Automated Design Space Exploration of FPGA-based FFT Architectures based on Area and Power Estimation

M.A. Sánchez [#1], M. Garrido [*2], M. López Vallejo [#3], C. López-Barrio [#4]

[#] *Dpto. de Ingeniería Electrónica*
Universidad Politecnica de Madrid
Ciudad Universitaria s/n
28040 Madrid, Spain
[1] masanchez@die.upm.es
[3] marisa@die.upm.es
[4] barrio@die.upm.es

[*] *Dpto. de Señales, Sistemas y Radicomunicación*
Universidad Politecnica de Madrid
Ciudad Universitaria s/n
28040 Madrid, Spain
[2] mgarrido@gmr.ssr.upm.es

***Abstract*— In this paper we present a tool aimed at generating Fast Fourier Transform (FFT) cores targeting FPGA platforms. The tool is able to generate different pipelined architectures of the FFT that provide different points of the design space: from high performance to low area implementations. The user can select the most suitable architecture based on a broad set of configuration parameters, as they are the number of points, sample size, truncation, etc. Moreover, a set of accurate estimators has been implemented to allow the designer an early and quick design space exploration before synthesizing the core. Experimental results validate our approach and provide significant measurements about the accuracy of the estimation and the tool execution time.**

I. 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, where margins are too tight and the *"time to market"* pressure is tremendous. Moreover, masks costs have reached a critical point, dominating the manufacturing process and requiring high finantial risk.

FPGAs offer significant advantages at a suitable low cost [1]: huge flexibility, reduced verification effort and short design cycle. Even though FPGAs are not so efficient as ASICs they can provide better performance than processor or DSP based systems. This fact, in conjunction with the enormous logic capacity allowed by today's technologies, makes FPGAs an attractive choice for the implementation of complex digital systems. Moreover, with their newly acquired digital signal processing capabilities, FPGAs are now expanding their traditional prototyping roles to help offload computationally intensive digital signal processing functions from the processor.

In this context, system designers demand additional support 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, bandwidth... Low level hardware details should be hidden to the system designer by high level design exploration tools, as proposed in this paper.

In this work we present a tool that allows the designer to perform the design space exploration of a key element of digital signal processing, the Fast Fourier Transform (FFT). With the tool the designer has access to different architectures of the FFT which offer a broad variety of possibilities in terms of area, performance and power. Our purpose is to allow the system designer to explore the different posibilities in a quick and easy way. To address this goal the tool integrates several fast and accurate estimators for key design parameters: area and power. Additionally, the reuse of previously designed cores can be carried out, with the subsequent advantages of reduced price and design cycle.

As base of this work we have chosen pipelined architectures that can be applied to data processing applications with continuous flow of input samples because they allow a high processing speed while keeping a simple and fixed control. We have used xHDL [2] as description language because it provides flexible mechanisms for component customization, instantiation and interconnection. With this, all generated FFTs can be perfectly characterized in terms of performance, area and power consumption when modifying important configuration parameters as they are the radix, the number of points, the

0-7803-9729-0/06/$20.00 © 2006 IEEE

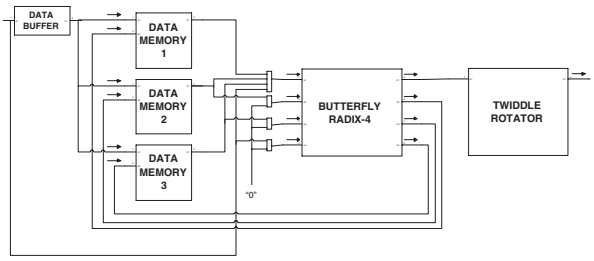

Fig. 1. Structure of a stage of the FB architecture.

bitwidth of input samples or the truncation or not of results through the different stages.

There are other approaches that develop a tool to generate FFTs cores or similar. The Xilinx LogiCore is a well-known example [3], but the degree of parametrization of this tool is significantly reduced when compared to our approach, as will be shown in the experimental results. Other interesting approach is Spiral [4], which is based on a different DFT architecture, the Pease FFT, that provides a parallelized implementation that can be considered in between the architectures presented here. The tool includes several area and performance estimators [5], but the accuracy of the results is far from what we have obtained. Other works on design space exploration approach the estimation problem from high-level especification languages, but they focus on LUT estimation on very small circuits [6] or need to generate RTL models with the corresponding impact on accuracy [7]. Finally, regarding power estimation, the main FPGA manufacturers (Xilinx [8], Altera) have developed power estimators, but they do not allow the evaluation of a core without its physical design (post-map implementation). Moreover, post-PAR may be required by some power estimation procedures [8], [9], which provide more precise results, but paying the price of long execution times.

The structure of the paper is the following. In section II some background on the implementation of the FFT is reviewed. After that, the core generation tool is introduced, Section IV describes in depth the estimation procedures implemented in the tool. Finally experimental results will be presented and some conclusions will be drawn.

II. Background on FFT Architectures

The most widely used algorithm for FFT computation is the one proposed by Cooley-Tukey [10], which is based on the successive decomposition of a DFT with length N into R DFTs with order N/R. R, known as *Radix*, is a power of two 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. There are two approaches to implement the algorithm: decimation in time (DIT) or decimation in frequency (DIF). The only difference between them is the way the algorithm performs the decomposition of the DFT, resulting in a different sequence of operations.

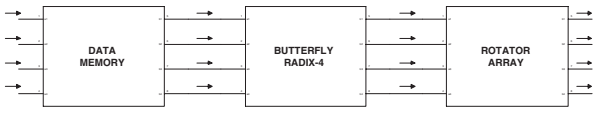

Fig. 2. Structure of a stage of the FF architecture.

Given that most digital signal processing applications require a continuous flow of samples, their implementations will be characterized by high throughput and short word lengths. The FFT implementations that better fit these requirements are parallel and pipelined architectures, where the processing is performed in several cascaded stages. We can classify those architectures into two main groups: with feedback (FB, in figure 1) or feed-forward (FF, in figure 2). There are three basic elements in both FB and FF architectures: a memory where data between stages are stored, a butterfly where low order DFTs are accomplished and finally an element to multiply samples by the corresponding twiddles (rotator). The architectures differ in the way these elements are interconnected and in the control of the sample flow. As result, these architectures provide opposite points in the design space: from the smallest area of the FB structure (resource sharing is fully exploited) to the fastest implementation provided by the FF architecture (with a high price in area).

Both architectures present three levels of pipeline in order to maximize the performance. The first pipeline level is implemented by the set of stages in which the architecture is decomposed. The second level is implemented with the components that form each stage, memory, butterfly and rotator, which are also pipelined (third pipeline level).

Architectures with feedback provide the output flow (samples/second) at the working frequency (MHz), because the FB structure allows the reuse of some of the elements present in every stage. On the other hand, FF structures provide a higher processing speed because reuse is not applied and there are multiple critical elements to avoid delays. Readers interested on more details are referred to [11].

III. FFT Design Space Exploration Tool

To help the system designer to explore the possibilities offered by the different architectures we have designed an FFT generation tool, whose user interface is shown in figure 3. This tool has been implemented in Java using as internal description language xHDL. The tool generates synthesizable VHDL taking as input the configuration options selected by the user. Moreover, the tool provides support to select and configure 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 exploration phase. The user must configure the following input parameters:

- Parameters related to the transform: the transform length (number of points) which is related with the parameters `RADIX` and number of `STAGES`. A special parameter in connection to the transform output is

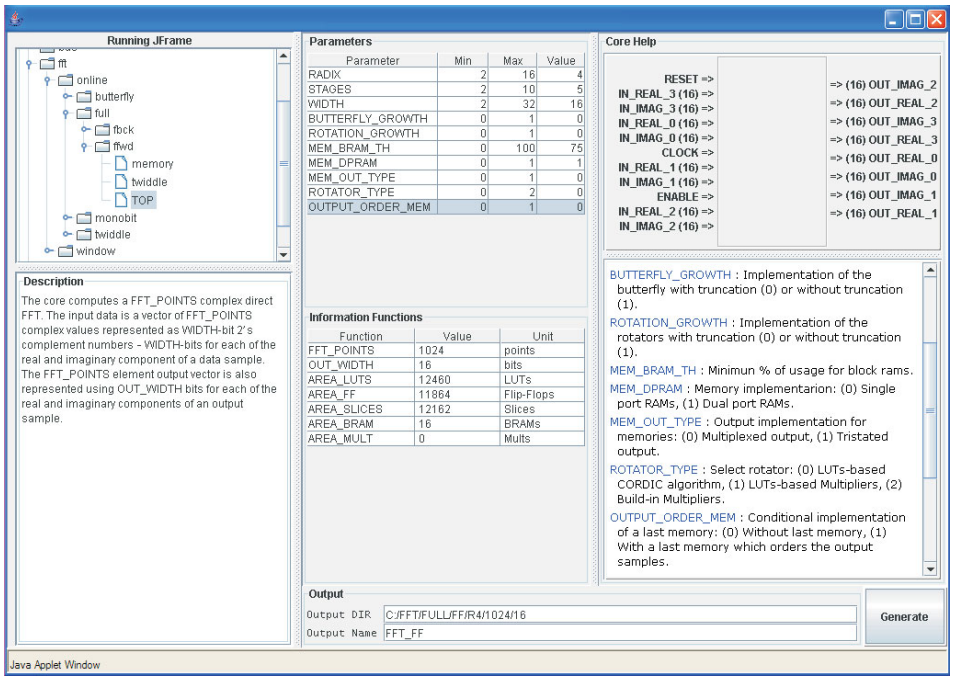

Fig. 3. User interface of the tool. Generation of a Radix 4 FF-FFT core with 16 input bits, 1024 points and no growth.

`OUTPUT_ORDER_MEM`, which sets the use or not 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 design to the physical device. If the FPGA where the design will be mapped includes particular blocks (multipliers, RAM blocks, 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`).

Moreover, the tool provides the limits within which the value of those parameters can range. As a 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 points of the FFT (`FFT_POINTS=Radix`^{Stages}) or the bitwidth of the output results (`OUT_WIDTH`).
- Area estimates: Number of LUTs (`AREA_LUTS`), number of flip-flops (`AREA_FFS`) and area of the FFT in slices (`AREA_SLICES`).
- Estimators of usage of special resources: Number of RAM blocks (`AREA_BRAM`) and number of multipliers (`AREA_MULT`).

At the core of this tool the generation and estimation processes carry out the most important procedures to characterize the FFT cores, thus their working methodology will be described in depth next.

IV. Generation and Estimation Methodology

Traditional hardware description languages are currently limited in their use to build complex systems through parameterization and reuse. For these reasons we have used xHDL, a meta-language designed to improve VHDL that provides flexible mechanisms for component customization, instantiation and interconnection. xHDL was conceived to ease the specification of highly parameterized cores and the reuse of already designed cores, keeping the currently available methodologies and synthesis tools. At the same time, it can help on parameter and component selection through the evaluation of functions that report on estimated characteristics of the design before the long synthesis phase.

The source files for xHDL are known as templates. They contain the code for the available cores as embedded VHDL source code. To allow reuse, xHDL implements a flexible way of communication between subcomponents, a calling mechanism. This helps in parameter fixing (e.g. a component can report to the core on the number of iterations needed for some computation), and also in getting feedback information from a subcomponent (e.g. the latency of a multiplier that determines some buffer latency).

We have built the two FFT architectures (FB and FF) using three common intermediate elements: memory, butterfly and rotators, each one being a complex and reusable core. The way these elements are used in FB and FF architectures is very diverse, resulting in different use of the FPGA resources by their internal library components.

Figure 4 shows the generation and estimation processes implemented in the FFT design exploration tool. The user must

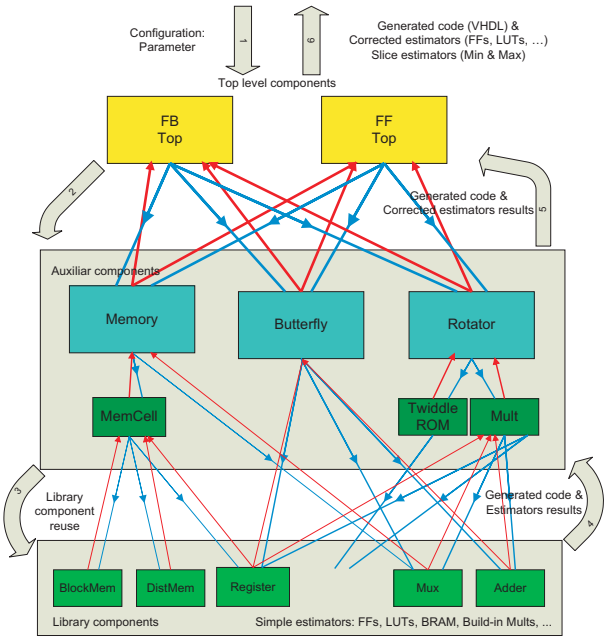

Fig. 4. Flow of information through the generation/estimation process.

first select and configure all input parameters required by the core, and can consult all functions previously defined (which will depend on the input parameters). During the generation process (represented in the figure by the top-down flow, steps 1 to 3), components of higher levels configure and select lower level components in a hierarchical way until the lowest level is reached. Once the lowest level components have been generated, the information they have included in functions and properties can be collected by components in upper levels. This is shown in figure 4 by the bottom-up flow (steps 4 to 6).

More detailed, the design process depicted in figure 4 includes the following steps:

1) Configuration of the desired FFT core parameters by means of the GUI.
2) Parameters of the top level components guide the generation process of the structure inside the core templates. The top templates of FB and FF architectures are based on the reuse of intermediate components (with medium complexity, they are memory, butterfly and rotator), which are selected and configured with the current values given by the parameters.
3) Intermediate components are based on the use of simple, well known and perfectly characterized low level components (register, mux, adder, etc.).
4) Once the low level components are configured, estimators can be accesed. Low level estimation will be described in section IV-A.
5) Simple low level estimates can be aggregated to estimate intermediate level components (it will also be explained in section IV-A).
6) Estimates from lower levels (which can be corrected with constants obtained from experimental measurements) and generated code are obtained.

Next, the estimationa methodology that we have followed is described in detail.

A. Basic Estimation

There are different ways to implement an estimation tool. A first procedure, followed by Spiral [5], would be performed through the measurement of resource use for a given set of examples carried out with different configuration options. From those measures estimation results can be extracted. A key point in this estimation procedure is that it is general-purpose, because there is no need to know the details of the design to estimate. However, the accuracy of this approach is quite reduced, and the results become worse as the complexity and the number of parameters of the design grow.

Another approach to perform a good estimation is based on a perfect knowledge of the design architecture and the way it will be synthesized, taking into account the values of the different parameters involved in its configuration. In this case, the estimates of the basic components of the architecture under study will be required. These low-level estimates can be easy and accurately computed using the previous approach, because of the simplicity of these components. Taking this library of estimates as base, a simple and accurate estimation of the higher level architecture can be accomplished, specially if the specification of the design has taken advantage of a language devised with this purpose, as is the case of xHDL. This is the approach we have followed in this work, and will be deeply explained next.

B. Area estimation

Area estimation is critical for our purposes because it will be used as base for power estimation. With this idea in mind, it is easy to understand that the accuracy of this first estimator will be the key to the success of our design exploration tool.

We have first implemented a library of basic logic and arithmetic components as registers, multiplexers, counters or adders. Given that the target FPGA platform is a Xilinx device, we have selected as area units the LUTs and flip-flops (Ffs) required by the basic components.

Taking advantage of xHDL facilites to specify functions and properties, we have defined the properties `AREA_LUTS` and `AREA_FFS` which offer to the higher level components a particular value computed by a simple estimator. Higher level elements, which will instantiate and configure those components, will take the value of these properties to aggregate to their own estimation. It is therefore possible that these high level components, which have a more global view of the design, could correct area estimation through the modification of the `AREA_XXX` estimates obtained from lower level components.

An additional area unit that can be considered for this kind of FPGAs is the use of special resources, as they are the memory blocks, built-in multipliers of even tri-state buffers. Their estimation is very easy and the accuracy that can obtained usually provides no error.

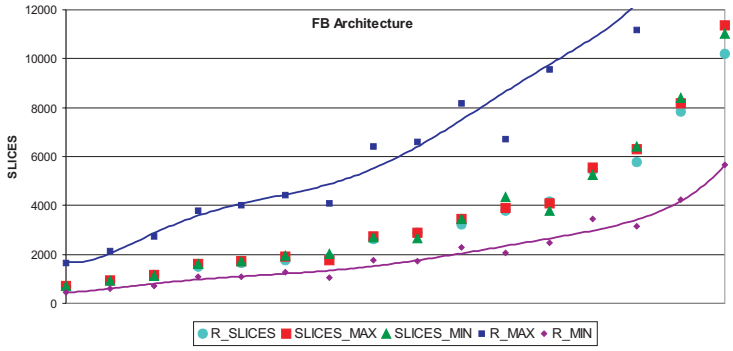

Fig. 5. Procedure to estimate slices, FB architecture.

1) Estimation of the Number of Slices: Even though LUTs and Ffs give a clear idea of the area usage of the FPGA, it is most widely used the figure *slices*. Moreover, the power estimation will be strongly dependent on the accuracy of the estimation of the total number of slices. The estimation of this kind of resources requires a global view of the design, because after synthesis up to two LUTs and Ffs coming from different basic elements may be grouped in a single slice.

To estimate the area in terms of slices we have used information from several implementations to establish the way the synthesis tool packets LUTs and Ffs into slices. For this set of implementations we have observed the post-synthesis results obtained after Xilinx tools. The number of slices is always between two bounds: the lower bound, corresponds to the maximum efficiency of the synthesis tool, when LUTs and Ffs are packed two by two into a slice ($R_{MIN} = Max(R_{FF}, R_{LUT})/2$). R_{FF} and R_{LUT} are real (synthesis) values. The upper bound corresponds to a very unefficient synthesis that assigns every LUT or FF to a single slice ($R_{MAX} = R_{FF} + R_{LUT}$). In figure 5 we can see how those values bound the slice estimation space.

When analyzing these bounds, we can see how the error made is practically the same and it does not depend on the architecture. We have therefore defined a correction constant for both, minimum and maximum bounds: $K_{MAX} = R_{MAX}/R_{Slices}$ and $K_{MIN} = R_{MIN}/R_{Slices}$, where R_{Slices} stands for the exact slice count. With these constants obtained from synthesis results we can implement a good slice estimator during the generation process, taking as input in this case the Ffs and LUTs estimated values (G_{LUT} and G_{FF}):

$$\begin{aligned} SLICES_{MAX} &= (G_{FF} + G_{LUT}) \times K_{MAX} \\ SLICES_{MIN} &= (Max(G_{FF}, G_{LUT})/2) \times K_{MIN}. \end{aligned}$$

2) Improving the accuracy of area estimation: The described estimation provides quite good results in most cases. However, there are some cases where the accuracy of the estimation is not good enough and deserves our attention. For instance, for the particular case when the FF architecture is generated for two stages we could appreciate a bigger deviation in the error obtained. Analyzing in depth this case we appreciated that the number of Ffs is much more larger than the number of LUTs. We have therefore defined a new variable considering the ratio between the number of LUTs and Ffs, FLR, which will define a validity range for the correction constants previously defined. The resulting values (calculated from experimental results) are as follows:

$$K^{FB}_{MAX} = \begin{cases} 0.42 & FLR \leq 0.8 \\ 0.53 & FLR > 0.8 \end{cases} \qquad K^{FB}_{MIN} = \begin{cases} 1.83 & FLR \leq 1 \\ 1.51 & FLR > 1 \end{cases}$$

$$K^{FF}_{MAX} = \begin{cases} 0.63 & FLR < 0.9 \\ 0.49 & 0.9 \leq FLR \leq 1.5 \\ 0.36 & FLR > 1.5 \end{cases} \qquad K^{FF}_{MIN} = \begin{cases} 2.13 & FLR < 1 \\ 1.79 & 1 \leq FLR \leq 1.5 \\ 1.18 & FLR > 1.5 \end{cases}$$

Applying those equations during the generation process we can obtain two estimates of the number of slices for both architectures, within the bounds given by R_{MAX} and R_{MIN} as can be seen in figure 5, and very close to the real value R_{SLICES}.

C. Power Estimation

A good dynamic power estimator requires not only the knowledge related to the particular design (logic used, final routing, etc.), but also information concerning the data used as input. The area estimates (in particular the slice estimation) can be used to characterize the resources used by the design under study, but it is well-known that in FPGAs the final routing has a critical impact on the final power consumption [12], [13]. The problem we face when estimating power is that this kind of information is not available until the P&R has been performed. After P&R we could obtain very accurate power figures, but paying the price of enormous execution time and maybe obtaining results too late in the design cycle.

With these limitations in mind, our main goal when estimating power will be to provide, early in the design cycle, some values that allow the designer to compare among different implementation options or have an idea of the final power consumption required by a core within a final design. The price we can pay is accuracy but, as we will show in section V, this can be assumed by the designer.

To carry out the power estimation we will use again the values obtained for experimetal measures performed with several case studies of our architectures using the Xilinx XPower tool. We fixed in our case studies the activity rates

at 25% (Virtex-II) with working frequencies of 20 and 100 MHz. With the results obtained in these conditions we can compute the average power dissipated per slice, resulting in approximately $5\mu W/slice/MHz$ (5.7 for the FB architecture and 5.5 for the FF). We have used this constant with the estimation of slices required by the design to obtain the power estimation during the core generation.

V. Experimental Results

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

A. Comparative study with other FFT generators

As mentioned in the related work section, there are two other tools that can be used to generate FFT cores: Xilinx Core Generator [3], and Spiral [14]. 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 I, 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). All area and performance results present two values that correspond to worst and best cases for the resource use. These different implementations depend on the efficiency of the resource mapping carried out by the design.

As can be seen in table I 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. Even though our tool can synthesize implementations with these built-in components, we have decided to 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 FPGA 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 two different capacity values (50% and 75%) to find the equivalent measure in slices. With these capacity values we have respectively obtained an area use of 850 and 1250 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 6

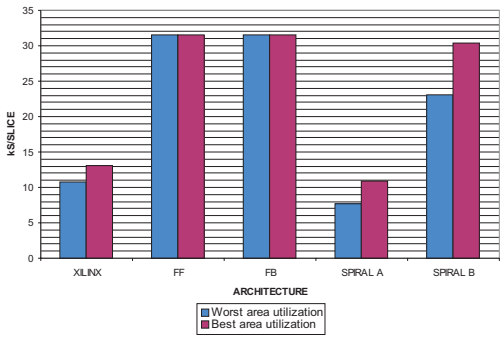

Fig. 6. Comparison of FB and FF architectures with Spiral and Xilinx in terms of Ks/slice.

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 until it reaches the area required by our FF implemetation, but with a performance comparable with our FB implementation. Consequently, the metric shows that this implementation is characterized by a poor ratio between area and performance.

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 area to performance ratio shows for the first architecture a similar behaviour than 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. Validation of the estimation results

We have generated a set of radix-4 FFT architectures to validate the design exploration tool and its estimation procedures. Table II shows the results obtained for both synthesis and estimation in all examples. The example `8_FB_4_0` stands for an 8 input bits FB architecture with 4 stages (256 points) and without growth through the stages. The Ffs and LUTs estimation results for basic components can be seen with more detail in figure 7, where the error in this estimation when compared with synthesis is in most cases below 10%. Ffs errors are mainly due to the removal of unnecessary registers. The LUTs error must be analyzed for both architectures: for the FB architecture error is mainly due to the ROM area estimation when memories are big enough to influence in the global area. In the FF architecture, even though more memories are required, this error is smaller because their size and their impact are smaller.

Regarding the estimation on the number of slices, the accuracy obtained is quite good, around 10%, and always

TABLE I

EXPERIMENTAL RESULTS FOR 16 BIT IMPLEMENTATIONS.

	SLICES	BRAM		MULT18x18		T SLICES	MHz	Mssec	kS/SLICE
XILINX	2744	7	8750	24	8400	19894	214	214	10,757
			5950		7680	16374			13,070
FF	21707	0	0	0	0	27702	213	852	30,756
			0		0	27702			30,756
FB	1	0	0	0	0	7956	251	251	31,549
			0		0	7956			31,549
SPIRAL P4_TH2	1509	64	80000	16	5600	87109	167	668	7,669
			54400		5120	61029			10,946
SPIRAL P4_TH128	3287	16	20000	16	5600	28887	167	668	23,125
			13600		5120	22007			30,354

TABLE II

EXPERIMENTAL RESULTS FOR THE AREA ESTIMATOR.

	Synt			Gen		Err FFs		Err LUTs		Slices		Slices	
Arch	FFs	LUTs	Slices	FFs	LUTs		%		%	Max	Err	Min	Err
8_FB_2_0	910	740	690	924	753	14	1,54%	13	1,76%	702	1,78%	702	1,75%
8_FB_3_0	1460	1269	1177	1480	1282	20	1,37%	13	1,02%	1157	-1,73%	1125	-4,45%
8_FB_4_0	2016	2078	1892	2042	2194	26	1,29%	116	5,58%	1774	-6,24%	2007	6,06%
8_FB_5_0	2578	4117	3782	2610	4776	32	1,24%	659	16,01%	3887	2,77%	4368	15,50%
16_FB_2_1	2537	1906	1752	2543	1983	6	0,24%	77	4,04%	1895	8,18%	1932	10,29%
16_FB_3_1	4553	3619	3237	4565	3713	12	0,26%	94	2,60%	3467	7,09%	3469	7,16%
16_FB_4_1	6878	6126	5421	6896	6356	18	0,26%	230	3,75%	5549	2,37%	5240	-3,34%
16_FB_5_1	9348	11322	10192	9556	12067	208	2,23%	745	6,58%	11379	11,64%	11036	8,28%
8_FF_2_0	2434	1545	1486	2470	1531	36	1,48%	-14	-0,91%	1435	-3,43%	1458	-1,90%
8_FF_3_0	3024	2622	3101	3090	2643	66	2,18%	21	0,80%	2805	-9,54%	2768	-10,75%
8_FF_4_0	4132	4224	4655	4230	4354	98	2,37%	130	3,08%	4200	-9,77%	4627	-0,60%
8_FF_5_0	5137	9375	10741	5378	9211	241	4,69%	-164	-1,75%	9220	-14,16%	9789	-8,86%
16_FF_2_1	7074	4690	4070	7186	4704	112	1,58%	14	0,30%	4265	4,78%	4241	4,20%
16_FF_3_1	11306	9235	9046	11510	9338	204	1,80%	103	1,12%	10201	12,76%	10309	13,96%
16_FF_4_1	17482	15755	14640	17822	16051	340	1,94%	296	1,88%	16574	13,21%	15962	9,03%
16_FF_5_1	23428	30781	29965	25178	30430	1750	7,47%	-351	-1,14%	35142	17,28%	32339	7,92%

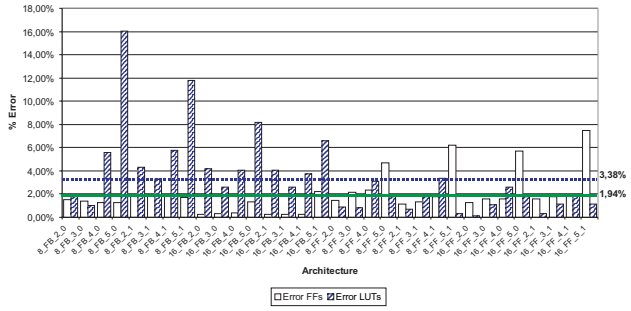

Fig. 7. Accuracy of the Ffs and LUTs estimation.

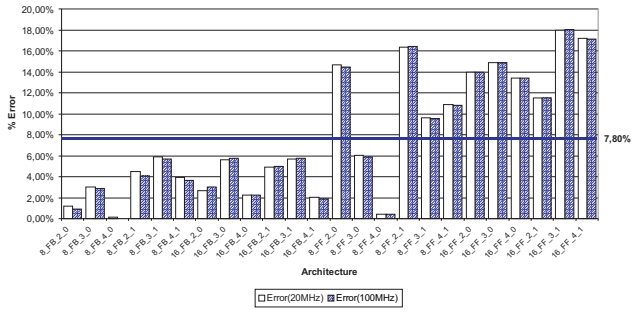

Fig. 8. Accuracy of power estimation.

below 20% of error. Worse results are due to a bad result in the required LUT estimation.

Figure 8 shows the error obtained in the dynamic power estimation for two working frequencies. Given that the error is below 18%, these results allow the comparison among the different implementations in a quick way.

Finally, figure 9 shows the execution time spent in all examples for both estimation (area and power) and synthesis (only area) processes. As can be seen, the time required to estimate both area and power is orders of magnitude shorter than the time required to synthesize a core, even though the accuracy of the estimation is very good. It is specially interesting in the case of power, that after synthesis requires long simulations to obtain any power measure.

These experimental results validate the rapid evaluation of design points that we targeted when designing the FFT generation tool, providing an excellent trade-off between accuracy and execution time.

VI. CONCLUSIONS

We have presented a tool to generate parameterized FFT cores targeting FPGA devices. This tool allows a quick generation of FFT architectures after performing an easy design space exploration based on area and power estimates.

Our approach allows the system designer to select, very early in the design cycle, the best architecture that the system demands. Accurate estimates on power and area will help

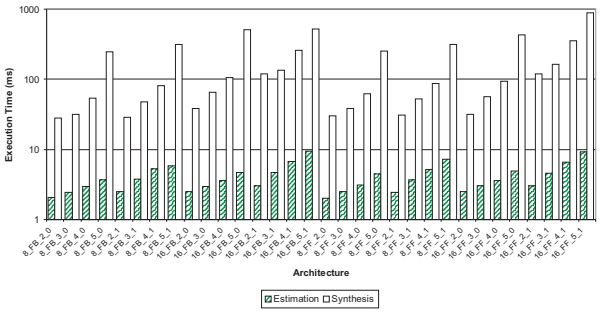

Fig. 9. Execution time: estimation time vs synthesis time.

him/her when making decisions at system level without taking care of implementation details. As future work, the proposed estimation approach will be easily extended to other kind of algorithms.

ACKNOWLEDGMENT

This work has been funded by CICYT project TIC2003-07036.

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

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

[4] P. M. Grace Nordind and, J. C. Hoe, and M. Pueschel, "Automatic Generation of Customized Discrete Fourier Transform IPs," in *Proc. IP Based SoC Design (IP-SOC 2004)*, 2004.

[5] P. A. Milder, M. Ahmad, J. C. Hoe, and M. Pueschel, "Fast and Accurate Resource Estimation of Automatically Generated Custom DFT IP Cores," in *Proc. FPGA2006*, 2006.

[6] D. Kulkarni, W. Najjar, R. Rinker, and F. Kurdahi, "Fast area estimation to support compiler optimizations in FPGA-based reconfigurable systems," in *FCCM'02*, 2002, pp. 239–247.

[7] S. Bilavarn and D. Gogniat, "Design Space Pruning Through Early Estimations of Area / Delay Trade-Offs for FPGA Implementations," *Trans. on CAD*, vol. 99, 2005.

[8] Xilinx Inc., "Xilinx Power Tools: XPower Documentation," http://www.xilinx.com/products/design_tools/xpower.htm.

[9] E. Todorovich, E. Boemo, F. Angarita, and J. Valls, "Statistical power estimation for FPGAs," in *Intl. Conf. on Field-Programmable Logic and Applications*, Aug. 2005, pp. 515–518.

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

[11] 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.

[12] K. Poon, A. Yan, and S. J. E. Wilton, "A flexible power model for FPGAs," in *Intl. Conf. on Field-Programmable Logic and Applications*, 2002, pp. 312–321.

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

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