ECE 428 Programmable ASIC Design

FPGAs in DSP Applications

Haibo Wang
ECE Department
Southern Illinois University
Carbondale, IL 62901

Overview

❑ Motivation

- ➢ Digital Signal Processing (DSP) is one of the most active area in VLSI applications
- ➢ Traditionally, DSP algorithms are implemented either using general purpose DSP processors (**Low speed, less expensive, flexible**) or using ASICs (**High speed, expensive, less flexible**)
- ➢ FPGAs provide solutions that maintain both the advantages of the approach based on DSP processors and the approach based on ASICs

❑ Outline

- ➢ Number Systems
  - — Fixed-Point Number System
  - — Floating-Point Number System
- ➢ VLSI Architectures for DSP Circuits
- ➢ Distributed Arithmetic Circuits

Fixed-Point Number System

- Binary number representation of fixed-point numbers

"Binary point"

$$b_n\, b_{n-1}\, b_{n-2} \bullet\bullet\bullet b_0 \bullet b_{-1}\, b_{-2} \bullet\bullet\bullet b_{-m}$$

$$= \sum_{i=1}^{n} b_i \bullet 2^i + \sum_{j=1}^{m} b_j \bullet 2^j$$

- Examples:

Binary		Decimal	Decimal	Binary
101.0101	⟹	5.3125	1.25	1.01
000.011	⟹	0.375	1.24	1.001111010•••

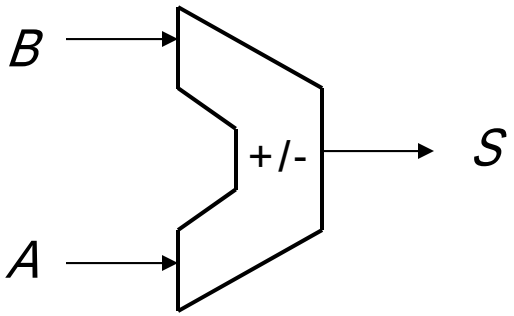
- ✓ If the binary number can have 8 bits for fractional part, we can use 1.00111101 (1.23828125) to approximate 1.24
- ✓ If the binary number can have 7 bits for fractional part, we can use 1.0011111 (1.2421875) to approximate 1.24

Arithmetic Operations for Fixed-Point Numbers

- Add/Subtract

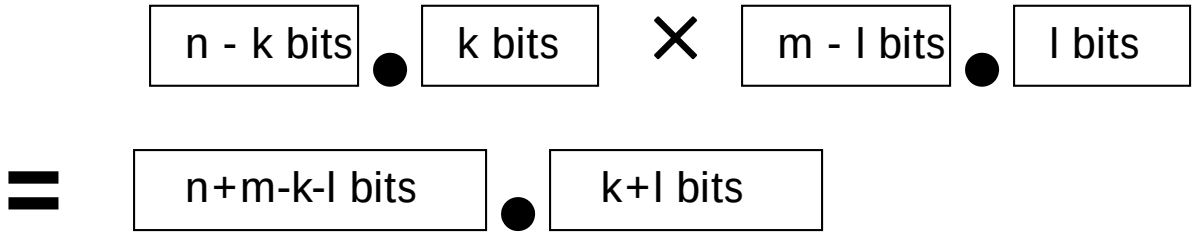
$$
\begin{array}{lll}
 & b_n\, b_{n-1}\, b_{n-2} \bullet\bullet\bullet b_0 \bullet b_{-1}\, b_{-2} \bullet\bullet\bullet b_{-m} & B \\
+/- & a_n\, a_{n-1}\, a_{n-2} \bullet\bullet\bullet a_0 \bullet a_{-1}\, a_{-2} \bullet\bullet\bullet a_{-m} & A \\
\hline
c & s_n\, s_{n-1}\, s_{n-2} \bullet\bullet\bullet s_0 \bullet s_{-1}\, s_{-2} \bullet\bullet\bullet s_{-m} & S
\end{array}
$$

- The addition or subtraction of two fixed-point numbers can be performed by regular adder or subtracter if the "binary points" of the two numbers are aligned. The "binary point" remains the same position in the resulted number.



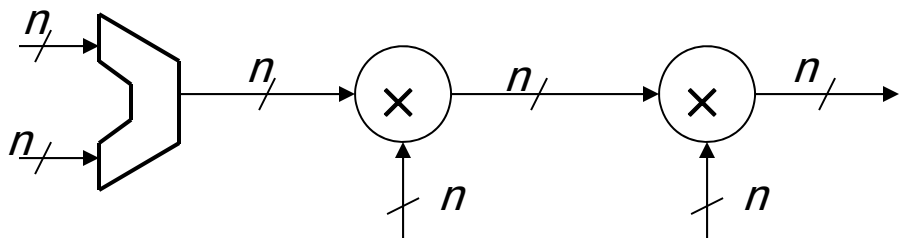
Arithmetic Operations for Fixed-Point Numbers

❑ Multiplication



- For the convenience of hardware implementation, we prefer to have the product of a multiplication keeping the same length as the multiplicand or the multiplier (assume they have the same length). To achieve this, we normally truncate the least significant bits of the product.

❑ Arithmetic operation with fixed word-length

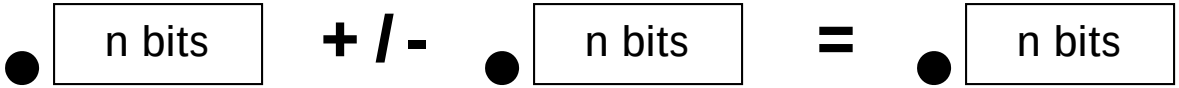


Arithmetic Operations for Fixed-Point Numbers

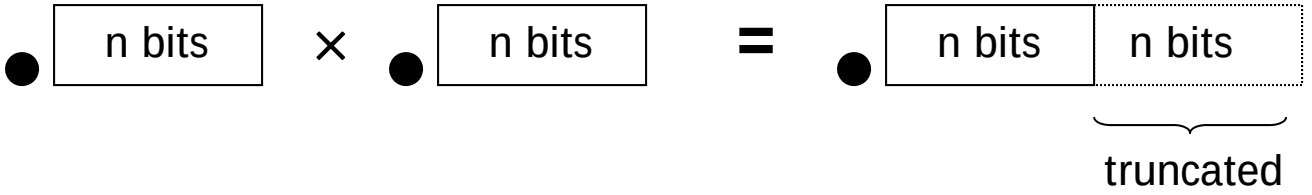
❑ Normalized fixed-point numbers

- Scaling all the numbers involved in computation by a factor K such that all the numbers are within the range from 0 to 1
- Fixed-point number after normalization

- Addition/Subtraction

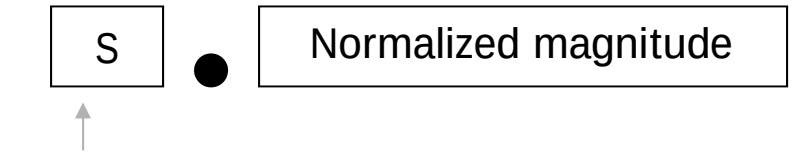


- Multiplication



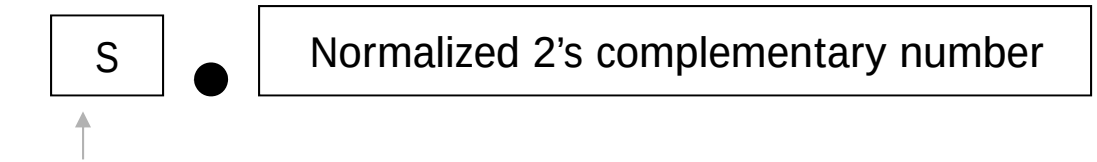
Representation of Negative Numbers

- Signed-magnitude numbers



Sign bit: 0 for positive number and 1 for negative number

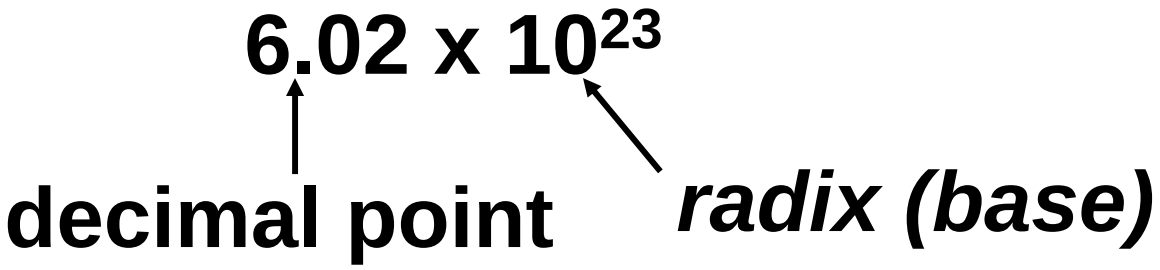
- 2's complementary numbers



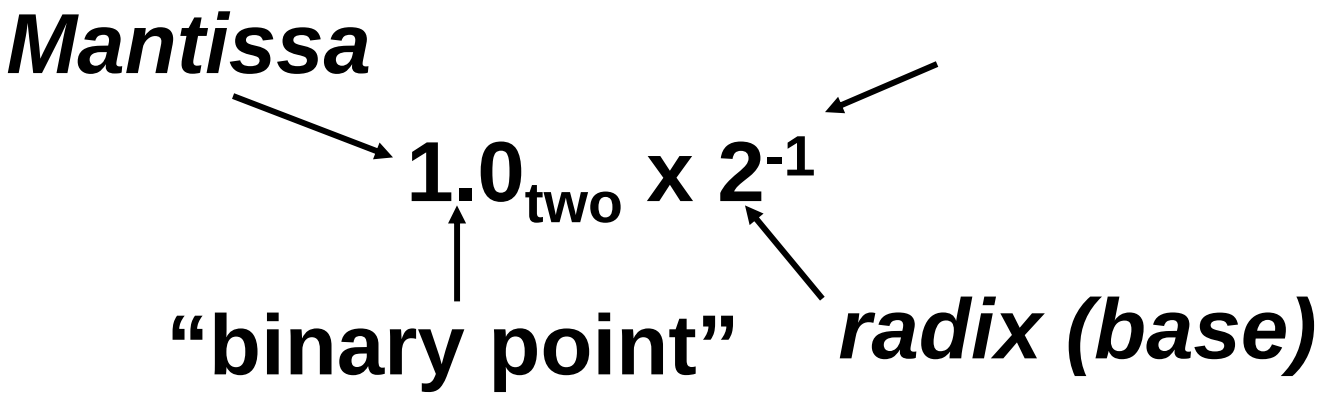
Sign bit: 0 for positive number and 1 for negative number

Floating-Point Numbers

- Scientific Notation



- Binary Floating-Point Numbers



Floating-Point Representation

- Normal format: $+1.xxxxxxxxxx_{two} * 2^{yyyy_{two}}$

S	Exponent	Significand

- S represents Sign
  - (1 for negative number and 0 for positive number)
- Exponent represents yyyy
  - (It is a biased number, is is also called as excess-bias number. E.g. if a number A is a excess-8 coding, the real value of the number is A-8)
- Significand represents xxxxxxxxx

$$(-1)^S * (1 + \text{Significand}) * 2^{(\text{Exponent} - \text{Bias})}$$

Arithmetic Operations of Floating-Point Numbers

- Assume number $X = X_M \bullet 2^{X_E}$ $Y = Y_M \bullet 2^{Y_E}$

- Addition/Subtraction:

$$X \pm Y = (X_M \bullet 2^{X_E - Y_E} \pm Y_M) \bullet 2^{Y_E}$$ Where $X_E < Y_E$

1. Compute Y_E-X_E, a fixed-point subtraction
2. Right shift X_M by Y_E-X_E bits to obtain $X_M \bullet 2^{Xe-Ye}$
3. Compute $X_M \bullet 2^{Xe-Ye} \pm Y_M$, a fixed-point addition or subtraction

- Multiplication:

$$X \bullet Y = (X_M \bullet Y_M) \bullet 2^{X_E + Y_E}$$

1. Compute $X_M \bullet Y_M$, a fixed-point multiplication
2. Compute $X_E + Y_E$, a fixed-point addition

DSP Applications

- Common DSP Functions that are implemented using VLSIs
  - Filters (FIR, IIR)
  - Fast Fourier Transform (FFT)
  - Direct Cosine Transform (DCT)
  - Encoder/decoder and error correction/detection functions
  - • • • • •
- FIR (Finite Impulse Response) Filter

$$y[n] = a_0 \bullet x[n] + a_1 \bullet x[n-1] + \cdots\cdots + a_k \bullet x[n-k]$$

1. Y[n] is the output at n*th* clock cycle; X[n] is the input at n*th* clock cycle
2. a_0, a_1, ... a_{k-1} are filter coefficients

- IIR (Infinite Impulse Response) Filter

$$y[n] = a_0 \bullet x[n] + \cdots + a_k \bullet x[n-k] + b_1 \bullet y[n-1] + \cdots + b_m \bullet y[n-m]$$

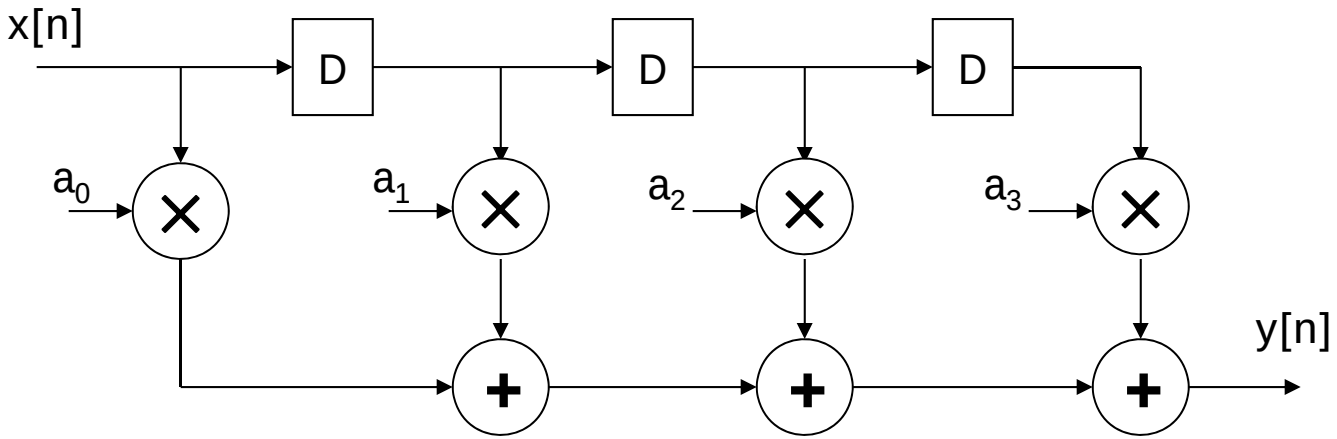
FIR Filter Implementation

- Example:

$$y[n] = \underbrace{a_0 \bullet x[n]}_{\text{Tap}} + a_1 \bullet x[n-1] + a_2 \bullet x[n-2] + a_3 \bullet x[n-3]$$

- This is a 4-tap FIR filter

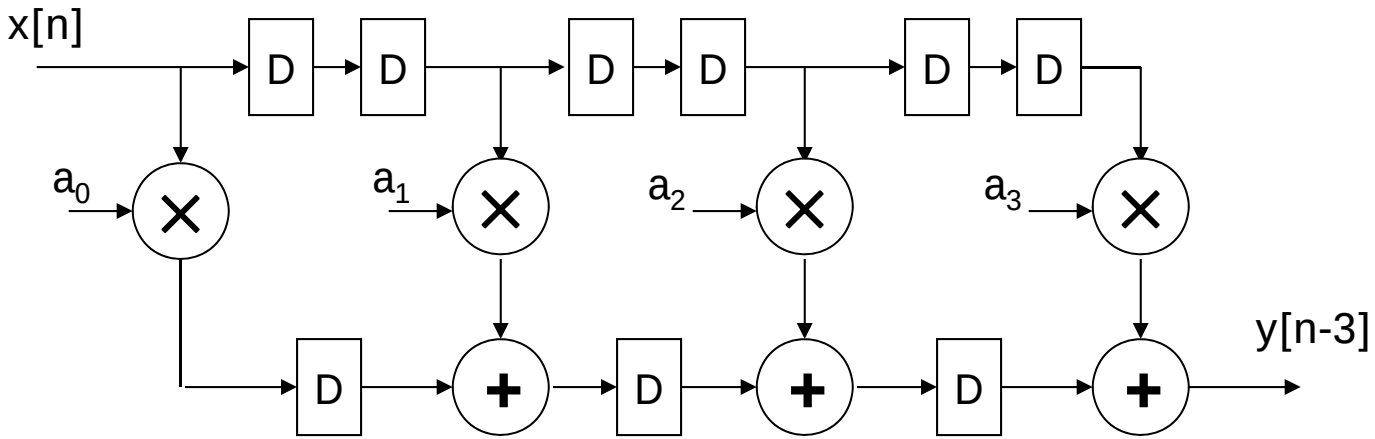
- Canonic form implementation:



Clock frequency $$f_{clk} \le \frac{1}{t_{mult} + 3 \bullet t_{adder}}$$

FIR Filter Implementation

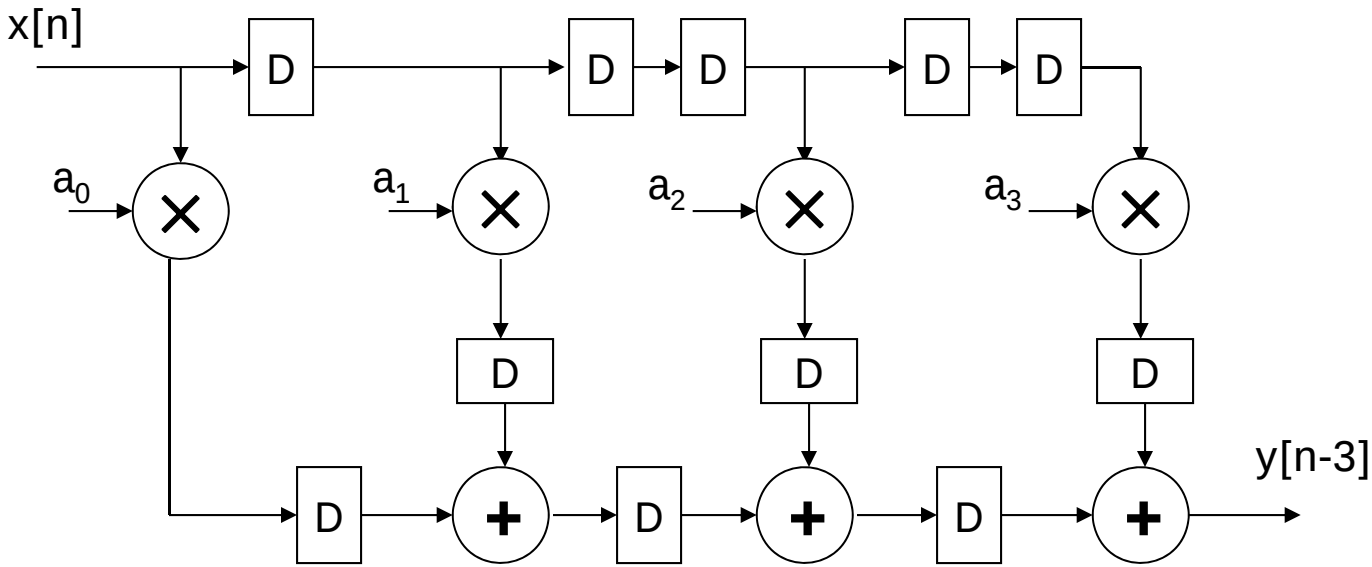
- Pipelined implementation 1:



Clock frequency $$f_{clk} \leq \frac{1}{t_{mult} + t_{adder}}$$

FIR Filter Implementation

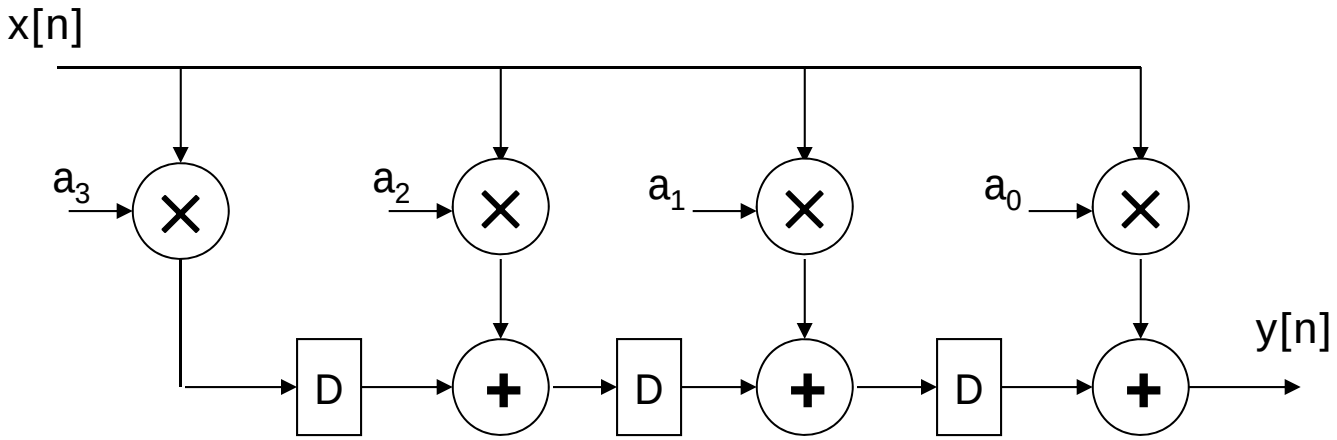
- Pipelined implementation 2:



Clock frequency $f_{clk} \leq \frac{1}{t_{mult}}$ (assume $t_{mult} > t_{add}$)

FIR Filter Implementation

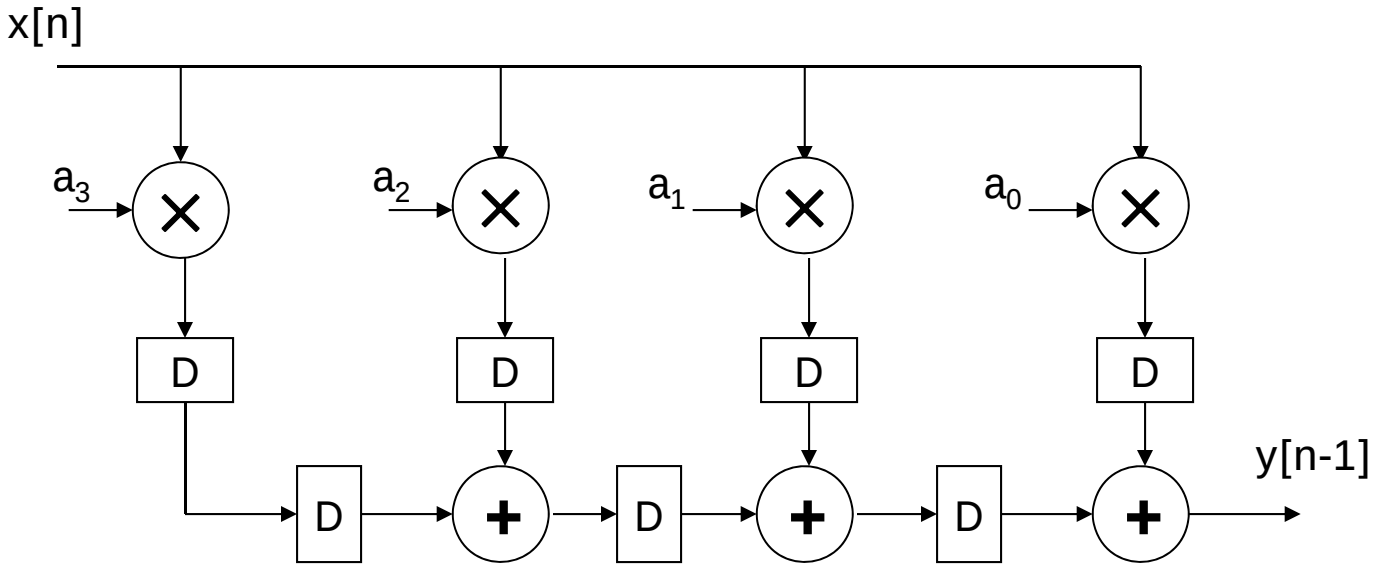
- Pipelined implementation 3 (inverted form):



Clock frequency $$f_{clk} \leq \frac{1}{t_{mult} + t_{adder}}$$

FIR Filter Implementation

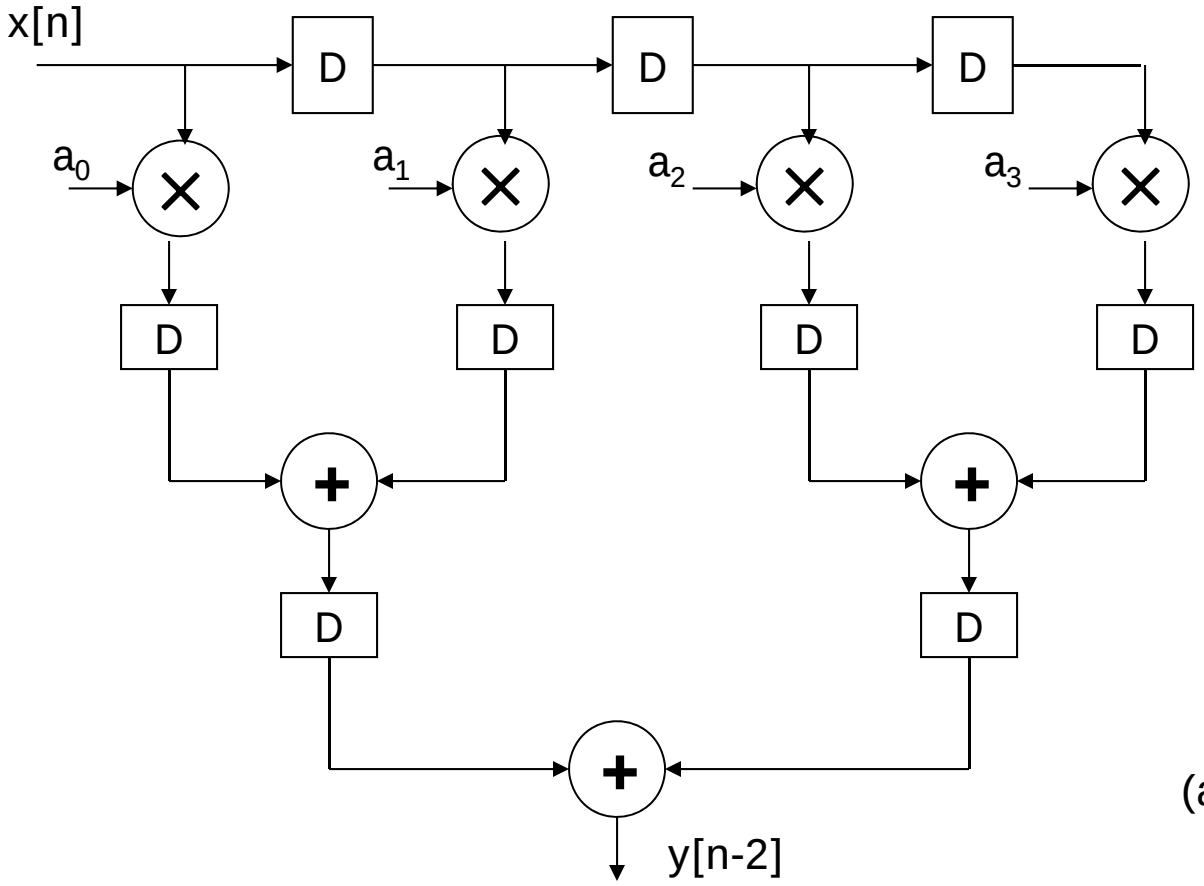
- Pipelined implementation 4:



Clock frequency $f_{clk} \leq \frac{1}{t_{mult}}$ (assume $t_{mult} > t_{add}$)

FIR Filter Implementation

- Pipelined implementation 5:



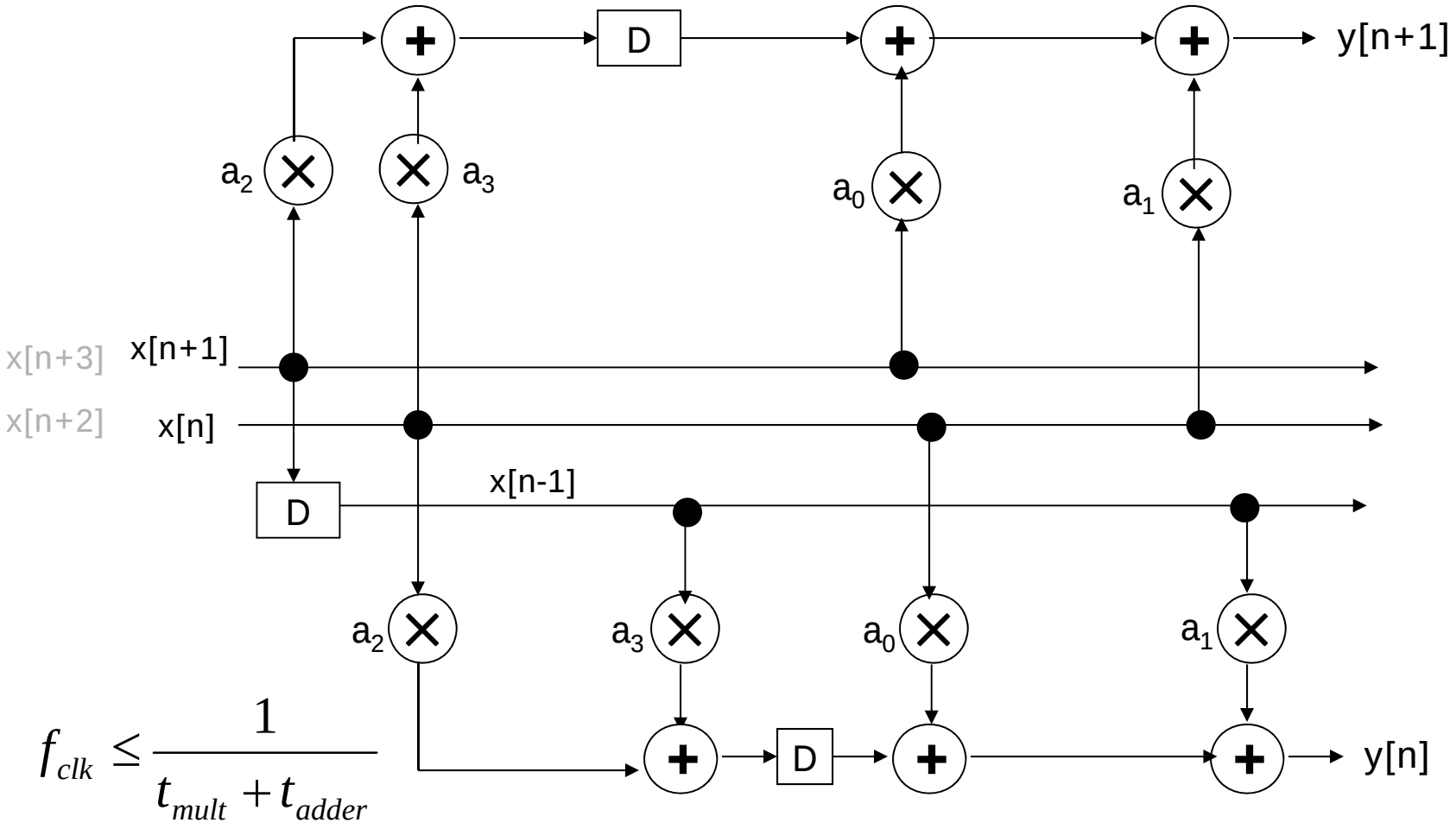
$$f_{clk} \leq \frac{1}{t_{mult}}$$

(assume $t_{mult} > t_{add}$)

- Difficult to layout

FIR Filter Implementation

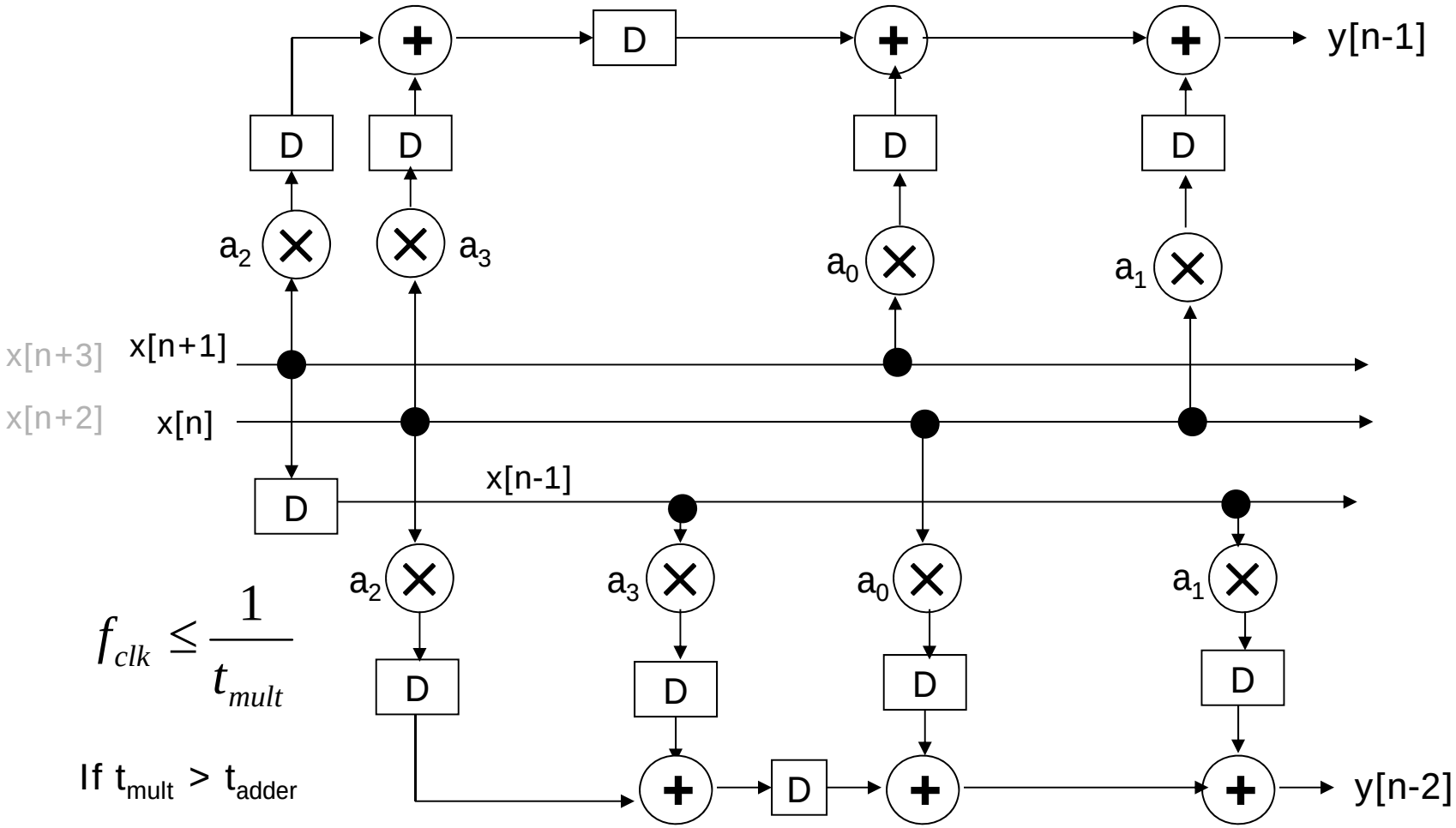
- Parallel implementation 1:



$$f_{clk} \le \frac{1}{t_{mult} + t_{adder}}$$

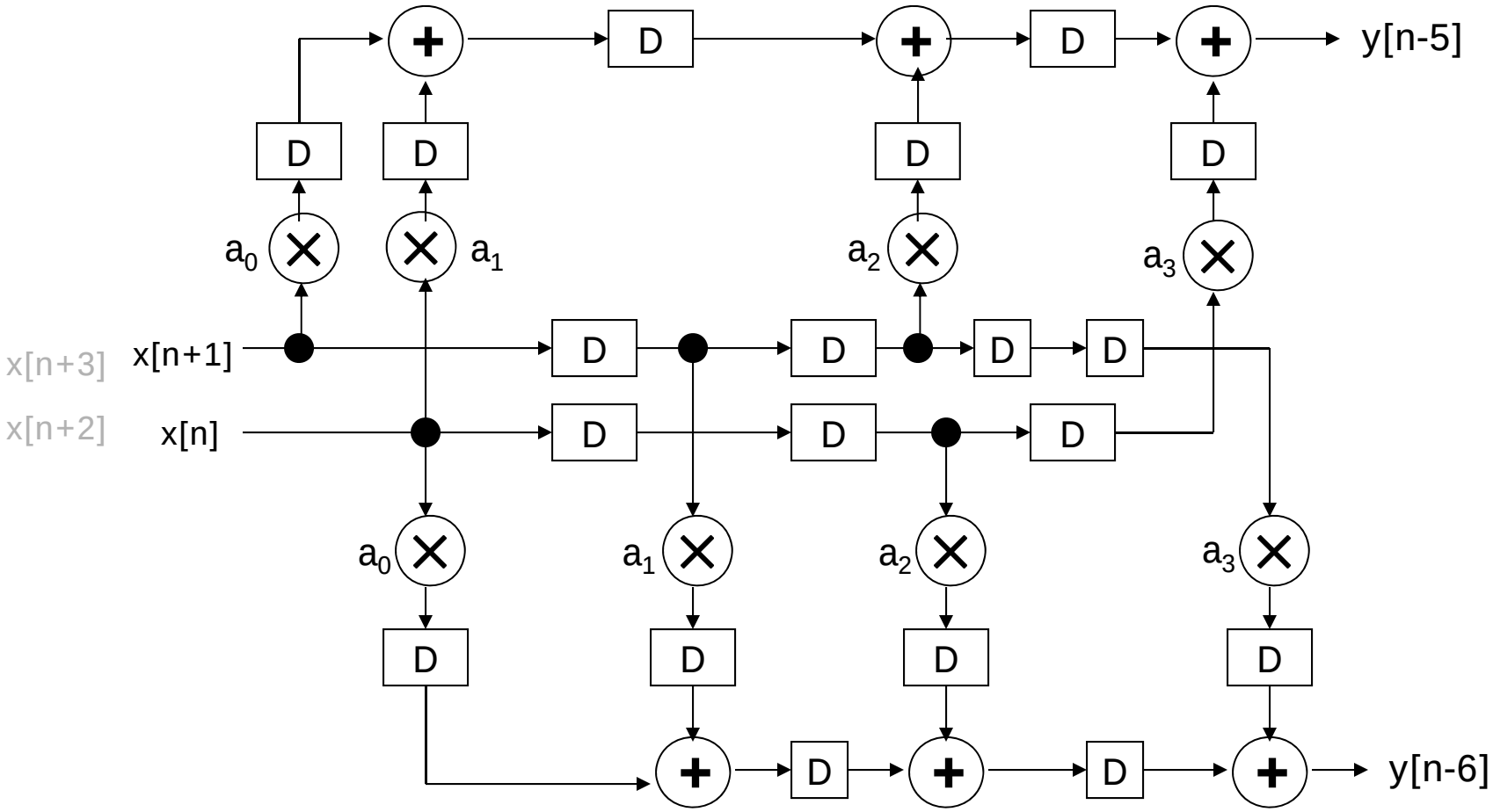
FIR Filter Implementation

- Parallel implementation 2:



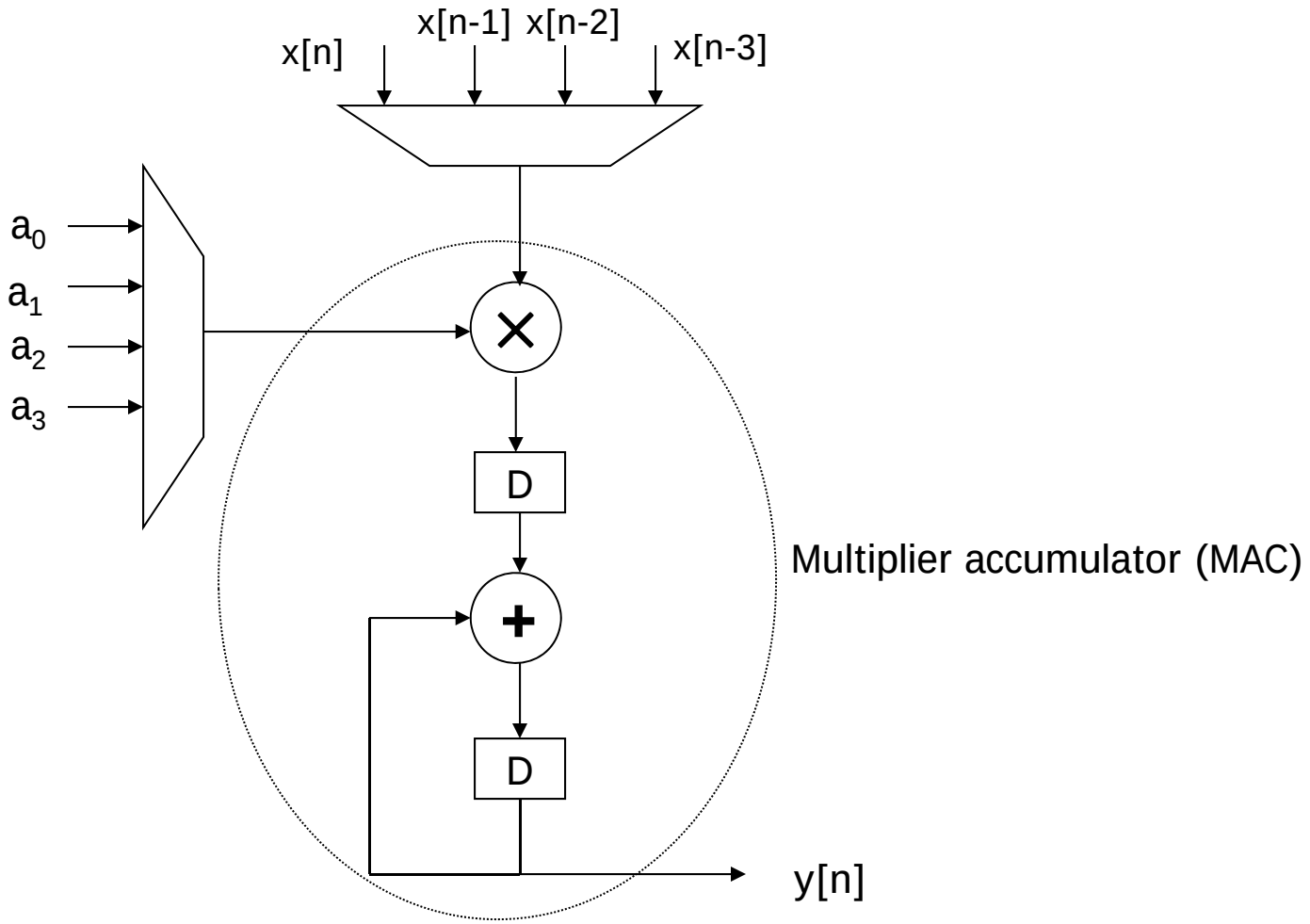
FIR Filter Implementation

- Parallel implementation 3:



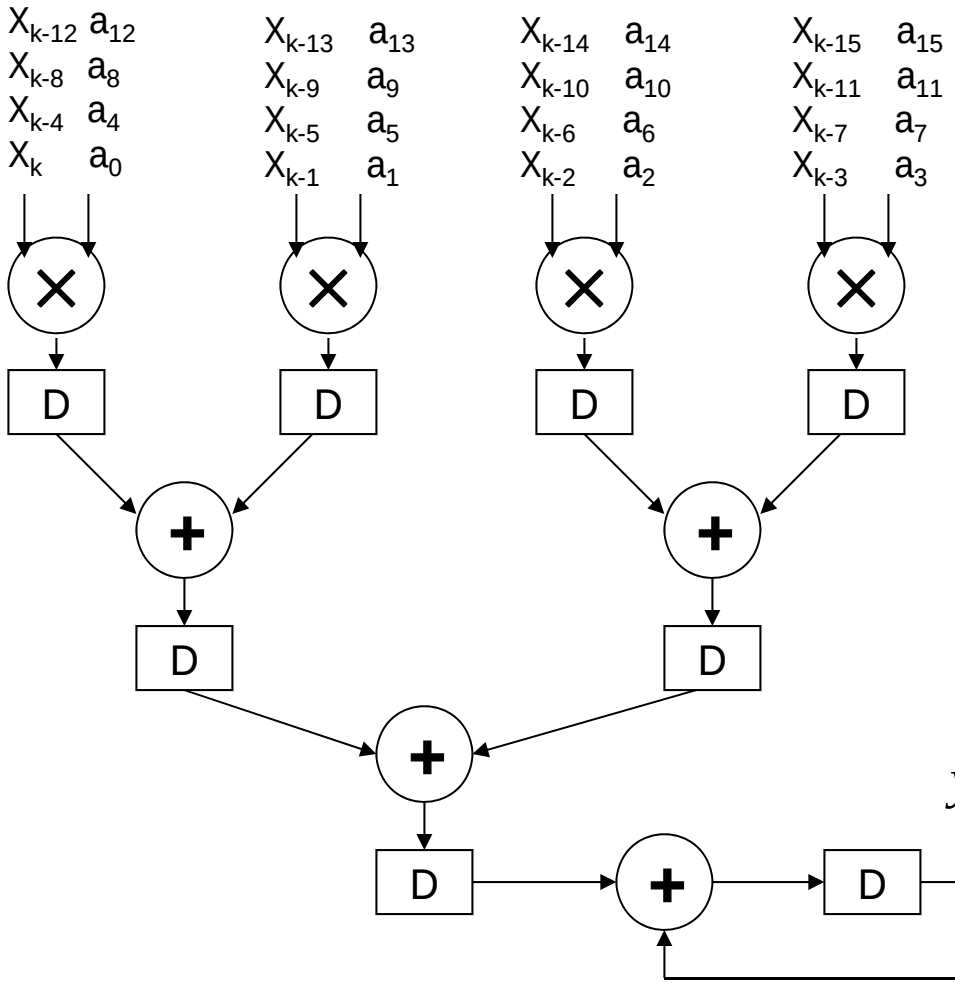
FIR Filter Implementation

- Serial implementation:



FIR Filter Implementation

- Implementation of FIR filters with large number of taps
  - — Examples: implementation of a 16-tap FIR filter



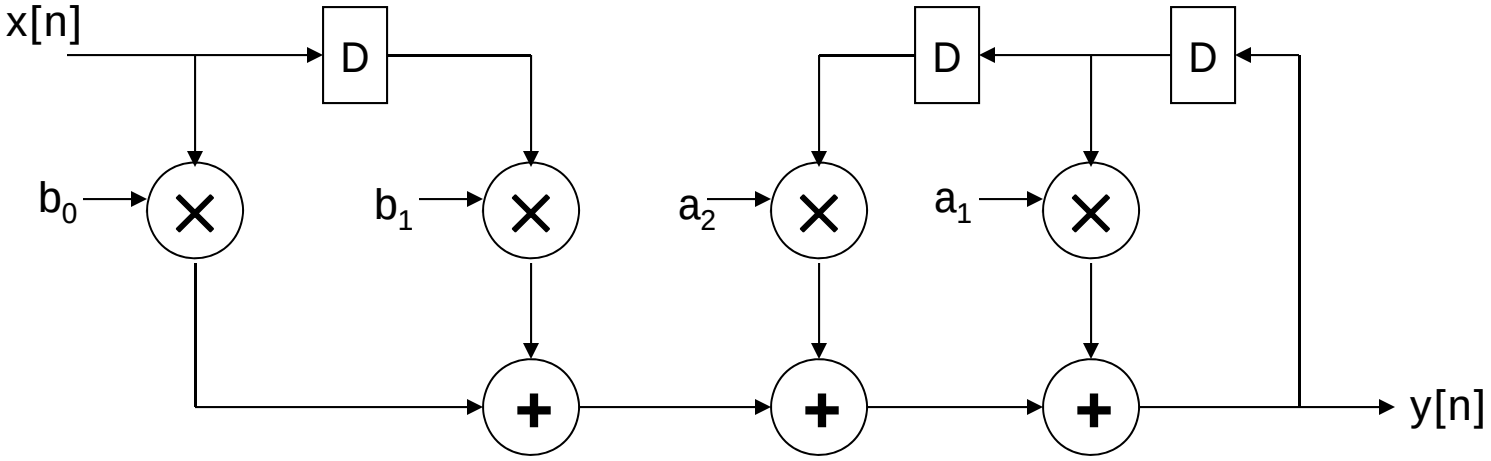
$$y[k] = \sum_{i=0}^{15} a_i \bullet x[k-i]$$

IIR Filter Implementation

- Example:

$$y[n] = b_0 \bullet x[n] + b_1 \bullet x[n-1] + a_1 \bullet y[n-1] + a_2 \bullet x[n-2]$$

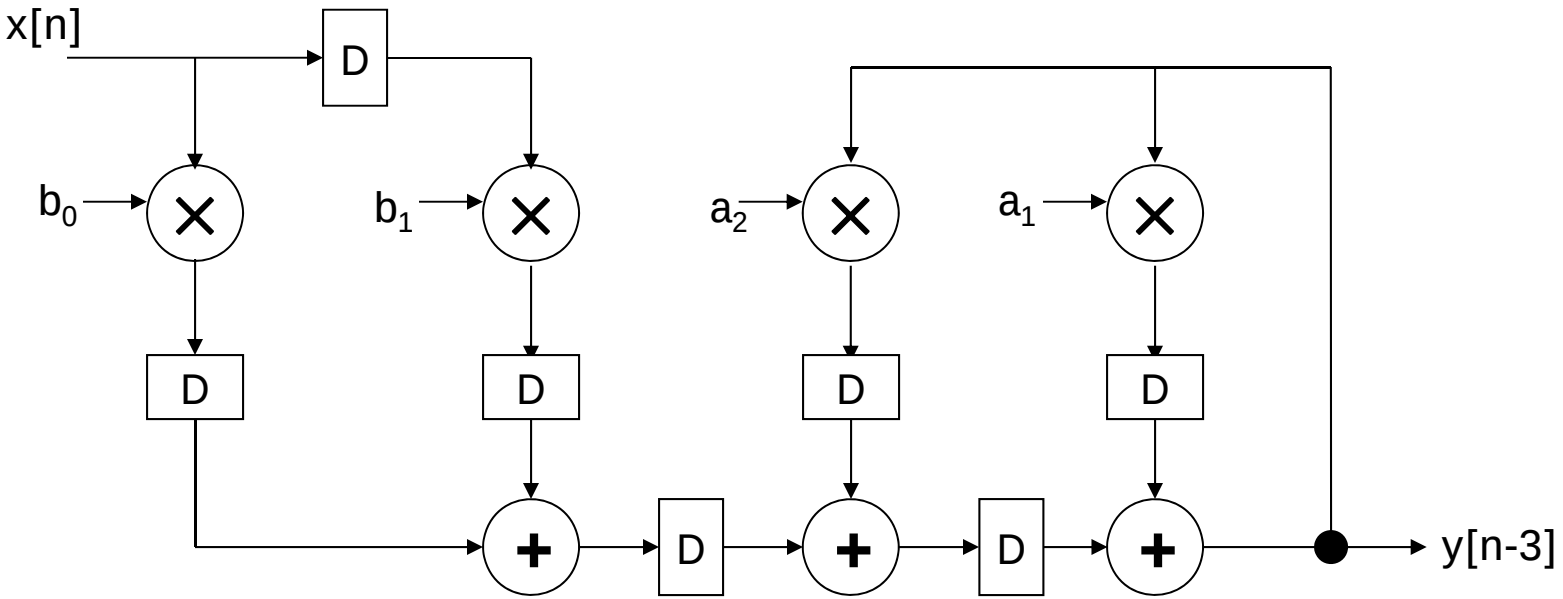
- Direct Implementation:



Clock frequency $$f_{clk} \le \frac{1}{t_{mult} + 3 \bullet t_{adder}}$$

IIR Filter Implementation

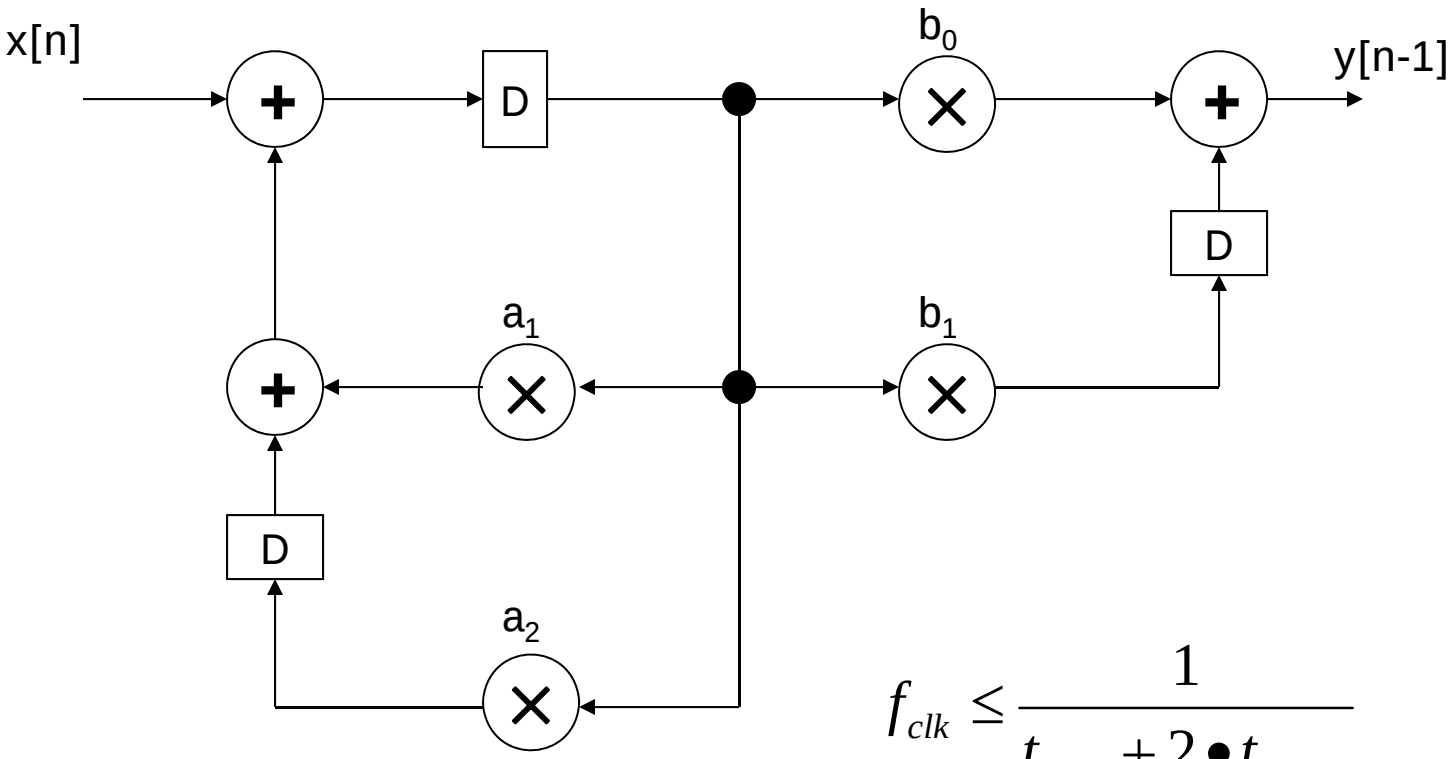
- Pipelined Implementation 1:



Clock frequency $f_{clk} \leq \frac{1}{t_{mult}}$ (assume $t_{mult} > t_{add}$)

IIR Filter Implementation

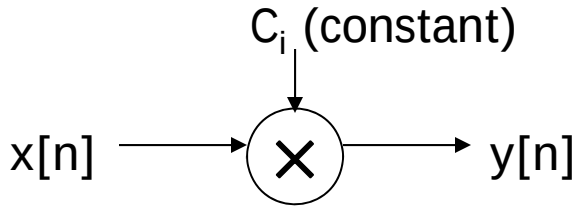
- Pipelined Implementation 2:



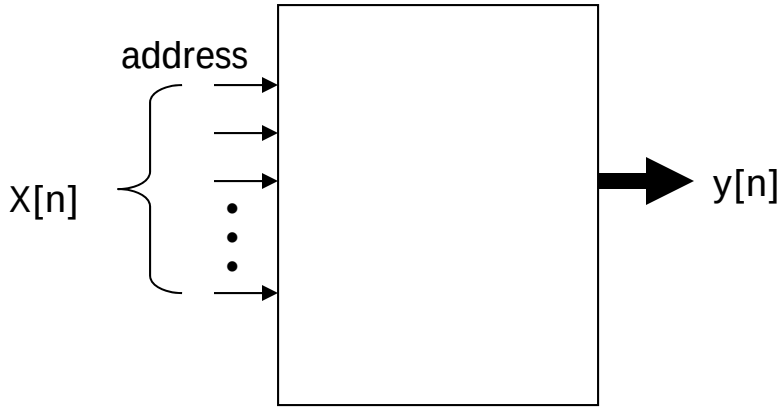
$$f_{clk} \le \frac{1}{t_{mult} + 2 \bullet t_{adder}}$$

LUT-Based Multiplier

- In many DSP circuits, multipliers always have one constant input.



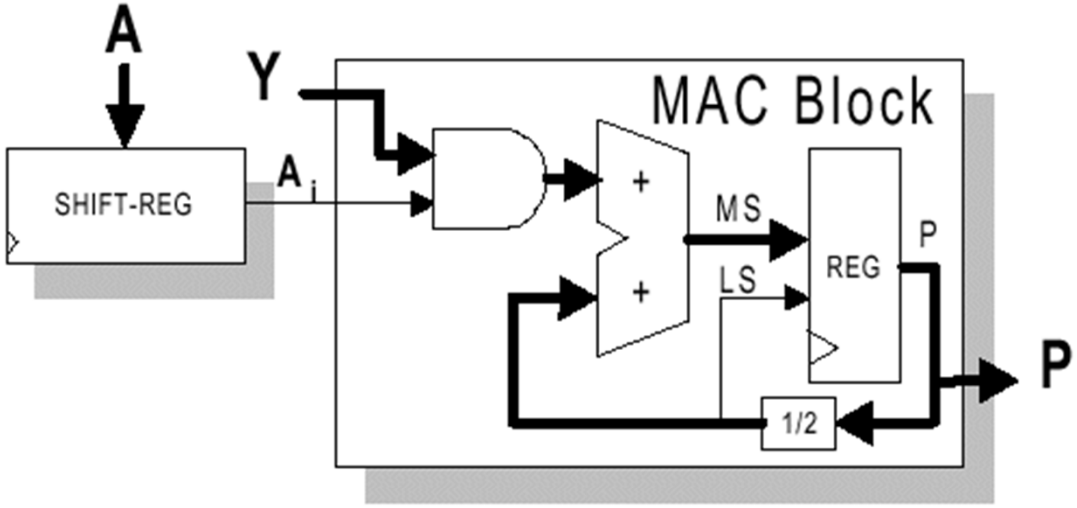
- For the above multiplier, y[n] purely depends on x[n]. Thus, a look-up table (LUT) can be used to implement the multiplier



- For example, a 256×16 bit memory can be used to implement a 8-bit multiplier if one of its input is always constant.

Distributed Arithmetic

- Multiplication by using shift-and-add technique



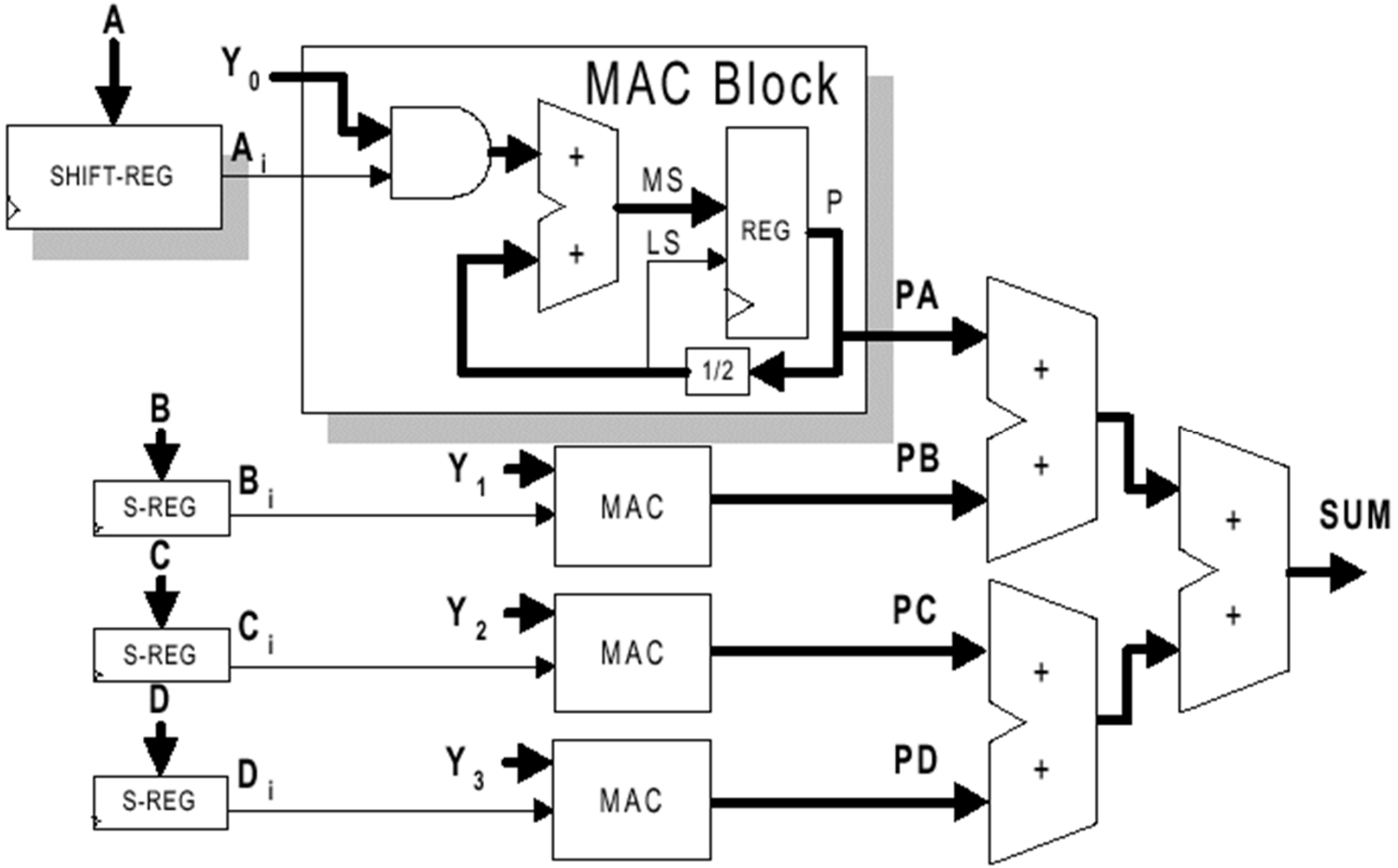
LET **A** = $A_3 A_2 A_1 A_0$ and **Y** = $Y_3 Y_2 Y_1 Y_0$ then, **A*Y = P** is,

$$
\begin{array}{rl}
 & A_0 * (Y_3\ Y_2\ Y_1\ Y_0) \langle 1/2 \rangle \\
+ & A_1 * (Y_3\ Y_2\ Y_1\ Y_0) \langle 1/2 \rangle \\
+ & A_2 * (Y_3\ Y_2\ Y_1\ Y_0) \langle 1/2 \rangle \\
+ & A_3 * (Y_3\ Y_2\ Y_1\ Y_0) \\
\hline
 & P_7\ P_6\ P_5\ P_4\ P_3\ P_2\ P_1\ P_0
\end{array}
$$

{ Start }
{ Cycle=0 }
{ Cycle=1 }
{ Cycle=2 }
{ Cycle=3 }
{ End }

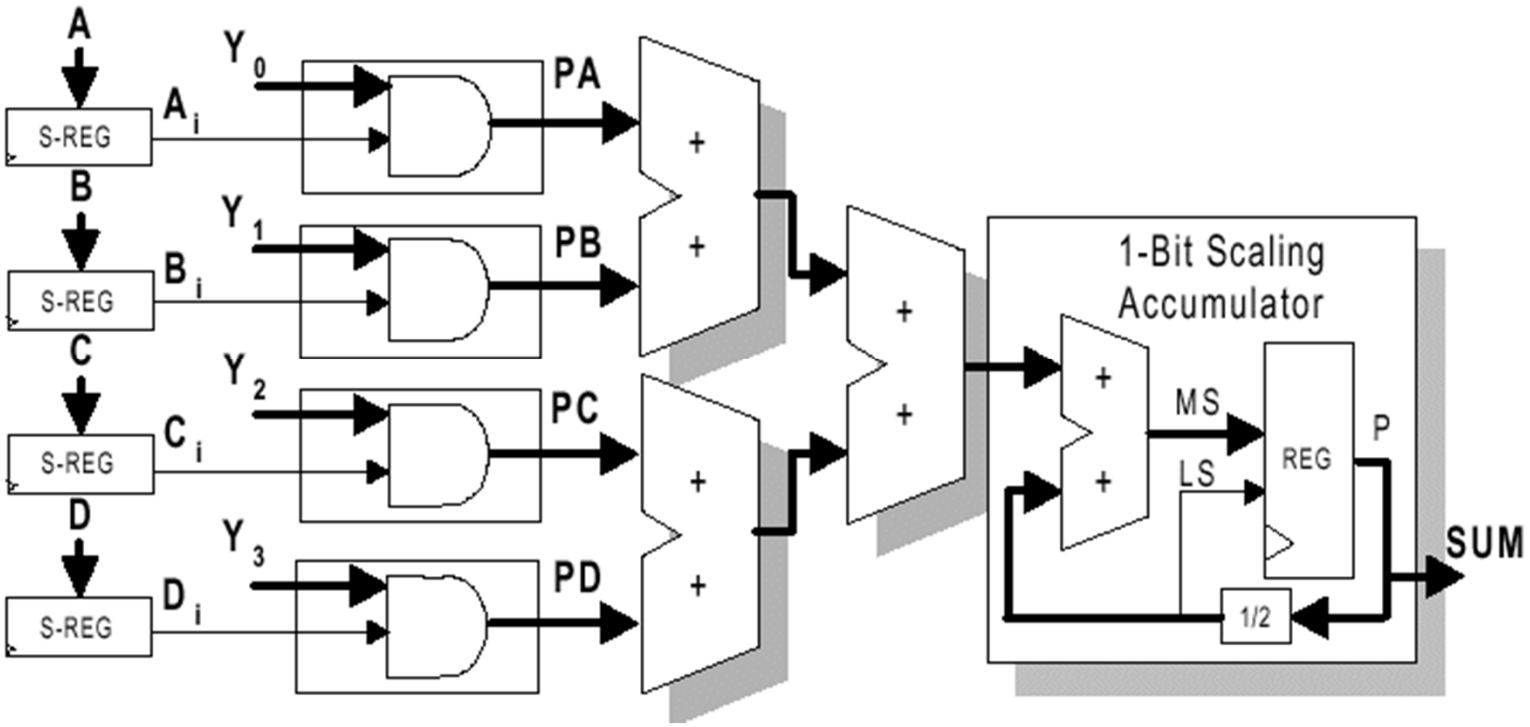
Distributed Arithmetic

- Calculate $A \bullet Y_0 + B \bullet Y_1 + C \bullet Y_2 + D \bullet Y_3$



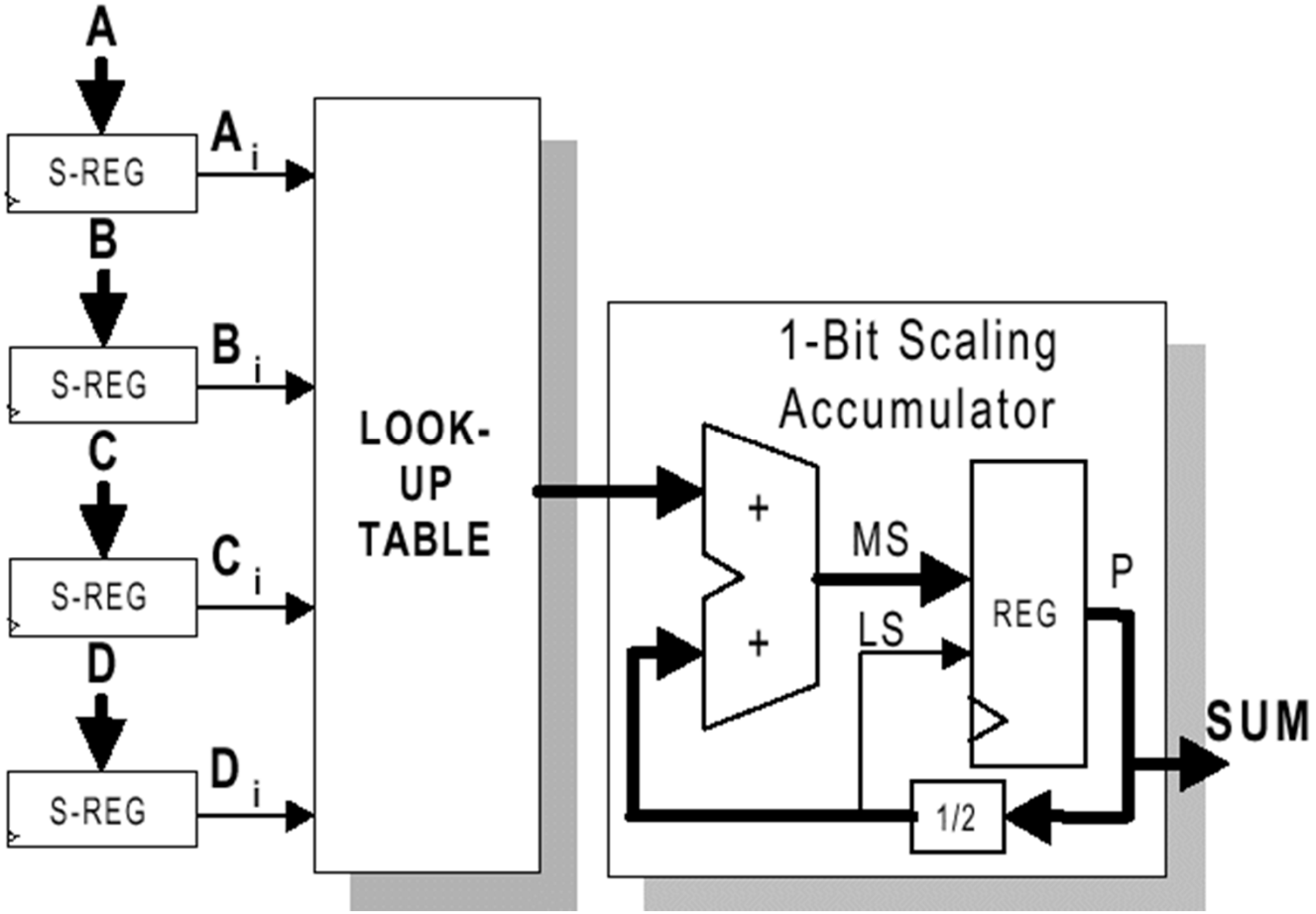
Distributed Arithmetic

- Serial Distributed Arithmetic for Computing $A \bullet Y_0 + B \bullet Y_1 + C \bullet Y_2 + D \bullet Y_3$



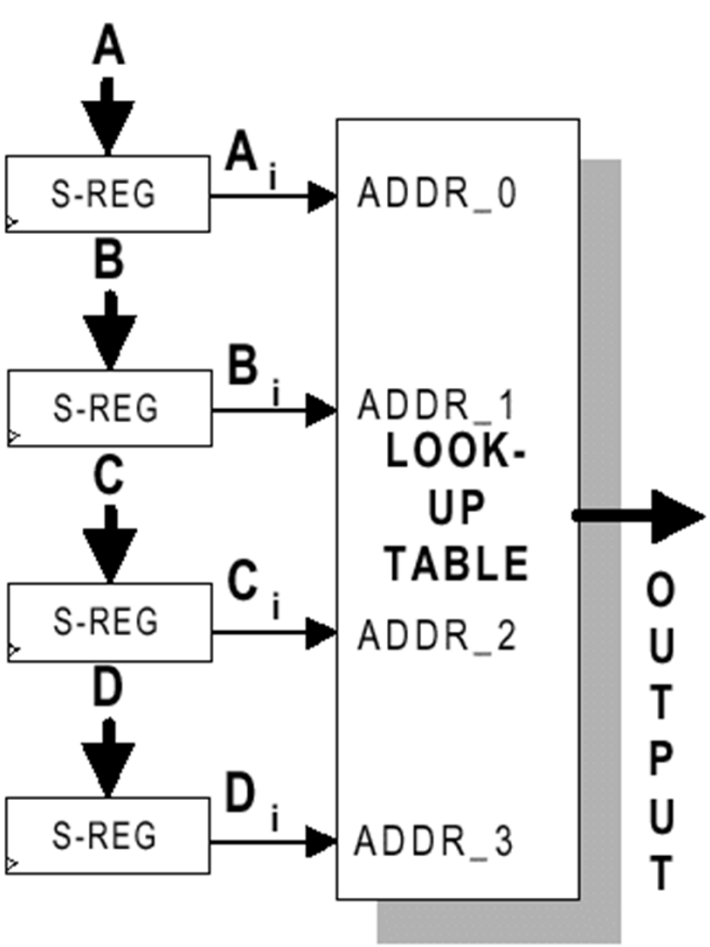
Distributed Arithmetic

- LUT-Based SDA for Computing $A \bullet Y_0 + B \bullet Y_1 + C \bullet Y_2 + D \bullet Y_3$



Distributed Arithmetic

- LUT Technique for Distributed Arithmetic

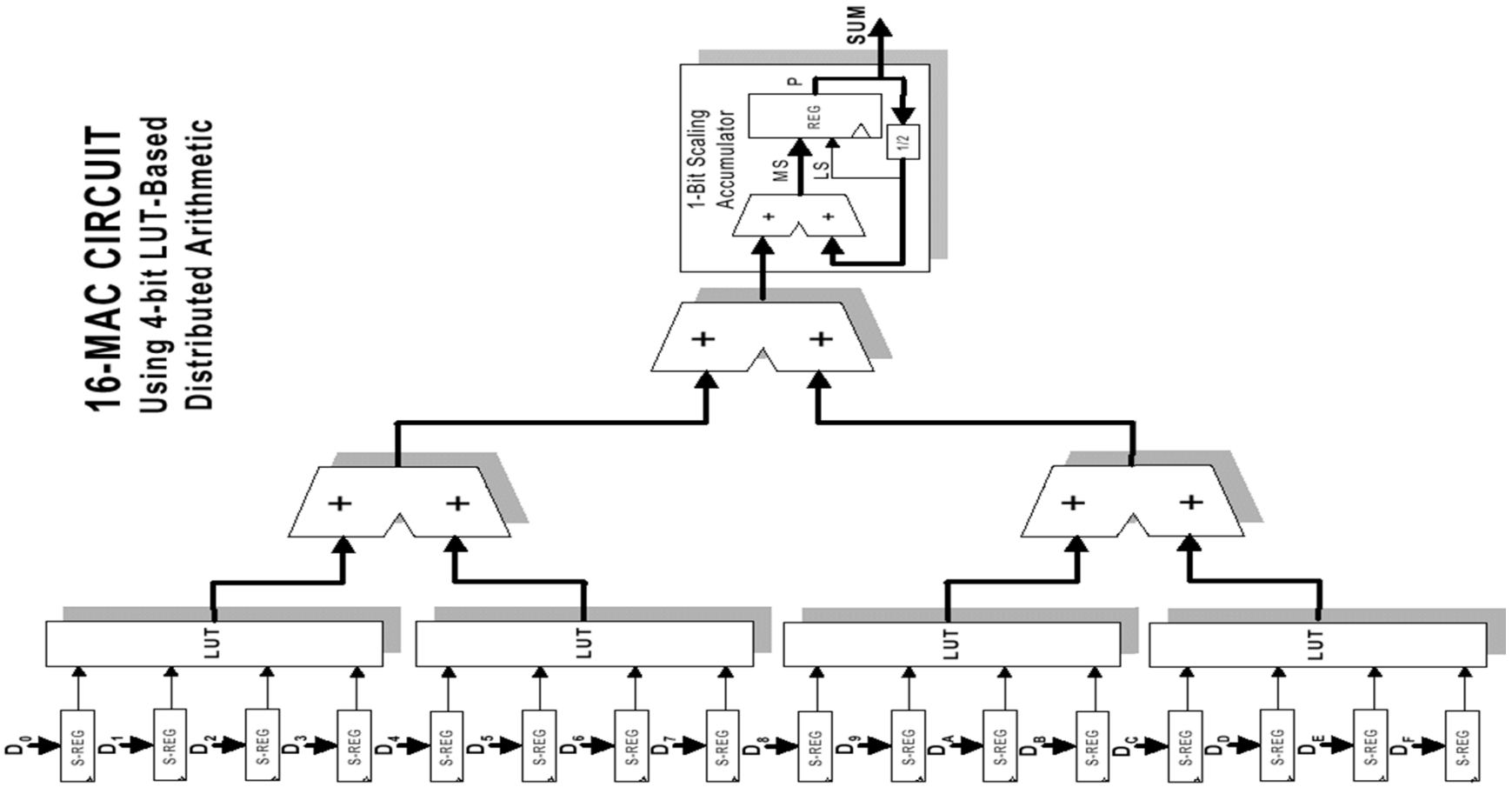


Look-Up Table

LUT Inputs				LUT Output
$<D_i$	C_i	B_i	$A_i>$	Partial Sum
<0	0	0	0>	0000...0
<0	0	0	1>	Y_A
<0	0	1	0>	Y_B
<0	0	1	1>	Y_B+Y_A
<0	1	0	0>	Y_C
<0	1	0	1>	Y_C+Y_A
<0	1	1	0>	Y_C+Y_B
<0	1	1	1>	$Y_C+Y_B+Y_A$
<1	0	0	0>	Y_D
<1	0	0	1>	Y_D+Y_A
<1	0	1	0>	Y_D+Y_B
<1	0	1	1>	$Y_D+Y_B+Y_A$
<1	1	0	0>	Y_D+Y_C
<1	1	0	1>	$Y_D+Y_C+Y_A$
<1	1	1	0>	$Y_D+Y_C+Y_B$
<1	1	1	1>	$Y_D+Y_C+Y_B+Y_A$

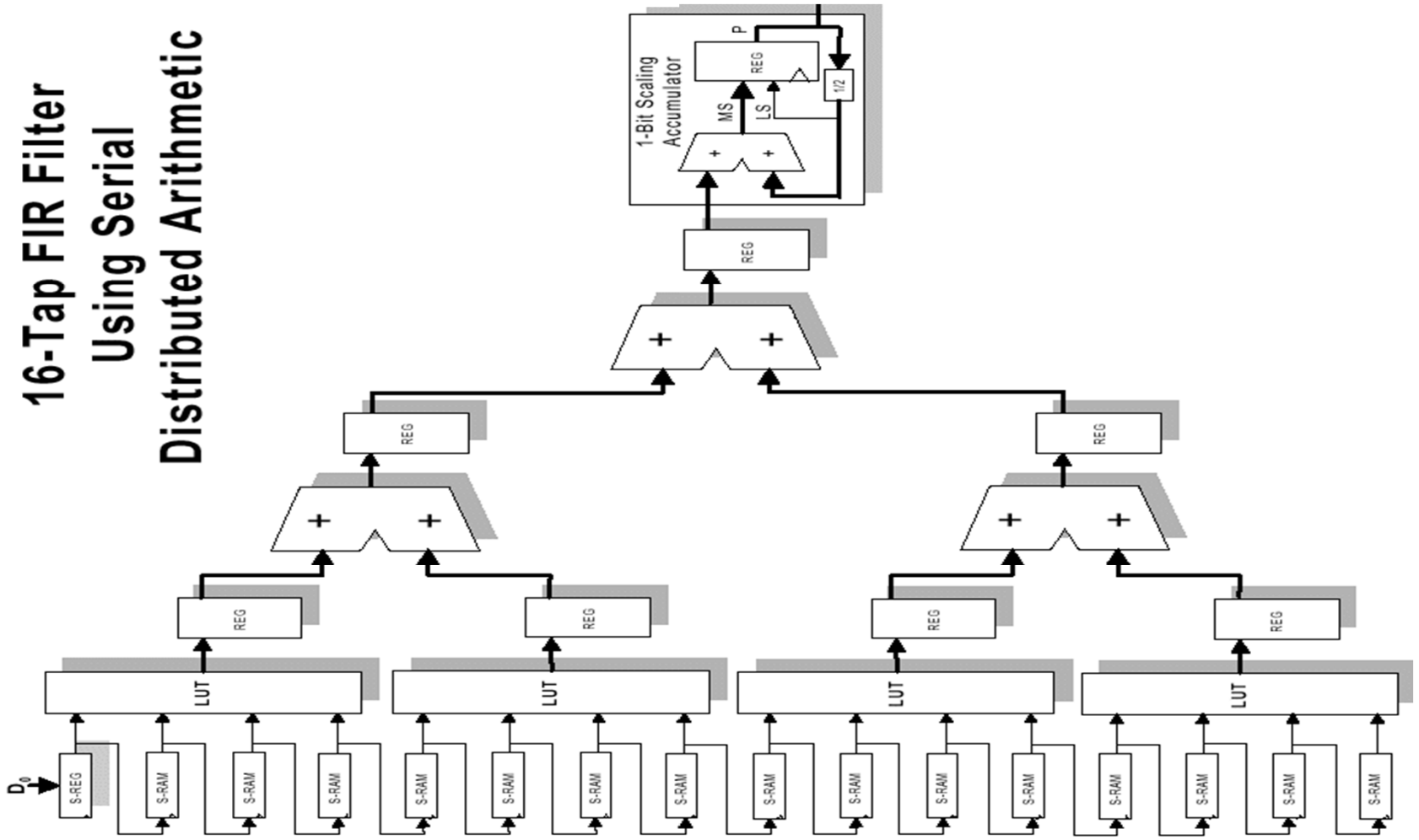
Distributed Arithmetic

- SDA 16-MAC Circuit



Distributed Arithmetic

- SDA 16-Tap FIR Filter



Parallel Distributed Arithmetic