FACULDADE DE INFORMÁTICA
PUCRS - Brazil
http://www.inf.pucrs.br

Automated versus Manual Design of Asynchronous Circuits in DSM Technologies

Matheus Moreira, Bruno Oliveira,
Julian Pontes, Ney Calazans

TECHNICAL REPORT SERIES

Number 065
July, 2011

Contact:

matheus.moreira@acad.pucrs.br

M. Moreira is a graduate student on Computer Engineering at PUCRS. He is Research Assistant at GAPH research group, funded by an undergraduate research grant from CAPES. His main research interests are asynchronous circuits design and CMOS implementation.

B. Oliveira is an undergraduate student on Computer Engineering at PUCRS. He is a research assistant at GAPH research group, funded by an undergraduate research grant from CNPq. His main research interests are asynchronous circuits design.

J. Pontes holds a M.Sc. degree obtained at the PPGCC/FACIN/PUCRS in 2008. He is Research Assistant at GAPH research group, funded by an undergraduate research grant from CNPq. His main research interests are asynchronous circuits design and low power techniques.

N. Calazans works at the PUCRS/Brazil since 1986. He is a professor since 1999. His main research topics are digital systems design, fast prototyping, and applications, reconfigurable systems, and embedded systems. Dr. Calazans is the head of the Hardware Design Support Group (GAPH) at PUCRS.

Published by PPGCC - FACIN - PUCRS
Av. Ipiranga, 6681
90619-900 Porto Alegre - RS – Brasil

Automated versus Manual Design of Asynchronous Circuits in DSM Technologies

Matheus Moreira, Bruno Oliveira, Julian Pontes, Ney Calazans

Faculty of Informatics, PUCRS, Porto Alegre, Brazil
{matheus.moreira, bruno.scherer}@acad.pucrs.br {julian.pontes, ney.calazans}@pucrs.br

Abstract. This work presents and compares two design flows for implementing asynchronous ASICs. The first flow is fully automated and generates circuits from Balsa descriptions, a high level language dedicated to the description of asynchronous circuits. Balsa descriptions are synthesized through Teak, a synthesis tool that accepts inputs in the Balsa language. The other flow uses conventional EDA tools to synthesize asynchronous circuits initially captured in some typical HDL, such as Verilog or VHDL, where each asynchronous component is described using language constructs. Both flows use the same commercial EDA tools for physical synthesis. An RSA cryptographic circuit was implemented in the STMicroelectronics 65nm technology, using a specially designed standard cell library for asynchronous circuits, using both flows. The resulting circuits were validated through simulation and then compared. This work shows that, albeit manually designed circuits provide higher performance, in terms of power and speed, for a lower area cost, their design and implementation present much higher complexity. In this way, automatically generated circuits are advantageous depending of the requirements and restrictions of an implementation.

Keywords: asynchronous circuits, design flow, automated flow, manual flow.

1. 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. The ITRS estimates that global clock-based integrated circuits (ICs), which comprised 93% of the chips sold worldwide in 2007, will have only 55% of the market by 2022. The remaining 45% will be local handshaking circuits, including clockless or multi-clock ICs. However, the lack of adequate electronic design automation (EDA) tools imposes a great barrier to the design of asynchronous circuits.

Most commercial EDA tools currently focus on purely synchronous designs. Also, typical standard cell libraries, one of the factors responsible for the rapid growth of IC technology [2], do not include components required to implement most of the asynchronous design styles in an off the shelf manner. Therefore, when implementing asynchronous circuits, designers have very limited support when it comes to design automation. Consequently, asynchronous designs are usually implemented through full-custom approaches. However, the semi-custom design of asynchronous circuits through higher levels of abstraction descriptions are becoming a reality as the interest for this paradigm increases.

This work presents a comparison between manual and automated semi-custom design of asynchronous circuits in DSM technologies with current EDA tools support. The objective is to provide a better understanding of the complexity of designing efficient asynchronous circuits in present. Two design flows are proposed and evaluated. Both make use of a specially designed standard cell library to implement asynchronous ICs. For the automated design flow, the Balsa language [3] is used to describe circuits. The Teak System [4] is the tool chosen for synthesizing Balsa descriptions. The generated netlist, composed by handshaking elements, is placed and routed using Cadence Encounter. For the manual design, the circuit is described in some HDL, for instance Verilog or VHDL, where asynchronous gates are manually interconnected in the description. Next, it is synthesized through the Cadence RTL Compiler. In this process, standard cells which were not manually deployed are automatically mapped.

The generated netlist is then placed and routed through Cadence Encounter. Both flows are compared in terms of design complexity, power, area and speed efficiency. More design flows will be evaluated and compared in future works.

The rest of the paper is organized in five sections. Section 2 describes basic concepts on asynchronous circuits, while Section 3 addresses asynchronous design processes. Next, Section 4 explores the automated and manual flows. Section 5 approaches a case study and the results of the comparisons. Finally, Section 6 draws some conclusions and directions for further work.

2. 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 components to synchronize, communicate and operate [3]. 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*) an encoding method; (*iii*) a set of basic devices. All these issues are briefly 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 [3]. Unfortunately, this class is too restricted. The addition of an assumption on wire delays in some carefully selected forks (called *isochronic 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. In practice, isochronic forks are carefully designed inside higher level asynchronous design primitives, so that the primitives themselves become small DI blocks of hardware that can be interconnected mostly without concerns for timing issues. QDI circuits are quite common, although other models, such as *bundled-data* are still used in specific contexts. This work assumes the use of QDI as target asynchronous design style.

There are different ways to encode data to adequately support asynchronous delay models. The use of regular binary encoding of data usually implies the use of request-acknowledge control signals separated from data signals. This is called a *bundled-data* asynchronous design style. 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 difficult, hard to scale and less robust to process variations. As an alternative, DI encodings are robust to wire delay variations, because request signals are embedded within data signals. An example of DI encoding is the dual-rail encoding, that uses two wires to represent each bit, and may represent bit values as well as the absence of data. The request signal is computed from the data and therefore demands extra hardware. Clearly, the wire overhead of dual rail encodings is quite high. More efficient DI encodings exist [5].

Together with ordinary logic gates like AND, OR, NAND and XOR, the C-element is one of the main primitives for many asynchronous circuit design styles. This is because the C-element may operate as a sequential event synchronizer. The truth table and the most commonly used symbols for a C-element with symmetric behavior appear in Figure 1. Its output will only switch when all inputs have the same logical value. When inputs A and B are equal, the output Q assumes this same value. However, when the inputs are different, the output keeps its previous logic value. It is possible to build and use several alternative similar behaviors, by individually negating inputs or the output, increasing the number of inputs and associating differentiated logic behavior to distinct inputs. This last characteristic produces devices that are sometimes called asymmetric C-elements, discussed for example by Thoms in [6].

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

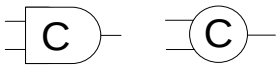


Figure 1 – Muller C-element truth table and some symbols used to represent it.

3. Design of Asynchronous Circuits

Some works propose methods and techniques to design asynchronous circuits using conventional IC design tools and standard cell libraries. For example, Chong et al. [7] suggest such a method, which enables creating latch-based pipelines and an asynchronous latch controller for stage synchronization. As example design the Authors propose an asynchronous finite impulse response (FIR) filter in a 0.35µm CMOS process. In another research effort, Cortadella et al. present the *desynchronization* paradigm [8] for automating the design of asynchronous circuits from synchronous specifications using conventional tools. The design flow consists in exchanging each pipeline edge-triggered register by two level-sensitive latches with local handshaking units for local synchronization, thus 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 the DLX processor. Both these works do not fully exploit the asynchronous paradigm capabilities, limiting attention to bundled-data styles, which prevent the use of the more robust DI styles [3]. Moreover, the need to adjust combinational logic delays by hand along the whole circuit makes the design harder to reuse. Also, both works assume the use of conventional standard cell libraries, which may lead to non-optimal designs in terms of area or power.

EDA tools and standard cell libraries development for asynchronous circuits still lag behind their synchronous counterpart by any measure. However, in the last years, different tools have been developed for this purpose, testifying the growing interest for asynchronous circuits. As instances of this trend, it is possible to cite, the design flows offered by Handshake Solutions or Tiempo, two asynchronous circuits EDA tools vendors. One drawback is that most of these tools are proprietary. However, there is an open source tool for automatically generating asynchronous circuits, developed in the University of Manchester, and called Balsa. Balsa is both a language to describe asynchronous circuits and a framework to synthesize such circuits [3]. The approach adopted by the language is the syntax directed compilation into handshaking components. The compilation of a Balsa description is rather transparent, because 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 described by a Balsa description.

Another option to synthesize Balsa descriptions is Teak [4] a second open source back-end system also designed in the University of Manchester. It differs from the traditional Balsa system flow by using a new target parameterizable component set and a synthesis scheme that aims at the improvement of circuits described in the Balsa language. The tool optimizes Balsa descriptions synthesis by replacing data-less activation channels with separate control channels. Albeit the Balsa System allows different data encodings over different communication protocols, Teak implementations are typically QDI four-phase dual rail asynchronous circuits. Hence, circuits synthesized through this tool are limited to that choice of protocol and data encoding. This work employs the Balsa language and Teak to automatically generate asynchronous circuits.

Finally, asynchronous circuits can also be designed through a schematic-like design approach. For example, Pontes et al. propose two asynchronous networks on chip designs in [9] and [10]. These authors manually design each asynchronous component and use conventional EDA tools to interconnect them in a schematic. The same approach adopted in those works will be used here to manually design asynchronous circuits.

4. Proposed Circuit Design Flows

This section presents two design flows for designing and implementing asynchronous ICs. Both flows make use of a specially designed standard cell library for building asynchronous circuits together with a conventional

commercial standard cell library. Figure 2 (a) depicts the fully automated design flow. The circuit is described in the Balsa language and simulated through the Teak simulator to verify if it meets the specification. Once the correct functionality of the circuit is verified through simulation, the circuit is synthesized by Teak using a standard cell library composed of typical components, provided by the foundry, and specific asynchronous components specially designed for the target technology at the standard cell level. The generated netlist is then simulated with Cadence Incisive, to check its functionality. Once the correct operation is verified, the circuit is placed and routed with Cadence Encounter. The resulting circuit has its paths delays annotated and is timing simulated with Incisive, to validate the circuit in the target technology.

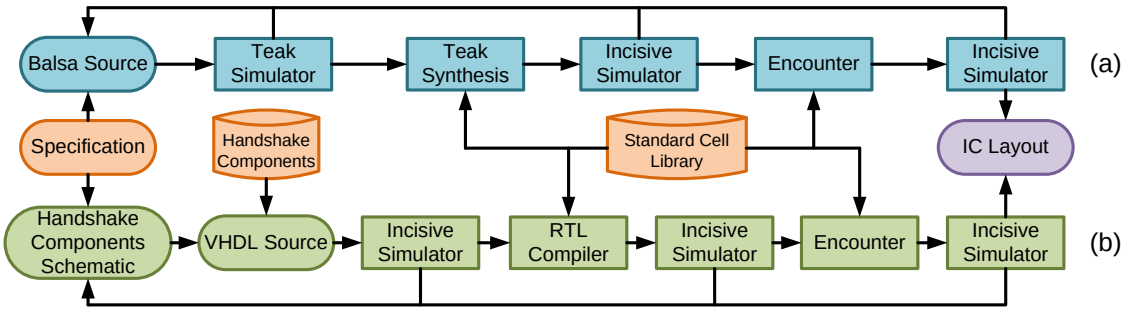


Figure 2 – Asynchronous circuits design flows, fully automated (a) and manual (b).

Figure 2 (b) illustrates the manual design flow. First, the designer draws a schematic for each handshaking component to fulfill the specification. This schematic is then described in VHDL making use of predesigned circuit blocks, which implement asynchronous logic and communication, defined as handshaking components. This description is then functionally simulated with Incisive. Once the description is functionally verified, it is synthesized by the Cadence RTL Compiler. However, this tool is not designed to support asynchronous circuit design. Therefore, the "preserve" property must be addressed so that during the synthesis, asynchronous cells, used to implement the handshaking components, will not be touched. The generated netlist is then simulated without taking into account any delay of wires and gates. Once validation is achieved, the Cadence Encounter tool places and routes the design, and the resulting circuit has its paths delays annotated. A final verification is performed through timing simulation, to verify the correct functionality of the designed circuit.

5. Case Study

In order to compare the efficiency of the design flows, a dual rail asynchronous 32 bit RSA cryptographic core was designed using each flow. Figure 3 shows the block diagram for the asynchronous RSA core.

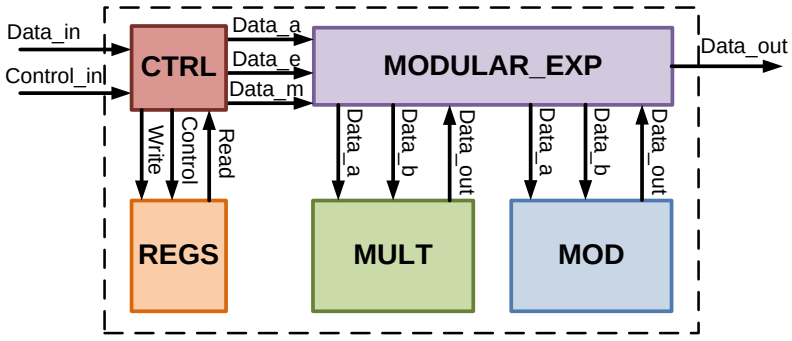


Figure 3 – Asynchronous RSA cryptographic circuit block diagram. Each arrow represents a handshake channel.

The circuit consists of a multiplier, a mod function, a modular exponentiation block, internal registers and a controller. Both the multiplier and the mod function operate shifting bits and performing successive sum operations.

Thus, the implementation is not targeted to high performance operation. The modular exponentiation block is responsible for implementing the RSA encryption/decryption algorithm. To do so, it requires the services of both the multiplier and the mod function. Four internal registers are available to perform the cryptographic operations: “rsa_n”, “rsa_e” and “rsa_d” for the public and private keys and “rsa_b” for the message. The controller is not only in charge of writing the registers, it also feeds the modular exponentiation core with a message and the private or the public key, for encryption or decryption operations, respectively. Table 1 details the operation of the circuit, as a function of its inputs.

Table 1 – Operation of the asynchronous RSA cryptographic circuit as a function of its inputs.

Control_in	***Operation***
0x0	Write “Data_in” value in “rsa_b” register
0x1	Write “Data_in” value in “rsa_e” register
0x2	Write “Data_in” value in “rsa_n” register
0x3	Write “Data_in” value in “rsa_d” register
0x4	Decrypt the message contained in “rsa_m” with the public key (“rsa_n”, “rsa_e”)
0x5	Encrypt the message contained in “rsa_m” with the public key (“rsa_n”, “rsa_d”)

The same functionality was implemented with both asynchronous design flows to compare design efficiency and resulting complexity. Two circuits were generated for the STMicroelectronics 65nm technology, and both were validated through simulation. The average delay to perform a cryptographic operation for the circuit generated through the automated flow was of 34.084µs, while for the manually generated circuit it was 14.902µs. These results were obtained for the same cryptography/decryptography operations scenario. In short, the circuit generated with the manual design flow managed to perform its functionality 229% faster than the one using the automated flow. Additionally, the obtained delays are reasonably large for the target technology. However, this is due to the fact that the circuits perform multiply and mod operations through successive sum operations.

Table 2 presents area results for the physical implementation of the circuits generated with each flow. The automatically generated circuit, as expected, presents a significant area overhead, of roughly 263%. This is due to the fact that synthesis tools for asynchronous circuits are still in their early stages of development, and the circuits that they generate are not really sufficiently optimized. The circuit generated through the manual flow presents better characteristics, but the complexity to design the circuit is huge compared to the automated flow. The total number of asynchronous devices testifies such affirmation. A total of 5795 asynchronous devices were used in 135 handshaking elements. Each of these was manually instantiated in a schematic approach during the design.

Table 2 – Standard cell area and wire length for the RSA cryptographic cores designed through manual and an automatic design flows.

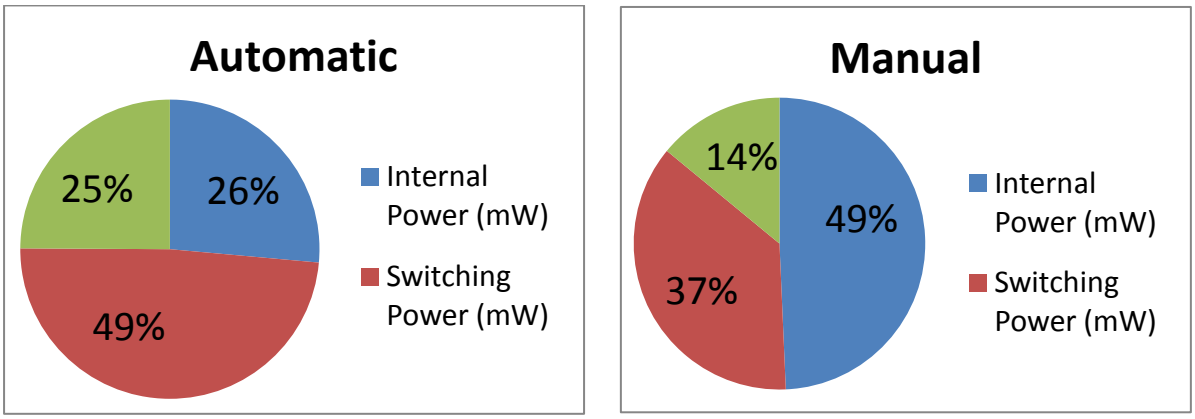
Design Flow	***Automatic Flow***	***Manual Flow***
Number of standard cells	132274	49965
Total cell area (mm^2)	0.411	0.156
Number of standard cells – physical cells	55290	20920
Cell area – physical cells (mm^2)	0.288	0.108
Number of asynchronous devices	17041	5795
Total wire length (mm)	843.669	315.282
Average wire length (mm)	0.015	0.015

Power results also illustrate how the manual design flow generates more efficient designs, as expected. Assuming a multiple cryptographic operations scenario for both circuits, Table 3 presents results obtained for power consumption. In this scenario, 10 messages are encrypted through a public key and then the encrypted message is decrypted through the private key. The automatically generated circuit presented overheads of 112%, 424% and 601% for internal, switching and leakage power consumption respectively, when compared to the manual design. The total power consumption overhead was of 308%.

Table 3 – Power results for the RSA cryptographic cores designed through a manual and an automatic design flow for a multiple cryptographic operations scenario.

Design Flow →	***Automatic***	***Manual***
Internal Power (mW)	2.335	1.103
Switching Power (mW)	4.3	0.82
Leakage Power (mW)	2.201	0.314
Total Power (mW)	8.836	2.165

The power consumption distribution is presented in Figure 4 (a) and (b) for the automatically and manually generated circuits, respectively. As it is clear, leakage power, which gets harder to deal with as CMOS technologies shrink [11], was significantly reduced in the circuit generated with the manual design flow. This is mainly due to the area overhead imposed by the automatically generated circuit. In this way, dynamic power consumption represents a bigger portion of total power consumption for the manually designed circuit. Additionally, the switching power is the main power component for the circuit generated through the automated design flow, while for the manually generated circuit the main power component is the internal power.



(a) (b)

Figure 4 – Power consumption distribution for the RSA cryptographic cores designed with manual and automatic design flows for the multiple cryptographic operations scenario.

One of the reasons for the automatically generated circuit to present such a hogh power consumption is the fact that Teak still cannot choose the best output driving strengths when mapping a cell. In other words, this tool does not consider the load that the cell must drive before it maps it. In this way, the cells can end up being underutilized. Moreover, the cell can also be overused, for instance if the load it needs to drive its output exceeds its capability. In this scenario, the circuit would end up operating at a lower speed.

6. Conclusions and Future Work

The development of EDA tools that support the asynchronous paradigm is still in its early stages of development. The circuits generated through automated tools and asynchronous description languages are still not optimized for specific requirements such as high density, low power or high performance. Manually generated

designs can still provide circuits that are much more optimized for these aspects. However, this work showed that the complexity to manually design asynchronous circuits is much higher which can prevent time-to-market fulfillment.

As ongoing work, the authors are conducting a study to scrutiny Teak synthesis. In this way it will be possible to detect where the tool fails to achieve designs that may compete in quality figure with manual design. An evaluation of other automatic design flows for asynchronous circuits is also expected as future work, to assess the relative efficiency of different available tools and methods. Finally, prototyping of the circuits described here is under way to validate the design flows on silicon and evaluate physical results.

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. Also, Authors would 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] J. Sparsø and S. Furber. “Principles of Asynchronous Circuit Design – A Systems Perspective”. Kluwer Academic Publishers, Boston, 354 p., 2001.
[4] A. Bardsley et al. “Teak: A Token-Flow Implementation for the Balsa Language”. In: ACSD, pp. 23-31, Jul 2009.
[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] W. B. Toms “Synthesis of Quasi-Delay-Insensitive Datapath Circuits” Phd Thesis, University of Manchester, 237 pages, Feb. 2006.
[7] 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.
[8] 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.
[9] J. J. H. Pontes, M. T. Moreira, F. G. Moraes, N. L. V. Calazans “Hermes-A – An Asynchronous NoC Router with Distributed Routing” International Workshop on Power and Timing Modeling, Optimization and Simulation, pp. 150-159, Sep. 2010.
[10] J. J. H. Pontes, M. T. Moreira, F. G. Moraes, N. L. V. Calazans “Hermes-AA: A 65nm Asynchronous NoC Router with Adaptive Routing” IEEE International SoC Conference (SOCC), pp. 493-498, 2010.
[11] N. Ekekwe “Power dissipation and interconnect noise challenges in nanometer CMOS technologies”. IEEE Potentials, Vol. 29(3), pp. 26-31, May 2010.