Implementing FFT-based Digital Channelized Receivers on FPGA Platforms

Miguel A Sanchez, Mario Garrido, Marisa Lopez-Vallejo, Jesus Grajal

Journal Article

Implementing FFT-based Digital Channelized Receivers on FPGA Platforms

Miguel A. Sánchez[†], Mario Garrido[‡], Marisa López-Vallejo[†] and J. Grajal[‡]

[†]Dept. de Ingeniería Electrónica
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid (Spain)
{masanchez,marisa}@die.upm.es

[‡]Dept. Señales Sistemas y Radiocomunicación
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid (Spain)
{mgarrido,jesus}@gmr.ssr.upm.es

Abstract

This paper presents an in-depth study, implementation, and validation of Fast Fourier Transform (FFT) pipelined architectures suitable for broadband digital channelized receivers. The implementation of the FFT algorithm has as a first goal to maximize throughput and to optimize area on Field-Programmable Gate Array (FPGA) platforms. Feedback and feedforward architectures have been analized regarding key design parameters: Radix, bitwidth, number of points and stage scaling. Moreover, a simplification of the FFT algorithm, the monobit FFT, has been implemented in order to get faster real time in broadband digital receivers. To support the signal processing designer when implementing this kind of systems on FPGA platforms we have developed a design space exploration tool for FFT architectures.

1 Introduction

Following the Moore's Law, current sub-micron technologies have allowed extraordinary integration densities in digital circuits. However, as processes scale down, incertainty increases (voltage, temperature, noise, coupling etc.), there is higher interconnect delay and the design process is more complicated, especially for ASICs (Application Specific Integrated Circuit), where margins are too tight and the *"time to market"* preassure is tremendous. Moreover, masks costs have reached a critical point, dominating the manufacturing process and requiring high finantial risk.

In this context, Field-Programmable Gate Arrays (FPGAs) offer significant advantages at a suitable low cost [1]. First, the well-known flexibility of FPGAs allows the implementation of different generations of a given application and provide space to designers to modify implementations until the very last moment, or even correct mistakes once the product has been released. Second, the verification of a design mapped into an FPGA is very simple, contrasting with the huge verification effort requited by ASICs. Finally, even though FPGAs are not so efficient as ASICs in terms of performance, area or power, it is true that nowadays they can provide better performance than standard or digital signal processor (DSP) based systems. This fact, in conjunction with the enormous logic capacity allowed by today's technologies, makes FPGAs an attractive choice for implementation of complex digital systems, as they are signal processing applications. Moreover, with their newly acquired digital signal processing capabilities [2], FPGAs are now expanding their traditional prototyping roles to help offload computationally intensive digital signal processing functions from the processor.

In the signal processing field, Electronic Warfare (EW) receivers are a good example of complex systems with hard constraints. The requirements for EW receivers are wide band frequency coverage, high sensitivity and dynamic range, high probability of intercept, simultaneous signal detection, frequency resolution and real time operation. A classical receiver which accomplishes these requirements is a channelized receiver [3] which separates signals according to their frequencies. Recent advancements in Analog-to-Digital Converters (ADC) technology and in the speed of digital processors have made it possible to design relatively wide band digital channelized receivers [4–7]. The use of digital channelization in comparison with the analog approach allows to improve the imbalance between filters, which is one of the fundamental problems in analog receivers (digital circuitry is inherently reproducible, reliable, and accurate). However, broadband digital channelized receivers, mainly based on Fast Fourier Transform-related processing, require intensive computation for real time applications. FPGAs can play a key role in the implementation of these receivers, changing the traditional method of implementing military devices using *commercial off-the-shelf* (COTS) platforms, which conventionally comprise arrays of DSP microprocessors. FPGAs have grown over the past decade to the point where broadband real-time operation digital channelized receivers can be implemented on a single FPGA device.

Typically, the system throughput of many signal processing algorithms can be improved by exploiting concurrency in the form of parallelism and pipelining [8]. FPGAs allow for true parallel processing, supporting multiple simultaneous threads of execution. This provides significant opportunities to exploit FPGAs to create very high performance processing circuits, through the use of multiple processing elements operating in a concurrent manner. FPGAs provide room for two-dimensional parallel architectures, where multiple parallel processing threads can be implemented in a pipelined manner.

As has been described, FPGAs offer new possibilities to the system designer, but additional support is required because the design cycle presents now more degrees of freedom. Design space exploration includes typical hardware variables such as area, clock frequency or power dissipation, together with classical signal processing issues: Throughput, detection performance, dynamic range, etc. Low level hardware details should be hidden to the system designer by high level design exploration tools, as proposed in this work.

In this paper we present an in-depth study, implementation, and validation of FFT pipelined architectures suitable for broadband digital channelized receivers with continuous flow of input samples. Our purpose is twofold. First, to accelerate the execution of FFT algorithms using FPGA platforms. Second, to allow the system designer to explore a broad variety of posibilities in a quick and easy way.

The structure of the paper is the following. Section 2 presents a review of related works on hardware implementations of the FFT and support tools. A general description of the FFT algorithm and its basic implementation is described in section 3, while section 4 describes the selected architectures in this work: Feedback and feedforward. Next, a monobit implementation will be introduced and compared to the classical FFT in section 5. Section 6 describes the high level design tool developed for design space exploration. Experimental results for the implemented architectures are analyzed in sections 7 and 8. Finnaly, the implementation of EW receivers based on the proposed architectures will be described in section 9 and some conclusions will be drawn.

2 Related work

Important work has been carried out on hardware FFT architectures, but this work represents a partial approach to the problem. As was noted before, pipelined architectures provide high throughput and allow working with continuous data flow, what results in real time processing rates. Basic pipelined implementations can be classified as feeback and feedforward architectures [9] and will be deeply described in this paper. A different approach to this kind of architectures is shown in [10], where a complex architecture that mixes feedback and feedforward implementations is presented to maximize the use of hardware and to reduce the size of the memory for the rotation angles.

In-place architectures [11] are based on the use of a single memory with several blocks to read and store samples in an iterative way, providing a slow structure with no pipeline. In [12] we can see a variation of in-place architectures based on the use of buffers instead of memories. This structure may provide a higher throughput but it requires more hardware components.

Another important issue to be taken into account when studying FFT implementations is the radix, which can play a key role in the performance-area trade-off. The most common value of radix is 4 [9–11, 13], because the number of rotators required is minimum. Actually, radix 8 requires extra rotators in the butterflies, what makes more expensive its implementation when considering hardware area. Other approaches use *Radix* 2^2 [14, 15], which is quite similar to radix 4, and whose only difference is that the radix 4 butterfly is implemented using two radix 2 butterflies.

An added value of any design is the possibility of parameterizing the architecture. This is one of the key goals of the work described here. Previous works have focused on allowing the variation of simple parameters, as they are the number of bits of the rotation angles and the input and output samples [14]. Other schemes can modify the radix (in order to change the parallelism) or the kind of memory that is used [16]. In this sense, the architectures described in this paper allow the modification of all those parameters (number of points, input bitwidth, truncation, radix...) in order to obtain a versatile design that can be easily adapted to the user requirements.

When reviewing FPGA-based implementations, there is a large number of FPGA-based FFT designs. However, the published designs usually focus on explaining their implemented hardware architecture [10] and, at most, the area and speed of the circuit [14], while the performance accuracy is not analyzed. A recent work [17] describes an area-efficient architecture based on the use of a CORDIC operator [18] to perform the rotations, as we do in our work, but it does not target parallel implementations. A closer approach is the one presented in [19], but no comparison to other architectures is provided. In [20] a parallel architecture is presented based on the use of a parallel multiplier and extended exploitation of the memory structure of the FPGA, lacking the generality of the architectures we present here. In [13] an architecture to process a high number of points is presented. However, the speed processing of this approach is significantly limited by the use of external memory. Other recent works pay special attention to particular design characteristics, as is the case of power minimization [16].

In this work we will provide a global view of the problem, paying attention not only to the design of the FFT architectures, but also analyzing the performance they provide from the detection point of view.

3 The FFT algorithm

The N-point Discrete Fourier Transform (DFT^N), $X_N(k)$ for a given sequence $x_N(n)$ is defined as:

$$X_N(k) = \sum_{n=0}^{N-1} x_N(n) \cdot W_N^{-nk} \;\; with \;\; k = 0, 1, \ldots N-1 \tag{1}$$

where $W_N^{-nk} = exp(-j\frac{2\pi kn}{N})$ is the so called DFT kernel.

The Fast Fourier Transform (FFT) is based on the decomposition of a sequence of the DFT into lower order computations, what results in a reduction in the number of operations. In this way the complexity of the computations is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(Nlog_2N)$.

There are many algorithms that compute the FFT. One of the most widely used was proposed by Cooley-Tukey [21] and is based on the successive decomposition of a DFT with length N into R DFTs with length N/R. R, known as *Radix*, is a power of 2 and as consequence the length of the transform will have a set of discrete values $N = R^S$, where S corresponds to the set of successive decompositions required for the whole transform (S stages). The decomposition continues until the length of the sub-DFT matches the radix (the lowest order sub-DFT).

There are two basic approaches to implement the algorithm: Decimation in time (DIT) or decimation in frequency (DIF). The difference between them is the way the algorithm performs the decomposition of the DFT into lower order DFTs, resulting in a different sequence of operations. For example, in the case of radix 2, the FFT of a sequence with length N ($N = 2^S$) can use the decomposition of this sequence into two sets with odd and even samples (DIT implementation) or two sets with the first and the second half of the samples (DIF implementation).

In the case of a DIF implementation with radix 2 the resulting decomposition is the following:

$$\begin{aligned} X_N(2k') &= \sum_{n=0}^{N/2-1} [[x_N(n) + x_N(n+N/2)] \cdot T_N^0(n)] \cdot W_{N/2}^{-k'n} = \sum_{n=0}^{N/2-1} x_{N/2}^0(n) \cdot W_{N/2}^{-k'n} \;\; with \;\; k' = 0, 1, \ldots N/2-1 \\ X_N(2k'+1) &= \sum_{n=0}^{N/2-1} [[x_N(n) - x_N(n+N/2)] \cdot T_N^1(n)] \cdot W_{N/2}^{-k'n} = \sum_{n=0}^{N/2-1} x_{N/2}^1(n) \cdot W_{N/2}^{-k'n} \;\; with \;\; k' = 0, 1, \ldots N/2-1 \end{aligned} \tag{2}$$

which is the decomposition of the DFT into two sub-DFTs with lower order. Successives decompositions can be carried out until no decomposition is possible. In these equations $T_N^m(n)$ are called *twiddle* terms and follow the expression:

$$T_N^m(n) = \exp \frac{-j2\pi m}{N} n \;\; with \;\; m = 0..(R-1) \tag{3}$$

These terms are not general multiplications because only a discrete set of angles is used and conse-

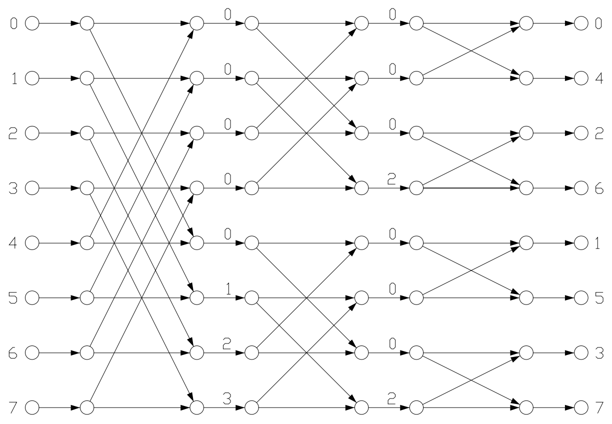


Figure 1: Data flow in a 8-point FFT with radix 2 (DIF implementation).

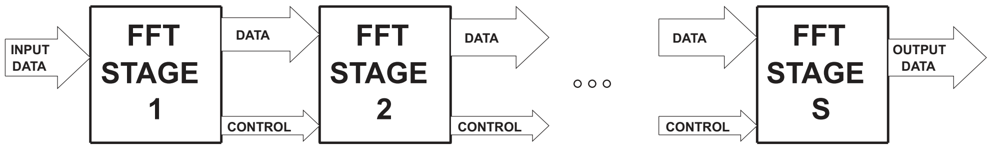


Figure 2: Pipelined implementation of the FFT.

quently they can be implemented as rotations of the vectors defined by the input samples.

Several issues can be outlined when analyzing equations 2. First, the input samples can be complex numbers. Second, the most internal operands of the middle equations (between brackets) represent a mixture of components known as *butterfly*, named from the shape of its flowgram. Third, we need to implement after that mixture the rotation of samples made by the corresponding *twiddle* terms. Once the samples are suitably mixed and rotated we obtain two half length sequences with independent samples, being therefore ready to operate separately. The previous decomposition procedure can be repeatedly applied to both sequences until the sub-DFT with the lowest order is reached (order 2 in this case) and the algorithm is completed. To illustrate how the FFT algorithm works, figure 1 shows the way data are processed in an 8-point DIF FFT with *radix* 2. As can be seen, three stages are required since $log_2(8) = 3$.

An analogous process can be done with a sequence of $N = 4^S$, consequently implemented with *Radix* 4, providing similar results. In this case the DFT is decomposed into four lower order sub-DFT instead of two, requiring a smaller number of decompositions to obtain the lowest order (in this case $log_4 N$).

Theoretically, DIT and DIF implementations only differ in the order followed by the previous operations and the order of samples. The analysis that we have carried out in this work is focused on DIF implementations, but can be inmediately applied to DIT implementations.

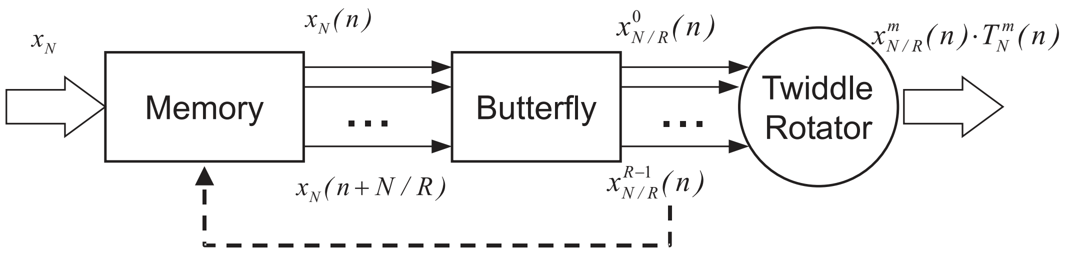


Figure 3: Structure of a stage of the proposed architectures.

4 Hardware Architectures for the FFT

There are many hardware implementations for both DIF and DIT algorithms. For instance, we can choose between serial or parallel arithmetic, or we can select between pipelined or iterative implementations. In general, the designer only focuses on two different goals: optimize area or maximize speed. In this work we present a wider approach in which the design of FFT-based architectures is automated and important issues like area, throughput or power can be easily explored by user defined combinations of parameters like bitwidth, radix, number of stages, etc.

In the case of data oriented applications presenting a continuous flow of samples, the best architectures will be those that potentiate speed instead of area. The implementations that better fit these requirements are both parallel and pipelined architectures, where the processing is performed in several cascaded stages, as can be seen in figure 2. We have chosen two main groups of FFT architectures, representing opposite points in the design space: Feedback (FB) and feedforward (FF) architectures. Architectures with feedback provide the output flow at the clock frequency (one sample per clock cycle), because the feedback structure allows the reuse of some elements present in every stage. This reuse provides a small area implementation. On the other hand, our feedforward structures provide a higher throughput because reuse is not applied and higher concurrency can be obtained, paying the price of a significant area overhead.

The general architecture of a pipelined implementation is based on a set of stages (S in section 3) and each stage performs the decomposition of the input sequence into sub DFTs, which will be implemented in later stages. Every stage is characterized by the *radix*, which also sets the number of required stages to process an input sequence of length N.

There are three basic elements in each stage of both architectures, depicted in figure 3: A memory where data between stages are stored, a butterfly where the mixture of samples is accomplished and finally an element to multiply samples by the corresponding twiddles. The architectures differ in the way these elements are interconnected and how the sample flow is controlled. The dashed line that appears in figure 3 represents the feedback loop of the FB architecture.

Once all stages have been accomplished (finishing the lowest order sub-DFT, which is of size *radix*), the processing can be considered finished, even though there is still a re-ordering phase left. This is an optional task, because output samples are not completely unordered, but they have a known sequence that can be considered as input for the next processing step.

In the following we will describe in detail the basic elements used in both architectures: Butterfly and rotators.

4.1 Butterfly

In pipelined architectures, butterflies can be implemented as a set of $\log_2(R)$ stages of adders and constant multipliers. The simplest implementation of this element is the radix-2 butterfly, which only requires two components: An adder and a subtracter, both complex. In this case, the twiddle terms are trivial and, consequently, there is no need of extra components to perform the rotation after the mixture of samples.

For implementations with other radices, most samples do not need rotations since their corresponding twiddle is 0^o, as is the case of radix-2 computation. Other rotations may be easy to compute, as is the case of radix-4 where the angle to rotate is -90^o and can be implemented as a swap of the sample components with sign change. Other angles like -45^o or -135^o, which are present in radix-8 architectures, can be implemented by two multipliers by real constants. For radix $R > 8$ butterflies, non trivial twiddles appear, with the number of these non-trivial twiddles increasing with the radix. In these cases complex multipliers must be used and the butterfly implementation requires bigger area. Actually, there is an exponential increase of area with the radix, as can be seen in figure 4. In a general way, we can say that the area required by a radix-R butterfly is the area of the basic butterfly multiplied by $R/2 \cdot \log_2(R)$. This value does not consider the area of the pipeline registers or the multipliers. This area increase precludes us to implement FFTs with radix bigger than 8, even though higher radix values reduce the number of stages.

Radix 4 presents the twofold advantage of including simple components and presenting a reduction of the number of stages regarding radix 2 implementations. This combination of advantages makes this value of radix the optimum for most applications.

An additional important issue to consider in the butterfly structure is that for every new stage we should decide the bitwidth to manage the overflow. This is due to the adders included in the butterflies.

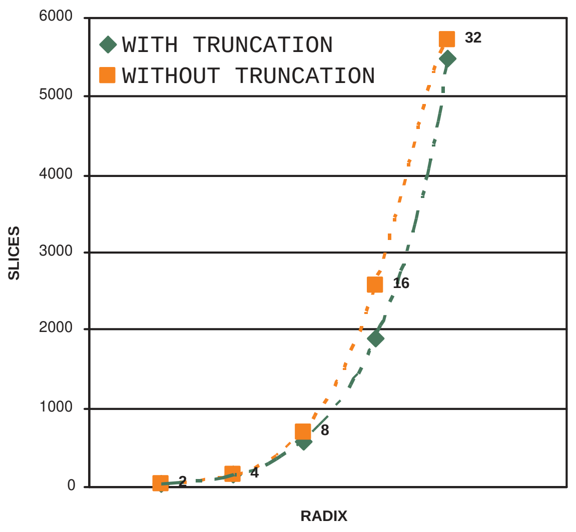


Figure 4: Area of the batterfly vs. radix.

If we keep fixed the bitwidth, the easiest approach is to truncate the output of the adders removing the less significant bit. This divides the output by 2. With this approach the final area is reduced, but the performance of the total FFT is affected by a clear degradation. This fact will be deeply analyzed in sections 7 and 8.

4.2 Rotators

Rotators are critical components in the FFT architecture because of their area. They are mainly composed of a first element that multiplies data by the twiddles and a memory that stores the twiddles. In our case we have implemented the rotator using the algorithm *Cordic* [18]. This algorithm performs the rotation of a complex vector by means of a series of shifts and additions. Every shift rotates the vector components a given angle from a set of elemental angles. This algorithm presents an intrinsic gain of approximately 1.647. Therefore, to keep the dynamic range of the input samples, this element would have to increase the data bitwidth in one bit. As was explained for the butterfly, this extra bit can be truncated after rotation takes place or it can be kept. It is important to remark that overflow is avoided in any case.

The Cordic algorithm takes as inputs both the data coming from the butterfly and the rotation angle. Given that the set of elemental angles that will be used by the twiddles is actually known, we compute in advance the rotations that will be performed by the Cordic algorithm. These values can be stored

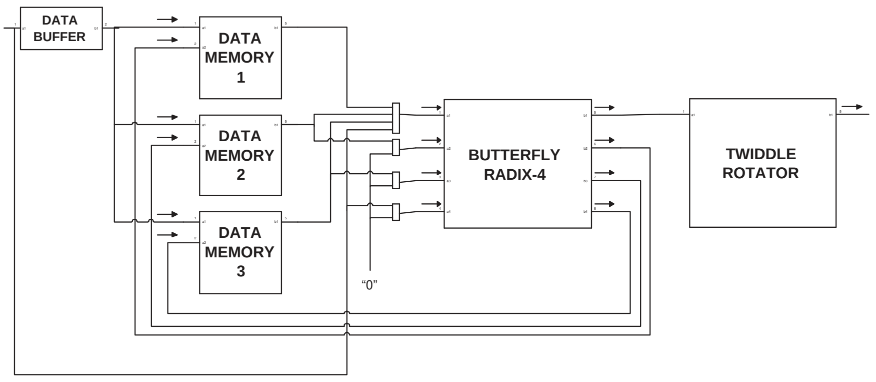


Figure 5: Structure of one stage of the FB architecture (radix 4).

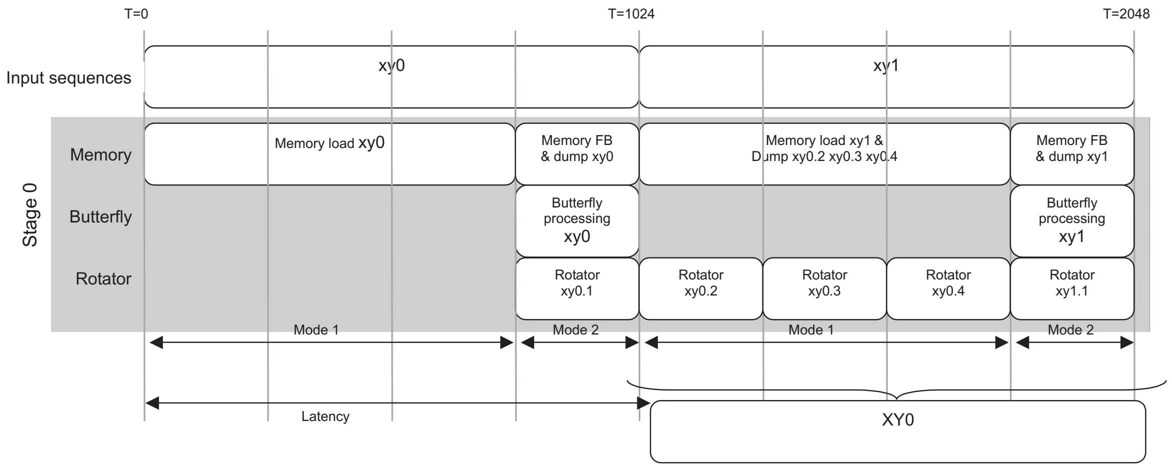


Figure 6: Time evolution for a 1024-point, radix 4, FB architecture.

in the memory instead of the angles, consuming approximately the same memory size, what results in a great improvement in the implementation of the algorithm in terms of area and speed. Finally, we have decomposed the implementation of the Cordic into a set of steps. For every step the vector is rotated by one of the pre-defined elemental angles, allowing the pipelined implementation of the algorithm.

4.3 Feedback Architecture

The feedback (FB) architecture, shown in figure 5, is composed of a first memory that stores the input samples, followed by a butterfly whose output is connected to a single rotator that multiplies by the twiddles. In this implementation, given that the rotator is a shared component, part of the butterfly outputs will be fed back to the memories to allow the use of the rotator all the time. Therefore, there will be two working modes, depicted in figure 6. A first mode is related to the arrival of samples from

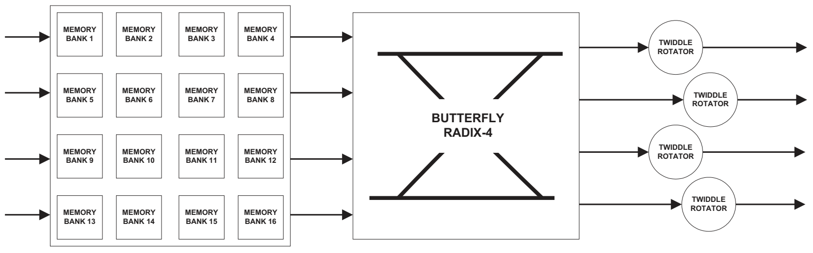


Figure 7: Structure of one stage of the FF architecture (radix 4).

the previous stage while samples coming from previous processing are extracted from memory (mode 1). During mode 2 the samples are processed and simultaneously data coming from the butterfly are stored in the memory because the rotator is busy. This working procedure is illustrated in figure 6, where we can see how two input sequences of 1024 samples ($xy0$ and $xy1$) enter the FB architecture and are sequentially processed by the memory, butterfly and rotator of the first stage. A similar processing is accomplished for the lower order sub-DFTs generated through the different stages ($xy0.1$, $xy0.2$...). Output results are labelled as $XY0$. It is important to remark that the 1024 samples need 1024 clock cycles to be processed.

A feedback implementation of the FFT with radix R requires $R-1$ blocks of dual port memory to store samples both coming from the input or fed back from the butterfly. Following this structure, every memory is designed to store N/R samples with N being the samples coming from the previous processing phase. During the first processing mode (Mode 1 in figure 6) data coming from the previous stage are loaded through one port while the other port is used to dump the currently processed data. When sample $N(R-1)/R$ arrives, the second phase starts, dumping a piece of data from each block to be processed in the butterfly while feedback data are loaded from the butterfly to be processed later in the rotator (Mode 2). This particular memory management does not only allow temporary storage of data, but also the re-ordering of samples to be processed in the current stage.

4.4 Feedforward Architecture

In the FF architecture the samples can go ahead once a stage is processed because there are several rotators (see figure 7). Following the memory a butterfly implements every low order DFT, and next an array of rotators (one per sub-DFT produced at the output) is required. Their outputs will feed the next stage in the chain. Actually, the performance provided by this architecture is $R\ samples/clk$, being R the radix. As can be seen in figure 8, the concurrent execution performed by this architecture allows the processing of an N-point FFT in $N/4$ cycles because a radix 4 is used in this example (in the figure, a 1024

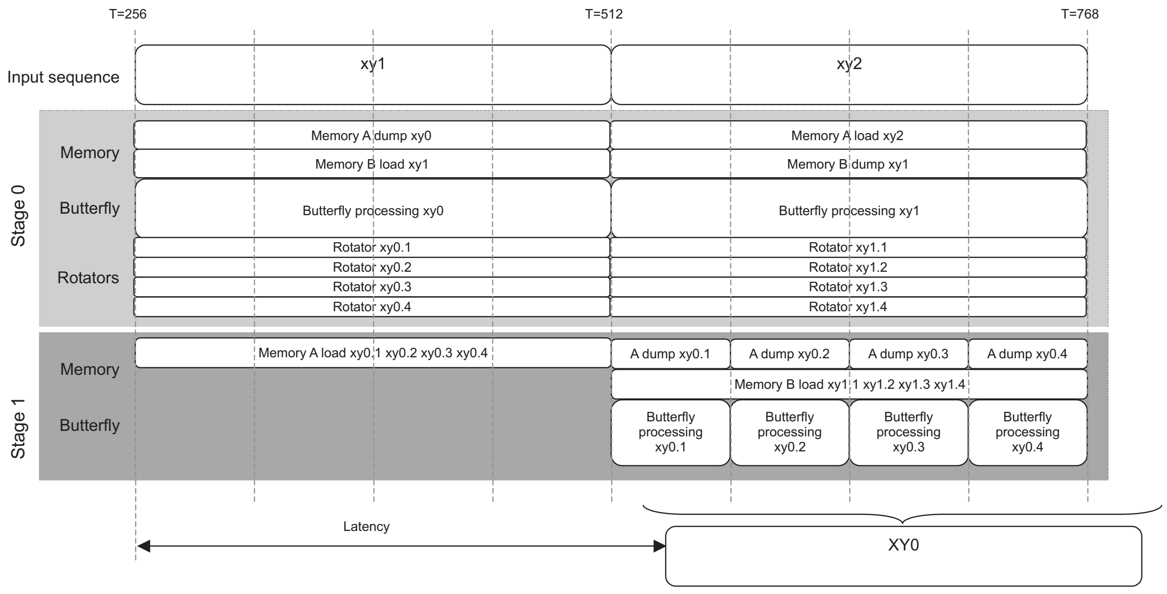


Figure 8: Time evolution in a 1024 points, radix 4, FF architecture.

points FFT is processed in 256 cycles). In this case, since data coming from the different sub-DFTs are passed in parallel, a different memory scheme is necessary, because parallel storage is required together with a *re-ordering* of data.

This architecture requires R ordered input samples. Moreover, the different sub-DFTs generated at the output will be processed by the rotators in parallel, and will be sent as a block to the following stage.

That kind of concurrency in the implementation can only be obtained through a memory structure able of simultaneously storing all input samples, re-ordering data and providing data to process the sub-DFT. We have used a matrix memory of $R \times R$ with *ping-pong* structure. As can be seen in figure 8, a memory called A is used to store input samples while a second memory, B, extracts data to feed the butterfly. During the next time slot, memories A and B exchange their roles allowing continous processing.

Additionally, in this architecture every Cordic element ($R - 1$ in a radix R implementation) requires a memory with the sequence of rotations to perform per twiddle.

4.5 Overview of the proposed architectures

To summarize the description of the designed FFT architectures, table 1 shows their requirements in terms of basic elements (memory, butterfly and rotators). Memory size is measured by the number of samples that it holds. As can be seen, the resource requirements are higher for the FF architecture

Table 1: Hardware resources required by FB and FF architectures.

Architecture	**Radix**	**Rotators**	**Adders**	**Memory**	
				Data	Rotations
Feedback (FB)	2	$S-2$	$2 \times S$	N	N
	4	$S-1$	$8 \times S$	N	N
	8	$S-1$	$24 \times S$	N	N
feedforward (FF)	2	$S-2$	$2 \times S$	$6 \times N$	N
	4	$3 \times (S-1)$	$8 \times S$	$4.6 \times N$	N
	8	$7 \times (S-1)$	$24 \times S$	$4.3 \times N$	N

than for the FB one. This is due to the higher degree of parallelism presented by this architecture, which additionally provides better performance. We should remember that the FB architecture is able to process a sample per clock cycle, while the FF architecture processes R samples per clock cycle, being R the Radix.

5 Monobit FFT Implementation

To maximize the throughput of the implemented FFT-based channelized receivers, measured as processed Msamples/s, the simplification of the computational complexity of the FFT is required. This can be accomplished by avoiding complex multiplications: Using a monobit kernel for the FFT [22, 23]. This algorithm can be further simplified by reducing the number of bits to represent the input samples. The increase in the throughput is obtained at expense of degradation in the dynamic range of the channelized receiver (see [23] for more details).

In the monobit FFT the twiddle terms are rounded using the function:

$$G(e^{j\phi}) = \begin{cases} 1 \quad if \quad \frac{-\pi}{4} \le \phi < \frac{\pi}{4} \\ j \quad if \quad \frac{\pi}{4} \le \phi < \frac{3\pi}{4} \\ -1 \quad if \quad \frac{3\pi}{4} \le \phi < \frac{5\pi}{4} \\ -j \quad if \quad \frac{-3\pi}{4} \le \phi < \frac{-\pi}{4} \end{cases} \tag{4}$$

This approximation of the rotation angles allows the hardware implementation of the monobit FFT to follow the structure of the previously described architectures, with the only difference that rotators always use the following angles: 0, $\pi/2$, π, or $-\pi/2$. Actually, those values make unnecessary any multiplier or rotator, what results in a clear reduction of the final area and a significant improvement in speed, as will be shown in section 7.

To perform the simplified rotations, the rotator present in every stage is implemented using a reduced set of multiplexers and adders. This particular rotator consumes less area, can work at a higher clock frequency and requires no memory.

The final performance of the monobit implementation is not only related to the previous modifications. For the classical FFT, the capacity of the parallel processing is limited by the radix used in the architecture, due to the high cost in area required to replicate the critical components. Also the area required to implement the butterfly grows exponentially with the radix (see figure 4), consequently, radix-8 butterflies are the largest that can be implemented in practice for conventional FFT architectures. Nevertheless, the monobit simplification allows the implementation of butterflies with a higher radix without an exponential growth of the area, by means of the transformation of the non-trivial twiddles into trivial twiddle terms. Again, architectures with greater radix result in a higher degree of parallelization, what translates into increasing processing speed.

6 The FFT Generation tool

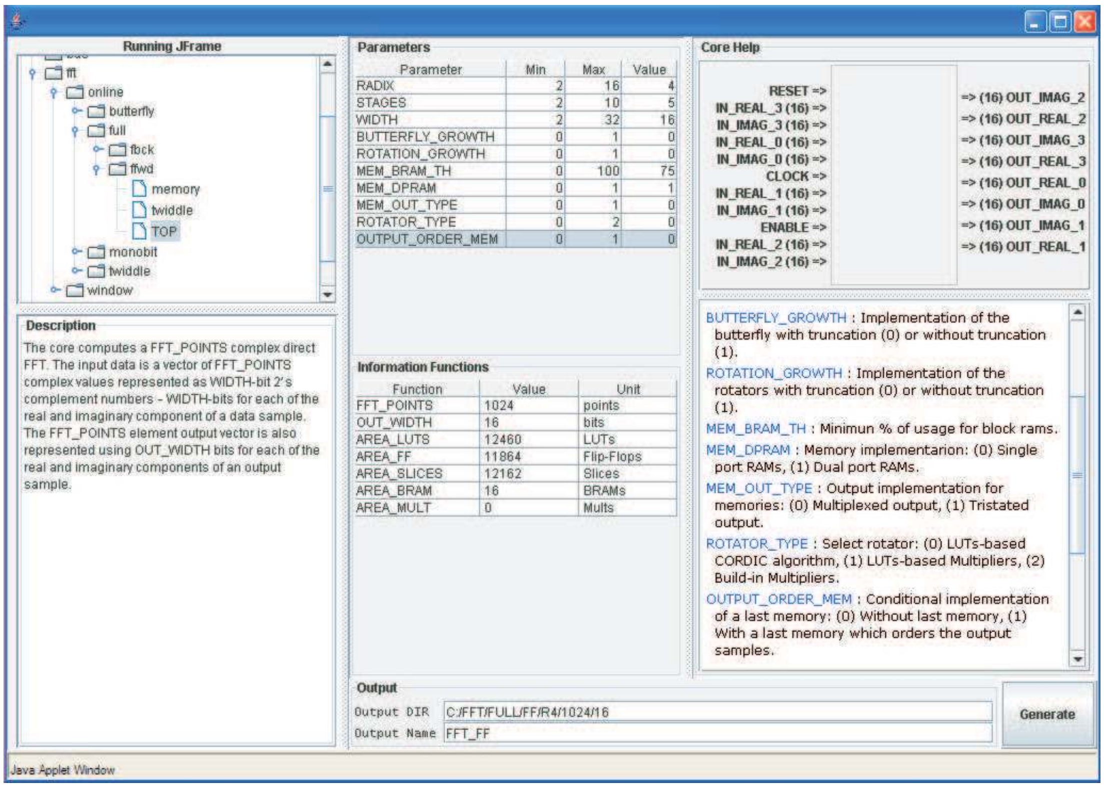
To help the system designer to explore the possibilities offered by the different architectures described above we have developed an FFT generation tool. This tool has been implemented in Java using as internal description language xHDL [24], a VHDL-based metalanguage that simplifies the specification and parameterization of designs. The tool generates synthesizable VHDL taking as input the configuration options selected by the user.

The tool provides support to select and configure from among the many parameters that characterize every single FFT architecture. Once a given architecture has been selected, the tool provides quick estimates on basic parameters and functions that help the user in the design space exploration phase. As can be seen in figure 9, the user may select the following input parameters:

- Parameters related to the transform. The transform length (number of points, N) which is related to the parameters `RADIX` (R) and number of `STAGES`, S (remember that $S = log_R N$). A special parameter related to the transform output is `OUTPUT_ORDER_MEM`, which selects the use of a last stage memory to reorder the output samples.

- Parameters which define the `WIDTH` of the input samples and the bit growth of data through the different components: `BUTTERFLY_GROWTH` and `ROTATION_GROWTH`.

- Parameters which drive the implementation to the physical device. If the FPGA where the design


Running JFrame

Description

The core computes a FFT_POINTS complex direct FFT. The input data is a vector of FFT_POINTS complex values represented as WIDTH-bit 2's complement numbers - WIDTH-bits for each of the real and imaginary component of a data sample. The FFT_POINTS element output vector is also represented using OUT_WIDTH bits for each of the real and imaginary components of an output sample.

Parameters

Parameter	Min	Max	Value
RADIX	2	16	4
STAGES	2	10	5
WIDTH	2	32	16
BUTTERFLY_GROWTH	0	1	0
ROTATION_GROWTH	0	1	0
MEM_BRAM_TH	0	100	75
MEM_DPRAM	0	1	1
MEM_OUT_TYPE	0	1	0
ROTATOR_TYPE	0	2	0
OUTPUT_ORDER_MEM	0	1	0

Information Functions

Function	Value	Unit
FFT_POINTS	1024	points
OUT_WIDTH	16	bits
AREA_LUTS	12460	LUTs
AREA_FF	11864	Flip-Flops
AREA_SLICES	12162	Slices
AREA_BRAM	16	BRAMs
AREA_MULT	0	Mults

Core Help

BUTTERFLY_GROWTH : Implementation of the butterfly with truncation (0) or without truncation (1).

ROTATION_GROWTH : Implementation of the rotators with truncation (0) or without truncation (1).

MEM_BRAM_TH : Minimun % of usage for block rams.

MEM_DPRAM : Memory implementarion: (0) Single port RAMs, (1) Dual port RAMs.

MEM_OUT_TYPE : Output implementation for memories: (0) Multiplexed output, (1) Tristated output.

ROTATOR_TYPE : Select rotator: (0) LUTs-based CORDIC algorithm, (1) LUTs-based Multipliers, (2) Build-in Multipliers.

OUTPUT_ORDER_MEM : Conditional implementation of a last memory: (0) Without last memory, (1) With a last memory which orders the output samples.

Output

Output DIR C:/FFT/FULL/FF/R4/1024/16

Output Name FFT_FF

Figure 9: User interface of the FFT generation tool when selecting a Radix 4 FF architecture with 16 input bits, 1024 points and no growth.

will be mapped includes specific blocks (multipliers, RAM, etc.) these parameters allow the use and configuration of this kind of built-in elements. For example, MEM_BRAM_TH is the percentage of use of a RAM block that should be filled by a given implementation to allow its use. The RAM can also be configured to be single or dual port (MEM_DPRAM).

The tool also provides the limits within which the value of those parameters can range. As quick help for the design exploration process, the tool computes or estimates the following functions:

- General functions which are directly computed from parameters as is the case of Number of stages of the FFT (FFT_POINTS=Radix^{Stages}) or the bitwidth of the output results (OUT_WIDTH).
- Area estimators: Number of LUTs[1] required (AREA_LUTS), Number of flip-flops (AREA_FFS) and Area of the FFT in slices (AREA_SLICES)[2].

[1]Look up Table, basic implementation element in the FPGAs for combinational logic.

[2]A slice is a basic implementation element in the FPGAs that holds two LUTs

- Estimators of usage of special resources: Number of RAM blocks (`AREA_BRAM`) and Number of multipliers (`AREA_MULT`).

With this tool, the system designer can explore the design space in a quick and easy way. Moreover, the tool provides the final code that will be synthesized into the FPGA, what makes the designer to get ride of low level implementation details, out of the scope of a signal processing engineer.

There are other appoaches that develop a tool to generate FFTs cores or similar. The Xilinx LogiCore is a well-known example [25], but the degree of parameterization of this tool is significantly reduced when compared to our approach. Other interesting approach, SPIRAL [26], is based on a different DFT architecture, the Pease FFT, which provides a parallelized implementation that can be considered in between the architectures presented here. Next section will outline the main differences among our approach and these implementations.

7 Experimental Results

All experimental results have been obtained targeting Xilinx FPGAs, (in particular the VIRTEX-II xc2v4000-6), using as development environment Xilinx ISE 7.1.

The various proposed FFT architectures and the large parameter set that can be used for their configuration provide a very wide experimental outcome. The key parameters under analysis in all these architectures are:

- Bitwidth of the input samples.
- Number of stages.
- Radix of the implementation (power of 2).
- Stage scaling (truncation in the butterflies).

The performance of the different solutions will be analyzed in terms of:

- Area (slices and BRAM used).
- Latency, time to process an FFT (from start to end).
- Clock speed (MHz) and throughput (Msamples/s).

- Power.
- Detection performance.

In order to organize the analysis of the experimental results we will perform it in three different scenarios. First and second scenarios will be devoted to the analysis of FB and FF architectures for both conventional and monobit FFTs, respectively. The third scenario will study the power dissipation of all architectures. Finally a last scenario will be devoted to analyzing the results provided by the FPGA implementation of the FFT when used in a digital channelized receiver, and due to the importance of these results, section 8 will be devoted to this analysis. But previously to the description of the above mentionned scenarios we consider that it is important to describe the influence that the target architecture, FPGAs, presents in all the results. This will be done in next subsection.

7.1 FPGA-based Implementation

An important issue to consider when designing FPGA-based systems is the physical device on which the design will be mapped. Most FPGAs are composed of several configurable blocks called slices. Every slice includes multiplexors, flip-flops and two 4-input Lookup tables (LUTs) to implement logic functions, which are the most common low level configurable components in the FPGAs. Additionally, some FPGAs include special resources as they can be memory blocks (BRAMs) or built-in multipliers. There are three basic memory structures: Distributed flip-flops, distributed memory (based on the aggregation of LUTs) and memory blocks.

Taking the final platform into account is very advantageous from the design point of view, because every particular device biases the implementation with a set of constraints as they are the available blocks, the speed, etc. The design must exploit the configuration options offered by these programmable devices. Even though we have targeted Xilinx FPGAs, the specification of the different FFT descriptions has been done as open as possible, exploiting common components to all the families (from Spartan-II to the latest Virtex-4 devices). Moreover, our designs are based on parameters that allow to use and exploit particular components when avaliable (built-in multipliers[3] and BRAM memories) and configure available Input/Output resources.

A critical point in the FFT design is the mapping of memory resources because this will clearly impact the final performance. Given the FFT algorithm structure (memory requirements scale down as

[3]When avaliable, built-in multipliers can been used. It is important to remark that their number is limited and they implement fixed-size multiplications (18x18).

Table 2: Experimental results for the FB and FF architectures for the conventional FFT.

STAGES	POINTS	FFT 8 BITS - FB RADIX 4									
		WITH TRUNCATION					WITHOUT TRUNCATION				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	638	0	0,128	274	274	806	0	0,139	267	267
3	64	1128	0	0,365	274	274	1604	0	0,405	262	262
4	256	1903	0	1,128	274	274	2914	0	1,254	256	256
5	1024	2589	3	3,996	274	274	4328	3	4,442	251	251

STAGES	POINTS	FFT 16 BITS - FB RADIX 4				
		WITH TRUNCATION				
		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec
2	16	1342	0	0,171	251	251
3	64	2452	0	0,462	251	251
4	256	4086	0	1,327	251	251
5	1024	7956	0	4,486	251	251

STAGES	POINTS	FFT 8 BITS - FF RADIX 2									
		WITH TRUNCATION					WITHOUT TRUNCATION				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
4	16	1133	0	0,225	254	507	1354	0	0,256	246	492
6	64	1816	0	0,640	236	472	2475	0	0,704	236	471
8	256	4533	0	2,031	227	454	6401	0	2,154	227	454
9	512	8683	0	4,652	184	368	12478	0	4,482	199	398
9	512	4471	8	3,754	228	456	7418	8	3,894	227	454
10	1024	15440	0	8,790	186	372	20873	0	9,032	186	372
10	1024	3862	12	7,182	228	455	6627	12	7,380	228	455

STAGES	POINTS	FFT 8 BITS - FF RADIX 4									
		WITH TRUNCATION					WITHOUT TRUNCATION				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	1235	0	0,103	244	975	1532	0	0,112	242	967
3	64	2830	0	0,271	244	975	3815	0	0,300	240	960
4	256	4586	0	0,810	236	943	6703	0	0,861	236	943
5	1024	13031	0	2,929	223	890	17751	0	3,030	222	887

STAGES	POINTS	FFT 16 BITS - FF RADIX 4				
		WITH TRUNCATION				
		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec
2	16	3578	0	0,149	235	940
3	64	7591	0	0,362	235	940
4	256	12034	0	0,932	235	940
5	1024	27702	0	3,235	213	852

STAGES	POINTS	FFT 8 BITS - FF RADIX 8									
		WITH TRUNCATION					WITHOUT TRUNCATION				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	64	4716	0	0,170	230	1840	5920	0	0,275	153	1224
3	512	10988	0	0,761	230	1840	15009	0	1,586	116	928

the stage number increases in DIF FFTs), we have assigned memory blocks to the very first stages of the algorithm. Once memory requirements are lower, we can use distributed memory, which is more flexible than memory blocks but can hold less samples. Finally, flip-flops will map the pipeline registers.

As multipliers, memory blocks constitute a limited resource, but their use can result in important area savings while avoiding the big size of distributed memory. The design exploration tool depicted in figure 9 allows the designer to choose the most convenient memory option to a particular implementation. For instance, the designer may choose the % of minimum usage that is required to change from distributed memories to block memories (`RAM_THRESHOLD`).

7.2 Scenario 1: Comparing the FB and FF Architectures

In the first scenario we will present a comparative study of the implementation of the FB and FF architectures for the conventional FFT.

Table 2 summarizes the results we have obtained for different implementations when exhaustively exploring the design space with our tool. In this way, we have generated 16, 64, 256 and 1024 point FFTs with both FB and FF architectures. The bitwidth was initially fixed to 8 and 16 bits and we considered both truncation and no truncation through the stages. Additionally, given that the FF architecture allows the parameterization of the radix, we have implemented radix 2, 4 and 8, while the FB architecture has

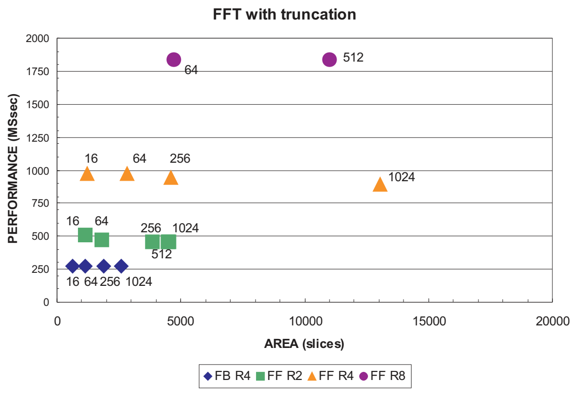


Figure 10: Area-performance plot for FB and FF architectures (8 bits, with truncation).

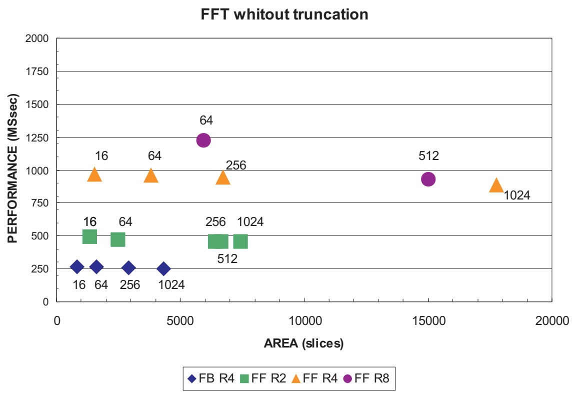


Figure 11: Area-performance plot for FB and FF architectures (8 bits, without truncation).

been implemented for radix 4. Moreover, large implementations have been generated with and without usage of BRAM.

In figures 10 and 11 we can see the area-performance trade-off that can be obtained for 8-bit designs in the FF and FB architectures. From a first analysis of table 2 and these graphs we can draw the following conclusions:

1. It can be clearly seen that the FB architecture requires lower area than the FF implementation due to resource sharing, especially when computing the FFT with a high number of points. On the other hand, the FF architecture processes several samples in parallel, which leads to a higher throughput.

2. As expected, all implementations with no truncation of bits in intermediate stages always present bigger area and lower speed than the ones with truncation.

3. The influence of the radix can be analyzed studying the FF architecture. As can be seen in Table 2, except for the 1024 points implementation, the area of architectures with radix 2 and 4 is more or less the same. However, performance results are completely different. Even though radix 2 implementation presents a slightly higher working frequency, radix 4 implementations have a higher throughput (almost double), due to the higher parallelization of the operations.

4. The latency values of Table 2 refer to the total computation of an FFT. Other computations can run simultaneously in the FPGA, what cannot be done in a DSP [27].

5. The use of BRAM blocks has only been worthwhile in those examples that require large memory

Table 3: Results for different 16 Bit 1024 point FFTs (radix 4).

	SLICES	BRAM		MULT18x18		T_SLICES	MHz	Mssec	kS/SLICE
		Num	EqSlices	Num	EqSlice				
XILINX	2744	7	5950	24	7680	16374	214	214	13,070
FF	21707	0	0	0	0	27702	213	852	30,756
FB	7956	0	0	0	0	7956	251	251	31,549
SPIRAL P4_TH2	1509	64	54400	16	5120	61029	167	668	10,946
SPIRAL P4_TH128	3287	16	13600	16	5120	22007	167	668	30,354

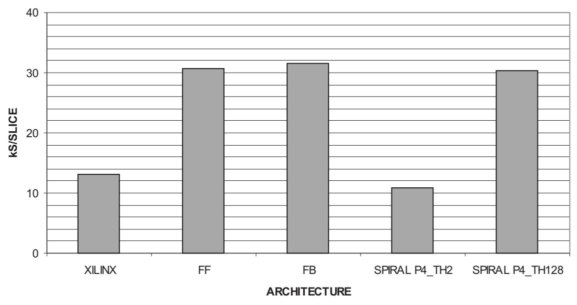


Figure 12: Comparison of FB and FF architectures with Spiral and Xilinx in terms of Ks/slice.

sizes: 512 and 1024 point FFTs. Actually, the FB architecture obtains clear benefits from the use of BRAMS, but the FF architecture can only take advantage of them for radix 2, when the required memory size is bigger than the threshold established to use of BRAM instead of distributed memory[4].

7.2.1 Comparative study with other implementations

As mentioned previously, there are two tools that can be used to generate FFT cores: Xilinx Core Generator [25], and Spiral [26]. We have generated with these tools different implementations of a 1024 points FFT with 16 input bits and no growth through the stages. In table 3, we can see the experimental results obtained from Xilinx Core Generator (Xilinx), our FB and FF architectures and two experimental results from Spiral (Spiral P4_TH2 and P4_TH128, with different degree of paralellism).

As can be seen in table 3 these architectures have very few common points and take advantage of the FPGA resources in different ways. All implementations make use of *slices* to implement logic functions, which could be a general measure of area. However, specific components can also be used, as is the case of BRAM memories or built-in multipliers, what results in important slice savings. It is therefore necessary to have a uniform measure to compare the different area and performance results. We have decided to

[4]We have considered in our results that BRAMs should be filled more than 50%, otherwise they could be used by other computations in the FPGAs. Nevertheless, the user of the design exploration tool can modify this threshold.

measure all area related issues in *slices*, which is the only component present in all FPGA families. To carry out this measurement, we need to know the equivalence of built-in components into *slices*. In this sense, to establish the area required by a built-in multiplier we have implemented with logic a 16x16 pipelined multiplier, which occupies around 350 *slices*. We have not implemented the 18x18 multiplier integrated in the Xilinx devices because it is not always fully exploited.

Regarding BRAMs, since they are large memory blocks that can be configured with different utilization, we have established a capacity value of 50% to find out the equivalent measure in *slices*. With this capacity value we have obtained an area use of 850 slices.

Finally, the best way to qualify a given architecture is to consider not only area, but also performance. In this sense we have defined a new measure called $KSsec/slice$ which provides a ratio between performance (Ksamples per second) and area (*slices*). This new measure has been plotted in figure 12 to better compare all architectures. Xilinx implementations do an intensive use of BRAMs and multipliers, with the subsequent low count of slices. However, if we map these components into equivalent slices, the area measurement grows significantly (see column `T_SLICES` in table 3), but with a performance comparable with our FB implementation. Consequently, the metric shows that this implementation is characterized by a poor performance-area ratio. Regarding the Spiral architectures, `SPIRAL P4_TH2` shows an intensive use of BRAM components with the corresponding reduced number of slices. The second architecture, `SPIRAL P4_TH128`, performs a more efficient BRAM mapping. The performance-area ratio shows for the first architecture a similar behaviour to the Xilinx implementation, whereas the second implementation improves this ratio significantly. We can conclude that FF and FB architectures provide the best implementation option, together with one of the Spiral architectures. In this sense, the FB architecture presents the best results in terms of area and the FF architecture shows the best performance figures, while `SPIRAL P4_TH128` provides a solution that can be placed in between.

7.3 Scenario 2: Analysis of the monobit FFT Implementation

In this scenario we will study the results obtained for the FB and FF architectures when implementing the monobit FFT. The monobit architectures have been implemented with different bitwidths of the input samples to observe the benefits in terms of area and performance obtained with the monobit simplification. Both truncation and no truncation options have been considered again. Table 4 summarizes these results. Additionally, given that the FF architecture allows the parameterization of the radix, we have explored implementations with other radices, as can be seen in table 5.

Table 4: Experimental Results for the monobit FFT Implementations.

		WITH TRUNCATION									
		FFT MONOBIT 16 BITS - FB RADIX 4					FFT MONOBIT 16 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
STAGES	POINTS	SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	933	0	0,080	299	299	2326	0	0,083	242	968
3	64	1585	0	0,261	299	299	4929	0	0,227	242	968
4	256	2636	0	0,926	299	299	7814	0	0,719	242	968
5	1024	3100	3	3,515	299	299	21678	0	2,885	218	872
		FFT MONOBIT 8 BITS - FB RADIX 4					FFT MONOBIT 8 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
STAGES	POINTS	SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	498	0	0,072	332	332	1167	0	0,083	242	968
3	64	862	0	0,235	332	332	2468	0	0,227	242	968
4	256	1430	0	0,868	319	319	3914	0	0,719	242	968
5	1024	1730	3	3,295	319	319	10708	0	2,885	218	872

		WITHOUT TRUNCATION									
		FFT MONOBIT 8 BITS - FB RADIX 4					FFT MONOBIT 8 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
STAGES	POINTS	SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	597	0	0,074	323	323	1352	0	0,082	245	980
3	64	1122	0	0,248	314	314	3070	0	0,224	245	980
4	256	1980	0	0,902	306	306	5192	0	0,710	245	980
5	1024	2599	3	3,527	298	298	13845	0	2,859	220	880
		FFT MONOBIT 4 BITS - FB RADIX 4					FFT MONOBIT 4 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
STAGES	POINTS	SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	377	0	0,070	341	341	774	0	0,082	245	980
3	64	754	0	0,235	332	332	1817	0	0,224	245	980
4	256	1358	0	0,865	319	319	3257	0	0,710	245	980
5	1024	1899	3	3,347	314	314	8436	0	2,859	220	880
		FFT MONOBIT 2 BITS - FB RADIX 4					FFT MONOBIT 2 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
STAGES	POINTS	SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	264	0	0,068	351	351	457	0	0,082	245	980
3	64	563	0	0,229	341	341	1214	0	0,224	245	980
4	256	1048	0	0,865	319	319	2252	0	0,710	245	980
5	1024	1535	3	3,295	319	319	5712	0	2,859	220	880

Table 5: Analysis of the influence of the radix.

		FFT MONOBIT 2 BITS - FF				
		AREA		LATENCY	SPEED	
RADIX	POINTS	SLICES	BRAM	usec	MHz	MSsec
16	256	7783	0	0,225	240	3840
8	512	5552	0	0,683	240	1920
32	1024	27409	0	0,500	180	5760
8	4096	21790	0	4,974	228	1824
16	4096	26903	0	2,342	240	3840

If we compare tables 2 and 4 we can observe how the monobit implementations present a significant increase in speed and a clear reduction in area and latency. Figure 13 shows the results of the design exploration performed with our tool for conventional and monobit FFTs in different points of the design space.

All monobit architectures present a performance increase obtained by means of a higher clock frequency. This is due to the substitution of the complex rotator in the conventional FFT by a monobit one, which is faster. FB architectures exhibit a better benefit from the monobit simplification because the rotators were in the critical path. On the other hand, the FF architecture hides these benefits because the limiting speed element is not the rotator but the memory. In terms of area, as expected, larger monobit FFTs save more area.

Regarding the radix variations, the benefits of the monobit implementation can be clearly seen in the case of 2 input bits (see table 5). The FF monobit architecture allows implementations with higher

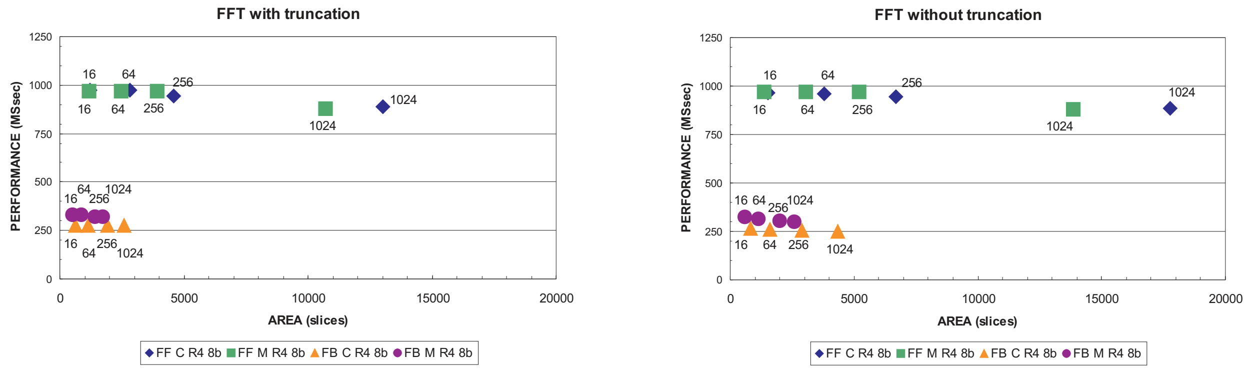


Figure 13: Conventional vs. monobit FFTs (with and without truncation).

radix (16, 32), with the corresponding performance improvement. These results cannot be obtained by conventional FF FFTs. In this case the clock frequency is more or less the same, because the critical path is related to the complex memory structure, but the higher the radix, the higher the performance can be achieved (up to GS/s).

7.4 Scenario 3: Power Consumption

A key parameter in most data processing applications is the power consumption of the resulting implementation. It is due to two main reasons. First, the power density in current FPGAs may produce an uneven distribution of temperature on the surface of the device with the corresponding hot-spots. This may produce a malfunctioning of the particular device or even of the whole system. Second, many current systems may be battery powered, what makes the power dissipation a new design dimension to be considered during the design cycle.

We have evaluated the power dissipation that our FFT architectures present. Actually, with the FFT generation tool a quick power estimator has been included. Figure 14 depicts the main results we have obtained when analyzing power dissipation. As can be seen, the power dissipation of FB and FF architectures has been evaluated for different clock frequencies. It is well known that dynamic power is directly proportional to the frequency, as can be observed in that plot. Of higher interest is the comparison that can be carried out between FB and FF architectures and between conventional and monobit FFTs. As expected, the monobit implementation exhibits the lowest power dissipation, mainly due to its lower complexity. Moreover, when comparing FB and FF architectures, the the FF architecture presents a higher power consumption due to its higher complexity.

As expected, the power consumption is directly related to the area of the implementation (including

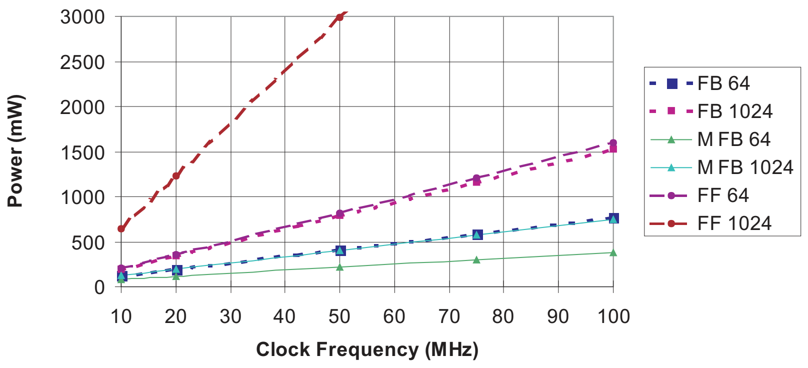


Figure 14: Dynamic Power Dissipation for different FFT architectures (8 bits, Radix 4, with truncation).

both logic and interconnection area). We should remark that most dynamic power in FPGAs is consumed in the interconnection resources (70%, as studied by [28]). For FFTs of the same length, the FF architecture always presents greater power consumption than the FB. Moreover, the monobit simplification obtained in terms of area can be also observed in terms of power. Even though the power results plotted in figure 14 are very high, recent advances in FPGA technology [29] include process and architecture innovations to reduce both static and dynamic power. For instance, the dynamic power consumption measured in the new Virtex-5 FPGAs presents a 55 % reduction when compared to the previous implementation family (Virtex4). Therefore, we expect that the power consumption of the FB and FF architectures will be reduced orders of magnitude with respect to the values plotted in figure 14, which correspond to VirtexII FPGAs, the family previous to Virtex4.

Regarding the influence of the input signals on the power consumption, we have evaluated our implementations with both gaussian noise and sinusoids with different amplitudes, and we have observed that the power consumption is similar. The reason for this performance is that the activity rates of the input signals are in all cases very similar, due to the two's complement representation of the input signal.

8 Analysis of a Channelized Receiver Performance based on the proposed FFT architectures

In this section we will study the proposed architectures from the signal detection point of view, assuming that the input is a sinusoidal signal with its associated real, additive, white, Gaussian noise with standard deviation, σ. False alarm probability, detection probability and dynamic range of several FFT implementations have been analyzed in order to determine under which circumstances the results of the

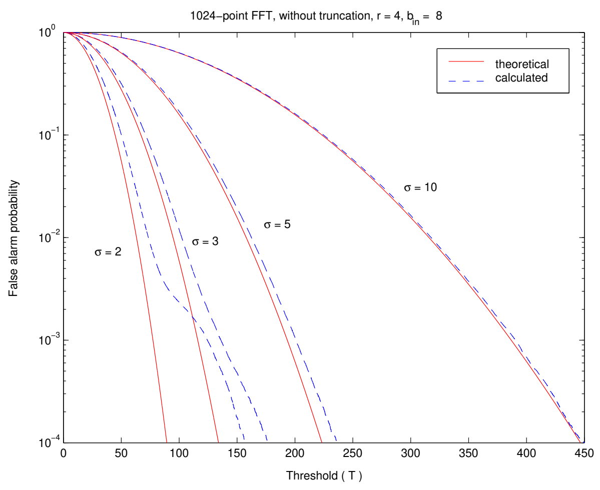


Figure 15: P_{fa} of a 1024-point FFT without truncation.

algorithm are similar to the ones obtained by ideal FFTs, depending on the finite arithmetic effects and the hardware parameters of the N-FFT: Input data bitwidth, truncation along the FFT stages, radix, and number of points, N. It is important to point out that FB and FF implementations have the same performance regarding detection capabilities.

All false alarm probability calculations have been obtained using Monte Carlo simulations with 10^6 independent trials. Regarding detection probabilities, 10^3 independent trials have been used. Channels 0 and $N/2$ are not considered in the calculations because noise statistics are different in these channels.

8.1 False Alarm Probability

8.1.1 Impact of rotator error

The first design under study is a 1024-point FFT, radix 4, 8 input bits, without truncation in the butterflies. As the input is represented with 8 bits, the signal amplitude ranges form -128 to 127. In this context, the false alarm probability per channel, P_{fa} has been calculated. The theoretical false alarm probability per channel for a linear detector is:

$$P_{fa} = exp(\frac{-T^2}{\sigma^2 N k^2}) \tag{5}$$

where T is the threshold of the detector, N stands for the number of points of the FFT, and k represents the intrinsic global gain due to the Cordic rotators [18].

Figure 15 shows the false alarm probability for different values of noise standard deviation, σ. As can

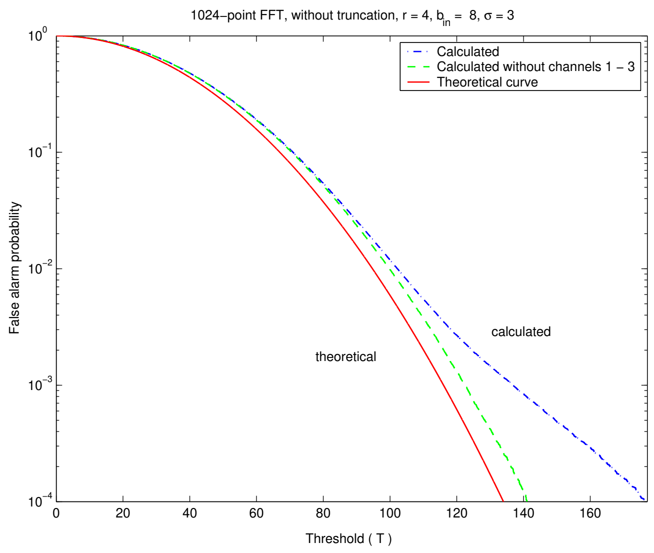


Figure 16: P_{fa} of a 1024-point radix-4 FFT without truncation and $\sigma = 3$.

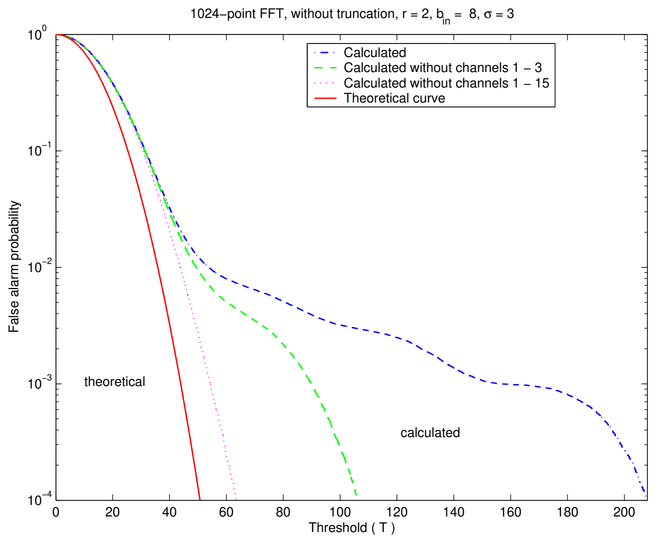


Figure 17: P_{fa} of a 1024-point, radix 2 FFT, without truncation and $\sigma = 3$.

be seen, when $\sigma \geq 5$ the experimental and the theoretical results are quite similar. As σ decreases, the discrepancies between theory and implementations increase. As there is no truncation in the butterflies and the quantization error of the input signal is not significant for $\sigma > 1$, this problem only depends on the error in the rotators. This effect is more significant at the lowest channels of the FFT output. This is shown in figure 16, where the curve between the calculated P_{fa} and the theoretical one represents the P_{fa} per channel when channels 1 to 3 are not considered.

Radix. The rotator error effect is more pronounced for radix 2 than for radix 4. This is related to the fact that the number of rotators in a radix 2 architecture is almost double and consequently, there exist more sources of error.

It can be noted that the effect presented before appears at a higher value of σ: P_{fa} calculated for $\sigma =$ 10 does not follow the theoretical expression, unless channels 1 to 3 are eliminated in the calculations. This was not the performance for the radix 4 implementation and $\sigma =$ 10, figure 15. Results for radix 2 and $\sigma = 3$ are depicted in figure 17. It can be noted the effects of the different statistics at the the first channels are more pronounced: Channels 1 to 16 must be eliminated instead of channels 1 to 3 in figure 16.

Number of points. When the number of points of the FFT increases, two circumstances must be considered. Firstly, the FFT has more stages, so more rotators are used. Secondly, the difference

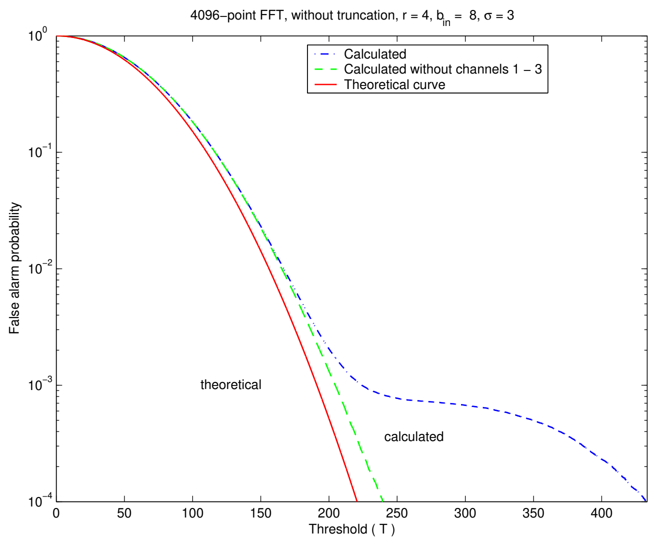


Figure 18: P_{fa} of a 4096-point FFT, radix 4, without truncation and $\sigma = 3$.

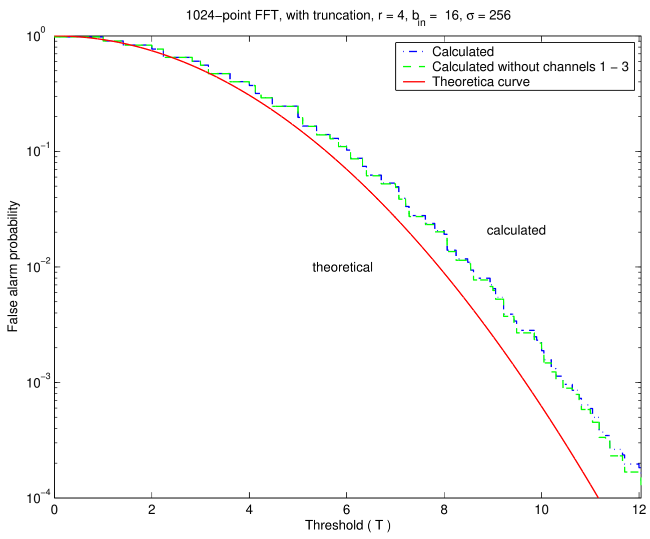


Figure 19: P_{fa} of a 1024-point FFT with truncation, 16 input bits, $\sigma = 256$.

between two rotation angles is smaller, so that more accurate rotations must be performed.

Figure 18 shows the P_{fa} of a 4096-point FFT, without truncation, radix $= 4$, and 8 input bits. For $\sigma = 10$ the calculated P_{fa} is similar to the theoretical one, as it happens with the 1024-point FFT. However, for $\sigma = 3$ the rotator error affects the results of the FFT, and the effect on the first four channels is more dramatic than in the 1024-point FFT (figure 16).

8.1.2 Impact of the butterfly truncation

If an input bitwidth of 8 bits and a 1024-point FFT with butterfly truncation are considered, it must be realized that altogether 10 bits are removed through the FFT (2 bits per stage for a radix-4 implementation), which leads to a lower performance. For example, a $Pfa = 10^{-3}$ cannot be achieved for $\sigma = 10$. Thus, in order to study the influence of the butterfly truncations, a 1024-point FFT with truncation and 16 input bits has been chosen, which is a widely used architecture.

Firstly, if σ is high, the P_{fa} is similar to the theoretical one, so neither the butterfly nor the multiplier truncations are significant. Figure 19 shows the difference between the theoretical and the calculated P_{fa} curves when $\sigma = 256$.[5] For lower values of σ, the butterfly truncation modifies the probability density functions of the noise samples. As a consequence, there appear big discrepancies between the calculated P_{fa} and the theoretical one deduced for continous gaussian noise samples.

[5]It can be observed that the calculated curve has steps due to the fact that both the real and the imaginary parts of the output are integers, so that the module range is not continuous.

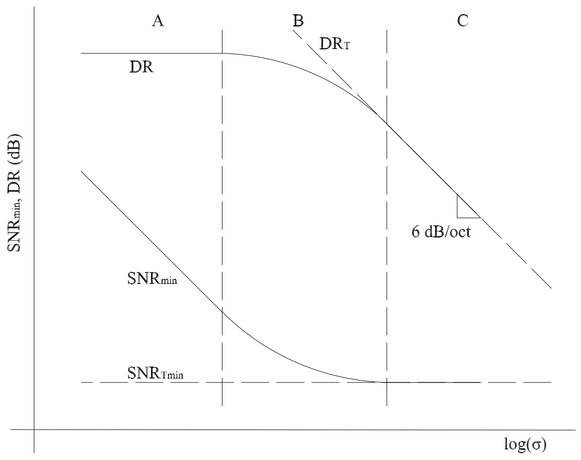


Figure 20: Typical curves of SNR_{min} and DR.

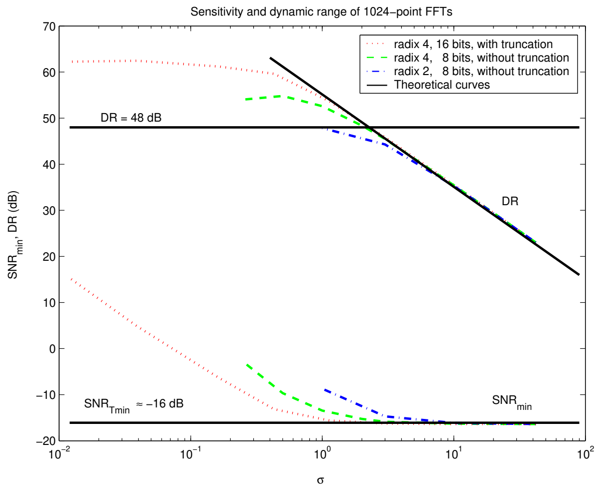


Figure 21: Sensitivity and dynamic range of 1024-point FFTs.

8.2 Detection probability and dynamic range

The sensitivity, $S = A_{min}^2/2$, is defined as the power of a sinusoid of amplitude A_{min} at the input that assures certain detection probability, P_d, for a fixed false alarm probability, P_{fa}. According to this, the minimum signal to noise ratio, SNR_{min}, is the quotient between the sensitivity and the input noise power: $SNR_{min} = \frac{A_{min}^2/2}{\sigma^2}$.

The theoretical SNR_{Tmin} may be calculated from the required SNR at the output, SNR_O, for given P_d and P_{fa} [30] and the FFT processing gain for a centered sinusoidal signal, $G = \frac{N}{2}$. Thus, the theoretical SNR_{Tmin} can be obtained as:

$$SNR_{Tmin} = \frac{SNR_{Omin}}{G} = \frac{SNR_{Omin}}{N/2} \tag{6}$$

On the other hand, the maximum signal to noise ratio, SNR_{Tmax}, is the quotient between the most powerful input sinusoid without truncation at the output of an ADC with b bits and the input noise power:

$$SNR_{Tmax} = \frac{A_{max}^2/2}{\sigma^2} = \frac{(2^{b-1}-1)^2/2}{\sigma^2} \tag{7}$$

Finally, the dynamic range, DR, is the quotient between the SNR_{max} and the SNR_{min}.

Figure 20 shows a schematic diagram with the behaviour of our FFT implementations (SNR_{min} and DR) versus the input noise standard deviation, σ. In this figure, three main regions can be distinguished.

When a high value of σ is selected (region C), the SNR_{min} remains constant (A_{min} decreases when σ decreases) and is independent of σ as in the theoretical case. This happens because, as studied before, the truncation effects are not relevant for high values of σ. Likewise, the DR follows the theoretical behavior: It increases 6 dB per octave. Therefore, the best performance is obtained for decreasing values of σ.

On the contrary, in region A the truncation effects are present because σ is low and both signal and noise may occupy a few quantification levels. Under these circumstances, the dynamic range is approximately constant because a significant reduction of σ hardly modifies the sensitivity, A_{min} , making the SNR_{min} get worse. As a result, the higher σ is, the better performance is obtained.

The performance of some 1024-point designs is depicted in figure 21, which shows the SNR_{min} and dynamic range in dB for a detection probability $P_d = 90\%$ and $P_{fa} = 10^{-3}$ depending on the input noise[6]. Under these circumstances, $SNR_{Omin} = 11\ dB$ according to [30], and, therefore, $SNR_{Tmin}(dB) = -16.09\ dB$. Following this, the theoretical DR is obtained as:

$$DR_T(dB) = SNR_{max} - SNR_{Tmin} = 10 \cdot \log \frac{(2^{b-1} - 1)^2}{2\sigma^2} + 16.09 \tag{8}$$

In order to be able to compare 8 and 16-bit architectures, σ for the 16-bit architecture is normalized by 2^8 in figure 21. From the analysis of this plot important conclusions can be drawn.

First of all, all FFTs with the same number of points have the same performance for high values of σ due to the fact that the truncation effects are not significant in this part of the curves. Moreover, the minimum reachable sensitivity of all designs is similar to the theoretical one and only depends on the number of points of the FFT. Therefore, the SNR_{min} decreases 3 dB and the dynamic range increases 3 dB when the number of points is doubled.

On the other hand, radix 4 architectures offer better performance than the radix 2 ones due to the larger number of rotators used in the radix 2 designs. Additionally, radix 4 architectures almost always take up less area than the radix 2 ones, so radix 4 is usually the best choice.

The ratio in dB between the maximum amplitude at the input of an ADC without truncation and the quantization level for an ADC with b bits is $6 \times b\ dB$. Thus, a maximum DR of 48 dB could be expected for an 8-bit FFT. However, as is shown in figure 21, a dynamic range of about 54 dB may be obtained with an 8-bit FFT without truncation. Consequently, signals with an amplitude lower than a quantization level can be detected. On the other hand, the results for a 16-bit FFT with truncation are

[6]Channels with different noise statistics due to the rotator error have not been included in the calculations.

better than the 8-bit implementations without truncation. However, it must be considered that a 16-bit FFT could achieve a DR higher than $6 \times b = 96$ dB and, due to the butterfly truncation, it can only obtain a DR of 62 dB and the lowest detectable signal has an amplitude $A_{min} \approx 25$. Therefore, five of the less significant bits are misused. As a result, instead of using a large number of bits and truncation, it may be more interesting to use an FFT with less bits and without truncation.

8.2.1 Monobit FFT

Rotators are the only difference between the implementations of the conventional and monobit FFTs. In the monobit FFT the rotations are accomplished by swapping the real and imaginary components of the signal and/or changing the sign of the components. Consequently, although the rotation angles are an approximation to the ones in the FFT, there exist no error in the rotations, and the gain of the monobit rotator is always $k = 1$ in Eq. 5.

On the other hand, the processing gain of the monobit FFT for centered sinusoids depends on the frequency bin and is always lower than the one of the conventional FFT, as is shown in figure 22 for the case of 1024 points and radix 4. This point was discussed in detail in [23]. The monobit implementation of the FFT algorithm also produces a degradation in the sidelobe levels which cannot be improved by windowing. The average of the highest sidelobe level is 9-10 dB below the mainlobe independent of the implementation and the number of points of the FFT (number of filters in the filter bank). This has a direct impact on the instantaneous dynamic range [4, 23].

Fig. 23 represents the sensitivity and the dynamic range for a 1024-point FFT and two frequency bins (bin 127 with processing gain G=24 dB, and bin=128 with G=26 dB). Two different implementations are analyzed: An FFT without truncation and 8-bit input bitwidth and an FFT with truncation and 16-bit input bitwith[7]. The same comments for Fig. 21 apply here. However, additional features appear in the monobit implementations. On the one hand, the discrepancies for high σ are due to the non-constant processing gains throughout the filter bank generated with the FFT. On the other hand, the improvement in the dynamic range and the lower deterioration in the sensitivity compared to the conventional implementation is related to the gain of the monobit rotator: $k = 1$.

[7]In order to be able to compare 8 and 16-bit architectures, σ for the 16-bit architecture is normalized by 2^8.

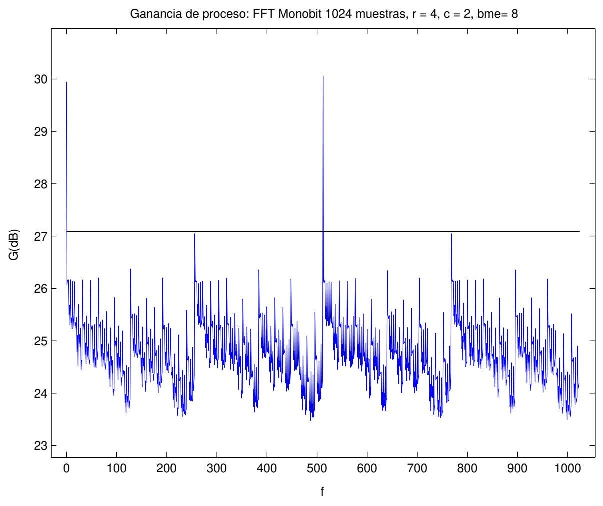


Figure 22: Gain of a 1024-point monobit FFT. Radix 4.

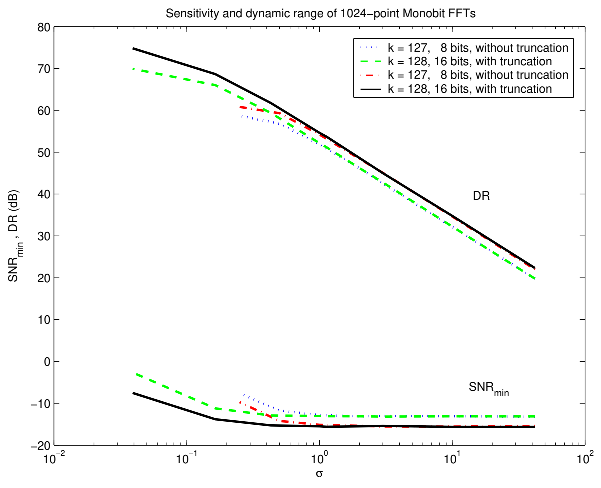


Figure 23: Sensitivity and dynamic range of a 1024-point monobit FFTs. Radix 4.

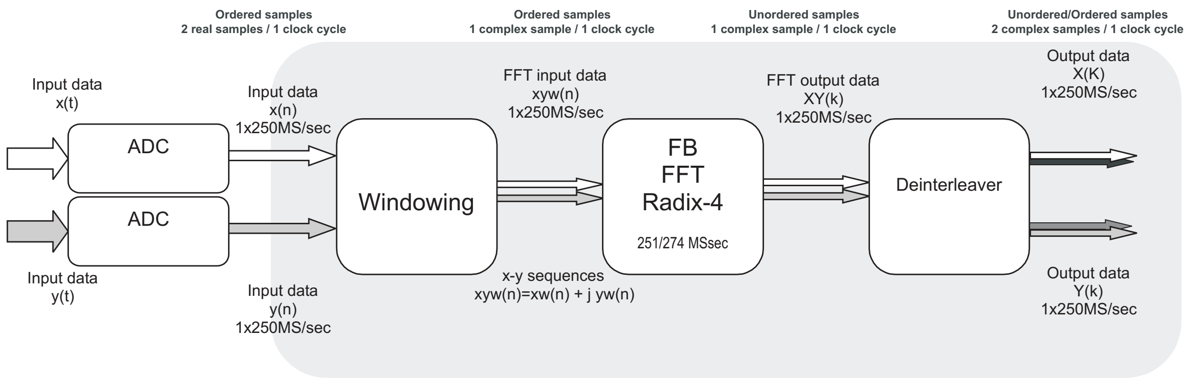


Figure 24: System diagram with the FB architecture.

9 Implementation of FFT-based digital channelized receivers

Even though the basic implementation of digital channelized receivers is based on the FFT algorithm, additional elements are required. This section is devoted to the analysis and design of the whole system, because the configuration and implementation of all the elements involved can significantly influence the final performance of the receiver. In this sense, windowing or ADCs may play an important role in the system because some FPGA resources may be required for its implementation.

Figure 24 illustrates the way the FB architecture can be used in a digital channelized receiver. In this case, a 1024 radix-4 FB FFT without (with) truncation can process 251 (274) MS/s, what requires an input data rate limited to 250 MS/s. Given that the FB FFT can process a sample per clock cycle, the input flow could be digitalized with a simple ADC (from $x(t)$ to $x(n)$). However, in the figure we can see

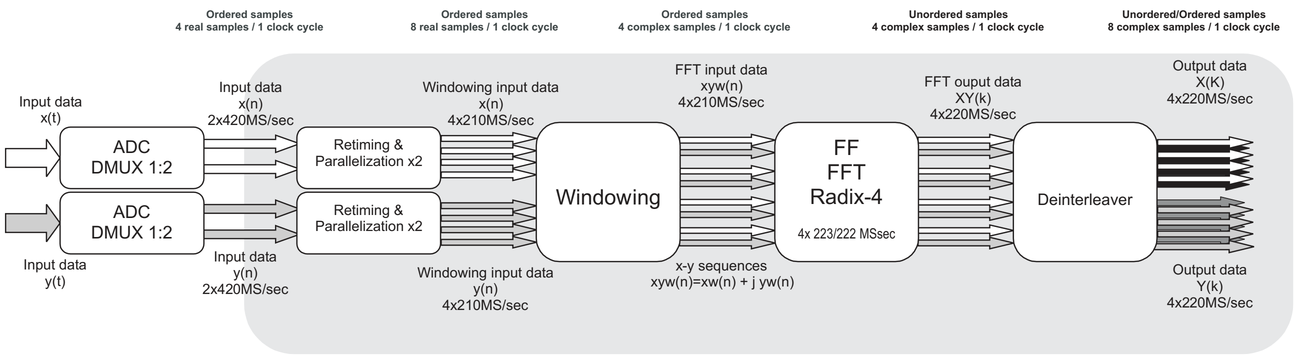


Figure 25: System diagram with the FF architecture.

how two analog signals come into the structure, $x(t)$ and $y(t)$, to produce two flows of real samples. This is due to the computation of the FFT for complex data sequences, that can process two real FFTs with a single architecture. After windowing, the two flows are mixed ($xyw(n)$ in figure 24) and an additional deinterleaving stage is needed to split the FFT output flows and order the complex transforms ($X(k)$, $Y(k)$).

Figure 25 illustrates how the FF architecture can be used in a digital channelized receiver, in a similar way to the FB architecture but with a higher degree of parallelism. The FF FFT can process several samples per clock cycle, and the throughput depends basically on the radix. For a radix 4 implementation four samples must be input to the structure per clock cycle, as shown in figure 25. The clock speed can be 222 MHz for both implementations with and without truncation, providing a processing speed of 888 MS/s. In this case, the input flow must be digitalized with a fast ADC and a demultiplexor has to be used, in addition to a parallelization stage. With this, the input flow, $x(t)$, can provide the four digital samples required by the FF architecture, $x(n)$. Again, we can see in the figure two sample flows coming into the structure taking advantage of the complex processing.

Finally, it is well known that the Input/Output data rate may be a serious bottleneck in current signal processing systems. In this sense, Xilinx FPGAs Virtex-II and Virtex-II Pro families can reach a maximum clock frecuency of 420 MHz. This clock frequency is also the limit for the frequency of the standart I/O on FGPAs (420 Mbps). However, it is possible to duplicate the I/O data rate to 820 Mbps using DDR signaling, what means that data can change on each edge of the clock, or differential signaling

(LVDS). The new Virtex-4 family, with a maximum internal clock of 500 MHz, provides IO data rates that are even higher. This family provides up to 600 MHz for standart IO (HSTL & SSTL), and up to 1 Gbps with DDR and differential signaling.

With these I/O data rates it is not possible to interconnect directly an FPGA with a high performance ADC. For example, the ADC TS83102G0B [31] can produce samples of 10 bits up to 2 GS/s. In this case, we can solve this problem by using a demultiplexor (DMUX) between the ADC and the FPGA, for instance the AT84CS001, as was depicted in figure 25. This component allows configurations of 1:4 (what results in 500 Mbps in each pin, being therefore necessary to use DDR and LVDS in the FPGA) or 1:8 (250 Mbps in each pin, data rate in the limit of the FPGA standart I/O). Actually, the last commercial version of this ADC integrates the DMUX to simplify the interface with FPGAs. For example the part AT84AS004 [31] is an ADC of 10 bits and 2GS/s with and integrated DMUX of 1:4.

10 Conclusions

We have presented a comparative study of parallel pipelined architectures of the FFT algorithm targeting FPGA devices for the implementation of digital channelized receivers. The in-depth exploration of the FFT design space has been carried out with the help of a developed automatic tool. Both the digital circuit designer point of view -where area, throughput, and latency are the main targets- and the system designer perpective - where signal processing capabilities and power consumption are the main concerns- are taken into account for a joint assessment. From our analysis we can conclude that the FF architecture offers the optimum throughput at the expense of a higher power consumption, which will be reduced in the new generation of FPGAs. On the other hand, the FB architecture is the optimum solution if the area and power requirements are critical. A monobit version of both architectures can improve area, throughput, and consumption with a degradation of the detection capabilities: It is suitable for the detection of a single signal.

The experimental results have shown how the FB architecture requieres lower area than the FF architecture but the last one allows to parallelize samples, which increases the throughput. Consequently, FB structures can be used for large number of points FFTs, while feedforward architectures are better suited for applications with hard real-time constraints.

References

[1] S. Hauck, "The Roles of FPGA's in Reprogrammable Systems ," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[2] N. Lall, "New XtremeDSP Slices Deliver As Much As 10X More GMACs Per Dollar," *Xcell Journal*, Winter 2004.

[3] J.Tsui, *Microwave Receivers with Electronic Warfare Applications.* John Wiley & Sons, 1986.

[4] ——, *"Digital Techniques for Wideband Receivers".* Artech House, 2001.

[5] J. Tsui and P. J. Stephens, "Digital microwave receiver technology," *IEEE Trans. on Microwave Theory and Techniques*, vol. 50, March 2002.

[6] C.-I. Chen, K. George, W. McCormick, J. Tsui, S. Hary, and K. Graves, "Design and performance evaluation of a 2.5-GSPS digital receiver," *IEEE Trans. on Instrumentation and Measurement*, vol. 54, no. 3, pp. 1089–1099, June 2005.

[7] D. Zahirniak, D. Sharpin, and T. Fields, "A hardware-efficient, multirate, digital channelized receiver architecture," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 34, no. 1, pp. 137–152, January 1998.

[8] K. Parhi, *Digital Signal Processing Systems-Design and Implementation.* John Wiley & Sons, 1999.

[9] M. Sanchez, M. Garrido, M. López-Vallejo, J. Grajal, and C. López-Barrio, "Digital Channelised Receivers on FPGA Platforms," in *Proc. IEEE International Radar Conference*, May 2005, pp. 816 – 821.

[10] C. Yun-Nan and K. Parhi, "An efficient pipelined FFT architecture," *IEEE trans. on Circuits and Systems II*, vol. 50, no. 6, pp. 322–325, 2003.

[11] S. Moon and I. Park, "Area-Efficient Memory-based Architecture for FFT Processing," in *Proc. ISCAS*, May 2003, pp. 25–28.

[12] Z. Guoping and F. Chen, "Parallel FFT with CORDIC for Ultra Wide Band," in *Proc. 15th IEEE PIMRC*, September 2004, pp. 1173–1177.

[13] I. Uzun, A. Amira, A. AhmedSaid, and F. Bensaali, "Towards a general framework for an FPGA-based FFT coprocessor," in *Proc. 7th Intl. Symposium on Signal Processing and Its Applications*, 2003, pp. 617–620.

[14] S. Sukhsawas and K. Benkrid, "A High-level Implementation of a High Performance Pipeline FFT on Virtex-E FPGAs," in *Proc. IEEE Computer society Annual Symposium*, February 2004, pp. 229–232.

[15] M. Grandmaison, J. Belzile, C. Thibeault, and F. Gagnon, "Reconfigurable and efficient FFT/IFFT architecture," in *Proc. Canadian Conference on Electrical and Computer Engineering*, May 2004, pp. 1115–1118.

[16] S. Choi, G. Govindu, J. Ju-Wook, and V. Prasanna, "Energy-efficient and parameterized designs for fast fourier transform on FPGAs," in *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 2003, pp. 521–524.

[17] T. Sansaloni, A. Pérez-Pascual, and J. Valls, "Area-efficient FPGA-based FFT processor," *Electronics Letters*, vol. 39, no. 19, 2003.

[18] J. E. Volder, "The CORDIC Trogonometric Computing Technique," *IRE Trans. on Electronic Computing*, 1959.

[19] J.-M. Rémondeau, "Scalable parallel architecture for ultra fast FFT in an FPGA," in *Proc. ICSPAT*, 1999.

[20] G. Szedo, V. Yang, and C. Dick, "High-performance FFT processing using reconfigurable logic," in *35 Asilomar Conference on Signals, Systems and Computers*, 2001, pp. 1353–1356.

[21] J. W. Cooley and J. W. Tukey, "An algorithm for machine calculation of complex Fourier series"," *Math. Comp.*, vol. 19, 1965.

[22] D. Pok, C. Chen, J. Schamus, C. Montgomery, and J. Tsui, "Chip design for monobit receiver," *IEEE Trans. on Microwave Theory and Techniques*, vol. 45, December 1997.

[23] J. Grajal, R. Blázquez, G. López-Risueño, J. M. Sanz, M. Burgos, and A. Asensio, "Analysis and characterisation of a monobit receiver for electronic warfare," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 39, no. 1, January 2003.

[24] M. Sanchez, A. Fernández, and M. López-Vallejo, "xHDL: Extending VHDL to Improve Core Parametrization and Reuse," in *Advances in Design and Specification Languages for SoCs*. Springer, 2005.

[25] Xilinx Inc., "Xilinx LogiCore: Fast Fourier Transform v3.1," 2004, http://www.xilinx.com/products/Broadband/.

[26] G. Nordin, P. A. Milder, J. C. Hoe, and M. Püschel, "Automatic Generation of Customized Discrete Fourier Transform IPs," in *Proc. Design Automation Conference (DAC)*, 2005.

[27] W. Jun, M. Shiyi, and W. Yuezhong, "Design and implementation of a high speed vector processor for real-time sar imaging," in *Proc. CIE International Conference on Radar*, 2001.

[28] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *ACM Intl. Symposium on Field Programmable Gate Arrays*, 2003, pp. 175–184.

[29] D. Curd, "Power Consumption in 65 nm FPGAs," 2006, white paper: Virtex-5 Family of FPGAs. http://www.xilinx.com.

[30] M. I. Skolnik, *Introduction to Radar Systems, 3rd ed.* McGraw-Hill, 2001.

[31] ATMEL Corporation, "Broadband Data Conversion Products," 2005, http://www.atmel.com/products/Broadband/.