Design of a Multi-Style and Multi-Frequency FPGA

Jotham Vaddaboina Manoranjan, Solomon Surya Tej Mano Sajjan, Vivek B. Gujari, Kenneth S. Stevens
Department of Electrical and Computer Engineering
University of Utah

***Abstract*—This paper presents an FPGA architecture capable of implementing relative timing based asynchronous designs. Modifications are made to a traditional synchronous FPGA architecture to make it asynchronous capable, while retaining its capability as a fully functional synchronous FPGA. Such a design permits multi-frequency implementations. A test FPGA fabric is developed and evaluated with the implementation of a MIPS processor. The asynchronous MIPS processor implemented on the designed FPGA provides a performance improvement of 1.7× while also providing a power improvement of 2.3× compared to the synchronous version of the MIPS implemented on a counterpart synchronous FPGA.**

I. Introduction

The current generation of FPGAs, built on deep sub-micron process technologies, are powerful devices with millions of programmable logic elements. This has enabled entire system-on-chip (SoC) designs to be implemented on FPGAs. However, it has become significantly harder to design efficient distribution of high speed clocks across these large chips. Maintaining synchronization between various SoC blocks operating at different clock domains can become power intensive. Asynchronous designs provide an alternative design style that can mitigate these issues significantly.

Asynchronous designs are shown to provide significant power and performance benefits on ASICs [1]. Relative timing (RT) based asynchronous designs have shown 3× benefit in energy, 1.5× in area and 1.2× in performance compared to synchronous counterparts [2]. RT uses path based timing constraints to guarantee functional correctness of a circuit.

The primary motivation for this paper is the desire to achieve the same level of power and performance benefit of asynchronous designs seen on ASICs when the designs are implemented on FPGAs. Commercial FPGAs are designed and optimized for clocked design, which poses significant hurdles in implementing asynchronous modules. For example, a significant problem in building asynchronous controllers with combinational feedback on FPGA fabrics is hazards [3].

In this paper we design an FPGA architecture that is capable of implementing RT based asynchronous designs. Certain additional circuits are included to make the architecture asynchronous compatible. The FPGA is designed on the 65 nm node and compared to a counterpart synchronous FPGA.

A unique feature of the proposed architecture is that this new FPGA retains complete support and functionality for traditional synchronous designs. The proposed architecture can be used independently for synchronous or asynchronous designs, and can also be used to combine the design styles. This merging of design styles can be employed to create hybrid solutions in numerous domains. For instance, synchronization issues that have become prevalent in high speed software defined radio could be resolved by utilizing asynchronous circuits to simplify global timing constraints while simultaneously using synchronous blocks for computation [4]. Other potential applications include low power domain crossing and communication backbones [5].

II. Background

Asynchronous FPGA architectures have been built for highly pipelined and high throughput applications, which prove valuable in applications such as cryptography [6]. One such asynchronous FPGA architecture was commercialized by Achronix Semiconductor Corp [7]. The architecture is based on micropipeline stages that enable high throughput applications. Novel configuration logic blocks that enable better pipelining and power reduction are implemented. The use of dual rail data encoding enables resistance to process variation, although this incurs an area penalty. Most of the silicon fabric was dedicated to routing resources. In this particular case 80-90% of the area was covered by configurable routing resources, and routing fabric consequently utilized a proportionally high power.

FPGA architectures capable of implementing various design styles have been explored [8]. A unique approach as part of the architecture is the inclusion of programmable delay elements (PDEs) for delay matching. The architecture targets reducing the tight connection between asynchronous design styles and design entry. However, the work does not elaborate on power/performance benefits.

Other approaches maintain logic element structures similar to those seen in synchronous FPGAs, while implementing asynchronous dual rail data routing structures and creating circuits to bridge between the two architectural styles [9]. The clock structure is replaced, which allows for power reduction. However, the use of dual rail protocols creates area overheads in the routing structure. As the design maintains synchronous logic blocks, conventional design tools can be used for logic clustering, while additional tools are built for the implementation of the asynchronous portions.

III. Approach

This paper provides a unique solution that addresses some of the drawbacks of the architectures discussed above. The following summarize the broad concepts of this approach:

- *Bundled data*: Numerous asynchronous data styles exist. As seen in Section II, a prevalent practice has been the use

978-1-5090-3561-8/16/$31.00

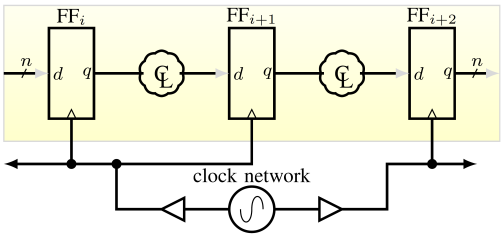


Fig. 1: Clocked design

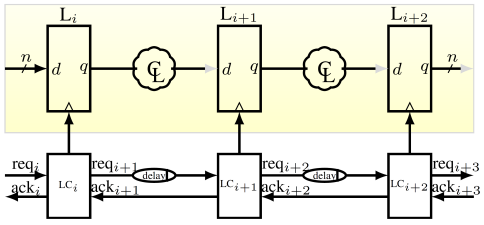


Fig. 2: Relative timed asynchronous bundled data design

of dual rail data protocols. Bundled data structures, on the other hand, avoid the area overhead of dual rail protocols as the data is relayed the same way as in synchronous designs. The FPGA design here incorporates the use of bundled data asynchronous designs.

- *Merging synchronous and asynchronous styles*: The logic and routing structure from synchronous FPGAs is largely retained in this new architecture due to the similarities in the data paths between bundled data asynchronous and synchronous designs. Only minor modifications and additions are made to synchronous FPGAs to make them asynchronous capable. Hence, designs that incorporate both synchronous and asynchronous designs can be efficiently implemented on this new FPGA.
- *Tools*: A significant barrier-to-entry for new digital design styles is the need to train engineers, and support them with the required tools. The current generation of FPGA tools are extremely well refined and an attempt to build these tools from scratch for asynchronous designs would require a large investment in time and money. Bundled data asynchronous designs require additional tool support because timing must be maintained between the handshaking protocols and data paths. This has been traditionally achieved in the ASIC realm through custom languages and tools for describing these timing relationship [10]. However, the bundled data asynchronous design style used in this work allows designers to retain the synchronous tool flow to build the asynchronous designs. On ASICs, asynchronous designs have been successfully implemented using commercial synchronous EDA tools [11]. Similarly, it is possible to retain synchronous designs tools for asynchronous designs on FPGAs.

IV. Relative Timing

Relative timing is a formal timing methodology that can apply to both synchronous and asynchronous designs. A comparison between synchronous and asynchronous architectures is provided in Fig. 1 and Fig. 2. Bundled data based

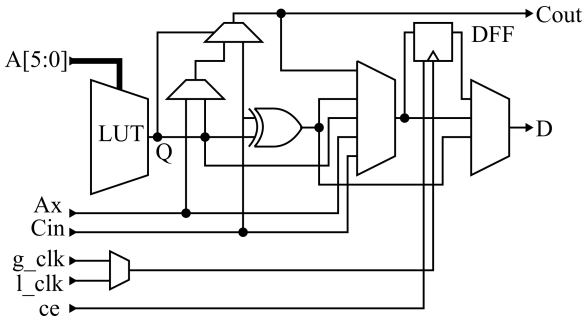


Fig. 3: Logic block design

asynchronous systems are partitioned into a control path and a data path. This can be seen in Fig. 2. The data path of the system consists of the logic and registers, similar to that in a synchronous pipeline (Fig. 1). The control path is primarily composed of modules called controllers. The controllers carry out a request-acknowledge (req-ack) handshaking between them and clock the registers. The control logic maintains the timing and functional relationship between pipeline stages.

RT provides a methodology to model and verify circuits and protocols with timing constraints [11]. The formal representation of an RT constraint is shown in Eqn. 1.

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

The pod and pocs are signal transitions or events in a circuit. Eqn. 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 m of separation is incorporated.

For the synchronous pipeline stages the timing relationship, in the presented formalism, is defined as follows:

$$FF_i/clk\uparrow_j \mapsto FF_{i+1}/d + m \prec FF_{i+1}/clk\uparrow_{j+1} \tag{2}$$

The above constraint sequences arrival of new data at the flip-flop (FF_{i+1}) corresponding to a pipeline stage before the clock edge (clk_{j+1}), following a clock edge (clk_j) at the flip-flop at the previous stage (FF_i). The constraint represents the timing relationship between pipeline stages and the global clock.

A similar constraint is constructed for the asynchronous pipeline:

$$req_i\uparrow \mapsto L_{i+1}/d + m \prec L_{i+1}/clk\uparrow. \tag{3}$$

Each $req_i\uparrow$ handshake on LC_i indicates new data presented to pin d of L_i. The delays in the circuit are sized as per the above RT constraint. Hence, after $req_i\uparrow$, the maximum delay to L_{i+1}/d must be smaller than the minimum delay to $L_{i+1}/clk\uparrow$. This would ensure that valid data is present when it is latched.

V. Baseline Synchronous Architecture

A. Logic Structures

The building blocks of FPGAs are look up tables (LUTs) that are programmable to create any n-input boolean function.

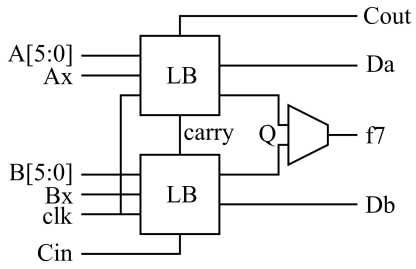


Fig. 4: Slice incorporating two logic blocks

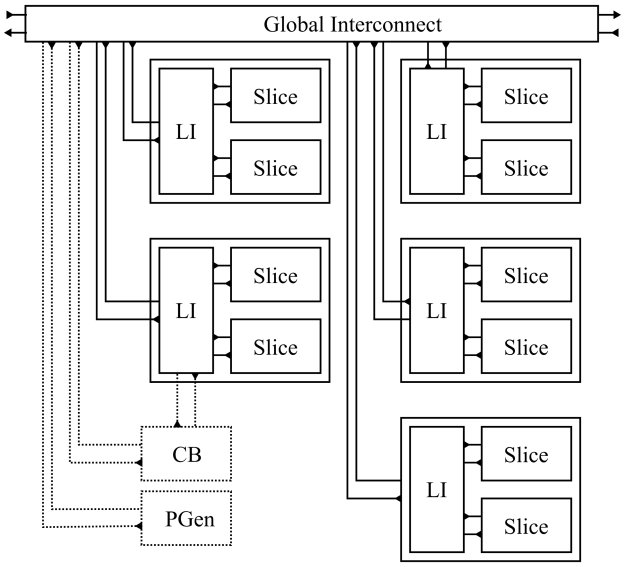


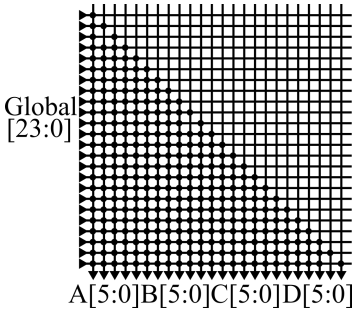
Fig. 5: Cluster of CLBs connected by a global interconnect: dotted lines show additions to synchronous baseline architecture

Fig. 3 shows the design of our logic block. It is a relatively standard FPGA logic block design incorporating a 6-input LUT, a carry chain, an XOR gate for addition functions, and a register. Two logic blocks are connected together to form a "slice". This is shown in Fig. 4. Within a slice, the output of the LUTs (Q in Fig. 3) pass through an additional multiplexer (MUX). This allows the two 6-input LUTs to be combined to form one 7-input LUT. The output of the 7-input LUT is $f7$ in Fig. 4.

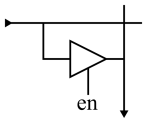
Two slices are then connected using a local interconnect. The local interconnect is essentially a switch matrix that routes signals locally. Two slices along with a local interconnect form a configuration logic block (CLB). The local interconnect from multiple CLBs also interface with a global interconnect. Fig. 5 shows this structure. It represents a "cluster" in the designed FPGA (in solid line). A cluster consists of five CLBs that are all connected to a single global interconnect.

B. Interconnect Structures

The interconnects are implemented using tri-state buffers. Configuration bits are used to connect an input to a certain output by enabling the required tri-state buffers. Fig. 6a shows a part of the local interconnect matrix. The 24 global inputs from the global interconnect are capable of being routed to



(a) Portion of local interconnect matrix



(b) Enable buffer connecting global inputs to a local input

Fig. 6: Design of interconnect

the inputs of the 4 LUTs within a CLB. This is done by using the tri-state buffer shown in Fig. 6b. Tri-sate buffers are implemented in each of the marked cross points in Fig. 6a. An input to a pin of a LUT is connected to outputs of multiple tri-state buffers, each of which allows the input to be driven by a different global signal. Only the lower left diagonal half of the matrix cross points are connected, reducing redundancy. The configuration bits ensure that only one of the tri-state buffers driving a wire is enabled. The authors are aware that the switch matrix being implemented is fairly rudimentary. However it suffices for the target MIPS application.

Asynchronous handshake signals are added to the routing matrix which are not in the baseline synchronous architecture. The data signals for both the clocked and the asynchronous FPGA are routed the same way. This is the advantage of utilizing relative timed bundled data asynchronous design; the data plane of asynchronous designs closely resemble the data path of synchronous designs.

C. Clock Distribution

A logic block is placed and routed first and then two copies are incorporated into a slice. Copies of the slice are incorporated to create a cluster. This creates uniform timing characteristics between various instances of the same block and slice. Copies of the CLB are then evenly distributed on the chip floor plan. Once placed and routed, a global clock tree is created and connected to the logic structure through the g_clk port (shown in Fig. 3). This creates a highly efficient low skew clock distribution network that spans the entire FPGA. This global clock tree does not flow through the interconnects. The slices can also be configured to use a clock that is sourced through the local interconnects (l_clk in Fig. 3).

VI. Multi-Style Multi-Frequency Additions

A. Handshake Controllers

Asynchronous controllers can be accommodated on an FPGA in two ways. The first involves building the handshake

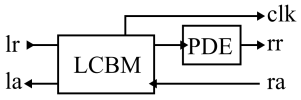


Fig. 7: Controller slice

controllers using the existing LUT structures on the FPGA. Unfortunately FPGAs have logic and routing structures that are not conducive to this approach owing to delay overheads and hazards [3]. Also, even though it is possible to implement these circuits on LUTs, the high activity factor of the controllers leads to very high power consumption.

The second method to accommodate the control path into the FPGA is to incorporate hard controllers into the FPGA. This is the method used in this paper. These are essentially control blocks implemented using traditional digital library gates. The controllers are connected to the routing matrix of the FPGA in a manner similar to how the inputs and outputs of the logic blocks are connected to the routing matrix. This enables the realization of a fast controllers with minimal delay and area overhead. Realizing the control logic with custom built hard coded circuits also leads to a much lesser power consumption when compared to implementing the same control circuitry on LUT based logic elements.

B. Protocol and Controller choice

Protocol and controller choice are interdependent issues. Various protocols exist that can be used to carry out handshaking between controllers. Controllers are then designed to implement these protocols. Controllers can interface with other controllers that work on compatible protocols. Two broad protocol categories are 4-phase and 2-phase protocols [12]. 4-phase protocols employ a four phase return to zero protocol. After each communication the handshake signals reset to zero, and data validity is expressed based on signal levels. We choose to utilize 2-phase protocols. These protocols employ signal transitions to validate data, and do not need to reset to zero. This reduces the number of signal transitions needed per data transfer, reducing the activity factor on the communication signals and in turn lowering power consumption.

As part of the FPGA design, we can choose more than one protocol, and implement more than one controller on the FPGA. This would give designers a choice. However, for the current paper, a C-element based two phase protocol is utilized. This was primarily chosen as it would be sufficient to implement the MIPS processor. A commercial implementation of this FPGA may require a wider selection of controllers.

C. Control Block

Since relative timing constraints order events on a circuit, the inclusion of PDEs can simplify the implementation of these constraints. Consider constraints similar to that in Eqn. 3; these constraints order the arrival of data at registers relative to the clock signal at the registers. Delays are usually employed in the handshake signals to ensure that the registers are not clocked/opened too early. This is done by matching the delay of the data path to the delay on the request signal. This is

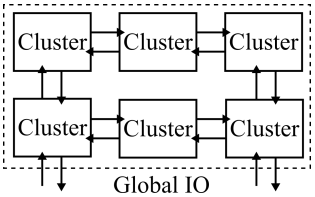


Fig. 8: FPGA test fabric

shown in Fig. 2. Delays are added on the req signals to match the combinational logic. This ensures that the relative timing constraint shown in Eqn. 3 is satisfied.

The PDEs are incorporated within a "controller block" that contains the controller and a PDE. This is shown in Fig. 7. The rr output of the controller goes through a PDE before it reaches the next controller. There are other easy methods of delaying signals on an FPGA. The routing structure can be used to create delay, and the LUTs can be programmed as delay buffers. In our approach, these methods are used to create coarse delays and the the PDEs are used to fine tune the delays. The design for this paper utilizes rudimentary MUX based PDEs with buffer chains to create delays. Custom PDEs can be used to reduce the area and power consumed [13].

D. Pulse Generator

This FPGA utilizes 2-phase protocols. This implies that at every transition on the request line, the data being propagated from one stage to another is valid. Hence data must be latched at these transitions. Therefore, the DFFs utilized in the logic blocks must register new data at both positive and negative edges. This is implemented here by incorporating a pulse generator that creates a pulse for every transition on its input.

E. Incorporating the Controller block and Pulse Generator

Fig. 5 shows the incorporation of the controller block and a pulse generator into the cluster (in dashed lines). Asynchronous designs use far more regular logic structures for data path logic than controllers. Hence the FPGA incorporates only sporadic instances of these controllers. A comparison with DSP slices is insightful. There are FPGAs with higher densities of DSP slices, and some with lower. Similarly, FPGAs can be built with varying number of specialized asynchronous blocks.

For the current design, each cluster is given only one controller block. The controller block is interfaced with not only the global interconnect, but also the local interconnect of a CLB. This is done to allow the IOs of the controller slice to enter and exit logic elements too, while also creating fast and efficient channels for controller to controller handshaking through the global interconnects. Similarly, one pulse generator per cluster is also interfaced to the global interconnect. Both the global and local interconnects are modified to incorporate routing capabilities for the additional signals.

TABLE I: Area (in μm^2) comparison between synchronous and multi-style FPGA

Design block	Synchronous FPGA	Designed FPGA	% penalty
Interconnect	79869.90	80508.85	0.8%
Logic elements	59019.84	59888.16	1.47%
Total	138889.74	140598.50	1.23%

F. Implementation

The FPGA is designed on the IBM 65 nm node using the Artisan standard cell library. Each of the building blocks, (Slices, local interconnect, global interconnect, controller slice and pulse generator) of the FPGA is independently designed, synthesized and placed and routed. The post layout blocks are then utilized to implement the FPGA. Placement constraints are used to create regular spacing between the blocks on the FPGA. Two versions of the FPGA are designed, one that is purely synchronous, and one that includes the asynchronous circuit additions, shown in dotted lines in Fig. 5. The global interconnects are also expanded to include additional routing structures for the asynchronous controller block. The configuration memory that stores the bitstream for the FPGA is emulated.

An area penalty is borne for the asynchronous version of the FPGA due to the fact that additional circuitry is added. The logic elements have a penalty due to the addition of controller blocks and pulse generators, whereas the interconnects have additional circuitry for routing the asynchronous communication signals. Table I shows the area penalty of the logic resources and the interconnects.

The FPGA design reported in this paper has six clusters connected as shown in Fig. 8. Furthermore, global IOs are also included into the design. This creates an effective fabric to test the benefits of the designed FPGA.

VII. Implementing a MIPS processor

An 8-bit MIPS processor based on the ISA provided in [14] is implemented to test the FPGA. The design utilizes a 5-stage structure. This MIPS architecture is designed to execute one instruction at a time. To accommodate this, a Johnson counter is used to create a clock-enable (ce) signal for each stage. Beginning from the first stage, the stages are sequentially enabled. This avoids any needless transitions that may incur additional power. The external RAM arrays that implement instruction memory and data memory, along with the internal MIPS register file, are simulated. The registers associated with each stage, the ALU and all other associated logic are implemented on the FPGA. The external memory registers are implemented with a delay of a single logic block. The Xilinx tool chain is used to synthesize the design, but the design is manually placed and routed onto the FPGA.

Two versions of the MIPS processor are built, one clocked and one asynchronous. The synchronous version utilizes the clock gating mechanism and a global clock. The asynchronous version replaces the global clock with handshake controllers and pulse generators. As asynchronous circuits only clock registers when data is valid, so there is no need for any clock gating. Hence the clock gating that is utilized in the synchronous MIPS is not used in the asynchronous design.

The minimum delays on the request line from one controller to another are designed to match the maximum delay on the corresponding data path in order to satisfy Eqn. 3. This is done by utilizing both the PDEs and free LUTs as delay elements.

VIII. Evaluating the FPGAs

A. Measured Metrics

For each of the following combinations of FPGA and design styles, power and performance was measured.

- *SonS* The synchronous version of the MIPS implemented on the synchronous FPGA.
- *SonM* The synchronous version of the MIPS implemented on this multi-style and multi-frequency FPGA.
- *AonM* The asynchronous version of the MIPS on this multi-style and multi-frequency FPGA.

The synchronous FPGA and the multi-style FPGA closely resemble each other. This allows us to compare the multi-style FPGA against a baseline synchronous FPGA. Comparison between SonS and AonM will gauging benefits gained by creating asynchronous designs, while comparison between SonS and SonM will identify penalties associated with the synchronous designs on the multi-style and multi-frequency FPGA.

B. Results and Analysis

The placed and routed designs on the two FPGAs were tested. Performance and power numbers are generated from vector based simulation using PrimeTime using post-layout back annotation from the sdf file for delay, and parasitic extracted values from a spef file. The MIPS is tested using a random instruction testbench. It is important to note that not only are the two FPGAs made to closely correspond to each other, the implemented synchronous and asynchronous MIPS utilize the same logic configuration, placement and interconnect routing in the data paths. This allows us to faithfully compare the FPGAs and designs without bias.

Table II details the results. The results of the synchronous MIPS implemented on the synchronous FPGA (SonS) is considered as the baseline for all comparisons. The power numbers are split into three parts: power consumed by the interconnect, the logic blocks, and the clock tree network. In the asynchronous MIPS implemented on the multi-style FPGA (AonM), the clock tree power is replaced by the power incurred by the control path of the asynchronous circuit. The following summarize key observations from the obtained results:

- The multi-style FPGA is capable of implementing synchronous designs with no speed penalty. This is due to no change in any of the logic structures. A minor power penalty of 2% is incurred due to the presence of additional circuit components and the need for additional routing capabilities.
- The asynchronous MIPS on the multi-style FPGA operates at much lower power. While the interconnect power

TABLE II: Power-performance for MIPS

Implementation style	Slice Utilization	Interconnect power (mW)	Clock/control path power (mW)	Logic block power (mW)	Total power (mW)	Instruction frequency (MHz)	Energy per instruction (pJ)	Energy benefit
SonS	90%	0.187	0.260	0.576	1.02	78.1	13.1	baseline
SonM	90%	0.191	0.269	0.576	1.04	78.1	13.1	0.98×
AonM	83%	0.214	0.005	0.231	0.45	133.3	3.38	3.88×

goes up due to additional signals flowing through it, this is more than compensated by the enormous saving in not using the global clock tree. The controllers and the pulse generator incur a fraction of the power of the global clock tree. Furthermore, the logic block power is also reduced. This is again primarily attributed to the perfect clock gating of the asynchronous design. In the synchronous design, the clock is being constantly propagated to the logic blocks, and each of the logic blocks contains circuitry for clock gating. This circuitry is constantly active, even though each stage operates only once every five cycles. The asynchronous design on the other hand clocks a logic block only when data is valid on its inputs.

- The logic block resources utilized by the asynchronous MIPS is lesser as the the Johnson counter based clock gating circuit is not used. This also contributes to the power reduction. The asynchronous design employs three logic blocks for delays.
- The asynchronous design on the designed FPGA is faster by 1.7×. The increase in speed is due to the fact that each pipeline stage can operate at its own optimal speed. There is no need to slow down a stage to match the critical path of another stage. Hence an instruction can flow through the entire system faster.

Overall, the asynchronous design on the designed multi-style and multi-frequency FPGA provides an energy per instruction benefit of 3.88× when compared to its fully synchronous counterpart. Furthermore, the new FPGA is shown to be capable of implementing synchronous designs with minimal penalty.

IX. Conclusion

The paper presents an FPGA architecture that uses a traditional synchronous FPGA as a starting point and makes alterations making the FPGA capable of implementing relative timing based asynchronous circuits. Hard-coded asynchronous controllers are implemented on the asynchronous fabric, in lieu of implementing LUT based asynchronous controllers. Pulse generators are added to the FPGA to utilize 2-phase asynchronous protocols.

Synchronous and multi-style versions of FPGAs are designed on the 65 nm node, and clocked and asynchronous 8-bit MIPS processors are implemented. The clocked design is placed and routed on the traditional FPGA, and the clocked and asynchronous version are implemented on the multi-style FPGA. The designs are compared for area, power, and performance. The asynchronous design on the multi-style FPGA provided a 3.8× reduction in energy per instruction and a 1.7× improvement in speed. The multi-style FPGA implements the synchronous design with a 2% energy per instruction penalty and no performance penalty.

The presented work makes a strong case for the benefits of asynchronous designs on programmable fabrics as well as multi-style and multi-frequency FPGA architectures. Relative timing asynchronous designs also provide significantly lower hurdles to adoption as traditional synchronous EDA tools can be used to implement asynchronous designs.

X. Acknowledgments

This material is based on work supported by the National Science Foundation under Grant Number 1111533.

References

[1] K. S. Stevens, S. Rotem, R. Ginosar, P. Beerel, C. J. Myers, K. Y. Yun, R. Kol, C. Dike, and M. Roncken, "An Asynchronous Instruction Length Decoder," *IEEE Journal of Solid State Circuits*, vol. 36, no. 2, pp. 217–228, Feb. 2001.

[2] 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 and Test in Europe (DATE)*. IEEE, Mar 2013, pp. 242–247.

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

[4] H. Duan, D. Huang, Y. Huang, Y. Zhou, and J. Shi, "A Time Synchronization Mechanism Based on Software Defined Radio of General-purpose Processor," in *Communications and Networking in China (CHINACOM), 2012 7th International ICST Conference on*, Aug 2012, pp. 772–777.

[5] V. Vij, R. Gudla, and K. Stevens, "Interfacing Synchronous and Asynchronous Domains for Open Core Protocol," in *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, 2014, pp. 282–287.

[6] K. Sun, L. Ping, J. Wang, Z. Liu, and X. Pan, "Design of a Reconfigurable Cryptographic Engine," in *Advances in Computer Systems Architecture*, ser. Lecture Notes in Computer Science, C. Jesshope and C. Egan, Eds. Springer Berlin Heidelberg, 2006, vol. 4186, pp. 452–458.

[7] J. Teifel and R. Manohar, "Highly Pipelined Asynchronous FPGAs," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '04, 2004, pp. 133–142.

[8] N. Huot, H. Dubreuil, L. Fesquet, and M. Renaudin, "FPGA Architecture for Multi-Style Asynchronous Logic." in *DATE*. IEEE Computer Society, 2005, pp. 32–33.

[9] D. Shang, F. Xia, and A. Yakovlev, "Asynchronous FPGA Architecture with Distributed Control," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, May 2010, pp. 1436–1439.

[10] H. Solutions, *TiDE Manual*, 2007.

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

[12] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design – A Systems Perspective*. Kluwer Academic Publishers, 2001.

[13] S. Kobenge and H. Yang, "A Power Efficient Digitally Programmable Delay Element for Low Power VLSI Applications," in *Quality Electronic Design, 2009. ASQED 2009. 1st Asia Symposium on*, July 2009, pp. 83–87.

[14] C. Ortega-Sanchez, "MiniMIPS: An 8-Bit MIPS in an FPGA for Educational Purposes," in *2011 International Conference on Reconfigurable Computing and FPGAs*, Nov 2011, pp. 152–157.