ECCTD'01 - European Conference on Circuit Theory and Design, August 28-31, 2001, Espoo, Finland

Digital Standard Cell Library Migration Using A Genetic Approach

Kenneth Francken* and Georges Gielen*

Abstract—**Digital standard cell libraries are a key element in every modern VLSI design flow. The most important issues are compactness and speed of the cells. Therefore, the performance of these cells and their layout are individually tuned. This job is not only complex, but also very time consuming considering the fact that this is mostly handcrafted work. Ofcourse, this only needs to be done once for every technology. But also in the case where multiple foundries are used, or where different flavors of the same process are used, a new optimized library is needed. Let us also keep in mind that new and smaller feature size technologies are continuously becoming available and that even 'older' processes get tweaked when new equipment is used to increase performance. On the other hand, market pressure demands quick product introductions. It would be beneficiary to have very quickly access to a first version of the new library that can still be tuned afterwards if the need arises. In order to speed up the migration an automated approach is necessary. We therefore present such a methodology for the standard cell sizing based on a genetic algorithm. Since we use a SPICE–level circuit simulator, accurate results w.r.t. the performance are guaranteed. The methodology has been implemented in an easy to use tool with graphical user interface. The description and evaluation statements of the digital cells are standard SPICE netlists that are parameterized.**

I. Introduction

In order to profit from the performance boost allowed by newer technologies, digital standard cell designers need quick access to a standard cell library for the new process to be used. Since the library cells will be instantiated many times, great care has to be taken w.r.t. their performance and their area. To ensure rapid availability of this library an automated approach – as opposed to hand–crafted tweaking – for the library migration seems unavoidable. In this paper we will present such an automated approach for the resizing part of digital standard cells based on a genetic algorithm.

When dealing with circuit synthesis a classification can be made between two common approaches [1]. A first class is the *knowledge–based* approach. The main advantage here is the computational speed. Drawbacks of this method are flexibility and the time needed to develop the knowledge plan. Even if, for our purposes, the topology is fixed and the design targets are invariable in type (not in value), the plan will never be applicable in the same form for all types of technologies. Another possible disadvantage is that the accuracy is proportional to the complexity of the plan. In a second class, the *optimization–based* approach, two subcategories can be found: equation–based optimization – which suffers from similar accuracy limitations – and simulation–based optimization. The big advantages of the latter are its flexibility and accuracy. It has the same accuracy as the simulator normally used in hand–crafted sizing. Here an optimizer iterates over simulations for different values of the device sizes to tune the cell's performance. However, the penalty clearly resides in the large CPU time usage. The continuous increase in computing power alleviates this drawback partially. A detailed overview of optimization techniques for digital circuits can be found in [2].

Our approach is simulation–based and the algorithm guiding the parameter selections is a differential–evolution genetic–based program [3]. An efficient and easy–to–use GUI has been written, enabling straightforward use of the tool. The netlist description of the cells is standard SPICE syntax and the specific performances are also represented in each netlist as measured variables. They are then automatically parsed by the tool. The properties of both source and target technology are specified in an ASCII configuration file. For the migration itself, a user can choose that the performances in the target process can be kept equal to those in the source process or they can be tuned by relaxing some specifications or making them more stringent. Of course, in practice, one will set the specifications – mainly in terms of delays – more stringent for the target technology. By making the netlists parameterizable, our methodology can retarget the given cells quickly to any given technology.

The paper is organized as follows. In section II we discuss how to input data of the source and target technologies, the cell descriptions and some control options. In section III, the program flow is then explained. In section IV an overview of the tool interface is presented as well as two experiments to validate the methodology. Finally, in section V, conclusions are drawn.

*Katholieke Universiteit Leuven,
Dept. of Electrical Engineering, ESAT-MICAS,
Kasteelpark Arenberg 10, B-3001 Leuven–Heverlee, Belgium.
E-mail:kenneth.francken@esat.kuleuven.ac.be,
Tel: +32 (0)16 32 10 76, Fax: +32 (0)16 32 19 75.

II. Overview of user–supplied data

There are a couple of ways in which the user inputs data to the tool. First, some general settings are set in a *circuit configuration file*. Note that the tool can work with different simulators. Secondly, each cell's specifics are defined in a netlist. Then, other – more general – settings can be done at the program level. We will discuss these input methods in more detail below.

A. The circuit configuration file

All basic settings for a library migration are specified in one ASCII configuration file. This file defines the following attributes:

- Circuit simulator settings
  – Simulator (e.g. HSPICE, ELDO, ...)
  – Technology (vendorname_cmos_0u35)
  – Model (bsim3v3, mos_model_9, ...)

- Model type (slow, fast, typical, slowfast, fastslow)
- Simulation type (dc, ac, tran)
- Model path (/path/to/model/files/)
- Simulation options filename
- Circuit path (/path/to/spice/files/)
- Circuit basename
- Simulation output directory (/path/to/output/dir/)

- Circuit simulator parameters
  - Parameter name:#SYMBOL#:value
- Porting technology settings
  - Target technology (vendorname_cmos_0u25)
  - Model (bsim3v3, mos_model_9, ...)
- Porting fixed parameters
  - Parameter name:#SYMBOL#:value

The first category defines the simulator to be used (currently HSPICE and ELDO are supported), the technology of the source process and the model, the simulation type and the appropriate directory paths. In a second category, all parameters are defined together with their values for the source technology process. The following category lists the target technology and model. Finally, in a last category, parameters for the target technology are given that are fixed during the optimization. These parameters have values that can be different from the source technology but need not be optimized (e.g. supply voltage). Parameters listed in the second category and absent from the last are taken as the optimization variables.

B. *The input netlists*

Each cell in the library is described by a standard SPICE netlist. The measurement statements are recognised by the optimizer and will be used as performances to be optimised unless they are unselected through the GUI. The input netlist can contain '.include' statements or the circuit configuration file can be used to include parts that are common for various cells.

C. *Other inputs*

Other, more general, inputs refer to the program itself. An example are the settings for the genetic optimization algorithm. These options can be set either in the program ASCII configuration file or entered using the GUI.

III. Program flow

In figure 1 an overview is shown of the program flow. The user provides the specifications of the performances that are evaluated by means of the measurement statements in the SPICE netlist. These specifications for the target technology can be chosen to be the same as in the source technology or other values can be specified. Also the type (less than or greater than) of the specification can be set. The tool then returns the optimum cell sizes that ensure that every performance satisfies its specification. The optimization algorithm is a genetic algorithm.

A. *Genetic algorithm*

As optimization algorithm we employed the differential–evolution algorithm used in [3], which we altered slightly. It is a genetic algorithm that searches for a global optimum and uses continuous parameter values. Among the changes compared with [3] are the inclusion of parameter bounding and stop criteria. We will not go into the details of the algorithm here – for that we refer to [3] – but we will show its effectiveness for our purpose in section IV.

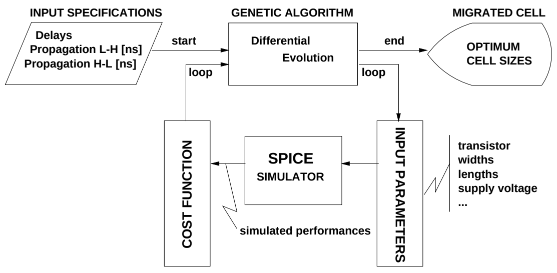


Fig. 1. The main program flow.

B. *Optimization parameters*

The genetic algorithm uses the parameters defined in the circuit configuration file. One population member in the genetic algorithm is therefore represented as shown in figure 2. These parameters are passed to the simulator which performs the requested analysis. The simulation results together with the specifications are then used to evaluate the fitness of the member by means of a cost function.

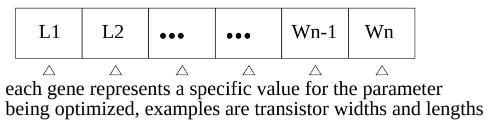

L1
L2
•••
•••
Wn-1
Wn
each gene represents a specific value for the parameter
being optimized, examples are transistor widths and lengths

Fig. 2. Representation of a population member in the genetic algorithm.

C. *Cost function formulation*

The cost function formulation is a crucial part in any optimization problem. Here, the cost function is defined as follows:

$$Cost = MAX_i \left(W \left(\frac{P_{spec_i} - P_{sim_i}}{P_{spec_i}} \right) \right) \tag{1}$$

This is a minimax problem formulation. The algorithm will try to minimize the cost, which is equal to the maximum normalized performance deviation from the specification. Each performance is thus normalized to have an equally important influence. Also, a weight factor W is included which is different when the specification is met (100) or not (100000). Note that with $W = 100$ and a cost threshold stop criterium of 1, a tolerance of 1 % is achieved.

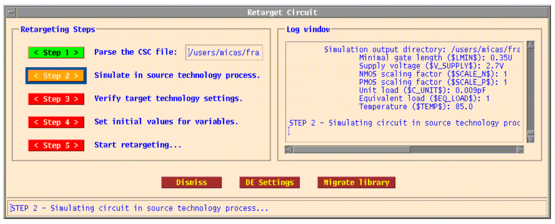
It is, however, also possible that the genetic algorithm proposes bad combinations of parameters (e.g. out of range). Then, a "high" cost is assigned (e.g. 1E8) to such solutions.

IV. Cell migration examples

In this section, we will exploit the capabilities of the tool to automatically find the scaling factors for the transistor widths

(NMOS and PMOS) that are necessary to migrate digital standard cells from one technology to a newer one. To have an optimal performance, the scaling factors are not necessarily the same for each type of cell. The source technology is a 0.35 μm process and the target technology has a 0.25 μm gate length. Since all cells have minimum gate length, we don't optimize the transistor lengths.

A first subsection introduces the graphical interface of the tool. In the next subsections, two experiments are reported. In the first experiment we specified the performance (in terms of propagation delay) of the target to be identical to the source. In the second experiment the performance specifications of the actual cells in the target technology were given as input to the tool.

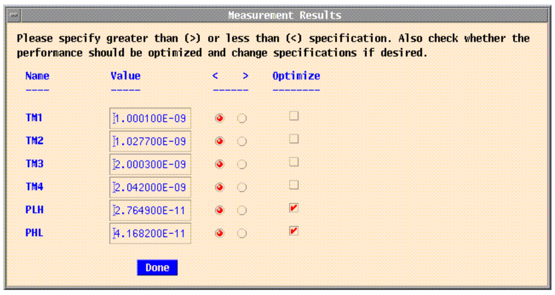


```
Retarget Circuit

Retargeting Steps
< Step 1 >  Parse the CSC file: /users/micas/fra
< Step 2 >  Simulate in source technology process.
< Step 3 >  Verify target technology settings.
< Step 4 >  Set initial values for variables.
< Step 5 >  Start retargeting...

Log window
      Simulation output directory: /users/micas/fra
              Minimal gate length ($LMIN$): 0.35U
              Supply voltage ($V_SUPPLY$): 2.7V
              NMOS scaling factor ($SCALE_N$): 1
              PMOS scaling factor ($SCALE_P$): 1
              Unit load ($C_UNIT$): 0.009pF
              Equivalent load ($EQ_LOAD$): 1
              Temperature ($TEMP$): 85.0

STEP 2 - Simulating circuit in source technology proc

STEP 2 - Simulating circuit in source technology process...
```

Fig. 3. The input window for the retargeting process.

A. *The graphical interface*

In figure 3 the tool input window is shown. The process of retargeting is divided into 5 steps which we will discuss briefly. All important information is printed in a log window. When a step has succesfully been completed, its color changes from red (to do) over orange (busy) to green. The first step parses the circuit configuration file discussed in subsection II-A. Based on this information, the circuit is simulated in the source technology process (step 2). The result of this simulation is shown in the *measurement results* window (see figure 4). In this window, the user can specify which measured performances are to be taken into account for the optimization as well as their type (less than or greater than specification). The default values are the simulation results of the source technology process.



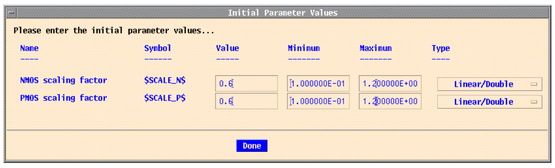
Measurement Results

Please specify greater than (>) or less than (<) specification. Also check whether the performance should be optimized and change specifications if desired.

Name	Value	<	>	Optimize
TM1	1.000100E-09	●	○	☐
TM2	1.027700E-09	●	○	☐
TM3	2.000300E-09	●	○	☐
TM4	2.042000E-09	●	○	☐
PLH	2.764900E-11	●	○	✓
PHL	4.168200E-11	●	○	✓

Fig. 4. The 'measurement results' window for the source technology simulation.

It is however possible to fill in other specification values and, f.i., tighten the specifications for the target technology. In the next step (the third step), the target technology settings in the circuit configuration file are verified. Then, in step 4, the user can select



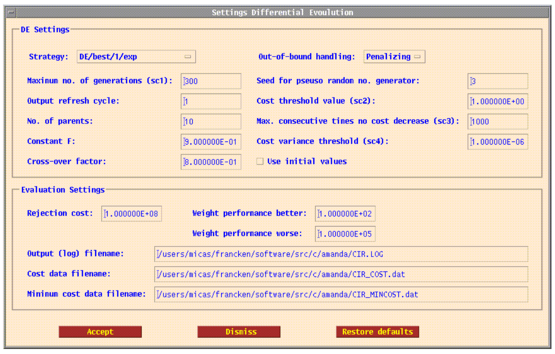
Initial Parameter Values

Please enter the initial parameter values...

Name	Symbol	Value	Minimum	Maximum	Type
NMOS scaling factor	$SCALE_N$	0.6	1.000000E-01	1.200000E+00	Linear/Double
PMOS scaling factor	$SCALE_P$	0.6	1.000000E-01	1.200000E+00	Linear/Double

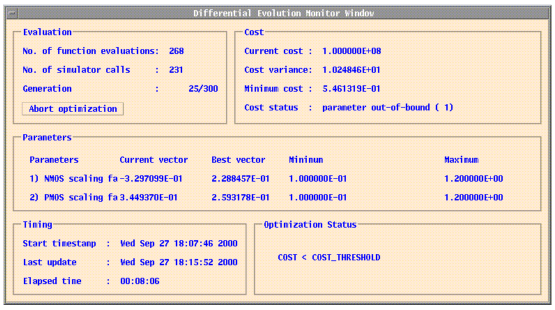
Fig. 5. The 'initial values' window for the target technology optimization.

initial, minimum and maximum values for the parameters to be optimized (figure 5). Also, the type of the variables can be set to be 'integer' (discrete values) or 'double' (continuous range) in combination with a linear or logarithmic pruning in the feasible range. Once all these steps have been performed succesfully, the actual retargeting process can commence. As we mentioned in subsection III-A, a genetic algorithm will be used as optimizer. By pushing the 'DE Settings' button, the user can tune a wide variety of settings for this algorithm and the cost function evaluation as well as specifying the output log files (figure 6).



```
Settings Differential Evoulution

DE Settings
Strategy: DE/best/1/exp                 Out-of-bound handling: Penalizing
Maximum no. of generations (sc1): 300   Seed for pseuso random no. generator: 3
Output refresh cycle: 1                 Cost threshold value (sc2): 1.000000E+00
No. of parents: 10                      Max. consecutive times no cost decrease (sc3): 1000
Constant F: 9.000000E-01                Cost variance threshold (sc4): 1.000000E-06
Cross-over factor: 8.000000E-01         ☐ Use initial values

Evaluation Settings
Rejection cost: 1.000000E+08    Weight performance better: 1.000000E+02
                                Weight performance worse:  1.000000E+05
Output (log) filename:     /users/micas/francken/software/src/c/amanda/CIR.LOG
Cost data filename:        /users/micas/francken/software/src/c/amanda/CIR_COST.dat
Minimum cost data filename: /users/micas/francken/software/src/c/amanda/CIR_MINCOST.dat
```

Fig. 6. The differential evolution algorithm and cost function evaluation settings.



```
Differential Evolution Monitor Window

Evaluation
No. of function evaluations:  268
No. of simulator calls     :  231
Generation                 :      25/300

Cost
Current cost :  1.000000E+08
Cost variance:  1.024846E+01
Minimum cost :  5.461319E-01
Cost status  :  parameter out-of-bound ( 1)

Parameters
 Parameters          Current vector    Best vector    Minimum         Maximum
 1) NMOS scaling fa -3.297099E-01      2.288457E-01   1.000000E-01    1.200000E+00
 2) PMOS scaling fa 3.449370E-01       2.593178E-01   1.000000E-01    1.200000E+00

Timing
Start timestamp  :  Wed Sep 27 18:07:46 2000
Last update      :  Wed Sep 27 18:15:52 2000
Elapsed time     :  00:08:06

Optimization Status
   COST < COST_THRESHOLD
```

Fig. 7. The monitor window shows intermediate results and cost function evolution.

Finally, when step 5 is activated, the intermediate results can be followed via the monitor window as illustrated in figure 7. We will now report some experimental results achieved with this tool.

B. *Experiment when keeping the performance specifications*

A first experiment migrates a simple inverter cell from a CMOS 0.35 μm to a 0.25 μm process, where we try to keep

TABLE I

SIMPLE INVERTER MIGRATION EXPERIMENT (PERFORMANCE KEPT EQUAL).

Cell	Scale N/P (real)	Scale N/P (optimized)	Δ [%]	PLH / PHL [ns] (real)	PLH / PHL [ns] (optimized)	Δ [%]	Final cost	time [h:mm:ss]	#gen
IV4	0.606 or 1/1.65	0.229 or 1/4.37	62.2	0.0276	0.0275	0.5	0.546	0:07:10	25
	0.694 or 1/1.44	0.259 or 1/3.85	62.6	0.0417	0.0416	0.2			

TABLE II

A SECOND EXPERIMENT TARGETS THE REAL LIBRARY PERFORMANCE SPECIFICATION.

Cell	Scale N/P (real)	Scale N/P (optimized)	Δ [%]	PLH / PHL [ns] (real)	PLH / PHL [ns] (optimized)	Δ [%]	Final cost	time [h:mm:ss]	#gen
IV4	0.606 or 1/1.65	0.625 or 1/1.60	3.1	0.0205	0.0203	0.8	0.995	0:11:09	38
	0.694 or 1/1.44	0.716 or 1/1.40	3.2	0.0310	0.0307	1.0			
AN2	0.625 or 1/1.60	0.651 or 1/1.54	4.2	0.1340	0.1334	0.5	0.742	0:03:20	11
	0.708 or 1/1.41	0.760 or 1/1.32	7.2	0.1267	0.1258	0.7			
EO	0.625 or 1/1.60	0.673 or 1/1.49	7.7	0.2880	0.2865	0.5	0.562	0:04:04	12
	0.652 or 1/1.53	0.687 or 1/1.46	5.3	0.3361	0.3342	0.6			

the performances. So, the question is: How small can the transistors be sized in the 0.25 μm technology as to still have the same performance as in the 0.35 μm technology? Note that the scaling factor for the transistor length is 0.714 (1/1.4). The results are given in table I, where a comparison is made with actual (real) cell data that were hand–crafted by the manufacturer in the same technology process. The final cost function value is given together with the time taken by the tool and the number of generations of the genetic algorithm. Although a genetic algorithm is very well suited for parallel execution, the numbers presented here are the results of execution on a single host computer (SUN Ultra 30). We can conclude from the table that the optimized performances are within the given tolerance of 1 % – 0.5 % for low–to–high and 0.2 % for high–to–low propagation delay respectively. Nevertheless, the optimized parameters – the NMOS and PMOS scaling factors – deviate by as much as 62 % from the real values that we had in the manufacturer's library. This is, of course, due to the fact that the speed specification is more strict for the target library in practice. Otherwise no advantage of the faster process would be taken.

C. Experiment with target performance specifications

In practice, when migrating to a smaller feature size technology, also the speed specification is increased. In table II we present the results of an experiment similar to the first but where the target performance specification is entered to be equal to the real simulated target cell specifications from the manufacturer's hand–crafted library. This is done for three different cells (inverter, 2–input and, exor). Again, a comparison has been made between results from the tool and the actual cell data. It is clear that the scaling factors now match better with the real values. Nevertheless, they deviate by 3 to 8 %, even though the optimized performance is within 1 % of the specification (as requested). This is probably due to design margins that are taken in a real design.

D. Discussion of the results

The above mentioned experiments show that the migration flow works and that the user can arbitrarily set the target specifications. The performances of the optimized cells are within the accuracy specified by the user (1 % in our example). Also, the optimization times are well within reasonable limits since the library migration will be done only one time for every new process. In addition, we didn't make use of parallel execution on different host computers, which would speed up the optimization even further.

V. CONCLUSIONS

We successfully introduced a tool that uses a genetic algorithm to perform the cell resizing in the migration of a digital standard cell library. The key advantages of the tool are its flexibility – both in parameters and specifications setting, its speed and its accuracy. By making templates of the cells only once, a fast migration at the level of cell sizing is possible for every subsequent technology. It is assumed that the cell topology doesn't change.

ACKNOWLEDGEMENTS

This work has been supported by the IWT project FrontEnds in cooperation with Alcatel Microelectronics.

REFERENCES

[1] R. Carley, Georges Gielen, R. Rutenbar, and Willy Sansen, "Synthesis tools for mixed–signal ics: progress on frontend and backend strategies," in *Proceedings Design Automation Conference (DAC)*, June 3–7 1996, 33rd, pp. 298–303, Las Vegas, USA.

[2] C. Visweswariah, "Optimization techniques for high-performance digital circuits," in *Proceedings IEEE/ACM International Conference on Computer–Aided Design (ICCAD)*, November 9-13 1997, pp. 198–207.

[3] R. Storn, "On the usage of differential evolution for function optimization," in *NAFIPS*, 1996, pp. 519–523.