FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing

I.S.Uzun* A.AmiraA.Bouridane

School of Computer Science
The Queen's University of Belfast
BT7 1NN Belfast, United Kingdom
**i.s.uzun@qub.ac.uk*

Abstract

Applications based on Fast Fourier Transform (FFT) such as signal and image processing require high computational power, plus the ability to experiment with algorithms. Reconfigurable hardware devices in the form of Field Programmable Gate Arrays (FPGAs) have been proposed as a way of obtaining high performance at an economical price. At present, however, users must program FPGAs at a very low level and have a detailed knowledge of the architecture of the device being used.

To try to reconcile the dual requirements of high performance and ease of development, this paper reports on the design and realisation of a High Level framework for the implementation of 1-D and 2-D FFTs for real-time applications. Results show that the parallel implementation of 2-D FFT achieves virtually linear speed-up and real-time performance for large matrix sizes. Finally, an FPGA-based parametrisable environment based on the developed parallel 2-D FFT architecture is presented as a solution for frequency-domain image filtering application.

1. Introduction

Fourier transforms play an important role in many digital signal processing applications including acoustics, optics, telecommunications, speech, signal and image processing [1,17,18]. However, direct computation of Discrete Fourier Transform (DFT) requires on the order of N^2 operations where N is the transform size. The FFT algorithm, first explained by Cooley and Tukey [1], opened a new area in digital signal processing by reducing the order of complexity of DFT from N^2 to $N \log_2 N$.

Since the early paper by Cooley and Tukey, a large number of FFT algorithms have been developed. Among these, the radix-2, radix-4, split-radix and FHT algorithms are the ones that have been mostly used for practical applications due to their simple structure, with a constant butterfly geometry, and the possibility of performing them "in place".

Most of the research to date for the implementation and benchmarking of FFT algorithms have been performed using general purpose processors [3,4], Digital Signal Processors(DSPs) [5,6,7] and dedicated FFT processor ICs [8,9]. However, as Field Programmable Gate Arrays (FPGAs) have grown in capacity, improved in performance, and decreased in cost, they have become a viable solution for performing computationally intensive tasks (i.e computation of FFT), with the ability to tackle applications for custom chips and programmable DSP devices [10-14].

Although there has been extensive research on the hardware implementation of the FFT algorithms, there are some inherent drawbacks of existing studies. They are designed and optimized for:

- fixed type of ASIC, DSP and FPGA platform;
- fixed type of FFT algorithm; and
- certain fixed design parameters such as transform size (N), input/output data wordlengths (L).

This narrows the application area of such implementations.

This paper is concerned with :

- the design and implementation of a parametrisable architecture, which provides a framework for the implementation of different types of 1-D FFT algorithms;
- the development of an FPGA-based FFT library by implementing radix-2, radix-4, split-radix and FHT algorithms under the same framework in order to provide system designers and engineers with the flexibility to meet different application requirements with given hardware resources;
- the design and implementation of a generic parallel 2-D FFT architecture for real-time image processing applications; and
- the development of an FPGA-based parametrisable system for frequency-domain filtering of large images.

The FFT architectures have been designed using Handel-C language [15]. Although the task could have been accomplished using traditional Hardware Description Languages (such as VHDL or Verilog), Handel-C has been selected as we aimed to evaluate its rapid design capabilities and suitability for the design of IP cores.

The target hardware for the implementation and verification of architectures is Celoxica RC1000-PP PCI based FPGA development board equipped with a Xilinx XCV2000E Virtex FPGA [10,16].

The composition of the rest of the paper is as follows. The proposed system for the implementation of 1-D and 2-D FFTs is given in section 2. A mathematical background of the FFT algorithms is given in Section 3. The proposed architectures for 1-D and parallel 2-D FFT are described in Section 4. The results and the analysis for the implementations of 1-D and 2-D FFT designs are presented in Section 5. The environment developed for frequency-domain image filtering application is described in Section 6. Concluding remarks are given in Section 7.

2. Proposed System

The environment for the proposed system as shown in Figure 1 consists of a GUI for use in mapping the 1-D and 2-D FFTs on the FPGAs. The user receives the result as parametrisable Handel-C code and/or EDIF netlist file from the library with the suitable generic specified parameters. The Handel-C codes stored in the FFT library are parameterisable and can be accessed using the GUI interface to implement the design on the FPGAs. The generic parameters for Handel-C code are respectively:

- the FFT dimension (*1-D or 2-D*);
- the FFT algorithm (*radix-2,radix-4,split-radix, FHT*);
- the transform length (*N*);
- the input and output data word-length (*Li, Lo*);
- the coefficients word-length (*W*),
- the number of processor elements (*p*) in parallel 2-D FFT architecture; and
- FPGA device type.

3. Fast Fourier Transforms: a review

3.1 1-D DFT and Its Fast Computation

The DFT of an *N*-point discrete-time complex sequence $x(n)$, indexed by $n = 0,1,\ldots,N-1$, is defined by

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_M^{kn}, \quad k = 0,1,..,N-1 \qquad (1)$$

where $W_N = e^{-j2\pi/N}$ and W_N is referred as the *twiddle factor*.

The excessively large amount of computations required to compute the DFT directly when *N* is large has prompted alternative methods for computing the DFT efficiently. This problem was alleviated with the development of special fast algorithms, collectively known as fast Fourier transform.

Most of FFT algorithms decomposes the overall *N*-point DFT into successively smaller and smaller

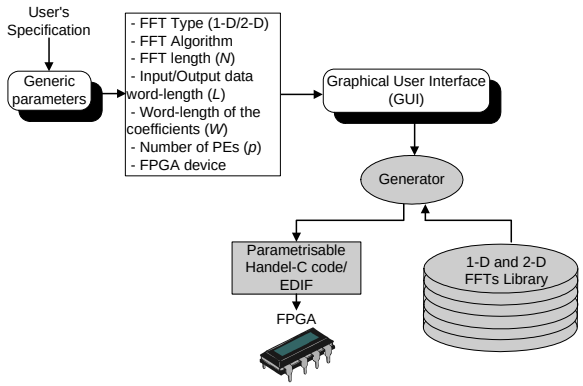

Figure 1. Proposed system for the FFTs implementation.

transforms known as a butterfly. An overview of the most common FFT algorithms is presented in the following subsections. Again, references [1,2,17,18,19] contain a detailed development of these FFT algorithms.

3.1.1 Radix-2^n FFT Algorithms

For Cooley-Tukey radix-2 algorithm, decimation-in-frequency (DIF) decomposition, Eq. (1) is decomposed into even and odd frequency components [1]:

$$X_{2k} = \sum_{n=0}^{N/2-1} (x_n + x_{n+N/2}) \cdot W_{N/2}^{nk} \qquad (2)$$

$$X_{2k+1} = \sum_{n=0}^{N/2-1} (x_n - x_{n+N/2}) \cdot W_{N/2}^{n} \cdot W_{N/2}^{nk} \qquad (3)$$

For radix-2 FFT algorithm, the smallest transform or butterfly (basic computational unit) used is the 2-point DFT as shown in Figure 2.a.

The radix-4 algorithm can be obtained by decomposing Eq. (2) and (3) into X_{4k}, X_{4k+2}, X_{4k+1} and X_{4k+3} frequency components. The radix-4 butterfly has 4 inputs and 4 outputs, essentially combining two stages of a radix-2 algorithm in one. Figure 2.b shows the radix-4 butterfly.

The split-radix algorithm [19] decomposes the odd frequency component in Eq (3) into $4k+1$ and $4k+3$ frequency components as follows;

$$X_{4k+1} = \sum_{n=0}^{N/4-1} (x_n - j \cdot x_{n+N/4} - x_{n+N/2} + j \cdot x_{n+3N/4}) \cdot W_N^{3n} \cdot W_{N/4}^{nk} \qquad (4)$$

$$X_{4k+3} = \sum_{n=0}^{N/4-1} (x_n + j \cdot x_{n+N/4} - x_{n+N/2} - j \cdot x_{n+3N/4}) \cdot W_N^{3n} \cdot W_{N/4}^{nk} \qquad (5)$$

The L-shape butterfly element of split-radix FFT algorithm is given in Figure 2.c.

3.1.2 The Fast Hartley Transform

The Discrete Hartley Transform (DHT) belongs to the family of frequency transforms. The significant difference between DHT and DFT is that DHT is a real-valued transform.

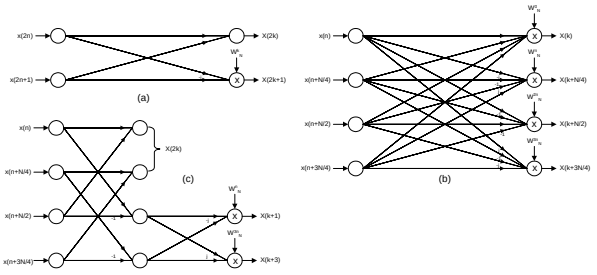

Figure 2. The "butterfly" used in (a) radix-2, (b) radix-4, (c) split-radix algorithms.

The DHT is defined for a real-valued N-point sequence $x(n)$, n=0,1,…,N-1, by the following equation [2]:

$$H_k = \sum_{n=0}^{N-1} x_n \cdot cas\left(\frac{2\pi}{N} kn\right), \quad k = 0,1,...,N-1 \tag{6}$$

where $cas(x) = \cos(x) + \sin(x)$.

The FHT algorithm for the computation of DHT resembles the radix-2 FFT algorithm. The FHT algorithm is based on the decomposition of Eq (6) as follow:

$$\begin{aligned} H_{2k} &= \sum_{n=0}^{N/2-1} \left[x_n + x_{n+N/2}\right] \cdot cas\left(4\pi nk / N\right) \\ H_{2k+1} &= \sum_{n=0}^{N/2-1} \left[x_n - x_{n+N/2}\right] \cdot \left\{ \begin{matrix} cas\left(4\pi nk / N\right) cos\left(2\pi n / N\right) + \\ cas\left(4\pi (N-n)k / N\right) sin\left(2\pi n / N\right) \end{matrix} \right\} \end{aligned} \tag{7}$$

Each of the (N/2)-point DHT's can be further decomposed in a similar fashion to complete the FHT DIF algorithm.

3.2 The 2-D FFT

The 2-D DFT of a NxN matrix is defined in a manner similar to the 1-D case [17,18]. The 2-D DFT is given by:

$$X(k_1,k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1,n_2) e^{-j2\pi (k_1 n_1 + k_2 n_2)/N} \tag{8}$$

where $0 \le k_1,k_2 \le N-1$.

A standard approach to computing the 2-D FFT of an NxN matrix is to perform a 1-D FFT on the rows of the matrix, followed by a 1-D FFT on the columns. The number of arithmetic operations required will therefore be $O(N^2 \log_r N)$, where r is the radix and N is the matrix size.

3.2.1 The Parallel Algorithm for 2-D FFT

Let $N=qp$, where N is the order of the squared input matrix, p is the number of (1-D FFT) processors and q is an integer. Each processor is allocated a unique working set of rows/columns. The algorithm consists of following four steps:

Step 1. 1-D FFT on rows: Processor i computes 1-D FFT on rows (qi, qi+1,…,qi+q-1) of input matrix, where i=0,1,…p-1. Because each processor executes, in parallel, a 1-D FFT on q different rows, this step has the computational complexity of : $O((N^2/p)\log_2 N)$.

Step 2. Transpose the matrix: Row/Column transposition of matrix is required before execution of FFT on columns.

Step 3. 1-D FFT on columns: Same as in Step-1, but on columns.

Step 4. Final matrix transposition of the result matrix.

4. FFT Architectures

4.1 (1-D) FFT Architecture

The 1-D FFT architecture consists of a single DIF radix-2,Radix-4, Split-Radix or FHT butterfly, two (single in case of FHT) dual-port RAMs and an address generator unit. The simplified architectural block diagram of the 1-D FFT design depicted in Figure 3.

Input and Output Data Memory : The 1-D FFT architecture supports two different memory modes, internal and external working modes.

i) Internal Memory Mode: There are two different internal memory configurations – single-memory and dual-memory configuration (also called as double-buffering method).

The single-memory configuration provides the simplest memory interface. In this mode, memory A is used as input, working and output memory. Input data is loaded into memory A (data load phase), then FFT is started (computation phase) and when the FFT is complete, the result vector is read out of the memory. In this mode, memory B is not used.

In the dual memory configuration; input, computation and output operations are overlapped, so that the FFT processor is never left in an idle state waiting for an I/O operation. During the execution of input data on memory A (*or B*), a new vector of input data is written into memory B (*or A*), concurrently. This mode provides high throughput rates for real-time applications.

ii) External Memory Mode: In external memory mode, real and imaginary parts of input and output data and the twiddle coefficients are stores in external SRAM memory banks. The data transfer mechanism to and from the external SRAM memory banks has been designed for use in a wide variety of applications. The storage of input/output data in external SRAM memory rather than internal memory (i.e. block RAM, distributed RAM) allows transformations up to 1M-point without the need for additional internal buffering.

Data Format : To provide the system designer with maximum flexibility, the input/output data and the twiddle factor word-lengths have been described as parameters so that the FFT designs can be used in a wide range of applications such as image/video processing, communications.

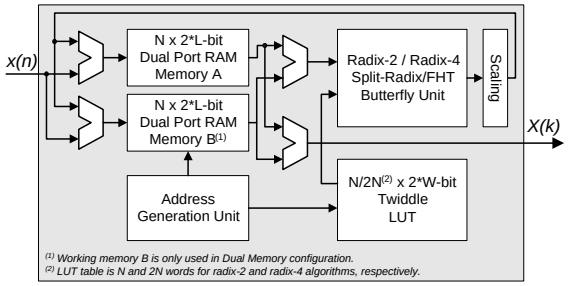

Figure 3. Functional block diagram of 1-D FFT architecture.

The input and output data is represented by *L*-bit fixed-point real and imaginary components (just real component in case of FHT) in the two's complement format. The input/output data is loaded into/from the FFT processor in the normal order. The twiddle factors (sine and cosine values), which are generated by the FFT processor internally, have been represented by *W*-bit fixed-point accuracy in the two's complement format.

Butterfly Unit : The butterfly operation is the heart of the FFT algorithm. It takes data words from memory and computes the FFT. The results are written back to the same memory locations since an in-place algorithm is used. The butterfly element is pipelined in order to compute a result every clock cycle. It consists of parallel *L*-bit multipliers, *L*-bit and (*L*+*W*)-bit adders/subtractors. The hardware utilization for the butterfly elements of each algorithm are summarized in Table 1. The block diagram of how the butterfly operation for radix-2 algorithm is partitioned between 7 pipeline stages is shown in Figure 4.

In the computation of the FFT, scaling of the intermediate results is necessary in order to prevent overflows. To avoid possibility of overflow, each stage of the FFT is scaled down by a factor of 2 and 4 in radix-2/FHT and radix-4/split-radix algorithms, respectively. In this way, the final FFT output is scaled down by N.

Address Generation Unit (AGU) : The purpose of the AGU is to provide the I/O RAMs and twiddle coefficient Look-Up Tables (LUTs) with the correct addresses. Since address generation during input, output and FFT computation processes are different, it keeps track of the mode of the operation and generates the required address. It is also responsible of unscrambling (bit-reversal) of output data at the end of each FFT execution.

Table 1. Hardware requirement comparison for different FFT algorithm butterflies.

FFT Algorithm	*L*-bit Multiplier	*L*-bit Add/Subt	*(L+W)*-bit Add/Subt
Radix-2	4	4	2
Radix-4	12	16	6
Split-Radix	8	16	4
FHT	4	6	2

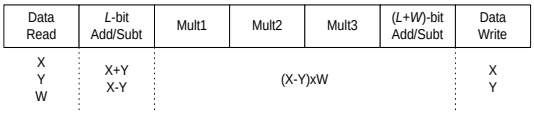

Figure 4. Pipeline diagram of DIF-FFT Radix-2 butterfly.

4.2 Parallel 2-D FFT Architecture

Figure 5 illustrates the structure of proposed parallel 2-D FFT processor architecture. This architecture is structured with *p* Processor Elements (PE) which share 4 memory banks (M_0,…M_3) under the control of one Control Unit (CU). Each PE_i is essentially a 1-D FFT processor with attached working registers and Processor Element local Memory (PEM_i) for the storage of associated row/column vectors. The 1-D FFT processor includes one of the *N*-point 1-D FFT cores (radix-2, radix-4, split-radix and FHT) associated with related twiddle factors LUTs.

The local memory PEMi consists of two (single in case of FHT) dual-port *N*x*L*-bit deep memories in order to store the real and imaginary (just real in case of FHT) parts of the *L*-bit precision input/output samples. A three-stage sequence of operations is used to compute transforms with this PE interface, as follows:

Step 1. The associated input row/column data vector is transferred into the local memory PEM_i of processor *i;*

Step 2. When the input data load operation has completed, the 1-D FFT PE is started.

Step 3. When the 1-D FFT is completed, the result vector is read out of the local memory. The bit-reversal operation according to FFT type is performed on the fly by the AGU, so that no clock cycles are wasted.

The CU serves as the "operating manager" of the entire system. The functions of the CU include resource (external shared memory) management and supervision of the parallel execution of PEs. A processor ID is allocated by CU to each PE. In this way CU keeps track of the associated row/column vector set for each PE. CU handles the scheduling of shared memory through two circular queues for reading and writing requests. Each PE sends its associated processor ID to the CU (read or write queue), when it is ready to accept a new input row/column vector or when it completes the1-D FFT.

The Memory Interconnection (MI) that includes a crossbar switch is designed to provide access for multiple memory requests. The service discipline for memory requests is such that no PE gets accesses to a memory bank while another is attempting to access the same bank.

The external shared memory consists of 4 memory banks which are directly connected to FPGA. Each memory bank can contain up to 2MBytes (arranged as 512Kx32-bit) of data giving a total physical address space of 8MBytes. The real part of the input/intermediate/output matrix is stored in the first two banks (bank0 and bank1) while the imaginary part is stored in bank2 and bank3.

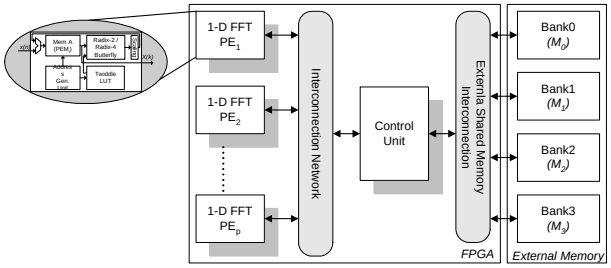

Figure 5. Functional block diagram of parallel 2-D FFT architecture.

5. FPGA Implementation

A parametrisable Handel-C coding approach has been used to implement proposed (1-D and 2-D) FFT designs to be independent of I/O data wordlengths (*L*,*W*), input vector/matrix size (*N*) and number of PEs (*p*).

The implementations target the Celoxica RC1000 PCI-based FPGA development board. The RC1000-PP board used is a standard PCI bus card equipped with the Virtex-E2000 FPGA chip (package :bg560, speed grade 6). It has 8MBytes of SRAM directly connected to the FPGA in four 32-bit wide memory banks. All are accessible by the FPGA and any device on the PCI bus in parallel. A schematic block diagram of the RC1000 board is shown in Figure 6.

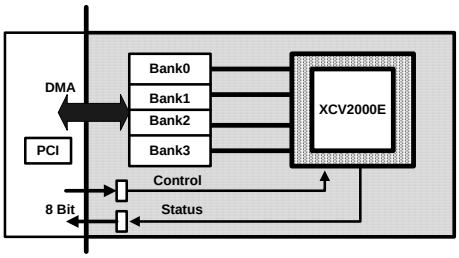

Figure 6. Schematic view of the FPGA/Banks part in the RC1000-PP board.

5.1 1-D FFT Implementation

Having implemented all the algorithms under a common framework, the performance of four different 1-D FFT algorithms have been comprehensively evaluated for the FFT lengths from 1K-point to 256K-point. Figure 7 illustrates the performance results of each algorithm in terms of maximum system frequency and chip area. It is worth mentioning that the radix-2 outperforms other algorithms in terms of area and system frequency.

Table 2. Computation time across different algorithms for a 1024-point FFT.

Design	Freq (MHz)	Comp. Time (μsec)
Radix-2	84	121.6
Radix-4	80	68
Split-Radix	63	160
FHT	60	343

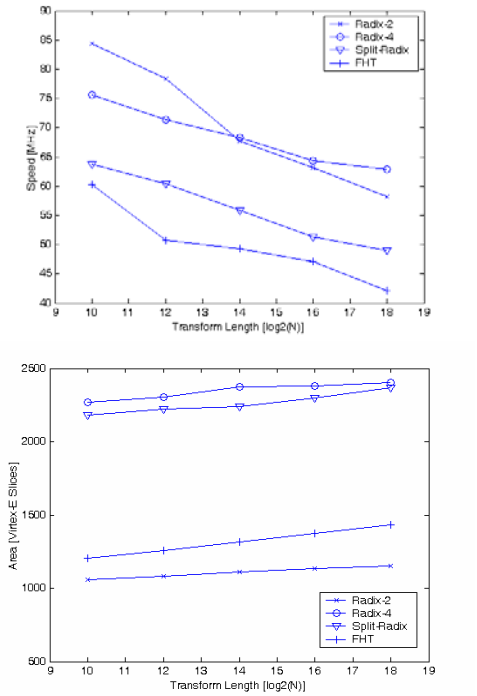

Figure 7. Performance results, influence of transform length.

Computation time across implemented algorithms for a 1024-point FFT is shown in Table 2. It is consistently seen in our evaluation that the radix-4 is by far the most efficient algorithm in terms of computation time. If the choice of algorithm is to be made solely based on memory requirements, the FHT algorithm is the best since the memory requirement of the FHT is exactly half of the other algorithms.

5.2 Parallel 2-D FFT Implementation

The 2-D FFT design has been implemented for the following matrix sizes (N=128, 256, 512, 1024) and number of PEs (p = 1,2,4,8) considering the hardware resources available on Virtex-2000E FPGA chip. Again, the designs are completely parametrisable, so higher input matrix sizes and number of PEs can be implemented on advanced development platforms (such as Virtex-II or Apex-II FPGAs interfaced with larger external memory).

The speed-up of a parallel algorithm is defined as $S_p=T_s/T_p$, where T_s is the execution time using one processor and T_p is the time of the algorithm executed using p processors. Figure 9 shows the computation time and speed-up figures as a function of the number of processors obtained for parallel 2-D FFT implementation based on different (*1-D*) FFT algorithms described in the previous section, for the matrix sizes N = 256 and 1024.

Shared memory designs are strongly affected by the design of memory arbitration unit. The speed-up decreases with an increase of the PEs because memory of

conflict delay increases. This effect can be seen in Figure 9, where speed-up figures are plotted against the number of processors in the system. The performance of parallel 2-D FFT design is also affected by type of FFT algorithm used.

Table 2. Frames per second (fps) for 2D-FFT (*a:* based on Radix-2, *b*: based on Radix-4 algorithm).

	Matrix Size					
	128	256		512	1024	
P	a	a	b	a	a	b
1	112	25	43	6	2	4
2	215	47	78	12	4	8
4	382	87	107	20	8	16
8	420	94	99	35	13	12

It is worth mentioning that the transposition steps defined in the parallel 2-D FFT algorithm in section 3.2.1 is implemented simultaneously with the transfer of column data vectors to PE's local memory with no delay penalty.

Table 3 compares the performance of parallel 2-D FFT processor implementations based on Radix-2 and Radix-4 FFT algorithms for different matrix size and number of processor values. It can be seen that a real-time performance is obtained using just one processor for the matrix sizes N=128 and 512. It is also worth noting that a real-time performance is obtained for the matrix size N=512 with 8 PEs, based on 1-D Radix-2 FFT core, using %45 of the slices and %30 of the BlockRAMs available on Virtex-2000E FPGA chip. For matrix size N=1024, it is possible to process 16 frames-per-second, with 4 PEs based on 1-D Radix-4 FFT core, which meets medical/astronomical image processing frame rate requirements.

Figure 10 illustrates the performance results of parallel 2-D FFT implementation in terms of maximum system frequency (fmax) and chip area against number of PEs for the matrix sizes N=256 and 1024. The chip area requirement increases linearly as a the number of PEs increases for all of the (1-D) FFT algorithms while fmax slightly decreases.

Table 3 shows the performance comparison of existing (FPGA-based) 2-D FFT works in terms of frame rate per second. Shirazi et.al. [20]. implemented 2-D FFT on a custom computing machine called Splash-2. Dick [21] proposed a reconfigurable architecture for 2-D FFT using polynomial transforms on XC4000E FPGA device. He proposed that his architecture is %46 efficient than a row-column processor. Dillon[22] developed a parallel system using two Virtex-II devices and achieved the performance of 120 fps. Our design shows improvements when compared to [20] and [21] in terms of performance. Although high performance can be achieved in [23], it is not a suitable FPGA solution for low cost applications.

Table 3. Performance comparison with existing designs based on FPGAs.

Design	FPGA Used	Input Size	Frame Rate
Shirazi et al. [20]	2 x XC4000E	512x512	2.12 fps
Dick [21]	XC4000E	512x512	24 fps
Dilon Eng. [22]	2 x XC2V6000	2048x2048	120 fps
Proposed	XCV2000E	512x512	35 fps

Since the 2-D FFT is frequently used as a benchmark for performance measuring, Table 4 gives performance results for a relevant comparison between our implementation and other implementations on multiprocessor platforms.

The data given by Table 3 and Tables 4 can be used to predict the usability of the proposed architecture as a general 2-D FFT engine.

Table 4. Confronting 2-D FFT performances on multiprocessor based platforms (in miliseconds).

Platform	1024x1024			256x256		
	1PE	4PE	16PE	1PE	4PE	6PE
Cavadini et.al [24] [(1)]	90.7	23.0	6.4	6.1	1.9	0.83
Hartley et.al. [25] [(2)]	3164	1169	264	157.5	62.7	13.9
Sgro [26] [(3)]	1045	272	74	52.2	13.6	3.7
Ercan [23] [(4)]	-	-	-	-	45	-

[(1)] Frequency Domain Engine (FDE) based on VLSI SPITFFIRE FFT Processor
[(2)] Multiprocessor system based on TI TMS320C40 DPS.
[(3)] Multiprocessor system based on SHARC DSPs.
[(4)] Multiprocessor system based on 4-SHARC DSPs.

6. Application Case Study: Frequency Domain Image Filtering

The frequency domain image filtering is one of the most important applications where 2-D FFT can be applied. The 2-D convolution in spatial/time domain is commonly used for image filtering. It is fast and easy if the input image and the filter kernel being used are relatively small. But, as the image or kernel grows in size the computational complexity increases geometrically. Filtering of large images can be done much faster in frequency domain using 2-D FFT, based on the convolution property of the Fourier transform as follows [27]:

Step 1. Compute 2-D FFT of input image and filter: FFT{I(x,y)} and FFT{H(x,y)}.

Step 2. Apply filter H(u,v) to the FFT of input image by point-by-point multiplication: Y(u,v) = I(u,v)*H(u,v).

Step 3. Compute the 2-D IFFT of result: IFFT{Y(u,v)}

The speed-up is approximately $N^2/N \log_r N = N/\log_r N$, which is significant when dealing with large images.

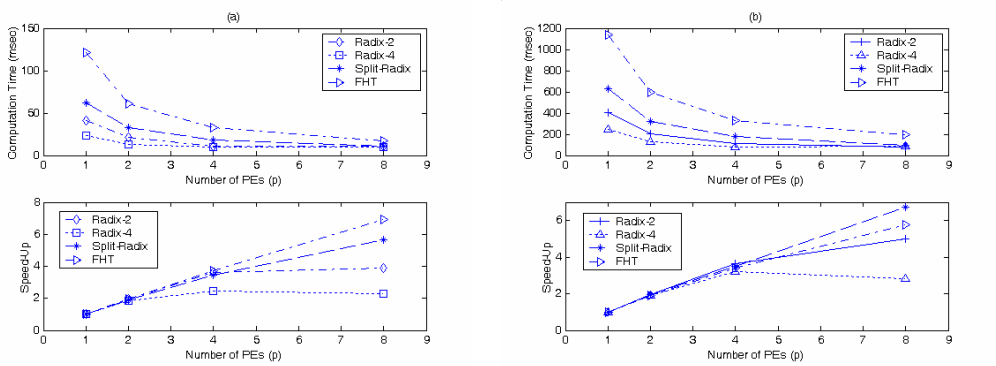

Figure 9. Computation time and speed-up versus number of PEs: (a) matrix size of 256x256. (b) matrix size of 1024x1024.

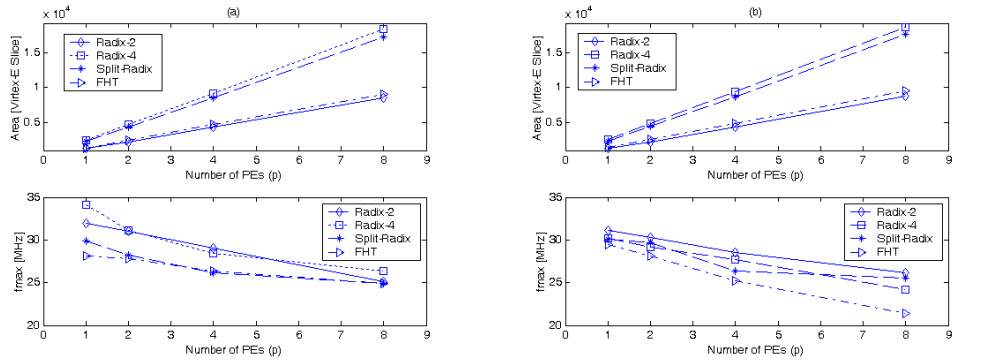

Figure 10. Area and fmax versus number of PEs: (a) matrix size of 256x256 (b) matrix size of 1024x1024.

6.1 Environment for Frequency Domain Image Filtering Application

Figure 11 illustrates the proposed environment for 2-D frequency-domain image filtering application. The environment consists of a host application (GUI), an image database and a single FPGA-chip coprocessor based on the RC1000 development board.

- Host Application(GUI): The GUI enables application users to concentrate on experimenting conveniently with different FFT algorithms and architectures to investigate best system trade-offs (such as area, speed, performance) rather than concentrating on the low level (and complex) structure of FPGAs. It gives the user the ability to set different design parameters according to application requirements (frame rate, image size) and hardware resources (FPGA area, memory etc.). The parameters include forward and inverse FFT core algorithm type (radix-2, radix-4, split-radix, FHT) the number of 1-D FFT/IFFT PEs (p=1,2,4,8) and filter type (band-pass, high boost etc.) along with associated filter parameters.
- Image Database: Image database includes sample test images and application specific images such as large medical and astronomical images.
- FPGA Coprocessor: The FPGA Coprocessor is based on RC1000 PCI development board. The frequency domain filtering is completely performed by FPGA co-processor. The real and imaginary parts of the input image are stored in SRAM Bank0 and Bank1 while the filter coefficient matrix H(u,v) is stored in Bank2 and Bank3 using 16-bit fixed-point format.

7. Conclusion

Due to the importance and use of FFT in many signal and image processing applications, a range of 1-D FFT algorithms including radix-2, radix-4, split-radix and FHT have been implemented using proposed parametrisable framework architecture. In addition, an efficient for the implementation of 2-D FFTs has also been proposed and implemented. A complete environment based on the developed parallel 2-D FFT architecture has been presented as a solution for 2-D frequency-domain image filtering application. The performances of implementations have been discussed, investigated and compared with existing works. High speed-up and efficiency have been attained for the parallel implementation of 2-D FFT compared to existing works.

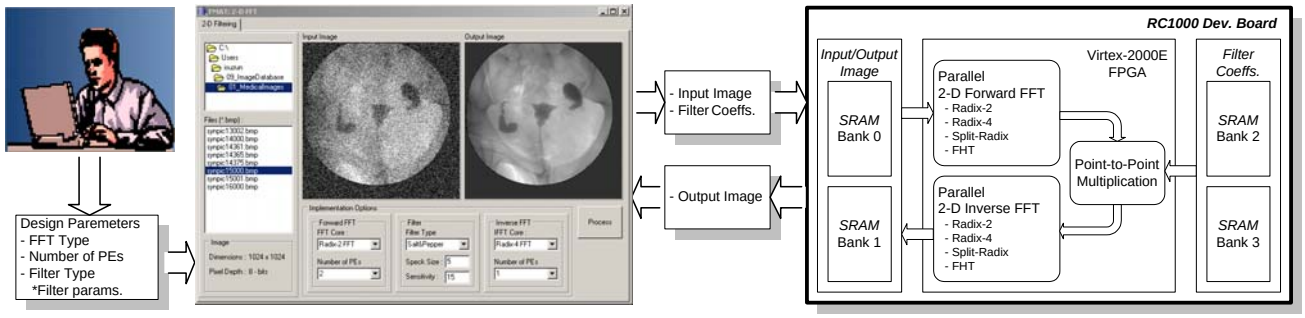

Figure 11. FPGA-based Frequency Domain Image Filtering Environment.

8. References

[1] J.W. Cooley and J. W. Tukey, "An Algorithm for the Machine Computation of the Complex Fourier Series," Math.of Computation, Vol. 19, April 1965, pp. 297-301.

[2] R.N. Bracewell, “Discrete Hartley Transform”, J. Opt. Soc. Amer., vol. 73, no. 12, pp. 1832–1835, 1983.

[3] Frigo, M. and Johnson, S. G, “FFTW: An Adaptive Software Architecture for the FFT”, ICASSP conference proceedings, 3:1381-1384, 1998.

[4] Ganapathiraju, A. et.al, “Contemporary View of FFT Algorithms”, Proc. of the IASTED, pp. 130-133, 1998.

[5] Datasheet, "Analog Devices DSP Selection Guide 2002 Edition", Analog Devices, 2002.

[6] Datasheet, "TI C62x and C67x DSP Benchmarks", Texas Instruments, 2002.

[7] Datasheet, "Motorola DSP 56600 16-bit DSP Family Datasheet”, Motorola Ltd.,2002.

[8] M. Wosnitza: "High Precision 1024-point FFT Processor for 2-D Object Detection", Ph.D. Thesis, ISBN 3-89649-443-0, 1999.

[9] Baas, B. M., “A low-power, high-performance 1024-point FFT processor”, IEEE Journal of Solid State Circuits. pp. 380-387, 1999.

[10] URL: www.xilinx.com.

[11] Datasheet, "FFT Megacore Function User Guide", Altera Ltd., 2002.

[12] Datasheet, "Xilinx 1024-point FFT/IFFT Core Datasheet", Xilinx Ltd., 2002.

[13] Datasheet, "CS248 FFT/IFFT Core Datasheet", Amphion Ltd., 2002.

[14] Datasheet,"FFT/WinFFT/Convolver Transform", Mentor Graphics, 2002.

[15] URL: www.celoxica.com.

[16] Datasheet, "RC1000 Reconfigurable hardware development platform", Celocixa Ltd., 2001.

[17] E.O. Brigham, "The Fast Fourier Transform and its Application", Prentice Hall, 1988.

[18] C.S. Burrus and T.W. Parks, "DFT/FFT and Convolution Algorithms", Wiley, New York, 1985.

[19] P. Duhamel, "Implementation of split-radix FFT algorithms for complex, real and real-symmetric data" IEEE Trans. on ASSP, vol. 34, pp. 285-295, April 1986.

[20] N. Shirazi et.al, "Implementation of a 2-D Fast Fourier Transform on a FPGA-Based Custom Computing Machine", IEEE Symposium on FPGAs for Custom Conf. Comp. Mach., September 1999.

[21] C. Dick, "Computing Multidimensional DFTs Using Xilinx FPGAs", The 8th Intr. Conf. On Sig. Pro. App. And Tech., September, 1998.

[22] T. Dillon, "Two Virtex-II FPGAs Deliver Fastest, Cheapest, Best High-Performance Image Processing System", Xilinx Xcell Journal, 41, 2001.

[23] M.F. Ercan, “A Parallel Architecture for Knowledge Based Autonomous Navigation,”, Singapore Robotic Games 2003 Symposium, 2003

[24] Cavadini, M. et.al., “Multiprocessor system for high-resolution image correlation in real time”, IEEE Transactions on VLSI Systems, 9:439 –449, 2001.

[25] D. A. Hartley and S. P. Kshirsagar, “Architecture and modeling of a parallel digital processor based image processing system,” in Proc. SPIE Int. Society Optical Eng., vol. 2308, 1994, pp. 1807–1815.

[26] J. A. Sgro, “An efficient MIMD/SIMD architecture for the analog devices SHARC(tm) DSP,” Tech. Rep., Alacron, Inc., 1998.

[27] R.C. Gonzales and R. E. Woods. "Digital Image Processing", Addison-Wesley, 2002.