

Low Latency Clock Domain Transfer for Simultaneously Mesochronous, Plesiochronous and Heterochronous Interfaces

Wade L. Williams, Philip E. Madrid, and Scott C. Johnson

Advanced Micro Devices

Email: wade.williams@amd.com

phil.madrid@amd.com

scottc.johnson@amd.com

Abstract

Microprocessors are employing higher levels of system integration for both higher performance and lower system cost. In doing so, problems that used to be apparent inter-device are now found intra-processor. The integration of the processor core with other units such as the north bridge often increase the number of clock domains within the device. In addition, the frequency of the external interfaces has increased at a much higher rate than the processor frequency. This trend will continue with the advent of multi-core processors which have increasing bandwidth demands. However, as the characteristics of the clock domains become more complex, the problem compounds the burden on the clock domain transfer mechanism to achieve low latency. Presented here is an easily implementable, low latency solution for clock domain transfer in the presence of high frequency mesochronous, plesiochronous, and heterochronous clock signaling.

1. Introduction

Clock domains either within a computer system or within a device itself are often clocked uniquely in fundamental ways. The domains may be running at different frequencies and clocked by different PLLs. The PLLs may run off reference clocks produced by different clock generators and different crystal oscillators. For interfaces on the same device, clock grid skew and PLL phase error differences can be greater than a processor clock period, thus discounting simpler techniques for domain crossing. These differences imply asynchronous or pseudo-synchronous interfaces between the two clock domains.

In the past the communication between devices in a microprocessor system was fully synchronous. Within the microprocessor itself clock domain crossings occurred only at the external interfaces of the chip. Since the interface was fully synchronous, the transfer was not particularly difficult, especially if the frequencies of the two sides of the interface were convenient multiples of each other. Later on the communication between these devices became source synchronous and new methods of clock domain crossing were employed. These interfaces still existed primarily on the external boundaries of the devices. Subsequent generations of high end microprocessor based systems will rely on even higher speed communication protocols such as SERDES (serialize-deserialize) style interfaces. This is driven by the bandwidth requirements of multi-core processors implying the need for high bandwidth external pin interfaces.

During the progression from fully synchronous external interfaces, there arose an increasing level of integration of system functions on the processor itself. An example is the adoption of chipset functions such as the memory controller on recent Advanced Micro Devices processors. This integration pushed the problem of clock domain transfers from the periphery into the processor itself. This is further impacted by modular System on a Chip methodologies which are architected to mix and match functional units with each other. These functional units are often designed independently, differing in their initial design constraints and complicating their integration into a single chip.

So we see two types of interfaces that require clock domain transfers. One is the external interface which had been slower than the processor clock frequency, then approximated it and soon will exceed it. The second are interfaces within the processor itself where different functional units communicate with

Apple Exhibit 1008

Apple Inc. v. Redstone Logics LLC

IPR2025-01532, Page 1 of 9

each other while running in different clock domains. In both these cases the need for low latency domain transfers is vital. The penalty to cross domains must be small as measured against the processor clock period. This is imperative so that the performance and cost benefit of integrating other functional units on the processor is not offset by high latency inter-unit transfers and also so that the high bandwidth external interfaces are not saddled by high latency. Ultimately the design challenge is to ensure low latency inter-device and intra-processor clock domain transfers.

There are several electrical sources of clock mismatch introduced across clock domains. A typical block diagram of a system using FIFO data transfer is shown in figure 1.

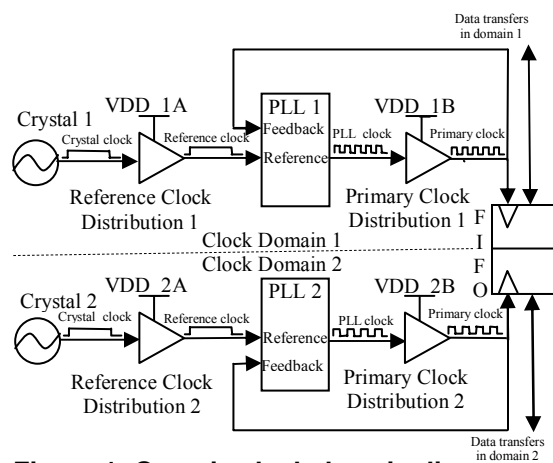


Figure 1: Generic clock domain diagram

This paper describes a clock domain transfer solution for interfaces of the types just described which have several defining characteristics. One is that the two clock domains can be skewed from each other. This type of interface is called mesochronous where the two domains run at the same nominal frequency but with an unknown phase relationship [1]. This skew derives from a couple different sources. One is simply due to a difference in insertion delays between the two independent clock grids. Another source may be due to the accumulated phase error between independent PLL sources, each one clocking one of the two domains. This phase error and skew is unknown though bounded.

A general characteristic of PLL phase error is that the phase changes very slowly. An on chip microprocessor PLL will have tracking bandwidths that are in the low megahertz range. For a microprocessor running in the gigahertz range, the PLL will only be able to change the clock phase at a slow rate dependent on its bandwidth and phase gain. The rate of change will typically be picoseconds per PLL

clock cycle. There can also be high frequency jitter caused by PLL power supply noise, but this is generally very small since PLL power supplies are heavily filtered.

A second interface characteristic for which a solution is proposed is called plesiochronous. In such an interface the two clock domains are nominally running at the same frequency but there may be constant drift between the two domains [1]. This drift is often caused by the two clock domains being sourced ultimately from different crystal oscillators. This drift may occur continuously over time, quite unlike mesochronous interfaces.

A fully synchronous FIFO design is not possible using two different crystals. When using an asynchronous FIFO, the difference in frequency must be taken into account in designing the dynamic separation. Spread spectrum techniques used to reduce radio emissions can also cause additional FIFO uncertainty.

Finally, the interfaces may be running at completely different nominal frequencies. This is referred to as heterochronous [1]. The implementation presented here covers frequencies which are different from each other but are fundamentally related by a ratio of integers. Additionally, either or both domains may change frequency from their nominal frequency by a power of two divisor. This implies the need for an interface that can operate at frequencies other than very simple ratio of integer relationships.

This paper thus presents an implementation for a low latency interface which transfers signals across domains which may be simultaneously mesochronous, plesiochronous and also a subset of heterochronous, running at different and somewhat arbitrary (but known) nominal multi-GHz clock frequencies.

1.1. Additional Sources of Uncertainty

Another source of clock delay mismatch is caused by the reference clock networks feeding the two clock domain PLLs. The reference clock network can easily have nanoseconds of delay. Systematic delay differences, process skew, temperature, and voltage offsets will cause delay differences on the two sides of the FIFO. Process skew will show up as delay uncertainty that is fixed but unchanging with time. Temperature differences will show up as slow changes of delay over microseconds. Voltage changes can cause fast fluctuations in delay. The maximum fluctuations occur at package resonance which typically occurs in the tens to hundreds of Megahertz range. However, the PLL will filter out fast delay changes in the reference clock distribution above the

PLL bandwidth. In addition, the delay fluctuation will be reduced if the reference clock network uniformly uses the same power supply.

The last and largest source of clock delay mismatch is the high speed primary clock distributions of the two FIFO clock domains. The primary clock networks tend to have large delays in them that are sensitive to process skew, temperature, and voltage offsets. There is usually a large clock delay difference between the two sides of the FIFO because the clock distributions are usually quite different in size. The primary clock distributions have the same sensitivity to process, skew, temperature, and voltage differences as the reference clock distribution. However, if the high speed clock distribution is in the PLL feedback loop, then the raw delay, process variation, and temperature offset will be largely cancelled.

Voltage differences in the clock tree cause the largest potential source of FIFO skew. Just like reference clock skew, the high speed clock distributions will see the maximum amount of delay change at package resonance (tens to hundreds of Megahertz). In some cases the primary clock distributions will be on a common power supply, so the delay differences will be reduced. However, in many cases the high speed clock distributions will be connected to different power supplies for system flexibility and power optimization. If different power supplies are used, the delay differences between clock domains can change rapidly. It is difficult for the FIFO to track fast delay changes. In this case, the FIFO depth may need to be increased to account for the fast untrackable delay changes.

2. Prior Work

There have been numerous solutions which simply solve domain crossing problems which are most often a subset of the problems presented here. One example is using synchronization chains to force the domain transfer with high reliability. Pipelined synchronization achieves the same while preserving bandwidth [2]. Often latency is not the primary consideration and so most of these solutions are not practical for modern high speed microprocessors. In other cases they do not solve all three of the problems presented here simultaneously.

In recent years, there have been significant efforts at solving this problem. Advances to the original STARI (Self-Timed At Receiver's Input) technique have been presented [3]. This technique achieves low latency transfers through the use of self-timed circuitry. This circuitry requires more care and characterization than the solution presented here with

non custom logic. In addition, the modified STARI solution will not scale well with very high speed clocks where it becomes increasingly difficult to design and time to ensure the proper generation of the clock phase produced by the latch controller.

Another solution was proposed which employs a FIFO using signals which indicate whether the FIFO is full or empty [4-5]. Variations on this basic idea of flow controlling transfers by monitoring the fullness or emptiness of the FIFO are presented which are reliable and protect against FIFO overflow or underflow. This solution has increased latency and reduced bandwidth anytime the FIFO is either empty or full. Admittedly this approach solves a problem which does not exist in our problem statement. Namely we know a priori the nominal frequency difference between the clock domains which the empty/full design does not require. Nevertheless, the FIFO empty/full approach has latency which can not be tolerated within the microprocessor and on the external interfaces when the nominal frequencies are known. This latency is not inherently required by the empty/full approach but exists when it strives to solve the problem generally (without knowing the nominal frequency difference).

The approach described here is one which strives to maintain very low latency when measured against a high frequency multi-GHz clock period and is also modular, frequency scalable, easily characterizable, and implemented efficiently with standard cell logic.

3. Description of the Solution

This design creates a low latency, high-speed interface through the integration of three major components: a data rate matching FIFO, a pointer tracking circuit, and a digital filter. All three are required to achieve the best performance and only the FIFO is capable of operating autonomously. The solution has no self-timed circuits and, if area is not a primary design constraint, the logic can be implemented without any custom circuits.

3.1 Data Rate Synchronization FIFO

The first piece of the solution is a FIFO of a given depth, implemented as a circular queue, with read and write pointers designed to advance at the nominal data rate of the interface. Since this design is intended to synchronize data from one clock domain to another, the read and write pointers are expected to exist on different clock grids, with the FIFO acting as a buffer between the two domains. With both pointers running at the same data rate, we eliminate the need for tracking whether the FIFO is full or empty as there

should never be an overflow or underflow of data. In mesochronous systems, no additional logic is required to ensure the FIFO pointers are running at similar frequencies since both clocks will be derived from the same reference.

For heterochronous systems whose clocks are ratios of one another, a throttling circuit is required to reduce the frequency of the faster clock and ensure both pointers are running at the same average data rate. Although different in frequency, the two clock domains must have a known integer relationship to facilitate the design of a counter based clock rate matching solution [3]. Additional clock ratios, created based on power of 2 divisors of these relationships, can be supported by multiplying the base ratio by the largest supported divisor. For example a system, supporting a low power mode where one clock is divided by 16, can transform a simple 2:3 ratio into 1:24 or 32:3. The architecture proposed here does not require the throttled clock to have any specific duty cycle. However, the run-time transition density of both domains needs to be matched as closely as possible. A sample pattern for a 2:3 clock ratio can be seen in Figure 2: Example of a Throttled Clock in a Heterochronous Interface. The throttled clock can be seen to have a modified duty cycle, but its average clock period is identical to the slow clock it is attempting to mimic.

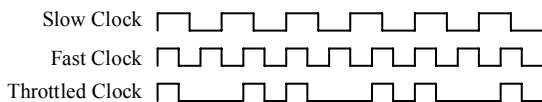


Figure 2: Example of a Throttled Clock in a Heterochronous Interface

If the two clocks are sourced from different references, creating a plesiochronous situation, the allowable frequency mismatch between the two domains is limited by the tracking response time of the final design implementation. The tracking logic is further described in section 3.2. In all clocking topologies, any differences between the read pointer and write pointer clock rates must be controlled to ensure they do not exceed the tracking bandwidth of the final design.

For all clock topologies except the plesiochronous interface, a FIFO with pointers operating far enough apart to absorb PLL wander and uncertainty at pointer initialization is an adequate solution for non-latency sensitive designs. This fact allows us to create a simple test mode for manufacturing and production by ensuring the final design has a FIFO large enough to operate without dynamically adjusting the pointers.

Thus the final design can be made to operate reliably and deterministically on a tester by placing the pointers further apart and disabling the asynchronous aspects of the system. The initial pointer placement is programmable, and it is still possible to reduce latency without using the tracking logic by adjusting the pointer separation based on the characterized clock deviations expected during operation.

3.2 Pointer Tracking Circuit

Critical to reducing the latency is the pointer tracking logic. Since the maximum possible latency is directly related to the FIFO depth, we can reduce this impact on performance through active monitoring and adjustment of the pointer separation. By minimizing the number of unread entries residing in the FIFO, we have essentially made the FIFO appear smaller and reduced the latency. Comprehended in our design is the fact that we are operating at frequencies where the absolute drift of the PLL is greater than the clock period. For the plesiochronous case this drift is unbounded. While drift over multiple clock periods is actively tracked by this logic, we also know that by definition the rate of drift is tightly controlled within each synchronous domain to minimize clock jitter. The fact that the clock drift is slow is a critical component to the design of our tracking logic as it greatly relaxes the response time requirement of the system and also permits us to remove the latency of the tracking circuit from the data path.

If we define any instant where the read pointer and the write pointer are equal as a fatal condition for this interface, then the theoretical lowest average latency for the base design is 2 data-rate clocks (i.e. FIFO entries). This limitation is based on the fact that we must detect any pointer separation deviations before they can be corrected. This requirement implies that operating the pointers with a desired separation of one FIFO entry would result in a pointer collision before the situation could be avoided. How closely the final design achieves this theoretical limit depends heavily on the filtering logic described in section 3.3. A further refinement is offered in section 3.3.3.

In order to track the pointer separation, the location of the pointer in one domain must be synchronized into the other domain and evaluated. Because the clocks are known to drift very slowly relative to the operating frequency, several simplifications are immediately possible. First there is no need to evaluate the pointer separation on every clock since clock-to-clock variation is negligible. Second we can choose to evaluate pointers at a convenient time to remove ambiguity as they wrap

around the FIFO structure. Finally we can treat pointer information, which is delayed while being locally synchronized, as the current state of the pointer in the other domain. Although the pointers will continue to drift while signals are being captured and the separation is being calculated, this uncertainty is negligible within the context of the whole system. This fact allows us to hide the latency of the synchronizer chain from the data path effectively eliminating the latency associated with using the empty/full handshaking typically found in FIFO interfaces.

Since the pointers advance at the data rate of the interface, which in an optimized design is exactly the same frequency as the slower clock domain, the most convenient signal for pointer tracking is the most significant bit (MSB) of the pointer itself. For any FIFO larger than two entries, using the MSB ensures we will safely capture the signal through a simple synchronizer chain of flops in the other clock domain since the MSB is guaranteed to toggle at a rate slower than the clocks controlling the FIFO pointers. By detecting the falling edge of the MSB, we have a clear indication of when the pointer has wrapped to entry 0 of the FIFO. If the synchronizer chain is built with a series of flops clocked at the data rate of the interface, the pointer location information can be reasonably preserved. One merely needs to include the number of synchronization flops when estimating the pointer's location in the other domain. We have now leveraged the aforementioned simplifications and created an interface using a single wire that conveys when the pointer in the other clock domain is at zero. This event occurs every time it wraps around the FIFO structure making the FIFO size a direct component in the sampling rate of the design. Uncertainty due to metastability in the first synchronizer stages is covered further in section 3.3.1.

The pointer tracking logic is designed to maintain the pointers at a specific, user programmable separation. Tracking accuracy is a function of ratios between the clock domains, the digital filter, and the pointer sampling rate. This programmability allows the performance to be adjusted based on the final application. Because the design does not automatically attempt to drive the interface to an absolute minimum latency, the risk of a complete operational failure is minimized. If the design is failing in a particular configuration, the pointer separation can be increased to achieve functional operation.

Relevant equations for this logic are the following:

F = Number of FIFO entries

S = Desired/Programmed separation

E = Expected local pointer location

A = Actual local pointer location

Δ = Pointer comparison result

If the pointer in the other clock domain is always evaluated when its value is zero then the following equations hold true for the local clock domain:

If the local domain is the read pointer:

$$E = F - S \text{ and } \Delta = A - E.$$

If the local domain is the write pointer:

$$E = S \text{ and } \Delta = E - A.$$

In both implementations, if Δ is greater than zero the pointers are too close together and if Δ is negative the pointers are too far apart. No adjustment is needed when $\Delta = 0$.

An example of this logic implemented in the read pointer domain can be seen in Figure 3a: Tracking Logic in the Read Pointer Domain. Waveforms for the logic detecting the pointer too far away are shown in Figure 3b: Timing Diagram Detecting Pointers Are Drifting Apart. The WrPtr[2] synchronization chain has been deliberately matched to the FIFO depth in order to simplify the logic. In this example $E=6$ and, when the Eval signal asserts, $A=5$. Thus $\Delta = 5 - 6 = -1$ and the pointers are detected as being too far apart. This information is then sent to the clock rate matching logic to generate an extra clock transition on the throttled clock effectively reducing the running pointer separation by one entry.

Design Assumptions:
8 entry FIFO
Fast Clock : Slow Clock = 3:2
Programmed Pointer Separation $\Delta = 2$
Write Pointer is on the Slow Clock

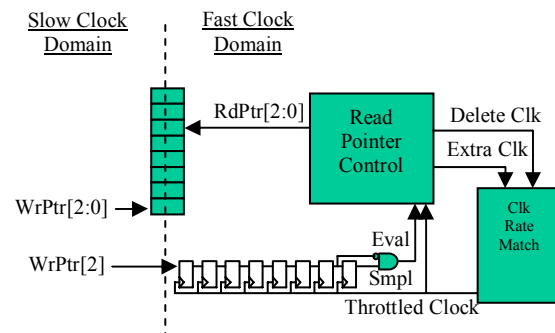


Figure 3a: Tracking Logic in the RdPtr Domain

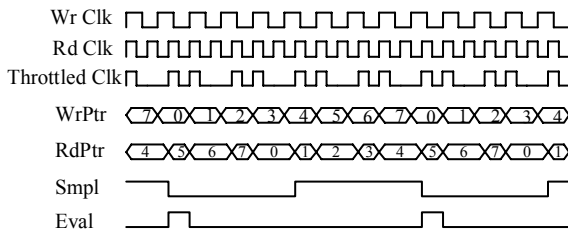


Figure 3b: Timing Diagram Detecting Pointers Are Drifting Apart

To optimize the performance of the design, all calculations must be done in the faster clock domain. It is desirable to essentially free-run the pointer in the slow domain thus maximizing the throughput of the interface. If both clock regions are the same nominal frequency, the logic will need to be mirrored and operated concurrently in both domains. This behavior is required since no mechanism is available to speed up the interface when both domains are matched in frequency and the tracking circuits will need to delay either the read or write pointer when making adjustments. A table depicting these various responses is shown:

Table 1: Pointer Adjustment Response Behavior

One Clock Is Nominally Faster Than Another	
Ptrs Too Close ($\Delta > 0$)	Suppress one Fast Clock
Ptrs Too Far ($\Delta < 0$)	Allow one extra Fast Clock
Neither Clock Is Nominally Faster	
Ptrs Too Close ($\Delta > 0$)	Suppress one clock on the Write Pointer
Ptrs Too Far ($\Delta < 0$)	Suppress one clock on the Read Pointer

One side benefit of the tracking logic is the facilitation of the initial pointer placement. It is extremely difficult to precisely place the read and write pointers during initialization. With the tracking logic activated, the initial pointer placement can be conservative. The design can rely on the tracking logic to safely bring the pointers closer together during operation without any risk of pointer collision.

3.3 Digital Filter

Because the tracking logic is susceptible to metastability on the synchronizer chain, and the data rate matching circuit may produce non-uniform clock patterns, it is necessary to add some form of digital filtering to the pointer tracking information. One example of filtering would be making an adjustment only if in m samples, there were n detections of the pointers too far, (or conversely too close) where n is an integer greater than $m/2$. This filtering helps the logic to avoid overreacting based on these raw sampling inaccuracies. Filter algorithms can also be biased, allowing the design to respond more aggressively to critical situations and more conservatively elsewhere. As with any closed loop system, improving the system response time comes at a direct cost to the tracking accuracy. Through inspection, it is readily apparent that as more samples are required to determine the pointer location, more clocks are required for each tracking adjustment.

3.3.1 Sampling Uncertainties

Uncertainty in the tracking logic arises from metastability on the synchronizer chain and non-uniformity of the throttled clock. Metastability errors may occur when the data being sampled changes within the setup and hold window of the capturing flop. By design any missed event is guaranteed to be captured on the very next clock, thus limiting the amount of uncertainty which can be injected into the system due to metastability. Because we are sampling at the data rate, this translates to one FIFO entry of uncertainty due to metastability.

The other main contributor to uncertainty is the irregular duty cycle of the throttled clock. In Figure 4: Uncertainty Due To Sample Jitter one can clearly see the variations in timing which occur as data is launched and captured off different clock edges. This inherent uncertainty, due to sample jitter introduced by the throttled clock, is illustrated in Case I. Even in the absence of clock drift, the capture of data between L1 and C1 has significantly different timing than data transferred between L2 and C2 due to variations in the throttled clock timing. Case II and Case III show additional sample variations which occur as the two clocks change their phase relationship over time. As the slow clock drifts earlier with respect to the throttled clock, data launched at L2 will eventually be captured at C3. Likewise if the slow clock drifts later in time relative to the throttled clock, the data launched at L1 will eventually miss timing for the C1 edge and be captured at C3.

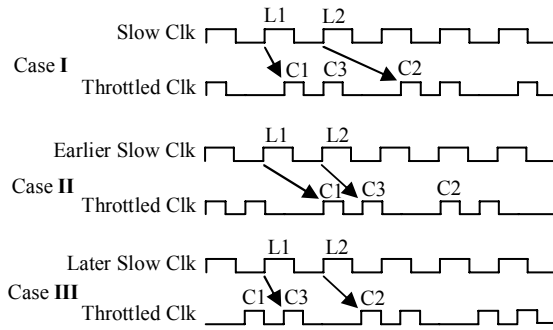


Figure 4: Uncertainty Due to Sample Jitter

The uncertainty due to sample jitter created by throttling the fast clock can introduce additional tracking inaccuracies since it is not random, but based off a rate matching counter. If the periodic nature of the throttled clock is a common multiple of the FIFO depth, the samples may consistently have a skewed timing. Ideally one should strive to make the throttled clock and the sample rate such that samples can occur across all timing combinations to eliminate this artificial biasing of the data.

One final source of inaccuracy occurs when the two clock domains are operating at close, but distinctly different frequencies. If the differences in frequency are significantly smaller than the tracking response of the system, then the window over which the pointers are evaluated can affect the outcome. For example if the clocks are operating at a ratio of 100:101, the slowly changing phase relationship due to the frequency mismatch will adversely affect pointer tracking. In this example the slower domain will appear to be early for 50 clocks and late for the other 50 clocks. If the tracking logic evaluates the pointer within the first 50 clocks, then the outcome can be different than when samples are taken in the last 50 clocks. This affect can be mitigated by ensuring that sampling occurs over the entire range of possible phase relationships. At some point the time required for the acquisition of samples impedes the tracking response making this issue problematic. Careful consideration should be made regarding the range of clock ratios anticipated during operation to ensure this issue does not create any fundamental problems.

3.3.2 Tracking Response Time

The rate at which the design can respond to drift is directly related to the number of clocks required to make a pointer correction.

F = Number of FIFO entries

S = Number of samples required by the digital filter
 λ = FIFO throttled data rate, typically the clock period of the slow domain

∂ = Maximum clock edge mismatch. This is the degree of phase mismatch between the throttled clock and the data-rate clock.

ψ = Maximum percentage of allowable clock mismatch

An estimate of the clock mismatch supported by the design can be found using the following formula:

$$\psi = \lambda / (((F * S) + (F - 1)) * \lambda + \partial) * 100$$

The required total sample time is represented by (F * S). Since there is no guarantee when the first sample might occur, the worst case latency to the first sample is almost an entire FIFO, or (F-1). As the pointer clocks in both domains begin to match each other in frequency, the uncertainty can be shown to approach one FIFO entry. Since this design intentionally strives to match the data rate on both pointers, and the clocks change their phase relationship very slowly, we can reasonably set $\partial = \lambda$. This simplification implies that in the worst-case sampling event, we have just missed a clock edge and must wait a full clock period to properly capture the signal at the next clock transition. This assumption reduces the formula to:

$$\psi = (\lambda / (((F * S) + (F - 1) + 1) * \lambda)) * 100 \\ = (1 / (F * (S + 1))) * 100$$

For many applications it is desirable to pipeline the clock events to facilitate scheduling in latency critical areas. For example a cross-bar interface to several devices may need to predict data availability to avoid delays and maximize the available bandwidth. By pipelining the throttled clock pattern which controls the faster domain's pointer, we can provide such logic with an accurate prediction of when data will be ready. This pipelining reduces the tracking response of the design and modifies the equation to:

$$\psi = (1 / (F * (S + 1) + P)) * 100 \\ \text{(where P is the number of pipeline stages)}$$

Figure 5: Maximum Drift Tracking Ability shows the tracking estimate for a typical application. This shows that an 8 entry FIFO using a digital filter requiring three sample points can adjust the pointers one entry every 40 data rate clocks. The most severe clock domain mismatch is typically found in plesiochronous systems. For these interfaces, the clock deviation is typically specified to be within 2000-5000

PPM, well below the tracking bandwidth of this example.

Design Assumptions:

8 Entry FIFO (F)

3 Sample Filter (S)

1 Clock Uncertainty (∂)

8 Clock Pipeline (P)

$$\psi = (1 / ((F * (S + 1)) + P)) * 100$$

$$\psi = (1 / ((8 * 4) + 8)) * 100 = 100/40$$

$$\psi = 2.5\% \text{ or } 25000 \text{ PPM}$$

Figure 5: Maximum Drift Tracking Ability

In many applications, concern about cycle-to-cycle jitter minimizes the overall rate of phase change seen at the FIFO interface long before this design's limitations become an issue. It is worth noting that logic operating exclusively in the fast clock domain can only decrease the pointer separation at the rate of the frequency mismatch. For very close, yet different clock frequencies, this frequency delta may actually limit the overall tracking of the system. Using our earlier 100:101 clock ratio, this observation would limit the tracking in one direction to 1/100 or 1%. This constraint imposed by the clock ratios is significantly less than the tracking response indicated by examining the logic outside the context of frequency ratio limitations. Additional tracking capability can be realized in such systems at the expense of the total throughput by throttling the clock in the slower domain. This is similar to the approach required to operate in purely mesochronous interfaces.

3.3.3 Further Refinements

In section 3.2 we mentioned that the best average latency is limited to two FIFO entries. This assessment was based on the statement that when the read pointer and the write pointer have the same value, the FIFO has underrun. In a purely integer sense, this is true. However, there exists a grey area where we can satisfy all the timing requirements on the FIFO and successfully read new data written in nearly the same clock cycle. One method for further reducing average pointer separation is to alter the timing between the pointer and its MSB captured for pointer tracking. By looking at the pointer separation slightly earlier in time, we can predict a pointer collision before it actually occurs. This ability to predict future events allows the pointers to safely run closer together. A simple way to create this early prediction is to invert the clock on the synchronizer chain making it operate

one phase (180°) earlier than the actual pointer. When the pointers are detected as colliding, they will actually still have a ½ clock of separation. Assuming the FIFO timing requirements are still satisfied, this could effectively allow the pointers to approach an average latency of 1.5 clocks. The idea of delaying a clock (or in this case a pointer) to detect a “near-miss” condition was first proposed by Chakraborty and Greenstreet [3] though their solution was custom circuit based.

Additional optimizations can be made in the digital filtering to finely tune the system to its operating environment. Because pointer collisions are fatal, it is often desirable to make the detection of pointers-too-close faster at the expense of accuracy. Similarly the detection of pointers-too-far is a non-fatal conditional which merely degrades the interface latency. As the pointers drift further apart, one might wish to track their location more accurately to ensure we don't inadvertently cause a pointer collision when we reduce their separation. Both of these behaviors can be accomplished by modifying the adjustment criteria of the digital filter.

4. Conclusion

This design effectively reduces the latency across two clock domains in systems where the clock drift is slow but unbounded in duration. The logic is adaptive to any interface, with no special consideration needed regarding the clock phase relationship between the domains. The digital nature of the design allows the implementation to scale in frequency without the potential risk of self-timed circuits. The final design can also be timed using conventional static timing tools. The ability to disable the non-deterministic logic greatly simplifies the manufacturability and testability in production. Easy changes can be made to the FIFO depth and filter logic to better suit individual applications. The only true constraint on its use is that the domain clock frequencies must be known prior to activating the FIFO in order to ensure the read and write pointers are advancing within the bandwidth of the tracking logic. An extension of this design could include frequency detection logic that establishes the data rate prior to enabling FIFO operation. Fortunately true asynchronous cases are rare in most system-on-a-chip designs, and those found are usually not latency sensitive. Creating a low latency interface between large pieces of logic is essential for leveraging the performance available as more functionality is integrated onto a single substrate. This design is proving valuable in realizing that performance.

Apple Exhibit 1008

Apple Inc. v. Redstone Logics LLC

IPR2025-01532, Page 8 of 9

5. References

- [1] D.G. Messerschmitt, "Synchronization in Digital System Design," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 8, pp. 1404-1419, 1990.
- [2] J.N. Seizovic, "Pipeline Synchronization", *Proceedings of the 1st International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 87-96, 1994.
- [3] A. Chakraborty and M.R. Greenstreet, "Efficient Self-Timed Interfaces for Crossing Clock Domains," *Proceedings 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'03)*, pp. 78-88, 2003.
- [4] T. Chelcea and S.M. Nowick, "A Low-Latency FIFO for Mixed-Clock Systems," *Proceedings IEEE Workshop on VLSI'00*, pp.119-128, 2000.
- [5] T. Chelcea and S.M. Nowick, "Robust Interfaces for Mixed-Timing Systems," *IEEE Transactions on VLSI Systems*, vol. 12, no. 8, pp. 857-873, 2004.