DISTRIBUTED ARITHMETIC

Distributed arithmetic is an efficient procedure for computing inner products between a fixed and a variable data vector. The basic principle is owed to Croisier *et al. (Patent)*, and Peled and Liu have independently presented a similar method.

Consider the sum-of-products (*inner products*)

$$y = \mathbf{a}^T \cdot \mathbf{x} = \sum_{i=1}^{N} a_i x_i$$

where the coefficients, a_i, $i = 1, 2, ..., N$ are fixed.

A two's-complement representation is used for the data components which are scaled so that $|x_i| \leq 1$.

The inner product can be rewritten

$$y = \sum_{i=1}^{N} a_i \left[-x_{i0} + \sum_{k=1}^{W_d - 1} x_{ik} 2^{-k} \right]$$

where x_{ik} is the kth bit in $\boldsymbol{x}_i$.

By interchanging the order of the two summations we get

$$y = -\sum_{i=1}^{N} a_i x_{i0} + \sum_{k=1}^{W_d - 1} \left[\sum_{i=1}^{N} a_i x_{ik} \right] 2^{-k}$$

which can be written

$$y = -F_0(x_{10}, x_{20}, \ldots, x_{N0}) + \sum_{k=1}^{W_d - 1} F_k(x_{1k}, x_{2k}, \ldots, x_{Nk}) 2^{-k}$$

where

$$F_k(x_{1k}, x_{2k}, \ldots, x_{Nk}) = \sum_{i=1}^{N} a_i x_{ik}$$

F is a function of N binary variables, the ith variable being the kth bit in the data x_i.

Since F_k can take on only a finite number of values, 2^N, it can be computed and stored in a look-up table.

This table can be implemented using a ROM (Read-Only Memory).

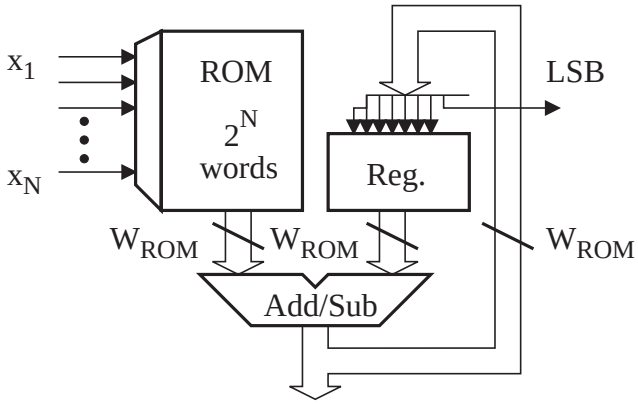
Using Horners method for evaluating a polynomial for $x = 0.5$, we can rewrite

$$y = -F_0(x_{10}, x_{20}, \ldots, x_{N0}) + \sum_{k=1}^{W_d - 1} F_k(x_{1k}, x_{2k}, \ldots, x_{Nk}) 2^{-k}$$

$$y = \Big(\Big(\ldots \Big(\Big(0 + F_{W_d - 1} \Big) 2^{-1} + \ldots + F_2 \Big) 2^{-1} + F_1 \Big) 2^{-1} - F_0 \Big)$$

$$y = \left(\left(\ldots\left(\left(0 + F_{W_d - 1}\right)2^{-1} + \ldots + F_2\right)2^{-1} + F_1\right)2^{-1} - F_0\right)$$

Inputs, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ are shifted bit-serially out from the shift registers with the least-significant bit first. Bits x_{ik} are used as an address to the ROM storing the look-up table.



The computational time is W_d clock cycles.

The word length in the ROM, W_{ROM}, depends on the F_k with the largest magnitude and the coefficient word length, W_c, and

$$W_{ROM} \leq W_c + \log_2(N)$$

Example 11.11

Determine the values that to be stored in ROM for the inner product

$$y = a_1 x_1 + a_2 x_2 + a_3 x_3$$

where $a_1 = \frac{33}{128} = (0.0100001)_{2C}$, $a_2 = \frac{85}{128} = (0.1010101)_{2C}$, and $a_3 = \frac{-11}{128} = (1.1110101)_{2C}$.

x_1 x_2 x_3	F_k	F_k	F_k
0 0 0	0	$(0.0000000)_{2C}$	0.0000000
0 0 1	a_3	$(1.1110101)_{2C}$	–0.0859375
0 1 0	a_2	$(0.1010101)_{2C}$	0.6640625
0 1 1	$a_2 + a_3$	$(0.1001010)_{2C}$	0.5781250
1 0 0	a_1	$(0.0100001)_{2C}$	0.2578125
1 0 1	$a_1 + a_3$	$(0.0010110)_{2C}$	0.1718750
1 1 0	$a_1 + a_2$	$(0.1110110)_{2C}$	0.9218750
1 1 1	$a_1 + a_2 + a_3$	$(0.1101011)_{2C}$	0.8359375

The shift-accumulator must be able to add correctly the largest possible value obtained in the accumulator register and in the ROM.

The largest value in the accumulator register is obtained when the largest (magnitude) value stored in the ROM is repeatedly accumulated.

$$y = \left(\left(\ldots\left(\left(0 + F_{W_d - 1}\right)2^{-1} + \ldots + F_2\right)2^{-1} + F_1\right)2^{-1} - F_0\right)$$

Thus, at the last clock cycle, corresponding to the sign bit, the value in REG is

$$|y| = \ldots(((0 + F_{max})2^{-1} + F_{max})2^{-1} + \ldots + F_{max})2^{-1} \leq F_{max}$$

Hence, the shift-accumulator must be able to add two numbers of magnitude $\leq F_{max}$.

The necessary number range is ± 1. The word length in the shift-accumulator must be extended with one guard bit for overflow detection = 1 + 8 bit word = 9 bits.

Notice the similarity between the equation for a scalar multiplication

$$\boldsymbol{y} = \boldsymbol{a} \cdot \boldsymbol{x} = \boldsymbol{a} \cdot \left\{ -x_0 + \sum_{k=1}^{W_d - 1} x_k 2^{-k} \right\}$$

and the inner product

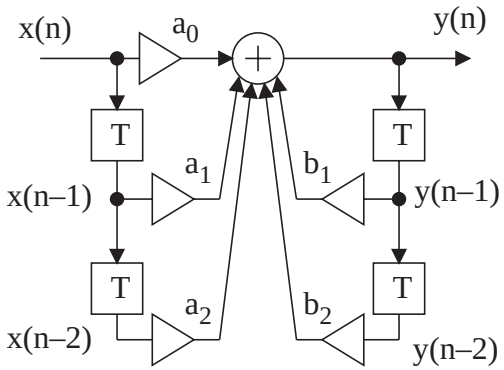
$$\boldsymbol{y} = -F_0(x_{10}, x_{20}, \ldots, x_{N0}) + \sum_{k=1}^{W_d - 1} F_k(x_{1k}, x_{2k}, \ldots, x_{Nk}) 2^{-k}$$

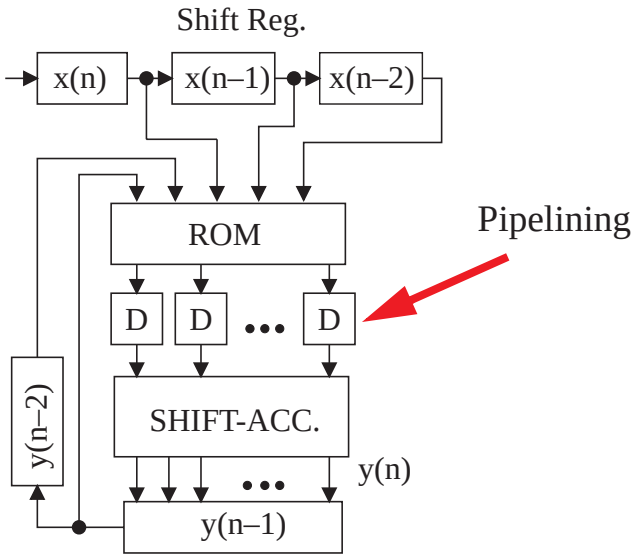
In both cases, the same type of shift-accumulator can be used.

Hence, the distributed arithmetic unit essentially consists of a serial/parallel multiplier augmented by a ROM.

Example 11.12

A second-order section in direct form I can be implemented by using only a single PE based on distributed arithmetic.



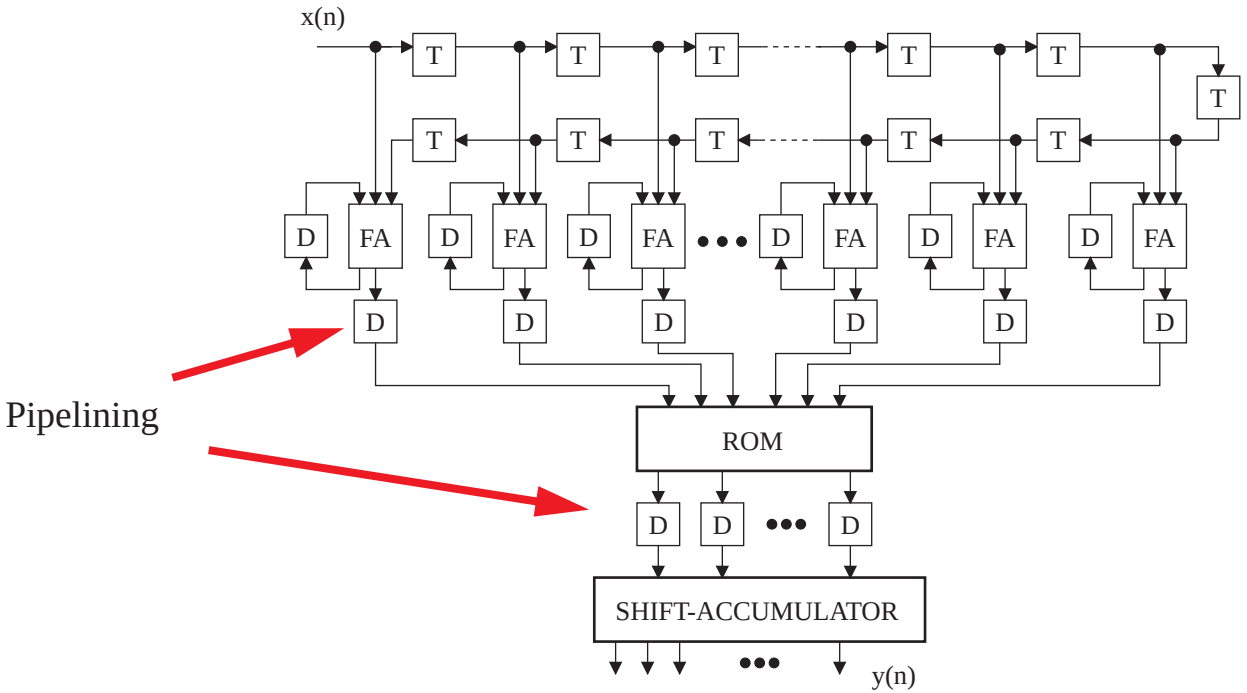


A set of D flip-flops has been placed between the ROM and the shift-accumulator to allow the two operations to overlap in time, i.e., the two operations are pipelined.

The number of words in the ROM is only $2^5 = 32$.

Example 11.13

An linear-phase FIR structure can also be implemented using distributed arithmetic. Assume that N is even.



The logic circuitry has been pipelined by introducing D flip-flops between the adders (subtractors) and the ROM, and between the ROM and the shift-accumulator.

$N/2$ bit-serial adders (subtractors) are used to sum the symmetrically placed values in the delay line. This reduces the number of terms in the inner product.

Only 64 words are required whereas $2^{12} = 4096$ words are required for the general case, e.g., a nonlinear-phase FIR filter.

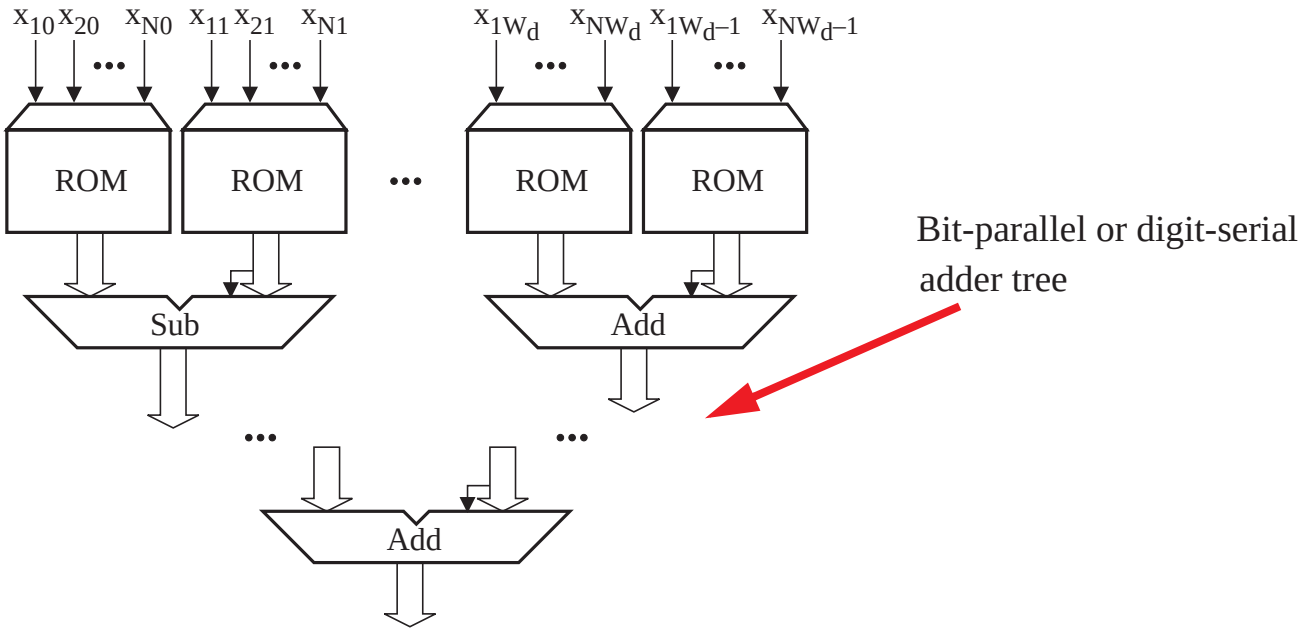
For higher-order FIR filters the reduction in the number of terms by 50% is essential.

The number of words in the ROM is 2^N where N is the number of terms in the inner product.

The chip area for the ROM is small for inner products with up to 5 to 6 terms. The basic approach is useful for up to 10 to11 terms.

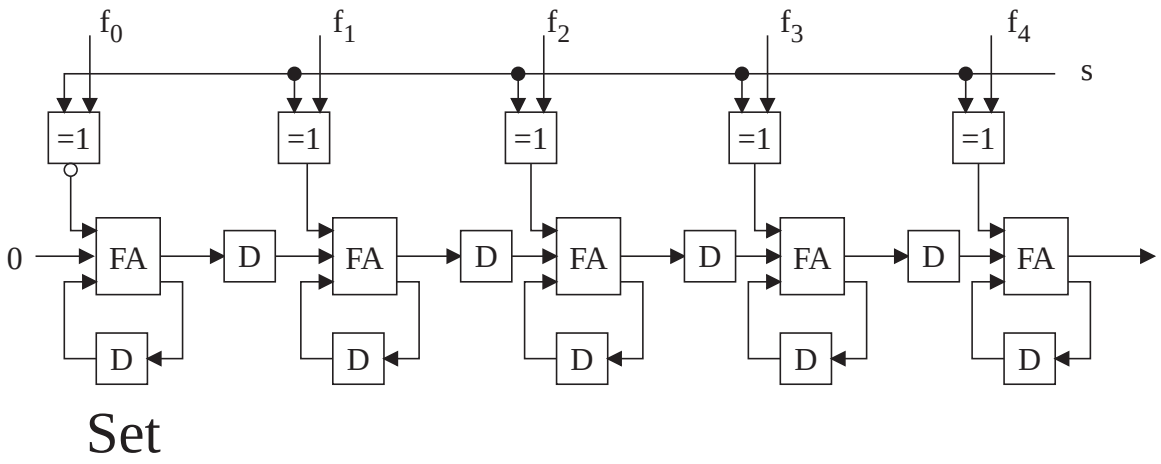
Parallel Implementation of Distributed Arithmetic

The inner products containing many terms, can be partitioned into a number of smaller inner products which can be computed and summed by using either distributed arithmetic or an adder tree.



Distributed arithmetic can also use two, or more bits a time

THE BASIC SHIFT-ACCUMULATOR

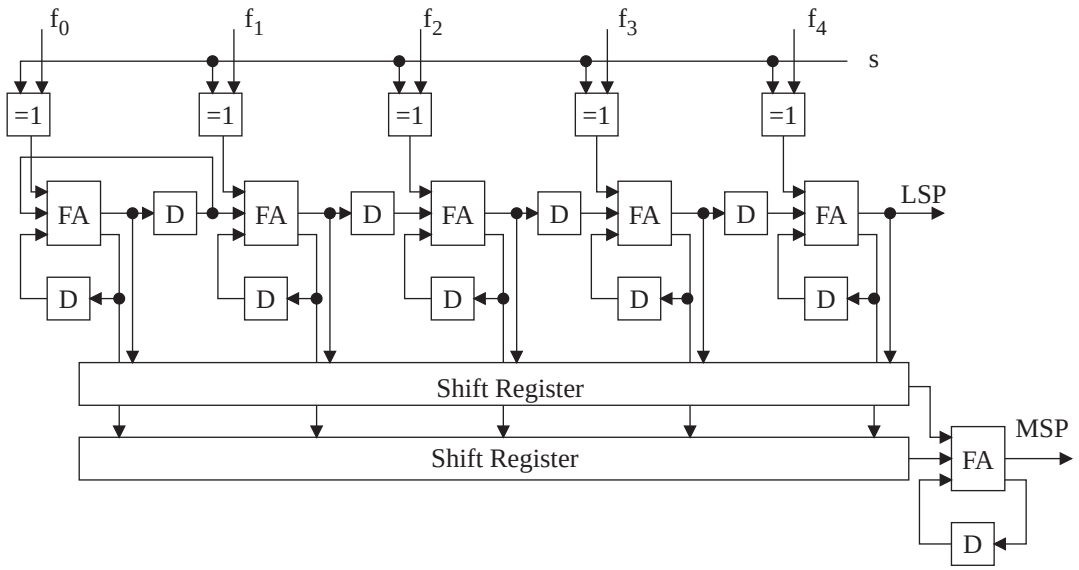


For F_0, the clock cycle corresponding to the sign bit of the data, F_0 should be subtracted.

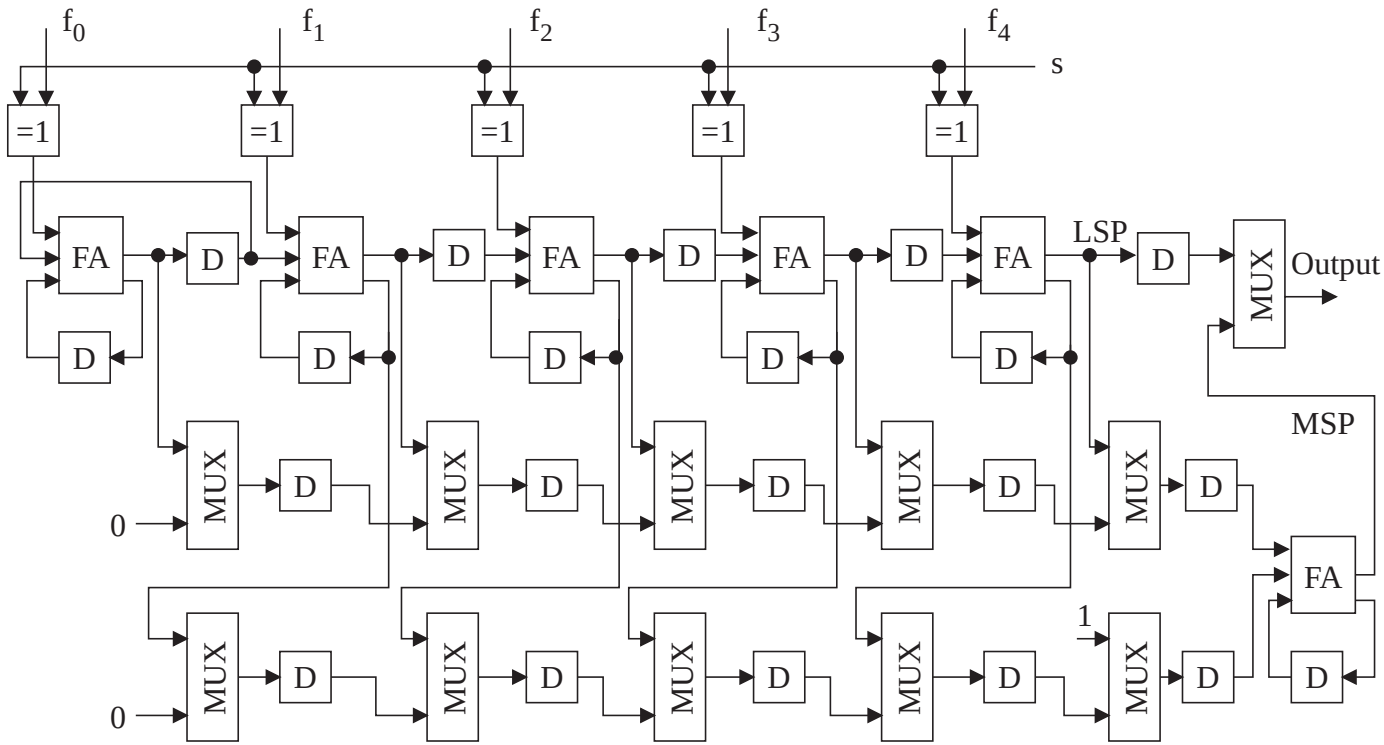
This is done by adding $-F_0$, i.e., inverting all the bits in F_0 using the XOR gates and the signal s, and adding one bit in the least-significant position.

After $-F_0$ has been added, the most significant part of the inner product must be shifted out of the accumulator. This can be done by accumulating zeros. The number of clock cycles for one inner product is W_d+W_{ROM}.

A more efficient scheme is to free the carry–save adders in the accumulator by loading the sum and carry bits of the carry–save adders into two shift registers. The outputs from these can be added by a single carry–save adder.



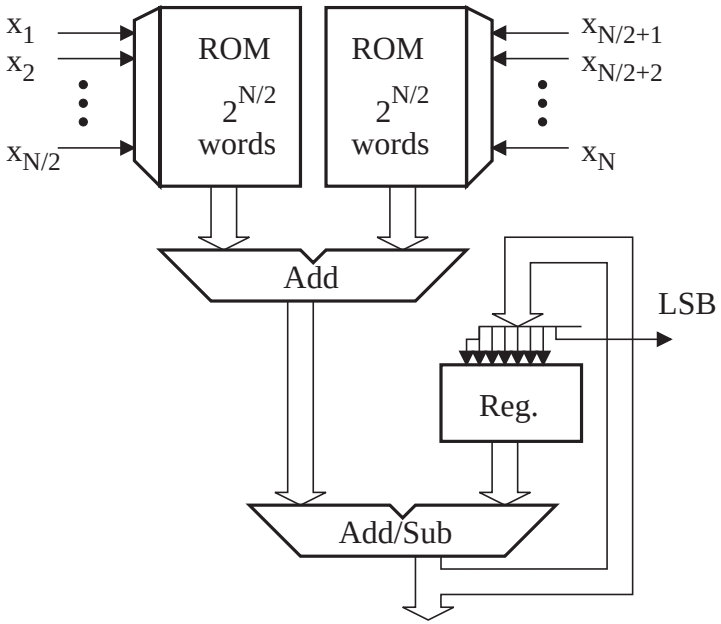
This scheme effectively doubles the throughput since two inner products are computed concurrently for a small increase in chip area.



REDUCING THE MEMORY SIZE

Memory Partitioning

One of several possible ways to reduce the overall memory requirement is to partition the memory into smaller pieces that are added before the shift-accumulator.



The amount of memory is reduced from 2^N words to $2 \cdot 2^{N/2}$ words if the original memory is partitioned into two parts.

For example, for $N = 10$ we get

$2^{10} = 1024$ words to $2 \cdot 2^5 = 64$ words.

Hence, this approach reduces the memory significantly at the cost of an additional adder.

Memory Coding

The second approach is based on a special coding of the ROM content. Memory size can be halved by using the ingenious scheme based on the identity

$$\boldsymbol{x} = \frac{1}{2}[\boldsymbol{x} - (-\boldsymbol{x})]$$

In two's-complement representation the identity can be rewritten

$$\boldsymbol{x} = \frac{1}{2}\left[-x_0 + \sum_{k=1}^{W_d-1} x_k 2^{-k} - \left(-\overline{x_0} + \sum_{k=1}^{W_d-1} \overline{x_k} 2^{-k} + 2^{-W_d+1}\right)\right] =$$

$$= -(x_0 - \overline{x_0})2^{-1} + \sum_{k=1}^{W_d-1} (x_k - \overline{x_k})2^{-k-1} - 2^{-W_d}$$

Notice that $(x_k - \overline{x_k})$ can only take on the values –1 or +1.

Inserting this expression into the inner product yields

$$y = \sum_{k=1}^{W_d - 1} F_k(x_{1k}, \ldots, x_{Nk}) 2^{-k-1} - F_0(x_{10}, \ldots, x_N) 2^{-1} + F(0, \ldots, 0) 2^{-W_d}$$

where $F_k(x_{1k}, x_{2k}, \ldots, x_{Nk}) = \sum_{i=1}^{N} a_i (x_{ik} - \overline{x_{ik}})$

The function F_k is shown in the table for $N = 3$.

x_1	x_2	x_3	F_k
0	0	0	$-a_1 - a_2 - a_3$
0	0	1	$-a_1 - a_2 + a_3$
0	1	0	$-a_1 + a_2 - a_3$
0	1	1	$-a_1 + a_2 + a_3$
1	0	0	$+a_1 - a_2 - a_3$
1	0	1	$+a_1 - a_2 + a_3$
1	1	0	$+a_1 + a_2 - a_3$
1	1	1	$+a_1 + a_2 + a_3$

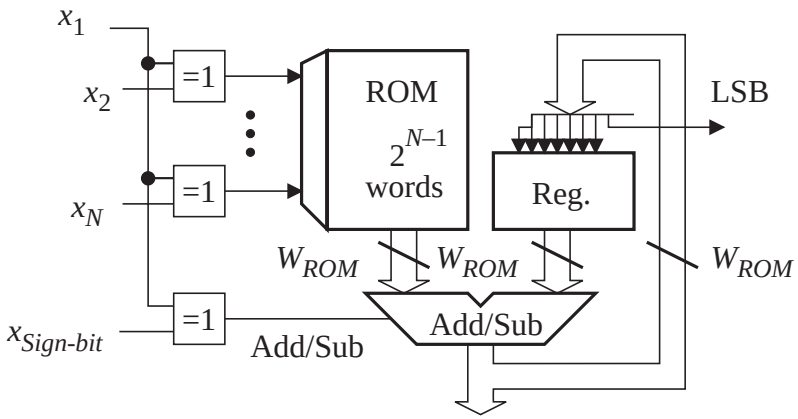
Antisymmetry

Notice that only half the values are needed, since the other half can be obtained by changing the signs.

To explore this redundancy we make the following address modification shown to the right in the table below.

$u_1 = x_1 \oplus x_2$ $\quad u_2 = x_1 \oplus x_3$ $\quad A/S = x_1 \oplus x_{sign-bit}$

x_1	x_2	x_3	F_k	u_1	u_2	A/S
0	0	0	$-a_1 - a_2 - a_3$	0	0	A
0	0	1	$-a_1 - a_2 + a_3$	0	1	A
0	1	0	$-a_1 + a_2 - a_3$	1	0	A
0	1	1	$-a_1 + a_2 + a_3$	1	1	A
1	0	0	$+a_1 - a_2 - a_3$	1	1	S
1	0	1	$+a_1 - a_2 + a_3$	1	0	S
1	1	0	$+a_1 + a_2 - a_3$	0	1	S
1	1	1	$+a_1 + a_2 + a_3$	0	0	S



Distributed arithmetic with halved ROM.

Only N–1 variables are used to address the memory.

COMPLEX MULTIPLIERS

Classical solution require 3 real multiplications and two adder networks.

Let

$$\boldsymbol{X} = a + jb \quad \text{and} \quad \boldsymbol{K} = c + jd$$

where $\boldsymbol{K}$ is the fixed coefficient and $\boldsymbol{X}$ is the variable. Once again we use the identity

$$\mathbf{x} = \frac{1}{2}[\boldsymbol{x} - (-\boldsymbol{x})] = \frac{1}{2}\left[-x_0 + \sum_{k=1}^{W_d-1} x_k 2^{-k} - \left(-\overline{x_0} \sum_{k=1}^{W_d-1} \overline{x_k} 2^{-k} + 2^{-W_d+1}\right)\right] =$$

$$= -(x_0 - \overline{x_0})2^{-1} + \sum_{k=1}^{W_d-1} (x_k - \overline{x_k})2^{-k-1} - 2^{-W_d}$$

Now, the product of two complex numbers can be written

$$\boldsymbol{K} \cdot \boldsymbol{X} = (ca - db) + j(da + cb)$$

$$= \left\{ -c(a_0 - \overline{a_0})2^{-1} + \sum_{i=1}^{W_d - 1} c(a_i - \overline{a_i})2^{-i-1} - c2^{-W_d} \right\} -$$

$$- \left\{ -d(b_0 - \overline{b_0})2^{-1} + \sum_{i=1}^{W_d - 1} d(b_i - \overline{b_i})2^{-i-1} - d2^{-W_d} \right\} +$$

$$+ j\left\{ -d(a_0 - \overline{a_0})2^{-1} \sum_{i=1}^{W_d - 1} d(a_i - \overline{a_i})2^{-i-1} - d2^{-W_d} \right\} +$$

$$+ j\left\{ -c(b_0 - \overline{b_0})2^{-1} + \sum_{i=1}^{W_d - 1} c(b_i - \overline{b_i})2^{-i-1} - c2^{-W_d} \right\} =$$

$$= \left\{ -F_1(a_o, b_0)2^{-1} + \sum_{i=1}^{W_d - 1} F_1(a_i, b_i)2^{-i-1} + F_1(0,0)2^{-W_d} \right\} +$$

$$+ j\left\{ -F_2(a_0, b_0)2^{-1} + \sum_{i=1}^{W_d - 1} F_2(a_i, b_i)2^{-i-1} + F_2(0,0)2^{-W_d} \right\}$$

Hence, the real and imaginary parts of the product can be computed using just two distributed arithmetic units.

The binary functions F_1 and F_2 can be stored in a ROM, addressed by the bits a_i and b_i. The ROM content is

a_i	b_i	F_1	F_2
0	0	$-(c-d)$	$-(c+d)$
0	1	$-(c+d)$	$(c-d)$
1	0	$(c+d)$	$-(c-d)$
1	1	$(c-d)$	$(c+d)$

Antisymmetry

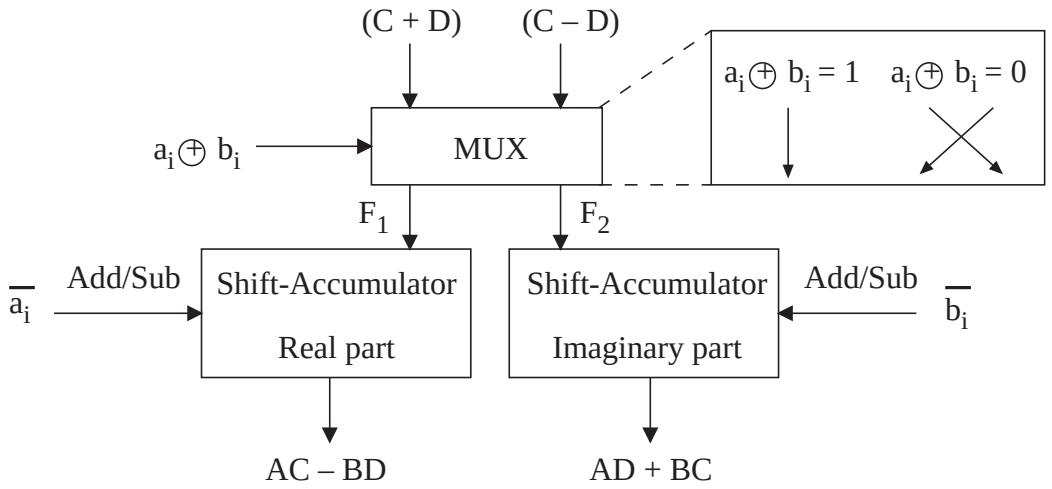
It is obvious from the table that only two coefficients are needed, (c+d) and (c–d).

The appropriate coefficients can be directed to the accumulators via a 2:2-multiplexer.

If $a_i \oplus b_i = 1$ the F values are applied directly to the accumulators, and

if $a_i \oplus b_i = 0$ the F values are interchanged.

The F values are either added to, or subtracted from, the accumulator's registers depending on the data bits a_i and b_i.

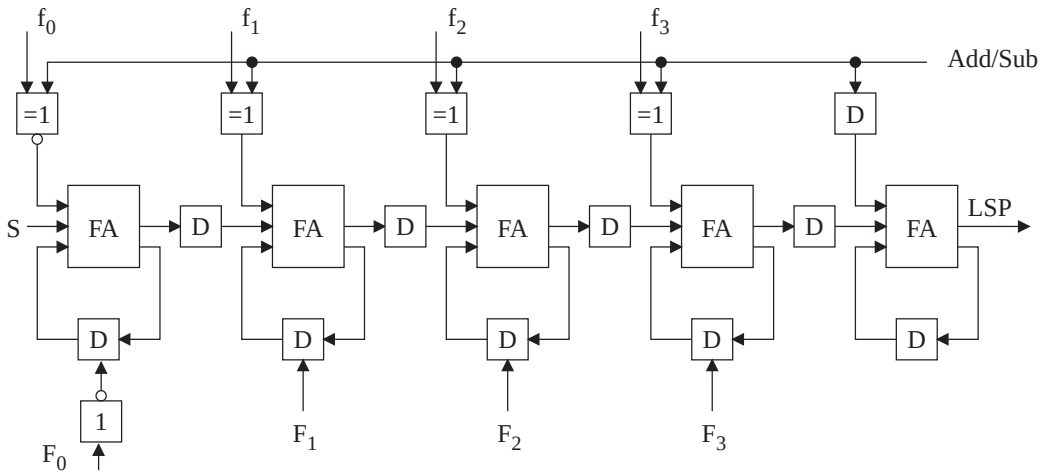


IMPROVED SHIFT-ACCUMULATOR

The last term in the real part (and the same for the imaginary part)

$$KA = \left\{ -F_1(a_o, b_0)2^{-1} + \sum_{i=1}^{W_d - 1} F_1(a_i, b_i)2^{-i-1} + F_1(0,0)2^{-W_d} \right\} + \text{imaginary part}$$

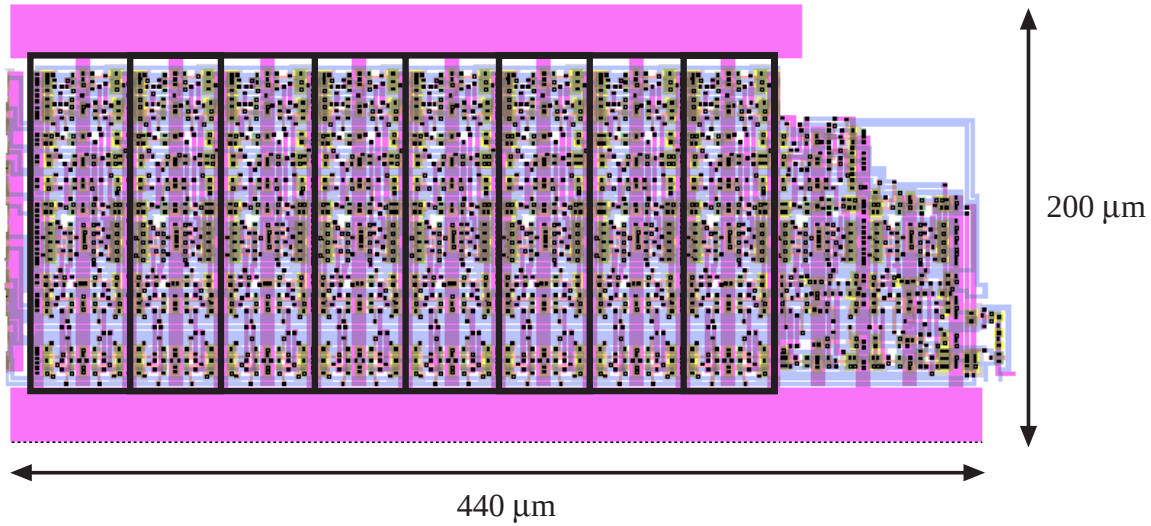
shall be added to the first term in the sum, F_{Wd-1}, at the same level of significance. This can be accomplished by initially setting the carry D flip-flops to $F(0, 0,..., 0)$, as illustrated below where only the upper part of the shift-accumulator part is shown.



Complex Multiplier Using Two-Phase Logic

Layout of one half of a complex multiplier based on the improved shift-accumulator using two-phase clocking. The coefficient word length is 8 bits. A 0.8-µm double metal CMOS process was used. The maximal clock frequency are about 175 MHz at 5 V.

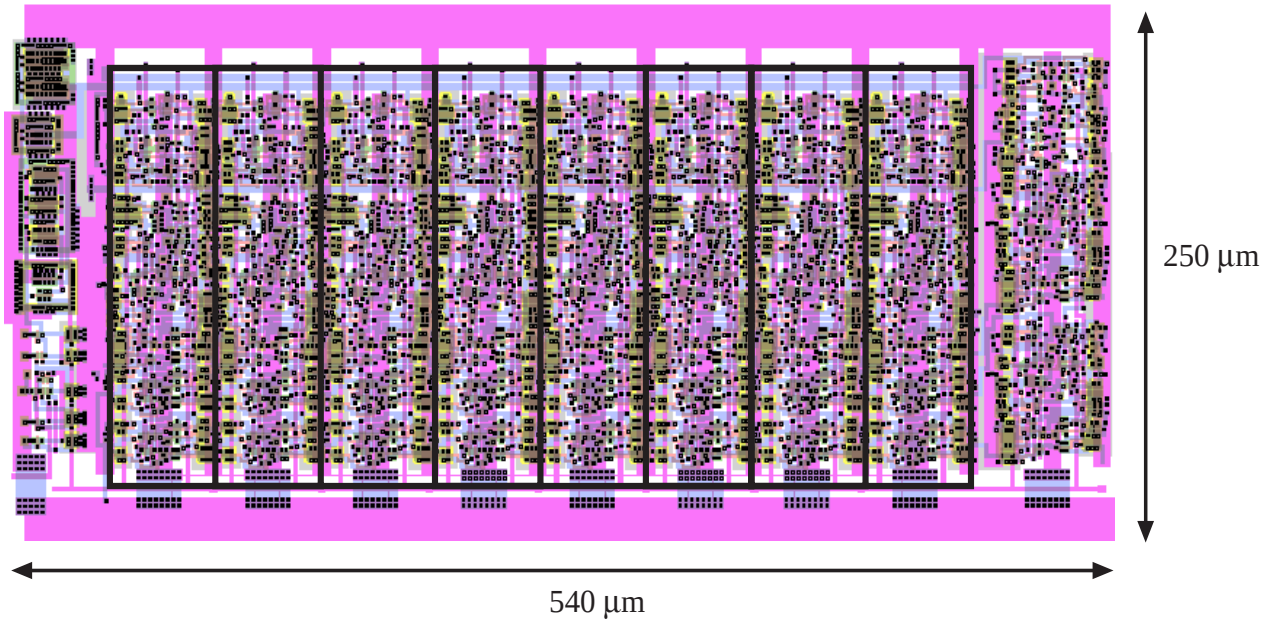
The chip area is 440 µm × 200 µm = 0.088 mm^2.



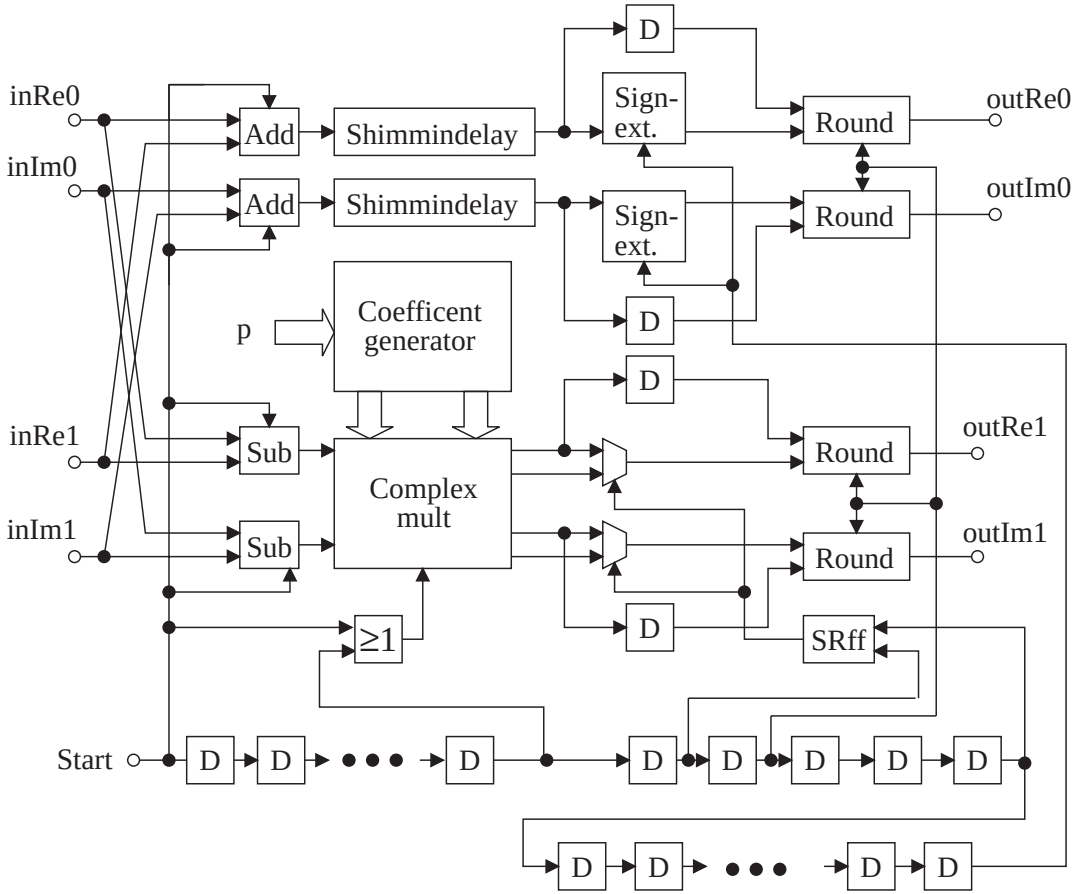
Complex Multiplier Using TSPC Logic

Layouts of one-half of a complex multiplier based on the improved shift-accumulator using TSPC (True Single-Phase Clocking).

The maximal clock frequency is about 400 MHz at 5 V. The chip area is 540 µm × 250 µm = 0.135 mm^2. Drivers for the clock estimated to require an additional 0.052 mm^2.



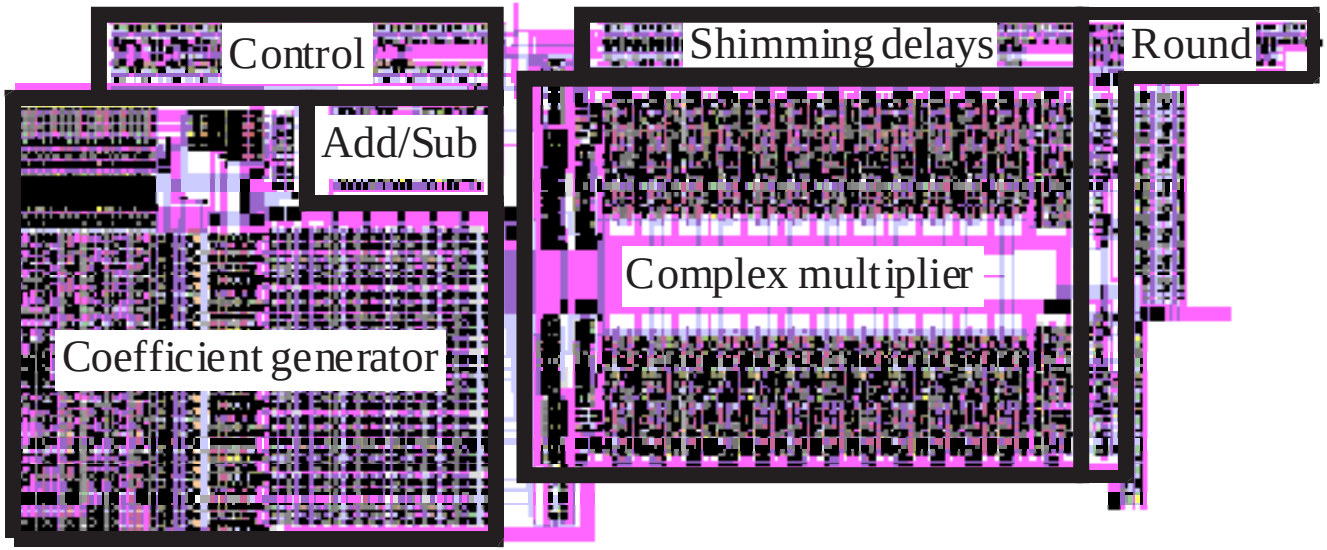
FFT PROCESSOR, CONT.



The decimation-in-frequency radix-2 bit-serial butterfly PE has been implemented in a 0.8 µm standard CMOS process.

The PE has a built-in coefficient generator that can generate all twiddle factors in the range 0 to 128, which is sufficient for a 1024-point FFT.

The layout using AMS 0.8-µm double metal CMOS process is shown below. It is clear that the coefficient generator and the complex multiplier occupy most of the area. The area is 1.47 mm^2.



The maximal clock frequency at 3 V supply voltage is 133 MHz with a power consumption of 30 mW (excluding the power consumed by the clock).

80% of the power is consumed in the complex multiplier and 5% in the coefficient generator.

The rest (15 %) is evenly distributed in the rest of the butterfly.

The D flip-flops and the gates at the bottom of the block diagram are the control.

Twiddle Factor PE

Twiddle factors can be generated in several ways: by using a CORDIC PE, via trigonometric formulas, or read from a precomputed table. Here we will use the latter method—that is, a PE that essentially consists of a ROM.

We have previously shown that it is possible to use only one W^p PE.

However, here it is better to use one for each butterfly PE, because the required chip area for a ROM is *relatively small*.

If only one ROM were used, it would have been necessary to use long bit-parallel buses, which are costly in terms of area, to distribute the twiddle factors to the butterfly PEs.

The values of the twiddle factors, $\boldsymbol{W}$, are spaced uniformly around the unit circle. Generally, there are N twiddle factors, but it is possible to reduce the number of unique values by exploring symmetries in the trigonometric functions. In fact, it can be shown that only $N/8 + 1$ coefficient values need be stored in a table.

Instead of storing W^p, we will store the values

$$\frac{C(a)+S(a)}{2} = \frac{1}{2}(\cos(a)-\sin(a)) = \frac{1}{\sqrt{2}}\sin\left(a-\frac{\pi}{4}\right)$$

$$\frac{C(a)-S(a)}{2} = \frac{1}{2}(\cos(a)+\sin(a)) = \frac{1}{\sqrt{2}}\sin\left(\frac{\pi}{4}+a\right)$$

where, $a = 2\pi p/N$.

The twiddle factors in the eight octants can be expressed in terms of the twiddle factors in the range 0 to $\pi/4$.

Octant	a	$a_0a_1a_2$	b	$\frac{C+S}{2}$	$\frac{C-S}{2}$
0	$0 \le a \le \frac{\pi}{4}$	000	a	$\frac{1}{\sqrt{2}}\sin\left(\frac{\pi}{4}-b\right)$	$\frac{1}{\sqrt{2}}\cos\left(\frac{\pi}{4}-b\right)$
1	$\frac{\pi}{4} \le a \le 2\frac{\pi}{4}$	001	$a-\frac{\pi}{4}$	$\frac{-1}{\sqrt{2}}\sin(b)$	$\frac{1}{\sqrt{2}}\cos(b)$
2	$2\frac{\pi}{4} \le a \le 3\frac{\pi}{4}$	010	$a-2\frac{\pi}{4}$	$\frac{-1}{\sqrt{2}}\cos\left(\frac{\pi}{4}-b\right)$	$\frac{1}{\sqrt{2}}\sin\left(\frac{\pi}{4}-b\right)$
3	$3\frac{\pi}{4} \le a \le 4\frac{\pi}{4}$	011	$a-3\frac{\pi}{4}$	$\frac{-1}{\sqrt{2}}\cos(b)$	$\frac{-1}{\sqrt{2}}\sin(b)$

DCT PROCESSOR, CONT.

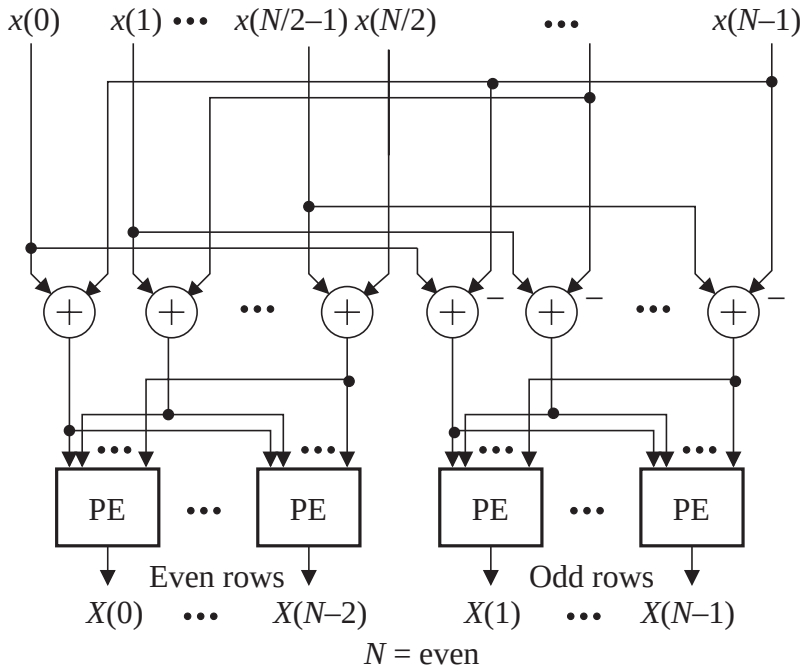
The chip area needed to implement a vector-multiplier using distributed arithmetic grows as $O(2^N)$ where N is the number of terms in the inner product.

The chip area required for implementing a one-dimensional MSDCT can be reduced by exploiting the symmetry (antisymmetry) of the basis functions.

A detailed study shows that the basic functions exhibit both symmetry and antisymmetry in their coefficients.

Using the same principle that was used in the linear-phase FIR filter structure, the pixels that are multiplied by the same coefficient are added (or subtracted).

This reduces the number of terms in the remaining inner products by 50%. The chip area is thereby reduced from $O(2^N)$ to $O(2^{N/2})$, which is a significant reduction.



A 2-D DCT for 16×16 pixels can be built using only one 1-D DCT PE which itself consists of 16 distributed arithmetic units with $N = 8$.

The TSPC based shift-accumulator can be used to implement a distributed arithmetic unit. The length of the shift-accumulator depends on the word length, W_{ROM}, which depends on the coefficients in the vector-products. In this case we assume that $W_{ROM} = W_c+1 = 12$ bits.