Reconfigurable Circuit for Implementation of Family of 4-phase Latch Protocols

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

Abstract—**This paper presents a novel reconfigurable circuit capable of implementing the entire family of 4-phase latch protocols. The architecture utilizes look-up-table based reconfigurable logic structures and fixed signal paths. The implemented circuit creates a fabric to realize a variety of high speed and low power controllers for asynchronous circuits on FPGAs. The circuit is implemented on the IBM Artisan 65nm node and its performance is compared with implementations on a Xilinx Virtex-5 chip that is manufactured on a similar node. A 4$\times$ improvement in speed and 3.3$\times$ improvement in energy per cycle was achieved.**

I. Introduction

FPGAs provide a versatile fabric for the implementation of digital designs. However, there are some unique challenges associated with FPGA built on deep sub-micron processes. It has become harder to design efficient clock distribution. This problem is compounded when designs require multiple clock domains. Asynchronous circuits provide an alternative design style that can mitigate these issues.

Asynchronous circuits often require special circuit components, such as asynchronous controllers for the implementation of local clocking structures. These controllers are usually Mealy state machines with combinational feedbacks for state holding. There are unique challenges with building asynchronous controller circuits on FPGAs. Unpredictable routing and signal delays make asynchronous controller implementation on synchronous FPGAs prone to hazards [1], [2]. Since controllers utilize handshaking protocols, it is necessary that hazards do not occur. Hazards lead to signal glitches. A glitch on a communication signal can cause the circuit to settle in an erroneous state.

Most methods implementing these circuits on synchronous FPGAs require individual attention to be paid to each controller [2], [3].Such approaches can be time consuming, given the unique timing characteristics of each FPGA and the numerous controllers possible. There is a plethora of handshaking protocols and controllers, that can be implemented based on the choice of the designer. The family of 4-phase latch protocols is a controller family that comprises 158 possible protocols [4]. When implemented on a regular FPGA using LUTs, each of these protocols have unique implementations and very specific timing constraints associated with them, in order to ensure that hazards are unreachable.

The presented work suggests an entirely difference approach compared to the traditional method of implementing controllers on generic FPGA logic structures. A unique programmable element capable of realizing a wide variety of asynchronous controllers is added to the FPGA fabric. The approach is akin to adding to an FPGA fabric DSP structures that are capable of multiplication and accumulation, even though these multipliers can be realized on the generic programmable structures. However, the advantage gained is faster and energy efficient operation.

This paper presents a novel reconfigurable circuit that is capable of implementing the entire family of 4-phase latch protocols. The circuit can be utilized to implement high speed and low power control circuitry for asynchronous designs.

II. Background and Motivation

A. Bundled Data Designs

This work targets bundled data based asynchronous designs, as it utilizes data paths similar to clocked designs. Fig. 1 shows a bundled data asynchronous design. The primary difference when compared with synchronous designs is that the global clock (that would be going to each of the pipeline registers), is replaced by the asynchronous controllers carrying out request-acknowledge (req-ack) handshaking and local clocking of pipeline stages. The data plane, shown in a shaded box, consisting of registers and logic, remains mostly unaltered.

B. Maximizing Flexibility and Performance

At an abstract level, there are two possible ways to realize the required asynchronous controllers on an FPGA; hard and soft controllers.

Hard controllers: The first would be to create hard controllers that implement a fixed protocol using standard cell library gates. The circuit can be plugged into the routing matrix during the design of the FPGA.

While these controllers would provide high performance metrics, there would be considerable limitations in flexibility. Each controller implementation would have a fixed protocol associated with it. Hence for each possible protocol that may be required by a design, a different controller would have to be be independently available on the FPGA. Within the family of 4-phase controllers there are 158 possible controllers. It would be uneconomical to implement all possible asynchronous controllers on an FPGA fabric, with each of them having multiple instances. This would lead to inefficient utilization of the silicon area.

Soft controllers The second method would be to implement soft controllers on the regular configurable logic structures of the FPGA. This would allow for greater flexibility in terms

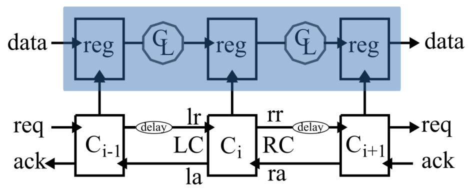

Fig. 1: Bundled data asynchronous 3-stage pipeline

of the protocols and variety of controllers. However, the use of generic resources such as look-up-tables (LUTs) and the routing interconnects lead to higher power consumption and lower speeds. Many of the controllers require 3-5 LUTs to be implemented, and have combinational feedbacks on performance critical paths. While greater flexibility is presented, FPGA routing structures slow the controllers down, and LUTs incur high power consumption during operation.

Present Approach: This approach creates a circuit that can be added to the FPGA fabric, which can provide protocol flexibility, and simultaneously, high performance and low power consumption. To achieve this, the circuit utilizes a best of both worlds approach. The circuit is built to eliminate routing overheads by using fixed signal paths for internal signals, while dividing the circuit into portions that can be implemented with standard cell logic, and portions that are implemented using LUTs. The LUTs create the needed flexibility, while the fixed logic and signal paths significantly reduce power and boost performance. The circuit is designed to implement all possible 4-phase latch protocols.

C. Incorporating the Custom Circuit into FPGAs

The ultimate goal is to create a reconfigurable circuit, that can be added to FPGAs to realize efficient controllers. In proposing an alteration to the FPGA fabric, the work delves into the area of Asynchronous FPGAs (A-FPGA). Prominent A-FPGA architectures have been based on the dual rail protocol as opposed to bundled data protocols [6], [7]. Dual rail protocols utilize two signals to encode a single bit, leading to heavy routing resource area and power overheads. Bundled data designs often eliminate these overheads.

The circuit built as part of this work, creates an A-FPGA that is capable of implementing bundled data based asynchronous designs as well as synchronous designs. This is possible due to the close correlation between synchronous designs and bundled data designs, as discussed in Sec. II-A.

No change to the synchronous FPGA logic and routing structures is required. The circuit can simply be plugged into the existing FPGA routing matrix. Instances of the circuit would be added to an FPGA similar to the way specialized circuits such as DSP slices are spread across the chip without any significant redesign of other portions.

III. 4-phase Latch Protocols

This section presents the 4-phase latch protocols. The circuit behavior is specified and the state space is described, bringing out the relevant portions as discussed in [4]. The circuit implements the family of 4-phase latch protocols.

A. The 4-phase Protocol

A controller has two channels, the left channel (LC) consisting of the input left-request (lr) and the output left-acknowledge (la) signals. Similarly, the right channel (RC) consists of the output right-request (rr) and an input right-acknowledge (ra) signals. In Fig. 1, for the controller C_i, a positive toggle on the lr signal, $lr\uparrow$, is associated with a new piece of data being presented at the input of the controller's register from the upstream pipeline. A $la\uparrow$ signal communicates to the upstream stage that the data has been latched. Similarly, a $rr\uparrow$ signals to the downstream pipeline that a new piece of data is available, and an $ra\uparrow$ from the downstream pipeline communicates that the data has been absorbed. The protocols being discussed are 4-phased, which implies that each channel resets its signals to 0 after each data transfer. The two channels, however, cannot operate independently. LC cannot accept any fresh data while the previous data has not been transmitted downstream and RC cannot send a $rr\uparrow$ until a fresh piece of data has been latched. This leads to the controllers state space diagram shown in Fig. 2 [4].

B. Concurrency Reduction and Protocols

Fig. 2 represents the most concurrent controller protocol. The state represented by (row,column) coordinates (1,4) is the initial state of the circuit. The (lr,la) and (rr,ra) labels on the axis give the signal levels for the states. The circuit moves to a different state when an input or an output toggle occurs. For instance, the circuit moves from state (1,4) to (1,5) when $lr\uparrow$ occurs.

Protocol alterations can now be made by removing states from the graph. These are essentially cuts in each row from the left side and/or the right side of the state graph. For instance the the cut $L2224$, removes the left most 2,2,2 and 4 states from rows 1,2,3 and 4 respectively. Since $L2224$ removes the state (2,5), the circuit is only permitted to move to state (1,6) from state (1,5). The cut $R0022$ removes the right most 2 states from rows 3 and 4. Rules governing this concurrency reduction by cutting away the state space is detailed in [4]. This reduction in concurrency leads to various protocols and classes of protocols including constant protocols, semi-regular protocols, regular protocols and stable protocols. There are 158 possible, non-deadlock protocols.

Implementing all these protocols on FPGAs can be tremendously time consuming given the timing and hazard characteristics of an FPGA. In highly concurrent protocols, the need to use combinational feedback for state holding adversely impacts performance. However, implementing various protocols is important as designers can choose one protocol over an other depending on the design and architectural requirements of the circuit being implemented.

IV. Circuit Design

A. Creating a Unique State Assignment

The state graph shown in Fig 2 shows the various possible states. There are a total of 32 states. Some of these states have

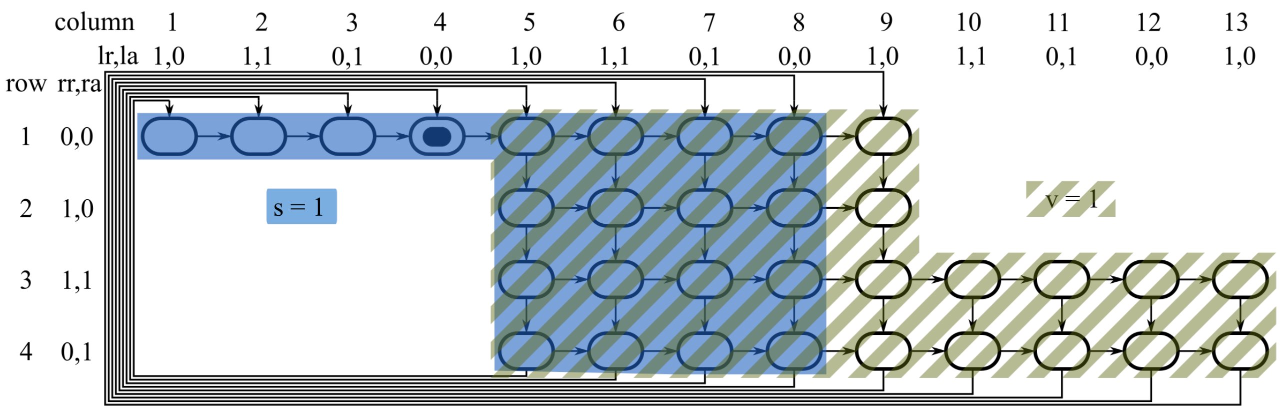

Fig. 2: State space for most concurrent 4-phase protocol

non-unique lr, la, rr, ra values. For instance, the states (1,4) and (1,8), have the same values for lr, la, rr, ra.

To resolve this, two additional signals v and s are added to create unique state assignments. Fig. 2 represents this by a color coded overlay on the states where the values of v and s are high. With the six circuit signals lr, la, rr, ra, s and v, all states can be differentiated.

These state signals are added with due diligence to avoid race conditions that can lead to hazards. For instance, if state signals are designed to be updated by both a possible input and output toggle, this could lead to a race condition between the two trying to update the same state signal. Hence in Fig. 2 state signals are only updated only during input transitions. Also, no transition is created that causes both state signals to be updated simultaneously, reducing the race conditions. Circuits with races are often resolved by adding timing constraints to internal circuit signals resolving these circuits. However, these timing constraints often impact circuit performance.

B. *Partitioning the Circuit*

The circuit is first partitioned into portions that need to be reconfigured and those that do not. Reconfigurability is primarily needed to implement the possible concurrency cuts as described in Sec. III-B. This is realized by utilizing two 6-input LUTs, one LUT for each of the outputs la and rr. Given that each LUT has all 6 signals as inputs (the IOs and state signals), they can easily be configured to create the state machine in Fig 2. Importantly any concurrency cut can be realized by reconfiguring the LUT and disabling required output transitions.

While the outputs are driven by LUTs that can be reconfigured, the operation (set and reset) of the two state signals v and s do not need to be reconfigured for any cut. Even if certain states are made unreachable by the LUT programing, the state signal can keep functioning as they would for a fully concurrent controller. Hence the circuitry for the state signals s and v are implemented using standard cell logic.

This approach is analogous to how some FPGAs use specialized carry chain circuitry with LUTs to realize mathematical functions such as addition. While the LUTs provide reconfigurability, the carry chain provides some measure of fixed functionality. The carry chain is thus implemented with standard cells, reducing power and improving performance.

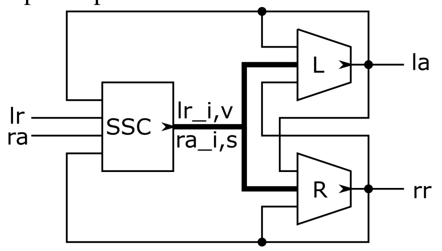

Fig. 3: Reconfigurable circuit to implement family of 4-phase latch protocols

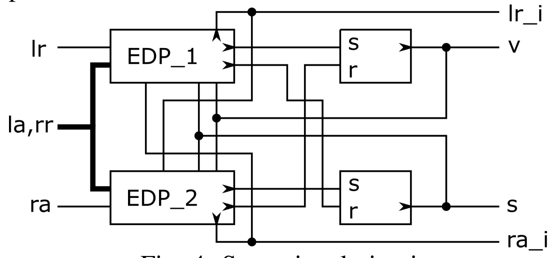

Fig. 4: State signal circuit

C. *Creating the Circuit*

Fig. 3 shows the circuit with two major components: the state signal circuitry (SSC) and the LUTs (L and R). The SSC is implemented with standard cell logic and functionally reads the input changes and updates the state signals.

The SSC is shown in more detail in Fig. 4. It consists of blocks named, edge detect/propagate (EDP) and two SR latches. The EDP blocks are used to gate inputs, allowing the states to be updated before the LUT sees input changes. Consider EDP_1 with the circuit is state (1,4). When an $lr\uparrow$ occurs, the circuit needs to move to state (1,5). This requires v to be set to 0 from 1. The EDP block recognizes this transition, and sends a set_v signal to the RS latch associated with v. Only when the signal v is set, does lr propagate to lr_i, which is the signal representing the input lr propagating to the LUTs. This is done on a circuit level by feeding back s and v to the EDP blocks. Similarly EDP_2 gates ra and propagates ra_i to the LUTs.

The outputs of the SSC lr_i, ra_i, s and v are all provided to the LUTs. Furthermore, as all the outputs are also needed for unique state assignment, the outputs of the LUTs are fed back to the SSC and the LUTs. The circuit is however not completely free of race conditions. When the circuit moves from state (1,4) to state (1,5), both lr and v toggle. A race condition is now created between the two signals. The EDP blocks resolve this by guaranteeing that the state signals are updated before an input is propagated to the LUTs, removing the race and possible hazard.

The LUTs are now programmed to realize the fully concurrent protocol, and can be simply reconfigured to realize concurrency cuts. The guaranteed signal arrival order between internal signals allow the LUT to be programmed appropriately without hazards.

D. Incorporating into an FPGA

Only the inputs and outputs of the circuit are now presented to the FPGA routing matrix. All other signals are made internal by fixing the routes. This eliminates routing overheads for internal wires that could slow down circuit performance.

By doing this, a draw back is that the LUTs cannot be used for any other operations when not in use as a controller. While not discussed in this work, an efficient local interconnect matrix can alternatively be implemented. The inputs of the LUTs can be MUXed to either connect to the local controller circuit structure or connect to the FPGA interconnect. This creates an optimized circuit for controller implementation, while also opening up the LUTs for global inputs.

V. Verification, Implementation and Results

A. Verification and Implementation

The circuit was designed on the IBM Artisan 65nm node. In addition to the signals shown in Fig. 3, a combinational $reset$ signal was added to the circuit to set the circuit to its initial state. A formal verification engine was utilized to verify the implemented circuit against the fully concurrent CCS (Calculus for Communicating Systems) behavioral specification of the circuit [5], [8]. The timing constraints associated with the propagation of inputs/outputs and the state signals were also provided to the verification engine.

B. Results

Since the circuit replaces the asynchronous controllers implemented on generic logic resources on FPGAs, it is benchmarked against these implementations. The circuit is compared to controllers implementations on the Xilinx Virtex-5 xc5Vlx20t-2ff323 chip. The Virtex-5 is also manufactured on a $65nm$ node. The voltage of the custom circuit was matched to that of the Virtex-5.

Cycle time and power were measured for the controllers. Using the two, the energy per cycle was also calculated. Simulation vector based metrics were generated. The controllers chosen represented the smallest and the most efficient for the Virtex-5 in terms of number of LUTs. The controllers were synthesized using Petrify [9] (a asynchronous synthesis tool) and the logic was used as the input for the Xilinx tool chain.

TABLE I: Controller implementation results: Virtex-5 (V-5) vs custom circuit (CC)

Controller name	Protocol category	V-5 power	V-5 cycle time	CC power	CC cycle time
L2002_R2264	Special O(8)	0.25mW	11.0ns	0.384mw	2.7ns
L2002_R2222	Stable O(16)	0.31mW	11.6ns	0.383mw	2.7ns
L3223_R2244	Regular O(16)	0.23mW	17.2ns	0.388mw	2.7ns

TABLE II: Energy per cycle: Virtex-5 vs custom circuit

Controller	Virtex-5	Custom circuit	Benefit
L2002_R2264	2.75pJ	1.036pJ	2.6×
L2002_R2222	3.60pJ	1.034pJ	3.4×
L3223_R2244	3.96pJ	1.047pJ	3.7×

Table I provides power and cycle time numbers. Only active power for both circuits were taken into account. Leakage power, IO power and other static power were neglected for both. An average speedup of over $4\times$ in terms of cycle time and $3.3\times$ benefit in terms of energy per cycle was achieved.

VI. Conclusion

The paper proposes a novel circuit for the implementation of the entire family of 4-phase controllers. The circuit utilized LUTs to create configurability, while utilizing fixed circuitry to optimize power. The LUTs are given all inputs needed to distinguish between states, allowing for the designer to individually remove states by altering the LUT configuration.

Instances of the circuit can be incorporated into a traditional FPGA circuit and plugged into the routing matrix to create the infrastructure needed to implement the control plane of asynchronous bundled data designs. The circuit was implemented on the IBM Artisan 65nm node and provided a $3.3\times$ benefit in terms of energy per cycle and a $4\times$ benefit in terms of cycle time, when compared to controller implementation on a Xilinx Virtex-5 FPGA chip which is manufactured on a similar node.

References

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

[2] 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.

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

[4] G. Birtwistle and K. S. Stevens, "The Family of 4-phase Latch Protocols," in *14th International Symposium on Asynchronous Circuits and Systems*. IEEE, April 2008, pp. 71–82.

[5] Y. Xu and K. S. Stevens, "Automatic Synthesis of Computation Interference Constraints for Relative Timing," in *26th International Conference on Computer Design*. IEEE, Oct. 2009, pp. 16–22.

[6] 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.

[7] 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.

[8] R. Milner, *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[9] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.