Optimized FPGA Based Continuous Wavelet Transform

Yahya T. Qassim[a,*], Tim R.H. Cutmore[b], and David D. Rowlands[a]

[a] *School of Electronic Engineering, Griffith University, 170 Kessels Road, Nathan, Brisbane, QLD 4111, Australia*
[b] *School of Applied Psychology, Griffith University, Mt Gravatt Campus, Kessels Road, Brisbane, QLD 4111, Australia*
[*] Corresponding author. Tel.: +61 431255842
E-mail addresses: yahya-taher.qassim@griffithuni.edu.au (Y.T. Qassim), t.cutmore@griffith.edu.au (T.R.H. Cutmore), d.rowlands@griffith.edu.au (D. D. Rowlands).

Abstract:

A memory efficient field programmable gate array (FPGA) method is described that facilitates the processing of the continuous wavelet transform (CWT) arithmetic operations. The CWT computations were performed in Fourier space and implemented on FPGA following several optimization schemes. First, the adapted wavelet function was stored in a lookup table instead of computing the equation each time. Second, the utilization of FPGA memory was highly optimized by only storing the nonzero values of the wavelet function. This reduces 89% of the memory storage and allows fitting the entire design into the FPGA. Third, the design decreases the number of multiplications and shortens the time to produce the CWT coefficients. The proposed design was tested using EEG data and demonstrated to be suitable for extracting features from the event related potentials. Fourth, wavelet function scales were eliminated which saves further resources. The achieved computation speed allows for real time CWT application.

***Index terms*-CWT, EEG, ERP, FPGA, Optimization.**

1. INTRODUCTION

The one dimensional continuous wavelet transform (1-D CWT) is a widely used feature extraction tool for nonstationary signals with applications to many different disciplines [1]-[3]. The complexity implied in the CWT belongs to the high number of convolutions involved especially between large sequences. In case of using the CWT in real time applications, high processing speed becomes critical to overcome these heavy convolution calculations and this can be achieved by using the field programmable gate array (FPGA) platform [4]. There are only a few studies in the literature concerning the implementation of the CWT into VLSI [5]-[9]. The complexity of mapping the CWT convolver in VLSI design was investigated in [5] and various realizations

were presented, although they all depend on the convolution method. Other works rely on a General Purpose Processor (GPP) or DSP processor [7], [8] or require high end FPGA devices [6], [9] to implement the CWT. What has not been published is a CWT design that is fast and can be implemented on low cost FPGA devices.

Mathematically, the CWT is the convolution between the analyzed signal *X(t)* and the wavelet function *ψ(t)* in the time domain such that [10]:

$$C(s,b) = \int_{-\infty}^{\infty} X(t).\psi_{s,b}^{*}(t).dt \qquad (1)$$

Where *C(s, b)* is the wavelet coefficient at time *b* and scale *s* and the symbol * refers to the complex conjugate.

Equation (1) can be implemented in the time domain for small size of input signals where the number of total convolutions is also small.

An alternative method to calculate the CWT for a sampled signal is to use Fourier space instead of the time domain [11]. This can be achieved by transforming both input signals of (1) ($X(t)$ & $\psi_{s,b}(t)$) into the frequency domain using the fast Fourier transform (FFT) which replaces the complex convolution with simple multiplication using the following relationship [12]:

$$g1(t) * g2(t) \Leftrightarrow G1(\omega) \times G2(\omega) \qquad (2)$$

Where lowercase g is the time domain components and uppercase G is the relative frequency domain representation. The product can be transformed back to the time domain using the inverse FFT giving the CWT coefficients [11]. Some of the most commonly used nonorthogonal wavelet functions in the CWT analysis are the Morlet, Paul and the Mexican hat. Their formulas in the time and frequency domain are shown in Table I [11] where the representation of these formulas in both time and frequency domains are computationally complex. For example, in the time domain, the Morlet wavelet function consists of a complex sinusoid in the term ($e^{i\omega_0 t}$) multiplied by a Gaussian envelope. The spectrum of this modulated Gaussian permits for simple interpretation of results due to its smoothness [1]. The normalization factor ($\pi^{-1/4}$) ensures that the Morlet wavelet has unit energy. In the frequency domain, the Morlet wavelet uses the Heaviside step function and in both domains, the Morlet expressions are complex. The mathematical background for the Paul and the DOG wavelet functions are also complex in both domains as one can see from Table I.

TABLE I
THREE DIFFERENT WAVELET BASES IN TWO DOMAINS [11]

Function	Time domain (*t*)	Frequency domain (*ω*)
Morlet (ω_0=frequency)	$\pi^{-1/4} e^{i\omega_0 t} e^{-t^2/2}$	$\pi^{-1/4}\, H(\omega) e^{-(s\omega-\omega_0)^2/2}$
Paul (*m*=order)	$\frac{2^m i^m m!}{\sqrt{\pi(2m)!}}(1-it)^{-(m+1)}$	$\frac{2^m}{\sqrt{m(2m-1)!}} H(\omega)(s\omega)^m e^{-s\omega}$
Mexican hat (*m*=derivative)	$\frac{(-1)^{m+1}}{\sqrt{\Gamma\left(m+\frac{1}{2}\right)}} \frac{d^m}{dt^m}(e^{-\frac{t^2}{2}})$	$\frac{i^m}{\sqrt{\Gamma\left(m+\frac{1}{2}\right)}}(s\omega)^m e^{-(s\omega)^2/2}$

H(ω) is the Heaviside step function (for Morlet & Paul), *H(ω)*= 1 if *ω*>0, *H(ω)*= 0 elsewhere, ω_0 is the nondimensional frequency, *s* is the scale and $\pi^{-1/4}$ is the normalization factor. Γ Stands for the gamma function.

The selection of the more appropriate wavelet function for a given application depends on the information required to be extracted from the signal. For example, detecting evolutionary or transient phenomena in a signal requires a wavelet function that reflects more localized wavelet coefficients [13].

Compared against the previous works presented above, the CWT design in this paper is a novel, generalized and configurable feature extraction engine for low end FPGA platforms. The design uses Fourier space techniques and employs several optimization methods to improve speed and significantly reduce resource requirements. The proposed design is not limited to a fixed wavelet function *ψ(t)* but can be easily adapted to different wavelet functions without the need to resynthesize or redesign the circuit. The proposed design uses optimized lookup tables (LUTs) in a block RAM (BRAM) to store the pre-calculated wavelet function. This is advantageous since the pre-calculated wavelet function only needs to be downloaded to the BRAM which means that the circuit design is not affected. It also means that the contents of the LUT can be easily replaced by another wavelet function when required by changing the LUT contents. This makes the low cost Spartan 3AN (1.4 M gate) [4] suitable for the proposed design.

This paper is organized as follows; Section 2 presents the CWT design description, Section 3 gives the optimizations used in the CWT design and Section 4 outlines an implementation example based on Electroencephalogram (EEG) analysis. Section 5 presents the discussion and conclusions are provided in Section 6.

2. THE PROPOSED DESIGN

The computation of the CWT in Fourier space has been previously shown as an efficient and quick approach to implement the CWT using the FFT [12]. The design flow for the FFT based digital CWT can be seen in Fig.

1 [14]. From Fig. 1, block *A* contains the wavelet function in the time domain, g2(t), at different scales and the FFT process required to transfer this function to the frequency domain. Both the input signal, g1(t), of length 2^n, and the wavelet function (g2(t)) are stored in a buffer. After applying the FFT on both g1(t) & g2(t) the results G1(ω) and G2(ω) are stored in buffers C1 and C2 respectively. The contents of these two buffers are multiplied before applying an IFFT to produce the wavelet coefficients at all the wavelet scales.

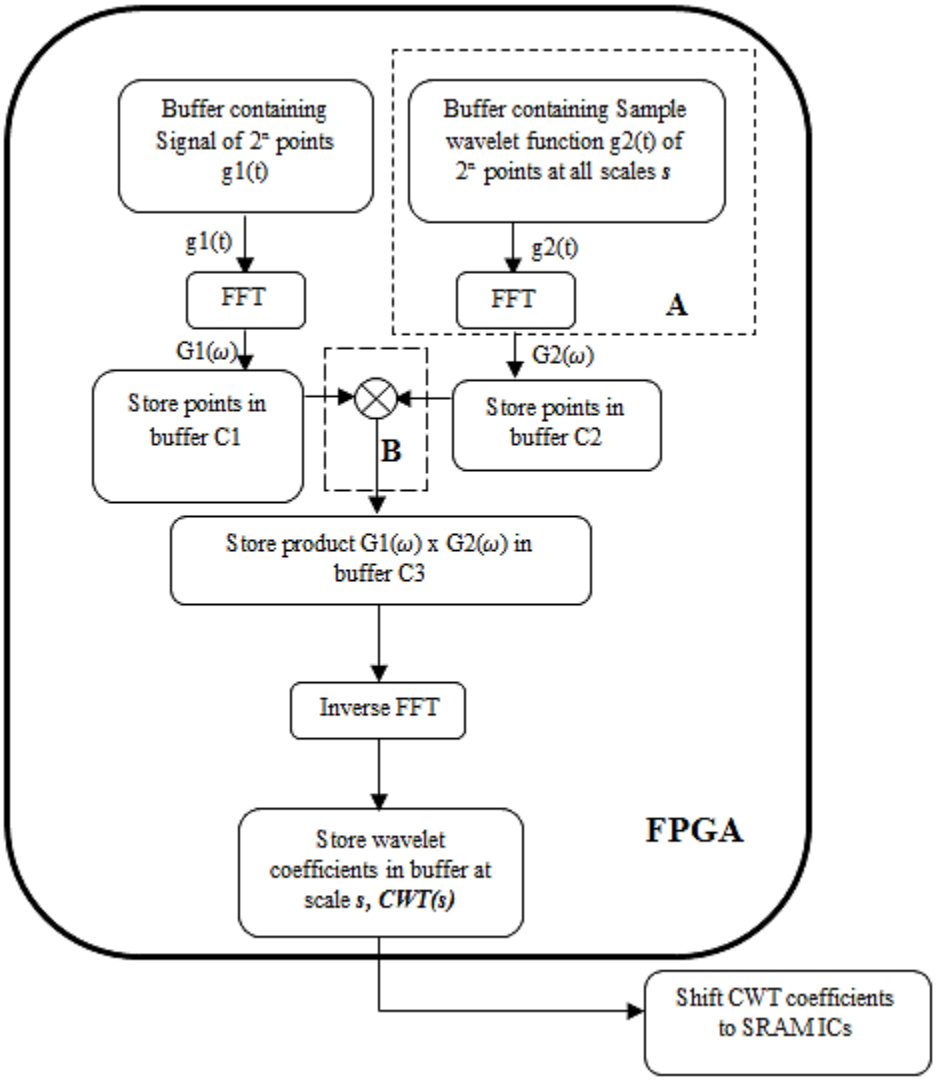


Fig. 1. Flow chart for the digital implementation of the FFT based CWT (BRAM is used as buffer; SRAM ICs are used for final storage. A & B are the optimized blocks in the design

Closer examination of the design flow for the CWT in Fig. 1 reveals that the wavelet functions in block *A* can be implemented using 3 possible methods:

1. Directly calculate the wavelet function in the time domain at different scales then apply the FFT to

transform both the wavelet function and the signal into the frequency domain then start the multiplications with the signal to be analyzed

2. Directly calculate the wavelet function in the frequency domain at different scales, then apply the FFT on the input signal and perform the multiplications between the wavelet function and the input signal.
3. Store the pre-calculated frequency domain-wavelet function points in a lookup table (LUT). In this case, once the length of the input signal and its sampling period are known, the points of the wavelet function can be pre-calculated and downloaded as a LUT. This also enables data reduction schemes to be employed on the wavelet values.

Methods 1 and 2 to calculate the CWT are structurally complex and implementing them in hardware cannot be easily realized. The trigonometric function, square root, power or factorials in the wavelet function in both domains (Table I) require significant FPGA resources as well as adding further computations which increase the run time. In addition, they require a separate design and synthesis for each new wavelet basis function to be used. Method 3 uses a LUT for implementing block *A* and avoids all the mentioned complexity in methods 1 and 2 where no transform operations or wavelet function calculations are required. Therefore Method 3 was selected for the CWT design because it reduces the complexity and the calculation time during real time application.

3. OPTIMIZATION

Although the FFT-IFFT method reduces the total operations required to calculate the wavelet coefficients, further optimization schemes were followed to reduce the FPGA resource usage and speed up the calculations. These optimizations result from adapting the LUT in the design as explained in the next sections.

a. LUT

The LUT was used to store the points of the wavelet function which were pre-calculated in the frequency domain G2(ω) and downloaded to the LUT (C2). The wavelet function stored in C2 is constant and can always be used with any input time series. When the wavelet base function needs to be changed, another wavelet function can be calculated and stored in C2. This approach makes the design versatile since it is applicable to many different types of wavelet functions.

b. Zero Exclusion

As the LUT method is used to store the points of the wavelet function in the frequency domain, the transform

of the wavelet function to the frequency domain can be optimized. The shape of the wavelet basis function at different scales can be used to optimize the storage size required as can be seen in Fig. 2. The shape of the wavelet function contains a large number of zeros especially at higher scales. These zeros represent a considerable portion of the points of the wavelet function at each scale. The wavelet functions in Table I at scales 10 and 20 for Morlet, Paul and the Mexican hat are shown in Fig. 2. It can be noticed that zeros represent a considerable part of each vector of 1024 points for each wavelet type. These zeros can therefore be excluded from the LUT with only the nonzero points being stored. This has a large benefit in saving area in the LUT as well as multiplication operations when mapped into FPGA design. This case does not apply when these wavelet functions are in the time domain [11].

It can also be seen in Fig. 2 that the higher wavelet scales (e.g. scale 20) are compressed and have the least number of nonzero points which facilitates memory optimization more so than the lower scales (e.g. scale 10). The lower scale (scale 10) has a wider distribution and occupies more memory locations.

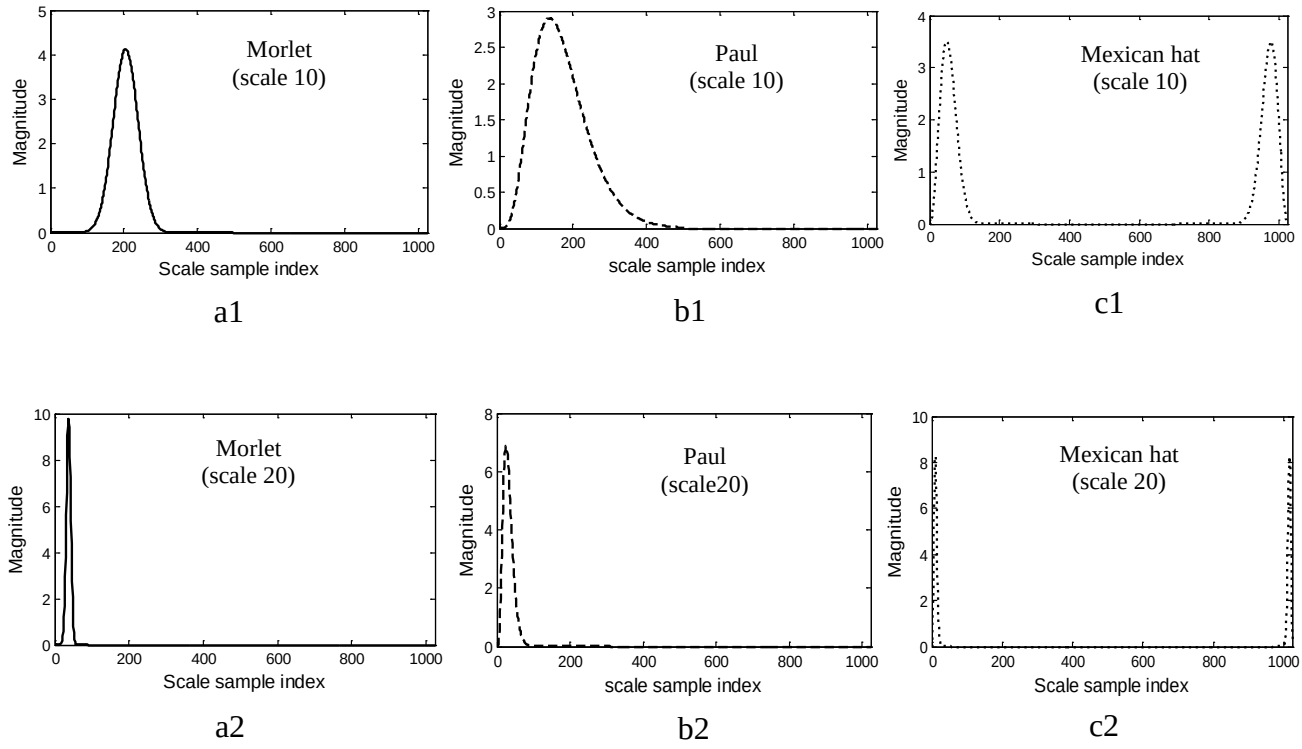


Fig. 2. Three different wavelet functions in the frequency domain, top row is in scale 10 and bottom row is in scale 20. (a1&a2) Morlet. (b1&b2) is Paul and (c1&c2) is the Mexican hat.

c. Multiplications

The exclusion of zeros in section 3.*b* allows further optimization to block *B* in Fig. 1 by reducing the total

number of multiplications required since multiplication with zero does not need to be performed as it is trivially equal to zero. The multiplication process between the input signal and the wavelet function was performed as follows: each sample of the wavelet function was multiplied by the corresponding real and imaginary samples from the signal (both in the frequency domain) using two multipliers in parallel. Since zeros were removed from the wavelet function, the indices of the input signal depend on the indices of the wavelet function at a certain scale ***s***. At higher scales, only a few points from the wavelet function are used with the multiplication circuit since most of the wavelet function vector points are zeros (see Fig. 2). The products of multiplication are stored in the LUT C3 (see Fig. 1).

d. Elimination of Certain Scales (Application specific)

Based upon prior knowledge of the input signal frequency components, certain wavelet scales can be eliminated that take calculation time in addition to LUT storage and hence only need to consider the scales that correspond to the frequencies of interest in the signal.

The optimization schemes in this section are effective for the CWT design and they are all implemented in a case study as outlined in the following section.

4. IMPLEMENTATION EXAMPLE

This section shows a design example for the proposed methodology which analyzes an EEG signal using the Morlet wavelet. The EEG is a weak signal with amplitudes < 100 μV and typically examined frequency components occur between 0.5-100 Hz [15]. The example presented in this section provides a demonstration for the optimizations listed in section 3. This section shows a CWT example using the Morlet wavelet function in transforming an EEG signal in FPGA. Section 4.*a* describes the application and shows how the zero-exclusion case is employed. Section 4 (*b*&*c*) presents a criteria to increase the frequency of operation for the whole system.

a. Morlet Based CWT Application

The Frequency-domain based Morlet wavelet function was chosen for the CWT design among the other functions since it is a simple function and suitable for spectral analysis [16]. This Morlet wavelet function has a center frequency equal to 1 for a good trade-off between time and frequency localization [2]. The Morlet wavelet was calculated in the frequency domain and represented with a sequence of 1024 points at each scale.

The EEG signal was broken into epochs of 1024 ms sampled at 1000 Hz. A typical EEG[1] used in this application can be seen in Fig. 3 which is the response when a participant is presented a specific stimulus. In the present study this was an unexpected (or "oddball") event. The brain response is contained within the event related potential (ERP). This reflects a synchronous firing of a population of neurons within few hundreds of milliseconds after the stimulus is presented and contains the well-known P300 component. The P300 component has a peak approximately 300-500 ms after the presentation of the stimulus [15]. A signal length of 1024 ms was sufficient for the stimulus to be processed and for a brain response to be recorded. The finite signal length in the ERP application (1024 ms) allows fixing the number of wavelet scales required for analysis to 37 where the number of these scales depends on the length of the analyzed signal and its sampling interval [11]. A fixed number of wavelet function scales enables pre-storing the wavelet function at those scales inside the FPGA BRAM. Both points of the Morlet wavelet and the EEG signal were represented by 16 bits in the design.

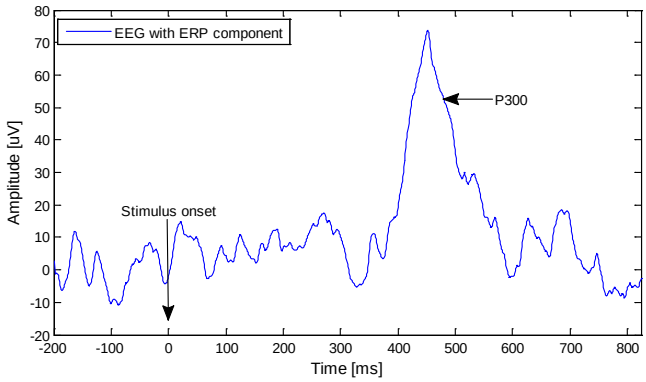


Fig. 3. The EEG signal used to test the CWT engine

The Xilinx Spartan 3AN FPGA was used as a target technology due to its low cost, availability and its common use. Previous work by the authors has employed a FFT-IFFT approach to calculate the CWT [17]. The FFT engine employed was a Xilinx FFT core V.5 and was configured with radix-4 burst I/O to perform the FFT-IFFT with the designed VHDL controllers. Radix-4 configuration option is an area-speed trade-off between radix-2 and the streaming architecture (other available configuration options for the FFT core) [18]. The detailed implementation architecture including FFT-IFFT, resource utilization are available in [17].

Instead of storing the Morlet wavelet for the whole scale of 1024 points and for 37 times, zeros can be excluded from computations and only nonzero points were considered as outlined in 3.*b*. The inclusion of the

[1] The Griffith University ethics number for human research was (PSY/92/09/HREC).

whole length of scales requires $(37 \times 1024\ point) \times 16\ bit/point = 606208$ bit locations. By excluding zeros, the total number of required locations including all the 37 scales was reduced to $4138 \times 16\ bit = 66208$ locations. To comply with the memory standard as a power of 2, the number of locations is further reduced to end with $4096 \times 16\ bit = 65536$ (or 4K × 16) total locations to store the Morlet function (about 89% reduction compared to using all points of all scales). The later reduction is done here by excluding some nonzero points from the wavelet function at scale number one which corresponds to the frequency 1000 Hz (normally considered to be well out of the EEG frequency range) and has no serious effect on the produced results. As a result of zero-exclusion, the total number of multiplications is reduced from 2 (real & imaginary components) ×37K) to only 2×4K. Table II shows the wavelet scales applied with the corresponding frequencies and the indices range with the length of the nonzero Morlet points. For example, at scale 17, Morlet nonzero wavelet values are in the index range 23-101 (out of 1024). This range is multiplied by the corresponding range from the EEG in the frequency domain.

TABLE II
SELECTIVE SCALES FOR MORLET WAVELET WITH THE CORRESPONDING FREQUENCIES AND THE NONZERO SAMPLE INDEX AT EACH SCALE. DOT POINT IN EMPTY CELLS REFER TO THE SCALES LOCATED BETWEEN SELECTIVE ONES

Morlet Scale #	**Frequency [Hz]**	**nonzero Morlet points index**		
		Start	**end**	**vector length (points)**
1	1000	406	513	108 (reduced to 65)
2	833	335	513	179
.	.	.	.	.
.	.	.	.	.
17	62.5	23	101	79
18	52.63	19	86	68
.	.	.	.	.
.	.	.	.	.
36	2.32	2	4	3
37	1.95	2	4	3
				Sum= 4096 points

When the multiplication process was completed, the next step was to perform the IFFT. The number of points at each scale product depends on the nonzero Morlet values at that scale. Zero insertion before and after each product array was performed during the IFFT data loading task to complete a vector of 1024 point. The IFFT was repeated 37 times. The resulting CWT coefficients required two arrays of bits, each being: $37\ scales \times 1024\ point \times 16\ bit$ memory size. One is for the real CWT component and another one for the imaginary component. Due to the limited size of memory inside the Spartan 3AN (BRAM) to save all the CWT coefficients, the SRAM chip was used (see Fig. 1).

b. SRAM Usage

The static RAM (SRAM) is located on the Altium NanoBoard which is the design environment [19] outside the FPGA and was used because the rest of the available BRAM was not enough to store the CWT coefficients on the size of two $37\ scale \times 1024\ point \times 16\ bit$ ($2 \times 37\ K$ points). For that, two BRAMs (each $1K \times 16$ bit) were used to temporarily store the wavelet coefficients at individual scales after the completion of every inverse FFT process. One of them was for real CWT component and the other for the imaginary component. While the FFT core loads new data and computes the inverse FFT of that data for the next scale, the CWT coefficients inside the BRAM were transferred into the SRAM. Before unloading new scale of wavelet coefficients, the FFT core requires 1024 clock cycles for loading + 1366 clock cycles for computation = 2390 clock cycles. The real and imaginary CWT coefficients were transferred from the BRAM into the SRAM in parallel using 2 clocks per coefficient (2048 clock cycles for one full scale). Therefore, there was enough time for the BRAM data to be copied to the SRAM before the BRAM is overwritten by the next scale of wavelet coefficients. Table III illustrates the data transfer schedule from the FFT core passing into the BRAM and then to the SRAM at two different times T1 and T2.

TABLE III
TIMING SCHEDULE FOR THE BRAM & SRAM WITH THE FFT CORE (CLOCK CYCLES NUMBER BETWEEN BRACKETS)

	T1= 2390 clock		**T2= 1024 clock**
FFT core (Inverse FFT process)	Compute the CWT from the loaded multiplication products at scale ***s*** (2390)		Unload the CWT coefficients into BRAM (1024)
BRAM	Data send to the SRAM (2048)	Idle	Data received from the FFT core (1024)
SRAM ICs	Data received from the BRAM (2048)	Idle	

Fig. 4 shows the schematic diagram control circuit for "write/read" of the BRAM in the Altium environment. As long as the data is valid by the FFT core output, the output *dvp* becomes high and the BRAM is addressed by the core output *xk_indexp*[9..0]. The *WE* input for the BRAM receives logic high from the and-gate (Fig. 4). The other input of the and-gate is from the core input *fwd_invp*. This pin controls the FFT process; when at '1', the core performs the forwards FFT whereas '0' changes the task to the inverse FFT. Accordingly, the BRAM cannot receive data unless the *fwd_dvp* = '0' and the *dvp*= '1' at the same time. When all the CWT coefficients for one scale are transferred into the BRAM, the *dvp* is switched to '0' to change the address source of the BRAM to be addressed by a VHDL based controller. This monitors the transfer of data from the BRAM into the SRAM. Not all the wiring connections are shown in Fig. 4.

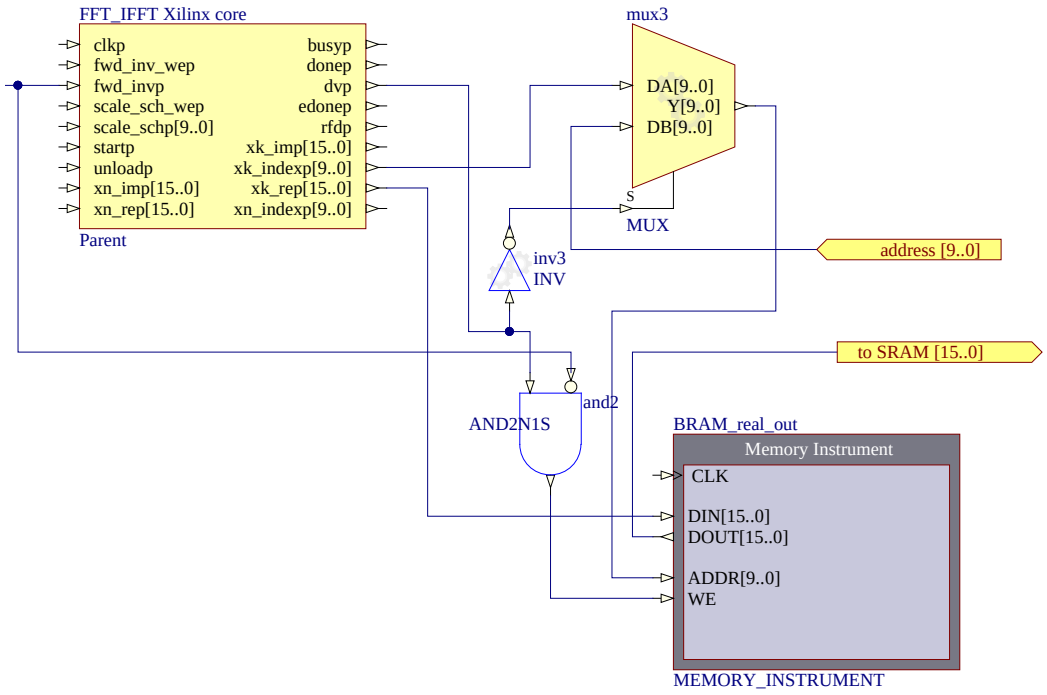


Fig. 4. The control circuit for the CWT coefficients in the BRAM

The implication for the involvement of the SRAM was that the maximum theoretical access time for the SRAM according to the data sheet is 12 ns (maximum speed of data transfer is about 80 MHz which justifies the reason for indirect storage in the SRAM). This forces the maximum frequency of operation for the system to be 80 MHz (with the SRAM). However, the critical path of the CWT design allows rising the clock frequency up to 125 MHz (without the SRAM). Since the SRAM was controlled to receive data at the rate 2 clock/sample, the design can operate at 125 MHz and the SRAM receives the CWT coefficients at half of this rate (62.5 M sample/second). This technique results in a very short time to produce the CWT coefficients. The total occupied BRAM for the CWT design was 90% from the available BRAM inside the FPGA compared to 140% utilization without the involvement of the SRAM in the design.

c. Scale Elimination

According to section 3.*d*, as most of the EEG frequency components are less than 60 Hz [20], upper frequency components can be omitted to reduce the number of computations and save required BRAM. Fig. 5 shows the FFT single sided amplitude spectrum of an EEG signal. The frequency components of the EEG are concentrated in the lower area in Fig. 5(a) representing the EEG frequencies of interest whereas Fig. 5(b) shows very small frequency components and therefore, the frequency range 60-200 Hz can be ignored and excluded from the CWT computation.

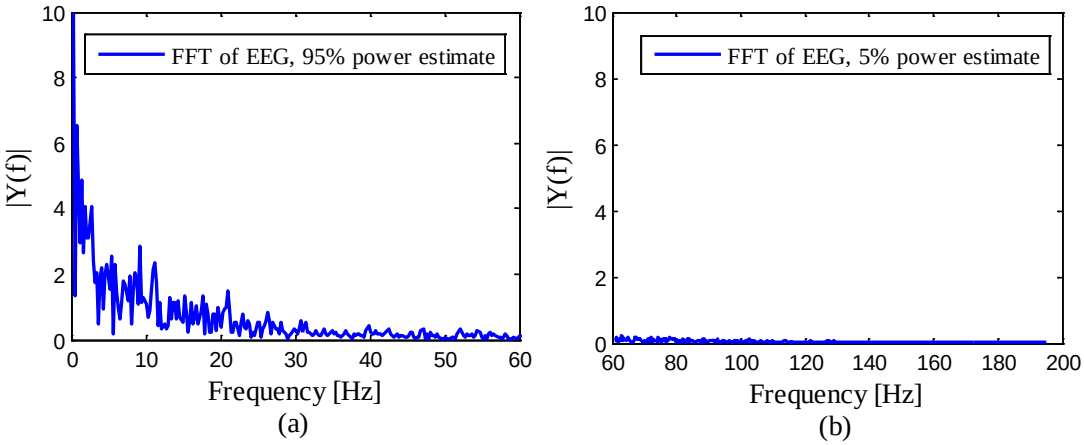


Fig. 5. The FFT of a 2200 ms EEG with the frequency range (a) 0.5-60 Hz, (b) 60-200 Hz.

Excluding higher frequencies means that the lower scales of Morlet are not going to be involved in the computation and therefore there is no need to calculate or store them in the BRAM. Only the CWT for the frequency range 2-60 Hz is covered which is where the major EEG frequency components are located. Fig. 6 shows the scales excluded from computation. These are scales 1-16 which reflect the frequency components in the EEG signal higher than 62 Hz as presented in Table II. The rest of the scales 17-37 (corresponds to 2-62.5 Hz) cover the typical frequencies of the EEG. The 4096 locations required to store the 4096 points for the wavelet function at 37 scales in Table II was reduced to only 499 locations by this exclusion which means saving of the BRAM by more than 87%. The number of total multiplications and the total computation time are also reduced.

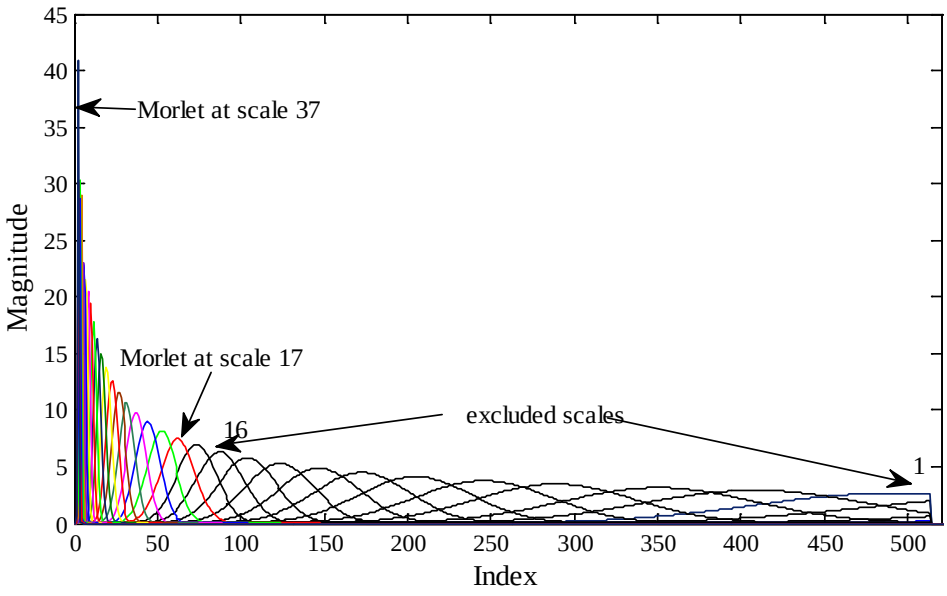


Fig. 6. The excluded Morlet scales (1-16) from the CWT computations.

The exclusion of the unnecessary scales resulted in an improvement in the timing report for the design compared to the first case that included the wavelet function at all scales. Table IV shows a comparison between the first case (37 scales included) and the second case (scales 17-37 only considered). From Table IV, it can be noticed that both designs operate at 125 MHz. However, in the scale-excluded case, it indicates a higher speed in calculating the CWT. A run time of 1 millisecond was achieved for the entire CWT scale set which is reduced to 0.6 ms in the scale-excluded set. In addition, the reduction in computations shortens the critical path which raises the design clock rate to 133 MHz resulting in lower computation time. With the 133 MHz, the time of computations is reduced to 0.57 ms instead of 0.6 ms, which is almost half the required time of the entire-scale case. The achieved improvement in the maximum clock rate is due to a shorter critical path as a result of reduction in complexity.

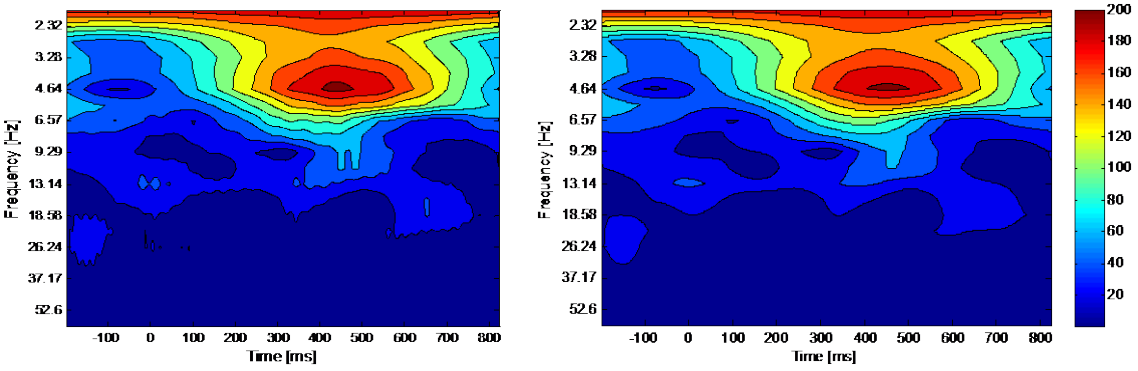
Despite the increase in the run time to calculate the CWT (in case the SRAM was not used) to few milliseconds only (calculated theoretically), the involvement of the SRAM was indispensable to overcome the space limitation in the FPGA BRAM.

TABLE IV
COMPARISON OF TIMING SCHEDULE FOR THE DESIGN IN TWO DIFFERENT RANGES OF WAVELET SCALE

		(A) 37 scales (1-37)	**(B) 21 scales (17-37)**
Task		**# Clocks required**	**# Clocks required**
FFT	load	1024	1024
	compute	1366	1366
	unload	1024	1024
Multiplications		4096	499
Inverse FFT (A) 37 times (B) 21 times	load	1024 x 37 (37888)	1024 x 21 (21504)
	compute	1366 x 37 (50542)	1366 x 21 (28686)
	unload	1024 x 37 (37888)	1024 x 21 (21504)
Initialization and task separation		405	405
Total clocks		134233@ 125 MHz= 1 ms	76012@ 125 MHz=0.6 ms 76012@ 133 MHz= 0.57 ms

Software calculation of the CWT was used to verify the result produced by the FPGA. Matlab was used to calculate the CWT as well as to display the results from the FPGA. Fig. 3 shows the ERP signal used to test the CWT-FPGA and also used to generate the same CWT-software based analysis. As a result, Fig. 7 (a) shows the CWT scalogram based on the FPGA and Fig. 7 (b) shows the CWT scalogram based on the software (Matlab)

calculation. Only the frequencies of interest were included. The differences between Fig. 7 (a&b) are due to the quantization error in Fig. 7 (a). The impact of quantization error on the results was addressed in [21] and indicated the validity of the undertaken FPGA method.



(a) (b)

Fig. 7. The CWT scalogram for the EEG in Fig. 3: (a) FPGA based. (b) Matlab based.

5. DISCUSSION

In order to check the performance of the CWT engine design, the time taken to compute the CWT needs to be compared to other designs in the literature [7]-[9] as given in Table V. Comparison is not easy due to the different implementations and different hardware platforms. Table V shows a range of parameters to allow a more fair comparison between these references. In addition, both [7] and [8] are applications in the biomedical field with the same signal size used in this work. The listed works in Table V shows different CWT algorithms implemented on different VLSI technologies and at different operational frequencies. Lower run time in the table indicates better implementation.

Although our FPGA target technology has moderate specifications, with the presented optimizations, it was suitable to the CWT design entirely and able to achieve the minimum run time of 0.57 ms compared to previous works listed in Table V. References [7] and [8] had employed different VLSI technologies, however, the signal length is close to the one adapted in this work. They achieved run times of 5 ms and 47.9 ms respectively. Reference [9] is closer to compare with the work presented in this paper since both employ FPGA chips whereas [7] and [8] used the processor solution. Reference [9] employed a high end FPGA device to implement the lifting scheme for a two-dimensional signal and obtained a run time of 12.16 ms. Finally, compared to the software case, the same processing algorithm implemented in this paper consumes 16 ms in Matlab using Apple

MacBook Pro PC running Windows 7-64 bits with the following specifications; dual core i5 CPU@ 2.4 GHz, 4 GB RAM.

TABLE V
COMPARISON OF VLSI BASED CWT IMPLEMENTATION FROM THE LITERATURE. LOWER RUN TIME VALUES ARE BETTER

Ref.	Year	CWT algorithm	Operational Frequency [MHz]	VLSI technology	Application	Run time [ms]	Size of signal
[7]	2004	t-CWT	704	General purpose microprocessor	ERP feature extraction	5 (average)	1000
[9]	2007	Lifting scheme	20	FPGA: Virtex 2pro-XC2V500	Image edge detection	12.16	512 x512 grayscale image
[8]	2009	B-spline	30	DSP processor: TMS320C6713	EMG interference pattern	47.9	1024
This work	2013	FFT Based	133	FPGA: Spartan 3AN-XC3S1400AN	ERP feature extraction	0.57	1024
			2400	Matlab running on windows 7 laptop with i5 processor and 4 GB RAM		16	

The achieved low run time in the presented design is highly related to the optimizations of using LUT, zero exclusion, reduction in multiplications and scale elimination followed in Section 3 of this paper that effectively reduce the operations and the CWT calculation time. The wavelet function was directly stored in the memory which saved extra 37 FFT computations. In addition, zero exclusion assisted in saving 89% of memory usage comparing with the case no zero exclusion being made. Furthermore, in the method of zero exclusion, the eliminated scales (1-16) have reduced 87% from the BRAM required to store the wavelet function compared against the case that uses the Morlet wavelet function at all scales (1-37). This leads to a reduction in the number of required operations, an increase in the operational frequency and reducing the total computation time. The differences between Fig. 7 (a & b) are belonging to the quantization error produced during the FFT-IFFT processing and in representing the points of the signals by a finite word length (16 bit/point) in the FPGA [21].

According to the applied optimization methods, the improvement in the CWT design is summarized in Table VI.

TABLE VI
IMPROVEMENTS COMPARISON WITH THE OPTIMIZATIONS APPLIED

	Optimization applied	BRAM locations required to store the Morlet function	Maximum FPGA clock rate (MHz)	Number of multiplications between the EEG and the Morlet	Running time (ms)
1.	Zero exclusion	4K × 16= 65536	125	2 × 4096	1
2.	Zero exclusion and scale reduction	499 × 16= 7984	133	2 × 499	0.57

From Table VI, one can notice that the scale reduction was useful in further reducing the design running time. Without the zero exclusion, the available BRAM is not enough to store the wavelet function and the design intermediate results since the wavelet function alone requires 606208 locations to be stored which exceed the available BRAM in the FPGA. This number is reduced to only 11% (65536 locations) when the zero exclusion was adapted. Storing the design input signals in the BRAM (inside the FPGA) was an aim to speed up the whole processing time and leave the SRAM ICs (outside the FPGA) to be occupied by the whole CWT coefficients.

6. CONCLUSIONS

This paper has presented a CWT architecture to process nonstationary signals with the Morlet wavelet function using a Spartan 3AN FPGA. The implemented architecture was flexible and generic due to the pre-calculations performed on the wavelet function. Since these calculations were performed in the frequency domain, the heavy convolutions were replaced with multiplications. In addition, as the shape of the wavelet function in the frequency domain contains a large number of zeros, the exclusion of these zeros assisted in reducing the total number of calculations required for the CWT. The presented design with optimizations was examined by an ERP signal that showed the validity of the technique. The achieved run time of 0.57 ms leads to the conclusion that the proposed CWT can be used to extract features from signals in real time applications. Future work will consider signals of variable length for analysis.

ACKNOWLEDGMENT

This work was supported by the Ministry of Higher Education and Scientific Research of IRAQ.

REFERENCES

[1] A. Klein, T. Sauer, A. Jedynak, and W. Skrandies, "Conventional and wavelet coherence applied to sensory-evoked electrical brain activity," *Biomedical Engineering, IEEE Transactions on,* vol. 53, pp. 266-272, 2006.

[2] A. Grinsted, J. C. Moore, and S. Jevrejeva, "Application of the cross wavelet transform and wavelet coherence to geophysical time series," *Nonlinear processes in Geophysics,* pp. 561-566, 2004.

[3] Y. Qi, V. Siemionow, Y. Wanxiang, V. Sahgal, and G. H. Yue, "Single-Trial EEG-EMG Coherence Analysis Reveals Muscle Fatigue-Related Progressive Alterations in Corticomuscular Coupling," *Neural Systems and Rehabilitation Engineering, IEEE Transactions on,* vol. 18, pp. 97-106, 2010.

[4] *Xilinx, Inc* "Spartan-3AN FPGA Family Data Sheet" [Online]. Available: http://*www.xilinx.com/support/documentation/**data_sheet**s/ds557.pdf*

[5] C. Chakrabarti and M. Vishwanath, "Efficient realizations of the discrete and continuous wavelet transforms: From single chip implementations to mappings on SIMD array computers," *Signal Processing, IEEE Transactions on,* vol. 43, pp. 759-771, 1995.

[6] S. Cheng, C.-H. Tseng, and M. Cole, "A novel fpga implementation of a wideband sonar system for target motion estimation," in *Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on*, 2008, pp. 349-354.

[7] V. Bostanov, "BCI competition 2003-data sets Ib and IIb: feature extraction from event-related brain potentials with the continuous wavelet transform and the t-value scalogram," *Biomedical Engineering, IEEE Transactions on,* vol. 51, pp. 1057-1061, 2004.

[8] S. Patil and E. Abel, "Real time continuous wavelet transform implementation on a DSP processor," *Journal of Medical Engineering & Technology,* vol. 33, pp. 223-231, 2009.

[9] S. Kang, P. Xuezeng, and P. Lingdi, "A Reconfigurable Computing Engine for Wavelet Transforms," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1-5.

[10] P. S. Addison, *The Illustrated Wavelet Transform Handbook*, First ed. Edinburgh, UK: Institute of Physics Publishing, 2002.

[11] C. Torrence and G. P. Compo, "A Practical Guide to Wavelet Analysis," *Bulletin of the American Meteorological Society* vol. 79, pp. 61-78, 1998.

[12] K. Najarian and R. Splinter, *Biomedical signal and image processing*: CRC press, 2012.

[13] X. Li, X. Yao, J. Jefferys and J. Fox, "Computational Neuronal Oscillations using Morlet Wavelet Transform," *in Engineering in Medicine and Biology 27th Annual Conference*, 2005, pp. 2009-2012.

[14] D. Jordan, R. Miksad, and E. Powers, "Implementation of the continuous wavelet transform for digital time series analysis," *Review of scientific instruments,* vol. 68, pp. 1484-1494, 1997.

[15] Y. T. Qassim, T. R. Cutmore, D. A. James, and D. D. Rowlands, "Wavelet coherence of EEG signals for a visual oddball task," *Computers in Biology and Medicine,* vol. 43, pp. 23-31, 2013.

[16] A. Mouraux and G. Iannetti, "Across-trial averaging of event-related EEG responses and beyond," Magnetic resounance imaging, vol. 26, pp. 1041-1054, 2008.

[17] Y. T. Qassim, T. Cutmore, D. James, and D. Rowlands, "FPGA implementation of Morlet continuous wavelet transform for EEG analysis," in *Computer and Communication Engineering (ICCCE), 2012 International Conference on*, 2012, pp. 59-64.

[18] The Xilinx Fast Fourier Transform core data sheet v7.1 [Online]. Available: *crkit.orbit-lab.org/export/4/.../**datasheets**/**Xilinx**/**FFT**/x**fft**_ds260.pdf*

[19] *Altium Limited*, "Nanoboard-3000" [Online]. Available: http://nb3000.altium.com/intro.html

[20] G. Benham, H. Rasey, J. Lubar, J. Frederick and A. Zoffuto, "EEG power-spectral and coherence differences between attentional states during a complex auditory task," *Neuro therapy*, vol. 2, pp. 1-9, 1997.

[21] Y. T. Qassim, T. Cutmore, and D. Rowlands, "Multiplier Truncation in FPGA Based CWT," in *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, 2012, pp. 947-951.

Biography

Yahya T. Qassim: received his BSc. in electrical engineering in 1994 and the MSc in computer engineering in 2004 both from University of Mosul in Iraq. Recently, he received his PhD from Griffith University. He has authored over ten publications in peer reviewed journals and international proceedings. His current research interests include biomedical signals, digital design and FPGA implementations.

Tim R. H. Cutmore: senior lecturer in the School of Applied Psychology at Griffith University. He holds an MSc in Computer Science and a PhD in Psychology, both from Queen's University Canada. Tim has published a number of studies on signal processing of EEG data. His current interest is in the application of pattern recognition algorithms for classifying ERPs in single trials.

David D. Rowlands: received his BSc degree in Physics (1986) from Griffith University in Brisbane Australia and received his PhD in Electronic Engineering (2001) from the same institution. David has researched in different areas including semiconductor devices and signal analysis of biological signals. He is currently researching in real time computing systems and inertial sensors for human movement analysis.