Predictive Synchronization for DVFS-Enabled Multi-Processor Systems

Mark Buckler[†‡], Wayne Burleson[†‡]
AMD Research, Boxborough, MA.[†]
University of Massachusetts, Amherst, MA.[‡]
mark.buckler@amd.com, wayne.burleson@amd.com

Abstract

Technology scaling has enabled the number of cores within a System on Chip to increase significantly. Globally Asynchronous Locally Synchronous (GALS) systems using Dynamic Voltage and Frequency Scaling (DVFS) operate each of these cores on distinct and dynamic clock domains. Typically, the interfaces between these clock domains experience multi-cycle latency due to their use of "brute force" synchronizers. Improvements in system performance can be achieved by replacing these brute force synchronizers with predictive synchronizers, which experience less than a cycle of latency. Unfortunately, the predictive synchronizers proposed so far require high power delay lines, rationally related clocks, or over a thousand cycles of latency for every change in frequency. Here we present a modified predictive synchronizer without these limitations. Our techniques allow for uninterrupted data flow during gradual frequency change and only a 20 cycle pause in data flow after an instant frequency change. In addition, we present an alternative clock domain interface which achieves less than a cycle of latency with an average 15% reduction in throughput.

Keywords

Predictive Synchronizer, Globally Asynchronous Locally Synchronous, Dynamic Voltage and Frequency Scaling

1. Introduction

Global clocks have been proven to be un-scalable for today's many-core systems, leading to the increase of GALS systems such as the one shown in Figure 1. Fine-grained DVFS is now used to achieve power savings by dynamically tuning each part of the system to its current load. Each of these cores operating on different frequencies need to communicate, resulting in the need low-latency, high-throughput, and high reliability inter-clock-domain communication. Even Systems on Chip (SoCs) with a relatively few number of cores depend on high performance communication between clock domains for last-level cache access. Examples include AMD's Bulldozer core systems and Intel's Core i7.

Communicating across clock domains requires a system with flow control to prevent under-running or over-running the receiver as well as synchronizer circuits to mitigate the risk of metastability (thereby increasing reliability). This paper defines any system performing both of these functions as a Clock Domain Interface (CDI). The most common kind of CDI is an asynchronous FIFO [1]. The full and empty signals generated by the FIFO inform the two domains when it is possible to transmit or receive, while brute force synchronizers pass pointers between the domains. These synchronizers contain multiple flip flops connected in series. Each additional flip flop reduces the chance for metastability, quantified as the Mean Time Between Failures (MTBF). Of course, these additional flip flops also add latency, resulting in a tradeoff between latency and MTBF.

Predictive synchronizers transcend this tradeoff, offering near perfect MTBF while still maintaining less than a cycle of latency. This is achieved by exploiting certain relationships between the two clocks to predict their behavior. Each predictive synchronizer has limitations of its own however. Some designs rely on rationally-related clocks and cannot be used with other frequency combinations [2][3]. Others depend on large power hungry delay lines which are continuously matched to the clock period [4]. One system proposed by Dally called the Even Odd Predictive Synchronizer (EOPS) overcomes these problems, but introduces limitations of its own [5]. The EOPS needs certain parameters to be measured before it can operate. When clock frequencies change, the system needs to stop data flow while it re-measures. Unfortunately, measurement relies on the overflow of large counters, resulting in a measurement time of over a thousand cycles. For systems that apply DVFS frequently, this long stall in data flow is unacceptable. This prevents the use of this otherwise robust low-latency synchronizer in practice.

This paper proposes two techniques to solve this problem. The first technique allows for data flow to continue uninterrupted during gradual changes in clock frequency. The second technique is for systems with frequency changes that are too fast to be measured. This solution allows data flow to resume less than 20 cycles after a fast frequency change instead of the full re-measurement time. To demonstrate the wider implications of this work, we also propose an alternative CDI which uses these techniques.

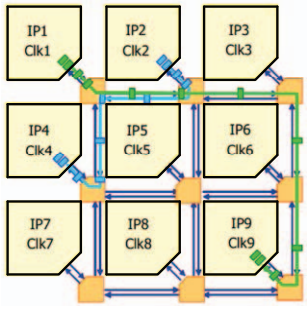

Figure 1: Example of a Multi-Processor GALS System [6]

1.1 The Even Odd Predictive Synchronizer

Since the DVFS strategies for predictive synchronization we propose build on and extend some aspects of the EOPS, it is necessary to understand its functionality completely. The EOPS leverages the periodic nature of the two clocks to estimate the transmit phase from the perspective of the

receive domain. These phase estimates are used to choose from one of two flip flops (Even or Odd) to sample in the receive domain. Figure 2 shows a high level view of the EOPS. Blocks modified by this work are shaded.

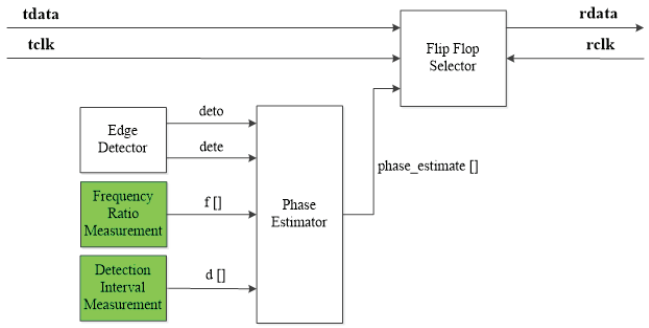

Figure 2: Internal EOPS structure

Phase estimation is achieved by first taking a digital measurement of both the ratio between the two clocks, and the size of a "detection interval". The detection interval represents the size of a delay line within a "phase detector" circuit. This alerts the system when two rising edges are within the detection region, meaning they are of the same phase at that time. Since this information needs to be sent through a brute force synchronizer, it is delayed by a certain number of receiver clock cycles.

The system needs current phase estimates however. This can be calculated by multiplying the ratio of the two clock frequencies by the number of delayed receiver clock cycles. This calculation determines how many transmit clock cycles (and fractions of cycles) have occurred since the system detected that the two domains were of the same phase. This in turn gives a phase estimate for the current receiver clock phase, noting that whereas they were within the range of the detection interval, they were not exactly of the same phase. This results in a level of built-in uncertainty in the phase estimate. To define this possible range of phases, upper bound and lower bound estimates are calculated by adding or subtracting the detection interval from the previous calculation [5].

2. Predictive Synchronizer DVFS Techniques

The phase estimation system used in the EOPS allows for its large reduction in synchronization latency. To function properly, the phase estimation system assumes that the value of the measured frequency ratio and detection interval remain constant. When DVFS is applied, these values need to be re-measured for each change in clock frequency. Naturally, this delay in communication for every change in frequency is detrimental for systems that apply DVFS aggressively at a fine-grain. The following two sections present solutions for two scenarios, specifically DVFS for gradual frequency change, or for fast (instant) frequency change, respectively.

2.1 Gradual Frequency Change

There are a number of reasons for gradual frequency change. For example, changing certain parameters in phase locked loops will result in a slow change in output from the previous frequency to the target frequency. Also, increasing the speed of a clock with a large distribution tree (such as in a large scale GPU) must be done slowly to avoid voltage drop in the power supply. Therefore, when gradual frequency change does occur, special care must be taken. We propose that instead of waiting until clocks have settled on their final values to re-measure, the system will measure continuously.

Both the measured frequency ratio and detection interval needed for the EOPS are represented as digital fractions. These digital fractions have a finite number of bits, meaning that their accuracy is limited. This accuracy limitation is accounted for in the design by being included when calculating the upper and lower bound phase estimations [5]. Because the system is built to handle this inaccuracy, another perspective would be that there is a certain range (the range of inaccuracy) in which the actual values of these factors can vary without causing failure in the system. If periodic measurement updates were sent to the phase estimator, and the actual values of the measured variables did not change so much as to go outside of this acceptable range in between updates, then the whole system can continue communication through clock frequency change.

To realize this idea, it is first necessary to generate periodic measurement updates. Basic measurement updates can be achieved by restarting both the frequency ratio and detection interval measurement circuits (same circuits in the Even Odd Synchronizer) each time they finish a measurement. Each time these circuits finish, they will provide updated measurements to the system, allowing it to adapt as clocks change. Unfortunately, the frequency of these updates is limited by the number of cycles the circuits need to finish the measurement.

To assess the adequacy of this design, the maximum speed that a clock can change in frequency in between updates may be calculated. This is the limiting factor that might prevent this design from practical application. If the measured fractions have b bits, then the range of inaccuracy is $\pm 2^{-(b+1)}$. A typical system might have a b of 10, therefore allowing for a variation of ± 0.000488. If we assume that both clocks are 1GHz, for the sake of illustration, then this means that one of these two clocks may change its frequency ±488 kHz and still stay within the measured range.

The time between updates is also determined based on the number of bits used to represent the measured fractions. This is because the number of bits determines the size of the counters that must overflow in the measurement blocks. Time between measurements is equal to 2^b receiver clock cycles, i.e. we must wait $2^{10} = 1024$ cycles in between each update for our typical system. This may also be called the cold start time, since this time must be spent on start up to begin estimating phases. When N measurement blocks are used the time between updates is (cold start time / N) as shown in Figure 3.

It is important to note that the measured values in each update will be based on the 1024 cycles before the update (the period which the system was measuring). As a result, these updates do not represent the frequency ratio and detection interval value at the exact time of the update. If we assume that the clock frequency is changing at a constant speed between updates, each measured update will represent

the frequency ratio and detection interval exactly half way between the start and stop of measurement. This measured value is used by the system until the next update. The time that the frequency needs to stay in the acceptable range is (time between updates) + (time between updates/2). For our typical system, we know that the receiver frequency needs to stay in the acceptable range for 1536 cycles. While this may be acceptable for some applications, others require faster frequency change. This necessitates faster measurement updates.

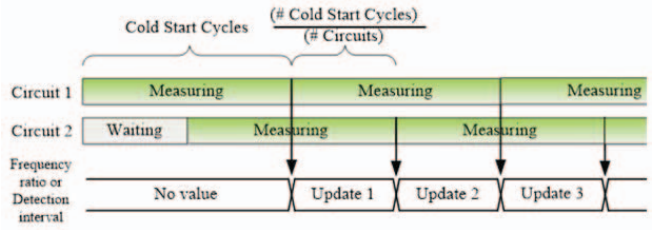

Figure 3: Continuous Measurement Strategy at Doubled Rate

Faster measurement updates may be achieved by including multiple measurement circuits in the design. These circuits are started in a staggered fashion and run in parallel, effectively hiding measurement latency as shown in Figure 3. Both the frequency ratio and detection interval can be measured in this manner.

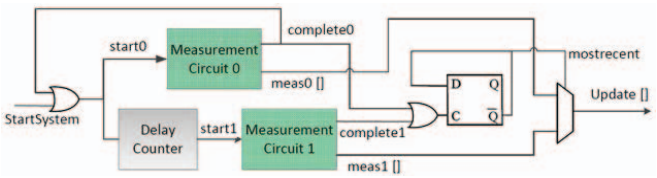

Figure 4: Continuous Frequency Measurement Circuit

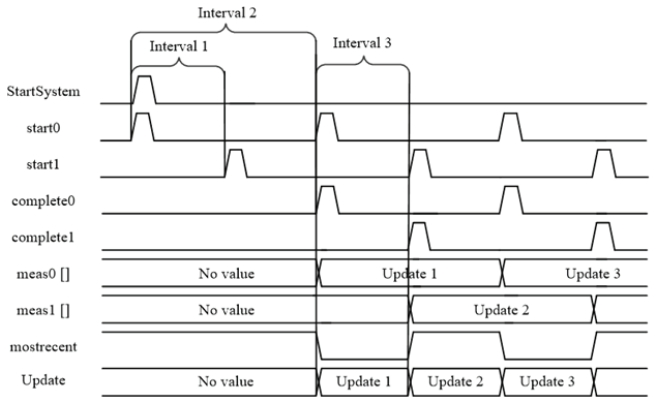

Figure 5: Continuous Frequency Measurement Timing

The circuit used to achieve this technique is shown in Figure 4 while the timing diagram in Figure 5 shows its behavior. The circuit is started with a *StartSystem* pulse, which initiates Measurement Circuit 0. This start pulse also enters a delay counter which delays the start signal by half of the measurement completion time (Interval 1). After this delay, Measurement Circuit 1 is started. Eventually, Measurement Circuit 0 finishes (measurement time is shown by Interval 2) and its output value is posted to *Update* as the most recent line is pulled low. This completion causes *start0* to be asserted, beginning the process all over again. After time interval 3, Measurement Circuit 1 finishes. This parallel circuit allows for updates to occur twice as fast as a single measurement structure, doubling the maximum frequency ramp speed. This method could be used for any number of additional measurement circuits, eventually allowing for an updated measurement every clock cycle if so desired.

2.2 Instant Frequency Change

In systems where clocks may change frequency very quickly, such as when digital frequency dividers are used, even parallel measurement circuits may not be fast enough. In this section, we present a solution which directly informs the predictive CDI when frequencies change and what they are changing to. When a frequency transition takes place the CDI pauses communication, reads in the new frequency value, calculates the new frequency ratio and detection interval directly (instead of measuring), and then waits for both domains to track before resuming communication.

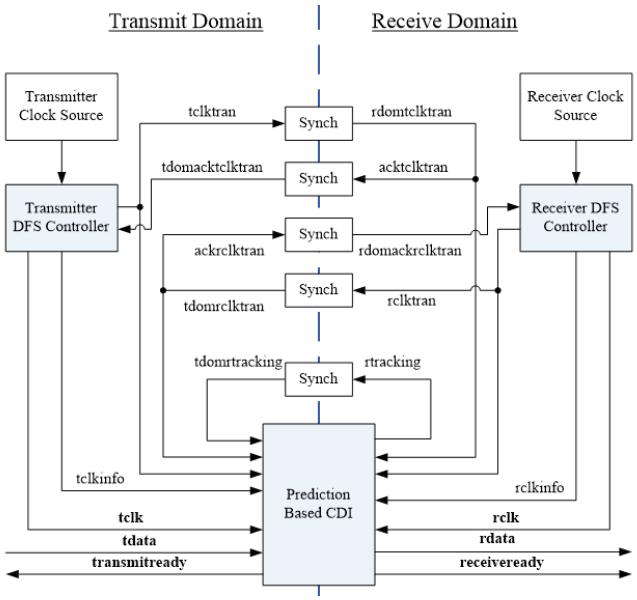

Figure 6: High Level Diagram of Fast DVFS Strategy

The system shown in Figure 6 uses a number of control signals to communicate the status during the process of a fast frequency change. Each DVFS controller supplies a clock (generated from the controller's clock source) and information about the clock to the Prediction Based CDI. This information could either be the specific frequency of the supplied clock or a code representing which frequency the clock currently is.

Each DVFS controller also has an output for requesting a frequency transition (*rclktran, tclktran*) and an input so that the controller knows when the opposite clock domain has acknowledged the frequency transition (*tdomacktclktran, rdomackrclktran*). These signals cross the clock domain boundary by using brute force synchronizers since the predictive CDI is off-line during frequency transitions. There are also signals for *rtracking* representing when the receive domain is ready. The two domains' clocks, data, and flow control signals (shown in bold at the bottom of the

figure) are the signals used during normal operation when communication is resumed.

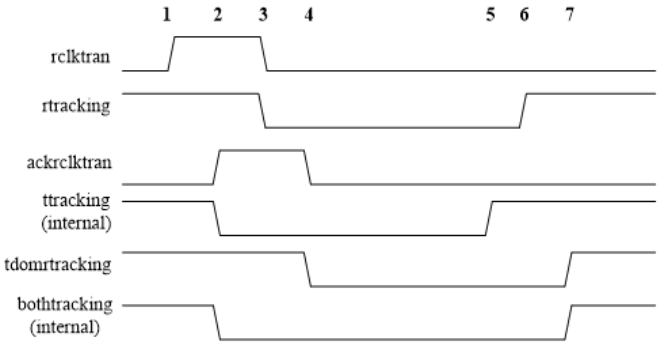

Figure 7: Fast Change in Receiver Clock Frequency Timing

Granted, this technique does require the CDI to pause communication for a certain period of time unlike the continuous measurement technique. To understand how long communication would need to be paused, we consider the timing behavior of the system. As seen in Figure 7, the receive domain informs the system of its need to change its frequency by asserting *rclktran* at time 1 as well as updating its *rclkinfo* to inform the CDI of the needed running frequency.

The *rclktran* signal propagates through the brute force synchronizer to the transmit domain by time 2. At this time the transmit domain acknowledges the change via a direct connection to the synchronized transition request signal and de-asserts its tracking flag (*ttracking*). Naturally this implies that both domains are no longer tracking and communication is paused. During this period *transmitready* is set low and *tdata* is kept at a "null" value which the receive domain will ignore.

For the system to resume, the following steps are taken. First, when the transmit domain acknowledges the change in frequency (time 2), it samples the new *rclk* info. This data is known to be constant since it has been a number of synchronizer cycles since the last change (the *tdomrclktran* signal acts as a data ready signal). The data is then used to calculate both the frequency ratio and detection interval necessary for phase prediction in the transmit domain. The edge detector in the transmit domain is also disabled to prevent the system from re-tracking before the receiver's clock is actually changed.

When the asserted *ackrclktran* signal is successfully synchronized to the receiver domain at time 3, the receiver DVFS controller completes the sequence by changing the receiver clock's frequency. This acknowledgement also triggers the receive domain to calculate its frequency ratio and detection interval which has been updated based on the new *rclk* information. With a new *rclk*, the phase estimator needs to re-track, so the *rtracking* flag is also de-asserted. Having completed its transition, the receiver clock's *rclktran* signal is de-asserted. At time 4 the de-assertion of both the *rclktran* and *rtracking* signals are received in the transmit domain. By this time instant, transmit domain knows that the receiver clock has been changed which prompts the transmit side phase detector to be re-enabled.

Once the receiver clock has been changed and the phase estimators are enabled in both domains, it is necessary to wait for the two domains to begin tracking. At time 5 the transmit domain begins tracking and at time 6 the receive domain begins tracking. When at time 7 the *rtracking* signal is synchronized to the transmit domain, the transmitter can resume communication.

The obvious question is how much time is necessary between stopping and starting of communication after a frequency change. The time range where communication is stopped starts at time 2 and restarts at time 7. During this time, control signals are passed between clock domains and the system waits for both domains to start tracking. Although different frequencies change the exact delay, the delay from two synchronizers comprise 8 cycles if 4 cycle synchronizers are assumed. Our experiments have shown that waiting for tracking rarely exceeds 12 cycles. This means that the system can re-start communication after only 20 cycles. This is a considerable improvement relative to the thousand cycles necessary for full re-measurement.

3. The Locally Controlled CDI

It is important to note that the usefulness of these DVFS enabled phase prediction techniques is not limited to the EOPS. Here, we describe an alternative Clock Domain Interface (CDI) which uses these DVFS enabled predictive phase estimator circuits. In this CDI, flow control is managed locally with no need for a FIFO.

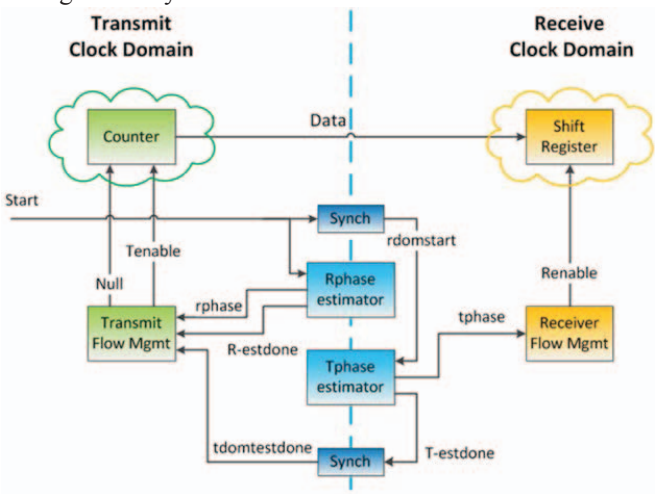

Figure 8: The Locally Controlled CDI

As seen in Figure 8, the Locally Controlled CDI uses a phase estimator for each domain. The phase estimates are then sent to the flow management logic which in turn controls the flow of data out of the transmit domain and into the receive domain. The clouds represent the entire transmit and receive domains respectively. The counter in the transmit domain and shift register in the receive domain are only used for testing the system.

The Locally Controlled CDI has two different modes of operation. One protocol is used when the receiver clock is faster while another protocol is used when the transmit clock is faster. The system decides which of these two protocols should be active by checking the ratio between the two frequencies provided by the phase estimator (this is one of the necessary values provided by the DVFS techniques shown in previous sections).

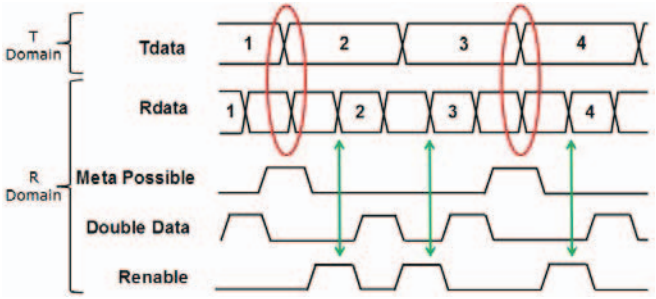

Figure 9: CDI protocol when the receive clock is fastest

Flow control and metastability avoidance is all contained within the receive domain when the receive clock is fastest. This protocol generates three control signals: *Meta Possible*, *Double Data*, and *Renable*. The signal *Meta Possible* informs the system when sampling on a particular edge has the risk of resulting in metastability. To avoid metastability the system needs to avoid sampling data when that data is changing. This is defined as the case when the rising edge of the transmit clock violates the receive domain's setup time or hold time. This situation can be avoided by checking the calculated phase estimates while preparing to sample.

As described in previous sections, the prediction circuits give upper bound and lower bound phase estimates as binary fractions of the transmit clock period. This binary fraction includes bits to the left and to the right of the decimal point. The bits to the right of the decimal point represent where the rising edge of the receive clock falls, within a single transmit clock cycle. The flow control logic checks these bits to see if the phase estimate overlaps the predefined keep out region around the rising edge. Figure 10 shows examples of safe sampling (a) and unsafe sampling (b) on transmit domain phase circles. The X and 1-X terms here are predefined as the keep out region (the setup and hold time). When there is overlap, the *Meta Possible* signal is asserted since the edges may be too close. This can be seen in Figure 9 where the ovals point out the transmit clock and receive clock are too close. Each of these edges is accompanied by a high *Meta Possible* signal.

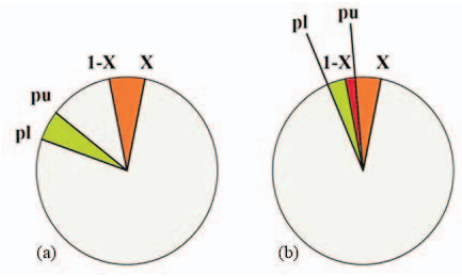

Figure 10: Phase estimation for safety detection

Of course, flow control is also important for a CDI. In the case where the receive clock is fastest the flow control logic must prevent the receiver from sampling data that it has already sampled. This is accomplished with the *Double Data* signal. For this the bits to the left of the decimal point of the phase estimate are utilized. These bits represent which transmit clock cycle the possibly sampled data belongs to. To generate the *Double Data* signal, the phase from the current receive clock rising edge and the phase from the previous receive clock rising edge are compared. If the two phases are in the same transmit clock cycle, then the data was already sampled on the previous edge and there is no need to sample again. If both the *Meta Possible* and *Double Data* signals are low, then *Renable* is high and the receive domain knows that it is the right time to sample.

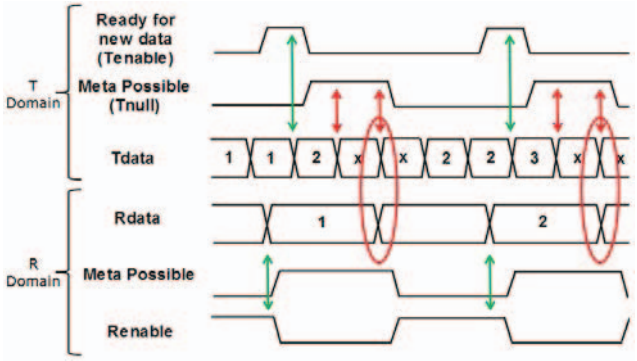

Figure 11: CDI protocol when the transmit clock is fastest

A different protocol is used if the transmit clock is faster than the receive domain (see Figure 11). For this protocol flow control signals are used in both domains. In the receive domain, metastability is predicted in the same way as previously described. Flow control is managed in the transmit domain for this protocol, setting *Renable* high as long as *Meta Possible* is low.

In the transmit domain, both an enable signal as well as a *Tnull* signal are used. The tenable signal represents when the receive domain is ready for new data. This is managed in a similar manner to the *Double Data* signal in the receive clock faster protocol. If the phase estimates of the current transmit clock edge and the previous transmit clock edge are in the same receive clock cycle, then the receive domain has not yet had the chance to sample the data.

Metastability management in the transmit domain is handled differently for this protocol. Data would be lost if the transmit and receive domains disagreed about if a given receive edge would cause metastability. Disagreement is possible since the domains do not send control signals between domains during operation and manage their own phase estimators with slightly different properties. To prevent this problem from occurring, the transmit domain nulls out data before and after an edge that it detects may possibly cause metastability. The transmit domain also increases its keep-out region slightly to make sure that the transmit domain will always be more protective of possible metastability than the receive domain. This way the only case for a disagreement is when the transmit domain expects metastability but the receive domain does not. In this situation the receive domain only sees the null value and knows to ignore this data which is guaranteed to be safe.

It is also important to note one of the weaknesses of this CDI. This CDI cannot be used when the transmitter and receiver clocks are rationally related as tracking is impossible when clocks are out of phase since rising edges will never occur close to each other. With no rising edges close to each other the system can never calibrate. This limitation restricts the range of frequencies that the CDI can be used with. Also, for the purposes of simplification, the implemented CDI is limited to working with clocks that are no more than twice the other or less than half the other. This

could be changed however by including more bits to the left of the decimal point when calculating phase estimates and making few changes to the logic.

4. Discussion

These designs have been implemented and verified in Verilog RTL. Synthesis was performed using TSMC 28nm technology standard cells. The test setup consisted of a 5 bit counter in the transmit domain to generate sample data and a shift register in the receive domain to hold the incoming data (as shown in Figure 8). The transmit clock's frequency was set at 1GHz while the receive frequency was swept from 500MHz to 2GHz at 1GHz intervals. This was repeated twice, once with gradual changes in frequency using the circuit shown in Figure 4, and then once with instant changes in frequency using the circuit shown in Figure 6.

Setup and hold time checks were used to verify no risk of metastability during all of operation. The contents of the shift register were monitored to verify that the data was passed without doubles or misses. Transfer of data was successful with the notable exception of rationally-related frequencies. Completion time was also measured to determine the throughput of the system.

Table 1: CDI Comparison

Metric	*Brute Force*	*EOPS*	*Locally Controlled*
Latency	4 cycles	**~0.5 cycles**	**~0.5 cycles**
MTBF	Limited	**~Infinite**	**~Infinite**
Layout Size	6606 μm^2	4473 μm^2	**3122 μm^2**
Throughput	**Maximum**	**Maximum**	~15% Reduction

As shown in Table 1, when compared to the Brute force CDI (an asynchronous FIFO using brute force synchronizers) and the EOPS CDI (an asynchronous FIFO using the EOPS), the Locally Controlled CDI has the best latency, Mean Time Between Failures (MTBF) and layout size. Both the EOPS CDI and the Locally Controlled CDI have the same latency and MTBF since they both use the same phase prediction circuits. The average latency of half a cycle is because the only waiting time is between the rising edge of the transmit clock with new data and the rising edge of the receive clock sampling this new data.

The MTBF for the EOPS CDI and the Locally Controlled CDI is near perfect since the only brute force synchronizers (the source for a possible metastability failure) in these CDIs are in the measurement circuits. These brute force synchronizers are out of the critical path and therefore can be made arbitrarily long to achieve almost infinite MTBF. Since the critical path for the brute force CDI goes directly through its brute force synchronizer it must make a compromise between reliability and latency resulting in a sub-optimal MTBF. In terms of layout area, the EOPS CDI is smaller than the brute force CDI since its reduced latency allows it to have a smaller FIFO, while the Locally Controlled CDI has no need for a FIFO and is therefore even smaller. The one disadvantage to the Locally Controlled CDI is its reduction in throughput, which was verified experimentally. This reduction in throughput is due to the system not sampling data when metastability is possible, effectively missing a cycle.

It should also be mentioned that while the Brute force CDI demonstrates poor latency, MTBF, and area, it does not require any extra circuitry for changes in frequency. Area estimates shown in Table 1 do not include area for extra measurement circuits for gradual frequency changes. Each additional measurement circuit (including both frequency ratio and detection interval) adds 306 μm^2 to the layout size.

The effect of jitter in each domain on the system must also be considered. If precautions are not taken, it is possible for phase predictions to exceed tolerance. If the nature of the jitter is known however, then this can be prevented by artificially increasing the size of the measured detection interval [5]. This technique applies for all uses of the phase prediction circuits (EOPS, Locally Controlled CDI, etc.).

The Locally Controlled CDI's inability to operate with clocks that are rationally related might also be mitigated. To allow for operation over a wider range of frequencies, the Locally Controlled CDI could be augmented with a CDI specifically design for rationally related frequencies [2][3]. Also, the work presented in this paper manages changes in clock frequencies, not changes in voltage. For this, level shifters must be applied to incoming and outgoing signals.

5. Conclusions

The DVFS techniques proposed in this paper may be applied to current systems (such as the EOPS) to enable low latency, high throughput, and high reliability inter-clock domain communication in DVFS enabled GALS SoCs. Due to their digital nature, these circuits are suitable for standard design and verification flows (including PVT management). Interruptions due to frequency changes are either non-existent (for gradual changes) or less than 20 cycles (for fast or "instant" frequency changes) for the upgraded EOPS CDI or the new Locally Controlled CDI. This reduction in GALS system overhead improves system performance as a whole, and may allow for the use of more aggressive low-level DVFS strategies.

In addition, the CDI presented in this paper exhibits low latency, high reliability, and a small layout footprint, while taking a minor reduction in throughput. This CDI is well suited for applications where latency, reliability, and size are critical but throughput is of lesser importance, such as in networks for on chip monitoring.

6. References

[1] Ginosar, R., "Metastability and Synchronizers: A Tutorial," IEEE Design & Test of Computers, 2011.

[2] Chabloz, J., Hemani, A., "Low-Latency Maximal-Throughput Communication Interfaces for Rationally Related Clock Domains," IEEE Tran VLSI Systems, 2011.

[3] Wang, R., Wang, H., Fan, B., Yang, L., "RIRI scheme: A robust instant-responding ratiochronous interface with zero-latency penalty," ISCAS, 2011.

[4] Frank, U., Kapshitz, T., and Ginosar, R., "A Predictive Synchronizer for Periodic Clock Domains," J. Formal Methods in System Design, 2006.

[5] Dally, W., Tell, S., "The Even/Odd Synchronizer: A Fast, All-Digital Periodic Synchronizer," ASYNC, 2010.

[6] Yakovlev, A., Vivet, P., Renaudin, M., "Advances in asynchronous logic: From principles to GALS & NoC, recent industry applications, and commercial CAD tools," DATE, 2013.