An Area Estimation Methodology for FPGA Based Designs at SystemC-Level

Carlo Brandolese, William Fornaciari, Fabio Salice
Politecnico di Milano – DEI
Piazza L. Da Vinci, 32 – 20133 Milano – Italy
{brandole,fornacia,salice}@elet.polimi.it

ABSTRACT

This paper presents a parametric area estimation methodology at SystemC level for FPGA-based designs. The approach is conceived to reduce the effort to adapt the area estimators to the evolutions of the EDA design environments. It consists in identifying the subset of measures that can be derived form the system level description and that are also relevant at VHDL-RT level. Estimators' parameters are then automatically derived from a set of benchmarks.

Categories and Subject Descriptors: B.6.3 [Logic Design]: Automatic Synthesis

General Terms: Design, Performance.

Keywords: FPGAs, SystemC, Area metrics.

1. INTRODUCTION

A fast and effective design space exploration implies focusing on functional descriptions while providing metrics and corresponding reliable estimators. We describe a methodology to extract from a SystemC description a set of estimators for an area metric [1], considering as target technology the Xilinx VirtexII-Pro FPGA family [2], for which area can be expressed in terms of number of Look-Up Tables (LUTs) and the number of Flip-Flops (FFs). The design flow adopted for assessing the methodology is based on Synopsys CoCentric SystemC Compiler [4] and Mentor Graphics Leonardo Spectrum.

A significant amount of research has been focused on area, timing and power estimation of implementations whose target is an FPGA. These activities can be roughly partitioned in two sets: those extracting information from RTL descriptions (VHDL, Verilog) and those extracting information form behavioral level descriptions. In the latter case, the problem is faced by translating behavioral descriptions (Matlab [5, 6], other [7, 8]) into VHDL-RT [5, 6] or DFG [7]. Though these transformations are required whenever close-fitting results are needed, the effort necessary to transform a high-level model into a more detailed model is not currently justified. The reason is twofold: on one hand logic-level tools provide a fast and accurate estimation, on the other hand the most time-consuming activity of the high-level synthesis process is CDFG extraction and manipulation and module pre-characterization. In short, an effective estimation methodology requires identifying *shortcuts* instead of *replications*.

Experience suggests that different implementations can be obtained depending on the combination of high-level and logic synthesis tools. To overcome the above problem, we structured the methodology in two levels: the higher one models the SystemC to RTL-VHDL transformation while the lower accounts for RTL-VHDL to FPGA netlist. Other approaches for the lower portion of the methodology are presented in [9, 10]. Such a solution shields the overall accuracy from possible changes of synthesis tools and algorithms. It is a tradeoff between accuracy and applicability tailored for design space exploration, in the sense that the estimated area is satisfactorily accurate, considering the amount of information that needs to be extracted from the system description and the complexity of their estimation.

2. METHODOLOGY

In most of the previous works, the main problem was to identify a fast and accurate estimation procedure without adopting synthesis-like algorithms while reducing the obsolescence of the estimators due to technology evolution. The dilemma can be solved if a class of parametric estimators that can be automatically re-tuned whenever tools and/or technology change are identified. Despite the appealing of generalizing the problem including the estimator identification in an automatic procedure, a rough analysis of a generic design flow highlights the following elements:

1. tools for logic synthesis are quite stable; thus, metrics produced at this level can be used as a reference point;
2. high-level synthesis target architecture (CU, DP, memories) seems to be excluded from the brawl involving high level languages and tools: it can be considered a stable interface between the two abstraction levels;
3. the proposed architectures of the data path (mux or bus based) do not impact on the methodology results;
4. high level synthesis is mainly based on templates.

These observations enforce a pragmatic approach to the estimators identification problem: high level and logic level have to be considered separately, taking into account their interaction only. This means considering the subset of those VHDL-RTL descriptions that are produced by high-level synthesis tools.

DAC 2004, June 7–11, 2004, San Diego, California, USA.

2.1 Introduction to methodology

The proposed methodology is composed by three separated and consecutive steps: *Models*, *Variables* and *Parameters*. The Models step copes with the problem of identifying the form of the estimator (algebraic and/or algorithmic). In this phase the analysis of the results produced by the tools is particularly significant. Model identification is a human activity to be carried out until the estimator set is satisfactory. The Variables step focuses on the identification of the variables set that is statistically significant and fulfill user directives and constraints. Variables set identification is a semi-automatic procedure: model variables are automatically extracted by a larger set of variables manually identified. The Parameters step identifies the parameters for each estimator by using two sets of benchmarks: set-up and validation. In this phase, coefficients for algebraic estimators and compensation coefficients for algorithmic estimators (when required) are automatically determined. Parameters identification is an operation that must be frequently performed, in particular, when a new set of benchmarks is available, when tools changes and, finally, when the target device changes.

2.2 Methodology framework

A generic system D, whose system-level description is $L(D)$, can be decomposed into five sets of functional units: combinational (RC), multiplexer (MX), sequential (FSM), operators (OP) and register (REG). Let Λ_a and P_a be the actual number of LUTs and registers required to implement the design D and Π_a the whole set of parameters characterizing the sets of functional units (number of inputs, number of processes, hierarchy level, number of VHDL lines, ...). Goal of the methodology is to identify an estimators set (λ, ρ and π) and a subset of variables $\Theta_a \subset \Pi_a$ such that:

$$\Lambda_{e,i} = \lambda(\Theta_{e,i}); \quad P_{e,i} = \rho(\Theta_{e,i}); \quad \Theta_{e,i} = \pi(L(D_i)) \quad (1)$$

$$\min ||\Lambda_a - \Lambda_e|| \wedge \min ||P_a - P_e|| \quad \forall D_i \in \Delta_{setup} \quad (2)$$

where Λ_a, Λ_e, P_a and P_e are vectors, $||\cdot||$ is the Euclidean norm and Δ_{setup} is a set of benchmarks. It is worth noting that the adopted procedure is valid in general but that the solution space has been reduced assuming that the design is synthesized without hierarchy flattening. This hypothesis, which surely holds for large designs, allows considering the system as a set of decoupled components so that area can be expressed as the sum of single components. Equations (1) and (2) thus become:

$$\Lambda_e = \textstyle\sum_{\omega \in \Omega} \alpha_\omega \Lambda_{e,\omega} \quad (3)$$

where $\Omega = \{RC, MX, FSM, OP, REG\}$ and:

$$\Lambda_{e,\omega} = \lambda_\omega(\Theta_{e,\omega}) \quad (4)$$

$$\Theta_{e,\omega} = \pi_\omega(L(D_i)) \quad (5)$$

Although this paper focuses on area estimation, other parameters can as well be estimated using this property. In particular, energy [11] can be expressed as a linear combination of the estimated energy of each part whereas time (in term of frequency or throughput) is bound by one or more components of the system. By considering the last formulation, the core of the estimation procedure is the identification of a subset of variables $\Theta_a \subset \Pi_a$ that can be both measured on the system description $L(D_i)$ and correlated with FFs and LUTs. Although other promising approaches can be considered, the variables used are derived from a classical high-level synthesis procedure. In particular, among information like loop nesting, number of code lines, input and output counts and size, variables lifeness, control steps, etc., the selected subset only includes the number of control steps, number of control inputs, number of control outputs, number and size of data registers, number and size of multiplexers and number, type and size of operators. At RT level, these variables are easily correlated to FFs and LUTs. Though at this level other variables are even better correlated with LUTs and FFs (e.g. the number of transitions in a FSM), their potential gain in accuracy does not balance the increased computational effort. These issues emphasize that, although several variables are available at high- and low-level, the constraint of exportability from HL to LL, the ease to calculate their value and the accuracy of the estimation reduce the variable sets. At high-level (HL), a preliminary operation consists in determining a subset of variables that can be extracted with a specific computational effort (the upper bound being that of high-level synthesis). At this level, the analysis of the synthesis procedure could point out some synthesis templates that can be profitably used to determine the value of some variables. For example, on a SystemC model compiled by Synopsys CoCentric SystemC Compiler, the relation between the number of control steps and some description characteristics (e.g. wait statements, loop boundaries, etc.) can be identified. At low-level (LL), the set of available variables is pruned by a statistical analysis (variable correlation with FFs/LUTs). Successively, the two subsets of variable (LL and HL) are automatically correlated to each other and analyzed to determine their relevance (principal component analysis). After this stage the two subsets constitute the base of the estimators.

2.3 Methodology structure

The methodology is based on two cascaded models related to the two synthesis phases as depicted in Figure 1. The upper slice of the figure shows a generic synthesis flow from SystemC to FPGA netlist. The lower portion depicts the proposed two-level methodology based on HL and LL models. A key point of this works is the automation of the tuning procedure to minimize the effort required to adapt the whole flow to changes of the synthesis tools that, in such process, can be seen as black-boxes.

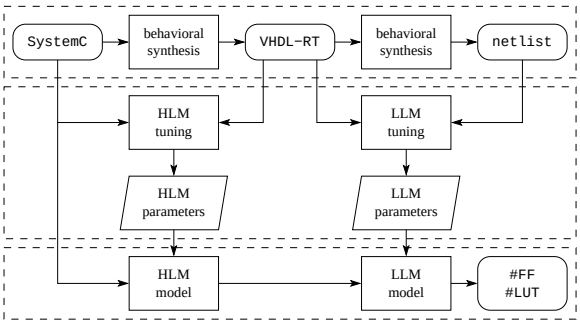

Figure 1: Methodology structure

The estimators for HL models are both algorithmic and algebraic; LL estimators are only algebraic.

3. MODELS, VARIABLES, PARAMETERS

The *High-Level model* starts from the analysis of the SystemC description and estimates a number of intermediate variables to be then fed to the LL model. At this level the

relevant figures are:

#CS, #CI, #CO	Number of control steps, inputs, outputs
DREGS	Number and size of data registers
MUXS	Number and size of multiplexers
OPS	Number, type and size of operators

The estimation is performed as shortly described below (details in [1]). First, each method in the source code is analyzed sequentially to determine the number of control steps #CS of the FSM resulting from behavioral synthesis (explicit CDFG construction is avoided). The result is the identification of a certain number of cut-points corresponding to the beginning of a new control state. The cut-points are determined either directly (wait() statements) or indirectly, based on the type of construct (loops, conditionals, ...). Concurrently, the number of control inputs #CI can also be derived; in particular, they can be easily approximated with the overall number of control constructs (loops, conditionals, function calls) found in each method. Registers belonging to the datapath (DREGS) can be divided into two sets: output registers and internal registers. The number and size of the former are directly related to the number and size of output ports in all modules, while the number and size of the latter depend both on the explicit signals (local variables) and implicit signals introduced by the synthesis algorithm to implement the loops. A combination of common subexpression elimination and lifeness analysis of all such variables leads to the estimate of DREGS. The number, type and size of the operators (OPS) are determined by analyzing the source code, already divided into the previously defined control steps and accounting for mutual exclusion and possible reuses. The algorithm used to this purpose mimics a simple allocation and binding procedure. A subsequent analysis on the set of operators associated to all control steps leads to the estimation of the number and size of multiplexers MUXS. To simplify this phase, only worst-case solutions are considered. Finally, the number of control outputs #CO can be estimated based on #CI, DREGS and MUXS by a statistical analysis yielding a multilinear relation.

The *Low-Level model* combines the high-level figures to determine the number of LUTs and FFs necessary for the final implementation of the system. In particular the following contributions are identified:

$\#FF_{FSM}$	Number of FFs for FSM registers
$\#FF_{DREGS}$	Number of FFs for data registers
#FF	Overall number of FFs
$\#LUT_{DREGS}$	Number of LUTs for data registers
$\#LUT_{FSM}$	Number of LUTs for FSM registers
$\#LUT_{MUXS}$	Number of LUTs for Multiplexers
$\#LUT_{OPS}$	Number of LUTs for operators
$\#LUT_{GLUE}$	Number of LUTs for glue logic
#LUT	Overall number of LUTs

The estimation of the number of flip-flops is simple:

$$\#FF_{FSM} = \lceil \log_2 \#CS \rceil \quad (6)$$

$$\#FF_{DREGS} = \sum_{R \in DREGS} w(R) \quad (7)$$

Equation (6) refers to a binary encoding of the state vector but can be easily adapted to other encoding schemes and the function $w()$ in Equation (7) returns the bit-width of a specific register in the set DREGS. As far as lookup tables are concerned, let considered those dedicated to the reset logic of the flip-flops:

$$\#LUT_{DREGS} = \sum_{R \in DREGS} w(R) \cdot reset(R) \quad (8)$$

where the function $reset()$ returns 1 when the register is subject to global reset and 0 otherwise. This can be determined by locating all the output ports written (write() method) before the beginning of the main loop of each method. The number of LUTs taken by combinatorial logic belonging to the control unit is:

$$\#LUT_{FSM} = \lceil 1.99 \cdot \#CS - 0.24 \cdot \#CI + 1.50 \cdot \#CO - 9.97 \rceil \quad (9)$$

The area required for the operators depends on the type and size of each of them. Relations for some operators can be found in literature [12] and easily converted from gates or equivalent-gates to LUTs. For each multiplexer M, two parameters must be considered: the number of inputs $in(M)$ and the bit-width $w(M)$. The number of LUTs is then given by:

$$\#LUT_{MUXS} = \sum_{M \in MUXS} \lceil (0.68 \cdot in(M) - 0.14) \cdot w(M) \rceil \quad (10)$$

These last relations express the overall area of combinatorial logic but neglect the contribution related to glue logic. Though such contribution cannot be derived from analytical considerations starting from information available at source level, it is reasonable to suppose that the larger the design, the more glue logic is required. A statistical analysis confirmed this hypothesis and led to the equation:

$$\#LUT_{GLUE} = 0.19(\#LUT_{FSM} + \#LUT_{OPS} + \#LUT_{MUXS} + \#LUT_{DREGS}) \quad (11)$$

The overall number of LUTs is thus the sum of all contributions. The next two sections will describe the procedure and the experimental setup adopted for model tuning and the consequent results obtained on some complex examples.

4. TUNING

Model tuning begins by identifying all the syntactic and semantic information (number of operators, variable sizes, loop nesting, code size, ...) that can be extracted from both the SystemC description and the resulting RTL-VHDL code. An accurate black-box analysis of the behavior of the SystemC compiler has driven the identification of the models and their parameters. The models parameters are first identified by considering as input the complete set of such information. A second step allowed pruning the initial set of parameters by removing: not *exportable* parameters, hardly *identifiable* parameters and statistically *negligible* parameters, in this order. A parameter is exportable when a good correlation can be found between its measure at SystemC level and at RTL-VHDL level. A parameter is hardly identifiable when the syntactic and/or semantic analysis required on the source code is too complex (e.g. algorithms very similar to those used for synthesis). The reduced set of parameters resulting from these first two steps has been subjected to principal component analysis in order to further shrink the set of parameters to be eventually considered that are highly independent from the synthesis tools. Whenever the synthesis and/or optimization algorithms change, a smooth degradation of the estimation accuracy is to be expected. Such loss of accuracy can be easily compensated by automatically retuning the models. The tuning process considered the 20 SystemC designs listed in Table 1. The parameters values have been extracted following a least-square approach. The high errors that, in a few cases, affect the estimates have a twofold motivation: the high level of abstraction and the fact that smaller benchmarks are better synthesized than larger ones. For all the parameters described in Section 3, a

thorough statistical analysis has been performed. At high-level the worst-case corresponds to #CO which has an average error around 25% and a correlation 0.95. At low-level the worst-case is $\#LUT_{FSM}$ with an average 30% error. All parameters, on the other hand have an average correlation of 0.98. The analysis on all parameters shows that the critical point is glue logic with errors as high as 120% and low correlation.

	Actual		Estimated		Error (%)	
Design	#LUT	#FF	#LUT	#FF	#LUT	#FF
SH16	56	16	75	10	34	38
CONIB	77	37	55	27	-29	27
FSM	86	7	144	7	67	0
SSUM	90	44	124	52	38	-18
DIS	110	45	128	44	16	2
WHEREN	157	61	162	56	3	8
CMUL	159	102	210	102	32	0
CNT16	189	55	124	52	-34	5
REC	217	28	173	29	-20	-4
PB	258	73	350	74	36	-1
POW8	297	125	357	100	20	20
CU	338	103	408	93	21	10
HS	403	106	593	103	47	3
CMOD	541	166	637	166	18	0
MA	721	221	719	221	0	0
FIFO	816	173	544	172	-33	1
CONBHO	928	133	939	119	1	11
FIR	1339	215	727	216	-46	0
DESKEY	1447	251	1026	219	-29	13
DESENC	1434	307	1033	299	-28	3

Table 1: Tuning results

5. VALIDATION

The identified models and parameters have been applied to a new set of designs (serial/parallel converter, statistical data analysis, Manchester encoding, Peterson ALFSR, 32-bit cyclic redundancy check, triple DES encoder/decoder) with different complexity. Table 2 reports the number of LUTs and the number of FFs, both actual and estimated, and the relative estimation error. Errors can be further reduced leaving the methodology unchanged but retuning it with a larger and more uniform set of designs and/or extracting more than one set of parameters specifically tailored for particular classes of designs. Decomposing the overall estimators for #LUT into their basic components and following the same procedure adopted in the tuning phase leads to the results in Table 3. The weighted error affecting the estimated number of LUTs dedicated to the implementation of the glue logic is significantly higher than those of all others. This is due to two main facts. On one hand, it is intrinsically complex to associate the glue logic to its origin in a high-level specification that concentrates on the functional and algorithmic behavior, leaving all the details to the synthesis process. This is the reason behind the high absolute error (59%) shown in Table 3. On the other hand, for the specific designs considered in this work, there is a mismatch on the expected weight of the glue logic in the tuning set and in the validation set. The coefficient 0.19 of equation (11) suggests that glue logic accounts, on average, for a fraction of 0.19/(1.0+0.19) of the whole design, i.e. approximately 16%. In the validation set, on the other hand, the glue logic requires as much as 31% of the overall combinatorial resources confirming that a more accurate tuning of the model parameters based on a larger set of benchmarks has to be performed.

	Actual		Estimated		Error (%)	
Design	#LUT	#FF	#LUT	#FF	#LUT	#FF
TR	104	53	118	53	14	0
HI	189	55	206	52	9	-5
ME	213	68	189	63	-11	-7
P	1359	200	1114	187	-18	-7
CRC	2799	180	2426	175	-13	-3
DES	5557	583	3518	460	-37	-21

Table 2: Validation results

Estimator	Weight	Absolute Error	Weighted Error
$\#LUT_{FSM}$	0.07	13%	9.0%
$\#LUT_{OPS}$	0.06	3%	2.0%
$\#LUT_{GLUE}$	0.31	59%	18.3%
$\#LUT_{MUXS}$	0.53	9%	4.8%
$\#LUT_{DREGS}$	0.03	68%	2.7%

Table 3: Estimators analysis

6. CONCLUSIONS

The paper presented a two-level model to estimate the area of a complex SystemC design. The model is supposed to be described in behavioral style and the area is expressed in terms of number of flip-flops and look-up tables. The results presented refer explicitly to a specific toolchain (Synopsys SystemC Compiler and Mentor Graphics Leonardo Spectrum) and a specific target device (Xilinx VirtexII-Pro) but the structure of the methodology allows semi-automated retargeting towards different combinations. The proposed framework is intended for fast area estimation with the explicit goal of design space exploration within a co-design environment. The overall accuracy of the models and estimators is more than satisfactory under such a point of view. Furthermore, an academic toolset supporting such and others (timing, power) metrics and allowing partitioning of hardware/software systems is currently being developed.

7. REFERENCES

[1] P. Micheli and L. Zampella,"Area Estiamtors for FPGA Behavioral SystemC Designs," *MS Thesis, Politecnico di Milano,* Italy, 2003.

[2] "Xilinx Home Page," Xilinx, *http://www.xilinx.com.*

[3] "SystemC 2.0 User's Guide,"*http://www.systemc.org.*

[4] "CoCentric SystemC Compiler, Behavioral User and Modelling Guide," Synopsys, *http://www.synopsys.com.*

[5] P. Bjurèus *et al*, "FPGA Resource and Timing Estimation from Matlab Execution Traces," *Proc. of CODES,* pp. 31–36, 2002.

[6] A. Nayak *et al*, "Accurate Area and Delay Estimators for FPGA," *Proc. of DATE,* pp. 862–869, 2002.

[7] P. Bilavarn *et al*, "Area Time Power Estimation for FPGA Based Design at Behavioral Level," *Proc. of ICECS,* pp. 524–527, 2000.

[8] C. Menn *et al*, "Controller Estimation for FPGA Target Architectures During High-Level Synthesis," *Proc. of ISSS,* pp. 56–61, 2002.

[9] F.J. Kurdahi and M. Xu, "Area and Timing Estimation for Look-up Table Based FPGAs," *Proc. of DATE,* pp. 151–157, 1996.

[10] D. Kulkarni *et al*, "Fast Area Estimation to Support Compiler Optimizations in FPGA-Based Reconfigurable Systems," *Proc. of FPCCM,* pp. 239–247, 2002.

[11] C. Brandolese *et al*, "Static power modeling of 32-bit microprocessors," *Transactions on CAD*, pp. 1306–1316, Vol. 21, 2002.

[12] K. Hwang, "Computer arithmetics," *John Wiley & Sons*, New York, 1979.