

# Computer Architectures for Embedded System-I

---

## Implementation of Fixed-Point FFT(512, 1K, 2K, 4K) on FPGA

*Submitted by*

**Adnan Muhammad Niazi**

Student No: s1029789

*and*

**Anoop Ambooken**

Student No: s1035312

*Submitted to:*

**Dr. André Kokkeler**

November 18, 2010

# Table of Contents

---

1.1 Fast Fourier Transform (FFT)	3
1.2 FFT Butterfly	5
1.2-1 DIT Butterfly	6
1.2-2 DIF Butterfly	6
1.3 Decimation in Time (DIT) and Decimation in Frequency (DIF) FFT	7
2.1 Mapping FFT on FPGA	8
2.2 FPGA Implementation of FFT via Sys Gen for DSP and Simulink	9
2.3 Specs of the FPGA Target Device	11
2.4 Profiling Results for FPGA Implementation	11
3.1 Mapping FFT on PC	14
3.1-1 Penryn Processor Architecture	14
3.1-1 Penryn Strength and Weaknesses	15
3.2 Mapping Radix2 DIT FFT on Penryn	15
References	16

## 1.1 Fast Fourier Transform (FFT)

---

The *Discrete Fourier Transform* (DFT) is a specific kind of Fourier transform which transforms one function into another, which is called the frequency domain representation, or simply the DFT, of the original function (which is often a function in the time domain). The DFT is given by:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-i \frac{2\pi}{N} kn} \quad (1)$$

$e^{-i \frac{2\pi}{N} kn}$  is commonly known as the Twiddle Factor.

Expression (1) is discrete Fourier transform — DFT. Here  $\{x_0, x_1, \dots, x_{N-1}\}$  is the input discrete function and  $\{X_0, X_1, \dots, X_{N-1}\}$  is the result of Fourier transform.

Calculating a DFT is very computationally intensive. DFT has a complexity of  $O(N^2)$ . Over the years, faster versions of DFT have appeared that are called Fast Fourier Transform or FFT. These have the complexity of  $O(N \log_2 N)$ . The *Cooley-Tukey algorithm*, named after J.W. Cooley and John Tukey, is the most common fast Fourier transform (FFT) algorithm. It re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size  $N = N_1 N_2$  in terms of smaller DFTs of sizes  $N_1$  and  $N_2$ , recursively, in order to reduce the computation time to  $O(N \log N)$  for highly-composite  $N$  (smooth numbers).

Let us put  $N=8$  and write down our DFT:

$$X_n = x_0 + x_1 e^{-i \frac{2\pi}{8} n} + x_2 e^{-i \frac{2\pi}{8} 2n} + x_3 e^{-i \frac{2\pi}{8} 3n} + x_4 e^{-i \frac{2\pi}{8} 4n} + x_5 e^{-i \frac{2\pi}{8} 5n} + x_6 e^{-i \frac{2\pi}{8} 6n} + x_7 e^{-i \frac{2\pi}{8} 7n}$$

We can split the sum into two similar sums separating odd and even terms and factoring out the latter sum:

$$X_n = [x_0 + x_2 e^{-i \frac{2\pi}{8} 2n} + x_4 e^{-i \frac{2\pi}{8} 4n} + x_6 e^{-i \frac{2\pi}{8} 6n}] + e^{-i \frac{2\pi}{8} n} [x_1 + x_3 e^{-i \frac{2\pi}{8} 2n} + x_5 e^{-i \frac{2\pi}{8} 4n} + x_7 e^{-i \frac{2\pi}{8} 6n}]$$

Now we can split the sums in brackets again:

$$X_n = [(x_0 + x_4 e^{-i \frac{2\pi}{8} 4n}) + e^{-i \frac{2\pi}{8} 2n} (x_2 + x_6 e^{-i \frac{2\pi}{8} 4n})] + e^{-i \frac{2\pi}{8} n} [(x_1 + x_5 e^{-i \frac{2\pi}{8} 4n}) + e^{-i \frac{2\pi}{8} 2n} (x_3 + x_7 e^{-i \frac{2\pi}{8} 4n})]$$

Thus, we have  $3 - \log_2 8$  — levels of summation. The deepest one is in parenthesis, the middle one in brackets and the outer one. For every level exponent multiplier for the second term is the same.

$$X_n = [(x_0 + x_4 e^{-i \pi n}) + e^{-i \frac{\pi}{2} n} (x_2 + x_6 e^{-i \pi n})] + e^{-i \frac{\pi}{4} n} [(x_1 + x_5 e^{-i \pi n}) + e^{-i \frac{\pi}{2} n} (x_3 + x_7 e^{-i \pi n})]$$

And now the most important observation one can make to get speed-up: periodicity of the exponent multipliers.

$$e^{i(\emptyset+2\pi)} = e^{i\emptyset}$$

For the exponent multiplier in parenthesis period is  $n=2$ , which means, sums in parenthesis are exactly the same for  $n=0, 2, 4, 6$  and for  $n=1, 3, 5, 7$ . It means on deepest level in parenthesis we need  $4 \times 2 = 8$  — number of sums times period — sums in total. And note, since  $n=1, 3, 5, 7$  corresponds to the half of the period  $\pi$ , exponent multiplier is the same as for  $n=0, 2, 4, 6$  but with the opposite sign.

$$e^{i(\emptyset+\pi)} = -e^{i\emptyset}$$

Indeed they are 1 for  $n=0, 2, 4, 6$  and -1 for  $n=1, 3, 5, 7$ .

For the exponent multiplier in brackets period is  $n=4$ , which means we have the same sums for pairs  $n=0, 4$ ;  $n=1, 5$ ;  $n=2, 6$  and  $n=3, 7$ . It means on the middle level in brackets we have  $2 \times 4 = 8$  sums and the second half of them could be received again by changing sign in first half of them — due to the fact distance between  $n$  and  $n+2$  is  $\pi$ . Thus, for  $n=0, 4$  the factor is 1 and for  $n=2, 6$  it is -1; for  $n=1, 5$  it equals  $-i$  and for  $n=3, 7$  it is  $i$ .

On the outer level we have just one sum for every transform component, and the period of the exponent multiplier is 8. This gives us  $1 \times 8 = 8$  sums and second half of them could be received by changing sign in the first half.

So, on every calculation level we have 8 sums. In terms of  $N$  it means we have  $\log_2 N$  levels and  $N$  sums on every level, which gives us  $O(N \log_2 N)$  order of number of operations. On the other hand having the constant number of sums on every level means we can process data in place.

In summary, we have got fast Fourier transform — FFT.

## 1.2 FFT Butterfly

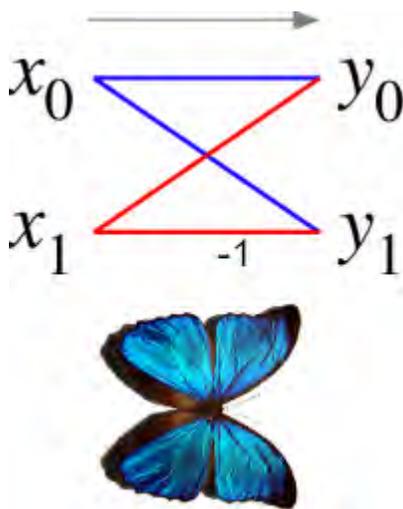
---

A butterfly is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into sub transforms). The name "butterfly" comes from the shape of the data-flow diagram in the radix-2 case, as described below:

$$y_0 = x_0 + x_1 W_N^x$$

$$y_1 = x_0 - x_1 W_N^x$$

where  $W_N^x$  is the twiddle factor.



*Figure 1.2: Data-flow diagram connecting the inputs  $x$  (left) to the outputs  $y$  that depend on them (right) for a "butterfly" step of a radix-2 Cooley-Tukey FFT. This diagram resembles a butterfly, and hence the name*

## 1.2-1 DIT Butterfly

---

The DIT butterfly shown in Figure 1.2-1 is the basic processing element of any FFT algorithm based on decimation in time approach. Mathematically, it can be expressed as :

$$y_0 = x_0 + x_1 W_N^x$$

$$y_1 = x_0 - x_1 W_N^x$$

where  $W_N^x$  is the twiddle factor.

The multiply operation occurs before add/subtract

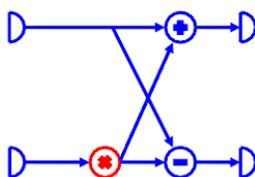


Figure 1.2-1: DIT Butterfly. The multiply operation occurs before add/subtract

## 1.2-2 DIF Butterfly

---

The DIF butterfly shown in Figure 1.2-2 is the basic processing element of any FFT algorithm based on decimation in frequency approach. Mathematically, it can be expressed as :

$$y_0 = x_0 + x_1$$

$$y_1 = (x_0 - x_1) W_N^x$$

where  $W_N^x$  is the twiddle factor.

The multiply operation occurs after add/subtract

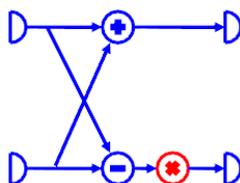


Figure 1.2-2: DIF Butterfly. The multiply operation occurs after add/subtract

### 1.3 Decimation in Time (DIT) and Decimation in Frequency (DIF) FFT

The FFT explained in 1.1 is an example of Decimation in time (DIT) FFT. It is called so because the time domain samples are decimated (i.e. rearranged in bit reserved order) and the frequency domain samples are discharged in the Natural order. This kind of FFT used a DIT butterfly as its basic processing engine.

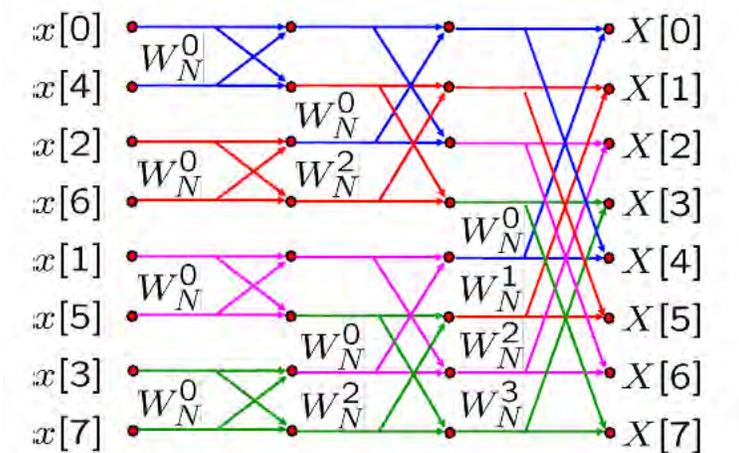


Figure 1.3-1: shows an 8-point Decimation in Time (DIT) FFT. Note that the time domain samples are in bit reversed order while the frequency domain samples are in the natural order.  $W_N^x$  are the various twiddle factors.

The streaming FFT implemented in FPGA is done using Decimation in frequency (DIF) FFT. In this approach, the time domain sample are input in the Natural order and the frequency domain output is decimated (i.e. arranged in a Bit reversed order). The fact the no bit reversed ordering is required at the input side make possible the streaming behavior of this FFT. The DIF butterfly is the basic processing engine of a DIF FFT.

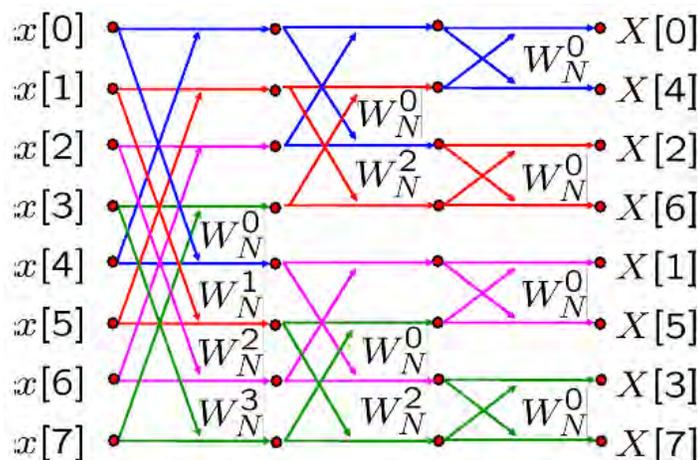


Figure 1.3-2: shows an 8-point Decimation in Frequency (DIF) FFT. Note that the frequency domain samples are in bit reversed order while the time domain samples are in the natural order.

## 2.1 Mapping FFT on FPGA

The fast Fourier transform (FFT) is one of the most widely used and important signal processing functions, for example, in applications related to digital communications and image processing. Since the computational complexity of the FFT is  $O(N \log_2 N)$ , where  $N$  is the number of inputs, the FFT potentially requires multi-cycle processing, and can become a major bottleneck for overall system performance. To relieve this bottleneck, many commercial FPGA IP blocks provide a streaming form of the FFT with single-cycle-per-sample throughput. This high-throughput form of FFT comes by exploiting the locality of reference and massive parallelism offered by modern FPGA fabrics. Multiple butterflies can be easily implemented on the FPGA. The twiddle factors and the intermediate data samples can be stored either in block RAM or Distributed RAM or a combination of both. This completely eliminates the need for off-chip memory access which may slow down the FFT considerably.

The Pipelined, Streaming I/O solution pipelines several Radix-2 butterfly processing engines to offer continuous data processing. More specifically the architecture used for FFT implementation is  $R_2^2$  SDF ( $R_2^2$  Single path delay Feedback- shown in Figure 2.1-2) which is an efficient version of  $R_2$  SDF ( $R_2$  Signal Path Delay Feedback shown in Figure 2.1-1) implementation. The successive stages or iterations of the fundamental algorithm are each carried out in the separate cascaded modules. Using shift registers as digital delay lines permits new data to be entered into the processor while the processing of earlier data blocks is carried out. In effect the overall delay required is equal to the time required to gather the analysis sample block  $N$  in each of the separate channels. As the  $N$ th complex data sample is loaded into the digital delay line, the first analysis frequency appears at the output.

The input data is presented in natural order. The unloaded output data can either be in bit reversed order or in natural order. When natural order output data is selected, additional memory resource is utilized. This architecture covers point sizes from 8 to 65536. The user has flexibility to select the number of stages to use block RAM for data and phase factor storage. The remaining stages use distributed memory.

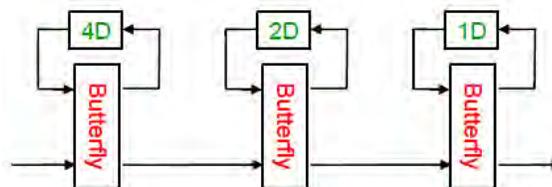


Figure 2.1-1:  $R_2$ SDF architecture for implementing an 8-point streaming FFT

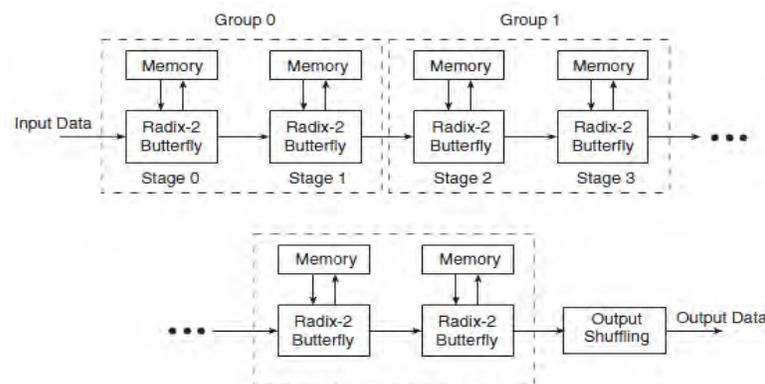


Figure 2.1-2:  $R_2^2$ SDF architecture for implementing an  $N$ -point streaming FFT

## 2.2 FPGA Implementation of FFT via Sys Gen for DSP and Simulink

Xilinx provides a synthesizable FFT core as part of the ISE design Suite: Embedded Edition. *Xilinx System generator for DSP* in conjunction with *Simulink* is used to synthesize this highly configurable core. This approach has the following benefits:

1. No need to write the FFT VHDL code
2. Test vectors can be generated in Simulink and applied to the FFT core
3. Computer results from the FFT core could be conveniently observed in the Simulink environment

This approach of using highly customizable cores can also be extended to Altera's FPGA product line. However in that case, *Altera DSP builder* in combination with Simulink should be used for the design process.

Figure 2.2.1 shows the Simulink model of the pipelined streaming FFT. A Cosine signal is given as the real input and a Sine signal is given as the imaginary input to the FFT core. The 'In' gateways are used to convert the double floating point precision of Matlab into a Fix point format that the FFT core can work with. Fix\_16\_15 shows that the fixed point number has one integer and 15 fractional bits. On the output side an 'Out' gateway is used to convert the fixed point numbers generated by the core to the double precision floating point representation used by the Simulink environment.

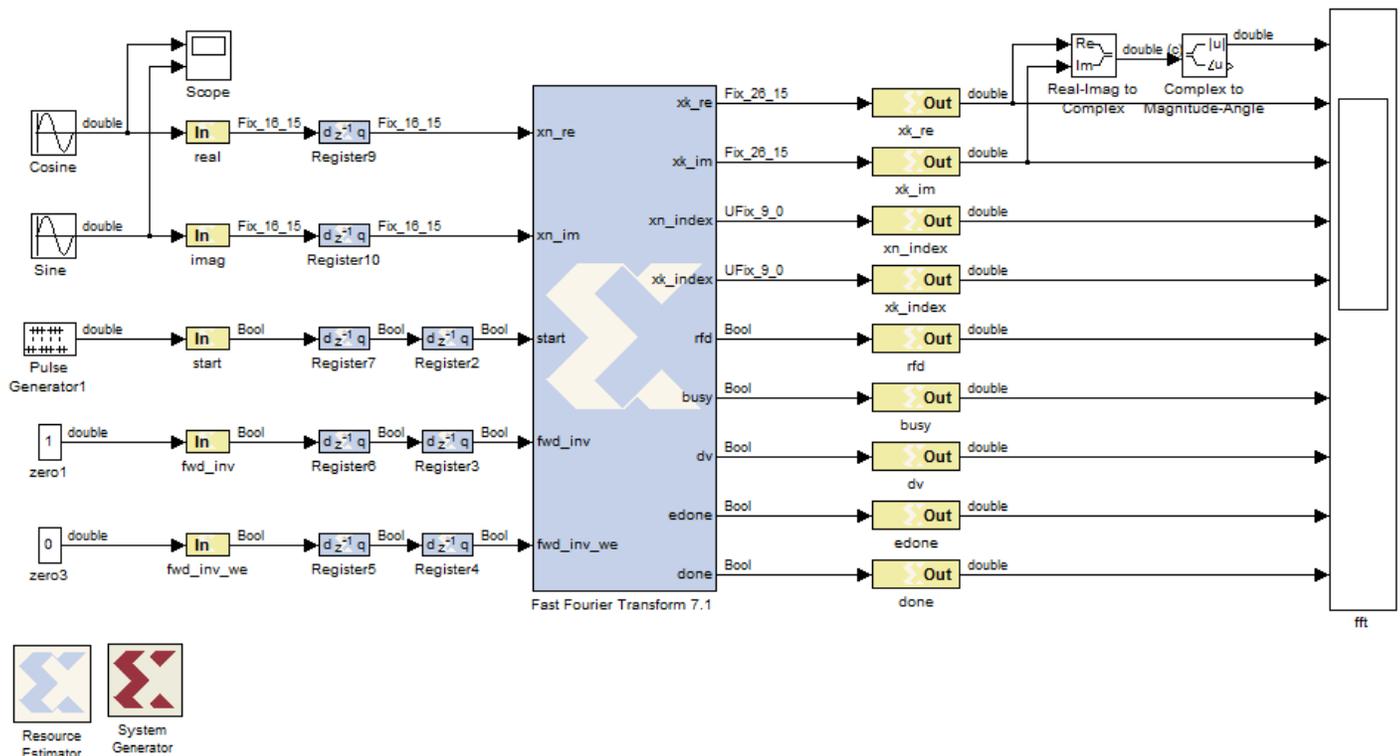


Figure 2.2-1: Simulink-System Generator Model for Streaming FFT

Figure 2.2-2 shows the post synthesis simulation results of the 512 point streaming FFT. The first trace labeled FFT shows the magnitude of the computed FFT. The two traces below it are the real and imaginary components of the FFT. The trace labeled  $xn\_index$  is the index of the input samples.  $xk\_index$  is the index of the output samples. Because the core was configured to display the FFT results in Natural order, therefore,  $xk\_index$  is a linearly increasing. If the core is configured to output the samples in bit reversed order then the  $xk\_index$  won't be displayed as linearly increasing (see Figure 2.2-3). The other signal like rfd, data valid, done and edone are not significant in our case but can be very useful when calculating and IFFT (Inverse fast fourier transform) immediately after calculating the FFT

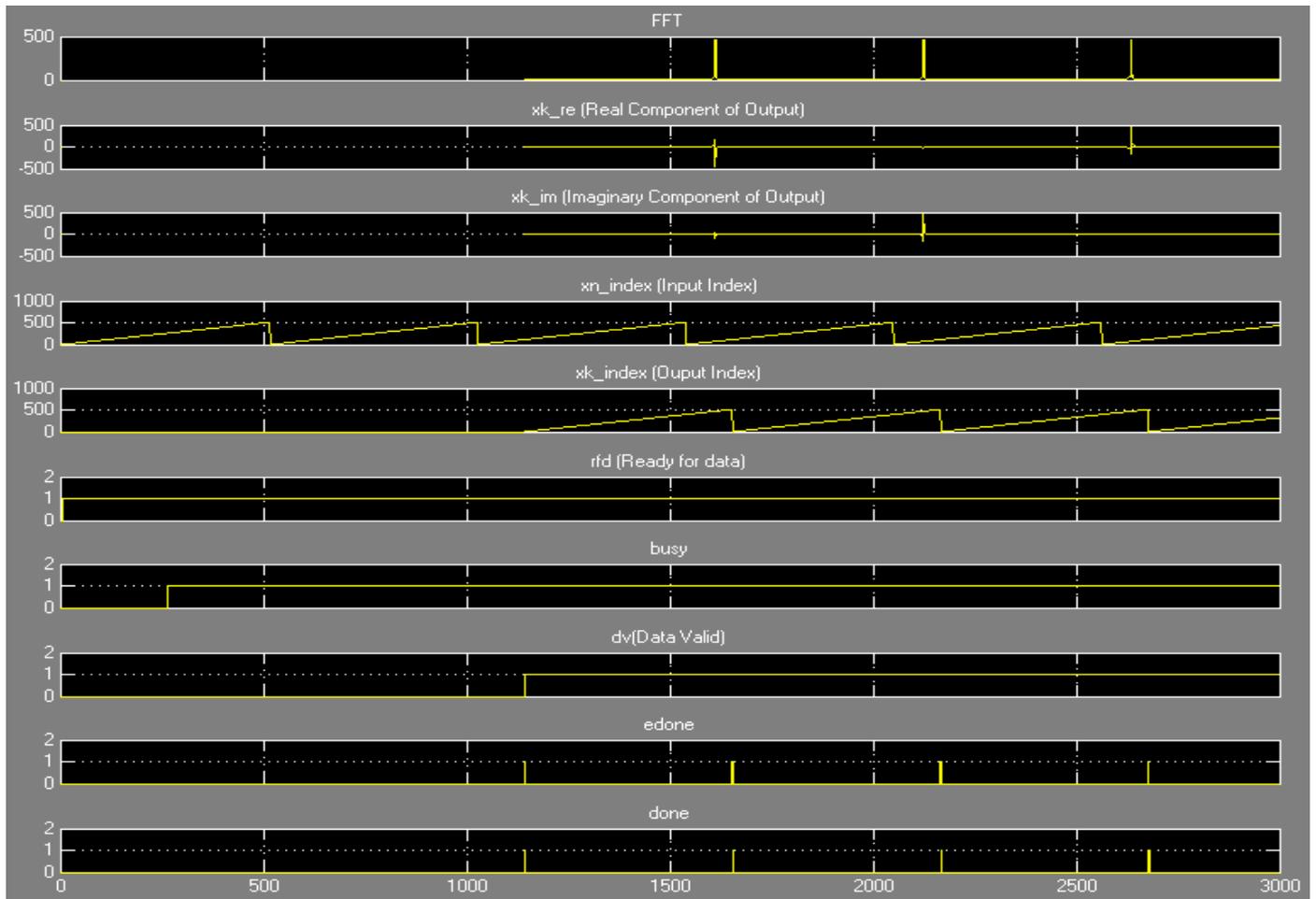


Figure 2.2-2: Scope view of the FFT and various signals of the FFT core. The core is configured to output the FFT in Natural order

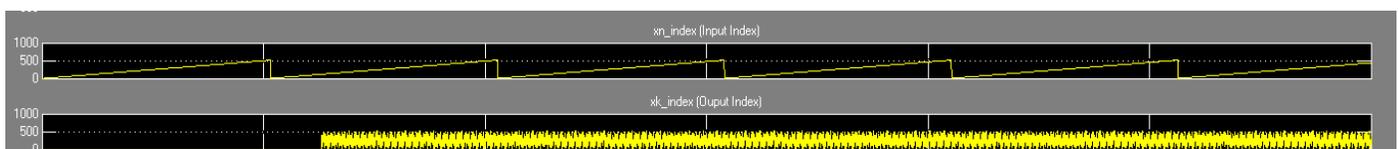


Figure 2.2-3: Scope view of the  $xn\_index$  and  $xk\_index$  when the core is configured to output the FFT in Bit reversed order. You can see that the output index  $xk\_index$  is no longer increasing linearly

## 2.3 Specs of the FPGA Target Device

---

Target Device: Virtex6 XC6VLX240T -1  
Speed Grade: -1 (Highest speed in the Virtex6 Family)  
Logic cells: 241,152  
<sup>1</sup>CLB Slices: 37,680  
Distributed RAM: 3,650 Kb  
<sup>2</sup>DSP48E1 Slices: 768  
Block RAM: 14,976 Kb  
Max User I/O: 720

## 2.4 Profiling Results for FPGA Implementation

---

When it comes to synthesis the Xilinx FFT LogiCORE provides two implementation options:

1. Resource Optimized Implementation &
2. Performance Optimized Implementation

### 1. Resource Optimized Implementation

In this implementation the core used minimum number of faster components such as Xtreme DSP slices and distributed RAM. Because most of the logic is implemented using CLB slices the implementation is slow. Figure 2.4-1 shows the resource utilization figure for 512, 1K, 2K, 4K FFT when the core is configured for resource optimization. From the graph it's obvious that the larger the N, the greater the resources utilized. Figure shows the power consumption and maximum clock frequency achieved by 512, 1K, 2K, 4K FFT when the core is configured for resource optimization. Because more resource utilization occurs as N increases therefore the power consumption increases as well. The through put falls as N increases because the maximum clock speed the core operates on, lowers. This happens because chain of cascaded butterflies increases as N increases which make the setup and hold times larger thereby, necessitating a decrease in the clock speed.

---

<sup>1</sup> Each Virtex-6 FPGA slice contains four LUTs and eight flip-flops, only some slices can use their LUTs as distributed RAM or SRLs.

<sup>2</sup> Each DSP48E1 slice contains a 25 x 18 multiplier, an adder, and an accumulator.

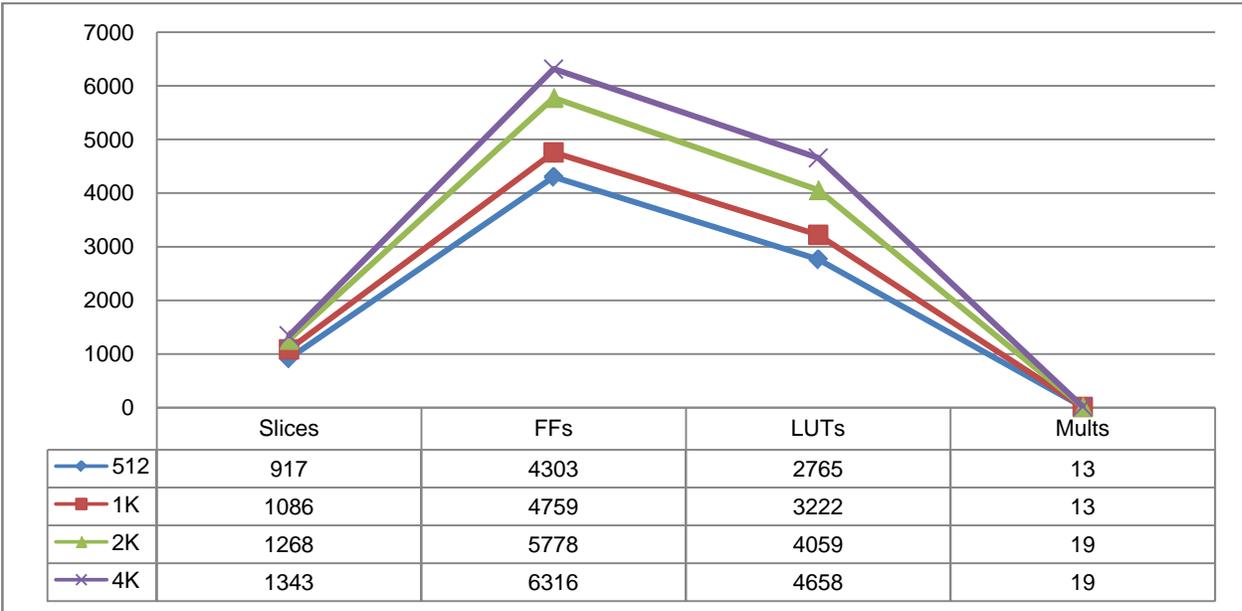


Figure 2.4.-1: Resource utilization statistics for 512,1k, 2K, 4K point FFT when the core is synthesized for resource optimization

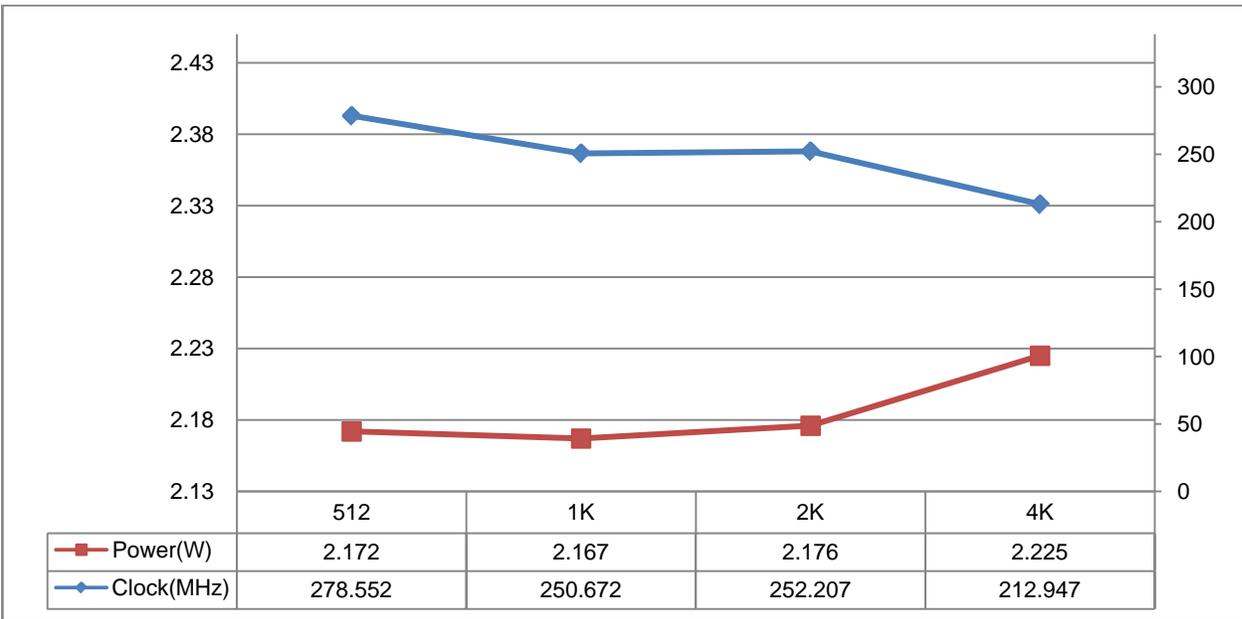


Figure 2.4-2: Power Consumption and Maximum achievable clock frequency statistics for 512,1k, 2K, 4K point FFT when the core is synthesized for resource optimization

## 2. Performance Optimized Implementation

In this implementation the core tries to utilize as much Xtreme DSP slices and distributed RAM as possible. Because the distributed RAM can be located anywhere in the FPGA fabric, therefore, the interconnect is much smaller which help increase the clock speed. A further increase in clock speed occurs because of the use of the fast and highly optimized Xtreme DSP slices like DSP48E1 multipliers. However, this increase in throughput comes at a price of higher power consumptions.

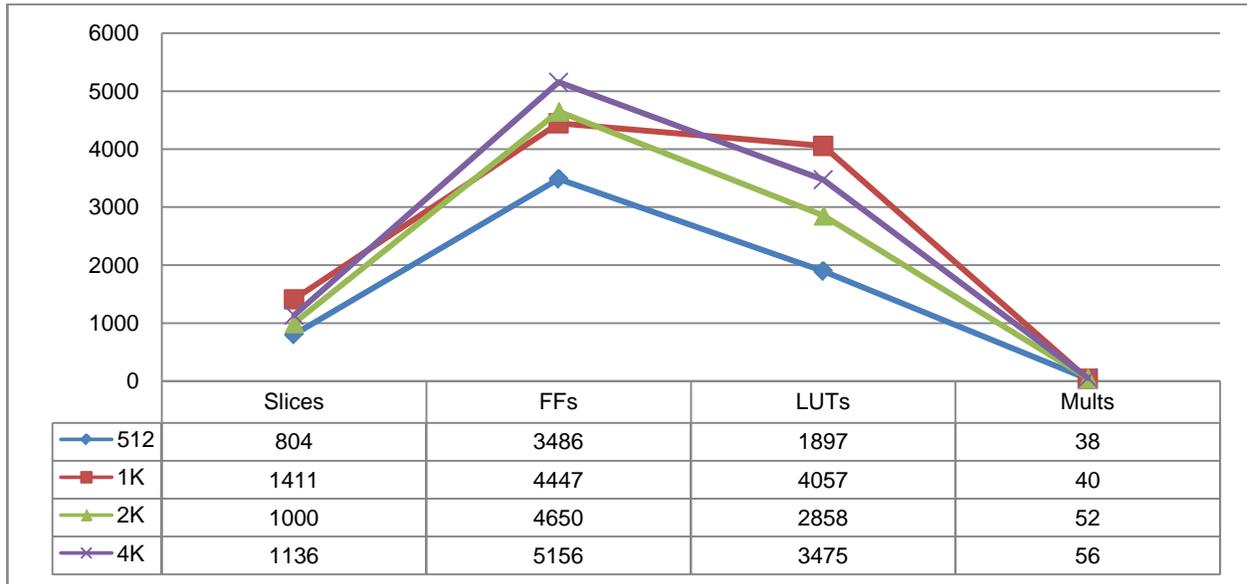


Figure 2.4-3: Resource utilization statistics for 512,1k, 2K, 4K point FFT when the core is synthesized for performance optimization

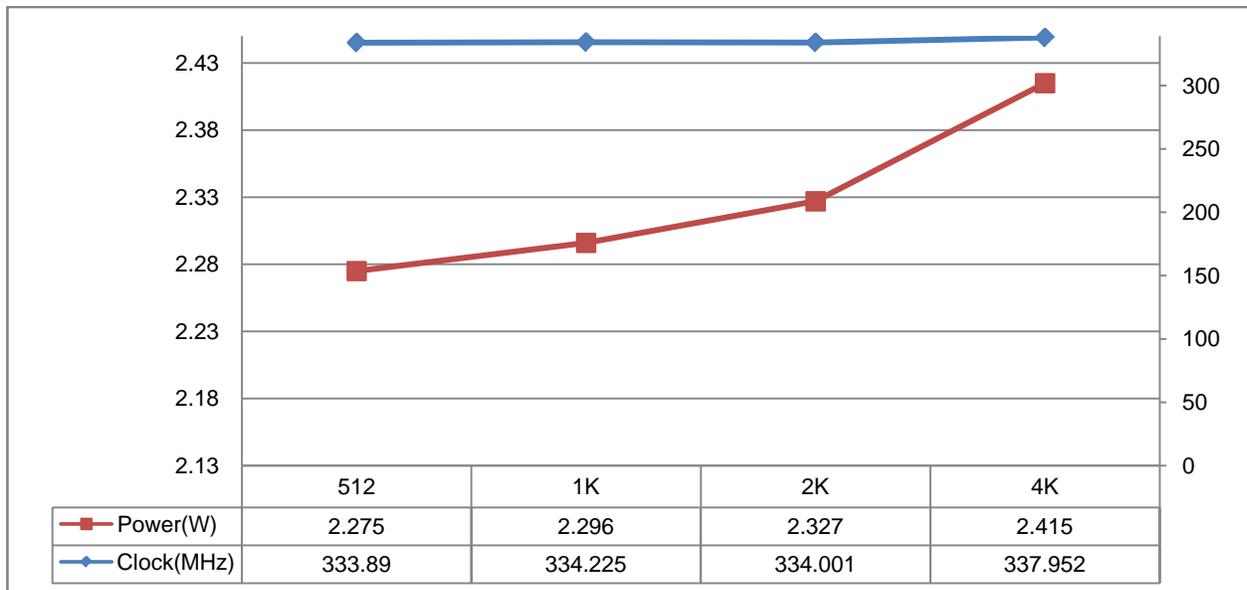


Figure 2.4-4: Power Consumption and Maximum achievable clock frequency statistics for 512,1k, 2K, 4K point FFT when the core is synthesized for performance optimization

## 3.1 Mapping FFT on PC

---

The Cooley-Tukey FFT algorithm was implemented on a Sony Vaio laptop, the basic specifications of which are as follows:

**Processor:** Intel Core2 Duo Penryn T6600

**Processor Clock:** 2.20 GHz

**Memory:** 4GB DDR3

**Front Side Bus:** 800MHz shared between the two processors

**Level 1 Cache:** 128KB dedicated

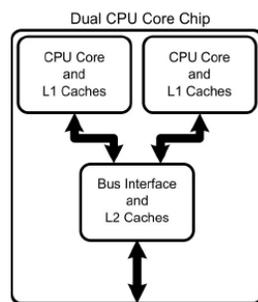
**Level2 Cache:** 2MB shared between the two cores

### 3.1-1 Penryn Processor Architecture

---

The Core 2 Duo, which was introduced on July 27 2006, is the direct successor of the Core Duo. Each core is based on the Pentium M micro architecture. Compared with the old Netburst architecture of the Pentium 4, the cores in the Core 2 Duo have shorter pipelines. Furthermore up to that point, any communication between cores had to take place via the FSB and main memory because each core would have its own, dedicated L1 and L2-caches. The Core 2 Duo architecture changed this: each core would still have access to its own, dedicated 32kB of L1-cache. The L2-cache (typically 2MB or 4MB) is shared between the cores, allowing for interaction between the cores there. Only one core at a time will have access to the main memory, as the FSB is still shared between the cores (see Figure 3.1.-1).

Each core has access to two 128-bit ALUs and one 128-bit FPU. A significant improvement has been made towards Core 2 Duo predecessors. Because of the three computational units per core, 128-bit SSE (SIMD) instructions can be performed in one clock cycle per core. This is particularly useful for e.g. matrix multiplications: a single 128-bit SSE operation can be seen as four smaller (32-bit) operations. So in fact, four FP computations can be done per clock cycle. Of course, Intel has integrated a multitude of smart technologies inside their processor for increased performance. These include caching algorithms, pipelining, dynamic scheduling, and energy saving technologies, all of which we will ignore in our discussion of the mapping.



*Figure 3.1-1: Basic memory hierarchy for the T660 Core 2 Duo processor with dedicated L1 caches and a shared L2 cache. Access to main memory is via FSB*

## 3.1-1 Penryn Strength and Weaknesses

---

This Penryn processor is well suited for desktop computing, where usually threaded programs run in a multi-tasking environment. Therefore, the ability to spread different tasks over different processors physically (as opposed to logically/time shared) is interesting, and the Penryn does just that. On two cores it can run a two-threaded program of which both threads can access the same variables easily all the time (because of the shared L2-cache).

However, in Penryn, the two cores share one FSB. In worst case scenario, when both cores would perform memory access and for both cores a cache miss would occur, the structure of the processor would introduce huge latencies.

## 3.2 Mapping Radix2 DIT FFT on Penryn

---

As explained in the previous section, the Intel Core architecture is able to perform four FP arithmetic operations per core per cycle. However, since the results have to be shared between cores, bus access plays a role in mapping the parallel Cooley-Tukey algorithm onto the Penryn processor. Let's consider a 4-point DIT FFT as an example.

In the first step of DIT FFT, the 4 point DFT is broken into 2 primitive DFTs of size 2. Each of these DFTs has to be scheduled on an available core. Equally dividing the workload for each core would assume one primitive DFT per core. A two input primitive DFT require one multiplication, one addition and one subtraction operation. This results in 6 operations. Assuming working with 32-bit precision and SSE (SIMD) instructions, thus having a Penryn core performing four operations simultaneously, implies that all operations can be done in a lower bound of 1 cycle. This calculation does not take into account any control overhead for organizing the data (getting data from cache into the right registers and writing back results). Furthermore, this assumes that all operations can be performed disjoint, which is not the case. For example, the addition and subtraction operations in one computation frame require the multiplication result to be available. Next, the results of these 2 primitive DFTs needs to be combined along with appropriate twiddle factors. This requires two butterfly operations with each butterfly requiring 1 addition, subtraction and multiplication. These 6 operations in total can be performed by the two core of Penryn in one clock cycle. Thus only two clock cycles are required to implement a 4 point FFT. Beware that our approach is very basic and simplistic. It does not take into account the processing needed for bit reversing the input stream for DIT butterflies. We are also neglecting any cache misses and accesses to the main memory.

## References

---

[1] Article 10: Fast Fourier transforms — FFT

<http://www.librow.com/articles/article-10>

[2] Cooley–Tukey FFT algorithm

[http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey\\_FFT\\_algorithm](http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm)

[3] Butterfly diagram

[http://en.wikipedia.org/wiki/Butterfly\\_diagram](http://en.wikipedia.org/wiki/Butterfly_diagram)

[4] Kee, Hojin, Newton Petersen, Jacob Kornerup, and Shuvra S. Bhattacharyya. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing. Las Vegas, Nevada,; n.p., 2008. Web. 11 Nov. 2010.

[5] Rabiner, L.R. and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, New Jersey, (1975).

[6] Xilinx documentation DS260: LogiCORE IP Fast Fourier Transform v7.1

[7] A.F.M. van Herk, S. van Loon, Mapping parallel FFT on different computer architectures

[8] Sabih H. Gerez, Hardware Implementations of the Fast Fourier Transform (FFT)