Digital Signal Processing ••• (••••) •••–•••

Contents lists available at ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

Recursive sliding discrete Fourier transform with oversampled data

A. van der Byl *, M.R. Inggs

Department of Electrical Engineering, University of Cape Town, Rondebosch 7701, South Africa

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Discrete Fourier transform
Recursive discrete Fourier transform
Sliding discrete Fourier transform
Running Fourier transform
Spectral updating

ABSTRACT

The Discrete Fourier Transform (DFT) has played a fundamental role for signal analysis. A common application is, for example, an FFT to compute a spectral decomposition, in a block by block fashion. However, using a recursive, discrete, Fourier transform technique enables sample-by-sample updating, which, in turn, allows for the computation of a fine time–frequency resolution. An existing spectral output is updated in a sample-by-sample fashion using a combination of the Fourier time shift property and the difference between the most recent input sample and outgoing sample when using a window of finite length. To maintain sampling-to-processing synchronisation, a sampling constraint is enforced on the front–end hardware, as the processing latency per input sample will determine the maximum sampling rate. This work takes the recursive approach one step further, and enables the processing of multiple samples acquired through oversampling, to update the spectral output. This work shows that it is possible to compute a fine-grained spectral decomposition while increasing usable signal bandwidths through higher sampling rates. Results show that processing overhead increases sub-linearly, with signal bandwidth improvement factors of up to 6.7× when processing 8 samples per iteration.

1. Introduction

The Discrete Fourier Transform (DFT) has played a fundamental role for signal analysis. Traditionally, samples are acquired and processed in a block-processing fashion, where the number of samples required per spectral update is a function of the desired spectral resolution. The result is a delayed output with the time resolution determined by the block capture and processing rate.

To improve time and frequency resolution, we need to either capture and process the block data at a higher rate, or, alternatively, utilise recursive Fourier transform methods [1–4]. Adopting Recursive Sliding discrete Fourier Transform techniques enables sample-by-sample updating with the flexibility of allowing the computation of finer time–frequency resolution. The sample-by-sample updating uses the Fourier time shift property to update an existing spectral output using the most recent input sample. To maintain sampling-to-processing synchronisation, a sampling constraint is enforced on the front–end hardware. The processing latency per input sample will determine the maximum sampling rate permitted to allow an updated output in a single sample period [5].

This work takes this technique one step further, permitting multiple samples gained through sampling rates higher than those permitted for single-input synchronisation, however still achieving synchronous processing (albeit with a marginal penalty). This in turn shows that it is possible to increase the sampling rate with the aim to increase usable signal bandwidths, and still achieve a fine time–frequency decomposition. Section 2 discusses the Recursive DFT and highlights previous work in the field including error correction for finite-bit arithmetic. Section 3 discusses the use of multiple samples for spectral updating, and discusses the costs and benefits of using higher sampling rates.

2. Recursive sliding discrete Fourier transform

The method employed in this study for DFT computation is the Recursive DFT (RDFT), where the RDFT is based on the principle of updating a current output $F[u]$ as new data is added to the input sequence. The addition of a new data point does not imply that the input sequence has to grow in size (from size N to $N+p$, where p is one for a single new input to the sequence), but rather imposes the constraint that a window function of size N is required, which shifts to include the new sample $(N+p)$, and removes the outgoing sample $(N+p-N=p)$.

If the output is known a priori, an update can be computed by utilising the Fourier shift theorem and computing a DFT on the new data that has been added, while removing the influence the outgoing sample had on the output (for a window of size N). The computational cost for the recursive DFT technique is far lower than the FFT $(O(n\log_2 n))$ [1,6], and is shown to improve by $\log_2 n$ [2].

The recursive technique is based on the work of Sherlock [2] and Kamei [7] and computes an updated $F[u]$ by removing the

* Corresponding author.
E-mail addresses: a.vanderbyl@uct.ac.za (A. van der Byl), Michael.Inggs@uct.ac.za (M.R. Inggs).

1051-2004/$ – see front matter
http://dx.doi.org/10.1016/j.dsp.2013.10.008

contribution of the outgoing sample and adding the contribution of the incoming sample using N length window [1,3,8].

Let:

$$f_{out} = f[0] \quad \text{be the outgoing sample} \tag{1}$$

$$f_{in} = f[N] \quad \text{be the incoming sample} \tag{2}$$

The new output can be computed using:

$$F_{new}[u] = \left[F_{current}[u] + \frac{f_{in} - f_{out}}{N} \right] W_N^u \tag{3}$$

where the complex exponential is defined in a more compact form:

$$\exp\left[j\frac{2\pi u}{N} \right] = W_N^u \tag{4}$$

Using this formula, an updated output can be computed for each DFT point based on the difference between the incoming and outgoing samples. A further advantage of this technique is each DFT point can compute an update independently (select frequencies of interest can be computed if desired), and only minimal data (new sample) needs to be transferred to the processing elements used for each DFT point. An FFT could be performed prior to using the recursive DFT (if block data available), or given a data sequence $f[n]$, the output is already known at time $t = 0$ ($F[u] = 0$ $\forall u$). Prior to any data entering the system, it can be assumed that the resulting output $F[u]$ is zero, and therefore can be used as the initial state for the recursive DFT. At the point where $t = N$, the resulting output matches the DFT output $F[u]$ for window of length N. Further samples can now be added and the resulting updated output computed. If the recursive DFT were implemented sequentially, the cost would be in the order of $O(N^2)$, however if concurrency were exploited by using many smaller processing elements, the cost reduces to $O(N)$, for computing a DFT of length N, if N processing elements are used [9].

3. Multi-sample updating

The work discussed in Section 2 only considers a single sample input per iteration (real or complex) based on a sampling rate with period T_s (and defined by the minimal processing time required to compute an update based on a single new sample). Processing of data assumes that the period (T_s) between input samples matches the processing latency of the underlying system computing the update to ensure no loss of information.

It would be beneficial to explore the possibility to capture multiple data samples at a scaled sampling rate ($\frac{T_s}{k}$) within the processing time, and present an updated spectra based not on one sample, but rather on k samples, where k represents a sampling rate scaling factor ($k \in Z$). Eq. (3) expresses a single sample based update in terms of the Fourier time shift property and the current DFT output vector $F_{current}[u]$ for point u. If a higher sampling rate were used, instead of f_{out} and f_{in} representing a single sample, they would represent multiple samples acquired during the processing latency inherent in computing Eq. (3). The total sample pairs represented by f_{out} and f_{in} are stipulated by the value of k, and are processed in the same manner, except for the value of the complex exponential (which is determined as a function of the time shift).

Computing an update using two sets of samples ($k = 2$) involves shifting in two new samples (f_{in_1} and f_{in_2}), and shifting out two samples (f_{out_1} and f_{out_2}) (required to maintain a constant window length). The difference between the incoming and outgoing sample pairs requires multiplication by $W_N^{(s+1)u}$ where $s = 0, 1$ ($s \in Z$), respectively, followed by summation. s will take on two values in this example, as two samples are shifted in and out, and each difference pair requires multiplication by a different phase. Re-writing Eq. (3) for any k:

$$F_{new}[u] = \left[F_{current}[u] \right] W_N^{ku} + \sum_{s=0}^{k-1} \left[\frac{f_{in_{(k-s)}} - f_{out_{(k-s)}}}{N} \right] W_N^{(s+1)u} \tag{5}$$

Two key points should be noted from Eq. (5). Firstly, the existing computed spectrum is now multiplied by a complex exponential influenced by the shifting parameter k. Secondly, the incoming and outgoing samples are handled in pairs, and multiplied by a complex exponential determined relative to the shift the pair of samples experienced. It should also be noted that when computing this step, the inherent parallelism can be exploited minimising the additional overhead required if processing resources permit it. The computation in Eq. (5) is suitable for both finite-bit and floating-point arithmetic, however the use of error correction would be needed to maintain a constant error rate when implementing finite-bit arithmetic. The following section details this inclusion.

3.1. Error correction

The recursive DFT as expressed in Eq. (3) allows for frequent spectral updating when a single sample is added, however, computational errors can accumulate if implemented with finite-bit arithmetic. The error is produced as a result of a quantisation and arithmetic round-off in the complex exponentials used per point, as well as a round-off used in the storage and computation of the DFT update. Furthermore, the error grows without bound due to the recursive nature of the algorithm [3,10].

It is possible to model and track errors as they develop, allowing on-the-fly dynamic error correction per point [11,5]. The correction vector (E_u) for a single sample shift is expressed in Eq. (6) for iteration l at DFT point u [5]:

$$E_u[l+1] = \sigma_u F_{current}[l] + \sigma_u \left[\frac{f_{in} - f_{out}}{N} \right] + W_N^u E_u[l] \tag{6}$$

where:

$$\sigma_u = \hat{W}_N^u - W_N^u \tag{7}$$

and:

W_N^u is the complex twiddle factor and
$\hat{W}_N^u$ is the finite-bit approximation of W_N^u
$F_{current}$ is the current DFT point output
f_{out}, f_{in} are the outgoing and incoming samples

Modifying Eq. (6) to handle multiple input sample pairs produces:

$$E_u[l+1] = \sigma_{ku} F_{current}[l] + \sum_{s=0}^{k-1} \sigma_{(s+1)u} \left[\frac{f_{in_{(k-s)}} - f_{out_{(k-s)}}}{N} \right] + W_N^{ku} E_u[l] \tag{8}$$

where:

$$\sigma_{ku} = \hat{W}_N^{ku} - W_N^{ku} \tag{9}$$

and

$$\sigma_{(s+1)u} = \hat{W}_N^{(s+1)u} - W_N^{(s+1)u} \tag{10}$$

Please cite this article in press as: A. van der Byl, M.R. Inggs, Recursive sliding discrete Fourier transform with oversampled data, Digital Signal Process. (2013), http://dx.doi.org/10.1016/j.dsp.2013.10.008

Table 1
Sequential computational costs for $k = 1, 2, 4, 8$. All processing times shown are relative to $k = 1$, and indicate the increase in overhead when additional sample pairs are computed.

DFT length	32	64	128	256	512
$k = 1$	1	1	1	1	1
$k = 2$	1.11×	1.11×	1.10×	1.11×	1.09×
$k = 4$	1.15×	1.14×	1.14×	1.14×	1.12×
$k = 8$	1.21×	1.20×	1.20×	1.19×	1.27×

To compute an error correction update based on the inclusion of additional sample pairs requires the same set of multiple complex exponentials as used in the spectral update. The difference lies in the use of the remaining error from the previous iteration as well the difference in the complex exponentials required for each shift of the newer samples. Practically, the difference (σ_{ku} and $\sigma_{(s+1)u}$) still requires a finite-bit holding registers, however these registers can permit all bits to be allocated to precision, and therefore are of higher accuracy.

3.2. Processing cost

Assessing and comparing the requirements to compute Eq. (5) compared to Eq. (3) shows that the difference lies in the growth in the number of input–output sample pairs, and the multiplication of the respective complex exponential. The complex exponential is determined by the relative shift each sample pair has experienced, and therefore needs to be computed independently. The summation of all products is later performed and summed to the product of the current spectral output point ($F_{current}[u]$) to complete the update.

The algorithm is inherently parallel, and can be computed as such, with the exception of the final summation stage. The update can be performed using floating-point arithmetic if appropriate resources are available, or alternatively fixed-point processing on platforms such as FPGA's could be used. The discussion from this point will consider only fixed-point computations, and will highlight results achievable in practise.

In our initial work [5,9], we have detailed the flexibility of this algorithm for a single sample input–output pair, and showed a parallel FPGA implementation for fixed-point arithmetic which included dynamic error correction to bound errors. The inclusion of additional samples would require additional processing for the newer sample pairs that are shifted in and out. This additional processing cost could be masked through parallelism if resources permitted, however can also be computed sequentially.

To assess the processing overhead, a worst-case sequential approach was adopted for testing. The computation of each DFT point u was computed using 32-bit signed arithmetic, where 24-bits were reserved for precision, and the error correction threshold fixed at 2^{-12} [5]. Various shift levels were implemented ($k = 1, 2, 4, 8$), and an average computational time was computed per shift value. In this assessment, the time factor of interest is the ratio of processing overhead increase per shift with respect to $k = 1$, due to the additional multiplications and summations required. To form the comparison, the time required to compute an update for $k = 1$ was used as a reference for all subsequent k values. The results are shown in Table 1.

Table 1 shows the computational increase with respect to the processing latency obtained for $k = 1$ (performed sequentially). The processing was performed using Matlab on an Intel i7 processor, using a non-optimal software implementation. Interestingly, the increase in processing cost is sub-linear, indicating a marginal increase in cost overhead when additional samples are processed. It should be noted that increasing the DFT length will increase the processing overhead overall, however the results in Table 1 indicate the relative increase in computing a single DFT point when additional samples are included. It can be seen that the additional cost for various k values remain on par irrespective of the DFT length. An increase in processing overhead is not ideal in most situations, however should be viewed with context of sampling rates and signal bandwidths (Section 3.3 discusses the advantage in higher sampling rates over the increase of computational cost).

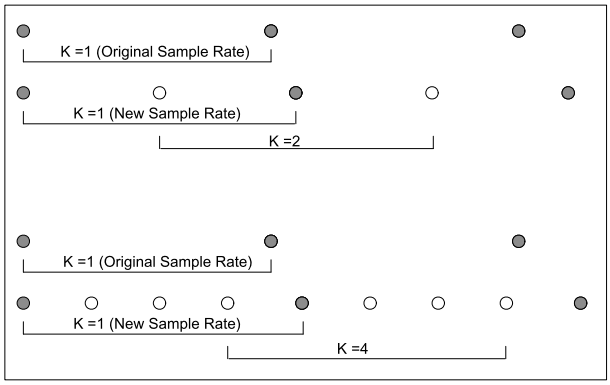

Fig. 1. Sampling rate diagram for $k = 1, 2$ and 4. The solid dots represent sampling at a rate of $k = 1$. The empty dots represent the additional samples acquired for $k = 2$ and $k = 4$.

The increase in processing time for a varied k does not come without limitations. To ensure a buffer overrun does not occur, it is important to make sure the data throughput can be handled at all stages. If the latency increases stipulated in Table 1 are used, data will be lost due to overflow, as the input data rate versus processing the data given the additional sample pairs is not synchronised. To avoid this pitfall, the latency increase for a given value of k needs to be known.

Using an example of $k = 4$, a DFT length of 256 takes approximately 1.14× longer to compute the additional samples. Using actual numbers, if the latency was 1 ms for $k = 1$, the latency would be 1.14 ms for $k = 4$. To ensure consistency, the maximum rate for $k = 1$ should be scaled by the difference of $2 - 1.14 = 0.86$ in this case. Stated differently, the *new* maximum sampling rate for $k = 1$ should be 86% of the original sampling rate for $k = 1$. This ensures that when $k = 4$, the processing time required to produce an output will not exceed the time taken to capture four samples. This concept carries for any k value, and is illustrated in Fig. 1.

3.3. Bandwidth gain

The results discussed in this work have initially concentrated on shifting values of $k = 1, 2, 4, 8$, however, the values for k can extend beyond these values shown here, and are limited by processing hardware capabilities. Focusing on the results shown in Table 1, a value of $k = 1$ indicates the nominal sampling rate possible based on the processing latency to compute an update on one new sample. Higher values for k indicate scaled sampling rates, and are oversampling factors.

The Shannon–Nyquist criterion stipulates that the minimal sampling rate for any band-limited signal should be twice the highest frequency component resident in that signal. Increasing the sampling rate by definition implies that a signal of wider bandwidth may be captured, and in context to this work, increasing k would imply that a signal of wider bandwidth may now be processed provided hardware permits adjustable sampling rates.

This introduces an interesting trade-off as the higher the value of k, the larger the processing overhead, however, the larger k is, the wider the permissible signal bandwidth. To illustrate the signal bandwidth gain, consider $f[n]$, the sum of three linearly sweeping chirp signals: 0–5 MHz; 10 MHz–15 MHz; 25 MHz–35 MHz. In

Please cite this article in press as: A. van der Byl, M.R. Inggs, Recursive sliding discrete Fourier transform with oversampled data, Digital Signal Process. (2013), http://dx.doi.org/10.1016/j.dsp.2013.10.008

this experiment, the value of $k = 1$ equates to a sampling rate of 25 MHz, and hence a maximum permissible signal frequency of 12.5 MHz.

An anti-aliasing filter is first applied to band limit the signal prior to sampling. Changing $k = 2$ implies a new sampling rate of 45 MHz, with a maximum permissible signal frequency of 22.5 MHz (and *not* 50 MHz due to the increase in processing overhead which limits the sampling rate scaling). This dataset is used for all cases of k, and is expressed using a finite-bit set as indicated previously.

Each processing run for $k = 1, 2, 4, 8$ differs only in the cut-off frequency of the anti-aliasing filter, and the number of samples available in the dataset. The number of processing iterations remains constant however, as multiple samples are computed simultaneously for a DFT update. Fig. 2 illustrates for $k = 1, 2, 4$ in (a), (b), and (c) respectively for a DFT length of 256.

Analysing Fig. 2, the three distinct chirp signals become evident as the sampling rate is increased. The three sub-figures (a, b and c) in Fig. 2 show a time–frequency plot of the generated chirp signals, and represent the sampling rate increase for $k = 1, 2, 4$ respectively. As k increases, the cut-off frequency is relaxed, permitting a wider signal bandwidth to be tolerated without aliasing. The benefit in signal bandwidth gain is apparent, however the signal bandwidth improvement with respect to the associated processing cost should be examined. This relationship is discussed in Section 3.4.

It is also worthwhile mentioning the time–frequency plots shown in Fig. 2 (a, b and c). A significant benefit to the recursive sliding discrete Fourier transform is availability of both time and frequency information while analysing a signal. The window used for the transform would not cover all time, and it is possible to resolve changes in frequency spectra as the window shifts over the signal of interest with respect to time. The illustrations in Fig. 2 display the change in frequency as the discrete time window shifts across the signal (linear combination of chirp signals in this case). The window length remains fixed, and only the number of samples that are removed and replaced per iteration changes as the value of k changes. Since the number of samples increases per processing iteration, and the latency does not scale linearly with the increase in k, a wider bandwidth can be processed with a minimal increase in processing cost.

3.4. Signal bandwidth gain-to-cost ratio

The inclusion of additional samples captured at a higher sampling rate provides a useful benefit to any spectral decomposition. Processing each new sample on a sample-by-sample basis provides the highest time–frequency resolution for the given dataset, however comes at a processing cost, as the hardware is required to execute with a latency inversely proportional to acquisition rate.

If a lower time–frequency resolution can be tolerated, and a higher sampling rate required, a trade-off is possible. Viewing this in the light of signal bandwidth improvement, for $k = 4$, which indicates a sampling rate increase by a factor of 3.51, the signal bandwidth can also be improved by a factor of 3.51 at a processing latency cost of approximately 1.14× the latency for processing a single sample input–output pair. Table 2 lists the possible signal bandwidth gain for various k values. It should be noted the cost incurred in the processing overhead will define the signal bandwidth gain in a given system. If the additional cost can be masked through parallelism, the signal bandwidth scaling will more closely match k, providing a closer approximation to linear scaling.

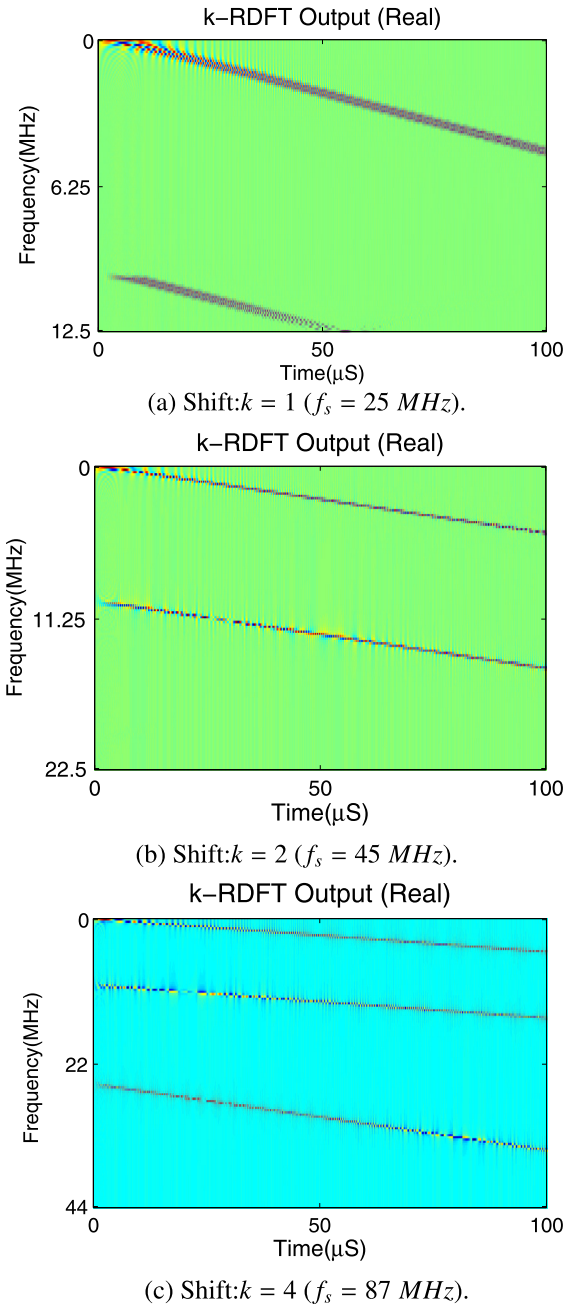

Fig. 2. Time–frequency representation of the three generated chirp signals for $k = 1, 2, 4$. Each plot represents a different sampling rate and signal bandwidth.

Table 2
Bandwidth gain with respect to number of input samples defined by k. As k is increased, the additional overhead places a sampling constraint which prevents the sampling rate scaling linearly.

k	Signal bandwidth improvement
1	1×
2	1.8×
4	3.51×
8	6.72×

4. Conclusion

This paper provides an extension to the theory of recursive Fourier transforms, and proposes a method to frequently update the spectral decomposition of a signal when acquired at rates

Please cite this article in press as: A. van der Byl, M.R. Inggs, Recursive sliding discrete Fourier transform with oversampled data, Digital Signal Process. (2013), http://dx.doi.org/10.1016/j.dsp.2013.10.008

faster than the single sample processing capabilities of the underlying hardware. A single sample update system provides the highest time–frequency resolution, but has a data acquisition constraint implied by the processing throughput. To overcome this boundary, multiple samples may be acquired during the processing latency period and an update computed with increase in overhead in the region of 1.2× the original processing cost. The higher sampling rates in turn enable wider signal bandwidths to be accommodated and analysed, and this work shows the improvement in signal bandwidth out-weighs the additional processing costs.

References

[1] E. Jacobsen, R. Lyons, The sliding DFT, IEEE Signal Process. Mag. 20 (2003) 74–80.
[2] B.G. Sherlock, D.M. Monro, Moving discrete Fourier transform, IEE Proc. 139 (1992) 279–282.
[3] A. Brown, Running Fourier transforms, Electron. World (1998) 959.
[4] S. Assous, L. Linnett, High resolution time delay estimation using sliding discrete Fourier transform, Digit. Signal Process. 22 (2012) 820–827.
[5] A. van der Byl, M.R. Inggs, R.H. Wilkinson, Recursive Fourier transform hardware, in: Radar Conference (RADAR), IEEE, 2011, pp. 746–750.
[6] A.V. Oppenheim, R.W. Schafer, Digital Signal Processing, Prentice Hall, 2002.
[7] H. Kamei, T. Harada, H. Kawarada, A fast algorithm of running Fourier transform, Electron. Commun. Jpn., Part 1, Commun. 71 (1988) 1–10.
[8] E. Jacobsen, R. Lyons, An update to the sliding DFT, IEEE Signal Process. Mag. 21 (2004) 110–111.
[9] A. van der Byl, M.R. Inggs, R.H. Wilkinson, A many processing element framework for the discrete Fourier transform, in: Proceedings of the 2010 International Conference on Field-Programmable Technology, IEEE, 2010, pp. 425–428.
[10] A. van der Byl, A parallel processing framework for spectral based computations, PhD thesis, Department of Electrical Engineering, University of Cape Town, 2012.
[11] J.-H. Kim, T.-G. Chang, Analytic derivation of the finite wordlength effect of the twiddle factors in recursive implementation of the sliding-DFT, IEEE Trans. Signal Process. 48 (2000) 1485–1488.

Andrew van der Byl is a research officer in the Department of Electrical Engineering, University of Cape Town, South Africa. He received his Ph.D. from the University of Cape Town, and his MTech, BTech and ND from the Cape Peninsula University of Technology in Cape Town, South Africa. His research interests include signal and image processing, machine learning, radar and reconfigurable computing.

Michael R. Inggs is a full Professor in the Department of Electrical Engineering, University of Cape Town, South Africa. He received his Ph.D. and DIC from Imperial College London, and his Honours Degree in Applied Mathematics and Physics from Rhodes University, South Africa. His research interests include radar, earth observation using radar, and high performance computing architectures and languages for signal and image processing.

Please cite this article in press as: A. van der Byl, M.R. Inggs, Recursive sliding discrete Fourier transform with oversampled data, Digital Signal Process. (2013), http://dx.doi.org/10.1016/j.dsp.2013.10.008