ISSN: 2993-5377

Research Article

Open Access Journal of Applied Science and Technology

Design for Resource-Efficient Parallel Solution to Real-Data Sparse FFT

Keith John Jones*

Consultant Mathematician(Retired), UK

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

Submitted: 2023, July 01; **Accepted:** 2023, July 21; **Published:** 2023, Aug 21

Citation: Jones, K. J. (2023). Design for Resource-Efficient Parallel Solution to Real-Data Sparse FFT. *OA J Applied Sci Technol, 1*(2), 68-88.

Abstract

The maximum size of data set being presented to the discrete Fourier transform (DFT) is becoming increasingly large to reflect the increasingly challenging problems being faced in today's 'big data' era, in areas such as astronomy, medical imaging and the real-time spectrum analysis of multi GHz radio frequency signals for cognitive radio networks. Such problems are typically addressed by means of the fast Fourier transform (FFT), but there will always be data sets – typically real valued in nature – that are too large to be efficiently processed in real time with existing computing technology, so that alternative approaches are needed. The approach pursued here for the spectrum analysis problem assumes that a relatively small number of outputs are likely to contain detectable levels of signal energy with such signals being detected through the use of a sparse version of the FFT (sFFT). A flexible and scalable sFFT design has been sought for implementation with silicon-based computing technology that's able to yield resource efficient low power solutions by maximizing the computational density through exploitation of both partitioned memory and the real valued nature of the data. A theoretical analysis shows how this may be achieved with a parameterized solution which, with a low-end field programmable gate array (FPGA) device, a 2 GHz sampling rate and a 100 MHz clock rate, is able to achieve a latency of < 1 ms for a 2 million point real-data sFFT together with low resource utilization and which compares favourably with other recently published FFT and sFFT solutions.

Keywords: FFT, FHT, FPGA, Sparse, Spectrum, Sub-Sampling

1. Introduction

The discrete Fourier transform (DFT) is an orthogonal transform which may be expressed in its normalized form as

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

for k = 0,1,…,N-1, where

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

is the N^{th} complex primitive root of unity [5,12,34]. The maximum size of data set being presented to the DFT is becoming increasingly large to reflect the increasingly challenging problems being faced in today's 'big data' era, in areas such as astronomy, medical imaging and the real-time spectrum analysis of multi GHz radio frequency signals – such as might be encountered, for example, with cognitive radio networks – where the power spectral density (PSD) may be obtained directly from the DFT coefficients. Such problems are typically addressed by means of a fast solution to the DFT, referred to generically as the fast Fourier transform (FFT) with much research being carried out in recent years into the design of such algorithms for the case where the transform is both 'dense' and very large, with references catering for transforms of up to 1 million (M) samples. However, *there will always be problems that are too large to be efficiently implemented with the existing computing technology* – in terms of both power consumption and silicon resources – so that alternative approaches need still to be found for addressing such problems [12,16,26,27].

When dealing with real-world spectrum estimation problems involving such large data sets it is generally the case that a relatively small number of the outputs will actually contain detectable levels of signal energy, with the remainder comprising just noise. As a result, the problem of detecting those frequency-dependant signals may be tackled through the use of a 'sparse' version of the FFT, referred to hereafter as the sFFT [15]. A generic version of the algorithm, based upon research carried out at the Massachusetts Institute of Technology (MIT) in the USA over the past decade or so is outlined in Fig. 1, which lists the various tasks that need typically to be performed. With this algorithm, which may be viewed as a key computational tool in the increasingly important field of *compressive sensing* wideband spectrum sensing systems may be defined whereby only those signal frequencies of interest are identified and the associated spectrum components computed [1,2,13,30].

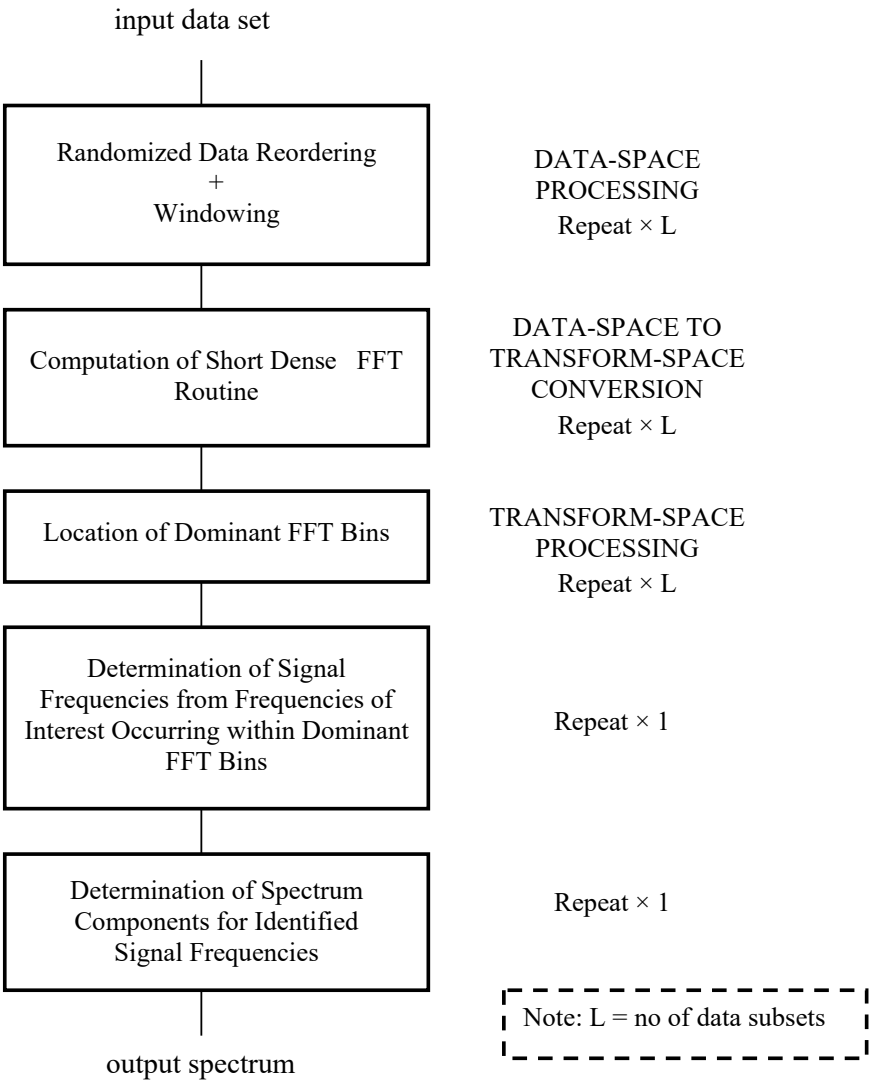

Figure 1: Outline of Generic Processing Scheme for Sparse FFT algorithm

As with most modern digital signal processing (DSP) algorithms, the input data set to the sFFT is invariably taken to consist of complex valued samples comprising both real (in-phase) and imaginary (quadrature) components so that the short dense (as opposed to sparse) FFTs needing to be performed by the algorithm are also assumed to operate on complex valued data. With most real world applications, however, the naturally (NAT) ordered input data typically starts out in real valued (fixed-point integer) form, as produced by the analog to digital conversion (ADC) unit, only to be subsequently processed with a complex data solution – regardless of the nature of the input data – given that the designs of most commercially available fixed radix FFTs are built around the adoption of the complex-data multiplier and accumulator (MAC). This is an arithmetic unit ideally suited to the implementation of the radix-2 butterfly, which is the computational engine used for carrying out the repetitive arithmetic operations required by the complex data version of the radix-2 FFT.

The complex-data approach might typically entail the initial conversion of the real valued data to complex-valued data via a wideband digital down-conversion (DDC) process or the adoption of a 'real from-complex' strategy whereby two real data DFTs are computed simultaneously via one full length complex-data FFT or where one real-data DFT is computed via one half-length complex-data FFT [25]. Each of the real from-complex solutions, however, involves a computational overhead when compared to the more direct approach of a real-data FFT in terms of increased memory, increased processing delay to allow for the possible acquisition/processing of pairs of data sets, and additional packing/unpacking complexity. With the DDC approach, the integrity of the information content of short duration signals may also be compromised through the introduction of the filtering operation.

A number of specialized FFT algorithms do exist for dealing with the case of real-valued data which compare favourably, in terms of arithmetic complexity and memory requirement, with those derived using the real-from-complex strategy, but suffer in terms of a loss of regularity (making silicon based hardware implementations somewhat less attractive) and reduced flexibility in that different algorithms are typically

required for the computation of the forward DFT and that of its inverse [7,11,31]. A more recent study has produced a solution, based upon modification (removing redundant operations) of the complex-data version of the familiar radix-2 Cooley-Tukey algorithm with NAT-ordered inputs, which possesses a more regular structure for an efficient *pipelined* implementation – although the processing requirements do vary from one pipeline stage to the next [3,9,14]. This is aimed at *streaming* (or continuous flow) rather than *block-based* (or batch) operation and is achieved at the expense of having the outputs produced in a non-standard (that is, not bit-reversed) order so that a more complex pipelining scheme is required for retrieving the data in the required NAT-ordered form [25]. Such solutions to the real-data FFT, however, are not scalable (which refers to the ease with which the solution may be modified in order to accommodate increasing or decreasing transform sizes or parameter changes) so that larger transforms require proportionately longer pipelines and thus increased latency as well as proportionately more resources for their implementation.

The aim of this paper, therefore, is to produce a flexible (achieved through parameterization) and scalable design for the real-data sFFT that is able, through maximization of the throughput per unit area of silicon – referred to as the *computational density* – to yield resource efficient low power solutions that are also able to exploit directly the real valued nature of the data, as produced by the ADC unit [25]. This is achieved through: 1) the extensive use of partitioned memory as this facilitates the parallel computation of the sFFT which enables the clock rate, and thus the power consumption, to be minimized, and 2) the adoption of a *resource efficient* and *scalable* solution to the discrete Hartley transform (DHT) [6,18] – referred to as the regularized fast Hartley transform (RFHT) – for carrying out, in optimal fashion, the short dense real-data FFTs – referred to as SDRD-FFTs – used for processing the multiple short randomly-generated (SRG) data sets, as derived from the input data set via the use of a random (RND) number generator and as required by most variations of the sFFT algorithm [23-25].

Note that most of the memories used for storage of the various data sets produced during the execution of the sFFT algorithm are each to be partitioned into eight equal sized memory banks made up of fast dual port random access memory (RAM). This makes them consistent with the operation of the RFHT whose performance has already been proven in silicon with a fixed-point implementation using field programmable gate array (FPGA) technology and with partitioned memory being used to facilitate an eight fold speed-up over a purely sequential solution to the real data FFT [24,33].

The validity of using the RFHT – which is simply a fast and highly-parallel solution to the real valued DHT – for computing the real data DFT derives from the fact that the output data sets produced by the two orthogonal transforms, the DFT, as given by Eqtn. 1, and the DHT, as given in its normalized form by the expression

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

for k = 0,1,…,N-1, where the transform kernel is given by the 'cas' function

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

may each be simply obtained, one from the other, so that the class of fast algorithms typically used for solving the DHT – referred to generically as the fast Hartley transform (FHT) and for which the RFHT is a member – may also be effectively used to solve the DFT, particularly when the input data set is known to be real valued in nature [6]. To see the truth of this, note that the equality

$$\mathrm{cas}\left(2\pi nk / N\right) = \mathrm{Re}\left(W_N^{nk}\right) - \mathrm{Im}\left(W_N^{nk}\right) \tag{5}$$

(where 'Re' stands for real component and 'Im' stands for imaginary component) relates the kernels of the two transformations. As a result

$$X^{(H)}[k] = \mathrm{Re}\left(X^{(F)}[k]\right) - \mathrm{Im}\left(X^{(F)}[k]\right), \tag{6}$$

which expresses the DHT outputs in terms of the DFT outputs, whilst the equations

$$\mathrm{Re}\left(X^{(F)}[k]\right) = \tfrac{1}{2}\left(X^{(H)}[-k] + X^{(H)}[k]\right) \tag{7}$$

$$\&\quad \mathrm{Im}\left(X^{(F)}[k]\right) = \tfrac{1}{2}\left(X^{(H)}[-k] - X^{(H)}[k]\right), \tag{8}$$

express the real and imaginary components of the DFT outputs, respectively, in terms of the DHT outputs and where, from transform periodicity, index '–k' may be regarded as being equivalent to 'N–k'. This enables the PSD outputs to be expressed directly in terms of either DFT or DHT outputs via the expression

$$\begin{aligned} PSD[k] &= \mathrm{Re}^2\left(X^{(F)}[k]\right) + \mathrm{Im}^2\left(X^{(F)}[k]\right) \\ &\equiv \tfrac{1}{2}\left(\left(X^{(H)}[k]\right)^2 + \left(X^{(H)}[-k]\right)^2\right), \end{aligned} \tag{9}$$

as will be required when searching for the dominant SDRD-FFTs outputs, to be discussed in Section 5.

Thus, following this introductory section, Section 2 provides a brief overview of the design process which involves a description of the basic tasks needing to be performed together with a discussion of some of the key design issues. Section 3 next discusses the first (or data space) stage of the processing chain (see Fig. 1), which is concerned with the derivation of the windowed versions of the multiple SRG (referred to as WSRG) data sets. This is followed in Section 4 with an account of the second stage of the processing chain which converts the problem from data-space to transform-space and relates to how the RFHT may be efficiently used for carrying out the SDRD-FFTs that operate upon the WSRG data sets. Section 5 next discusses the third and final (or transform space) stage of the processing chain (see Fig. 1), namely the determination of: 1) the locations of the dominant frequency bins for each set of SDRD FFT outputs – after first converting the data from Hartley space to Fourier

space and generating the PSD data, together with 2) those signal frequencies appearing as frequencies of interest (FOI) within the dominant frequency bins of each set of SDRD FFT outputs and, finally, 3) the spectrum components for those identified signal frequencies. This is followed in Section 6 with a discussion of the possible complexity trade offs – in terms of both 'space' and 'time' complexities – to be considered in producing a resource-efficient parallel solution for the proposed real-data sFFT algorithm, with a detailed illustration in Section 7 of a 2M point multi GHz example. Finally, a summary and conclusions is provided in Section 8.

2. Design Issues for Proposed Scheme

The problem to be addressed in this paper involves the design of a fixed-point sFFT algorithm for carrying out the DFT of a large real valued data set, of length 'N', where N is taken (for ease of analysis) to be a radix-2 integer. The generic version of the algorithm, as illustrated in Fig.1, involves repeatedly mapping N frequencies into a much smaller number of frequency bins, say 'P', this being achieved by carrying out an SDRD-FFT upon multiple WSRG data sets, each of length P, where the window (such as the Dolph Chebyshev function is designed to extract just a subset of the input data set and possesses the attraction of having a narrow support in both data space and transform space [17,23-25]. The energy contained within any given frequency bin is obtained as the sum of the spectrum components corresponding to those N/P frequencies that are mapped into the bin. By carefully choosing the P samples to be processed it is possible to ensure that with each data set different frequencies will map into different bins with a high probability. By repeating this binning process, say 'L' times, each time using a WSRG data set obtained with a different set of permuted input samples, the algorithm is able to permute the spectrum components and to randomize the mapping of frequencies to bins so that those signal components that were initially closely spaced in frequency will, with a high probability, become sufficiently isolated within the permuted spectrum to allow for their unambiguous recovery.

Note, however, that when dealing with real-world data the recovery of the individual frequency dependent signal components will be heavily dependent upon the available signal-to-noise ratio (SNR), where the lower the SNR the more difficult the problem of signal detection becomes. However, the situation is improved somewhat in that the detection is to be carried out in the frequency domain, rather than the time domain, so that the SNR will benefit from an increase provided by the coherent gain of the SDRD-FFT, namely $10.\log_{10}P$ dB, so that the longer the SDRD-FFTs can be made without adversely affecting the real-time capability of the proposed sFFT algorithm the better the performance in terms of both detection (due to reduced variance in the spectral data) and false alarm rates (arising from reduced collisions where multiple frequencies map to the same SDRD-FFT bin). This can be achieved by having the SDRD FFTs carried out via the RFHT given its already proven implementational attractions in terms of both resource efficiency and scalability.

The speed at which the NAT-ordered input data samples are to be produced by the ADC unit is typically equal to, or some integer multiple of, the clock rate of the target computing device – assumed here to be an FPGA. This sampling rate dictates the value of the 'data set refresh rate', which is defined as the rate at which each new input data set is transferred from the ADC to the external data space memory (DSM). The DSM is taken to be a partitioned memory – consisting of eight memory banks with each memory bank containing N/8 samples – which is maintained in double-buffered form in order to facilitate continuous real-time operation. The associated time period is referred to as the 'update period' and consecutive samples are taken to be stored cyclically across the eight memory banks with memory bank no 8 always being followed by memory bank no 1. Clearly, with the multi GHz wide bandwidth signals of interest it is evident that each clock cycle will yield multiple samples as there is a clear need for the clock rate of the target FPGA device (typically measured in MHz rather than GHz) to be kept as low as possible in order that the power consumption be minimized. This relationship is evident by noting that the dominant dynamic power component, P_D, may be expressed as

$$P_D = C \times V^2 \times f \tag{10}$$

where 'C' is the capacitance of the node switching, 'V' the supply voltage and 'f' the clock or switching rate, so that reducing the clock rate will lead to a reduction in the power consumption [25].

The objective of the research described in this paper is thus to come up with a flexible and scalable design for the real-data sFFT that's able to produce solutions yielding a new sparse spectrum with every update period and such that the required silicon resources and the clock rate enable the associated costs and power consumption to be minimized. As already stated, the approach taken involves producing a design based upon the maximizing of the computational density through the combined use of partitioned memory, which is a key technique for enabling the adoption of a low clock rate, and the RFHT, which is a resource efficient and scalable means of carrying out the DHT and/or the real data DFT. The solutions will be assessed in terms of their space and time complexities where, for any given task, the 'space complexity' is defined as comprising arithmetic and memory components where the arithmetic component corresponds to the numbers of fast multipliers and adders required to carry out that task and the memory component to the required amount of fast dual port RAM. The 'time complexity' corresponds to the number of clock cycles required to carry out the task which, for a realizable solution, needs clearly to be sufficiently less than that corresponding to the update period.

3. Data-Space Processing Requirement

The first stage of the processing chain involves the derivation of the multiple WSRG data sets, as obtained from the partitioned DSM and the window function coefficients. The resulting data sets will be subsequently used as inputs to the second stage, namely that concerning the computation of the SDRD FFTs via the RFHT. As already stated, emphasis is to be placed on the exploitation of partitioned memory as this will facilitate the parallel computation of the proposed solution and thus enable the adoption of a lower clock rate for the target computing device –

as required if the power consumption is to be minimized. Thus, given the partitioning of each of the various memories into eight equal sized banks, each eight sample data set (with at most two samples per memory bank) will be referred to hereafter simply as a 'woctad' (where 'octad' means set of eight objects so that 'woctad' is defined as meaning set of eight words where each word holds either a single sample of data or a single coefficient or a single data address) in order to avoid unnecessary verbiage.

3.1 Random Reordering of Input Data Set and its Spectrum

Suppose now that the NAT-ordered input data set, denoted {x[n]}, as stored in the partitioned DSM, is to be reordered or permuted according to the RND-generated index mapping [5]

$$\Phi[n,\sigma] \equiv \sigma \times n \bmod N \tag{11}$$

to yield a RND-reordered data set, {y[n]}, where

$$y[n] = x[\Phi[n,\sigma]] \tag{12}$$

for n = 0 up to N-1, with 'σ' being an RND-generated (or pre-selected) 'invertible' integer such that

$$\sigma \times \sigma^{-1} \equiv 1 \bmod N . \tag{13}$$

This requires that the integer σ should be relatively prime [34] to N which may be easily satisfied by simply selecting N to be an even-valued integer (such as a radix-2 integer assumed here) and σ to be an odd-valued integer.

Then, when viewed in Fourier-space, following the application of an N-point dense FFT, it can be shown that the spectra obtained from the processing of the NAT-ordered and RND-reordered data sets, denoted {X[n]} and {Y[n]}, respectively, will be related via the inverse index mapping [19]

$$\Phi^{-1}[n,\sigma^{-1}] \equiv \sigma^{-1} \times n \bmod N \tag{14}$$

with

$$Y[n] = X[\Phi^{-1}[n,\sigma^{-1}]], \tag{15}$$

so that the two spectra may be simply obtained, one from the other.

Thus, for a given index of the RND-reordered spectrum, {Y[n]}, the mapping Φ of Eqtn. 11 tells us to which index of the NAT-ordered spectrum, {X[n]}, it corresponds. Conversely, for a given index of the NAT-ordered spectrum, {X[n]}, the inverse mapping Φ^{-1} of Eqtn. 14 tells us to which index of the RND-reordered spectrum, {Y[n]}, it corresponds. As a result, if the n'th element of the NAT ordered spectrum is f[n], then after permutation by Φ^{-1} the n'th element of the resulting RND reordered spectrum will be given by $f[\Phi^{-1}(n,\sigma^{-1})]$ and after this spectrum is itself permuted by Φ the n'th element of the resulting spectrum will be given by

$$f[\Phi[\Phi^{-1}[n,\sigma^{-1}]],\sigma] = f[(\sigma \times \sigma^{-1} \times n) \bmod N] = f[n], \tag{16}$$

thereby enabling the original frequency to be recovered. Note, from Eqtns. 11 and 14, that the index sequences, $\{\Phi[n,\sigma]\}$ and $\{\Phi^{-1}[n,\sigma^{-1}]\}$, are *arithmetic sequences* so that consecutive indices may be simply expressed in recursive forms, under the modulo operation, as

$$\Phi[n+1,\sigma] \equiv \Phi[n,\sigma] + \sigma \bmod N \tag{17}$$

$$\& \quad \Phi^{-1}[n+1,\sigma^{-1}] \equiv \Phi^{-1}[n,\sigma^{-1}] + \sigma^{-1} \bmod N \tag{18}$$

respectively, enabling the computation of successive indices to be greatly simplified by having an integer multiplication replaced by an integer addition – followed, in each case, by a possible range reduction.

Suppose now that the length of the reordered data set is to be reduced from N down to P, where P << N and P|N, prior to being transformed to Fourier-space via the application of a P-point SDRD-FFT. Then it can be shown that consecutive sets, each of N/P RND reordered spectral samples, as obtained from the NAT reordered spectrum via the inverse mapping Φ^{-1}, will map into consecutive SDRD FFT bins with the n'th output of the N-point spectrum, Y[n], mapping into $\tilde{Y}[\tilde{n}]$ where $\tilde{Y}$ represents the P point SDRD-FFT spectrum and $\tilde{n}$ is such that [19]

$$\tilde{n} = \left\lfloor n / (N/P) \right\rfloor \tag{19}$$

which, when N/P is a radix-2 integer (which may be guaranteed through appropriate choice of the parameters N and P), may be simply computed by means of a right-shift operation followed by σ truncation of the result.

Now, for any valid pair of σ (odd-valued integer) and N (even-valued integer), each woctad of SRG data samples retrieved from the partitioned DSM will be such that one sample comes from each of the eight memory banks, so that with dual port RAM each woctad of data samples may be retrieved within a single clock cycle – in fact, two woctads may be simultaneously retrieved with such memory although we'll restrict ourselves to one woctad per clock cycle for compatibility with subsequent processing. The set of indices required for each woctad of SRG data samples retrieved from the partitioned DSM may be replaced by a set of eight address pairs, $[m_n,t_n]$, for n = 0 to 7, where the first 3-bit number of each pair represents the memory bank address of the n'th sample within the set and the second $\log_2(N/8)$ bit number of each pair represents the corresponding time slot address within that memory bank. Thus, for a fixed pre-chosen value of σ, by storing each address pair within a single word of memory the P address pairs may be pre computed and stored within the partitioned data address memory (DAM), which consists of eight memory banks with each bank containing Q = P/8 addresses, in order to simplify the task of producing L SRG data sets, each of length P, from the original N sample input data set and, in so doing, to reduce the associated addressing complexity. Each address stored within the DAM consists of a W_A-bit word, with $W_A \geq \log_2(N/8)+3$, whilst each (integer-valued) sample of data stored within the DSM consists of a W_D-bit word. For ease of analysis, however, all data samples and addresses will be assumed hereafter to be of common length,

namely W bits, where W is such that $W \geq \max(W_A, W_D)$.

Thus, ignoring the memory requirement for the external storage of the initial NAT-ordered input data set within the partitioned and double-buffered DSM, if L SRG data sets are to be produced, where distinct values of σ are required to be generated and used according to the constraint of being odd-valued, then the space complexity required for carrying out the RND reordering of data in a fine-grained parallel (i.e. exploiting partitioned memory), coarse-grained sequential (i.e. SRG data sets processed one at a time) fashion will involve a zero arithmetic component and a memory component consisting of: 1) L×P words for the storage of the resulting SRG data sets within the partitioned transform space memory (TSM), with each data set being assigned its own version of the memory consisting of eight memory banks with each bank containing Q samples, together with 2) L×P words for the storage of the pre computed sample addresses within the partitioned DAM, with each address set being assigned its own version of the memory consisting of eight memory banks with each bank containing Q addresses. This results in a total memory component, denoted M_{RND}, of

$$M_{RND} \approx 2(L \times P) \tag{20}$$

words, whilst the associated time complexity, denoted T_{RND}, for carrying out these tasks in such a fashion will involve Q clock cycles for producing each SRG data set, resulting in a total for all L data sets of

$$T_{RND} \approx L \times Q = \tfrac{1}{8}(L \times P) \tag{21}$$

clock cycles, where one woctad of data is written to the TSM whilst the next woctad of data and its addresses are being read from the DSM and the DAM, respectively. However, before being written to the TSM, there is another task that must first be performed.

3.2 Windowing of Randomly-Generated Data Sets

The next task in the processing chain is concerned with the application of a window function, of length P, to each of the SRG data sets with the resulting WSRG data sets being written to the partitioned TSM. Each of the SRG data sets uses its own set of pre-computed window coefficients which are stored within its own partitioned window coefficient memory (WCM). Each set of coefficients is obtained from an N-point NAT-ordered version of the full window function (which does not need to be computed and stored) via the same sets of addresses, obtained from the DAM, as used for accessing the data from the partitioned DSM. For the task to be efficiently carried out in a parallel fashion a set of eight fast multipliers is provided which enables the latest SRG data woctad retrieved from the DSM to be multiplied, sample-by-sample, by the appropriate elements of the coefficient woctad, obtained from the WCM, before being written to the TSM – see Fig. 2. This task takes place whilst the next data and coefficient woctads are being retrieved from the DSM and WCM, respectively. By overlapping the two processing steps in this way, the set of eight fast multipliers – which each typically operates in a pipelined fashion over several clock cycles – may be fed continuously with new data and coefficient woctads every clock cycle.

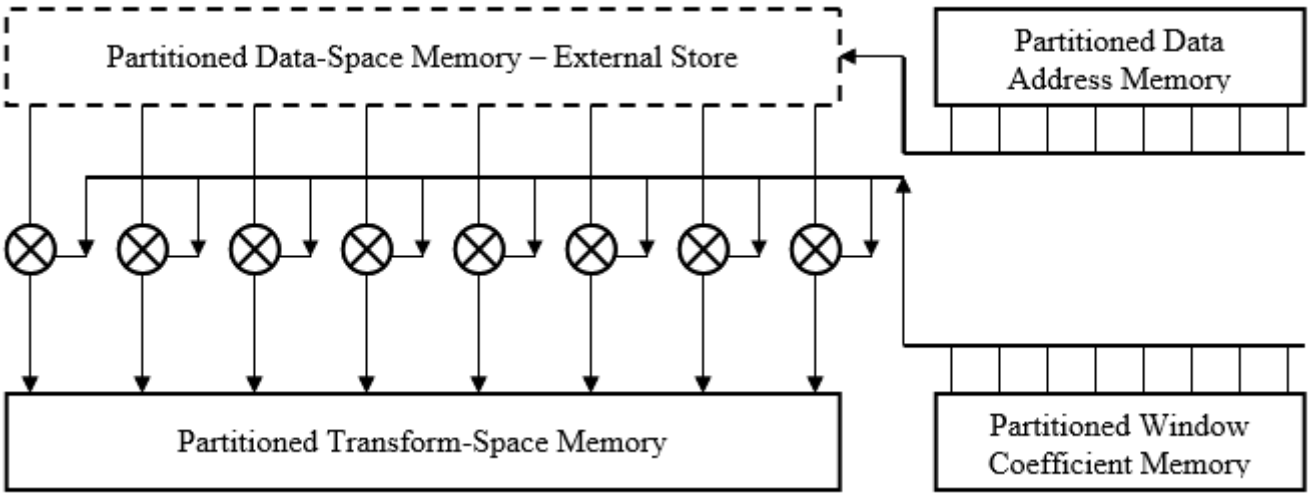

Figure 2: Scheme for Generation and Windowing of Reordered Samples Selected from Input Data Set

The space-complexity required for carrying out the windowing of the multiple SRG data sets in a fine grained parallel, coarse-grained sequential fashion involves an arithmetic component, denoted A_{WND}, of

$$A_{WND} = 8 \text{ multipliers } \& \text{ 0 adders} \tag{22}$$

together with a memory component, denoted M_{WND}, of

$$M_{WND} \approx L \times P \tag{23}$$

words. Due to the overlapping of: 1) the retrieval of the data and coefficient woctads, with 2) the application of the coefficients to the data samples, the associated time complexity, denoted T_{WND}, for carrying out the windowing in such a fashion will be 'effectively' zero, apart from that involving the small start-up delay required for the pipelining of the fast multipliers.

3.3 Summary of Complexity Requirements

At this point L WSRG data sets, each of length P and stored within its own version of the TSM, are available for input to

the next stage of the processing chain concerning the use of the RFHT, the outputs of which will also be stored within the TSM (by simply over-writing the WSRG data) for subsequent conversion from Hartley-space to Fourier-space. From the complexity results provided in this and the previous sections it is evident that the space complexity for the first stage of the processing chain involving the production of the multiple WSRG data sets in a fine grained parallel, coarse-grained sequential fashion possesses an arithmetic component, denoted A_{STG1}, of

$$A_{STG1} = A_{RND+WND} = 8 \text{ multipliers \& } 0 \text{ adders} \quad (24)$$

together with a memory component, denoted M_{STG1}, of

$$M_{STG1} = M_{RND+WND} \approx 3(L \times P) \quad (25)$$

words, whilst the associated time-complexity, denoted T_{STG1}, for carrying out the combined task in such a fashion may be expressed as

$$T_{STG1} = T_{RND+WND} \approx (L \times Q) = \tfrac{1}{8}(L \times P) \quad (26)$$

clock cycles, after taking into account the overlapping of their operations.

Note that although the memory component of the space-complexity has been increased in order to cater for the storage of the pre-computed sample addresses and window coefficients associated with each WSRG data set, the time complexity has been considerably reduced as their pre computation and storage avoids the need for their costly on-the-fly computation.

4. Data-Space to Transform-Space Conversion

The second stage of the processing chain is concerned with the computation of low-resolution spectral samples via the L SDRD FFTs, this being achieved by applying the RFHT to the WSRG data sets produced by the first stage. Note, however, that the conversion routine for transferring the data from Hartley space to Fourier space – as described by Eqtns. 7 and 8 – is omitted at this stage as it may be included more naturally as part of the transform-space processing of Section 5.

4.1 Parallel Data Reordering via Dibit-Reversal Mapping

Being a radix-4 decimation-in-time (DIT) algorithm the input data to the RFHT – as stored within the partitioned TSM – needs first to be reordered according to the dibit-reversal (DBR) mapping which is a radix-4 digit reversal permutation in which the radix-4 digits of the index of each element are reversed in order to obtain the permuted index [25,31]. The DBR reordered input data set may then be transferred from the TSM to the partitioned memory used for the storage of the RFHT input/output data with consecutive data samples being stored cyclically within consecutive memory banks. On completion of the RFHT, the NAT ordered output data set may be read out from the partitioned memory with consecutive data samples being retrieved cyclically from consecutive memory banks. With each partitioned memory made up of dual-port RAM and comprising eight memory banks it can be shown that the DBR reordered samples may be transferred from one set of memory banks to another at the rate of two woctads (with two samples from each memory bank) per clock cycle, so that the time complexity, denoted T_{DBR}, for the construction and transfer of each DBR reordered data set of length P may be expressed as [25]

$$T_{DBR} \approx P/16 \quad (27)$$

clock cycles, whilst the associated space-complexity – apart from a small fixed amount of intermediate memory – is insignificant.

A number of alternative implementations of the digit reversal mapping – as required for those fixed-radix FFTs where the radix is a power of two – have been produced in recent years, such as those pipelined implementations described in which are designed for consistency with the streaming operation of the pipelined FFT [8,29]. With the DBR mapping and the adoption of four-fold parallelism that is, processing four samples at a time for the corresponding radix-4 real-data FFT – such solutions are able to achieve, for the case of large P, a latency of ~ P/4 clock cycles at the expense of ~ P words of additional memory. Such solutions, however, are not scalable with larger transforms (and digit-reversal routines) requiring proportionately longer pipelines and thus increased latency as well as proportionately more resources for their implementation.

4.2 The Regularized Fast Hartley Transform – A Summary

The correctness of operation of the RFHT – which has shown itself ideally suited to block based rather than streaming operation – has already been proven in silicon with a fixed point implementation using FPGA technology where the storage of the data and the trigonometric coefficients was carried out in each case using dual port RAM [24]. A brief overview of the algorithm is now provided in order to highlight the merits of its application to the current problem, namely the computation of multiple SDRD FFTs where the length P of each transform may be freely chosen to be a radix-4 integer.

4.2.1 Overview

The RFHT is a resource efficient and scalable means of carrying out the DHT in a highly parallel fashion, whilst it's being 'regularized' refers to the fact that the algorithm structure has been made regular (by maximizing the amount of repetition and symmetry present in the design) so that the conventional need for two separate *butterfly* designs for the fixed-radix FHT is thus avoided [25]. The design includes two key features: 1) an architecture based upon the use of a single *processing element* (PE), as shown in Fig. 3, which exploits partitioned memory to facilitate the parallel computation of the butterfly operation, and 2) *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 (or *twiddle factors*), as stored in the PE's internal coefficient memory (PCM). These features, when combined with pipelining techniques for the internal operation of the PE, enable the *generic double butterfly* – the large computational engine used by the RFHT – to produce output woctads at the rate of one per clock cycle.

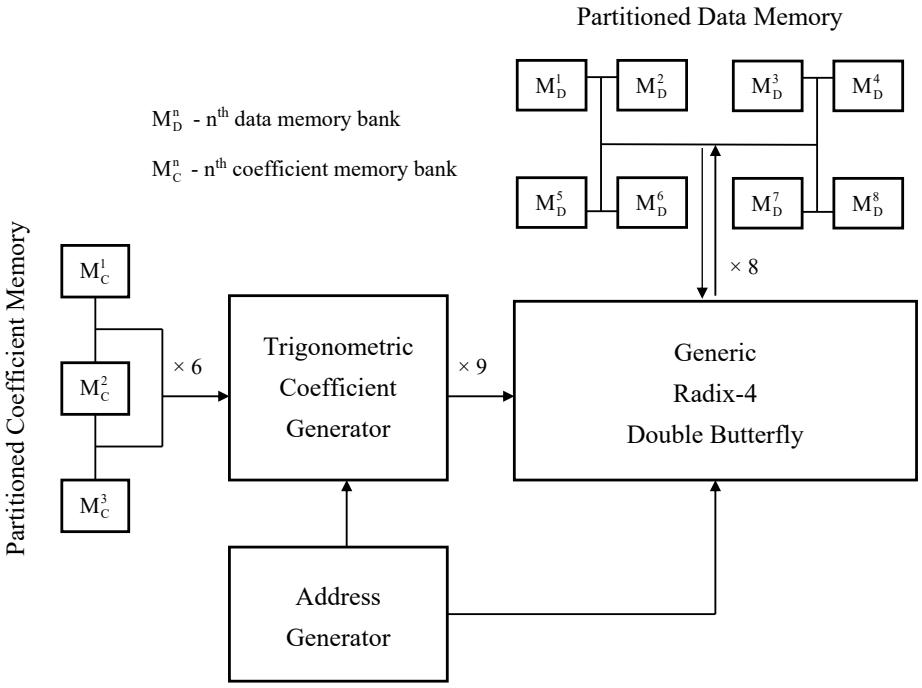

Figure 3: Single-PE Recursive Architecture for Regularized FHT

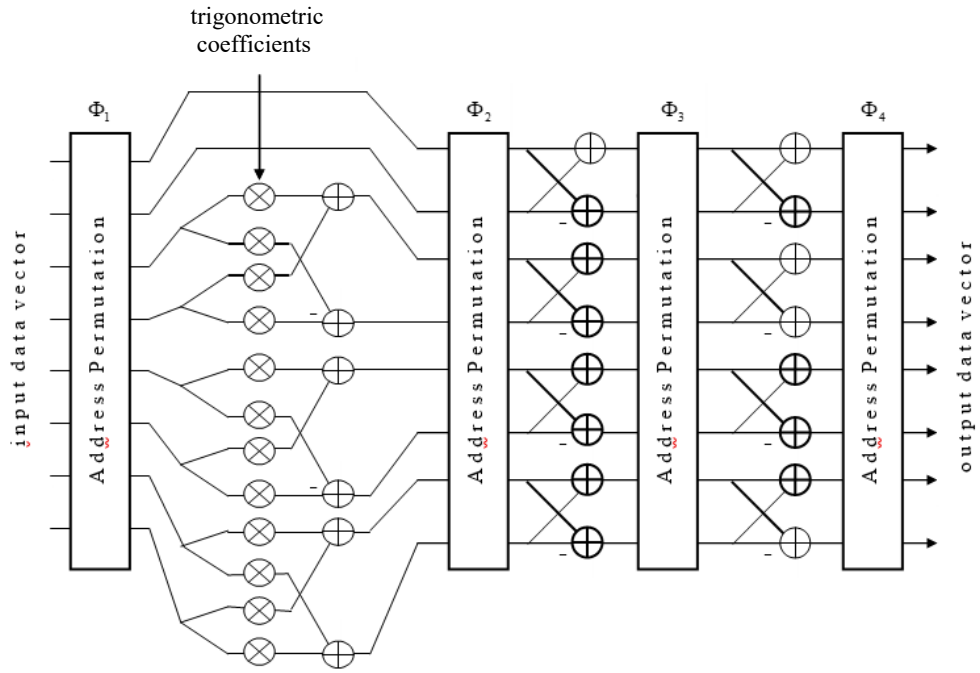

Figure 4: Signal Flow Graph for Twelve-Multiplier Version of Generic Double Butterfly

The original design [23] required 12 multipliers and 22 adders for carrying out the double butterfly operation, as shown in Fig. 4, with: 1) each woctad (as obtained from four or eight memory banks) being read/written in parallel from/to the partitioned PDM, configurable as an array of eight memory banks, and 2) 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 [23]. The PDM addressing, over two consecutive clock cycles, enables all those samples required by consecutive 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 woctad per clock cycle [25].

4.2.2 Resource-Constrained Design Variations

Three additional versions of the PE have been subsequently derived 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 [25]. The use of two level LUTs results in a reduced memory requirement of O($O(\sqrt{P})$rds, as opposed to the

O(P) 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, although it should be noted that a fifth version using a PE based upon the use of Co Ordinate Rotation DIgital Computer (CORDIC) arithmetic has also been successfully produced [25,36]. With each version, the RFHT achieves an O(P×logP) time complexity which corresponds, in clock cycles, to the total number of double butterflies to be executed per transform, namely $1/8(P\times\log_4 P)$. The adoption of Version II of the RFHT is to be assumed hereafter when assessing space-complexity.

Version of Solution	Arithmetic-Complexity				Memory Requirement (words)		Time-Complexity (clock cycles)
	Processing Element		Coefficient Generator		Data Memory (Single-Buffer)	Coefficient Memory	Update Time / Latency
	Multipliers	Adders	Multipliers	Adders			
I	12	22	0	0	N	$\frac{3}{4}N$	$\frac{1}{8}N.\log_4 N$
II	9	25	0	6	N	$\frac{3}{4}N$	$\frac{1}{8}N.\log_4 N$
III	12	22	7	8	N	$\frac{3}{2}\sqrt{N}$	$\frac{1}{8}N.\log_4 N$
IV	9	25	7	14	N	$\frac{3}{2}\sqrt{N}$	$\frac{1}{8}N.\log_4 N$

Table 1: Performance/Resource Comparison for Fast Multiplier Versions of N-Point Regularized FHT

4.3 Summary of Complexity Requirements

The space complexity for carrying out the second stage of the processing chain involving the construction and transfer of the L DBR reordered data sets followed by the corresponding P point RFHTs, in a fine grained parallel, coarse-grained sequential fashion, possesses an arithmetic component, denoted A_{STG2}, of

$$A_{STG2} = A_{DBR+FHT} = 9 \text{ multipliers \& } 31 \text{ adders} \quad (28)$$

together with a memory component, denoted M_{STG2}, of

$$M_{STG2} = M_{DBR+FHT} \approx \frac{7}{4}P \quad (29)$$

words, whilst the associated time complexity, denoted T_{STG2}, for carrying out the task in such a fashion may be expressed as

$$T_{STG2} = T_{DBR+FHT} \approx \frac{1}{16}(L\times P)\times(2\log_4 P+1) \quad (30)$$

clock cycles. Note, however, that each TSM may be used for the storage of both the input and the output data sets to/from the RFHT, as the input data woctads may be simply overwritten by the corresponding output data woctads given that the input data set is accessed by the PE from the PDM rather than the TSM.

The reasoning behind the RFHT design was that a solution should be found to the problem of computing the DHT and/or the real data DFT which possessed a *regular structure*, for ease of implementation, and met with a given latency constraint – namely that the throughput rate should be able to keep up with the data set refresh rate of P clock cycles for each length P data set – whilst using minimal silicon resources. When applied to the real data radix-4 FFT, the RFHT-based solution showed itself able to achieve the computational density of the most advanced commercially available FFT solutions for just a fraction of the silicon resources with the arithmetic requirement of the generic double butterfly being equivalent to that achievable for the butterfly of an optimally designed complex data radix 4 FFT [24].

5. Transform-Space Processing Requirement

The third and final stage of the processing chain involves the identification of the signal's dominant frequencies together with the subsequent computation of the sparse spectrum for those particular frequencies. Before proceeding, however, it is first necessary that each RFHT output data set – as produced from the second stage of the processing chain – is converted from Hartley-space to Fourier space so that it's in the required form for subsequent processing. The squared magnitudes of the Fourier-space outputs – representing the PSD – are also computed for

use in determining the locations of the dominant FFT bins.

5.1 Hartley-Space to Fourier-Space Conversion and PSD Estimation

From Eqtns. 7 and 8 it was seen how the real and imaginary components of the DFT outputs could be obtained straightforwardly from the DHT outputs, with the construction of each set of four consecutive complex-valued DFT outputs needing access to eight real-valued DHT outputs – four with consecutive positive indices and four with consecutive negative indices. Given that each RFHT may have its output data set stored within its own version of the TSM each of the eight memory banks will contain at most two of the required eight samples from the RFHT output data set, one corresponding to the positive index and the other corresponding to the negative index. Given the dual-port nature of the memory, this means that the RFHT output data required for the construction of each set of four DFT outputs, as given by Eqtns. 7 and 8, may be obtained within a single clock cycle. Thus, with fine-grained pipelined processing, it is possible that with eight adders operating in parallel upon the RFHT outputs, each set of four consecutive SDRD-FFT outputs may be constructed whilst the next set of eight RFHT outputs is being accessed, so that each set of complex-valued SDRD-FFT outputs (there are P/2 of these produced from each set of P RFHT outputs) may be produced from the set of real-valued RFHT outputs in just P/8 clock cycles.

Note that to maintain this computational throughput in an efficient manner it is necessary, for each RFHT, that the subsequent SDRD-FFT outputs are efficiently stored, as they are produced, in the existing TSM. This may be achieved by having, for each RFHT output woctad, the set of four positive index outputs overwritten by the real components of the corresponding set of four SDRD-FFT outputs and the set of four negative index outputs overwritten by the imaginary components.

The space-complexity required for carrying out the conversion of all L sets of the transform outputs from Hartley-space to Fourier-space in a fine-grained parallel, coarse-grained sequential fashion, involves an arithmetic component, denoted A_{CON}, of

$$A_{CON} = 0 \text{ multipliers } \& \text{ 8 adders} \tag{31}$$

together with a zero memory component. The associated time complexity, denoted T_{CON}, for carrying out the conversion routine for all L Hartley-space data sets in such a fashion may be expressed as

$$T_{CON} \approx \tfrac{1}{8}(L \times P) \tag{32}$$

clock cycles.

Turning to the PSD estimation, it was seen from Eqtn. 9 how the PSD estimates could be obtained straightforwardly in terms of either the DHT or the DFT outputs. Thus, with fine grained pipelined processing, it is possible that with eight multipliers operating in parallel upon the real and imaginary components of the DFT outputs followed by four adders operating in parallel upon the resulting squared terms, each set of four consecutive PSD outputs may be constructed and stored within its own partitioned power spectrum memory (PSM) – which consists of eight equal sized memory banks each comprising Q/2 words – whilst the next set of four SDRD-FFT outputs are being produced, as illustrated in Fig. 5, so that apart from a short start-up delay for the computational pipeline, the sets of complex-valued SDRD-FFT outputs and real-valued PSD outputs, each of length P/2, may both be produced in just P/8 clock cycles.

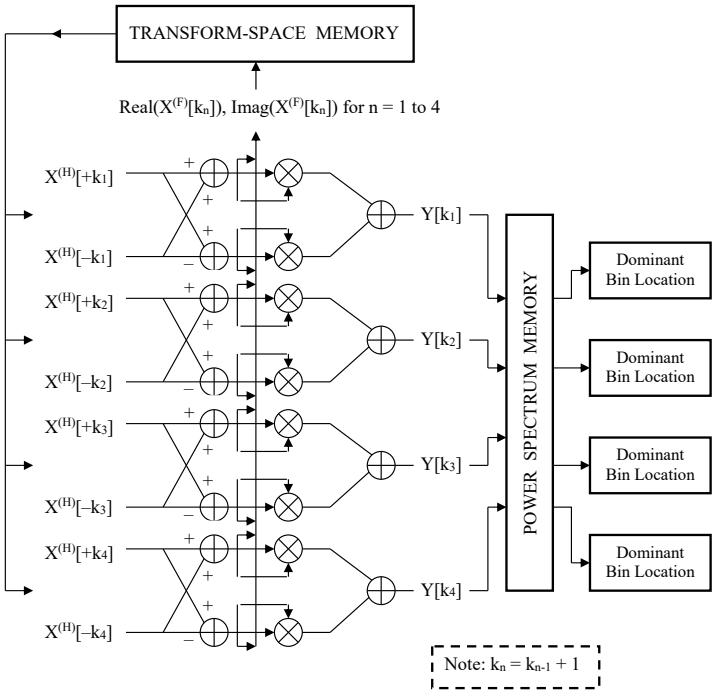

Figure 5: Signal Flow Graph for Derivation of Fourier-Space and Power Spectrum Data + Dominant Bin Location Using Four Modules Operating In Parallel

The space-complexity required for carrying out the PSD estimation for all L Fourier-space data sets in a fine-grained parallel, coarse-grained sequential fashion, involves an arithmetic component, denoted A_{PSD}, of

$$A_{PSD} = 8 \text{ multipliers } \& \text{ 4 adders} \quad (33)$$

together with a memory component, denoted M_{PSD}, of

$$M_{PSD} \approx \tfrac{1}{2}(L \times P) \quad (34)$$

words for the partitioned PSMs. Due to the pipelining of the operations required for carrying out both the conversion routine and the PSD estimation, the associated time complexity for carrying out the PSD estimation for all L Fourier-space data sets will be 'effectively' zero, apart from that involving the small start-up delay required for the pipelining of the fast multipliers.

At this point L sets of both Fourier-space and PSD data are available within the partitioned TSMs and PSMs, respectively, for input to the remaining key tasks required of this final stage of the processing chain concerning the derivation of the individual components of the sparse spectrum. The overall space complexity required for obtaining these two data sets in a fine-grained parallel, coarse grained sequential fashion, is given by an arithmetic component of

$$A_{CON+PSD} = 8 \text{ multipliers } \& \text{ 12 adders} \quad (35)$$

together with a memory component of

$$M_{CON+PSD} \approx \tfrac{1}{2}(L \times P) \quad (36)$$

words, whilst the associated time-complexity for carrying out the combined task for all L data sets in such a fashion may be expressed as

$$T_{CON+PSD} \approx \tfrac{1}{8}(L \times P) \quad (37)$$

clock cycles, where each of the above data sets is stored within its own partitioned memory.

5.2 Location of Dominant SDRD-FFT Bins

Given that the maximum number of SDRD-FFT bins, 'K_D', considered to contain a detectable signal component is such that $K_D << P << N$ and $K_D|P$, the task now is to determine the locations of those dominant SDRD-FFT bins, for each of the L spectral data sets, from examination of the associated PSD outputs – as illustrated in Fig. 5. To achieve this, a full sort routine could simply be used, but this would involve a time complexity of $O(P \times \log_2 P)$ clock cycles which would be unnecessarily complex for the problem being addressed [10,22]. A partial sort routine would be a more appropriate solution for the problem at hand with an attractive means of achieving this being to create a 'Heap' data structure – or, more specifically, a 'Min Heap' data structure – which for every iteration would keep track of the largest K_D values and their addresses from the currently processed PSD set and would enable the time complexity to be reduced to just $O(P \times \log_2 K_D)$ clock cycles – a reduction of $O(\log_2 P/\log_2 K_D)$ [4].

The approach is thus to create a Min-Heap data structure which will yield the largest K_D PSD outputs using the two heap operations of 'insert' and 'delete', each of which may be assumed to involve up to $\log_2 K_D$ exchanges yielding a time-complexity of $O(\log_2 K_D)$ clock cycles. After initializing the heap with the first K_D PSD outputs using the insert operation, the partial sort algorithm iterates through the remaining $P/2–K_D$ outputs comparing each new value with the existing minimum value of the heap. If the value is less than the existing minimum value then the processing for that iteration terminates, whereas if the value is greater than the existing minimum value then that minimum value is deleted and the new value inserted into the heap. After all the iterations have been completed the Min-Heap will contain the values of the K_D largest PSD outputs, together with the corresponding PSD addresses within the PSM – noting that the corresponding SDRD-FFT output has the same address within the TSM as the PSD output within the PSM. The heap data – which is updated and stored within the dominant bin memory (DBM) – consists of K_D pairs of numbers where the first number of each pair is the value of a dominant PSD output whilst the second number is its address within the PSM. The addresses are then used to set up a binary indicator array (BIA), of length P/2, where the presence/absence of a '1' indicates the presence/absence of a dominant signal component residing at that SDRD-FFT index.

Thus, with a sequential approach, where the insert and delete operations are each assumed to be achievable in at most $\alpha \times \log_2 K_D$ clock cycles, for some factor 'α', the initialization of the heap may be carried out in at most $\alpha \times K_D \times \log_2 K_D$ clock cycles, whilst the remaining $P/2–K_D$ iterations of the partial sort algorithm may be carried out in at most $2\alpha \times (P/2–K_D) \times \log_2 K_D$ clock cycles, although this figure will only be realized with the pathological situation where successive PSD outputs occur in a monotonically increasing fashion whereby each new iteration requires that the existing minimum value be deleted from the heap and the new value inserted. A more realistic situation involves using randomly generated values for the PSD outputs so that the K_D maximum values may be assumed to be randomly distributed across the spectrum, as anticipated with the permuted spectral data. This situation has been modelled in Monte-Carlo fashion using parameter values of P = 16K (where 1K = 1024) and K_D = 512, with the results showing that the maximum number of inserts and deletes to be made in the second iterative phase of the algorithm never exceeded 20% of the maximum number possible (which is 1536 for the chosen parameter set), suggesting that an attractive low-complexity solution would be to terminate the processing once such a limit has been reached. The occasional loss of a detectable signal component would seem a reasonable compromise to make in order to ensure a realizable solution and could be overcome through the subsequent averaging over several consecutive sparse spectral output data sets, a commonly adopted practice used to reduce the variance.

As a result, the space-complexity for carrying out this task for all L sets of PSD outputs in a fine grained sequential, coarse-grained sequential fashion, possesses a zero arithmetic component together with a memory component, denoted M_{LOC}, of

$$M_{LOC} \approx 2(L \times K_D) + (L \times P/2W) \quad (38)$$

words, where the dominant PSD values and addresses occur in pairs within each DBM. The associated (worst-case) time complexity, denoted T_{LOC}, (where maximum allowable number of inserts and deletes to be made in second phase of the algorithm – at which point the processing is terminated – is taken to be 20% of maximum possible) for carrying out the task in such a fashion may be expressed as

$$T_{LOC} \approx \alpha/5 (L \times (P + 3K_D)) \times \log_2 K_D \quad (39)$$

clock cycles, where the set of addresses for each set of dominant PSD outputs is stored within its own BIA.

5.3 Determination of Dominant Signal Frequencies

At this point the locations of the K_D dominant bins are now known for each of the L sets of SDRD FFTs outputs where, for each such bin, there corresponds a set of N/P FOIs whose indices within the full N/2 point NAT reordered spectrum – which thus correspond to actual frequencies – are as given by Eqtn. 11. The task now is to determine, in a computationally efficient manner, the best frequency estimates from those available that correspond to the presence of detectable signal components in all of the L SDRD FFT output data sets – where each set is stored within its own version of the TSM. This equates, in 'set theoretic' terms, to finding the intersection of the L sets where each set contains the indices of $K_D \times N/P$ FOIs, with a total of just K_D frequency indices being actually sought. Processing all combinations of FOIs across all L sets of N/P indices in a brute-force set theoretic manner would however be computationally unrealistic as the time complexity involved would prove prohibitively large. The task is thus to produce a simplified solution able to identify and discard invalid FOIs (namely, those that do not correspond to valid signal components) as soon as they are encountered so as to avoid unnecessary frequency comparisons.

One way to achieve this would be to successively compute, using the recursive form of the mapping Φ of Eqtn. 17, the FOIs corresponding to the K_D sets of N/P permuted frequency indices for the first SDRD FFT and to test whether each FOI, as it is produced, lies within a dominant bin for each of the remaining L-1 SDRD FFT output sets. This testing process may be achieved, for each SDRD FFT, through the application of the inverse mapping Φ^{-1} of Eqtn. 14 to each FOI using the appropriate versions of the parameter σ^{-1} (given that each SDRD FFT output set will have been derived using a different value for the parameter σ and may be carried out in a pipelined fashion by means of an (L-1)-stage filter as illustrated in Fig. 6, where the m'th stage of the filter/pipeline uses the function Φ^{-1} together with the parameter σ_{m+1}^{-1} to determine within which SDRD-FFT bin, if any, the FOI belongs [2]. As soon as an FOI fails to fall within a dominant bin for a given SDRD FFT output set – as determined through comparison with the appropriate element of its BIA – it is discarded so that only K_D FOIs should manage to pass successfully through all L-1 stages of the filter with most FOIs being discarded after passing through the first stage of the filter. Each stage of the filter that is successfully traversed forwards the address of the dominant SDRD-FFT bin (as yielded by its BIA), together with previously forwarded addresses followed by the FOI, so that if and when the final stage is successfully traversed all L dominant bin addresses will be available for storage in the frequency address memory (FAM) followed by the corresponding FOI.

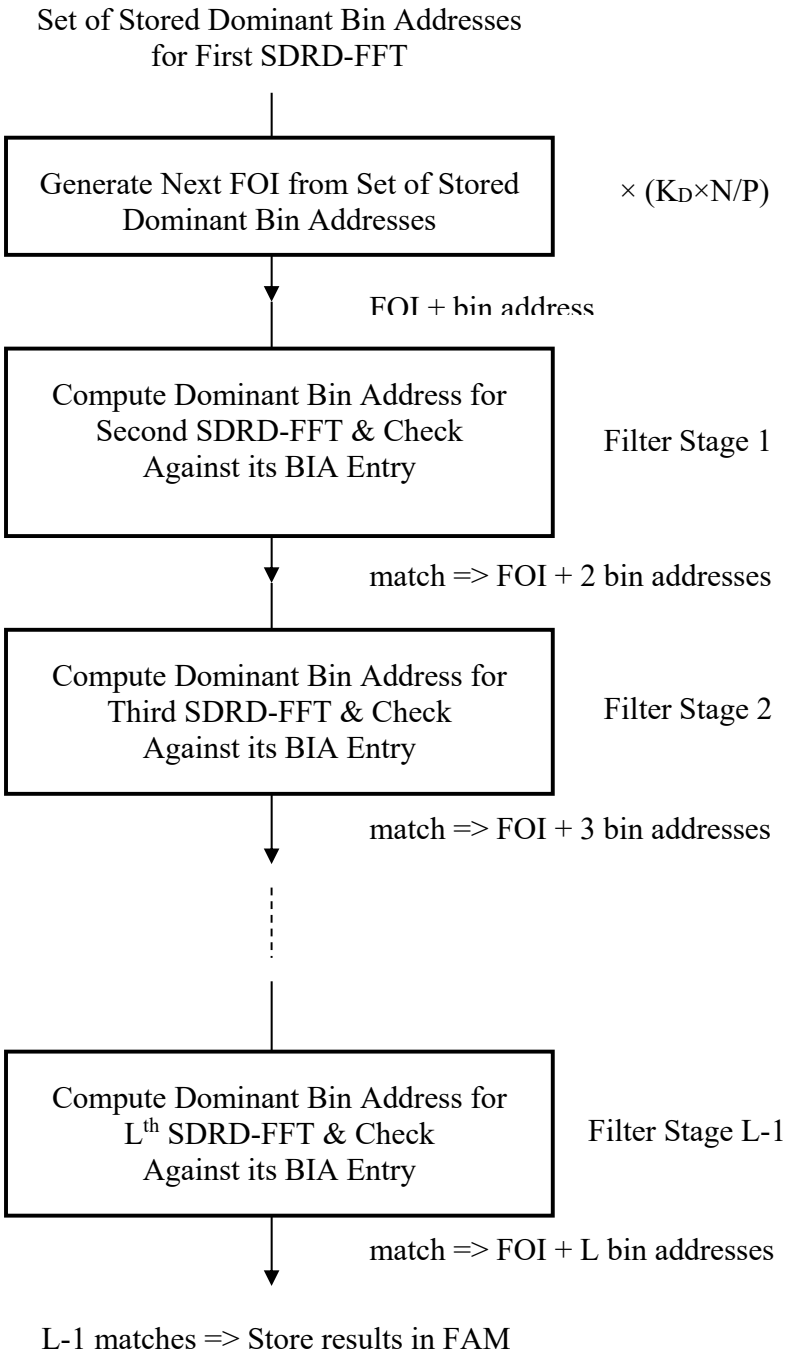

Figure 6: Multi-Stage Filter Used for Determining Dominant Signal Frequencies

The space-complexity for carrying out this multi-stage filtering task comprises an arithmetic component, denoted A_{FRQ}, of

$$A_{FRQ} = \text{L-1 multipliers \& L adders} \quad (40)$$

with the dominant bin address offsets (which are multiples of N/P, a radix-2 integer) being computed via simple left shift operations and each multiplier involving a fixed multiplicand, together with a memory component, denoted M_{FRQ}, of

$$M_{FRQ} \approx K_D \times (L+1) \quad (41)$$

words, with L addresses and one FOI being allocated to each of the K_D dominant signal components. The associated time complexity, denoted T_{FRQ}, for carrying out the task in such a fashion may be expressed as

$$T_{FRQ} \approx (K_D \times N/P) + \beta (L-1) \quad (42)$$

clock cycles, where the first part of the expression corresponds to the recursive generation of the FOIs (Eqtn. 17) and the second part to the filter delay. The term ' β' is a small integer that represents the length of each filter stage's internal pipeline, as required for computing the dominant bin address via the inverse mapping of Eqtn.14 followed by the looking up of the appropriate element of the BIA and the subsequent forwarding (where appropriate) of the dominant bin addresses.

5.4 Determination of Sparse Spectrum Components

At this point in the processing chain: 1) L sets of complex-valued SDRD-FFT outputs have been produced and stored in the partitioned TSMs (as described in Section 5.1), each set being of length P/2, together with 2) the spectral locations of the K_D dominant outputs, as stored in the BIAs, for each such set (as described in Section 5.2) and 3) the values of the corresponding K_D signal frequencies together with, for each, the associated set of L spectrum addresses, one per SDRD-FFT output set, as stored in the FAM (as described in Section 5.3). Thus, the value (both real and imaginary components) and frequency index of the K_D dominant signal components for each of the L spectral data sets are now available for further processing. For each frequency index, the L addresses preceding it within the FAM are used to access the SDRD FFT outputs stored within the TSMs. The L real components are then averaged to yield a representative value for the real component of the sFFT at that frequency, whilst in similar fashion the L imaginary components are averaged to yield a representative value for the imaginary component. The processing scheme for carrying out the task may be expressed via a computational pipeline, as illustrated in Fig. 7 for the case where the parameter L has a value of 8 so that the pipeline is of length $\log_2 L = 3$. The K_D pairs of real and imaginary spectral components produced by the pipeline are then stored in the sparse spectrum memory (SSM), together with the frequency index specifying its position within the idealized NAT ordered N/2 point spectrum – as would be obtained with a dense real-data FFT of length N.

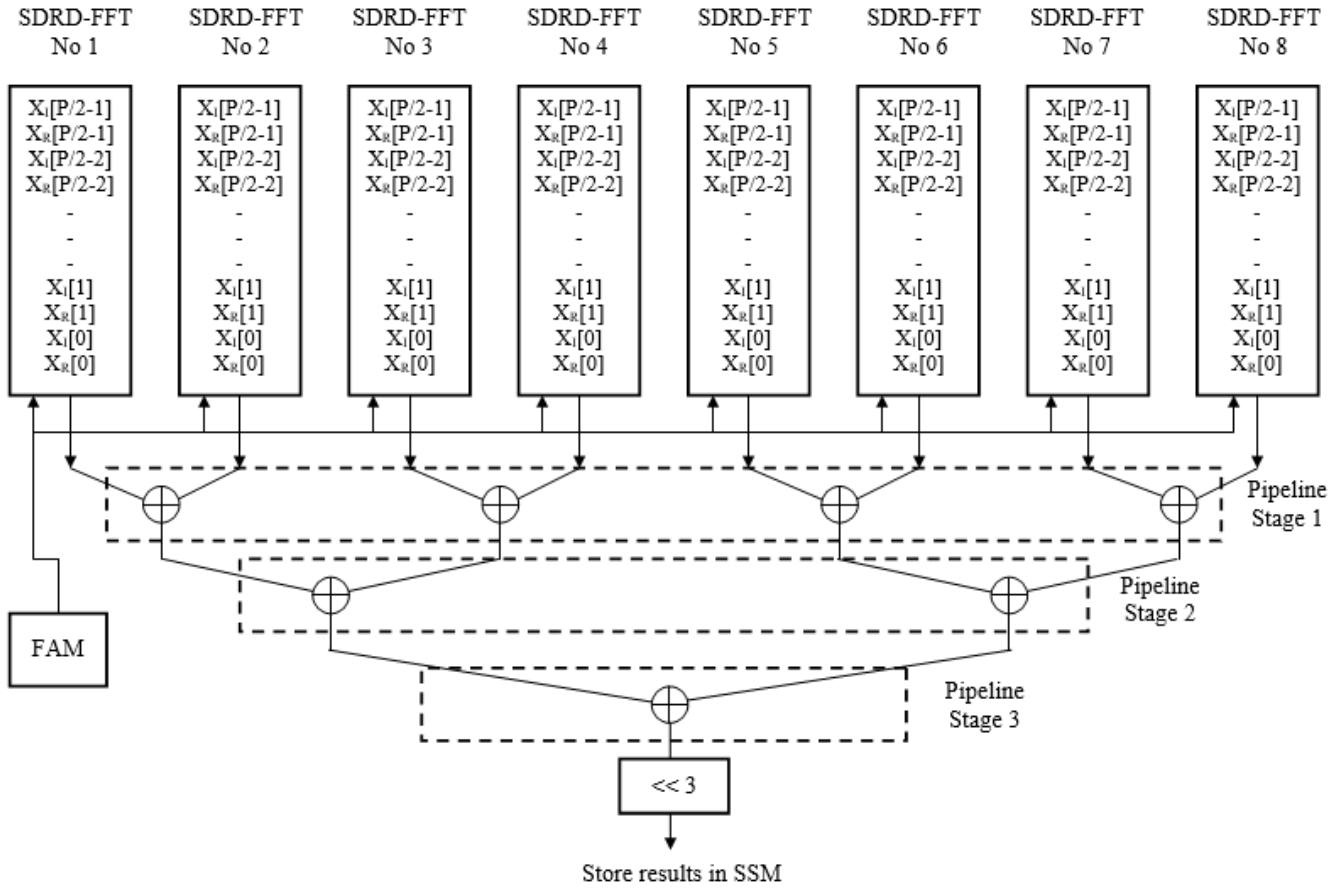

Figure 7: Computational Pipeline for Determining Spectrum Components for Case Where L = 8

The space-complexity for carrying out this task in a pipelined fashion possesses an arithmetic component, denoted A_{SSP}, of

$$A_{SSP} = 0 \text{ multipliers \& L-1 adders} \quad (43)$$

together with a memory component, denoted M_{SSP}, of

$$M_{SSP} \approx 3K_D \quad (44)$$

words, this involving real and imaginary components of the sFFT output together with the associated spectrum address for each of the K_D dominant spectral components. The associated time complexity, denoted T_{SSP}, for carrying out the task in such a fashion may be expressed as

$$T_{SSP} \approx 2K_D + \log_2 L \quad (45)$$

clock cycles – for the simple case where L is a radix-2 integer.

5.5 Summary of Complexity Requirements

The overall space complexity for carrying out the third and final stage of the processing chain involving: 1) the conversion of data from Hartley-space to Fourier space, 2) the PSD estimation, 3) the determination of the dominant signal frequencies, and 4)

the determination of the corresponding spectrum components, involves an arithmetic component, denoted A_{STG3}, of

$$A_{STG3} = A_{CON+PSD}+A_{LOC}+A_{FRQ}+A_{SSP}$$
$$= L+7 \text{ multipliers \& } 2L+11 \text{ adders} \quad (46)$$

together with a memory component, denoted M_{STG3}, of

$$M_{STG3} = M_{CON+PSD}+M_{LOC}+M_{FRQ}+M_{SSP}$$
$$\approx \tfrac{1}{2}(L \times P)+(K_D \times (3L+4))+(L \times P/W) \quad (47)$$

words, whilst the overall time-complexity, denoted T_{STG3}, may be expressed as

$$T_{STG3} = T_{CON+PSD}+T_{LOC}+T_{FRQ}+T_{SSP}$$
$$\approx \tfrac{\alpha}{5} \times (L \times (P+3K_D)) \times \log_2 K_D$$
$$+ \tfrac{1}{8}(L \times (P+8\beta))+(K_D \times (2+N/P)) \quad (48)$$

clock cycles, these results based (for the most part) upon the processing being carried out in a fine-grained parallel, coarse-grained sequential fashion.

6. Trading off Latency Against Silicon Resources

The previous three sections have provided descriptions of the individual tasks needing to be carried out by the proposed real-data sFFT algorithm, together with their respective space and time complexities when the multiple data sets are processed sequentially. Complexity trade-offs are now considered including how multiple instances of certain tasks might best be carried out in parallel, via the adoption of the single instruction, multiple-data (SIMD) approach which uses multiple identical memories and processing streams to process multiple data sets simultaneously [3]. This will facilitate coarse-grained parallel processing which, together with the fine grained parallelism already achieved via the adoption of partitioned memory and pipelining, will enable latency to be traded off against silicon resources.

6.1 Data-Space Processing + Short Dense Real-Data FFTs

Suppose that the outputs of the first stage of the processing chain, the L WSRG data sets, are produced prior to the execution of the second stage concerning the execution of the RFHTs. Then the time complexity for carrying out the first two stages is simply derived from the combined timing figures of Eqtns. 26 and 30, which may be expressed as $T_{STG1+STG2} = T_{STG1} + T_{STG2}$. In reality, however, the first of the RFHTs (bearing in mind that multiple RFHT modules might be used) may commence processing as soon as there is sufficient data available to process which, if each WSRG data set is stored within its own version of partitioned TSM, is as soon as the first such data set has been produced, namely after P/8 clock cycles, which from the contents of Table 1 is considerably less than the latency of the RFHT for all transform sizes of interest. The last WSRG data set, in turn, will be available for processing after L×P/8 clock cycles, so that $T_{STG1+STG2} \neq T_{STG1}+T_{STG2}$ and the processing for the two stages may be overlapped in order to reduce the overall time complexity.

Suppose, in addition, that the time-complexity is further reduced through the parallel operation of multiple processing streams where each stream involves the processing, via the RFHT, of one or more of the WSRG data sets, so that there would be between 1 and L such processing streams operating independently of each other in an SIMD fashion. The exact number of streams is denoted by parameter 'S_1', where $S_1|L$, so that each stream is assigned the processing of exactly L/S_1 WSRG data sets in an interleaved fashion so that with the case of two streams, for example, one stream might deal with the processing of the even addressed data sets whilst the other deals with the processing of the odd addressed data sets. This enables the time complexity to be expressed via the more general expression

$$T^{(P)}_{STG1+STG2} \approx \tfrac{1}{L}(S_1 \times T_{STG1})+\tfrac{1}{S_1}T_{STG2}$$
$$= \tfrac{1}{8}(S_1 \times P)+\tfrac{1}{16S_1}(L \times P) \times (2\log_4 P+1) \quad (49)$$

clock cycles, where the superscript 'P' refers to the fact that the complexity now reflects the adoption of coarse-grained parallel computation techniques.

The corresponding space-complexity figures for carrying out the first two stages of the processing chain will as a consequence increase to account for the scaling up of the second stage by a factor of S_1, so that

$$A^{(P)}_{STG1+STG2} = A_{STG1}+(S_1 \times A_{STG2})$$
$$= 8+9S_1 \text{ multipliers \& } 31S_1 \text{ adders} \quad (50)$$

and

$$M^{(P)}_{STG1+STG2} = M_{STG1}+(S_1 \times M_{STG2})$$
$$\approx \tfrac{1}{4}(12L+7S_1) \times P \quad (51)$$

words.

6.2 Transform-Space Processing

The parallelization is now extended to cater for the third and final stage of the processing chain, with the space-complexity for the initial conversion of all L sets of RFHT outputs from Hartley space to Fourier-space and of the subsequent production of the PSD data – and using the same level of parallelism as for the computation of the multiple RFHTs – involving a total arithmetic component of

$$A^{(P)}_{CON+PSD} = 8S_1 \text{ multipliers \& } 12S_1 \text{ adders} \quad (52)$$

together with a total memory component of

$$M^{(P)}_{CON+PSD} \approx \tfrac{1}{2}(L \times P) \quad (53)$$

words, whilst the associated time-complexity for the combined task may be expressed as

$$T^{(P)}_{CON+PSD} \approx \tfrac{(L \times P)}{8S_1} \quad (54)$$

clock cycles, where S_1 processes are now assumed to be operating in parallel in SIMD fashion upon the L sets of Hartley space data with L/S_1 data sets being assigned to each process.

For the parallelization of the next task concerning the determination of the dominant bin locations for each of the L SDRD-FFT output data sets, the space complexity – just as for the sequential solution – will possess a zero arithmetic component together with a memory component, denoted $M_{LOC}^{(P)}$, of

$$M_{LOC}^{(P)} \approx L \times (2K_D + (P/2W)) \quad (55)$$

words, whilst the associated (worst-case) time complexity, denoted $T_{LOC}^{(P)}$, may be expressed as

$$T_{LOC}^{(P)} \approx \left(\alpha/5S_2\right) \times (L \times (P + 3K_D)) \times \log_2 K_D \quad (56)$$

clock cycles, where 'S_2' processes are now assumed to be operating in parallel in SIMD fashion upon the L sets of PSD data with L/S_2 data sets being assigned to each process and where S_2 is such that $S_2|L$. Note, however, that a highly-parallel alternative to having multiple sequential solutions running in parallel may be achieved for each data set by pipelining the iterations, in a fine-grained fashion, so that updated versions of the Min-Heap may be produced every O(1) clock cycles [21], rather than every $O(\log K_D)$ clock cycles (as is required for each of the insert and delete operations), at the expense of an $O(\log K_D)$ increase in the space complexity.

For the parallelization of the next task concerning the determination of the signal frequencies, the space complexity will possess an arithmetic component, denoted $A_{FRQ}^{(P)}$, of

$$A_{FRQ}^{(P)} = (L\text{-}1) \times S_3 \text{ multipliers \& } L \times S_3 \text{ adders} \quad (57)$$

together with a memory component, denoted $M_{FRQ}^{(P)}$, of

$$M_{FRQ}^{(P)} \approx K_D \times (L+1) \quad (58)$$

words, whilst the associated time complexity, denoted $T_{FRQ}^{(P)}$, may be expressed as

$$T_{FRQ}^{(P)} \approx (K_D/S_3) \times (N/P) + \beta(L-1) \quad (59)$$

clock cycles, where 'S_3' multi-stage filters are now operating in parallel in SIMD fashion upon the sets of FOIs so that parameter S_3 must be such that $S_3|K_D$ with K_D/S_3 sets, each of N/P FOIs, being assigned to each pipelined process. A memory-efficient implementation that avoids having to replicate the set of BIAs for each version of the multi-stage filter may be achieved by setting S_3 equal to L and processing L consecutive sets of FOIs in parallel where, to avoid addressing conflicts: 1) the BIA used to feed the L multi-stage filters is different for each with the BIA used by the n'th multi-stage filter (where n varies from 1 to L) being set to that defined for the n'th SDRD-FFT, and 2) the BIA used by the m'th stage (where m varies from 1 to L-1) of the n'th multi-stage filter is set to that defined for the k'th SDRD-FFT, where

$$k(m,n) = (m+n-1) \bmod L + 1. \quad (60)$$

In this way, the ordering of the L BIAs is simply rotated by one position with increasing n so that all the BIAs may be used simultaneously for all L-1 stages of the L multi-stage filters. Care needs to be taken, however, to ensure that when all stages of a given filter are successfully traversed the resulting set of L addresses is suitably rotated to enable them to be stored in their correct order.

For the last task of the final stage concerning the determination of the spectrum components, the low space and time complexities for carrying out this task suggest that no additional parallelization is required so that the solution described in Section 5.4 – which consists of a single pipelined process – is retained, this possessing an arithmetic component, denoted $A_{SSP}^{(P)}$, of

$$A_{SSP}^{(P)} = 0 \text{ multipliers \& L-1 adders} \quad (61)$$

together with a memory component, denoted $M_{SSP}^{(P)}$, of

$$M_{SSP}^{(P)} \approx 3K_D \quad (62)$$

words, whilst the associated time complexity, denoted $T_{SSP}^{(P)}$ for carrying out the task may be expressed as

$$T_{SSP}^{(P)} \approx 2K_D + \log_2 L \quad (63)$$

clock cycles – for the simple case where L is a radix-2 integer.

Combining these complexity results, the space-complexity for carrying out the transform-space processing in a coarse-grained parallel fashion involves an arithmetic component, denoted $A_{STG3}^{(P)}$, of

$$\begin{aligned} A_{STG3}^{(P)} &= A_{CON+PSD}^{(P)} + A_{FRQ}^{(P)} + A_{SSP}^{(P)} \\ &= 8S_1 + (L\text{-}1) \times S_3 \text{ multipliers \&} \\ &\quad 12S_1 + (L \times S_3) + (L\text{-}1) \text{ adders} \end{aligned} \quad (64)$$

together with a memory component, denoted $M_{STG3}^{(P)}$, of

$$\begin{aligned} M_{STG3}^{(P)} &= M_{CON+PSD}^{(P)} + M_{LOC}^{(P)} + M_{FRQ}^{(P)} + M_{SSP}^{(P)} \\ &\approx \tfrac{1}{2}(1 + 1/W) \times (L \times P) + (K_D \times (3L+4)) \end{aligned} \quad (65)$$

words, whilst the associated time complexity, denoted $T_{STG3}^{(P)}$, for carrying out the task may be expressed as

$$\begin{aligned} T_{STG3}^{(P)} &= T_{CON+PSD}^{(P)} + T_{LOC}^{(P)} + T_{FRQ}^{(P)} + T_{SSP}^{(P)} \\ &\approx (L \times P)/8S_1 + (K_D/S_3) \times (N/P) + (2K_D + \log_2 L) \\ &\quad + \left(\alpha/5S_2\right) \times (L \times (P + 3K_D)) \times \log_2 K_D \end{aligned} \quad (66)$$

clock cycles.

6.3 Discussion

This section has provided space and time complexity results for parallel versions of all three stages of the processing chain, with the results for the first two stages concerning the data-space

processing and the RFHTs being combined (Eqtns. 49 to 51) due to the overlapping of their operations and those for the last stage concerning the transform-space processing (Eqtns. 64 to 66) being based upon each of the individual tasks being completed before its successor can commence – although complexity results for the first two tasks concerning the data conversion and PSD estimation routines are combined via the pipelining of their operations. As a result, for the case of interest where the parameters L, K_D, P and N are such that $L << K_D << P << N$, the overall time-complexity is of

$$O\left((L\times P)\times\left(\log P/S_1+\log K_D/S_2\right)+K_D\times\left((N/P)/S_3\right)\right) \quad (67)$$

where S_1, S_2 and S_3 are the parallelization parameters, which

$$\rightarrow O\left(K_D\times\left((N/P)/S_3\right)\right) \quad (68)$$

when the performance-related parameters L and P are fixed and the size-related parameters N and K_D are allowed to increase, and

$$\rightarrow O\left((L\times P)\times\left(\log P/S_1+\log K_D/S_2\right)\right) \quad (69)$$

when instead the parameters N and K_D are fixed and the parameters L and P are allowed to increase.

Note also that the block based nature of its operation enables arbitrary large transforms to be realized through the replication of silicon resources by having multiple sFFT's applied to consecutive input data sets, in turn, so as to produce interleaved output data sets.

7. A Detailed Example: 2M-Point Real-Data Sparse FFT

A parameterized model has been produced in MatLab [32] for evaluating the timing and resource requirements which, for a given set of constraints relating to the data set refresh rate (and, equivalently, the update period) and the available silicon resources, enables solutions to be identified which are able to meet those constraints and thus to be actually realized. A typical parameter set is now described, based upon a hypothetical FPGA implementation, consisting of:

1) sampling rate of

$$F_\Delta = 2\times10^9 \text{ Hz} = 2 \text{ GHz}, \quad (70)$$

where the samples are real-valued (assumed here to be 18-bit integers);

2) FPGA clock rate of

$$F_C = 100\times10^6 \text{ Hz} = 100 \text{ MHz}; \quad (71)$$

3) input data set comprising

$$N = 2\times2^{20} \sim 2M \ (2{,}097{,}152) \quad (72)$$

real-valued samples, leading to a real-data sFFT of ~ 2M points (only one half of which are independent, making problem computationally equivalent to 1M-point complex-data sFFT);

4) SDRD-FFT of length

$$P = 16K \quad (73)$$

which after transform-domain conversion yields 8K complex-valued FFT outputs;

5) maximum number of dominant signal frequencies of

$$K_D = 512; \quad (74)$$

6) number of SDRD-FFTs – each carried out by means of the RFHT – to be given by

$$L = 4; \quad (75)$$

and, finally,

7) parameter 'α' relating to the K_D-element Min-Heap data structure of Section 5.2 to be given by

$$\alpha = (\log_2 K_D+2)/\log_2 K_D, \quad (76)$$

so as to yield latency results consistent with those derived for a data structure of length 16K [28] whereby each 'insert' and 'delete' operation was catered for with just 16 clock cycles.

The data set refresh rate, F_R, which has already been defined as the rate at which each new input data set is transferred from the simple ADC-based sampling system to the DSM, dictates in turn the corresponding update period, P_U, which for this example is given by

$$P_U = F_R\times F_C = (N\times F_C)/F_\Delta = 104{,}858 \approx 105\times10^3 \quad (77)$$

clock cycles. Thus, for real-time operation of the proposed block-based solution, this update period must be able to accommodate the execution of the three stages of the processing chain as described in Sections 3 to 5, namely: 1) the derivation of the multiple WSRG data sets, followed by 2) the carrying out of the SDRD FFTs upon the data sets, and 3) the construction of the sparse spectrum from the processing of the resulting sets of SDRD FFT outputs.

The model showed that an attractive solution could be found by setting the parallelization parameters S_1, S_2 and S_3 to values of 2, 4 and 4 (i.e. $S_3 = L$), respectively, where the constraints are such that $S_1|L$, $S_2|L$ and $S_3|K_D$, so that: 1) the data-space processing, followed by the RFHTs, the Hartley-space to Fourier space conversion and the PSD estimation are each partitioned into two parallel processing streams, whilst 2) the dominant bin location and signal frequency estimation of the transform-space processing are each partitioned into four parallel processing streams – as illustrated in Fig. 8. This level of coarse-grained parallelism, combined with the fine-grained parallelism already achieved via the use of partitioned memory, enables the computation of the 2M point real data sFFT to be carried out in an efficient fashion by means of the proposed design with 512 dominant spectrum outputs being produced for each new input data set in ~ 95.8×10^3 clock cycles, which with a 100 MHz clock rate equates to a latency of ~ 0.96 millisecond (ms). This is less than the limit imposed by the update period of Eqtn. 77, as required for a realizable solution, yielding a safety margin of ~ 8.7%. This performance, which involves the production and processing of 32K low resolution spectral samples in order to compute the dominant outputs from the 1M independent outputs

available, is obtained at the expense of 54 fast multipliers (with 12 involving a fixed multiplicand), 105 adders (noting that each adder requires just O(W) logic slices when implemented in logic) and 0.28 Mwords of RAM – the external storage of the input data set as held in the double buffered DSM not being included in these figures.

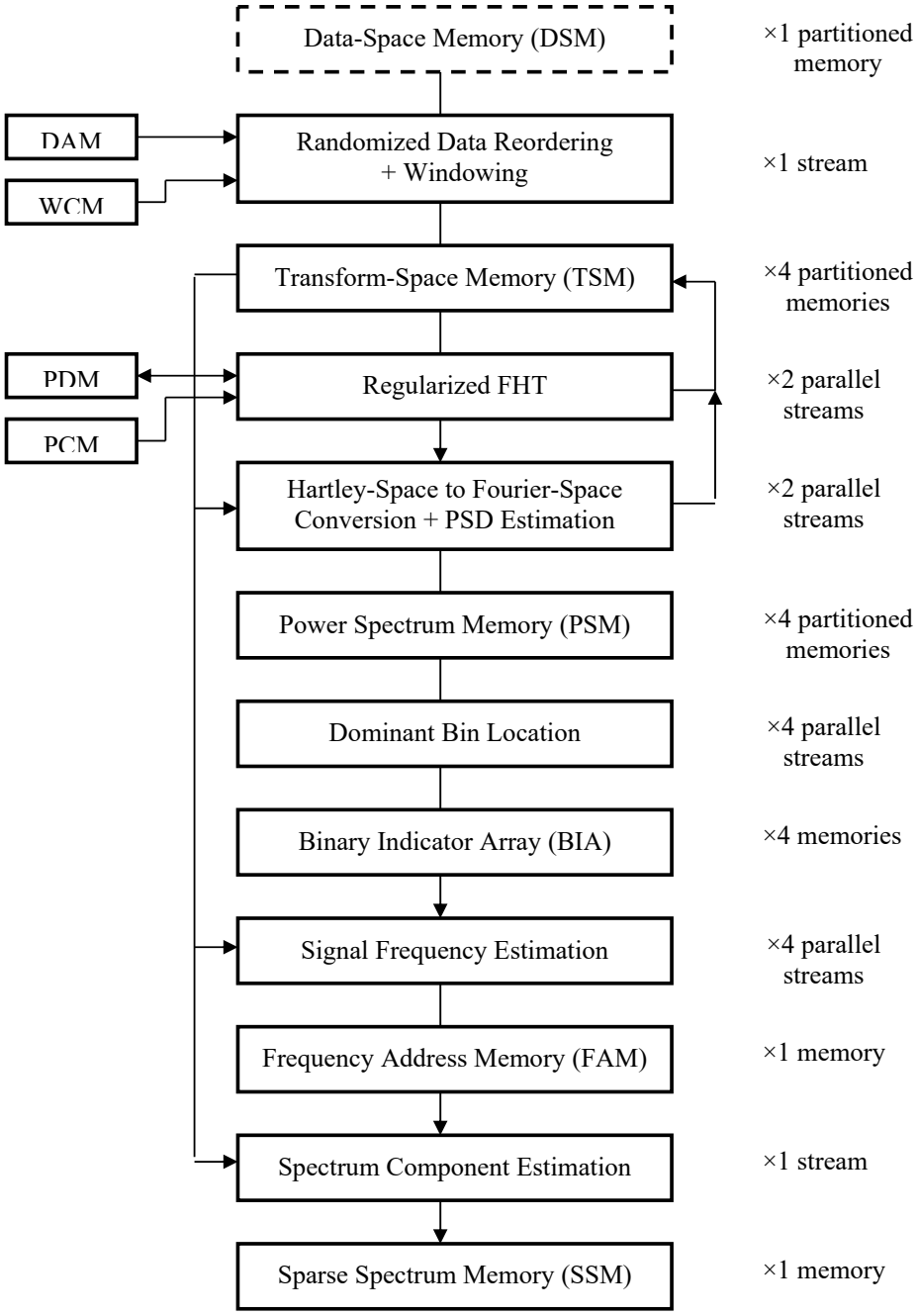

Figure 8: Processing Chain for Carrying out 2M-Point Real-Data sFFT with Parallelization Parameters: $S_1 = 2$ & $S_2 = S_3 = 4$

With the adoption of a low-end Xilinx Vertex-6 FPGA such as the XC6VLX240T device, a total of 768 embedded multipliers would be available together with ~ 0.82 Mwords of 18-bit block RAM (BRAM), so that the fast multiplier requirement of the proposed real data sFFT would involve a utilization figure of ~ 7% and the memory requirement a utilization figure of ~ 34% (with 23% accounting for the multiple DAMs and WCMs). [37]. When the data-space processing (randomized data reordering and windowing) is excluded from these figures – as was the case for the MIT streaming solution of Agarwal et al – the fast multiplier requirement reduces to a utilization figure of ~ 5.5% and the memory requirement (still including access to TSMs) to ~ 19.5%, with a latency of ~ 0.92 ms [2]. With the same device and clock rate, the Agarwal solution (which, like the proposed solution, involves the production and processing of 32K low resolution spectral samples in order to cater for the same sized spectrum of ~ 1M independent samples) achieves a latency of ~ 1.39 ms and an update time of ~ 1.16 ms at the expense of a fast multiplier utilization of ~ 16% and a memory utilization of ~ 26% – see Table 2.

Solution	Space-Complexity (Utilization %)		Time-Complexity (clock cycles × 10^3)		Clock Rate (MHz)
	Multipliers	Memory	Latency	Update Time	
Proposed sFFT Solution [1] ~ 2M points, real data Short Dense FFTs + Transform-Space Processing	~ 5.5	~ 19.5	~ 93 (~ 0.93 ms)	~ 93 (~ 0.93 ms)	100
Proposed sFFT Solution [1] ~ 2M points, real data # Data-Space Processing # + Short Dense FFTs + Transform-Space Processing	~ 7	~ 34	~ 97 (~ 0.97 ms)	~ 97 (~ 0.97 ms)	100
Agarwal sFFT Solution [1] ~1M points, complex data Short Dense FFTs + Transform-Space Processing	~ 16	~ 26	~ 139 (~ 1.39 ms)	~ 116 (~ 1.16 ms)	100
Kanders FFT Solution [2] ~1M points, complex data	~ 4	~ 67	-	~ 1000 (~ 4.29 ms)	233
Kamazaki FFT Solution [3] ~1M points, complex data	high	high	~ 500 (~ 4.00 ms)	-	125

[1] Vertex-6 FPGA XC6VLX240T: Resources = 768 embedded multipliers & ~ 0.82 Mwords of BRAM
[2] Vertex Ultrascale FPGA XVCU095: Resources = 768 embedded multipliers & ~ 1.69 Mwords of BRAM
[3] Vertex-4 FPGA XC4VLX60: Resources = 64 embedded multipliers & ~ 0.16 Mwords of BRAM

Table 2: Comparative Performance Figures for Implementation of sFFT & FFT FPGA Based Solutions

Thus, for the simplified problem (involving data-space to transform-space conversion and the transform-space processing only) discussed above, the proposed real-data sFFT solution, when compared to the Agarwal solution, is able to achieve a *latency reduction* of ~ 33% whilst at the same time achieving *reductions in resource utilization* of ~ 66% for the multipliers and ~ 25% for the memory – although the associated logic requirement cannot be properly assessed without information derived from a real world implementation. Note, however, that in comparing the two solutions, whereas the Agarwal solution uses eight short dense FFTs, each of length 4K, the proposed solution uses four such FFTs, each of length 8K (as obtained from the 16K Hartley space outputs), so that the proposed solution possesses the attraction of an extra 3 dB of SNR in the low resolution spectral data (due to increased coherent gain) but at the expense of a doubling of the variance in the real and imaginary components of the resulting sFFT outputs. At the time of publication, however, the Agarwal solution appeared to offer significant improvements over existing sFFT software implementations such as the multi-threaded software implementation of that required 100 ms for addressing the same size of problem [2].

Finally, for the purposes of comparison, Table 2 outlines the performances of the dense 1M-point complex-data FFTs of: 1) Kanders et al which shows itself able to produce a full 1M-point spectrum but, compared to the proposed solution, requires approximately 4 times the memory requirement and 7/3 times the clock rate whilst achieving an update time that's approximately 4 times longer; and 2) Kamazaki et al [26], which also shows itself able to produce a full 1M-point spectrum but, compared to the proposed solution, requires 5/4 times the clock rate whilst achieving a latency that's approximately 4 times longer – the small amount of resources available on the chosen device also suggests that the utilization must be particularly high [26,27]. An additional comparison may be made with the dense 1M point complex-data FFT of Han et al [16] which requires the benefits (and far greater expense) of an application-specific integrated circuit (ASIC) implementation [33], with comparable 40 nm technology and a 500 MHz clock rate, in order to achieve a high

computational density yielding a latency of 14.8 ms [16,33].

8. Summary and Conclusions

The aim of the research described in this paper has been to produce a flexible and scalable design for the real-data sFFT that's able to yield resource efficient low power solutions, when implemented with silicon-based computing technology, this being achieved by maximizing the computational density through the exploitation of both partitioned memory and the real valued nature of the data. The parameterization of the design enables different problem sizes and requirements to be simply catered for through the replication of silicon resources (i.e. multiple memories and processing modules), rather than through changes to the basic design, resulting in a flexible and easily modifiable solution. A theoretical analysis has shown that with a low-end FPGA device it would be possible, with frequency sparse data comprising ~ 2M real-valued samples, for the frequencies and values of the 512 dominant spectral components to be determined via the proposed sFFT in < 1 ms whilst maintaining low resource utilization. Such a performance would appear to compare favourably – in terms of fast multiplier and memory utilization, together with latency – with other recently published FFT and sFFT silicon-based solutions.

References

1. Abo-Zahhad, M. M., Hussein, A. I., & Mohamed, A. M. (2015). Compressive sensing algorithms for signal processing applications: A survey. International journal of communications, network and system sciences, 8(06), 197.
2. Agarwal, A., Hassanieh, H., Abari, O., Hamed, E., & Katabi, D. (2014, September). High-throughput implementation of a million-point sparse Fourier transform. In 2014 24th International Conference on Field Programmable Logic and Applications (FPL) (pp. 1-6). IEEE.
3. Akl, S. G. (1989). The design and analysis of parallel algorithms. Prentice-Hall, Inc.
4. Al-Jaloud, E., Al-Aqel, H., & Badr, G. (2014). Comparative performance evaluation of heap-sort and quick-sort algorithms. International Journal of Computing Academic Research, 3(2), 39-57.
5. Birkhoff, G., & Mac Lane, S. (2017). A survey of modern algebra. CRC Press.
6. Bracewell, R. N. (1986). The Hartley transform. Oxford University Press, Inc.
7. Bruun, G. (1978). Z-transform DFT filters and FFT's. IEEE Transactions on Acoustics, Speech, and Signal Processing, 26(1), 56-63.
8. Cheng, C., & Yu, F. (2015). An optimum architecture for continuous-flow parallel bit reversal. IEEE Signal Processing Letters, 22(12), 2334-2338.
9. Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. Mathematics of computation, 19(90), 297-301.
10. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to Algorithms, McGraw-Hill.
11. Duhamel, P., & Vetterli, M. (1987). Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data. IEEE Transactions on Acoustics, Speech, and Signal Processing, 35(6), 818-824.
12. Duhamel, P., & Vetterli, M. (1990). Fast Fourier transforms: a tutorial review and a state of the art. Signal processing, 19(4), 259-299.
13. Foucart, S., & Rauhut, H. (2013). A Mathematical Introduction to Compressive Sensing, Birkhauser.
14. Garrido, M., Parhi, K. K., & Grajal, J. (2009). A pipelined FFT architecture for real-valued signals. IEEE Transactions on Circuits and Systems I: Regular Papers, 56(12), 2634-2643.
15. Gilbert, A. C., Indyk, P., Iwen, M., & Schmidt, L. (2014). Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data. IEEE Signal Processing Magazine, 31(5), 91-100.
16. Han, F., Li, L., Wang, K., Feng, F., Pan, H., Zhang, B. & Lin, J. (2016). An ultra-long FFT architecture implemented in a reconfigurable application specified processor. IEICE Electronics Express, 13(13).
17. Harris, F. J. (1978). On the use of windows for harmonic analysis with the discrete Fourier transform. Proceedings of the IEEE, 66(1), 51-83.
18. Hartley, R. V. (1942). A more symmetrical Fourier analysis applied to transmission problems. Proceedings of the IRE, 30(3), 144-150.
19. 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.
20. Hassanieh, H., Indyk, P., Katabi, D., & Price, E. (2012, May). Nearly optimal sparse Fourier transform. In Proceedings of the forty-fourth annual ACM symposium on Theory of computing (pp. 563-578).
21. Ioannou, A., & Katevenis, M. G. (2007). Pipelined heap (priority queue) management for advanced scheduling in high-speed networks. IEEE/ACM Transactions on Networking, 15(2), 450-461.
22. Jmaa, Y.B., Atitallah, R.B., Duvivier, D., & Jemaa, M.B. (2019). A comparative study of sorting algorithms with FPGA acceleration by high level synthesis. Computación y Sistemas, 23(1), 213-230.
23. Jones, K. J. (2006). Design and parallel computation of regularised fast Hartley transform. IEE Proceedings-Vision, Image and Signal Processing, 153(1), 70-78.
24. 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.
25. Jones, K. J. (2021). The Regularized Fast Hartley Transform: Low-Complexity Parallel Computation of the FHT in One and Multiple Dimensions. Springer Nature.
26. Kamazaki, T., Okumura, S.K., Chikada, Y., Okuda, T., Kurono, Y., Iguchi, S. & Sano, R. (2012). Digital spectro-correlator system for the Atacama Compact Array of the Atacama Large Millimeter/Submillimeter Array. Publications of the Astronomical Society of Japan, 64(2), 29.
27. 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.
28. Katevenis, M., & Ioannou, A. (2001, June). Pipelined Heap (priority queue) Management for Advanced Scheduling in High Speed Networks. In IEEE Int. Conf. on Communications (lCC2001).
29. Li, W., Yu, F., & Ma, Z. (2015). Efficient circuit for parallel bit reversal. IEEE Transactions on Circuits and Systems II: Express Briefs, 63(4), 381-385.
30. López-Parrado, A., & Velasco-Medina, J. (2016). Algorithm for wideband spectrum sensing based on sparse Fourier transform. DYNA, 83(198), 79-86.
31. Martens, J. B. (1984). Discrete Fourier transform algorithms for real valued sequences. IEEE transactions on acoustics, speech, and signal processing, 32(2), 390-396.
32. www.mathworks.com.
33. Maxfield, C. (2004). The design warrior's guide to FPGAs: devices, tools and flows. Elsevier.
34. Niven, I., Zuckerman, H. S., & Montgomery, H. L. (1991). An introduction to the theory of numbers. John Wiley & Sons.
35. Schumacher, J., & Püschel, M. (2014, October). High-performance sparse fast Fourier transforms. In 2014 IEEE Workshop on Signal Processing Systems (SiPS) (pp. 1-6). IEEE.
36. Volder, J. E. (1959). The CORDIC trigonometric computing technique. IRE Transactions on electronic computers, 8(3), 330-334.
37. www.xilinx.com.