Low size FFT core for OFDM communications

Andrés D. Cassagnes, Federico G. Zacchigna and Octavio Alpago
Facultad de Ingeniería
Universidad de Buenos Aires
{acassagnes,fzacchigna,oalpago}@fi.uba.ar

Ariel Lutenberg
Facultad de Ingeniería
Universidad de Buenos Aires
and CONICET
lse@fi.uba.ar

***Abstract*—Two FFT architectures are presented in this work. The architectures are based in the Radix algorithm. In particular, a radix-2 and a radix-4 are implemented to be used in a ISDB-T OFDM modulator.**
The main objetive is to achieve a very small footprint, in terms of the resources/space demanded by the core, keeping the performance of the standard FFT cores used as ISDB-T OFDM modulator, which will be the final use of the core.
Radix algorithm has been selected becasue it provides high re-utilization, implemented over an iterative structure, using only one design of a butterfly module, multiplier and memory. In this scheme, the main complexity is in the control unit and the datapath.
The FFT core was described using the Verilog hardware description language and the test scripts were written in the Matlab script language.

I. Introduction

The growing demand of speed in telecommunications leads to the implementation of faster transmission systems.
One of the most used data transmission systems is Orthogonal Frecuency Division Multiplexing, *OFDM* [2], which uses multiple carriers to modulate the transmission data.
The main difference between the traditional frecuency multiplexing systems and OFDM is that in in the last one the carriers are overlapping, taking advantage of their orthogonality, in contrast to the traditional system where the carriers have a gap between them to prevent inter-carrier interference [2].

The basis of the OFDM transmission is the sum of sub-carriers (which can be expressed by a complex exponential, or a *frequency* in the complex plane) multiplied by the data symbols as expressed in equation (1), where *k* is the sub-carrier number.

$$s_k(t-kT) = w(t-kT) \sum_{i=-N/2}^{N/2-1} x_{i,k} e^{j2\pi\left(\frac{i}{T_{FFT}}\right)(t-kT)} \quad (1)$$

It's easy to recognize in equation (1) an Inverse Discrete Fourier Transform (where the points in the frequency domain are translated to the time domain).
Then, the OFDM modulator bank can be replaced by the computation of an IDFT and the demodulator bank by the computation of a DFT. It's possible to optize the implementation by using of efficient IDFT/DFT algorithms known as Fast Fourier Transform.

The objetive of this work is to obtain an FFT core, small enough to be included in a complete OFDM transeiver without consuming too much resources, but efficient and accurate enough to be usefull in an ISDB-t television system.

II. Architechture selection

There are several algorithms to calculate the FFT. Each of them has some advantages and disadvantages. As we are trying to achieve a small implementation, the radix-r algorithm is selected beacause of it's particularity of using the same modules for all the FFT calculation steps [1].
Radix-r algorithms are a variation of Cooley-Tukey algorithms [3]. Radix-r factorices the FFT length, N, in the form of $N = r^\nu$, so the N-point FFT is decomposed in ν r-points sub-FFTs. The main advantage of the factorization in r is that the computation module can be replied or reused for each sub-FFT calculation.

A. Radix-r Algorithm

This algorithm is based in the factorization of the FFT the length N through the bidimentional mapping:

$$n = N_2 n_1 + n_2 \qquad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \quad (2)$$

$$k = k_1 + N_1 k_2 \qquad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \quad (3)$$

where n is the time domain index and k is the frequency domain index, and $N = N1 * N2$.

Expressing the DFT and IDFT in the forms of equations (4) and (5)

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad (4)$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad (5)$$

where $W_N^{kn} = e^{\frac{-j2\pi kn}{N}}$ are known as *twiddle factors*. n and k can be replaced by (2) and (3), in (4):

$$X[k1,k2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2k_2} \left(W_N^{n_2k_1} \sum_{n_1=0}^{N_1-1} x[n_1,n_2] W_{N_1}^{n_1k_1} \right) \quad (6)$$

The inner summation in (6) is a N_1 points DFT multiplied by the factor $W_N^{n_2k_1}$. Taking $\tilde{x}[n2,k1] = W_N^{n_2k_1} \sum_{n_1=0}^{N_1-1} x[n_1,n_2] W_{N_1}^{n_1k_1}$ and replacing in (6):

$$X[k1,k2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2k_2} \tilde{x}[n2,k1] \quad (7)$$

Equation (7) shows the N_2 points $\tilde{x}$ DFT. It demostrates the power of the algorithm by replacing an N point DFT with two smaller sequential DFT.

Sub-DFTs can be divided applying the described method in turn to reduce the original DFT to several sub-DFTs of smaller length and simpler to operate.

Another advantage of this algorithm is the possibility of in-place memory using, where the results of an operation is holded in the memory position of the operands, so for an N point DFT, an N length memory is needed.

The value of r affects the type and quantity of operations needed, as can be seen in table I.

TABLE I. OPERATIONS NEEDED BY DIFFERENT VALUES OF *r*

r	Multiplications	Non trivial multiplications	additions
2	2	0	2
3	3	2	6
4	4	0	8
5	6	5	17
7	9	8	36

For $r = 2$ and $r = 4$ there aren't non trivial multiplications, so this are the values chosen for r.
A radix-r FFT has $log_r N$ stages. As table I shows, for $r = 4$ more operations per stage are needed but there are half the stages than for $r = 2$. Then, both are implemented in order to compare them and bring the possibility to choose depending on the requirements of the specific application.

B. Implementation of the Radix-r architechture

Figure 1 shows the simplified scheme for 8 points radix-2 FFT.

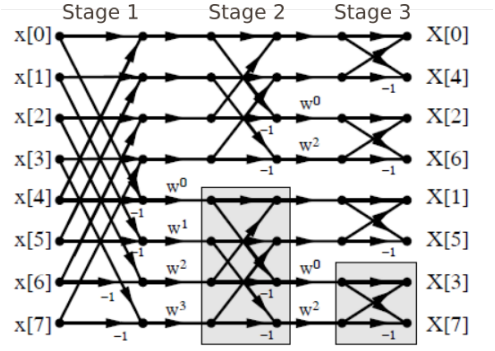

Fig. 1. 8 points Radix-2 FFT algorithm representation

Each node represents an addition and the arrows represents a multiplication by the value over it. Figure 1 shows the stage division of the algorithm, each of them performing a 2 points DFT.
In general, for a N points DFT $\frac{N}{2} * \log_2(N)$ butterflies and $\frac{N}{2} * (\log_2(N) - 1)$ complex multipliers are needed. There are different implementations for the radix algorithms which provides optimizations in different aspects of the performance. Possible implementations are:

- **Parallel** All *butterfly* and multipliers are implemented in similar scheme as the one showed in figure 1.
- **Unrolled** Single Delay Feedback (SDF) architechture [4]. Uses a *butterfly* and a complex multiplier per stage.
- **Iterative** Only one *butterfly* and one complex multiplier, used secuencially by all the stages. Figure 2 shows an 8 points iterative radix-2 FFT scheme.

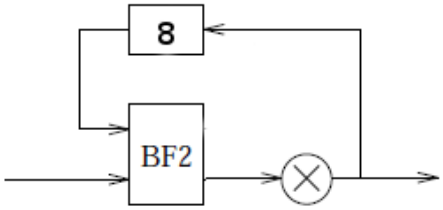

Fig. 2. Iterative Radix-2 block diagram

Table (II) shows the comparision of the characteristics of the three implementations described above.

TABLE II. COMPARATIVE BETWEEN PARALLEL, UNROLLED AND ITERATIVE RADIX-R

Characteristic	Parallel	Unrolled	Iterative
# *butterfly*	$\frac{N}{\nu} * \log_\nu(N)$	$\log_\nu(N)$	1
# multipliers	$\frac{N}{\nu} * (\log_\nu(N) - 1)$	$log_\nu(N) - 1$	1
Memory length	0	$N - 1$	N
Bus type	Parallel	Serie	Serie
throughput	N points per cycle	1 point per cycle	1 per $\log_\nu(N)$ cycles
pipeline	Yes	Yes	No

In order to ensure the low space and low resource requirement, iterative implementation is chosen.

C. Twiddle factors multiplication

Radix algorithms require the multiplication by twiddle factors. It is well known that multiplications in digital implementation are expensive, spacially and temporally.

Three methods were analized:

- Cordic Algorithm
- BKM Algorithm
- Efficient complex multiplication

1) Cordic Algorithm: Twiddle factors have the form $W_N^{kn} = e^{\frac{-j2\pi kn}{N}}$. They represent a rotation in the complex axis. A well known and well proved algorithm for rotations is the cordic algorithm [5]. It is based on successive aproximations by micro-rotations until the desired angle is reached.

The main advantage of this algorithm is that it only uses additions (and substractions) and shifts, both of them very cheap in terms of resources. Also it can be pipelined, improving the speed of processing.

2) BKM Algorithm: Like the cordic algorithm, it resolves elemental equations using only additions and shifts [6]. In comparison with Cordic Algorithm, BKM requires more storage and is more complex. In addition, it's greater efficience is obtained using redundant numeric system, which is not the case of the present work. Because of this reasons, BKM is discarded for this project.

3) Efficient Complex multiplier: Cordic algorithm is widely used in FFT implementation because of its very low cost in terms of space and resources. But in an iterative implementation, where only one multiplier is required, the difference between the cordic core and a complex multiplier is despicable. For twiddle factors, the multiplication required is:

$$R+jI = (A+jB)*(C+jD) = (A*C-B*D)+j(A*D+B*C) \tag{8}$$

where $(C+jD)$ is the twiddle factor. A straight implementation would need four multipliers, but pre-calculating some of the factors and storing them in memory (as the $tg\theta$ in cordic) can reduce the implementation to three multiplications.
Pre-calculated factors are C, $(C+D)$ and $C-D$. Then, pre-calculated values are used to obtain $Z = Cx(A-B)$ and then:

$$R = (C-D) \times B + Z \tag{9}$$

$$I = (C+D) \times A - Z \tag{10}$$

Taking into account that several FPGAs have DSP integrated modules, the implementation of multipliers could be very efficient.

D. Summary of the proposed implementation

As it has been exposed in this section, the following architechtures are implemented:

- Radix-2 iterative architechture.
- Radix-4 iterative architechture.
- Cordic algorithm for twiddle factors multiplications, for radix-2 and radix-4.
- Efficient complex multiplier for twiddle factors multiplications, for radix-2 and radix-4, as an alternative to cordic algorithm.

III. RADIX-2

In figure (1) are shown the different stages of radix-2 FFT implementation. On each clock cycle, one of two posible operations can be performed:

- A point is stored in memory while another point is sended from memory to twiddle factor multiplier or to the core output.
- A butterfly operation between a core-input point or last-stage point and a memory-stored point. Two points results from the butterfly operation: one is stored in memory while the other is sended to the twiddle factor multiplier or to the output.

Main components of the core are: N point memory, butterfly, multiplier an the control unit.

A. Memory

Due to the type of memory operations used simultaneous store and read data is needed, so the memory unit is implemented as a dual-port RAM memory of length N.

B. Butterfly

The butterfly unit has to perform the two-operands complex operations:

$$\begin{aligned} c &= a + b \\ d &= a - b \end{aligned} \tag{11}$$

C. Control Unit

The control unit has to set the multiplexers according to the operation that is performed in that clock cycle, address the memory to the position where the actual operand has to be readed or stored, and generate the twiddle factors for the multiplier.
Given that the core has $log_2(N)$ stages, the control unit has a stage-counter with length $log_2(log_2(N))$. Another counter, with a length of $log_2(N)$ counts the number of points that have entered into the core by that moment. A state machine is controlled with these counters, and controls the architecture in base of the stage and point number.

D. Integration

Figure 3 presents the integrated core. Control unit signal are showed as arrows to keep the graphic clear.
An additional register is placed before the multiplier because one result of a given stage is used in the following stage, so it must be keeped for one clock cycle. Another register is placed in the output in order to bring sequential sinchronization with the circuit connected to the core.
An optional rounding/clipping unit is provided after the butterfly to give a method to deal with overflow. This can be turned on selectivelly for each stage in real time.

IV. RADIX-4

Radix-4 algorithm divides a N-point DFT in ν 4-point DFT, so that $N = 4^{\nu}$. The following expressions resumes the operations that the radix-4 has to process [3]:

$$y_n = (x_n + x_{n+\frac{l}{4}} + x_{n+\frac{l}{2}} + x_{n+\frac{3l}{4}}) \tag{12}$$

$$z_n = ((x_n - x_{n+\frac{l}{2}}) - j(x_{n+\frac{l}{4}} - x_{n+\frac{3l}{4}}))W_N^k \tag{13}$$

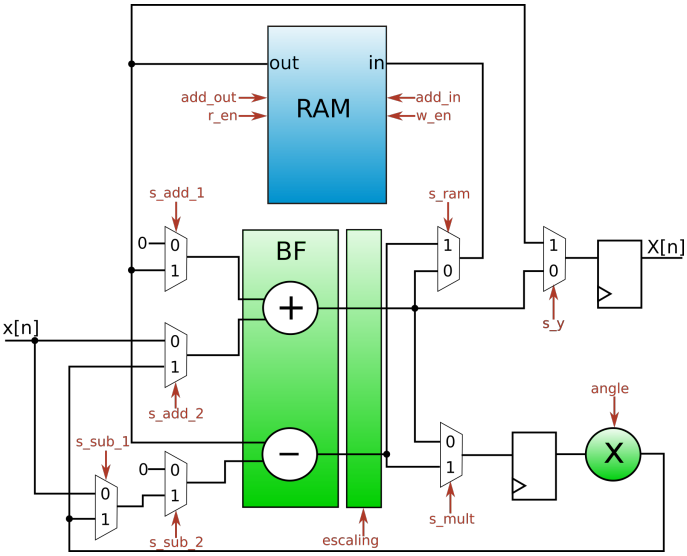

Fig. 3. Iterative Radix-2 implementation diagram

$$g_n = ((x_n + x_{n+\frac{l}{2}}) - (x_{n+\frac{l}{4}} + x_{n+\frac{3l}{4}}))W_N^{2k} \quad (14)$$

$$h_n = ((x_n - x_{n+\frac{l}{2}}) + j(x_{n+\frac{l}{4}} - x_{n+\frac{3l}{4}}))W_N^{3k} \quad (15)$$

for $k = 0, 1, \ldots, \frac{N}{4} - 1$, where l depends on the current processing stage. Figure 4 shows the operational scheme of a 16-points radix-4 algorithm.

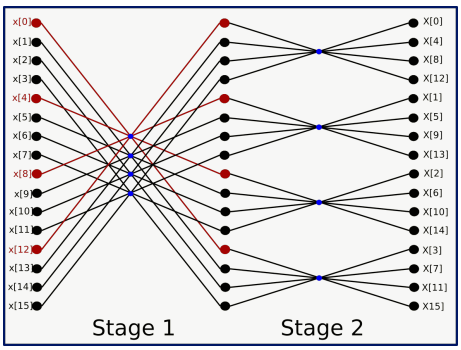

Fig. 4. 16 points Radix-4 FFT algorithm representation

On each clock cycle, one out of four possible operations is performed:

- A point is stored in a memory sub-block from the core input or the twiddle factor multiplier, while another point is sent from the same memory sub-block to the multiplier or the output. For every sub-block, a different operation is considered.
- An arithmetic operation is performed with a point from the core input or the previous stage and 3 points from memory, each from a different memory sub-block.

Main components are the memory, the arithmetic unit (called dragonfly), multiplier and the control unit, specially designed for radix-4.

A. Memory

Each arithmetic operation needs four operands, one coming from the core input or the previous stage and another three coming from the storage, so a 3-way in/3-way out memory is needed. In order to take advantage of memory blocks present in most FPGAs, a special memory is designed for this core. It is formed by three dual-port RAMs similar to radix-2 memories, so in each arithmetic operation, an operand of each memory sub-block can be readed and stored simultaneously.
As the directions may not be succesive, because in each clock cycle a different stage operation is performed, each sub block is divided in ν regions delimited by addresses. Sub-block lenght decreases successively in the form $N/4$, $log_4(N/4)$, $Log_4(Log_4(N/4)) \ldots 1$.

B. Dragonfly

The arithmetic unit calculates the equations (12) to (15). In each stage two different types of operations can be performed: a four point arithmetic calculation or a memory data traslation. In the last case, a point from a memory sub-block is trasferred to the multiplier, while a point is stored in the next stage memory sub-block region. The dragonfly unit includes an inner datapath which guides the data according to the operation in progress.

C. Control unit

As well as in radix-2, the control unit configures the datapath and generate the memory addresses and the twiddle factors for the multiplier. As in radix-2, two counters are used: a $log_2(N)$ length points counter and a $log_2(log_4(N))$ length stage counter. Determining which operation must be perform in a given clock cycle is done by evaluating the pair of the point counter's bits refered by the value of the stage counter. Memory addressing is done by mapping directly the point counter to the memory address. The sub-block region is selected with the stage counter because each sub-block is subdivided in regions for each stage. The read and write control signalas are controlled according to the type of operation: in memory transfer operations only one sub-block is enabled while in arithmetical operations all three sub-blocks are allowed to be readed and written.

D. Integration

Figure (5) presents the iterative radix-4 core. As in radix-2, extra registers are added after arithmetic unit and in the output. Also the rounding/clipping unit is added to the dragonfly to prevent overflow.

V. Architecture Characterization

Beside of the individual tests for every composing unit, a set of tests is made over the entire architectures in order to verify and validate the design. For a complete description of tests and results, refer to [9].

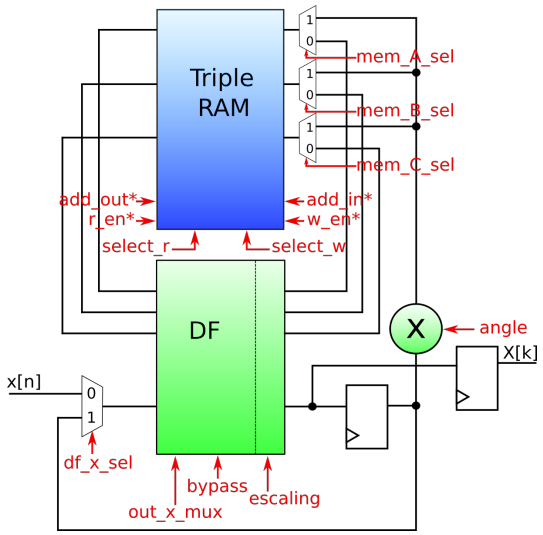

Fig. 5. Datapath with control signals

A. Standard signals

First, a set of standard signals is applyed to the cores, and the result is compared to the expected result. The cores are configurated in IFFT mode. The signals are:

- Delta in frequency component '0'. Expected a continuous signal.
- Delta in frequency component 6. Expected six cycles of a sin.
- Sin of period N/6. Expected a delta in time component 6.

All the tests were passed correctly.

B. Error measurement

In order to measure the architecture error, Matlab FFT is taken as a benchmark because it operates with 64 bits floating point numbers.

Two metrics are used for error measuring, E_∞ and E_2:

$$E_\infty = MAX(\frac{X_o[n] - X_{dut}[n]}{X_o[n]}) \tag{16}$$

$$E_2 = \| \frac{X_o[n] - X_{dut}[n]}{X_o[n]} \|_2 \tag{17}$$

It's important to note that the radix architectures are non linear, so every test consists of 1024 simulations with random vector inputs. In each simulation, the result of the core computation is compared with Matlab results to obtain the error. After the 1024 simulations, error values are averaged to obtain the final error values.
This is done for 12 and 16 bit implementations of radix-2 and radix-4 cores. This is because 12 bits is a standard word length in OFDM communication systems and 16 bits is a standard in signal processing. Also cordic and complex multiplier versions are tested, in order to comparate their performance.
As an extra test bench, a widely used, 16 bit integer C++ FFT core is tested [7]. Test results are shown in table III and table IV.

TABLE III. E_∞ FOR 1024 SIMULATIONS WITH RANDOM INPUTS

	1024, 12 bits	1024, 16 bits	4096, 12 bits	4096, 16 bits
Radix-2, Cordic	0.092	0.006	0.099	0.008
Radix-2, Mult.	0.232	0.003	0.340	0.108
Radix-4, Cordic	0.077	0.003	0.074	0.007
Radix-4, Mult.	0.224	0.002	0.334	0.105
Kiss FFT		0.017		0.035

TABLE IV. E_2 FOR 1024 SIMULATIONS WITH RANDOM INPUTS

	1024, 12 bits	1024, 16 bits	4096, 12 bits	4096, 16 bits
Radix-2, Cordic	0.095	0.007	0.116	0.053
Radix-2, Mult.	0.257	0.004	0.356	0.131
Radix-4, Cordic	0.084	0.002	0.094	0.027
Radix-4, Mult.	0.258	0.003	0.358	0.126
Kiss FFT		0.017		0.035

From tables III and IV is clear that the performance of the cores is perfectly suitable for OFDM systems. Moreover, the cores can be used in signal processing as the error is under 1%. For complex multiplier the error can be cutted down by increasing the word length of the factors stored in memory. For Cordic rotator, the error can be cutted down by adding rotation steps.

C. THD

In order to measure the THD of the architectures, sixteen test are performed. One for each architecture, radix-2 or radix-4, for all options described before: 12 or 16 bits, 1024 or 4096 points and cordic or complex multiplier for twiddle factors. Additionally, THD test is made over KISFFT to get a testbench [7].
Each test runs consecutive simulations that use as input a tone in every input point each time. That way, a graphic can be made with the harmonic response to every frequency tone. Figures 6 and 7 shows some of the results.

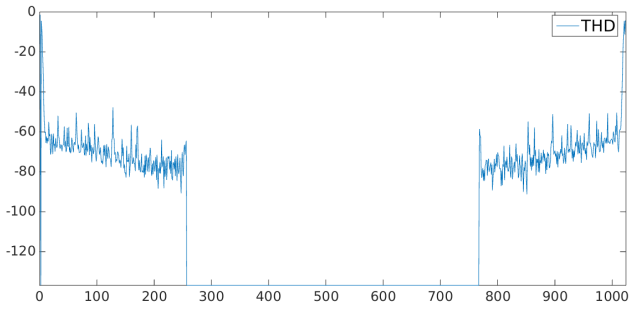

Fig. 6. THD [dB] vs input tone [Hz] for radix-2, Cordic, 12 bits.

D. Test on Hardware

For hardware validation, a Xilinx XC5XVL110 Virtex-5 FPGA is used. 1024 points, 12 bits iterative radix-2 and radix-4 are sinthetyzed with Xilinx ISE v13.4 and routed into the FPGA along with a testbench circuit which provides PC connection via a UART port.
Standard signal and error tests, described above, are repeated

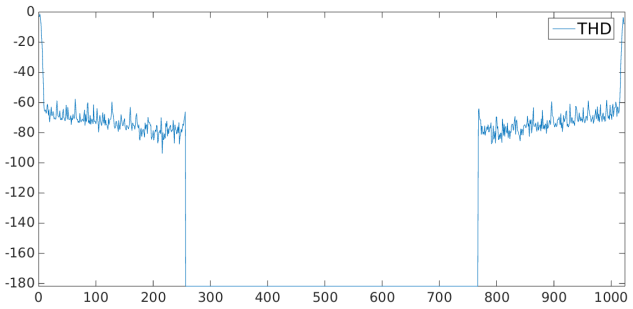

Fig. 7. THD [dB] vs input tone [Hz] for radix-4, Cordic, 12 bits.

on the hardware implementation obtaining same results as simulation tests, providing the succesfull on-hardware validation of the cores.

E. Resource ocuppation

The main requirement for the design is the low space/resource ocupation. In order to validate this requirement accomplishment, 1024 and 4096, 16 bits, iterative radix-2 and radix-4 architectures are sinthetyzed. To have a comparative reference, a 16 bits radix-2 sdf is implemented for 1024 and 4096 points. Also, as a reference testbench, Xilinx's LogiCORE FFT v7.1 [8] is sinthesized.

Figures 8 and 9 clearly shows that the designed cores are really small compared with another implementations, includding a propietary, device-optimized one like the LogiCORE FFT. Another important fact is that radix-4 and radix-2 needs approximately the same resource quantity, but radix-4 has double the throughtput than radix-2.

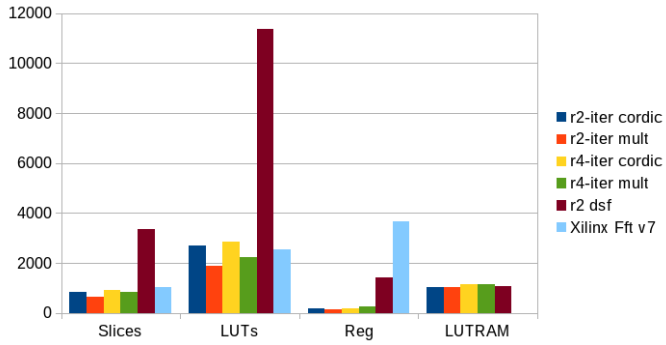

Fig. 8. Size/resource comparition for 1024 points FFT

VI. Conclusion

This paper presented two iterative radix-r FFT computing cores, designed for OFDM communication systems. Their main advantage is the extremly low space/resource consuption, which made them suitable for integration in large systems without impacting in the resource distribution, in case of FPGA implementation, or space in case of ASIC implementation. A complete description of the design and the verification and validation process has been presented.

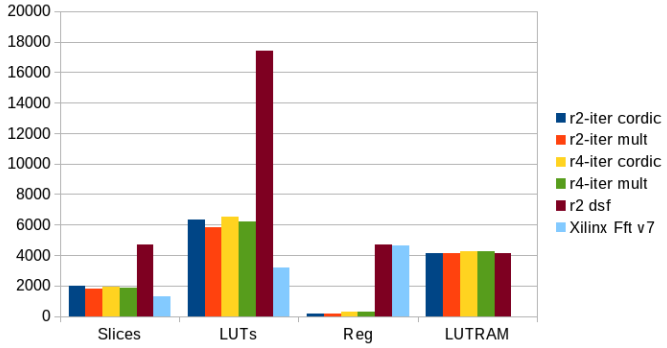

Fig. 9. Size/resource comparition for 4096 points FFT

The architectures were implemented in Verilog HDL code. Also there were developed several testing tools comprising Matlab, C++ programs and Verilog testbenchs.
For future work, it can be considerated to add a dithering system, in order to reduce the noise generated by the architectures, and to implement a pipelined cordic without modifying the global architecture timing, in order to improve the throughtput.

References

[1] Oppenheim, A., Shafer, R. (1999). The Discrete Fourier Transform. In *Discrete-Time Signal Processing*, USA: Prentice Hall.

[2] Prasad, R. (2004), Orthogonal Frequency-Division Multiplexing. In *OFDM for Wireless Communications Systems* (p.p. 11-15), UK: Artech House.

[3] Meyer-Baese, U. (2007). Fourier Transform. En *Digital Signal Processing with Field Programmable Gate Arrays* (p.p. 363-373), Berlin: Springer-Verlag.

[4] Shousheng H., Torkelson M. (1996). A New Approach to Pipeline FFT Processor. *International Parallel Processing Symposium*

[5] Volder, J. (1959). The Cordic computer technique. *IRE. Trans. Elect. Comput.*

[6] Bajard, J.C., Kla, S., Muller, J.C. (1994, Agosto). BKM: A New Hardware Algorithm for Complex Elementary Functions. *IEEE Transaction on Computers 43*(8)

[7] https://sourceforge.net/projects/kissFFT/?source=navbar *Kiss FFT is a very small, reasonably efficient, mixed radix FFT library that can use either fixed or floating point data types.*

[8] LogiCORE IP Fast Fourier Transform v7.1. (2011) Xilinx

[9] Cassagnes, A., Lutenberg, A., Giordano Zacchigna, F. (2016) "Implementacion y analisis de algoritmos de calculo de Transformada Rapida de Fourier para su aplicacian en sistemas OFDM" http://laboratorios.fi.uba.ar/lse/tesis/LSE-FIUBA-Tesis-Grado-Andres-Cassagnes-2016.pdf