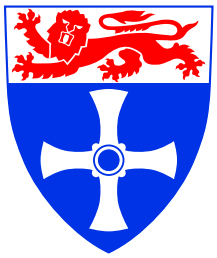
School of Electrical, Electronic & Computer Engineering

UNIVERSITY OF
NEWCASTLE UPON TYNE



Modeling and Performance Analysis of GALS Architectures

Sohini Dasgupta, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2006-114

April 2006

Contact:

Sohini.Dasgupta@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant GR/S12036

NCL-EECE-MSD-TR-2006-114

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Modeling and Performance Analysis of GALS Architectures

Sohini Dasgupta, Alex Yakovlev

April 2006

Abstract

Due to the increase in complexity of distributing a global clock over a single die globally asynchronous and locally synchronous systems are becoming an efficient alternative technique to design distributed SoCs. Number of independently clocked synchronous domains can be integrated by clock pausing, clock stretching or data driven clock techniques. Such techniques are applied on point-to-point inter-domain communication schemes.

We present here a comparison of these schemes and how it can be applied to an exisiting partitioned synchronous architecture to obtain a reliable, low latency and efficient clock control architectures. The comparison highlights the advantages and disadvantages of one scheme over the other in terms of logical correctness, circuit implementation, performance and relative power consumption. We also present here circuit solutions for stretchable and data driven clocking schemes. These circuit solutions can be easily plugged into existing partitioned synchronous islands. To enable early evaluation of functional correctness, this paper proposes the use of Petri net modeling technique to model the asynchronous control blocks that constitute the interface between the synchronous islands.

1 Introduction

Clocking circuits are becoming increasingly hard to design with larger chip sizes, higher clock rates and larger wire delays. The integration of various IP (intellectual property) cores on complex systems on chip requires a multitude of clock frequencies on a single die. Such integrations are enabled by modern deep sub-micron fabrication technologies in the form of chips with more than a billion transistors [11]. Globally asynchronous and locally synchronous (GALS) architectures aid such integration by allowing the synchronous blocks to operate independently with other synchronous blocks through asynchronous communication channels.

The GALS paradigm can be customised to meet the power and performance requirements to suit the target technology. There are a range of clocking strategies that can be applied to meet the above requirements. The modeling of the control circuit, enabled by tools like PEP [10] and its rendition into a gate level model using logic synthesis tools like Petrify [8], aids the exploration of the designs at a higher level of abstraction. This type of modeling abstractions are useful for analysis and validation of different design alternatives.

This report analyses the pausible, stretchable and data driven clocking approaches applied to a GALS architecture. This analysis will help the designer to make design decisions based on power and performance of different systems.

1.1 Contribution of the paper

The main goal of this paper is to present the comparison between three different GALS approaches. This comparison highlights the advantages and disadvantages of the three design solutions based on logical correctness, circuit implementations, power and performance analysis. The implementation of synchronous computational blocks are not cycle accurate, while the communication blocks are modeled in a cycle accurate manner. Petri net excels in its usefulness to model systems at higher levels of abstraction and tools like Petrify aid their translation into a gate level implementation. This type of modeling provides the designer with fast verification and implementation of the system. This paper presents the Petri net models of the three GALS architectures. These models are verified for correctness using in-house verification tools PUNF/CLP [7]. The verified models are fed into Petrify to produce logic equations for gate level implementation. We use two pre-synthesized blocks, namely, Mutual Exclusion Element [9] and FIFO [3] and these are plugged into the circuit implementation of each model obtained from Petrify. The gates are implemented on Cadence (AMS CMOS 0.35 micron technology library). This paper presents novel design solution for stretchable and data driven clocking scheme from prevalent conceptual models. The GALS architecture with pausible clocking scheme is obtained from [2] and compared for efficiency and power consumption with the above mentioned approaches.

2 Globally Asynchronous and Locally Synchronous Systems

To avoid the complexity of distributing a single global clock across the entire chip area and the varying power requirements for different blocks, the synchronous blocks can employ an independent internal clock. The frequency can be scaled up or down depending on the performance requirements of the system. In addition to local clock generator circuitry, GALS systems require synchronization between clock domains for reliable data transfer. The synchronization strategy stops the clock when the data transfer takes place to avoid metastability. The different clock control scheme involve different ways of stopping the clock. To obtain a GALS implementation for a given multi processor system, three clock control architectures can be employed. This implementation is extended to a system with two clocked domains, one producer and the other receiver, to replicate communication between two synchronous islands. The two clocked domains communicate via a two stage asynchronous FIFO. Such a system architecture is shown in Figure 1. The $Req1$ and $Req2$ will be replaced by a pause clock, stretch clock or start clock request depending on the type of clocking scheme.

The clock generator block is controlled by asynchronous port controllers, namely, the Input Port (IP) and the Output Port (OP). Such a system is scalable, with as many IPs and OPs as there are inputs and outputs from a particular synchronous island. A request-acknowledge pair of handshake signals accompanies each data entering or leaving the synchronous module. The validity of data is signalled by a four phase protocol depicted by $R+$, $A+$ $R-$ $A-$ and data is guaranteed to be valid between $R+$ and $A-$. The clock generator sends clock pulses to the synchronous module to carry out synchronous computations, while the communication is inactive and vice versa. There are three ways by which the clock generator can be controlled, namely, pasuible clocking scheme, stretchable clocking scheme and data driven clocking scheme.

2.1 Models of the clocking schemes

Figure 2 depicts the models of the consumer block, i.e. the async-sync interface, of each of the three GALS clocking schemes. For simplicity, the interaction of the communication interface with the synchronous module is not shown in the petri net models. This includes the signal $sync\_ack$ going to the synchronous module), which in turn releases an enable signal $send\_data$, denoting the availability of data to send on the producer side. Similarly, on the consumer side, the synchronous request($sync\_req$) is sent to the synchronous module after the reception of enable signal $accept\_new$ from it, depicted in Figure 1. The dotted lines in three models denote that signal $b+ \rightarrow A1+$and $b- \rightarrow A1-$ take place in the presence of the enable signal $accept\_new$, produced by the consumer synchronous module, which is not depicted in the figure 2.

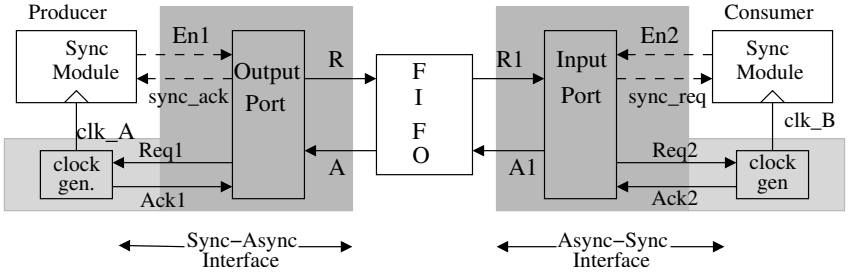
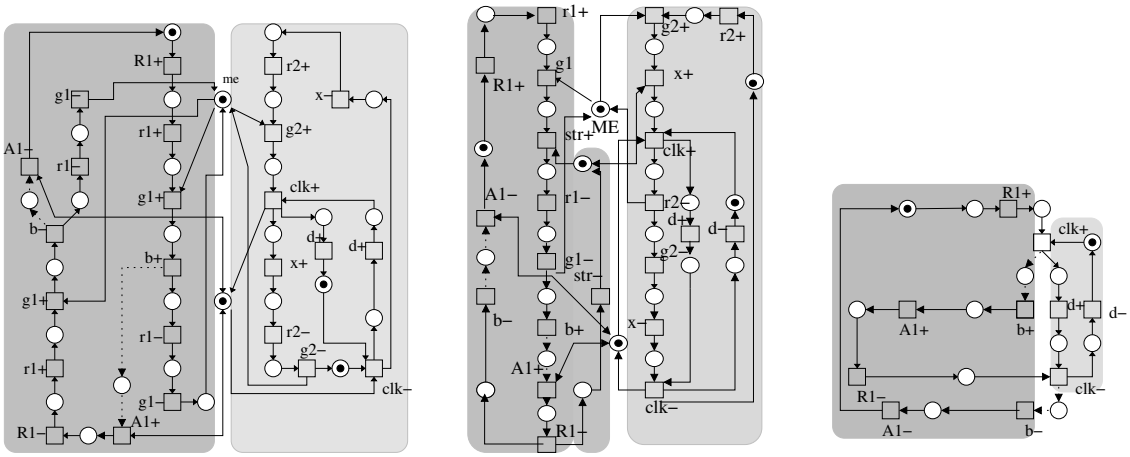


Figure 1: Overall system architecture for producer-consumer interface



(a) Pausible clocking scheme (b) Stretchable clocking scheme (c) Data driven clocking scheme

Figure 2: Petri net models of GALS architectures

Pausible clock - The pausible clocking scheme offers an elegant solution to metastability issue which comes into play when there is cross domain communication. Pausible clocks are characterised by a free running clock. A Mutual Exclusion (ME) element is inserted in the circuit to allow the clock to be interrupted when a data is ready to be transferred. The interruption of the clock enables safe transfer of asynchronous data. and the Petri net model of the async-sync interface (consumer side) of this system is shown in Figure 2(a). The signal $r1$, produced by $R1$, requests for a clock pause, while signal $r2$ requests for granting the clock pulse. Signals $g1$ and $g2$ are mutually exclusive and granting of $g1$ interrupts

the clock. This leads to an asynchronous data transfer. This request is acknowledged on reception of the positive edge of the clock signal. Once the clock goes low, it is triggered again after a tunable delay d.

Stretchable clocking scheme- A stretchable clock can also be viewed as a free running clock like the pausible clock. The difference between two is that a stretchable clock knows in advance that the next clock cycle should wait for an asynchronous input. Therefore only in the absence of input request signals, the clock would be free running. This architecture leads to an increased throughput, since the request does not have to compete with the clock for an asynchronous data transfer. The async-sync interface of this system is depicted in Figure 2(b). The signal $clk+$ can only proceed if signal str is low, which denotes that there is no data to tranfer from the FIFO. When the signal str is high, the data is transfered from the FIFO to the consumer block. The request received is acknowledged when the clock goes low. The delay line, like the pausible clocking schemes delays the rising and falling of the clock signal clk. The delay line is parallel to the arbitration block in the local clock generator block denoted by the light grey shaded portion.

Data driven clock- In data driven clock scheme clock edges are produced in response to the presence of data at the input ports of the IP block. Therefore, the clock is not free running, unlike pausible and stretchable clocking schemes. The Petri net model of the async-sync interface of this system is shown in Figure 2(c). Signal clk is asserted on the reception of the positive edge of the signal $R1$. clk is deasserted on the reception of the negative edge of $R1$ and after a tunable delay d. The clock is inactive in the absence of signal $R1$.

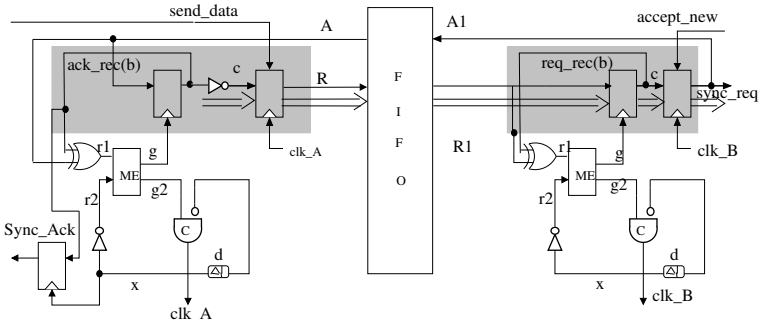


Figure 3: Pausible clocking scheme

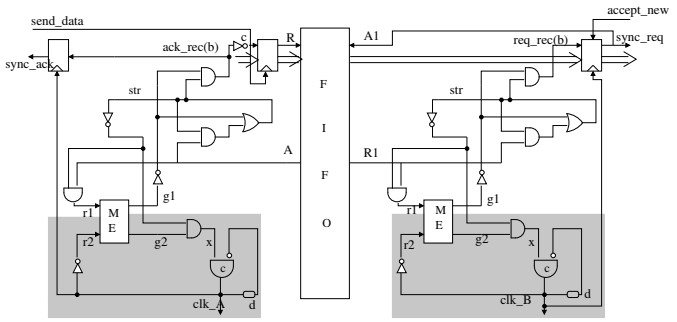


Figure 4: Stretchable clocking scheme

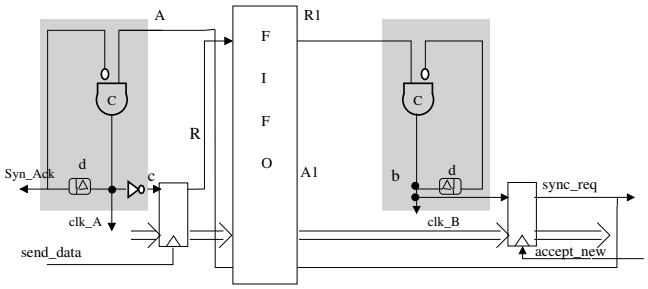


Figure 5: Data driven clocking scheme

3 Verification and Logic Synthesis

The Signal Transition Graph models [8] shown in Figure 2 have been designed on PEP tool and verified for functional properties like safeness and deadlock freedom using in-house tools PUNF/CLP. The circuit implementation for stretchable and data driven clocking schemes have been obtained from the logic synthesis of their respective STG descriptions, using Petrify. In order to logically synthesize a given STG, using Petrify, it is necessary to check that it is free from safeness and liveness problems.

The statistics obtained after verification are listed in Table 1. This table presents the number of conditions and events generated from each of the models. It also shows that each of the models satisfy safeness and liveness properties. We also present the statistics for pausible clocking scheme for the sake of comparison with the other models.

The verified models of the sync-async interface (producer side) and async-sync interface (consumer side), depicted in Figure 2 (b) and (c) together with the FIFO were composed together to form a closed system as depicted in Figure 3, 4 and 5.

Model name	|B|	|E|	Liveness	Safeness
Pausible clock	115	81	√	√
Stretchable clock	37	30	√	√
Data driven clock	20	15	√	√

|B| = Number of Conditions

|E| = Number of Events

Table 1: Verification statistics for Petri net models of GALS architectures

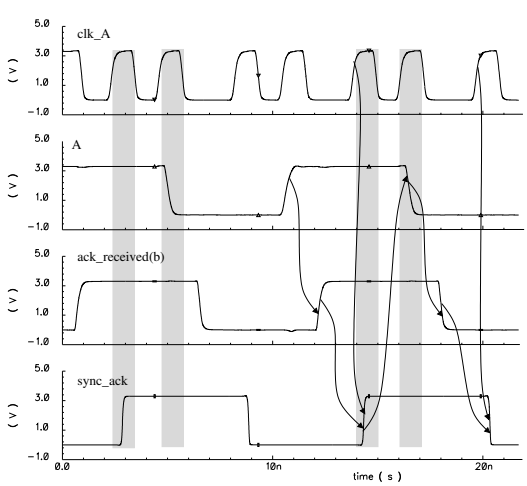
4 Circuit Implementation

In this section we present the circuit implementation of the three clocking schemes. The systems consists of producer and consumer synchronous modules communicating via a two stage FIFO. The leftmost block and the FIFO block constitute the interface between synchronous producer and asynchronous receiver, while the block on the right side of the FIFO denote the interface between asynchronous producer and synchronous consumer.

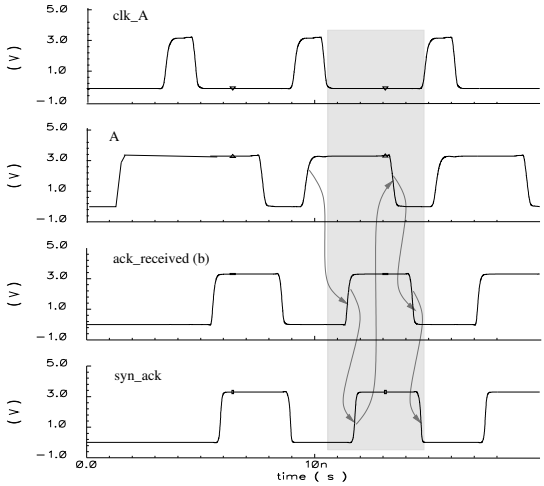
Pausible clock - The implementation of a producer-consumer block over an asynchronous interface is depicted in Figure 3. The asynchronous interface arbitrates between granting in favour of the $r1$ signal, to transfer data to subsequent synchronous blocks or a clock request, to generate clock (clk_A) for its locally synchronous module. If the $r1$ is granted the data is latched in the first latch and the hold is released on the mutex. This allows clock request to win over the mutex. Therefore, data is stable before the clock arrives at the next stage of latch avoiding metastability at the second edge triggered latch. Figure 6(a) shows the phase relation between signals clk_A, ack, ack_rec, $sync\_ack$. The shaded portion denotes the window when asynchronous data is received. The synchronous module always waits for a synchronous syn_ack. On reception of the synchronous $sync\_ack$, the module releases an enable signal for new data transfer. This type of design methodology is also explored in [1].

Stretchable clock - The system architecture of this scheme is depicted in Figure 4. The assertion of the stretch signal (str) prevents the clock from going high before the assertion of signal ack_rec/req_rec, deasserting $R/A1$, which in turn deasserts signal str . On the producer side the synchronous module waits for a synchronous $sync\_ack$, in a manner similar to pausible clocking scheme. Hence, signal ack_rec has to be synchronized to the clock to produce $sync\_ack$. As can be seen from the stretchable clock architecture in Figure 4, signals ack_rec+ and $clk+$ are mutually exclusive due to signal str (this can also be seen on the consumer side (async-sync interface) of the system, in the Petri net models shown in Figure 2(b), where req_rec+ is mutually exclusive to $clk+$). Therefore, positive edge of signal ack_rec cannot be synchronized on the positive edge of signal clk. If the signal ack_rec is synchronized to the negative edge of the clock cycle with a flip-flop, the system could run into a deadlock. This is due to the fact that if signal clk has already gone low before the triggering of signal $str+$, and then if $str+$ occurs stopping signal $x+$(which causes $clk+$) from firing, signal $ack+$ would wait for the faling edge of signal clock, which would not be triggered till $str-$ occurs . Hence, ack_rec+ will never meet the set up and hold time of the falling edge of clock signal. Therefore the only solution is to use a latch, instead of a flip flop. The latch is made to sample the signal ack_rec when the clock is low. This synchronized ack_rec is then sent to the synchronous module, which in turn sends an enable signal to indicate a data-ready-to-send status. This enable signal latches the $ack\_received'(c)$ in the final set of latches to assert the request signal for sending new available data. A similar scheme is presented in [5]. A phase relation between signals *clk_A, ack, ack_rec, Sync_ack* at the sync-async interface for stretchable clock, similar to pausible clocking scheme, is depicted in Figure 6(b).

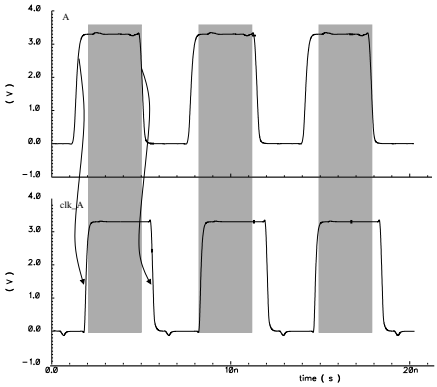
Data Driven clock - Such an architecture is depicted in Figure 5. Since, power is an important issue in SoC applications, design methodologies which provide circuit solutions with reduced power consumption becomes highly attractive. In this scheme the local clock oscillates at a frequency determined by the availability of data signalled by the request signal. Therefore the circuit is switched off when there is no data to send. This scheme significantly reduces power consumption as clock is only started when enough inputs have been received to carry out a particular computation. Unlike the previous two clocking schemes, there is no added synchronization required for the ack_rec/req_rec, since the signals are already synchronized to the clock and can be directly sent to the synchronous module on the reception of enable signals as denoted in the figure. An extensive design solution for this approach can be found in [6]. The simple phase relation between $a1$ and clk_A, on the sync-async interface for this scheme is shown in Figure 6(c).



(a) ack+->ack- phase relation for pausible clock



(b) ack+->ack- phase relationship for stretchable clock



(c) ack+->ack- phase relationship for data driven clock

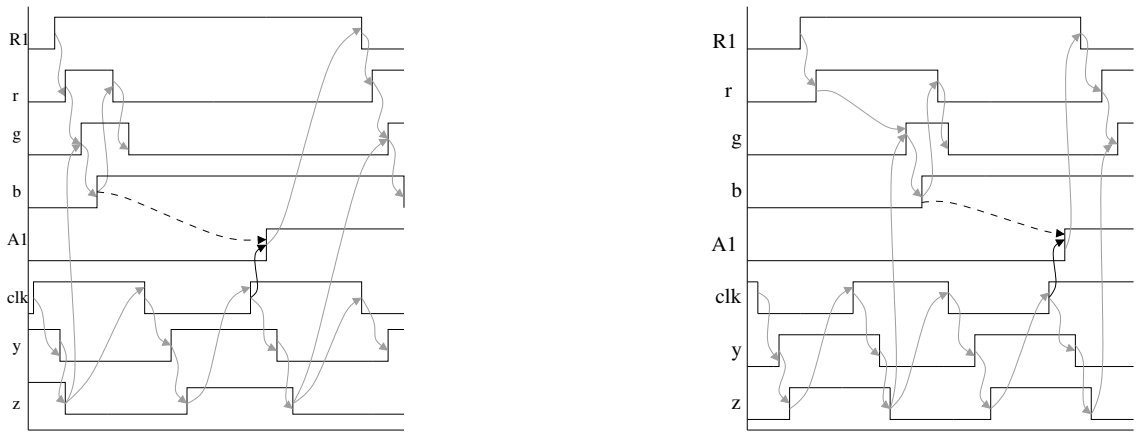
Figure 6: Asynchronous Communication-phase relationship at the producer block

5 Performance Analysis

In this paper, it is assumed that the system is partitioned logically into synchronous islands and that they communicate with other synchronous blocks through an asynchronous interface. The asynchronous interface interacts with the clock generation circuit of these synchronous block for cross domain data transfer. The Petri net models of the asynchronous interface and clock control circuit developed are fed to Petrify to give logic equations to build the gate level implementation of the architecture. These circuits are simulated on Cadence. We used mixed signal simulations to aid the monitoring of several signals using digital specification, while leaving other parts of the circuit to run analog simulations. The design has also been incorporated with various digital blocks to reduce the time taken for analog simulation.

5.1 GALS system characterization parameters

To characterize any design based on SoC applications, we need to define some metrics that are applicable to power and performance of a system. Similarly, for GALS systems we need to define such metrics. These



(a) Best case latency

(b) Worst case latency

Figure 7: Req-Ack latency in the producer block of pausible clock

metrics are evaluated to analyze an architecture for studying the effects of different system parameters on the performance of the system.

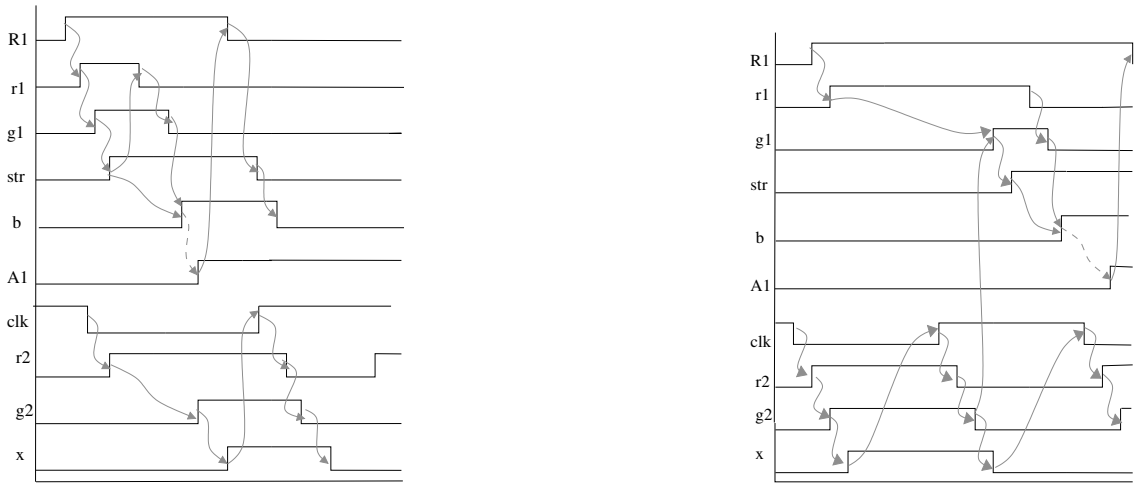
The metrics that are relevant for the analysis of pausible clock circuitry of GALS architecture are the number of times a clock is paused for a given simulation time and the average latency incurred due to such clock pauses. Another important system analysis metric for efficiency comparison is the throughput of the system, i.e., the average production/processing capacity of a system.

Due to increasing clock frequencies and smaller device sizes, it is becoming particularly important to consider the total power consumption metric in deciding on a particular design methodology. GALS based architectures reduce power consumption due to the ability to shift to an asynchronous mode when the local clock of the synchronous system is paused. Hence, a comparison of energy consumption in different GALS architectures would help choose between the different asynchronous communication circuitry. Therefore, an analysis of these metrics is useful for the designers to estimate the performance penalties in using one clocking scheme over the other.

5.2 Mixed Signal Simulation

For simulating the pausible clock circuitry, we have employed mixed signal simulation technique. This technique enables us to simulate a combination of both analog and digital signals. We avoided using complete analog simulation technique in order to write functional blocks in verilog to be incorporated in the system and small verilog codes to monitor and evaluate the metrics, discussed above, for analysing the system. Hence, the inputs and outputs could be monitored digitally. The core analog blocks of the circuit can be efficiently wrapped by digital blocks which preserves the precise estimation of delays within these analog blocks.

Such a simulation can be done by connecting the inputs and outputs of the analog block (pausible clock circuitry) into small functional blocks which connects the analog inputs and outputs to digital inputs and outputs, respectively for analog-to-digital and digital-to-analog conversion.



(a) Best case latency (b) Worst case latency

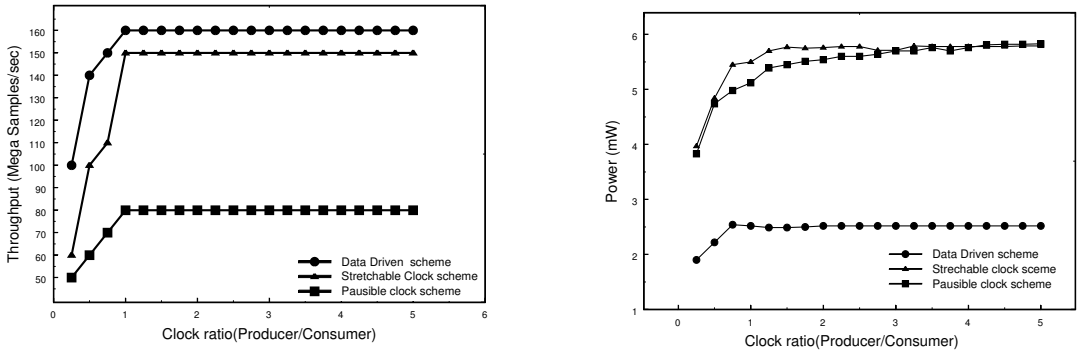
Figure 8: Req-Ack latency in the producer block of stretchable clock

5.3 Model Level Analysis

We present here the analysis of the best and worst case delay between the pausible and stretchable clocking schemes. These delays can help us estimate the usefulness of using one scheme over another. We take into consideration the latency between sending a request $R1$ from the FIFO to the consumer module and receiving an acknowledge $A1$ at the FIFO input from the consumer module, to be sent to the producer module, denoting a complete transfer of an item of data sent by the producer. Since, the delay of the logic circuit (i.e. the logic gates) for asynchronous data transfer and clock generation is comparable and can vary with different implementations of the same logic, we mainly take into account the number of clock cycles needed to obtain the desired output. Here, we assume that signals $y+$ and $r1+$ for pausible clock and signals $r1+$ and $r2+$ for stretchable clock arrive with a delay of δ unit of time between them, such that δ is greater than time under which metastability may happen within the mutual exclusion element, to avoid the possibility of the resolution leading to a random selection of outputs from the element. We present the timing diagrams for the best case and the worst case delay scenarios for both the clocking schemes in Figure 7 and 8. For the best case delay, we assume that $r1+$ for both the clocking schemes arrive δ unit of time before signal $y+$ and $r2+$ for pausible and stretchable clocking schemes, respectively. As shown in Figure 7(a) , signal $A1+$ occurs after at least one clock cycle for pausible clocking scheme. In contrast, for stretchable clocking scheme, signal $A1+$ occurs in less than half a clock cycle. The dashed line denotes that $A1$ is caused by signal b in the presence of an enable signal $accept\_new$ not shown in the graph.

Such an observation is due to the fact that in pasusible clocking scheme, the final set of latches, shown in Figure 3 waits for a positive clock edge before sending the signal to the next clock domain. It is easy to observe that the arrival of signal b misses the first clock edge and has to wait for the next clock edge to appear. The latch is enabled by a signal sent from the producer module which indicates when it is ready to receive new item of data. In stretchable clocking scheme, as shown in Figure 4, the latches are triggered when clock goes low. This latch also waits for the enable signal sent by the consumer module, similar to the enable signal used in pausible clocking scheme, when it is ready to receive new data.

Similarly, for the worst case scenario where signal $y+$ and $r2+$ from pausible and stretchable clocking



(a) Throughput analysis

(b) Effective power consumption at interfaces

Figure 9: Throughput and Power Consumption for the GALS architectures

scheme, respectively, arrive δ unit of time before $r1+$ for both the clocking schemes. The delay between the reception of request $R1$ and the emission of acknowledge $A1$, is over one and a half clock cycle for pausible clock. For stretchable clocking scheme, the delay is just over a clock cycle. Therefore, it is observed that we are able to save half a clock cycle on every data transfer for stretchable clocking scheme.

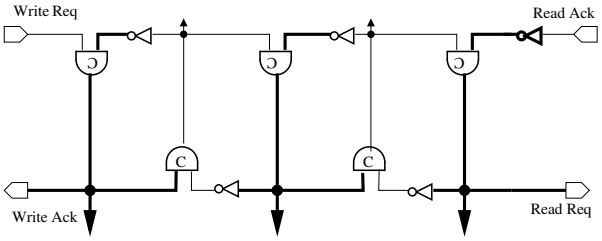
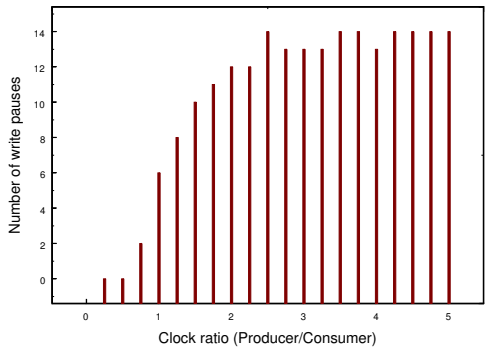


Figure 10: FIFO design

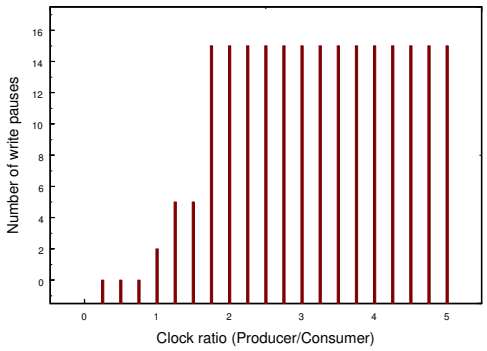
6 Circuit Level: Experimental Results

This section presents the results of power and performance analysis of GALS architecture with the three clocking schemes.

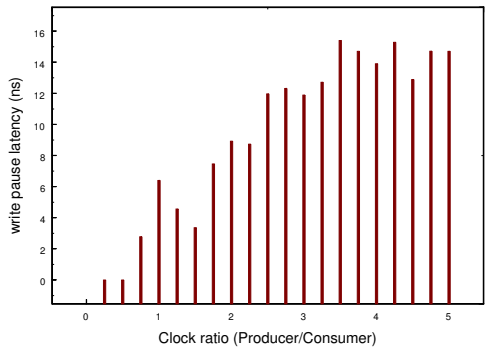
In our experimental setup, we use a 2 stage FIFO inter-module communication scheme. In the experiments we vary an input parameter, namely, the producer clock frequency. It is varied from 125 MHz to 2.5 GHz to observe the behaviour. The frequency of the consumer clock is maintained at 500 MHz. Higher frequencies are possible depending upon the complexity of the producer and consumer blocks. The frequency of the clock is varied by varying the delay d, in the three clocking schemes. This delay extends the clock period, thus changing the frequency of the clock. The ratio between the producer clock and consumer clock is called clock ratio. The clock ratio is varied from 0.25 to 5 in steps of 0.25. This allows us to study the different phase relationship between the consumer and producer clocks.



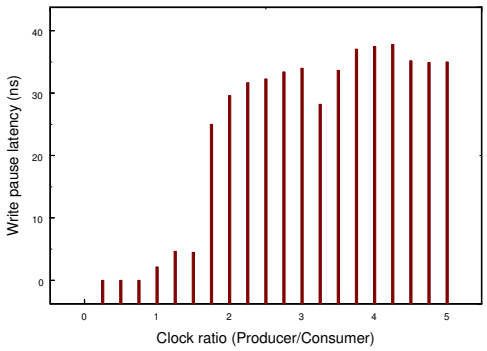
(a) Number of clock pauses in producer for pausible clock



(b) Number of clock pauses in producer for stretchable clock



(c) Total clock pause time in producer for pausible clock

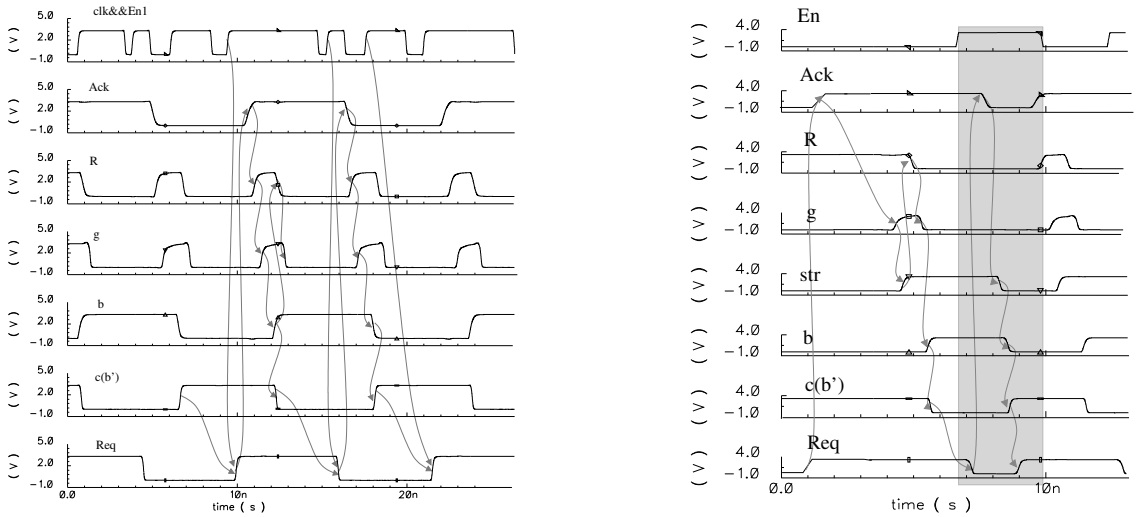


(d) Total clock pause time in producer for stretchable clock

Figure 11: Pause and latency Analysis

Figure 11(a) and (b) shows the number of clock pauses in the producer for pausible and stretchable clocking schemes, as the clock ratio is increased. We see that as the frequency of the producer clock increases, the number of pauses increases for a given simulation time. The asynchronous data transfer logic operates at a particular frequency. This frequency depends on the rate of production of R signal from the producer block and rate of reception of A signal from the consumer block. The transfer frequency becomes smaller than the frequency of the producer clock as the producer clock frequency increases and becomes higher than the consumer clock frequency. Hence, it takes longer to finish the cycle that de-asserts the grant on the arbiter. Due to this we observe more clock pauses as the period of the clock is too small to mask this delay. At lower frequencies, the time period is large enough to mask the pause during its lower half period.

The number of clock pauses in pausible and stretchable clocking scheme are comparable due to the scenario described above. But it can be observed from the graphs depicting total time incurred by these latencies that they are no longer comparable. The stretchable clocking scheme incurs longer latencies than than pausible clock. This is because the clock is only asserted when signal str is low. The arrival of signal



(a) R–>R+ delay for pausible clock

(b) R–>R+ delay for stretchable clock

Figure 12: Request delay analysis

A on the producer side or signal $R1$ on the consumer side, asserts signal str. When the producer frequency increases and becomes more than the consumer frequency, the FIFO gets filled up as more requests are produced than it can be consumed by the consumer module. Hence, the de-assertion of signal A is delayed. This phenomenon is exemplified in Figure 10. The FIFO is made up of a set of C-elements [4]. The shaded lines depict the signals that are asserted, while the non-shaded lines depict de-asserted signals. It can be observed that when the FIFO is full $Write\ Ack$ (A) remains asserted and is only de-asserted when an item of data is read from the FIFO, i.e. $Read\ Ack(A1)$ is asserted. The delay in the de-assertion of A, delays the de-assertion of signal str, which in turn delays the assertion of signal clk. This leads to a prolonged clock stretch. Such an occurrence is not observed in pausible clocking scheme. This is because, the reception of $b+$ immediately releases the grant on the arbiter and at this stage, the clock can arbitrarily win the grant to assert itself. This justifies the graphs shown in Figure 11(c) and (d).

Figure 9(a) shows the impact of changing clock ratio on the throughput of the communication channel. We observe that as the frequency of the consumer clock increases the throughput increases linearly up to clock ratio 1. This is because more data is being read by the consumer in the same period of time. After this time, the throughput reaches a saturation point. This is because the consumer clock operates at a lower clock frequency compared to the producer clock. Hence, there is no additional increase in throughput.

The throughput values obtained for stretchable and data driven clock are higher than pausible clock. This is due to the delay between two consecutive rising edges of the request signal ($R+$). A detailed phase relation between signals that cause this delay is shown in Figure 12. It is observed that, this delay is $12ns$ for pausible clock and $8ns$ for stretchable clocking scheme. The throughput is maximum for data driven clock. It is higher than stretchable scheme since the signal A in the stretchable clocking scheme waits for synchronization for crossing over to synchronous domain to produce $Sync\_ack$. On the contrary, no such synchronization is needed for data driven clock as the clock starts when there is data to transfer and hence the signal A thus produced is already synchronized to the clock. This explains the trend of the curves in the graph that depicts the throughput of the different clocking schemes.

Figure 9(b) shows the power consumption, at an operating volatge of $3.3V$, with varying clock ratio.

This plot refers to the effective power consumed over the time period needed to send a packet(same for all three protocols). We observe that as the clock ratio increases power consumption increases. This is because, as clock ratio increases, the throughput and operating frequencies of the synchronous islands, increases leading to an increased power consumption. It is observed that the lowest power consumption is demonstrated by data driven clocking scheme as it doesn't have a free running clock and can be switched off when there is no data to send. Since, the implementation of the FIFO is same for all the protocols, complexity of port controller implementation of pausible and stretchable clocking schemes is also a factor that gives rise to such observations.

7 Conclusion

This paper presented the classification of different clocking schemes for Globally Asynchronous and Locally Synchronous architectures. These schemes have been modeled using Petri nets. A Petri net model of these interconnect architectures allows the designer to use existing logic synthesis tools, like Petrify to obtain gate level design solutions. Such solutions for GALS systems with stretchable and data driven clocking schemes have been presented in this paper. All the three clocking schemes exhibited reliable data transfer between the synchronous domains. A complex SoC can exploit any of the above given architectures depending on the requirements of the target system. These models can be plugged into existing partitioned synchronous blocks. These schemes can be extended to employ various power reduction methodologies in the wrapper without affecting the synchronous IP blocks.

In addition to the classification and design solutions for the three clocking schemes this paper also analyses the three systems on performance and power consumption criteria. Stretchable and data driven clocking schemes demonstrated higher throughput and lower power consumption charateristice, respectively, compared to the prevalent pausible clocking scheme. The stretchable and pausible clocking schemes are further compared on two other metrics , namely, the number of times the clock is paused or stretched and the total latency incurred by these pauses. Such an analysis aids the designer to make different design decisions based on power and performance.

Future work Future work includes the development of a library of such Petri net models of each of the GALS clocking techniques, for different input coupling schemes (e.g. arbitrated, synchronized and sampled). We are also in the process of developing an automated GALS design tool which plugs these interconnects to already partitioned synchronous islands. This tool would ease the integration of the different interconnect models with the existing partitioned synchronous islands.

References

[1] K. Yun, R. P. Donhue, Pausible clocking: A First Step Towards Heterogeneous Systems. In proceedings of International Conference on Computer Design, October 1996, Austin, TX.

[2] S. W. Moore, G. S. Taylor, R. D. Mullins, P. Robinson, Point-to-Point GALS Interconnect. In proceedings of Eighth International Symposium om Advanced Research in Asynchronous Circuits and Systems, 2002.

[3] I. Sutherland, Micropipelines: Turing Award Lecture. In Communications of the ACM, 32(6):720-738, June 1989.

[4] J. Sparso, S. Furber, Principles of Asynchronous Circuit Design - A System's Perspective. Kluwer Academic Publishers, 2001.

[5] J. Kessels, A. Peeters, P. Wielage, S. Kim, Clock Synchronization through Handshake Signalling. In International Symposium on Asynchronous Circuits and Systems, 2002.

[6] M. Krstic, E. Grass, C. Stahl, Request Driven GALS Technique for Wireless Communication Systems. In proceedings of 11th International Symposium om Advanced Research in Asynchronous Circuits and Systems, 2005.

[7] V. Khomenko, Model checking based on prefixes of petri net unfoldings, PhD thesis, University of Newcastle, (2003).

[8] J. Cortadella, M. Kishnivsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Synthesis of Asynchronous Controllers and Interfaces. Springer, Berlin, 2002.

[9] C. Mead, L. Conway, Introduction to VLSI systems. Addison-Wesley Publication, October 1980.

[10] S. Melzer, S. Römer, and J. Esparza, Verification using PEP. In Proceedings of AMAST, 1996.

[11] S. Naffziger, The Implementation of a 2-core Multi-threaded Itanium Family Processor. In Proceedings of ISSCC, 2005.