Reducing the Cost of Removing Border Artefacts in Fourier Transforms

Donald G. Bailey
School of Engineering and Advanced Technology,
Massey University,
Palmerston North,
New Zealand
D.G.Bailey@massey.ac.nz

Faisal Mahmood
Biomedical Engineering Department,
John Hopkins University,
Baltimore, MD
United States of America
faisalm@jhu.edu

Ulf Skoglund
Structural Cellular Biology Unit,
Okinawa Institute of Science and Technology,
Okinawa, Japan
ulf.skoglund@oist.jp

ABSTRACT

Many image processing algorithms are implemented in a combination of spatial and frequency domains. The fast Fourier transform (FFT) is the workhorse of such algorithms. One limitation of the FFT is artefacts that result from the implicit periodicity within the spatial domain. A new periodic plus smooth decomposition has recently been proposed for removing such artefacts, although this comes at the cost of an additional 2D FFT. In this paper, we restructure the decomposition to enable it to be calculated with a single 1D FFT, which can significantly accelerate artefact free Fourier transformation. The cost of this acceleration is a small amount of additional storage to hold the representation of the smooth image component.

1. INTRODUCTION

The Fourier transform is one of the fundamental transforms within image processing [3]. For $N \times N$ digital images, the discrete Fourier transform is used to transform from the spatial domain $f[x, y]$ to the spatial frequency domain $F[u, v]$:

$$F[u,v] = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f[x,y] W_N^{xu+yv} \tag{1}$$

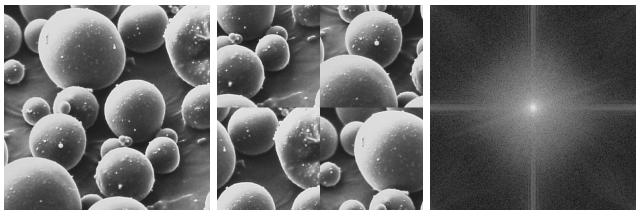
where the basis functions

$$W_N^{xu+yv} = e^{-j2\pi(xu+yv)/N} \tag{2}$$

are periodic in both space and frequency. This periodicity enables the transform to be factorised into a series of simpler stages, giving the fast Fourier transform (FFT) algorithm [4].

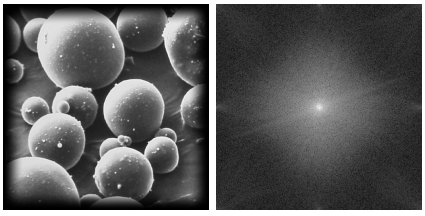
Unfortunately, this periodicity can also result in artefacts, because natural images are not truly periodic. When images are tiled, any mismatch in pixel values between the left and right borders, or between the top and bottom borders, results in a sharp discontinuity (as illustrated in Figure 1(b)). The artefacts are clearly seen in Figure 1(c) as the '+' shaped vertical and horizontal frequency components. Apart from the obvious distortion to the magnitude in the frequency domain, there are at least two other key problems of this. First, the phase is also distorted, which can give errors in frequency domain based image registration, or other methods which rely on the phase. Second, in frequency domain filtering or interpolation applications, the border discontinuity often results in ringing, particularly around the edges of the image.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.



(a) (b) (c)

Figure 1: Artefacts resulting from image border mismatches: (a) example image; (b) tiled image; (c) magnitude of the discrete Fourier transform (log scaled).



(a) (b)

Figure 2: Effect of windowing on artefacts: (a) windowed image; (b) and the resulting Fourier transform (log scaled).

1.1 Mitigating edge artefacts

The most common method to remove the edge artefacts is to remove the discontinuities by multiplying the image by a windowing or apodization function [7]. While windowing is clearly effective in reducing the border artefacts (see Figure 2), it also slightly blurs the content within the frequency domain. This is because multiplying by a spatial function is equivalent to a convolution by its Fourier transform in the frequency domain. Usually however, this effect is small, and can actually have the benefit of reducing

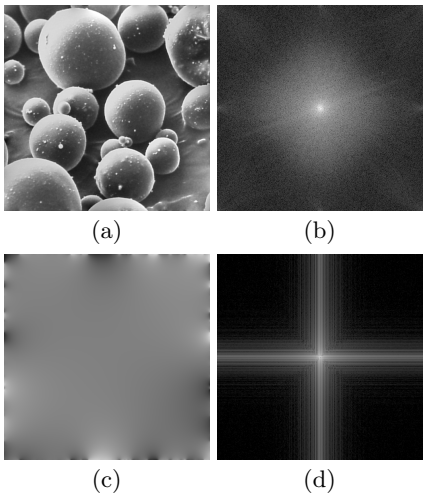


Figure 3: Periodic plus smooth decomposition: (a) periodic component; (b) its Fourier transform (log scaled); (c) smooth component; (d) its Fourier transform (log scaled).

spectral leakage because the Fourier transform of the window function has better side-lobe characteristics than the implicit rectangular window function. However, the main disadvantage of windowing is that when an inverse transform is performed, the resulting image will have the tapered effect around the edges. This loss of information at the edges of the image is usually unacceptable in filtering or interpolation applications.

Another technique commonly used to reduce the edge artefacts is to mirror the image at the borders [12]. This replaces the discontinuity of the image pixel values at the borders with a discontinuity in the first derivative, which has a significantly lower artefacts. Such symmetrisation doubles the size of the image, both horizontally and vertically, increasing the computation cost. It also effectively replaces the discrete Fourier transform with a discrete cosine transform. A side effect of this is to make the transform purely real (because the mirrored image has even symmetry, the imaginary components will be zero) and symmetric. The loss of phase information is critical in many applications (for example registration), and the symmetrisation in the frequency domain results in an orientation ambiguity in the spatial domain.

A new technique which has recently been proposed by Moisan [12] is a periodic plus smooth decomposition. This decomposes the image into a sum of two components: the periodic component (Figure 3(a)), which has no sharp border discontinuities, and will therefore not have the border artefacts; and the smooth component (Figure 3(c)), which embodies the border discontinuities, but is otherwise smooth throughout the image. The Fourier transform of the smooth component reflects the image border artefacts, which can be removed in the frequency domain simply by subtracting that component.

In the literature, there is only one reported FPGA implementation of a periodic plus smooth decomposition. Mahmood's system [10] transferred both the image (N^2 pixels) and the border image ($2N$ pixels) from the host PC. While the image was being transformed ($2N$ 1D FFTs), the border image was transformed in parallel ($N+1$ 1D FFTs) to derive the Fourier transform of the smooth image. Finally, the smooth image was subtracted from the original image in the frequency domain to obtain the periodic image (in the frequency domain) and remove the border artefacts.

1.2 Contributions

The primary contribution of this paper is the optimisation of the periodic plus smooth decomposition for hardware computation, and its integration within a 2D FFT. Specific novel features of the architecture are:

- The reduction of the time required for removing the border artefacts to that of performing a single 1D FFT.
- The optimisation of the storage required for the smooth component to a single row buffer (N complex values) plus a table of $\frac{N}{4}$ constant values.
- The real-time (60 frames per second) implementation on a low cost commodity FPGA (Cyclone V) of the FFT, including removing border artefacts, on a 512×512 image as it is streamed from a camera to a display.

The remainder of this paper is structured as follows. The periodic plus smooth decomposition is described in section 2, including the optimisation of the algorithm to reduce the computation required. Section 3 briefly reviews other FFT implementations on FPGAs. Our implementation of the FFT is described in section 4. This also describes the architecture for the periodic plus smooth decomposition for border artefact removal, and shows how it can be integrated within the original FFT with minimal overhead of both time and resources.

2. PERIODIC PLUS SMOOTH DECOMPOSITION

The periodic plus smooth decomposition [12] decomposes an image, f, into a periodic component, p, and smooth component, s:

$$f[x,y] = p[x,y] + s[x,y]. \tag{3}$$

The Fourier transform without border artefacts comes from the periodic component, which is achieved by subtracting the smooth component in the frequency domain:

$$P[u,v] = F[u,v] - S[u,v]. \tag{4}$$

By definition, the Laplacian of the smooth component is zero everywhere, except for at the borders of the image [12], i.e.

$$s[x,y] \circledast l[x,y] = b[x,y] \tag{5}$$

where $\circledast$ represents convolution, l is the Laplacian filter kernel, and $b = r + c$ is the border image with

$$r[x,y] = \begin{cases} f[N-1-x,y] - f[x,y], & x=0 \text{ or } x=N-1 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$c[x,y] = \begin{cases} f[x,N-1-y] - f[x,y], & y=0 \text{ or } y=N-1 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

representing the row-wise and column-wise borders respectively. From (5), the smooth component can be calculated directly in the frequency domain by inverse filtering the border image with the Laplacian filter kernel:

$$S[u,v] = \frac{B[u,v]}{L[u,v]} = \frac{B[u,v]}{2\cos\frac{2\pi u}{N} + 2\cos\frac{2\pi v}{N} - 4}. \tag{8}$$

Note that the denominator goes to 0 at the origin, but the border image also has zero mean. This can be satisfied by also requiring that the smooth image is also zero mean, i.e. $S[0,0] = 0$.

2.1 Optimising the computation

In applying a 2D FFT to the border image, separability can be exploited. Mahmood [10] made the observation that all of the columns (apart from the first and last) are mostly zero, enabling the column transform to be pre-calculated, and scaled by the border pixels:

$$\begin{aligned} B_{col}[x,v] &= (1 - W_N^{-v})c[x,0] \\ &= (1 - W_N^{-v})(f[x,N-1] - f[x,0]). \end{aligned} \quad (9)$$

Only the first column requires the Fourier transform to be calculated (of $r[0,y]$). Finally, N row transforms are required to determine B because B_{col} is no longer sparse. Therefore a total of $N+1$ 1D FFTs was used to transform the border image.

However, this can be significantly optimised further by exploiting the linearity of the Fourier transform. Each of the border components, r and c can be transformed separately, exploiting both sparsity and separability. Each border component is essentially one dimensional. Let

$$\hat{r}[y] = r[0,y] = f[N-1,y] - f[0,y] \quad (10)$$

$$\hat{c}[x] = c[x,0] = f[x,N-1] - f[x,0] \quad (11)$$

be the 1D border components. Each of these can be transformed by a 1D FFT, and then scaled in the other dimension to give the corresponding 2D FFTs:

$$R[u,v] = \hat{R}[v](1 - W_N^{-u}) \quad (12)$$

$$C[u,v] = \hat{C}[u](1 - W_N^{-v}). \quad (13)$$

The smooth component is then given by

$$S[u,v] = \frac{R[u,v] + C[u,v]}{2\cos\frac{2\pi u}{N} + 2\cos\frac{2\pi v}{N} - 4}. \quad (14)$$

Since both $\hat{r}$ and $\hat{c}$ are real, their Fourier transforms will be conjugate symmetric. Therefore, for a square image, they can both be transformed by a single FFT by combining them as a single signal [1]

$$\hat{b} = \hat{r} + j\hat{c} \quad (15)$$

with the individual frequency domain components extracted as

$$\hat{R}[v] = \tfrac{1}{2}(\hat{B}[v] + \hat{B}^*[N-v]) \quad (16)$$

$$\hat{C}[u] = \tfrac{1}{2j}(\hat{B}[u] - \hat{B}^*[N-u]) \quad (17)$$

where $\hat{B}^*$ is the complex conjugate of the Fourier transform of $\hat{b}$.

3. FPGA IMPLEMENTATIONS OF FFT

When N is a power of 2, the radix-2 algorithm decomposes a 1D FFT into $\log_2 N$ stages, with each stage performing $\frac{N}{2}$ 2-point Fourier transforms, with the output of each stage scaled by a complex twiddle factor. Each such 2-point transform is called a 'butterfly' operation. Bergland [2] identified four main architectures for implementing an FFT in hardware.

The first is a sequential processor, building a single butterfly processor, which sequentially performs all of the butterfly operations. Such a processor can operate on the data in place in memory. Examples of sequential implementations include: Uzun [19] used Handel-C to implement radix-2, radix-4 and split radix implementations; Derafshi [6] used a radix-2 butterfly; and Qian [16] used a split radix design, which minimises the number of multiplications required, and enables lower power.

The next architecture is a cascade processor, which builds $\log_2 N$ butterflies, one for each stage of the transform. A cascade processor is capable of directly processing streamed data, with the data streamed between successive stages. He [8] explored such a design for a range of radices, and proposed a radix-2^2 butterfly, which decomposes a radix-4 stage into two radix-2 stages, where a complex twiddle factor is only required after every second stage. Other cascade designs were implemented by Sukhsawas [18], Saeed [17] and Zhou [20].

A parallel iterative architecture builds $\frac{N}{2}$ butterflies to perform a single stage in parallel. The processor is then iterated over each stage of the transform. The main problem with this approach is overcoming the bandwidth issues of loading sufficient data in parallel to keep the butterfly processors busy [9]. An alternative is to partially parallelise each stage (subject to bandwidth availability), using several iterations to process each stage. Ouerhani [14] took such an approach to implement an 8-point FFT in parallel. However, even in this case the processing speed was limited by the serial input bandwidth, and building up the parallel data to process. Palmer [15] used a hybrid approach, with a four input parallel stage for the input, with the results feeding into 4 parallel cascade processors for the subsequent stages of processing.

The final architecture is an array processor, which builds all of the required $\frac{N}{2}\log_2 N$ butterfly processors and operates them in parallel. However, both the hardware cost and the bandwidth limitations make this impractical for an FPGA implementation.

When performing the complex multiplications by the twiddle factors, it is rounding errors which dominate the accuracy of a practical implementation of the transform. For 8-bit input images, a 16-bit fixed point representation is the minimum that gives acceptable results. Oraintara [13] proposed an integer FFT, which uses lifting when performing the multiplication by the twiddle factors to enable perfect reconstruction, even in the presence of rounding errors. Such a design was implemented on an FPGA using a split radix butterfly by Mohammadnia [11]. However, any processing in the frequency domain will modify the values, negating perfect reconstruction. A minor disadvantage of using lifting is that the three real multiplications to perform a complex multiplication are in series rather than in parallel, giving an increased latency, although this can be mitigated through appropriate pipelining.

Few of the implementations in the literature consider border artefacts. Bailey [1] outlined multiplying by an apodization function as the data are streamed into the FFT, and Mahmood [10] implemented a serial plus parallel decomposition for artefact removal, as described earlier.

4. IMPLEMENTATION

The goal of this study was to implement a 512×512 FFT

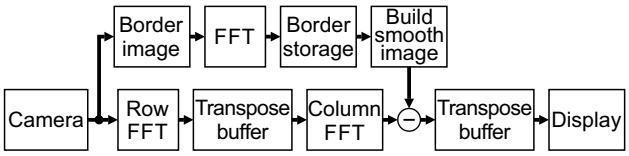


Figure 4: Logical structure of the algorithm.

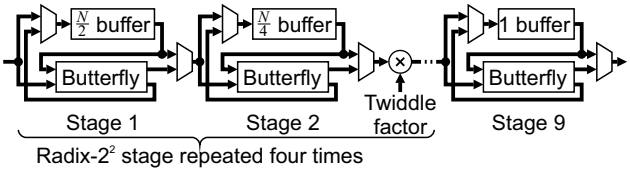


Figure 5: Structure of the radix-2^2 FFT.

with border artefact removal in real time (60 frames per second) as an image was streamed from a digital camera onto a display. This was to demonstrate that the cost of artefact removal did not have significant impact on either resources or time, enabling implementation on a low cost commodity FPGA. For this study, a *Terasic DE10-Standard* development board was used, which is based around a Cyclone V 5CSXFC6D6F31C6 FPGA. The image was streamed from a *TRDB_D5M* camera module, with the 512×512 image displayed on the left, and the FFT of the previous frame displayed on the right of a 1024×768 display. Since the camera and display have different timing requirements, they are run row synchronously with different clocks. A single row buffer is used to transfer the image between clock domains. The design was implemented in VHDL, and compiled for the FPGA using Quartus Prime 16.1.

4.1 System architecture

The logical structure of the algorithm is shown in Figure 4. The 2D FFT is implemented separably, with the row transform applied directly to the pixels as they are streamed in from the camera, with the intermediate results stored in a transpose buffer. In parallel, the row and column border images ($\hat{r}$ and $\hat{c}$) are extracted from the incoming pixel stream. Once the image is loaded, the border data is transformed with $\hat{B}$ saved into the border storage. The intermediate image results are read column-wise from the transpose buffer and a column FFT is applied. The *build smooth image* block extracts the border components and scales them to obtain the smooth image component, which is subtracted from the outputs of the column FFT as they are written back to the transpose buffer. Finally, the frequency domain image is read from the transpose buffer row-wise and is streamed to the display.

Since the 1D FFT must process streamed data, a cascade architecture was used with a radix-2^2 butterfly [8]. A 512-point transform has 9 radix-2 stages, with complex multiplications after every second stage, as shown in Figure 5. The intermediate twiddle factors are $-j$, which are trivial to implement in hardware. The size of the buffer is halved with each stage, with the longer buffers implemented using dual-port memory blocks, and the shorter buffers implemented using registers.

Although 3 FFTs are shown in Figure 4, in practise, only a singe FFT block is constructed and multiplexed between the various instances, as illustrated in Figure 6. For real images,

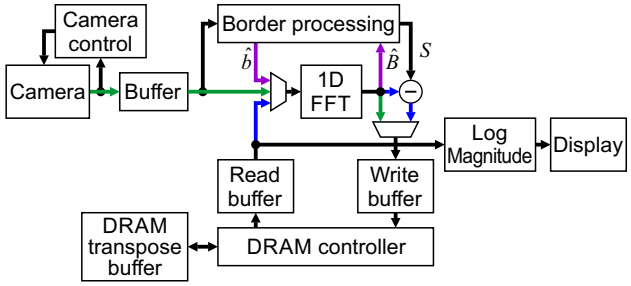


Figure 6: System architecture. The different colours show the different paths for the FFT: green = row FFT, blue = column FFT, purple = border FFT.

the result of the row transform is conjugate symmetric, so it is only necessary to save the results for $0 \leq u \leq \frac{N}{2}$ into the transpose buffer. The values for $u = 0$ and $u = \frac{N}{2}$ are both real, and come from the FFT in successive clock cycles, so are saved together. As the image is too large to store on-chip, external DRAM was used for the transpose buffer. The write buffer for the DRAM controller is sufficient to hold one row or column of the image. Bit-reversed addressing to be used to reorder the samples into their natural order before writing the rows to DRAM. After completing the row FFTs, all of the border data is available, so the border signal, $\hat{b}$, is then transformed while pre-fetching the column data from the DRAM into the read buffer. The 256 columns are then processed, with the smooth image component subtracted before returning them to the DRAM based transpose buffer in bit-reversed order. Finally, while beginning the process again with the next frame, the image is read from the DRAM for display. The displayed Fourier transform is log scaled to enable the details within the frequency spectrum to be seen more clearly.

One issue with using DRAM as the transpose buffer is that accessing the memory by column can severely impact the speed because of the overheads of repeatedly activating and precharging memory rows [5]. To hide these overheads, it is necessary to have a burst of sequential accesses within each memory row, and have successive rows accessed be located in different banks. This was accomplished by reordering the physical memory address lines, as shown in Figure 7 so that there were at least parts of four image rows within each memory row.

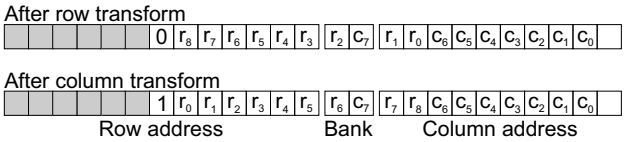


Figure 7: Mapping of the DRAM memory address bits. Each word is 16 bits, so a complex number requires 2 words. Note that the column transform is stored in bit-reversed order.

4.2 Border processing

The novel aspect of this work is the detailed architecture of the border processing using, which calculates the smooth image component, S. The detailed architecture for this is shown in Figure 8.

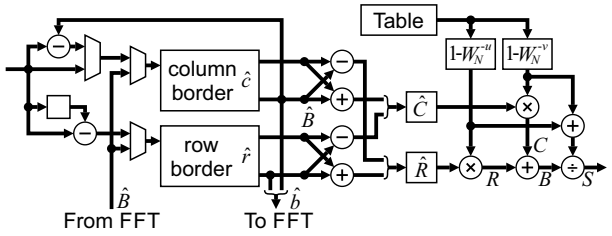


Figure 8: Detailed architecture of the border processing for calculating the smooth image component.

First, the border pixels are extracted from the incoming pixel stream and are saved in memory buffers. The row border, $\hat{r}$, registers the first pixel of each row, and subtracts it from the last pixel in each row according to (10) before saving it to the row border buffer. Similarly, the first row is stored in the column border buffer, and is subtracted from the last row when it arrives, as in (11). This requires that the column border buffer be dual-port.

For the FFT, $\hat{r}$ and $\hat{c}$ are combined as in (15) so that they can be processed using a single FFT. The result is saved back into the row and column border buffers, which collectively represent the border image $\hat{B}$ in the frequency domain.

As the frequencies are output from the column FFT, it is necessary to provide the corresponding pixel from the smooth image, S. The values of $\hat{R}$ and $\hat{C}$ are derived from $\hat{B}$ by adding and subtracting the real and imaginary parts of the positive and negative frequencies according to (16) and (17). With dual-port buffers, both positive and negative frequency component can be accessed in the same clock cycle. For each sample, $(1 - W_N^{-v})$ can be read for scaling the output. These are cross-multiplied to give R and C and then added to give the uncompressed border image, B. Note that $\hat{C}[u]$ and $(1 - W_N^{-u})$ are constant for each column, and can be read and registered once at the start of the column.

The table for $(1 - W_N^{-v})$ actually only contains $\cos \frac{2\pi v}{N}$. While the table could be compressed to only hold values for $0 \leq v < \frac{N}{4}$ (because the other values can be obtained from symmetry relations), the full table fits into a single memory block, so nothing is gained by this compression for a 512×512 image, and using the full table reduces the decoding logic. The value from the table is subtracted from 1 to give the real part of the scaling. The imaginary part, $-j \sin \frac{2\pi v}{N}$, can be obtained from the same table by offsetting the address by $\frac{N}{4}$, and using dual-port memory to allow two parallel accesses.

Finally, the inverse filter given in the denominator of (14), is twice the sum of the real parts of $(1-W_N^{-u})$ and $(1-W_N^{-v})$. These can be added, and then divided into B to give S.

4.3 Results and discussion

The design was implemented, and runs with a 94 MHz pixel clock for the camera and a 65 MHz pixel clock for the display. To meet bandwidth requirements, the DRAM was run with a 100 MHz clock, and the read and write buffers were used for transferring data between the clock domains. The processing speed in this design was constrained primarily by the display rate, with a secondary factor being the DRAM bandwidth (which was 80% used). However this was sufficient to successfully perform a 512×512 FFT at 60 frames per second. Note that the 1D FFT unit was only running at about 37% capacity, allowing it to easily be used to also implement the inverse FFT required for frequency domain filtering or other related applications at this frame rate with minimal additional computational resources.

Table 1: Resources for capturing and displaying a 512×512 image and its Fourier transform with border artefacts removed when implemented on a Cyclone V 5CSXFC6D6F31C6 FPGA.

Module	ALUTs	Registers	M10Ks	DSPs
Camera control	228	221	1	0
Bayer to greyscale	103	96	1	0
VGA timing	47	30	0	0
1D FFT	1603	884	7	8
DRAM controller	691	346	4	0
Border processing	1077	429	3	4
Log magnitude	1033	175	0	0
Misc top level logic	172	68	1	0
Total	4954	2249	17	12
out of	83200	83200	553	112

The resources required by the different modules are listed in Table 1. The camera control module included initialisation of the camera (via I^2C), automatic exposure and gain control, and the buffer to transfer the image between clock domains. The output of the camera was passed through a Bayer filter which converted the Bayer pattern to a greyscale image. The memory used by this filter was to buffer one image row at 12 bits per pixel.

For the FFT, complex multiplication by each of the twiddle factors required 2 DSPs. Only the first 7 stages implemented the buffers using memory blocks - the last 2 stages were short enough to implement directly using registers.

The DRAM controller required 2 memory blocks for each of the read and write buffers. This was sufficient for each buffer to hold the data from one complete 1D FFT.

Border processing used significantly fewer resources than the FFT. Each of the border buffers required a single memory block, as did the lookup table for the pre-calculated cross terms. The 4 DSP blocks were used by the complex multiplications to constitute the border image, B. Although the logic requirements were relatively high, most of this was required to implement the pipelined division on the output (764 ALUTs and 237 registers).

Finally, the magnitude was implemented using a CORDIC unit, followed by a similar shift and add algorithm to calculate the logarithm. The iterations for both operations were unrolled (giving the relatively large logic requirements) and pipelined over 5 clock cycles, however this was necessary to give the required throughput of 1 pixel per clock cycle.

4.4 Comparison with other systems

It is difficult to compare this implementation with the only other prior reported system to implement the FFT with artefact removal [10] as that paper did not report resource usage. Their implementation used a Xilinx Kintex 7, a similar low cost FPGA, with a reported processing time for a 512×512 FFT of 32.4 ms. The system described here operates at 60 frames per second (16.7 ms), almost twice as fast as the prior system. It should be noted that this timing was governed primarily by the display frame rate, and could be reduced further if this constraint was removed. The other li-

miting factor with this design was the DRAM bandwidth. A wider (>16 bits) and higher frequency (>100 MHz) DRAM would enable the frame rate to be significantly increased.

5. CONCLUSIONS

This design has demonstrated that it is possible to significantly accelerate the border artefact removal associated with an FFT. In terms of timing, the FFT unit was shared between the main FFT processing and border processing. Through careful analysis of the algorithm, border processing was reduced to a single 1D FFT. For a 2D FFT, this overhead is negligible, and can be performed during idle time while data is being transferred to and from the transpose buffer.

This is a considerable improvement over prior implementations of periodic plus smooth decompositions on an FPGA.

The rest of the construction of the border processing image can be performed in parallel with the column FFTs, without significantly impacting latency. It has been demonstrated that a single complex buffer, $\hat{B}$ is sufficient to store the compressed border image in the frequency domain, with an additional lookup table of constant values required for constructing the smooth image component. Again, this provides a significant improvement over prior work where the whole 2D smooth image was required to be stored in external memory.

The system was implemented on a low cost, commodity FPGA, enabling real time operation at 60 frames per second when processing a 512×512 image ($\sim$16 ms processing time). In this, the periodic plus smooth image decomposition was successfully integrated with the normal FFT processing to remove border artefacts from the Fourier transform. On the *DE10 Standard* development board, the primary factor limiting the processing speed was the bandwidth of the external DRAM used for the transpose buffer. With increased bandwidth, higher frame rates or image sizes could easily be achieved, especially by further exploiting parallelism with multiple FFT engines.

Acknowledgements

The authors wish to thank Terasic for providing the *DE10 Standard* development board for evaluation and review.

6. REFERENCES

[1] D. G. Bailey. *Design for embedded image processing on FPGAs*. John Wiley and Sons (Asia) Pte. Ltd., Singapore, 2011.

[2] G. D. Bergland. Fast Fourier transform hardware implementations - an overview. *IEEE Transactions on Audio and Electroacoustics*, 17(2):104–108, 1969.

[3] E. O. Brigham. *The fast Fourier transform and its applications*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[4] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[5] B. Davis. *Modern DRAM Architectures*. Phd thesis, University of Michigan, 2001.

[6] Z. H. Derafshi, J. Frounchi, and H. Taghipour. A high speed FPGA implementation of a 1024-point complex FFT processor. In *2010 Second International Conference on Computer and Network Technology*, pages 312–315, 2010.

[7] F. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[8] S. He and M. Torkelson. A new approach to pipeline FFT processor. In *10th International Parallel Processing Symposium*, pages 766–770, 1996.

[9] H. Kee, S. S. Bhattacharyya, N. Petersen, and J. Kornerup. Resource-efficient acceleration of 2-dimensional fast Fourier transform computations on FPGAs. In *2009 Third ACM/IEEE International Conference on Distributed Smart Cameras*, 8 pages, 2009.

[10] F. Mahmood, M. Toots, L. G. Ofverstedt, and U. Skoglund. 2D discrete Fourier transform with simultaneous edge artifact removal for real-time applications. In *International Conference on Field Programmable Technology*, pages 236–239, 2015.

[11] M. R. Mohammadnia and L. Shannon. Minimizing the error: a study of the implementation of an integer split-radix FFT on an FPGA for medical imaging. In *International Conference on Field Programmable Technology*, pages 360–367, 2012.

[12] L. Moisan. Periodic plus smooth image decomposition. *Journal of Mathematical Imaging and Vision*, 39(2):161–179, 2011.

[13] S. Oraintara, Y. Chen, and T. Nguyen. Integer fast Fourier transform. *IEEE Transactions on Signal Processing*, 50(3):607–618, 2002.

[14] Y. Ouerhani, M. Jridi, and A. Alfalou. Implementation techniques of high-order FFT into low-cost FPGA. In *2011 IEEE 54th International Midwest Symposium on Circuits and Systems*, 4 pages, 2011.

[15] J. Palmer and B. Nelson. A parallel FFT architecture for FPGAs. In *14th International Conference on Field Programmable Logic and Application*, LNCS volume 3203, pages 948–953, 2004.

[16] Z. Qian, N. Nasiri, O. Segal, and M. Margala. FPGA implementation of low-power split-radix FFT processors. In *24th International Conference on Field Programmable Logic and Applications*, 2 pages, 2014.

[17] A. Saeed, M. Elbably, G. Abdelfadeel, and M. I. Eladawy. Efficient FPGA implementation of FFT / IFFT processor. *International Journal of Circuits, Systems and Signal Processing*, 3(3):103–110, 2009.

[18] S. Sukhsawas and K. Benkrid. A high-level implementation of a high performance pipeline FFT on Virtex-E FPGAs. In *IEEE Computer Society Annual Symposium on VLSI*, pages 229–232, 2004.

[19] I. Uzun, A. Amira, and A. Bouridane. FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEE Proceedings - Vision, Image and Signal Processing*, 152(3):283–296, 2005.

[20] B. Zhou, Y. Peng, and D. Hwang. Pipeline FFT architectures optimized for FPGAs. *International Journal on Reconfigurable Computing*, 9 pages, 2009.