International Journal of Advanced Science and Technology
Vol.61, (2013), pp.53-62
http://dx.doi.org/10.14257/ijast.2013.61.06

Design and Simulation of FFT Processor Using Radix-4 Algorithm Using FPGA

N. Amarnath Reddy, D. Srinivasa Rao and J. Venkata Suman

GMR Institute of Technology
Asst. Prof. Dept. of ECE, GMRIT, RAJAM

amar.nagu02@gmail.com, srinivasarao.d@gmrit.org, venkatasuman.j@gmrit.org

Abstract

A parallel and pipelined Fast Fourier Transform (FFT) processor for use in the Orthogonal Frequency division Multiplexer (OFDM) and WLAN, unlike being stored in the traditional ROM. The twiddle factors in our pipelined FFT processor can be accessed directly. A novel simple address mapping scheme and the modified radix 4 FFT also proposed. FPGA was majorly used to develop the ASIC IC's to which was implemented. Here we simulated and synthesized the 256- point FFT with radix-4 using VHDL coding and 64 point FFT Hardware implementation we designed code using System C. Finally, the pipelined 256-point FFT processor can be completely implemented within 19.103ns.

Keywords- *FFT, DIT, Radix- 4, FPGA*

1. INTRODUCTION

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT) and it's inverse. A Fourier transform converts time (or space) to frequency and vice versa; FFT rapidly computes such transformations. As a result, fast Fourier transforms are widely used for many applications in engineering, science, and mathematics. The basic ideas were popularized in 1965, but some FFTs had been previously known as early as 1805. Fast Fourier transforms have been described as "the most important numerical algorithm of our lifetime".

There are many different FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory; this article gives an overview of the available techniques and some of their general properties, while the specific algorithms are described in subsidiary articles linked below.

The discrete Fourier transform is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields (see DFT for properties and applications of the transform) but computing it directly from the definition is often too slow to be practical. An FFT is a way to compute the same result more quickly: computing the DFT of N points in the naive way, using the definition, takes $O(N^2)$ arithmetical operations, while a FFT can compute the same DFT in only O(N log N) operations. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. In practice, the computation time can be reduced by several orders of magnitude in such cases, and the improvement is roughly proportional to N / log(N). This huge improvement made the calculation of the DFT practical; FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers.

The best-known FFT [2] algorithms depend upon the factorization of N, but there are FFTs with O(N log N) complexity for all N, even for prime N. Many FFT algorithms only

depend on the fact that $e^{-\frac{2\pi}{N}}$ is an N-the primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.
An FFT computes the DFT [5] and produces exactly the same result as evaluating the DFT definition directly; the only difference is that an FFT is much faster. Let $x_0 \ldots x_{N-1}$ be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2j\pi k\frac{n}{N}} \qquad k = 0 \ldots\ldots N-1 \quad \ldots\ldots\ldots\text{(1)}$$

Evaluating this definition directly requires $O(N^2)$ operations: there are N outputs X_k, and each output requires a sum of N terms. An FFT is any method to compute the same results in O(N log N) operations. More precisely, all known FFT algorithms require Θ(N log N) operations (technically, O only denotes an upper bound), although there is no known proof that a lower complexity score is impossible.(Johnson and Frigo, 2007)

To illustrate the savings of an FFT, consider the count of complex multiplications and additions. Evaluating the DFT's sums directly involves N^2 complex multiplications and N(N−1) complex additions [of which O(N) operations can be saved by eliminating trivial operations such as multiplications by 1]. The well-known radix-2 Cooley–Turkey algorithm, for N a power of 2, can compute the same result with only $(N/2)\log_2(N)$ complex multiplications (again, ignoring simplifications of multiplications by 1 and similar) and $N\log_2(N)$ complex[12] additions. In practice, actual performance on modern computers is usually dominated by factors other than the speed of arithmetic operations and the analysis is a complicated subject [4] (see *e.g.*, Frigo & Johnson, 2005), but the overall improvement from $O(N^2)$ to O(N log N) remains.

The naive implementation of the N-point digital Fourier transform involves calculating the scalar product of the sample buffer (treated as an N-dimensional vector) with N separate basis vectors. Since each scalar product involves N multiplications and N additions, the total time is proportional to N^2 (in other words, it's an $O(N^2)$ algorithm). However, it turns out that by cleverly re-arranging these operations, one can optimize the algorithm down to O(N log(N)), which for large N makes a huge difference. The optimized version of the algorithm is called the fast Fourier transform, or the FFT.

The standard strategy to speed up an algorithm is to divide and conquer. We have to find some way to group the terms in the equation

$$V[k] = \sum_{n=0....N-1} W_N^{kn} v[n]$$

Let's see what happens when we separate odd ns from even ns (from now on, let's assume that N is even):

$$\begin{aligned} V[k] &= \Sigma_n \ \text{even}\, W_N^{kn} v[n] + \Sigma_n \ \text{odd}\, W_N^{kn} v[n] \\ &= \Sigma_{r=0....N/2-1} W_N^{k(2r)} V[2r] + \Sigma_{r=0....N/2-1} W_N^{k(2r+1)} V[2r+1] \\ &= \Sigma_{r=0....N/2-1} W_N^{k(2r)} V[2r] + W_N^k \Sigma_{r=0....N/2-1} W_N^{k(2r)} V[2r+1] \\ &= \left(\Sigma_{r=0....N/2-1} W_{N/2}^{k(r)} V[2r]\right) + W_N^k \left(\Sigma_{r=0....N/2-1} W_{N/2}^{k(r)} V[2r+1]\right) \end{aligned}$$

Where we have used one crucial identity:

$$W_N^{k(2r)} = e^{-2\pi i*2kr/n}$$

$$= e^{-2\pi i * 2kr/(n/2)} = W_{N/2}^{kr}$$

Notice an interesting thing: the two sums are nothing else but N/2-point Fourier transforms of, respectively, the even subset and the odd subset of samples. Terms with k greater or equal N/2 can be reduced using another identity:

$$W_{N/2}^{m+n/2} = W_{N/2}^{m} W_{N/2}^{n/2} = W_{N/2}^{m},\ W_{m}^{m} = e^{-2\pi i} = \cos(-2\pi) + i \quad \sin(-2\pi) = 1$$

If we start with N that is a power of 2, we can apply this subdivision recursively until we get down to 2-point transforms. We can also go backwards, starting with the 2-point transform:

$$V[k] = W_2^{0*k} y[0] + W_2^{1*k} v[1], \qquad k = 0{,}1$$

The two components are:

$$V[0] = W_2^0 V[0] + W_2^0 V[1] = v[0] + W_2^0 V[1], V[1] = W_2^0 V[0] + W_2^1 V[1] = v[0] + W_2^1 V[1]$$

We can represent the two equations for the components of the 2-point transform graphically using the, so called, butterfly

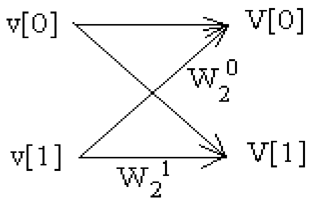


Figure 1. Butterfly Calculation

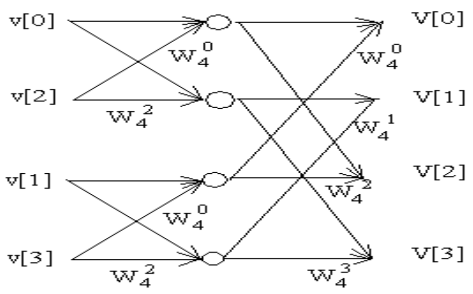


Figure 1.1. 4-point Fourier Transform

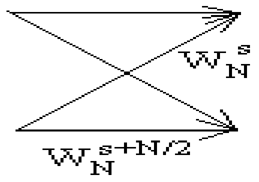


Figure 1.2. Generic Butterfly Graph

Furthermore, using the divide and conquer strategy, a 4-point transform can be reduced to two 2-point transforms: one for even elements, one for odd elements. The odd one will be multiplied by W_4^k Diagrammatically. This can be represented as two levels of butterflies. Notice that using the identity $W_{N/2}^n = W_N^{2n}$, we can always express all the multipliers as powers of the same W_N (in this case we choose N=4).

2. RADIX-4 FFT

The decimation-in-time (DIT) radix-4 FFT recursively partitions a DFT [5] into four quarter-length DFTs of groups of every fourth time sample. The outputs of these shorter FFTs are reused to compute many outputs, thus greatly reducing the total computational cost. The radix-4 decimation-in-frequency FFT groups every fourth output sample into shorter-length DFTs to save computations. The radix-4 FFTs require only 75% as many complex multiplies [6] as the radix-2 FFTs. The radix-4 decimation-in-time and decimation-in-frequency fast Fourier transforms (FFTs) gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT[23] frequency outputs. The radix-4 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation into four parts. The DFT[22] sums over all groups of the every fourth discrete-time index $n=[0,4,8,\ldots,N-4]$, $n=[1,5,9,\ldots,N-3]$, $n=[2,6,10,\ldots,N-2]$ and $n=[3,7,11,\ldots,N-1]$ as in Equation 1. (This works out only when the FFT length is a multiple of four.) Just as in the radix-2 decimation-in-time[22] FFT, further mathematical manipulation shows that the length-N DFT can be computed as the sum of the outputs of four length-N4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors $W_N^k = e^{-(i2\pi kN)}, W_N^{2k}$, and W_N^{3k}

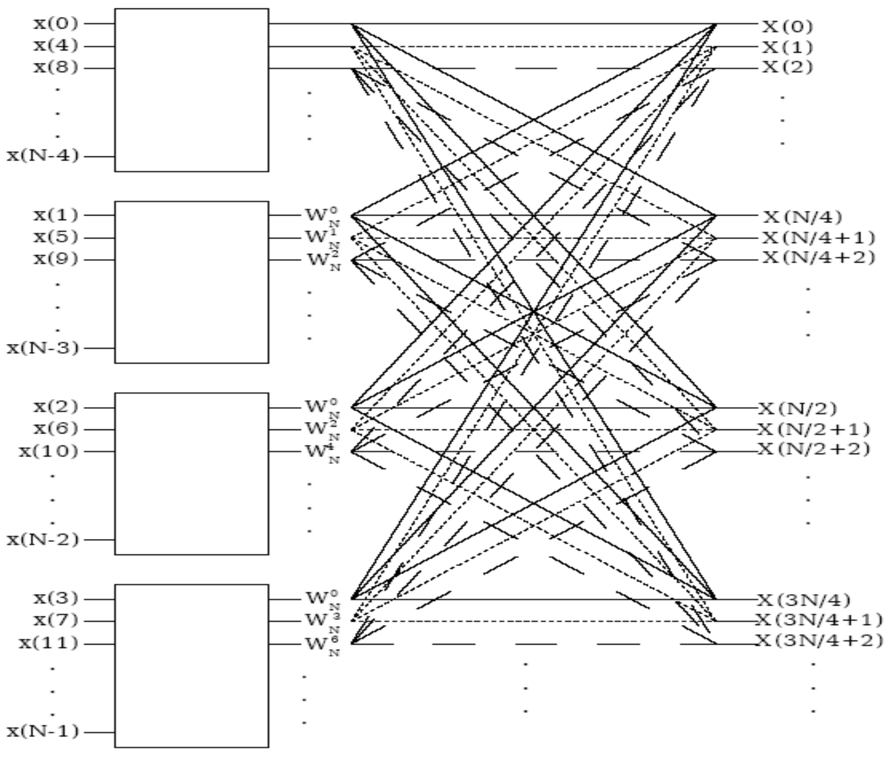


Figure 2. Radix-4 DIT Structure

This is called decimation in time because the time samples are rearranged in alternating groups and a radix-4 algorithm because there are four groups. Figure 1 graphically illustrates this form of the DFT computation. It is this reuse that gives the radix-4 FFT its efficiency. The computations involved with each group of four frequency samples constitute the radix-4 butterfly, which is shown in Figure 2. Through further rearrangement, it can be shown that this computation can be simplified to three twiddle-factor multiplies and a length-4 DFT! The theory of multi-dimensional index maps shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFT [25] with twiddle-factor multiplications in between.

3. FPGA Implementation

The Field Programmable Gate Array is majorly used for generation ASIC IC's to the computations. They offer more speed in execution process. SO, for generation ASIC IC's FPGA's [21] are majorly used. The 64 FFT with radix 4 is simulated and synthesized as well as implemented on the FPGA of below configuration.

Table 3.1. Configuration of FPGA

Property Name	Value
Family	Spartan 3
Device	XC3S200
Package	TQG144
Speed Grade	-4

4. Simulation Results:

The RTL view of the butterfly structure obtained after the simulation of the 256-point FFT block, Decimation in time domain is shown next and also the internal architecture of the butterfly block is shown.

BUTTERFLY16
BUTTERFLY16

Figure 4.1. RTL View of a Butterfly Component Used In 256-Point FFT

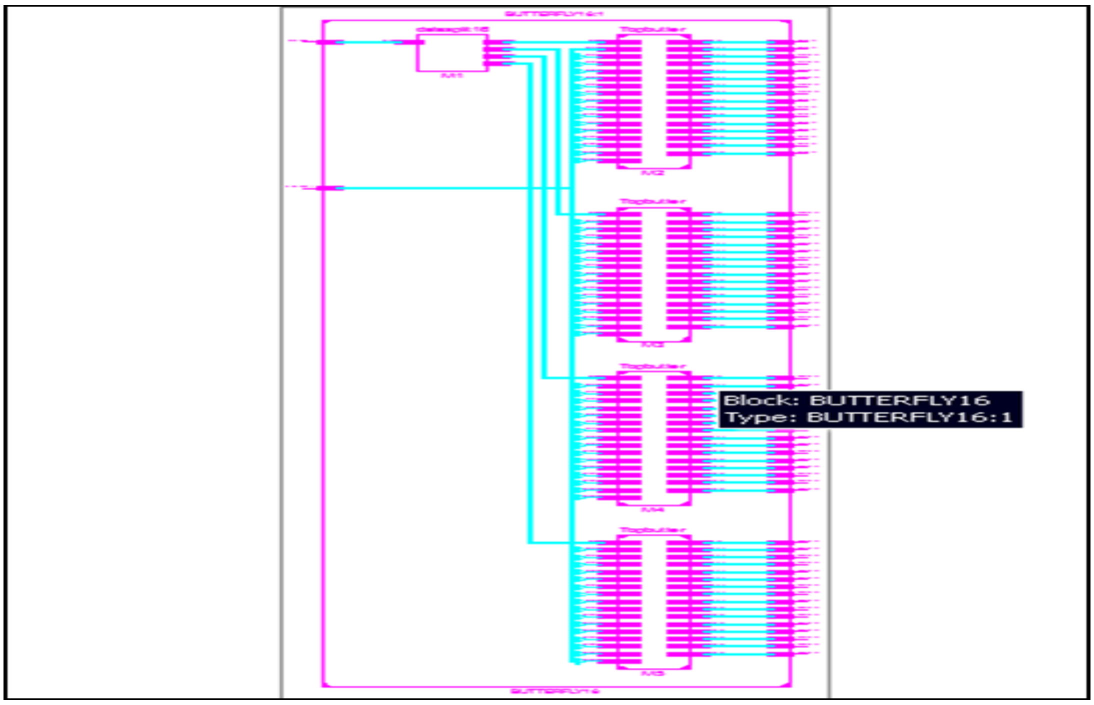
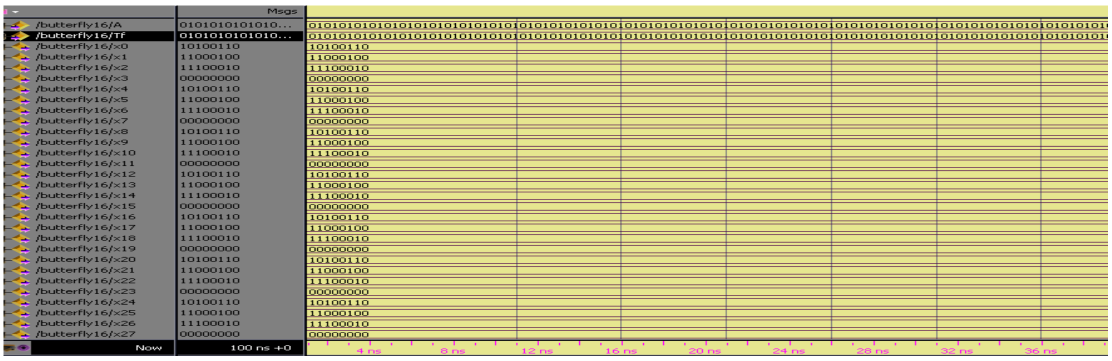
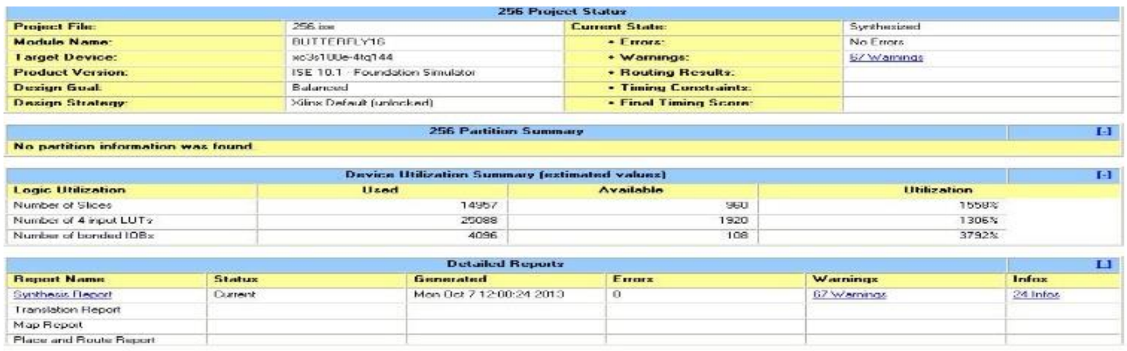


Figure 4.2. Internal Architecture of the Butterfly Component



```
                        Msgs
/butterfly16/A          0101010101010...   0101010101010101010101010101010101010...
/butterfly16/Tf         0101010101010...   0101010101010101010101010101010101010...
/butterfly16/x0         10100110           10100110
/butterfly16/x1         11000100           11000100
/butterfly16/x2         11100010           11100010
/butterfly16/x3         00000000           00000000
/butterfly16/x4         10100110           10100110
/butterfly16/x5         11000100           11000100
/butterfly16/x6         11100010           11100010
/butterfly16/x7         00000000           00000000
/butterfly16/x8         10100110           10100110
/butterfly16/x9         11000100           11000100
/butterfly16/x10        11100010           11100010
/butterfly16/x11        00000000           00000000
/butterfly16/x12        10100110           10100110
/butterfly16/x13        11000100           11000100
/butterfly16/x14        11100010           11100010
/butterfly16/x15        00000000           00000000
/butterfly16/x16        10100110           10100110
/butterfly16/x17        11000100           11000100
/butterfly16/x18        11100010           11100010
/butterfly16/x19        00000000           00000000
/butterfly16/x20        10100110           10100110
/butterfly16/x21        11000100           11000100
/butterfly16/x22        11100010           11100010
/butterfly16/x23        00000000           00000000
/butterfly16/x24        10100110           10100110
/butterfly16/x25        11000100           11000100
/butterfly16/x26        11100010           11100010
/butterfly16/x27        00000000           00000000
Now                     100 ns +0          4 ns  8 ns  12 ns  16 ns  20 ns  24 ns  28 ns  32 ns  36 ns
```

Figure 4.3. Simulation result of 256 FFT



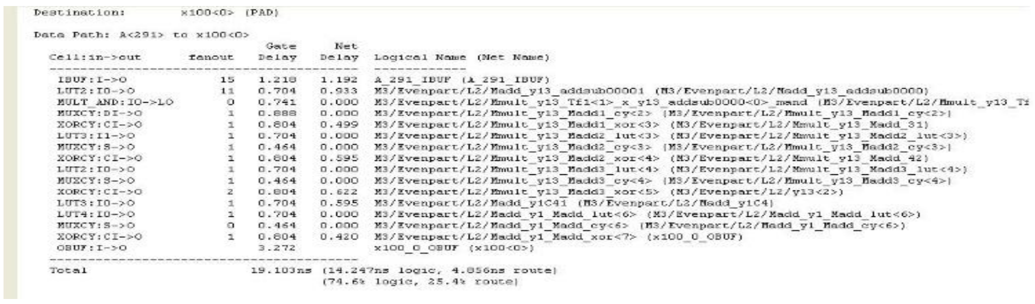
256 Project Status			
Project File:	256.ise	**Current State:**	Synthesized
Module Name:	BUTTERFLY16	• **Errors:**	No Errors
Target Device:	xc3s100e-4tq144	• **Warnings:**	67 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• **Routing Results:**	
Design Goal:	Balanced	• **Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	• **Final Timing Score:**	

256 Partition Summary	[-]
No partition information was found.	

Device Utilization Summary (estimated values)				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	
Number of Slices	14957	960	1558%	
Number of 4 input LUTs	25088	1920	1306%	
Number of bonded IOBs	4096	108	3792%	

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Mon Oct 7 12:00:24 2013	0	67 Warnings	24 Infos
Translation Report					
Map Report					
Place and Route Report					

Figure 4.4. Synthesis Report

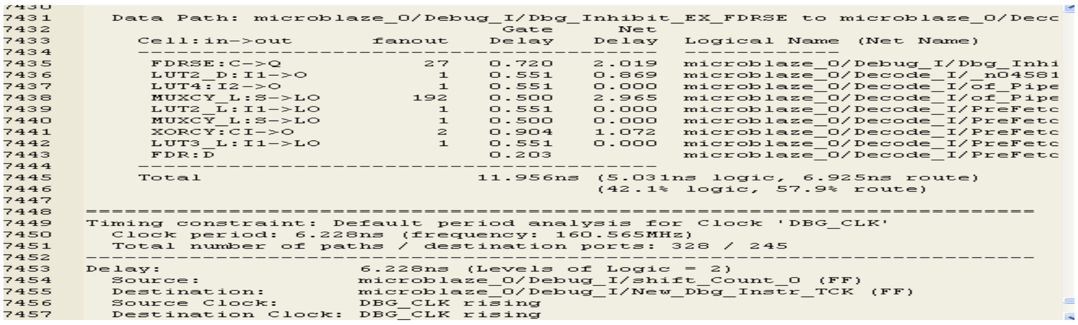


```
Destination:        x100<0> (PAD)

Data Path: A<291> to x100<0>
                                Gate     Net
  Cell:in->out      fanout     Delay    Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O           15     1.218    1.192  A_291_IBUF (A_291_IBUF)
   LUT2:I0->O          11     0.704    0.933  M3/Evenpart/L2/Madd_y13_addsub00001 (M3/Evenpart/L2/Madd_y13_addsub0000)
   MULT_AND:I0->LO      0     0.741    0.000  M3/Evenpart/L2/Mmult_y13_Tf1<1>_x_y13_addsub0000<0>_mand (M3/Evenpart/L2/Mmult_y13_Tf
   MUXCY:DI->O          1     0.888    0.000  M3/Evenpart/L2/Mmult_y13_Madd1_cy<2> (M3/Evenpart/L2/Mmult_y13_Madd1_cy<2>)
   XORCY:CI->O          1     0.804    0.499  M3/Evenpart/L2/Mmult_y13_Madd1_xor<3> (M3/Evenpart/L2/Mmult_y13_Madd_31)
   LUT3:I1->O           1     0.704    0.000  M3/Evenpart/L2/Mmult_y13_Madd2_lut<3> (M3/Evenpart/L2/Mmult_y13_Madd2_lut<3>)
   MUXCY:S->O           1     0.464    0.000  M3/Evenpart/L2/Mmult_y13_Madd2_cy<3> (M3/Evenpart/L2/Mmult_y13_Madd2_cy<3>)
   XORCY:CI->O          1     0.804    0.595  M3/Evenpart/L2/Mmult_y13_Madd2_xor<4> (M3/Evenpart/L2/Mmult_y13_Madd_42)
   LUT2:I0->O           1     0.704    0.000  M3/Evenpart/L2/Mmult_y13_Madd3_lut<4> (M3/Evenpart/L2/Mmult_y13_Madd3_lut<4>)
   MUXCY:S->O           1     0.464    0.000  M3/Evenpart/L2/Mmult_y13_Madd3_cy<4> (M3/Evenpart/L2/Mmult_y13_Madd3_cy<4>)
   XORCY:CI->O          2     0.804    0.622  M3/Evenpart/L2/Mmult_y13_Madd3_xor<5> (M3/Evenpart/L2/y13<2>)
   LUT3:I0->O           1     0.704    0.595  M3/Evenpart/L2/Madd_y1C41 (M3/Evenpart/L2/Madd_y1C4)
   LUT4:I0->O           1     0.704    0.000  M3/Evenpart/L2/Madd_y1_Madd_lut<6> (M3/Evenpart/L2/Madd_y1_Madd_lut<6>)
   MUXCY:S->O           0     0.464    0.000  M3/Evenpart/L2/Madd_y1_Madd_cy<6> (M3/Evenpart/L2/Madd_y1_Madd_cy<6>)
   XORCY:CI->O          1     0.804    0.420  M3/Evenpart/L2/Madd_y1_Madd_xor<7> (x100_0_OBUF)
   OBUF:I->O                  3.272           x100_0_OBUF (x100<0>)
  ----------------------------------------
  Total                      19.103ns (14.247ns logic, 4.856ns route)
                                      (74.6% logic, 25.4% route)
```

Figure 4.5. Timing Report of 256 FFT

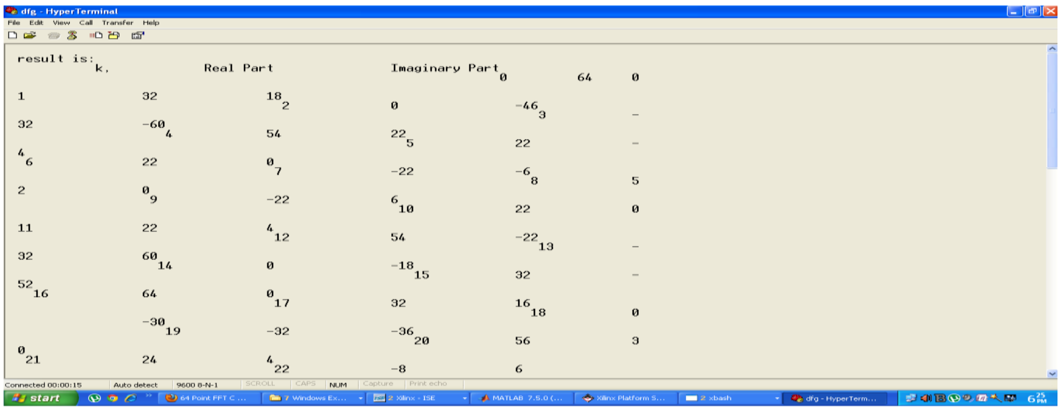
Hardware implementation was through system C coding and its results are as follows



```
  Data Path: microblaze_0/Debug_I/Dbg_Inhibit_EX_FDRSE to microblaze_0/Decc
                                       Gate     Net
    Cell:in->out           fanout     Delay    Delay  Logical Name (Net Name)
    ----------------------------------------------  ------------
     FDRSE:C->Q               27     0.720    2.019  microblaze_0/Debug_I/Dbg_Inhi
     LUT2_D:I1->O              1     0.551    0.869  microblaze_0/Decode_I/_n04581
     LUT4:I2->O                1     0.551    0.000  microblaze_0/Decode_I/of_Pipe
     MUXCY_L:S->LO           192     0.500    2.965  microblaze_0/Decode_I/of_Pipe
     LUT2_L:I1->LO             1     0.551    0.000  microblaze_0/Decode_I/PreFetc
     MUXCY_L:S->LO             1     0.500    0.000  microblaze_0/Decode_I/PreFetc
     XORCY:CI->O               2     0.904    1.072  microblaze_0/Decode_I/PreFetc
     LUT3_L:I1->LO             1     0.551    0.000  microblaze_0/Decode_I/PreFetc
     FDR:D                            0.203          microblaze_0/Decode_I/PreFetc
    ----------------------------------------------
    Total                           11.956ns (5.031ns logic, 6.925ns route)
                                             (42.1% logic, 57.9% route)

================================================================================
Timing constraint: Default period analysis for Clock 'DBG_CLK'
  Clock period: 6.228ns (frequency: 160.565MHz)
  Total number of paths / destination ports: 328 / 245
--------------------------------------------------------------------------------
Delay:                  6.228ns (Levels of Logic = 2)
  Source:               microblaze_0/Debug_I/shift_Count_0 (FF)
  Destination:          microblaze_0/Debug_I/New_Dbg_Instr_TCK (FF)
  Source Clock:         DBG_CLK rising
  Destination Clock:    DBG_CLK rising
```

Figure 4.6. Timing Report on Hardware



```
result is:
        k,            Real Part               Imaginary Part
                                                             0          64    0

1            32          18                0             -46            
                           2                                  3         -
32           -60         54                22            22             -
                4                             5
4
 6           22          0                 -22           -6             5
                          7                                 8
2            0           -22               6             22             0
              9                             10
11           22          4                 54            -22            -
                          12                                13
32           60          0                 -18           32             -
               14                             15
52
  16         64          0                 32            16             0
                          17                                18
             -30         -32               -36           56             3
                19                            20
0
 21          24          4                 -8            6
                          22
```

Figure 4.7. Result of 64-point FFT on Hardware Implementation

5. Conclusion

In this project it is shown that a baseband ASIC can be fast and at the same time flexible with a low power consumption. The term fast do not refer to extreme clock frequencies but to the fact that no part of the designs needs more than one clock cycle to process a sample once the pipe is filled. Hence, the design does not need to be clocked any faster than the requested bandwidth and compared to modern CMOS technology this is a low number, in the order of 5-100 MHz There are several advantages with a low clock frequency, firs it is possible use a low power/low speed cell library with low static leakage current and secondly, it is easier to create a clock tree. Since flexibility is achieved with independent modules, where the operation mode decides if a module should be used or not. The unused modules are not clocked and hence only consume static leakage power and when you implement 256 point FFT in FPGA than better results are out.

References

[1] P. D. Welch, "A fixed-point fast Fourier transform error analysis", IEEE Trans. Audio Electroacoustics, 10.1109/TAU.1969.1162035, vol. 17, no. 2, **(1969)**, pp. 151–157.

[2] N. Brenner and C. Rader, "A New Principle for Fast Fourier Transformation", IEEE Acoustics, Speech & Signal Processing, vol. 24, no. 3, pp. 264-266.

[3] P. Duhamel, "Algorithms meeting the lower bounds on the multiplicative complexity of length-2^n DFTs and their connection with practical algorithms", IEEE Trans. Acoust. Speech. Sig. Proc., 38 (9): 1504–151:10.1109/29.60070, **(1990)**.

[4] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3", Proceedings of the IEEE 93, **(2005)**, pp. 216–231.

[5] S. B. Weinstein and P. M. Ebert, "Data transmission by frequency division multiplexing using the discrete Fourier transform", IEEE Transactions on Communications, vol. 19, **(1971)** October, pp. 628–634.

[6] A. Peled and A. Ruiz, "Frequency domain data Transmission using reduced computational complexity algorithms", Int. Conf. Acoustic, Speech, Signal Processing, Denver, CO, **(1980)**, pp. 964-967.

[7] L. J. Cimini, "Analysis and Simulation of a Digital Mobile Channel Using Orthogonal Frequency Division Multiplexing", IEEE Transactions on Communications, vol. 33, **(1985)** July, pp. 665-675.

[8] ETSI TS 101 475, "Broadband Radio Access Networks (BRAN); HIPERLAN Type 2 Physical (PHY) layer, v1.1.1," 2000, http://portal.etsi.org/bran/.

[9] IEEEstd 802.11a, "High-speed Physical Layer in 5 GHz Band", http://ieee802.org/, **(1999)**.

[10] J. Bingham, "Multicarrier Modulation for Data Transmission: An Idea Whose Time Has Come", IEEE Communications Magazine, vol. 8, **(1990)** May, pp. 5-14.

[11] O. Edfors, M. Sandell, J. van de Beek, D. Landström and F. Sjöberg, "An introduction to orthogonal frequency-division multiplexing", TULEA 1996:16, Div. of Signal Processing, Luleå, Tech. Rep., a University of Technology, Luleå, **(1996)**.

[12] J. W. Cooley and J. W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," Math. Compute, vol. 19, **(1965)**, pp. 297-301.

[13] R. Grunheid, E. Bolinth and H. Rohling, "A blockwise loading algorithm for the adaptive modulation technique in OFDM systems", Proc. of Vehicular Technology Conference, VTC 2001 Fall, Atlantic City, NJ, USA, **(2001)** October 7-11, pp. 948-951.

[14] J. G. Proakis, "Digital Communications", McGraw-Hill, **(2001)**.

[15] M. Russel and G. Stuber, "Interchannel interference analysis of OFDM in a mobile environment", Proc. IEEE Vehic. Technol. Conf., vol. 2, Chicago, IL, **(1995)**, pp. 820-824.

[16] N. Petersson, "Peak and power reduction in multicarrier systems", licentiate thesis, Lund University, Sweden, **(2002)**.

[17] S. Johansson, "ASIC Implementation of an OFDM Synchronization Algorithm", Licentiate Thesis, Lund University, Sweden, **(2000)**.

[18] R. Morrison, L. J. Cimini and S. K. Wilson, "On the Use of a Cyclic Extension in OFDM", Proc. of Vehicular Technology Conference", VTC 2001 Fall, vol. 2, Atlantic City, NJ, USA, **(2001)** October 7-11, pp. 664–668.

[19] J. Rabaey, A. Chandrakasan and B. Nikolic, "Digital Integrated Circuits, a Design Perspective", Prentice-Hall, **(2003)**.

[20] K. Parhi, "VLSI Digital Signal Processing Systems", New York, NY, USA: John Wiley & Sons, **(1999)**.

[21] H. He and H. Guo, "The Realization of FFT AlgorithmBased on FPGA Co-processor", Second International Symposium on Intelligent Information Technology Application, vol. 3, **(2008)** December, pp. 239-243.

[22] J. G. Proakis and D. G. Manolakis, "Digital Signal Processing", Prentice-Hall, **(1996)**.

[23] S. He, "Concurrent VLSI Architecture for DFT Computing and Algorithms for Multi-output Logic Decomposition", Ph.D. dissertation, Lund University, **(1995)**.

[24] K. Nazifi and G. Hasson, "Industry's First RTL Power Optimization Feature Significantly Improves Power Compiler's Quality of Results", www.synopsys.com/news/pubs/rsvp/spr98/rsvp spr98 6.html, **(1998)**.

[25] W. Li and L. Wanhammar, "A Pipelined FFT Processor", IEEE Workshop on Signal Processing Systems, **(1999)**, pp. 654–662.

[26] S. Johansson, S. He, and P. Nilsson, "Word length Optimization of a Pipelined FFT Processor", Proc. of 42nd Midwest Symposium on Circuits and Systems, Las Cruces, NM, USA, **(1999)** August 8-11.

Authors

Nagu Amarnath Reddy Pursuing M.Tech (VLSI & ES) in GMRIT, Rajam, A.P, India and Received B.Tech (ECE) from Newton's institute of engineering under JNTUK. Major areas are interested in DSP and VLSI, ES.

D. Srinivasa RaO has completed B.Tech in Electronics and Communication Engineering from JNTU, Hyderabad. He completed M.E from Anna University with specialization in Communication Systems. Presently he is working as an Assistant Professor in Department of ECE, GMR Institute of Technology, Rajam. Major research areas include wireless communications, Information theory and coding. He is Life Member of ISTE.

Jami Venkata Suman has completed B.Tech from Tontadarya College of Engineering, under VTU, Belgaum, Karnataka. He Received Master of Technology in VLSI System Design from Annamacharya Institute of Technology and Sciences, Rajampet, under JNTU, Hyderabad, A.P and Master of Business Administration in HRM and MRKT from A.U, Visakhapatnam, A.P. He is currently working as an Assistant Professor in the Department of Electronics and Communication Engineering at GMR Institute of Technology, Rajam, A.P .Major Research areas include Radar, Signal Processing and VLSI. Life Member of ISTE.