A Minimal Source-Synchronous Interface

Ajanta Chakraborty and Mark R. Greenstreet
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada
{chakra,mrg}@cs.ubc.ca

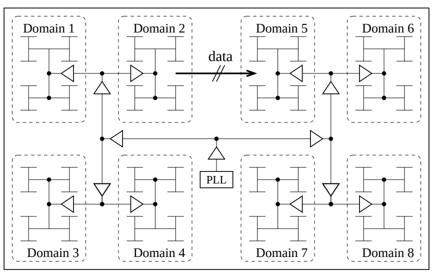

Figure 1. Multiple Clock Domains in a SOC

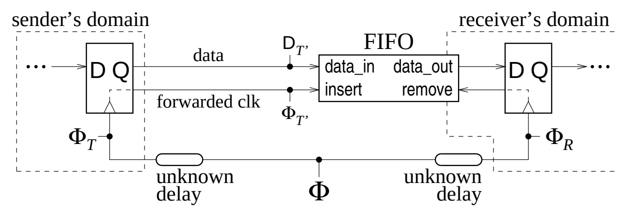

Figure 2. Source Synchronous Communication

Abstract

We present a novel implementation of source synchronous communication. Our design appears to the designer as a latch with two clock inputs, one from the transmitter and the other from the receiver. Our circuit is simple and provides a skew tolerance of nearly two clock periods. The analog dynamics of our circuit provide a simple initialization mechanism that maximizes the robustness of the interface to skew variations.

1 Introduction

As shown in figure 1, large SOC designs are typically partioned into multiple clock domains. Within each domain, clock skews are relatively small, allowing operation at high clock rates. Between clock domains, skews may be much larger. For example, in figure 1, domains 2 and 5 are at separate leaves of the clock tree distribution network. Although circuits in these two domains may be physically adjacent, there may be large, unpredictable phase difference between their clock signals [4]. Accordingly, compensation measures must be taken to compensate for these skews when data is transferred between domains.

When the clocks for communicating domains are derived from a single clock generator, source-synchronous signaling can be used to compensate for clock skew [5, 3]. As shown in figure 2, the transmitter forwards its clock along with its data to a FIFO along paths with closely matched delays. Because clocks $\Phi_{T'}$ and Φ_R have *exactly* the same period, the data insertion and data removal rates for the FIFO are *exactly* matched as well. If the FIFO is initialized to be roughly half-full, it will remain within one data item of that and never underflow or overflow. Accordingly, a source synchronous interface transfers data between the two clock domains without the overhead of synchronization or flow control.

We present a minimalist implementation of source synchronous communication where the FIFO has a capacity of a single data word. In this case, the FIFO consists of a latch controller and a single data latch as shown in figure 3. The latch controller uses the transmitter's forwarded clock, $\Phi_{T'}$, and the receiver's clock Φ_R to produce an intermediate clock, Φ_X. The timing of this clock guarantees that all set-up and hold requirements for the FIFO's latch and the receiver's latch are satisfied. Our latch control circuit is a simple adaptation of a self-timed handshake circuit and does not require critical delay matching.

Simlicity distinguishes our circuit from other approaches for crossing clock domains such as pointer-FIFO based interfaces or GALS (Globally Asynchronous Locally Synchronous). This simplicity makes it practical to implement

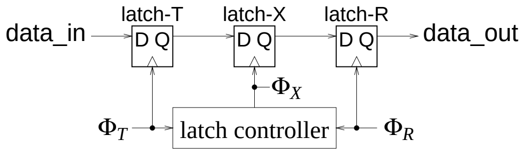

Figure 3. A Minimalist Interface

our design as a special latch with two clock inputs with little penalty in area, power, or latency compared with a traditional latch that cannot tolerate arbitrary skews. Our design is further distinguished by the initialization technique described in section 4 that exploits the analog dynamics of the handshake circuit to provide a simple mechanism that initializes the interface for maximum robustness against further variations in skew.

2 The Skew Tolerant Latch Controller

Figure 4 shows the timing constraints that must be satisfied by the latch controller. Vertical bars mark clock events, and arrows indicate constraints between these events. For simplicity, assume that the latch-T, latch-X, and latch-R all have the same timing characteristics with set-up time t_{set-up}, hold time t_{hold}, and minimum and maximum clock-to-Q propagation delay of $t_{clk\to\text{Q,min}}$ and $t_{clk\to\text{Q,max}}$ respectively. For any latch the set-up and hold window must be non-empty: $t_{set-up} + t_{hold} > 0$. Let P be the clock period. We will now show that if

$$P \geq 2(t_{set-up} + t_{hold} + (t_{clk\to\text{Q,max}} - t_{clk\to\text{Q,min}}))$$

then, the latch controller can output a Φ_X that satisfies the timing constraints.

Let Δ_{TR} be the time from an event of clock Φ_T until the next event of clock Φ_R. First, consider the case when $\Delta_{TR} \geq 2(t_{clk\to\text{Q,max}} + t_{set-up})$ time units before the next event of clock Φ_R. This is the scenario depicted in figure 4. Then, the latch controller can produce a Φ_X event $t_{clk\to\text{Q,max}} + t_{set-up}$ time units after the Φ_T event. This satisfies the set-up requirements for latches latch-X and latch-R. The time from a Φ_X event to the next Φ_T event is

$$\begin{aligned} & P - (t_{clk\to\text{Q,max}} + t_{set-up}) \\ & \quad \geq \quad 2(t_{set-up} + t_{hold} + (t_{clk\to\text{Q,max}} - t_{clk\to\text{Q,min}}) \\ & \qquad\qquad -(t_{clk\to\text{Q,max}} + t_{set-up})) \\ & \quad = \quad t_{hold} - t_{clk\to\text{Q,min}} + (t_{set-up} + t_{hold}) \\ & \quad > \quad t_{hold} - t_{clk\to\text{Q,min}} \end{aligned}$$

Thus, the hold-time requirement for latch-X is satisfied. Likewise, the time from a Φ_R event to the next Φ_X event

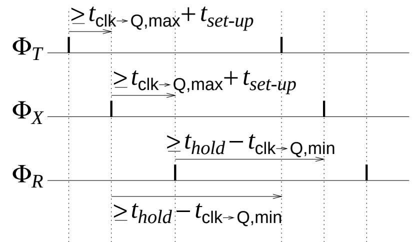

Figure 4. Timing Constraints

is

$$\begin{aligned} & (P + (t_{clk\to\text{Q,max}} + t_{set-up})) - \Delta_{TR} \\ & \quad \geq \quad 2(t_{set-up} + t_{hold} + (t_{clk\to\text{Q,max}} - t_{clk\to\text{Q,min}})) \\ & \qquad\qquad +(t_{clk\to\text{Q,max}} + t_{set-up}) \\ & \qquad\qquad -2(t_{clk\to\text{Q,max}} + t_{set-up}) \\ & \quad = \quad t_{hold} - t_{clk\to\text{Q,min}} + (t_{set-up} + t_{hold}) \\ & \qquad\qquad +(t_{clk\to\text{Q,max}} - t_{clk\to\text{Q,min}}) \\ & \quad > \quad t_{hold} - t_{clk\to\text{Q,min}} \end{aligned}$$

Thus, the hold-time requirement for latch-R is satisfied as well. Therefore, all set-up and hold requirements are satisfied when $\Delta_{TR} \geq 2(t_{clk\to\text{Q,max}} + t_{set-up})$.

Now consider the case when $P - \Delta_{TR} \geq 2(t_{hold} - t_{clk\to\text{Q,min}})$. Note that $P - \Delta_{TR}$ is the time from an event of Φ_R until the next event of Φ_T. For this case, the latch controller can produce a Φ_X event $t_{hold} - t_{clk\to\text{Q,min}}$ time units after the Φ_R event. Reasoning similar to that above shows that all set-up and hold requirements are satisfied for this case as well.

Finally, note that if $\Delta_{TR} \leq 2(t_{clk\to\text{Q,max}} + t_{set-up})$, then $P - \Delta_{TR} \geq 2(t_{hold} - t_{clk\to\text{Q,min}})$. Therefore, one of the two cases above always applies. For some offsets, both cases apply and the latch controller can operate according to either case. It is this flexibility that gives the latch controller its skew tolerance.

Figure 5 shows a finite state machine that implements the operations described above. One event is output on Φ_X each time it has received an event on Φ_T and an event on Φ_R. For $\Delta_{TR} \geq 2(t_{set-up} + t_{clk\to\text{Q,max}})$, the controller starts in state 0. Upon receiving a Φ_R event, it moves to state R. When the controller receives a Φ_T event, it moves to state TR. After a delay of $t_{set-up} + t_{clk\to\text{Q,max}}$, the controller outputs a Φ_X event and returns to state 0. Likewise, to implement the case for $P - \Delta_{TR} \geq 2(t_{hold} - t_{clk\to\text{Q,min}})$, the controller starts in state 0, moves to state T upon receiving a Φ_T event, moves to state TR upon receiving a Φ_R event, and after a delay of $t_{set-up} + t_{clk\to\text{Q,max}}$, outputs a Φ_X event and returns to state 0.

Figure 6 shows five timing scenarios for the latch controller for various phase relations between clocks Φ_T and Φ_R. Each event is marked with a letter or number to indi-

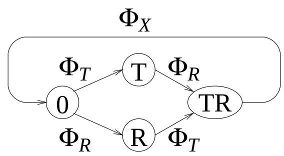

Figure 5. Latch Controller State Diagram

cate the data value loaded into the corresponding latch by that clock event, thereby showing how data flows through the interface. In the first scenario, $\Delta_{TR} = 2(t_{set-up} + t_{clk\rightarrow\mathrm{Q,max}})$. The controller generates a Φ_X event $t_{set-up} + t_{clk\rightarrow\mathrm{Q,max}}$ time units after the Φ_T event, and the data value loaded into latch-T by a Φ_T event is loaded into latch-R $2(t_{set-up} + t_{clk\rightarrow\mathrm{Q,max}})$ time units later. In this scenario, the Φ_T event for datum A triggers the output of a Φ_X event and returns the controller to state 0. The Φ_R for datum A moves the controller to state R where it waits for the next Φ_T event. The Φ_T event for datum B moves the controller to state TR, triggering another Φ_X event and repeating the cycle.

The remaining scenarios show operation as clock Φ_R arrives progressively later relative to Φ_T. In scenario 2, Φ_R is later than in scenario 1, but the controller remains in the 0 $\rightarrow$ R $\rightarrow$ TR $\rightarrow$ 0 cycle.

In scenario 3, the Φ_R event is slightly later than the corresponding Φ_T event, and Φ_X events are now triggered by the events of Φ_R. This scenario starts in state 0 and moves to state T with the Φ_T event for datum A. At this point, latch-T holds datum A and latch-X holds datum 0. The next Φ_R event loads datum 0 into latch-R, and the controller moves to state TR, then outputs a Φ_X event to load datum A into latch-X, and the controller returns to state 0 to begin a new cycle. Scenarios 4 and 5 show even greater delays.

The interface operates correctly for any phase offset between the one depicted in scenario 1 and the one for scenario 5. The skew can change in this range without any dropping or duplication of data or other failure. In scenario 1, the transmitter clock event occurs $2(t_{set-up} + t_{clk\rightarrow\mathrm{Q,max}})$ time units before the receiver event that loads the same data value. In scenario 5, the receiver clock event occurs $2(t_{hold} - t_{clk\rightarrow\mathrm{Q,min}})$ time units before the transmitter event that is two cycles later than the one that produced the datum being loaded into the receiver latch.

Let Δ_σ denote the width of the skew tolerance window. The analysis above yields:

$$\Delta_\sigma \quad \leq \quad 2(P - (t_{set-up} + t_{hold}) - (t_{clk\rightarrow\mathrm{Q,max}} - t_{clk\rightarrow\mathrm{Q,min}})) \tag{1}$$

In other words, the skew tolerance is two clock periods minus the overhead of the latch set-up and hold window and uncertainties in the latch propagation delay.

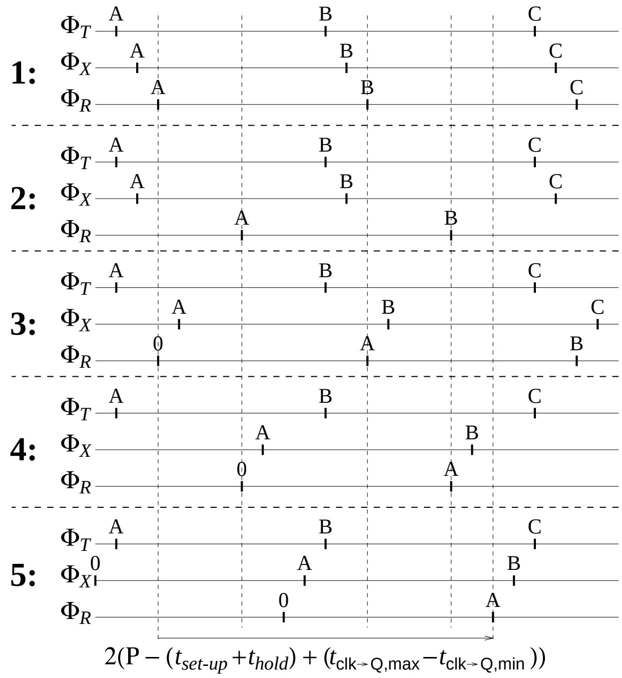

Figure 6. Five Timing Scenarios

At the extremes of the skew tolerance window, the latch controller must complete a cycle in response to an event on Φ_T (resp. Φ_R) before the event on Φ_R (resp. Φ_T) for the next cycle. Let γ denote the time for the controller to traverse the path from state R (resp. T) to state 0 in response to an event on Φ_T (resp. Φ_R). We note that operation for case 1 above requires $\Delta_{TR} \geq \gamma$, and operation for case 2 requires $P - \Delta_{TR} \geq \gamma$. This places a second bound on Δ_σ:

$$\Delta_\sigma \quad \leq \quad 2(P - \gamma) \tag{2}$$

In practice, this tends to be the constraint that determines the actual skew tolerance.

Note that our design has maximum skew tolerance when Φ_T and Φ_R are nearly coincident. This is precisely the situation that leads to timing failures for a traditional latch. Because a traditional latch has a non-empty set-up and hold window during which its input data must be stable, the skew tolerance of such a latch must be less than a full clock perod. Our interface avoids this problem by clocking the intermediate latch safely after the coincident clock events. From this scenario, the two clocks may drift in relative phase in either direction for nearly a full clock period. This gives our design a skew tolerance of nearly two clock periods. As described in section 4, this allows our interface to work properly with any intial skew between the sender and receiver.

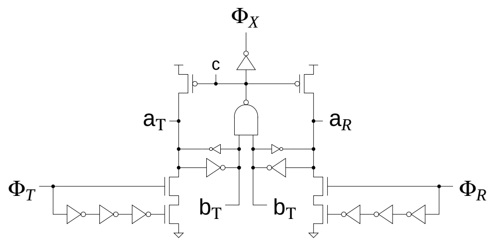

Figure 7. The Latch Control Circuit

3 A Self-Resetting Implementation

Figure 7 shows our implementation of the latch controller. We designed the controller for positive edge triggered latches, thus clock events are rising edges. Our design is a self-resetting CMOS circuit [2] similar to GasP handshaking circuits [6]. State 0 of the state machine from figure 5 corresponds to a state where nodes a_T, a_R, and c are high, and nodes b_T, b_R, and Φ_X are low. A rising edge on Φ_T causes node a_T to drop, and a rising edge on Φ_R causes node a_R to drop. When both are low, nodes b_T, b_R, go high; node c goes low, and Φ_X goes high. The delay of the path from a rising edge of clock Φ_T or Φ_R to a rising edge of Φ_R is sufficient to satisfy the set-up and hold requirements described in section 2. The low value on node c initiates the self-reset, bringing the circuit back to the state described at the beginning of this paragraph.

The timing requirements for our latch controller are fairly simple. The delay through the chains of three inverters for the edge catching circuits for Φ_T and Φ_R must be long enough to ensure that nodes a_T and a_R complete their downward transition in response to a rising input clock edge. The delays of the chain must also be small enough to ensure that nodes the pull-down paths for nodes a_T and a_R are disabled when node c goes low. The delays for upward transitions on nodes a_T and a_T must not be so badly mismatched that one rises and resets c to high before the other rises. In practice, these conditions are easily satisfied.

4 Initialization

The previous sections described how our interfaces function in steady state. Here, we consider how to initialize an interface into an acceptable steady state cycle. For example, scenarios 1 and 3 of figure 6 have the same value for Δ_{TR}. Scenario 1 has lower latency; however, after initialization the relative clock phases may change due to power supply noise, temperature variations, etc. Wherease, scenario 3 can tolerate substantial changes of the skew in either direction, scenario 1 will fail with any further advance of Φ_R relative to Φ_T. Typically, scenario 3 will be the prefered initialization. Similarly, scenarios 2 and 4 have the same Δ_{TR}. Scenario 2 is slightly more robust to later skew variations and will be the prefered initialization for many designs. In typical designs, the drift in skew under operation could be just as much of an advance as of a retard (note: one domain's advance is another domain's retard). With this assumption, the most robust initialization is the one that tolerates the greatest variation in either direction.

The analog dynamics of our circuit provide a simple mechanism for initializing the latch controller to the most robust operating point. With our method, the internal delays of the latch controller are modified during initialization. We start with a slow controller and gradually bring it up to full speed. We do this in our implementation by using a separate ground signal for the latch controller connected to an internal voltage reference. This voltage sweeps from 1.8V (equal to Vdd) down to 0V (normal operation). The controller speeds up during this sweep according to the well-known relationship between power supply voltage and speed.

When the controller is sufficiently slow, it cannot cycle as fast as the clocks. Under these conditions, nodes a_T and a_R will still go low in response to their respective clock inputs, and when both go low, the controller will generate a Φ_X event and return to state 0. However, it may miss incoming clock events that occur before the reset is complete.

Assume that $\Delta_{TR} < P - \Delta_{TR}$ as in scenarios 1 and 3, and consider operation at a time during the initialization when the controller takes time time Δ_{TR} to traverse a path from state TR to state 0. If the latch controller reaches state TR in response to a Φ_R event, then it will return to state 0 in time for the next Φ_T event and will continue to cycle correctly. On the other hand, if the controller reaches state TR in response to a Φ_T event, then it will return to state 0 *after* the next Φ_R event. It will remain in state 0 until the next Φ_T event and transition to state T. With the next Φ_R event, the controller will move to state TR and continue to cycle properly from there. This corresponds to scenario 3, the more robust initialization as noted above. Having reached this cycle, the controller will continue to complete all transitions on time with further reductions of its internal delays. Thus, it will remain in the preferred cycle.

Metastable behaviour [1] is possible if $\Delta_{TR} \approx P/2$. In this case, the controller can settle to either of two scenarios that are nearly equally robust to future variations in the skew. As with other metastable situations, the probability of remaining in an indeterminate state decays exponentially with time. Accordingly, our circuit can be initialized very reliably, and no metastability can occur after successful initialization.

5 Implementation

We have designed a proof-of-concept chip for our interface which we are fabricating using the TSMC 0.18μ process through CMC, the Canadian Microelectronics Corporation. Our chip consists of an LFSR to generate pseudo-random data from the transmitter, our interface circuit with the dynamic initialization technique described in section 4, and an LFSR checker in the receiver. In this section, we briefly summarize additional issues that we addressed in our design.

The set-up and hold times for our latches are roughly 150ps and 90ps respectively. These times are much shorter than the delays though the latch control circuitry. We widen the skew tolerance window by delaying the clocks for latches latch-T and latch-R. With this padding, the skew tolerance of the interface is determined by the minimum cycle time of the latch controller. This cycle time is 340ps. Thus, the skew tolerance window has width $2P - 680ps$. The skew window is wider than the clock period for a clock period of 1400MHz or lower. Under these conditions, the interface can operate with an arbitrary fixed skew.

Initially, the self-resetting latch controller generated narrow pulses on Φ_X that were marginal for triggering our latches. We did not want to modify the latch controller as this would increase its cycle time and decrease its skew tolerance. Instead, we used a self-resetting buffer to generate Φ_X and widened the pulse by including sufficient delay in the reset path for the buffer.

6 Conclusions

We have presented a novel source synchronous interface for crossing between clock domains within chip. Our design consists of a standard latch and a simple circuit for deriving the required clock signal; it requires little area; and it adds minimal latency to communication paths. These "lightweight" properties of our interface offer the designer opportunities to partition a design into clock domains to simply the clock distribution network, reduce the number of global timing constraints, and increase clock frequency. Exploiting the analog dynamics of the self-resetting latch controller, our interface automatically determines its ideal latency according to the actual skews encountered during initialization. We have designed a proof-of-concept chip to demonstrate our interface and are currently fabricating it in a 0.18μ CMOS process.

While our proof-of-concept design is full-custom layout, our latch controller circuit could be included in a standard cell library and used in typical ASIC design flows. Such a cell would appear to the designer as a latch with two clock inputs as depicted in figure 8. One clock input is for the interface to the transmitter of data and the other for the receiver. While we used a dynamic, self-resetting design, static implementations are possible for less stout-hearted designers who require less performance.

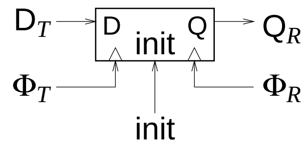

Figure 8. A Clock-Domain Crossing Latch

Our current design requires the sender's and receiver's clocks to be exactly matched in frequency. We are currently exploring variations for interfaces where the two clock frequencies are rational multiples of one another, or where the two frequencies are closely but not exactly matched. We believe that the simplicity of the circuit presented here will make it the method of choice when exact frequency match is possible. We anticipate extensions to this simple approach will provide a comprehensive set of solutions for communication between clock domains within a SOC.

Acknowledgements

Jo Ebergen of SUN Microsystems pointed out the similarities between our controller and GasP handshaking circuits. Roozbeh Mehrabadi, Roberto Rosales, and Brian Winters have provided valuable suggestions during the design of our chip. This work has received support from NSERC and CMC.

References

[1] T. Chaney and C. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computers*, C-22(4):421–422, Apr. 1973.

[2] T. I. Chappell, B. A. Chappell, et al. A 2-ns cycle, 3.8-ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture. *IEEE Journal of Solid-State Circuits*, 26(11):1577–1585, Nov. 1991.

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

[4] A. L. Fisher and H. Kung. Synchronizing large VLSI processor arrays. *IEEE Transactions on Computers*, C-34(8):734–740, Aug. 1985.

[5] M. R. Greenstreet. *STARI: A Technique for High-Bandwidth Communication*. PhD thesis, Department of Computer Science, Princeton University, Jan. 1993.

[6] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proceedings of the Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53, Apr. 2001.