Design and Implementation of Fast Fourier Transform Algorithm in FPGA

Adriana Bonilla R., Roberto J. Vega L., Karlo G. Lenzi e Luís G. P. Meloni

***Abstract*— This paper shows a design and implementation of a radix-4 FFT in FPGA using a Xilinx Spartan-6. The decimation in time equations are reviewed and in sequence several FPGA modules are presented according to algorithm architecture looking for optimization in execution time and occupied device area. Several tests were performed in order to validate the algorithm performance, FFT functionality, and time performance analysis. The proposed architecture is of low cost and very efficient for FFT computation.**

Keywords—FFT, VHDL, FPGA, Radix-4, Dragonfly.

***Resumo*—Este trabalho apresenta o projeto de um algoritmo de FFT radical-4 para FPGA usando-se a Spartan-6 da Xilinx. É feita uma revisão do algoritmo de decimação no tempo, e na sequência, os diversos módulos da FPGA são apresentados conforme a arquitetura do algoritmo, procurando-se a otimização do tempo de execução e da área ocupada no dispositivo. Vários testes foram realizados de forma a validar o desempenho do algoritmo, a operação da FFT e análise de desempenho de tempo de execução. A arquitetura proposta é de baixo custo e muito eficiente para o cálculo da FFT.**

Palavras-Chave—FFT, VHDL, FPGA, Radical-4, libélula.

I. INTRODUCTION

There are several methodologies and techniques that already offer hardware and software solutions for computing fast Fourier transform (FFT), which have advantages for specific applications. These solutions are developed for running in several platforms, such as GPU, DSP, FPGA and ASIC and they are usually described in C/C++ language or HDL. Implementation intended for reconfigurable logic is usually described in HDL, such as VHDL or Verilog.

Different FFT algorithms have been proposed to exploit certain signal properties to improve the trade-off between computation time and hardware requirements. Radix-4 based algorithms improve computation time by a factor of two, compared with radix-2 based algorithms, increasing hardware requirements by the same factor.

Considering that low-cost, high-density reconfigurable devices are already available, an optimized price/performance FPGA implementation of the 1024-point radix-4 FFT is feasible.

II. RADIX-4 FFT

The N-point discrete Fourier transform (DFT) is defined by equation [1]

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

where

$$W_N^{kn} = e^{-2\pi j\left(\frac{kn}{N}\right)} = \cos\left(2\pi\frac{kn}{N}\right) - j\sin\left(2\pi\frac{kn}{N}\right) \qquad (2)$$

The DFT calculation demands a complex implementation (requires N^2 complex multiplications and N(N-1) complex additions), so we have to find a more efficient way to perform this calculation. The Fast Fourier Transformation (FFT) was proven to be a faster and more efficient algorithm to compute Fourier transform. We use the decimation in Frequency (DIF) radix-4, which is the most used to calculate the FFT because of its reduce computational complexity.

The radix-4 DIF FFT divides an N-point discrete Fourier transform (DFT) into four N/4-point DFTs, then into 16 N/16-point DFTs, and so on. In the radix-2 DIF FFT, the DFT equation is expressed as the sum of two calculations. One calculation sum for the first half and one calculation sum for the second half of the input sequence [1]. Similarly, the radix-4 DIF fast Fourier transform (FFT) expresses the DFT equation as four summations, and then divides it into four equations, each of which computes every fourth output sample.

The following equations illustrate radix-4 decimation in frequency. Equation (1) can be written as follows [2]:

$$X(k) = \sum_{n=0}^{N/4-1} x(n)W_N^{nk} + \sum_{n=3N/4}^{2N/4-1} x(n)W_N^{nk} + \sum_{n=2N/4}^{3N/4-1} x(n)W_N^{nk} + \sum_{n=3N/4}^{N-1} x(n)W_N^{nk} \qquad (3)$$

$$X(k) = \sum_{n=0}^{N/4-1} x(n)W_N^{nk} + \sum_{n=0}^{N/4-1} x\left(n+N/4\right)W_N^{(n+N/4)k} + \sum_{n=2N/4}^{3N/4-1} x\left(n+2N/4\right)W_N^{(n+2N/4)k} + \sum_{n=3N/4}^{N-1} x\left(n+3N/4\right)W_N^{(n+3N/4)k} \qquad (4)$$

Adriana Bonilla R., Departamento de Engenharia de Petróleo, Roberto J. Vega L. e Luís G. P. Meloni, Departamento de Comunicações, Universidade Estadual de Campinas,
Karlo G. Lenzi, Centro de Pesquisa e Desenvolvimento (CPqD), Campinas-SP, Brasil, E-mails: adriana@dep.fem.unicamp.br, robertovega79@gmail.com, meloni@decom.fee.unicamp.br, klenzi@cpqd.com.

$$X(k) = \sum_{n=0}^{N/_4-1} \Big[x(n) + x\big(n + {}^{N}/_{4}\big) W_N^{({}^N/_4)k} + x\big(n + {}^{2N}/_{4}\big) W_N^{({}^{2N}/_4)k} + x\big(n + {}^{3N}/_{4}\big) W_N^{({}^{3N}/_4)k} \Big] W_N^{nk} \qquad (5)$$

The three twiddle factor coefficients can be expressed as follows:

$$W_N^{({}^N/_4)k} = \left[cos\left(\frac{\pi}{2}\right) - j\, sin\left(\frac{\pi}{2}\right)\right]^k = (-j)^k$$
$$W_N^{({}^{2N}/_4)k} = [cos(\pi) - j\, sin(\pi)]^k = (-1)^k$$
$$W_N^{({}^{3N}/_4)k} = \left[cos\left(\frac{3\pi}{2}\right) - j\, sin\left(\frac{3\pi}{2}\right)\right]^k = (j)^k \qquad (6)$$

To arrive at a four-point DFT decomposition, let:

$$\mathrm{W_N^4} = \mathrm{W_{N/_4}} \qquad (7)$$

Then the Equation (6) can then be written as four N/4-point DFTs, or

$$\mathrm{X(4k)} = \sum_{\mathrm{n=0}}^{\mathrm{N/_4-1}} \left[\mathrm{x(n) + x\big(n + {}^{N}/_{4}\big) + x\big(n + {}^{2N}/_{4}\big) + x\big(n + {}^{3N}/_{4}\big)} \right] \mathrm{W_{N/_4}^{nk}}$$

$$\mathrm{X(4k+1)} = \sum_{\mathrm{n=0}}^{\mathrm{N/_4-1}} \left[\mathrm{x(n) - j\,x\big(n + {}^{N}/_{4}\big) - x\big(n + {}^{2N}/_{4}\big) + j\,x\big(n + {}^{3N}/_{4}\big)} \right] \mathrm{W_{N/_4}^{nk} W_N^{n}}$$

$$\mathrm{X(4k+2)} = \sum_{\mathrm{n=0}}^{\mathrm{N/_4-1}} \left[\mathrm{x(n) - x\big(n + {}^{N}/_{4}\big) + x\big(n + {}^{2N}/_{4}\big) - x\big(n + {}^{3N}/_{4}\big)} \right] \mathrm{W_{N/_4}^{nk} W_N^{2n}}$$

$$\mathrm{X(4k+3)} = \sum_{\mathrm{n=0}}^{\mathrm{N/_4-1}} \left[\mathrm{x(n) + j\,x\big(n + {}^{N}/_{4}\big) - x\big(n + {}^{2N}/_{4}\big) - j\,x\big(n + {}^{3N}/_{4}\big)} \right] \mathrm{W_{N/_4}^{nk} W_N^{3n}}$$

$$\text{For } k = 0 \text{ to } {}^{N}/_{4} - 1 \qquad (8)$$

$\mathrm{X(4k), X(4k+1), X(4k+2), X(4k+3)}$ are N/4-point DFTs. Each of their N/4 points is a sum of four input samples $\mathrm{x(n), x\big(n + {}^{N}/_{4}\big), x\big(n + {}^{2N}/_{4}\big), x\big(n + {}^{3N}/_{4}\big)}$, each multiplied by either +1, -1, j, or -j. The sum is multiplied by a twiddle factor($\mathrm{W_N^0, W_N^n, W_N^{2n}, W_N^{3n}}$). Figure 1.

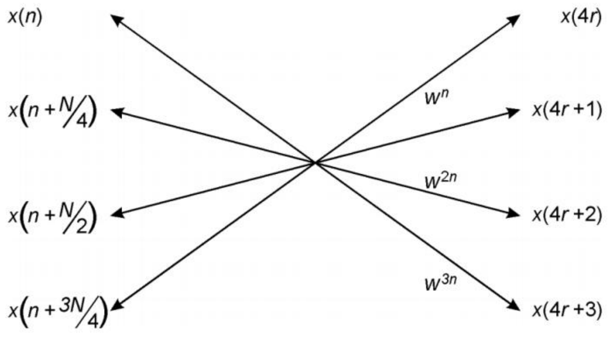

Fig. 1. Radix-4 DIF FFT Dragonfly

The four N/4-point DFTs together make up an N-point DFT. Each of these N/4-point DFTs is divided into four N/16-point DFTs. Each N/16 DFT is further divided into four N/64-point DFTs, and so on, until the final decimation produces four-point DFTs. The four-point DFT equation makes up the dragonfly calculation of the radix-4 FFT.

Let $g_0(n) = \mathrm{x(n) + x\big(n + {}^{N}/_{4}\big) + x\big(n + {}^{2N}/_{4}\big) + x\big(n + {}^{3N}/_{4}\big)}$
Then the equation (7) for N/4-point FFT can be written as

$$\mathrm{X(4k)} = \sum_{\mathrm{n=0}}^{\mathrm{N/_4-1}} [\,\mathrm{g_o(n)}]\, \mathrm{W_{N/_4}^{nk}}$$

Then

$$\mathrm{X(16K)} = \sum_{\mathrm{N=0}}^{\mathrm{N/_{16}-1}} \left[\mathrm{G_O(N) + G_O\big(N + {}^{N}/_{4}\big) + G_O\big(N + {}^{2N}/_{4}\big) + G_O\big(N + {}^{3N}/_{4}\big)} \right] \mathrm{W_{N/_{16}}^{NK}}$$

When using Radix-4 decomposition, the N-point FFT consists of log4 (N) stages, with each stage containing N/4 Radix-4 dragonflies. From the formulas we calculate the 1024-point FFT. That will be 5 stages where dragonflies will run for the 1024-points. This can be seen in the Fig. 2, just for 3 stages. [3]

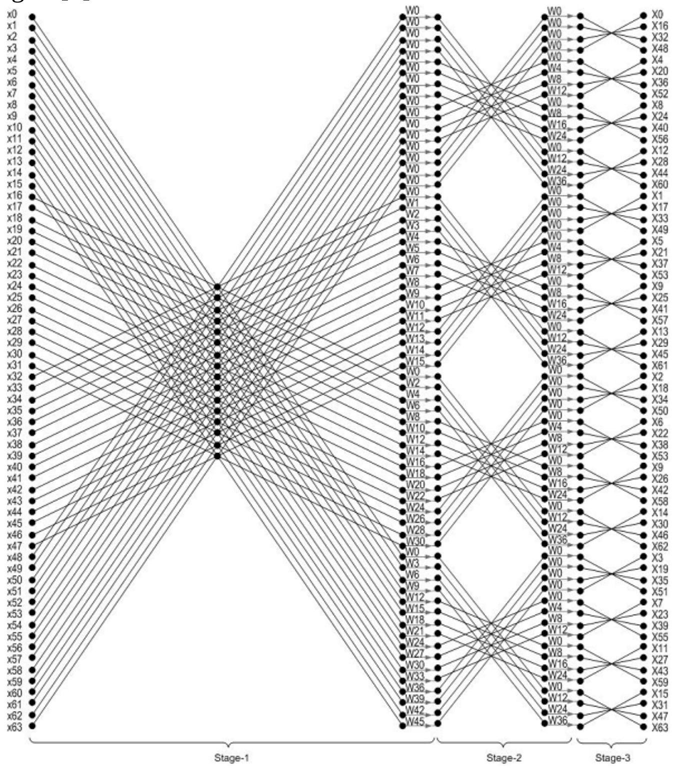

Fig. 2. Radix-4 64 Point FFT Structure

The points of the FFT Radix-2 algorithm are calculated using the following formula:

$$X(2k_2) = \sum_{n=0}^{N/_2-1} \left[\left(x(n_2) + (-1)^{k_1} x\big({}^{N}/_{2} + n_2\big) \right) W_N^{k_1 n_2} \right] W_{N/_2}^{k_2 n_2}$$

Compared with the Radix-2 algorithm, we will get a more complex algorithm but with less computational cost.

Taking a 1024-point sequence, radix-2 would require 40960 additions and 20480 multiplications. Radix-4 requires 30720 additions and only 5120 multiplications. The comparison between Radix-4 and Radix-2 implementation [4] is shown in Table I.

TABLE I. IMPLEMENTATION RESULTS ON SPARTAN-3 DEVICES.

Point Size	Algorithm	Slices	Max. Speed (MHz)	Latency (cycles)	Transform Time Cycles	Throughput (MS/s)
1024	Radix-2	4409	123.84	1041	1024	123.84
1024	Radix-4	2802	95.25	1042	1024	95.25

III. ARCHITECTURE

The 1024-point FFT processes 1024 complex samples, with 64-bit length (32-bit word for real and imaginary part). Those samples are store in the memory RAM1, each one with a direction. The dragonfly takes 4 samples and operates, this process is repeated 256 times and it's stored in the memory RAM2. Then it replaces the RAM1 with the RAM2 information and process the dragonfly again. This process is done 5 times in order to finish the calculations of the Radix-4 FFT. We can see the flowchart in the Figure 3.

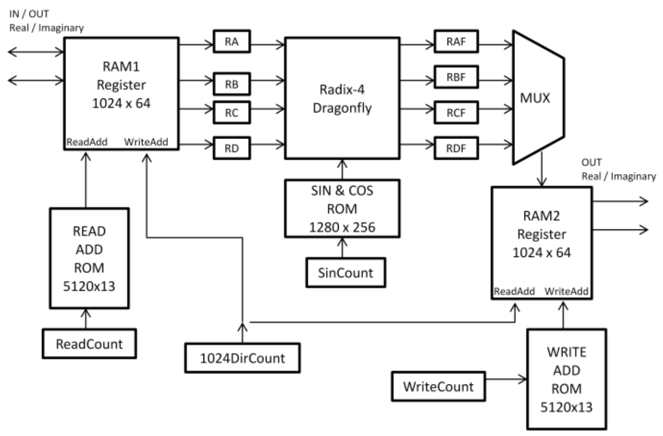

Fig. 3. Radix-4 DIF FFT Flow Chart

The dragonfly accepts 64-bit words consisting of a two IEEE-754 single-precision format with 32-bits (1-bit sign, 8-bit exponent, and 23-bit fraction). The first one is for the real part and the second one is for the imaginary part. Figure 4.

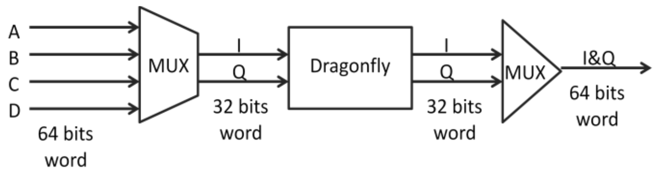

Fig. 4. FFT Bit Length

The 1024 complex input signals are stored into the RAM1. This register mixes the real part and an imaginary part, the input (complex) gets a new sample every clock cycle. When the store register gets full, it generates a valid signal that trigger and starts the FFT process.

The program starts by defining a main unit call "FFTfinal", this unit run two sub-stages: "cont_gen" and "unidadbasica", each one with a different task. As shown in Figure 5.

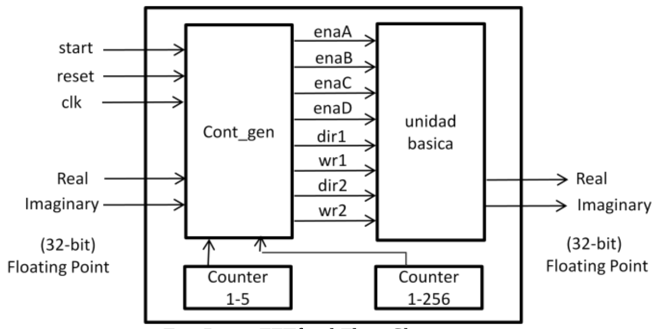

Fig. 5. FFTfinal Flow Chart

The first one is the controller; in this stage we can see the states of the operation. They are divide in fifteen states and each one have the key parameterization for the next stage. It's deploying with a switch-case command.

The second stage operates the states for the controller. It has counters to get the synchronization and direction of the data. The directions of in and out are charge in two separate ROMs and we have two units to access to the RAM, who

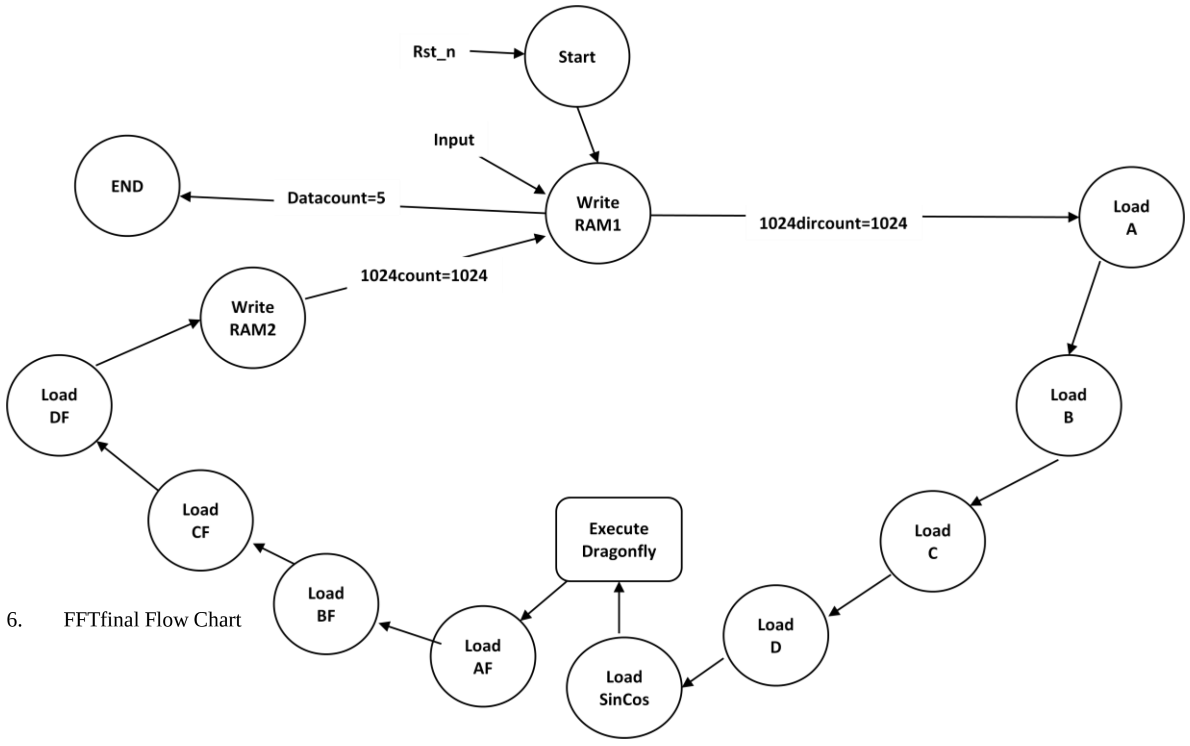

Fig. 6. FFTfinal Flow Chart

contains the original function, one unit to read and another to overwrite. The rest of parts are for execute the dragonfly.

The program starts loading the data into memory "RAM1", then copied into the registers"LoadA", "LoadB", "LoadC" and "LoadD". Twiddles coefficients are loaded from the ROM "LoadSinCos" and runs the dragonfly.

The dragonfly produces four records "LoadAF", "LoadBF", "LoadCF" and "LoadDF", they are stored in memory "RAM2" and once the memory is full proceeds to overwrite memory "RAM1". The counter"DataCount" expected to run 5 times the same process and terminates execution. The overall algorithm computation is supervised by a sequencer with the work flow as shown in Figure 6.

Processes that require synchronization are counters and register writes, so these will need a clock signal "clk".

The main block for the FFT computation is the dragonfly processor which contains complex float-point single-precision multipliers and adders. These operations are implemented with an IP Core Multiply Adder v2.0 [5], which one is already available in the Spartan 6.

The dragonfly is divided in two parts; those are shown in Fig. 7. The first one “ArmaReI” makes the multiplications and adds with the twiddles coefficients, cosine and sine complex form. The second part “sumasfinales” makes the adders and subtractions, and gives the output orderly.

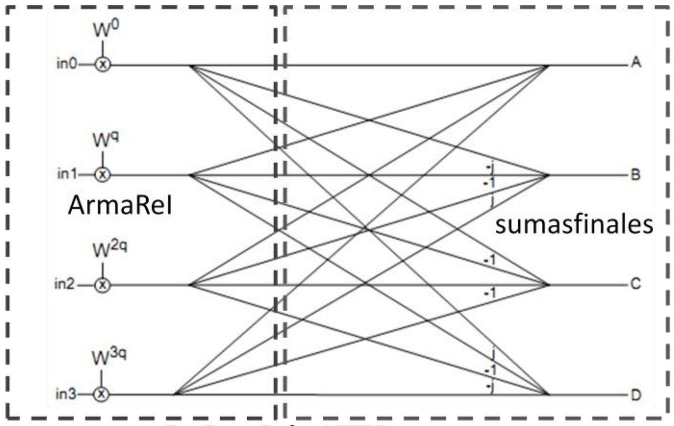

Fig. 7. Radix-4 FFT Representation

During the implementation we optimize the use of the clock in order to optimize the time of response.

IV. RESULTS

The time-computation performance of the FFT is estimated by the software for the FPGA Spartan 6 and given us the following result:

- Clock period: 7.127ns (frequency: 140.308MHz).

We can see the optimum device utilization in the following estimated table:

TABLE II. IMPLEMENTATION RESULTS ON SPARTAN-3 DEVICES.

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	419	4800	8 %
Number of Slice LUTs	1202	2400	50 %
Number of fully used LUT-FF pairs	147	1474	9 %
Number of bonded IOBs	7	102	4 %
Number of Block RAM/FIFO	10	12	83 %
Number of BUFG/BUFGCTRLs	1	16	6 %

The implementation reaches a very near result in compare with the Xilinx FFT core [6] and better result than other FFT core [7], as shown in Table IV.

TABLE III. USED RESOURCES REPORT.

Utilization	Xilinx FFT Core	FFT Core [6]	Case study [7]
Slice Registers	47%	93%	38%
Slice LUTs	43%	84%	50%
LUT-FF pairs	9%	43	9%
RAM/FIFO	16%	66%	66%
Average	38%	72%	41%

For validation purposes, we have captured the output of the simulation and compared with the FFT in MatLab [8], which is shown at Figure 8. The Matlab FFT is computed by the following code:

```
t=1:1:1024;                                  % 1024 Point
x=cos(2*pi*0.35*t)+cos(2*pi*0.38*t);      % Test Input
% We save the output of the FPGA in the w register.
subplot(311); plot(abs(w));
axis([0 1024 0 500]);title('1024-point FFT FPGA');
subplot(312); plot(abs(fft(x)));
axis([0 1024 0 500]);title('FFT MATLAB');
subplot(313); plot((abs(x)-w)*1000000000);
axis([0   1024   -5*(10^-5)    5*(10^-5)]);title('FFT
MATLAB');title('Error');
```

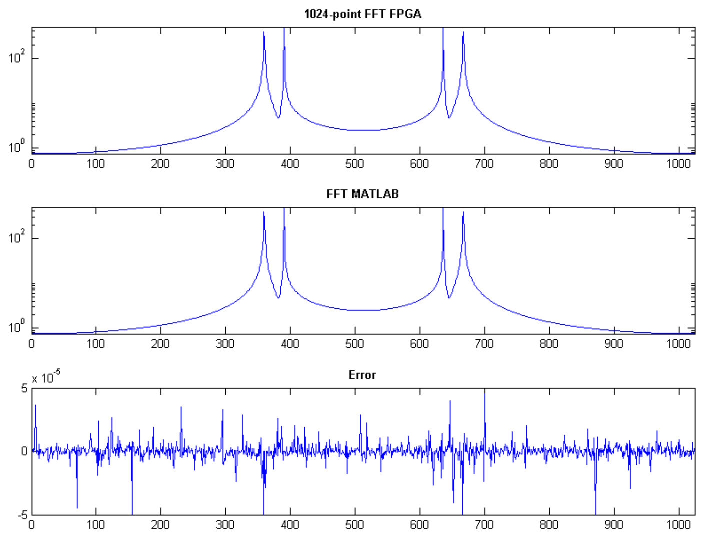

Fig. 8. Compare FFT MatLab with FPGA result

V. CONCLUSIONS

This work has described a radix-4 algorithm pipeline FFT in float point and was compared with a float-point model in MATLAB, reaching a result very accurate. We have also compared with the core 1024-Point Radix-4 FFT Computation [7], and we have observed similar results.

As the Radix-4 FFT algorithm utilizes less complex multipliers than the Radix-2 FFT algorithm, the Radix-4 algorithm is preferable for hardware implementation. A parallel programming approach seems to be a good choice when a real time system with high sampling rate is desired. For reaching an acceptable level of phase error, it is desirable to use 32 bits precision on the input signal and the twiddle factor.

ACKNOWLEDGEMENTS

This work is supported by RNP – Rede Nacional de Ensino e Pesquisa - Brazil, contract no. 001334, "*Construindo Cidades Inteligentes: da Instrumentação dos Ambientes ao Desenvolvimento de Aplicações*".

REFERENCES

[1] A. V. Oppenheim and R. W. Shafer, *Discrete-Time Signal Processing*, 2nd Upper Saddle River, NJ: Prentice Hall, 1998.

[2] J. G. Proakis and D. G. Manolakis, *Tratamiento Digital de Señales: Algoritmos y Aplicaciones*, 3rd edition, Prentice Hall, Madrid, 1998.

[3] W. Hussain, F. Garzia, J. Nurmi, "Evaluation of Radix-2 and Radix-4 FFT Processing on a Reconfigurable Platform", *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010

[4] B. Zhou, Y. Peng and D. Hwang, "Pipeline FFT Architectures Optimized for FPGAs", *IEEE International Journal of Reconfigurable Computing*, Hindawi Publishing Corporation 2009

[5] Xilinx® LogiCORE IP, Multiply Adder v2.0, Xilinx® 2011

[6] Xilinx® LogiCORE IP, Fast Fourier Transform V7.1, Xilinx® 2011.

[7] J. A. Vite-Frias, R. J. Romero Troncoso and A. Ordaz Moreno, "VHDL Core for 1024-Point Radix-4 FFT Computation", *IEEE International Conference on Reconfigurable Computing and FPGAs.*

[8] Magnus Nilsson, "FFT, Realization And Implementation In FPGA", Signal Processing Laboratory, School of Microelectronic Engineering, Griffith University, Brisbane/Gothenburg 2000/2001.