Analysis and Optimization of Pausible Clocking based GALS Design

Xin Fan, Miloš Krstić, and Eckhard Grass
IHP Microelectronics, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany
{fan, krstic, grass}@ihp-microelectronics.com

Abstract — **Pausible clocking based globally-asynchronous locally-synchronous (GALS) system design has been proven a promising approach to SoCs and NoCs. In this paper, we analyze the throughput reduction and synchronization failures introduced by the widely used pausible clocking scheme, and propose an optimized scheme for higher throughput and more reliable GALS design. The local clock generator is improved to minimize the acknowledge latency, and a novel input port is applied to maximize the safe timing region for the clock tree insertion. Simulation results using the IHP 0.13-µm standard CMOS process demonstrate that up to one-third increase in data throughput and an almost doubled safe timing region for clock tree distribution can be achieved in comparison to the traditional pausible clocking scheme.**

I. INTRODUCTION

With the growing complexity of systems-on-chips (SoCs), traditional synchronous digital circuits become increasingly difficult to implement. A major challenge is the distribution of a low skew global clock. The large number of required buffer cells can lead to 40% of the total power dissipation and occupy significant silicon area.

By eliminating the global clock, globally-asynchronous locally-synchronous (GALS) design provides a promising solution to SoCs. The most straightforward way to GALS systems is to insert synchronizer circuits between different clock domains [1]. Normally a synchronizer consists of two or more cascaded flops, and it introduces latency in data transfer. Another approach to GALS systems is the use of asynchronous FIFOs [2, 3], and this results in overheads in both area and power. In recent years, an alternative method to GALS design, which is mainly based on pausible local clocks, has been developed [4, 5, 6, 7, 8, 9, 10, 11, 12]. Communication between asynchronous modules is achieved using a pair of request-acknowledge handshaking signals, and the local clocks are paused and stretched, if necessary, to avoid metastability in data transfer.

Until now, most of the silicon validated pausible clocking systems [13, 14, 15, 16] are designed based on the scheme proposed and improved in [5, 6, 8]. As a latest example, this scheme is applied in [17] to implement a dynamic voltage frequency scaling (DVFS) NoC. Fig. 1 depicts a point to point GALS system based on this well-known scheme and its waveforms of the handshake signals Each synchronous module is surrounded by an asynchronous wrapper, which mainly consists of a local clock generator and several asynchronous I/O ports. A four-phase bundled-data protocol is used and a single latch is deployed in the input port to load the data from the output port. In [13, 14] the design of asynchronous wrappers was discussed in detail.

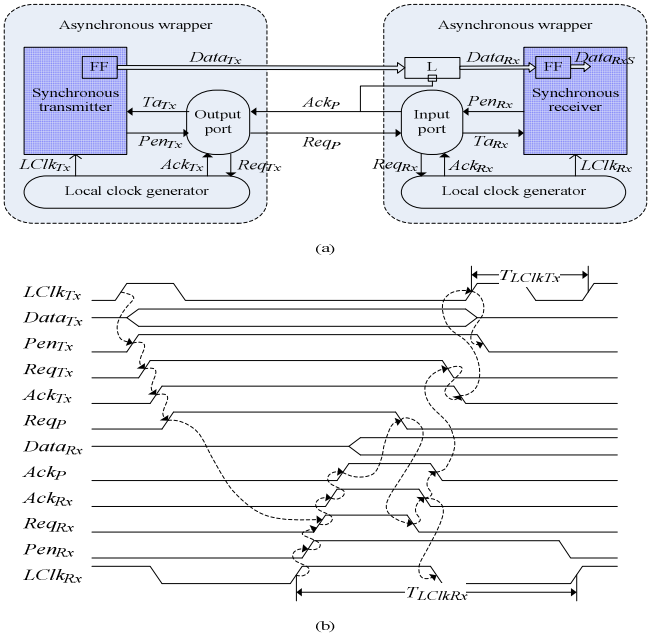


Fig. 1 A GALS system (a) and its handshake signals (b)

In this paper, we focus on the analysis and optimization of this widely used pausible clocking scheme. Section 2 studies the acknowledge latency of local clock generators, and demonstrates its impacts on the system throughput and data synchronization. Section 3 shows an improved scheme with minimum latency from the clock generator and maximum safe timing region for the clock tree insertion. The proposed scheme is implemented and evaluated using the IHP 0.13-µm CMOS process in Section 4. Finally, a brief conclusion is given in Section 5.

II. ANALYSIS OF PAUSIBLE CLOCKING SCHEME

A typical local clock generator used in pausible clocking schemes is depicted in Fig. 2 [8, 9, 17, 18]. A programmable delay line is employed to generate the clock signal *LClk*. An array of MUTEX elements is used to arbitrate between port requests *Reqx* and the request clock signal *RClk*. If any *Reqx* gets acknowledged, *LClk* will be paused.

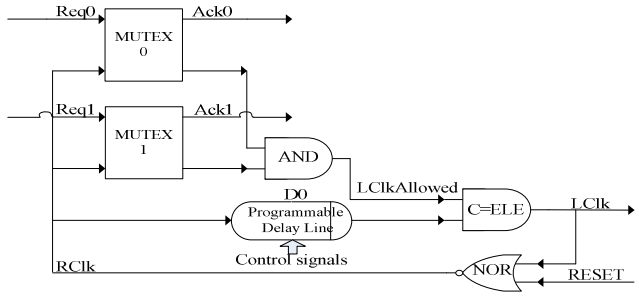


Fig. 2 A typical local clock generator (with 2 ports)

A. Acknowledge Latency of Clock Generator

For a MUTEX element, at any time only one of two incoming events *Reqx*+ and *RClk*+ is allowed to pass on a first come first serve basis. A *Reqx*+ arriving before *RClk*+ will be acknowledged immediately by the MUTEX, but a *Reqx*+ arriving after *RClk*+ will not be acknowledged until *RClk*- happens. If *Reqx*+ and *RClk*+ arrive simultaneously, the MUTEX element will decide randomly which signal should be acknowledged.

For clarity in the following discussion, we define the request acknowledge window (*RAW*) in a local clock generator as the duration in each cycle of *LCLK* when the port requests can be acknowledged. For the clock generator shown in Fig. 2, its *RAW* is the inactive phase of *RClk*, which corresponds to the active phase of *LClk* as shown in Fig. 3. Considering 50% duty cycle of *LClk*, the duration of the *RAW* in this clock generator can be deduced as follows:

$$t_{RAW} = t_{RCLK=0} \approx t_{LCLK=1} \approx T_{LCLK} / 2 \,. \tag{1}$$

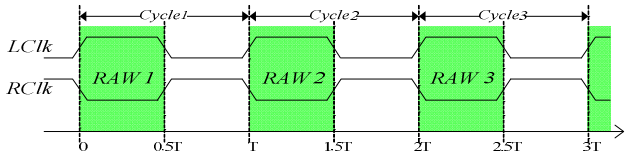


Fig. 3 Request acknowledged window

Any *Reqx*+ occurring outside of the *RAW* will lead to increased acknowledge latency. The worst situation happens if *Reqx*+ arrives concurrently with *RClk*+ and *RClk* is acknowledged by the MUTEX. Then *Reqx* can't be granted until *RClk* goes low. Therefore, based on the *RAW*, we can further derive the maximum acknowledge latency caused by the above local clock generator in equation (2). In the following, we will analyze the impacts of the acknowledge latency on system throughput and data synchronization.

$$\max(Latency_{Ack}) = t_{RCLK=1} = T_{LCLK} - t_{RAW} \approx T_{LCLK} / 2 \,. \tag{2}$$

B. Throughput Reduction

For simplicity, a point to point GALS communication, as shown in Fig. 1, is taken as an example. We firstly discuss the data transfer from a demand-type output (D-OUT) port to a poll-type input (P-IN) port, and similar analysis is then applied to the other three communication channels used in point to point GALS systems [8, 13].

1) D-OUT Port to P-IN Port Channel

For the receiver equipped with a P-IN port, the local clock $LClk_{Rx}$ will be paused after Req_{Rx}+ occurs [8]. Req_{Rx} will be asserted after a Req_P+ is detected, which is generated by the output port in the transmitter running at an independent clock. Therefore, without loss of generality, the arrival time of Req_P+, and then the arrival time of Req_{Rx}+, can be modelled as a uniformly distributed random variable within a period of $LClk_{Rx}$. Since $t_{RAW} = T_{LCLKRx}/2$ in the above clock generator, there is 50% probability that a Req_{Rx}+ is extended to be acknowledged in the next *RAW*. Moreover, since the data is sampled in the receiver at the next rising edge of $LClk_{Rx}$, $Data_{RxS}$ will be delayed for one cycle of $LClk_{Rx}$. As an example, the latencies of acknowledge signal Ack_{Rx} and sampled data $Data_{RxS}$ are depicted in Fig. 4.

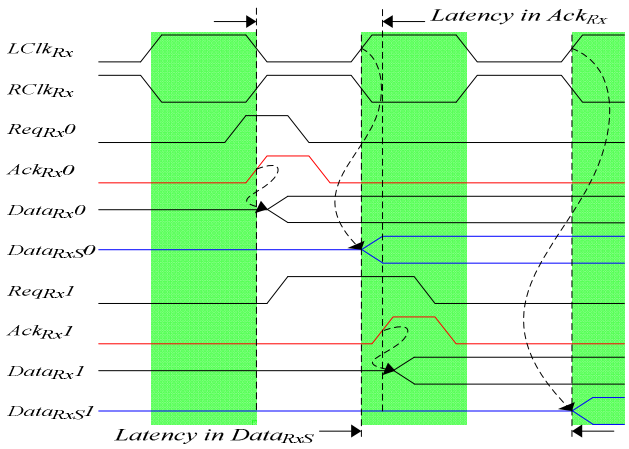


Fig. 4 Latencies in Ack_{Rx} and sampled data $Data_{RxS}$

For the transmitter equipped with a D-OUT port, its local clock $LClk_{Tx}$ is paused before Req_P is asserted by the output port, and $LClk_{Tx}$ will not be released until Req_P gets acknowledged [8]. Since Req_P will not be acknowledged by the input port until Req_{Rx}+ is acknowledged by the clock generator on the receiver side, there is maximum a $T_{LClkRx}/2$ latency in acknowledging Req_P as well. Consequently, the latency in the receiver is propagated into the transmitter. If the period of $LClk_{Rx}$ is much longer than that of $LClk_{Tx}$, this latency will result in a multi-cycle suspension in $LClk_{Tx}$. Because the data is processed synchronously to $LClk_{Tx}$ in the transmitter, the suspension in $LClk_{Tx}$ will eventually result in a delay in data transfer.

For instance, considering a D-OUT port to P-IN port channel, where the periods of $LClk_{Tx}$ and $LClk_{Rx}$ satisfy the following condition:

$$(N-1)\cdot T_{LCLKTx} = \frac{3}{2}\cdot T_{LCLKRx}, \tag{3}$$

and the requests in the transmitter, Req_{Tx}, is asserted every N cycles of $LClk_{Tx}$, as shown in equation (4):

$$T_{ReqTx} = N \cdot T_{LCLKTx} \,. \tag{4}$$

After Req_{Tx} is asserted in the transmitter, a Req_P+ will be generated by the output port controller. If a Req_P+ arrives in the receiver in the on-phase of $RClk_{Rx}$, it will not be acknowledged until $RClk_{Rx}$ turns low. During that period $LClk_{Tx}$ is paused and stretched in its off-phase. After $RClk_{Rx}$- happens, which corresponds to $LClk_{Rx}$+ occurring, Req_P will be acknowledged and then $LClk_{Tx}$ will be released. As a result, the next rising edges of both $LClk_{Tx}$ and $LClk_{Rx}$ are automatically synchronized to occur at almost the same time. After $LClk_{Rx}$ and $LClk_{Tx}$ are synchronized, Req_P will be asserted in *(N-1)* cycles of $LClk_{Tx}$ in the transmitter. According to condition (3), $RClk_{Rx}$ will also be asserted after one and a half cycles of $LClk_{Rx}$ in the receiver. Therefore, the next Req_P+ and $RClk_{Tx}$+ will occur simultaneously again, and the extension in Req_P+ and the suspension in $LClk_{Tx}$ will repeat as well.

Above analysis illustrates that, for systems satisfying (3) and (4), once a Req_P+ occasionally arrives in the active phase of $RClk_{Rx}$, all following Req_P+ will be synchronized to occur at the same time with $RClk_{Rx}$+. Consequently, each Req_P will be extended for $max(Latency_{Ack}) = T_{LClkRx}/2$ to be

acknowledged, and $LClk_{Tx}$ will be periodically suspended for $T_{LClkRx}/2$ as well. This suspension in $LClk_{Tx}$ will cause a delay for data transfer. As an example, Fig. 5 illustrates a waveform fragment of data transfer under this circumstance, where $N = 7$.

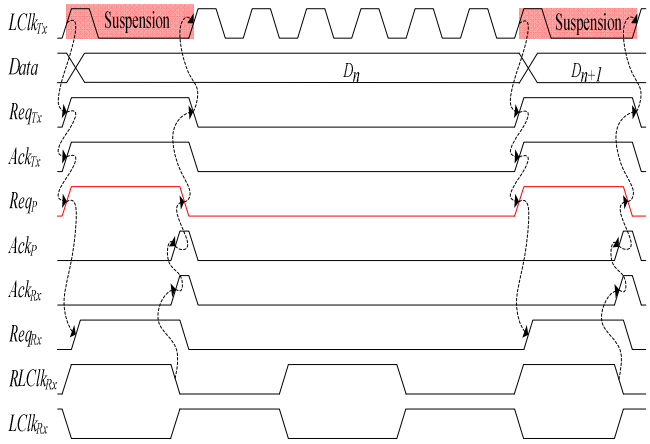


Fig. 5 Extension in Req_P and suspension in $LClk_{Tx}$

Every time $LClk_{Tx}$ is suspended, its inactive phase will be stretched for a period of $(T_{LClkRx}/2 - T_{LClkTx})$. Based on conditions (3) and (4), the throughput reduction R_{Tx} caused by the suspension of $LClk_{Tx}$ is deduced in equation (5). We see that the exact percentage is determined by the value of N. With the increase in the value of N, the limit of R_{Tx} reaches 1/3, as shown in equation (6). It means that up to one-third reduction in data throughput could be introduced by the acknowledge latency.

$$R_{Tx} = (\frac{1}{2} \cdot T_{LCLKRx} - T_{LCLKTx}) / T_{ReqTx} = \frac{N-4}{3 \cdot N} . \quad (5)$$

$$\lim_{N->+\infty} R_{Tx} = \lim_{N->+\infty} \frac{N-4}{3 \cdot N} = \frac{1}{3} . \quad (6)$$

2) Other Point to Point Channels

A similar analysis can be easily applied on the other three point to point communication channels. In Tab. 1 we present the impacts of acknowledge latency on handshake signals and local clocks for the four channels. It can be seen that the only exception occurs when both input port and output port are of demand type. No matter whether there is data ready to be transferred or not, the clocks on both the transmitter side and the receiver side will be paused as soon as the ports get enabled. Although there is no extension in the handshake signals or suspension in the clock signals caused by the acknowledge latency of local clock generators, this channel is prone to unnecessary long suspensions in both $LClk_{Rx}$ and $LClk_{Tx}$, and a huge drop in data throughput could happen. Careful design is required for applying this type of channel to reduce system power consumption.

Tab. 1 Impacts of acknowledge latency

Channel Type	Extended Signals	Suspended Clock	Maximum Suspension
D-OUT to P-IN	Req_P+	$LClk_{Tx}$	$T_{LClkRx}/2$
P-OUT to D-IN	Ack_P+	$LClk_{Rx}$	$T_{LClkTx}/2$
P-OUT to P-IN	Req_P+, Ack_P+	$LClk_{Rx}$	$T_{LClkTx}/2$
D-OUT to D-IN	*NO*	*NO*	*0*

C. Synchronization Failure

A significant benefit from GALS design is to simplify the global clock distribution by a set of independent local clock networks. For pausible clocking schemes, however, a crucial issue is the synchronization failure caused by the local clock tree insertion delay in receivers. Fig. 6 depicts a failure case occurring in the traditional scheme (Fig. 1). As the clock tree insertion delay is irrelevant to the handshake signals' propagation delay, $LClk_{RxDly}$+ can arrive at the sampling flip-flop *FF* at simultaneously with loading data into the input port latch *L*. Then metastability occurs in *FF*.

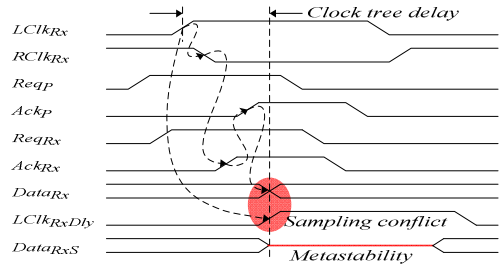


Fig. 6 A synchronization failure case

1) $\Delta_{LClkRx} < T_{LClkRx}$

Data synchronization issues in pausible clocking schemes were first discussed in [7]. The author suggested integrating a clock buffer network into the local ring oscillator, and proposed a pipelined interface to hide the control overhead. This method is only suitable for pipelined systems. Recent work in [12, 19] reveals that, for clock delays satisfying $\Delta_{LClkRx} < T_{LClkRx}$, there are two timing regions in each cycle of $LClk_{Rx}$, as shown in Fig. 7 for example, where negligible synchronization failure probability can be expected [12].

In Fig. 7, *Cycle 1* illustrates the situation that the data is safely sampled by *FF* before *L* turns to be transparent. It contributes the safe timing region *S1* of Δ_{LClkRx} as follows:

$$0 \le \Delta_{LClkRx} < d_{NOR} + d^{0}_{MUTEX} + d_{Port} - t_{hold} , \quad (7)$$

where d^{0}_{MUTEX} and d_{Port} denote the MUTEX delay time from Req_{Rx}+ to Ack_{Rx} + without metastability and the asynchronous port delay from Ack_{Rx}+ to Ack_P +. On the contrary, *Cycle 2* represents another situation that data is safely sampled after it is latched in *L*. We draw the pessimistic case that Req_{Rx}+ happens concurrently with *RClk*+ and metastability occurs in the MUTEX. This leads to the safe region *S2*:

$$T_{LCLKRx}/2 + (d_{NOR} + d^{0}_{MUTEX} + \Delta d_{MUTEX} + d_{Port} + d_{Latch}) + t_{setup} \le \Delta_{LCLK} < T_{LCLKRx} , \quad (8)$$

where Δd_{MUTEX} denotes the additional delay of the MUTEX to resolve metastability, and d_{Latch} is the delay of *L* from asserting gate enable (Ack_P+) to data being stable.

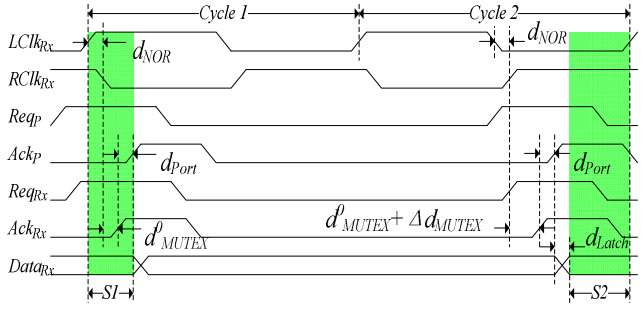


Fig. 7 Safe regions for $\Delta_{LClkRx} < T_{LClkRx}$

Inequality (8) shows that the width of *S2*, W_{S2}, depends on the period of $LClk_{Rx}$. Analyze W_{S2} in two typical cases:

a) If $T_{LCLKRx}/2 \approx d_{NOR} + d^0_{MUTEX} + \Delta d_{MUTEX} + d_{Port} + d_{Latch} + t_{setup}$, *then* $W_{S2} \approx 0$. It means that the MUTEX needs to consume half of a clock cycle to resolve metastability. As a result, only the safe region *S1* is valid, which is dominated in width by a number of gate delays shown in (7).

b) If $T_{LCLKRx}/2 >> d_{NOR} + d^0_{MUTEX} + \Delta d_{MUTEX} + d_{Port} + d_{Latch} + t_{setup}$, *then* $W_{S2} \approx T_{LCLKRx}/2$. With the increase of T_{LCLkRx}, W_{S2} is widened, but the hazard region is also extended. The ratio of W_{S2} to T_{LCLKRx} reaches at most 50%.

Therefore, we see that the width of the safe regions falls inside the range of $W_{S1} < W_{S1+S2} < T_{LCLKRx}/2$. For small T_{LCLkRx}, there is a rather narrow region *S1* to insert clock tree. Even for large T_{LCLkRx}, only half of a clock period is allowed.

2) $\Delta_{LCLkRx} \geq T_{LCLkRx}$

It should be noticed that the safe regions within each cycle of $LClk_{Rx}$, as discussed in the above, is always aligned with $LClk_{Rx}$+. A stretch in $LClk_{Rx}$ will lead to a delay in the safe regions of the next cycle. If the clock tree delay meets $\Delta_{LCLkRx} \geq T_{LCLkRx}$, this delay in safe regions also need to be considered for data synchronization. Take Fig. 8 for instance, where $\Delta_{LCLkRx} \approx 1.8 T_{LCLkRx}$. During *Cycle 1*, the rising edge of the delayed clock $LClk_{RxDly}$ falls in the safe region of $LClk_{Rx}$, and data is sampled correctly by *FF*. Then a stretching on $LClk_{Rx}$ happens, and the safe regions in *Cycle 2* are delayed. But there is another $LClk_{RxDly}$+ scheduled in the clock tree before the stretched clock, which arrives at *FF* without any delay. This eventually leads to a sampling conflict.

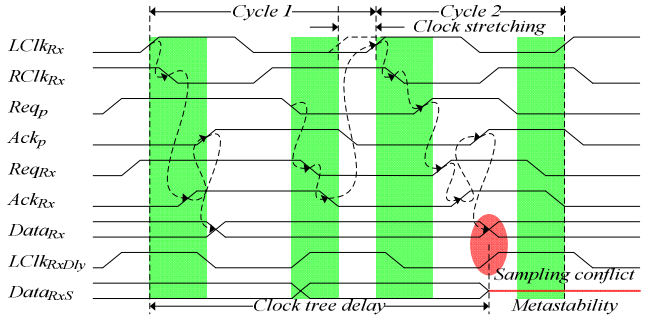


Fig. 8 Synchronization failure when $\Delta_{LCLKRx}=1.8T_{LCLKRx}$

In bellow, the stretching on $LClk_{Rx}$ is analyzed according to the type of input ports used on the receiver side:

a) P-IN port

In this situation, a maximum $T_{LClkTx}/2$ suspension on each cycle of $LClk_{Rx}$ could be introduced by the acknowledge latency as shown in Tab. 1. So the stretching on $LClk_{Rx}$, and then the delay of safe regions, is up to $(T_{LCLKTx}/2\text{-}T_{LCLKRx})$. Since T_{LCLKTx} is independent from T_{LCLKRx}, this delay could be long enough to mismatch the safe regions of successive cycles of $LClk_{Rx}$, as illustrated in Fig. 9. There turns to be no common safe region for the clock tree insertion. Moreover, if $\Delta_{LCLKRx} > 2T_{LCLKRx}$, more than one cycle of $LClk_{Rx}$ could be stretched within the clock tree delay, and an accumulated delay in safe regions should be considered.

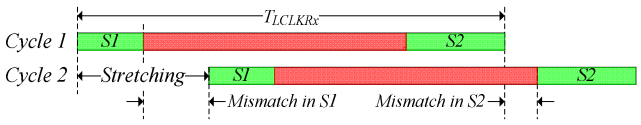


Fig. 9 Safe region mismatch due to the clock stretching

b) D-IN port

In this case, $LClk_{Rx}$ is paused and stretched by the input port controller until Req_P- is triggered by the output port controller. The stretching on $LClk_{Rx}$ as well as the delay in safe regions is unpredictable. That means no common safe region exists for the multi-cycle clock delay.

Now we can conclude that, for the clock tree insertion delay exceeding one clock period, the uncertainty on clock stretching must be taken into account, and no matter what type of input port is utilized, there is no safe region in the traditional scheme. In fact, in most of the reported pausible clocking systems, the local clock trees were deliberately distributed to satisfy $\Delta_{LCLkRx} < T_{LCLkRx}$ [13, 14, 15, 16].

For a multiple cycle clock tree delay, an asynchronous FIFO was suggested in [20] to synchronize the input data with the delayed clock, which leads to increased latency in the datapath and additional overheads in area and power. An interface circuit using partial handshake signals was shown in [11] for high-speed systems with large clock delay, while there is an unknown nonzero probability of failure in the circuits. For the design of GALS systems insensitive to the clock tree delay, a synchronizing scheme based on locally delayed latching (LDL) was presented in [12, 19]. Since the clocks can't be paused in the LDL interface, it introduces additional timing constraints on both the asynchronous input port controller and the combinational logic following the sampling register *FF*, which limits its application. Hence, more stable and efficient synchronizing circuits are required for inserting local clock trees with multi-cycle delay in the pausible clocking based GALS systems.

III. Optimization of Pausible Clocking Scheme

In this section, the pausible clocking scheme is optimized in two respects. The local clock generator is first improved to minimize the acknowledge latency, and then a novel input port, including the data latching mechanism and the port controller, is suggested to maximize the safe region for the clock tree distribution.

A. Optimized Local Clock Generator

Behind the acknowledge latency is the fact that in Fig. 1 the local clocks on both the transmitter side and the receiver side need to be paused for safe data transfer. To avoid the acknowledge latency, we can deploy an asynchronous FIFO as work [9] to decouple local clocks, with overheads in area and power as penalty. Another simple solution, however, is to widen the *RAW* of the clock generator as shown in Fig. 10. There are two delay lines, the programmable delay line *D0* followed by the fixed delay line *D1*, used in the local ring oscillator. The delay lengths of *D0* and *D1* are as below:

$$d_{D0} = T_{LClk}/2 - (d_{D1} + d_{C-ELE} + d_{NOR}), \quad (9)$$

$$d_{D1} = d_{AND0} + d_{AND1} + (d^0_{MUTEX} + \Delta d_{MUTEX}). \quad (10)$$

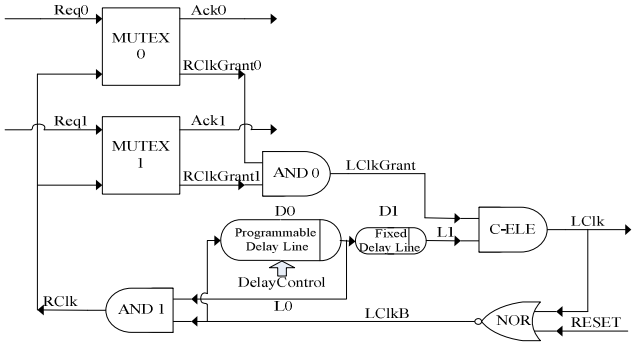


Fig. 10 An optimized local clock generator (with 2 ports)

The request clock *RClk* is now generated by an AND operation between *LClkB*, being the inverted signal of *LClk*, and L_0, being the output signal of *D0*. It is asserted after both *LClkB* and *L0* are high and is de-asserted as soon as *LClkB* turns low. The on-phase period of *RClk* in each cycle of *LClk* is the sum of the delays of following gates:

$$MUTEX->AND0->C-ELEMENT->NOR->AND1 .$$

If such a delay is shorter than the half period of *LClk*, the *RAW* in this clock generator will be wider than that in Fig. 2. For instance, if the period of the clock *LClk* is 10ns and the summation of above delays is 1.5ns, the *RAW* in Fig. 10 is 8.5ns, while it is only 5ns in Fig. 2. Assuming a uniform distribution of the arrival time of Req_{Rx}+ in each cycle of $LClk_{Rx}$, the probability drops from 50% to 15% for a Req_{Rx} to introduce one cycle latency in the receiver. Fig. 11 depicts a comparison in *RAW*, Ack_{Rx} latency and $Data_{RxS}$ latency between the two clock generators.

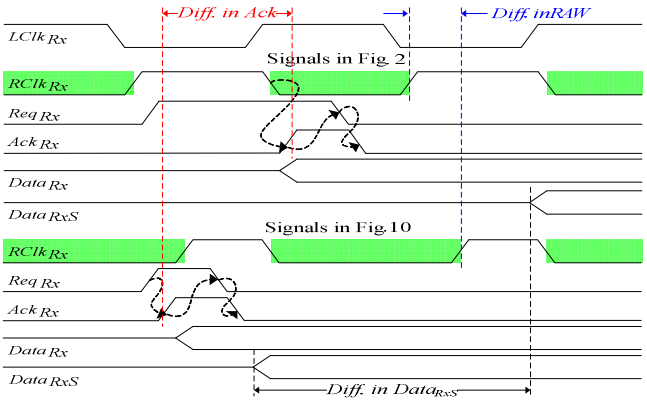


Fig. 11 Comparison in *RAW*, *Ack* and $Data_{RxS}$

The fixed delay line D_1 is employed in Fig. 10 to remain *LClk* at 50% duty cycle. The delay from *LClk*+ to *LClk*- is $d_{NOR}+d_{D0}+d_{D1}+d_{C\_ELE}$, and the delay from *LClk*- to *LClk*+ is $d_{NOR}+d_{D0}+d_{AND1}+(d^0_{MUTEX}+\Delta d_{MUTEX})+d_{AND0}+d_{C\_ELE}$. Since the delay time of *D1* is configured to match the total delay of *AND0*, *AND1* and the MUTEX as shown in (10), both of the delay paths are balanced.

It is well-known that there is no upper bound on the resolution time of the MUTEX elements [21]. A practical solution is to estimate the resolution time based on the mean time between failures (MTBF) according to (11). From [1, 19], 40 FO4 inverter delays are sufficient for metastability resolution, *i.e.*, for a MTBF of 10,000 years. It's long enough for normal applications.

$$MTBF = e^{d_{MUTEX}/\tau} / W \cdot F_C \cdot F_D \quad . \qquad (11)$$
$$where: \tau = d_{FO4}, W = 2 \cdot d_{FO4}, F_C = F_D = 1/100 \cdot d_{FO4}$$

Take the IHP 0.13-µm CMOS process as an example, where $d_{FO4} \approx 30ps$. The resolution time is estimated to be around $d_{MUTEX} = d^0_{MUTEX} + \Delta d_{MUTEX} \approx 40 d_{FO4} \approx 1.2ns$. Considering the delays of *AND0* and *AND1* in (10), we can fix the delay length of *D1* at 1.5ns, which equals to about $50d_{FO4}$. Based on the delay length of *D1*, the active phase of *RClk* is shown in equation (12), and furthermore, the duration of the *RAW* and the maximum acknowledge latency in this optimized clock generator are deduced as shown in equation (13). It reveals that the *RAW* is determined by the period of *LClk*. If $T_{LCLK} > 100 d_{FO4}$, typically which represents the shortest clock cycle for standard cells based SoCs [19], the optimized local clock generator provides a wider *RAW* than the traditional one shown in Fig. 2.

$$t_{RCLK=1} \approx d_{D1} + d_{C-ELE} + d_{NOR} \approx d_{D1}, \qquad (12)$$

$$t_{RAW} = t_{RCLK=0} = T_{LCLK} - t_{RCLK=1} \approx T_{LCLK} - d_{D1} \quad . \qquad (13)$$
$$\max(Latency_{Ack}) = t_{RCLK=1} \approx d_{D1}$$

B. *Optimized Input Port*

In this section, a double latching mechanism is applied to widen the safe region, and the port controller is improved to minimize the uncertainty on clock stretching.

1) Double Latching Mechanism

To widen the safe regions for the clock tree delay meeting $\Delta_{LCLKRx} < T_{LCLKRx}$, a double latching mechanism, which is based on the optimized clock generator, is proposed in Fig. 12. The first stage of latch *L1* loads the data from the transmitter, and then the second stage of latch *L2* feeds the data into the receiver. Since *L1* and *L2* are enabled by the acknowledge signals of the MUTEX, there is only one latch transparent at any time. Therefore, data is transferred by two mutually exclusive coupling latches in this scheme, instead of the single latch *L* in Fig. 1.

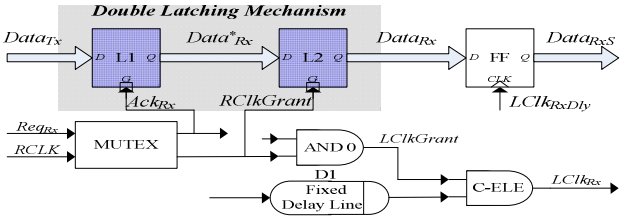


Fig. 12 Double latching mechanism

During the off-phase of *RClk*, *RClkGrant* remains low, and $Data_{Rx}$ is latched in *L2* stably. Any $LClk_{RxDly}$+ arriving at *FF* in the inactive phase of *RClk* can sample $Data_{Rx}$ safely. If *RClk* turns high, *RClkGrant*+ is triggered, and *L2* will get enabled to load $Data^*_{Rx}$. Once Req_{Rx}+ occurs simultaneously with *RClk*+, *RClkGrant* will be asserted by the MUTEX in a random resolution time. Consequently, any $LClk_{RxDly}$+ falling in the on-phase of *RClk* could conflict with loading $Data^*_{Rx}$ in *L2*. Therefore, the safe timing region for the clock tree distribution in this double latching mechanism is the off-phase period of *RClk* as shown in (14), which is exactly the same to the *RAW* in the optimized clock generator:

$$W_S = t_{RCLK=0} = t_{RAW} \approx T_{LCLK} - d_{D1} . \qquad (14)$$

Analyze (14) in the following two typical cases:

a) If $T_{LCLKRx} \approx 2d_{D1}$, *then* $W_s \approx d_{D1} \approx T_{LCLKRx}/2$. It occurs when d_{D0} is programmed to be zero and only d_{D1} is valid. The optimized clock generator is then set to be working as the traditional clock generator, while it provides a half cycle wide safe timing region.

b) If $T_{LCLKRx} >> d_{D1}$, *then* $W_S \approx T_{LCLKRx}$. With the increase of T_{LCLKRx}, the safe region is widened, but the hazard region is fixed to be d_{D1}. The percentage of W_s in T_{LCLKRx} will reach almost 100%, covering the entire clock period.

Above discussion shows that the range of W_S in Fig. 12 falls inside $T_{LCLKRx}/2 \le W_S < T_{LCLKRx}$. For small T_{LCLKRx}, there is half a clock cycle for clock tree insertion. For large T_{LCLKRx}, almost the entire clock period is safe. Fig. 13 illustrates that, for any clock period, the width of the safe timing region is approximately doubled in the double latching mechanism.

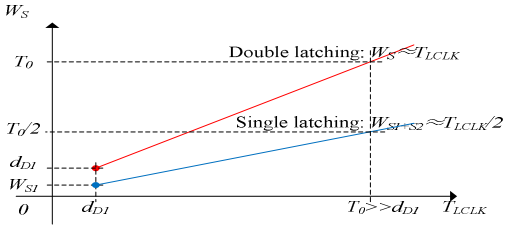
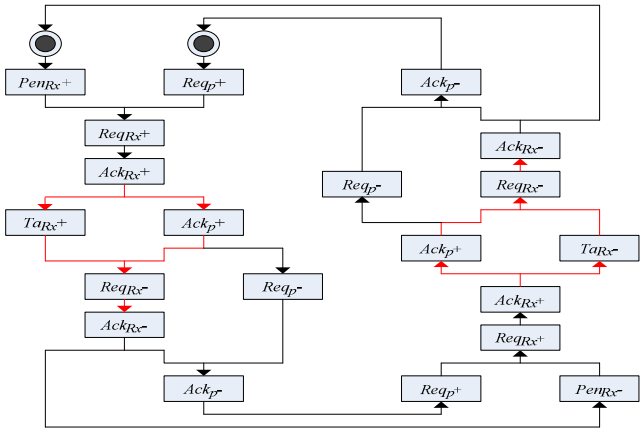


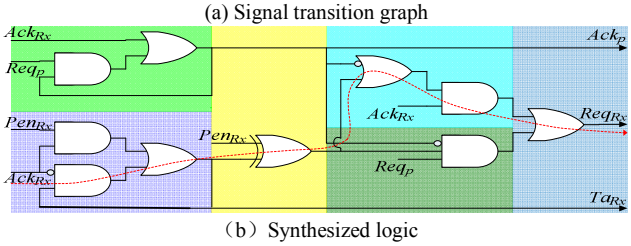
Fig. 13 Comparison of W_S in two mechanisms

2) Optimized Input Port Controller

In the traditional input port controllers, Req_{Rx}- is triggered by Req_P-. It means that $LClk_{Rx}$ can't be released until Req_P is de-asserted by the output port controllers. This accounts for the large and unpredictable stretching on $LClk_{Rx}$, another factor leading to synchronization failures for the multi-cycle clock tree delay. In Fig. 14, we present an optimized signal transition graph for the asynchronous input port controller, and the corresponding logic synthesized with Petrify [22].



(a) Signal transition graph



（b）Synthesized logic

Fig. 14 An optimized poll-type input port controller

We see that it is a poll-type input port controller with the transition sensitive enable signal Pen_{Rx}. Once $LClk_{Rx}$ has been paused, which is indicated by Ack_{Rx}+ from the clock generator, the input port controller will assert both Ack_p and Ta_{Rx}, and the combinational event of Ack_p+ and Ta_{Rx}+ will then trigger Req_{Rx}- to de-assert Ack_{Rx}, which signifies the release of $LClk_{Rx}$. These transitions are highlighted in Fig. 14(a), and their delay time determines the on-phase period of Ack_{Rx} and the maximum stretching on $LClk_{Rx}$. The longest delay path from Ack_{Rx}+ to Req_{Rx}- is shown as the red line in Fig. 14(b), which consists of only 4 complex gates. So the stretching on $LClk_{Rx}$ introduced by this optimized input port controller is small and predictable, as shown below:

$$0 \le Stretching_{LClkRx} \le t_{AckRx=1} = d_{AckRx+\to ReqRx-} + d^0_{MUTEX}. \quad (15)$$

We use n_{CT} to denote the maximum number of rising edges in the clock tree at one time, and it also determines the maximum number of the stretching on $LClk_{Rx}$ within the clock tree delay. To sample data correctly, a common safe region is required for the multi-cycle clock tree delay. The location of the common safe region provided by the input port is shown in equation (16), and the width of safe regions for different n_{CT} is deduced in equation (17). As an example, a scenario for $n_{CT}=3$ is depicted in Fig. 15.

$$(n_{CT}-1)\cdot(T_{LClkRx}+t_{AckRx=1})+t_{setup} < \Delta_{LClkRx} < n_{CT}\cdot T_{LClkRx} - d_{D1} - t_{hold}, \quad (16)$$

$$W_S(n_{CT}) \approx (T_{LClkRx} - d_{D1}) - (n_{CT}-1)\cdot t_{AckRx=1}. \quad (17)$$

Fig. 15 A scenario of common safe region

Note that the transfer acknowledge signal Ta_{Rx} is required in the receiver to indicate the arrival time of valid input data. So an additional latch, which is enabled by signal *RClkGrant*, is needed in the input port to synchronize Ta_{Rx} to $Data_{Rx}$.

3) Performance Comparison

Compared to the other schemes proposed in [11, 12, 19, 20] for GALS design with multi-cycle clock tree delays, this timed local clock tree insertion has the following advantages:

a) Low latency. In the double latching mechanism, *L1* and *L2* get enabled to latch the input data in opposite phases of $RClk_{Rx}$, and then the *FF* samples the data immediately at the next rising edge of $LClk_{RxDly}$. The maximum latency for data synchronization is only one clock cycle. Fig. 16 depicts the typical data flow in the improved input port.

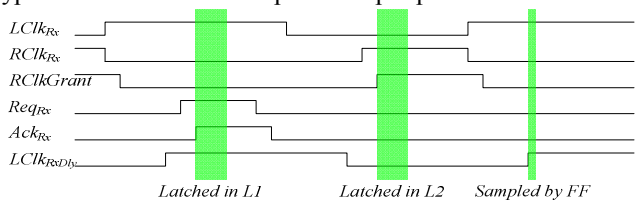


Fig. 16 The typical data flow

b) Low overhead. Except for an additional stage of latches used in the double latching mechanism, there is no extra overhead caused by the input port. Also, there is no timing restriction on the delay of the asynchronous port or the combinational logic following the sampling register *FF*.

IV. Experimental Results

A. Input Wrapper Simulation

As a simple example, an asynchronous input wrapper with the optimized clock generator and the novel input port was designed and simulated at transistor level using the IHP 0.13-µm CMOS process. The delay slice shown in [18], whose delay is measured to be 0.13ns in the experiment, is used to generate the delay lines. According to the analysis in section III.A, the delay of D_1 is fixed to be 1.56ns (12 delay slices), and the delay of D_0 is programmed to be 0.52ns (4 delay slices). Hence, the period of $LClk_{Rx}$ is 4.16ns (240MHz). In the input port controller, the on-phase duration of Ack_{Rx}, $t_{AckRx=1}$, is measured to be about 0.67ns, which decides the maximum clock stretching on $LClk_{Rx}$. Based on the above timing parameters, we derive the safe region widths from equation (16) for different clock tree depths n_{CT} as shown in Tab. 2. We see that, for $n_{CT} \le 4$, which indicates a rather large range of the clock tree delay within $\Delta_{LClkRx} < 4T_{LClkRx}$, it is possible in the wrapper to avoid synchronization failures by distributing the clock tree within the safe timing region.

Tab. 2 Safe region width vs. clock tree depth (T_{LClkRx}=4.16ns)

n_{CT}	*1*	*2*	*3*	*4*	*5*
W_S (ns)	**2.64**	**1.97**	**1.30**	**0.63**	**NA**
W_S / T_{LClkRx}	*63%*	*47%*	*31%*	*15%*	0

The simulation waveforms of the input wrapper using the Cadence Virtuoso Spectre in the case of n_{CT} = 2, i.e., $T_{LCLKRx} \le \Delta_{LClkRx} < 2T_{LCLKRx}$, is presented in Fig. 17, where the port request Req_P is asserted every 6.6ns (150MHz) in association with a 16-bit input data. First, the exact location of the common safe timing region, which is covered by the green area in Fig. 17, is calculated using equation (16):

$$4.95ns \approx T_{LClkRx} + t_{AckRx=1} + t_{setup} = T_0 < \Delta_{LClkRx} < T_1 = 2T_{LClkRx} - d_{D1} - t_{hold} \approx 6.65ns .$$

Second, $LClk_{Rx}$ is propagated through a delay line to arrive at *FF*, where $\Delta_{LClkRx} \approx 5.5ns \approx 1.32 \cdot T_{LClkRx}$, representing the multi-cycle clock tree delay falling inside the above common safe region in the receiver.

Fig. 17 illustrates the transfer of the first 3 data items in the input wrapper. Each input data is first loaded in *L1* when Ack_{Rx}=1, and then it is loaded in *L2* when *RClkGrant*=1, and finally it is sampled by *FF* at the next $LClk_{RxDly}$+. As Ack_{Rx} and *RClkGrant* are used as gating signals, their active phase should satisfy the minimum pulse width restriction, which is 0.14ns in the IHP 0.13-µm process. According to equation (15), $t_{AckRx=1}$ is dominated by the delays in the input port controller, which is measured to be 0.67ns. As to $t_{RClkGrant=1}$, it gets the minimum value, min($t_{RClkGrant=1}$), when $LClk_{Rx}$ is stretched, which is measured to be 0.28ns. Therefore, safe latching is guaranteed in the double latching scheme. Once data is loaded in *L2*, it will be sampled by *FF* immediately at the next rising edge of $LClk_{RxDly}$. Even if $LClk_{Rx}$ is stretched and *RClkGrant*+ is delayed, as seen in the transfer of data *D1* in Fig. 17, there is a sufficient timing margin from loading data in *L2* at *RClkGrant*+ to sampling data by *FF* at the next $LClk_{RxDly}$+. Therefore, safe data synchronization is achieved. Also, there is no additional latency in synchronization in the input wrapper.

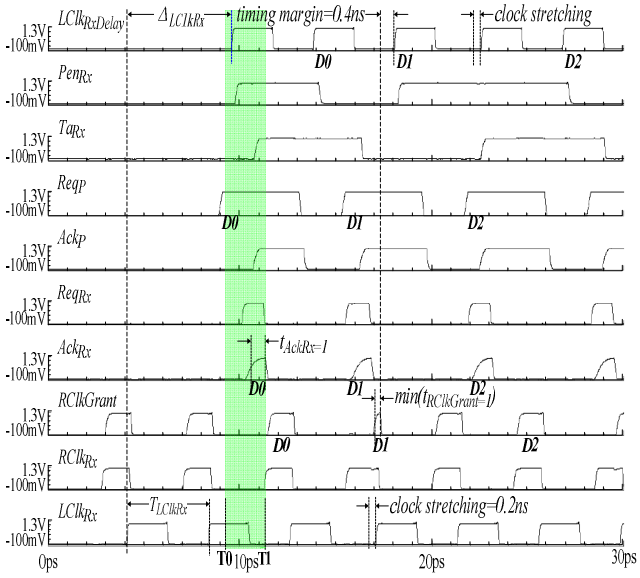


Fig. 17 Simulation waveforms with n_{CT}=2

B. Point to Point Communication

A point to point GALS system discussed in section II.B is implemented at gate level to demonstrate the throughput increase from the optimized scheme. On the transmitter side, the delay line is configured to generate a serial of different clock periods T_{LClkTx} as shown in Tab. 3. On the receiver side, the delay of the ring oscillator is fixed to be 12.48ns (96 delay slices), thus T_{LClkRx} is 24.96ns. Given T_{LClkRx} and T_{LClkTx}, the parameter *N* deduced from (3) is shown in Tab. 3.

Tab. 3 Configuration of the transmitter clock

Num. of delay slice	*36*	*24*	*22*	*16*
T_{LClkTx} (ns)	**9.36**	**6.25**	**5.72**	**4.16**
N in Equ. (2)	*5*	*7*	*8*	*10*

For each value of T_{LClkTx} in Tab. 3, the traditional scheme in Fig. 1 was firstly used to transfer 32 data items. Then the proposed clock generator and input port was applied in the scheme running simulation for exactly the same duration. Tab. 4 is the amount of data transfers accomplished using the optimized scheme and the percentage of improvement in data rate compared to the traditional scheme. It exhibits that the optimized scheme leads to much higher throughput, and the increase becomes pronounced for the large value of *N*. As an example, Fig. 18 presents a waveform fragment for *N*=7. In Fig. 18(a), where the traditional scheme is used, the *RAW* of the clock generator is $T_{LClkRx}/2$=12.48ns. All Req_{tx} are extended for $LClk_{Rx}/2$ and $LClk_{Tx}$ is periodically suspended. In Fig. 18(b), where the optimized clock generator is adopted, the *RAW* is (T_{LClkRx} - d_{D1}) =23.4ns, and no extension in Req_{tx} or suspensions in $LClk_{Tx}$ occurs. As a result, in Fig. 18(b) there were 8 data shown in signal $Data_{Rx}$ being transferred, but only 7 data items were transferred in Fig. 18(a). As high as 1/7 improvement in throughput is achieved, which matches well with the theoretical deduction in equation (5).

Tab. 4 Comparison in throughput (T_{LClkRx}=24.96ns)

T_{LClkTx} *(ns)*	*9.36*	*6.25*	*5.72*	*4.16*
Throughput in Fig. 2	*32*	*32*	*32*	*32*
Throughput in Fig. 10	*34*	*36*	*37*	*38*
Improvement	**6.3%**	**12.5%**	**15.6%**	**18.8%**
Improvement in Equa. (5)	*1/15*	*1/7*	*1/6*	*1/5*

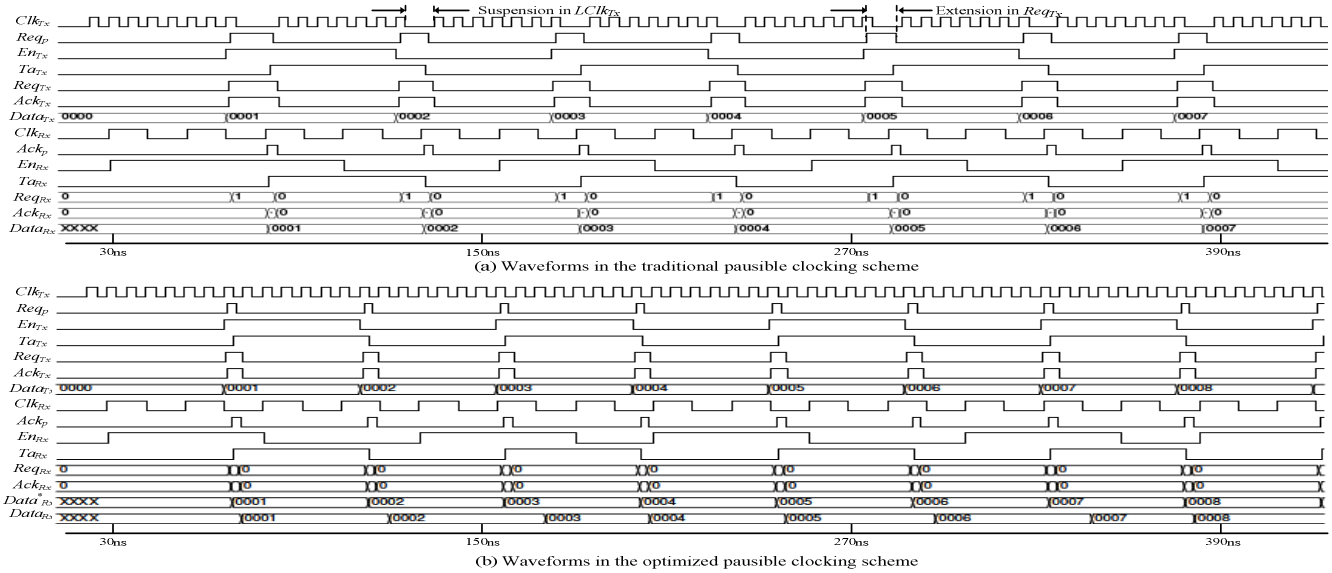


Fig. 18 Simulation waveforms (15-420ns) with two pausible clocking schemes

V. CONCLUSION

Pausible clocking based GALS design has been widely studied and is accepted as an approach to high performance SoCs and NoCs. However, some critical issues for the well known pausible clocking scheme needed to be solved. In this paper, we have proven that up to one-third reduction in throughput can be introduced by the acknowledge latency of pausible clock generators. We also demonstrate that, due to the uncertainty of clock stretching, there is no safe region for a clock tree with a multi-cycle delay. To address these issues, the pausible clocking scheme has been optimized in two aspects. The local clock generator is firstly optimized to minimize the acknowledge latency, and secondly, a novel input port is applied to maximize the safe region for clock tree delay. The proposed scheme has been verified in the IHP 0.13-µm CMOS process. This work contributes to improve throughput and reliability in GALS system design.

ACKNOWLEDGEMENT

This work has been supported by the European Project GALAXY under grant reference number FP7-ICT-214364 (www.galaxy-project.org).

REFERENCES

[1] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems*, 2003.

[2] Mark R. Greenstreet, "Implementing a STARI chip," in *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 1995.

[3] I. Sutherland and S. Fairbanks, "GasP: a minimal FIFO control," in *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 2001.

[4] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," PhD thesis, Standford Univ., 1984.

[5] K. Y. Yun and R. Donohue, "Pausible clocking: a first step toward heterogeneous system," In *Proc. Intl. Conf. Computer Design,* 1996.

[6] D. B. Bormann and P. Cheung, "Asynchronous wrapper for heterogeneous systems," In *Proc. Int. Conf. Computer Design,* 1997.

[7] A. E. Sjogren and C. J. Myers, "Interfacing synchronous and asynchronous modules within a high-speed pipeline," In *Proc. Intl. Symp. Advanced Research in VLSI,* 1997.

[8] J. Muttersbach, T. Villiger, and W. Fichtner, "Practical design of globally-asynchronous locally-synchronous systems," In *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 2000.

[9] S. Moore, G. Taylor, R. Mullins and P. Robinson, "Point to point GALS interconnect," In *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 2002.

[10] J. Kessels, A. Peeters, P. Wielage, and S.-J. Kim,"Clock synchronization through handshaking," In *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 2002.

[11] J. Mekie, S. Chakraborty, and D. K. Sharma, "Evaluation of pausible clocking for interfacing high speed IP cores in GALS framework," In *Proc. Intl. Conf. VLSI Design,* 2004.

[12] R. Dobkin, R. Ginosar, and C. P. Sotirou, "Data synchronization issues in GALS SoCs," In *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 2004.

[13] J. Muttersbach, "Globally asynchronous locally synchronous architecture for VLSI Systems," *PhD thesis, ETH Zürich,* 2001.

[14] T. Villiger, H. Kaslin, F. K. Gurkaynak, S. Oetiker, and W. Fichter, "Self-time ring for globally-asynchronous locally-synchronous systems," In *Proc. Intl. Symp. Advanced Research in Asynchronous Circuits and Systems,* 2003.

[15] D. Bormann, "GALS test chip on 130nm process," In *Proc. Workshop Formal Methods GALS Design,* 2005.

[16] F. K. Gurkaynak, "GALS system design – side channel attach secure cryptographic accelerator," *PhD thesis, ETH Zürich,* 2006.

[17] E. Beigne, F. Clermidy, S. Miermont, and P. Vivet, "Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC," In *Proc. Intl. Symp. Networks-on-Chip,* 2008.

[18] S. W. Moore, G. S. Taylor, P. A. Cunningham, R. D. Mullins, and P. Robinson, "Self-calibrating clocks for globally asynchronous locally synchronous systems," In *Proc. Intl. Conf. Computer Design,* 2000.

[19] Rostislav (Reuven) Dobkin, Ran Ginosar, and C. P. Sotiriou, "High rate data synchronization in GALS SoCs," In *IEEE Trans. VLSI Systems,* 2006.

[20] R. Mullins, and Simon Moore, "Demystifying data-driven and pausible clocking schemes," In *Proc. Intl. Symp. Asynchronous Circuits and Systems,* 2007.

[21] D. J. Kinniment, A. Bystrov, and A. Yakovlev, "Synchronization circuit performance," In *IEEE Journal of Solid-State Circuits,* 2002.

[22] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," IEICE Trans. Inf. and Syst., March, 1997.