A Predictive Synchronizer for Periodic Clock Domains

Uri Frank and Ran Ginosar

VLSI Systems Research Center,
Technion—Israel Institute of Technology, Haifa 32000, Israel
[ran@ee.technion.ac.il]

Abstract. An adaptive predictive clock synchronizer is presented. The synchronizer takes advantage of the periodic nature of clocks in order to predict potential conflicts in advance, and to conditionally employ an input sampling delay to avoid such conflicts. The result is conflict-free synchronization with minimal latency. The adaptive predictive synchronizer adjusts automatically to a wide range of clock frequencies, regardless of whether the transmitter is faster or slower than the receiver. The synchronizer also avoids sampling duplicate data or missing any input.

1 Introduction

Large systems on chip (SoC) typically contain multiple clock domains. Inter-domain communications require data synchronization, which must avoid metastability while typically facilitating low latency, high bandwidth, and low power safe transfer. The synchronizer must also prevent missing any data or reading the same data more than once.

Communicating clock domains can be classified according to the relative phase and frequency of their respective clocks [1][7][14]. *Heterochronous* or *periodic* domains operate at nominally different frequencies, *plesiochronous* domains have very similar clock frequencies, *multi-synchronous* domains have the same clock frequency but a slowly drifting relative phase, and *mesochronous* domains have exactly the same frequency.

The simplest solution for inter-domain data transfer is the two-flip-flop synchronizer [1][12][13]. The main problem with that synchronizer is its long latency: Typically, a complete transfer incurs waiting about one to two clock cycles at each end. Although it is a very robust solution, it is sometimes misused or even abused in an attempt to reduce its latency [9].

Another commonly used synchronizer is based on dual-clock FIFO [1][8]. In certain situations, especially when a complete data packet of a pre-defined size must be transferred, this may be an optimal solution. Another advantage is that synchronization is safely contained inside the FIFO, relieving designers of the communicating domains of this delicate design task. The main drawback of FIFOs is their one-to-two cycle latency that is incurred when the FIFO is either full or empty, and that scenario is highly typical with periodic clock domains where the clock frequencies are different.

Mesochronous synchronizers are described in [1][3][4][14]. A rational clocking synchronizer for a special case of periodic domains in which the two clocks are related by the ratio of two small integers is described in[6]. Another synchronizer for a limited case of periodic domains is described in [5]. A plesiochronous synchronizer is proposed in [2]. It incorporates an exclusion detection controlling a multiplexer that selects either the data or a delayed version of it. While having a low latency and a “duplicate and miss” algorithm for the plesiochronous case, it is inapplicable to periodic domains. Another predictive synchronizer for plesiochronous domains is presented in [11]. It predicts the transmit clock behavior compared to the receive clock in advance. Using this data an “unsafe” signal is produced to control data latching.

Dally & Poulton [1] suggest a predictive synchronizer for *periodic* clock domains in which two versions of the data are latched and selected according to the output of a phase comparator that compares the two clocks. The circuit is non-adaptive, requiring advanced knowledge of the two frequencies, and does not handle missed or duplicate data samples.

We investigate *Adaptive Predictive Synchronizers* for low-latency bridging periodic domains where the two frequencies are unknown in advance at design time and may also change from time to time. Such synchronizers are also suitable for other types of domain relationships. In particular, predictive synchronizers are designed for high performance minimal latency transfer of data almost every cycle (of the slower clock). The synchronizers must prevent missing any data or sampling the same data more than once.

In Section 2 we describe the problem and introduce the circuits of the predictive synchronizer. Different components of the synchronizer are presented in Section 3, and miss and duplicate handling are discussed in Section 4. Section 5 analyzes the adaptation time of the synchronizer.

2 Synchronizer Overview

Consider high bandwidth data sent from a transmitter clock domain to a receiver domain. The data lines change state simultaneously with the rising clock of the transmitter domain, and the transmitter clock is sent to the receiver together with the data, serving as a “data valid” or “ready” signal (this is also termed “source synchronous” data transfer). Metastability may happen at the receiving end if the receiver (sampling) clock rises simultaneously with the transmitter clock.

The *Two-Way Adaptive Predictive Synchronizer* (**Fig. 1.** A Two-Way Predictive Synchronizer with Miss and Duplicate Protection) synchronizes bi-directional communications between two periodic clock domains (‘Left’ and ‘Right’). The synchronizer receives the two clocks, and manages safe data transfers both ways. It produces SEND and RECV control outputs to both domains, indicating when it is safe to receive and send new data on both sides, avoiding data misses and duplicates due to mismatched clock frequencies.

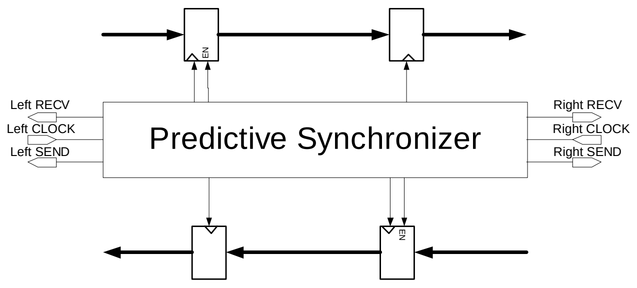


Fig. 1. A Two-Way Predictive Synchronizer with Miss and Duplicate Protection

Fig. 2 shows the internal structure of the two-way predictive synchronizer. Since it is symmetric, in the following we will mostly consider only one half of it, the part that moves data from Left to Right. We adopt the term “Local Clock” for the receiver’s clock (the Right Clock in this case) and “External Clock” for the sender’s (Left) clock. In the following we describe the synchronizer circuits in a top-down manner.

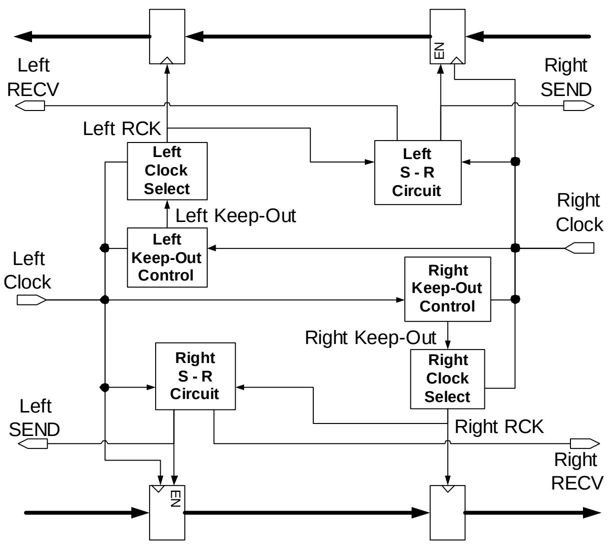


Fig. 2. Architecture of the two-way predictive synchronizer

One way of avoiding synchronization failures is to delay the sampling (local) clock when the input changes simultaneously with it. In order to decide if we need to delay, we track the rising edges of both the local and external clocks. If the rising edges occur "close" to one another (within a predetermined range d) we delay the sampling clock by a predetermined amount T_{KO} to "keep out" of the danger zone. We can realize such a scheme by using the keep-out signal as a selector to a multiplexer on the clock (**Fig. 3**).

Thanks to the periodic nature of the clocks, we can reduce the number of cycles needed for synchronization by predicting the relative clock timing one cycle in advance. Let T_{LOCAL} and T_{EXT} be the clock periods of the receiver (local) and transmitter (external) clocks, respectively. Let's assume that we have a conflict at time zero. In order to have a second conflict, an integral number of cycles of the local clock must span the same time as an integral number of cycles of the external clock, namely there should exist some *K* and *N* such that:

$$N \times T_{local} = K \times T_{ext}$$

$$(N-1)T_{local} = KT_{ext} - T_{local} = KT_{ext} - \frac{K}{N}T_{ext} = \Delta + \left(K - \left\lceil \frac{K}{N} \right\rceil\right)T_{ext}$$

$$\Rightarrow \Delta = \left\lceil \frac{K}{N} \right\rceil T_{ext} - T_{local}$$

This means that if we delay the external clock signal by Δ we will get a conflict after *N-1* cycles of T_{local} instead of *N*. The predictor (contained in the "Keep-Out Control") is shown in Section 3. We could similarly predict the clock more than one cycle in advance, but the further in advance we try to predict, the larger the cumulative error (resulting from skew, drift, and jitter in the prediction circuits).

The predicted external clock is phase-compared to the local clock. The phase comparator detects a conflict when the rising edges of the two clocks happen at less than a certain delay d between them. The detection is used as a selector to the clock delay multiplexer (**Fig. 3**). **Fig. 4** shows waveforms of a conflict and its resolution (when the sampling clock is delayed). Combinational logic can be placed right after the sampling register of the receiver, as can be seen in Fig. 3. The maximal allowed combinational delay is T-TKO, and thus the incurred synchronization latency is only TKO, a small portion of the cycle time.

3 Keep-Out Control

The Keep-Out Control (**Fig. 3**) consists of the clock predictor and the phase comparator. A conceptual clock predictor is shown in **Fig. 5**[1]. The "T_{LOCAL} delay" block is an adaptive delay line tuned to the cycle time of the local clock. The "programmable delay" block is another adaptive delay line. The feedback circuit adjusts it so that the two inputs to the conflict detector rise at approximately the same time. Once adjusted, the "predicted clock" output provides a copy of the external clock one local cycle time in advance.

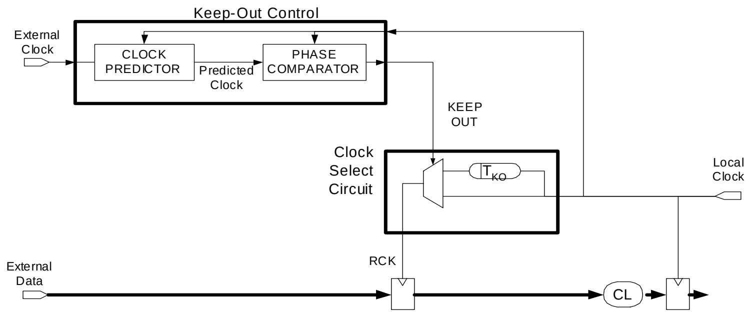


Fig. 3. The Keep-Out Control, sampling clock selector, and the receiver's sampling register

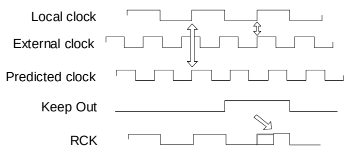


Fig. 4. A conflict is predicted one cycle in advance, delaying input sampling

The programmable delay line consists of a simple digital tapped inverter chain [4][15]. The delay can be either increased or decreased one step each cycle. A four-flip-flop circuit is used to construct both a basic conflict detector (Fig. 6) and a two-way conflict detector (Fig. 8). These two conflict detectors allow about one half cycle time for any metastability in the first sampling stage to resolve. We show below (Section 3.1) that it is an extremely safe conflict detector.

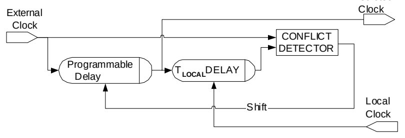


Fig. 5. Adaptive clock predictor

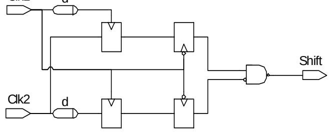


Fig. 6. Conflict Detector

The conflict detector (**Fig. 6**) detects simultaneous rising edges (within *d* delay of each other). The "T_{LOCAL} delay" block (**Fig. 7**) is a simplified digital DLL, consisting of a conflict detector and two programmable delays. This circuit starts with a minimal delay and increases the delay until it is equal to a full cycle. The flip-flop provides a loop delay (of two local clock cycles) until the lower programmable delay line has had time to adjust to a new value and the conflict detector has responded to that new value. Once the lower delay line has converged to T_{LOCAL}, its programming code is copied to the upper delay line.

This circuit, however, does not track T_{LOCAL} very well: If the local clock cycle decreases, the delay line would have to be reset and tuned from the beginning. Hence we employ a two-way conflict detector (**Fig. 8**), which operates bi-directionally to either decrease or increase the delay. It detects which clock rises first ("a < b" in the figure means "a rises before b" and "a~b" means that they rise simultaneously). If the delayed clock rises before the non-delayed one, then the delay is increased, and vice versa.

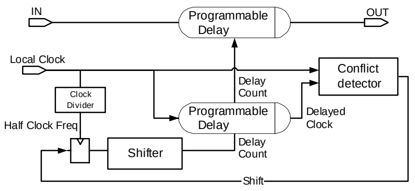


Fig. 7. The adjustable T_{LOCAL} Delay Line

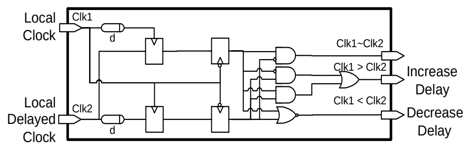


Fig. 8. Two-way conflict detector

To understand the (180°) phase shift signal in **Fig. 8**, consider the scenario in **Fig. 9**. When the two clocks have precisely inverted phases, the T_{LOCAL} block cannot decide whether to increase or decrease the delay. We resolve that ambiguity by deciding to always increase the delay in that case.

The T_{LOCAL} DLL (**Fig. 7**) converges to the cycle time of the local clock as follows. The minimum step by which the delay line can be increased or decreased is *q*. The DLL starts from a small delay and increases the delay by steps of size *q* up to the complete local cycle time, taking T_{LOCAL}/q steps. Each step takes two local clock cycles, and thus the total convergence time of the DLL is $\left(T_{LOCAL}/q\right)\times 2\times T_{LOCAL}$.

We now examine the clock predictor circuit in more detail (**Fig. 10**). Programmable Delay 1 is started at zero delay, and the delay is increased progressively until the two inputs to the conflict detector concur. The loop delay in this case (the delay between successive steps of delay adjustment) must be the maximum of the two clock cycles. Since it is unknown in advance which clock is slower (the external or the local clock), the rate reducer (**Fig. 11**) waits for at least one cycle of each, synchronizing to each clock in turn by means of two flip-flops.

The delay introduced by the rate reducer between successive adjustments of Programmable Delay 1 (two passes around the circle in the rate reducer) is $4T_{LOCAL}+4T_{EXTERNAL}$ (in the worst case). Programmable Delay 1 must be tuned to a total delay of $\varepsilon=\left\lceil K/N\right\rceil\times T_{EXTERNAL}-T_{LOCAL}$ (as in Section 2). Since each time the delay is increased only by *q*, the number of steps required to tune Programmable Delay 1 is $\frac{\left(\left\lceil K/N\right\rceil\times T_{EXTERNAL}-T_{LOCAL}\right)}{q}$ As the delay between successive steps is determined by the rate reducer, the total tuning time for Programmable Delay 1 is

$$\frac{\left(\left\lceil K/N\right\rceil\times T_{EXTERNAL}-T_{LOCAL}\right)}{q}\times 4\left(T_{LOCAL}+T_{EXTERNAL}\right)$$

The total adaptation time comprises the above expression plus the DLL convergence time, $(T_{LOCAL}/q)\times 2\times T_{LOCAL}$. This adaptation time is further analyzed in Section 5.

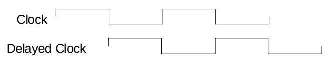


Fig. 9. 180º Phase shift

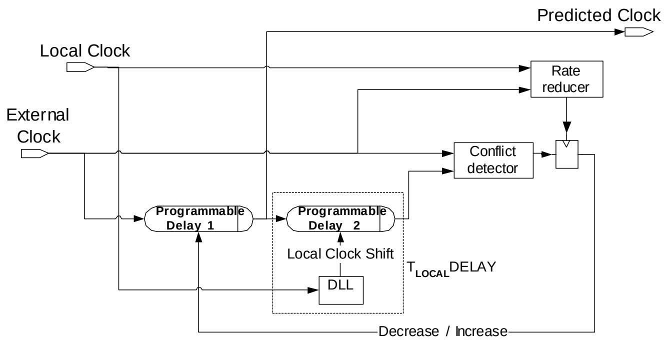


Fig. 10. Adaptive clock predictor

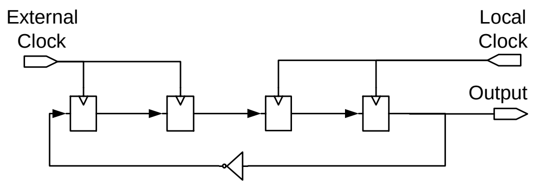


Fig. 11. Rate Reducer

3.1 Metastability of the conflict detector

We note that, in a standard SoC (namely a digital IC based on standard cells and designed with standard EDA tools) the shortest clock cycle is typically about 160 FO4 inverter delays [16]. The nominal FO4 inverter delay depends mostly on the process technology (e.g. about 30ps for 0.13µm logic CMOS technology). Thus, the shortest high phase (with a 50% duty cycle clock) is about 80 inverter delays long. The lower bound is determined as follows. When the two clocks differ by about *d*, the conflict detector can enter metastability and take a long time to resolve. The MTBF of such a conflict detector can be determined as follows. Assume τ is one inverter delay, W is two inverter delays, and F_D=F_C (conflict is possible every cycle when the two clocks are about equal frequency), then [12]:

$$MTBF = \frac{e^{T/2/\tau}}{WF_C F_D} = \frac{e^{80}}{2 \times \frac{1}{160} \times \frac{1}{160}} \times \tau \approx 10^{39} \text{ inverter delays} \approx 10^{20} \text{ years}$$

That very safe MTBF (quoted in years) scales quite well over a number of process technologies. In fact, even if the time reserved for metastability resolution is halved to about 40 FO4 inverter delays, the MTBF would still safely exceed 10,000 years, an acceptable goal for most SoCs.

The conflict resolution delay *d* should be large enough to accommodate local jitter and delay variations inside the conflict detector. For instance, *d*=10 FO4 inverter delays could be used. The "keep out" delay T_{KO} must be longer than *d* and shorter than one half cycle of the fastest possible clock. For example, T_{KO}=20 inverter delays could be used.

Thus, while it is unlikely that any conflict detector has not resolved by the time it is used, it may have resolved to an unexpected value. Let us consider that situation for each of the conflict detectors. Note that in each half of the predictive synchronizer (**Fig. 2**) only the Keep-Out Control receives both clocks. Inside the Keep-Out Control, any one of the three conflict detectors may be affected. The effect of bad resolution of the conflict detectors in the T_{LOCAL} DLL and in the loop that adjusts Programmable Delay 1 (**Fig. 10**) is merely a potential extension of the convergence time. In the case of the phase comparator that generates Keep-Out (**Fig. 3**), note that it could become metastable when the local and external clocks are about *d* apart. However, a wrong value of Keep-Out is inconsequential in this case, and the data is sampled safely whether Keep-Out is asserted or not. The different cases are exemplified in **Fig. 12**.

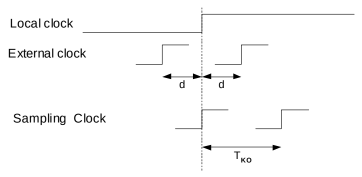
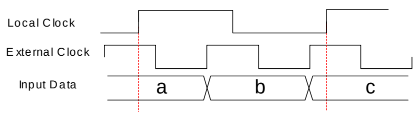


Fig. 12. Cases of incorrectly resolved Keep-Out: Input data is sampled either at local clock or after T_{KO} delay; in either case, it is safely separate from the rising edge of the external clock.

4 Misses & Duplicates

When the transmitter clock is faster than the receiver's, the transmitter cannot send new data every cycle or else some data values will be missed by the receiver (**Fig. 13**). Conversely, when the receiver uses a faster clock, it cannot sample the input every cycle or else it will sample the same data more than once (**Fig. 14**).



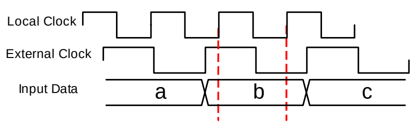


Fig. 13. Miss Condition

Fig. 14. Duplicate Condition

The complete two-way predictive synchronizer (**Fig. 1**) provides control signals to avoid missed and duplicate data. The SEND signal (generated on the receiver side) guarantees (when low) that a fast sender will keep its output unchanged until it is sampled, and the RECV signal, generated by the receiver, stops (when low) a fast receiver from using the same data more than once. **Fig. 15** describes the algorithm that generates SEND and RECV in terms of a Signal Transition Graph (STG). RECV is set upon a rising edge of the external clock (new data is available) and reset by a rising edge of the local clock (new data has been received, ready to receive the next one). SEND is simply the opposite of RECV. The STG is implemented, for instance, by the circuit of **Fig. 16**. Note that RCK is employed instead of the Local Clock; otherwise, that circuit would have been subject to metastability. RCK is now guaranteed to never coincide with the External Clock. Note also that this circuit cannot be synthesized directly from the STG by tools such as Petrify [17].

An example waveform of a fast receiver is given in **Fig. 17** and a fast sender scenario is shown in **Fig. 18**.

Although metastability cannot occur in the S-R circuit, it can occur in the conflict detector leading to a wrong keep-out signal resulting in a wrong delaying of RCK. This can happen as shown before in **Fig. 12** when the clocks are *d* time apart. The case when the external clock rises *d* time before the local clock is inconsequential because the relative ordering of the two clocks is unaffected (RCK succeeds the external clock).

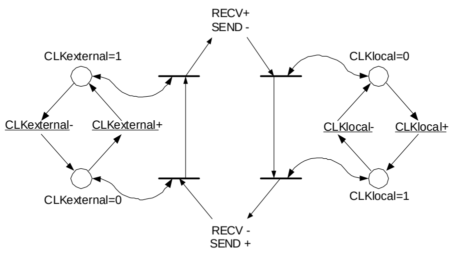


Fig. 15. SEND and RECV Control STG (double arrows are probe arcs: the transition happens if the place holds a token)

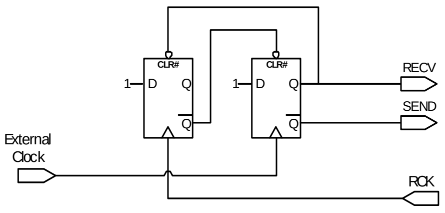


Fig. 16. Duplicate and Miss Control Circuit

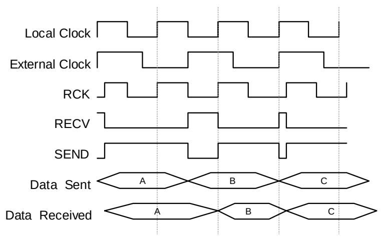


Fig. 17. Fast Receiver Waveforms

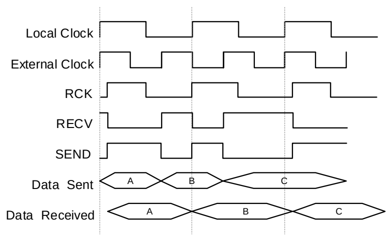


Fig. 18. Fast Sender Waveforms

We now look at the case when the external clock rises after the local clock. In the case of a fast sender, RECV and SEND are extended (**Fig. 19**) and sampling the data by the receiver is delayed until the next cycle (**Fig. 18**). The worst case may happen when both clocks have the same frequency, and (when the external clock lags the local clock by *d* time) Keep-Out happens to oscillate every cycle—the resulting data transfer rate is effectively cut in half (but misses and duplicates are avoided).

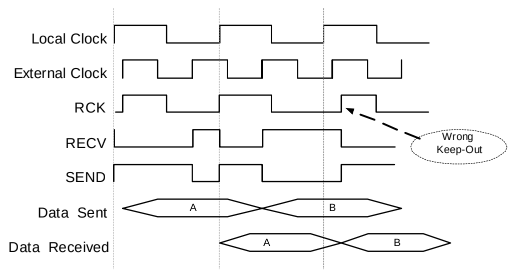


Fig. 19. Wrong Keep-Out on Fast Sender

5 Analysis

As shown in Section 3, the total adaptation time is

$$\frac{\left(\left\lceil {K}/{N} \right\rceil \times T_{EXTERNAL} - T_{LOCAL}\right)}{q} \times 4\left(T_{LOCAL} + T_{EXTERNAL}\right) + \frac{T_{LOCAL}}{q} \times 2 \times T_{LOCAL}$$

The chart in **Fig. 20** shows the adaptation time, measured in cycles of a 100MHz local clock as a function of the external clock frequency, where the delay resolution *q*=100ps. Note that this is a very simplistic analysis; minor modification of the circuits could enable binary-search type of convergence, which would reduce total adaptation time by a logarithmic factor.

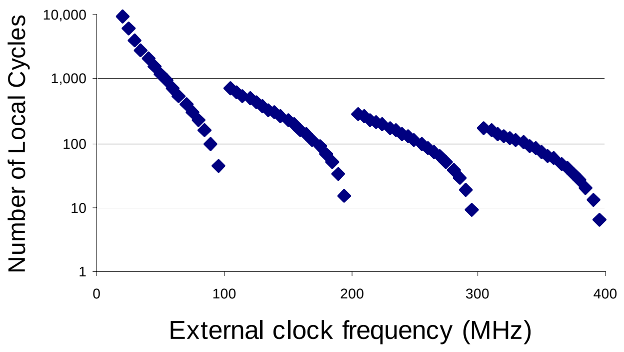


Fig. 20. Adaptation time of the predictive synchronizer (100MHz local clock, 100ps digital delay resolution)

6 Conclusions

A two-way adaptive predictive synchronizer for SoC with multiple clock domains has been presented. The synchronizer takes advantage of the periodic nature of clocks in order to predict potential conflicts in advance, and to conditionally employ an input sampling delay to avoid such conflicts. The result is conflict-free synchronization with almost zero latency (much less than one cycle). The adaptive predictive synchronizer adjusts automatically to a wide range of clock frequencies, regardless of whether the transmitter is faster or slower than the receiver. The synchronizer also avoids sampling duplicate data or missing any input. Adaptation to changing clock frequencies have been shown to require anywhere from tens of cycles to ten thousand cycles, depending on the relative frequencies.

References

[1] W.J. Dally and J.W. Poulton, “Digital Systems Engineering,” Cambridge University Press, 1998.

[2] L.R. Dennison, W.J. Dally and D. Xanthopolous, “Low-latency plesiochronous data retiming,” *Proc.16th Conf. Adv. Res. in VLSI, pp.* 304-315, 1995.

[3] A. Chakraborty, M.R. Greenstreet, “Efficient self-timed interfaces for crossing clock domains,” *Proc.9th IEEE Int. Symp. Asynchronous Circuits and Systems (ASYNC’03),* pp. 78-88, 2003.

[4] Y. Semiat and R.Ginosar, “Timing Measurements of Synchronization Circuits,” *Proc. 9th IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC’03),* 2003.

[5] J. Gandhi, “Apparatus for fast logic transfer of data across asynchronous clock domains” USA Patent 6,172,540, 2001.

[6] L.F.G. Sarmenta, G.A. Pratt, S.A. Ward, “Rational Clocking,” *Proc. ICCD,* pp.217-228, 1995.

[7] D.G. Messerschmitt, “Synchronization in Digital System Design,” *IEEE J. Selected Areas in Communication,*8(8), 1990.

[8] T. Chelcea and S.M. Nowick, “Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols,” *Proc. ACM/IEEE Design Automation Conference*, 2001.

[9] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer,” *Proc. 9th IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC03),* 2003.

[10] Kessels, Peeters, Kim, “Bridging Clock Domains by synchronizing the mice in the mousetrap,” Proc. *PATMOS,* 2003.

[11] W.K. Stewart, S.Ward, “A solution to a special case of Synchronization Problem,” *IEEE Trans. Comp.,*37(1), 1988.

[12] C. Dike and E. Burton, “Miller and Noise Effects in a Synchronizing Flip-flop,” *IEEE J. Solid-State Circuits,* 34(6), pp. 849-855, 1999.

[13] D. J. Kinniment, A. Bystrov, and A. Yakovlev, “Synchronization Circuit Performance,” *IEEE J. Solid-State Circuits,*37, pp. 202--209, 2002.

[14] R. Ginosar and R. Kol, “Adaptive Synchronization,” *Proc. ICCD*, 1998.

[15] S.W. Moore, G.S. Taylor, P.A. Cunningham, R.D. Mullins, and P. Robinson, “Self-Calibrating Clocks for Globally Asynchronous Locally Synchronous Systems,” *Proc. ICCD*, 2000.

[16] International Technology Roadmap for Semiconductors (ITRS), 2001.

[17] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," IEICE Transactions on Information and Systems, Vol. E80- D, No. 3, pp. 315– 325, 1997.