Using Commercial FPGAs for Prototyping GALS Circuits

Mehrdad Najibi Kamran Saleh Mohsen Naderi Hossein Pedram Mehdi Sedighi
{najibi, k.saleh, naderi, pedram, msedighi}@ce.aut.ac.ir
Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic)
424 Hafez Ave, Tehran 15785, Iran

ABSTRACT

This paper introduces a methodology for prototyping Globally Asynchronous Locally Synchronous (GALS) circuits on synchronous commercial FPGAs. A library of required elements for implementing GALS circuits is proposed and general design considerations to successfully implement a GALS circuit on FPGA are discussed. The library includes clock generators and arbiters, and different port controllers. Different implementations of these circuits and their advantages and disadvantages are explored. At the end we present a GALS Reed-Solomon decoder as a practical example. The results show that the GALS approach improves the performance of the circuit by 11% and reduces the power consumption by 18.7% to 19.6% considering different error rates. On the other hand, the area of the circuit is increased by 51% which is acceptable considering that a pure synchronous circuit including a central controller is decomposed to generate GALS system and 29% of this overhead belongs to distributing controller in different modules. Deploying better decomposition methods can reduce this overhead substantially.

Keywords

GALS, Prototyping, FPGA

1. INTRODUCTION

The new SoC designs face the challenge of distributing a high-speed low-skew clock in a large die. Proper clock distribution needs numerous buffers and a carefully designed clock tree which introduces a considerable area and power overhead. In a high-performance CPU, near 40% of the total power consumption of the circuit is consumed by the clock [1].

Asynchronous design methodologies can eliminate such overheads naturally by removing the clock signal from the design. However, these circuits are far from being a widely accepted solution yet due to the lack of reliable design tools for asynchronous circuits. The Globally Asynchronous Locally Synchronous (GALS) [2] have emerged to solve clock distribution problem and offer low power and low EMC circuits while allowing synchronous design benefits.

The design of a GALS system generally starts with partitioning a fully designed and tested synchronous circuit. Availability of a large number of verified synchronous IP blocks is a one of the motivations behind this approach. The synchronous circuit should be partitioned into independent locally synchronous islands (LSI). This process includes partitioning the data path, and decomposing the central controller to generate autonomous synchronous islands. While LSIs perform their internal tasks synchronously, they must be equipped with asynchronous wrappers to be able to participate in a globally asynchronous communication.

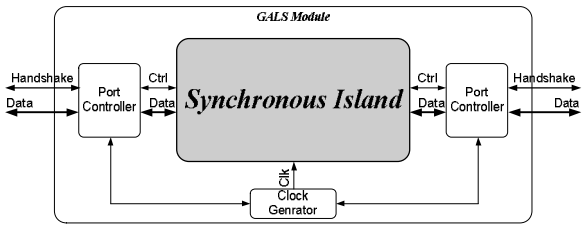

Figure 1. General GALS module

Figure 1 shows an LSI with its corresponding asynchronous wrapper. The wrapper contains a local clock generator and asynchronous input/output controllers to communicate with other GALS or even fully asynchronous modules. One of the most challenging issues in GALS circuit design is the GALS library and the methodology to build asynchronous appearance of synchronous islands. A detailed discussion of a promising solution for this challenge will be provided in subsequent sections of this paper.

On the other hand, FPGAs are widely recognized as an appropriate means for rapid prototyping. However, all

commercially available FPGAs are designed to accommodate purely synchronous circuits. Even though considerable efforts are being made towards the design of a dual-technology synchronous/asynchronous FPGA [3] which allows GALS circuits functional validation, an economically viable version of such prototyping medium remains to be developed. Therefore, any scheme that exploits the existing synchronous FPGAs to implement GALS systems rapidly will greatly reduce these systems' design cycle. Such a scheme will be proposed in the section 2.1 and 3.2 of this paper.

To verify the validity and demonstrate the application of the proposed methodology, a Reed-Solomon decoder was designed and implemented once as a fully synchronous circuit and another time in form of a GALS system. Section 4 includes the details of these implementations along with a performance comparison. A final conclusion of the outcomes of this research effort is provided in Section 5.

2. PORT CONTROLLERS

A globally asynchronous circuit requires asynchronous communication schemes. Asynchronous communications can use two-phase or four-phase handshaking protocol and ports can act in active or passive manner [4]. Another characteristic of an asynchronous communication controller is the encoding mechanism used for data transfer. One of the most famous techniques is the bundled data model, which is used usually for global communication due to its lower overhead. In this case, data lines are grouped with a request or strobe line which shows validity of data on the channel. The delay of the request line must be greater than all the data lines to ensure a safe communication. However, this is not so important because the delay of the control part that generates the request signal is usually larger than the delay of the data path.

There are two main types of port controllers in designing GALS circuits: demand controllers, and poll controllers [5][6][7]. In the demand port controllers, it is assumed that the data being transferred is required immediately after communication. So, in this type of controllers the clock of the LS module should be immediately stopped and it is reactivated when the communication is made. On the other hand, poll type port controller does not stop the clock immediately and so the LS circuit has the opportunity to continue its operation to do some other useful task. This type of port controller targets to gain higher performance, but its efficacy highly depends on the nature of the task and the circuit. So this type of controller can't be used extensively. Due to power saving by their clock-gating property we decided to implement our system with D-ports only.

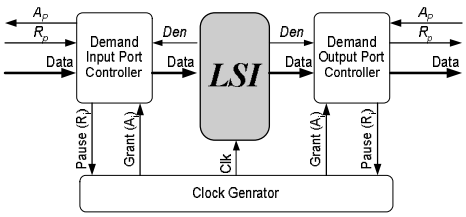

Figure 2. Input and output D-type port controller

Input and output D-port controllers are depicted in figure 2. The asynchronous channel used in this system are push channels, this means that all inputs are passive and all outputs are active [7]. Globally asynchronous communication channels use four-phase bundled data handshake protocol, on the other hand *Den* signal uses two-phase handshake protocol to make the system capable of performing a data transfer in each clock cycle.

An input or output data transfer operation can be summarized as follows: the LS module toggles *Den* signal to indicate a new data transfer request. Upon receiving an event on *Den* signal, port controller activates R_i+, and requests to stop the clock. Local clock generator stops the clock as soon as receiving the pause request and acknowledges the requesting port controller that the clock is safely stopped. Hereafter, the input and output ports behave differently as follows:

- The output port controller activates R_p+ to request a data transfer on its asynchronous data channel. On completion of the four phase asynchronous handshake, the controller inactivates R_i to cancel out its clock pause request. The port controller's job is finished by receiving A_i- and A_p- which indicates that the clock is resumed and communication is finished.
- The input port controller waits to receive R_p+ and A_i+ to be assured that there is a data on the channel and local clock is stopped. It acknowledges by A_p+ and latches the received data. The clock is resumed after the completion of data transfer on the asynchronous channel and the circuit returns to its initial state.

2.1 FPGA Implementation of Input Port Controller

Since the behavior and implementation of D-type input and output port controllers are similar, we describe the implementation of an input port controller without loss of generality. The port controllers are implemented using SI complex gate method by means of Petrify [8]. The STG diagram of the input D-port controller is mentioned in figure 3.

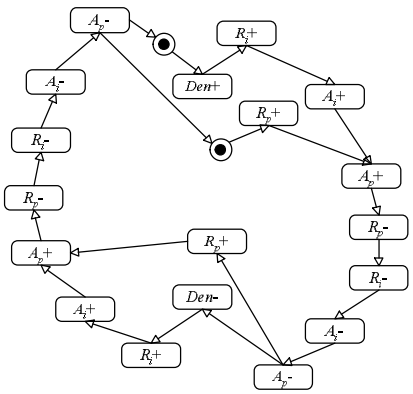

Figure 3. STG diagram of the input D-port

By definition, complex gates should have the following properties: First, they must be hazard free at least for one-input

transitions. Second, paths from all inputs to the output of the gate should have the same delay such that the delay can be lumped into the output. In most commercial FPGAs such as Xilinx Virtex this assumptions are valid for LUTs. Hence, for implementation of port controllers we have assumed that LUTs can be considered as complex gates. The benefit of this approach is that the synthesized asynchronous circuit can be directly mapped into LUTs as building blocks of FPGAs with minimum overhead.

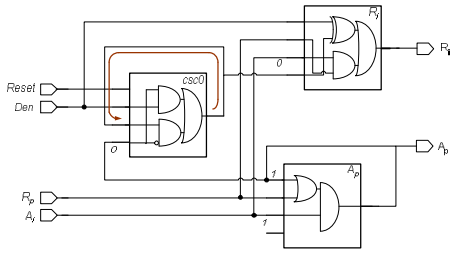

Figure 4. LUT implementation of input D-port

We have synthesized the port controller and mapped the resulting asynchronous circuit to LUTs as shown in figure 4.

The following timing constraints should be satisfied for this implementation to work properly:

First, due to the feedback loop it is necessary to guarantee that after each change in the inputs of the csc0 LUT, the circuit reaches a steady state before any other changes in its inputs are made. In other words, the delay of the feedback loop must be less than the delay of environmental feedback circuitry. Second, the isochronic requirement of SI circuit must be assured.

The first prerequisite is definitely satisfied noting that the environment of the input port controller includes the corresponding output port controller on the other side of the channel plus the local clock generator. This path includes at least 2 LUTs and the necessary routing paths. In our implementation, the feedback loop delay is below 1*ns*. Assuming the delay of 0.4*ns* for each LUT as reported by Xilinx tools and at least 0.2*ns* delay for the shortest path available for routing, delay of the environmental circuitry is at least 1.4*ns*.

The isochronic fork constraint is a consequence of using the SI design methodology, which states that some wire forks must be implemented such that the difference between delays of branches must be negligible compared to gate delays [4]. In general, a fork with all their branches acknowledged on positive and negative edges are not isochronic. The fork is *weak isochronic* if only one of its branches is acknowledged on both positive and negative transitions. The remaining forks are considered as *strong isochronic*. While weak Isochronic forks can be treated like the strong ones but it is sufficient that the delay of the branch which gets acknowledgement be greater than all other branches of the fork. This can simplify the constraint on weak forks substantially.

There is considerable hardware support for implementing isochronic forks in most synchronous FPGAs. In other words, due to their regular structural organization, it is possible to find many forks which meet the isochronic requirement. But from a software point of view, no acceptable support is available to utilize this hardware ability. So we had to use some mapping and placement constraints to realize isochronic forks, because synchronous placement and routing tools have no understanding about the isochronic fork. Among the available mapping and placement constraints, relative location constraints and locking the order of LUT input pins are helpful for implementing isochronic forks and we have used them extensively. Our experiments show that while these implementation constraints can be helpful, the correct realization of isochronic forks can not be guaranteed in this manner, and proper implementation of such forks must be verified after final implementation. We have used Xilinx FPGA Editor to verify the final implementation of Isochronic forks and manually fix the violating ones.

Table 1. Delays and isochronic constraints

Fork Name	Fork Type	Constraint
R_p	Strong Iso.	Equal Delays
A_i	Weak Iso.	$A_{i0} < A_{i1}$
Den	Normal	None
A_p	Weak Iso.	$A_{p0} < A_{p1}$

Table 1 summarizes the isochronic forks and their corresponding constraints in the input D-port controller. It must be noted that all the isochronic forks in our circuit are implemented to have delay deference of at most 40 *ps* which is far less than the delay of LUTs as functional elements.

3. CLOCK GENERATOR

The main idea of the clock generator circuit is to postpone the rising edge of the clock signal until the completion of input-output communication actions [9]. On receiving a request to pause the clock on R_i, the clock generator suspends the next rising edge of the clock and acknowledges the requesting port controller by activating A_i. When the communication is finished the port controller inactivates R_i and the clock generator resumes the clock. The simplest form of clock generator circuit introduced in [5] is shown in figure 5.

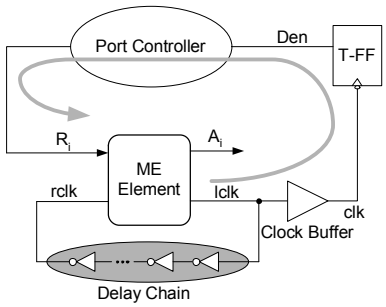

Figure 5. Basic clock generator

The frequency of the clock generator must be adjusted precisely using an adjustable delay chain [10]. It is also notable that the effect of the clock tree and clock buffers may limit the maximum clock frequency which is not considered in details

previously. The clock is paused at its source, while the clock is delayed as it is distributed across the clock domain. When the clock releases the ME element there may still be a clock edge in the buffer tree. Therefore the delay of the chain must be longer than the delay of clock buffer and tree [11].

Moreover, to stop the next rising edge of the clock, ME must block the rising edge of the *rclk*. In this scheme negative edge of the clock signal activates R_i [5]. To ensure that a transition on R_i can stop exactly the next rising edge of the clock, the propagation delay from *lclk*- to R_i+ has to be smaller than low phase of the *rclk*. This path consists of clock tree buffers, a T-flip flop, and the port controller as shown in figure 5. As can be seen in figure 6, the clock buffer causes a skew between the clock of the LSI and the clock observed by the ME element inside the clock generator.

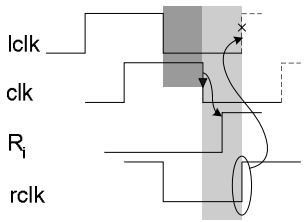

Figure 6. Effect of clock tree delay on timing of R_i signal

The larger the clock tree delay, the less time available for generating R_i. In the extreme case when the clock generator is tuned for maximum frequency, delay chain must have a delay equal to the clock buffer and so there is no time for port controllers to generate R_i. In other words, the growth of the dark gray area in figure 6 leaves less room for light gray area. This means that the maximum reachable frequency is far lower than the upper limit. Therefore, this approach can not be applied to circuits with a large clock tree.

To solve the stated problems we have decided to put clock buffer inside the delay loop of the clock generator. This can also reduce the number of required inverters in the chain. Three possible relative positions of clock buffer and the inverting delay chain are shown in figure 7.

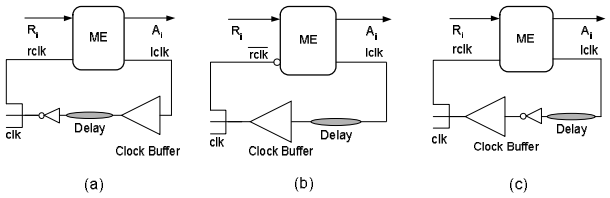

Figure 7. Three possible positions of clock buffer in delay chain

The configuration in figure 7(a) is not a valid solution, because the clock should be directly derived from clock buffer. Both figure 7(b) and figure 7(c) are logically equivalent but the former stretches the low phase of the clock while the latter stretches the high phase of the clock. Therefore, the former design is better due to lower power consumption.

The solution proposed in [5] used negative edge of the clock to generate *Den* signal while it had to use positive edges for other parts of the circuit. It may limit the clock frequency when the logic which generates *Den* falls into critical path which is a common situation in circuits with large and complex state machines. The major motivation for using this approach is that it causes a limited modification of the original state machine. Although, in reality it may reduce the speed of the circuit by dividing the clock phases between the synchronous and asynchronous parts as depicted in figure 8(a). Using the positive edge of the clock for generating *Den* the synchronous circuit can use the whole clock cycle to generate *Den*. This can be seen in Figure 8(b).

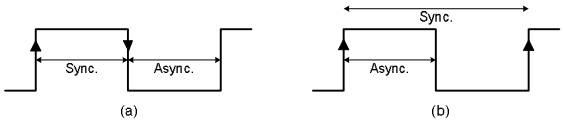

Figure 8. Using positive and negative edges of the clock to generate *Den* signal

To be able to use the positive edge of the clock to generate *Den* signal the state machine must be modified to generate *Den* signal in the pervious cycle of circuit operation whenever a global communication is needed. The new designs yet require meeting the timing constraints of generating R_i in the half period of the clock as shown in figure 9, but in this case the circuit is insensitive to clock buffer and tree delays.

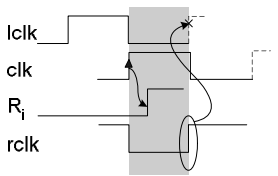

Figure 9. Timing constraints of R_i signal

3.1 Combinational circuits in input and output

Previously proposed methods assume that both sides of all channels in a GALS system must be terminated to registers. This presumption is made because immediately after inactivating R_i, next positive edge of the clock leaves the clock generator and reaches the registers, so the combinational logic may not have sufficient time to complete its operation.

In general, when a synchronous circuit is partitioned into locally synchronous blocks to form a GALS system, the communication channel may fall between a register and a combinational circuit or between two combinational logic networks. In a rare condition, both sides of a channel may be connected to registers. Therefore, assuming no combinational logic in both sides of the channels may necessitate significant changes to the original synchronous design before partitioning for GALS implementation.

The clock generator depicted in figure 7(b) allows the circuit to contain some combinational logic in inputs, because the positive edge of clock released from ME, propagates through

the delay chain before reaching the LS registers. Therefore, the combinational logic in input ports has one half of a clock period to produce its output.

At the outputs, such a time is not available naturally. In a communication channel, the receiver stores data in a latch when it becomes valid and the validity of data is indicated by handshake signals. Data is guaranteed to be valid between R_p+ and A_p-. Hence A_p can be used to sample the data (Figure 10).

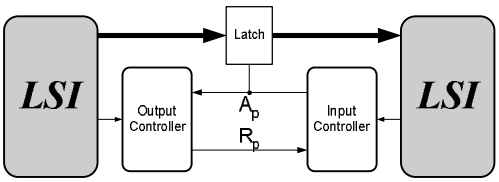

Figure 10. Using A_p to latch the transmitting data

To ensure that the combinational logic in output has enough time to finish, the A_p signal must be delayed. It is obvious that, the sender has no control over the A_p signal. Instead, A_p is controlled by R_p, and R_p is not activated until A_i+. Thus, by postponing A_i, we can push back A_p. Figure 11(a) shows the timing of A_p signal.

To provide enough time for the output ports and allow combinational logic in an output of the locally synchronous island, we can optionally add the C-element of figure 12 to clock generator circuit to suspend the acknowledgement of outgoing communications until the falling edge of clock. We have used an asymmetric C-element, because the output of C-element should rise when both inputs are high, but the input indicated by a small dot, connected to a_i, should be able to reset the output.

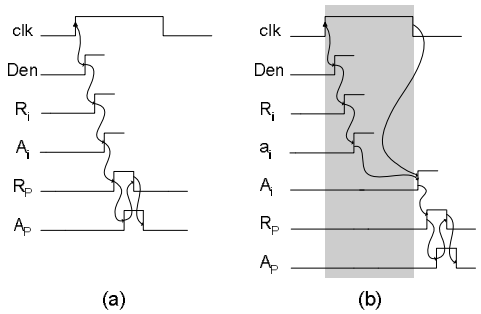

Figure 11. Timing of A_p before and after adding C-Element

In the first half of the clock, the C-Element prevents the rising edge of A_i. Consequently, port controllers accomplish the handshaking in the second half of the clock cycle and the combinational circuit can use the first half of clock to produce its output. It must be noted that this solution can be used optionally when it is required to have combinational logic in the output. Figure 11(a) shows the timing of A_p and Figure 11(b) depicts the timing after adding C-Element. The light gray zone is the time provided for combinational circuits to accomplish their job.

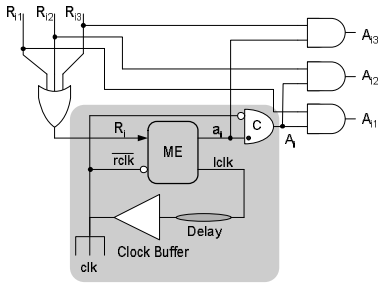

Figure 12. A full clock generator with two outputs and an input port

Note that the input controllers inherently provide the required time for combinational circuits and do not require additional circuitry. So, the A_i signal should be generated separately for input and output controllers. Figure 12 shows a complete clock generator for a circuit with one input and two output ports. R_{i1} and R_{i2} come from output ports and R_{i3} comes from an input port.

A circuit that uses the clock generator shown in figure 12 could have combinational circuits with a delay equal to one half of clock period in its input and output ports. As mentioned, this is useful in partitioning the pre-designed synchronous circuits into separate locally synchronous island. By allowing combinational logic in the inputs and outputs of the LS module, partitioning of synchronous circuit into GALS islands can be done with less constraints.

3.2 FPGA implementation of Clock Generator

All implementations proposed for the local clock generator employed a complete form of the arbiter module. The main application of the arbiter module or mutual exclusion element (ME) is to choose between two independent asynchronous requests which in general may arrive at the same time. If these request signals reach the arbiter in a period of time less than at least one gate delay, metastability [12] may be introduced into the system.

Metastability situation is a condition of the circuit in which a Flip Flop or latch falls into an unknown state. The duration of this situation is unknown but the probability of stabilization grows exponentially by time. In ASICs, the ME element is utilized with an analog filter to eliminate the metastability condition. Many forms of ME element have been proposed for commercial FPGAs [10], but they generally suffer from their own restrictions. These solutions, generally based on a local clock generator; the frequency determines the ME element arbitration resolution. The arbiter resolution is the time needed for the arbiter to stabilize after a change in one of its inputs before the other one can change. Although increasing the frequency of the local oscillator leads to lower probability of failure, the arrival of requests in a time interval less than the arbitration resolution, causes metastability to occur. There is no proper solution yet available in current FPGAs and it seems that the ultimate solution for the metastability problem on

FPGAs is to exploit ME elements in their internal structure as shown in [13] [14].

However, in our GALS implementation method we have only used D-type port controllers and it is not necessary to use the complete form of an ME element. Considering the implementation of the port controllers, it is guaranteed that the request signals R_i and *rclk* (figure 12) reach the arbiter with at least two LUT delays; R_i is generated from *Den* which is synchronous with respect to the *rclk*. Our arbiter is designed as a simple latch, as depicted in figure 13.

The arbiter is composed of LUTs and feedback lines. To gain maximum arbitration resolution, feedback lines must be implemented with minimum possible delays. To meet this requirement we have used mapping and relational placement constraints. Both LUTs are placed in the same CLB and shortest available local routing resources are used which result in a delay of less than 0.2*ns* for the feedback lines. The arbitration resolution in this implementation can be considered as the sum of the feedback delay and one LUT delay that is about 0.6*ns*. For correct operation it is only required that the delay between *clk* and the R_i signals become larger than this value and this is very simple constraint to meet.

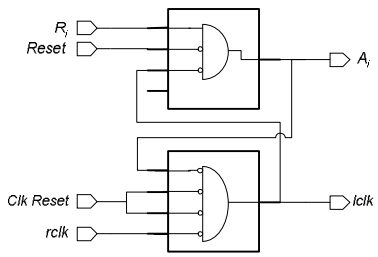

Figure 13. LUT implementation of arbiter

Xilinx Virtex xcv50 FPGA [15] has clock buffer delay of around 3.3*ns*. Considering minimum delay of 1*ns* for other required routing, maximum clock frequency is limited to 116*MHz*. A minimum system consisting of this clock generator and two port controllers can operate at 85*MHz* maximum frequency.

4. RESULTS ANALYSIS

We selected a Reed-Solomon (15,11) decoder [16][17] as a test-bench for our GALS methodology. Error control systems usually consist of error detection and error correction units. Error detection unit is always active and processes each input codeword. On the other hand, the error correction unit is only active when an error occurs. Because of this inherent property, these types of circuits are suitable for low-power implementation. By disabling the error correction unit when the codeword is error-free considerable amount of power can be saved.

The RS decoder receives codewords of 60-bit length as 4-bit symbols belonging to Galois Field 2^4. The circuit can correct two symbols (8-bit) errors. The block diagram of the circuit is shown in figure 14.

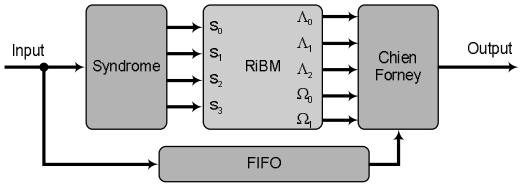

Figure 14. Reed-Solomon block diagram

The RS circuit is functionally partitioned into four blocks as shown in figure 14. We used the same partitioning for the GALS implementation and so the central controller of the original circuit is replicated and partially simplified to make locally synchronous modules independent of each other. To evaluate the efficiency of our method, we designed and synthesized both GALS and synchronous circuits using the same synthesis tool. Table 2 shows the speed of the original circuit and separated LSIs.

Table 2. Speed of the GALS and synchronous RS

Circuit		Clock Frequency
Synchronous RS(15,11)		55.5 *MHz*
GALS RS(15,11)	Syndrome	72.0 *MHz*
	RiBM	82.5 *MHz*
	Chien-Forney	81.0 *MHz*
	FIFO	105.2 *MHz*

Simplifying LSI controllers causes an increase in the speed of each LS module compared to the original circuit. From the performance point of view, the input rate of the GALS circuit, considering the port controller overhead is equivalent to a 61*MHz* synchronous circuit, which shows an 11% increase in circuit performance.

Table 3 shows the area of the synchronous and GALS implementation. As it can be seen the size of the circuit is increased by 51%. Asynchronous wrappers are responsible for 22% of this overhead and 29% is caused by replication of the controller.

Table 3. Area of the GALS and synchronous RS

	Sync. RS(15,11)	GALS RS(15,11)		
		LS	Wrapper	Total
LUT	**440**	574	71	**645**
DFF	**116**	149	51	**200**

Considering the normal error rate of a communication channel between 10^{-7} and 10^{-4}, the GALS circuit consumes 18.7% to 19.6% less power than synchronous circuit, which consumes 74*mW*, at the same operating speed. Principal reason of this power reduction is that the RiBM circuit which is the largest part of the system is suspended approximately 80% of the time.

5. CONCLUSION

This paper explored the timing and functional issues of a clock generator for implementation of GALS systems. We have shown that a GALS system may be vulnerable to problem of

multiple clock edges in the clock tree if the delay of clock buffer is not considered. Then we proposed to put the clock buffer inside the delay chain to eliminate the risk of unwanted clock edges. In addition, we proposed a method to allow the LSIs to have combinational logic in their input and output. This feature is necessary to ease the partitioning of a pre-designed synchronous system.

We have shown that the implementation of isochronic forks on synchronous FPGAs is possible, and consequently we presented an implementation of asynchronous port controllers and the clock generator. This library of components helps to prototype the GALS systems on existing synchronous FPGAs, and to use the supportive features of commercial CAD tools.

As an example, we demonstrated the implementation of a fully functional GALS RS decoder on a commercial synchronous FPGA. The results show that the circuit is both faster and more power-efficient than its fully synchronous counterpart. The only drawback is the area overhead of the GALS system. Deploying better decomposition methods can reduce this overhead substantially.

6. REFERENCES

[1] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Öberg, P. Ellervee, and D. Lundqvist. Lowering power consumption in clock by using globally asynchronous, locally synchronous design style. In *Proc. ACM/IEEE Design Automation Conference*, pp. 873-878, June 1999.

[2] D.M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, 1984.

[3] S. Hauck, G. Borriello, S. Burns, and C. Ebeling. "*MONTAGE: An FPGA for Synchronous and Asynchronous Circuits*". In Workshop on Field Programmable Logic and Applications, 1992.

[4] J. Sparso, S. Furber, "*Principles of Asynchronous Circuit Design – A System Perspective*", Kluwer Academic Publishers, 2002.

[5] J. Muttersbach, T. Villiger, and W. Fichtner. Practical Design of Globally-Asynchronous Locally-Synchronous Systems. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 2000.

[6] D.S. Bormann and P.Y.K. Cheung. Asynchronous Wrappers for Heterogeneous Systems. In *Proc. International Conference on Computer Design (ICCD)*, pp. 307–3014, October 1997

[7] S. Moore, G. Taylor, R. Mullins, and P. Robinson. Point to point GALS interconnect. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 69-75, April 2002.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *IEICE Trans. on Information and Systems*, pp. 315-325. March 1997.

[9] K.Y. Yun and A. E. Dooply. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, December 1999.

[10] G. Taylor, S. Moore, S. Wilcox, and P. Robinson. An on-chip dynamically recalibrated delay line for embedded self-timed systems. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 2000.

[11] A. Royal, P. Y. K. Cheung. Globally Asynchronous Locally Synchronous FPGA Architectures. In *Proc. Field Programmable Logic and Application, 13th International Conference (FPL 2003)*, LNCS 2778 Springer, pp. 355-364, September 2003.

[12] Ø. Damhaug, and T. Njølstad. Arbitration and Meta-Stability Management in Globally Asynchronous Locally Synchronous Circuits. In *Proc. 5th NORDIC Signal Processing Sym.*, October 2002.

[13] R. E. Payne. *Self-Timed Field Programmable Gate Array Architectures*. PhD thesis, University of Edinburgh, 1997.

[14] Catherine G. Wong, Alain J. Martin, and Peter Thomas. An Architecture for Asynchronous FPGAs. *Proc. IEEE Int. Conf. on Field-Programmable Technology (FPT)*, December 2003

[15] http://www.xilinx.com

[16] K. Saleh, "Hardware Architectures for Reed-Solomon Decoders", *Technical Report*, Amirkabir University of Technology, January 2003

[17] S. Wicker, V. K. Bhargava, "*Reed-Solomon Codes and Their Applications*", IEEE Press, 1994