Implementing FFT-Based Digital Channelized Receivers on FPGA Platforms

This paper presents an in-depth study of the implementation and characterization of fast Fourier transform (FFT) pipelined architectures suitable for broadband digital channelized receivers. When implementing the FFT algorithm on field-programmable gate array (FPGA) platforms, the primary goal is to maximize throughput and minimize area. Feedback and feedforward architectures have been analyzed 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 achieve faster real time performance in broadband digital receivers. The influence of the hardware implementation on the performance of digital channelized receivers has been analyzed in depth, revealing interesting implementation trade-offs which should be taken into account when designing this kind of signal processing systems on FPGA platforms.

I. INTRODUCTION

Following Moore's law, current submicron technologies have allowed extraordinary integration densities in digital circuits. However, as processes scale down, uncertainty increases (voltage, temperature, noise, coupling etc.) and the design process becomes more complicated, especially for ASICs (application specific integrated circuit), where margins are extremely tight. Moreover, mask costs have reached a critical point, dominating the manufacturing process and requiring high financial risk.

In this context, field-programmable gate arrays (FPGAs) offer significant advantages at a suitable low cost [1]. First, the flexibility of FPGAs provides 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 required by ASICs. Finally, even though FPGAs are not as efficient as ASICs in terms of performance, area or power, it is true that now they can provide better performance than general purpose CPUs 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. Moreover, due to inclusion of 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 receivers are a good example of complex systems with stringent requirements [3, 4]. The requirements for these receivers are wideband frequency coverage, high sensitivity and dynamic range, high probability of intercept, simultaneous signal detection, excellent frequency resolution, and real time operation. A classical receiver which accomplishes these requirements is a channelized receiver 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 wideband digital channelized receivers [4, 5, 6, 7, 8, 9]. Fundamental problems in the analog receivers such as the imbalance between receiver channels can be surpassed by migrating to digital implementations. Digital implementations are inherently reliable and accurate. However, broadband digital channelized receivers, mainly based on fast Fourier transform (FFT) related processing, require intensive computation for real time applications. Typically, the system throughput of many signal processing algorithms can be improved by exploiting concurrency in the form of parallelism and pipelining [10]. FPGAs allow for true parallel processing by supporting multiple simultaneous threads of execution. Moreover, 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.

Even though FPGAs offer new possibilities to the system designer, additional support is required to explore the design space because new design magnitudes must be considered. Typical hardware variables such as area, clock frequency, or power dissipation should be evaluated together with classical signal processing issues such as throughput, detection performance, and dynamic range. In this paper we present an in-depth study on the hardware implementation of FFT pipelined architectures suitable for broadband digital channelized receivers with continuous flow of input samples. Our purpose is to provide an analysis and comparison of existing FFT algorithms to help the signal processing engineer when selecting the best hardware architecture for a given application. Selection can be based on a broad range of parameters at two different levels: low level hardware implementation details (bitwidth, bit truncation, area, clock frequency, etc.) or high

Manuscript received July 21, 2006; revised June 15 and November 23, 2007; released for publication March 22, 2008.

IEEE Log No. T-AES/44/4/930740.

Refereeing of this contribution was handled by J. Lee.

0018-9251/08/$25.00 © 2008 IEEE

level application details (FFT radix and number of points, detection capabilities, etc.). Furthermore, the use of simplified FFT algorithms such as the monobit FFT [11, 12] is also considered. The exploration of all the implementation possibilities is supported by a high level design exploration tool especially developed for this purpose [13], which alleviates the system engineer's work by hiding low level hardware details.

Important work has been carried out on hardware FFT architectures, in particular for FPGA implementation [14, 15, 16], but all those previous works represent a partial approach to the problem. In this paper we provide a global view of the problem, paying attention not only to the design of the FFT architectures, but also analyzing all important parameters involved in their hardware implementation on FPGAs and studying the performance they provide from the detection point of view.

The structure of the paper is the following. A general description of the FFT algorithm and its basic implementation is described in Section II, while Section III describes the selected architectures in this work: feedback and feedforward. Experimental results are analyzed in Sections IV and V. Finally, the implementation of a channelized receiver based on the proposed architectures is described in Section VI and some conclusions are drawn.

II. THE FFT ALGORITHM

In this section we review the main equations of the discrete Fourier transform (DFT) to point out their mapping on hardware. The N-point DFT (DFT^N), $X_N(k)$ for a given sequence $x_N(n)$ is defined as

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

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

The FFT is based on the decomposition of a sequence of the DFT into lower order computations, which results in a reduction in the number of operations. In this way the complexity of the computations is reduced from $O(N^2)$ to $O(N \log_2 N)$.

There are many algorithms that compute the FFT. The one proposed by Cooley-Tukey [17], which provides a very regular hardware implementation, 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 a 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). Even though this work focuses on DIF implementations, it could be easily extended to DIT implementations.

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

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

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

which is the decomposition of the DFT into two sub-DFTs with lower order. Successive 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:

$$T_N^m(n) = \exp -\frac{j2\pi m}{N} n \quad \text{with} \quad m = 0 \ldots (R-1): \tag{3}$$

Several issues can be outlined when analyzing the equations in (2). First, the input samples can be complex numbers. Second, the most internal operands of the middle equations (between brackets) represent a mixture of samples 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 which can be manipulated 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 maximize the throughput of the implemented FFT, measured as processed Msamples/s, the

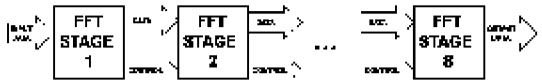

Fig. 1. Pipelined implementation of FFT.

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 [11, 12]. With this approximation the rotators always use the following angles: 0, ¼=2, ¼, or ¡¼=2. Hence, rotators are not necessary and this results in significant reduction in area and improvement in speed. However, this increase in the throughput is obtained at the expense of performance degradation. In the application described in this paper, there exists a degradation on the detection probability and dynamic range of the channelized receiver, as analyzed in Section V (see also [12] for more details).

III. HARDWARE ARCHITECTURES FOR THE FFT

There are many hardware implementations for both DIF and DIT algorithms. For instance, we can choose between digit-serial or bit-parallel arithmetic, or we can select between pipelined or iterative implementations. In the case of data-oriented applications presenting a continuous flow of samples, the best architectures are those that favor speed over area. The implementations that better fit these requirements are bit-parallel and pipelined architectures, where the processing is performed in several cascaded stages, as can be seen in Fig. 1. We have chosen two main groups of FFT architectures, representing opposite points in the area-performance design space: feedback (FB) and feedforward (FF) architectures. Architectures with FB 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. On the other hand, FF structures provide a higher throughput (R samples per clock cycle, R being the radix) 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 II) and each stage performs the decomposition of the input sequence into sub-DFTs, which are 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 Fig. 2: a memory where data between stages are stored, a butterfly where the mixture of samples is accomplished, and finally an

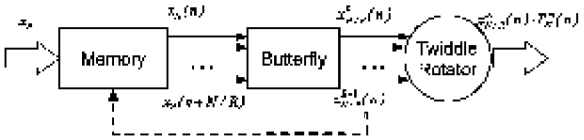

Fig. 2. Structure of stage of proposed architectures.

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 Fig. 2 represents the FB 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 reordering phase left. This is an optional task, because output samples are not completely unordered, as they have a known sequence that can be considered as input for the next processing step.

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

A. 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 $0^{\pm}$, 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 $\pm 90^{\pm}$ and can be implemented as a swap of the sample components with sign change. Other angles like $\pm 45^{\pm}$ or $\pm 135^{\pm}$, which are present in radix-8 architectures, can be implemented by two multipliers by real constants. For radix $R > 8$ butterflies, nontrivial twiddles appear, with the number of these nontrivial twiddles increasing with the radix. In these cases complex multipliers must be used and the butterfly implementation requires bigger area. In fact, there is an exponential increase of area with the radix. In general, 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 increase in area precludes the implementation of standard FFTs with radix larger than 8, even though higher radix values reduce the number of stages. This is not the case of the monobit

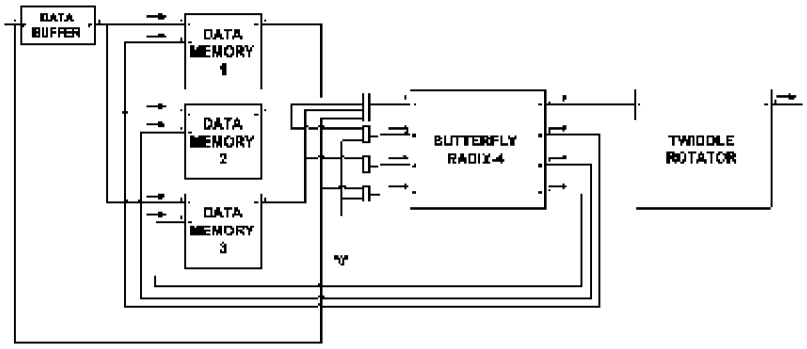

Fig. 3. Structure of one stage of FB architecture (radix 4).

FFT, which allows the use of higher radices because rotations become trivial.

Radix 4 presents the twofold advantage of including simple components and presenting a reduction of the number of stages when compared with 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. If we keep the bitwidth fixed, 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 degradates. This fact is thoroughly analyzed in Sections IV and V.

B. Rotators

Rotators are critical components in the FFT architecture because they require a significant percentage of the total 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 CORDIC algorithm [18], which allows to perform the previously mentioned multiplication by the twiddles without multipliers. 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 [18]. Therefore, to keep the dynamic range of the input samples, this element would have to increase the data bitwidth by 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 input complex components, given by the butterfly, and performs the rotation of a given angle through several rotations using a set of well-defined angles (elemental angles). For the FFT, the angles (twiddle) are known in advance, so for our implementation instead of storing these angles, we compute the rotations of these elementary angles that the twiddle will produce (one bit per elemental angle), and store this information in memory. The amount of memory required is approximately the same as if we store the twiddle angles, but the implementation of the algorithm can be simplified, because now we do not need the hardware that controls the rotations. This implementation saves area and results in a higher clock frequency.

C. Feedback Architecture

The FB architecture, shown in Fig. 3, 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 Fig. 4. A first mode is related to the arrival of samples from 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 Fig. 4, 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.

An FB 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

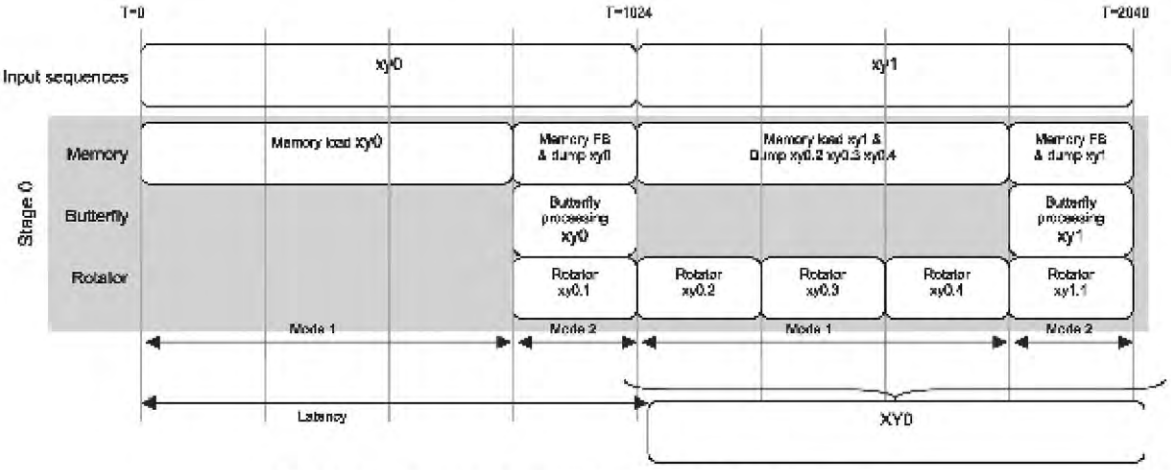

Fig. 4. Time evolution for 1024-point, radix-4, FB architecture.

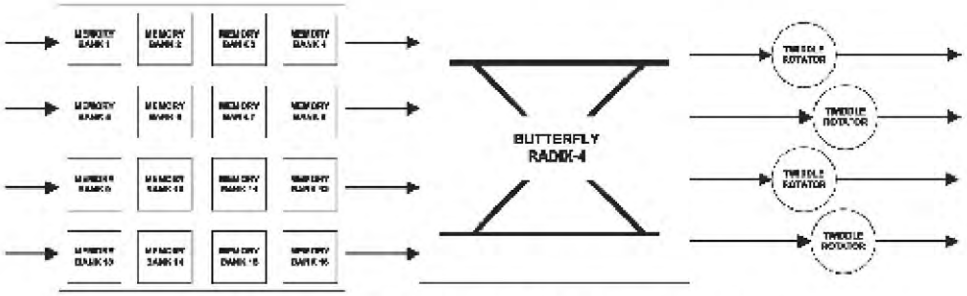

Fig. 5. Structure of one stage of FF architecture (radix-4).

the samples coming from the previous processing phase. During the first processing mode (mode 1 in Fig. 4) 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 FB data are loaded from the butterfly to be processed later in the rotator (mode 2). This particular memory management not only allows temporary storage of data, but also the reordering of samples to be processed in the current stage.

D. Feedforward Architecture

In the FF architecture the samples can go ahead once a stage is processed because there are several rotators (see Fig. 5). 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 Fig. 6, 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-point 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 reordering of data. We have used a matrix memory of R £ R with ping-pong structure. As can be seen in Fig. 6, 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 continuous processing.

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

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.

E. Overview of the Proposed Architectures

To summarize the description of the FFT architectures under study, Table I shows their requirements in terms of basic elements (memory, rotators, and butterflies expressed as adders) for different values of radix R. 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 than for the FB one. This is due to

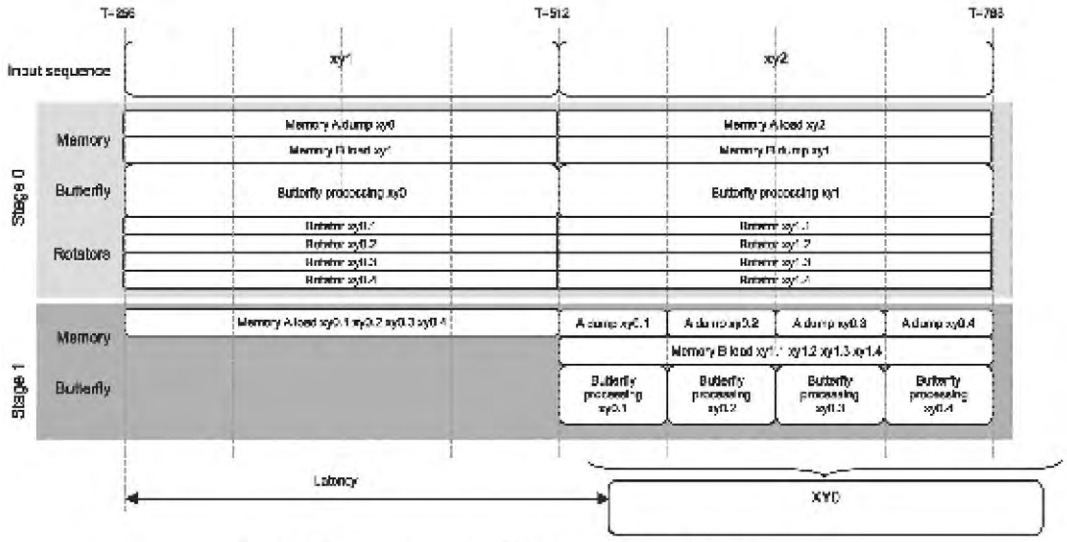

Fig. 6. Time evolution in 1024-point, radix-4, FF architecture.

TABLE I
Hardware Resources Required by FB and FF Architectures

Architecture	Radix	Rotators	Adders	Memory	
				Data	Rotations
Feedback (FB)	2	$S - 2$	$2 \cdot S$	N	N
	4	$S - 1$	$8 \cdot S$	N	N
	8	$S - 1$	$24 \cdot S$	N	N
Feedforward (FF)	2	$S - 2$	$2 \cdot S$	$6 \cdot N$	N
	4	$3 \cdot (S - 1)$	$8 \cdot S$	$4{:}6 \cdot N$	N
	8	$7 \cdot (S - 1)$	$24 \cdot S$	$4{:}3 \cdot N$	N

Note: N is number of points and $S = \log_R N$ is number of stages.

the higher degree of parallelism presented by the FF architecture, which additionally provides better performance in terms of throughput. 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.

IV. 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. To quickly explore all the possibilities offered by the different architectures we have developed a design exploration tool [13], which 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. The key parameters under analysis in all these architectures are:

1) bitwidth of the input samples,
2) number of points N,
3) number of stages S,
4) radix of the implementation R (power of 2),
5) stage scaling (truncation in the butterflies).

The performance of the different solutions is analyzed in terms of:

1) area (slices and memory blocks used[1]),
2) latency, time to process an FFT (from start to end),
3) clock speed (MHz) and throughput (Msamples/s),
4) power.

In order to organize the analysis of the experimental results we will perform it in three different scenarios. The first and second scenarios are devoted to the analysis of FB and FF architectures for both conventional and monobit FFTs, respectively. The third scenario studies the power dissipation of all architectures.

Finally a last scenario is devoted to analyzing the results provided by the FPGA implementations of FFTs when used in a digital channelized receiver, and due to the importance of these results, Section V is devoted to this analysis.

[1]Most Xilinx 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.

TABLE II
Experimental Results for FB and FF Architectures for Conventional FFT

STAGES	POINTS	FFT 8 BITS – FB RADIX 4									
		WITH TRUNCATION					WITHOUT TRUNCATION				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	(ms)	MHz	MSps	SLICES	BRAM	(ms)	MHz	MSps
2	16	638	0	0,128	274	274	806	0	0,139	267	267
3	64	1128	0	0,360	274	274	1604	0	0,400	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	(ms)	MHz	MSps
2	16	1342	0	0,171	251	251
3	64	2402	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	(ms)	MHz	MSps	SLICES	BRAM	(ms)	MHz	MSps
4	16	1135	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	8883	0	4,852	184	368	12478	0	4,482	199	398
9	512	4471	8	3,754	228	456	7418	8	3,894	227	454
10	1024	15442	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	(ms)	MHz	MSps	SLICES	BRAM	(ms)	MHz	MSps
2	16	1235	0	0,105	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	942
5	1024	13031	0	2,929	223	890	17751	0	3,030	222	887

STAGES	POINTS	FFT 16 BITS – FF RADIX 4									
		WITH TRUNCATION					WITHOUT TRUNCATION				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	(ms)	MHz	MSps	SLICES	BRAM	(ms)	MHz	MSps
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	(ms)	MHz	MSps	SLICES	BRAM	(ms)	MHz	MSps
2	64	4716	0	0,170	230	1842	5920	0	0,275	153	1224
3	512	10988	0	0,761	230	1840	15009	0	1,586	116	928

A. Scenario 1. Comparing the FB and FF Architectures

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

Table II summarizes the synthesis results we have obtained for different implementations when exhaustively exploring the design space. 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 been implemented for radix 4. Moreover, implementations with a large number of points have been generated with and without usage of BRAM.

In Figs. 7 and 8 we can see the area-performance trade-off that can be obtained for 8-bit designs in the FF and FB architectures with and without truncation. From a first analysis of Table II 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 II, except for the 1024-point 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 implementations present a slightly higher working frequency, radix-4 implementations have a higher throughput (almost double), due to the higher parallelization of the operations. As is well known, the final speed obtained in an FPGA depends on the particular place and route that is carried out, which directly impacts on the length of the critical path. In this sense, it is interesting to remark that for larger transforms the resulting frequency is smaller, due to the higher complexity of interconnections.

4) The latency values of Table II refer to the total computation of an FFT. Furthermore, other

TABLE III
Results for Different 16 Bit 1024-Point FFTs (Radix 4)

	SLICES	BRAM		MULT18x18		I_SLICES	MHz	Msec	kS/SLICE
		Num	EqSlices	Num	EqSlice				
XTREMX	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	34400	16	5120	61029	167	668	10,946
SPIRAL P4_TH128	3287	16	13600	16	5120	22007	167	668	30,354

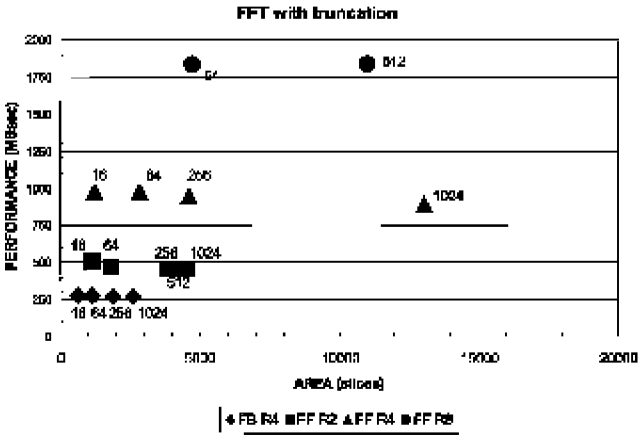

Fig. 7. Area-performance plot for FB and FF architectures (8 bits, with truncation).

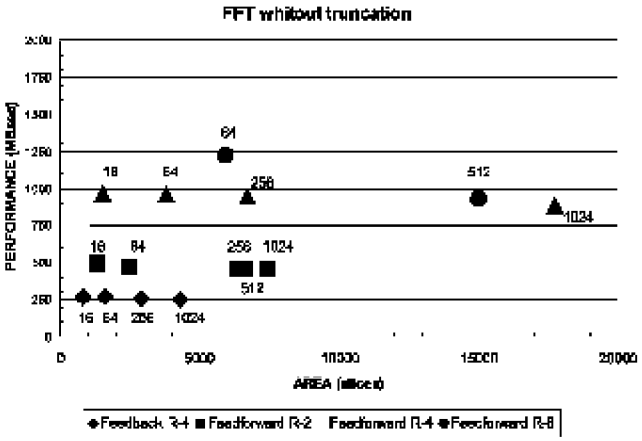

Fig. 8. Area-performance plot for FB and FF architectures (8 bits, without truncation).

computations can run simultaneously in the FPGA, which cannot be done in a DSP [19].

5) The use of BRAM blocks has only been worthwhile in those examples that require large memory 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 BRAM instead of distributed memory.[2]

1) *Comparative Study with Other Implementations*: There are only a few tools able to generate FFT cores. The Xilinx LogiCore is a well-known example [20], but the degree of parameterization of this tool is significantly reduced when compared with our approach. Another interesting approach, SPIRAL [21], 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. We have generated with these tools different implementations of a 1024-point FFT with 16 input bits and no growth through the stages. In Table III, 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 III 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, and this 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 measure all area-related issues in slices, which is the only component present in all FPGA families. We should not forget that the idea is to implement the whole system on the FPGA, being therefore convenient to leave special purpose resources (RAM, multipliers:::) for other design modules. 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 16 £ 16 pipelined multiplier, which occupies around 350 slices. We have not implemented the 18 £ 18 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

[2]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 FFT generation tool can modify this threshold as convenient.

TABLE IV
Experimental Results for Monobit FFT Implementations

		METRIC TRUNCATION									
STAGES	FFT_SIZE	FFT MONOBIT 16 BITS - FB RADIX 4					FFT MONOBIT 16 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		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
STAGES	FFT_SIZE	FFT MONOBIT 8 BITS - FB RADIX 4					FFT MONOBIT 8 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	496	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

		METRIC TRUNCATION									
STAGES	FFT_SIZE	FFT MONOBIT 8 BITS - FB RADIX 4					FFT MONOBIT 8 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		SLICES	BRAM	usec	MHz	MSsec	SLICES	BRAM	usec	MHz	MSsec
2	16	997	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
STAGES	FFT_SIZE	FFT MONOBIT 4 BITS - FB RADIX 4					FFT MONOBIT 4 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		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
STAGES	FFT_SIZE	FFT MONOBIT 2 BITS - FB RADIX 4					FFT MONOBIT 2 BITS - FF RADIX 4				
		AREA		LATENCY	SPEED		AREA		LATENCY	SPEED	
		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

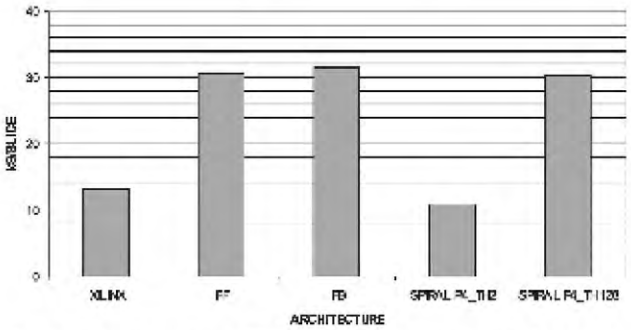

Fig. 9. Comparison of FB and FF architectures with Spiral and Xilinx in terms of KSsec/slice.

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).[3] This new measure has been plotted in Fig. 9 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 III), but with a performance comparable to 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

[3]The clock frequency associated to a given implementation depends on the particular architecture; we have therefore considered that a measure based on the number of samples processed per second is more fair. Moreover, the cost associated to that performance is given by the logic required (number of slices in an FPGA).

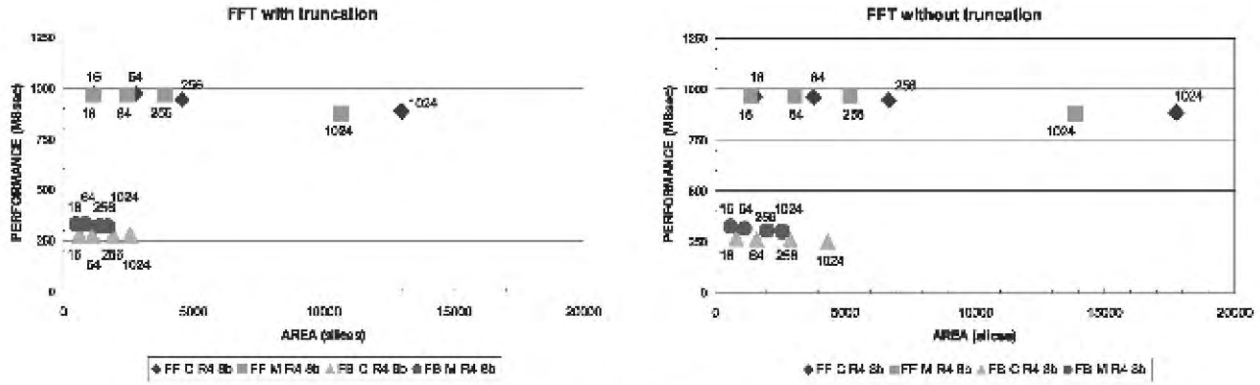

Fig. 10. Conventional versus monobit FFTs (with and without truncation).

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.

B. Scenario 2. Analysis of the Monobit FFT Implementation

In this scenario we study the results obtained for the FB and FF architectures when implementing the monobit FFT. The monobit FFT [11, 12] simplifies the computation of the FFT kernel by approximating the rotations with simple angles, 0, ¼=2, ¼, or ¡¼=2, which makes unnecessary any multiplier or rotator. This results in a clear reduction of the final area and a significant improvement in speed. 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 IV 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 V.

If we compare Tables II and IV we can observe how the monobit implementations present a variable increase in speed and a clear reduction in area and latency. In particular, the area of FB architectures is reduced a 20% together with a 20% improvement in speed. Fig. 10 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 FB 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, which is in the critical path, by a monobit one, which is faster. 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.

TABLE V
Analysis of Influence of Radix

RADIX	POINTS	FFT MONOBIT 2 BITS – FF				
		AREA		LATENCY	SPEED	
		slices	DSP	usec	MS/s	KS/cc
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

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

C. 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.

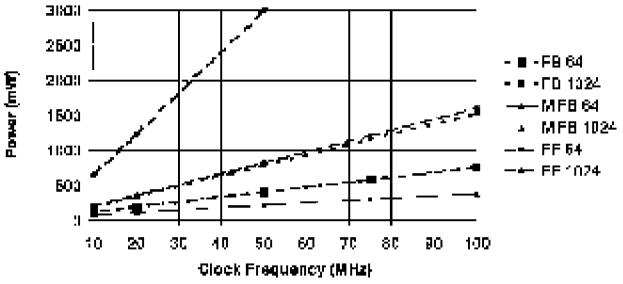

Fig. 11. Dynamic power dissipation for different FFT architectures (8 bits, radix 4, with truncation, Virtex-II FPGA).

We have evaluated the power dissipation that our FFT architectures present with the XPower tool, from Xilinx, and represented it in Fig. 11. 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 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 both logic and interconnection area). We should remark that most dynamic power in FPGAs is consumed in the interconnection resources (70%, as studied by [22]). For FFTs of the same length, the FF architecture always presents greater power consumption than the FB one. Moreover, the monobit simplification obtained in terms of area can also be observed in terms of power. Even though the power results plotted in Fig. 11 are very high, recent advances in FPGA technology [23] 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 with 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 Fig. 11, 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.

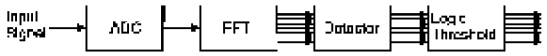

Fig. 12. Schematic diagram of analyzed channelized receiver.

V. ANALYSIS OF A CHANNELIZED RECEIVER PERFORMANCE BASED ON THE PROPOSED FFT ARCHITECTURES

The channelized receiver to be characterized is depicted in Fig. 12. This receiver consists of an ADC followed by an FFT-based filter bank. Finally, a decision is taken applying a linear detector to the outputs of the filter bank.[4]

We have studied this channelized receiver 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 ¾ which is bandlimited to the Nyquist frequency. False alarm probability, detection probability, and dynamic range of several FFT implementations have been analyzed in order to determine their performance in terms of input data bitwidth, truncation along the FFT stages, radix, and number of points N. Throughout this section a rectangular window without normalization has been employed.

It might be important to note that the performance of the FB and FF implementations is identical because they are different hardware implementations of the same algorithm.

A. False Alarm Probability

The impact of some hardware design considerations (rotator error and butterfly truncation) on the false alarm probability is analyzed. All false alarm probability simulations have been obtained using the Monte Carlo method with 10^6 independent trials. The simulated false alarm probability P_{fa} in these figures is obtained as an average of the probabilities through the channels of the FFT. All the channels in the FFT, except channels 0 and N=2, have the same theoretical statistics, and, therefore, the same theoretical P_{fa}. However, the lowest channels in the implemented FFTs are more affected by the

[4]From the detection point of view, both linear and quadratic detectors have the same sensitivity in this receiver [24], although the threshold for the linear detector T_L is the square root of the threshold for the quadratic detector T_Q: $T_L^2 = T_Q$. From the implementation point of view, a linear detector implies a square root if we calculate the modulus following the straighforward definition. However, the Cordic algorithm can also be used to determine the modulus without multiplications and square roots. On the other hand, the number of bits to represent the modulus is half the number of bits to represent the output of a quadratic detector.

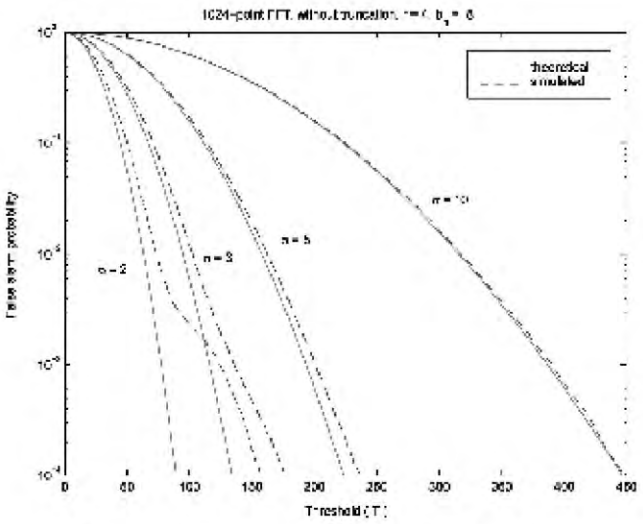

Fig. 13. P_{fa} of 1024-point FFT without truncation.

rotator errors, which results in different statistics at their outputs (different means), and, therefore, higher thresholds must be used to fix P_{fa} to the selected value in those channels.

The theoretical false alarm probability per channel for a linear detector is

$$P_{fa} = \exp\left(\frac{-T^2}{2\sigma^2 N k^2}\right) \tag{4}$$

where T is the threshold of the detector, N stands for the number of points of the FFT, and k represents the intrinsic global voltage gain due to the CORDIC rotators [18], which depends on the number of stages.

1) *Impact of Rotator Error*: The first design under study is a 1024-point FFT, radix 4, without truncation in the butterflies, and with an input bitwidth of 8 bits.[5] Fig. 13 shows the averaged false alarm probability per channel for different values of noise standard deviation σ. When $\sigma \geq 5$ the experimental and the theoretical results are quite similar. As σ decreases, the discrepancies between theory and implementations increase. An in-depth analysis of this problem has shown that the deviation of the theoretical and the simulated results is more significant for the lowest channels of the FFT. This is shown in Fig. 14, where we have reproduced the results for $\sigma = 3$ from Fig. 13. We have also added the averaged false alarm probability per channel when channels 1 to 3 are not considered, which is very close to the theoretical one. The lowest channels have different statistics at their outputs, and, therefore, higher thresholds must be used to fix P_{fa} to the selected value in these channels.

There are two reasons for this behavior: first, the modification of the statistics at the input of the FFT processor due to quantization noise of the ADC that is more significant for low input power noise (low σ); second, it can be shown that the propagation of this quantization noise through an FFT architecture based on nonideal rotators has more influence on the lowest channels.

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.

Results for radix 2 and $\sigma = 3$ are depicted in Fig. 15. It can be noted that the effects of the different

[5] As the input is represented with 8 bits, the signal amplitude ranges from -128 to 127.

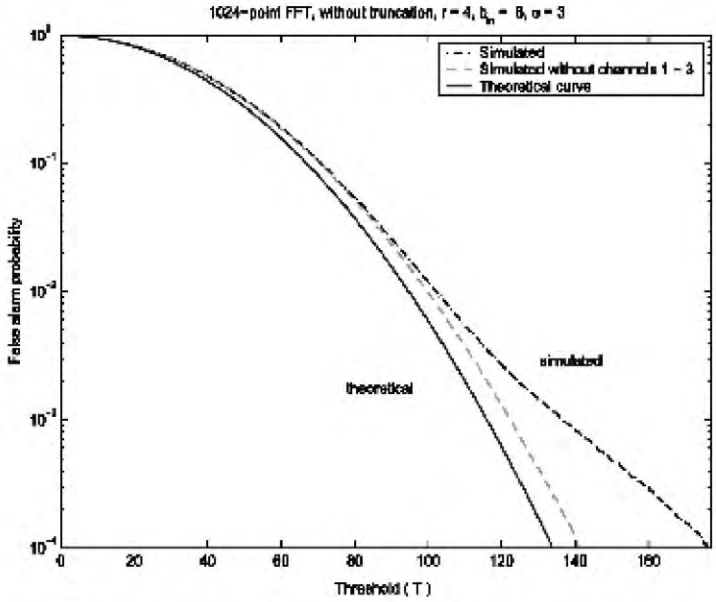

Fig. 14. P_{fa} of 1024-point radix-4 FFT without truncation and $\sigma = 3$.

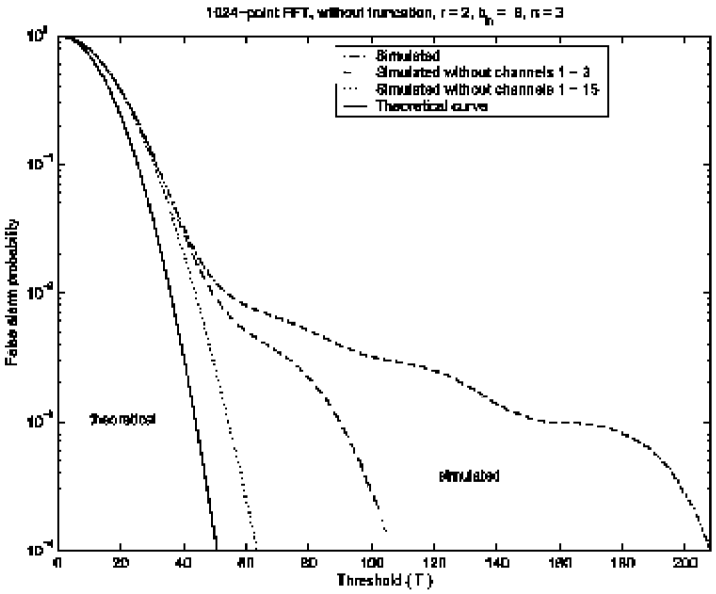

Fig. 15. P_{fa} of 1024-point, radix 2 FFT, without truncation and ¾ = 3.

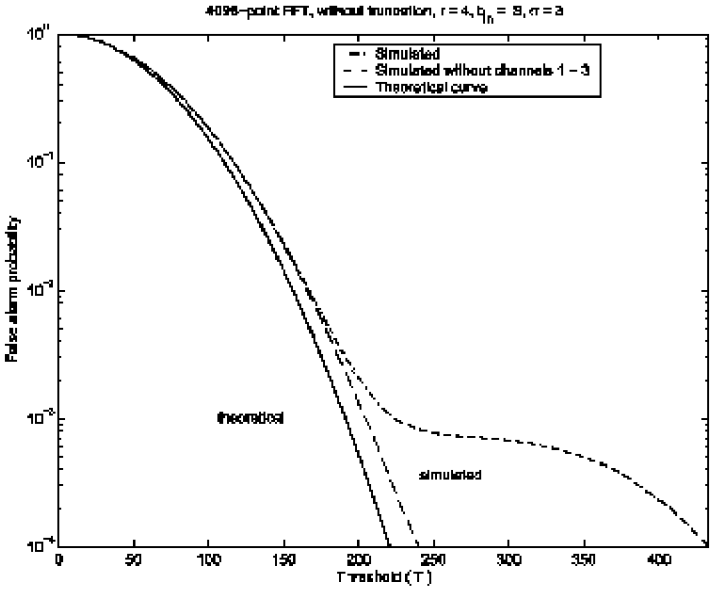

Fig. 16. P_{fa} of 4096-point FFT, radix 4, without truncation and ¾ = 3.

statistics in the the first channels due to errors in the rotators are more pronounced: channels 1 to 15 must be eliminated in the calculations of the averaged P_{fa} instead of channels 1 to 3 in Fig. 14 in order to follow an exponential-like false alarm probability.

Number of Points. When the number of points of the FFT increases, two circumstances must be considered. First, the FFT has more stages, so more rotators are used. Second, the difference between two rotation angles is smaller, so that more accurate rotations must be performed.

Fig. 16 shows the P_{fa} of a 4096-point FFT, without truncation, radix = 4, and input bitwidth of 8 bits. For ¾ = 10 the simulated P_{fa}, not plotted in the figure for readability, is similar to the theoretical one, as it happens with the 1024-point FFT. However, for ¾ = 3 the rotator error affects the results of the FFT, and the effect in the lower channels is more deleterious than that in the 1024-point FFT (Fig. 14).

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 simulations (2 bits per stage for a radix-4 implementation), which leads to a lower performance compared with FFT implementations without truncation. For example, a $P_{fa} = 10^{i\,3}$ cannot be achieved for ¾ = 10. Thus, in order to study the influence of the butterfly truncations, a 1024-point FFT with 16 input bits has been chosen, which is a widely used architecture.

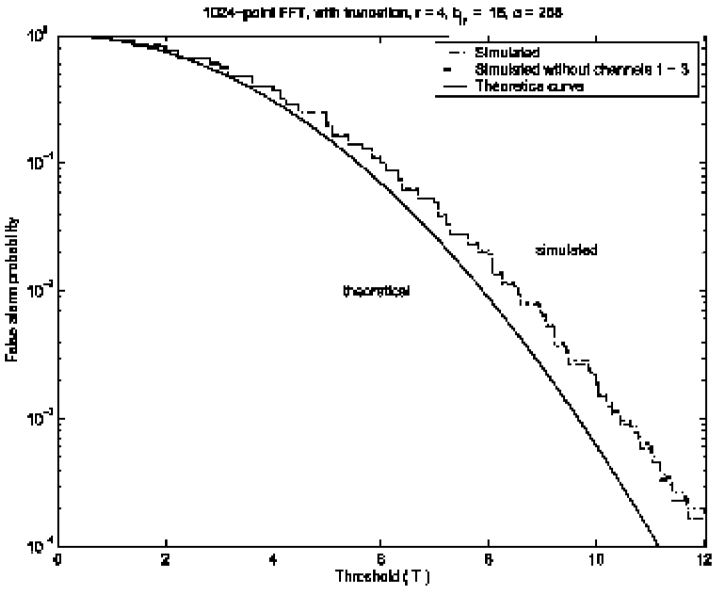

Fig. 17. P_{fa} of 1024-point FFT with truncation, 16 input bits, ¾ = 256.

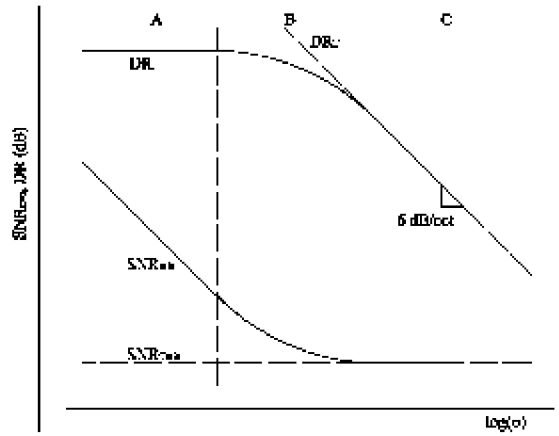

Fig. 18. Typical curves of SNR_{min} and DR.

Firstly, if ¾ is high, the P_{fa} is similar to the theoretical one, so neither the butterfly nor the rotator truncations are significant. Fig. 17 shows the difference between the theoretical and the simulated P_{fa} curves when ¾ = 256:[6] For lower values of ¾, the butterfly truncation modifies the probability density functions of the noise samples. As a consequence, big discrepancies between the simulated P_{fa} and the theoretical one which was deduced for continuous Gaussian noise samples are observed.

B. Detection Probability and Dynamic Range

The impact of some hardware design considerations on detection probability and dynamic range are analyzed. All simulated detection probabilities have been obtained using Monte Carlo simulations with 10^3 independent trials. Channels 0 and N=2 are not considered in the calculations because noise statistics are different in these channels.

The sensitivity $S = A^2_{min}=2$ is defined as the power of a sinusoid of amplitude A_{min} at the input of the channelized digital receiver that assures certain detection probability P_d for a fixed false alarm probability P_{fa}. Thus, the minimum signal-to-noise ratio, SNR_{min}, is the quotient between the sensitivity and the input noise power: $SNR_{min} = (A^2_{min}=2)=¾^2$. The theoretical SNR_{min} can be calculated from the required SNR at the input of the detector, SNR_O, for given P_d and P_{fa} [24], and the FFT processing gain.[7] Thus, the theoretical SNR_{min} for a centered sinusoid and rectangular windowing can be obtained as

$$SNR_{min} = \frac{SNR_{O\,min}}{G} = \frac{SNR_{O\,min}}{N=2}: \tag{5}$$

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

$$SNR_{max} = \frac{A^2_{max}=2}{¾^2} = \frac{(2^{b\,i\,1}\ ¡\ 1)^2=2}{¾^2}: \tag{6}$$

Finally, the dynamic range, DR, is the quotient between the SNR_{max} and the SNR_{min}. It is a well-known result from radar detection theory [24] that an increase in the threshold will reduce the false alarm probability and will cause an increase in the

[6]The stepped shape of the simulated curve results from the quantization noise.

[7]The processing gain for a centered sinusoidal signal and a rectangular window of length N is G = N=2. See [25] for an in-depth analysis and evaluation of the processing gain for different windows, and the frequency straddle loss for noncentered signals.

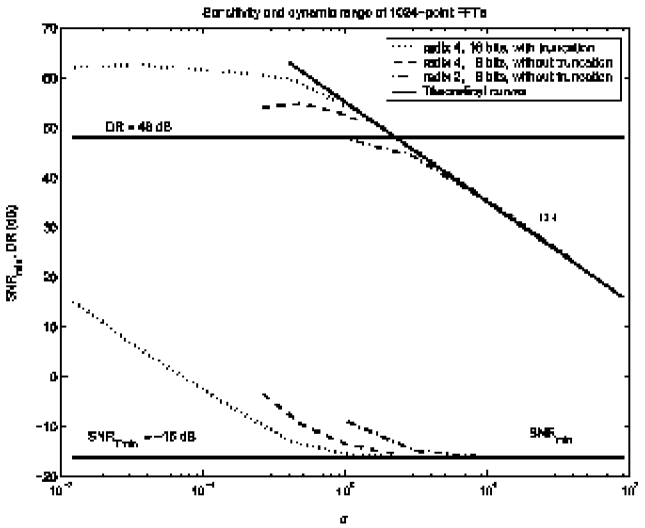

Fig. 19. Sensitivity and dynamic range of 1024-point FFTs. ¾ for 16-bit architecture is normalized by 2^8.

required signal-to-noise ratio to get a fixed detection probability. This increase in the signal-to-noise ratio will also cause a reduction of the dynamic range.

Fig. 18 shows a schematic diagram of the behaviour of our FFT implementations (SNR_{min} and DR) versus the input noise standard deviation ¾. In this figure, three main regions can be distinguished. When a high value of ¾ is selected (region C), the SNR_{min} remains constant (A_{min} decreases when ¾ decreases) and is independent of ¾ as in the theoretical case. This happens because the quantization noise due to the ADC is not relevant for high values of ¾. Likewise, the DR follows the theoretical behavior: it increases 6 dB per octave. Therefore, the best performance is obtained for decreasing values of ¾.

On the contrary, in region A the quantization noise is appreciable because ¾ is low and both signal and noise may occupy a few quantization levels. Under these circumstances, the dynamic range is approximately constant because a significant reduction of ¾ hardly modifies A_{min}, making the SNR_{min} get worse. As a result, the higher ¾ is, the better the performance obtained.

The performance of some 1024-point FFT designs is depicted in Fig. 19, 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.[8] In order to compare architectures with input bitwidths of 8 and 16 bits, ¾ for the 16-bit architecture is normalized by 2^8. Under these circumstances, $SNR_{O\,min} = 11$ dB according to [24], and, therefore, $SNR_{min}(dB) = -16{:}09$ dB. Following this, the theoretical DR is obtained as

$$DR_T(dB) = SNR_{max} - SNR_{min} = 10 \log \frac{(2^{b-1} - 1)^2}{2¾^2} + 16{:}09. \quad (7)$$

From the analysis of this figure, important conclusions can be drawn. All FFT implementations follow the performance described in Fig. 18, although there are some differences which are necessary to point out. All FFT implementations with the same number of points have the same performance for high values of ¾. Moreover, the minimum reachable sensitivity is equal 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 increases 3 dB when the number of points is doubled. The range of ¾ where the analyzed system performs ideally is wider for the radix-4 architectures than for the radix-2 ones. The reason is the larger number of rotators used in the radix-2 designs, which results in an increase in the threshold to fix P_{fa}, and, therefore, a decrease in the sensitivity. 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 $6 \cdot b$ dB. Thus, a maximum DR of 48 dB could be expected for an 8-bit FFT. However, as shown in Fig. 19, a dynamic range of about 54 dB may be obtained with an 8-bit FFT without saturation. 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 better than the 8-bit implementations without truncation. However, it must be considered that a 16-bit FFT could achieve a DR higher than $6 \cdot b = 96$ dB but, due to the butterfly truncation, it can only obtain a DR of 62 dB and the lowest detectable signal has an amplitude $A_{min} = 25$. Therefore, five of the less significant bits are wasted. As a design rule, it is more interesting to use an FFT with less bits and without truncation than an FFT with a large number of bits and truncation.

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 approximations to the ones in the FFT, there exists no error in the rotations, and the gain of the monobit rotator is always $k = 1$ in (4). However, the approximate rotations imply modifications in the coefficients of the filters obtained from the FFT algorithm, which results in high sidelobe levels [12]; see Fig. 20. The average of the highest sidelobe levels for the filters of the monobit FFT-based filter

[8]Channels with different noise statistics because the rotator error has not been included in the calculations.

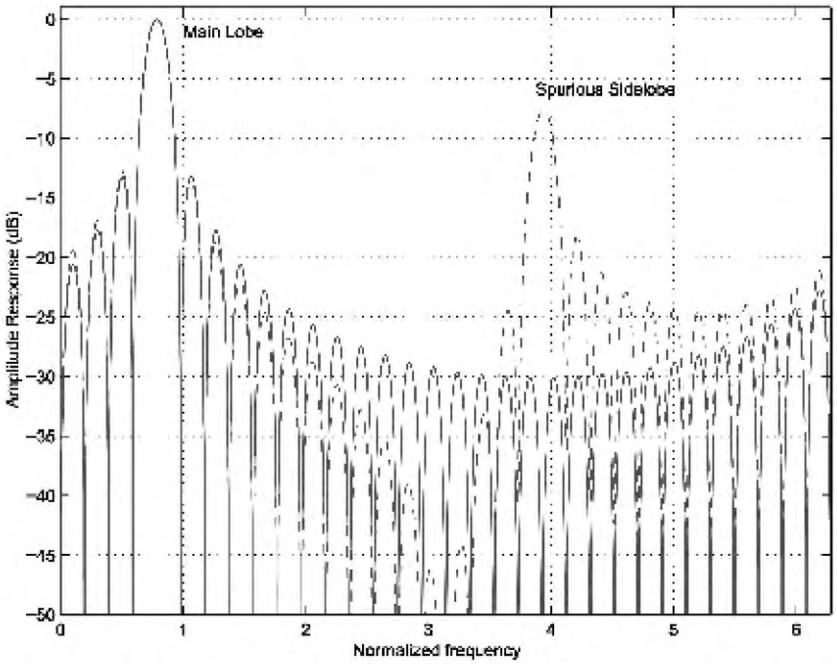

Fig. 20. Filter response for channel 4 from ideal 32-point FFT and from monobit 32-point FFT (dashed line).

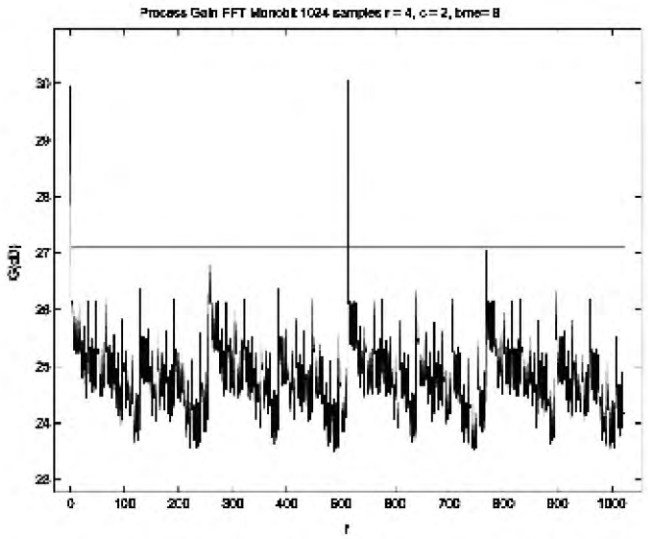

Fig. 21. Processing gain of 1024-point monobit FFT. Radix 4.

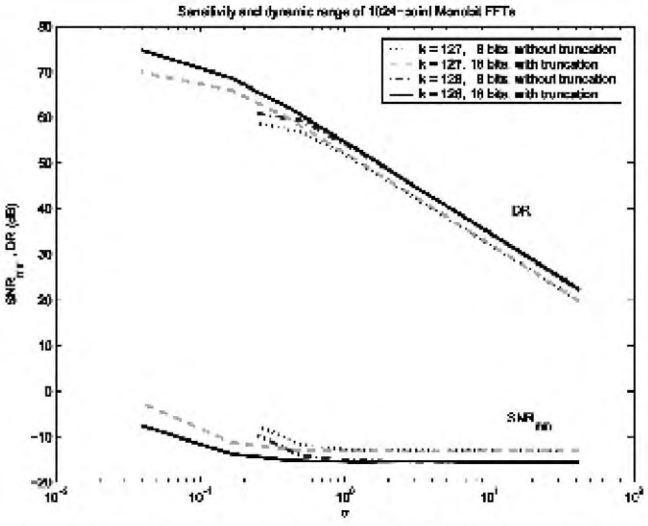

Fig. 22. Sensitivity and dynamic range of 1024-point monobit FFTs. Radix 4.

bank is 9–10 dB below the mainlobe, which cannot be improved by windowing [12]. This represents a serious limitation for the instantaneous dynamic range (the capability to detect simultaneous signals with different powers) of a channelized receiver [4, 12].

On the other hand, the processing gain of the monobit FFT for centered sinusoids depends on the frequency bin and is always lower than that of the conventional FFT. Fig. 21 shows that there is a degradation of 1–3 dB in the processing gain for the case of 1024-point and radix 4 FFT. This degradation calls for an increase in the signal-to-noise ratio at the input of the channelized receiver to maintain the detection probability. This point was discussed in detail in [12].

Fig. 22 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 bitwidth.[9] The same comments for Fig. 19 apply here. However, additional features appear in the monobit implementations. On the one hand, the difference in the sensitivity for high ¾ is due to the nonconstant processing

[9] In order to be able to compare 8 and 16-bit architectures, ¾ for the 16-bit architecture is normalized by 2^8.

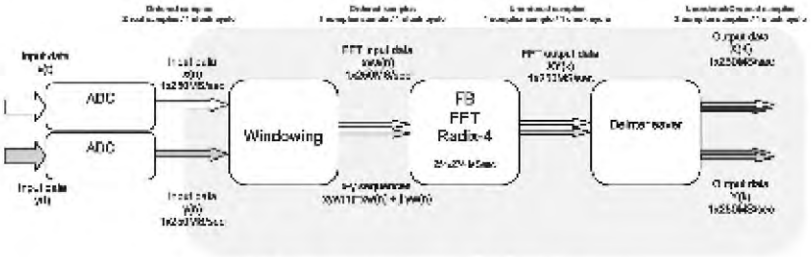

Fig. 23. System diagram with FB architecture.

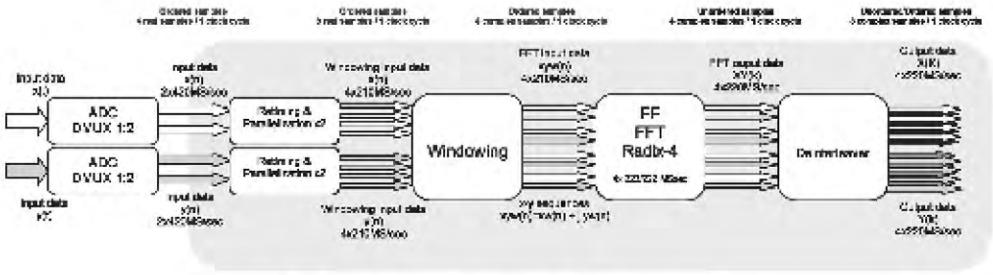

Fig. 24. System diagram with FF architecture.

gains throughout the filter bank generated with the monobit FFT. On the other hand, the improvement in the dynamic range and the lower deterioration in the sensitivity compared with the conventional implementation, Fig. 19, is related to the gain of the monobit rotator, k = 1, in (4), and the elimination of the errors in these rotators.

VI. 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.

Other important issues in FFT-based digital channelized receivers such as windowing, data reuse in overlapped FFTs, coherent and noncoherent processing, detection, and automatic modulation classification of simultaneous broadband signals were deeply analyzed by the authors in [26].

Fig. 23 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, which 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 digitized with a simple ADC (from x(t) to x(n)). However, two analog real signals x(t) and y(t) can be processed in parallel given that our implementations for the FFT are prepared for complex input data sequences. A deinterleaving stage is needed afterwards to split the FFT output and order the complex transforms (X(k), Y(k)).[10]

Fig. 24 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 Fig. 24. 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 digitized 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).

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 standard I/O on FGPAs (420 Mbit/s). However, it is possible to duplicate the I/O data rate to 820 Mbit/s using DDR signaling, which means that data can change on each edge of the clock, or differential signaling. The Virtex-4 family, with a maximum internal clock of 500 MHz, provides IO

[10] xw(n) represents the input sequence x(n) after windowing: xw(n) = x(n)¢w(n). XY(k) means that the FFTs of the input sequences are mixed and a deinterleaving stage is necessary.

data rates that are even higher. This family provides up to 600 MHz for standard IO, and up to 1 Gbit/s 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 [27] 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 Fig. 24. This component allows configurations of 1 : 4 (which results in 500 Mbit/s in each pin, being therefore necessary to use DDR and LVDS in the FPGA) or 1 : 8 (250 Mbit/s in each pin, data rate in the limit of the FPGA standard I/O). Actually, the last commercial version of this ADC integrates the DMUX to simplify the interface with FPGAs. For example the part AT84AS004 [27] is an ADC of 10 bits and 2 GS/s with and integrated DMUX of 1 : 4.

VII. 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 and instantaneous dynamic range.

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

MIGUEL A. SÁNCHEZ
Dept. de Ingeniería Electrónica
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Madrid, 28040
Spain

MARIO GARRIDO
Dept. Señales Sistemas y Radiocomunicación
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Madrid, 28040
Spain

MARISA LÓPEZ-VALLEJO
Dept. de Ingeniería Electrónica
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Madrid, 28040
Spain
E-mail: (Marisa@die.upm.es)

JESÚS GRAJAL
Dept. Señales Sistemas y Radiocomunicación
E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Madrid, 28040
Spain

REFERENCES

[1] Todman, T., Constantinides, G., Mencer, S. W. O., and Cheung, W. L. P.
Reconfigurable computing: Architectures and design methods.
IEE Proceedings, Computers and Digital Techniques, **152**, 3 (2005), 193–207.

[2] Lall, N.
New XtremeDSP slices deliver as much as 10X more GMACs per dollar.
Xcell Journal, (Winter 2004).

[3] Tsui, J.
Microwave Receivers with Electronic Warfare Applications.
New York: Wiley, 1986.

[4] Tsui, J.
Digital Techniques for Wideband Receivers.
Norwood, MA: Artech House, 2001.

[5] Tsui, J., and Stephens, P. J.
Digital microwave receiver technology.
IEEE Transactions on Microwave Theory and Techniques, **50** (Mar. 2002).

[6] Chen, C.-I., George, K., McCormick, W., Tsui, J., Hary, S., and Graves, K.
Design and performance evaluation of a 2.5-GSPS digital receiver.
IEEE Transactions on Instrumentation and Measurement, **54**, 3 (June 2005), 1089–1099.

[7] Zahirniak, D., Sharpin, D., and Fields, T.
A hardware-efficient, multirate, digital channelized receiver architecture.
IEEE Transactions on Aerospace and Electronic Systems, **34**, 1 (Jan. 1998), 137–152.

[8] George, K., Chen, C.-I., Tsui, J., Hary, S., and Graves, K.
Extension of two-signal spurious-free dynamic range of wideband digital receiver using Kaiser window and compensation method.
IEEE Transactions on Microwave Theory and Techniques, **55**, 4 (Apr. 2007), 788–794.

[9] George, K., and Chen, C.-I.
Configurable and expandable FFT processor for wideband communications.
In *Proceedings of IEEE International Instrumentation and Measurement Technology Conference*, May 2007, 1–6.

[10] Parhi, K.
Digital Signal Processing Systems-Design and Implementation.
New York: Wiley, 1999.

[11] Pok, D., Chen, C., Schamus, J., Montgomery, C., and Tsui, J.
Chip design for monobit receiver.
IEEE Transactions on Microwave Theory and Techniques, **45** (Dec. 1997).

[12] Grajal, J., Blázquez, R., López-Risueño, G., Sanz, J. M., Burgos, M., and Asensio, A.
Analysis and characterisation of a monobit receiver for electronic warfare.
IEEE Transactions on Aerospace and Electronic Systems, **39**, 1 (Jan. 2003).

[13] Sanchez, M., Garrido, M., López-Vallejo, M., and López-Barrio, C.
Automated design space exploration of FPGA-based FFT architectures based on area and power estimation.
In *Proceedings of IEEE International Conference on Field Programmable Technology* (FPT 2006), 2006, 127–134.

[14] Yun-Nan, C., and Parhi, K.
An efficient pipelined FFT architecture.
IEEE Transactions on Circuits and Systems II, **50**, 6 (2003), 322–325.

[15] Sansaloni, T., Pérez-Pascual, A., and Valls, J.
Area-efficient FPGA-based FFT processor.
Electronics Letters, **39**, 19 (2003).

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

[17] Cooley, J. W., and Tukey, J. W.
An algorithm for machine calculation of complex Fourier series.
Mathematics of Computation, **19** (1965).

[18] Volder, J. E.
The CORDIC trogonometric computing technique.
IRE Transactions on Electronic Computing, (1959).

[19] Jun, W., Shiyi, M., and Yuezhong, W.
Design and implementation of a high speed vector processor for real-time SAR imaging.
In *Proceedings of CIE International Conference on Radar*, 2001.

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

[21] Nordin, G., Milder, P. A., Hoe, J. C., and Püschel, M.
Automatic generation of customized discrete Fourier transform IPs.
In *Proceedings of Design Automation Conference* (DAC), 2005.

[22] Li, F., Chen, D., He, L., and Cong, J.
Architecture evaluation for power-efficient FPGAs.
In *Proceedings of ACM International Symposium on Field Programmable Gate Arrays*, 2003, 175–184.

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

[24] Skolnik, M. I.
Introduction to Radar Systems (3rd ed.).
New York: McGraw-Hill, 2001.

[25] Harris, F. J.
On the use of windows for harmonic analysis with the discrete Fourier transform.
Proceedings of the IEEE, **66**, 1 (Jan. 1978), 51–83.

[26] López-Risueño, G., Grajal, J., and Sanz-Osorio, A.
Digital channelized receiver based on time-frequency analysis for signal interception.
IEEE Transactions on Aerospace and Electronic Systems, **41**, 3 (July 2005).

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

Wideband DOA Estimation Algorithms for Multiple Moving Sources using Unattended Acoustic Sensors

The problem of direction of arrival (DOA) estimation for multiple wideband sources using unattended passive acoustic sensors is considered. Several existing methods for narrowband DOA estimation are extended to resolve multiple closely spaced sources in presence of interference and wind noise. New wideband Capon beamforming methods are developed that use various algorithms for combining the narrowband power spectra at different frequency bins. A robust wideband Capon method is also studied to account for the inherent uncertainties in the array steering vector. Finally, to improve the resolution within an angular sector of interest and to provide robustness to sensor data loss, the beamspace method is extended and applied to the wideband problems. These methods are tested and benchmarked on two real acoustic signature data sets that contain multiple ground vehicles.

I. INTRODUCTION

The problem of detection, and localization of multiple ground targets using unattended acoustic sensors is complicated due to various factors. These include: variability and nonstationarity of source acoustic signatures, signal attenuation and fading effects as a function of range and Doppler, coherence loss due to environmental conditions and wind effects, near field and nonplane wave effects, and high level of acoustic clutter and interference. In addition, presence of multiple closely spaced targets that move in tight formations, e.g. staggered, abreast or single-file, further complicates the direction of arrival (DOA) estimation, data association, and localization processes. Clearly, optimum performance for detection and localization of multiple acoustic sources is

Manuscript received May 30, 2007; revised February 6, 2008; released for publication April 3, 2008.

IEEE Log No. T-AES/44/4/930741.

Refereeing of this contribution was handled by B. LaScala.

This work was funded by Army SBIR-Phase II Contract DAAE30-03-C-1055. The first data set was provided by the U.S. Army ARDEC, Picatinny Arsenal. The second data set was collected by Information System Technologies, Inc.

0018-9251/08/$25.00 © 2008 IEEE