Quantitative Analysis of State-of-the-Art Synchronizers: Clock Domain Crossing Perspective

N. Sharif, N. Ramzan, F. K. Lodhi and O. Hasan
School of Electrical Engineering and Computer Science
National University of Sciences and Technology (NUST)
Islamabad, Pakistan
{naeha.sharif, nadra.ramzan, faiq.khalid, osman.hasan}
@seecs.nust.edu.pk

S. R. Hasan
Department of Electrical Engineering
École Polytechnique de Montréal
Montréal, Canada
hasan@grm.polymtl.ca

Abstract- **Reliable transferring of data from one clock domain to another requires synchronization. Therefore, synchronizers play an important role in clock domain crossing (CDC). But despite their wide applications, there is no standard quantitative metric available to analyze various synchronizer configurations on common grounds. To overcome this limitation, this paper presents a comparison of three basic and widely used synchronizers. Latency and power consumption metric are measured for level, edge-detecting and pulse generating synchronizers. Furthermore, effects of these synchronizers are studied when applied to some commonly used asynchronous handshaking protocols under 90nm CMOS technology.**

Keywords-Synchronizers, Handshaking Protocol, System on chip (SOC)

I. Introduction

Recent microprocessors and graphical processing units contain several clock domains within a chip [1, 2]. Modules in different clock domains need to communicate with each other and hence require crossing the clock domain, a phenomenon that is commonly called as clock domain crossing (CDC).

When a signal crosses the clock domains it is treated asynchronously in the receiving domain. Such a signal is called mutually asynchronous signal and for a system to work properly this signal requires safe synchronization. In modern deep sub-micron (DSM) technologies; factors like clock skew and jitter due to process, voltage and temperature (PVT) variations [3] adversely affect the timing behavior of signals. In the case of high frequency applications, if not treated properly, these timing uncertainties in mutually asynchronous signals can cause malfunctioning of the system. Therefore, recently extensive research is underway for reliable design protocols to solve this problem of synchronization.

Several synchronizer solutions have been proposed to solve these issues. Sarwary et al. [4] Presented an enable based synchronizer. The protocol assumes the data to be stable when enable is asserted. The basic synchronizer in this protocol is the traditional two flop synchronizer. This approach addresses issues such as metastability convergence. Jabulani et al. [5] proposed a synchronization scheme for network-on-chip (NOC) systems. The scheme consists of FIFO designs [6] that interface system on chip modules running at different frequencies. The authors have compared their protocol to the one proposed by Chelcea and Nowick [7]. They showed that their architecture has low latency and power consumption. Ginosar and Semiat in [8] have given an analysis of a standard two flop synchronizer and some pleisochronous synchronizers. Each of these CDC schemes is implemented and analyzed using different design constraints and technologies. Thus, identifying the most efficient synchronizer for a given system is a very challenging task, if not impossible. Usually the main tools available to the designer for selecting a synchronizer are intuition and prior experience. Therefore, there is a dire need to quantitatively analyze the context and develop the parameters of interests so that designers can evaluate the available CDC methodologies on common ground and identify the most efficient contextual based option. In [9] Ginosar also reflected upon the same idea. However in general these studies focus on the design of synchronizer and the contextual based implementation details are left unexplored.

In order to overcome the above mentioned synchronizer selection problems, this paper presents a thorough analysis of three of the widely used synchronizers. The three configurations include level synchronizer, edge-detecting synchronizer and pulse generating synchronizer. Initially, in this paper some parameters of interest for designers are identified, such as absolute latency, clock cycle latency, and relative frequency of the communicating modules and overall power consumption of the system for a reasonable pre-established criterion. The comparison is made using same parameters and technology for all synchronizers. The design is done using the 90nm CMOS technology in CADENCE [10], which is an efficient IC designing tool. A comprehensive analysis is performed on the implementation of these synchronizers for full handshaking and partial handshaking asynchronous protocols. To the best of our knowledge, this is a premier work in contextual based synchronizer suitability analysis.

The rest of the paper is organized as follows: Section II explains the three synchronizers that are analyzed. Later, Section III discusses the hardware implementation of the analyzed synchronizers. In section IV, experimental results performed using 90nm processing technology are presented followed by some discussions in Section V. Finally, section VI concludes the paper.

II. STATE OF ART SYNCHRONIZERS

Synchronization is a process of imposing or identifying an ordering of event on the signal lines [11]. The purpose of synchronization is to prevent metastability in logic flow. Traditional synchronizers comprise of flip-flops that are combined together without any combinational logic between them. It is also worth mentioning that for proper synchronization the signal from flip-flop in one clock domain should enter the flip-flop in the second clock domain without passing through a combinational logic. Fulfillment of this condition is necessary because the first stage of the synchronizer is sensitive to glitches and can force it to give an incorrect output to rest of the design.

The synchronizers can be broadly classified into three main categories; level, edge-detecting and pulse. Other designs also exit but these synchronizers fulfill most of the requirements, designer's experience. This paper analyzes and compares these three types of synchronizers.

A. Level Synchronizer

The level synchronizer [12] consists of two flip-flops connected back to back as shown in Fig. 1. It is also known as two-flop synchronizer and is the basic component of all synchronizers. The mutually asynchronous signal may fail to meet the setup or hold time requirement of the first flip-flop in the receiver domain causing it to fall into a metastable state. If this synchronizer is made up of only one flip flop instead of two then due to the occurrence of metastable output the receiving domain would be vulnerable to the downstream logic from the metastable state of the first flip flop. In contrary, the two flip flop configuration allows the synchronizer one clock cycle duration to resolve the metastability

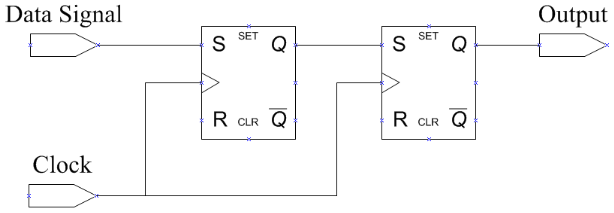


Figure 1. Block Diagram of Level Synchronizer

In a level synchronizer, signal crossing clock domain stays high and low for more than two clock cycles in the new domain. It is requisite for the circuit that signal changes its state from valid to invalid before becoming valid again. This condition is necessary because each time the signal changes its state from invalid to valid state the receiving circuit considers it as an event. It does not depend on how long the signal remains valid.

B. Edge-detecting Synchronizer

The second type of synchronizer is the edge-detecting synchronizer [12] (Fig. 2). It consists of a level synchronizer and an additional flip-flop at its output. The inverted output of this added flip flop and the output of the level synchronizer are ANDed as shown in Fig. 2. The purpose of this synchronizer is to detect the rising edge of the input and generate a clock wide active high pulse at the output. The circuit can be modified to detect the falling edge of the input. It can be done by switching the inverter on the AND gate. If a NAND gate is used instead of an AND gate then the circuit generates an active low pulse.

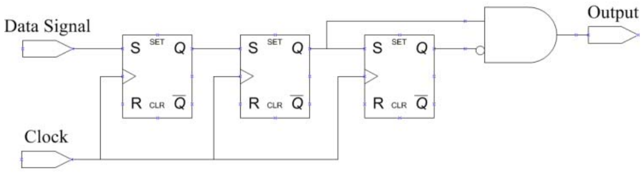


Figure 2. Block Diagram of Edge-detecting Synchronizer

C. Pulse Generating Synchronizer

The third type of synchronizer discussed here is the pulse synchronizer [12] (Fig. 3). The input of a pulse synchronizer is a single clock wide pulse, which triggers the toggle circuit in the originating domain. The toggle circuit changes its output state whenever it receives an input signal. The output of the toggle circuit passes through the level synchronizer and acts as an input to the XOR gate. The second input of the XOR gate is a one clock cycle delayed version of the output of the level synchronizer.

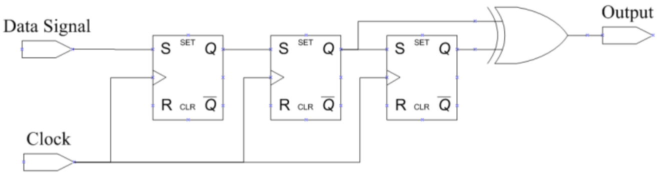


Figure 3. Block Diagram of Pulse Generating Synchronizer

III. HARDWARE IMPLEMENTATION USING HANDSHAKING PROTOCOL

This section explains the hardware implementation of the above mentioned synchronizers. Fig. 4 shows a general block diagram of two modules that are "Receiver module" and "Sender Module". The two modules exist in different clock domains and communicate with each other through synchronizers.

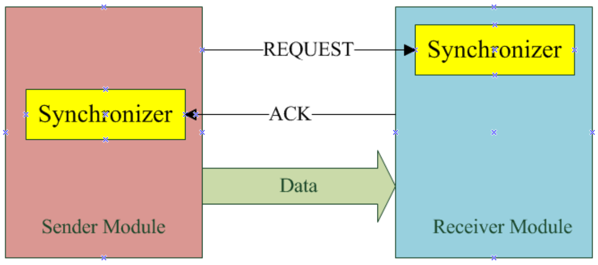


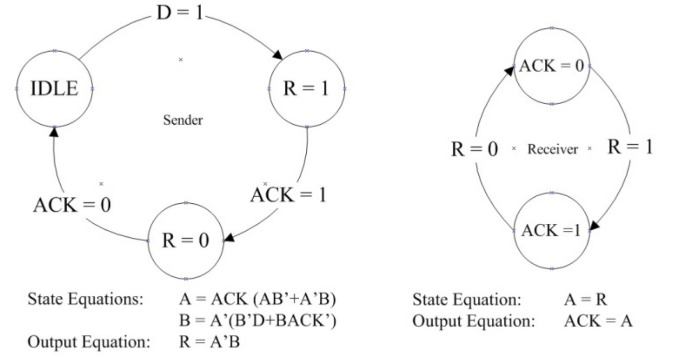
Figure 4. Synchroniztion with Handshaking Protocol

The design shown in Fig. 4 consists of two phases; synchronization phase and data transmission phase. A state machine which is part of the design (not shown in the figure) decides the phase. In the synchronization phase the synchronizers with the help of a handshaking protocol, synchronize the signals crossing the clock domains. Handshaking allows circuits to communicate effectively when their response time is unpredictable. On the completion of the synchronization process the data is transmitted from one domain to the other during the data transmission phase.

We used three versions of handshaking protocols; a full handshake and two partial handshake techniques.

A. Full Handshake

In the full hand shake protocol, modules wait for each other's acknowledgement before inserting or removing their respective signals. The state diagram for respective handshake is shown in Fig. 5.



State Equations: A = ACK (AB'+A'B)
B = A'(B'D+BACK')
Output Equation: R = A'B

State Equation: A = R
Output Equation: ACK = A

Figure 5. State diagram of Full Handshake Protocol

The sender and receiver module communicate with each other using R (request) and ACK (acknowledgement) signals. At first the sender module asserts a request which is detected by the receiver module. After verifying that the signal is valid the receiver module asserts an acknowledgement signal. The sender module performing similar verification then drops its request. It does not assert a new request until the receiver module drops it acknowledgement signal.

This type of handshake uses level synchronizers as depicted in (Fig. 6). It is used when the receiver module needs to inform the sender module that the request is being processed. A requirement of this protocol is that the sender module does not send a request signal unless it gets an invalid acknowledgement signal.

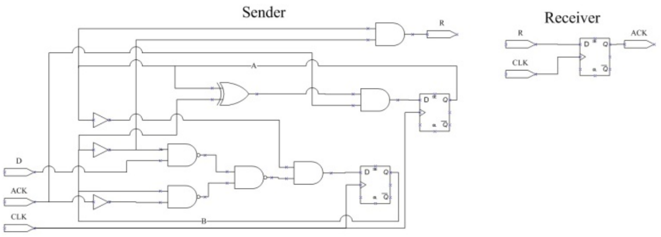
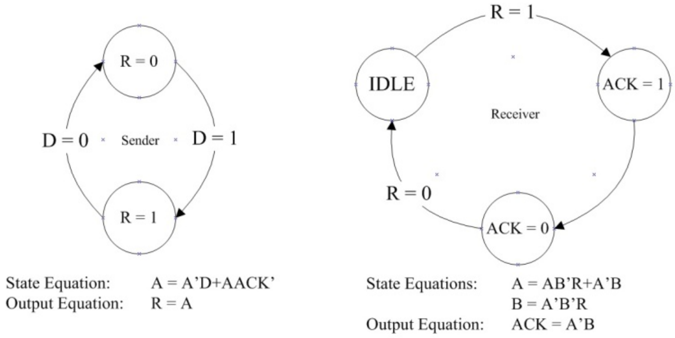


Figure 6. Hardware Implementation of Full Handshake Protocol

B. Partial Handshake I

The second type of protocol is the partial handshake. In this type of handshake, the sender and receiver modules do not wait for each other before they drop their respective signals and move on with the handshaking process (Fig. 7).

The sender module asserts its request signal as an active high level, and the receiver module acknowledges it with a single clock wide pulse. In this case, the receiver is not concerned when the sender will drop a request. On the other hand, to make the protocol work properly, the sender needs to drop the request signal for at least one clock cycle as otherwise the receiver cannot differentiate between a new and a preceding request.



State Equation: A = A'D+AACK'
Output Equation: R = A

State Equations: A = AB'R+A'B
B = A'B'R
Output Equation: ACK = A'B

Figure 7. State diagram of Partial Handshake I Protocol

For this type of handshake a pulse synchronizer is used at the sender's end while a level synchronizer is used at the receiver's end (Fig. 8).

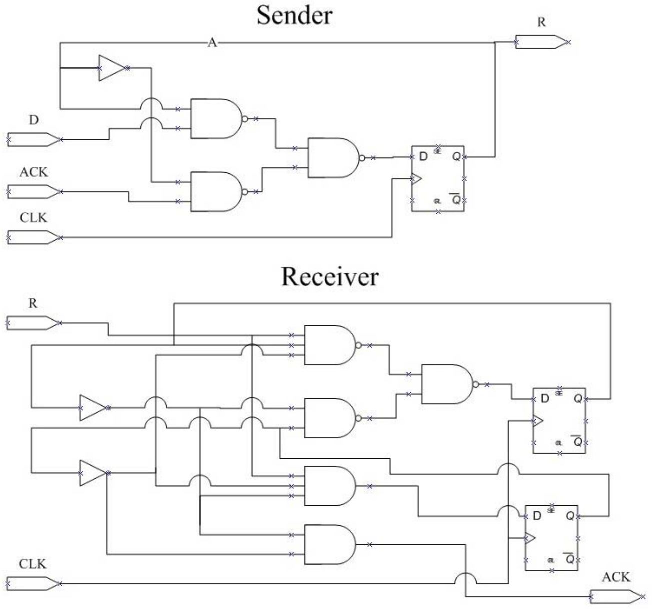


Figure 8. Hardware Implementation of Partial Handshake I Protocol

C. Partial Handshake II

A second type of partial handshake scheme can also be used. In this protocol the sender asserts its request signal consisting of a single clock wide pulse, which is acknowledged by the receiver with a single clock wide pulse (Fig. 9).

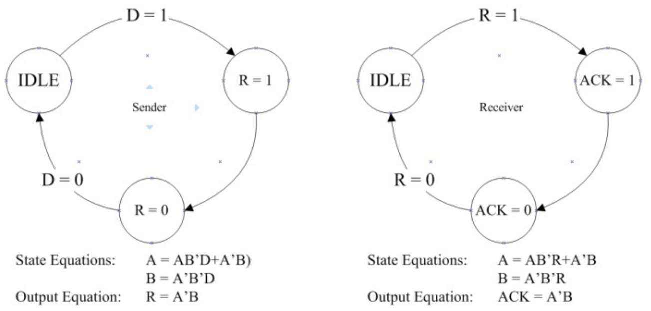


Figure 9. State diagram of Partial Handshake II Protocol

This type of handshake uses pulse synchronizer both at the sender and receiver ends (Fig. 10). But if one module has a clock twice as faster than the other then it can use an edge detecting synchronizer instead.

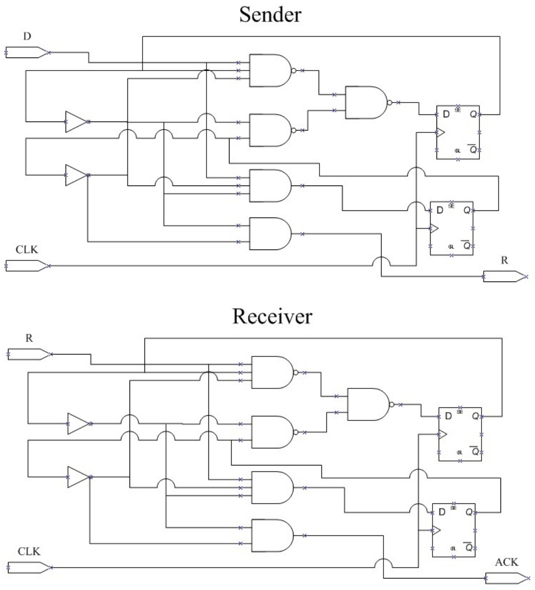


Figure 10. Hardware Implementation of Partial Handshake II Protocol

IV. Experimental Results

State machine implementation of the three synchronizers; level, edge-detecting and pulse using different handshaking protocols have been shown in Figures 5, 7 and 9, respectively. This section summarizes the results obtained after analyzing and comparing these synchronizers.

All the electrical simulations are performed using 90nm CMOS IHP technology. For gate level implementation, pass transistor logic has been used. The flip-flop in each synchronizer is a true single phase clock (TSPC) DFF. To eliminate ideal characteristics of the pulse sources, buffers are connected at the inputs and a capacitive load of 900pF has been added at the output to mimic as standard driven load. The comparison among the synchronizers has been done based on the latency and power consumption.

A. Latency

Latency is the measure of time delay in a system. In case of synchronizers, it refers to the time taken by a synchronizer to synchronize the signal and provide a valid output [15]. In our analysis the latency of each synchronizer is calculated as the time difference between the point at which the synchronizer samples the data and the point where the sampled data appears at its output. It is represented as the duration between points A and B, in Fig. 11. This figure shows the simulation results for a level synchronizer. The synchronizer clock is operating at a frequency of 1.5GHz whilst the maximum input data rate (data-in) of once every three clock cycle is assumed. Absolute latency of about approximately 711ps is obtained between point A and B in Fig. 11, which is slightly more than one clock cycle.

Similarly, we measured the latency for the edge-detecting and pulse generating synchronizers, keeping with the same operating frequency and input data rate. It should be noted that the data is operating at a much slower frequency. The results are summarized in Table 1.

For further analysis, we implemented the three widely used handshaking protocols, mentioned in section III, using above synchronizers and FSM's. The synchronizers are chosen depending on the type of handshaking protocol.

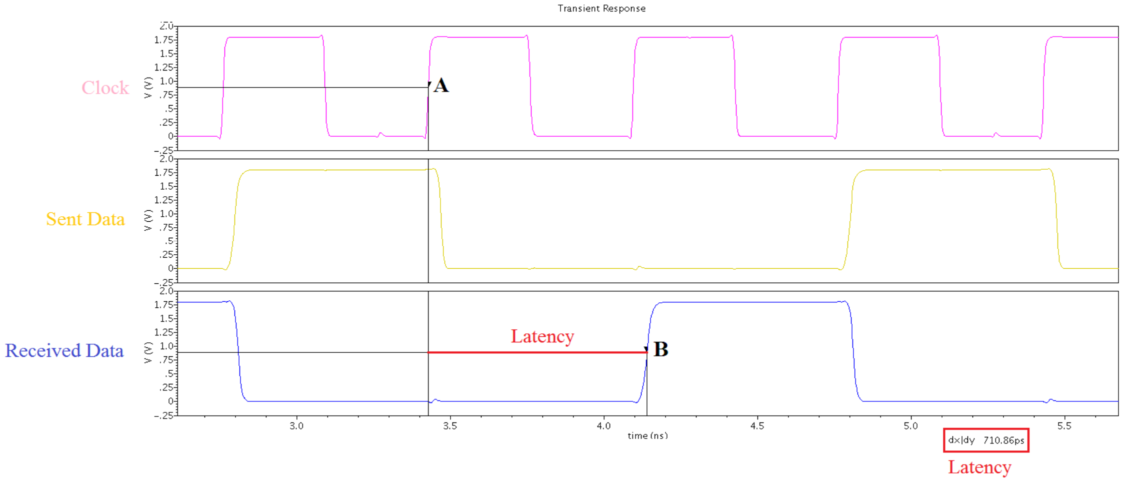


Figure 11. Simulation Results for latency calculation of a level synchronizer

Fig. 12 shows the simulation result for the full handshake protocol. Both receiver and sender modules are operating at the same frequency of 1.5GHz. The active high data signal indicates that the data is present at the sender module. Using the simulation waveform, the latency is calculated as the time difference between the Request sent and ACK received by the sender. Similar analysis was done for the other two protocols as well and the results are given in Table 2.

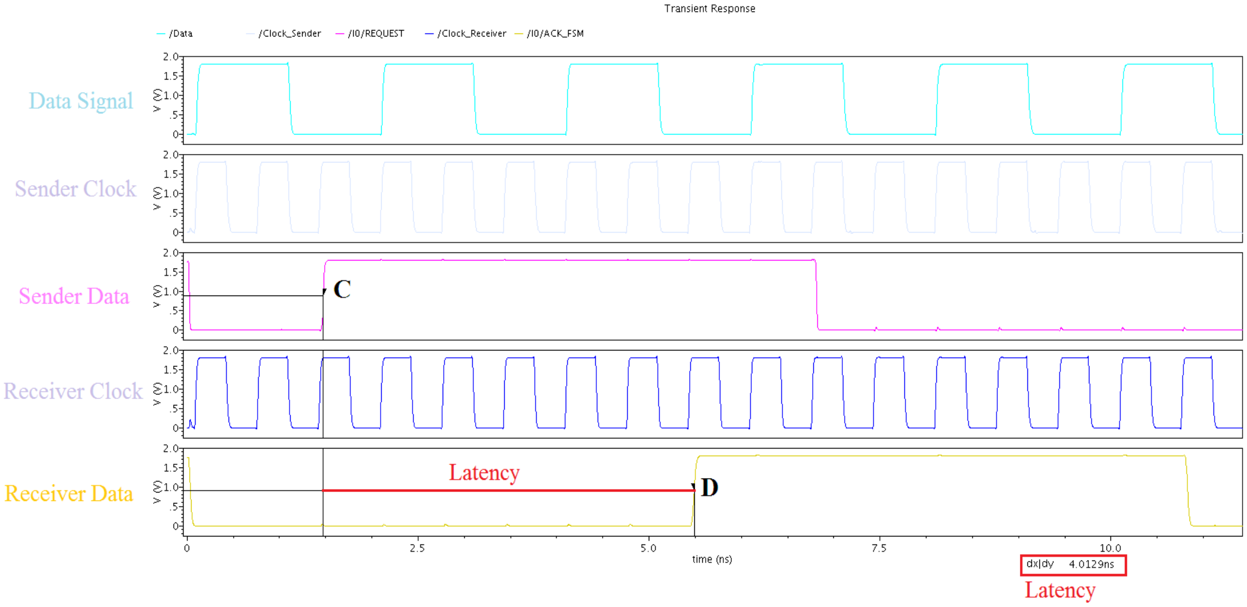


Figure 12. Simulation Results for latency calculation of a Full Hanshake Protocol

TABLE I. LATENCY OF DIFFERENT TYPES SYNCHRONIZERS WITHOUT PROTOCOL IMPLEMENTATION

Latency of the Synchronizers without protocol implementation (approx)	
Synchronizers	***Latency*** ***ps***
Level	711
Edge-detecting	738
Pulse	750

TABLE II. LATENCY OF DIFFERENT TYPES SYNCHRONIZERS WITH PROTOCOL IMPLEMENTATION

Latency of the Synchronizers with Protocol implementation (approx)	
Protocols	***Latency*** ps
Full Handshake	4012
Partial Handshake I	3380
Partial Handshake II	3340

B. Power

Increase in speed and complexity of circuits available today implies a significant increase in power consumption for very large scale integrated circuits. Since power consumption plays an important role in evaluating the efficiency of a system, designers have always looked for different design protocols to reduce power consumption.

To measure the power, we connected a dummy test source in series with the power supply of the circuit. Using the calculator available in the CADENCE tool, we calculated the DC current and voltage at adjacent node and branch of the test, respectively, and then evaluated the power consumption. This way we made sure that the current supplied to the circuit is considered in total. The values of the power measurement for the three synchronizers and handshaking protocols are shown in Tables III and IV, respectively.

TABLE III. POWER OF DIFFERENT TYPES SYNCHRONIZERS WITHOUT PROTOCOL IMPLEMENTATION

Power Consumption of the Synchronizers without Protocol implementation (approx)	
Synchronizers	***Power(mW)***
Level	1.09
Edge-detecting	1.12
Pulse	1.12

TABLE IV. POWER OF DIFFERENT TYPES SYNCHRONIZERS WITH PROTOCOL IMPLEMENTATION

Power Consumption of the Synchronizers with Protocol implementation (approx)	
Protocols	***Power(mW)***
Full Handshake	2.85
Partial Handshake I	2.59
Partial Handshake II	2.72

V. DISCUSSIONS

In this paper, a comparison of the three widely used synchronizers i.e. level, edge detecting, pulse synchronizer has been provided in terms of latency and power consumption. These synchronizers were also incorporated in full handshake, partial handshake I and partial handshake II protocols to compare the working and performance based on the quantitative metrics. The results in Tables I and III clearly show that the level synchronizer offers the lowest latency and power consumption as compared to the edge detecting and pulse synchronizers because its architecture is free from any combinational logic. But further analysis of its architecture and functionality reveals that it works properly only if the input data has a frequency equal or slower than the frequency of synchronizer or the data moves from a faster to slower clock domain.

The latency of the edge detecting synchronizer lies midway between that of level and pulse synchronizer, whereas the power consumption of the edge detecting and pulse synchronizer is equivalent (as shown in Table I and III). The requirement for correct operation of an edge-detecting synchronizer is that the input pulse width should be greater than the clock period of the synchronizer in addition to the hold time required by the first flip-flop of the synchronizer.

Although the pulse synchronizer falls at the bottom in terms of the latency and power consumption metric but it solves the problem of the edge-detecting synchronizer, i.e., it gives a valid output for a data signal moving from a slower to faster domain. But if the input pulses are too close to each other than the output pulse has a width greater than one clock cycle. The situation worsens if the input pulse has a clock period twice the synchronizer period. In this case, some logic 1's in the data may be missed. This synchronizer works better for transferring data from faster to slower domain.

As for the handshaking protocols, the full handshake protocol is robust because the communicating modules can know each other's states by looking at the request and acknowledgement signals. The only drawback is that about 6-7 clock cycles are wasted to complete the sequence. In case of the partial handshake protocols, although 4-5 clock cycles are used but the communicating modules need to save states to indicate a pending request.

From the latency point of view, partial handshaking protocol II gives the best results because it has the swiftest FSM in terms of number of clock cycles and the critical path of its architecture is the fastest as compared to that of the full handshake and partial handshake I protocol (Figure 10). Therefore, from these results we conclude that for the power consumption the partial handshaking protocol I is the most optimum choice.

It can be inferred that the level synchronizer has the least power consumption and latency because it does not have combinational logic gates. As far as the handshaking protocols are concerned, the partial handshaking protocols seem to be optimum. The reason is that these protocols involve less number of events resulting into lower latency and less number of components in their architecture. The pros and cons of the level, edge detecting and pulse synchronizers mentioned above can be a useful tool for the designers to select the best suited synchronizer amongst the three for a specific application in addition to their intuition and prior experience.

VI. CONCLUSIONS

This paper presents a detailed analysis of three synchronizers and their usage to implement asynchronous handshaking protocols. Using CADENCE simulations, we evaluated the performance of each synchronizer by calculating its latency and power consumption.

We have provided a comprehensive performance analysis of the synchronizers, i.e., level, edge-detecting and pulse, based on two factors, namely latency and power. Level Synchronizer with a latency of approximately 711ps and power consumption of 1.09mW comes out as the best option among the three synchronizers while operating at frequency of 1.5GHz and a data rate of 500MHz. As for the handshaking protocol implementation, the smallest latency is achieved by partial handshake II, which is approximately 4012ps, while partial handshake I has the least power consumption of 2.59mW. The outcome of the results may be used in the selection of appropriate synchronizers which fulfill the requirements of the design. Each of these synchronizers has its own advantages and disadvantages. It is demonstrated that in order to leverage upon their strengths they have to be used in appropriate scenarios.

REFERENCES

[1] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, “Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling” in High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium, pp. 29 – 40.

[2] E. Rotem, A. Mendelson, R. Ginosar and U. Weiser “Multiple Clock and Voltage Domains for Chip Multi Processors” in Microarchitecture, 2009. 2nd Annual IEEE/ACM International Symposium, pp. 459 – 468.

[3] Q. K. Zhu, “High-speed clock network design” Kluwer Academic Publisher, Botson, 2003.

[4] S. Sarwary and S. Verma, “Critical clock-domain crossing bugs” in Atrenta Inc - EDN, April 2, 2008

[5] J. Nyathi, S. Sarkar, and P. Pande “Multiple Clock domain Synchroniztion for Network on Chip” in IEEE International SOC Conference, 2007, pp. 291 – 294.

[6] J. Díaz, D. Koukopoulos, S. Nikoletseas, M. Serna, P. Spirakis and Dimitrios M. Thilikos, “Stability and non-stability of the FIFO protocol”, Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures.

[7] T. Chelcea and S. M. Nowick, “Robust Interfaces for Mixed-Timing Systems,” IEEE Transactions on Very Large Scale Integration Systems, Vol. 12, No. 8, Aug. 2004, pp. 857-873.

[8] Y. Semiat, R. Ginosar, Timing measurements of synchronization circuits, ASYNC (2003) 68–77.

[9] R Ginosar, “Fourteen ways to fool your synchronizer in Asynchronous” in Circuits and Systems, 2003. Proceedings. Ninth International Symposium, pp. 89-96

[10] Cadence design system, Inc.“Cadence Virtuoso Custom Design Platform”

[11] W.J. Dally and J.W. Poulton, "Digital Systems Engineering", Cambridge University Press, 1998.

[12] N.H.E. Weste and D. Harris “Cmos Vlsi Design: A Circuits And Systems Perspective”