Design space pruning through early estimations of area / delay trade-offs for FPGA implementations

Sebastien Bilavarn, Guy Gogniat, Jean-Luc Philippe and Lilian Bossuet

Abstract

Early performance feedback and design space exploration of complete FPGA designs are still time consuming tasks. We propose an original methodology based on estimations to reduce the impact on design time. We promote a hierarchical exploration to mitigate the complexity of the exploration process. Therefore this work takes place before any design step, such as compilation or behavioral synthesis, where the specification is still provided as a C program. The goal is to provide early area and delay evaluations of many RTL implementations to prune the design space. Two main steps compose the flow: (1) a structural exploration step defines several RTL implementations, and (2) a physical mapping estimation step computes the mapping characteristics of these onto a given FPGA device. For the structural exploration, a simple yet realistic RTL model reduces the complexity and permits a fast definition of solutions. At this stage, we focus on the computation parallelism and memory bandwidth. Advanced optimizations using for instance loop tiling, scalar replacement or data layout are not considered. For the physical estimations, an analytical approach is used to provide fast and accurate area / delay trade-offs. We do not consider the impact of routing on critical paths or other optimizations. The reduction of the complexity allows the evaluation of key design alternatives, namely target device and parallelism that can also include the effect of resource allocation, bitwidth or clock period. Due to this, a designer can quickly identify a reliable subset of solutions for which further refinement can be applied to enhance the relevance of the final architecture and reach a better use of FPGA resources, i.e. an optimal level of performance. Experiments performed with Xilinx (VirtexE) and Altera (Apex20K) FPGAs for a 2D Discrete Wavelet Transform and a G722 speech coder lead to an average error of 10% for temporal values and 18% for area estimations.

Keywords

Design Space Exploration, area and delay estimation, C specification, H/CDFG representation, graph scheduling, architectural synthesis, technology projection, FPGA device.

I. Introduction

THE continuous increase in the complexity of applications and architectures leads to prohibitively long design cycles. Usually, designers perform hardware exploration through several iterations of a synthesis flow to reach a good constraint compliant solution. As a result, exploring solutions able to match the parallelism potential of the application with the architecture is an uncertain and time consuming process often left to designer experience. To address such an issue, we present an automated exploration approach applicable from system specifications given in the form of a C program. FPGAs are considered for implementation because of their ability to cope with future design perspectives [1][2] and to exploit vast amounts of parallelism within new generations of high performance computing and reactive applications (adaptive video streaming, software radio, . . .)[3][4].

Basically, the approach computes early area and delay values of RTL designs at a glance. So it is important to know what are its limits and the impact on the exploration space we analyze. There is obviously a trade-off between the exploration space coverage and the relevance of the architectural solutions, especially because of the abstraction level gap between a behavioral specification at the input and the FPGA layout characteristics at the output. The key point here has been to reduce complexity enough in order to allow fast exploration of a large space. For that, we defined a two-step process: (1) structural exploration performs automatic definition of several RTL solutions, and (2) physical mapping estimation computes the corresponding FPGA layout characteristics in the form of area / performance trade-offs.

The structural step considers a realistic architectural model including datapath, control and memory units. It focuses on the exploration of the parallelism potential (memory bandwidth and computation) but advanced optimizations such as loop tiling, scalar replacement, data layout, data reusing, memory sharing and memory pipelining are not considered. Also, a simplified execution model for loop unrolling and folding is used in the current state of the tool. Concerning the physical step, the analytical estimation of the mapping results is based on area and delay predictions. Advanced considerations at the logic level such as the impact of routing on critical path (that is on clock period) or others are not analyzed. In other words, we do not take into account all the possible optimization models in the exploration, we restrict its scope from the reliable parameters we can consider at a system level. The motivation is to prune the design space to point out a set of promising solutions for further refinement. We believe such a pragmatic approach permits mitigating the complexity of exploring the design space. Finally, the dependence of the estimations on a target device and low level synthesis tools is thus reported on the physical step, and simplified through the use of libraries. This way, application to several FPGA families (including recent devices) has been made possible. This approach has been integrated in a CAD framework for the codesign of heterogeneous SoCs called *Design Trotter*.

S. Bilavarn is with the Signal Processing Institute School of Engineering, Swiss Federal Institute of Technology, CH 1015 Lausanne Switzerland (email:sebastien.bilavarn@epfl.ch).

G. Gogniat, J.L. Philippe and L. Bossuet are with the Laboratory of Electronic and REal Time Systems (LESTER), University of South Britanny (UBS), Lorient, France (email: guy.gogniat@univ-ubs.fr; jean-luc.philippe@univ-ubs.fr; lilian.bossuet@univ-ubs.fr).

Authors	Estimator Input	Estimator Outputs	Design Space exploration	Architecture Model	FPGA Model	Complexity	Accuracy
Xu and Kurdahi 1996 [10][11]	Netlist	Area, Delay *Tool optimization*	No *Partitioning + analytical*	Datapath, Control logic Operator, Register	XC4000, LUT Interconnections	$O(n)$ $O(n^2 log(n))$ Min-cut	±10%
Enzler et al. 2000 [8]	DFG	Area, Delay	Yes, direct Pipelining Replication Decomposition *Analytical*	Datapath Operator, Register	XC4000E, LUT	$O(n)$	±20%
Nayak et al. 2002 [7]	MATLAB to RTL VHDL	Area, Delay *Tool optimization*	Yes, iterative compilation Loop unrolling, pipelining *Scheduling + analytical*	Datapath, Control logic Operator, Register	XC4010, LUT Interconnections	$O(n^2)$ FDS Left-edge	±15%
Bjureus et al. 2002 [12]	MATLAB to DFG	Area, Delay	Yes, iterative Stream data rate Device clock speed *Trace + analytical*	Datapath Operator, Register	?	$O(n)$	±10%
Kulkarni et al. 2002 [3]	SA-C to DFG	Area *Tool optimization*	Yes, iterative compilation Parallelization Loop unrolling *Analytical*	Datapath Operator, Register	XCV1000, LUT	$O(n)$	±5%
So et al. 2003 [4][15]	C to DFG (loop body)	Area, Delay	Yes, iterative compilation Parallelization Loop unrolling Memory bandwidth *Analytical*	Datapath, Memory bandwidth Operator, Register	XCV1000, LUT	$O(n)$	±20%
Authors This paper	C to HCDFG	Area, Delay	Yes, direct Parallelization Loop Unrolling Memory bandwidth *Scheduling + analytical*	Datapath, Control logic Operator Memory bandwidth	XCV400, EP20K200 LUT, BRAM DSP blocks	$O(n)$	±20%

TABLE I
Most relevant estimator tools dedicated to FPGAs

The remainder of the paper is organized as follows: Section II reviews some prior contributions in the fields of design space exploration and area / delay estimators for FPGAs. Section III focuses on the exploration and estimation flow. This section provides the necessary information to understand the detailed description of the flow reported in sections IV and V. Section IV exposes the structural exploration principles whereas section V presents the physical estimations. Section VI illustrates the approach on several examples to stress the benefits of providing such system level estimations to a designer and the ability to enhance design space exploration compared to classical iterative methods. Section VII presents future work and concludes the paper.

II. Related Work

The Design Space Exploration (DSE) problem related to FPGA implementation is the task of exploring different RTL architectures, where the FPGA architecture is set and different implementation possibilities of an application are analyzed: computation parallelism, pipelining, replication, resource binding, clock value, . . . Such an exploration is motivated by vast amounts of resources available within an FPGA that can speedup the execution of the algorithm until several orders of magnitude. Three types of exploration approaches can be considered: (1) synthesis [5][6]; (2) compilation [3][4][7]; and (3) estimation [8]. The third approach (namely *estimate and compare*) relies on estimations to perform DSE. In that case the synthesis or compilation steps are replaced by low complexity estimators. Once estimations are completed, each RTL architecture is characterized by an area and delay doublet corresponding to the system characteristics when the application is mapped onto the FPGA. As for the second approach, synthesis steps are still required to actually design the final RTL architecture. In the following section, a detailed presentation of some major contributions in the fields of DSE and area / delay estimators for FPGAs is presented. Table I summarizes their main characteristics.

A. DSE and area / delay estimators for FPGAs

A first technique proposed in [9] by Miller and Owyang is based on a library of benchmarks. A set of circuits is implemented and characterized for several FPGAs. Area and delay estimation is performed by partitioning the application into several circuits, that are then substituted by the most similar benchmark. The drawback of this is related to the difficult task of maintaining the library for different devices and applications. Another methodology described in [10][11] by Xu and Kurdahi computes area and delay values from an estimation of the mapping and place & route steps. Starting from a logic level description, they first build a netlist which is then used to compute the actual performance of the system. During the estimation task, wiring effects and logic optimizations are considered to obtain more accurate results. However, this method does not address the DSE problem and is very technological dependent since it is dedicated to the XC4000 family (CLB-based architecture).

The method defined by Enzler et al. [8] performs an estimation from higher abstraction levels (Data Flow Graph, namely DFG). Area and delay are estimated using a combination of both an algorithm characterization (e.g. number of operations, parallelism degree) and an FPGA mapping model (based on operation characteristics in terms of area and delay). The DSE is performed by analyzing the improvements when using pipelining, replication and decomposition of the DFG specification. Their estimator targets a XC4000E device (CLB-based architecture) and uses an analytical approach. Extension to other architectures is not obvious and may need further developments. Their approach is interesting but the limitation to DFG specifications does not allow considering the control and multidimensional data overhead.

The five next methods are based on the compilation of high level specifications and target loop transformations (except [12]). Nayak et al. [7] propose an estimation technique dealing with a MATLAB specification. Their method computes area and delay estimates for a XC4010 device through a two-step approach: first they use the MATCH compiler [13] to perform the DSE (e.g. code parallelization, loop unrolling) and generate a RTL code in VHDL. Then, estimators are used to compute area and delay: area estimation is processed through scheduling and register allocation (to define the number and the type of operators), delay estimation is based on IP characterization and considers the interconnection cost overhead. They also consider some synthesis optimizations (through a multiplicative factor) to obtain more accurate results. Another interesting feature of their approach is to take into account the datapath and control logic in the RTL implementation model. The main limitations are due to the memory unit left unconsidered and control implementation using CLBs, whereas recent FPGAs permit efficient integration of product terms and ROMs using dedicated resources (e.g. Apex Embedded System Blocks [14]).

Bjureus et al. [12] propose a simulation-based approach that computes area and delay from MATLAB specifications. During the simulation of the MATLAB code, a trace is generated to build an acyclic DFG that contains all the operations needed to execute the algorithm. Note that all loops are unfolded to build this DFG. Scheduling and binding are then applied using greedy algorithms. The FPGA architecture is represented through a performance model that is used to compute area and delay estimations. Each resource is modeled with a function that maps an operation to an area and delay tuple. DSE is iteratively processed and considers several design alternatives like the number of input channels, the bitwidth of the input stream, the device clock speed or the device area. Their approach is interesting but limited by the dataflow representation: it is dedicated to applications dealing with scalar variables since they do not consider memory and control units. In [12] the authors do not provide which CLB-based architecture has been used to build their performance model.

Kulkarni et al. [3] propose an iterative compilation-based method starting from an SA-C specification. Their approach expects the compiler to apply extensive transformations to achieve a code that exploits more efficiently the available parallelism. For each transformation, a DFG is generated to derive the impact on area. Their estimator is based on the mapping of the DFG nodes onto the FPGA architecture where each node is represented through an approximation formula. The method also takes into account some synthesis optimizations to enhance accuracy (e.g. shift operations instead of multiplication by a power of 2), but it is restricted to loop bodies and does not consider the memory and control overhead. Moreover like in [8], the estimation of area is restricted to one kind of FPGA resource (Configurable Logic Cells in [8] or number of Look Up Tables in [3]) and does not allow considering dedicated high performance resources like embedded operators and memories.

So et al. [4] also propose a compiler-based approach starting from a C specification. However, compared to [7] and [3] they introduce a key parameter that impacts greatly the DSE problem: memory bandwidth. Their approach is based on the DEFACTO compiler [15] that can successfully identify multiple accesses to the same array location across iterations of multi-dimensional loop nests. This analysis is used to identify opportunities for exploiting parallelism, eliminating unnecessary memory accesses and optimizing the mapping of data to external memories. Thus, their approach deals with advanced optimizations and provides behavioral VHDL codes for each transformation. Area and delay estimation is based on the synthesis results of a HLS tools. The simplified performance model of the FPGA results in an estimation accuracy that depends on the complexity of the loop body.

Finally, Shayee et al. [16] propose a very accurate area and delay modeling approach using analytical and empirical techniques. Their method targets loop nests described as a C specification and evaluates the impact of multiple loop transformations on the datapath and memory interface resources. DSE is iterative and estimations include datapath, memory and control costs.

To summarize previous efforts, the following analysis can be done (Table I). Most studies deal with loop nests and derive a DFG to compute area and delay estimations. Most of them apply iterative DSE. Some studies consider design optimizations to enhance accuracy but they mainly rely on a corrective factor. Datapath estimation is always targeted and memory and / or control units are only considered in a few studies. Finally, most efforts consider a single RTL architecture except [7][8][16] that propose several implementation models (e.g. pipelining, replication). Concerning this point, Choi et al. [17] propose an interesting contribution based on the definition of efficient performance models for specific application domains. Due to this, they are able to analyze a large design space and to take into account the

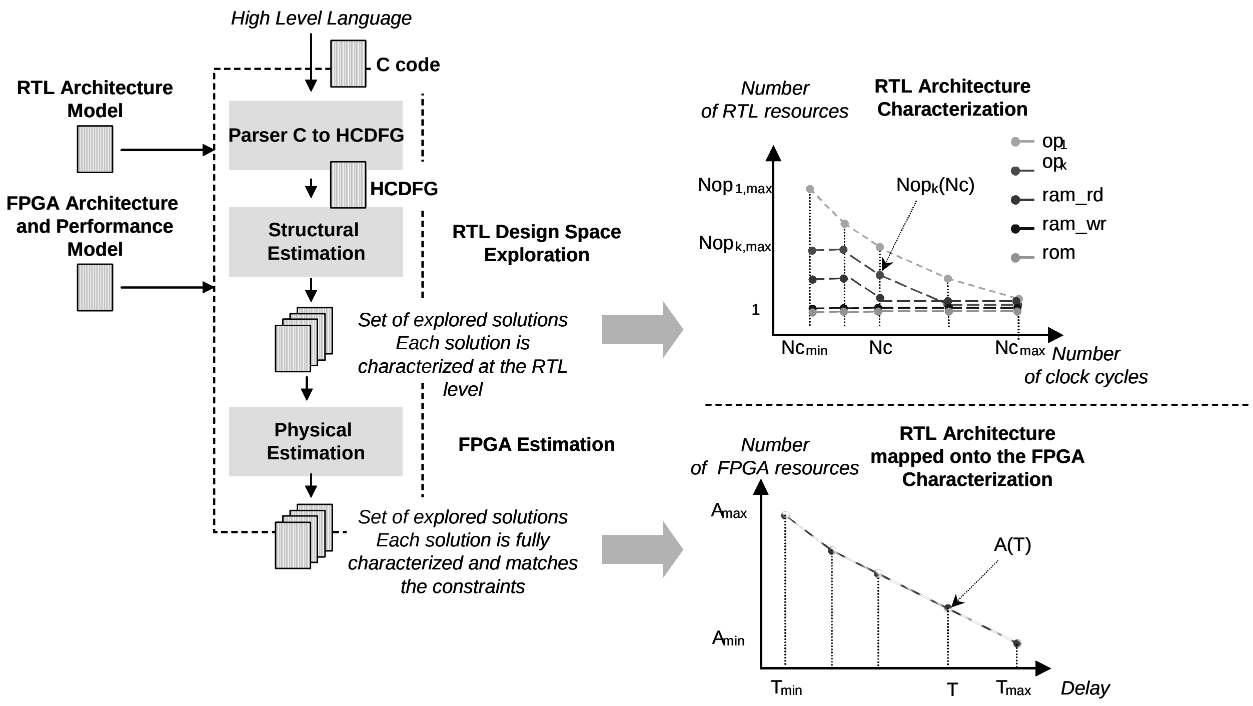
High Level Language
C code
RTL Architecture Model
Parser C to HCDFG
HCDFG
FPGA Architecture and Performance Model
Structural Estimation
RTL Design Space Exploration
Set of explored solutions Each solution is characterized at the RTL level
Physical Estimation
FPGA Estimation
Set of explored solutions Each solution is fully characterized and matches the constraints
Number of RTL resources
RTL Architecture Characterization
op1
opk
ram_rd
ram_wr
rom
Nop1,max
Nopk,max
Nopk(Nc)
1
Ncmin
Nc
Ncmax
Number of clock cycles
Number of FPGA resources
RTL Architecture mapped onto the FPGA Characterization
Amax
A(T)
Amin
Tmin
T
Tmax
Delay

Fig. 1. Exploration / Estimation Flow: a two-step approach composed of 1) the structural definition of several solutions (i.e. at the RTL Level) and 2) the estimation of the physical characteristics of each solution (i.e. FPGA use rate vs. algorithm execution time).

important features of the applications. The main limitation is related to the difficult task of defining the performance model which requires expertise of the application domain.

B. *Discussion*

The work presented in the paper differs from these efforts in several respects. First, we target a more global exploration as we do not restrict our input specification to basic blocks or loop nests. We consider applications including control structures, arrays and computations (due to the use of the H/CDFG as will be described in the paper). We target a single FPGA-based system, that is datapath, memory and control units are all implemented within the FPGA which differs from [4][16] who consider external memories and from other works as they do not consider any memories. We perform an exploration for a RTL architecture model composed of a control unit, a memory unit and a datapath (up to our knowledge, no current work consider such a complete characterization). Our automatic DSE methodology exhibits the memory and computation parallelism potential, this point is original since most efforts only consider the computation parallelism potential and perform iterative DSE (except [4][16]). However, as we aim to prune the design space using low complexity estimators, the RTL architecture does not capture all of the optimizations which can be considered as a limitation. Compared to [3][4][7][16] our RTL architecture model is simpler (but obviously realistic) since we do not consider loop tiling [16], scalar replacement [4][16], data layout [4][7][16], data reusing [4][16], memory sharing [4][16] and memory pipelining [16]. Furthermore, we only deal with a simplified model of loop unrolling and folding. However, the simplicity of our model corresponds to a trade-off to achieve a low complexity exploration: our approach requires only a few minutes to prune the design space for a complete application which enables designers to focus on the highlighted subset of the design space using for example some of the efforts presented above. Concerning the area and delay performance model, we take into account both dedicated resources (embedded memory, DSP block) and LUT. However, compared to [3][7] we do not take into account routing impact on critical path, that is on clock period and logic optimizations. Furthermore like most efforts, we need to maintain a library of operators for each new FPGA which can be considered as a limitation unless it is a first step toward tool and technology independence.

In conclusion, the main contribution of our approach compared to previous efforts consists of dealing with both a complete application, a complete RTL architecture model (even if relatively simple), and a complete FPGA model. We perform an automatic DSE of main design parameters to highlight most promising subsets of the design space which then have to be refined to actually implement the final solution.

III. Exploration & Estimation Flow

In this section we present a general overview of the exploration approach. It is based on two main steps: (1) Structural Exploration and (2) Physical Mapping Estimation [18][19]. The first step performs an exploration at the RTL level that leads to highlight several architectures and characterize them in terms of execution resources and execution cycles. To provide a reliable characterization, Processing, Control and Memory units are considered and modeled through the following parameters:

- For the processing unit (datapath):
  - number and type of each operator;
  - bitwidth of each operator;
  - number of registers;
- For the control unit:
  - number of control states;
  - number of control signals;
- For the memory unit:
  - total memory size;
  - number of simultaneous reads from the RAMs;
  - number of simultaneous writes to the RAMs;
  - number of simultaneous reads from the ROMs;

The goal of this is to provide the main characteristics of several RTL architectures in order to point a relevant subset of the design space; we do not actually build the RTL architectures but instead we define their main characteristics based on the RTL model presented in Section III-B. The upper right corner of Fig. 1 provides the 2D chart used for a convenient exhibition of the results: each solution corresponds to a number of execution cycles N_c on the horizontal axis and the previously described parameters on the vertical axis.

The second step is called "physical mapping estimation"; it computes the expected physical values of area and delay corresponding to the previous RTL solutions mapped onto an FPGA. Device characteristics such as operator / memory area and delay are used to compute accurate estimations of:

- the execution time;
- the number of required FPGA resources, including specific resources like dedicated operators / embedded memories;
- the corresponding FPGA use rate;

FPGA resources considered are logic cells (e.g. slices, logic elements), dedicated cells (e.g. embedded memories, DSP operators), I/O pads and tristate buffers. The results are gathered in a 2D chart illustrated in the lower right corner of Fig. 1. Each solution corresponds to an execution time value (on the horizontal axis) and is characterized by the number of FPGA resources of each type used (on the vertical axis). At the end of the DSE process, several implementation alternatives are thus provided and characterized through area vs. delay trade-offs.

Additional information concerning the inputs of the methodology is introduced now. First, we describe the intermediate representation model (H/CDFG), then the underlying implementation model is detailed.

A. *From C to the H/CDFG model*

The input specification is given in a subset of the C language. The use of a high level software language for the purpose of hardware implementation imposes some restrictions: only a basic subset of the C language is used namely for the specification of control structures (conditional / iterative structures), basic data types, arrays, function calls. Complex constructs like pointers / records / dynamic memory size allocation are not accepted.

The C code is automatically parsed into a Hierarchical Control and Data Flow Graph (H/CDFG) that has been specifically defined for our exploration & estimation purpose [20][21]. This model is composed of three types of elementary (i.e. non hierarchical) nodes: processing, memory and conditional nodes. A processing node represents an arithmetic or logic operation. A memory node is a data access and a conditional node, a test / branch operation (e.g. *if*, *case*, *loops*). A Data Flow Graph (DFG) is a basic block, it contains only elementary memory and processing nodes. Namely it represents a sequence of non conditional instructions in the source code. For example the graph on the right of Fig. 2 is a DFG. A CDFG represents conditional or loop structures with their associated DFGs. The graph in the center of Fig. 2 is a CDFG. Finally, a H/CDFG is a composite graph that can include other H/CDFGs and / or CDFGs. It is used to encapsulate the entire application hierarchy, i.e. the nesting of control structures and graphs executed according to sequential or parallel patterns. The graph on the left of Fig. 2 is a H/CDFG.

A key point is the notion of I/O, local and global data. The parser automatically does a distinction between several types of memory nodes:

- $\eta_{I/O}^{G}$: global I/O which is an I/O data of the entire H/CDFG (whole application)
- $\eta_{I/O}^{L}$: local I/O which is an I/O data of a given subgraph. This data crosses the hierarchical levels of the H/CDFG representation.

- η^L_{data}: local data used to store internal processing results (temporary data within a DFG).
- $\eta_{constant}$: constant data.

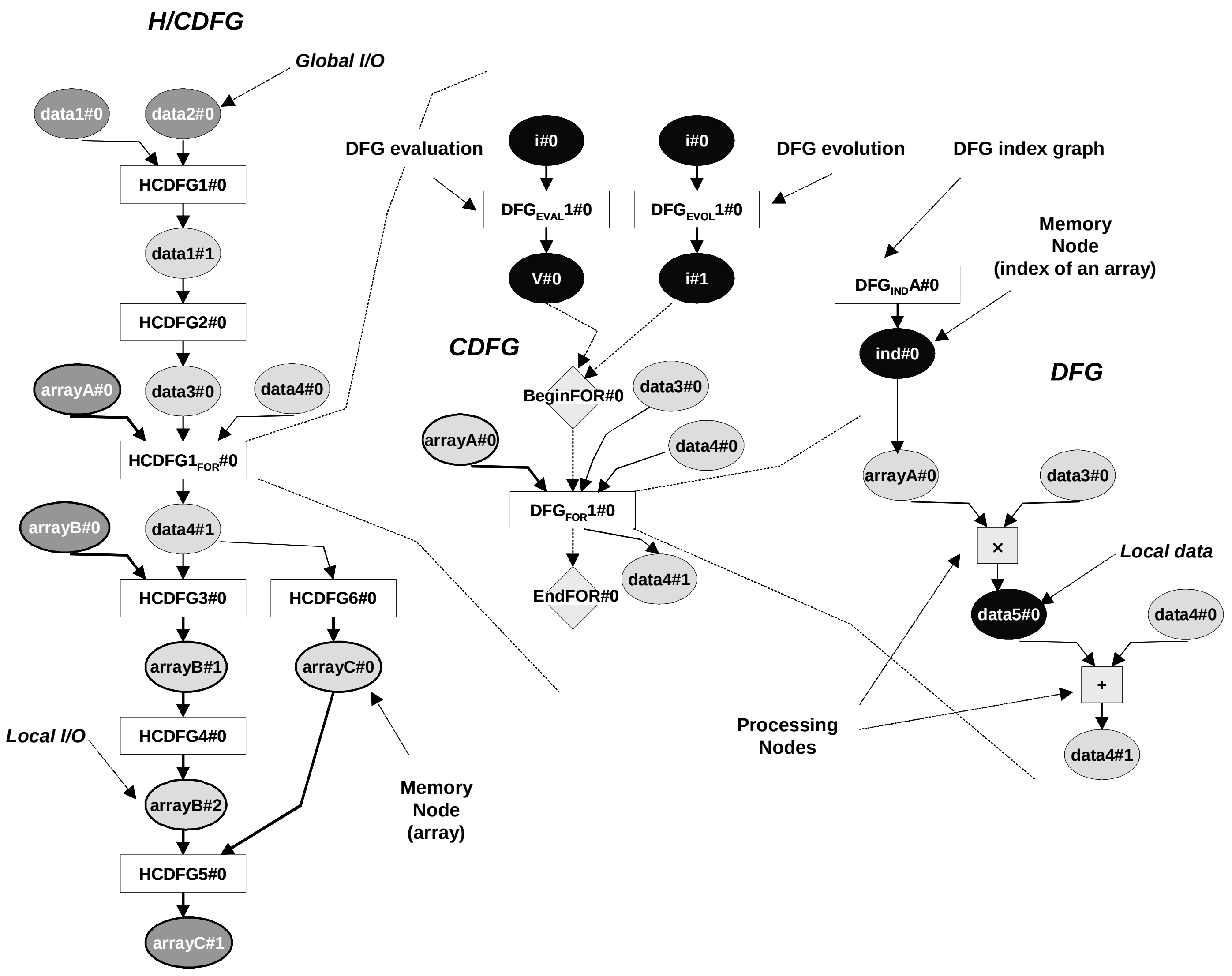

Fig. 2. Elements of a H/CDFG

The creation rules of a H/CDFG from a C code are based on a depth-first search algorithm [22]. A H/CDFG is created each time a conditional instruction (control or loop) is found in the original C code. When no conditional instructions remain in the current hierarchy level, a DFG is built.

B. *RTL Architecture Model*

As pointed out before, Processing, Control and Memory units have been considered to provide more realistic evaluations. Regarding this, some architectural assumptions have been made. We expose those choices that have been mainly defined to cope with FPGA specificities:

- The processing unit is composed of registers, operators and multiplexers. A general bus-based architecture has been preferred compared to a general register-based architecture since this solution minimizes the number of interconnections between resources within the processing unit and promotes parallel execution. We assume that each output of an operator is connected to a register (within the same logic cell). This assumption is directly derived from the fact that a logic cell always includes one or two registers for output synchronization in an FPGA. So doing this does not introduce any area overhead and prevents from applying a register allocation step.
- Concerning the control unit, Moore Finite State Machines (FSMs) are considered. We assume the use of microcode ROMs since recent devices permit efficient ROM integration using dedicated resources, way to keep the logic cells for the processing unit.

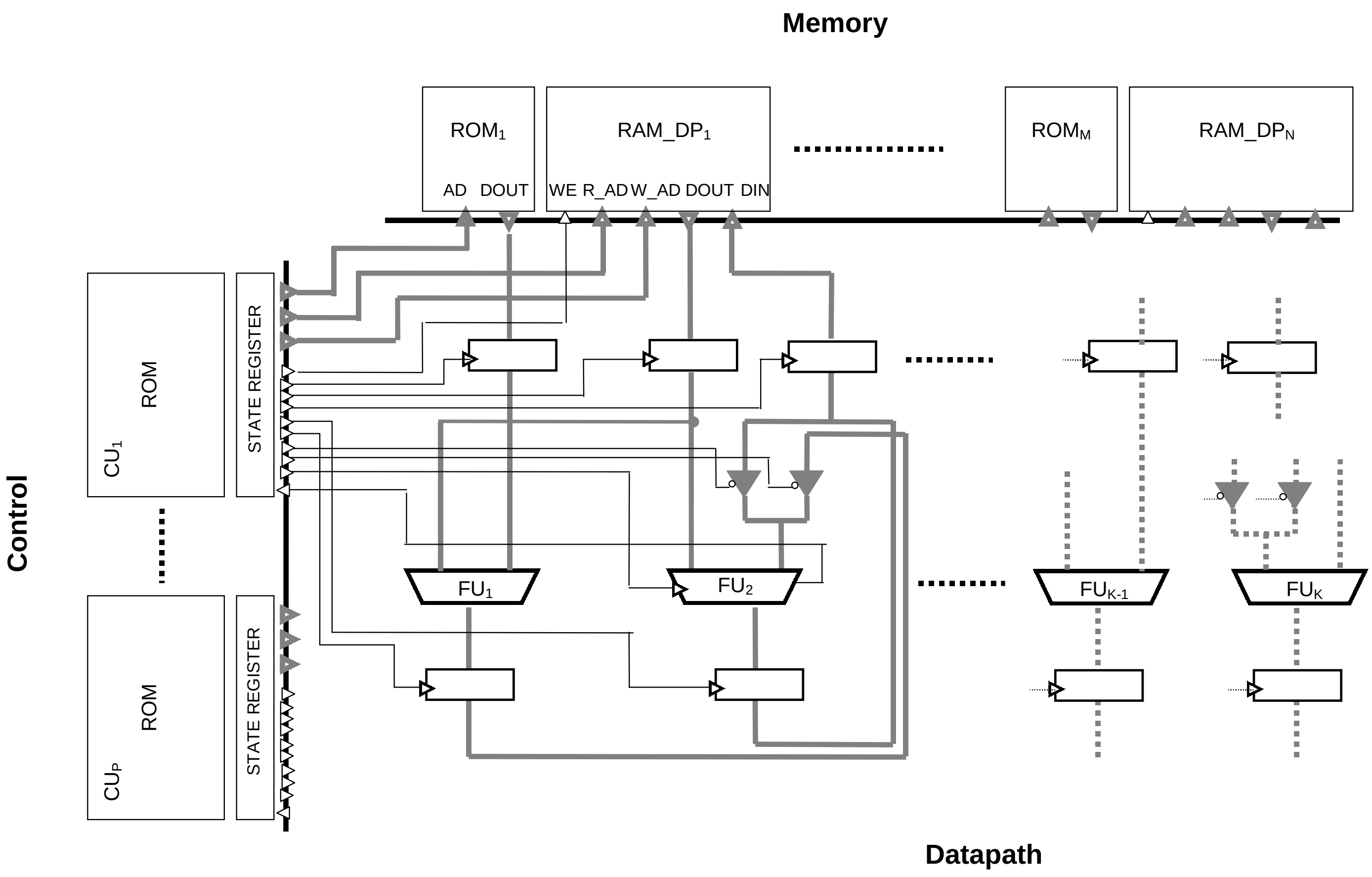

Fig. 3. RTL Architectural Model

- The memory unit is composed of one or several RAM / ROM memories. ROMs are dedicated to constant storage and concerning RAMs, only dual port memories are considered since embedded memories are based on their use.

The choices presented above imply some design permissions / restrictions. However, this underlying implementation model is required to promote the definition of realistic solutions. It also allows reducing the gap between the estimation values of area and execution time (results of the physical mapping estimation step) and the actual design characteristics (results of the final physical synthesis). Other models could have been defined. In the following section, we will focus on the one presented in Fig. 3 and refer to it as our architecture template. Possibility to change / adapt the design template will be addressed in the Result Discussion section (Section VI-D).

IV. Structural Exploration

The structural definition step performs an exploration at the RTL level using the model of Fig. 3. Compared to a typical DSE flow, our approach is not iterative: an automated parallelism exploration is applied. Each RTL solution corresponds to a parallelism degree characterized for a cycle budget N_c by the number and the type of required resources (respectively $Nop_k(N_c)$ and op_k). N_c ranges from a sequential execution (when only one resource of each type is allocated) to the most parallel execution scenario (maximum number of resources is allocated; it corresponds to the critical path).

The definition of several RTL architectures starts with the scheduling of the DFGs (section IV-C). Then analytical heuristics are used to deal with control patterns (section IV-D, CDFG estimation) and application hierarchy (section IV-E.3, H/CDFG combination). Then a progressive bottom-up combination approach allows deriving an efficient schedule of the entire graph at low complexity costs. During the exploration step, both processing resources and memory bandwidth are considered through:

- The number and the type of execution units $N_{op_k}(N_c)$
- The number of simultaneous reads (writes) from (to) the RAMs $N_{ram\_rd}(N_c)$ and $N_{ram\_wr}(N_c)$
- The number of simultaneous reads from the ROMs $N_{rom\_rd}(N_c)$.
- The number of control states $N_s(N_c)$.

Each solution is thus characterized in terms of Processing (number of execution units) Control (number of control steps) and Memory (number of memory accesses). The computation is divided into five steps: (1) Pre-estimation, (2)

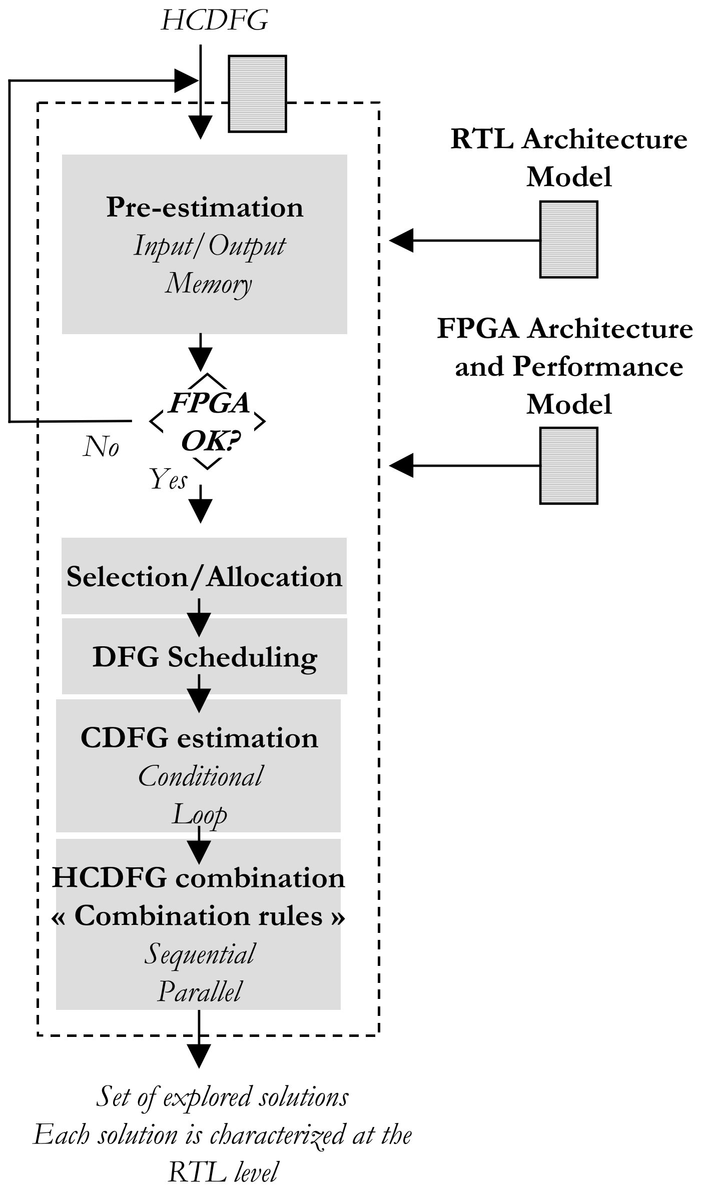

Fig. 4. Structural Estimation Flow

Selection, (3) DFG scheduling, (4) CDFG Estimation and (5) H/CDFG Combination (Fig. 4).

A. Pre-estimation

Pre-estimation checks whether a given FPGA device is suited or not in terms of I/O pads and memory resources for the application. During that step, the set of data nodes labeled $\eta^G_{I/O}$ are compared to the number of I/O pads in the target device. As stated before, $\eta^G_{I/O}$ is automatically derived from the number of formal parameters in the C code. Thus, from the number of data in the set $\eta^G_{I/O}$ and their corresponding bitwidth, the number of I/O pads required is computed and compared to the number of I/O pads available in the device. This information is described in a technology file called the FPGA Technology Architecture and Performance Model (TAPM, section V-A). Multiplexing techniques of I/O data are not considered.

The estimation of the total RAM size is computed from the set $\eta^L_{I/O}$. The technique used is the one proposed by Grun et al. [23] to compute a fast and reliable RAM size estimation from high level specifications. The estimation of the ROM size is derived from the number of elements in the set $\eta_{constant}$. Once computed, RAM and ROM sizes are compared to the amount of memory resources available in the FPGA.

If I/O pads and memory size conditions are satisfied, the next step of the flow is performed otherwise the designer has to select another device for evaluation.

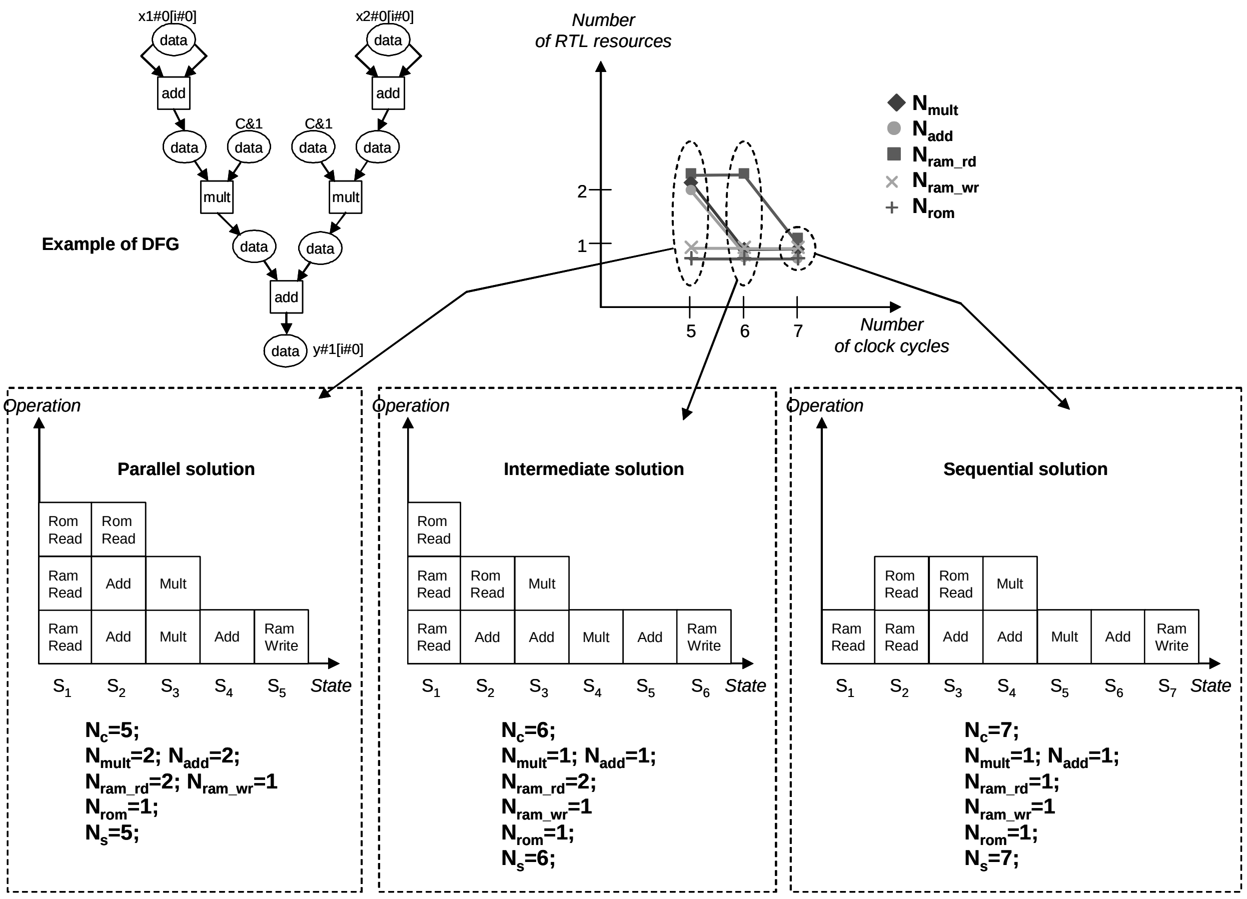

Fig. 5. DFG scheduling example

B. Selection of Execution Units

First, a depth-first search algorithm [22] is used to list all the operations in the H/CDFG. The selection of the execution units is applied using a greedy algorithm. All available execution units are described and classified (speed decreasing order) in the TAPM file. For each operation, the first execution unit supporting the operation in the TAPM file is selected. Although, such an approach may not always give the best solution, it permits a first and rapid evaluation. However, regarding the greedy algorithm limitations, the designer has also the possibility to apply a manual selection in order to choose the most interesting execution resources, on the basis of his design experience. This approach can also be considered to refine the results obtained from a first greedy approach. Then, a clock value can be set. As previously, the designer has the possibility to manually choose a clock value, otherwise it is set to T_{clk_max} which is the propagation delay of the slowest execution unit. An example of clock period exploration and related limitations is given in section VI-C.

C. DFG Scheduling

When the execution units have been selected and a clock value has been set, scheduling is applied to each DFG. A time-constrained *list-scheduling* heuristic extended to deal with both processing and memory nodes [24] is used with the goal to minimize the number of execution units and bandwidth requirements. Several time constraints (N_c) are considered and range from the most parallel solution (fastest execution) to the most sequential one (slowest execution) as illustrated in Fig. 5. Thus, the whole DFG parallelism is explored and each parallelism solution corresponds to a given N_c value (on the horizontal axis of the 2D chart).

In the example of Fig. 5, three solutions representing different number of resources / clock cycles trade-offs are provided. To implement the first solution (most parallel one), the processing unit must include 2 adders and 2 multipliers, and the memory unit must permit 2 simultaneous read accesses (and 1 write access). Note here that only I/Os of the graph and constants are mapped to the memories since internal data are assigned to registers (there is no register allocation as stated in section III-B). Finally, the control unit requires 5 states to manage the scheduling of this DFG.

Once scheduled, a DFG is replaced in the graph by its estimation results (resources vs. clock cycles curve). Then, progressive combination of the curves is applied using a bottom-up approach and leads to the estimation results of the

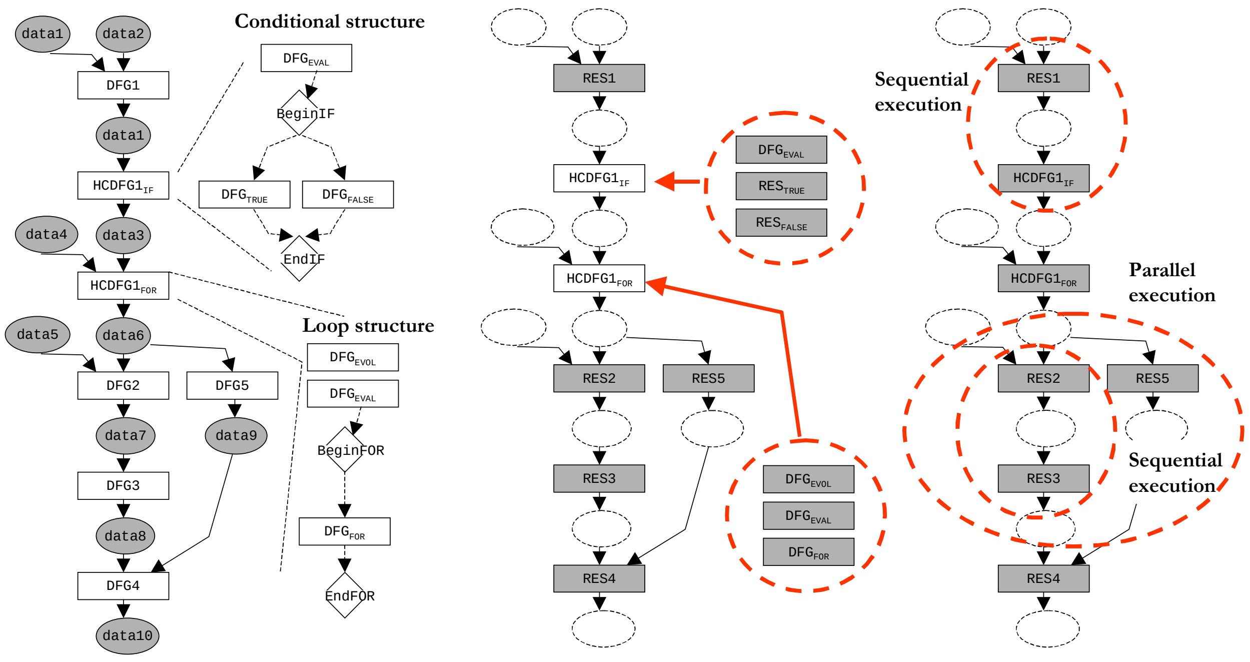

Fig. 6. Combination schemes

entire specification. The combination rules that are used depend on the type of dependence between the parent graphs (Fig. 6). A H/CDFG may be composed of four types of dependences: two of them correspond to control dependences (conditional and loop structures) and the two others correspond to execution dependences (sequential and parallel execution). For each type of dependence, the estimation curves are combined according to the equations presented below.

D. *CDFG estimation*

This section explains how control constructs (CDFGs) are combined. DFG cores within CDFGs are processed as explained previously. Then we have to deal with two types of control: (1) conditional structures (tests) and (2) loop structures (iterations).

D.1 CDFG conditional structures

Conditional structures correspond to CDFGs. They are equivalent to an "if" statement in a programming language. An "if" structure is composed of three subgraphs (DFGs): one for the evaluation of the condition and two for the true and false branches (respectively corresponding to the three subscript 0, 1 and 2 in the following equations). Branching probabilities (P_b) are used to balance the execution time of each branch. Application profiling is used to obtain the execution probabilities.

Each scheduled DFG is characterized by a set of solutions with different trade-offs representing the number of execution units vs. clock cycles. In our approach, we first compute the exhaustive combination of all these solutions and then we remove all non optimal results (i.e. we only keep the Pareto solutions [25]). Thus, the estimation of the solutions (characterized by a cycle budget N'_c) is obtained as follows:

$$N'_c = \lceil [N_{c0} + (P_{b_1} * N_{c1}) + (P_{b_2} * N_{c2}) + 1] \rceil$$
$$N'_s(N'_c) = N_{s0}(N_{c0}) + N_{s1}(N_{c1}) + N_{s2}(N_{c2}) + 1$$
$$N'_{op_k}(N'_c) = MAX[N_{op_{k_0}}(N_{c0}), N_{op_{k_1}}(N_{c1}), N_{op_{k_2}}(N_{c2})]$$

where $N_{s_i}(N_{c_i})$ and $N_{op_{k_i}}(N_{c_i})$ are respectively the number of states and the number of operators / memory accesses in the DFG labeled i for an execution time of N_{c_i} cycles. Execution units and memory accesses are estimated under the assumption of maximum sharing between the three DFGs by taking the maximum value (since they are never executed simultaneously). We assume data are always available in the expected memories. The total number of control states is the sum of the number of states of each DFG, plus one for the branching to the first state of the branch to execute. The equations above are computed for each possible combination of solutions, i.e. each possible combination of N_c (exhaustive approach) and results in a new resources vs. clock cycle curve.

D.2 CDFG loop structures

The most common scheme used to estimate a loop structure is based on a sequential execution. This scenario is considered when the number of iterations is not known statically or in case of data dependences between iterations. In that case, the estimation is computed like this: (1) first, the three subgraphs composing the loop (evaluation, core and evolution) are processed through DFG scheduling. (2) then, a sequential combination of the three subgraphs is applied. (3) and finally, the entire loop structure is estimated by repeating N_{iter} times the loop pattern (with N_{iter} the number of iterations). This last step leads to the equations below.

- Sequential execution:

$$N'_c = N_{iter} * (N_c + 1)$$
$$N'_s(N'_c) = N_s(N_c) + 1$$
$$N'_{op_k}(N'_c) = N_{op_k}(N_c)$$

where N'_c, $N'_s(N'_c)$ and $N'_{op_k}(N'_c)$ correspond to the solution resulting from the combination.

In this case the execution model does not take into account memory and computation pipelining *of loop iterations*, the execution of the iterations is totally sequential. However, we also propose a low complexity technique to unfold loops (partial unrolling and folding execution) in a way to explore more efficiently the available parallelism. Unfolding loops leads to the exploration of the memory and computation parallelism and to reduce the critical path. This technique is based on a simplified execution model since data sharing, data layout and advanced unrolling schemes are not considered. Several parallelism factors f_p are defined to help exploring this intra-loop parallelism.

- Partial unrolling and folding:

$$N'_c = N_c + (N_{iter}/f_p - 1)$$
$$N'_s(N'_c) = N_s(N_c) + (N_{iter}/f_p - 1)$$
$$N'_{op_k}(N'_c) = N_{op_k}(N_c) * f_p$$

where f_p corresponds to the number of parallel executions of the loop kernel (f_p takes a value between 1 and N_{iter}). A f_p value equal to one corresponds to a full pipeline execution of each iteration of the loop kernel. When f_p is set to N_{iter}, it corresponds to a full parallel execution of each iteration of the loop kernel. This estimation of pipelined and parallel executions reduces the total number of execution cycles. Consequently, the number of resources (thus area) increases according to the parallelism factor f_p.

This approach may not always be applicable especially in case of dependences (resource, control and data). But it can be very helpful to exhibit the parallelism level that will allow respecting a given time constraint. Then, one can partially unfold the loop specification according to the parallelism factor and apply a full DFG schedule. This way of proceeding is close to optimality but it is at the expense of complexity. We are developing by now a new heuristic able to consider data dependences based on the analysis of array index variations.

E. H/CDFG exploration

This section explains how we derive the entire estimation of a H/CDFG. When conditional and loop structures have been estimated (section IV-D), the two following combination rules are applied to address sequential and parallel combinations of CDFGs.

E.1 H/CDFG sequential execution

The analytical equations for the combination of two CDFGs executed sequentially (labeled 1 and 2 in the following equations) are based on the assumption of a maximum reuse of the execution units. This means that the execution units are shared for the execution of both graphs:

$$N'_c = N_{c_1} + N_{c_2}$$
$$N'_s(N'_c) = N_{s_1}(N_{c_1}) + N_{s_2}(N_{c_2})$$
$$N'_{op_k}(N'_c) = MAX(N_{op_{k_1}}(N_{c_1}), N_{op_{k_2}}(N_{c_2}))$$

First, an exhaustive combination of the solutions between the two graphs is computed according to the equations above. Then, all non optimal solutions are rejected. The same combination scheme is repeated for each op_k and for the memory accesses.

E.2 H/CDFG parallel execution

The estimation results for a parallel execution of two graphs are computed as follows:

$$N'_c = MAX(N_{c_1}, N_{c_2})$$

$$N'_s(N'_c) = N_{s_1}(N_{c_1}) + N_{s_2}(N_{c_2})$$
$$N'_{op_k}(N'_c) = N_{op_{k_1}}(N_{c_1}) + N_{op_{k_2}}(N_{c_2})$$

The execution time is the maximum of both execution times. Execution units and memory accesses are estimated in the worst case by taking the maximum value. Resource sharing is not considered (unlike sequential execution) and may lead to an overhead in some cases. The reason of this is that such an analysis requires the storage and the analysis of each operation schedule. We believe the impact on the exploration algorithm complexity is too high regarding the improvement of the estimation accuracy.

Given this, the number of states $N'_s(N'_c)$ is the sum of the number of states of each graph since we do not merge FSMs, again for complexity reasons. Instead, we suppose each sub part of the architecture has its own FSM. This results in a hierarchical FSM composed of several microcode ROMs. Like in the case of sequential execution, this combination scheme is repeated for each possible combination of solutions N_c and each op_k, for both execution units and memory accesses.

Alg. 1 H/CDFG exploration algorithm

```
HCDFG_ Expl(graph $current\_G$)
{
$nb\_composite = 0$
for each node in $current\_G$ do
  if current node is a composite node then
    $nb\_composite = nb\_composite + 1$
    if composite node is not a result node then
      $temp\_G \leftarrow$ sub graph of the composite node
      HCDFG_ Expl(graph $temp\_G$)
    end if
  end if
end for

$father\_type$ = type of $current\_G$ father
if $father\_type$ = if then
  Apply $if\_combination$ for $current\_G$
else if $father\_type$ = for then
  Apply $for\_combination$ for $current\_G$
else if $nb\_composite > 1$ then
  Apply $execution\_combination$ for $current\_G$
end if
}


execution_combination(graph $current\_G$)
{
Initialize the list of current node $LCN$ with root nodes in $current\_G$
while $LCN$ non-empty do
  $LCN$ = successors($LCN$)
  for each node $\in LCN$ do
    Parallel combination of predecessor nodes
    Sequential combination of current node and predecessor node
  end for
end while
}
```

E.3 H/CDFG global combination algorithm

The goal of that step is to correctly take into account the control and execution dependences for the entire H/CDFG. At the beginning, all the nodes of the H/CDFG are analyzed with a depth-first search algorithm [22]. When a node is identified as a composite one (namely a hierarchical node containing subgraphs), a recursive function is called until a DFG is reached (Algorithm 1). All the DFGs are scheduled this way and replaced by their corresponding structural characterization. Then, conditional / loop structures are analyzed with respectively the *if_combination* and *for_combination* routines (implementing the heuristics of Sections IV-D.1 and IV-D.2) until there remains zero CDFG unprocessed. The *execution_combination* routine is invoked to process all the possible sequential / parallel execution of subgraphs (heuristics of Sections IV-E.1 and IV-E.2). And finally, the global combination algorithm combines recursively all the nodes until there exists only a single node corresponding to the entire application. These combinations are illustrated on the example of Fig. 6. They lead to the final characterization curve of the whole specification. The last step to complete the Exploration / Estimation flow is to compute the FPGA resources use vs. execution time estimates (Physical Mapping Estimation) for each RTL solution.

V. Physical Mapping Estimation

The physical characterization provides the FPGA resources use $A(T)$ vs. temporal constraint T (physical time unit, ns, μs). It is derived from previous structural characterization of each solution. The resources of the FPGA considered to compute $A(T)$ are logic cells, dedicated cells, tristate buffers and I/O pads. An FPGA characterization file is used to describe a given device (Technology Architecture and Performance Model). It contains the following information:

- The number and the type of the FPGA resources (i.e number of LCs, DCs, multiplexers / tri-state buffers and I/O resources);
- The area and delay of operators;
- The area and access times of memories (dual port RAMs and ROMs);

This information is obtained from the data sheet of the target device and from the synthesis of basic arithmetic / logic operators and memories.

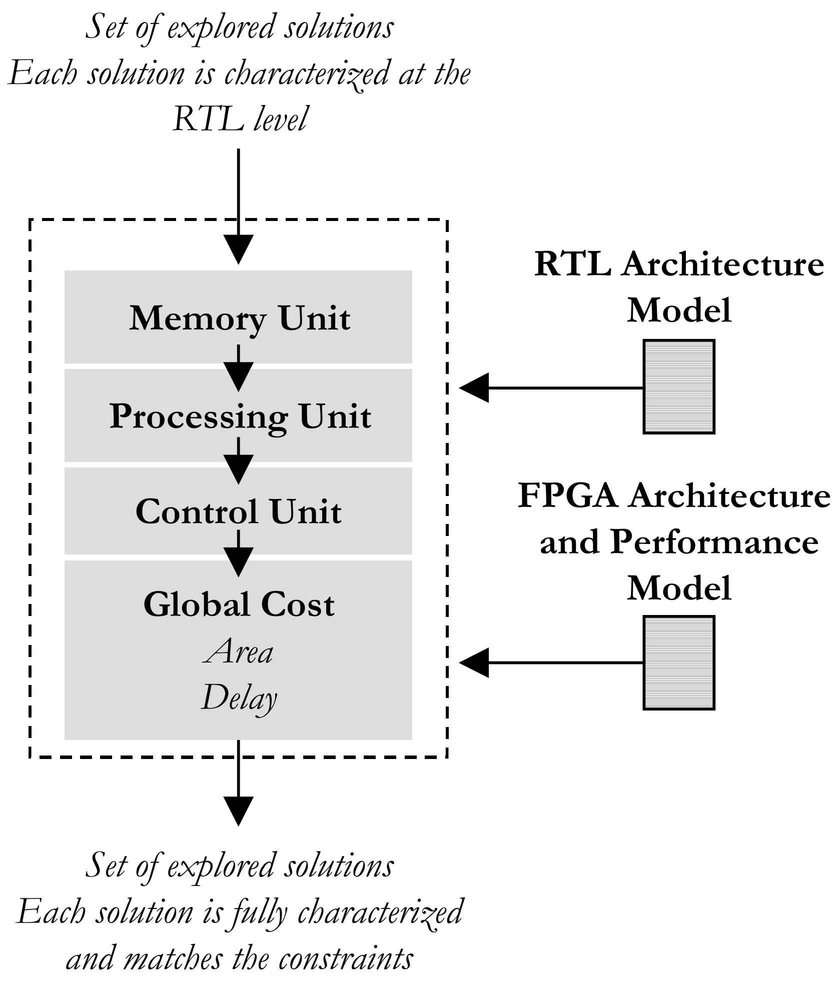
Set of explored solutions
Each solution is characterized at the RTL level
Memory Unit
Processing Unit
Control Unit
Global Cost
Area
Delay
RTL Architecture Model
FPGA Architecture and Performance Model
Set of explored solutions
Each solution is fully characterized and matches the constraints

Fig. 7. Physical Estimation Flow

A. FPGA Technology Architecture and Performance Model

Table II gives a simplified example of characterization for a Virtex V400EPQ240-7 device [26]. This FPGA contains 4800 Logic Cells and 40 Dedicated Cells (Slices and BRAM respectively). Each operator is characterized for usual bitwidths (8, 16 and 32 bits) by the number of FPGA resources used, by the corresponding delay and the number of control signals. To give an example, an eight bits adder is characterized as follows: 4 slices, 4.9ns, 8 bits, 1 control line (corresponding to the signal to drive the output register of the adder). Those values are obtained after a logic synthesis step. Memories are characterized by the type of FPGA resources needed for implementation, the storage capacity and read / write delays. A generic characterization (i.e. independent from size and bitwidth) has been defined on the basis of the number of memory bits per cell and the worst case access time. Furthermore, two choices are left to implement

Virtex V400EPQ240-7						
FPGA characteristics	LC	# LC	DC	# DC	# Tri	# I/O
	slice	4800	BRAM	40	4800	48
Execution units	op	impl	latency	area	bitwidth	ctl lines
adder	+	LC	4.9ns	4	8	1
sub	-	LC	4.9ns	4	8	1
mult	×	LC	12.3ns	36	8	1
comp	<,>,>=,<=	LC	4.8ns	6	8	1
equal	=	LC	5.0ns	4	8	1
shift reg	<<,>>	LC	4.9ns	4	8	1
reg		LC		4	8	1
mux		Tri		4	8	2
Memory	impl	access time	bit per cell	latency read	latency write	
RAM DP	LC	rw	32	13.4ns	13.4ns	
RAM DP	DC	rw	4096	7.2ns	7.2ns	
ROM	DC	r	4096	7.2ns		

TABLE II
Virtex V400EPQ240-7 FPGA characterization

a memory in both Xilinx and Altera devices. For example in an Apex EP20K200RC208-1, logic cells (*logic elements*) or dedicated cells (*ESBs*) are available for implementation. The respective characterizations are 16 bits / logic element, 11.2ns and 2048 bits / ESB, 4ns. As we can notice, dedicated cell implementation is faster, but it is also more area efficient. In our approach, dedicated cell implementation is always a preferable solution since it allows saving the logic cells for custom user defined functions.

B. *Technology projection*

From the characteristics of a given RTL solution and a target FPGA, we derive a simple and accurate estimation of the expected FPGA use rate and algorithm execution time. For better accuracy, a specific "projection" process has been defined for each unit of the architecture.

B.1 Memory unit

A simple approach is applied to evaluate the area of the memory unit. It is based on the total memory size, the number of simultaneous accesses, and the characteristics of the memory resources within the FPGA. As stated before, two types of implementation are considered: logic cells or dedicated cells. According to the type of cells used, the area of the memory unit is estimated as follows:

- Logic cell implementation:

$$N_{lc}^{ram} = \lceil (MS_{RAM} * W_{ram}/N_{bits/lc}^{ram}) \rceil$$
$$N_{lc}^{rom} = \lceil (MS_{ROM} * W_{rom}/N_{bits/lc}^{rom}) \rceil$$

where MS_{RAM} and MS_{ROM} are respectively the total memory size for the RAM and the ROM memories. $N_{bits/lc}^{ram}$ and $N_{bits/lc}^{rom}$ are the number of memory bits per logic cell and W_{ram}, W_{rom}, the bitwidth of the data to be stored.

- Dedicated cell implementation:

In a first approach, the number of memories can be approximated to the maximum of the number of simultaneous read and write operations $N_{mem}^{ram} = MAX[N_{ram\_rd}, N_{ram\_wr}]$. However, the type of resource used is important because only one memory can be integrated in a single dedicated cell. So in this case, the number of embedded memories is computed by taking the maximum value between the number of dedicated cells needed to implement the total memory size and the number of simultaneous accesses:

$$N_{dc}^{ram} = MAX[\lceil (MS_{RAM} * W_{ram}/N_{bits/dc}^{ram}) \rceil, N_{ram\_rd}, N_{ram\_wr}]$$
$$N_{dc}^{rom} = MAX[\lceil (MS_{ROM} * W_{rom}/N_{bits/dc}^{rom}) \rceil, N_{rom}]$$

$N_{bits/dc}^{ram}$ and $N_{bits/dc}^{rom}$ correspond to the number of memory bits per dedicated cell and W_{ram}, W_{rom}, the bitwidth of the data to be stored.

Once the number of memories of each type is known, the number of control signals to drive the RAM (N_{cs}^{ram}) and ROM memories (N_{cs}^{rom}) are derived. They correspond to the address and *write enable* signals (Fig. 3):

$$N_{cs}^{ram} = (2 * W_{adr}^{ram} + 1) * MAX(N_{ram\_rd}, N_{ram\_wr})$$

$$N_{cs}^{rom} = W_{adr}^{rom} * N_{rom}$$

where the sizes of the address bus (W_{adr}^{ram} and W_{adr}^{rom}) are derived from the number of words in the memories:

$$W_{adr}^{ram} = \lceil log_2(MS_{RAM}/MAX(N_{ram\_rd}, N_{ram\_wr}))\rceil$$
$$W_{adr}^{rom} = \lceil log_2(MS_{ROM}/N_{rom})\rceil$$

The current RTL model does not include a specific address generator. Hence, the control of the address signals of the RAMs and ROMs is left to the control unit. Thus, the expressions of N_{cs}^{ram} and N_{cs}^{rom} take respectively into account W_{adr}^{ram} and W_{adr}^{rom}. The total number of control signals for the memory unit is:

$$N_{cs}^{mu} = N_{cs}^{ram} + N_{cs}^{rom}$$

B.2 Processing unit (datapath)

The area of the processing unit is computed by adding the contribution of each execution unit ($N_{lc}^{op_k}$ and $N_{dc}^{op_k}$):

$$N_{lc}^{pu} = \sum_{op_k} N_{op_k} * N_{lc}^{op_k}$$
$$N_{dc}^{pu} = \sum_{op_k} N_{op_k} * N_{dc}^{op_k}$$

For example, three eight bits adders require 12 slices in a Virtex V400E ($N_{lc}^{add_{8bit}} = 4\ slices$). Like in the case of the memory unit, the number of control signals required to drive the datapath is considered. There are four types of control signals, according to the current model of Fig. 3:

- signals to control the output register of each execution unit. The number of signals of this type is equal to the number of execution units N_{cs}^{op}.
- signals to select the operation for multi-functional units $N_{cs}^{multi\_op}$.
- signals to control the registers of the memories read / write ports $N_{cs}^{reg} = N_{reg}^{ram} + N_{reg}^{rom}$.
- signals to control multiplexors / tristates N_{cs}^{mux}.

The number of control signals for the processing unit is then

$$N_{cs}^{pu} = N_{cs}^{op} + N_{cs}^{multi\_op} + N_{cs}^{reg} + N_{cs}^{mux}$$

and the total number of control signals for the entire architecture is

$$N_{cs} = N_{cs}^{pu} + N_{cs}^{mu}$$

B.3 Control unit

The area of the control unit is derived from the number of states and the number of control lines [27]. Control logic is supposed to be integrated in a ROM memory. The area is computed from its number of words and data bitwidth.

$$N_{bits\_state\_reg} = \lceil log_2(N_s)\rceil$$
$$N_{bits\_rom} = N_{bits\_state\_reg} + N_{cs}$$

where N_s is the total number of states needed to schedule the entire graph. The area of the control logic is obtained from the area used by a $N_s * N_{bits\_rom}$ ROM. Then, the corresponding FPGA resources for a logic cell or a dedicated cell implementation are respectively:

$$N_{lc}^{rom} = \lceil(N_s * N_{bits\_rom}/N_{bits/lc}^{rom})\rceil$$
$$N_{dc}^{rom} = \lceil(N_s * N_{bits\_rom}/N_{bits/dc}^{rom})\rceil$$

B.4 Global cost characterization

The total area in term of FPGA use is computed for each type of FPGA resource by adding the contribution of each unit of the architecture.

$$N_{lc} = N_{lc}^{mu} + N_{lc}^{pu} + N_{lc}^{cu}$$
$$N_{dc} = N_{dc}^{mu} + N_{dc}^{pu} + N_{dc}^{cu}$$
$$N_{tristate} = N_{tristate}^{pu}$$

The physical value of the execution time is computed from the value of the clock period defined during the selection step (section IV-B):

$$T = N_c * T_h$$

The above computation process is then iterated for each architectural solution resulting from the structural explorations and leads to the final cost vs. performance characterization (figure 1).

VI. Experiments and Results

This section is organized as follows: first, we apply the exploration methodology on several examples representative of different algorithmic complexities and processing characteristics (intense data processing, control dominated, high / low parallelism potential). Then, we address the problem of estimation accuracy. For this purpose, we compare the synthesis results of one architectural solution with the area and delay estimates given by our exploration tool. Three approaches have been considered for comparison: the first one is based on hand coded design, the second one on HLS design and the last one is based on pre-characterized IPs. Then, we apply the methodology on a 1D Discrete Wavelet Transform to illustrate how easy and independent from any design experience background the exploration process can be performed. Finally, a discussion is provided in order to analyze the drawbacks / benefits of the approach and stress the differences compared to existing works.

	Virtex V400EPQ240		Apex EP20K200EFC484	
	sol.	expl. time	sol.	expl. time
FIR	5	0.04 sec	5	0.03 sec
Volterra	11	5.6 sec	11	4.2 sec
F22	40	2.3 sec	40	2.6 sec
Adaptive	4	2.6 sec	4	1.8 sec
FFT	56	0.4 sec	56	0.4 sec
DWT	342	8.7 min	342	9.4 min
DCT	14	1.5 sec	14	1.6 sec
G722	16	0.7 sec	16	0.4 sec
MPEG	2	3.1 sec	2	3.1 sec
Huffman	4	4.2 sec	4	4.5 sec

TABLE III
Number of solutions generated vs. exploration times for several DSP applications

A. *Applications*

The methodology presented in this paper has been integrated in a framework for the codesign of SoCs called *Design Trotter* [21]. With the help of this tool, early exploration and design space pruning of applications from C specifications are fast and easy. Applications representative of several algorithmic complexities and processing characteristics have been used to analyze the design space coverage vs. exploration time trade-offs. The first seven examples in table III tackle typical DSP processing: filtering (FIR, Volterra, F22 and Adaptive), transforms (Fast Fourier Transform, 2D Discrete Wavelet Transform and Discrete Cosine Transform). These examples exhibit high amounts of memory and computation parallelism with intense data processing / storage, especially in the case of the 2D DWT. The last examples (G722 speech coding recommendation, MPEG and Huffman coding) are more control dominated systems with low parallelism potential.

Table III presents the number of solutions explored and the related exploration time obtained with a Pentium 3 processor running at 1.2 GHz. The results show the ability of the tool to define several architectural solutions in a reasonable amount of time, even in the case of complex specifications. In the 2D DWT for instance *Design Trotter* generates about 350 RTL solutions and the corresponding area / delay estimates within 10 minutes. For comparison,

	Virtex V400EPQ240-7				Apex EPK20K200EFC484-2X			
	Estimation		Synthesis		Estimation		Synthesis	
	slices	T_{ex} (ns)	slices	T_{ex} (ns)	lgc elt	T_{ex} (ns)	lgc elt	T_{ex} (ns)
Parrec	9	6	10	6	17	9	19	9
Recons	9	6	10	6	17	9	19	9
Upzero	217	1224	255	1171	608	1504	400	1272
Uppol2	257	292	303	300	857	358	718	302
Uppol1	216	230	275	254	589	282	612	236
Filtez	150	77	163	88	518	94	648	103
Filtep	181	593	177	511	484	728	497	515
Predic	9	6	10	6	17	9	49	9
G722Predictor	1166	1224	1263	1317	3132	1504	3027	1318

TABLE IV
Estimation vs. Synthesis for the G722 Predictor function

the time required to perform only the *logic synthesis* of *one single* solution is several orders of magnitude higher (about one day).

The results of table III show the effectiveness of a global "low complexity" estimation framework operating from early specifications and exploring several RTL solutions characterized in terms of processing, control and memory. In the following, we address the problem of estimation accuracy in order to evaluate the relevance of the estimates provided.

B. *Accuracy*

To make a valuable evaluation of accuracy, three approaches have been carried out: compare estimations with hand coded designs, compare estimations with HLS design and compare estimations with pre-characterized IPs. The first approach is needed because the accuracy of the estimations is strongly related to the RTL model of Fig. 3. Thus, it highlights the accuracy of the physical mapping estimation which is important since it provides the designers with reference values allowing reliable comparison of area / performance trade-offs. The second and the third approaches can lead to more significant variations because the RTL models used are different. But on the other hand, a comparison with an automated architectural synthesis tool and with IP designs is required to discuss the relevance of the architectural solutions provided.

B.1 Comparison with hand coded designs

The two applications considered are a speech coder (G722) [28] and a 2D Discrete Wavelet Transform [29]. Concerning the G722 recommendation, we focused on the predictor which is the processing core of the application. The predictor is composed of eight sub-functions that are executed concurrently and represents an average of 260 lines of C codes. The DWT algorithm considered is based on a lifting scheme process composed of twelve filtering functions applied sequentially. Six loops are first executed to compute the horizontal transform and then six loops are executed to compute the vertical transform. Each loop is a second order nested loop. The complexity of the algorithm is about 250 lines of C codes that corresponds to almost 500 lines of H/CDGH grammar.

In the following, we compare exploration times vs. logic synthesis times (it does not include the time spent for architecture synthesis), for two representative devices of recent FPGA families: Virtex V400EPQ240-7 [26] and Altera Apex EP20K200EFC484-2X [14]. The ISE Foundation and Quartus synthesis tools have been used to target the respective device.

In order to provide a significant evaluation, the entire applications have been synthesized in both cases (G722 Predictor and 2D DWT), as well as each sub-function independently. This way, the average accuracy has been computed on a total of 22 designs. Results are given in Table IV and Table V. Table IV reports the number of slices and logic elements after estimation and synthesis. Table V gives the average error in percent and reports the exploration vs. synthesis time. The average accuracy is about 13.5% and 10% respectively for area and execution time values. Some variations may be noticed locally, like in the case of function *upzero*, that are due to logic optimizations applied by the logic synthesis tool. In that case, the difference is due to the simplification of a multiplier with one of its operands remaining constant. Other variations like *horizontal* and *vertical rearrange* in the DWT are caused by the address generation model left to the control unit. This is especially significant in the case of this function where memory accesses are critical.

Concerning the processing times, only the logic synthesis times have been reported for comparison. The exploration times in the case of the G722 predictor and DWT are respectively 1 second and 5 minutes (to generate 16 and 342 solutions) while the respective logic synthesis times (for only *one* solution) are respectively 10 minutes and more than 1 day, plus the additional time needed to write the RTL description by hand (about 1 month). These results stress the benefits of the approach on the design times and the ability to rapidly select the most interesting solutions based on relatively accurate results.

	Virtex V400EPQ240-7				Apex EPK20K200EFC484-2X			
	Accuracy (%)		Explo vs lgc synth		Accuracy (%)		Explo vs lgc synth	
	slices	T_{ex}	T_{expl} (sec)	T_{synth} (min)	lgc elt	T_{ex}	T_{expl} (sec)	T_{synth} (min)
Parrec	-10	0	0.05	1	-10.5	0	0.05	1
Recons	-10	0	0.05	1	-10.5	0	0.05	1
Upzero	-14.9	+4.5	0.22	5	+52	+18.2	0.06	5
Uppol2	-15.2	-2.7	0.11	5	+19.4	+18.2	0.11	5
Uppol1	-21.5	-9.4	0.11	5	-3.8	+19.5	0.11	5
Filtep	-8	-12.5	0.05	1	-20.1	-8.7	0.05	1
Filtez	+2.2	+16	0.05	2	-2.6	+41.4	0.05	2
Predic	-10	0	0.05	1	-10.5	0	0.06	1
G722Predictor	-7.7	-7.1	0.9	15	+3.4	+14.1	0.4	10
1stHLftStep	+6.9	+7.1	0.1	5	+1.4	+7.6	0.05	8
1stHDLftStep	+4	+0.6	0.05	5	+2.6	+1.9	0.06	8
2ndHLftStep	+5.1	+13.9	0.06	5	+2.8	+9.3	0.05	8
2ndHDLftStep	+2.5	+9.8	0.06	5	-0.2	+1.7	0.05	8
Hscaling	+2.7	+3.6	0.1	5	+4.9	+3.6	0.1	8
Hrearrange	+46.8	-25	0.06	5	+67	+9.1	0.05	8
1stVLftStep	+7.1	+25.5	0.1	5	-0.2	+2.9	0.05	8
1stVDLftStep	+5	+16.9	0.05	5	-0.6	+5.1	0.06	8
2ndVLftStep	+5.1	+18.5	0.06	5	+1.1	+2.9	0.05	8
2ndVDLftStep	+3.4	+18.3	0.06	5	-2.6	+7.7	0.05	8
Vscaling	+3.4	+5.5	0.1	5	+3.2	+3.8	0.1	8
Vrearrange	+50.9	-5.5	0.06	5	+61	+3.8	0.05	8
DWT 2D	+35.9	+18.2	5min	1.5days	+37	+3.1	5min	2days

TABLE V

Estimation vs. Synthesis error and Exploration vs. (logic) Synthesis time for G722 Predictor function and DWT 2D function

FIR 16 (Design Trotter)	FIR16 (GAUT)
1 add 16 bit	1 add 16 bit
1 mult 16 bit	1 mult 16 bit
3 reg 16 bit	6 reg 16 bit
1 RAM, 1 ROM	Mem unit not generated
19 control steps	19 control steps
0.3 sec to generate 5 solutions	1 min to generate this solution
+ area / time estimates	+ 5 minutes for logic synthesis

TABLE VI

Design Trotter FIR 16 vs. HLS (GAUT) FIR 16

B.2 Comparison to HLS design

In this section, we propose to compare our exploration / estimation methodology with a High Level Synthesis approach. Thus, we perform an exploration using both *Design Trotter* (DT) approach and a HLS tool (GAUT HLS [30]). The example is a 16-tap FIR filter for which DT generates 5 RTL solutions. For the purpose of our comparison, we selected one of those 5 solutions and constrained GAUT HLS tool with the corresponding time constraint (304 *ns*) and clock period (16 *ns*). The left row of Table VI reports the characteristics of the DT solution and gives the necessary information to design the RTL architecture with the HLS tool. As we can see, there are some slight differences which are mainly due to the different RTL models used in both tools:

- concerning the number of registers, the difference is due to the fact that each functional unit is assumed to be associated with an output register in our approach (section III-B). So the actual number of registers is equal to 3 (originally estimated) plus 2 (one at the output of adder and multiplier).
- concerning memory, *Design Trotter* computes a basic estimation of the number of memories that is based on the analysis of simultaneous accesses (N_{ram_rd}, N_{ram_wr} and N_{rom}). GAUT HLS does not address the memory requirements in the current version used for this evaluation.

Finally, from the analysis of the processing times (bottom of Table VI) it arises the complementarity of the exploration methodology with High Level Synthesis more than an opposition: the use of a HLS tool may need several time consuming iterations before defining a suitable architecture. With our tool, several parallelism solutions are automatically generated with an average accuracy of 11.6%. Once a suitable solution has been selected, the designer can then constrain the HLS tool (using DT based directives about local / global time constraints, parallelism, scheduling, allocation, ...) to meet the solution. The confidence in finding more surely a suitable implementation is thus significantly enhanced.

	Solutions			
	pipelined		sequential	
	Xilinx	DT	Xilinx	DT
BRAM	1	16	1	1
Slice	1542	2368	157	168
Delay (*ns*)	306	304	2394	3213
Frequency(*MHz*)	81.5	62.5	116	62.5

TABLE VII
Design Trotter FIR 16 vs. IP (Xilinx) FIR 16

B.3 Comparison with pre-characterized IPs

To compare our exploration approach with existing IPs, we also considered a 16-tap FIR filter obtained from the Xilinx core generator [31]. Two IPs have been used: the first one is a full pipelined filter whereas the second is a full sequential filter. Design Trotter provides 5 solutions, we considered the fastest and the slowest ones for comparison with Xilinx's IPs in table VII. For the fastest solution, 16 BRAMs are needed because a maximum of 16 simultaneous memory accesses have been estimated while Xilinx's solution uses only one single BRAM. The clock value we have set corresponds to the delay of the slowest execution unit (multiplier in our case). It keeps the same value for both solutions because we do not consider any logic optimizations. Compared to Xilinx's solutions, the differences are due to the high optimization effort on the IP core. Unfortunately no information is available on this to permit comparison, for obvious confidentiality reasons. However, we believe the accuracy of estimations is reasonable regarding the abstraction level of the input specification. Further improvement on this point is possible and will be discussed in section VI-D.

The last part of this section emphasizes the possibility of design analysis introduced by our methodology (analysis of design parameters, exploration / synthesis relation).

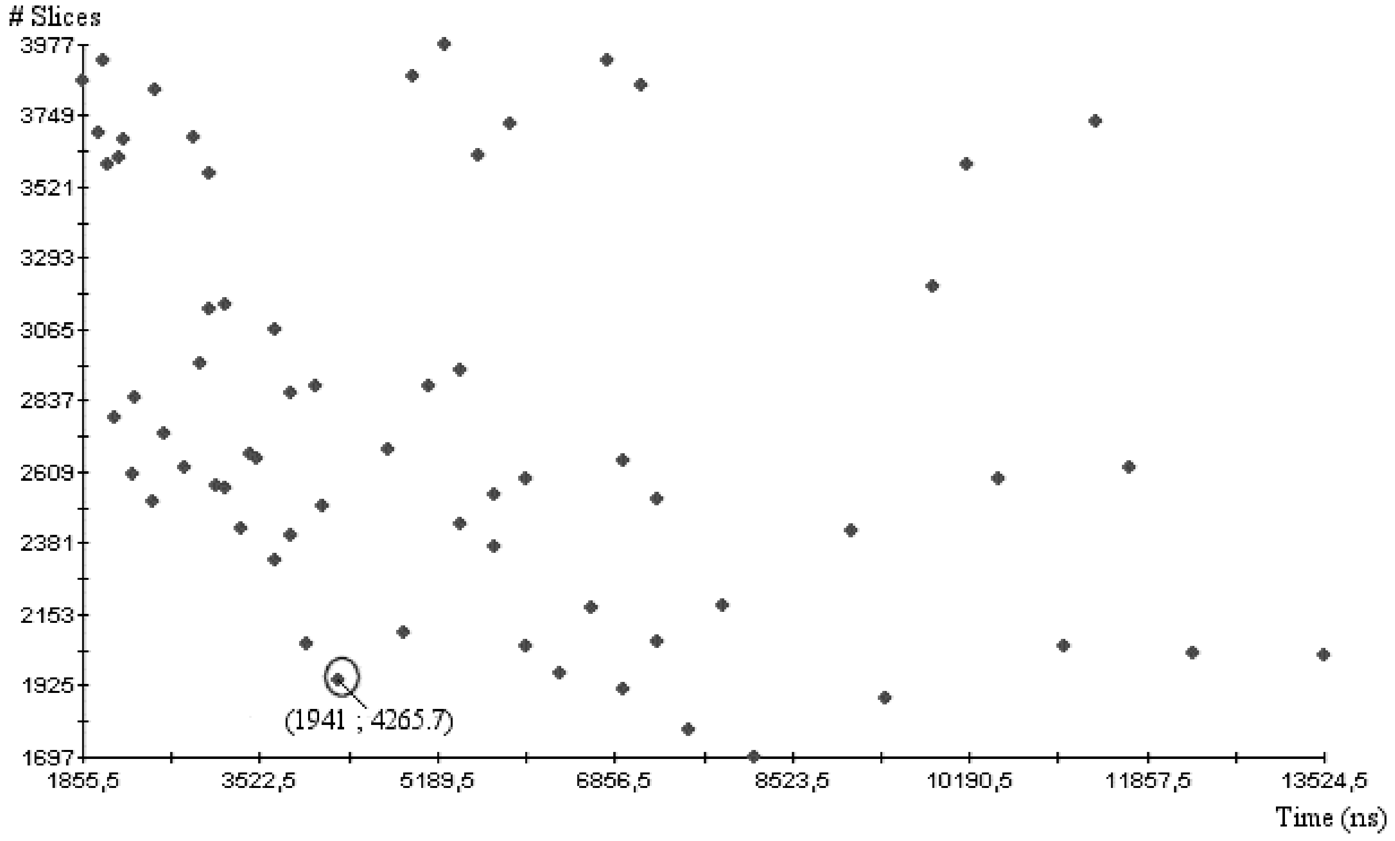

Fig. 8. Horizontal DWT exploration results (Virtex) - *slices* vs. time

C. *Exploration Approach*

The application considered is a 1D Discrete Wavelet Transform, Fig. 8 presents the estimation trade-offs for a Virtex V400EPQ240-7 device (Xilinx). For this example, the tool evaluates a total of 342 architectures, each one corresponding to a particular parallelism solution.

If we consider the solution highlighted (respectively 1941 slices / $4265.7ns$) that corresponds to the most interesting area / delay trade-off, the designer can refine the exploration around that solution. In this example, the exploration of several clock values and data bitwidths have been performed, results have been reported in figure 9 (labels correspond to clock period - data bitwidth values).

When a solution has been pointed out (for example clock = 20ns, bitwidth = 16), analyzing the details of the RTL architecture results provides useful information. In the example of table VIII, we are aware of that the solution is composed of 4 multipliers and 8 adders for an execution of 223 cycles. Mapped onto the FPGA, it corresponds to an occupation of 1941 (/4000) slices, 12 (/40) BRAMs and 256 (/4960) tristate buffers for a $4.3\mu s$ physical execution time. Due to the hierarchical approach of combinations, partial results are available at each level of the graph hierarchy. To refer our example, the For12_body sub-function is composed of 1 multiplier and 2 adders for the datapath and the RAM memory bandwidth is 1 write and 3 reads. Such partial results characterize each architectural solution and provide key design information. All this information can also be used to guide a HLS tool in the design of the solution.

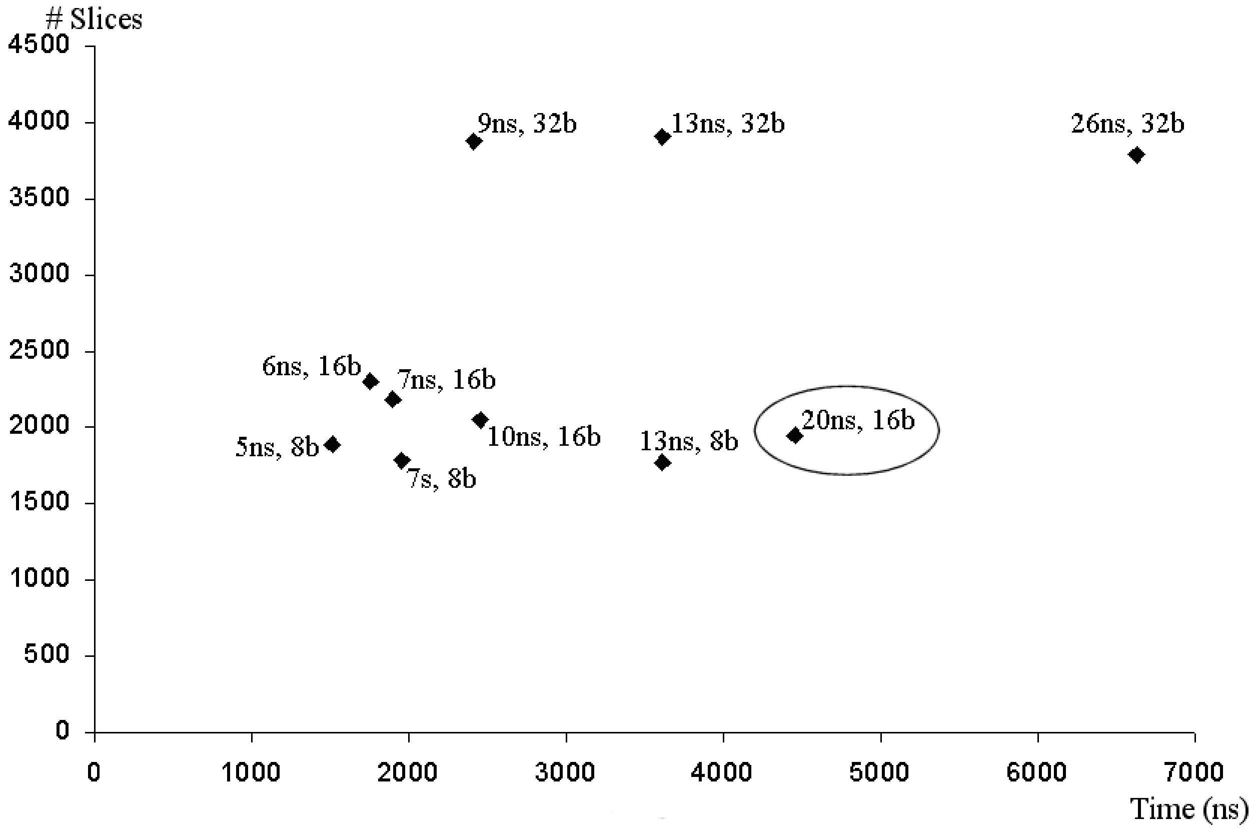

Fig. 9. Bitwidth and clock period exploration for a Virtex implementation

D. *Result Discussion*

The experiments performed pointed out some benefits / limitations of the exploration methodology. Hereafter, we discuss the relevance of the results and propose some possible enhancements. In the next section, we will focus on the contribution over related work in the context of future complex / heterogeneous system designs.

The structural exploration represents the core of the methodology which efficiency strongly depends on the feasibility and the relevance of the different parallelism solutions. As a starting point, we defined a first approach based on DFG scheduling and analytical CDFG combinations using simple execution models. The main reason of doing this was to reach a complexity level that could permit a large coverage of the design space from early specifications. This has been shown possible as regard to the accuracy and processing times reported (table V). Moreover, some enhancements are possible, of course at the expense of processing complexity, these are exposed below:

Graph	Cycles	States	Mul16	Add16	Reg16	RAM (wr)	RAM (rd)	ROM
For12_body	5	5	1	2		1	3	1
H1stLftStep	32	32	4	8		4	12	4
For22_body	5	5	1	2		1	3	1
H1stDLftStep	32	32	4	8		4	12	4
For32_body	5	5	1	2		1	3	1
H2ndLftStep	32	32	4	8		4	12	4
For42_body	5	5	1	2		1	3	1
H2ndDLftStep	32	32	4	8		4	12	4
For52_body	3	3	2			2	2	2
Hscaling	66	66	4			4	4	4
For62_body	2	2				2	2	
Hrearrange	33	33				8	8	
HDWT	223	223	4	8	28	8	12	4
	$T_{ex} = 4.3\mu s$		$slice = 1941$		$BRAM = 12$		$3state = 256$	

TABLE VIII

SOLUTION DETAILS FOR THE SELECTED RTL ARCHITECTURE LEADING TO THE FOLLOWING PERFORMANCE: 1941 SLICES, EXECUTION TIME $4265.7ns$

• Parallelism exploration: the weak point of our approach is the simplicity of the combination heuristics and the underlying execution model as we do not consider loop tiling and only consider a simplified model for loop unrolling and folding. On the other hand, an optimal schedule of the whole graph is something that would greatly impact on the estimation complexity. That's the reason we decided to schedule only the DFGs of the graph as a trade-off. A solution to enhance the quality of the solutions would be to apply a finer schedule of the critical processings, especially in the case of iterative structures with data dependences. We believe this is something that can be done without too much penalty on the exploration times, regarding the current complexity of the exploration methodology. We are investigating new loop scheduling heuristics able to analyze array index variations and consider all dependences to reach better schedules of loop iterations.
• Simplicity of the memory model: in the current version, memory estimation is based on a basic load / store model and characterized through the estimation of memory size and bandwidth requirements. We do not consider scalar replacement, data layout, data reusing, memory sharing and memory pipelining. This is justified in part by the storage resources within an FPGA that is composed of several distinct memories. For complex memory structures and execution model (using caches, pipelined execution modes, resources sharing), suited models and heuristics must be defined. There exist relevant works in this field that could be efficiently used for this purpose [4][16][32].
• Another way to enhance the quality of the RTL solutions is to allow the characterization of IP cores in the FPGA characterization (TAPM) file. For example, a Discrete Cosine Transform implementation can be described in the TAPM file (in terms of FPGA resources and delay) and used for estimation provided a specific DCT node is present in the H/CDFG. This possibility is enabled by the H/CDFG representation and could greatly enhance the exploration reliability. As an extension, any processing sequence or common functionality can be associated with an actual implementation provided information on area and delay is available in the technology file (in the manner of [9]). This could greatly help the problem of software compilation with the extraction of processing patterns associated with specific VLIW implementations (multiply-accumulate typically) or even including pre-characterized IPs for significant acceleration of critical kernels.

Concerning the physical estimations, some variations have been noticed when comparing physical synthesis results with area and performance estimates (table V). This is mainly due to some unconsidered low level optimizations.
• Control unit / address generation: variations may occur in case of high data processing / address generation. A solution is to consider a separate address generation unit from the control unit model.
• Logic optimizations: improvements are possible by considering some low level optimizations like operator area reduction when one operand remains constant, or a shift operation instead of a multiplication by a power of two.
• Clock period: clock value is defined before actually placing and routing the entire design and relies only on the characteristics described in the TAPM file. Thus, the actual critical path may be different than the estimated one. This difference might increase with the FPGA use rate since routing becomes more complex.
The impact of these low level effects is not critical since the average estimation accuracy is already around 15%. We believe the structural estimation step is a more important concern where additional processing complexity should focus on. Other scheduling strategies can be defined to better cope with the processing requirements of an application domain, a HLS tool procedure or an architectural implementation style. The architecture and scheduling models may have to be adapted in this case.

E. Comparison with existing approaches

Compared to the works presented in Table I, our approach can be summarized as follows:

- The estimator deals with a *behavioral* C description which is then parsed into a H/CDFG representation. So a complete characterization of the application is achieved in terms of Processing, Control and Memory unlike most approaches.
- Although it may be considered simplistic, the underlying implementation model is realistic since datapath, control and memory units are considered. It has been defined to cope with FPGA specificities and used in a way to enhance the complexity / accuracy trade-off in the exploration process. Other models are currently under study to refine the exploration of loop nests.
- The estimator provides area and delay values. No optimizations at the logic level are currently considered.
- The exploration of the design space is automatic. This means that several RTL solutions are defined for a given specification, without making several iterations of the exploration process.
- A variety of design parameters can be explored: implementation device, resource selection, clock period, data bitwidth, parallelism, ...
- The choice of an FPGA device is large and includes up to date devices. The TAPM characterization file includes logic cells as well as dedicated cells (DSP operators, embedded memories) which are important features in modern reconfigurable devices.
- The complexity of the algorithm is low ($O(n)$) since only a list-scheduling algorithm combined with a simple analytical method are applied. We are exploring finer schedule heuristics that could be applied to better analyzed critical parts of the application (complex loop processing with data dependences).
- The average accuracy is 10% for delay estimations and 20% for area values.

The application of this work has shown the possibility of using accurate estimations from *behavioral* C specifications to perform early design space pruning. The use of such low complexity estimations permits comparing very quickly different implementation alternatives and making reliable choices that are derived directly from the processing characteristics (in particular *the parallelism*) of the specification. The main conditions that have enabled this are the use of (1) a complete representation model, (2) a realistic implementation model and (3) scheduling heuristics combined with analytical approaches. These 3 conditions can be changed / adapted to cope with a specific design flow / application domain.

Concerning this, the definition of new models suited to the design procedure of a specific High Level Synthesis flow (we are currently exploring the use of Celoxica DK suite) would greatly enhance a complete design process by providing the HLS tool with essential information, especially concerning the *location and the amount* of parallelism in the behavioral specification. Moreover, an extension to ASIC design is possible through the definition of an appropriate TAPM file. This could lead to a complete framework for fast hardware prototyping.

VII. Conclusion and Perspectives

The starting objective of this work was to define an accurate estimation methodology from early specifications to move more efficiently through a design space pruning process. The accuracy available in the current (first) version of the exploration tool is high considering the abstraction level at the input. This results from the use of a complete specification model (H/CDFG) and realistic implementation models for the datapath, control and memory units. The interest of a scheduling approach is to bring both confidence concerning the feasibility of the solutions (compared to a true estimation approach) and reliable information to guide their design. The simplification of the scheduling process is important to quickly explore a wide range of parallelism solutions. The drawback maybe the relevance of the solutions defined in some cases, which is unavoidable given the abstraction level of the specification. But the complexity / design space coverage trade-off provided lets us expect easy adaptation to other implementation and execution models in a way to provide better quality to the final design.

The benefits of this methodology on a design process are multiple: it allows significantly reducing the design cycle (especially if used in complementarity with a HLS tool), to be less dependent from designer experience and synthesis tools (thus from technology evolution perspectives), and to converge faster toward an efficient and constraint compliant solution. Better application / architecture / device matching is reached due to the exploration coverage in terms of parallelism, devices, clock period, and functional unit allocation.

This approach has been integrated in a CAD framework for the codesign of heterogeneous SoCs called *Design Trotter* (https://designtrotter.univ-ubs.fr/).

References

[1] R. Hartenstein, *Are we ready for the Breakthrough ?*, Proceedings of the 10th Reconfigurable Architectures Workshop 2003 (RAW 2003), Nice, France, April 22, 2003.

[2] N. Tredennick and B. Shimamoto, *The Rise of Reconfigurable Systems*, In Proceedings of Engineering of Reconfigurable Systems and Application Conference (ERSA'2003), June 23-26, 2003, Las Vegas, Nevada, USA.

[3] G. Kulkarni, W. A. Najjar, R. Rinker, F. J. Kurdahi, *Fast Area Estimation to support Compiler Optimizations in FPGA-based Reconfigurable Systems*, Proceedings of International Symposium on Field-Programmable Custom Computing Machines (FCCM'02), April 21-24, 2002, Nappa, California, USA.

[4] B. So, P. C. Diniz and M. W. Hall, *Using Estimates from Behavioral Synthesis Tools in Compiler-Directed Design Space Exploration*, Proceedings of IEEE Design Automation Conference (DAC'03), June 2-6, 2003, Anaheim, California, USA.

[5] MATLAB-based IP for DSP Design of FPGAs and ASICs, White Paper, May 2004, www.accelchip.com

[6] The application of retiming to the synthesis of C based languages using the Celoxica DK Design Suite, White Paper, March 2004, www.celoxica.com

[7] A. Nayak, M. Haldar, A. Choudhary and P. Banerjee, *Accurate Area and Delay Estimators for FPGAs*, Proceedings of International Conference on Design Automation and Test in Europe (DATE'02), March 4-8, 2002, Paris, France.

[8] R. Enzler, T. Jeger, D. Cottet, and G. Tröster, *High-level area and performance estimation of hardware building blocks on FPGAs*, Proceedings of International Conference on Field-Programmable Logic and Applications (FPL'00), volume 1896 of Lecture Notes in Computer Science, pages 525-534. Springer, 2000.

[9] W. Miller and K. Owyang, *Designing a high performance FPGA – using the PREP benchmarks*, Proceedings of WESCON'93 Conference, pages 234-239, 1993.

[10] M. Xu and F.J. Kurdahi, *Area and Timing Estimation for Lookup Table Based FPGAs*, Proceedings of the European Design and Test Conference (ED&TC'96), March, 1996.

[11] M. Xu and F.J. Kurdahi, *Layout-Driven RTL Binding Techniques for High-Level Synthesis Using Accurate Estimators*, in ACM Transactions On Design Automation of Electronic System, Vol. 2, No. 4, p313-343, October 1997.

[12] P. Bjureus, M. Millberg and A. Jantsch, *FPGA Resource and Timing Estimation from Matlab Execution Traces*, Proceedings of Tenth International Symposium on Hardware/Software Codesign (CODES'02), May 6-8, 2002, Estes Parks, Colorado, USA.

[13] P.Banerjee, N. Shenoy, A. Choudhary, S. Hauck, A. Nayak and S. Periyacheri, *A MATLAB Compiler for Distributed, Heterogeneous, reconfigurable Computing Systems*, Procedings of IEEE Symposium on FPGA as Custom Computing Machines (FCCM 2000), 17-19 April 2000 - Napa Valley, CA.

[14] ALTERA, APEX 20K Programmable LogicDevice Family, August 2001, www.altera.com

[15] P. Diniz, M. Hall, J. Park, B. So and H. Ziegler, *Bridging the Gap between Compilation and Synthesis in the DEFACTO System*, Proceedings of the Languages and Compilers for Parallel Computing Workshop (LCPC'01), Aug 2001, Cumberland Falls, KY, USA.

[16] K. R. S. Shayee, J. Park, and P. C. Diniz, *Performance and Area Modeling of Complete FPGA Designs in the Presence of Loop transformations*, Proceedings of International Conference on Field-Programmable Logic and Applications (FPL'03), volume 2778 of Lecture Notes in Computer Science, pages 313-323. Springer, 2003

[17] S. Choi, J.W. Jang, S. Mohanty and V.K. Prasanna, *Domain-Specific Modelling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures*, Engineering of Reconfigurable Systems and Algorithms (ERSA'02), June 2002.

[18] S. Bilavarn, *Exploration Architecturale au Niveau Comportemental - Application aux FPGAs*, PhD, University of South Britanny, Feb 2002.

[19] S. Bilavarn, G. Gogniat and J.L. Philippe, *Fast Prototyping of Reconfigurable Architectures: An Estimation And Exploration Methodology from System-Level Specifications*, Proceedings of Eleventh ACM International Symposium on Field-Programmable Gate Arrays (FPGA'03), February 23-25, 2003, Monterey, California, USA.

[20] J.P. Diguet, G. Gogniat, P. Danielo, M. Auguin, J.L. Philippe, *The SPF Model*, Proceedings of International Forum on Design Languages (FDL'00), Tübingen, Germany, September 2000.

[21] Y. Moullec, J.P. Diguet and J.L. Philippe, *Design-Trotter: a Multimedia Embedded Systems Design Space Exploration Tool*, Proceedings of International IEEE Workshop on Multimedia Signal Processing (MMSP'02) December 9-11, 2002, St. Thomas, US Virgin Islands.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press and McGraw-Hill, September 2001.

[23] G. Grun, N. Dutt and F. Balasa, *System Level Memory Size Estimation*, Technical Report, University of California, 1997.

[24] D. Gajski, N. Dutt, A. Wu and S.Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[25] S. A. Blythe and R. A. Walker, *Efficient Optimal Design Space Characterization Methodologies*, in ACM Transactions On Design Automation of Electronic System, Vol. 5, No. 3, July 2000.

[26] XILINX, Virtex-II 1.5 Field Programmable Gate Arrays, July 2001, www.xilinx.com

[27] S. Narayan and D.D. Gajski, *Area and Performance Estimation from System-Level Specifications*, Technical Report, University of California, 1992.

[28] Recommandation G722, *Codage audiofrequence a 7KHz a un debit inferieur ou egal a 64Kbits/s*, Melbourne, 1988.

[29] I. Daubechies, *The Wavelett Transform, time-frequency localization and signal analysis*, IEEE Transactions on Information Theory, Vol. 36, No. 5, September 1990.

[30] E. Martin, O. Sentieys, H. Dubois, J. L. Philippe, *GAUT, an Architecture Synthesis Tool for Dedicated Signal Processors*, Proceedings of International EURO-DAC Conference, pp. 14-19, 1993.

[31] Creating a CORE Generator Module, ISE 6 In-Depth Tutorial, www.xilinx.com

[32] F. Catthoor, S. Wuytack, E. DeGreef, F. Balasa, L. Nachtergaele and A. Vandecappelle, *Custom Memory Management Methodology*, Kluwer Academic Publishers, 1998.