Real-time signal frequency analysis in variable speed drives using the sparse fast Fourier transform (sFFT)

Mehanathan Pathmanathan, Luca Peretti
ABB Corporate Research, Department of Electrical Systems
Forskargränd 7, 72178 Västerås, Sweden

***Abstract*—This paper investigates the implementation and the use of the sparse fast Fourier transform algorithm in the converter control of a variable speed drive. The algorithm is proposed due to the reduction in computational complexity compared to the conventional fast Fourier transform for the special case of sparse signals. After discussing the theory and a simulation model, experimental results obtained using a field programmable gate array (FPGA) implementation are presented, showing the effectiveness of the proposed solution.**

Keywords—Signal frequency analysis; FPGA; FFT

I. INTRODUCTION

Variable speed drives (VSDs) are commonly used in a wide range of applications. In addition to offering dynamics and efficiency improvements to electrical machines, VSDs can offer other features, through the compensation of disturbances such as torque oscillations [1] or mechanical vibrations [2].

Compensation algorithms require the knowledge of the harmonic content in the signal which is being compensated. This information can be obtained through any frequency analysis algorithm. The mainstream algorithm is the fast Fourier transform (FFT) [3]. In the case of real-time signal frequency analysis, the FFT has to be computed in parallel to the other operations being conducted by the VSD control. It is known that the FFT requires a minimum of $O(N \log N)$ operations to complete, where N is the number of samples in the analysed signal. The higher the value of N, the more difficult it is to execute the FFT in addition to the usual control functions of a VSD.

The sparse FFT (sFFT) [4, 5] was introduced as an extension of the FFT which can be applied to sparse signals. A sparse signal is defined as one with a small number of peaks in its magnitude spectrum. This definition holds for a vast group of signals and variables in a VSD system, and in general for any power electronics related signal. It was demonstrated [4,5] that the sFFT required less operations than a FFT when dealing with sparse signals. This property has been utilised to enhance performances of spectrum sensing [6], magnetic resonance imaging [7], light field photography [8] and GPS [9].

This paper demonstrates how the sFFT algorithm can be exploited in a VSD system, adding the electrical machine control as a possible application area of the sFFT. The proposed implementation makes use of a field programmable gate array (FPGA) to perform the sFFT calculations in parallel with the operations required for a conventional field-oriented control of a synchronous electrical machine.

The paper is organised as follows. Sect. II recalls the fundamentals of FFTs, while Sect. III concentrates on the sFFT with the perspective of a real-time implementation in VSDs. Sect. IV discusses the simulation model. Sect. V shows some experimental results, followed by some conclusive remarks.

II. CONVENTIONAL FOURIER TRANSFORM METHODS

The discrete Fourier transform (DFT) uses discrete-time signals which have a finite time duration, and linearly combines their samples to obtain the frequency spectrum. Since the DFT needs N^2 multiplications, each DFT requires $O(N^2)$ operations to be completed. The FFT is an evolution of the DFT that reduces its computational complexity by eliminating redundant operations. A known FFT implementation is the one by Cooley and Tukey in 1965 [3], which requires $O(N \log N)$ operations if N is equal to a power of 2. More recent implementations such as the Fastest FFT in the West (FFTW [10]) are able to achieve the same computational complexity for signals of any length.

III. MATHEMATICAL OPERATIONS IN THE SFFT

A sparse signal is a signal where the majority of the Fourier coefficients in its DFT (or FFT) have values which are close or equal to zero. The spectrum of such a signal will therefore contain a small number of k peaks. The aim of the sFFT is to provide the same k peaks as the output of the frequency analysis, further reducing the required computational effort by exploiting the assumption of sparse signals.

The sFFT algorithm investigated in this paper is so-called version 1 [4], which has a computational complexity of $O(\log N \sqrt{Nk\ logN})$. The four core mathematical operations of the sFFT are discussed below, although the Reader is redirected to [4] for a more detailed analysis.

A. The pseudorandom signal permutation

Spectral peaks which occur at frequencies close to each other can be difficult to detect by the sFFT. A pseudorandom spectral permutation is a technique which can separate these close peaks.

According to [11], if N is a power of two the numbers which are relatively prime to N are the odd numbers in the range [0, … , N-1]. The modular multiplicative inverse of an integer σ modulo N is defined as an integer σ^{-1} which results in $\sigma \cdot \sigma^{-1} = 1\ (mod\ N)$. The modular multiplicative inverse of σ modulo N exists if and only if σ and N are relatively prime. As a result, we can conclude that σ is an odd number in the range [0, … , N-1].

The random odd number σ is used to dilate the signal to be permutated in the time domain. This action will result in a

translation in the frequency domain by σ^{-1}. This process is described by the equation shown below, where y is the time domain signal and Y is the frequency domain signal, with ω representing the general angular frequency [11]:

$$y(t) = x[\sigma t] \Leftrightarrow Y[\omega] = X[\sigma^{-1}\omega] \quad (1)$$

From an implementation point of view, (1) is performed by first storing *x(t)* in a buffer. When the buffer is full, the value of σ is generated by using a uniform pseudorandom number generator, as the one used in [12], which is a very simple and effective solution for real-time implementations. The number σ must be odd, which means that in case of a generated even number, σ is taken as the even number plus one. Then, the new vector $y(t)$ is calculated by sweeping the buffer $x(t)$ using σ as the base counter. At the same time, the modular multiplicative inverse σ^{-1} is calculated and stored for later use.

B. *The windowing function*

Two window functions are required by the sFFT algorithm: the first is used during the location of a signal's peaks while the second is used for the estimation of the amplitudes. The window used for frequency location ideally has a narrow bandwidth. For this purpose, a conventional Hanning window was used due to its good frequency resolution characteristics [13].

The window function used to estimate magnitude is shown in Figure 1. A key requirement for this windowing function is that it should have a flat passband with minimal ripple [4]. A simple method of constructing such a window was detailed in [5]. This method involved taking the convolution of a rectangular window and a Gaussian function in the frequency domain. Due to the duality of the convolution in the frequency domain and the multiplication in the time domain, this procedure could be implemented by multiplying a Gaussian with a *sinc* function in the time domain.

Both the Hanning window and the window for the magnitude estimation can be pre-calculated offline and their coefficients stored in the FPGA, as long as the number of acquired samples of the signal under analysis does not change in real-time. Of course, it could be possible to recalculate the window coefficients even in the case of variable buffer length, at the cost of a slightly higher computational effort.

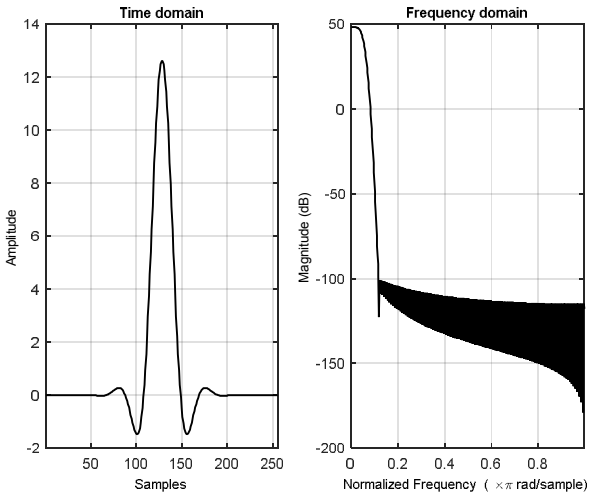

Figure 1: Window function used for estimation loops in the time domain (left) and frequency domain (right).

C. *The subsampled FFT*

The reduction of the number of operations in a sFFT is mainly achieved by running a conventional FFT over a subsampled version of the input signal, after its pseudorandom permutation and windowing. If the length of the subsampled data is B, and the original set of data is x, the expected number of operations of a FFT on a subsampled signal is $O(\text{supp}(x)+B\log B)$.

Naturally, the decrease of samples leads to a less accurate FFT. Figure 2 shows a comparison of a full-length and a subsampled FFT taken on a signal consisting of the summation of a sinewave with amplitude 1 at 500 Hz and amplitude 0.5 at 1 kHz, with some additional noise added. The number of samples in the signal is 4096, which is also the length of the full-length spectrum. It is apparent that the full-length spectrum has peaks of 0.5 at 500 Hz and 0.25 at 1 kHz, as expected. The subsampled spectrum has a length B of 256, and is spread compared to the full-length spectrum.

The effects of a less precise detection of the spectral lines are evident, although the subsampling process leads to faster calculations. However, if the subsampled FFT is performed a certain number of times over different pseudorandom permutated and windowed versions of the same input signal, and their results are analysed as later explained, the sFFT can provide as precise results as the FFT with an overall decrease of computational effort. This is the core of the sFFT algorithm as described in [4,5].

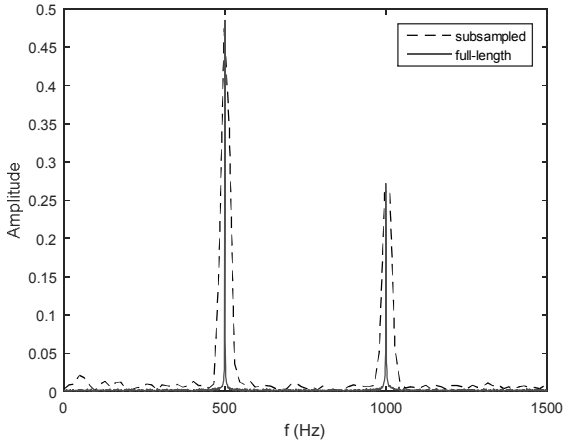

Figure 2: Comparison of magnitude spectrum obtained from a time domain signal using a FFT with length 4096 and a subsampled FFT with length 256.

D. *Inverse hash function*

The mathematical operations described by the permutation, windowing and subsampled FFT correspond to a hash function, which hashes the N frequencies of the original signal into B bins of a subsampled spectrum. Once the peaks within the B bins have been estimated, the coordinates of the original frequencies corresponding to these peaks should be determined.

Firstly, assume that a peak at a normalised frequency B_p has been found in the subsampled spectrum. Since the width of the

bin B_p is larger than the width of the bin N (meaning that the frequency resolution of the subsampled spectrum is worse than the original signal spectrum), we form upper and lower limits for mapping the width of B_p to the original frequency spectrum of length *N*. These upper and lower limits are called *Low* and *High*, which are calculated according to the following equations.

$$Low = [ceil\left(\left(B_p - 0.5\right)B\right) + N]mod(N) \quad (2)$$

$$High = [ceil\left(\left(B_p + 0.5\right)B\right) + N]mod(N) \quad (3)$$

After defining *Low* and *High*, the inverse hash function is achieved by using the modular multiplicative inverse σ^{-1} of the number σ used to perform the pseudorandom signal permutation, as described in the equation below. The variable *index* is ranging from *Low* to *High*.

$$loc = (\sigma^{-1}index)mod(N) \quad (4)$$

It should be noted that the values of *loc* are spread throughout the complete range of *N* due to the presence of the *mod* function, and are not bounded by *Low* and *High*. Moreover, the equations (2)-(4) do not present any complication from an implementation point of view, since standard operators are used. A graphical description of the inverse hashing process is given in Figure 3.

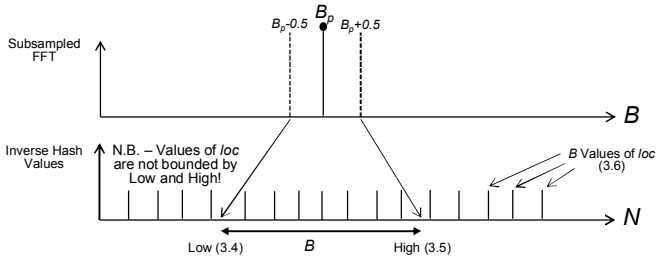

Figure 3: Graphical description of the inverse hash function process.

IV. The Complete sFFT Simulation Model

The four core operations described above were used in a simulation model to verify the complete algorithm before implementation. The model is divided into two main sections: the location loop, which searches for the frequencies at which peaks occur, and the estimation loop, which finds the magnitudes at these frequencies. The location and estimation loops are repeated for a number of iterations to obtain a statistically reliable estimation of the frequencies and magnitudes [4]. Figure 4 shows a sketch of the model.

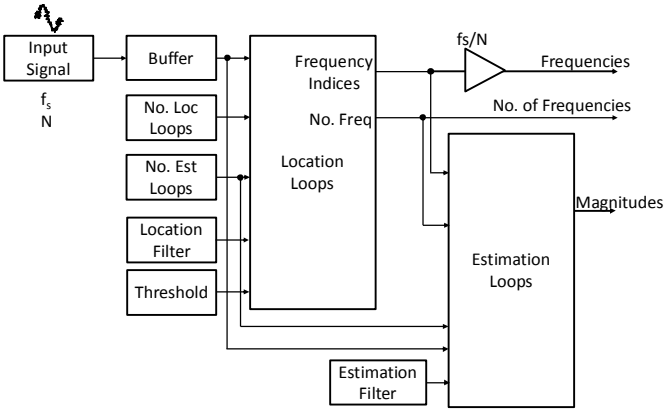

Figure 4: Simulation model of the sFFT.

A. Location loop

At the beginning, an input signal is sampled at a frequency f_s and used to fill a buffer of length *N*. The buffered data is then fed to the location loop which performs the pseudorandom signal permutation, the windowing using the *Location Filter* (Hanning window) and a subsampled FFT using a predefined value of *B*. The peaks above a limit *Threshold* in the subsampled spectrum are found, then subjected to an inverse hash function which calculates the corresponding indices in the full spectrum *N*. The location loop is iterated for a number of times given by *No. Loc Loops*, and the indices in *N* which correspond to a peak in more than *No. Loc Loops*/2 are counted as indices where a peak occurs in reality. The left-hand side of Figure 5 shows a block diagram of the location loop model.

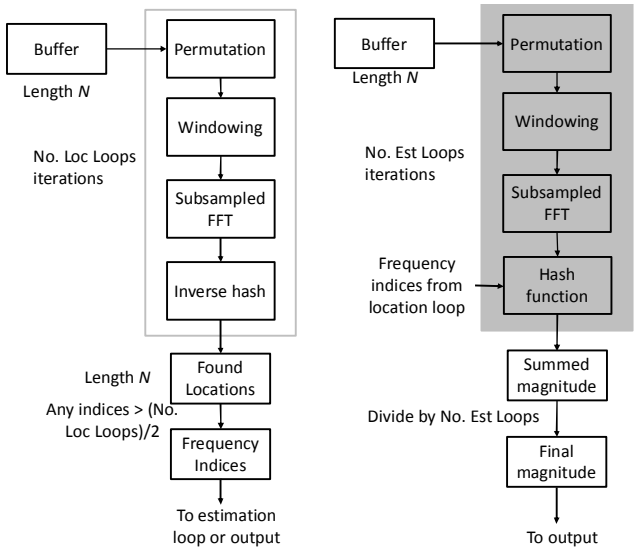

Figure 5: Block diagram of the location loop (left) and estimation loop (right).

B. Estimation Loop

The estimation loop makes use of the frequency indices where a peak is found in the location loop. The process of permutation, windowing and subsampled FFTs is repeated, with the difference that the flat estimation filter is used for the windowing. A hash function is used by the estimation loop to map the found indices in the original spectrum into the subsampled spectrum:

$$h_\sigma(i) = round\left(\frac{\sigma i B}{N}\right) \quad (5)$$

The magnitude of the subsampled spectrum at $h_\sigma(i)$ is taken as the magnitude of the original spectrum at the located index *i*. The estimation loop is iterated in a similar fashion to the location loop, in order to obtain a number of magnitude estimation for the same located indices. The final magnitude value is taken as the median of the different estimation magnitudes. The right-hand side of Figure 5 shows the block diagram of the estimation loop.

C. Simulation results

Figure 6 shows an example of a signal which is treated by the sFFT algorithm in the time domain (left) and frequency domain (right). The signal consists of the summation of a sinusoid with frequency of 20 Hz and amplitude of 0.8, a sinusoid with frequency of 1 kHz and magnitude of 1, and a sinusoid with a frequency of 1.8 kHz and a magnitude of 0.5. The signal is sampled at a frequency f_s = 4096 Hz, and routed to a buffer of N = 16384 cells. The time required to fill the buffer in this case is equal to N/f_s, and is equal to 4 s.

The simulation was run for a time of 1200 s, which was equivalent to 300 operations of the sFFT. The length of the subsampled spectrum B was set to 128, the number of location loops was set to 20, the number of estimation loops 10 and the threshold to 0.15. Note that a full-length FFT would require $O(16384\ log 16384) = O(229376)$ operations, while the sFFT asks for $O\left(\log 16384 \sqrt{16384 \cdot 3 \log 16384}\right) = O(11613)$ operations, a reduction by a factor of almost 20.

A 100% accuracy rate was observed in the 300 runs when determining the frequencies where spectral peaks were located. Some error was noted in the estimation of the magnitudes of each peak. Figure 7 shows the error percentage between the magnitude obtained by the sFFT simulation and the actual magnitude of the 20 Hz, 1 kHz and 1.8 kHz peaks. It is clear that signal-to-noise ratio plays a large role in the error percentage, since the 1 kHz peak (unity magnitude) has the lowest error, and the 1.8 kHz peak (magnitude of 0.5) has the largest error.

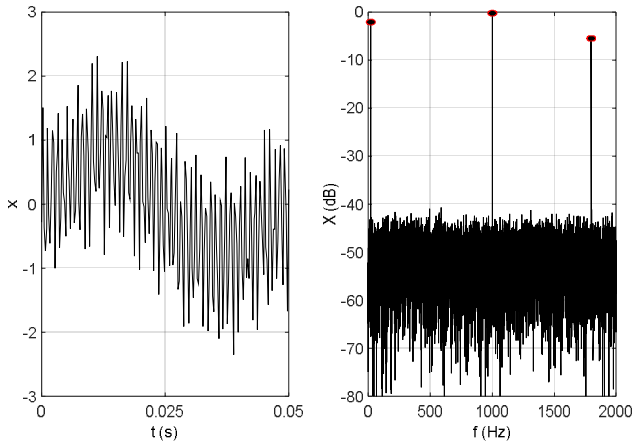

Figure 6: Time domain (left) and frequency domain (right) representations of the signal used to test the sFFT simulation model.

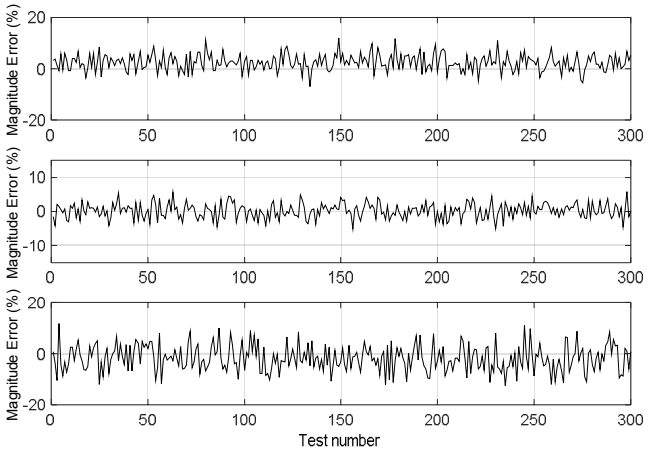

Figure 7: Simulated error percentage between magnitudes calculated by the sFFT and actual values for the 20 Hz peak (top), 1 kHz peak (center) and 1.8 kHz peak (bottom).

V. Experimental Results

A. Experimental setup

A picture of the experimental setup is shown in Figure 8. The load machine (an 11 kW interior permanent magnet machine) is visible on the left hand side of the image, and is used as a wind turbine load emulator [12]. The machine under test, used as a generator, is an 11 kW synchronous reluctance machine (SynRM) visible on the right hand side of the image.

The load machine is controlled by an ABB ACS850 converter. The test machine is connected to an ACS850, but is supplied with control signals from an OPAL-RT Technologies OP5600 system. The OP5600 is equipped with a quad-core Intel DSP processor at 2.4 GHz and a Virtex 6 FPGA, and the selected PWM switching frequency is 2 kHz. The phase currents and the DC-bus voltage are measured with a custom measurement box and connected to the A/D board of the OP5600. The digital I/Os of the OP5600 are used to communicate with the ACS850 power unit through a custom interface.

The load machine was used to inject different artificial torque disturbances. The magnitude of these harmonic components could either be constant, or randomly generated based on the wind turbine load model in [12].

Figure 8: Experimental setup for the sFFT algorithm verification.

B. FPGA implementation of the SFFT

The sFFT was conducted on a mechanical speed signal which had the DC component filtered away. The structure of the sFFT implementation is shown in Figure 9. The filtered speed signal is first buffered into two RAM blocks. The time required to fill up these buffers is given by N/f_s. The number of samples N in the buffer was set to 4096. Since the sampling of the speed signal was synchronized to the PWM frequency (2 kHz), the time required to fill the buffers was equal to 8.192 s. This means that the minimum time required for new sFFT results to be available was 8.192 s.

As in the simulated case, the sFFT algorithm was split into location and estimation loops. One of the key advantages of

FPGAs is the ability to have parallel paths which are executed simultaneously. This feature is utilized in the location loops, where two parallel execution paths are used to compute the frequencies at which peaks are detected. Such an implementation reduces the execution time of the sFFT algorithm compared to the case with a single execution path.

Once the frequencies at which peaks occur have been found by the location loops, the estimation loops are executed. Only one execution path was used for the estimation loop, as shown in Figure 9. The measured input signal was sampled at a frequency f_s then stored in two buffers of length L. A time of $(1/f_s)L$ seconds was required to fill these buffers. Once both buffers were filled, the location loops were executed in parallel. The peaks found by the location loops are used in the estimation loop, finding the corresponding magnitudes.

The sections of the FPGA used to implement the pseudorandom permutation, the windowing and the subsampled FFT operations were reused from the location loops in the estimation loops. For the purposes of device area comparison, it was found that the FPGA implementation of the sFFT took approximately 10% of the available slice logic on a Xilinx Virtex 6 FPGA. In contrast, a full-length (4096 point) FFT took approximately 15% of the available slice logic.

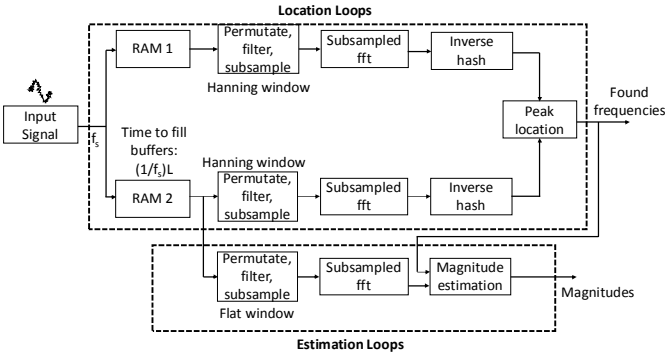

Figure 9: FPGA implementation of the sFFT.

C. Experimental Results

The performance of the FPGA implementation of the sFFT were compared with an off-line FFT analysis of a test waveform. Tests were conducted at rotational speeds of 10 and 20 rad/s. In both cases, the load machine torque was set to 1 Nm, with second and third harmonic torque components of 1 Nm added.

The filtered rotational speed signal without the DC component is shown in Figure 10 for the 10 rad/s case. With a rotational speed of 10 rad/s, the mechanical frequency of the fundamental is equal to 1.592 Hz, meaning that the second and third harmonics are located at 3.183 Hz and 4.775 Hz.

The magnitude spectrum obtained from the off-line full-length FFT is shown in Figure 11 as a solid line. Peaks were noted at frequencies of 3.174 Hz and 4.761 Hz, which correctly correspond to the 2nd and 3rd harmonic injected disturbances.

The on-line sFFT was conducted on the same signal, with N=4096. In this test, 10 location loops were used, along with 8 estimation loops and a threshold of 0.1. An example of one run of the subsampled FFT obtained in the location loops with a σ of 17 is shown in Figure 11 as a dashed line. The shifted and larger peaks are a result of the permutation and subsampling process described in Sect. III, and they correspond to the original FFT higher harmonics for $\sigma = 17$.

After multiple iterations of the location loop, the sFFT noted peaks at 3.174 Hz, 4.761 Hz, 1995 Hz and 1997 Hz. The

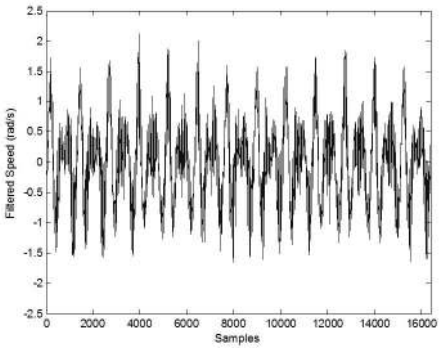

Figure 10: Rotational speed obtained with a load torque of 1 Nm, and second and third harmonics of 1 Nm injected at a speed of 10 rad/s.

magnitudes of these peaks were calculated as 0.3145, 0.25, 0.25 and 0.3145 respectively by the estimation loop. The first two peak frequencies are the very same 2nd and 3rd harmonic components found with the off-line FFT, while the second and the third ones are simply their aliased values in the complete spectrum - as it happens with a conventional FFT. Note also that the peaks of the subsampled spectrum in Figure 11 are quite wide, as a result of the subsampling process. Without the spectrum permutation, it would have been impossible to find the original signal harmonics since they are very close to each other.

The filtered rotational speed signal without the DC component for the 20 rad/s case with 2nd and 3rd harmonics injected is shown in Figure 12.

In this case, the fundamental rotational frequency is at 3.183 Hz, meaning that the 2nd harmonic is located at 6.366 Hz and the 3rd is located at 9.549 Hz. The magnitude spectrum obtained with an off-line FFT is shown in Figure 13 as a solid line. There are peaks measured at 6.348 Hz and 9.522 Hz, which correspond to the harmonic content injected into the load torque.

The on-line sFFT with N=4096, 10 location loops, 8 estimation loops and a threshold of 0.1 was then conducted. An

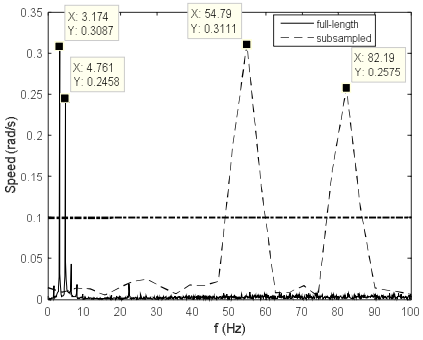

Figure 11: Magnitude spectrum obtained for the 10 rad/s case, using an offline FFT. The threshold used for the sFFT is shown as a horizontal line. The subsampled spectrum obtained during one run of the location loop with σ = 17 is also shown.

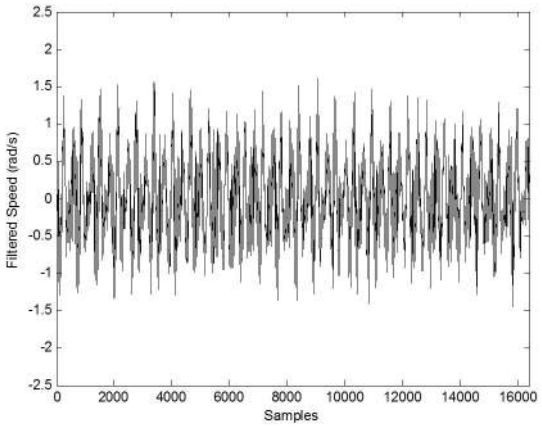

Figure 12: Rotational speed obtained with a load torque of 1 Nm, and second and third harmonics of 1 Nm injected at a speed of 20 rad/s.

example of one run of the subsampled FFT obtained in the location loops with a σ of 11 is shown in Figure 13 as a dashed line. Once again, the peaks in the subsampled spectrum were located at different frequencies than their actual values due to signal permutation with this value of σ. Again, the permutation process was fundamental to separate close harmonics.

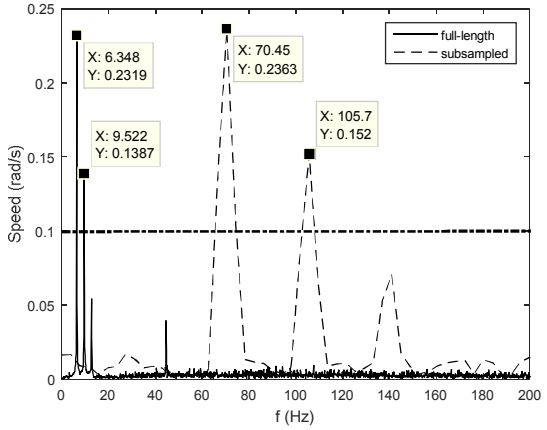

Figure 13: Magnitude spectrum obtained for the 20 rad/s case, using an offline FFT. The threshold used for sFFTs is shown as a horizontal line. The subsampled spectrum obtained during one run of the location loop with $\sigma = 11$ is also shown.

Peaks were found at 6.348 Hz, 9.521 Hz, 1990 Hz and 1994 Hz by the sFFT (after iterations of the location loop). The magnitudes of these peaks were 0.2432, 0.1504, 0.1504 and 0.2432 respectively. These values correspond to the 2nd and 3rd harmonic components and their aliased values, as expected. A summary of the results obtained from the off-line FFTs and on-line sFFTs is given below in Table 1.

Table 1: Comparison of off-line FFT and on-line sFFT results.

Harmonic number	**Off-line FFT**		**On-line sFFT**		**Error**
	freq (Hz)	mag	freq (Hz)	mag	mag (%)
2nd (10 rad/s)	3.174	0.3087	3.174	0.3145	1.88
3rd (10 rad/s)	4.761	0.2458	4.761	0.25	1.71
2nd (20 rad/s)	6.348	0.2319	6.348	0.2432	4.87
3rd (20 rad/s)	9.522	0.1387	9.521	0.1504	7.78

The tests showed that the sFFT can be used to implement a real-time frequency analysis algorithm in VSD control boards with results that match those of an off-line full-length FFT.

VI. Conclusions and Future Work

This paper applies the sparse fast Fourier transform (sFFT) as a real-time signal frequency analysis tool in variable speed drives. A theoretical background with an implementation perspective is given. Simulation results are presented, along with the description of the FPGA implementation. The sFFT algorithm is able to run in parallel with the field-oriented control of an 11 kW synchronous reluctance machine. Experimental results prove that the sFFT correctly estimates the frequency and magnitude of harmonic components which are injected into the load torque. The experimental evidence proves that at least some aspects of machine diagnostics and prognosis could be a natural part of the converter control of a drive, instead of confining them to external equipment solutions. This could be an interesting path for future works.

References

[1] L. Peretti, "Active torque harmonic compensation for wind turbine drive trains," in Proceedings of the 7 th IET International Conference on Power Electronics, Machines and Drives (PEMD 2014), Bristol, UK, Apr. 8–10, 2014, pp. 1–6.

[2] D.H. Lee, J.H. Lee, J.W. Ahn, "Mechanical vibration reduction control of two-mass permanent magnet synchronous motor using adaptive notch filter with fast Fourier transform analysis", IET Electr. Power Appl., 2012, vol. 6, no. 7, pp 451-461.

[3] J.W. Cooley, J.W. Tukey, "An Algorithm for the Machine Computation of the Complex Fourier Series", Mathematics of Computation, vol.19, Apr. 1965, pp. 297-301.

[4] H. Hassanieh, P. Indyk, D. Katabi, E. Price, "Simple and Practical Algorithm for Sparse Fourier Transform", Symposium on Discrete Algorithms 2012, Kyoto, 17-19 Jan. 2012, pp. 1183-1194.

[5] H. Hassanieh, P. Indyk, D. Katabi, "Nearly Optimal Sparse Fourier Transform", 44th Symposium on Theory of Computing 2012, New York, 19-22 May 2012, pp. 563-578.

[6] H. Hassanieh, L. Shi, O. Abari, E. Hamed, D. Katabi, "GHz-Wide Sensing and Decoding Using the Sparse Fourier Transform," Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), Toronto, Canada, Apr. 27 – May 2, 2014, pp. 2256-2264.

[7] L. Shi, O. C. Andronesi, H. Hassanieh, B. Ghazi, D. Katabi, and E. Adalsteinsson. "MRS sparse-FFT: reducing acquisition time and artifacts for in vivo 2D correlation spectroscopy", International Society for Magnetic Resonance in Medicine (ISMRM), Salt Lake City, USA, 20-23 Apr. 2013.

[8] L. Shi, H. Hassanieh, A. Davis, D. Katabi, F. Durand, "Light Field Reconstruction Using Sparsity in the Continous Fourier Domain", ACM Transactions on Graphics, vol. 34, no. 1, Nov. 2014.

[9] H. Hassanieh, F. Adib, D. Katabi, P. Indyk, "Faster GPS via the Sparse Fourier Transform", Proceedings of the 18th Annual International Conference on Mobile Computing (MobiCom), Istanbul, Turkey, 22-26 August, 2012, pp. 353-364.

[10] "FFTW", http ://www.fftw.org/, accessed 2016-03-29

[11] A.C. Gilbert, M.J. Strauss, J.A. Tropp, "A Tutorial on Fast Fourier Sampling – How to apply it to problems", IEEE Signal Processing Magazine, vol. 25, no. 2, Mar. 2008, pp. 57-66.

[12] L. Peretti, V. Särkimäki, J. Faber, "A wind turbine emulator for generator control algorithm development", 2013 IEEE International Conference on Industrial Technology (ICIT), Cape Town, 25-28 Feb. 2013, pp. 228-233.

[13] Understanding FFTs and windowing", http ://www.ni.com/white-paper/4844/en/, retrieved 2016-04-01