DESIGN ASSISTANT: AN EXPERT TOOL FOR ASIC DESIGN

Jean-Marc Bournazel - Jacques-Olivier Piednoir

VLSI Technology - European Research Center
HB1 Les Taissounières - 06560 Valbonne FRANCE

Abstract

The `DesignAssistant` is an expert tool used for high-level "what-if" considerations from the very early stage of a design up to the the point where the design capture is complete. It works on a high level structural representation of a design so that it can be used before any logic is captured and supports a real top-down design approach with a refinement mechanism. Its goal is to answer questions such as: How big will this circuit be? What will the power dissipation be? Will it fit on a Gate-Array? Which package should I use? Should the design use one or two chips? These questions are very critical in the feasability study of a design. They need to be answered at the very beginning of the design. The tool, a collection of designers' expertise, has been built on the top of a Prolog expert system containing all the technical data and estimation rules, and providing a powerful and flexible environment.

1 - Design Analysis: A Real Scenario

The `DesignAssistant` has been used to perform the feasability study and to analyze the design of a 20 MHz 16x16 Discrete Cosine Transfrom Integrated Circuit. The DCT chip consists of a dedicated, pipelined processor cell with additional circuitry for operand storage and memory addressing [`Smith 89`]. The goal was to analyze the implementation alternatives during the early stages of design, before the logic was captured.

1.1 - High level description

The design was described at a high structural level as a set of interconnected blocks. Four types of blocks were used:

Block of gates: The schematic of the block has not been captured yet but its equivalent number of gates is estimated from past experience. The input bank, the output bank, the address and the clock generator were specified this way.

List of cells: The block is defined as a list of cells from VLSI libraries, or TTL parts each with an associated count. The processor array was described this way.

Compiler's instance: A cell containing the parameters for the compiler can directly be instanced. The RAMs were specified this way.

External instance: It represents the connections to the external world.

This description was captured using the `LogicAssistant` [`VLSI-LA`], a high level front-end tool. *Figure 1* shows the structure of the chip as it appears in the front end tool.

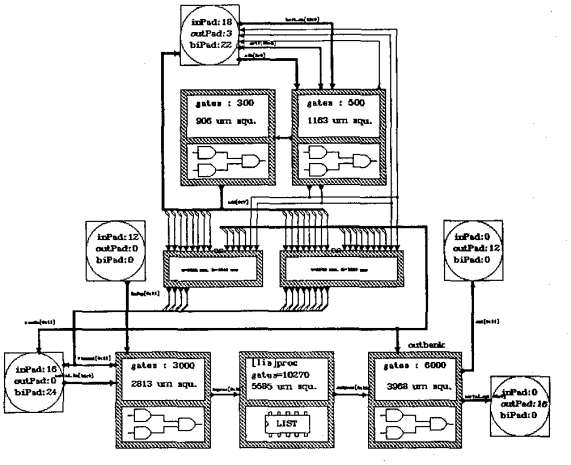

Figure 1 : Chip's block diagram.

It is not necessary to describe in detail the connectivity of the blocks, but only the main busses which will affect the chip layout size. This high level description can refer to blocks of gates, enumerated lists, compiler instances as shown in the example but also instances of schematic, netlists, megacells and physical blocks.

Each instance holds parameters called *attributes* which are used to specify inputs to the estimates such as the

operating frequency, the activity factor or the average fanout. Some parameters can also be set at the chip. level, like the list of alternative libraries, or the capacitive load of the pads.

1.2 - Estimates and results

The `DesignAssistant` supplies commands to estimate each instance separately or to estimate the entire chip. The estimates of each instance are stored on the instance's attributes. Estimates can be displayed on each instance or for each chip.

accept | add | del | move

name	value
cellname	[lis]proc
chip	chip_1
srmodel	default
frequency	55
activity	40
fanout	1
technology	cmn16
size	5585 um squ.
gates	10270
power	Low: 1142 Typical: 1192 High: 1242 mW

Figure 2 : Instance attributes

accept

alternative	
name	chip_1
padload	50
coremhz	20
padmhz	10
inputs	46
outputs	77
packagetypes	pdip● jpqfp● plcc● llccc● ldccc● sb● ppga● cpga●
libraries	vsc10● vsc120● vsc300● vsc320● vgt200● vgt300●
recommended vsss	4
recommended vdds	4
technology	vsc120
floorplan	approximation
density	core limited (plp)
standard cell area	8859 um squ.
standard cell gates	20070
megacell area	2625 um squ.
core x	8642 microns (340 mils)
core y	8642 microns (340 mils)
core+pads x	9977 microns (393 mils)
core+pads y	9977 microns (393 mils)
core+pads+scribe x	10231 microns (403 mils)
core+pads+scribe y	10231 microns (403 mils)
power	Low: 2090 Typical: 2407 High: 2719 mW
package 1	23-80010: 144 pin plastic pin grid array
package 2	23-70020: 172 pin ceramic pin grid array

Figure 3 : Chip estimates

The table in *Figure 4* lists the estimated size for each instance along with the real sizes measured after the design has been captured and the layout has been performed. The sizes are in microns per side.

	PROC	INBANK	OUTBANK	CHIP
DA ESTIMATE µ/side	5585	2813	3968	9977
SIZE µ/side	5152	2822	3651	10048
DX x DY µ	8131 x 3263	1908 x 4174	2740 x 4867	10000x11000
ACCURACY %	14.90	0.64	15.34	1.43

Figure 4 : DA estimates Vs real sizes

The discrepancy in the estimation of the processor array is caused by its very regular structure which leads to a routing complexity lower than for an average random logic.

1.3 - Design Partitioning and Alternatives

Since the type of RAMs used in the design could not be placed on a gate-array, the cell based alternative was the only one investigated by the `DesignAssistant`. To get more alternatives, we can assign the RAMs off-chip. The result of the analysis shows now three implementation choices: Two using 1 micron and 1.5 micron gate-array and one in cell based. The tool takes into account the extra number of pins required by the partitioning. *Figure 5* shows the estimations for the 1 micron gate-array alternative.

accept

alternative	1 2
name	chip_1
padload	50
coremhz	20
padmhz	10
inputs	46
outputs	77
packagetypes	pdip● jpqfp● plcc● llccc● ldccc● sb● ppga● cpga●
libraries	vsc10● vsc120● vsc300● vsc320● vgt200● vgt300●
recommended vsss	4
recommended vdds	4
technology	vgt300
floorplan	vgt300077 (1 u vgt300 array)
core+pads x	8521 microns (335 mils)
core+pads y	8521 microns (335 mils)
core+pads+scribe x	8775 microns (345 mils)
core+pads+scribe y	8775 microns (345 mils)
occupancy	.757432% of max occupancy
sites (Cells)	52790
sites (Blocks)	0
power	Low: 2183 Typical: 2366 High: 2548 mW
package 1	23-80006: 144 pin plastic pin grid array
package 2	23-70015: 132 pin ceramic pin grid array

Figure 5 : Estimation for a 1 micron gate-array

For each alternative, the output of the analysis includes estimated size for the different part of the chip, estimates of the power dissipation, the selection of a package in each package family, and a recommended number of power pads. When gate-array implementation are possible, the tool will choose the smallest base fitting the design in each gate-array family.

1.4 - Refinement - Design Capture

The `DesignAssistant` supports a top-down design methodology and can be used throughout the capture of the design. It is closely coupled with the `LogicAssistant`, the schematic entry tool. This allows users to run new estimations on an instance or a chip after its logic capture. For instance, *Figure 6* shows the new estimates of the DCT chip output bank after its logic has been captured.

The accuracy on the instance estimate is now of 1%. The `DesignAssistant` can be called to get estimates on any sub-part of the design, allowing design trade-offs throughout the design process.

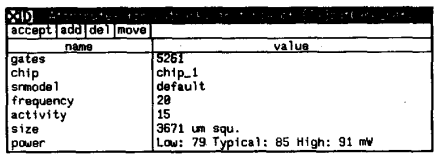

Figure 6 : New estimates of the output bank.

The real chip size after the whole schematic has been captured, placed and routed was of 1x1.1 cm. The accuracy of the early estimate was 11 %.

2 - Implementation of the Design Assistant

2.1 - Implementation Architecture

The `DesignAssistant` architecture has been designed to offer a maximum flexibility both to users and to designers wishing to understand, or modify the estimation rules. *Figure 7* shows the architecture of the tool.

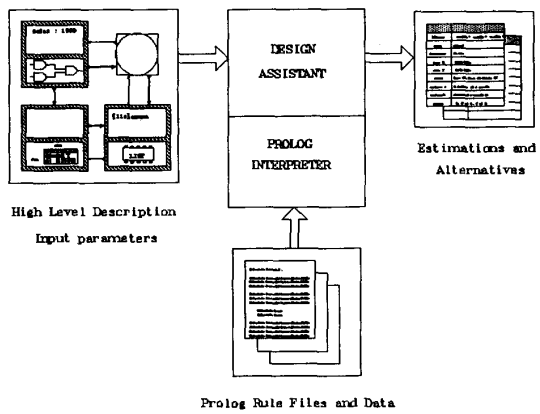

Figure 7 : Implementation of the Design Assistant

The Prolog Interpreter is a Mainsail [`Mainsail`] implementation of the Edinburgh Prolog [`Prolog`] completely independent of the `DesignAssistant` itself. However, all the technical data and estimation rules are written in Prolog files all of which are given to the user with the application. They can be accessed, modified or enhanced at any time without restriction.

The `DesignAssistant` code itself is a black box containing the core of the application, but where no technical data nor estimation rules are stored. This core, written in Mainsail, can be seen as a filter taking the high level description of a design, and displaying the results of Prolog computations.

When opening the tool, a Prolog session is started. The application asks Prolog to load the rule file. The role of the core application is then to format the input and send it to the Prolog system which processes the information and returns the results to the DA core. All the estimations, the calculations or the alternatives investigations are performed within the Prolog rules. The DA core, when receiving results from Prolog, stores and displays them in tables, graphical outputs or text files.

2.2 - Prolog Environment

An Expert System is a perfect framework for implementing the `DesignAssistant`. Prolog provides the powerful and flexible environment the tool requires with facilities such as its inference mechanism, the run-time modifications, data base queries, or trace mode.

At any time when using the application, the user can go into the Prolog system environment, ask the value of some data, change them if necessary, follow an estimation step by step, modify a rule and process the estimations again, save information, going back and forth from one world to the other. An interface has been implemented between the Mainsail world and Prolog. Facts and requests can be sent to Prolog and the results passed back to Mainsail.

2.3 - Prolog estimation data

Three types of data are stored in the Prolog rule files:

Technical data: They list all the existing cells of all the available libraries with the relating technical information such as width, number of terminals or power consumption. They also contain packages and gate-array bases characteristics.
Constants: Some technical constants, the values of which are based on the expertise of designers or the characterictics of each library, are defined such as average fanout, node capacitance or scribe line.
DA Interface parameters: Some Prolog rules define characteristics of the DA/Prolog interface. For instance the list of the alternative libraries, or the available technologies are defined in Prolog. The type of results the application expects are based on a rule also defined within Prolog.

2.4 - Prolog estimation rules

All the estimation rules are defined in Prolog. The instances to be estimated are classified in 3 types:

Layout object: Its layout is known. The size estimation is obvious. The power estimation may be based on the size of the layout or on the relating technical data.

Compiled instance: Each VLSI compiler like the RAM compiler or the data-path compiler comes with an estimator. It performs the size and power dissipation estimation and computes the number of gate without compiling the instance. This Mainsail module is always written by the designer of the compiler himself.

Standard cell block: The size estimation is based on empirical formulae derived from many examples and validated on several designs. It is a function of the cell list and the average fanout. The power estimation, following the method described in the libraries data-book, is a function of the cell list, the average fanout, the average load, the activity factor and the operating frequency.

The chip size estimation, the gate-array base and the package selection are also implemented in Prolog. The chip size estimation is based on the size estimates of the instances and an approximation of the inter-blocks routing. The power estimates is the sum of the instances power estimates. The gate-array selection rule chooses, in each gate-array family, the smallest base fitting a design (number of gates, number of pads). The package selection rule selects, in each package family, the smallest package fitting the chip size and number of pins.

2.5 - Validation Strategy

The designers using the `DesignAssistant` must have a very high level of confidence in its results and estimations. The role of this expert tool is crucial in the technical decisions which may be taken when quoting a design. Tests have to be regularily performed to check the estimations. Any error, even minor, can ruin the credibility of the tool. Therefore, having a very strict validation strategy is quite essential. It basically covers two different domains of action:

Benchmarks:
They address the validation of the estimations. The estimation results given by the tool from the high level description of the design are compared with the actual size of the final physical chip(s) once sent to fabrication. More than 95% of the designs have an error margin below 10%. The remaining 5%, well identified, can be addressed with other methods.

DA Expert Teams:
It addresses the validation of the technical data stored in the Prolog data base and the coordination of the application with other tools. A network of designers and developers have been set up within the company whose function is to constantly use the tool and check that the technical data (libraries, packages, techincal constants) and the estimations rules (cell compilers estimators, calculation methods, results) are always perfectly up-to-date. This is a quite new approach which insures that the tool is constantly up-to-date and reflects the present expertise of a maximum of engineers within the company.

Conclusion

The `DesignAssistant` is a powerful tool that can collect and formalize the tremendous amount of expertise of the design engineers. An expert system to capture this expertise provides a perfect environment to reach that goal. It offers many utilizations to the user: First, it is a `quote tool`: A user can enter the basic high level description of his design, start working on the fundamental partitioning and get in less than one hour all the technical alternatives and estimations required for the feasability and the price study of his design. This is also a `design tool` offering, with its refinement mechanism, a real top-down design strategy from the high level description to the final implementation. Estimations can be performed along the design capture to help the designer in his decisions, and to provide him with diagrams and results for the documentation of his project. The `DesignAssistant` can also be used as an `expert data base` collecting all the technical information available in the books or the expertise of designers. Finally this is also a powerful `training tool` through which new VLSI designers can learn and understand the state-of-the-art of senior designers expertise.

References

[`Mainsail`] Mainsail Language Users' Manual
Xidak Inc. Menlo Park - California - USA

[`Prolog`] Edinburgh Prolog Interpreter
Departement of Artificial Intelligence
University of Edinburgh - Scotland

[`Smith 89`] S.G Smith, J.M Rischard
20 MHz 16 x 16 Discrete Cosine Transform IC:
CAD and Architectural Methodology, p 369-378, VLSI'89
Musgrave & Lauther, Elsevier Science Publishers

[`VLSI-LA`] VLSILogicAssistant
VLSI Logic Assistant User's Manual.
VLSI Technology Inc. San Jose - California - USA

Circuit Partitioning and Resynthesis

Sujit Dey[1] **Franc Brglez**[2] **Gershon Kedem**[1]

[1]Department of Computer Science
Duke University, Durham, NC 27706

[2]Microelectronics Center of North Carolina
Research Triangle Park, NC 27709

Abstract

This paper introduces a circuit partioning method based on analysis of reconvergent fanout. We consider a DAG model for a circuit. We define a corolla as a set of overlapping reconvergent fanout regions. We partition the DAG into a set of non-overlapping corollas and use the corollas to resynthesize the circuit. We show that resynthesis of large benchmark circuits consistently reduces transistor pairs and layout area while improving delay and testability.

1 Introduction

There have been two basic approaches to multi-level logic synthesis: 1) algebraic/boolean factoring, starting with a PLA and producing a minimized multi-level circuit, and, 2) synthesis of multi-level network of simple gates into another optimized multi-level network. There are a number of effective algorithms for minimizing, decomposing and factoring PLAs [1,2]. In contrast, when specifications are in the form of a multi-level Boolean network, there is no single effective technique that would minimize the size of such networks. Some of the early techniques used local transformations [3], and identification and removal of redundancies [4], and, more recently, global flow analysis [5], and partitioning [6].

The improved analysis of signal reconvergence has steadily contributed to advances in automatic test generation, fault simulation and testability analysis [7,8]. The notion of signal reconvergence is in part covered with the concept of a dominator [9]; notions that have been developed subsequently include Stem Region [7] and Supergates [8]. While our work builds on the notion and terminology of the stem region analysis, it is aimed at circuit partitioning for resynthesis.

We introduce a new technique to partition a large multi-level circuit, based on an analysis of signal reconvergence. We model the circuit with a DAG whose nodes are the primary inputs, the gates, and the fanout points of a circuit; and whose edges are interconnecting lines, oriented in the direction of the signal flow. The signal reconvergence regions between the fanout points and all other nodes defines a unique set of primitives in our partitions, the petals. We define a corolla as a set of overlapping petals, forming a multi-input, multi-output module. We partition the DAG into a set of non-overlapping corollas.

In this paper, we discuss the application of corolla-based partitioning to improve the synthesis of large multi-level logic circuits. We start by defining stem regions, petals and corollas. Next, we outline the partitioning algorithm, and describe the resynthesis process. We conclude with a section that summarizes results of partitioning and logic resynthesis on a variety of large circuit benchmarks. Most notably, we find that logic resynthesis based on corolla partitioning consistently reduces recovergent fanout branches, transistor pairs and layout area while improving circuit delay and testability. The details of the algorithm and heuristics we have used are available in [10].

2 Stem Regions, Petals and Corollas

We introduce the key concepts used in our partitioning approach beginning with stem regions. We define a petal as the basic primitive of our partitioning, and define our partition, corolla, as a collection of petals. We model the circuit with a graph $G(V, E)$. Each primary input, gate, and fanout point of the circuit is represented by a distinct node in the graph, and each wire in the circuit is represented by an edge in the graph. Since we consider only combinational circuits, the circuit graph is acyclic, and is directed from the primary input side to the primary output side. Figure 1a shows a multi-level combinational circuit consisting of simple gates, and Figure 1b shows the corresponding graph. The shaded nodes in the graph represent the fanout stems to distinguish them from gates and inputs which are drawn as unshaded nodes. For our partitioning approach, we introduce a primary stem region as a subgraph of stem region defined in [7].

Definition 1 (Fanout Stem and Fanout Branches) *A node of the circuit graph with an outdegree greater than one is called a* **fanout stem**, *and its outgoing edges are called its* **fanout branches**.

In the graph of Figure 1b, traversing from the primary output side to the primary input side, y, u, l, m, n, g, h and a are the fanout stems. The fanout branches of the fanout stem m are (m, p) and (m, q).

Definition 2 (Reconvergent fanout stem and Reconvergent node)
Let $s \in V$ be a stem node. If there are more than one disjoint path from s to another node $r \in V$, then s is a **reconvergent fanout stem** *and r is a* **reconvergent node** *for s. If s has no such reconvergent node, then s is a* **non-reconvergent fanout stem**.

In the graph of Figure 1b, u, n, g, h and a are reconvergent fanout stems, while y, l, m are non-reconvergent fanout stems. The reconvergent nodes of reconvergent fanout stem a are i, o, p, q, r and v.