Burst-Mode Asynchronous Controller Implementation on FPGA Using Relative Timing

Jotham Vaddaboina Manoranjan Kenneth S. Stevens
Department of Electrical and Computer Engineering
University of Utah

***Abstract*—A new methodology for the design of glitch free burst-mode asynchronous controllers on FPGAs is presented. The approach is based on relative timing, which enables timing driven asynchronous design. On ASICs, relative timing based asynchronous designs have achieved notable benefits in terms of power, performance and area, when compared to their synchronous counterparts. This paper adopts the relative timing based design methodology on FPGAs and presents a methodology to extract and map timing constraints to guarantee correct operation. The method presented in this paper can be used to implement a wide variety of burst-mode controllers, across various FPGAs. This will form the foundation for seamless ASIC prototyping of asynchronous designs on FPGAs as well as implementation of low power asynchronous designs on FPGAs.**

I. Introduction

In the recent years, two factors have played a dominant role in the semiconductor industry. First being power which has replaced performance as the primary quality metric of most designs. This is driven by the increased use of mobile devices as well as process scaling, which allows chip implementations containing billions of transistors. The second factor is cost as a function of time to market and adaptability. The fast and inexpensive Field Programmable Gate Array (FPGA) design cycle results in the need to prototype ASIC designs on FPGA devices, as well as employ FPGAs in end products.

High transistor densities have made it harder to efficiently design synchronous clocking throughout the chip. This has become expensive in terms of power and design effort. Modular designs that allow for independent operation and clocking provide an easy solution to this problem. Asynchronous circuits inherently achieve this modularity in their designs, and have also shown power and performance benefits on ASICs [1], [2]. Some relative timing based asynchronous designs have shown 2.4×, 2.4× and 3.2× benefits in terms of area, energy and performance when compared to synchronous versions [3]. Relative timing uses explicitly defined circuit requirements to guarantee the conformance of a circuit to its specification [4].

FPGAs provide an economical way to prototype and develop digital circuits. Most commercial FPGAs are built and optimized for clock based designs. This leads to asynchronous designs on FPGAs being prone to hazards, primarily due to unpredictable routing delays and mapping processes [5], [6]. Burst-mode controllers are Mealy type finite state machines that control data flow and local clocking in asynchronous designs. Since the controllers use handshaking protocols between them, it is imperative that they operate without glitches. We present a formal methodology that uses a formally proven verification engine to extract timing constraints and maps the controllers appropriately, satisfying the constraints. The implemented timing constraints guarantee glitchess operation.

II. Background

Burst-mode finite state machines provide a suitable method to build asynchronous systems. Bundled data based asynchronous systems are partitioned into a control path and a data path. The data path comprises the logic and the registers or latches, similar to that in a synchronous system. The global synchronizing clock, however, is replaced by the control path. It is the responsibility of the control logic to maintain the timing and functional relationship between pipeline stages through handshake based local clocking. Numerous handshake protocols between modules are possible and various controllers exists that are capable of carrying out specified protocols. A detailed presentation of applying relative timing in bundled data based asynchronous systems can be found in [7].

FPGAs introduce unique challenges when compared to ASICs in implementing burst-mode controllers. Previous work shows that functional hazard-free boolean functions implemented in LUTs do not introduce additional logical hazards [8], [9]. However, this behavior as proposed is limited to single input change state machines operating in fundamental mode where the circuit, including state feedback, stabilizes before a new input arrives. Other work has shown that mapping controllers within a Configuration Logic Block (CLB) or a slice of an FPGA provides considerable advantage in being able to build a glitchless circuits [10]. The approaches in [11]–[13] also limit operation to the fundamental mode. Furthermore, the assumption of zero delay feedback wires in [13] is unrealistic on FPGAs.

Another way to implement extended burst-mode machines that are essential hazard-free is by reducing input concurrency and changing the functional behavior by introducing additional signals [6]. This implementation is based on a standard RS architecture using a hazard free logic minimization algorithm [14]. Using a RS latch based system may degrade performance on some FPGAs, since there are no physical RS latches on the chip. Rather RS latches are constructed using logic elements such as the LUTs and registers. Introduction of this additional logic may impact the efficiency of the controllers.

978-1-4799-6848-0/14/$31.00

Another approach is to build a standard set of hazard free Muller gates and create a library that is then given as an input to the FPGA synthesis tool [10]. A corroborating tool flow is required to use the standard set of gates. The gates in themselves are implemented within single CLBs to avoid logic hazards. However, the need to balance, sometimes even manually, isochronic forks that exist within circuits may make this process arduous.

The design flow shown in this paper incorporates burst-mode controllers operating in multiple-input switching mode. No additional change is made to the given controller logic/boolean functions to avoid hazards, rather timing constraints are used to make hazards unreachable. Also, the flow is optimized to achieve the highest possible performance metrics from the controllers.

III. Relative Timing

A hazard exists in a circuit if there exists a sequence of inputs or transitions that cause a glitch or an unwanted signal transition. Hazards can be fatal in the sequential control logic of an asynchronous system since the glitch can be interpreted as a valid signal transition and can put the system in a wrong state. [15]

Relative timing (RT) provides a methodology to model and verify circuits and protocols with timing constraints [4]. RT uses path based timing constraints to build asynchronous circuits. Yang developed the Automatic Relative Timing Identifier based on Signal Traces (ARTIST) tool, that uses a unique algorithm to determine the RT constraints based on signal traces generated from a formal verification engine that supports relative timing constraints [17]. RT constraints, simply put, specify specific event ordering. When these orderings are enforced, the circuit will behave correctly and hazards will not occur. An instance of an RT constraint is shown below.

$$pod \ \mapsto \ poc_0 + m \ \prec poc_1; \tag{1}$$

Equation 1 specifies that the maximum delay between pod (point-of-divergence) and the point-of-convergence poc_0 is less than the minimum delay between pod and poc_1. The constraint orders events, causing poc_0 to always precede poc_1. A margin of m is incorporated for added robustness.

RT constraints in asynchronous designs are generated to ensure glitchless operation by making hazards in a design unreachable. As part of this work the relative timing based methodology has been integrated into the synthesis and generation of burst-mode controllers on FPGAs. The first step, discussed in Sec. IV, was to characterize the FPGA being used with the aim of understanding FPGA routing, mapping and wire delays. A methodology is developed in Sec. V using the characterization data that can be adapted across all FPGA devices. In Sec. VI, the implementation of a burst-mode controller using this methodology is shown. The results and implementation in this paper are based on the Xilinx Spartan 6 XC6LX16-CS324 FPGA (Spartan6) on the Digilent NEXYS 3 board. All FPGA architectural comments in the paper are based on Xilinx FPGAs.

IV. FPGA Characterization

While FPGA architectural information provided by FPGA manufactures is sufficient for synchronous design, asynchronous designs in most cases require additional information. In an RT based methodology wire and logic delays must be known in order to ensure correct ordering of signal events. Another key aspect not present in clocked design is to know the capabilities and delays of feed-back loops in the FPGA. Feedback signals are used in asynchronous finite state machines to implement state holding logic. Thus to support RT design, FPGAs need to be characterized with regard to the following:

1) Routing delays.
2) Routing capabilities, especially for feedbacks.
3) Logic delays.

In the application presented here, the above is limited to intra-CLB routing delays and not inter-CLB. The Xilinx XST synthesis tool provides access to Xilinx primitives, which allow us to imply specific resources present on the FPGA. Using these primitives, and advance placement constraints, it is possible to target and map functions to specific LUTs and CLBs. Each CLB on the Spartan6 has two slices. Henceforth, the two slices will be referred to individually as either the "odd-slice" or the "even-slice". The odd and even nomenclature is based on the coordinates of the slice on the FPGA. Each slice in turn has four 6-input LUTs (ALUT, BLUT, CLUT and DLUT). The 6-input LUTs can be configured either to be used as a single 6-input LUT or two separate 5-input LUTs with independent functions and independent outputs but shared inputs. Each 6-input LUT has two outputs for this purpose. For example, the ALUT has two outputs, A and Amux, and has six inputs A1-A6. The Xilinx FPGA editor tool allows the observation of placed and routed designs and the extraction of most of the information needed to characterize the FPGA. For the purposes of this work, we have limited ourselves to the SLICEX type of slice on the Spartan6. Additional techniques to measure the delays of resources on an FPGA can be found in [18].

A. Routing Delays

Multiple combinations of feedback and routing were run within a CLB. The delays between the output of the LUTs (one of A, Amux, B, Bmux, C, Cmux, D, Dmux) to the input pins of the LUT (one of A1-A6, B1-B6, C1-C6, D1-D6) were noted for each run. The FPGA editor tool was used to extract delays. This process was automated using tcl scripts. Table I shows the routing delay values obtained. This particular result only shows the delays obtained from the intra-slice routing on the even slice of the CLB, i.e., all input and output pins belonged to the even slice. Data similar to Table I was also tabulated for the odd slice. Further tables with the routing delays between the two slices (inter-slice delays) were extracted. The results shown in Table I are indicative of results in the other data sets.

Table I does not indicate the number of times a certain routing delay was observed. Those results, in their entirety,

TABLE I
ROUTING DELAYS WITHIN CLB (PS)

PIN	A	Amux	B	Bmux	C	Cmux	D	Dmux
A1	650,657,663	556,563,569,571			532,539			
A2	701,708,826				731	725,732		
A3				381,387			350,357,363	
A4								
A5	895		196,203	425,432			321	227,233,235
A6								
B1			661,668,674	567,574,580			543,550	757,764
B2			953,960	977		871	953,960	
B3							1120	1062
B4	428,434	334,340	965		303,310	539		989,996
B5						469		1205
B6	143,150							
C1	538,545	752,759			656,663,669	562,569,575		
C2	875	824,830			800	899		
C3								
C4	955,1070		371,378	607			502	402,408,410
C5				455,462,468		834	431	
C6			274		799		143,150,156	
D1			548,555,556				666,673,674,679	572,579,580,585
D2			759	776,783,789			752,759,765,767	
D3	342					373,379		
D4						518		
D5	382	282,288,290			251,258	487		
D6								

would be too extensive to publish. However, Table II gives a snippet of the results. It shows the number of times the specific delay values between Amux to A1 in Table I were observed. This information will also play a key role later. From the tables we conclude the following:

1) It was observed from multiple attempts that delay values between two points tend to show low variance, although it cannot always be guaranteed.
2) The range of the routing delays is extremely large. In this case it ranges from 0.143ns to 1.203ns. This allows target a wide range of delays.

The delay values presented in the tables are those observed in one particular SLICEX CLB on the Spartan6. Delay value extraction was also done on similar CLBs across the chips, and the observed values were very rarely different, and variation was insignificant. Hence it is safe to use the extracted table data to describe the characteristics of all CLBs with SLICEX on the Spartan6.

TABLE II
EVEN SLICE ROUTING DELAY COUNTS FROM AMUX TO A1

Delay (ns)	Count
0.556	36
0.563	22
0.569	10
0.571	1

B. Routing Capabilities

Using a similar exploratory approach, we were able to observe certain behaviors regarding signal routing. Firstly, there are favorable paths for routing within the slice. For example, it can be seen in Table I that for a wire being driven by A, and being fed back into the A1-A6 inputs of the LUT, input pins A1, A2 and A5 were favored and never A3, A4 or A6. Similar observations can be made for the pins too. Note that even though the input set to a LUT can be explicitly defined, the tools can not be forced to connect a specific input signal to a specific input pin of the LUT. The tools in most cases reorder the inputs to a LUT while maintaining functional integrity.

Second, it was observed that the CLBs and slices have sufficient feedback routing capabilities to implement asynchronous burst-mode controllers. Knowing the limits on signal delays and feedback paths is fundamental to building any automated tool that maps to FPGA architectures.

C. LUT Logic Delays

The Xilinx AC and DC switching characteristics user guides provide the maximum propagation delays through the LUTs. These numbers are employed to calculate logic delays and will be combined with the characterized wire delays.

V. METHODOLOGY

Steps to the RT design methodology are presented here. The inputs include the synthesized sequential logic expressions for the burst-mode controller and the delay information extracted during the FPGA characterization phase. The boolean logic is synthesized using asynchronous synthesis tools such as MEAT [19], in order to minimize hazards. The over-arching intent is to use routing delay values obtained for the FPGA to enforce the required RT constraints in order to avoid hazards.

A. Step 1. Logic Clustering into LUTs

When controllers are implemented in ASICs, the circuits are based on standard library cells that are available to the synthesis tools. These libraries usually do not contain high-input gates, i.e., the number of inputs to each gate is usually limited to a small number. So to realize a six-input NAND

function, ASICs decompose the function into a set of smaller fan-in NAND and NOR gates. Such decompositions often create logic hazards due the race paths that exist in the intermediary signals. Therefore, it is advantageous to implement the full boolean function in a single gate as these race paths are eliminated. Similarly on an FPGA, such technology mapping hazards are avoided since most boolean functions can be implemented in a single six-input LUT.

To avoid the introduction of hazards, boolean functions for the controllers are clustered into LUT boundaries. The boolean logic is first cycle cut by replacing feedback paths with an output-input combination. A graph based partitioning algorithm is used to combine the logic into k-feasible logic blocks, where k is the maximum number of inputs to a LUT.

A situation may arise where a boolean logic function has more inputs than that permitted by a single LUT. In such a case boolean logic decomposition must be performed. Algorithms which create a hazard free decomposition can be employed [20].

B. Step 2. LUT Packing

The previous step produces a set of potential LUT boundaries for the implementation. The next step is to extend the clustering to pack multiple boolean logic functions within a single LUT where possible. A 6-input LUT can implement two independent 5-input functions with shared inputs. This input sharing can be employed to reduce the hazard properties of a design since both logic functions will observe the shared inputs simultaneously, removing a possible race path. When hazards are possible, RT constraints are required to ensure the hazards are not reachable. Thus, logic packing will produce a smaller design and may also produce a design with fewer timing constraints.

C. Step 3. Extracting RT Constraints

Once the possible implementations and combinations of the LUTs are known, a preliminary circuit structure is formed. Before proceeding to the next stage, it is important to check if the chosen combinations can be realized by the Xilinx synthesis and placement tool. This is performed by specifying the design in terms of the Xilinx primitives such as the "LUT6", "LUT5", "LUT4" etc. Additionally, placement constraints are added to tie function blocks into the locations that provide advantageous delay properties.

Once it is determined that the implementation candidates are feasible, a delay-insensitive (DI) model of the circuit is produced. DI design employs unbounded delays for both gates and wires, and thus is the timing model that will produce the most circuit hazards. All wire forks must be modeled, since a signal going to two different LUTs may have vastly different delays. One exception is made to the DI model of the implementation. If two independent boolean functions are packed into a single LUT, the shared inputs to these functions use a speed-independent model where the inputs are observed to change simultaneously by both functions. This can remove hazards in a design by creating "isochronic forks", or signal fanout that is observed as simultaneous events. Using ARTIST [17], the relative timing constraints for the circuit are now obtained by verifying the circuit implementation against it behavioral specification. Thus a set of timing constraints are generated that have been formally proven correct.

D. Step 4. Solving Constraints and Placement

The relative timing constraints generated above provide a set of inequalities that must hold on the design. These inequalities are path based constraints, where the maximum delay from point a to point b must be less than the minimum delay from point a to point c. The task in this step is to create a physical design that meets all of the timing constraint paths in the system. Creating a design that is correct and has good power and performance numbers is challenging. Timing constraints may be double sided, with some logic and signal paths having both maximum and minimum timing constraints. This is exacerbated in an FPGA because LUT delays are fixed, the available set of routing paths are very limited, paths through the routing network exhibit a very large range of variation, and it generally is impossible to guarantee that a specific signal path through the routing network is employed.

The inequalities are initially mapped to a circuit realization that employ the LUT delays and allow for the range of delays possible through the routing network. The maximal delay path, from point a to point b, is the most critical because additional delay can always be added to the minimum delay path from a to c – at least for paths without double sided constraints. Thus these constraints are given priority by minimizing the maximal delay paths.

Using Table I as a reference, the design is placed into LUTs that create signal path delays that adhere to the RT inequalities. Since none of the routing delays can be guaranteed between different implementations, it is important to query the FPGA editor to extract the actual routing delay and make sure that all inequalities are satisfied.

The first iteration tries to solve the inequalities exclusively using the predefined packed LUTs and the available routing resources. If such a solution cannot be found, delay is strategically added to a to c paths by passing them through additional LUTs which serve as logic buffers. The tool can be instructed to not optimize the inverter away, by invoking the "KEEP" constraint.

This step has not yet been fully automated. Future work will automate this step using the information in tables I and II, and the RT equations. A mixed integer linear programming model seems promising. The cost function is to give the best cycle time for the controllers. Such a solution should work well for burst-mode controllers which typically map to a very small number of LUTs, and in most case can implemented within a single CLB. We continue to use the Xilinx Tools for this step, and simply look to alter the design by adding specific placement constraints.

E. Step 5. Verification

The controller design implemented in the previous steps will have numerous instantiations all across the FPGA. The delays

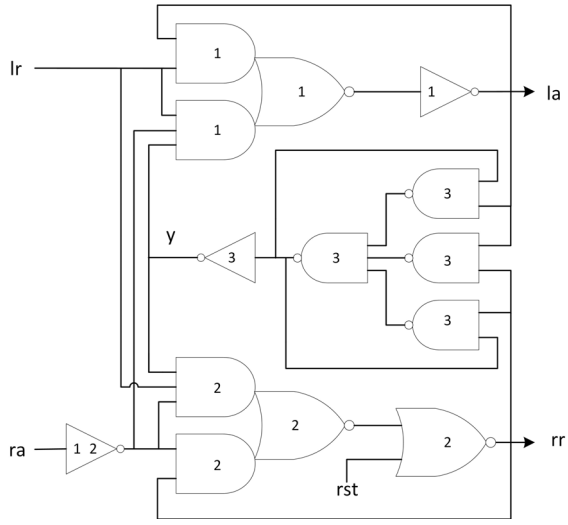

Fig. 1. Burst-Mode Controller with LUT Boundaries

LEFT = lr.c1.la.c2.lr.la.LEFT
RIGHT = c1.rr.c2.ra.rr.ra.RIGHT
SPEC = (LEFT | RIGHT) $\setminus\{c1, c2\}$

Fig. 2. CCS Specification of the Burst-Mode Controller

with each instantiation would have to be back annotated and must be verified to ensure that the RT constraints are all met. A second order goal is to find a solution that uses a minimal number of placement constraints to the Xilinx tools. This should simplify burst-mode controller placement and increase the likelihood the timing constraints are met in all of the instantiations.

VI. Implementation of Burst-Mode Controller

Fig. 1 shows the ASIC logic for a burst-mode controller. The behavioral CCS specification of the controller is given in Fig. 2. Implementation Step 1 defines LUT boundaries for the circuit. Graph based partitioning was used to cluster the logic gates into individual LUTs. Each logic element in Fig. 1 has been labeled with a number that indicates the LUT they were absorbed into. Note that the the ra inverter is labeled both *1* and *2*. This is because the inverter logic is absorbed into both LUT1 and LUT2, since both use ra as an input. In implementation step 2, it was determined that LUT1 and LUT3 could be implemented as a 4-input LUT and 3-input LUT with two shared inputs into a single LUT on the Spartan6.

Fig. 3 shows the reduced circuit after the logic into combined into LUTs. The original logic has reduced multiple gates on an ASIC into single LUTs. The figure also shows the combining of LUT1 and LUT3 in a single LUT with independent outputs, and shared inputs. LUT2 and LUT3 could not be combined as they have six independent inputs. Similarly LUT1 and LUT3 could not be combined due to high number of inputs.

A delay insensitive model is next defined. In Fig. 3, every fork has been labeled independently to indicate its unique delay. For example, even though input ra drives both LUT1 and LUT2, we represent them as two separate labels ra0 and ra1 since they may have different delays. However, since LUT1 and LUT3 are being packed together, the input la is now shared. Hence $la0 = la1$, and this fork can be modeled as isochronic. The RT constraints are then extracted using ARTIST for the model shown in Fig. 3. A total of 18 RT constraints were extracted. A subset of the constraints is shown below:

$$lr\uparrow \;\mapsto\; rr\uparrow + m \;\prec y0\uparrow; \tag{2}$$

$$lr\uparrow \;\mapsto\; la\uparrow + m \;\prec y0\uparrow; \tag{3}$$

The relative timing constraint in Eqn. 2 is ordering certain events following a transition on the input lr. In essence it states that after a transition on lr, output rr must occur before y0. The RT constraints are converted to inequalities by tracing the path from the point-of-divergence to the two points-of-convergence. Delay values d1, d2, d3 are assigned to LUT1, LUT2 and LUT3 respectively. The net names have been used to indicate delays through the wires. The two RT constraints shown have been converted to an inequality listing path based delays as follows:

$$lr1 + d2 + rr + m < lr1 + d2 + rr1 + d3 + y0 \tag{4}$$

$$lr0 + d1 + la + m < lr0 + d1 + la0 + d3 + y0 \tag{5}$$

An appropriate margin m is added to the equations to account for change in delays caused by temperature, voltage or process variations. At the end of step 6 a suitable implementation for the circuit was found. LUT1, LUT2 and LUT3 were tied to A, Bmux and Amux respectively. LUT1 and LUT3 were implemented in the even slice of the CLB and LUT2 in the odd slice. Using this configuration all the RT constraints were verified to be satisfied with sufficient margin. There was no need to add any additional delay elements. The controller was then shifted to different CLBs within the chip, while maintaining the same LUT packing, and all the constraints were still met.

TABLE III
Cycle Time of Implemented Controller

Number of Cycles (10^6)	Run Time (s)	Cycle Time (ns)	Frequency (MHz)
1037.9	3.3	2.91	342.5
2277.5	6.62	2.90	344.0

VII. Results

A five stage pipeline of the controller was built and the operation was timed. This was done by having a timed synchronous counter and second counter that was incremented by every positive lr transition of one stage of the pipeline. The results obtained are shown in Table III. The cycle time of a five stage asynchronous pipeline frequency is observed to average around 343MHz. Signals were probed using an oscilloscope and no glitches were observed.

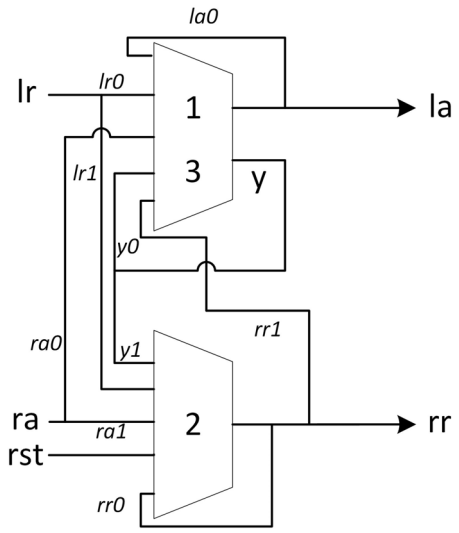

Fig. 3. Burst-Mode Controller with Labeled Delays

VIII. Conclusion

A methodology for implementing burst-mode controllers on FPGAs is presented. Timed asynchronous controllers are optimized for area and performance and are proven correct and hazard free based on relative timing constraints. We have tried to use the structure of the FPGA and its inherent routing to our advantage, and create a controller with minimal placement constraints. To support this approach, logic and wire delays inside an FPGA slice are characterized. Collaboration with FPGA vendors would allow simplification of the FPGA characterization phase. With robust min-max values we can have smaller delay margins. The boolean functions implementing the gates are mapped onto LUTs inside a slice and packed together where possible. The packed implementation is then verified using delay insensitive semantics to create the set of timing inequalities that must hold for the circuit to operate correctly and without hazards. The RT constraints are extracted using a proven verification engine. The approach was validated by placing the mapped controllers in various locations on the FPGA. A pipeline of controllers was implemented and measured for performance.

This methodology scales well to include system level architectures. However, larger circuits that span multiple CLBs will require characterization of the global routing delays of the FPGA.

IX. Acknowledgments

This material is based upon the work supported by the National Science Foundation under Grant Number 0810408.

References

[1] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 10, pp. 1904–1921, 2006.

[2] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev, "RAPPID: An Asynchronous Instruction Length Decoder," in *Advanced Research in Asynchronous Circuits and Systems, 1999. Proceedings., Fifth International Symposium on*, 1999, pp. 60–70.

[3] W. Lee, V. S. Vij, A. R. Thatcher, and K. S. Stevens, "Design of Low Energy, High Performance Synchronous and Asynchronous 64-point FFT," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 242–247.

[4] K. Stevens, R. Ginosar, and S. Rotem, "Relative timing [asynchronous design]," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 1, pp. 129–140, 2003.

[5] K. Maheswaran and J. Lipsher, "A Cell Set for Self-Timed Design Using Xilinx XC4000 Series FPGAs," Tech. Rep., 1994.

[6] D. Oliveira, S. Sato, O. Saotome, and R. de Carvalho, "Hazard-Free Implementation of the Extended Burst-Mode Asynchronous Controllers in Look-Up Table based FPGA," in *Programmable Logic, 2008 4th Southern Conference on*, 2008, pp. 143–148.

[7] K. Stevens, Y. Xu, and V. Vij, "Characterization of Asynchronous Templates for Integration into Clocked CAD Flows," in *Asynchronous Circuits and Systems, 2009. ASYNC '09. 15th IEEE Symposium on*, May 2009, pp. 151–161.

[8] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for Implementing Asynchronous Circuits," *Design Test of Computers, IEEE*, vol. 11, no. 3, pp. 60–, 1994.

[9] B. Lin and S. Devadas, "Synthesis of Hazard-Free Multilevel Logic Under Multiple-Input Changes from Binary Decision Diagrams," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 8, pp. 974–985, 1995.

[10] C. Pham-Quoc and A.-V. Dinh-Duc, "Hazard-free Muller Gates for Implementing Asynchronous Circuits on Xilinx FPGA," in *Electronic Design, Test and Application, 2010. DELTA '10. Fifth IEEE International Symposium on*, 2010, pp. 289–292.

[11] S. Unger, "A Row Assignment for Delay-Free Realizations of Flow Tables Without Essential Hazards," *Computers, IEEE Transactions on*, vol. C-17, no. 2, pp. 146–151, 1968.

[12] F.-C. Cheng and L. Plana, "Exact Essential-Hazard-Free State Minimization of Incompletely Specified Asynchronous Sequential Machines," Tech. Rep., 1994.

[13] D. Armstrong, A. D. Friedman, and P. Menon, "Realization of Asynchronous Sequential Circuits Without Inserted Delay Elements," *Computers, IEEE Transactions on*, vol. C-17, no. 2, pp. 129–134, 1968.

[14] H. Jacobson and C. Myers, "Efficient Algorithms for Exact Two-Level Hazard-Free Logic Minimization," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 11, pp. 1269–1283, 2002.

[15] C. Myers, *Asynchronous Circuit Design*, 2001.

[16] K. Stevens, R. Ginosar, and S. Rotem, "Relative timing," in *Advanced Research in Asynchronous Circuits and Systems, 1999. Proceedings., Fifth International Symposium on*, 1999, pp. 208–218.

[17] Y. Xu and K. Stevens, "Automatic Synthesis of Computation Interference Constraints for Relative Timing Verification," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, 2009, pp. 16–22.

[18] N. M. N. H. Benjamin Gojman, Sirisha Nalmela and A. DeHon, "GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays using Timing Extraction," in *21st ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '13)*, 2014.

[19] B. Coates, A. Davis, and K. S. Stevens, "Automatic Synthesis of Fast Compact Self-Timed Control Circuits," in *In 1993 IFIP Working Conference on Asynchronous Design Methodologies*, 1993.

[20] S. M. Burns, "General Conditions for the Decomposition of State Holding Elements," in *In International Symposium on Advanced Research in Asynchronous Circuits and Systems, Aizu*. Society Press, 1996, pp. 48–57.