Automated Synthesis of Cell Libraries for Asynchronous Circuits

Matheus Trevisan Moreira
GAPH - PPGCC - PUCRS
Porto Alegre, Brazil
matheus.moreira@pucrs.br

Michel Arendt
GAPH - PPGCC - PUCRS
Porto Alegre, Brazil
michel.arendt@acad.pucrs.br

Adriel Ziesemer Jr.
PGMicro - PPGC - UFRGS
Porto Alegre, Brazil
amziesemerj@inf.ufrgs.br

Ricardo Reis
PGMicro - PPGC - UFRGS
Porto Alegre, Brazil
reis@inf.ufrgs.br

Ney Laert Vilar Calazans
GAPH - PPGCC - PUCRS
Porto Alegre, Brazil
ney.calazans@pucrs.br

ABSTRACT

Asynchronous techniques are regaining relevance in the VLSI research community as they allow increasing robustness against process variability considerably, by relaxing timing assumptions. In addition, asynchronous circuits enable achieving low-power and high-speed designs. However, due to the absence of commercial dedicated standard cell libraries to take the most of asynchronous design, such circuits implementations are relegated to full-custom approaches only. This limits applicability of asynchronous solutions and avoids further development of dedicated design automation tools. This paper describes an improvement to this situation by proposing a fully-automated design-flow called ASCEnD-A, able to implement standard cells specifically required for asynchronous circuits design. The flow is capable of generating cells at the layout level, providing physical, power and timing models required by cell-based flows available in the state-of-the-art technologies.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids; B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Design

Keywords

Cell library, semi-custom, asynchronous circuits

SBCCI '14, September 01 - 05 2014, Aracaju, Brazil

ACM 978-1-4503-3156-2/14/09 ...$15.00.
http://dx.doi.org/10.1145/2660540.2660984

1. INTRODUCTION AND RELATED WORK

A key factor for the rapid growth of VLSI systems is the support provided by cell-based, semi-custom design flows. Such flows lower design complexity by using pre-designed and pre-characterized functional components called *(standard) cells*, instead of assuming that designers need to draw, place and connect each transistor by hand. This facilitates to timely build complex systems. Also, as CMOS technology evolves into deep submicron nodes, asynchronous techniques gain relevance in the VLSI research community, due to their ability to cope with problems that are hard to solve with the synchronous paradigm [13]. In fact, the last decades have witnessed a substantial development of new techniques and tools for semi-custom asynchronous design. Among the ones available in the state-of-the-art see for example [2–7, 9, 11, 12, 16, 18, 20, 21], which allow automating and optimizing either quasi-delay-insensitive (QDI) and/or bundled-data design.

The drawback is that, albeit these works allowed exploring optimizations for asynchronous design, all of them have a common assumption: the availability of new gates at the cell level. Indeed, these gates are unavailable in commercial standard cell libraries, severely constraining the adoption of the techniques and tools proposed to date, as designers need to implement the required gates by hand. In fact, this limits the modularity of asynchronous design and increases design complexity, making it less appealing. Among new gates that are typically required by asynchronous design techniques and tools it is possible to cite Null Convention Logic (NCL) gates, C-Elements (which are actually a subset of NCL gates), precharge half-buffers and mutual exclusion elements, all discussed in detail e.g. in [8, 13]. Most mentioned components can be constructed using standard cells available off-the-shelf in conventional libraries. However, this is not efficient and can introduce hazards. One of the factors that can facilitate the adoption of asynchronous techniques is the availability of cell libraries to support them.

However, designing such libraries is a very laborious task, as it requires significant expertise in microelectronics and asynchronous design. Having a mature design flow that automates the tasks involved in the library development is of extreme importance. Some of the few works found in current literature are ASCEnD [14], the former version of the flow

proposed herein, and cellTK [10], an automated layout generator that was recently proposed by Karmazin *et al.* The drawback is that the former had no support for layout and characterization design phases, which imply a high degree of manual labor for obtaining the library. Karmazin *et al.* [10] on the other hand, propose an automated layout generator cellTK. The drawback is that this generator is not compatible with techniques and tools for asynchronous design automation proposed to date as it employs a non-standard flow, rather than targeting a cell-based semi-custom flow. Also, the tool provides no support for automatic transistor dimensioning and there is no mention of how the generated layouts can be characterized to generate power and timing models, which limits its usability in semi-custom design tools. Another drawback is the area, energy and delay penalties imposed by cellTK when compared to manual designs, 51%, 12% and 9% in average, respectively [10].

This work presents ASCEnD-A (ASCEnD-Astran), a design flow devised for generating components required by asynchronous circuits at the cell level. The flow uses a set of specially designed tools, which are parameterizable for any CMOS technology. Its basic inputs are a transistor level netlist, together with an electrical specification that defines the driving strength of the gate. From these, ASCEnD-A can automatically generate the layout of the cell and produce power and timing models in the Liberty format after layout extraction. In fact, the difference of ASCEnD-A and the original flow, ASCEnD, is the integration of the support of automatic layout generation with ASTRAN [22]. Layout verification and extraction relies on the use of conventional tools. Electrical characterization is done using LiChEn [17], an inhouse tool specially devised for characterizing asynchronous cells. All single output gates required by contemporary techniques and tools can be designed using ASCEnD-A. Hence, the flow stands off by providing an automated solution for the generation of cell libraries compatible with state-of-the-art methods and tools for asynchronous design automation.

2. CELL DIMENSIONING

The quality of a cell-based design is a direct function of the availability of cells with different functionality and different driving strengths. The former depends on the logic function implemented by the cells and the latter depends on the capability of charging/discharging a specific output load in a specific period of time. Obviously, bigger driving strengths require bigger transistors. Usually, cells are available in several different driving strengths for a same logic function, to provide more optimization opportunities in operating speed and power. To allow a range of driving strengths to be implemented, the first step in the ASCEnD-A flow, as Figure 1 shows, is the process of setting the dimensions of the cell transistors.

To do so, the designer must provide the following inputs: the transistor level schematic of the cell and a configuration file for the ROGen tool, specifically designed for the proposed flow [14]. Note that this tool also requires the schematics of a basic 2-input NAND and an inverter with the same driving strength of the cell to be dimensioned (called reference inverter). Albeit this can be library specific, i.e. for each cell library different standards for driving strengths may be defined, we strongly advise that standards defined in the core library (the basic library that typically ships with a design kit) are adopted. In fact, when developing our cell libraries, we employ inverters and NANDs of the core library to respect driving strength standards. Note that if the cell to be dimensioned performs a non-inverting functionality, it employs an output inverter, required by the inverting nature of CMOS logic. Hence, this inverter must have transistors of size similar to those of the reference inverter. Also, gates required for asynchronous design typically employ memory schemes and most often present feedback loops. Accordingly, transistors involved in feedback loops are always minimum size to interfere the least in the cell performance. However, we advise designers to use static versions of these schemes to avoid crosstalk and PVT variations problems [4]. Table 1 shows the variables that can be defined in the the configuration file provided to ROGen.

This information is provided in a text based file and most of the parameters are generic for a same cell library. Given the set of inputs, the dimensioning process begins with the generation of a simulation environment described in SPICE that exhaustively varies transistor sizes as specified and performs the following measurements for each variation: (i) propagation delays for rising and falling transitions, (ii) transition delays for rising and falling transitions, (iii) static power and (iv) dynamic power. As Figure 1 shows, simulation is currently performed using Cadence Spectre, albeit any SPICE simulator can be supported. Note that technology models must be provided for the simulation to take place. From the resulting measurements, ASCEnD-A will dimension the transistors of the standard cell according to a specific cost function. As Figure 1 shows, this is done by feeding another in-house tool, called CeS [14], with the simulation report and a cost function using any combination of (i), (ii), (iii) and (iv) and the following arithmetic operators: +, -, /, *. The designer can also use parenthesis and pipes. The former allow ensuring the order of operations and the latter returns absolute values only. From this cost function, CeS will generate the best NMOS-PMOS size matches in descending order, according to the cost function.

This allows exploring tradeoffs and cell sizing design space. For instance, a cost function can focus only in high-speed: *1/(prop_rise+prop_fall)*, where *prop_rise* and *prop_fall* correspond to rising and falling propagation delays, respectively. Another possibility is having the best tradeoff between delay and power: *1/((prop_rise+prop_fall)*dynpwr)*, where *dynpwr* corresponds to the measured dynamic power. Also, the tool allows an incremental analysis to take place. To do so,

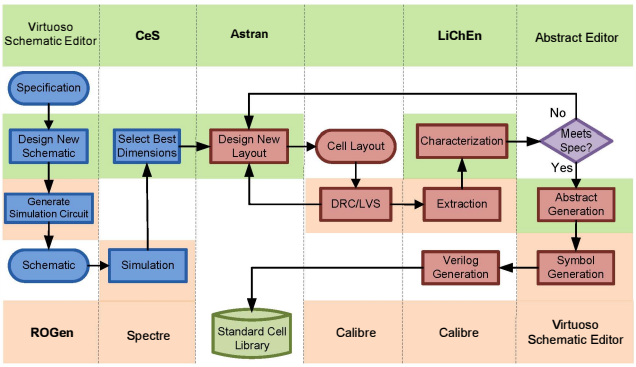

Figure 1: ASCEnD-A design flow.

Table 1: Variables for the configuration file.

1. The number of gate inputs;
2. If the gate has a reset and/or set input. This is very useful for constructing gates employed in registers for QDI circuits [13];
3. If a possible reset and/or set input is active low or high;
4. If the gate has an inverted function;
5. The interface of the cell using the reserved words *in*, *out*, *set*, *rst*, *vdd* and *gnd* for identifying the pins;
6. The threshold of the gate. Used for NCL and NCL+ gates. For any other gate, the value must be defined as the same of the inputs number;
7. If the gate assumes return-to-zero (RTZ) or return-to-one (RTO). The default value is 0, when set to 1, it is used for dimensioning gates based on the RTO protocol, such as NCL+ [15];
8. If the cell employs differential logic. Note that albeit transistor dimensioning and automatic layout generation are already supported for differential gates, electrical characterization is not yet supported, as LiChEn supports only single output gates [17];
9. Fanout. Default is FO4;
10. The operating temperature;
11. The operating voltage;
12. Voltage levels for logic 1 and 0 and low-to-high and high-to-low switching thresholds;
13. Time and dimension units;
14. Simulation step and total time;
15. Start time and end time for measuring static and dynamic power;
16. Maximum size of NMOS and PMOS transistors for a single transistor finger. This allows the flow to perform realistic measurements, while respecting layout definitions;
17. Minimum and maximum PMOS and NMOS sizes and variation step. Note that during dimensioning, gates will have their transistor dimensions varied according to these definitions.

a first cost function is defined, for selecting only the best NMOS-PMOS size matches and next another cost function is applied to that set. For instance, assume that the designer defines an incremental analysis, providing the following initial cost function: $1/|prop\_rise\text{-}prop\_fall|$. This will select only NMOS-PMOS size matches that allow balanced rising and falling propagation delays. Next, if the designer provides the $1/(prop\_rise+prop\_fall)/(leakpwr)$ cost function, where *leakpwr* is the measured leakage power, CeS will select the best NMOS-PMOS size match, from the previously selected set, that presents the best delay and leakage power tradeoffs. In other words, the selected match will present the best delay and leakage power tradeoff among the most balanced matches, in terms of rising and falling propagation delays. Using cost functions, the designer can specify different versions of each gate. This enables enriching the library. After having the transistors dimensioned, the next step is to generate the layout.

3. LAYOUT DESIGN

The academic netlist-to-layout tool ASTRAN [22] is used in this work to synthesize cell layouts from the transistor level schematic using the dimensions defined by CeS. ASTRAN was developed to do the layout of any transistor network to allow the implementation of the methodology presented in [22]. ASTRAN supports unrestricted circuit structures, continuous transistor sizing, folding, poly and

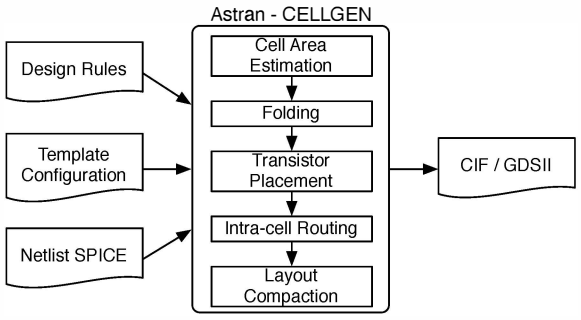

Figure 2: Layout Synthesis Flow

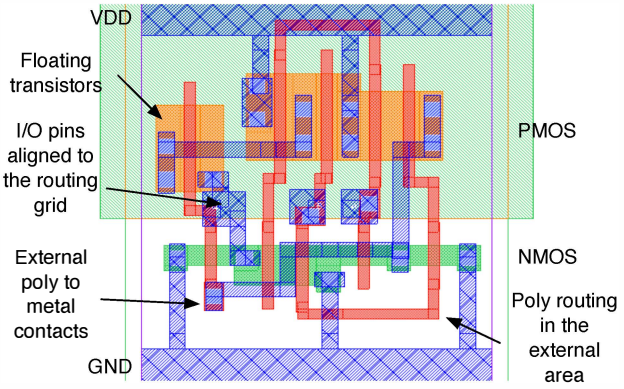

Figure 3: ASTRAN Layout style

over-the-cell metal 1 routing, redundant contacts insertion, minimum distance relaxation for DFM and conditional design rules. It features a transistors placement algorithm for width reduction and an intra-cell router. Mixed-Integer Linear Programming (MILP) is used for two-dimensional compaction. It produces the final layout according to the results provided by the placement/routing steps and regarding the technology design rules.

The input of ASTRAN is a transistor level description of a circuit in SPICE format. Each circuit, denoted by ".subckt", can be synthesized into a cell-level layout. The technology rules are set according to the values defined by the foundry. ASTRAN supports rules defined by most of the processes down to $45nm$. The cell topology (height, routing grid, wells/power rails position and other library specific aspects) are defined according to the target library. The flow employed by ASTRAN is illustrated in Figure 2. It makes use of the 1D layout style which consists of two rows of PMOS and NMOS transistors with vertical gates as Figure 3 shows. The resulting layouts can be exported to the CIF format and then imported into Cadence where DRC, LVS and extraction can be performed with conventional tools, in this case Mentor Calibre. In order to automate the cell layout generation and integrate it in ASCEnD-A, we developed a set of scripts that import layouts generated by ASTRAN into the Cadence Framework for verification and extraction. After the layout is fully verified, a physical view is generated with Cadence Abstract Editor, albeit any other conventional tool could be used, and exported to the Library Exchange Format (LEF), widely accepted by EDA vendors.

4. CELL CHARACTERIZATION

Characterizing cells required for asynchronous design using conventional tools can be a challenging task. In fact, it is the authors own experience that their use in the characterization of asynchronous standard cells is very laborious. The problem is that when the logic behavior of the cell is not automatically identified by these tools, the designer needs to manually specify it inside the database. This means that the characterization process must be stopped, while the modification is done. This requires the generation of some text files by the tool, which are then modified by the designer, according to a specific syntax. Once the correct logic behavior is specified, the designer must update the database and the tool can continue its characterization flow. Therefore, LiChEn [17] was designed to overcome this problem. From the netlist extracted from the generated layout, LiChEn automatically generates power and timing models for each cell in the Liberty format, widely accepted by EDA vendors. The tool is based on the generation of a simulation environment where all arcs and static states of a cell are exercised. All commands are given through a command line interface, and text-based scripts can help the automation of standard cell libraries characterization. As Figure 4 shows, the flow employed by the tool comprises three main steps, represented by diamonds: (i) producing a cell simulation environment, (ii) finding static states and transition arcs, and (iii) generating timing and power figures through SPICE simulation.

For configuring the simulation environment, technology-specific data, providing corner selection, operating conditions, electrical specifications and simulation parameters are required. Accordingly, the following information must be provided for the tool: Operating voltage; Voltage levels for valid logic 1 and 0 and low-to-high and high; Operating temperature; and Maximum simulation time and simulation step. Pin-to-pin propagation and transition delays and dynamic power are dependant of input slew rate and output capacitance load. Hence, these delay and power figures are modeled in non-linear tables, where each value depends on a combination of an input slew rate and an output load capacitance. To do so, input slew rate and output load capacitance vectors are required, to generate the characterization simulation environment. The quality of the provided vectors are strictly related to the quality of the models generated by the tool. In this way, we advise the designer to employ vectors similar to those used in the core library for same driving strengths.

After configuring the simulation environment and providing the extracted netlist of the cell to be characterized, the designer must provide the functionality of the cell. This guides the tool in its search for transition arcs and static states, which are used to conduct the required simulations. In LiChEn, logic functions can employ the following logic operations: conjunction ($*$), disjunction ($+$) and complement ($\sim$). Moreover, parenthesis can be used to express hierarchy. For instance, the basic C-element with inputs A and B and output Q has the following logic function: $Q = (A*B)+(A*Q)+(B*Q)$. From this function, LiChEn employs a branch and bound algorithm to find all transition arcs and static states (SSs). Basically, the tool initially sets all input and output pins to logical 0 and computes the resulting value of the output according to the logic function, obtaining the first static state. Next, it switches the logical value of one input at a time and evaluates the next output value through a recursive function. Each new input/output value after a transition is accounted as a new SS. To avoid exponential computation time, as soon as LiChEn detects that a previously computed SS was already computed previously, it stops the recursion. During switching of the inputs logical values, LiChEn evaluates in which cases the output value changes when an input switches. These cases are called Dynamic Transition Arcs (DTAs). The cases where input switching does not cause an output switching are called Internal Transition Arcs (ITAs). This simple algorithm is capable of detecting all SSs, DTAs and ITAs, even those that depend on feedback loops. This allows efficiently characterizing the electrical behavior of asynchronous cells. Yet, currently, the tool supports only single output cells. However ongoing work includes implementing the support to multi-output.

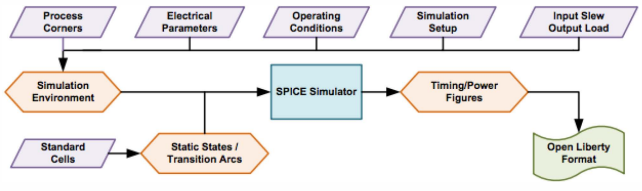

Figure 4: Electrical characterization flow implemented by LiChEn.

LiChEn may start the characterization process once all SSs, DTAs and ITAs have been computed. Based on these results, the tool generates SPICE files that implement each transition arc and each SS. Measurements are conducted during SPICE simulation, based on the configuration given in the first stage of the characterization flow. During characterization, LiChEn measures input gate capacitance for low-to-high and high-to-low transitions, static and dynamic power and timing figures. Timing figures are measured as pin-to-pin propagation delays and output transition delays. The static power is measured for each SS. Dynamic power, in turn, is divided in two parts: switching and internal power, measured as the power consumed in DTAs and ITAs, respectively. The input gate capacitance is computed as the average current for low-to-high and high-to-low input transitions and exported as rising and falling input capacitances, respectively. Static power, on the other hand, is measured for each SS according to the average current drawn from the power source. In addition, the static power consumption of each SS is measured for each power source independently. After simulation and data collection, LiChEn exports the obtained timing and power figures to a text-based file according to the Liberty format. Finally, after characterization symbol and behavioral Verilog views can be generated from templates, as they don't vary from library to library.

5. EXPERIMENTS AND DISCUSSION

Previously to ASCEnD-A, a library composed by more than 500 cells library had its layouts designed by hand targeting the STMicroelectronics 65nm bulk CMOS technology. This library was composed of C-Elements and NCL gates of different functionality and driving strengths. Accordingly, we selected a set of these gates and automatically implemented them targeting the same technology using the

Table 2: Set of cells chosen for comparison.

1. Resettable 2 inputs Sutherland C-Elements (RSUC2) with 6 driving strengths (X2, X4, X7, X9, X13, X18);
2. Resettable 2 inputs Weak Feedback C-Elements (RWFC2) with 6 driving strengths (X2, X4, X7, X9, X13, X18);
3. Resettable 2 inputs van Berkel C-Elements (RVBC2) with 6 driving strengths (X2, X4, X7, X9, X13, X18);
4. 2 inputs Sutherland C-Elements (SUC2) with 5 driving strengths (X2, X4, X7, X9, X13);
5. 2 inputs Weak Feedback C-Elements (WFC2) with 6 driving strengths (X2, X4, X7, X9, X13, X18);
6. 2 inputs van Berkel C-Elements (VBC2) with 6 driving strengths (X2, X4, X7, X9, X13, X18);
7. 3 inputs Sutherland C-Elements (SUC2) with 6 driving strengths (X2, X4, X7, X9, X13, X18);
8. 3 inputs Weak Feedback C-Elements (WFC2) with 5 driving strengths (X2, X4, X7, X9, X13);
9. 1-of-2, 1-of-3, 1-of-4, 2-of-3, 2-of-4, 3-of-4 and 3-of-5 NCL gates (NCL12, NCL13, NCL14, NCL23, NCL24, NCL34 and NCL35) with X4 driving strength.

flow presented in this article for comparing the handcrafted gates with the automatically generated ones. This allows a perspective of the quality of gates delivered by ASCEnD-A. The chosen set of 53 cells for the comparison, described in Table 2, includes different topologies and functionalities of C-Elements [19] and NCL [8] gates.

The different C-Element topologies allow different trade-offs to be explored and are typically required by asynchronous templates. Each topology have a different transistors arrangement and present different levels of internal routing congestion. Resettable versions are also very important as they are the basis for implementing QDI registers. Moreover, 3 inputs implementations of the component are usually required by QDI combinational blocks that require functionality of more than two variables. In addition, NCL gates are usually more complex cells and employ more transistors in their design. In this way, the selected set of gates allow comparing how the proposed design flow cope with topological aspects such as different degrees of internal routing congestion, different transistors arrangements and connectivity. Also, having different driving strengths for each gate allows evaluating how the flow copes with different transistors dimensions, which can be challenging because cells must all have the same height and big transistors may have to be folded.

All results presented in this section are comparative. Therefore we present them as a relation between aspects of cells generated with ASCEnD-A and ASCEnD in %. Note that positive % values denote improvements while negative % indicates overheads. Accordingly, Figure 5 presents the obtained results. The area of the case study gates was measured directly from their layout. As the first row of charts show, ASCEnD-A provided area reductions in the majority of the evaluated cells. In fact, it achieved improvements of roughly 19.2% in average and 45% in best case. From the set of gates, only two versions of WFC3 presented overheads, which were minimum (8%). Parasitics of all gates were extracted from layout using Calibre PEX assuming worst case RC extraction. The second row of charts of Figure 5 shows the observed improvements and overheads provided by ASCEnD-A in total internal parasitics. The flow provided size reductions in the design of many of the gates, with a best case of 60%. However for some gates it caused overheads of up to 65%. In fact, improvements and overheads were quite balanced over the chosen set of gates and the average improvement was of roughly 1%. Note that problematic gates were typically those with a complex internal routing, such as VBC2 and RVBC2. After extraction, the worst cases of observed input capacitance were also compared. The obtained results are summarized in the third row of Figure 5. Accordingly, ASCEnD-A provided improvements in the majority of the gates. In fact, such improvements were of 59% in best case and roughly 17% in average. Note that overheads were limited to a small set of gates and in the worst case were of 21%.

The extracted netlists of each gate was simulated using Cadence Spectre and had all its transition arcs and static states exercised. During simulation, energy per transition and propagation delay of each arc and leakage power of each static state were measured. Simulation assumed typical process with gates operating at typical voltage and temperature (1 V and 25 C). Using the measured values we computed average energy per transition, average delay and average leakage power of each gate, for enabling a fair comparison. Accordingly, as the fourth and fifth rows of charts in Figure 5 show, ASTRAN typically provides better average energy per transition and delay figures. In fact, the observed variations between cells generated by ASTRAN and those handcrafted were very small for C-Elements. On the other hand, NCL gates generated by ASTRAN presented improvements of up to 46% in both average energy per transition and delay. This indicate the suitability of the proposed tool for coping with high complexity gates, as NCL gates are typically more complex than C-Elements. Note that the cases where ASTRAN caused overheads in energy and delay were precisely the cases that it caused overheads in internal parasitics. In this way, reducing the parasitics of gates generated by ASTRAN will lead to further improvements in the design flow. In fact, this is ongoing work. Finally, ASCEnD-A provided reductions in leakage power in the majority of the evaluated gates. Albeit these reductions were quite modest, 3.6% in average and 9% in best case, they can be substantial in the ultra-deep submicron era, where leakage power is comparable to dynamic power.

The obtained results demonstrate the suitability of ASTRAN to automatic generation of layouts in the proposed flow. Accordingly, the tool enabled better figures in most of the cases. Also, the reductions provided in input capacitance do not have a direct effect in the experiments presented herein. However it is expected that circuits employing gates with lower input capacitance present better energy and speed trade offs, as the energy and speed of a cell are widely affected by the load in their outputs. Another very important aspect of having ASTRAN integrated in the flow is the fact that the case study set of gates was generated in 2 days by a single designer. Note that, albeit layout generation is automated and as many layouts as the number of available computers can be done at the same time, the designer has always to verify the generated layout to guarantee that no abnormality was generated. Also, for some layouts, minor DRC errors required to be manually corrected. These errors were usually related to over-insertion of cores in polysilicon and metal layers and the effort to correct them was minimum. Having that said, manually designed layouts of ASCEnD generally took half a day for a single designer to

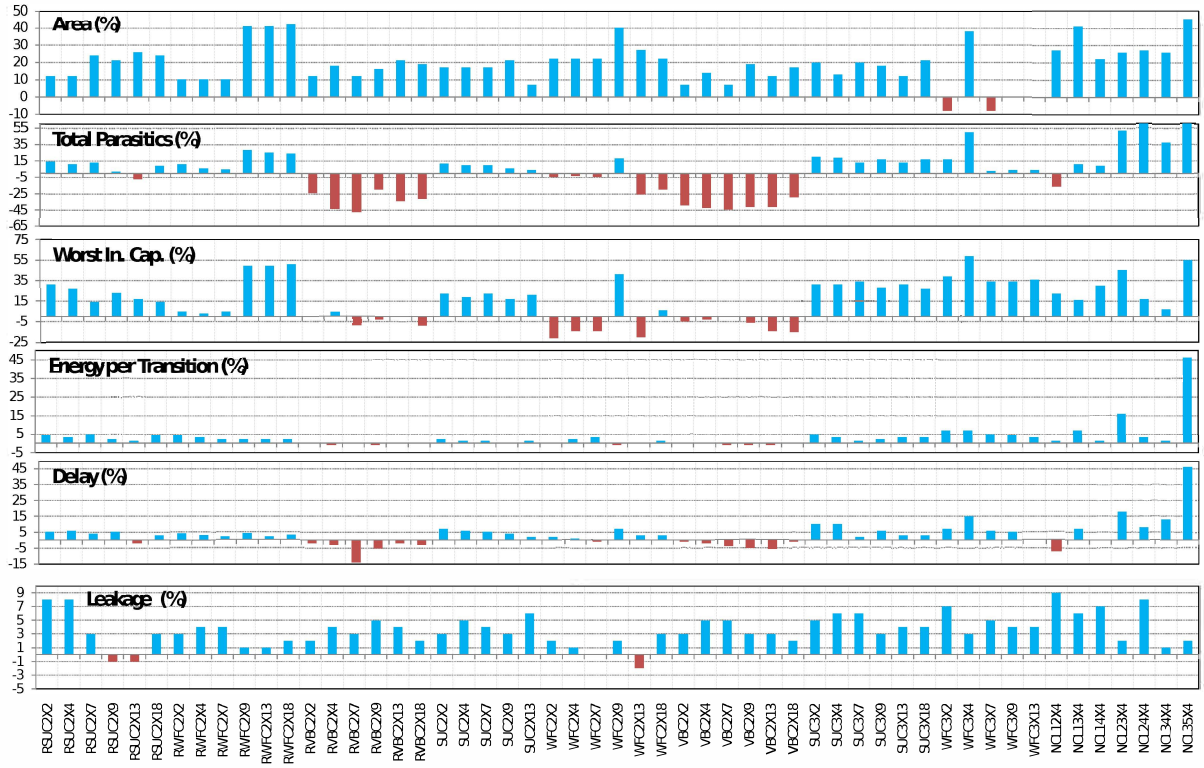

Figure 5: Comparison between cells manually generated in ASCEnD and automatically generated in ASCEnD-A.

project it. Moreover, complex NCL gates took several days. This means that 53 gates would require at least 27 days to be designed. In this way, ASCEnD-A allows one order of magnitude improvements in productivity. It is important to highlight that during layout generation of the case study set of gates, ASTRAN was configured with its most basic definitions in order for it to be easily integrated in ASCEnD-A. This led to non-optimum design of the cells. We are currently tuning the tool to provide a set of optimizations that will enable reducing parasitics and, as consequence, improving delay, energy and power figures.

6. CONCLUSIONS AND FUTURE WORK

ASCEnD-A enables the automatic synthesis for the rapid generation of cell libraries for semi-custom asynchronous design. Experimental results reported in this article demonstrate that the quality of cells generated by the flow was superior to those manually generated in ASCEnD in many aspects such as area, parasitics, input capacitance, energy per transition, delay and leakage power. This is in contrast to state-of-the-art proposals for automatic layout generation for asynchronous circuits. In this way, ASCEnD-A can become an important step towards wider acceptance of the asynchronous paradigm, as cell libraries generated by it can be coupled to state-of-the-art design flows to enable a semi-custom approach.

Currently, the access to ASCEnD-A is free of charge and the tools that comprise it can be adjusted to a specific technology. Also, a library composed by C-Elements, metastability filters and NCL gates targeting the STMicroelectronics 65nm bulk CMOS technology is currently available given that the interested party has access to this technology [14]. A version of the library targeting the IBM 130nm bulk CMOS process is ongoing work. This later technology is compatible with MOSIS [1], which provides free access and prototyping targeting this technology for academic institutions. As future work the authors devise to fabricate a test-chip of several case study circuits employing cells designed with ASTRAN. Finally, the design of metastability filters in ASCEnD-A is still manual, only automatic layout is supported. Ongoing work will allow overcoming this limitation as the tools of the flow are being optimized to support multi-output cells.

7. ACKNOWLEDGMENT

Authors acknowledge the support of CNPq under grants 401839/2013-3, 200147/2014-5 and 310864/2011-9 and the support of FAPERGS under grant 11/1445-0.

8. REFERENCES

[1] Mosis integrated circuit fabrication service - http://www.mosis.com.

[2] A. Bardsley. *Balsa: An asynchronous circuit synthesis System*. PhD thesis, University of Manchester, 1998.

[3] A. Bardsley et al. Teak: A token-flow implementation for the balsa language. In *International Conference on Application of Concurrency to System Design*, pages 23–31, 2009.

[4] P. A. Beerel et al. Proteus: An asic flow for ghz asynchronous designs. *IEEE Design & Test of Computers*, 28(5):36–51, 2011.

[5] I. Blunno and L. Lavagno. Automated synthesis of micro-pipelines from behavioral verilog hdl. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 84–92, 2000.

[6] J. Cheoljoo and S. Nowick. Technology mapping and cell merger for asynchronous threshold networks. *IEEE Transactions on CAD*, 27(4):659–672, 2008.

[7] I. David et al. An efficient implementation of boolean functions as self-timed circuits. *Computers, IEEE Transactions on*, 41(1):2–11, 1992.

[8] K. Fant and S. Brandt. Null convention logic^{TM}: a complete and consistent logic for asynchronous digital circuit synthesis. In *Proceedings of International Conference on Application Specific Systems, Architectures and Processors*, pages 261–273, 1996.

[9] B. Folco et al. Technology mapping for area optimized quasi delay insensitive circuits. In *IFIP Conference on Very Large Scale Integration Systems*, pages 55–69, 2005.

[10] R. Karmazin et al. celltk: Automated layout for asynchronous circuits with nonstandard cells. In *International Symposium on Asynchronous Circuits and Systems*, pages 58–66, 2013.

[11] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous cad tools. *IEEE Design Test of Computers*, 19(4):107–117, 2002.

[12] M. Ligthart et al. Asynchronous design using commercial HDL synthesis tools. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114–125, 2000.

[13] A. Martin and M. Nystrom. Asynchronous techniques for system-on-chip design. *Proceedings of the IEEE*, 2006.

[14] M. Moreira, B. Oliveira, J. Pontes, and N. Calazans. A 65nm standard cell set and flow dedicated to automated asynchronous circuits design. In *IEEE International SOC Conference*, pages 99–104, 2011.

[15] M. Moreira, C. Oliveira, R. Porto, and N. Calazans. Ncl+: Return-to-one null convention logic. In *IEEE International Midwest Symposium on Circuits and Systems*, pages 836–839, 2013.

[16] M. T. Moreira et al. Semi-custom ncl design with commercial eda frameworks: Is it possible? In *International Symposium on Asynchronous Circuits and Systems*, 2014.

[17] M. T. Moreira, C. H. M. Oliveira, N. L. V. Calazans, and L. C. Ost. Lichen: Automated electrical characterization of asynchronous standard cell libraries. In *Euromicro Conference on Digital System Design*, pages 933–940, 2013.

[18] S. Nowick and M. Singh. High-performance asynchronous pipelines: an overview. *Design & Test of Computers, IEEE*, 2011.

[19] M. Shams et al. A comparison of cmos implementations of an asynchronous circuits primitive: the c-element. In *Low Power Electronics and Design, 1996., International Symposium on*, pages 93–96, 1996.

[20] Y. Thonnart et al. A pseudo-synchronous implementation flow for wchb qdi asynchronous circuits. In *International Symposium on Asynchronous Circuits and Systems*, pages 73–80, 2012.

[21] K. Van Berkel et al. The vlsi-programming language tangram and its translation into handshake circuits. In *Proceedings of the European Conference on Design Automation*, pages 384–389, 1991.

[22] A. Ziesemer et al. Automatic layout synthesis with ASTRAN applied to asynchronous cells. In *Latin American Symposium on Circuits and Systems*, pages 1–4, Feb 2014.