DIGIT–SERIAL PROCESSING ELEMENTS

Digit-serial arithmetic processes one digit of size d in each time step.

if $d = W_d$ => conventional bit-parallel arithmetic

if $d = 1$ => bit-serial arithmetic

d in the range 1 – 4 is probably a good choice

Potential advantages:

+ Smaller chip area

±? Power consumption

±? Design complexity

±? Available tools

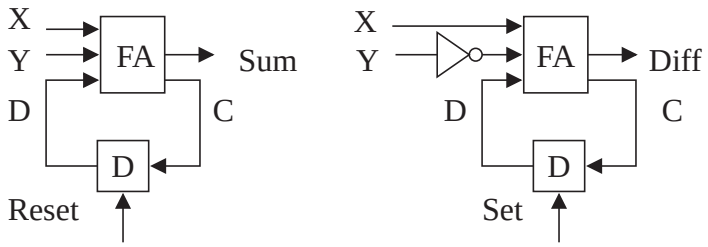
BIT-SERIAL ARITHMETIC

Bit-Serial Addition and Subtraction

$$X = x_0 \bullet x_1 x_2 \ldots x_{Wd-1}$$

$$Y = y_0 \bullet y_1 y_2 \ldots y_{Wd-1}$$

MSB LSB



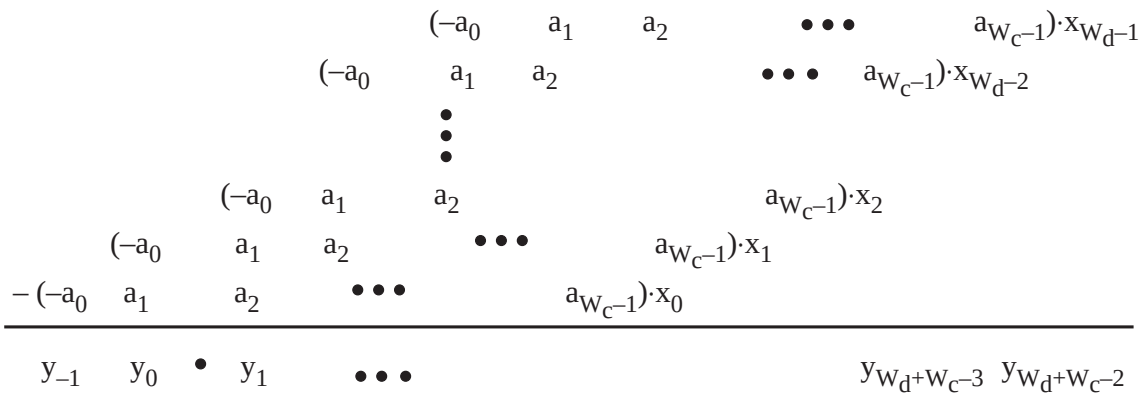
LSB first is commonly used

The MSB first case is more difficult, but it is sometimes used

Bit-Serial Multiplication

Serial/Parallel Multipliers

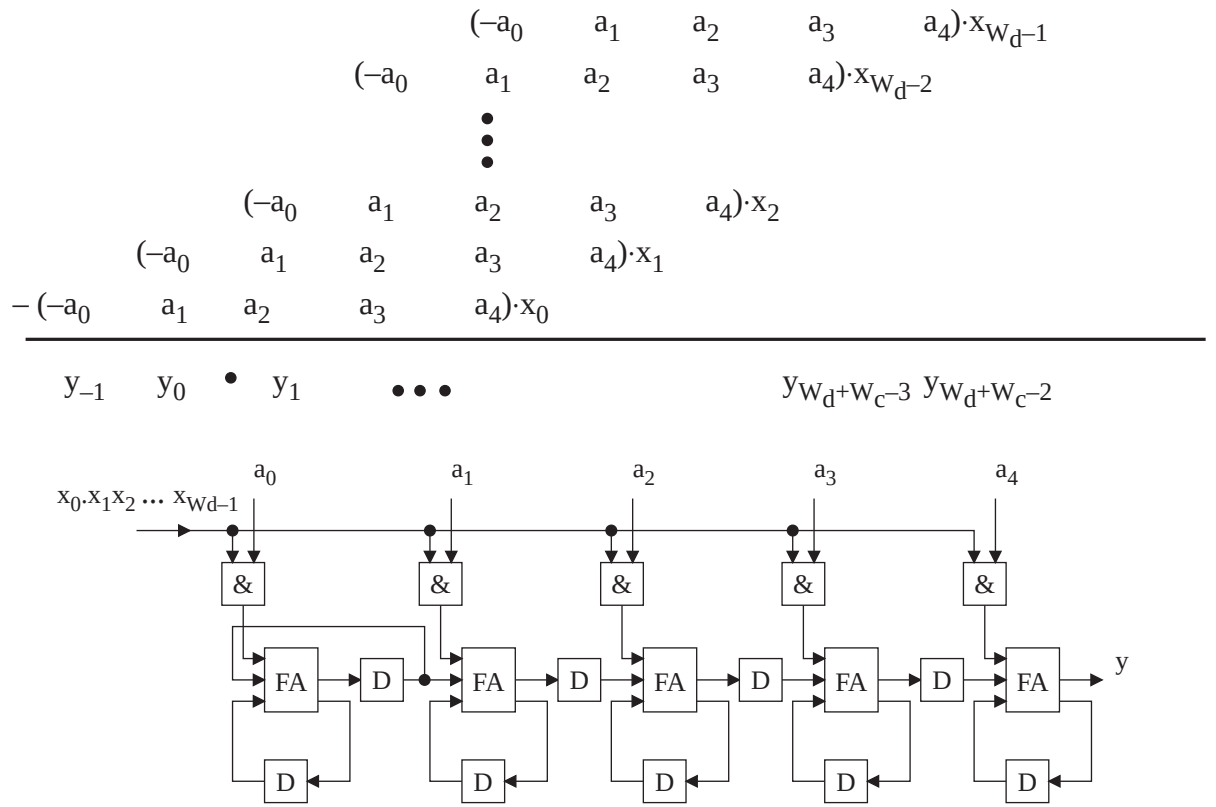
In a serial/parallel multiplier the multiplicand, ***x***, arrive bit-serially while the multiplier, ***a***, is applied in a bit-parallel format



Many different schemes for bit-serial multipliers have been proposed, but all are based on the add-and-shift principle.

They differ mainly in which order bit-products are generated and added and in the way subtraction is handled.

Special case when data is positive, $x \geq 0$



During the first W_d clock cycles, the least significant part of the product is computed and the most significant is stored in the D flip-flops.

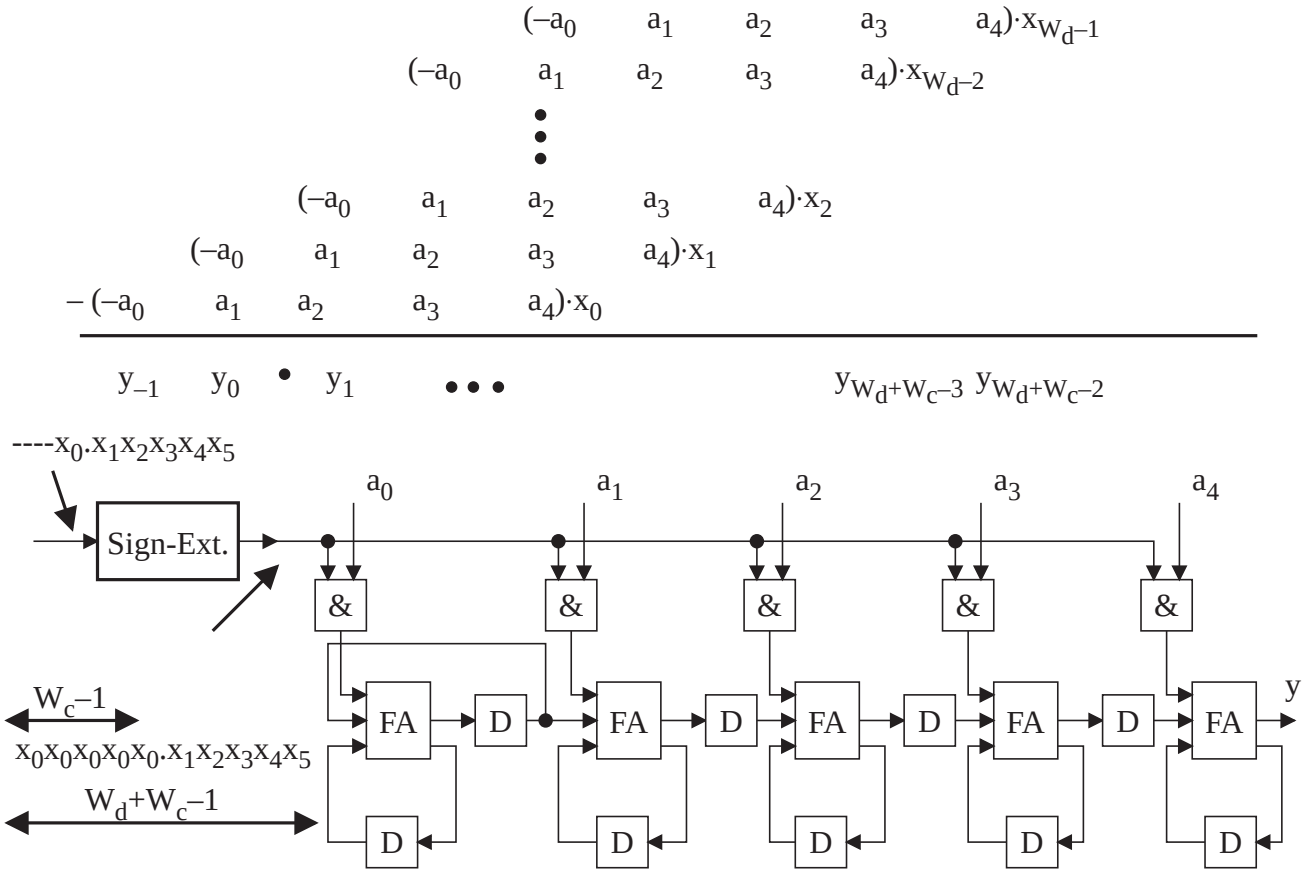
In the next W_c–1 clock cycles, zeros are therefore applied to the input so that the most significant part of the product is shifted out of the multiplier.

Hence, the multiplication requires W_d+W_c–1 clock cycles.

Two successive multiplications must therefore be separated by W_d+W_c–1 clock cycles.

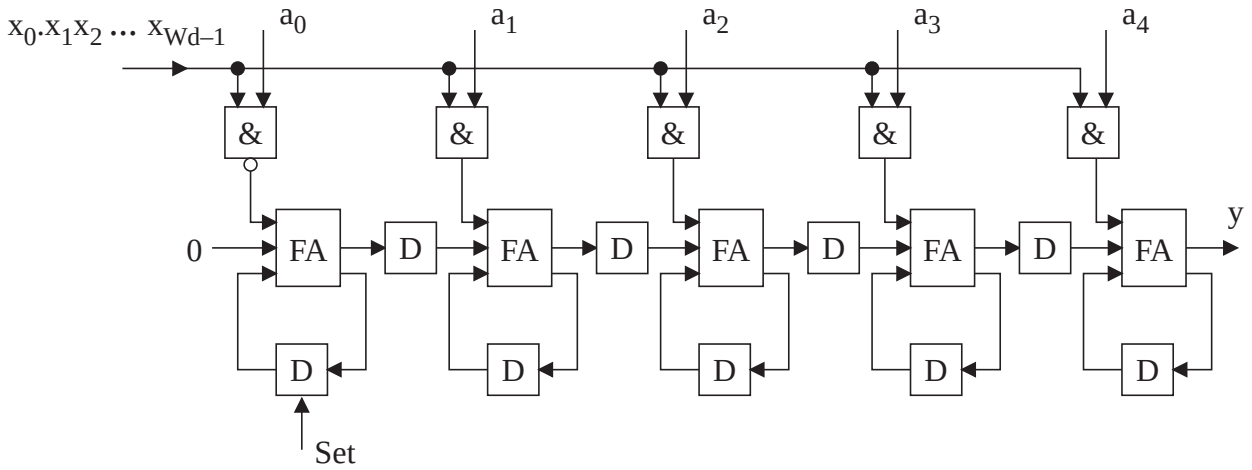
Signed Multiplicand – two's-complement

The subtraction of the bit-products required for the sign-bit can be avoided by extending the input by W_c–1 copies of the sign-bit.



A 16-bit serial/parallel multiplier implemented using two-phase logic in a 0.8-µm CMOS process requires an area of only 90 µm × 550 µm ≈ 0.050 mm^2.

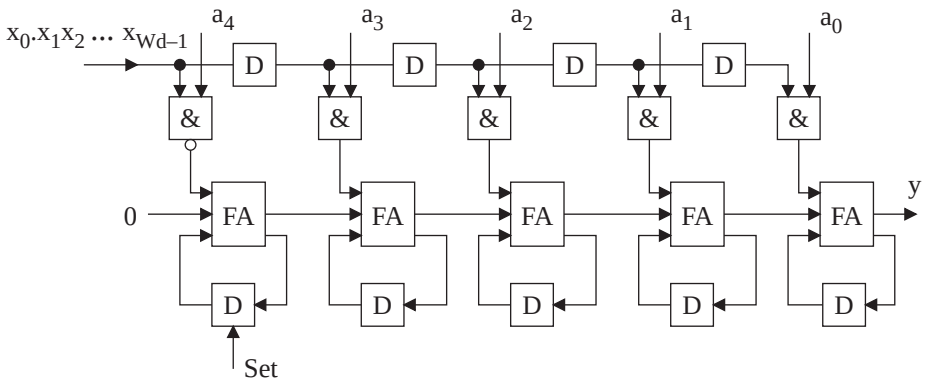
An alternative version that is the more favorable.



Modified Serial/parallel multiplier

Transposed Serial/Parallel Multiplier

An alternative realization of the serial/parallel multiplier which adds the bit-products diagonal-wise – Lyon's multiplier

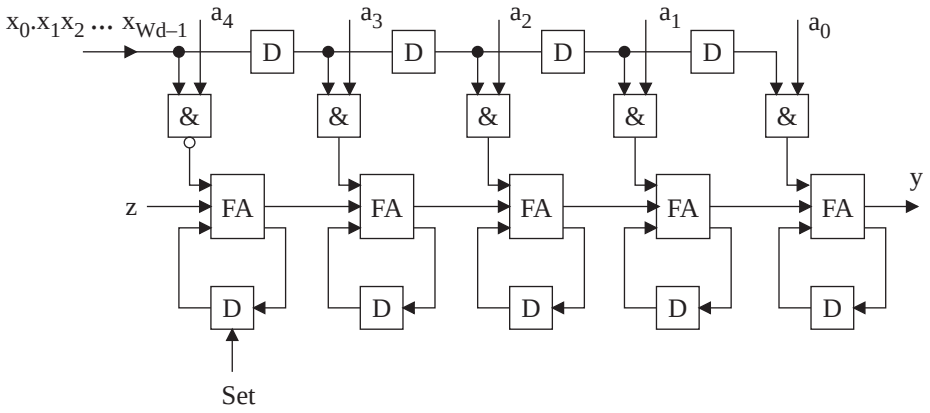


Transposed serial/parallel multiplier

No obvious advantages, but is commonly referred to in the literature!

Serial/Parallel Multiplier–Accumulator

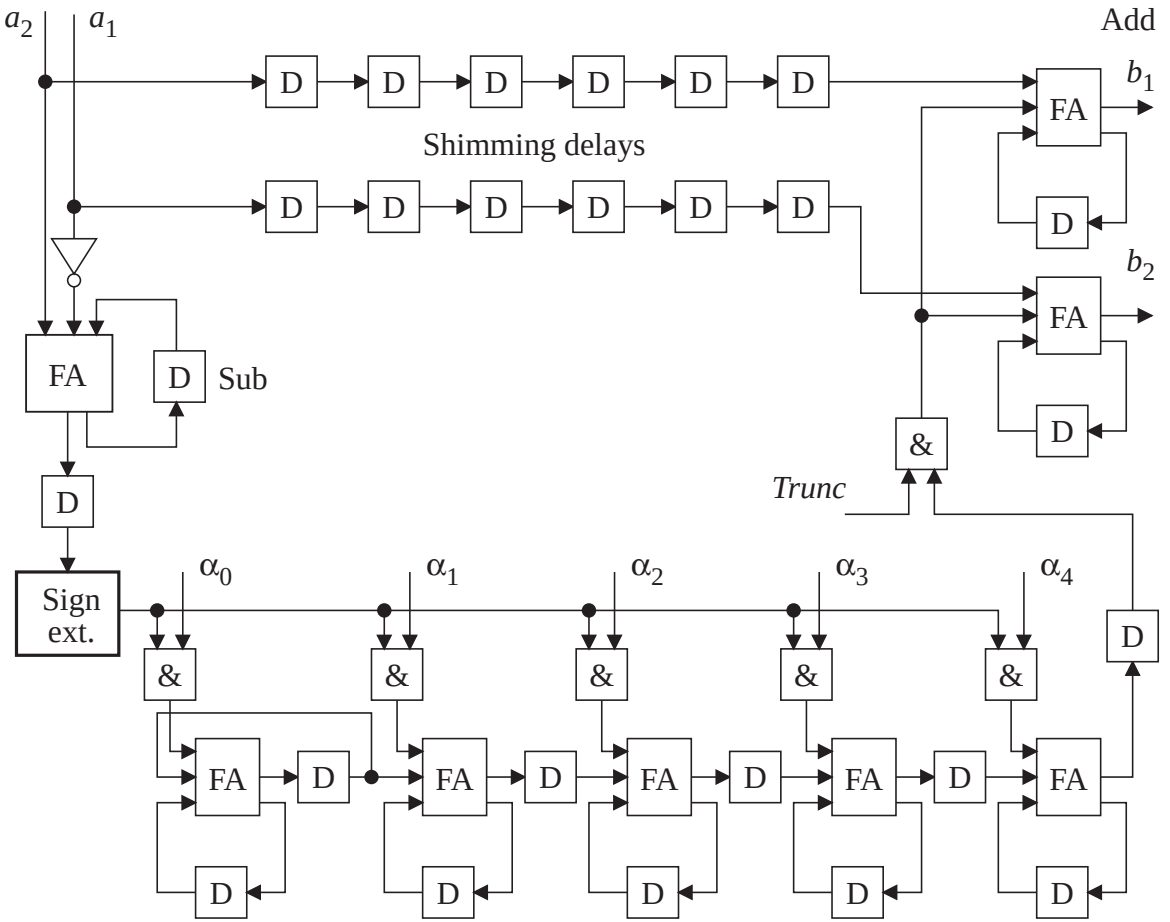
$$y = a \cdot x + z$$



Serial/parallel multiplier with an inherent input for addition

Restrictions on the accumulators wordlength – built-in truncation

Bit-Serial Adaptor

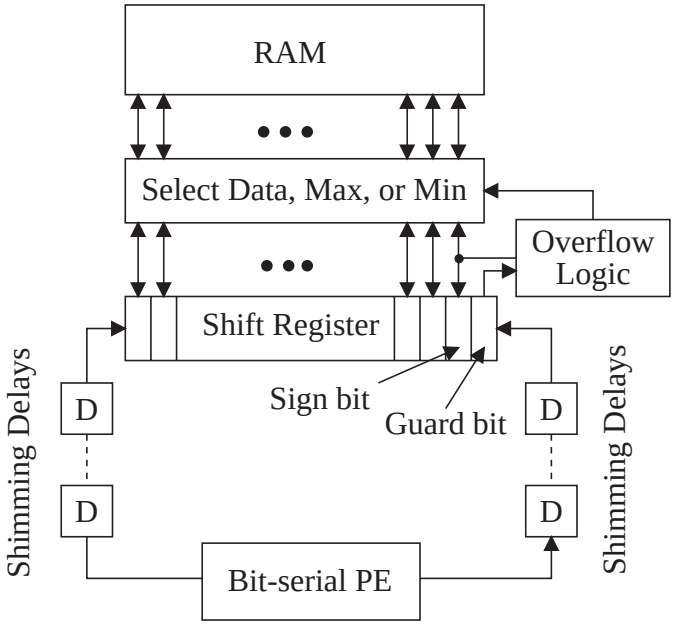


In order to avoid limit cycles in wave digital filters we should quantize the outputs of the adaptors:

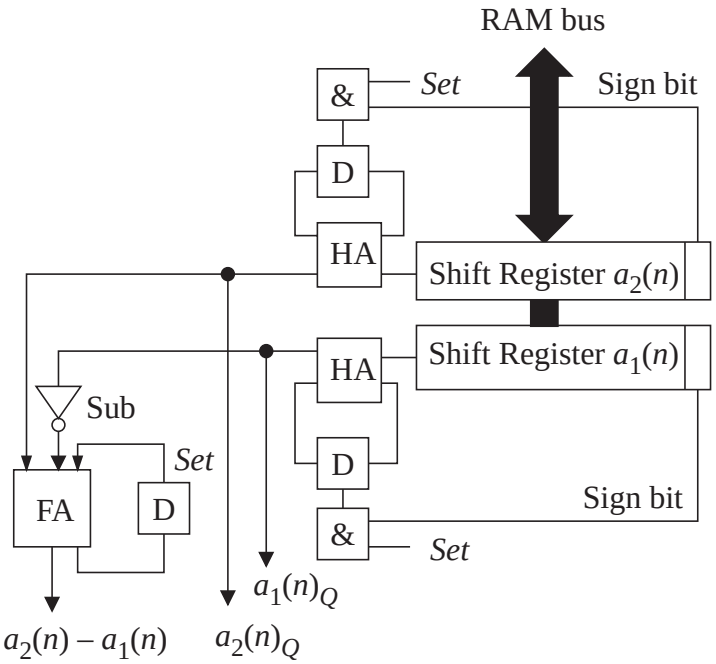
$$\left|b_i(n)_Q\right| < \left|b_i(n)_{exact}\right| \quad \text{(Magnitude truncation)}$$

Hence, we must know the sign-bit and the guard-bit!

If sign-bit ≠ guard-bit => overflow!



Bit-serial PE with overflow correction



Magnitude truncation in two-port adaptors

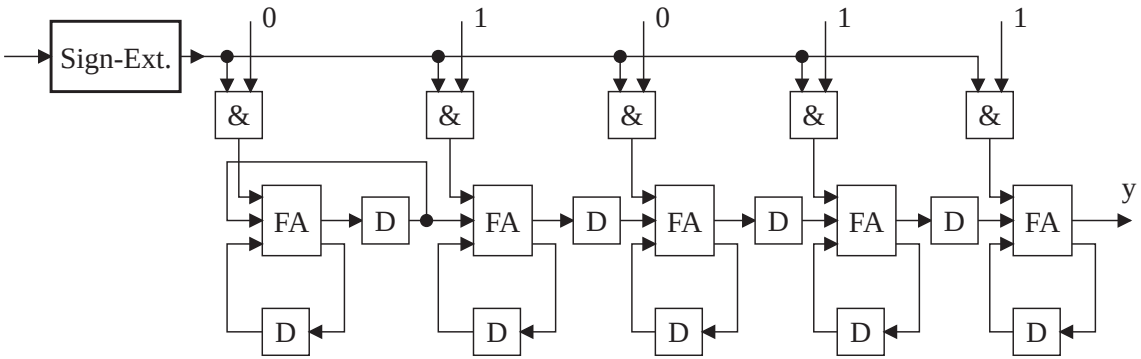
Examples of performing operations during data transport!

S/P MULTIPLIERS WITH FIXED COEFFICIENTS

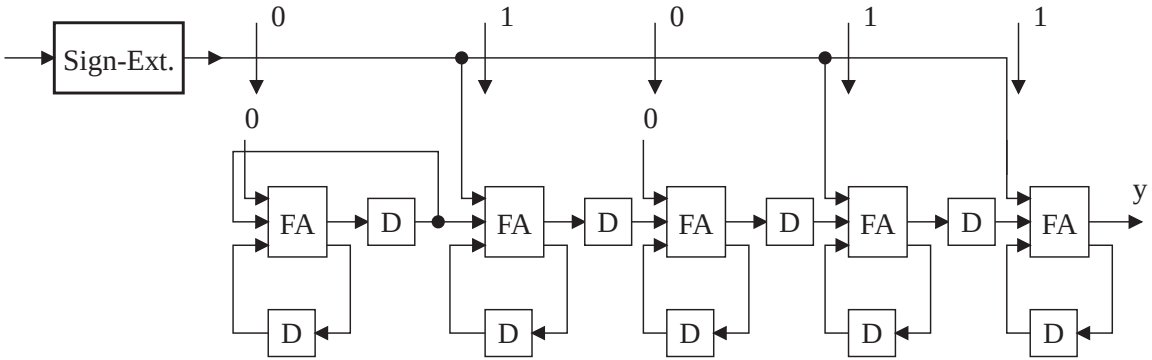
Serial/Parallel Multipliers with a Positive Coefficient

Example

Fixed positive coefficient $\boldsymbol{a} = (0.1011)_{2C}$.



First, we remove unnecessary AND gates.

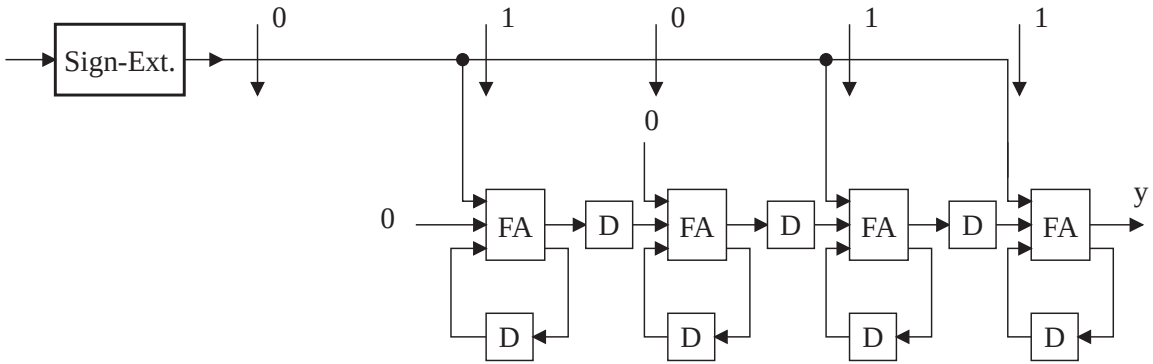


Serial/parallel multiplier with AND gates removed

Next, we notice that the D flip-flops are cleared at the beginning of the multiplication.

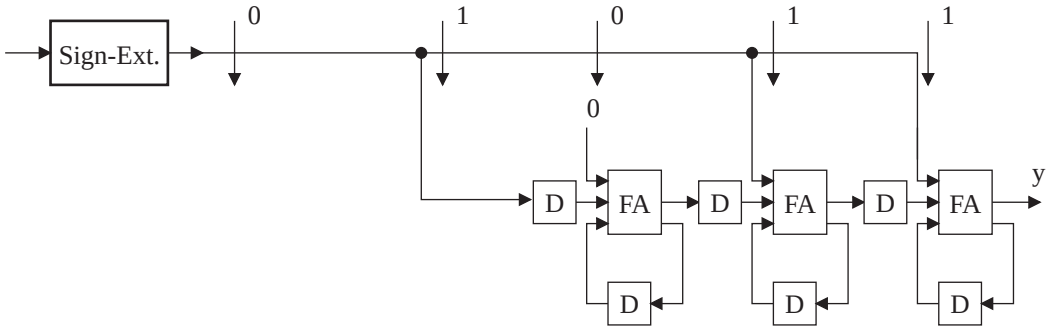
Hence, the left-most FA therefore has only zeros as inputs.

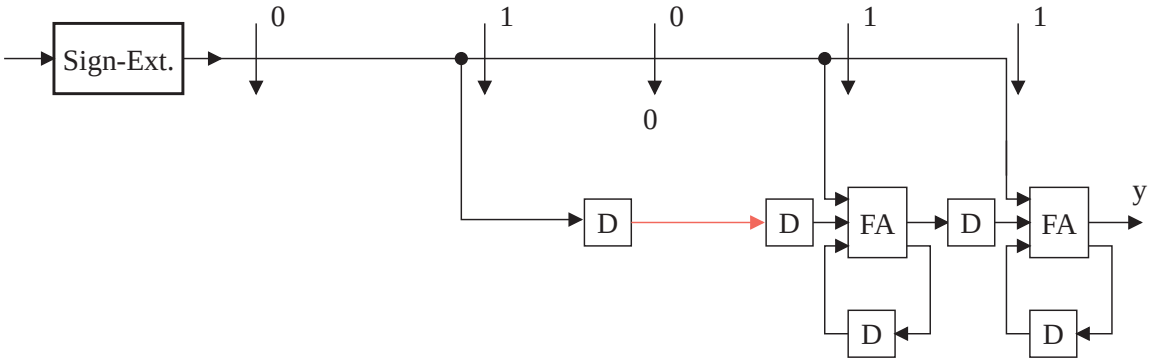
Hence, it will always produce sum and carry bits that are zero, and can therefore be removed.



Simplified serial/parallel multiplier

Next step





1. **Remove all AND gates.**

2. **Remove all FAs and corresponding D flip-flops, starting with the most significant bit in the coefficient, up to the first 1 in the coefficient.**

3. **Replace each FA that corresponds to a zero-bit in the coefficient with a feedthrough.**

The number of FAs is one less that the number of 1's in the coefficient.

The number of D flip-flops equals the number of 1-bit positions between the first and last bit positions.

Thus, substantial savings can be made for fixed coefficients.

Serial/Parallel Multipliers with a Negative Coefficient

Special case of CSDC!

Serial/Parallel Multipliers with a CSDC Coefficient

The serial/parallel multiplier can be simplified significantly if the coefficient is fixed.

The cost is essentially determined by the number of nonzero bits in the coefficient.

In CSDC, the average number of nonzero bits is only about $W_c/3$, compared to $W_c/2$ for two's-complement numbers. Further reductions in hardware resources are therefore possible.

A number in CSDC representation can be written

$$\boldsymbol{c} = \pm c_0 \pm c_1\, 2^{-1} \pm c_2\, 2^{-2} \pm c_3\, 2^{-3} \pm \ldots \pm c_{Wc-1}\, 2^{-Wc+1}$$

where most of the bits are zero.

The number $\boldsymbol{c}$ can be rewritten as a difference between two numbers with only positive coefficients,

$$\boldsymbol{c} = \boldsymbol{c}_+ - \boldsymbol{c}_-$$

A multiplication can now be written

$$\boldsymbol{y} = \boldsymbol{c}\,\boldsymbol{x} = (\boldsymbol{c}_+ - \boldsymbol{c}_-)\,\boldsymbol{x} = \boldsymbol{c}_+\,\boldsymbol{x} + \boldsymbol{c}_-(-\boldsymbol{x}) = \boldsymbol{c}_+\,\boldsymbol{x} + \boldsymbol{c}_-(\bar{\boldsymbol{x}} + 2^{-Wd+1})$$

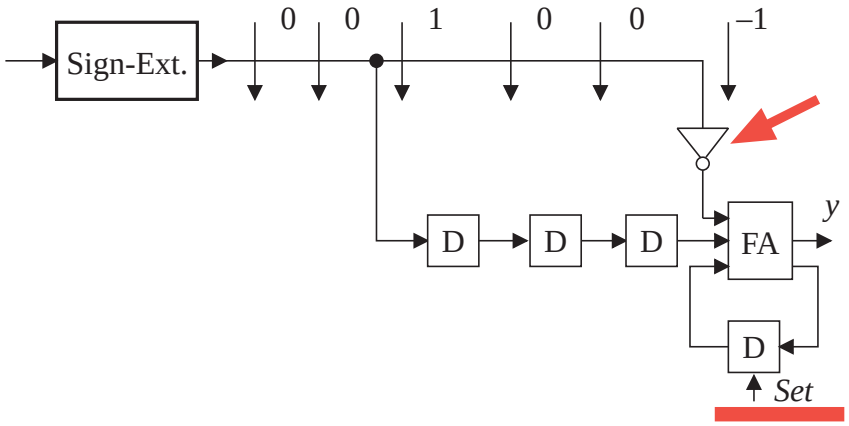
where $\bar{\boldsymbol{x}}$ represents the original value with all bits inverted.

Obviously, the multiplication can be implemented using the technique discussed above, except the $\boldsymbol{x}$-inputs to the FAs in bit positions with negative coefficient weights are inverted, and the corresponding carry D flip-flops are initially set.

Example

Fixed coefficient $\boldsymbol{a} = (0.00111)_{2C}$ using CSDC. We first rewrite the coefficient, as just discussed, using CSDC.

$$\boldsymbol{a} = (0.00111)_{2C} = (0.0100\bar{1})_{CSDC} = (0.01000)_{2C} - (0.00001)_{2C}$$



Serial/parallel multiplier with CSDC representation of the coefficient $\boldsymbol{a} = (0.0100\bar{1})_{CSDC}$

Only one FA and four D flip-flops are needed, while an implementation using the two's-complement coefficient would require two FAs and five D flip-flops. Hence, the CSDC implementation is better in this case.

MINIMUM NUMBER OF BASIC OPERATIONS

There are many applications of fixed-point multiplications with fixed coefficients.

In such cases the implementation cost can often be reduced if the multiplications are replaced by elementary operations.

The most interesting cases are when the multiplication is realized by only using the following operations:

- Addition only
- Addition and subtraction
- Addition and shift
- Addition and subtraction and shift

As before, we will not differentiate between addition and subtraction, since their implementation cost is roughly the same.

A shift operation corresponds to a multiplication with a power-of-two.

A shift operation can be implemented either in bit-parallel arithmetic by a barrel shifter if the number of shifts vary or simply by a skewed wiring if the number of shifts is fixed - almost negligible cost.

In bit-serial arithmetic a shift operation corresponds to a cascade of D flip-flops. Thus, both the chip area and power consumption are significant in bit-serial arithmetic.

Multiplication with a Fixed Coefficient

As discussed before, a multiplication with a fixed coefficient (multiplicand) can be simplified if the latter is expressed in canonic signed digit code, CSDC).

The number of add/sub operations equals the number of nonzero digits in the multiplicand minus one.

However, the number of adders/subtractors required by this approach is not always a minimum if the multiplicand is larger than 44.

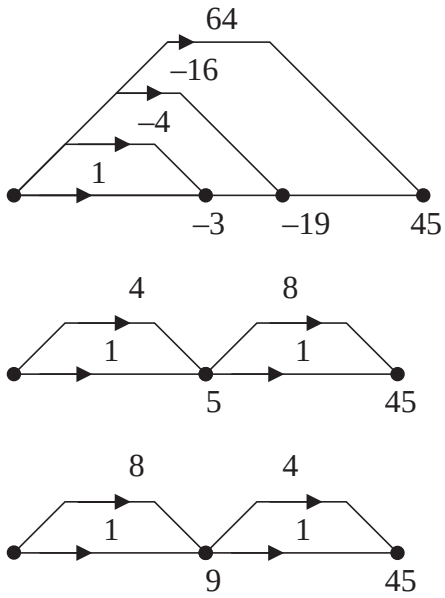
For example, $(45)_{10} = (10\bar{1}0\bar{1}01)_{CSDC}$ requires three additions (subtractions)

Another more efficient alternative representation is

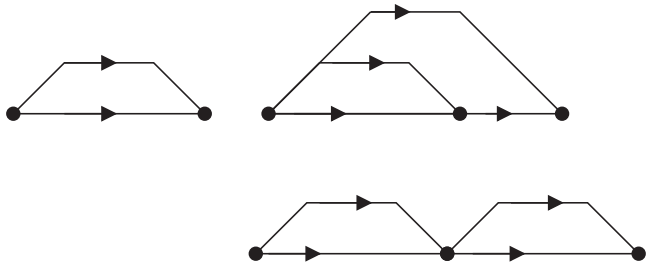
$$(45)_{10} = (2^3 + 1)(2^2 + 1)$$

which requires only two additions.

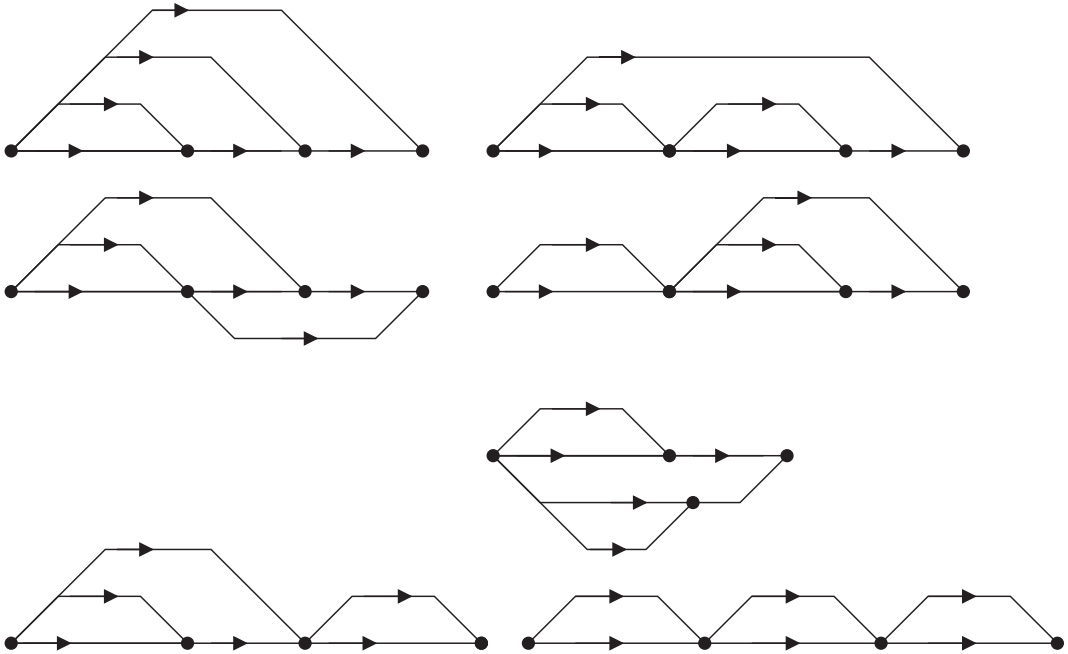
Hence, it is of great interest to find the best algorithm to perform the multiplication with fewest basic operations.



Alternative graphs representing multiplication with 45



Graphs with one and two adders/subtractors multipliers



Graphs for three adders/subtractors multipliers

There are seven different graphs with three adders/subtractors.

32 different graphs with four adders/subtractors.

All numbers in the range [–4096, 4095], which correspond to a 12-bit word length, and, of course, many outside this range can be realized with only four adder/subtractors.

The number of possible graphs grows rapidly with the number of adders/subtractors.

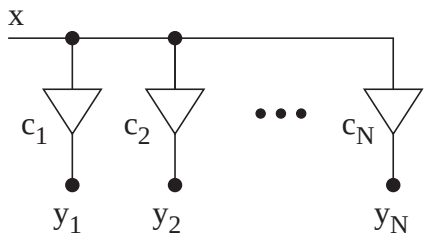
Obviously there must exist a representation that yields a minimum number of elementary operations (add/subtract–and–shift), but this number may not be unique.

Note that for some numbers the CSDC representation is still the best.

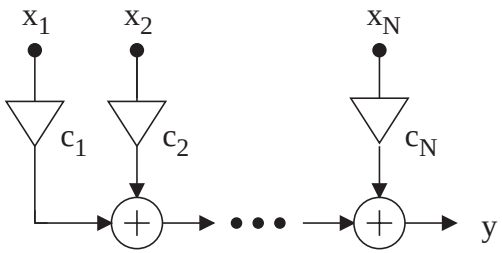
For a 12-bit word length, the optimal multipliers achieve an average reduction of 16% in the number of adders required over CSDC.

Note, however, that for a particular multiplicand the reduction may be much larger.

Multiple-Constant Multiplication

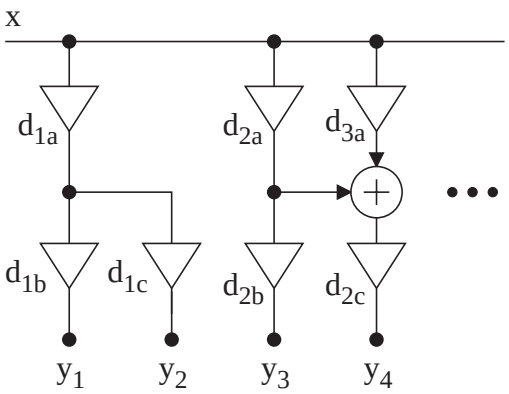


Multiple-constant multiplication



Transposed multiple-constant multiplication

The same problem – the same solution!



Exploiting common subexpressions to simplify multiple-constant multiplication

Active research topic – Look at our web site for recent papers!

DIGIT-SERIAL ARITHMETIC

From speed and power consumption points of view it may sometimes be advantageous to process several bits at a time, so-called digit-serial processing.

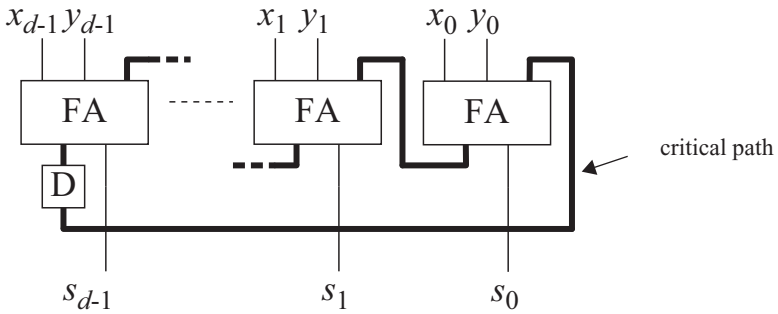
The number of bits processed in a clock cycle is referred to as the digit size, *d*.

Most of the principles, to be discussed above, for bit-serial arithmetic can easily be extended to digit-serial arithmetic.

The motivation for digit-serial processing is to find an optimum trade-off between power, chip area and processing capacity.

Active research topic!

Traditionally, digit-serial multipliers has been obtained either via unfolding of a bit-serial multiplier or via folding of a bit-parallel multiplier.



Digit-serial adder with digit-size d obtained from unfolding an bit-serial adder

The problem with these approaches is that the obtained circuits have not been pipelinable at the bit-level.

The recursive loop prohibits the insertion of pipelining to reduce the critical path to less than *d* full-adders, but solutions to this problem has been proposed by Oscar Gustafsson.

The latency of digit-serial processing elements are conveniently described in terms of number of clock cycles.

An adder has a latency of zero clock cycles while a serial/parallel multiplier has a latency of $\lceil W_f/d \rceil$ clock cycles, where W_f is the number of fractional bits of the coefficient.

The clock frequency will be determined by the longest critical path, which basically is the number of adjacent operations.

Introducing a pipelining stage will increase the latency with one clock cycle, but the critical (electrical) path will be decreased, and, thus, the clock frequency increased.

Look at our web site for techniques to find the optimal level of pipelining in maximally fast, digit-serial, implementations and much more.