COMBINATIONAL BLOCK GENERATION FOR LIBRARY VALIDATION BENCHMARK CIRCUIT

[1]Mateus V. Gomes, [1]Simone Bavaresco, [2]André I. Reis, [1]Renato P. Ribas

[1]Instituto de Informática – UFRGS, Porto Alegre, Brazil
[2]Nangate Inc., Menlo Park, CA, Herlev, Denmark
{mvngomes, simoneb, rpribas@inf.ufrgs.br}, are@nangate.com

ABSTRACT

The implementation of combinational blocks used in a benchmark for validation of digital cell libraries, applied in standard cell IC design methodology, is proposed in this work. The automatic generation of this kind of circuit becomes is quite important when library-free technology mapping is addressed due to the fact that it is based on virtual libraries, whose original cells are not previously designed and physically verified. This work describes the requirements and techniques used to generate the benchmark combinational blocks.

1. INTRODUCTION

The approach to validate library cells proposed in this work is based on standard cell IC design methodology which represents nowadays the most applied strategy to ASIC design, where the library cells are reused for a great variety of circuits and applications. In the standard cell methodology, the library gates used by a technology mapper can be specified using electrical parameters (also known as library-based mapping) or proprieties such as the number of inputs and series/parallel devices (also known as library-free mapping). The concept of library-free design is based on using a virtual library available through a layout generator instead of using a set of pre-designed cells, like in library-based design. The set of available cells is given, for instance, by a user-defined constraint in the number of transistors in series. As the cells are generated on-the-fly, a virtual library contains a great number of poorly characterized cells when compared to pre-designed standard cell libraries [1][2]. It means that such virtual libraries are, in fact, used at the first time together with the target ASIC.

The validation and physical characterization of the set of cells, included in a library, are usually done through specific structures and benchmark circuits. Such test structures are composed of ring oscillators, delay chains, counters, and others [3][4]. They are generally designed in full custom style, and must be carefully built for a specific process. Benchmark circuits, on the other hand, correspond to different applications and architectures in order to represent real circuits and system blocks, such as purely combinational circuits, finite state machines, arithmetic blocks, and so on. Therefore, a benchmark set consists of a collection of circuits in a common format, which attempt to represent a range of problems for evaluating algorithms and tools within an important problem domain. In principle, if everyone uses the same test cases to evaluate similar tools, it should be straightforward to compare the results. Several benchmark sets are widely used. The following are some of the most important ones: ACM/SIGDA design automation benchmarks, ITC'99 benchmarks, Politecnico di Torino benchmarks [5]. The use of benchmarks for validation and physical characterization of standard cell libraries may lead to two situations: (a) not all cells from the library are used in the benchmark circuit design, and (b) the cells used in this circuit are not stimulated by all possible input combinations, not guaranteeing the complete functionality of these cells.

The combinational blocks generation of the test circuit (benchmark), proposed in [6], are described in this work. This specific benchmark was proposed to test and validate the entire set of cells from a library. All the library cells are present in such benchmark circuit and all possible input combinations are applied at each cell. Timing and power consumption verification is also taken into account. The idea is not only use this circuit for simulation, but indeed it will be prototyped in the same ASIC die representing a kind of 'certification circuit' of the non-pre-designed library used in such ASIC.

The main contribution of this work is the specification of the combinational blocks implementation. These combinational blocks are the main structures of the benchmark circuit and they will be responsible for important benchmark characteristics as the use of all cells from a library under test and for the application of all input combinations in each cell, guaranteeing the validation of the entire digital cell library. The automatic generation of this kind of circuit becomes even more important when library-free technology mapping is addressed due to the fact that it is based on virtual libraries, whose original cells are not previously designed and physically verified.

The benchmark architecture is presented in Section 2. Section 3 presents the procedure for the combinational blocks generation. Experimental results are given in Section 4 and the conclusions are discussed in Section 5.

2. BENCHMARK ARCHITETURE

In terms of logic cell functionality, three main groups may be identified: (1) inverters and buffers; (2) sequential cells; (3) combinational cells. Group (1) is easily verified since such a kind of cell presents only one input signal. Group (2), in turn, has generally a small and limited number of different latches and flip-flops, facilitating its verification. Moreover, in this group the timing performance is usually more important than the logic functionality that is somewhat trivial in the pass and storage modes. In the case of group (3), the number of cells is generally more expressive than the other ones. Moreover, the number of input nodes in these cells makes the functional test a more complex task due to the 2^n different input combinations, being 'n' the number of input nodes.

The most naïve strategy to test the group (3) consists in placing all cells connected in parallel to the same input bus, where all input combinations are applied simultaneously. However, by doing so, input buffering must be considered due to the high node capacitances (great number of inputs connected to the same node). Furthermore, multiplexers should be used at the output signals to reduce the number of output pads. In this approach, timing and power consumption characteristics are not easily verified since the signals propagate only through single cells in testing.

The proposed architecture consists in building combinational blocks that receive an input bus, where all signal combinations are provided, and produce a sequence of output vectors also presenting all possible signal combinations, to be then applied to the next combinational block. The combinational block is illustrated in Fig. 1.

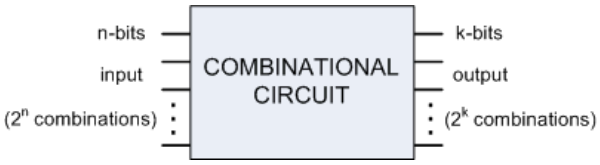

Figure 1. Combinational block illustration.

The cells used in a block should not be re-used in other ones. Moreover, the sequence of input and output signals should not be similar in order to avoid a very simple mapping or even short-circuits between input and output nodes. The implementation of these small combinational blocks guarantees the use of the whole set of combinational cells from the library, being all input vectors applied at the input nodes. The combinational blocks can be cascaded allowing some timing and consumption verification in longer and heterogeneous paths, which can be even evaluated in respect to the power supply variation. Then, once the combinational blocks have been generated, they can be connected in chain, using the output signals from one circuit as inputs of the subsequent block, as shown in Fig. 2.

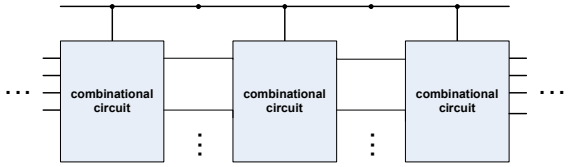

Figure 2. Combinational blocks connected in chain.

3. COMBINATIONAL BLOCKS GENERATION

3.1. Requirements

The input file of the presented tool is a library description in Liberty format. All the present cells are to be put in at least one block. Each cell, except the first one placed in the block, will be tested with some possible input vectors.

Each block has the minimum number of inputs defined by the block cell with the biggest number of inputs.

Two stages are present: (1) the first one consists of the library cells connected to the primary inputs and (2) the second is a set of functions built with some library cells in order to create a vector with all possible combinations to be the input of the next block. Therefore, the output of the 1st block is the input for the 2nd one, and so on.

So, the input of the first block (input for the first stage) is an n-bit vector with 2^n combinations and the output must be a vector with k-bits where the 2^k signals combinations occur, being $n \geq k$. A combinational block is shown in Fig. 3.

In order to respect this condition, 2^k different vectors must be identified at the outputs of the first stage. It is obtained with a minimum number of 'k' cells and a maximum of '2^k-1' cells. The permutation and inversion of cell inputs can be explored to attain a certain number of cells at the first stage.

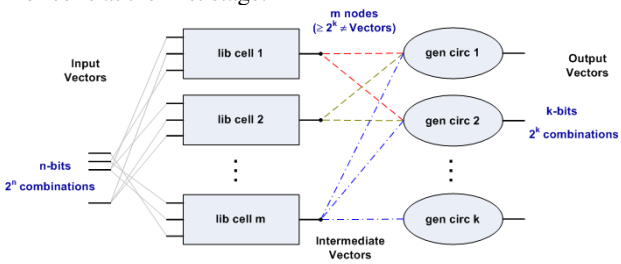

Figure 3. Combinational circuit block diagram.

3.2. Implementation

The generation of the combinational blocks starts with the parsing of the Liberty library file, which is loaded into the tool data structure.

The initial stage consists in sorting the available cells. This choice will influence on the order in which they will be used later. Cells can be sorted through four methods: (a) alphabetical ordering; (b) input number; (c) quantity of minterms; and (d) random ordering.

Once the set of cells is sorted, the construction of the first stage of a block is performed. Each picked cell, except the first one of the block, is tested with all the

combinations of the cell inputs, if no stopping criterion was used regarding the number of combinations, or it is tested with some of the cell inputs in the case of using a stopping criterion. The combinations of cell inputs can consider permutations and inversions.

The goal is to achieve the maximum number of different combinations. One of the biggest problems in the block construction arises at this point.

Being n the number of block inputs and z the number of cell inputs:

(1) For $n=z$, the number of cell inputs combinations is $n!$ (without considering inputs inversion) and $n!\times 2^n$, considering the permutations with possible input inversions.

(2) For $n>z$, the number of cell inputs combinations is $\frac{n!}{(n-z)!}$ (without considering inputs inversion) and, $\frac{n!}{(n-z)!}\times 2^z$ considering the permutations with possible input inversions.

Therefore, exhaustive search can become impracticable for a library with cells that have more than 4 inputs.

To solve such problem, three different stopping criteria have been implemented, besides exhaustive search (which would be up to the user to try). The first one is the number x of tests performed. The cell is put into the block with its best input order after x tests. Another method is stopping when the combinations number is increased by one. If there are currently x different combinations, when $x+1$ is reached tests will stop. And the last method is breaking the search after a certain time (defined by the user) expires. All three methods are important because they guarantee the completion of the blocks building task in a short time.

Whenever the number of distinct output vector combinations desired, i.e. the 2^k different signal combinations are achieved, a block is complete and the construction of another one starts.

Until all cells in library have been used at the first stage of one of the blocks, the generation continues.

The last block in the structure may have to use in the first stage one or more cells that have already been used before, because they may be needed to complete the number of distinct output vectors required.

After the first stage is complete in every block, it is necessary to generate the equations which will generate the k-output vector. If we have an n-input block with m cells, there will be 2^k expected output vectors from the first stage, and 2^m-2^k vectors that are not supposed to happen. The first ones are used to recreate the desired signals, while the latter will be marked as *don't cares*, in order to optimize the generated equations.

3.3. Sample block

Supposing a block with 4-bit input vector in which it is desired to generate a 4-bit output vector. For that, 2^4 distinct signal combinations must be identified at the outputs of the first stage, i.e. it will be necessary at least 4 cells in the first stage of this block. Starting the process, the first cell is picked with its normal inputs (without trying permutations and inversions). When the second cell is picked it is tested with all inputs combinations, i.e. in this blocks generation, no stopping criterion was used regarding the number of combinations of cell inputs to test the cell. The combination of inputs which generate more distinct output signal combinations will be fixed as the input of that cell. At this point, having 2 cells the maximum number of outputs combinations will be 2^2. The third cell is selected and also tested with all inputs combinations. Now, the maximum number of outputs combinations will be 2^3. In the same way, the fourth cell is selected and tested with all inputs combinations. At this point, the maximum number of outputs combinations will be 2^4, which corresponds to the minimal number of signal combinations necessary to generate the 4-bit output vector. If it was possible to achieve this target, the generation of the first stage of this block is completed; otherwise it will be necessary to add other cells until all the 2^4 combinations occur. In Fig. 4 it is shown a sample block.

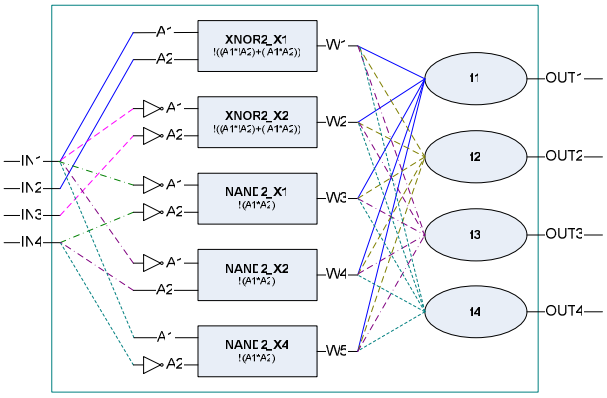

Figure 4. Sample block illustration.

In the generation of this block the four first cells were selected and fixed, but it was not possible to generate the 2^4 output signal combinations. As shown in Table I, it was generated 12 distinct combinations. Thus, the fifth cell was added. Considering this cell $\boldsymbol{C_5}$ with A1(not_IN1), A2(not_IN4) as inputs, still only 12 distinct vectors were identified. Testing $\boldsymbol{C_5}$ with A1(IN1), A2(not_IN4) as inputs, i.e. another inputs combination, 16 distinct vectors were generated as also shown in Table I. At this point, the generation of the first stage of this block is completed.

In the second stage it is necessary to generate the 4 functions which will generate the 4-output vector. Since the first stage of this block has 5 cells, 2^5 vectors could occur. The output vector must have 4 bits, so there will be 2^4 expected vectors from the first stage, and 2^5-2^4 vectors are supposed not happen. The expected vectors are used to recreate the desired signals, while the latter will be marked as *don't cares*, as shown in Table II.

Table I: Sample block with 4 cells and with 5 cells.

I_1	I_2	I_3	I_4	C_1	C_2	C_3	C_4	C_1	C_2	C_3	C_4	C_5
0	0	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	1	1	1	0	1	1	1	0	1
0	0	1	0	1	0	0	1	1	0	0	1	1
0	0	1	1	1	0	1	0	1	0	1	0	1
0	1	0	0	0	1	0	1	0	1	0	1	1
0	1	0	1	0	1	1	0	0	1	1	0	1
0	1	1	0	0	0	0	1	0	0	0	1	1
0	1	1	1	0	0	1	0	0	0	1	0	1
1	0	0	0	**0**	**0**	**1**	**1**	0	0	1	1	0
1	0	0	1	**0**	**0**	**1**	**1**	0	0	1	1	1
1	0	1	0	**0**	**1**	**1**	**1**	0	1	1	1	0
1	0	1	1	**0**	**1**	**1**	**1**	0	1	1	1	1
1	1	0	0	**1**	**0**	**1**	**1**	1	0	1	1	0
1	1	0	1	**1**	**0**	**1**	**1**	1	0	1	1	1
1	1	1	0	**1**	**1**	**1**	**1**	1	1	1	1	0
1	1	1	1	**1**	**1**	**1**	**1**	1	1	1	1	1

4. RESULTS

A commercial library with 90 cells, out of which 64 cells are purely combinational, was used as test vehicle. Eight blocks were generated from the 64 combinational cells, containing 10, 11, 13, 8, 7, 7, 6, and 5 cells in the first stage of each combinational block, respectively. The functionality of such blocks was validated through functional and electrical simulations. Also, the final layout of the circuit composed of the 8 combinational blocks was designed. Another library composed by 220 combinational cells was tested. It generated 38 blocks, containing 8, 6, 6, 6, 7, 5, 5, 7, 7, 6, 5, 6, 7, 5, 7, 6, 6, 5, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 5, 5, 6, 6, 6, 6, 5 and 5 cells in the first stage of each combinational block, respectively. The blocks were generated in 6 minutes, approximately and no stopping criterion was used regarding the number of combinations of cell inputs to test the cells.

5. CONCLUSIONS

The main contribution of this work is the implementation of combinational blocks in order to generate a benchmark circuit capable of validating cell libraries. It is difficult to find a benchmark that uses all cells from a library and when it uses all cells of one, it may probably not use all cells of other library. Thus, it will be necessary exhaustive searches in order to find a benchmark that uses all cells, for every target library. Even when all cells from a library are used in this circuit, they are eventually not stimulated by all possible input combinations, not guaranteeing the complete functionality of these cells. Timing and consumption values can also be extracted from this benchmark circuit.

Table II: Second stage functions.

C_1	C_2	C_3	C_4	C_5	f_1	f_2	f_3	f_4
0	0	0	0	0	x	x	x	x
0	0	0	0	1	x	x	x	x
0	0	0	1	0	x	x	x	x
0	0	0	1	1	0	1	1	0
0	0	1	0	0	x	x	x	x
0	0	1	0	1	0	1	1	1
0	0	1	1	0	1	0	0	0
0	0	1	1	1	1	0	0	1
0	1	0	0	0	x	x	x	x
0	1	0	0	1	x	x	x	x
0	1	0	1	0	x	x	x	x
0	1	0	1	1	0	1	0	0
0	1	1	0	0	x	x	x	x
0	1	1	0	1	0	1	0	1
0	1	1	1	0	1	0	1	0
0	1	1	1	1	1	0	1	1
1	0	0	0	0	x	x	x	x
1	0	0	0	1	x	x	x	x
1	0	0	1	0	x	x	x	x
1	0	0	1	1	0	0	1	0
1	0	1	0	0	x	x	x	X
1	0	1	0	1	0	0	1	1
1	0	1	1	0	1	1	0	0
1	0	1	1	1	1	1	0	1
1	1	0	0	0	x	x	x	X
1	1	0	0	1	x	x	x	X
1	1	0	1	0	x	x	x	X
1	1	0	1	1	0	0	0	0
1	1	1	0	0	x	x	x	X
1	1	1	0	1	0	0	0	1
1	1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1	1

6. REFERENCES

[1] A. Reis, R. Reis and M. Robert, "Topological Parameters for Library Free Technology Mapping", SBCCI, 1998.

[2] A. Reis and R. Reis, "Covering strategies for library free technology mapping", In: ACM International Workshop Logic Synthesis, Lake Tahoe, 1999.

[3] S. Long, "Test Structures for Propagation Delay Measurements on High-Speed Integrated Circuits", IEEE Transactions on Electron Devices, vol. ED-31, N 8, 1984.

[4] M. Bhushan, M.B. Ketchen, S. Polonsky and A. Gattiker, "Ring Oscillator Based Technique for Measuring Variability Statistics", ICMT, 2006.

[5] J.E. Harlow, "Overview of popular benchmark sets", IEEE Design & Test of Computers, vol.17, no.3, p.15-17, 2000.

[6] M.V.N. Gomes, C.A. Silva, S. Bavaresco, G.H. Sartori, L. Rosa Jr., A.I. Reis and R.P. Ribas, "Test Circuit for Functional Verification of Automatically Generated Cell Library", LATW, 2007.