Research Article

Open Access Journal of Applied Science and Technology

A Comparison of Two Recent Approaches, Exploiting Pipelined FFT and Memory-Based FHT Architectures, for Resource-Efficient Parallel Computation of Real-Data DFT

Dr. Keith Jones*

Consultant Mathematician (Retired), UK

***Corresponding Author**
Keith John Jones, Consultant Mathematician, Weymouth, Dorset, UK.

Submitted: 2023, June 13; **Accepted:** 2023, July 05; **Published:** 2023, July 14

Citation: Jones, K. J. (2023). A Comparison of Two Recent Approaches, Exploiting Pipelined FFT and memory-based FHT Architectures, for Resource-Efficient Parallel Computation of Real-Data DFT. *OA J Applied Sci Technol, 1*(2), 46-55.

Abstract

This paper provides a comparison and assessment of both the performance and the capabilities of two recently developed approaches to the problem of computing the real-data DFT. The approaches exploit pipelined FFT and memory-based FHT architectures and aim to produce resource-efficient parallel solutions as required for use in resource and power constrained environments. The FFT-based solutions involve multi PE pipelined designs, geared to streaming (or serial) operation, that exploit the conjugate symmetric nature of the real-data DFT spectrum. The FHT based solutions, which are suitably optimized versions of the regularized FHT, are geared to batch (or block-based) operation and involve a memory-based single-PE design that exploits partitioned memory in order to achieve eight fold parallelism within the PE. After outlining the performance objectives of each approach the study highlights the key properties and relative advantages/disadvantages of each, showing how the arithmetic complexity may be traded off against the memory requirement in order to optimize the use of the available silicon resources on the target computing device and to meet the appropriate timing objectives or constraints. A number of additional design issues not addressed with recent real-data FFT research – in particular, those relating to design simplicity, regularity and scalability – are also discussed which enable a more comprehensive assessment of a solution's capabilities.

Keywords: FFT, FHT, mRSC, RFFT, RFHT, RSC, RSDF

1. Introduction

Many real-world spectrum analysis problems, such as those involving biomedical signals, require the computation of the real-data discrete Fourier transform (DFT) a unitary transform given by

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

where the transform kernel derives from the term

$$W_N = \exp(-i2\pi / N), \qquad (2)$$

the N^{th} complex primitive root of unity [3,5,15]. The transform maps elements of the linear space of real valued N-tuples, R^N, to elements of its complex-valued counterpart, C^N, and when carried out in hardware is conventionally achieved via a real-from-complex strategy using a complex-data version of the ubiquitous fast Fourier transform (FFT) [5,15]. Such algorithms are typically derived by exploiting the property of symmetry, whether it exists just in the transform kernel or, in certain circumstances, in the input data and/or output data as well.

The reason for choosing the computational domain of real-data problems such as this to be C^N, rather than R^N, is due in part to the fact that computing equipment manufacturers have invested so heavily in producing digital signal processing (DSP) devices built around the design of the complex-data fast multiplier and accumulator (MAC), an arithmetic unit ideally suited to the implementation of the complex-data radix-2 butterfly, a computational unit used by the familiar class of radix-2 FFT algorithms [5,15]. The net result is that the problem of the real-data DFT is effectively being modified so as to match an existing complex-data solution rather than a solution being sought that matches the actual problem needing to be solved. Many genuine real data solutions, referred to as real-data FFTs (or RFFTs), have been developed to address this problem, with a few of the most recent and most promising in terms of possessing low size, weight and power (SWAP) – as required for use in resource and power constrained environments as with applications typified by that of mobile communications being described in this paper [2,6,8-11,17-19].

Three such RFFT designs are considered, each possessing a pipelined architecture involving the use of a separate processing element (PE) for each stage of butterflies, that offer genuine real-data solutions by exploiting the conjugate-symmetric

nature of the real-data DFT spectrum [1,8,10,17]. The three algorithms considered are those based upon: 1) the real-valued serial commutator (RSC) architecture, 2) the modified RSC (mRSC) architecture and 3) the real-valued single path delay feedback (RSDF) architecture, with all three algorithms being of decimation-in-time (DIT) type [5,8,10,15,17]. The first two of these architectures are derived from the serial commutator (SC) architecture which was designed to maximize resource utilization by converting each stage of the pipeline to a half-butterfly operation involving real-valued rather than complex-valued inputs, whilst the third is derived from the single path delay feedback (SDF) architecture which was designed using delay lines, implemented with memory and shift registers, to reorder the data at each stage of the pipeline [9,10,12]. The resulting solutions for these three architectures are each geared to streaming (or serial) operation whereby individual samples (assumed hereafter to be real-valued) or sets of samples are processed as they are acquired with different levels of parallelism being exploited within the PEs by each in order to achieve their respective performance objectives.

An alternative approach is also described which is based upon the use of the regularized fast Hartley transform (FHT) (or RFHT), which involves a single-PE design for computing the real-valued discrete Hartley transform (DHT) [4,15], an orthogonal transform given by

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

where the input/output data sets both belong to R^N 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 [3,13,15]. The RFHT is also of DIT type and may also be used for computing the real-data DFT, given that the DFT and DHT output data sets as produced by Eqtns. 1 and 3 may be straightforwardly obtained, one from the other, via the following:

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

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

The RFHT design possesses a memory-based architecture which involves double-buffering of the input data [18]. The solution is geared to batch (or block-based) operation whereby the entire input data set must be available before processing can commence with the required computational throughput being achieved by using a single highly parallel PE built from a small quantity of silicon resources. The resulting design can thus be shown to possess low SWAP as well as a number of attractive properties that pipelined RFFT solutions typically do not.

The design constraints and performance objectives of each of these two approaches – namely, those based upon the use of either the pipelined RFFT or the memory-based RFHT – to the efficient computation of the real-data DFT are described together with a theoretical performance comparison of the resulting solutions as expressed by their space (arithmetic requirement, in terms of the required numbers of real multipliers and adders, together with the memory requirement, in terms of the required amount of dual-port random access memory (RAM)) and time (achievable latency/throughput) complexities. This includes a discussion of the key properties and relative advantages/disadvantages of each of the solutions together with a few illustrative examples for carrying out the real-data DFT of small to medium sized data sets. The aim with each approach, essentially, is to maximize the throughput per unit area of silicon, a metric that's more commonly known when assessing the silicon-based implementation of signal/image processing algorithms as the 'computational density' [15].

The analysis presented is not intended to be exhaustive in terms of the various RFFT and FHT solutions considered as a number of comprehensive studies have already been carried out in recent years (such as those discussed in) for dealing with the many RFFT variations [2,6,11,18,19]. Just a few key RFFT architectures are therefore considered, whilst the choice of FHT is limited to that of the regularized version of the FHT, namely the RFHT, in order to highlight the relative advantages/disadvantages of a memory-based solution (geared to batch operation) to that of a pipelined solution (geared to streaming operation). Thus, following this introductory section, brief accounts are given of the real-data FFT designs in Section 2 and that of the regularized FHT in Section 3. The various design constraints and performance objectives of the two approaches are then discussed in Section 4, this including a theoretical performance/resource comparison of the resulting solutions in terms of their respective arithmetic and memory requirements and achievable latency/throughput, together with a discussion of the key properties and relative advantages/disadvantages of each solution. The paper finishes with a brief summary and conclusions in Section 5.

2. Recent Developments with Real-Data FFT Design

The most commonly used FFT algorithms are those of the fixed-radix type and, in particular, those with a radix of 2 where typically the DIT version of the algorithm involves the use of bit-reversed inputs and naturally ordered outputs, whilst the decimation-in-frequency (DIF) version involves the use of naturally ordered inputs and bit reversed outputs [5,15]. An 8-point DIT FFT algorithm for the case of real-valued data is illustrated in Figure 1 whereby the two 4-point FFTs resulting from the decomposition each have real-valued inputs and 'odd' conjugate symmetric outputs, as expressed by

$$X[k] = -X^*[N-k]. \tag{7}$$

By utilizing the odd conjugate symmetry of the two 4-point FFTs it is possible to apply the same principle, recursively, to each of the half-length FFTs so as to break them down into even smaller 2-point FFTs and to reuse the same hardware units, without modification, to carry them out.

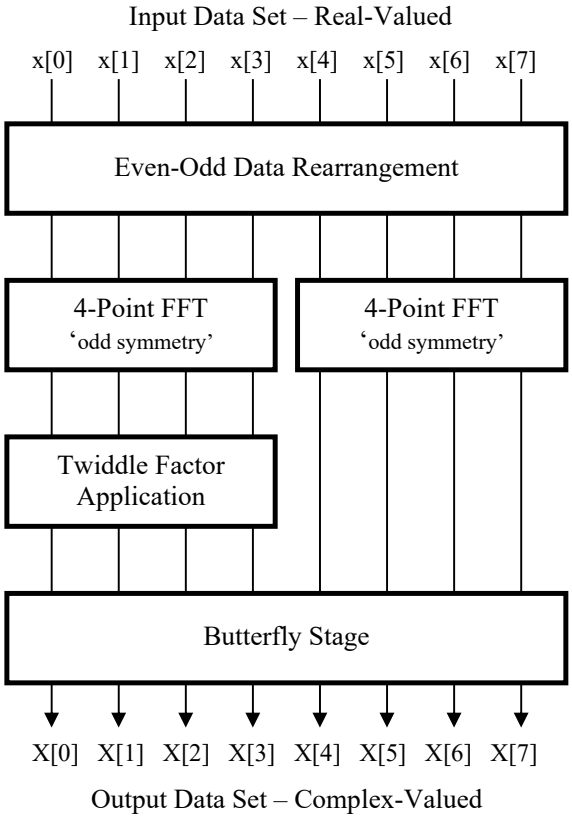

Figure 1: signal flow graph for 8-point DIT real-data FFT

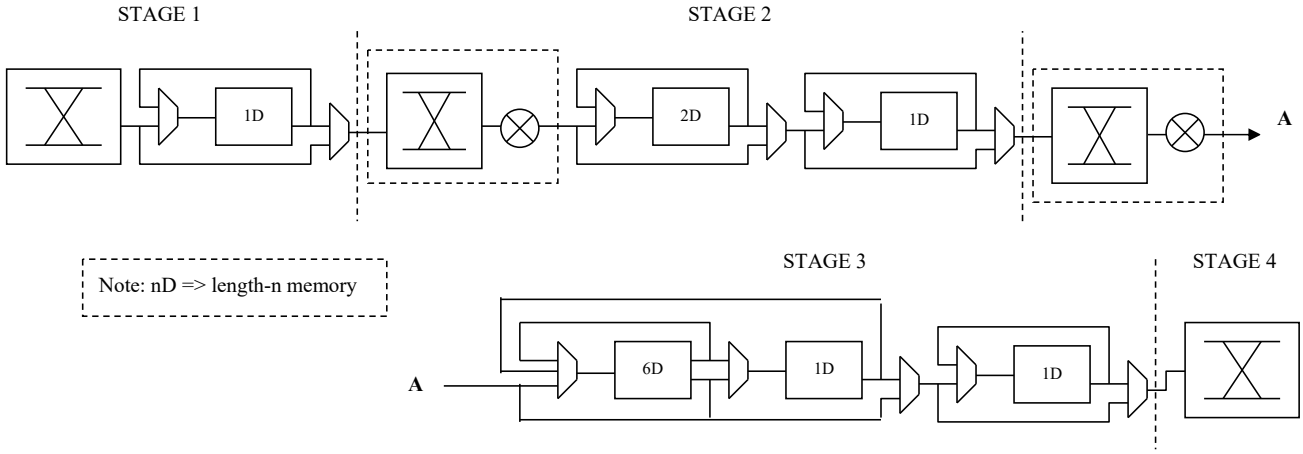

Figure 2: RSC architecture for 16-point radix-2 real-data FFT

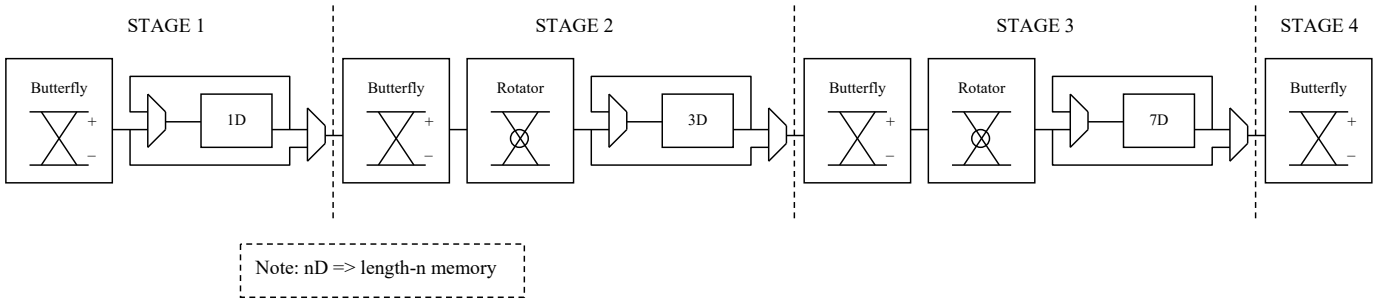

Figure 3: mRSC architecture for 16-point radix-2 real-data FFT

A number of DIT-type RFFTs have been derived in recent years from the SDF and SC architectures which, for a data set of size N, require that: 1) a memory of more than N words is available to enable the correct timing of the multiplications involving the trigonometric coefficients (or twiddle factors) at each stage of the processing, and 2) the resulting N scrambled outputs be suitably reordered. To meet these requirements and to further improve and optimize the use of the required hardware units, designs for the RSC architecture, as illustrated in Figure 2, and its subsequently improved version, the mRSC architecture, as illustrated in Figure 3, have been produced. Each comprises three types of hardware unit: 1) the butterfly, which performs an addition and a subtraction, 2) the rotator, which performs a complex multiplication involving trigonometric coefficients, and 3) the data management circuits, which are responsible for ensuring the correct timing of the resulting solution [10,17]. The RSC-based and mRSC based solutions each require $2.\log_2 N-2$ real adders and $\log_2 N-2$ real multipliers, whilst the RSC based solution requires $N+9.\log_2 N-19$ words of memory which the mRSC-based solution reduces to just $N+5.\log_2 N-9$ words. The latency of both solutions is given by $N+3.\log_2 N-8$ clock cycles whilst the outputs of each are left in scrambled form so that bit reversal reordering is required in each case for application to both the input and output data sets. Although such reordering of the outputs may be carried out as an additional stage of the pipeline it nevertheless results in an even more complex and irregular design.

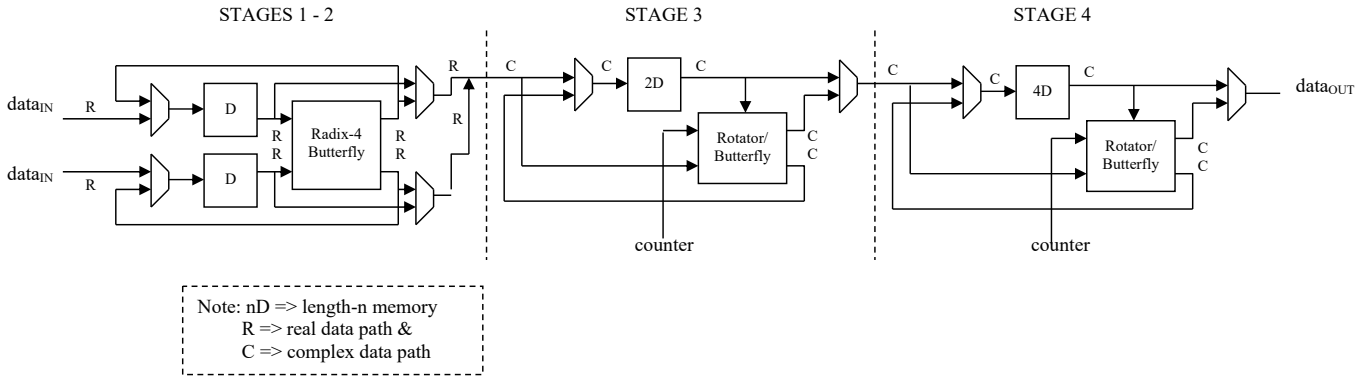

Figure 4: RSDF architecture for 16-point radix-2 real-data FFT

A more recent real-data FFT solution, which is derived from the SDF architecture and again of DIT type, is that possessing the RSDF architecture, as illustrated in Figure 4 [8]. The solution exploits the odd conjugate symmetry at each stage of processing by utilizing a specific DIT technique that requires bit reversed inputs but that produces naturally ordered outputs, rather than the scrambled outputs of the previous two solutions, thus avoiding the need for additional circuitry after the pipeline's final stage of butterflies. The solution enables the number of operations to be halved for each stage of processing as well as allowing for the production of two outputs at a time, rather than the one output of the previous two solutions, leading to an increased throughput. This increased throughput is only achieved, however, by increasing the I/O rate from one sample per clock cycle, as required by the RSC based and mRSC based solutions, to two samples per clock cycle – this, in turn, requiring the use of increased numbers of multipliers and adders.

With all three RFFT illustrated architectures, as given by Figures 2 to 4, a radix-2 solution is assumed so that a total of $\log_2 16 = 4$ stages are required for carrying out the processing in a pipelined fashion where each stage is assigned its own PE. With the RSDF architecture, however, the first two stages, which would each be expected to involve a radix-2 butterfly, are merged into a large single stage by performing a radix-4 butterfly rather than two radix-2 butterflies. It is this merging of stages which enables the processing of two real-valued inputs at a time, instead of just one – as is the case with the RSC and mRSC architectures. Also, it is evident that the RSC and mRSC architectures, although constructed from the same set of basic components, each require three distinct PE designs whilst the RSDF requires two. As a result, the three pipeline designs can be said to be neither strictly 'regular' (whereby the design involves the presence of large amounts of repetition and symmetry) nor 'scalable' (whereby the only design change needed in going from one size of data set to another is in the amount of memory required for storage of the input data and trigonometric coefficients). In fact, the consequence of the complexity reduction techniques utilized by these and other RFFT solutions tends to be a loss of regularity in the resulting design.

3. Computing Real-Data DFT via Regularized FHT

Given the ease with which the output data sets of the DHT and the real-data DFT may be obtained, one from the other, an FHT algorithm was sought which would enable attractive parallel solutions to be produced for the computation of the real-data DFT. The result was the RFHT which is a radix-4 DIT algorithm whose correctness of operation has already been proven in silicon with a fixed-point implementation using field-programmable gate array (FPGA) technology and with the storage of data and coefficients carried out with fast dual-port RAM [13,14,16].

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 5, 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 PE's internal data memory (PDM), and the trigonometric coefficients, as stored in the PE's coefficient memory (PCM). These features, when combined with pipelining and single instruction multiple-data (SIMD) processing techniques for the internal operation of the PE, enable the resources residing on the PE to be maximally utilized and each instance of the generic 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 [1].

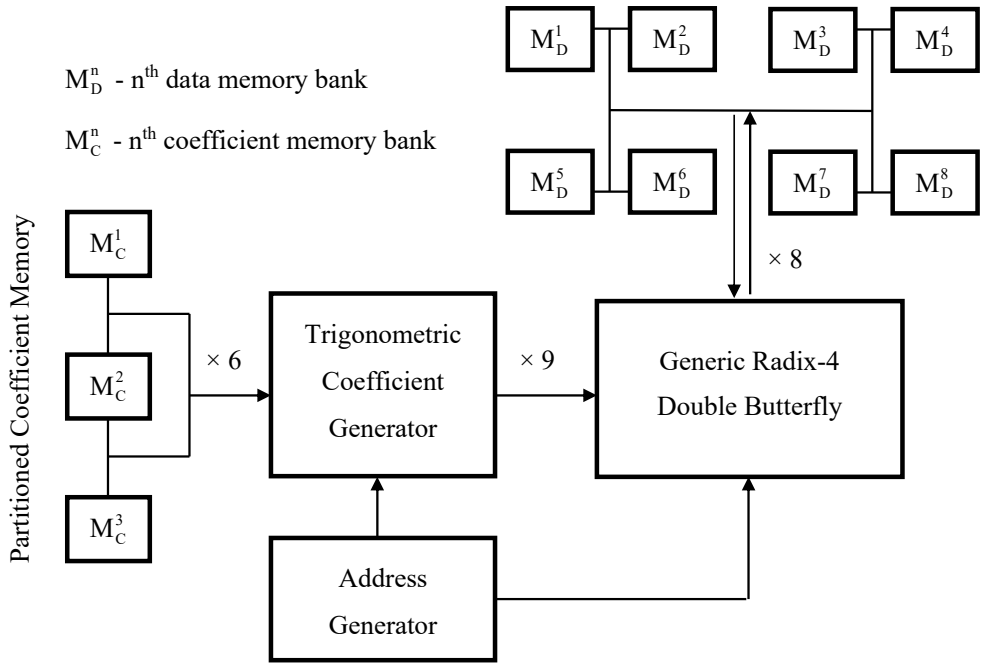

Figure 5: single-PE recursive architecture for regularized FHT

Note that the single-PE architecture may be regarded as being 'recursive' in the sense that the output from each stage of double butterflies is fed back as input to the succeeding stage, with the same set of computational units being used to perform the same set of operations on the input data to each and every stage. The original design required 12 multipliers and 22 adders for carrying out the double butterfly operation, with: a) each eight sample (one sample per memory bank) data set being read/written in parallel from/to the partitioned PDM, configurable as an array of eight memory banks; and b) the trigonometric coefficients being read in parallel from the partitioned PCM, configurable as an array of three one-level look up tables (LUTs), with each LUT storing a single quadrant of the sine function. The addressing of the PDM, 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 PDM, processed and then written back to the PDM in a conflict free and in place manner at the rate of one eight-sample data set per clock cycle.

Being a radix-4 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 the one bit of the bit -reversal mapping), enabling the input data set to be then written to the PDM 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 PDM with consecutive data samples being retrieved cyclically from consecutive memory banks.

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 two-level LUTs for the PCM [15]. The use of two-level LUTs results in a reduced memory requirement of $O(\sqrt{N})$ words, as opposed to the $O(\sqrt{N})$ 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. A theoretical performance/resource comparison of all four versions of the RFHT is provided in Table 1, with each version achieving an O(N. log N) latency which corresponds, in clock cycles, to the total number of double butterflies to be executed per transform, namely $N/8.\log_4 N$. An O(N) update period (or refresh rate) 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 illustrated in Figure 6.

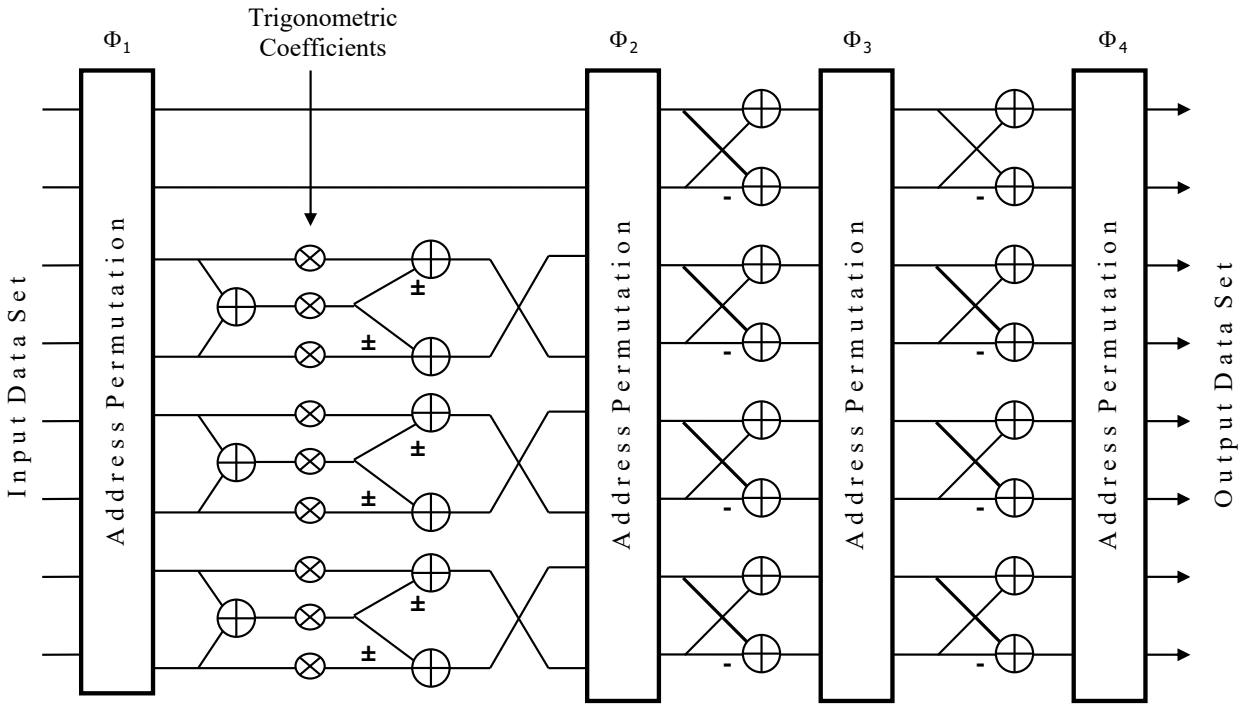

Figure 6: signal flow graph for nine-multiplier version of generic double butterfly

Version of Solution	Arithmetic Complexity				Memory Requirement (real words)		Time Complexity (clock cycles)	
	Processing Element		Coefficient Generator		Input/Output Data (Double-Buffered)	Trigonometric Coefficients	Latency	Update Period
	No of Real Multipliers	No of Real Adders	No of Real Multipliers	No of Real Adders				
I	12	22	0	0	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{1}{4} N = \frac{3}{4} N$	$\frac{1}{8} N.\log_4 N$	N
II	9	25	0	6	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{1}{4} N = \frac{3}{4} N$	$\frac{1}{8} N.\log_4 N$	N
III	12	22	7	8	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{1}{2} \sqrt{N} = \frac{3}{2} \sqrt{N}$	$\frac{1}{8} N.\log_4 N$	N
IV	9	25	7	14	$2 \times 8 \times \frac{1}{8} N = 2N$	$3 \times \frac{1}{2} \sqrt{N} = \frac{3}{2} \sqrt{N}$	$\frac{1}{8} N.\log_4 N$	N

Table 1: performance/resource comparison for fast multiplier versions of N-point regularized FHT

4. Comparative Analysis of Pipelined FFT and Memory-Based FHT Approaches

The relative advantages/disadvantages of the two approaches to the problem of computing the real-data DFT – namely, those based upon the RFFT, as dealt with in Section 2, and the RFHT, as dealt with in Section 3 – are now discussed in some detail.

4.1 A Few Desirable Properties – Bilateralism, Regularity and Scalability

The DHT, unlike the DFT, is equal to its own inverse – that is, a 'bilateral' transform. As a result, the RFHT may be effectively used for dealing with those real data problems involving the computation of both forward and inverse transforms – such as for carrying out the finite impulse response (FIR) filtering operation via the transform-domain approach, in a computationally efficient manner, by invoking the circular convolution theorem [5]. The RFFT algorithms, on the other hand, would need to be suitably modified in order to deal with the corresponding inverse transform. Thus, whilst each RFFT algorithm could be used for carrying out the fast computation of the forward DFT, where the input/output data sets are real-valued/complex-valued, a modified version would be required for carrying out the fast

computation of the inverse DFT, where the input/output data sets are complex-valued/real-valued – the complex-valued data set, in each case, being assumed to be conjugate symmetric.

The scalable RFHT design is also very simple and regular – as is evident from the illustrated architecture of Figure 5 – comprising one column of real multipliers operating in parallel, in SIMD fashion, together with three (or four, depending upon the chosen algorithm, as both nine-multiplier and twelve-multiplier versions of the PE are available) columns of adders with each column also operating in parallel. The RFFT designs, on the other hand, are evidently more complex and, as evident from the discussion in Section 2, less regular in the sense of requiring the design of two or more PE variations as opposed to the single PE design of the RFHT. Thus, the three RFFT solutions, although each possessing design commonalities amongst the multiple PEs (as evidenced from the illustrated architectures of Figures 2, 3 and 4) cannot, unlike the RFHT, be regarded as being either strictly regular or scalable. For real world commercial operation the questions of bilateralism and of design simplicity, regularity and scalability become critical issues to be considered in the design process as these may have a serious impact on the costs encountered by the organization involved for its various signal processing requirements (given that all such organizations will be ultimately constrained by costs!).

4.2 Computational Accuracy for Fixed-Point Implementation

With a fixed-point implementation of the proposed algorithms the question of computational accuracy also becomes important, so that a suitable scaling strategy would be needed in order to prevent arithmetic overflow from occurring. With the RFFT-based solutions the recursive nature of the architecture (whereby the butterfly stages are processed one at a time) suggests the adoption of a conditional (or 'optimal') scaling technique, such as the block floating-point technique, the most accurate scheme whereby the fixed-point data is only shifted when an overflow occurs [15]. With the RFFT-based solutions, however, the pipelined nature of the architecture (whereby all the butterfly stages are processed simultaneously) suggests the adoption of an unconditional (or 'sub-optimal') scaling technique as being more appropriate in order that undesirable timing delays be avoided – these delays due to the need for the periodic updating of a common scaling exponent applicable to the outputs of each and every stage of butterflies. Thus, for implementations with fixed-point data of given word-length, an unconditional scheme shifts the data regardless of whether an overflow has occurred and would therefore inevitably result in less accurate results than a conditional scheme.

4.3 Timing Constraints of Both Approaches

Being a batch processing algorithm, the RFHT has a timing constraint which requires that the latency is less than the update period of the input data set, namely N clock cycles for an N sample data set. On the other hand, the RFFT algorithms operate in a streaming fashion so that the timing constraint requires that each algorithm is able to process the input samples serially as they are acquired – this means individual samples for when the I/O rate is set to one sample per clock cycle (as with the RSC based and mRSC-based solutions) or sets of samples for when the I/O rate is greater than one sample per clock cycle (as with the two samples per clock cycle of the RSDF-based solution).

The fact that the latency of the RFHT solution can be shown to be considerably less than the update period for small to medium sized data sets means that the hardware resources are not being maximally utilized. This suggests that a reduction in the power consumption may be achieved or, alternatively, that post RFHT processing may be carried out on the RFHT output data before the next input data set is available for processing. This could, for example, include the additional tasks of converting the transform outputs from Hartley space to Fourier space, which may be straightforwardly achieved via the simple additions and subtractions (plus right shifts, of length one, of the results) of Eqtns. 5 and 6, or of converting the RFHT outputs directly to power spectral density (PSD) estimates via the expression

$$PSD[k] = \left|X^{(F)}[k]\right|^2 = \frac{1}{2}\left(\left|X^{(H)}[N-k]\right|^2 + \left|X^{(H)}[k]\right|^2\right) \quad (8)$$

for subsequent signal detection/classification processing.

4.4 Performance/Resource Comparison

The relative performances of the pipelined FFT and memory-based FHT solutions to the real data DFT in terms of resource requirements and achievable latency/throughput are as summarized in Tables 2 and 3, with Table 2 providing generic performance parameters as well as highlighting those key properties held by the various solutions – including those relating to computational accuracy, bilateralism, design regularity and scalability, whilst Table 3 provides a detailed performance/resource comparison for a number of small-to-medium sized data sets – namely, for data sets comprising 64, 256 and 1024 real valued samples. Only the Versions II and IV solutions of the RFHT are considered, with Version II being optimized to minimize the arithmetic complexity (at the expense of increased memory requirement) and Version IV optimized to minimize the memory requirement (at the expense of increased arithmetic complexity).

Solution to N-point Real-Data DFT	Arithmetic Complexity		Data+Coefficient Memory (real words)	Latency / Update Period (clock cycles)		No of Data Reorderings	Fixed-Point Scaling Performance	Properties: Regular / Scalable / Bilateral
	No of Real Multipliers	No of Real Adders						
RFFT (RSC)	$\log_2 N-2$	$2.\log_2 N-2$	$N+9.\log_2 N-19$	$N+3.\log_2 N-8$	N	2 – Input&Output	Sub-Optimal[1]	No / No / No
RFFT (mRSC)	$\log_2 N-2$	$2.\log_2 N-2$	$N+5.\log_2 N-9$	$N+3.\log_2 N-8$	N	2 – Input&Output	Sub-Optimal[1]	No / No / No
RFFT (RSDF)	$3.\log_2 N-4$	$8.\log_2 N-8$	N–2	N/2	N/2	2 – Input&Output	Sub-Optimal[1]	No / No / No
RFHT (V II)	9	25	2N+3N/4	$N/8.\log_4 N$	N	1 – Input	Optimal[2]	Yes / Yes / Yes
RFHT (V IV)	16	39	$2N+3\sqrt{N}/2$	$N/8.\log_4 N$	N	1 – Input	Optimal[2]	Yes / Yes / Yes

Table 2: performance/resource comparison for low-complexity solutions to N-point real-data DFT
Note: 1 => unconditional scaling strategy, 2 => conditional scaling strategy

Length of Data Set N	Solution	Arithmetic Complexity		Data+Coefficient Memory (real words)	Arithmetic+Memory Sizing Estimate (logic slices)	Latency / Update Period (clock cycles)		No of Data Reorderings
		No of Real Multipliers	No of Real Adders					
64	RFFT (RSC)	4	10	99	2304	74	64	2 – Input&Output
	RFFT (mRSC)	4	10	95	2240	74	64	2 – Input&Output
	RFFT (RSDF)	14	40	64	3584	32	32	1 – Input
	RFHT (V II)	9	25	176	4456	24	64	1 – Input
	RFHT (V IV)	16	39	140	5112	24	64	1 – Input
256	RFFT (RSC)	6	14	309	6016	272	256	2 – Input&Output
	RFFT (mRSC)	6	14	287	5664	272	256	2 – Input&Output
	RFFT (RSDF)	20	56	254	7712	128	128	1 – Input
	RFHT (V II)	9	25	704	12904	128	256	1 – Input
	RFHT (V IV)	16	39	536	11448	128	256	1 – Input
1024	RFFT (RSC)	8	18	1095	18944	1046	1024	2 – Input&Output
	RFFT (mRSC)	8	18	1065	18464	1046	1024	2 – Input&Output
	RFFT (RSDF)	26	72	1022	21088	512	512	1 – Input
	RFHT (V II)	9	25	2816	46696	640	1024	1 – Input
	RFHT (V IV)	16	39	2096	36408	640	1024	1 – Input

Table 3: performance/resource comparison of solutions for various sized data sets
Note: simplified logic-based sizing estimate assumes 16-bit fixed-point processing & excludes contribution of control logic

Referring firstly to the properties of the various real-data DFT solutions described by the contents of Table 2, the three RFFT solutions would appear to offer similar performances with the RSC-based and mRSC-based solutions requiring the reordering of both the input and output data sets but an I/O rate of just one sample per clock cycle, whereas the RSDF-based solution requires the ordering of just the input data set but an increased I/O rate of two samples per clock cycle. Each of the RFHT solutions, in comparison, requires the reordering of just the input data set and an I/O rate of just one sample per clock cycle whilst the arithmetic requirement is fixed for each size of data set. This means that as the size of the input data set is increased the corresponding increase in silicon resources for the RFHT will be due primarily to the increased memory requirement, whilst for the three RFFT solutions the arithmetic component of the space complexity will also be significant with the number of real multipliers increasing logarithmically with each multiplier requiring $O(L^2)$ logic slices when implemented in logic.

Referring next to the detailed contents of Table 3, the RSC-based and mRSC-based RFFTs carry out each of the complex multiplications using just one real multiplier and two real adders whereas the RSDF-based solution exploits two-fold parallelism by processing two samples at a time using three

real multipliers and four real adders. As a result, the latency of the RSDF-based solution is able to be reduced, relative to the RSC-based and mRSC-based solutions, at the cost of increased arithmetic complexity. The two versions of the RFHT – which each achieve eight fold parallelism within the PE – are able to match the reduced latency of the RSDF-based RFFT, but only at the expense of increased silicon resources arising from the increased memory requirement (resulting from the double buffering of the input data). For the smallest 64 sample data set the latency of the two RFHTs is actually considerably lower than that achieved by all three RFFTs with the arithmetic requirement of the Version II solution being also considerably lower than that of the RSDF-based solution – the lowest arithmetic complexity for the examples considered is achieved by the RSC-based and mRSC-based solutions although, for $N \geq 4096$, the Version II solution of the RFHT will always achieve the lowest arithmetic complexity of the solutions discussed.

4.5 Approximate Sizing of Solutions

The simplified logic-based sizing estimates provided in Table 3 is obtained by making the following simplifying assumptions, namely that: a) an L×L pipelined multiplier will require of the order of $5L^2/8$ logic slices in order to produce a new output each clock cycle, whilst b) an L-bit adder will require L/2 logic slices, and c) the memory, in the form of dual port RAM, will require L logic slices for each memory location [7]. Note, however, that the availability of ever-more efficient embedded fast multipliers with the latest FPGA technologies would suggest some benefit in the increased use of such multipliers in assessing and carrying out the trade-off of arithmetic complexity against memory requirement – the sizing estimates of Table 3 cannot reflect the efficiencies available through the use of such optimized components. The adoption of such multipliers would make the memory optimized Version IV solution of the RFHT a potentially more attractive option than its arithmetically-optimized counterpart, the Version II solution, as the arithmetic complexity would become less critical than the memory requirement in terms of its contribution to the total silicon area as the size of the input data set is increased. Note also that the logic-based sizing estimates do not account for the control logic requirement which, for the RFHT solutions, would be expected to remain relatively constant as the size of the input data set is varied and would be expected to be lower than the size-dependent requirements of the more complex designs of the RFFT solutions [14].

5. Summary and Conclusions

The main objective of the study has been to provide a comparison and assessment of both the performance and the capabilities of two recently developed approaches to the problem of computing the real-data DFT. The approaches exploit pipelined FFT and memory-based FHT architectures and aim to produce resource-efficient parallel solutions that are able to optimize in some way the required amount of silicon resources so as to yield solutions possessing low SWAP – as required for use in resource and power constrained environments. The study has highlighted the key properties and relative advantages/ disadvantages of each approach, showing with each how the arithmetic complexity may be traded off against the memory requirement in order to optimize the use of the silicon resources available on the silicon-based computing device chosen for its implementation and to meet the appropriate timing objectives or constraints. A number of design issues not addressed with recent real-data FFT research – in particular, those relating to design simplicity, regularity and scalability – are also discussed which enable a more comprehensive assessment of a solution's performance and capabilities to be made.

The first approach involved the design of pipelined RFFT solutions geared to streaming operation and exploiting a multi-PE pipelined architecture, whilst the second approach involved the design of memory-based RFHT solutions geared to batch operation and exploiting a single-PE recursive architecture. Both approaches have yielded attractive power-efficient solutions, although when compared to those of the RFFT approach, the RFHT solutions possess the additional attractions of bilateralism and of increased design simplicity, regularity and scalability the RFFT solutions would need to be optimized for each particular application, a potentially costly process – as well as lending themselves more naturally to the adoption of an accurate conditional scaling strategy for fixed-point operation. Therefore, as the contents of Tables 2 and 3 illustrate, the 'best' choice of solution is a complicated one to resolve with no one performance parameter able to tell the whole story. The best choice is dependent upon a number of often conflicting factors deriving from the performance objectives, the constraints (such as those relating to latency and/or power) of the particular application (or set of applications) of interest and of the available silicon resources – bearing in mind that the target device may typically be required to carry out several functions aside from that of the real data DFT.

Declarations

Funding and/or Competing Interests

The work described in this study has been carried out without access to external funding and without the involvement of any outside interests. With regard to data availability, the question is not applicable to this study.

References

1. Akl, S. G. (1989). The design and analysis of parallel algorithms. Prentice-Hall, Inc..
2. Ayinala, M., & Parhi, K. K. (2013). FFT architectures for real-valued signals based on Radix-2^3 and Radix-2^4 algorithms. IEEE Transactions on Circuits and Systems I: Regular Papers, 60(9), 2422-2430.
3. Birkhoff, G., & Mac Lane, S. (2017). A survey of modern algebra. CRC Press.
4. Bracewell, R. N. (Ed.). (1986). The hartley transform. Oxford University Press, Inc..
5. Brigham, E. O. (1988). The fast Fourier transform and its applications. Prentice-Hall, Inc..
6. Chinnapalanichamy, A., & Parhi, K. K. (2015, April). Serial and interleaved architectures for computing real FFT. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 1066-1070). IEEE.
7. Dick, C. (2000). FPGAs: The high-end alternative for DSP

applications. DSP Engineering, 1165-1170.
8. 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.
9. Garrido, M., Huang, S. J., Chen, S. G., & Gustafsson, O. (2016). The serial commutator FFT. IEEE Transactions on Circuits and Systems II: Express Briefs, 63(10), 974-978.
10. 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.
11. Garrido, M. (2022). A survey on pipelined FFT hardware architectures. Journal of Signal Processing Systems, 94(11), 1345-1364.
12. He, S., & Torkelson, M. (1998, October). Designing pipeline FFT processor for OFDM (de) modulation. In 1998 URSI International Symposium on Signals, Systems, and Electronics. Conference Proceedings (Cat. No. 98EX167) (pp. 257-262). IEEE.
13. Jones, K. J. (2006). Design and parallel computation of regularised fast Hartley transform. IEE Proceedings-Vision, Image and Signal Processing, 153(1), 70-78.
14. Jones, K. J., & Coster, R. (2007). Area-efficient and scalable solution to real-data fast Fourier transform via regularised fast Hartley transform. IET Signal Processing, 1(3), 128-138.
15. Jones, K.J. (2022). The Regularized Fast Hartley Transform: Low-Complexity Parallel Computation of the FHT in One and Multiple Dimensions. 2nd Edition, Springer.
16. Maxfield, C. (2004). The design warrior's guide to FPGAs: devices, tools and flows. Elsevier.
17. 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.
18. Pattan, A. B., & Latha, M. M. (2014). Fast Fourier Transform Architectures: A Survey and State of the Art. International Journal of electronics & communication technology, IJECT, 5(4), 94-98.
19. Salehi, S. A., Amirfattahi, R., & Parhi, K. K. (2013). Pipelined architectures for real-valued FFT and hermitian-symmetric IFFT with real datapaths. IEEE Transactions on Circuits and Systems II: Express Briefs, 60(8), 507-511.