Fast Prototyping of Reconfigurable Architectures: An Estimation and Exploration Methodology from System-Level Specifications

S. Bilavarn
EPFL - ITS - LTS2
Swiss Federal Institute of Technology
sebastien.bilavarn@epfl.ch

Guy Gogniat
LESTER
South Britany University
Lorient, France
guy.gogniat@univ-ubs.fr

Jean Luc Philippe
LESTER
South Britany University
Lorient, France
jean-luc.philippe@univ-ubs.fr

September 27, 2002

Abstract

Rapid evaluation and design space exploration at the algorithmic level are important issues in the design cycle. In this paper we propose an original area vs delay estimation methodology that targets reconfigurable architectures. Two main steps compose the estimation flow: i) the structural estimation which is technological independent and performs an automatic design space exploration and ii) the physical estimation which performs a technologic mapping to the target reconfigurable architecture. Experiments conducted on Xilinx (XC4000, Virtex) and Altera (Flex10K, Apex) components for a 2D DWT and a speech coder lead to an average error of about 10 % for temporal values and 18 % for area estimations.

Contents

1 Introduction

The evolution of telecommunication and multimedia applications towards new standards requires innovative architectures in order to respect always tighter constraints (performance, power consumption, ...). The recent evolutions of reconfigurable architectures, in terms of capacity and performances, efficient resource integration (like DSP operators and memories), or flexibility through the possibility of run time reconfiguration, offer a very promising issue for reconfigurable system on chip. As a result, the choice of a suitable target component, satisfying both physical (area, performances, ...) and marketing (final product cost, time to market, ...) constraints is a complex issue often left to the designer experience. Dealing with such problems as application parallelism exploration and FPGA architecture matching, impose to define new design methodologies in order to find more quickly and surely an integration solution satisfying all the design constraints.

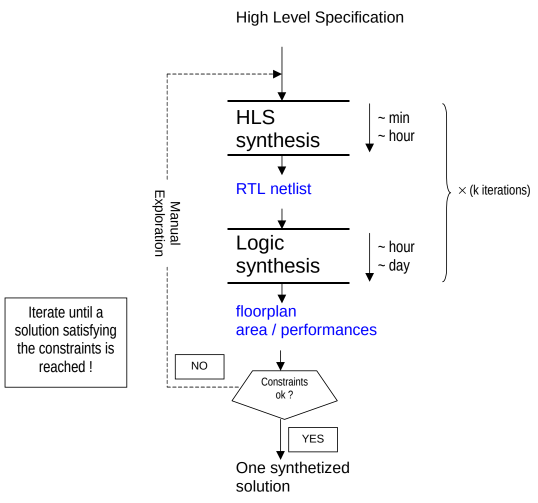


Figure 1: Design space exploration approaches

Until now, typical hardware design methodologies start, from an algorithmic description of the application (e.g. VHDL, SystemC), with an architectural synthesis step to obtain a description at the RT level. Then logic synthesis and place & route steps are performed to obtain the final description of the circuit and precise values of area (FPGA occupation) and performances (execution time). These two steps are very time consuming since the only architectural synthesis step can take from several hours (with a High Level Synthesis tool, HLS in the following) to several months (hand coding) to overcome. As shown in figure 1, design space exploration may need several iterations if constraints are not met, what can lead to prohibitive design times.

2 Objectives & Contribution

The purpose of the work presented in this paper is to define an efficient exploration methodology starting from system level specifications that allows: i) to define several architectural solutions and ii) to compute the corresponding estimated area and execution time values. The second point allows the designer to make a choice of a solution, while the first point gives information for the selected solution design. To find an interesting answer to this problem, the following considerations have been addressed:

- Define a method operating at a high level of abstraction, from system level specification including control structures, multidimensional data and hierarchy to deal with complex modern applications.
- Give realistic cost characterization: estimation takes into account all the different units of the architecture (datapath, control unit, memory unit).
- The method should explore the application parallelism: several architectural solutions are defined for a given specification.
- The method should be applicable to several FPGA families (Xilinx XC4000 & Virtex families and Altera FLEX & Apex families).
- Define feasible solutions and give sufficient information for post exploration steps (selected architecture design).
- Low complexity to enable large design space exploration.

The methodology developed can be seen as a global exploration / estimation technique based on the numerous existing works in the field of estimation and HLS (memory size estimation, scheduling techniques, data flow modelling, ...). Compared to other estimation approaches, the definition of effective architectures have been emphasized: each solution is implementable and corresponds to a given resource allocation, clock period value and scheduling. Their definition relies on a precise architectural model (not only datapath, but also memories and control units) and takes care of modern FPGA architectural specificities. Compared to a typical design exploration flow, we do not need to make a complete and precise description of the circuit. For example, we do not need to go until the precise description of the connections between resources, or to build a floorplan. Those steps are only needed to be computed once in the design cycle and are left to the steps following the exploration process (synthesis / refinement / optimization). The reduced complexity allows then to explore quickly the effect of different implementation possibilities (intra loop parallelism exploration, resource allocation, clock period, evaluation of several target FPGAs). Obviously, the solutions defined may be suboptimal in some cases, but they always correspond to implementable solutions. So estimation values computed (area and execution time) are more representative of the system's feasibility. Moreover, those metrics give a designer usefull information that allow to make an easiest choice for implementation (satisfying both area and execution time constraints). Once a solution selected, application synthesis and solution refinement / optimization can be performed in a classical way with the use of a HLS tool for example, thanks to the rich set of information given by the architecture definition step. This fast system level exploration allows then to evaluate many design possibilities very early in the design cycle, where choices have a great impact on the final system performances. The evaluation of several design possibilities allows moreover to converge more quickly and surely towards an optimal implementation solution.

3 Related Work

Most of the works concerning FPGA estimation focus on the problem of architecture and CAD tool optimizations. Only few methods deal with the problem of area and performance estimation on a FPGA technology, at the algorithmic level.

The approach proposed in [1] by Miller and Owyang is based on a benchmark library. A set of circuits are implemented and measured on a variety of FPGAs. Area and performance prediction is performed by partitioning the application into several components, that are substituted by the most similar benchmark circuit. However, the drawback of this approach is related to the difficult task of maintaining the library for different components and applications. Another methodology described in [2] by Xu and Kurdahi computes area and timing values based on models of the mapping process. Starting from a logic level netlist, it performs a CLB netlist construction which is then computed through a timing estimator that takes into account the overhead introduced by wiring effects. This method is very technological dependant since it targets the XC4000 family and can only be performed after a RTL synthesis step.

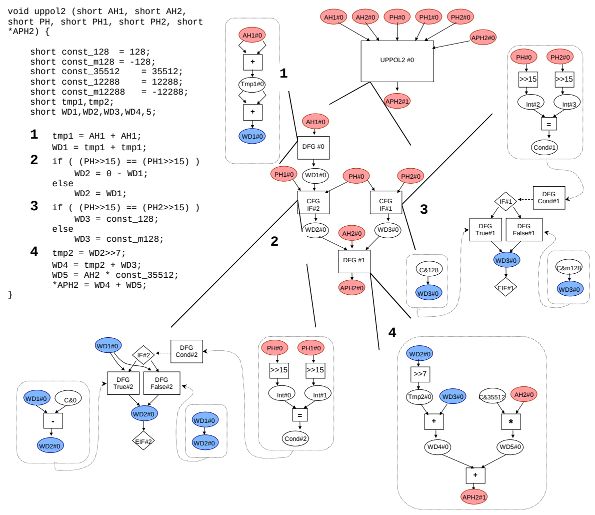
A method defined by Enzler & al. [3] allows estimation from a higher abstraction level. Area and delay are predicted by mean of combination of algorithm characterization (e.g. number of operations, parallelism degree) and FPGA mapping representation (operation mapping characteristics in terms of area and delay). Their approach is interesting but is limited to DFG specification, so it does not allow to deal with complex applications that involve control structures and multidimensional data. The same remark can be noticed concerning the method described in [5], which is based on a projection of the dataflow graph nodes, and where node characteristics are given by an approximation formula. Both methods do not take into account the memory and control overhead which may be critical in modern applications (video processing for example). Moreover, area estimation is limited to one kind of resource (Configurable Logic Cells in [3] or number of Look Up Tables in [5]), while modern FPGAs architectures contain dedicated resources for efficient implementation of specific functionalities (operators, memories, I/O, tri state buffers, ...). This particularity must be taken into account as it has a non negligible influence on the final component occupation and application performances.

Among the methods that care about data storage and control overhead, one can notice two interesting methods. Nayak & al. [4] propose a technique that performs estimation at the algorithmic level starting from MATLAB specifications. Their method estimates area and delay performances for a XC4010 component. Area estimation is performed after a scheduling step in order to define the number and the type of operator used. Delay estimation is based on IP characterization and takes into account interconnection overhead. However, they do not estimate the memory unit and they implement the control unit into CLBs, which may not always be the optimal solution as modern components allow efficient integration of product terms or ROMs into dedicated resources (Apex Embedded System Blocks for example). The other method [6], targets VLSI implementation but proposes an original technique for memory and control unit estimation. The method does not allow to automatically explore several implementation solutions since only one estimation is computed for a given specification. Nevertheless, the realistic cost characterization approach (processing, control, memory) and low complexity of the method are interesting properties in order to develop a global exploration technique.

In our approach, we propose an original area and delay estimator dealing with complex applications (including control structures and arrays) specified at an algorithmic level that takes into account datapath, control and memory units. Compared to published work, we perform a wide design space exploration, with complete cost characterization and target various significant components (XC4000, Virtex, Flex10K, Apex).

4 Exploration / estimation Methodology

First, the system level specification is given in a high level language (C language), and is then translated into an intermediate representation, the HCDFG model [7]. This model is a hierarchical control and data flow graph allowing efficient algorithm characterization and exploration of complex modern applications including control flow and multi-dimensional data. As illustrated in figure 2 a C program is decomposed into control structures called CFGs and into linear sequences of operations called DFGs. For example the If-Then-Else construct labeled 2 is composed of three DFGs, one for the evaluation of the condition and two for the True and False sequences of code. Hence, using the HCDFG model, the C program is converted into a hierachical graph. For further information about the HCDFG model please refer to [7].



```
void uppol2 (short AH1, short AH2,
short PH, short PH1, short PH2, short
*APH2) {

    short const_128  = 128;
    short const_m128 = -128;
    short const_35512    = 35512;
    short const_12288    = 12288;
    short const_m12288   = -12288;
    short tmp1,tmp2;
    short WD1,WD2,WD3,WD4,5;

    1   tmp1 = AH1 + AH1;
        WD1 = tmp1 + tmp1;
    2   if ( (PH>>15) == (PH1>>15) )
            WD2 = 0 - WD1;
        else
            WD2 = WD1;
    3   if ( (PH>>15) == (PH2>>15) )
            WD3 = const_128;
        else
            WD3 = const_m128;
    4   tmp2 = WD2>>7;
        WD4 = tmp2 + WD3;
        WD5 = AH2 * const_35512;
        *APH2 = WD4 + WD5;
}
```

Figure 2: C to HCDFG format

Starting from this specification and given a target component, the architectural exploration methodology (figure 3) consists in defining several implementation solutions and estimating FPGA resource occupation and algorithm execution time. To perform this estimation, we need to know the target FPGA characteristics which are described in a technology file [10]. Moreover, to give realistic estimation values, we use a specific architectural model and take memory requirements into account (the total memory size needed is estimated).

The Exploration / estimation flow is composed of two steps: i) structural estimations and ii) physical estimations.

The first step is technological independent and performs architectural exploration based on the considered architectural model (figure 6). Each solution is characterized for a number of cycle budget N_c by the number $Nop_k(N_c)$ and the type (op_k) of resources required to execute the application for this cycle budget. By changing the number of cycle budget N_c, we explore the design space. Another important characteristic of the architecture is the number of simultaneous read and write accesses to a RAM and read accesses to a ROM which are also computed during the structural estimations. This exploration conducts to several

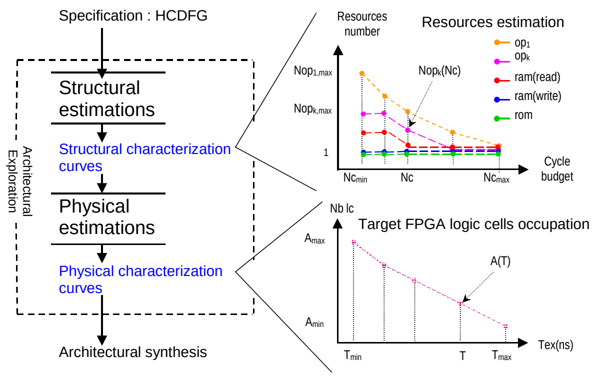


Figure 3: Exploration / estimation flow

architectural solutions that are characterized for a given cycle budget N_c: i) by the number and type of functional units ($N_{op_k}(N_c)$), ii) by the number of simultaneous read(write) from(to) RAM ($N_{ram\_rd}(N_c)$, $N_{ram\_wr}(N_c)$), iii) by the number of simultaneous read from ROM ($N_{rom\_rd}(N_c)$) and iv) by the number of control states ($N_s(N_c)$). All these results are gathered together in a 2D representation, where the vertical axis corresponds to the number of resources and memory accesses, and the horizontal axis to the number of clock cycles.

The second step is technological dependent and targets a specific FPGA technology. During that step, each architectural solution is characterized for a temporal constraint T by the FPGA resources occupation $A(T)$. FPGA resources considered are logic cells (resources that allow the configuration of user defined functions, e.g. slices, logic elements), dedicated cells (e.g. Block SelectRAM, Embedded System Block, those resources allow efficient implementation of specific functionalities like memories, product terms, DSP operators ...), tristate buffers and I/O pads. The FPGA description is given in a technology file that contains: i) the characteristics of the target FPGA (number of logic and dedicated cells, I/O pads, tristate buffers), ii) the characteristics of each functional units (area and delay) and iii) the characteristics of the memories (number of bits per logic or dedicated cells, access time). Note that the technology file is derived from the data sheet of the target FPGA and from the synthesis of basic arithmetic and logic operators.

4.1 Structural Estimations

The structural estimation step explores automatically different architectural solutions. It can be divided into four steps: i) Pre-estimation, ii) Selection/Allocation, iii) DFG scheduling and iv) Node combination.

Pre-estimation consists in verifying if a target FPGA is well suited for the considered application in term of I/O pads and memory resources. The number and the size of the formal I/O parameters used in the specification are compared to the number of I/O pads. Total memory size is computed (based on the technique proposed by Grun et al. described in [8]) and is compared to the amount of memory resources available into the target FPGA. If both conditions are verified, the next step is performed otherwise the designer has to select another target component.

Functional units are then selected. Each functional unit is described in a component library (technology file) and is characterized by the following features: bitwidth, delay, type and number of resources consumed in the FPGA. For example an eight bits adder implemented into a Virtex V400EPQ240-7 is characterized as follows: 8 bits, 4.9 ns, 4 slices (these results are obtained from the Xilinx Foundation tool). The designer has three approaches to select/allocate functional units to operations: i) an automatic exhaustive analysis, ii) an heuristics based approach and iii) a manually guided approach. When the designer wants to explore the whole design space an automatic exhaustive analysis can be selected or the designer can manually select the more interesting functional units according to his or her experience. Once functional units are allocated, the clock period analysis is performed. As for the allocation step, the designer can select to make an exhaustive analysis of each clock period between T_{min} and T_{max}, where T_{min} (respectively T_{max}) corresponds to the propagation delay of the faster (respectively slower) functional unit or to make a guided exploration. Moreover, this approach allows to be less design experience dependant as one can make an exhaustive exploration and refine

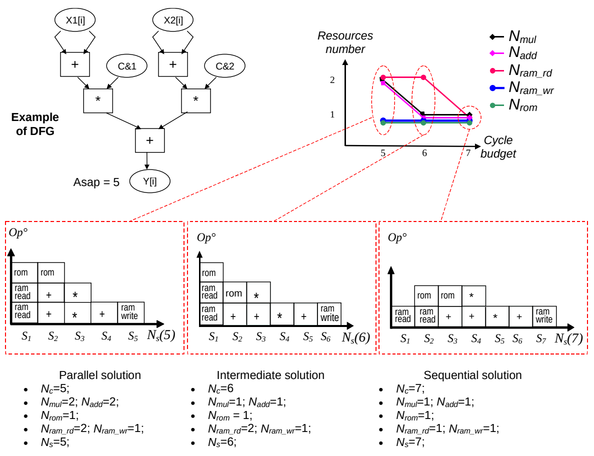


Figure 4: DFG scheduling example

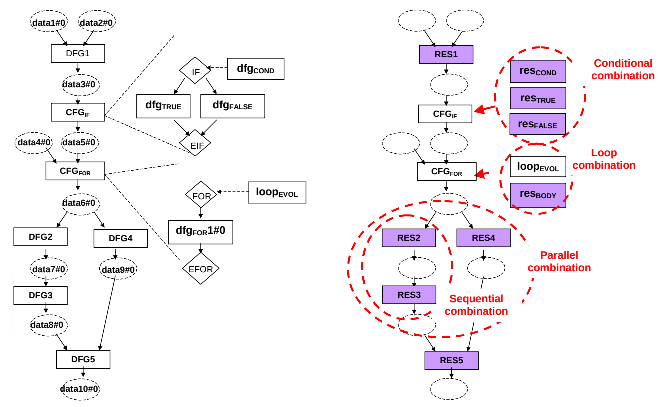


Figure 5: Combination heuristics

progressively the results.

The scheduling step is applied to all the basic blocks of the graph (i.e. DFGs). As illustrated figure 4, scheduling is performed for several time constraints (N_c), from the most parallel solution (corresponding to the critical path) to the most sequential one (where only one functional unit of each type is needed). Two scheduling algorithms are currently integrated in our exploration tool and can be used during that step, a Force Directed Scheduling or a List Based Scheduling [11]. Both algorithms have been extended in order to compute the number of simultaneous memory accesses which are requested to estimate the memory unit characteristics as explained in the following. According to the DFG complexity and the design space the designer wants to explore, one of the two algorithms can be selected. Extension to other scheduling algorithms can be done easily.

Once a DFG have been scheduled, it is replaced by its estimation result (i.e. resources vs cycles curve). In order to estimate the whole HCDFG specification, combinations of DFG results must be realized. Since a HCDFG specification can be composed of four types of dependencies between DFGs (figure 5): i) two of them correspond to execution dependencies (i.e. sequential and parallel) and ii) the two others correspond to control dependencies (conditional and loop structures), it is necessary to take them into account. For each type of dependency, the DFG estimation result points (corresponding to the time constraints N_{c_i}) are combined by pair according to the following equations (combination rules are detailed in [10]):

Sequential combination:

$$N'_c = N_{c_1} + N_{c_2}$$

$$N'_s(N'_c) = N_{s_1}(N_{c_1}) + N_{s_2}(N_{c_2})$$

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

Parallel combination:

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

$$N'_s(N'_c) = N_{s_1}(N_{c_1}) + N_{s_2}(N_{c_2})$$

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

Conditional combination:

$$N'_c = N_{c0} + [P_{b_1} * N_{c_1}] + [P_{b_2} * N_{c_2}] + 1$$

$$N'_s(N'_c) = N_{s0}(N_{c0}) + N_{s_1}(N_{c_1}) + N_{s_2}(N_{c_2}) + 1$$

$$N'_{op_k}(N'_c) = MAX[N_{op_{k_0}}(N_{c0}), N_{op_{k_1}}(N_{c_1}), N_{op_{k_2}}(N_{c_2})]$$

Loop combination (2 scheduling possibilities):

Sequential execution:

$$N'_c = N_{iter} * (N_c + 1)$$

$$N'_s(N'_c) = N_s(N_c) + 1$$

$$N'_{op_k}(N'_c) = N_{op_k}(N_c)$$

Partial unrolling and folding:

$$N'_c = N_c + (N_{iter}/f_p - 1) * k'$$

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

$$N'_{op_k}(N'_c) = N_{op_k}(N_c) * f_p$$

Sequential execution of two graphs is estimated under the assumption of maximum resource sharing: the number of functional units is the maximum number of functional units and the execution time is the sum of the two execution times. Parallel execution is estimated without considering resource sharing to avoid a significant increase of the estimator complexity. Concurrent controllers are used in this case. Conditional structure estimations need the help of branching probabilities (P_{b_1} and P_{b_2}) that can be obtained by profiling the application. Concerning loop estimation, partial unrolling may be analyzed when possible. In this case, several parallelism degrees are represented by the parallelism factor f_p. k' represent the latency of the slowest functional unit (multi cycle functional unit execution case).

The HCDFG specification is recursively analyzed thanks to those combination heuristics. It starts with DFG nodes combination (bottom level of the hierarchy) and the process goes on until it remains only one node, representing the whole application (top level of the hierarchy), so we obtain the whole graph estimation results. The next step computes the area / time trade-off for all the solutions explored.

4.2 Physical estimations

Physical estimations allow to compute the FPGA resources occupation (logic cells lc, dedicated cells dc, ...) and performances (execution time of the algorithm given in ns, μs) of the previously defined solutions over the targeted FPGA. Technology mapping of each solution is performed through a complete characterization process that takes care of each unit of the architecture. A precise model have been defined for the memory unit, datapath and control unit (figure 6) in order to cope with FPGA architecture specificities. We first analyze the memory unit cost: the number and size of memories, and the number of control signals are derived from the total memory size estimation results, simultaneous memory accesses count, and memories characteristics (ROM / RAM, number of ports). The processing unit area and control lines needed to drive the datapath are computed from the knowledge of the number of functional unit of each type and from their characteristics (area, bitwidth, ...). The total number of states and control signals allows to compute an estimation of the control unit cost. The following equations describe the technology mapping process. Details about architectural model and equation definition are related in [10].

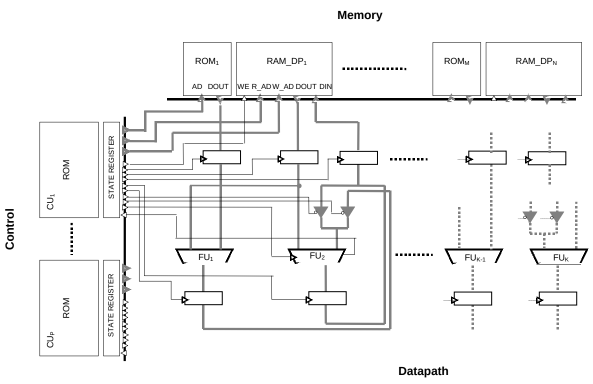


Figure 6: Architectural Model

Memory unit:

Logic cell based implementation:

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

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

Dedicated cell based implementation:

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

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

Control signals:

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

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

Address bus size:

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

$$W_{adr}^{rom} = \lceil log_2(MS_{ROM} / N_{rom}) \rceil$$

Total number of control signals:

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

where MS represents the total memory size (in term of words) estimated at the pre-estimation step, and where W corresponds to the size of a word.

Processing unit:

Area:

$$A_{PU} = \sum_{op_k} N_{op_k} * A_{op_k}$$

Control signals:

According to the architectural model, there are four types of control signals: signals for the control of the output register associated with each functional unit (N_{cs}^{op}), signals for multi-functional units operation selection ($N_{cs}^{multi\_op}$), signals for the control of registers associated with the memories read/write ports ($N_{cs}^{reg} = N_{reg}^{ram} + N_{reg}^{rom}$) and signals for multiplexors / tristate control (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 is:

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

Control unit:

ROM size estimation:

$$N_{bits\_state\_reg} = log_2(N_s)$$

$$N_{bits\_rom} = N_{bits\_state\_reg} + N_{cs}$$

Global cost computation:

$$T = N_c * T_{clk}$$

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

This computation process is then iterated for each architectural solution defined by the structural estimation step and leads to the final cost vs performance curve (figure 3).

5 Experiments & Results

5.1 From specification to synthesis

In this section, the design cycle described above is applied to the example of a half Discrete Wavelet Transform (DWT). Specification is written in the C language for test and simulation, and is then translated into the intermediate representation model (HCDFG) on which the exploration / estimation tool works. The DWT application is composed of 4 filtering / lifting schemes followed by a scaling process and image re-arrange, described by 2^{nd} order nested loops. Figures 7 and 8 shows the exploration results for two target components: Xilinx Virtex V400EPQ240-7 and Altera Apex EP20K200EFC484-2X. We have only represented the logic cells occupation (where the maximum number is respectively 4000 *slices* for Virtex and 8320 *logic elements* for Apex) vs excution time (*ns*) curves as they represent the most significant FPGA resource occupation for this example. As we can see on the figures, exploration provides 65 architectural solutions in both cases, each one corresponding to a different parallelism degree. Let's for example consider the solution highlighted in figures 7 and 8 since it corresponds to a good area/speed trade-off.

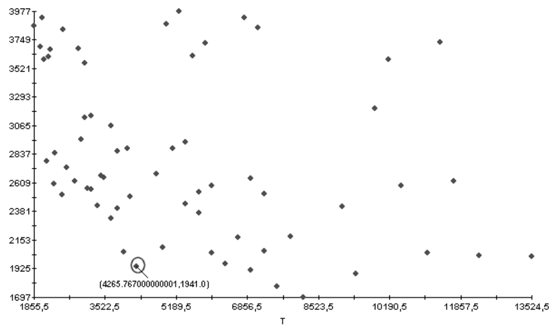


Figure 7: Horizontal DWT exploration results (Virtex) - *slices* vs time

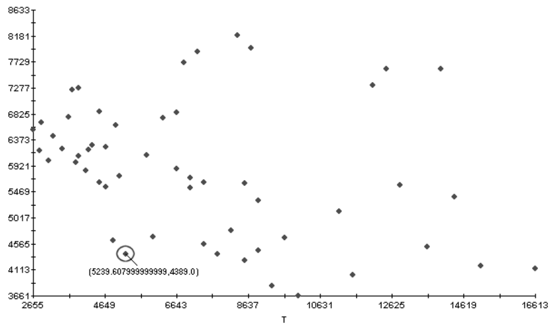


Figure 8: Horizontal DWT exploration results (Apex) - *logic elements* vs time

Based on that solution the designer may want to refine the exploration. For example, in this experiment the default clock period value corresponds to the slowest functional unit delay used in the architecture. Hence, the designer can refine the exploration results obtained previously by analyzing the effect of different clock periods and resource allocation. For the solution selected before, several clock values and data bitwidths are estimated in figure 9 (labels correspond to a couple clock period value - data bitwidth). Thanks to those information, the designer can quickly evaluate if a solution defined by a parallelism degree, clock value, resource allocation and target FPGA, can meet the design constraints or not.

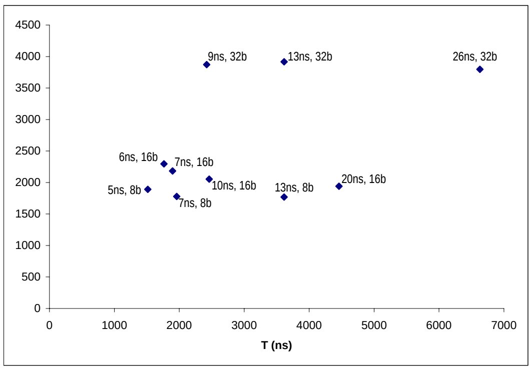


Figure 9: Bitwidth and clock period exploration

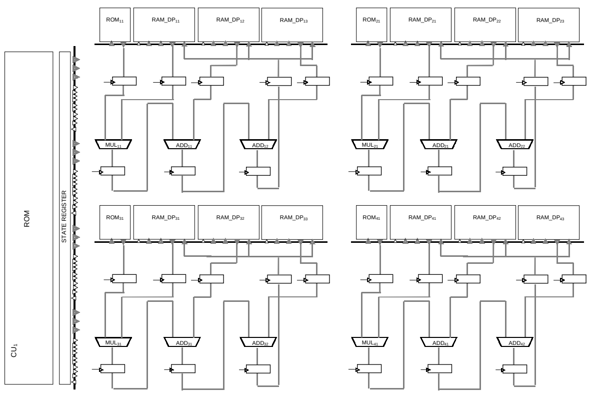


Figure 10: Selected solution architecture

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	--
Hreaarange	33	33	--	--	--	8	8	--
Hdwt	223	223	4	8	28	8	12	4
	T_{ex} : 4.5 µs		Slices : 1941		BRAM : 12		3 state : 256	

Table 1: Selected solution details

Once a solution have been selected (for example the one with a clock period value equal to 20 ns and a bitwidth equal to 16), details and corresponding structural estimation results for each hierarchy level (each subgraph of the specification) are also available in our tool (table 1). Those partial results fully characterize each architectural solution and give the designer all the necessary information needed for the system design. In the case of our example, we can see that the selected solution is composed of 4 multipliers and 8 adders for a 223 cycles execution, which correspond to a resource occupation of 1941 (/4000) *slices*, 12 (/40) BRAMs (dedicated resources for memory implementation) and 256 (/4960) tristate buffers (used in case of resource sharing or conditional branches) for a $4.5\mu s$ execution time. The corresponding architecture is given in figure 10.

5.2 Precision & Exploration time

In this section, we discuss the precision of the occupation vs execution time estimations and give values of the exploration time vs logic synthesis time needed. These measures have been performed with two representative of recent FPGA families (Virtex and Apex) for a speech coder (G722) and a 2D DWT (table 2). The corresponding architectures have been synthesized in order to study estimation values precision. Note here that to study this precision, the architectures have been hand coded at the RTL level (it tooks about one month to write each solution) in order to cope exactly with our architectural model (the use of a HLS tool would have lead to significant estimation errors as it does not generate the same architecture). That's the reason why in the following, exploration times are only compared to the *logic synthesis* times (architectural synthesis times are about hours for a HLS tool and months for hand coding). The Foundation and Quartus synthesis tools have been used to target respectively Virtex and Apex FPGA.

The speech coder application is composed of eight functions that correspond for example to filtering and prediction operations. These functions are mainly control and computation oriented. The 2D DWT example is characterized by numerous memory accesses and computations. The average error is about 10 % for temporal values and 18 % for area estimations which represent a good bound for the designer since the application is described at the algorithmic level. Locally more important errors can be noticed which are due: i) to logic optimizations automaticaly performed by the synthesis tools which are not taken into account in our approach or ii) to the considered control unit architectural model that has the charge of

EXAMPLE	Virtex V400EPQ240-7				Apex EP20K200EFC484-2X			
	Precision (%)		Expl vs lgc synth		Precision (%)		Expl vs lgc synth	
	slices	T_{ex}	T_{expl}	T_{synth}	lgc elt	T_{ex}	T_{expl}	T_{synth}
Parrec	-10	+1.4	0.05 sec	1 min	-10.5	+4.9	0.05 sec	1 min
Recons	-10	+1.1	0.05 sec	1 min	-10.5	+4.9	0.05 sec	1 min
Upzero	-14.9	+4.5	0.22 sec	5 min	+52	+18.3	0.06 sec	5 min
Uppol2	-15.2	-2.7	0.11 sec	5 min	+19.4	+18.6	0.11 sec	5 min
Uppol1	-21.5	-9.8	0.11 sec	5 min	-3.8	+19.6	0.11 sec	5 min
Filtep	-8	-13.1	0.05 sec	1 min	-20.1	-8.4	0.05 sec	1 min
Filtez	+2.2	+16.1	0.05 sec	2 min	-2.6	+41.4	0.05 sec	2 min
predic	-10	-2.2	0.05 sec	1 min	-10.5	+4.9	0.06 sec	1 min
G722Predictor	-7.7	-7	0.9 sec	15 min	+3.4	+14.1	0.4	10 min
1stHLftStep	+6.9	+7.1	0.1 sec	5 min	+1.4	+7.6	0.05 sec	8 min
1stHDLftStep	+4	+0.6	0.05 sec	5 min	+2.6	+1.9	0.06 sec	8 min
2ndHLftStep	+5.1	+13.9	0.06 sec	5 min	+2.8	+9.3	0.05 sec	8 min
2ndDHLftStep	+2.5	+9.8	0.06 sec	5 min	-0.2	+1.7	0.05 sec	8 min
Hscaling	+2.7	+3.6	0.1 sec	5 min	+4.9	+3.6	0.1 sec	8 min
Hrearrange	+46.8	-25	0.06 sec	5 min	+67	+9.1	0.05 sec	8 min
1stVLftStep	+7.1	+25.5	0.1 sec	5 min	-0.2	+2.9	0.05 sec	8 min
1stVDLftStep	+5	+16.9	0.05 sec	5 min	-0.6	+5.1	0.06 sec	8 min
2ndVLftStep	+5.1	+18.5	0.06 sec	5 min	+1.1	+2.9	0.05 sec	8 min
2ndDVLftStep	+3.4	+18.3	0.06 sec	5 min	-2.6	+7.7	0.05 sec	8 min
Vscaling	+3.4	+5.5	0.1 sec	5 min	+3.2	+3.8	0.1 sec	8 min
Vrearrange	+50.9	-5.5	0.06 sec	5 min	+61	+3.8	0.05 sec	8 min
DWT 2D	+35.9	+18.2	5 min	1.5 days	+37	+3.1	5 min	2 days

Table 2: Estimation vs Synthesis error and Exploration vs (logic) Synthesis time

setting the address signals. This is particularly true for the 2D DWT example where numerous memory accesses are performed.

The exploration / estimation computational time is very fast since in the case of the G722, 16 solutions are estimated in about 1 second and in the case of the 2D DWT, 350 solutions are estimated in 5 minutes on a Pentium III running at 800 MHz. In table 2, solutions for both applications have been manually written at the RTL level and then logic synthesis and place & route steps have been done automatically. As exhibited in the figure, the exploration / estimation approach enables to reduce strongly the design cycle. Hence, the designer can focus on a subset of architectural solutions that presents the best delay vs area trade-offs.

6 Conclusion & Perspectives

In this paper we present an automatic exploration / estimation methodology at the algorithmic level. This approach, which has been integrated in the codesign environment *Design Trotter* [9], enables to explore a large design space at an early stage of the design cycle and to characterize each solution in terms of area vs delay. In order to provide the designer useful bounds, the control, datapath and memory units are considered and several FPGA technologies can be targeted. The time saving resulting from this approach is significant and allows to shorten strongly the time to market constraints as well as to converge towards a better application / component matching. Some extensions of this work are currently being studied to consider a separated address generation unit, to take into account some synthesis optimizations to improve local errors and to include power consumption estimation.

References

[1] W. Miller and K. Owyang, *Designing a high performance FPGA – using the PREP benchmarks*, in Wescon'93 Conf. Record, pages 234-239, 1993.

[2] M. Xu and F.J. Kurdahi, *Area and Timing Estimation for Lookup Table Based FPGAs*, Proceedings of ED&TC, March 1996.

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

[4] A. Nayak, M. Haldar, A. Choudhary and P. Banerjee, *Accurate Area and Delays Estimators for FPGAs*, Proceeding of DATE, March 2002, Paris, France.

[5] G. Kulkarni, W. A. Najjar, R. Rinker, F. J. Kurdahi, *Fast Area Estimation to support Compiler Optimizations in FPGA-based Reconfigurable Systems*, Field Programmable Custom Computing Machines (FCCM'02), Nappa, California 2002.

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

[7] J.P. Diguet, G. Gogniat, P. Danielo, M. Auguin, J.L. Philippe, *The SPF Model*, FDL, Tübingen, Germany, September 2000.

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

[9] Y. Moullec, J.P. Diguet and J.L. Philippe, *Design-Trotter: a Multimedia Embedded*

Systems Design Space Exploration Tool, IEEE MMSP02 (Workshop on Multimedia Signal Processing) December 9-11, 2002, St. Thomas, US Virgin Islands.

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

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