ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Solution for Minimizing the Latency Using Sequenced Latching

T.Sandhya Rani
M.Tech Student,
Dept of ECE,
Abdulkalam Institute of Technological Sciences,
Kothagudem.

B.Bashu
Assistant Professor,
Dept of ECE,
Abdulkalam Institute of Technological Sciences,
Kothagudem.

Rajaiah Gabbeta
Professor & HOD,
Dept of ECE,
Abdulkalam Institute of Technological Sciences,
Kothagudem.

ABSTRACT:

Modern multicore systems have a large number of components operating in different clock domains and communicating through asynchronous interfaces. These interfaces use synchronizer circuits, which guard against metastability failures but introduce latency in processing the asynchronous input. We propose a speculative method that hides synchronization latency by overlapping it with computation cycles. We verify the correctness of our approach through a Micro wind. The results reveal that our approach achieves average savings of area costs and power costs compared to other speculative techniques.

Index Terms:

Duplication, latency, metastability, speculation, synchronization.

INTRODUCTION:

The crossing of data between different clock domains requires re-timing the data according to the receiver's clock and is prone to metastability failures . While it is impossible to completely prevent these failures, their probability can be reduced to an acceptable level by re-sampling the input signal through a cascade of flip-flops known as a synchronizer. This ensures that any occurrences of metastability are given enough time to resolve to valid logic states before being interpreted by other circuits. However, this also introduces latency and degrades the performance of the communication link.

Therefore, a synchronizer design presents a reliability versus performance tradeoff where the mean time between failures (MTBF) of the design is traded for the time available for synchronization (the settling time ts). The relationship between the MTBF and ts is captured by the textbook formula

$$\mathrm{MTBF} = \frac{e^{t_s/\tau}}{f_c \times f_d \times T_w}$$

where τ is the metastability regeneration time constant, fc is the clock frequency, fd is the data arrival rate, and Tw is a reference time window for the exponential relationship. For a given technology and operating conditions, the MTBF of synchronization is adjusted by the designer's choice of ts .

The latency of synchronizer chains has a detrimental impact on the performance of latency-sensitive applications and so alternative methods to reduce or eliminate this latency have been proposed. These methods can be divided into three categories as follows:

Circuit-Level Designs:

Synchronization time can be reduced by using faster (lower τ) flip-flops. The Jamb latch was an early attempt to meet the special performance requirements of synchronization. A similar improved latch was presented in featuring additional transistors that increase the regenerative loop gain when a metastable state is detected. Another improved design was presented in demonstrating lower susceptibility to supply voltage variation

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Exploiting Known Timing Relationships:

Synchronization latency can be avoided in special cases when the communicating clocks share a timing relationship. This is because it becomes possible to detect when clockdata conflicts might occur and avoid sampling the input in the vicinity of hazardous intervals. Solutions of this form exist for clocks with equal frequency but unknown skew , closely-matched (plesiosynchronous) clocks, rationally related clocks, and periodic clocks.

Using Pausible/Stretchable Clocks:

The need for synchronization can be eliminated by allowing the receiver's clock to pause for an unbounded amount of time. This is typically done by adding a mutual-exclusion element that arbitrates between the receiver's clock and handshake requests and pauses the receiver's clock until any encountered metastable states are resolved .

Existing System:

In this section, we review the existing speculative method of reducing synchronization latency.

To the best of our knowledge, only that method appear in the literature.The method (Data path Unfolding) is, in fact, a general-purpose hardware duplication solution that can be used to mitigate latency in various contexts, including synchronization.

Overview of Speculation:

Speculation is the use of either time or resource redundancy to perform potentially useful work. Modern digital systems employ speculation at different abstraction levels. For example, memory management speculatively populates cache hierarchies with prefetched data to reduce the impact of slow memory access on processing speed .

Data path Unfolding:

Performing speculative computations in pipelined systems is particularly easy because restoring the state of a pipeline in the case of misspeculation is trivial. For example, when a pipelined processor mispredicts a branch, invalid instructions in the fetch and decode stages can be discarded. However, the same cannot be said about nonpipelined systems.

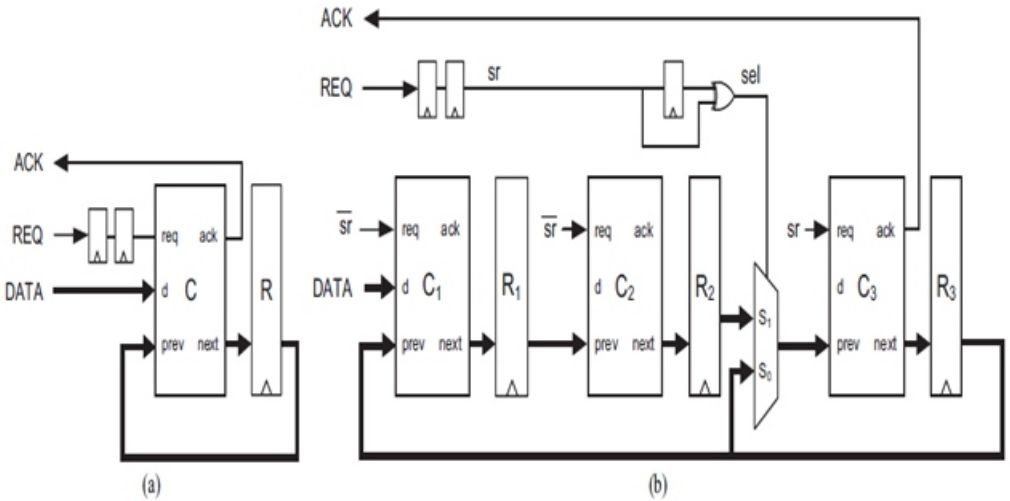


Fig 1 Hiding synchrozination latency using data

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

path unfolding:

a)Asynchronous receiver
b)Equivalent unfolded cyclic pipeline

speculative computations cannot be "reversed" in a similarly straightforward manner. This is because non pipelined systems have loop dependencies1 (i.e., feedback paths), such as the one represented by the expression x x + 1. The existence of these dependencies means that miss speculated operations may cause irreversible state corruption. Nevertheless, arbitrary designs can benefit from speculation by first being converted into functionally-equivalent pipelines. This can be done by unfolding these designs (e.g., converting x x + 1 into x1 x2 + 1). Unfolded pipelines produce the same output as the original design after a number of cycles equal to the pipeline . .depth. Unfolding is widely used by compilers and schedulers to increase execution throughput. The technique is also equivalently-capable of resolving loop dependencies in hardware implementations; it is extensively used in digital signal processors and has been proposed for general purpose synthesis .

c) depth.Unfolding is widely used by compilers and schedulers to increase execution throughput. The technique is also equivalently-capable of resolving loop dependencies in hardware implementations; it is extensively used in digital signal processors and has been proposed for general purpose synthesis. Consider the asynchronous receiver shown in Fig. 1(a). The receiver is depicted as generic Moore machine consisting of the combinational block C and the state register R.

The transfer of asynchronous data is controlled by the handshake signals REQ and ACK. We will assume, for illustration purposes, that this receiver uses a four-phase handshake protocol. Therefore, the sender makes data available on the bus and asserts REQ. The controller then latches the data, performs few data-dependents computations, and subsequently asserts ACK to acknowledge the completion of data consumption. Afterwards, the sender de-asserts REQ and the receive de-asserts ACK. To maintain reliability, two flip-flops are used to synchronize REQ.This delays the latching of data and the onset of subsequent computations by two cycles. Nevertheless, this latency can be hidden by using two additional data path instances to speculatively compute the first two data path states following a transition of REQ. An arrangement which accomplishes this is shown in Fig. 1(b). Here, the receiver data path has been unfolded into a three-stage cyclic pipeline. The third stage represents the original (safe) copy of the data path while the first two perform speculative computations assuming the data bus holds valid data.

At each cycle, the state stored in R2 represents what the state in R3 would have been if REQ transitioned two cycles earlier. When an actual REQ transition appears at the synchronizer output, the speculative state stored in R2 is used by C3 to compute the third state following data arrival. Subsequently, the original data path (R3, C3) resumes computations and acknowledges the sender upon completion.2 This scheme requires an isochronic handshake protocol:

REQ must arrive after data by a sufficiently-large delay. This ensures that the speculative state in R2 is based on valid data every time a transition of REQ appears at the synchronizer output. Note that invalid (or metastable) data path states may still be latched by R1 and passed to R2. However, isochrony ensures these states always precede the appearance of a REQ transition at the synchronizer output.

Proposed System:

In this section, we describe a novel technique to latch data reliably during synchronization cycles. In short, we use the synchronizer as a state machine to sequence a series of latching operations. The synchronizer is constrained such that its state does not change when a latching operation fails. Therefore, any failed latching attempts are automatically retried in the subsequent cycles.

We call this method sequenced latching and demonstrate, in that it can be exploited to perform speculative computations using only two datapath instances. The following analysis uses assumptions about the behavior of metastable flip-flops and so this is discussed first.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

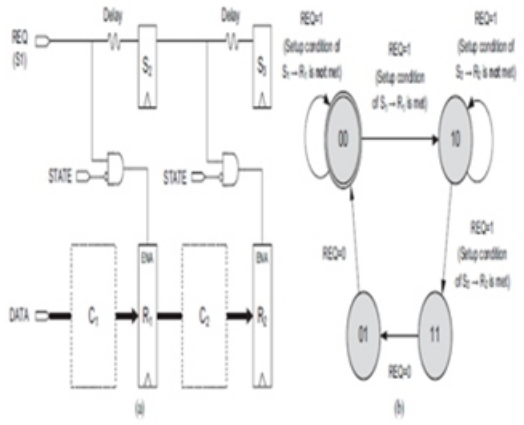


Fig 2. Sequenced latching: pipeline

a) pipeline with a synchronizer act as control logic
b) synchronizer state diagram

Assumptions About Metastable Behavior

In our analysis, we will assume, in accordance with, that a metastable state can cause an increase in the clock-to-q delay of a flip-flop but does not change the monotonic nature or the rise/fall time of its output. We have simulated several flip-flop designs from a commercial 90-nm standard cell library and found their behavior consistent with that reported in .

Isochronicity

Similar to the case of data path unfolding, our technique is based on isochronic handshakes. This means that the data and request signals travel separately and that data availability is indicated to the receiver by a transition of the request signal that is guaranteed to arrive a sufficient time later.

Technique Overview

We illustrate our technique by referring to Fig. pipeline with a synchronizer acting as control logic in sequenced latching. In the depicted design, two flip-flops (S2 and S3) synchronize an asynchronous handshake request (REQ) and simultaneously act as a state machine to control the propagation of the asynchronous data item through a pipeline.

IMPLEMENTATION:

We have proved that the state of a constrained synchronizer can be used to sequence a series of latching operations correctly. This is done by making the satisfaction of the setup conditions of all the paths from the synchronizer flip-flops to the data registers a necessary condition for a change in the synchronizer state. Satisfying Inequality 3 guarantees this behavior regardless of the combinational logic used to decode the synchronizer state (two AND gates in Fig pipeline with a synchronizer acting as control logic in sequenced latching). In this section, we demonstrate how to exploit this behavior to perform speculative computations using only two datapath instances. In short, we connect the two instances in a cyclic pipeline and use sequenced latching to enable them alternately. When one state register fails to latch the next state.

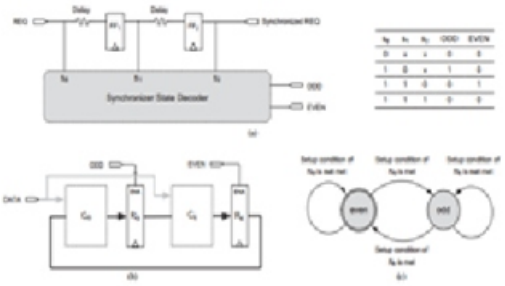


Fig 3.Sequenced latching: Datapathpath

a)Synchronizer state decoder b)Cyclic pipeline of data path instances.c) State diagram
operating Principle
The basic application of sequenced latching to a non-pipelined datapath is demonstrated in above figure. Two datapath instances are connected in a cyclic pipeline [Fig. cyclic pipeline of datapath instances] and enabled alternately using two signals, ODD and EVEN, which are combinationally derived from the synchronizer nodes and asserted during the odd and even cycles of synchronization, respectively, Fig. Synchronizer state decoder. For a positive-edge four-phase protocol (assuming n is even)

$$\text{ODD} = \sum_{i \in (0,2,\ldots,n)} (s_i \wedge \neg s_{i+1})$$

$$\text{EVEN} = \sum_{i \in (1,3,\ldots,n-1)} (s_i \wedge \neg s_{i+1}).$$

Similarly, for a two-phase protocol (n is even)

$$\text{ODD} = \sum_{i \in (0,2,\ldots,n)} (s_i \oplus s_{i+1})$$

$$\text{EVEN} = \sum_{i \in (1,3,\ldots,n-1)} (s_i \oplus s_{i+1}).$$

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Proposed Implementation

Fig. Asynchronous receiver shows the block diagram of the proposed implementation. Here, the handshake controller sits outside the datapath and communicates with it using the signals VALID and COMPLETE. VALID is asserted by the controller at the end of synchronization while COMPLETE is asserted by the datapath to mark the end of computations.

During synchronization cycles, the sequencing logic block undertakes the generation of ODD and EVEN to complete a number of datapath state transitions equal to the number of synchronization cycles. Delay elements are added between the synchronizer flip-flops to satisfy criteria gi (Inequality 3).

If the datapath computation cycles (m) are more than synchronization cycles (n) then the alternating behavior of ODD and EVEN needs to be maintained after synchronization. This can be achieved by adding a toggle flipflop to the sequencing logic as shown in fig 3(b).

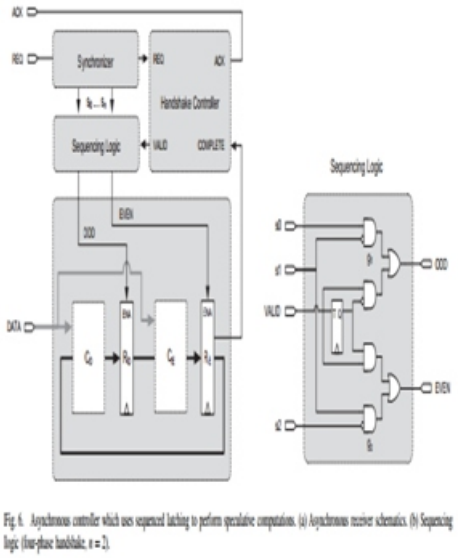


Fig 4. Asynchronous controller which uses sequenced latching to perform speculative computations

a)Asynchronous receiver schematics.
b)Sequencing logic

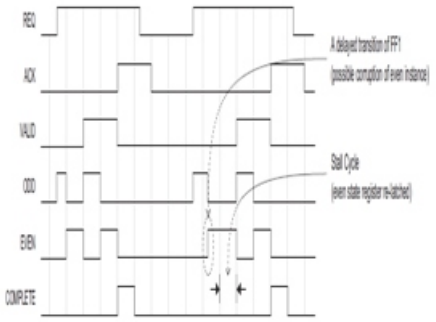


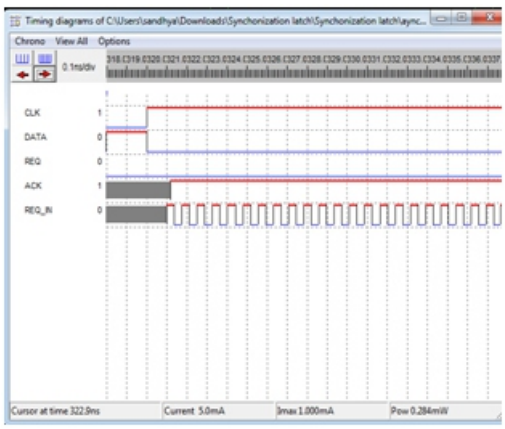
Fig. 5. Signal waveforms illustrating the behavior of the system when a synchronizer flip-flop becomes metastable (four-phase handshake protocol).

Example of Behavior

In this section, we will explain in detail how the implementation in Fig. 4 behaves when one of the synchronizer flip-flops becomes metastable. We will refer to Fig. 5 which shows the signal waveforms for two handshakes; a typical case metastability-free handshake and another where a metastable state is encountered. In the first handshake, ODD and EVEN begin toggling as soon as REQ is high and COMPLETE goes high four cycles later to signal the handshake controller to pull ACK high and end the handshake. This is the typical-case behavior of the sequencing logic in Fig. 2(b) when the synchronizer flip-flops' outputs (s0, s1, and s2) transition with nominal clock-to-q delays. In this example, the data path asserts COMPLETE after four computation cycles. In the second handshake, REQ transitions to logic high and RO latches a data path state simultaneously. However, the arrival of REQ causes the first synchronizer flip-flop (FF1) to become metastable.

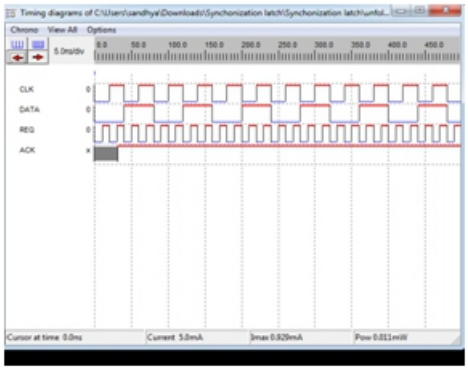
Results

Asynchronous receiver:

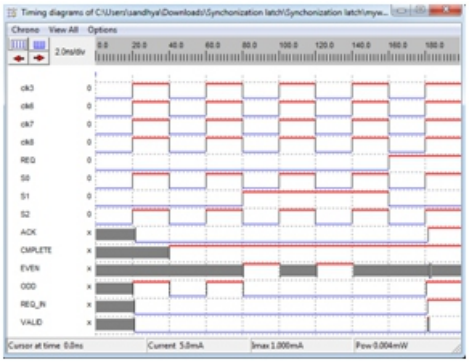


ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Unfolded pipeline:



Sequenced latching:



CONCLUSION:

We presented a method of performing speculative computations during synchronization cycles and hence prevented synchronization time from incurring latency. Our method relies on using the synchronizer state to sequence the latching of data during synchronization cycles and automatically re-latch any corrupt data. We verified that our approach works in practice by implementing it on an Microwind. Our method outperforms existing speculative methods in several aspects; it provides zero latency, its area overhead is smaller (and does not increase with the number of synchronization cycles) and its power overhead is negligible.

REFERENCES:

[1] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," IEEE Trans. Comput., vol. 22, no. 4, pp. 421–422, Apr. 1973.

[2] I. W. Jones, S. Yang, and M. Greenstreet, "Synchronizer behavior and analysis," in Proc. 15th IEEE Symp. Asynchron. Circuits Syst., May 2009, pp. 117–126.

[3] R. Ginosar, "Metastability and synchronizers: A tutorial," IEEE Design Test Comput., vol. 28, no. 5, pp. 23–35, Sep.–Oct. 2011.

[4] D. Kinniment, A. Bystrov, and A. Yakovlev, "Synchronization circuit performance," IEEE J. Solid-State Circuits, vol. 37, no. 2, pp. 202–209, Feb. 2002.

[5] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev, "A robust synchronizer," in Proc. IEEE Comput. Soc. Annu. Symp. Emerg. VLSI Technol. Archit., Mar. 2006, pp. 442–443.

[6] S. Yang, I. Jones, and M. Greenstreet, "Synchronizer performance in deep sub-micron technology," in Proc. 17th IEEE Int. Symp. Asynchron. Circuits Syst., Apr. 2011, pp. 33–42.

AUTHORS:

T.Sandhya Rani has received B.Tech Degree in E.C.E froms Abdul Kalam Institute of Technological Sciences(JNTUH) in 2013.Presently Studying M.Tech in Embedded Systems and VLSI System Design at AKITS.

B.Bashu has received his B.tech Degree in Electronics and Communication Engineering from aizza Engineering College (JNTUH) in 2006 and M.Tech in communicationsystems from aurora Engineering College (JNTUH) in 2010.Presently working as an Assistant Proffesor in Abdul Kalam Institute of Technological Sciences

G.Rajaiaha has received his B.tech Degree in Electronics and Instrumentation Engineering from Kakatiya Universit in 1997 and M.Tech in Instrumentation and Control System Design from JNTU Kakinad in 2005. Presently working as a Proffesor and HOD in Abdul Kalam Institute of Technological Sciences. He has contributed more than 20 reviewed publications in Journals.