The Feedforward Short-Time Fourier Transform

Mario Garrido Gálvez

Journal Article

Mario Garrido Gálvez , The Feedforward Short-Time Fourier Transform, IEEE Transactions on Circuits and Systems - II - Express Briefs, 2016. 63(9), pp.868-872.
http://dx.doi.org/10.1109/TCSII.2016.2534838
Postprint available at: Linköping University Electronic Press

http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-131671

The Feedforward Short-Time Fourier Transform

Mario Garrido, *Member, IEEE*

***Abstract*—This paper presents the feedforward short-time Fourier transform (STFT). This new approach is based on reusing the calculations of the STFT at consecutive time instants. This leads to significant savings in hardware components with respect to FFT-based STFTs. Furthermore, the feedforward STFT does not have the accumulative error of iterative STFT approaches. As a result, the proposed feedforward STFT presents an excellent trade-off between hardware utilization and performance.**

***Index Terms*—STFT, FFT, feedforward, pipelined architecture**

I. Introduction

The short-time Fourier transform (STFT) is a linear transform used to calculate the evolution of a signal over time, offering a trade-off between spectral and temporal resolutions. Whereas the fast Fourier transform (FFT) is adequate for the study of stationary signals, i.e., those signals whose parameters do not vary over time such as sinusoids, the STFT is adequate for non-stationary signals. In this case, parameters such as amplitude, frequency and phase can vary over time.

The STFT is mainly used in spectral analysis [1]–[3], being a key element in many applications such as medical applications [4]–[6], digital receivers [7]–[9], and musical and audio signal analysis [10]–[12].

Nowadays there are two main approaches to calculate the STFT in hardware. The first one consists of using several FFT modules in parallel [1]. Each of these modules calculates all the STFT frequencies at a certain time instant. The second approach obtains the values for each frequency independently following an iterative fashion [13], [14].

Although both approaches are useful for calculating the STFT, they have important drawbacks. On the one hand, the FFT-based STFT has a high cost in terms of hardware, as it needs the implementation of multiple FFTs in parallel. On the other hand, the iterative STFT generates an accumulative error that increases at each iteration.

This paper presents the feedforward STFT. This new approach has the advantage that it requires significantly less hardware than the FFT-based STFT and, at the same time, it does not generate accumulative errors like the iterative STFT.

The paper is organized as follows. Section II reviews previous approaches for the STFT. Section III explains the basic principle in which the feedforward STFT is based. Section IV presents the feedforward STFT algorithm. Section V shows the proposed feedforward STFT architecture. Section VI presents the STFT for real-valued inputs. Finally, Section VII summarizes the main conclusions of the paper.

M. Garrido is with the Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, e-mail: mario.garrido.galvez@liu.se

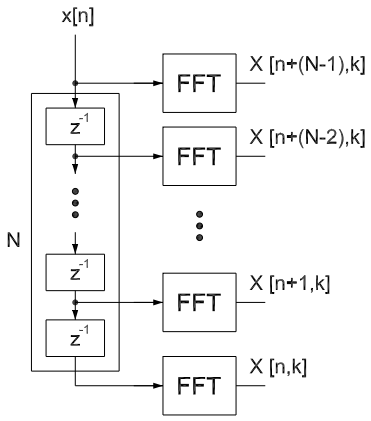

Fig. 1. FFT-based STFT.

II. STFT Review

In digital systems the STFT of a discrete signal $x[n]$ is defined as

$$\text{STFT}_x[n,k] = X[n,k] = \sum_{m=n}^{n+(N-1)} x[m]e^{-j\frac{2\pi k}{N}m}, \quad (1)$$

where $k = 0, 1 \ldots, N-1$. In the equation, time and frequency are represented by the discrete variables n and k, respectively.

The STFT at a certain time n corresponds to the FFT of the sequence from samples $x[n]$ to $x[n+(N-1)]$. Thus, one approach to calculate the STFT is to use N FFT processors in parallel [1], as shown in Fig. 1. In this case, all the processors start to calculate the FFT at the same time. Due to the delay of the buffer, the FFTs start with different samples and, thus, calculate the FFT at different time instants.

Considering that each FFT is implemented by a single-path delay feedback (SDF) FFT [1], each FFT processor has $2\log_2 N$ adders, $\log_2 N$ multipliers and a total memory size of N. Therefore, the entire STFT has $2N\log_2 N$ adders, $N\log_2 N$ multipliers and a memory size of N^2.

Another approach consists of calculating the STFT for each frequency independently [13], [14]. This is done by the iterative structure in Fig. 2, based on the equation:

$$\text{STFT}_x[n,k] = e^{j\frac{2\pi}{N}k}(\text{STFT}_x[n-1,k]+x[n-1+N]-x[n-1]) \quad (2)$$

Therefore, at each time instant, each frequency is updated with the incoming value. In this case the number of adders and multipliers is N, and the total memory size is $2N$.

The hardware cost is smaller in the iterative approach compared to the FFT-based one. However, the iterative approach has the drawback that the quantization error accumulates at each iteration [14].

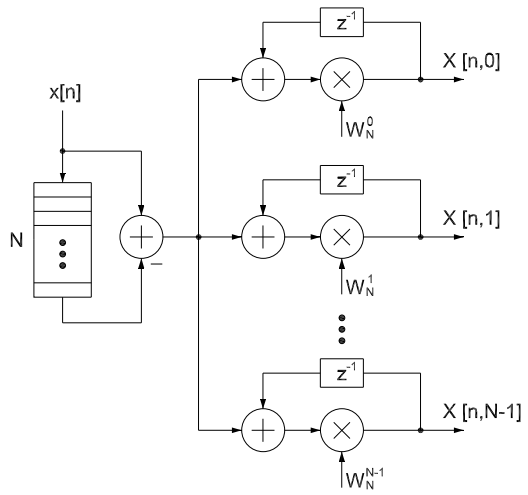

Fig. 2. Iterative STFT.

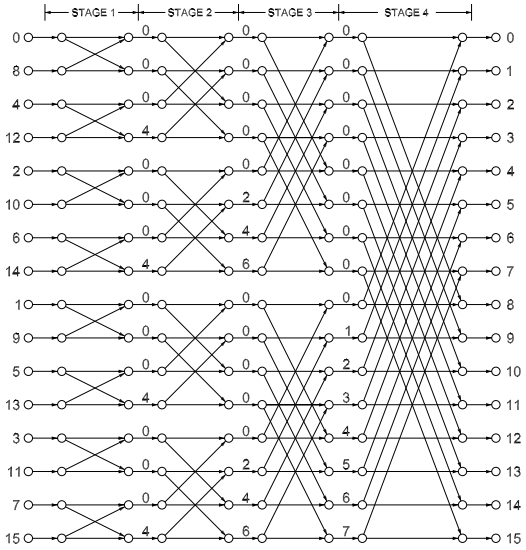

Fig. 3. Flow graph of a 16-point radix-2 DIT FFT.

III. Basic Principle of the Feedforward STFT

The feedforward STFT is based on the radix-2 decimation in time (DIT) FFT. The DIT algorithm has the property that the STFT can reuse operations of consecutive FFTs as shown next.

The flow graph of a radix-2 DIT FFT for $N = 16$ points is shown in Fig. 3. The numbers at the input represent the index of the input sequence, $x[n]$, whereas those at the output are the frequencies, k, of the output signal $X[k]$. In the flow graph, the input values are depicted in bit-reversed order and the output frequencies are in natural order.

The flow graph consists of a series of n stages, $s \in \{1 \dots n\}$, where additions, subtractions and complex multiplications are calculated. Additions and subtractions come in pairs, forming a structure called butterfly. The flow graph in Fig. 3 assumes that the lower edge of each butterfly is always multiplied by -1. These -1 are usually not depicted in order to simplify the graphs.

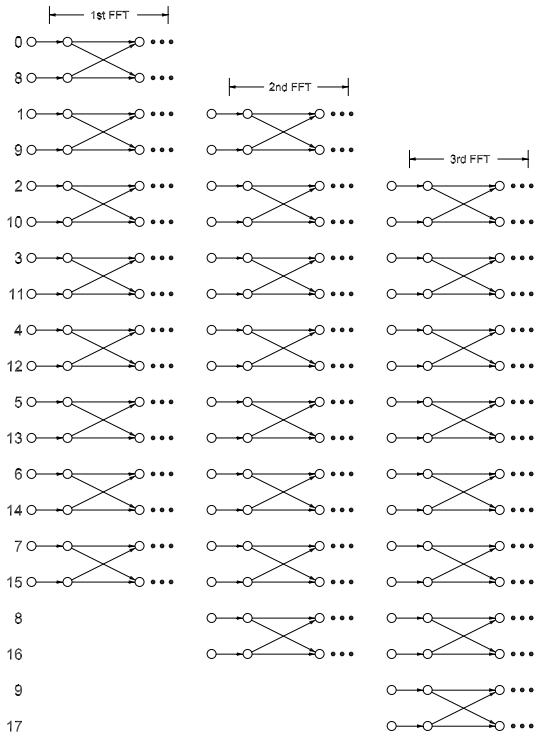

Fig. 4. Basic Principle of the proposed STFT.

The multiplications are represented by the numbers between the stages. Each number, ϕ, in between the stages indicates a multiplication by the twiddle factor:

$$W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi} \tag{3}$$

Let us imagine that we are calculating the STFT and, therefore, we have to calculate consecutive FFTs. Fig. 4 shows this case. The first FFT is calculated on samples $x[0]$ to $x[15]$, the second FFT is calculated on samples $x[1]$ to $x[16]$, and the third FFT is calculated on samples $x[2]$ to $x[17]$. In each consecutive FFT a new sample arrives at the input and another sample is discarded. The second FFT discards sample $x[0]$ and incorporates sample $x[16]$. Fig. 4 also shows the butterflies of the first stage of the algorithm. It can be observed that all the butterflies of the first stage can be reused for the next FFT except one of them. This means that there are many operations that are repeated among consecutive FFTs and only need to be calculated once. Therefore, if we store previous results, we only need to calculate one butterfly for each incoming sample in order to calculate the STFT. If we generalize this, for the second stage we have to calculate 2 butterflies for each incoming sample and for any stage $s \in \{1 \dots n\}$, we have to calculate 2^{s-1} butterflies.

Regarding multiplications, the DIT algorithm has the particularity that multiplications can be reused for the STFT in the same way as the butterflies. Indeed, in Fig. 3 the flow graph until stage 2 is formed by four equal parts and the flow graph until stage 3 is formed by two equal parts. This shows the large number of operations that are repeated among consecutive STFTs.

IV. Proposed Feedforward STFT Algorithm

```
for i = 1 : length(input),

    x00 = input(i);                    −− Input Sample

    m1 = mod(i, N/2);                  −− Stage 1
    x10 = b10(m1) + x00;
    x11 = b10(m1) − x00;
    b10(m1) = x00;

    m2 = mod(i, N/4);                  −− Stage 2
    xr11 = (−j) ∗ x11;
    x20 = b20(m2) + x10;
    x21 = b20(m2) − x10;
    x22 = b21(m2) + xr11;
⋆   x23 = b21(m2) − xr11;
    b20(m2) = x10;
    b21(m2) = x11;

    m3 = mod(i, N/8);                  −− Stage 3
    xr21 = (−j) ∗ x21;
    xr22 = (0.7071 − 0.7071j) ∗ x22;
⋆   xr23 = (−0.7071 − 0.7071j) ∗ x23;
    x30 = b30(m3) + x20;                                   (4)
    x31 = b30(m3) − x20;
    x32 = b31(m3) + xr21;
⋆   x33 = b31(m3) − xr21;
    x34 = b32(m3) + xr22;
    x35 = b32(m3) − xr22;
⋆   x36 = b33(m3) + xr23;
⋆   x37 = b33(m3) − xr23;
    b30(m3) = x20;
    b31(m3) = x21;
    b32(m3) = x22;
⋆   b33(m3) = x23;

    STFT(i, 0) = x30;                  −− Outputs
    STFT(i, 1) = x34;
    STFT(i, 2) = x32;
⋆   STFT(i, 3) = x36;
    STFT(i, 4) = x31;
    STFT(i, 5) = x35;
⋆   STFT(i, 6) = x33;
⋆   STFT(i, 7) = x37;

end
```

The pseudocode of the proposed feedforward STFT algorithm is shown in equation (4) for the case of $N = 8$ points. At each iteration the algorithm takes one input value $x[i]$ from the input signal $x[n]$ and provides 8 output frequencies corresponding to time i. They are provided in STFT(i, k), $k = 0, \ldots, N-1$.

The proposed algorithm is separated in stages for an easier explanation. First, the input value $x[i]$ is saved in the variable $x00$. The first stage consists of a memory $b10$ and the two variables $x10$ and $x11$. The memory implements a circular buffer of size $N/2$, addressed by the variable $m1$. Thus, when all the memory has been filled, it starts to be filled from the beginning. The two variables $x10$ and $x11$ are the output of the butterfly of the first stage, whose inputs are the value in the buffer and the input signal.

Note that the first stage only calculates a butterfly for each incoming input sample. This agrees with Fig. 4, where only the calculation of one butterfly is needed at the first stage. Likewise, two samples that are operated in the butterfly differ by $N/2$ samples. This is the reason why the buffer has a length of $N/2$.

The second stage includes a buffer of length $N/4$ and four variables $x20$ to $x23$ that save the results of the butterflies. Previous to this, the input $x11$ is multiplied by the twiddle factor $W_8^2 = -j$. After the butterflies are calculated, the buffers $b20$ and $b21$ are updated.

The third stage is similar to the second one. It only differs in the twiddle factors that are calculated and the number of butterflies.

After the third stage, the output of the STFT is obtained and stored in the variable 'STFT'. For this purpose, the outputs are assigned according to the bit reversal algorithm [15].

In a general case for any N, the pseudocode of the feedforward STFT algorithm is shown in equation (5).

```
for i = 1 : length(input),
   for s = 1 : n,
      m = mod(i, N/2^s);
      for k = 0 : 2 : 2^s − 1,
         xr = x_{s−1,k/2} ∗ TW(s, k, n);
         x_{s,k} = b_{s,k/2}(m) + xr;
         x_{s,k+1} = b_{s,k/2}(m) − xr;                    (5)
      end
      for k = 0 : 2^{s−1} − 1,
         b_{s,k}(m) = x_{s−1,k};
      end
   end
end
```

The number of additions of the proposed algorithm is $2N - 2$ and the number of multiplications $N - 1$. This can be compared to the use of an FFT-based STFT in software: The FFT-based STFT requires to calculate one FFT for each incoming sample. In software this means $N \log_2 N$ additions and $N/2 \log_2 N$ multiplications per incoming sample. Thus, the proposed algorithm has less additions and multiplications.

Fig. 6 compares the proposed algorithm to the FFT-based STFT for the case of an 8-point STFT , using an Intel CORE i3 and MATLAB R2009b. The simulation runs 100 iterations of each algorithm. Fig. 6 shows the time of each iteration. In the proposed algorithm the average time per iteration is 13.29 μs whereas the average time for the FFT-based approach is 15.69 μs. This represents savings of 18 % for the proposed algorithm compared to using an FFT-based approach.

V. Proposed Feedfoward STFT Architecture

The hardware implementation of the feedforward STFT is shown in Fig. 5 for the case of $N = 16$. It consists of four

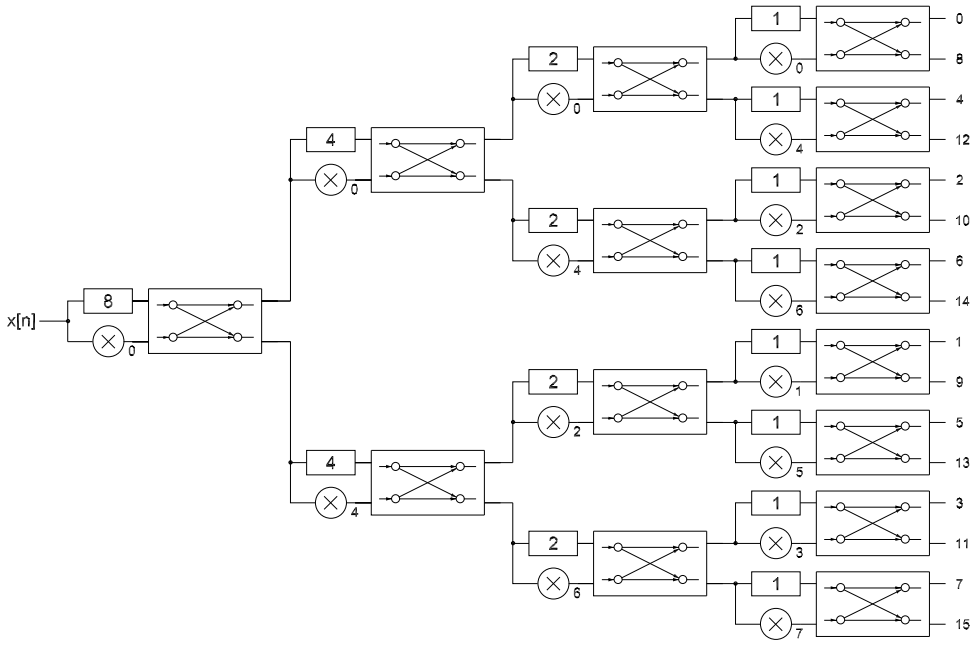

Fig. 5. Hardware architecture of the proposed 16-point radix-2 DIT STFT.

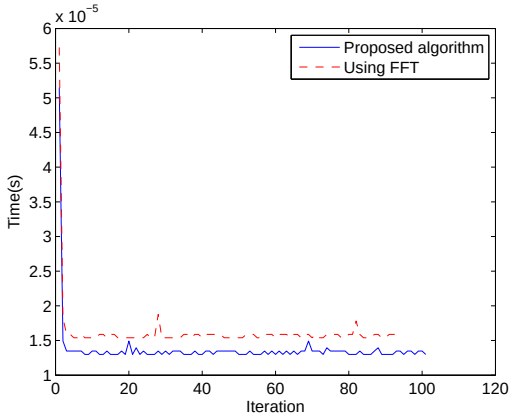

Fig. 6. Execution time of the proposed vs. FFT-based algorithm, in an Intel CORE i3 using MATLAB R2009b.

stages with butterflies, multipliers and buffers. Boxed numbers represent the length of the buffers, whereas numbers close to the multipliers indicate the value ϕ of the twiddle factor. Note that all the multipliers in the feedforward STFT multiply by a constant, which simplifies the design.

Like the feedforward STFT algorithm, the number of butterflies and multipliers increases with the stage. Specifically, the number of adders in stage s is 2^s, the number of constant multipliers is 2^{s-1}, and the memory is $N/2$. This leads to a total of $2N-2$ adders, $N-1$ multipliers and a memory size of $(N/2)\log_2 N$ in the entire architecture.

From the DIT FFT algorithm, the feedforward STFT architecture inherits the property that the output of each stage s provides the result of a 2^s-point STFT.

Table I compares the FFT-based, iterative and feedforward STFTs. Compared to the FFT-based STFT, the feedforward

TABLE I
COMPARISON OF STFT HARDWARE IMPLEMENTATIONS

STFT Implementation	FFT-based	Iterative	Feedforward
Adders	$2N\log_2 N$	N	$2N-2$
Multipliers	$N\log_2 N$	N	$N-1$
Memory	N^2	$2N$	$(N/2)\log_2 N$
Accumulative Error	No	Yes	No

STFT reduces the amount of adders, multipliers and memory significantly. Compared to the iterative STFT, the feedfoward STFT has comparable amount of adders and multipliers and has the advantage that it does not have accumulative error in the calculations.

VI. STFT FOR REAL-VALUED INPUTS

The STFT version for real-valued inputs is derived from the proposed STFT by following the approach in [16]. This approach considers the property that in a real-valued FFT $X[N-k] = X^*[k]$. This allows for removing the calculation of part of the flow graph as shown in Fig. 8. Specifically, $X[12]$ is obtained from $X[4]$; $X[14]$ and $X[6]$ from $X[2]$ and $X[10]$; and $X[15]$, $X[7]$, $X[11]$ and $X[3]$ from $X[1]$, $X[9]$, $X[5]$ and $X[13]$, respectively.

The impact in the feedforward STFT algorithm is that certain lines of the algorithm can be removed. Specifically, the lines with a star ($\star$) in equation (4) do not need to be calculated. They correspond to the frequencies that can be obtained from their symmetric frequency.

With respect to the STFT architecture a number of butterflies, buffers and multipliers can be removed from the feedfoward STFT in Fig. 5, leading to the architecture in Fig. 7. Specifically, the datapaths for the frequencies that can be obtained from their symmetric frequency are removed. Furthermore, in the real feedforward STFT in Fig. 7, some

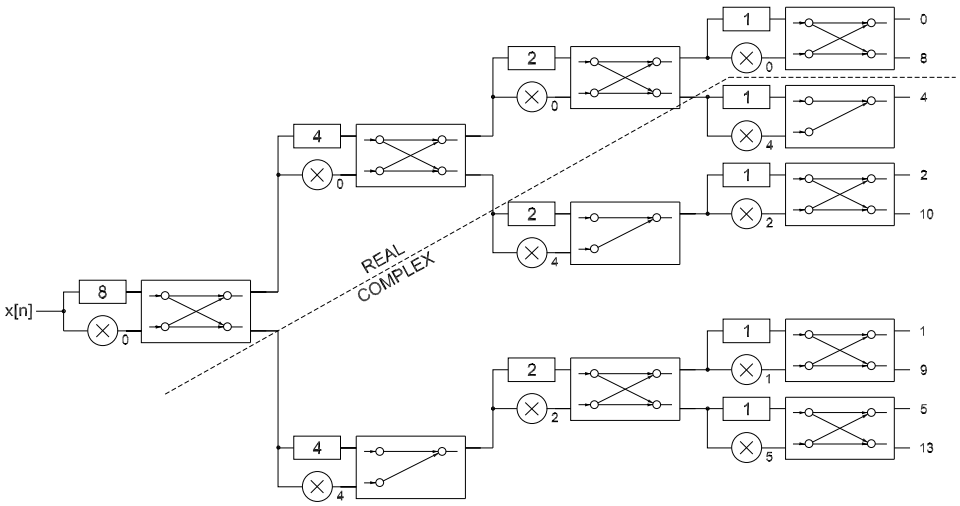

Fig. 7. Hardware architecture of the proposed 16-point radix-2 DIT STFT for real-valued inputs.

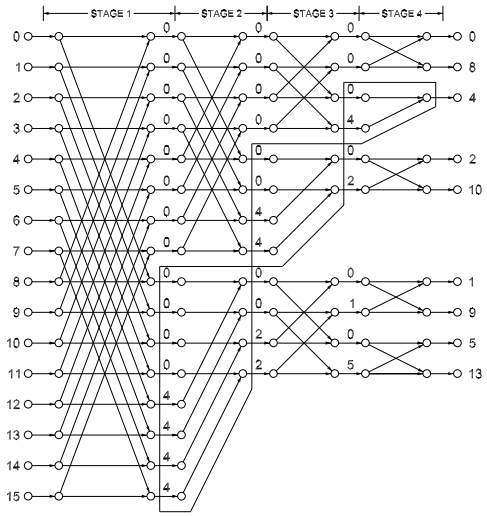

Fig. 8. Flow graph of a 16-point radix-2 DIT FFT for real-valued inputs.

of the datapaths are real, which simplifies the structure. This comes from the flow graph in Fig. 8, where all signals are real until they reach the first complex multiplications, highlighted in boxes.

VII. Conclusions

This paper has presented the feedforward STFT. It reuses operations among consecutive time instances of the FFT calculations. At the algorithmic level this results in less operations than calculating the STFT with the use of the FFT. At architectural level the result is less operations than FFT-based STFTs, without the accumulative error of iterative STFTs.

The feedforward STFT can be adapted to calculate it over real-valued inputs, leading to further savings in calculations.

References

[1] S. Zhang, D. Yu, and S. Sheng;, "A discrete STFT processor for real-time spectrum analysis," in *IEEE Asia Pacific Conf. Circuits Syst.*, Dec. 2006, pp. 1943–1946.

[2] V. Ivanovic, R. Stojanovic, S. Jovanovski, and L. Stankovic, "An architecture for real-time design of the system for multidimensional signal analysis," in *Signal Process. Conf.*, Sept 2006, pp. 1–5.

[3] V. Ivanovic and R. Stojanovic, "An efficient hardware design of the flexible 2-D system for space/spatial-frequency signal analysis," *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 3116–3125, June 2007.

[4] M. Jacobson, "Time-frequency analysis of heart rate variability," in *International Symp. Signal Process. Applications*, Feb. 2007, pp. 1–4.

[5] M. Glos, "Heart rate and systolic blood pressure variability before and during obstructive sleep apnea episodes," in *International Conf. IEEE Engineering Medicine Biology Society*, Aug. 2007, pp. 263–266.

[6] Z. Weidong and L. Yingyuan, "EEG real-time feedback based on STFT and coherence analysis," in *International Conf. IEEE Engineering Medicine Biology Society*, vol. 2, Oct. 2001, pp. 1869–1871.

[7] M. Sánchez, M. Garrido, M. López, J. Grajal, and C. López-Barrio, "Digital channelised receivers on FPGAs platforms," in *IEEE International Radar Conf.*, May 2005, pp. 816–821.

[8] M. Sánchez, M. Garrido, M. López, and J. Grajal, "Implementing FFT-based digital channelized receivers on FPGA platforms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 44, no. 4, pp. 1567–1585, Oct. 2008.

[9] D. Zahirniak, D. Sharpin, and T. Fields, "A hardware-efficient, multirate, digital channelized receiver architecture," *IEE Trans. Aerospace Electronic Systems*, vol. 34, no. 1, pp. 137–152, Jan 1998.

[10] H. Thornburg, R. Leistikow, and J. Berger, "Melody extraction and musical onset detection via probabilistic models of framewise STFT peak data," *IEEE Trans. Audio Speech Language Process.*, vol. 15, no. 4, pp. 1257–1272, May 2007.

[11] S. Amornwongpeeti, N. Ono, and M. Ekpanyapong, "Design of FPGA-based rapid prototype spectral subtraction for hands-free speech applications," in *Conf. Asia-Pacific Signal Information Process. Association*, Dec 2014, pp. 1–6.

[12] M. McCallum and B. Guillemin, "Stochastic-deterministic MMSE STFT speech enhancement with general a priori information," *IEEE Trans. Audio Speech Language Process.*, vol. 21, no. 7, pp. 1445–1457, July 2013.

[13] K. R. Liu, "'Novel parallel architectures for short-time Fourier transform'," *IEEE Trans. Circuits Syst. II*, vol. 40, no. 12, pp. 786–790, Dec. 1993.

[14] D. Petranovic, S. Stankovic, and L. Stankovic, "Special purpose hardware for time frequency analysis," *Electronics Letters*, vol. 33, no. 6, pp. 464–466, Mar. 1997.

[15] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit reversal," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 657–661, Oct. 2011.

[16] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.