Clocked and Asynchronous FIFO Characterization and Comparison

HoSuk Han Kenneth S. Stevens
Electrical and Computer Engineering
University of Utah

***Abstract*—Heterogeneous blocks, IP reuse, network-on-chip interconnect, and multi-frequency design are becoming more prevalent in integrated circuit design. Communication amongst these blocks typically employs first-in-first-out (FIFO) buffering for flow control. This paper characterizes and evaluates several common designs in order to determine which structure is best for various specific applications. Two clocked and four clockless asynchronous FIFO designs are compared varying capacity, bit width, and structural configurations. The FIFO layouts are characterized in the IBM 65nm 10sf process for latency, throughput, area, and power. First order models are created to aid in CAD for FIFO synthesis, modeling, and optimization. Comparative results show that the asynchronous designs uniformly out perform the clocked designs in nearly every aspect.**

I. Introduction

FIFOs are an increasingly important component as design has become more modular. The choice of which structured memory to employ can have significant impact on the power, performance, and cost of a design. The choices are broad and range from clocked to asynchronous designs [1]–[6]. Some excellent work on the properties of FIFOs has been published [7], [8]. Yet a clear understanding of the comparative cost of different designs in terms of power, throughput, latency, and area – and of the key differences between specific structures – is not generally available. This paper reports on a study performed for a two fold purpose: to help designers choose the best FIFO for their target design, and to develop the foundation for an automatic CAD tool for selecting and synthesizing the best structure.

The most common clocked and asynchronous designs are compared across a broad range of design metrics. The designs are characterized for buffering capacity, energy per data word, leakage energy, width of the data path, forward and backward latency, throughput for a given occupancy, and area. Many of the asynchronous designs have various structural choices, and these structures are compared and optimized. First-order equations are that allow designs to be compared across arbitrary parameterized ranges are also developed.

This work evaluates the most common FIFO structures. Two clocked and four asynchronous FIFO classes are investigated. The clocked designs are assumed to operate entirely in a single clock domain. Synchronization costs are ignored in this analysis if the asynchronous design is placed in a clock domain. No status information beyond full at the write port and empty at the read port is assumed.

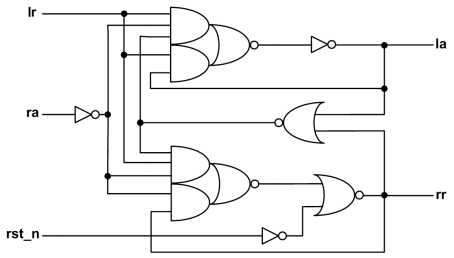

Fig. 1. Schematic of the linear pipeline controller

The end goal is to develop a tool to algorithmically evaluate the merits of various FIFO structures and generate a parameterized synthesis system that will create the most appropriate design. The choice can have significant impact. For instance, the best 8 word asynchronous FIFO expends half the energy with a forward latency that is almost a third that of the best clocked FIFO while achieving nearly identical maximum throughput. In general, the asynchronous designs are shown to significantly out perform clocked designs for nearly every metric and across nearly the full range of design parameters.

II. Design and Characterization

The results reported in this paper are derived from the layout of designs that have been automatically synthesized and characterized. The designs have been physically placed and routed in the IBM 10sf 65nm process technology using the Artisan RVt 12T library. Simulation results use full parasitic extraction from layout.

The implementations are designed to achieve the goal of making them as comparable as possible. This is accomplished by employing the same universal subcomponents to construct both the asynchronous and clocked FIFO designs. For instance, the same clocked one-hot shift register is used for address selection in the clocked head / tail pointer design as well as the asynchronous parallel and rectangular FIFOs. Likewise, the same pipeline controller, shown in Fig. 1, is used in all the asynchronous designs.

The design and characterization flow proceed as follows. First, a small set of shared circuit templates were designed. Each are implemented as a behavioral or structural Verilog module. The structural modules are mapped to the Artisan 65nm static library. There are 10 separate modules, three of which are asynchronous state machines, the rest consist of "clocked" components. The data registers are composed of

banks of flip-flops for the clocked head / tail pointer design, and data latches for all other designs.

The asynchronous modules consist of either pipelined stages which contain a latch bank to store data, or unpipelined stages that steer the control and data bits. The asynchronous unpipelined modules consist entirely of "clocked" elements, such as the one-hot shift register. The pipelined modules have an asynchronous state machine that controls the clock signal to the latch bank that stores the data, and performs flow control between buffer stages. The linear pipeline controller circuit of Fig. 1 is a typical example of an asynchronous finite state machine (AFSM) controller. The other asynchronous AFSM designs include the binary toggle and merge components that buffer and steer data between three modules.

The three asynchronous finite state machine controllers were synthesized or hand designed, and implemented as structural Verilog modules mapped to the static Artisan cell library. These circuits used a characterization flow that permits their use in clocked ASIC CAD flows [9]. This characterization includes formal verification to prove behavioral and timing correctness. Constraints are employed to ensure the structure of the cells are not modified by the CAD tools, but that the drive strengths can be correctly optimized for power and delay by using the `set_size_only` commands for these modules. Specific timing in the asynchronous modules not understood by the clocked CAD are defined with `set_max_delay`, `set_min_delay`, and `set_data_check` commands. These sdc commands are used by the logic synthesis and place and route tools. They are employed in a way that ensures that the asynchronous designs are power and timing optimized just as the clocked modules.

The "clocked" templates include a one-hot shift register for address generation, a counter, unpipelined n-way toggle and merge modules, and an elastic half buffer pipeline controller [10], [11]. Each are implemented as a behavioral or structural Verilog module depending on the design.

These components are assembled together to build the six FIFO classes reported in this paper. A custom parameterized script was written to synthesize the completed FIFO architectures as Verilog designs. The result of the script is a Verilog register transfer level (RTL) design. The RTL typically consists of both structural and behavioral modules.

The Verilog designs are synthesized using Design Compiler and then physically placed and routed using SoC Encounter. The design area is measured from SoC Encounter and then simulated for power and performance using Modelsim. Results from Modelsim import delays calculated from parasitic extraction from the physical layout in SoC Encounter using the standard delay format (sdf). One simulation in Modelsim is designed to measure the power. This simulation generates a value change dump (vcd) file that logs the switching activity on every node in the design. The vcd file is imported into SoC which generates the power results based on the parasitics and actual activity factors of the nodes from simulation.

The frequency of the clocked FIFOs is determined by simulating the design, increasing the clock frequency until the design fails, then backing off the timing by 15%. The frequency of the asynchronous FIFO designs are a result of the system logic delays and margins designed in the race paths.

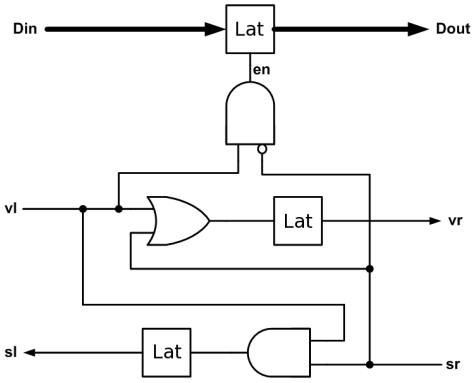

Fig. 2. Phased Elastic Half Buffer (pEHB) design.

Multiple simulations were performed on each design to evaluate latencies, worst case throughput, and energy under various activity factors. A simulation algorithm to ensure a fair energy comparison was developed by generating patterns that guaranteed identical activity factors on all transfers between storage elements. Data activity factors ranging from 6.25% to 50% on all data paths were simulated. The designs were also simulated across all valid occupancies to generate throughput versus occupancy graphs. Due to the volume of the tests, simulation and characterization scripts were developed to automate FIFO characterization.

Hundreds of different designs were synthesized and characterized. Each FIFO class, and at times many structural variants, were designed with capacities ranging from two through 50 words and data path widths ranging from from zero up to 64 bits. The scripts developed to generate the Verilog FIFO designs is therefore highly parameterized. The parameters include selections to identify (i) one of the six FIFO classes, (ii) the structural configuration specific to that class (e.g. different widths and depths described in Sec. IV), (iii) the capacity of the FIFO, and (iv) the width of the data path. Behavioral test interfaces are attached to the head and tail of the FIFOs to aid in the characterization.

The simulation results are evaluated and first order models developed for capacity, throughput, energy scaling, and performance values. These models can be implemented in a CAD tool to automatically select and synthesize the best FIFO for specific design parameters and quality metrics.

III. FIFO Classes

Two clocked structures are used: linear flow-through FIFO and head / tail pointer. The linear clocked FIFO is implemented as phased elastic half buffer cells (Fig. 2) that employ latches for data storage [10], [11]. This logic implements a latency insensitive protocol [12], [13]. Elastic half buffers must be connected where the control and data latches alternate between transparent high and low cells in adjacent stages.

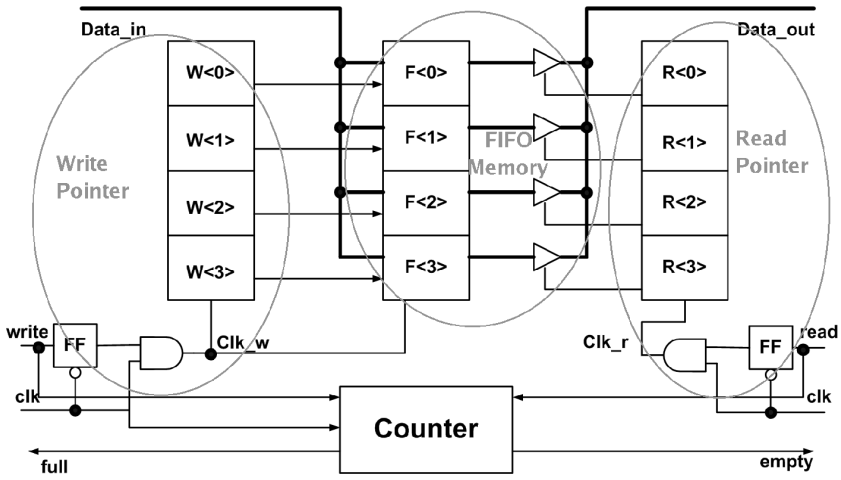

Fig. 3. Design of the Clocked Head/Tail Pointer FIFO. Read and write pointers are one-hot shift registers.

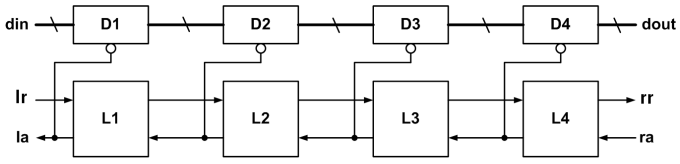

Fig. 4. Block diagram of a 4-deep Linear FIFO. L1–L4 are the controller of Fig. 1, D1–D4 are latch banks.

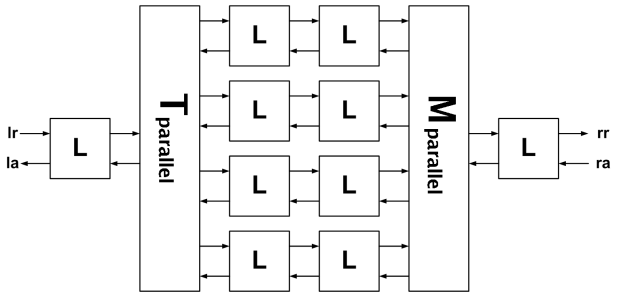

Fig. 5. Block diagram of the parallel FIFO. Each L blocks is the same as a pipelined controller / latch pair in Fig. 4.

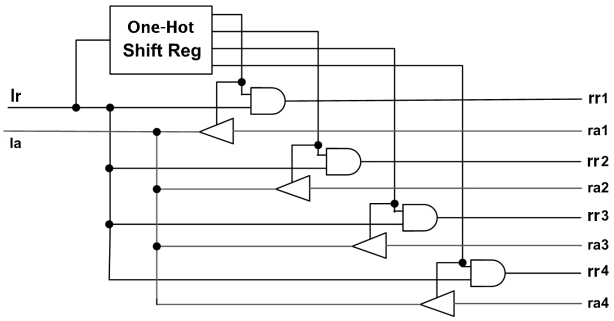

Fig. 6. Schematic of the unpipelined $T_{parallel}$ module

The implementation of the clocked pointer FIFO is shown in Fig. 3 [3]. The read and write pointers are implemented with a circular clocked one-hot shift register for address selection.

Four asynchronous structures are implemented: linear, parallel, binary tree, and square [1], [4], [8]. A linear structure with capacity of four is shown in Fig. 4. All asynchronous FIFOs and the clocked linear FIFO use latches for data storage. The structure of Fig. 4 is used in each of the asynchronous designs, where the combination of the linear controller and latch bank form a pipeline stage, just like a Di Li pair in Fig. 4. A "normally closed" latching protocol is used for energy efficiency where the latches are only briefly transparent to store the data.

A block diagram of the parallel FIFO is shown in Fig. 5. The T and M blocks are unpipelined modules that steer data to and from a set of parallel "legs". The capacity of the legs is configurable; each leg in Fig. 5 has a capacity of two. The parallel design is the same as a linear structure in the degenerate case with only a single leg. Likewise the configuration where each leg contains a single buffer is an asynchronous implementation of the clocked H/T pointer FIFO.

The design of the toggle template for the parallel FIFO is shown in Fig. 6. The one-hot shift register is the same design as the circular shift register used in the read and write pointers of the clocked head / tail pointer FIFO (Fig. 3. This template implements an unpipelined protocol that steers the handshake signals without storing the data. The data is "broadcast" to all downstream modules. This module reduces the frequency of the parallel FIFO to the first order by about a factor of two when placed between two linear controllers because it is not pipelined. However, it is highly modular and scales well. The parallel merge module has a design similar to the parallel toggle, but requires a mux to select and steer data.

The block diagram of the binary tree FIFO is shown in Fig. 7. In this design the data is steered through a pipelined logarithmic fanout tree to a number of parallel legs. The legs are then merged through a logarithmic binary tree to the output. The parallel legs can have a capacity $>= 0$. The binary toggle and merge modules are pipeline asynchronous templates with a protocol and implementation similar to the linear controller in Fig. 1.

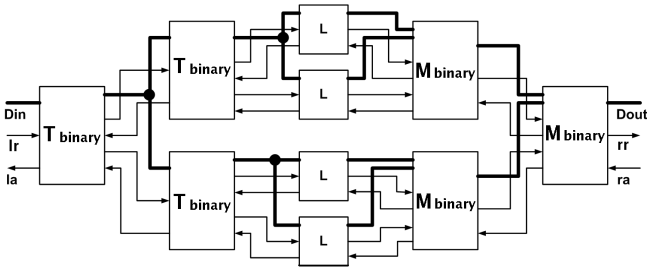

Fig. 7. Block diagram of Tree FIFO. The T and M blocks are pipelined.

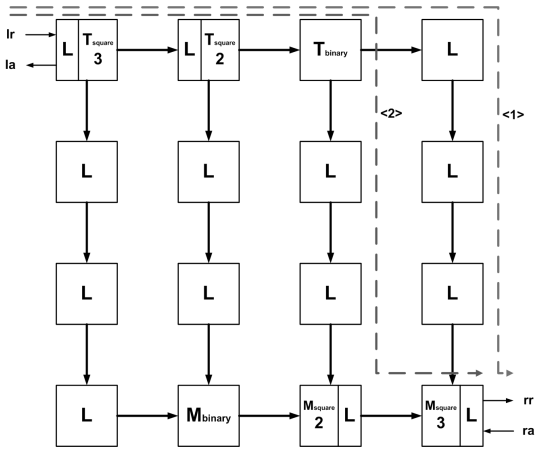

Fig. 8. Block diagram of the S-4-4 square FIFO. $\langle 1 \rangle$ shows the path of the first datum, $\langle 2 \rangle$ of the second.

The block diagram of a square FIFO is shown in Fig. 8. This design consists of a row of steering cells on the top and bottom, with parallel legs in between. Data flows across the top row to each leg in order, down, and then out at the bottom right. The path of the first two tokens is shown in the figure with arcs labeled $\langle 1 \rangle$ and $\langle 2 \rangle$. The controllers in the bottom row first steer the datum from the top to the output, then take one or more tokens from the left based on the location in the rectangle. The degenerate case of a single leg equals a linear structure. The square toggle and merge templates, like the parallel modules, are not pipelined. Fig. 9 shows a pipelined linear controller connected to a square toggle module. Like the unpipelined parallel design, these templates also reduce the frequency of the design approximately in half.

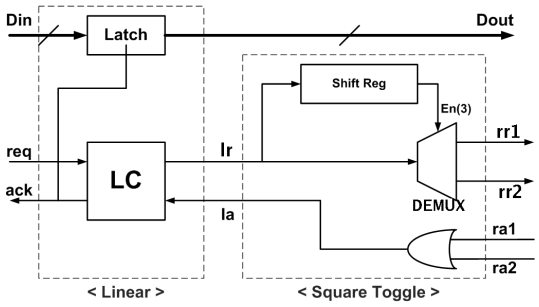

Fig. 9. Linear controller with a $T_{square}3$ template.

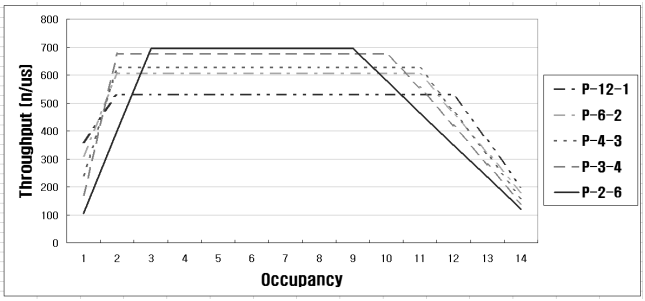

Fig. 10. Throughput vs. occupancy for five 14-deep parallel FIFO configurations. Labels are shown in inverse order of maximum throughput.

IV. Characterization

First order equations were developed to represent the structure, capacity, and forward latency for each design. The equations are based on the number of parallel legs N_l, the capacity of each leg C_l, the capacity of the FIFO C_f, the clock period P_c and the latency of the asynchronous linear controller L_c. The top two classes in the following table are clocked, the latter four are asynchronous.

Design	Structure	Capacity	Fwd Latency
Clk Linear	–	C_l	$(C_f/2)P_c$
Clk H/T	–	N_l	$3P_c$
A. Linear	–	C_l	$C_f L_c$
Parallel	P-N_l-C_l	$2 + N_l C_l$	$(C_l + 4)L_c$
Tree	T-N_l-C_l	$2(N_l - 1) + N_l C_l$	$(2\log_2 N_l + C_l)L_c$
Square	S-N_l-$(C_l + 2)$	$N_l(C_l + 2)$	$(2Nl + Cl)L_c$

The structures are specified by the class, number of parallel legs, and capacity of each leg. Thus the parallel FIFO of Fig. 5 is a P-4-2 FIFO and the tree design of Fig. 7 is a T-4-1 configuration.

A. Latency and Throughput

Forward latency is defined as the delay from placing a token into the head of an empty FIFO until it has been read from the tail. The backward latency is its dual: the delay from removing a token from the tail of a full FIFO until one can place a new token in the head.

Latency has a major effect on FIFO throughput under certain occupancy ranges. Throughput is limited by the latency in all designs when the occupancy is near empty or full [7]. The throughput of a FIFO is therefore dependent on its occupancy and cycle time.

Throughput vs. occupancy is measured keeping the number of data items in the FIFO constant at all times. When a datum is removed from the tail of the FIFO, a new datum is simultaneously added to the head of the FIFO. The FIFO is first initialized with a particular occupancy. When a FIFO is near empty, throughput is reduced due to the forward latency of tokens. More data could be added to the FIFO, but this must be delayed until data is valid at the output to maintain a constant occupancy. When the FIFO is nearly full, the dual applies. New data can not be removed from the FIFO until a bubble, or empty position, propagates backward to the input. Throughput reaches a condition where it is limited by the cycle

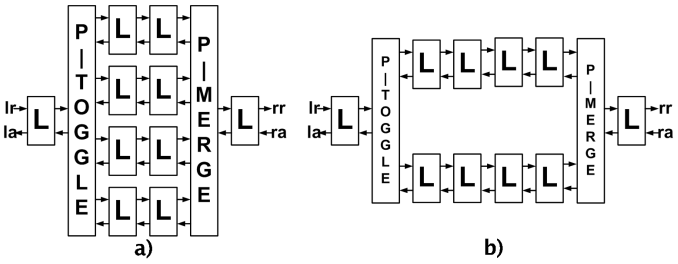

Fig. 11. A P-4-2 and P-2-4 parallel design w/ capacity of 10

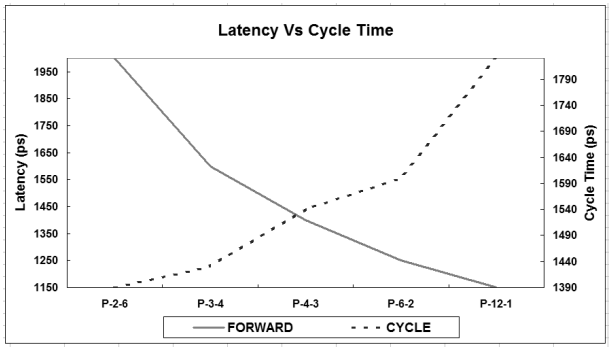

Fig. 12. Latency and cycle time of five 14-deep FIFOs

time of the design if forward and backward latencies are small. This results in the graphs in Fig. 10 where throughput triangles saturate due to the maximum cycle time of the design.

Designs with small forward and backward latencies saturate their throughput quickly based on the maximum frequency of the design. However, designs with large latencies restrict throughput across a large range of occupancies. The clocked linear FIFO has a very high latency. For this design the throughput is reduced due to latency in all cases except for FIFOs of even length that are exactly half full.

The forward and backward latencies for clocked designs are equivalent. However, The forward and backward latencies of asynchronous structures can have different values which are dependent on the handshake protocol. In this study we selected a protocol with a faster forward latency than backward latency. This results in different slopes as can be seen in Fig. 10. Since the forward latency is less than the backward latency for this design, the maximum throughput is reached when the FIFO contains two or three tokens. However, maximum throughput is not reached until up to 5 bubbles exist in the designs when the FIFO is nearly full.

$$\frac{L_f}{t_c} \;\leq\; O_t \;\leq\; C_f - \frac{L_b}{t_c} \qquad (1)$$

Equation 1 models the effect of latency on throughput. L_f and L_b are the forward and backward latencies, C_f is the total capacity, t_c is the cycle time, and O_t is the range of number of tokens across which the optimal throughput is reached. The smaller the forward and backward latency, the sooner the maximum throughput of the FIFO is reached.

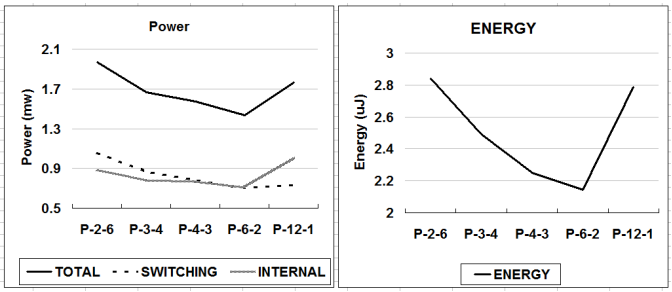

Fig. 13. Power and Energy of 14 deep Parallel FIFOs

B. Optimal Asynchronous Structures

All asynchronous FIFOs except the linear structure have multiple configurations that achieve similar capacity. Fig. 11 shows two configurations of a parallel structure each with a capacity of 10. The first part of our characterization determines the best configuration for the asynchronous designs. This is illustrated with a parallel FIFO having a capacity of 14 words. Five structures are synthesized and characterized: the P-2-6, P-3-4, P-4-3, P-6-2, and P-12-1. As the number of parallel legs increases, the cycle time increases but the forward latency decreases. This creates the tradeoff shown in Fig. 12. Fig. 13 shows the energy per token tradeoff for each of the configurations, showing P-6-2 as clearly the best. Throughput versus occupancy is shown in Fig. 10, displaying the tradeoff between maximum throughput and the range over which the maximum throughput is obtained. The graph for area is not shown, but it correlates rather well to the cycle time in Fig. 12. The area increases from 20,636 to 22,093 to 26,767 μm^2 for the P-4-3, P-6-2, and P-12-1 configurations. The parallel FIFOs with capacity of two in each leg achieves the best cycle time $\times$ area $\times$ energy $\times$ throughput versus occupancy. Hence these configurations are used in this paper.

The same tradeoffs occur in the tree structures with a similar optimum. The results can be explained as follows: (i) Latency, energy, and the breadth of the maximum throughput versus occupancy improve by reducing the number of linear stages a datum must pass through. (ii) The steering logic is significantly more expensive than the linear pipeline logic in terms of latency, energy, and area. (iii) The difference between a capacity of one and two tokens per leg doubles the cost of the steering logic, but only increases the number of tokens a datum flows through by a small percentage (depending on total FIFO capacity).

V. Results

Fig. 14 shows a comparison of the latencies of the six FIFO designs across capacities ranging from three to 50 words. One of the graphs highlights small capacity FIFOs. Forward latency is a measure of how quickly maximum throughput is reached, as well as the time to propagate a value through the FIFO. The asynchronous linear structure has the best forward latency for very small capacities of four or less. Between four and 16 the parallel and tree structures have the best latency.

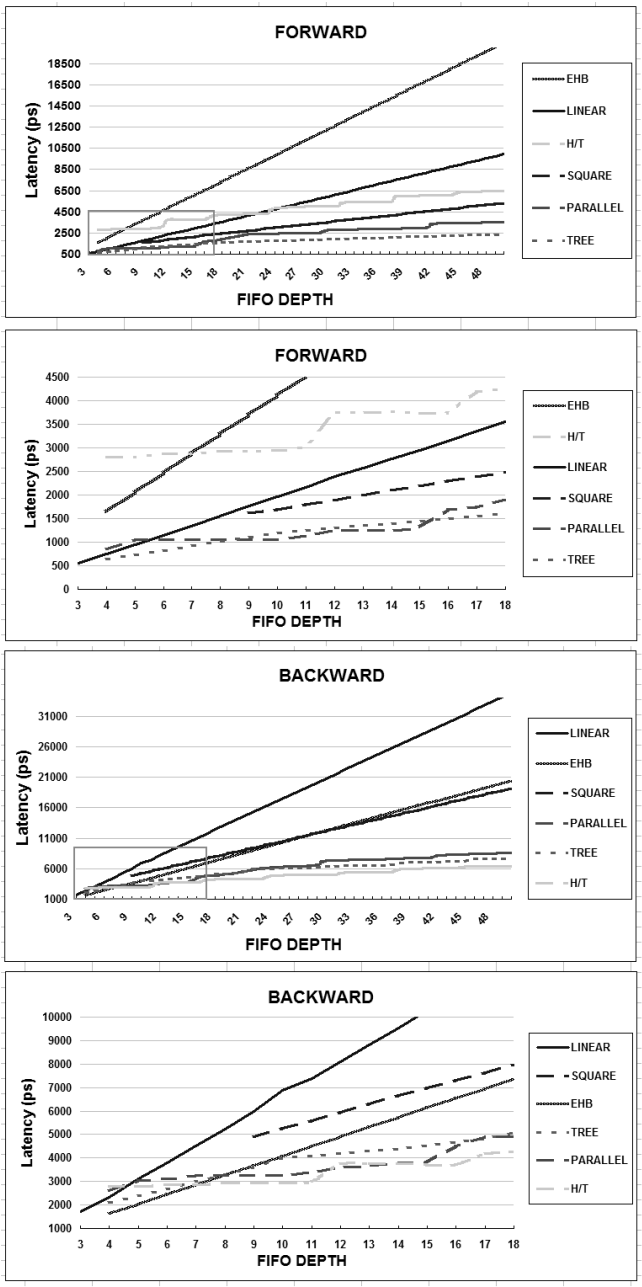

Fig. 14. Forward and Backward Latencies of 64-bit FIFOs. Labels in decreasing order at maximum depth.

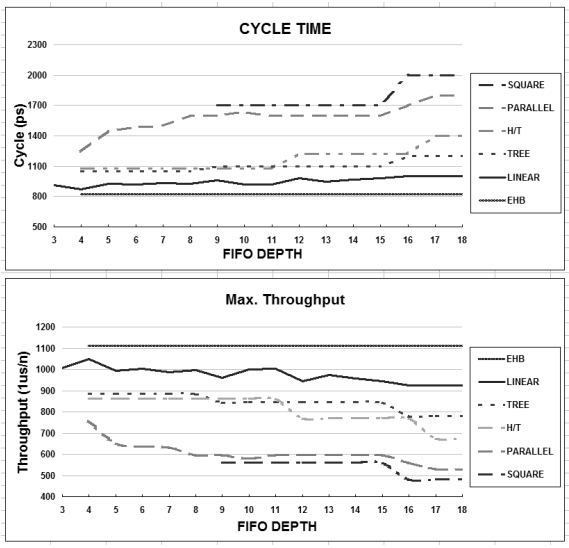

Fig. 15. Cycle time and maximum throughput

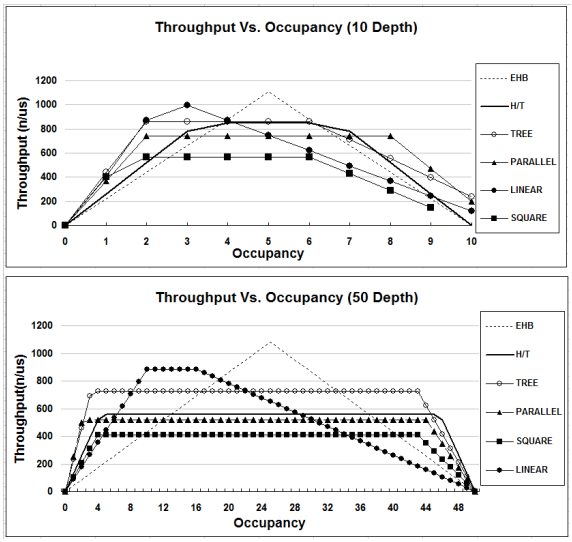

Fig. 16. Throughput versus occupancy comparison for FIFO capacities of 10 and 50 tokens

Beyond 16 elements the tree architecture is best. Backward latency produces a result similar to forward latency. However, backward latency in the asynchronous designs is degraded. Thus the elastic half buffer is best up to about seven data items, after which the head / tail pointer is the best. This implies that under a stalled condition, the clocked designs will recover to full throughput quicker than these asynchronous designs.

The cycle time and maximum throughput of the designs are compared across many capacities. Fig. 15 shows the results for small capacity designs. The design with the highest throughput is the clocked linear structure. However, this architecture has the largest latency by far of any design and only achieves maximum throughput under a single token occupancy value. The next highest throughput is the asynchronous linear design, but it suffers similar problems with the clocked elastic buffer design. For some designs, these could be the optimal choice – if throughput is the primary metric and the FIFOs could be maintained in the small optimal range. However, for FIFOs that require high throughput *and* low latency the asynchronous tree FIFO and the clocked head / tail pointer FIFO are the best up to a capacity of around 16, beyond which the tree FIFO has the highest throughput.

Fig. 16 compares the throughput versus occupancy for designs with a capacity of 10 and 50 words. The asynchronous tree and parallel designs reach maximum throughput sooner than all other designs with a broad maximum throughput range, and the tree reaches a significantly higher maximum.

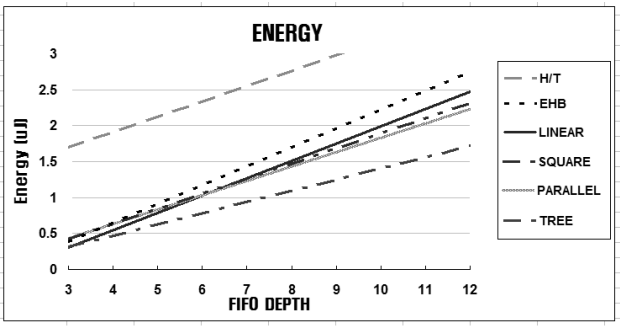

Fig. 17. Energy comparison of small capacity FIFOs

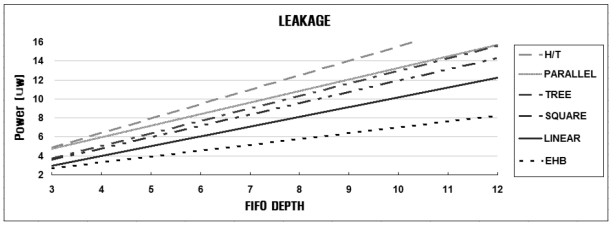

Fig. 18. Leakage power of small capacity FIFOs

For an application where dynamic buffering is required with good throughput and low latency from the empty state, the asynchronous tree design appears to be the best option.

Fig. 17 compares the average energy for a datum to pass through the various structures for small capacity designs. The asynchronous tree design is the most energy efficient. The clocked head / tail pointer expends substantially more energy than all other designs. This relationship and the relative slopes hold for the full design space investigated, including designs ranging from a 6.25% to 50% data activity factor, and for data widths ranging from zero to 64 bits.

Leakage power is compared in Fig. 18. These values also correspond well to the layout area of the designs. The clocked linear design gives lowest leakage, but this does not account for leakage in the clock distribution network and clock gating circuitry.

VI. Summary

This paper reports on several common FIFO structures that can be used for flow control. They are compared with maximum throughput, throughput versus occupancy, energy efficiency, area, leakage energy, and latencies. First order equations are derived to model the capacity, latency, and maximum throughput based on occupancy of the designs. Most asynchronous FIFO classes have multiple configurations that result in similar capacity but different power and performance values. The optimal configurations for a given capacity were determined. A huge design space is investigated through generating modular designs and synthesizing hundreds of instances of six FIFO classes. Results are for physical layout in a 65nm process with parasitic extraction, and include varying the capacity, data width, configurations, and data activity factors.

Clocked and asynchronous designs are compared and contrasted to determine the best structure for a specific need. The square FIFO, while academically interesting, is shown to be an impractical design as it is never the best choice for any parameter. In general an asynchronous FIFO is the best choice across nearly every capacity and for most metrics. Latency, energy, and throughput will usually be the primary factors used to select the best design. In such cases, one of three asynchronous FIFO structures are usually the best. The clocked linear FIFO does attain the highest throughput of any design. However, this performance is only reached for a single occupancy value.

This work facilitates generating CAD that will weigh the priorities, utilize the first order equations to select a structure, and synthesize the correct design for the application.

VII. Acknowledgments

This material is based upon work supported by Semiconductor Research Corporation task 1817.001 and the National Science Foundation under Grant No. 0702539. The full physical Artisan library was provided under a grant from Arm.

References

[1] E. Brunvand, "Low Latency Self-Timed Flow Through FIFOs," in *16th Conference on Advanced Research in VLSI*, UC Santa Cruz, March 1995, pp. 76–90.

[2] R. W. Apperson, Z. Yu, M. J. Meeuwsen, T. Mohsenin, and B. M. Bass, "A Scalable Dual-Clock FIFO for Data Transfers Between Arbitrary and Haltable Clock Domains," *IEEE Transactions on Very Large Scale Integration*, vol. 15, no. 10, pp. 1125–1134, Oct 2007.

[3] Altera Corporation, *Single & Dual-Clock FIFO Megafunctions User Guide*, June 2003. [Online]. Available: http://www.altera.com/literature/ug/ug_fifo.pdf

[4] J. Ebergen, "Squaring the FIFO in GasP," in *7th International Symposium on Asynchronous Circuits and Systems*, March 2001, pp. 194–205.

[5] I. M. Panades and A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures," in *2nd International Symposium on Networks-on-Chip*. ACM/IEEE, April 2008, pp. 139–148.

[6] T. Chelcea and S. M. Nowick, "Low-latency asynchronous FIFO's using token rings," in *6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)*. IEEE, apr 2000, pp. 210–220.

[7] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland, "Two FIFO Ring Performance Experiments," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 297–307, Feb 1999.

[8] J. T. Yantchev, C. G. Huang, M. B. Josephs, and I. M. Nedelchev, "Low-latency asynchronous FIFO buffers," in *2nd Working Conference on Asynchronous Design Methodologies*, May 1995, pp. 24–31.

[9] 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.

[10] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proceedings of the Digital Automation Conference (DAC06)*. IEEE, July 2006, pp. 657–662.

[11] J. You, Y. Xu, H. Han, and K. S. Stevens, "Performance Evaluation of Elastic GALS Interfaces and Network Fabric," *Electronic Notes in Theoretical Computer Science*, vol. 200, no. 1, pp. 17–32, February 2008, elsevier.

[12] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 9, pp. 1059–1076, Sep 2001.

[13] H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers, "Synchronous interlocked pipelines," in *8th International Symposium on Asynchronous Circuits and Systems*, Apr. 2002, pp. 3–12.