We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

172,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Fully Systolic FFT Architectures for Giga-sample Applications

D. Reisis

*Department of Physics, Electronics Laboratory,
National Kapodistrian University of Athens,
Greece*

1. Introduction

This chapter presents a technique for designing architectures, which execute Fast Fourier Transform (FFT) algorithms involving a large number of complex points and they target real-time applications (Ersoy, 1997). Hitherto published techniques and FFT architectures include mainly ASIC designs (Thompson, 1983; Wold and Despain, 1984; He and Torkelson, 1996; Choi et al., 2003; Uzun et al., 2005; Bouguezal et al., 2004, 2006; Jo & Sunwoo, 2005; Chang & Nguyen, 2006; Yang et al., 2006; Lin et al., 2005; Takala & Punkka, 2006; Wang & Li, 2007; Reisis & Vlassopoulos, 2006), which vary with respect to the level of parallelism, the throughput rate, the latency, the hardware cost and the power consumption. The most common ASIC architectures are the fully unfolded FFT realizations (Rabiner & Gold) utilizing large memory arrays between their successive stages and also the latency and memory efficient cascade FFT topologies (Thomson, 1983; He & Torkelson, 1996, 1998). The cascade solutions though, as well as the high radix techniques lead to complicated designs in the case of architectures parameterized with respect to the pipelining of computations, the FFT size and the data length.

This chapter describes a technique to design efficient, very high speed, deeply-pipelined FFT architectures maximizing throughput and keeping control and memory organizations simple compared to the cascade and the fully unfolded FFT architectures. Moreover, the design is proven more efficient comparing to the previously mentioned architectures in terms of scalability, maximum operating frequency and consequently, in terms of power consumption, pipeline depth and data and/or twiddle bit-widths. The technique improves the latency and the memory requirements -particularly for large input data sets- of systolic FFT architectures by combining three (3) Radix-4 circuits to result in a 64-point FFT engine.

The efficiency of organizing 64-point FFT engines based on Radix-4 FFT engines is shown by a 4096-complex point design. This architecture requires only two dual memory banks of 4096 words each and on a Xilinx Virtex II FPGA performs at 200 MHz to sustain a throughput of 4096 points/20.48 us. The design implemented on a high performance 0.13 um, 1P8M CMOS (standard cell) process from UMC achieved a worst-case (0.9V, 125 C) post-route frequency of 604.5 MHz, while consuming 4.4 Watts. It is interesting to point out

that the design exceeded the 1 GHz frequency (rate of 1 GSample/sec) for typical conditions (1.0V, 25C).

Towards designing architectures FFTs with large input data sets we consider the 4096 complex point FFT architecture as a core. The core constitutes the basis of FFT architectures computing transforms of 16K, 64K and 256K complex points. These architectures implemented in the 0.13 CMOS process perform at 352, 256 and 188 Mhz worst-case (0.9V, 125 C) post-route frequencies respectively. The 16K and the 64K architectures have a four point parallel input/output achieving throughputs of 1.4 and 1 GSample/sec respectively. Further, this chapter will present a technique, which allows the parallelization of the memory accesses in hardware implementations of the FFT algorithm. This technique enables each processor to perform a radix-b butterfly by loading the b-tuple data from b memory banks in parallel, then by operating on the b data and finally, by storing the resulting b-tuple in b memory banks in parallel. Hence, the speedup and the throughput increase by b. Techniques parallelizing the FFT accesses are reported in (Johnson, 1992; Ma, 1999; Reisis & Vlassopoulos, 2006). We describe the technique in (Reisis & Vlassopoulos, 2006), which is developed for arbitrary radix and straightforward to implement.

The chapter is organized in three technical sections. Section 2 shows how to organize radix-4 computations to result in a radix-4³-equivalent to radix-64-computation and describes the fully systolic 4096-point architecture. Section 3 will describe the 16K, 64K and 256K point architectures. Section 4 presents the access parallelization technique in the FFT architectures and section 5 concludes the chapter.

2. A Fully Systolic 4096 complex point FFT Architecture

This section presents an attractive solution combining the advantages of both the cascaded (Thompson, 1983; He & Torkelson, 1998; OH & Lim, 2005) and the unfolded (Rabiner & Gold; Thompson, 1983) structures. The organization uses the unfolded structure with radix-64 (R64) stages. We note that 64 is a power of either 2 or 4 and it is sufficiently large to result in small number of stages. Realizing the 64 point FFT at high speed with radices greater than 2 and 4 becomes inefficient, because the use of higher radices imposes complicated structures and leads to lower the architecture's performance. R4 computations are equally straightforward to implement as the R2, but R4 designs reduce the number of complex multipliers to the half of those required by the R2 designs. Also, R4 process 4 points in a 2-step computation *accumulating-multiplying* with the *multiplying* reduced to *sign-inversion* and *swap* operations. This result minimizes the number of possible bottlenecks in the data flow and allows the maximal pipelining and/or parallelization of the four computing steps. Consequently, the choice of R4 leads to a more efficient design with respect to the multipliers, the pipeline depth and the scalability of the operating frequency and/or of the word length.

2.1 Radix-4³ Algorithm

To produce the R4³ structure we start with the Discrete Fourier Transform (DFT) of a signal $x[n]$ of length N,

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad (1)$$

where, $W_N = e^{-j\frac{2\pi}{N}}$ are the twiddle factors and denote the N-th primitive root of unity (Oppenheim, 1975). The architecture presented in this section is based on a four-dimensional index map and on a R4 decomposition of the DFT series. The derivation of the Radix - 4³ algorithm lies on three (3) steps in the cascade decomposition. In the frame work of these 3 steps, the linear mapping transforms into a four-dimentional index map [10,4] as follows:

$$\begin{aligned} n &= n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 + \frac{N}{4}n_4 \\ k &= 64k_1 + 16k_2 + 4k_3 + k_4 \end{aligned} \quad (2)$$

Applying equation 2 to the DFT equation yields

$$X(64k_1 + 16k_2 + 4k_3 + k_4) = \sum_{n_1=0}^{\frac{N}{64}-1} \sum_{n_2=0}^3 \sum_{n_3=0}^3 \sum_{n_4=0}^3 x(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 + \frac{N}{4}n_4) W_N^{nk} \quad (3)$$

The composite twiddle factors can be expressed as follows

$$\begin{aligned} W_N^{kn} &= W_N^{[n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 + \frac{N}{4}n_4][64k_1 + 16k_2 + 4k_3 + k_4]} \Rightarrow \\ W_N^{kn} &= (-j)^{n_2k_2 + n_3k_3 + n_4k_4} W_{16}^{n_3k_4} W_{64}^{n_2(4k_3 + k_4)} \times W_N^{n_1(16k_2 + 4k_3 + k_4)} W_{\frac{N}{64}}^{n_1k_1} \end{aligned} \quad (4)$$

Applying equation 4 into equation 3 and expanding the summation with index n_4 yield

$$\begin{aligned} X(64k_1 + 16k_2 + 4k_3 + k_4) &= \\ &\sum_{n_1=0}^{\frac{N}{64}-1} \sum_{n_2=0}^3 \sum_{n_3=0}^3 \left[B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 \right) \right] \times \\ &\times (-j)^{n_2k_2 + n_3k_3} W_{16}^{n_3k_4} W_{64}^{n_2(4k_3 + k_4)} \times W_N^{n_1(16k_2 + 4k_3 + k_4)} W_{\frac{N}{64}}^{n_1k_1} \end{aligned} \quad (5)$$

Where $B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 \right)$ denotes the first butterfly unit and can be written as

$$\begin{aligned} B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 \right) &= x \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 \right) + (-j)^{k_4} x \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 + \frac{N}{4} \right) \\ &+ (-1)^{k_4} x \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 + \frac{N}{2} \right) + j^{k_4} x \left(n_1 + \frac{N}{64}n_2 + \frac{N}{16}n_3 + \frac{3N}{4} \right) \end{aligned} \quad (6)$$

Expanding equation 5 with respect to the next summation with index n_3 yields

$$X(64k_1 + 16k_2 + 4k_3 + k_4) = \sum_{n_1=0}^{\frac{N}{64}-1} \sum_{n_2=0}^3 H_{\frac{N}{16}}^{k_3 k_4} \left(n_1 + \frac{N}{64} n_2 \right) (-j)^{n_2 k_2} \times W_{64}^{n_2 (4k_3 + k_4)} W_N^{n_1 (16k_2 + 4k_3 + k_4)} W_{\frac{N}{64}}^{n_1 k_1} \quad (7)$$

where $H_{\frac{N}{16}}^{k_3 k_4} \left(n_1 + \frac{N}{64} n_2 \right)$ is the secondary butterfly structure and can be expressed as

$$H_{\frac{N}{16}}^{k_3 k_4} \left(n_1 + \frac{N}{64} n_2 \right) = B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64} n_2 \right) + (-j)^{k_3} W_{16}^{k_4} B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64} n_2 + \frac{N}{16} \right) + (-1)^{k_3} W_8^{k_4} B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64} n_2 + \frac{N}{8} \right) + j^{k_3} W_8^{k_4} W_{16}^{k_4} B_{\frac{N}{4}}^{k_4} \left(n_1 + \frac{N}{64} n_2 + \frac{3N}{16} \right) \quad (8)$$

Finally, expanding the summation of equation 7 with regard to index n_2 provides a set of 64 DFTs of length $N/64$.

$$X(64k_1 + 16k_2 + 4k_3 + k_4) = \sum_{n_1=0}^{(N/64)-1} \left[T_{N/64}^{k_2 k_3 k_4} (n_1) W_N^{n_1 (16k_2 + 4k_3 + k_4)} \right] W_{N/64}^{n_1 k_1} \quad (9)$$

where $T_{N/64}^{k_2 k_3 k_4} (n_1)$ represents the third butterfly and is expressed according to equation 10

$$T_{\frac{N}{64}}^{k_2 k_3 k_4} (n_1) = H_{\frac{N}{16}}^{k_3 k_4} (n_1) + (-j)^{k_2} W_{16}^{k_3} W_{64}^{k_4} H_{\frac{N}{16}}^{k_3 k_4} \left(n_1 + \frac{N}{64} \right) + (-1)^{k_2} W_8^{k_3} W_{32}^{k_4} H_{\frac{N}{16}}^{k_3 k_4} \left(n_1 + \frac{N}{32} \right) + j^{k_2} W_{64}^{3(4k_3 + k_4)} H_{\frac{N}{16}}^{k_3 k_4} \left(n_1 + \frac{3N}{64} \right) \quad (10)$$

Equations 5 to 10 describe a radix-64 based FFT. Further, equations 6, 8 and 10 describe the internal structure of the radix-64 butterfly based on three radix-4 butterflies and therefore called Radix - 4^3 ($R4^3$).

2.2 $R4^3$ Features

Figure 1 shows the flow diagram of the $R4^3$ computations where Δ , $\circ$ and $\triangleright$ present multiplications by $(-j)^{k_3} W_{16}^{k_3}$, $(-1)^{k_3} W_8^{k_4}$ and $j^{k_3} W_8^{k_4} W_{16}^{k_4}$ respectively, while the $\diamond$, ∇ , $\triangleleft$ present multiplications by $(-j)^{k_3} W_{16}^{k_3} W_{64}^{k_4}$, $(-1)^{k_2} W_8^{k_3} W_{32}^{k_4}$ and $j^{k_2} W_{64}^{3(4k_3 + k_4)}$. Note that the order of the twiddle factors in the $R4^3$ differs from the respective order of the $R64$.

The FFT architecture with $R4^3$ stages and input N complex points uses $\log_4 N - 1$ complex multipliers. With each $R4$ using 3 complex adders to produce 1 result/cycle the architecture has $3\log_4 N$ complex adders. The memory size is $(N/3)\log_4 N$. Comparing to the cascaded $R2^2$ (He & Torkelson, 1998) and $R2^4$ (OH & Lim, 2005) the proposed unfolded $R4^3$ has equal number of multipliers (Table 1). The $R4^3$ has less complex adders ($3\log_4 N$) than the cascaded ($4\log_4 N$) but uses larger memory size. The $R4^3$ FFT design achieves the least number of multipliers and adders in the literature, equal to that in (Bidet et al., 1995), and although it requires larger memory, it uses simple control and it can achieve high frequency performance as it is shown in the following subsection.

2.3 Architecture

Figure 2 depicts the architecture of the 4096-point FFT. The implementation of the 4096-point FFT using $R4^3$ butterflies includes: first the use of a 2-dimensional index map based on $R64$, second the decomposition of the 4096-point series into two sums using the $R64$

	Complex Multipliers	Complex Adders	Memory	Control
R2MDC *	$2\log_4 N - 1$	$4\log_4 N$	$3N/2 - 2$	Simple
R2SDF *	$2\log_4 N - 1$	$4\log_4 N$	$N - 1$	Simple
R4SDF *	$\log_4 N - 1$	$8\log_4 N$	$N - 1$	Medium
R4MDC *	$3(\log_4 N - 1)$	$8\log_4 N$	$5N/2 - 4$	Simple
R4SDC *	$\log_4 N - 1$	$3\log_4 N$	$2N - 2$	Complex
R2 ² SDF **	$\log_4 N - 1$	$4\log_4 N$	$N - 1$	Simple
$R4^3$	$\log_4 N - 1$	$3\log_4 N$	$(N/3)\log_4 N$	Simple

*(He & Torkelson, 1998), **(OH & Lim, 2005)

Table 1. FFT architecture hardware complexity comparison

butterfly and finally the replacement of each $R64$ with a $R4^3$. Consequently, the architecture consists of two $R4^3$ engines, two 4096-word dual bank memory modules, one 4096 point read-only memory storing the W_{4096} twiddles and one complex multiplier. The control is local to each module and a global sync signal synchronizes the modules. The throughput is 1 complex-point/clock-cycle.

The overall design allows the optimization of the architecture, either globally, or locally: The architecture can be improved with respect to the operating frequency, or the area, or both without alterations in the control, the scheduling of the 4K buffers, the registers and the memories within the $R4^3$ engines. Hence, we can modify the pipelined computations and exploit the specifics of each technology to maximize the operating frequency.

The following subsections describe the details of the $R4$ processor used as the basis of the $R4^3$, the $R4^3$ architecture, the addressing scheme used for each $R4^3$ engine and discuss the performance of the entire FFT architecture.

2.3.1 The Radix-4 basis of the $R4^3$

Figure 3 depicts the Radix-4 engine architecture used in the $R4^3$. Each Radix-4 engine consists of one complex Accumulator and one "Swap" module. The task of the "Swap" module is to switch the real part of each input value with the corresponding imaginary and

the opposite in order to perform the correct butterfly operations. Figure 4 depicts the architecture of the Accumulator module. The accumulator consists of 8 registers and 3 add/sub units. The four "A" registers are used as input registers storing every 4-tuple of data, on which the R4 butterfly operation will be applied. The four "B" registers are used as an intermediate stage holding the 4-tuple (while the next 4-tuple of input data is shifted into the "A" registers) and loading the data in parallel to the add/sub units. The add-sub units form an adder-tree architecture, thus avoiding the feedback loops that common accumulators use.

Fig. 1. Radix – 4^3 Butterfly SFG

The data flow of the Radix-4 engine starts with the data entering the Radix-4 engine as a word serial input stream at a rate of one complex-point/cycle. During each period of 4 cycles four consecutive input data are shifted in the 4 input registers (Reg A). In the fifth cycle this 4-tuple is latched in the 4 "storage" registers (Reg B), while the next input 4-tuple is starting its input to the "A" registers. During the sixth cycle the 4-tuple enters in parallel the adder-tree to produce the first of the four R4 results. During the following 3 cycles, the

adder-tree uses as input the 4-tuple stored in the "B" registers and it produces the remaining 3 of the four R4 results, one result/cycle, by following the Radix-4 computational flow. Therefore, the total latency of each accumulator is $5+2k$ cycles, where k is the latency of each add/sub unit, $k = 5$ in the implementation.

A control unit synchronizes the operations of the 'Swap' module and the accumulator. A 2-bit counter generates the necessary signals, controlling the add/sub units in order to perform the correct additions and/or subtractions according to the Radix-4 schema.

Fig. 2. Block diagram of the proposed 4K FFT Architecture

Fig. 3. Overall Engine of the Radix - 4

2.3.2 The $R4^3$ Architecture

Figure 5 shows the $R4^3$ engine consisting of 3 R4 butterflies, 2 complex multipliers, 2 Dual Bank memory elements (1 consisting of 2×64 words and 1 of 2×16 words) and 2 Read Only Memories storing the W_{64} and W_{16} twiddles. The complex twiddle multipliers are 4 integer-multipliers of input 14×14 and are pipelined with depth 5. The control of each pipeline-stage is local and all stages are synchronized through a global sync signal. Realizing the R64 stage by the $R4^3$ engine offers a combination of advantages: the simplicity of the circuits as a result of using as basic butterfly small radix (4) and the regularity of the architecture, which is built hierarchically with contiguous $R4^3$ engines at the higher level and with each $R4^3$ formed by three R4 pipelined computations at the basic level. Furthermore, the $R4^3$ design is VSLI area efficient because it utilizes the complex multipliers 75% of the cycles and 100% of the other resources at each cycle. The FFT achieves the dynamic range of 84 dB (amplitude) by using the width of 14 bits for each real and imaginary part of each sample. To maintain the maximum precision (14 bits) for each part, the $R4^3$ uses increased precision for internal

calculations. The bit width of each part is 16 and 20 bits at the output of the first and second R4 stages respectively. At the output of the final (third) R4 stage, the real and imaginary parts are truncated to provide an output of 14 bits each. The twiddle factors are 18 bits real

Fig. 4. The Adder Tree of the Complex Accumulator

Fig. 5. Block diagram of the Radix - 4^3 Butterfly Architecture

and 18 bits imaginary. By applying the above mentioned data format, the implementation of the 4096 point FFT with fixed point calculations is almost as accurate as a floating point implementation given the dynamic range of 84 dB. The fixed point FFT's outputs are within a margin of ± 1 compared to the output of a floating point FFT whose values are normalized (divided by $N = 4096$) and truncated to 14 bit integers.

2.3.3 Data Scheduling and Addressing in R4³

The first stage of the 4K point FFT algorithm is realized by the first R4³ processor, which computes $\frac{4096}{64}$ FFTs with each transform consisting of 64 points. Each transformation 64-tuple contains the elements whose address is of the form $64i + k$, where $i = 0, \dots, 63$ and k is the tuple index, ranging from 0 to $\frac{4096}{64} - 1$.

In the first R4³ processor the data within each 64-tuple are processed in terms of quadruples, so that the first R4 processor computes the DFT of the elements with address $16i_1 + k_1$, $i_1 = 0, \dots, 3$ and $k_1 = 0, \dots, \frac{64}{4} - 1$ is again the tuple index. We can use the notation $[a_{11}, \dots, a_0]$ for each point's input address (position within the 4096 points) and $[d_5d_4d_3d_2d_1d_0]$ for the address of each bank of the first intermediate memory. The resulting 64 elements are written in one bank of the first intermediate memory in consecutive addresses, i.e. $[a_{11}a_{10}a_9a_8a_7a_6] \rightarrow [d_5d_4d_3d_2d_1d_0]$.

The second R4 reads the data from the first intermediate bank of size 64 by using the address permutation $[a_1a_0a_3a_2a_5a_4] \rightarrow [d_5d_4d_3d_2d_1d_0]$. Every 4 consecutive results of 4-point DFTs produced by the second R4 are stored in one of the two banks of the second intermediate memories of size 16. When the third R4 starts reading these data the second R4 will store the next 4 DFT results into the second intermediate memory of size 16.

Note that the second R4 produces results so that the third R4 can use all the data stored in the intermediate memory to complete its 4-point DFT fully pipelined and with 100% resource utilization. With $[d_3d_2d_1d_0]$ denoting the address within the second intermediate memory, the third R4 reads the data with address permutation $[a_1a_0a_3a_2] \rightarrow [d_3d_2d_1d_0]$.

During the second stage of the 4K FFT algorithm, the second R4³ processor computes $\frac{4096}{64}$ FFTs with each transform consisting of 64 points, so that each transformation 64-tuple contains the elements whose address is of the form $64k + i$, where $i = 0, \dots, 63$ and k is the tuple index, ranging from 0 to $\frac{4096}{64} - 1$. The first R4 produces 64 elements, which are written in one bank of the first intermediate memory in consecutive addresses, i.e. $[a_5a_4a_3a_2a_1a_0] \rightarrow [d_5d_4d_3d_2d_1d_0]$. Apart from this difference the addressing and the data scheduling within the second R4³ is exactly the same as in the first R4³ described above.

2.3.4 FPGA and VLSI Implementation

The proposed architecture has been implemented in RTL VHDL using fixed point arithmetic. The project aims at both high capacity and high speed FPGA devices such as the XILINX Virtex II 6000, as well as a high performance 0.13um standard cell process. For the Xilinx implementation we have used the Xilinx tool and the automatic floor planning option. The Xilinx implementation on the 6000 part results in 20% of logic area utilization and 25% Blocks RAM utilization. The use of optimized Xilinx components (CoreLib multipliers) reduces the area to 13% of the total resources of the 6000 part.

The TSMC 0.13 library implementation achieved 604 MHz worst case. It includes 96K standard cells and 64 RAMs (1361 standard cell rows), ($\times 64$ configuration in Table 2) occupying a silicon area of 2630×5129 um square (1.42×10^7 um sq) at 84.2% utilization and achieves a worst-case (0.9V, 125C) post-route performance of 604.5 MHz and a 4.4 Watts

power consumption. The use of typical process parameters (1V, 25C) results in exceeding the 1GHz post-route frequency mark (data rate 1 GSample/sec), making the proposed architecture the fastest standard-cell 4096 complex point FFT implementation reported in the literature. In addition, a second 4094 complex point engine has been implemented in the same standard cell library, this time using deeper RAMs ($\times 16$ configuration in Table 2). Power consumption in this case was substantially reduced to 722.8 mW from the 4.4 W of the $\times 64$ configuration. Figure 6 depicts the VLSI layout of the Radix – 4^3 engine, while figure 7 depicts the final layouts of the 4K FFT, of both the $\times 16$ and the $\times 64$ implemented.

The 4K FFT has been designed for an experiment involving a frequency analyzer for a bandwidth of 200 MHz. The band has been divided into four sub-bands and each sub-band has been accommodated by a 4K FFT architecture. The FPGAs perform at 102.5 MHz on a 18-layer board which has a compact-PCI interface performing at 51.25 MHz. The task is to perform FFT and use a "Threshold" filter to identify the frequencies of high power within each sub-band. The expected output set includes at most 10 frequencies per sub-band, per FFT. After the prototype completion the 4K FFT has been delivered as an IP core with specifications achieving 5 times the performance of the FPGA prototype. The 16K, 64K and 256K have been realized as IP cores for research purposes.

2.4 Architecture's Performance and Advantages

The proposed architecture demonstrates improved latency compared to other unfolded architectures because it requires data buffering only between the two $R4^3$ stages instead of the buffering required at all the 6 stages of the unfolded FFT using $R4$. The existing cascade FFT architectures require 6 $R2^2$ (He & Torkelson, 1996, 1998) and although they use 1/4 of the memory of the proposed $R4^3$ architecture for 4096 points, they are not scalable and they must be designed for specific performance due to their organization involving "closed computing loops".

The achieved performance establishes the proposed architecture as a real-time high performance FFT realization, the importance of which can be further pointed out by

Fig. 6. VLSI implementation layout of the Radix – 4^3 engine

comparing its characteristics to other relevant results. Tables 3 and 4 present a comparison of the proposed FFT architecture with related results. We distinguish related published results into two categories.

	FFT 4096 Ramsx16	FFT 4096 Ramsx64
Clock (period)	1.6 ns	1.2 ns
Fmax in MHz	386.8	604.5
Std Cells (RAMs)	100347(16)	95897(64)
Std Cells (rows)	805	1361
Chip Size (um sq)	5.34E+06	1.42E+07
Util (%)	80.30%	84.20%
X(um)	1681	2630
Y(um)	3165	5129
Power (mW)	722.8	4414.1

Table 2. 4K FFT VLSI Implementation Routing Results

In the first category we compare the hitherto published architectures executing FFT algorithms up to 128 points to the features of the R-4³ playing the role of a complete 64 complex point FFT architecture. This comparison is shown in Table 3. The second category includes the architectures solving FFTs of size 1024 to 4096 points presented in Table 4. The comparison includes FFT size, word length, algorithm, FFT architecture, technology process, voltage, area, power, maximum operating frequency and sustained throughput in both MSamples/s and Gbits/s. Since the FFT designs vary with respect to FFT size, algorithm and architecture we have also included the Normalized Area (Bidet et al., 1995), in order to evaluate the silicon cost. Moreover, we compare the efficiency (performance/cost) of the proposed design to the related results by using the fraction Sustained Throughput/Normalized Area.

The proposed FFT architecture achieves the highest sustained throughput compared to all the other designs. Furthermore, the efficiency expressed as the fraction Sustained Throughput/Normalized Area of the proposed design is the highest considering both small input size (Table 3) and large input size FFT architectures (Table 4). Note that, in both categories the architectures R - 4³ and the proposed 4K occupy more area than their competitors respectively. This is a penalty though in achieving the highest throughput possible.

Also note that (Lin et al., 2005) performs transformations of only 128 points, and there is no provision taken so that it will constitute a core for scalable architectures with respect to FFT size. The 64-point Fourier transform chip, presented in (Maharatna et al, 2004) operates at 20 MHz with latency 3.85 us, comparing to the R4³ processor performing a 64 complex point FFT while operating at a 200 MHz clock frequency with latency 0.32 us.

The architecture described in (Lenart & Owall, 2003) is a 2K complex point FFT processor which achieves maximum operating frequency of 76MHz and sustains a throughput of 2048 points/26us. The design presented in (Cortes et al., 2006) implements a 2K/4K/8K multimode FFT and achieves 9 MHz clock frequency, at a computation time of up to 450us.

Fig. 7. VLSI implementation layout of the 4K FFT processor: a) 16 × 4096 Configuration, b) 64 × 4096 Configuration

Characteristics	(Lin et al., 2005)	(Maharatna et al., 2004)	Proposed Design (R-4 ³)
FFT size	128p	64p	64p
Word Length, bit	10	16	14
Algorithm	R-2, R-8	R-2	R-4 ³
FFT Architecture	MRMDF	Unrolled	Unrolled
Process (um)	0.18	0.25	0.13
Voltage (V)	3.3	1.8	0.9
Area (mm ²)	3.52	6.8	5.09
Normalized Area (mm ²)	3.52	3.5	9.7
Power (mW)	77.6	41	78.2
Fmax (MHz)	110	20	604.5
Sustained Throughput in MSamples/s	480	16.6	1052
Sustained Throughput in Gb/s	9.6	0.53	29.5
Throughput(Gb/s)	2.72	0.15	3.04
Norm.Area			

Table 3. Comparison Table with FFT designs from 64-128 points

Characteristics	(Lenart & Owall, 2003)	(Cortes et al., 2006)	(Swartzlander, 2007)	Proposed Design
FFT size	2K	2K/4K/8K	4K	4K
Word Length, bit	10	16	16	14
Algorithm	$R - 2^2$	$R - 2^2$	$R - 2$	$R - 4^3$
FFT Architecture	SDF	SDF	Split Systolic	Unrolled
Process (um)	0.35	0.35	0.25	0.13
Voltage (V)	N/A	3.3	N/A	0.9
Area (mm ²)	6	18.7	N/A	13.48
Normalized Area (mm ²)	1.58	4.9	N/A	25.84
Power (mW)	N/A	114.65	260	4414.1
Fmax (MHz)	76	9.1	100	604.5
Sustained Throughput in MSamples/s	75.8	18.2	200	1052
Sustained Throughput in Gb/s	1.5	0.582	6.4	29.5
Throughput(Gb/s) Norm.Area	0.94	0.11	N/A	1.14

Table 4. Comparison table with FFT designs from 1K-4K

Finally, a single ASIC chip, systolic FFT processor, developed by the Mayo Foundation computes 4096-point FFTs sustaining a throughput of 200 Ms/s (Swartzlander, 2007). Considering FPGA implementations, the corresponding XILINX designs (www.xilinx.com) achieve equal maximum operating frequency of 200MHz, but occupy considerably larger chip area than the $R4^3$ approach. Also note that, ALTERA designs (www.altera.com) utilize FFT cores with FFT length varying from 64 points up to 4K points. They demonstrate a maximum operating frequency of 300 MHz. Among the ALTERA’s FFT designs we compare the 64 point FFT at 300 MHz to the $R4^3$ performance, which realized on the same ALTERA FPGA (ALTERA STRATIX II EP2S30F484C3) achieves operating frequency of 350 MHz.

3. 16K, 64K and 256K complex point FFT Architectures

The high utility prospects of the efficient 4K point FFT design as presented above becomes tangible through the utilization of this architecture to develop larger size FFT architectures to compute 16K, 64K and 256K point transforms. This section describes the exploitation of the 4K point FFT as a “radix-4096” processor resulting in architectures meeting different requirements with respect to parallelism, silicon area and throughput. These architectures can be tailored to a broad area of applications.

3.1 Deriving the 16K point Architecture

The DFT equation for a 16K point FFT takes the form

$$X[k] = \sum_{n=0}^{16383} x[n] W_{16384}^{nk} = \sum_{n=0}^{4 \cdot 4096 - 1} x[n] W_{4 \cdot 4096}^{nk} \quad (11)$$

Setting $n = n_1 + \frac{N}{4096} n_2$ and $k = k_1 4096 + k_2$:

$$k \cdot n = \left(n_1 + \frac{N}{4096} n_2 \right) (4096 k_1 + k_2) \Rightarrow k \cdot n = n_1 k_1 + n_1 k_2 + N n_2 k_1 + \frac{N}{4096} k_2 \Rightarrow$$

$$W_{4 \cdot 4096}^{kn} = W_4^{n_1 k_1} W_{4 \cdot 4096}^{n_1 k_2} W_{4096}^{n_2 k_2}$$

Therefore, the transform becomes

$$X[k] = \sum_{n_1=0}^3 W_4^{n_1 k_1} \left(\sum_{n_2=0}^{4095} x[n] W_{4096}^{n_2 k_2} \right) W_{4 \cdot 4096}^{n_1 k_2} \quad (12)$$

According to equation 12, the 4K points FFT can be extended to 16K points. This is accomplished by first performing four 4K point FFT transforms. Next, the data is multiplied by the twiddle factors that correspond to a 16K points FFT and finally, a radix-4 stage completes the 16K point FFT computation.

The architecture is presented in figure 8. There are four 4K FFT blocks operating in parallel. The architecture has a four complex point input per cycle. The 16K FFT architecture was implemented in a high performance, 0.13um, 1Poly-8Copper layer standard cell technology from TSMC. A flat back-end flow was used in which the design was first synthesized to gates using Synopsys Design Compiler and optimized for a frequency of 300 MHz. The uniquified netlists was then read into Cadence SoC Encounter where floorplanning, power planning, clock-tree-synthesis, placement, routing and IPO routing took place. Finally, the design was brought back into the top-level for final top-level placement, routing and timing analysis. Table 5 shows the Implementation Routing Results and figure 8 depicts the VLSI Cells for the 16K FFT. The throughput is 1.4 Gs/sec (39.2 Gbits/sec).

Fig. 8. Block diagram of the 16K FFT - Parallel/Parallel Architecture

Parameter	FFT 16K
Core Size	1.6561e + 07um ²
Std Cell Rows	1102
Number of Cells	885456
Number of RAMs	64
Statistical Power	3.45 W
Fmax	352 MHz

Table 5. VLSI Implementation Routing Results of the 16K FFT architecture

The 16K architecture implemented on the Xilinx Virtex 5 (-2) achieves operating frequency of 250MHz, occupies 12264 slices and sustains a throughput of 1Gs/s (28 Gbits/sec).

3.2 64K and 256K complex points FFT Architectures

The DFT equation for a 64K point FFT takes the form

$$X[k] = \sum_{n=0}^{65535} x[n] W_{65536}^{nk} = \sum_{n=0}^{4 \cdot 16384 - 1} x[n] W_{4 \cdot 16384}^{nk}$$

(13)

Setting $n = n_1 + \frac{N}{16384} n_2$ and $k = k_1 16384 + k_2$:

$$k \cdot n = 16384 n_1 k_1 + n_1 k_2 + N n_2 k_1 + \frac{N}{16384} k_2 \Rightarrow W_{4 \cdot 16384}^{kn} = W_4^{n_1 k_1} W_{4 \cdot 16384}^{n_1 k_2} W_{16384}^{n_2 k_2}$$

Therefore, the transform becomes

$$X[k] = \sum_{n_1=0}^3 W_4^{n_1 k_1} \left(\sum_{n_2=0}^{16383} x[n] W_{16384}^{n_2 k_2} \right) W_{4 \cdot 16384}^{n_1 k_2}$$

(14)

According to equation 14, the 16K points FFT can be extended to 64K points. This is accomplished by first performing a 16K point FFT transform. Next, the data is multiplied by the twiddle factors that correspond to a 64K points FFT and finally, a R4 stage completes the 64K point FFT computation, as shown in figure 10. The 64K FFT has been VLSI (and FPGA) implemented by using the 16K parallel/parallel computation with a R4 stage with four parallel inputs and outputs. The architecture has a post routing frequency of 256 Mhz with a throughput of 1 Gs/sec (28 Gbits/sec). Figure 11 depicts the VLSI Cell for the 64K FFT design. The 64K has been implemented on the Xilinx Virtex 5 (-2) achieving an operating frequency of 125MHz and using 13461 slices. The architecture has a four data parallel input and output and sustains a throughput of 500Ms/s (14 Gbits/sec).

3.3 256K complex point FFT Architecture

Figure 12 depicts the 256K FFT architecture, which is a straightforward application of the $R4^3$ algorithm. Three consecutive $R4^3$ engines are used to accomplish the task.

Fig. 9. VLSI implementation layout of the 16K FFT Parallel/Parallel architecture

Fig. 10. Block diagram of the 64K complex point FFT Architecture

Parameter	FFT 64K	FFT 256K
Core Size	3.4894e + 08um ²	2.7647e + 08um ²
Std Cell Rows	1600	4500
Number of Cells	896148	735945
Number of RAMs	192	384
Statistical Power	9.8 W	35.75 W
Fmax	256.5 MHz	188 MHz

Table 6. VLSI Implementation Routing Results of the 64K and 256K FFT designs

Fig. 11. VLSI implementation layout of the 64K FFT

In the case of the very large 256K FFT, tool capacity mandated the use of a hierarchical flow. Following the same front-end synthesis process (again optimized for 300 MHz), the optimized netlist was read into Cadence SoC encounter where partitioning was first performed. This process created six instances of the 256K memory block (for a total of 1.5 MB of on-chip SRAM) which were individually placed and routed. The same process was performed for the R4³ engines and the twiddle ROM block. Note that to complete the entire FFT computation it requires a 3-frame latency (786792 cycles) and the computation latency within the three R4³ (360 cycles), which is 4.1 msec. The 256K FFT architecture has a post routing frequency of 188 Mhz. Finally, the design was brought back into the top-level for

final top-level placement, routing and timing analysis. Figure 13 depicts the VLSI Cell for the 256K FFT design and Table 6 shows the VLSI Implementation Routing Results of the 64K and 256K FFT designs.

Fig. 12. Block diagram of the 256K FFT Architecture

4. Parallel Accessing in FFT Architectures

This section presents a technique, which allows the parallelization of the memory accesses in hardware implementations of the FFT algorithm by enabling a processor during a FFT stage to access in parallel b memory banks without conflicts. Consequently the processor performs a radix- b butterfly by loading the b -tuple data from b memory banks in parallel, then by operating on the b data and finally by storing the resulting b -tuple in b memory banks in parallel. Hence, the speedup and the throughput increase by b .

4.1 Problem Definition and Related Work

Let $N = b^p$ denote the number of points of the Fast Fourier Transform, where b is an arbitrary base such that $b \geq 2$ and $p \geq 1$. Using b as a base, the input address $0, \dots, N-1$ (or index) of each element is represented as $[a_{p-1} \dots a_0]$, where $0 \leq a_i \leq b-1$. We will denote each element only by its address $[a_{p-1} \dots a_0]$, which represents the element in the input stream with index $a_{p-1}b^{p-1} + \dots + a_1b + a_0$. This notation holds for each of the p stages of the FFT.

We will use $P_{b[n \rightarrow m]}$ to denote a processor performing all radix- b computations, which can also read (load) n data in parallel and write (store) m data in parallel. We assume a generic architecture using ψ , $1 \leq \psi \leq \log_b N$, $P_{b[b \rightarrow b]}$ processors and $2 \cdot b \cdot (\psi + 1)$ memory banks, so that each bank can store N/b elements. Each processor realizes N/b consecutive stages of the FFT computation.

Fig. 13. VLSI implementation layout of the 256K FFT

The memory banks are arranged according to the scheme described in figure 14. Since, we are using a radix- b processor, the DFT decomposition consists of $\log_b N$ stages. If we assume $\psi = \log_b N$ then we consider a $P_{b[b \rightarrow b]}$ processor realizing each stage of the algorithm. The stages are indexed from 0 to $\log_b N - 1$. Between two (2) consecutive stages i and $i+1$ there is a memory of size $2 \cdot N$ divided into two (2) sets of size N each. We will denote the memory between stage i and stage $i+1$ as memory with index i . The memories among the FFT stages are indexed as 0 to $\log_b N - 2$. There are memories indexed as -1 and $\log_b N - 1$ serving as input and output memories respectively. The memory of each set i is divided into b memory banks with indices 0 to $b-1$.

At each clock cycle, the $P_{b[b \rightarrow b]}$ processor of stage i loads from its "read" memory (memory $i-1$), b elements that form a transform b -tuple. These are the elements whose address is of the form $[a_{p-1}, \dots, a_{i+1} | a_{i-1}, \dots, a_0]$, where the indices $[a_0, \dots, a_{i-1}, a_{i+1}, \dots, a_{p-1}]$ are the same for all elements in the b -tuple and l ranges from 0 to $b-1$. These elements will be loaded in parallel if they are stored in distinct memory banks. During the next $((i + 1)^{\text{th}})$ step of the algorithm, these b elements have the same a_{i+1} address digit and hence they belong to

different transformation b -tuples. These elements must be stored in the processor's write memory so that they can be read in parallel from the processor at stage $i + 1$.

Fig.14. Generic arrangement of a $P_{b[b \rightarrow b]}$ processor and its write memories

The straightforward approach is to distribute the N elements to the b memory banks using their i^{th} address digit. Then the processor at stage i can load them in parallel. However, if we try to store the same b elements of each b -tuple to the memory bank at stage $(i+1)$ according to their $(i+1)^{\text{th}}$ digit of their address, we will notice that all the elements in each b -tuple must be stored in the same memory bank. Since we cannot store more than one element in a memory bank at each clock cycle this situation constitutes a “conflict”.

To illustrate the situation where a “conflict” occurs, consider a 16 point transform that is based on $P_{4[4 \rightarrow 4]}$ processors. The address of each element in the input stream is $[a_1 a_0]$, where $a_i = 0, 1, 2, 3$. Figure 15 shows the first stage of the example. The first 4-tuple to be transformed consists of the elements $[(0, 0) (0, 1) (0, 2) (0, 3)]$. Note that, these elements can be read in parallel from the set of memory banks of stage $i - 1$, but must be stored in the same memory bank of memory i .

The first technique for parallelizing the memory accesses in FFT architectures (Johnson, 1992) describes a hardware architecture designed for in-place computation of the FFT algorithm. This technique uses r (r being the radix) banks and permutes the output data of each processor, to be written in the same memory locations within each bank as the input data have been read. (Ma, 1999) describes a technique for radix-2 based FFT by using queues at the input of each memory to rearrange the data before they are stored. Johnson's technique can be extended to the more general case of a pipelined FFT implementation at the cost of complex addressing hardware implementation. (Thomas & Yelick, 1999) present a technique that is used in vector processors. This technique permutes the input data prior to the radix calculations in order to maximize the efficiency of the algorithm. These permutations are performed within the vector registers (In-register transpose). This functionality has been implemented by extending the instruction set of the vector processor with two instructions that shift the data within a register, so that each permutation is performed by using a register to register copy and a shift instruction.

All the FFT techniques (Johnson, 1992; Ma, 1999), with b parallel memory accesses, involve at each stage b banks for reading and b banks for writing the N FFT data with bank size N/b . The in-place case uses the same b banks (each bank of size N/b) for both read and write. To compare the hardware complexity of the previously published results, we first consider designs that are based on queues, such as the one described in (Ma, 1999).

Fig. 15. First stage of a 16-point transform using $P_{4[4 \rightarrow 4]}$ processors

Such designs require $O(b^2)2w$ -bits wide registers, where w is the word length, arranged in b queues with each queue having $b - 1$ registers. b is the processor radix, or in the general case, the number of parallel read/write accesses. Each set of $b-1$ registers is arranged so that $b-1$ elements are loaded in parallel while one element is written into the memory bank. The other $b-1$ elements are shifted -one element per memory access cycle- into each memory bank. The technique in (Johnson, 1992) uses fixed hardware to implement first a permutation on the input data and then this specific hardware can apply a permutation at each FFT stage. More specifically, (Johnson, 1992)'s approach gives only a subset of the possible solutions. This subset of solutions considers only in-place implementations ($\psi = 1$ and memory size N). The circuit in (Johnson, 1992) is optimized for implementing these solutions with respect to the VLSI area. The address generation is based on a circuit involving a tree of modulo- b adders with $O(b^2)$ exclusive-OR gates.

In this section we describe a technique (Reisis & Vlassopoulos, 2006) that can be used both for pipelined architectures and in place implementations (memory size equals to N). We follow a different approach than (Johnson,1992) leading to a different proof and providing a

general solution, which includes that in (Johnson,1992). Moreover, the presented technique results in a simple and improved hardware implementation compared to (Johnson,1992). The presented solution shows improved performance with respect to the latency and the hardware cost compared to other solutions (Ma, 1999; Suter & Stevens,1998; He & Torkelson; Thomas & Yelick, 1999) while it provides the same throughput as these. Our technique is based on memory address permutations and can be realized using look up tables with each stage table occupying $O(b^2)$ bits. The presented approach results in a set of permutations, which can be applied in the design of pipeline FFT architectures with parallel memory accesses per stage. A subset of these permutations accommodates the in-place implementations.

4.2 Parallelizing the Memory Accesses

The following Lemmas 1 and 2 prove the existence of the required permutations and Theorem 1 proves that these permutations result in a correct FFT algorithm.

Lemma 1: Assume that the $N = b^p$ elements with indices $[a_{p-1} \dots a_0]$, $0 \leq a_i \leq b-1$, are distributed in b sets S_l , $l = 0, \dots, b-1$, such that: $S_l = \{[a_{p-1} \dots a_{i+1} l a_{i-1} \dots a_0]\}$

Let $f_j : \{0, \dots, b-1\} \rightarrow \{0, \dots, b-1\}$, where $0 \leq j \leq b-1$, be a set of functions such that

- $\exists f_j^{-1} \Rightarrow f$ is 1-1 and onto, and
- $\forall m, \forall n$, where $m, n \in \{0, \dots, b-1\}$,

If $m \neq n$, then $\forall x, f_m(x) \neq f_n(x)$

The sets $S'_{l'}$, where $l' = f_{a_{i+1}}(l)$ are defined as follows $S'_{l'} = \{[a_{p-1} \dots a_{i+2} f_{a_{i+1}}(l) l a_{i-1} \dots a_0]\}$

Where l is fixed and $a_{i+1} = 0, \dots, b-1$

Under the above conditions, the following hold:

- 1) Each $S'_{l'}$ contains at most N/b items
- 2) Every two elements of two sets S_{l_1}, S_{l_2} whose indices differ only in the i^{th} digit, i.e. q and r will be distributed in different S' sets, namely the $S'_{q'}$ and $S'_{r'}$.

Proof of Lemma 1: Each set $S'_{l'}$ contains the data with indices $[a_{p-1} \dots a_{i+2} f_{a_{i+1}}(l) l a_{i-1} \dots a_0]$, where $a_{i+1} = 0, \dots, b-1$ and l is fixed. To prove (1) assume that there exists a set that contains more than N/b elements. Then, there is at least one element with index $[a_{p-1} \dots a_{i+2} f_{a_{i+1}}(l') l'' a_{i-1} \dots a_0]$ where l' is different than l'' . Since such an element does not belong to the set $S'_{l'}$, we conclude that each set cannot contain more than N/b elements.

Similarly, to prove (2), assume that two elements of the sets S_q and S_r , respectively, whose indices differ in the i^{th} digit are distributed in the same set, $S'_{s'}$. Then, for these elements,

$$[a_{p-1} \dots a_{i+2} f_{a_{i+1}}(q) l a_{i-1} \dots a_0] = [a_{p-1} \dots a_{i+2} f_{a_{i+1}}(r) l a_{i-1} \dots a_0] \Rightarrow f_{a_{i+1}}(q) = f_{a_{i+1}}(r).$$

Since the f_j are invertible, $f_{a_{i+1}}^{-1}(f_{a_{i+1}}(q)) = f_{a_{i+1}}^{-1}(f_{a_{i+1}}(r)) \Rightarrow q = r$, which contradicts our initial assumption that $l_1 \neq l_2$. The following Lemma shows that these functions exist.

Lemma 2: There exists a set of functions $f_j : \{0, \dots, b-1\} \rightarrow \{0, \dots, b-1\}$, $j = 0, \dots, b-1$ and $b \geq 2$ such that the conditions of Lemma 1 are satisfied.

Proof of Lemma 2: Let $M = \{0, \dots, b-1\}$. Further, let $C(M)$ denote the set of cyclic permutations of length b on the set M . These permutations are of the form

$$\begin{pmatrix} 0 & 1 & \dots & (b-1) \\ (p) \bmod b & (1+p) \bmod b & \dots & (b-1+p) \bmod b \end{pmatrix}$$

for $p = 0, \dots, b-1$. Now, these permutations are invertible and there are exactly b such permutations. To prove that

$$f_i(x) \neq f_j(x) \quad (15)$$

for $i \neq j$ and $i, j = 0, \dots, b-1$ it suffices to show that $(x+i) \bmod b \neq (x+j) \bmod b$, which is true since i and j could only be in the same equivalence class, if $i = k \cdot j$, where k is an integer, but this can be true only for $k = 1$, since $i, j \leq b-1$. Therefore equation 15 holds.

Theorem 1: Let $N = b^P$. Assume that we have a $P_{b[b \rightarrow b]}$ processor. Then, a radix- b based FFT having $\log_b N$ stages can use b memory banks at each stage, such that all the read and write operations are performed in parallel.

Proof: Let $[a_{p-1} \dots a_0]$, $0 \leq a_i \leq b-1$, be the address of each element according to its initial position within the input data set of the algorithm. The i^{th} stage of the algorithm has arranged the N elements in b memory banks according to their i^{th} digit of their address:

$$[a_{p-1} \dots a_{i+1} a_{i-1} \dots a_0][a_i] \quad (16)$$

where a_i is the index of the memory bank.

To write the outputs of a radix- b processor into b distinct memory banks in parallel, we use the results of Lemma 1. First, we assign each memory bank to a set, S_i , so that each transform b -tuple consists of elements that differ in the i^{th} element of their address. Next, we select a set of b functions $f_0, f_1, \dots, f_{(b-1)}$ using Lemma 2.

Since the requirements of Lemma 1 hold, we can distribute the elements according to the equation:

$$[a_{p-1} \dots a_{i+2} a_i \dots a_0][f_{a_{i+1}}(a_i)] \quad (17)$$

These elements will be distributed in distinct memory banks and can be stored in parallel. Executing the following stage, we will use the inverse permutation to read each b -tuple in parallel: During the $(i+1)^{\text{th}}$ stage of the FFT algorithm, the $(i+1)^{\text{th}}$ processor should read the data whose address is

$$[a_{p-1} \dots a_{i+2} a_i \dots a_0][a_{i+1}] \quad (18)$$

where $a_{i+1} = 0, \dots, b-1$ and $a_j = \text{const}$, for $j \neq i+1$. The inverse permutation is given by

$$[a_{p-1} \dots a_{i+2} a_i \dots a_0][f_{a_{i+1}}^{-1}(a_i)] \quad (19)$$

where $a_i = \text{const}$ and $a_{i+1} = 0, \dots, b-1$, since this permutation yields a b -tuple whose elements differ only in the $(i+1)^{\text{th}}$ digit. To prove that these elements are stored in b distinct memory banks assume that this is not the case, i.e. there exist two distinct elements, $q_1 = [a_{p-1} \dots a_{i+2} a_i \dots a_0][f_{a_{i+1}}^{-1}(a_i)]$ and $q_2 = [a'_{p-1} \dots a'_{i+2} a'_i \dots a'_0][f_{a'_{i+1}}^{-1}(a'_i)]$ that are

in the same b -tuple and at the same time are stored in the same memory bank. Since q_1 and q_2 are on the same b -tuple, therefore $a_j = a'_j$ for $j \neq i+1$. Further, $f_{a'_{i+1}}^{-1}(a_i) = f_{a_{i+1}}^{-1}(a_i)$, which is true only when $a'_{i+1} = a_{i+1}$. Therefore, the elements q_1 and q_2 coincide.

To conclude the proof we show that the forward and inverse permutations at all stages result in a correct FFT algorithm. We proceed by induction on the number of stages. During the first stage of the FFT the data are written in the input memory banks (memory -1) according to $[a_{p-1} \dots a_1][a_0]$. The outputs of the butterfly are written to the first intermediate memory (memory 0) using the permutation $[a_{p-1} \dots a_2 a_0][f_{a_1}(a_0)]$, $a_0 = 0, \dots, b-1$. We assume that the functions f_j are identical for all stages, although this may not be the case. The only actual restriction is to use a permutation P_i for writing the data at stage i and its inverse permutation P_i^{-1} for retrieving the data at stage $i+1$. Applying the inverse permutation we obtain $[a_{p-1} \dots a_2 a_0][f_{a_1}^{-1}(a_0)]$, with $a_0 = \text{constant}$ and $a_1 = 0, \dots, b-1$. The set $\{f_{a_1}^{-1}(a_0)\}$ consists of b distinct elements, such that their address differs only in the digit a_1 and therefore, they constitute a valid transform b -tuple. Further, assume that in the i^{th} stage the elements are stored according to equation 17. Using the same reasoning as above, we can see that the inverse transform yields a valid b -tuple and the proof is complete. Note that the write operation of the elements in the first set of banks (memory 0) does not require any permutation on the input data set. The elements are written to the bank corresponding to their 0^{th} (zero) address digit, in radix b notation. Similarly, during the final step of the algorithm, the data can be written in the same addresses and banks as those that have been read from, since no computation follows.

The current section has shown the technique and the properties of the permutations required to the parallel access of each b -tuple at each FFT stage. An engineer can realize the technique by choosing among the straightforward solutions, e.g. the cyclic permutations. A ninteresting research topic is to identify those permutations, which can be realized with minimal interconnection and address generation circuits and thus lower the VLSI cost.

5. Conclusion

The present chapter has shown a technique to design FFT architectures for real-time applications, which involve input with large number of complex points. The technique bases on combining three consecutive R4 stages to realize a R64 computation. The resulting R4³ as well as the systolic architectures, which utilize a R4³ as a stage for executing 4K, 16K, 64K and 256K point FFTs, have been shown to provide higher throughput compared to hitherto published architectures solving the corresponding transformations. Moreover, the R4³ and 4K FFT architectures achieve the highest ratio of throughput to normalized area. Furthermore, the chapter has proven a technique for parallelizing the memory access for each butterfly radix- b computation so that the throughput can be improved further by a factor of b .

6. References

- A. Cortes, I. Velez, I. Zalbide, A. Irizar, J.F. Sevillano "An FFT Core for DVB-T/DVB-H Receivers ICECS'06, page(s):102-105, December 2006.
- A. Oppenheim, R. Schafer Digital Signal Processing. Prentice Hall, 1975.
- B. G. Jo and M. H. Sunwoo, "New Continuous-Flow Mixed-Radix (CFMR) FFT Processor Using Novel In-Place Strategy," IEEE Trans. Circuits Syst. I, vol.52, no.5, May 2005.
- B. Suter and K. S. Stevens "A Low Power, High Performance approach for Time-Frequency / Time-Scale Computations," Proceedings SPIE98 Conference on Advanced Signal Processing Algorithms, Architectures and Implementations VIII. Vol. 3461, pp. 86-90, July 1998.
- C. D. Thompson, "Fourier Transforms in VLSI", IEEE Transactions on Computers, Vol. 32, 1047 - 1057, 1983
- D. Harper, "Block , Multistride Vector, and FFT Accesses in Parallel Memory Systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 2, No. 1, pp. 43 - 51, January 1991
- D. Reisis, N.Vlassopoulos, "Address Generation Techniques for Conflict Free Parallel Memory Accessing in FFT Architectures" ICECS, pp.1188-1191, December 2006.
- E. Bidet, D. Castelain, C. Joanblanq and P. Stenn "A fast single-chip implementation of 8192 complex point FFT" IEEE Journ. of SSC, 30(3):300-305, Mar. 1995.
- E. H. Wold and A. M. Despain "Pipeline and Parallel FFT Processors for VLSI Implementations," IEEE Transactions on Computers, vol. C-33, 1984.
- Earl E. Swartzlander, Jr "Systolic FFT Processors: A Personal Perspective Journal of VLSI Signal Processing, June 2007.
- I. S. Uzun, A. Amira and A. Bouridane, "FPGA implementations of Fast Fourier Transforms for real-time signal and image processing," IEEE Vision, Image and Signal Processing, 2005.
- J. Lee, J. Lee, M. H. Sunwoo, S. Moh and S. Oh "A DSP Architecture for High-Speed FFT in OFDM Systems," ETRI Journal, 2002.
- J. Takala and K. Punkka Scalable FFT Processors and Pipelined Butterfly Units Journal of VLSI Signal Processing 43, 113-123, 2006.
- J. Y. OH and M. S. Lim, "New Radix-2 to the 4th Power Pipeline FFT Processor," IEICE Trans. Electron., VOL. E88-C, NO. 8, August 2005.
- J.W. Cooley and J.W. Tukey, "An algorithm for the machine computation of complex Fourier series", Mathematics of Computation, 1965
- K. Babionitakis, K. Manolopoulos, K. Nakos, N. Vlassopoulos, D. Reisis, V. Chouliaras, "A High Performance VLSI FFT Architecture", 13th IEEE International Conference on Electronics, Circuits and Systems, pp. 810-813, December 2006
- K. Maharatna, E. Grass, and U. Jagdhold, "A 64-Point Fourier Transform Chip for High-Speed Wireless LAN Applications Using OFDM," IEEE Journal of Solid State Circuits, VOL. 39, NO. 3, March 2004.
- K. Manolopoulos, K. Nakos, D. Reisis, N.Vlassopoulos, V.A. Chouliaras "High Performance 16K, 64K, 256K complex points VLSI Systolic FFT Architecture" ICECS, pp. 146-149, December 2007.
- L. G. Johnson, "Conflict Free Memory Addressing for Dedicated FFT Hardware", IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, Vol. 39, No. 5, May 1992

- L. R. Rabiner and B. Gold "Theory and Application of Digital Signal Processing," Prentice-Hall.
- L. Yang, K. Zhang, H. Liu, J. Huang and S. Huang "An Efficient Locally Pipelined FFT Processor," IEEE Trans. Circuits Syst. II, vol. 53, no. 7, July 2006.
- N. Hu, O. Ersoy, "Fast Computation of Real Discrete Fourier Transform for Any Number of Data Points", IEEE Transactions on Circuits and Systems, Vol. 38, No. 11, pp. 1280 - 1292, November 1991
- O.K . Ersoy, Fourier-Related Transforms, Fast Algorithms and Applications. Englewood Cliffs, NJ:Prentice Hall, 1997.
- R. Thomas and K. Yelick, "Efficient FFTs on IRAM", Proceedings of the 1st Workshop on Media Processors and DSPs, 1999
- S. Bouguezzel, M. O. Ahmad, and M. N. S. Swamy, "A New Radix-2/8 FFT Algorithm for Length- $q \times 2m$ DFTs," IEEE Trans. Circuits Syst. I, vol. 51, no. 9, September 2004.
- S. Bouguezzel, M. O. Ahmad, and M. N. S. Swamy, "New Radix- $(2 \times 2 \times 2)/(4 \times 4 \times 4)$ and Radix- $(2 \times 2 \times 2)/(8 \times 8 \times 8)$ DIF FFT Algorithms for 3-D DFT," IEEE Trans. Circuits Syst. I, vol. 53, no. 2, February 2006.
- S. Choi, G. Govindu, J. W. Jang, V. K. Prasanna "Energy-Efficient and Parameterized Designs of Fast Fourier Transforms on FPGAs," The 28th International Conference on Acoustics, Speech, and Signal Processing (ICASSP), April 2003.
- S. He and M. Torkelson "A New Approach to Pipeline FFT Processor," Proceedings of the IPPS, 1996.
- S. He and M. Torkelson "Design and Implementation of a 1024-point Pipeline FFT Processor," IEEE 1998 Custom Integrated Circuits.
- S.S. Wang and C.S. Li An Area-Efficient Design of Variable-Length Fast Fourier Transform Processor Journal of VLSI Signal Processing, March 2007.
- T. Lenart and V. Owall, "A 2048 Complex Point FFT Processor Using a Novel Data Scaling Approach," IEEE ISCAS 2003.
- W. H. Chang, T. Nguyen, "An OFDM-Specified Lossless FFT Architecture," IEEE Trans. Circuits Syst. I, vol. 53, no. 6, June 2006.
- www.altera.com
- www.xilinx.com
- Y. Ma, "An Effective Memory Addressing Scheme for FFT Processors", IEEE Transactions on Signal Processing, Vol. 47, No. 3, 907 - 911, May 1999
- Y. Ma, L. Wanhammar, "A Hardware Efficient Control of Memory Addressing for High-Performance FFT Processors", IEEE Transactions on Signal Processing, Vol. 48, No. 3, 917 - 921, March 2000.
- Y.N. Lin, H.Y. Liu, and C.Y. Lee "A 1-GS/s FFT/IFFT Processor for UWB Applications," IEEE Journ. of SSC, vol. 40, Issue 8, Aug. 2005.

VLSI

Edited by Zhongfeng Wang

ISBN 978-953-307-049-0

Hard cover, 456 pages

Publisher InTech

Published online 01, February, 2010

Published in print edition February, 2010

The process of Integrated Circuits (IC) started its era of VLSI (Very Large Scale Integration) in 1970's when thousands of transistors were integrated into one single chip. Nowadays we are able to integrate more than a billion transistors on a single chip. However, the term "VLSI" is still being used, though there was some effort to coin a new term ULSI (Ultra-Large Scale Integration) for fine distinctions many years ago. VLSI technology has brought tremendous benefits to our everyday life since its occurrence. VLSI circuits are used everywhere, real applications include microprocessors in a personal computer or workstation, chips in a graphic card, digital camera or camcorder, chips in a cell phone or a portable computing device, and embedded processors in an automobile, et al. VLSI covers many phases of design and fabrication of integrated circuits. For a commercial chip design, it involves system definition, VLSI architecture design and optimization, RTL (register transfer language) coding, (pre- and post-synthesis) simulation and verification, synthesis, place and route, timing analyses and timing closure, and multi-step semiconductor device fabrication including wafer processing, die preparation, IC packaging and testing, et al. As the process technology scales down, hundreds or even thousands of millions of transistors are integrated into one single chip. Hence, more and more complicated systems can be integrated into a single chip, the so-called System-on-chip (SoC), which brings to VLSI engineers ever increasingly challenges to master techniques in various phases of VLSI design. For modern SoC design, practical applications are usually speed hungry. For instance, Ethernet standard has evolved from 10Mbps to 10Gbps. Now the specification for 100Mbps Ethernet is on the way. On the other hand, with the popularity of wireless and portable computing devices, low power consumption has become extremely critical. To meet these contradicting requirements, VLSI designers have to perform optimizations at all levels of design. This book is intended to cover a wide range of VLSI design topics. The book can be roughly partitioned into four parts. Part I is mainly focused on algorithmic level and architectural level VLSI design and optimization for image and video signal processing systems. Part II addresses VLSI design optimizations for cryptography and error correction coding. Part III discusses general SoC design techniques as well as other application-specific VLSI design optimizations. The last part will cover generic nano-scale circuit-level design techniques.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

D. Reisis (2010). Fully Systolic FFT Architectures for Giga-Sample Applications, VLSI, Zhongfeng Wang (Ed.), ISBN: 978-953-307-049-0, InTech, Available from: <http://www.intechopen.com/books/vlsi/fully-systolic-fft-architectures-for-giga-sample-applications>

INTECH
open science | open minds

www.intechopen.com

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](https://creativecommons.org/licenses/by-nc-sa/3.0/), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen