Timing Measurements of Synchronization Circuits

Yaron Semiat and Ran Ginosar
VLSI Systems Research Center, Technion—Israel Institute of Technology, Haifa 32000, Israel
ran@ee.technion.ac.il

Abstract

A regular (two-flop) synchronizer and six multi-synchronous synchronizers are implemented on a programmable logic device and are measured. An experiment system and method for measuring synchronizers and metastable flip-flops are described. Two separate settling time constants are shown for a metastable flop, confirming earlier results of Dike and Burton [1]. Clocking cross-talk between asynchronous clocks is demonstrated. The regular synchronizer is useful for communications between asynchronous clock domains, while the other synchronizers can provide higher bandwidth communications between multi-synchronous and mesochronous domains.

1. Introduction

Synchronization is a challenging topic that has been investigated intensively. Most treatments of the subject, however, were limited to paper designs and analytic studies. Actual laboratory measurements and in-depth analysis of synchronizers have been performed in very few cases [1][2][3][4].

Large VLSI chips tend to employ asynchronous inter-module timing due to two principal reasons. First, it is sometimes more economical (in terms of area, power and design time) to break a large synchronous chip (or section of a chip) into *multi-synchronous* modules, which use the same basic clock frequency but do not require the exact same phase of the clock [5]. Multi-synchronous timing can be based on thrifty clock distribution networks, which avoid the heavy area and power penalty of assuring minimal skew across a large chip. Second, interfacing the chip to a variety of external busses ticking at different frequencies imposes a requirement for the chip to contain multiple unrelated clock domains. A communication chip that connects a 100MHz data link to a 66MHz PCI bus is one such example. Both types of multiple-clock domain chips are sometimes termed GALS (globally asynchronous, locally synchronous) systems.

Two separate clock domains are 'mesochronous' if they are clocked at the same frequency but at a fixed relative phase difference [6]. If the phase difference drifts over time, they are called 'multi-synchronous' [5]. If the clock frequencies are close but different, they are 'plesiochronous.' In multi-synchronous GALS systems, all modules receive the same clock frequency. The design of inter-module communications can take advantage of that fact and employ mesochronous synchronizers for higher bandwidth than possible with the more general two-flop synchronizers [7]. However, relative clock phases drift over time (typically due to intra-die temperature and voltage temporal variations) thus requiring adaptive multi-synchronous synchronizers [5] that either periodically or continuously adapt to the varying phase differences. Similar conditions often arise among separate chips on a board, where the chips are clocked by the same system clock.

The analysis of synchronizers for on-chip cross-clock domain communications is quite difficult. Circuit simulations of synchronizers only provide a partial characterization [1][2][4]. Typical logic validation tools are totally ignorant of synchronization failures. Post-production testing also provides very little help. The only useful metric proposed in the literature is that of MTBF, which is only indirectly driven out of approximately defined parameters. In this paper we extend the work of [1] [2][3], which have considered only simple synchronizers, and we show lab measurement validation of a variety of multi-synchronous adaptive synchronizers.

Section 2 explains the experiment system and the data analysis method, and demonstrates them investigating a metastable flip-flop. Section 3 describes seven different synchronizers, one general and six for multi-synchronous applications, and validates their performance by means of timing measurements. The findings are analyzed in Section 4. Full-color figures of this paper are available on the web [11].

2. The Experiment System

This section describes the setup used for all the experiments. It demonstrates the waveforms of normally switching and metastable flip-flops, and discusses clock cross-talk noise.

2.1 Hardware

Figure 1 describes the hardware used for the experiments. The setup follows that of Dike & Burton [1]. All designs have been implemented on the same Altera EPF10K20 programmable logic device (PLD). Two pulse

generators provide the clock and data inputs, operating at 25.175 and 25.2 MHz, respectively. The slight difference in the frequencies result in inputs having uniformly distributed delays in reference to the clock, at a periodicity of 25 KHz. The data output of the PLD is connected to the trigger input of a HP83480A oscilloscope. The clock signal is connected through the PLD to the data channel of the scope. The digital sampling scope is capable of continuous data accumulation and the results are available for statistical analysis.

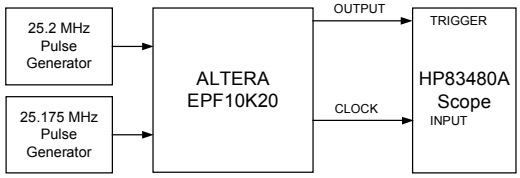

Figure 1: Experiment setup

2.2 Synchronous Sampling of a Flip-Flop

In this experiment the data pulse-generator is turned off. A toggling flop's output serves as a synchronous input for the flop under test (Figure 2). Each data point accumulated by the scope represents one sampled rising transition of the CLOCK signal (Figure 3). Its horizontal displacement indicates the delay from the clock input to the data output of the flip-flop.

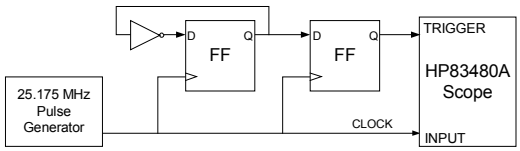

Figure 2: Synchronous sampling circuit

Figure 4 shows the result of synchronous sampling by a flip-flop. The delay between the clock and the output edge is fixed, with only about 10ps delay variation (the scope's jitter at its trigger input is specified at 2.5ps). The variation appears to be symmetrically distributed around the center value. This experiment represents an accumulation of 2.2 million data points, collected over about 5 minutes. Note that only 0.06% of all edges are sampled—this is used below to filter out safe edges and to constrain the measurement to potential metastable ones. The distribution of the accumulated data along the horizontal (time) axis is shown in Figure 5; the method used to generate that chart is explained below in Section 2.4.

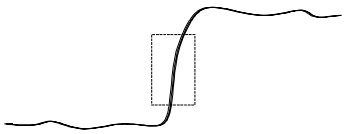

Figure 3: Collected data (the dotted area)

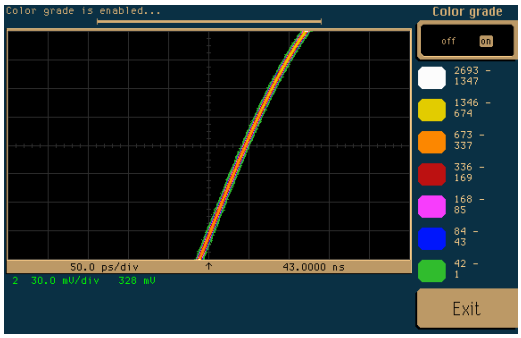

Figure 4: A synchronous flop

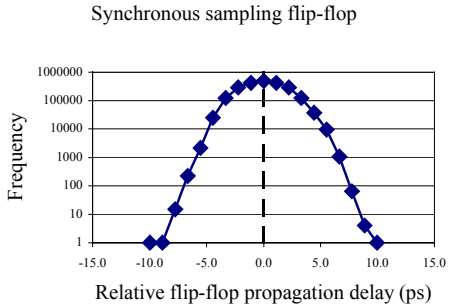

Figure 5: Flip-flop propagation delay frequency distribution for synchronous sampling; t=0 represents the average propagation delay

2.3 Clock Cross-Talk Noise

The following experiment demonstrates the effect of clock cross-talk noise. The circuit shown in Figure 2 uses only one clock source. When turning the second pulse generator on, a much noisier waveform is received, even though that other signal is not used in the design. Figure 6 demonstrates this phenomenon.

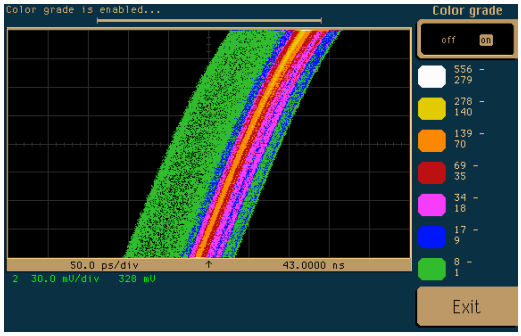

Figure 6: Synchronized sampled data with clock cross-talk noise

The noise distorts the 'clean' signal of Figure 4 in two ways: Delay variation increases from 10ps to 150ps, and the distribution around the mean is asymmetric and multimodal. It appears that the noise is injected through capacitive coupling of the two clock distribution networks, which may be laid down side by side in the PLD. Note that different designs may demonstrate different levels of cross-talk noise, depending on the actual placement and routing that happen to be generated for each implementation. Unlike custom chip design, the PLD user typically has very little control of such physical parameters. These distortions should be taken into consideration while analyzing the experiments in this paper, which all depend on the two clocks operating simultaneously.

2.4 Data Analysis

The digital sampling scope, operating in a special data accumulation mode, collects the data in the form of a two-dimensional array of counter values. Each counter represents the number of edges that have been sampled when passing through the corresponding point on the scope display. The array consists of 256×451 counter values. The data file is subsequently aligned to enable accumulation along the curved data axis, as shown on the left side of Figure 7. Next, the data values are integrated along the vertical axis, resulting in the chart on the right hand side of the figure.

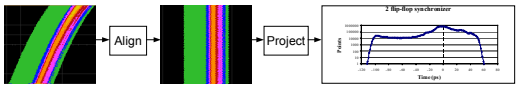

Figure 7: Data analysis method

2.5 Metastable Behavior of a Flip-Flop

The two pulse generators, being asynchronous to each other, provide the clock and data inputs of the synchronizer flop, as in Figure 8. A clock-data conflict occurs roughly at a 25 KHz rate, and occasionally it leads to metastable behavior. Most normal switching events are filtered out by means of the clear signal, which precedes the data input transition by T_d.

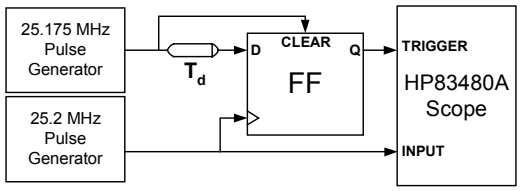

Figure 8: Circuit demonstrating metastability

The accumulated data is shown in Figure 9. The frequency distribution of the incremental flop clock-to-output propagation delay (that is, how much longer it takes relative to normal propagation delay) is plotted in Figure 10. The peak around zero represents a large number of cases with normal propagation delay. The dip in the chart between zero and 300ps is explained by Dike & Burton [1] as resulting from Miller effects. When closely examined, the chart to the left of 300ps reveals two slightly different slopes, representing two separate values for the time constants τ, as first discovered by Dike & Burton [1]: Deep metastability is characterized by τ_{ms}=130ps in the region to the left of the 700ps point, and the deterministic region (which shows an incremental delay between 300-700ps) shows τ_{det}=120ps. Thus, the deterministic region includes cases where the flop propagation delay is longer by up to 700 ps relative to the normal flop delay, and in deep metastability the flop takes more than 700 ps to resolve and settle to either 0 or 1. Note how very close these two τ values are.

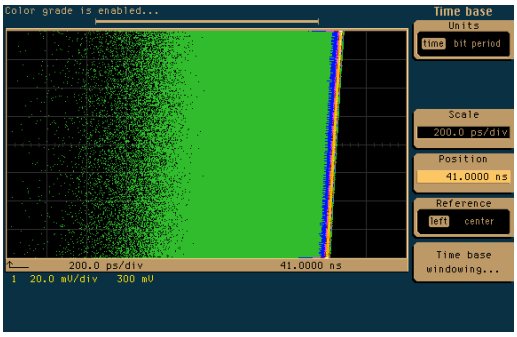

Figure 9: A metastable flip-flop

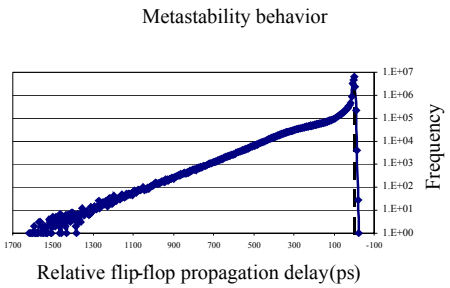

Figure 10: Flip-flop propagation delay frequency distribution for metastable sampling; t=0 represents the normal propagation delay

3. Adaptive Synchronization Circuits

In this section different adaptive synchronizers are presented. The test circuit is shown in Figure 11. As explained in Section 2.1, the slight frequency difference of the two pulse generators results in the data edges continuously sweeping over the clock edges, and resulting in periodic clock-data conflicts (at the beat frequency, namely the difference of the two clock frequencies). This setup actually creates a plesiochronous environment in order to test mesochronous [6][7] and multi-synchronous [5] synchronizers.

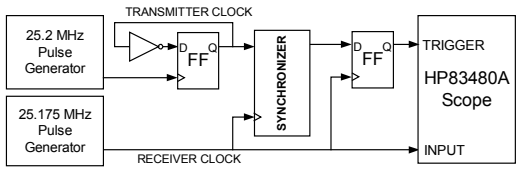

Figure 11: Synchronizer experiments setup

3.1 Two-Flop Synchronizer

A standard two-flop synchronizer is shown in Figure 12. It incurs at least one clock cycle penalty on latency, and in certain cases it may consequently limit the throughput.

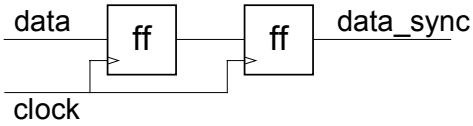

Figure 12: A two-flop synchronizer

The measured behavior of a two-flop synchronizer is shown in Figure 13. MTBF for the two-flop synchronizer is estimated at

$$\frac{e^{T/\tau}}{T_W F_C F_D} = \frac{e^{40ns/130ps}}{200ps \cdot 25MHz \cdot 1MHz} = \frac{e^{308}}{5000} \approx 10^{122} \text{ years.}$$

This is quite safe—recall that our universe is only 10^{10} years old… The synchronized output should appear like that of Figure 4, but, apparently due to clock noise, it resembles that of Figure 6 instead.

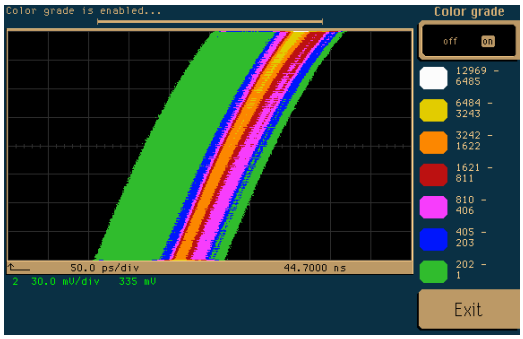

Figure 13: Two-flop synchronizer output

3.2 Data Delay Synchronizer

The data delay adaptive synchronizer [5] delays the input data if the data are suspected to switch during the 'danger zone' around the clock rising edge. A programmable digital delay line is inserted before the first latch (Figure 14). Data timing is determined and the delay is programmed during a *training session.* In our measurements, the output of the synchronizer is blocked during the training session. Note that the adaptation rate, which is also the rate of entering a training session, is 25 KHz in the measurement system (the beat frequency). The synchronized output is shown in Figure 20. The delay variation is approximately one half that of the two-flop synchronizer, but we believe that it is only an artifact of the specific PLD place and route of this circuit. Figure 21 shows the output of the delay line; multiple delay regions are evident. They are all aligned once latched, as in Figure 20.

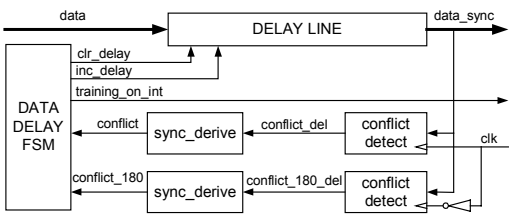

Figure 14: The data delay synchronizer

The delay-line delays the input signal to a 'safe' time, when sampling of the signal can be done correctly, without metastable behavior. The delay line FSM controller generates the clr_delay signal at the start of a training session, setting the delay line to its minimum value. The inc_delay control signal increments the amount of delay in the delay-line by one unit every iteration of the training session.

A training session begins upon arrival of the conflict signal, which suggests that the data is in the danger zone. The training session stops when conflict180 arrives, indicating that the data is around the middle of the cycle, and therefore safe to sample. Those two conflict signals are synchronized by the sync_derive units. The conflict detect modules are designed with margins so that missing a signal (due to metastable state in the sync-derive synchronizers) will not prevent the circuit from entering its training session eventually.

Figure 15 describes the implementation of a delay-line in the Altera PLD device, using multiplexers and Altera delay elements ("lcells"), which provide a nominal 1.5 nsec fixed delay each. Minimal delay is achieved by loading the shift register with all '1's. The delay is increased by left shift with a '0' entering from the right.

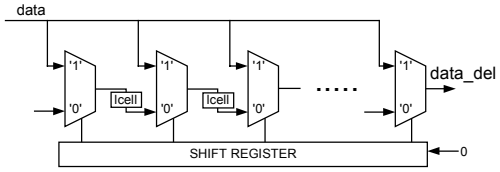

Figure 15: Delay-line circuit in Altera PLD

The conflict detect module (Figure 16) is designed to detect situations where the data signal is inside a time window around the clock rising edge. Both rising and falling edges of the data signal are analyzed. The time window is set by the fixed delay-lines in the module. Note that conflict signals in the module have half a cycle width: If the delayed_data_g2 (or delayed_data_n_g2) signal is high it will be high only when its ME's R2 input is high, and since the latter is a clock signal it will be high only for half a cycle. Since it is connected to an AND gate, conflict1 and conflict2 will be of half-cycle width, propagating to the output conflict signal. The conflict signal may altogether be missed, due to the very clock-data conflict that it tries to detect! The sync_derive circuits, which synchronize the conflict signals, sample the conflict at both the rising and falling edges of the clock, in order not to miss its high value, and repeat the sampling every cycle to increase the probability of catching a conflict.

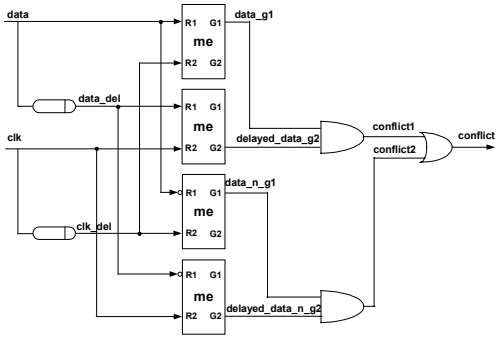

Figure 16: Conflict detector circuit

The mutual exclusion element (ME) is shown in Figure 17. This LUT-based PLD implementation provides a very high probability of mutual exclusion, and in practice we have been unable to detect any failure. But this circuit cannot completely guarantee it; for absolute mutual exclusion, either a full-custom or gate-based cells are required [8]. To counter that limitation, the conflict detector is operated multiple times during a training session; the probability of a mutual exclusion element failing on all attempts is negligible.

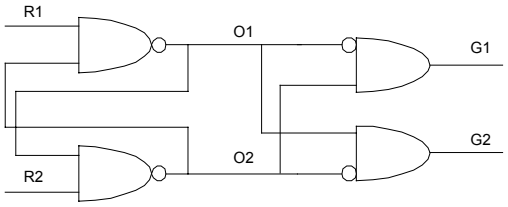

Figure 17: PLD mutual exclusion element

The sync derive module (Figure 18) synchronizes the conflict-detector output. The conflict signal can suffer delay and the chance of being missed, as described above. The synchronizer samples the conflict signal every cycle, to increase the likelihood that it is eventually caught. The training session allows ample time for conflict resolution; it may take many cycles. Hence, the conflict signal is synchronized with a two-flop synchronizer. Since the conflict signal is only half a clock period wide, it is sampled at both the rising and falling edges of the clock.

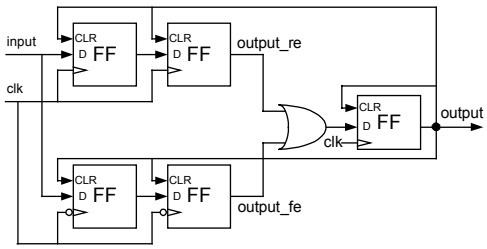

Figure 18: Sync derive circuit

Upon arrival of a conflict signal, the FSM (Figure 19) resets the delay line to its minimum value. It subsequently iterates between WAIT_ CONFLICT180 and TRAINING_INC states. In each iteration it increments the delay line by one unit, and checks whether conflict_180 signal has arrived.

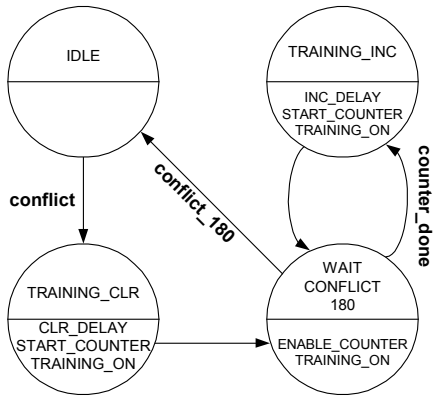

Figure 19: Data Delay FSM

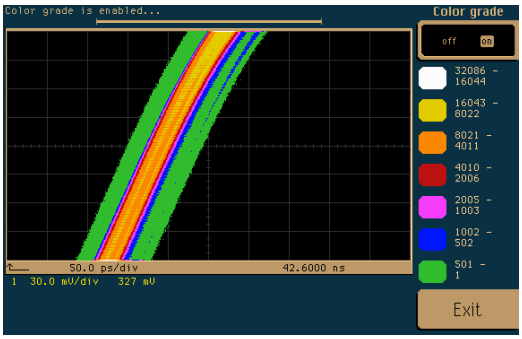

Figure 20: Data delay synchronizer output

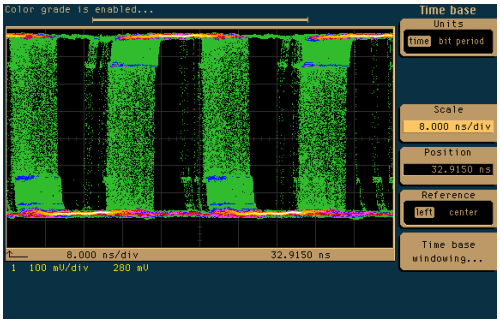

Figure 21: Delay line output of the data delay synchronizer

3.3 Dual Data Delay Synchronizer

To minimize the slow-down caused by training sessions in plesiochronous or periodic applications [5][7], a second delay line is introduced in Figure 22. Imminent clock-data conflicts are continuously monitored; once detected, the delay is adjusted off-line and subsequently loaded into the data delay. The synchronized output is shown in Figure 24. Note that the synchronizer is insensitive to small mismatches in the delays of the two delay lines. The PLD technology, however, can assure sufficiently tight matching of the two delays; similar matching is also feasible in custom VLSI designs.

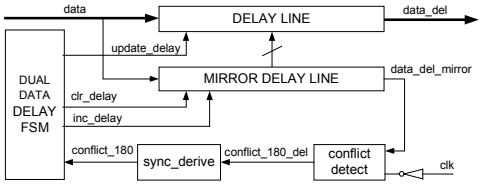

Figure 22: The dual data delay synchronizer

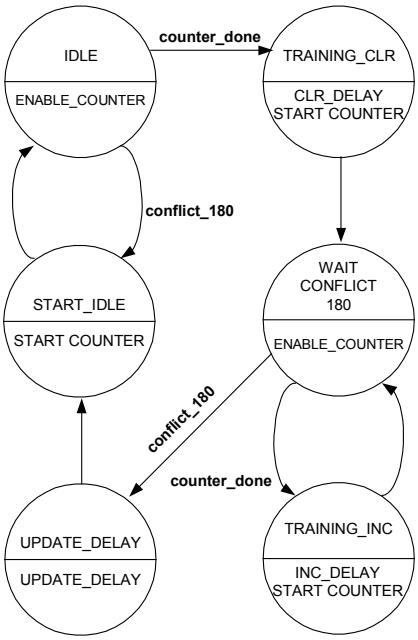

Figure 23: Dual data delay FSM

If a conflict_180 signal has not arrived for a given number of cycles (Figure 23), counter_done rises and the machine clears the delay line. Then it waits until a conflict_180 signal arrives. If it doesn't arrive after a given number of cycles it increments the delay in TRAINING_INC state. When conflict_180 signal arrives, the value of the mirror delay is transferred to the delay line through the UPDATE_DELAY state.

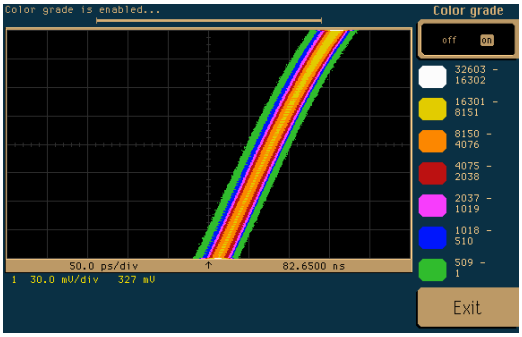

Figure 24: Dual data delay synchronizer output

3.4 Clock Delay Synchronizer

While Data Delay Synchronizers may adjust the delay independently for each separate input channel, they employ a large number of delay elements, which may incur a prohibitive area and power price. In contrast, the Clock Delay Synchronizer [7] adjusts only the timing of the first latching clock (Figure 26). The output is shown in Figure 25.

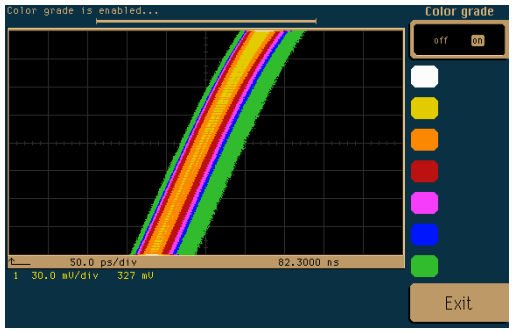

Figure 25: Clock delay synchronizer output

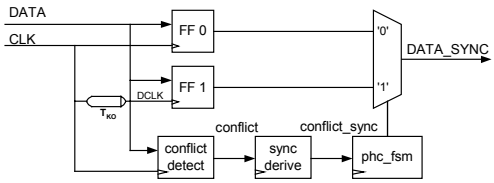

Figure 26: Clock delay synchronizer

The input data are sampled by two flip-flops FF0 and FF1. CLK is the receiving module's clock. CLK samples FF0 and CLK delayed by T_{KO} samples FF1. The conflict-detector and sync_derive circuits are described in Section 3.2. The control FSM is shown in Figure 27. Upon conflict_sync, FF1 is selected. Otherwise, FF0 is selected. Note that the clock delay synchronizer may be adjusted continuously, rather than through training sessions. The two alternative selections are evident in Figure 28: The bright line on the right is the output of FF0, and the one on the left is FF1. The two darker lines result from switching of the selector. An alternative design, more appropriate for wide data buses, samples the data only once but applies either CLK or CLK+T_{KO} to the sampling register (Figure 29).

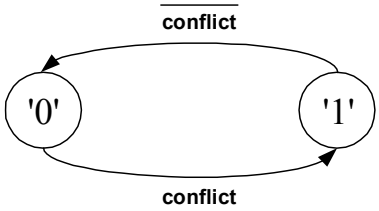

Figure 27: Clock delay synchronizer FSM

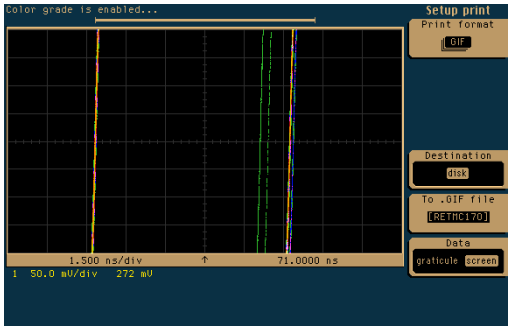

Figure 28: Selector output in the clock delay synchronizer

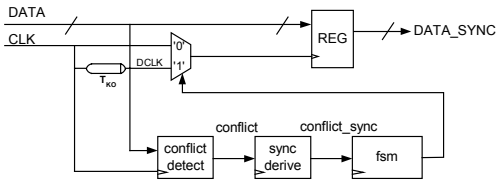

Figure 29: Simpler Clock delay synchronizer

3.5 FIFO Synchronizer

The dual-clock asynchronous FIFO provides another common means for crossing clock domains. A simplified version is suitable for multi-synchronous applications [7]. The input data are latched into a cyclic FIFO register (Figure 30) using XCLK, the transmitter clock, and are read using the receiver's clock, CLK. The XP pointer selects the register that latches the input data, while RP selects the register being read. The pointers are incremented by their respective clocks. The difference between XP and RP determines the FIFO latency, and it can be set as low as approximately one clock cycle

(depending on the phase difference of the two clocks). During a training session, XP is synchronized with CLK and compared with RP. If they are misaligned, RP is modified. Alternatively, if a longer latency is allowed, RP may be adjusted continuously during operation. The waveform of the experiment is shown in Figure 31.

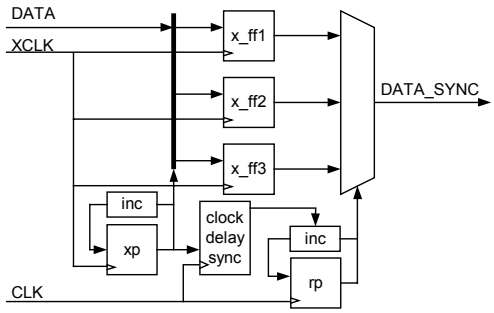

Figure 30: FIFO synchronizer with multi-synchronous support

It is interesting to compare the area utilization of the FIFO and the delay-line synchronizer circuits (Section 3.2). Assume that for a wide data bus (e.g., 32 bits) we can ignore the control circuits. The FIFO synchronizer requires three flops per data bit. When operating at, e.g., 200 MHz (a typical SOC clock frequency), and assuming that the delay of each delay unit is 50 psec, the delay-line synchronizer requires about 100 buffers per data line, making the FIFO synchronizer a favorable solution. On the other hand, in very high-speed designs operating at, e.g., 2 GHz (a typical CPU design), only 10 delay buffers per data line may be required. In such cases, the delay-line synchronizer may turn out to be more area-efficient. Further analysis is provided in Section 4.

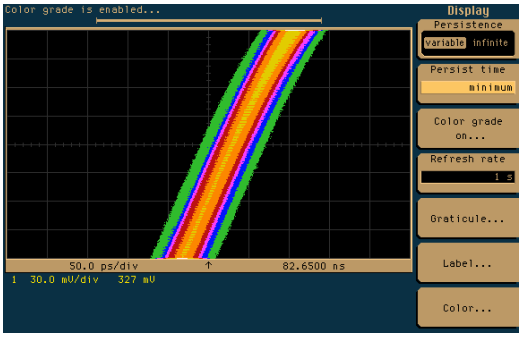

Figure 31: FIFO synchronizer output

3.6 Clock-Edge Synchronizer

The clock-edge synchronizer [9] analyzes four different samples of the same input data, taken with four different phases of the clock, and selects the safest sample out of the four possibilities. This synchronizer incurs a significant data latency, because phase selection is performed only after data sampling and analysis. Another disadvantage is the requirement for a 90 degrees phase of the input clock. Figure 32 shows the structure of this synchronizer, and the measured data are shown in Figure 33.

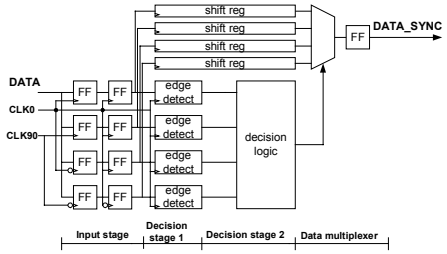

Figure 32: The clock edge synchronizer

The input is sampled by four different phases of the clock, and an additional clock cycle is allowed for settling of any metastability and aligning the four samples with the clock. Edge detectors locate the edge at a quarter cycle resolution. The decision logic selects the sample that is about one half cycle away from the first edge. The entire selection process takes about five clock cycles, and the latched inputs are delayed by the shift-registers until the selection process is done.

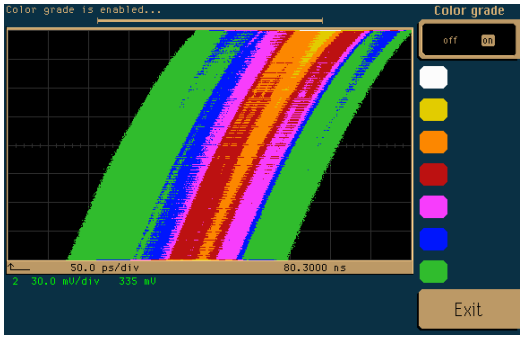

Figure 33: Clock-edge synchronizer output

3.7 Low Latency Clock Edge Synchronizer

The low-latency clock-edge synchronizer is an adaptation of the clock-edge synchronizer described above to multi-synchronous synchronization. We take advantage of the fact that most of the time the data arrival phase stays stationary. In Figure 34, the decision logic monitors the input and changes the selection setting only if there is a change in data arrival timing. Note that only two data phases are employed, as opposed to four in the original design. The resulting output waveform is shown in Figure 35, and is quite similar to that of Figure 33, validating that in multi-synchronous situations there is no need for the more complex circuit of Section 3.6. A similar synchronizer has been proposed in [10].

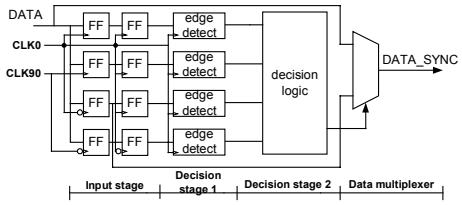

Figure 34: Low latency clock-edge synchronizer

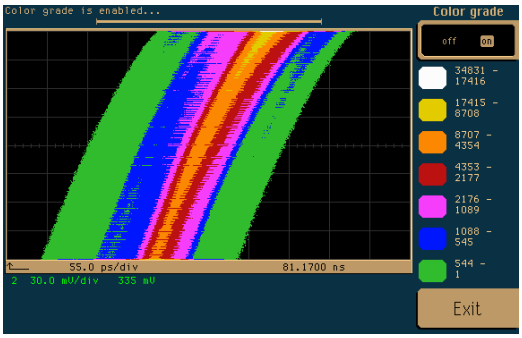

Figure 35: Low-latency clock-edge synchronizer output

4. Analysis

Table 1 shows hardware complexity (in terms of PLD logic elements), latency (in clock cycles), throughput and applicable modes of adaptation for the synchronizers and 32-bit data channels. All synchronizers are appropriate for multi-synchronous clock domains, while the two-flop one can also bridge asynchronous domains. In some cases, continuous adaptation can be obtained at the cost of either increased latency (Clock Edge and FIFO synchronizer) or increased hardware (Dual Data Delay synchronizer). The Clock Delay synchronizer is suitable for continuous operation without any added complexity.

Table 1: Comparing the synchronizers for 32-bit wide data channels

Synchronizer	HW (LE)	Latency (cycles)	Throughput	Adaptation Mode
Two-Flop	35	1-2	1/Latency	None
Data Delay	700	0-1	1	Training
Dual Data Delay	725	0-1	1	Continuous
Clock Delay	90	0-1	1	Continuous
FIFO (*)	135	0-1	1	Training
Clock Edge	676	5	1	Continuous
Low Latency Clock Edge	100	0-1	1	Training

(*) FIFO can adapt continuously at the cost of increased latency.

5. Summary

Actual measurements of a variety of synchronizers have been presented. We follow the methods of [1] and [2] for testing the metastability settling time constants of synchronization circuits. Failures of simple flip-flops has been demonstrated and measured. Subsequently, we analyzed a standard two-flop synchronizer as a base line, and investigated the characteristic settling of a number of plesiochronous synchronizers. All appear to perform successfully, providing a spectrum of area, power, latency and throughput trade offs to the designer.

All experiments were carried out on a standard programmable logic device, requiring a fully digital implementation and not allowing for any custom circuits or layout. Thus, the tested synchronizers are appropriate for FPGA and SoC designs based on standard cells. The two-flop synchronizer is appropriate for any asynchronous clock domain crossing, while the remaining synchronizers are suitable only for mesochronous, multi-synchronous or plesiochronous domain crossings.

We have also demonstrated the effects of multiple clocks on synchronization. Cross-talk between clocks of different frequencies has shown to result in increased jitter and calls for additional safety margins in the construction of successful synchronizers.

Additional research is required to further study synchronizers, such as identifying the time width of the danger zone, establishing methods for pre-silicon validation, optimizing the circuits, and verifying them on ASIC and full-custom VLSI.

Acknowledgement

We are grateful for the many helpful comments received from Charles Dike and the anonymous referees, and to the staff of the High Speed Digital Systems Lab at the Technion, who made these measurements possible.

References

[1] C. Dike and E. Burton, "Miller and Noise Effects in a Synchronizing Flip-Flop," *IEEE Journal of Solid-State Circuits,* **34**(6), pp. 849-855, 1999.

[2] J. Jex and C. Dike, "A fast resolving BiNMOS synchronizer for parallel processor interconnect,"-. *IEEE Journal of Solid-State circuits,* **30**(2), pp. 133-139, 1995.

[3] C. Foley, "Characterizing metastability," Proc. 2nd IEEE Symp. Adv. Res. Asynchronous Circuits and Systems, pp. 175-184, 1996.

[4] D.J. Kinniment, A. Bystrov, and A.V. Yakovlev, "Synchronization Circuit Performance," *IEEE*

Journal of Solid-State Circuits, **37**(2), pp. 202-209, 2002.

[5] R. Ginosar and R. Kol, "Adaptive Synchronization," 1998 IEEE International Conference on Computer Design (ICCD'98), Oct. 1998.

[6] D.G. Messerschmitt, Synchronization, in T.H. Meng, *Synchronization Design for Digital Systems,* Kluwer Academic Publishers, 1991.

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

[8] K. van Berkel, F. Huberts and A. Peeters, "Stretching Quasi Delay Insensitivity by Means of Extended Isochronic Forks," Proc. 2nd Working Conf., Asynchronous Design Methodologies, pp. 99-106, 1995.

[9] N. Sawyer, "Data to clock phase alignment," Xilinx application note XAPP255, v1.0 Sep 2000.

[10] F. Mu, C. Svensson, "Self-Tested Self-Synchronization Circuit For Mesochronous Clocking", IEEE Transactions on Circuits and Systems-II Analog and Digital Signal Processing, **48**(2), pp.129-140, 2001.

[11] A color version of this paper is under http://www.ee.technion.ac.il/~ran → publications.