MULTIPLIERLESS FILTER DESIGN

- Realization of filters without full-fledged multipliers
- Some slides based on support material by W. Wolf for his book *Modern VLSI Design, 3rd edition.* © W
- Partly based on following papers:
  - Hewlitt, R.M. and E.S. Swartzlander, "Canonical Signed Digit Representation for FIR Digital Filters", *IEEE Workshop on Signal Processing Systems, SiPS 2000*, Lafayette, LA, pp. 416-426, (2000).
  - Voronenko, Y. and M. Pueschel, *Multiplierless Multiple Constant Multiplication*, ACM Transactions on Algorithms, Vol. 3(2), (May 2007).
  - Aksoy, L., P. Flores and J. Monteiro, *A Tutorial on Multiplierless Design of FIR Filters: Algorithms and Architectures*, Circuits, Systems and Signal Processing, Vol.33(6), pp. 1689-1719, (2014).

TOPICS

- Multiplier wrap-up:
  - Array multiplier
  - Booth multiplier
- Filter structures: direct, transposed and hybrid forms
- Canonical signed digit
- Optimal single and multiple-constant multiplication
- Choosing coefficients

MULTIPLICATION

- Distinguish between:
  - Multiplication of two variables
  - Multiplication of one variable by a constant (*scaling*) $\Rightarrow$ opportunities of optimization
- Constants:
  - Can be considered as given
  - Can be specially chosen
- Implementation:
  - One-to-one
  - Resource sharing
  - In software, on processor without hardware multiplier

ELEMENTARY SCHOOL ALGORITHM

Unsigned numbers!

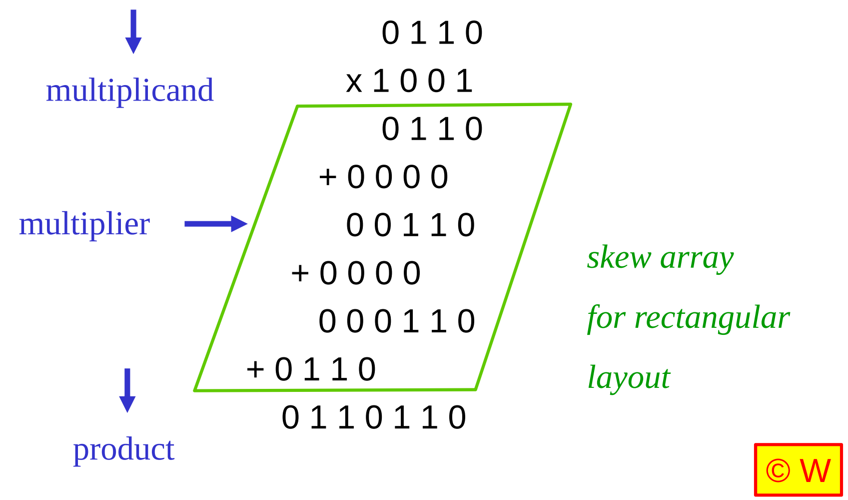
```
      0 1 1 0      multiplicand
    x 1 0 0 1      multiplier
    ---------
      0 1 1 0      <- partial product
  + 0 0 0 0
  ---------
    0 0 1 1 0
 + 0 0 0 0
 ----------
  0 0 0 1 1 0
+ 0 1 1 0
-----------
0 1 1 0 1 1 0
```

© W

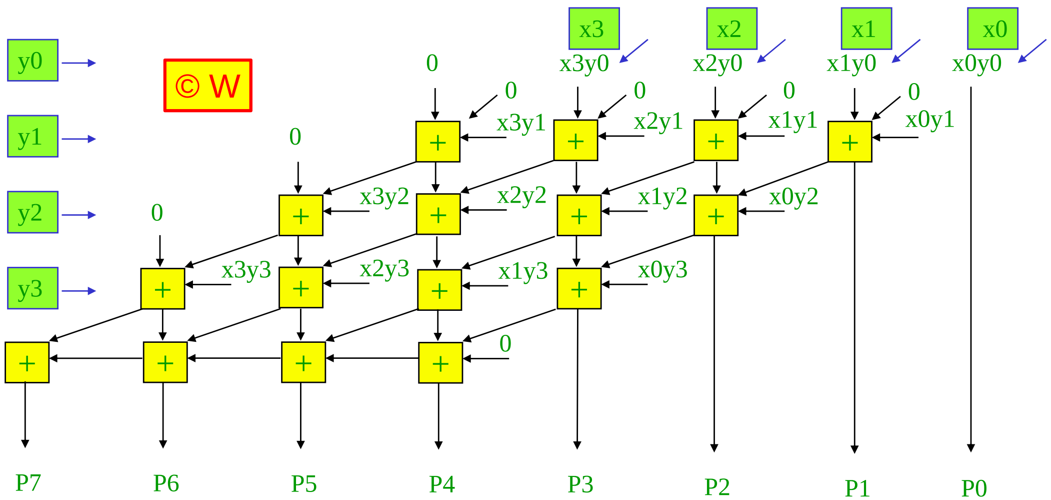
ARRAY MULTIPLIER

- Array multiplier is an efficient layout of a combinational (parallel-parallel) multiplier.
- Array multipliers may be pipelined to decrease clock period at the expense of latency.

ARRAY MULTIPLIER ORGANIZATION

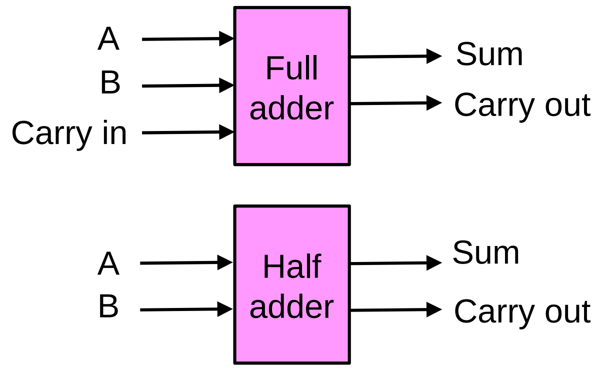


UNSIGNED 4X4 ARRAY MULTIPLIER



ARRAY MULTIPLIER COMPONENTS

- AND gates
- FULL ADDERs
- HALF ADDERs



- Fast multiplication amounts to reducing the critical path.

2'S COMPLEMENT MULTIPLICATION (1)

- An n-bit number X, and an m-bit number Y:

$$X = -x_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

$$Y = -y_{m-1} 2^{m-1} + \sum_{i=0}^{m-2} y_i 2^i$$

2'S COMPLEMENT MULTIPLICATION (2)

- Product:

$$P = XY = x_{n-1} y_{m-1} 2^{m+n-2} +$$
$$\sum_{i=0}^{n-2} \sum_{j=0}^{m-2} x_i y_j 2^{i+j} +$$
$$-2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i - 2^{m-1} \sum_{i=0}^{n-2} x_i y_{m-1} 2^i$$

2'S COMPLEMENT MULTIPLICATION (3)

- Note that: $-x \cdot 2^n = -2^n + \overline{x} \cdot 2^n$
- and: $\sum_{i=0}^{k} -2^i = 1 - 2^{k+1}$
- Therefore:

$$-2^{n-1} \sum_{i=0}^{m-2} y_i x_{n-1} 2^i = 2^{n-1} \sum_{i=0}^{m-2} -2^i + 2^{n-1} \sum_{i=0}^{m-2} \overline{y_i x_{n-1}} 2^i$$
$$= -2^{n+m-2} + 2^{n-1} + 2^{n-1} \sum_{i=0}^{m-2} \overline{y_i x_{n-1}} 2^i$$

2'S COMPLEMENT MULTIPLICATION (4)

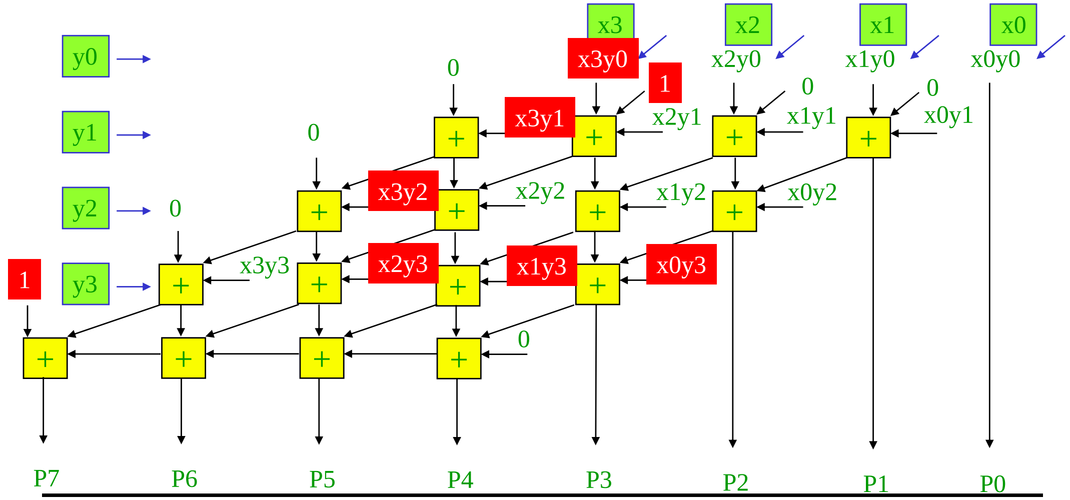
- The product becomes:

$$P = XY = x_{n-1} y_{m-1} 2^{n+m-2} +$$
$$\sum_{i=0}^{n-2} \sum_{j=0}^{m-2} x_i y_j 2^{i+j} - 2^{n+m-1} + 2^{n-2} + 2^{m-2}$$
$$+2^{n-1} \sum_{i=0}^{m-2} \overline{y_i x_{n-1}} 2^i + 2^{m-1} \sum_{i=0}^{n-2} \overline{x_i y_{m-1}} 2^i$$

BAUGH-WOOLEY MULTIPLIER

- Algorithm for two's-complement multiplication.
- Careful processing of partial products leads to:
  - Array with only additions, no subtractions
  - No hardware for sign extensions in upper left corner
- Achieved by:
  - Negation of some partial products
  - Injection of ones in some array positions

BAUGH-WOOLEY SIGNED 4X4 ARRAY MULTIPLIER



BOOTH MULTIPLIER

- Encoding scheme to reduce number of stages in multiplication.
- Performs two bits of multiplication at once; requires half the stages.
- Each stage is slightly more complex than an adder.

BOOTH ENCODING

- The wanted product: x*y.
- Two's-complement form of multiplier:

  $y = -2^n y_n + 2^{n-1} y_{n-1} + 2^{n-2} y_{n-2} + \ldots$
- Rewrite using $2^a = 2^{a+1} - 2^a$:

  $y = 2^n(y_{n-1} - y_n) + 2^{n-1}(y_{n-2} - y_{n-1}) + 2^{n-2}(y_{n-3} - y_{n-2}) + 2^{n-3}(y_{n-4} - y_{n-3}) + \ldots$

  $y = 2^{n-1}(2(y_{n-1} - y_n) + (y_{n-2} - y_{n-1})) + 2^{n-3}(2(y_{n-3} - y_{n-2}) + (y_{n-4} - y_{n-3})) + \ldots$
- Consider first two terms: by looking at three bits of y, we can determine whether to add x, 2x,-x, -2x,or 0 to partial product.

BOOTH ACTIONS

$y_i\ y_{i-1}\ y_{i-2}$	increment $(2(y_{i-1} - y_i) + y_{i-2} - y_{i-1})$
0 0 0	0
0 0 1	1x
0 1 0	1x
0 1 1	2x
1 0 0	-2x
1 0 1	-1x
1 1 0	-1x
1 1 1	0

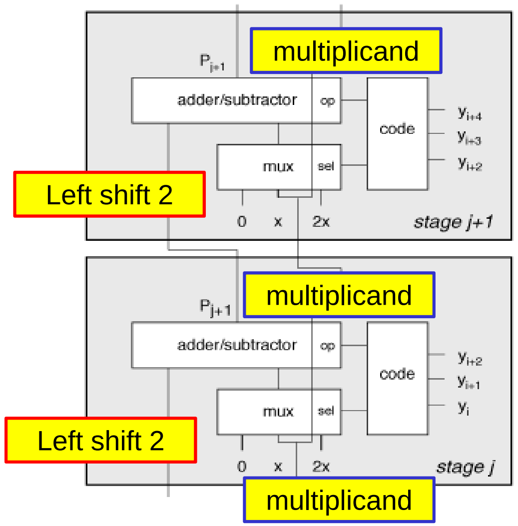
© W

BOOTH EXAMPLE

- x = 011001 (25_{10}), y = 101110 (-18_{10}).
- $y_1 y_0 y_{-1}$ = 100, $P_1 = P_0$ - (10 · 011001) = 11111001110. $-2 \cdot 1 \cdot x$
  -50_{10}
- $y_3 y_2 y_1$= 111, $P_2 = P_1$+ 0 = 11111001110. $0 \cdot 4 \cdot x$
  -50_{10}
- $y_5 y_4 y_3$= 101, $P_3 = P_2$ - 0110010000 = 11000111110 (-450_{10}).
  -50_{10} -400_{10} $-1 \cdot 16 \cdot x$

© W

BOOTH STRUCTURE



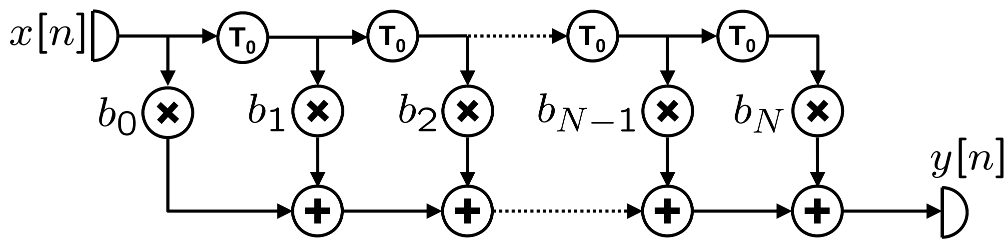
© W

FIR-FILTER DIRECT FORM (1)

- FIR = *finite impulse response*
- Difference equation:

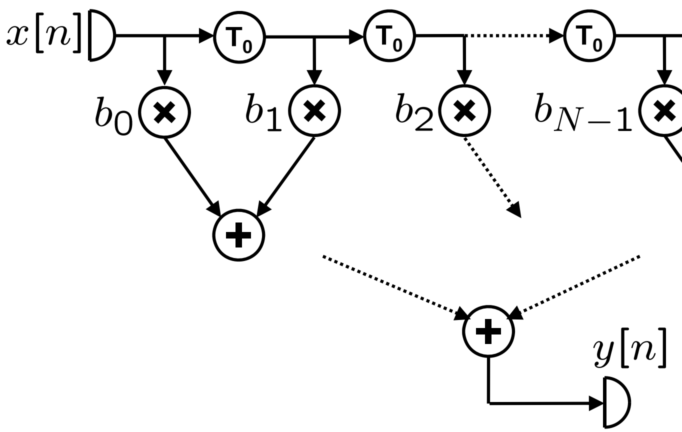
$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k]$$

- Where is the critical path?
- How long is it as function of *N*?



FIR-FILTER DIRECT FORM (2)

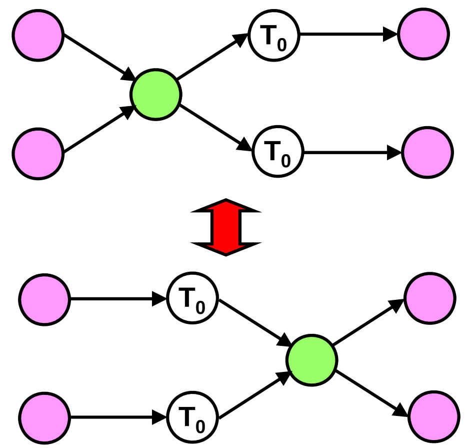
- Use a binary tree structure for the additions:



Critical path as function of *N*?

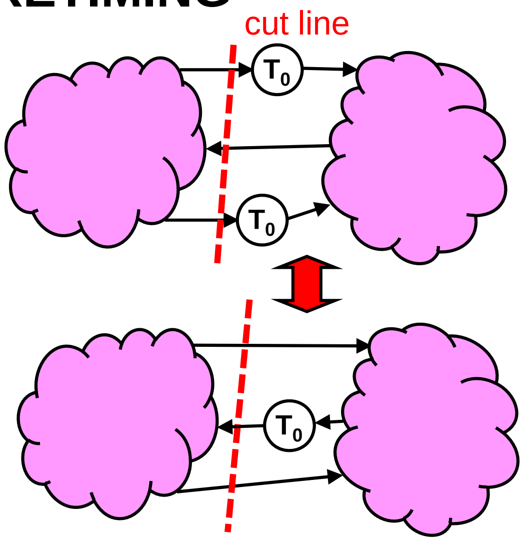
CLASSICAL RETIMING

- It is allowed to “push delay elements” through a computation:
  - From inputs to outputs or
  - From outputs to inputs
- Compute-and-then-delay is the same as delay-and-then-compute.
- Allowed in cyclic DFGs.

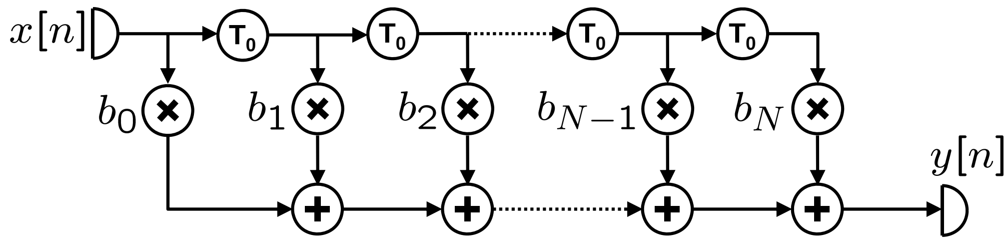


CUT-SET RETIMING

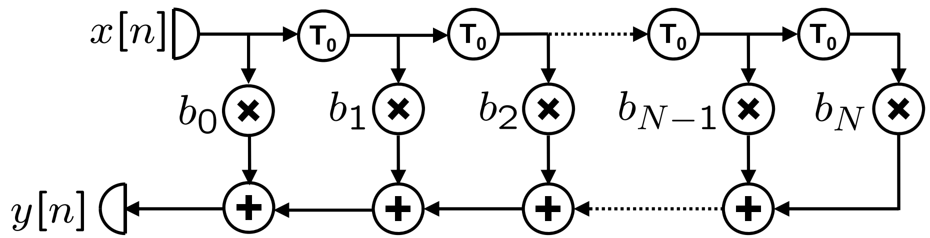
- Generalization of classical retiming.
- Cut-set = set of edges that cuts a graph in two when removed.
- Given a cut-set of any DFG, the DFG’s behavior remains unchanged if the same number of delays are added (removed) on incoming edges as are removed (added) on outgoing edges.



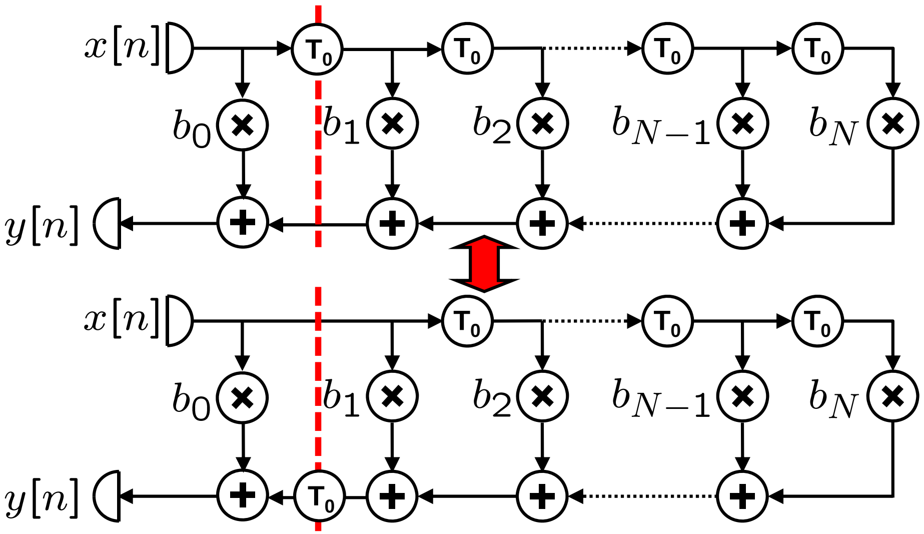
FIR-FILTER DIRECT FORM (3)



- Reverse order of additions:

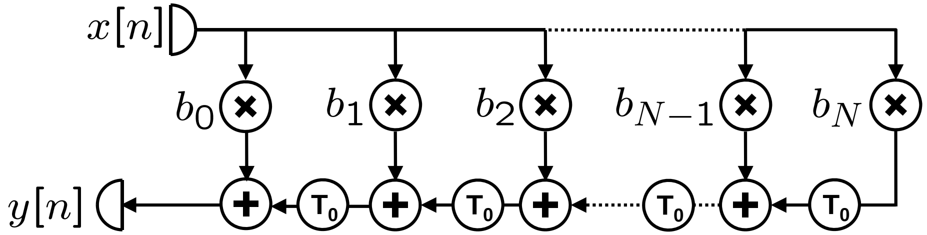


CUT-SET RETIMED FIR-FILTER



FIR-FILTER TRANSPOSED FORM

- Computationally equivalent to direct form
- Can be obtained by systematically applying cut-set retiming.
- Now, all multiplications share one input



FIR FILTER HYBRID FORM

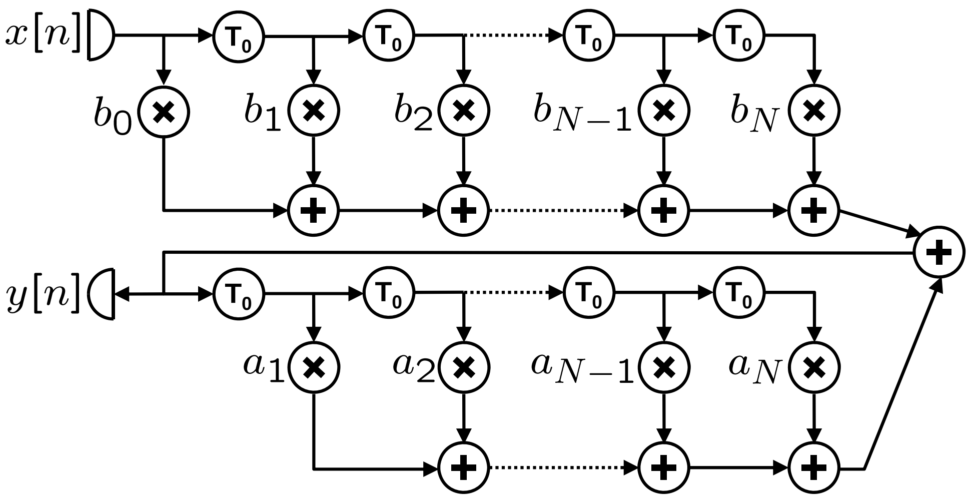
- The direct-form-implementation has all its delays in the input line.
- The transposed-form implementation has all delays on the output line.
- Hybrid-form implementation has part of the delays in the input line and part on the output line. See paper by Aksoy et al. for more details.

IIR FILTER

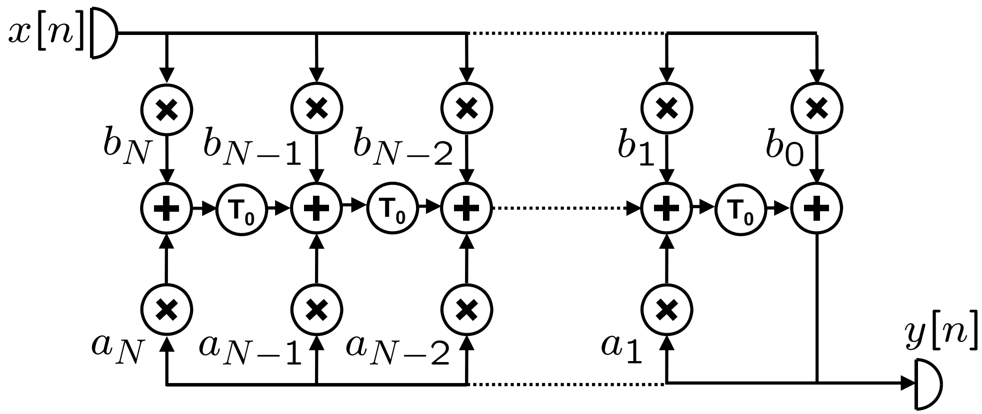
- IIR = *infinite impulse response*
- Difference equation:

$$y[n] = \sum_{k=1}^{N} a_k \cdot y[n-k] + \sum_{k=0}^{N} b_k \cdot x[n-k]$$

IIR-FILTER DIRECT FORM 1



IIR-FILTER TRANSPOSED FORM



SCALING: BOUNDS ON ADDITIONS (1)

- Consider multiplication of *x* by 71 = 1000111_2.
- Additions-only solution:

  71*x* = (*x* << 6) + (*x* << 2) + (*x* << 1) + *x*

  (realized by means of 3 shifts and 3 additions; shifts by a constant costs only wires in hardware)
- Subtractions-only solution:

  71*x* = ((*x* << 7) – *x*) – (*x* << 5) – (*x* << 4) – (*x* << 3)

  (realized by means of 4 shifts and 4 subtractions)

SCALING: BOUNDS ON ADDITIONS (2)

- In general, if *b* is the number of bits, *z* the number of zeros and *o* the number of ones (*b* = *z* + *o*):
  - The additions-only solution requires *o* – 1 additions.
  - The subtractions-only solution requires *z* + 1 subtractions.
- There is always a solution with at most *b*/2 + *O*(1) additions or subtractions (just take the cheapest of the two solutions).
- The *average* cost is also *b*/2 + *O*(1).
- Booth encoding has also the same cost.
- Can it be done better?

SIGNED POWER-OF-TWO REPRESENTATION

- Uses three-valued digits instead of binary digits: $0, 1, \overline{1}$
- A 1 at position k means a contribution of 2^k to the final value (as usual).
- A $\overline{1}$ at position k means a contribution of -2^k to the final value.
- Example: $1 0 1 \overline{1} 0 0 \overline{1} = 64 + 16 - 8 - 1 = 71$

CANONICAL SIGNED-DIGIT (CSD)

- Special case of signed-digit power-of-two, with minimal number of non-zero digits.
- Canonical = unique encoding.
- When used to minimize additions in constant multiplication, reduces number of operations to *b*/3 + *O*(1) in average, but still *b*/2 + *O*(1) in worst case.
- Example: $1 0 0 1 0 0 \overline{1} = 64 + 8 - 1 = 71$

TWO'S COMPLEMENT TO CSD CONVERSION (1)

- Two's complement number: $X = x_{n-1} x_{n-2} \cdots x_1 x_0$
- Target: $C = c_{n-1} c_{n-2} \cdots c_1 c_0$
- Start from LSB and proceed to MSB using table on next slide
- Dummy value (sign extension): $x_n = x_{n-1}$
- Carry-in, initialized to 0.

2'S COMPLEMENT TO CSD CONVERSION (2)

carry-in	x_{i+1}	x_i	carry-out	c_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	-1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	-1
1	1	1	1	0

Hewlitt & Swarzlander, Table 2

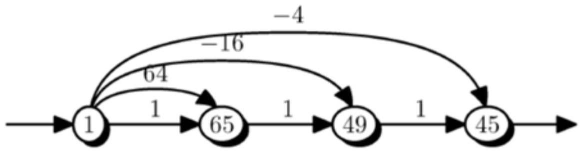
CSD NOT OPTIMAL

- CSD has minimal number of non-zeros, but is still not optimal for the “single constant multiplication” problem.
- How come?

SINGLE-CONSTANT MULTIPLICATION

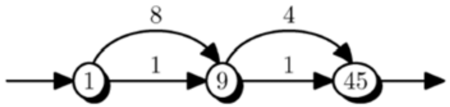
- Number of operations can be reduced by allowing shifting and adding intermediate results
- Example, goal is to multiply by $45 = 101101_2 = 10\bar{1}0\bar{1}01$

Voronenko & Pueschel, Figure 2



3x add/sub

$$65x = x + 64x$$
$$49x = 65x - 16x$$
$$45x = 49x - 4x$$



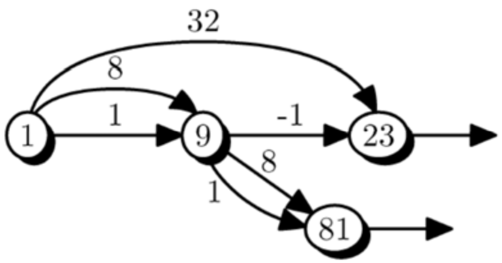
2x add/sub

$$9x = 8x + x$$
$$45x = 5(9x) = 9x + 4(9x)$$

MULTIPLE-CONSTANT MULTIPLICATION

- Even more opportunities for optimization occur when multiple constants can be optimized at the same time (think of the transposed form of a FIR filter).
- Example:

Voronenko & Pueschel, Figure 5



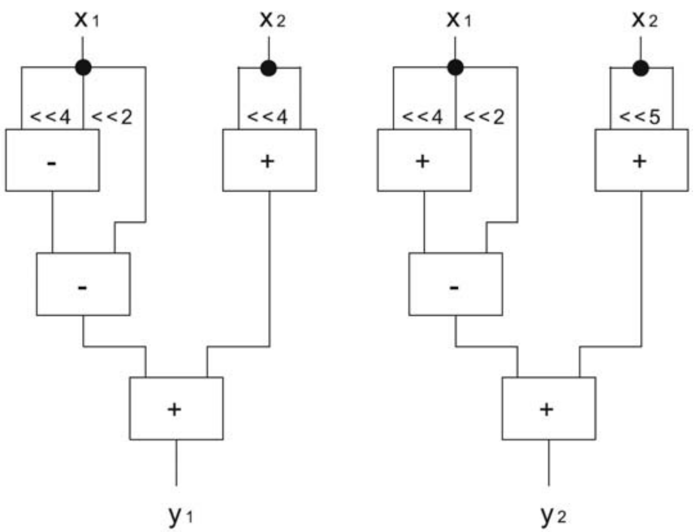
$$9x = 8x + x$$
$$23x = 32x - 9x$$
$$81x = 8(9x) + 9x$$

COMPUTATIONAL COMPLEXITY

- The optimization of the implementation for both the single-constant and multiple-constant multiplication problems is NP-complete.
- Powerful heuristics are available.
- Try SPIRAL on-line application:

http://spiral.ece.cmu.edu/mcm/gen.html

CONSTANT MATRIX-VECTOR MULT. (1)



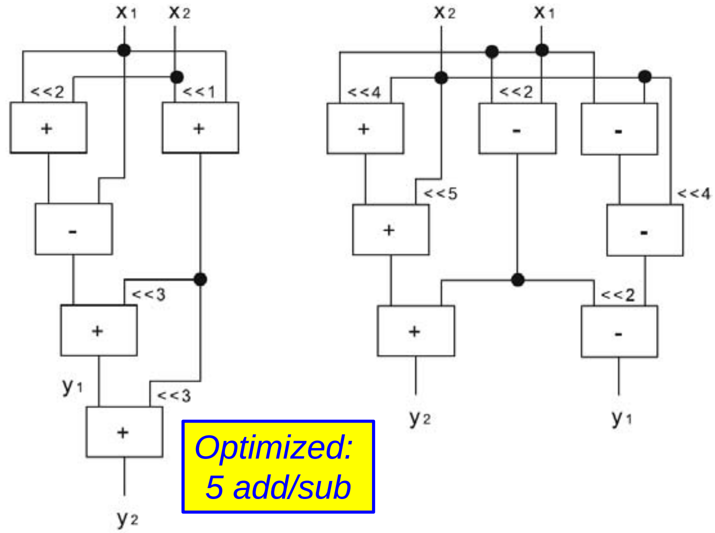
Applications in hybrid implementations of FIR filters

$y_1 = 11*x_1 + 17*x_2$
$y_2 = 19*x_1 + 33*x_2$

Unoptimized: 8 add/sub

Aksoy et al., Figure 3

CONSTANT MATRIX-VECTOR MULT. (2)



Optimized with depth constraint of 3: 7 add/sub

Optimized: 5 add/sub

Aksoy et al., Figure 5

CHOOSING THE COEFFICIENTS

- Until now, the discussion was about implementing filters with given constant coefficients as efficiently as possible.
- Classical approach starts from floating-point coefficients as e.g. computed in Matlab and a “blind” fixed-point conversion.
- It is even more interesting to take cheap implementation as a criterion during filter design. A problem description could e.g. be:
  - Given a number *T,* construct a filter with at most *T* non-zero bits in its set of coefficients while at the same time satisfying the usual criteria such as “bandwidth”, “pass band ripple”, etc.
- See e.g. tools at: https://at1x.nl/asic-tools/