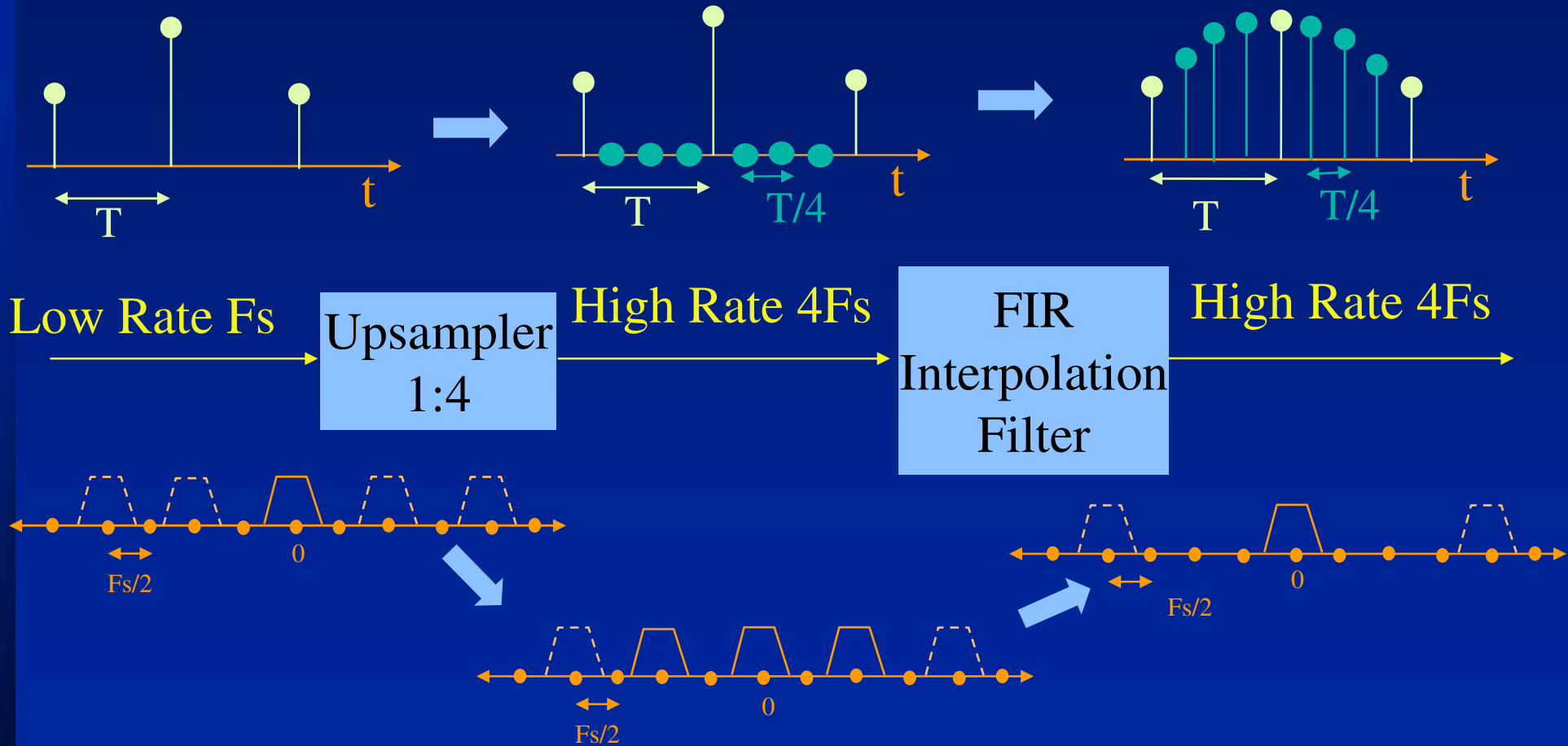
DALUT contents of Biquad IIR

0	0	6	$B2+B1$	16	$-A2$
1	$B0$	7	$B2+B1+B0$	17	$-A2+B0$
2	$B1$	8	$-A1$
3	$B1+B0$	9	$-A1+B0$	24	$-A2-A1$
4	$B2$
5	$B2+B0$	15	$-A1+B2+B1+B0$	31	$-A2-A1+B2+B1+B0$

IIR Filters

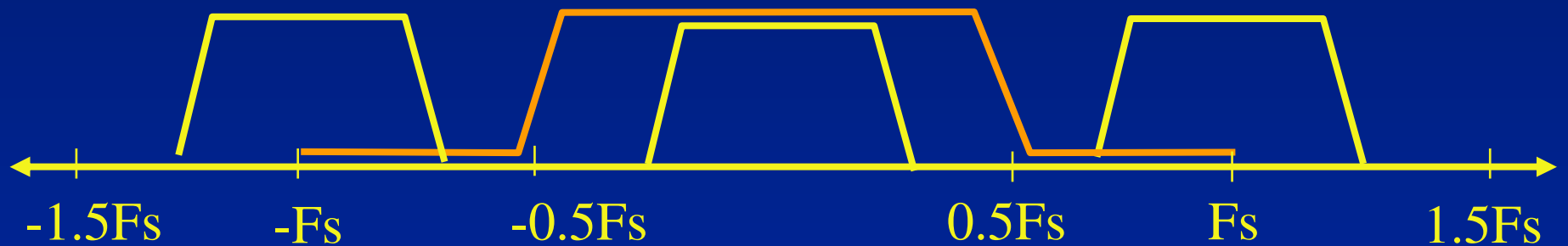
- ◆ Since the output is fed back, the delays in the output path are an issue.
- ◆ Fully pipelined operations are not viable.
- ◆ Good Performance can be achieved for Biquad Filters even without pipelining.

Interpolation Filter



1:2 Interpolation FIR Filter

- ◆ For 1:2 interpolation, sample frequency changes from F_s to $2F_s$.



— Low Pass Filter Response to remove the aliased frequency component beyond $0.5F_s$

10 Tap 1:2 Interpolating FIR

- ◆ Insert one zero every other data sample

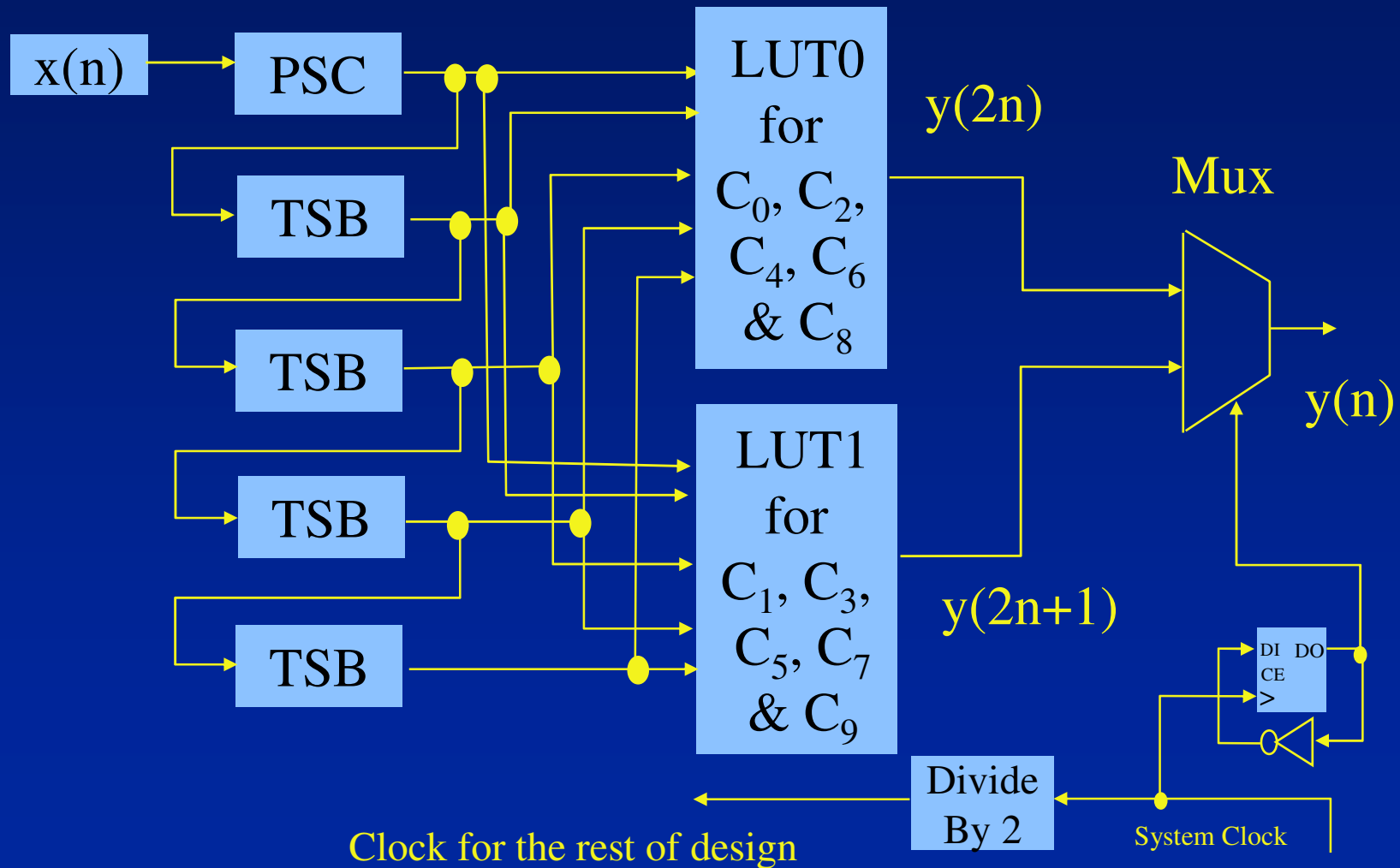
- ◆ Even Outputs:

$$\begin{aligned} \text{— } y(2n) &= C_0 * x(n) + C_1 * 0 + C_2 * x(n-1) + C_3 * 0 + C_4 * x(n-2) \\ &\quad + C_5 * 0 + C_6 * x(n-3) + C_7 * 0 + C_8 * x(n-4) + C_9 * 0 \\ &= C_0 * x(n) + C_2 * x(n-1) + C_4 * x(n-2) + C_6 * x(n-3) + C_8 * x(n-4) \end{aligned}$$

- ◆ Odd Outputs:

$$\begin{aligned} \text{— } y(2n+1) &= C_0 * 0 + C_1 * x(n) + C_2 * 0 + C_3 * x(n-1) + C_4 * 0 + \\ &\quad C_5 * x(n-2) + C_6 * 0 + C_7 * x(n-3) + C_8 * 0 + C_9 * x(n-4) \\ &= C_1 * x(n) + C_3 * x(n-1) + C_5 * x(n-2) + C_7 * x(n-3) + C_9 * x(n-4) \end{aligned}$$

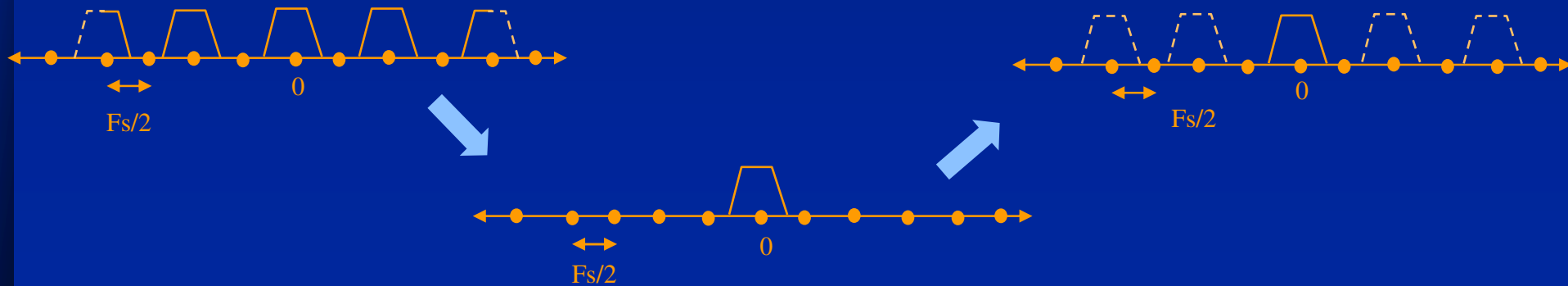
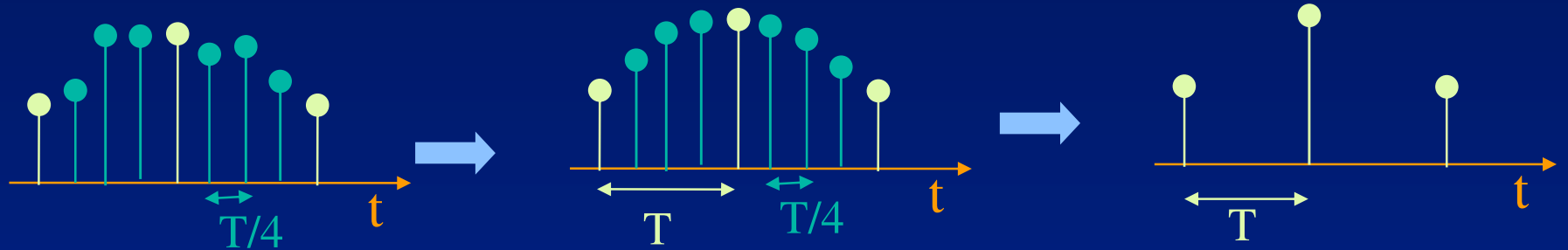
10 Tap 1:2 Interpolating FIR



Interpolating Filter

- ◆ In 1:n interpolation, we have n output sample for every 1 input samples. Pay attention to clocking scheme and the timing diagram.
- ◆ A Interpolation Filter can exploit the symmetrical nature of the coefficients by rearranging the Timing Skew Buffer to the inputs of DALUTs. The number of DALUTs reduce by half.

Decimation Filter



4:1 Decimation Filter

- ◆ For 4:1 decimation, sample frequency changes from F_s to $0.25F_s$.



— Low Pass Filter Response to remove the frequency component beyond $0.125F_s$

8 Tap 4:1 Decimating FIR

- ◆ Consider four consecutive outputs

$$y(4n) = C_0x(n) + C_1x(n-1) + C_2x(n-2) + C_3x(n-3) + C_4x(n-4) + C_5x(n-5) + C_6x(n-6) + C_7x(n-7)$$

$$y(4n-1) = C_0x(n-1) + C_1x(n-2) + C_2x(n-3) + C_3x(n-4) + C_4x(n-5) + C_5x(n-6) + C_6x(n-7) + C_7x(n-8)$$

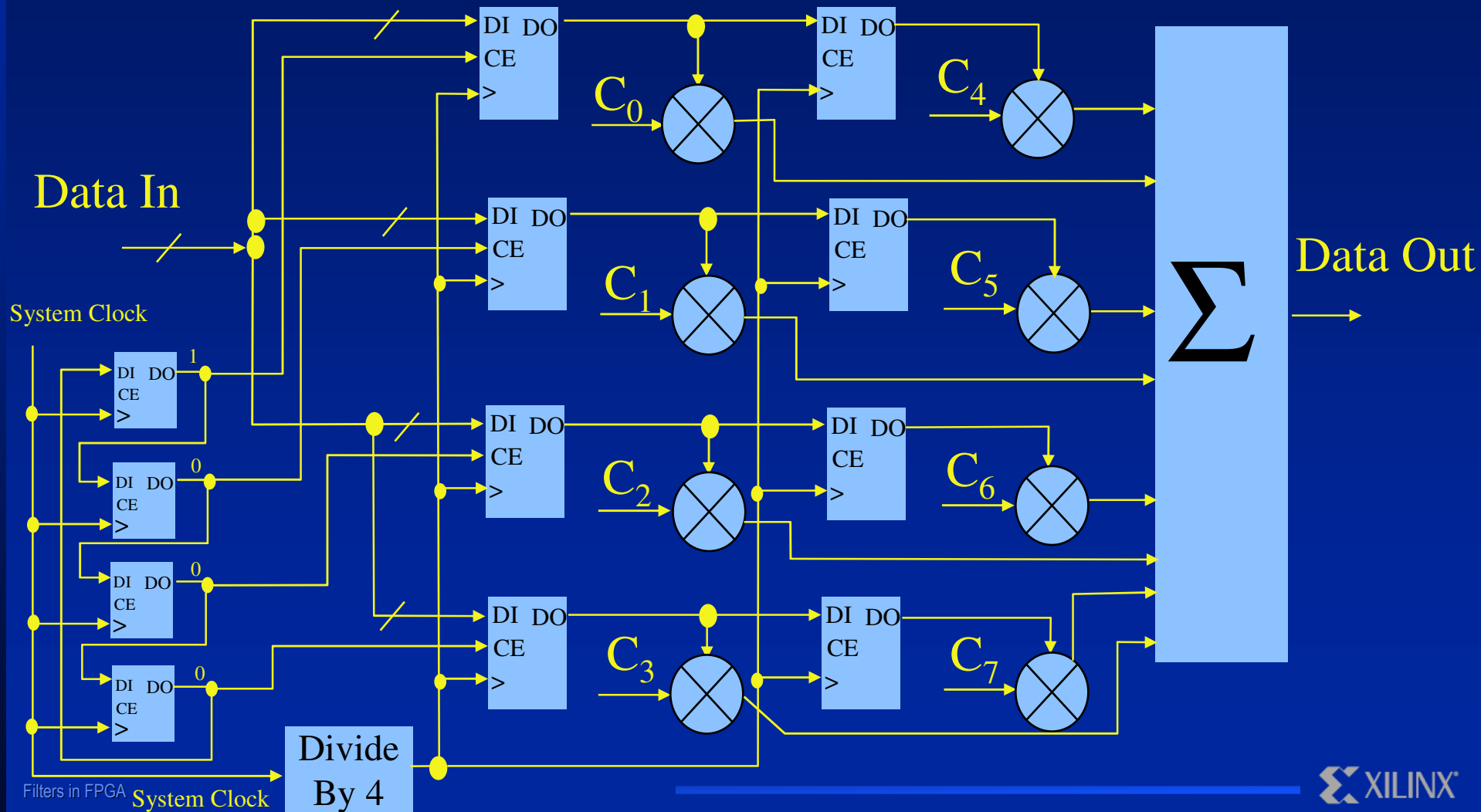
$$y(4n-2) = C_0x(n-2) + C_1x(n-3) + C_2x(n-4) + C_3x(n-5) + C_4x(n-6) + C_5x(n-7) + C_6x(n-8) + C_7x(n-9)$$

$$y(4n-3) = C_0x(n-3) + C_1x(n-4) + C_2x(n-5) + C_3x(n-6) + C_4x(n-7) + C_5x(n-8) + C_6x(n-9) + C_7x(n-10)$$

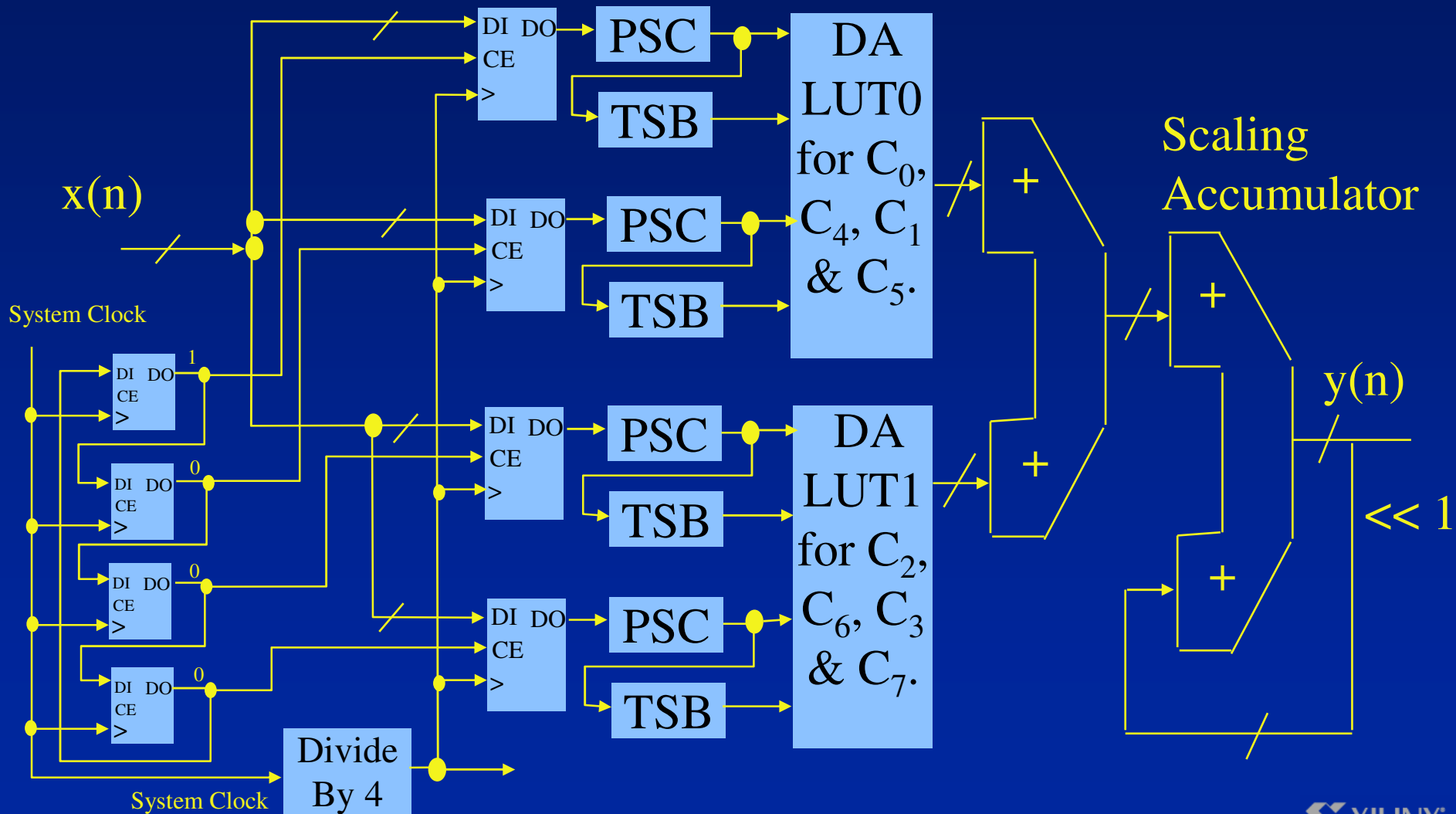
$$y(4n-4) = C_0x(n-4) + C_1x(n-5) + C_2x(n-6) + C_3x(n-7) + C_4x(n-8) + C_5x(n-9) + C_6x(n-10) + C_7x(n-11)$$

- ◆ Downsampler accepts one sample ($y(4n)$) and rejects three samples ($y(4n-1)$, $y(4n-2)$ and $y(4n-3)$). Do not do the extra multiplies.

8 Tap 4:1 Decimating FIR



8 Tap 4:1 Decimating FIR



Decimating Filter

- ◆ In $n:1$ decimation, we have 1 output sample for every n input samples. Pay attention to clocking scheme and the timing diagram.
- ◆ A Decimation Filter can exploit the symmetrical nature of filter coefficients by rearranging the Timing Skew Buffer and adding Multiplexers to the inputs of DALUTs. The number of DALUTs reduce by half.

Adaptive FIR Filters

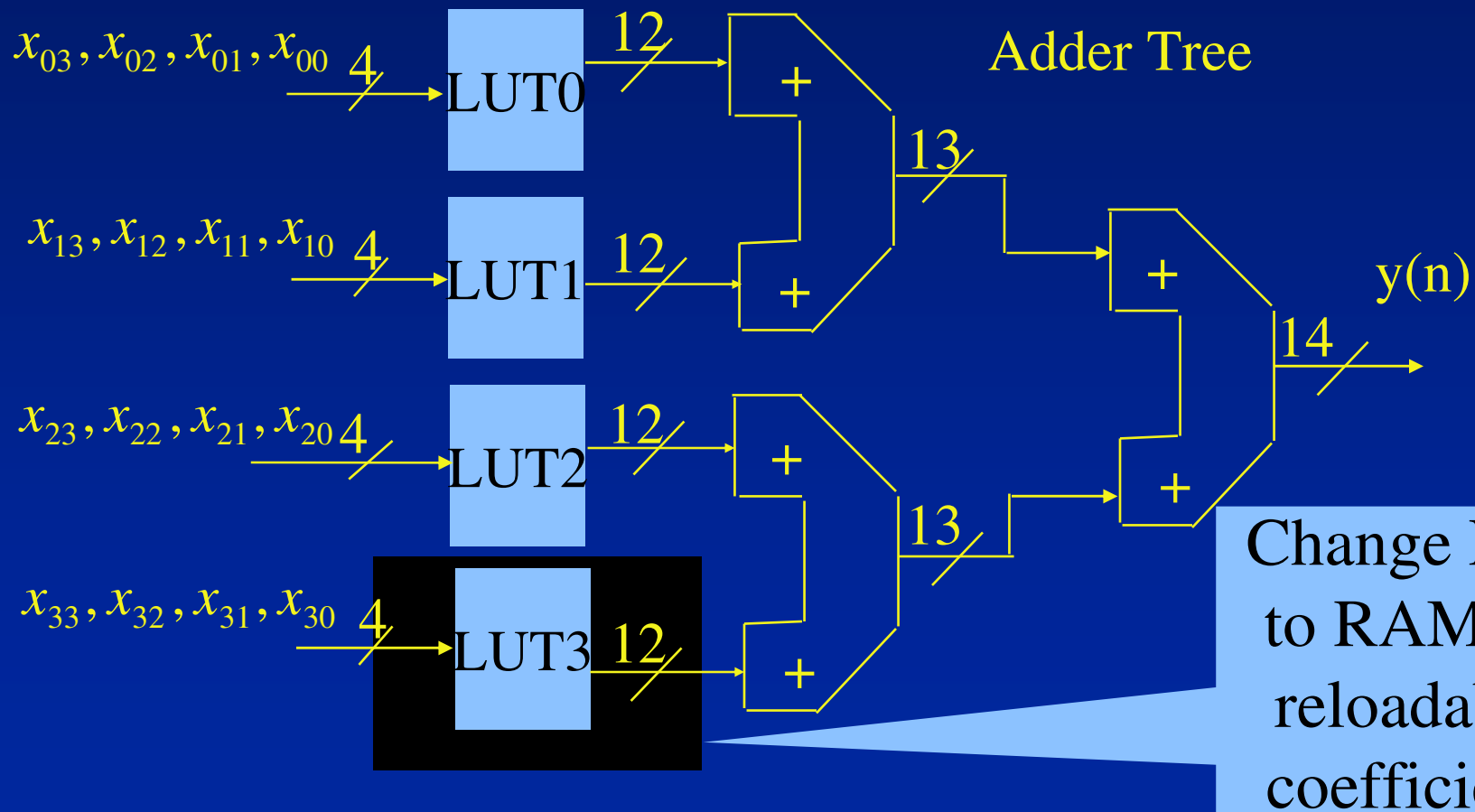
- ◆ Coefficients change according to an algorithm
- ◆ Pay Attention to
 - Delays in the Feedback Path
 - Latency in the Coefficient Updating Logic
 - How coefficients get updated
- ◆ Design of Adaptive Filters closely related with the application where it gets used

Implementations of Adaptive Filter

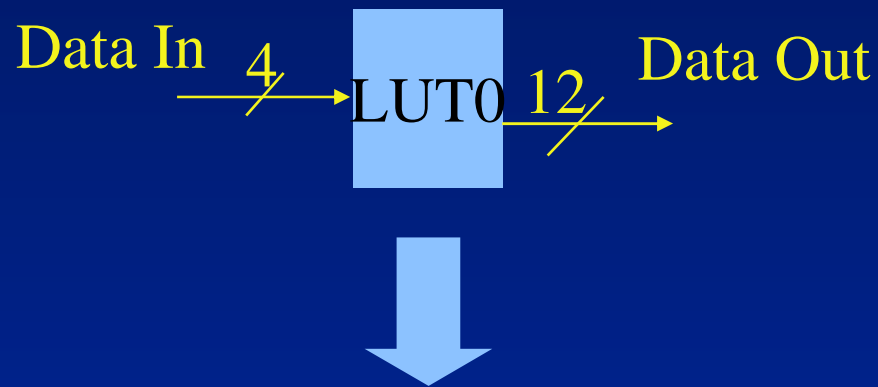
- ◆ Usage of Efficient Variable Multiplier is a viable option in Virtex
- ◆ The Distributed Arithmetic Implementations can be made reloadable by using Distributed RAM or Shift Register LUT instead of LUTs
 - Logic to update the coefficients depend on the DA Implementation
 - Two copies of DA RAMs allow filter operation in one RAM while other gets updated. Switch between two DA RAMs appropriately

Fully Parallel Four Input Bit Implementation for 4 Tap FIR

- ◆ Assume coefficient width is 8 and input bit width is 4



Reloadable Coefficients

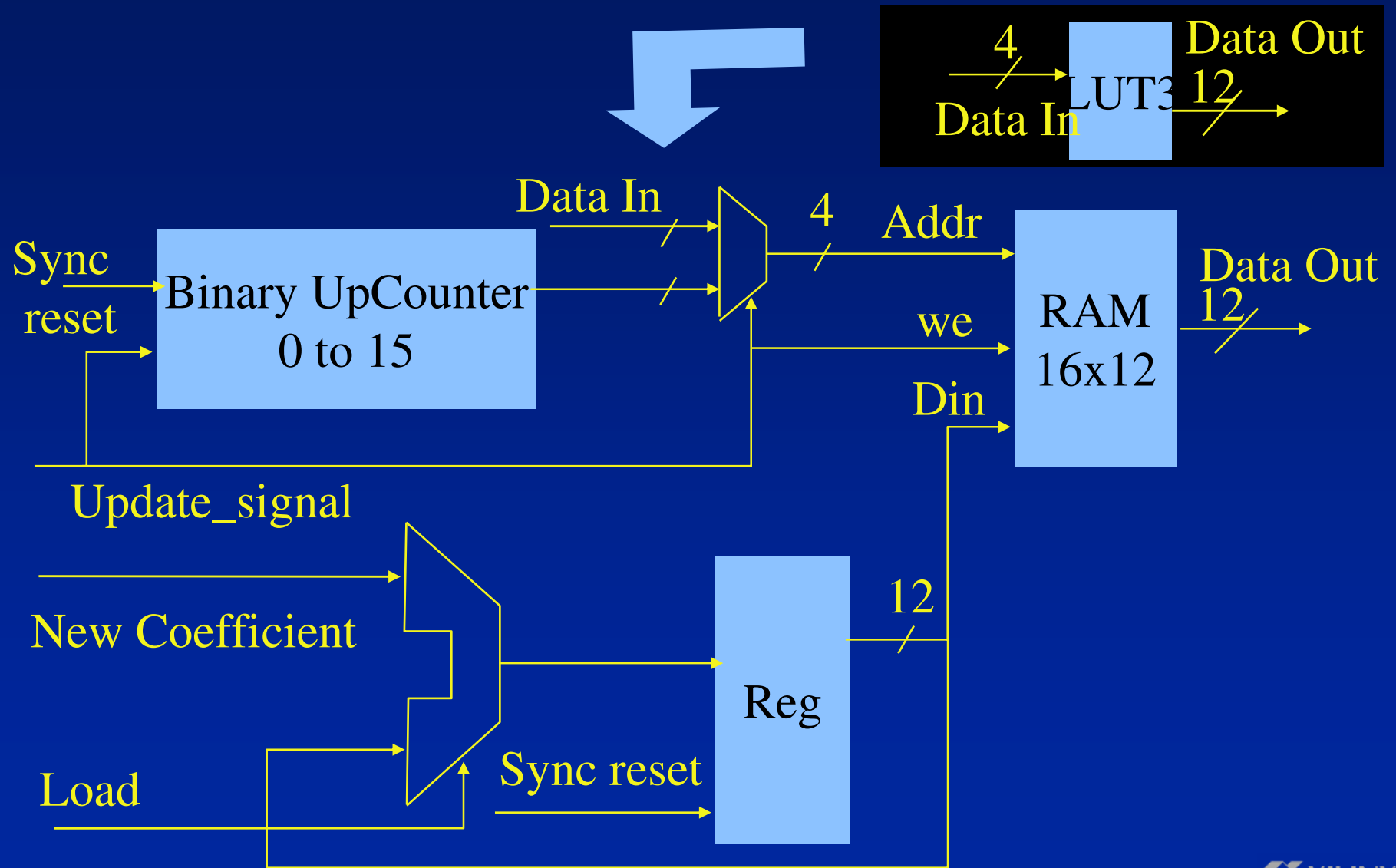


Change it to a
reloadable RAM
based Multiplier

Stored Data Values

0	0	8	$8 \cdot C0$
1	$C0$	9	$9 \cdot C0$
2	$2 \cdot C0$	10	$10 \cdot C0$
3	$3 \cdot C0$	11	$11 \cdot C0$
4	$4 \cdot C0$	12	$12 \cdot C0$
5	$5 \cdot C0$	13	$13 \cdot C0$
6	$6 \cdot C0$	14	$14 \cdot C0$
7	$7 \cdot C0$	15	$15 \cdot C0$

Reloadable Coefficients



DA Implementations of Adaptive Filter

- ◆ Updating typically takes 16 cycles for a RAM16x1 implementation
- ◆ Avoid Address Multiplexing by using Shift Register LUT instead of RAM16x1 in Virtex
- ◆ Two copies of DA RAMs allow filter operation in one RAM while other gets updated. Switch between two DA RAMs appropriately
 - Look at <http://www.xilinx.com/xapp/xapp055.pdf> for an example implementation

Conclusion

- ◆ FPGA Architectures are nicely suited for Filter Applications
- ◆ Design Tricks explained the area-speed tradeoffs
- ◆ Distributed Arithmetic is key to a lot of Filter Implementations
- ◆ Do Filters in FPGA and you will

