Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 53(67), Fascicola 1, 2008

Transforming synchronous standard socket designs to GALS designs.

Razvan Jipa[1]

Abstract – The paper presents a solution for transforming a fully synchronous socket based design into a GALS structure that eliminates the need for a global clock, thus reducing the power consumption. The solution presents a set of asynchronous wrappers that transforms the standards synchronous interface into an asynchronous one while provide a stoppable clocking solution to further reduce the consumed power. The solution was proved in simulation using a communication system with one master and multiple slaves.
Keywords: OCP, GALS, asynchronous wrapper

I. INTRODUCTION

The advance in microelectronic technology that follows Moor's law offers the digital circuit designer the possibility to build bigger and more complex, but in the same time creates several challenges when large building large structures. One of these challenges is to distribute a global clock across the design with minimum skew without implying large design efforts and occupied area. A second problem arises from the increase of the power consumption since more and more transistors are integrated on the same area unit. This type of problem is reflected in the design efforts that must compensate for the unwanted effects (IR drop, hot electron, etc.) and in the suitability of the final product for mobile application where the consumption is a major concern.
A possible solution to these problems is the hybrid synchronous-asynchronous circuits or GALS (globally asynchronous locally synchronous) circuits. There circuits were first introduced by Chapiro [1] and they advocate for a synchronous design stile for the modules while the interconnections between them are asynchronous communication channels as described in Fig. 1. This type of design brings the advantage of removing the global clock network and diminishing the clock network power consumption with up to 70% that accounts up to 20% from the total power consumption of a synchronous circuit [2]. An additional advantage of the GALS structures is that the modules are not constrained to work on the same clock frequency, each of them operating at different clock frequencies, thus creating the opportunity for designers to optimize the module for area and not for speed, when speed is not a concern.

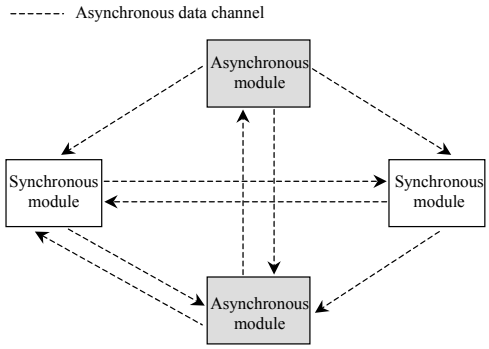

Fig. 1. Asynchronous communications.

II. ASYNCHRONOUS WRAPPERS

Interconnecting synchronous modules with an asynchronous environment represents a problem since the asynchronous modules are not coupled to any timing grid, therefore a metastable behavior may occur on the synchronous module boundary flip-flops. The conventional approach to this problem is to use synchronizers that use double latching technique or FIFO synchronization. This approach although simple in implementation, introduces a communication delay and when a lot of signals are crossing the synchronous-asynchronous boundaries it becomes an unattractive solution.
A different approach is to generate the synchronous module clock locally and be able to stretch the clock pulse to prevent metastability. Such approaches were described in [1] and [3] but they have their drawbacks and limitation and a new type of solution was introduced: asynchronous wrappers.
An asynchronous wrapper embeds the synchronous module transforming the synchronous interface into an asynchronous one. It contains a local clock generator, usually ring a oscillator and asynchronous finite state machines (AFSM) for synchronous-asynchronous transformation. A wrapper implementation with a modular approach designed with minimal latency that uses 4-phase bundled data

[1] Facultatea de Electrotehnica, Departamentul Electronica si Calculatoare, Universitatea "Transilavnia" Brasov
Str. Politehnicii Nr 1, 500024 Brasov, e-mail jipa@vega.unitbv.ro

protocol was proposed by Muttersbach in [4]. (Fig. 2).

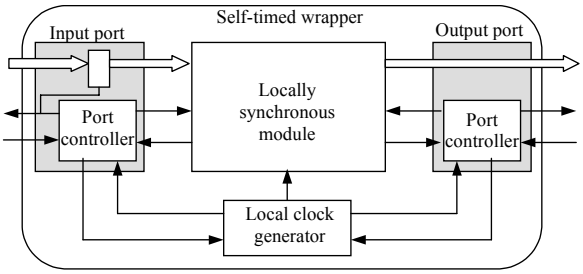

Fig. 2: Asynchronous wrapper.

The input/output ports embed in the asynchronous wrapper are responsible for the data transfer integrity. They ensure the transfer is hazard free and no metastability appears during asynchronous-synchronous boundary occurs, by stretching the locally generated clock such way to meet the setup-hold requirements.

Depending on the function mode of the synchronous module there are defined 2 families of asynchronous ports:

- Demand-type ports. These family ports will keep the clock gated off until the asynchronous transfer is finished. This port is used when the synchronous module can not continue until the data requested is received.
- Poll-type ports. These ports are used when the synchronous module can continue its normal functions (no clock gating) while the asynchronous port handles the data transfers.

Both types of ports require an "enable" signal to be provided in order to start the asynchronous handshake. Additionally, input ports generate back to the synchronous module a signal indicating when a data transfer is in progress – transfActive, as described in Fig. 3.

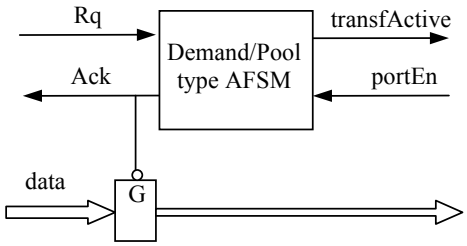

Fig. 3: Input port structure.

A. *Asynchronous finite state machine*

In order to transmit the control signal as fast and efficient as possible the controllers need to be independent of the local clock signal. This is achieved by implementing them as asynchronous finite state machines (AFSM). The behavior of the AFSM is described using extended burst mode description ([5]). From such a description using the 3D tools ([5]) one can obtain a hazard free two-level AND-OR implementation. The extended burst mode description of a poll type input AFSM is described in Fig. 4. The complete description of all AFSM for poll and demand type port can be found in [6]

The extended burst mode is used to describe FSM that are sensible to signal changes not to their levels. To signalize a rising edge a '+' is used while for a falling edge '-' is used. When a signal may have any transition or no transition in the current state and its transitions are verified subsequent states after it was introduced is called a direct don't care and is described using "*" after the signal name.

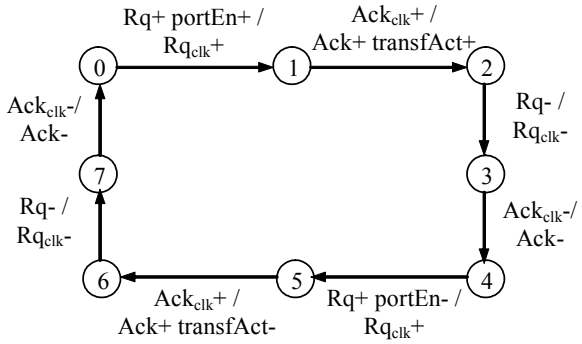

Fig. 4: Poll type input port controller AFSM.

A similar asynchronous state machine is used to perform the synchronization of the "transfer active" signal generated by the input port from the asynchronous domain to the local synchronous domain. The synchronization is required in order to avoid the metastable state that might appear when sample the signal with the local clock. Also the synchronization machine is used to convert from a edge-based event signaling to level-based event signaling as described in Fig. 5.

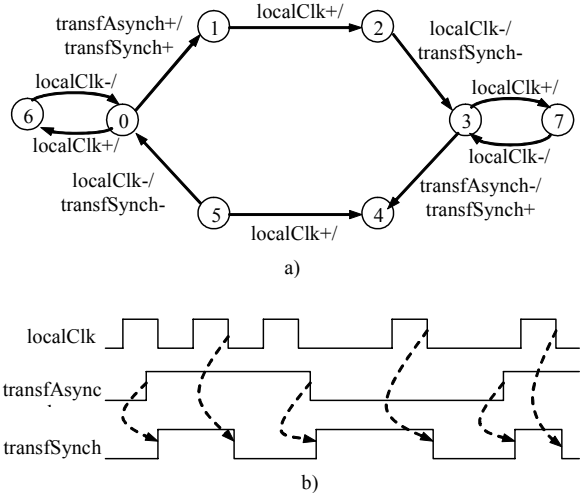

Fig. 5: Transfer active synchronization AFSM (a) and corresponding wave forms (b).

B. *Local clock generator*

The local clock generator is obtained with a ring oscillator and is able to stretch the low level of the clock when a request is made. To implement this feature and avoid ant glitches in output clock a mutual

exclusion module is used. It arbitrates between the locally generated clock and the "clock stretch" request two and if both signals changes at once it "tosses a coin" to decide which signal to pass, but the outputs are guaranteed to be mutual exclusive. The mutual exclusive module is build at transistor level and has the structure described in Fig. 6.

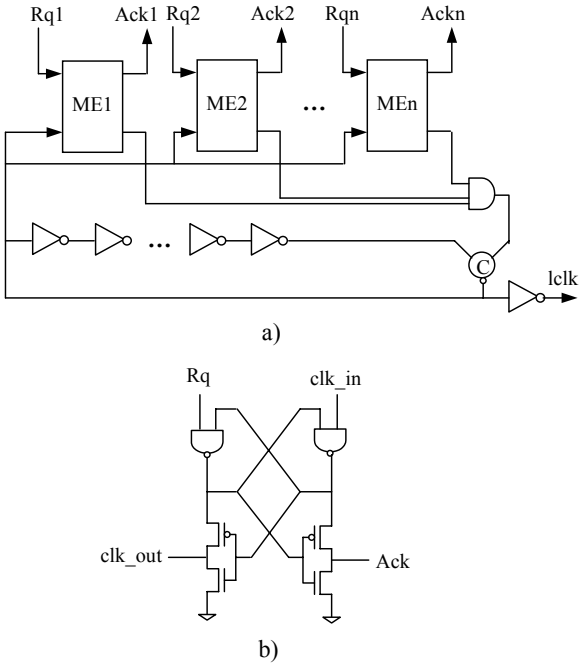

Fig. 6: Local clock generator (a) and mutual exclusion element (b).

Usually an asynchronous wrapper contains more than one port and each port must be able to control the local clock by stretching it. The clock generator implementation that allows multiple clock-stretch requests uses multiple mutual exclusion elements and a Muller C element to avoid any hazards on the output clock ([7]).

III. PROPOSED SOLUTION

Due to the large design effort complex projects often employ IP blocks that needs to be connected between them as easy as possible. To leverage the usage of pre-designed IP block standardized SoC busses and sockets (AMBA, CoreConnect, OCP, Wishbone, SilliconBlackplane, etc) are used.

The solution presented in the paper aims to convert an existent implementation of a synchronous designs based on standard bus/sockets into a GALS design with minimum effort while reducing the overall power consumption. The specific topology that is targeted is a 1-to-many configuration with 1 master and several slaves. The conversion is accomplished by creating a library of specific components containing a set of modified asynchronous wrappers, based on the Muttersbach implementation and local clock generators. The library component is verified trough functional simulation.

The designs targeted to be converted to GALS architecture are standard socket designs that contains one masters and several slave modules connected through a data switch structure like in Fig. 7.

The conversion process from fully synchronous to GALS relies on adding the asynchronous wrappers to the original design and performing minor modifications modification the central data switch like described in Fig. 8. This is similar to encapsulate the original synchronous protocol intro an asynchronous 4 phase bundled data protocol.

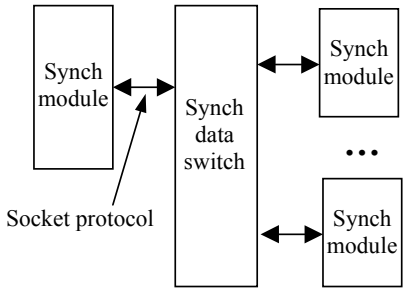

Fig. 7: Socked based designs.

The solution introduces a "gasket" between the synchronous module and asynchronous wrapper that performs the conversion from synchronous protocol signals to asynchronous wrapper interface. This adapter module ensures a synchronous protocol independent implementation for the asynchronous wrapper and do not require any modification of the synchronous module to generate any specific signals required by the wrapper. One can build various sets of "gasket" modules, one for each synchronous socket protocol employed in a specific design.

The original synchronous design transformed into a GALS design with the additional modules is presented in Fig. 8

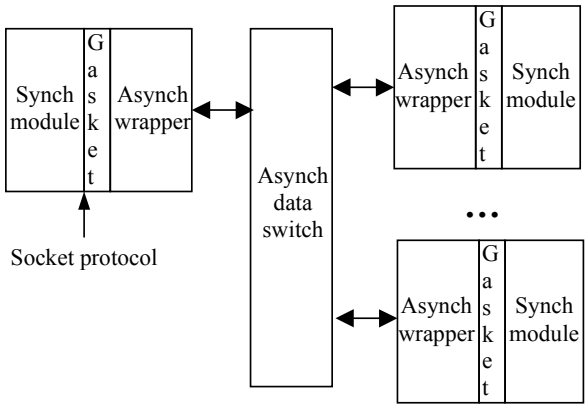

Fig. 8: GALS socked based design.

The central switch that allows 1-to-many communication is the only module that is being regenerated from scratch. This is a very simple operation since the entire switch is based on a 1-to-2 split module that distributes the incoming 4-pahse bundled data protocol to the two target modules. The module is completely combinatorial and performs the selection of the target module is based on the OCP address encapsulated in the asynchronous data bus.

The architecture of the entire switch is very scalable and can accommodate any number of target modules. The structure of the simple 1-to-2 split is presented in Fig. 9

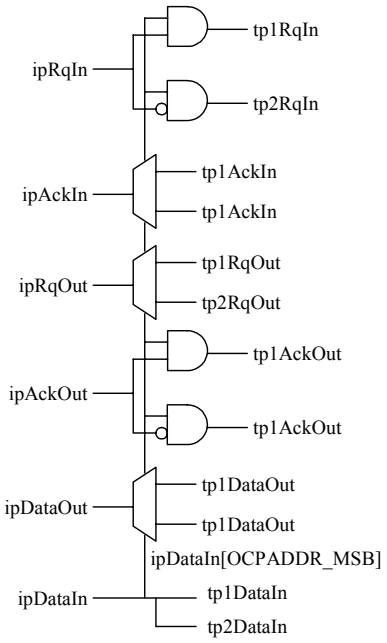

Fig. 9: GALS socked based design.

The entire solution is thought to be presented as a library component that is used as input in a automatic process of converting the synchronous standard bus modules to GALS modules. In particular, for OCP protocol such an automated solution is implemented through a Perl script that requires as input parameters the address and data width and the type of module to be translated (initiator or target). The output of the script is the modified top level where each synchronous module is replaced with its asynchronous one, the asynchronous wrappers build around the synchronous modules and the asynchronous centra switch. This automated solution, presented in Fig. 10 targets the designs that have an internal architecture build around a central switch as described in Fig. 7.

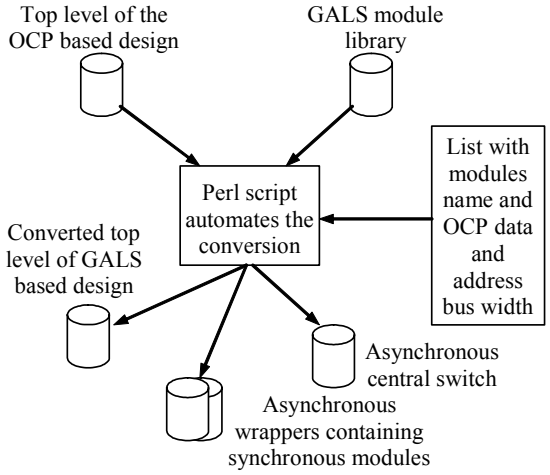

Fig. 10: Automated conversion flow.

IV. MODIFIED WRAPPERS

The implementation of the AFSM proposed by Muttersbach does not exploit the possibility of gating off the local clock when there are no active transactions in process because there was no assumption of the synchronous module protocol. For a standard bus/socket protocol the idle periods are clearly defined and this can be used to further improve the power consumption by shutting off the local clock.

A. Modified clock generator

To be able to shut down the clock when there is no active transaction, the local clock generator has to incorporate an asynchronous state machine that arbitrates between the input port request of enabling the clock when a valid transaction is received and the output port request to shut down the clock when the response was sent.
The modified clock generator structure is described in Fig. 11 while the AFSM extended burst mode description and the AND-OR implementation is described in Fig. 12.

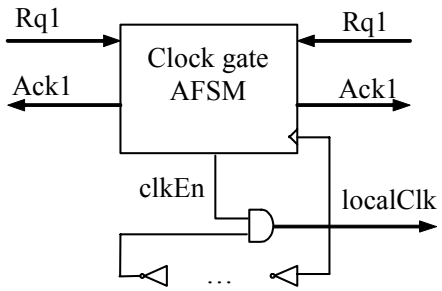

Fig. 11: Local clock generator with clock gate option.

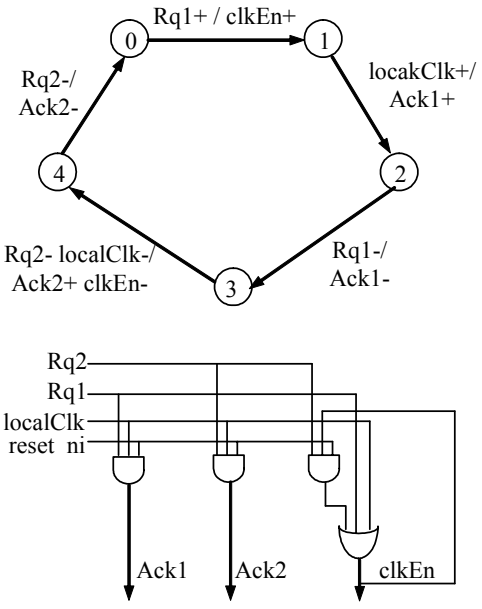

Fig. 12: Modified AFSM and its implementation.

B. *"Gasket" modules*

For the validation of the proposed approach an OCP protocol was selected. The basic protocol interface consists of a master command, address and data lines and slave command accept, data and data response lines ([8]). The test case uses a basic interface and the sequential working mode (not in burst), that means that every command is acknowledged and response to before a new one can be issued. The "gasket" modules need to generate enable signals for the input and output ports based on the OCP interface signal values that are different for the OCP initiator and target modules.

For initiator module the asynchronous port is activated when the OCP command is different from idle. The "port enable" signal is consumed by an 4-pahse asynchronous circuit thus is active on both edges the "gasket" module has to toggle it every time a new transaction is issued. For situations when active transaction are interleaved with idle states on OCP bus a toggle flip flop is enough, but for cases when the transaction are continuous (no idle state) the synchronized version of "transfer active" signal needs to be used. The OCP initiator port gasket implementation that meets the requirement and the associated waveforms are described in Fig. 13.

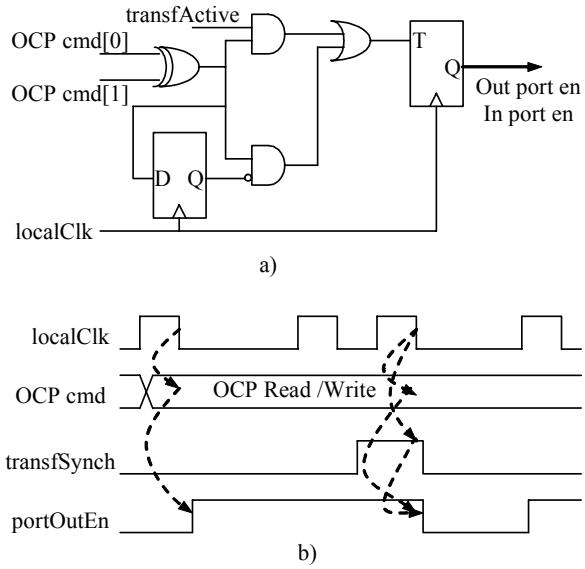

Fig. 13: OCP initiator gasket implementation (a) and the corresponding waveforms (b).

For target port the gasket module has a simple implementation since the port should be enable almost all the time except when the transfer is in progress. The implementation is presented in Fig. 14.

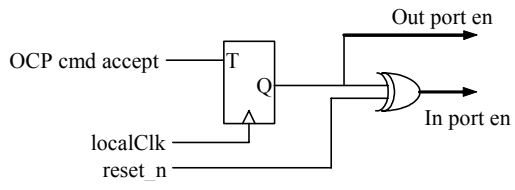

Fig. 14: OCP target gasket implementation.

V. VALIDATION AND RESULTS

As verification frame an OCP based design was selected that comprises one master (JTAG port) and two slave modules (register bank and an SPI controller). All the GALS specific modules (AFSM clock generators, mutual exclusion elements) and the gaskets are described in Verilog together with the rest of the test environment.

The verification sequence contains interleaved access to all slave modules with JTAG clock running at different frequencies from 1 MHz to 100 MHz. The slave modules all had different frequencies:

- SPI interface runs at 40 MHz
- Register back runs at 100 MHz

Besides the functional verification of the proposed solution the simulations process aims to determine an estimate of the power consumption improvement. Starting from the power consumption relation (1) one can obtain the consumed power ratio between the synchronous case and the GALS implementation as a ratio of the activity factors (α) of the two implementations.

$$P_{dyn} = \alpha \cdot f_{clk} \cdot C_L \cdot V_{DD}^2 . \qquad (1)$$

The activity factor (α) is obtained as the weighted sum of the individual activity factors of the design modules where the coefficients are considered area module as percentage of the entire area as described in relation (2)

$$\alpha = \sum A_i(\%) \cdot \alpha_i . \qquad (2)$$

Computing the number of clock cycles the synchronous module would have used for a certain time interval and considering this as an activity factor equal to unity then by counting the clock cycles each GALS module is active and applying formula (3) one can compute the activity factor ratio.

$$\alpha_i = \frac{Cycle\_no_{GALS} \cdot T_{clk\_synch}}{TotalTime} . \qquad (3)$$

The comparison is performed in the best case (for asynchronous version) when transactions are issued at random moments and in worst case when transactions are issued in long continuous bursts with the latency of the transfer varying from no latency at all to 5 cycles latency.

To be able to compute the estimated power saving first the area occupied by each module has to be determined. For this, the synthesis is run on the synchronous design using a 90 nm standard cell library, the same library being used for asynchronous wrapper synthesis.

The synthesis results are described in Table 1: Synchronous design module area.

Module	Area (eq. gates)	Area (%)
JTAG controller	1100	17.6
SPI controller	3000	48.4
Register bank	2100	34

Table 2 while the estimated power savings are presented in Table 2.

Table 1: Synchronous design module area.

Module	Area (eq. gates)	Area (%)
JTAG controller	1100	17.6
SPI controller	3000	48.4
Register bank	2100	34

Table 2. Estimated power savings

Case	Power reduction (%)
Random transaction with no latency	81
Random transaction with 5 cycles latency	64
Burst transaction with no latency	76
Burst transaction with 5 cycles latency	53

The results reveal an impressive power savings compared to synchronous module, but this is the theoretical number since there the simulation models are not a very accurate one from the power consumption perspective. The additional power consumption incurred new 4-phase protocol lines were not considered. However the numbers reveals that serious power savings can be accomplished by using GALS architectures.

A synthesis of the modules that do not contain asynchronous logic specific elements (Muller C element) was performed to determine the area of the asynchronous wrappers proposed. The results reveled that the area penalty incurred by the usage of the asynchronous wrappers is rather small approx 300 equivalent gates for a wrapper. The complete details about modules occupied area computed for an OCP data bus and address bus width of 8 bits is presented in Table 3. The clock generators area depends on the number of the inverters from the ring oscillator. We considered a inverter string with 500 inverters to be able to generate frequencies low enough.

Table 3: Asynchronous module area.

Module	Area (eq. gates)
Gasket (initiator port)	15
Gasket (target port)	9
Asynchronous input port	31
Asynchronous output port	21
Clock generator (stretch)	250
Clock generator (gate)	260
Synchronizer for data transfer active flag	8
Basic splitter (1-to-2)	25

By transforming the synchronous design that occupies 6200 equivalent gates into a GALS design we obtain an area increase of 990 equivalent gates that is 16 % increase area where most of the additional occupied area (770 equivalent gates or 12.5 %) belongs to the local clock generators.

VI. CONCLUSIONS AND FUTURE WORK

The paper introduced a method of transforming a socket based fully synchronous design intro a GALS design with similar functionality with the advantage of reducing the power consumption by employing a new asynchronous wrapper able to stop the local clock between accesses. The proposed method relies on a pre-defined library of components that ensure an easy transition from synchronous to asynchronous protocols as well as a protocol independent implementation of the asynchronous wrappers through the usage of the "gasket" modules.

The area penalty incurred by the usage of the asynchronous wrapper library appears because of the large inverter string used in the local clock generators. This area can be further reduced by a layout design of the inverter string.

The proposed method and the Verilog component library created are an intermediate step toward an automated method of converting socket-based synchronous design intro and asynchronous one aiming to reduce the power consumption.

However, the author recognizes that the power reduction estimation is based on the simulation only and this may not be accurate, and that mode accurate models (SPICE) need to be used. Also the comparison did not take into consideration the real throughput of the GALS implementation, a better metric being the dissipated energy per transmitted byte. This metrics can only be used when accurate power calculations are performed.

The future investigations will focus on delivering accurate dissipated power estimation based on SPICE models and comparison on more adequate metrics.

REFERENCES

[1] D. Chapiro. "Globally-Asynchronous Locally-Synchronous Systems". PhD Thesis, Stanford University, Report No. STAN-CS-84-1026, Oct. 1984.

[2] A. Hemani et al., "Lowering power consumption in clock by using globally asynchronouslocally synchronous design style", Design Automation Conference, 1999. Proceedings. 36th Volume , Issue , 1999 Page(s):873 – 878

[3] K. Y. Yun, R. P. Donohue. "Pausible clocking: A first step toward heterogeneous systems". In Proc. International Conf. Computer Design (ICCD), Oct. 1996.

[4] J. Muttersbach, T. Villiger, W. Fichtner, "Practical design of globally-asynchronous locally-synchronous systems", Advanced Research in Asynchronous Circuits and Systems, 2000. (ASYNC 2000) Proceedings. Sixth International Symposium on 2-6 April 2000 Page(s):52 - 59

[5] K. Y. Yun, D. L. Dill. "Automatic synthesis of extended burst-mode circuits: Part I and II",. IEEE Transactions on Computer-Aided Design, 18(2):101–132, Feb. 1999.

[6] A.R. Ravi “Globally-Asynchronous, Locally-Synchronous Wrapper Configurations for Point-to-Point and Multi-Point Data Communication” MSc Thesis, University of Central Florida, 2004
[7] T. Villiger, H. Kaeslin, F. K. Gurkaynak, S. Oetiker, W. Fichtner: "Self-Timed Ring for Globally-Asynchronous Locally-Synchronous Systems", Proceedings of the 9th IEEE International Symposium on Asynchronous Circuits and Systems, Vancouver, BC, Canada, pp. 141-150 , May 12-16 2003.
[8] ***, “Open Core Protocol Specification”, Version 1.2. Sonics Inc. (www.sonics.com) 2000