A Novel Clock Generation Scheme for Globally Asynchronous Locally Synchronous Systems: An FPGA-Validated Approach

Kamran Saleh Mehrdad Najibi Mohsen Naderi Hossein Pedram Mehdi Sedighi

{k.saleh, najibi, 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 focuses on a clock generation scheme for implementation of GALS circuits on commercial FPGAs which are mostly synchronous. Previously overlooked timing problems of existing pausible clock generators are explored and a novel clock generator is introduced. To validate the proposed solution we implemented the clock generator and a simple port controller on FPGA. In addition, general design considerations to successfully implement a GALS circuit on FPGAs are discussed. At the end we present a GALS Reed-Solomon decoder as a practical example. The results show that the GALS approach can improve both power and performance of the system using different partitioning strategies.

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles–*Sequential circuits;*
B.7.1 [**Integrated Circuits**]: Types and Design Styles-*Gate arrays*

General Terms

Design

Keywords

GALS, On-Chip Clock Generation, 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. For example, in a high-performance CPU about 40% of the total power of the circuit is consumed by clock [1].

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

The general architecture of a GALS circuit is shown in figure 1 which consists of a Locally Synchronous Island (LSI) surrounded by 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.

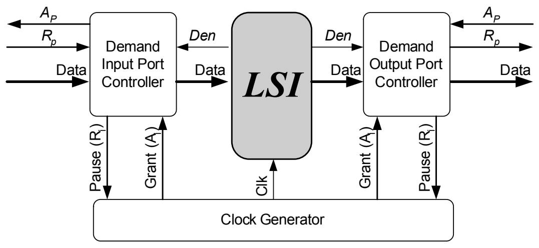

Figure 1. An LSI with corresponding wrapper

In general, a GALS system contains multiple modules; each one working with its own clock rate. In addition, a GALS module should be able to stop its local clock to prevent metastability whenever a communication is in progress. So, using off-chip clock generators is impossible in this case. Many of proposed on-chip clock generators for GALS systems are based on pausible clocking [3][4][5]. However, these schemes overlooked some timing problems that must be considered in practical designs.

Section 2 deals with detailed timing problems with pausible clock generators and their interactions with the LSI and port controllers. In this section we will show that the previously proposed clock generators may fail to stop the clock safely if the delay of the clock distribution network is not considered.

In Section 3, we will propose a solution to fix these timing problems. Furthermore, our solution allows combinational circuits in inputs and outputs of LSI modules which is greatly helpful when partitioning a pre-designed synchronous circuit into separate locally-synchronous islands.

Section 4 is dedicated to the implementation of asynchronous wrappers on commercial synchronous FPGAs. This section illustrates the problems that a designer may encounter during implementation of an asynchronous circuit on a totally synchronous platform. Finally, this section provides proper solutions to all these problems and presents the FPGA implementation of a simple port controller along with a clock generator.

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 the form of a GALS system. Section 5 includes the details of implementations along with a performance comparison. A final conclusion of the outcomes of this research effort is provided in Section 6.

2. CLOCK GENERATORS ISSUES

The main idea behind on-chip pausible clock generation is to postpones the rising edge of the clock signal until the completion of input-output communication actions [6]. The simplest form of clock generator circuit introduced in [3] is shown in figure 2. The LSI informs the port controller of an input/output request by toggling *Den* signal. To prevent metastability during asynchronous communication, the port controller requests to stop the local clock using R_i. Upon 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 deactivates R_i and the clock generator resumes the clock.

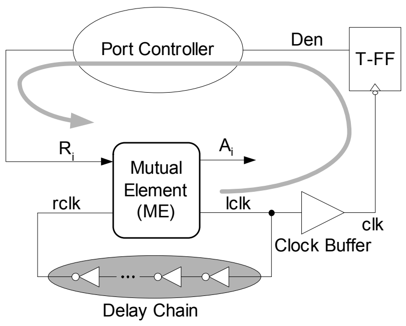

Figure 2. Basic clock generator

Although this scheme is widely accepted as a general template for clock generator in GALS systems, there is two considerable problems with it; clock distribution delay is not considered, and combinational logic in input and output is not permitted. The following two subsections state these problems.

2.1 Clock Distribution Delay

When the clock generator stops the clock, there may still be clock edges in the buffer tree and therefore, the delay of the chain must be longer than the delay of clock buffer and tree [7].

Moreover, to stop the next rising edge of the clock, ME must block the rising edge of the *rclk*. If we use negative edge of the clock to activate R_i as proposed in [3], 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 2. As can be seen in figure 3, the clock buffer causes a skew between the clock of the LSI and the clock observed by the ME element inside the clock generator.

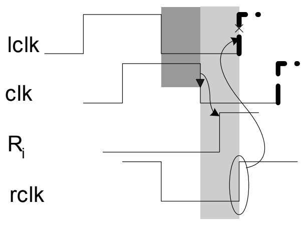

Figure 3. Effect of clock tree delay on timing of R_i signal

The larger the clock tree latency, 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, growth of dark gray area in figure 3 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.

2.2 Combinational Logic in I/O ports

Design of a GALS system usually starts with partitioning a fully designed synchronous circuit. Availability of a large number of verified synchronous IP blocks is a one of the motivations behind this approach. This process includes partitioning the data path, and decomposing the central controller to generate autonomous synchronous islands. Previously proposed GALS architectures inherently 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.

3. A NEW CLOCK GENERATOR

To remove the skew caused by clock distribution network, we put clock buffer inside the delay loop of the clock generator. This may 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 4.

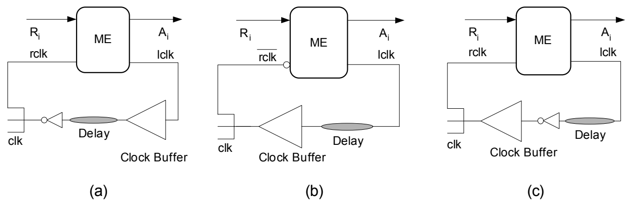

Figure 4. Three possible positions of clock buffer in delay chain

The configuration in figure 4(a) is not a valid solution, because the clock should be directly derived from clock buffer. Both figure 4(b) and figure 4(c) are logically equivalent, but the former stretches the low phase of the clock while the latter stretches the high phase of the clock. We have chosen the former design because stretching low phase of clock is more compatible with common sense.

Another problem with using negative edge of the clock to generate *Den* signal and using positive edges for other parts of the circuit is that it may limit the clock frequency when the logic which generates *Den* falls into critical path. This is a common situation in circuits with large and complex state machines and may reduce the speed of the circuit by dividing the clock phases between the synchronous and asynchronous parts as depicted in figure 5(a). Using 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 5(b).

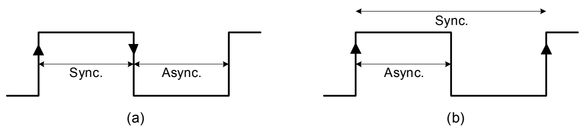

Figure 5. Using positive and negative edges of the clock to generate *Den* signal

The only motivation for generating Den using negative edge of the clock is that it causes a limited modification of the original state machine to handle *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 6, but in this case the circuit is insensitive to clock buffer and tree delays.

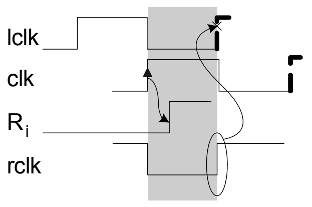

Figure 6. Timing constraints of R_i signal

The clock generator depicted in figure 4(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 data (Figure 7).

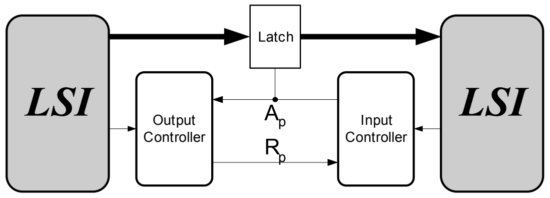

Figure 7. Using A_p to latch the transmitting data

To ensure that the combinational logic in output has enough time to finish, the A_p signal must be delayed. Obviously, 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 8(a) shows the timing of A_p signal.

To provide enough time for the output ports and allow combinational logic in the output of the locally synchronous island, we can optionally add the Muller C-element [8] to the clock generator of figure 9 for suspending 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.

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 8(b) depicts the timing of A_p after adding C-Element. The light gray zone show the amount of time provided for combinational circuits to accomplish their job.

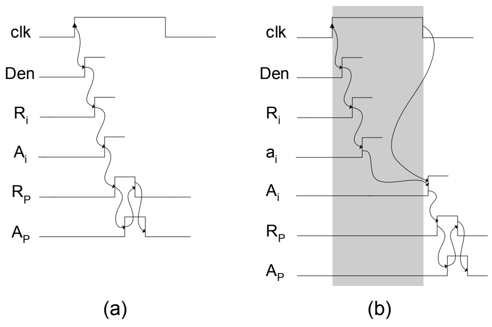

Figure 8. Timing of A_p before and after adding C-Element

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 9 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.

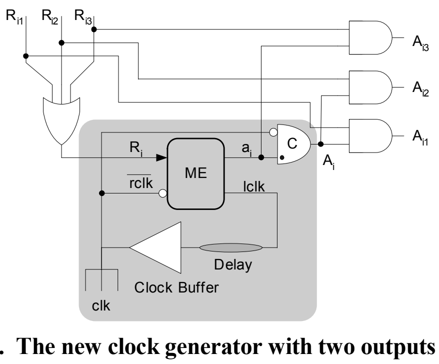

Figure 9. The new clock generator with two outputs and one input port

A circuit that uses the clock generator shown in figure 9 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 a pre-designed synchronous circuit 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.

4. FPGA IMPLEMENTATION

As stated before, GALS wrappers are asynchronous in nature while available FPGAs are built for synchronous circuits. Therefore, special consideration must be applied to correctly implement asynchronous requirements. Different asynchronous design methodologies require their own specific constraints to be correctly implemented on FPGAs. We have used Speed Independent (SI) [8] methodology for designing asynchronous parts of the wrappers. Petrify [9] is utilized to realize these parts using the complex gate implementation method. The benefit of this approach is that the synthesized asynchronous circuit can be directly mapped into Look-Up Tables (LUTs) as building blocks of FPGAs with minimum overhead.

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 family, these assumptions are valid for LUTs [15]. Hence, for implementation of port controllers we have assumed that LUTs can be considered as complex gates.

4.1 Clock Generator

Most 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 [10] may be introduced into the system.

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 [11], but they generally suffer from their own restrictions. These solutions are generally based on a local clock generator and its frequency determines the ME element resolution. Although increasing the frequency of the ME 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 [12][13] [14].

However, in our GALS implementation method we have only used D-type port controllers (will be discussed later) 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 9) reach the arbiter with at least two LUT delay separation because R_i is generated from *Den* which is synchronous with *rclk*. Our arbiter is designed like a simple latch, as depicted in figure 10.

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 (Configurable Logic Block) and shortest available local routing resources are used which result in a delay of less than 0.2*ns* for the feedback lines (reported by Xilinx FPGA Editor). 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 constraint can be met simply.

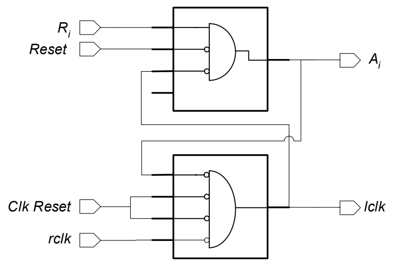

Figure 10. LUT implementation of arbiter

Considering that the Xilinx Virtex xcv50 FPGA [15] has clock buffer delay of around 3.3*ns* and 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.

4.2 Port Controllers

There are two main types of port controllers: demand controllers, and poll controllers [3][4][5]. 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.

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. *Den* signal is used by LSI to inform the port controller of a new communication request.

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.

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 pause request (R_i+), and requests to stop the clock. Local clock generator stops the clock as soon as receiving R_i+ and acknowledges the requesting port controller that the clock is safely stopped.

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. The STG diagram of the input D-port controller is mentioned in figure 11. We have synthesized the port controller by means of Petrify and mapped the resulting asynchronous circuit to LUTs as shown in figure 12.

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.

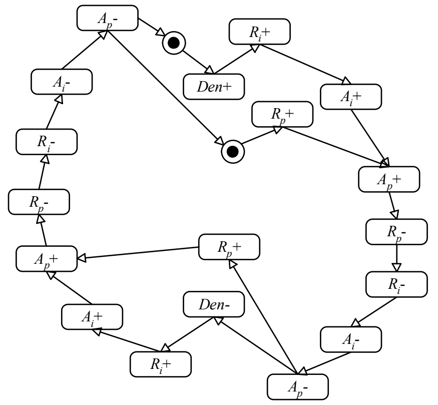

Figure 11. STG diagram of the input D-port

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*.

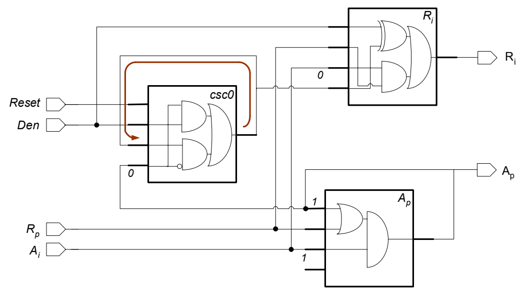

Figure 12. LUT implementation of input D-port

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 [8]. In general, a fork is not isochronic iff all of its branches are acknowledged on positive and negative edges. In other words, the fork being isochronic means that some transitions on one of the branches of the fork are not acknowledged by any other transition in system. In the circuit of figure 12 among existing forks, R_p, A_i, and A_p are isochronic.

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 FPGA place and route tools don't serve acceptable support to utilize this hardware ability. So we had to use some mapping and placement constraints to realize isochronic forks. 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 utilized them extensively. We have used Xilinx FPGA Editor to verify the final implementation of Isochronic forks and manually fix the violating ones. 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.

5. RESULTS ANALYSIS

We selected a Reed-Solomon (15,11) decoder [16] 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. Block diagram of the circuit is shown in figure 13.

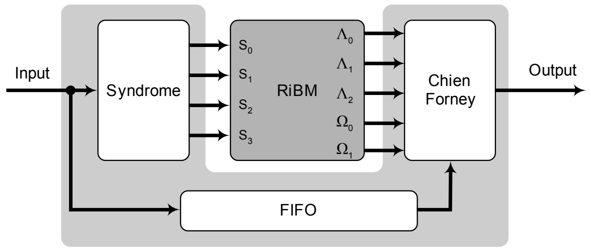

Figure 13. Reed-Solomon block diagram

The RS circuit is functionally partitioned into four blocks as shown in figure 13. 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. Later, we grouped Syndrome, FIFO, and Chien-Forney modules (the shaded area in figure 13) to a single LSI to reduce GALS overheads because these circuits have a lot of communication with each other and are active simultaneously. Table 1 shows the speed of the original circuit and separated LSIs for Xilinx Virtex xcv50.

Table 1. Speed of the GALS and synchronous RS

Circuit		Clock Frequency
Synchronous RS(15,11)		55.5 *MHz*
GALS[1] RS(15,11)	Syndrome	72.0 *MHz*
	RiBM	82.5 *MHz*
	Chien-Forney	81.0 *MHz*
	FIFO	105.2 *MHz*
GALS[2] RS(15,11)	Syndrome FIFO Chien-Forney	57.8 *MHz*
	RiBM	82.5 *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 handshake rate of the GALS[1] circuit, considering the port controller overhead is equivalent to a 61*MHz* synchronous circuit, which shows an 11% increase in circuit performance. However, GALS[2] implementation has an equivalent frequency of 53*MHz* which is close to the synchronous circuit.

Table 2 shows the area of the synchronous and GALS implementations. As it can be seen the size of the circuit is increased by 51% in GALS[1] and 35% in GALS[2]. In the former case, asynchronous wrappers are responsible for 22% of this overhead and 29% is caused by replication of the controller while in the latter case, 15% is by wrapper and 20% is by controller replication.

Table 2. Area of the GALS and synchronous RS

Circuit		LUT	DFF
Sync		**440**	**116**
GALS[1]	LS	574	149
	Wrapper	71	51
	Total	**645**	**200**
GALS[2]	LS	542	127
	Wrapper	43	42
	Total	**585**	**169**

Considering the normal error rate of a communication channel between 10^{-7} and 10^{-4}, the GALS[1] circuit consumes 18.7% to 19.6% less power than synchronous circuit, which consumes $74mW$, at the same operating speed. This power saving is 34.1% to 36.2% in GALS[2] implementation. 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. It is nice to mention that one may achieve these results by using clock gating methods; however, clock gating may increase the clock skew. The main purpose of GALS design is to cope with the problem of clock skew in large circuits by partitioning a large clock domain into smaller ones.

6. CONCLUSION

GALS methodology can achieve some power improvements in the circuit while smoothing the clock skew problems. However, as we have shown, a GALS system may be vulnerable to the problem of multiple clock edges in clock tree if the delay of clock buffers is not considered. This paper explored the timing and functional issues of a clock generator for implementing GALS systems. We proposed to put the clock buffer inside the delay chain of the local clock generator to eliminate the risk of unwanted clock edges. In addition, we suggested 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.

In order to validate our approach, we presented a prototype of asynchronous port controllers along with a clock generator on an FPGA. To be able to use synchronous FPGAs to implement asynchronous wrappers, we used mapping and placement implementation constraints to meet the isochronic fork requirement. Proposed library enabled us 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 Reed Solomon decoder on a commercial synchronous FPGA with better power and performance characteristics comparing with the pure synchronous circuit.

7. REFERENCES

[1] A. Hemani, T. Meincke, et al. 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] 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.

[4] 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

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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