Synchronization in Asynchronously Communicating Digital Systems

Priyadharshini Shanmugasundaram

Abstract – Two digital systems working in different clock domains require a protocol to communicate with each other in order to ensure validity of the data being shared between the two systems. Such systems are said to be asynchronous, and the protocol used for communication is said to synchronize the two systems. Asynchronous design techniques are not as widely used as synchronous design techniques are. This report points out the merits and demerits of asynchronous design techniques over synchronous techniques. It presents some of the commonly used asynchronous handshake protocols, and their implementation. Some common mistakes made in asynchronous design are also discussed.

I. Introduction

A synchronous circuit, as shown in Figure 1, consists of components that share a common and discrete notion of time, defined by a clock signal distributed throughout the circuit. In such circuits, the longest path in the combinational logic determines the minimum clock period. However, during a typical clock cycle, all the components in the circuit may reach their steady states well before the next clock signal. Also, each register dissipates energy during every clock cycle. If dynamic logic is used, the combinational logic also dissipates clock power during each clock cycle. Since the clock modulates the overall supply current, peaks in power-supply noise and in electromagnetic emission occur at the clock frequency leading to higher harmonics. Growing interconnect delays and the heterogeneous nature of systems-on-a-chip architecture place enormous constraints on the synchronous operation of components.

Meanwhile, synchronous design techniques are well established. Hence, the design effort is lesser for such circuits since the libraries and tools required for synchronous design are well developed. Also, tools for test vector generation are readily available making these circuits much easier to test than asynchronous circuits

Asynchronous circuits [1] are fundamentally different from synchronous circuits in the sense that the components in the circuit do not share a common and discrete time. Such circuits use handshaking between their components to perform synchronization, communication, and sequencing of operations. This gives asynchronous

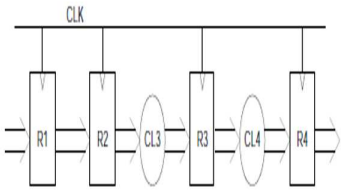

Fig. 1. A synchronous circuit.

design a number of advantages over synchronous design. Asynchronous circuits consume low power due to fine-grain clock gating and zero standby power consumption. They can operate at higher speeds since the operating speed is determined by actual local latencies rather than global worst-case latency. The emission of electromagnetic noise is less in these circuits since local clocks tend to tick at random points in time. They are robust towards variations in supply voltage, temperature, and fabrication process parameters since timing is based on matched delays. In fact, some of them can even be insensitive to circuit and wire delays as will be discussed in subsequent sections. They possess better composability and modularity because of the simple handshake interfaces and the local timing. These circuits do not face clock distribution and clock skew problems since there is no global signal that has to be distributed with minimal phase skew across the circuit.

However, the asynchronous control logic that implements the handshaking represents an overhead in terms of area, speed and power consumption. Also, there are a number of roadblocks in the design of such circuits, such as a lack of CAD tools and strategies, and a lack of tools for testing and test vector generation.

II. Basic Concepts

Synchronization is the process of enforcing an ordering of events on signals. For example, synchronization is required while sampling an asynchronous signal with a clock or when a signal that is synchronous to one clock is sampled by another clock. The difficulty of synchronizing a signal with a clock depends on the predictability of events on the signal relative to the clock. When a signal

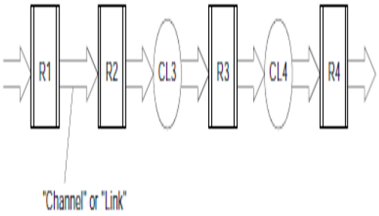

Fig. 2. An abstract data-flow view of the asynchronous circuit.

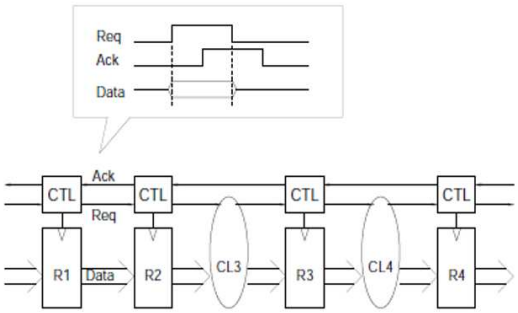

Fig. 3. A request-acknowledge based handshake.

is to be synchronized to a clock with the same frequency as the sample clock but with an arbitrary phase, the signal and clock are said to be mesochronous [2]. In such systems, a single phase measurement suffices to predict possible transition times arbitrarily far in the future. If the signal is generated by a clock with a slightly different frequency, there is a slow drift in the phase, and the signal and clock are said to be plesiochronous [2].

Figure 2 is an abstract representation of an asynchronous system. In an asynchronous circuit, the clock signal is replaced by some form of handshaking between neighboring registers. A simple request-acknowledge based handshake protocol is shown in Figure 3. The data and handshake signals connecting one register to the next can be thought of as a handshake channel or link. The data stored in the registers can be thought of as tokens tagged with data values that may be changed along the way as tokens flow through combinational circuits. The combinational circuits can be thought of as being transparent to the handshaking between registers. A combinational circuitry simply absorbs a token on each of its input links, performs its computation, and then emits a token on each of its output links.

Correct operation requires that data tokens flowing in the circuit do not disappear, that one token does not overtake another, and that new tokens do not appear out of nowhere.

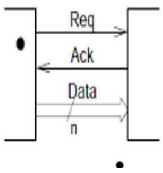

Fig. 4. A bundled-data channel.

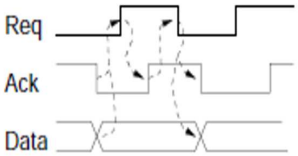

Fig. 5. A 4-phase bundled-data protocol.

III. Handshake Protocols

A. Bundled-data protocols

Bundled-data protocols comprise data signals that use normal Boolean levels to encode information, and have separate request and acknowledge wires bundled with the data signals, as shown in Figure 4. The bundled-data protocol is also known as the single-rail protocol.

4-Phase Bundled-Data Protocol

In this type of protocol, the request and acknowledge wires also use normal Boolean levels to encode information. The term 4-phase refers to the communication actions:

- the sender issues data and sets request high,
- the receiver absorbs the data and sets acknowledge high,
- the sender responds by taking request low, at which point the data is no longer guaranteed to be valid, and
- the receiver acknowledges this by taking acknowledge low. At this point the sender may initiate the next communication cycle.

Figure 5 illustrates the communication actions explained above.

This protocol is familiar to most digital designers, but it has a disadvantage in the superfluous return-to-zero transitions that cost unnecessary time and energy.

2-Phase Bundled-Data Protocol

In the 2-phase bundled-data protocol, the information on the request and acknowledge wires is encoded as signal

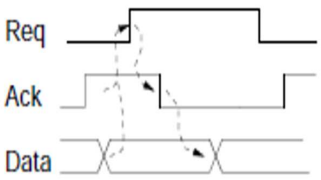

Fig. 6. A 2-phase bundled-data protocol.

transitions on the wires and there is no difference between a 0 to 1 and a 1 to 0 transition, they both represent a single event, as shown in Figure 6.

B. Dual-Rail Protocol

The request signal is encoded into the data signals using two wires per bit of information that has to be communicated. In other words, the dual rail protocol uses two request wires per bit of information d; one wire d.t is used for signaling a logic 1 (or true), and another wire d.f is used for signaling a logic 0 (or false). This protocol is very robust since two components can communicate reliably regardless of delays in the wires connecting the two components, or in other words, the protocol is delay-insensitive.

4-Phase Dual-Rail Protocol

The information is encoded as follows: {x.f,x.t} = {1,0} for a logic 0 and {x.f,x.t} = {0,1} for a logic 1 represent valid data and {x.f,x.t} = {0,0} represents no data. The codeword {x.f,x.t} = {1,1} is not used, and a transition from one valid codeword to another valid codeword is not allowed.

An abstract view of 4-phase dual-rail handshaking, as shown in Figure 7, can be explained as:

- the sender issues a valid codeword,
- the receiver absorbs the codeword and sets acknowledge high,
- the sender responds by issuing the empty codeword, and
- the receiver acknowledges this by taking acknowledge low. At this point the sender may initiate the next communication cycle.

A more abstract view of what is seen on a channel is a data stream of valid codewords separated by empty codewords.

The above protocol can be extended to an N-bit data channel. The codewords for an N-bit data channel can be divided into

- the empty codeword where all N wire pairs are {0,0}

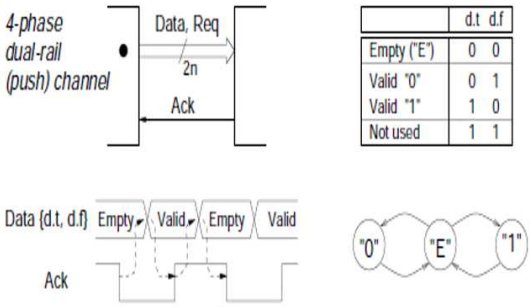

Fig. 7. A delay-insensitive channel using the 4-phase dual-rail protocol.

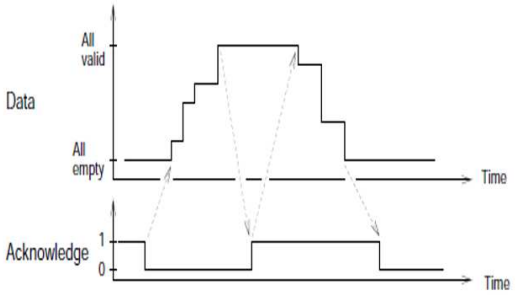

Fig. 8. Handshaking on a 4-phase dual-rail channel.

- the intermediate codewords where some wire-pairs assume the empty state and some wire pairs assume valid data
- the 2N different valid codewords

Figure 8 illustrates the handshaking on an N-bit channel: a receiver will see the empty codeword, a sequence of intermediate codewords, and eventually a valid codeword. After receiving and acknowledging the codeword, the receiver will see a sequence of intermediate codewords, and eventually the empty codeword to which the receiver responds by driving acknowledge low again.

2-Phase Dual-Rail Protocol

The 2-phase dual-rail protocol also uses 2 wires per bit, but the information is encoded as transitions. On an N-bit channel a new codeword is received when exactly one wire in each of the N wire pairs has made a transition. There is no empty value; a valid message is acknowledged and followed by another message that is acknowledged. Figure 9 shows a 2-phase dual-rail protocol.

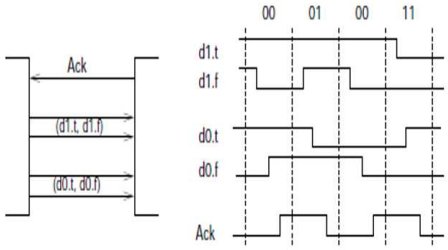

Fig. 9. Handshaking on a 2-phase dual-rail channel.

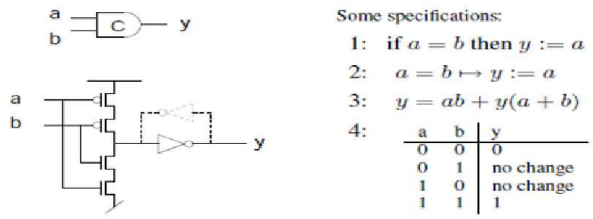

Fig. 10. The Muller C-element.

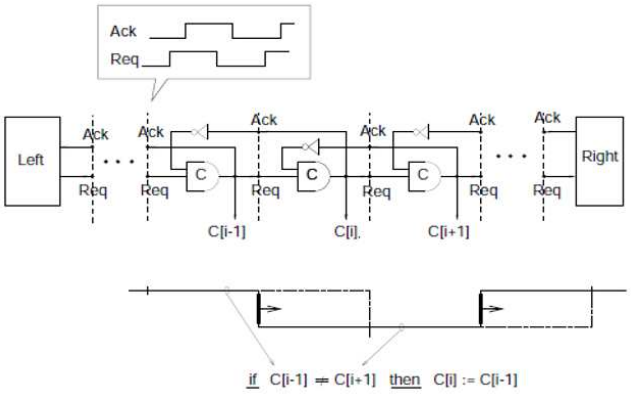

Fig. 11. The Muller pipeline.

C. Other Protocols

The previous sections introduced the four most common channel protocols, but there are other possibilities as well. The two wires per bit used in the dual-rail protocol can be seen as a one-hot encoding of that bit and often it is useful to extend to 1-to-n encodings in control logic and higher-radix data encodings.

If the focus is on communication rather than computation, m-of-n encodings may be of relevance. The solution space can be expressed as the cross product of a number of options including: {2-phase, 4-phase} X {bundled-data, dual-rail,1-of-n,} X {push, pull}

The choice of protocol affects the circuit implementation characteristics such as area, speed, power, robustness, etc.

IV. The Muller C-element and The Indication Principle

In asynchronous circuits, signals are required to be valid all the time, every signal transition has a meaning and, consequently, hazards and races must be avoided. The concept of indication or acknowledgement plays an important role in the design of such circuits.

For instance, consider a 2-input OR gate. An output change from 1 to 0 leads to a conclusion that both inputs are now at 0. However, an output change from 0 to 1 indicates that at least one input is 1, but does not indicate which. In other words, the OR gate only indicates or acknowledges when both inputs are 0. Similarly an AND gate indicates only when both inputs are 1.

Signal transitions that are not indicated or acknowledged in other signal transitions are the source of hazards and should be avoided.

A circuit that is better in this respect is the Muller C-element shown in Figure 10. It is a state-holding element much like an asynchronous set-reset latch. When both inputs are 0 the output is set to 0, and when both inputs are 1 the output is set to 1. For other input combinations the output does not change. Hence, an output change from 0 to 1 indicates that both inputs are now at 1; and similarly, an output change from 1 to 0 indicates that both inputs are now at 0. The Muller C-element is a fundamental component used extensively in asynchronous circuits because of this property.

V. The Muller Pipeline

Figure 11 shows a circuit built from C-elements and inverters. The circuit is known as a Muller pipeline or a Muller distributor. Variations and extensions of this circuit form the backbone of almost all asynchronous circuits.

The Muller pipeline is a mechanism that relays handshakes. After all the C-elements have been initialized to 0 the left environment may start handshaking. The i th C-element C[i] will propagate a 1 from its predecessor, C[i-1], only if its successor, C[i+1], is 0. Similarly, it will propagate a 0 from its predecessor only if its successor is 1.

On any interface between C-element pipeline stages, correct handshaking will be observed, but the timing may differ from the timing of the handshaking on the left hand environment. Eventually the first handshake (request) injected by the left hand environment will reach the right

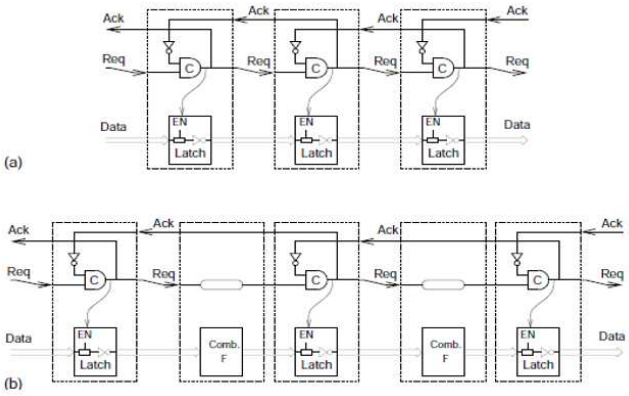

Fig. 12. A 4-phase bundled-data pipeline.

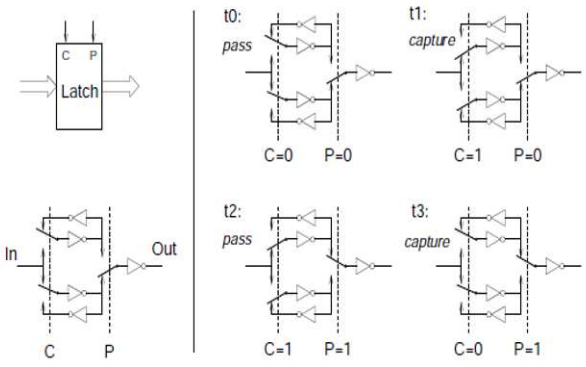

Fig. 13. Latch in 2-phase bundled-data pipeline.

hand environment. If the right hand environment does not respond to the handshake, the pipeline will eventually fill. If this happens the pipeline will stop handshaking with the left hand environment the Muller pipeline behaves like a ripple through FIFO.

The implementation of the Muller pipeline is the same for both 2-phase and 4-phase handshaking. The difference lies in the way the signals are interpreted and in the way the circuit is used. Also, the circuit operates equally well from right to left. If the definition of signal polarities is reversed and if the role of the request and acknowledge signals are reversed, the circuit can be operated from right to left. The circuit works correctly regardless of delays in gates and wires and hence, the Muller-pipeline is delay-insensitive.

VI. Circuit Implementation

The choice of handshake protocol affects the circuit implementation in terms of area, speed, power, robustness, etc. Most practical circuits use one of the following protocols: 4-phase bundled-data, 2-phase bundled-data or 4-phase dual-rail. The circuit implementations of these protocols use variations of the Muller pipeline for controlling the storage elements.

A. 4-Phase Bundled-Data

A Muller pipeline is used to generate local clock pulses. The clock pulse generated in one stage overlaps with the pulses generated in the neighboring stages in a carefully controlled interlocked manner. Figure 12(a) shows a FIFO, i.e. a pipeline without data processing, and Figure 12(b) shows how combinational circuits (also called functional blocks) can be added between the latches. To maintain correct behavior matching delays have to be inserted in the request signal paths.

This circuit may be viewed as a traditional synchronous data-path, consisting of latches and combinational circuits that are clocked by a distributed gated-clock driver or as an asynchronous data-flow structure composed of two types of handshake components: latches and functional blocks.

The pipeline implementation shown in Figure 12 is simple but it has some drawbacks: when it fills, the state of the C-elements is (0, 1, 0, 1, etc.), and hence, only every other latch is storing data. This is not any worse than in a synchronous circuit using master-slave flip-flops, but it is possible to design asynchronous pipelines and FIFOs that are better in this respect. Also, the throughput of a pipeline or FIFO depends on the time it takes to complete a handshake cycle and for this implementation, this involves communication with both neighbors. This leads to a lower speed.

B. 2-Phase Bundled-Data (Micropipelines)

A 2-phase bundled-data pipeline also uses a Muller pipeline as the backbone control circuit, but the control signals are interpreted as events or transitions. Therefore, special capture-pass latches are needed: events on the C and P inputs alternate, causing the latch to alternate between capture mode and pass mode. This requires a special latch design as shown in Figure 13. The switch symbol in Figure 13 is a multiplexer, and the event controlled latch can be viewed as two ordinary level sensitive latches operating in an alternating fashion followed by a multiplexer and a buffer.

Figure 14 shows a pipeline without data processing. Combinational circuits with matching delay elements can be inserted between latches in a way similar to the 4-phase bundled-data approach.

The 2-phase bundled-data approach is elegant and efficient compared to the 4-phase bundled-data approach

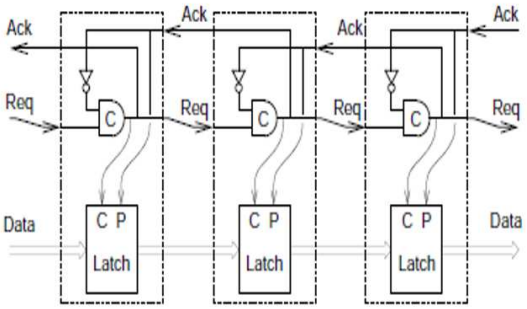

Fig. 14. A 2-phase bundled-data pipeline.

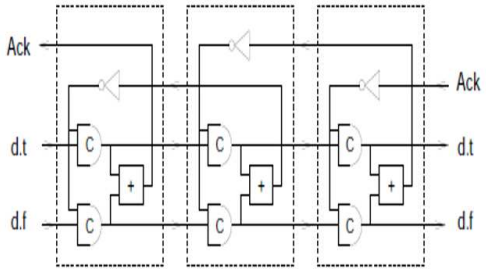

Fig. 15. A simple 3-stage 1-bit wide 4-phase dual-rail pipeline.

that incurs unnecessary power and performance loss due to the return-to-zero part of the handshaking. However, the implementation of components that respond to signal transitions is often more complex than the implementation of components that respond to normal level signals.

C. 4-Phase Dual-Rail

A 4-phase dual-rail pipeline is also based on the Muller pipeline, but in a more elaborate way that has to do with the combined encoding of data and request. Figure 15 shows the implementation of a 1-bit wide and three stage deep pipeline without data processing. It can be viewed as two Muller pipelines connected in parallel, using a common acknowledge signal per stage to synchronize operation. The pair of C-elements in a pipeline stage can store the empty codeword {d.t, d.f} = {0, 0}, causing the acknowledge signal out of that stage to be 0, or it can store one of the two valid codewords {0,1} and {1, 0}, causing the acknowledge signal out of that stage to be logic 1. The codeword {1, 1} is illegal and does not occur. The acknowledge signal generated by the OR gate safely indicates the state of the pipeline stage as being valid or empty.

An N-bit wide pipeline can be implemented by using a number of 1-bit pipelines in parallel. This does not guarantee to a receiver that all bits in a word arrive at the same time, but often the necessary synchronization is done in the function blocks.

VII. Scan Chain with Mixed Edge-Triggered Flip-Flops

Scan testing is a widely used technique in the test of sequential circuits. In this technique, the memory elements in the circuit are linked together to act as shift registers during testing and to function as memory elements during normal operation. This is done to increase the controllability and observability of the circuit during testing.

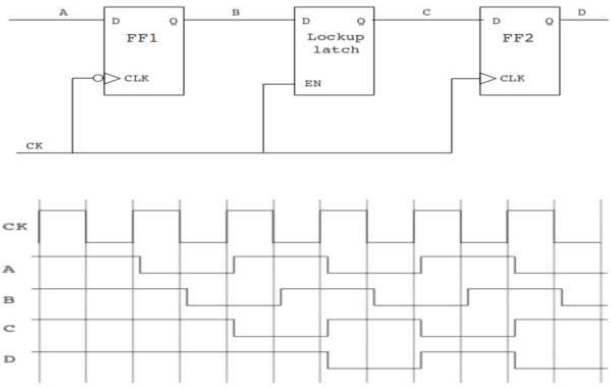

Fig. 16. Lockup latch in a scan chain.

Clock skew between successive scan-storage cells must be less than the propagation delay between the scan output of the first storage cell and the scan input of the next storage cell. Otherwise, data that latches into the first scan cell also latches into the second scan cell. This results in an error because the second scan cell should latch the first scan cell's old data rather than its new data.

Thus, when a circuit consists of both positive and negative edge-triggered flip-flops, timing problems occur. In this case, all the negative edge-triggered flip-flops are linked together to form one scan chain and likewise all the positive edge triggered flip-flops are linked together to form a second scan chain. These two scan chain are linked together to form a single scan chain through a lockup latch.

Lockup latches [3] are nothing more than transparent latches. They are used to connect two scan-storage elements in a scan chain in which excessive clock skew exists. Figure 16 illustrates the use of lockup latches. The circuit contains two flip-flops. Flip-flop 1 represents the end of the scan chain that consists of negative edge-triggered flip-flops. Flip-flop 2 represents the beginning of the scan chain that contains positive edge-triggered

flip-flops. The latch has an active-high enable, which becomes transparent only when the clock to the negative-edge triggered flip-flops goes high and effectively adds a half clock of hold time to the output of flip-flop 1. Thus, the two scan chains are synchronized by the addition of a simple latch.

VIII. Common Mistakes During Synchronization

This section reviews some common causes of errors in circuits employing more than one time domain [4].

A. Avoiding the Synchronizer

The most common synchronization error is the transfer of a signal from one clock domain into another without any synchronization since in some cases, the designer might feel that failure probability is too low to worry over. Also, in some cases, if the receiver operates at a much higher clock frequency than the sender, the receiver is expected to always be fast enough to catch the signal and hence synchronization seems unnecessary.

If the incoming data is used as a combinational input to a combinational circuit, which eventually feeds into a flip-flop, there is no way to guarantee the timing of the output of the combinational circuit since the timing of the input is unknown. In particular, it may change simultaneously with the sampling edge of the clock, and the receiving flip-flop may enter metastability or take excessively long to respond, hampering correct operation of the next stage of logic. This error can sometimes evade detection by normal logic validation tools.

B. Sneaky Path

Occasionally, a signal sneaks through a clock domain boundary unintentionally and unsynchronized. For instance, a signal is sometimes moved from one clock domain to another as part of redesign, and some uses of the signal in its old domain are overlooked.

C. Wrong Protocol

Consider the following example. The sender is a CPU that can be tuned to operate in the range of 60-100 MHz. The receiver is a communication modem based on a 55 MHz clock. A 2-phase bundled-data protocol is used to transfer data from the CPU to the modem. It is found that it would take four cycles of the receiver's clock to latch the data. Based on the relative speeds, this would mean up to eight cycles of the faster sender's clock. To save time and logic, the designer eliminates the synchronizer and inserts a nine-cycle delay in the sender's FSM.

There would be two problems with this design. First, the safety requirement of the protocol that transitions must be acknowledged is violated. Although the data would be safely latched, at times the receiver might be busy doing something else and would not manage to make use of the data before a new set of data arrives, overwriting the old.

Second, while the modem would remain at 55 MHz, if the CPU were to be sped up in a later chip generation, the sender's nine clock cycles will not be sufficient to cover the four modem cycles anymore.

D. Global Reset

In a multi-frequency GALS (Globally Asynchronous, Locally Synchronous) SOC, a global reset signal is naturally asynchronous to at least some of the clock domains. The leading edge of the reset signal is harmless, as it forces all circuits to a known starting state. The trailing edge, on the other hand, could cause some damage. During global reset, the various clocks are started and all PLLs settle into their respective different frequencies. When the reset is removed, it can happen simultaneously with the sampling edge of one of the clocks. The global reset is typically connected into the asynchronous clear input of many flip-flops, and its trailing edge must respect a setup constraint, or else the flip-flops may enter metastability. This is true for other asynchronous signals such as the asynchronous clear or preset of flip-flops as well.

E. DFT Leakage

Simple production testers may have only a single clock. To test a GALS SOC on such testers, all clocks are shorted together. Static faults and some dynamic faults are properly tested that way. The clock shorts of course must be ignored during path analysis, but certain changes of the design may result in an error (sneaky) path masked by the list.

IX. Conclusion

This report was aimed at giving the reader a basic idea of how to design simple asynchronous circuits. The properties of asynchronous circuits and their advantages and disadvantages over synchronous circuits were discussed. Different handshake mechanisms used in asynchronous circuits and their implementations were explained. The Muller C-element, an important component used in the implementation of these handshake techniques, was analyzed. A typical application concerning scan chains was

explained to point out that asynchronous components appear even in normal synchronous circuits. Also, the common mistakes committed during asynchronous design were discussed.

References

[1] Asynchronous Circuit Design A Tutorial by Jens Sparso obtained from http://www.pdfee.com/asynchronous-circuit-design-a-tutorial.html

[2] Chapter 10 in Digital Systems Engineering by William J. Dally, John W. Poulton obtained from http://books.google.com/

[3] 10 Tips for Successful Scan Design: Part One by Ken Jaramillo and Subbu Meiyappan obtained from http://www.edn.com/article/CA46603.html

[4] Fourteen Ways to Fool your Synchronizer by Ginosar, R. from Asynchronous Circuits and Systems, 2003. Proceedings. Ninth International Sumposium, May 2003, pp. 89-96