Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Efficient FPGA implementation of convolution

Khader Mohammad, Sos Agaian
Electrical and Computer Engineering Department
University of Texas at San Antonio
1 UTSA Circle, San Antonio, TX 78249-0669
Email: hajkhader@gmail.com, sos.agaian@utsa.edu

***Abstract*—This paper presents a direct method of reducing convolution processing time using hardware computing and implementations of discrete linear convolution of two finite length sequences (NXN). This implementation method is realized by simplifying the convolution building blocks. The purpose of this research is to prove the feasibility of an application specific integrated circuit (ASIC) that performs a convolution on an acquired image in real time. The proposed implementation uses a modified hierarchical design approach, which efficiently and accurately speeds up computation; reduces power, hardware resources, and area significantly. The efficiency of the proposed convolution circuit is tested by embedding it in a top level FPGA. Simulation and comparison to different design approaches show that the circuit uses only 5mw that saves almost 35% of area and is four times faster than what is implemented in [5]. In addition, the presented circuit uses less power consumption and has a delay of 20ns from input to output using 32nm process library. It also provides the necessary modularity, expandability, and regularity to form different convolutions for any number of bits.**

Keywords: Convolution, Verilog, implementations, FPGA, Design and Implementation for discrete linear convolution.

I. INTRODUCTION

Many image processing operations such as scaling and rotation require re-sampling or convolution filtering for each pixel in the image [3]. Convolutions on digital images are important since they represent operations that are more general than the operations that can be performed on analog images. Digital images can be modified (through convolution) by neighborhood operations; these operations go beyond point wise operations, and include smoothing, sharpening, and edge detection [2]. Convolution has many applications which have great significance in discrete signal processing. It is usually difficult to deal with analog signals. Hence signals are converted to digital state. Filtering of signals is very important in order to determine which one to accept and which one to reject, and all of that is done by convolution. Some of the major uses of convolution are state Image processing; Wavelets generated by using discrete singular convolution kernels and Fourier transform applications [1].

Many approaches have been attempted to reduce the convolution processing time using hardware and software algorithms. But they are restricted to specific applications [6]. [11] Presented a design for fast convolve for CDMA signals. This is based on avoiding complex operations such as FFT-based convolves. They used substitution of the FFT for a Walsh which reduces the operations three times because it uses only real additions but it requires more hardware like counters, and RAM blocks which increases activity factor. Using image processing functions such as convolution filtering, high performance can be achieved by exploiting parallelism and minimizing hardware cost, but different filter widths and thus potentially different hardware structures are needed for different applications. It is therefore difficult to make a fixed parallel structure efficient. In an application involving spatial scaling of images, for example, a larger filter kernel would be required for large scale factors, a small one for modest scaling. It would be expensive to implement the entire largest desired filter kernel, and wasteful for small scale factors [3].

It is proven that convolution can check all the phase shifts in one step. This is usually done by using the known FFT-based convolution [11]. Each FFT (or IFFT) requires N*log*N complex multiplications and N*log*N complex additions. Therefore, some algorithm require approximately 3N(*log*N)+N complex multiplications and 3N(*log*N)+N additions [2]. Implementing the algorithm in parallel hardware will speed up the process but the implementation itself is very complex and requires a huge silicon area.

The main problem in implementing and computing convolution is speed, area and power which affect any DSP system. Speeding up convolution using a Hardware Description Language for design entry not only increases (improves) the level of abstraction, but also opens new possibilities for using programmable devices. Today, most DSPs suffer from limitations in available address space, or the ability to interface with surrounding systems. The use of high speed FPGAs, together with DSPs, can often increase the system bandwidth, by providing additional functionality to the general purpose DSPs [5].In this paper, a novel method for computing the linear convolution of two finite length sequences is presented. A 4x4 convolution circuit can be instantiated for larger ones. This method is similar to the multiplication of two decimal numbers, this similarity that makes this method easy to learn and quick to compute [1].

This paper is organized as follows. Section II investigates the related convolution algorithm implementation. In section III, circuit implementations are presented. Section IV presents the verification of the proposed design. In section V, evaluation and comparison of the design are presented. Finally, the conclusion is obtained.

II. BACKGROUND AND REALATED WORK

The main assumption of the consistency principle and the mutual correspondence principle between continuous and digital transformations is that the signal is represented discretely through shift sampling and reconstruction. An image

convolution is a filtering step in which an image is the input and a computed image is the output, with each sample of the output image calculated by individually weighting and then constructively and/or destructively summing the samples from some neighborhood of the input image [7]. Analog to digital conversion produces digital signals sampled at a particular Sampling interval, Δt (or Δx). Assuming we have both s(t) and h(t) digital functions with a sampling interval of unity, the convolution operation is defined :

$$y(t) = S_k * h_k \qquad \sum_{k=-\infty}^{+\infty} S_k * h_{j-k}, j = 1,2,3...$$

The summation holds for all products for which the values S_k and h_{j-k} exist. For example, if

$$S_K = S_1, S_2 \ldots\ldots, S_m \quad \text{(m samples)}$$

$$h_k = h_1, h_2 \ldots\ldots, h_n \quad \text{(n samples) where m>n.}$$

The Convolution y_j has $y_j = y_1, y_2, y_3 \ldots, y_{m-n+1}$ (m+n+1 semple). We did implement for the algorithm shown below and mentioned in [1]. We take the two discrete finite length sequences and lines the columns up like regular multiplication but rather than carrying the number over to the next column he writes it down in the same column. For example lets say that we are given two discrete finite length sequences x[n] and h[n] where x[n] = {a1 a2 a3} and h[n] = { b1 b2 b3 b4} are convolved, y[n] =x[n]*h[n], in a way that is similar to regular multiplication as shown below in Tabel 1

Tabel 1

h[n]		*b1*	*b2*	*b3*	*b4*
x[n]			*a1*	*a2*	*a3*
		a3b1	*a3b2*	*a3b3*	*a3b4*
	a2b1	*a2b2*	*a2b3*	*a2b4*	
a1b1	*a1b2*	*a1b3*	*a1b4*		
		a3b1+	*a3b2+*		
	a2b1+	*a2b2+*	*a2b3+*		
a1b1	*a1b2*	*a1b3*	*a1b4*	*3b3+a2b4*	*a3b4*

As we were evaluating possible design approaches to achieve low speed, our research took us through the following progression. Figure 1 shows the convolution flow of two 16-bit numbers, in 4-bit segments. The letters A, B, C, D, E, F, G, and H each represent 4 bits of the 16 bits number. We sum the partial product along each column; HD0 is the LS 4 bits of the product while HD1 is the MS 4 bits of the product.

The Digital Convolution is summarized as: first Flip (reverse) one of the digital functions, second Shift it along the time axis by one sample. Third, multiply the corresponding values of the two digital functions. Fourth, sum the products from step 3 to get one point of the Digital Convolution. And finally repeat steps 1-4 to obtain the digital convolution at all times that the functions overlap. For example, let

X= [1 2 3 4 5] and v = [-1 5 3 -2 1].

				A	B	C	D
							Conv
				E	F	G	H
				0	0	HD1	HD0
			0	0	HC1	HC0	
			0	HB1	HB0		
			HA1	HA0			
			0	0	GD1	GD0	
		0	0	GC1	GC0		
		0	GB1	GB0			
		GA1	GA0				
		0	0	FD1	FD0		
	0	0	FC1	FC0			
	0	FB1	FB0				
	FA1	FA0					
	0	0	ED1	ED0			
0	0	EC1	EC0				
0	EB1	EB0					
EA1	EA0						ADD

Convolution Results

Figure 1

A discrete convolution of these two discrete signals equals:

-1 3 10 15 21 33 10 -6 5

We used Matlap to check the results which is shown in figure 2. For continuous function, y(t) = x(t)*h(t) where the input, x(t), and the impulse response, h(t) has a sufficiently small delta to make the result to be accurate. The e results are shown in figure 3.

*x= [-2*ones(1,400) zeros(1,1000) 3*ones(1,100)]*
h=ones(1,300);
conv(x,-3,h,-2,0.01)

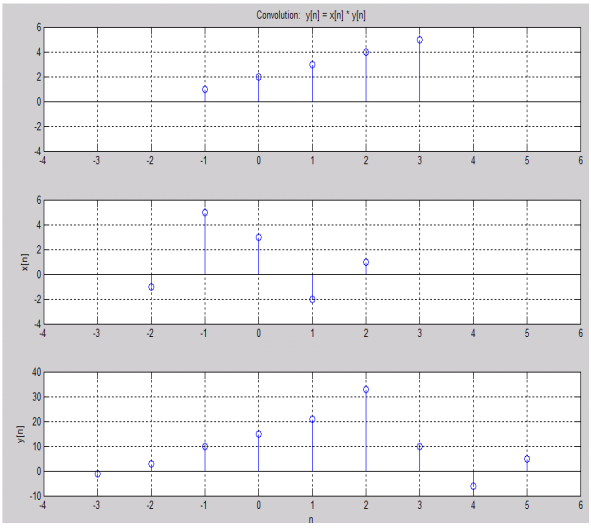

Figure2

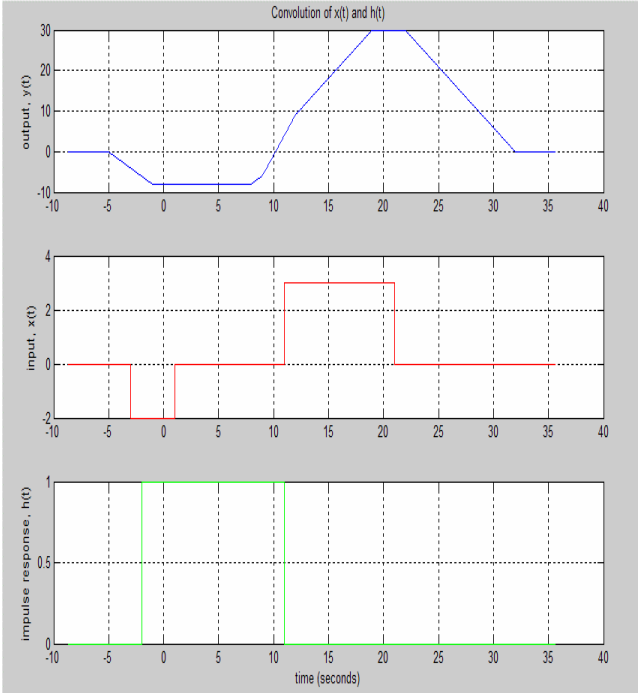

Figure 3

High performance Digital Signal Processing chips have been widely employed to solve signal processing problems. Many of these signal processing solutions can be implemented in a Field Programmable Gate Array (FPGA) instead of a DSP chip. This is possible because the gate densities available in FPGAs have increased rapidly within the last few years and now allow fairly sophisticated DSP algorithms to be implemented within a single chip [5]. In [5] they try to implement the convolution in an FPGA. Their approach in calculating a finite number of L convolution samples requires approximately 3L+L(L+1)/2 clock cycles and addresses for the two data memories which cost lots of access time resources. In their design they extend the result of the multiplication by six more overflow bits before the results are added to the previous sum of products. This is done so they can prevent overflow which is costly.

Depending on the application and desired quality (i.e. the width of the filter kernel), computing this weighted sum of neighboring pixels can require significant amounts of computation, thus suggesting a highly parallel implementation in special-purpose hardware. In [3] they discuss parameterized program generation of convolution filters in an FPGA for applications in image processing including real-time video and desktop publishing. They show an example of 2-D filter pipeline assembled from a set of multipliers and adders, which are in turn generated from a canonical serial-parallel multiplier stage. They show a 3x3 convolution filter for video applications. The drawback In their research is they have a high fan-in and because of the pipeline delay, output pixels may be rewritten directly into the source image memory [3].

It is important to point out the emerging field of algorithm derivation and implementation, which could be used as a basis for future work. In [4] it is shown there are no restrictions imposed on the convolution length other than to be composite, but they pointed out FPGA implementation will be a future work. Breitzman [15] shows the automatic derivation and implementation of fast convolution algorithms and Arce-Nazario [16] presents an automated methodology designed for the high-level partitioning of discrete signal transforms onto distributed hardware architectures [4]

To efficiently control the number of required multipliers, at the cost of a reasonable number of adders, a study was done on a hardware efficient fast cyclic convolution algorithm [9]. It shows the I/O cost can be kept low and the throughput rate high. Thus, it is much more efficient than previous cyclic convolution implementation methods. But independently applying this algorithm for prime-length DFT will still require huge amount of hardware cost [6]. The DFT designs in [7], [8] remove the multiplication operations, but they require a large number of adders and RAM/ROM resources.

Another approach people use is to go through Matlab. It is used to automatically generate Verilog code for the hardware implementation of convolution algorithms. This automation is very efficient when the coefficients change. As mentioned in [10] when they are trying to implement FIR filter, some inputs go through two consecutive subtraction operators. This optimization can be done when the Verilog code is being automatically generated. In their implementations they used carry-save adders to accumulate consecutive adders which are slow compared to using other adders as will be discussed in the next section. Note that the number of required additions is dependent on the order of iterations. The iteration order for short convolutions should be 4x4, 3x3 and 2x2, as this will lead to the lowest implementation cost [10].

The research paper in [11] shows a substitute algorithm for calculating the convolution that requires less computation time. It is shown that CDMA receivers require a long time to acquire the signals. This is mostly due to the use of expensive FFT-based convolvers in the acquisition process. The permutations usually can be stored in lookup tables .This type of implementation is not efficient since it will cost additional hardware to store and time to retrieve.

III. PROPOSED IMPLEMNTATION CIRCUIT

NXN was selected, and the implementation for 4x4 was prepared in order to have short convolutions that will lead to the lowest implementation cost, as mentioned in [10]. The circuit deals with two signals having N values each. We selected N=4 in our implementations. We consider the two numbers like two arrays having four locations each to store values. Each array is fed into a quadruple 4X1 Mux separately. Hence we can have each signal value up to 4 bit. The selection of values is done by selection switches of each Mux. The selected values go into the Array Multiplier and from there they are routed into Parallel Load Registers through a 1X16 Demux. Afterwards the stored values are added to get the convolved Result [4]. The block diagram of the circuit is shown in figure 4.

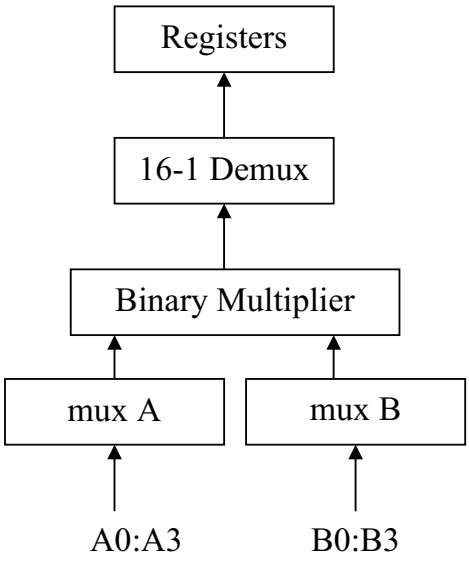

Figure 4

The basic concept of convolution is to flip, multiply and add. Now for two signals of four values each, we have to flip (invert one of the signals) multiply and then add the values. The flipping of the values is done by selection of the 4X1 Multiplexer. Figure 4 shows the basic building blocks used in the design. The design is built in Verilog and implemented on an FPGA. We parameterized the inputs to N, so we can setup the values to whatever number we need. Further, a 16X1 De-multiplexer is used to store the multiplied values in different registers. The values of the first and seventh registers are first and seventh output values respectively. The other values are obtained by adding the corresponding values. For addition an 8-bit Full Adder was made by instantiating eight 1-bit Full Adders. Figure 5 show sub blocks used inside the design.

For adding three and four values additional circuitry was made by using Full Adders and Half adders. Simple registers were replaced with parallel load registers. First, all the loads were enabled. After the use of each register its load was disabled, so that the value remains saved. The traditional multiplication is done using the Array Multiplier. A 4 bit Array Multiplier was used to get an 8-bit output. This kind of multiplier is selected based on performance after comparison of different multiplier design as shown in table 2.

Table 2 Simulation Power, area, Components for 4 bit different multiplier

	Power		Number of component	Area mm^2	Delay (ns)	Power-delay product
	VDD=5V	VDD=3.3V				
Array Multiplier	4.447	1.85	490	0.533	14.4	64.03
Booth multiplier	11.42	4.72	1278	0.134	16.8	191.8

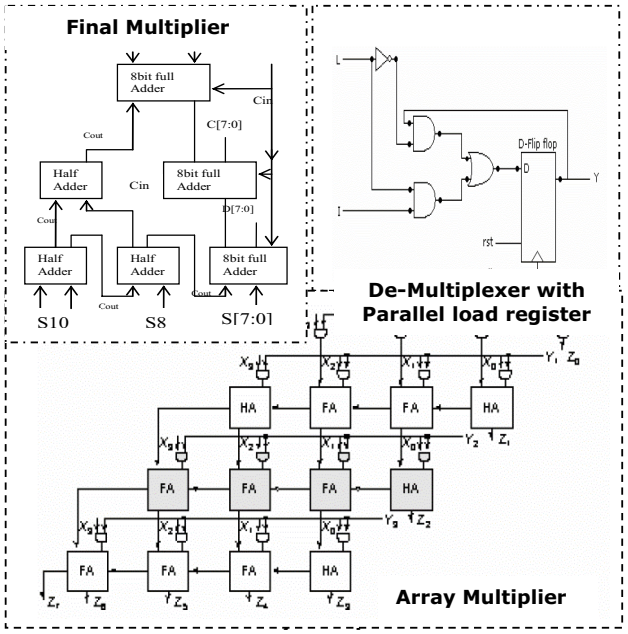

Figure 5 sub-blocks

IV. DESIGN VERIVICATION

Verification is completed using the Modelsim simulator. The IO blocks and data format conversion were designed first and tested in the FPGA. The functionality of some of the blocks was verified by simulations before being tested in Hardware. We used these two numbers:

A = 15 15 15 15 B = 15 15 15 15

The output of the Modelsim simulation to verify functionality is shown in figure 6 and figure 7 although, we have 2 numbers with 4 decimal points the output will be a number with 8 decimal points. The output ranges are from 0 to 7. The top level schematic is shown in figure 8

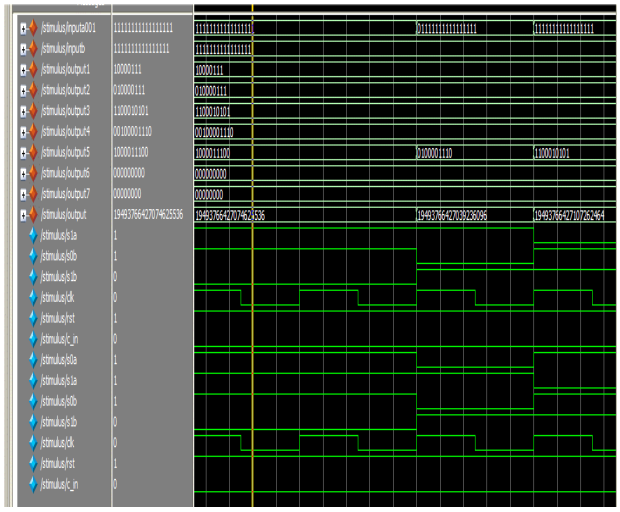

Figure 1. Figure 6 Top level models verification.

Input		
A0 | A1 | A2 | A3	1111|1111|1111|1111	
B0 | B1 | B2 | B3	1111|1111|1111|1111	
Output		
First value	**Fourth value**	**Sixth value**
1|11100001	1|1010100011	1|1010100011
0|11100001	0|1010100011	0|1010100011
second value	1|00111000010	1|111000010
1|11100001	0|00111000010	0|111000010
0|11100001	1|01010100011	1|111000010
1|111000010	0|01010100011	0|111000010
0|111000010	1|01110000100	**Seventh value**
1|111000010	0|01110000100	1|111000010
0|111000010	1|01110000100	0|111000010
Third value	0|01110000100	1|11100001
1|111000010	**Fifth value**	0|11100001
0|111000010	1|01110000100	
1|0111000010	0|01110000100	
0|0111000010	1|0111000010	
1|1010100011	0|0111000010	
0|1010100011	1|1010100011	
1|1010100011	0|1010100011	
0|1010100011	1|1010100011	
	0|1010100011	

Figure 7 Modelsim for each point

V. EVALUATION OF THE PROPOSED DESIGN

Several design challenges have been brought up for implementation. The new implementation has been implemented with a unique multiplier. Power and area savings have been achieved by aggressively minimizing the number of gates. This architecture can get the output in 3 clock cycles but that will be based on how many bits you have. For example, for 32 bit if you use a regular multiplier you will need to shift for 32 cycles to get the output. Area-wise it is smaller compared to any other multiplier because we can reuse the same 4 bits. In general, as the bit width increases, the amount of pipelining producing the lowest energy delay also increases. Table 3 and 4 shows Area, number of cells and flops.

Table 5 shows the difference between our FPGA results and what has been implemented in [5]. Figure 8 shows the power consumption chart based on the cell placement. The maximum power consumption is about 5mw. The resulting delay of the synthesized design is 10ns in a 32nm process. This shows better delay results than the one used in the FPGA because of process difference and constraints that we used.

Table 5

	data implementations[5]	Proposed	% improvement
I/Os	192	124	54.8
Logic cells	480	353	35.9
Delay (ns)	65	19.298	236.8
Time to out		7 secs	

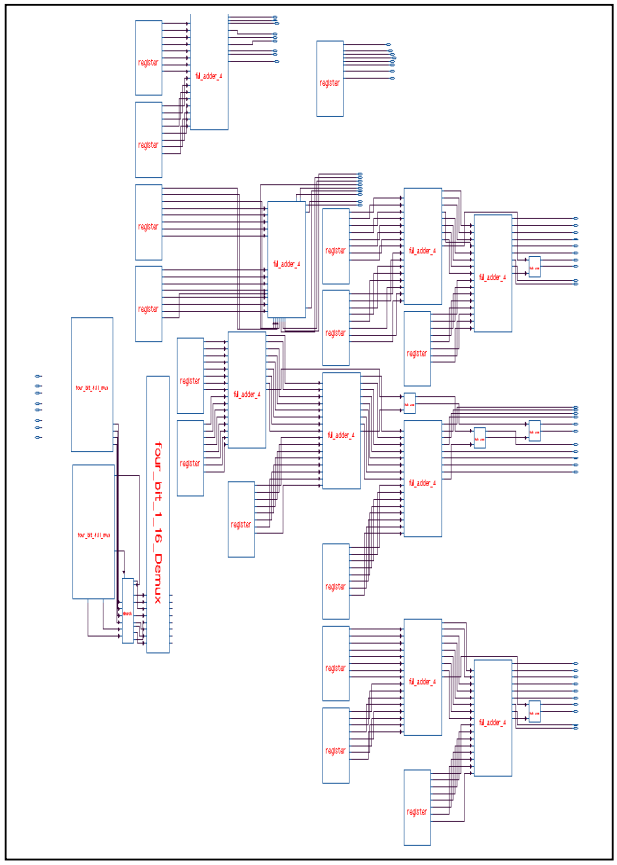

Figure 2 Top level

Table 3 DC area results

Number of ports:	124
Number of nets	916
Number of cells	856
Combinational area	839.2um^2
Non combinational area	620.5 um^2
Net Interconnect area	5.1 um^2

Table 4 FPGA results

Function	Number
logic cells	353
LUT2	10
LUT3	174
LUT4	160
MUXF5	8
FlipFlops/Latches	128
Clock Buffers	1
BUFGP	1
IO Buffers	123
INBUF	58
IOBUF	65

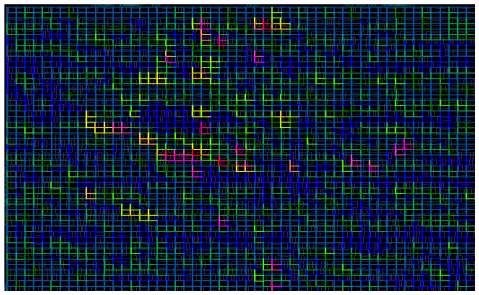

Figure 8

VI. CONCLUSION

In this paper, we presented an optimized implementation of discrete linear convolution. This particular model has the advantage of being fine tuned for signal processing; in this case it uses the mean squared error measurement and objective measures of enhancement to achieve a more effective signal processing model. This implementation has the advantage of being optimized based on operation, power and area. To accurately analyze our proposed system, we have coded our design using the Verilog hardware description language and have synthesized it for FPGA products using ISE, Modelsim and DC compiler for other processor usage. Second, we implemented an illustrative example 4X4 convolver. Similarly, the presented concept can be extended on an NXN case. The functionality of the convolver was tested and verified successfully on a XILINIX SE FPGA and design compiler. The proposed circuit uses only 5mw and saves almost 35% area and it takes 20ns to complete. This shows improvement of more than 50% less power. As FPGA technology matures and much larger arrays become practical, techniques that allow the automatic generation of highly-parallel architectures will become central to high performance computing. We have described some simple techniques for generation of convolution pipelines for image processing and other applications. Higher level techniques and approaches are also needed. FPGAs permit restructurable processing, and restructurable interconnects are also becoming available.

REFERENCES

[1] John W. Pierre, "A Novel Method for Calculating the Convolution Sum of Two Finite Length Sequences", IEEE transaction on education, VOL. 39, NO. 1, 1996.

[2] W. W. Smith, J. M. Smith, "Handbook f Real-Time Fast Fourier Transforms", IEEE Press, 1995, p. 28.

[3] R. G. Shoup, "Parameterized convolution filtering in a field programmable gate array," in selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs. Oxford, United Kingdom: Abingdon EE&CS Books, 1994, pp. 274–280.

[4] Iván Rodríguez, "Parallel Cyclic Convolution Based on Recursive Formulations of Block Pseudocirculant MatricesMarvi Teixeira", IEEE, transaction on signal processing,2008

[5] Thomas Oelsner ,"Implementation of Data Convolution Algorithms in FPGAs" , QuickLogic Europe http://www.quicklogic.com/images/appnote18.pdf

[6] Chao Cheng , Keshab K. Parhi ,"Low-Cost Fast VLSI Algorithm for Discrete Fourier Transform", IEEE,. IEEE transaction on circuits and systems, VOL. 54, 2007

[7] J. I. Guo, C. M. Liu, and C. W. Jen, "The efficient memory-based VLSI array designs for DFT and DCT," IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 37, no. 10, 1992, pp. 723–733.

[8] T. S. Chang, J. I. Guo, and C. W. Jen, "Hardware-efficient DFT designs with cyclic convolution and subexpression sharing",IEEE Trans. Circuits Syst. II, Analog Digital Signal Process., vol. 47, no. 9, 2000, pp. 886–892.

[9] C. Cheng and K. K. Parhi, "Hardware efficient fast DCT based on novel cyclic convolution structures", IEEE Trans. Signal Process., vol. 54, no.11, 2007, pp. 4419–4434.

[10] Chao Cheng , Keshab K. Parhi "Hardware Efficient Fast Parallel FIR Filter Structures Based on Iterated Short Convolution" IEEE, and, IEEE transaction on circuits and systems, VOL. 51, NO. 8, 2004 http://www.tc.umn.edu/~chen0867/ParallelFIR2004_TCASI.pdf.

[11] Abdulqadir Alaqeeli, Janusz Starzyk, "Hardware Implementation for Fast Convolution with a PN Code Using Field Programmable Gate", Ohio University, http://www.ent.ohiou.edu/~starzyk/network/Research/Papers/Recent%20conferences/Conv_FPGA_PN_code_SSST2001.pdf.