Co-optimization of FFT and FIR in a delayless acoustic echo canceller implementation

Berkeman, Anders; Öwall, Viktor

Published in:
[Host publication title missing]

DOI:
10.1109/ISCAS.2000.857408

2000

ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland

Co-Optimization of FFT and FIR in a Delayless Acoustic Echo Canceller Implementation

Anders Berkeman and Viktor Öwall

CCCD, Dept. of Applied Electronics, Lund University, Sweden. E-mail: {abn,vikt}@tde.lth.se

Abstract

In application specific implementation of digital signal processing algorithms optimization is important for a low power solution, not only on block level but also between blocks. This paper presents a co-optimization of a fast Fourier transform and a finite impulse response filter in a silicon implementation of an acoustic echo. The optimization gain can be measured in the number of operations and memory accesses performed per second, and therefore processing power. The optimization can also be applied to other algorithms with a similar constellation of Fourier transforms and finite impulse response filters.

1 Introduction

To reduce power consumption at the architectural level, care has to be taken both on the design of the computational units as well as on the memory management [1]. Optimization of an architecture has to be performed both on each functional block as well as in the connection between them. This paper present a co-optimization of a Fourier transform connected to a Finite Impulse Response (FIR) filter in an acoustic echo canceller. The optimization makes use of a distributed arithmetic multiplier [2].

In wireless communication systems delay in the signal path is a serious obstacle and care has to be taken to reduce delay in all signal processing parts of the signal path. An application specific implementation of an acoustic echo canceller algorithm with no signal path delay sustains the real-time signal processing with affordable power consumption.

The main parts of the algorithm are shown in figure 1. At the upper left corner the far end signal $x(n)$ enters and connects to the canceller and to the loudspeaker. The signal from the microphone $y(n)$ is the other input to the canceller. Echoes and the near end talker signal $v(n)$ is added in the acoustic path between the loudspeaker and the microphone. The AFIR block contains an estimate of the impulse response of the acoustic path. The signal $x(n)$ is convolved with this estimate, and the output is subtracted from the microphone signal, optimally leaving nothing but the near end talker. The difference signal $e(n)$ is then fed back to the far end.

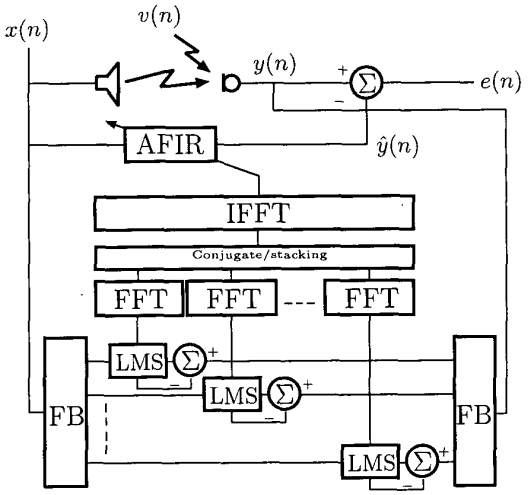

Figure 1: The subband echo cancellation algorithm, from [3]. Echo cancellation is done in the time domain.

The heart of the algorithm is a number of adaptive Least-Mean-Squares (LMS) filters that track the frequency response from speaker to microphone. Each of these LMS filters act on a small frequency band of the x and y signals which have been divided by the filterbanks (FB).

The estimated filter taps from one adaptive filter represent a part of the total impulse response in a certain frequency band. To make a fullband impulse response, these taps have to be Fourier transformed, stacked in frequency and inverse transformed. The fullband impulse response is used to filter the far end signal to simulate the effect of the sound propagating through the room. Typical sizes for the estimated impulse response is $N = 1000$ to 4000 taps at 16 kHz sample rate. The fullband impulse response is a delayed estimate of the room acoustics due to the delays in the filterbanks, adaptive filters and FFTs.

0-7803-5482-6/99/$10.00 ©2000 IEEE

2 The FFT and FIR part of the echo canceller

The filterbanks split the $x(n)$ and $y(n)$ signals into M complex subbands. Due to symmetry in the frequency plane, only $M/2+1$ bands contain unique information [3], and therefore there are $M/2+1$ LMS filters operating on these subbands.

The adaptive weights calculated in the LMS filters are combined into a fullband impulse response of length N. This is achieved by Fourier transforming the LMS weights, stacking them in frequency into a fullband frequency function, and inverse Fourier transform to get an impulse response in the time domain.

To achieve a fullband impulse response of length N, $N/2$ taps from the Fourier transform of the LMS filter taps are saved and stacked in increasing frequency order from position 0 to $N/2-1$, in what will be the fullband impulse response. Then their complex conjugates are repeated in reversed order from $N/2$ to $N-1$ sequentially. The bins are then transformed back to time-domain by an N-point inverse FFT (IFFT). The reverse repetition of the complex conjugates at the input of the IFFT generates a real-valued output of the transform. This output is the estimated room acoustic impulse response, $\hat{h}$, consisting of the taps $\hat{h}(0), \hat{h}(1), \ldots \hat{h}(N-1)$. These coefficients are used to calculate an estimate $\hat{y}(n)$ of the signal $y(n)$, according to the convolution

$$\hat{y}(n) = \sum_{k=0}^{N-1} \hat{h}(k)x(n-k) \tag{1}$$

which is performed in the Adaptive (AFIR) block of figure 1.

It is convenient to implement the FFTs and IFFT using the radix–2 decimation in frequency (DIF) or decimation in time (DIT) schemes. The radix–2 addition and subtraction butterfly has a high regularity and is therefore suitable for hardware implementation with low control requirements. Twelve butterflies of an FFT are depicted in figure 2. The last stage is always without twiddle-factor multiplication, independent of whether it is FFT or IFFT, DIF or DIT.

When an FFT or IFFT is used to calculate coefficients for an FIR filter, the last stage of the FFT is connected to the filter as shown in figure 3. In the figure, the FFT and the FIR filter is of size N. The signals $\hat{h}_{pre}$ are intermediate results inside the Fourier transform. The relationship between $\hat{h}$ and $\hat{h}_{pre}$ is

$$\begin{cases} \hat{h}(2k) &= \hat{h}_{pre}(2k) + \hat{h}_{pre}(2k+1) \\ \hat{h}(2k+1) &= \hat{h}_{pre}(2k) - \hat{h}_{pre}(2k+1) \end{cases} \tag{2}$$

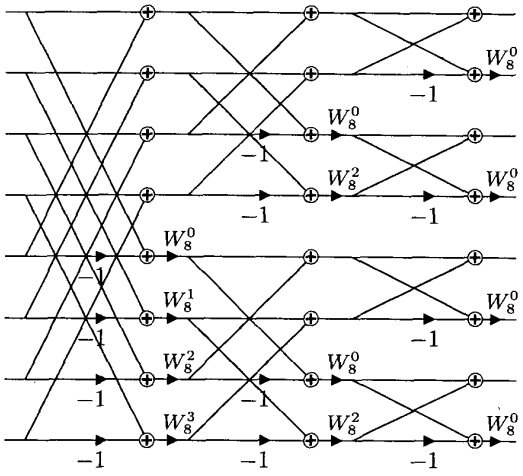

Figure 2: Eight bin decimation in frequency (DIF) FFT. W_N denotes $e^{j2\pi/N}$. Note that $W_N^0 \equiv 1, \forall N$ and thus the last stage is without twiddle multiplication.

or, the other way around

$$\begin{cases} \hat{h}_{pre}(2k) &= \frac{1}{2}(\hat{h}(2k) + \hat{h}(2k+1)) \\ \hat{h}_{pre}(2k+1) &= \frac{1}{2}(\hat{h}(2k) - \hat{h}(2k+1)). \end{cases} \tag{3}$$

Due to the stacking at the input of the IFFT as described earlier, the signals $\hat{h}(k)$ are real. Therefore the signals $\hat{h}_{pre}(k)$ must also be real.

The sum and difference of the $\hat{h}$ signals in equation (3) looks similar to the precomputation required by inner product computation using distributed arithmetic.

3 The Distributed Arithmetic Multiplier

By using distributed arithmetic [4] the order of multiplication and addition in an inner product is reversed. If precomputation is applied to one of the inputs, the number of partial inner products can be significantly reduced. When the inner product is of length two, as in

$$P = A_0x_0 + A_1x_1 \tag{4}$$

precomputation is limited to one addition and one subtraction.

If A is an L-bit fractional number in two's complement, the value of A is calculated according to

$$A = -a_0 + \sum_{\ell=1}^{L-1} a_\ell 2^{-\ell}. \tag{5}$$

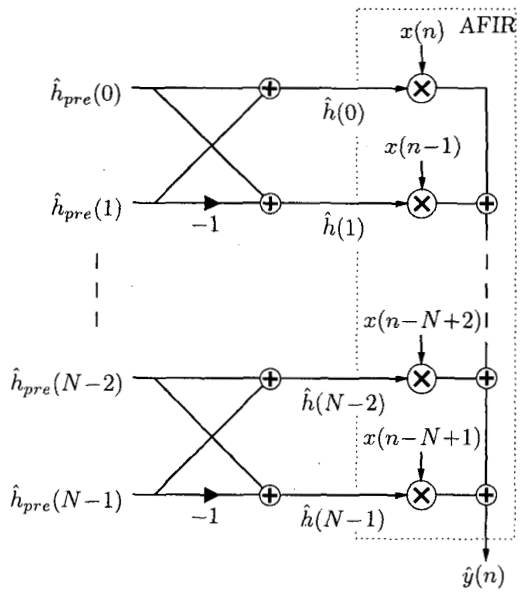

Figure 3: FIR filter preceded by sum/difference units. The FIR filter is inside the dotted box.

By using the identity

$$A = \frac{1}{2}\Big(A - (-A)\Big) \tag{6}$$

and the rule for negating a two's complement number

$$-A = \overline{A} + 2^{-(L-1)}, \tag{7}$$

equation (5) can be written as

$$A = -(a_0 - \overline{a_0})\,2^{-1} + \sum_{\ell=1}^{L-1}(a_\ell - \overline{a_\ell})\,2^{-\ell-1} - 2^{-L}. \tag{8}$$

Introduce $\alpha_0 = (\overline{a_0} - a_0)$, and for $k \neq 0$, $\alpha_k = (a_k - \overline{a_k})$, note that all $\alpha_k \in \{-1, +1\}$. Using this notation, A can be written as

$$A = \sum_{\ell=0}^{L-1}\alpha_\ell\,2^{-\ell-1} - 2^{-L}. \tag{9}$$

The relationship between a_ℓ and α_ℓ is

$$\alpha_\ell = \begin{cases} +1\ , \text{ if } a_{\ell\neq 0} = 1 \text{ or } a_0 = 0 \\ -1\ , \text{ if } a_{\ell\neq 0} = 0 \text{ or } a_0 = 1. \end{cases} \tag{10}$$

Using this encoding equation (4) can be written as

$$P = \sum_{\ell=0}^{L-1}(x_0\alpha_{0_\ell} + x_1\alpha_{1_\ell})\,2^{-\ell-1} - (x_0 + x_1)\,2^{-L}. \tag{11}$$

The expression $x_0\alpha_{0_\ell} + x_1\alpha_{1_\ell}$ is for $\ell \neq 0$ examined in the following table

α_{0_ℓ}	α_{1_ℓ}	a_{0_ℓ}	a_{1_ℓ}	$x_0\alpha_{0_\ell}+x_1\alpha_{1_\ell}$
−1	−1	0	0	$-x_\Sigma$
−1	1	0	1	x_Δ
1	−1	1	0	$-x_\Delta$
1	1	1	1	x_Σ

where

$$\begin{cases} x_\Sigma &= x_0 + x_1 \\ x_\Delta &= x_0 - x_1 \end{cases} \tag{12}$$

From the table it is clear that $p_\ell = (a_{0_\ell} \oplus a_{1_\ell})$ and $q_\ell = \overline{a_{0_\ell}}$ can be used to select x_Σ or x_Δ. Treating p_ℓ and q_ℓ as integers holding the values 0 or 1 and $\vee$ as a bitwise inclusive-or operator, equation (11) can be written as

$$P = \sum_{\ell=0}^{L-1}(-1)^{q_\ell}\left(p_\ell x_\Delta \vee \overline{p_\ell} x_\Sigma\right)2^{-\ell-1} - x_\Sigma\,2^{-L} =$$
$$\sum_{\ell=0}^{L-1}\left(q_\ell \oplus \left(p_\ell x_\Delta \vee \overline{p_\ell} x_\Sigma\right) + q_\ell\right)2^{-\ell-1} - x_\Sigma\,2^{-L}. \tag{13}$$

When evaluating the sums, q_ℓ should be replaced with $\overline{q_\ell}$ for the case $\ell = 0$, since the zeroth index has negative weight in two's complement representation. The partial inner product

$$q_\ell \oplus \left(p_\ell x_\Delta \vee \overline{p_\ell} x_\Sigma\right) + q_\ell \tag{14}$$

is realized as a multiplexer selecting $\pm x_\Sigma$ or $\pm x_\Delta$, depending on the values of p_ℓ and q_ℓ. The logic required to generate p_ℓ and q_ℓ is one inverter and one exclusive-or gate per index ℓ.

The number of partial inner product bits to be added together for the expression $A_0x_0 + A_1x_1$ assuming the A and x are 16 bits wide each is 512 when ordinary multipliers are applied. Using the distributed arithmetic approach the number is, dependent of the implementation, approximately 289, or slightly less than a 50% reduction. Addition of the partial inner products can efficiently be implemented using an adder tree [2].

4 Co-Optimization

The last stage of butterflies partitions the FIR filter into $N/2$ partial convolvers, each computing

$$\begin{aligned} \hat{y}_k(n) \;=\; & \hat{h}(2k)x(n-2k) \\ & + \hat{h}(2k+1)x(n-(2k+1)), \end{aligned} \tag{15}$$

such that the complete convolution can be written as

$$\hat{y}(n) = \sum_{k=0}^{\frac{N}{2}-1}\hat{y}_k(n) \tag{16}$$

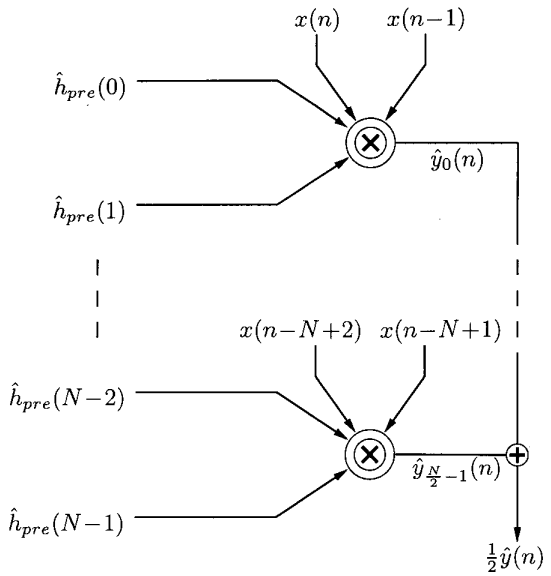

Figure 4: Same function as figure 3, but with distributed arithmetic multipliers. The multiplier with an extra circle is used as a symbol for the distributed arithmetic multiplier.

see figure 3. If a distributed arithmetic multiplier is used to calculate the partial convolutions $\hat{y}_k(n)$, its inputs has to be $x(n-2k)$ and $x(n-(2k+1))$, and the sums and differences of $\hat{h}(2k)$ and $\hat{h}(2k+1)$. These sums and differences are visible in equation (3) and, except for the factor $1/2$, equal to $\hat{h}_{pre}(2k)$ and $\hat{h}_{pre}(2k+1)$. Therefore, all butterflies in the last stage of the IFFT can be removed, and every pair of multipliers replaced by one distributed arithmetic multiplier, as depicted in figure 4. The factor $1/2$ is compensated by a logic shift of one bit to the left.

The co-optimization reduces the number of arithmetic operations, as well as the number of memory accesses. Load and store operations consume energy for memory access, address calculation and bus driving. Reduction of the number of memory accesses is crucial to reduce power in an application specific integrated circuit [1]. The effects of the optimization is shown in table 1.

Removal of the last addition and subtraction stage of the IFFT reduces the number of loads and stores with N each. As one adder, one subtracter and two multipliers are replaced by one distributed arithmetic multiplier, the number of adders and subtracters are reduced by $N/2$ each. Further, the number of products to be added in the filter is reduced from $N-1$ to $N/2-1$ and the N multipliers are replaced by $N/2$

Element	without opt.	with opt.
Butterfly adders and subtracters	N	0
FIR adders	$N-1$	$\frac{N}{2}-1$
FIR Multipliers	N	0
Distributed arithmetic multipliers	0	$\frac{N}{2}$
Memory reads before last butterfly stage	N	0
Memory writes after last butterfly stage	N	0

Table 1: Optimization results.

distributed arithmetic multipliers. The distributed arithmetic multiplier is comparable in size to one ordinary multiplier. Different schemes can be applied to reduce the complexity of an ordinary multiplier, for example modified Booth encoding [5], but the reduction of the number of memory accesses and additions is made possible due to the distributed arithmetic multiplier.

5 Conclusion

This paper presents a co-optimization of an IFFT and a FIR filter by using distributed arithmetic multipliers. The co-optimization reduces the number of memory accesses as well as the number of additions due to the removal of the last stage of the IFFT.

References

[1] Francky Catthoor, Sven Wuytack, Eddy De Greef, Florin Balasa, Lode Nachtergaele, and Arnout Vandecappelle. *"Custom Memory Management Methodology"*. Kluwer Academic Publishers, 1998.

[2] V. Öwall A. Berkeman and M. Torkelson. A low logic depth complex multiplier using distributed arithmetic. Scheduled for publication in *IEEE Journal of Solid State Circuits*.

[3] Dennis R. Morgan and James C. Thi. "Adaptive echo Cancellation for Speech Signals". In *IEEE Transaction on Signal Processing*, volume 43, No. 8, August 1995.

[4] S.G. Smith and P.B. Denyer. "Efficient Bit-Serial Complex Multiplication and Sum-Of Products Computation Using Distributed Arithmetic". In *Proc. of IEEE ICASSP*, 1986.

[5] A. D. Booth. A signed binary multiplication technique. *Q. J. Mech. Appl. Math.*, 4, 1951, pp. 236–240.