Evaluation of pausible clocking for interfacing high speed IP cores in GALS Framework

Joycee Mekie Supratik Chakraborty Dinesh K. Sharma
Indian Institute of Technology, Bombay, Mumbai 400076, India
jrm@ee,supratik@cse,dinesh@ee.iitb.ac.in

Abstract

Pausible clocking schemes have been proposed by GALS architects as a promising mechanism for reliable data transfer between synchronous modules fed by low-speed independent clocks. In this paper, we argue that existing schemes are not well-suited for interfacing high-speed IP cores with large clock-distribution tree delay and high communication rates. We propose an alternative interface circuit design for such IP cores that works with partial handshake between communicating modules and minimizes the performance penalty of the sender and receiver. Our circuit, unlike pausible clocking, has a small probability of failure.

1 Introduction

The increasing complexity of System-on-Chip (SoC) designs has given rise to chips with multiple IP cores, possibly fed by different clocks, integrated on the same die. These systems are best viewed as *globally asynchronous locally synchronous* (GALS) systems, in which locally synchronous IP cores (henceforth called *LS modules*) compute synchronously and communicate asynchronously to implement the overall system functionality. Techniques like adaptive synchronization [1], STARI [2] and self-timed interfaces [3] can be used to interface LS modules in such systems if the communicating modules are clocked by phase-shifted or frequency-scaled versions of a global clock. For synchronizing data transfers across unrelated clock domains, multi-flip-flop synchronizers are typically used. Pipeline synchronization [4] has also been used by researchers for interfacing disparate clock domains. While these synchronizer based schemes are simple to implement, they introduce a latency penalty and suffer from non-zero probability of failure due to metastability. A more complex interfacing scheme, called pausible clocking [5, 6, 7, 8, 9], has been shown to eliminate metastability-induced failures in GALS systems. Unfortunately, existing work on pausible clocking has focused on small LS modules running at speeds of up to a few hundred megahertz. In this paper, we take a critical look at the applicability of pausible clocking schemes to GALS systems with large LS modules running at gigahertz and above frequencies. We also propose an alternative interface circuit design for such systems.

The input clock signal in large LS modules (e.g., CPU and DSP cores) is typically buffered through a clock distribution tree before being fed to the flip-flops (see Fig. 1). In the subsequent sections, we argue that existing pausible clocking schemes are not well-suited for interfacing such LS modules running at multi-gigahertz frequencies, if we desire high rates of data transfer and minimal adverse impact on overall system performance. We then propose a new interface circuit design for solving the interfacing problem for these modules. Our circuit allows data transfer between two high-speed LS modules with large clock buffer trees and with a *partial handshaking* protocol, while minimizing the performance penalty. The choice of a partial handshaking protocol instead of a complete handshaking protocol is motivated by the fact that the former allows a higher rate of data transfer, and is, therefore, better suited for high-performance GALS systems. Simulation results show that our interface works correctly for sender and receiver frequencies of up to 2.8 GHz across a temperature range of 0^{o}C to 70^{o}C, and for all four process corners. However, our circuit has a non-zero probability of failure. We identify the conditions for its failure using static timing analysis techniques.

The remainder of this paper is organized as follows. Section 2 describes the interfacing problem we wish to address. Section 3 outlines the difficulties in applying existing pausible clocking schemes to our interfacing problem. In Section 4, we present a new interface circuit design. Section 5 describes sufficient timing constraints for correct operation of our circuit. We present simulation results in Section 6, and conclude the paper in Section 7.

2 Problem description

We focus on GALS systems with communicating LS modules having the following characteristics: (i) each module is clocked by an independent clock ticking at gigahertz and above frequencies, (ii) each module has an internal clock buffer tree with *multi-cycle delay*, (iii) the modules have interleaved bursts of computation and communication, (iv) during communication, the inter-module data transfer rate can be as high as the slowest of the two clocks, and (v) the modules employ a sender-initiated *partial handshake* protocol to facilitate a high rate of data transfer. Such a GALS system can be envisaged in future SoCs containing high-speed CPU, DSP and/or embedded processor cores.

Our goals in this paper are: (i) to critically examine the applicability of existing pausible clocking schemes to the above interfacing problem, and (ii) to design an interface circuit that allows correct data transfer between the above modules with a high (ideally, one) probability. In addition, we wish to minimize the adverse impact of the interfacing scheme on the overall performance of the system. Thus, synchronous computations within LS modules and asynchronous communication between them should be minimally stalled.

3 Pausing clocks with large buffering trees

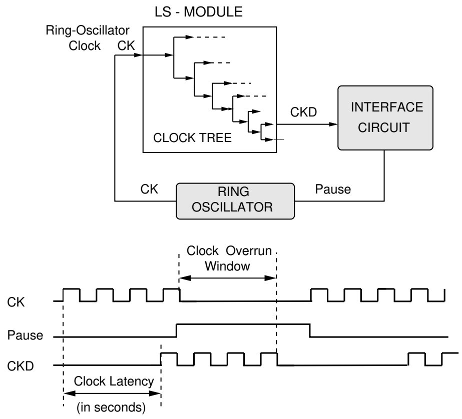

Figure 1. Clock latency and clock overrun

The fundamental assumption underlying existing pausible clocking schemes [7, 8] is that all activities in an LS module are frozen within one clock period after the input clock (ring-oscillator clock) is paused. Thus, whenever we detect a situation wherein the sampling of data lines by the receiver module can lead to potential metastability, the receiver's clock is paused to prevent synchronization failure. Similarly, pausing the sender module's clock prevents new data from being sent until the previous data has been correctly sampled by the receiver. Thus, pausing the clock under the above assumption serves two distinct purposes: preventing synchronization failure at the receiver's end, and achieving flow control at the sender's end.

Unfortunately, while the above assumption holds for low-speed designs (up to a few hundred megahertz) [9], it breaks down for large LS modules operating at multi-gigahertz frequencies. The input clock signal in such large LS modules is typically buffered through a clock distribution tree before being fed to the flip-flops. This introduces a delay, called *clock latency*, between the input clock signal and the clock signal available at the flip-flops (see Fig. 1). For large modules clocked at multi-gigahertz frequencies, clock latencies can easily add up to a few clock cycles. Multiple cycles may therefore elapse in the window between the pausing of the input clock and the stalling of the flip-flops. We call this window the *clock overrun window*, as shown in Fig. 1.

In a pausible clocking scheme, if the clock overrun window spans multiple cycles, the receiver's flip-flops can continue to sample data even after potential metastability condition has been detected and the receiver's clock input has been paused. Clearly, this can lead to synchronization failure at the receiver's end. One way to circumvent this problem is to pad each data line that is sampled by the receiver with a delay equal to the receiver's clock latency. Ideally, this matched delay padding should be done after metastability detection but before the data lines are sampled by the receiver. Unfortunately, this is a hardware-intensive solution and may not be very practical in general. In addition, if we assume a complete request/acknowledge handshake protocol between the communicating modules, the above solution constrains the maximum rate of data transfer, as shown in []. On the sender's side, flow control can still be achieved with a multi-cycle clock overrun window if we use a buffer to hold data items sent during the overrun window. However, this requires the sender's clock to be paused whenever the buffer is non-empty. This can adversely affect the maximum rate of data transfer between the modules and cause the sender module to stall unnecessarily. Thus, the overall system performance is hampered.

In view of the above difficulties in applying pausible clocking schemes to the interfacing problem described in Section 2, we describe a new interface circuit, also called a *wrapper*, in the remainder of the paper.

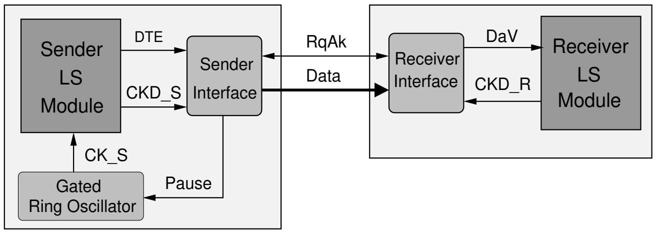

Figure 2. Functional units in GALS wrapper

4 Wrapper Architecture

For purposes of this discussion, we will refer to the clock inputs (ring oscillator clocks) of the sender and receiver modules by CK_S and CK_R, respectively. Similarly, we will use CKD_S and CKD_R to refer to the clock signals fed to the flip-flops of the sender and receiver modules. We will also assume the following sender-initiated partial handshake protocol for inter-module communication. Whenever the sender wishes to send data to the receiver, it raises a control signal called Data_Transmission_Enable (or DTE) and simultaneously places data on the data bus, synchronously with CKD_S (see Fig. 2). If the sender

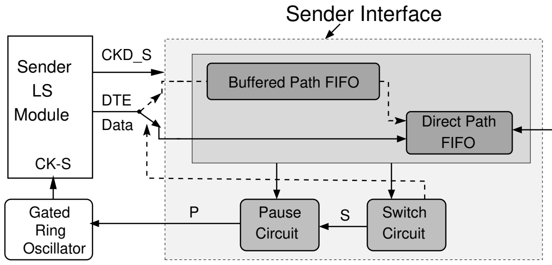

Figure 3. Schematic of sender-interface

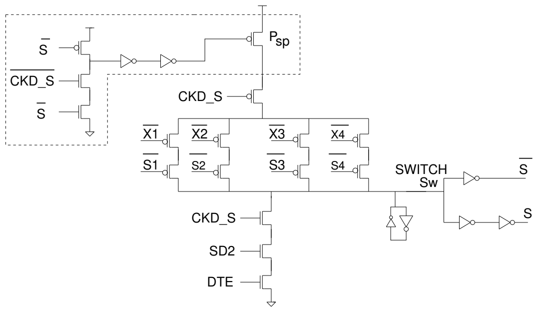

Figure 5. Switch circuit

wishes not to send data in a particular cycle, it lowers DTE synchronously with CKD_S. The receiver module samples the Data_Valid (or DaV) signal and also the value on the data bus with each rising edge of CKD_R. It accepts the value sampled from the data bus as valid data if DaV is sampled *low*. Note that the sender does not wait for an acknowledgment from the receiver to continue with its next data transfer. Similarly, the receiver does not generate an acknowledgment after it has sampled a valid data. Under ideal circumstances, this partial handshake allows the sender and receiver modules to communicate at a rate determined by the slower of CK_S and CK_R. Note also that we have chosen DaV to be an active low signal.

Fig. 2 shows the basic architecture of our wrapper design. It consists of three functional units: (i) sender interface, (ii) receiver interface and (iii) clock generator for the sender module. The clock generator is used to pause the sender's clock to achieve flow control. For reasons outlined in Section 3, we use a buffering scheme to store data items sent by the sender during its clock overrun window. We do not use pausible clocking for synchronization on the receiver's side. Instead, we use an extension of the basic synchronizer idea along with a mutex element and self-timed circuits to achieve synchronization. The self-timed sender and receiver interfaces interact with each other and with the clock generator to effect data transfer from the sender to the receiver. We now describe each component of our architecture in detail.

Sender Interface: Fig. 3 shows a block diagram of the sender interface circuit. This circuit uses the DTE and CKD_1 signals to pull signal RqAk low, indicating a request to the receiver interface. After the receiver has latched the data, the receiver interface deasserts the same RqAk signal by pulling it high. Thus, the two interfaces communicate by driving a single line in a mutually exclusive manner. If the sender has a new data to be sent before the receiver interface has deasserted RqAk, the sender's clock must be paused to achieve to avoid the previous data from being overwritten. For reasons explained in Section 3, the sender may, however, continue to send data during its clock overrun window. To avoid losing this data, we use a FIFO to buffer it. The sender interface thus consists of (i) a *direct path*, which may also have a FIFO to counter the effect of long interconnect delays between modules [7], and (ii) a *buffered path*, which has an additional FIFO to store data during clock overrun. As shown in Fig. 3, data flows through the direct path when the sender's clock is not paused, and through the cascade of the additional FIFO and the direct path when the clock is paused. A *switch circuit* controls the switching of data and control flow from the sender module to the buffered path or the direct path. The sender interface also has a *pause circuit* that generates the gating signal necessary to pause the sender's clock. While it is tempting to use the same signal for switching the buffers and also pausing the sender's clock, we chose to separate them to allow us greater freedom in releasing the clock whenever possible. This minimizes the stalled cycles of the sender.

For the buffers in the direct and buffered paths, we have used linear GasP FIFOs [10], as shown in Fig. 4. The length of the FIFO in the direct path depends on the ratio of the interconnect delay to the clock period [7]. Since this is not our focus in the current work, we have arbitrarily fixed this to be 2. The number of stages in the buffered path FIFO must be greater than the number of clock cycles in the clock overrun window. Assuming a clock-latency of 2 cycles, we have used a $3-$stage GasP FIFO in the buffered path. In Fig. 4, N-stacks connected to nodes SF1 and SD1 are used to pull the corresponding nodes low, effectively steering a request in the buffered or direct path, respectively. The switch signal S is used to multiplex data and control lines between the direct and buffered paths. The signal $\overline{DCKD}$ is the inverted and delayed version of CKD_S. Using this signal along with CKD_S in the N-stack ensures that there is a conducting pulse (instead of a continuously conducting path) that pulls the corresponding node (SF1 or SD1) low. Fig. 5 shows the details of the switch circuit. The signal S signal is normally high and causes data and control to be steered into the direct path. However, if the direct path FIFO becomes full (indicated by SD2 in Fig. 4 being high) and the sender has a new data to be transferred (indicated by DTE and CKD_S going high), S is pulled low by the N-stack shown in Fig. 5. This steers data and control into the buffered path FIFO. The signal S is restored to its high value when the buffered

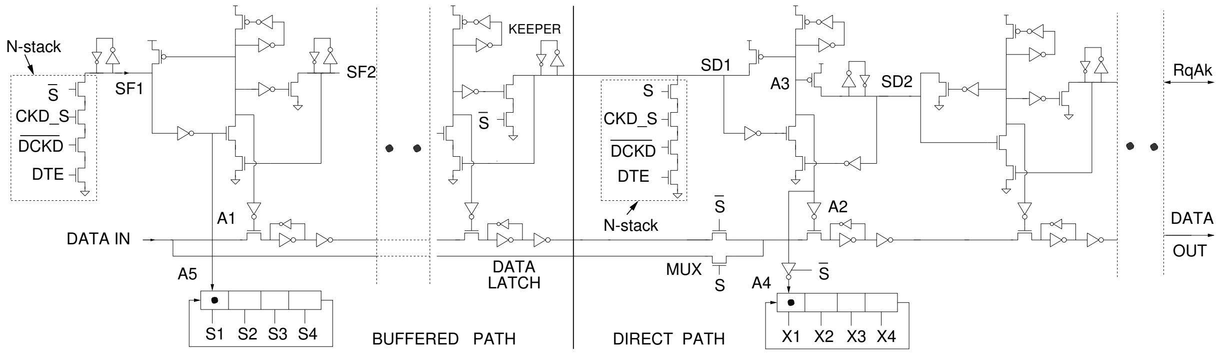

Figure 4. Data buffering scheme

path FIFO becomes empty. To detect this condition, we use two ring-counters driven by signals A5 and A4, as shown in Fig. 4, that keep track of data items entering and leaving the buffered path FIFO. The additional circuit within the dotted box in the top left corner of Fig. 5 is added to meet timing constraints imposed by the counters. This circuit does not affect the functionality of the switch circuit. Details about this can be found in Mekie et al's report[11].

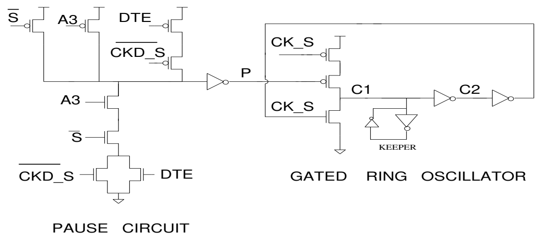

Figure 6. Pause and clock generator circuit

As mentioned above, we have chosen to separate the circuit for switching paths from the circuit that determines when the sender's clock should be paused. We observe that the sender's clock can be released when either data and control are being steered into the direct path (indicated by S being high), or whenever there is an "empty space" created in the FIFO. The latter situation can arise even when data and control are being steered into the buffered path FIFO if (i) a previous data has been latched by the receiver (indicated by a high value on signal A3 in Fig. 4), or (ii) the sender did not send a data item in a clock period during the clock overrun window (indicated by a low value on DTE when CKD_S is high). Detection of these conditions is achieved by the pause circuit in Fig. 6, where a high value of P pauses the gated ring oscillator.

Gated Ring Oscillator Circuit: It can be seen from Fig. 6 that signal P is asynchronous with respect to CK_S, and hence can cause runt pulses in the ring oscillator. To avoid these pulses, certain timing constraints are imposed on the occurrence of CK_S and P. As detailed in Mekie et al's re-

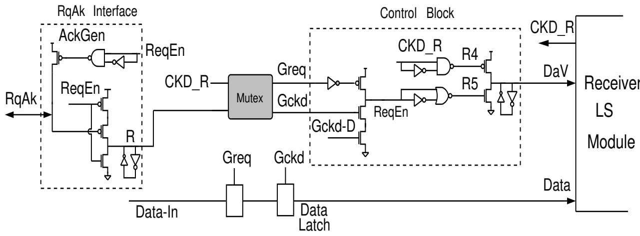

Figure 7. Receiver interface circuit

port [11] these can be met by design.

Receiver Interface: The receiver interface receives a request as a falling transition on the RqAk signal and forwards it to the receiver module, as shown in Fig. 2. Specifically, it (a) generates an active low DaV signal that is sampled by the receiver's clock, and (b) deasserts the RqAk signal by pulling it high, thereby acknowledging receipt of data to the sender interface. Fig. 7 shows the structure of the receiver interface circuit. It consists of three blocks: a RqAk interface block, a mutex element, and a control block. In order to understand the operation of this circuit, we must focus on four key internal signals: R, Greq, Gckd and ReqEn. Signal ReqEn is low to begin with. A request from the sender interface, indicated by a falling transition on RqAk, causes signal R in the RqAk interface block to go high. Signal R and the buffered clock, CKD_R, of the receiver serve as inputs to the mutex element. Depending on the relative times of arrival of R and CKD_R, the mutex grants one of Greq and Gckd. Note that since CKD_R is never paused, Greq is guaranteed to be granted within half a clock cycle after R goes high. Once Greq is granted, the input data is latched by the first latch shown in Fig. 7. In addition, signal ReqEn is pulled high in the control block. The rising transition on ReqEn eventually pulls RqAk high in the RqAk interface block, thereby deasserting RqAk. The rising of ReqEn also pulls R low and makes the RqAk interface block opaque to further changes on the RqAk signal until the interface circuit has ensured that the current data can be safely latched by the receiver.

After signal R goes low, the next rising edge of CKD_R causes the mutex element to grant Gckd. This causes the input data to be latched by the second latch in Fig. 7. In addition, ReqEn is pulled low in the control block. This gives rise to a high pulse on R5 in the control block, which pulls DaV low. This value of DaV is sampled by the next rising edge of CKD_R, which also resets DaV to high through the pulse generator circuit in the control block. The falling transition on ReqEn also makes the RqAk interface block transparent to further requests from the sender interface.

The receiver interface circuit can malfunction if Gckd is granted shortly before the falling edge of CKD_R, causing a runt pulse on Gckd. The associated probability of failure can be reduced by using a narrow-pulse suppressing filter that makes use of Gckd and a delayed version, Gckd-D, of the same signal. This is shown in the control block in Fig. 7, and is detailed in Mekie et al's report [11]. Thus, our synchronizer circuit has a non-zero probability of failure like multi-flip-flop synchronizers. Nevertheless, the synchronizing component of our receiver interface requires almost half the number of gates compared to that required by a two-flop synchronizer using pass-gate design. In addition, while a two-flop synchronizer has a fixed latency of 2 clock cycles, our receiver interface can have a latency of either one or two clock cycles depending upon the instant of arrival of RqAk with respect to CKD_R. The best possible throughput for our receiver interface is 1 data transfer per clock cycle which is double that of the standard two-flop synchronizer.

5 Timing analysis of interface circuits

Our interface circuit designs make use of GasP FIFOs and self resetting loops, that operate correctly only if certain timing constraints are met. In this section, we list some of the timing constraints required for correct operation of our design. A more detailed analysis as well as the complete set of constraints can be found in Mekie et al's report [11]. Our timing analysis proceeds by identifies conditions for proper charging and discharging of all circuit nodes driven by pull-up and pull-down networks. Specifically, we ensure the following: (i) once a pull up (pull-down) stack starts conducting, it conducts long enough to charge up (discharge) the corresponding node, and (ii) the pull up and pull down stacks for the same node never conduct simultaneously. Since we use keeper circuits, we need not ensure that at least one of the stacks always conducts.

For clarity of exposition, let δ_{pu-x} (δ_{pd-x}) denote the minimum duration for which the pull up (pull down) network for node x must conduct to charge (discharge) node x. We use $t_x \uparrow$ and $t_x \downarrow$ to denote the times of rising and falling transitions of node x, and δ_G to represent the delay of gate G. We also denote the clock period of the sender by T_S and that of the receiver by T_R. For lack of space, we list below sufficient timing constraints for proper operation of the counter circuits in Fig. 4 and of the receiver circuit in Fig. 7. We have chosen these two units because unlike other units, their timing constraints cannot be enforced by design, and point to conditions under which our interface circuit can malfunction.

Timing constraints for counter circuit: Each X_i output of the counter driven by A4 in Fig. 4 is required to change before the falling edge of the sender's buffered clock CKD_S. In other words, $t_{CKD\_S} \downarrow - t_{X_i} \uparrow > \delta_{pd-Xi}$ for all $i \in \{1, 2 \ldots\}$

Timing constraints for receiver interface: The labels used in these constraints refer to Fig. 7.

$$\begin{aligned} \delta_{pd-ReqEn} &< t_{Gckd} \downarrow - t_{Gckd-D} \uparrow \\ \frac{T_R}{2} &> \delta_D + \delta_{pd-ReqEn} + \delta_{nor} + \delta_{pd-DaV} + t_{su} \\ t_{Gckd} \uparrow &\geq t_{CKD\_R} \uparrow + \delta_m \end{aligned}$$

where (i) δ_D denotes the delay between Gckd and Gckd-D, (ii) t_{su} denotes the setup time for latching DaV, (iii) δ_{nor} is the delays of the NOR gate fed by ReqEn in Fig. 7 and, (iv) δ_m denotes the maximum delay of the mutual exclusion element.

On listing the timing constraints for all nodes in the circuit [11], we find that all but the ones listed above can be met by appropriate transistor sizing. In fact, the above constraints can indeed be violated due to bad timing of the signals with respect to the sender and receiver module's clock edges. Thus, like synchronizer-based schemes, our interfacing scheme also suffers from a non-zero probability of failure.

6 Simulation Results

We have designed SPICE models of our interface circuit assuming an 8-bit wide data path between the modules. SPICE simulations of our circuits have been carried out assuming 0.18μ CMOS technology. Fig. 8 shows the results of simulation for the sender module operating at 2.8 GHz and the receiver module operating at 1.5 GHz. These frequencies have been chosen to illustrate the operation of our interface circuit when interfacing disparate clock domains. We have simulated a scenario where the sender wants to send 6 data bytes labeled U, V, W, X, Y and Z. In Fig. 8, tracks A2 and A1 show the latching pulses that are generated when data flows through the direct path FIFO and the buffered path FIFO, respectively (see Fig. 4). When the sender wants to send data, it raises DTE synchronously with CKD_S. The first and second bytes are steered into the two-stage direct path FIFO, shown by U and V pulses on track A2. The RqAk signal for U, generated by the sender interface, is deasserted by the receiver interface before the third clock cycle, as shown in track RqAk. Consequently, the third data item W also gets steered into the direct path FIFO. However, when the sender wants to send the fourth

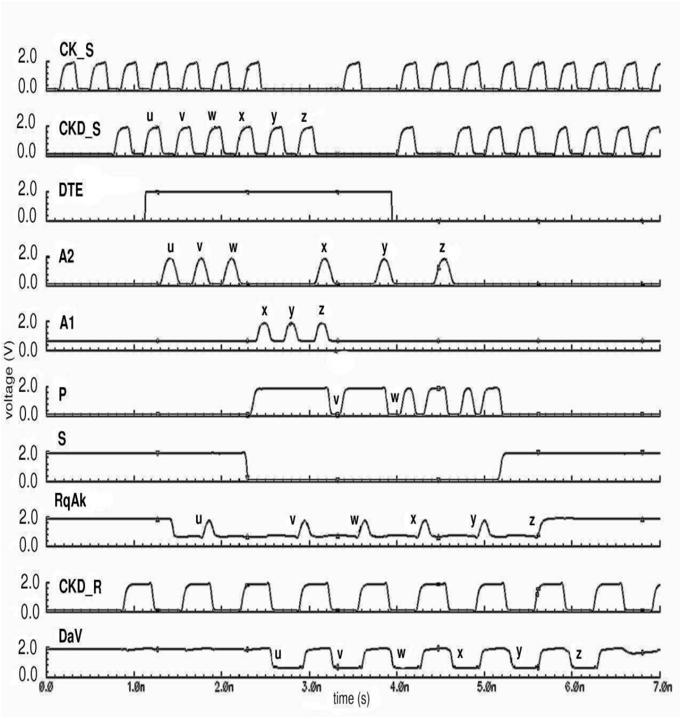

Figure 8. Simulation Results

data item, X, the direct path FIFO is full as the RqAk for V has not been deasserted by the receiver interface. So, switch S is pulled low in the fourth clock cycle of CKD_S, and data gets steered into the buffered path FIFO. At this instant, the pause signal P is also pulled high and the ring oscillator clock CK_S is paused. During the clock overrun window, data bytes Y and Z are released and get stored in buffered path FIFO. This is shown on track A1 which has 3 pulses for X, Y and Z. Even in the switched condition, pause P is released when the receiver acknowledges receipt of V by deasserting RqAk. This allows one clock pulse of CK_S to be released. A clock pulse is also released when DTE is pulled low. This clearly shows the advantage obtained by separating the switch and pause circuits. If the same signal was used for both switching and pausing, then the clock would have remained paused until the buffered path FIFO became empty. Note that the receiver clock is never paused. DaV is pulled low for every valid data transfer and is pulled high after the data is latched by the receiver module.

The interface circuit has been simulated to work correctly for temperatures ranging from 0^{o}C to 70^{o}C for all four corners of technology variation. The dynamic power consumption of our interface circuit is about 17 mW in the control circuit and 5 mW in 8-bit data bus, which is marginal vis-a-vis the power consumption in large IP cores. Following the technology and device geometries chosen for the interface circuit, we can support LS modules running at up to 2.8 GHz with a throughput rate of 1.1 gigabytes per second.

7 Conclusion

In this paper, we evaluated the applicability of pausible clocking schemes to the problem of interfacing large high-speed IP cores in SoCs. Our investigation revealed that if the clock buffering delay exceeds the clock period of an LS module, and if we require high rates of communication and minimal performance penalty, existing pausible clocking schemes are not well-suited for the interfacing problem. We also designed a new interface circuit for the above interfacing problem assuming a partial handshake between LS modules. Our interfacing scheme does not pause the receiver clock for synchronization purposes. We, however, pause the sender clock to achieve flow control. However, our circuit is optimized to minimize the stalls in the system. Simulation results show that our circuit works at all process corners and with different sender and receiver frequency combinations. However, our timing analysis reveals the presence of a non-zero probability of failure, as in synchronizer based designs. Unlike two-flop synchronizer based systems, however, our interface circuit has a lower average latency penalty. Thus, we believe that our circuit enjoys the best of both pausible clocking and synchronizer worlds when interfacing large high-speed modules.

References

[1] R. Ginosar and R. Kol, "Adaptive synchronization," in *Proc. of ICCD*, pp. 188–189, October 1998.

[2] M. R. Greenstreet, *STARI: A Technique for High-Bandwidth Communication*. PhD thesis, 1993.

[3] A. Chakraborty and M. R. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Proc. of ASYNC'03*, pp. 78–88, May 2003.

[4] J. N. Seizovic, "Pipeline synchronization," in *Proc. of ASYNC'94*, pp. 87–96, 1994.

[5] D. M. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, Oct. 1984.

[6] C. L. Seitz, *System Timing Introduction to VLSI Systems Ch. 7*. Addison-Wesley Pub. Co., 1980.

[7] K. Y. Yun and A. E. Dooply, "Pausible clocking based heterogeneous systems," in *IEEE Trans. on VLSI systems*, pp. Vol.7, no.4:482–487, Dec. 1996.

[8] J. Muttersbach, *Globally-Asynchronous Locally-Synchronous Architechtures for VLSI Systems*. PhD thesis, ETH Zurich, 2001.

[9] A. E. Sjogren and C. J. Myers, "Interfacing synchronous and asynchronous modules within a high-speed pipeline," in *Advanced Research in VLSI*, pp. 47–61, 1997.

[10] I. Sutherland and S. Fairbanks, "Gasp: A minimal fifo control," in *Proc. of ASYNC'01*, 2001.

[11] J. Mekie, S. Chakraborty, and D. K. Sharma, "Wrapper design for high-speed gals systems," Tech. Rep. 10-2003, IIT Bombay, 2003. Also available at http://www.ee.iitb.ac.in/uma/∼jrm/Tech01.ps.