2010 IEEE Symposium on Asynchronous Circuits and Systems

The Even/Odd Synchronizer: A Fast, All-Digital Periodic Synchronizer

William J. Dally*†, Stephen G. Tell†
*Stanford University, †NVIDIA Corporation

***Abstract*—We describe an all-digital synchronizer that moves multi-bit signals between two periodic clock domains with an average delay of slightly more than a half cycle and an arbitrarily small probability of synchronization failure. The synchronizer operates by measuring the relative frequency of the two periodic clocks and using this frequency measurement, along with a phase detection, to compute a phase estimate. Interval arithmetic is used for the phase estimate to account for uncertainty. The transmitter writes a pair of registers on alternating clock cycles and the receiver uses the estimate of the transmitter's phase to always select the most recently written value that is safe to sample. We show how to incorporate this design into a FIFO to give a fast periodic synchronizer with flow control. We present a number-theoretic argument that the synchronizer works for all frequency combinations. An implementation of the synchronizer using standard cells is also presented.**

I. Introduction

Many digital systems are organized into multiple synchronous clock domains each with separate, periodic clocks. For example, a modern microprocessor and its memory controller may operate off completely separate clocks. A synchronizer is required to pass signals from one clock domain to another to prevent synchronization failure. Most existing systems use a FIFO synchronizer [1] to pass signals between clock domains. These incur a significant area overhead for the FIFO memory. They also add several cycles of delay as the Gray-coded input and output pointers of the FIFO must be synchronized through multiple flip-flops to move them across clock domains.

Synchronizers that exploit the periodic nature of the phase relationship to avoid the area and delay overhead of a FIFO have been described by Stewart & Ward [2][3], Dennison et al. [4], Dally & Poulton [1], and Frank et al. [5]. However these systems are either limited to clocks with nearly identical frequencies [4] or require calibrating analog delay lines to match the clock period [2][3][5]. Creating such matched delay lines is costly in area and power and increases system jitter. Sarmenta et al. [6] have described a periodic synchronizer for rationally related signals that avoids the use of analog delay lines. However this synchronizer cannot handle general periodic signals. Chakraborty and Greenstreet [7] use a single-stage FIFO to perform mesochronous synchronization with skew tolerance and then use rate multipliers to apply this circuit to other types of clocking.

In this paper we present an all-digital synchronizer that exploits the predictability of periodic clocks to achieve low (0-1 cycle) delay, and small area. Our synchronizer operates by digitally measuring the relative frequency between the transmit and receive clocks and then uses this frequency measurement to digitally estimate the relative phase of the two clocks. This phase estimate is used to predict when it is safe to directly sample with a receive clock signals synchronous with a transmit clock. Interval arithmetic is used to track the accuracy of the phase estimate.

The contributions of this paper include:

1) A digital method of, within one clock domain, estimating the frequency and predicting the phase of a second clock domain.
2) The design of the even/odd synchronizer that uses a pair of registers and the phase prediction to provide safe, low-latency synchronization.
3) The integration of the even/odd synchronizer into a FIFO synchronizer to provide low-latency synchronization with flow control.
4) An analysis of the even/odd synchronizer including a number theoretic argument of its correctness.

The remainder of this paper describes our digital periodic synchronizer in more detail. The symbols used in the paper are summarized in Table I. Typical values are indicated where appropriate. In Section II we show how a phase estimate can be used to construct a fast, simple forward synchronizer that moves a parallel signal from one periodic clock domain to another. We show how this synchronizer can be employed in a FIFO synchronizer to provide synchronization and flow control in Section III. Section IV describes how our system measures the relative frequency of the two clocks and uses this estimate to generate a phase estimate with error bounds. A state diagram for the synchronizer is described in Section V. We analyze the behavior of the system and give an informal argument of its correctness in Section VI. We discuss the properties of the synchronizer in Section VII and conclude in Section VIII.

II. Even/Odd Forward Synchronizer

Fig. 1 shows a forward synchronizer that moves a parallel signal forward - from the transmit clock (tclk) domain to the receive clock (rclk) domain - safe from synchronization failure, but without flow control. The transmitter writes data alternately to a pair of registers: to the E register on even tclk cycles and to the O register on odd tclk cycles. The receiver selects the most recently written register, E or O, that is safe to sample at the end of the current rclk cycle. This selection is based on the predicted tclk phase at the end of the current

978-0-7695-4032-0/10 $26.00 © 2010 IEEE
DOI 10.1109/ASYNC.2010.20

TABLE I
SYMBOLS USED IN THIS PAPER

Sym	Description	Units	Typ
A	Advance in the phase estimator, $A = S + 1$	cycles	5
b	Fraction bits in the frequency and phase estimates	bits	10
C	Upper bound on denominator D, $C = \lceil \frac{1}{2d} \rceil$		6
d	Half-width of phase detection region , $d = \frac{t_d}{t_{tcy}}$	UI	0.13
D	Denominator of a rational fraction $\frac{N}{D}$		
e	Difference from "nearest" rational fraction, $f = \frac{N}{D} + e$		
f	Relative fractional transmit clock frequency, $f = \frac{f_T}{f_R} \bmod 1$. (For purposes of keeping even and odd cycles, we keep $f \bmod 2$.)	UI/Cy	1.33
f_g	Guard-band frequency ($f_g = \frac{g}{A}$), the frequency below which a detection will occur at least A cycles before a keep-out event	UI/Cy	0.02
f_j	Jitter frequency.	Hz	
f_R	Receive clock (rclk) frequency	Hz	7.5e8
f_T	Transmit clock (tclk) frequency	Hz	1e9
g	Guard band between detection and keep-out regions, $g = d - x$	UI	0.1
k	Threshold for entering plesiochronous mode, i.e. we enter plesiochronous mode when $\lvert \mathsf{pu} - \mathsf{pl} \rvert > k$	UI	0.50
N	Numerator of a rational fraction $\frac{N}{D}$		
ϕ	Phase	UI	
p	Estimated phase	UI	
S	Synchronizer delay	cycles	4
t_d	Delay time used in phase detector	ps	130
t_{jh}	Bound on absolute value of high-frequency jitter (jitter with $f_j > \frac{f_R}{A}$). Peak-to-peak jitter is twice this value.	ps	30
t_{jm}	Bound on absolute value of medium-frequency jitter (jitter with $f_g f_R < f_j < \frac{f_R}{A}$)	ps	30
t_{rcy}	Receive clock (rclk) cycle time	ps	1333
t_{tcy}	Transmit clock (tclk) cycle time	ps	1000
t_x	Keep-out time ($t_x = t_s + t_h + t_{jh}$)	ps	60
x	Half-width of keepout region (UI), $x = \frac{t_x}{2t_{tcy}}$	UI	0.03

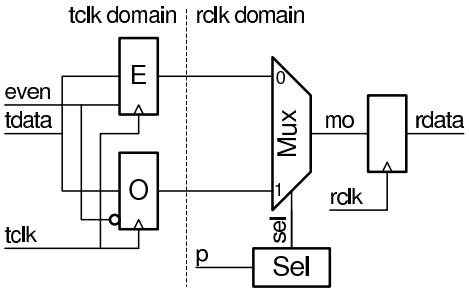

Fig. 1. An even/odd forward synchronizer

rclk cycle, p. Because it operates by discriminating even and odd tclk cycles, we refer to this periodic synchronizer design as an even/odd synchronizer or E/O synchronizer.

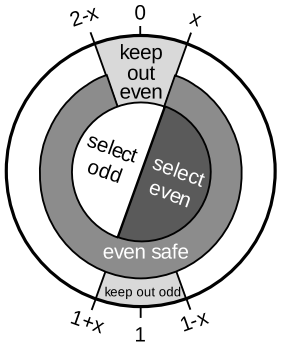

Fig. 2. Phase circle showing even and odd keep-out regions and region in which the even register is selected

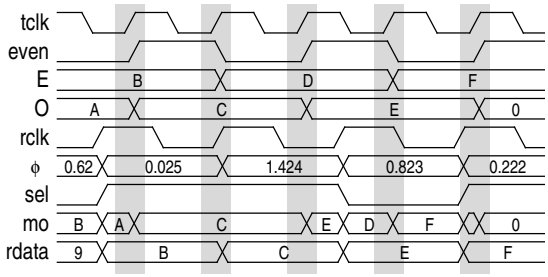

Fig. 3. Waveforms showing operation of the synchronizer of Fig. 1.

The transmitter phase, ϕ, is a real number in the range $\phi \in [0, 2)$ which we can visualize on a phase circle as shown in Fig. 2. Odd clock cycles have a phase $\phi \in [0, 1)$, and even clock cycles have $\phi \in [1, 2)$. An odd (even) clock cycle ends in an odd (even) clock edge, and the signal even is high during even clock cycles (i.e., when $\phi \in [1, 2)$) . To avoid synchronization failure, is important that the receiver not sample the even (E) register during the even keep-out period $\phi \in [2-x, x]$ and that the O register be avoided during the odd keep-out period $\phi \in [1 - x, 1 + x]$. These keep out periods are shaded light grey in the figure. The keep out window, with width $2x$, represents the setup and hold window of the sampling flip-flop. The width of these keep-out regions are exaggerated in the figure. A typical 40nm flip-flop has a keep-out window of about 60ps or just 6% of a 1GHz clock period.

To meet our rule of selecting the most recently written register that is safe to sample, the selection logic selects the even (E) register when $\phi \in (x, 1 + x]$ as shown by the dark grey shaded arc in Fig. 2. The E register is selected as soon as the phase clears the even keep out region at $\phi = x$. The E register is safe to sample during the large medium grey arc shown in the figure, $\phi \in (x, 2 - x)$ (everywhere but the even keep out region). However it is only the most recently written safe register up until $\phi = 1 + x$. When $\phi \in (1 + x, x]$ the odd register (O) is the most recently written safe register.

A timing diagram illustrating typical operation of the forward synchronizer in the case where tclk is faster than rclk is shown in Fig. 3. The shaded areas of the figure show the keep-

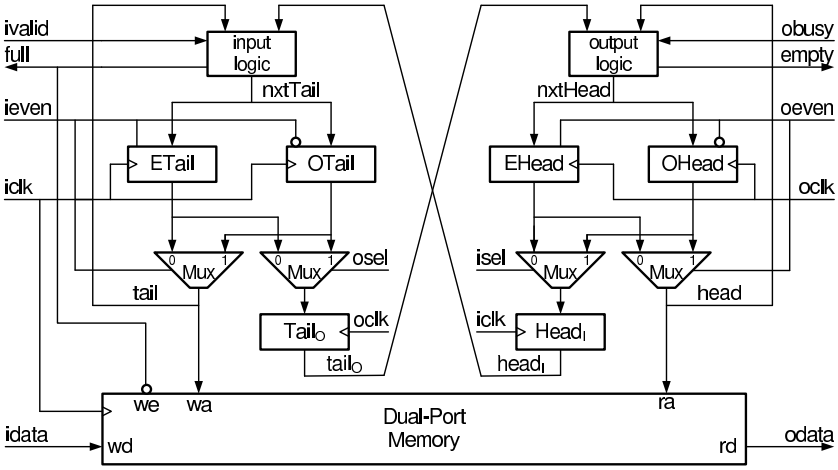

Fig. 4. FIFO Synchronizer using E/O Synchronizers for Head And Tail Pointers

out regions of the transmit clock (tclk).[1] Waveform ϕ shows the phase that tclk will have at the end of each rclk cycle. The timing diagram shows that the circuit always samples the most recently written register that is not in an unsafe region. The rising first edge of rclk samples the value "B" from register E because this edge occurs in an odd cycle of tclk. The next edge samples "C" from register O because this rclk edge is in the unsafe region, only 2.5% of the way into a tclk cycle. The third edge samples "E" from register O because it occurs safely in an even tclk cycle. Finally, the last rclk edge samples "F" from register E because it occurs in an odd cycle, just before the odd keep-out region of tclk. Note that value "D" is never sampled because tclk is faster than rclk. Conversely, if tclk were slower than rclk, some values would be sampled multiple times. Ensuring that every value is sampled exactly once requires a synchronizer with flow control, such as the FIFO synchronizer described in Section III.

So far we have been assuming that the receiver knows the tclk phase, ϕ. In practice, the receiver uses an estimate of tclk phase, p. To account for the estimation error, $\epsilon = |\phi - p|$, we either add a guard band to our detection of keep-out region, or we compute our phase estimate using interval arithmetic as described in Section IV, giving a bound on ϵ. When using interval arithmetic, we make our selection decision using the lower-bound of the phase (pl), since this always selects the most recently written register that we are sure is safe to sample.

III. A Low-Latency FIFO Synchronizer

A low-latency synchronizer with flow control can be realized by using E/O synchronizers to synchronize the head and tail pointers of a conventional FIFO synchronizer as shown in Fig. 4. This eliminates the high delay of the brute-force synchronizers typically used for pointer synchronization and also avoids the need to Gray-code the pointers.

[1]For illustrative purposes, the the keepout region shown in Fig. 3 is much larger than the typical value of x.

Fig. 4 shows a FIFO synchronizer realized using two E/O synchronizers that are combined with the head and tail pointer registers. The FIFO uses a dual-port memory that is written synchronously and read asynchronously to hold data in transit. For small FIFOs, this memory is implemented as a flip-flop or latch array. Larger FIFOs use a RAM or register file macro. The FIFO memory is written and the tail pointer incremented on the rising edge of the input clock (iclk) when input valid (ivalid) is true and full is false. The head pointer selects the value at the head of the FIFO to appear at the output port of the memory. The head pointer increments on the rising edge of the output clock (oclk) when empty is false and output busy (obusy) is false. The tail pointer and full logic are in the iclk domain and the head pointer and empty logic are in the oclk domain.

A pair of E/O synchronizers moves the head and tail pointers between the two clock domains. For the tail synchronizer tclk=iclk and rclk=oclk while for the head synchronizer tclk=oclk and rclk=iclk. The tail synchronizer keeps even (ETail) and odd (OTail) versions of the tail pointer. One multiplexer, controlled by ieven selects the current tail pointer for use in the iclk domain. A second multiplexer, controlled by osel selects the most recently written tail register that is safe to sample in the oclk domain. A frequency estimator, a phase detector, and a phase estimator, as described in Section IV are used to generate osel. A second copy of the frequency estimator, phase detector, and phase estimator are use to generate isel.

Compared to the conventional approach of using brute-force synchronizers to pass the head and tail pointers between clock domains, using even/odd synchronizers has advantages of speed and simplicity. The latency of the FIFO synchronizer is reduced because the even/odd synchronizer has a delay of 0.5 cycles on average, compared to a brute-force synchronizer with a delay of S+0.5 cycles (typically 4.5 cycles), where S is the delay of a brute-force synchronizer. The design is also simpler because the head and tail pointers can be kept in

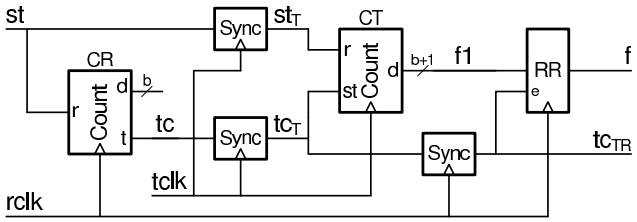

Fig. 5. Frequency measurement block diagram

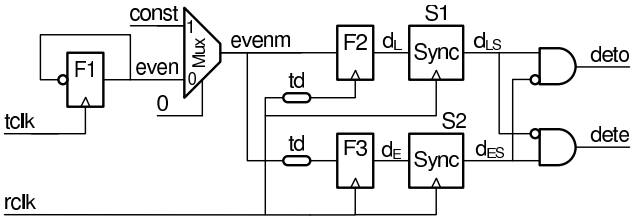

Fig. 6. Phase Detector

binary form. With conventional brute-force synchronizers, the pointers must be Gray-coded to prevent more than a single bit from changing at one time.

IV. Frequency and Phase Estimation

The synchronizer of Fig. 1 depends on having an accurate estimate of the transmit clock phase at the end of each receive clock cycle. This estimate is generated by first measuring the frequency of the transmit clock relative to the recieve clock and then using this frequency estimate, along with a phase detector, to generate a phase estimate. The phase estimate is computed using interval arithmetic to maintain an accurate error bound on phase.

A. Frequency Estimation

Fig. 5 shows the block diagram of the frequency measurement unit which uses a pair of counters to compute f, the frequency of the transmit clock relative to the receive clock. The frequency measurement process is initiated by a start signal, st. The rising edge of st resets the receive counter (CR). The start signal is also passed into the transmit clock (tclk) domain through a brute-force synchronizer, producing signal $\mathsf{st_T}$ which is used to reset the transmit counter (CT). When the receive counter reaches a terminal count (e.g., a count of 1023 for a $b = 10$-bit counter) signal tc is asserted and is synchronized into the tclk domain. This synchronized terminal count signal,$\mathsf{tc_T}$, stops the transmit counter. The delay of the st and tc synchronizers are balanced (± 1 cycle of uncertainty) so that the final count out of CT reflects the number of tclk cycles that occurred during 2^b receive clock (rclk) cycles, i.e., the relative frequency of the transmitter, $f = f_T/f_R$. The terminal count signal is synchronized back into the rclk domain to produce signal $\mathsf{tc_{TR}}$ which indicates when the frequency measurement f is ready and enables its capture in the result register (RR).

Counter CT produces a $b+1$ bit result so that f is generated modulo 2. It is a fixed-point number with one bit to the left of the binary point and b bits to the right. We compute the transmit frequency estimate modulo 2 rather than modulo 1 so that the phase estimator (described below) can track whether the transmitter is in an odd or even clock cycle.

There are three brute force synchronizers in the frequency measurement block of Fig. 5. These synchronizers are used only once - when frequency is measured after reset. All of these synchronizers are off of the critical path, so their delay can be made arbitrarily high to achieve an arbitrarily low probability of synchronization failure. Typically a delay S of four or five clock cycles suffices to give a failure probability of less than 10^{-40}.

The start signal and terminal count synchronizers each introduce one cycle of uncertainty in the frequency measurement. Hence the output of the frequency measurement block is accurate to ± 1 LSB, i.e., $\pm 2^{-b}$.

B. Phase Detection

The phase detector, shown in Fig. 6, detects when a transition on a transmit data signal falls in a window of $\pm t_d$ around the receive clock edge. The phase detector samples transmit signal evenm (signal even after a multiplexer delay) which toggles every cycle. This signal is high during even tclk cycles and low during odd tclk cycles. The timing of flip-flop F1 and the multiplexer match the timing of the E and O registers and the multiplexer of Fig. 1 so that transitions of evenm occur with the same timing as transitions of mo (in Fig. 1) except for component mismatches.

Flip-flop F2 samples evenm with rclk delayed by t_d giving $\mathsf{d_L}$, a sample of evenm t_d after the rising edge of rclk, i.e., a late sample. An early sample, $\mathsf{d_E}$ is produced by F3 which samples evenm delayed by t_d. If an edge of evenm occurs between t_d before rclk and t_d after rclk, the values sampled by F3 and F2 will be different. The early and late samples are synchronized to the receive clock domain by a pair of brute-force synchronizers generating synchronized early and late samples $\mathsf{d_{ES}}$ and $\mathsf{d_{LS}}$ respectively. Differences between the synchronized early and late samples are detected by a pair of AND gates. If the early sample is high and the late sample low, an even edge of tclk (one that ends an even cycle) is detected and dete is asserted. If the early sample is low and the late sample is high, an odd edge of tclk is detected and deto is asserted.

There are two brute-force synchronizers in the phase detector that operate every cycle of rclk. These synchronizers, however, like those in the frequency measurement unit, are off the critical path, so their delay can be made large to make the frequency of synchronization failure arbitrarily small. A combined delay, S, of 4 or 5 cycles for the sampling flip-flop plus synchronizer is typically adequate to maintain a failure frequency less than 10^{-40} Hz.

The delay lines in the phase detector are typically realized by an even number of inverters connected in series. To initialize the phase estimator, as described below, we need to bound the value of t_d. While it is possible to compute a

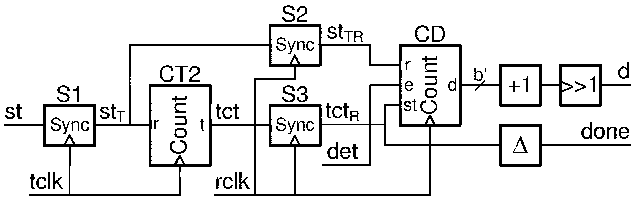

Fig. 7. Circuit to measure detection interval, d

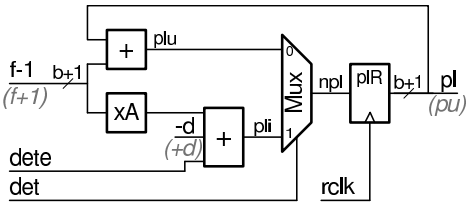

Fig. 8. Logic for tracking lower (upper) bounds on phase estimate

worst-case upper bound on t_d, we can get a more accurate phase estimate if we measure the instantaneous value of t_d and then add a guard band to this measurement to account for variation of t_d with temperature and voltage.

If the tclk and rclk are not rationally related, we can measure $2d = 2t_d/t_{tcy}$ by detecting the fraction of tclk cycles that result in a detection. This can be accomplished at the same time we are doing our frequency measurement. If the two clocks are not rationally related, edges of rclk will be uniformly distributed across the phase of tclk. A phase detection will occur $2d = 2t_d/t_{tcy}$ of the time. By counting the number of phase detections that occur in $2^{b'}$ tclk cycles, a measurement of d is obtained in a form directly usable by the phase estimator. A guard band is added to this estimate to produce a bound on d that accounts for voltage and temperature variation and for medium-frequency jitter (Section VII-C).

The circuit of Fig. 7 operates by counting the number of tclk cycles during which det (det = dete $\vee$ deto) is true during the $2^{b'}$ tclk cycles it takes for counter CT2 to reach its terminal count. This gives $2d$ as a b'-bit binary fraction. We increment the output of CD to give an upper bound on $2d$. We may add an additional value at this point (not shown) to provide guard band as described above. Right-shifting this number by one bit position gives d. The done signal indicates when the measurement of d is complete.

To give an accurate estimate of d, the width b' of counters CT2 and CD must be made large enough to uniformly sample the phase space for all $f \geq f_g$. To estimate d with an accuracy of ϵ requires that we sample at least $1/\epsilon$ phase cycles, or at least $\frac{1}{\epsilon f_g}$ tclk cycles. For example, to get an accuracy of 10% ($\epsilon = 0.1$), with $f_g = 0.02$ (Table I) requires at least 500 cycles, and hence a b' of at least 9 bits. For frequencies $f < f_g$ the system will operate in plesiochronous mode and the accuracy of d is not critical.

The circuit shown in Fig. 7 only uniformly samples tclk phase if the clocks are not rationally related. If they are rationally related then the receive clock repeatedly visits the same D (denominator of rational ratio in reduced form) points on the unit phase circle. If D is large enough, this is sufficient. The estimation error is less than $1/D$. For small D, d can be measured using an independent frequency source - for example a ring oscillator - to drive the CD counter.

C. Phase Estimation

Fig. 8 shows the phase estimation logic. Two copies of this logic are required to produce the lower (pl) and upper (pu) bound estimates of tclk phase. When the phase estimates are valid, the transmitter phase is known to be in the interval $[\mathsf{pl}, \mathsf{pu}]$. On each phase detection, the phase estimates are initialized to $[-d, d]$, because the phase is known to be in this interval during a detection, the even bit (msb) is set if an even edge was detected, and then the phase estimate is advanced in time by $A = S + 1$ cycles. On cycles when a detection does not occur, the phase estimates are updated by adding $f - 2^{-b}$ and $f + 2^{-b}$ to pl and pu respectively. We use $f - 2^{-b}$ and $f + 2^{-b}$ for these updates to account for the ± 1 LSB uncertainty in the frequency estimate.

The A-cycle time advance is accomplished by adding A times the frequency estimate ($f - 2^{-b}$ or $f + 2^{-b}$) to the initial phase. The advance A here is $S + 1$ where S is the delay of the synchronizers (including the sampling flops) in the phase detector. Advancing by S cancels the delay of the phase detector, and advancing one more cycle makes the phase estimate reflect the tclk phase at the end of the current rclk cycle (rather than at the beginning).

The operation of the phase estimator is more easily expressed as verilog code:

```
always @(posedge rclk) begin
  pu <= det ? {dete, bzeros} + d + (f+1)*A
            : pu+f+1 ;
  pl <= det ? {dete, bzeros} - d + (f-1)*A
            : pl+f-1 ;
end
```

We select the most recent safe register to sample based on the lower bound of the transmit phase, pl. When $\mathsf{pl} \in (x, 1+x]$ we set sel=0 to sample the even (E) register. Otherwise we sample the odd (O) register. Tracking the upper bound of the transmit phase allows us to determine when our phase estimate is no longer accurate enough to be used. The phase estimation is no longer useful when $\mathsf{pu} - \mathsf{pl} > 1 - 2x$. We can still operate the synchronizer in this state using plesiochronous mode as described in Section V. In practice the synchronizer should enter plesiochronous mode when $\mathsf{pu} - \mathsf{pl} > k = 0.5$. A smaller bound may be used to reduce synchronizer delay.

V. Synchronizer States

A state diagram for the fast periodic synchronizer is shown in Fig. 9. The states are described in Table II.

On reset the synchronizer enters the frequency acquisition (FA) state. It starts its pair of counters to measure the frequency of the "other" clock. During this state the synchronizer checks to see if there is a phase detection (phase falling into

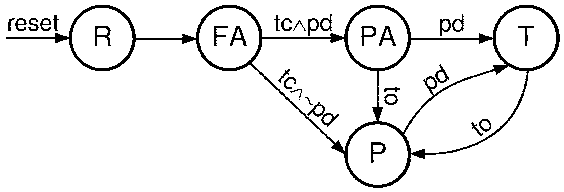

Fig. 9. Synchronizer State Diagram

TABLE II
SYNCHRONIZER STATE DESCRIPTIONS

R	Reset	Start frequency measurement process
FA	Frequency Acquisition	Wait for frequency measurement to complete. Record if a phase detection occurs during this period. If no phase detection → Plesiochronous Otherwise → Phase Acquisition
PA	Phase Acquisition	Wait for phase detection (pd). Time out → Plesiochronous Otherwise → Tracking
T	Tracking	Track estimated phase If $\mathsf{pu} - \mathsf{pl} > k$ → Plesiochronous
P	Plesiochronous	Operate in Plesiochronous mode If phase detection → Tracking

the detection region). If there was no phase detection during the frequency estimation, then the clocks are plesiochronous or rationally related ($f = N/D$) (or nearly rationally related) with a phase offset so that the D hits around the phase circle stay out of the detection region. In this case we go to the P state since the phase precession is guaranteed to be slow enough that we will detect it before an error occurs.

Once frequency is acquired, we enter the phase acquisition (PA) state and wait for another phase detection. Once we have both a frequency estimate, f, and a phase estimate, p, we enter the tracking state (T). If a timeout occurs before a phase detection, we enter the P state.

In the tracking state (T) we update our phase estimate each cycle and use this phase estimate to avoid sampling a register in its keep-out region. If the uncertainty in our phase estimate increases by more than a threshold $\mathsf{pu} - \mathsf{pl} > k$, we enter the P state.

If the phase is drifting very slowly (mesochronous or plesiochronous modulo a rational fraction) we can safely synchronize without prediction. In this case (the P state) we directly use the even signal from the tclk domain to select between the E and O registers. This is safe because the phase is changing slowly enough, and the guard band, $g = d - x$, is large enough, so that a phase detection will occur, returning us to the T state, before this signal becomes unsafe. In the rational case as the phase drifts into the detection region, we will get a 1-of-D detection pattern (where D is the rational denominator) - going to the T state on the first detection handles this correctly.

Operation in the T state depends on the frequency being constant or nearly constant. This is the whole point of a periodic synchronizer. In some systems, however, frequency may change during short periods of time - e.g., when changing between power states. The FIFO synchronizer of Section III can be adapted to work in this mode by Gray-coding the pointers and operating a pair of brute-force synchronizer in parallel with the E/O synchronizers. When frequencies are changing, the FIFO switches to using the brute-force synchronizers. Once the frequencies stabilize, it changes back to using the fast periodic synchronizers.

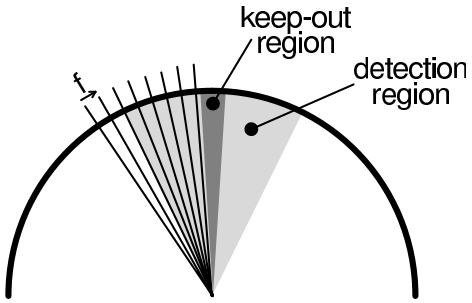

Fig. 10. Safe Plesiochronous synchronization

VI. ANALYSIS

When we have an accurate phase estimate, the system clearly guarantees a safe synchronization. At the sample time, the transmit phase is known to be $\phi \in [\mathsf{pl}, \mathsf{pu}]$, and if $\mathsf{pl} \in [x, 1 + x)$ the even register is safe to sample. If the system parameters, b, and d are chosen properly, the system also guarantees safe synchronization in plesiochronous mode, when we do not have an accurate phase estimate. In this case, we show that the frequency will be in a range where a phase detection will occur sufficiently far in advance of a keep-out event for the synchronizer to guarantee proper sampling.

To show that the synchronizer operates properly, we need to show that either (a) detections will occur often enough that we never enter plesiochronous mode (i.e, that we always have an accurate phase estimate), or (b) when in plesiochronous mode, a detection ($\phi \in [-d, d]$) occurs at least A cycles before a keep-out event ($\phi \in [-x, x]$).

In this section we assume that $f_t > 0.5 f_r$. We discuss how to extend the argument to an arbitrary frequency range in Section VII-E. Without loss of generality, we consider $f \in [0, 0.5]$ (f is the fractional relative frequency, $f = f_t/f_r \bmod 1$). The cases where $f \in [-0.5, 0)$ are identical but with the phase rotating counterclockwise.

Consider the following cases for f:

For $f < f_g$ we will have a detection before an error. In this case the phase moves slowly enough into the detection region that detection will take place at least A cycles before the phase enters the keep-out window, giving us time to synchronize the detection, update the phase estimate, and avoid sampling the unsafe register. This case is illustrated in Fig.10 which shows the phase ϕ (radial lines) over eight clock cycles for a small value of f. Because $f < f_g$, the phase is in the detection region for more than A cycles (six in the figure) before entering the keep-out region.

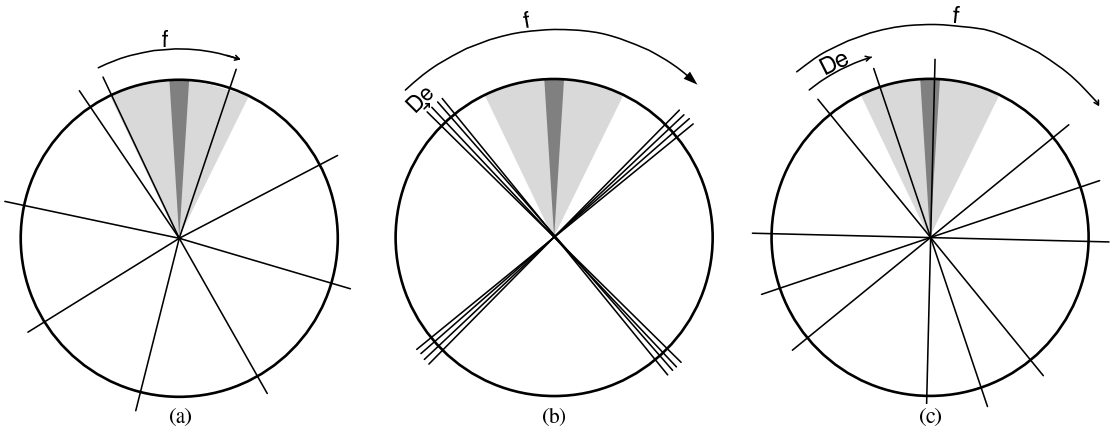

Fig. 11. Cases for f: (a) maximum redetection interval, (b) nearly rational with small De, and (c) nearly rational with large De.

For $f_g \leq f < 2d$ we will have a detection every $N = 1/f < A/g$ cycles, at least once each time the phase rotates around the unit circle. As long as $2^{-b} < gk/A$, we will not enter plesiochronous mode because the phase bounds will diverge by only $2^{-b}A/g$ between detections. For example, for the numbers of our example we have $gk/A = (0.1)(0.5)/(5) = .01$, and b = 7 bits is sufficient precision. This case is illustrated in Fig.11a. Because $f < 2d$, the phase cannot "jump over" the detection region and we are guaranteed to get at least one detection each time the phase rotates about the unit circle. Because $f > f_g$, this rotation will take at most $1/f_g = A/g$ cycles. Fig.11a shows a detection at least every nine cycles.

For $f > 2d$ we represent f as a rational fraction with a bounded denominator plus an error term, $f = N/D \pm e$ where $D \leq C = \lceil 1/2d \rceil$. As we shall show below, the properties of sequences of fractions with bounded denominators, called Farey Sequences, guarantees that $eDC < 1$. In this case we have a repeating pattern of D points around the phase circle that shifts by De each D-cycle period. This gives us the same two cases as for $f < 2d$.

If $De < f_g$, the phase shift each period is small enough that we will have detection before error, the same as when $f < f_g$. This case is illustrated in Fig.11b for $D = 4$. In fact, the constraint here is a bit easier because D cycles elapse each time one of the phase "groups" advances De, hence we will get detection A cycles before a keep-out event as long as $De < \frac{g}{\lceil A/D \rceil}$ which is a looser constraint.

If $f_g \leq De < 2d$ (Fig.11c) then we will get a detection every $\frac{1}{De} < A/g$ cycles so if $2^{-b} < gk/A$ we will detect before we accumulate too much error. The requirement on b here is exactly the same as in the $f_g \leq f < 2d$ case above.

We need to show that for $f > 2d > 1/C$ we can always represent f as $f = N/D \pm e$ with $D \leq C$ and $eDC < 1$. Consider the Farey Sequence $F(C)$, the sequence of rational numbers between 0 and 1 with denominators $D \leq C$. For two adjacent elements of this sequence, $\frac{p}{q}, \frac{r}{s}$, it will always be the case that $\frac{r}{s} = \frac{ps+1}{qs}$ where $q, s \leq C$ and $ps + 1 = qr$ [8]. Thus, the distance between the two adjacent sequence elements $\frac{p}{q}$ and $\frac{r}{s}$ is $\frac{1}{qs}$. We allocate $f \in [\frac{p}{q}, \frac{p}{q} + \frac{1}{q(s+q)}]$ to $\frac{p}{q}$ and $f \in [\frac{r}{s} - \frac{1}{s(s+q)}, \frac{r}{s}]$ to $\frac{r}{s}$. Then we have $e \leq \frac{1}{q(s+q)}$, so $eDC \leq \frac{qC}{q(s+q)} = \frac{C}{s+q} < 1$, because $s + q > C$ due to the properties of Farey Sequences [8].

VII. Discussion

A. Rational Frequency

We often need to synchronize signals between two clock domains that have frequencies that differ by a rational number, i.e., $f_T = \frac{N}{D} f_R$ for integer N and D. This happens, for example, when both frequencies are generated from a common crystal reference frequency by PLLs that divide down the reference frequency and then multiply up. With rational frequencies, we assume that the system provides us with N and D. The phase between the two clocks is assumed to be unknown and may be slowly varying.

To handle rational frequencies we modify our synchronizer design to eliminate the frequency estimator and to keep the phase estimate scaled by D. We keep all phase values multiplied by D to simplify arithmetic. With this representation we deal only with the numerators of rational fractions. That is, we represent the phase as a fixed-point number $i.f$ where i is the integer part and f is the fractional part so that $p = \frac{i.f}{D}$. The phase estimator increments this scaled representation $\bmod D$. The detection value, d, and keep-out value, x, are similarly scaled by D. Because we know the frequency exactly, we keep only the lower bound phase value, pl, and increment pl by N each cycle. The phase estimate never diverges, so once the synchronizer enters the tracking state, it remains in that state indefinitely.

With rational frequency, the phase visits the same D points on the phase circle each D cycles (the actual number of unique points is $\frac{D}{\gcf(N,D)}$). If there are no detections in $D+1$ cycles, the synchronizer enters plesiochronous mode. If there is a detection, it will repeat every D cycles and will be accurately predicted by the phase estimate each time.

Sarmenta et al. [6] have previously described a synchronizer for rationally related clocks. Compared to that work, our approach has two advantages. First, the Sarmenta approach assumes that the phase difference between the two clocks is known and static. This is not the case for most real systems where there may be a significant unknown and slowly varying skew between the two clock domains. Our approach uses a phase detection (or lack thereof) to dynamically detect the phase difference. Second, the Sarmenta approach requires a table of size D to store the conflict pattern. Our approach dispenses with the table by detecting conflicts from the estimated phase. This saves considerable area, especially for large D.

If the phase between the two clocks changes slowly (by $< \Delta p$ per cycle - i.e., at least temporarily the actual frequency is $f_T = \frac{N}{D} f_R + \Delta p$), then the system described above works as long as $\Delta p < \min(\frac{g}{D}, \frac{g}{S})$. This constraint ensures that the first step into the detection region (of at most $D\Delta p$) will not penetrate into the keep-out region and also that once detected, the detection will be synchronized before the phase enters the keep-out region.

B. Plesiochronous and Mesochronous Frequency

The synchronizer described above works properly for the special cases of plesiochronous and mesochronous clocks. For plesiochronous clocks, the behavior is similar to that of the synchronizer of Dennison et al. [4]. Before the transmit clock phase drifts into the unsafe region, a detection occurs. This gives an accurate phase estimate which enables the synchronizer to avoid sampling the unsafe value. After the transmit phase drifts out of the detection region, the system uses the phase estimate to make sampling decisions until the estimates diverge by threshold, k. Whether this happens before the next detection depends on the values of b and f. If divergence does occur before detection, the system enters plesiochronous mode until the next detection.

Compared to Dennison's system [4], our system has the advantage that signals are only sampled by the rising edge of the transmit clock (into the E and O registers) and the rising edge of the receive clock (into the output register). In contrast, Dennison's system requires sampling on a negative edge of the transmit clock to generate a half-cycle delay (other synchronizers also employ negative edge sampling or the use of delayed clocks [1]) which introduces timing complications.

For mesochronous clocks, the transmit clock phase will either be in the detection region - in which case our system stays in the locked state - or outside the detection region - in which case our system stays in the plesichronous state. Either way, correct sampling is assured. The system is also tolerant of a drifting mesochronous phase which moves between the two states.

C. Jitter

The effect of jitter on the periodic synchronizer described above depends on the frequency of the jitter. Low-frequency jitter, $f_j < f_g f_r$, is completely rejected by the system and has no effect. This jitter is slow enough not to affect plesiochronous mode and is tracked when the system is in the tracking state.

Medium-frequency jitter, $f_g f_r \leq f_j < \frac{f_r}{A}$, affects tracking mode but not plesiochronous mode. Jitter in this portion of the spectrum, with magnitude t_{jm} (in ps), affects the position of clock edges between detections and can be accommodated by adding margin to upper and lower bounds on each phase detection (i.e., initializing pl to $-d - \frac{t_{jm}}{t_{tcy}}$).

High-frequency jitter, $f_j > \frac{f_r}{A}$ affects both tracking and plesiochronous mode and can be accounted for by including the peak-to-peak magnitude of the jitter in the width of the keep out region, $t_x = t_s + t_h + 2t_{jh}$.

D. Synchronizer Parameters

The two free synchronizer parameters are d, the half-width of the phase detection region, and k, the threshold for entering plesiochronous mode. Given a keep-out region, $2x$, which is a property of the synchronizer flip-flops and high-frequency jitter, choosing d gives the value of the guard band, $g = d - x$, which in turn determines the number of bits required for the frequency and phase estimates, $2^{-b} < \frac{gk}{A}$, so $b > lg\left(\frac{A}{gk}\right)$. Choosing a small d, gives a more precise phase estimate, and hence reduces synchronizer delay, but at the expense of requiring more bits in the frequency and phase estimators to ensure correct behavior in the presence of a narrow guard band.

Choosing the value of k, gives a similar tradeoff. Choosing a small k gives a lower average synchronizer delay - because the synchronizer will enter plesiochronous mode (with no delay) sooner. However choosing a small k also requires more bits of precision in the estimates. In practice a value of $k = 0.5$ or 0.25 works well and simplifies the check for $\mathsf{pu} - \mathsf{pl} > k$.

E. Frequency Range

The even/odd synchronizer works correctly over a large range of relative tclk and rclk frequencies. The synchronizer will work for arbitrarily fast f_t - until t_{tcy} is so small that there is not room for safe sampling with reasonable margins. However, additional precision may be needed for very slow f_t where we need $\hat{b} = \lceil \lg\left(\frac{f_r}{f_t}\right) \rceil$ additional bits of precision to make sure our phase estimates do not diverge between tclk edges.

To analyze low tclk frequencies, represent t_{tcy} as $t_{tcy} = (P+q)t_{rcy}$ for integer P and $|q| < 1$. Then $f = \frac{t_{rcy}}{t_{tcy}} = \frac{1}{P+q} = \frac{1}{P} + r$ where $r = \frac{-q}{P(P+q)} < \frac{1}{P^2}$. In this case, every P rclk cycles we get a single tclk edge with phase advanced by at most $1/P$ - up to one whole rclk cycle - from the last tclk edge. The situation is identical to a system with $t_{tcy} = (1+q)t_{rcy}$ except that it proceeds P times slower. Hence we need $\hat{b} = lg(P)$ additional bits of precision to prevent our phase estimates from diverging during the P cycles between tclk edges. In most practical cases, f_t and f_r are within an order of magnitude of one another and the additional precision required is at most 4 bits.

F. Updating Phase Estimates on non-Detection

The phase estimator of Section IV resets the phase estimate when detection occurs and simply integrates to advance these estimates in the absence of a detection. A slightly more accurate phase estimate can be realized by updating the phase estimate to exclude the detection region when no detection occurs. That is, in the absence of detection, we integrate the phase estimate interval $[\mathsf{pl}, \mathsf{pu}]$ and then intersect this interval with $[d, 1-d]$.

Consider, for example a case where $f = 1.001$ - i.e., the transmit clock is just slightly faster than the receive clock - taking 1,000 cycles to advance around the phase circle. On the first non-detection cycle after a string of detections, the phase interval, ignoring the advance, will be approximately $[0.001 - d, 0.001 + d]$. Most of this interval is in the detection region. However, we know we are outside the detection region (because there was no detection), so we can tighten our bounds considerably to $[d, 0.001 + d]$.

This refinement is particularly advantageous with closely spaced rational clocks where once tightened, this phase estimate does not diverge until the next detection.

G. More than Two Phases

When t_{cy} is very small or t_{jh} is very large, the keep out region may be so large that it is not possible to build a working synchronizer with just two phases (even and odd). These cases can be handled by using a larger number of phases. For example, one can build a modulo-3 synchronizer where the E and O registers are replaced by three registers (R1, R2, and R3), and the phase estimate is kept modulo-3. Like the E/O synchronizer, the transmitter writes the three registers in sequence, and the receiver uses the phase estimate to sample the most recently written *safe* register. This organization can be extended to an arbitrary number of phases to handle arbitrarily small t_{cy} or large t_{jh}.

H. Simulation Results

We constructed a Verilog RTL model of the periodic synchronizer described in Sections II and IV, and used two such synchronizers to build a flow-controlled FIFO as described in Section III. The delay lines in the phase detectors were modeled behaviorally, and all flip-flops were instrumented with setup- and hold-time checks. Verilog simulations were performed with one clock fixed at 1GHz and the other clock set to 2000 randomly chosen frequencies between 500MHz and 2GHz. The phase of the 1GHz clock was swept slowly back and forth over a range of 1600ps, changing at a rate of 1ps every 10 cycles, to ensure that all relative clock phases were tested. No timing errors were detected in any simulation.

I. Implementation

We synthesized the logic for each of the synchronizer's components using a TMSC 40nm standard cell library and targeting a 900 MHz clock frequency. The results are shown in Table III for several values of estimator precision, b, and data path width, w. The data path is smaller than the area of a comparable width brute-force synchronizer. The frequency and phase estimator area is comparable to the area of a 64-bit datapath.

TABLE III
SYNTHESIZED GATE AREA OF COMPONENTS

Component	Parameter	Area (μm^2)
Phase Detector		50
Frequency Estimator	b = 8	298
Frequency Estimator	b = 10	358
Frequency Estimator	b = 12	415
Phase Predictor	b = 8	468
Phase Predictor	b = 10	589
Phase Predictor	b = 12	736
Data Path	w = 32	666
Data Path	w = 64	1331
Data Path	w = 256	5328

TABLE IV
ESTIMATED FORWARD SYNCHRONIZER AREA

b	w	M	Area Calculation	Area (μm^2)
10	32	1	50+358+589+666	1663
10	64	1	50+358+589+1331	2328
10	256	1	50+358+589+5328	6325
8	64	5	5*(50+468+1331)+298	9543
10	64	5	5*(50+589+1331)+358	10208
12	64	5	5*(50+736+1331)+415	11000

Using the sizes from Table III, we estimate the total area of the complete forward synchronizers shown in Table IV for different combinations of b, w, and M multiple data paths. When synchronizing data from M different blocks running at the same tclk frequency into a common rclk, the various source clocks may arrive with different phase delays even though their clocks are derived from a common source. In this case, the frequency estimator can be shared, but independent phase detectors and phase estimators are required.

Reduced synchronizer latency allows shallower FIFOs to be used with no loss of performance. The FIFO needs to be at least deep enough to cover the round trip latency of pointer synchronization to operate at full throughput. Table V compares the area of several alternative w=64-bit wide FIFOs using flip-flop memories and both brute-force (BF, 4 stage) and fast-periodic (FP) synchronizers. Table V shows that adding the fast periodic synchronizer (with $b = 10$) to a FIFO of depth 11 increases the area by 30%. However, the reduced latency of FP allows the depth of the FIFO to be reduced to 4 with no loss of performance and an 33% reduction in area. With wider FIFOs memory area dominates, so the FP synchronizer's lower latency can result in even more area reduction.

Since the phase detector and data path elements involve two unrelated clocks, the timing requirements between those clocks can be difficult to capture in a synthesis tool's constraint language. Instead of constraining the synthesis, placement, and routing of these critical elements, for actual silicon im-

TABLE V
ESTIMATED AREA OF 64-BIT FIFO USING BF AND FP SYNCHRONIZERS

Sync	Depth	Area (μm^2)	Area(%)
BF	11	6741	100%
FP	11	8742	130%
FP	4	4565	67%

plementation we plan to assemble and analyze semi-custom assemblies of standard cells for the phase detector and data path, while using traditional synthesis for the phase predictor and surrounding logic.

When clock skew optimization is performed as part of a synthesis and timing flow, the clocks to any two flip-flops in the same clock domain may be skewed by the optimization to aid overall timing. If the clock to our phase detector is skewed significantly from the clock to the corresponding synchronizer data path, phase prediction will produce a select signal that is correct for the clock arriving at the phase detector but results in mis-sampling by the data path. Clock distribution to the phase detector and data path should be designed for equal arrival times at both blocks. This can be achieved by constraining the timing optimization or placing the phase detector and data path into a single semi-custom block.

VIII. CONCLUSION

We have described the even/odd synchronizer, an all-digital periodic synchronizer with very low latency ($0.5+x$ cycles on average, where x is the keep-out region of the flip-flop) and an arbitrarily small probability of synchronization failure. This performance is achieved by moving synchronization off the critical path of the synchronizer - into the frequency estimator and phase detector - where an arbitrarily long time period can be used to allow metastable states to decay. As with Stewart and Ward [3] we exploit the periodic nature of the clocks to predict clock phase several cycles in the future to hide the synchronization latency. Unlike previous work, however, we perform this prediction digitally - eliminating the problematic analog delay lines of previous predictive synchronizers. Unlike other previous synchronizers (e.g., [4]) we create safe signals to sample using only the positive edge of the transmit clock - avoiding timing issues associated with clocking signals on negative edges or using delayed versions of clocks.

The even/odd synchronizer operates by having the transmitter write to an even register on even clock cycles and an odd register on odd clock cycles. The receiver digitally estimates the frequency and the phase of the transmit clock and uses the phase estimate to select the most recently written transmit register which is safe to sample. A synchronizer with flow control can be realized by using a pair of even/odd synchronizers to pass the head and tail pointers between the input and output clock domains of a FIFO. The resulting FIFO synchronizer has very low latency ($1.5+x$ cycles on average) and can use binary-coded (rather than Gray-coded) head and tail pointers. Reducing FIFO synchronizer latency reduces the area of the dual-port memory required for the FIFO which must be sized large enough to handle the worst-case round-trip control latency.

We present a number-theoretic argument for the correctness of our synchronizer. Based on the properties of Farey Sequences [8] we show that if a sufficient guard band is provided and the frequency and phase estimates are kept to sufficient accuracy, then the synchronizer will never sample an unsafe signal. For one set of frequencies we always have an accurate phase estimate because detections occur with sufficient frequency. For all other frequencies, we show that our phase is changing slowly enough that a detection in the guard band precedes a keep-out event by more than the delay of the synchronizer.

The even/odd synchronizer can be adapted to handle a rational relative frequency ($f = \frac{N}{D}$) by keeping all phase quantities scaled by D. In the rational case the frequency is known exactly so phase estimates, once generated, never diverge. Compared to the rational synchronizer of Sarmenta et al. [6], the even/odd synchronizer does not require a table and can handle an arbitrary, unknown, and slowly varying phase difference between the two clocks.

REFERENCES

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

[2] Stewart, William K., *A Solution to a Special Case of the Synchronization Problem*, MIT BS Thesis, June 1983.

[3] Stewart, W.K., and Ward, S.A., "A Solution to a Special Case of the Synchronization Problem", *IEEE Transactions on Computers,* 31(1):123-125, January 1988.

[4] Dennison, L., Dally, W.J., and Xanthopoulos, D., "Low-Latency Plesiochronous Data Retiming," *Proceedings of the 16th Conference on Advanced Research in VLSI (ARVLSI'95)*, Chapel Hill, NC, pp. 304-315, 1995.

[5] Frank, U., Kapshitz, T., and Ginosar, R., "A Predictive Synchronizer for Periodic Clock Domains," *J. Formal Methods in System Design*, 28(2):171-186, 2006.

[6] Sarmenta, L.F.G., Pratt, G.A., Ward, S.A., "Rational clocking," *Proceedings of the IEEE International Conference on Computer Design, ICCD-95*, pp.271-278, 1995.

[7] Chakraborty, A., and Greenstreet, M. R., "Efficient Self-Timed Interfaces for Crossing Clock Domains," *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems (ASYNC-03)*, pp. 78-88, 2003.

[8] Hardy, G.H., and Wright, E.M., *An Introduction to The Theory of Numbers*, Chapter 3, Oxford Science Publications, Fifth Edition, 1979.