Implementation of a 2-D Fast Fourier Transform on a FPGA-Based Custom Computing Machine

Nabeel Shirazi, Peter M. Athanas, and A. Lynn Abbott
Virginia Polytechnic Institute and State University
Bradley Department of Electrical Engineering
Blacksburg, Virginia 24061-0111

Abstract. The two dimensional fast Fourier transform (2-D FFT) is an indispensable operation in many digital signal processing applications but yet is deemed computationally expensive when performed on a conventional general purpose processors. This paper presents the implementation and performance figures for the Fourier transform on a FPGA-based custom computer. The computation of a 2-D FFT requires $O(N^2 \log_2 N)$ floating point arithmetic operations for an NxN image. By implementing the FFT algorithm on a custom computing machine (CCM) called Splash-2, a computation speed of 180 Mflops and a speed-up of 23 times over a Sparc-10 workstation is achieved.

1 Introduction

Two dimensional convolution is a fast and simple way of filtering an image in the spatial domain if the template being used is relatively small (i.e., 8x8 pixels). As the template grows in size, the computational burden increases geometrically. Convolution of larger templates can be done much faster by converting an image in the spatial domain to the frequency domain and then applying a filter by doing point-by-point multiplication[6]. The filtered image in the frequency domain is then converted back to the spatial domain by doing an inverse Fourier transform.

Image and digital signal processing (DSP) applications typically require high calculation throughput [4,10]. The 2-D fast Fourier transform application presented here was implemented for near real-time filtering of video images on the Splash-2 FPGA-based custom computing machine (CCM). This application requires the ability to do floating point arithmetic. The use of floating point allows a large dynamic range of real numbers to be represented and helps to alleviate the underflow and overflow problems often seen in fixed point formats. An advantage of using a CCM for floating point is the ability to customize the format and algorithm data flow to suit the application's needs.

An overview of the FFT algorithm and the method used for filtering video images are given in Section 2. A description of the floating point format used in the application is given in Section 3. The implementation of the 2-D FFT on the Splash-2 architecture is shown in Section 4. In Section 5, error analysis is presented to show that the chosen floating-point format used is adequate for this application. The performance of this implementation of an FFT was compared to a wide range of architectures in Section 6.

2 Image Filtering using the Fourier Transform

An example illustrating the application of the Fourier transform to images is shown in Figure 1. An exponential highpass filter is used to attenuate the low frequency components in order to perform edge detection. The black pixels of the filter in Figure 1 correspond to zero values and the white pixels correspond to values of one with the remaining gray pixels ranging between 0.0 and 1.0. The four corners of the images in the frequency domain are the locations of the low frequency components. The high-frequency components are located near the center of the image.

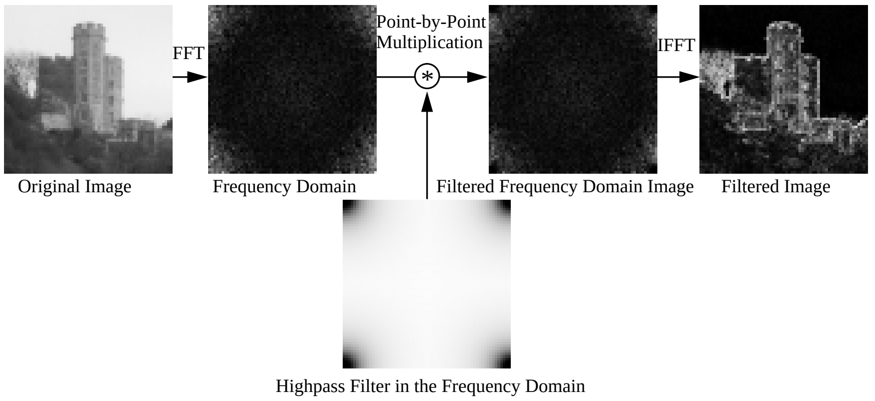

Original Image
FFT
Frequency Domain
Point-by-Point Multiplication
*
Filtered Frequency Domain Image
IFFT
Filtered Image
Highpass Filter in the Frequency Domain

Figure 1: Fourier Transform Filtering Method**.**

2.1 Discrete Fourier Transform

The 2-D DFT of a NxN image, *f(x,y)*, is defined by the expression,

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\, e^{-j2\Pi(xu+vy)/N}$$

for $u, v \in [0, N-1]$

The 2-D DFT expression can be decomposed into multiple 1-D Fourier transforms. The above equation can be expressed in the form:

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1} F(x,v)\, e^{-j2\Pi ux/N}$$

for $u \in [0, N-1]$

and,

$$F(x,v) = \frac{1}{N}\sum_{y=0}^{N-1} F(x,y)\, e^{-j2\Pi vy/N}$$

for $x, v \in [0, N-1]$

where $W^k_N = e^{-j2\pi x/N}$ or $e^{-j2\pi y/N}$ and is called the twiddle factor.

This shows that an NxN 2-D DFT can be computed by first performing N 1-D DFTs (one for each row), followed by another N 1-D DFTs (one for each column).

2.2 Fast Fourier Transform

The fast Fourier transform algorithm (FFT) consists of a variety of tricks for reducing the computation time required to compute a DFT[10]. The number of complex multiplications and additions required to implement an N-point DFT is proportional to N^2. The 1-D DFT can be decomposed so that the number of multiply and add operations is proportional to $N \log_2 N$. The FFT algorithm achieves its computational efficiency through a divide and conquer strategy. The essential idea is a grouping of the time and frequency samples in such a way that the DFT summation over N values can be expressed as a combination of two point DFTs. The two point DFTs are called butterfly computations and requires one complex multiply, and two complex additions to compute. The notation used for a butterfly structure is shown in Figure 2. By using the FFT partitioning scheme, an 8 point FFT can be computed as shown in Figure 2. Each stage of the N point FFT is composed of N/2 radix-2 butterflies and there are a total of $\log_2 N$ stages. Therefore there are a total of $(N/2)\log_2 N$ butterfly structures per FFT. In addition, the input is in bit-reverse order and the output is in linear order. A 2-D FFT can be decomposed into two arrays of 1-D DFTs, each of which can be computed as a 1-D FFT.

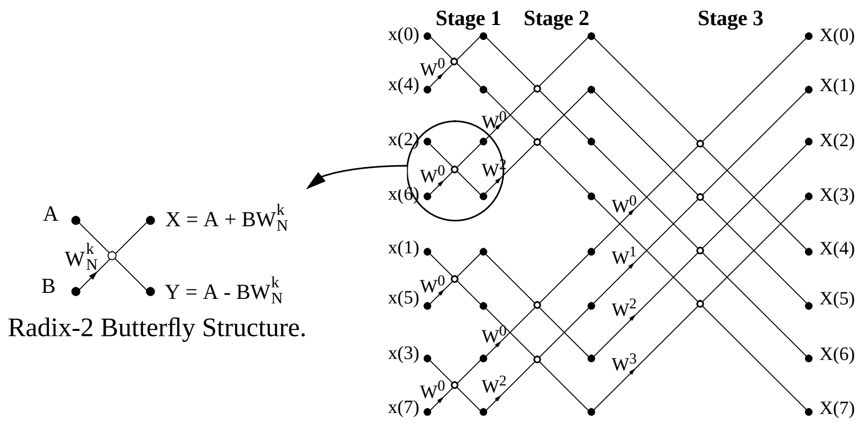


Figure 2: Decimation-in-Time Eight Point FFT.

3 Floating Point Arithmetic

In order to implement an FFT on Splash-2, [1,2] floating point arithmetic adder/subtracter and multiplier units were selected to satisfy the numerical dynamics of this application [12]. Until recently, any meaningful floating point arithmetic has been virtually impossible to implement on FPGA based systems due to the limited density, routing resources and speed of older FPGAs. In addition, mapping difficulties occurred due to the inherent complexity of floating point arithmetic. With the introduction of high level languages such as VHDL [9], rapid prototyping of floating point formats has become possible making such complex structures more feasible to implement. Although low level design was possible, the strategy used in all application development was to specify all aspects of the design in VHDL and rely on automated synthesis to generate the FPGA mapping.

3.1 Floating Point Format Representation

The floating-point format used in this application is similar the IEEE 754 standard for storing floating point numbers [7]. For the FFT implementation presented here, a smaller 18-bit floating-point format was developed. The format was chosen to accommodate two specific requirements: (1) the dynamic range of the format needed to be quite large in order to represent very large and small, positive and negative real numbers accurately, and (2) the data path width into one of the Xilinx 4010 processors [14] of Splash-2 is 36 bits wide and real and imaginary operands of a complex number are needed to be input on every clock cycle. Based on these requirements the format in Figure 3 was used.

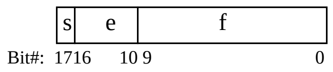


Figure 3: 18 Bit Floating-Point Format.

The 18 bit floating point value (v) is computed by:

$$\mathbf{v = -1^{s}\, 2^{(e\,-\,63)}(1.f)}$$

The range of real numbers that this format can represent is $\pm 3.6875 \times 10^{19}$ to $\pm 1.626 \times 10^{-19}$.

3.2 Floating-Point Addition/Subtraction and Multiplication

The aim in developing a floating point adder/subtracter routine was to pipeline the unit in order to produce a result every clock cycle. The floating point addition and subtraction algorithm that was implemented is similar to what is done in most traditional processors; however, the computation is performed in three stages to improve performance. A summary of the resulting size and speed of the 18-bit floating point units is given in Table 1.

Floating point multiplication is much like integer multiplication. Because floating-point numbers are stored in sign-magnitude form, the multiplier needs only to deal with unsigned integer numbers. Like the architecture of the floating point adder, the floating point multiplier unit is a three stage pipeline that produces a result every clock cycle. The bottleneck of this design was the integer multiplier. For more information regarding the algorithms used the reader is referred to [12].

The Synopsys Version 3.0a VHDL compiler was used along with the Xilinx 5.0 tools to compile the VHDL description of the floating point arithmetic units. The Xilinx timing tool, *xdelay*, was used to estimate the speed of the designs.

	Adder/ Subtracter	Multiplier
FG Function Generators	28%	44%
Flip Flops	14%	14%
Stages	3	3
Speed	8.6 MHz	4.9 MHz
Tested Speed	10 MHz	10 MHz

TABLE 1. Summary of Properties of 18-bit Floating Point Units.

The floating point arithmetic units have also been incorporated in another application: an FIR filter [13]. The FFT application operates at 10 Mhz and the results of the transform are stored in memory on the Splash-2 array board. These results were checked by doing the same transform on a Sparc workstation and we noted the results matched. Therefore, the maximum clock speed of the arithmetic units given by the *xdelay* program is conservative and we conclude that the arithmetic units can operate at least at 10 MHz.

4 FFT Implementation

Implementing filtering method discussed in Section 2 involved mapping a 2-D FFT, a filter, and a 2-D IFFT to a two-board, Splash-2 system [1, 2]. The filtering method was constructed in such a way that the FFT and the IFFT are computed in parallel and are continuously provided video images from a frame buffer. This section discusses the recirculation method used to implement the FFT, the butterfly operator used in the FFT, and the filtering process.

4.1 FFT Recirculation Method

To calculate a 2-D FFT, a method requiring the recirculation of data through a butterfly operator was implemented. A block diagram of this method is shown in Figure 4. Two banks of memory are used to store the input and output data of each stage of the FFT. A bank of memory consists of three processing elements that store the real and imaginary components of the two 18-bit floating point numbers into their local memories. Since the local memories are only 16 bits wide, the two 18-bit floating point values are divided between the three memories. To compute an FFT on an input image, a frame of data is accepted from the frame buffer, converted from 8-bit integer values to 18-bit floating point values, and stored into Bank 1. The 2-D FFT is computed by first computing a 1-D FFT of each row of the image and then a 1-D FFT on each column of the row transforms. The 1-D FFT is computed in the same manner as shown in Figure 2. The first stage of the FFT is computed by reading each row of data points in bit-reversed order from Bank 1 and passing it to the butterfly operation. The results of the butterfly operation are stored in the second bank of memory. Once each butterfly is computed in the first stage, the second stage is computed by first reading the data out of the second bank of memory in linear order, and then into the butterfly operator. The results of this stage are stored in Bank 1. The recirculation method continues by reading data out of one bank of memory while the other bank of memory is storing the results. The recirculation terminates when a 1-D FFT is calculated on each row of the image. The second set of 1-D FFTs are computed in the same manner except it is done on each column of the result of the first set of 1-D FFTs. Once the final stage of the last FFT is calculated, the data is passed over the crossbar from X11 to X15 where the data is filtered. The complete 2-D FFT process involves $2N^2 \log_2 N$ passes through the butterfly operator.

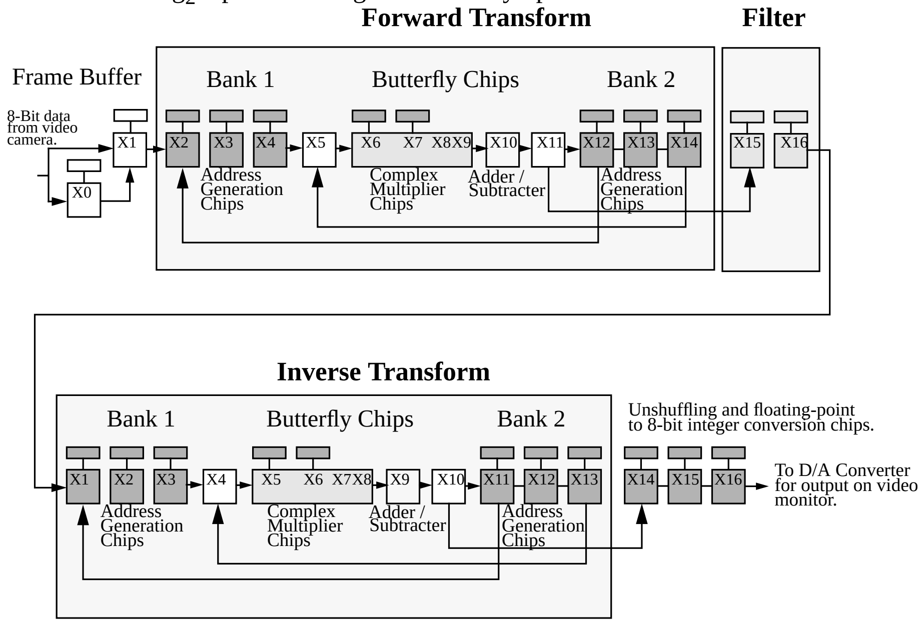


Figure 4: Splash-2 FFT Image Filtering Method.

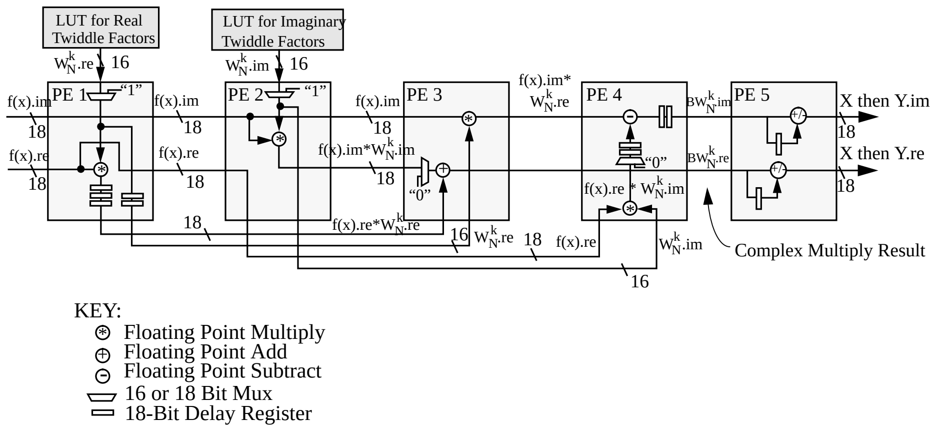
4.2 Butterfly Implementation

The butterfly operation is the heart of the FFT algorithm. It is pipelined in order to compute a real and complex result every clock cycle. The butterfly diagram shown in Figure 2 involves calculating a complex floating point multiplication and two floating point additions/subtractions. The complex multiply involves four multiplications and two additions/subtractions. In total, eight floating point operations are calculated every clock cycle at 10 Mhz. The throughput of the butterfly operation is therefore 80 Mflops.

Figure 5, shows a block diagram of how the butterfly operation was partitioned between five processing elements on Splash-2. The real and imaginary parts of the complex multiplication of BW^k_N is given respectively by the equations:

$$BW^k_N.re = B.re\, W^k_N.re + B.im\, W^k_N.im \qquad (4.1)$$

$$BW^k_N.im = B.re\, W^k_N.im - B.im\, W^k_N.re \qquad (4.2)$$



Note: f(x) = Input Data, A in the first clock cycle and then B in the second clock cycle.

Figure 5: Block diagram of a five PE Splash-2 design for a butterfly operation.

Both the A and B inputs of the butterfly operation shown in Figure 2 are denoted in Figure 5 as *f(x)*. The *A* value is inserted into the pipeline followed by the *B* value on the next clock cycle. The A input is not multiplied by the twiddle factor. In order to pass the *A* value through the pipeline without changing its value, multiplexers are used to multiply it by one and add zero to it. When the real and imaginary values of B are inserted into the pipeline, these pass through four processing elements in order to calculate the complex multiply of BW^k_N. The first processing element (PE 1) reads the real component of the appropriate twiddle factor and multiplies it by real component of B. The result and the twiddle factor is passed via the crossbar to PE 3, and the real and imaginary components of B are passed to PE 2. The second PE multiplies the imaginary component of B and the appropriate twiddle factor, and the result is passed to the third PE. The third PE reads the result from PE 1 (B.reW^k_N.re) off the crossbar and adds it to the result from PE 2 (B.imW^k_N.im) to produce the final result of the real component of the complex multiply (BW^k_N.re). The imaginary component, BW^k_N.im, of the complex multiply is computed in the same manner in PEs 3 and 4. The butterfly operation is completed by adding A to BW^k_N in the first clock cycle to produce X, and subtracting BW^k_N from A in the second clock cycle to produce Y.

The 18-bit format was not used to store the twiddle factors in the local memories of PE 1 and 2 since the memory data bus width is only 16 bits wide. A smaller 16-bit format was created by decreasing the exponent field of the 18 -bit floating point number by 2 bits. Since twiddle factors can be expressed in terms of sine and cosine functions by using Euler's rule, the value of the floating point number will never have an exponent greater than 0 (because the value will always be less than or equal to one). Because of this, the exponent field was changed to range from 0 to -31 instead of 63 to -63 in order to decrease the size of the exponent field from 7 bits to 5 bits. When the twiddle factor is read into the processing element, a conversion is done from the 16-bit format to the 18-bit format used in the arithmetic units.

4.3 Filtering

Once the input image is transformed to the frequency domain, point-by-point multiplication of the matrix filter coefficients, H(u,v) and the transformed image can be computed to filter the image. The values of the elements of the matrix H(u,v), range between 0 and 1.0 and are stored in the local memories in the same manner as the twiddle factors of the butterfly operation. The filter coefficients are calculated before run-time and stored in the local memories of chips X15 and X16 as indicated in Figure 4. Filter chips consist of a floating point multiplier unit and filter coefficient addressing logic. X15 and X16 are used to filter, respectively, the real and imaginary components of the transformed image.

Many different types of filters have been calculated, such as, ideal, Butterworth, exponential and trapezoidal filters[6]. These filters can be down-loaded on the fly from the Sparc-2 host to the local memories of the Splash board in approximately 400ms.

5 Error Analysis

To test round-off error associated with 18-bit floating point format, a forward FFT followed by an inverse FFT was calculated without doing any filtering This process should ideally result in an image which is exactly the same as the original image. However, due to round off error the output image differed slightly.

Statistics such as RMS and absolute error were calculated to quantify the error. Equations used for calculating the RMS and absolute error are:

$$\textit{RMS Error} = \sqrt{\frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}(I(x,y)-I'(x,y))^2} \qquad (5.1)$$

$$\textit{Absolute Error} = \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}|I(x,y)-I'(x,y)| \qquad (5.2)$$

where I(x,y) is the original image, and I'(x,y) is the output image.

Multiple images were tested and the average RMS error was 0.4% and the average absolute error was 0.2%. Each pixel value can have a gray-scale value from 0 to 256. The output image had a maximum deviation of 2 gray-scale values from the corresponding pixel in the original image. Subjectively, no difference could visually be seen between the original and output images.

The calculated statistical values indicate that the smaller 18-bit floating-point format is adequate for this application. By down-sizing the floating point format we were able to do more floating-point operations per Splash board resulting in increased performance.

6 Performance

In order to compare the performance of this application thoroughly, a wide range of architectures were selected. The architectures ranged from a general purpose workstation to special purpose DSP processors. Since the FFT is a common DSP algorithm, it is used as a benchmark by many DSP chip manufactures. Two DSP chips were selected; one which has approximately the typical performance of a DSP chip, and one which is representative of high end performance.

The test case used to evaluate the different architectures is a 2-D spatial filter of dimensions 512x512 pixels. This process involves performing a 512x512 2-D FFT, filtering the image by doing 512x512 point-by-point multiplications for each of the real and imaginary values of the image in the frequency domain, followed by a 512x512 2-D IFFT to convert the image back to the spatial domain. The execution time, Mflops rating, and the speed-up factor of the Splash-2 implementation over the given architecture is shown in Table 2.

	Execution Time (sec)	**Mflops**	**Speed-up Factor**
Splash-2	**.47**	**180**	**1**
Sparc-2	18	32	38.3
Sparc-10	11	60	23.4
Intel i860	.35	200	.74
TI DSP TMS320C40	1.7	80	3.6
Sharp LH9124 DSP	.08	240	.17

TABLE 2. Comparison of Splash-2 Implementation with Other Architectures.

The performance of the Splash-2 implementation of the FFT was calculated in the following way: The number of clock cycles required to compute the NxN 2-D FFT is $2\ N^2\ \log_2 N$. The application was run at 10 Mhz and therefore the execution time for doing a 512x512, 2-D FFT is $2\ (512)^2\ \log_2(512)\ /\ 10\text{x}10^6$ = .47186 seconds or 2 frames per second. Since the FFT and the IFFT are pipelined and are being computed concurrently, the time for the complete filtering process is the time for calculating one 2-D FFT. The speed of this application was verified by using a logic analyzer to check the time between output frames. In addition, there are 18 floating point units distributed between the FFT, filter and IFFT designs. These units output a result every clock cycle at 10 Mhz therefore, this application operates at 180 Mflops (integer-to-floating point and floating-point-to-integer operations are not included in this figure).

The Cooley-Tukey FFT algorithim[6] was implemented in C and was compiled using the highest optimization level of the *gcc* compiler on a Sparc workstation.

The execution time for the Intel i860 based processing board, Texas Instruments and Sharp DSP chips was calculated by doubling the time required to do a single 512x512 2-D complex FFT and adding a very small amount of time for the filtering process. The time for doing an FFT was doubled in order to account for the time to do and IFFT. The i860 processing board consisted of two, 50 Mhz, i860 chips and 200 Mbytes of RAM. The Sharp DSP chip was chosen since it was the fastest DSP chip surveyed out of almost 60 DSP chips[3]. The Sharp DSP can calculate a complex multiply in one clock cycle at 40 Mhz[11]. The Texas Instruments DSP chip was selected because its performance was about average for the DSP chips in the survey. The algorithm used in the survey to benchmark the DSP chips was a 1024, one dimensional FFT.

It is essential to note that the other implementations used 32-bit single precision floating point arithmetic, and the Splash-2 design takes advantage of the ability to use a smaller 18-bit floating point format. However, this smaller format requires less computation per floating point arithmetic unit than the 32-bit implementation. To implement single precision floating point arithmetic units on the Splash-2 architecture, the size of the floating point arithmetic units would increase between 2 to 4 times over the 18 bit format. A three stage floating point multiply unit would require two Xilinx 4010 chips and a three stage adder/subtracter unit could fit into a single Xilinx chip. The 24-bit multiplier needed in single precession floating point multiply can be broken up into four 12-bit multipliers, allocating two per chip[5]. We found that a 16x16 parallel bit multiplier was the largest parallel integer multiplier that could fit into a Xilinx 4010 chip. When synthesized, this multiplier used 75% of the chip area. However, there was no need to emulate the 32-bit floating-point arithmetic since the desired accuracy was achieved with an 18-bit format. This illustrates an important advantage of FPGA-based computers over traditional approaches.

The Splash-2 performance is more than an order of magnitude better than a general purpose workstation and is similar to an i860 processing board which is faster than many DSP processors. In addition, the Splash-2 implementation is less than six times slower than one of the fastest DSP processors on the market.

7 Conclusions

Due to the flexibility of a CCM, customization of the floating point format was performed to achieve maximum accuracy with the smallest number of bits. By taking advantage of the parallelism of the Splash-2 architecture, address calculation, butterfly operations and filtering could be done concurrently. By pipelining the butterfly operation, a real and complex result was obtained every clock cycle at 10 Mhz. The performance of this application is much faster than a Sparc-10 workstation and is similar to that of a typical DSP processor.

The Splash-2 architecture has been used to improve the performance of a wide range of applications and can be considered as a general purpose custom computing platform. Applications include pattern matching, text searching and genome data base searching, and many different image processing algorithms[2, 8]. The genome base search implementation has shown a speed-up of three orders of magnitude over the MasPar-1. The Splash-2 implementation of a 2-D FFT has shown that the performance is similar to a DSP chip and has shown that floating point arithmetic can be done on CCMs effectively.

References

[1] J. Arnold, D. Buell and E. Davis, "Splash 2," *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, June, 1992, pp. 316-322.

[2] P. Athanas and L. Abbott, "Real-Time Image Processing on a Custom Computing Platform," *IEEE Computer*, Vol. 28, No. 2, February 1995, pp. 16-24.

[3] *Computer Design*, "1995 Product Trends and Resource Guide," February 1995, pp 44-47.

[4] J. Eldon and C. Robertson, *A Floating Point Format for Signal Processing*, *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1982, pp. 717-720.

[5] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," *IEEE Transactions on VLSI*, Vol. 2, No. 3, September 1994, pp. 365-367.

[6] R. Gonzalez and P. Wintz, *Digital Image Processing*, Addison-Wesley Publishing Company, 1977.

[7] IEEE Task P754, "A Proposed Standard for Binary Floating-Point Arithmetic," IEEE Computer, Vol. 14, No. 12, March 1981, pp. 51-62.

[8] D. Hoang, "Searching Genetic Databases on Splash-2", *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 185-191.

[9] R. Lipsett, C. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design*, Kluwer Academic Publishers, Boston, MA., 1989.

[10] L. Rabiner B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, 1975.

[11] Sharp Electronics Corp., "Fast Fourier Transform," Sharp Application Note for the LH9124 DSP Chip, November 1992.

[12] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," To appear at *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1995.

[13] A. Walters, *An Indoor Wireless Communications Channel Model Implementation on a Custom Computing Platform*, VPI&SU Master Thesis in progress.

[14] Xilinx, Inc., *The Programmable Logic Data Book*, San Jose, California, 1995.