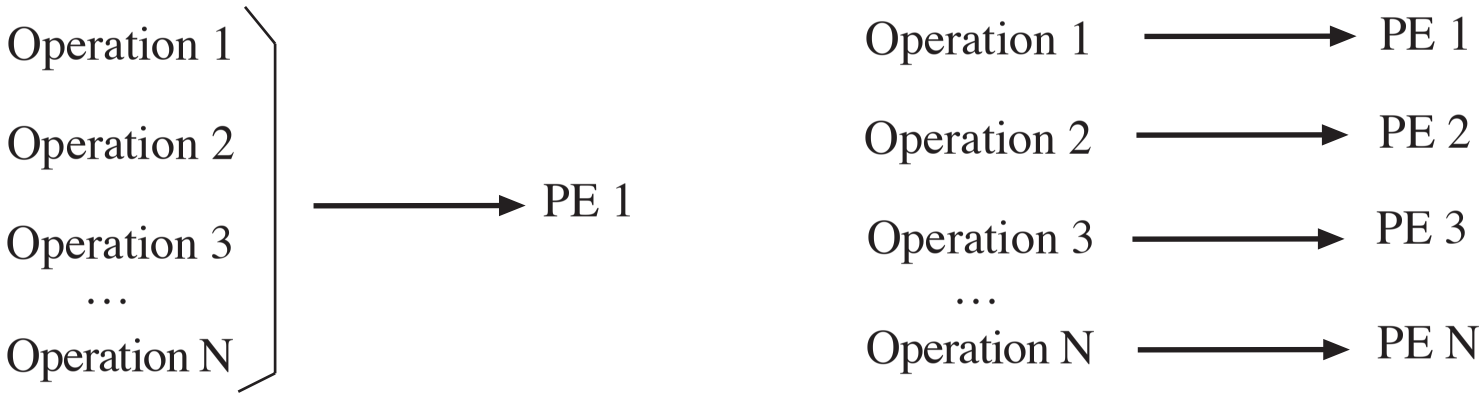
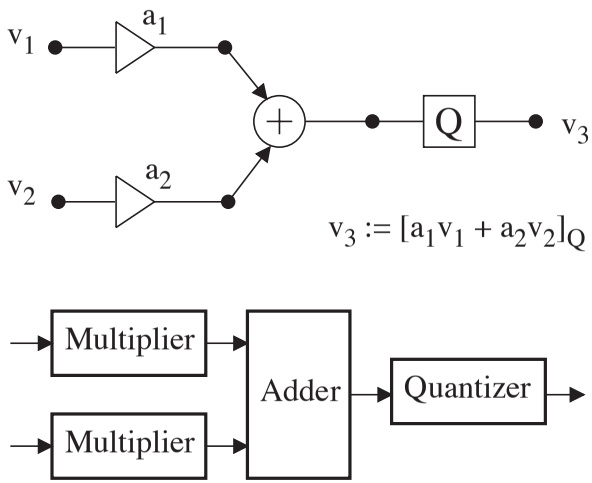
# SYNTHESIS OF FIXED DSP CORES

## MAPPING OF DSP ALGORITHMS ONTO HARDWARE

In practice, mapping of operations to hardware structures



# ISOMORPHIC MAPPING OF SFGs



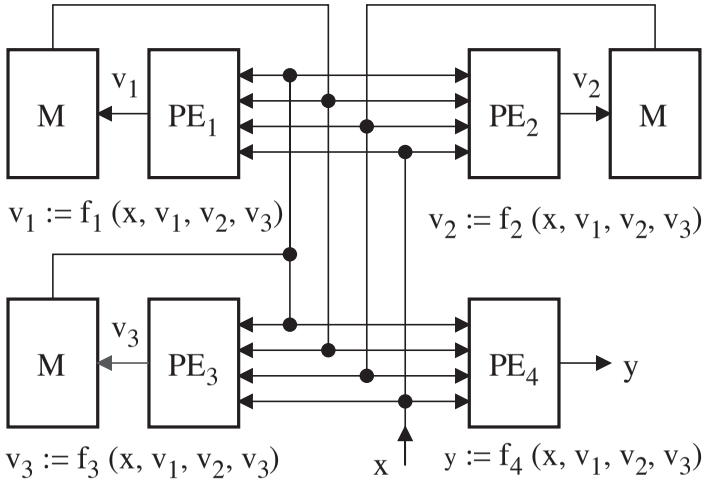

The PEs are interconnected according to the signal-flow graph.

There is thus an isomorphy between the interconnection network and the signal-flow graph.

Does not generally lead to maximally fast implementations or a high resource utilization unless the scheduling of the operations are performed over multiple sample intervals.

# IMPLEMENTATIONS BASED ON COMPLEX PEs

The basic approach is to use a dedicated PE for each value in the algorithm that has to be computed explicitly.



# Vector-Multiplier-Based Implementations

The basic operation used in digital filters and many other DSP algorithms is the sum-of-products

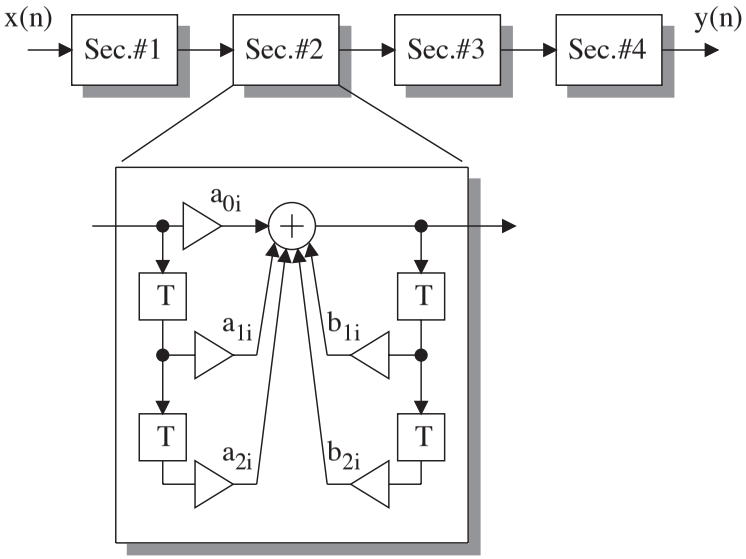
$$y = \sum_{i=1}^{N} a_i x_i = \mathbf{a}\ \mathbf{x}$$

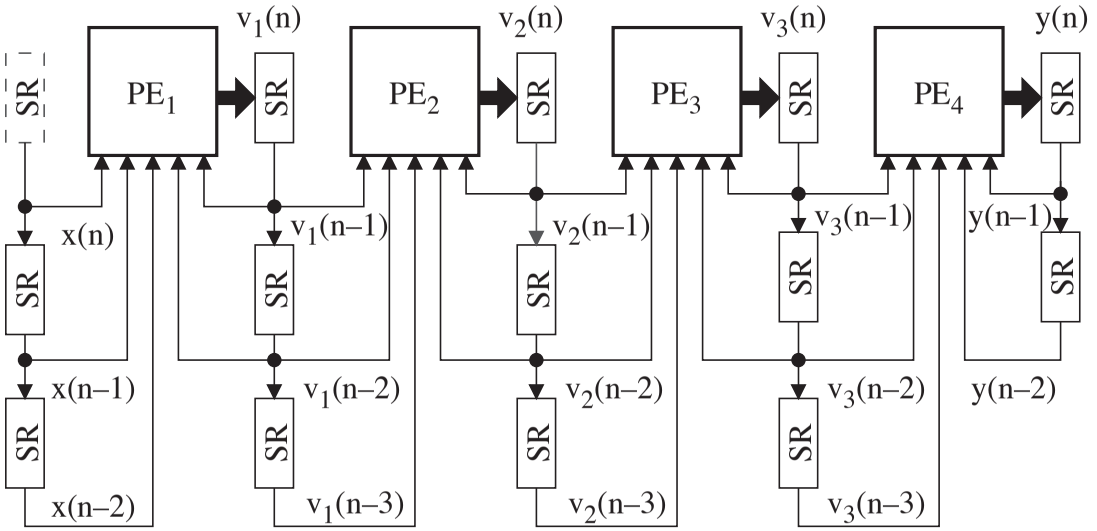
which also can be viewed as a vector multiplication, or inner product, or dot product, between a constant vector $\mathbf{a}$ and a data vector $\mathbf{x}$.

A vector multiplication can be implemented efficiently using distributed arithmetic.

# Example 9.5

The bandpass filter was implemented in cascade form with four second-order sections in direct form I. The numbers of scalar multipliers and adders for a classical implementation are 20 and 16, respectively. The number of shift registers is 14.



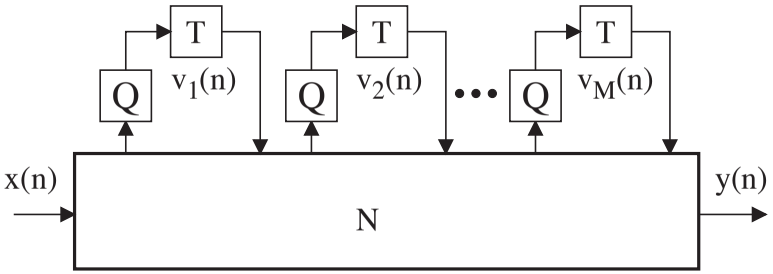



# Numerically Equivalent Implementation

The numerically equivalent algorithm is equivalent to, or in some respects even better than, numerical properties compared to the original algorithm.

Consider the recursive digital filter below.

Arithmetic operations are represented by the network *N with quantized coefficients*.




All values, $v_i(n)$, are computed with full precision and that the quantizations are done only in front of the delay elements in the original algorithm.

Hence, no overflow or rounding errors occur, and the outputs and $v_i(n)$, $i = 1,\ldots, M$ are computed exactly before they are quantized.

The node values, which must be computed explicitly, are the outputs and the values that shall be stored in the delay elements.

These values can be computed in either of two numerically equivalent ways

- by doing the multiplications using extended precision so that no errors occur,
- by precomputing new *equivalent coefficients* we get a set of expressions with fewer multiplications, but usually more than in the original algorithm. Thus, the new algorithm is numerically equivalent to the original since the quantization errors that occur in the two algorithms are the same.

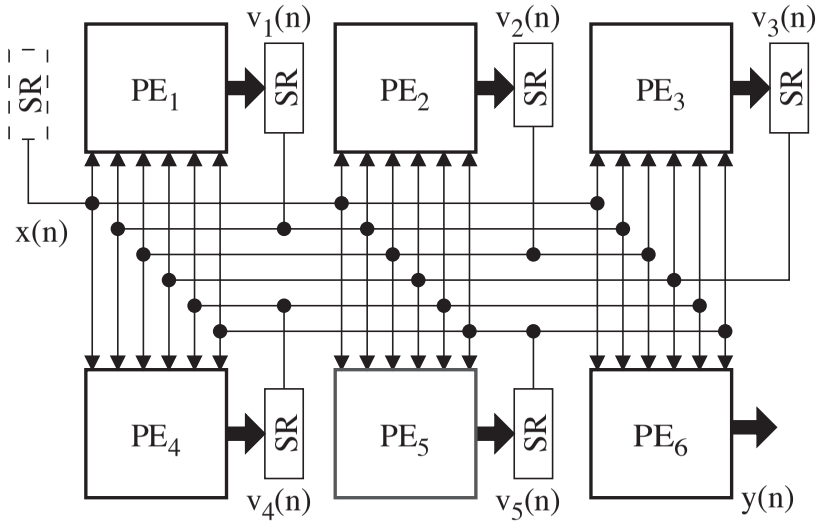
The new algorithm can be described by the state-space equations

$$\begin{cases} \mathbf{v}(n+1) = \mathbb{A}\mathbf{v}(n) + \mathbb{B}x(n) \\ y(n) = \mathbb{C}\mathbf{v}(n) + \mathbb{D}x(n) \end{cases}$$

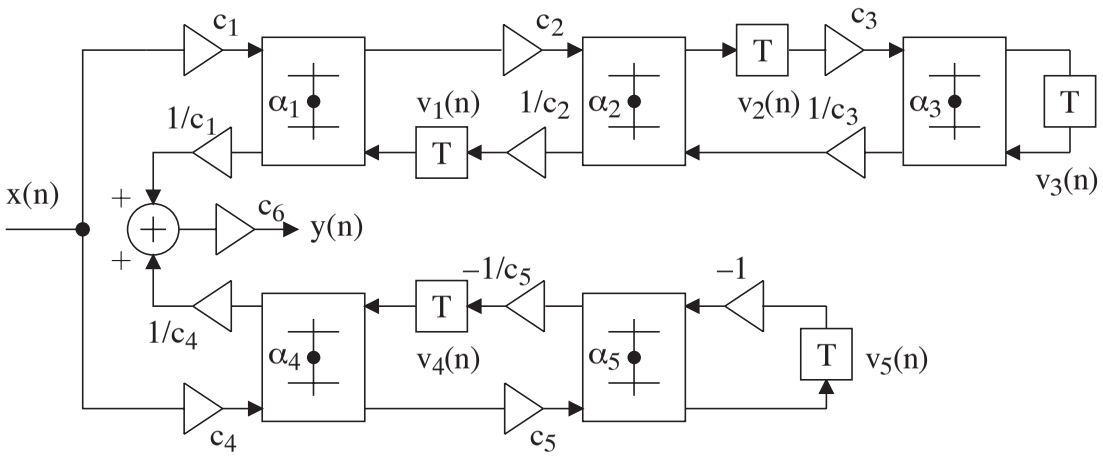
where $\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$, $\mathbb{D}$ and $\mathbf{v}$ are matrices and vectors with dimensions $M \times M$, $M \times 1$, $1 \times M$, $1 \times 1$, and $M \times 1$, respectively.

# Example with $M = 5$

$$\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \\ v_3(n+1) \\ v_4(n+1) \\ v_5(n+1) \\ y(n) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & b_{11} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & b_{21} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & b_{31} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & b_{41} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & b_{51} \\ c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & d_{11} \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \\ v_3(n) \\ v_4(n) \\ v_5(n) \\ x(n) \end{bmatrix}$$



# Numerically Equivalent Implementation of WDFs




Lattice WDF with the branches implemented using Richards' structures

$$\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \\ v_3(n+1) \\ v_4(n+1) \\ v_5(n+1) \\ y(n) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & b_{11} \\ a_{21} & a_{22} & a_{23} & 0 & 0 & b_{21} \\ 0 & a_{32} & a_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} & b_{41} \\ 0 & 0 & 0 & a_{54} & a_{55} & b_{51} \\ c_{11} & 0 & 0 & c_{14} & 0 & d_{11} \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \\ v_3(n) \\ v_4(n) \\ v_5(n) \\ x(n) \end{bmatrix}$$

$$\begin{bmatrix} v_1(n+1) \\ v_2(n+1) \\ v_3(n+1) \\ v_4(n+1) \\ v_5(n+1) \\ y(n) \end{bmatrix} = \begin{bmatrix} \frac{819}{1024} & \frac{-61}{512} & \frac{125}{4096} & 0 & 0 & \frac{189}{512} \\ \frac{1651}{8192} & \frac{3843}{4096} & \frac{-7875}{32768} & 0 & 0 & \frac{381}{4096} \\ 0 & \frac{3}{8} & \frac{61}{64} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{3255}{4096} & \frac{-1}{4} & \frac{-7223}{16384} \\ 0 & 0 & 0 & \frac{-6615}{32768} & \frac{31}{32} & \frac{14679}{131072} \\ \frac{29}{64} & 0 & 0 & \frac{23}{64} & 0 & \frac{1}{256} \end{bmatrix} \begin{bmatrix} v_1(n) \\ v_2(n) \\ v_3(n) \\ v_4(n) \\ v_5(n) \\ x(n) \end{bmatrix}$$