On-Silicon Testbench to Validate Soft Logic Cell Libraries

R. P. Ribas¹, S. Bavareseco¹, M. Lubaszewski¹, A. I. Reis²
¹Instituto de Informática – UFRGS, Av. Bento Gonçalves, 9500, CEP 91501-970, Porto Alegre, Brazil
²Nangate Inc., Menlo Park, CA, 94025, USA
rpribas@inf.ufrgs.br, simoneb@inf.ufrgs.br, luba@ece.ufrgs.br, are@nangate.com

Abstract

This work proposes a validation methodology to silicon-prove a set of logic cells generated by software. It also presents an approach for the automatic design of testbenches to validate the cells in the set.

1. Introduction

Cell-based design is definitely the most applied approach in the ASIC market today. This design approach implies in re-using library cells to build more complex digital circuits. A typical standard cell design environment includes timing and power analysis, as well as automatic assembling of circuit layout through place and route tools. In standard cell libraries, three groups of cells co-exist: (1) inverters/buffers; (2) combinational cells and (3) sequential ones. Mainly due to the large number of different logic functions and driving strength options needed in typical designs, the largest of the three aforementioned groups is the set of combinational logic gates. The handcraft creation of standard cell libraries demands skilled designers and long development times, even when simply dealing with technology migration for the same set of cells. Each cell must be carefully designed and characterized for different input slopes, output loads and design corners. In practice, the high engineering costs of these tasks imposes a limitation on the number of available combinational cells in libraries.

As the technology mapping step in standard IC design flow is based on pre-characterized data of pre-designed cells, the ASIC design space and efficiency turns to be bounded by the library in use. The more cells and drive strength options are available, the larger are the possibilities to improve the circuit design. The enrichment of a library can be done by adding only new drive strengths [1], or through the addition of new functions with standard series/parallel implementation [2], or even with special transistor topologies [3-5]. New topologies can also be considered for in-place optimization (IPO), including in-context cell sizing [6]. The use of extended libraries leads to an optimized fit for particular applications. Due to this added flexibility to the design space, there is an increasing commercial interest for approaches that consider on-the-fly generation of cells, like [1, 6-9]. These approaches are sometimes referred as library-free or as soft-cell or liquid library based.

The main drawback of library-free technology mapping technique is the use of such soft IPs or non-silicon-validated set of cells in the ASIC design. This fact makes conservative customers reluctant in adopting this design technique.

This work proposes a validation methodology to silicon-prove the set of automatically designed logic cells. This validation methodology covers the full functionality of the cells and provides timing and power consumption data useful to validate and fine tune cell data models derived from electrical characterization. Notice that, the circuit speed and consumption estimation tasks, during the design flow, are performed based on cell data from electrical characterization, available in a LibertyTM file, for instance. Indeed, the on-silicon testbench for soft libraries, proposed herein, allows checking whether data used in the performance estimation during the circuit design flow produces good prediction of silicon behavior. A method to automatically generate the validation circuit specific for a given initial set of cells is also proposed.

The remainder of this paper is organized as follows. In Section 2, the motivations for this work are further discussed. The way the library cells are used to build the basic combinational blocks of the testbench is described in Section 3. In Section 4, the circuit architecture is then presented and its operating modes are shown pointing out their goals. The overhead analysis is given in Section 5, and conclusions are outlined in Section 6.

2. Problem Statement and Goal of Work

ASIC designs are usually bounded by the standard cell library in use. For this reason, there are a number of commercial and academic efforts that take advantage of on-the-fly creation of cells. The method proposed by DeDood et al. [1] creates new drive strengths for existing cells (from a starting cell library), in order to save power and reduce delay in a target ASIC. A method that creates dedicated complex cells to reduce delay in applications needing high-speed ASICs is introduced in [6, 8]. Jones et al. [9] proposed a method to optimize a design by changing transistor sizes in the available cells and then redesigns the original library to accommodate these new cells. For advanced technology nodes, research initiatives by leading semiconductor companies are considering the importance of the target library as part of the design
space [10]. The use of more complex gates can reduce the overall number of transistors and provide layouts that are less dense in transistors/mm² but denser in terms of logic/mm². This idea has been pointed as part of a regular layout solution for process variability [11]. The use of such complex gates can also bring advantage to the regular ASIC world. Indeed, several methods to generate efficient transistor networks [3, 5, 7, 11] and to perform technology mapping targeting complex gates [2, 4, 6, 8] have been recently proposed.

Conservative customers feel uncomfortable with the idea of using a 'non silicon-proven' library. The silicon proof should address two main issues. First, it has to guarantee that the models used during the synthesis flow represent adequately the final silicon performance. Second, it has to prove that the cells are reliable under the expected working conditions. This aspect guarantees that the cells present a sufficient number of contacts, do not have latch-up problems, and so on. Notice that this proof has to be done properly by library providers every time a new technology node is available.

The most straightforward way to design a test circuit for a set of cells consists in instantiating all cells using shared primary inputs (for test controllability), while individual output signals could be multiplexed in order to reduce the number of circuit I/O pins, while maintaining test observability. This strategy can obviously provide full functionality validation of each logic gate. However, cell timing information is somewhat difficult to obtain through a circuit with only one-level of logic depth. Additionally, the multiplexers at the output and eventual buffering to compensate the high capacitance on input nodes, as well as the packaging structure, make this approach impractical to measure the timing characteristics of a single gate. That is one of the reason for using ring oscillators and oscillation test structures in delay characterization and testing, respectively [12, 13].

However, the extensive use of ring oscillators would result into a large number of instances of each cell to be validated, which is impractical for libraries with a large number of cells. Another drawback of this library validation approach is that the test for reliability (sufficiency of contacts, electromigration, etc), based on the continuous application of input vectors and the monitoring of the outputs, requires an automatic test equipment (ATE) to apply the vectors and check the results, increasing the test cost. Therefore, an approach that can make these tests viable without additional ATE equipment is strongly desirable. One possibility for this would be to design an application circuit whose functionality could be easily verified. However, it is important to notice that not all the cells available in a library are necessarily used by the technology mapping tool for a given circuit. This fact is shown in Table I, where it is possible to see that out of a set of 64 available gates not all of them were instantiated after a given circuit was mapped, with exception of circuit 'tv80_core', which used all the cells. For these reasons, more efficient testbenches for testing a set of cells may be proposed.

<table>
<thead>
<tr>
<th>Benchmarks*</th>
<th>Number of cells instances</th>
<th>Number of distinct cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>c7552</td>
<td>1,311</td>
<td>41</td>
</tr>
<tr>
<td>i2c_master_top</td>
<td>679</td>
<td>42</td>
</tr>
<tr>
<td>Iu</td>
<td>9,203</td>
<td>43</td>
</tr>
<tr>
<td>mc_top</td>
<td>6,245</td>
<td>58</td>
</tr>
<tr>
<td>tv80_core</td>
<td>5,594</td>
<td>64</td>
</tr>
<tr>
<td>wb_conmax_top</td>
<td>28,089</td>
<td>43</td>
</tr>
</tbody>
</table>


An efficient approach to generate a testbench for testing a set of new cells, possibly created on-the-fly, should cover the following aspects:

1) to ensure complete functionality test for the instantiated cells;
2) to ensure coverage (instantiation) of all the cells to be tested;
3) to allow the verification of the accuracy of the models used in the design process;
4) to provide means to perform long and medium term reliability tests (needed for electromigration, sufficiency of contacts, NBTI degradation, etc) without additional equipment;
5) to have a feasible number of cell instances compared to the set of cells to be tested.

The goal of this work is to propose a methodology to automatically generate testbench circuits for on-silicon soft-library validation meeting the aspects above. To the best of our knowledge, previous approaches fail to cope efficiently with this goal.

In this work, a straightforward and efficient testbench methodology is proposed aiming the validation of an entire set of soft-cells in terms of logic and electrical behavior. The presented solution merges well-established design and test concepts to cope with the five aspects mentioned above. A specific combinational block is built to guarantee the logic coverage (aspect 1) of a sub-set of the cells to be validated, and to provide at the output the same bit vector received at its inputs, allowing thus to cascade long chains with these blocks. The use of several blocks allows to instantiate all the cells (aspect 2). The circuit architecture is then composed of such combinational chain in a ring configuration, synchronized by a register barrier. Both synchronous and asynchronous operating modes provide different features.
for the proposed goals. The ring configuration allows verifying the accuracy of the models, by comparing with the predicted circuit behavior (aspect 3). The oscillation BIST technique is also included in the circuit operation for a wide range of different paths [13], and allows medium and long term tests (aspect 4). In case of an eventual error, the circuit diagnostic is facilitated through an arrangement of multiplexers. Finally, as the ring oscillator is composed of a variety of cells, the number of instances is not very expensive compared to the initial set of cells (aspect 5), as shown in the overhead section.

From a business model point-of-view, the methodology presented herein is useful for the soft-library vendor and to the ASIC designer client. For the vendor, it is quite important to dispose of a physical testbench in order to guarantee the correctness of its EDA environment, as well as to verify the quality of the generated cells in terms of performance and reliability, including design-for-manufacturability issues [14]. This is essential for the continuous improvement of the library generation CAD tool. For the ASIC designer, a circuit that validates all distinct cells created on-the-fly to be used in a specific circuit provides means to exclude that errors on silicon due to the cell generators. If this test circuit is fabricated in the same die of the ASIC, it can act as a kind of "certification circuit" for the soft-library, in different design corners and operating conditions. In this case, the overhead in terms of area and I/O pins is a compromise in fabricating together the test circuit and the ASIC, and the low cost approach presented here is very attractive.

The detailed descriptions of the combinational blocks, overall architecture and operation modes of the testbench are given in next sections.

3. Testbench: the Combinational Blocks

The combinational blocks are built in a way to guarantee the complete and correct logic behavior of all cells included in the soft-library under test. To attain this, each combinational block is composed by two sub-blocks or stages. The first one is built by cells in a single logic depth level, all of them excited by the (shared) primary inputs. This is similar to the straightforward arrangement to allow test controllability and observability, discussed in the previous section, but now including only a small sub-set of cells. These cells, in the first stage, are connected to the block inputs for full logic exercising. The output signals of the cells in the first stage are then used as inputs to the second stage, which recomputes the same bit vector applied at the first sub-block primary inputs. As a result, the primary inputs of the first sub-block are equal to the primary outputs of the second sub-block. The internal interface of the two sub-blocks can be viewed as an intermediate code for which primary inputs are translated and then recovered.

Before detailing each stage of the combinational blocks, the following principles and requirements might be pointed out:

1) all blocks present the same number of input and output nodes, which must be equal or higher than the biggest number of inputs in a single cell;
2) input and output vectors have equal steady state logic values.
3) every cell has to be instantiated at least once in the first stage of a combinational block;
4) the total number of combinational blocks depend on the size of the cells set and also the quantity of cells necessary to compose the first stage of each block;
5) the second stage of the combinational blocks is synthesized taking into account only the cells present in the soft-library to be validated.

To generate the first stage of a block, the whole set of cells is initially ordered according to one of the following criteria: alphabetic order; number of cell inputs; quantity (or rate) of 0s and 1s provided by the logic function; or random order. Once the cells are ordered, they are taken one-by-one to create the circuit in this first stage. Assuming 'n' the number of signals at both the input and output block interface, the goal is to minimize the length of the internal intermediate code in the interface of the two sub-blocks. Notice that the length of the intermediate code corresponds to the number of cells used in the first stage, if single output cells are assumed. The minimization procedure has to choose a number of cells ('m') that produces at least $2^m$ different values at the m-bit output of the first stage. The $2^n$ different intermediate code values (distinct m-bit values) at the output of the first stage represent the minimum required to reconstruct the original $2^n$ input combinations at the n-output signals at the second stage of the block, with a one-to-one correspondence.

The reason to minimize the length of the intermediate code (represented by the number of cells in the first stage) is that the complexity of the second stage of the block is proportional to the size (number of bits) of the intermediate code. Thus, reducing the number of bits in the intermediate code reduces the area overhead of the second stage. One way to perform this minimization is the following. The first selected cell will give a 1-bit length code where the codes ‘0’ and ‘1’ are possible. Next, cells will add one bit each and a given number of distinct vectors. The criterion to select new cells it to choose them to maximize the number of different intermediate vectors at each new instantiation (that adds one bit to the intermediate code). To increase the number
of new vectors in the intermediate code, the input signals of each cell can be permuted (P) and/or negated (N), and the best option is then selected. Cells that are included in a first stage sub-block will not be considered in the generation of subsequent ones. Cells that do not increase the number of different intermediate vectors are not instantiated and remain to be used by a subsequent first stage sub-block generation. The combinational block generation stops when there is no unused cell remaining. Already used cells can be reused during the generation of the last block, as few options of unused cells are available. Consider a soft-library where only single-output cells are available and where the validation circuit is composed of n-bit blocks (n-inputs and n-outputs). The minimum number of cells at the first stage is equal to 'n', as the intermediate code cannot have a smaller length than the input/output codes for the block. The worst case is composed by maximum 2^n-1 cells, as the first cell gives two distinct vectors and subsequent cells are required to introduce at least one new vector.

To illustrate the construction of the first stage, consider a 3-input combinational block, as shown in Fig. 1. The intermediate code composed of 'Wi' bits requires 2^3 distinct vectors, necessary to rebuild the 3-bits input vectors at the 3-bits block outputs. This requirement was not attained with the three first selected cells, as illustrated in Table II (second column). Thus, at least one additional cell C4 must be added to the first stage. The third and the fourth columns in Table II show two alternative C4 input connections where only one provides the 2^3 necessary distinct vectors. At this point, the first stage of this combinational block is concluded.

In this work, the generation of the first stage of each block was automated by using a specific CAD tool, developed in Java platform. The second stage, in turn, is synthesized with standard technology mapping engines.

The construction of the combinational blocks has been exercised for different sets of cells in order to evaluate the complexity (size) of the circuits generated with this methodology. Fig. 2 shows the block building considering a set of 208 cells, with up to 7 inputs, obtained from the ‘genlib_44-6’ library [15]. Only logically distinct functions are included in this set. The possibility of applying permutations (P) and negations (N), or both (NP), at the inputs during the cell instantiation was also evaluated.

In Fig. 2a, the X-axis represents the number of blocks generated and the Y-axis shows the number of cells in the first stage of each block, which is equivalent to the length of the intermediate code. As it can be observed, the possibility to consider the permutation and negation (NP) of the cell inputs resulted in smaller circuits at the first stage. Consequently, more blocks are necessary to instantiate all cells. The total number of instances with P-variants was 10,538 (232 for the first stages + 10,306 for second stages), while it decreased to 8,074 (214 + 7,860) with N-variants and to 7,680 (208 + 7,472) with NP-variants. The conclusion is that the extra flexibility given by the use of NP-variants allows obtaining a more efficient intermediate code, which minimizes the overall number of instances. Additionally, no cells were repeated to conclude the last block in the NP-version; while in the N- and P-versions, 6 and 24 cells were re-used to finish this task, respectively. The computation time for block generation was 25,087 sec for the NP variants, 15 sec for the N variants and 432 sec for the P variants, by using a 1.8GHz dual-core processor, with 4Gb SRAM and 1Mb cache.

The number of cells in the second stage can be observed in Fig. 2b. It was mapped using the ABC tool [15]. This stage can be considered as the overhead to re-create the block input vectors from the intermediate code. Notice that increasing the quantity of cells at the first stage (length of the intermediate code) increases significantly the size of the second stage, generating an area overhead. This is the motivation to reduce the number of cells while generating the first stage. Also, the number of input/output pins in the block should be the minimum required, which is the largest number of inputs in a single cell in the set. This requirement is necessary to provide test controllability for the cell.

Figure 1 - 3-bits combinational block: example.

The second stage re-creates the input vectors at the output of the block, from the intermediate codes represented at ‘Wi’. The output signals of the first stage are used as inputs to the second stage, as shown in Table III. Since the length of the intermediate code can be larger than the length of I/O codes, some ‘Wi’ combinations will never occur. This way, don’t cares are used to optimize the synthesis of the second stage.
Table II – Construction of the first stage of the combinational block in Fig. 1.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>3 cells</th>
<th>( C_4 )</th>
<th>( C_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{IN}(3) )</td>
<td>( \text{IN}(2) )</td>
<td>( \text{IN}(1) )</td>
<td>( w_3 )</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 1</td>
<td>1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 1 0</td>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1 0</td>
<td>1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 1 1</td>
<td>0 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 0 1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

Table III - Truth table used to synthesize the second stage of the combinational block, illustrated in Fig. 1 and Table II.

<table>
<thead>
<tr>
<th>( \text{IN}(3) )</th>
<th>( \text{IN}(2) )</th>
<th>( \text{IN}(1) )</th>
<th>( W_4 )</th>
<th>( W_3 )</th>
<th>( W_2 )</th>
<th>( W_1 )</th>
<th>( \text{Out}(3) )</th>
<th>( \text{Out}(2) )</th>
<th>( \text{Out}(1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No vector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>No vector</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>No vector</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>No vector</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>No vector</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>No vector</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>No vector</td>
<td>No vector</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No vector</td>
<td>No vector</td>
</tr>
</tbody>
</table>

Figure 2 - Number of cells per block according to the permutation (P) and negation (N), or both (NP) procedures in the logic gate inputs, considering a set of 208 cells, with up to 7 inputs: (a) first and (b) second stage complexity.
By using Mentor Graphics DFT tools [16], it was verified that all single faults in the cells of the first stage are observable at the output of the block, for combinational blocks designed as described above. Some faults in the second stage may not be observable at the output of the block. However, since the full fault coverage in the first stage is sufficient to prove the correctness of the set of blocks designed as described above. Some faults in the second stage may not be observable at the output of the block, for combinational blocks described in the previous section, guarantee the functional validation of the entire architecture. Long paths can be built, for instance, by cascading the combinational blocks in chain configuration. This way, the primary input values should be observed at the output of chain in the case of fault free behavior.

The global circuit architecture is presented in Fig. 3. To provide a sequence of test vectors with minimum external intervention, the signals at the end of the chain are reconnected to the primary inputs. A register barrier, composed by D-type flip-flops, is added to the feedback path to avoid racing. An adder is available to increment the binary vector and make the circuit act as a counter to modify the feedback signals and provide the chain input variation. The adder can perform sums by ‘K’ allowing other than just a counting 1-by-1 operation, allowing thus different vector transitions that are important to check charging and discharging conditions at internal nodes intra- and inter-cells.

The basic architecture is slightly modified by adding a comparator and multiplexers to allow different operating modes which provide distinct forms of data evaluation. Notice that, the aim of this circuit is to validate the full functionality of the entire set of cells, as well as evaluate the accuracy of the electrical characterization values of the cells (timing and power dissipation data) through the correlation of the static timing analysis (STA) and power analysis with experimental measures.

The circuit operating modes and their individual contributions to meet the five requirements described in Section 2 are discussed in detail in the next sub-sections. They are:

- synchronous mode;
- asynchronous mode;
- oscillation BIST mode;
- diagnosis mode.

**4.1 Synchronous Mode**

In the synchronous operating mode, the register barrier is controlled by an external clock signal, i.e. the ‘Ext_CK’ signal indicated in Fig. 3. The adder is used to increment the vector in the ring, acting as a synchronous ‘+K’ counter. The right behavior of the counting sequence demonstrates the correct functionality of the combinational blocks and, consequently, the whole set of cells under test.

The maximum operation frequency of the circuit, which indicates the worst case path delay, can be obtained by increasing the clock frequency until the ring counter gives an erroneous result. Such critical path delay will be probably different by changing the increment value ‘K’.

Another significant benefit of the synchronous mode is the evaluation of the power dissipation, including its dynamic and static components. The external control of the clock signal imposes the frequency operation for switching, and the dynamic power consumption can be related to that. The static power, on the other hand, can be measured at low frequencies or even by using an external clock manually controlled. At each new input state in the chain the static consumption can be obtained since such power dissipation component depends strongly on the circuit steady state.

Fig. 4 shows the electrical simulation of the circuit for both synchronous (before 100ns) and asynchronous (after 100ns) modes. According to Fig. 3, the ‘CK’ waveform is the internal clock signal; the ‘ctrl_CK’ signal is the multiplexer control to switch between an external clock signal and the comparator output (asynchronous clock); the ‘In(1)’ to ‘In(4)’ signals represent the 4-bit data in the ring counter; and the ‘I(vdd)’ waveform shows the power
supply current. Notice that in the synchronous mode the supply current allows the identification of dynamic and static dissipation components, while in the asynchronous mode the circuit is always in dynamic operation.

4.2 Asynchronous Mode

In the asynchronous mode, or self-timed ring configuration, the clock signal of the flip-flops is provided by the comparator that checks whether or not the input vector \( In(n..1) \) has already propagated to the end of the chain \( Out(n..1) \) (Fig. 3). When the same vector applied to the circuit inputs get to the end of the chain, the comparator switches from ‘0’ to ‘1’, clocking the register. The new data is stored in the register and passed to the adder. The adder increments the register output and applies the new vector to the chain. At this moment, since \( In(n..1) \) no longer equals \( Out(n..1) \), the comparator output is back to ‘0’ and remains at this state until the new vector propagates through the whole chain of combinational blocks. This behavior can be observed at the right side of Fig. 4.

In this operation mode, just an external signal transition is enough to start the self-timed counting, and the right operation keeps the circuit running. For instance, the starting of the self-timed operation can be achieved by modifying an external adder input bit ‘Ki’. If a cell is defective, the data at the end of the chain will not be equal to the data at the circuit inputs. This way, the comparator will not switch to ‘1’ and the self-timed execution will stop, as verified in Fig. 5.

The self-checking property of the asynchronous mode makes it quite appropriate for functional cell verification with least external intervention. The correct logic operation can be checked by just monitoring the internal clock signal or only one data bit in the loop path.

Timing information can be extracted from complete ‘+K’ counting cycles. This measure represents the average logic path delay since in a self-timed circuit the speed is as fast as possible according to the delay of each transition, or the time to finish a computation. These measures can be used to validate the models if a simulation is done in the same operating mode.

Notice that a set of circuits working continuously in this mode can be used to check the library reliability and robustness against degradations. A continuous operation without errors proves the reliability of the cells against a number of issues like insufficiency of contacts, electromigration [17], negative bias temperature instability (NBTI) [18, 19], or other causes. NBTI is a degradation of PMOS transistors that depends on the amount of time the output of the cell is connected to Vdd; see [18, 19] for details. Notice that the circuits will not need expensive equipments to be monitored, once defects are self monitored, as shown in Fig. 5. Even if the circuits do not fail after being continuously stressed for a long period of time, they can be used to measure performance degradation caused by NBTI.

4.3 Oscillation BIST

In synchronous or asynchronous mode, the same binary value of an i-index input is expected to re-appear at the corresponding i-index output of the testbench circuit. This property is ensured by construction of the combinational blocks. If the \( i_{th} \)-output is directly connected to the \( i_{th} \)-input, the \( i_{th} \)-path is kept in steady state, while closing the feedback loop. However, when the \( i_{th} \)-output is inverted before connecting to \( i_{th} \)-input, a negative polarity logic feedback occurs and the \( i_{th} \)-path oscillates. According to this principle, in oscillation mode the feedback loop is closed such that at least one of the primary inputs of the chain receives the negation of its previous value, as illustrated in Fig. 6.
The bit inversion necessary to ensure the oscillation condition is implemented through the inverter placed at the input of the multiplexer in Fig. 3, indicated by an ‘*’. This multiplexer is depicted in Fig. 7. Note that only one bit from the vector Out(n.1) is selected at a time to provide only one bit oscillation. The other ones are then fixed by the value provided in the input In(n.1). That is, if a single input-output pair is considered, just the associated path will oscillate, and all others are kept in steady state.

Notice that, the internal path followed by the oscillating signal through the chain of combinational blocks will depend on the static binary values to which the steady-state inputs of the first block are set. This feature allows configuring a wide range of different oscillating paths. Fig. 8 shows the electrical simulation of the circuit in oscillation BIST mode. While the bit ‘In(1)’ is kept oscillating, the others are modified at lower frequency to cover all the eight possible combinations. At the end of this exercise, the oscillation is switched to the bit ‘In(3)’. Table IV gives the signal period of the oscillating bit for all different states illustrated in Fig. 8.

The main contribution of the oscillation BIST mode to the validation process is that it makes possible to measure the signal delay propagation through different logic paths involving different cell instances. These delay results are then compared to those obtained from a previous static timing analysis, which took into account cell timing data from characterization (possibly done by electrical simulations). This way it is possible to correlate the models used for design with real silicon behavior.

Notice that, the internal path followed by the oscillating signal through the chain of combinational blocks will depend on the static binary values to which the steady-state inputs of the first block are set. This feature allows configuring a wide range of different oscillating paths. Fig. 8 shows the electrical simulation of the circuit in oscillation BIST mode. While the bit ‘In(1)’ is kept oscillating, the others are modified at lower frequency to cover all the eight possible combinations. At the end of this exercise, the oscillation is switched to the bit ‘In(3)’. Table IV gives the signal period of the oscillating bit for all different states illustrated in Fig. 8.

The main contribution of the oscillation BIST mode to the validation process is that it makes possible to measure the signal delay propagation through different logic paths involving different cell instances. These delay results are then compared to those obtained from a previous static timing analysis, which took into account cell timing data from characterization (possibly done by electrical simulations). This way it is possible to correlate the models used for design with real silicon behavior.
The diagnosis can also be performed in open ring configuration, as the multiplexer used to provide the oscillation BIST mode (see Fig. 7) can also be applied to interrupt the ring configuration without additional circuitry. The open chain mode allows the external control of the signal stimuli that are sent to the combinational blocks chain. It is quite useful for the identification of a faulty cell.

Table V – Electrical simulation data of the diagnosis mode, considering one data bit in oscillation mode.

<table>
<thead>
<tr>
<th>Interval (ns)</th>
<th>Chain configuration</th>
<th>Period (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-35</td>
<td>complete chain</td>
<td>4.07</td>
</tr>
<tr>
<td>35-45</td>
<td>half-chain</td>
<td>2.52</td>
</tr>
<tr>
<td>45-60</td>
<td>one-fourth chain</td>
<td>1.64</td>
</tr>
</tbody>
</table>

5. Overhead Analysis

The testbench circuit proposed here can be integrated in the same die with an ASIC, resulting in an area overhead as illustrated in Fig. 9. Two experiments have been done to evaluate this overhead.

Figure 9 – ‘Certification circuit’ prototyped in the same die of the target ASIC.

In the first experiment, the set of 64 cells previously mentioned in Table I was considered. This set is composed of cells with up to 4 inputs. The number of input/output bits for the generation of combinational blocks was kept at four bits to ensure the controllability of the instantiated cells. Eight combinational blocks were automatically generated to build a circuit having all library cells instantiated in the first stage of blocks. The complete circuit used around 500 instances. The testbench circuit ensures the verification of all 64 cells in the library, with an average of 7.03 instances per cell from the library. Notice that, the ‘tv80_core’ circuit in Table I, which used all 64 cells of the library on its implementation, has 5,597 cell instances. Suppose that the verification circuit generated here was the test vehicle to verify the library used to implement the ‘tv80_core’ circuit. In this case, an overhead of 8% in terms of cell instances would be produced. However, if ‘wb_conmax_top’ was chosen to validate the library, a circuit with 28,089 instances would be produced and yet not all the cells from the library would be instantiated.

The second experiment was the design of a testbench circuit to test a library containing 208 cells, mainly CMOS complex gates, with up to 7 inputs. This example resulted in a testbench composed of around 8,000 instances, to validate the complete set of cells. The number of instances per cell library is around 16, which is still acceptable. This can be considered a very small circuit to test such a huge group of cells once an ASIC requiring such amount of cells can easily have more than 100k instances.

Therefore, the test circuit brings little silicon overhead to the final design, allowing the fabrication embedded with the target ASIC, as illustrated in Fig. 9.

The overhead in terms of I/O pins is described in the following, considering n-bit combinational blocks. The value ‘n’ represents the maximum number of inputs in a single cell to be validated. In this case, the final validation circuit presents:

- ‘n’ input signals – \( K(n..1) \);
- ‘n’ output signals – \( Out(n..1) \);
- one external clock signal;
- one D-type flip-flop reset signal;
- some multiplexer control signals, being that the number can be reduced by using a decoder circuit since some multiplexers are dependent on each others.

Depending on the complexity of the target ASIC, such a pin overhead may prove unaffordable for certification at the chip level. In this case, considering its little silicon overhead, the testbench could be though as a certification vehicle at the wafer-level and be used to validate sets of neighbouring ASICs.

From this analysis, one can conclude that the testbench methodology represents a low cost solution for validating all soft-cells included in an ASIC, and may act as a library certification circuit at chip or wafer-level. In terms of EDA vendor interest, the proposed methodology represents an efficient way to validate soft-libraries by using very compact circuits.

6. Conclusions

The use of automatically generated CMOS logic gates in standard cell IC design flow represents an attractive
perspective for ASIC design quality improvement. Automatic cell generators can be considered as soft IPs and represent the key elements for the library-free technology mapping approach, already proposed in literature and now being adopted by the industry. This methodology leads to an IC design flow based on logic cells which are created on-the-fly by software, which have not been previously validated on silicon yet, until the target ASIC is prototyped. This fact makes conservative customers reluctant in adopting this design technique. This work proposed a validation methodology to silicon prove the set of automatically designed logic cells. The validation covers the full functionality of the cells and provides means to compare cell data models derived from electrical characterization against silicon performance. This methodology allows verifying the reliability of the cell models to be used in the performance estimation during the circuit design. A method to automatically generate the validation circuits from an initial set of cells was also proposed. The method has low area overhead and supports several operation modes. Medium and long term standalone runs allow verifying the circuits for degradation effects like NBTI and electromigration. Self-testing modes to perform medium and long term reliability certification without depending on ATE allow to reduce costs and to increase the number of samples under test.

References