System Level Power and Performance Modeling of GALS Point-to-point Communication Interfaces

Abstract

Due to difficulties in distributing a single global clock signal over increasingly large chip areas, a globally asynchronous, locally synchronous design is considered a promising technique in the system on a chip (SoC) era. In the context of today's increasingly complex SoCs, there is a need for design methodologies that start at higher levels of abstraction. Much of the previous work has been devoted to design of asynchronous communication schemes such as mixed clock FIFOs and pausible clocks for globally asynchronous, locally synchronous systems, but at low levels of abstraction, such as circuit level. To enable early design evaluation of such schemes, this paper proposes to use a SystemC-based modeling methodology for the asynchronous communication among various locally synchronous islands. The modeling framework encompasses various levels of abstraction and enables system-level validation of circuit or RT level hardware descriptions, as well as their impact on high-level design decisions.

1 Introduction

Due to increasing die sizes, higher clock speeds and high clock skews, future digital VLSI designs will require a paradigm shift from the globally synchronous design style. In addition, the integration of various IP (Intellectual Property) cores on complex systems on a chip requires a multitude of available clock frequencies on a single die. A globally asynchronous, locally synchronous design (GALS) paradigm enables such integration, by allowing for synchronous blocks to operate asynchronously with respect to each other. In such a scenario, not only the speed, but also the voltage of each block can be customized or chosen so as to meet the power and performance requirements of the target application. This design paradigm is also particularly attractive for a system-on-chip where circuit building blocks (or IP blocks) from a number of design houses are integrated onto a single chip.

Communication between the building blocks of a SoC is a complex problem, particularly when a range of clocking strategies has to be tailored to each building block in order to obtain a required performance within a power budget. Also, in the context of the increasing complexity of systems-on-chip (SoCs) and time-to-market pressures, the design abstraction has to be raised to the system level to increase design productivity. Transaction level modeling [6], which is enabled and supported by system level languages such as SystemC, can be used to separate the computation components from the communication components. Communication can be modeled as channels, while transaction requests take place by calling the interface of these channel models. Unnecessary details of communication and computation can be hidden in a TLM and can be added later. This enables speeding up of simulation and allows for exploration and validation of design alternatives at a higher level of abstraction.

1.1 Paper contributions

This paper addresses the problem of power and performance analysis of GALS based systems, using transaction level modeling, in which the computation components are modeled as processes (with or without cycle accurate representations) while the communication is modeled in a *cycle accurate* manner. SystemC excels in its usefulness to model designs at system level, while still supporting synthesizable RT level hardware descriptions. Thus, a seamless refinement of a design can occur such that each part of the design is implemented independently, without resorting to changes of other parts of the design. This paper advances the state-of-the-art by providing ways of using SystemC to model mixed clock communication channels of primarily two types: mixed clock FIFOs [2,3] and pausible clocks [4,5]. The computation processes are modeled as untimed algorithmic descriptions in a procedural language (such as C) that interface with the communication channel in a cycle accurate manner.

To this end, this paper introduces a system level methodology amenable for analyzing the power and performance characteristics of asynchronous/mixed clock communication interfaces that have already been ***designed*** and ***validated*** at ***circuit-level***. A system level model of such interfaces built by abstracting these circuit level characteristics enables plug-and-play capabilities for these interfaces into any SoC application and provides the designer fast analysis of the communication overhead in terms of power and/or performance. The proposed system level modeling methodology also enables design exploration of these applications in terms of which communication interfaces or architectures are more suitable for deploying. Since power is an important metric for SoC applications, providing reliable estimates for the power overhead introduced by various on-chip communication schemes on target real life SoC applications is of extreme importance, and is thus a main ingredient of the approach proposed herein.

1.2 Paper Organization

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 introduces GALS based systems, while Section 4 describes specific GALS based communication architecture that we model in this paper. Section 5 shows how system level analysis would aid a system designer make design decisions based on power and performance. Section 6 shows the experimental results, while Section 7 concludes the paper with final remarks and directions for future research.

2 Related Work

Chapiro has first introduced and studied GALS systems in detail in his thesis [1]. His work covers metastability issues in GALS systems and outlines a stretchable clocking strategy which provides a mechanism for asynchronous communication.

In GALS systems, the asynchronous modules have to communicate with each other asynchronously which may lead to metastability issues. Chelcea *et al.* [2] use mixed clock FIFOs as low latency communication mechanism between synchronous blocks. Cummings [3] uses a memory based mixed clock FIFO to communicate between different clock domains. We use his work and a pausible clocking scheme by Yun *et al.* [4] to model GALS communication interfaces at the system level. Mutterbasch *et al.* [5] have implemented asynchronous wrappers around synchronous blocks. Most of this existing work is done at the RTL or circuit level. Thus, there is a need for system level tools for analyzing these communication architectures, which we attempt to address in this paper. Transaction level modeling [6] has been researched in the system level language and modeling area. The concept of *channel*, which enables separating communication from computation, has been introduced and discussed in [7]. [8] broadly describes the transaction leveling modeling features based on the channel concept and presents some design examples.

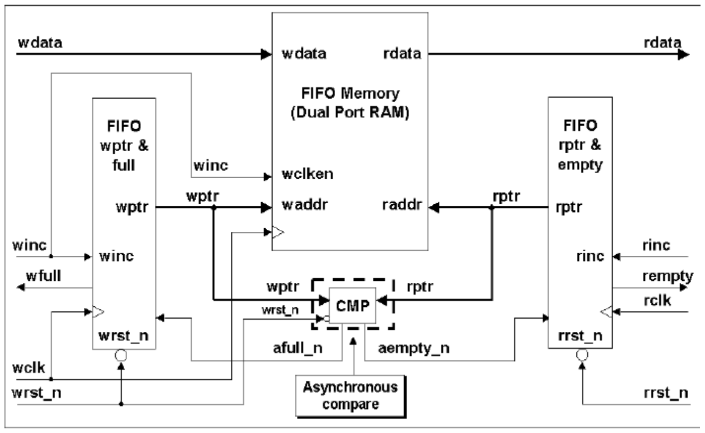

Figure 1. Mixed clock FIFO [3]

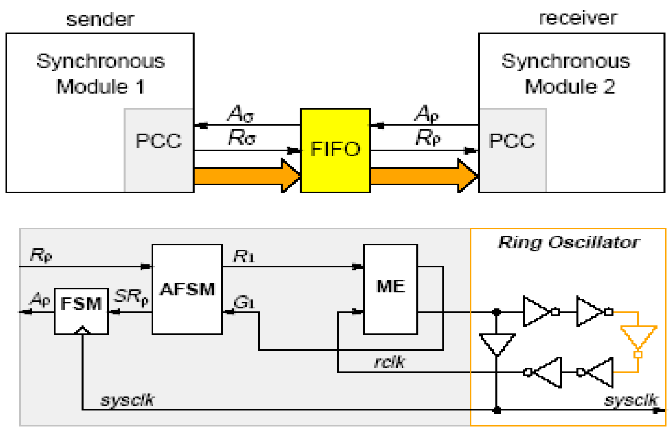

Figure 2. Pausible clock architecture [4]

We use these transaction level concepts, but our focus is on GALS communication interfaces, which has not been looked into before. Various on-chip communication architectures, such as networks on chip (NoCs) have been recently introduced and studied [9,10] in multiple voltage/frequency island environments to achieve low power and high performance.

3 GALS Systems

Globally asynchronous locally synchronous systems may offer a solution for SoC implementations seeking good performance and low power consumption. Locally clocked building blocks can be integrated on a single chip via asynchronous interconnect between them. This may lead to the common problem of metastability due to non-synchronization conditions of data and clock signal. This can be crudely resolved by using a double latching mechanism [11] to allow for metastability resolution. However, such a mechanism introduces an additional latency in the circuit.

An alternative strategy is to stretch the clock when there is a risk of metastability [4]. Such a scheme relies on generating the clock from a delay line because such a clock is simple to stop by gating the clock pulse. Self-calibrating delay lines may be used to provide an accurate timing reference. In this paper, we use SystemC to model such a clock-pausing scheme and evaluate certain metrics of using such communication architecture in a simple producer consumer scheme as well as real life applications that may be implemented as a GALS based system-on-a-chip.

A second solution is to use a mixed clock FIFO to account for the burstiness of data exchanged between the producer and the consumer, as well as interface the two different clock domains. Several examples of such mixed clock FIFOs [2,3] exist in literature. We also use a SystemC modeling methodology for evaluating a mixed clock FIFO at the system level for studying its impact on power and performance of real life and synthetic applications.

4 Communication Circuits and Architecture

In this section we describe the implementation of the communication architecture for point-to-point interconnect between locally synchronous modules. We describe two such communication schemes: (I) a memory based mixed clock FIFO and (II) a pausible clocking scheme.

4.1 Mixed Clock FIFO Architecture

In this case, we propose the use of a mixed clock FIFO for reading and writing data from and to locally clocked synchronous islands with different rates of producing or consuming data items. In the proposed scenario, we use a RAM based design for the FIFO, with read and write addresses being passed by the producer and the consumer modules. **Figure 1** shows a detailed description of the logic level circuit for the mixed clock FIFO implementation. The FIFO memory buffer is a dual ported RAM module that is accessed by both the read and write clock domains. The asynchronous pointer comparison module (*Asynchronous compare* in **Figure 1**) is used to generate signals that control the assertion of the asynchronous full and empty status bits. This module only contains combinational comparison logic. The *FIFO- wptr-&-full* block in **Figure 1** is synchronous to the write clock domain and contains the FIFO write pointer and full flag logic. Assertion of the *afull_n* signal is synchronous to the write clock domain since *afull_n* can only be asserted if the write pointer is incremented. However, the de-assertion of the *afull_n* signal happens when the read pointer is incremented which is asynchronous to the write clock. Thus we need a pair of latches to synchronize this signal to the write clock domain. The same holds good for the *FIFO-rptr-&-empty* logic block in **Figure 1** that is synchronous to the read clock domain. Assertion of the *aempty_n* signal is synchronous to the read clock, but de-assertion of the signal is asynchronous to the read clock and must be passed through a double latch synchronizer. Thus the main overhead in a mixed clock FIFO can be attributed to the synchronization of the full and empty signals to the write and read logic blocks respectively.

4.2 Pausible clocking based communication architecture

In this type of asynchronous communication between synchronous islands, we use a pausible clocking based scheme as proposed by Yun *et al.* [4]. Synchronous clock domains communicate with each other via completely asynchronous FIFO channels as opposed to mixed clock FIFOs as described in the earlier scheme. The interfaces between the synchronous modules and the FIFO are *pausible clocking control* (PCC) circuits. A block diagram of the communication architecture is shown in **Figure 2**. The signals $A\rho$, $R\rho$, $A\sigma$, $R\sigma$ are handshaking signals to request and acknowledge data transfers between the producer and consumer modules. The PCC is a scheme used to avoid synchronization failure by adjusting the local clock. A synchronization failure at the module interface occurs when the arrival times of an external signal transition and a sampling edge of the clock are indistinguishable by the sampling latch at the module boundary. In this case, pausing or stretching the local clock module when necessary circumvents the synchronization failure. The scheme uses a mutual exclusion element (ME) to force the temporal separation of the sampling edges of the clock and the external signal transitions. A ring oscillator is used instead of a crystal

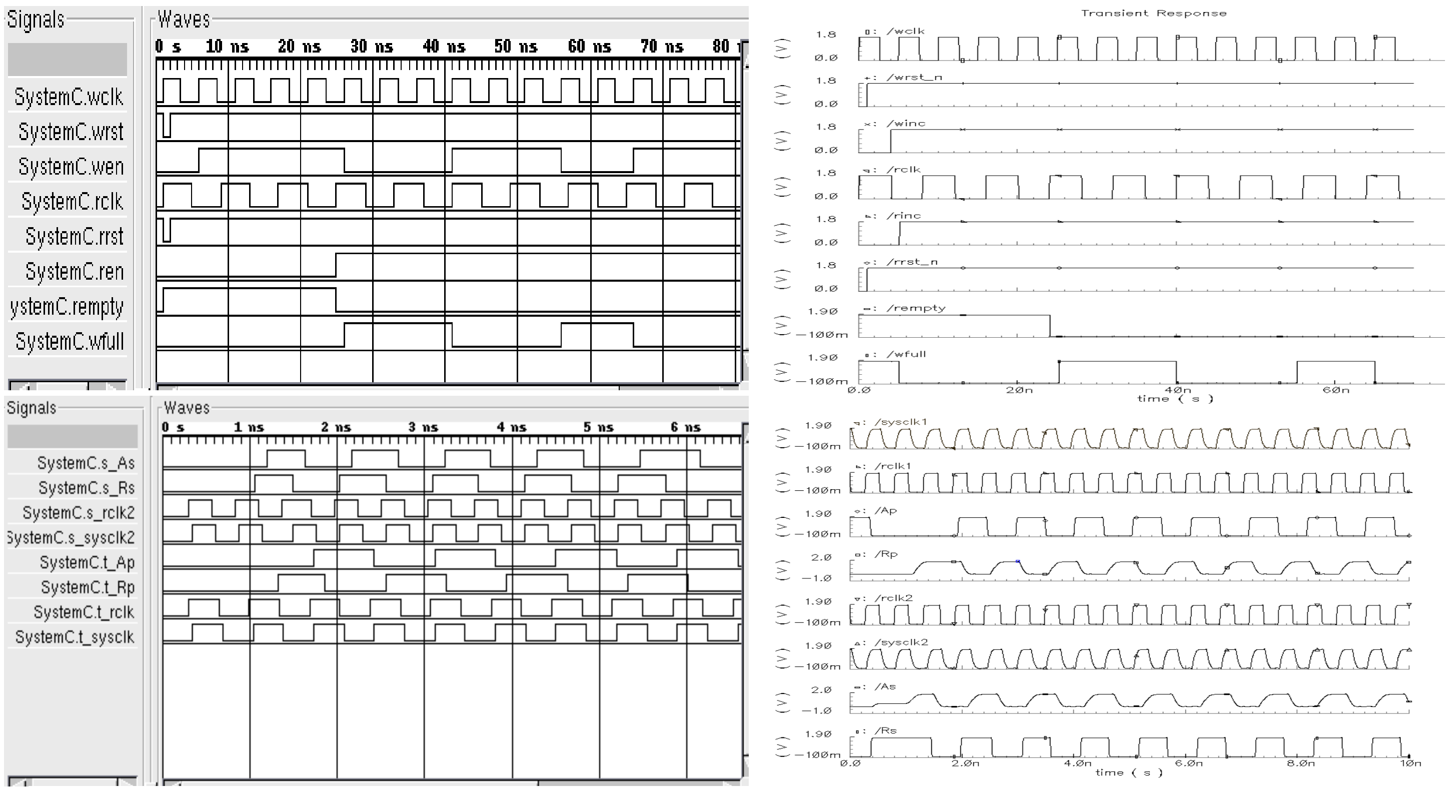

Figure 3. **SystemC/SPECTRE comparison of mixed clock FIFO (above) and pausible clock (below)**

oscillator in order to be able to adjust the duration of the off phases of the clock. As shown in **Figure 2**, a request event $R\rho$ is forwarded to the mutual exclusion element (ME) via the asynchronous finite state machine (AFSM). If *rclk* is low when R_1 rises, then the ME immediately raises G_1, which prompts the AFSM (algorithmic finite state machine) to generate an event on SR_p. This event is eventually synchronized to *sysclk,* i.e., guaranteed not to induce a synchronization failure when sampled by the FSM.

The important difference between the mixed clock FIFO architecture and the pausible clock based architecture is that the latter ensures that metastability does not occur, while the former has a very small (albeit, non-zero) probability of entering a metastable state depending on the time allowed during extra latching for synchronization purposes in the read and write logic blocks.

5 System Level Analysis of GALS based SOCs

Due to complexity incurred in distributing a single global clock across the entire chip area, and the varying power requirements for different functional blocks of system-on-chip applications, next generation systems will most certainly be implemented using multiple voltage/frequency islands [12]. Each such Voltage/Frequency Island (VFI) would have its own internal clock for its logic and powered by an off-chip or on-chip voltage source. This would enable designers to scale up or down the voltage and frequency of an on-chip module based on its performance requirements, thereby saving dynamic and static power. In this paper, we assume that an application is already logically partitioned into several on-chip synchronous modules communicating asynchronously with each other through GALS communication interfaces as described in the previous section. To this end, the proposed methodology relies on cycle-accurate models for the mixed-clock communication interfaces, validated against detailed, circuit level implementations. By using power and performance macro models validated against real implementations, we are thus able to provide highly reliable models for use at system level.

5.1 Modeling and Validation of GALS Interfaces

We have developed both SystemC models and complete circuit implementations of the mixed clock FIFO and pausible clock based communication interfaces. The circuit implementation is done using STMicroelectronics 130nm technology. SystemC enables modeling of these interfaces at various levels of abstraction. Thus, these models can be used at both RTL as well as transaction level depending on the stage of the design. Since SystemC is primarily used for modeling synchronous clock based systems, a completely asynchronous interface needs to be modeled and analyzed at the circuit level in order to extract the relevant delay parameters, which can be plugged into SystemC. To our knowledge, there has not been any similar effort in past literature to characterize such asynchronous interfaces in SystemC. Circuit-level accurate characterizations are used to validate and build the system-level models for the asynchronous interfaces. **Figure 3** shows the SystemC and SPECTRE waveforms for a mixed clock FIFO [3] and pausible clock circuit [4]. The mixed clock FIFO has the write clock running at twice the frequency of the read clock. This makes the *wfull* (write full) signal go high at time t=25ns and t=55ns respectively. For the pausible clock case, we run the producer and the consumer modules at 1.89 GHz and 1.47 GHz respectively. This causes a clock pause at the producer (signal *sysclk2*) at t=1.4ns. As described in Section 4.2, this is caused by arbitration between the clock signal and the acknowledgement signal received from the consumer.

5.2 Metrics for Characterizing Impact of GALS Interfaces

In order to characterize at system level GALS based SoC applications, we need to define metrics relevant at both power and performance, as well as input parameters affecting these metrics. The most important metric is the *throughput* of the application. System level analysis of the application can not only allow designers to analyze the effect of system parameters such as FIFO sizing, producer rate and consumer rate on the throughput of the application, but also enable them to do a cycle by cycle analysis of the throughput of the system.

For a *mixed clock FIFO interface*, we define a few metrics relevant to this type of GALS interface. The first one is related to the number of times an application experiences an additional clock cycle latency due to synchronization of the full/empty signals (*NUM_FE_LATENCY*) and what is the average duration of this latency in terms of clock cycles (*AVG_FE_LATENCY*). Such metrics enable designers to evaluate the performance penalty in using a mixed clock interface with respect to the single clock domain case.

In case of the pausible clock circuitry, two other metrics are relevant for its behavior and impact on overall performance or power. The first one is related to the number of times the clock signal of a synchronous module is paused (*NUM_RW_PAUSES*) and the second is associated to the total latency incurred by such read-write pauses over a specified simulation time (*TOT_RW_LATENCY*). Again, such metrics enable the system level designer to estimate the performance penalty for a pausible clock based circuit and explore other GALS communication architectures. For the pausible clock metrics, we use latency values obtained from SPECTRE simulations of the pausible clock asynchronous circuitry and plug them into the SystemC simulation environment. Finally, in both cases, we also consider the ratio of communication cycles to the computation cycles for a particular application, which helps the designer to analyze whether the application is communication or computationally intensive.

Due to increasing clock frequencies and smaller device sizes, power is an equally important metric in SoC based applications. Since GALS based architectures incur an extra overhead in terms of asynchronous communication circuitry, it is useful to characterize the power consumption of the computation cores and the communication interfaces for a particular application. To this end, we also evaluate the power requirements of both the communication elements using circuit level simulation, as well the computational cores using an architectural simulator to compare the energy consumption in different architectures.

6 Experimental Setup and Results

In this section we describe case studies on synthetic, as well as real life applications, which can be implemented as heterogeneous system-on-chip applications. We implement the GALS based communication interfaces as SystemC modules with input and output ports which can be plugged into any application implemented in SystemC either at the behavioral or RT level.

6.1 Synthetic Trace Case Study

The synthetic case studies are based on a simple producer consumer model, where the user can vary input parameters of the producer and consumer such as rate of production, rate of consumption, burst size of data, FIFO size and clock frequency of the producer and the consumer modules. The *producer rate* is defined as the probability that the producer will send a token to the consumer at a producer clock cycle (same definition holds for the *consumer rate*). The *burst size* is defined as the number of data tokens transferred by the producer during one data transfer from the producer to the consumer. For the experiments related to synthetic applications, we varied the clock frequency of the producer module from 200 MHz to 300 MHz. The *clock ratio* between the producer and the consumer modules is defined as the ratio of the consumer clock frequency to the producer clock frequency. We varied the *clock ratio* from 0.3 to 5 in steps of 0.15, which allows us to examine various phase relationships between the producer and the consumer clock. We also vary the *FIFO size* between 4 and 16 to observe the impact of FIFO size on throughput and other performance related metrics. We vary the producer and consumer rate between 0.4 and 1, while the burst size is varied from 1 to 8.

We implemented both the mixed clock FIFO and the pausible clock circuits in ST Microelectronics 130 nm technology. We performed SPECTRE simulations in order to verify our SystemC simulation results and also to abstract the delay parameters for the asynchronous logic in the pausible clock circuit. Experimental results for the synthetic models are shown in **Figures 4-10**.

Figures 4 and 5 show the impact of the clock ratio and FIFO size on the average throughput with the rates and burst size kept constant, in case of mixed clock FIFO and pausible clock based interfaces. We see an almost linear increase in throughput as the clock ratio increases, since this corresponds to an increase in the clock frequency of the consumer clock frequency, which translates to more data being read in the same period of time. Also we see that the curves saturate when the clock ratio reaches one. This happens because, after the consumer module operates at a faster clock frequency than the producer, there is no additional increase in throughput. There is a marginal increase in throughput due to increase in FIFO size when the producer and consumer operate at the same clock frequency. By analyzing the curves closely, we can see that the maximum impact of FIFO size on throughput occurs at a clock ratio of 0.9 to 1. For smaller values of clock ratio, the consumer module operates at a much slower rate than the producer and thus after an initial period of instability, the system reaches a steady state when the reads and writes occur according to the consumer clock frequency. For large values of clock ratio, the consumer is always much faster than the producer and thus the FIFO never gets full.

Figure 6 shows the impact of clock ratio and FIFO size on the number of stalls in the producer due to synchronization of the full signal. As expected, the number of stalls is maximized when the consumer module runs at a much slower frequency than the producer, since the FIFO fills up quickly. Also, it is seen that the number of such stalls becomes higher with increasing producer clock frequency. Again, when the producer and consumer clock frequencies are nearly equal, an increase in FIFO size reduces the number of stalls since we have less instances of the FIFO filling up.

Figure 7 shows the average duration of synchronizing the full signal of the producer with varying phase differences between the producer and consumer clocks (clock ratio). It can be seen that the number of synchronization cycles lies between 1 and 2, and is completely arbitrary depending on the phase difference between the producer and the consumer clocks. The phase difference between the clocks is dependent on the clock ratio between the producer and consumer clocks.

Figures 8 and 9 show the distribution of the number of clock pauses and the total latency due to clock pauses in the producer with varying phase differences (clock ratios) between the producer and consumer clocks for different values of producer rate, consumer rate, burst sizes and FIFO sizes. We can see that the maximum number of pauses occurs between clock ratios of 0.5 and 1. This kind of information may be very useful to decide which ratios of clock frequencies to avoid at an early stage of the GALS based design.

Finally, **Figure 10** shows the average power consumption for an eight bit four stage FIFO implementation, with varying clock ratios. As the clock ratio increases, the consumer operates at a faster rate and the throughput starts increasing. Due to increasing throughput and increasing consumer frequency, power consumption increases. It can be seen that the pausible clock consumes more power than the mixed clock FIFO because of its

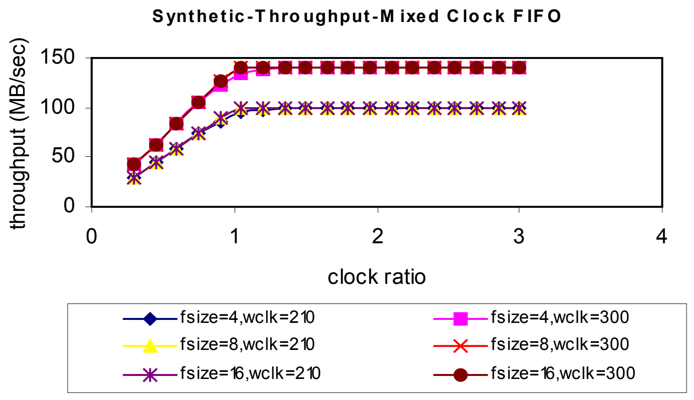

Figure 4. Average throughput for mixed clock FIFO

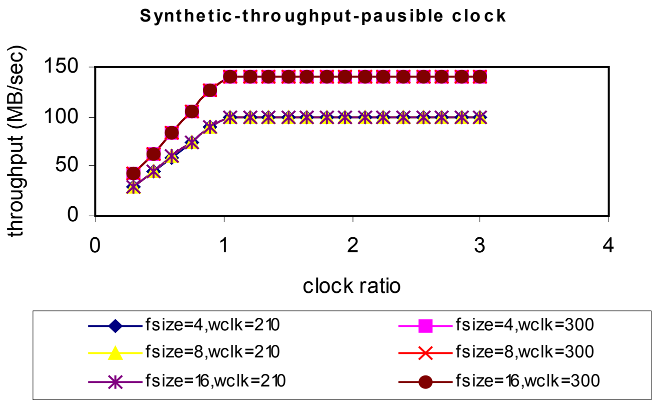

Figure 5. Average throughput for pausible clock circuit

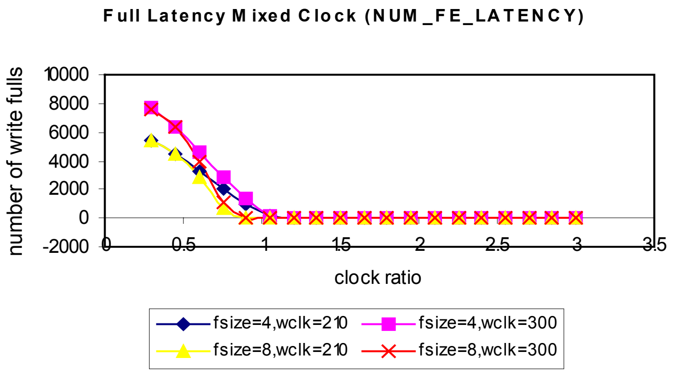

Figure 6. Number of synchronization stalls in producer

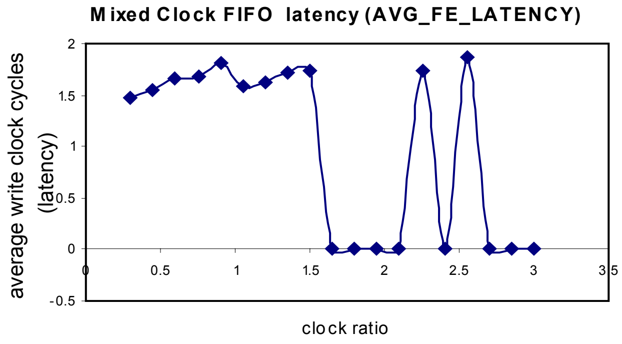

Figure 7. Average duration of synchronization stall in producer

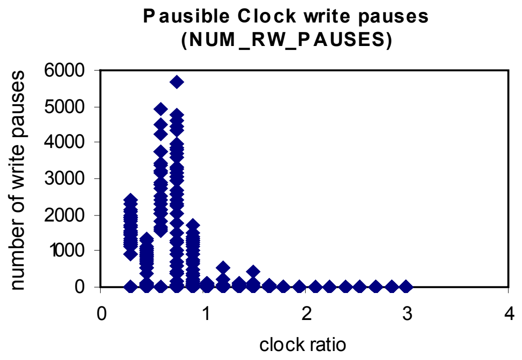

Figure 8. Number of clock pauses in producer

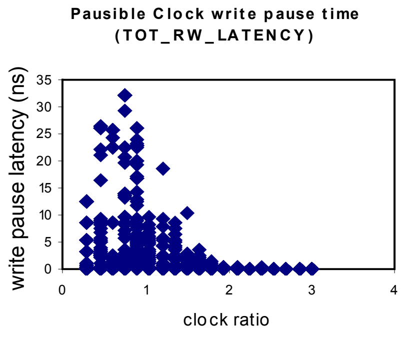

Figure 9. Total clock pause time in producer

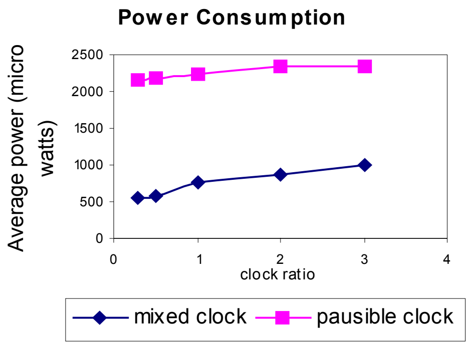

Figure 10. Power consumption in interfaces

complex asynchronous circuitry. However, it must be noted that the power number for the pausible clock circuitry includes the current drawn by the local ring oscillator clock.

6.2 Real Application Case Study

The real life application under consideration is *software defined radio* [13], which is partitioned into five components - source, low pass filter (*LPF*), demodulator (*DEMOD*), equalizer (*EQ*). Each component is assumed to be implemented as a stand-alone application executing on a single processor as shown in **Figure 11**. The source module generates samples at a fixed rate (1 KHz), that are sent to the LPF node through a GALS based communication interface, which may be either a mixed clock FIFO, or a pausible clock interface.

We performed static profiling of each module on an in-house multi-core simulator Myrmigki [14], to obtain the computation cycles and power consumption using instruction level models of the Hitachi SH core. **Figure 12** shows the impact of FIFO sizing on the communication cycles in each module. It is seen that there is some improvement in the equalizer node when the FIFO size is increased from 4 to 16, while the other modules do not show much improvement in terms communication latency. **Table 1** shows the ratio of communication to computation cycles in each module of the software radio system-on-chip for processing one sample. The number of communication cycles is negligible compared to the computation workload in this particular application. **Table 1** also

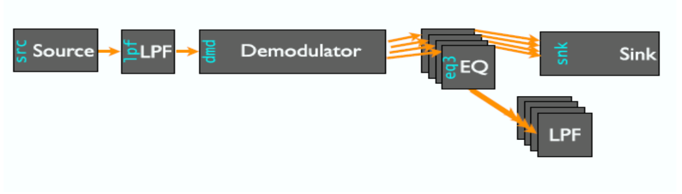

Figure 11. Software radio application

shows the breakdown of *NUM_FE_LATENCY* and *NUM_RW_PAUSES* by each module of the software radio application. The majority of the full empty stalls and read-write pauses occur in the *LPF* and *EQ* modules since these operate at a larger frequency than the other modules of the application.

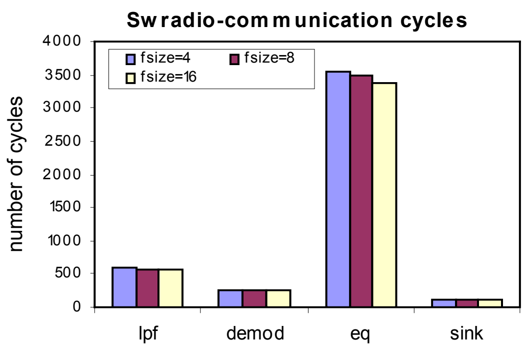

Figure 12. Impact of FIFO size on cycles

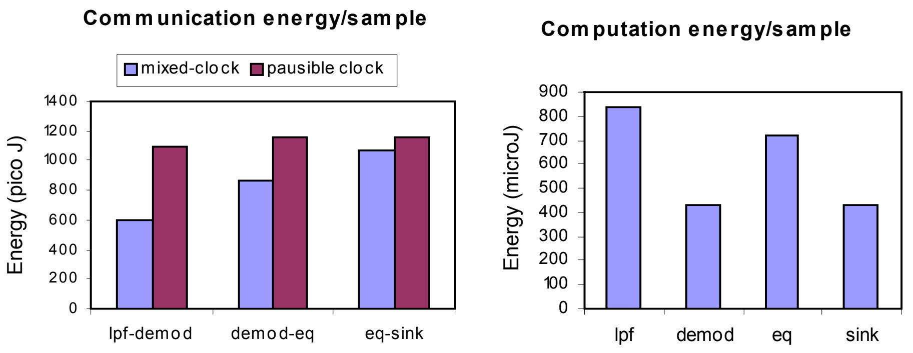

Figure 13. Computation and Communication energy in software radio

Figure 13 shows the energy consumption per sample in the communication interfaces between the components as well as the computational IPs in the system-on-chip. [1] The pausible clock circuit consumes more average power than the mixed clock FIFO interfaces as can be seen in the figure. The energy consumption in the interfaces between the computational elements depends on the clock ratio of the computational IPs and the time for sending and receiving one sample, which corresponds to 500ns at a source frequency of 1 KHz. The energy consumption in the computational elements which are Hitachi SH3 cores in this case is much larger compared to the communication energy consumption. The computation energy is measured using instruction level power estimation in Myrmigki [14]. The power numbers for the communication elements is obtained from SPECTRE simulations of the mixed clock FIFO and pausible clock circuits.

Module	computation/ comm. cycles	*NUM_RW_PAUSES*	*NUM_FE_LATENCY*
LPF	61194/583	102	338
DEMOD	33086/254	0	3
EQ	463190/3501	145	255
SINK	32736/127	0	1

Table 1. Metrics for Software Radio application

From a performance perspective, both the mixed clock FIFO and pausible clock circuits show similar throughput characteristics. However, the designer must keep in mind that while the pausible clock design eliminates metastability, it introduces undesirable circuit level characteristics like clock jitter due to pauses in the local ring oscillator. From a power perspective, the extra logic in the asynchronous circuit elements in the pausible clock circuit burn more power compared to the relatively simple decoder and full empty logic in the mixed clock FIFO.

7 Conclusion

This paper describes a system level methodology to evaluate the power and performance of GALS based interconnect systems. We evaluate two main interconnect architectures namely a mixed clock FIFO and a pausible clock based scheme. A system level model of these interconnect architectures allows the system level designer to design an application at the transaction or RTL level using system level models of such point to point asynchronous interconnect structures. Such system level characterization of GALS based interconnect reduces simulation time of an application by orders of magnitude compared to a Verilog or SPICE simulation. Further, for asynchronous circuits, circuit level delay parameters can be abstracted and plugged into re-usable SystemC models, thereby providing a vertical integration from the circuit to the system level. Future direction includes developing system level models of GALS based bus interconnects and building a library of such interconnect structures for easy use of the system level developer.

[1] Figure 13 (communication energy) shows energy in picoJoules, while Figure 10 shows power in microWatts. For energy, multiplying power with time in nanoseconds gives us picoJoules.

8 References

[1] D.M. Chapiro, "Globally Asynchronous Locally Synchronous Systems", PhD Thesis, Stanford University, 1984.

[2] T. Chelcea, S.M. Nowick, "Robust Interfaces for Mixed Timing Systems with Application to Latency Insensitive Protocols", Proceedings of IEEE Design Automation Conference, June 2001," Las Vegas, Nevada.

[3] C.E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO design," SNUG 2002, San Jose, CA.

[4] K. Yun, R..P. Donhue, "Pausible Clocking: A First Step Toward Heterogeneous Systems", Proceedings of International Conference on Computer Design, October 1996, Austin, TX.

[5] J. Muttersbach, T. Villiger, and W. Fichtner., "Practical Design of Globally Asynchronous Locally Synchronous Systems," Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems, April 2000.

[6] L. Cai, D. Gajski, "Transactional Level Modeling: An Overview," Proceedings of IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, October 2003, Newport Beach, CA.

[7] D.D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, "SpecC: Specification Language and Methodology," Kluwer Academic Publishers, March 2000.

[8] T. Groker et al., "System Design with SystemC," Kluwer Academic Publishers, 2002.

[9] W. Dally, B. Towles, "Route Packets, not wires: On chip interconnection networks," in Proceedings of ACM/IEEE Design Automation Conference, June 2001.

[10] T. Dumitras, S. Kerner, R. Marculescu, "Enabling On-chip Diversity through Architectural Communication Design," Proceedings of ACM/IEEE Asia-Pacific Design Automation Conference," Tokyo, Japan, January 2004.

[11] R. Ginosar, "Fourteen Ways to Fool your synchronizer," Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems, April 2003.

[12] K.Niyogi, D. Marculescu, "Speed and Voltage Selection for GALS Systems based on Voltage Frequency Islands,", Proceedings of the ACM/IEEE Asia-Pacific Design Automation Conference," China, January 2005.

[13] B. D. Van Veen and K. M. Buckley, " Beamforming: a versatile approach to spatial filtering," IEEE ASSP Magazine, vol.5, no.2, pp.4-24, April 1988.

[14] P. Stanley-Marbell, M. Hsiao, "Fast Flexible Cycle Accurate Energy Estimation," ACM/IEEE International Symposium of Low Power Electronics and Design, August 2001.