A Power-Scalable Switch-Based Multi-Processor FFT

Bassam Jamil Mohd
Qualcomm, Inc.

Earl E. Swartzlander, Jr.
University of Texas

Abstract

This paper examines the architecture, algorithm and implementation of a switch-based multi-processor realization of the fast Fourier transform (FFT). The architecture employs M processing elements (PEs), and provides a speedup of M compared with systems that use a single PE. An algorithm is provided to detect and resolve memory conflicts. A CMOS implementation of a four-PE processor is presented. The design is reconfigurable to compute various FFT sizes. The design power consumption is scalable based on the number of active PEs. The timing, area and power results are discussed.

1. Introduction

The Fast Fourier Transform (FFT), introduced in [1], is a standard method for computing the Discrete Fourier Transform. Many FFT processor designs have been proposed to enhance the processor speed, memory access and power dissipation. The cache-FFT processor (presented in [2] and [3]) increases the processor speed and reduces communication energy by employing a specialized cache unit between the processor and memory. The design, however, is restricted to a single processor. The processor ring, proposed by [4] [5], employs four processing elements connected in a ring topology. The processor ring is capable of processing different FFT sizes with power scalable across FFT sizes. Several pipeline FFT designs were surveyed in [6]. The radix-r pipeline FFT processor has a speed up of $\log_r N$ compared with single PE. However, most conventional implementations of the pipeline FFT have high power dissipation due to extensive use of delay elements (i.e., shift registers) for interstage reordering [7]. The modular pipeline, introduced in [8], reduces significantly the number of delay elements by replacing the N-point pipeline with two $\sqrt{N}$-point pipelines combined via a special central element.

Several memory access schemes have been proposed. A multi-bank memory address assignment for a radix-r FFT was developed in [9]. An address generation scheme that uses four memories has been described [10]. The addressing scheme simplifies the address generation and reduces power. However, the scheme does not allow concurrent butterfly processing and requires complex memory initialization.

This paper presents a switch-based processor that employs M processing elements, $2M$ data memories and M Read Only Memories. Each processing element performs a radix-2 butterfly operation. In the following description, the use of a radix-2 decimation in frequency butterfly operation is assumed. Alternatively a decimation in time butterfly or a different radix could be used with minor revision to the architecture and the algorithm. The data memories are comprised of single-port memories where each has a size of $N/(2M)$; where N is the number of FFT points. Compared with a single processing element this approach provides a speed up of M.

A novel algorithm is implemented to detect and resolve memory conflicts. An early version of the switch-based processor with two PEs was discussed in [11]. A pipeline version of the architecture was introduced in [12].

In this paper, the switch-based approach is generalized for any number, M, of PEs $M \leq N/2$. A four-PE system has been simulated in 65nm CMOS technology [13]. The value of N and number of active PEs can be configured. The timing and power results are analyzed and compared with other designs. The power scalability feature is examined.

The paper is organized as follows. Sections II and III discuss the switch-based architecture and conflict free algorithm. In Section IV, the processor design and implementation are discussed in details. Finally, Section V presents conclusions.

2. Architecture

The N-point radix-2 FFT algorithm consists of $\log_2 N$ stages, where each stage executes $N/2$ complex butterfly operations. The operations in stage i depend

on the results from stage *(i-1)*, creating potential data dependencies between the stages. The switch-based architecture exploits parallelism within each stage by utilizing *M* PEs.

The architecture connects the elements of the FFT processor using a switch fabric. As a result, data can flow from any component to any component providing maximum data transfer flexibility. The data width is complex (i.e., double) words; where one word represents the real part and the other word represents the imaginary part of the complex data. An overview of the switch-based architecture is shown in Figure 1. The main components of the design are:

1) A switch fabric that connects PEs and memories.
2) PEs that perform the butterfly operations. The PE performs the radix-2 decimation in frequency butterfly operation:
   $c = a + b$
   $d = (a - b) * w$
   w is the twiddle factor. The PE consists of six real adders and four real multipliers. There are M PEs per stage, where:
   $N/2 \geq M \geq 1$,
   $M = 2^p$, where p is a non-negative integer ,
3) Memory arrays that store the intermediate results. There are 2*M memories, and the size of each memory is at least N/(2*M) entries. Each entry is two words wide, to store the real and imaginary parts for a complex data. Memories can be implemented as single-port RAM, caches or register files, based on the cost constraints.
4) Read-Only Memory Arrays (ROMs) that store the twiddle factors. There are M ROMs per stage, each with at least N/2 entries, where each entry stores the real and imaginary parts of one complex twiddle factor.

Figure 2 shows an example for *N*=16 and *M*=2. At each stage, two operations are executed simultaneously (i.e., one by each of the PEs). Hence, the eight operations of any stage require four cycles. Mapping an operation to a specific PE is handled by the memory management algorithm which is discussed later.

3. Conflict Free Algorithm

Memory conflicts occur when multiple PEs attempt to simultaneously access the same physical memory. In the selected radix-2 decimation in frequency FFT, memory conflicts do not occur in the early stages; they occur from stage $\log_2(M)$ to the last stage. The 16-point decimation in frequency FFT, shown on Figure 2, demonstrates memory conflicts. Stages 0 and 1 are conflict free, whereas stages 2 and 3 experience conflicts. Specifically, in stage 2, the inputs for the top PE are $x_2(0)$ and $x_2(2)$, both of which reside in MEM 0. Similarly, in stage 3 the inputs for the top PE are $x_3(0)$ and $x_3(1)$, both of which reside in MEM0.

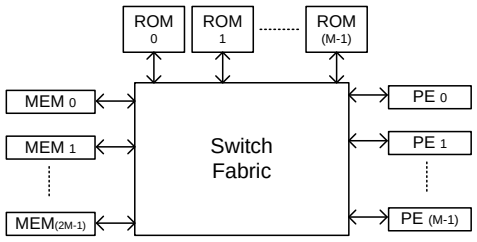

Figure 1. Switch-based Architecture

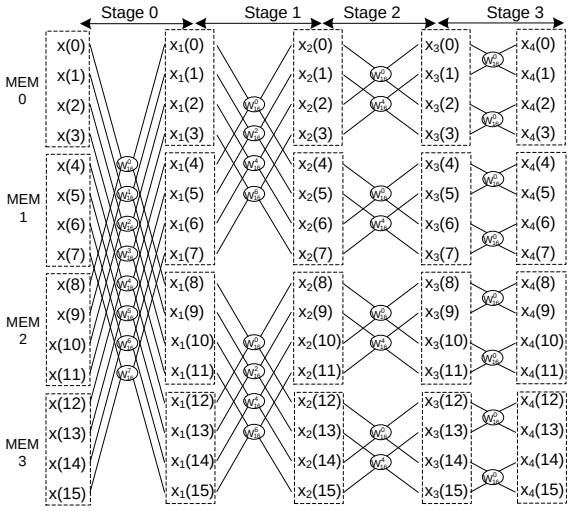

Figure 2. 16-Point FFT

To avoid memory conflicts, the proposed algorithm manipulates intermediate results in the memories using memory management operations. Let $x_i(t)$ and $x_j(t)$ be the i-th and j-th elements in stage t and i < j. Define the memory management operations as follows (see Figure 3):

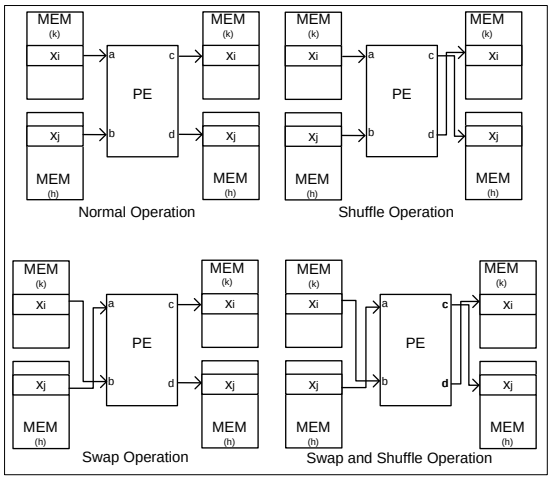

Figure 3. Memory Management Operations

- Normal Operation: Input x_i and x_j are provided to the first and second inputs (a and b) of the PE. The results (c and d) are saved in x_i and x_j,
- Shuffle Operation: Affects how PE results are written back in memory. In shuffle operation, the results (c and d) are saved in x_j and x_i,
- Swap Operation: The swap operation affects the order of PE inputs. In swap operation, x_i is provided to b and x_j is provided to a,
- Swap and shuffle operation: A PE operation can have both swap and shuffle memory operations at the same time.

As a result of reordering data in the processor, results from the last stage should be reordered.

Figure 4 shows the intermediate and final memory locations for a conflict free 16-point FFT. Also shown in Figure 4 are the swap and shuffle operations. Compare the following memory locations to those of Figure 2:

- In Stage-2 the inputs for the top butterfly are $x_2(0)$ and $x_2(2)$. There is no conflict since $x_2(0)$ and $x_2(2)$ reside in MEM 0 and MEM 1, respectively.
- Similarly, in Stage-3 the inputs for the top butterfly are $x_3(0)$ and $x_3(1)$ which reside in MEM 0 and MEM 1, respectively.

3.1. Conflict Free Algorithm Pseudocode

This section presents the algorithm pseducode.

Table *1* describes the important variables used in the algorithm. Table 2 describes the operation notation used in the algorithm.

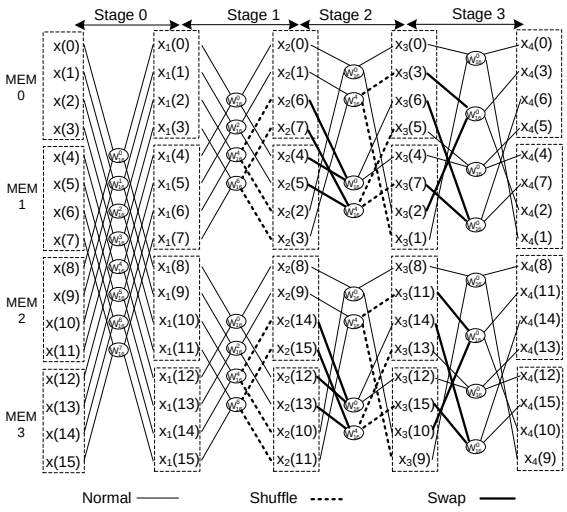

Figure 4. Conflict Free 16-Point FFT

Table 1. Algorithm Variable Description

Variable	Description
Number_Of_Stages	Total Number of Stages, which equal to $\log_2 N$
Safe_Stage	The highest safe
Cycles_Per_Stage	Number of cycles per stage
Mem_size	Memory Size = N/(2*PE_NUM)
Stage_Counter	The stage counter"
Cycle_Counter	The cycle counter
PE_ID	The PE_ID Counter
PE_Cycle_Counter	An intermediate constructed as follows: PE_Cycle_Counter=[PE_ID,Cycle_Counte]
Current_Group	An intermediate counter constructed as follows: Current_Group = the upper k-bits of PE_Cycle_Counter where k= Stage_Counter
Current_Operation	An intermediate counter constructed as follows: Current_Group = the lower k-bits of PE_Cycle_Counter where k= Number_Of_Stages - Stage_Counter-1

Table 2. Operations Description

Operation	Description
floor	The mathematical floor operation
mod	The modular operation
Memory$_{(M0\_Select)}$	Memory number M0_Select, where read and write and expressed as: **M0_data= Memory**$_{(M0\_Select)}$**[M0_addr]** **Memory**$_{(M0\_Select)}$ **[M0_addr]=Result1**
Cycle_Counter[i]	The i-th bit of Cycle_Counter

Algorithm Pseudocode

```
// Preparation Step
Number_Of_Stages   = log2 N
Cycles_Per_Stage   = N/(2*NUMBER_OF_PE)
Mem_size           = N/2^(NUMBER_OF_PE+1)
Safe_Stage         = log2NUMBER_OF_PE
// Start main loops
for Stage_Counter=0 to (Number_Of_Stages -1)
 Group_Size = N/2^(Stage_Counter+1)
 for Cycle_Counter=0 to (Cycles_Per_Stage -1)
  for PE_ID=0 to (NUMBER_OF_PE -1)

   // Calculate Operation Indices
   PE_Cycle_Counter = Cycles_Per_Stage *
                      PE_ID
                      + Cycle_Counter
   Stage_Counter_Rev = Number_Of_Stages -
                       Stage_Counter - 1
   Current_Group     = floor(PE_Cycle_Counter/
                       2^Stage_Counter_Rev)
   Current_Operation = PE_Cycle_Counter mod
                       (2^Stage_Counter_Rev)

   // Generate memory selects and addresses
   M0_addr = Cycle_Counter
   If Stage_Counter <= Safe_Stage
     M1_addr = M0_addr
   Else
     K = Stage_Counter - Safe_Stage
     M1_Addr = Invert upper K-bits of M0_Addr0
   End

   If Stage_Counter <= Safe_Stage
     Group_Offset = Current_Group *
                    (N /2^Stage_Counter)
     Group_Count  = PE_Cycle_Counter mod Group_Size
     Memory_Count = floor (Group_Count / Mem_size)
     Offset       = Memory_Count * Mem_size
     M0_offset    = Offset + Group_Offset
     M1_offset    = Offset + Group_Offset
                  + Group_Size
   Else
     Memory_Count = PE_ID
     Offset    = 2 * Memory_Count * Mem_size
     M0_offset = Offset;
     M1_offset = Offset + 2 * Mem_size
   End

   M0_Select = floor ( M0_offset / Mem_size);
   M1_Select = floor ( M1_offset / Mem_size);

   // Read data and perform swap operation
   If Stage_Counter > Safe_Stage  AND
     Cycle_Counter[Number_Of_Stages-Stage_Counter-1]
     M1_data = Memory(M0_Select)  [ M0_addr ]
     M0_data = Memory(M1_Select)  [ M1_addr ]
   else
     M0_data = Memory(M0_Select)  [ M0_addr ]
     M1_data = Memory(M1_Select)  [ M1_addr ]
   End

   ROM_SELECT  = PE_ID
   ROM_Address = Current_Operation * 2^Stage_Counter
   W    = ROM(Stage_Counter, ROM_SELECT)  [ROM_Address ]

   // Enable PE to perform FFT butterfly operation
   [Result1, Result0] =
         PE_PE_ID(M0_data, M1_data, W);

   // Write the results to the Memory
   If  Stage_Counter >= Safe_Stage
    AND  Stage_Counter < (Number_Of_Stages-1)  AND
    Cycle_Counter[Number_Of_Stages-Stage_Counter-2]
     // Shuffle the results
     Memory(M0_Select)  [ M0_addr ] = Result1
     Memory(M1_Select)  [ M1_addr ] = Result0
   Else
     // No Shuffling
     Memory(M0_Select)  [ M0_addr ] = Result0
     Memory(M1_Select)  [ M1_addr ] = Result1
   End

  end // PE_ID
 end // Cycle_Counter loop
end // Current_Stage loop
```

4. Processor Design

This section discusses the design and implementation of a four-PEs switch-based processor. The specifications of the intended design are outlined in Table 3. The processor execution is configured by the parameters: the number of PEs, M, the transform length, N and the number of blocks.

Table 3. FFT Processor Specifications

Parameter	Specification
FFT Type	Radix-2 decimation-in-frequency
Number of PEs (*M*)	Four
Number of points (*N*)	Configurable to: 64, 256, 1024 and 4096 points
Number of blocks (*B*)	Configurable to up to 15 blocks of *N*-points, one block is *N*-points
Word length	Word length is 32-bit.
Data format	Two's complement fixed point format.
Power scalability	Scales with number of active PEs
Target Clock Rate	200 MHz
Target Throughput (N=1024)	50 MSPS
Target Technology	Low power 65-nm CMOS [13]

4.1. Design Overview

Figure 5 provides a more detailed view of the processor. The design consists mainly of the memory and the switch sub-systems. The memory sub-system contains two sets of RAMs and an external memory controller. The primary task of the memory sub-system is to move data between the RAMs and the external memory. The RAMs serve as caches in the design. When an FFT computation is being preformed on one set of RAMs, the other set swaps data with the external memory. Each RAM set employs eight 512 by 64-bit RAMs. Each RAM 64-bit word contains the two components of one complex data word.

The switch sub-system consists of the switch fabric, four PEs, four ROMs and a register file. The switch sub-system computes the FFT for one of the RAM sets. The processor can be programmed to process multiple blocks of N-points. A block consists of one set of N-points.

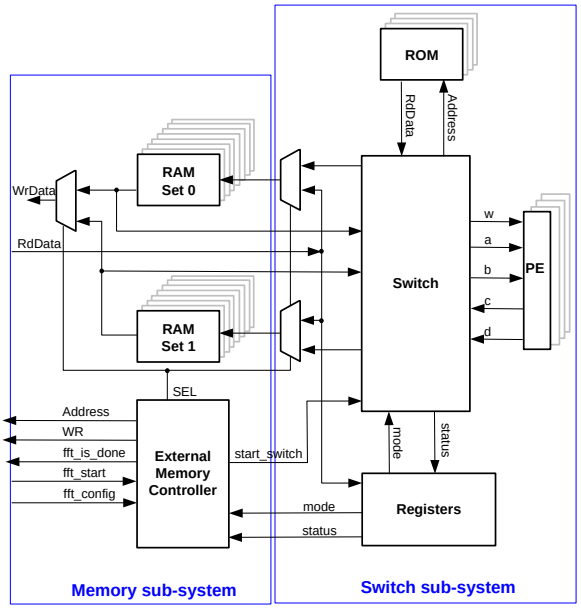

Figure 5. Processor Block Diagram

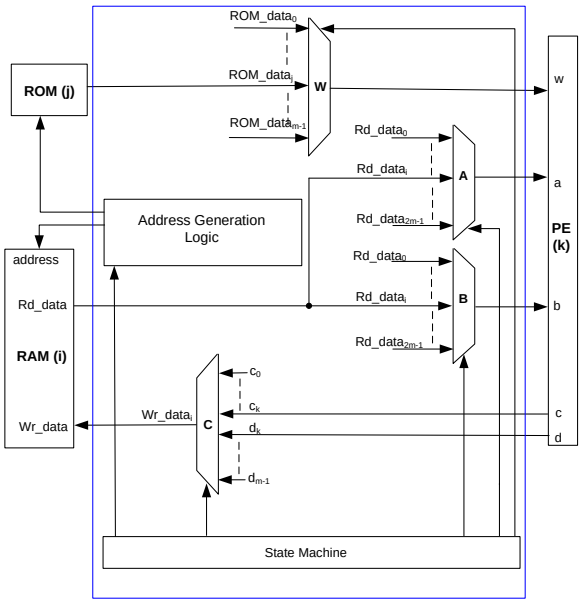

Figure 6. The Switch Block Diagram

The main components of the switch are shown in Figure 6. They are the data routing multiplexers, the state machine and the address generation logic. The state machine controls the operation of the switch and generates the multiplexer selection signals and updates the counter values. The multiplexers select the appropriate input data for the PE input ports and the RAM input ports. The address generation logic creates the addresses for the ROM and RAM using the stage counter and cycle counter. The stage counter indicates the current stage number in the FFT algorithm. Each stage has $N/(2*M)$ cycles. The cycle counter indicates the current cycle number. The generation of the multiplexer selection and addresses is implemented as outlined in the pseudo-algorithm (see Section 4.1.)

4.2. Implementation Results

Table 4 summarizes the physical design of the processor.

Table 4. Physical Design Results

Category	Parameter	Parameter
Area	Area	1.65 mm^2
Timing	Frequency	188 MHz
Power	Power Equation	30mW* Num_Active_PEs
Switch Overhead	Switch Power Overhead	11%
	Switch Timing Overhead	20%

The main critical timing path in the layout design is the path timing path starts and ends with two RAMs and propagates through the PE logic. The path travels from the source RAM to the switch logic, to PE_0, back to the switch and then finally to the destination RAM. The data shows that: The overhead of the switch delay is 20%. With 64% of the timing path in its logic, the dominant component in the critical path is the PE delay. On the other hand, the impact of the switch architecture on timing is less than one-fourth of the cycle time. This indicates that when increasing the number of PEs, the switch is not likely to be the gating factor for timing.

The power analysis for the processor is based on the computation of 1024-point FFTs. Figure 7 shows the power consumption of the following components (see

Figure 5): PE, RAMs, ROMs, switch and memory controller. Additionally, the clock power represents the clock tree power and the buffer component represents the buffers and invertors added to buffer long wires in the layout. The power (in Figure 7) is illustrated for different number of active PEs, where. n-PE designates n-active PE case.

Power analysis indicates the following observations. The switch power dissipation is 11% of the total power. The power consumption of all elements (except for the clock and the external memory controller) are proportional to the number of active processors. The clock and external memory controller exhibit constant power consumption independent of the number of active processors. Fortunately, their contributions are

Table 5. Comparing the Switch-based Processor and the Ring-based Processor

Processor	Clock Cycles (N = 1024)	Data Rate (N = 1024)	Energy (N = 1024)	Active Power
Switch-based processor	3840	50.1 MSPS	2.4 µJ	120 mW
Ring-based processor	5280	38.8 MSPS	10.8 µJ	410 mW

small compared to other linear components. Moreover, the PEs are equally active in across all the cases. This is to avoid hot spot build ups inside the processor. Furthermore, the total power consumption is approximately 30 mW per active PE.

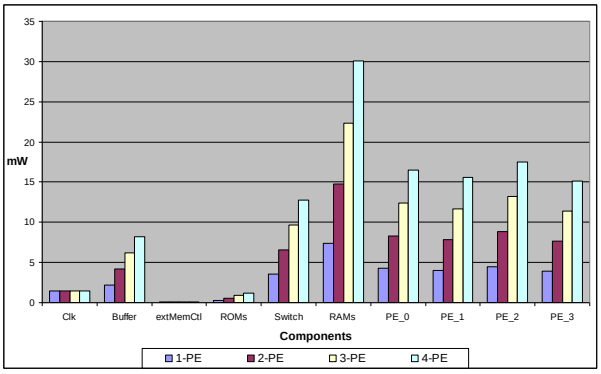

Figure 7. Power Consumption vs. Number of Active PEs

4.3. Comparison

The switch based processor is compared with the ring-processor [4] [5] since it presents a multi-processor design which utilizes the same number of PEs. Also, the ring processor has similar memory. The comparison criteria are explained below.

The total number of cycles to process an *N*-point FFT is an important criterion to compare different architectures since it is a measure of the efficiency of the architecture [4] The second column of Table 5 illustrates the number of cycles to process a 1024-point FFT. The switch-based processor requires about 1/3 fewer cycles than the ring processor.

The last three columns in Table 5 compare features that are technology and process dependent. It is extremely difficult to accurately compare the speed and power of implementations from different technologies. For example, each technology node (e.g., 65nm bulk CMOS) has several flavors (hi/normal/low V_T, low-power, etc.) and differ significantly from vendor to vendor. Furthermore, the physical design tools and standard-cell libraries that are used have an effect on the performance of the final design.

The third column in Table 5 gives the sustained data rate for a 1024-point FFT (i.e., samples per second). The switch-based processor has about a 30% higher sustained data rate than the ring processor. The fourth column shows that the switch-based processor consumes about ¼ as much energy to compute a 1024-point FFT as the ring processor. Moreover, while running, the switch-based processor consumes about 1/3 the power of the ring processor.

5. Conclusion

This paper has examined the realization of an FFT by a switch-based architecture that consists of a switch fabric, *M* processing elements and 2*M* memories. A four-PE processor was designed and implemented in 65 nm low power bulk CMOS technology. The timing and power analysis was presented in detail.

The results show that switch has very modest impact on the processor timing and power. This suggests that the number of PEs could be increased with a proportionate reduction in the time to compute a transform. Similarly, reducing the number of processors will reduce the power consumption of the switch-based processor.

6. References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297-301, 1965.

[2] B.M. Baas, "A generalized cached-FFT algorithm," *IEEE International Conference on Acoustic, Speech and Signal Processing*, Vol. 5, pp. 89-92, March 2005.

[3] B. M. Baas, "A low-power high-performance 1024-point FFT processor," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 380-387, March 1999.

[4] G. Zhong, F. Xu and A. N. Willson, Jr., "A power-scalable reconfigurable FFT/IFFT IC based on a multi-processor ring,"

IEEE Journal of Solid-State Circuits, Vol. 41, pp. 483-495, February 2006.

[5] G. Zhong, F. Xu and A. N. Willson , Jr., "An energy-efficient reconfigurable FFT/IFFT processor based on a multi-processor ring," *XII European Signal Processing Conference (EUSIPCO)*, pp. 2023-2026, 2004.

[6] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," *Proc. of URSI International Symposium on Signals, Systems, and Electronics*, pp. 257-262, 1998.

[7] Earl E. Swartzlander, Jr., "Systolic FFT processors: past, present and future," *IEEE Conference on Application-Specific Systems, Architectures, and Processors*, pp. 153-158, September 11-13, 2006.

[8] A. M. El-Khashab and E. E. Swartzlander, Jr., "A modular pipelined implementation of large fast Fourier transforms," *Proc. of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, pp. 995-999, November 2002.

[9] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Transactions on Circuits and Systems, II*, vol. 39, pp. 312-316, 1992.

[10] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Transactions on Signal Processing*, vol. 48, pp. 917-921, 2000.

[11] H. Saleh, B. Mohd, A. Aziz and E. E. Swartzlander, Jr., "Contention-free switch-based implementation of 1024-point Fourier transform engine," *25th IEEE International Conference on Computer Design*, pp. 7-12, October 2007.

[12] B. J. Mohd, A. Aziz and E. E. Swartzlander, Jr., "The hazard-free superscalar pipeline fast Fourier transform algorithm and architecture," *15th Annual IFIP VLSI SoC Conference*, pp. 194-199, Atlanta, October 2007.

[13] S. Fung, *et al.*, "65nm CMOS high speed, general purpose and low power transistor technology for high volume foundry application," *Digest of Technical Papers Symposium on VLSI Technology***,** pp. 92-93, June 2004.