Size saving FFT core for OFDM comunications

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

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

***Abstract*—Two FFT architectures are presented. The architectures are based in the Radix algorithm. In particular, a radix-2 and a radix-4 are implemented.**
The main objetive is to achieve a very small architecture, in terms of the resources/space demanded by the core, keeping the performance of a regular FFT core. That restriction is due to compliance the specifitacions of a ISDB-t oriented OFDM modulator, which will be the final use of the core.
Radix algorithm has been selected becasue it provides high modules re-utilization, implemented over an iterative structure, using only one butterfly module, one multiplier and one memory. In that scheme, the main complexity is in the control unit and the pipeline.
The design was made in Verilog hardware description language and the test scripts were made in Matlab scripting language.

I. Introduction

The continuous growing demand for speed in telecomunications leads to the implementation of faster transmision systems.
One of the most used data transmision systems is Orthogonal Frecuency Division Multiplexing, *OFDM*, which uses multiple carriers to modulate the transmited data.
The main difference between the traditional frecuency multiplexing systems and the orthogonal frecuency multiplexing systems is that in OFDM modulations the carriers are overlapped, taking advantage of their orthogonality as seen in Fig. (1), in oposition to the traditional system where the carriers have a gap between them to prevent inter-carrier interference. OFDM is used widely in nowadays communications, being one of the most extended [2].

Fig. 1. OFDM sub-carriers scheme

The high number of sub-carriers needed to perform a communication at the speeds required nowadays, makes it imposible to be implemented with dedicated modulators and demodulators for each sub-carrier. The most optimal implementation for the OFDM modulator-demodulator is by using Discrete Fourier Transform.

The basis of the OFDM transmision is the sum of sub-carriers (wich can be expressed by a complex exponential, or a *frequency* in the complex plane) multiplied by the data complex symbols. Matematically, it is expressed as seen in equation (1)

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

where k is the sub-carrier number. It's easy to recognize in this equation the form of an Inverse Discrete Fourier Transform (where the points in the frequency domain are translated to the time domain).
Using this, the OFDM modulators bank can be replaced by the computation of a IDFT and the demodulators bank by the computation of a DFT, making it possible to implement an OFDM modulator/demodulator using a mathemathic computing core. It's even possible to make the implementation more optimal by the use of efficient IDFT/DFT algorithms known as Fast Fourier Transform.

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

II. Architechture selection

There are several algorithms for FFT calculation. Each has some advantages and disadvantages. As we are trying to achieve the smallest implementation, the radix-r algorithm is selected. It has the particularity of using equal modules in every step of the transform, so it's the best choice in terms of the implementation efficiency [1].
Radix-r algorithms are a variation of Cooley-Tukey algorithms [3]. In Cooley-Tukey algorithms the FFT calculation is reduced to m sub-FFTs through the factorization of the number of points, (N). Radix-r variations factorices 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:

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

$$k = k_1 + N_1 k_2 \qquad \begin{cases} 0 \leq k_1 \leq N_1 - 1 \\ 0 \leq k_2 \leq N_2 - 1 \end{cases} \tag{3}$$

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)

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

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

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):

$$W_N^{kn} = W_N^{N_2 n_1 k_1 + N_1 N_2 n_1 k_2 + n_2 k_1 + N_1 n_2 k_2} \tag{6}$$

As W_N^{nk} has order $N = N_1 N_2$ it becomes that $W_N^{N_1} = W_{N_2}$ and $W_N^{N_2} = W_{N_1}$, wich replaced in 6:

$$W_N^{kn} = W_{N_1}^{n_1 k_1} W_N^{n_2 k_1} W_{N_2}^{n_2 k_2} \tag{7}$$

Using (7) in (4), results in:

$$X[k1, k2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \left(W_N^{n_2 k_1} \sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_1}^{n_1 k_1} \right) \tag{8}$$

The inner summation in (8) is a N_1 points DFT multiplied by the factor $W_N^{n_2 k_1}$. Taking $\tilde{x}[n2, k1] = W_N^{n_2 k_1} \sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_1}^{n_1 k_1}$ and replacing in (8):

$$X[k1, k2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \tilde{x}[n2, k1] \tag{9}$$

(9) shows the N_2 points $\tilde{x}$ DFT, which represents the main advantage of this algorithm, for any factorization of N in the form of $N = N_1 N_2$, the N point DFT of $x(n)$ can be calculated following the steps:

- Map the input index using (2)
- Calculate N_1 points DFT of $x(n)$.
- Multiplie the resulting points by the twiddle factors.
- Calculate N_2 point DFT of the resulting secuence.
- Map the output index using (3)

Here we can subdivide the sub-DFTs applying the described method in turn to reduce the original DFT to several sub-DFTs of smaller length and simpler to operate.

An extra advantage of this algorithm is the posibility of in-place memory using, where the results of an operation is holded in the memory position of the operands, so for a N point DFT needs a N length memory.

Radix-r is a variation of this algorithm where $N = r^\nu$. The value of r affects the type of operations needed by the algorithm, as can be seen in table I.

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
8	8	2	26
9	11	10	44

TABLE I. COMPLEX OPERATIONS QUANTITY FOR DIFFERENT VALUES OF *r*

For $r = 2$ and $r = 4$ there aren't any 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. Both are implemented in order to make a comparition between them and provide the posibility to choose depending on the requirements of the implementation.

B. Implementation of the Radix-r architechture

Figure 2 shows the simplified scheme of an 8 points radix-2 FFT.

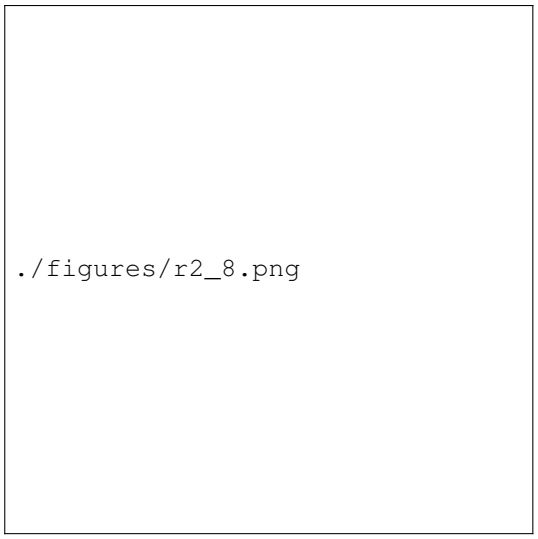

Fig. 2. 8 points Radix-2 FFT

Each node represents an addition and the arrows represents the multiplication by the value over the arrow. The figure shows the stage division of the algorithm, each one performs a 2 points DFT. Each 2 points DFT is known as a Butterfly. For a radix-4 DFT, the scheme is similar but every node is a four points addition.
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. But there are alternative implementations for the radix algorithms wich provides optimizations in butterflies and multipliers quantity, memory length, throughput (related with the transmition speed) and control-unit complexity.
The most-common implementations are:

- **Parallel** All *butterfly* and multipliers are implemented in similar scheme as the one showed in 2.
- **Unrolled** Single Delay Feedback (SDF) architechture [4]. Figure 3 shows a 4 point radix-2 SDF FFT scheme. It uses a *butterfly* and a complex multiplier per stage.
- **Iterative** It implements only one *butterfly* and one complex multiplier, wich performs secuencially the operation for every stages. Figura 4 shows a 8 points iterative radix-2 FFT scheme.

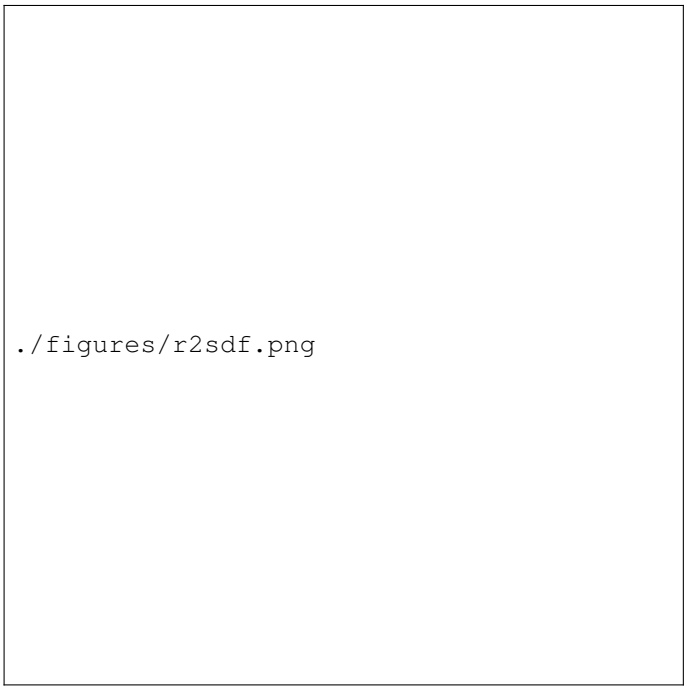

Fig. 3. Unrolled SDF Radix-2

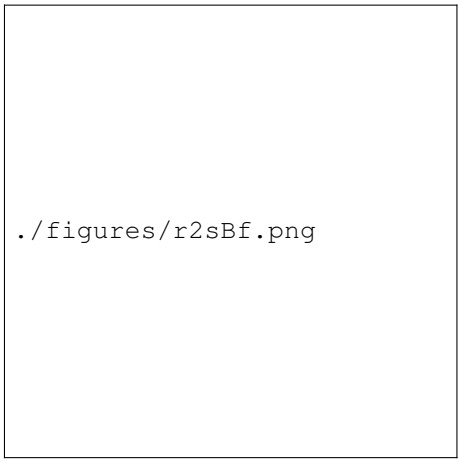

Fig. 4. Iterative Radix-2

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

One restriction of the implementation is the space efficiency, so the iterative implementation is selected, because it only needs one butterfly and one multiplier independently

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 cicle	1 point per cicle	1 per $\log_\nu(N)$ cicles
pipeline	Yes	Yes	No

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

of FFT length. In terms of space, the relation between the iterative vs the unrolled efficiency is $log_r(N) : 1$ Only the memory size depends on FFT length, but it is equal to the unrolled implementation, so there no advantages. This ensures the low space and low energy needed by the core.
Even though it represents low throughput, it is easily solved by feeding the core clock with the correct frequency.

C. Twiddle factors multiplication

Radix algorithms require the multiplication by twiddle factors. Is well known that multiplications in digital implementation are worthy, spacial and temporaly. So to choose the better alternative needs a carefully analisys.

Three methods were analised:

- Cordic Algorithm
- BKM Algorithm
- Efficient complex multiplication

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

$$x_{i+1} = x_i * \cos\theta_{i+1} - y_i * \sin\theta_{i+1} \tag{10}$$

$$y_{i+1} = y_i * \cos\theta_{i+1} + x_i * \sin\theta_{i+1} \tag{11}$$

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.

For more details of Cordic Algorith refer to [5].

2) BKM Algorithm: This algorithm, like the cordic algorithm, try to resolve elemental equations using only additions and shifts.
It is based on the following equations:

$$\begin{cases} L_{n+1} = L_n(1 + d_n * 2^{-n}) \\ E_{n+1} = E_n - \ln(1 + d_n * 2^{-n}) \end{cases} \tag{12}$$

In comparition with Cordic Algorithm, BKM requires more storage and is more complex. In addition, it's main efficience is

obtained using redundant numeric system. In the case of this work, the implementation is made usign two's complement number representation, wich is not redundant.
Because of this reasons, BKM is discarded for this project. For more information about BKM algorithms refer to [6].

3) Efficient Complex multiplier: Cordic algorithm is widely used in FFT calculation 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 very little.

For twiddle factors, the multiplication required is:

$$R+jI = (A+jB)*(C+jD) = (A*C-B*D)+j(A*D+B*C) \tag{13}$$

where $(C + iD)$ 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 only three multiplications, in fact, a 25%.

Pre-calculated factors are C, $(C+D)$ and $C-D$. Then, getting into the multiplier with the twiddle factor, pre-calculated values are used to obtain $Z = Cx(A - B)$ and then:

$$R = (C - D) \times B + Z \tag{14}$$

$$I = (C + D) \times A - Z \tag{15}$$

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

D. Summary of 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 twiddel factors multiplications, for radix-2 and radix-4, as an alternative to cordic algorithm.

III. RADIX-2

As it has been explained in previous sections, an iterative implementation for the N points radix-2 algorithm uses only one butterfly for the calculations fo every stage consecutevely. This ledas to waitng $log_2(N)$ cicles between two entry point and between two output points.
In Figure (2) are shown the different stages of radix-2 FFT implementation.

On each clock cicle, 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 dthe control unit. an input ultiplexer determines if the butterfly operation will be done with the core entry point or with a point stored in memory.

A. Memory

Due to the type of memory operations, simultaneous store and read data, the memory unit is implemented as a two-port RAM memory of length N. In fact, there is only $N-1$ memory positions needed, but for ease a N memory is implemented, because there is always one point from a stage that is used in the next stage, so it has not to be stored in memory. Memory positions have 2 word length in order to store the real and imaginary part of every point.

B. Butterfly

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

$$\begin{aligned} c &= a + b \\ d &= a - b \end{aligned} \tag{16}$$

C. Datapath

The datapath must comprise both types of operations described above, and trhee different variants for eacg one: the incoming data for a given stage could come from the core input or from the previous stage and/or from the memory, and the resulting data could go to the core output or to the multiplier and/or to the memory.
When a memory-store operation is performed, a cero ('0') is added or substracted to the operand so it is stored in memory through the butterfly unit.
This is done by a set of multiplexers controlled by the control unit.

D. Control Unit

The control unit has to set the multiplexers according to the operation that is performed in that clock cicle, address the memory to the position wherer 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. Is this state machine the

one wich controlls the architechture.

To decide if a particular stage performs an arithmetihci or a memory transfer operation, a bit of the points counter is evaluated. The position of that bit is determined by the stage counter, as shown in Figure (5) for an 8 point radix-2.

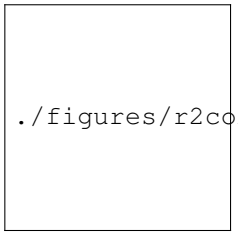

Fig. 5. Point counter bit selection

Memory control is doing by the reading and writing addresses and the read and write control signals. As in this case the memory is readed and writed in every clock cicles, control signals are always asserted. Respecting the addressing, being the radix-2 an in-place storing algorithm, in every clock cicle the result of an operation is stored in the same memory position of one of its operands. So the reading address and the writting address are the same, and corresponds to the points counter value.

E. Integration

Figure (6) presents the integrated core. Control unit signal are shown 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 cicle. Another register is placed in the output in order to bring secuencial sinchronization with the circuit connected to the core.
An optional scaling unit is provided after the butterfly to give a method to deal with overflow. The scaling unit 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 $N = 4^{\nu}$.
Breaking up the N-point DFT in 4 DFT of $N/4$ points each, it becomes to the next four expressions wich resumes the operations the radix-4 has to process ([3]):

$$y_n = (x_n + x_{n+\frac{l}{4}} + x_{n+\frac{l}{2}} + x_{n+\frac{3l}{4}}) \tag{17}$$

$$z_n = ((x_n - x_{n+\frac{l}{2}}) - j(x_{n+\frac{l}{4}} - x_{n+\frac{3l}{4}}))W_N^k \tag{18}$$

$$g_n = ((x_n + x_{n+\frac{l}{2}}) - (x_{n+\frac{l}{4}} + x_{n+\frac{3l}{4}}))W_N^{2k} \tag{19}$$

$$h_n = ((x_n - x_{n+\frac{l}{2}}) + j(x_{n+\frac{l}{4}} - x_{n+\frac{3l}{4}}))W_N^{3k} \tag{20}$$

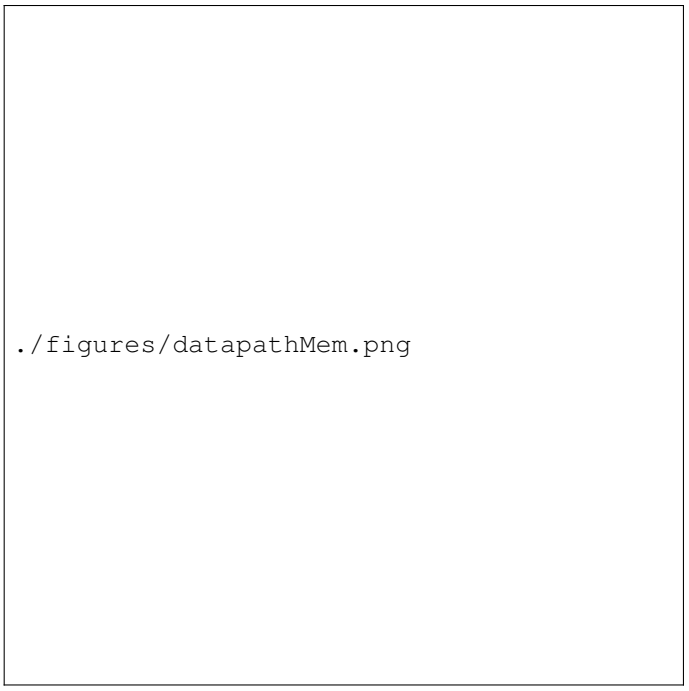

Fig. 6. Datapath with control signals

for $k = 0, 1, \ldots, \frac{N}{4} - 1$, where l depends on the current processing stage:

$$\begin{aligned} l_1 &= N \\ l_2 &= \log_4(N) \\ l_3 &= \log_4(\log_4(N)) \ldots \\ l_\nu &= 4 \end{aligned} \tag{21}$$

where l_i corresponds to the i-ith stage of a $N = 4^{\nu}$ point DFT.

It's clear that the radix-4 algorithm must process four points in each arithmetic operation: x_n, $x_{n+\frac{l}{4}}$, $x_{n+\frac{l}{2}}$ and $x_{n+\frac{l}{4}}$.

Figure (7)shows the operatinal scheme of a 16-points radix-4 algorithm.

On each clock cicle, one of four posible operations is performed:

- A point is stored in memory sub-block A, from the twiddle factor multiplier, while another point is sended from memory sub-block A to the multiplier or the output.
- B point is stored in memory sub-block B, from the twiddle factor multiplier, while another point is sended from memory sub-block B to the multiplier or the output.
- C point is stored in memory sub-block C, from the twiddle factor multiplier, while another point is sended from memory sub-block C to the multiplier or the output.
- An arithmetic operation is performed with a point from the core input or the previous stage and 3 points

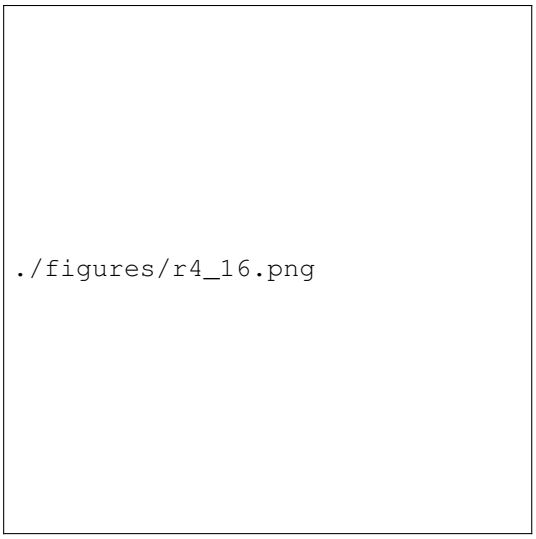

Fig. 7. 16 points Radix-4 FFT diagram

from memory, each from a different memory sub-block.

It can be appreciated that each arithmetic operation needs 4 operands, one will come from the core input or the previous stage and the other 3 will come from the storage memory, so a special memory is designed for this implementation.
Again, the main components are the memory, the aritmetic unit (called dragonfly), the datapath and the control unit.

A. Memory

As saying above, artihmetic operations needs 3 operands from memory, so a 3-way in 3-way out memory is needed. In order to take advantage of the memory blocks present in most FPGAs, a special memory is designed for this core, formed by 3 dual-port RAMs similar to radix-2 memories. This way, 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 cicle a different stage operation is performed, each sub block is divided in ν regions delimited by the address. Sub-block address regions are showed in Figure (8).

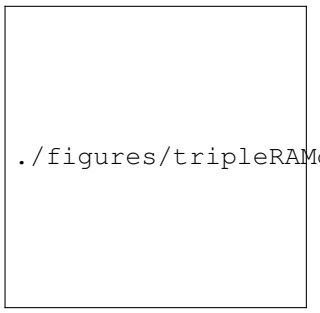

Fig. 8. Memory sub-blck addressing map

B. Dragonfly

The arithmetic unit performs the equations (17) to (20), wich are directly implemented in the logic.

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. The dragonfly unit includes an inner datapath wich guides the data according to the operation in progress.
A block diagram of the arithmetic is presented in Figure (9).

Fig. 9. Arithmetic unit block diagram

C. Datapath

The datapath must comprise both types of operations (memory transfer are in fact three different operations as it can be done from and to one of the three memory sub-blocks). The design schem is similar to radix-2 datapath, leaving the sub-block selection path to the dragonfly inner datapath.

D. 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. But in this case, a new point arrives every $log_4(N)$ clock cicles, while in radix-2 it occurs every $log_2(N)$, wich determines the stage number.
Also, two counters as used: a $log_2(N)$ length points counter and a $log_2(log_4(N))$ length stage counter. Determination of wich operation must be perform in a giiven clock cicle is done by the evaluation of pair of bits fo the point counter refered by the value of the stage counter, as shown in Figure (10).

Fig. 10. Point counter bits for operation selection

Memory addressing is done mapping directly the point counter to the memory addresses. The sub-block region selecetion is done using 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 the correct sub-block is enabled, in arithmetical operations all three sub-blocks are allowed to be readed and writed.

E. Integration

Figure (11) presents the iterative radix-4 core. As in radix-2, extra registers are added after arithmetic unit and in the output. A rounding/clipping unit is added to the dragonfly ti prevent overflow.

Fig. 11. Datapath with control signals

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 verificate and validate the design.
For a complete description of tests and results, refer to [9].

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 cicles 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, wich is mpre accurate than 12 or 16 bits integer numbers of the cores.

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

$$E_\infty = MAX(\frac{X_o[n] - X_{dut}[n]}{X_o[n]}) \tag{22}$$

$$E_2 = \|\frac{X_o[n] - X_{dut}[n]}{X_o[n]}\|_2 \tag{23}$$

It's important to note that the radix architectures are non lineal, so for solving this, every test consists of 1024 simulations with random vector inputs. In each simulation, the result of the core computation is compared with Matlab result and the errors are compupted. After the 1024 simulations, error values are promediated to obtain the final error values.
This is done for 12 and 16 bit implementation aof radix-2 and radix-4 cores. This is because 12 bits is a standard word length in OFDM comunication 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].

	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 III. E_∞ 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

TABLE IV. E_2 FOR 1024 SIMULATIONS WITH RANDOM INPUTS

In tables III and IV is clear that 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, 16 test are performed. One for each architecture, radix-2 or radix-4, for all flavours 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.
Each test run consecutive simulations using as input a tone in every input point each time. That way, a graphic can be made

Fig. 12. R-2, complex mult., 12 bits

Fig. 13. R-2, Cordic, 12 bits

with the armonic response to every frequency tone. Figures (12) to (15) shows some of the resulting graphics.
That graphics shows that the core response is similar as the KISSFFT response and also as the Matlab FFT response. So the THD is perfectly acecptable.

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 sinthesized with Xilinx ISE v13.4 and routed into the FPGA along with a testbench circuit wich provides PC connection via a UART port.
Testbench circuit is showed in Figure (16).

Standard signal tests and error test, described above, are repeated on the hardware implementation obtaining the same results, providing the succesfull on-hardware validation of the cores.

Fig. 14. R-4, complex mult., 12 bits

Fig. 15. R-4, Cordic, 12 bits

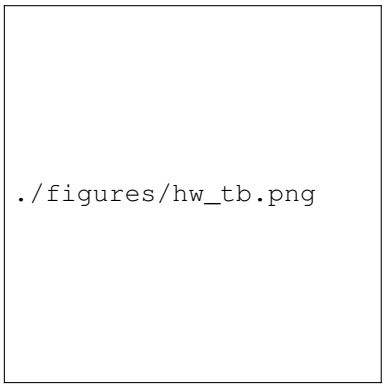

Fig. 16. Testbench for hardware validation

E. Resource ocupation

The main requirement for the design is the low space/resource ocupation.

To validate the requirement accomplishment, 1024 and 4096 16 bits iterative radix-2 and iterativa radix-4 architectures are sinthesized. To have a reference for comparition, a 16 bits radix-2 sdf (unrolled) is implemented for 1024 and 4096 points. Also, as a valid testbench, Xilinx's LogiCORE FFT v7.1 [8], is sinthesized for both point quantity.

Fig. 17. Size/resource comparitionComparativa for 1024 points FFT

Fig. 18. Size/resource comparitionComparativa for 4096 points FFT

Figures (17) and (18) clearly shows that the designed cores are really small compared with another implementations, inclulding a propietary, device-optimized one like the LogiCORE FFT. Another important fact is that radix-4 and radix-2 needs aproximated the same resources, but radix-4 has double the throughput than radix-2 (one point every $log_4 N$ vs one point every $log_2 N$).

VI. Conclusion

This paper presented two iterative radix-r FFT computing cores, designed for OFDM comunication systems. Their main advantage is the extremly low space/resource comsuption, wich made the 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.

A lot of effort was spent in the verification process, making a lot of simulations and tests, in order to provide a usable ip core.

The architectures are implemented in Verilog HDL code, comprising about 20 coding files. Also there were developed testing tools in form of Matlab/Octave scripting, C++ programs and Verilog testbenchs.

For future work, can be considerated to add a dithering system, in order to reduce the noise generated by the architectures, and implement a pipelined cordic without modifing the global architecture timing, in order to improve the throughput.

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) "Implementacin y anlisis de algoritmos de clculo de Transformada Rpida de Fourier para su aplicacin en sistemas OFDM" http://laboratorios.fi.uba.ar/lse/tesis/LSE-FIUBA-Tesis-Grado-Andres-Cassagnes-2016.pdf