A High-Speed GCD Chip: A Case Study in Asynchronous Design

Gennette Gill, John Hansen, Ankur Agiwal, Leandra Vicci and Montek Singh
University of North Carolina at Chapel Hill
Chapel Hill, NC, 27599-3175
{gillg,jbhansen,ankur,vicci,montek}@cs.unc.edu

***Abstract*—This paper presents the design of a greatest common divisor (GCD) chip as a case study in asynchronous or clockless design. The design uses fine-grain asynchronous pipelining to achieve fairly high performance. At the same time, the use of robust asynchronous handshaking in lieu of clocking allows the design to gracefully adapt its operation to voltage and temperature variations, without the need for clock recalibration.**

The design was fabricated in a 0.13μm CMOS process, using standard cells and with full testability support. Resulting chips were evaluated for performance and robustness, using a large set of test vectors for good fault coverage. Under nominal operating conditions (1.5V and 27°C), the fabricated parts were able to deliver up to 8 giga GCD algorithmic iterations per second (equivalent to 1 GHz clock speed). Moreover, they were functionally correct across a wide range of voltages (0.5V to 4V) and temperatures (-45°C to 150°C). This case study bolsters our confidence in the potential of aynchronous design techniques to help produce reliable ASICS that are fast, testable, and that operate under a wide range of conditions.

I. Introduction

This paper presents the design of a greatest common divisor (GCD) chip as a case study to investigate some of the potential benefits of asynchronous or clockless design. In particular, we are witnessing a trend where mobile applications increasingly require chips that can deliver high performance when needed, but that are ideally also capable of operating robustly on scaled-down supply voltages when battery life is at a premium [6]. While researchers believe that asynchronous technology can deliver chips to satisfy this need, very few concrete demonstrations of high-performance yet robust asynchronous designs exist [1]. The goal of our case study was to design, fabricate, test and evaluate a medium-complexity chip, and to report whether the goals of high performance, voltage and temperature robustness, and testability can be achieved while expending reasonably low designer effort.

The GCD chip presented in this paper implements a modified version of Euclid's iterative algorithm. A total of 98 self-timed stages connected into the shape of a ring implement the iterative algorithm. Each trip through the 98 stages represents eight algorithmic iterations (*i.e.,* an eight-way unrolled loop). This level of fine-grain pipelining results in a fairly high performance, equivalent to a 1 GHz clocked ASIC. However, unlike a synchronous implementation, the asynchronous nature of this design results in good robustness to temperature and voltage variation.

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. This work was supported in part by a contract through Boeing under the DARPA CLASS program. The chip described was first publicly demonstrated at ASYNC07.

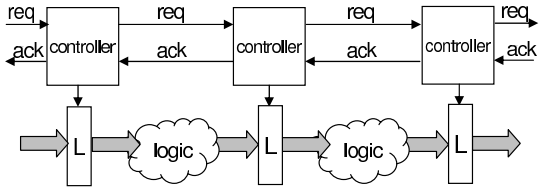

Fig. 1: A simple self-timed pipeline

The design was fabricated in a 0.13μm CMOS process, using standard cells and with full testability support. Since the design is clockless, its performance is measured and reported in terms of the number of GCD algorithmic iterations completed per second. A total of 24 parts were received and all were tested. At nominal temperature and voltage (1.5V and 27°C), the fabricated chips complete about 8 giga algorithm iterations per second (equivalent to 1 GHz clock speed). Testing was performed over a wide range of supply voltages (0.5–4.0V) and temperatures (-45–150°C), and all of the parts were found to be functionally correct. The asynchronous nature of the implementation allows the performance to gracefully vary with varying operating conditions without the need for clock recalibration. For example, at nominal temperature and 0.7V, which is less than half of the nominal voltage, the performance slows down to 1.4 giga iterations per second, but still remains functionally correct. At a temperature of 150°C but nominal voltage, a performance of 6.7 giga iterations per second is obtained. Finally, the implementation required a fairly modest designer effort—the entire physical design entailed 5 person-weeks of effort—which likely would not have been possible but for the inherent modularity of asynchronous components and the timing robustness of their interaction.

The remainder of this paper is organized as follows. Section II gives background on asynchronous pipelines in general, and on the Mousetrap pipeline style in particlar. Section III presents the GCD chip's design and implementation in detail. Section IV describes the test setup and presents experimental results. Finally, Section V gives conclusions.

II. Background

A. Asynchronous Pipelines

1) General Pipeline Operation: Figure 1 shows the basic structure of a bundled-data self-timed pipeline. Each pipeline

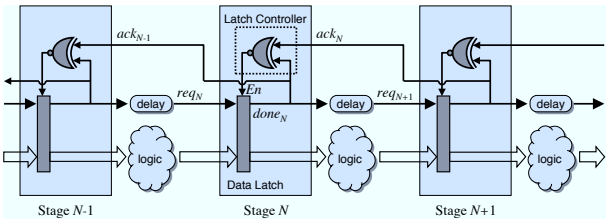

Fig. 2: A Mousetrap pipeline

stage consists of a controller, a storage element ("data latch"), and processing logic. In general, the data along with a request arrives at a stage, the data is latched, and the request is sent on to the next stage. Simultaneously, the stage acknowledges the receipt of the data to the previous stage. This "handshaking" allows the data to progress forward through the pipeline as each stage completes its computation.

2) Mousetrap Pipelines: Mousetrap [8] is a high-speed bundled-data asynchronous pipeline style that uses transition signaling for its handshake protocol and static logic gates in its datapath. A key feature of this pipeline style is that it can be efficiently implemented using only standard cells.

a) Structure: Figure 2 shows the structure of a fragment of a Mousetrap pipeline. Each stage consists of a data latch and single XNOR gate that serves as a latch controller. Function logic composed of static gates is inserted between the data latches, along with matched delay elements to appropriately delay the associated requests.

b) Operation: Initially, when the pipeline is empty, all its latches are transparent and all the *done, req* and *ack* signals are low. Each data item that enters the pipeline flips the values of all these signals *exactly once* as it passes through each successive stage. Thus, each transition, whether up or down, represents a distinct event: the arrival of a new data item.

c) Timing Constraints: Two simple one-sided timing constraints must be satisfied for the correct operation of Mousetrap pipelines: *setup time* and *hold time.* Setup time violations occur when the request arrives too late relative to the data, while hold time violations occur when the latch is kept transparent too long after data arrives. Setup time violations can result in data corruption, and hold time violations can result in either data corruption or loss of entire data items. [2]

B. Testing

As Shi *et al.* [7] describe, stuck-at faults in Mousetrap pipelines cause either deadlocks or loss of data. Both can be detected using functional testing, which does not require additional test circuitry. Mousetrap pipelines may also contain timing violations; Gill *et al.* [2] describe a non-intrusive functional testing method for detecting certain specific types of timing constraint (*i.e.,* setup and hold) violations for Mousetrap. It requires no test circuitry within the chip, but relies instead on carefully chosen test vectors applied in a specific order. While neither testing method requires the addition of test circuitry inside the chip, both do require sufficient controllability and observability inside the chip through the chip's external interface.

C. Performance Analysis

1) Pipeline Stages: Three key metrics characterize the performance of a single pipeline stage: (i) the *forward latency,* L_{Stage_i}, is the time it takes one data item to flow through $Stage_i$ assuming the stage was empty and ready; (ii) the *reverse latency,* R_{Stage_i}, is the time it takes a "hole" to flow backward through $Stage_i$ assuming the stage was initially full; and (iii) the *cycle time,* T_{Stage_i}, is the minimum time that must elapse between two successive data items entering or leaving that stage. The cycle time depends on the forward and reverse latencies and on the type of handshaking used. Typically, for full-capacity stages, a complete cycle consists of one forward and one reverse latency, so the cycle time is the sum of the two latencies: $T_{Stage_i} = L_{Stage_i} + R_{Stage_i}$. [10]

2) Self-Timed Rings: The classic work on analyzing self-timed rings is by Williams and Horowitz [10] and by Greenstreet et al. [3]. The throughput of the ring—measured as the number of data items crossing any stage boundary per second— is highly dependent on the ring's *occupancy, i.e.,* the number of data items revolving inside it. In particular, the plot of the maximum throughput versus occupancy, resembles the shape of a canopy, and will be referred to in this paper as a "canopy graph." Figure 3 shows an example.

Data Limited Operation. When the number of data items in the ring is small, the throughput is low because the stages are underutilized, and the pipeline is said to be "data limited." In particular, if there are k items in the ring, then in the time a particular data item completes one revolution around the ring (*i.e.,* $\sum_i L_{Stage_i}$), all k items would have crossed any stage boundary in the ring. Hence, the maximum ring throughput is proportional to the ring occupancy: $tpt_{Ring} \leq k/\sum_i L_{Stage_i}$.

Hole Limited Operation. If the ring is filled with data items in nearly all stages, then the ring throughput is limited because holes are needed to allow data items to flow through the pipeline; the pipeline is said to be "hole limited." If there are h holes in the ring, then in the time a particular hole completes one revolution around the ring (*i.e.,* $\sum_i R_{Stage_i}$), all h holes would have crossed any stage boundary in the ring, traveling in a direction opposite to data. Hence, h data items would have crossed any stage boundary in the forward direction. Thus, if N is the number of stages in the ring, then $h = N - k$, and the maximum ring throughput is proportional to the number of holes: $tpt_{Ring} \leq (N-k)/\sum_i R_{Stage_i}$.

III. Design and Implementation

Figure 4 shows the overall architecture of the core of the GCD chip. The design is organized as a self-timed ring, which iteratively computes the GCD function. To simply implementation, the operand width was chosen to be 8 bits, though the architecture described here could be adapted for use with wider operands (with proper control buffering). The datapath of the entire ring was pipelined into a total of 98 stages, each implemented Mousetrap pipeline stages. Specifically, the body

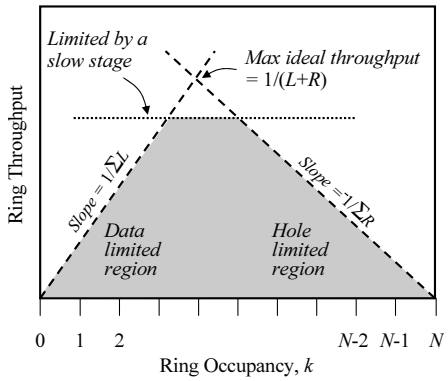

Fig. 3: The throughput of a ring as a function of the number of data items (a "canopy graph")

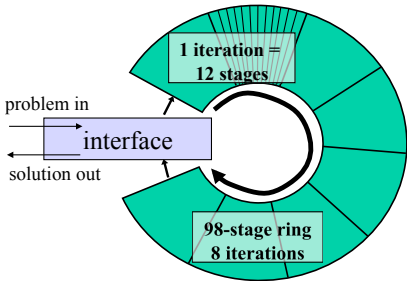

Fig. 4: Overall architecture of GCD chip

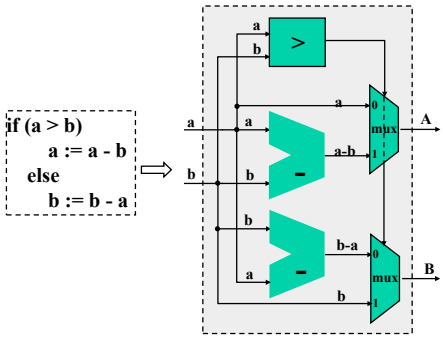

Fig. 5: Basic GCD algorithm, and the corresponding dataflow implementation of one iteration

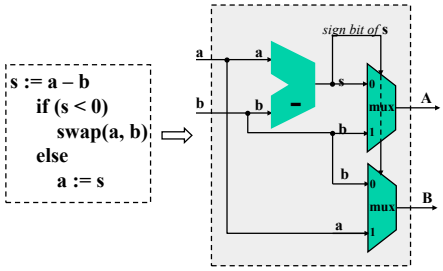

Fig. 6: Area-optimized GCD algorithm, and the dataflow implementation of one iteration

of the GCD algorithm is repeated 8 times to create a ring, and each iteration comprises 12 stages. Two "corner-turner" stages were also added. The rest of this section details the design of an individual GCD iteration as well as the ring interface.

A. Implementation of a Single Iteration

1) Datapath Design and Optimization: Figure 5 shows the body of the iterative GCD algorithm along with a straightforward implementation. Both $(a-b)$ and $(b-a)$ are computed using two separate subtract units, and then the values of a and b are updated depending upon the test $(a > b)$. This straightforward implementation has significant area overheads, because two subtract units are used as well as a comparator. Figure 6 shows a simple way to optimize the loop body that requires only a single subtract operation along with a conditional swap based on the sign bit of the subtraction.

2) Pipelining an Iteration: The pipelined implementation one algorithmic iteration, shown in Figure 6, consists of 12 Mousetrap pipeline stages. Eight stages make up the subtract block while an additional four stages implement the conditional swap. Figure 7 shows the layout of one iteration.

a) Subtract: The subtract block consists of a simple, ripple-borrow, which is identical in architecture to a ripple-carry adder [4]. If X and Y are the operand bits at a given stage and B_{in} is the input borrow, then the result D and the borrow output B_{out} generated at this stage are given by:

$$D = X \oplus Y \oplus B_{in} \quad (1)$$

$$B_{out} = \overline{X} \cdot (Y + B_{in}) + Y \cdot B_{in} \quad (2)$$

The left portion of Figure 7 shows the layout of the eight stages of the subtract block.

b) Conditional Swap: The swap operation uses multiplexors and consists of four pipeline stages. This level of pipelining reduces the load driven by each individual stage and facilitates routing of the data bits. The right portion of Figure 7 shows the layout of the four stages of the conditional swap.

3) Stage Implementation Details: This section presents details of the implementation of each pipeline stage, including design decisions to improve robustness. Figure 8 shows the detailed implementation of each pipeline stage.

a) Datapath: The datapath consists of standard level-sensitive D-type transparent latches and static logic function blocks (either subtract logic or swap logic) that are implemented using gates from a standard-cell library.

b) Control: The control logic consists of the XNOR gate, its output buffers, and a matched delay, all of which were implemented using standard gates. Two series inverters amplify the controller's output, in order to provide adequate strength to drive the entire 24 bit datapath (8 bits each for a, b and s). A matched delay consisting of four series inverters was used which, for all stages, adequately matched the worst-case delay through the function logic.

c) Robustness Considerations: Two design decisions were made in order to ensure high robustness to timing variations. First, buffer insertion did not employ the technique of *control kiting* [5], [11]. In kiting, the control signal is sent ahead of the data, with the assumption that the data will arrive

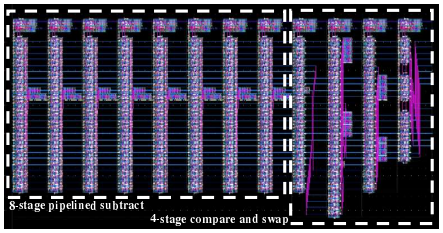

Fig. 7: Layout of one iteration

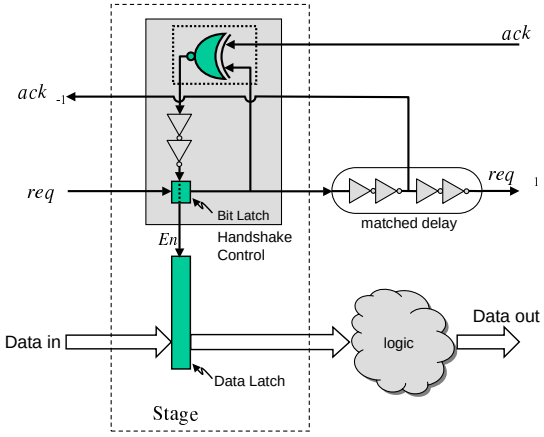

Fig. 8: Implementation of a stage: delay matching and control buffering

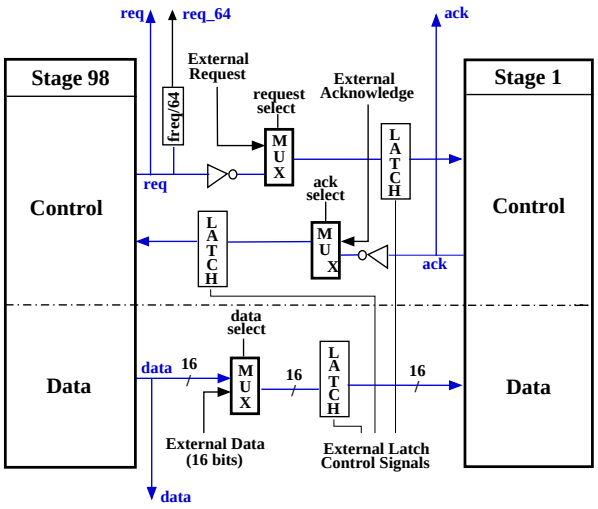

Fig. 9: Detail of the chip interface

before the latch becomes opaque. Kiting has the disadvantage of reducing timing margins, thereby sacrificing robustness. Therefore, this work did not use kiting, opting instead for higher timing robustness. Second, as shown in Figure 8, the acknowledgements were delayed by two inverter delays before being sent to the previous stage. This delay was added in order to give the current stage additional time to make its latch opaque before the acknowledgement stimulates the previous stage to generate the next data item. Overall, the cost of higher timing robustness was an increase in the cycle time by four inverter delays: two inverters due to control buffering, and another two due to the delayed acknowledgments.

B. Interface

To aid in testing, the GCD interface allows full controllability of all inputs and full observability of all outputs. In addition, during steady-state operation the interface provides a decimated version of the ring's frequency (divided by 64) so that slow testing equipment can be reliably used to determine the frequency.

1) Structure: As seen in Figure 9, the interface allows the external environment to passively observe the request, acknowledge, and data signals, as well as to actively control them. In addition, the interface contains a frequency divider—consisting of a series of six edge-triggered flip-flops—that produces a frequency-decimated version of the request signal (frequency divided by 64). To provide the environment with full control, multiplexers and latches are added to the path of each signal. This signals allow the environment to both stop and start the cycling of data through the ring.

2) Metastability: Since the interface allows the chip to be stopped suddenly during operation by essentially "cutting" the ring open using the latch and multiplexer signals, the interface has some non-zero probability of causing a metastable condition. In a metastable scenario, the acknowledge leaving the first stage is in the process of transitioning *at the same instant* that the environment makes the latch on the acknowledge path opaque. This causes the outgoing acknowledgement to erroneously toggle several times, thereby sending incorrect signals to the environment. In such a situation, one or more data items may be lost from the last stage of the ring. In order to estimate the likelihood of the latter harmful metastable scenario, we performed over 60,000 tests on our fabricated chip; none resulted in any loss of data.

3) Support for Testability: The interface allows enough controllability of inputs and observability of outputs to support testing for stuck-at faults [7] and timing violations [2]. In particular, the interface allows for a new data input to be applied at each of the input latches simultaneously. Additionally, it allows the environment to remove individual data items while observing the output. These are both necessary conditions for exposing timing violations in a high-speed pipeline.

IV. Results

A. Layout and Fabrication

Our chip was implemented in a 0.13μm process (IBM 8RF CMOS). It was designed using the Cadence tool suite, using a standard-cell library from the University of Washington.

Two versions of the GCD ring were fabricated on the same die: a slower version with conservative timing margins and a faster version with slightly tighter timing margins. Each version contains 50K transistors and occupies 1.5X.5mm^2. Figure 10 shows the layout of the completed chip and a photomicrograph of the fabricated die. Both versions were

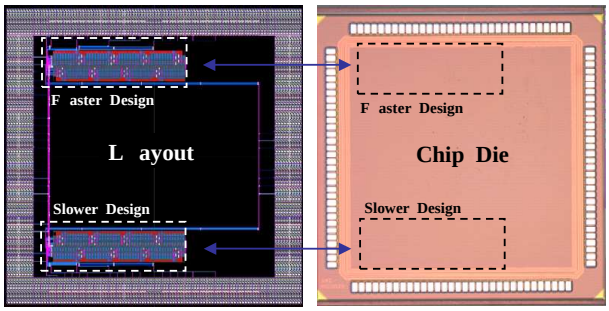

Fig. 10: Fabricated chip layout and photomicrograph

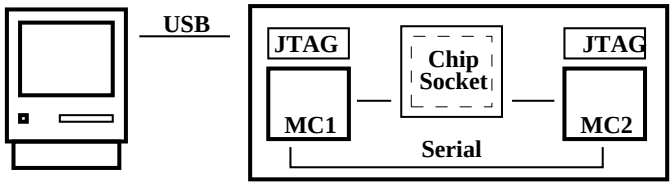

Fig. 11: Testing and evaluation setup

found fully functional during testing. Results for the faster version are presented here; the throughput of the slower version was about 15% lower.

Since the design was standard-cell, only coarse-grain gate sizing and delay matching could be performed. In particular, ideal gate sizes were computed using Logical Effort [9], and then standard gates were chosen from the library that most closely fit these calculations. SPICE simulations were performed to estimate timing; all timing constraints were satisfied with healthy margins.

B. Testing and Evaluation

1) Board Design and Operation: A custom test board, shown in Figure 11, has a socket where the GCD chip is inserted for testing. The chip interacts with two TI MSP430 microcontrollers: the left microcontroller manages data and control to the chip, and the right microcontroller reads results from the chip, including the operating frequency at the interface. The microcontrollers communicate with each other using a two-wire serial connection. The results are relayed to a PC via on-board JTAG headers.

The operation of the board is in turn managed on the PC using a custom application written in Java. The user inputs data on the PC and sends it to the left microcontroller through a USB connection. The microcontroller then initializes the GCD chip, loads data into the ring, and begins the ring operation. The operating throughput is measured by the right microcontroller and reported to the PC. Finally, the left microcontroller opens the ring and drain the results, which are sent to the PC for display and verification.

2) Test Operation: The operation of the chip consists of the following four phases in sequence: (i) initialization, (ii) loading of problem instances, (iii) computation by the ring, and (iv) draining of results.

During initialization, an externally-supplied *reset* signal initializes all of the pipeline stages in the ring to be empty. Next the environment loads a set number of problem instances by toggling the ingoing request signal while supplying the desired test patterns.

The environment begins the computation phase by setting the multiplexers in the interface to transmit signals between the first and last ring stages. During the computation phase, the external environment can observe the request and the toggling at the interface of the chip. Since the ring was designed to operate in the gigahertz range, it also includes divide-by-64 circuitry that produces a signal slow enough for our testing equipment to read.

Finally, the environment drains the results. First it stops the computation by making the interface latches opaque so no requests can cycle through the ring. Then it toggles the acknowledgement signal to the last pipeline stage while reading out each resulting dataset one by one.

3) Fault Testing: The test patterns generated for this circuit according to the approach of [7] gave over 98% fault coverage. For testing violations of the two timing constraints of Mousetrap, a custom tool implementing the the non-intrusive functional test approach of [2] was used. Test patterns generated achieved 100% coverage for hold time violations and about 33% coverage for setup time violations, yielding an overall 66% fault coverage.

C. Results and Discussion

A total of 24 chip samples were tested, and all were found fully functional, with no faults detected. Figure 12 presents the throughput measured at various ring occupancies. The results for the fastest and slowest chips tested are shown. The fastest chip reported 1.01 billion toggles per second at the observable interface to the chip, which indicates that the chip was completing about 8.08 billion GCD iterations each second, while the slowest chip was about 10% slower.

Note that the shape of the graph of Figure 12 agrees with the expected performance of a pipelined ring as discussed in Section II-C2. Although individual stages within the chip are not directly observable, their forward and reverse latencies can still be estimated. In particular, the total forward latency is inversely proportional to the slope of the line in the data-limited region, and the total reverse latency is inversely proportional to the slope of the line in the hole-limited region. The forward latency of an individual stage is therefore estimated to be 170ps and the reverse latency 300ps.

1) Robustness Tests: In addition to basic test and evaluation, the chips were subjected to extensive robustness tests by varying temperature and voltage quite beyond nominal operating conditions. The results below show performance when the occupancy is 45, which is within the peak performance range of the canopy graph; the effect on performance at this occupancy is representative of the effects at other occupancies.

Figure 13 demonstrates the effect of scaling the voltage on peak performance. Extensive testing was performed over the range of 0.5V–4V. The throughput changed automatically

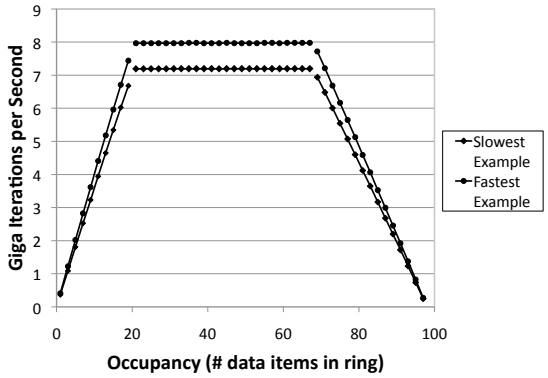

Fig. 12: Throughput vs. occupancy for the fastest and slowest chips tested

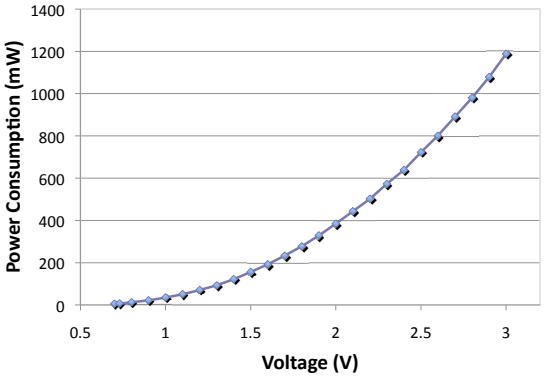

Fig. 14: Power consumption varies with voltage

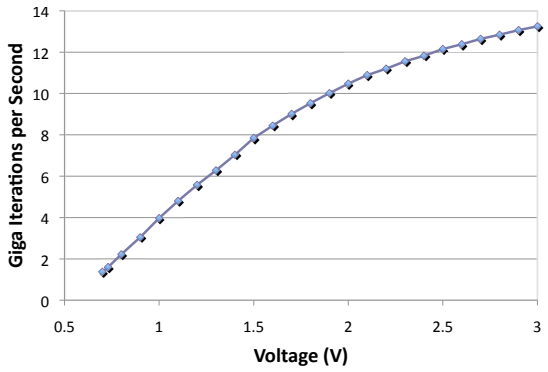

Fig. 13: Speed increases with increased voltage

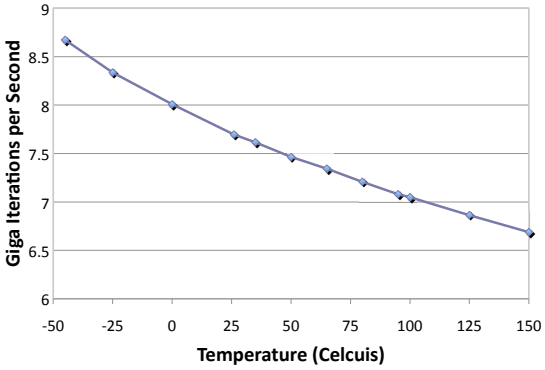

Fig. 15: Impact of temperature on performance

with the scaled voltage, and correct operation was observed throughout this range. Figure 14 shows power consumption (in milliWatts) for the chip. At nominal voltage, the peak power consumption (measured at peak throughput) was 75 mW, and the power consumption increases with increasing voltage.

Finally, Figure 15 demonstrates the effect of temperature on the chip's throughput. This test was performed at nominal voltage. As expected, throughput decreases with an increase in temperature. The chip was fully functional over the entire range of temperatures tested (-45°C to 150°).

V. Conclusion

This paper presented the design and analysis of a GCD chip as a case study in asynchronous design. It was implemented as a self-timed ring using Mousetrap pipeline style [8], and fabricated in a 0.13μm CMOS process. The chip was tested for stuck-at faults and timing constraint violations, and evaluated for performance over a wide range of supply voltages (0.5V to 4V) and temperatures (from -45°C to 150°C). A total of 24 parts were received, and all were tested to be functionally correct. The performance obtained was about 8 giga GCD algorithmic iterations completed per second, which is equivalent to a 1 GHz clocked ASIC. This case study bolsters our confidence that asynchronous design techniques can be useful for the design of reliable ASICS that are fast, testable, and that can operate under a wide range of conditions.

References

[1] U. Cummings. Terabit crossbar switch core for multi-clock-domain SoCs. In *Symposium Record of Hot Chips 15*, Aug. 2003.

[2] G. Gill, A. Agiwal, M. Singh, F. Shi, and Y. Makris. Low-overhead testing of delay faults in high-speed asynchronous pipelines. In *Intl. Symp. on Asynchronous Circuits and Systems*, Mar. 2006.

[3] M. R. Greenstreet and K. Steiglitz. Bubbles can make self-timed pipelines fast. *Journal of VLSI Signal Processing*, 2(3):139–148, Nov. 1990.

[4] K. Hwang. *Computer Arithmetic: Principles, Architecture, and Design*. Wiley, 1979.

[5] 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*, 87(2):297–307, Feb. 1999.

[6] J. M. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996.

[7] F. Shi, Y. Makris, S. M. Nowick, and M. Singh. Test generation for ultra-high-speed asynchronous pipelines. In *Proc. Int. Test Conf.*, Nov. 2005.

[8] M. Singh and S. M. Nowick. Mousetrap: High-speed transition-signaling asynchronous pipelines. *IEEE Trans. on VLSI Systems*, 15(6):684–698, June 2007.

[9] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, Inc., 1999.

[10] T. E. Williams, M. Horowitz, R. L. Alverson, and T. S. Yang. A self-timed chip for division. In P. Losleben, editor, *Advanced Research in VLSI*, pages 75–95. MIT Press, 1987.

[11] K. Y. Yun, P. A. Beerel, and J. Arceo. High-performance asynchronous pipeline circuits. In *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, Mar. 1996.