A NEW EFFICIENT IMPLEMENTATION OF TDAC SYNTHESIS FILTERBANK BASED ON RADIX-2 FFT

Anup K.C and Ajay Kumar Bangla

Audio Group, Emuzed – A Flextronics Company.
#02-03, Jeevan Bhima Nagar, HAL-3rd Stage, Bangalore-560075, India
phone: + (91)08025252223, fax: + (91)08025203257
email: anup.chathoth, ajaykumar.bangla@flextronicssoftware.com
web: www.emuzed.com

ABSTRACT

An improved implementation of the TDAC synthesis filterbank is proposed which is well suited for fixed point implementation on embedded platforms. The new implementation is based on computation of IMDCT by Radix-2 FFT. A new fast and regular structure has been derived by modifying the pre-processing, FFT and post-processing stages of the implementation. An efficient implementation of windowing for embedded platforms is also suggested.

1. INTRODUCTION

Audio players have become ubiquitous in consumer electronics ranging from portable music/media players to mobiles phones. Most of these devices have software decoders running on DSPs or DSP enhanced microcontrollers like ARM cores rather than hardware decoders. Achieving real-time software playback on such resource constrained platforms calls for extremely efficient implementations. In such players, the contribution of the synthesis filterbank to the overall complexity can be as high as 70%.

Modified discrete cosine transform/inverse modified discrete cosine transform (MDCT/IMDCT) based analysis/synthesis filterbanks have been adopted in state of the art audio compression schemes such as AC-3, AT&T PAC, SONY ATRAC, ATRAC2, Ogg Vorbis, and MPEG-1,2,4. Such filterbanks are broadly known as Time Domain Alias Cancellation (TDAC) filterbanks. The synthesis filterbank[1] in such decoders consists of IMDCT (Eq.1) followed by windowing and overlap addition (Eq.2).

$$x(n) = \frac{2}{N}\sum_{k=0}^{N/2-1} X(k)\cos\frac{\pi(2k+1)(2n+1+N/2)}{2N}; n = 0,...N-1 \quad (1)$$

$$p_m(n) = w(n+\tfrac{N}{2})x_{m-1}(n+\tfrac{N}{2}) + w(n)x_m(n); n = 0,....\tfrac{N}{2}-1 \quad (2)$$

where $w(n)$ is the synthesis window function, $X(K)$ are the m^{th} frame MDCT coefficients and $p_m(n)$ is the reconstructed m^{th} frame. For simplicity, the factor 1/N in Eq.1 is merged into the window function $w(n)$.

Several algorithms have been suggested for their fast implementation. The fastest methods either factorize the IMDCT computation to DCT[1,2,3,4] or FFT computed using Split Radix FFT (SRFFT)[5,6]. DCT and SRFFT based solutions are not suitable for embedded cores because of their irregular structure which results in complex data addressing and code size that increases rapidly with the order of the transform which can be as high as 8192[7]. Due to these problems most embedded implementations are based on IMDCT by Radix-2 FFT which is explained in Section 2. Based on this implementation, the derivation of a more efficient structure in explained in Section 3. Section 4 summarizes the main results of this work.

2. TRADITIONAL FFT BASED ALGORITHM

The following steps illustrate the fast implementation of TDAC synthesis filterbank (Eq.1 and Eq.2) using Generalized DFT (GDFT)[8].

Step 1: Input combination

$$Y(k) = X(\tfrac{N}{2}-2k-1) + jX(2k); k = 0,...\tfrac{N}{4}-1 \quad (3)$$

Step 2: GDFT

$$y(n) = \sum_{k=0}^{\frac{N}{4}-1} Y(k)e^{-j\frac{2\pi}{N}(2n+\frac{1}{2})(2k+\frac{1}{2})}; n = 0,...\tfrac{N}{4}-1 \quad (4)$$

Step 3: Windowing and de-interleaving:

$$\begin{aligned}
&x(2n) = -y_i(\tfrac{N}{8}+n)w(2n)\\
&x(2n+1) = y_r(\tfrac{N}{8}-n-1)w(2n+1)\\
&x(\tfrac{N}{4}+2n) = -y_r(n)w(\tfrac{N}{4}+2n)\\
&x(\tfrac{N}{4}+2n+1) = y_i(\tfrac{N}{4}-n-1)w(\tfrac{N}{4}+2n+1)\\
&x(\tfrac{N}{2}+2n) = -y_r(\tfrac{N}{8}+n)w(\tfrac{N}{2}-2n-1)\\
&x(\tfrac{N}{2}+2n+1) = y_i(\tfrac{N}{8}-n-1)w(\tfrac{N}{2}-2n-2)\\
&x(\tfrac{3N}{4}+2n) = y_i(n)w(\tfrac{N}{4}-2n-1)\\
&x(\tfrac{3N}{4}+2n+1) = -y_r(\tfrac{N}{4}-n-1)w(\tfrac{N}{4}-2n-2)\\
&n = 0,....\tfrac{N}{8}-1
\end{aligned} \quad (5)$$

Step 4: Overlap and Add

$$\begin{aligned}
&p(n) = 2\left[x(n) + delay(n)\right]\\
&delay(n) = x(\tfrac{N}{2}+n)\\
&n = 0,...\tfrac{N}{2}-1
\end{aligned} \quad (6)$$

In the traditional implementation, the GDFT (Eq.4) is modified as follows:

$$y(n)=\sum_{k=0}^{\frac{N}{4}-1}Y(k)*e^{-j\frac{2\pi}{N/4}nk}e^{-j\frac{2\pi}{N}(n+\frac{1}{8})}e^{-j\frac{2\pi}{N}(k+\frac{1}{8})} \quad (7)$$

Hence, Step 2 is computed as follows

Step 2.1: Pre-processing:

$$Z(k)=Y(k)*W_N^{k+1/8};k=0,....\tfrac{N}{4}-1 \quad (8)$$

Step 2.2: FFT:

$$z(n)=\sum_{k=0}^{\frac{N}{4}-1}Z(k)*W_{N/4}^{kn};n=0,....\tfrac{N}{4}-1 \quad (9)$$

Step 2.3: Post-processing:

$$y(n)=z(n)*W_N^{n+1/8};n=0,....\tfrac{N}{4}-1 \quad (10)$$

where $W_N^n=e^{-j\frac{2\pi}{N}n}$

The computational complexity for this method is given in Tables 1 and 2. The contribution of pre and post processing to the total computations in IMDCT varies from 34% to 79% depending its order. Duhamel[5] has shown that steps 2.1, 2.2 and 2.3 can actually be performed inside the SRFFT. As explained in previous section, SRFFT is not a viable option for embedded platforms. Šević[6] has proposed the merging of the windowing of Step 3 with the post-processing. However this scheme is only applicable to special windows.

3. PROPOSED ALGORITHM

The GDFT (Eq.4) can be alternately split as

$$y(n)=\sum_{k=0}^{\frac{N}{4}-1}Y(k)e^{-j\frac{2\pi}{N/4}nk}e^{-j\frac{\pi}{N}(2n+\frac{1}{2})}e^{-j\frac{2\pi}{N}k} \quad (11)$$

Hence, Step 2 is now implemented in the following sub steps:

Step 2.1: Pre-processing:

$$Z(k)=Y(k)W_N^k;k=0,....\tfrac{N}{4}-1 \quad (12)$$

Step 2.2: FFT:

$$z(n)=\sum_{k=0}^{\frac{N}{4}-1}Z(k)W_{N/4}^{kn};n=0,....\tfrac{N}{4}-1 \quad (13)$$

Step 2.3: Post-processing:

$$y(n)=z(n)W_N^{n+1/4};n=0,....\tfrac{N}{4}-1 \quad (14)$$

Novel modifications are suggested in the following subsections which results in a new structure with fewer computational requirements.

3.1. Pre-processing & FFT

The first stage of N/4 point FFT has been merged with the pre-processing stage to reduce the overall computations. Using the radix-2 decimation-in-frequency scheme[9], Step 2.2 (Eq.13) can be split as

$$\begin{aligned}z(2n)&=\sum_{k=0}^{\frac{N}{8}-1}G_1(k)W_{N/8}^{kn}\\ z(2n+1)&=\sum_{k=0}^{\frac{N}{8}-1}G_2(k)W_{N/8}^{kn}\\ n&=0,....\tfrac{N}{8}-1\end{aligned} \quad (15)$$

where

$$\begin{aligned}G_1(k)&=Z(k)+Z(k+\tfrac{N}{8})\\ G_2(k)&=\left[Z(k)-Z(k+\tfrac{N}{8})\right]W_{N/4}^k\\ k&=0,....\tfrac{N}{8}-1\end{aligned} \quad (16)$$

Combining Step 2.1 with Eq.16

$$\begin{aligned}G_1(k)&=Y(k)W_N^k+Y(k+\tfrac{N}{8})W_N^{k+\frac{N}{8}}\\ G_2(k)&=\left[Y(k)W_N^k-Y(k+\tfrac{N}{8})W_N^{k+\frac{N}{8}}\right]W_{N/4}^k\end{aligned} \quad (17)$$

The new pre-processing factors W_N^k possesses the following property.

$$W_N^{k+N/8}=e^{-j\frac{\pi}{4}}*W_N^k;k=0,...N/8-1 \quad (18)$$

Using this property, Eq.17 can be written as

$$\begin{aligned}G_1(k)&=[Y(k)+e^{-j\frac{\pi}{4}}*Y(k+\tfrac{N}{8})]W_N^k\\ G_2(k)&=[Y(k)-e^{-j\frac{\pi}{4}}*Y(k+\tfrac{N}{8})]W_N^{5k}\end{aligned} \quad (19)$$

Eq.19 and Eq.15 become the new Steps 2.1 (Pre-processing) and 2.2 (FFT Stage) respectively. The resulting computation structure is shown in Fig.1.

The new pre-processing stage now has N/8 multiplications with $e^{-j\pi/4}$, which can be implemented using only 2 real mults and 2 adds, on input side and N/4-2 complex multiplications on the output side. Out of the N/4-2 constants required by this stage, N/40-1 are shared between the upper (W_N^k) and lower (W_N^{5k}) halves. Furthermore N/64-1 and N/320-1 constants are shared by the upper and lower halves, respectively, with N/8 point FFT twiddle factors. These redundancies can be exploited to reduce the ROM requirements at the cost of regular memory access in pre-processing stage.

3.2. Post-processing

The post-processing stage has N/4 complex multiplications. We propose to reduce the computations in this stage by modifying the windows coefficients. The post-processing rotation factors are given by

$$W_N^{n+\frac{1}{4}}=\cos(\tfrac{2\pi}{N}\overline{n+\tfrac{1}{4}})-jSin(\tfrac{2\pi}{N}\overline{n+\tfrac{1}{4}});n=0,...\tfrac{N}{4}-1$$

These rotation factors can be rearranged as

$$W_N^{n+\frac{1}{4}}=\begin{cases}\left[1-j\tan\left(\frac{2\pi}{N}\overline{n+\frac{1}{4}}\right)\right]\cos\left(\frac{2\pi}{N}\overline{n+\frac{1}{4}}\right), n=0,...\frac{N}{8}-1\\ \left[\cot\left(\frac{2\pi}{N}\overline{n+\frac{1}{4}}\right)-j\right]\sin\left(\frac{2\pi}{N}\overline{n+\frac{1}{4}}\right), n=\frac{N}{8},....\frac{N}{4}-1\end{cases}$$

The cosine and sine values can be easily merged with the window coefficients without disturbing the symmetry.

The new rotation factors are well suited for fixed point implementation since the tan and cot values lie between 0 and 1 for the given range. Although complex, only two real multiplications are required for complex multiplications involving them. The ROM requirement for storing the post-rotation factors is halved since only the real or imaginary parts have to be stored.

3.3. Windowing and overlap addition

The Windowing (Step 3) and Overlap addition (Step 4) can be combined as

$$p(2n) = 2\left[-y_i(\tfrac{N}{8}+n)w(2n) - y'_r(\tfrac{N}{8}+n)w(\tfrac{N}{2}-2n-1)\right]$$
$$p(2n+1) = 2\left[y_r(\tfrac{N}{8}-n-1)w(2n+1) + y'_i(\tfrac{N}{8}-n-1)w(\tfrac{N}{2}-2n-2)\right]$$
$$p(\tfrac{N}{4}+2n) = 2\left[-y_r(n)w(\tfrac{N}{4}+2n) + y'_i(n)w(\tfrac{N}{4}-2n-1)\right]$$
$$p(\tfrac{N}{4}+2n+1) = 2\left[y_i(\tfrac{N}{4}-n-1)w(\tfrac{N}{4}+2n+1) - y'_r(\tfrac{N}{4}-n-1)w(\tfrac{N}{4}-2n-2)\right]$$
$$n = 0,\ldots\tfrac{N}{8}-1$$

where y' corresponds to the previous frame. This can be re-written as

$$p(2n) = 2\left[-y_i(\tfrac{N}{8}+n)w(2n) - y'_r(\tfrac{N}{8}+n)w(\tfrac{N}{2}-2n-1)\right]$$
$$p(2n+1) = 2\left[y_r(\tfrac{N}{8}-n-1)w(2n+1) + y'_i(\tfrac{N}{8}-n-1)w(\tfrac{N}{2}-2n-2)\right]$$
$$p(\tfrac{N}{2}-2n-2) = 2\left[-y_r(\tfrac{N}{8}-n-1)w(\tfrac{N}{2}-2n-2) + y'_i(\tfrac{N}{8}-n-1)w(2n+1)\right]$$
$$p(\tfrac{N}{2}-2n-1) = 2\left[y_i(\tfrac{N}{8}+n)w(\tfrac{N}{2}-2n-1) - y'_r(\tfrac{N}{8}+n)w(2n)\right]$$
$$n = 0,\ldots\tfrac{N}{8}-1$$

These real computations can be converted to complex multiplications as shown

$$p(2n) + jp(\tfrac{N}{2}-2n-1) = 2\left[-y_i(\tfrac{N}{8}+n) - jy'_r(\tfrac{N}{8}+n)\right]\left[w(2n) - jw(\tfrac{N}{2}-2n-1)\right]$$
$$p(2n+1) + jp(\tfrac{N}{2}-2n-2) = 2\left[y_r(\tfrac{N}{8}-n-1) + jy'_i(\tfrac{N}{8}-n-1)\right]\left[w(2n+1) - jw(\tfrac{N}{2}-2n-2)\right]$$
$$n = 0,\ldots\tfrac{N}{8}-1$$

Now it is possible to reduce the multiplications at the cost of additions. This technique is beneficial on platforms like ARM9T cores where the multiplier takes more than three cycles.

4. RESULTS

The complex multiplication can be implemented using 4 mults + 2 adds or 3 mults + 5 adds or 3 mults + 3 adds. The 3 mults + 3 adds scheme increases the ROM requirements for complex constants ($c+jd$) by 33% as c, $c+d$ and $c-d$ have to be stored. It also means that there is an extra load. For most embedded platforms, load is multicycle while add is single cycle. Thus, while this scheme is generally used to illustrate the results, it is not very suitable for an embedded scenario.

The computational complexities are tabulated in Table 1 for 4 mults + 2 adds and 3 mults + 5 adds schemes. The percentage reduction in IMDCT computation for commonly used orders is illustrated in Fig. 2. The method explained in section 3.3 provides a flexibility to use 3 mults + 5 or 3 adds scheme in windowing and overlap addition stage.

The ROM requirements of the proposed method are compared with the traditional method in Table 2. The redundancies in the pre-processing constant can be exploited to reduce the additional memory requirements to 10%.

5. CONCLUSION

The proposed structure and methods result in significant reduction in computations with slight increase in the ROM requirements. Also the advantage of the traditional method viz. regular structure is preserved.

REFERENCES

[1] D.Y. Chan, J.F. Yang and S.Y. Chen, "Regular implementation of algorithms of time domain aliasing cancellation," *IEE Proc. Vis. Image Signal Process.,* Vol. 143, Issue 6, pp. 387-392, Dec. 1996.

[2] Mu-Huo Cheng and Yu-Hsin Hsu, "Fast IMDCT and MDCT Algorithms – A Matrix Approach," *IEEE Trans. Signal Processing,* Vol. 51, pp. 221-229, Jan. 2003.

[3] Y. H. Fan, V. K. Madisetti, and R. M. Mersereau, "On fast algorithms for computing the inverse modified discrete cosine transform," *IEEE Signal Process. Lett.*, vol. 6, pp. 61–64, Mar. 1999.

[4] Vladmimir Britanak and K.R. Rao, "An Efficient Implementation of the Forward and Inverse MDCT in MPEG Audio Coding," *IEEE Signal Process. Lett.*, vol. 8, pp. 48–51, Feb. 2001.

[5] P. Duhamel, Y. Mahieux, and J. P. Petit, "A fast algorithm for the implementation of filter banks based on time domain aliasing cancellation," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, pp. 2209–2212, May 1991.

[6] D. Šević and M. Popović, "A new efficient implementation of the oddly stacked Princen–Bradley filter bank," *IEEE Signal Processing Lett.,* vol.1, pp. 166–168, Nov. 1994.

[7] "Ogg Vorbis I specification", http://xiph.org/vorbis/doc/Vorbis_I_spec.html.

[8] *Digital Audio Compression (AC-3) Standard*, Audio Specialist Group T3/S7, Dec. 1995.

[9] John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing,* Prentice-Hall, India, 2002.

	4 Mults + 2 Adds scheme				3 Mults + 5 Adds scheme			
	Traditional Method		Proposed Method		Traditional Method		Proposed Method	
	Mults	Adds	Mults	Adds	Mults	Adds	Mults	Adds
Pre processing	N	N/2	5N/4-8	5N/4-4	3N/4	5N/4	3N/2-10	3N/2-6
Post-processing	N	N/2	N/2	N/2	3N/4	5N/4	N/2	N/2
FFT[i]	(N/2)v[ii]-13N/8)+10	3N/4v-(6N/8)+4	(N/2)v-(17N/8)+14	3N/4v-(3N/2)+6	(3N/8)v-(19N/16)+7	9N/8v-(33N/16)+13	(3N/8)v-(33N/16)+14	(9/8)Nv-(43N/16)+14
Window	2N	N	2N	N	2N	N	2N or 3N/2	N or 5N/2

Table 1: Complexity of the traditional and proposed methods.

	Traditional Method	Proposed Method	% Increase
Pre-processing	N/2	N/2 or 17N/40	50% or 35%
Post-processing		N/4	
FFT	N/8	N/16	-50%
Windowing	N/2	N/2	0%
Total	9N/8	21N/16 or 99N/80	16.67% or 10%

Table 2: ROM Requirements of traditional and proposed methods

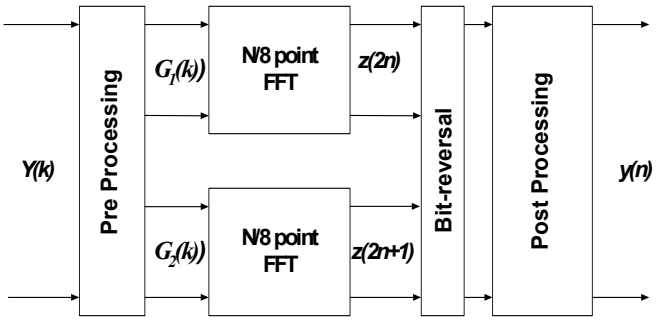
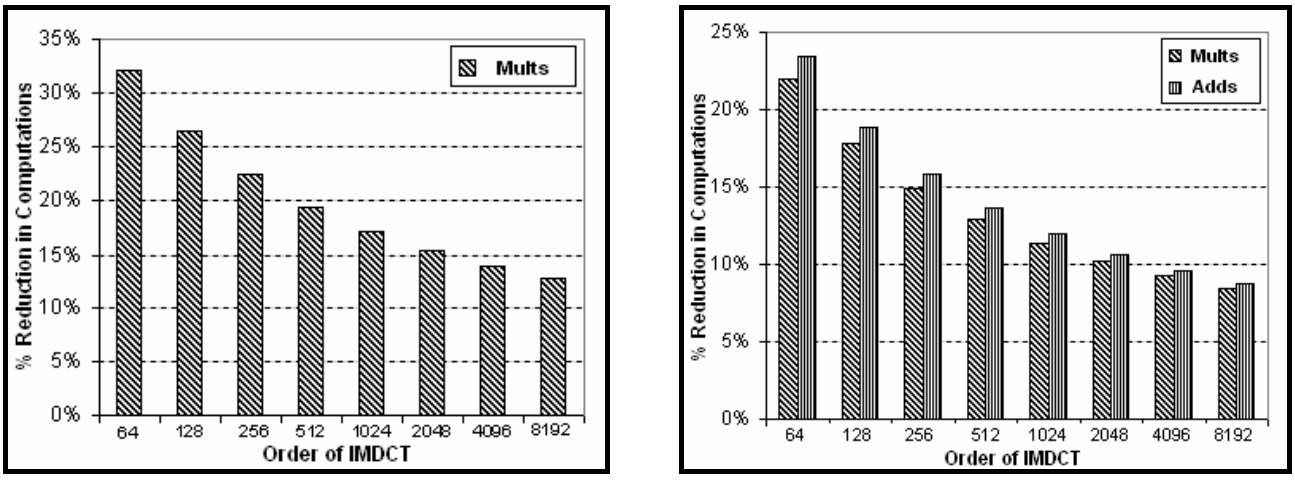


Fig.1: Proposed structure for synthesis filterbank computation



(a[iii]) (b)

Fig.2: Percentage reductions in additions and multiplications for (a) 4 mults + 2 adds scheme and (b) 3 mults + 5 adds scheme.

[i] while computing the computations for FFT, gains due to the properties of twiddle factors 1, -j and $e^{-j\frac{\pi}{4}}$ are taken into account.
[ii] v = $\log_2(N/4)$ where N is the order of IMDCT.
[iii] The number of additions for the proposed method is two less than that for traditional method for all orders.