ISSN: 2993-8643

Research Article

Engineering: Open Access

Design of Scalable Architecture for Real-Time Parallel Computation of Long to Ultra-Long Real-Data DFTs

Keith Jones*

Consultant Mathematician (Retired), Weymouth, Dorset, UK

*Corresponding Author
Keith Jones, Consultant Mathematician (Retired), Weymouth, Dorset, UK.

Submitted: 2024, Oct 16; **Accepted**: 2024, Nov 18; **Published**: 2024, Nov 19

Citation: Jones, K. (2024). Design of Scalable Architecture for Real-Time Parallel Computation of Long to Ultra-Long Real-Data DFTs. *Eng OA, 2(*4), 01-11.

Abstract

With the current trend in large scale, big data applications, there is an increasing need for the design and efficient implementation of long to ultra-long Fourier-based transform algorithms, such as with fast Fourier transforms (FFTs) where the transform length varies from long (of order one million) up to ultra-long (of order one billion). To address such problems, the paper shows how the applicability of the scalable, memory-based architecture of the regularized fast Hartley transform (FHT) or RFHT – which has proved an attractive alternative to the FFT for the computation of the discrete Fourier transform (DFT) for when the data is real-valued, as is the case with many real-world applications – may be straightforwardly extended to enable the efficient parallel computation of long to ultra long transforms to be achieved and maintained in a continuous real-time fashion. In order to implement such algorithms, however, when using the memory-based architecture of the RFHT, a timing constraint (and hence transform size limitation) arising from the combined effects of the update period and the I/O rate needs to be overcome and the formidable data and coefficient (or twiddle factor) memory requirement minimized. With this in mind and with a processing element (PE) defined as comprising one complete RFHT module, it is seen how the design of a scalable dual-PE architecture may be derived as a simple extension of the single - PE case – thus possessing a number of attractive properties, as held by the RFHT, but not by pipelined real-data FFT implementations – this being achieved in such a way that the transform size limitation is overcome and, when combined with the use of memory-efficient multi level look-up table (LUT) techniques for the coefficient generation and storage, offers the 'potential' for achieving and maintaining the real - time parallel computation of real-data transforms where the transform length may now range up to one billion. Two hypothetical implementations are briefly discussed which illustrate how the dual-PE solutions for the computation of both the one and four million-point real-data transforms may each be mapped onto a single commercially-available field programmable gate array (FPGA) device using only fast on - chip random access memory (RAM) for the data and coefficient storage.

Keywords: DFT, FFT, FHT, FPGA, Long, Ultra-Long

1. Introduction

With the current trend in large-scale, big data applications, a need has arisen for the design and efficient implementation of Fourier-based transform algorithms where the transform length ranges from long (of order one binary million – or 2^{20}, referred to hereafter as simply one million – as might be encountered with the processing of wideband signals embedded in astronomical data) up to ultra-long (of order one binary billion – or 2^{30}, referred to hereafter as simply one billion – as might be encountered with the processing of ultra-wideband signals embedded in cosmic microwave data). The particular problem area of interest is that of spectrum estimation which may be effectively used to produce precise high-resolution images to facilitate the detection and identification of those objects or phenomena – such as the relative motion and chemical composition of stars and galaxies – that are of particular interest to the researcher. The basic algorithm to be solved is thus that of the discrete Fourier transform (DFT), as given for the case of the N - point transform by the expression

$$X^{(F)}[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n].W_N^{nk} \qquad (1)$$

for k = 0,1, … ,N-1, where the inputs/outputs are typically complex-valued and

$$W_N = \exp(-i2\pi / N), \qquad i = \sqrt{-1}, \qquad (2)$$

the primitive Nth complex root of unity [1,2]. The DFT is an orthogonal transform that is typically carried out by means of a suitably chosen member of the class of fast Fourier transform

(FFT) algorithms, where the complex exponential terms, W_N^{nk}, each comprise two trigonometric components, more commonly referred to as twiddle factors, which are required to be fed into each instance of the FFT's butterfly, this being the name of the computational engine used for carrying out the algorithm's repetitive arithmetic operations [3,4].

A version of this problem, which is of particular interest here, is when the input data to the DFT is real - valued in nature (as is often the case in many real - world applications where the signals of interest are wideband in nature), rather than complex-valued, with the transform length being typically expressed as the power of some fixed integer radix (typically taken as two or four). The conventional approach to such a problem involves the adoption of a specialized real-data FFT (or RFFT) algorithm, based upon the familiar pipelined computing architecture, which necessitates the use of multiple processing elements (PEs) in order to achieve and maintain continuous real-time operation. Such designs, however, are typically highly complex and involve the need for two or more distinct PE designs [5-8].

An attractive alternative to the standard FFT-based approach to the solution of the real-data DFT involves the derivation of resource-efficient parallel solutions to the discrete Hartley transform (DHT) algorithm [9], another orthogonal transform which for the case of the N-point transform is given by

$$X^{(H)}[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n].\mathrm{cas}(2\pi nk/N) \tag{3}$$

for k = 0,1, … ,N-1, where the inputs/outputs are real-valued and the transform kernel is given by

$$\mathrm{cas}(2\pi nk/N) = \cos(2\pi nk/N) + \sin(2\pi nk/N) \tag{4}$$

which is referred to in the literature as the 'cas' function [9]. The trivial conversion of the outputs from Hartley-space to Fourier-space – to yield the required real-data DFT outputs – is as given by the expressions

$$\mathrm{Re}(X^{(F)}[k]) = \tfrac{1}{2}(X^{(H)}[N-k] + X^{(H)}[k]) \tag{5}$$

$$\&\quad \mathrm{Im}(X^{(F)}[k]) = \tfrac{1}{2}(X^{(H)}[N-k] - X^{(H)}[k]) \tag{6}$$

where 'Re' stands for the real DFT component and 'Im' for the imaginary DFT component. This process may be straightforwardly performed post-DHT, although if the power spectral density (PSD) is to be the required output data format then the conversion process may be simply discarded as the PSD may be obtained directly from either the DFT or the DHT outputs [10].

The solution to the DHT that will be pursued here is that of the regularized fast Hartley transform (FHT) or RFHT, which possesses a simple memory-based architecture and has the attractions of being resource-efficient, scalable, bilateral (that is, equal to its own inverse) and highly parallel (yielding eight-fold parallelism) [10]. The simplicity of the single-PE RFHT design results in minimal design costs, in terms of both time and resources, whilst its scalability enables the same design to be used to cater for multiple digital signal and image processing applications possessing varying requirements (in terms of transform length), again at minimal expense.

Comparing the RFFT and RFHT approaches to solving the real-data DFT – as is discussed in some detail in – the RFFT approach is geared to streaming operation and exploits a multi-PE pipelined architecture, whilst the RFHT approach involves the design of a memory-based solution geared to batch operation and exploits a single-PE architecture. Both approaches have yielded attractive power - efficient solutions, although when compared to those of the RFFT approach, the RFHT solution, as stated above, possesses the additional attractions of bilateralism and of increased design simplicity, regularity and scalability – the RFFT solution would need to be optimized for each particular application, a potentially costly process – as well as lending itself more naturally to the adoption of an accurate conditional scaling strategy for fixed-point operation (to be briefly discussed in Section 5).

There is a problem to be overcome, however, with the adoption of a memory-based architecture in that the maximum achievable transform length is constrained by the combined effects of the update period (or refresh rate) and the I/O rate. The aim of this research is thus to produce the design of a simple scalable computing architecture, based upon the RFHT module, which overcomes this size limitation, so that the attractive properties of the FHT – which is a radix 4 fixed-radix algorithm – as stated above, may be effectively exploited for transform lengths ranging from the large (taken here to be 4^{10}, or one million) up to the ultra large (taken here to be 4^{15}, or one billion). It should be noted, however, that the maximum achievable transform length will also be constrained by the existing technology in terms of the amount of fast on-chip random access memory (RAM) available on the chosen parallel computing device for dealing with the memory requirement, although the storage requirements for the coefficients (or twiddle factors) may be minimized through the adoption of a suitably defined multi-level look-up table (LUT) scheme.

Thus, following this introductory section, an outline is given in Section 2 of the regularized FHT, this including mention of: 1) the scalable memory-based architecture; 2) the large highly parallel double butterfly; and 3) the multi-level LUT-based schemes. In Section 3, after first defining the timing constraints associated with the operation of both single-PE and dual-PE solutions, a simple complexity analysis is carried out in terms of the memory and arithmetic requirements, where the adoption of suitable parallel computing equipment is assumed and where the arithmetic requirement is expressed very simply in terms of the required numbers of fast multipliers (as made available by the

equipment manufacturer) and adders (as implemented very simply in programmable logic). This is followed in Section 4 with a brief discussion of two hypothetical implementations which illustrate how the dual-PE solutions for the computation of both the one and four million-point real-data transforms might each be mapped onto a single commercially-available field programmable gate array (FPGA) device using only fast on-chip RAM for the data and coefficient storage. Finally, a brief description of a potential fixed-point scaling strategy is outlined in Section 5, followed by a summary and conclusions in Section 6 [12].

2. A Brief Outline of Regularized FHT

The RFHT is a resource - efficient means of carrying out the DHT (and thus the real-data DFT) that is both highly parallel and scalable, whilst its being 'regularized' refers to the fact that the algorithm structure has been made regular so that the conventional need for two separate butterfly designs for the fixed-radix FHT is thus avoided. The design includes two key features: a) an architecture based upon the use of a single PE, as illustrated in Figure 1, which exploits partitioned memory to facilitate the parallel computation of the large double butterfly operation; and b) conflict-free and in-place parallel memory addressing schemes for both the data, as stored in the data memory (DM) – which needs to account for double buffering in order for real time operation to be achieved and maintained – and the twiddle factors, as stored in the coefficient memory (CM).

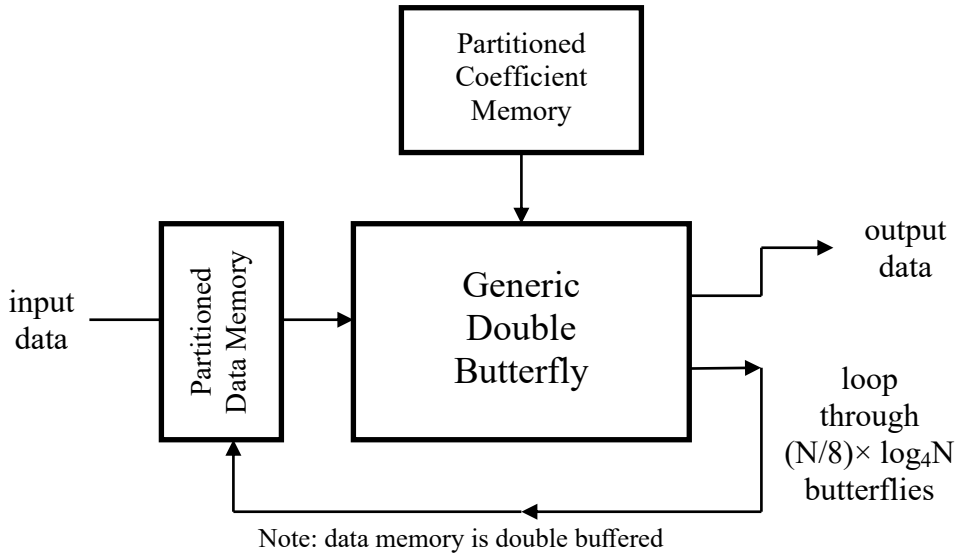

Figure 1: Single-PE Architecture for Computation of N-Point Regularized FHT

2.1 The Double Butterfly

These features of the RFHT enable the resources residing on the PE to be maximally utilized and each instance of the double butterfly – the computational engine producing eight outputs from each set of eight inputs – to produce a new output data set with each clock cycle.

The original design for the double butterfly required 12 multipliers and 22 adders for carrying out the associated operation, with: a) each eight sample (one or two samples per memory bank) data set being read/written in parallel from/to the partitioned DM, configurable as an array of eight memory banks; and b) the coefficients being read in parallel from the partitioned CM, configurable as an array of three one-level or multi-level LUTs (one per non-trivial twiddle factor). The addressing of the DM, over two consecutive clock cycles, enables all those samples required by the two corresponding instances of the double butterfly operation to be read from the DM, processed and then written back to the DM in a conflict - free and in - place manner at the rate of one eight-sample data set per clock cycle [10].

Being a radix-4 decimation-in-time (DIT) algorithm, the input data to the RFHT needs first to be reordered according to the dibit-reversal mapping (that is, involving the exchange of two bits at a time rather than just the one bit of the bit - reversal mapping), enabling the input data set to be then written to the DM with consecutive data samples being stored cyclically within consecutive memory banks, whilst on completion of the RFHT, the naturally ordered output data set may be read out from the DM with consecutive data samples being retrieved cyclically from consecutive memory banks [3,4].

2.2 Trading Off Memory Against Arithmetic

Three additional versions of the PE have been subsequently derived (as well as a CORDIC version not considered here) which enable the arithmetic component of the space - complexity to be traded off against the memory component, which varies according to the use of either one - level or multi - level LUTs for storing the coefficients [10]. The one-level LUT-based scheme, which is the standard approach, involves the sinusoidal and cosinusoidal components of the twiddle factors being typically read from a sampled version of the sine function with argument defined over a

single quadrant, namely from 0 up to π/2 radians.

The aim of the multi-level schemes – which, essentially, involves the exploitation of multiple small one-level LUTs – is to reduce the total memory requirement at the expense of increased arithmetic complexity – see results of recent study [13]. The two-level scheme, for example, comprises one *coarse-resolution* angular region catering for both the sine and cosine functions, covering 0 up to π/2 radians, and one *fine-resolution* angular region for each of the sine and cosine functions, covering 0 up to π/2L radians, where the optimal choice for L (which is the length of each one-level LUT) can be shown to be equal to $\sqrt{N}/2$, where N is the length of the transform to be computed. The required twiddle factors may then be obtained from the contents of the two-level LUT through the application of the standard trigonometric identities

$$\cos(\theta+\phi)=\cos(\theta)\times\cos(\phi)-\sin(\theta)\times\sin(\phi) \quad (7)$$

$$\& \quad \sin(\theta+\phi)=\sin(\theta)\times\cos(\phi)+\cos(\theta)\times\sin(\phi) \quad (8)$$

where θ corresponds to the angle defined over the coarse-resolution angular region and ϕ to the angle defined over the fine - resolution angular region – see the simplified illustration of Figure 2 for the decomposition of the cosine function into coarse-resolution and fine-resolution angular regions, each of length 4.

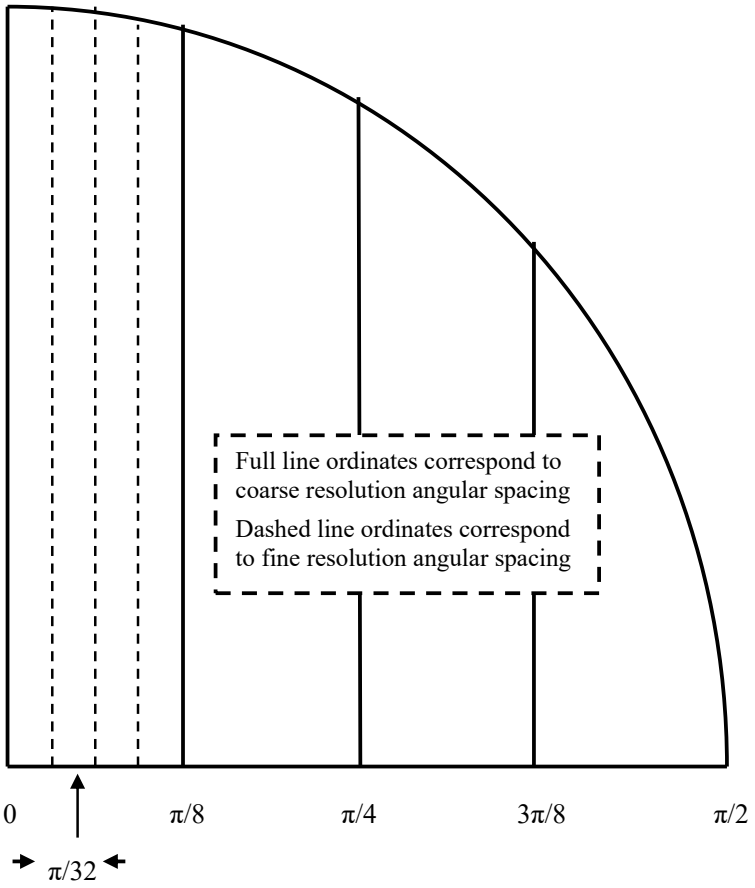

Figure 2: Decomposition of Single Quadrant of Cosine Function Into Coarse-Resolution and Fine-Resolution Angular Regions Using Single-Level Luts Each of Length 4

Thus, generalizing the results of the above two-level LUT-based scheme and expressing them in a concise mathematical form, the adoption of K - level LUTs may be said to result in a reduced memory requirement of $O(\sqrt[K]{N})$ words, as opposed to the O(N) requirement of the one - level LUTs, this reduction being obtained at the expense of increased addressing complexity through the need for the combined use of both coarse - resolution and fine - resolution LUTs – as discussed in greater detail in [13]. The equations representing the generalized form of the LUTs, where K is taken to be an arbitrary integer, may be straightforwardly obtained via the repeated application of the standard trigonometric identities of Equations 7 and 8.

2.3 Discussion

A theoretical performance/resource comparison of all four versions of the RFHT is provided in Table 1, with each version achieving $O(N\times\log N)$ latency which corresponds, in clock cycles, to the total number of double butterflies to be executed per transform, namely $(N/8)\times\log_4 N$. Note, however, that a small $O(1)$ increment to the latency is required to account for the pipelining of the coefficient generation process, when using a multi-level LUT-

based scheme, as is required if real-time operation is to be achieved and maintained. An O(N) update period for each input/output data set is achieved for each solution which corresponds to an I/O rate of just one sample per clock cycle. The signal flow graph for the nine - multiplier version of the generic double butterfly is as illustrated in Figure 3.

Version of Solution	Arithmetic Complexity				Memory Requirement (words)		Time Complexity (clock cycles)
	Double Butterfly		Coefficient Generator		Data Memory (Double-Buffered)	Coefficient Memory	Update Time / Latency
	Multipliers	Adders	Multipliers	Adders			
I	12	22	0	0	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{1}{4} N = \frac{3}{4} N$	$\frac{1}{8} N \times \log_4 N$
II	9	25	0	6	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{1}{4} N = \frac{3}{4} N$	$\frac{1}{8} N \times \log_4 N$
III	12	22	12	18	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{3}{2} \sqrt{N} = \frac{9}{2} \sqrt{N}$	$\frac{1}{8} N \times \log_4 N$
IV	9	25	12	24	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{3}{2} \sqrt{N} = \frac{9}{2} \sqrt{N}$	$\frac{1}{8} N \times \log_4 N$

Note: single-level LUTs used for versions I and II & two-level LUTs used for versions III and IV

Table 1: Performance/Resource Comparison for Computation of N-Point Regularized FHT

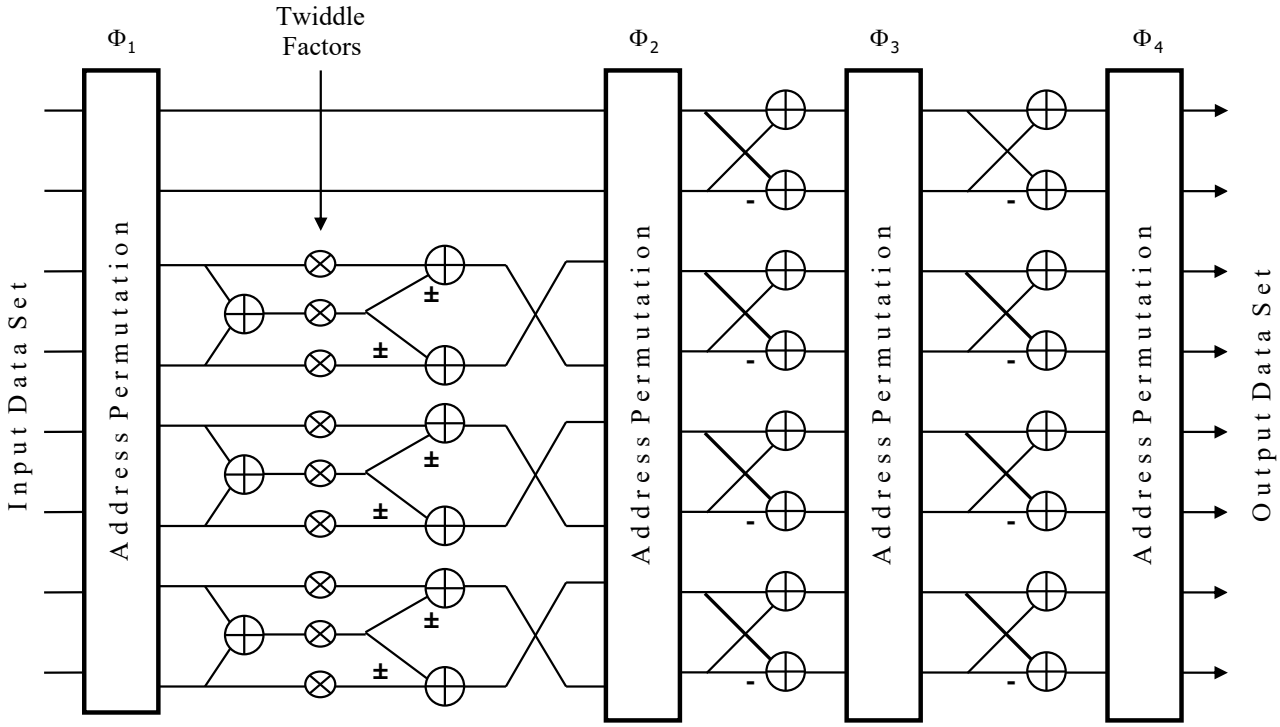

Figure 3: Signal Flow Graph for Nine-Multiplier Version of Generic Double Butterfly

3. Complexity of Single-PE and Dual-PE Solutions

Having described the key components of the regularized FHT, the space and time complexities of the single-PE and dual-PE solutions to the real-data DFT are now considered where the nine-multiplier version of the double butterfly is to be assumed and where maximum parallelism is to be exploited through the use of separate multi-level LUTs for each of the three non-trivial twiddle factors required for input to the double butterfly [14]. The time-complexity involves the derivation of the timing constraint that needs to be met if continuous real-time operation is to be achieved and maintained, whilst the space-complexity involves the derivation of the memory and arithmetic requirements, as expressed in terms of the amount of fast on-chip RAM and the number of fast multipliers required for its implementation, respectively. A wordlength of 18 bits, or 2.25 bytes, is to be assumed for the data storage, as this fits well with the sizes of block (and Ultra) RAM available with the current family of Xilinx FPGA devices – as proposed for the hypothetical implementations considered in Section 4 – with the wordlength for the coefficient storage being 27 bits, as dictated by the size of the fast multiplier [15]. The DM will need to be of dual-port type for handling both read and write operations, as required for its repeated updating, whilst the CM need only be of single-port type for handling the read-only operations associated with accessing the contents of the LUTs.

3.1 Timing Constraints

The latency, denoted T_L, of the RFHT-based PE for the case of N input/output samples is given by

$$T_L \approx \left(N/8\right) \times \log_4 N \tag{9}$$

clock cycles, whilst the double-buffered DM is updated with a new N-point data set every update period of N clock cycles, given that the transfer of data from the external source to the fast on-chip RAM is assumed to be carried out at the rate of one sample per clock cycle. Therefore, the single-PE solution – where a PE is taken to consist of a single RFHT module – achieves a continuous real-time performance when

$$T_L < N \tag{10}$$

clock cycles, which occurs when

$$N \leq 4^7 = 16{,}384 \tag{11}$$

The operation of the dual-PE solution is defined by having one PE process all the even-addressed input data sets and the other PE all the odd-addressed data sets, as illustrated in Figure 4, with each input data set comprising N real-valued samples. In this way, each PE is able to process a new N-point data set every 2N clock cycles with the dual-PE solution thereby able to produce a new output data set of N samples every N clock cycles. As a result, the dual-PE solution is able to achieve a continuous real-time performance when

$$T_L < 2N \tag{12}$$

clock cycles, which occurs when

$$4^8 = 65{,}536 \leq N \leq 4^{15} = 1{,}073{,}741{,}824 \tag{13}$$

or up to one billion samples.

Note that if the latency is sufficiently lower than the update rate (which is N clock cycles for the single-PE solution and 2N clock cycles for the dual-PE solution), then there might well be sufficient down-time available for carrying out those additional post-DHT functions, such as the Hartley-space to Fourier-space conversion or the PSD estimation, before the next input data set is available for processing. Also, assuming the input data and its subsequent processing to be fixed-point in nature, the down-time may also be used to deal with the incorporation of a suitable scaling strategy – as will be briefly discussed in Section 5 – which will involve an additional overhead, in terms of latency, following the completion of each stage of double butterflies.

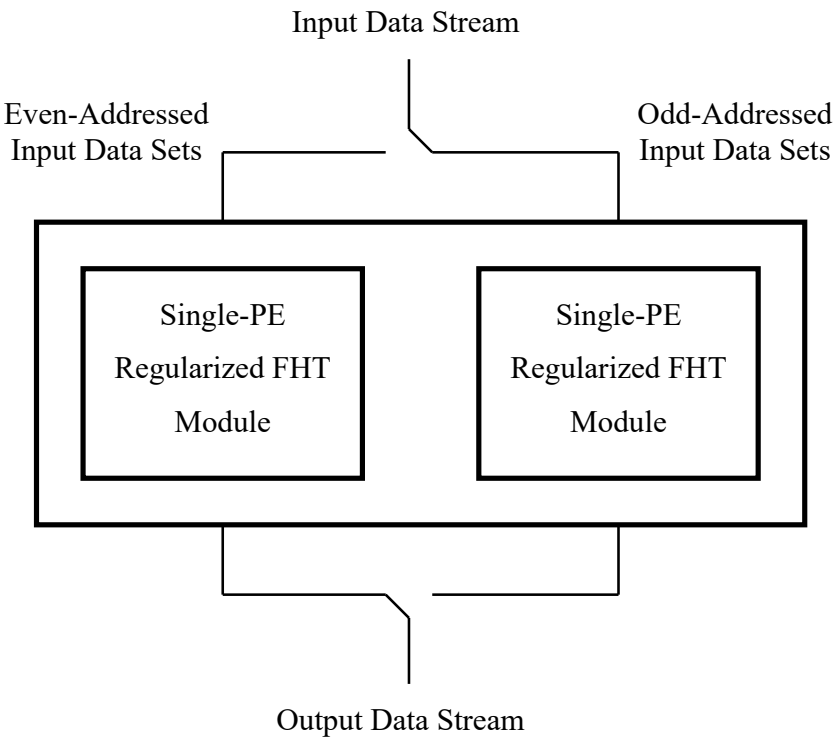

Figure 4: Dual-PE Architecture for Computation of Real-Data DFT Via Regularized FHT

3.2 Memory Requirement

The DM requirement, per PE, is given by

$$\text{DM} = 2\text{N words} = \tfrac{9}{2} \times \text{N bytes} \tag{14}$$

this figure accounting for the double-buffering of the input data whereby one half of the DM is being updated with new data whilst the data from the other half is being processed.

To determine the CM requirement, suppose that the figures are firstly to be based upon the adoption of two-level LUTs (as might be appropriate for long transforms of order one million samples, say, with each two-level LUT consisting of three one-level LUTs [13]) for each of the three non-trivial twiddle factors (in order to facilitate their simultaneous access) required by the highly-parallel double butterfly, with the length of each one-level LUT being of $O\left(\sqrt{N}\right)$. Then the CM requirement, per PE, is given by

$$\begin{aligned} \text{CM}^{(2)} &= 3\times(\,3\times\sqrt{\text{N}/4}\,) = \tfrac{9}{2}\times\sqrt{\text{N}}\ \text{words} \\ &= \tfrac{81}{8}\times\sqrt{\text{N}}\ \text{bytes} \end{aligned} \tag{15}$$

with the total memory requirement for the single-PE solution, denoted $M_S^{(2)}$, given by

$$\begin{aligned} M_S^{(2)} &= (2\times\text{N}) + \left(\tfrac{9}{2}\times\sqrt{\text{N}}\right)\ \text{words} \\ &= \left(\tfrac{9}{2}\times\text{N}\right) + \left(\tfrac{81}{8}\times\sqrt{\text{N}}\right)\ \text{bytes} \end{aligned} \tag{16}$$

and the total memory requirement for the dual-PE solution, denoted $M_D^{(2)}$, given by

$$\begin{aligned} M_D^{(2)} &= (4\times\text{N}) + \left(9\times\sqrt{\text{N}}\right)\ \text{words} \\ &= (9\times\text{N}) + \left(\tfrac{81}{4}\times\sqrt{\text{N}}\right)\ \text{bytes} \end{aligned} \tag{17}$$

Suppose now that the CM figures are to be based upon the adoption of three-level LUTs (as might be appropriate for ultra-long transforms of order one billion samples, say, with each three-level LUT consisting of five one-level LUTs [13]) for each of the three non-trivial twiddle factors (in order to facilitate their simultaneous access) required by the highly parallel double butterfly, with the length of each one-level LUT being of $O\left(\sqrt[3]{N}\right)$. Then the CM requirement, per PE, is given by

$$\begin{aligned} \text{CM}^{(3)} &= 3\times(\,5\times\sqrt[3]{\text{N}/4}\,) = 15\times\sqrt[3]{\text{N}/4}\ \text{words} \\ &= \tfrac{135}{4}\times\sqrt[3]{\text{N}/4}\ \text{bytes} \end{aligned} \tag{18}$$

with the total memory requirement for the single-PE solution, denoted $M_S^{(3)}$, given by

$$\begin{aligned} M_S^{(3)} &= (2\times\text{N}) + \left(15\times\sqrt[3]{\text{N}/4}\right)\ \text{words} \\ &= \left(\tfrac{9}{2}\times\text{N}\right) + \left(\tfrac{135}{4}\times\sqrt[3]{\text{N}/4}\right)\ \text{bytes} \end{aligned} \tag{19}$$

and the total memory requirement for the dual-PE solution, denoted $M_D^{(3)}$, given by

$$M_D^{(3)} = (4 \times N) + \left(30 \times \sqrt[3]{N/4}\right) \text{ words}$$
$$= (9 \times N) + \left(135/2 \times \sqrt[3]{N/4}\right) \text{ bytes} \quad (20)$$

Note that the superscripts, in each case, simply refer to the size of the multi-level LUT-based coefficient generation scheme to be adopted, which is set to either 2 for the two-level case or 3 for the three-level case.

3.3 Arithmetic Requirement

With the adoption of the two-level LUT-based coefficient generation scheme, the twiddle factors require three sets of arithmetic, per PE, each of: 4 multipliers and 8 adders, yielding a total per PE, denoted $A_{TF}^{(2)}$, as given by

$$A_{TF}^{(2)} = 12 \text{ multipliers \& } 24 \text{ adders} \quad (21)$$

whilst the double butterfly computation requires a total per PE, denoted $A_{DB}^{(2)}$, as given by

$$A_{DB}^{(2)} = 9 \text{ multipliers \& } 25 \text{ adders} \quad (22)$$

Thus, the total arithmetic requirement for the single-PE solution, denoted $A_S^{(2)}$, is given by

$$A_S^{(2)} = 21 \text{ multipliers \& } 49 \text{ adders} \quad (23)$$

whilst the total arithmetic requirement for the dual-PE solution, denoted $A_D^{(2)}$, is given by

$$A_D^{(2)} = 42 \text{ multipliers \& } 98 \text{ adders} \quad (24)$$

Similarly, with the adoption of the three-level LUT-based coefficient generation scheme, the twiddle factors require three sets of arithmetic, per PE, each of: 8 multipliers and 16 adders, yielding a total per PE, denoted $A_{TF}^{(3)}$, as given by

$$A_{TF}^{(3)} = 24 \text{ multipliers \& } 48 \text{ adders} \quad (25)$$

whilst the double butterfly computation requires a total per PE, denoted $A_{DB}^{(3)}$, as given by

$$A_{DB}^{(3)} = 9 \text{ multipliers \& } 25 \text{ adders} \quad (26)$$

Thus, the total arithmetic requirement for the single-PE solution, denoted $A_S^{(3)}$, is given by

$$A_S^{(3)} = 33 \text{ multipliers \& } 73 \text{ adders} \quad (27)$$

whilst the total arithmetic requirement for the dual-PE solution, denoted $A_D^{(3)}$, is given by

$$A_D^{(3)} = 66 \text{ multipliers \& } 146 \text{ adders} \quad (28)$$

Note, as before, that the superscripts, in each case, simply refer to the size of the multi-level LUT based coefficient generation scheme to be adopted, which is set to either 2 for the two-level case or 3 for the three-level case.

3.4 Discussion

The total resource requirements for the one million-point and one billion-point transforms are as outlined in Table 2, from which it is evident that the only change in the two sets of figures lies in the memory requirement arising primarily from the three orders of magnitude difference in the sizes of the input/output data sets. The difference in the memory requirements of the two-level LUTs (for one million-point transform) and three-level LUTs (for one billion-point transform) for the coefficient storage is minimal in comparison. The resource requirements for the 4^5-point (or one thousand-point) transform – which requires a single-PE solution and uses the simple single-quadrant scheme for the coefficient generation and storage – are provided purely for the purposes of comparison [13].

Size of Transform	Arithmetic Complexity				Memory Requirement (words)		Time Complexity (update periods)
	Double Butterflies		Coefficient Generators		Data Memory (Double-Buffered)	Coefficient Memory	Update Time / Latency
	Multipliers	Adders	Multipliers	Adders			
$N = 4^5$ (one thousand)	9	25	0	6	$2 \times N = 2 \times 4^5$	$3 \times \frac{1}{4} \times N = 3 \times 4^4$	$\frac{1}{8} \times \log_4 N = \frac{5}{8}$
$N = 4^{10}$ (one million)	$2 \times 9 = 18$	$2 \times 25 = 50$	$2 \times 12 = 24$	$2 \times 24 = 48$	$2 \times (2 \times N) = 4^{11}$	$2 \times \left(3 \times \frac{3}{2} \times \sqrt{N}\right) = \frac{9}{4} \times 4^6$	$\frac{1}{8} \times \log_4 N = \frac{5}{4}$
$N = 4^{15}$ (one billion)	$2 \times 9 = 18$	$2 \times 25 = 50$	$2 \times 24 = 48$	$2 \times 48 = 96$	$2 \times (2 \times N) = 4^{16}$	$2 \times \left(3 \times 5 \times \sqrt[3]{N/4}\right) \approx \frac{6}{5} \times 4^7$	$\frac{1}{8} \times \log_4 N = \frac{15}{8}$

Note: 1) two-level LUTs used for one million-point transform & three-level LUTs used for one billion-point transform
2) one thousand-point transform requires single-PE solution & single-level LUT for single-quadrant storage scheme

Table 2: Space and Time Complexities of Dual-PE Solutions to One Million-Point and One Billion-Point Transforms

Note that with the adoption of a clock frequency of 233 MHz, say, the dual-PE solution would be able to produce a new one million-point output data set approximately every 4.29 ms (or, equivalently, 233 new one million-point output data sets every second). The throughput of the single-PE solution is achieved primarily through the eight-fold parallelism attained by the double butterfly, whilst the throughput of the dual-PE solution is further enhanced through the simultaneous operation of the two PEs.

4. Mapping of Long Transforms onto FPGA

This section provides a brief discussion of two hypothetical implementations which illustrate how the dual-PE solution to the computation of two long real-data transforms might each be mapped onto a single commercially-available FPGA device using only fast on-chip RAM for the data and coefficient storage – as measured in binary KBytes and MBytes.

4.1 One Million-Point Transform

Suppose, for our first example, that the real-data transform of interest is of length

$$N = 4^{10} \tag{29}$$

which equates to one million samples. Then from the timing constraints of Section 3.1, the solution requires the adoption of the dual-PE architecture in order to maintain continuous real-time operation as the associated latency, T_L, is given from Equation 9 by

$$T_L \approx \frac{5}{4} N \tag{30}$$

clock cycles, which is clearly in excess of the update period of N clock cycles for each input data set, but less than twice the update period of 2N clock cycles.

The proposed computing device for its hypothetical implementation is taken to be a Virtex UltraScale (model VU125) FPGA, which has a total memory capacity of approximately 13 MBytes of RAM – this comprising 11.75 MBytes of block RAM and 1.25 MBytes of distributed RAM – and an arithmetic provision of 1200 fast multipliers, each of size (18 bit)×(27-bit). Then the two sets of double-buffered DM data, where each set involves $2 \times 4^{10} \times 2.25$ bytes of input data, when added to the two sets of CM data, where each set is based upon the adoption of a two-level LUT-based scheme for the coefficient generation and storage and involves approximately 10.2 KBytes of data, yields a total memory requirement of approximately 9 MBytes. This equates to the utilization, denoted U_M, of the block RAM available on the chosen device, of

$$U_M \approx 77\% \tag{31}$$

leaving the distributed RAM to cater for additional processing tasks needing to be performed on the device. The corresponding arithmetic requirement, on the other hand, is minimal, involving the use of just 42 of the 1200 fast multipliers available on the device which equates to the utilization, denoted U_A, of just

$$U_A \approx 3.5\% \tag{32}$$

whilst the 86 associated adders may be easily and efficiently implemented in silicon through the use of programmable logic.

4.2 Four Million-Point Transform

Suppose, for our second example, that the real-data transform of interest is of length

$$N = 4^{11} \tag{33}$$

which equates to four million samples. Then, as with the first example, the solution requires the adoption of the dual-PE architecture in order to maintain continuous real-time operation as the associated latency, T_L, is given from Equation 9 by

$$T_L \approx {}^{11}\!/_{8}\, N \tag{34}$$

clock cycles, which is clearly in excess of the update period of N clock cycles for each input data set, but less than twice the update period of 2N clock cycles.

The proposed computing device for its hypothetical implementation is taken to be a Virtex UltraScale+ (model VU9P) FPGA, which has a total memory capacity of approximately 46.25 MBytes of RAM – this comprising 9.5 MBytes of block RAM, 33.75 MBytes of Ultra RAM and 3 MBytes of distributed RAM – and an arithmetic provision of 6840 fast multipliers, each of size (18 bit)×(27-bit). Then the two sets of double-buffered DM data, where each set involves $2 \times 4^{11} \times 2.25$ bytes of input data, when added to the two sets of CM data, where each set is again based upon the adoption of a two-level LUT-based scheme for the coefficient generation and storage and involves approximately 20.25 KBytes of data, yields a total memory requirement of approximately 36 MBytes. This equates to the utilization, denoted U_M, of the block+Ultra RAM available on the chosen device, of

$$U_M \approx 83\% \tag{35}$$

(assuming that the Ultra RAM is able to be effectively utilized) leaving the distributed RAM to cater for additional processing tasks needing to be performed on the device. The corresponding arithmetic requirement, on the other hand, is again minimal, involving the use of just 42 of the 6840 fast multipliers available on the device which equates to the utilization, denoted U_A, of just

$$U_A \approx 0.6\% \tag{36}$$

whilst the 86 associated adders may again be easily and efficiently implemented in silicon through the use of programmable logic.

4.3 Discussion

Note that for the dual-PE solution to the one million-point transform the latency is given by just 5/4 times the update period (of one million clock cycles). As a result, each PE – which includes its CM and double-buffered DM – is actually utilized for just 5/8 of the available processing time, although the two PEs are actually operating simultaneously for just 1/4 of the available processing time. The resulting down-time for each PE – which corresponds to the up-time of the remaining PE and therefore accounts for 3/8 of the available processing time – could of course be used for carrying out other tasks, as is briefly discussed in Section 3.1, or alternatively left inactive to enable the associated power consumption – which with the large memory requirement of the memory-based dual-PE architecture could be significant – to be kept to a minimum.

Note also that with the FPGA implementation of the one million-point transform discussed in which assumes complex-valued rather than real-valued input data (which might typically involve the need for a digital down conversion (DDC) process to be carried out prior to the execution of the complex-data FFT, adding to the overall timing/resource requirements), similar utilization figures are achieved as to those derived above but using the smaller VU095 model, which is one lower in the UltraScale family to the VU125 model and possesses approximately 2/3 of its memory and arithmetic resources [16]. This reduction in resource requirements is only achieved, however, through the use of a highly-optimized complex design whose advantages/disadvantages need to be carefully weighed against those of the much simpler memory-based designs, such as that of the RFHT-based dual-PE architecture adopted here, which from the timing constraints of Section 3.1 caters for transforms possessing lengths ranging from 4^8 up to 4^{15}, with only the memory capacity needing to be modified from one application (or transform length) to another, as demonstrated here for both the one and four million-point examples.

Thus, with the appropriate choice of computing device, possessing sufficient fast memory, it is theoretically possible for both the one and four million-point real-data transforms to each be implemented in a very straightforward manner using only fast on-chip RAM and a trivial quantity of fast multipliers. The current situation for the billion-point transform is somewhat different, however, as the limitations on the availability of fast on-chip memory with existing silicon-based FPGA or application-specific integrated circuit (ASIC) technologies would necessitate the reliance on the use of slower off-chip memory which would severely degrade the potential for obtaining a solution capable of achieving continuous real-time operation. The most likely way forward, at present, with transforms of this length, is via the adoption of sparse FFT techniques which can perform well using limited quantities of suitably randomized data provided the signal being processed comprises a limited number of significant spectral components [17,18].

5. Scaling Strategy for Fixed-Point Processing

With the adoption of fixed-point processing – as is to be assumed here – a suitable scaling strategy would be needed in order to prevent arithmetic overflow from occurring, as each instance of the PE's large double butterfly may incur up to a maximum of three bits of word growth (one bit for each stage of adders following the fast multipliers). Such a situation needs to be accounted for within the processing in order to avoid a possible loss of precision – and thus of signal-to-noise ratio (SNR) – through the loss of one or more of the data's most significant bits.

The best way to achieve this will be by applying an optimal or 'conditional' scaling strategy in the form of the block floating-point scheme to the output of each stage of double butterflies, with the resulting scaling factor being then applied, in each case, to the input data for the succeeding stage of double butterflies. This ensures that the scaling factor obtained for each stage of double butterflies is optimized and that any magnification incurred during the last stage of double butterflies is not scaled out of the results [10].

Note that although an optimal scheme such as this comes at a

computational cost (including the marginally increased latency of each stage of double butterflies), it is conceptually simple to apply with the memory-based architecture of the RFHT. However, this is not the case with the familiar pipelined FFT architectures [11], which must rely upon the use of a sub-optimal or 'unconditional' scaling strategy, whereby the data is typically over-scaled in order to prevent arithmetic overflow from occurring, resulting in reduced SNR when compared to that achieved with the memory-based approach.

6. Summary and Conclusions

With the current trend in large scale, big data applications, there is an increasing need for the design and efficient implementation of long to ultra-long Fourier-based transform algorithms, such as with FFTs where the transform length varies from long up to ultra-long. This paper has shown that in order to implement such algorithms when using the memory-based architecture of the RFHT, a timing constraint (and hence transform size limitation) due to the combined effects of the update period and the I/O rate needs to be overcome and the formidable data and coefficient memory requirement minimized if continuous real-time operation, using suitably defined parallel computing equipment, is to be achieved and maintained.

With this in mind and with a PE defined as comprising one complete RFHT module – which has the attraction of being resource-efficient, scalable and highly parallel (yielding eight-fold parallelism) – it has been demonstrated how the design of a scalable, dual-PE architecture may be derived as a simple extension of the single-PE case – thus possessing a number of attractive properties, as held by the RFHT, but not by pipelined RFFT implementations – this being achieved in such a way that the transform size limitation resulting from the timing constraint may be effectively overcome. When combined with the use of memory-efficient multi-level LUT-based schemes (a two-level scheme being adopted here for both the one and four million-point cases) for the coefficient generation and storage, this offered the 'potential' for achieving and maintaining the parallel computation of real-data transforms, in a continuous real-time fashion, for transform lengths of up to one billion.

Finally, the study concluded with a brief description of two hypothetical implementations of real-time parallel solutions to the real-data DFT, these illustrating in particular how the dual-PE solutions for the parallel computation of both the one and four million-point real-data transforms may each be mapped onto a single commercially-available FPGA device, with each implementation using only fast on-chip RAM for the data and coefficient storage, so as to achieve and maintain continuous real-time operation.

The author declares **No Conflicts of Interest** relating to the production of this paper.

References

1. Birkhoff, G., & Mac Lane, S. (2017). *A survey of modern algebra. AK Peters/CRC Press.*
2. McClellen, J. H., & Rader, C. M. (1979). *Number theory in digital signal processing.* Prentice Hall Professional Technical Reference.
3. Brigham, E. O., & Yuen, C. K. (1978). The fast Fourier transform. *IEEE Transactions on Systems, Man, and Cybernetics, 8*(2), 146-146.
4. Chu, E., & George, A. (1999). *Inside the FFT black box: serial and parallel fast Fourier transform algorithms.* CRC press.
5. Garrido, M., Unnikrishnan, N. K., & Parhi, K. K. (2017). A serial commutator fast Fourier transform architecture for real-valued signals. *IEEE Transactions on Circuits and Systems II: Express Briefs, 65*(11), 1693-1697.
6. Park, S., & Jeon, D. (2020, October). A modified serial commutator architecture for real-valued fast Fourier transform. In *2020 IEEE Workshop on Signal Processing Systems* (SiPS) (pp. 1-6). IEEE.
7. Eleftheriadis, C., & Karakonstantis, G. (2022). Energy-efficient fast Fourier transform for real-valued applications. *IEEE Transactions on Circuits and Systems II: Express Briefs, 69*(5), 2458-2462.
8. Akl, S. G. (1989). *The design and analysis of parallel algorithms*. Prentice-Hall, Inc.
9. Bracewell, R. N. (Ed.). (1986). *The Hartley transform.* Oxford University Press, Inc.
10. Jones, K. (2022). The Regularized Fast Hartley Transform: Low-Complexity Parallel Computation of FHT in One and Multiple Dimensions, 2nd Edition, Springer.
11. Jones, K. (2023, July). A Comparison of Two Recent Approaches, Exploiting Pipelined FFT and Memory-Based FHT Architectures, for Resource-Efficient Parallel Computation of Real-Data DFT, Journal of Applied Science and Technology (Open Access), Vol. 1, No. 2
12. Maxfield, C. (2004). *The design warrior's guide to FPGAs: devices, tools and flows*. Elsevier.
13. Jones, K. (2024, July). Schemes for Resource-Efficient Generation of Twiddle Factors for Fixed-Radix FFT Algorithms, Engineering (Open Access), Vol. 2, No. 3.
14. Harel, D., & Feldman, Y. A. (2004). *Algorithmics: The spirit of computing*. Pearson Education.
15. https://www.amd.com/en.html
16. Kanders, H., Mellqvist, T., Garrido, M., Palmkvist, K., & Gustafsson, O. (2019). A 1 million-point FFT on a single FPGA. *IEEE Transactions on Circuits and Systems I: Regular Papers, 66*(10), 3863-3873.
17. Hassanieh, H., Indyk, P., Katabi, D., & Price, E. (2012, January). Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms* (pp. 1183-1194). Society for Industrial and Applied Mathematics.
18. K. Jones (2023, August). Design for Resource-Efficient Parallel Solution to Real-Data Sparse FFT. Journal of Applied Science and Technology (Open Access), Vol. 1, No. 2.