A 65nm STANDARD CELL SET AND FLOW DEDICATED TO AUTOMATED ASYNCHRONOUS CIRCUITS DESIGN

Matheus Moreira, Bruno Oliveira, Julian Pontes, Ney Calazans

Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre - RS - Brasil
{matheus.moreira, bruno.scherer}@acad.pucrs.br {julian.pontes, ney.calazans}@pucrs.br

ABSTRACT

This work proposes a new design flow for rapid creation and characterization of standard cell sets for asynchronous design. The flow is fully automated except for the cell layout generation step. It has been applied to the design of a standard cell set supporting the Teak asynchronous synthesis tool. Cells use a 65 nm gate length commercial CMOS process. An asynchronous RSA cryptography circuit provides the design flow validation.

I. INTRODUCTION

The interest in non-synchronous circuits is increasing. The International Technology Roadmap for Semiconductors (ITRS) in its 2008 edition [1] describes a clear need for asynchronous communication protocols in integrated circuits (ICs) control and synchronization along the next decades. For example, the ITRS estimates that global clock-based ICs, which comprised 93% of the chips sold worldwide in 2007, will have only 55% of the market by 2022. The other 45% of the chips will be local handshaking circuits, including clockless or multi-clock ICs. However, the lack of adequate electronic design automation (EDA) tools imposes a barrier to the design of asynchronous circuits. Most commercial EDA tools focus currently on purely synchronous designs.

A *standard cell* is an elementary device, such as a logic gate, defined at the chip layout level, with some predefined characteristics that usually comprise cell height and current driving strength. Chip vendors and associated enterprises provide designers with libraries of standard cells and support to define design methods based on them. Cells are characterized in detail, and their standardization facilitates automation of IC design. Most current Application Specific Integrated Circuit (ASIC) System on Chip (SoC) designs use standard cells extensively, e. g. to meet time to market constraints. Indeed, these are the key to speed up the design of high performance ICs and are often referred as the main success factor for the rapid growth of integrated systems technologies [2].

The efficient implementation of asynchronous circuits often requires devices other than those available at current commercial standard cell sets. Moreover, asynchronous design is in fact not a single alternative to the synchronous paradigm, but a collection of methods, several assuming the existence of a specific set of basic devices. Thus, no single set of new cells may be useful for every asynchronous design technique.

This work proposes a set of cells, along with an automated novel design and characterization flow. This enables the construction of asynchronous standard cell based ASICs. The cell set is integrated with the Teak System [3], a synthesis tool for high level asynchronous circuit descriptions. Teak netlists are quasi delay insensitive (QDI) and employ four-phase dual rail protocols. They can be placed and routed with commercial tools.

The rest of the paper is organized in six sections. Section II describes basic concepts. Section III presents related work, while Section IV explores asynchronous circuits related synthesis tools. Section V presents the proposed cell set and the flow to design it. Section VI shows a case study built with the proposed cell set and the results of its simulation. Finally, Section VII draws some conclusions and directions for further work.

II. ASYNCHRONOUS CIRCUITS

A digital circuit is *synchronous* if its design implies the use of a single clock signal controlling all events. Otherwise it is called *non-synchronous*. As a special case, a digital circuit is *asynchronous* when no clock signal is used to control any sequencing of events. They employ explicit handshaking between its components to synchronize,

978-1-4577-1617-1/11/$26.00 ©2011 IEEE

communicate and operate [4]. The resulting behavior is similar to a synchronous system where registers are clocked only when and where necessary. Characterizing an asynchronous design style requires (*i*) the choice of a delay model, (*ii*) a method to encode information and (*iii*) a set of basic devices. Each item is explored in the rest of this Section.

Asynchronous circuits can be classified according to several criteria. One important criterion is based on the delays of wires and gates. The most robust and restrictive delay model is the *delay-insensitive* (DI) model, which operates correctly regardless of gate and wire delay values. Unfortunately, this class is too restricted. The addition of an assumption on wire delays in some carefully selected forks enables the *quasi-delay-insensitive* (QDI) circuit class. Here, signal transitions must occur at the same time only at each end point of the mentioned forks. QDI circuits are quite common, although other models, such as *bundled-data* [4] are still used in specific contexts. This work assumes the use of QDI as target model.

There are different ways to encode data to adequately support delay models. The use of regular binary encoding of data implies the use of separate request-acknowledge control signals. While this makes design straightforward for those used to synchronous techniques, the timing relationship between control and data signals need to be guaranteed at every handshake point, making design of large asynchronous modules unfeasible. As an alternative, DI encodings are robust to wire delay variations, because request signals are embedded within data signals. An example is dual-rail encoding, that uses two wires to represent each bit, and can represent bit values as well as the absence of data. The request signal is computed from the data and therefore demands extra hardware. Clearly, wire overhead of such encodings exceeds Boolean encoding. More efficient DI encodings exist [5].

One fundamental device in asynchronous design that is not available in conventional standard cell libraries is the Muller C-element. Its importance derives from the fact that C-elements operate as event synchronizers. Its output only switches when all inputs are at the same logical value. Its truth table and usual symbols are represented in Figure 1. When inputs (A and B) have the same value, the output (Q) assumes this same value. When inputs are different, the output keeps its previous logic value. Input-asymmetric behavior versions exist as well. With C-elements and standard logic gates it is possible to build other asynchronous primitives as macro blocks such as fork, join and merge [4].

A	B	Q_i
0	0	0
0	1	Q_{i-1}
1	0	Q_{i-1}
1	1	1

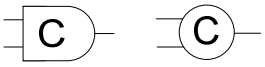

Figure 1 – Muller C-element truth table and used symbols.

III. RELATED WORK

Some works propose methods to design asynchronous circuits with conventional IC design tools and standard cell libraries. For example, Chong et al. [6] suggest a method that enables creating latch-based pipelines and an asynchronous latch controller for stage synchronization. As example design the work proposes an asynchronous FIR filter in a 0.35µm CMOS process. Cortadella et al. present the *desynchronization* paradigm [7] for automating the design of asynchronous circuits from synchronous specifications using synchronous tools. The design flow consists in exchanging each pipeline edge-triggered register by two level-sensitive latches with local handshake units for local synchronization, eliminating the need for a global clock. Combinational logic delays need to be matched in control lines. Validation occurs through desynchronized versions of a DES cryptography core and a DLX processor. The two works do not fully exploit the asynchronous paradigm capabilities, limiting themselves to bundled data protocols, which prevent the use of more robust and performing DI approaches [4]. Moreover, the need to adjust combinational logic delays by hand results in circuits that are hard to reuse in other contexts. Additionally, both works assume the use of conventional standard cell libraries, which may lead to designs that spend more area and/or power.

The asynchronous literature proposes a few standard cell libraries. Ferretti and Beerel suggest the single-track full-buffer (STFB) template library, designed to support high-speed area-efficient asynchronous nonlinear pipeline designs [8]. It employs the TSMC 0.25µm technology and contains cells to support dual-rail and 1-of-3 data encoding circuits, being available through the MOSIS prototyping service. Ozdag and Beerel propose the QDI pre-charged half-buffer (PCHB) template library for asynchronous low-power high perfor-

mance circuits [9]. It also uses TSMC 0.25µm and is available through MOSIS. This library counts 40 dynamic cells to implement function blocks and 10 cells to implement control logic. Gulati and Brunvand describe a specific macro cell library to support the design of asynchronous microengines [10]. The library employs AMI 0.5µm technology, contains 66 cells and was validated through a test chip that implements a differential equation solver microengine. Most cells in the three discussed libraries display a short range of output load capacitance, due to the fact that they deem to validate a template-based design technique or support specific applications. Thus, no fine grain optimization occurs in the design of the libraries themselves. Besides, they limit the design of asynchronous circuits to specific templates. Other asynchronous design styles may require different sets of devices.

To overcome specificity issues, the French labs TIMA and LETI developed a library with cells to support more generic QDI circuit design. Maurine et al. present an STM 130nm technology version of such library [11]. A private communication to authors of this paper informs that this library is currently available in the STM 65nm CMOS process and several chip designs use it. The work proposed here is based on the STM 65nm ASCEnD library, which contains over 250 cells and can be rather easily ported to similar CMOS technologies. It is fully integrated with back-end commercial EDA tools and some front-end asynchronous tools. ASCEnD targets semi custom design of a large range of asynchronous circuits. The cell subset this work addresses enables automatic generation of circuits from high level descriptions and allows designers to experiment with different asynchronous templates and styles like bundled data and dual-rail DI. The cell design flow is fully automated, except for the physical cell layout generation and technology specific requirements.

IV. ASYNCHRONOUS CIRCUITS SYNTHESIS

In the last years, different tools have been proposed to support asynchronous design, testifying the growing interest in asynchronous circuits. Among these, Balsa [4] [12] stands as a comprehensive open source environment. It has been under development at the University of Manchester for a long time. Balsa is both a language to describe asynchronous circuits and a framework to synthesize them. It was first developed in response to a need of a language to replace the Philips Tangram system [13]. The Balsa language can be viewed as an extension of the Tangram language. The system approach is syntax-directed compilation into handshake components. The compilation of a Balsa description is transparent, since language constructs are directly mapped into handshake circuits. The advantage is that it is relatively easy for the user to visualize the circuit-level architecture of a Balsa description. Balsa description changes reflect in predictable changes in the resulting circuit. Additionally, different technologies can be used to synthesize Balsa circuit descriptions, from FPGA to ASICs. Back-end generated gate level netlists can be imported into commercial EDA systems. Recently, another synthesis tool for asynchronous design appeared, Teak [14]. Teak has a new target parameterizable component set and a synthesis scheme that aims at the improvement of circuits described in Balsa. It optimizes Balsa by replacing data-less activation channels with separate control channels. Albeit Balsa allows different data encodings over 2-phase and 4-phase protocols, Teak results are typically 4-phase QDI dual rail asynchronous circuits.

V. CELL SET DESIGN FLOW

To enable implementing netlists generated by Teak in real processes, a very small specific set of standard cells is required. Table 1 show this cell set. All cells are variations of the basic C-element. The proposed flow for each cell appears in Figure 2 and comprises three main steps: specification, design and validation. To synthesize Balsa descriptions, Teak also needs conventional standard cells like ANDs, ORs and AND-OR-INVERTERs.

Table 1 – Proposed cell set logic functions.

Cell	Logic Function
C2	$Q_{i+1} = (A \wedge Q_i) \vee (A \wedge B) \vee (Q_i \wedge B)$
C3	$Q_{i+1} = (A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)$
C2R1	$Q_{i+1} = RST \wedge ((A \wedge Q_i) \vee (A \wedge B) \vee (Q_i \wedge B))$
C1U1	$Q_{i+1} = B \wedge ((A \wedge B) \vee (Q_i \wedge B))$

ASCEnD is fully compatible with the basic STM 65nm standard cell library, making use of both libraries in designs a seamless process. The first step of the proposed flow to design a standard cell is to specify its functionality and electrical requirements.

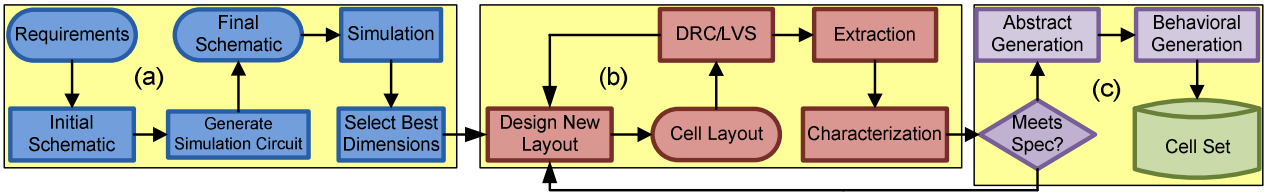

Figure 2 – Standard cell proposed design flow. The three main steps are: (a) Specification, (b) Design and (c) Validation. Actions are represented by boxes, decisions by diamonds, descriptions as rounded corners boxes and the repository as a cylinder.

Functionality is given as an expression defining the output(s) as a function of the circuit input(s), as in Table 1. The electrical requirements consist in the speed for a cell to charge or discharge its output(s). The electrical specification of a standard cell is a tradeoff between area, power and speed. High performance gates require larger transistors and consequently consume more power and demand more silicon area. Usually, different cells with a same logical function support different output driving strengths. In the proposed flow, the designer defines the required cell speed to meet constraints or for compatibility with another gate set.

The next step is to design an initial schematic of the cell. Figure 3 presents this schematic implementation for the logic function of the C2 cell from Table 1, a two input C-element.

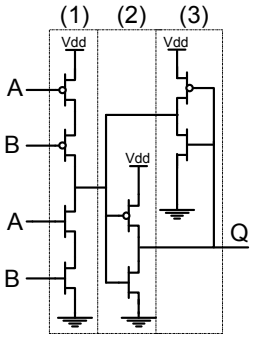

Figure 3 – Example of schematic for the C2 cell, a two input C-element.

The schematic is then exported to a SPICE description. The basic structure of the example C-element (in fact, of any similar cell) can be represented by blocks (1), (2) and (3) of Figure 3. In [15] a similar approach is used to implement C-elements in CMOS technologies. Here, (3) represents the state keeper, (2) is the output driving inverter and (1) is the inverted logic function. The former is usually set to minimum transistor sizes, since it does not influence the switching performance of the output and is used only to keep a static state. Block (2) is responsible for driving (charging/discharging) the output load of the cell. Its transistors size is obtained by simulating an inverter with varying transistor sizing until the required output driving strength is obtained (as defined by the electrical specification). Once the dimensions of the driving inverter are fixed, a specifically tool designed in the context of this work, called Ring Oscillator Generator (ROGen) automatically produces a circuit to simulate for defining the size of the transistors that compose block (1).

The circuit generated by ROGen is the ring oscillator represented in Figure 4, composed by a NAND and 10 of the cells to be sized. The NAND at the left is required to control oscillation.

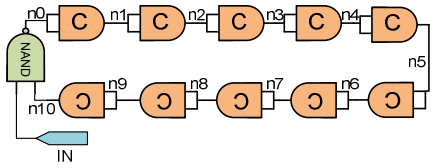

Figure 4 – Example of the gate level view of a simulation circuit generated by ROGen.

Next, extensive simulation defines the number of cell instances in the ring (10) as an amount sufficient to normalize the effect of the NAND in the circuit and allow correct evaluation of the C-elements. The circuit is described in SPICE and variations on the size of the inverted logic transistors are achieved through the use of the SPICE **.alter** function from minimum size transistors to three-finger transistors with maximum size. A finger is the maximum transistor size that can be drawn without using layout folding.

Simulating the SPICE circuit description, information about electrical behavior of each transistor sizing is obtained through the SPICE **.measure** function. This information comprises: dynamic and leakage power of the whole ring, rise and fall propagation times and transition delay of each cell, and operating frequency of the whole ring. Leakage power is the average power consumed from the power source while the circuit is in a static state (**IN=0**) and the dynamic power is the average pow-

er consumption from the source when the ring is oscillating (**IN=1**). A second tool designed in the context of this work is the Cell Specifier (CeS). It is used to analyze the results obtained through simulation and choose the PMOS transistors size that best drives the cell for each NMOS transistor size variation. The result is the set of the best PMOS-NMOS combination for each NMOS size variation. The fastest combination from this set is the best size set and results are presented in a chart.

Once the size of the transistors is defined, step (b) of the Figure 2 flow may occur. From the circuit schematic, the physical characteristics of the standard cell can be designed with the use of a layout editor. The layout must respect design rule check (DRC) and design for manufacturability (DFM) rules along with a layout versus schematic (LVS) check. Once the layout is designed and fully verified, the parasitic effects must be extracted and electrically characterized for the specific process. The electrical characterization is a crucial step in the generation of standard cells, since it defines the behavior of the circuit for different operating conditions and generates information required by EDA tools. After characterization, a cell must be properly verified.

The last flow step consists in the validation of the cell by checking if the information generated during characterization is equivalent to that defined in the specification. Moreover, timing simulation must be carried out for a design composed by a single cell to check if the delay obtained during characterization is correctly annotated and the behavioral description is followed. If the cell passes this verification step, it can be used in a design.

Besides the layout/schematic views, two other views are needed: an abstract one, used by place and route tools and to assemble the circuit on a chip; and a behavioral view, for use in high level simulations. The abstract view is usually generated automatically by a layout tool. Its format is a designer choice. The behavioral view can be captured by some HDL code, like Verilog or VHDL. In Verilog for instance, cell behavior can be implemented through user defined primitives (UDPs) defined as truth tables, and a Verilog module associates the UDP to a cell behavioral view and defines the cell pins. Once the cell is fully designed, all the views are generated and specification is met, it may follow to the cell set repository. With this repository, the designer is able to implement QDI dual rail asynchronous circuits.

VI. CASE STUDY

The cell set specified in Section V was designed using the proposed flow for the STMicroelectronics 65nm, general purpose, standard Vt technology process. A 32-bit public key cryptographic circuit running the RSA algorithm was described in Balsa and implemented using the cells. The circuit was synthesized with Teak, obtaining a netlist. This netlist was automatically exported by Teak to a Verilog netlist composed of a set of components in the target library, in this case, the combination of the conventional STM standard cell library and our asynchronous cell set.

The Verilog netlist was simulated with a VHDL testbench, running multiple encryption and decryption operations. After verifying functional circuit correctness through simulation, the netlist was placed and routed using the abstract views provided by STM and those designed within our cell set. The design uses a total of 132,274 standard cells, for a total area of 0.41 mm^2. The circuit employs five metal layers to route, giving a total wire length of 843,668.855 µm.

To compare results, a synchronous version of this circuit was also implemented. Its area was roughly 15 times smaller than the asynchronous circuit obtained through Teak synthesis, as detailed in Table 2.

Table 2 – A comparison of standard cell area and wire length for asynchronous and synchronous implementations of RSA in the STMicroelectronics 65nm technology.

	Asynchronous RSA	**Synchronous RSA**
Standard Cells	132,274	7,712
Total cell area	0.41mm^2	0.027mm^2
Total wire length	843,668.855µm	57,691.770µm

This overhead is not unexpected, not only because asynchronous logic takes more area, but mostly because asynchronous synthesis techniques are still largely undeveloped. Also, the total power consumption proved to be 3.5 times higher in the asynchronous circuit, as detailed in Table 3. This is due to the elevated leakage and switching power consumption, consequences of such a large area overhead. Internal power increase however, was not so significant, just 1.5 times higher.

After place and route, the delay of each path of the circuit was annotated, using the electrical cha-

racteristics provided by STM and those of the proposed cell set. Next, the post placed and routed netlist was generated and timing simulations for different encryption and decryption operations were conducted to verify the functionality of the mapped circuit. The correct operation of the circuit proved the successful integration of Teak, the ST conventional library and the proposed cell set.

Table 3 – A comparison of power consumption for asynchronous and synchronous implementation of RSA in STMicroelectronics 65nm technology.

	Asynchronous RSA	**Synchronous RSA**
Internal Power	2.625mW	1.816mW
Switching Power	4.834mW	0.515mW
Leakage Power	2.201mW	0.411mW
Total Power	9.66mW	2.742mW

VII. CONCLUSIONS AND FUTURE WORK

The proposed cell set and the validation design showed a successful vertical integration of front-end (Balsa-Teak) back-end (Cadence) and standard cell libraries for achieving asynchronous designs. Current work includes prototyping the test chip to validate the cell set on silicon.

The elevated area consumption revealed in the case study is a consequence of a clear lack of development in asynchronous EDA tool technology. These results are certainly worse if techniques like utilizing conventional logic gates to implement asynchronous logic were employed. For example, a 3-input C-element cell in a 65nm technology has an area of 7.80μm^2, while the required conventional cells to implement the same logic would require 16.64 μm^2, counting cell area alone. In the asynchronous RSA circuit case study, roughly 20% of the standard cells were 3-input C-elements. Implementing this circuit with conventional cells would result in an area overhead of 0.240mm^2, even not taking into account other C-element variations used in the design. Such increase in the area would dramatically affect static power consumption and average latency, easily leading to depletion of the advantages of using asynchronous circuits.

As future work, the authors are undertaking a study to identify components and circuit constructions that cause the immoderate area overhead. This can be achieved by comparing manually implemented asynchronous circuits with those generated by automated tools. By reducing the area overhead of automatically generated circuits, improvement in performance and power consumption can be obtained. Improving EDA tools and specific cells to design asynchronous circuits can present a solution to deal with some of the emerging constraints of deep submicron (DSM) technologies.

ACKNOWLEDGEMENTS

This work is partially supported by the CNPq (under grants PNM 551473/2010-0 and 301599/2009-2), by the CAPES-PROSUP and the CNPq-PIBIC Program. Authors would also like to acknowledge the National Science and Technology Institute on Embedded Critical Systems (INCT-SEC) for the support to this research.

REFERENCES

1. Semiconductor Industry Association. “The International Technology Roadmap for Semiconductors” ITRS 2008 Edition.
2. H. Eriksson et al. “Full-Custom vs Standard-Cell Design Flow – an Adder Case Study”. In: ASPDAC, pp. 507-510, Jan 2003.
3. A. Bardsley et al. “Teak: A Token-Flow Implementation for the Balsa Language”. In: ACSD, pp. 23-31, Jul 2009.
4. J. Sparsø and S. Furber. “Principles of Asynchronous Circuit Design – A Systems Perspective”. Kluwer Academic Publishers, Boston, 354 p., 2001.
5. M. Agyekum and S. Nowick. “An error-correcting unordered code and hardware support for robust asynchronous global communication”. In: DATE, pp 765-770, 2010.
6. K. Chong et al. “A Simple Methodology of Designing Asynchronous Circuits Using Commercial IC Design Tools and Standard Library Cells” In: ISIC, pp. 176-179, Jan 2007.
7. J. Cortadella et al. “Desynchronization: Synthesis of Asynchronous Circuits from Synchronous Specifications”. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 25(10), pp. 1904-1921, Oct 2006.
8. M. Ferretti and P. Beerel. “High Performance Asynchronous Design Using Single-Track Full-Buffer Standard Cells”. *IEEE Journal of Solid-State Circuits*, 41(6), pp. 1444-1454, Jun 2006.
9. R. Ozdag and P. Beerel, “An Asynchronous Low-Power High-Performance Sequential Decoder Implemented with QDI Templates”. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(9), pp. 975-985, Sep 2006.
10. G. Gulati and E. Brunvand, “Design of a Cell Library for Asynchronous Microengines”. In: GLVLSI, pp. 385-389, Apr 2005.
11. P. Maurine et al. “Static Implementation of QDI Asynchronous Primitives”. In: PATMOS, pp. 181-191, 2003.
12. D. Edwards et al. “Balsa: A Tutorial Guide”. 157 p., 2006.
13. K. van Berkel et al. “The VLSI programming language Tangram and its translation into handshake circuits”. In: EURODAC, pp. 384-389, Feb 1991.
14. A. Bardsley et al., “Teak: A Token-Flow Implementation for the Balsa Language”. In: ACSD, pp. 23-31, Jul 2009.
15. M. Shams et al. “Modeling and comparing CMOS implementations of the C-element” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 6(4), pp. 563-567, Dec. 1998.