PART I: Why there is a need for developing algorithms at the system level? – Efficient DSP algorithms and their optimization for signal processor and VLSI implementations

---

- This part serves as an introduction to the course.

IEEE CAS DISTINGUISHED LECTURE PROGRAM

Tapio Saramäki
Institute of Signal Processing
Tampere University of Technology

Lecture # 1:

EFFICIENT DSP ALGORITHMS AND THEIR OPTIMIZATION FOR VLSI AND SIGNAL PROCESSOR IMPLEMENTATION

1) Historical background

2) How to generate efficient algorithms and products based on the use of these algorithms

3) Some hints for developing efficient implementations of DSP Algorithms

4) Theory versus Practice

EFFICIENT DSP ALGORITHMS AND THEIR OPTIMIZATION FOR VLSI AND SIGNAL PROCESSOR IMPLEMENTATIONS

Tapio Saramäki

Signal Processing Laboratory

Tampere University of Technology

P. O. Box 553, FIN-33101 Tampere, Finland

E-mail: ts@cs.tut.fi

- Historical background

- How to generate efficient algorithms and products based on use of these algorithms

- Some hints for developing efficient implementations of DSP algorithms

- Theory versus Practice

HISTORICAL BACKGROUND

- Intensive research on DSP algorithms started two decades ago.

- However, it took almost these two decades before also small companies became interested in replacing their continuous-time systems by their digital counterparts.

- Advances in VLSI circuit technology as well as the development in signal processors have had a major impact on this.

- Nowadays, more complicated algorithms are implementable faster and faster as well as in a smaller and smaller silicon area.

- In order to accelarate this preferable development, also the algorithms must be further developed by taking into account the bottlenecks of the implementation form, e.g., VLSI circuit and signal processor implementations.

- The knowledge obtainable from textbooks is not at all enough for generating more efficient implementations of DSP algorithms.

HOW TO GENERATE EFFICIENT ALGORITHMS AND PRODUCTS BASED ON THEM

- The generation of efficient algorithms and products is based on the following facts:

   I. A thorough knowlegde of the area as well as the state-of-art know how in order to be able to use intuition or a common sense thinking.

   II. Best ideas are normally simple and not generated starting with formulas!!

   III. Good tools (usually, optimization programs) to test the ideas as well as the tradeoffs between different factors affecting the practical implementation.

   IV. It is preferred to have a small team (3 to 5 persons) where each person has his own area of expertise. Furthermore, the team members must have a 'common language' and there must be a close feedback between the team members.

- Example: A stereo sigma-delta A/D converter designed by VLSI Solution Oy.

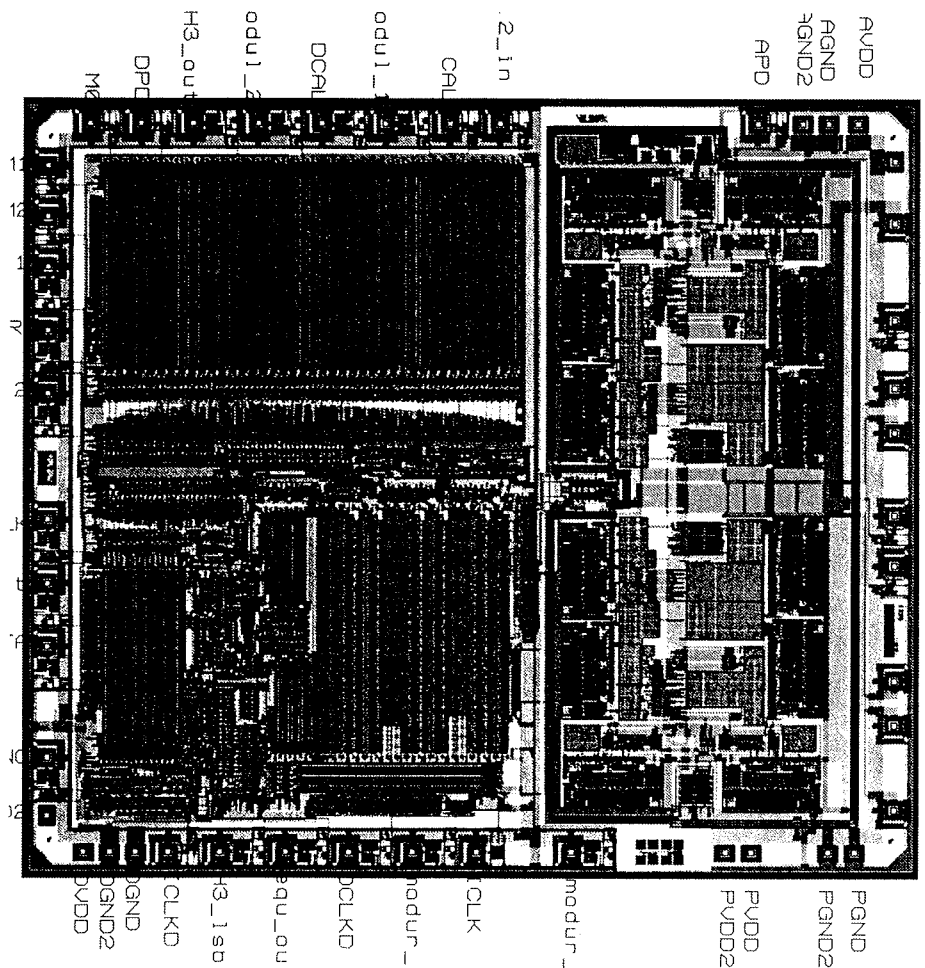# A stereo sigma-delta A/D converter for audio applications

Figure 1: AD-converter for audio use

# SOME HINTS FOR DEVELOPING EFFICIENT IMPLEMENTATIONS OF DSP ALGORITHMS

- **General Hints:**

  - Split the algorithms into very simple sub-algorithms.

  - Use sampling rate alteration whenever possible to both reduce the number of data samples to be treated and to make the actual algorithm simpler.

  - For instance, in the case of finite-impulse response filter, the filter order is reduced, and in the case of infinite-impulse response filters, the finite wordlength effects become much milder.

  - The following four transparencies exemplify the effectiveness of the sampling rate alteration.

Linear-Phase FIR Filter with passband and stopband edges at 250 Hz and 500 Hz and passband and stopband ripples of 0.01 and 0.001 for the amplitude response
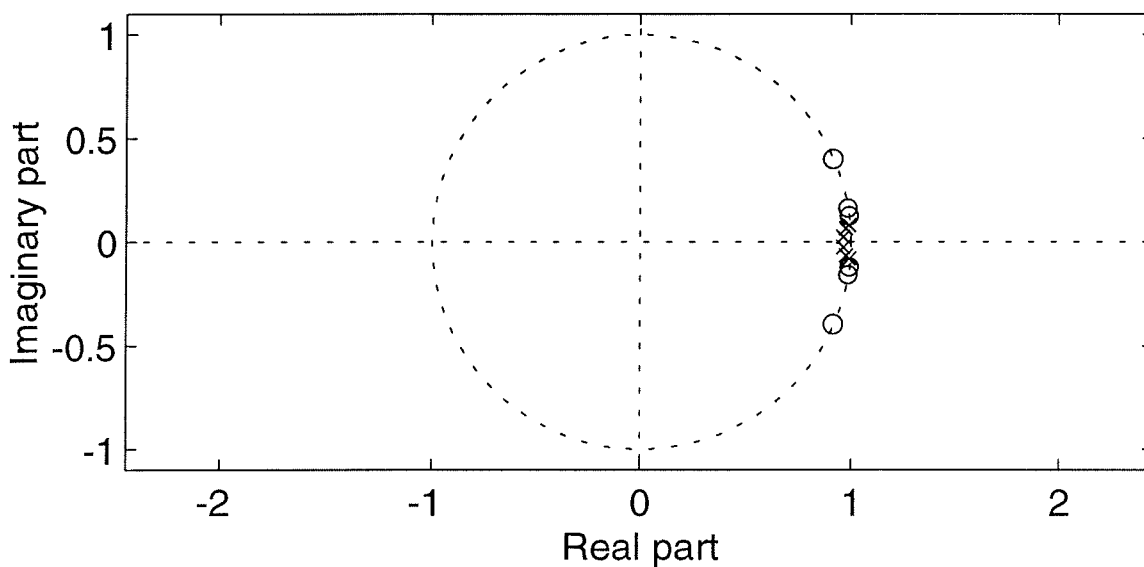

FIR Filter of Order 216 for fs=20kHz

FIR Filter of Order 21 for fs=2kHz
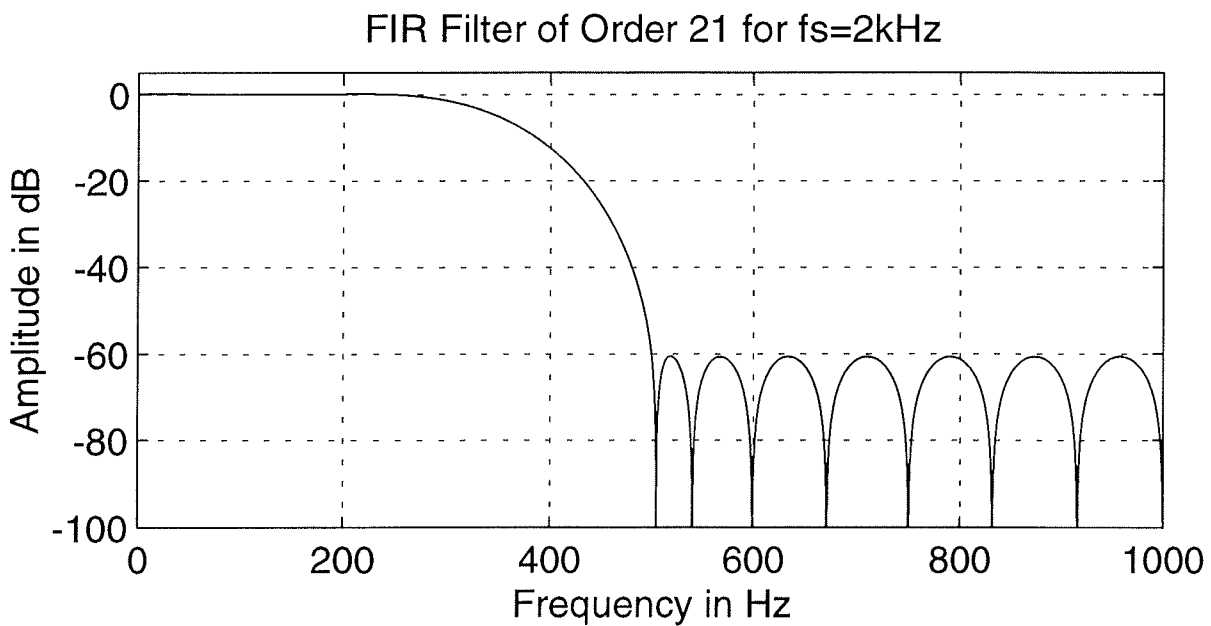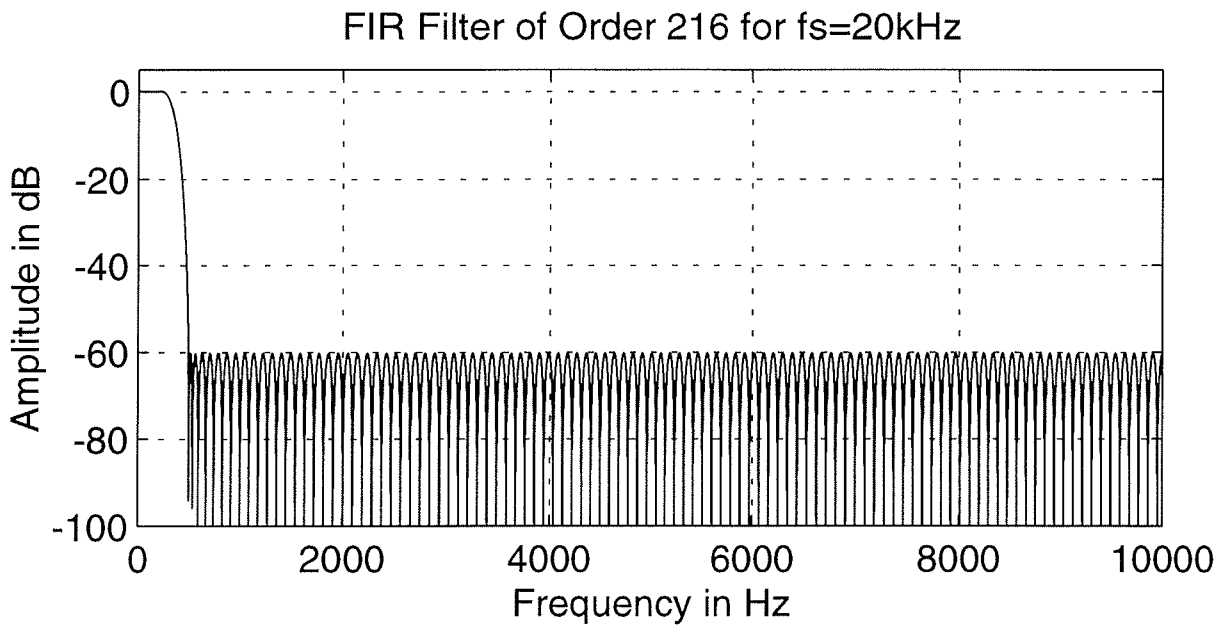
Elliptic Filter with passband and stopband edges at 250 Hz and 500 Hz and passband and stopband ripples of 0.2 dB and 60 dB

Elliptic Filter of Order 5 for fs=20kHz

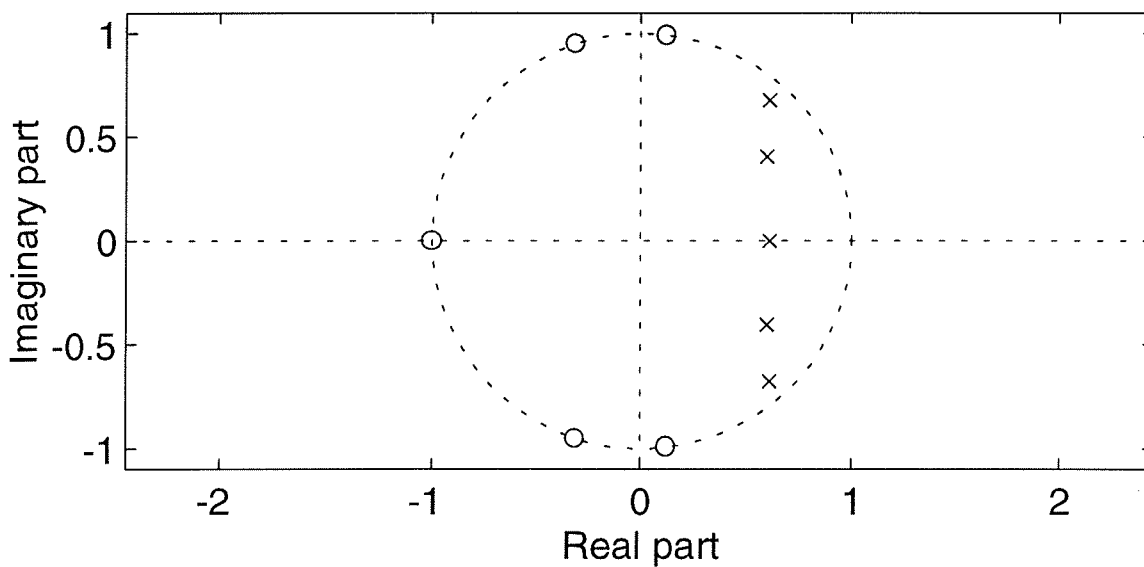Elliptic Filter of Order 5 for fs=2kHz
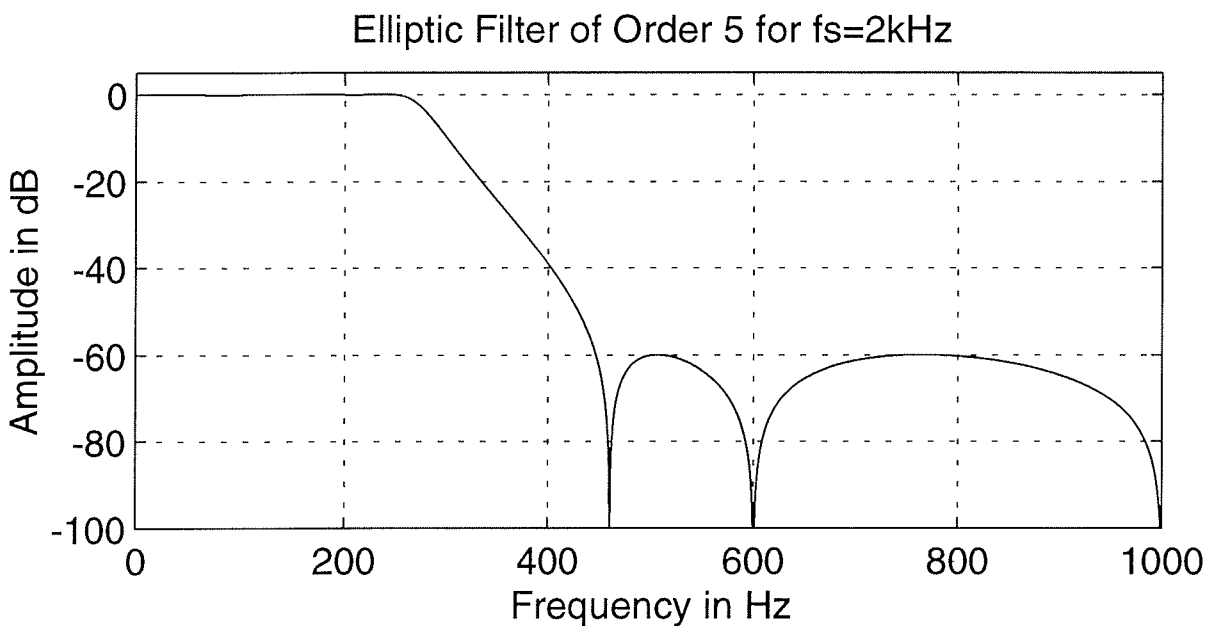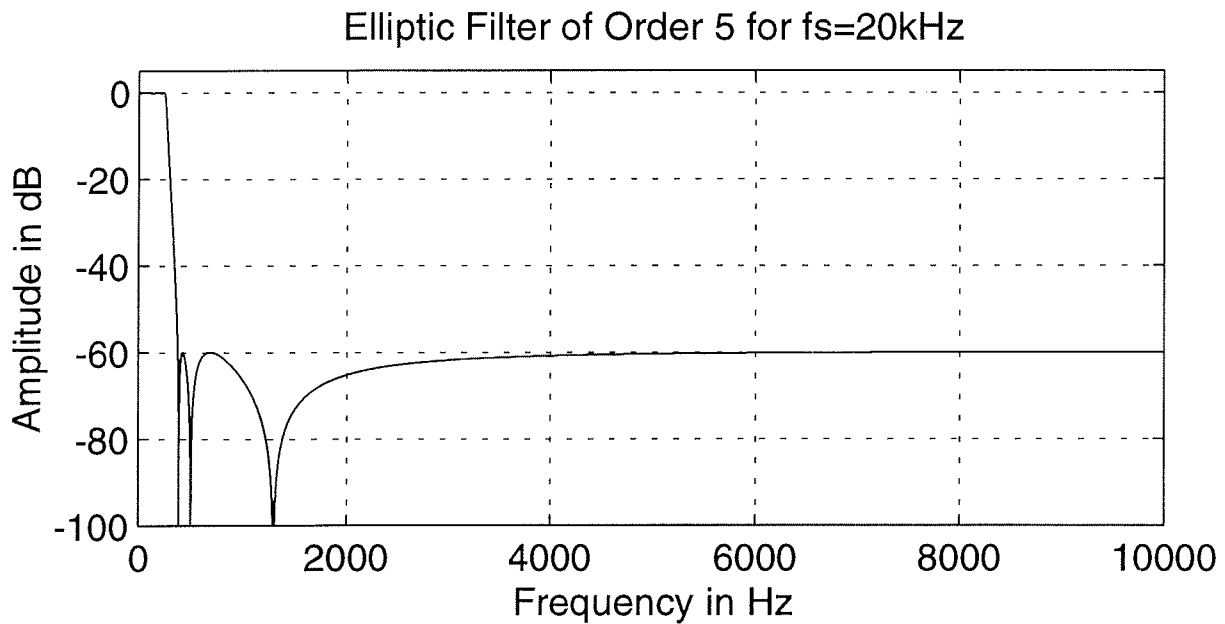
- For both $f_s = 20$ kHz and $f_s = 2$ kHz, the filter order is the same.

- Since the filter poles for $f_s = 2$ kHz are not so close to the unit circle, the coefficient sensisitivity as well as the output noise are much lower than for $f_s = 20$ kHz.

Elliptic Filter of Order 5 for fs=20kHz

Elliptic Filter of Order 5 for fs=2kHz

Design of a Narrowband Linear-Phase FIR Filter

Example: Passband edge = 0.0095 · $f_s/2$
Stopband edge = 0.01 · $f_s/2$
Passband ripple = 0.001
Stopband ripple = 0.0001 (80 dB)

Direct-form conventional FIR filter of order 15590:
7796 multiplications per input sample

FIR filter implemented using decimation and interpolation:

x(n) → $H_1(z)$ → ↓15 → $H_2(z)$ → ↓6 → $H_3(z)$

y(n) ← $15H_1(z)$ ← ↑15 ← $6H_2(z)$ ← ↑6 ←
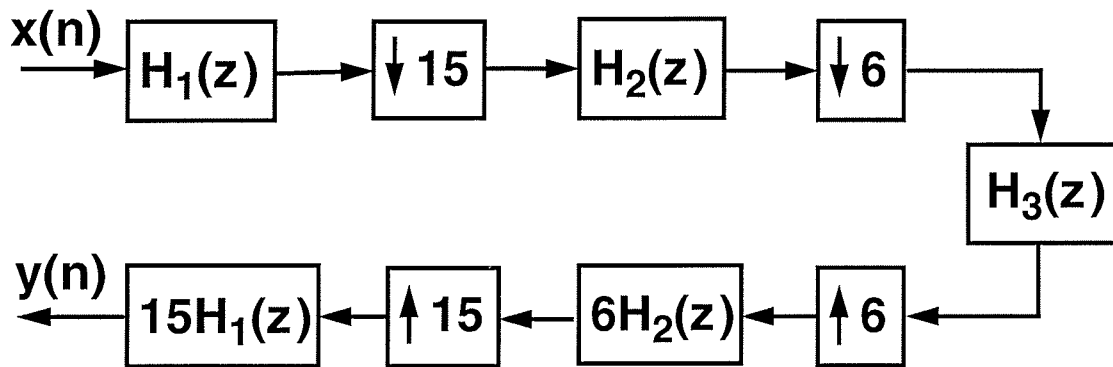
$H_1(z)$: order = 39

$H_2(z)$: order = 40

$H_3(z)$: order = 197

Only 4.16 multiplications per input sample

HINTS FOR SIGNAL PROCESSOR IMPLEMENTATIONS

- For most signal processors, during one instruction cycle, it is possible to implement either a unit delay or a multiple delay.

- Using this fact, it is attractive to use as building blocks filters obtainable from conventional filters by replacing a unit delay by a multiple delay.

- This reduces drastically the code length.

- The following two transparencies exemplify this.

EXAMPLE: FIR filter with $\omega_p = 0.4\pi$, $\omega_s = 0.402\pi$, $\delta_p = 0.01$ ($A_p = 0.17$ dB), $\delta_s = 0.0001$ ($A_s = 80$ dB).

- Linear-phase FIR filter: order 3138, 3139 multipliers
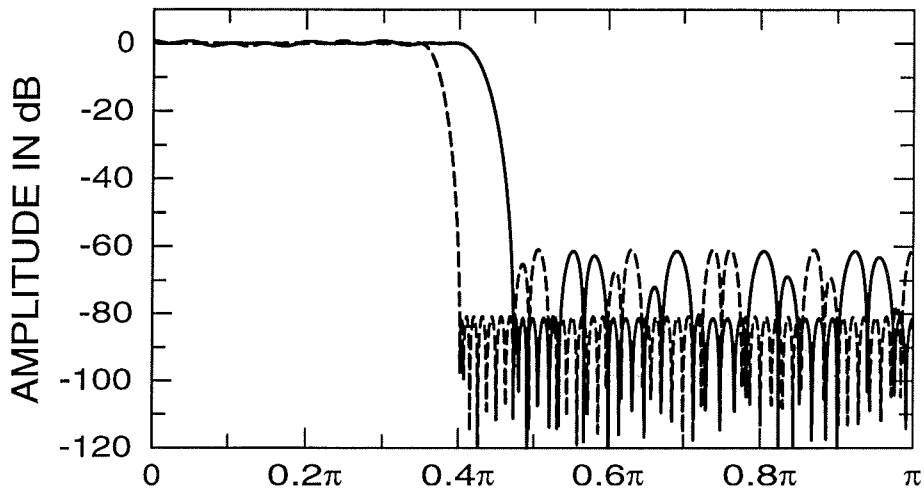
- Frequency-response masking approach
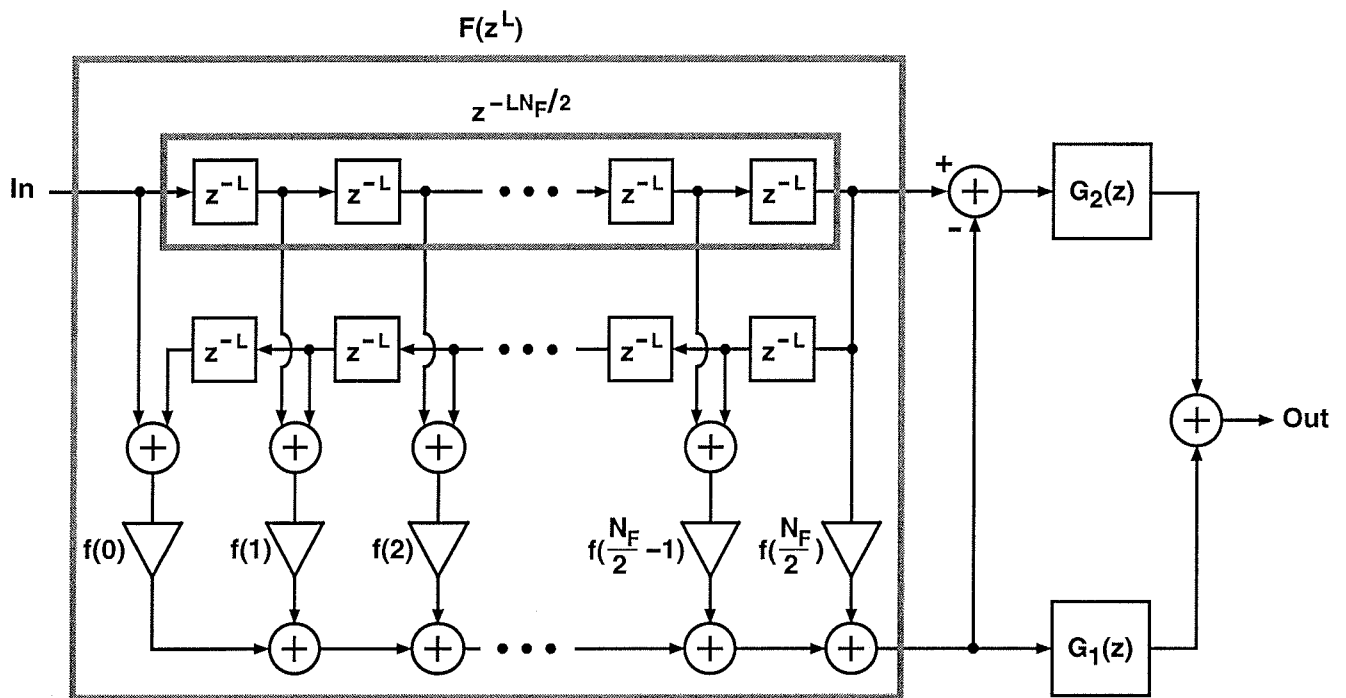
- $L = 16$, $F(z)$ of order 200, $G_1(z)$ of order 84, and $G_2(z)$ of order 122
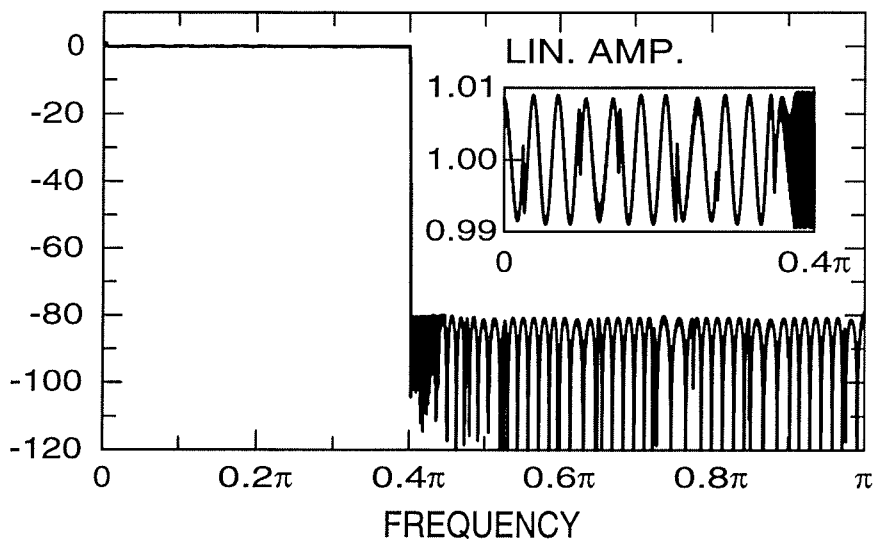
- Only 409 multipliers

Frequency responses: (a) for $F(z^L)$; solid and dashed lines in (b) for $G_1(z)$ and $G_2(z)$; (c) for overall filter
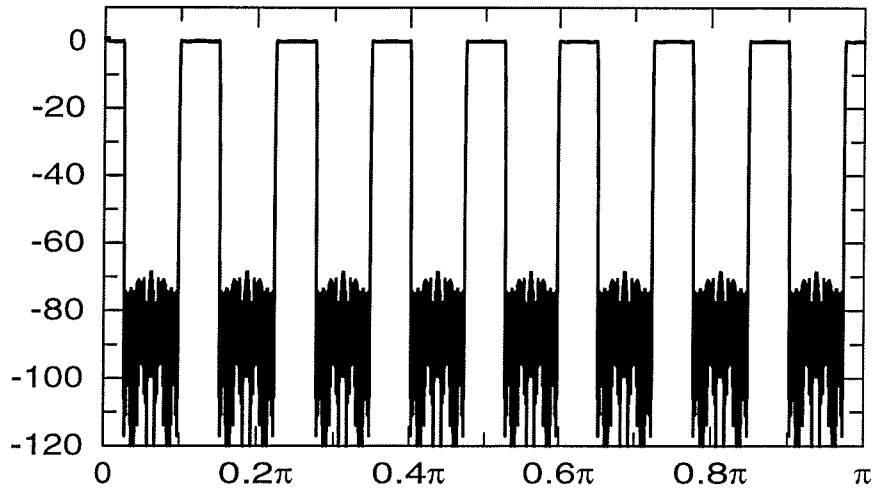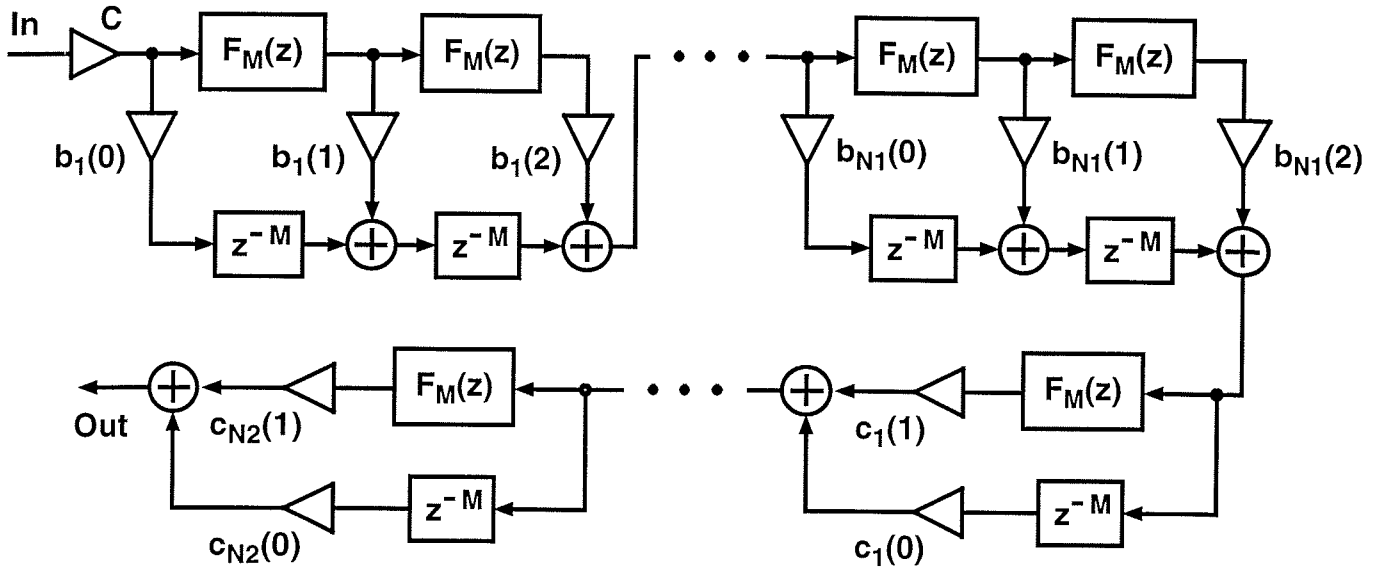
HINTS FOR VLSI IMPLEMENTATIONS

- A general multiplier is very costly and takes a huge amount of silicon area.

- To get around this problem, we can use a special network generating all the coefficient values simultaneously using a multiplier-free network (patented by VLSI Solution Oy).

  - This idea has been used in the VLSI-circuit of transparency 4.

  - In the circuit, there are altogether 4 digital FIR filters.

  - For the longest filter, the arithmetic occupies only 10 to 20 percent of the silicon area required by the data memory (delays).

  - The overall silicon area is due to the special network only one third compared to other existing circuits.

- Another alternative to avoid the use of general multiplier elements is to use special filter structures where the coefficient values are representable in the form $\pm 2^{-P_1} \pm 2^{-P_2}(\pm 2^{-P_3})$.

- A tapped cascaded interconnection of identical sub-filters is a systematic technique for synthesizing such filters.

  - The additional tap coefficients are determined such that

  1) They have the desired simple respresentation form.

  2) The passband and stopband variations for the amplitude response of the subfilter become huge.

- Due to huge ripple values, it is trivial to quantize the subfilter coefficients to a few number bits.

- The following three transparencies exemplify the use of this technique for synthesizing a very selective FIR filter.

Additional tap coefficients for the specifications: $\omega_p = 0.4\pi$, $\omega_s = 0.402\pi$, $A_p = 0.017$ dB and $A_s = 80$ dB ($\delta_p = 0.01$ and $\delta_s = 10^{-4}$)

$$b_1(2) = 2^0 - 2^{-3} + 2^{-6}$$
$$b_1(0) = 2^0 + 2^{-3} - 2^{-8}$$
$$c_1(1) = 2^{-1} + 2^{-4} + 2^{-7}$$
$$c_2(1) = 2^{-1} + 2^{-6}$$
$$c_3(1) = 2^{-1} + 2^{-3}$$
$$c_4(1) = 2^0$$
$$c_5(1) = 2^0 - 2^{-5}$$
$$c_6(1) = 2^0 - 2^{-5}$$
$$C = -2^8 + 2^4 + 2^3 - 2^0$$

$$b_1(1) = -2^1 + 2^{-4} + 2^{-8}$$

$$c_1(0) = -2^{-1} - 2^{-2}$$
$$c_2(0) = 2^{-4} - 2^{-8}$$
$$c_3(0) = 2^{-5} + 2^{-6}$$
$$c_4(0) = 2^{-7}$$
$$c_5(0) = -2^{-4}$$
$$c_6(0) = -2^{-3} + 2^{-6}$$

FIR subfilter $F_M(z)$: $\omega_p = 0.4\pi$, $\omega_s = 0.402\pi$, $A_p = 3.2$ dB and $A_s = 18$ dB ($\delta_p = 0.18$ and $\delta_s = 0.12$)
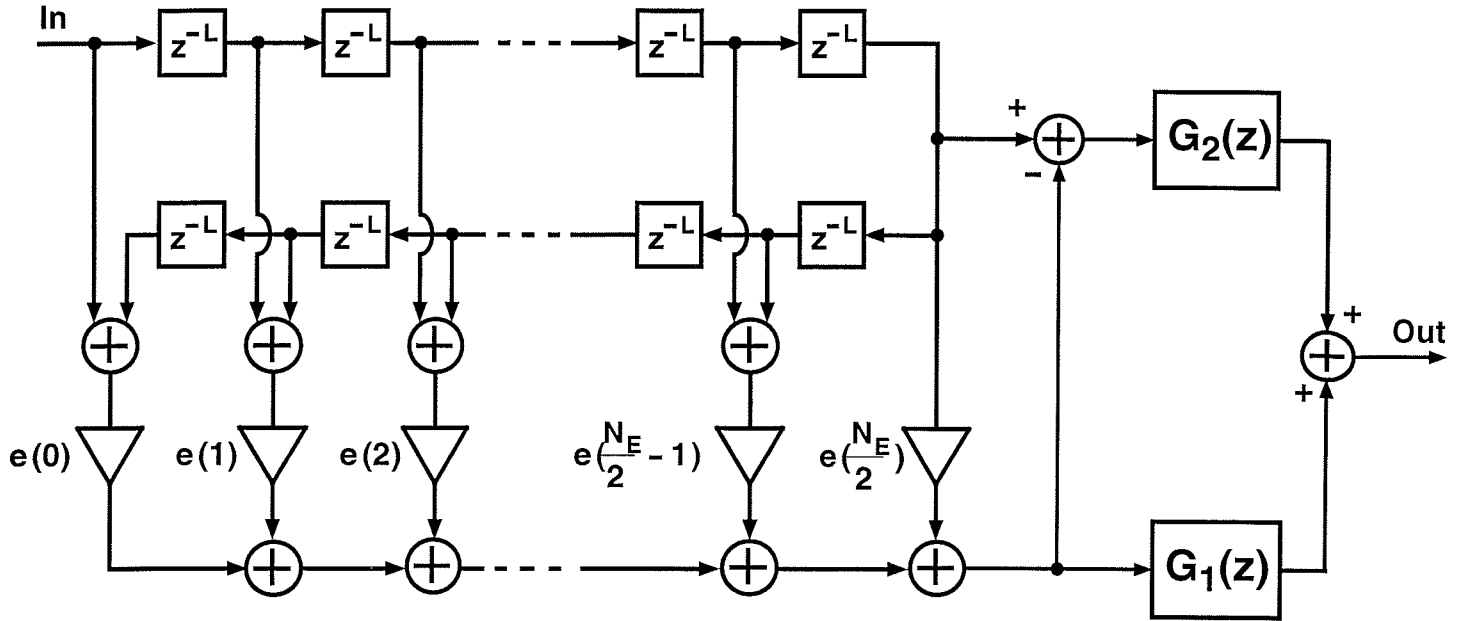

- $L = 16$, $N_E = 40$, $N_{G_1} = 22$, $N_{G_2} = 32$

$e(0) = 2 \cdot 2^{-6}$	$e(1) = -4 \cdot 2^{-6}$	$e(2) = -3 \cdot 2^{-6}$	$e(3) = -2 \cdot 2^{-6}$
$e(4) = 1 \cdot 2^{-6}$	$e(5) = 0$	$e(6) = -1 \cdot 2^{-6}$	$e(7) = -2 \cdot 2^{-6}$
$e(8) = 0$	$e(9) = 2 \cdot 2^{-6}$	$e(10) = 1 \cdot 2^{-6}$	$e(11) = -2 \cdot 2^{-6}$
$e(12) = -3 \cdot 2^{-6}$	$e(13) = 1 \cdot 2^{-6}$	$e(14) = 4 \cdot 2^{-6}$	$e(15) = 1 \cdot 2^{-6}$
$e(16) = -5 \cdot 2^{-6}$	$e(17) = -6 \cdot 2^{-6}$	$e(18) = 5 \cdot 2^{-6}$	$e(19) = 17 \cdot 2^{-6}$
$e(20) = 28 \cdot 2^{-6}$			

$g_1(0) = 2 \cdot 2^{-6}$	$g_1(1) = 3 \cdot 2^{-6}$	$g_1(2) = -1 \cdot 2^{-6}$	$g_1(3) = -2 \cdot 2^{-6}$
$g_1(4) = -1 \cdot 2^{-6}$	$g_1(5) = 3 \cdot 2^{-6}$	$g_1(6) = 2 \cdot 2^{-6}$	$g_1(7) = -4 \cdot 2^{-6}$
$g_1(8) = -5 \cdot 2^{-6}$	$g_1(9) = 4 \cdot 2^{-6}$	$g_1(10) = 20 \cdot 2^{-6}$	$g_1(11) = 28 \cdot 2^{-6}$

$g_2(0) = -3 \cdot 2^{-6}$	$g_2(1) = -1 \cdot 2^{-6}$	$g_2(2) = 1 \cdot 2^{-6}$	$g_2(3) = 2 \cdot 2^{-6}$
$g_2(4) = 1 \cdot 2^{-6}$	$g_2(5) = -1 \cdot 2^{-6}$	$g_2(6) = -2 \cdot 2^{-6}$	$g_2(7) = 0$
$g_2(8) = 3 \cdot 2^{-6}$	$g_2(9) = 3 \cdot 2^{-6}$	$g_2(10) = -1 \cdot 2^{-6}$	$g_2(11) = -5 \cdot 2^{-6}$
$g_2(12) = -2 \cdot 2^{-6}$	$g_2(13) = 7 \cdot 2^{-6}$	$g_2(14) = 19 \cdot 2^{-6}$	$g_2(15) = 24 \cdot 2^{-6}$

Amplitude responses for the subfilter and the overall filter
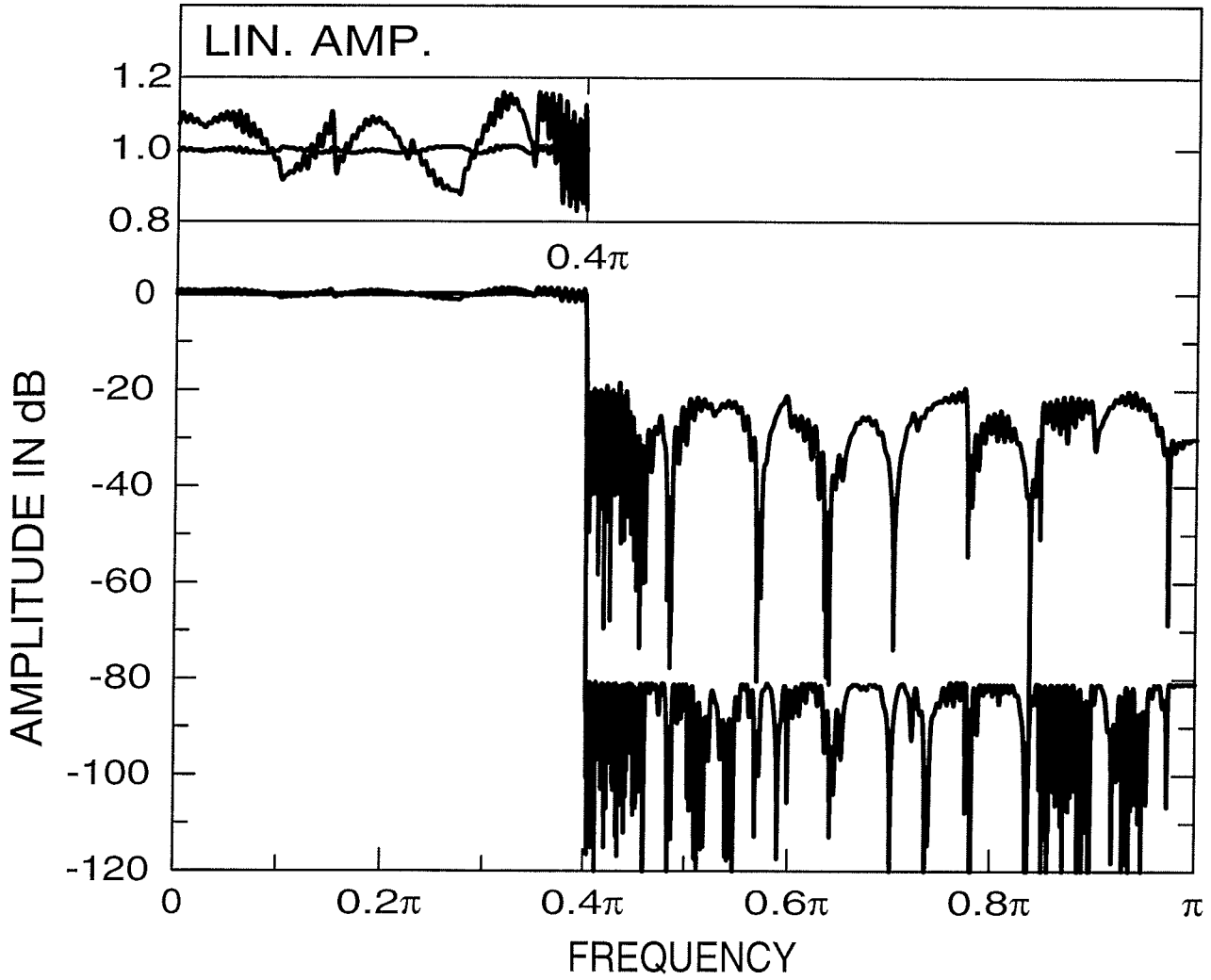
THEORY VERSUS PRACTICE

- Some reseachers are very keen on developing algorithms which are optimal and perfect in some sense even though in practice these are not necessarily needed.

- A typical example is perfect-reconstruction filter banks for subband conding.

- Because of coding, the perfect-reconstruction property is not practically taking place.

- Either the filter bank performace can be made better or the filter lengths can be drastically reduced by designing the overall system such that a small reconstruction error and very small aliasing are allowed.

- The following transparency illustrates this.

FILTER BANK WITH 32 FILTERS OF LENGTH 512

Perfect–reconstruction filter bank

Filter bank: recontsruction error < 0.0001, aliasing < 114 dB