The asynchronous pipeline logic ‚MOUSETRAP‘

Erman Culhacik
Department of Computer Engineering (ZITI)
The University of Heidelberg
68159 Mannheim, Germany
culhacik@stud.uni-heidelberg.de

***Abstract*—In the paper MOUSETRAP by M. Singh and S. M. Nowick in 2001 the authors stated that the MOUSETRAP has developed and in this paper information was given about MOUSETRAP, which has the highest throughput and lowest energy consumption. Initial pre-layout HSPICE simulations of a 10 stage FIFO on a 16 bit wide data path indicate throughput of 3.51 GHz in 250 nm TSMC CMOS, using a conservative process. In the next step, Post-layout SPICE simulations of a ten-stage pipeline with a 4 bit and 16 bit wide data path indicate throughputs of 2.1–2.4 GHz in an 180 nm TSMC CMOS process. In this study there is a research about asynchronous pipeline logic MOUSETRAP. Some sentences which describe this subject well cite in different references.**

Keywords—MOUSETRAP; Asynchronous pipeline; transition signaling; clocked CMOS (C²CMOS); Pipeline; Latch controllers; pipeline processing; high performance; low power; low energie consumption.

I. INTRODUCTION

MOUSETRAP [1] was introduced in 2007 for high speed applications. This application proposed MOUSETRAP as a latch-based implementation for two phase handshake protocols. These latches are similar to the D latches.

Asynchronous circuits have some opportunity like high performance, low power, improved noise and electromagnetic compatibility (EMC) properties, and a natural match with heterogeneous system timing than synchronous circuits [2]. Asynchronous circuits have a natural elasticity, therefore the size of data and speed of external interfaces can vary dynamically. This system can be facilitating modular and reusable design. According to the results asynchronous pipelines have very high throughput. Besides, asynchronous pipelines consume less energy than synchronous pipelines.

In a new asynchronous pipeline which is MOUSETRAP pipeline the data path uses standard small and fast transparent latches. The asynchronous control consist of a single gate per pipeline stage for a basic linear pipeline. Per pipeline stages relate only with immediate neighbour pipeline stages. The timing constraints are all local, simple and one-sided [1].

There are various stages of general FIFO MOUSETRAP. With Clocked CMOS dual-rail XNOR and various speed-up experiments using, the faster and more energy efficient pipeline MOUSETRAP has been tried to be created. Initially, MOUSETRAP pipeline created as linear pipeline, with Fork and Join phase it has become non-linear pipeline. To accomplish this operation, Müller C-element was utilized. Two key measures of the performance of the pipeline are discussed: forward latency and cycle time and also for Non-linear MOUSETRAP pipelining fork and join calculations was displayed.

II. PIPELINING

A. Asynchronous & Synronous Pipelines

Fig.1 shows a synchronous pipeline and an asynchronous pipeline. The focus is simply on the local stage communication. Each asynchronous stage including both data and control links. Communication is usually single-rail that transfers the data to one direction on only way, and is implemented by a handshaking protocol: data is sent from left to right, and an acknowledgement control signal is sent from right to left.

Synchronous and asynchronous pipeline have differences which shown in Fig 1. In synchronous pipelines, the clock drives the all system in clock step: every data item moves to the next stage on the active clock edge. Hence, the pipeline acts as a form of synchronous shift register with computation blocks. Also, each stage's critical-path delay must be less than the fixed clock period. As a result, all stages typically have balanced delays.

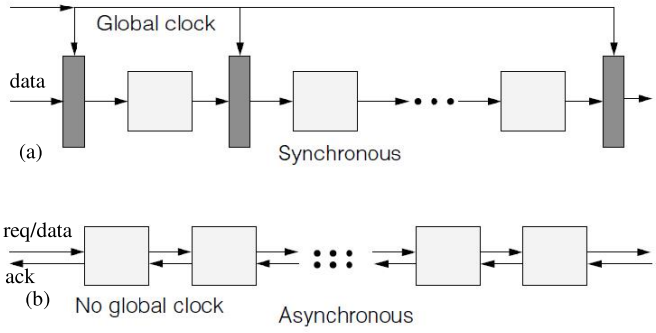


Fig.1. (a) synchronous pipelines, (b) asynchronous pipelines [3].

Asynchronous pipelines have no main clock. The advance of data items is stage controlled. Typically, a stage *N* can accept a new data item if two conditions hold: its left neighbour, stage *N-1*, is providing new data; and its right neighbour, stage *N+1*, has stored (Capture Protocol) the previous data item.

Ref [3] gives an introduction about four important features of asynchronous approach: "First, stages need not have equal delays. In most synchronous systems, the worst-case stage delay must be less than the clock period. In contrast, in an asynchronous system, although balanced stages tend to provide the highest system performance, this balance is not a requirement for correct operation. Consequently, stages of widely different static delays can be concatenated to form a working system. In addition, in some asynchronous pipeline styles, each stage may have a dynamically varying delay. Hence, this per-stage variability can naturally be exploited to improve average system latency and throughput. Second, asynchronous pipelines inherently provide elasticity. If there is no congestion, data items are widely spaced in the pipeline and travel rapidly through. If input rates are higher, spacing becomes tighter between items. In the extreme case, with a slow or stalled output environment, data items become bunched or stalled at close intervals. In all cases, there is no wait for a clock edge. Hence, the token spacing and the throughput rate are determined dynamically. Third, asynchronous pipelines automatically provide flow control. In contrast, synchronous pipelines by default include no flow control. Synchronous flow control is typically supported using complex decoupled latch control. A wait signal, used for back pressure, must also be synchronized to the clock at every stage. Fourth, asynchronous pipelines consume dynamic power only on demand. That is, switching activity occurs only when data items are being processed. Furthermore, asynchronous pipelines inherently obviate the need for global clock distribution power".

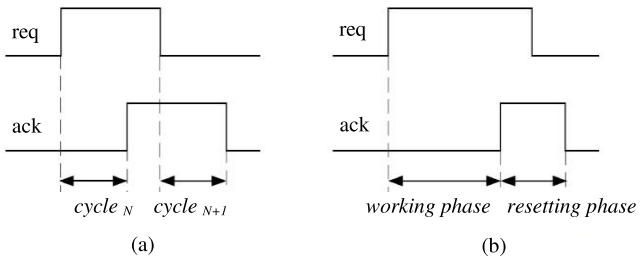


Fig.2. (a) 2-phase protocol, (b) 4-phase protocol [4].

B. Two-Phase and Four-Phase Handshaking

In two-phase handshaking a single request transition from the sender starts a transaction, which is followed by a single acknowledgement transition from the receiver. The protocol requires only one round-trip communication per transaction [3]. Ref [4] gives an introduction for two and four-phase handshaking: "In 2-phase protocol, *req* and *ack* are identified at a transition of the control signals either from low-to-high or high-to-low, and the levels of control signals have no significance. As shown in Fig. 2(a), a whole *req-ack* cycle is completed when both signals make the same transition. MOUSETRAP, a simple and robust linear pipeline controller, is based on 2-phase protocol. In four-phase handshaking as shown in Fig. 2(b), a given cycle working phase and resetting phase. From the rising-edge of *req* to the rising-edge of *ack* is the working phase where a request is handled and completion is notified. The return-to-zero of both *req* and *ack* signals occurs the resetting phase [4]. 4-phase handshaking also causes more delay than 2-phase handshaking."

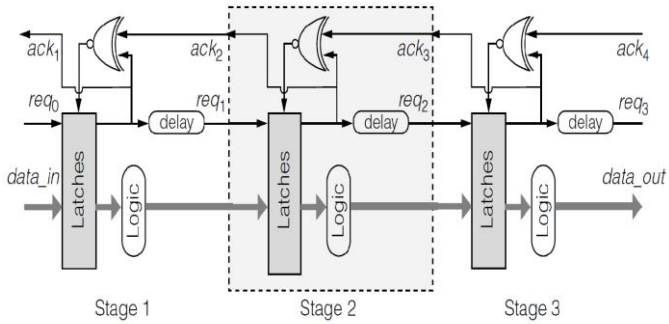


Fig.3. MOUSETRAP Pipeline [3].

III. The Mousetrap Pipeline

MOUSETRAP developed at Columbia University to be a high-performance pipeline that supports the use of a standard cell methodology. It uses capture-pass protocol like micropipelines, it has less complex signalling and far lower overhead, and it uses simpler XNOR-controller and data latches. Fig. 3 shows a basic MOUSETRAP pipeline. MOUSETRAP Pipeline uses a 2-phase protocol. XNOR provides local control for each stage [3]. In simple FIFO MOUSETAP used D-latches to Data capture protocol.

Ref [3] gives an introduction about the MOUSETRAP pipelining and how to use latchs and controller for doing signalling: "In MOUSETRAP, suppose all *reqN* and *ackN* signals are at 0, and local latch controls at 1. All data latches are therefore transparent, forming a flow-through combinational system. Once data arrives at the left channel, it passes directly through the entire series of data latches to the right channel. At each stage, after the data arrives and passes through its latches, the corresponding *reqN* bundling signal toggles from 0 to 1. As a result, the stage's XNOR2 control output toggles from 1 to 0 and captures the data, thereby protecting it from being overwritten by any new data item from the left neighbour. At the same time, when data is about to be captured at any stage n, the stage requests the next data item from its left neighbour. In particular, stage *N* signals to its predecessor, stage *N-1*, through a transition on its *ackN* output from 0 to 1. This enables the pass operation of stage *N-1*, thereby causing the latches of stage n-1 to become transparent. This backward synchronization effectively indicates that stage *N* is in the process of safely storing the current data item, and informs stage *N-1* that its output will no longer be needed and can be overwritten. Likewise, stage *N* itself enters its pass phase when stage *N+1* indicates its data is being stored. Subsequently, when the next data item enters the pipeline, the same two-phase protocol is repeated, but with the bundling signal at each stage toggling in reverse from 1 to 0."

The latching action by a pipeline stage is analogous to the operation of a household MOUSETRAP: latches remain transparent before data arrives; they are closed as soon as data passes through. It is important to note that this behaviour is very different from that of most synchronous, and many

asynchronous, pipelines in which latches are opened only after new data arrives.

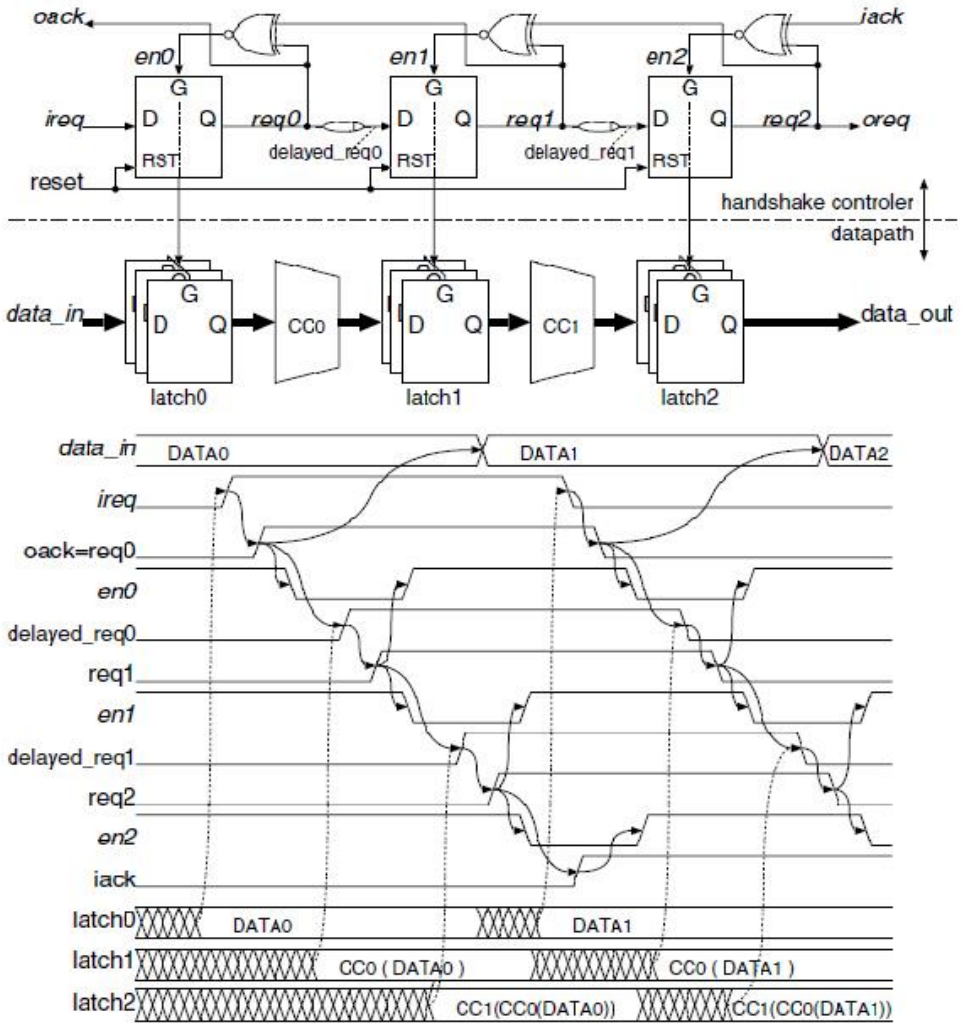


Fig.4. MOUSETRAP Pipeline and timechart [5].

As shown in Fig.4, MOUSETRAP tested by a test pattern. A Timing chart for this test are given just under the MOUSETRAP. In this display, the output request signal *req0* from the first stage is delayed by the delay element whose delay value matches to the critical path delay of the corresponding combinational circuit CC0. CC0 data arrives the next stage with approximately *delayed_req0* [5].

A. General Pipeline Implementation

Only pipelines without logic processing simple FIFOs, have been considered. Fig. 3 shows how basic logic processing can be added to the asynchronous pipeline. Blocks of combinational logic and matching delay elements are simply inserted between pipeline stages.

The standard asynchronous bundled data scheme is used: *reqN* must arrive at stage *N* after the data inputs to that stage have stabilized. Therefore, the latency of the delay element must match the worst-case delay through the combinational block. A benefit of this approach is that the data path itself can use standard single-rail blocks. Moreover, even when worst-case matched delays are used, this approach has two advantages over synchronous design: different pipeline stages (different delays) and local delay (not by a global clock) [1].

B. Gate-Level Piepelines Using C^2MOS

To target extremely high throughput, as a special case, gate level pipelines can be used. As an additional benefit, the absence of latches also translates into savings of chip area and power consumption. Fig. 5 shows the structure of a general clocked CMOS gate. The clock input *En* directly controls the gate through two transistors. When *En* is asserted, the gate is enabled and a new output is produced. When *En* is deasserted, the gate holds its output value with a Keeper logic. clocked CMOS has been proposed as a synchronous technique, but it can be naturally adapted to very high-speed asynchronous pipelines using local handshake signals to replace the clock [1].

C. Dual-Rail XNOR Optimization

To eliminate gate delays from the critical path Dual-Rail XNOR optimization developed. Since many transparent latches as well as clocked CMOS gates require both true and complemented enables, a useful optimization for both of the proposed pipeline schemes is to implement the XNOR as a dual-rail gate, providing both XOR and XNOR outputs. In dual-rail

XNOR as shown in Fig. 6, has two acknowledgement and two done signals [1]. With this method dual-rail XNOR shows only one logical delay. A single rail XNOR showed that has a minimum 2 logic delay.

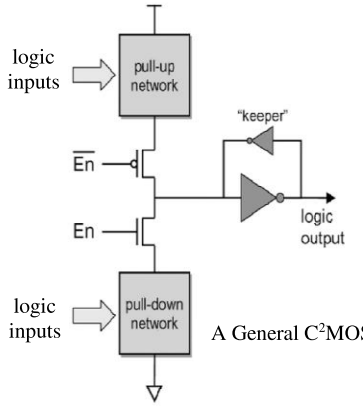


Fig.5. Clocked CMOS Logic[1].

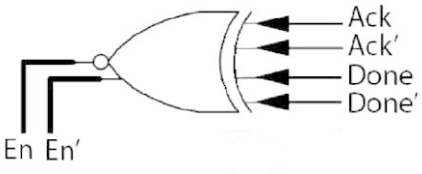


Fig.6. Dual-Rail XNOR.

D. Pipeline Performance and Timing Constraints

1. *Performance:* The pipelining method has two performance measurements: forward latency and cycle time.

   Forward latency is the time it takes a data item to pass through an firstly empty pipeline. Since, in an empty pipeline, all the latches are transparent, the pipeline latency per stage L is simply the stage's latch delay + logic delay.

$$L = T_{Latch} + T_{Logic} \quad (1)$$

   Cycle time is the time between successive data items from the pipeline when the pipeline is operating at maximum speed.

$$L = 2.T_{Latch} + T_{Logic} + T_{XNOR\uparrow} \quad (2)$$

   For the special case of C2MOS pipelines, there are no explicit latches. If the delay through a C^2MOS gate is denoted by T_{C^2MOS} , the latency and cycle time are given by

$$L_{C^2MOS} = T_{C^2MOS} \quad (3)$$

$$T_{C^2MOS} = 2.T_{C^2MOS} + T_{XNOR\uparrow} \quad (4)$$

2. *Timing Constraints:* Setup and hold time is calculated on the Ref[1]. There are two simple one-sided timing constraints: setup time and hold time, which must be satisfied for the correct operation of the pipeline.

   Setup Time: Once a latch is enabled and receives new data at its inputs, it must remain transparent long enough for data to pass through. "XNOR switching low"(*Treq to En*) must be longer than the setup time. This constraint is usually easily satisfied because the delay from *reqN* to *doneN* typically exceeds the setup time.

$$T_{reqN} \rightarrow done N + T_{XNOR\,N\downarrow} > T_{setup} \quad (5)$$

   Hold Time: Once data enters a stage, it should be securely captured before new data is produced by the previous stage. Otherwise, stage *N*'s data will be overwritten by new data. Therefore, since *ackN-1* and *doneN* are generated in parallel, the path from *ackN-1* to stage *N*'s data inputs must be longer than the time to close *N*'s latch, plus a hold time. Eq. (7) shows the Hold time is smaller than forward latency of previous stage.

$$T_{XNOR\,N-1\uparrow} + T_{Latch\,N-1} + T_{Logic\,N-1} > T_{XNOR\downarrow} + T_{hold} \quad (6)$$

$$T_{XNOR\,N-1\uparrow} \approx T_{XNOR\,N\downarrow}$$

$$T_{Latch\,N-1} + T_{Logic\,N-1} > T_{hold} \quad (7)$$

E. Handling Wide Datapaths

In asynchronous pipelines a single control signal for a pipeline stage must be broadcast across many latches. In principle, such control distribution may introduce sizable delays in the critical path. The most effective way to solve this is "Control Kiting" per latch to use. Control kiting is buffer to skewed Enable of Latch [1]. For each Data different size delay element is used. This is necessary to reach at the same time data in latch.

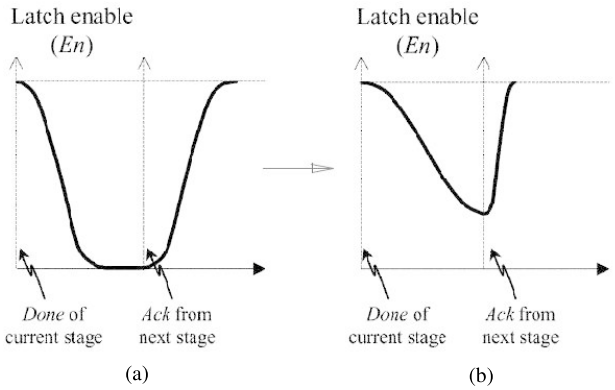


Fig.7 (a) no waveform shaping, (b) waveform shaping: Reduce voltage swing [1].

F. Pipeline Speed-up

A circuit-level optimization can further improve the pipeline's performance under steady-state operation. The key idea is to shape the output of the latch controllers through transistor sizing, such that the critical cycle is further shortened at the expense of some loss of timing margins.

There is a no waveform shaping in Fig. 7(a). It shows a normal operation of XNOR gate. Voltage swing reduction as shown in Fig. 7(b); if waveform shaping is applied beyond the limiting case, the controller output no longer exhibits a full voltage swing. It is under steady-state operation, the pipeline latches are never fully disabled. The re-enabling of the latches occurs even faster, hereby further shortening the cycle time and improving the throughput.

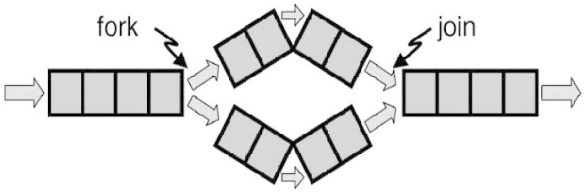


Fig.8 Nonlinear Pipelining: Fork and Join [1].

G. Nonlinear Pipelining

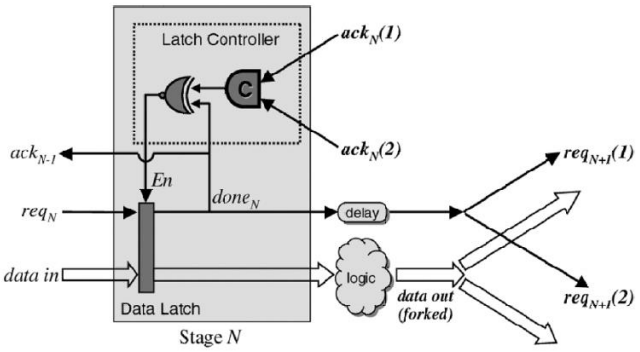
According to the Ref[1]: "In complex system architectures, nonlinear pipelining is often needed. There are two simple implementations "fork and join" of MOUSETRAP. Fig. 8 shows Fork and Join phases in nonlinear pipelining. In a fork stage, the data output and corresponding *req* are both simply forked to the two or more destination stages. In turn, the two or more *ack* signals are combined through a Müller C-element. A symmetric C-element makes a transition when all of its inputs change exactly once. In a join stage, the *ack* is simply a forked wire communicating with all sender stages, but there are multiple *req*'s that must be combined. Once again an asymmetric C-element can simply be used to combine the multiple requests, and the result treated as a unified request that is fed into the latch."

For a fork stage as shown in Fig. 9(a), cycle time increase from that in Eq. (3) by an amount equal to the latency of the C-element (T_C). As shown in Fig. 10, there are minimum 2 input of C-element. If C-elements each input high or low at the same time, output would be changed, otherwise the output keep in memory previous value.

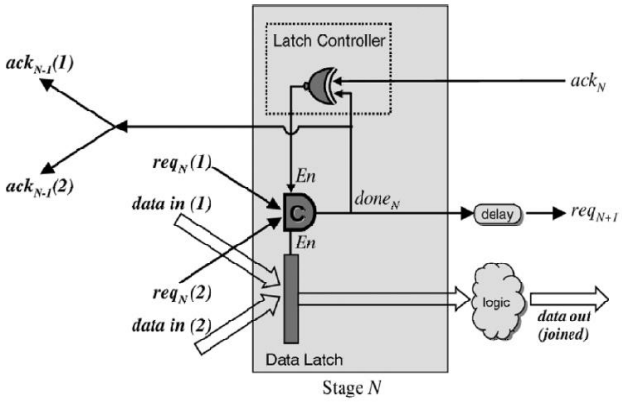
$$T_{fork} = 2.T_{Latch} + T_{Logic} + T_C + T_{XNOR\uparrow} \quad (8)$$

For a join stage as shown in Fig. 9(b), cycle time doesn't change much (T_{aC}). This C-element was used instead of latch element. Therefore there is less delay than T_{fork} delay. T_{aC} named as an asymmetric C-element. As shown in Fig. 11, there are minimum 2 input of C-element. If C-elements each input high or low and enable be high at the same time, output would be changed, otherwise the output keep in memory previous value.

$$T_{join} = T_{aC} + T_{Logic} + T_{Latch} + T_{XNOR\uparrow} \quad (9)$$



(a)



(b)

Fig.9 (a) Fork stage, (b) Join stage [1].

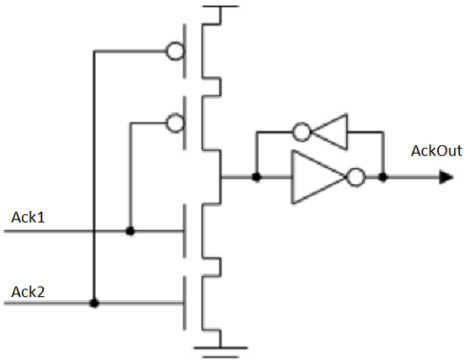


Fig.10 A symmetric C-element.

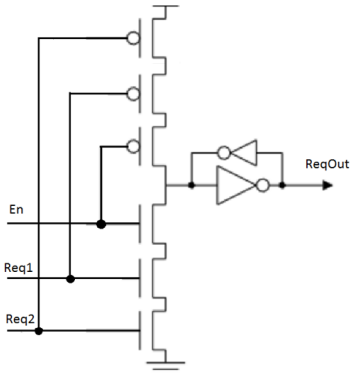


Fig.11 An asymmetric C-element.

IV. Experimential Results

According to the Ref[1]: "Results of post-layout SPICE simulation for basic MOUSETRAP pipelines are presented. A simple 10-stage FIFO was simulated. The FIFO was laid out using the Cadence tool suite in an 180 nm TSMC process. Two versions of the pipeline were simulated: the "un-optimized" pipeline style and an "optimized": Voltage swing reduced. Both a 4 bit FIFO and a 16 bit FIFO were simulated for the un-optimized style. Identical control circuits were used in both the cases, but control kiting was used for the 16 bit design. For the optimized style, the waveform shaping optimization was performed on the 4 bit FIFO to obtain further improvement in throughput, though at the expense of some loss of timing margins. A pass-gate implementation of a dual-rail XNOR/XOR pair, and an eight-transistor dynamic D-latch."

Pipeline Design	latch delay t_{Latch} (ps)	XNOR delay $t_{\text{xnor}\uparrow}$ (ps)	$t_{\text{xnor}\downarrow}$ (ps)
MOUSETRAP	188	102	115
MOUSETRAP$_{opt}$	179	63	131

Pipeline Design	Cycle Time, T Analytical Formula	(ps)	Throughput (GigaHertz)
MOUSETRAP	$2 \cdot t_{\text{Latch}} + t_{\text{xnor}\uparrow}$	477	2.10
MOUSETRAP$_{opt}$	$2 \cdot t_{\text{Latch}} + t_{\text{xnor}\uparrow}$	421	2.38

Table.1 Performans of Moustrap FIFO 180 nm TSMC Technologie [1].

In Table 1 the simulation results summarized. The overall pipeline cycle time T is given. The latch delay and Controller gate delays are also given. First row is for 4 bit and 16 bit FIFOs without the optimization. Second rows shows an optimized performance of MOUSETRAP pipeline with 4 bit data wide. Un-optimized MOUSETRAP has 2.1 GHz throughput but Optimized MOUSETRAP pipeline has 2.38 GHz throughput (13% improvement.).

According to the Ref[3]: "These speeds compare with IPMOS style of Schuster. Their results have 3.3-4.5 GHz for High-Performance IBM 180 nm process. But IBM 180 nm technologies are faster than the TSMC process.

Different pipeline styles reviewed and represent different trade-offs between performance, power, and ease of design. Of the three static pipelines, the GasP (designed by Ivan Sutherland and Scott Fairbanks in Sun Microsystems Laboratories, USA) approach offers the highest performance, although it involves significant design effort because of its complex circuit structure and operation, as well as its stringent timing constraints, and is therefore more difficult to use with automated synthesis flows. The MOUSETRAP approach is next in performance, and has the added benefit of an entirely standard cell implementation; it is therefore well-suited for automation. Of the three dynamic pipelines, PS0 is the simplest to implement, but it offers the lowest throughput. The HC style has the highest throughput and lowest energy consumption of the three, but it requires matching of bundling delays. PCHB uses quasi-delay-insensitive (QDI) control logic and DI (delay-insensitive) coding of the data path, and hence is highly robust to variability."

In MOUSETRAP operation only active state exhibit switching activity, but in synchronous pipeline all stage have switching activity. Table 2 shows a result for power and energy consumption.

Pipeline Design	Power (mW)	Energy/item/stage (pJ)
MOUSETRAP	30.8	1.49
PS0	26.3	5.20

Table.2 Power and Energy Consumption of 10-stage FIFOs [1].

In particular, MOUSETRAP consumes 17% higher power than PS0 (designed by Williams and Horowitz in 1986-1991). This higher power consumption in MOUSETRAP is solely because of its significantly higher throughput (2.1 GHz), i.e., it performs more "work" per second than the PS0 pipeline (0.51 GHz). MOUSETRAP consumes 71% lower energy per item per stage.

V. Conclusion

In the paper it can be seen that the MOUSETRAP is a great step towards energy efficiency. The paper did show, that the MOUSETRAP could be used as a high performance system. Logic and Circuit level optimizations are showed, to improve throughput. CMOS optimization resulting in a style that is particularly well-suited for gate-level pipelining. Dual-Rail optimization removes critical inverter delays from the cycle time by implementing the control circuits in dual rail. Finally, a circuit-level optimization further speeds up critical events through a "waveform shaping" (reduce voltage swing) approach. In steady-state operation, the pipeline performance is comparable to that of wave pipelines.

References

[1] M. Singh and S. M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines, in IEEE Transactions on VLSI System archive, vol. 15, iss 6, p.684-698, June 2007.

[2] C. van Berkel, M. Josephs, and S. Nowick, "Scanning the technology: Applications of asynchronous circuits," Proc. IEEE, vol. 87, no. 2, pp. 223–233, Feb. 1999.

[3] S. M. Nowick, M. Singh: High-Performance Asynchronous Pipelines: An Overview, Design & Test of Computers, IEEE vol.28, iss: 5 , p.8-22, Oct. 2011.

[4] C. Mannakkara, T. Yoneda: Asyncronous Pipeline Controller Based on Early Acknowledgement Protocol, Application of Concurrency to System Design, p.118-127, June. 2008.

[5] K. Tereyama, A. Kurokawa, M. Imai: Scan Test of Latch-based Asynchronous Pipeline Circuits under 2-phase Handshaking Protocol, R2-3, SASIMI 2015 Proceedings.

[6] S. E. Schuster, P. W. Cook: Low-Power Synchronous to Asynchronousto Synchronous Interlocked Pipelined CMOS Circuits Operating at 3.3–4.5 GHz, p. 622-630, IEEE VOL. 38, NO. 4, April 2000.