VLSI Implementation of a Pipelined 128 points 4-Parallel radix-2^3 FFT Architecture via Folding Transformation

James J. W. Kunst jjwk89@gmail.com
Kevin H. Viglianco kevinviglianco@gmail.com
Daniel R. Garcia dani6rg@gmail.com

Digital Signal Processing in Very Large Scale Integration Systems

Autumn 2019

Dr. Keshab K. Parhi
Dr. Ariel L. Pola

Universidad Nacional de Córdoba - FCEFyN
Av.Vélez Sársfield 1611, X5016GCA, Córdoba, Argentina

Fundación FULGOR
Ernesto Romagosa 518, Colinas V. Sarsfield, X5016GQN, Córdoba, Argentina

***Abstract*—- This work describes the design and the VLSI implementation of a 4-parallel pipelined architecture for the complex fast Fourier transform (CFFT) based on the radix-2^3 algorithm with 128 points using folding transformation and register minimization techniques. In addition, different synthesis reports from the Hardware Description Language (HDL) using different optimization techniques were studied in order to obtain good performance on speed and area using an open-source FreePDK45 of 45 nm CMOS technology [1].**

I. Introduccion

The Fast Fourier Transform (FFT) is widely used in different applications fields, particularly in algorithms that involves applying digital signal processing, e.g., calculate the Discrete Fourier Transform (DFT) efficiently. Nowadays is common the use of FFT algorithm for real time applications and parallel-pipelined hardware architecture, this allows to achieve good performance with high throughput rates.

There are two main types of pipelined FFT architectures [2]. On one hand, feedback architectures (FB) which can be divided into Single-path Delay Feedback (SDF) and Multi-path Delay Feedback (MDF), both methods transfer data samples between stages serially and use feedback loops. On the other hand, feedforward architectures such as Multi-Path Delay Commutator (MDC) transfers more than one sample per clock cycle and do not use feedback loops.

This work focuses on the design of 4-parallel pipelined architecture *radix*-2^3 128-points for Complex FFT-DIF (Decimation In Frequency). Section II, describes the equations that correspond to Butterfly structure of *radix*-2^3 FFT-DIF. In Section III, the design of a 2-parallel pipelined architecture, *radix*-2^3 16-points FFT via folding transformation is presented. In Section IV, the previous design is translate to a 4-parallel, 128-points *radix*-2^3 DIF complex FFT, and a float-point simulator in *Matlab* is elaborated, to later be compared with a fixed-point model in order to obtain the best Signal to Quantization Noise Ratio (SQNR). In Section V, different power, area and timing reports with different optimizations such as varying pipelining levels, and the application of canonical signed digit (CSD) are compared to obtain the best performance with a clock frequency of $500MHz$. Finally in Section VI some conclusions and discussions are presented.

II. Radix-2^3 FFT algorithm

The N-point DFT of an input sequence $x[n]$ is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n]\dot{W}_N^{nk}, \quad k = 0, 1, ..., N-1 \tag{1}$$

where $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$.

The FFT based on Cooley-Tukey algorithm is most commonly used to compute the DFT efficiently, this allows to reduce the number of operations from $O(N^2)$ for the DFT to $O(Nlog_2N)$. Direct computation of the DFT is basically inefficient primarily because it does not exploit the symmetry and periodicity properties of the phase factor W_N, these two properties are:

$$\text{Symmetry property: } W_N^{k+N/2} = -W_N^k \tag{2}$$

$$\text{Periodicity property: } W_N^{k+N} = W_N^k \tag{3}$$

The development of computationally efficient algorithms for DFT is possible if a *Divide and Conquer* approach is adopted. This approach is based on the decomposition of an N point

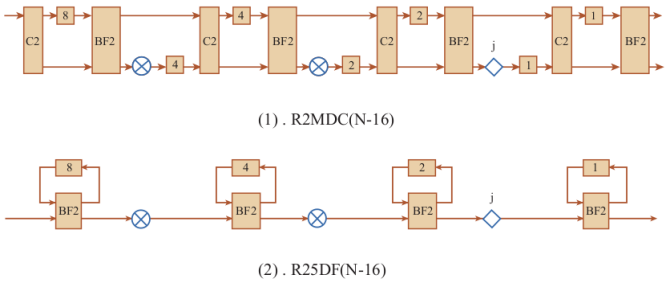

Figure 1: Types of pipelined FFT archicectures for 16 points [10].

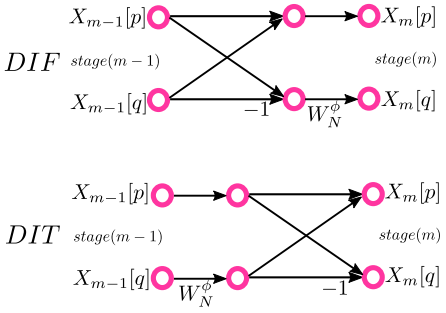

Figure 2: Basic butterflies computation in the decimation in time and frequency.

DFT into successively smaller DFTs. In accordance with this, the DFT is calculated in series of $s = log_\rho N$ stages, where ρ is the base of the *radix*. In this work this factor is two, so the number of stages for 128 points is 7.

According to [3], [4], There are two methods to design FFT algorithms:

a) Decimation in time (DIT): In this method the N-point data sequence $x[n]$ is split into two $N/2$-point data sequences, thus, is possible to obtain two different functions by decimating $x[n]$ by a factor of 2. The decimation of the data sequence can be repeated again until the resulting sequences are reduced to one-point sequence.

b) Decimation in frequency (DIF): This method is based on the divide-and-conquer technique, where the DFT formula is split into two summations, one of which involves the sum over the first $N/2$ data points and the second sum involves the last $N/2$ data points.

In each decomposition, the basic computing unit that processes the samples is called *butterfly*. In general, each butterfly involves one complex multiplication and two complex additions. The main difference between DIT and DIF is the instant in which the multiplication by W_N^{ϕ} is accomplished, the input samples can be multiplied before or after the split and the samples are later added inside the butterfly structure, as is depicted in Fig.2.

Another difference between the two methods are that the input samples in algorithms DIF are organized in natural order but its output are not in order, in which case a reordering circuit at the output is needed. On the other hand, the input sequence for DIT algorithms are not in order and its output are in natural order.

According the methodology presented in [3], it is possible to apply the mathematical expressions of *radix*-2^3 DIF explained in [5].

These fundamental equations are in :

$$C_{8k+0} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n + x_{n+\frac{N}{2}}) + (x_{n+\frac{N}{4}} + x_{n+\frac{3N}{4}})] + [(x_{n+\frac{N}{8}} + x_{n+\frac{5N}{8}}) + (x_{n+\frac{3N}{8}} + x_{n+\frac{7N}{8}})] \Big\} W_N^{0n} W_{N/8}^{nk} \tag{4}$$

$$C_{8k+4} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n + x_{n+\frac{N}{2}}) + (x_{n+\frac{N}{4}} + x_{n+\frac{3N}{4}})] - [(x_{n+\frac{N}{8}} + x_{n+\frac{5N}{8}}) + (x_{n+\frac{3N}{8}} + x_{n+\frac{7N}{8}})] \Big\} W_N^{4n} W_{N/8}^{nk}$$

$$C_{8k+2} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n + x_{n+\frac{N}{2}}) - (x_{n+\frac{N}{4}} + x_{n+\frac{3N}{4}})] - j [(x_{n+\frac{N}{8}} + x_{n+\frac{5N}{8}}) - (x_{n+\frac{3N}{8}} + x_{n+\frac{7N}{8}})] \Big\} W_N^{2n} W_{N/8}^{nk}$$

$$C_{8k+6} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n + x_{n+\frac{N}{2}}) - (x_{n+\frac{N}{4}} + x_{n+\frac{3N}{4}})] + j [(x_{n+\frac{N}{8}} + x_{n+\frac{5N}{8}}) - (x_{n+\frac{3N}{8}} + x_{n+\frac{7N}{8}})] \Big\} W_N^{6n} W_{N/8}^{nk}$$

$$C_{8k+1} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n - x_{n+\frac{N}{2}}) - j(x_{n+\frac{N}{4}} - x_{n+\frac{3N}{4}})] + W_N^{N/8} [(x_{n+\frac{N}{8}} - x_{n+\frac{5N}{8}}) - j(x_{n+\frac{3N}{8}} - x_{n+\frac{7N}{8}})] \Big\} W_N^{n} W_{N/8}^{nk}$$

$$C_{8k+5} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n - x_{n+\frac{N}{2}}) - j(x_{n+\frac{N}{4}} - x_{n+\frac{3N}{4}})] - W_N^{N/8} [(x_{n+\frac{N}{8}} - x_{n+\frac{5N}{8}}) - j(x_{n+\frac{3N}{8}} - x_{n+\frac{7N}{8}})] \Big\} W_N^{5n} W_{N/8}^{nk}$$

$$C_{8k+3} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n - x_{n+\frac{N}{2}}) + j(x_{n+\frac{N}{4}} - x_{n+\frac{3N}{4}})] + W_N^{3N/8} [(x_{n+\frac{N}{8}} - x_{n+\frac{5N}{8}}) + j(x_{n+\frac{3N}{8}} - x_{n+\frac{7N}{8}})] \Big\} W_N^{3n} W_{N/8}^{nk}$$

$$C_{8k+7} = \sum_{n=0}^{N/8-1} \Big\{ [(x_n - x_{n+\frac{N}{2}}) + j(x_{n+\frac{N}{4}} + x_{n+\frac{3N}{4}})] - W_N^{3N/8} [(x_{n+\frac{N}{8}} - x_{n+\frac{5N}{8}}) + j(x_{n+\frac{3N}{8}} - x_{n+\frac{7N}{8}})] \Big\} W_N^{7n} W_{N/8}^{nk}$$

Fig. 3 and Fig. 4 show the equivalent diagram of interconnections and data flows from the equations presented in (4).

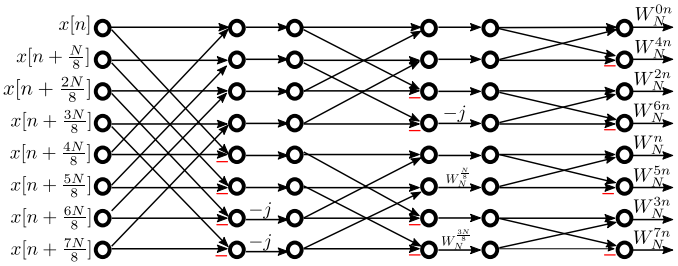

Figure 3: Structure of interconnection for *radix*-2^3 DIF DFT.

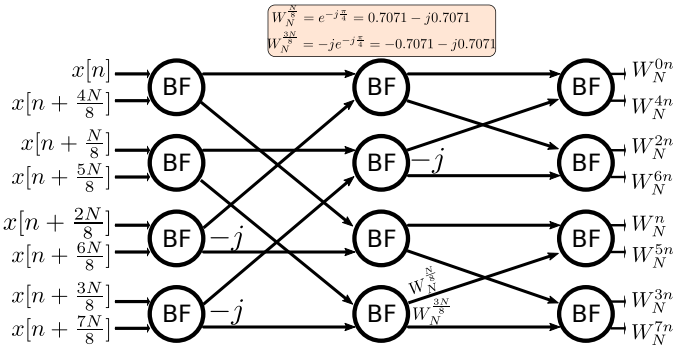

Figure 4: Data flow graph (DFG) based in equations (4).

A. 16 points DFT

The next step is to find the suitable rotator factors for the 16 point DFT, the equations in (4) are essential for this design and where evaluated to get $C_{8k+i} = \sum_{n=0}^{16/8-1}\{\cdot\}$, for $k = 0, 1$. The structure for the 16 point DFT is described in Fig. 5 and Fig. 6.

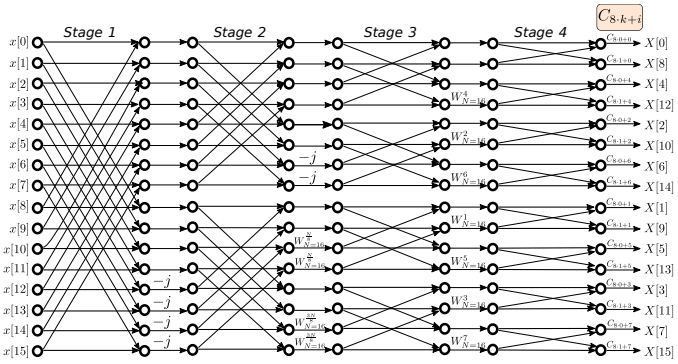

Figure 5: Flow graph of a *radix*-2^3 16-point DIF DFT

B. 128 points DFT

With these first approaches is possible the application of the divide and conquer strategy by decomposing the 128-point DFT and calculating each coefficient $C_{8k+i} = \sum_{n=0}^{128/8-1}\{\cdot\}$, for $k = 0, 1, ..., (128/8)-1$, this way a chain sequence of butterflies obtain together with its corresponding rotation factor and the correct index of the samples in which they must be added or

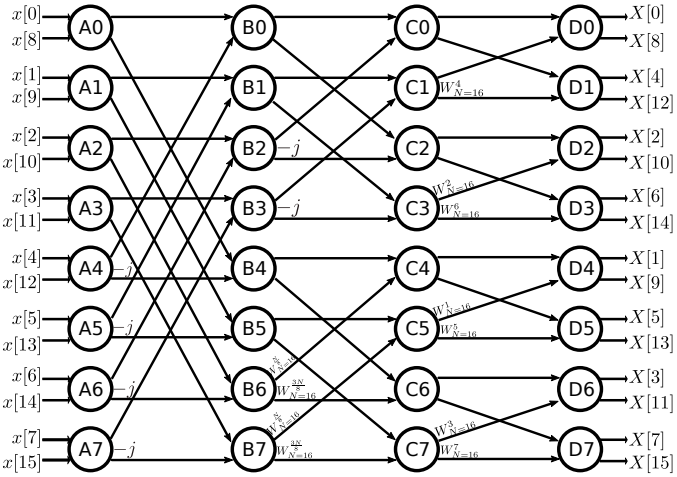

Figure 6: Data flow graph (DFG) for a *radix*-2^3 16-point DIF DFT

multiplied. With this technique is possible to do a subdivision of butterflies stages, as is shown in Fig. 7, the decomposition of the 128 point DFT involve three stages of butterflies to finally arrive to a set of eight 16 point DFTs.

III. Design of a FFT architecture via folding transformation

In this section, we illustrate the folding transformation method to derive a 16-point DIF FFT 4-parallel architecture as an example and then, using the same method, we extend it to 128-point architecture. To do this, we will use the architecture proposed in [6], to do the folding transfomation and register minimization techiques we will use [7].

A. 4-Parallel radix-2^3 16-Points

In Fig. 5 we can see the flow graph of a 16-point DIF FFT radix-2^3 with main base radix-2. The graph is divided into four stages and each of them consist of a set of butterflies and multipliers. The twiddle factor in between the stages indicates a multiplication by W_N^k, where W_N denotes the Nth root of unity, with its exponent evaluated modulo N. This can be represented as a DFG as shown in Fig. 6 where the nodes represents the butterfly computations of the radix-2 FFT algorithm.

The folding transformation is used on the DFG to derive a pipelined architecture. To do this we need a folding set, which is an ordered set of operations executed by the same functional unit. Each folding set contains K entries, where K is called the folding factor. The operation in the jth position within the folding set (where goes from 0 to $K-1$) is executed by the functional unit during the time partition, this term is called the folding order.

First we need to derive the folding equations, to do this consider an edge e connecting the nodes U and V with $w(e)$ delays. Let the executions of the lth iteration of the nodes U and V be scheduled at the time units $Kl + u$ and $Kl + v$ respectively, where u and v are the folding orders of the nodes U and V, respectively. The folding equation for the edge e is:

$$D_F(U \rightarrow V) = Kw(e) - P_U + v - u \qquad (5)$$

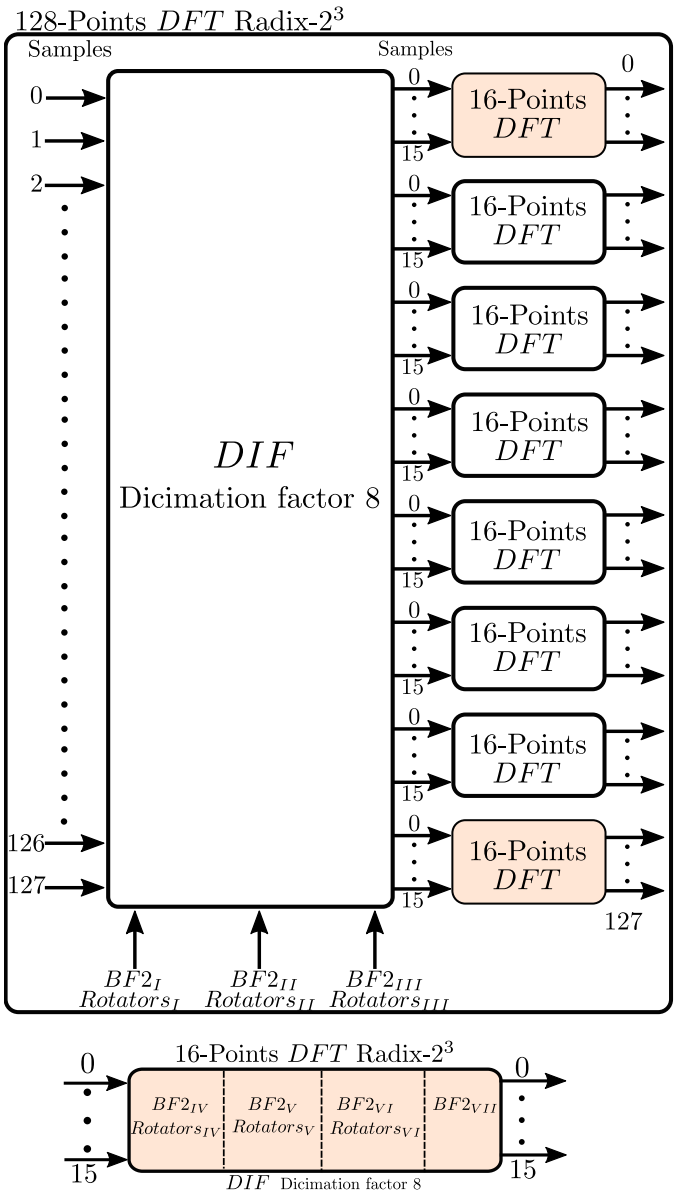

Figure 7: Decomposing a *radix*-2^3 128-point DFT.

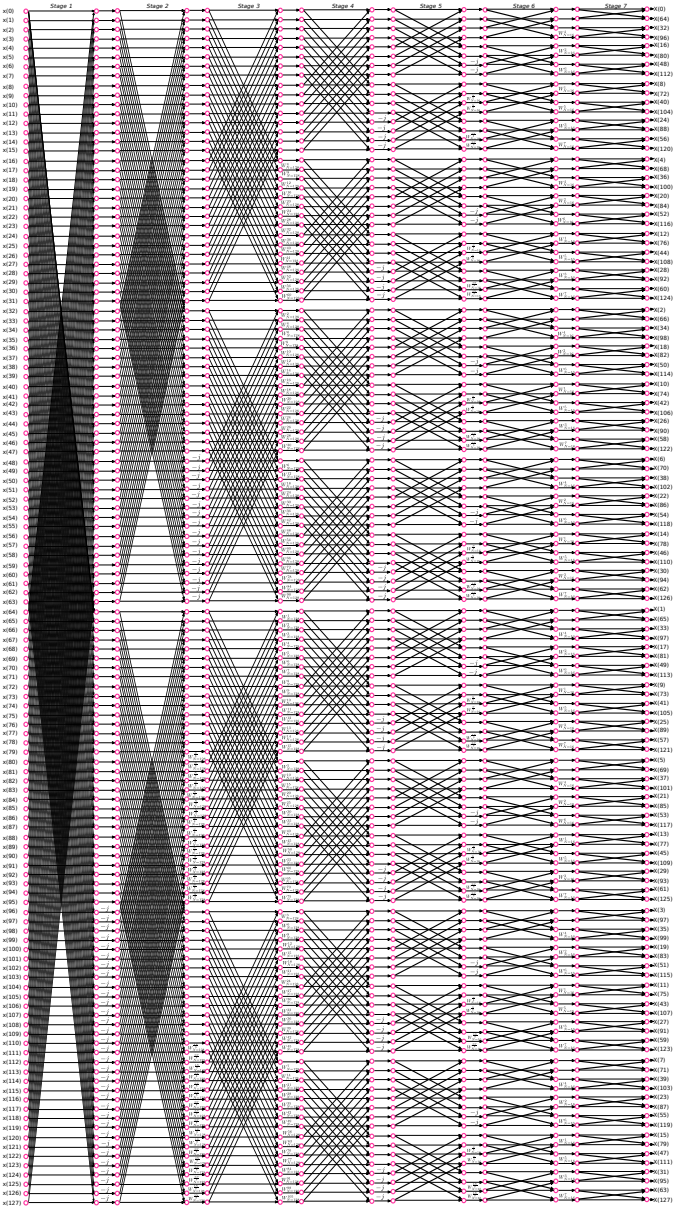

Figure 8: Flow graph of a *radix*-2^3 128-point DIF DFT

where P_U is the number of pipeline stages in the hardware unit which executes the node U.

Consider folding of the the DFG in Fig. 6 with the folding sets:

$$A = \{A0, A2, A4, A6\} \qquad A' = \{A1, A3, A5, A7\}$$
$$B = \{B1, B3, B0, B2\} \qquad B' = \{B5, B7, B4, B6\}$$
$$C = \{C2, C1, C3, C0\} \qquad C' = \{C6, C5, C7, C4\}$$
$$D = \{D3, D0, D2, D1\} \qquad D' = \{D7, D4, D6, D5\}$$

Assuming that the butterfly operations do not have any pipeline stages ($P_A = P_B = P_C = P_D = 0$), the folding equations can be derived for all edges. Thus, we can obtained from (5) the expressions *without apply retiming*.

$$D_F(D0 \to B0) = 2 \qquad D_F(D0 \to B4) = 2$$
$$D_F(D1 \to B1) = 0 \qquad D_F(D1 \to B5) = 0$$
$$D_F(D2 \to B2) = 2 \qquad D_F(D2 \to B6) = 2$$
$$D_F(D3 \to B3) = -1 \qquad D_F(D3 \to B7) = -1$$
$$D_F(D4 \to B0) = 0 \qquad D_F(D4 \to B4) = 0$$
$$D_F(D5 \to B1) = -1 \qquad D_F(D5 \to B5) = -1$$
$$D_F(D6 \to B2) = 0 \qquad D_F(D6 \to B6) = 0$$
$$D_F(D7 \to B3) = -2 \qquad D_F(D7 \to B7) = -2$$
$$D_F(E0 \to C0) = 1 \qquad D_F(E0 \to C2) = -2$$
$$D_F(E1 \to C1) = 1 \qquad D_F(E1 \to C3) = 2$$
$$D_F(E2 \to C0) = 0 \qquad D_F(E2 \to C2) = -3$$
$$D_F(E3 \to C1) = 0 \qquad D_F(E3 \to C3) = 1$$
$$D_F(E4 \to C4) = 1 \qquad D_F(E4 \to C6) = -2$$
$$D_F(E5 \to C5) = 1 \qquad D_F(E5 \to C7) = 2$$
$$D_F(E6 \to C4) = 0 \qquad D_F(E6 \to C6) = -3$$
$$D_F(E7 \to C5) = 0 \qquad D_F(E7 \to C7) = 1$$
$$D_F(F0 \to D0) = -2 \qquad D_F(F0 \to D1) = 0$$
$$D_F(F1 \to D0) = 0 \qquad D_F(F1 \to D1) = 2$$

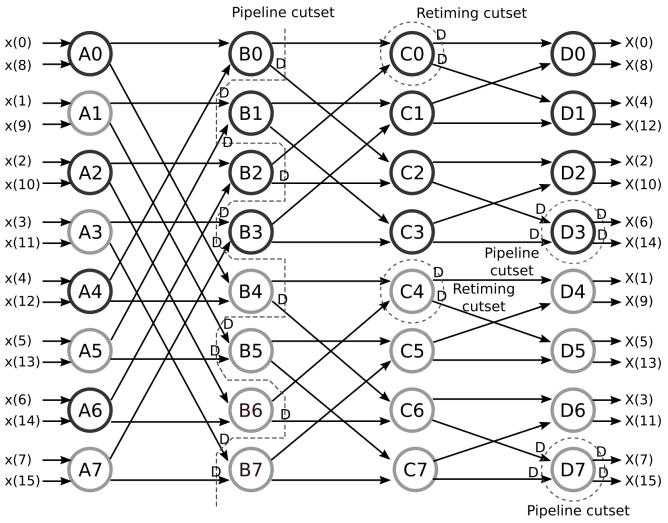

Figure 9: Data Flow graph (DFG) of a radix-2 16-point DIF FFT with retiming and pipeline.

For the folded system to be realizable, $D_F(U \rightarrow V) \geq 0$ must hold for all the edges in the DFG. Retimming and/or pipeline can be applied to satisfy this property, if the DFG in Fig. 6 is pipelined/retimmed as shown in Fig. 9 the system is realizable and the folded delays for the edges are given by the equations that represent a folding set *with retiming*.

$$
\begin{aligned}
&D_F(D0 \rightarrow B0) = 2 && D_F(D0 \rightarrow B4) = 2 \\
&D_F(D1 \rightarrow B1) = 4 && D_F(D1 \rightarrow B5) = 4 \\
&D_F(D2 \rightarrow B2) = 2 && D_F(D2 \rightarrow B6) = 2 \\
&D_F(D3 \rightarrow B3) = 3 && D_F(D3 \rightarrow B7) = 3 \\
&D_F(D4 \rightarrow B0) = 0 && D_F(D4 \rightarrow B4) = 0 \\
&D_F(D5 \rightarrow B1) = 3 && D_F(D5 \rightarrow B5) = 3 \\
&D_F(D6 \rightarrow B2) = 0 && D_F(D6 \rightarrow B6) = 0 \\
&D_F(D7 \rightarrow B3) = 2 && D_F(D7 \rightarrow B7) = 2 \\
&D_F(E0 \rightarrow C0) = 1 && D_F(E0 \rightarrow C2) = 2 \\
&D_F(E1 \rightarrow C1) = 1 && D_F(E1 \rightarrow C3) = 2 \\
&D_F(E2 \rightarrow C0) = 0 && D_F(E2 \rightarrow C2) = 1 \\
&D_F(E3 \rightarrow C1) = 0 && D_F(E3 \rightarrow C3) = 1 \\
&D_F(E4 \rightarrow C4) = 1 && D_F(E4 \rightarrow C6) = 2 \\
&D_F(E5 \rightarrow C5) = 1 && D_F(E5 \rightarrow C7) = 2 \\
&D_F(E6 \rightarrow C4) = 0 && D_F(E6 \rightarrow C6) = 1 \\
&D_F(E7 \rightarrow C5) = 0 && D_F(E7 \rightarrow C7) = 1 \\
&D_F(F0 \rightarrow D0) = 2 && D_F(F0 \rightarrow D1) = 4 \\
&D_F(F1 \rightarrow D0) = 0 && D_F(F1 \rightarrow D1) = 2 \\
&D_F(F2 \rightarrow D2) = 2 && D_F(F2 \rightarrow D3) = 4 \\
&D_F(F3 \rightarrow D2) = 0 && D_F(F3 \rightarrow D3) = 2 \\
&D_F(F4 \rightarrow D4) = 2 && D_F(F4 \rightarrow D5) = 4 \\
&D_F(F5 \rightarrow D4) = 0 && D_F(F5 \rightarrow D5) = 2 \\
&D_F(F6 \rightarrow D6) = 2 && D_F(F6 \rightarrow D7) = 4 \\
&D_F(F7 \rightarrow D6) = 0 && D_F(F7 \rightarrow D7) = 2
\end{aligned}
\tag{6}
$$

We can see that the number of registers required to implement the folding equations in (6) is 80. For minimize the number of registers we use the register minimization technique. If we let the output of node $A0$ be $y_{(0)}$ and $y_{(8)}$ and the output of the node A1 be $y_{(1)}$ and $y_{(9)}$, applying this successively with the rest of the nodes A we can obtain the linear life time chart for this stage in Fig. 10. Applying this criteria to the rest of the stages we can obtain the life time chart 11 and 12 for the outputs of the nodes B and C respectively, we can see that the numbers of maximum registers in each stage are 8, 4 and 8 respectively, therefore the number of registers is reduced from 80 to 20. More information about this method can be found on [7].

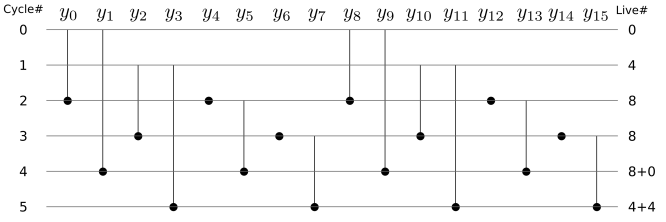

Figure 10: Linear lifetime chart for the variables $y_{(0)}, y_{(1)}, ..., y_{(15)}$ for a 16-point FFT architecture.

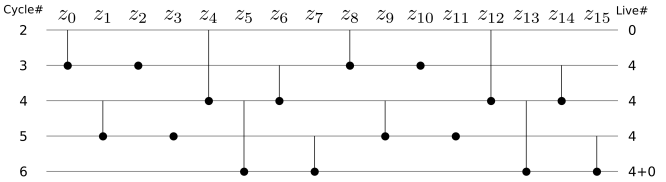

Figure 11: Linear lifetime chart for the variables $z_{(0)}, z_{(1)}, ..., z_{(15)}$ for a 16-point FFT architecture.

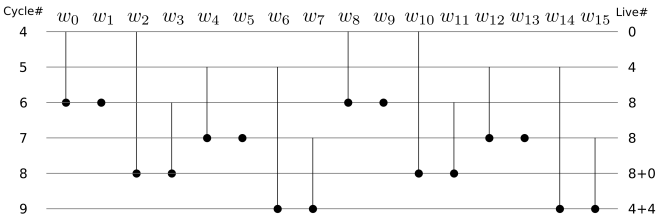

Figure 12: Linear lifetime chart for the variables $w_{(0)}, w_{(1)}, ..., w_{(15)}$ for a 16-point FFT architecture.

The register allocation tables for the lifetime charts are shown in Fig. 13, 14 and 15 for stage 1, 2 and 3 respectively. In the Fig. 16 and 17 is shown the designations of registers for the stage 1 and 2 respectively used in the allocation tables, the designation for stage 3 are similar for stage 1. The folded architecture in Fig. **??** is synthesized using the folding equations and the register allocation tables. The dataflow for each stage can be see in Tab. I, the control signal for stage 1 and 2 can be implemented by dividing the clock signal to 4 and 2 respectively, for stage 3 the control signal is the same that the stage 1.

The inputs of each folding node are represented with a matrix where the values in the same column are data that flow in parallel and values in the same row flow through the same path in consecutive clock cycles. The first two rows represents the inputs of the superior BF and the others two represents the

	I/P	R1	R2	R3	R4	R5	R6	R7	R8
0	y0,y8,y1,y9								
1	y2,y10,y3,y11	y1		y0		y9		y8	
2	y4,y12,y5,y13	y3	y1	y2	y0	y11	y9	y10	y8
3	y6,y14,y7,y15	y5	y3	y1	y2	y13	y11	y9	y10
4		y7	y5	y3	y1	y15	y13	y11	y9
5			y7		y3		y15		y11

Figure 13: Register allocation table for the data represented in 10

	I/P	R1	R2	R3	R4
2	z0,z4,z8,z12				
3	z2,z6,z10,z14	z4	z0	z12	z8
4	z1,z5,z9,z13	z6	z4	z14	z12
5	z3,z7,z11,z15	z5	z1	z13	z9
6		z7	z5	z15	z13

Figure 14: Register allocation table for the data represented in 11

input of the inferior BF. The same criteria is used for represent the constants of rotators, where each number k of the matrix represent a multiplication by W_N^k.

As we can see in Fig. **??**, we suppose that the inputs and output are not ordered, to order these variables extra logic are needed, using more registers and multiplexers.

The different types of rotators used in Fig. **??** are shown in Fig. 18, the description of each of them can be found below.

- Trivial rotator: They can be carried out by interchanging the real and imaginary components and/or changing the sign of the data.
- Constant CSD rotator: They can be carried out by interchanging the real and imaginary components and a multiplication by a unique constant fractional number, in this case we will use a CSD multiplier to perform the area utilized.

	I/P	R1	R2	R3	R4	R5	R6	R7	R8
4	w0,w2,w8,w10								
5	w4,w6,w12,w14	w2		w0		w10		w8	
6	w1,w3,w9,w11	w6	w2	w4	w0	w14	w10	w12	w8
7	w5,w7,w13,w15	w3	w6	w2	w4	w11	w14	w10	w12
8		w7	w3	w6	w2	w15	w11	w14	w10
9			w7		w6		w15		w14

Figure 15: Register allocation table for the data represented in 12

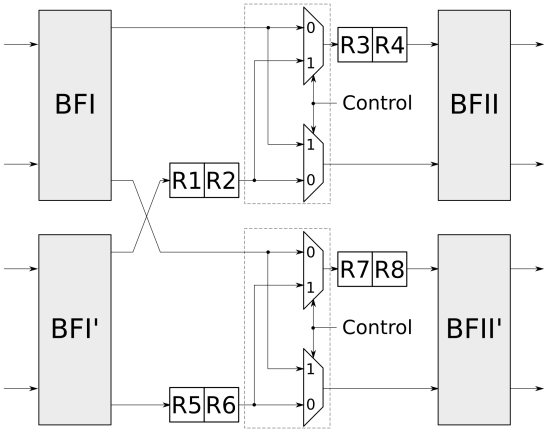

Figure 16: Registers names used in Fig. 13 for stage 1.

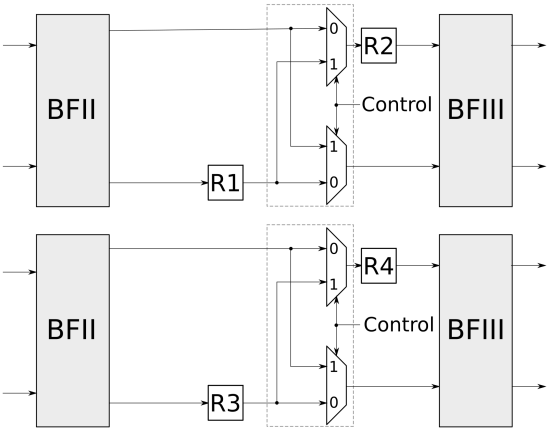

Figure 17: Registers names used in Fig. 14 for stage 2.

- General rotator: They can be carried out by interchanging the real and imaginary components and/or a multiplication by more than one constant fractional numbers, in this case we will use a general multiplier.

IV. 4-Parallel radix-2^3 128-Points

We can deduce the folding architecture for 128 points following the same method than used with the 4-Parallel radix-2^3 16-Points. The DFG and folding set used can be shown in Fig. **??** and Table II. The architecture can be seen in Fig. **??**.

A. Implementation of a 128-point FFT

In this section we are going to begin with the process of how accomplish the implementation of our 4-parallel architecture for the computation of 128-point *radix*-2^3 DIF complex FFT. First, we wrote a *MATLAB* simulator to

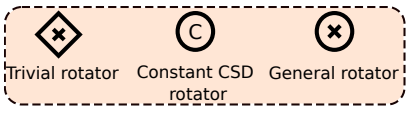

Figure 18: Symbols used for the different types of rotators

#Cycle	Stage 1		Stage 2		Stage 3	
	Dataflow	Control	Dataflow	Control	Dataflow	Control
0	$y0 \rightarrow R3$ $y8 \rightarrow R7$	0	$z0 \rightarrow R2$ $z8 \rightarrow R4$	0	$w0 \rightarrow R3$ $w8 \rightarrow R7$	0
1	$y2 \rightarrow R3$ $y10 \rightarrow R7$	0	$(z2, z10) \rightarrow i/p$ $R1 \rightarrow R2,\ R3 \rightarrow R4$	1	$w4 \rightarrow R3$ $w12 \rightarrow R7$	0
2	$(y4, y12, R4) \rightarrow i/p$ $R2 \rightarrow R3,\ R6 \rightarrow R7$	1	$z1 \rightarrow R2,\ z9 \rightarrow R4$ $R1 \rightarrow i/p,\ R3 \rightarrow i/p$	0	$(w1, w9, R4) \rightarrow i/p$ $R2 \rightarrow R3,\ R6 \rightarrow R7$	1
3	$(y6, y14, R4) \rightarrow i/p$ $R2 \rightarrow R3,\ R6 \rightarrow R7$	1	$(z3, z11) \rightarrow i/p$ $R1 \rightarrow R2,\ R3 \rightarrow R4$	1	$(w5, s9, R4) \rightarrow i/p$ $R2 \rightarrow R3,\ R6 \rightarrow R7$	1
4	$(R2, R4) \rightarrow i/p$	0	$R1 \rightarrow i/p,\ R3 \rightarrow i/p$	0	$(R2, R4) \rightarrow i/p$	0
5	$(R2, R4) \rightarrow i/p$	0	$R1 \rightarrow R2,\ R3 \rightarrow R4$	1	$(R2, R4) \rightarrow i/p$	0

Table I: Dataflow and mux control for each stage based on registers showed in Figure 16 and 17.

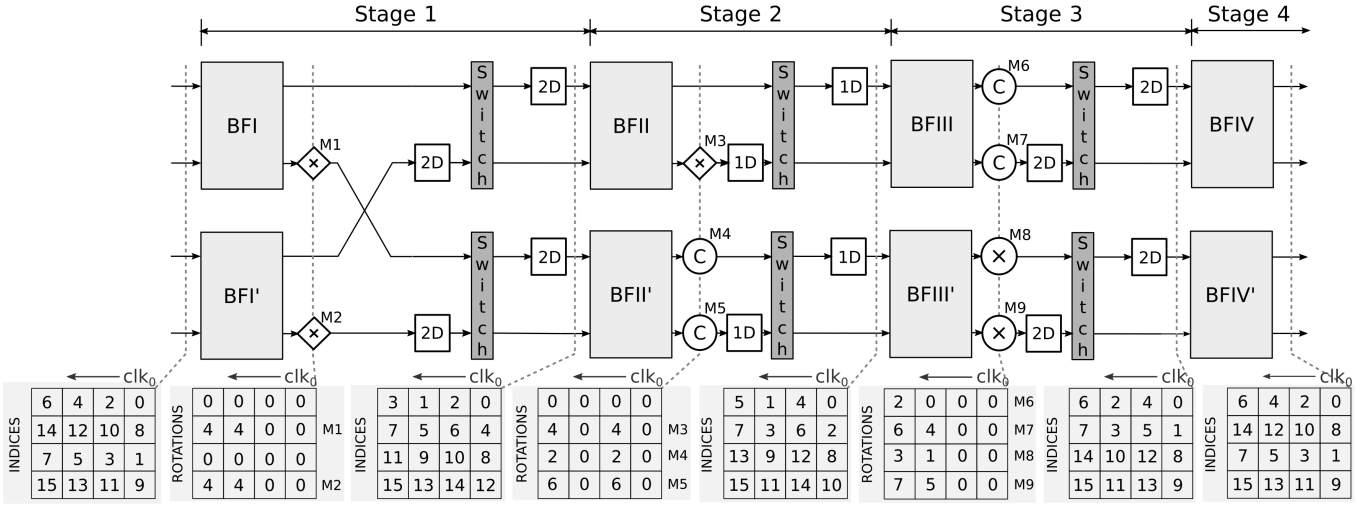

Figure 19: Folding architecture for the computation of a radix-2^3 16-point DIF complex FFT.

Table II: Folding set for the DFG showed in Fig. **??**

A	A0	A2	A4	A6	A8	A10	A12	A14	A16	A18	A20	A22	A24	A26	A28	A30
	A32	A34	A36	A38	A40	A42	A44	A46	A48	A50	A52	A54	A56	A58	A60	A62
A'	A1	A3	A5	A7	A9	A11	A13	A15	A17	A19	A21	A23	A25	A27	A29	A31
	A33	A35	A37	A39	A41	A43	A45	A47	A49	A51	A53	A55	A57	A59	A61	A63
B	B1	B3	B5	B7	B9	B11	B13	B15	B17	B19	B21	B23	B25	B27	B29	B31
	B0	B2	B4	B6	B8	B10	B12	B14	B16	B18	B20	B22	B24	B26	B28	B30
B'	B33	B35	B37	B39	B41	B43	B45	B47	B49	B51	B53	B55	B57	B59	B61	B63
	B32	B34	B36	B38	B40	B42	B44	B46	B48	B50	B52	B54	B56	B58	B60	B62
C	C16	C18	C20	C22	C24	C26	C28	C30	C1	C3	C5	C7	C9	C11	C13	C15
	C17	C19	C21	C23	C25	C27	C29	C31	C0	C2	C4	C6	C8	C10	C12	C14
C'	C48	C50	C52	C54	C56	C58	C60	C62	C33	C35	C37	C39	C41	C43	C45	C47
	C49	C51	C53	C55	C57	C59	C61	C63	C32	C34	C36	C38	C40	C42	C44	C46
D	D8	D10	D12	D14	D16	D18	D20	D22	D24	D26	D28	D30	D1	D3	D5	D7
	D9	D11	D13	D15	D17	D19	D21	D23	D25	D27	D29	D31	D0	D2	D4	D6
D'	D40	D42	D44	D46	D48	D50	D52	D54	D56	D58	D60	D62	D33	D35	D37	D39
	D41	D43	D45	D47	D49	D51	D53	D55	D57	D59	D61	D63	D32	D34	D36	D38
E	E4	E6	E8	E10	E12	E14	E16	E18	E20	E22	E24	E26	E28	E30	E1	E3
	E5	E7	E9	E11	E13	E15	E17	E19	E21	E23	E25	E27	E29	E31	E0	E2
E'	E36	E38	E40	E42	E44	E46	E48	E50	E52	E54	E56	E58	E60	E62	E33	E35
	E37	E39	E41	E43	E45	E47	E49	E51	E53	E55	E57	E59	E61	E63	E32	E34
F	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30	F1
	F3	F5	F7	F9	F11	F13	F15	F17	F19	F21	F23	F25	F27	F29	F31	F0
F'	F34	F36	F38	F40	F42	F44	F46	F48	F50	F52	F54	F56	F58	F60	F62	F33
	F35	F37	F39	F41	F43	F45	F47	F49	F51	F53	F55	F57	F59	F61	F63	F32
G	G3	G5	G7	G9	G11	G13	G15	G17	G19	G21	G23	G25	G27	G29	G31	G0
	G2	G4	G6	G8	G10	G12	G14	G16	G18	G20	G22	G24	G26	G28	G30	G1
G'	G35	G37	G39	G41	G43	G45	G47	G49	G51	G53	G55	G57	G59	G61	G63	G32
	G34	G36	G38	G40	G42	G44	G46	G48	G50	G52	G54	G56	G58	G60	G62	G33

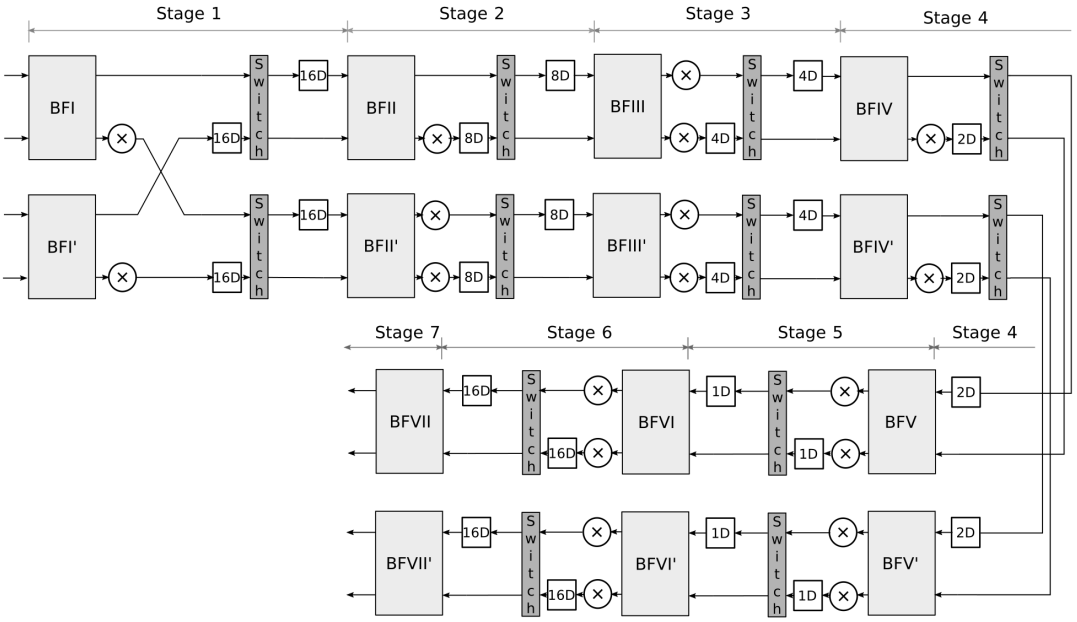

Figure 20: Folding architecture for the computation of a radix-2^3 128-point DIF complex FFT.

validate the operation of our design and the design presented in [8], [9]. Therefore, we can have a reference architecture for compare the design that we deduce. After that, we take all these information to generate a Synthesizable *Verilog* code with different levels of optimizations with a powerful tool like *Synopsys* to get a final design integrated of Standard Cells @45nm.

B. Floating and Fixed point Simulator

The input signal Fig. 22 to our architecture will be a mixture of sinusoid signals with two frequency values, theses signal will be normalizing to the unity and in this way we can have a test bench design.

$$x'[n] = cos(2\pi f_1 n T_s) + cos(2\pi f_2 n T_s) \tag{7}$$
$$x[n] = x'[n]/max\{x'[n]\}$$

where $f_1 = 100Hz$, $f_2 = 1000Hz$ and T_s is the sampled period.

In each stage of the Fig. 21, input samples that propagate stage by stage will be carefully quantized with the purpose of getting a high SQNR (Signal to Quantization Noise Ratio).

$$SQNR_{dB} = 10log_{10}\left(\frac{Var\{Signal_{FloatPoint}\}}{Var\{Signal_{FloatPoint} - Signal_{FixedPoint}\}}\right)$$

SQNR computation represents the logarithmic relationship between float signal variance over error variance from an signal given.

The input signal $x[n]$ is quantized with a value of $S(10, 9)$, that representation means a number signed (S) with 10 total bits and 9 fractional bits. The value calculated of SQNR for the input is 56.9dB. Following the same steps, we can compute the SQNR for the *twiddle* factors and the architecture's output.

Twiddles factor are quantized with a relation of $S(11, 9)$ and the complex output signal $X[k]$ with $S(22, 15)$. Output quantization for the real part is 46.8dB and for the imaginary part 47.3dB. In general, a value of quantization close to 50dB is a good approximation. Out signal from our *MATLAB* fixed-point model from the architecture in Fig. 21 the output signal is given in Fig. 23, that represents a first approach in our calculation of a DFT without optimization.

The combination between latencies (delays registers) and switches in all stages in Fig. 21 is the equivalent circuit shows in Fig. 24, that is used to appropriately order samples in each butterflies input.

Whole elements on the architecture such as multipliers and butterflies are complex as Fig. 25 and 26. On an implementation is essential divide the signal in its real and imaginary part with the purpose of process them independently. A *general rotator* (full complex multiplier) generate a real and imaginary component that is composed by a real multiplication an one addition, but a *trivial rotator* only change the components of a complex signal. These relations are important at the moment of doing the quantization process to ensure an appropriate quantity of bits.

C. Verilog (HDL) Model

In this subsection we are going to talk about the different design instances to model the DFT in hardware. We made four design implementations with the purpose of have a global view of optimization levels to achieve the requested *Timing* at a working frequency of 500MHz obtained from the synthesis processes.

In the first instance of design was made from the base design showed in Fig. 21, this hardware model has butterflies modules that work in combination with multipliers and each multiplier has associated a memory block that contains

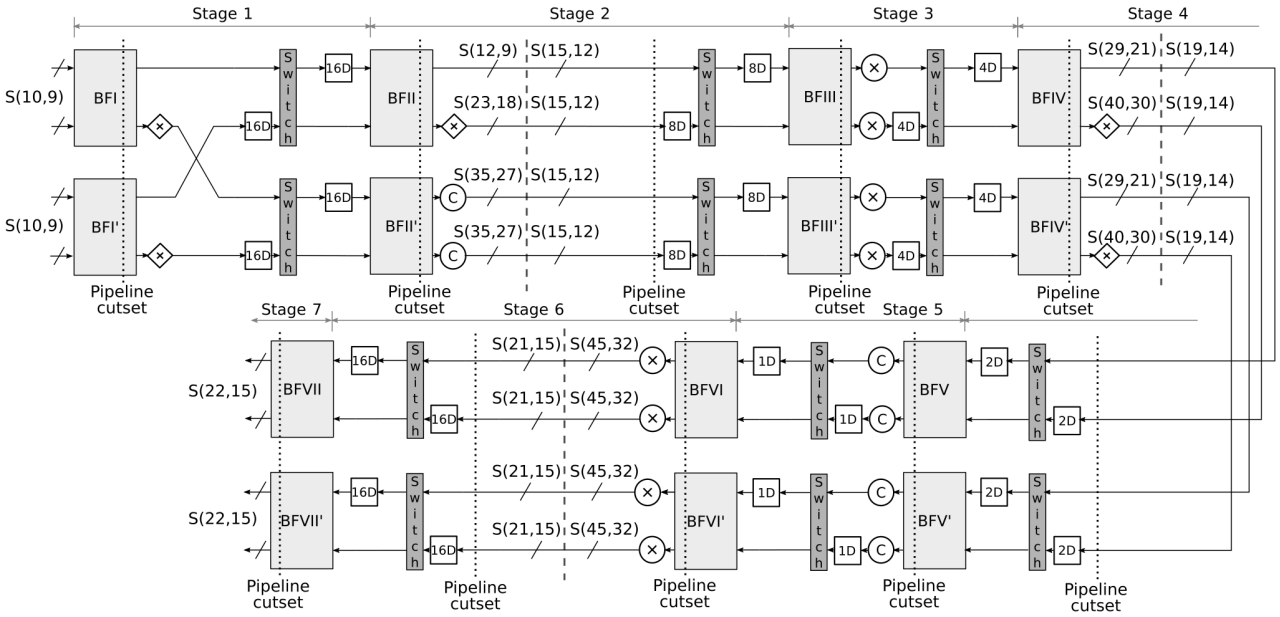

Figure 21: Quantization for a 128-point 4-parallel complex FFT architecture

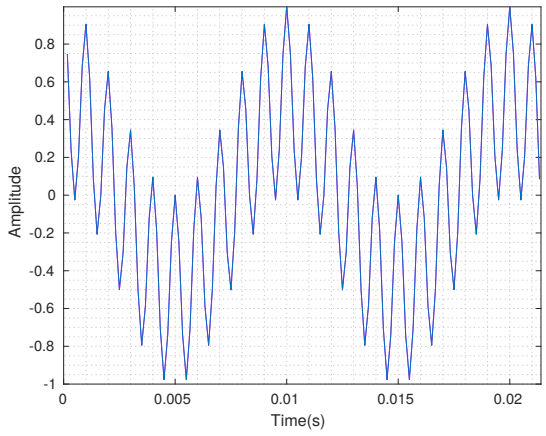

Figure 22: Input signal $x[n]$ in time domain

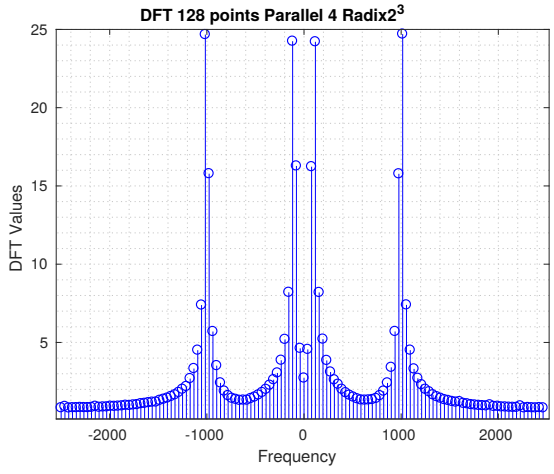

Figure 23: Output samples, absolute value vs frequency $|X[k]|$

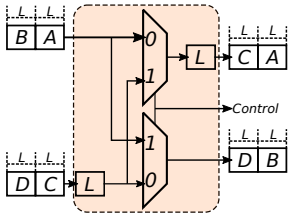

Figure 24: Circuit for data shuffling

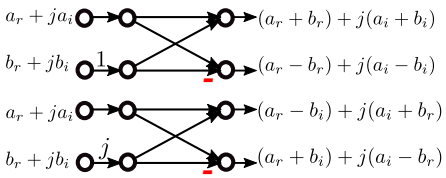

Figure 25: Complex butterfly

twiddle factors. In this first approach all multipliers are *full*, each stage has a control module that enable and disable the inversion of the switching block, this control signal is also sent to the multipliers to work synchronously with the switching. To avoid the bit growth generated by the addition and multiplication, a quantization block is necessary, the quantizer consisted of saturation and truncate operations.

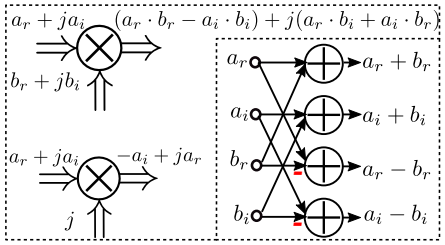

Figure 26: Complex multiplier and complex adder

Table III: Design instance 1. Timing-Area-Power Report at 500MHz

Point	Path(ns)
data arrival time	5.60
clock CLK (rise edge)	2.00
clock network delay (ideal)	2.00
library setup time	1.95
data required time	1.95
data arrival time	-5.60
slack (VIOLATED)	-3.65

Logical Elements	
Number of ports	1228
Number of nets	112376
Number of cells	102726
Number of combinational cells	95310
Number of sequential cells	7404
Number of macros/black boxes	0
Number of buf/inv	27817
Combinational area	291201.116637
Buf/Inv area	44892.767764
Noncombinational area	58830.508095
Total cell area	350031.624731

Power Group	Internal	Switching	Leakage	Total Power
io pad	0.0000	0.0000	0.0000	0.0000
clock network	34.8220	603.2268	1.4573e+06	639.6328
register	54.2228	0.2699	4.0540e+05	54.8980
sequential	0.0000	0.0000	0.0000	0.0000
combinational	0.2884	0.7304	1.5016e+04	1.0337
Total	89.333mW	604.227mW	1.877e+06nW	695.564mW

In the second place with the goal of minimize the *critical path* in our design, we add *pipelined* registers after quantization blocks. In the third case of implementation incorporate *trivial* multipliers, imaginary multiplication by -1j, with this kind of operation we can reduce the size of the binary word.

In the fourth level of optimization showed in Fig. **??**, we place inside each butterfly blocks an internal pipelined to improve the required timing. Lastly, the full multipliers from stage two and five are modified to efficiently work with constant coefficients, these new multipliers are *CSD* because the twiddle factors in these stages are always $-je^{-j\frac{\pi}{4}}$ and $e^{-j\frac{\pi}{4}}$.

V. Results

All designs were implemented with different levels of optimization to accomplish the desired working frequency. The design instances were synthesized by Synopsys tool in order to build an interconnection of Standard Cells to get and generate a complete set of *timing-area-power report* showed in Table III, IV and V. According to the first results, data arrival time is greater than minimum period established 2ns and we can detected that there is a time Violated, with this first approach we decide make a set of optimizations to folding architecture as a pipelined cutset between stages of DFT. In this way as we can see in Table IV, we reduced power, area and timing.

In contrast with the first instance of design, we found in the last case of implementation a slack of time equal to zero, that means the clock period satisfy the time constraint at 500MHz.

Table IV: Design instance 4. Timing-Area-Power Report at 500MHz

Point	Path(ns)
data arrival time	1.94
clock CLK (rise edge)	2.00
clock network delay (ideal)	2.00
library setup time	1.94
data required time	1.94
data arrival time	-1.94
slack (MET)	0.00

Logical Elements	
Number of ports	2315
Number of nets	75713
Number of cells	66284
Number of combinational cells	58017
Number of sequential cells	8240
Number of macros/black boxes	0
Number of buf/inv	14789
Combinational area	195877.841872
Buf/Inv area	24429.880240
Noncombinational area	64463.046570
Total cell area	260340.888442

Power Group	Internal	Switching	Leakage	Total Power
io pad	0.0000	0.0000	0.0000	0.0000
clock network	18.1655	581.6307	7.8320e+05	600.6672
register	58.2275	0.2873	4.4422e+05	58.9591
sequential	0.0000	0.0000	0.0000	0.0000
combinational	0.7768	0.9958	1.5970e+05	1.9322
Total	77.169mW	582.913mW	1.387e+06nW	661.558mW

Table V: Design instance 5. Timing-Area-Power Report at 500MHz

Point	Path(ns)
data arrival time	1.94
clock CLK (rise edge)	2.00
clock network delay (ideal)	2.00
library setup time	1.94
data required time	1.94
data arrival time	-1.94
slack (MET)	0,00

Logical Elements	
Number of ports	2315
Number of nets	75713
Number of cells	66284
Number of combinational cells	58017
Number of sequential cells	8240
Number of macros/black boxes	0
Number of buf/inv	14789
Combinational area	195877.841872
Buf/Inv area	24429.880240
Noncombinational area	64463.046570
Total cell area	260340.888442

Power Group	Internal	Switching	Leakage	Total Power
io pad	0.0000	0.0000	0.0000	0.0000
clock network	18.1655	581.6307	7.8320e+05	600.6672
register	58.2275	0.2873	4.4422e+05	58.9591
sequential	0.0000	0.0000	0.0000	0.0000
combinational	0.7768	0.9958	1.5970e+05	1.9322
Total	77.169mW	582.913mW	1.387e+06nW	661.558mW

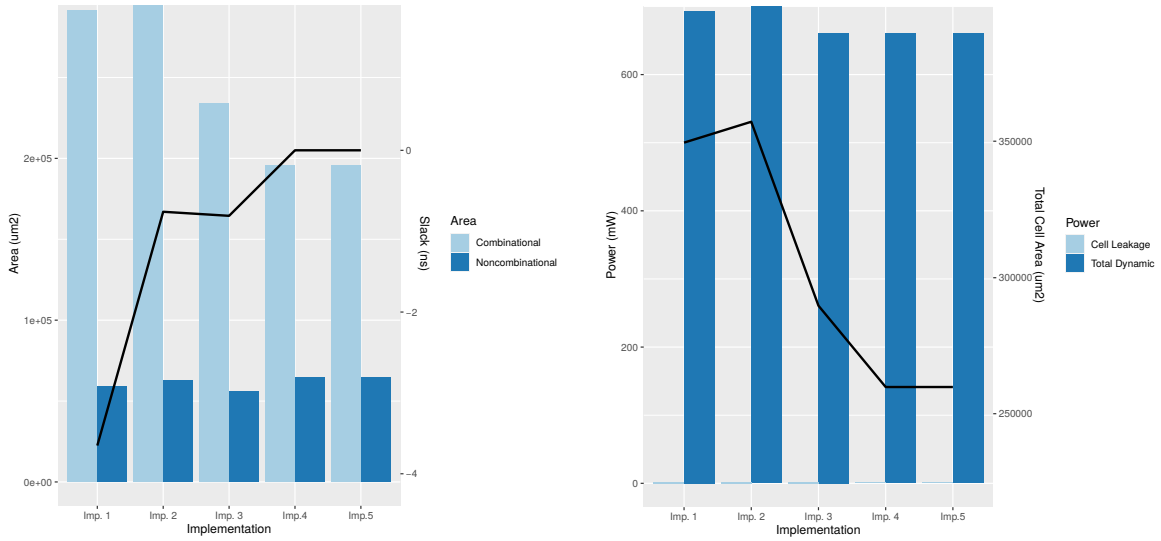

Figure 27: Timing-Area-Power evolution.

VI. Conclusions

This work has presented a VLSI Implementation of a Pipelined 128 points 4-Parallel radix-2^3 FFT Architecture via Folding Transformation. The Fourier Transform without folding technique processes 128 entry points of samples which are ordered but its output are disordered and it is necessary a reordering circuit. In the other hand a Fourier Transform with folding technique also needs a reordering circuit in the input. This circuit present an additional area and power consumption to be considered in the final design.

The folding transformation applied reduce significatively (equal to the folded factor, i.e. 64 times) the number of functional units, and therefore the silicon area, at the expense of increasing the computation time by the same factor. Therefore is necessary to add pipeline cutsets and a careful quantization in order to achieve the required sample period with an acceptable SNR.

Folding technique, in this case, results in an architecture that uses a large number of register, to avoid this is necessary to apply a register minimization technique and allocate data to these registers. The number of registers applying this method is reduced significatively, from to . This results in a final design with less area and power consumption.

The number of DFT points is related with the radix-base, we can write as *radix*-8, this representation can be expressed as *radix*-2^3 that means we can decompose in smaller radixes with main base-2, an advantage of using a hight radix is reduced the number of multipliers at expense of increase the routing.

Our design has implemented a quantizer block which works with saturation and truncated, the round method was not used because we reached a high SQNR. Lastly a series of optimization were necessary to accomplish the required frequency. The DFT implementation without any optimization level got to work at 166MHz, applying pipelines cutsets and quantization blocks, the final architecture is implementable at the required clock frequency (500 MHz) at the cost of incrementing the numbers of sequential cells. Finally, with the CSD multipliers, is possible to reduce the combinational cells significatively.

References

[1] R. Thapa, S. Ataei, and J. E. Stine, "WIP. Open-source standard cell characterization process flow on 45 nm (FreePDK45), 0.18 μm, 0.25 μm, 0.35 μm and 0.5 μm," in *2017 IEEE International Conference on Microelectronic Systems Education (MSE)*. Lake Louise, AB, Canada: IEEE, May 2017, pp. 5–6. [Online]. Available: http://ieeexplore.ieee.org/document/7945072/

[2] Shousheng He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," in *1998 URSI International Symposium on Signals, Systems, and Electronics. Conference Proceedings (Cat. No.98EX167)*. IEEE, pp. 257–262. [Online]. Available: http://ieeexplore.ieee.org/document/738077/

[3] J. G. Proakis and D. G. Manolakis, "DIGITAL SIGNAL PROCESSING," p. 1033.

[4] A. V. Oppenheim and R. W. Schafer, *Tratamiento de señales en tiempo discreto, tercera edición*. Pearson Educación, OCLC: 843859190.

[5] L. Jia, Y. Gao, and H. Tenhunen, "Efficient VLSI implementation of radix-8 FFT algorithm," p. 4.

[6] M. Ayinala, M. Brown, and K. K. Parhi, "Pipelined Parallel FFT Architectures via Folding Transformation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1068–1081, Jun. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/5776727/

[7] K. K. Parhi, *VLSI Digital Signal Processing Systems. Design and implementation*. JOHN WILEY & SONS, INC., 1999, ch. Folding Transformation, pp. 151–163.

[8] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined Radix-2^{k} Feedforward FFT Architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 23–32, Jan. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6118316/

[9] M. Garrido, S.-J. Huang, and S.-G. Chen, "Feedforward FFT hardware architectures based on rotator allocation," vol. 65, no. 2, pp. 581–592. [Online]. Available: http://ieeexplore.ieee.org/document/8010853/

[10] Vladimir Stojanović, "Fast Fourier Transform: VLSI Architectures" , 973 Communication System Design – Spring 2006 Massachusetts Institute of Technology, Lecture 10.